SESSION

PARALLEL COMPUTING AND ALGORITHMS + MULTI-CORE AND ENERGY-AWARE COMPUTING

Chair(s)

TBA

Modeling the Effects on Power and Performance from Memory Interference of Co-located Applications in Multicore Systems

Daniel Dauwe[1], **Ryan Friese**[1],
Sudeep Pasricha[1,2], **Anthony A. Maciejewski**[1], **Gregory A. Koenig**[3], **and Howard Jay Siegel**[1,2]
[1]Department of Electrical and Computer Engineering
[2]Department of Computer Science
Colorado State University
Fort Collins, CO, 80523
[3]Oak Ridge National Laboratory
Oak Ridge, TN 37830

Abstract— *In this study, we analyze interference trends when co-running multiple applications possessing varying degrees of memory intensity on multi-core processors. We conduct tests with PARSEC benchmark applications and explore energy consumption, execution times, and main memory accesses when interfering applications share last-level cache. We also explore how co-running applications are impacted when the processor frequency is modified using dynamic voltage and frequency scaling (DVFS). A portable and lightweight testing framework is presented and results are shown for experiments conducted on an Intel i7 quad-core system. It is shown that the degree of degradation due to co-location interference on execution time is highly dependent on the types and number of applications co-located on cores that share the last-level cache.*

Keywords: memory interference; application co-location; benchmarking; energy-aware computing; dynamic voltage and frequency scaling; multi-core processors

1. Introduction

There is an ever present desire to increase the performance and capabilities of current high performance computing systems. Frequently, increased performance comes at the cost of increased power dissipation, which has become a major challenge in large scale computing systems. According to the 2012 DatacenterDynamics census [DcD12], global datacenter power requirements in 2007 were 12 GW, but doubled to 24 GW in 2011. Then, in 2012 the power requirements grew by 63% to 38 GW [DcD12]. There is thus a critical need to profile and characterize power dissipation in computing nodes, as a precursor to developing techniques that can minimize power dissipation.

The use of multiple cores in today's processing units is commonplace as parallel processing remains a popular technique for speeding up the execution time of workloads. In an ideal system, doubling the number of cores in a server doubles the performance as long as the workload is perfectly parallel. However, cores in today's processors typically share resources in some manner (such as last-level cache, DRAM, network, and storage), causing contention for these resources and degrading performance, potentially resulting in larger amounts of energy being consumed.

Understanding the effects of resource contention caused by co-locating applications in multi-core processors is becoming increasingly important as the number of cores per processor continues to increase. Co-location occurs when more than one application is executing and sharing resources on a multi-core processor. Different applications have different resource requirements, and by studying the execution time and energy effects across several applications we can better understand the implications associated with co-locating applications on multi-core processors. A better understanding of these effects becomes critical to intelligently mitigating interference and improving performance and energy consumption.

This work examines the effects of memory interference on energy usage and application execution time (or application throughput). Memory interference is introduced by executing a number of applications across the cores of a given processor, resulting in the applications sharing L2 or L3 caches in the memory hierarchy. Specifically, the research provides a testing infrastructure that monitors and collects data on the following attributes as various applications are executed on a multi-core processor: energy usage, execution time, and main memory (DRAM) accesses. Applications that access main memory frequently can be considered to be "memory intensive." This research also analyzes how applications with varying levels of memory intensity affect the attributes of applications with which they are co-located.

We consider the attributes of four different workloads taken from the PARSEC benchmarking suite [Par14] for measurement and analysis. The four workloads were selected to represent a range of applications with varying memory intensity. The workloads are executed on a quad-core Intel i7 processor. We chose to conduct experiments on this processor because it is being currently used in compute nodes within several datacenters that are increasingly deploying systems with low-end commodity processors (instead of high end server processors) to keep hardware costs manageable at a large scale. Results are first gathered for the case where each application executes by itself on a single core in the processor. These results are then used as a baseline to contrast the degradation in performance when there are two and four applications co-located on the cores of the processor. It is shown that for more memory intensive applications, there is a greater decrease in performance as memory interference increases. Additionally, this research incorporates the dynamic voltage and frequency scaling (DVFS) property of the processor, and runs all the tests across a range of different voltages and frequencies. The results from this research can be used to provide highly accurate execution time and energy consumption information for use in the area of resource allocation in high performance computing systems, where application tasks are co-located on the cores of multi-core processors[FrB13], [FrK13], [OxP13].

In summary, this work makes the following contributions: (a) proposes a portable benchmark testing environment capable of measuring and analyzing the effects of cross-application interference on energy consumption, execution time, and memory accesses of various applications across CPU frequencies; (b) provides an analysis for a set of real-world workload execution scenarios using this testing environment to perform interference tests on a modern quad-core machine; and (c) gives insights into how large computer systems based on multi-core processors may be able to improve their energy use based on the memory interference caused by co-located applications.

The remainder of this paper is organized as follows. The next section will discuss related work. Section 3 will describe the portable testing environment. The workloads being tested and the experimental setup will be explained in Section 4. Section 5 will present and provide analyses for the results of the experiments. Conclusions and plans for future work will be discussed in Section 6.

2. Related Work

The authors from [KiC12] perform experiments to measure the effect of varying processor frequency on the energy consumption of workloads with varying levels of memory intensity on a single system. Multiple instances of each application are executed concurrently on the test system with each instance pinned to a separate core. The results of [KiC12] imply that the memory intensive tasks may be more energy efficient at slower frequencies while CPU intensive tasks may be more efficient at higher frequencies. The work in [KiC12] does not quantify memory intensity nor does it consider the effects of co-locating applications, be it instances of the same application or instances of different applications.

A study of how different architectures can affect the impact of DVFS on energy savings is analyzed in [SuH10]. Three generations of AMD processors from 2003 through 2009 are tested using a memory intensive workload across the various frequencies available to the processors. It was found that for the older processors, the most energy efficient frequencies existed in the middle of the dynamic frequency range, while for the newest processor, it was most energy efficient to run at the fastest frequency. The authors of [SuH10] only examine a single application and do not consider how performance may be affected by co-location.

The work in [TaM11] presents a study that investigates the impact of co-locating threads from multiple applications with diverse memory behavior on a quad-core system. The authors show that the execution time of co-located tasks can vary greatly depending on the other tasks that are executing. The impact co-location has on energy consumption is not considered in [TaM11], nor does the work try to isolate the effects different applications have on each other by pinning applications to specific cores.

The memory characterization of workloads from SPEC CPU2000 and SPEC CPU2006 are analyzed and presented in [Jal07]. It is shown that changes in the size of the cache of a processor can greatly affect the memory intensity of a given workload. This implies that as cache size increases, a workload can switch from being memory intensive to being CPU intensive. It is important to study the effects co-location could have on such tasks as it may make them memory intensive again once they have to contend for cache with other applications. The work in [Jal07] does not consider co-locating nor does it consider energy consumption.

A method is presented in [GoL11] to predict the performance degradation of workloads from interference due to shared processor cache (co-location). Each workload is encapsulated within its own virtual machine, and each virtual machine is pinned to its own core. The authors were able to reasonably predict the degradation due to co-location for various applications. The work in [GoL11] does not consider how the frequency of the CPU can affect the performance of the workloads nor does it examine impact of co-location on the energy consumption of individual tasks.

3. Testing Environment

3.1 Operating System

One of the goals of this research is to create a testing environment that is portable across a variety of machines and system architectures, as well as being "lightweight" to minimize noise that may occur from OS (operating system) jitter. OS jitter occurs when processes unrelated to the tested workload are executed (i.e., graphics procedures, mail daemons, security services, etc.). These processes can cause anomalies to appear in the data. The operating system used for this research is a lightweight command-line version of Ubuntu 12.04 Linux. The operating system runs Linux kernel version 3.8.0.29-generic and uses the Linux default "SCHED_OTHER" thread scheduling policy.

To ensure consistency across different experiments, any power saving features of the operating system have been disabled (e.g., screensaver and automatic processor throttling). The operating system has been configured in such a way that it can be used on a variety of different systems. Currently, it has been installed onto a bootable USB drive, but in the future, the OS could be run from a live CD or from a netboot.

3.2 Processor Performance Counters

Processors today have the ability to measure and report on numerous hardware events such as the number of cache misses or number of instructions executed through the use of performance counters. By recording different events, it is possible to gain insight into the characteristics of a given application on a given processor architecture. Typically, there are a large number of hardware events that can be measured but a small number of performance counters (e.g., in an Intel i7 there are only seven performance counters, but over 50 different measurable events [Int14]) meaning that only a limited number of events can be measured at any given time. To further complicate matters, due to differences between microarchitectures, the number, type, and availability of performance counters and measurable events can vary greatly from system to system. To allow the testing environment to be portable, we make use of additional tools to monitor and collect performance counter data across multiple systems.

The first tool used is the Performance Application Programming Interface (PAPI) which, is a portable API to hardware performance counters that simplifies interfacing between software and the native hardware performance counters of processors [Pap14]. Due to the issues involved with the variability in numbers and types of native hardware events available across microprocessor architectures, PAPI attempts to define a standard set of performance counters ("presets") that can be found in most microprocessors. These preset performance counters try to abstract away the architecture specific details to provide an easy way to manipulate and measure similar hardware events across numerous platforms. The PAPI library contains over 100 preset performance counters that are available for use [PaE14], however as stated earlier due to differences in architectures, not all performance counters are available on every system.

The HPCtoolkit [Htk14] interfaces with PAPI and provides a set of software tools that facilitates measuring and collecting the performance data of an application. The HPCtoolkit was designed to perform benchmark testing using performance counters and therefore there is very little overhead associated with utilizing this tool. Specifically for this research, the "hpcrun-flat" tool was used to execute and monitor the test applications.

3.3 Measuring Memory Intensity

To understand the effects of co-locating multiple applications, additional metrics of performance besides execution time and energy consumption should be measured. One hypothesis of this research is that applications that frequently need to access the last-level of cache and main memory have a greater degradation in performance when co-located with other memory intensive applications compared to applications that are mostly CPU intensive. To test this hypothesis, the measure we use here for relative memory intensity is last-level cache misses divided by instructions executed.

Relative memory intensity indicates how often an application must access main memory (DRAM) per instruction executed. The number of times an application accesses main memory can be measured by keeping track of the number of last-level cache misses. The total number of last-level cache misses is, however, not enough to classify the memory intensity of an application, as different applications can execute for different amounts of time, and have different instruction counts. Therefore, we normalize our last-level cache misses by the number of total instructions executed. The resulting metric called relative memory intensity allows for the comparison of memory intensity across applications regardless of the instruction counts of the applications.

The number of last-level cache misses and the total number of instructions executed are measured using performance counters. PAPI does not contain a standard "last-level cache miss" performance counter and thus the appropriate level (L2 or L3) cache miss event must be set depending on the specific microprocessor architecture.

3.4 Processor Performance States (P-states)

Performance states (P-states) utilize dynamic voltage and frequency (DVFS) capability in processors to control the power consumption and speed at which the processor is operating. P-states are denoted by integer numbers, with P-state P0 indicating the highest voltage and highest (fastest) frequency. Higher P-state numbers indicate lower voltages and lower (slower) CPU frequencies. The number of P-states available for any given CPU, as well as the voltage and frequency pairings that each P-state represents, are different for every processor architecture, and therefore trends and behaviors of applications on one system may be significantly different when compared to another system. Generally, across different processor microarchitectures, operating in higher P-states results in increased execution time.

Furthermore, P-states only control the portion of a processors total power consumption called the dynamic power. The remaining power in the processor is called the static power and is assumed to remain constant regardless of the P-state. The ratio of dynamic power to static power differs from architecture to architecture, and using P-states to decrease the CPU frequency and voltage will decrease the dynamic power use but also likely increases an application's execution time. Due to static power being constant across P-states and increased execution time, there is no guarantee that operating in a slower P-state will result in reduced energy consumption.

This research provides an infrastructure for analyzing how the performance in terms of execution time, energy consumption, and memory intensity changes across different P-states, in modern multi-core processor architectures.

Table 1: PARSEC Applications

Application	Description
canneal	cache-aware simulated annealing to optimize routing cost of a chip design
streamcluster	online clustering of an input stream
blackscholes	option pricing with Black-Scholes Partial Differential Equation (PDE)
bodytrack	body tracking of a person

3.5 "Watts Up? PRO" Power Meter

We used a *Watts Up? PRO* power meter [Wat14] to collect and calculate the energy used while an application executes on microprocessors. The *Watts Up?* meter connects to and measures the system at the "outlet" level, meaning that all the power used by the whole system is measured. The meter is able to measure the instantaneous power draw of the system at one second intervals. To calculate the energy consumed by an application, the power samples, which are the average power over the one-second intervals and so are simply summed over the execution time.

4. Experimental Setup

4.1 PARSEC Benchmark Suite

The workloads used in this research are taken from the PARSEC benchmarking suite [Par14], which consists of a diverse set of applications from many different computing areas. Four benchmarking applications were chosen from the PARSEC suite with the intention of providing a representative set of applications having varying degrees of memory intensity. The applications chosen for experimentation, along with a brief description, can be found in Table 1, organized from most memory intensive to least memory intensive.

4.2 Co-location Experiments

To measure the effect of co-location on the energy consumption, execution time, and relative memory intensity of an application, a set of baseline tests were conducted to establish measurements that these three metrics could be compared to, across changes in the number of processor cores. For a baseline test, the applications were executed by themselves on the multi-core processor. During execution, the application was pinned to a specific core using the Linux "taskset" command and was the only process executing other than OS-related processes. Additionally, each baseline test was executed at different processor frequencies (P-states) ranging from 3.40GHz, the CPUs default highest speed, down to 1.70GHz. Energy consumption, execution time, and relative memory intensity were measured for each application in each P-state.

From the relative memory intensity results of the baseline tests (Table 2), the applications were then classified into three groups based on the magnitude of their relative memory intensities. The applications are shown in order of decreasing memory intensity. As indicated in the table, canneal is the most memory intensive application with over two last-level cache misses per 100 instructions executed while blackscholes is the least memory intensive application with fewer than one last-level cache miss per 100,000 instructions executed.

Table 2: Memory Intensity Classification

Applications	Classification	Relative Memory Intensity
canneal	Intensity III	2.25×10^{-2}
streamcluster	Intensity III	1.64×10^{-2}
blackscholes	Intensity II	2.29×10^{-5}
bodytrack	Intensity I	7.44×10^{-6}

The canneal and streamcluster applications exhibit the most interactions with main memory, and are categorized as being “Intensity III” memory intensive tests, blackscholes is categorized a “Intensity II” memory intensive application, and the bodytrack application, accessing memory the least, is categorized as being a “Intensity I” memory intensive application.

To test the effect of co-location on energy use, execution time, and relative memory intensity, two additional sets of experiments were performed. The first set contains experiments where two applications were co-located each application is pinned to its own core. Each application could be paired with one of the other applications or a copy of itself. For brevity, a subset of these possible pairs is presented in Table 3. The selection of this subset of pairs is due to the fact the canneal application is the most memory intensive application that was tested. canneal is therefore tested against the other applications to determine how those applications are impacted when co-running with a highly memory intensive application.

The second set of experiments increases the level of co-location from two applications up to four applications, using the “taskset” command to pin each application to its own core within the quad-core processor. Again, a subset of the numerous possible combinations for co-locating the applications is presented in this work for brevity. These subset combinations are shown in Table 4.

Table 3: Two Core Interference Tests

Test Type	Applications co-located together
2 Intensity III	canneal, streamcluster
1 Intensity III and 1 Intensity II	canneal, blackscholes
1 Intensity III and 1 Intensity I	canneal, bodytrack

Table 4: Four Core Interference Tests

Test Type	Applications co-located together
4 Intensity III	2 canneal, 2 streamcluster
1 Intensity III, 1 Intensity II, and 2 Intensity I	1 canneal, 1 blackscholes, 2 bodytrack
2 Intensity III and 2 Intensity I	1 canneal, 1 streamcluster, 2 bodytrack

5. Results

For the results presented in this section, tests were performed on the 64-bit Intel i7 3770 3.40 GHz quad-core processor [Int12]. In this processor, each core has its own private L1 and L2 caches, but a shared (8MB) L3 cache, that is shared by all four cores. The operating frequency of the processor can vary from 3.40 GHz to 1.70 GHz over the range of P-states, and because DVFS is used to do this, it prevents the processor from over-clocking the CPU into Intel’s “turbo” mode during the baseline tests when only one core is being used. The average of nine runs of each test are reported.

The results of the co-location tests are shown in Figures 1 to 3. Each figure contains the results for relative memory intensity, execution time, and energy consumption of a single application across multiple processor frequencies for the baseline (red), two-way co-location (green), and four-way co-location (blue) tests. Subfigure (a) of each figure shows the relative memory intensity, Subfigure (b) shows the execution time, and Subfigure (c) shows the energy consumption.

Note that for the memory intensity and execution time subfigures (Subfigures 1(a) and (b) through Subfigures 4(a) and (b), the results shown are for the individual application being presented. For example, when looking at the execution times of canneal in Subfigure 1(b), the execution times for “2 Intensity III” and “4 Intensity III” bars are the actual execution times of only the canneal application, not the execution times of all the co-located applications. In contrast, due to the fact the *Watts up? PRO* meter takes measurements at the “outlet” level, the energy results in Subfigures 1(c) through 4(c) show the energy consumption for all co-located applications. Furthermore, because the applications have different execution times, the energy consumed that is reported for an application is the total energy consumed during that application’s execution time.

Figures 1 and 2 contain the results for the canneal and streamcluster applications. Recall that canneal and streamcluster are the two “Intensity III” memory intensive applications that were tested. When examining the effect of co-location on the relative memory intensity Subfigures 1(a) and 2(a) it is apparent that when multiple Intensity III applications are co-located the number of last-level cache misses increase, resulting in more memory accesses, and implying that the applications are creating interference by evicting lines from the last-level cache that were being used by other applications. The large variations between the baseline results and the dual and quad-core interference results seen between every application is a result of differences in how memory interference affects each application. It should be noted that the scale of each graph changes between figures. The execution time of both applications are shown in Subfigure 1(b) and Subfigure 2(b) respectively. Co-location increases the execution time of applications.

The results for the blackscholes and bodytrack applications are shown in Figures 3 and 4, respectively. These two application are significantly less memory intensive than canneal and streamcluster. The results for relative memory intensity (Subfigures 3(a) and 4(a)) indicate that co-location can greatly increase the memory intensity of “Intensity I” memory intensive tasks in a relative sense, but it is important to note that even in the worst case (highest memory intensity) these two applications still have fewer last-level cache misses than either of the best cases (lowest memory intensity) for the canneal and streamcluster applications. This is also the reason

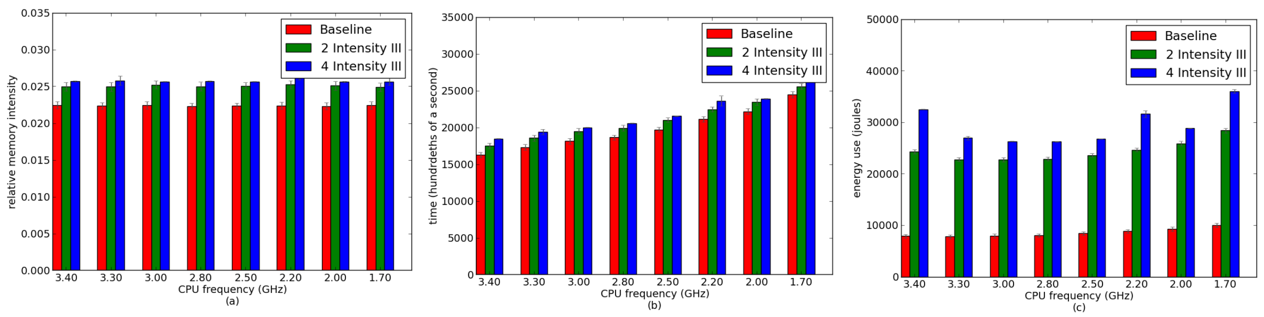


Fig. 1: The baseline (red), two-way co-location (green), and four-way co-location(blue) tests for canneal. The x-axis of each subfigure is the frequency of the processor. (a) Memory intensity results, the y-axis is the relative memory intensity measured as the number of last-level cache misses per instruction executed. (b) Exectuion time results, the y-axis is the execution time measured in hundredths of a second. (c) Energy consumption results, the y-axis is the energy consumed measured in Joules.

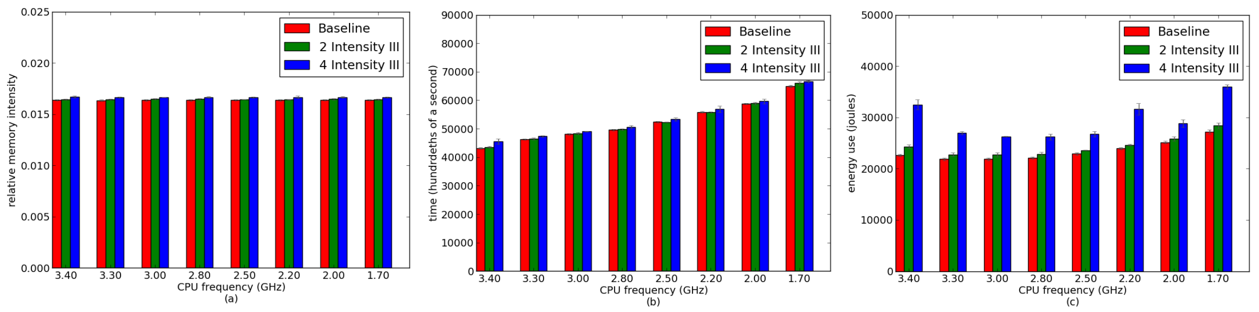


Fig. 2: The baseline (red), two-way co-location (green), and four-way co-location(blue) tests for streamcluster. The x-axis of each subfigure is the frequency of the processor. (a) Memory intensity results, the y-axis is the relative memory intensity measured as the number of last-level cache misses per instruction executed. (b) Exectuion time results, the y-axis is the execution time measured in hundredths of a second. (c) Energy consumption results, the y-axis is the energy consumed measured in Joules.

for the apparent fluctuations in the memory intensity values for blackscholes and bodytrack. As can be seen by examining the variations shown in the error bars, even the smallest variations present in the highly memory intensive canneal data is still greater than the largest variations in the blackscholes data (a difference of 7.05×10^{-5} for canneal as compared to 6.16×10^{-5} for blackscholes). Further analysis of the execution times for blackscholes and bodytrack (Subfigures 3(b) and 4(b)) indicates that even though co-location may have a significant relative increase in memory intensity for these applications, it minimally affects the execution times.

In general, across all applications, changing processor frequency (P-state) has minimal impact on the relative memory intensity (Subfigures 1(a), 2(a), 3(a), 4(a)), as processor speed should not affect the memory behavior of an application. Changing processor frequency does have a noticeable impact on execution time (Subfigures 1(b), 2(b), 3(b), 4(b)). As expected, when the frequency decreased, the execution times of all the applications increased. This increase in execution time is not uniform across all the applications. In general, the "Intensity I" memory intensive tasks are more sensitive to changes in frequency, meaning they experience a larger percent increase in execution time compared to the "Intensity III" memory intensive tasks. Most likely this is due to the fact that the "Intensity I" memory intensive tasks perform most of their computing on the CPU, operating mainly out of the cache, thus they do not need to access main memory very often.

An important distinction to make when analyzing the energy results (Subfigures 1(c), 2(c), 3(c), 4(c)) of these experiments is that the baseline results show the energy consumed during a single task's execution, while the co-location results show the energy consumed for *multiple* task's execution. For example in Figure 1(c) the energy use shown in the graph is a measure of the entire test's energy use up until the time that the canneal application finishes executing, whereas while the energy use of streamcluster in Figure 2(c) is taken from the same data measurements as that of Figure 1(c) the execution time of the the streamcluster application is longer, so its power use is greater than that of canneal.

It can be observed that even though the energy consumed to execute a task increases when co-located with other tasks, the total energy used to completely execute all the tasks actually decreases. For example, as detailed in table 5, if canneal and streamcluster were executed independently without any co-location they would require approximately 30,509 Joules total to run, but when co-located with one another they require approximately only 24,281 Joules total. This is because when run independently, both applications will separately incur the static energy present in the system in addition to the dynamic energy used during their execution. When co-located, both applications are able to share the

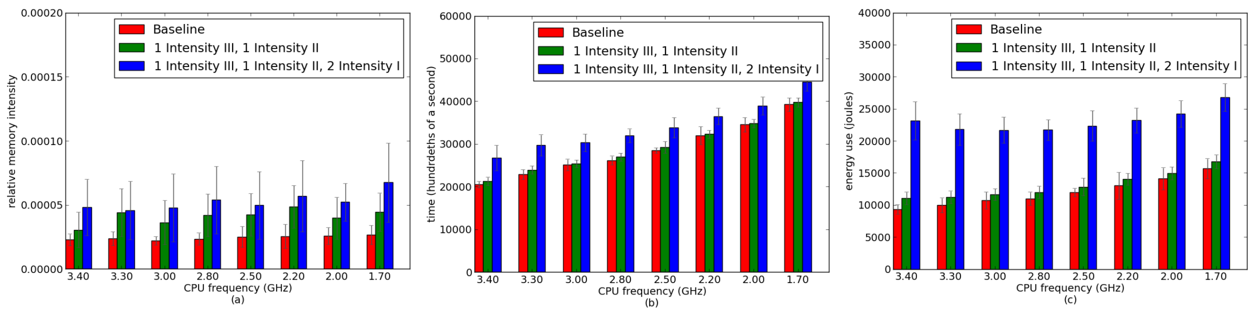


Fig. 3: The baseline (red), two-way co-location (green), and four-way co-location(blue) tests for blackscholes. The x-axis of each subfigure is the frequency of the processor. (a) Memory intensity results, the y-axis is the relative memory intensity measured as the number of last-level cache misses per instruction executed. (b) Execution time results, the y-axis is the execution time measured in hundredths of a second. (c) Energy consumption results, the y-axis is the energy consumed measured in Joules.

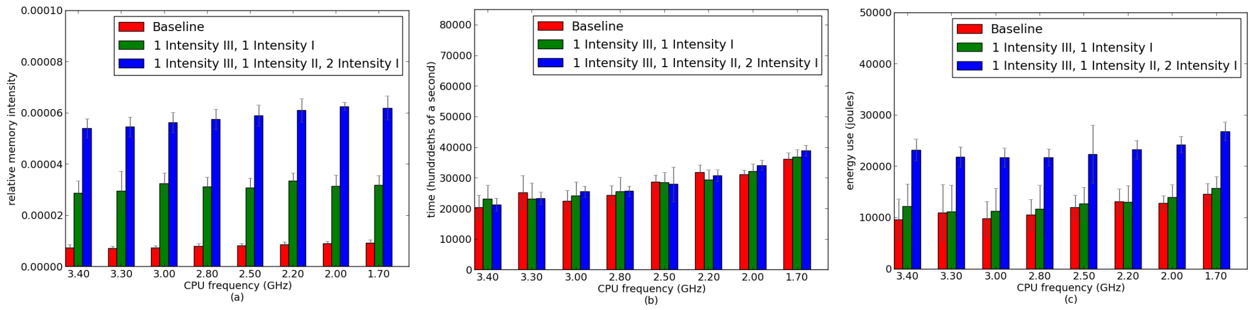


Fig. 4: The baseline (red), two-way co-location (green), and four-way co-location(blue) tests for bodytrack. The x-axis of each subfigure is the frequency of the processor. (a) Memory intensity results, the y-axis is the relative memory intensity measured as the number of last-level cache misses per instruction executed. (b) Exectuion time results, the y-axis is the execution time measured in hundredths of a second. (c) Energy consumption results, the y-axis is the energy consumed measured in Joules.

static power resulting in less static energy being consumed during execution, and less total energy used overall. Only a subset of these results are shown in the table, but they are consistent across all test runs. Furthermore, from the Subfigures 1(c), 2(c), 3(c), 4(c) it can be seen that the optimal p-state for minimizing dynamic energy is different between applications, and can change between an application's baseline, two-core, and four-core co-location tests.

This behavior along with the fact that minimal performance degradation occurs when "Intensity III" and "Intensity I" memory intensive tasks are co-located with one another provides many interesting and exciting possibilities for smart resource allocation managers in high performance computing (HPC) systems. For example, task schedulers could use this information to intelligently co-locate applications with differing memory intensities on multi-core processors in server nodes to minimize performance loss and decrease system energy consumption.

6. Conclusion and Future Work

With the desire for increased performance in computing systems, multi-core processors have become a popular and prevalent method of achieving higher performance. These multi-core processors are able to execute multiple applications at one time, however there exist complex interactions between the memory access behavior of applications that may cause degradations in performance and increased energy consumption for each individual application. This work examined the impact of memory interference on execution time, energy consumption, and relative memory intensity for co-located applications on multi-core processors. Specifically, a portable and lightweight testing framework was presented where four workloads taken from the PARSEC benchmark suite were run on an Intel i7 quad-core machine. The results verify that applications that have "Intensity III" memory intensity are more sensitive in terms of execution time and energy consumption when co-located with other "Intensity III" memory intensive applications, while "Intensity I" memory intensive applications are much less

Table 5: Test Energy Savings from Sharing Static Power (units in Joules, results are taken from tests run at 3.40GHz)

Application	Energy Use
canneal	7873
streamcluster	22,636
blackscholes	9347
bodytrack	9583
canneal + blackscholes	11,055 (Co-located)
canneal + streamcluster	24,281 (Co-located)
canneal + blackscholes + 2 bodytrack	28,729 (Co-located)
2 canneal + 2 streamcluster	32,504 (Co-located)

susceptible to degradations in their performance when co-located with other applications. For the specific system tested, it was shown that the optimal processor frequency for minimizing energy consumption could change based on the application and the co-location workload due to variations in use of dynamic power, the increased execution time, and static power present within the system. It was also found that co-running applications can also lead to sharing of static power use among the simultaneously running applications, which ends up decreasing the energy consumed for each application. Changing the frequency of the processor had negligible effect on the memory intensity of the applications.

Some possible directions for future work include increasing the number of PARSEC test applications as well as including applications from additional benchmark suites and performing co-location tests on a variety of systems. Information gathered from these tests could then be used by high performance scheduling systems to co-locate applications in a manner that minimizes performance degradation and energy consumption, as well as being extended to resource management in heterogeneous multicore-based distributed systems; e.g., [ApY11], [YoA13], [YoP13].

7. Acknowledgements

This research used resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, supported by the Extreme Scale Systems Center at ORNL, which is supported by the Department of Defense. This work was supported by the National Science Foundation (NSF) under grant numbers CNS-0905339, CCF-1252500, CCF-1302693, and an NSF Graduate Research Fellowship. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF. The authors thank M. Hilton, B. Khemka, S. Powers, R. Rambharos, and M. Wright for their valuable comments on this work.

References

[ApY11] J. Apodaca, D. Young, L. Briceño, J. Smith, S. Pasricha, A. A. Maciejewski, H. J. Siegel, S. Bahirat, B. Khemka, A. Ramirez, and Y. Zou, "Stochastically robust static resource allocation for energy minimization with a makespan constraint in a heterogeneous computing environment," *9th IEEE/ACS International Conference on Computer Systems and Applications (AICCSA '11)*, Dec. 2011, pp. 22–31.

[DcD12] (2012) 2012 DatacenterDynamics Industry Census, http://www.datacenterdynamics.com/blogs/industry-census-2012-emerging-data-center-markets.

[FrB13] R. Friese, T. Brinks, C. Oliver, A. A. Maciejewski, H. J. Siegel, and S. Pasricha, "A machine-by-machine analysis of a bi-objective resource allocation problems," *The 2013 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2013)*, July 2013, pp. 3–9.

[FrK13] R. Friese, B. Khemka, A. A. Maciejewski, H. J. Siegel, G. A. Koenig, S. Powers, M. Hilton, J. Rambharos, G. Okonski, and S. W. Poole, "An analysis framework for investigating the trade-offs between system performance and energy consumption in a heterogeneous computing environments," *22nd Heterogeneity in Computing Workshop (HCW 2013), in the proceedings of the IPDPS 2013 Workshops & PhD Forum (IPDPSW)*, May 2013.

[GoL11] S. Govindan, J. Liu, A. Kansal, and A. Sivasubramaniam, "Cuanta: Quantifying effects of shared on-chip resource interference for consolidated virtual machines," *2nd Symposium on Cloud Computing (SOCC'11)*, 2011, pp. 1–14.

[Htk14] (accessed Mar. 2014) HPCToolkit, http://hpctoolkit.org/.

[Int12] Intel. (2012) Intel core i7-3770 processor, http://ark.intel.com/products/65719/.

[Int14] "Intel 64 and ia-32 architectures software developer's manual volume 3b: System programming guide, part 2," Tech. Rep., Feb 2014, http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.pdf.

[Jal07] A. Jaleel, "Memory characterization of workloads using intrumentation-driven simulation," Tech. Rep., 2007, http://www.jaleels.org/ajaleel/workload/SPECanalysis.pdf.

[KiC12] S. Kim, C. Choi, H. Eom, and H. Y. Yeom, "Energy-centric DVFS controlling method for multi-core platforms," *5th International Workshop on Multi-Core Computing Systems (MuCoCoS'12), as part of Super Computing 2012*, Nov 2012.

[OxP13] M. Oxley, S. Pasricha, H. J. Siegel, and A. A. Maciejewski, "Energy and deadline constrained robust stochastic static resource alloation," *1st Workshop on Power and Energy Aspects of Computation (PEAC 2013), in the proceedings of the 10th International Conference on Parallel Processing and Applied Mathematics (PPAM 2013)*, Sep 2013, p. 10.

[PaE14] (accessed Mar. 2014) PAPI events by architectures, http://icl.cs.utk.edu/projects/papi/presets.html.

[Pap14] (accessed Mar. 2014) Performance application programming interface, http://icl.cs.utk.edu/papi/.

[Par14] (accessed Mar. 2014) PARSEC benchmark suite, http://parsec.cs.princeton.edu/.

[SuH10] E. L. Sueur and G. Heiser, "Dynamic voltage and frequency scaling: The laws of diminishing returns," *The 2010 International Conference on Power Aware Computing and Systems (HotPower '10)*, Oct 2010, p. 5.

[TaM11] L. Tang, J. Mars, N. Vachharajani, R. Hundt, and M. L. Soffa, "The impact of memory subsystem resource sharing on datacenter applications," *38th Annual International Symposium on Computer Architecture (ISCA'11)*, June 2011, pp. 283–294.

[Wat14] (accessed Mar. 2014) Watts Up? plug load meters, https://www.wattsupmeters.com/secure/products.php?pn=0.

[YoA13] B. D. Young, J. Apodaca, L. D. Briceño, J. Smith, S. Pasricha, A. A. Maciejewski, H. J. Siegel, B. Khemka, S. Bahirat, A. Ramirez, and Y. Zou, "Deadline and energy constrained dynamic resource allocation in a heterogeneous computing environments," *The Journal of Supercomputing*, Vol. 63, No. 2, Feb. 2013, pp. 326–347.

[YoP13] D. Young, S. Pasricha, A. A. Maciejewski, H. J. Siegel, and J. T. Smith, "Heterogeneous energy and makespan-constrained dag scheduling," *International Workshop on Energy Efficient High Performance Parallel and Distributed Computing (EEHPDC-2013), sponsor: ACM*, Jun. 2013, pp. 3–11.

Processing NOAA Observation Data over Hybrid Computer Systems for Comparative Climate Change Analysis

Xuan Shi[1,], **Dali Wang** [2]
[1] Department of Geosciences, University of Arkansas, Fayetteville, AR 72701, USA
[2] Environmental Science Division, Oak Ridge National Lab Oak Ridge TN 37831, USA

Abstract - *With the rapid development of weather monitoring system, numerous observational data are available. For example, NOAA provides Global Surface Summary of Day (GSOD) data that incorporates daily weather measurements from over 9000 weather stations around the world. In this paper, a comprehensive workflow and methodology is presented to elaborate how to transform GSOD data into a new and useful format so as to generate interpolated product of daily, monthly and annual mean surface temperature datasets by using advanced computation platforms. The quality of this gridded, high resolution (at ¼ degree) daily product is further examined by comparing to an existing global climate dataset. A preliminary comparison on global surface temperature shows a consistent agreement between these two datasets, with the major differences located in a few regions. The interpolated GSOD data products are supplementary to existing datasets by providing new gridded, high resolution observation-based daily temperature information over three decades (1982-2011), which are very useful for decadal climate change researches.*

Keywords: GSOD/NOAA Observational Data, Interpolation, Parallel Computing, GPU

1 Introduction

Historical weather datasets have been extensively used as a source of information to study global climate change and to validate and verify earth system models [1][2]. NOAA provides Global Surface Summary of the Day (GSOD) data through an FTP server ftp://ftp.ncdc.noaa.gov/pub/data/gsod/. GSOD data incorporates daily weather measurements (temperature, dew point, wind speed, humidity, barometric pressure, and so forth) from over 9000 weather stations around the world. GSOD data are available from 1929 to the present, with the data from 1973 to the present being the most complete. GSOD data, in its original format, however, could hardly be utilized directly and efficiently by researchers. Since each weather station is identified by a specific code, it is difficult to know where those stations are located in the archived data files when the location [latitude and longitude] information of weather observation station, as well as other descriptive information about the data, is documented in the metadata file, which is stored separately from the source data in ASCII format. This means, even the users can download the data from the FTP Server, it is difficult for them to understand where the weather stations are located if they cannot link the location information within the metadata with the ASCII file, let alone to find station(s) for a specific area or place. It is impossible to derive the global climate change information from such individual, station-based, unstructured data for any given temporal scope.

This paper describes the workflow and method to transform GSOD raw data into an applicable format and how to produce interpolated temperature data from daily meteorological observations at any arbitrary resolution. Considering the general interest of journal readership, global daily temperature results on a 0.25 degree x 0.25 degree surface grid were generated for duration from 1982 to 2011. Furthermore, time-series monthly average temperature grids were generated to compare with widely used high resolution gridded datasets (http://www.cru.uea.ac.uk/cru/data/hrg/), which contain the time series of monthly average temperature developed by the Climatic Research Unit at the University of East Anglia (CRU TS). The preliminary comparison result indicates the overall consistence between these two datasets, while major differences are located around the Tibet plateau. The future work will focus on the improvements of station selection for interpretation, the topography-dependent heterogeneity of surface temperature measurement, as well as comparison with other existing global datasets, such as NASA Goddard Institute for Space Studies (GISS) and Moderate Resolution Imaging Spectroradiometer) MODIS datasets.

2 Methodology and Workflow

GSOD data contains a variety of observed weather information including the mean, maximum and minimum temperature, mean dew point, mean sea level and station pressure, precipitation amount and snow depth, as well as other elements. However, such an invaluable data has been archived by individual station in unstructured ASCII format. In order to efficiently utilize GSOD data for climate change research, station-based data has to be transformed into date-based data in which the location of the station is embedded and merged into the time series datasets. As a result, daily global mean surface temperature can be approximated by applying interpolation algorithm. Furthermore, monthly and annual mean surface temperature as well as anomaly can be developed.

Interpolation is a method to estimate the value of unsampled location based on the values of existing observations. Interpolation can be implemented by different approaches in different domain science applications. Among these approaches, Kriging is a geostatistical interpolation method that is effective for predicting the spatial distributions of geographic features, although Kriging has complex implementation and a large computation load [3][4][5][6]. We applied Kriging interpolation in this pilot study.

Technically, the data processing workflow contains four steps (as shown in Figure 1), including 1) data transformation, which converts station-based ASCII file into date-based data by integrating the location information into the new daily dataset; 2) data interpolation, which generates interpolated daily mean surface temperature; 3) data aggregation, which generate the monthly and annual mean surface temperature or 30 year anomaly for example; and 4) data subtraction, which derives the temperature change information.

Scientifically, the new data product will help user and researchers to 1) identify and understand the spatial and temporal differentiation of climate change in the past decades at the global and local scale; 2) explore and understand the climate change tendency by analyzing and visualizing historical data; 3) compare, validate or examine potential climate models and results; 4) integrate the output into other research projects as the source of data.

3 Computational Platform

Kriging interpolation is data and compute intensive especially when great circle distance is applied to identify a given number of nearest neighboring observation stations. When high resolution output grids are generated, it may take hundreds of days to process the entire data for three decades. Parallel computing over the Graphic Processing Units (GPU) can significantly accelerate the time-consuming calculation process to improve the performance in verities of scientific computation. When multiple GPUs can be utilized to accelerate Kriging calculation, the processing time can be reduced from dozens of minutes on the serial program to dozens of seconds on a single GPU over a desktop computer and to a few seconds on Keeneland [7], which is a hybrid computer system that has 240 CPUs and 360 GPUs.

We started the development process to implement Kriging computation over a desktop computer in order to establish a standard for quality control and performance comparison to the parallel solution and products. The desktop computer has an Intel Pentium 4 CPU with 3.00 GHz main frequency, while the RAM size is 4 GB. The desktop machine has a graphic processing unit (GPU) that is a NVIDIA GeForce GTS 450, which has 192 cores and has 1 GB global memory. According to the technical specification, this GPU has 24 streaming multiprocessors (SM). Each SM has 8 CUDA cores called as streaming processor (SP). In this GTS 450 with a compute capability of 2.1, up to 1024 threads can be assigned to each SM. Thus a maximum of 1024 x 24 = 24,576 threads can run concurrently in parallel on the physical GPU, although the maximum sizes of each dimension of a block is 512 x 512 x 64 and the maximum

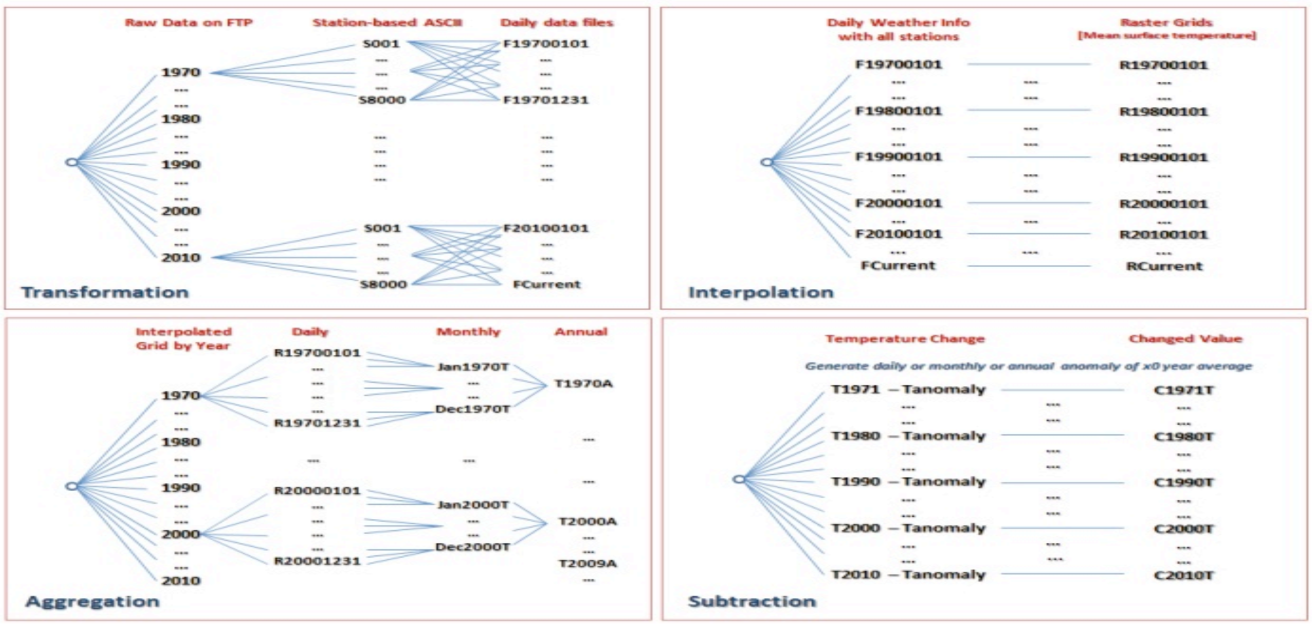

Raw Data on FTP
Station-based ASCII
Daily data files
Transformation
Daily Weather Info with all stations
Raster Grids [Mean surface temperature]
Interpolation
Interpolated Grid by Year
Daily
Monthly
Annual
Aggregation
Temperature Change
Changed Value
Generate daily or monthly or annual anomaly of x0 year average
Subtraction

Figure 1: Workflow of GSOD data transformation and computation

sizes of each dimension of a grid is 65535 x 65535 x 1. If the number of threads is more than the maximum number [24,576], the remaining threads have to wait.

After the Kriging interpolation algorithm is implemented and validated over both the serial program and the parallel solution over desktop GPU, Kriging is implemented over Keeneland, a hybrid computer system jointly developed by Georgia Institute of Technology, the University of Tennessee at Knoxville and the Oak Ridge National Laboratory sponsored by NSF, to accelerate the computation over more than 10,000 daily temperature estimation.

Keeneland is composed of an HP SL-390 (Ariston) cluster with Intel Westmere hex-core CPUs, NVIDIA 6GB Fermi GPUs, and a Qlogic QDR InfiniBand interconnect. The system has 120 nodes, each with two CPUs and three GPUs, while all CPUs and GPUs are bridged together through one I/O hub from which the CPUs can read/write data. Generally the CPUs serve as a high-level controller coordinated through Message Passing Interface (MPI), while GPUs implement the intensive computation job at a relatively low-level. By utilizing multiple GPUs on Keeneland, the computational time on interpolation was reduced to 3-4 seconds when one Keeneland node with three GPUs was utilized even without applying any spatial index

4 Implementation Details

Within the workflow of data transformation and computation, Kriging interpolation could be the most time-consuming procedure when a resolution of 0.25 x 0.25 degree grid is designed as the output product that has 1440 x 720 = 1,036,800 cells. If the GSOD data has 2,000+ to 5,000+ records, Kriging through serial computer program needs more than 11 or 30 minutes. While the weather observational stations in the world have been increasing in recent decades, GSOD data may have more than 10,000 daily records thus Kriging may need 40-50 minutes to process one daily data. If interpolating one day data needs 30 minutes by average through serial program, it would take about 328,500 minutes or 228 days to process 30 years of daily data from 1982 to 2011. For this reason, we pursued high performance computing solution by utilizing hybrid computer system to implement kriging through combined message Passing Interface (MPI) and Graphics processing unit (GPU) programs.

4.1 Implementation on desktop GPU

GPU was traditionally utilized in computer graphics applications. Considering the massive parallelism enabled by the GPU, it can be used for general-purpose computing and thus called GPGPU. By executing tens of thousands of threads concurrently, GPGPU enables high performance computing even on desktop or laptop computers. Compute Unified Device Architecture (CUDA) is NVIDIA's general-purpose parallel computing architecture. Here, the Central Processing Unit (CPU) is referred to as a host, while an individual GPU is referred to as a device. Normally the GPU executes the data computation process while I/O is done on the CPU, which also manipulates the workflow. The kernel is the function that runs on the device and is executed by an array of threads, while all threads can run the same code concurrently. Each thread has a unique thread identifier and can be accessed via the threadIdx variable. Thread identifiers (threadIDs) can be defined in one, two or three dimensions. Furthermore, threads can be grouped into thread blocks and grids. Threads in same thread block can cooperate with each other via shared memory, atomic operations or barrier synchronization. Threads in different blocks cannot cooperate. A user-defined number of threads can be organized in a block with a maximum number of 512 threads. Similarly a group of thread blocks can be organized into a grid in which each thread may be executed independently and thus may execute in parallel.

The first test is on the desktop computer through Visual Studio .Net 2010. Interpolation calculation could be a perfect match for parallel computing using GPUs. In essence, interpolation can be treated as a matrix calculation which is generic in GPGPU applications. We specify a number of columns and rows to define the dimension of the output grid. For each cell, we need to first find a given number of nearest neighboring points that have observational records. Then we implement the Kriging algorithm over each cell for interpolation calculation to derive the approximated value based on the observational values of its nearest neighbors. The calculation on each cell has no dependence on the other cells thus interpolation can be processed as an embarrassingly parallelism.

A general scheme for Kriging by CUDA C program can be *summarized as:*

1. Specify the types and sizes of input and output data;
2. Allocate memory on GPU for input data, output data, and intermediate data;
3. Allocate the computing resource on GPU, i.e. specify number of threads per block and total number of blocks;
4. Copy both input and output data from CPU to GPU;
5. Execute the algorithm for kriging computation;
6. Copy both input and output data from GPU to CPU;
7. Write the output data in ASCII grid format;
8. Free the allocated GPU memory.

To achieve high performance, we specify the number of blocks to be used and the number of threads in each block. In this case, for example, if 20,000 concurrent threads can be used to run Kriging interpolation, the program will be executed 50 times if the output grid has 1 million cells.

Table 1. Performance comparison based on different scale of data between serial program, a single GPU over desktop machine, as well as combined MPI and GPU program over Keeneland. Time is counted in seconds.

Data Size	**Time/speedup on desktop**		**Time/Speedup on Keeneland**			
	1 CPU	**1 GPU**	**1 GPU**	**3 GPUs**	**6 GPUs**	**9 GPUs**
2191	669	56 / 12	7 / 96	4 / 167	6 / 112	6 / 112
4596	1570	66 / 24	8 / 196	5 / 314	6 / 262	7 / 224
6941	1960	65 / 30	7 / 280	4 / 490	7 / 280	6 / 327
9817	2771	52 / 53	6 / 462	4 / 693	7 / 396	6 / 462

4.2 Implementation on Keeneland

Implementing the spherical interpolation computation on Keeneland is a combination of MPI and CUDA programs. The CUDA program is responsible for the computation of a block of the interpolated raster grid divided by horizontal rows. Each MPI process has a unique process rank number which is used to specify how many and which rows each CUDA program will process on the GPU node. The MPI processes read the input data, assign the jobs to the GPU nodes to implement the spherical interpolation program, and write a segment of the output data into a file in parallel. When all MPI processes are completed, one MPI process merges all segments of the output data into a single file.

5 Performance Evaluation

A varied scale of datasets is used in the performance testing. Given the size of the output grid as 1440 x 720 = 1,036,800 cells, Tables 1 displays the performance of the Kriging interpolation over different datasets using a single CPU, and the performance and speedups of the parallelized solutions on a single GPU on a desktop computer, and on 1, 3, 6, and 9 GPUs on Keeneland. In this case, 10 nearest neighboring points that have observational values are used in the Kriging calculation. The advantage of using the GPU is noticeable along with the increasing data scale as the speedup increases even when a single GPU is used on a desktop computer.

In Keeneland, the maximum speedup is achieved when one node with 3 GPUs was utilized at all scales of input data. This result implies that at current scale of input data size, utilizing more GPUs may result in more overhead for data manipulation between the host and device. If larger scale of data could be applied, it may have a different performance pattern or result over different number of GPUs utilized in the calculation.

6 Visualization and Analytics of Interpolated GSOD Data

Now that GSOD data has been transformed into the new format, such observational data can be visualized and analyzed through geographic information system (GIS) software for example. Figure 2 displays the distribution of the weather observation stations on the globe on July 1, 2009 and the global mean surface temperature on this specific day modeled by Kriging calculation. Time series of temperature evolution can be visualized as a movie or animation.

For any given location on the earth, we can to search query on the GSOD data through the identification function call. By clicking on the map interface, we can identify the location (latitude and longitude) of the clicked point and retrieve the daily mean surface temperature of this given location for any year. In this way, we can examine the quality of the Kriging result by comparing the Kriging result with the original GSOD data for any known station so as to validate the methodology for further improvement of this work on the one hand. On the other hand, we can offer the capability or service to allow users to search query over the local or

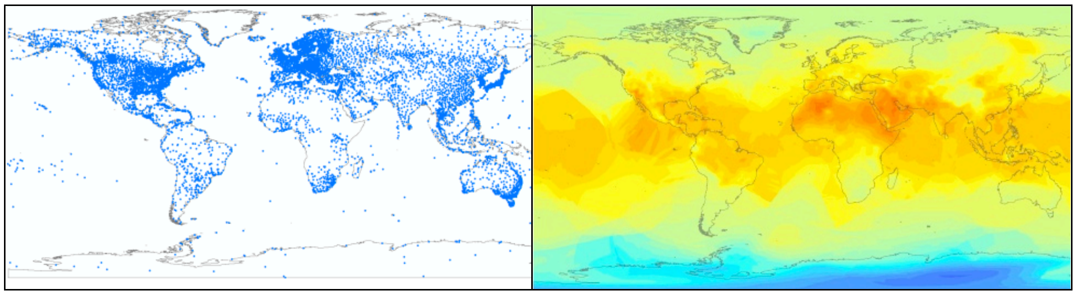


Figure 2: Distribution of weather observation stations and the global mean surface temperature on 07/01/2009

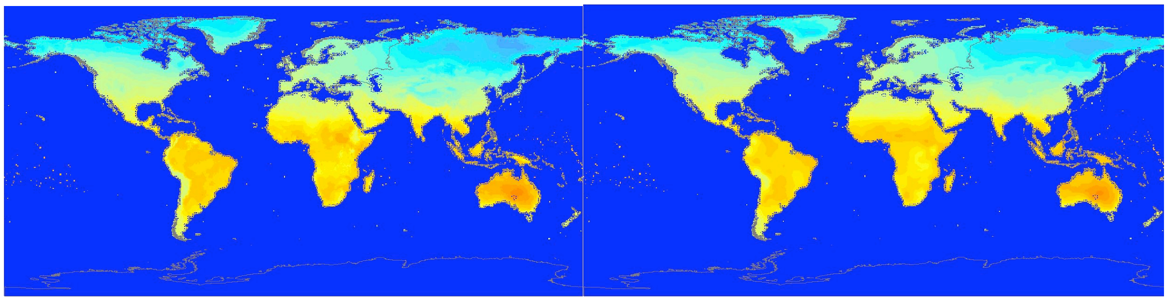


Figure 3: Global temperature profiles in winter month (Jan, 2006) (left: CRU TS data, right: gridded GSOD product with the same color scheme).

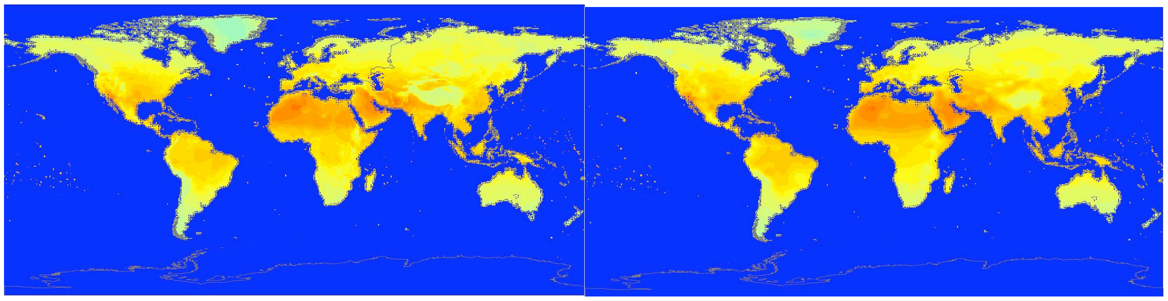


Figure 4: Global temperature profiles in summer month (July 2006) (left: CRU TS data, right: gridded GSOD product with th same color scheme).

regional temperature change for a certain time period to enhance the domain science research and application development.

7 Comparison with Existing Global Climate Dataset

7.1 Dataset description

Climatic Research Unit (CRU) TS (time-series), or CRU TS, datasets contain month-by-month variations of global climate information over the last century or so. CRU TS datasets are archived as high-resolution (0.5 x 0.5 degree) grids of monthly mean temperatures derived from more than 4000 weather stations distributed around the world. CRU TS data includes weather information such as cloud cover, diurnal temperature range, frost day frequency, precipitation, daily mean temperature, monthly average daily maximum temperature, vapor pressure and wet day frequency. At present, the British Atmospheric Data Center holds the CRU TS 3.0 datasets for the period 1901-2006 as well as the CRU TS 3.1 datasets for the period 1901-2009. In this study, the monthly temperature dataset from CRU TS 3.0 [8] is used for the comparison to the gridded daily product of GSOD temperature.

7.2 Comparison

Since CRU TS only has half degree grids of monthly average temperature over six earth continents (without Antarctica) , we filtered out all the data covering the ocean and Antarctica for comparison to the monthly mean surface temperature derived from the interpolated GSOD daily mean surface temperature over those six continents for comparison. Figures 3 and 4 display the spatial distribution of the average temperature of two specific months, specified by CRU TS dataset in winter and summer month (January and July) in 2006. As shown in Figure 3 and 4, the temperature distribution pattern exemplifies a good match between CRU TS data and our gridded, GSOD product in a majority of areas, while major difference exists around Himalaya mountain areas.

8 Conclusions and Future Work

This paper presented the method and workflow to transform NOAA Global Surface Summary of Day (GSOD) data into a more useful format to support climate change research. While the location of weather observation stations is embedded, date-based GSOD data can be further transformed into detailed gridded data products at very fine spatial and temporal scale. By deploying hybrid computer architecture and systems, interpolating global daily mean surface temperature in the past 30 years can be accomplished within two hours. The quality of interpolated GSOD products exemplifies satisfied quality in most regions over the continents in the world. The preliminary comparison between CRU TS data and our new gridded data products derived from GSOD shows a consistent match between these two datasets, with the major difference identified around boundaries of Tibetan plateau. With the increasing demands on the research of decadal climate change and its impact, our gridded GSOD data products can serve as a high fidelity benchmark datasets to validate and verify those finer scale climate simulation results. It can also be used as fine scale (both temporal and spatial) external forcing to investigate regional climate impacts. The future work will focus on the improvements of station selection for interpretation, and the topography-dependent heterogeneity of surface temperature measurement in the data generation procedure. Further comparison with other existing global climate datasets, such as NASA GISS datasets (http://data.giss.nasa.gov) and MODIS datasets (lpdaac.usgs.gov), will help to understand the different models and the output results for climate change research. Our gridded GSOD data product [i.e. global daily mean surface temperature grids at a resolution of 0.25 degree x 0.25 degree for a duration between 01/01/1982 and 12/31/2011] is now available upon request, and authors are making plans to make the product available via Distributed Active Archive Center for Biogeochemical Dynamics at Oak Ridge National Laboratory. All the datasets generated by this study are available upon request, and DOI was requested for those datasets.

9 Acknowledgement

This research was supported partially by the National Science Foundation through the award OCI-1047916. This research used resources of the Keeneland Computing Facility at the Georgia Institute of Technology, which is supported by the National Science Foundation under Contract OCI-0910735. Oak Ridge National Laboratory is managed by UT-Battelle LLC for the Department of Energy under contract DE-AC05-00OR22725.

10 References

[1] National Research Council. A National Strategy for Advancing Climate Modeling . Washington, DC: The National Academies Press, 2012.

[2] Trenberth, K. E., Anthes, R. A., Belward, A., Brown, O., Haberman, E., Karl, T. R., Running, S., Ryan, B., Tanner, M., and Wielicki, B., 2012: Challenges of a sustained climate observing system. In Climate Science for Serving Society: Research, Modelling and Prediction Priorities, Hurrell, J. W. and Asrar, G. eds., Springer, accepted.

[3] Oliver, M. A. and R. Webster. 1990. Kriging: a method of interpolation for geographical information systems. International Journal of Geographical Information Science, Vol. 4, No. 3. (1990), pp. 313-332.

[4] Tang, Tao. 2005. Spatial Statistic Interpolation of Morphological Factors for Terrain Development. GIScience & Remote Sensing. Volume 42, Number 2 / April-June 2005. pp 131-143.

[5] Cheng, T.; Zhang, Y.; Li, D.; Wang, Q. 2010, A component-based design and implementation for parallel Kriging library, Information Science and Engineering (ICISE), 2010 2nd International Conference on , vol., no., pp.1-4, 4-6 Dec. 2010.

[6] Srinivasan, B. V.; R. Duraiswami; and R. Murtugudde. 2010. Efficient kriging for real-time spatio-temporal interpolation. Online proceedings of the 20th Conference on Probability and Statistics in the Atmospheric Sciences.

[7] Vetter, J.S., R. Glassbrook, J. Dongarra, K. Schwan, B. Loftis, S. McNally, J. Meredith, J. Rogers, P. Roth, K. Spafford, and S. Yalamanchili, “Keeneland: Bringing heterogeneous GPU computing to the computational science community,” IEEE Computing in Science and Engineering, 13(5):90-5, 2011. http://dx.doi.org/10.1109/MCSE.2011.83 http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5999785

[8] CRUNECP, Mitchell, T.D. and Jones, P.D., 2005: An improved method of constructing a database of monthly climate observations and associated high-resolution grids. International Journal of Climatology 25, 693-712 doi:10.1002/joc.1181

High Efficiency Video Decoding on Multicore Processor

Hyeonggeon Lee[1], Jong Kang Park[2], and Jong Tae Kim[1,2]
Department of IT Convergence[1]
Sungkyunkwan University
Suwon, Korea

Department of Electrical and Computer Engineering[2]
Sungkyunkwan University
Suwon, Korea

Abstract**—***In this paper we present a High Efficiency Video Coding(HEVC) decoder implemented using multicore processor. HEVC can support Ultra High Definition (UHD) digital TV and resolution up to 8192x4320. It aims to achieve compression rate in the range of 50% bit-rate relative to existing standards. And decoding speed should be over the 30 fps(frames per second). Although multicore processors have sufficient performance and enough memory, HEVC software cannot make use it very well. So not only the proper architecture for HEVC is needed but also modified HEVC software should be considered. Gem5 simulator was used to simulate the performance of our decoder. Base on the simulation results we suggest appropriate multicore architecture platform which can satisfy the goal of HEVC decoding speed.*

Keywords: HEVC, Multicore, Parallelization

1 Introduction

In recent days, as digital display technology has remarkably developed and high-definition of digital TV is needed for various media resources, Joint Collaborative Team on Video Coding (JCT-VC) which ISO/IEC Moving Picture Experts Group (MPEG) and ITU-T Video Coding Experts Group (VCEG) have established develops the High Efficiency Video Coding (HEVC) standard. HEVC can handle the 8 K Ultra High Definition (UHD) and resolution up to 8192x4320. And it improves the data compression ratio compared to H.265/MPEG-4 AVC which is pre-version of HEVC. H.264/MPEG-4-AVC can be performed in general computer enough. But HEVC controls more fragmented unit for high definition video resources and adopts technologies to accomplish the parallelization tasks. So, new platform architecture which can realize the encoding and decoding these feature of HEVC tasks is needed. But there are so many possible architecture components and their composition. And also specification of each hardware component can influence the performance of HEVC encoding and decoding process. [1][6]

In section 2, we present basic architecture of HEVC encoder and decoder. In section 3, we describe the basic ideas which we proposed to find proper architecture in multicore platform for HEVC. In section 4, HEVC decoder software is modified to utilize the multicore architecture using parallelizing method and appropriate architecture is proposed based on simulation results. At last section, we define what is done in this paper and finish the paper.

2 Related works

2.1 Feature of HEVC

1) Wavefronts: In the side of increasing the possibility of parallelism in HEVC, we can define the multiple cores can be used in parallel. HEVC adopts the wavefronts concept which splits the picture into CTU rows. This each CTU can be processed in a different processor or thread. So if the architecture has many processing components which can handle the CTU rows, user may configure the encoding or decoding processor can be split into the many CTUs. With multiple processing components, increasing of split depth does not cause increasing of total processing time by using the parallelism. Not only case of wavefornts, there are many possible tasks which can utilize the parallelism of multiple processing components because almost processing unit has similar structure with CTUs.[1][2] [6]

2) Slices: HEVC divide a frame into slice which is groups of LCUs in scan order. It can be used for packetization in NAL unit when transmission in network. But more important role of slice is parallel processing in decoding the CUs. Because each thread or processor handle the each slice, so parallelize is improve. But every slice should contain the slice header which includes size, boundaries and parameters. This can distortion transmit rate and the dependencies at each slice's boundary can interrupt the decoding performance. So, in some cases, exclude slice information when encoding the picture. [1][3][4][6]

3) Dependency: For applying the pipelining or parallelize to HEVC decoder software, Dependency between each tasks or target CUs should be considered as a basis of

smooth decoding process. In case of task's dependency, all decode tasks should be carried out in regular sequence. So, between neighboring tasks, parallelizing cannot be adopted but pipelining is acceptable. Enter the specific task, for example decompress task, there is dependency between circumjacent CUs because of intra prediction directions as shown in Fig.3. So, when wavefronts is used for parallelizing the decoder, each row of CUs should be decoded after two CUs of previous row are decoded.[1][3][6]

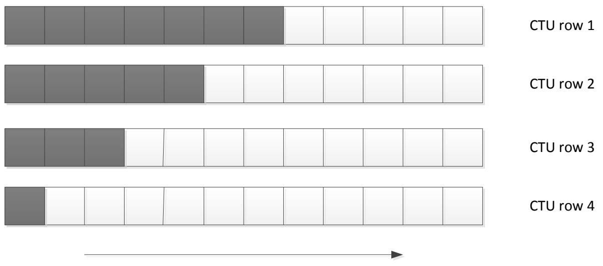


Fig. 1. Waveforнts Processing of CTUs

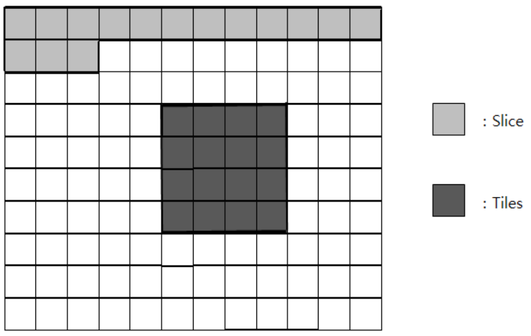


Fig. 2. Slices and Tiles

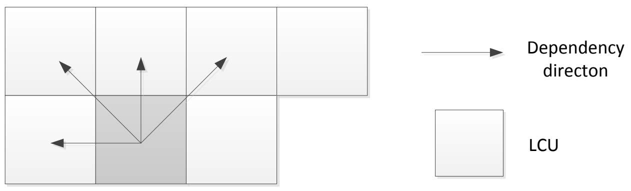


Fig. 3. Dependency Direction in Decompress

Each CU has their address number which start from 0 to (total CU numbers -1). This number cannot be used for slice and tile because they have their numbers. For example, in Fig.2, one tile has 16 CUs so, the address number of each CU is 0 to 15 and other tile has also number of 0 to 15 for included CUs. Foregoing address number can be used for wavefronts parallelize and after-mentioned address number also can be employed for parallelize using allocate each tile to processor or thread. As multi-processor architecture is fastest growing issue in hardware architecture, HEVC concentrates on parallelize decoding process which can make the best use of multi-processor architecture. So, in this paper we modify the HEVC software to put this feature to practical use.

2.2 HEVC Decoder

HEVC decoder goes along in the opposite direction of its encoding process. First step of decoder is Read NAL unit task. Because H.264 and HEVC is developed for the purpose of network transmission, encoded data take a form of NAL unit. NAL unit is made for efficiency transmission of picture stream data in network. It consist of parameter set block (SPS, PPS) and slice of picture data. So in this first function identifies what this block is and informs it. Second step is DecodeSlice which consist of initCU, decodeCU and DecompressCU. InitCU just initialize of decoding process. DecodeCU conduct entropy decoding of bitstream. In high efficiency mode of HEVC, entropy encoding conducts based context-based binary arithmetic coding (CABAC). DecompressCU which is the next step of decodeCU in DecodeSlice made up largest number of decoder process. Major roles of this part are dequantization, inverse-transformation, motion compensation and intra/inter prediction. After decompressCU, decoded picture is stored in decoded picture buffer (DPB). This decoded picture is using for deblocking filter (DF), sample adaptive offset (SAO), adaptive loop filter(ALF) and inter prediction. These three filters are part of ExecuteLoopFilter function. So, this step consume much time in access memory. Last step is write picture. When the one picture is decoded completely, this picture is written in decoded picture file. For the most features of each blocks, HEVC have tended to center around the question of how we can achieve more highly resolution pictures and how we can raise a compression ratio with not much more bitstream. But there is no clear solution for architecture to sufficiently handling the most high resolution video stream. To make the best use of HEVC structure, we should put knowledge of characteristics about HEVC to system-level design space exploration. We will see in section 3 how this methodology is being unveiled. [1][4]

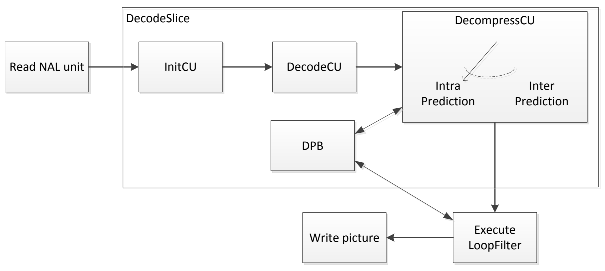


Fig. 4. HEVC Decoder Process

3 Proposed methodology of implementing multicore platform for HEVC decoding

In the present paper, we will see the methodology to take appropriate multicore architecture for HEVC decoder.

Before we apply the several CPU architectures, we should analysis HEVC using HM11.0 in the general PC environment so that we choose the primary tasks and define the relation between each task. Some basic features of HEVC can be come to the front from this process. And this process will offer further evidence for the selection of fundamental components of architecture which we will choose as a consideration for simulation. The point which we especially concentrate on is a task which can use the parallelized architecture and consumes much memory bandwidth. Because this is a main feature that different from H.264/MPEG-4. HM11.0 provides the performance information such as bit-size of each slice and total processing time when user performs encoding and decoding process with sample input file. And when we encode the input video file, we can configure most parameters of tasks in HEVC structure. Because as resolution of input video file is increasing and as what the user has more interest between image quality and compression efficiency, composition of encoder options, decoder structure and combination of each parameter set have a broad range. In addition, some tasks do not perform at some cases. So it is material to selecting the main component of architecture by profiling the HEVC using the HM11.0 software as benchmark software.

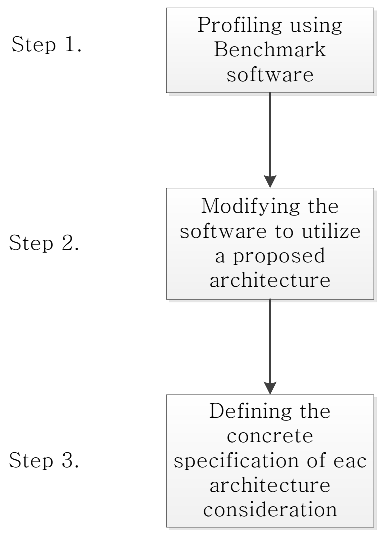

Step 1.
Profiling using Benchmark software
Step 2.
Modifying the software to utilize a proposed architecture
Step 3.
Defining the concrete specification of eac architecture consideration

Fig. 5. Proposed step-wise methodology

The second step decides the modified form of target software. Information for modifying the software is gained from first step because the tasks and architecture component which we should consider importantly are exposed come out from first step. From this step, designer should ensure that which performance indices will be considered main issue of the model. If the designer set a goal of much increase in compression rate, all tasks which can play a part of compression operation should be considered to be very important tasks.

At the last step, we would already know from previous step that which component has a great influence to a full architecture so that we should analyze that task deeper to make optimum configured components for reaching designer's goal. Naturally, other components of architecture will receive less attention. In this step, more detailed architecture requirement which can satisfy the designer's goal in terms of his performance set.

4 Implementation of HEVC decoder using multicore architecture

According to profiling result of decode function HM11.0, most of performance time is consumed in decompressCU function but initCU and decodeCU functions also have the portion not to be ignored. And because these three functions should be performed in serial with CU order, the dependency between each task is a point to be specially considered. But naturally, decompressCU function is most important part to be parallelize or pipelining for applying to multicore architecture.

HEVC software which is offered from standard association is not optimized to utilize the multicore architecture. And because HEVC is more complicated than H.264 to achieve 50 % higher compression ratio as maintaining the same PSNR, there are some possibility which can draw up the decoding speed using parallelize. But in the multicore environment with OS, it is not easy to guarantee the stable decoding speed.

In this paper, we obtain the trend of decoding speed with various hardware and software architecture using gem5 simulator with changing hardware components and each component's specification. So in this proposed implementation methodology, both software and hardware can be considered to analyzing decoding speed of HEVC decoder software and proposing appropriate architecture platform.[5]

4.1 Parallelizing of HEVC decoder software

HEVC decoder software is comprised of tiles, slices and CTU rows which are concepts for applying parallelizing. But decoder speed as increasing the number of cores is not changed largely. The reason why decoder speed is not changed is seen plainly through simulation result from gem5 simulator. The stat information which is the result of gem5 simulator tells the fact that however the number of core increased, HEVC decoder software can employ only a one core out of them. So with the original HEVC decoder software, the goal of decoding speed cannot be accomplished. In this reason, openMP is applied to HEVC decoder software. As the profiling results of HM11.0, the most important task is the decode function and CU is the processing unit in that function. Among the ways of using openMP in HEVC decoder software, we choose the decodeCU and decompressCU to parallelize applying region.

We make parallelizing using openMP. In this parallelizing, decodeCU is comprised in parallelizing the decompressCU. Because the time which is consumed during operating a one CU on decodeCU is just one tenth of that of decompresCU, specific CU can be decoded before that CU starts decompress.

But the start CU of each parallelized row should be decoded before it starts. Therefore, address of the number between 1~52 should be decoded before parallelized decompress starts as shown in Fig.6, there are totally three steps for decoding 104 CUs and both second and third steps are applied to openMP. In the decompressCU parallelizing method, each row has a gap of two CUs. It is because of dependency between CUs when intra-prediction is operated.[3]

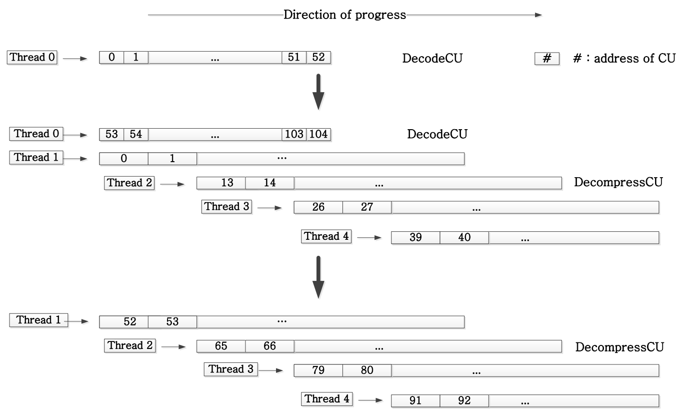


Fig. 6. Case 1 of Parallelize using OpenMP

4.2 Simulation Results

We simulate the HEVC decoder software on Gem5 simulator with X86 instruction set and CentOS environment. Default interconnection type is a coherence bus model and multicore type is SMP model. Used decoder software is HM 11.0 and the encoder software is also HM 11.0. HEVC decoder on the gem5 simulator works in practice. Because the Gem5 simulator offers a multicore architecture which has its all components and porting a real OS on the platform, the result of HEVC decoder on simulator is real YUV type video. [5]

First simulation result is decoding speed per number of core. We set the clock speed at 3.3 GHz and memory at 256MB. Although original HEVC decoder software which we named serial case cannot utilize the multicore architecture, the modified HEVC decoder in section 4 is used to simulate HEVC decoder software on multicore architecture. The region of biggest increasing in decoding speed is between 1 core and 3 cores. And after 3 core, decoding speed over the 30 fps which is the goal of HEVC decoding speed. The max number of threads which is used at the same time is 5 when a one decodeCU row and four decompressCU rows are decoded as shown in Fig.4. So the peak value of decoding speed is achieved when number of cores is 5. After then, decoding speed is decreased when 6 cores are used and it is saturated with 33.3 fps from 7 cores. So, we need more than 3 cores to derive a decoding speed over 30 fps when using multicore architecture with modified HEVC decoder software. And with this modified HEVC decoder software, more cores over 7 is not needed because the decoding speed is saturated after that. And next, same simulation with various clock speeds is done. Under the 3 GHz clock speed, 30 fps is not achieved. With 3.1 GHz clock speed and over the 4 cores, decoding speeds can achieve 30 fps. With all clock speed cases, most significant increasing of decoding speed is shown in range of 1 core to 3 core. But over the 3 cores, there are no regular trends in decoding speed. In some cases, decoding speed is increasing steadily, but in most cases, decoding speeds repeat increasing and decreasing. [2]

Second simulation option is memory size. Tasks which access and use a memory in HEVC decoder software are decompressCU, decode of parameter set, decode of coefficient and three kinds of loop filter. Especially, DPB(Decoded Picture Buffer) occupied much size of memory for saving a temporary decoded picture. Simulation result is represented as decode time of decode function which is composed of decodeCU and decompressCU. Decode speed is stabilized when memory size is over 64MB. If the memory size is smaller than 30 MB, HEVC decoder stops decoding operation because of resource unavailable. In this simulation environment, OS runs the only a one program. This result can says the minimum memory requirement for decoding HEVC software is 32 MB with 832x480 resolution video file. And to avoid the restriction of memory size, memory size should be over the 64 MB.

Based on these simulation results we can propose minimal cost multicore architecture platform which can satisfy the goal of HEVC decoding speed. Table1 shows that there are many combinations of specification of each component which can achieve the decoding speed of over the 30 fps. So we can choose minimal cost combination set which can make good use of parallelism of HEVC software. The proposed multicore architecture set consists of 3 cores, clock speed of 3.1GHZ and memory size of 64 MB with SMP.

TABLE I. SPECIFICATION OF MULTICORE ARCHITECTURE FOR HEVC

	Clock speed	Memory size	Number of cores
Over the 30 fps	3.1GHz(3 core) ~ 4GHz(1 core)	52 MB ~ 128 MB	1core(4GHz) ~8 core
Proposed Platform	3.1 GHz	64 MB (SMP)	3 cores (X86)

5 Conclusion

This paper suggests the multicore architecture which can fully achieve a goal of decoding speed. Target software is HEVC decoder which is a next generation video compression coding. So we find the appropriate multicore architecture aimed to specific software using proposed implementing multicore architecture platform methodology.

In the first step, we analyze the HEVC decoder software using profiling tool of visual studio 2010 in the first place. The profiling results give information that which task consumes most time in decoding time so that we decide on target task to modify the software for parallelizing. After then we apply the openMP to parallelizing software and proceed the simulation on Gem5 simulator. The main considerations are clock speed, number of cores and memory size. Through simulation, we figure out the final multicore architecture platform which can decode an encoded video file with decoding speed over 30 fps. So we can suggest multicore architecture for HEVC decoder with over the 3.1 GHz clock speed, over the 3 cores and over the 64 MB memory size with modified HEVC software for parallelizing.

6 References

[1] Gray J. Sullivan, "Overview of the High Efficiency Video Coding (HEVC) Standard", IEEE TRANSACTION ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, VOL. 22, NO. 12, DECEMBER 2012

[2] Frank Bossen, "HEVC Complexity and Implementation Analysis", IEEE TRANSACTION ON CICUITS AND SYSTEMS FOR VIDEO

[3] Chi Ching, "Parallel Scalability and Efficiency of HEVC Parallelization Approaches", IEEE TRANSACTIONS ON CIRCUIT AND SYSTEM FOR VIDEO TECHNOLOGY, VOL. 22, NO.12, DECEMBER 2012

[4] Mauricio Alvares-Mesa, "PRALLEL VIDEO DECODING IN THE EMERGING HEVC STANDARD", Acoustic, Speech and Signal Processing(ICASSP), 2012 IEEE International Conference on.

[5] Nathan Binkert, "The gem5 Simulator", ACM SIGARCH Computer Architecture New Volume 39 Issue 2, May 2011 pages 1-7

[6] Philippe Bordes, "An Overview of the emerging HEVC standard", IEEE TRANSACTIONS ON CIRCUIT AND SYSTEMS FOR VIDEO TECHNOLOGY, VOL. 22, NI.12, DECEMBER 2012

Design of an In-Memory Database Engine Using Intel Xeon Phi Coprocessors

Michael Scherger
Department of Computer Science
Texas Christian University
Fort Worth, TX, USA
Email: m.scherger@tcu.edu

Abstract – *(PDPTA'14) This research presents the design and initial implementation of a database engine using an Intel Xeon Phi co-processor. The many integrated cores (MIC) of the Xeon Phi make this hardware accelerator a natural computing platform for an in-memory database engine or server. The database tables reside in the memory space of the MIC thus supporting fast in-memory database applications. This achieved by developing a coalescing parallel memory manager to allocate parallel variables in the same manner that fields are created in a table using a SQL CREATE TABLE command. The SQL interface was created using a database driver toolkit that provides an interface to the Xeon Phi server and client application. Once the basic framework was established, the algorithms for SQL select, insert, update, delete, and join were created to manipulate database information in the memory of the Xeon Phi.*

Keywords: parallel databases, parallel hardware accelerators, special purpose architectures

1 Introduction

Massively data parallel computers and the SIMD model of parallel computation can be a natural model of parallel computation to consider for massively parallel database servers. Since the cores are extremely close to the parallel memory, fast parallel memory searching make it a natural platform for data parallel computing intensive applications. As described in Potter in [9] and [10], data parallel models of computation such as the SIMD, ASC, or SITDAC model conform to the concept of a parallel database server since the data can logically and physically partitioned similar to the data organization of a database table or spreadsheet [2][8][9][10][11] and [12].

This research paper discusses the initial design of an in-memory database server using a Intel Xeon Phi co-processors. This research will discuss the design considerations and challenges for a database server and SQL engine that interfaces with the memory of a hardware accelerated data parallel computer. This system design can promote the use of massively parallel computers as database servers for use in embedded database systems, real-time database systems, and fast parallel associative search engines.

Database management systems (DBMS) provide a structured mechanism for storing, organizing, and retrieving data in a way that is consistent with the database's format [14]. System software will allow data storage and access to a database without the user's knowledge about the internal data representation either in persistent storage or in the computer's memory. A DBMS usually has but is not limited to the following components [14]:

- Processors and main memory – the hardware of the DBMS for data selection and computation
- Secondary storage – disks for data persistence and offline storage
- Database manager – software for creating and maintaining databases, tables, fields, and relations
- Utilities – software for database maintenance, data integrity and security, and database repair
- Application development tools – software for database application development integrated into the DBMS
- Report writers – software modules for presentations and reports based on tables and queries from database information
- Design aids – software to assist in the design of databases, tables, fields, indexes, and relationships

The organization of this research paper is as follows. Section 2 will use the tracking and correlation problem in air traffic control as a motivating example. Section 3 present an overview of the Intel Xeon Phi co-processor and system software. Section 4 will present the hardware and physical design of the database server including the mapping of table records into the memory of the parallel computer. Section 5 will discuss the techniques of sequential and parallel database query processing. Section 6 will present the system software design of the parallel SQL engine and the algorithms for the basic database server operations. Section 7 will discuss the conclusions and future research.

2 Example Application: ATC Tracking and Correlation

Consider a real world and real time application of air traffic control. The following example is an extremely simplified version of the air-traffic tracking and control problem, but provides enough detail to illustrate the system software design for a parallel database server and SQL interface [11]. Some of the basic tasks of air traffic control are:

- Tracking and correlation – The radar will generate reports of flights of returns that must be correlated to tracks of flights currently in the system.
- Conflict detection – The computer system must then determine if there are any tracks/flights that will conflict/collide with a time look ahead of a predetermined number of minutes or miles.
- Flight plan update – Based on the tracking and correlation information and combining it with the conflict detection, the flight plan information will be updated.

There are many other important tasks in air traffic control, but this is an example of a real-time processing problem [6],[7], and [8]. There are also hard deadlines for computation imposed on the above tasks. They must be completed prior to the next hard deadline in this real time system.

The flight plans and tracks from radar can be stored in a simplified tabular format (flat table) in a database table similar to illustration in Figure 1.

AID	LAT	LON	ALT	HDG	AS
CO56	40.55	81.26	270	0	450
AA123	39.9	84.31	60	225	275
UA722	41.24	81.51	150	0	525
AA1223	40.0	82.53	290	315	305
…	…	…	…	…	…
CA2341	39.54	84.13	70	90	330

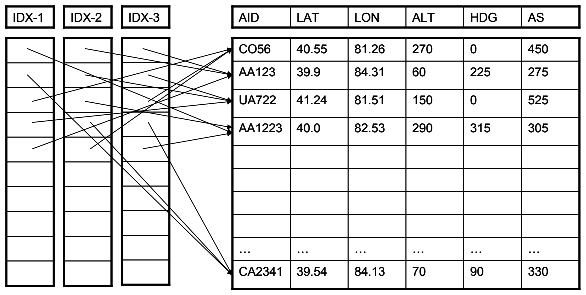
Figure 1: Sample database to store flight plan information.

The ATC system software will/may have to perform the following operations when receiving a new set of track information.

- Insert a new flight into the table. As aircraft enter the airspace, they need to be stored into the flight table. This involves searching for an open/free record in the table and then copying the flight information into that newly created record.
- Deleting a flight from the table. As aircraft leave the airspace, they need to be deleted from the flight table. This involves searching for the record of the flight to delete and marking that the record is inactive.
- Selecting a flight from the table. Selection involves identifying one or more flights for further processing. The selection must scan the data in the fields for this table and then return that information back for further processing.
- Updating the flight information. Updating a flight begins with a search followed by a copy of new information into the selected record from the track information.

Each of these frequent operations (insert, select, update, delete) requires some type of a parallel memory search. In the case of insert, the search operation is for an open record in the table. In the case of the select, update, and delete operations, the search required is based on the data stored in the records of the table. This is a contextual search or associative based search.

The most common method to improve search performance in a database server is to use index tables. This is illustrated in the Figure 2.



AID	LAT	LON	ALT	HDG	AS
CO56	40.55	81.26	270	0	450
AA123	39.9	84.31	60	225	275
UA722	41.24	81.51	150	0	525
AA1223	40.0	82.53	290	315	305
…	…	…	…	…	…
CA2341	39.54	84.13	70	90	330

Figure 2: Flight table illustrating the use of index tables to improve performance.

An index table stores the record indexes based on some ordering criteria or sorting functions. In Figure 2, an index table may store pointers to the indices for the aircraft sorted by aircraft id. Another index may store pointers to indices for the aircraft based on altitude. Finally another index may store pointers to the indices for the aircraft based on airspeed.

In theory, a database table can have one or more indices for each field. However, this dramatically reduces the performance of the insert, update, and delete operations at the benefit of doing fast searches [1][3][4][5]. As new records (flights) are entered into the table, the index tables need to be updated and maintain their sorted order. The constant resorting of each index table becomes increasingly computational demanding. The same is true for the delete and update operations when the flight information changes. The performance degradation is further amplified when multiple index tables must change.

3 Overview of the Intel Xeon Phi

The Intel Xeon Phi co-processors have 60 in-order Intel MIC architecture cores running at 1 GHz. The Intel MIC architecture is based on the x86 ISA, extended with 64-bit addressing and 512-bit wide SIMD vector instructions and registers. Each core supports 4 hardware threads. In addition to the cores, there are multiple on-die memory controllers and other components.

As shown in Figure 3, each core has a newly designed Vector Processing Unit (VPU). Each VPU unit contains 32 512-bit vector registers. To support the new vector processing model, a new 512-bit SIMD ISA was introduced. The VPU is a key feature of the Intel MIC architecture based cores. Fully utilizing the vector unit is critical the best performance. The Intel MIC architecture cores do not support other SIMD ISA's such as MMX, SSE, or AVX.

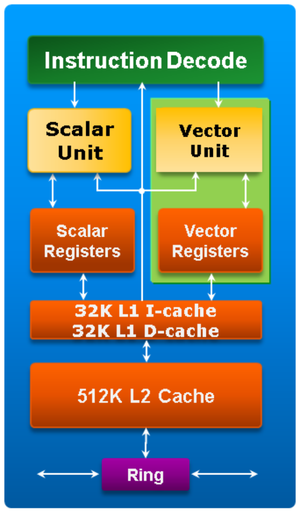


Figure 3: Intel Xeon Phi MIC core block diagram.

Each core has a 32KB L1 data cache, a 32KB L1 instruction cache, and a 512KB L2 cache. As shown in Figure 4, The L2 caches of all cores are interconnected with each other and the memory controllers via a bidirectional ring bus, that effectively creates a shared last-level cache of up to 32 MB. The design of each core includes a short in-order pipeline. There is no latency in executing scalar operations and low latency in executing vector operations. Since the in-order pipeline is short, the overhead for branch misprediction is low.

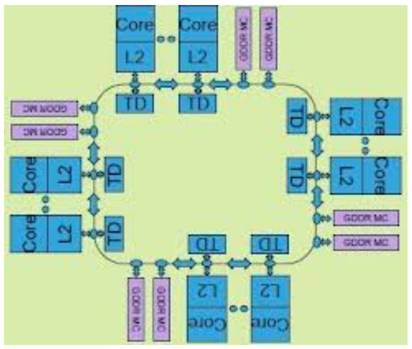


Figure 4: Logical MIC core layout and ring communication bus.

4 Database Engine Hardware Design and Architecture

There are a few assumptions regarding the design of the parallel database server [11].

1. The database, tables, and records in the parallel database server are memory resident. Storage is completely volatile and there is no persistent storage in the cells or array memory implemented in this design. For real-time computation, storing data and record information in in secondary storage is costly in terms of access time. Having the data reside in memory, close to the processing elements is more conducive for real-time applications.

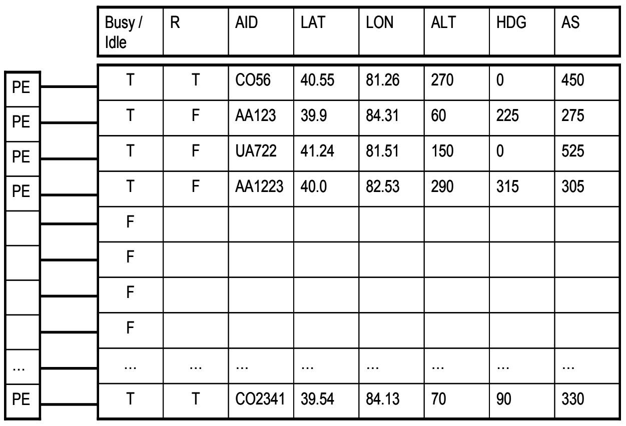
2. The data parallel memory map is similar to the field layout planned of a database table. If TABLE_A has fields F_1, F_2, F_3 created in that order, then the parallel memory map will have parallel variables F_1[$], F_2[$], and F_3[$] located in lower to higher parallel memory addresses.

3. The number of actual processing elements is fixed during the execution of the parallel database server. This is not an unrealistic assumption since the Intel Xeon Phi has a fixed number of cores (or hyper-thread processors).

4. The amount of memory per processing element is fixed. Again, the memory in the Intel Xeon Phi separate "parallel memory space" than the memory of the host computer. Albeit the parallel memory space is often smaller than the host memory, for most database applications, the amount of parallel storage is adequate.

Since this model is using massively parallel search and responder processing as a model of data parallelism, database index tables are no longer required. Each database field (column) can be searched for the desired value in constant time. Data parallelism can also support efficient software for associative searches.

The cores, or processing elements (PE) of the Intel Xeon Phi will be used to assist in the basic database operations and searching. This is illustrated in Figure 5. In this figure, the database table is superimposed on the memory and processing elements of a SIMD computer. Two additional fields have been prefixed to the table: a busy-idle flag to indicate if the PE or record is active and a responder flag used for search operations. Using this approach, each individual record is located in the memory of a PE. Using massive parallel searching, processing elements can scan their individual memories and set the responder flag or turn their busy-idle flag on or off.



	Busy / Idle	R	AID	LAT	LON	ALT	HDG	AS
PE	T	T	CO56	40.55	81.26	270	0	450
PE	T	F	AA123	39.9	84.31	60	225	275
PE	T	F	UA722	41.24	81.51	150	0	525
PE	T	F	AA1223	40.0	82.53	290	315	305
	F							
	F							
	F							
	F							
...	...	...	...	...	...	...	...	...
PE	T	T	CO2341	39.54	84.13	70	90	330

Figure 5: Flight table superimposed onto the PEs and memory of a SIMD computer.

Database tables are dynamic objects; there is typically no *a priori* knowledge of the number of table records. If the number of records in a table exceeds the number of physical PEs in the system (parallel memory overflow) the database server will use a cyclical data placement strategy when inserting new records. This is a form of virtual parallelism that is maintained by the parallel database server and not the operating system. This cyclical placement will manage multiple tables with multiple folds in an interleaved fashion as determined by the amount of data in the tables. For example, in Figure 6, Table A utilizes only 4 PEs, while the number of records in Table B has exceeded the number of PEs resulting in multiple folds.

An insert operation for Table A will add a new record into the area occupied by fold 1 for table A. For table B, the next insert operation will be in fold 2. If enough records are added to Table A to exceed the capacity of fold 1, a new fold will be created in the free memory space to the right of fold 2 of Table B [11].

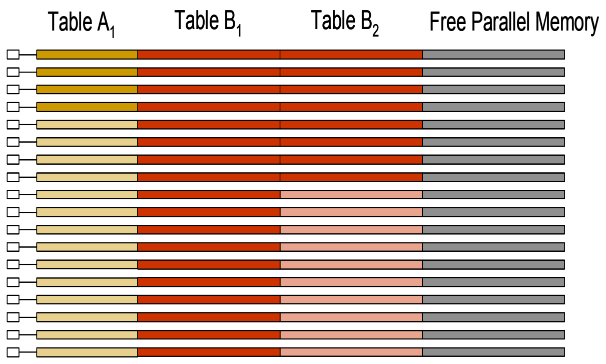


Figure 6: Multiple database tables, table folds, and unused parallel memory.folds, and unused parallel memory.

By necessity, data memory management becomes the responsibility of the parallel database server instead of the parallel compiler or other system software [11]. A coalescing parallel memory manager (CPMM) was developed to keep track of table, field and fold addresses. Figure 7 illustrates some of the administrative data structures that must be maintained for folded tables.

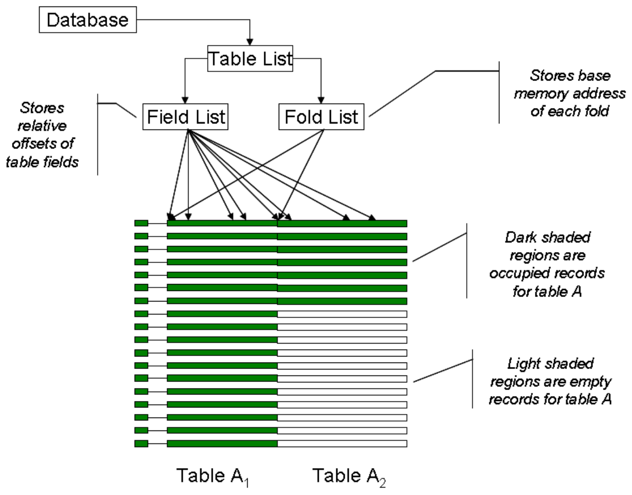

Database
Table List
Stores relative offsets of table fields
Field List
Fold List
Stores base memory address of each fold
Dark shaded regions are occupied records for table A
Light shaded regions are empty records for table A
Table A_1
Table A_2

Figure 7: Data structures for logical database tables, folds, records, and fields.

The parallel database server will maintain the controlling data structures to manage the database, tables, folds, records, and column addresses. These data structures reside in the sequential memory of the control unit or front-end computer. Dark shaded regions in this figure represent active records in the table. Note that there are two folds for this table and the field list is replicated for each fold. The table fold is an absolute parallel memory address while the field address is a relative parallel memory address. By adding the two memory addresses together, the physical memory address for a database field within a fold can be determined. The parallel memory manager also created extra hidden table fields used for basic database operations (described in a later section). These hidden table fields included several responder bits, a busy/idle flag, and timestamp fields for record insertion, selection, and update.

5 Sequential and Parallel Query Processing

Query processing refers to the range of activities involved with extracting data from a database. The activities include translation of queries in high-level database languages into expressions that can be used at the physical (storage) level. The fundamental steps a database server must perform when processing a database query appear in Figure 8:

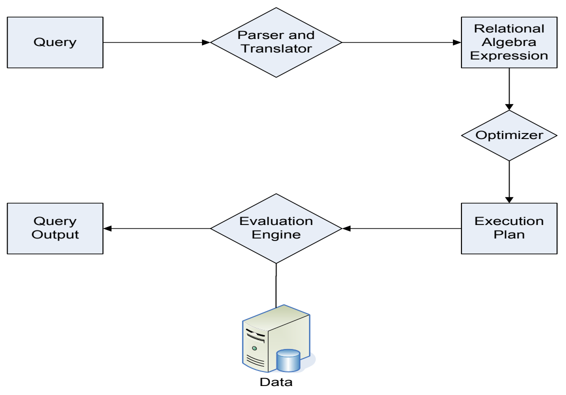


Figure 8: Major functional components of database query processing.

Before query processing can begin, the system must translate the query into a usable form. A language such as SQL is appropriate for software application development, but is not amenable to be the system's internal representation of a query. As shown in Figure 3, the first step the system must take in query processing is to translate a given query into its internal form. This translation process is similar to the work performed by the parser of a compiler. In generating the internal form of the query, the parser checks the syntax of the user's query and verifies that the query names appear in the database. The system then constructs a parse tree representation of the query, which it then translates into a relational algebra expression.

The sequence of steps in query processing is representative. Not all databases exactly follow these steps. However, the concepts that have been described form the basis of query processing in databases.

5.1 Sequential Query Processing Algorithms

There are several sequential query processing algorithms defined in the literature [14] and [15]. Each algorithm has a particular use when the query processing evaluation takes place.

The most relevant query processing algorithm related to this research is the A1 – Linear Search algorithm, which is now described. In a linear search, the system scans each file block and tests all records to see whether they satisfy the selection condition. An initial seek is required to access the first block of the file. The cost of linear search, in terms of number of disk operations, is one seek plus b_r block transfers, where b_r denotes the number of blocks in the file. Equivalently, the time cost is $t_S + b_r * t_T$.

Although the A1 – Linear Search algorithm may be slower on sequential computers than other algorithms for implementing selection and other query processing tasks, it is the most natural algorithm in terms of conversion to a massively parallel equivalent since the linear search on a parallel variable can be accomplished in constant time on SIMD (or MASC) computers assuming the database can be held entirely in memory.

6 System Software Design and Architecture

Now that the basic parallel memory management issues have been addressed, the system software design of the database server is described.

A client application will use the database driver manager to interface with the client database driver. The client database driver communicates with the SQL Engine. The SQL Engine will call process these instructions and then call the appropriate parallel database server, where there will be a corresponding function call to perform an operation in the memory of the parallel computer. The parallel database server will then receive the request from the database driver and control the databases, tables, records, and columns in the parallel memory.

6.1 Parallel SQL Insert Algorithm

The task of the parallel SQL insert operation is to insert new data into a free record located anywhere in the table in any fold. An example of the SQL INSERT statement is the following:

```
INSERT INTO FLIGHTS( AID, LAT, LON, ALT, AS )
VALUES( 'CO128', 43.39, 83.67, 190, 450 )
```

This insert statement will insert a new record into the FLIGHTS table (reference the database table in Figure 4) and assign the respective values to the AID, LAT, LON, ALT, and AS fields.

For inserting a record into a parallel memory space, the basic parallel insert algorithm is the following:

```
Algorithm Par_SQL_Insert( RecordData )

  open_record_found = FALSE

  For each table fold

    Perform associative search on the
    table's BI field where BI field is
    false (i.e. record is empty – there
    may be multiple records returned)

    if ( idle records found )
       select one record;
       BI = TRUE
       open_record_found = TRUE
       break

    // no open record is found
    // in any fold
    if ( open_record_found == FALSE )
      create a new fold
      select first record in the new fold
      BI = TRUE
      break

  Copy the data into the parallel memory record
  Return success or failure
```

Figure 8: Parallel SQL insert algorithm.

The algorithm Figure 8 begins by searching for an open or idle record in each of the table folds in turn. If idle records are found, then PE identification number and the fold select one record and field addresses are used to copy the data into the parallel memory record. If no idle record is found, then the parallel memory manager must create a new fold. This can be accomplished by allocating space from the unused space in parallel memory the same width as previous folds and recording the new base address in sequential memory. Since a new fold is created, the parallel memory manager can select any PE for the insertion; e.g. the first PE (lowest PE id number) can be used. The basic parallel search can be done in *O(1)* time. However, since each table fold may have to be scanned, the running time is *O(#folds)* which is typically small and normally still *O(1)* since the number of folds normally constant and not a function of higher complexity.

6.2 Parallel SQL Delete Algorithm

The task of the parallel SQL delete operation is to delete records according to some searching or selection criteria. An example of the SQL DELETE statement is the following:

```
DELETE FROM FLIGHTS
WHERE AID = 'NW 545'     /* delete criteria */
```

This delete statement will delete all records where the AID (aircraft ID) is 'NW 545'. For deleting a record from the parallel memory space, the parallel delete algorithm is the following:

```
Algorithm Parallel_SQL_Delete( DeleteCriteria )
returns Boolean

  For each table fold

    Perform associative search where the Delete
    Criteria is TRUE and set responders
    appropriately

    If (the responder is TRUE)
       Reset the Busy-Idle flag

    If all records in the current fold are idle
       CPMM marks the current fold as free

  Return success or failure
```

Figure 9: Parallel SQL delete algorithm.

The algorithm in Figure 9 begins by looping through each table fold and having each cell evaluate the appropriate fields as specified in the delete criteria clause. For those cells where the delete criteria clause is True, the responder busy-idle flag is reset. If all the records in a given fold have their busy-idle flag reset, the coalescing parallel memory manager (CPMM) marks that fold as completely unused and returns it to the free pool of parallel memory. The basic parallel delete can be done in *O(1)* time. However, since each table fold will have to be scanned, the running time is *O(#folds)*.

7 Conclusions and Future Work

This research paper has presented an initial design of an in-memory database engine utilizing Intel Xeon Phi co-processors. Also presented was the system software design and interface for sequential programs and applications to interface with the server. This was achieved by designing and developing the set of algorithms for common database operations that would support the functionality of a parallel database server. The SQL operations presented include insert and delete and execute in O(#folds) steps. The update and selection operations are similar. The design of a coalescing parallel memory manager was also developed to manage large tables and virtual parallelism.

An area of future research explores how the parallel database handles virtual parallelism. The present design uses a coalescing parallel memory manager to control the table folds in the memory of the parallel computer. This parallel memory manager is a built in component of the parallel database server because the Xeon Phi environment assumed that the number of processing elements was fixed at runtime and could not change.

Another area of future research could explore how the tables, records, and fields are physically mapped to the memory of the parallel computer. Presently, the parallel memory map as shown in Figures 6 and 7 indicate that the parallel variables are allocated for all processing elements in a given fold regardless of the number of records actually storing information. This leads to a waste of processing elements for tables with only a few records.

8 References

[1] Babb, E, "Implementing a Relational Database by Means of Specialized Hardware", *ACM Transactions on Database Systems*, Vol., 4, No. 1, 1979, pp. 1-29.

[2] Batcher, Kenneth, "The STARAN Series E, *Proceedings of the International Conference on Parallel Processing*, 1977, pp. 140-143.

[3] Berra, P. Bruce, "Some Problems in Associative Processor Applications to Database Management", *National Computer Conference and Exposition*, Vol. 43, 1974, pp. 1-5.

[4] DeFiore, Casper R, P. Bruce Berra, "A Quantitative Analysis of the Utilization of Associative Memories in Data Management", *IEEE Transactions on Computers*, Vol. c-23, No. 2, February, 1974, pp. 121-132.

[5] Hsiao, D., and M. J. Menon, "Design and Analysis of a Multi-Backend Database System for Performance Improvement, Functionality Expansion and Growth (Part 1)", *Technical Report, OSU-CISRC-TR-81-7*, The Ohio State University, Columbus, Ohio, July, 1981.

[6] Jin, Mingxian, Johnnie Baker, and Kenneth Batcher, "Timings for Associative Operations on the MASC Model", *Proc. of the 15th International Parallel and Distributed Processing Symposium (Workshop in Massively Parallel Processing)*, abstract on page 193, full text on CDROM, April 2001.

[7] Lockheed Martin Company - formerly Loral Defense Systems, *ASPRO-VME Parallel/Associative Computer: Technical Overview*, Oct. 1993.

[8] Meilander, Will, Johnnie Baker, and Mingxian Jin, "Importance of SIMD Computation Reconsidered", *Proc. of the 17th International Parallel and Distributed Processing Symposium (Workshop on Massively Parallel Processing)*, abstract on page 266, full text on CDROM, April 2003.

[9] Potter, Jerry L., *Associative Computing: A Programming Paradigm for Massively Parallel Computers*, Plennum Press, New York, NY, 1992.

[10] Potter, Jerry, Johnnie Baker, Stephen Scott, Arvind Bansal, Chokchai Leangsuksun, and Chandra Asthagiri, "ASC: An Associative Computing Paradigm", *Computer*, Nov. 1994, pp. 19-25.

[11] Scherger, Michael, "An Object Model Framework, Runtime Environment Support, and Database System Software for a Multiple Instruction Stream Associative Model of Parallel Computation", *PhD Dissertation*, Department of Computer Science, Kent State University, Kent, Ohio 2005.

[12] Michael Scherger, Johnnie Baker, and Jerry Potter, "Multiple Instruction Stream Control for an Associative Model of Parallel Computation", *Proc. of the 16th International Parallel and Distributed Processing Symposium*, abstract on page 266, full text on CDROM, April 2003.

[13] Scherger, Michael, Johnnie Baker, and Jerry Potter, "An Object Oriented Framework for and Associative Model of Parallel Computation", *Proc. of the 16th International Parallel and Distributed Processing Symposium*, abstract on page 166, full text on CDROM, April 2003.

[14] Scherger, Michael, "On the Design of a Massively Parallel Database Server for In-Memory and Real Time Database Applications", Proceedings of the Int'l Conf. on Parallel and Distributed Processing Techniques (PDPTA), Las Vegas, June, 2007.

[15] Silberschatz, Abraham, Henry F. Korth, and S. Sudarshan, *Database System Concepts, 4th ed.*, McGraw-Hill, Boston, MA, 2002.

[16] Su, Stanley Y. W., *Database Computers: Principles Architectures and Technologies*, McGraw-Hill, New York, NY, 1988.

[17] Intel Corporation, Intel Xeon Phi Coprocessor Developer's Quick Start Guide, Version 1.7, 2013.

[18] Intel Corporation, Intel Xeon Phi Coprocessor Architecture, 2013.

[19] Intel Corporation, Intel Xeon Phi Coprocessor System Software Developer's Guide, 2013.

Accelerating Medical Image Registration Using a SIMD Array

I. K. Jeong[1], M. S. Kang[1], C. H. Kim[2] and J. M. Kim[1,*]
[1]School of Electrical Engineering, University of Ulsan, Ulsan, South Korea
[2]School of Electronics and Computer Engineering, Chonnam National University, Gwangju, South Korea

Abstract - *Medical image registration plays an important role in medical imaging in the early detection of cancers. An essential component in most medical registration approaches is resampling algorithms. These algorithms, however, demand tremendous computational power associated with similarity computation. The increasing availability of parallel computers makes parallelizing these tasks an attractive option. This paper presents parallel approaches for the resampling algorithms using a representative parallel Single Instruction, Multiple Data (SIMD) processor array to meet the computational requirements. This paper also presents not only a general theory of resampling algorithms including rotation, scaling, and translation, but also parallel implementations of these algorithms on the SIMD processor array. Experimental results show that parallel approaches achieve a speedup of 2.6x over the FPGA implementations with the same clock frequency of 80 MHz.*

Keywords: Medical image registration, parallel processing, SIMD processor arrays, resampling algorithms

1 Introduction

In medical imaging techniques, an important step for intensity-based image registration is resampling [1][2]. It is utilized when a discrete image is transformed into a new set of coordinate points and changes the number of sample (or pixels) per unit length of the directions of the image. During transforming parameters, resampling including rotation, scaling, and translation is estimated by geometrically mapping intensity coordinates in the reference (fixed) image to corresponding locations in the sensed (moving) image. However, these resampling algorithms require tremendous computational power due to the iterative nature of the algorithms.

Application-specific integrated circuits (ASICs) can meet the needed performance for such algorithms, but they provide limited, if any, programmability or flexibility needed for varied application requirements. General-purpose microprocessors (GPPs) offer the necessary flexibility. However, they will not be able to meet the much higher levels of performance required by emerging medical imaging applications on higher resolution images. This is because they lack the ability to exploit the full data parallelism available in these applications.

Among many computational models available for imaging applications, single instruction multiple data (SIMD) processor arrays are promising candidates for application-specific applications including medical imaging since they replicate a simple processing element (PE), data memory, and I/O to provide high processing performance with low node cost. Whereas instruction-level or thread-level processors use silicon area for large multiported register files, large caches, and deeply pipelined functional units, SIMD processor arrays contain many simple processing elements for the same silicon area. As a result, SIMD processor arrays often employ thousands of PEs while possibly distributing and co-locating PEs with the data I/O to minimize storage and data communication requirements.

This paper presents parallel approaches for the resampling algorithms to meet the computational requirements using a representative SIMD array architecture. This paper also evaluates the impact of the parallel approaches on processing performance and compares with the performance of FPGA (Field Programmable Gate Array) solutions. Experimental results show that our parallel approaches achieve a speedup of 2.6x over the FPGA implementation using modified compensated CORDIC [3] with the same clock frequency of 80 MHz.

The rest of the paper is organized as follows. Section 2 presents a SIMD processor array architecture used in this paper and parallel approaches for resampling algorithms. Section 3 describes parallel implementations of rotation, scaling, and translation algorithms. Section 4 evaluates the performance of the resampling algorithms, and Section 5 concludes this paper.

2 Parallel Approaches for Medical Image Registration using SIMD Processor Arrays

2.1 SIMD Processor Array Architecture

A block diagram of the SIMD model [4] used here is illustrated in Figure 1. This SIMD processor architecture is symmetric, having an array control unit (ACU) and an array consisting of processing elements (PEs). When data are distributed, the PEs executes a set of instructions in a lockstep fashion.

* Corresponding author.

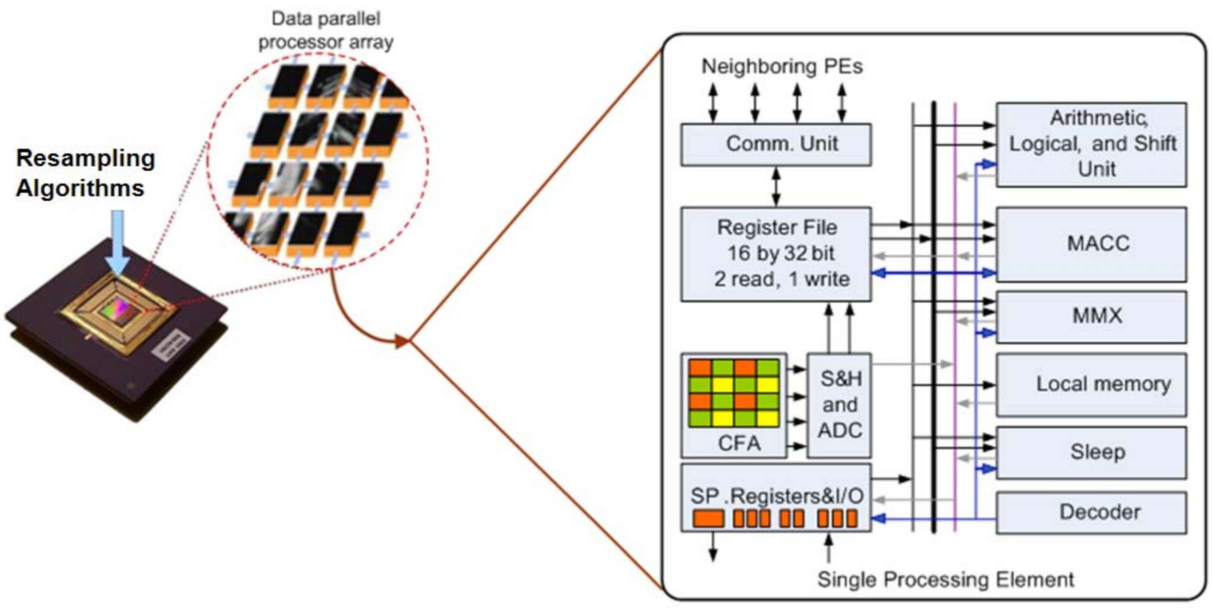

Data parallel processor array
Resampling Algorithms
Neighboring PEs
Comm. Unit
Register File 16 by 32 bit 2 read, 1 write
CFA
S&H and ADC
SP .Registers&I/O
Arithmetic, Logical, and Shift Unit
MACC
MMX
Local memory
Sleep
Decoder
Single Processing Element

Fig. 1. A block diagram of a SIMD processor array.

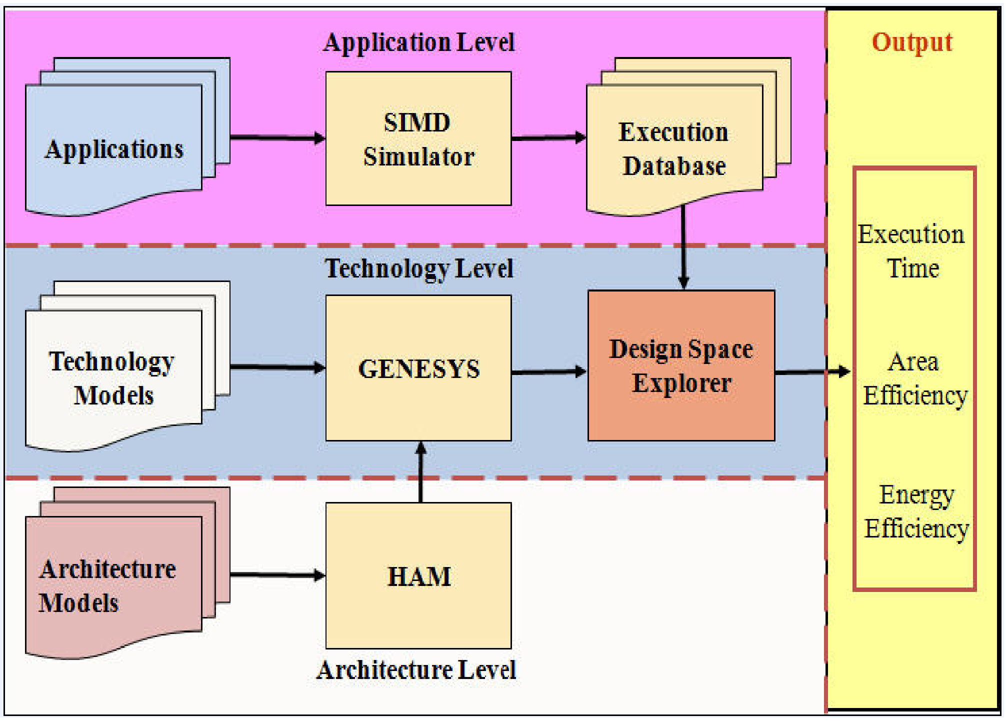


Fig. 2. Methodology infrastructure for the SIMD array simulation.

With 4x4 pixel sensor sub-arrays, each PE is associated with a specific portion (4x4 pixels) of an image frame, allowing streaming pixel data to be retrieved and processed locally. Each PE has a reduced instruction set computer (RISC) datapath with the following minimum characteristics:

- ALU – computes basic arithmetic and logic operations,
- MACC – multiplies 32-bit values and accumulates into a 64-bit accumulator,
- Sleep – activates or deactivates a PE based on local information,
- Pixel unit – samples pixel data from the local image sensor array,
- ADC unit – converts light intensities into digital values,

- Three-ported general-purpose registers (16 32-bit words),
- Small amount of local storage (64 32-bit words),
- Nearest neighbor communications through a NEWS (north-east-west-south) network and serial I/O unit.

2.2 Methodology Infrastructure

Figure 2 shows a methodology infrastructure that is divided into three levels: application, architecture and technology. At the application level, an instruction-level simulator was used to profile execution statistics such as cycle count, dynamic instruction frequency and PE utilization by retargeting and optimizing the resampling algorithms based on the architecture and its execution properties. At the architectural level, the heterogeneous architectural modeling (HAM) [5] of functional units for the specified SIMD array was used to calculate the design parameters of the PE configuration. The design parameters were then passed to the technology level. At the technology level, the Generic System Simulator (GENESYS) [6] was used to calculate technology parameters such as latency, area, power and clock frequency. Finally, a design space analysis tool collected and combined the database information (e.g., cycle times, instruction latencies, instruction counts, areas and powers of the functional units) obtained from the application, architectural and technology levels in order to determine execution times, area efficiency and energy efficiency.

2.3 Parallel Approaches for Resampling Algorithms

We implement resampling algorithms in parallel using the SIMD model. When data is distributed, a small pixel region (4x4 pixels) of the entire image space is assigned to each PE as shown in Figure 3. Then, the PEs execute a set of operations in a lockstep fashion.

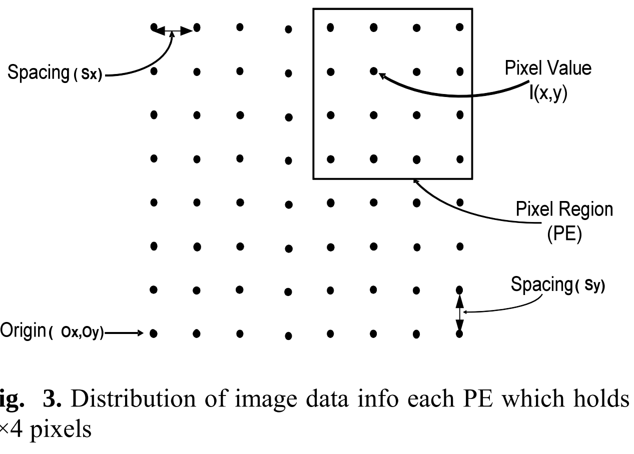


Fig. 3. Distribution of image data info each PE which holds 4×4 pixels

In medical image registration, resampling is a phase to transform the image I (x, y) to a rotated, scaled, and translated version of this image, I' (x, y), defined as

$$I'(x,y) = I(\sigma(x\cos\alpha + y\sin\alpha) - x0, \sigma(-x\sin\alpha + y\cos\alpha) - y0) \quad (1)$$

where α is the angle of rotation, σ is the factor of scaling, and (x_0, y_0) is shift amount by x_0, y_0 against x, y axes, respectively.

3 Parallel Implementations of Rotation, Scaling, and Translation

3.1 Rotation Algorithm

The rotation algorithm is an essential component of medical image processing. The basic image rotation operation is defined as

$$\begin{bmatrix} x' \\ y' \end{bmatrix} \equiv \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \quad (2)$$

A pixel at position (x, y) in the original image is mapped to the position (x', y') in the destination image by following the rotation of angular magnitude α [7].

Many different implementations of this algorithm have been developed to meet the computation requirements. This study prefers to overcome to computational burden by using a parallel implementation of a skew transformation where each pixel is shifted in parallel with each coordinate axis by means of the neighbor communication unit of PE. The skew transformation algorithm takes the rotation matrix in (2) and splits it on the multiplication of three matrixes, defined as

$$\begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix} = \begin{bmatrix} 1 & -\tan(\alpha/2) \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ \sin\alpha & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & -\tan(\alpha/2) \\ 0 & 1 \end{bmatrix} \quad (3)$$

These three matrixes are applied independently to the pixels of the image to find out the new location of a given pixel in the picture. The first and the last matrixes cause a skew west in the image. After multiplying the first or the last matrix by the coordinates of a given pixel, (4) is produced.

$$\begin{bmatrix} 1 & -\tan(\alpha/2) \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \equiv \begin{bmatrix} x - y \times \tan(\alpha/2) \\ y \end{bmatrix} \quad (4)$$

where the skew west is a displacement of the rows to the left by the multiplication of the row number by tan (∝/2).

The second matrix in (3) causes a skew north. After multiplying the second matrix by the coordinates of a given pixel, (5) is produced.

$$\begin{bmatrix} 1 & 0 \\ \sin\alpha & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \equiv \begin{bmatrix} x \\ y + x \times \sin\alpha \end{bmatrix} \qquad (5)$$

where the skew north is a displacement of the columns to the top by the multiplication of the column number by sin (∝).

According to the above skew transformation, the image expands after the west and north skews in (3). Since the SIMD processor array is fixed and the image is filled in completely, there is no space to hold the expanded image part. Thus, the image must be compressed first to be able to skew the image. Since the skew west expands the image to the left and the skew north expands to the top, the image is compressed to the right-bottom corner of it, as shown in Figure 4(a).

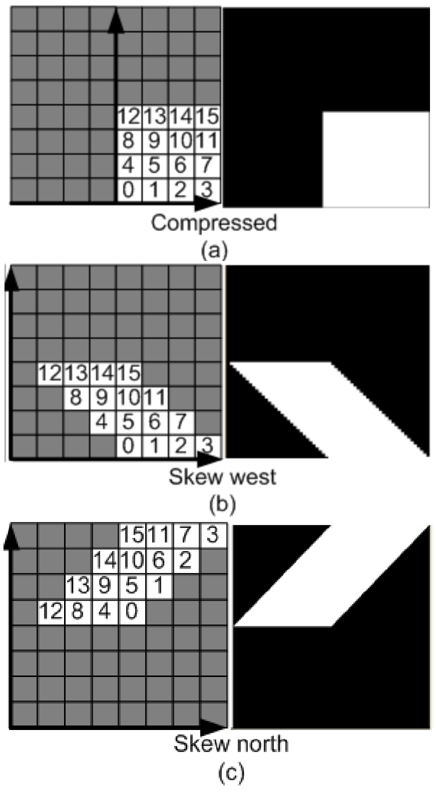


Fig. 4. Three steps for rotation: (a) compression, (b) skewwest, (c) skew north.

Figure 4 demonstrates two kinds of figures. The first image is compressed into 16 processors with labels from 0 to 15 in order, and the second image is compressed with white color. After applying (4) to the compressed image, Figures 4(b) and 4(c) are generated by the skew west and the skew north, respectively. With particular characteristics of parallel processing, all the pixel regions are processed simultaneously. Figure 5 shows some rotated brain images with different angles.

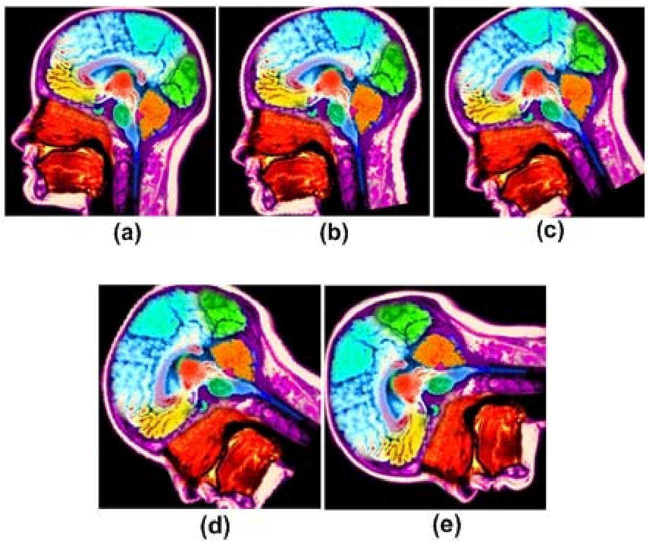


Fig. 5. The original image (a) is rotated with different angles : (b) rotated 10˚, (c) rotated 30˚, (d) rotated 60˚, and (e) rotated 90˚

3.2 Scaling and Translation Algorithms

From (1), scaling and translation are defined as

$$I'(x, y) = I(\sigma x - x_0, \sigma y - y_0) \qquad (6)$$

In practice, both scaling and translation are often used together for transformations. Using these algorithms, we can transfer pixels between PEs to achieve a target picture size from the original picture.

3.2.1 Translation

For a parallel implementation of translation, all the pixels are mapped to every PE as shown in Figure 6(a). Then the transformation equation, I' (x, y) = I (x – x_0, y – y_0) where x_0, y_0 are translated distances, is applied to the original image, producing a translated image, shown in Figure 6(b).

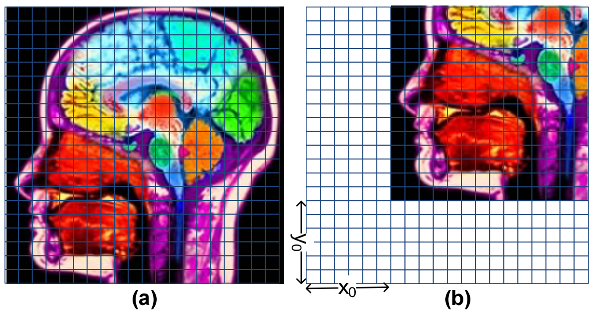


Fig. 6. (a) Original image mapped to PEs, (b) Translated image after x_0, y_0 loops. Note that a square stands for one PE containing 16 pixels.

To perform translation in parallel, the image is equally separated by x axis with the x_0 distance. Then 16 pixels of each PE are transferred to the neighbors with x_0 distance loops. For the y direction, y_0 loops are applied. Since all PEs execute in parallel, the image is translated very fast just by x_0 loops for x axis and y_0 loops for y axis.

3.2.2 Scaling

Image scaling is a frequent operation in medical imaging to enlarge or shrink an image. This section presents a parallel implementation of scaling. An image could be scaled separately in column and row directions. Suppose an image is scaled in column direction. The image is divided into slices in vertical direction. Each column occupies a slice of pixels and then all columns are carried out in parallel relying on PEs which involve those pixels. From (6), we have I'(x, y) = I (σx, y) for the column approach.

Figure 7 shows an example of the image scale down from a resolution 4 by 5 pixels to a resolution 4 by 4. It is equivalent with σ=5/4. We assume that the area of a source pixel is 1.0, therefore, if columns are numbered as {0, 1, 2, …, i, …} and $\{x_{source}\}$={0, 1, 2, …, i, …} for X-axis, $x_{i\ source}$=i. The first target, $x_{0\ target}$ = ½(σ-1). For example, with σ=5/4, we have $x_{0\ target}$ = ½(5/4-1)=1/8 and $x_{i\ target}$ = ½(σ-1) + σ.i as shown in Figure 7.

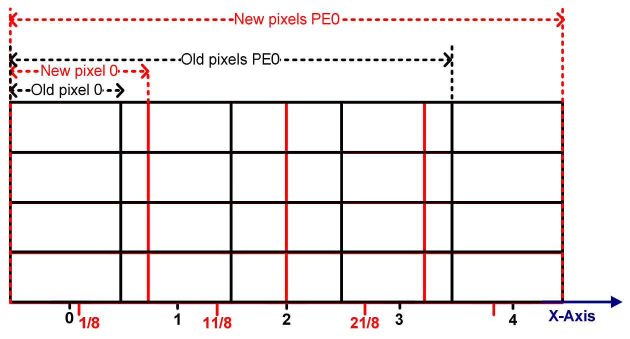


Fig. 7. Image scaling from a resolution of 4×5 pixels to a resolution of 4×4 pixels. Note that (1) one PE contains 4x4 pixels, therefore, it has 4 pixels in X-axis and (2) target PE0 is drawn by red color and source PE0 is drawn by the black with values in X-axis.

Finally, one PE contains two types of x values as shown in (7)

$$\{x_{i\ \text{source}} = i;\ x_{i\ \text{target}} = \tfrac{1}{2}(\sigma - 1) + \sigma . i\} \qquad (7)$$

The remaining task is how to determine pixel values of I' (x, y) of target pixels. Several interpolation algorithms perform this, including nearest neighbor, bilinear, and bicubic interpolations. In this paper, the nearest neighbor interpolation is applied to all pixels based on x-axis as well as related pixel values to calculate the X-axis of all the pixels. Figure 8 illustrates the operation of scale down with σ=5/4. We assume that source pixels require 10 PEs and they are scaled down target pixels which require 8 PEs. According to X-axis, a loop is performed 2 times. For each loop, one PE keeps target pixels and source pixels received from a neighbor. Then the PE relies on $\{x_{i\ source}; x_{i\ target};$ the proposed interpolation} to determine value pixel (i) of the PE.

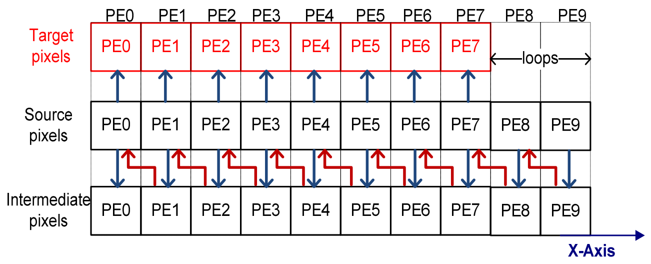


Fig. 8. Every PE consists of three parts: 16 target pixels, 16 source pixels, and 16 intermediate pixels. The number of loops is the difference between the source PE number and the target number. After one loop, pixels are transferred to source pixels of the next west neighbor, while saving source pixels to intermediate pixels for the next transferring.

$$I'(x, y) = F\{x_{i\ \text{source}}; x_{i\ \text{target}}; \text{the proposed interpolation}\} \qquad (8)$$

After only a few loops, the image is scaled. The speed of this algorithm is considerably improved by the operation of PEs in parallel. Figure 9 illustrates a brain image which is scaled up 5/4 for X-axis and then 5/4 for Y-axis using the nearest neighbor interpolation algorithm.

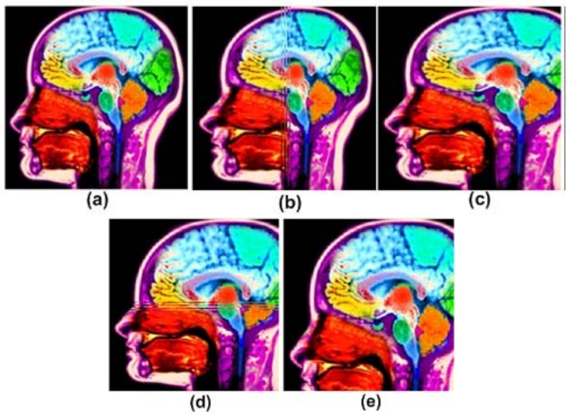


Fig. 9. Scale up 5/4 for X-axis and Y-axis: (a) original image, (b) after some loops for X-axis ,(c) image by scaled up 5/4 X-axis, (d) after some loops for Y-axis, (e) scaled up image.

4 Experimental Results

To implement and determine the performance of the resampling algorithms, we use cycle accurate SIMD simulator [8]. We develop the algorithms in their respective assembly languages for the SIMD processor array. In this study, an image size of 256 × 256 pixels is used. For a fixed 256 × 256 pixel system, the number of 4,096 PEs is used because each PE contains 4×4 pixels. Table 1 summarizes the parameters of the system configuration.

Table 1. Modeled system parameters.

Parameter	Value
Number of PEs	4,096
Pixels/PE	16
Memory/PE [word]	256 [32-bit word]
VLSI Technology	100 nm
Clock Frequency	80 MHz
Interconnection Network	Mesh
intALU/intMUL/Barrel Shifter/intMACC/Comm	1 / 1 / 1 / 1 / 1

We evaluate the performance of the algorithms in terms of execution time and sustained throughput, defined in Table 2.

Table 2. Summary of evaluation metrics.

Execution time	Sustained throughput
$t_{exec} = \frac{C}{f_{ck}}$	$\eta_E = \frac{O_{exec} \cdot U \cdot N_{PE}}{t_{exec}} [\frac{\text{Gops}}{\text{sec}}]$

where C is the cycle count, is the clock frequency, is the number of executed operations, U is the system utilization, and NPE is the number of processing elements.

Table 3 summarizes the execution parameters for each algorithm in the SIMD processor array. Scalar instructions control the processor array. Vector instructions, performed on the processor array, execute the algorithm in parallel. System utilization is calculated as the average number of active processing elements. This table lists the statistics for specific cases such as rotation with α = 300, scale down and up with factor 5/4 and translation with the distance of 30% of the original. As expected, the rotation algorithm takes longer time than others due to its inherent complex skew operations. Overall, the execution times are in the order of milliseconds. Thus, these algorithms are executed at a real-time frame rate (30 frame/sec or 33 ms).

Table 3. Algorithm performance on a 4,096 PE system running at 80 MHz.

Algorithm	Vector Instruction	Scalar Instruction	System Utilization [%]	Total Cycle [cycles]	t_{exec} [ms]	Sustained Throughput [Gops/sec]
Rotation (α = 30˚)	160,193	71,759	31.4	232,022	2.9	71
Scale down (σ = 5/4)	20,259	9,050	60.3	29,326	0.37	135
Scale up (σ = 5/4)	18,967	8,126	63.6	27,124	0.34	145
Translation (x_0,y_0=30%)	6,094	2,766	100	8,862	0.11	227

Table 4. The execution time comparison of parallel approaches and other approaches.

System / Angle	SIMD (clock freq. 80 MHz)	FPGA Using Modified Compensated CORDIC [3] (clock freq. 80 MHz)	Reconfigurable FPGAs [9] (clock freq. 20 MHz)	
			static	dynamic
10˚	1.51 ms	-		24.5 ms
30˚	2.9 ms	-		
45˚	3.99 ms	-	40 ms	
60˚	5.16 ms	-		
150˚	5.7 ms	14.7 ms		

Figure 10 shows the distribution of vector instructions for the resampling algorithms. Each bar divides the instructions into the arithmetic-logic-unit (ALU), memory (MEM), communication (COMM), PE activity control unit (MASK), and image loading (PIXEL). The ALU and MEM instructions are computation cycles while COMM and MASK instructions are necessary for data distribution and synchronization of the SIMD processor array. Results indicate that the resampling algorithms are dominated by ALU and MEM operations.

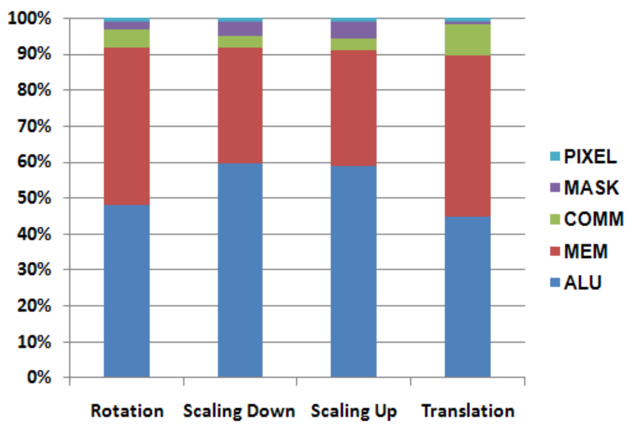


Fig. 10. The distribution of vector instructions for the resampling algorithms.

Table 4 shows the performance comparison of our parallel approaches and other approaches including FPGA implementations. Parallel approaches outperform FPGA implementations for image rotation in terms of execution time. For all cases, a 256×256 pixel is used. Our approaches achieve a speedup of 2.6x over the FPGA implementation using modified compensated CORDIC [3] with the same clock frequency of 80 MHz. These results demonstrate that parallel approaches on the SIMD processor array are suitable candidates for performance-hungry medical imaging.

5 Conclusion

The increasing availability of parallel computers makes parallelizing performance-hungry medical imaging tasks an attractive option. This paper presented parallel implementations of resampling algorithms including rotation, scaling, and translation in medical image registration. Using a representative SIMD processor array, the execution times of these algorithms are in the order of milliseconds, executing at a real-time frame rate (30 frame/sec or 33 ms). In addition, our parallel approaches outperform other FPGA implementations, reducing significant computation time. These results demonstrate that parallel approaches on the SIMD processor array are suitable candidates for emerging medical imaging. In the future, we will explore other medical imaging applications on the SIMD array.

6 Acknowledgement

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST) (No. NRF-2013R1A2A2A05004566)

7 References

[1] J. B. A. Maintz, and A. Viergever, “A survey of medical image of intensities registration” ; Medical Image Analysis, Vol. 2, No. 1, pp 1—36, Mar., 1998.

[2] B. Zitova and J. Flusser, “Image registration methods: A survey”; Image and Vision Computing, Vol. 21, No. 11, pp 977—1000, Oct., 2003.

[3] Xiao-Gang Jiang, Jian-Yang Zhou, Jiang-Hong Shi, Hui-Huang, “FPGA Implementation of Image Rotation Using Modified Compensated CORDIC”; in Proc. of 6th Intl. Conf. on ASIC, Vol. 2, pp. 752—756, Oct., 2005.

[4] A. Gentile and D. S. Wills, “Portable Video Supercomputing”; IEEE Trans. on Computers, Vol. 53, No. 8, pp. 960—973, Aug., 2004.

[5] S. M. Chai, T. Taha, D. S. Wills, J. D. Meindl, “Heterogeneous Architecture Models for Interconnect-Motivated System Design”; IEEE Trans. VLSI Systems, Vol. 8, No. 6, pp. 660—670, Dec., 2000.

[6] S. Nugent, D. S. Willis, J. D. Meindl, “A Hierarchical Block-based Modeling Methodology for SoC in GENESYS”; in 15th Annual IEEE International ASIC/SOC Conference, p. 239—243, Sep., 2002.

[7] S. M. Bhandakar, Y. Huaiyuan, “VLSI implementation of real-time image rotation”; in Proc. of Intl. Conf. on Image Processing, Vol. 2, pp. 1015—1018, Sep., 1996.

[8] H. H. Cat, A. Gentile, J. C. Eble, M. Lee, O. Vendier, Y. Joo, D. S. Wills, M. Brooke, N. M. Jokerst, A. S. Brown, and R. Leavitt, “SIMPil: An OE Integrated SIMD Architecture for Focal Plane Processing Applications”; in Massively Parallel Processing using Optical Interconnection (MPPOI-96), pp. 44—52, Oct., 1996.

[9] E.B. Bourennane, S. Bouchoux, J. Miteran, M. Paindavoine, and S. Bouillant, “Cost comparison of image rotation implementations on static and dynamic reconfigurable FPGAs”; in Proc. of IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing (ICASSP '02), Vol. 3, pp. 3176—3179, May, 2002.

Processing Hard Sphere Collisions on a GPU Using OpenCL

Zachary Langbert[1] **and Mark C. Lewis**[1]
[1]Computer Science, Trinity University, San Antonio, TX, USA

Abstract— *Physically accurate hard sphere collisions are inherently sequential as the order in which collisions occur can have a significant impact on the resulting system. This makes processing hard sphere collisions on parallel hardware challenging. We present an approach to solving this problem that can be implemented using OpenCL that runs on current hardware. This approach makes significant use of atomic operations to prevent race conditions, even across thread groups. We find that an unoptimized implementation of the approach provides speed on modest GPUs that is on par with our earlier OpenMP parallel CPU approach and the OpenCL running on a CPU is faster than the OpenMP code. Full timing results using commodity GPU and using OpenCL on multi-core chips are presented.*

Keywords: Simulation, collisions, parallel, discrete-event, GPU

1. Introduction and Related Work

Many problems in the field of simulation involve collisions between bodies/particles. In this paper we will focus on particles that are represented as spheres. These types of collisions are typically modelled in one of two ways. Soft sphere collisions allow the bodies to overlap and restoring forces are applied at intervals to cause them to bounce. Hard sphere collisions treat the collisions as instantaneous events where an impulse is applied to the particles to produce the bounce.

Collision detection has been done on GPUs for a long time. For example, Kolb et al. used pixel-shaders before tools like OpenCL and CUDA were available and used depth maps stored in texture memory to make the runtime more efficient [4]. Soft sphere collisions can also be implemented efficiently using general N-body methods [1], [2]. The soft-sphere approaches are also applicable to other problems that involve interactions between nearby particles such as SPH codes [12]. The processing of soft-sphere collisions is basically a problem of detecting proximities between bodies, as the exact time of overlap is not resolved or dealt with. Separate GPU implementations exist for solving this more general problem for both simple and complex geometries [5].

Physically accurate hard-sphere collisions are more challenging. They are basically discrete event simulations, and the time ordering of the events is important. Any given collision can alter the ones that follow it. Many collisions can be done in parallel, assuming that they are far enough apart spatially so that the result of one doesn't alter the other. We have previously created methods for doing this on multi-core machines [7], [11], [6]. GPUs present a number of different challenges, and the previous methods will not work well in that context. Not only does efficient use of the GPU require that more threads be active at any given time, workloads on GPUs are best split across multiple thread groups and synchronization across the thread groups is more challenging than inside of the thread groups. For this reason, the use of a single shared queue structure becomes ineffective.

There are also multiple steps to the collision finding process. Some of them involve building data structures that are used for fast searching of the particles for collisions. These steps could be done on the CPU then copied to the GPU, but that would significantly degrade performance. Ideally, we want all the work to be done on the GPU so that the only data that is moved back to the CPU is particle data, and that should only be done when required for I/O.

Playne and Haywick present work on doing hard-sphere collisions using a multi-GPU approach [3], [13]. Their work included both soft-sphere elements with particle-particle forces and hard-sphere interactions when the particles get sufficiently close. However, their approach to hard-sphere interactions use posteriori collisions instead of discrete event priori methods. This means that the particles are allowed to advance to the end of the time step, then they are checked to see if they overlap at the end and corrections are applied to handle the collisions. These methods are not as accurate as using the priori discrete-event approach described here, they can miss collisions if the time step is too long, and they don't do a good job of resolving multiple collisions in a single time step in dense systems. Posteriori methods are easier to process though, especially on GPUs.

Our goal in this work is to deal with the problem of creating a physically accurate, discrete-event, priori algorithm for doing hard sphere collisions that can allow long time steps and run efficiently on a GPU. Section 2 outlines an algorithm for doing this in a basic multithreaded environment. Section 3 describes in detail the algorithm we have developed for doing this on a GPU. This is followed by sections showing the results of a basic implementation of this algorithm and our conclusions.

2. Multicore, Threaded Algorithm

To facilitate the discussion of the GPU approach, it is advantageous to begin by looking at a rough outline of the approach taken in a single-threaded implementation and how that is modified for multiple threads on a multi-core processor. The single-threaded implementation can be described by the following pseudo-code [9].

```
1) Build spatial data structure.
2) Find potential collisions based on initial conditions and
   add them to a priority queue.
3) While there are potential collisions on the priority
   queue.
   a) Remove the next item from the priority queue.
   b) Process that collision.
   c) Remove all future potential collisions involving
      either of the colliding particles from the priority
      queue.
   d) Find new potential collisions involving those
      particles based on their new trajectories. Add
      those happening in the current time step to the
      priority queue.
```

This algorithm uses the term "potential collision" to refer to a triple of two particles and a time, where those two particles would impact at that time given their current trajectories. The word "potential" is significant because many of these won't actually come to pass if an earlier collision alters the path of either of the particles involved. Step 3c can remove many potential collisions when one is processed. Unfortunately, there is no simple way to know if a potential collision will actually be processed as a real collision without running through and processing them to find out. It is worth noting that because of this step, the standard priority queue implementation of a binary heap is not efficient.

This algorithm can be updated to work on multiple threads with a few alterations. The approach to parallelizing the building of the spatial data structure varies by data structure. Some care must be take to avoid race conditions when particles are added in. It is also possible to parallelize the discovery of the initial collisions as long as the priority queue is thread-safe or is locked on each add operation.

The parallelization of the processing loop is more interesting and requires a bit more information about the nature of collisions. Of primary significance is that while the order of collisions is important, information about collisions is propagated at a finite speed. That means that two collisions that are sufficiently far apart can be processed in parallel assuming that they happen close enough together in time. It simplifies things to use the simulation time step as a conservative estimate of the time. The spatial data structure can be used to keep track of where collisions are being processed at any given time to determine if the next one on the queue is safe or not. Here again, the priority queue needs to be thread safe or we must do locking to prevent multiple threads from altering it at the same time.

This approach has been shown to work well for tens of threads, but it doesn't scale well for a GPU where we would ideally like to have thousands of threads. In that situation, the single priority queue becomes a bottleneck that will not scale efficiently.

3. GPU Algorithm

The general outline of the algorithm shown above is maintained when we move to the GPU. First, we need to build our spatial data structure, then we find the initial potential collisions, then we run through and process the collisions. We will look at how each of these steps is adjusted to the GPU here.

3.1 Kernel 1 - Building the Spatial Data Structure

To keep things simpler for this project, we use a regular 2-D grid as the spatial data structure. Each grid cell keeps a list of the particles whose centers are located in it. This can be done with two arrays of integers. One is the size of the grid and stores the index of the first particle in that cell. The second has the same length as the number of particles and stores the next particle found in the grid cell. All elements of both arrays begin with a value of -1 to denote no link.

This process is done with threads spawned on a per particle basis. This opens the possibility of race conditions on the grid values storing the head as every particle in a given cell could, in the worst case, be processed at the same time. Fortunately, the order of particles doesn't matter and OpenCL supports atomic operations on any primitive value [14]. The operation of adding to the front of one of these lists can be done with an atomic read/set. This is done on the value in the grid that stores the first particle currently in that cell. It returns the previous value, which is stored in the second array at the location of the particle that is the new head.

The fact that only one thread will be responsible for a given particle means that we never have a situation where two threads alter the same location in the second array. Every atomic read/set on the grid will find a different value, so the linked list will be correctly built regardless of when the links in the second array are stored.

The grid based approach has been used previously both for sequential and parallel code [9], [10], [7]. The key is that the grid cells need to be large enough that particles in one cell can only collide with particles in adjacent cells during a given time step. That is to say that

$$\Delta x = r_{max} + c \times \Delta t \tag{1}$$

where Δx is the size of the grid cell, r_{max} is the largest particle radius in the simulations, Δt is the length of the time step, and c is a value several times larger than the velocity dispersion. This approach works well when the system includes forces other than those modelled with discrete events. It is also possible to pick a more arbitrary grid size and model the passing of particles from one grid cell to another as events, but that is not the approach we take here.

3.2 Kernel 2 - Finding Initial Potential Collisions

After we have built the spatial data structure that tells us where all the particles are located, we can use it to find the initial potential collisions. This work is done with threads allocated per cell. Instead of having a single queue of all the potential collisions, we keep one queue per cell. The fact that there should be few particles per cell makes it feasible to use something as simple as a sorted array for the queue. This could be changed to a binary heap if the queues were longer, but that introduces some overhead and because of the way that work is distributed, it is probably better to use a finer grid to keep the queues shorter than to improve efficiency for larger queues. This will be an area of future study.

To find the potential collisions, we have each cell check the particles in it against the others in that cell as well as all particles in the cells that are in the cell to the right and the three cells below it. Checking against more cells would cause potential collisions to be double counted. Any pairs that are found to collide are stored, along with the potential collision time, in the queue for that cell.

This can be done with a 3-D array where the third dimension is the potential collisions, but this approach is inflexible and can lead to a lot of wasted memory as we have to make the third dimension large enough to handle whatever collisions might occur in a single cell. An alternative approach is to again use a pool of memory and have the queues stored as linked lists in that pool. We keep a single value for the number of potential collisions currently stored and this variable is altered using an atomic read/increment operation. This tells us the next pool element to store a potential collision in and increments it so that no two threads for the cells will write to the same location in the pool. The potential collisions are first added to an array in local memory, then they are moved as a block to the global memory once the search is complete. This minimizes the number of atomic operations to one call per thread and groups the memory move to reduce the number of accesses to global memory, which is generally much slower than local memory on GPUs. Unfortunately, at this time there is no memory copy directive in OpenCL. Such a directive would have benefits to this segment of the code were it to be added in the future.

Using this approach, the memory overhead is greatly reduced as the total size of the pool needs to scale as the total number of potential collisions, not the maximum number in one cell. This means that non-uniform geometric distributions don't lead to significant wasted memory.

At the end of this process, we will have all of the initial potential collisions for each cell in unsorted lists.

3.3 Inside while Loop

Kernels 3-5 happen in a while loop that executes as long as there are more collisions left to process in the current time step. These steps are broken into separate kernels primarily for synchronization purposes. Each one must fully complete before the next one begins.

3.3.1 Kernel 3 - Sort Initial Potential Collisions

The third kernel is again done by allocating threads by cells in the grid. This pass only sorts the lists that were produced in the previous kernel. We have implemented this as a simple insertion sort. Here again, we expect the number of potential collisions in a given cell to be small. In an ideal configuration the average would be of order unity so the sorts should be doing very little work. To improve the memory performance, we first walk the list of potential collisions in each queue and copy it to a local array. Then we do the insertion sort on that array and copy the potential collisions back into the same locations in the list, preserving the earlier links. This approach is taken to reduce the number of accesses to global memory.

3.3.2 Kernel 4 - Processing Collisions

The fourth kernel does significantly more work. as this is where we actually process the potential collisions. As was discussed above, the primary challenge of hard-sphere collisions is that the order in which they are processed is significant and we can't process all the collisions simultaneously. The spatial data structure gives us a simple way to handle this.

Handle Safe Collisions

In this pass one thread is spawned per cell and each thread compares the time of its first collision to that of the adjacent cells. Only those threads which have a collision time lower than their neighbours will actually process a collision. Cells that have no potential collisions don't process and aren't considered in the comparison to see if a cell is the local minimum.

There is no synchronization needed for this, because the times on the first collisions are only read, not written to, and one particle can only take place in collisions that are in adjacent cells. Given that no two adjacent cells can be processing at the same time, it is safe for each thread to write out and change the particle positions and velocities when a collision is processed.

Mark Indices in Collided Particles Array

In addition to processing the collisions, each thread that does process a collision does one additional task. It stores an appropriate value in an array of the same length as the number of particles that we call the "collided particles array". The purpose of this array is to keep track of which particles were involved in a collision in that pass through the grid. This information is used in the next kernel.

There are no synchronization issues with this task for two reasons. First, given the way that we determine if a collision can be processed, no particle could be involved in two collisions at one time in this pass. What is more, even if two threads could process the same particle, they would write the same value out to the array so it wouldn't matter which write happens first, and no data is read from this array in this kernel.

3.3.3 Kernel 5 - Delete Marked Potential Collisions and Find New Potential Collisions

The last kernel cleans up the potential collisions lists for each grid cell, then finds new collisions based on the particles in each cell that had just been involved in collisions. The first half of this work involves running through the list of potential collisions in each cell and checking each potential collision to see if one of the particles in it was collided during this pass. This is the first use of the collided particles array that was initialized in the previous kernel. In this kernel, that array is read only. The loop runs through the linked list of potential collisions, and if either the first or second particle involved has been marked in the collided particles array, that node is marked as unused and linked around so that it is no longer considered to be a potential collision.

After that has been done, each thread then goes looking for new collisions involving any of the particles in that cell that had been in a collision. This is done by walking the list of particles in the cell, and checking if that particle had been marked in the collided particles array. If it has, then a search is performed against all other particles as well as those in the eight adjacent cells.

The new potential collisions that are found are added to an array in local memory. Once the search is complete, the elements that were used in that local array are moved into the list of potential collisions. This is done using the same type of procedure as in the initial finding routine. This involves a single atomic increment by to appropriate amount, followed by a loop copying up elements into the array of potential collisions.

It is worth nothing that we are not keeping a list of free nodes in the potential collision pool. This will lead to some waste, but at the current time it appears to be required to maintain performance and thread safety. We considered keeping a free list and had included it into a fair bit of the code, but then we ran into a thread safety problem. When we want to grab a new node we have to advance the first free reference to the next element. Because the reference to the first free element is shared across threads, this must be done using an atomic operation. That would lead to a line of code like the following.

```
int oldFF = atomic_xchg(ff,
   potentialCollisionPool[*ff].next);
```

Unfortunately, this isn't really thread safe. The reference to the first free value, `ff`, in the second argument to `atomic_xchg` is a read that isn't protected by the atomic call. That means that two threads could get the same value for `ff` before either one of them does the atomic exchange to update to the next one. The result would be two threads each holding the same node to store a potential collision in which would clearly lead to errors.

It is unclear at this time how critical a problem this is in general. For the tests presented here it was not a problem. Limiting the length of a time step can reduce the number of total collisions so that the potential collision pool doesn't overflow. However, it is likely that we need to find better ways to address this challenge. That will be the subject of further research.

Grid	N	Cells/N	OCL CPU	GPU1	GPU2	OMP CPU
$2k^2$	4m	1:1	630ms	1034ms	1117ms	727ms
$2k^2$	2m	2:1	235ms	530ms	842ms	315ms
$1k^2$	1m	1:1	81ms	223ms	137ms	189ms
$2k^2$	1m	4:1	120ms	375ms	737ms	163ms
$1k^2$	512k	2:1	43ms	126ms	79ms	85ms
$2k^2$	512k	8:1	95ms	318ms	721ms	99ms
$1k^2$	256k	4:1	27ms	80ms	62ms	42ms
$2k^2$	256k	16:1	49ms	267ms	692ms	84ms
$1k^2$	128k	8:1	15ms	64ms	57ms	27ms
$1k^2$	64k	16:1	10ms	64ms	48ms	19ms

Table 1
THIS TABLE SHOWS THE TIMING RESULTS FOR THE OPENCL CODE RUNNING ON BOTH A CPU AND TWO DIFFERENT GPUS COMPARED TO THE EXISTING, MULTITHREADED CODE ON A CPU FOR A VARIETY OF PARTICLE COUNTS AND GRID SIZES. FOR EACH CONFIGURATION, A WARM-UP SIMULATION WAS DONE FIRST, FOLLOWED BY FIVE RUNS THAT WERE AVERAGED TO GET THESE RESULTS.

4. Results

4.1 CPU/GPU Comparisons

The above algorithm was implemented in OpenCL for a 2-D system in which particles move in a straight line between collisions. We first tested the performance of this implementation on a PC with an NVIDIA GeForce GTX 670 graphics card (hereafter GPU1) and an Intel(R) Core(TM) i7-3930K CPU @ 3.20GHz CPU, and separately with a AMD Radeon HD 7970 (hereafter GPU2). The timing results are shown in table 1 and figure 1.

The simulations were set up with a square grid with grid cells that were one unit of length on each side. Particles 0.2 units in radius were placed in the simulation area uniformly with random velocities on the order of one unit. The simulation was run for one time step of 0.1 time units. For the simulations involving 1 million particles this set-up requires that nearly 1 million particle pairs be checked for collisions and over 20,000 potential collisions be added to the queues. Most of those potential collisions turn out to be real collisions that are actually processed.

When looking at the results, one should keep in mind that the OpenCL implementation has not yet been optimized,

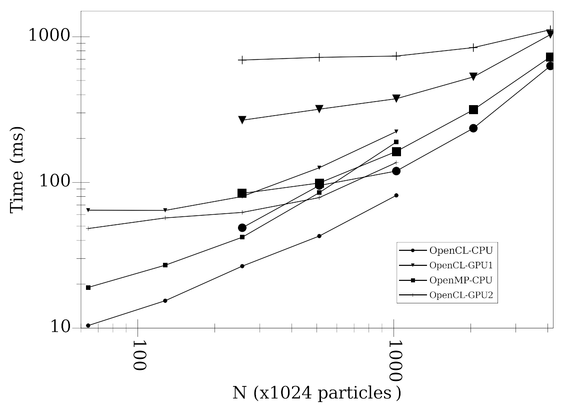


Fig. 1
This figure shows the timing results given in table 1. Smaller symbols are used for the results with the grid that has 1024 cells on the edge and larger symbols are used for the grid with 2048 cells on the edge. The plot makes it clear that the OpenCL implementation of this algorithm is faster on the CPU than our previous OpenMP code. However, the GPU results generally lag behind both codes on the CPU. The one exception is the two largest simulations with the small grid on GPU2, which outperform the OpenMP code.

especially for running on a GPU. On the other hand, the OpenMP implementation is one that we have had for a number of years and have been able to tweak for performance. Despite this, the OpenCL performs better on the CPU and is within a factor of several of the OpenMP code on the GPUs using a commodity gaming graphics card. The gap between OpenCL on the GPUs and the OpenMP code also closes as the particle count increases, with GPU2 beating OpenMP for the two largest particle counts using the smaller grid. For the largest simulations at each grid size, the difference is well below a factor of two on GPU1.

The table lists the ratio of number of grid cells to number of particles because empirical work on the original version of this code [9] found that for a fixed number of particles, the code gave optimal performance with a maximum ratio of 7:1. We have no yet performed a full set of tests to determine the optimal ratio for the OpenCL code, but these tests do show that this algorithm appears to have a greater cost for a larger number of grid cells, particularly on on the AMD GPU, GPU2. The OpenCL code is faster for using a smaller number of grid cells than a larger one for all particle counts. This isn't too surprising given how many of the passes launch threads based on the number of grid cells. As a result, it appears that the OpenCL code likely has an optimal ratio with fewer grid cells per particle. Resolving the ideal ratio is an area for future work.

The performance of the GPUs in these tests is consistent with the fact that the code has not yet been optimized for the GPU. Each thread group on a GPU is basically a SIMD unit that is most efficient when all the threads are doing the same thing, running a single instruction on multiple pieces of data. We have not yet optimized the code to try to keep threads doing the same thing. The kernels were written to contain as much work as was possible for a particular decomposition of the work without running into race conditions. The code likely needs to be refactored in a number of ways, including breaking up the kernels, to keep the work more consistent across the threads so that the GPUs can run more efficiently.

4.2 CPU Scaling

A second set of tests was run using only the CPU to look at how well this new algorithm, using OpenCL, scales relative to the old one using OpenMP. The results of these tests are shown in figure 2. These simulations were run on a Dell server with four 16-core Opteron 6272 processors, each with 16 GB of local RAM. So the machine has 64-cores and 64 GB or RAM total, allowing us to scale the simulations be to significantly larger than the earlier tests. The initial conditions were the same as before. For these simulations, the ratio of grid cells to particles was kept fixed at 1:1 for the OpenCL simulations and 4:1 for the OpenMP code. The number of cores used in the simulations was set at 16, 32, and 64 for each of the codes for a variety of particle counts. The timing results are shown in figure 2.

Both code scale remarkably linearly with particle count up to more than 67 million particles. Consistent with earlier results, the OpenCL code holds a reasonable speed advantage up to 4 million particles. Most of the advantage disappears at 16 million particles and there is remarkably little spread in runtime between the methods or the core counts in the largest simulations. More work needs to be done to determine why the OpenCL implementation loses ground for the largest simulations and what can be done, either in the code or the configuration, to prevent this.

4.3 Double Precision

These timing results came from code using single precision floating point values and arithmetic. Current GPUs tend to be much faster with single precision than double precision in most benchmarks. Unfortunately, there are some simulations that explore problems of scientific interest that require the additional precision. Hard sphere collisions for large N happen to be one of those problems. For that reason, we wanted to explore the impact of using double precision numbers here because this happens to be just such a situation.

The importance of double precision for these simulations was discovered when we put in a consistency check on the

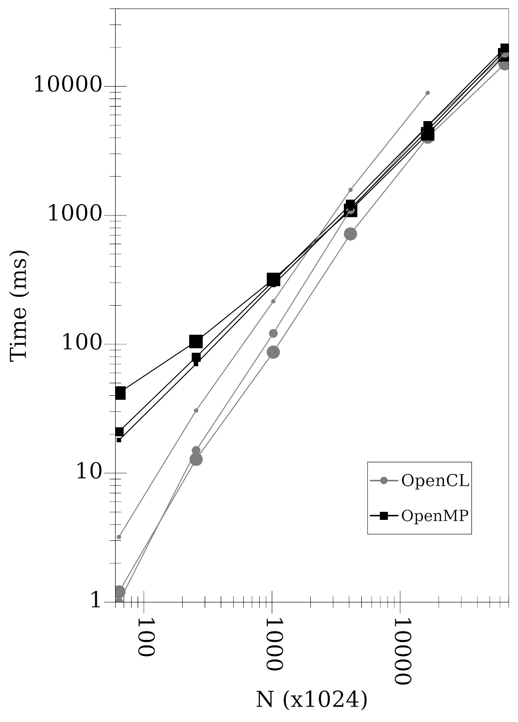


Fig. 2
This figure shows the timing results on a 64-core machine comparing the OpenCL implementation of this algorithm with our earlier OpenMP code. For these simulations, the ratio of grid cells to particles was fixed at 1:1 for the OpenCL code and 4:1 for the OpenMP code. The symbol size indicates the number of cores involved. Small symbols are 16 cores, medium are 32, and large symbols are 64.

hard-sphere collisions. This check tested to see that when particles were advanced to the time of a potential collision that they were actually touching. This check failed when we required the separation between particles be within 1% of the sum of the particle radii. After double checking all the math and manually testing some of the values for the root finding, we realized that the problem was actually the accuracy of single precision floating point values. The simulation region was 1000 units across and the particles were 0.2 units in radius. That means that we were testing for an accuracy on the order of 10^{-5}. This is below the expected accuracy of single precision arithmetic, but would be met easily by double precision values.

Running a few tests showed us the very surprising result that changing the code to use the `double` type instead of `float` didn't have a significant impact on speed for our hardware. Our first assumption was that the cards, being commodity graphics cards and not cards specifically designed for HPC and GPGPU, were defaulting to single precision despite being told to use double precision. Attempts to measure the value of ϵ on the graphics cards showed that they were doing something different with `double` than with `float`, but the results those tests produced were not consistent with IEEE double precision numbers so more work is needed to determine what is happening in this area.

5. Conclusions

We found that a hard sphere collision algorithm using separate queues for each of many small spatial regions was implementable in OpenCL and that even without significant tuning, this code could outperform an earlier OpenMP implementation using a single queue when running on the CPU. Unfortunately, the GPU performance of the implementation currently lags behind that on the CPU.

There are many open questions and areas left for future work. In addition to those mentioned earlier in the paper, there are a few other areas that remain to be explored. First, does the difference in the ideal grid cell size alter how the performance varies with length of time step. The previous methods tend to scale the grid with the length of the time step. Because a simulation needs to go for a certain period of time, and the number of particles searched for collisions scales as the area of a grid cell, the total run time of a simulation tended to scale as $1/dt + dt^2$. This leads to an optimal time step that is fairly short to keep the cell sizes small. This work has already shown that this new algorithm performs better with larger grid cells. This could allow longer time steps to be taken, providing shorter total run times, even if the performance for a single time step is inferior.

This work looked only at the most basic of particle configurations where only collisions are considered. In realistic systems, the particle distribution is typically not so uniform and other forces, like gravity, can cause particles to clump. This alters the geometric distribution of the collisions. It is unclear at this time what impact that would have on this method.

Lastly, when the geometric distribution becomes significantly non-uniform, the spatial grid that was used here becomes ineffective. We have extended the older approach to use spatial trees as a more dynamic searching and locking data structure [8]. This same thing could be done on the GPU, though it is likely to be significantly more challenging to do so.

References

[1] Erich Elsen, Vaidyanathan Vishal, Mike Houston, Vijay S. Pande, Pat Hanrahan, and Eric Darve. N-body simulations on gpus. *CoRR*, abs/0706.3060, 2007.

[2] Simon Green. Particle simulation using cuda. *NVIDIA Whitepaper, December 2010*, 2010.

[3] K. A. Hawick and D. P. Playne. Hard-sphere collision simulations with multiple gpus, pcie extension buses and gpu-gpu communications. In *Proceedings of the Tenth Australasian Symposium on Parallel and Distributed Computing - Volume 127*, AusPDC '12, pages 13–22, Darlinghurst, Australia, Australia, 2012. Australian Computer Society, Inc.

[4] A. Kolb, L. Latta, and C. Rezk-Salama. Hardware-based simulation and collision detection for large particle systems. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, HWWS '04, pages 123–131, New York, NY, USA, 2004. ACM.

[5] C. Lauterbach, Q. Mo, and D. Manocha. gproximity: Hierarchical gpu-based operations for collision and distance queries. *Computer Graphics Forum*, 29(2):419–428, 2010.

[6] Mark Lewis, Matthew Maly, and Berna L Massingill. Hybrid parallelization of n-body simulations involving collisions and self-gravity. In *Proc. Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTAâĂŹ09)*, pages 324–330, Las Vegas, USA, July 2009. CSREA.

[7] Mark Lewis and Berna L Massingill. Multithreaded collision detection in java. In *Proc. Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTAâĂŹ12)*, pages 583–592, Las Vegas, USA, July 2006. CSREA.

[8] Mark Lewis and Cameron Swords. Lock-graph: A tree-based locking method for parallel collision handling with diverse particle populations. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications PDPTA'11, Volume I*, pages 157–161. CSREA Press, 2011.

[9] Mark C Lewis and Glen R Stewart. A new methodology for granular flow simulations of planetary rings-collision handling. In *Modelling and Simulation*, pages 292–297, 2003.

[10] Mark C Lewis and Nick Wing. A distributed methodology for hard sphere collisional simulations. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications-Volume 1*, pages 404–409. CSREA Press, 2002.

[11] Berna L Massingill and Mark Lewis. Parallelizing a collisional simulation framework with plpp (pattern language for parallel programming). In *Proc. Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTAâĂŹ06)*, pages 608–614, Las Vegas, USA, July 2006. CSREA.

[12] Hammad Mazhar, Toby Heyn, and Dan Negrut. A scalable parallel method for large collision detection problems. *Multibody System Dynamics*, 26(1):37–55, 2011.

[13] D. P. Playne and K. A. Hawick. Classical mechanical hard-core particles simulated in a rigid enclosure using multi-gpu systems. In *Proc. Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTAâĂŹ12)*, pages 76–82, Las Vegas, USA, 16-19 July 2012. CSREA.

[14] John E. Stone, David Gohara, and Guochun Shi. Opencl: A parallel programming standard for heterogeneous computing systems. *IEEE Des. Test*, 12(3):66–73, May 2010.

Scalable Self-Tuning Implementation of Smith-Waterman Algorithm for Multicore CPUs

Faisal Sikder and Dilip Sarkar

***Keywords*-CPU Oblivious, Scalable Algorithm, Multicore CPU, OpenMP, Parallel algorithm, Shared memory architecture, Smith-Waterman algorithm.**

***Abstract*—Improved version of the Smith-Waterman algorithm (SWA) is most widely used for local alignment of a pattern (or query) sequence with a Database (DB) sequence. This dynamic-programming algorithm is computation intensive. To reduce time for computing alignment score matrix, parallel versions have been implemented on GPUs and multicore CPUs. These parallel versions have shown significant speedup when compared with their corresponding sequential versions.**

Our initial evaluation of an OpenMP parallelization of SWA has shown linear speedup on multicore CPUs, but a closer look at performance data from both sequential and parallel versions have revealed two undesired effects: (i) As the length of the DB sequence increases, the number of elements of the alignment score matrix H computed in per unit time initially increases, then reaches a maximum, and finally decreases continuously; (ii) the length of the DB sequence where decline starts is different for different CPUs. To overcome the computation rate decline we have proposed a run-time self-tuning algorithm. It determines the length, l, of a DB sequence that maximize computation rate during execution time. Then, divide computation of H into computation of a set of submatrices, such that the number of columns in each submatrix is about l.

Our study also found that the number of per-core-threads that delivers the highest rate of computation differs from CPU to CPU. Our proposed algorithm determines optimal number of threads during execution time and creates optimal number of threads for highest possible computation rate. Our extensive evaluations of the proposed self-tuning algorithm on three different multicore multi-CPU shared memory machines have shown significant performance improvement.

I. Introduction

Now most, if not all, computers are powered with one or more multicore CPUs. Since each of these cores can execute independent task, they collectively may increase computing power of a CPU. However, increase in computing power depends not only on the number of cores available, but it also depends on organization of the cores and the cache in the CPU, shared-memory access mechanism, and performance of translation lookahead buffer (TLB) used for virtual address translation.

While utilization of these cores for multiple *independent tasks* is straight forward, *efficient utilization* of these cores to solve *one large problem* requires a parallel algorithm that creates multiple tasks for concurrent execution on these cores. Moreover, for computation-intensive algorithms as the problem size grows memory access time may increase and instruction execution rate may reduce and performance may fail to scale up. Thus, implementation of parallel algorithms on a multicore CPU brings a new challenge for computation-intensive problems. Especially, for algorithms that use arrays of dimension two or higher as data structure, such as, matrix operations and dynamic programming.

Furthermore, as more cores are added to a single CPU, the designers alter cache organization, cache size, and interfaces between cache and cores. Also, memory access methodology changes. Thus, an efficient parallel algorithm implemented to solve a problem on one multicore CPU may not be efficient on another CPU, or future multicore CPUs. Therefore, *it is important to develop and implement parallel algorithms that are efficient and easily portable from one multicore CPU to other multicore CPUs while they maintain high efficiency, and will continue to be portable and efficient for evolving future generations of multicore CPUs.*

Another performance issue is instruction execution rate, which is affected by memory access rate and could be problem size dependent. This is specially true for algorithms which use arrays of dimension two or higher (see [12]). To overcome these issues cache-oblivious and resource-oblivious algorithms have been proposed (see [5], [6], [7], [12], and [23], and references therein). These algorithms recursively divide problem to be solved until the subproblems are small enough for holding in the cache. They are asymptotically optimal for sequential algorithms, but they require parallelization or need support from special software systems — both of which are nontrivial tasks. To the best of our knowledge no scalable self-tuning parallel algorithm exists for Smith-Waterman algorithm [21].

In this paper we propose, implement, and evaluate a parallel version of Smith-Waterman algorithm (SWA) [21] that is scalable as well as self-tuning. The algorithm does a two-step self-tuning: first it executes a sequential algorithm to find an optimal size for the problem, and then it finds an optimal number of threads for each core for the highest speedup. Finally, it divides the problem to be solved into smaller subproblems and *execute each of them in parallel* using optimal number of threads.

The rest of the paper is organized as follows. We briefly review work related to our work in Section II. In Section III, we present sequential Smith-Waterman algorithm and a naive parallel implementation of it using OpenMP. In Section IV, we present our self-tuning scalable parallel algorithm for Smith-Waterman algorithm. Some typical results from our evaluation of the proposed algorithm are presented in Section V. Section VI discusses our findings and potential future work.

Faisal Sikder and Dilip Sarkar are with the Department of Computer Science, University of Miami. E-Mails: f.sikder@umiami.edu and sarkar@miami.edu; May 28, 2014

II. Related Work

For space limitation we include only a few references. A more comprehensive list of references can be found in our report [20].

Sequence alignment is one of the most fundamental task in bioinformatics. Because of the size of the databases, sequence alignment is very time consuming. To reduce alignment time heuristic-based pairwise sequence alignment algorithms and software tools have been developed. These tools can be grouped into two general categories: *hash-table* based algorithms and *suffix-tree* based algorithms.

Hash table based software tools include BLAST [3] and FASTA [18]. As the number of data-sets in databases grows everyday, so does the computation time for finding all alignments of a query sequence. To curb the growth of computation time, mpiBLAST [8] and RPAlign [4] have exported BLAST and FASTA to distributed computing clusters. Because of higher memory and time complexity, suffix-tree based tools are fewer than hash-table based software tools. A prominent suffix-tree based software is MUMmer [10].

These heuristic-based tools work very well for finding locations of exact or near-exact alignments of a query sequence. However, if some symbols of a query sequence differs in few locations because of sequencing error, or some symbols were added or deleted because of sequencing errors they fail to identify locations of optimal alignments as can be done with the Smith-Waterman algorithm [21].

The Smith-Waterman [21] algorithm and its improved version [13] (which is also known as Smith-Waterman algorithm) have much better sensitivity and specificity, but require quadratic computation time and space. Thus, sequential version of the algorithm is impractical for long query and/or database sequences. To extend the scope of usefulness, parallel versions of Smith-Waterman algorithm have been implemented on various parallel computing platforms, including multicore CPUs [1], [22] and SIMD instruction set [11], [19], GPUs [9], [15], CPU and GPU based hybrid systems [16], CPU and FPGA based hybrid systems [17].

One of the parallel implementation of Smith-Waterman algorithm using GPU is CUDASW++ [15]. This CUDA based parallel implementation is claimed to have demonstrated significant speedup over CPU implementation. Khajeh-Saeed et al. implemented Smith-Waterman algorithm for single and multiple GPU systems [14]. They reported 45 times speedup over CPU version. Also, for 4-GPU systems they reported a speedup of about four over single GPU system.

Rongers [19] implemented SIMD based version of Smith-Waterman algorithm and claimed six times speedup. Farrar also used SIMD based implementations and received 2-8 fold speedup over others [11].

High Performance Genomics project recently implemented a parallel version of the Smith-Waterman algorithm using OpenMP, which is know as HPG-SW [1]. The algorithm has neither been published yet nor any documentation is available. However, the source code is available from the Internet [1]. In this algorithm, the number of threads to be used during run-time is hard-coded and requires to be changed manually.

Any GPU implementation is highly platform dependent and has restricted memory access mechanism. On the other hand, even low cost CPUs from different manufacturers have 4 to 8 cores. Thus, an efficient, scalable, and self-tuning parallel implementation of Smith-Waterman algorithm for all multicore CPU-types is very important.

III. Sequential and a Naive Parallel Implemntation of Smith-Waterman Algorithm

For the ease of presentation and completeness, first a sequential version of Smith-Waterman algorithm and then an OpenMP-based parallelization of it are presented. A brief evaluation of these two versions are also presented in this section to motivate our readers.

A. Sequential Version of Smith-Waterman Algorithm

Let $DB =< d_1, d_2, ..., d_m >$ be a database sequence of length m and $PT =< p_1, p_2, ..., p_n >$ be a pattern sequence of length n. For local alignment of a PT with a DB the Smith-Waterman algorithm [13], [21] is the most widely used algorithm. This computation intensive dynamic-programming algorithm runs in two phases: in *phase 1* it creates an alignment score matrix $H_{m+1 \times n+1}$, and in *phase 2* it obtains the optimal alignment.

Phase 1*-Compute the alignment score matrix:* The elements of first row and first column of H are initialized to 0, that is, for $i = 0$ and $0 \leq j \leq m$, $H_{0,j} = 0$, and for $1 \leq i \leq n$ and $j = 0$, $H_{i,0} = 0$. Other alignment scores for elements of H are computed using the equation 1; a reward is added to, or a penalty value is subtracted from $H_{i-1.j-1}$ for obtaining $H_{i,j}$. If $p_i = d_j$, a match occurs and the reward value is w_r. If $p_i \neq q_j$, a mismatch occurs and the penalty value is w_{pm}. If a gap is inserted in PT, the penalty value is w_{pg}. If an element is deleted from DB, the penalty value is w_{pd}.

$$H_{i,j} = \max \begin{cases} 0 & \\ H_{i-1,j-1} + w_r & \text{for a match} \\ H_{i-1,j-1} - w_{pm} & \text{for a mismatch} \\ H_{i-1,j} - w_{pd} & \text{for a deletion} \\ H_{i,j-1} - w_{pd} & \text{for an insertion} \end{cases} \quad (1)$$

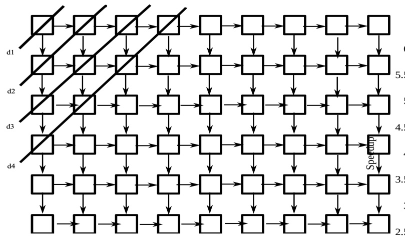
Phase 2*-Obtain the Optimal Alignment:* In this phase, starting from a highest score cell in H, an optimally aligned sequence is obtained. A move to the left cell inserts a gap in PT. An upward move represents a deletion of a symbol from DB. Finally a diagonally upward move indicates a matched or a mismatched symbol between PT and DB.

An outline for an initialization procedure is shown in **Algorithm 1**. The input to the initialization procedure are: m – the length of database sequence, n – the length of pattern sequence. The output from the procedure is initialized alignment score matrix H. A procedure for phase 1 of the Smith-Waterman algorithm is shown in **Algorithm 2**.

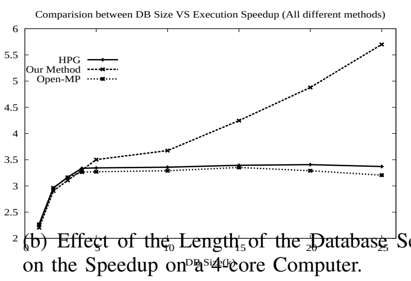
An OpenMP version is presented next.

B. Parallel Version of the Smith-Waterman Algorithm

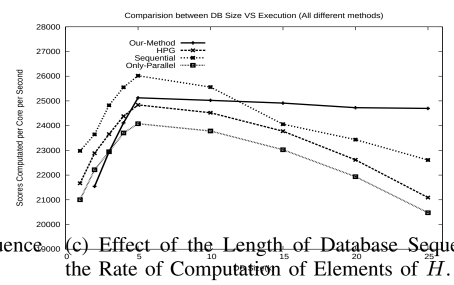
After discussing data-dependency for computing H, a naive parallel version of the algorithm is presented in this section.



(a) Data dependency Graph for Computing Alignment Score Matrix H.



(b) Effect of the Length of the Database Sequence on the Speedup on a 4-core Computer.



(c) Effect of the Length of Database Sequence on the Rate of Computation of Elements of H.

Fig. 1. Data Dependency for computing alignment score matrix, Effect of the DB length on Speedup, and Effect of Number Threads for each Core.

Algorithm 1 Initialization of Sequential Version of the SWA

```
1: procedure SWInitialization($m$, $n$, $H$)
2:     // Set elements of the 0th row of $H$ to zero
3:     for ($i = 0; i < n; i$++) do $H(i, 0) = 0$;
4:     // Set elements of the 0th column of $H$ to zero
5:     for ($j = 0; j < m; j$++) do $H(0, j) = 0$;
6: end procedure
```

Algorithm 2 Sequential Version of the SWA

```
1: procedure SWSequential($m, n, H$)
2:     SWInitialization($m, n, H$)
3:     // Using Equation 1 compute elements of $H$
4:      for ($i = 1; i < n; i$++) do
5:         for ($j = 1; j < m; j$++) do Compute $H(i, j)$;
6: end procedure
```

Data Dependency for Computing H: Data dependency graph for computing alignment score matrix H is shown in Fig. 1(a). We can see from the data-dependency graph that alignment score matrix H can be computed in three different orders: (i) Row-wise from the top-most row to the bottom-most row in that order, and in each row elements are computed from the left-most column to the right-most column. (ii) Column-wise from the left-most column to the right-most column, and in each column elements are computed from the top-most row to the bottom-most row. (iii) Diagonally, from the top-left corner to the bottom-right corner as indicated by diagonal lines.

While row-wise and column-wise computation impose their restrict sequential orders, diagonal-wise computation eliminates this strict sequential order. Computation of all the elements in a diagonal can be done in any order or in parallel. However, computation of the elements of a diagonal requires that computation of all elements in the diagonal just above it have been completed.

Parallelization of Computation of H: Without loss of generality, let us assume that a multicore CPU has C cores, and for a nonzero integer k, $T(= k.C)$ rows of H can be computed in parallel by maintaining a diagonal order on these T rows. Let us assume that computation of r rows of H has been completed, and rows (r+1) to $(r + T)$ to be computed.

An outline of our naive OpenMP version is shown in **Algorithm 3**. This procedure sets the number of threads to

Algorithm 3 A Simple Parallelization of SWA Using OpenMP

```
1: procedure SWOpenMP($m$, $n$, $H$, $T$)
2:     SWInitialization($m, n, H$);
3:     omp_set_num_threads($T$);
4:     $TPos[T] = 0$;
5:     // $TPos[t]$ stores the col # thread $t$ has computed last
6:     $omp\_set\_num\_threads(T)$
7:     // Create a parallel section of $T$ threads
8:     #pragma omp parallel
9:         for ($k = 0; k < \lceil n/T \rceil; k$++) do
10:            // The thread $t, 0 \leq t < T$, computes
11:            //    rows $1, t + 1, 2t + 1, \cdots$ of $H$
12:            $i = k * T + t + 1$;
13:            for ($j = 1; j < m, j$++) do
14:                while $(TPos[t - 1] < j)$ $wait()$;
15:                $Compute\ H(i, j)$;
16:                $TPos[t] = j$;
17:     end parallel section
18: end procedure
```

T, an integer multiple of C. Then it creates a `parallel section`, where all T threads compute the values of H in parallel. Since there are n rows, each thread computes approximately $\lceil n/T \rceil$ rows. If identification number of a thread is t, it computes rows $(t + 1)$, $2 * T + t + 1$ and so on. Because of an strict order for computation of elements of H, the element $H(i - 1, j)$ must be computed before the element $H(i, j)$ can be computed. To ensure this restriction, each thread waits in a while loop until the element just above it has been computed. An array $TPos$ is used for lock-free implementation of the restriction.

C. Perfomance Evaluation of the Naive Version

We implemented the procedure shown in **Algorithm 3** using OpenMP 3.0. To evaluate performance of our parallel version, we obtained the best known parallel implementation [1] (in the rest of the paper this implementation is refereed to as HPG-SW) and measured speedup on three computer systems (see Section V-A). We observed that for a PT sequence of fixed length as the length of the DB sequence is increased, the speedups steadily increase to a maximum (see Fig. 1(b)).

Effect of DB Length on Computation Rate: The most interesting observation is illustrated in Fig. 1(c). As the length

of the DB sequence increases all implementations (except implementation proposed later) share a similar trend: the computation rate for each implementation i) initially increases, ii) then reaches a maximum, and iii) finally starts to decline.
Our first hypothesis: *For a given length of PT sequence, there is an optimal length for the DB sequence.*

Our proposed method for implementation of self-tuning parallel algorithms is presented next.

IV. Proposed Scalable Self-Tuning Parallel Implementation of SWA for Multicore CPUs

In this section, we first describe a procedure to identify an optimal length for the DB sequence. Then we propose a procedure to determine optimal number of threads for each core. Finally, we use this procedures in our implementation of a self-tuning parallel Smith-Waterman Algorithm.

A. Procedure for Identification of an Optimal Size DB

Identification of an optimal length of the DB sequence starts with a given PT of relatively small size, say 200 symbols. The initial length of the DB sequence is same as that of the PT sequence, and the length of the DB sequence is increased by a constant $IncC$ until an optimal length is found. We used $IncC = 25$ for our experiments. For each DB sequence average time to compute one element of H is computed and compared with previously computed minimum average time. If most recently computed time is smaller than the currently known minimum, the minimum time is updated and corresponding length of the DB sequence is recoded. For determination of termination point, one can use a rule of $DblPc\%$: If most recently computed average time is $DblPc\%$ higher than the currently known minimum, the algorithm stops. We used 15% rule. An outline of the algorithm is shown in **Algorithm 4**.

Algorithm 4 Identification of Optimal Length DB Sequence

```
1: procedure FindOptimalLengthDB($OpS$)
2:     $TBound = -100; n = 200; m = 200;$
3:     $BestTime = 200$; //A very large value
4:     while $TBound < DblPc$ do // use $DblPc\%$ rule
5:         $start = time();$
6:         SWSequential($m, n, H$);
7:         $end = time();;$
8:         $CTime = (n * m/(end - start));$
9:         $TBound = \frac{CTime - BestTime}{BestTime} \times 100;$
10:        if ( $BestTime > CTime$) then
11:            $BestTime = CTime; OpS = m;$
12:        $m = m + IncC$; // $IncC$ is increment constant
13: end procedure
```

Results obtained from running this algorithm on three shared-memory computers are illustrated in Fig. 2(a) (see Section V-A). An average computation time for one element of H was obtained by dividing total time for computing H by the number of elements in H. From the plots we observe that as the length of the DB size grows, the computation time for each computer goes through three distinct phases:(i) initially average computation time remains almost constant, (ii) then it decreases to a minimum, (iii) finally it starts to increase.

Therefore, empirical evidence support our hypothesis that there is an optimal length for the DB sequence. Thus, for achieving optimal computation rate the DB sequence must be partitioned and computation for each partition has to be performed at a time.

Next we focus on identification of optimal number of threads for each core.

B. Procedure for Idetification of Optimal Number of Threads

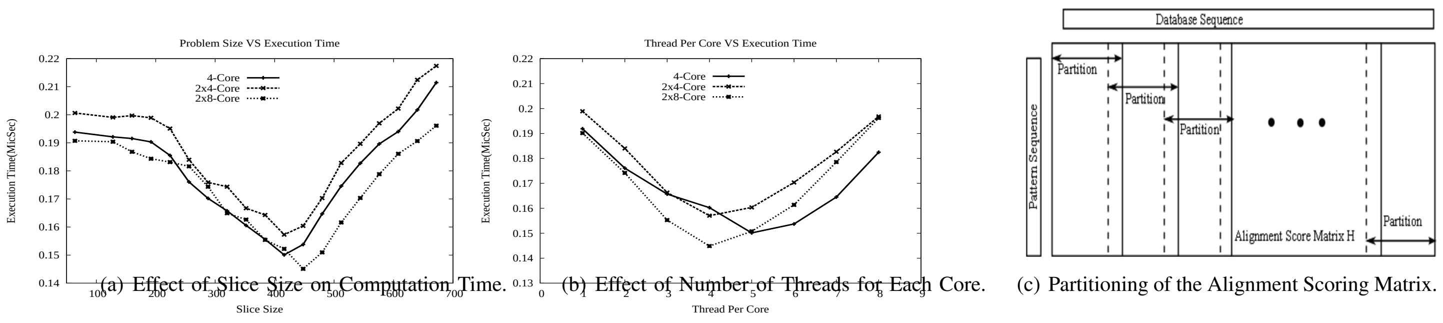
It is obvious that the number of threads should be no less than the number of cores. But can more threads per core help? If so, the next question is how many threads for each core? Also, is the thread to core ratio CPU/machine independent?

To answer these questions, we executed our OpenMP implementation of **Algorithm 3** on three different shared-shared memory computers (see Section V-A). Some results from this experiment are illustrated in Fig. 2(b). Three interesting observations can be made from the plots: (i) more than one thread per core is beneficial, (ii) there is an optimal number of threads for each core, and (iii) the optimal number varies from system to system. For the single CPU system the optimal ratio is five, while for the 2-CPU systems the optimal ratio is four.
Our second hypothesis: *For a given multicore computer, there is an optimal number of threads for each core.*

Algorithm 5 Identification of Optimal Number of Threads

```
1: procedure OptimalThreads($OnTs$)
2:     $n = 200; m = 200$; $TBound = -100;$
3:     // $NtPc$ is # of threads for each core, $C$ is # of cores
4:     $NtPc = 1$; // initially 1 thread per core
5:     $BestTime = 200$; //a large initial value
6:     //number of cores: C
7:     while ($TBound < ThnPc$) do // use $ThnPc\%$ rule
8:         $start = time();$
9:         SWOpenMP($m$, $n$, $H$, $C * NtPc$);
10:        $end = time();$
11:        $CTime = (C * tr * n * m/(end - start));$
12:        $TBound = \frac{CTime - BestTime}{BestTime} \times 100;$
13:        if ($CTime < BestTime$) do
14:            $BestTime = CTime$; $OnTs = C * NtPc$;
15:        $NtPc$++
16: end procedure
```

Identification of Optimal Number of Threads: The proposed steps for determining optimal number of threads for each core are similar to that for determining optimal length for DB sequence. We start with one thread per core, and increase it by one at every iteration, until we find the number that minimize computation time using our rule of $ThnPc\%$. The procedure is shown in **Algorithm 5**. Inside the while loop of this algorithm we call **Algorithm 3** with PT and DB whose lengths have been determined by calling procedure in **Algorithm 4**, or something similar. Unless the lengths are too small, it does not matter, since we are determining optimal number of threads.



(a) Effect of Slice Size on Computation Time. (b) Effect of Number of Threads for Each Core. (c) Partitioning of the Alignment Scoring Matrix.

Fig. 2. Evidence of Optimal DB Length, Optimal Number of Threads for each Core, and Optimal Partitioning of the Alignment Scoring Matrix.

C. Procedures for Initialization of Submatrices

Let OpS be the length of a DB sequence for which computation rate is highest. The proposed algorithm first executes procedure presented in **Algorithm 4** to determine OpS. Then computation of alignment score matrix H is divided into computation of $\lceil m/OpS \rceil$ smaller alignment score submatrices obtaied by dividindg H vertically, as shown in Fig. 2(c). Notice that the first row of each partition, except for first partition, is the last row of the partition just before it. The computation of each submatrix is done in parallel. *For keeping all elements of a partition in contiguous locations in the memory, we use a 3-D array. The first dimension of the array represents the partition number of H.*

Initailaiztion of Partitions of H: The initialization procedure assigns zeros to all elements of the 0th row all partitions. It also assigns zeros to all elements of the 0th column of the first partition. An outline of the procedure is shown in **Algorithm 6**. The elements of 0th column of other partitions are initialized by copying last column of the previous partition at the beginning of computation of the partition.

Algorithm 6 Initialization of the Partitions of H

```
1: procedure INTIPARTITIONEDH($m$, $n$, $H$, $OpS$)
2:     $partitoins = \lceil m/OpS \rceil$
3:     for ($k = 0; k < partitions; k$++) do
4:         // Set elements of the 0th row of $H$ to zero
5:         for ($j = 0; j < OpS; j$++) do $H(k, 0, j) = 0$;
6:     // Set elements of the 0th column of $H$ to zero
7:     for ($i = 0; i < n; i$++) do $H(0, i, 0) = 0$;
8: end procedure
```

Next we present a procedure that is executed by each thread for computing its share of the alignment score matrix H.

D. Procedure for Each Thread

The procedure shown in **Algorithm 7** is executed by all threads created in **Algorithm 8**. Each thread first computes $numRows$, the number of rows the thread has to compute. Next it enters into a loop to compute these rows of H. It repeats the process for all the partitions. Recall that there is an strict order for computation of elements of H. The element $H(k, i-1, j)$ must be computed before the element $H(k, i, j)$ can be computed. To ensure this restriction we use an array $TPos$ as shown in the procedure.

Algorithm 7 Task for Each Thread

```
1: procedure THREADTASK($n$,$H$,$T$,$partitions$,$TPos$)
2:     for ($k = 0; k < partitions; k$++) do
3:         $numRows = \lceil n/T \rceil$; // rows per thread
4:         // $s$ keeps the count of rows computed by the thread
5:         for ($s = 0; s < numRows; s$++) do
6:             // compute next row $i$ for thread $t$
7:             $i = s * T + t + 1$;
8:             //set the starting position of the DB slice
9:             for ($j = 1; j < OpS; j$++) do
10:                while ($TPos[t-1] < j$) do $wait()$;
11:                $Compute\ H(k, i, j)$; $TPos[t] = j$;
12: end procedure
```

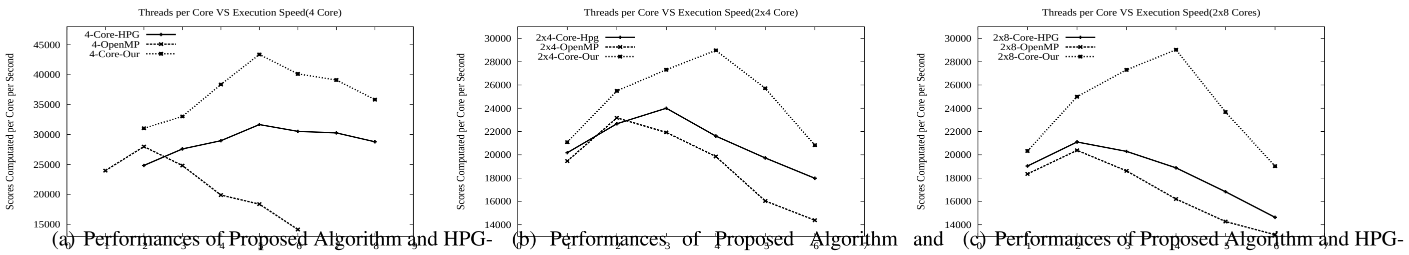
Algorithm 8 Self-Tuning Parallel SWA for Multicore CPUs

```
1: procedure SELFTUNINGOPTIMALTHREADS($m$, $n$, $H$;)
2:     FINDOPTIMALDB($OpS$)
3:     INTIPARTITIONEDH($m$, $n$, $H$, $OpS$);
4:     $partitions = \lceil m/OpS \rceil$;
5:     OPTIMALTHREADS($OnTs$);
6:     //$TPos$ Shared array to hold current position
7:     $TPos[OnTs] = 0$
8:     //$t$ set number of threads for OpenMP derivatives
9:     $omp\_set\_num\_threads(OnTs)$
10:    #pragma omp parallel
11:        THREADTASK($n$,$H$, $OnTs$, $partitions$, $TPos$)
12:    end parallel section
13: end procedure
```

E. Scalable Self-Tuning Implementation of SWA for Multicores

Now we have all necessary procedures for a modular description of the *scalable self-tuning* parallel algorithm for computing alignment score matrix H. An outline of the algorithm that uses four procedures described earlier is shown in **Algorithm 8**.

The algorithm first calls the procedure shown in **Algorithm 4** for computing an optimal length DB sequence. Next it calls the procedure shown in **Algorithm 6** for creating and initializing a 3-D array for alignment score matrix H. Then it calls the procedure shown in **Algorithm 5** for computing optimal number of threads, $OnTs$, and it is used for creating $OnTs$ threads for concurrent execution. Finally, from the



(a) Performances of Proposed Algorithm and HPG-SW on a 4-Core Single CPU Computer.

(b) Performances of Proposed Algorithm and HPG-SW on a 4-Core 2-CPU Computer.

(c) Performances of Proposed Algorithm and HPG-SW on 8-Core 2-CPU Computer.

Fig. 3. Comparison of Performances of Proposed Algorithm with Other Algorithms on Three Different Multicore Shared-Memory Computer Systems.

parallel section of the algorithm each thread invokes procedure shown in **Algorithm 7**.

In the next section we present some typical results obtained during our extensive evaluation of the algorithm [20].

V. Empirical Evaluation

Before presenting the results we briefly describe three shared memory computers that have been used for our evaluation.

A. Experimental Systems

We used both single CPU and double CPU systems. One computer has a single CPU and other two systems have double CPUs. Brief description of these machines, including operating systems and compilers used, are described next.

a) Single CPU System: This computer has a single AMD 4-Core 3.6GHz CPU with 8MB shared cache and 16GB RAM shared by all cores and is running 64-bit Ubuntu version 12.04. We used GCC compiler vesion 4.6.5 with OpenMP 3.0.

b) Double CPU Systems I: This computer system has two 4-Core AMD 2.2GHz CPUs. Each core has dedicated 512 KB cache. Two CPUs share 16GB RAM and is running 64-bit Red Hat Linux version 4.1.2. We used Intel C++ compiler version 11.1 with OpenMP 3.0.

c) Double CPU Syatem II: It has two 8-Core Intel Xeon 2.6GHz CPU with 20 MB of shared cache and 128GB RAM shared by both CPUs and is running 64 bit Red Hat Linux version 4.4.5. We used Intel C++ compiler version 11.1 with OpenMP 3.0.

B. Dataset for Evaluation

For evaluation of all algorithms we used genome sequence of *Pseudomonas aeruginosa* bacteria that was downloaded from NCBI genome database [2]. For testing purpose we used only a part of the sequence. The length of database sequence was varied from 1K to 25K and that of the query pattern was varied from 512 to 5K.

C. Performance Evaluation

We will present three sets of results in this section. The first set shows how computation rate is affected as the length of the database sequence is increased. We also show how speedup varies with the length of the database sequence. The second set demonstrates effect of length of partition of the database sequence on the per core computation rate. And the final three Figures show variation of computation rate as the number of threads per core is varied.

1) Effect of Length of the Database Sequence: The plots in Fig. 1(c) show that as the length of database sequence increases, rate of computation initially increases to a maximum and then decreases for all algorithms, except implementation proposed here. For ease of comparison computation rates have been normalized to per core. The proposed optimal algorithm reaches to a maximum and then retains the rate as the length of the database sequence increases.

The plots in Fig. 1(b) show speedup for our simple OpenMP parallel version, the HPG-SW implementation, and our optimal implementation. The simple OpenMP implementation has slightly lower speedup than the HPG-SW implementation. *Our optimal implementation not only has higher speedup, but it also has super-linear speedup when the length of the database sequence is greater than 10K.* The super-linear speedup is attributed to the fact that its computation rate does not decrease as the sequential version does (see Fig. 1(c)). Thus, mere speedup measurement may not reveal performance of parallel algorithm. We need to look more closely for increase and/or decrease of computation rate as the problem size increases.

2) Effect of the Length of Partition of H: We already know from our discussion in Section III-C that there is an optimal partition size for the H Matrix. But the plots in Fig. 2(b) reveal that the optimal size is not unique — computer system dependent. For both the 4-core single CPU computer and the 8-core double CPU computer the best performance is obtained when the partition length of H is about 375. On the other hand, for the 4-core double CPU computer the best performance is obtained when the length of the partition of H is about 350.

3) Effect of Number of Threads per Core: Recall that we have seen in Fig. 2(c) that there is an optimal number of threads for highest computation rate. We discovered several other interesting facts from our evaluation of the effect of the number of threads per core, some of which are presented here. We will divide our discussion into three paragraphs: 4-core single CPU computer, 4-core double CPU system, and 8-core double CPU system.

a) Single 4-core CPU Computer: Figure 3(a) shows that our optimal implementation attains highest computation rate when the number of threads per core is five, but for simple OpenMP implementation the highest computation rate is obtained when the number of threads per core is two. For the HPG-SW implementation the highest performance is obtained when number of threads per core is five, same as our optimal implementation.

b) Double 4-Core CPU Computer: Figure 3(b) shows that our optimal implementation attains highest computation rate when the number of threads per core is four, but for simple OpenMP implementation the highest computation rate is obtained when the number of threads per core is two. For the HPG-SW implementation the highest performance is obtained when number of threads per core is three.

c) Double 8-Core CPU Computer: Figure 3(c) shows that our optimal implementation attains highest computation rate when number of cores is four, but for simple OpenMP implementation and the HPG-SW implementation the highest performance is obtained when number of threads per core is two.

A general observation is that our optimal implementation significantly outperforms both simple OpenMP and HPG-SW implementations.

VI. Conclusions and Future Work

The parallel algorithm implementation technique presented in this paper for parallelization of Smith-Waterman algorithm for alignment of a query or pattern sequence, PT, with a database sequence, DB, is a new approach for implementing scalable self-tuning parallel algorithms for shared memory systems. The parallel algorithm we developed using the proposed approach scales well as the problem size grows. Moreover, the parallel algorithm implemented using the proposed technique deploys optimal number of threads for highest possible speedup. To the best of our knowledge, no other attempt has been made to optimize number of threads for achieving highest speedup.

A naive implementation using OpenMP failed to scale-up as the DB size increased. Similarly, the best known parallel implementation for multicore system, HPG-SW, from High Performance Genomics project [1] failed to scale-up. As discussed in Section V, our implementation maintained highest computation rate for all DB sizes, except when the DB size is too small. As a matter of fact, if DB size is too small, parallel algorithm should be avoided altogether.

Our implementation can be improved by parallelizing initialization part of the program. Also, the proposed technique can be used for implementation of other computation intensive problems, including most dynamic programming algorithms and matrix operations. *Efficient implementation of any algorithm has an added benefit of reduced energy consumption. Reduction of computation time for computation-intensive problems magnify the benefit of energy consumption.*

It would be interesting to develop analytical model for computing optimal number of threads for each core.

References

[1] HPG-SW. http://docs.bioinfo.cipf.es/projects/hpg-sw/files, downloaded in April 2013.

[2] NCBI resources. http://www.ncbi.nlm.nih.gov/, 2013.

[3] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[4] Sanghamitra Bandyopadhyay and Ramakrishna Mitra. A parallel pairwise local sequence alignment algorithm. *NanoBioscience, IEEE Transactions on*, 8(2):139–146, 2009.

[5] R.A. Chowdhury, Hai-Son Le, and V. Ramachandran. Cache-oblivious dynamic programming for bioinformatics. *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, 7(3):495–510, 2010.

[6] Rezaul Alam Chowdhury and Vijaya Ramachandran. Cache-efficient dynamic programming algorithms for multicores. In *Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures*, pages 207–216. ACM, 2008.

[7] Richard Cole and V. Ramachandran. Efficient resource oblivious algorithms for multicores with false sharing. In *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 201–214, 2012.

[8] Aaron Darling, Lucas Carey, and Wu-chun Feng. The design, implementation, and evaluation of mpiBLAST. *Proceedings of ClusterWorld*, 2003, 2003.

[9] Edans Flavius de O.Sandes and Alba Cristina M.A. de Melo. Retrieving Smith-Waterman alignments with optimizations for megabase biological sequences using GPU. *Parallel and Distributed Systems, IEEE Transactions on*, 24(5):1009–1021, 2013.

[10] Arthur L Delcher, Simon Kasif, Robert D Fleischmann, Jeremy Peterson, Owen White, and Steven L Salzberg. Alignment of whole genomes. *Nucleic Acids Research*, 27(11):2369–2376, 1999.

[11] Michael Farrar. Striped Smith-Waterman speeds database searches six times over other SIMD implementations. *Bioinformatics*, 23(2):156–161, 2007.

[12] M. Frigo, C.E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 285–297, 1999.

[13] Osamu Gotoh. An improved algorithm for matching biological sequences. *Journal of molecular biology*, 162(3):705–708, 1982.

[14] Ali Khajeh-Saeed, Stephen Poole, and J Blair Perot. Acceleration of the Smith-Waterman algorithm using single and multiple graphics processors. *Journal of Computational Physics*, 229(11):4247–4258, 2010.

[15] Y. Liu and B. Schmidt. CUSHAW2-GPU: empowering faster gapped short-read alignment using GPU computing. *Design Test, IEEE*, PP(99):1–1, 2013.

[16] Fernando Machado Mendonca and Alba Cristina Magalhaes Alves de Melo. Biological sequence comparison on hybrid platforms with dynamic workload adjustment. In *International Symposium on Parallel & Distributed Processing Workshops and PhD Forum*, 2013.

[17] N. Neves, N. Sebastiao, A. Patricio, D. Matos, P. Tomas, P. Flores, and N. Roma. BioBlaze: Multi-core SIMD ASIP for DNA sequence alignment. In *Application-Specific Systems, Architectures and Processors (ASAP), 2013 IEEE 24th International Conference on*, pages 241–244, 2013.

[18] William R Pearson and David J Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences*, 85(8):2444–2448, 1988.

[19] Torbjrn Rognes and Erling Seeberg. Six-fold speed-up of Smith-Waterman sequence database searches using parallel processing on common microprocessors. *Bioinformatics*, 16(8):699–706, 2000.

[20] Faisal Sikder and Dilip Sarkar. CPU oblivious and scalable parallel algorithm for Smith-Waterman algorithm for multicore and shared-memory compters. Technical report, University of Miami, 2013.

[21] Temple F Smith and Michael S Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

[22] Guangming Tan, Shengzhong Feng, and Ninghui Sun. Locality and parallelism optimization for dynamic programming algorithm in bioinformatics. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, SC '06, New York, NY, USA, 2006. ACM.

[23] Yuan Tang, Rezaul Chowdhury, Chi-Keung Luk, and Charles E Leiserson. Coding stencil computations using the pochoir stencil-specification language. In *3rd USENIX Workshop on Hot Topics in Parallelism (HotPar'11)*, 2011.

Low-power Task Scheduling in Homogeneous Quad-core Processor

Kyuhan Kim[1], Myungha Kim[2], Hyeonggeon Lee[1], Jong Kang Park[2], and Jong Tae Kim[1,2]
Department of IT Convergence[1]
Sungkyunkwan University
Suwon, Korea

Department of Electrical and Computer Engineering[2]
Sungkyunkwan University
Suwon, Korea
jtkim@skku.edu

Abstract— *In this paper, we consider the low-energy task scheduling algorithm in quad-core environment. As adding the factor of energy consumption in the DAG, we can consider the factor of energy consumption in algorithm level, and we propose the low-energy task clustering and scheduling algorithm. As a result, we can improve energy efficiency for the task scheduling in the DAG with time constraint. We focus on reducing the core idle state energy consumption and propose the modified task clustering method for lowering energy consumption for idle stats core execution.*

Keywords: DAG, Task clustering, Task scheduling

1 Introduction

As adopted multi-core processing techniques, the needs have been satisfied with the performance, but operating a large number of cores such as dual-core and quad-core brings about some issues related to low power design like power consumption and long battery life, and so on. In responding to this trend, we are concerned some critical issues for the task scheduling to promote energy efficiency in multi-core processor environment [1].

In this paper, we consider the low-energy task scheduling algorithm in a multi - core system, especially quad-core environment. Clustering is a mapping of the nodes of a DAG onto labeled clusters. A task is an indivisible unit of execution and presented by a node in a DAG. A set of task forms a cluster and all tasks in a cluster must execute on the same processor. Scheduling in DAGs is a set of tasks to a processor mapping and a task to time table mapping, expressed by the Gantt chart of a schedule. Generally, the problem of finding the optimum scheduling is to minimize the parallel time. [2]

In homogeneous quad-core processing environment, we propose the modified task clustering and allocation method to find the low-energy task scheduling algorithm. As we convert the execution cost and the communication cost expressed as time metric to energy metric (Joule) presented as energy consumption, we add the energy consumption factor in DAGs. As adding the factor of energy consumption in the DAG, previously based on time metric represented by the execution and communication cost, we can consider the factor of energy consumption in algorithm level, and we propose the low-energy task clustering and scheduling algorithm.

Many of task clustering heuristics are classified into different categories [3]. Some of the scheduling algorithms assume the availability of an unlimited number of processors, while some other algorithms assume the availability of a limited number of processors. The former classes are called the Unbounded Number of Processors (UNC), and the latter classes are called the Bounded Number of Processors (BNC). In both classes of algorithms, assume that the processors are fully connected and link contention or routing strategies are completely ignored. In both classes of algorithms, the target system is assumed to be a network Processing Element (PE) that consist of a processor and a local memory unit and the communication relies on message passing. These algorithms are task clustering heuristics included in UNC class of algorithms.

(1) Kim's and Brown's linear clustering algorithms

(2) Sarkar's clustering algorithm

(3) Dominant Sequence Clustering (DSC) algorithm

This paper is organized as follows. In section.2, we describe the problem motivation and a proposal idea. In section.3, we show the implementation of our task clustering method. Finally, we conclude the prospect of our proposed idea, in section.4.

2 Low-energy task scheduling algorithm

2.1 Motivation

We assume the simple architecture that consists of bus architecture and homogeneous quad-core processor [9] and add some kind of assumptions such as full utilization of bus transferred throughput, full utilization of the processor core, and so on. Organizing these assumptions, we convert the time value expressed by the execution costs and communication costs into the execution energy consumptions and communication energy consumptions.

First of all, we show the transformation procedure that expresses the conversion of the execution cost into execution energy consumption. The execution cost, which is the time needed to execute a task, defines the time consumption from starting time for task execution to completion time of task execution. We assume that the time delay not occurred during task executing on a core even though an event caused by interruption or other reason arises. When a task executes on a core, we assume that the core is utilized at maximum utilization. And then, the core operates at a maximum operating frequency, if we are able to know the cycle per instructions data on a core, we can calculate the total number of instructions per second. If we multiply the power consumption required to execute instruction by the number of executing instructions per second on a core, we can calculate the power consumption per second on a core at maximum utilization. Expressed by a formula [4], [5],

$$\mathbf{E_{execution}} = \mathbf{N_{instruction}} \times \mathbf{P_{instruction}} \times \mathbf{COST_{execution}}$$
$$\mathbf{N_{instruction}} = \mathbf{f_{peak}} \times a$$

where $\mathbf{E_{execution}}$ is the total energy consumption when a task node i was executed, $\mathbf{N_{instruction}}$ is the number of instructions executed by a core during a second $\mathbf{P_{instruction}}$ is the power consumption per instruction, $\mathbf{COST_{execution}}$ is the time presented by execution cost, $\mathbf{f_{peak}}$ is the core execution frequency, a is the cycle per executed instructions.

Subsequently, we show the transformation procedure that expresses the conversion of the communication cost into communication energy consumption. The communication cost, which is the time delay caused by data dependency between cores when tasks are assigned, defines the edge value in the DAG. If the data transmission is through a bus and the bus throughput is the same as full utilization of bus bandwidth, we assume that the total volume of the transferred data is expressed by the multiplication of the bus maximum bits per second and communication cost. As a result of the calculation above, total energy consumption caused by data transferring during the communication delay is the multiplication of power consumption per transfer bit through a bus and the volume of the total transferred data. Expressed by a formula [4], [5],

$$\mathbf{E_{communication}} = \mathbf{D_{BUS}} \times \mathbf{P_{bus\_line}} \times \mathbf{COST_{communication}}$$
$$\mathbf{D_{BUS}} = \mathbf{BW_{bus\_peak}} \times \mathbf{1\ s}$$

where $\mathbf{E_{communication}}$ is the total energy consumption when transferring data through bus architecture, $\mathbf{D_{BUS}}$ is the volume of transferring data with full utilization of bus throughput, $\mathbf{P_{bus\_line}}$ is a power consumption per bus line, $\mathbf{COST_{communication}}$ is the time presented by communication cost, $\mathbf{BW_{bus\_peak}}$ is the bandwidth of the bus architecture with peak data throughput.

TABLE I. COST TRANSFORMATION TABLE

	Cost(Time)	Energy
Node(Task execution)	1s	60.0mJ
Edge(Communication)	1s	22.6mJ
Node(Task idle)	1s	18.0mJ

As mentioned above, applying the assumptions, we assume that the ARM-core frequently used in embedded system is employed in the quad-core system and Advanced Microcontroller Bus Architecture (AMBA) is used as a bus. As we make reference to the other paper based on ARM core specification data, we are able to calculate more realistic value about energy consumption in the DAG.

We assume that the core operating frequency is 200MHz, having 1 cycle per instruction value and the bandwidth of the AMBA bus is 200Mbps. As a result, we obtain the table above, and we are able to add the energy consumption value to existing DAG based on the value of a table [6]. The energy consumption caused by the idle state time of core execution regard as 30% of execution energy consumption.

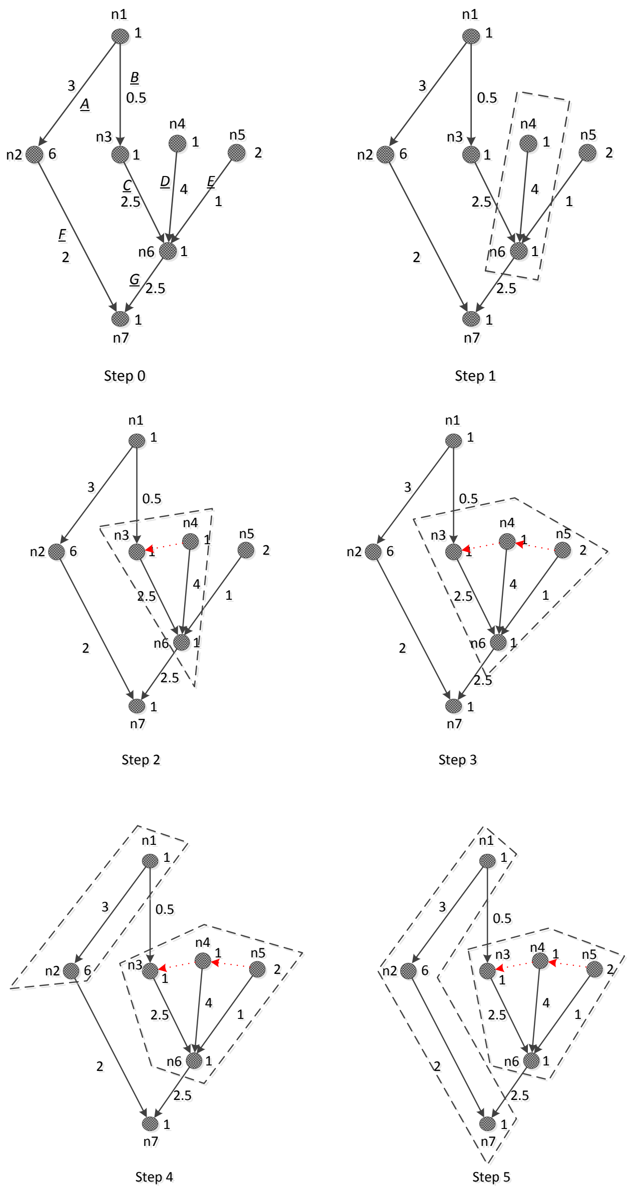
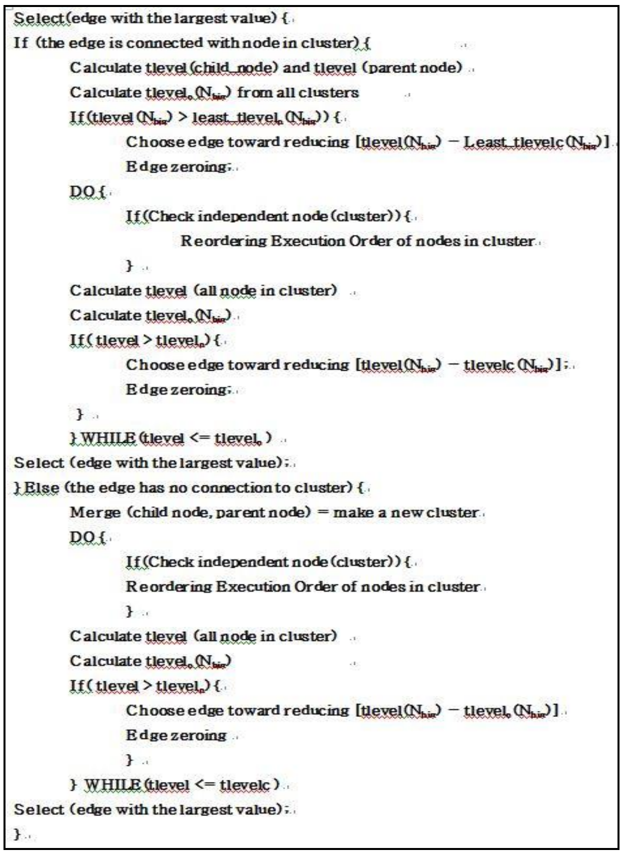


Fig. 1. Clustering steps of proposed method

2.2 Method of task scheduling

As mentioned, we propose the modified task clustering method that focuses on the reduction of idle state time on the core. We explain the method in detail in Fig.1. The main idea of the method is that edge zeroing should occur to forcing the reduction of the difference between the t-level and t-levelc which estimates the summation of cost from top level node in the cluster to the examined node. This has the same effect as assigning the task that has no dependency with examined node to the empty space on the core. To explain the method more easily, we depict a simple example DAG to apply the modified method. We demonstrate the proposal clustering method steps by using a DAG example in Fig.1 – step 0. The steps 1, 2, 3, 4, and 5 are shown in Fig.1. The dashed edges between n3, n4, n5 are the execution order within a cluster.



```
Select(edge with the largest value) {
If (the edge is connected with node in cluster) {
        Calculate tlevel (child_node) and tlevel (parent node)
        Calculate $tlevel_c(N_{big})$ from all clusters
        If(tlevel ($N_{big}$) > least_tlevel$_c$($N_{big}$)) {
                Choose edge toward reducing [tlevel($N_{big}$) − Least_tlevelc($N_{big}$)]
                Edge zeroing;
        DO {
                If(Check independent node (cluster)) {
                        Reordering Execution Order of nodes in cluster
                }
        Calculate tlevel (all node in cluster)
        Calculate $tlevel_c(N_{big})$
        If( tlevel > $tlevel_c$) {
                Choose edge toward reducing [tlevel($N_{big}$) − tlevelc ($N_{big}$)];
                Edge zeroing;
         }
        } WHILE (tlevel <= $tlevel_c$ )
Select (edge with the largest value);
} Else (the edge has no connection to cluster) {
        Merge (child node, parent node) = make a new cluster
        DO {
                If(Check independent node (cluster)) {
                Reordering Execution Order of nodes in cluster
                }
        Calculate tlevel (all node in cluster)
        Calculate $tlevel_c(N_{big})$
        If( tlevel > $tlevel_c$) {
                Choose edge toward reducing [tlevel($N_{big}$) − $tlevel_c$($N_{big}$)]
                Edge zeroing
                }
        } WHILE (tlevel <= tlevelc )
Select (edge with the largest value);
}
```

Fig. 2. Proposed Clustering Method

3 Experiment and evaluation

3.1 Comparison of Proposed Method and Sarkar's Algorithm

At the idle time, the proposed method has the number of cases, which have better or equal performance, numerically about 72.5% of all cases compared with Sarkar's algorithm. More than half of all cases show the reduction of the idle time compared with Sarkar's algorithm. But, the result about 72.5% shows that the portion of comparison for Sarkar's is lower than the portion of comparison for DSC algorithm. In the total energy consumption, as the portion of better or equal cases is 63.75% of all cases, the result is smaller than the case of idle time. In the parallel time, in most cases the proposed method has the equal or worse performance, numerically about 87.5% of all cases and this result show that has no effect in reducing the parallel time.

TABLE II. ANALYSIS EXPERMENT CASES IN COMPARISON OF SARKAR'S ALGORITHM

	Idle Time	Parall el Time	Total Energy Consumption
Better cases	27	10	25
Same cases	31	37	26
Worse cases	22	33	29
(Better + Same)cases (%)	72.5	58.75	63.75

To describe in more detail, we show the average ratio value in Table.3. The parallel time has 0.95, less than 1. This shows that the proposed method has less effect in reducing the parallel time. The results in Table.3 show that there are a lot of cases having better or equal performance for the idle time compared with Sarkar's algorithm, but some worst cases compared with other cases affect serious effect on average ratio value. As a result, the average ratio value has a poor result, numerically expressed by 5.34% degradation in the parallel time. While the average ratio of total energy consumption and idle time show the value 1.03 and 1.01, bigger than '1', the proposed method is a better than Sarkar's algorithm for total energy consumption and idle time aspects. Expressed by numerical way, the proposed method improves the idle time as reducing idle time about 0.51% and the total energy consumption as reducing energy consumption about 2.5% compared with Sarkar's algorithm.

TABLE III. AVERAGE RATIO IN COMPARISON OF SARKAR'S ALGORITHM

Proposed/DSC	Idle Time	Parallel Time	Total Energy Consumption
Average Ratio	1.01	0.95	1.03

3.2 Comparison result of the proposed method and other algorithms

We show the result of comparison of the proposed method and other algorithms, such as Sarkar's algorithm and DSC algorithm. In Table.4, we describe the best performance cases for all algorithms in approximately 80 cases of DAGs randomly generated. In the idle time and total energy consumption aspects, the proposed method represents that proposed method indicate better performance for 56 cases of 80 cases in idle time, numerically about 70% of all random DAGs, and for 54 cases of 80 cases in total energy consumption, numerically about 67.5% of all random DAGs. We show the comparison results that are graphs expressed as normalized value for each performance metric for 80 DAG cases. We express the normalized average values in the Table.5. In Table.5 we show that our proposed method is definitely better than other algorithms such as Sarkar's algorithm and DSC algorithm for the idle time and total energy consumption aspects.

According to Table.5, our proposed method reduces the idle time about 0.51% compared with Sarkar's algorithm and about 36.96% compared with DSC algorithm. And our proposed method reduces the total energy consumption about 2.5% compared with Sarkar's algorithm and about 18.26% compared with DSC algorithm. On the other hand, in the aspect of the parallel time, we cannot produce the improvement. The parallel time increases about 5.34% compared with Sarkar's algorithm and about 4.02% compared with DSC algorithm. Through increasing parallel time about 4~5%, we can reduce the idle time about 0.51 ~ 36.96% and the total energy consumption about 2.5~18.26%.

TABLE IV. THE NUMBER OF CASAES FOR BEST PERFORMANCE

Number of Best Cases			
	Idle	Parallel	Total Energy
Proposed	56	40	54
Sarkar's	50	58	49
DSC	20	53	16

TABLE V. NORMALIZED AVERAGE VALUES

Normalized Average Values			
	Idle	Parallel	Total Energy
Proposed	1	1	1
Sarkar’s	1.01	0.95	1.026
DSC	1.59	0.96	1.22

3.3 Analysis of the evaluation results

Through the experiment, we find that the proposed method is definitely better than the DSC algorithm in aspect of reducing idle time and energy consumption, but, in comparison of the proposed method and Sarkar's algorithm, we know otherwise. The proposed method has poor performance in aspect of reducing idle time compared with Sarkar's algorithm because there are some severe cases that has a negative effect on the average ratio value. We analyze some cases that have definitely large ratio value and finally figure out the reason to degrade performance.

First, the reason of degradation is stated that the proposed method is only focused on reducing idle time. As the number of tasks is static, which means that the task execution time is also static, the idle time naturally has a direct relationship to parallel time, which tends to increase according to parallel time. The second reason of performance degradation is continuous with the volume of tasks. If the volume of tasks is small, the number of cores used in executing tasks should keep less. If the clustering steps are needed to use a lot of cluster despite the number of tasks executed on cores small, overall parallel time can be reduced perhaps, but the idle times of each core seriously increase. To reduce the idle time in the cases of existing small tasks set, we must refine the proposed method as adding the function keeping the number of clusters less on proceeding clustering steps in the proposed method.

4 Conclusion

In this paper, we deal with the problem that how to assign a set of tasks in DAG expressed by time metric to achieve low energy effectiveness in an algorithmic level of abstraction.

To reduce overall energy consumption in the DAG, we propose the method to diminish the idle state time of the core caused by the data dependency. The proposed method is better than DSC algorithm in aspect of reducing the idle time and total energy consumption. The proposed method reduces 36.96% of the idle time and 18.26% of total energy consumption compared with DSC algorithm. But compared with Sarkar's algorithm, the proposed method needs more improvement in all aspects. The proposed method reduces very tiny value, 0.51% of the idle time and 2.5% of total energy consumption compared with Sarkar's algorithm.

5 References

[1] Brian Jeff, “Advances in big.LITTLE Technology for Power and Energy Savings (Improving Energy Efficiency in High-Performance Mobile Platforms)”, White Paper released by ARM, Sep.2012

[2] Tao Yang, Apostolos Gerasoulis, “A Fast Static Scheduling Algorithm for DAGs on an Unbounded Number of Processors”, supercomputing ’91 Proceeding of the ACM/IEEE Conference, 1991

[3] Apostolos Gerasoulis, Tao Yang, “A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors”, Journal of Parallel and Distributed Computing 16, page. 276-291, 1992,

[4] R.S. Bajwa, N. Schumann, H. Kojima, “Power analysis of a 32-bit RISC microcontroller integrated with a 16-bit DSP”, ISLPED ’97 Proceedings of the 1997 international symposium on Low power electronics and design

[5] Yan Zhang, Chen, T.Y., Wu Ye, Irwin, M.J., “System Level Interconnect Power Modeling”, ASIC Conference 1998 Proceedings, Eleventh Annual IEEE International

[6] Kanishka Lahiri, Anand Raghunathan, ”Power analysis of system-level on-chip communication architectures”, Hardware/Software Codesign and System Synthesis, 2004, CODES + ISSS 2004 International Conference

An SIMD Architecture for Shortest-Path Search and Its FPGA Implementation

Yasuhiro Takei, Masanori Hariyama and Michitaka Kameyama
Graduate School of Information Sciences, Tohoku University
Aoba 6-6-05, Aramaki, Aoba, Sendai, Miyagi, 980-8579, Japan
Email: {takei, hariyama, kameyama}@ecei.tohoku.ac.jp

Abstract— *Shortest-path search over graphs plays an important role in various applications. However, shortest path algorithms such as the Dijkstra's algorithm include complex processings. It is difficult for accelerators with fixed-datapath such as GPUs to accelerate these algorithms efficiently. This paper presents an FPGA-based accelerator with a SIMD architecture for the shortest-paths algorithm. In the proposed architecture, operations in the Dijkstra's algorithm are done with a high degree of parallelism, and the memory usage is reduced by using a memory management scheme. According to the evaluation, the proposed architecture is able to deal with graphs with more than 800,000 nodes on the Altera Stratix V.*

Keywords: Dijkstra's algorithm, Single Instruction Multiple Data, FPGA

1. Introduction

Recently, there is a huge demand to find the shortest-path for large scale graphs in many applications such as traffic simulation, social networking services and bioinformatics. The shortest-path problem is mainly classified into two types: single-source shortest-path problem (SSSP) and all pair shortest-path problem (APSP). Dijkstra's algorithm [1] and Bellman-Ford algorithm [2] are used to solve the SSSP. Warshall-Floyd Algorithm [3] is used to solve the APSP.

To accelerate the processing of large-scale graphs, there have been many software-based studies in terms of improving a data structure and reducing a computational amount reduction. Moreover, GPU[4] and FPGA[5] are used to accelerate the Warshall-Floyd Algorithm. However, it is difficult to accelerate these algorithms efficiently since most of the shortest-path algorithms include serial and complex data-flows. In order to process the shortest-path problem for large scale graphs, PC clusters with many CPUs are often used [6] because of their large memory capacity. However, these computing systems need very large space and power consumption.

To solve this problem, some FPGA-based accelerators have been proposed. FPGAs can implement application-specific data-paths by reconfiguration after fabrication. Moreover, the power consumptions of FPGAs are less than one-tenth of those of CPUs and GPUs. Tommiska[7], Fernandez[8], and Sridharan [9] have designed the FPGA-based architecture for SSSP with the Dijkstra's algorithm. However, their work did not consider processing large-scale graphs.

This paper presents an FPGA-based accelerator for the Dijkstra's algorithm. In order to accelerate processing and memory access in the Dijkstra's algorithm, we design the SIMD (single instruction multiple data) architecture. We consider how to search the shortest path with a high degree of parallelism, and how to reduce the memory usage on a limited memory space. The proposed architecture is implemented on the FPGA board for evaluating the resource usage.

2. Dijkstra's algorithm and implementation of an FPGA

The Dijkstra's algorithm is one of the most popular algorithm to solve SSSP. Because it is easy to implement, this algorithm is used in various applications such as analysis of the internet, traffic simulation and so on. Let the node where we are starting with be called S. Let $d(y)$ be the distance from S to node y. The flow of the Dijkstra's algorithm is represented by the following steps.

Step1: Assign to every node a tentative distance: set it to zero for S, and to infinity for all other nodes. Mark all nodes *"unvisited"*.

Step2: Select the *unvisited* node which has the smallest tentative distance and make it the *"current node"*.

Step3: For the *current node*, consider all of *unvisited* neighbor nodes and update their tentative distance. If the *current node* is A , and one of the *unvisited* neighbor node is B, set the tentative distance of B ($td(B)$) to $\min(td(B), d(A) + l_{AB})$,where l_{AB} is the length of the edge between A and B. When considering all of *unvisited* neighbor nodes of the *current node*, mark the *current node "visited"*.

Step4: Until all nodes are marked *visited*, go back to Step2.

The processing time of the Dijkstra's algorithm depends on searching the minimum distance in Step2 and updating tentative distances in Step3. In these processings, there are many comparison operations on multiple node data. The SIMD architecture is suitable for processing in parallel.

To process the Dijkstra's algorithm, a memory space for tentative distances and paths is required. Since these data are read and updated frequently, on-chip memory on an FPGA is suitable. However, the capacity of the on-chip memory is small. The memory management is required for reducing the memory usage and the processing time. It is unnecessary to store the distance on the visited nodes and infinity. These unused data should be replaced or avoid storing in the memory module.

3. Architecture

Figure 1 shows the overall architecture of the proposed FPGA-based accelerator. This architecture consists of a SIMD module, a memory controller, a FIFO, an adder and a current node register. The SIMD module is used for searching the minimum distance and updating tentative distances. The memory controller and the FIFO are used for transferring the graph data from an external memory to the SIMD module. The current node register stores the current node number and the distance from the start to the current node.

Figure 2 shows the architecture of the SIMD module. This module consists of block RAMs, comparators, and a counter-based address generation unit(AGU). The block RAMs store the values of node number, the tentative distance and the previous node. In the initial state, the block RAMs are empty.

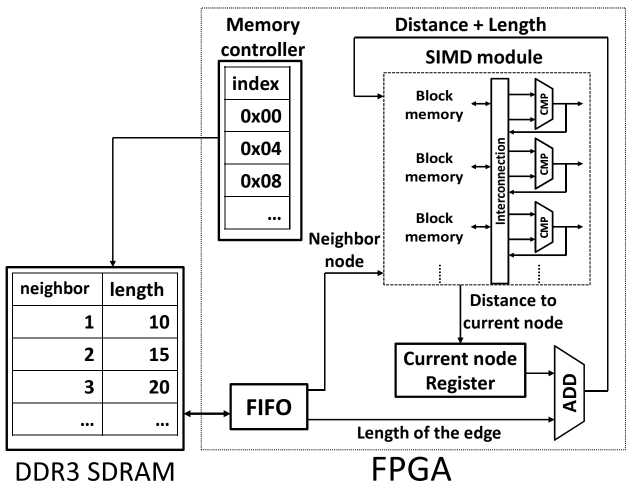
For updating the tentative distances, the neighbor node number is searched in parallel as shown in Fig.3. Then the tentative distance at the neighbor node is compared with the sum of the distance at the current node register and the length of the edge. The tentative distance and the previous node are updated as shown in Fig.4. If the neighbor node number is not in the block RAMs, the data of the neighbor node number, the sum of the distance and the length, and the current node number are stored as new data.

For the searching of the minimum distance, comparators are connected as shown in Fig.5. When the minimum value searching is completed, the value of the minimum distance and the node number are stored in the current node register. The memory space for the current node can be overwritten as shown in Fig.6. Hence, the memory usage for the tentative distance can be reduced.

Let us consider implementing the graph data on the FPGA. In related works of the FPGA implementation of the Dijkstra's algorithm [7],[8],[9], the adjacency matrix is used because the overhead of the memory access is small. However, very large memory space is required for unnecessary data that indicates unconnected edges if the graph is sparse. In this work, adjacency list is used for using a limited memory space in FPGA efficiency. The lengths of edges and the neighbor node number are stored in the external memory since the amount of these data is large. By using a index pointer in the memory controller, the length of the edge and the neighbor node number are transferred from the external memory to the FPGA.

4. Evaluation of the proposed architecture

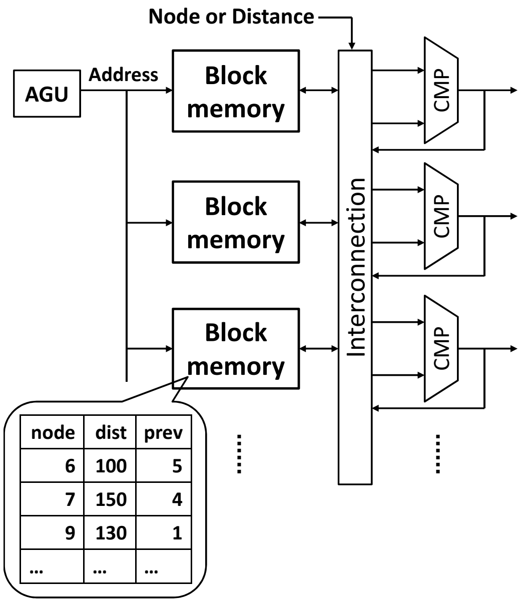
We use the Terasic DE5-NET FPGA board [10]. This board includes an Altera StratixV 5SGXEA7N2F45C2, and a DDR3 SDRAM (4GB). Altera Quartus 13.1 is used for design. For a proto-type design, we implement the SIMD module for shortest-path search in 32 nodes. Table 1 shows the resource usage. The resource usage changes by the numbers of block memories and comparators. The degree of parallelism can increase if many block memories and comparators are implemented. However, the logic utilization becomes large.



index
0x00
0x04
0x08
...

neighbor	length
1	10
2	15
3	20
...	...

Fig. 1: Overall architecture



node	dist	prev
6	100	5
7	150	4
9	130	1
...	...	...

Fig. 2: SIMD module

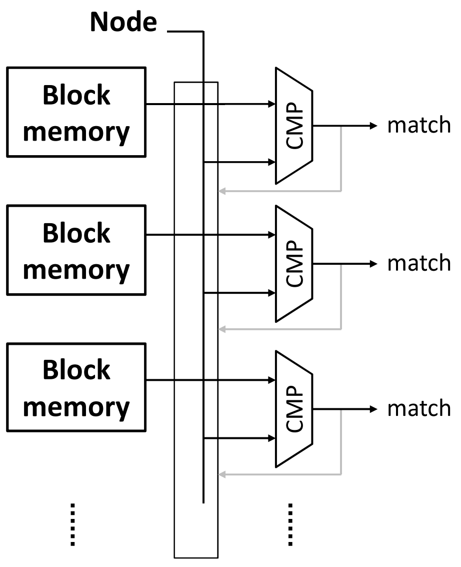


Fig. 3: Searching a node number in the SIMD module

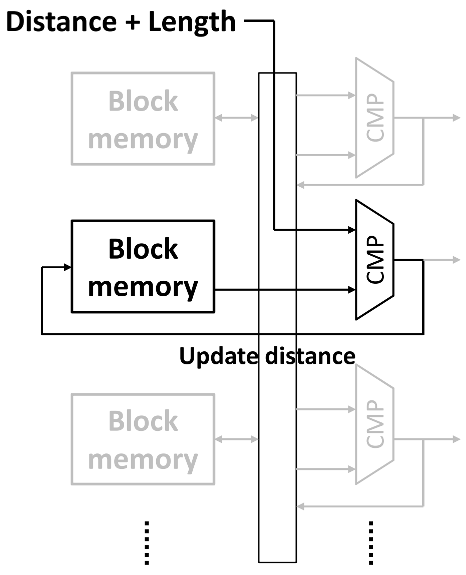


Fig. 4: Updating data in the SIMD module

Note that the number of block memory module on the FPGA is 5120.

Let us consider the memory usage for storing the tentative distances and node numbers. The FPGA has about 50M bits of on-chip memory. Let the number of bits in every node be 64, about 800,000 nodes can be stored in the FPGA. According to Section 3, the data of the visited nodes can be replaced by the data of the unvisited nodes, and a memory space is not required if the tentative distance is infinity. Hence, the proposed architecture can process the shortest path search on very large scale graphs with more than 800,000 nodes.

The processing time depends on the amount of data in the block memories. If the input graph is sparse, the amount of data in the block memories is small, and the processing time is small. Most of the large scale graphs in the real world, such as road network, traffic network, social network and so on, are sparse. Hence, the proposed architecture may be suitable for processing the large scale graphs in the real world. We are now designing the overall architecture as shown in

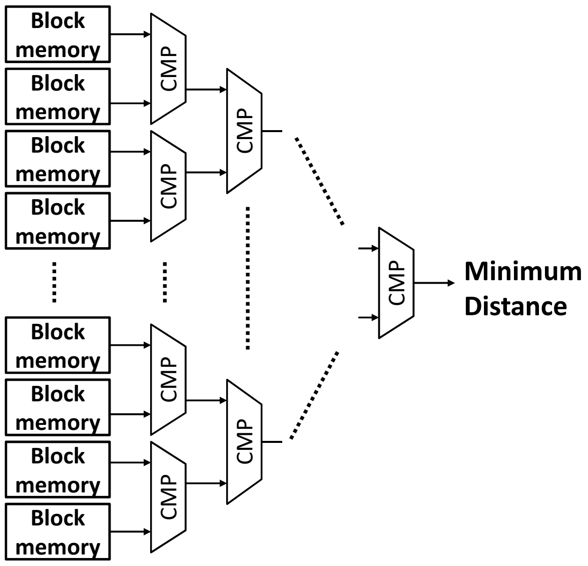
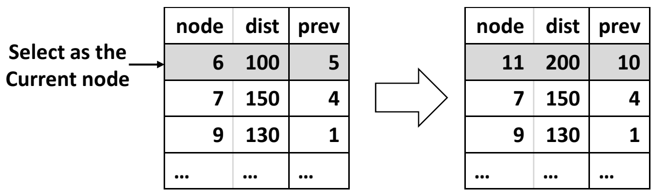


Fig. 5: Searching the minimum distance in the SIMD module



node	dist	prev
6	100	5
7	150	4
9	130	1
...	...	...

node	dist	prev
11	200	10
7	150	4
9	130	1
...	...	...

Fig. 6: Overwriting unused data in the block memory

Fig.1, and measuring the total processing time of the Dijkstra's algorithm.

5. Conclusions

We have proposed an FPGA-based accelerator with an SIMD module for a shortest-path search. We discussed about how to parallelize the Dijkstra's algorithm and how to use limited memory space on FPGA boards. According to the evaluation, the proposed architecture can accelerate shortest-path search on large scale graphs with more than 800,000 nodes on the Altera Stratix V.

In future works, we are going to implement large scale architecture on the FPGA board in order to accelerate applications with shortest-path problems such as the traffic network analysis. Moreover, it is very interesting to implement improved shortest-path algorithms, such as the A* algorithm[11] and the high-way dimension algorithm[12] on an FPGA.

Table 1: Resource usage

Block memory	Comparator	LUT	Register	Memory bit
2	8	282	152	512
4	4	137	68	512
2	16	572	321	1024
4	8	282	382	1024

Acknowledgement

This work is supported by JSPS KAKENHI grant number 24300013.

References

[1] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs", Numerische Mathematik, 1(1): pp.269–271, 1959.
[2] R. Bellman, "On a Routing Problem", Technical report, DTIC Document, 1956.
[3] R. W. Floyd, "Algorithm 97: Shortest Path", Commun. ACM, 5(6) pp.345–346, June 1962.
[4] G. J. Katz and J. T. Kider Jr, "All-Pairs Shortest-Paths for Large Graphs on the GPU", In Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware, 2008.
[5] U. Bondhugula, A. Devulapalli, J. Fernando, P. Wyckoff, and P. Sadayappan, "Parallel FPGA-Based All-Pairs Shortest-Paths in a Directed Graph", In Proceedings of Parallel and Distributed Processing Symposium, 2006.
[6] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. "Pregel: a System for Large-Scale Graph Processing", In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, pp. 135–146, 2010.
[7] M. Tommiska and J.Skytta. "Dijkstra's Shortest Paths Algorithm in Reconfigurable Hardware", In Proc. Field Programmable Logic and Applications, pp. 653–657, 2001.
[8] I. Fernandez, J. Castillo, C. Pedraza, C. Sanchez, and J. I. Martinez, "Parallel Implementation of the Shortest Path Algorithm on FPGA" In Proc. 4th Southern Conf. on Programmable Logic., pp. 245–248, 2008.
[9] K. S. T.K.Priya and P. Kumar, "Hardware Architecture for Finding Shortest Paths", In Proc. IEEE Region 10 Conf., pp. 1–5, 2009.
[10] Terasic, "DE5-NET FPGA Development Kit", http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=EnglishCategoryNo=158No=526.
[11] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths", Systems Science and Cybernetics, IEEE Transactions on, 4(2):pp.100–107, 1968.
[12] I. Abraham, A. Fiat, A. V. Goldberg, and R. F. Werneck, " Highway Dimension, Shortest Paths, and Provably Efficient Algorithms", Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 782–793, 2010.

Parallel Fuzzy Filter for Impulse Noise Removal

J. Arnal[1], L. A. Drummond[2], L. B. Súcar[1], and V. Vidal[3]
[1]Departamento de Ciencia de la Computación e Inteligencia Artificial,
Universidad de Alicante, San Vicente del Raspeig, Alicante, Spain
[2]Lawrence Berkeley National Laboratory, Berkeley, CA, USA
[3]Departamento de Sistemas Informáticos y Computación,
Universidad Politécnica de Valencia, Grao de Gandia, Valencia, Spain

Abstract—*A parallel algorithm for reducing impulse noise from color images is proposed. The algorithm is based on a fuzzy metric and is performed in two steps followed by noise filtering using the vector median filter. An implementation of the algorithm on multi-core interface using the Open Multi-Processing (OpenMP) is presented. A performance analysis with large images is conducted. Performance is evaluated in terms of execution time and in terms of PSNR. Results show that the proposed filter obtains good performance in terms of PSNR. After applying the multicore optimization strategies, the observed time shows that the proposed filter is able to remove impulse noise in real-time.*

Keywords: Parallel computing, OpenMP, Colour image filter, Fuzzy metric, Impulse noise

1. Introduction

Digital images are often corrupted by noise during acquisition and transmission processes. An important problem in image processing is to remove that noise preserving some image features such as edges, textures, and fine details. An specially common type of noise is the impulse noise [1], [2]. Impulsive noise is commonly caused by the sensor malfunction and other hardware in the process of image formation, storage or transmission [3]. For example, impulsive noise is found in situations where quick transients, such as faulty switching, take place during image processing. This type of noise affects some individual pixels, by changing their original values. The most common noise impulsive model is the Salt and Pepper noise (or fixed value noise), which considers that the new, wrong, pixel value is an extreme value within the signal range. This is the noise type considered in this paper.

Many methods have been introduced to remove impulse noise (see e.g. [3]–[13]).

In [13] a two-step procedure using the fuzzy ROD (ROD was introduced in [14], and generalized to colour images in [15]) statistic is proposed. In the first step are diagnosed pixels which are clearly noisy or clearly noise-free, and in the second step are diagnosed pixels which are difficult to classify.

Experimental results showed that this filtering technique exhibits competitive results with respect to other state-of-the-art methods. On the other hand, because of the large data set size of high-resolution image data, sequential computers do not have sufficient computing power to perform this algorithm in real-time. Then, this filter has shown good results in quality but does not seem appropriate for real-time processing.

Moreover, this algorithm exhibits a high degree of data locality and parallelism and thus is suitable for parallel computing hardware. Due to these causes, in this paper we introduce a parallel version of fuzzy peer group based on filters introduced in [13] in order to retain their good quality results while trying to improve their performance, so as to make them usable in real-time processing.

We have tested this parallel algorithm developing programs for multi-cores, obtaining a nearly linear speedup as a function of the number of processors used. Nowadays, multi-cores are widely available, and then the introduced approach is an effective, practical, and economical mode for real-time image processing.

This paper is organised as follows: Section 2 explains the proposed parallel noise removal method. Experimental results are shown in Section 3, and finally, the conclusions are presented in Section 4.

2. Parallel noise removal method

Let the color image A be defined as a mapping $\mathbb{Z}^2 \to \mathbb{Z}^3$. That is, the color image is defined as a two-dimensional matrix A of size $M \times N$ consisting of pixels $x_i = (x_i(1), x_i(2), x_i(3))$, indexed by i, which gives the pixel position on the image domain Ω. Components $x_i(l)$, for $i = 1, 2, ..., M \times N$ and $l = 1, 2, 3$, represent the color channel values in RGB quantified into the integer domain.

Let W represents a square filtering window consisting of $n \times n$ color pixels centered at pixel x_0. And let $x_i \in W$, $i = 1, \ldots, n^2 - 1$ denote the pixels in the neighborhood of x_0. The parallel denoising algorithm introduced in this study uses the *fuzzy peer group* of a central pixel x_i in a window W according to [16] and using a *fuzzy metric*. In order to describe the parallel algorithm, and how the pixels were

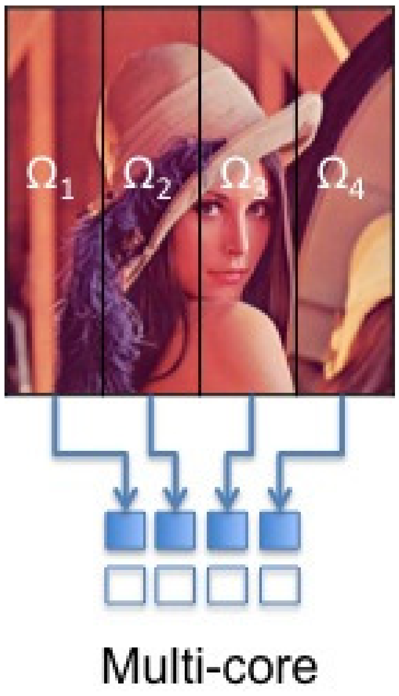


Fig. 1: Image domain decomposition: Distributed image on 4 cores.

assigned to each computing element, we consider a domain decomposition of the image domain Ω in P subdomains $\{\Omega_i\}_{i=1}^{P}$, where P is the number of processors. Fig. 1 shows an example of the image domain decomposition used in the experiments.

Fig. 2 shows the parallel filtering algorithm. The method is divided into two stages: noise detection and noise removal. To detect the impulse noise a two steps detection process is used. In the first step, pixels which are clearly noisy or clearly noise-free are classified. The filtering scheme used is based on the FROD statistic described in the following lines.

Consider for each pixel $x = (x(1), x(2), x(3))$, in RGB format, a $n \times n$, window W_x centered at x. Let W_x^0 the set of neighbours pixels of x in W_x, i.e., $W_x^0 = W_x - \{x\}$. In order to compute the ROD statistic [14] the distances $d_{x,x_i},\ x_i \in W_x^0$ are ordered in an ascending sequence obtaining a set of non-negative real numbers $r_j(x)$ such that: $r_1(x) \leq r_2(x) \leq \ldots \leq r_{n^2-1}(x)$ Then, fixed a positive integer $m \leq n^2 - 1$, the m rank-ordered difference statistic ROD_m is defined in [14] as,

$$ROD_m(x) = \sum_{j=1}^{m} r_j(x). \quad (1)$$

Then, ROD_m expresses the global distance between x and its m closest neighbors. This distance is expected to be greater for impulse noise pixels than for noise-free pixels.

```
Require: Image $A$, a domain decomposition $\{A_{\Omega_k}\}_{k=1}^{P}$, $th_1, th_2, th_3$
Ensure: Filtered image.
 1: for $k = 1, \ldots, P$, in parallel do
 2:   Impulse noise detection: Step 1
 3:   for $x_i$ pixel in $A_{\Omega_k}$ do
 4:     Processor $k$ calculates: $d = FROD_m(x_i)$;
 5:     if $(d > th_1)$ then
 6:       pixel $x_i$ and $\forall\ x_j$ used in $FROD_m(x_i)$ are classified as noise-free;
 7:     else
 8:       if $(d < th_2)$ then
 9:         $x_i$ is classified as noisy;
10:       else
11:         $x_i$ is classified as non-diagnosed;
12:       end if
13:     end if
14:   end for
15:   Impulse noise detection: Step 2
16:   for $x_i$ pixel in $A_{\Omega_k}$ classified as non-diagnosed in Step 1 do
17:     Processor $k$ calculates $d = FROD_{m'}(x_i)$ excluding the pixels previously classified as noisy;
18:     if $(d > th_3)$ then
19:       pixel $x_i$ and $\forall\ x_j$ used in $FROD_{m'}(x_i)$ are classified as noise-free;
20:     else
21:       $x_i$ is classified as noisy;
22:     end if
23:   end for
24:   Impulse noise reduction:
25:   for $x_i$ pixel en $A_{\Omega_k}$ classified as noisy do
26:     $x_i$ s replaced with $\mathrm{VMF_{out}}$
27:   end for
28: end for
```

Fig. 2: Parallel filtering algorithm.

To obtain the distances $d_{x,x_i},\ x_i \in W_x^0$, we use the fuzzy metric M_∞ [13], that has been proven to be especially sensitive to impulsive noise. This metric, given two RGB color image vectors x_i, x_j, is defined by

$$M_\infty(x_i, x_j) = \min_{l=1}^{3} \frac{\min\{x_i(l), x_j(l)\} + K}{\max\{x_i(l), x_j(l)\} + K} \quad (2)$$

We have set $K = 1024$ which has been proved to be an appropriate value for RGB colour vectors [17].

Considering the usage of the M_∞ fuzzy metric to obtain the distances $d_{x,x_i},\ x_i \in W_x^0$, the fuzzy ROD (FROD) statistic is defined as follows. Taking the fuzzy distances d_{x,x_i} ordered in a descending sequence $s_1(x) \geq s_2(x) \geq \cdots \geq s_{n^2-1}(x)$ the $FROD_m$ statistic is defined by

$$FROD_m(x) = \prod_{j=1}^{m} s_j(x). \qquad (3)$$

An impulse noise pixel will present a low value of $FROD_m$ because is not expected to be similar to its neighbours, whereas noise-free pixels are expected to have a $FROD_m$ value closer to 1.

In order to classify pixels which are clearly noisy or clearly noise-free we use the $FROD_m(x)$ value, where $m < n^2 - 1$ is a filter parameter. If $FROD_m(x)$ is greater than a first parameter th_1, then x and its m neighbours used in the computation of $FROD_m(x)$ are classified as noise-free. If $FROD_m(x)$ is less than a second parameter th_2 ($th_2 < th_1$), x is classified as noisy. If x satisfies $th_1 \geq FROD_m(x) \geq th_2$, then we conclude that it is not possible to classify x at this step, and it is analyzed in a second step. In the second step, a third threshold parameter th_3 is used. In this step $FROD_{m'}(x)$ is computed on W_x^0 excluding the pixels previously classified as noisy, and using another filter parameter $m' < m$. If $FROD_{m'}(x) > th_3$, then x and its m' neighbors involved in the computation of $FROD_{m'}(x)$ are classified as noise-free. Otherwise, x is classified as noisy.

After the noise detection steps in the noise reduction stage, each pixel classified as noisy is replaced with VMF_{out} [18] operating over its noise-free neighbours in a $n \times n$ window.

3. Experimental Results

We carried out specific experiments and developments using two different machines and software settings which are included in the following list:

- Multi-core 1: Intel Xeon CPU E5320 (8 cores), 1.86 GHz, 8GB RAM, Linux Ubuntu 8.04.1. GNU Fortran compiler.
- Multi-core 2: Intel XEON X5660 (12 cores), 2.8 GHz, 48 GB RAM, Linux CENTOS 5.6. Intel Fortran Compiler.

Different test images shown in Fig. 3 were used in the experiments: Lenna [4], Caps [19], Motorbikes [19], Statue [19], Bus [20], and Toy [20]. These images have been corrupted with impulse noise. The random-value impulse noise [1] was considered. We denote by p the noise appearance probability. In our tests we have used $p \in [0, 0.1]$.

Fig. 1 shows an example of the image domain decomposition used in the experiments using 4 cores. In order to adjust of the filter parameters th_1, th_2, and th_3 in [13] the filter performance was analyzed in terms of *Peak Signal to Noise Ratio* (PSNR), as a function of th_1, th_2, and th_3 contaminating images with different probabilities of impulse noise p. Accordingly to that study, our results were obtained setting th_1, th_2, and th_3 proportionally to p as follows.

$$\begin{aligned} th_1 &= 0.90 + \frac{p}{0.4}0.07, && (4)\\ th_2 &= 0.87 + \frac{p}{0.4}0.06, && (5)\\ th_3 &= 0.97 + \frac{p}{0.4}0.01, && (6)\end{aligned}$$

According to previous research [13], in the experiments we have considered a 3×3 filter window ($n = 3$) and $m = 3,\ m' = 1$.

We designed both the serial code and parallel code and then compared the execution time.

Tables 1–4 show the results obtained on Multi-core 2 dividing the image among different number of cores and Fig. 4, presents the speedup obtained for different sizes of Caps and Statue images. To quantify parallel performance, parallel speedup S_P is computed as:

$$S_P = \frac{T_{seq}}{T_P} \qquad (7)$$

where T_{seq} is the execution time of the sequential algorithm and T_P is the execution time of the parallel algorithm using P processors. The results show that a significant speedup is achieved. On the other hand, Figs. 4 shows that the optimal number of processors to filter the image depend on the image size.

The filter performance has been evaluated using the *Peak Signal to Noise Ratio* (PSNR), that measures the noise suppression capability [1]. Fig. 5 shows that PSNR performance improves as image size increases. From the visual point of view, by inspecting the denoised images in Fig. 6, it can be concluded that the filter obtains robust results. $FROD_m$ consistently diagnose and reduce impulses while preserving the quality of image edges and details.

4. Conclusion

A parallel algorithm based on a fuzzy metric has been proposed to detect and remove impulse noise in digital images. We have implemented it on multi-cores using OpenMP. The parallel algorithm introduced demonstrated a significant speedup in processing large images compared to sequential algorithm. The algorithm obtained a nearly linear speedup as a function of the number of processors used. Multi-cores are widely available, so the introduced approach is an effective, practical, and economical mode of real-time image processing.

References

[1] K. N. Plataniotis and A. N. Venetsanopoulos, *Color Image Processing And Applications*. New York, USA: Springer-Verlag, 2000.

[2] C. Boncelet, *Image Noise Models*. London: Academic Press, 2000, pp. 325–335.

[3] B. Smolka, "Peer group switching filter for impulse noise reduction in color images," *Pattern Recognit. Lett.*, vol. 31, no. 6, pp. 484–495, 2010.

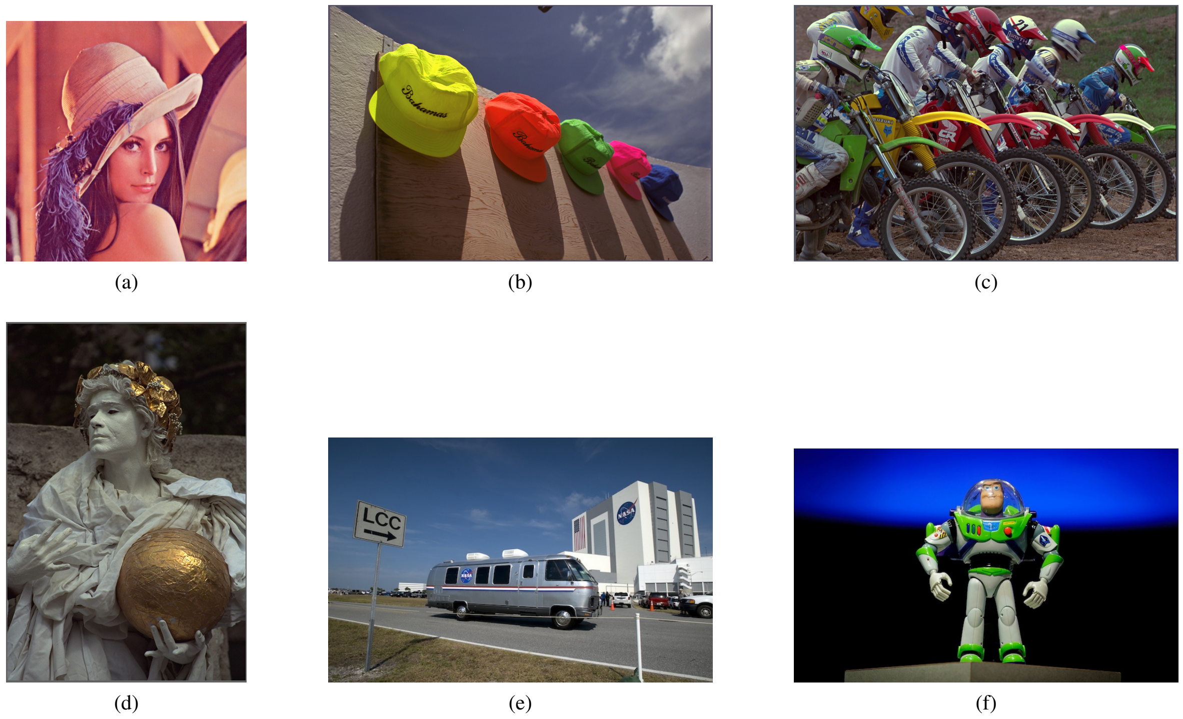


Fig. 3: Test images: (a) Lenna, (b) Caps [19], (c) Motorbikes [19], (d) Statue [19], (e) Bus [20], and (f) Toy. [20]

Table 1: Computational time in seconds on Multi-core 2. Caps image. 1600 $\times$ 1067 pixels

Impulse Noise	Number of processors							PSNR
	1	2	4	6	8	10	12	
$p = 0.05$	0.481173992	0.243747950	0.124355078	0.085099936	0.068102121	0.054935932	0.054033995	39.05
$p = 0.1$	0.694947004	0.351624966	0.178093195	0.119786024	0.091516972	0.075078964	0.064303875	32.90

Table 2: Computational time in seconds on Multi-core 2. Motorbikes image. 1600 $\times$ 1067 pixels

Impulse Noise	Number of processors							PSNR
	1	2	4	6	8	10	12	
$p = 0.05$	0.475349188	0.240406036	0.122515917	0.084091902	0.06705308	0.055345058	0.064940929	36.30
$p = 0.1$	0.685279131	0.34661603	0.175352097	0.118137121	0.091350079	0.075542927	0.06315589	30.67

Table 3: Computational time in seconds on Multi-core 2. Bus image. 3986 $\times$ 2538 pixels

Impulse Noise	Number of processors							PSNR
	1	2	4	6	8	10	12	
$p = 0.05$	2.842068911	1.434518814	0.723886013	0.485067129	0.367190838	0.300970078	0.271684885	37.51
$p = 0.1$	4.100522995	2.073998928	1.045986891	0.700412035	0.528025866	0.433892012	0.377089977	31.68

[4] J.-G. Camarena, V. Gregori, S. Morillas, and A. Sapena, "Fast detection and removal of impulsive noise using peer groups and fuzzy metrics," *J. Visual Commun. Image Represent.*, vol. 19, no. 1, pp. 20 – 29, 2008.

[5] A. Toprak and I. Güler, "Impulse noise reduction in medical images with the use of switch mode fuzzy adaptive median filter," *Digital Signal Process.*, vol. 17, no. 4, pp. 711–723, 2007.

[6] S. Schulte, M. Nachtegael, V. D. Witte, D. V. der Weken, and E. E. Kerre, "A fuzzy impulse noise detection and reduction method," *IEEE Trans. Image Process.*, vol. 15, no. 5, pp. 1153–1162, 2006.

[7] S. Schulte, S. Morillas, V. Gregori, and E. E. Kerre, "A new fuzzy color correlated impulse noise reduction method," *IEEE Trans. Image Process.*, vol. 16, no. 10, pp. 2565–2575, 2007.

[8] S. Schulte, V. D. Witte, M. Nachtegael, D. V. der Weken, and E. E. Kerre, "Fuzzy two-step filter for impulse noise reduction from color images," *IEEE Trans. Image Process.*, vol. 15, no. 11, pp. 3567–3578, 2006.

[9] ——, "Fuzzy random impulse noise reduction method," *Fuzzy Sets Syst.*, vol. 158, no. 3, pp. 270–283, 2007.

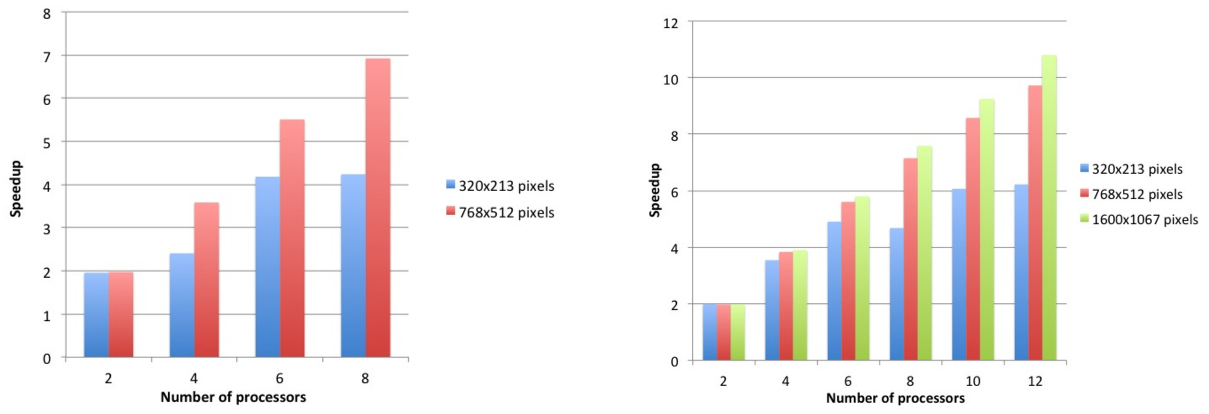
[10] T. Melange, M. Nachtegael, and E. E. Kerre, "Fuzzy random impulse noise removal from color image sequences," *IEEE Trans. Image Process.*, vol. 20, no. 4, pp. 959–970, Apr. 2011.

[11] J.-G. Camarena, V. Gregori, S. Morillas, and A. Sapena, "Some improvements for image filtering using peer group techniques," *Image Vision Comput.*, vol. 28, no. 1, pp. 188–201, 2010.

[12] S. Morillas, V. Gregori, and G. Peris-Fajarnés, "Isolating impulsive noise pixels in color images by peer group techniques," *Comput. Vision Image Understanding*, vol. 110, no. 1, pp. 102–116, 2008.

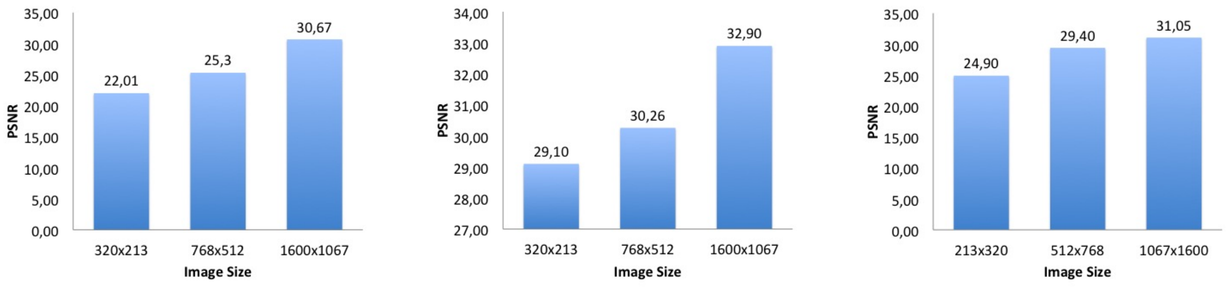
Table 4: Computational time in seconds on Multi-core 2. Toy image. 3551×2160 pixels

Impulse Noise	Number of processors							PSNR
	1	2	4	6	8	10	12	
$p = 0.05$	1.905487061	0.956923962	0.513839006	0.354580879	0.270764112	0.221177816	0.227982998	39.74
$p = 0.1$	2.46735096	1.232701063	0.655728817	0.454741001	0.346143007	0.28381896	0.243230104	32.02



(a) Multi-core 1 (b) Multi-core 2

Fig. 4: Speedup for different image sizes. Caps Image corrupted with impulse noise $p = 0.1$.



(a) Motorbikes image (b) Caps image (c) Statue image

Fig. 5: PSNR for different image sizes. Images corrupted with impulse noise $p = 0.1$.

[13] J.-G. Camarena, V. Gregori, S. Morillas, and A. Sapena, "Two-step fuzzy logic-based method for impulse noise detection in colour images," *Pattern Recognit. Lett.*, vol. 31, no. 13, pp. 1842–1849, 2010.

[14] R. Garnett, T. Huegerich, C. K. Chui, and W. He, "A universal noise removal algorithm with an impulse detector," *IEEE Trans. Image Process.*, vol. 14, no. 11, pp. 1747–1754, 2005.

[15] G. Peris-Fajarnés, B. Roig, and A. Vidal, "Rank-ordered differences statistic based switching vector filter," in *Proceedings of ICIAR 2006, Third International Conference on Image Analysis and Recognition, Póvoa de Varzim, Portugal*, ser. Lecture Notes in Computer Science, vol. 4141. Springer, Berlin, 18-20 September 2006, pp. 74–81.

[16] S. Morillas, V. Gregori, and A. Hervás, "Fuzzy peer groups for reducing mixed gaussian-impulse noise from color images," *IEEE Trans. Image Process.*, vol. 18, no. 7, pp. 1452–1466, 2009.

[17] S. Morillas, V. Gregori, G. Peris-Fajarnés, and P. Latorre, "A fast impulsive noise color image filter using fuzzy metrics," *Real-Time Imaging*, vol. 11, no. 5-6, pp. 417–428, Oct. 2005.

[18] J. Astola, P. Haavisto, and Y. Neuvo, "Vector Median Filters," *Proc. IEEE*, vol. 78, no. 4, pp. 678–689, APR 1990.

[19] (2013) Kodak lossless true color image suite. [Online]. Available: http://r0k.us/graphics/kodak/index.html

[20] (2013) Nasa hq photo. [Online]. Available: http://www.flickr.com/photos/nasahqphoto/

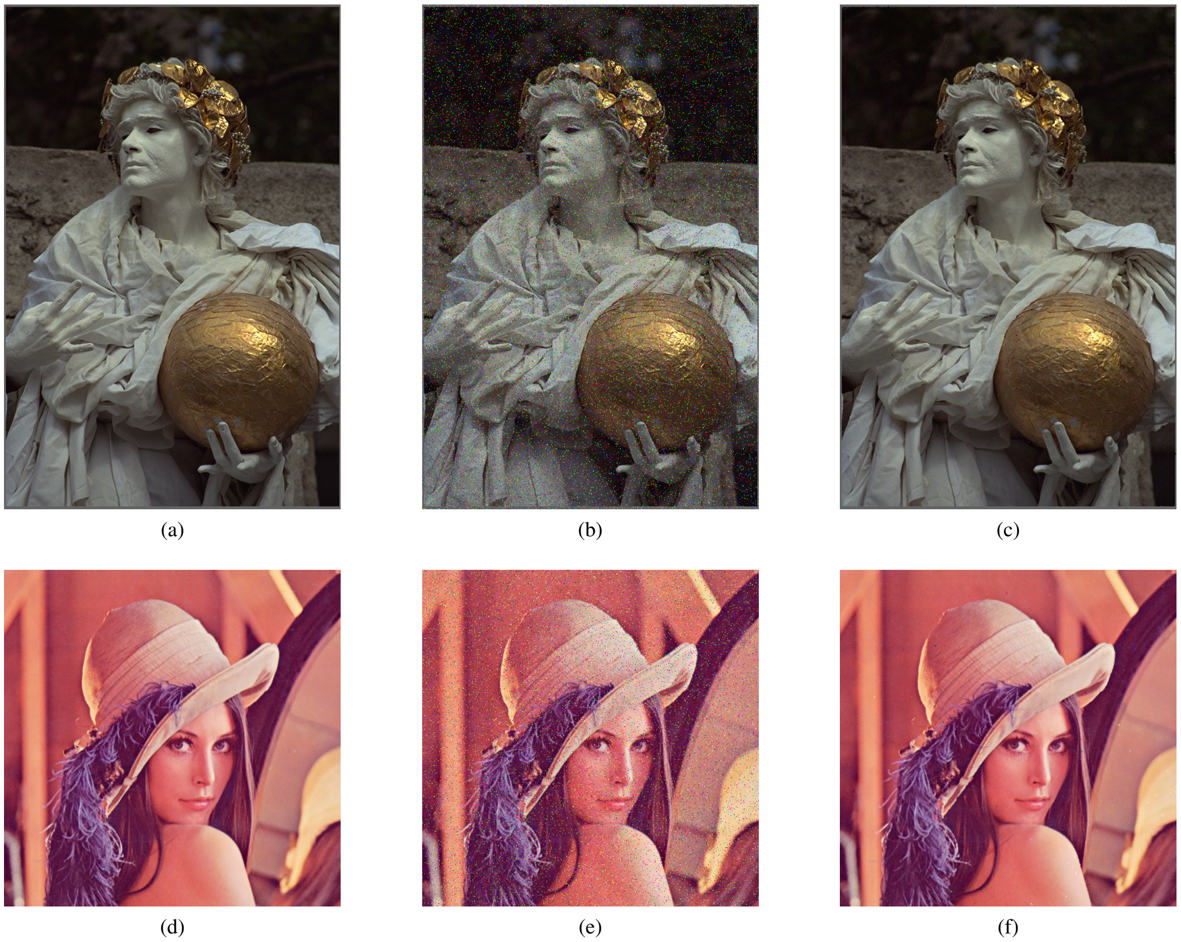


Fig. 6: Filter outputs for visual comparison: (a) Statue image, (b) image corrupted with $p = 0.05$ impulse noise, (c) filter ouput, (d) Lenna image, (e) image corrupted with $p = 0.05$, and (f) filter ouput.

Parallel Insertion Merge

Fernando Belmiro do Couto[1] **and Fabio Silva do Couto**[2]
[1]Divisao de Ti, BBDTVM, Rio de Janeiro, RJ, Brasil
[2]UFRJ, Rio de Janeiro, RJ, Brasil
PDPTA

Abstract – Parallel Insertion *Merge is an algorithm that maps the subsets, ordering them with merge and insertion routines and using parallel processing. Most sort algorithms in use nowadays treat the mass of data without previously analyzing its distribution, no matter if dealing with partially ordered datasets. According to the methodology applied in this algorithm the whole data is in the worst case ordered each two elements. According to the same method we prove that at random distribution, are statistically distributed among subsets of 2 + 3 elements. The major parallel routine order two subsets using X threads in the lowest subset, dividing this subset in X parts, and searching the position of the edge elements in highest subset. The next step starts without consider the edge elements used and if they point to the same target, not using the elements between them, because all those elements have their positions assigned.*

1. Introduction

Merge is the fastest way to group ordered lists of data, and binary search is the fastest way to seek one element position in a list. We can increase the speed using multithreads and if we have large ordered lists.

Merge, insertion, multithreads and ordered lists are the focus of this algorithm. This is not a insertion sort algorithm variant, since insertion sort use one element each time and this algorithm often insert blocks of elements, taking advantage of the data distribution. *This algorithm tends to use the multiprocessing capability of current computers and is able to adapt itself to increasingly coprocessors quantity. The algorithm first scans a sequence of N elements, comparing each element with the next, verifying if they are ordered according to a previously established criterion and gathering them on positive or negative value subsets if they obey or not that criterion respectively.*

There is order in chaos it depends the way we look, in the first task of the algorithm we will search for order.

2. Mapping Data

The algorithm first scans a sequence of N elements, comparing each element with the next, verifying if they are ordered according to a previously established criterion and gathering them on positive or negative value groups if they obey or not that criterion respectively. At the end of this process, if the distribution is ordered, we have only one index which get a positive N value. On the other hand, if the distribution is inversely ordered, this index will get negative value (Fig. 1) Assuming other hypothesis, index will range from 2 to (N / 2) +1 in proportion to total data. On average we will have 2N/5 index. To improve speed we divide the mass of data into fixed sized pieces and implement this in several threads adding the last index to the next index in the next piece according to the value. The last element of a piece is the first element of the next piece.

2	4	6	3	1	9	8	5	7	10	SEQ1
3			-2		-3			2		Index

1	2	3	4	5	6	7	8	9	10	SEQ1
10										Index

10	9	8	7	6	5	4	3	2	1	SEQ1
-10										Index

Figure 1.

3. Worst Case

For any distribution the worst case for implementation of the algorithm to initial 4 elements is described as shown in Table I:

TABLE I.

INITIAL 4 ELEMENTS {A,B,C, D…}				
A < B	***B < C***	***C < D***	***Index(0)***	***Index(1)***
TRUE	FALSE	TRUE	2	2
TRUE	FALSE	FALSE	2	-2
FALSE	TRUE	FALSE	-2	-2
FALSE	TRUE	TRUE	-2	2

The worst case is the way mergesort begins sort.

4. Average Case

As we see, the worst case splits the distribution into two ordered elements groups. We can then infer that on a random distribution the probability to occur 3 ordered elements is 50%. Therefore a minimal elements group to define a random distribution would be 2+3 elements type. This feature is advantageous to mass ordering against Merge

Sort algorithm, which in the beginning divide the distribution at each two elements, sort then and merge the resulting groups until the end. We realize 100,000 Monte Carlo Simulations using 100,000 randomized integers and always obtaining 41.318 %, near 40% expected for phenomenon 2 + 3.

5. Optimal Order

Grouping data according to the smallest group size optimizes non parallel merge routines performance. Nevertheless with multiprocessing the major factor to time optimization is the number of threads on simultaneously work. This algorithm always uses the lowest subset to choose elements that will seek their position in the greatest subset and gathering their own final position. When two or more elements point to the same insertion place the elements between then have their final positions assigned too.

6. Parallel Insertion Merge

The major algorithm routine order two subsets using X threads in the lowest subset (B), dividing this subset in X parts, and searching the position of the edge elements in the greatest subset (A) using Binary Search. At each step the edge elements of subset B have their position assigned and the block of elements between them when two or more elements have the same target (Fig. 2). When the first element of subset B is greater than some elements of subset A, those elements of A have their position assigned (Fig. 3). In the other hand when the last element of B is less than some elements of A those elements are positioned too(Fig. 4). As we can see in some situations we don't need to compare the elements between threads pointers with same target or when outers pointers having targets inside the greater subset.

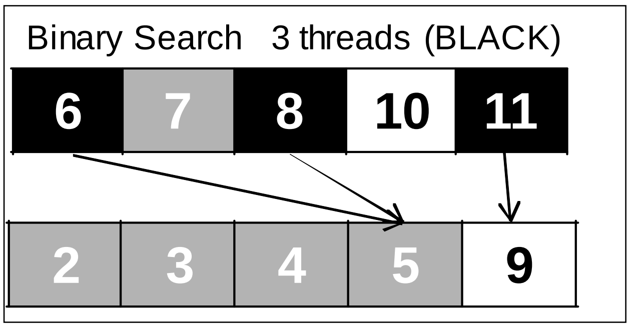


Figure 2

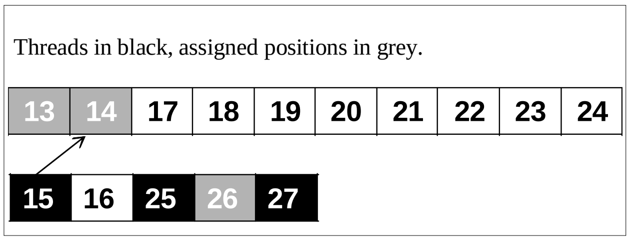


Figure 3

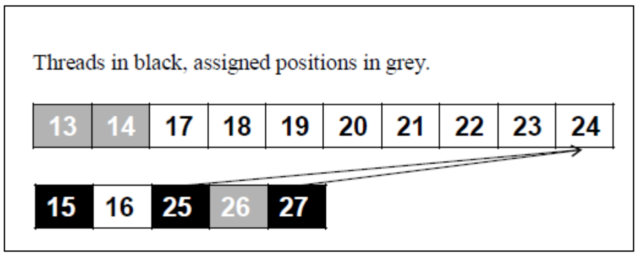


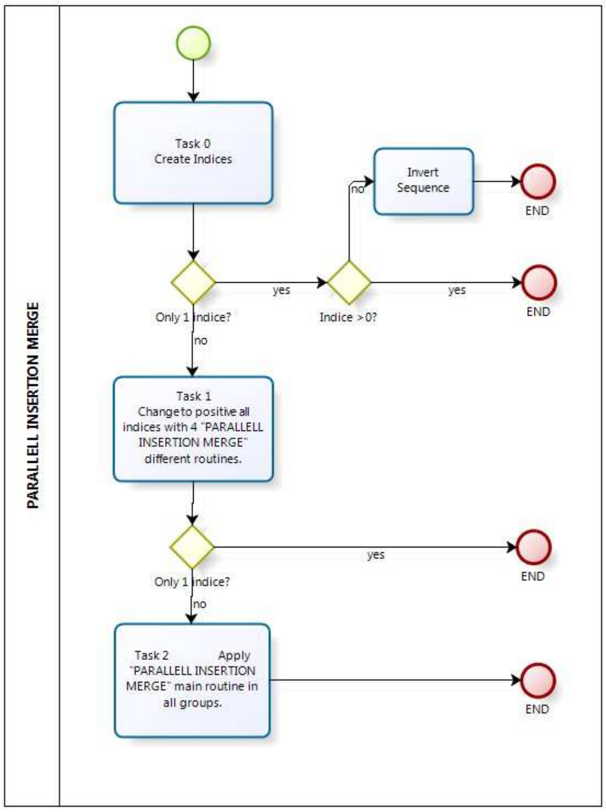
Figure 4

7. Data Conflicts

We have no problems with data conflicts because threads have access to the same element in read routines only, and when threads are writing they set the element in its final position.

8. The Algorithm

The algorithm is composed basically by 3 major Tasks:


PARALLELL INSERTION MERGE
Task 0
Create Indices
Only 1 indice?
yes
Indice > 0?
no
Invert Sequence
END
yes
END
no
Task 1
Change to positive all indices with 4 "PARALLELL INSERTION MERGE" different routines.
Only 1 indice?
yes
END
no
Task 2 Apply "PARALLELL INSERTION MERGE" main routine in all groups.
END

The function of Task0 is to identify in the sequence, ordered and inversely ordered groups. To improve this task the mass of data is divided into groups of N/m +1 elements and parallel threads identify indexes. If they are ordered or inversely ordered the next task are not started and the process is redirected to a conclusion.

If the sequence is not ordered Task1 starts. The function of Task1 is to get each group of subsets and call one of 4 recursive routines to merge according ordered subsets or inversely ordered subsets into one list. This routine takes the initial mapping to optimize the sorting, because if the sign of the index of adjacent groups are opposing, a position already be set (Table II).

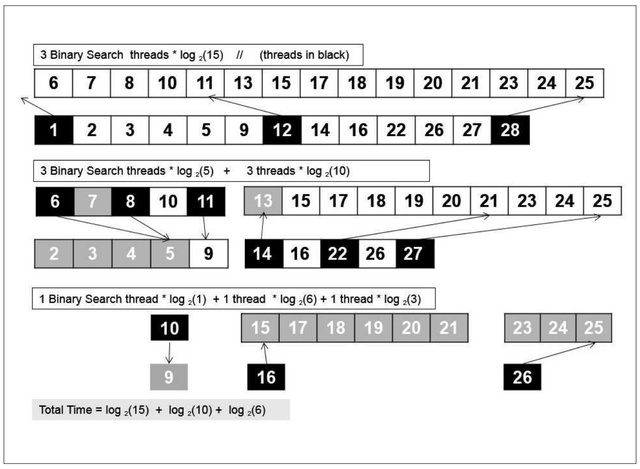
4 ELEMENTS {…,A,B,C, D…}					
A < B	***B < C***	***C < D***	***Index (6)***	***Index(7)***	***Position***
TRUE	FALSE	TRUE	2	2	
TRUE	**FALSE**	FALSE	2	-2	B greater
FALSE	TRUE	FALSE	-2	-2	
FALSE	TRUE	TRUE	-2	2	B minor

Table II

If we have more than 1 indices Task 2 starts. The function of Task 2 is to apply Insertion Merge in each remaining group.

The algorithm terminates when Task2 have only one index and its value is equal to N.

The definition of the maximum number of simultaneously threads depending on the characteristics of the hardware. In this paper we choose 3 to explain, but hundreds of threads may be used to do the job.


3 Binary Search threads * log $_2$(15) // (threads in black)
6 7 8 10 11 13 15 17 18 19 20 21 23 24 25
1 2 3 4 5 9 12 14 16 22 26 27 28
3 Binary Search threads * log $_2$(5) + 3 threads * log $_2$(10)
6 7 8 10 11
13 15 17 18 19 20 21 23 24 25
2 3 4 5 9
14 16 22 26 27
1 Binary Search thread * log $_2$(1) + 1 thread * log $_2$(6) + 1 thread * log $_2$(3)
10
15 17 18 19 20 21
23 24 25
9
16
26
Total Time = log $_2$(15) + log $_2$(10) + log $_2$(6)

9. Conclusion

The purpose of this algorithm is:

- identify pre-existing organizations in the mass of data;
- seek the smallest number of iterations for the mass of data;
- distribute the problem in order to fully utilize the processing hardware capabilities;
- consume the smallest possible space allocation,

to achieve the solution of the problem in less time.

I hope this text will contribute to the improvement of the processes of sorting (using insertion and merging) and help fellow developers around the world.

10. Acknowledgments

To our families.

11. References

[1] CORMEN, Thomas H., Introduction to Algorithms, Second Edition, McGraw-Hill Book Company 2001.

[2] MIT 6.172 Lecture 13 Analysis of multithreaded algorithms http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-172-performance-engineering-of-software-systems-fall-2009/lecture-notes/MIT6_172F09_lec13.pdf

[3] SZWARCFITER, Jaime Luiz; MARKENZONI, Lilian, (1994) Estrutura de dados e seus algoritmos, Editora LTC 2ª Edição.

[4] BUCKNALL, Julian; Algoritmos e estruturas de dados com Delphi, Editora Berkeley.

[5] CUDA,GPU_COMPUTING, Https://developer.nvidia.com/resources

Maximizing Hardware Performance via Non-blocking Collective Communication for All Pairs Shortest Paths Computation on Heterogeneous Multi-core Processors

Eduardo A. Colmenares[1], **Yu Zhuang**[1]
[1]Department of Computer Science, Texas Tech University, Lubbock, Texas, USA

Abstract - *Dynamic data sharing among cores during computation in a multi-core architectural environment has been recognized as one of the factors that add to the cost of the total execution time. One way of reducing the impact of the latency generated by intense data sharing is through communication hiding.*
The idea behind communication hiding is to create opportunities in the computation algorithms that can make use of system and/or hardware resources that allow the processors to engage in useful work while the sharing of information is taking place. One of the possible ways to achieve overlapping of communication and computation is by making use of non-blocking resources. In this paper, we present an algorithmically restructured solution for the All Pairs Shortest Path Problem that makes use of non-blocking features supported by a heterogeneous multi-core architecture. We compare our restructured approach against the traditional blocking approach in order to evaluate its potential.

Keywords: non-blocking, multi-core, all pairs shortest path, communication-computation overlapping.

1 Introduction

In today's world, more scientists are not only making use of but also considering high performance computing one of the leading tools that can help them to find solutions to computationally intensive problems that will shape not only our present but also our future [3].

Most of the possible solutions to realistic scientific and engineering problems that can be achieved through high performance computing are not embarrassingly parallel solutions which require little or no communication among processors. Many of the possible parallel solutions to our current scientific problems require a good percentage of interaction among participant processors which translates into communication cost and hence an increase in the total execution time [3,4].

According to a recent DOE report, communication hiding is identified as one of the fundamental algorithmic research areas over the next decade to harness computing power of extreme scale machines. Non-blocking collectives could be one of the ways to achieve it [5]. The use of Non-blocking collectives is a very resourceful approach that leads to non-blocking algorithms that require either partial or full restructuring in order to achieve better performance than their corresponding existing blocking approaches [5].

Following the principle explained above and after studying the traditional parallel approach of the All Pairs Shortest Path Problem, we present a new and restructured parallel implementation of the All Pairs Shortest Path Problem that makes extensive use of the non-blocking features of a multi-core heterogeneous system in order to minimize the effects of the intense data sharing among processors and target better performance than the traditional approach..

2 Floyd's Algorithm

First, we introduce some terminology used throughout the remainder of the paper. Floyd's algorithm is a well known algorithm used to find the shortest path between all pairs of nodes in a graph. Its applications include maps, networks and subroutine for other algorithms. A graph G can be represented as G = (V, E); where the cost of going from vertex i to vertex j, E[i,j], is always positive. V={1,….,n} are the vertices of G. Floyd's algorithm computes the following matrices.

$$D_{i,j}^{(0)} = \begin{Bmatrix} 0 \rightarrow i = j; (diagonal) \\ E[i,j] \rightarrow i \neq j \end{Bmatrix}$$

$$d_{i,j}^{(k)} = \min\left\{ d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)} \right\} \quad (1)$$

Where $d_{i,j}^{(k)}$ means the shortest path from i to j that does not go through any vertex bigger than k. Floyd's main objective is to compute $D_{i,j}^{(n)}$ [2, 7, 12].

2.1 Sequential Approach

According to [7], the sequential Floyd's algorithm can be expressed as follows:

```
1. procedure FLOYD_ALL_PAIRS_SP(E)
2. begin
3.   $D^{(0)} = E$;
4.   for k := 1 to n do
5.     for i := 1 to n do
6.       for j := 1 to n do
7.         $d_{i,j}^{(k)} = \min\left\{d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)}\right\};$   (1)
8. end FLOYD_ALL_PAIRS_SP
```

Figure 1: Sequential Floyd's Algorithm

2.2 Traditional Row-Wise Parallel Version

A number of different approaches to implement a parallel Floyd's algorithm can be found in the literature [1, 2, 7]. Floyd's Algorithm using a row-wise distribution will be presented on this section and is based on the work of [7]. It is important to mention that this approach does not attempt to use overlapping of communication and computation, and it corresponds to what can be viewed as a standard traditional parallel implementation of Floyd's. Therefore, this approach will be referred to as the Traditional All Pairs Shortest Path or TAPSP approach for the remainder of this paper.

The TAPSP implementation is based on a conceptual topology. This conceptual topology is a one-dimensional row-wise decomposition. The $P_{i,1}$ nomenclature is introduced to facilitate the reference of a particular process in the conceptual topology when required. Variable *i* references the row number and $P_{i,1}$ references the process in the *i*th row and $D^{(0)}$ is the adjacency matrix.

```
1. procedure FLOYD_ROW_WISE($D^{(0)}$)
2. begin
3.   for k := 1 to n do
4.   begin
5.     each process in the $P_{i,1}$ decomposition ask
       themselves if they own a segment of the k^th row of
       $D^{(k-1)}$ . The one that owns it (Manager); broadcasts
       it to the $P_{i,1}$ processes (workers);
6.     each process waits to receive the needed segments
       (workers);
7.     each process $P_{i,1}$ computes its part of the $D^{(k)}$ matrix
       (All);
8.   end
9. end FLOYD_ROW_WISE
```

Figure 2: TAPSP Algorithm for Floyd's Parallel Formulation Using a Row-Wise Approach.

Step 7 of figure 2 computes the value of $d_{i,j}^{(k)}$ using equation 1, see figure 1. The distribution of matrix $D^{(k)}$ among *p* processes in some cases will create situations where some members of this computation will not be located in the local process; instead, these relevant values will be located in other processes. In order to complete the computation of $d_{i,j}^{(k)}$ for the current value of *k*, local process $P_{i,1}$ must receive those relevant segments of the *k*th row of $D^{(k-1)}$ matrix.

In this version, it is clear that all iterations are highly synchronized. For example, the next iteration (*k+1*) will not begin for all participant processors until the *k*th iteration has been completed at all processors, and the segment of interest of the *k*th iteration of the *k*th row have been received by all processors.

2.3 Description of our algorithm

This algorithm makes extensive use of the non-blocking hardware supported features of a multi-core processor system to minimize the cost of data communication among participant processors.

Each Process initializes the submatrix of $D^{(0)}$ with the submatrix of A.

Step 0 (*Manager*): Processor $P_{i,1}$ (i=1,…,p) broadcast the segment of the 1st row of $D^{(0)}$ to processes $P_{i,1}$ for all $i\neq1$; and processes $P_{i,1}$ for all $i\neq1$ receive the segment of the 1st row of $D^{(0)}$.

For k starting from 1 through (n)

> Step 1 (*Manager*). Each Processor $P_{i,1}$ owning a segment of (k+1)th row of $D^{(k)}$ (Manager) computes entries on the (k+1)th row of $D^{(k)}$ as indicated by equation 1 of figure 1, and signal the other processors (workers) so that they can retrieve those just computed entries from the manager.
>
> Step 2 (*Workers*). Each Processor $P_{i,1}$ owning no segment of (k+1)th row of $D^{(k)}$ compute uncomputed entries of $D^{(k)}$ using equation 1; while receiving entries on the (k+1)th row of $D^{(k)}$ copied from the processor owning them.
>
> Step 3 (*Manager*). The processor owning a segment of (k+1)th row compute uncomputed entries using equation 1 of figure 1.

end for

Figure 3: PAPSP Algorithm for Floyd's Parallel Formulation Using a Row-Wise Approach.

The algorithm is based on a one-dimensional partition called row-wise, which is basically a one-column multiple row distribution of the processors.

This algorithm is derived based on two critical observations. First, if a process owns $D^{(k)}$ samples that are required by any other processes during the following iteration of k, then the processor owning the samples does not have to compute those $D^{(k)}$ samples right after computing any other samples of $D^{(k)}$ [13]. This is possible because no order (data dependency) is imposed by Floyd's algorithm on the computation of different samples of $D^{(k)}$ for each possible value of k [13].

Second, a processor owning $D^{(k)}$ samples that are required by any other processes during the following iteration of k does not have to send or allow to copy those $D^{(k)}$ samples immediately before other processes need them for computing $D^{(k+1)}$ samples [13].

These two critical observations are the base of the strategy to be followed during the k^{th} iteration. This strategy is divided into two sub-strategies; computation-reordering and early copy.

3 Architectural Restrictions and Its Influence in Our Design

The multi-core heterogeneous system that we chose for this study imposed some important architectural restrictions that led to the chosen parallel implementation of both, traditional and proposed parallel approaches.

The first restriction influenced the logical distribution of our processors. The traditional two-dimensional grid style distribution of processors is only possible when the number of participant processors has an exact square root value [7]. Our heterogeneous multi-core architecture offered a maximum number of six usable cores [9]. Six cores is acceptable to exploit hardware level parallelism but we also wanted to make use of a second level of parallelism, instruction level parallelism, which was offered by the Single Instruction Multiple Data (SIMD) capabilities of each one of the cores of our multi-core system. SIMD are easily coded and used if the data follows a row-wise distribution.

The second restriction was the maximum size of the only memory that each one of the cores had direct access to, which is only 256 KB [6, 9, 15]. This posed a major constraint regarding the sizes of the problem to be analyzed. In order to circumvent this limitation, we fixed the number of rows of our matrix to 60 while the numbers of columns were 1024, 2048 and 4096 unsigned shorts. For the reasons expressed above, a row-wise distribution was the clear choice for the logical distribution of the processors.

4 Experimental Evaluation

In this section we demonstrate the performance of our approach (PAPSP) regarding the traditional approach (TAPSP). For performance evaluation of both approaches we use a Cell Broadband Engine processor and an IBM Full System Simulator [14]. This heterogeneous multi-core architecture consists of one 64 bit Power architecture element in charge of running the operating system (PPE) and eight Synergistic Processing Elements (SPE's). The SPE is composed mainly by two elements, the Synergistic Processing Unit (SPU) and the Memory Flow Controller (MFC). The SPU's are capable of performing high speed Single Instruction Multiple Data (SIMD) operations and have direct access to 256 KB of local memory known as the Local Store (LS) [6]. The LS is where each one of the SPU's stores its data and instructions [9]. We chose this architecture as each one of the six usable cores of this processor have hardware support for Direct Memory Access (DMA), which is managed by a Memory Flow Controller (MFC). The MFC allows the Synergetic Processing Unit (SPU) to focus exclusively on computation while non-blocking communication with its neighbors takes place [9, 10, 15].

For both of our parallel approaches, the PPE does not participate in any computation; it will only host two matrices, the one used for initialization and its solution. This strategic decision was made as the DMA between SPE's and PPE is too costly when compared to SPE-SPE DMA [6, 9]. We tested three problem sizes which we will referred to by using the following nomenclature, matrix(n, r, c) where n={2,3,4,5,6 spe's}, r={60 rows} and c={1024, 2048, 4096 unsigned shorts}. We chose r to be divisible by six as the hardware imposes a restriction on the maximum number of usable SPE's. C was chosen to be exactly divisible by eight in order to make use of SIMD instructions. Figure 4 provides a visual perspective of the testing conditions.

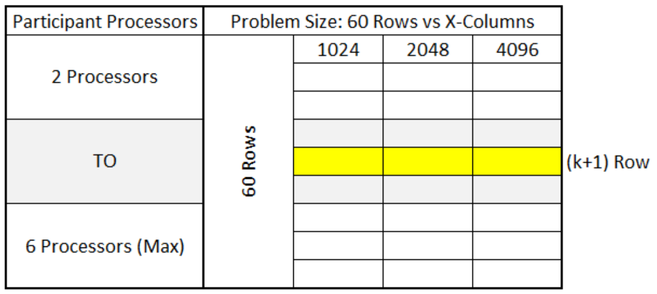


Figure 4: Testing Scenarios.

4.1 Sequential Approach

Implemented as shown on Figure 1. This approach was executed and timed on the PPE. We made use of the __mftb() command which counts the number of ticks required by a routine, we also made use of gettimeofday( ) to validate its accuracy and double check for overflow [9].

We considered two reasons why the sequential approach was not timed on the SPE. The first reason is that the largest problem size of 60 rows and 4096 columns does not fit in the LS of a SPE. The second reason is that although they are different hardware, both PPE and SPE's work at the same frequency of 3.2 GHz [9]. This approach does not make use of any SIMD instruction and it was used for our speedup and efficiency analysis.

4.2 Experimental Results

Table 1: Testing Calculations for serial algorithm, TAPSP and PAPSP on one CBE for multiple problem sizes.

Problem Size 60 Rows by x Columns	Serial Algorithm at PPE Time (usecs)	Parallel Algorithms –Execution Times (usecs)									
		Number of Participant SPE's									
		2		3		4		5		6	
		TAPSP	PAPSP	TAPSP	PAPSP	TAPSP	PAPSP	TAPSP	PAPSP	TAPSP	PAPSP
1024	26897.32	1692.98	1647.33	1234.85	1200.36	1031.22	996.36	927.43	881.96	878.45	831.51
2048	54083.96	3339.49	3154.66	2291.80	2199.32	1849.24	1719.85	1613.80	1474.26	1478.41	1322.59
4096	114570.4	6326.42	-	4382.70	4211.02	3464.63	3269.61	2932.19	2707.86	2618.67	2367.30

Initially, we implemented our routines on one multi-core heterogeneous system and then we timed each one of the cases on the IBM Full-System Simulator. Table 1 includes all tested cases for the Traditional case of the All Pairs Shortest Path (TAPSP) and the Proposed All Pairs Shortest Path approach (PAPSP). All the execution times presented on table 1 correspond to the average of the total time of all participant processors. The total execution time of each core was measured using the decrementer which works in a similar fashion as the __mftb( ) used to time the sequential algorithm on the PPE [9].

Table 1 presents all the quantitative results for both, the traditional and our proposed parallel approaches. By comparing the execution time of the PAPSP and TASPS, we can notice that for each one of the problem sizes and any number of the participant processors, our PAPSP approach performs faster than the TAPSP approach, this is supported by table 2. Execution time it is the easily recorded metric of performance [12].

Table 2 provides the percentage of improvement achieved by our PAPSP regarding the TAPSP. This percentage of improvement was computed as $100-\left\{\frac{Time_{PAPSP}\times 100}{Time_{APSP}}\right\}$. For none of the testing cases the percentage of improvement was negative, this means that our approach performed better than the traditional approach. From table 2, it is also possible to infer that the optimal size for our testing was 60x2048, which is the case with the higher percentage of improvement.

Table 2: Percentage of Improvement for PAPSP regarding TAPSP for multiple problem sizes and participant SPE's.

Problem Size 60 Rows by x columns	PAPSP vs TAPSP [% of Improvement]				
	Number of Participant SPE's				
	2	3	4	5	6
	%	%	%	%	%
1024	2.70	2.79	3.38	4.90	5.34
2048	5.53	4.04	7.00	8.65	10.54
4096	-	3.92	5.63	7.65	9.60

Table 3 present the achieved speedup and efficiency for our proposed approach. *Efficiency is not in percentage notation*. It can be concluded from Table 3 that along with the increase of participant cores, the speedup rate becomes higher and higher. It is also possible to observe that as the problem grows in size and more participant cores intervene in the solutions of the chosen matrix, the efficiency becomes also higher. Both claims are not only supported by the results shown on table 3 but also figures 5 and Figure 6.

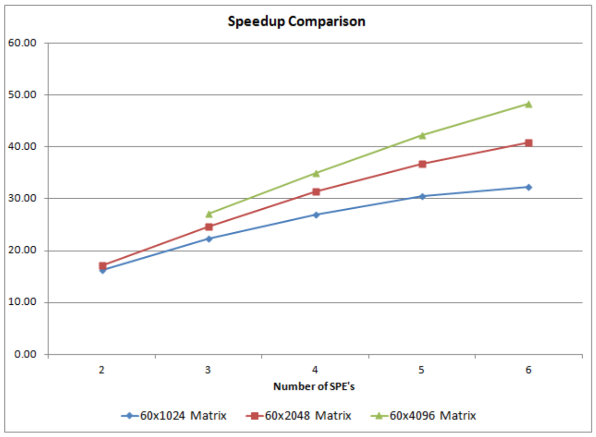


Figure 5: Speedup for the PAPSP algorithm for multiple problem sizes and SPE's.

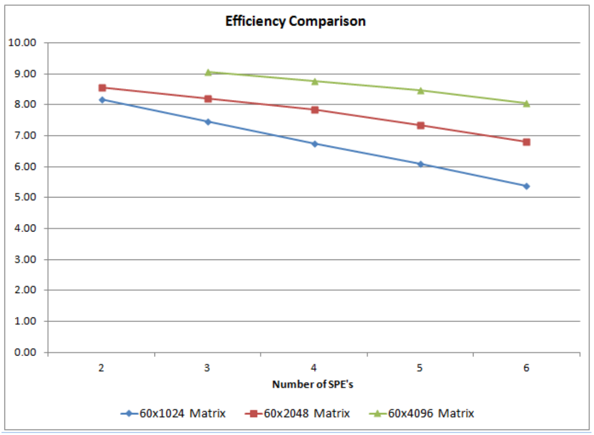


Figure 6: Efficiency for the PAPSP algorithm for multiple problem sizes and SPE's.

Table 3: Speedup and Efficiency for the PAPSP algorithm for multiple problem sizes and participant SPE's.

Problem Size 60 Rows by x columns	PAPSP Parallel Algorithm									
	Number of Participant SPE's									
	2		3		4		5		6	
	Speedup	Efficiency	Speedup	Efficiency	Speedup	Efficiency	Speedup	Efficiency	Speedup	Efficiency
1024	16.33	8.16	22.41	7.47	27.00	6.75	30.50	6.10	32.35	5.39
2048	17.14	8.57	24.59	8.20	31.45	7.86	36.69	7.34	40.89	6.82
4096	-	-	27.21	9.07	35.04	8.76	42.31	8.46	48.40	8.07

5 Conclusions

Non-blocking collectives help to reduce the cost of communication-intensive applications, and its use should be considered if the underlying hardware supports it. The use of non-blocking algorithms, as well as, non-blocking collectives has an implicit design cost that requires, in several cases, a restructuring or complete redesign of traditional parallel solutions. Experimental results have demonstrated that our proposed approach not only satisfied the principles of non-blocking algorithms but also decreased the total execution time by reducing the cost of collective data sharing while achieving a better efficiency when compared to the traditional row-wise approach.

6 References

[1] A. Grama, G. Karypis, V. Kumar, A. Gupta., *Introduction to Parallel Computing*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc, 2nd ed, 2002.

[2] Diament, B. and Ferencz, A. (1999, May). "Comparison of Parallel APSP Algorithms". Available: http://citeseer.ist.psu.edu/334881.html [Oct 08, 2013]

[3] Guvvala, Y. R. and Zhuang, Y. "Communication Delegation Method for Exascale Systems," in *Proc. AusPDC*, 2014, ACS, pp 51-54.

[4] Hoefler, T.; Lumsdaine, Andrew; Rehm, Wolfgang, "Implementation and performance analysis of non-blocking collective operations for MPI," *in proc. SC 2007*, pp. 1, 16.

[5] J. Ang, K. Evans, A. Geist, M. Heroux, P. Hovland, O. Marques, L. McInnes, E. Ng, and S. Wild, "Report on the workshop on extreme-scale solvers: Transitions to future architectures". Office of Advanced Scientific Computing Research, U.S. Department of Energy, 2012. Washington, DC, March 8-9, 2012.

[6] Kistler, M.; Perrone, M.; Petrini, F., "Cell Multiprocessor Communication Network: Built for Speed," *Micro, IEEE*, vol.26, no.3, pp.10,23, May-June 2006.

[7] Vipin Kumar and Vineet Singh. "Scalability of parallel algorithms for the all-pairs shortest-path problem." *Journal of Parallel and Distributing Computing*, vol 13, pp. 124-138, Oct. 1991.

[8] Khunjush, F.; Dimopoulos, N.J., "Extended characterization of DMA transfers on the Cell BE processor," *Parallel and Distributed Processing*, 2008. IPDPS, 2008, pp.1,8, 14-18.

[9] Scarpino M., *Programming the Cell Processor: For Games, Graphics, and Computation*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1 ed, 2009.

[10] SARC European Project, "Parallel Programming Models for Heterogeneous Multicore Architectures." *IEEE Micro* 30.5 (2010): 42-53.

[11] Saif, T. , and M. Parashar. "Understanding The Behavior and Performance of Non-blocking Communications in MPI". *10th International Euro-Par Conference*, 2004, pp.173-182.

[12] Vinjamuri, S.; Prasanna, V.K., "Transitive closure on the cell broadband engine: A study on self-scheduling in a multicore processor," *IPDPS*. 2009, pp.1,11, 23-29.

[13] Colmenares E., Andersen P, Y. Zhuang. "Overlapping Communication and Computation with MPI-2 for Floyd's Algorithm", MSCS Thesis, Texas Tech University, 2008.

[14] IBM Full-System Simulator User's Guide. Available: https://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/B494BF3165274F67002573530070049B/$file/SystemSim.Users.Guide.pdf [Sep 12, 2013]

[15] Programming the Cell Broadband Engine Architecture. Available: http://www.redbooks.ibm.com/redbooks/pdfs/sg247575.pdf [Sep 03, 2013]

How to find the best rules for CA-based PRNG

Miroslaw Szaban
Institute of Computer Science
Siedlce University
Poland
E-mail: mszaban@uph.edu.pl

***Abstract*—In this paper is consider a problem of generation by cellular automata of high quality pseudorandom sequences useful in cryptography. For this purpose one dimensional nonuniform cellular automata (1D CA) is used. The quality of pseudorandom bit sequences generated by cellular automata depends on collective behavior of rules assigned to cellular automata cells. In the paper is presented new method of constructing CA rules, based on balance of the CA and CA rule. The paper gives an answer how to find rules, which do not generate stable sequences, passes cryptographical tests (provide high quality pseudorandom sequences) and show maximal as possible set of such CA rules. So, presented set of rules is the best for use in 1D CA-based Pseudorandom Number Generator (PRNG). Sequences generated by these PRNB are suitable for symmetric key cryptography and can be used in different cryptographic algorithms.**

***Index Terms*—Cellular Automata, Pseudorandom Number Generator, Balance, Boolean Function, Cryptography.**

I. Introduction

Increasing need for safety and privacy of digital information in many areas can be observed today. Public and private organizations have become increasingly dependent on cryptographic techniques by growth of digital information storage and transmission through global networks. However, digital resources are still not safe from attacks. Use of digital tools in communication, data exchange, fast development of electronic commerce transactions and increasing use of digital signatures quickly causes the creation of new generations of secure mechanisms. Cryptography techniques are an essential component of any secure communication. Nowadays two main cryptography systems are used: secret and public-key systems. An extensive overview of currently known or emerging cryptography techniques used in both type of systems can be found, e.g. in [10]. One of such promising for cryptography technique is application of cellular automata (CA).

CA were proposed for public-key cryptosystems by [2] and [6]. Such systems requires two types of key: one key for encryption and the other one for decryption. One of them is held in private, the other rendered public. However, the main concern of this paper are cryptosystems with a secret key. In such systems the encryption and the decryption key are the same. The encryption process is in particular based on generation of pseudorandom bit sequences, and CA can be effectively used for this purpose. CA for systems with a secrete key were first studied by Wolfram [17], who proposed for 1D CA-based PRNG with rule 30, and later by Habutsu et al. [4], and Nandi et al. [8], who proposed rules 90 and 150, and also Gutowitz [3]. After that, this subject was studied by Tomassini et al. [15], [16], where the set of rules was enlarged to rules: 90, 105, 150, 165. Next in paper [11] authors have considered one or two dimensional (2D) CA for encryption. This paper is an extension of these recent studies and concerns an application of one dimensional (1D) CA for the secret key cryptography. Authors presented new larger set of rules {86, 90, 101, 105, 150, 153, 165, 1436194405}, discovered with use of evolutionary technique called cellular programming (CP) and concerning CA rules suitable for cryptography, presented in earlier papers. These set of rules consists of rules with neighbourhood radius equal to 1 and 2 (last rule in set), and gives similar results in the sense of passing tests like: entropy test and FIPS 140-2 (standard tests for basic analysis the quality of PRNG's), but offered larger space of keys (different bit sequences) than previous proposals. Key space for proposals in [11] is $8^N * 2^N$, where in [15] and [16] key space was $4^N * 2^N$, in [8] $2^N * 2^N$ and in [17] $N * 2^N$.

The paper is organized as follows. The next section presents the idea of an encryption process based on Vernam cipher. The main concepts of CA are outlined in section 3. Section 4 describes the statement of the problem. In section 5 are described construction of CA-based PRNG and new method of suitable CA rules selection. Last section concludes the paper.

II. Symmetric Key Cryptography and Vernam Cipher

The main idea of cryptography using a symmetric key is that both sides of cryptographic process apply the same key to encrypt and decrypt the message. The key is secret and most secure because only two persons can use it and other people can know only encrypted message, which is too difficult to encrypt without knowing the key. In our study we continue Vernam's approach to cryptography with the secret key.

Let P be a plain-text message consisting of m bits $(p_1 p_2 ... p_m)$ and $(k_1 k_2 ... k_m)$ is a bit stream of a key k. Let c_i be the $i - th$ bit of a cipher-text obtained by applying XOR (exclusive-or) enciphering operation:

$$c_i = p_i XOR k_i. \quad (1)$$

The original bit p_i of a message can be recovered by applying the same operation XOR on c_i (bit of a cipher-text) using the same bit stream key k:

$$p_i = c_i XOR k_i. \quad (2)$$

The enciphering algorithm called Vernam Cipher is known (see, [7], [10]) as perfectly safe if the key stream is truly unpredictable and used only one time. In this paper we give the answer to the questions: how to provide a pure randomness of a key bit stream and unpredictability of random bits, and how to obtain such key with a length large enough to encrypt practical amounts of data. We can apply CA to generate high quality PNSs and use them as the safe secret key. We will show that by using 1D CA, the quality of PNSs for secret key cryptography and the safety of the key can be increased.

III. Cellular Automata

1D CA is in the simplest case a collection of two-state elementary cells arranged in a lattice of the length N, and locally interacting in a discrete time t. For each cell i called a central cell, a neighborhood of a radius r is defined. The neighborhood consists of $n_i = 2r + 1$ cells, including the cell i. A cyclic boundary condition is applied to a finite size of CA, which results is in a circle grid. It is assumed that a state q_i^{t+1} of a cell i at the time $t+1$ depends only on states of its neighborhood at the time t, i.e.:

$$q_i^{t+1} = f(q_{i-r}^t, ..., q_{i-1}^t, q_i^t, q_{i+1}^t, ..., q_{i+r}^t), \qquad (3)$$

the transition function f is called a rule, defining the rule of updating the cell i. We will call all rules for CA with $r = 1$ the short rules and rules for CA with $r = 2$ the long rules. The length L of a rule and the number of neighborhood states for a binary CA is $L = 2^n$, where $n = n_i$ is a number of cells of a given neighborhood, and a number of such rules can be expressed as 2^L. For CA with e.g. $r = 2$ the length of the rule is equal to $L = 32$, and a number of such rules is 2^{32} and grows very fast with L.

CA can change states in time with use of one the same rule assigned to all CA cells and it is called an uniform CA. If two or more different rules are assigned to update cells, CA is called nonuniform CA. Wolfram system [17] was uniform, because he used 1D CA with r=1, and rule 30. Other mentioned systems were nonuniform. System proposed in [15] use 1D CA with $r = 1$ and four rules 90, 105, 150 and 165, which provide high quality PNSs and a huge space of possible secret keys which is difficult for cryptanalysis. They used CP to search these rules. Moreover, system proposed in [11] use also 1D, nonuniform CA, but in the system are used two sizes of rule neighborhoods, namely a neighborhood of radius $r = 1$ (86, 90, 101, 105, 150, 153, 165) and $r = 2$ (1436194405). Last earlier mentioned system [13], [14], use only rules with neighbourhood radius $r = 2$ with nonuniform CA, i.e.: 1436194405, 1436965290, 1721325161, 1704302169, 1705400746.

IV. Statement of the problem

Closer analysis of selected subset of rules conducted by Bouvry et al. [1] have shown that some specific assignments of these rules to CA cells leads to bad statistical quality of PNSs generated by CA (CA cells generate stable or particular stable sequences of 0s or 1s). Figure 1 shows space-time diagram

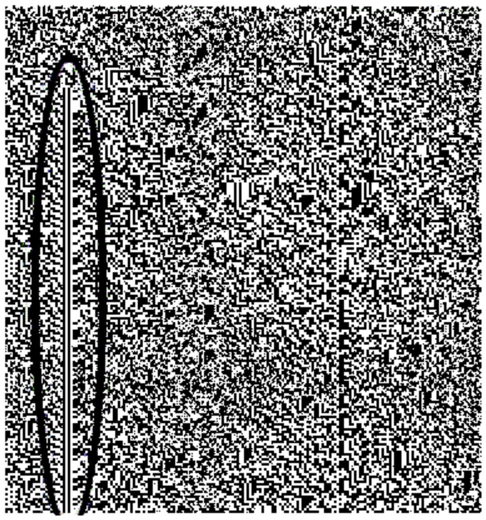
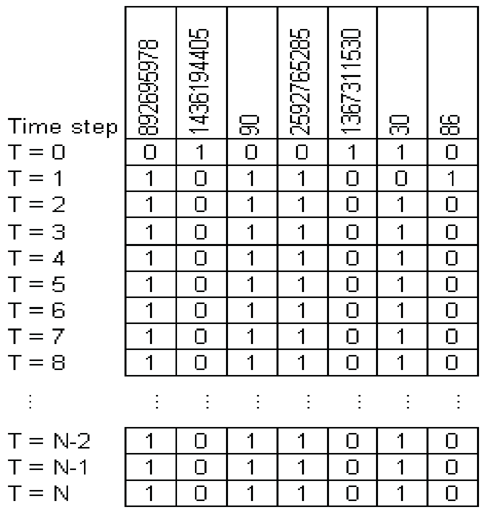


Fig. 1. Examples of bad assignments of rules to CA cells: problem described in [1] – stable vertical lines of 0s and 1s in space-time diagram



Time step	892695978	1436194405	90	2592765285	1367311530	30	86
T = 0	0	1	0	0	1	1	0
T = 1	1	0	1	1	0	0	1
T = 2	1	0	1	1	0	1	0
T = 3	1	0	1	1	0	1	0
T = 4	1	0	1	1	0	1	0
T = 5	1	0	1	1	0	1	0
T = 6	1	0	1	1	0	1	0
T = 7	1	0	1	1	0	1	0
T = 8	1	0	1	1	0	1	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
T = N-2	1	0	1	1	0	1	0
T = N-1	1	0	1	1	0	1	0
T = N	1	0	1	1	0	1	0

Fig. 2. Examples of bad assignments of rules to CA cells: specific assignment of 7 rules generate stable sequences of 0s and 1s

with stable sequences produced by set of CA rules from [11]. Figure 2 shows an example of some bad assignation of rules to CA. Circled 4 vertical lines present stable sequences of 1s, 0s and 1s, respectively. Figure 2 presents CA consisting of 7 cells to which 7 rules were assigned. One can see that starting from time step $T = 2$ all CA cells generate constant sequences 0s and 1s. So the purpose of work presented in the papers [13], [14] was to eliminate bad rules from the set of rules discovered earlier, and to find subsets of rules which will be suitable for cryptographic purposes, for any assignments of them into CA cells. Search of these sets of rules was performed by genetic algorithm (GA), with special constructed GA operators. Discovered new sets of rules presented in the papers [13], [14] are consisted of 5 rules {1436194405, 1436965290, 1721325161, 1704302169, 1705400746} only with neighbourhood radius equal to 2. These set of rules also passed mentioned earlier cryptographical tests. Despite the fact, that key space of these set of rules is $5^N * 2^N$, is smaller than in [11], new set did not generate stable bit sequences. Moreover, each of the subset of these rules is as good for PRNG as whole new set of 5 rules, what leads to conclusion that, when each of subsets would be applied for PRNG, that generated key space will be higher.

Despite the works and analyses on the application of 1D CA for PRNG presented before and also in [9], there is no definitely answer which rules are the best for use in 1D CA-based PRNG. In means that applied rules or sets of rules (a) do not generate stable sequences, (b) pass cryptographical tests, and (c) set of rules is maximal. So, this paper gives an answer how to select rules for 1D CA-based PRNG, satisfying (a), (b) and (c).

V. Construction 1D CA-based PRNG

A. Conception of 1D CA-based PRNG

Let's PRNG based on 1D CA will be a classical 1D CA with lattice of the length N, locally interacting in a discrete time t. A rule (rules) of the CA controlling cells are like described in equation (3). So, the seed of the generator consists of few elements: initial configuration of CA ($N(0)$) - at the time $t = 0$ ($0 \leq t \leq T$), set of CA rules, number of the cell (N_i) which generates bit sequence and number of time steps (T) - length of bit sequence. Nevertheless, CA-based PRNG should create cryptographically strong bit sequences independently to initial configuration $N(0)$, selected cell N_i and time steps T, than set of rules for managing the CA should be precisely selected. It is wide known that not every rules and set of rules are suitable for these purpose. Especially, some combination of rules do not cooperate with each other (see, [13], [14]), and generate stable bit sequences. To eliminate such kind of behaviour and also satisfy one of most important cryptographic criteria, the balance of the CA should be analyzed.

B. Balance of 1D CA-based PRNG

Balance (regularity) is another important criterion which should be fulfilled by a Boolean function used in ciphering (see, [18]). This means that each output bit (0 or 1) should appear in equally number of times for all possible values of inputs. For CA-based generator it means that independently on the initial configuration (except consisted of only 0s or only 1s), in each time step in CA state number of 0s and 1s should be equal ($\sum_{i=1}^{N} N_i = \frac{N}{2}$). It is one of the basic requirements and conditions measured by the tests qualified as a good quality generator. Balance of the CA could be satisfied by using balanced rules of CA. CA rule as a boolean function $f : Z_2^n \to Z_2$ maps n binary inputs (neighbourhood state of CA) to a single binary output. So, the balance of a Boolean function is measured using its Hamming Weight, and is defined as:

$$HW = \frac{1}{2}(2^n - \sum_{x \in B^n} \hat{f}(x)), \quad (4)$$

where $\hat{f}(x) = (-1)^{f(x)}$. Boolean function is balanced when its Hamming Weight is equal to 2^{n-1}. Balanced Boolean function is also named bijective Boolean function. It means that balanced rule has in binary form equal number of 0s and 1s. As we can see the rules described in [17], [8], [15], [11], [13], [14] are balanced both with $r = 1$ and $r = 2$. For single rule, as rule of CA-based PRNG, balance is enough criterion except formal tests, but for a set of rules it is insufficient.

C. Selection of set rules for the 1D CA-based PRNG

Set of rules which collectively change states of CA cells should be carefully selected. Some composition of rules to the CA cells could lead to unwonted stable sequences created by cells (see, Figure 1 and Figure 2). Those bad combination of rules should be eliminated by the appropriate selection of rules. To select the rules which will cooperate with other rules, the rule of CA should be analyzed as a Boolean function. We should analyze all inputs and outputs of CA rule. Composition of outputs for successive inputs (neighbourhood state) of the rule gives us the binary form of CA rule. As we know from previous subsections, proper rules are balanced, but from such rules we should select rules ready to cooperate with other rules in the set for CA-based PRNG.

Let's analyze CA rule with $r = 1$ as Boolean function.

TABLE I
CA rule as a Boolean function.

Input	111	110	101	100	011	010	001	000
Output	a	b	c	d	e	f	g	h

The letters $a, ..., h \in \{0, 1\}$ as a Boolean outputs. When $a + ... + h = 4$ then rule is a balanced one. For a good mixing bits, rule for CA-based PRNG should changing the same number of times: $0s \to 0s$, $0s \to 1s$, $1s \to 0s$ and $1s \to 1s$. Except these output 0s and 1s should be selected in $a, ..., h$ in special configuration. Bad schedule gives a rule easy to produce a stable sequences in cooperation with other rule. I.e.: rule 30 (see, [17]) as a single rule creates a sequences of a quite good quality, but in cooperation with rule 86 (see, Figure 2) creates stable sequences. Keep attention that both rules 30 and 86 are balanced. Mentioned problem in the consequence gives a bad fitness between inputs and outputs, i.e. bad selection of rules for set.

Selection of rules resistant for bad cooperation with other rules leads to appropriate composition of outputs. Lest analyze Table I. If for the inputs $\{111\}$ and $\{011\}$ outputs will be equal to 1, then obtained rule will be generate 1s from 1s for input configuration {X11} where X could be any kind of bit value. For other rule, if the inputs $\{111\}$ and $\{110\}$ outputs will be equal to 1, then obtained rule will be generate 1s from 1s for input configuration {11X}. Summarizing, for state of 1D CA {...X11X...} such rules will be cooperate and create stable sentences of 1s in successive number of time. Similar situation is presented in Figure 2 for rules 30 and 86, and for CA state {...X10X...}. To prevent these situation rule should fulfill logical sentence, based on designation form Table I:

$$(a \neq b) \wedge (a \neq e) \wedge (c \neq d) \wedge (c \neq g) \wedge \\ \wedge (f \neq b) \wedge (f \neq e) \wedge (h \neq d) \wedge (h \neq g). \quad (5)$$

In Equation 5: $(a \neq b)$ protect rule against input configuration {11X} translating $1s \rightarrow 1s$, which leading to generation stable sequence of 1s, $(a \neq e)$ similarly protect against input configuration {X11}, $(c \neq d)$ protect against input configuration {10X}, $(c \neq g)$ protect against input configuration {X01}, $(f \neq b)$ protect against input configuration {X10}, $(f \neq e)$ protect against input configuration {01X}, $(h \neq d)$ protect against input configuration {X00}, $(h \neq g)$ protect against input configuration {00X}.

Equation 5 could be simplified to the logical sentence:

$$(a = f) \wedge (b = e) \wedge (c = h) \wedge (d = g) \wedge (a \neq b) \wedge (c \neq d). \quad (6)$$

Rules created on the base of Equation 6 are resistant for bad cooperation with other rules, and never generate stable sequences of bits.

Lest us solve the Equation 6: Let $a = 0$ from here $f = 0$, $b = e = 1$, also let $c = 0$ from here $h = 0$, $d = g = 1$, then we obtain binary rule $(01011010)_2$ it means decimal rule 90_{10}. Let $a = 0 \wedge c = 1$ from here $b = e = h = 1$ and $d = f = g = 0$, then we obtain binary rule $(01101001)_2$ it means decimal rule 105_{10}. Let $a = 1 \wedge c = 0$ from here $b = e = h = 0$ and $d = f = g = 1$, then we obtain binary rule $(10010110)_2$ it means decimal rule 150_{10}. Let $a = 1 \wedge c = 1$ from here $f = h = 1$ and $b = d = e = g = 0$, then we obtain binary rule $(10100101)_2$ it means decimal rule 165_{10}. Summarizing, we obtain set of rules {90, 105, 150, 165} it is set of rules selected by Tomassini and Perenoud and described in [15].

This set of rules is maximal length set of rules with neighbourhood radius $r = 1$, which only fulfill upper conditions and collectively generates bit sequences free from stable ones. Moreover, each subsets of these rules also do not produce stable sequences.

Presented method of selection of the maximal set of rules for $r = 1$ could be successfully applied for rules with $r = 2$ and larger. This method allow to select CA rules in the maximal set in kind and easy way. Also we can be certain that rules such way constructed are free form bad collaboration with other rules, and in consequence free from generation stable bit sequences in CA-based PRNG.

VI. Conclusion and future works

In the paper is presented new method of selecting rules for cellular automata based pseudorandom number (bit sequences) generator. Presented method is on the base of balance (regularity) the CA and CA rule, and gives an answer which rules of CA are suitable to use them in the generator. Also the maximal set of such rules (for rules with $r = 1$) is shown, as the final answer concerning more suitable sets. Rules selected by the presented method are free from generating stable sequences of bits. Connection of these rules in one set, and application as a rules managing the CA in CA-based PRNG leads to creation a good quality bit sentences, which could be use in different cryptographical algorithms.

In the next studies will be shown new set of CA rules suitable for use in CA-based PRNG. These set will be selected from CA rules with $r = 2$, with use of presented method based on balance of CA rule. Also will be shown formula which gives an answer how many such rules exists for CA with $r > 1$. Also selected new set of rules ($r = 2$) will be tested by the set of Marsaglia Diehard tests, which gives an answer how good is tested CA-based PRNG. Obtained results will be compared with earlier proposed PRNG based on CA.

References

[1] Bouvry, P., Klein, G. and Seredynski, F. (2005) 'Weak Key Analysis and Micro-controller Implementation of CA Stream Ciphers', *LNAI 3684*, Springer, pp. 910–915.

[2] Guan, P. (1987) 'Cellular Automaton Public-Key Cryptosystem', *Complex Systems*, Vol. 1, pp. 51–56.

[3] Gutowitz, H. (1993) 'Cryptography with Dynamical Systems', *in E. Goles and N. Boccara (Eds.) Cellular Automata and Cooperative Phenomena*, Kluwer Academic Press.

[4] Habutsu, T., Nishio, Y., Sasae, I. and Mori, S. (1991) 'A Secret Key Cryptosystem by Iterating a Chaotic Map', *Proc. of Eurocrypt'91*, pp. 127–140.

[5] Hortensius, R.D., McLeod, R.D. and Card, H.C. (1989) 'Parallel random number generation for VLSI systems using cellular automata', *IEEE Trans. on Computers*, Vol. 38, LNCS 218, Springer, pp.1466–1473.

[6] Kari, J. (1992) 'Cryptosystems based on reversible cellular automata', *Personal Communication*.

[7] Menezes, A., van Oorschot, P. and Vanstone, S. (1996) *Handbook of Applied Cryptography*, CRC Press.

[8] Nandi, S., Kar, B.K. and Chaudhuri, P.P. (1994) 'Theory and Applications of Cellular Automata in Cryptography', *IEEE Trans. on Computers*, Vol. 43, pp.1346–1357.

[9] Sang-Ho Shin1, Kee-Young Yoo, An Improved PRNG Based on the Hybrid between One- and Two- Dimensional Cellular Automata, chapter 15 in Cellular Automata - Innovative Modelling for Science and Engineering, Ed. Alejandro Salcido, 2011

[10] Schneier, B. (1996) *Applied Cryptography*, Wiley, New York.

[11] Seredynski, F., Bouvry, P. and Zomaya A. (2004) 'Cellular Automata Computation and Secret Key Cryptography', *Parallel Computing*, Vol. 30, pp.753–766.

[12] Sipper, M. and Tomassini, M. (1996) 'Generating parallel random number generators by cellular programming', *Int. Journal of Modern Physics C*, Vol. 7, No. 2, pp.181–190.

[13] Szaban, M., Seredynski, F., Bouvry, P. (2006) Collective Behaviour of Rules for Cellular Automata-based Stream Ciphers, 2006 IEEE Congress on Evolutionary Computation (CEC 2006), July 16-21, Vancouver, Canada, IEEE, pp.486-490

[14] Szaban, M., Seredynski, F., Bouvry, P. (2006) Evolving Collective Behavior of Cellular Automata for Cryptography, The 13 IEEE Mediterranean Electrotechnical Conference (MELECON 2006), Benalmadena (Malaga), Spain, May 16-19, pp. 799-802

[15] Tomassini, M. and Perrenoud, M. (2000) 'Stream Ciphers with One- and Two-Dimensional Cellular Automata' *in M. Schoenauer et al. (Eds.) Parallel Problem Solving from Nature - PPSN VI*, LNCS 1917, Springer, pp.722–731.

[16] Tomassini, M. and Sipper, M. (2000) 'On the Generation of High-Quality Random Numbers by Two-Dimensional Cellular Automata' *IEEE Trans. on Computers*, Vol. 49, No. 10, pp.1140–1151.

[17] Wolfram, S. (1986) 'Cryptography with Cellular Automata', *Crypto '85 Proceedings*, LNCS 218, Springer, pp.429–432.

[18] Youssef, A., Tavares, S., Resistance of Balanced S-boxes to Linear and Dierential Cryptanalysis, Information Processing Letters 56, (1995) 249–252

SESSION

COMMUNICATION SYSTEMS AND INPUT OUTPUT SYSTEMS + INTERCONNECTION NETWORKS AND TOPOLOGIES + ROUTING METHODS + NEW PROTOCOLS + VANET + PEER-TO-PEER NETWORKS + SENSOR NETWORKS AND APPLICATIONS

Chair(s)

TBA

A Fault-tolerant Routing Algorithm based on Safety Levels in a Hyper-Star Graph

Yuko Sasaki Yuki Hirai Hironori Nakajo Keiichi Kaneko
Graduate School of Engineering
Tokyo University of Agriculture and Technology
Koganei-shi, Tokyo 184-8588, JAPAN
{50013646114@st,yhirai@cc,nakajo@cc,k1kaneko@cc}.tuat.ac.jp

Abstract ***A hyper-star graph $HS(n,k)$ provides a promising topology for interconnection networks of parallel processing systems because it combines the merits of a hypercube and a star graph. In this study, we propose a fault-tolerant routing algorithm that establishes a fault-free path between any pair of non-faulty nodes in an $HS(n,k)$ with faulty nodes by using limited global information called safety levels. In addition, we carried out a computer experiment to verify the effectiveness of the algorithm.***

Keywords: multicomputer, interconnection network, parallel processing, hypercube, star graph, faulty node, performance evaluation

1 Introduction

With the development of research on parallel processing systems, many new topologies for interconnection networks have been proposed [1, 4, 5, 6, 8, 10, 12] instead of simple topologies such as a ring, a mesh, a torus, a hypercube [13], and so on. A hyper-star graph $HS(n,k)$ provides a such new topology, and it is promising because it combines the merits of a hypercube and a star graph [9]. Algorithms should be designed and developed presuming the existence of faulty elements in a large-scaled parallel system. Therefore, in this paper, we focus on a hyper-star graph $HS(n,k)$ that has faulty nodes, and propose an adaptive fault-tolerant routing algorithm between non-faulty nodes.

If each non-faulty node collects information of all faulty nodes as global information, optimal fault-tolerant routing is possible. However, this approach is impractical since it requires space and time complexities, whose orders are equal to the number of nodes in the graph. On the other hand, if each non-faulty node collects the status of its neighbor nodes only as local information for fault-tolerant routing, high reachability cannot be attained. Therefore, some approaches collect a part of the global information to attain high reachability. The information is called limited global information.

For a hypercube, there are several approachs based on the limited global information. By recursively classifying non-faulty nodes into safe, ordinary unsafe, and strongly unsafe nodes depending on the classification of neighbor nodes, Chiu and Wu have proposed an efficient fault-tolerant routing algorithm [2]. To improve the algorithm, Chiu and Chen introduced the routing capabilities that are obtained by classifying the safety nodes with respect to the Hamming distance to the destination nodes [3]. Wu has also proposed a similar fault-tolerant routing algorithm independently by introducing the safety vectors [14]. In addition, Kaneko and Ito have proposed a fault-tolerant routing algorithm based on classification of ordinary and strongly unsafe nodes with respect to the Hamming distance as well as an efficient method to obtain classification of them [7].

For a star graph, Yeh et al. have proposed an algorithm based on the safety vectors to attain efficient fault-tolerant routing [15]. The routing in a star graph is more complicated than that in a hypercube. Hence, the safety vectors on a star graph are based on routing patterns while those on a hypercube are based on distances.

For a regular hyper-star graph $HS(2n, n)$, Nishiyama et al. have proposed an algorithm based on the safety levels, which represent limited global information [11]. In our approach, we introduce the safety levels for a generic non-regular hyper-star graph to attain high reachability.

The rest of this paper is structured as follows. In Section 2, a hyper-star graph, a safety level, and other requisite concepts are defined, and some properties are proved. In Section 3, we describe the fault-tolerant routing algorithm based on the safety levels. In Section 4, by a computer experiment, we verify the effectiveness of our algorithm. In Section 5, we give conclusions and a future work.

2 Preliminaries

In this section, we give a definition of a hyper-star graph and lemmas about its properties. We also introduce a definition of a safety level.

Definition 1 (hyper-star graph $HS(n, k)$) An $HS(n, k)$ is an undirected graph, which has ${}_nC_k$ nodes. Each node $\boldsymbol{a}$ consists of n bits $(a_1, a_2, \ldots, a_n)$. Among these bits, k bits are always equal to 1 while the remaining $(n-k)$ bits are always 0 ($\boldsymbol{a} \in \{0,1\}^n$, $\sum_{i=1}^{n} a_i = k$). For two nodes $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)$, $\boldsymbol{b} = (b_1, b_2, \ldots, b_n)$, an edge $(\boldsymbol{a}, \boldsymbol{b})$ exists if and only if there exists $j (\in \{2, 3, \ldots, n\})$ such that $b_1 = \bar{a}_1$, $b_j = \bar{a}_j = a_1$, $b_i = a_i$ $(2 \le i \ne j \le n)$. □

Figure 1 shows an example of $HS(6, 2)$. Table 1 shows comparison of a hyper-star graph $HS(n, k)$ $(n > 2k)$ with a hypercube Q_n, a hierarchical hypercube HHC_{2^n+n}, and a hierarchical cube network HCN_n.

In an $HS(n, k)$, for two nodes $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)$ and $\boldsymbol{b} = (b_1, b_2, \ldots, b_n)$, the distance between them $d(\boldsymbol{a}, \boldsymbol{b})$ is given by $\sum_{i=2}^{n} a_i \oplus b_i$.

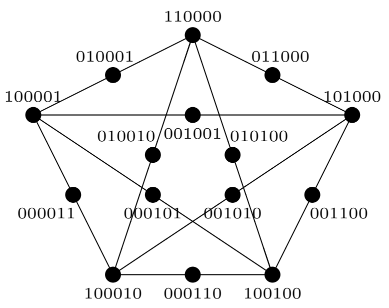


Figure 1: An example of a hyper-star graph $HS(6, 2)$.

Table 1: Comparison of a hyper-star graph with other topologies.

	#nodes	degree	connect.	diameter
$HS(n, k)$	${}_nC_k$	k	k	$2k$
Q_n	2^n	n	n	n
HHC_{2^n+n}	2^{2^n+n}	$n+1$	$n+1$	2^{n+1}
HCN_n	2^{2n}	$n+1$	$n+1$	$\lfloor 4(n+1)/3 \rfloor$

Moreover, among the neighbor nodes of $\boldsymbol{a}$, let $Pre(\boldsymbol{a}, \boldsymbol{b}) = \{\boldsymbol{n} \mid \boldsymbol{n} \in N(\boldsymbol{a}), d(\boldsymbol{n}, \boldsymbol{b}) = d(\boldsymbol{a}, \boldsymbol{b}) - 1\}$ and $Spr(\boldsymbol{a}, \boldsymbol{b}) = \{\boldsymbol{n} \mid \boldsymbol{n} \in N(\boldsymbol{a}), d(\boldsymbol{n}, \boldsymbol{b}) = d(\boldsymbol{a}, \boldsymbol{b}) + 1\}$ be the neighbor nodes that are on the shortest paths from $\boldsymbol{a}$ to $\boldsymbol{b}$ and those on the detour paths from $\boldsymbol{a}$ to $\boldsymbol{b}$, respectively.

Lemma 1 For an $HS(n, k)$, its diameter $diam(HS(n, k))$ is given as follows:

$$diam(HS(n,k)) = \begin{cases} 2(n-k) & (n < 2k) \\ 2k - 1 & (n = 2k) \\ 2k & (n > 2k) \end{cases}$$

(Proof) For two nodes $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)$ and $\boldsymbol{b} = (b_1, b_2, \ldots, b_n)$ in an $HS(n, k)$, if $n < 2k$, the diameter $2(n-k)$ is, for instane, given by $d(\boldsymbol{a}, \boldsymbol{b})$ where $a_1 = a_2 = \cdots = a_k = 1$, $a_{k+1} = \cdots = a_n = 0$, $b_1 = 1$, $b_2 = b_3 = \cdots = b_{n-k+1} = 0$, $b_{n-k+2} = \cdots = b_n = 1$. From symmetric property, if $n > 2k$, the diameter is equal to $2k$. If $n = 2k$, the diameter $2k - 1$ is, for instance, given by $d(\boldsymbol{a}, \boldsymbol{b})$ where $a_1 = a_2 = \cdots = a_k = 1$, $a_{k+1} = \cdots = a_n = 0$, $b_1 = b_2 = \cdots = b_k = 0$, $b_{k+1} = \cdots = b_n = 1$. □

Lemma 2 For a node $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)$ in an $HS(n, k)$, $|N(\boldsymbol{a})| = k$ if $a_1 = 0$, and $|N(\boldsymbol{a})| = n - k$ if $a_1 = 1$.

(Proof) If $a_1 = 0$, there are k 1's in $a_2, a_3, \ldots, a_n$. Therefore, $|N(\boldsymbol{a})| = k$. If $a_1 = 1$, there are $(n-k)$ 0's in $a_2, a_3, \ldots, a_n$. Therefore, $|N(\boldsymbol{a})| = n - k$. □

Lemma 3 For two nodes $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)$ and $\boldsymbol{b} = (b_1, b_2, \ldots, b_n)$ in an $HS(n,k)$, $|Pre(\boldsymbol{a},\boldsymbol{b})| = \lceil d/2 \rceil$ and $|Spr(\boldsymbol{a},\boldsymbol{b}) = |N(\boldsymbol{a})| - \lceil d/2 \rceil$ where $d = d(\boldsymbol{a},\boldsymbol{b})$.
(Proof) If $d = \sum_{i=2}^{n}(a_i \oplus b_i)$ is even, $a_1 = b_1$ since $\sum_{i=1}^{n}(a_i \oplus b_i)$ is always even. Hence, among d bits of $a_2, a_3, \ldots, a_n$ that are different from corresponding $b_2, b_3, \ldots, b_n$, $d/2$ bits are equal to $\bar{a}_1$. Thus, $|Pre(\boldsymbol{a},\boldsymbol{b})| = d/2$, and $|Spr(\boldsymbol{a},\boldsymbol{b})| = |N(\boldsymbol{a})| - d/2$. On the other hand, if d is odd, $a_1 = \bar{b}_1$. Hence, among d bits of $a_2, a_3, \ldots, a_n$ that are different from corresponding $b_2, b_3, \ldots, b_n$, $(d+1)/2$ bits are equal to $\bar{a}_1$. Thus, $|Pre(\boldsymbol{a},\boldsymbol{b})| = (d+1)/2$ and $|Spr(\boldsymbol{a},\boldsymbol{b})| = |N(\boldsymbol{a})| - (d+1)/2$. To recap, $|Pre(\boldsymbol{a},\boldsymbol{b})| = \lceil d/2 \rceil$ and $|Spr(\boldsymbol{a},\boldsymbol{b})| = |N(\boldsymbol{a})| - \lceil d/2 \rceil$. □

For a node $\boldsymbol{a}$ in an $HS(n,k)$ and a distance d $(1 \leq d \leq diam(HS(n,k)))$, we introduce an safety level $S_d(\boldsymbol{a})$ so that it indicates that for any non-faulty node which is located with distance d from the node $\boldsymbol{a}$, a fault-free path of length d from $\boldsymbol{a}$ to the node can be established.

Definition 2 For a node $\boldsymbol{a}$ in an $HS(n,k)$ with a set of faulty nodes F, a safety level $S_d(\boldsymbol{a})$ with respect to a distance d is defined as follows:

1. $S_d(\boldsymbol{a}) = 1$ if $\boldsymbol{a} \notin F$, $d = 1$,
2. $S_d(\boldsymbol{a}) = 1$ if $\boldsymbol{a} \notin F$, $d \geq 2$, and for any $J(\subset N(\boldsymbol{a}))$ such that $|J| = \lceil d/2 \rceil$, there exists $\boldsymbol{n}(\in J)$ such that $S_{d-1}(\boldsymbol{n}) = 1$,
3. $S_d(\boldsymbol{a}) = 0$ otherwise. □

Since it takes much time to calculate safety levels in each node based on Definition 2, we introduce a simple calculation method based on the following lemma.

Lemma 4 For a node $\boldsymbol{a}(\notin F)$ in an $HS(n,k)$ with a set of faulty nodes F and a distance $d(\geq 2)$, the following two conditions are equivalent:

1. For any $J(\subset N(\boldsymbol{a}))$ such that $|J| = \lceil d/2 \rceil$, there exists a node $\boldsymbol{n}(\in J)$ such that $S_{d-1}(\boldsymbol{n}) = 1$.
2. $|\{\boldsymbol{n} \mid \boldsymbol{n} \in N(\boldsymbol{a}), S_{d-1}(\boldsymbol{n}) = 1\}| \geq |N(\boldsymbol{a})| - \lceil d/2 \rceil + 1$.

(Proof) Since $|J| = \lceil d/2 \rceil$, $|N(\boldsymbol{a}) \setminus J| = |N(\boldsymbol{a})| - \lceil d/2 \rceil$ holds. Hence, Condition 2 implies Condition 1. Therefore, sufficiency is proved. For necessity, we assume that Condition 2 does not hold. Then, from $|\{\boldsymbol{n} \mid \boldsymbol{n} \in N(\boldsymbol{a}), S_{d-1}(\boldsymbol{n}) = 1\}| \leq |N(\boldsymbol{a})| - \lceil d/2 \rceil$, there exists $J(\subset N(\boldsymbol{a}))$ such that $|J| = \lceil d/2 \rceil$ and $\{\boldsymbol{n} \mid \boldsymbol{n} \in N(\boldsymbol{a}), S_{d-1}(\boldsymbol{n}) = 1\} \subset N(\boldsymbol{a}) \setminus J$. Therefore, Condition 1 does not hold, either. Necessity is also proved. From above discussion, the lemma is proved. □

3 Fault-tolerant routing algorithm

In an $HS(n,k)$, from Lemma 4, for a non-faulty node $\boldsymbol{a}$ and a distance d, we can compare $\sum_{\boldsymbol{n} \in N(\boldsymbol{a})} S_{d-1}(\boldsymbol{n})$ with $|N(\boldsymbol{a})| - \lceil d/2 \rceil + 1$ to judge sufficiency of Condition 2 in Definition 2. Figure 2 shows an algorithm as Procedure SL to calculate safety levels $S_d(\boldsymbol{a})$ $(1 \leq d \leq diam(HS(n,k)))$ at a node $\boldsymbol{a}$. The procedure must be executed in synchronization at all nodes.

```
procedure SL($\boldsymbol{a}$, $F$)
begin
  for $d$ := 1 to $diam(HS(n,k))$ do
    if $\boldsymbol{a} \in F$ then $S_d(\boldsymbol{a})$ := 0
    else if $d$ = 1 then $S_d(\boldsymbol{a})$ := 1
    else if $\sum_{\boldsymbol{n} \in N(\boldsymbol{a})} S_{d-1}(\boldsymbol{n})$ >= $|N(\boldsymbol{a})| - \lceil d/2 \rceil + 1$ then
      $S_d(\boldsymbol{a})$ := 1
    else $S_d(\boldsymbol{a})$ := 0
end
```

Figure 2: Algorithm to calculate safety levels.

Theorem 1 At each node in an $HS(n,k)$, the time complexity to calculate safety levels with respect to all distances d $(1 \leq d \leq diam(HS(n,k)))$ is $O(n^2)$.
(Proof) From Lemma 4, to calculate a safety level $S_d(\boldsymbol{a})$ with respect to a distance $d(\geq 2)$ at

a node $\boldsymbol{a}$, it is necessary to collect $S_{d-1}(\boldsymbol{n})$ from each node $\boldsymbol{n}$ in neighbor nodes $N(\boldsymbol{a})$ of $\boldsymbol{a}$ and sum up them. This process requires $O(n)$ time complexity. Therefore, it takes $O(n^2)$ time in total to calculate safety levels for all distances d $(2 \leq d \leq diam(HS(n,k)))$. □

From Theorem 1, it takes $O(n^2)$ time to calculate the safety levels with respect to all distances at each node. If all the nodes calculate the safety levels in synchronization, the total time complexity is $O(n^2)$.

In an $HS(n,k)$ with a set of faulty nodes F, a fault-tolerant routing algorithm based on safety levels is shown in Figure 3 as Procedure FTS. To send a message from a non-faulty node $\boldsymbol{s}$ to a non-faulty node $\boldsymbol{d}$, we should call this procedure as FTS($\boldsymbol{s}$, $\boldsymbol{d}$, F).

```
procedure FTS($\boldsymbol{c}$, $\boldsymbol{d}$, $F$)
begin
  $d$ := $d(\boldsymbol{c}, \boldsymbol{d})$;
  if $d$ = 0 then deliver the message to $\boldsymbol{c}$
  else if $d$ = 1 then FTS($\boldsymbol{d}$, $\boldsymbol{d}$, $F$)
  else if $\exists \boldsymbol{n}^* \in \{\boldsymbol{n} \mid \boldsymbol{n} \in Pre(\boldsymbol{c},\boldsymbol{d}), S_{d-1}(\boldsymbol{n}) = 1\}$ then
    FTS($\boldsymbol{n}^*$, $\boldsymbol{d}$, $F$)
  else if $\exists \boldsymbol{n}^* \in \{\boldsymbol{n} \mid \boldsymbol{n} \in Spr(\boldsymbol{c},\boldsymbol{d}), S_{d+1}(\boldsymbol{n}) = 1\}$ then
    FTS($\boldsymbol{n}^*$, $\boldsymbol{d}$, $F$)
  else if $\exists \boldsymbol{n}^* \in Pre(\boldsymbol{c},\boldsymbol{d}) \setminus F$ then FTS($\boldsymbol{n}^*$, $\boldsymbol{d}$, $F$)
  else if $\exists \boldsymbol{n}^* \in Spr(\boldsymbol{c},\boldsymbol{d}) \setminus F$ then FTS($\boldsymbol{n}^*$, $\boldsymbol{d}$, $F$)
  else error('delivery failed')
end
```

Figure 3: Fault-tolerant routing algorithm based on safety levels.

It takes $O(n)$ time to identify $Pre(\boldsymbol{c},\boldsymbol{d})$ and $Spr(\boldsymbol{c},\boldsymbol{d})$, and to check $S_{d-1}(\boldsymbol{n})$ to find the node $\boldsymbol{n}^*$. Note that Algorithm FTS may cause infinite loops.

4 Computer experiment

In this section, we give the detail of the results of a computer experiment conducted to compare our algorithm FTS and a simple algorithm SMP shown in Figure 4. Note that Algorithm SMP also has possibility to cause infinite loops. The computer experiment was carried out for an $HS(n,k)$ where $(n,k) = (9,2),(9,3),(9,4),(10,2),(10,3)$, and $(10,4)$ changing the ratio of faulty nodes α from 0.0 to 0.9, and we have measured the ratio of successful routings and their path lengths.

```
procedure SMP($\boldsymbol{c}$, $\boldsymbol{d}$, $F$)
begin
  $d$ := $d(\boldsymbol{c}, \boldsymbol{d})$;
  if $d$ = 0 then deliver the message to $\boldsymbol{c}$
  else if $\exists \boldsymbol{n}^* \in Pre(\boldsymbol{c},\boldsymbol{d}) \setminus F$ then SMP($\boldsymbol{n}^*$, $\boldsymbol{d}$, $F$)
  else if $\exists \boldsymbol{n}^* \in Spr(\boldsymbol{c},\boldsymbol{d}) \setminus F$ then SMP($\boldsymbol{n}^*$, $\boldsymbol{d}$, $F$)
  else error('delivery failed')
end
```

Figure 4: A simple fault-tolerant routing algorithm.

Concretely, first, in an $HS(n,k)$, we selected faulty nodes randomly with the ratio α. Next, we selected the source node $\boldsymbol{s}$ and the destination node $\boldsymbol{d}$ from non-faulty nodes. Finally, after checking the connectivity of $\boldsymbol{s}$ and $\boldsymbol{d}$, we applied the fault-tolerant routing algorithms. If $\boldsymbol{s}$ and $\boldsymbol{d}$ are not connected, that is, there is no fault-free path between them, we start over from the selection of faulty nodes. For each pair of (n,k) and α, we executed at least 100,000 trials. Figures 5 to 10 show the ratios of successful routings by Algorithms FTS, and SMP, respctively. Also, Figures 11 to 16 show the average path lengths by Algorithms FTS, and SMP, respctively.

From these figures, we can see that Algorithm FTS shows better performance than Algorithm SMP in any pair of (n,k) with small amount of additional costs.

5 Conclusions and future work

In this paper, we have introduced the concept of safety levels in a hyper-star graph $HS(n,k)$ and proposed a fault-tolerant routing algorithm. We have proved that the time complexity to calculate safety levels with respect to all the distances at each node is $O(n^2)$. Moreover, we have carried out a computer experiment and verified high reachability to the destination nodes.

As a future work, it is interesting to introduce a stochastic framework into our method.

Acknowledgments

This study is partly supported by a Grant-in-Aid for Scientific Research (C) of the Japan Society for the Promotion of Science (JSPS) under Grant No. 25330079.

References

[1] S.B. Akers and B. Krishnamurthy, "A group-theoretic model for symmetric interconnection networks," IEEE Transactions on Computers, vol.38, no.4, pp.555–566, April 1989.

[2] G.M. Chiu and K.S. Chen, "Use of routing capability for fault-tolerant routing in hypercube multicomputers," IEEE Transactions on Computers, vol.46, no.8, pp.953–958, Aug. 1997.

[3] G.M. Chiu and S.P. Wu, "A fault-tolerant routing strategy in hypercube multicomputers," IEEE Transactions on Computers, vol.45, no.2, pp.143–155, Feb. 1996.

[4] P.F. Corbett, "Rotator graphs: An efficient topology for point-to-point multiprocessor networks," IEEE Transactions on Parallel and Distributed Systems, vol.3, no.5, pp.622–626, May 1992.

[5] K. Ghose and K.R. Desai, "Hierarchical cubic networks," IEEE Transactions on Parallel and Distributed Systems, vol.6, no.4, pp.427–435, April 1995.

[6] J.S. Jwo, "Properties of star graph, bubble-sort graph, prefix-reversal graph and complete-transposition graph," Journal of Information Science and Engineering, vol.12, no.4, pp.603–617, Dec. 1996.

[7] K. Kaneko and H. Ito, "Fault-tolerant routing algorithms for hypercube interconnection networks," IEICE Transactions on Information and Systems, vol.E84-D, no.1, pp.121–128, Jan. 2001.

[8] S. Latifi and P.K. Srimani, "A new fixed degree regular network for parallel processing," Proceedings of the Eighth IEEE Symposium on Parallel and Distributed Processing, pp.152–159, Oct. 1996.

[9] H.O. Lee, J.S. Kim, E. Oh, and H.S. Lim, "Hyper-star graph: A new interconnection network improving the network cost of the hypercube," Proceedings of the First EurAsian Conference on Information and Communication Technology, London, UK, pp.858–865, Springer-Verlag, Oct. 2002.

[10] Q.M. Malluhi and M.A. Bayoumi, "The hierarchical hypercube: A new interconnection topology for massively parallel systems," IEEE Transactions on Parallel and Distributed Systems, vol.5, no.1, pp.17–30, Jan. 1994.

[11] Y. Nishiyama, Y. Hirai, and K. Kaneko, "Fault-tolerant routing based on safety levels in a hyper-star graph," Proceedings of IADIS Applied Computing 2012, pp.348–352, Oct. 2012.

[12] F.P. Preparata and J. Vuillemin, "The cube-connected cycles: A versatile network for parallel computation," Communications of ACM, vol.24, no.5, pp.300–309, May 1981.

[13] C.L. Seitz, "The cosmic cube," Communications of the ACM, vol.28, no.1, pp.22–33, Jan. 1985.

[14] J. Wu, "Adaptive fault-tolerant routing in cube-based multicomputers using safety vectors," IEEE Transactions on Parallel and Distributed Systems, vol.9, no.4, pp.322–334, April 1998.

[15] S.I. Yeh, C.B. Yang, and H.C. Chen, "Fault-tolerant routing on the star graph with safety vectors," Proceedings of the Sixth Annual International Symposium on Parallel Architectures, Algorithms, and Networks, pp.301–306, May 2002.

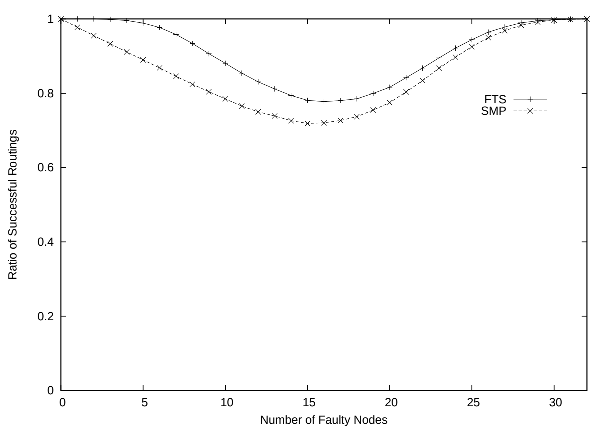


Figure 5: The ratios of successful routings by Algorithm `FTS` and `SMP` in an $HS(9,2)$.

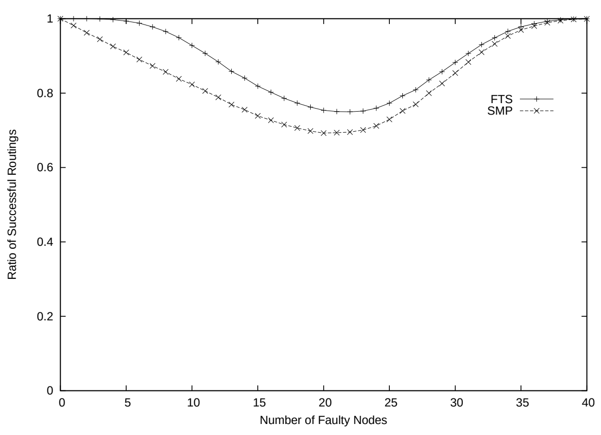


Figure 8: The ratios of successful routings by Algorithm `FTS` and `SMP` in an $HS(10,2)$.

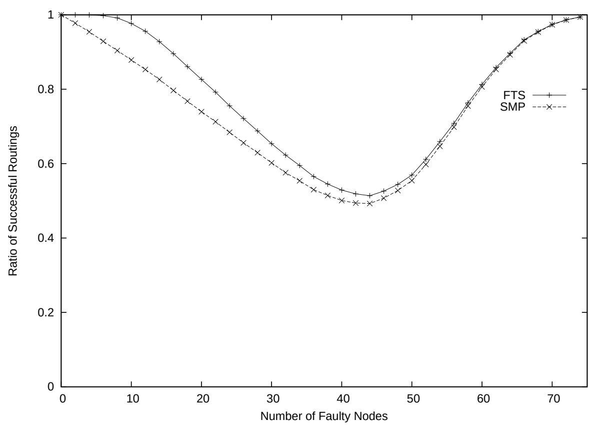


Figure 6: The ratios of successful routings by Algorithm `FTS` and `SMP` in an $HS(9,3)$.

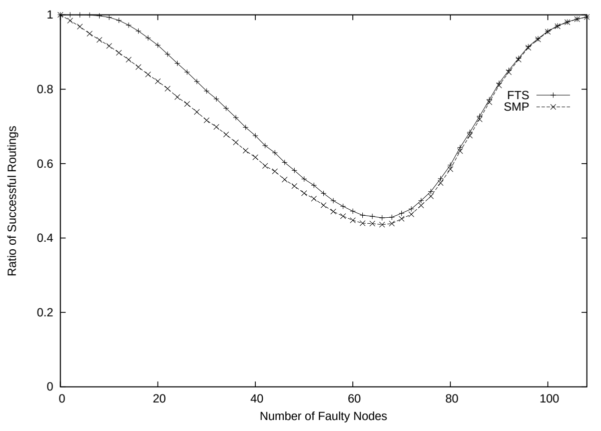


Figure 9: The ratios of successful routings by Algorithm `FTS` and `SMP` in an $HS(10,3)$.

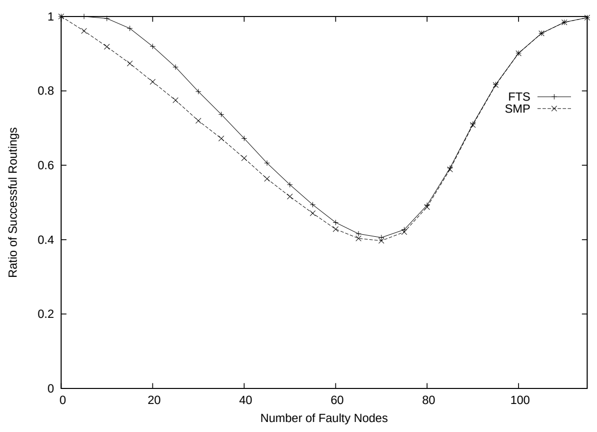


Figure 7: The ratios of successful routings by Algorithm `FTS` and `SMP` in an $HS(9,4)$.

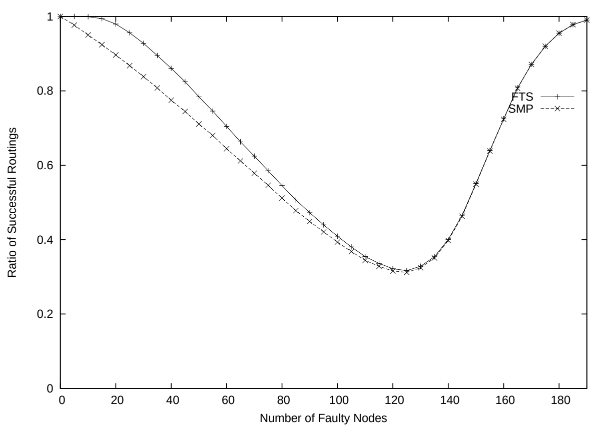


Figure 10: The ratios of successful routings by Algorithm `FTS` and `SMP` in an $HS(10,4)$.

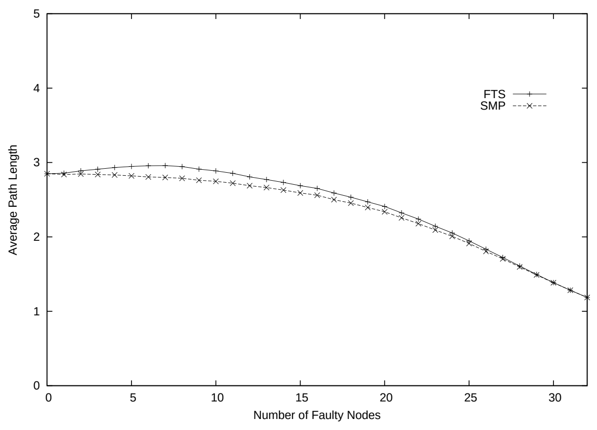


Figure 11: The average path lengths by Algorithm FTS and SMP in an $HS(9,2)$.

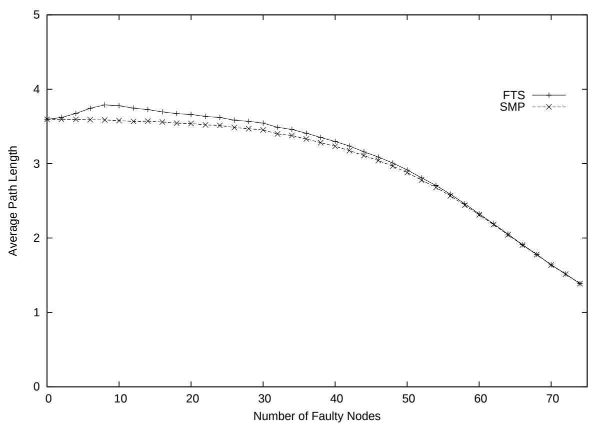


Figure 12: The average path lengths by Algorithm FTS and SMP in an $HS(9,3)$.

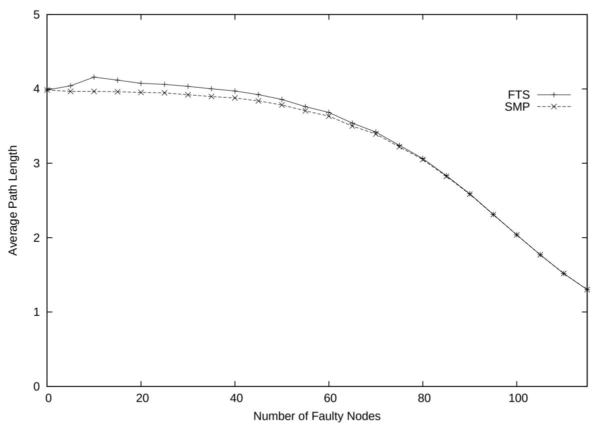


Figure 13: The average path lengths by Algorithm FTS and SMP in an $HS(9,4)$.

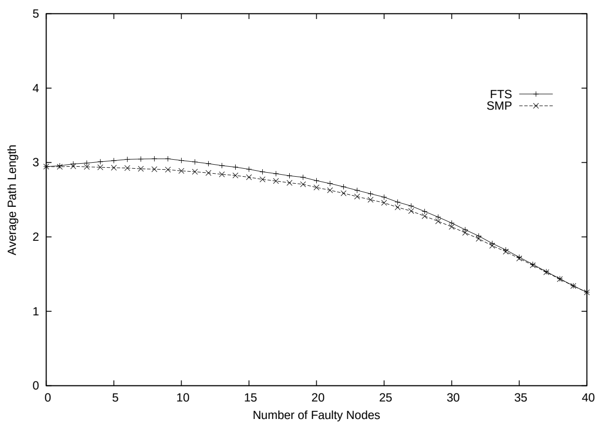


Figure 14: The average path lengths by Algorithm FTS and SMP in an $HS(10,2)$.

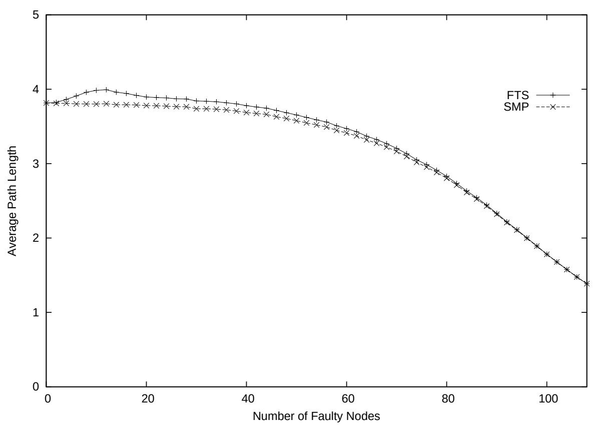


Figure 15: The average path lengths by Algorithm FTS and SMP in an $HS(10,3)$.

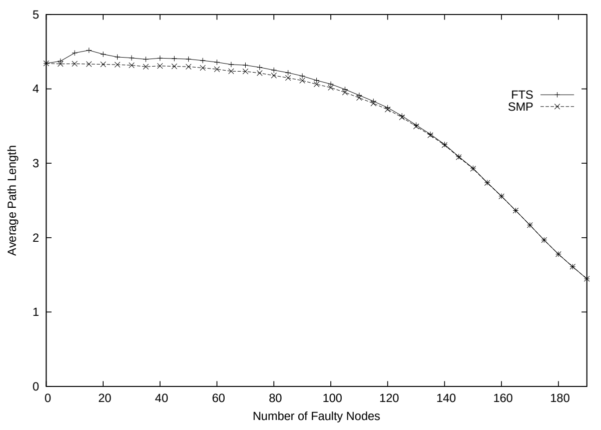


Figure 16: The average path lengths by Algorithm FTS and SMP in an $HS(10,4)$.

A Novel Quorum Protocol for Improved Performance

A. Parul Pandey[1], **B. M Tripathi**[2]
[2]Computer Science, Institution of Engineering and Technology , Lucknow, U.P., India

Abstract— *In this paper, we present an efficient quorum protocol for reading data with minimum read quorum size. This protocol for managing replicated data is named as Wheel Quorum Protocol. We impose a logical wheel structure on the set of copies of an object. The protocol ensures minimum read quorum size of one, by reading one copy of an object while maintaining acceptable size of write operations. In this paper, we also analyze several quorum types in terms of quorum size and message overhead. Our protocol proves to incur minimum communication overhead. Wheel structure has a wider application area as it can be imposed in a network with any number of nodes. This protocol is especially beneficial for read intensive applications.*

Keywords: Replica-control, distributed database, quorum consensus.

1. Introduction

In a distributed database system, data is replicated [1] to achieve fault-tolerance. One of the most important advantages of replication is that it masks and tolerates failures in the network gracefully and increases availability. In particular, the system remains operational and available to the users despite failures. In case of multiple access a problem that must be solved while using replication is about maintaining the copies in a consistent state. To keep logical data consistent, there must exist a control protocol responsible for synchronizing the access. A popular method for maintaining consistency of replicated data is weighted voting [2] which is a generalization of the majority consensus method presented in [3]. In the quorum consensus (QC) [4] algorithm, we assign a non-negative weight to each copy x_A of x. We then define a read threshold RT and write threshold WT for x, such that both $2WT$ and $(RT + WT)$ are greater than the total weight of all copies of x. A read (or write) quorum of x is any set of copies of x with a weight of at least RT (or WT). For better performance, some logical structure is imposed on the network, and the quorums are chosen under the consideration of such structures. Such logical structures include the tree [5], diamond [6], ring [7], triangular mesh [8], and grid [9] structures. A geometric approach for dealing with logical structures is proposed in [10].

In this paper we propose a novel protocol, which is called The *Wheel Quorum Consensus Protocol* or simply *The Wheel Protocol*, for managing replicated data. In this protocol, the sites in the network are logically organized into a wheel structure. This protocol can be viewed as specialized version of ring and tree protocol. As compared to tree, grid, diamond and mesh protocol, wheel protocol is very flexible in arranging nodes in a network into the logical structure. Any number of nodes can be easily organized into a wheel structure.

The paper is organized as follows. In Section 2 we describe the system model. Section 3 discusses wheel quorum protocols which elaborates the motivation behind it, wheel structure and its quorum construction for read and write. In Section 4, we present performance evaluation and section 5 discusses the related work. We conclude the paper in Section 6.

2. Model

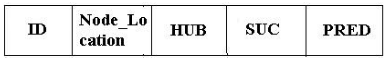
A distributed system consists of a set of distinct sites that communicate with each other by sending messages over a communication network. No assumptions are made regarding the speed, connectivity, or reliability of the network. It is assumed that sites are fail-stop [11] and communication links may fail to deliver messages.

Replication of data is achieved by storing copies of the same logical data item at different nodes. Read and write operations can be performed on replicated data. A node needs to obtain permission from a number of copies (quorum) before performing the operation using a control protocol.

In a replicated database, copies of an object may be stored at several sites in the network. Multiple copies of an object must appear as a single logical object to the transaction. This is termed as one-copy equivalence [12] and is enforced by the replica control protocol. The correctness criteria for replicated databases is one-copy serializability [12], which ensures one-copy equivalence and serializable execution of transactions. In order to ensure one-copy equivalence, a replicated object z may be read by reading a read quorum of copies, and it may be written by writing a write quorum of copies. The following restriction is placed on the choice of quorum assignments:

Quorum Intersection Property: For any two operations o[Z] and ó[z] on an data item x, where at least one of them is a write, the quorums must have a nonempty intersection.

Version numbers or timestamps are used to identify the current copy in a quorum. Each node is logically characterized by few attributes as shown in figure 1. **ID** which is a unique sequential ID. In our discussion, IDs are numbered as 0, 1, 2, 3,... n. **Node _ Location** is the location where the node is physically residing. In other words this is the address of a node in the network. **HUB** contains the ID of the node in the wheel which is currently acting as hub. In our discussion, ID of the HUB node is 0. **SUC** contains the ID of the successor w_{i+1}, which is the next node in the wheel. **PRED** contains the ID of the predecessor w_{i-1}, which is the previous node in the wheel.



ID	Node_Location	HUB	SUC	PRED

Fig. 1: Wheel Structure

The election quorum ensures that the HUB's ID is always 0.

3. Wheel Quorum Protocol

3.1 Motivation

Tradeoff between the cost for reading, writing, data availability and node fault tolerance is the deciding feature of all existing control protocols for replicated data . For example, the read-one write-all scheme needs only one copy as read quorum, but has the convenience of having a write quorum equal to the total number of copies (thus not tolerating a single node of failure).

The main motivation for our work was to develop a protocol which had a constant minimum cost for reading, while maintaining an acceptable cost for writing, since we are interested in systems where read operations are much more frequent than write operations.

To achieve this property, a logical wheel structure will be imposed on the set of copies of the object . This structure is used by operations to determine the copies that must be read or written. Figure 2, represents 5 nodes arranged in a wheel structure. Wheel logical structure can be arranged on any number of nodes, whereas other logical structures have constraints with nodes arrangement. We note that this structure is logical, and does not have to correspond to the actual physical structure of the network connecting the sites, storing the copies. This wheel structure is used to motivate the protocol.

3.2 The Wheel Structure

Let W_n = w_0, w_1, w_2,...,w_{n-1} be the set of nodes that store copies of a replicated data item. **A wheel**, W_n is a logical structure with n nodes, formed by connecting a single node called HUB to all vertices of an *(n-1)* cycle. The numerical notation for wheels is used inconsistently in the literature: some authors instead use *n* to refer to the length of the cycle, so their W_n is the graph we would denote as W_{n+1}. All nodes in the cycle maintain adjacency relationship by maintaining ID's of their successor and predecessor. Each node is defined by attributes ID, Node_Location, HUB, Suc, and Pred as shown in figure 1. Wheel structure is easily imposed on the set of nodes by selecting first node as HUB and adding other nodes as spokes in cycle by defining the successor (Suc(i)) , predecessor (Pred(i)) operations and by setting HUB in each spoke. Other operations are GetPermission(i) and rand(1..n).

GetPermisson(i), returns TRUE if the node w_i allows access to its own copy of the item. GetPermisson(i) returns FALSE when either node w_i refuses access or cannot be contacted due to failure. rand(1..n) selects and returns random number from 1 to n, where n is the number of nodes in wheel. This random number represents ID of selected node.

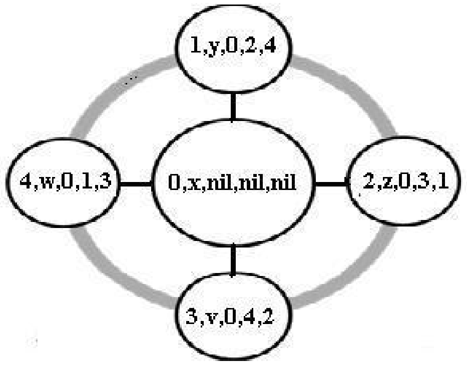


Fig. 2: Wheel Structure

3.3 The Wheel Protocol

In this protocol, all copies of a replicated data item are organized into a wheel structure. Specific algorithms are used for read and write quorums construction. There is one election algorithm for electing new HUB in case of failure of HUB or in case load threshold exceeds its limit. These algorithms use the adjacency information to guarantee quorum intersection, and to maintain the quorum sizes small. There are three type of quorums, Read, Write, and Election quorum.

Read Quorum is formed by getting access permission from HUB.

Write quorum is obtained by getting access permission from HUB and half of alternating nodes in the cycle, thus requiring the majority of the total number of copies. As an example, consider a replicated data item with six copies arranged in a wheel structure as shown in figure 3. Eligible read quorum is 0 (i.e. HUB) and sets eligible for write quorum are : {0,1,3,5}, {0,1,2,4}, {0,3,5,2}, {0,4,1,3} and {0,5,2,4}.

Notice that eligible quorums are coteries, satisfying the minimality and intersection properties[1].

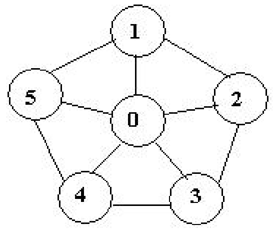


Fig. 3: 6 copies organized into a logical structure

Election quorum is called in two situations

1) When HUB crosses its load threshold
2) When HUB is unavailable

In both the above cases, the node initiating election quorum algorithm, selects randomly any 2 adjacent nodes, checks their version and makes the latest one the HUB by changing the location address between the old HUB and the newly elected one. This logically swaps the location of the two nodes. Other nodes are unaffected as they identify HUB by its ID, which is 0. Only the node_location is changed.

Advantages of this Election Quorum are-

1) HUB is never overloaded, as it gets swapped with a latest node whenever load _ threshold crosses its limit.
2) Improved load distribution. Assuming that each node in cycle has the equal probability of being selected as a new HUB, no node will be working as HUB for a longer time.
3) Constant minimum possible Read Quorum size of one. As, even if HUB is failed , it will be replaced with a new HUB. Thus ensuring that a request always reads data from HUB.

[1]The fact that the quorums are distinct and have the same size shows that they satisfy the minimality property: the intersection property will be shown later, when providing the protocol correctness

Without using election quorum, in the failure of HUB, Read Quorum can be achieved by accessing any 2 adjacent nodes in the cycle, which is double the cost of doing it with HUB. Our system has more number of reads as compared to write , so reads will keep on costing double till HUB recovers. All this can be avoided by using election quorum and electing new HUB. This way, present as well as subsequent reads can be satisfied by reading only HUB.

3.3.1 Quorum Construction

There are three algorithms for the wheel protocol. Algorithm 1, 2, 3 for read, write and election quorum respectively.

Algorithm 1 defines read quorum construction. This algorithm returns the HUB as the read quorum. In case of a HUB failure, the new HUB is elected by invoking the ElectionQuorum Protocol, which uses a random node in cycle.

Algorithm 1 Read Quorum(i)

```
if Empty(Wheel) then
    Return(nil)
else if GetPermission(HUB) is False then
    r= rand(1 .. n)
    Get ElectionQuorum(r)
    Return(HUB)
else
    Return(HUB)
end if
```

Algorithm 2 is to find write quorum. This protocol collects majority of nodes forming quorum between nodes in cycle of wheel in list, Quorum_list[]. This Quorum_list[] along with HUB makes write quorum. Protocol tries to form write quorum with current_node by traversing the cycle until, either a quorum is obtained or all copies have been examined (in which case quorum was not obtained and the request for writing is refused). In case of HUB failure Election Quorum elects a new HUB.

In case of HUB failure, Election Quorum (Algorithm 3) elects a new HUB. Election quorum selects two adjacent nodes(using successor function), selects the node with latest value and makes it the HUB.

Algorithm 2 Write Quorum(i)

```
– Main routine
1: nodes_covererd=0
2: current_node = i
3: if GetPermission(HUB) is False then
4:     n= random(cycle nodes)
5:     Get ElectionQuorum(n)
6:     GetPermission(HUB)
7: end if
8: if current_node is HUB then
9:     current_node = rand(1..n)
10: end if
11: while Empty QuorumList[] and $nodes\_covered < n$ do
12:     Quorum_list[]= Check(current_node)
13:     current_node=Suc(current_node)
14:     nodes_covered++
15: end while
16: Return(HUB $\bigcup$ QuorumList[])
– Check(i)
1: Quorum_list[] = null
2: Fail= nodes_checked=0
3: while $Fail \neq 1$ and $nodes\_checked < \lfloor n/2 \rfloor$ do
4:     if GetPermission(i) then
5:         Quorum_list.add(i)
6:         i=Suc(Suc(i))
7:         nodes_checked++
8:     else
9:         Fail=1
10:    end if
11: end while
12: if Fail then
13:     Quorum_list.flushall()
14:     return(Quorum_list[])
15: else
16:     return(Quorum_list[])
17: end if
```

Algorithm 3 Election Quorum(i)

```
1: current_node=i
2: Quorum=0
3: nodes_done=0
4: if current_node is HUB then
5:     current_node=rand(1..n-1)
6: end if
7: while Quorum is Empty or $nodes\_done < n$ do
8:     if current_node is accessible then
9:         if SUC(current_node) is accessible then
10:            Latest_node=Node_Location with most recent value
11:            Swap Node_Location of HUB and Latest_node
12:            Quorum=Latest_node
13:        else
14:            current_node=Suc(Suc(current_node))
15:            nodes_done=nodes_done + 2
16:        end if
17:    else
18:        current_node=Suc(current_node)
19:        nodes_done=nodes_done+1
20:    end if
21: end while
```

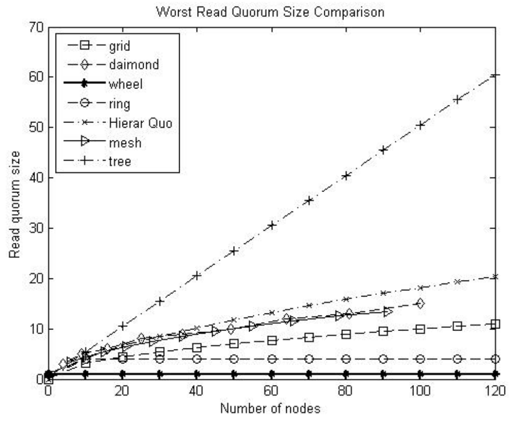
4. Performance Evaluation

Different logical structures and quorum forming methods result in different performances in different metrics. In this section, we present performance analysis of wheel protocol under different metrics. We compare the protocol with known protocols of majority quorum consensus[3], the grid protocol[9], the tree protocol[13], the hierarchical quorum consensus[14],the diamond protocol[6], the triangular mesh protocol[8], and the ring protocol[7].

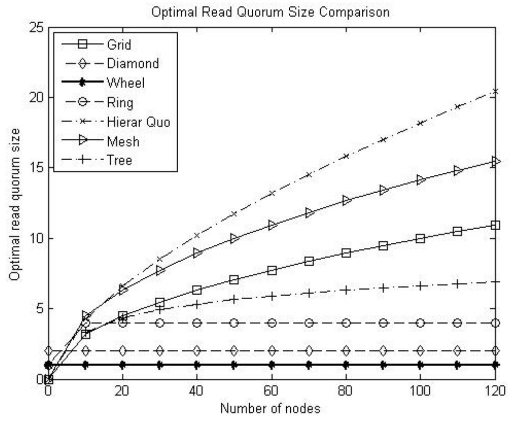
4.1 Quorum Size

In this section we examine optimal and worst case read and write quorum sizes. We analyze quorum sizes based on the type of applications where a number of read operations are much higher than the number of write. Read and write quorum sizes for majority quorum is $\lceil \frac{N+1}{N} \rceil$. We have not included majority in our quorum comparison, but its very clear that it has larger read and write quorum sizes. Hierarchical quorum consensus has best, as well as worst quorum size of $N^{0.63}$. In the case of the tree quorum protocol ,the size of read quorums vary from 1 to $(d+l)^h$. On the other hand, the cost of write operations is $[(d+1)^{h+1}-1]/d$ $[(d+1)^{h+1}-1]/d$. For a grid protocol, we assume that the grid structure is approximately a square, the read quorum size is approximated by $\sqrt(N)$, and the write quorum size is approximated by $2\sqrt(N)$. For diamond quorum consensus optimal read quorum size is 2 and is independent of the total number of sites. Worst case read quorum size is $\lceil\sqrt(2N)\rceil$. Optimal and worst quorum sizes for diamond write quorum are $\lceil\sqrt(2N)\rceil$ and 2 $\lceil\sqrt(2N)\rceil$ - 2 respectively. For the special case taken in [7], read and write quorum sizes are given by $q_r = n^{log_d 2}$ and $q_w = (\lfloor \frac{d}{2} \rfloor + 1)log_d n$ for the ring protocol. The quorum size in three triangular-mesh based protocols is k, which is $\lceil \sqrt{(2N)} \rceil$.

The wheel protocol has exceptionably minimum read quorum size of one(independent of number of sites), as read can be satisfied by reading only HUB. The remarkable point is that even in worst case the read quorum of wheel protocol remains one, by ensuring availability of HUB. Election quorum is used to elect new HUB whenever current HUB fails. Systems with more number of read operations than write get benefited by this smallest possible read quorum size and thus reduces the cost of operation. Write quorum size is $\lceil \frac{N-1}{2} \rceil + 1$. This large write quorum is justifiable against the constant small quorum size of one. This protocol is especially beneficial for systems with much more number of reads than write for eg. web based shopping sites.



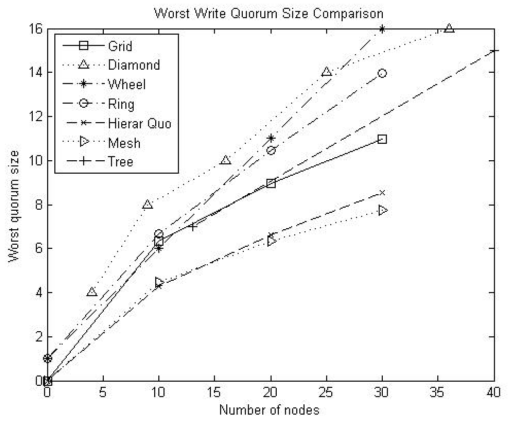
(a) Worst Quorum Size



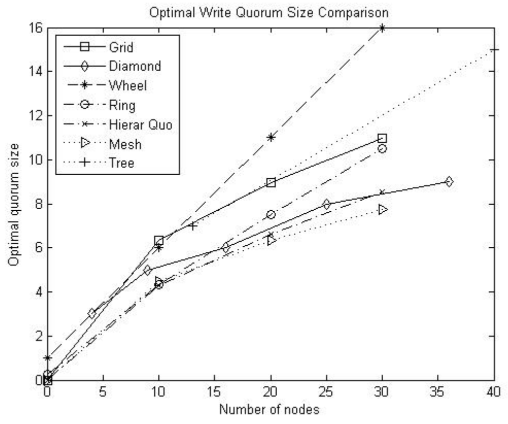
(b) Optimal Quorum Size

Fig. 4: Read Quorum Comparison

Figure 4. shows that wheel protocol has best read quorum size in optimal as well as worst case. Diamond has comparable read quorum size in optimal case but size increases upto $\lceil \sqrt{(2N)} \rceil$ in worst case. Similarly ring is comparable in worst case but not in optimal case. Figure 5 shows optimal and worst write quorum comparison.



(a) Worst Quorum Size



(b) Optimal Quorum Size

Fig. 5: Write Quorum Comparison

4.2 Message Overhead

This section presents message overhead analysis of different quorum protocols. Analysis is based on model and message overhead relations used in [15]. For the purposes of our study, we consider only the best possible implementation (the one with the least number of messages). The proportion of update operations in the load is represented by w. A small w indicates that there are generally few write operations in the system. For instance, the workload can have many queries (read-only transactions) and few update transactions. Each of these few update transactions, however, can have many write operations. With this, assuming that a transaction contains on average o_w write operations, the message overhead for wheel protocol are given below.

Message overhead for point-to-point is given as:

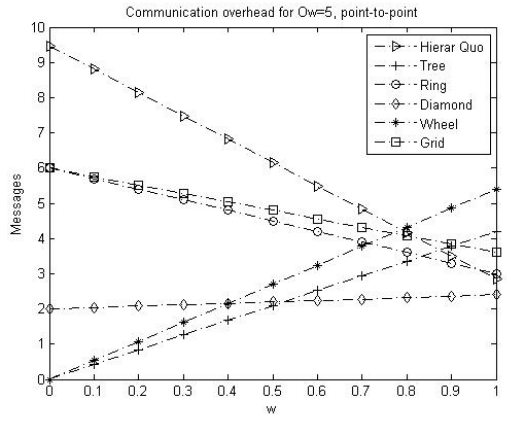
$$msg = \frac{3w\lceil\frac{n+1}{2}\rceil}{o_w} \quad (1)$$

Message overhead for multicast is given as:

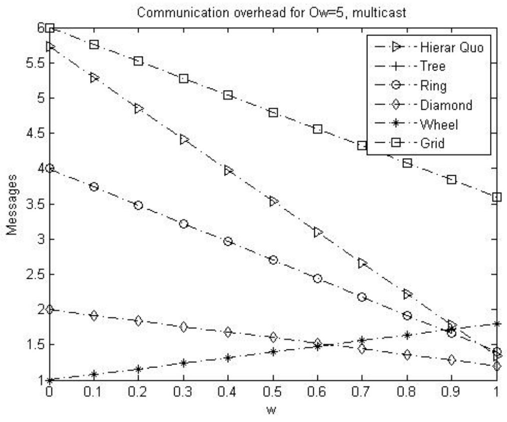
$$msg = w\frac{\lceil\frac{n+1}{2}\rceil + 2}{o_w} + (1 - w) \quad (2)$$

Comparison among different protocols message overhead for point-to-point is shown in figure 6(a). Minimum communication overhead is achieved by tree and wheel protocols. Overhead increases with increasing value of w. Tree shows better performance than wheel protocol, whereas, its quorum size is larger than wheel protocol. Wheel protocol, ensures read quorum size of 1 even in worst case, whereas, it becomes as big as $(d+l)^h$ in tree. So, for read intensive applications, wheel gives both the advantages of smallest read quorum and smaller communication overhead.

In case of multicast figure 6 (b), wheel and tree incur minimum message overhead than other protocols. Overhead increases with increasing number of writes in transactions (i.e. w).



(a) Message overhead, point-to-point



(b) Message overhead, multicast

Fig. 6: Message Overhead

5. Related Work

In this section we compare the wheel protocol to other existing protocols for maintaining the consistency of replicated data. The simplest replica control protocol is the read-one write-all protocol, where a read operation is executed by reading any copy and a write, writes all copies of the object. In order to increase the fault-tolerance of write operations, voting protocols were proposed [16], [2], where write operations are not required to write all copies. In a failure free system, both the static and the dynamic protocols require read operations to access several copies. Dual Quorum [17] reduces size of both read and write quorum by not making them intersect and regular semantics are enforced by communication between both quorums. This increases its communication overhead, whereas Wheel has no overhead like this.

Finally, the notion of imposing logical structures on a network of sites has been proposed before to solve different problems. Maekawa [18] proposed imposing a logical grid on a set of sites to derive efficient $O(sqrtN)$ solutions for mutual exclusion. Agrawal and El Abbadi [19] proposed imposing a logical tree to solve the mutual exclusion problem using $O(logn)$ messages. This approach was extended to replica control protocols that use several logical structures imposed on set of copies . Kumar [13] constructs a logical tree on a set of copies, where the copies actually correspond to the leaves of the tree. This results in a protocol where read and write quorums are of size $N^{0.63}$. Agrawal and El Abbadi [20] imposed a logical tree on the set of nodes and reduced read size to one when there is no fault. But read quorum size increases with failures to $\lceil\frac{N+1}{2}\rceil$ and write quorum size is $[(d+1)^{h+1} - 1]/d$. Storm [21] proposes a flexible hetrogeneous quorum based which is again based on on tree shaped voting structutes. Triangular mesh protocol [8], in which the nodes in the system are organized into a triangular mesh which has a quorum size of $O(\sqrt{2N})$. Our protocol draws on many of these ideas, and extends them to develop an efficient and fault-tolerant replica control protocol. The distinguishing feature of our approach is that we directly address the issue of low cost read operations, and unlike other logical structure based approaches, the wheel quorum protocol, in a failure-free system or with failure does not require read operations to access more than one copy.

6. Conclusion

In this paper we have proposed a new fault tolerant protocol for replicated data control in which the read quorum is constant of size one, without incrementing the write quorum. The design of the protocol directly addresses one of the main problems of replicated data: the necessity of read operations to access several copies in order to ensure the fault-tolerance of write operations. In case of failure, the wheel protocol continues executing both read and write operations with a high probability, although cost of execution is higher. Wheel protocol provides smallest quorum size with minimum message overhead. Message overhead for wheel protocol [22] multicast never exceeds 2, infact for lesser number of writes its smaller than 1.5. In particular, our protocol performs well in systems where read operations are requested more frequently than write ones.

References

[1] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso, "Understanding replication in databases and distributed systems," in *In Proceedings of 20th International Conference on Distributed Computing Systems (ICDCSŠ2000*, 2000, pp. 264–274.

[2] H. Gifford, "Weighted voting for replicated data," *in Proceedings of 7th Symposium on operating Systems, ,ACM*, pp. pp 150–162, 1979.

[3] R. H. Thomas, "A Majority consensus approach to concurrency control for multiple copy databases," *ACM Trans. Database Syst.*, vol. 4, no. 2, pp. 180–209, June 1979. [Online]. Available: http://dx.doi.org/10.1145/320071.320076

[4] M. L. Liu, D. Agrawal, and E. A. Abbadi, "Abbadi. On the implementation of the quorum concensus protocol," in *In Proc. Parallel and Distributed Computing Systems*, 1995. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.36.6898

[5] A. E. A. Divyakant Agrawal, "The tree quorum protocol: An efficient approach for managing replicated datain," *Proceedings of the 16th International Conference on Very Large Data Bases (1990),*, pp. pp. 243–254., 90:.

[6] A. W.-C. Fu, Y. S. Wong, and M. H. Wong, "Diamond quorum consensus for high capacity and efficiency in a replicated database system," *Distrib. Parallel Databases*, vol. 8, pp. 471–492, October 2000. [Online]. Available: http://dl.acm.org/citation.cfm?id=360808.360834

[7] N. C. MendonÃČÂğa and R. O. Anido, "The hierarchical ring protocol: An efficient scheme for reading replicated data," Department of Computer Science, University of Campinas, Tech. Rep. DCC-93-02, February 1993, in English, 30 pages.

[8] Y.-J. Chang, "A triangular-mesh-based approach to fault-tolerant distributed mutual exclusion," Master's thesis, National Sun Yat-sen University, June, 1995.

[9] M. H. A. S. Y. Cheung and M. Ahamad, "The grid protocol: A high performance scheme for maintaining replicated data," *IEEE Transactions on Knowledge and Data Engineering*, vol. Vol. 4, No. 6, pp. pp. 582–592, Dec. 1992.

[10] Y.C.Kuo and S. Huang, "A geometric approach for constructing coteries and k-coteries," *IEEE Transaction Parallel and Distributed Systems*, vol. 8(4), p. 402 411, April 1997.

[11] R. D. Schlichting and F. B. Schneider, "Fail-Stop Processors: An Approach to Designing Fault-Tolerant Computing Systems," *Computer Systems*, vol. 1, no. 3, pp. 222–238, 1983. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.20.4967

[12] P. A. Bernstein and N. Goodman, "A proof technique for concurrency control and recovery algorithms for replicated databases," *Distributed Computing, Springer- Verlag*, vol. 2(1):32-44, January 1987.

[13] D. Agrawal and A. E. Abbadi, "The Tree Quorum Protocol: An Efficient Approach for Managing Replicated Data," in *VLDB '90: Proceedings of the 16th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990, pp. 243–254. [Online]. Available: http://portal.acm.org/citation.cfm?id=645916.671977

[14] A. Kumar, "Hierarchical quorum consensus: a new algorithm for managing replicated data," *Computers, IEEE Transactions on*, vol. 40, no. 9, pp. 996–1004, 1991. [Online]. Available: http://dx.doi.org/10.1109/12.83661

[15] R. Jiménez-Peris, M. Patiño-Martínez, G. Alonso, and B. Kemme, "Are quorums an alternative for data replication," *ACM TRANSACTIONS ON DATABASE SYSTEMS*, vol. 28, p. 2003, 2003.

[16] S. Jajodia and D. Mutchler., "Dynamic voting." *in Proceedings of the ACM SIGMOD Internationat Conference on Management of Data,*, pp. pages 227–238,, June 1987.

[17] L. Gao, M. Dahlin, J. Zheng, L. Alvisi, and A. Iyengar, "Dual-quorum: A highly available and consistent replication system for edge services." *IEEE Trans. Dependable Sec. Comput.*, vol. 7, no. 2, pp. 159–174, 2010.

[18] M. Maekawa, "A sqrt(N) algorithm for mutual exclusion in decentralized systems," *ACM Trans. Comput. Syst.*, vol. 3, no. 2, pp. 145–159, May 1985. [Online]. Available: http://dx.doi.org/10.1145/214438.214445

[19] D. Agrawal and A. E. A. A. Efficient, "An efficient solution to the distributed mutual exclusion problem." *In Proceedings of the Eighth ACM Symposium on Principles of Distributed Computing,*, pp. pages 193–200,, August 1989.

[20] D. Agrawal and A. El Abbadi, "The generalized tree quorum protocol: an efficient approach for managing replicated data," *ACM Trans. Database Syst.*, vol. 17, pp. 689–717, December 1992. [Online]. Available: http://doi.acm.org/10.1145/146931.146935

[21] C. Storm, T. Warns, and O. Theel, "Flexible heterogeneous strict quorum-based dynamic data replication schemes," in *Proceedings of the 2008 14th IEEE Pacific Rim International Symposium on Dependable Computing*, ser. PRDC '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 200–207. [Online]. Available: http://dx.doi.org/10.1109/PRDC.2008.49

[22] M. T. P.Pandey, "Message overhead analysis of quorum protocol," *in Proceedings of International Conference on Advances in Computing*, vol. 174, pp. pp 237–245, 2012.

Incentive Scheme for P2P Live Streaming Systems Being Aware of the Upload Capability of the Participants

Shogo KANDA[1] **and Satoshi FUJITA**[1]
[1]Department of Information Engineering, Hiroshima University
Higashi-Hiroshima, 739-8527, Japan

Abstract— *In this paper, we propose an incentive scheme for P2P live streaming systems being aware of the upload bandwidth of the participants. The basic idea of the scheme is to combine a point-based incentive scheme with the notion of minimum guaranteed services. The performance of the proposed scheme is evaluated by simulation. The result of simulation indicates that: 1) the proposed scheme increases the utility of poor peers by 30% compared with Sepidar which is a typical auction-based incentive scheme and 2) compared with the Chu's scheme, which is a typical taxation-based scheme, the proposed scheme gradually increases the utility of rich peers as the amount of contributions increase, whereas the utility under the Chu's scheme does not change after reaching the limit determined by the taxation rate.*

Keywords: Peer-to-Peer live streaming systems, auction-based incentive scheme, taxation scheme.

1. Introduction

Recently, Peer-to-Peer (P2P) technology has been used in many fields as a way of realizing scalable network services. Voice over IP such as Skype[1] and video streaming services such as PPLive[2] and PPStream[3] are representatives of such applications. Among those applications, in this paper, we focus on the live streaming based on the P2P technology which is generally referred to as the **P2P live streaming**. In P2P live streaming systems, each peer (node) participating in the system is encouraged to *contribute* its communication bandwidth to the system, so that a copy of data received from an adjacent peer is uploaded to another adjacent peer by using the upload bandwidth. In other words, the system is designed so that the increase of the number of participants also increases the total amount of upload bandwidth.

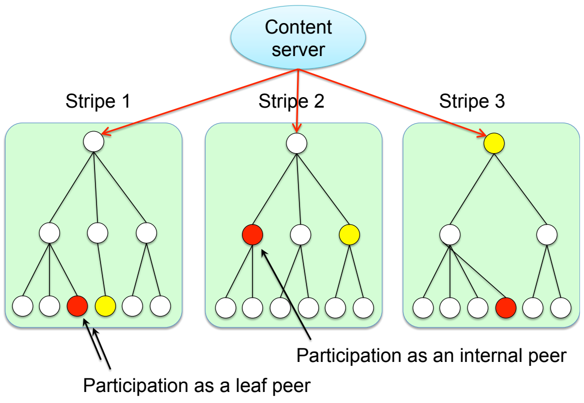
However, the performance of such P2P systems is severely affected by the behavior of each participant, since it causes an overload of specific peers and the degradation of provided services if the number of *free-riders* which do not contribute to the system increases. To overcome such a situation, P2P systems should have an **incentive scheme** which strongly encourages the participants to provide their resources as much as possible. Conventional incentive schemes for P2P live streaming systems are designed so that a peer contributing to the system can receive a high quality service in return for the contribution. Such a mechanism works well for the peers to have enough resources. However, for the peers to have small amount of resources, it is not user-friendly since it (strictly) differentiates available services by the amount of resources held by the peer, i.e., it discourages many "poor" peers to participate in the system to contribute as an uploader.

In this paper, we propose an incentive scheme for P2P live streaming systems such that all contributors become happy. In other words, we determine the return of contribution so that every peer can receive a return even if the amount of contributions is small as long as it repeats contributions, and it can receive a larger return as the amount of contributions increases. The basic idea of the scheme is to combine a point-based incentive scheme with the notion of minimum guaranteed services. More concretely, we design the scheme so that: 1) the given live stream is divided into k sub-streams which are delivered to the participants using different trees, where we assume the existence of an appropriate encoding method such that the original stream is decoded from any subset of sub-streams while the quality of the decoded stream depends on the number of sub-streams; 2) we differentiate the number of sub-streams acquired by each peer according to the amount of contributions; and at the same time, 3) we guarantee the minimum service for every peer by allowing peers to participate in at least R_d trees, where the value of R_d is dynamically updated according to the change of the status, such as the number of participants and the total amount of bandwidth.

The performance of the proposed scheme is evaluated by simulation. The result of simulation indicates that: 1) the proposed scheme increases the utility of poor peers by 30% compared with Sepidar which is a typical auction-based incentive scheme for P2P live streaming, and 2) compared with the Chu's taxation scheme, which is a typical taxation-based scheme, the proposed scheme gradually increases the utility of rich peers as the amount of contributions increase, whereas the utility under the Chu's scheme does not change after reaching the limit determined by the taxation rate.

The remainder of this paper is organized as follows. Section 2 describes preliminaries and Section 3 overviews related works. Section 4 describes the details of the pro-

[1]URL: http://www.skype.com
[2]URL: http://www.pplive.com
[3]URL: http://www.ppstream.com



Content server
Stripe 1
Stripe 2
Stripe 3
Participation as an internal peer
Participation as a leaf peer

Fig. 1: Multi-tree structure.

posed scheme. Section 5 describes the result of simulations. Finally, Section 6 conlcudes the paper with future works.

2. Preliminaries

In P2P live streaming systems, a given stream is delivered to the subscribers through a logical network called P2P overlay. In this paper, we are particularly interested in the *multi-tree structure* used in SplitStream [1] as the underlying P2P overlay, because of the ease of the maintenance compared with mesh structured overlays and the high fault tolerance compared with a single tree. In multi-tree structured P2P live streaming systems, each stream is divided into several sub-streams called **stripes**, and those stripes are delivered through different trees. Figure 1 illustrates the delivery of a stream through multi-trees. The split of a stream into stripes is done by encoding schemes such as MDC [3], in such a way that: 1) any subset of stripes can be decoded into a stream and 2) the quality of the resulting stream monotonically increases as the number of stripes increases.

In order to decode a high quality stream, each peer tries to participate in as many trees as possible, while it can participate in at most one tree as an internal peer, where a peer with a child in a tree is called an **internal peer** and a peer with no child is called a leaf peer. In each tree, each internal peer forwards a stripe received from the parent to the children by using its upload bandwidth. In this paper, we assume that the upload of one stripe consumes one unit of upload bandwidth called **upload slot**, and we call the maximum number of upload slots used by a peer the **fan-out** of the peer. The reader should note that the fan-out of a peer is calculated by dividing the upload bandwidth of the peer by the bandwidth required for the upload of a stripe.

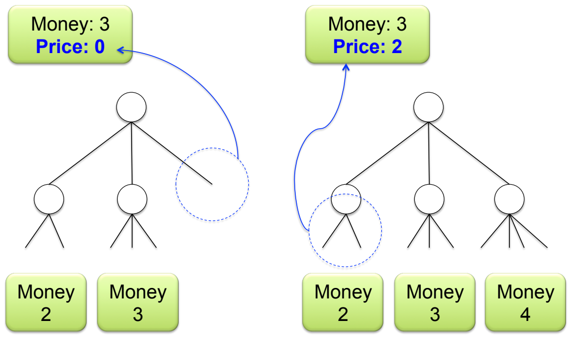


Fig. 2: Bidding in Sepidar.

3. Related Work

3.1 Auction-Based Incentive Schemes

There are many auction-based incentive schemes proposed for tree-based P2P live streaming systems. Sepidar [5] and Tan's scheme [8] are two representatives in this category.

The bidding procedure adopted in Sepidar is based on the notions of money and price. The money $m[i]$ held by peer i is the (declared) fan-out of the peer. The price $p[i]$ of peer i is set to be zero if it has an unused upload slot, and otherwise, $p[i]$ is set to be the minimum money over all children of i. For example, in Figure 2, the price of peer b is two since it has three slots connected to children with money 2, 3 and 4, respectively. Suppose that a peer a wishes to become a child of peer b. Then peer a initiates an auction and bids all of its money $m[a]$ for an upload slot of b. If the bid received from a is greater than $p[b]$, then peer b accepts a as a new child, and turns out a child with the least amount of money from the tree (by this operation, all descendants of the removed child will also be turned out from the tree and they should initiate another auction to subscribe to the stream). The reader should note that the above bidding mechanism gives incentives to all peers to declare a large fan-out so that they are not turned out by the other peers.

Tan's scheme conducts such a bidding periodically. It divides the playback time of the video stream into periods of fixed length as $T_0, T_1, T_2, \cdots$, and during each period, each peer earns reward points from its children as a consideration for the upload of the video stream. More concretely, in the ℓ^{th} period, a peer j receives all reward points from child i which were earned by peer i in the $(\ell-1)$st period. Reward points earned at the beginning of the ℓ^{th} period are used for the bidding for an upload slot for the $(\ell+1)$st period. The bidding within the ℓ^{th} period proceeds by repeating bidding round in the following manner:

1) Participants of each bidding round are peers which did not find the parent for the next period.

2) Each participant a randomly selects an internal peer b which has already found the parent and has enough upload slots, and bids all reward points earned at the beginning of the period for an upload slot of b.
3) Among submitted bids, peer b selects bidders with the largest bids as the winners and makes them children in the next period.
4) All losers of the bidding receive a list of winners from peer b, so that it could be used as the candidate for internal peers in the next bidding round.

In the Tan's scheme, each peer can participate in any number of trees as an internal peer, while after exhausting the upload slots, it should participate in each of the remaining trees as a leaf peer. This indicates that it may happen a situation such that a loser of an action cannot find a peer to have enough upload slot within a period. In such a case, it should try to find an internal peer by traversing the tree from the root to leaves in a best effort manner.

3.2 Taxation Scheme

Chu *et al.* point out that auction-based incentive schemes described in the last subsection would cause a significant gap of the quality of received services due to the difference of the amount of resources intrinsically held by each participant, and propose a taxation scheme to overcome this issue [2].

The key idea of the Chu's taxation scheme is to combine the taxation with a fixed tax rate $t\ (> 1.0)$ with the notion of basic income called demogrant. The maximum number of trees in which a peer can participate is a linear function of the amount of contribution of the peer. More concretely, to participate in r_i different trees, peer i must contribute at least $f_i = t \times r_i$ upload slots in a tree. By consuming $\sum_i \lfloor f_i/t \rfloor$ upload slots among collected $\sum_i f_i$ slots, there remain *internal reserves* of amount $\sum_i (f_i - \lfloor f_i/t \rfloor)$ in the system, which are equally redistributed over all participants (independent of the magnitude of contribution) as a demogrant. With such a redistribution mechanism, the quality of services received by poor peers increases, which relaxes the gap of the quality of services caused in the auction-based schemes. An apparent drawback of the Chu's taxation scheme is the lack of incentives for rich peers, because they will not contribute more than $f_i = t \times r_i$ slots, even if they have more resources.

4. Proposed Scheme

4.1 Overview

Most of existing incentive schemes for P2P live streaming systems are designed so that a heavy contributor can enjoy high quality services. In other words, the quality of received services increases as the amount of contribution increases. However, a naive application of such a *survival of the fittest* approach discourages peers with few resources to participate in the system since they always miss high quality services.

To overcome such an issue, in the proposed scheme, we combine the notion of minimum guaranteed services with a point-based incentive scheme. More concretely, the basic idea of the proposed scheme is: 1) to differentiate the quality of service according to the amount of contribution using a point-based scheme, and at the same time, 2) to guarantee the amount of minimum service for every peer by allowing peers to participate in at least R_d trees (the role of parameter R_d is similar to the demogrant used in the Chu's taxation scheme, but it is calculated in a different manner). This idea is inspired by the Weber-Fechner law which states that the just-noticeable difference between two stimuli is proportional to the magnitude of the stimuli, or an increment is judged relative to the previous amount. In our case, the judged quality of a live stream is proportional to the *logarithm* of the number of subscribed stripes concerned with the stream, which indicates that in order to increase the overall utility, we need to increase the number of stripes subscribed by "poor" peers enjoying low quality streams, by decreasing the number of stripes subscribed by "rich" peers enjoying high quality streams.

In the following, after clarifying the underlying P2P architecture in Section 4.2, we describe the details of the point-based bidding procedure in Section 4.3. We then describe the way of tuning the value of R_d in Section 4.4 and describe the way of carrying over unused points in Section 4.5.

4.2 P2P Architecture

We consider a P2P system consisting of N homogeneous peers, a content server and a point server. Peers are assigned a unique ID and can subscribe to a stripe by participating in a tree associated with the stripe. Each peer periodically earns k **reward points** from the point server by forwarding stripes to k children, and as will be described later, such earned points are used for the **bidding** for the upload slot of an internal peer in another tree. The interval of earning points is called the earning interval. Reward points unused in an earning interval are stored at the point server, to encourage poor peers to earn reward points for the future use. We use symbol $\sigma[i]$ to denote the reward points currently held by peer i. The way of managing stored reward points, which is mandatory to avoid the inflation of the points, will be described in Section 4.5.

For each tree, the content server keeps tracks of the ID of the root of the tree, and maintains the following two sets InP and LP representing the availability of the upload slots in the tree, which can be referred by all peers in the system:

- InP is the set of internal peers to have an available upload slot; i.e., peers whose fan-out is not exhausted.
- LP is the set of leaf peers which have already participated in more than R_d trees, where R_d is the minimum number of trees guaranteed by the system. Peers in LP are ordered in a non-increasing order of $\sigma[\cdot]$.

4.3 Bidding Procedure

In the proposed scheme, each peer can participate in at most one tree as an internal peer, while it can participate in any number of trees as a leaf as long as no conflict occurs. Each peer a which wishes to join a tree first refers to the InP of the tree. If it is not empty, it completes the join after becoming a child of a peer in the InP, and if it is empty, it conducts one of the following operations depending on the (expected) role in the tree:

1) If peer a joins as an internal peer, then after identifying a leaf peer b in the tree, it inserts itself between b and its parent, i.e., it becomes the parent of b and a child of the former parent of b.
2) If a joins the tree as a leaf peer, then it refers to the LP of the tree, and if it is empty, a gives up the join at this moment (it retries to join after waiting a certain time from 10 to 15 sec). Otherwise, after selecting a peer with the smallest $\sigma[\cdot]$ from the LP, a initiates an auction for the upload slot of the parent b of the peer.

The auction proceeds as follows. For each peer i, let $r(i)$ denote the number of trees in which peer i currently participates and $C(i)$ denote the set of children of i in the tree we are currently considering. The procedure separately considers the case of $r(a) \geq R_d$ (Case 1) and the case of $r(a) < R_d$ (Case 2).

Case 1) In this case, in order to keep that all children of peer b participate in at least R_d trees, peer a competes with peers in set $\{i \in C(b) : r(i) > R_d\}$ for the upload slots of b. Each player i bids $\sigma[i]$ reward points, and after receiving all bids, b selects peers which submitted the largest bids as the winners, and makes them as its new children. Each winner i pays 5% of its reward points to the system and each looser tries to join a tree after passing a certain time.

Case 2) In this case, we modify the procedure for Case 1 so that peer a is removed from the set of competitors. In other words, peer a can always join the tree as a leaf peer without paying reward points unless the LP of the tree is not empty.

4.4 Minimum Guaranteed Trees

The above procedure is designed to guarantee that every peer participates in at least $R_d(\geq 1)$ trees. However, an appropriate value of R_d would change according to the join and the leave of peers. In order to reflect such a dynamic change to R_d, in the proposed scheme, the value of R_d is periodically updated by the content server in the following manner:

- The server identifies the set of internal peers with their fan-out and the set of leaf peers with their reward points for each tree. Such an identification is realized by collecting information from participants through tree edges. The reader should note that InP and LP concerned with each tree can also be constructed using the collected information.
- Let F be the estimated fan-out of the overall system which is obtained by summing the fan-out over all trees. Let N be the estimated number of participants and ΔN be the number of peers which newly joined the system during the current update period.
- Using those values, the server updates R_d according to the following rule: if $F > 1.5 \times (R_d + 1)N$ then increment R_d by one and if $F \leq 1.25 \times R_d(N + \Delta N)$ then decrement R_d by one.

Table 1: Parameters.

Number of peers N	2000
Simulation time T	3600 sec
Bitrate of stream	400Kbps
Number of stripes	8
Earning interval of points	10 sec
Update interval of R_d	30 sec

After that, the content server broadcasts a message containing the new value of R_d to all peers to update the variable locally held by each peer. The behavior of each peer after receiving the update is as follows: When R_d increases, a peer participating in less than R_d trees tries to join a new tree (to become the participant of at least R_d trees) after waiting a certain time. When R_d decreases, it does nothing until it becomes a looser of an auction.

4.5 Carry-Over of Earned Points

In the proposed scheme, each peer periodically earns reward points from the system by serving as an uploader and bids those points for an upload slot of the other peers in the same bidding period. If it wins, it pays 5% of the bidded points to the system, and it carries over the remaining points for the future use. However, if we allow an unlimited carry-over of the remaining points, it causes the inflation of reward points which enlarges the gap between rich peers and poor peers. To overcome such an issue, in the proposed scheme, we bound the amount of carry-over of each peer i by the following value:

$$\theta_i \stackrel{\text{def}}{=} 10 \times \log_{1.1} p[i].$$

With this mechanism, even poor peers to have few upload slots could earn sufficient amount of points by repeating contributions, and simultaneously, it could avoid rich peers to be a sole winner for a long time.

5. Evaluation

5.1 Setup

We evaluate the performance of the proposed scheme by simulation. In the simulation, all peers are synchronized to the global clock, and within a step of the clock corresponding to one second, each peer completes any procedure

Table 2: Distribution of upload bandwidth.

Upload bandwidth [KB/sec]	Fan-out	Percentage of peers
50-399	1-7	70%
400-599	8-11	11%
600-799	12-15	11%
800-999	16-19	4%
1000-1199	20-23	4%

including the join to a tree and the bidding for an upload slot. Parameters used in the simulation are summarized in Table 1. According to the observation shown in [7], we assume that the upload bandwidth (and fan-out) of each peer follows the distribution shown in Table 2. Peers arrive at the system according to a Poisson distribution with mean 1 [peer/sec], and upon arrival, each peer is assigned a longevity t according to the following distribution [6]:

$$F(t) = 1 - 1.23 \times e^{-(\frac{t}{1572})^{0.59}}.$$

Each peer does not leave until longevity exhausts, and the behavior of the peer is not affected by the rest of its life.

Each peer tries to participate in all trees, and if it participates in a tree as an internal peer, it uses all of its fan-out (determined as in Table 2) for the other peers in the tree. Let $s_{i,j}$ denote the number of stripes subscribed by peer i in the j^{th} step. Then, the **utility** u_i of peer i during simulation of length T is defined as

$$u_i \overset{\text{def}}{=} \log_8 \left(\sum_{j=1}^{T} \frac{s_{i,j}}{T} \right),$$

where the reason of taking logarithm is that we are assuming the Weber-Fechner law as was described in Section 4.1 and we use eight as the base of the logarithm to normalized it in the range of $[0, 1]$.

Each run of the simulation consists of five consecutively executed sessions of 3600 sec and we conducted 50 independent runs to take an average of them. Carry-over of reward points (shown in Section 4.5) takes place within each run and the assignment of upload bandwidth is fixed in each run.

5.2 Comparison with Other Schemes

At first, we compare the performance of the proposed scheme with previous schemes described in Section 3. A comparison with Sepidar is illustrated in Figure 3. The horizontal axis is the fan-out of peers and the vertical axis is the average utility over all peers to have a given fan-out. This result indicates that compared with Sepidar, the proposed scheme certainly increases the utility of poor peers and the amount of increase is about 30% when the number of upload slots is small. In addition, the average utility over all peers is 0.87 in the proposed scheme, which significantly increases the average utility of 0.75 in Sepidar.

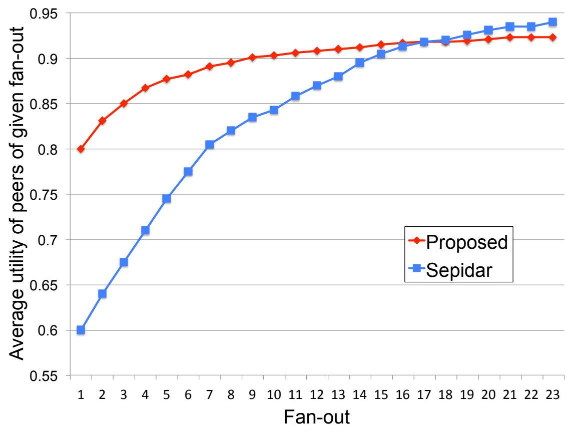


Fig. 3: Comparison with Sepidar.

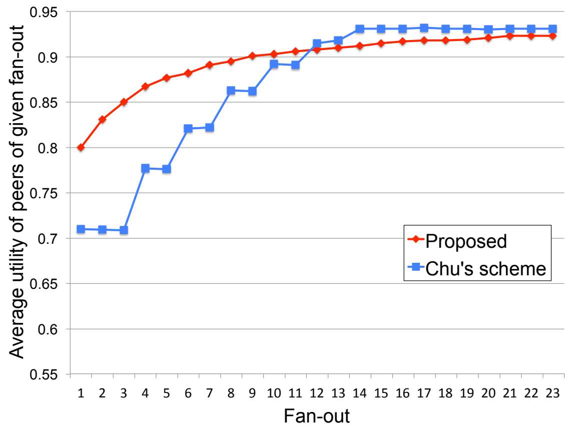


Fig. 4: Comparison with the Chu's taxation scheme.

Next, we compare the performance of the proposed scheme with the Chu's taxation scheme. Figure 4 illustrates the result. In the taxation scheme, the utility of rich peers does not change after the fan-out reaches 14. In other words, under this scheme, there are no incentives for rich peers for the further contribution when the number of contributed slots exceeds 14. In contrast to that, in the proposed scheme, the utility of rich peers gradually increases as the fan-out increases, which indicates that it certainly gives incentives to rich peers for the contribution of their resources.

This figure also shows that the absolute value of the utility of rich peers is smaller than the utility of rich peers in the taxation scheme, which is because of the following reasons. In the proposed scheme, the number of trees in which a peer can participate is determined by the bidding of reward points held by the peer. Hence even if it has a large fan-out, it could not join a sufficient number of trees if it has less reward points than the other peers. On the other hand, under

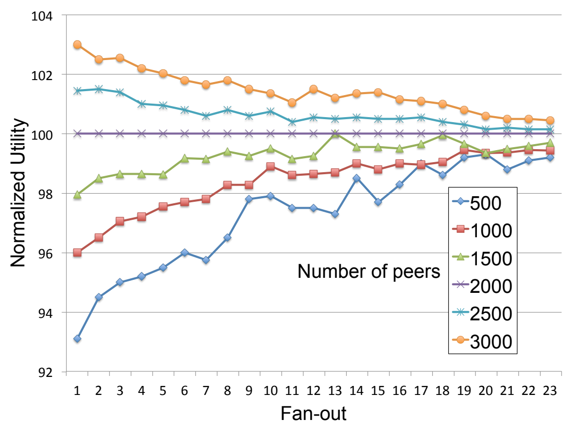


Fig. 5: Impact of the number of participants to the performance of the proposed scheme.

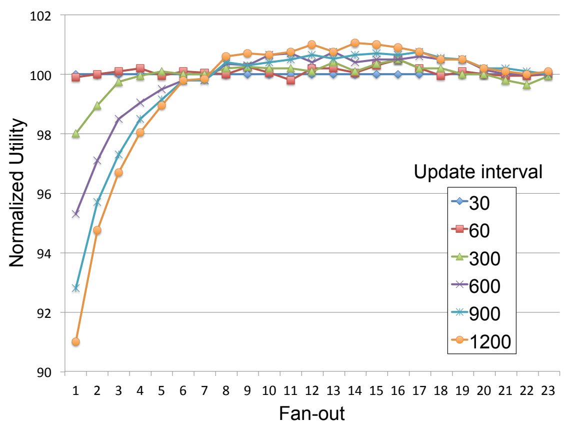


Fig. 6: Impact of the update interval of R_d to the performance of the proposed scheme.

the Chu's scheme, the participation in a tree is controlled by the content server, and each peer contributing f_i upload slots participates in $r_i(= \lfloor f_i/t \rfloor)$ trees with high probability.

5.3 Impact of the Number of Participants

Next, we evaluate the impact of the number of participants to the performance of the proposed scheme. In the simulation, we vary the number of participants from 500 to 3000 by keeping the distribution of peers for each fan-out as shown in Table 2. The result is illustrated in Figure 5. The vertical axis of the figure is normalized by the utility in the case of 2000 participants. This figure indicates that for any fan-out, the utility of peers increases as the number of participants increases, and the amount of increase is large for the peers to have small fan-out. The increase of the utility is apparently because of the increase of the number of available upload slots caused by the increase of the number of participants. In addition, a large increase for the peers with small fan-out is due to the low utility compared with the peers with large fan-out as was shown in Figure 3.

5.4 Impact of Two Intervals to the Performance

In this subsection, we evaluate the impact of two intervals used in the proposed scheme to the performance, i.e., earning interval described in Section 4.2 and the update interval of parameter R_d described in Section 4.4.

Figure 6 shows the impact of the update interval to the utility (recall that in previous subsections, this interval was fixed to 30 sec as shown in Table 1). Each curve in the figure corresponds to an update interval, which is ranged from 30 sec to 1200 sec, and the vertical axis is the utility normalized by the result for 30 sec, i.e., "value 100" corresponds to the utility at 30 sec. As the interval increases, the utility of peers with small fan-out rapidly decreases, while that of peers with moderate fan-out slightly increases. Such a badness for poor

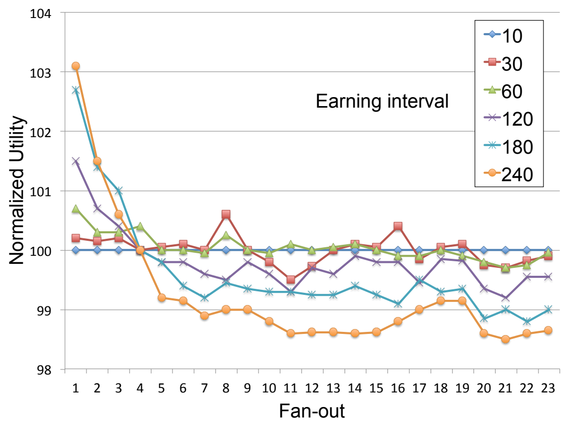


Fig. 7: Impact of the earning interval to the performance of the proposed scheme.

peers is due to the way of updating R_d. More specifically, even if there are enough upload slots due to the arrival of new peers, the scheme increments R_d one-by-one in each interval, which significantly loses the benefit of the minimum guarantee in the proposed scheme.

Finally, we evaluate the impact of the earning interval to the performance. The result is shown in Figure 7. The vertical axis of the figure is the utility normalized by the result for 10 sec, as before. The increase of the earning interval decreases the utility of most of the peers to have fan-out of more than four, while it increases the utility of peers to have small fan-out of less than four. This phenomenon could be explained as follows. A long earning interval reduces the chance of earning reward points for all peers. In addition, each winner should pay 5% of the reward points and the points which can be carried over to the next earning interval is bounded by a logarithm of the current point (see Section 4.5). Thus by increasing the earning interval, the competitive power of a peer reduces if it has large fan-out and increases if it has small fan-out.

6. Concluding Remarks

This paper proposes an incentive scheme for P2P live streaming system which is aware of the upload bandwidth of each participant. The proposed scheme is a combination of an auction-based incentive scheme and a taxation scheme based on the notion of minimum guaranteed services. The result of simulation indicates that it could increase the utility of poor peers by 30% compared with a conventional auction-based scheme and could improve the shape of utility function of rich peers so that it gradually increases as the amount of contribution increases.

An important future work is to refine the scheme by considering the cost, such as the maintenance cost of data structures and the cost for securely exchanging reward points. Another issue is to refine the model of simulation so that it reflects the dynamic change of the environment.

Acknowledgements

This paper was supported by the Grant-in-Aid for Scientific Research No.24500082 of the JSPS.

References

[1] M. Castro, P. Druschel, A. M. Kermarrec, and A. Nandi. “SplitStream: High-Bandwidth Multicast in Cooperative Environments.” In *Proc. of the 19th ACM Symposium on Operating Systems Principles (SOSP ’03)*, 2003, pages 298–313.

[2] Y. H. Chu, J. Chuang, and H. Zhang. “A Case for Taxation in Peer-to-Peer Streaming Broadcast.” In *Proc. of ACM SIGCOMM Workshop on Practice and Theory of Incentives in Networked Systems*, 2004, pages 205–212.

[3] V. K. Goyal. “Multiple Description Coding: Compression Meets the Network.” *IEEE Signal Processing Magazine*, 18(5): 74–94, 2001.

[4] V. Padmanabhan, H. Wang, and P. Chou. “Resilient Peer-to-Peer Streaming.” In *Proc. of the 11th IEEE International Conference on Network Protocols (ICNP)*, 2003, pages 16–27.

[5] A. H. Payberah, F. Rahimian, S. Haridi, and J. Dowling. “Sepidar: Incentivized Market-Based P2P Live-Streaming on the Gradient Overlay Network.” In *Proc. of IEEE International Symposium on Multimedia*, 2010, pages 1–8.

[6] T. Silverston and O. Fourmaux. “Measuring P2P IPTV Systems.” ACM NOSSDAV ’07, 2007.

[7] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang. “The Feasibility of Supporting Large-Scale Live Streaming Applications with Dynamic Application End-Poins.” In *Proc. of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2004, pages 107–120.

[8] G. Tan and S. A. Jarvis. “A Payment-Based Incentive and Service Differentiation Scheme for Peer-to-Peer Streaming Broadcast.” *IEEE Trans. Parallel Distrib. Syst*, 19(7): 940–953, 2008.

On Deriving Mean Chord Length for Structured Overlay Network

Pankaj Anthwal
Centre for Security, Theory and Algorithmic Research
IIIT-Hyderabad
Hyderabad, Andhra Pradesh
India - 500032

Abstract— *With an aim to stress the need for a rigorous formalization of routing schemes for structured peer-to-peer overlay networks, in this work, we demonstrate the usefulness of a formal definition that we have come up with, by elaborating on one of the several network properties that are either determined or limited by the routing scheme that builds the network. After briefly presenting the intuition behind the necessity and sufficiency of our formal definition of a routing scheme for structured overlay networks, we elaborate on the notion of average path length for structured overlays and derive its expression in terms of the routing scheme parameters to show how the overlay network's average path length is determined by its routing scheme.*

Keywords: Theory, Routing, Overlay Networks, Average Path Length

1. Introduction

Structured Overlay Networks (SONs, for short) are abstractions over real physical networks, which allow imposition of any arbitrary structure on a highly dynamic and heterogeneous underlying network, by having the connections among their constituent logical nodes established *selectively*, independent of how the underlying network is organized. We use the term *routing scheme* to refer to the component of the system which is responsible for assigning a specific structure to the overlay network. Since the inception of Distributed Hash Tables (the most popular data structure used to implement structured overlays) in the early 2000s, several overlay properties like *degree, diameter, congestion, resilience, flexibility, load-balancing, latency,* etc., have been identified as evaluation criteria for the overlays. *We* are interested in exploring which ones among the lot are *routing scheme dependent* and how that determines the intrinsic relation among these properties - a task which, we believe, cannot be accomplished without having a rigorous **formal structure** define a generic routing scheme and its characteristics for SONs. Structured overlay networks have now been in use for nearly a decade, but the more we go through existing literature on DHTs, a majority of which deals with presenting specific routing algorithms, comparisons among specific geometries/algorithms, statistical analysis, experimental results, designing guidelines, etc., the tougher we feel it is, to compare the overlay routing strategies, and decide what network suits what needs best, leading us to believe that the principles on which routing in structured overlays works, are still not very well understood. With cloud computing at its peak and new ambitious uses of DHTs like next generation internet [26] being proposed, we are convinced that a theory of routing in SONs is required now more than ever. Our belief is reinforced by hints of interest in that direction indicated by recent works like [24] and [25]. For space limitations, we are going to keep the discussion on the *why* of the components, that, according to us, constitute the formal definition of a routing scheme for structured overlays very brief and informal in the section **The Framework**. Our major focus in this work is to demonstrate the usefulness of our definition by showing how the generic expression for one of the most practically relevant properties of a SON - its *average path length* - or as we like to call it - the **mean chord length** of the overlay - can be derived for an overlay in terms of the routing scheme definition components.

2. Related Work

A considerable amount of work, in the past decade, has gone into building various kinds of Distributed Hash Tables, each one with its own specific routing algorithm [1], [2], [3], [4], [5], [6], [7], [8], etc. Initially, one of the most explored aspects of DHTs was the *state-efficiency* trade-off they achieved. When [9] put forward several open questions like whether a logarithmic (in network size, N) routing state to achieve logarithmic network diameter (in N) was optimal for the DHTs, the effect of the state-efficiency trade-off on other network properties like *congestion, resilience, flexibility*, etc. was inevitably explored. [10], for instance, showed, as had been correctly speculated earlier by [9], that, even though latter DHTs like [6], [7], [8], etc., provided a negative answer to the state-efficiency trade-off question by achieving a logarithmic diameter in *constant* routing state, they did so at the expense of introducing *congestion* into the network. [11] analyzed the state-efficiency trade-off question in light of another parameter - *fault resilience* of the network - and carried out a graph-theoretic comparison of the routing performance and fault resilience of various DHTs. [12] examined the effect of *routing geometry* on network characteristics - *resilience* and *proximity*. The authors drew a clear demarkation between a *routing algorithm* and a *routing geometry* and chose to work with specific geometries. According to *their* definition, our work deals with routing algorithms. [18] and [19] carried out experimental analysis of specific DHTs with the former analyzing the effect of neighbour selection on resilience and the latter providing a performance versus cost framework for DHT designers to analyze and compare the various network parameters under *churn*. In surveys like [20] and [21] properties of various DHTs were analyzed and compared. And though theoretical frameworks like [13], [14], [15] constituting formal definitions for identifier spaces and routing table construction, have been presented, invariably all deal with only specific routing table construction algorithms. The list of related literature on frameworks, guidelines, design issues, network architecture, specific routing protocols, routing table constructions techniques, etc., is endless, but none of these, to the best of our knowledge, aims at coming up with a formal definition of the routing scheme for a SON that *determines* (rather than *depends on*) the routing geometry for the network which subsequently determines the network properties. Therefore, although *average path length* and *congestion* are no new concepts for overlay networks, evaluation of these properties has conventionally been carried out only for specific geometries unlike this work that presents generic derivations in terms of the generic routing scheme definition presented in [0]. The closest a work comes to ours is [24], where the authors present *stochastic modeling of routing in DHTs* using *renewal processes* and present upper bounds on *per hop routing progress* for classes of DHTs defined by them. The models, definitions, and results given in [24], however, are quite different from ours and the authors don't deal with the properties that we derive in this work.

3. The Framework

Informally, by the term *'routing scheme'*, we refer to the *set of rules* that determine - 1. the *addressing format* of the nodes in overlay network - the number and the type of symbols that form the address, 2. how the *routing table* - set of neighbours - for every node in the network, and hence the overlay topology, is constructed, and 3. the *forwarding logic* - how the next hop neighbour at a host node is chosen from its routing table, given a specific destination address. We refer to the overlay features which are determined by the routing scheme as the routing scheme's *basic characteristics*. In this work, we shall only deal with routing schemes that satisfy what we refer to as the ***reachability condition*** which requires that **the routing scheme should be able to deterministically route any message to *any destination* originating from *any source* in the network within a bounded number of steps**, i.e., every node must be *reachable* from every other node in the network within a bounded number of steps. A routing scheme that satisfies the *reachability condition* shall be referred to as a **complete routing scheme**.

3.1 The Generic Address Space

We restrict the scope of our definition to cover the routing schemes that operate with a class of address spaces characterized by three properties. *Firstly*, address spaces that are **totally ordered sets** on their members addresses. We refer to this property of our generic address space as ***complete indexability***. Since, in every totally ordered set, the set members can be *grouped* by their *relative order* w.r.t each other and that can be recursively repeated within each group to form smaller groups, it implies that a *structured recursive hierarchy* in which, instead of being a *set of addresses*, every constituent search space (of size > 1) is a ***set** of smaller search spaces*, can be defined on every complete indexable address space. We refer to each distinct level in the *search space hierarchy* as a **search level** or a **level** and to the smaller search spaces available as forwarding options at any search level as **blocks**. So, under our current scope, the generic address space of a generic routing scheme for structured overlays can always be organized into *levels* and *blocks*. And since every destination node (by its address) belongs to *some block* at every search level in the search space hierarchy for every source node, the routing scheme forms *directed connections* across blocks at all search levels using a function, ϕ_t - the *phase transition function*, to ensure progress towards the destination via a level by level block selection process. *Secondly*, we choose to deal with only *constraint-free address spaces*, spaces where the address digits for every member address can be chosen independent of each other. Such an address space with 'd' levels and 'b' blocks can be geometrically represented by a discrete and finite 'd'-dimensional newtonian space with each dimension size 'b'. We refer to this space, in which addresses can be seen as points with integral co-ordinates, as a d-address space. We also refer to ϕ_t as an *impulse* in the d-address space as its application on the host address to obtain the next hop address is what *displaces* the message across the address points in the address space. The displacement caused by ϕ_t at level i is captured by its *component along* i - ρ_{g_i} which computes the *next hop block* v_h' given the *host block* v_h and the *destination block* v_d at level i, i.e., $\rho_{g_i}(v_h, v_d) = v_h'$. And *thirdly*, we choose to deal with only *search uniform* 'ϕ_t's - functions that create a *uniform search space* by connecting blocks at all levels in the d-address space in the exact same manner, i.e., by using the same component ρ_g at all levels in the address space.

Each routing step in the overlay corresponds to a *block shift* - migration (of the message) from the block containing the source node to one of the next hop blocks which is closest to the destination block - at some level in our d-address space. In a uniform search space with blocks *stitched* together by ϕ_t, at any level i, we define the *distance* (all connections being directed, all distances are *asymmetric*) ***from*** block v_s ***to*** block v_d, denoted by $d_{\phi_t}(v_d, v_s)$, as the *minimum number of block shifts it takes for a message to reach from a node in block v_s to one in block v_d at level i*. For instance, with 4 blocks at level i connected by ϕ_t as - $v_0 \to v_4 \to v_7 \to v_2 \to v_0$ - $d_{\phi_t}(v_4, v_0) = 1$ while $d_{\phi_t}(v_2, v_0) = 3$. We showed that, at every level, there is exactly one shortest link of blocks on which both v_s as well as v_d appear. The *level diameter* - 'z' - is defined as the maximum distance between any two blocks in the same level which is also the maximum number of routing steps the scheme may take to carry a message from the i_{th} search level to the $(i+1)_{th}$ search level.

3.2 Definitions

*Definition 1 (**Address**):* In a *'d'-address space* with each dimension size 'b' denoted by $\mathbb{A}_{b,d}$, an **address** is defined as a **sequence of 'd' digits** in base **'b'**.

$A[i]$ denotes the value of the i^{th} digit in A. For a positive integer n, we use W_n to represent the set of whole numbers less than n. For any $w \subseteq W_d$, $A(w)$ denotes the subsequence of A corresponding to the indices in the set w. We say w is *an index set* (the same subsequence may have multiple index sets in the same parent sequence) of $A(w)$ in A. The size of a subsequence α is denoted by $|\alpha|$. For an address A in address space $\mathbb{A}_{10,10}$, let $A[i] = v_i$ and let α be the subsequence corresponding to index set $w = \{0, 1, 3, 6, 7\}$, then,

$$W_n = \{0, 1, 2, \ldots, n-2, n-1\}$$

$$\mathbb{A}_{b,d} = \{W_b\}^d$$

$$A = \langle v_0, v_1, \ldots, v_i, \ldots, v_{d-1}\rangle,\ v_i \in W_b$$

$$\alpha = A(w) = \langle v_0, v_1, v_3, v_6, v_7\rangle$$

$$|\alpha| = |A(w)| = |w| = 5$$

Though the indices in an address A_i correspond to the search levels in the search view of node n_i while the values at specific indices correspond to the blocks at the respective levels, the subtlety in the usage of these terminologies is explained under the bit *search space* and *address format* in the *Appendix*.

*Definition 2 (**Transition Function**):* We say any function of type $f : \mathbb{A}_{b,d} \to \mathbb{A}_{b,d}$ is a **transition function**, denoted by 'f_t', in $\mathbb{A}_{b,d}$, i.e., $f(A) = B$ for addresses $A, B \in \mathbb{A}_{b,d}$.

The transition function, denoted by 'f_t', can be used to create a directed link *from A to B*. From the message's point of view, we also refer to f_t as an *impulse* in $\mathbb{A}_{b,d}$ as, during the routing process, the message undergoes a transition from A to B in $\mathbb{A}_{b,d}$ via f_t (hence the name). We say address B is ***reachable*** from A in $\mathbb{A}_{b,d}$ through f_t if $\exists$ a positive integer k such that $f_t^k(A) = B$. By definition, the corresponding digits in any two addresses A and B should be given by $A[i]$ and $B[i]$ for any $i \in W_b$. But, an f_t, that may **shuffle** the digits (which may be required to create topologies like de-bruijn graphs) around in host address A_h to compute a next hop address A_{h+1}, may, as a result, *disturb the mapping* between the corresponding digits in A_h and A_{h+1} (and hence between A_{h+1} and A_d, the destination address). So, in order to we keep track of the *mapping between corresponding digits* in any host address A_h and the source address A_s during the routing process, we define an *index sequence* for a host address A_h w.r.t a source address A_s and a transition function f_t, that represents the *permutation* of indices in A_h w.r.t A_s that captures any reshuffling of digits f_t may cause.

*Definition 3 (**Index Sequence:**):* The **index sequence** in host address A_h w.r.t source address A_s and transition function f_t, denoted by $I_{seq}(A_h, A_s, f_t)$, is the **permutation** P of the sequence $\langle 0, 1, \ldots, d-1\rangle$ such that the i^{th} member of P, say P_i, represents the index that the digit $A_h[P_i]$ had in A_s.

For instance, let $A_s = A_0$ and $A_h = A_2$. Let $f_t : A_j[i] = A_{j-1}[(i+2)\%d],\ \forall i \in W_d$ be applied during the process $A_0 \to A_1 \to A_2$. The index sequence in any source address like A_0 is always $\langle 0, 1, \ldots, d-1\rangle$. Applying f_t twice on A_0 to get A_2 makes the index sequence at A_2 a '*left shifted by 4 units permutation*' of the initial sequence (at A_0) according to the definition of f_t, i.e. $I_{seq}(A_2, A_0, f_t) = \langle 4, 5, \ldots, d-1, 0, 1, 2, 3\rangle$. Note that, any $w \subseteq W_d$ has a ***unique*** *index set* in any index sequence. For any $w \subseteq W_d$ and any *index sequence* P, we use $I_{set}(w, P)$ to denote the *index set* the values of w have in index sequence P. For instance let $w = \langle 0, 2, 3, 7\rangle$ and let index sequence $P = \langle 7, 0, 1, 3, 6, 2, 5, 4\rangle$, then $I_{set}(w, P) = \{0, 1, 3, 5\}$.

*Definition 4 (**In Phase Subsequences:**):* Let, for addresses A and B in $\mathbb{A}_{b,d}$, $\alpha = A(w_1)$ and $\beta = B(w_2)$ for some sets $w_1, w_2 \subseteq W_d$. Subsequences α and β are said to be ***in phase*** w.r.t a transition function 'f_t' defined in $\mathbb{A}_{b,d}$ if,

1) $|w_1| = |w_2|$, and
2) $\exists$ a positive integer k_1 such that $f_t^{k_1}(A) = A'$, where (a) $I_{set}(w_1, I_{seq}(A', A, f_t)) = w_2$ and (b) $A'(w_2) = B(w_2)$
3) $\exists$ a positive integer k_2 such that $f_t^{k_2}(B) = B'$, where

(a) $I_{set}(w_2, I_{seq}(B', B, f_t)) = w_1$ and (b) $B'(w_1) = A(w_1)$

We say, 'β' is k_1-*in phase* with 'α' (while 'α' is k_2-*in phase* with 'β') w.r.t to 'f_t' in $\mathbb{A}_{b,d}$. Addresses A and B are said to be ***isomorphic*** w.r.t f_t in $\mathbb{A}_{b,d}$ if the entire sequence A is in phase with the entire sequence B, i.e., if $A(w_1)$ is *in phase* with $B(w_2)$ for $w_1 = w_2 = W_d$. That is to say, addresses A and B are isomorphic under f_t in $\mathbb{A}_{b,d}$, if B is reachable from A and vice-versa through *the same* transition function, f_t, i.e., if $\exists$ positive integers k_1 and k_2, such that $f_t^{k_1}(A) = B$ and $f_t^{k_2}(B) = A$. We say, B is the k_1-*isomorph* of A while A is the k_2-*isomorph* of B w.r.t f_t in $\mathbb{A}_{b,d}$.

*Definition 5 (**Phase Preserving Function**):* A *phase preserving function* 'ϕ_p' is a transition function with the additional property of being **self-invertible** in its address space $\mathbb{A}_{b,d}$, i.e., for every *phase preserving function* ϕ_p in $\mathbb{A}_{b,d}$,

1) $\phi_p : \mathbb{A}_{b,d} \rightarrow \mathbb{A}_{b,d}$, and
2) $\forall\ A \in \mathbb{A}_{b,d}$, $\exists$ integer $k_m > 0$ such that $\phi_p^{k_m}(A) = A$

where k_m gives the **period** of ϕ_p w.r.t A, generally denoted by $\tau_{\phi_p,A}$ or simply **per**(ϕ_p, A) with τ_ϕ denoting the maximum period of ϕ_p for any $A \in \mathbb{A}_{b,d}$. For address A_0 and phase preserving function ϕ_p defined in address space $\mathbb{A}_{b,d}$, with per(ϕ_p, A_0) = τ, an **isomorphic ring** for ϕ_p and A_0 in $\mathbb{A}_{b,d}$, R_{ϕ_p,A_0}, is defined as the sequence of addresses obtained by applying ϕ_p 'i' times on A_0 to get the i^{th} member of the sequence $\forall i \in W_\tau$, i.e., $R_{\phi_p,A_0} \leftarrow \phi_p^i(A_0) = A_i$, $\forall i \in W_\tau$, $R_{\phi_p,A_0} = \langle A_0, A_1, \ldots, A_{\tau-2}, A_{\tau-1} \rangle$.

*Definition 6 (**Phase Transition Function:**):* A ***phase*** *transition function* 'ϕ_t' is defined w.r.t to a ϕ_p in $\mathbb{A}_{b,d}$ as a transition function which *throws* ***some digits*** in any address $A \in R_{\phi_p,A_0}$ of *any ring* formed by ϕ_p in $\mathbb{A}_{b,d}$, *out of phase* with the other addresses in R_{ϕ_p,A_0} w.r.t ϕ_p, i.e., for any address $A \in R_{\phi_p,A_0}$, $\forall A_0 \in \mathbb{A}_{b,d}$, $\phi_t(A) = A'$ such that $A' \notin R_{\phi_p,A_0}$.

*Definition 7 (**Routing Step**):* For a route of length 'm' - number of distinct jumps the message makes from the source till the destination - between a source-destination pair in the network, the source and the destination addresses can be denoted by A_0 and A_m, respectively. We say the message transfer, $A_{i-1} \rightarrow A_i$, denoted by s_i, on the route - $A_0, A_1, \ldots, A_{m-1}, A_m$ - constitutes the 'i^{th}' ***routing step*** of the *routing process*.

Address A_i reached after the i^{th} routing step can be divided into *three non-overlapping* subsequences - ***resolved digits*** denoted by r_i - digits that are *in phase* with the *corresponding destination address digits* (ones with the same index sets at the beginning of the routing process) w.r.t ϕ_p after the 'i^{th}' routing step, ***target digit*** denoted by t_i - digit that shall undergo a ***phase transition*** (w.r.t the other digits in A) during the $(i+1)^{th}$ routing step, and ***unresolved digits*** denoted by u_i - digits that are neither resolved nor target digits at the current routing step. The unresolved digits can further be categorized into - *deterministic* and *probabilistic* - types. Values for unresolved digits in next hop addresses (neighbours in the routing table) in the former category are deterministically bound to the values of the corresponding digits in the host address whereas the ones in the latter category, may take any value (so long as they are unresolved) from I_b in the routing entries and referred to as **don't care digits** (until targeted). *Routing step* s_i,

$$\mathbf{s_i} : r_{i-1} \rightarrow r_i;\ t_{i-1} \rightarrow t_i;\ u_{i-1} \rightarrow u_i$$

4. The Generic Routing Scheme - 'Ψ'

Parameters 'b' and 'd' organize the N overlay nodes into a discrete, finite, constraint free d-dimensional space with each dimension of size b and the points (with integral coordinates) representing node addresses. Routing is reduced to a task of *moving* the message from a source point to some destination point in $\mathbb{A}_{b,d}$. Function ϕ_t stores impulses (makes connections via routing entries) at every point which can be used to launch the message in a specific direction with a specific magnitude. The number and nature of connections (hence the routing table size) to be made at a node depends upon the intermediate topology presented to ϕ_t by ϕ_p which makes the background space *dynamic* (captured by the intermediate topology) by making it participate in the routing process in a *deterministic* way. As the final bit, we use J_s to represent the function that determines the level-to-index mapping, i.e., *the* ***order*** *in which the address digits are resolved*, for the address. So, until there can be identified intrinsic routing aspects of structured overlays that are beyond the scope of parameters - b, d, ϕ_p, ϕ_t, and J_s - (in which case our definition will need to be extended), our formal definition of a generic routing scheme for SONs, denoted Ψ, can be represented by the five tuple $\langle b, d, \phi_p, \phi_t, J_s \rangle$. We now show how the Ψ parameters *determine* the - *average path length* and *inherent congestion* - for an overlay network built by Ψ.

4.1 Basic Routing Characteristics

The most general evaluation criteria for software systems includes the system's - **performance, cost, fault-tolerance** and **scalability**. Further, for a *distributed, decentralized* and *abstracted* system like a SON, additional features - **fairness, self-maintenance, abstraction overhead** become relevant. Based on various works like [16], [17], [20], [21], [22], [23], [27], etc., the various metrics on which SON features are evaluated are - (performance) *worst case hops, avg. hops, path latency*; (cost) *routing table size, forwarding complexity*; (fault-tolerance) *node/path* *failures tolerated*; (fairness) *data distribution, routing load distribution*; (self-maintenance) *recovery time, routing table size*; (abstraction overhead) *path stretch*; and (scalability) *manageable network size*. Of these, data distribution is the responsibility of the load balancing technique while the remaining metrics are determined/affected by the routing scheme. For that reason, we also refer to the network properties (listed below) evaluated on these metrics as the ***basic routing characteristics***. Well known network properties - *network diameter* and *node out-degree* - correspond to maximum no. of hops and routing table size, respectively, while more abstract notions like *static resilience* [12], *congestion* [10], and *robustness* [23], aim at capturing metrics related to fault-tolerance, fairness, and self-maintenance, respectively. We have come up with - ***strong*** static resilience, ***weak*** static resilience and ***inherent*** network congestion - as tighter definitions for the first two notions. As scalability is just a measure of how all other network properties are affected with a change in the network size while forwarding complexity can be safely assumed to never be a bottleneck metric, we define network properties for the remaining metrics - 1. **Mean Chord Length** - represents the *avg. no. of hops*, 2. **Inherent Network Congestion** - reveals the *distribution of routing load* among the nodes and the edges in the network, 3. **Network Flexibility** - limits *path latency, path stretch,* and *recovery time*, 4. **Network Tenacity** - quantifies ***strong*** *static resilience*, and 5. **Network Integrity** - quantifies ***weak*** *static resilience*. For the space available, we restrict ourselves to only the first property - the *mean chord length* of a SON - in this work.

Mean Chord Length: We refer to the average (expected) route length for a source-destination pair in a network G formed by routing scheme Ψ, as the *mean chord length* of the network. We go about the derivation by first computing the expected route length *for a specific source-destination pair* (n_s, n_d) (eqn. (5)) and then summing over the results for all such pairs in the network (as each distinct route in the network is built by Ψ independent of every other route in it) to derive it's mean chord length. In order to calculate the expected route length for pair (n_s, n_d), we need to estimate all possible paths from n_s to n_d. Now, since, for any source-destination pair (n_s, n_d), (addresses (A_s, A_d)) the next hop neighbours during the routing process are chosen *deterministically* from the routing table, i.e., the *index to be resolved* (that determines the row no.) and the *target value at the target index* (that determines the column no.) is known at every routing step, it implies that the *index sequence* - the order (by index) in which the address digits are targeted for resolution during the routing process - can be identified at the beginning of the routing process for (n_s, n_d). Let j_i denote the index in A_s that corresponds to the i^{th} level (which is, therefore, the i^{th} index to be resolved) in the search space and let k_i denote the *block distance* from the host block to the destination block at the i^{th} search level, i.e., if, with $K_i = \sum_{j=0}^{i} k_i$, $A_{s_i} = \phi_p^{K_{i-1}}(A_s)$ and $A_{d_i} = \phi_p^{K_{i-1}}(A_d)$ denote the source and destination address isomorphs after the ${k_{i-1}}^{th}$ routing step (immediately after the $(i-1)^{th}$ has been resolved) while j'_i (index of digit $A_s[j_i]$ in A_{s_i}) denotes the next index (in A_{s_i}) to be resolved, then $k_i = d_{\phi_t}(A_{d_i}[j'_i], A_{s_i}[j'_i])$. With j_0, A_d, and A_s known initially, we can compute $k_0 = d_{\phi_t}(A_d[j_0], A_s[j_0])$, the no. of routing steps Ψ takes to resolve $A_s[j_0]$. Further, since j'_1 and A_{d_1} can be computed by applying ϕ_p on j_1 and A_d, respectively, if the value $A_{s_1}[j'_1]$ (a function of ϕ_t and $A_s[j_1]$) is known, too, then computing $k_1 = d_{\phi_t}(A_{d_1}[j'_1], A_{s_1}[j'_1])$, becomes trivial, as well. And similarly, we can compute the entire *hop sequence* - $\langle k_0, k_1, \ldots, k_d \rangle$ - for pair (n_s, n_d) in network G (build by Ψ), where, for $i \in [0, d-1]$, k_i denotes the exact no. of routing steps required to resolve the i^{th} search level for (n_s, n_d) while $k_d = \tau(\phi_p) - (K_{d-1} \bmod \tau(\phi_p))$ denotes the no. of extra routing steps required to complete a period of ϕ_p (to bring the address space to its original configuration) *after* the last digit has been resolved.

Note that, till routing step 's', that completes resolution (locating the correct block) at the i^{th} level, exactly K_i routing steps are carried out. If the next target digit j'_i in host address A_{s_i} was a don't care digit during 's', then it may take any value from I_b in A_{s_i} with equal likelihood, otherwise, it equals $A_{s_i}[j'_i]$. In case of the non-deterministic shift in value of the current target digit during the preceding routing step, $(z+1)$ values for k_{i+1} (no. of steps that shall be required to resolve the current target) are possible, while in case of a deterministic shift only 1 value, $k_{i+1} = d_{\phi_t}(A_{d_i}[j'_i], A_{s_i}[j'_i])$, is possible. That implies that if Ψ uses non-deterministic allocation of initial values (before their resolution starts) to a total of x digits, i.e., if there are a total of x digits (excluding the first digit to be resolved because its value is *always* determined in the beginning) in every address which are *don't care digits* before a routing step turns them into *target digits*, a total of $(z+1)^x$ distinct hop sequences are possible for every source-destination pair in the network, where every distinct hop sequence represents a distinct *type* of path for the pair. Now, since, its Ψ that determines the *no. of don't care digits* and *their indices* in any address during the routing process, all the hop sequences for a given source-destination pair can be exhausted using Ψ. For instance, let $d = 4, z = 2$ and let the index sequence for pair (n_s, n_d) come out to be $\langle 1, 0, 3, 2 \rangle$. Further, let the $k_0 = 2, k_2 = 1$, and $k_4 = 0$ while indices 0 and 2 represent digits that are don't care before being targeted for resolution during the routing process. The $(2+1)^2 = 9$ hop sequences that exhaust all possible types of paths Ψ may built from n_s to n_d are - $\langle 2,2,1,2,0 \rangle$, $\langle 2,2,1,1,0 \rangle$, $\langle 2,1,1,2,0 \rangle$, $\langle 2,1,1,1,0 \rangle$,

$\langle 2,1,1,0,0\rangle$, $\langle 2,0,1,2,0\rangle$, $\langle 2,0,1,1,0\rangle$, and $\langle 2,0,1,0,0\rangle$.

In order to understand what *path type* each distinct hop sequence for a source-destination pair (n_s, n_d) in G represents, we classify the sources for a particular destination in G into distinct types. Let $S_x(n_d)$ denote the set of source nodes that have exactly $(d-x)$ digits resolved in their addresses w.r.t A_d and let $S_{x,k}(n_d)$ denote a subset of $S_x(n_d)$ where the value at the target index of every address can be resolved (w.r.t A_d) in exactly k routing steps. Let $\langle A_1, A_2, \ldots, A_{k_i}\rangle$ be the *sub-path*, that the message is relayed on, during the k_i routing steps (with $A_j \rightarrow A_{j+1}$ denoting the j^{th} step) carried out to resolve the i^{th} search level for pair (n_s, n_d). Now, note that, address A_j in the sub-path *must belong to* set $S_{(d-i+1),(k_i-j+1)}$ because each address on the sub-route has exactly $(i-1)$ digits resolved w.r.t A_d and it takes exactly (k_i-j+1) routing steps from A_j for the digit corresponding to the i^{th} level to be resolved. If among the x unresolved digits, there are x_p don't care digits and v_t distinct values, that require exactly (k_i-j+1) routing steps to be resolved, possible at the target index, then the size of set $S_{x,k}(n_d)$ would be $v_t b^{x_p}$ as while the values of the other digits are *deterministically bound* to their initial values, different values for the target and the don't care digits allow addresses to qualify as members of $S_{x,k}(n_d)$. A hop sequence $H : \langle 0,1,2,0,2,1,0\rangle$ for the pair (n_s, n_d), therefore, represents the path type - $r(H) : \langle S_{5,1}, S_{4,2}, S_{4,1}, S_{2,2}, S_{2,1}, S_{1,1}\rangle$ - which means that the j^{th} routing hop is an address that comes from the j^{th} member of sequence $r(H)$. Since, one or more sets from $r(H)$ may have multiple addresses, sequence H represents multiple paths (hence path type) rather than a singe path from n_s to n_d. Note that, by their definition, the sets $S_{x,k}(n_d)$ are all *non-overlapping* (have no common addresses) and together exhaust the $(N-1)$ sources possible for A_d in G. For Ψ with $d=4, z=2$, let index sequence for (n_s, n_d), $\langle 0,1,2,3\rangle$. Let $A_s \in S_{3,1}(n_d)$ (i.e, $k_0 = 0$ as the 0^{th} level is already resolved in A_s w.r.t A_d and $k_1 = 1$), let $k_2 = 1$ and $k_4 = 0$, while indices 1 and 3 correspond to don't care digits, then, following $(2+1)^1 = 3$ (as one of the two don't care digits is resolved at the beginning of the routing process) is the exhaustive list of route types from A_s to A_d.

1) $\langle 0,1,1,2,0\rangle : A_{3,1}, A_{2,1}, A_{1,2}, A_{1,1}, A_d$
2) $\langle 0,1,1,1,0\rangle : A_{3,1}, A_{2,1}, A_{1,1}, A_d$
3) $\langle 0,1,1,0,0\rangle : A_{3,1}, A_{2,1}, A_d$

For consecutive addresses A_{x_1,k_1} and A_{x_2,k_2} on any route ($A_d = A_{0,0}$), if $k_1 \in [2, z]$, then $x_2 = x_1$ and $k_2 = k_1 - 1$ (as the same digit (level) is targeted in consecutive routing steps (block shifts) until it is resolved and with every block shift at the same level, ϕ_t reduces the host to destination block distance by exactly 1 unit) whereas if $k_1 = 1$ (i.e., if the target digit is one routing step away from being resolved), then $x_2 < x_1$, as at least one more digit (the target digit) gets resolved in the routing step s: $A_{x_1,k_1} \rightarrow A_{x_2,k_2}$. In addition, $x_n = (x_1 - x_2 - 1)$ digits may get resolved, too, *by chance*, in the same routing step as a result of random selection of the don't care digit values for neighbour A_{x_2,k_2} by ϕ_t in A_{x_1,k_1}'s routing table. If ϕ_t installs $\hat{\mathfrak{b}}$ entries per routing cell in the row (for search level i) of A_{x_1,k_1}'s routing table that contains A_{x_2,k_2}, the probability that an additional x_n digits (besides the current target) get resolved by chance in at least one of the $\hat{\mathfrak{b}}$ entries for routing step s is $1-(1-1/b^{x_n})^{\hat{\mathfrak{b}}}$. In general, let $x_n = \max\{0, x_2 - x_1 - 1\}$, then $p(x_1 \rightarrow x_2)$, probability a routing link exists from an address with x_1 digits resolved (w.r.t A_d) to one with x_2 digits resolved, is given by,

$$p(x_1 \rightarrow x_2) = 1 - \left(1 - \frac{1}{b^{x_n}}\right)^{\hat{\mathfrak{b}}} \tag{1}$$

Further, depending upon whether the value to the next target digit (in A_{x_2,k_2}), say v_2, is allotted *deterministically* (in which case, $v_2 = f(v_0, \phi_p)$, for some function 'f' while v_0 is the value the target digit had in the source address) or *non-deterministically* (when the target digit is a don't care digit in A_{x_1,k_1}), k_2 takes a single value ($g(v_0, \phi_p)$, for some function 'g') or it takes any of the I_b values with equal likelihood, respectively. At any level in the search space, let $B_{v,k}$ denotes the set of block values from which block v is exactly k steps away. $B_{v,k}$ can be computed using the inverse relation for ϕ_t, which can be derived by inverting ϕ_t's component impulse ρ at all the levels at which ϕ_t operates. It follows from the fact that no matter how ϕ_t connects the blocks at the target levels, that, every distinct source-destination block pair lies on *exactly* 1 *shortest chain* at every target level, given any two blocks v_{d_i} and v_{s_i} at level i, and an offset k, function $\rho^{-1}(v_{d_i}, v_{s_i}, k)$ that returns the block v_{h_i} from which v_{d_i} is exactly k steps away on the shortest chain connecting v_{s_i} to v_{d_i} at level i, can be derived given the function ρ that forms connections amongst the blocks at level i. Applying ρ^{-1} for a specific k, $v_d = v$ and $\forall v_s \in I_b$, we can derive the set $B_{v,k}$. Now, let v_h and v_d denote the values of the target digit in A_{x_2,k_2} and A_d, respectively, then $v_h \in B_{v_d,k_2}$. Let b_t denote the *total* no. of values the target digit in A_{x_2,k_2} may take and let b_f denote the total no. of values it may take to be a member of B_{v_d,k_2}. Note, that when choosing the value of the target digit in A_{x_2,k_2} non-deterministically, $b_t = b$ and $b_f = |B_{v_d,k_2}|$, whereas in the deterministic case, $b_t = 1$ as well as $b_f = 1$. So, in general, we have, $p(v_2 \in B_{v_d,k_2})$, the probability that the *next* target digit value (v_2) in at least one of the $\hat{\mathfrak{b}}$ entries installed per cell in the row corresponding to level i in A_{x_1,k_1}'s routing table, belongs to B_{v_d,k_2}, given by,

$$p(v_2 \in B_{v_d,k_2}) = 1 - \left(1 - \frac{b_f}{b_t}\right)^{\hat{\mathfrak{b}}} \tag{2}$$

Combining eqns (1) and (2), we get the probability that A_{x_2,k_2} *exists as the next hop address towards destination* A_d in A_{x_1,k_1}'s routing table built by Ψ, $p(A_{x_1,k_1} \rightarrow A_{x_2,k_2})$,

$$= \left[1 - \left(1 - \frac{1}{b^{x_n}}\right)^{\mathfrak{b}}\right]\left[1 - \left(1 - \frac{b_f}{b_t}\right)^{\mathfrak{b}}\right] \tag{3}$$

Using this result, $p(A_s \dashrightarrow A_d, H)$, the probability that path $(A_s \dashrightarrow A_d)$ is of the type represented by hop sequence H: $\langle h_1, h_2, \ldots, h_{x-1}, h_x\rangle$ is given by,

$$p(A_s \dashrightarrow A_d, H) = \prod_{j=1}^{l(H)} p(A_{x_j,k_j} \rightarrow A_{x_{j+1},k_{j+1}}) \tag{4}$$

where '$l(H)$' is the path length given by $\sum_{j=1}^{x} |h_j|$ while A_{x_j,k_j} is the j^{th} address on the path type determined by hop sequence H. Now, we know that addresses in the set $S_{x,k}(n_d)$ have exactly $(d-x)$ digits resolved w.r.t A_d with have 1 target digit in the remaining x digits. Let x_d (in the remaining $(x-1)$ digits) denote the no. of digits the values of which after any routing step are *deterministically bound* to their initial values by Ψ. Then, x_p, the no. of don't care digits in every address in the set $S_{x,k}(n_d)$, equals $(x - x_d - 1)$. Since, x_d is determined by Ψ, x_p can be computed for the set $S_{x,d}(n_d)$. In every address $A \in S_{x,k}(n_d)$, $(d-x)$ resolved digits along with another x_d digits in the remaining x have their values fixed while the d_p don't care digits lead to a total of b^{d_p} distinct subsequences and for each such subsequence, the target digit leads to another $b_f = (1$ or $|B_{v_d,k_2}|)$ (as explained earlier) subsequences. The no. of distinct addresses in $S_{x,k}(n_d)$, therefore, comes out to be $|S_{x,k}(n_d)| = b_f b^{d_p}$. Let, $\#(A_s \dashrightarrow A_d, H)$ denote the total no. of distinct paths represented by hop sequence H for pair (n_s, n_d). Let $S_{(x,k)_i}(n_d)$ denote the i^{th} set in the path type $r(H)$ (length $l(H)$) corresponding to hop sequence H for (n_s, n_d), then we have,

$$\#(A_s \dashrightarrow A_d, H) = \prod_{j=1}^{l(H)} |S_{(x,k)_j}(n_d)|$$

while if $\#H(n_s, n_d) = (z+1)^{x'}$ (x' : don't care digits in *intermediate* hops) denotes the total no. of distinct hop sequences for the pair (n_s, n_d), $\#(A_s \dashrightarrow A_d)$, the total no. of distinct paths represented by all H_j's combined for the pair (n_s, n_d) (which is also the total no. of distinct paths possible from n_s to n_d in G), is given by

$$\#(A_s \dashrightarrow A_d) = \sum_{j=1}^{\#H(n_s,n_d)} \#(A_s \dashrightarrow A_d, H_j)$$

Now, note that, as can be seen in the previous example of exhausting path types for a source-destination pair, the $\#H(n_s, n_d)$ hop sequences for the pair (n_s, n_d) are all distinct in the sense that every sequence differs from every other sequence at at least one hop. And since, all sets $S_{x,k}$ are non-overlapping, it follows that all hop sequences for the same source-destination pair are mutually exclusive. So, if H_j denotes the j^{th} hop sequence for (n_s, n_d), while l_j denotes the length of the paths represented by H_j, we get the expected route length for the pair (n_s, n_d) in network G (built by Ψ), $\tilde{l}_G(A_s \dashrightarrow A_d)$

$$= l_j \sum_{j=1}^{\#H(n_s,n_d)} \left[\frac{\#(A_s \dashrightarrow A_d, H_j)}{\#(A_s \dashrightarrow A_d)}\right] p(A_s \dashrightarrow A_d, H_j) \tag{5}$$

where z represents the level diameter. Let $Pr[s = A_s, d = A_d]$ denote the probability that a randomly chosen source-destination pair has n_s as the source node and n_d as the destination node. Since all of the $N(N-1)$ pairs are equally likely to be chosen in a random draw, $Pr[s = A_s, d = A_d] = 1/N(N-1)$ for any pair in the network. That combined with using eqn. (5) for all sources and all destinations in the network, we get the expression for $\tilde{\mathbf{l}}_G$, the *mean chord length of the network*,

$$\tilde{\mathbf{l}}_G = \sum_{A_d, A_s \in \mathbb{A}_{b,d}} \frac{\tilde{l}_G(A_s \dashrightarrow A_d)}{N(N-1)} \tag{6}$$

Sets $S_{x,k}(n_d)$ for a given node are computed using ${\phi_t}^{-1}$ and ${\phi_p}^{-1}$, relations that can be derived given ϕ_t and ϕ_p, respectively, while the required probabilities depend on b and d, too, besides ϕ_p and ϕ_t. The mean chord length of the network is, therefore, captured by Ψ parameters - b, d, ϕ_p, ϕ_t, and J_s.

Application: As argued by *Kersch et. al* in [24], the average no. of look up hops may sometimes turn out to be a more informative performance metric than the worst case hops. This is because the existence of small paths in a network doesn't imply they can always be discovered [28]. We can always compare the routing schemes by the respective network chord lengths, they lead to, in such cases. Also, as the expected chord length would decrease with an increase in no. of don't care digits in the routing entries, allowing for more don't care digits improves the expected chord length while it makes the routing table less compact. That implies that there exists an inherent trade-off among the properties - *network diameter, expected chord length,* and the *node out-degree* - of an overlay network.

5. Conclusion and Future Work

We showed in this work how a formal definition of a generic routing scheme can be used to derive one of the most crucial network properties of a structured overlay network - its *mean chord length*. Besides being informative criterion on which various existing DHTs may be compared, the generic mean chord length expression can be used by the overlay designer to *customize* the routing scheme by tweaking its parameters and tune the network mean chord length according to his requirements. We are currently working on extending the definition by deploying a **class of 'ϕ_p's** instead of a single ϕ_p with each ϕ_p having a say in the structure of the network leading to a potentially useful hybrid structure - one which, we hope, could mould itself in response to network phenomena like - congestion, churn, node failures - etc., with minimum effect on the basic routing characteristics. The theory that we aim to establish is with the vision that it could of great help to systems designers in choosing the appropriate routing scheme parameters - b, d, ϕ_t, ϕ_p, J_s - to easily evaluate the trade-off between all (or at least most) of the network characteristics to choose these values in accordance with their system requirements.

References

[1] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. *In Proceedings of the ACM SIGCOMM 2001, San Diego, CA, USA, August 2001.*

[2] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content-Addressable Network. *In Proceedings of the ACM SIGCOMM 2001 Technical Conference, San Diego, CA, USA, August 2001.*

[3] A. Rowstron and P. Druschel. "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," *in Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001), November 2001.*

[4] B. Y. Zhao, J. D. Kubiatowicz and A. D. Joseph. "Tapestry: An infrastructure for fault-tolerant wide-area location and routing." *U. C. Berkeley Technical Report UCB/CSD-01-1141, April, 2001.*

[5] Petar Maymounkov and David Mazieres Kademlia: A Peer-to-peer Information Systems Based on the XOR Metric. *In Proceedings of the IPTPS 2002, Boston, March 2002.*

[6] F. Kaashoek and D. Karger. "Koorde: A simple degree-optimal distributed hash table," *in Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03), February 2003.*

[7] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: A Scalable Dynamic Emulation of the Butterfly. *In Proceedings of the PODC, 2002.*

[8] G.S. Manku, M. Bawa, and P. Raghavan. "Symphony: Distributed hashing in a small world." *in Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems, (Seattle, WA, USA), pp. 127-140, 2003.*

[9] S. Ratnasamy, S. Shenker, and I. Stoica, "Routing algorithms for dhts: Some open questions," *in Proc. of 1st Workshop on Peer-to-Peer Systems (IPTPS '01), Cambridge, MA, USA, 2001, pp. 45-52.*

[10] J. Xu, "On the fundamental tradeoffs between routing table size and network diameter in peer-to-peer networks," *in Proceedings of the 22nd Infocom, Mar. 2003.*

[11] Dmitri Loguinov, Anuj Kumar, Vivek Rai, and Sai Ganesh. "Graph-Theoretic Analysis of Structured Peer-to-Peer Systems: Routing Distances and Fault Resilience." *In Proceedings of the ACM SIGCOMM '03 Conference, Karlsruhe, Germany, August 2003.*

[12] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. "The impact of DHT routing geometry on resilience and proximity." *In Proc. ACM SIGCOMM, Aug. 2003.*

[13] Alima, L.O., Ghodsi, A., Haridi, S. "A Framework for Structured Peer-to-Peer Overlay Networks." *In: LNCS post-proceedings of Global Computing, Springer Verlag (2004) 223-250.*

[14] F. Klemm, S. Girdzijauskas, J.-Y. Le Boudec, and K. Aberer. "On Routing in Distributed Hash Tables." *In The Seventh IEEE International Conference on Peer-to-Peer Computing, 2007.*

[15] G.S. Manku "Routing networks for distributed hash tables." *In Proc. ACM PODC, Jul. 2003, pp. 133-142.*

[16] V. Darlagiannis "Overlay Network Mechanisms for Peer-to-Peer Systems." *Ph.D. dissertation, TU Darmstadt, 2005.*

[17] K. Aberer, L. O. Alima, A. Ghodsi, S. Girdzijauskas, S. Haridi, and M. Hauswirth. "The Essence of P2P: A Reference Architecture for Overlay Networks." *In Proceedings of the 5th International Conference on Peer-To-Peer Computing (P2P'05), pages 11-20. IEEE Computer Society, 2005.*

[18] B.G. Chun, B. Zhao, and J. Kubiatowicz "Impact of Neighbor Selection on Performance and Resilience of Structured P2P Networks." *in Proc. IPTPS, Feb. 2005, pp. 264-274.*

[19] J. Li, J. Stribling, F. Kaashoek, R. Morris, and T. Gil. "A Performance vs. Cost Framework for Evaluating DHT Design Tradeoffs under Churn." *In INFOCOM, 2005.*

[20] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. "A survey and comparison of peer-to-peer overlay network schemes." *IEEE Communications Surveys & Tutorials, vol. 7, no. 2, pp. 72-93, 2005.*

[21] J. Risson and T. Moors. "Survey of research towards robust peer-to-peer networks: Search methods." *Computer Networks, 2006.*

[22] JF Buford, H Yu. Peer-to-peer networking and applications: synopsis and research directions *in Handbook of Peer-to-Peer Networking, Springer, pp. 3-45 (2010)*

[23] Aleksandra Kovacevic, Kalman Graffi, Sebastian Kaune, ChristofLeng, and Ralf Steinmetz. "Towards Benchmarking of Structured Peer-to-Peer Overlays for Network Virtual Environments." *In 14th IEEE International Conference on Parallel and Distributed Systems, pages 799-804. IEEE, December 2008.*

[24] Kersch P, Szabo R. "Mathematical modeling of routing in DHTs[M]." *Heidelberg: Springer-Verlag, 2010,3: 367-401.*

[25] K. Junemann, P. Andelfinger, and H. Hartenstein. "Towards a Basic DHT Service: Analyzing Network Characteristics of a Widely Deployed DHT." *in Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on, 31 2011-aug. 4 2011, pp. 1-7.*

[26] O. Hanka, C. Spleib, G. Kunzmann, J. Eberspacher. "A DHT-Inspired Clean-Slate Approach for the Next Generation Internet" *Institute of Computer Networks, Technische Universitat Munchen (TUM), Germany, 2009.*

[27] C. Wang and K. Harfoush "On the stability-scalability tradeoff of DHT deployment" *INFOCOM 2007, pp. 2207-2215, 2007.*

[28] J. M. Kleinberg "The small-world phenomenon: an algorithmic perspective," in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, (Portland, OR, USA), pp. 163-170, 2000.

Appendix

Node: The term *'node'* refers to a *logical peer* in the virtual overlay network. If the logical peer is bounded to a physical machine from the underlay network, we say the node is *occupied*, otherwise we say the node is *empty*. The size of the overlay network, denoted by **'N'**, is the total number of nodes in the network, which is also the maximum number of physical machines that can be mapped onto the overlay network.

Address and **Digit**: The address of a node is a *label* associated with the node that *uniquely identifies* it in the overlay network. During any session (join to leave period) of its participation in the overlay network, a physical peer in the underlay is represented by *some* logical peer in the overlay via a binding between the former's *physical address* and the latter's logical address for the session. In the scope of this work, as shown in [0] that, without loss of generality, a node address for *any routing scheme* for SONs can always be thought of as a *sequence of symbols* where each symbol takes its value from some alphanumeric domain. The constituent symbols of an address are called its *digits*.

Message: By *message* we refer to the *data* that is routed across the network from the source node to the destination node. Besides the *payload information* that the source may want to communicate with the destination, the message contains *routing information* required by the routing scheme to guide the message to its destination.

Host Node: The node from which the message originates and the node to which it must be taken are referred to as the *source* and the *destination* nodes, respectively. Any other node that the message visits on its path from the source to the destination shall be referred to as an *intermediate* node. By the term *host node* we shall refer to a node that holds the *complete message* at any point of time during the routing process.

Search Space and **Address Format**: The address of a node encodes its unique location in an *absolute search space* defined w.r.t some base address. While the *indices* of the address digits correspond to the *search levels*, the value of the digit at a specific index maps to the unique *block* the node belongs to at the corresponding level. Since the search levels are labelled by their *depth* in the search hierarchy while *the order* in which the indices of an address may be *operated upon* (to find addresses with required destination values at these indices) by the routing scheme may vary for different source-destination pairs in the network, the index to level mapping may not always be *exact* meaning that the i^{th} index in an address may not necessary represent the i^{th} search level in a node's search view. So, though we often use the phrase '*the level* ***corresponding to*** *index* i *or vice-versa*' to refer to the index-level correspondence, even when we use the terms interchangeably, the meaning conveyed the phrase shall be implied. However, since, at any search level, a node belongs to exactly one block, the block's identifier within the level can be used as the digit value, too, at the *corresponding index* in the address, meaning that if there are y blocks at a level, we can label them $\{0, 1, ..., y-1\}$ and, for a node belonging to the i^{th} block at this level, the digit value corresponding to the level would be i. The two different notions - *levels-blocks* and *digit indices and values* - that essentially carry the same meaning, do not render each other redundant because while the former notion, that captures a d-dimensional hypercube representation of the nodes in the network, makes it convenient for us to establish results like *theorems* 1 *and* 2, it makes semantic sense and is much simpler and more conventional to use the latter - unique identifier - as a parameter for routing functions that build the connections amongst the overlay nodes and for forwarding functions that carry out the routing process.

Linux Software RAID Level 0 Technique for High Performance Computing by using PCI-Express based SSD

Jae Gi Son, Taegyeong Kim, Kuk Jin Jang, *Hyedong Jung
Department of Industrial Convergence, Korea Electronics Technology Institute, Korea
jgson@keti.re.kr, taegyeong@keti.re.kr, kjang@keti.re.kr, *hudson@keti.re.kr

Abstract—*The Linux-based legacy server systems are configured and used with software RAID to improve the performance of the disk I/O. Server systems requiring high performance prefer a special SSD that connects directly with the PCI-express bus to the SATA interface. However, the problem is that the current Linux kernel and software RAID are difficult to optimize the high-performance SSD based on PCI-Express because it is designed to be optimized for the conventional hard disk. Therefore, we propose the efficient method using re-combination and re-mapping techniques to improve the performance of software RAID level-0 provided on the Linux kernel level. This proposed method is designed to have more bandwidth at a time by reducing the number of system calls considering the block I/O characteristics of Linux kernel and RAID level 0. As a low-level I/O benchmarking tool, XDD is used to evaluate the performance of the proposed method. According to the experimental results, our performance gains are 28.4% on write bandwidth and 13.77% on read bandwidth compared with legacy software RAID. Moreover, CPU occupancy rates are decreased 81.2% and 77.8%, respectively.*

Keywords—*Software RAID, PCI-E Express SSD, High Performance Computing, Disk I/O, Memory Block Device*

1. Introduction

As big data, parallel and distributed processing become widespread, the demand for high-speed data storage technologies continues to rise. Moreover, more recent applications require fast response time with high throughput. However, as hard-disk input-output speed depends on mechanical movement, the speed lacks in comparison to processor and network transfer speed. Therefore, flash-memory based SSD have been increasingly used in personal computers and laptops and research is actively pursued related to non-volatile memory [1]. Currently companies such as FusionIO, Intel, etc. have developed and commercialized PCI-Express based high performance SSD products. Unlike SSD based on the common SATA interface, PCI-Express based SSD directly access the system bus architecture. Therefore, in this paper we designate all such devices as memory block devices. Despite the improved performance of PCI-Express based SSD when compared to hard disk drives of SATA SSD, system using such devices are configured using software RAID (Redundant Array of Inexpensive Disks). This is due to the non-existence of hardware-level RAID for SSD. Therefore in most Linux systems, software RAID (MD: Multiple Disk) is used to improve performance. However, the software RAID provided with standard Linux distributions was developed without consideration for high-speed block devices resulting in suboptimal performance when used with block devices. This is due to the fact that software RAID does simple mapping of input-output (IO) requests to the actual devices which are collected and optimized by the IO scheduler. In the case of high-speed block memory devices, as the devices are connected directly to the system bus in order to improve performance, the IO Scheduler does not exist. Therefore, a need exists to design the architecture for software RAID taking memory block devices into consideration. Therefore, in this paper we propose a method for improving the performance when applying software RAID Level 0 provided by standard Linux distributions. The proposed method has the following characteristics: First, the number of system calls within the Linux system is reduced to lower processor overhead. Second, the fast response times and high throughput of memory block devices are considered to improve the performance of software RAID.

The paper is organized as follows. In section 2, the technological background is described followed by an explanation of the proposed method for software RAID Level 0 for PCI-Express based high-performance SSD. Section 3 describes proposed method and error handling. In section 4, the performance is evaluated. Section 5 is the conclusion and direction for future research.

2. Background

In Linux, two general approaches for software RAID are provided. These are the multiple disk (MD) and device mapper (DM). Though these two methods are similar, MD was developed with RAID considerations. DM was developed for virtual block devices resulting in a framework which allows for direct access to real block devices. In this section we concentrate on MD which has shown better performance comparatively.

2.1 RAID Level 0 in MD

An MD device is a Linux virtual block device consisting of several disk drives combined together to provide larger capacity and improved reliability [2]. Figure 1 depicts the general hierarchical structure of an MD device and general disk hierarchy. From Figure 1, MD functions as a virtual device between the file system layer and the block device layer. The virtual MD layer receives IO requests and distributes them accordingly to the underlying disk layer.

RAID Level 0 comprised of data striping to provide improved performance. In general, RAID systems interleave sections of consecutive data on different devices in order to improve parallelism. These sections of data which are interleaving and allow parallel access are called stripes [3]. For example, for writing 1MB file, the files are divided into sections of 64KB and recorded by interleaving the section across each of the disks. Even though the data itself is not accessed in parallel, performance improves when compared to using a single device.

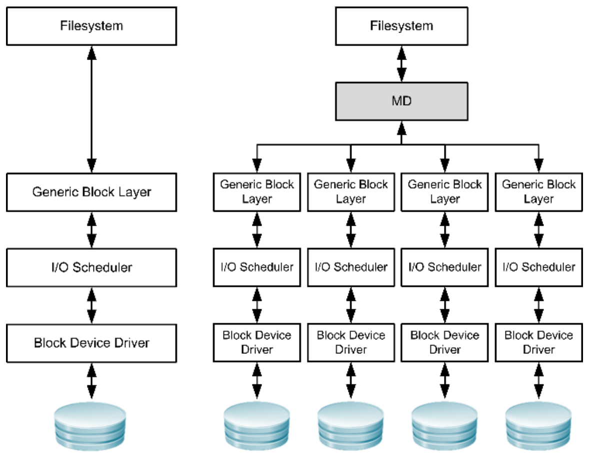


Figure 1. General disk hierarchy (left) and general hierarchical structure of an MD device. (right)

The MD provided by the Linux kernel handles 'bio request' in the sequence depicted in Figure 2 in order to execute IO on the actual device.

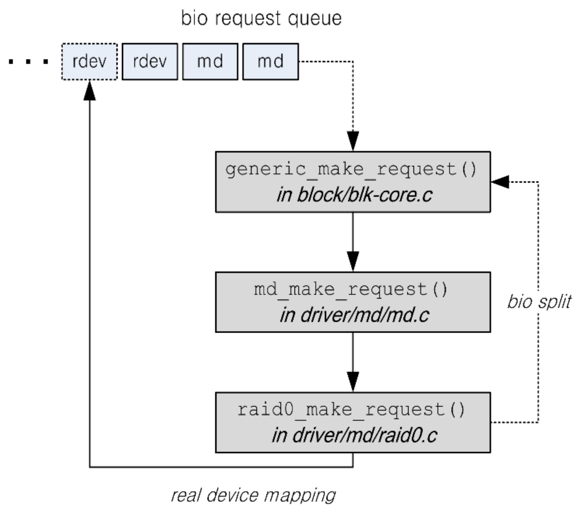


Figure 2. RAID level 0 processing of an MD

Additionally, the procedure for handling of 'bio request' is shown in Code 1. As shown in Code 1, the key procedure of software RAID can be analyzed as follows:

1. From the software RAID device of bio (bio bi_bdev) passed to the 'generic make request()' function, a function call is made to the callback 'md make request'

2. The 'md make request ()' function makes a subsequent call to the proper 'make request' function depending on the RAID Level. For RAID Level 0, the 'raid0 make request()' function is called
3. From the RAID Level 0 handler 'raid0 make request ()' analyzes the sector and size of the bio passed to the function and maps it to the actual device such as a hard disk drive. For bio which exist across two block devices are divided and mapped accordingly.
4. The mapped block device make a call once again to the 'generic make request' and passes the 'bio request' to the actual device

```
1.  __generic_make_request()
 Call make_request followed by bio->bi_bdev.
2. md_make_request()
 Call raid0_make_request for raid 0 module.
3. raid0_make_request()
 Check real block device to process the request, map
bio->bi_bdev to the block device
4. __generic_make_request()
 Send switched bio->bi_bdev.
5. __make_request()
 Insert request to request queue of real block
device.
6. Process request in real block device.
```

Code 1. Procedure of request in MD

As can be seen in the key highlight of the MD handling procedure, sequential IO events occur and in the case of large data recursive calls can result in drawbacks of high system overhead. In particular, performance improvement is limited due to the IO characteristics of disk devices and the stripe-level transfer of data for conciseness of software RAID. In the case of memory block devices with fast response times and high throughput, random memory access is possible therefore it is advantageous to send as much data as possible. However, the current software RAID is unable to utilize such characteristics.

2.2 Device Mapper

The device mapper was not originally developed for RAID but as a framework for storage devices. Thus the architecture allows for direct mapping of a virtual block device to an arbitrary physical device. DMRAID was developed to add support for RAID in DM [4].

3. Linux Software RAID Level 0 Technique for PCI-Express based High-Performance SSD

3.1 Proposed RAID Level 0 Technique

PCI-Express based high performance SSD have greatly improved IO performance when compared to general disk drives. However, as the current Linux IO layer is optimized for low-speed devices, it is unable to utilize the full performance potential of high speed block devices. Unlike slower hard disk drives, high-speed memory block devices are directly connected to the PCI-Express bus and therefore is unaffected by the Linux IO scheduler. Moreover, as the architecture of direct memory access (DMA) is organized in Scatter-Gather structure, which allows for transfer of non-consecutive segments of memory, large data transfers are possible. The proposed methods was designed to operate under the following assumptions. The memory block device is a device with fast response time and high throughput supports random access in a Scatter-Gather DMA structure. Analyzing the MD procedure from the Linux kernel under this assumption, an increase in performance can be expected from sending the first bio (block IO) and the (number of devices)*n+1 bio. Figure 3 depicts how the Linux kernel IO process when the strip size is 4 KB. Each 4KB bio is added on the underlying physical device in sequence.

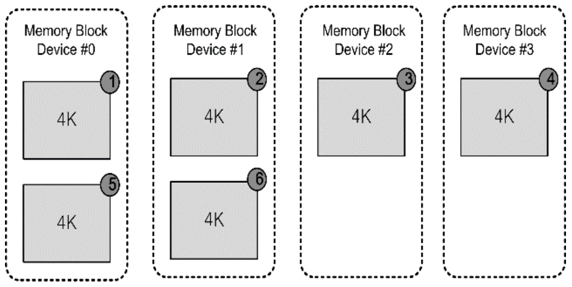


Figure 3. Block IO process in MD of Linux kernel

As can be seen in Figure 3, bio1 and bio5 occur on the same physical device. A significant gain in performance can occur if bio1 and bio5 can be processed in one IO event. In Figure 4, a total of 6 system calls occurs to transfer 24KB. In other words (Number of system calls) = (IO Data size) / (stripe size).

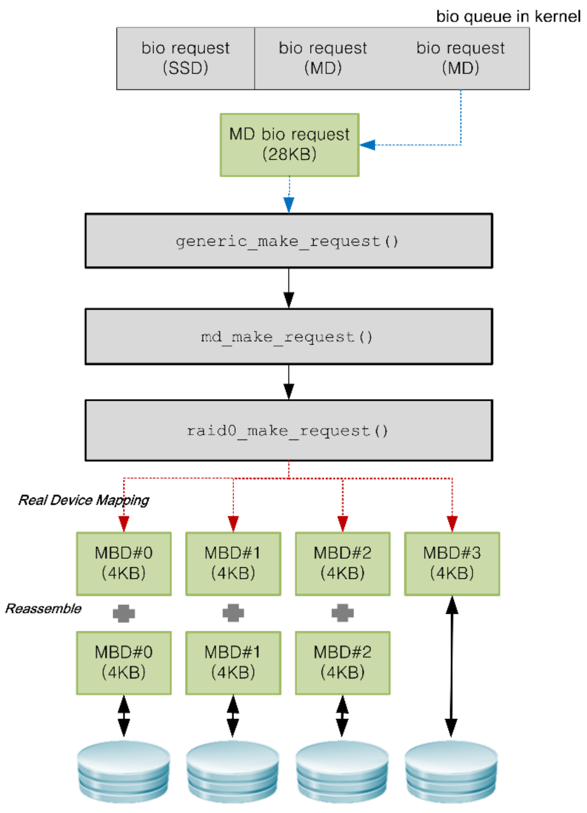


Figure 4. Proposed RAID Level 0 method

If we can combine IO data then, (Number of system calls) = (IO Data Size/Max Page Cache size +1) * Stripe Size, and in the case of Figure 4, the number of system calls is reduced to 4. The maximum allocation size for page cache is 1MB for the Linux kernel. However, allocating a large cache size results in a significant overhead cost.

Though the stripe size of software RAID can be increased to improve performance, this results in a tradeoff of IO occurring on only a subset of the devices. For example, with 4 devices with a stripe size of 512KB and an IO of size 1024KB results in IO on only 2 device. However, the proposed method allows for IO to occur distributed evenly across all devices.

Taking into consideration the characteristics of the 'bio request' and the MD process of the Linux kernel, and the assumption of low delay and high throughput of the high-speed memory block device the following methods are proposed:

1. Variable-length IO: The fixed stripe size of the bio request of MD is changed to allow variable length
2. Remapping & Reassemble: Remapping and reassemble occurs taking into consideration the characteristics of RAID Level 0 of MD

To support high-speed memory block devices such as PCI-Express based SSD according to the proposed RAID Level 0 method, the included software RAID Level 0 of the standard Linux kernel was modified and improved according to the proposed method. The improved software RAID in this paper is designated as Enhanced MD.

3.2 Error Handling

In IO systems, error handling is an important factor that has a significant impact on performance. bio of the Enhanced MD applied with the proposed method generates a new 'bio request' through remapping & reassembly. The error handling of the newly generated 'bio request' is forwarded to the Linux 'bio error handler'.

4. Performance Evaluation

4.1 Evaluation Environment

The performance of the proposed method for software RAID Level 0 was evaluated using the PCI-Express based 910 SSD from Intel. The Intel 910 SSD is configured with 2 physical SSD on a single interface card. For the performance evaluation, 2 interface cards were connected to PCI-Express 4x for a total of 4 physical SSD. The specifications of the evaluation environment are shown in detail in Table 1. We used CentOS with 2.6.32_x86_64 kernel and set the stripe size to 4096 bytes. Intel Xeon X5550 and 8GB memory were used for the system.

Table 1. Evaluation environment

	Specification
CPU	Xeon® X5550 * 2EA
RAM	8GB
OS	CentOS Linux 6.0 (final)
Kernel	2.6.32_x86_64
SSD Device	Intel 910 SSD (200Gx4ea): PCI-E
Stripe Size	4096 bytes

For evaluation, the stripe size was set to 4KB using the mdadm utility of MD. For comparison, an unmodified MD installed by default on CentOS 6.0 was used with the same environment to compare measurements of CPU usage and IO bandwidth. The performance evaluation tool XDD was used [5]. The XDD benchmark is a tool to measure disk performance and can be used to analyze low-speed IO in software RAID. The tool is commonly used to effectively evaluate block device performance. The XDD performance test was set to run with IO on 16 threads to allow operation under a regular system load.

4.2 EXPERIMENTAL RESULTS

The developed software RAID Level 0 showed optimal performance with IO of size greater than (number of disk times stripe size) due to the characteristics of the key operating algorithm. From Figure 5 comparing the write bandwidth, it can be seen that the bandwidth greatly improves with Enhanced MD. When the IO size is 4-16 KB, the results are similar to the characteristics of RAID Level 0. In this case, the request are handled as sequential writes to a single memory block device. There is a slight performance degradation when the IO size in 8KB. When the IO size is 32 ~ 1024 KB, significant gains in performance is achieved by the proposed method. When the size is over 2048 KB, it was analyzed that performance improvements are no longer achieved due to the fact that the size is over the maximum page cache allocation of Linux. Overall, the write bandwidth of Enhanced MD achieved an improvement in performance on average of 28.24%.

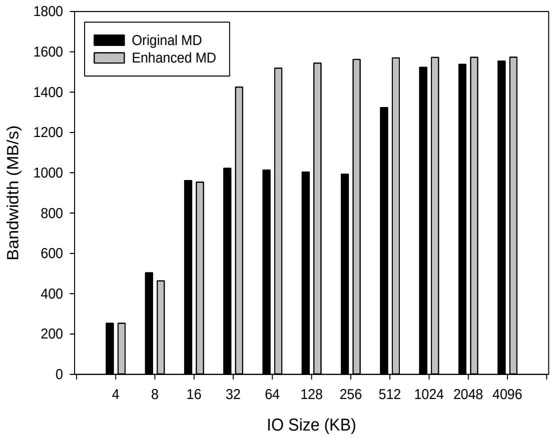


Figure 5. Comparison of writing throughput

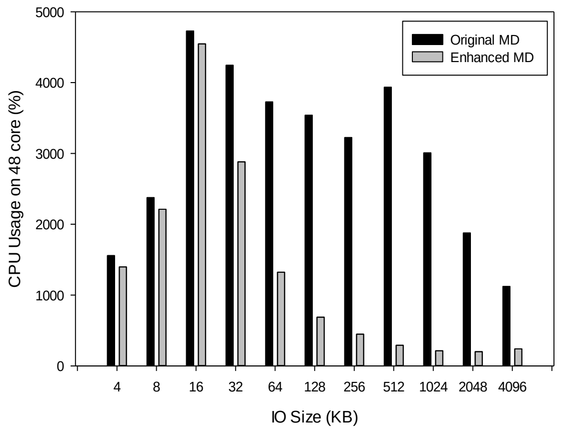


Figure 6. Comparison of CPU usage in writing

Figure 6 compares the CPU usage from the evaluation. From the results, the overall CPU usage across all sizes is less with Enhanced MD. This was analyzed to be due to the proposed methods remapping and reassembly, which results in a significant reduction in the number of system calls. As shown in Figure, the CPU usage is reduced by an average of 77.8%. The results for read performance were also improved in general as shown in Figure 7. As before with the result of write bandwidth, similar performance when compared with MD is shown for sizes of 4 - 16KB due to the characteristics of RAID

Level 0. However, for IO sizes of 32 - 128 KB, the memory allocation of the proposed method was analyzed to be the cause of the performance degradation in page caching. In general, the read bandwidth showed an improvement of about 13.77% on average.

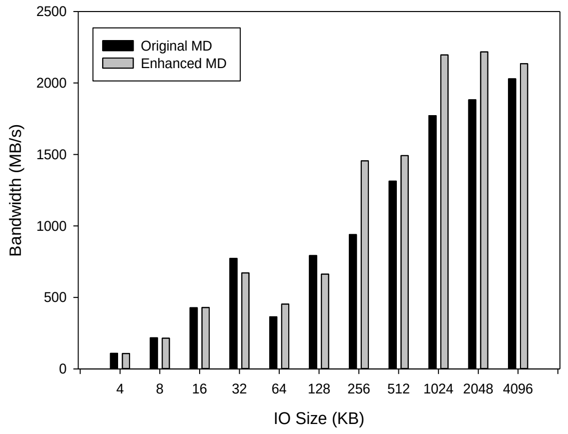


Figure 7. Comparison of reading throughput

The CPU usage for reads is similar to that of write operations as shown in Figure 8. From the figure, the overall CPU usage has been reduced significantly. These results confirm the positive improvements of the proposed system. For read operations there was an average of 81.2% reduction in CPU usage.

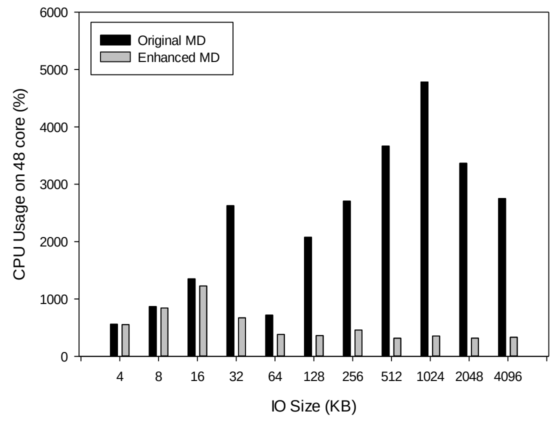


Figure 8. Comparison of CPU usage in reading

From the overall analysis of the entire experimental results, it can be seen that the performance of the proposed method shows significant improvement compared to the software RAID of standard Linux. Moreover, reduction in CPU overhead and improvements in IO performance can lead to improvements in overall system performance. In particular, as shown from the results from XDD experiments, by improving low-level IO performance, the proposed method will lead to significant improvements to systems which do not use page caching such as database systems.

5. CONCLUSION

The standard Linux kernel and software RAID is optimized for traditional block devices and is not suitable for current high-speed memory block devices. In the case of server systems which require high performance, limitations are shown in performance despite the use of high-speed memory block devices configure with SW RAID. Therefore, this paper proposed a method which provides a solution to the performance degradation when using high-speed memory block devices with the standard Linux kernel and software RAID Level 0. The proposed method has the following characteristics. First, the stripe unit of data used in RAID block IO was improved for variable length resulting in a reduction of system call within the Linux kernel. This reduces the overhead of the processor. Second, taking into consideration the fast response time and large bandwidth of the memory block device, the performance of software RAID is greatly improved. In addition, the performance was confirmed through evaluation using the low-level IO performance evaluation tool XDD. Based on the results in this paper, further research is needed to apply the method to RAID Level 4 and Level 5 while improving overall system reliability and safety. Furthermore, research regarding block migration on write operations which occurs due to the characteristics of SSD is needed

ACKNOWLEDGEMENT

This work was partly supported by the R&D program of MSIP/LEIT [10044910, Development of Multi-modality Imaging and 3D Simulation-Based Integrative Diagnosis-Treatment Support Software System for Cardiovascular Diseases] and [10045299, Development of Integrated Resource Management Solution for VDI Service System].

REFERENCES

[1] Alexandre P. Ferreira, Miao Zhou, Santiago Bock, Bruce Childers, Rami Melhem and Daniel Mosse, “Increasing PCM Main Memory Lifetime”, Proceedings of the Conference on Design, Automation and Test in Europe. European Design and Automation Association, pp.914-919, March 2010.

[2] Neil Brown, “Software RAID in 2.4”, Proceedings of linux.conf.au, Sydney, Australia, January 2001.

[3] Chen, P. M., Lee, E. K., Gibson, G. A., Katz, R. H., Patterson, D. A., “RAID: High-Performance, Reliable Secondary Storage”, ACM Computing Surveys, Vol.26, No.2, pp.145-185, June 1994.

[4] Heinz Mauelshagen, “dmraid - device-mapper RAID tool”, Proceedings of the Linux Symposium, Vol.1, pp.289-295, Ottawa, Ontario, Canada, July 2005.

[5] XDD, http://www.ioperformance.com/

[6] A. Brown and D. A. Patterson., “Towards Maint-ainablity, Availabilitsty, and Growth Benchmarks: A Case Study of Software RAID Systems”, In Proceedings of the USENIX annual Technical Conference (USENIX ‘00), pp.263-276, San Diego, Carlifornia, June 2000.

[7] D. Patterson, G. Gibson, and R. Katz., “A Case for Redundant Arrarys of Inexpensive Disks(RAID)”, In Proceedings of the 1988 ACM SIGMOD Conference on the Management of Data (SIGMOD ‘88), pp.109-116, Chicago, IL, June 1988.

The Static and Dynamic Performance of an Adaptive Routing Algorithm of 2-D Torus Network Based on Turn Model

Yasuyuki Miura [1], **Kentaro Shimozono** [2], **Kazuya Matoyama** [3], **and Shigeyoshi Watanabe** [1]
[1] Department of Information Science, Shonan Institute of Technology (SIT), Fujisawa, Kanagawa, Japan
[2]TOP Engineering Inc., Minato-Ku, Tokyo, Japan
[3] Net-Research Co.Ltd., Kawasaki, Kanagawa, Japan

Abstract - *A 2-D torus network is one of the most popular networks for parallel processing. Many algorithms have been proposed based on the turn model, but most of them cannot be applied to a torus network without modification. In this paper, we propose the North-South First Routing (NSF Routing) which combined the North First method (NF) and the South First method (SF). NF and SF are algorithms yielded by the turn model. NSF Routing is applicable to 2-D Torus. The dynamic performance was evaluated by a software simulation, and the static performance was also evaluated. As a result, it was shown that a throughput improves in some communication patterns, and about the same performance of a "The Average Number of Shortest Paths (ANSP)" static property.*

Keywords: Network on Chip, Interconnection Network, Adaptive Routing, Turn model

1 Introduction

The interconnection network is an important topic in the field of parallel processing. Parallel computers have processing elements (PEs) that are directly connected to a network such as a *k*-ary *n*-cube. Parallel processing is also performed in a *Network on Chip (NoC)* between PEs located on one chip. Many different interconnection networks for parallel processing have been proposed, and the 2-D torus network is one of the most popular.

The routing algorithms of interconnection networks are classified into deterministic routing, in which paths are fixed, and adaptive routing [1]-[7], in which paths are changed to avoid failures or congestion. Because of its tolerance to failures and congestion, adaptive routing has been the topic of a lot of research. Various adaptive routing algorithms have been proposed for *k*-ary *n*-cubes [3]-[7]. However, these methods require additional hardware for virtual channels comparison with deterministic routing (its name is *Dimension Order Routing,* DOR) on a 2-D torus.

A number of adaptive routing algorithms based on the turn model [8]-[10] do not need additional virtual channels. However, most of these algorithms cannot be applied to torus networks without change because most of those methods are algorithms for mesh or hyper-cube network. If an adaptive routing algorithm for a torus network could be realized by modifying the turn model, it would be possible to realize adaptive routing without having to install additional virtual channels.

In this paper, we propose the North-South First Routing (NSF Routing) which combined the North First method (NF) and the South First method (SF) and evaluate the performance by software simulation and a static property. NF and SF are the part of the algorithms by a Turn model. NSF Routing is applicable to 2-D Torus. Moreover, performance is evaluated by a software simulation.

2 2-D Torus Network

The 2-D torus network has an $N \times N$ 2-dimensional structure, and its four edges are connected by wraparound links. It is used in many parallel computers and some interconnection networks include this.

Dimension order routing (DOR) is generally used for deterministic routing on a 2-D torus. In DOR, the packet moves on channels in the y-direction before moving to the x-direction. To avoid deadlocks on a 2-D torus, DOR needs two virtual channels (channel-L and channel-H).

The method of selecting a virtual channel in the case of DOR on a 2-D torus network is as follows:

- Choose channel-L when starting routing in the y-direction.
- When the head of the packet passes through a wraparound link, move the packet to channel-H.
- When the routing in the y-direction is completed, move the packet in the x-direction; use channel-L regardless of the current channel.
- When the head of a packet passes through a wraparound link, moves the packet to channel-H. Use channel-H until the routing finishes.

Figures 1 and 2 show the link selection function and channel selection function of DOR on an $N \times N$ torus. Here, the address of each PE of the torus is shown in terms of their coordinates (x, y). Moreover, the y-direction channels are written as Y+ and Y−, and the x-direction channels are written as X+ and X−. The four inputs of the link selection function indicate the *x* and *y* coordinates of the present PE, and the *x* and *y* coordinates of the destination PE. The function outputs the link of either X+, X−, Y+, Y− or "OUT", which is an output link to a node.

The three inputs of the channel selection function correspond to the current direction, current channel, and direction of the next hop. The current direction and the direction of the next hop have four states, i.e., X+, X−, Y+, and Y−. The current channel has three states, i.e., channel-L (L), channel-H (H), and wraparound channel (W). Although the output has two states (L and H), it unconditionally serves as W when the selected link is a wraparound link.

```
// Link Selection Function for Dimension-Order Routing
Link_Select_DOR (cx, cy, dx, dy)
    cx, cy;          // current node        0 ≦ cx, cy ≦ N−1
    dx, dy;          // destination         0 ≦ dx, dy ≦ N−1
{
    if(cy≠dy){                              // dimension Y
      dist_y = (N+dy-cy)%N;
      if(1≦dist_y ≦N/2)                  return Y+;
      else                               return Y-;
    }
    else if(cx≠dx){                         // dimension X
      dist_x = (N+dx-cx)%N;
      if(1≦dist_x ≦N/2)                  return X+;
      else                               return X-;
    }
    else                                 return OUT;
}
```

Fig.1 The Link Selection Function of the Dimension-Order Routing

```
// Channel Selection Function for DOR
Channel_Select_DOR (cd, cc, nd)
    cd;         // current direction       ∈{Y+, Y-, X+, X-}
    cc;         // current channel         ∈{L, H, W}
    nd;         // next direction          ∈{Y+, Y-, X+, X-}
{
    if(cc∈L)                    return L;  // before wrap around
    else                                   // after wrap around
       if(cd∈{X+,X-} & nd ∈{Y+,Y-})
                                return L;  // Y→X
       else                     return H;
}
```

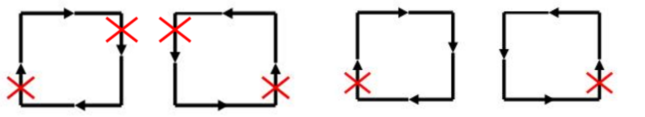
Fig.2 The Channel Selection Function of the Dimension-

3 Adaptive Routing of *k*-ary *n*-cube

3.1 Turn Model

The turn model [8] is used by some adaptive routing algorithms [9][10]. Packet cycles can be prevented by adding a restriction to a path change (turn) of a packet. In the case of a 2-D mesh, there are eight kinds of turn, and the various turn model methods put restrictions on two of the eight turns. There is essentially no difference between these methods other than the choice of turn to be restricted. In this paper, we shall incorporate the North First (NF) algorithm and South First (SF) algorithm into one (NSF) and apply it to a 2-D torus.

The turn model of DOR for a 2-D mesh is shown in Fig.3 a), and the turn model of the NF algorithm is shown in Fig.3 b). DOR restricts four out of eight turns, whereas the NF algorithm restricts only two, i.e., X− (left, west)→ Y+ (upper, north) and the X+ (right, east)→ Y+ (upper, north). The South First algorithm, by which the Y− (South) direction is chosen at the beginning of a routing path, is similar.



a) Dimension -Order Routing   b) North First (NF) Algorithm

Fig.3 The Turn Model for 2D-Mesh Network

3.2 Application of the Turn Model to a Torus Network

When applying a turn model such as the NF algorithm to a torus network without change, the following differences from the case of a mesh network have to be considered.

1) In a torus network, when the packet passes through a wraparound channel, a deadlock by cyclic dependency can occur. Therefore, it is necessary to impose an additional restriction.
2) At least two virtual channels are needed for routing in a torus network. As a result, adaptive routing with higher pliability is attained by applying different turn models to each channel.

An example of a cyclic dependency that occurs in the NF algorithm is shown in Fig.4. Here, packets A-D mutually block a path, causing a deadlock. By contrast, the deadlock does not happen in DOR because packets A and C do not turn in Fig.4. This problem illustrates that it is necessary to take into consideration complicated turn restrictions in adaptive routing on a torus network. Our method deals with this issue by applying the NF and SF algorithms to channel-H and channel-L.

4 North-South First Routing

If the turn model such as NF or SF algorithms is used for 2-D torus, the circulation as shown in Fig. 4 occurs by packets of wraparound channels. To avoid the sort of deadlock described above, additional restrictions have to be put on the NF and SF algorithms:

1) The SF algorithm does its routing on channel-H. However, a cycle may occur when a path is chosen in which a packet returns to channel-L through channel-H, and for this reason,

DOR is carried out instead of the adaptive routing. In DOR, the x-direction channel chosen after a vertical (y-direction) wraparound channel has to be channel-L.

2) The NF algorithm does its routing on channel-L. Because the path of channel-H→channel-L exists after a wraparound channel, the cycle shown in Fig.4 occurs. As shown in Fig.5, though, the cycle can be avoided by adding one more restriction to the other two. Here, three restrictions are put on eight turns, specifically, right→upper, left→upper, and right→lower. This algorithm was named *restricted North First (rNF)*.

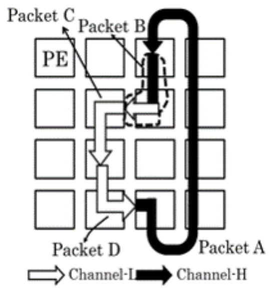


Fig.4 The Cyclic Dependency by the Application of the NF Algorithm in Torus Network

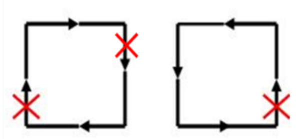


Fig.5 The Restricted North First Routing

4.1 Definitions

From here on, all channels will be described in terms of their dimension $d \in \{X, Y\}$, direction $\delta \in \{+, -\}$, channel type $c \in \{L, W, H\}$, i.e., $(d\delta, c)$. X means X dimension, Y means Y dimension, and L, W, and H means channel-L, wraparound channel, and channel-H. $(d+, c)$ and $(d-, c)$ will be shown as a set, written as $(d\pm, c)$.

4.2 Routing Algorithm

In our method, the restricted NF algorithm is carried out in channel-L and the SF algorithm is carried out in channel-H. Since $(Y-, L)$ and $(Y+, H)$ are respectively used in the restricted NF algorithm and SF algorithm, we will study cases in which $(Y+, c)$ is used and not used, and cases in which the horizontal and vertical wraparound channels are used and not used.

Figures 6 and 7 show the link selection function and channel selection function of the proposed method on a $N \times N$ torus. As in the case of DOR in Fig.1, the link selection function outputs X+, X−, Y+, Y−, or "OUT" (an output link to a node). The proposed method needs the "current channel" as an input in addition to the inputs of DOR.

The channel selection policy varies depending on whether $(Y+, c)$ is used or not. If it is used, adaptive routing is carried out only when the wraparound channels is not to be used from that point on. If $(Y+, c)$ is not used, the restricted NF algorithm is carried out from the source PE until the first wraparound channel (or destination PE) is reached.

The algorithm of Fig.6, in ①, first determines whether the wraparound links of X and Y are used. In this case, the determination is based on the X and Y coordinates of the source and destination PEs as follows:

- When the difference between the X coordinates of the current PE and destination PE is less than *N*/2, h_wrap is set to 0 because the wraparound channel of the x-direction is not straddled. If not, h_wrap is set to 1.
- When the difference between the Y coordinates of the current PE and destination PE is less than *N*/2, v_wrap is set to 0. If not, v_wrap is set as 1.

Next, the link is chosen on the basis of whether the Y+ channel (channel $(Y+, c)$) is used or not, as follows:

- When $(Y+, c)$ is used, the procedure ② is carried out. In this case, since the restricted NF in channel-L is equivalent to DOR, only the adaptive routing of the SF method in channel-H is carried out. If neither wraparound channel is used in going from the current PE to the destination, the packet can be sent over channel-H and routing can be continued. Thus, adaptive routing can be carried out with the SF method. The only other case in which channel-H may be used is after the packet has passed through a vertical wraparound channel (Y+, W) and is due to pass through a horizontal wraparound channel (X±, W). Even in this case, it is thought that adaptation routing using the SF method is possible. However, since it is difficult to prove that is deadlock-free, only the X-directional routing is carried out at first and SF is applied after the packet has passed through channel (X±, W). In the other case, DOR is carried out because only the channel-L is used.
- When $(Y+, c)$ is not used, the procedure ③ is carried out. Since the SF method in channel-H is equivalent to DOR, only the adaptive routing of the restricted NF method in channel-L is carried out. In this case, the following restriction is added in order to make the order of passage in a wraparound channel into (Y−, W) → (X±, W).
  - ➢ Restricted NF is carried out only when (Y−, W) is not be passed from the current PE to the destination or the next channel is not (X±, W). DOR is carried out otherwise.

Besides the three inputs of the channel selection function of DOR in Fig.2, the channel selection function needs four inputs that indicate the x and y coordinates of the source and destination PEs. These new inputs can be used to judge the possibility of the packet passing through a wraparound channel. Based on the judgment, channel-H is chosen only when the wraparound channel is not to be used and $(Y+, c)$ is to be used. DOR is carried out otherwise. As in the case of DOR, the output has two states, L and H. However, an output unconditionally serves as W when the selected link is a wraparound link.

```
// Link Selection Function for Proposed Algorithm
Link_Select_Prop (cx, cy, cc, dx, dy)
    cx, cy;          // current node       0 ≦ cx, cy ≦ N－1
    cc;              // current channel    ∈{L, H, W}
    dx, dy;          // destination        0 ≦ dx, dy ≦ N－1
{
    if(|dx-cx|≧N/2)            h_wrap = 1;
    else                       h_wrap = 0;          ①
    if(|dy-cy|≧N/2)            v_wrap = 1;
    else                       v_wrap = 0;

    dist_y = (N+dy-cy)%N;
    if(1≦dist_y ≦N/2)                  // Y+ direction
        if(h_wrap=0 & v_wrap=0)
                               return adaptive_SF(cx, dx);
        else if(h_wrap=1 & v_wrap=0)                    ②
                               return DOR(cx, cy, dx, cy);
        else                   return DOR(cx, cy, dx, dy);
    else if(cy≠dy)      // Y- direction
        if((cc∈L) and ((cx≠0) or (v_wrap=0))
                               return adaptive_NF(cx, dx);  ③
        else                   return DOR(cx, cy, dx, dy);
        else if(cx≠dx)         return x_route(cx, dx);
    else return OUT;
}

adaptive_SF(cx, dx){       //adaptive routing of SF algorithm
    if(cx=dx)                  return Y+;
    else if(buffer_is_full(Y+, H)=TRUE)
                               return x_route(cx, dx);
    else                       return Y+;
}

adaptive_NF(cx, dx){          //adaptive routing of NF algorithm
    dist_x = (N+dx-cx)%N;
    if(cx=dx)                  return Y-;
    else if(N/2＜dist_x)                   // X- direction
                               return X-;
    else if(buffer_is_full(Y-, L)=TRUE)   // X+ direction
                               return X+;
    else                       return Y-;
}

x_route(cx, dx){
    dist_x = (N+dx-cx)%N;
    if(1≦dist_x ≦N/2)          return X+;
    else                       return X-;
}

DOR (cx, cy, dx, dy){
    return Link_Select_DOR (cx, cy, dx, dy);
}
```

Fig.6 The Link Selection Function of the Proposed Algorithm

4.3 Deadlock Avoidance

A channel dependency graph is drawn in order to prove that the routing algorithm described in the previous section does not cause a deadlock[11][12]. The channel dependency graph is a directed graph in which nodes (channels) with dependencies are connected by an arrow. Specifically, nodes (channels) with dependencies are pairs of nodes (channels) in which a packet may be directly transmitted and received while routing.

```
// Channel Selection Function for Proposed Algorithm
Channel_Select (cx, cy, dx, dy , cd, cc, nd)
    cx, cy;          // current node         0 ≦ cx, cy ≦ N－1
    dx, dy;          // destination          0 ≦ dx, dy ≦ N－1
    cd;              // current direction    ∈{Y+, Y-, X+, X-}
    cc;              // current channel      ∈{L, H, W}
    nd;              // next direction       ∈{Y+, Y-, X+, X-}
{
    if(dx-cx≧N/2)                  h_wrap = 1;
    else                           h_wrap = 0;
    if(dy-cy≧N/2)                  v_wrap = 1;
    else                           v_wrap = 0;

    dist_y = (N+dy-cy)%N;
    if((1≦dist_y ≦N/2)                        // Y+ direction
   and (h_wrap=0 & v_wrap=0))      return H;
    else                                        // Others
                              return DOR_Channel (cd, cc, nd);
}

DOR_Channel (cd, cc, nd){
                         return Channel_Select_DOR (cd, cc, nd);
}
```

Fig.7 The Channel Selection Function of the Proposed Algorithm

First, the channel dependency graph is drawn. Then, each channel is numbered. If it is proved that the channel numbers are in ascending order (or descending order) in the direction of the arrows of the channel dependency graph, deadlock does not happen. In such case, the channels are said to have an ordered relation and the corresponding channel will not cause a cyclic dependency.

A routing algorithm based on the turn model generally assigns numbers to the output channels from the PE on the basis of the PE address. As mentioned above, a 2-D torus network has two virtual channels. Accordingly, the following 4-dimensional channel numbers CN are given to the 4 links $\times 2$ channels (=8 channels) in each PE of an $N \times N$ torus.

$$CN(x, y, d, ch) = (g_m, c_1, g_s, c_2) \qquad (1)$$

Here, x $(0 \leq x \leq N-1)$ and y $(0 \leq y \leq N-1)$ is x and y coordinates of PE address, $d \in \{Y+, Y-, X+, X-\}$ is the direction of the channel, and $ch \in$ {L,H,W} is the type of channel. Also, g_m, c_1, g_s, and c_2 are named as *Main Group*, *First Coordinate*, *Sub Group*, and *Second Coordinate*, respectively.
The channel number in each channel is as Fig.8. When a channel number is set as Fig.8, deadlock occurrence can be avoided because channel numbers will become an ascending order through a routing path[13].

5 Dynamic Performance Evaluation

5.1 Environment

We used a wormhole routing simulator to evaluate the dynamic communication performance of our algorithm on a

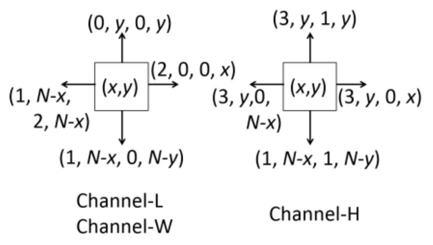


Fig.8 The Channel Number

16 × 16 2-D torus/mesh network with 256 PEs. Dynamic communication performances are simulated for dimension-order routing algorithm and proposed algorithm (North South First Routing). Extensive simulations have been carried out for uniform, matrix-transpose, and bit-reversal.

The dynamic communication performance of an interconnection network was characterized by the average transfer time and throughput. The average transfer time was the average value of the latency for all packets. Latency was the time between the injection time of the first flit and the reception time of the last flit at the destination. Throughput was the average value of the number of flits which a PE receives in each clock cycle. In the evaluation of dynamic communication performance, flocks of messages were sent in the network so that they competed for the output channels. Packets were transmitted with a request probability *r* during *T* clock cycles and the number of flits which reached the destination PE and their transfer times were recorded. The average transfer time and throughput were then calculated and plotted. The request probability *r* was varied. The packet size was 16 flits, and flits were transmitted for 50,000 cycles, i.e., *T*=50000. Two virtual channels per physical link were simulated. The buffer length of each channel was 8 flits.

5.2 Uniform Traffic

In uniform traffic, destinations are randomly chosen with equal probability among the nodes in the network. The result of the uniform traffic pattern is shown in Fig.9. As shown in Fig.9, the throughput of the proposed method is slightly higher than DOR. In the communication pattern such as uniform traffic, the whole network is equally crowded. So the effect of avoidance from crowded links is limited. However in our method, since some packets are directly sent into the channel-H, the load of the channel is distributed. So the throughput is improved.

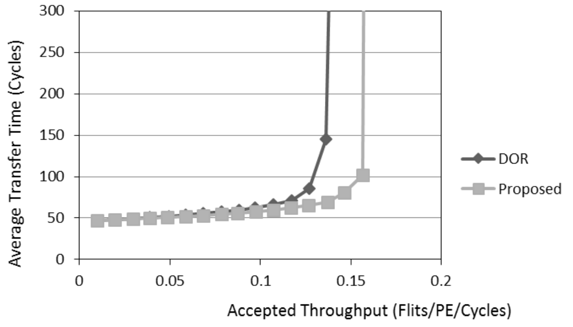


Fig.9 The Result of Uniform Traffic Pattern

5.3 Matrix-Transpose

The matrix-transpose is a traffic pattern based on the transposition of matrix. In this pattern, packets are transmitted between PEs over a diagonal line. In this research, it was assumed that the number of PEs and data are same. About each element of the matrix $A = \{a_{ij}\}$, a_{ij} was assigned to PE (i, j) and the communication of transposition was carried out. Therefore, the traffic pattern of the matrix-transpose is the communication between PE(i, j) and PE(j, i). The result of the matrix-transpose traffic pattern is shown in Fig.10 As shown in Fig.10, the throughput of DOR is a limit by the throughput of 0.1. On the other hand, it turns out that the throughput is extended to 0.14 by the proposed method.

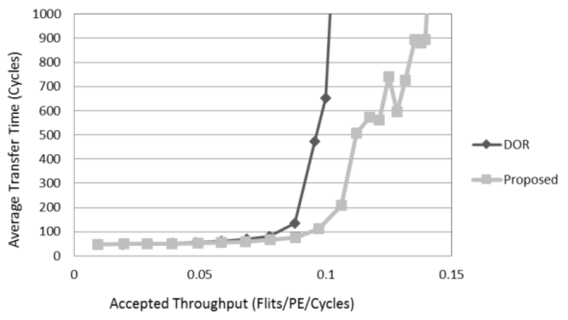


Fig.10 The Result of matrix-transpose

5.4 Bit-reversal

Bit-reversal is traffic pattern to other PE that the bit of the address by binary expression becomes reverse. Since the number of PEs is 256 in this experiment, this pattern is the communication from PE(x, y)=PE$(x_3x_2x_1x_0, y_3y_2y_1y_0)$ to PE$(y_0y_1y_2y_3, x_0x_1x_2x_3)$. The result of the bit-reversal traffic pattern is shown in Fig.13. As the result of matrix transpose, it turns out that the limit of network load improves by the proposed method.

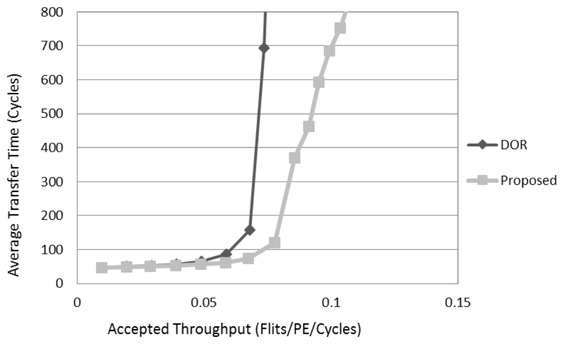


Fig.11 The Result of Bit-reversal

6 Static Performance

6.1 Derivation

In this section, *The Average Number of Shortest Paths (ANSP)* is derived in order to evaluate what pliability the proposed method has. Here, ANSP is defined as "the average number of shortest paths in all the combinations from source to destination". The ANSP of interconnection network with $M = N \times N$ nodes is defined as follows:

$$\text{ANSP} = \sum_{i=0}^{M-1}\sum_{j=0}^{M-1} Np(i,j)/M \times M \qquad (2)$$

Here, $Np(i,j)$ is the number of the shortest paths from node i (the coordinate of node i is $(x_i,\ y_i)$) to node j (the coordinate of node j is $(x_j,\ y_j)$).

Based on (2), ANSP is derived as follows:

- Deterministic Routing

In the Deterministic Routing (DOR), $Np(i,j) = 1$ constantly. Thus, $\sum_{i=0}^{M-1}\sum_{j=0}^{M-1} Np(i,j)$ becomes $M \times M$. So ANSP becomes 1.

- South First Algorithm

In the South First Algorithm of the mesh network with $M = N \times N$ nodes, $Np(i,j)$ is derived as follows:

- When y-coordinate of the node j is lower than or equal to the node i ($y_j \le y_i$), $Np(i,j)$ is 1 because the packet moves as the DOR.
- When y-coordinate of the node j is higher than the node i ($y_j > y_i$), the adaptive routing is carried out. When the minimum number of hops from i to j is n and the distance of y-coordinate is v and the distance of x-coordinate is $h = n - v$, Np becomes $Np(i,j) = {}_nC_v$.

Since $Np(i,j)$ depends on n and v, the table of $Np(i,j)$ from the combination of n and v (from here, it is described as (n,v)) can be obtained. Also, since one (n,v) is obtained from 1 set of combinations of i and j, the number of counts of (i,j) can be obtained from (n,v). From the above tables, the value of ANSP can be derived. In the case of 4×4 mesh, the number of counts of (i,j) (num.of (i,j)) based on n and v is obtained as table 1 (when $v = 0$ or $h = 0$, $Np(i,j) = 1$ obviously. So these cases were excepted from the table). For example, since n and v become $n = 6$ and $v = 3$, "the number of counts of (i,j)" becomes 2 when i,j are upper-left and lower-right, or upper-right and lower-left respectively.

Thus, from the 256 combinations of source and destination in 4×4 mesh with 16 nodes, $Np(i,j) = 20$ is 2 patterns, $Np(i,j) = 10$ is 8 patterns, $Np(i,j) = 6$ is 8, $Np(i,j) = 4$ is 12, $Np(i,j) = 3$ is 24, $Np(i,j) = 2$ is 18, and $Np(i,j) = 1$ is 184 patterns. So $\sum_{i=0}^{M-1}\sum_{j=0}^{M-1} Np(i,j)$ becomes 508, then ANSP≒1.98 by (2).

Table 1 The Number of Counts of (i,j) And the Value of $Np(i,j)$ from n n, v, h in the South First Routing of 4×4 Mesh Network (The Case of ($y_j > y_i$))

n	v	h	num. of (i,j)	$Np(i,j)$
6	3	3	2	20
5	3	2	4	10
	2	3	4	10
4	3	1	6	4
	2	2	8	6
	1	3	6	4
3	2	1	12	3
	1	2	12	3
2	1	1	18	2

- North-South First Algorithm

For the analysis about North-South First Algorithm of 4×4 torus, the table 1 is modified as follows:

- When ($y_j > y_i$), the SF method is carried out when $v \ne 3$, and the RNF method via wraparound link is carried out when $v = 3$. Furthermore, some algorithm of routing via horizontal wraparound link may be carried out when $h = 2{,}3$. Thus the rows of $v = 3$ and $h = 2, 3$ are modified from table 1. Since the row of $v = 3$ and $h = 3$ are changed to $v = 1$ and $h = 1$, the value of n and $Np(i,j)$ are changed. In the case of $h = 2$, since the horizontal wraparound link may be used, $Np(i,j)$ is modified in this case.
- When ($y_j < y_i$), the RNF is carried out when $v = 1$, and the SF may be carried out when $v \ne 1$. as same as above, some algorithm of routing via horizontal wraparound link may be carried out when $h = 2{,}3$. So, all rows except $h = v = 1$ are modified based on table 1. Also, the half of (i,j) patterns in RNF become $Np(i,j) = 1$ in all cases, and those values are modified.

The modified tables by above procedure are shown in table 2 and table 3. Those tables show the value of n, v, h, the number of counts of (i,j), and the value of $Np(i,j)$ in 4×4 torus with 16 nodes. n',v', and h' in tables are the value of n, v, and h when the 4×4 torus is assumed as 4×4 mesh (when the routing process is carried out without wraparound channels). The positions in tables of them are same as table 1. (n), (v), and (h) in those tables are the true values of n, v, and h as routing paths of 4×4 torus.

Thus, from the 256 combinations of source and destination in 4×4 torus with 16 nodes, $Np(i,j) = 6$ is 8, $p(i,j) = 3$ is 38, $Np(i,j) = 2$ is 40, $Np(i,j) = 1$ is 170 patterns. So $\sum_{i=0}^{M-1}\sum_{j=0}^{M-1} Np(i,j)$ becomes 412, then ANSP≒1.61 by (2).

6.2 Evaluation Result

The same evaluation as the previous section was carried out to some network topologies. The evaluation results are shown in table 4. As shown in the table, The ANSP of proposed method is a few lower than SF algorithm of mesh of same size. This may be caused by that the number of hop of mesh network

Table 2 The Number of Counts of (i,j) And the Value of $Np(i,j)$ from n n, v, h in the North-South First Routing of 4×4 Torus Network (The Case of ($y_j > y_i$))

n' (n)	v' (v)	h' (h)	num. of (i,j)	$Np(i,j)$
6(2)	3(1)	3(1)	2	1
5(3)	3(1)	2	4	3,2,3,1
	2	3(1)	4	3
4 (4 or 2)	3(1)	1	6	2 or 1
	2	2	8	6,6,6,6, 6,6,1,1
	1	3(1)	6	2
3	2	1	12	3
	1	2	12	3,3,3,3,3,3, 3,1,3,1,3,1
2	1	1	18	2

Table 3 The Number of Counts of (i,j) And the Value of $Np(i,j)$ from n n, v, h in the North-South First Routing of 4×4 Torus Network (The Case of ($y_j < y_i$))

n' (n)	v' (v)	h' (h)	num. of (i,j)	$Np(i,j)$
6(2)	3(1)	3(1)	2	1
5(3)	3(1)	2	4	1
	2	3(1)	4	1
4 (4 or 2)	3(1)	1	6	1
	2	2	8	1,1,1,1, 3,1,3,1
	1	3(1)	6	1
3	2	1	12	1,1,1,1,1,1, 2,1,2,1,2,1
	1	2	12	3 or 1
2	1	1	18	2 or 1

Table 4 The ANSP of Some Types of Networks

Topology/ Routing Algorithm	Maximum Number of Hops	$\sum_{i=0}^{M-1}\sum_{j=0}^{M-1} Np(i,j$	ANSP
Mesh, Torus / Deterministic	6/4	256	1
4×4 Mesh / SF Algorithm	6	508	1.98
4×2 Mesh / SF Algorithm	4	84	1.31
3×3 Mesh / SF Algorithm	4	115	1.42
4×4 Torus/ NSF Routing	4	347	1.36

is higher than torus network. Then, the comparison with SF algorithm of the mesh network with the same hops as torus was carried out. As the result, it was shown that the ANSP of the proposed method is a little higher than SF algorithm of mesh. From those results, it is thought that the proposed method has about the same performance as SF algorithm of mesh.

7 Conclusions

In this paper, we proposed the North-South First Routing (NSF Routing) which combined the North First method (NF) and the South First method (SF). Also, the communication performance was evaluated by the software simulation. As a result, it was shown that a throughput improves in some communication patterns, and about the same performance of a ANSP static property. From now on, theoretical analysis of the other properties and evaluation about the fault tolerance are remaining as future work.

References

[1] M.M. Hafizur Rahman, Yukinori Sato, Yasuyuki Miura, and Yasushi Inoguchi, Dynamic Communication Performance of a Hierarchical 3D-Torus Network, IASTED, In 10th International Conference Parallel and Distributed Computing and Networks (PDCN 2011), 2011.2.

[2] Yasuyuki Miura, Masahiro Kaneko, M.M.Hafizur Rahman and Shigeyoshi Watanabe, Adaptive Routing Algorithms and Implementation for TESH Network, Communications and Network (CN), Vol.5, No.1, pp.34-49, 2013.02.

[3] W.J. Dally and Hiromichi Aoki, Deadlock-Free Adaptive Routing in Multicomputer Networks Using Virtual Channels, IEEE Trans. On Parallel and Distributed Systems, Vol,4, pp. 466-475, 1993

[4] M.P.Merlin and J.P.Schweitzer, Deadlock Avoidance in Store-and-Forward Networks-1: Store and Forward Deadlock, IEEE Trans. On Comm., Vol.COM-28, No.3, pp.345-354, 1980

[5] W.J.Dally and C.L.Seitz. Deadlock-Free Message Rouring in Multiprocessor interconnection Networks. IEEE Trans. On Computers, Vol. C-36, No.5, pp.547-553, 1987.

[6] C. S. Yang and Y. M. Tsai, "Adaptive Routing in k-ary ncube Multicomputers," Proc. of ICPADS '96, pp. 404-411, 1996.

[7] C. J. Glass and L. M. Ni, "Maximally Fully Adaptive Routing in 2D Meshes," ISCA92, pp. 278-287, 1992.

[8] C.J.Glass, L.M.Ni, The Turn Model for Adaptive Routing, The 25th Annual International Symposium on Computer Architecture, pp.441-450, 1998.

[9] Jie Wu, A fault-tolerant and deadlock-free routing protocol in 2D meshes based on odd-even turn model, IEEE Trans. on Computers, Vol.52, No.9, pp.1154-1169, 2003.

[10] A.Jouraku, M.Koibuchi, H.Amano, An Effective Design of Deadlock-Free Routing Algorithms Based on 2D Turn Model for Irregular Networks, IEEE Trans. on Parallel and Distributed Systems, Vol.18, No.3, pp.320-333, 2007.

[11] J.Duato, A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks, IEEE Trans. on Parallel and Distributed Systems, Vol.4, No.12, pp.1320-1331, 1993.

[12] E. Fleury and P.Fraigniaud, A General Theory for Deadlock Avoidance in Wormhole-Routing Networks, IEEE Trans. Parallel and Distributed Systems, Vol.9, No.7, pp.626-638, 1998.

[13] Yasuyuki Miura, Kentaro Shimozono, Kazuya Matoyama, Shigeyoshi Watanabe, An Adaptive Routing of the 2-D Torus Network Based on Turn Model, Proc. of 4th International Workshop on Advances in Networking and Computing, pp.587-591, 2013.12.

A Prediction-Based Approach for Collective I/O Optimization

Chaoqun Sha[#], Hua Nie[#], Huaiming Song[*], Chenming Zheng[#], Xiaojun Yang[*], Chungjin Hu[#]
[#]School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083, P.R.China
[*]Dawning Information Industry Co., Ltd. Beijing 100193, P.R.China

Abstract - *Computing is becoming increasingly data-centric. I/O data access is identified as a critical performance bottleneck of end-to-end performance of high-end computing. In this paper, we propose a lightweight approach to automatically identify and prevent harmful collective I/O specifically on MPI_IO_Read. In our approach, we first give an analytic model to analyze the performance of independent and collective I/O. Then, we design a mechanism to put our model in use. At last, we incorporate our fine-grained mechanism into the ROMIO MPI I/O library for performance testing. Experimental results show that the accuracy of our analytic model reaches about 92%. The proposed model-prediction mechanism is simple and practical, with a complexity of O(1). Analytical and experimental results confirm the practical usability of the proposed collective I/O improvement.*

Keywords: *collective I/O; Data-intensive; Independent I/O; Two-phase collective I/O*

1 Introduction

Computing is becoming increasingly data-centric. Rapid advance in microprocessor technology makes computing speed unprecedentedly fast, whereas newly emerged computer applications such as computer animation and information retrieval are more and more data intensive. Computing practitioners are faced with mountains of data today [1]. Unfortunately, despite advanced parallel file systems have been developed in recent years, e.g. PVFS2[2], Lustre[3], GPFS [4], etc., the storage systems are still lagging far behind and become a critical technical hurdle of further success of high performance computing (HPC). Increasing I/O performance is a timely research issue facing the HPC community.

In this paper, we introduce a lightweight solution to achieve good I/O performance by several steps, which mainly focus on improving the reading performance. First, we propose an analytic model to theoretically analyze the performance of independent I/O and two-phase collective I/O in different scenarios. Then, we use this model to predict and compare their performance. At last, we guide the selection of MPI-IO Read functions with the results of this model to optimize I/O performance. The results show that, the accuracy of our model is up to 92% and with the improved version of ROMIO, the improvement of I/O bandwidth could be up to about 40%.

For general and practical purpose, our study has the following design goals:

- *Approachable parameters in analytic model*: the parameters used in our model should be all obtainable. In this model, we only assume the constant cost on the initialization of MPI-IO Calls, which may vary with different hardware and software configurations. For a determined HPC scenario, this cost is a constant and measurable.
- *High accuracy*: The proposed model can accurately model the cost separately on I/O layer and local memory layer, from which the accuracy is up to 92% in our experimental results for different I/O access patterns.
- *Lightweight implementation*: In the current implementation of ROMIO, the performance of collective I/O is roughly evaluated, which could not always guarantee good performance for diverse data-intensive applications. But, its interface can be easily reused to provide an autonomic function to optimize I/O performance. In this work, a mechanism is given and used in ROMIO to guide the use of I/O operations instead of its currently simple offsets comparison and evaluation. The time complexity of our work is O(1), hence, the overhead of our method is trivial.

In summary, this paper plans to offer a lightweight approach to optimize I/O performance of data-intensive applications – specifically, we guide the selection of I/O operations by using an analytic model. The remainder of this paper is organized as follows: the background is presented in Section2. Section3 discuss the analytic model and its implementation in ROMIO, followed by its evaluation and experimental results in Section4.. Section 5 concludes the paper.

2 Background

2.1 I/O Characteristics of Data-intensive Applications

Studies [1,5,6] of the I/O characteristics of data-intensive applications have showed clearly that one major reason of the low I/O performance is that, these applications usually need to access a large number of small, noncontiguous pieces of data.

However, for good performance, the size of the I/O requests should be large. Some scientists [7] found that, in data-intensive applications, the data requested by different processes are often interleaved and may together span larger, contiguous portions of the file. As a result, many techniques are designed to benefit from this I/O character. In general, the data access patterns can be categorized into 3 sets: random access pattern, sequential access pattern, and data interleaved pattern. In our work, we first found that the performance of two-phase collective I/O is sensitively related to the size of interleaved data. Beyond the qualitative analysis in ROMIO, our model can analyze I/O performance quantitatively based on this. For this part, we will discuss more in the following sections.

2.2 Independent I/O

The key idea of independent I/O is to let processes independently access I/O servers, shown as Fig.1. In independent I/O, standard POSIX is used as I/O interface. In this figure, two I/O nodes serve six processes from P0 to P5. In Fig.1, it shows clearly that all I/O requests are delivered to I/O servers and processed independently. In independent I/O, when the size of requested data is greater than the stripe size used on I/O server, one I/O request will be served by the cooperation of both two I/O servers. In this case, the factual I/O requests being processed on I/O servers are twice as much as the original requests initialized by computing processes.

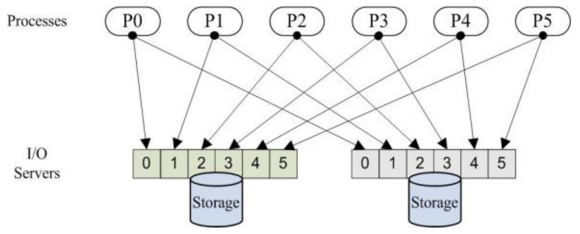


Fig.1. The rationale of independent I/O

Because of the large number of direct communication on I/O layer and the relative low efficiency of the utilization of I/O bandwidth, the performance of independent I/O is not ideal for many scientific applications. Some researchers believe that memory, with higher bandwidth compared to that of I/O, and may be used to speed up the I/O performance. The technique, two-phase collective I/O, is introduced based on this idea.

2.3 Two-phase collective I/O

Two-phase collective I/O merges small noncontiguous requests from the group of processes into the larger size of contiguous data requests with fewer I/O communications. Some processes, named as aggregators, are chosen from the group to communicate with I/O servers. The local buffers of the aggregators are employed to do the data exchange in the group, which is named as file domains.

For clearer description, the rationale of collective I/O is shown as Fig.2. In this figure, P0 and P3 serve as aggregators and their local buffers are used as file domains. In the first phase, P0 and P3 will send I/O requests assembled from three processes to I/O servers, respectively; in the second phase, other processes will retrieve their requested data from aggregators. The second phase involves overhead for communication in processes. However, due to the bandwidth advantage of point-to-point communication and the benefits of fewer I/O accesses at I/O server layer, the performance of collective I/O is believed better than independent I/O.

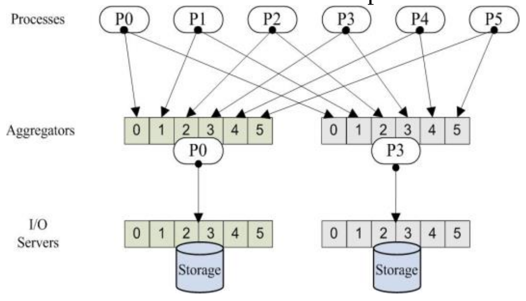


Fig.2. The rationale of collective I/O

In ROMIO, the condition for the use of two-phase collective I/O is to compare the beginning and end offsets of requested data in the group of processes in one time MPIIO call: if they are interleaved, do collective I/O. Otherwise, proceed independent I/O. In practice, it is simple and high efficient. However, this kind of coarse-grained algorithm may make mistakes and decrease the I/O performance sometimes.

2.4 Resonance Phenomenon in independent I/O and collective I/O

Zhang et al. [8] pointed out that for some specific scenarios, even though the data-intensive applications have the character as data interleaved access pattern, their I/O performance is not as good as expected. Sometimes, it is even worse than the independent I/O.

This work is the start of the art to analyze deeper in the performance of two-phase collective I/O and independent I/O. Based on their work, our solution starts from the question as when data-intensive applications can benefit from two-phase collective I/O and when they cannot. In the first step, we analyze their performance theoretically.

3 Analytic Model on Collective I/O and Independent I/O

Before giving any formula to model I/O performance, we first describe parameters used in our model, whose efficiency and measurability will be verified in the experimental part.

3.1 Terminology

Table I: Terminology of Parallel IO

Parameters	Description
n	Number of processes employed in one application
$B_{I/O}$	I/O Bandwidth
m	Number of employed I/O servers
s	Stripe size in I/O servers
$B_{network}$	Network Bandwidth

k	Number of aggregators chosen to proceed I/O accesses in two-phase collective I/O
D_{raw}	Size of data requested by one process for one time MPI-IO function call
D	The whole chunk of data from the beginning offset to the end offset requested in the group of processes for one time collective I/O function call
$D_{aggregator}$	Size of data requested by one aggregator in one time collective I/O function call
λ	Initialization time for one time MPI-IO function call
$\alpha_{network}$	Start up time for data exchange in file domains
$\alpha_{I/O}$	Start up time for data exchange in I/O level
ρ	The ratio of data interleaved among raw I/O requests
$T_{collective}$	Total time cost of collective I/O
T_{domain}	Total time cost of data exchange in file domain for collective I/O
$T_{C\_I/O}$	Total time cost of data exchange in I/O level for collective I/O
$T_{independent}$	Total time cost of independent I/O

3.2 Model on independent I/O

In parallel communication implemented by MPI-IO2, initialization time consumed for synchronization in the communication group is inevitable. In practice, this cost may relate to HW/SW configurations, the number of communicators, the number of computing nodes and processes, and even the skills of the programmers. As a result, it is difficult to evaluate this cost theoretically. In our model, we value it as a constant λ, since for a determined HPC scenario, it is firmed. Otherwise, the number of I/O requests could be different for different programmers, which will directly affect the I/O performance. For keeping the generality, in our model, we assume that the function as FILE_SET_VIEW [9] is used and the number of the MPI-IO calls is the minimum.

Here, we model the performance of independent I/O and two-phase collective I/O with only horizontal layout manner [10]. Because of other layout manners, the performance of collective I/O is obvious worse than independent I/O. Then it is less meaningful to compare their performance under these layout manners.

For independent I/O, Studies[10,13] gives the layout model for independent I/O with different layout models. We extend their horizontal model to evaluate the performance of independent I/O in our model.

In one time MPI-IO call, the group of processes with size n, access m I/O servers. For each I/O request, the size is D_{raw}.

If $D_{raw} \le s$ and $n \le m$

$$T_{independent} = \lambda + \alpha_{I/O} + \frac{D_{raw}}{B_{I/O}} \tag{1}$$

Or if $D_{raw} \le s$ and $n > m$

$$T_{independent} = \lambda + \left\lceil \frac{n}{m} \right\rceil \alpha_{I/O} + \frac{D_{raw}}{B_{I/O}} \left\lceil \frac{n}{m} \right\rceil \tag{2}$$

Or if $D_{raw} > s$,

$$T_{independent} = \lambda + \left\lceil \frac{D_{raw} n}{sm} \right\rceil \alpha_{I/O} + \frac{s}{B_{I/O}} \left\lceil \frac{D_{raw} n}{sm} \right\rceil \tag{3}$$

3.3 Model on Two-Phase Collective I/O

For analyzing two-phase collective I/O under the same situation, we use the same assumption as independent I/O: n raw I/O requests with the size D_{raw}.

For two-phase collective I/O, the total time cost comes from three parts: initialization, cost in file domain and cost in I/O communication.

$$T_{collective} = \lambda + T_{domain} + T_{I/O}$$

For data exchange in file domain, the resource (memory) is exclusively consumed. In another words, one aggregator could only communicate with one process at one time, either another aggregator or one process in its domain. The I/O request initialized by aggregator is named as $D_{aggregator}$. For theoretical analysis, we simplify the diversity of the sizes of $D_{aggregator}$ and treat them as equal. In practice, it is possible that the size of data requested by aggregators are various according to the configuration of two-phase collective I/O configuration by users [9]. Then, the size of $D_{aggregator}$ is $D_{aggregator} = \frac{D}{k}$.

The total cost T_{domain} is:

$$T_{domain} = \left\lceil \frac{n}{\min(k, \frac{B_{domain}}{\rho D_{raw}})} \right\rceil \alpha_{I/O} + \left\lceil \frac{n}{\min(k, \frac{B_{domain}}{\rho D_{raw}})} \right\rceil \frac{\rho D_{raw}}{B_{I/O}}$$

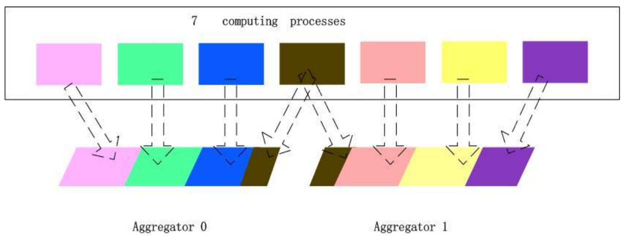


Fig.3. Two Aggregators Serve 7 Computing Processes

As discussed in the previous section, the number of processes in I/O communication n is equal or greater than the number of aggregators k, and the size of initial I/O request D_{raw} usually equal or smaller than the size of data requested by aggregator, since suppose every aggregator could hold data for itself for reducing communication cost, the requested data need to be transferred at most twice, and the data exchange size is about $\frac{D_{raw}}{2}$, shown as Fig. 4. In theoretical, we analyze the worst case as given.

For the communication cost on I/O layer, the situation is more complicated: all of different data stripe sizes, patterns and the number of employed I/O servers will directly influence performance. When collective I/O requests delivered to I/O layer, the number of these I/O requests is the number of aggregators k, the size of each collective request $D_{I/O}$ is

$$D_{I/O} = D_{aggregator} = \frac{D}{k}$$

For horizontal manner, if $D_{aggregator} \leq s$ and $k \leq m$; then

$$T_{I/O} = \alpha_{I/O} + \frac{D_{aggregator}}{B_{I/O}}$$

Or if $D_{I/O} \leq s$ and $k > m$; then

$$T_{I/O} = \left\lceil \frac{k}{m} \right\rceil \alpha_{I/O} + \frac{D_{aggregator}}{B_{I/O}} \left\lceil \frac{k}{m} \right\rceil$$

if $D_{I/O} > s$; here we make an assumption that the load balance of data deployment in m I/O servers is achieved. Then

$$T_{I/O} = \left\lceil \frac{kD}{sm} \right\rceil \alpha + \frac{s}{B_{I/O}} \left\lceil \frac{kD}{sm} \right\rceil$$

After we theoretically model the performance of collective I/O and independent I/O, its correctness and uses should be verified. In the next section, we will start from the verification of this analytic model.

3.4 Implementation

The implementation of our mechanism is straight forward in MPICH2-1.2. The user collective I/O interface is MPIOI_File_read_all(), which in turn calls ADIO_ReadStridedColl(), one of the common ADIO functions applied by MPI to separate the concerns for file systems. This ADIO code will finally call a file-system-specific function to perform the actual I/O operation. But before that, it will determine whether or not the following I/O should be collective. The course of the decision is a typical redundant computing. As presented above, each process in the communicator will exchange the offset of their read interval, then compute the possibility of interleaving. The real decision is only made by a if-else statement, in which true lead to collective I/O; false to independent. Therefore, we modify the if-else condition, replace it with the decision Boolean expression $T_{independent} \leq T_{collective}$. The calculation of each time cost strictly conforms to the formulas (1)-(4), hence trivial for further description. Here we list all the parameters used in expression calculation. The source column describes the method to obtain them. Original code implies that this parameter is a variable; hint means parameter is passed through MPI hint mechanism by user; Modification indicates that we need to add more code to calculate the parameter. The last column gives a detailed description on how to obtain the parameter. Hint type is ignored here for their simplicity.

Table II. Parameters of implementation

Parameters	Source	Description
n	Original Code	Reuse nprocs
ρ	Modification	Calculated with offset_list, len_list from all processes
k	Original Code	Reuse nprocs_for_coll
D_{raw}	Modification	Calculated with offset_list, len_list
D	Modification	Calculated with start_offset, end_offset
$D_{aggregator}$	Modification	Calculated with start_offset, end_offset
λ	Modification	Timing the preparation phase
$B_{network}$, $B_{I/O}$, m, s, $\alpha_{network}$, $\alpha_{I/O}$	Hint	Through MPI hint mechanism

4 Experiments

4.1 Experimental Setup

Our experiments platform is a 16-node cluster, in which all nodes are equipped with 4X InfiniBand network. Each node has 2 Quad-Core Processors, whose model is AMD Opteron(tm) Processor 2376, 8GB memory and 250GB 7200RPM SATA hard drive. The OS is Ubuntu 9.04, Linux kernel 2.6.28.10. We use PVFS2 version 2.8.1 as our parallel file system. For all experiments, we employ 4 nodes as computing processes and use the file with size about 4G. For each node, we generate 5-25 processes with different benchmarks. For the model verification, we choose three representative benchmarks.

- *IOR*: The IOR benchmark is a benchmark developed by LLNL and usually used to measure the performance of storage systems. Otherwise, IOR can test the performance of parallel I/O with several I/O interfaces, including POSIX, collective I/O, as well as I/O through higher level libraries [11]. In our experiments, its sequential pattern is used to evaluate the performance of collective I/O and independent I/O, test the accuracy and the efficiency of our model.
- *MPI-Tile-I/O*: The MPI-Tile-I/O benchmark is a synthetic benchmark, which is part of the Parallel I/O benchmarking Consortium benchmark suite. It simulates tile access on one two-dimensional dataset, with the overlapped data between sequential tiles. In MPI-Tile-I/O, the implementation of both POSIX I/O and MPI I/O are provided for users. In our experiments, its character as data interleaved access pattern is used to verify our model.
- *MPI-IO Test*: MPI-IO test is a synthetic checkpoint tool developed by LANL, which supports several parallel file systems. In our experiments, MPI-IO Test is used as one kind of data-intensive applications in HPC to test the correctness and efficiency of our model.

4.2 Performance evaluation of Independent I/O and Two-phase Collective I/O

As discussed previously, it is possible for independent I/O and two-phase collective I/O that they may achieve better performance alternatively. The purpose of this set of experiments is to prove this argument. We will respectively experiment on IOR and MPI-Tile IO.

- Experiments with IOR Benchmark

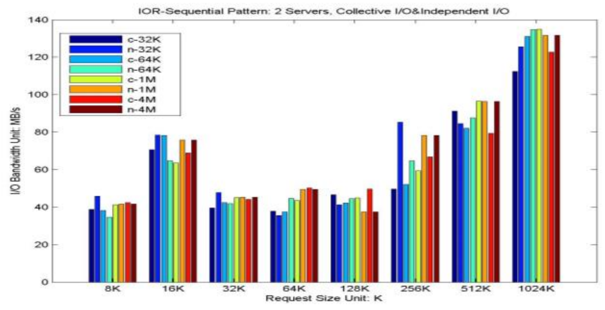


Fig.4 The performance of collective I/O and independent I/O in IOR with 2 I/O servers: In this figures, 'c-stripe size' and 'n-stripe size' respectively represent the performance of collective I/O and independent I/O. For example, c-1M means the performance of collective I/O with stripe size as 1M.

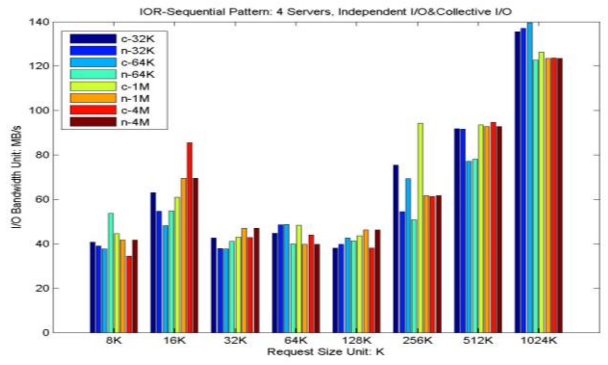


Fig.5 The performance of collective I/O and independent I/O in IOR with 4 I/O servers.

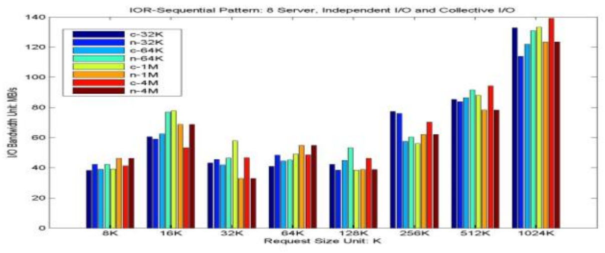


Fig.6 The performance of collective I/O and independent I/O in IOR with 8 I/O servers

In this set of the experiments, we propose to compare the performance of independent I/O and two-phase collective I/O on IOR with different system configurations. For this goal, we vary the number of I/O servers (2 I/O servers, 4 I/O servers, and 8 I/O servers), change stripe size (32K, 64K, 1M and 4M) and increase request size (8K, 16K, 32K, 64K, 128K, 256K, 512K, 1M). For this set of experiments, we use 4 processors as clients, which generate 20 processes in total. The results for separate I/O servers are shown as Fig.4, 5, 6.

- Experiments with MPI-Tile IO

In this set of the experiments, we emphasize on comparing the performance of independent I/O and two-phase collective I/O on interleaved data access pattern with different system configurations. For this goal, we employ 2 I/O servers, with various stripe sizes as 32K, 64K, 1M and 4M. For capturing the performance variety with different interleaved data size, we fix the request size as 1M and increase the sizes of interleaved data from 20K to 180K. For this set of experiments, we use 4 processors as clients, which generate 100 processes in total. The results are shown as Fig.7.

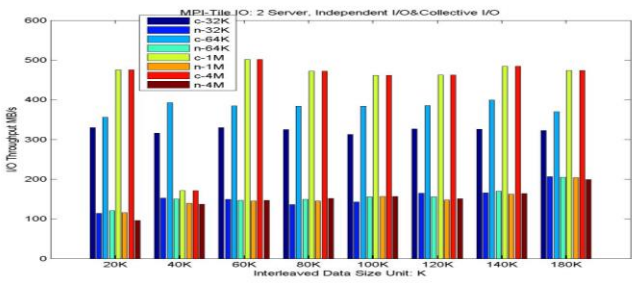


Fig.7 The performance of collective I/O and independent I/O in MPI-Tile IO with 2 I/O servers.

4.3 Model Verification

- Parameter Measurement

In our model, many parameters are introduced to predict the performance of collective I/O and independent I/O. As a result, the first thing for us is to make sure that all these parameters could be measurable or estimated. In fact, all parameters used in our model could be categorized into four columns: I/O configurations (I/O server number, stripe size), collective I/O configuration (aggregator number), environmental parameters (I/O bandwidth, network bandwidth, overheads), I/O access patterns (interleaved data size, request size), most of which can be measured directly from the systems except I/O access patterns. However, some scientists focus on the study on I/O signature, and I/O trace [12], who can capture and summarize I/O access patterns. Otherwise, for two-phase collective I/O, it is possible to capture and predict the separate performance of file domain layer and I/O layer. In MPICH2-1.2, functions of MPI-IO are all stored in ADIOI_Hints(), from which we could easily trace the separate cost in two-phase as discussed in the implementation part collective I/O, moreover, use this info in our model. In summary, all parameters used in our model are measurable.

- Model Verification

In this set of the experiments, we will test the correctness and accuracy of our model. For this goal, we compare the results of the tests from real system and the prediction results from our model for all three benchmarks. Since we tried to simulate the real scenarios, in which the storage resources are usually short of, in this set of experiments, we fix the server number as 2 I/O servers. Additionally, for fair evaluation, we value the mathematical expectation as: $E(B_{I/O}) = \frac{\sum_{i=8,16,...1024} B_{s_j - I/O} * D_{raw}}{\sum i}$. Here, $B_{I/O}, B'_{I/O}$ are used to represent the value from tests and model respectively, s_j means the different stripe size on I/O servers, $s_j \in (32K, 64K, 1M, 4M)$, i means the value of request size, $i \in (8K, 16K, ...1024K)$. Specifically, in the context of MPI-Tile IO benchmark, i means the value of overlapped data size, $i \in (20K, 40K, ...120K)$.

In Fig. 8, 9, 10 we show the comparison of real runs and model estimation for all three benchmarks. Table 1 shows the errors of our model for the three benchmarks. From the results, we can conclude that our analytic model can accurately predict the performance of independent I/O and two-phase collective I/O. In the next section, we will explore its efficiency on IO optimization.

TABLE1
THE ERRORS OF OUR MODEL IN COLLECTIVE I/O AND INDEPENDENT I/O

Stripe Size / I/O Operation		32K	64K	1M	4M
IOR	Collective	18.6%	8.16%	8.65%	8.33%
	Independent	11.2%	17.4%	12.3%	8.49%
MPI-Tile IO	Collective	11.1%	11.8%	16.5%	17.9%
	Independent	17.5%	12.5%	17.5%	15.7%
MPI-IO	Collective	17.3%	7.1%	18.6%	18.9%
	Independent	9.89%	17.5%	12.8%	8.4%

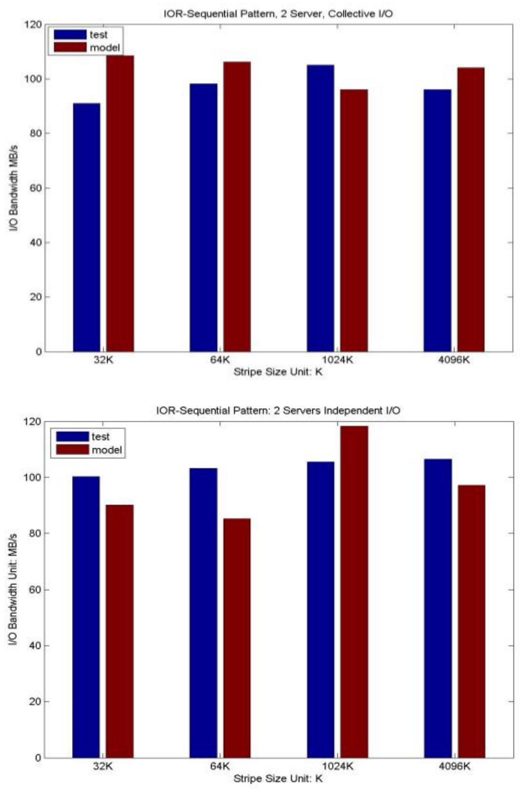


Fig.8 the results of model verification in IOR-sequential access pattern

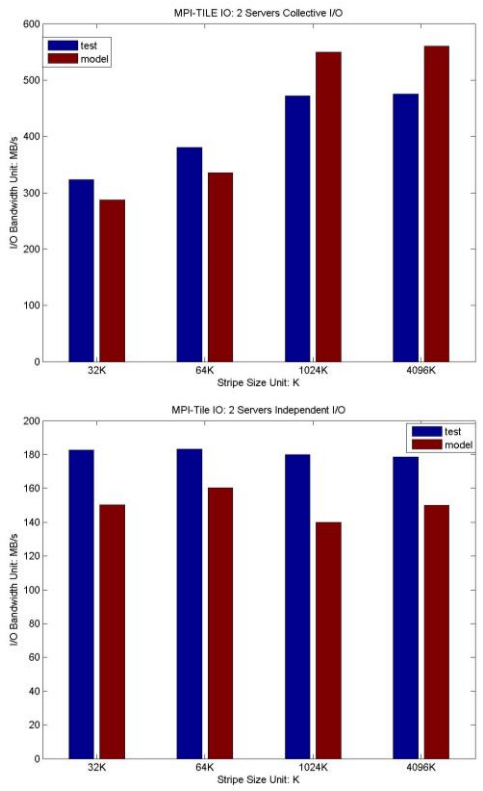


Fig.9 the results of model verification in MPI-Tile IO separately for collective I/O and independent I/O

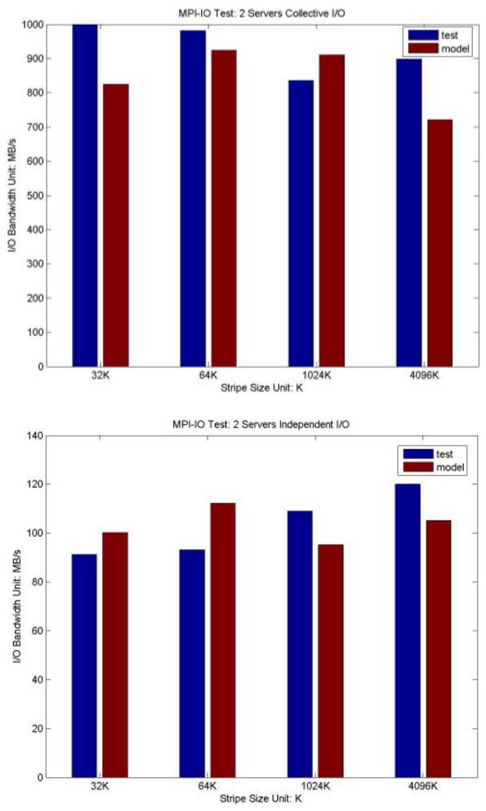


Fig.10 the results of model verification in MPI-IO Test separately for collective I/O and independent I/O

4.4 I/O Performance Optimization

As claimed at the beginning, our model can be used to optimize the I/O performance. With our implementation in ROMIO, we could review back these three benchmarks to see the performance of our model. In this set of experiments, our goal is to test the optimization efficiency of our mechanism. In our ROMIO version, we use θ to represent the performance advantage of collective I/O, in our experience, when $\theta = \frac{(collective\ \ performance - independent\ performance)}{independent\ \ performance} > 15\%$, we will use collective I/O to take IO operation. Otherwise, independent I/O is used. The results are shown in Fig. 11, 12 and 13. We can see that the proposed model can achieve upto 40% performance improvement.

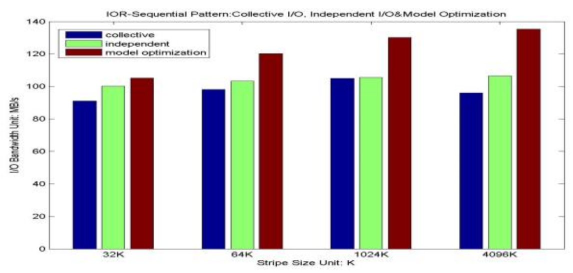


Fig.11 The Performance Improvement in IOR: As shown clearly, our model optimization yields the best performance compared to both collective I/O and independent I/O. The greatest performance improvement compared separately to collective I/O and independent I/O are 40.2% and 32.1%, whose configurations are both 4M stripe size.

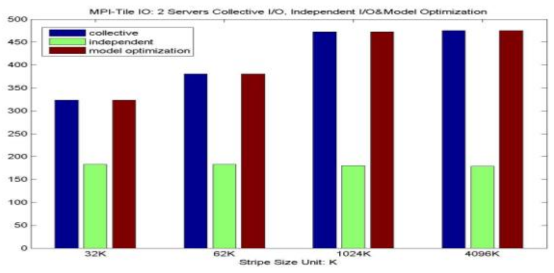


Fig.12 The Performance Improvement in MPI-Tile IO: As shown, our model optimization has the same performance as two-phase collective I/O. In our mechanism, only when collective I/O has worse performance (less than 15% advantage) the independent I/O will be called. In MPI-Tile IO, collective I/O has times performance advantage. As a result, in every MPI-IO read call, we will choose MPI_IO_READ_ALL() to use collective I/O instead of the use of independent I/O. Then, the performance improvement of model optimization is the same as that of two-phase collective I/O.

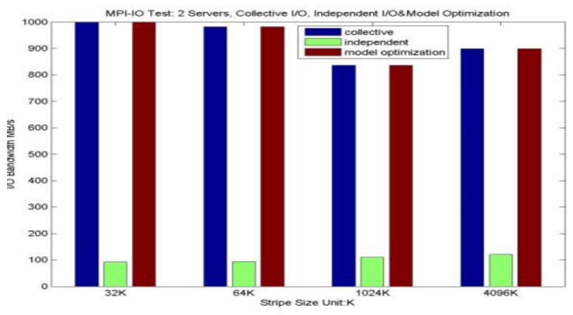


Fig.13 The Performance Improvement in MPI-IO test: This case is similar with MPI-Tile IO, in which two-phase collective I/O has clear advantage. The results show our model makes the correct choice for each MPI-IO read call.

5 Conclusion

We propose a prediction-based approach for collective I/O optimization. We have conducted a full-cycle of the performance optimization study. Through our study we have further demonstrates the need of performance optimization of two-phase collective I/O. We have shown increasing the parallelism of parallel file system does not necessary improve the I/O performance. As shown clearly in our IOR benchmark studies, with the growth of I/O servers, the I/O performance does not move to a better. We have illustrated that the performance of two-phase collective I/O could vary wildly from one I/O access pattern to another I/O access pattern. Prediction-guided collective I/O can improve the I/O performance considerably. We also have exhibited that sometime the simple independent I/O approach could be the best choice.

6 References

[1] R. Ross, R. Thakur, W. Loewe, and R. Latham, “Parallel i/o in practice,” in SC ’06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing. New York, NY, USA: ACM, 2006, p. 216.

[2] P. H. Carns, W. B. Ligon, III, R. B. Ross, and R. Thakur, “PVFS: A parallel file system for linux clusters,” In Proceedings of the 4th Annual Linux Showcase and Conference. USENIX Association, 2000, pp. 317–327.

[3] W. Yu, J. Vetter, R. S. Canon, and S. Jiang, “Exploiting lustre file joining for effective collective io,” in CCGRID ’07: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid. Washington, DC, USA: IEEE Computer Society, 2007, pp. 267–274.

[4] J.-P. Prost, R. Treumann, R. Hedges, B. Jia, and A. Koniges, “Mpi-io/gpfs, an optimized implementation of mpi-io on top of gpfs,” SC Conference, vol. 0, p. 58, 2001.

[5] (2009) Computing and storage. [Online]. Available: http://www.alcf.anl.gov/resources/storage.php

[6] (2008) Worldwide lhc computing grid. [Online]. Available: http://public.web.cern.ch/public/en/LHC/Computing-en.html

[7] R. Thakur, W. Gropp, and E. Lusk, “Data sieving and collective i/o in romio,” Frontiers of Massively Parallel Processing, Symposium on the, vol. 0, p. 182, 1999.

[8] Z. Zhang, K. Lee, X. Ma, and Y. Zhou, “Pfc: Transparent optimization of existing prefetching strategies for multi-level storage systems,” Distributed Computing Systems, International Conference on, vol. 0, pp. 740–751, 2008.

[9] R. Thakur and R. T. E. Lusk, “Users guide for romio: A high-performance, portable mpi-io implementation,” 2002.

[10] X.-H. Sun, Y. Chen, and Y. Yin, “Data layout optimization for petascale file systems,” in PDSW ’09: Proceedings of the 4th Annual Workshop on Petascale Data Storage. New York, NY, USA: ACM, 2009, pp. 11–15.

[11] IOR Benchmark, www.llnl.gov/asci/purple/benchmarks/limited/ior.

[12] S. Byna, Y. Chen, X.-H. Sun, R. Thakur, and W. Gropp, “Parallel i/o prefetching using mpi file caching and i/o signatures,” in SC ’08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing. Piscataway, NJ, USA: IEEE Press, 2008, pp. 1–12.

[13] H. Song, Y. Yin, Y. Chen, X.-H. Sun: A cost-intelligent application-specific data layout scheme for parallel file systems. in Proc. of the 20th International ACM Symposium on High Performance Distributed Computing (HPDC’11), pp. 37-48, 2011

GPU-based String Matching Method using Warp Shuffle Instructions for Service-oriented Routers

Satoshi Koibuchi, Kazumasa Ikeuchi, Shinichi Ishida, Hiroaki Nishi
Graduate School of Science and Technology, Keio University, Japan
{satoshi, ikeuchi, sin}@west.sd.keio.ac.jp, west@sd.keio.ac.jp

***Abstract*— Service-oriented Router (SoR), a new router architecture for providing useful Internet services that could not be given by a traditional router. As a service of SoR, to prevent a network intrusion in a network will become a significant service. To attain the service, we proposed SoR-Network Intrusion Detection System (SoR-NIDS) using deep packet inspection (DPI) in order to protect malicious streams on the router. Typical applications like this SoR-NIDS require an effective analysis mechanism of traffic information. Namely, a string matching function is an essential problem. Moreover, router architecture becomes more commoditized. It will be possible in the future to use GPUs for accelerating processing on routers. We propose a new GPU-based string matching design and efficient multistring matching function for multiple streams on a service-oriented router using warp shuffle (shfl) instructions to accelerate data stream analysis. The proposed method was evaluated, and the effectiveness of the method was confirmed.**

Keywords—Service-oriented router; GPU; string matching; warp shuffle instructions; application layer analysis

I. Introduction

The Internet has become an indispensable communication tool, and the amount of Internet traffic is continually increasing. Accordingly, the threat of attacks on the Internet is also increasing. In particular, the number of attacks that exploit software vulnerabilities on client PCs is increasing. In general, vendors provide software patches for known vulnerabilities; however, the user is responsible for the installation of such patches, which can contribute to lack of security. In the client–server network model, the administrator of each end-host has discretion over all security; thus, the security level depends on the discretion. Therefore, a new security system that does not depend on the security level of the end-host is required. In future, many sensors will be distributed around the world, and these sensors will not have sufficient battery power, processing power, and memory. In addition, it may be difficult to install antivirus software on such sensors. If these sensors are cracked, a new threat will be introduced. This will strengthen the need for antivirus functionality on the Internet.

A router relays communication between end-hosts in a network. A typical router forwards a packet to the appropriate destination based on a routing table and the destination IP address contained in the packet. We propose a service-oriented router (SoR) as new router architecture. SoR reconstructs TCP streams in a router using the packet information of the plurality of fragments in a network by considering memory efficiency. Moreover, SoR can decode, extract, and analyze application layer information. In addition, based on the results of analysis, SoR can provide a new service. A network intrusion detection system (NIDS) will be one of the new services that SoR can provide, which we refer to as SoR-NIDS. It is possible to increase security of an end-host network by matching with a black list. In addition, NIDS can prevent potential threats and provide warnings to users. In general, NIDS searches for a signature represented as a string or regular expression to distinguish whether the target data can be permitted. SoR-NIDS can be provided to all Internet users. Similar to a general antivirus system, it is possible to provide robust security against new attack methods by updating the blacklist on SoR.

However, a problem must be solved before realizing SoR-NIDS. First, wire-rate processing throughput must be achieved in the router. Second, intrusion detection processing must be realized for multiple streams. A large number of streams flow through a router; thus, it is necessary to achieve high throughput processing for multiple streams. As an existing method, dedicated hardware, such as network processors or FPGAs, have been studied to achieve high throughput [1][2]. However, to reflect recent backbone router trends, the use of conventional cost-effective devices will be a practical solution to achieve SoR-NIDS. In addition, it is preferable to implement programmability and to continue with architecture trends for Internet backbone routers, i.e., commodity devices.

To achieve high-throughput processing of string matching functions, it is indispensable to parallelize the process. Graphics processing units (GPUs) [3] are used as co-processors to realize high throughput. A GPU has hundreds or thousands of processing cores on one semiconductor die. GPUs can process at high throughput by using these cores in parallel. In addition, a GPU has dedicated memory, and its bandwidth is several times high than main memory. This means that it can obtain higher processor-memory bandwidth than common processors if conditions are appropriate.

In a general system, a GPU is connected through a peripheral component interconnect (PCI) interface that transfers data and instructions. Recently, router architecture that can connect general-purpose co-processors to the system through a PCI interface have been developed [4, 5], and fast NIDS methods using GPUs have also been studied [6, 7]. Therefore, a GPU can be considered effective as a high-speed method for NIDS processing on SoR.

We propose a fast NIDS processing method for router architecture equipped with a general-purpose GPU and CPU using warp shuffle (shfl) instructions, which are extension instructions of recent GPUs. In this study, we discuss only the problem of string matching that does not include regular expressions because it is possible that a SoR-NIDS service can be provided using only string matching initially. This remainder of this paper is organized as follows. We describe a heterogeneous CPU and GPU system in Section 2. We discuss related string search research in Section 3. In Section 4, we propose a string matching method that is more efficient for multiple streams and discuss algorithmic descriptions using shfl functions. An evaluation of the proposed method is presented in Section 5, and the paper is concluded in Section 6.

II. Heterogeneous CPU and GPU system

Since it is impossible to control an entire program with only a GPU, it is necessary to use the GPU as a co-processor with the CPU. A GPU is connected to a system board through a PCI interface. The main system with a CPU and main memory is called a "host," and the GPU subsystem connected to the host via a PCI interface is called a "device." Instructions and all data processing performed on the device must be transferred through the PCI interface from the host. Therefore, processing throughput of the GPU is limited to the PCI interface's bandwidth. For this reason, GPU processing is preferable because the cost of calculation processing is much larger than that of memory transfer.

GPU processing is based on the single instruction multiple data (SIMD) processing scheme. SIMD processes data to multiple columns of a single instruction sequence. NVIDIA GPUs have adopted parallelism with the SIMD scheme in part by operating 32 processing cores as one operation unit called a streaming multiprocessor (SM). However, NVIDIA released the Kepler architecture [8] in 2012. The Kepler architecture adopts a next-generation streaming multiprocessor (SMX). In an SMX, one operation unit consists of 192 processing cores. An SMX has 192 single-precision arithmetic units, 64 double-precision arithmetic units, 32 dedicated function operation units, and 32 load/store units. In addition, an SMX has four instruction schedulers for each arithmetic unit and eight instruction dispatchers. The instruction schedulers in the Kepler architecture are based on a simple implementation concept. While eliminating hardware stages to avoid data hazards in the calculation data path, the compiler predetermines the dependencies of instructions and incorporates them into the instruction information. An SMX has 65,536 32-bit bandwidth registers available to all cores and 64 KB integrated shared memory and L1 cache. The NVIDIA GTX680 has eight SMXs, i.e., it has 1,536 processing cores that perform in parallel.

GPUs have various types of memory, device memory with large capacity and large access latency, shared memory with small capacity and small access latency, and texture memory for accelerating the access of high spatial locality by mapping a texture region in the device memory.

III. Multipattern string matching algorithm

Here, string matching algorithms that search for a specific string from text is discussed. There are two types of string matching algorithms: string matching using a single pattern and string matching using a pattern set that consists of multiple patterns. Knuth–Morris–Pratt (KMP) and Boyer–Moore are well-known string matching methods that use single patterns. Rabin–Karp and Aho–Corasick (AC) are common string matching methods that use pattern sets. Parallel failure-less AC (PFAC) is a typical and efficient method for string matching on a GPU. PFAC is a specialized AC-based string matching algorithm for GPU architectures.

PFAC can perform a search process with high throughput in linear proportion to the size of a text. It is impossible to pre-predict the data size of a stream in a network. PFAC can cope with a wide range of data sizes and achieve high-performance string matching with SoR. Therefore, we conclude that the PFAC algorithm is the most suitable method for implementing GPU-based string matching on SoR-NIDS.

A. Aho–Corasick algorithm

The AC algorithm [9] is an algorithm extended from KMP. AC uses deterministic finite automaton (DFA). In this paper, DFA used in AC is referred to as the AC automaton, and the table that defines the transition of the automaton is called the state transition table. The AC automaton consists of three processing steps. The first step is a transition function that defines the next state when an expected value is input. The second step is a failure function that defines the next state when an unexpected value is input. The third step is an output function that outputs the location where a pattern is matched. Figure 2 shows the AC automaton built from the pattern set {"http," "https," "html," "ssh," "smtp"}.

To avoid complexity, the description of the failure transitions to state 0 is omitted from Figure 1. The state denoted by a double circle indicates that a pattern has been matched and is included in the pattern set. The AC automaton state begins at state 0, receives one character from the first character of text, and continues transitioning until all processing is finished. The termination condition of the process is to reach the end of the searched text. It is indispensable for AC string matching to build an AC automaton as a pre-process. Building an AC automaton process consists of a type of lexicographic pattern set, construction of the state transition table for the transition function, and construction of the state transition table for the failure function. If the type of characters in the pattern set is sufficiently diverse, the calculation cost of these processes is proportional to the number of characters in the pattern set. If the sum of all characters included in the pattern set is M, the calculation cost is $O(M)$. If the length of the input text is m, the search process cost is $O(m)$. Therefore, the calculation cost of the entire AC algorithm processing is $O(M+m)$.

B. Parallel failure-less Aho–Corasick

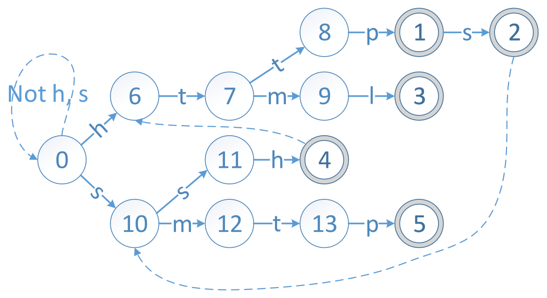


Fig. 1. Example behavior of DFA with five patterns {"http," "https," "html," "ssh," "smtp"}in the AC algorithm

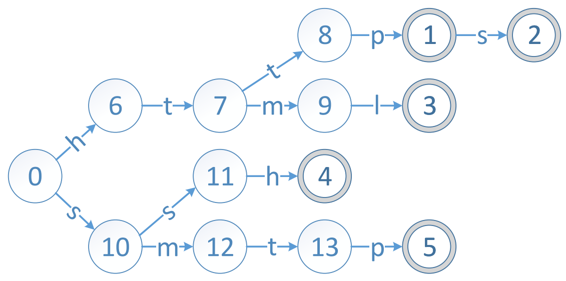


Fig. 2. Example behavior of DFA with five patterns {"http," "https," "html," "ssh," "smtp"}in the PFAC algorithm

The PFAC algorithm [10, 11] has been applied to the AC algorithm in GPU architecture. For parallel string matching, if the input text is divided into some pages and one thread explores one page, there is a boundary problem, i.e., a string that exists across pages is not explored. In the PFAC algorithm, each thread searches a substring starting from the position shown in Figure 2. Each thread continues to search until a miss occurs. Processing finishes when miss occurs. In the PFAC algorithm, a boundary problem does not occur because the text of all positions is processed equivalently. Furthermore, since the search processing is conducted from all text positions, it is unnecessary to guarantee consistency such that the pattern matches after a miss–hit occurs. Therefore, the PFAC algorithm can improve performance by reducing failure transitions in the AC algorithm.

IV. Design of string matching methods using a GPU for multiple streams

In this implementation, we propose a method of string matching using shfl functions based on the PFAC algorithm. We also use a task controller [12]. The task controller monitors the status of threads and multiple stream buffers and issues search processes to threads. Processing using a GPU cannot achieve high throughput if the stream size is very small. The task controller determines whether processing is performed by the GPU or CPU according to the status, such as stream buffer size. For GPU processing, a task controller monitors the available capacity of device memory and issues process only if there is sufficient processing capacity. Here, a task controller is a master thread that issues some threads as slave threads. Communication between the master thread and slave threads is established using a global variable. Processing in a thread consists of checking the parameter, searching, and storing the result. A thread determines whether string matching occurs on the CPU or GPU based on parameters from the task controller. If the CPU is specified, the search process is executed on the CPU with the AC method. If the GPU is specified, the search process is executed on the GPU with the extended string matching method based on PFAC.

A. Proposed method to improve PFAC by shfl functions

We propose an extended string matching method based on PFAC using shfl functions. The shfl functions are extended instructions implemented on the Kepler GPU architecture. In conventional GPU architecture, data accessed by multiple threads must be placed in shared memory. This requires two cycles. The shfl functions make it possible to access local variables in one cycle; variables are on the register of threads in a warp. The functions are executed in parallel for all active threads in a warp. Figure 3 shows an example of the shfl functions. The "__shfl" functions access the register of the thread by addressing the specified index. Each thread obtains a certain value as per the ascending/descending order of threads using "__shfl_down"/"__shfl_up" functions. The "__shfl_xor" instruction exchanges the values of the threads using a butterfly method.

In the PFAC algorithm, an input stream is split and stored in shared memory. Threads fetch the data from the shared memory. In the proposed method, threads obtain the stream using shfl functions in the beginning of the matching process. This is performed from shared memory after several cycles of the matching process. Figure 4 shows the flow for obtaining text using shfl functions. First, a series of data that each thread has in the register is collated. Then, all threads in the warp obtain the character using the "__shfl_up" instruction. All threads perform matching regardless of the failure or success of matching to obtain the character consistently. In contrast to PFAC, failed threads in the matching process become inactive. In an evaluation, efficient processing by shfl functions was investigated by analysis of data traffic and rule sets. In the proposed method, there is a trade-off between the character fetching cycle and efficiency of the thread allocation. The shfl instructions can access faster than shared memory. However, if a thread results in matching failure, the thread becomes inactive, and this thread cannot be reused while shfl is used. The matching process will be efficient if the number of failing threads is large and these threads are reused. This trade-off is described further in Section 5. As a result, matching process threads obtain the stream using shfl functions from the first matching process to the fourth matching process and obtain the stream from shared memory from the fifth matching process.

Slave threads transfer a stream to a device, perform a search process using PFAC, and transfer the result to the host. Streams are stored in both a register and shared memory. In a search process, GPU threads load stream data from the register and shared memory, and use the PFAC automaton in texture memory. If the thread finds an expected pattern, the result is stored in shared memory. Then, if there is no valid transition in a GPU thread, the automaton process is finished and all

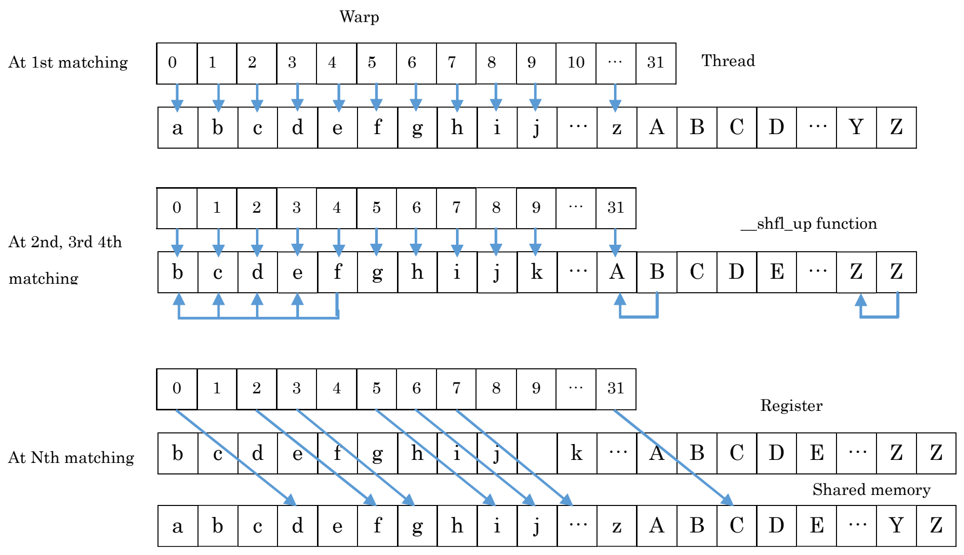


Fig. 4. Flow of obtaining text using shfl functions

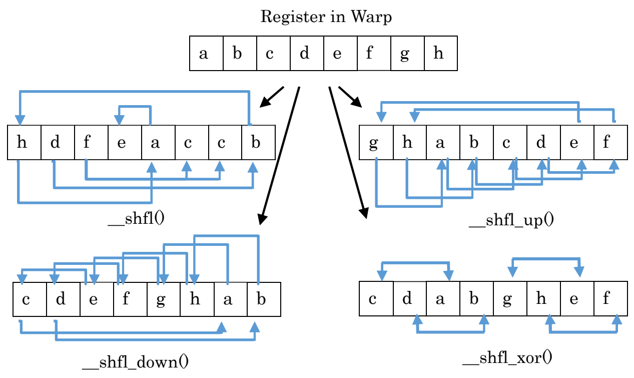


Fig. 3. Example of shfl functions

results are stored in device memory from shared memory. If all GPU threads finish the matching process, the process in the kernel is finished. Then, the results are transferred to host memory. Finally, slave threads provide the results as a global variable to the task controller and all processes of slave threads are terminated.

B. Proposed memory implementation

To perform a string matching process using a GPU, it is necessary to transfer a stream to be searched to device memory. Main memory and device memory have different address spaces; thus, data is transferred using the cudaMemcpy memory transfer API. Since host and device are connected by a PCI interface, the transfer of all processes is performed through the PCI interface. Figure 5(1) shows the flow for transferring data using cudaMemcpy. First, when the host requests that data be transferred from main memory to device memory, data is written back to main memory to guarantee data coherence in main memory. Next, if data coherence is guaranteed, the data is copied to the kernel and the PCI buffer. Then, the data is copied to device memory as a packet through the PCI interface.

In this study, we propose a method to improve transfer delay of stream data by skipping these steps. Recently, the cost of memory has decreased. Thus, it is easy to reserve abundant memory resources for a general-purpose device. We use main memory spaces as physical memory and prohibit swapping to disk by the OS, as shown in Figure 5(2). Therefore, it is possible to skip the write-back of data that is swapped on the disk. This method for using memory space as physical memory is referred to as pinned memory implementation.

In addition, the host does not load from or store to the memory space of the stream while processing occurs on the GPU. Therefore, we propose a method that uses the write-through cache line on the CPU. Thus, it is possible to always maintain data coherence in main memory. This means that it is unnecessary to write back CPU cache when transferring data. This host memory method is called the write-combined (WC) memory implementation. Figure 5(3) shows a summary of data transfer using the WC memory implementation. WC memory implementation improves the data transfer delay in the PCI interface. However, there is a disadvantage; the throughput for loading data from main memory by the host is slower than that of the malloc API.

Device memory and host memory address spaces can be managed as an integrated memory space. Therefore, by determining the data dependencies in the kernel at the time of compiling, data transfer and kernel execution can be processed as a single execution unit. Therefore, a programmer does not need to specify the timing of the transfer. This method is called the mapped memory implementation. Figure 5(4) shows a summary of data transfer using mapped memory implementation.

C. Implementation

The string matching application proposed in this study is expected to perform on a single machine built with a general-purpose GPU. We used an ASUS RAMPAGE 4 EXTREME motherboard with one Intel Core-i7 3930K quad-core processor and 64 GB of DDR3 memory (1,600 MHz operation frequency). The GPU is an NVIDIA GTX680 that has 8 units of 192 SMX cores (1,006 MHz operation frequency) and 2,048 MB of DDR5 memory (3,004 MHz operation frequency). The host and the device were connected through a PCI Express 2.0 ×16 interface. The operating system used was Ubuntu 13.04 with Linux kernel version 3.8.0-35. The program was written in C/C++. The program was compiled with the gcc 4.6.4/g++ 4.7.3 [13] and nvcc 5.0 compilers.

We used two types of rule sets, as shown in Table 1, i.e., the Snort set and the HTTP set. The Snort set [15] consists of 23,139 patterns. The average length of the patterns is 29.0 bytes. We also used the HTTP set, which is generated by selecting the HTTP rules from the Snort set. The HTTP set consists of 2,089 patterns. The average length of the patterns is 13.3 bytes.

We used two types of traffic data, as shown in Table 1, i.e., real traffic captured in the Nishi laboratory and an artificial trace in which a match occurs frequently. A normal trace was captured at the Nishi laboratory by tcpdump [14] of the 1000BASE-T Internet gateway port. This trace was captured on May 27, 2009, and we extracted the HTTP stream, which specifies port number 80 as the destination or source port. The dump file was 14 GB. The extracted L7 information was 11 GB, and the number of streams was 281,479. We created an artificial trace for evaluation in which matching of strings occurs frequently. The artificial trace only consists of patterns squeezed from the Snort set. We considered that the processing throughput becomes low using the artificial trace because the storing process increases with the frequent pattern matches. We used this artificial trace to evaluate the worst performance of SoR-NIDS.

TABLE 1 RULE SETS AND TRAFFIC ENVIRONMENT

	Name	Details
Rule set	Snort set	The rule sets from Snort set
	HTTP set	The rule sets targeting HTTP traffic that selected from Snort set
Traffic	Normal trace	Captured trace at Nishi laboratory
	Artificial trace	Artificial trace that a match occurs frequently

V. Evaluation

Here, we evaluate the method for improving the PFAC algorithm using shfl functions. The failure ratio of GPU threads is considered to affect the performance in the method for obtaining a stream by shfl functions. Here, the active ratio of GPU threads is the ratio of the GPU threads that continue processing to the GPU thread that terminates by a false match. This active ratio is equal to $1 - \text{failure ratio}$. The active ratio of GPU threads is determined by an input stream and pattern sets. If the prefixes of patterns are likely to match characters of a stream, the active ratio of the GPU thread is high. If the active ratio of GPU threads in warp is high, the number of accesses to inactive threads is reduced. However, using shfl functions to access inactive threads is a disadvantage. Therefore, if the number of accesses to inactive threads is small, the string matching process can be performed effectively.

First, we evaluated the active ratio of GPU threads in the

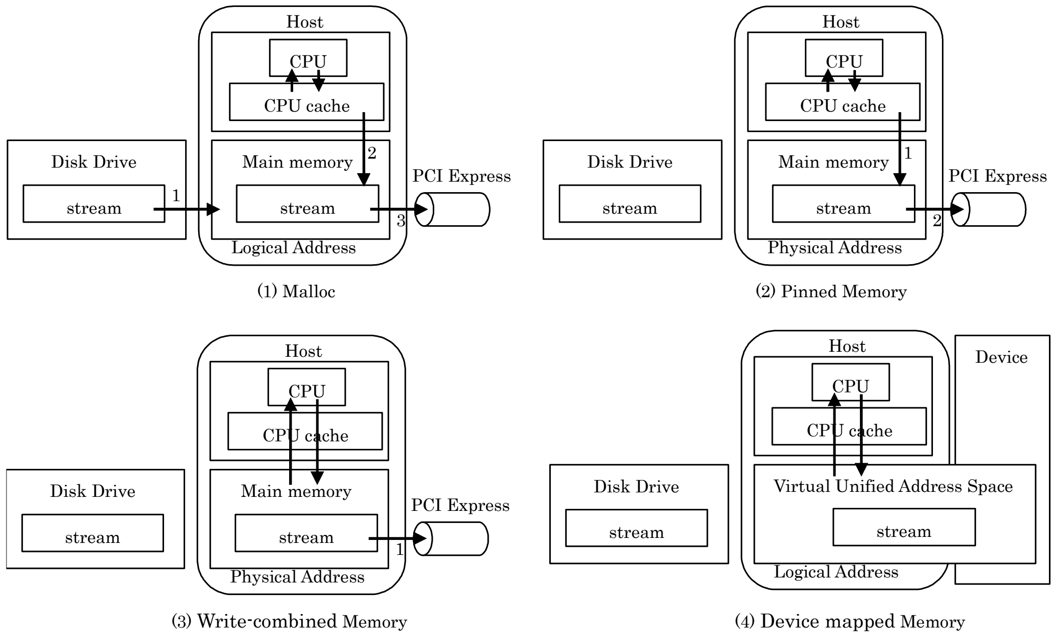


Fig. 5. Flow of data transfer by proposed memory implementation methods

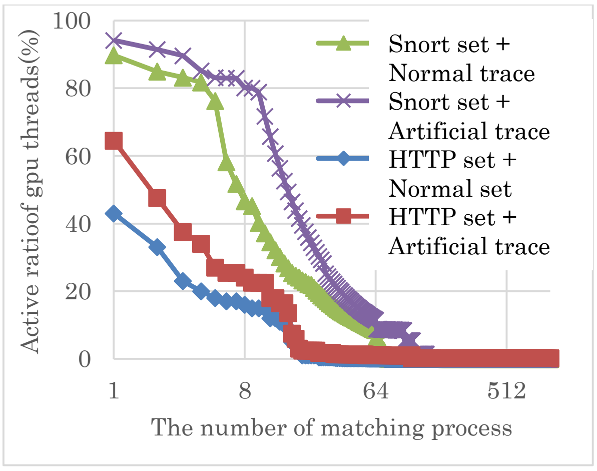


Fig. 6. Active ratio of GPU threads

proposed condition, as shown in Figure 6. When the number of times matching occurs is large, the active ratio decreases gradually. In Figure 6, the active ratio is comparatively high from approximately the beginning of the fourth matching process. Therefore, we estimate that the best time to shift from shfl operations to shared memory operations is after the fourth matching process. Table 2 shows the active ratio of GPU threads at fourth matching.

In this condition, we evaluated the difference using malloc, mapped memory, pinned memory, and WC memory implementations. Figure 7 shows the throughput of search processing using the HTTP set and the normal trace. The throughput using shfl function is slightly lower than that without shfl functions. According to the evaluation of active ratio of GPU threads, the active ratio of GPU threads using the HTTP set and the normal trace was always 43% or lower. Therefore, data fetching by inactive threads demonstrated reduced throughput.

Figure 8 shows the throughput of search processing using the Snort set and the normal trace. Overall, the throughput was improved. In this case, the throughput obtained using shfl functions was higher than that obtained without shfl functions. When the stream size was small, the performance improvement was significant. In this case, throughput increased by 11.1% on average. In the mapped memory implementation, the improvement was the highest and throughput increased by 12.0%. If the stream size is increased, performance improvement decreases. For the 128 MB stream, the throughput increased by 2.1%. It is considered that the shfl functions improved throughput because the active ratio of GPU threads was high at the initial search processing when using the Snort set.

Figure 9 shows the throughput of search processing using the HTTP set and the artificial trace. When the stream size

TABLE 2 ACTIVE RATIO OF GPU THREADS AT THE FOURTH MATCHING

	Normal trace	Artificial trace
Snort set	81.7%	85.2%
HTTP set	20.0%	34.0%

was small, the throughput decreased slightly. When the stream size increased, the performance gap narrowed. The throughput decreased by 1.4% to 6.5% when the stream size was smaller than 8 MB. When the stream size was 8 MB or greater, throughput using shfl functions was approximately equal to the throughput without shfl functions. Otherwise, the active ratio of GPU threads becomes low using the HTTP set. In addition, throughput decreased significantly when the stream size was small.

Figure 10 shows the throughput of search processing using the Snort set and the artificial trace. In this combination of rule sets and traffic, the active ratio of GPU threads was the highest among all combinations, and the processing throughput was low because many matches occurred. As a result, the throughput using the shfl functions was higher than that without the shfl functions. Note that the improvement of throughput does not depend significantly on stream size. The throughput increased by 3.2% to 11.4% overall. The performance improvement in the WC memory implementation showed the highest improvement rate (9.1% for all stream sizes on average). Therefore, it was expected that the throughput using shfl functions would be higher than that without shfl functions when the active ratio of GPU threads was high. However, throughput was reduced when the size of the rule sets was small and the active ratio of GPU threads was low.

VI. Conclusion

In this study, we have proposed a high throughput string matching method using a general-purpose GPU for NIDS on SoR. We proposed and evaluated a fast string matching method using shfl functions and several memory implementation methods. In this implementation, we used and improved a string matching method based on the PFAC algorithm to use shfl functions to improve the performance of a string matching process. The performance was largely dependent on the combination of traffic and rule sets. While the performance with shfl functions was lower than that without shfl functions when the active ratio of GPU threads was low, the performance with shfl functions was 12.0% higher than that without shfl functions when the active ratio of GPU thread was high. Thus, we have confirmed the effectiveness of the proposed string matching method using shfl functions.

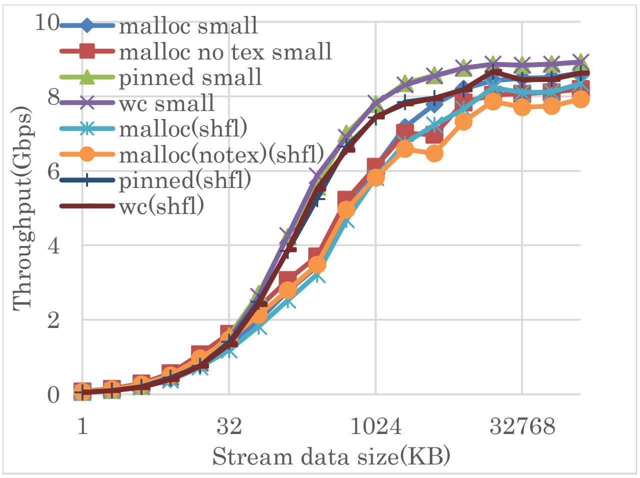


Fig. 7. Throughput of search processing using the HTTP sets and the normal trace

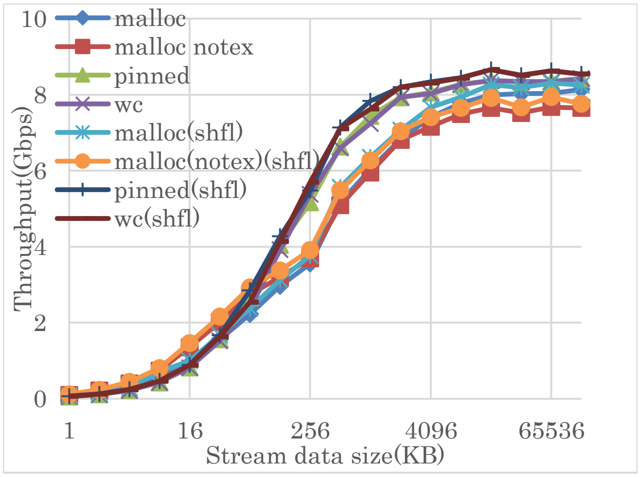


Fig. 8. Throughput of search processing using the Snort sets and the normal trace

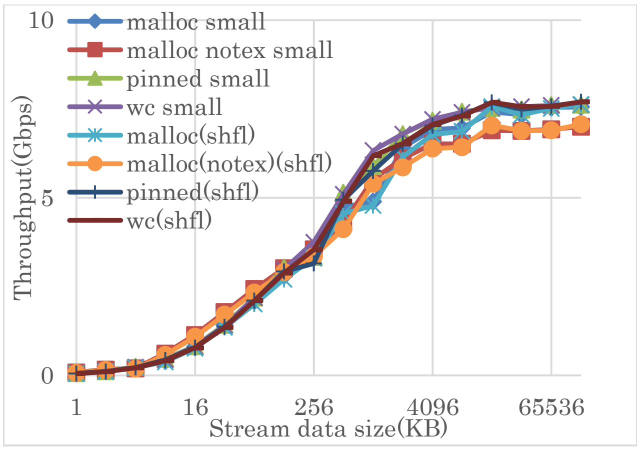


Fig. 9. Throughput of search processing using the HTTP set and the artificial trace

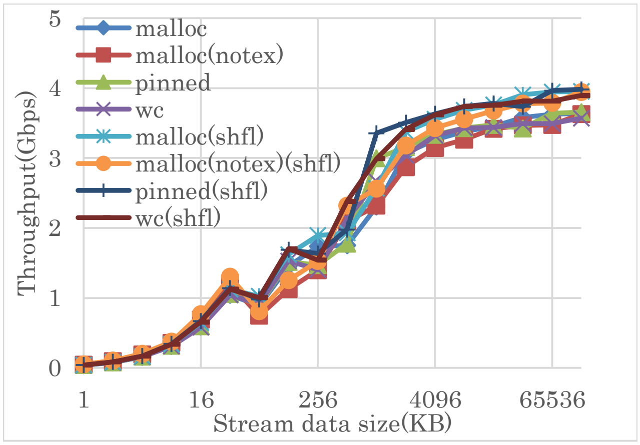


Fig. 10. Throughput of search processing using the Snort sets and the artificial trace

REFERENCES

[1] Paolieri, M.; Bonesana, I.; Santambrogio, M. D., “ReCPU: A parallel and pipelined architecture for regular expression matching,” Very Large Scale Integration, 2007. VLSI-SoC 2007. IFIP International Conference on, pp. 19–24, IEEE, Oct. 2007.

[2] Sidhu, R. and Prasanna, V. K., “Fast regular expression matching using FPGAs,” in IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM01), pp. 227–38, April 2001.

[3] NVIDIA. http://www.nvidia.com/page/home.html

[4] Juniper Networks. http://www.juniper.net.

[5] Maxeler Technologies. http://www.maxeler.com/

[6] Mahdinia, P.; Berenjkoob, M.; Vatankhah, H., “Attack signature matching using graphics processors in high-performance intrusion detection systems,” Electrical Engineering (ICEE), 2013 21st Iranian Conference on, pp 1–7, IEEE, May 2013.

[7] Jiang, H.; Zhang, G.; Xie, G.; Salamatian, K.; Mathy, L., “Scalable high-performance parallel design for network intrusion detection systems on many-core processors,” Architectures for Networking and Communications Systems (ANCS), 2013 ACM/IEEE Symposium on, pp. 137–146, IEEE, May 2013.

[8] NVIDIA Kepler GK110 http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf

[9] Aho, A. V. and Margaret, J. C., “Efficient string matching: an aid to bibliographic search.” Commun. ACM, Vol. 18, No. 6, pp. 333–340, June 1975.

[10] Lin, C. H.; Tsai, S. Y.; Liu, C. H.; Chang, S. C.; Shyu, J. M., “Accelerating string matching using multi-threaded algorithm on GPU,” Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE, pp.1–5, IEEE, Dec. 2010.

[11] Zha, X.; Sahni, S., “Multipattern string matching on a GPU,” Computers and Communications (ISCC), 2011 IEEE Symposium on, pp. 277–282, IEEE, July 2011.

[12] Ikeuchi, K.; Wijekoon, J.; Ishida, S.; Nishi, H., “GPU-based multi-stream analyzer on application layer for service-oriented router,” Embedded Multicore SoCs (MCSoC), 2013 IEEE 7th International Symposium on, pp.171–176, IEEE, Sept. 2013.

[13] The Gnu Compiler Collection. http://gcc.gnu.org/

[14] TCPDUMP/LIBPCAP public repository. http://www.tcpdump.org/

[15] Snort.http://www.snort.org

Study of Dynamically-Allocated Multi-Queue Buffers for NoC Routers

Yung-Chou Tsai
Department of Electrical Engineering
National Tsing Hua University
Hsinchu, Taiwan
d923935@oz.nthu.edu.tw

Yarsun Hsu
Department of Electrical Engineering
National Tsing Hua University
Hsinchu, Taiwan
yshsu@ee.nthu.edu.tw

***Abstract*—A large portion of area and power in Network-on-Chip (NoC) routers is consumed by buffers, and hence these costly storage resources must be utilized well. However, some early related literatures are not suitable for modern NoC router architecture as well as various complicated traffic loads anymore. In this work, we refine the dynamically-allocated multi-queue (DAMQ) buffer organization and propose a new one that can accommodate multiple packets more than the number of virtual channels, named DAMQ with multiple packets (DAMQ-MP). The DAMQ-MP scheme can solve certain data transmission issues under some circumstances, such as heavy network congestion or short packets, to improve performance. We also introduced two methods applicable to DAMQ-based buffers, which are adding priorities for switch allocation and reserving virtual channels for high-priority packets. Experimental results show that DAMQ-MP routers can have up to 24.52% higher saturated throughput than SAMQ and DAMQ counterparts.**

Keywords-buffer; virtual channel; router; Network-on-Chip;

I. INTRODUCTION

Buffers are usually added in NoC router nodes to provide temporal storage space for arriving data to wait for resources to traverse these router nodes. In most cases, statically-allocated multi-queue (SAMQ) buffers in input-queued switches are adopted due to their simple flow control mechanism and small hardware overhead. However, each queue of SAMQ buffers contains a fixed amount of its own buffer spaces and never shares them with other queues. This kind of static allocation on input buffers results in storage resource wastage for lack of flexibility on buffer management. For the same input buffer, it is possible that some queues are overloaded in need of more buffer spaces while some queues are nearly empty with plenty of buffer spaces unused. Especially buffers consume a great portion of power and area in a NoC router [1], and thus these costly buffer resources must be effectively utilized when designing router architecture.

Tamir and Gregory [2] first proposed a novel buffer organization named DAMQ to provide better flexibility of buffer utilization. This DAMQ buffer organization can dynamically partition buffer storage with linked lists to handle variable length packets for achieving higher performance. Another approach using self-compacting buffers (SCB) to implement DAMQ switches was introduced in [3] to reduce the hardware overhead and complexity. Liu et al. [4] improve the DAMQ switches adopting SCB by letting two sets of virtual channels in different dimensions share buffer space. A special architecture called high-performance input-queued switch (HIPIQS) also uses a DAMQ organization, pipelined access to multi-bank input buffers with chucks, and many small additional cross-point buffers, to deliver high performance [5]. However, current NoC architecture designs prevalently use wormhole switching to relax the constraints on buffer size, virtual channel flow control to avoid deadlocks and head-of-line (HoL) blockings, and mesh-based topology to fit chip layouts. The above mentioned searches are not suited to the requirements of modern NoCs anymore and therefore have to be resurveyed.

Rezazad et al. [6] showed that the optimal number of virtual channels and buffer length of mesh-based interconnection networks highly depends on the traffic pattern as follows. Under light traffic, the buffer structure extends virtual channel depth for continual transfers to improve latencies; under heavy traffic, the buffer structure dispenses many virtual channels for congestion avoidance to increase throughput. Based on these concepts, two buffer structures of dynamically changing their number of virtual channels were proposed [7][8]. However, these two buffer structures have to put a lot of effort to manage the varying number of virtual channels, such as tracking tables, and that makes them hard to scale. In addition, arbiters used in virtual channel allocations must be simplified to prevent the hardware overhead from dramatically increasing as the number of virtual channels goes up.

In this paper, we propose the DAMQ-MP scheme by allowing multiple packets that are more than the number of virtual channels coexisting inside a DAMQ-based input buffer. Because this scheme breaks the limitation of one packet per virtual channel in conventional virtual channel routers, it can overcome the possible issues resulting from waiting for switching packets, handling short packets, and HoL blockings. We believe that DAMQ-MP is the easiest and feasible method to improve network performance and buffer utilization without sophisticated hardware design modifications. Besides, it is suitable to some applications like reserving virtual channels for high-priority packets.

II. MOTIVATION

A. Packet Switching Latencies

In conventional SAMQ and DAMQ buffers, one virtual channel typically only holds one packet each time for easy control and preventing HOL blockings. Therefore, if all virtual channels contain packets whose tail flits have entered but not left yet, the remaining free buffer resources are wasted until some packet is gone and another new packet arrives, as illustrated in Figure 1a. Even though the DAMQ mechanism lets the remaining buffer resources be used by packets that have not completed their delivery as shown in Figure 1b, bringing more flits of the these long packets merely helps little to deal with the following blanks after any other packet leaves. The duration between the two departures of the tail flit belonging to the current packet and the head flit belonging to the next packet is pretty long. This duration includes the time of handling the following procedures: notifying upstream router that this virtual channel has become free, allocating a new packet to this virtual channel, delivering the flits through the switch and physical link into this virtual channel, doing routing computation and output virtual channel allocation of its downstream router, and possibly incurring stalls due to contention. This kind of "packet-switching" latency can't be neglected because data transmission within the related virtual channel stops and unable to provide any contribution to throughput during this period of time.

Usually the packet-switching latencies can be hidden by using multiple virtual channels: some virtual channels may still work well while some virtual channels are reloading their new packets. However, if the interconnection network is seriously congested and most of virtual channels in routers are halted by contention stalls, any loss of virtual channels caused by switching packets will make the situation worse and the influence of these packet-switching latencies becomes more apparent. The way solving this issue is to find out how to shorten the packet-switching latencies, and thus one straightforward method is to bring the next incoming packet into the input buffer in advance without waiting for a virtual channel released to be free and standing by. This helps halted virtual channels returning back to work normally as soon as possible so that more packets can be provided to choose to traverse the switch fabric.

B. Traffic Loads with Short Packets

Especially in some scenarios, interconnection networks carry traffic loads mixing long data packets with short control packets (e.g., commands, acknowledgments, etc.). Each short packet still occupies one virtual channel but brings only few flits to make the physical channel and unused buffers idle in most of the time. Short packets also make virtual channels switching more frequently than long ones, and have more chances of letting virtual channels halted due to waiting for the arrivals of new packets. When encountering some blocking, each blocked short packet will stay in its virtual channel and thus be distributed into one of many different routers. However, this situation can be improved by allowing several short packets which stop forwarding to be compacted in a few routers as long as there are enough buffer spaces available in these routers. Having more packets contained in the input buffers not only leads to higher buffer utilization rate but also releases more available virtual channel entities for transmitting more packets in the whole interconnection network.

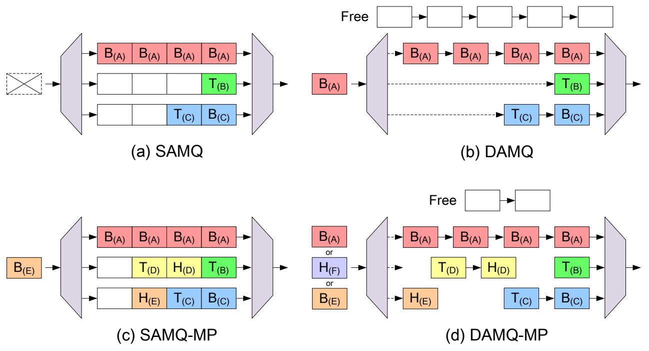


Figure 1. Comparison of buffer organizations. Each flit is labeled with its type (H, B, and T standing for head flit, body flit, and tail flit respectively) and the packet it belongs to (i.e., the smaller letter within the parentheses).

C. Dynamic Input Buffer Management

Based on the reasons mentioned above, to make an input buffer accommodate multiple packets whose amount is more than its number of virtual channels can take some advantages. However, in order to keep more associated packet information due to the increasing number of packets, extra hardware such as registers used for state fields as well as pointers and control logics must be added. If applying this method to the SAMQ organization and then becoming the SAMQ with multiple packets (SAMQ-MP) organization as shown in Figure 1c, it is obviously unsuitable and impractical. First of all, letting the same virtual channel commonly shared by multiple packets will result in HOL blocking problems. Although moving the blocked packet from its current virtual channel to another free virtual channel can overcome the HOL blocking, but the price on the extra hardware cost and complexity is too expensive. In addition, like the buffer allocation in SAMQ buffers is partitioned statically, these extra registers used for state fields and pointers waste along with reserved buffer resources if their associated packets belonging to the dedicated virtual channels are absent.

On the contrary, the DAMQ organization is more suitable to this scheme due to its linked-list buffer structure, as shown in Figure 1d. In the original DAMQ buffer, each set of state fields and pointers is dedicated to one virtual channel. However, if the number of packets in a buffer is no more limited by the number of virtual channels but by a quota of packets, an input buffer must reserve one set of state fields and pointers for each packet, not for each virtual channel. Because every packet is stored as a linked list and buffer spaces as well as virtual channels are used among packets, even if the number of packets in use is far less than the reserved quota of packets in a buffer at all, there are no buffer resources waste for unused packets except the associated sets of packet state fields and pointers.

III. THE DAMQ-MP SCHEME

A. *Overview of DAMQ Buffers*

A DAMQ buffer is structured as linked lists for dynamically adjusting the depths of virtual channels for more efficiently utilizing the input buffer. In this way, the precious memory resources are shared by all virtual channels in the same input unit, and busy virtual channels can get more buffer spaces than idle ones. An evident difference between the SAMQ and DAMQ organization is the mechanism of credit management. The DAMQ router must collect all credit information and then put it together in the virtual channel allocator to make centralized credit management with several extra counters. In practice, some restrictions also need to apply to the DAMQ buffer allocation policies for avoiding deadlock and load imbalance.

B. *DAMQ-MP Router Architecture*

In DAMQ-MP buffers, whenever the tail flit of a packet enters the input buffer via one of the virtual channel entries connecting with the physical link, this packet will never use this virtual channel entry from now on and certainly can tear the corresponding allocation relationship by alerting the notification signal to its upstream router. Therefore, a new packet from its upstream router can be sent out and received via this free virtual channel entry. Then the newly arriving packet begins its routing computation, and all following flits belonging to this packet are linked together. After the result of routing computation comes out and is stored into its corresponding state field, this packet will occupy one of the unused virtual channel exits connecting with the switch fabric to request output virtual channel and switch traversal bandwidth for departure.

Here we use the terms of “entry” and “exit” to distinguish the virtual channels for receiving packets from the virtual channels for sending packets. These virtual channel entries and exits are respectively only responsible to the usage rights of the physical link and the switch fabric connecting with the input buffer. In the original DAMQ organization, one packet enters, occupies, and leaves its virtual channel, and the entry and the exit it uses both belong to the identical virtual channel it occupies; nevertheless, in the DAMQ-MP organization, an entry or an exit of the virtual channels is just an access gateway to write or read flits, and neither entries nor exits have to coexist in pairs. It is possible that a packet (like Packet D in Figure 2a), which has brought all its flits inside the DAMQ-MP buffer through an entry, merely stays alone and waits for an exit to depart.

As illustrated in Figure 2b, DAMQ-MP buffers just like DAMQ buffers need several pairs of head and tail pointers, one extra free-list pointer, and counters for all linked lists. Except for the original state fields used for virtual channel exits to store their virtual channel status and assigned output virtual channel number, additional state fields for all packets are also needed to keep the routing computation results and some other associated attributes, for instance, the allocation priority of the packet. Essentially every packet in a DAMQ-MP buffer exists individually and must possess a unique packet ID number. These packet ID numbers are uniformly assigned and managed by the centralized virtual channel allocator, and they are always carried with the head flits of their packets to their next routers for use of recognition while manipulating the input buffers, just like the virtual channel ID number carried with every flit.

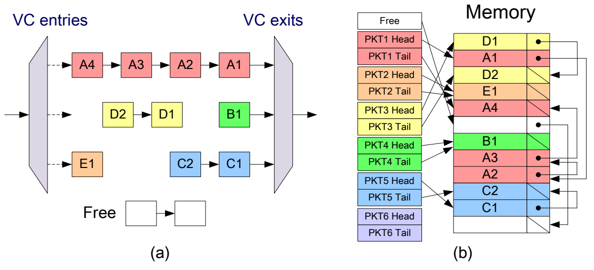


Figure 2. DAMQ-MP buffers and linked-list memory space.

In order to keep track of which packets are currently using which access gateways to move in and out of the input buffer, each virtual channel entry and exit has to record the packet ID number of the packet which resides in it. These above packet ID numbers are stored in two arrays of registers named as “vc_entry” and “vc_exit” respectively. Meanwhile, another array of registers named as “vc_list” is used to record the arriving order of the packets which are waiting for free virtual channel exits, and these registers assist the input unit to decide the allocation order of the virtual channel exits. There are two copies of these three register arrays for managing one input buffer: one is certainly built inside the input buffer itself for buffer storage handling, and another is located in the virtual channel allocator of its upstream router for buffer storage allocation. The information of these three array registers (i.e., packet ID numbers) are continuously updated to keep their consistency according to the transitions of the “vc_free” and “pkt_left” signals as well as the head flits of packets. The “vc_free” signals are exactly identical to the same signals in the DAMQ router to notify its upstream router that there is a free virtual channel ready for being used again, but now they only refer to the occupation of the virtual channel entries in a DAMQ-MP input buffer. On the other hand, the “pkt_left” signals are similar to the “vc_free” signals for indicating the departures of packets and merely respond to the availability of the virtual channel exits by giving notifications to the upstream router. The information about mapping a packet to a virtual channel entry is carried with the head flit of a packet and passed to the input unit of its downstream router after transmitting this head flit.

While comparing the proposed DAMQ-MP router with the original DAMQ router, they almost have identical hardware components and behavior in appearance, but there are two huge differences inside the input units and the virtual channel allocator. The first different part is relative to the novel “packet number system”. Every packet existing in an input buffer must have a unique ID number to distinguish itself from others. Handling any proceedings about packets will need to use these packet ID numbers. All packet ID numbers belonging to the same input buffer are centrally managed and distributed by the virtual channel allocator of its upstream router. While a packet in a virtual channel exit requests an output virtual channel to the input buffer of its downstream router, it will also need to be assigned an available packet ID number for the input buffer. Certainly this assigned packet ID number will be used until the associated packet has completely left the input buffer, and then the input unit must notify the virtual channel allocator of its upstream router via the "pkt_left" signal that this packet ID number is no longer used and available again.

The second different part is that the DAMQ-MP router induces the concept of "decoupling the virtual channel entries and exits". Although the same number of virtual channel entries and exits in a DAMQ or DAMQ-MP router, a packet in the DAMQ router passes through the input buffer via an entry and an exit belonging to the identical virtual channel, whereas any pair of a virtual channel entry and a virtual channel exit in the DAMQ-MP input buffer are irrelative and handle packets independently. To manage all of these virtual channel entries and exits in DAMQ-MP buffers relies on the above-mentioned three array registers (i.e., "vc_entry", "vc_exit", and "vc_line") as well as two notification signals (i.e., "vc_free" and "pkt_left").

C. *Characteristics*

Because of the intrinsic characteristic of the linked-list data structure, a DAMQ router can more easily enhanced to bring in more packets inside than a SAMQ router. As mentioned previously, one major cause of improving performance is that the DAMQ-MP mechanism reduces the idle time of reloading a new packet into the halted virtual channel exit which the contained packet inside just leaves. Another cause is that the DAMQ-MP buffers let all virtual channel entries keep receiving packets all the time if there are enough packet sources coming from the upstream router, especially for short packets. Therefore, the DAMQ-MP organization can be beneficial for the cases of short packets, large buffer capacity (relative to packet lengths), heavy traffic congestion, and small number of virtual channels.

Because the DAMQ-MP scheme handles buffer resource allocation in units of packets instead of virtual channels, the increased hardware costs are just several registers for pointers, counters, state fields, and virtual channel mapping arrays, as well as the extra signals and control logics for packet system management and virtual channel decoupling. Owing to the additional mapping transformations from virtual channels to packets and vice versa, the data access delay may be prolonged to make the performance degraded. Certainly, we can add more sophisticated hardware such as register pre-fetch units to avoid or compensate the loss on performance, and they are trade-offs while designing DAMQ-MP routers.

Another possible problem is that too many packets residing in a crowded but small DAMQ-MP buffer will compress the available spaces that a packet possibly can get. If lots of flits belonging to one packet have already existed in the input buffer, bringing one more remaining flit of this packet still can't make it forward further until all of its preceding flits have left. Therefore, one simple method is adding priorities to the allocating switch process to always first bring a flit that is the most likely to depart the input buffer soon. The priority levels can be estimated by counting the minimum amount of flits which are in front of the candidate flit and leave the input buffer possibly before it. Then the switch allocator sets priorities of all requesting packets based on the counting results. Imposing the priorities on the switch allocation can effectively solve the load imbalance problem occurring inside a DAMQ-based input buffer and make the data transmission ceaselessly.

D. *Case Discussion: Cut-in-Line for High-Priority Packets*

In most cases, the packets encapsulating control information are shorter than the packets encapsulating raw data in packet length. In addition, these control packets are usually much more important than raw-data packets to a certain extent and demand faster transmission by possessing higher priority level. In the case of all packets sharing the same network fabric, the simplest method to prevent high-priority packets from being blocked by other packets is reserving some network resources, for instance at least one virtual channel, for high-priority packets to establish fast transferring routes all the time.

The proposed DAMQ-MP scheme is especially suitable for such reservation method due to its linked-list data structure and short lengths of high-priority packets. Because the occurrence probability and the amount of high-priority packets are both in very small proportions, the high-priority dedicated virtual channels can keep the least minimum number of reserved flit buffers while idle at most of time and can be dynamically inserted more available flit buffers while they are in use. Furthermore, if a DAMQ-MP input buffer has more than one high-priority packets inside, switching process for high-priority packets in the virtual exits can immediately accomplish without letting any virtual exits idle.

Because many packets appear at the same time in the DAMQ-MP input buffer, one conceptual "cut-in-line" behavior possibly happens to shorten the waiting time to the high-priority packets for the virtual channel exits. As shown in Figure 3, when a virtual channel exit is freed up (by Packet B), the control logics definitely have to choose the foremost one of the high-priority packets (Packet E) to take that virtual channel exit, whether any low-priority packets

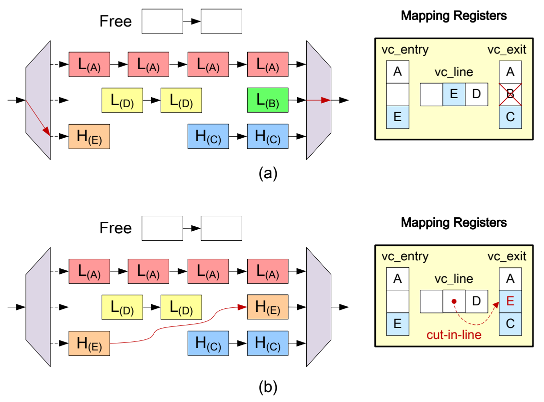
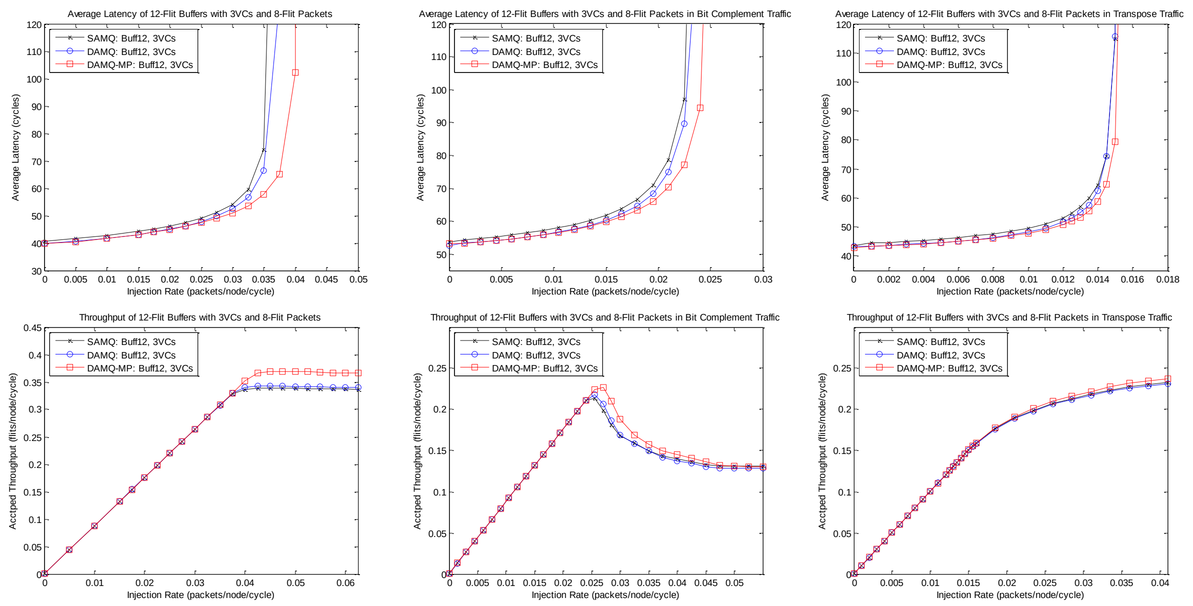


Figure 3. Examples of the "cut-in-line" behavior. Each flit is labeled with its priority (H and L standing for high-priority and low-priority respectively) and the packet it belongs to (i.e., the smaller letter within the parentheses).



(a) Uniform Random (b) Bit Complement (c) Transpose

Figure 4. Performance of routers with 12-flit buffers and carrying traffic loads of 8-flit packets under different traffic patterns.

(Packet D) are in front of that chosen high-priority packet or not. This kind of "cut-in-line" behavior can be easily made by inserting a newly coming high-priority packet into the place after all present high-priority packets as well as before all other packets in the waiting line "vc_line", rather than directly adding it to the place right behind the last packet in the waiting line.

IV. EXPERIMENTAL RESULTS

We build a cycle-accurate flit-level simulator in SystemC to carry out all experiments. The interconnection network we simulate is an 8 x 8 mesh topology adopting 4-stage router pipeline, X-Y routing and wormhole switching flow control. We set the warm-up phase of 10,000 cycles to wait the network into its steady state, and then start to sample data during the measurement phase of 100,000 cycles. Unless otherwise specified, all buffer schemes are tested under the synthetic uniformly distributed random traffic pattern and the maximum number of packets in a DAMQ-MP buffer is unlimited. The latency of a packet is the time interval measured from the time the head flit of the packet is generated by the traffic generator of a source node to the time the last flit of the packet leaves the network. The throughput is the average accepted traffic amount by a destination node per cycle.

A. Traffic Pattern

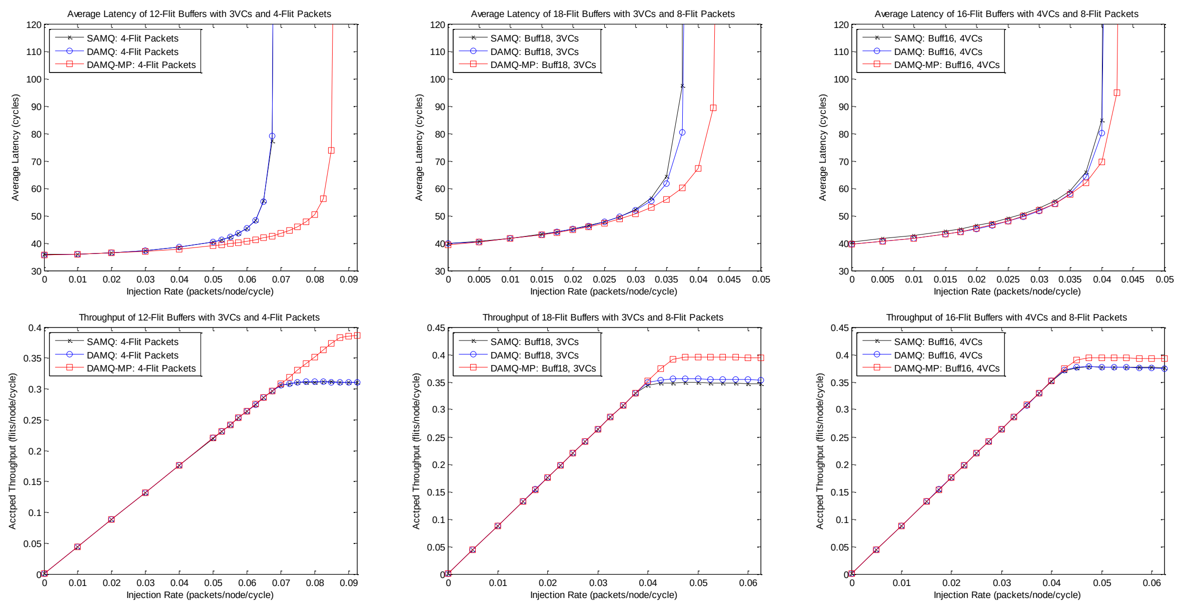
Figure 4 shows the performance results of routers with buffer capacity of 12 flits and 3 virtual channels conveying 8-flit packets under different traffic patterns. No matter what kind of traffic pattern is, the DAMQ-MP organization always has the best performance among all three buffer structures. Especially when the network starts to get saturated, the dynamic buffer allocation and reduction of packet switching latencies in DAMQ-MP organization makes the whole network accommodate more flits and reach a higher throughput value. Owing to the balanced loads in uniform random traffic distribution, the saturated throughput of the DAMQ-MP routers is steadily about 8.53% higher than the other two. Even under the bit complement and transpose traffic patterns, the peak throughput improvements are still 9.56% and 1.56% respectively.

B. Packet Length and Buffer Structure

Compared with the result under random traffic in Figure 4a, we make a series of experiments of changing the ratio of packet length to buffer capacity that determines the expected amount of packets possibly residing in a buffer. As the result shown in Figure 5a, when packet length is relatively small to buffer capacity, the DAMQ-MP routers can hold as many short packets as possible but the SAMQ and DAMQ ones can't due to the limitation of one packet per virtual channel. The saturated throughputs of the DAMQ-MP routers are 24.52% higher than the other two for 4-flit packets.

Then if the buffer size is increased to 18 flits with the same amount of virtual channels, the performance improvement between DAMQ-MP routers and the other two expands as the input buffer capacity grows. As shown in Figure 5b, the improvement for the DAMQ-MP routers relative to the DAMQ ones in the saturated throughput rises to 11.24%. This proves that providing more sufficient buffer spaces for DAMQ-MP routers to accommodate more packets at the same time can make them achieve better performance.

If the buffers are kept to have the same average number of 4 flits per virtual channel, the difference of performance improvement between DAMQ-MP and the others shrinks as adding more virtual channels. As shown in Figure 5c, the improvement of saturated throughput drops to 4.79%. The mechanism of multiple packets in the DAMQ-MP routers facilitates the input unit to fast reload new packets to any idle



(a) 4-Flit Packets (b) 18-Flit Buffer Size (c) 4 Virtual Channels

Figure 5. Performance of routers with short packets, large buffer size, and more virtual cahnnels.

virtual channel exits and keep as many virtual channels unceasingly running as possible. However, if there are already many virtual channels within a buffer, these efforts that the DAMQ-MP scheme does become less apparent relative to the DAMQ organization.

C. *Maximum Number of Packets*

Intuitively, setting the parameter "maximum number of packets" must depend on the buffer capacity and the length of packets. Hence, we choose a large 32-flit buffer with 4 virtual channels for this experiment. In Figure 6a, the results show that there is almost no performance improvement when the maximum number of packets is greater than 6. This is because the probability of more than one virtual channel entry, which have been freed up by these long packets, becoming available at the same moment is pretty low. Therefore, basically choosing the maximum number of packets one or two larger than the number of virtual channels for being standby is enough in most common cases, and this means that building the DAMQ-MP router needs only few additional registers and control logics.

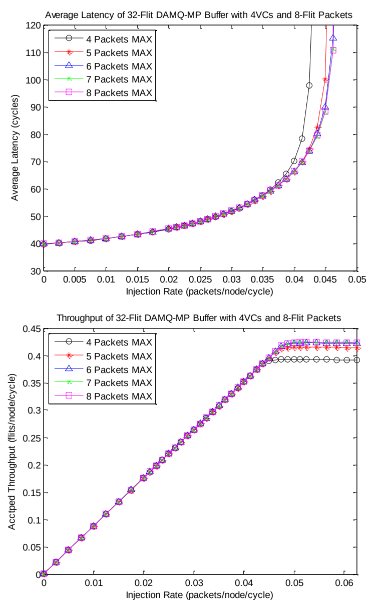
D. *Priorities for Switch Allocation*

We pick a small 12-flit buffer with 3 virtual channels to test the proposed method of adding priorities for switch allocation. The main purpose of this method is trying to arrange the limited free buffer resources to packets which are almost running out of flits and desirous of supplies. The priorities are divided into three levels according to the counting values of 0 to 3, 4 to 5, and greater than 5. In Figure 6b, this scheme seems to have no effects on the DAMQ routers, but it improves the performance of the DAMQ-MP routers when traffic loads are highly congested. Although the saturated throughput of DAMQ-MP routers only increases 1.65% after applied the priority scheme, this simple method indeed can prevent load imbalance and be easily integrated into any routers with the design of handling packet priority.

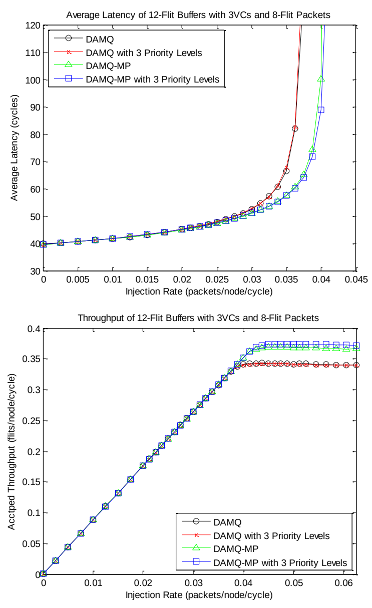
E. *Virtual Channel Reservation for High-Priority Packets*

In the previous case discussion, the DAMQ-based buffers should more fitting than the SAMQ-based ones to the method of reserving certain high-priority dedicated virtual channels. Furthermore, high-priority packets inside the DAMQ-MP buffers have bigger chances of doing "cut-in-line" behaviors to pass through as fast as possible. Hence we design this experiment to compare the performance of DAMQ and DAMQ-MP routers with or without reservations of virtual channels to deal with high-priority packets. The buffers are set to 16-flit with 4 virtual channels, and the traffic loads are made up by 1% 1-flit high-priority and 99% 8-flit low-priority packets. If using high-priority reservation scheme, at least one of these virtual channels is reserved for high-priority packets; otherwise, all virtual channels are identical and can be used by any kinds of packets.

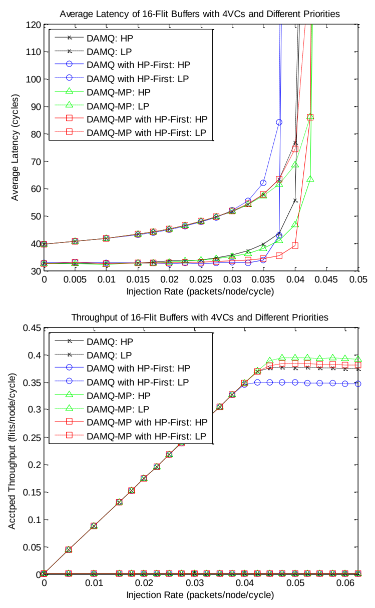
In Figure 6c, average latencies of high-priority and low-priority packets are saturated simultaneously because both kinds of packets injected by the same router node commonly come from the identical source queue. There are no differences among the performance results of these four sets while the packet injection rates are low. This observation proves that the dynamic buffer allocation mechanism of DAMQ and DAMQ-MP organizations indeed reduces the impact of taking some virtual channels for high-priority reservations on performance. Besides, this reservation scheme can facilitate an interconnection network to keep the average latency of high-priority packets almost constant until the injection rate reaches a higher value. The curves of average latency for low-priority packets in both organizations adopting the reservation scheme rise earlier than their counterparts without using the scheme. It is



(a) Maximum Number of Packets



(b) Priorities for Switch Allocation



(c) VC Reservation for HP Packets

Figure 6. Performance of DAMQ-based routers with different schemes. HP and LP stand for high-priority and low-priority respectively.

because the maximum amount of virtual channels which can be used by the low-priority packets decreases if using the reservation scheme; this also affects the saturated accepted throughputs. DAMQ-MP routers can sustain low-latency transferring services for high-priority packets at a higher traffic injection rate than DAMQ routers after they use the reservation scheme. This is because not only a DAMQ-MP buffer can accommodate more high-priority packets but also the "cut-in-line" behaviors happen to speed up the high-priority packets passing through the buffer internally.

V. Conclusion

In this paper, we introduced a novel DAMQ-MP organization for input buffers which can accommodate multiple packets more than the number of virtual channels. Because DAMQ-MP buffers are not constrained by the limitation of one packet per virtual channel existing in these old-fashioned SAMQ and DAMQ buffers, they can well utilize those scarce buffer storage and physical link resources. In addition, the DAMQ-MP scheme can bring standby packets in advance to reduce the waiting latencies of switching packets in virtual channel exits. The original DAMQ organization solves the problem of shallow virtual channel depth while using small SAMQ input buffers, and the proposed DAMQ-MP scheme further solve the problem of few working virtual channels for DAMQ routers while carrying heavy traffic loads. In addition, we discussed two methods applicable to DAMQ-based buffers, which are adding priorities for switch allocation and reserving virtual channels only for high-priority packets. We performed some experiments on three buffer organizations and proved that the DAMQ-MP scheme can take advantages in many scenarios. These experimental results showed that DAMQ-MP routers can have up to 24.52% higher saturated throughput than SAMQ and DAMQ counterparts.

Acknowledgment

The authors thank the support from NSC under grants 102-2219-E-007-002 and 102-2220-E-007-003 and MOEA under grant 102-EC-17-A-02-S1-202.

References

[1] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-GHz Mesh Interconnect for a Teraflops Processor", *IEEE Micro*, vol. 27, no. 5, pp. 51-61, 2007.

[2] Y. Tamir and G. L. Frazier, "Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches", *IEEE Transactions on Computers*, vol. 41, no. 6, pp. 725-737, 1992.

[3] J. Park, B. W. O'Krafka, S. Vassiliadis, and J. Delgado-Frias, "Design and Evaluation of a DAMQ Multiprocessor Network with Self-Compacting Buffers", In *Proceedings of ACM/IEEE Conference on Supercomputing*, pp. 713-722, 1994.

[4] J. Liu and J. G. Delgado-Frias, "A Shared Self-Compacting Buffer for Network-On-Chip Systems", In *Proceedings of IEEE International Midwest Symposium on Circuits and Systems*, pp. 26-30, 2006.

[5] R. Sivaram, C. B. Stunkel, and D. K. Panda, "HIPIQS: A High-Performance Switch Architecture Using Input Queuing", *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 275-289, 2002.

[6] M. Rezazad and H. Sarbazi-Azad, "The Effect of Virtual Channel Organization on the Performance of Interconnection Networks", In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, 2005.

[7] C. A. Nicopoulos, D. Park, J. Kim, N. Vijaykrishnan, M. S. Yousif, and C. R. Das, "ViChaR: A Dynamic Virtual Channel Regulator for Network-on-Chip Routers", In *Proceedings of International Symposium on Microarchitecture*, pp. 333-346, 2006.

[8] M. Lai, Z. Wang, L. Gao, H. Lu, and K. Dai, "A Dynamically-Allocated Virtual Channel Architecture with Congestion Awareness for On-Chip Routers", In *Proceedings of ACM/IEEE Design Automation Conference*, pp. 630-633, 2008.

SIMULATION OF A CENTRALIZED REPUTATION SYSTEM FOR VANETS

Mário Cleber Bidóia[1], Marcos Antônio[1] Cavenaghi, Roberta Spolon[1], Renata Spolon[2], Aleardo Manacero Jr.[2], Daniel Correa Lobato[3]
[1] Department of Computing, Univ. EstadualPaulista - UNESP, Bauru, SP, Brazil
[2] Computer Science and Statistics Dept, Univ. EstadualPaulista - UNESP, São José do Rio Preto, SP, Brazil
[3] Instituto Federal de Educação, Ciência e Tecnologia de São Paulo – IFSP, Catanduva, SP, Brazil

Abstract:*A new network paradigm Ad HoC has been widely studied and developed; these are the vehicular ad hoc networks (VANETs - Vehicular Ad hoc Networks). Assuming that cars are increasingly "smart", the VANETs emerged with the proposal to provide communication among vehicles of highways. This communication aims to ensure greater security for members involved in transit. Seeking to ensure this security it is necessary that the data exchanged among vehicles are reliable. One of the techniques discussed in this proposal is the use of reputation systems. In this technique, the confidence in the vehicles is based on votes from its neighbors. In other words, the more vehicles have positive reviews, the more reliable they are and vice versa. In this work, the proposal was to model and simulate a centralized reputation system, where a central unit is responsible for managing reputations. With this centralized system, the reputations of vehicles that have been evaluated are stored indefinitely and can be accessed anytime and anywhere. With the implementation and simulation, we were able to demonstrate that a vehicle, after being evaluated, has its reputation available at any time and place, preventing this malicious vehicle to succeed in later attacks.*

Keywords: VANET, security, reputation system

1 - Introduction

Vehicular Ad Hoc Networks (VANETs) [1] are a subclass of MANETs, which play crucial roles in road safety, as the detection of traffic accidents and traffic congestion reduction. Currently it has been the subject of several studies, mainly focused on the area of traffic safety, such as traffic information or alert collisions. However, as regards the exchange of information among vehicles, it is important that they are reliable. As this type of network has an ephemeral feature, due to the high mobility and speed of the nodes, it is necessary to use more robust techniques to ensure the security of information exchanged by vehicles.

One proposal that has been studied is based on reputation systems. In these systems, the reliability of vehicles is based on a collection of opinions about a particular vehicle. This technique utilizes the use of helpful vehicles near certain individual to determine whether their information should be accepted or repealed.

This paper aims at the proposal of the implementation and simulation of a new approach for the reputation system using a centralized architecture.

In Section II the VANET architecture and its main features is presented. Session III is discussed in the related work. In Section IV characteristics of a centralized reputation is displayed. Section V is discussed in the simulations and their results. Finally, in Section VI the conclusions and proposals for future work is presented.

2 – VANET

An evolution in the automotive field is the idea of "consciousness", meaning that a vehicle is aware of its neighborhood, including the presence and location of other vehicles. Conscious vehicles have a network of sensors connected to a central data processing that provides communication based on Ethernet interfaces, USB, Bluetooth, 802.11 standards, among others.

In this scenario, a set of vehicles that communicate with each other is an example of a wireless mobile network inserted in a context of ubiquitous computing. In this case, in addition to the systems implementing the communication between the vehicles, they also need to protect the personal information of their users. This type of network is called VANET (Vehicular Ad hoc Networks) [1].

In this type of network vehicles act like us. One of the major goals of this network is to provide safe conditions for the traffic of vehicles. This can be done through the exchange of information among vehicles, warning about accidents, congestion, and others.

The architecture of a VANETs is formed by the communication among vehicles (V2V), through wireless connections, and among the vehicles and fixed base stations (RSU - Road Side Unit) that makes the street or highway infrastructure (V2I) (Figure 1).

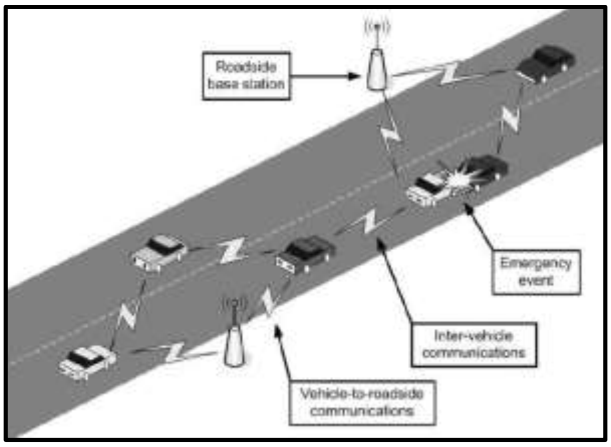

Roadside base station
Emergency event
Inter-vehicle communications
Vehicle-to-roadside communications

Figure1 - - Basic architecture of a VANET (Raya e Hubaux, 2005)

3 – Related Work

Distributed systems in which there is no centralized coordination, such as Peer-to-Peer (P2P) networks VANETs and MANETs, are subject to various types of enemies and attacks. Some simple techniques such as encryption can neutralize certain external threats because the enemies are not able to access the data exchanged over the network. However, there is the risk of an authenticated member take advantage of the situation to attack other network members. Thus, it is necessary to use more effective defense mechanisms, which are capable of shooting down these deficiencies. One proposal that has been approached trying to secure the cooperation among the nodes of the network is the use of reputation systems.

The purpose of the reputation system is to build trust value for each node of this network, and based on these values the other nodes can decide who should they rely on, encouraging trustworthy behavior. Resnick e Zeckhauser[3] lists three targets for reputation systems:

- Provide information to distinguish between reliable and unreliable pair.
- Encourage peers to act in a trustworthy manner.
- To discourage suspected peers of participating in this service, reputation mechanism is introduced.

The concept of reputation can be defined as a collective measure of reliability in a person or thing based on indications or assessments of members of a community. Thus, the individual level of trust in that person or thing can be obtained from a combination of the received nominations and the personal experiences [4]. In VANETs, the reputation of a vehicle can be considered as a set of beliefs held by other vehicles on it. That reputation is then used as an important standard in making decisions, such as forwarding or discarding packets sent by that vehicle, considering it or disregarding it as an option in the routing of data, considering or disregarding information it passed by, etc.

In the Vehicle Ad-Hoc Reputation System (VARS) [5], each vehicle adds its opinion about the veracity of the messages it received and forwarded, so everyone can use these opinions to calculate reputation.

The work of Wang e Chigan [6] proposes a mechanism that, instead of considering past records, has only vehicle behavior at runtime to define instant reputations. The work is only concerned about possible changes of message data during routing, without evaluating the content of the message itself. Thus, messages containing erroneous information can be easily distributed over the network.

Ostermaier, Dotzer e Strassberger[7] present a system based on votes to increase the security of the decisions taken by the vehicles on events reported in LDW applications schema. From the messages received within the area of dissemination, the paper evaluates the performance of four different methods of decision executed when the vehicle enters the area of decision Event: Last message, Most messages, Most recent messages X Latest X Messages considering a lower limit. The efficiency of these four methods is measured in a network subjected to attacks from false alarms and switched type alarms.

Lo e Tsai[8] present a reputation system based on events. The value of the reputation of an event defines the intensity of this event on the network in which its initial value is zero.

The mechanism Patwardhan et al [9] adds the reputations using a simple voting scheme, like Most of messages [7]. Thus, a vehicle collects information about events until data sent by a trusted source are received when the decision is based solely on information contained in this message, or a minimal amount of messages is received, when the voting scheme is implemented. The proposed solution does not allow the sharing of reputation data among the vehicles of the network, which can represent a considerable delay in the establishment of trust and intrusion detection process.

InDhurandher et al [10], the authors introduced VSRP, an algorithm based on reputation that uses confidence values assigned to nodes to detect and eliminate the network malicious nodes. In fact, this work is mainly in the detection of malicious nodes.

Huang [11] presents a reputation system based on cascade votes in which the weight of the information is related to the vehicle position, that is, the closest vehicles to an event have to evaluate a greater weight information. This system aims to assess the information, i.e. it tries to assess which information is false and which is not instead of trying to find a malicious vehicle.

4 – Centralized Reputation System (CRS)

Much of the research and targeted work for reputation systems have approached systems based on distributed architectures, in which the cars themselves store information related to reputations of neighboring vehicles. These systems seem the most sensible for this type of approach, due to the assumption that the vehicle shall be responsible for storing information about the neighboring vehicles. However, due to high mobility of the connections among vehicles, there has been some related data traffic connections problems: the amount of information exchanged, for how long a vehicle should store the information about the vehicle, etc.

To minimize these problems and seeking a new approach, this study aims to develop and simulate a centralized reputation for vehicular ad hoc networks. In this system, the Base Stations (RSU) are responsible for storing the information about the reputations of vehicles, and are positioned on the sides of highways. In addition, there will be a central unit (CU) that will connect all RSU, and will be responsible for storing the reputations of all vehicles (Figure 2).

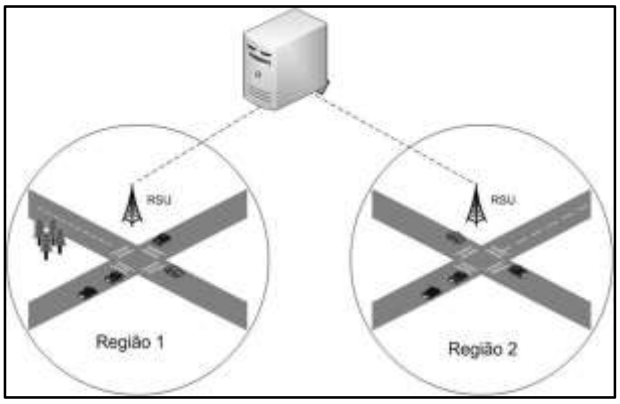


Figure2 - Centralized architecture

These Base Stations (RSU) are connected to a database management system (DBMS) that stores information about the reputations of vehicles. The fields of the table are the following:

- **ID_Vehicle**: Field identification of the vehicle. This field is unique to each vehicle. Some studies [12] propose to use pseudonyms instead of using an ID for the vehicle, so that the privacy of users is guaranteed. However, the certification unit that manages these nicknames know the ID of the vehicle, so we will consider this ID to our architecture.

- **Reputation**: degree of reputation of the vehicle. The degree of reputation is contained in the interval [0, 10], where 0 is the minimum value and 10 the maximum value.

The vehicles will make the decisions regarding an information-based degree of reputation of the issuing vehicle information. The higher the degree of reputation of the vehicle the more reliable it will be.

5 – Simulations

The simulator used to measure performance and operation of the CRS was NCTUns [13].

For the evaluation of the results obtained, the following behaviors of vehicles have been checked:

- **Average speed:** average speed of vehicles before and after receiving an alert message.
- **Distance:** distance traveled by a vehicle while waiting for a response from RSU
- **Response time:** Analyzing the speed and distance data you can determine the waiting time for response of RSU

Four scenarios have been simulated with different characteristics. Each simulation had a duration of 60 seconds and for each one there was a vehicle responsible for sending the alert message.

The first scenario simulated a malicious vehicle launched an attack, sending a false warning (warning of congestion, for example), in the region. At the first moment, there is no information about the vehicle that sent the message, this way other vehicles tend to trust the message and take some decision.

Figure 3 shows the average speed of the vehicles before and after receiving the alert and getting the response of RSU. As it can be seen the attack can be considered successful because the vehicles reduced their speed substantially.

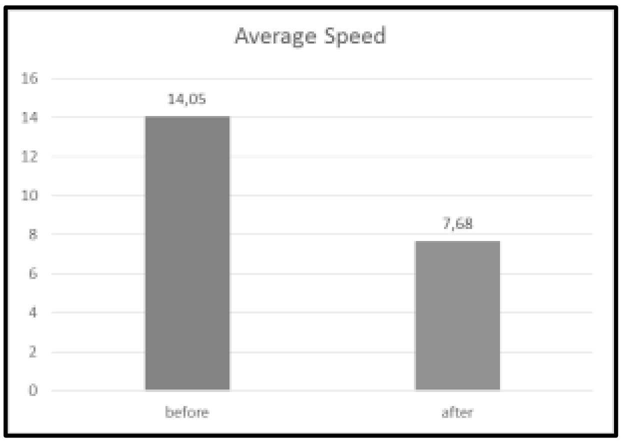


Figure3 - Average speeds before and after receiving the alert in the scenario 01

In Figure 4 it is presented the distance and time traveled by the vehicle while waiting for response from RSU 01 in the scenario.

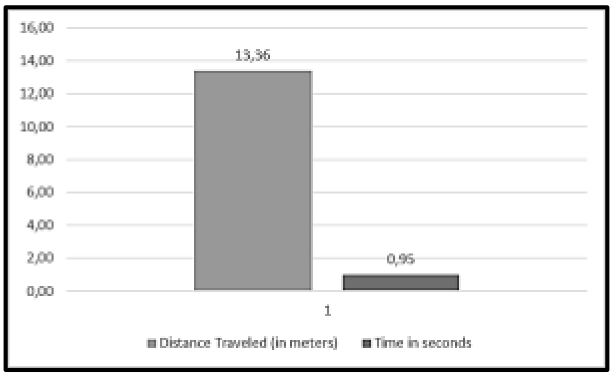


Figure4 - Average waiting time and distance traveled in response RSU scenario 01

In scenario 01, the attack can be considered a success because the attacker got what he wanted: a free highway. This was due to the fact that his reputation is unknown, and for this case, we chose to make vehicles rely on messages, which led to the result. However, this vehicle would not succeed in further attacks, because it was negatively evaluated and it has been presented in this work, this reputation is shared by the RSUs and UC.

To simulate the scenario 02 it was considered that the same vehicle that performed the attack in the previous scenario travels a route to a new area and performs the attack again. However for this scenario it assumes that after the first attack this vehicle was negatively evaluated and is therefore with a low reputation in UC, whereas RSU1 relayed the information to register.

As the attacker vehicle was negatively assessed, and therefore has a low reputation, it is expected that the vehicles by receiving the information from the reputation ignore the messages received from this vehicle and do not alter their route and speed, making it an inefficient attack.

Figure 5 shows the average of the speed of vehicles. In addition, as it can be seen, in this scenario the attack was not successful because the vehicles received the response of RSU reported that the reputation of the issuing vehicle alert was low, so the vehicles ignored the message.

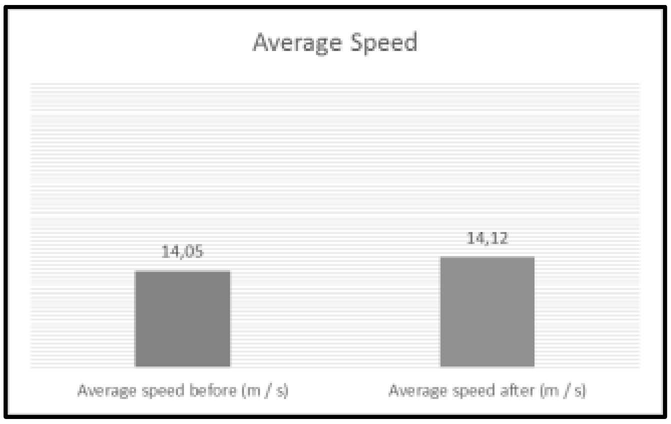


Figure5 - Average speeds before and after receiving the alert in the scenario 02

In Figure 6 it is presented the distance and time traveled by the vehicle while waiting for response from RSU in the scenario 02.

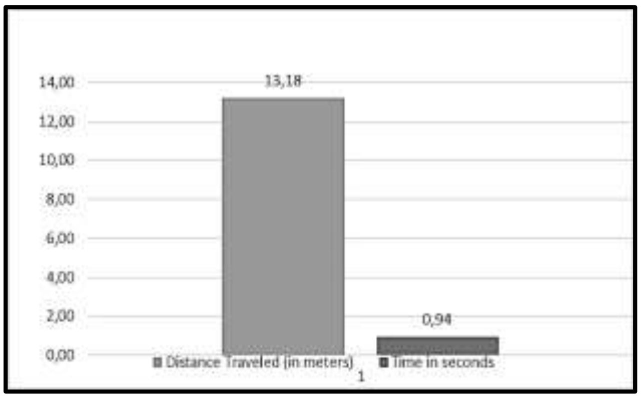


Figure6 - Average waiting time and distance traveled in response RSU scenario 02

The result of this negative evaluation is presented in the scenario 02 where the same vehicle, which performed the earlier attack, tries to execute another attack. In this scenario it is verified one of the main advantages of a centralized reputation system, since it has information about the reputation of a vehicle no matter which location he is it is possible to get a response on this reputation.

In Scenario 02, it can be considered that the attack was not successful because when the vehicles receive the alerts they refer to RSU, which does not have the reputation of the vehicle information so they consult UC to get an answer. Having received the update reputation of this vehicle, UC returns the response to RSU and consequently passes it on to the vehicles. Vehicles which receive the answer check that the reputation of the sender vehicle is low and therefore do not rely on messages sent by this vehicle. In the simulation results presented in the scenario 02, we can see that the attack did not affect the behavior of vehicles that kept their routes and speeds.

In scenario 03 it is considered a vehicle with high reputation, i.e. it is a reliable vehicle. This transmitter vehicle will send a real alert. Once the vehicles receive this message, they follow the architecture of centralized reputation, consulting the RSU to verify the reputation of the issuing vehicle.

Figure 7 presents the average speed of vehicles. As expected the vehicles after receiving information from the high reputation of the sender vehicle alert, trust the message and change their routes and speeds to avoid other accidents.

In Figure 8 it is shown the time and distance traveled by the vehicle while awaiting the response of RSU in scenario 03.

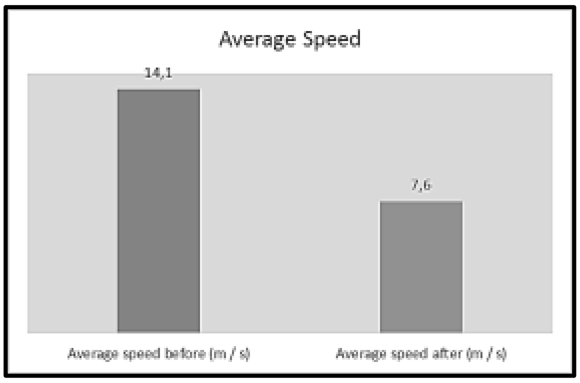


Figure7 - Average speeds before and after receiving the alert in the scenario 03

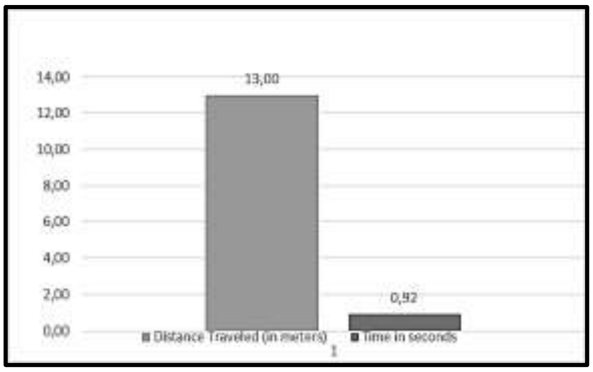


Figure8 - Average waiting time and distance traveled in response RSU scenario 03

The simulated environment in scenario 03 presented a vehicle with high reputation sending an alert message. With the simulation results we can see that the centralized reputation system worked as expected, i.e., vehicles, by receiving a message from a vehicle with high reputation alert, relied on this message and consequently took the precautionary steps, or is, slowed down and sought a new route.

In the fourth and last scenario, it was considered the same vehicle from the previous scenario, high reputation, which sent a real message but in this moment it changed the region and started sending a false alert.

In Figure 9, the average speed of the vehicles is presented. As it can be observed in this scenario the attack succeeded because the vehicles decreased their speeds.

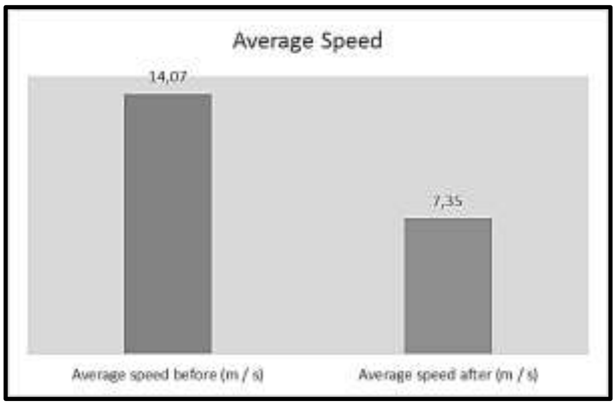


Figure9 - Average speeds before and after receiving the alert in the scenario 04

In Figure 10 it is presented the time and distance traveled by the vehicle while waiting for response from RSU in the scenario 04.

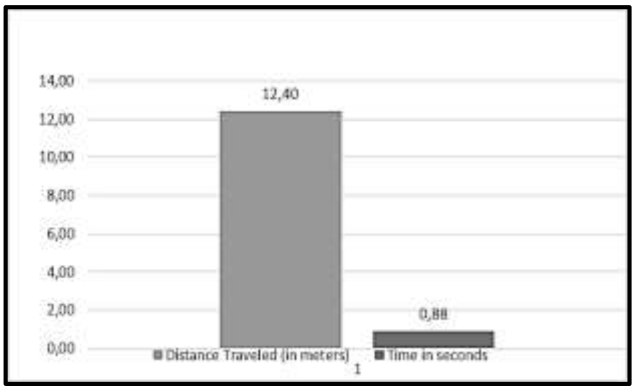


Figure10 - Average waiting time and distance traveled in response RSU scenario 04

The scenario 04 presented one of the main disadvantages of the SRC, where a vehicle with a known and high reputation decides, for some reason, to send a false alert. This is a critical case, as for having a high reputation, other vehicles tend to trust the received message, but this time the message is false. With the collected results of the simulation, we can see that the attack was successful, because the other vehicles altered their routes and slowed down.

Other analyzed data was the response time of the RSU to vehicle that performs a query. In all scenarios, the average response time was less than one second, which makes the vehicles travel a short distance and makes it possible to take a decision on a considerable time.

6 – Conclusions

There are currently several proposed reputation systems, however, as they were presented, all of them work in a distributed architecture. In the developed work in this project it was chosen to present and simulate a new architecture for the reputation system. This architecture is a Centralized Reputation System, in which a central unit is responsible for the reputation of the vehicles.

After studying the existing architectures, the research of this paper presented a new approach for reputation systems. With the study in this paper it was possible to verify some advantages and disadvantages of this architecture over the others.

- **Advantages**
  - Indefinite storage of reputation;
  - Infinite amount of stored reputations;
  - Reputation may be consulted at any place and time.
- **Disadvantages**
  - High cost to implement the architecture to provide RSU in all regions;
  - High cost with high performance processors in order to process all requests;
  - The problem of a vehicle with high reputation send a false alert;

One of the main advantages is related to the position of the attacker vehicle, because in this architecture if a vehicle was negatively evaluated in a given region, its reputation may be consulted at any time and at any place, it means that even if the vehicle changes neighborhood, city, state or even country (depending on the infrastructure) whenever it launches a new attack, vehicles connected to it can check its reputation, because the Central Unit will have received at a previous time this information. In other architectures, this could not be verified because the reputation of the vehicle would be assessed only at the moment.

Another advantage that can be noted is related to the storage time of the vehicle's reputation. In distributed architectures there is a great discussion on how long the vehicle should store information about the reputation of a vehicle, because due to mobility of the network, it may be that the vehicle never connects again with the vehicle which keeps the information, but it may occur that they connect again the next day or month.

As it could be verified the centralized reputation system shows to be more advantageous than a distributed architecture, because it delegates the responsibility to another entity. However, this centralized architecture has its drawbacks. The main one is related to the infrastructure. In order to apply this architecture there should be base stations for all streets, roads and highways to be perfectly functional in addition to having high-performance processors to meet all requests, and this is not something easy to apply, mainly because of the financial cost.

7 – References

[1] J. Luo, J.P. Hubaux, “A survey of inter-vehicle communication,” Tech. Rep. IC/2004/24, EPFL, Lausanne, Switzerland, 2004.

[2] J J. Blum, A. Eskandarian, and L.J. Huffman, “Challenges of intervehicle ad hoc networks,” IEEE Transactions on Intelligent Transportation Systems, vol. 5, no. 4, pp. 347–351, 2004.

[3] Resnick, P.; Zeckhauser, R. Trust among strangers in Internet transactions: Empirical analysis of eBay's reputation system. In Michael R. Baye, editor, The Economics of the Internet and E-Commerce, volume 11 of Advances in Applied Microeconomics, pages 127-157. Elsevier Science, 2002.

[4] Swamynathan, G. et al.Globally decoupled reputations for large distributed networks. Adv. MultiMedia, 2007(1):12-12.

[5] Dotzer, F.; Fischer, L. e Magiera, P. Vars: A vehicle ad-hoc network reputation system. In WOWMOM '05: Proceedings of the Sixth IEEE International Symposium on World of Wireless Mobile and Multimedia Networks, pp. 454-456, Washington, DC, USA. IEEE Computer Society - 2005.

[6] Wang, Z. e Chigan, C. Countermeasure uncooperative behaviors with dynamic trust-token in vanets.In ICC '07: Proceedings of IEEE International Conference on Communications, pp. 3959-3964, Glasgow, Scotland. IEEE - 2007.

[7] Ostermaier, B.; Dotzer, F. e Strassberger, M. Enhancing the security of local danger warnings in vanets - a simulative analysis of voting schemes. In ARES '07: Proceedings of the The Second International Conference on Availability, Reliability and Security, pp. 422-431, Washington, DC, USA. IEEE Computer Society - 2007.

[8] Nai-Wei Lo, Hsiao-Chien Tsai, “A Reputation System for Traffic Safety Event on Vehicular Ad Hoc Networks”,

EURASIP Journal on Wireless Communications and Networking Volume 2009, Article ID 125348, 10 pages.

[9] Patwardhan, A. et al. A data intensive reputation management scheme for vehicular ad hoc networks. In Proceedings of the Second International Workshop on Vehicle-to-Vehicle Communications.IEEE - 2006.

[10 Dhurandher, S. K.; Obaidat, M. S.; Jaiswal, A.; Tiwari, A.; and TYAGI, A. Securing vehicular networks a reputation and plausibility checks-based approach. IEEE GLOBECOM Workshops, pages 1550-1554, 2010.

[11] Z. Huang, "On reputation and data-centric misbehavior detection mechanisms for vanet," Master's thesis, School of Electrical Engineering & Computer Science, Faculty of Engineering, Ottawa, Canada, 2011.

[12] Hubaux, J. P.; Capkun, S. e Luo, J.The security and privacy of smart vehicles.IEEE Security and Privacy, 2(3):49-55- 2004.

[13] Wang, S.Y, Chou, C. L., Chiu, Y. H., Tzeng, Y. S., Hsu, M. S., Cheng, Y. W., Liu, W. L. e Ho, T. W. NCTUns 4.0: An Integrated Simulation Platform for Vehicular Traffic. In Communication, and Network Researches, WiVec'07, 2007.

How to Tolerate Simultaneous Leave of Peers in Tree-Structured P2P Live Streaming Systems

Tatsuya KOUCHI[1] **and Satoshi FUJITA**[1]
[1]Department of Information Engineering, Hiroshima University
Higashi-Hiroshima, 739-8527, Japan

Abstract—*A characteristic of Peer-to-Peer (P2P) systems is that several peers tend to leave the system in a short time. Such a simultaneous leave of peers causes serious problems such as the leave of backup peers and the occurrence of cyclic reference to the backup peers. In this paper, we propose several techniques to enhance the resilience of tree-structured P2P live streaming systems to such simultaneous peer leaves. The effect of the proposed techniques is evaluated by simulation. The result of simulation indicates that the proposed scheme reduces the number of fails to connect to a backup peer and the time required for the reconnection after a fail, even under a high churn rate.*

Keywords: Peer-to-Peer live streaming, peer churn, resilience to simultaneous leave, cyclic reference.

1. Introduction

Recently, Peer-to-Peer (P2P) technology has attracted considerable attentions as a way of distributing large contents to many users without causing a bottleneck at specific nodes. Many network applications based on the P2P technology are widely used, which include file sharing, VoIP phone, video-on-demand, and live streaming. Among those applications, **P2P live streaming** is one of the most promising approaches to improve the performance of existing systems with respect to the scalability, the reliability and the extendibility.

The architecture for P2P live streaming systems can be classified into three types by the structure of the overlay, i.e., tree-type, mesh-type and their hybrid. In tree-structured P2P live streaming systems [2], [5], a live stream is fed to the root of the tree and is delivered to the other peers by repeating the forwarding of the received stream toward down-streams. The overhead of such a simple forwarding scheme is small, while it has a serious drawback such that *the leave of a non-leaf peer immediately stops the feeding to down-steams*. This motivates the proposal of techniques to tolerate such a "peer churn" in tree-structured systems, such as the preparation of backup peers and the use of multiple-trees. On the other hand, in mesh-structured systems such as CoolSreaming [8], the delivery of a live stream is realized by repeating the data exchange between adjacent peers. Thus in contrast to tree-structured systems, mesh-structured systems are more resilient to peer churns, since even if one adjacent peer leaves, it still has a chance to receive a copy of the stream from other peers. However, such a redundancy significantly increases the overhead to maintain the link to all adjacent peers, which frequently causes a long delay in the delivery of live streams. Hybrid architecture was proposed to overcome the drawback of the above two types, which includes HON [9] and ChunkySpread [6] as the representatives.

In this paper, we propose several techniques to enhance the resilience of tree-structured P2P live streaming systems to peer churns. More concretely, we focus on a hybrid architecture proposed by Wang *et al.* called mTreebone [7], and propose a way to tolerate simultaneous leave of several peers. The mTreebone realizes a resilience to peer churns in the following manner: 1) it introduces the notion of stability representing the tendency of staying in the system and organizes a tree-structured overlay consisting merely of such stable peers (a formal definition of "stability" will be given in Section 2); 2) for each peer in the tree-structured overlay, it selects a candidate for the reconnection so that it can immediately start the communication with the candidate when the current parent leaves the system. Although such techniques used in the mTreebone are effective to tolerate the leave of a single peer, in actual systems, several peers tend to leave the system in a short time period. Such a *simultaneous leave of several peers* would cause the following serious problems: 1) scarcity of the upload slot of candidate peers due to the exhaustion by several peers; 2) the occurrence of a cycle by connecting to a descendant peer which was not being a descendant at the time of selecting candidate; and 3) the leave of the candidate peer which does not occur when merely a single peer leaves. In this paper, we propose several techniques to overcome such issues.

The effect of the proposed techniques is evaluated by simulation. The result of simulation indicates that the proposed scheme reduces the number of fails to connect to a candidate peer, which indicates that candidates are appropriately selected so that it causes no concentration of the selections, and it reduces the time required for the reconnection after a fail even under a high churn rate such that several peers simultaneously leaves.

The remainder of this paper is organized as follows. Section 2 overviews techniques used in the mTreebone. Section 3 describes the proposed method and Section 4 describes the simulation result. Finally, Section 5 concludes the paper with future work.

2. mTreebone

In the mTreebone, several peers satisfying a certain stability condition organize a tree-structured overlay called **tree-bone** and a live stream is given to the root of the tree-bone and is delivered to all peers along tree-bone edges. Peers which do not participate in the tree-bone can receive the stream from any peer participating in the tree, and a mesh-structure is used to compensate the weakness of the tree-structure against peer churns. In this section, we describe key techniques used in the mTreebone to be resilient to peer churns.

2.1 Stability of Peers

In the mTreebone, a peer is said to be **stable** if it stays in the system for a time exceeding a threshold $T(t)$, where t is the elapsed time after starting the current session of live stream (i.e., the threshold used in the stability condition is a function of the elapsed time). Threshold $T(t)$ is determined so that the expected service time (EST) of each peer is maximized. Assume that a session starts at time $t = 0$ and ends at time $t = L$. Let $f(\cdot)$ be the probability density function of the life time of a peer. The reader should note that the maximum service time of a peer which joins the system at time $t = \tau$ is at most $L - \tau - T(\tau)$ since the session ends at time L and it can start the service to other peers after passing $T(\tau)$ time after the join (recall that the mTreebone does not allow "unstable" peers to participate in the tree-bone as an uploader). Hence, since EST is obtained by subtracting $T(\tau)$ from the expected life time, it can be represented as follows:

$$EST(t) = \frac{\int_{T(t)}^{L-t} xf(x)dx + \int_{L-t}^{\infty}(L-t)f(x)dx}{\int_{T(t)}^{\infty} f(x)dx} - T(t) \quad (1)$$

Here the integral in the numerator reflects the fact that the upper bound on the life time is $L - \tau$, and the denominator normalizes it so that the integral from $T(\tau)$ to the infinity becomes one.

According to the observation shown in [1], [4], the mTreebone assumes that the probability density function follows a Pareto distribution with shape parameter k [7]. For such a specific function, $EST(t)$ is calculated as follows:

$$EST(t) = \frac{T(t)}{k-1}\left[1 - \left(\frac{T(t)}{L-t}\right)^{k-1}\right],$$

which takes the maximum value when

$$T(t) = (L-t)\left(\frac{1}{k}\right)^{\frac{1}{k-1}}. \quad (2)$$

From Equation (2), we can observe that an optimal value of the threshold is proportional to the remaining time of the session at the time of join. In addition, the coefficient $(1/k)^{1/(k-1)}$ converges to 0.3 as the value of k converges to 1. Thus, if the life time follows a Pareto distribution and the shape parameter k is close to 1, the optimal value of the threshold can be approximated by the 30% of the remaining time of the session at the time of join.

2.2 Probabilistic Promotion

The simplest rule for the participation in the tree-bone as an uploader is that *each peer which joins the system at time t can participate in the tree-bone at time $t + T(t)$ or later*. However, if we strictly apply this rule, it causes the lack of upload bandwidth particularly in an early stage of the live streaming session. To overcome such an issue, in the mTreebone, it takes an approach such that it randomly promotes unstable peers as a tree-bone peer. More concretely, when the staying time of a peer is s, it is promoted as a tree-bone peer with the following probability:

$$p(s) = \frac{1}{T(t) - s + 1}.$$

Using such a probabilistic mechanism, we can realize a situation such that the probability of participating in the tree-bone is $s/T(t)$, which reaches one when $s = T(t)$ [7].

2.3 Candidate for the Reconnection

To watch live streams in a continuous manner, each peer which detects the leave of the parent should immediately find another tree-bone peer to have enough upload slot, and establish a connection to the peer to receive the suspended stream. In addition, to reduce the suspension time as much as possible, such a candidate for the reconnection should be selected before actually detecting the leave of the parent. In the mTreebone, each tree-bone peer randomly selects candidates to have enough upload slots from adjacent peers in the mesh-structured overlay, and the other peers, which do not have a child in the overlay, select no candidate in advance to reduce the cost for the maintenance.

3. Proposed Scheme

The mTreebone described in the last section tolerates the leave of peers by preparing a candidate for reconnection for each tree-bone peer. Such a simple approach works well if the frequency of the leave of peers is not high. However, in actual systems, several peers tend to leave the system in a short time period as in the case of the halftime of football game and the end of the performance of specific musicians. Such a *simultaneous leave of several peers* would cause the following serious problems: 1) scarcity of the upload slot of candidate peers due to the exhaustion by several peers; 2) the occurrence of a cycle by connecting to a descendant peer which was not being a descendant at the time of selecting candidate; and 3) the leave of the candidate peer which does not occur when merely a single peer leaves. In this section, we propose three techniques to resolve those issues.

3.1 Fractional Reservation of Upload Slot

Upload slots of a candidate peer will be exhausted if it is selected by many peers as a candidate and its upload slots are consumed by those peers due to the leave of their parent. As was described above, in the mTreebone, such a concentration of the selection is relaxed by using a randomized approach. In contrast to that, in the proposed scheme, we take an approach such that each selection *reserves* a small fraction of the bandwidth at the time of selecting the candidate (hereafter, we will call this technique **Proposal 1**).

Assume that each stream consumes one unit of the upload bandwidth. Let $c[i]$ denote the residual bandwidth of peer i including the bandwidth which was reserved but is not being used by the other peers. In the proposed scheme, when peer j selects peer i as a candidate, peer j reserves the upload bandwidth of peer i of amount $\alpha \in (0, 1]$. According to this modification, the rule for the selection of candidates in the mTreebone is modified as follows:

> Peer j can select peer i as a candidate only when $c[i]$ minus reserved bandwidth is at least α, and j can select candidate i as the parent only when $c[i] \geq 1$.

The concrete behavior of peer j after detecting the leave of the parent is as follows. Assume that peer j selected peer i as a candidate; i.e., it has successfully reserved the bandwidth of i of amount α. If $c[i] \geq 1$, peer j simply connects with peer i after the leaving of its (former) parent. If there are k peers which select i as a candidate and the use of the bandwidth by peer j reduces $c[i]$ to be less than $1+(k-1)\alpha$, i requests those k peers to change the candidate from i, in an increasing order of receiving such a request, until the condition of $c[i] \geq 1 + (k-1)\alpha$ holds (note that the value of k decreases if a peer changes its candidate from i).

3.2 Prevention from the Occurrence of a Loop

Our second technique is to prevent from the occurrence of a cyclic reference of the candidate peers. We propose two ideas in this subsection. The first idea, which will be called **Proposal 2a** hereafter, is to prepare a list of peers at each peer to keep the set of ancestors in the tree-bone. For example, if peer c receives the stream from the source through tree-bone peers a and b, then the list held by c is determined as $[a, b, c]$. This list is updated whenever the parent changes, by receiving the list held by the new parent and by adding itself to the end of the received list. If the updated list contains a peer j which selects i as a candidate, since this implies that j is selecting a descendant peer as a candidate, peer i sends a message to j to change the candidate from i. For each of the other children j', i forwards the updated list to j' to keep their list up-to-date.

The second idea, which will be referred to as **Proposal 2b** hereafter, is to prepare a local variable representing the depth in the tree-bone for each peer[1], and to select a peer to have the depth smaller than the current peer as the candidate. With such a variable, as long as the value of the variable correctly reflects the actual depth of the peer, it can prevent from the occurrence of a cyclic reference to the candidates. The power of this technique is heavily dependent on the accuracy of the variables, which causes a trade-off between the risk of cyclic reference and the cost for the maintenance.

3.3 Recovery from Cyclic Reference

The third technique, which is called **Proposal 3** hereafter, is to realize a quick recovery from cyclic reference to the candidates. The basic idea is to transmit a “probe message” toward the root of the tree-bone whenever a peer changes its parent. Each peer can proactively detect the occurrence of a cycle by receiving the message transmitted by itself, which can shorten the time required for the recovery from cyclic reference. Upon detecting a cycle, which is conducted merely by the peer which recently changes its parent, the peer disconnects the link to the parent, and tries to find (and connect to) another parent. In addition to the above simple idea, to reduce the number of reconnections as much as possible, we design the scheme in such a way that if a peer which submitted a probe message receives another probe message, it compares its ID with the ID of the originator of the message, and it forwards the message to the parent *only when* its ID is smaller than the ID of the originator. With such a mechanism, we can disconnect only one link contained in a cyclic reference even if several peers cause a cyclic reference by changing their parent.

4. Evaluation

4.1 Setup

We evaluate the performance of the proposed scheme by simulation using PeerSim simulator [3]. One step of the simulation is set to 0.02 sec and we assume that in each step, each peer conducts the following operations:

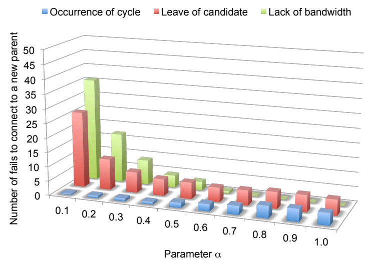
1) Receive messages from the input buffer.
2) Calculation according to the received messages.
3) Send one message to an adjacent peer,

where a message sent in the i^{th} step is received by the receiver at the beginning of the $(i+3)$rd step; i.e., we do not consider the variance of the delay of the message delivery.

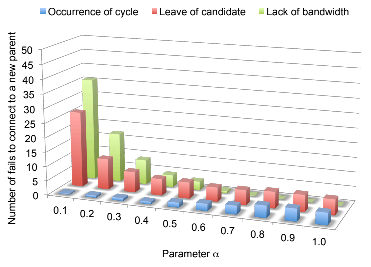
In the simulation, we used two churn models described below. In the first model, peers arrive at the system according to a Poisson distribution with mean λ and leave the system according to a Pareto distribution. The probability density function of Pareto distribution is given as

$$f(t) = \frac{k{t_m}^k}{t^{k+1}}, \tag{3}$$

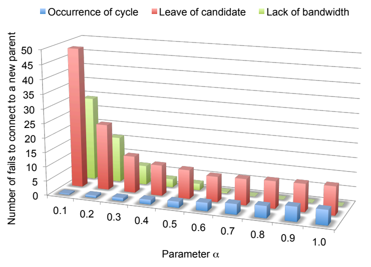
[1]The depth of a peer is defined to be the length of the path from the root to the peer.



(a) $\lambda = 4$.

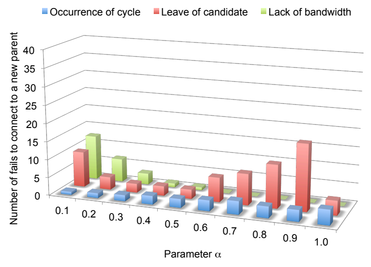


(b) $\lambda = 8$.

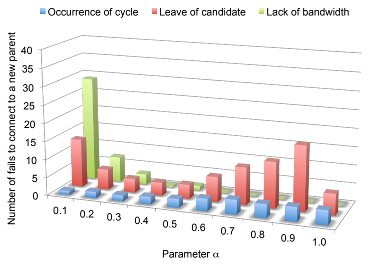


(c) $\lambda = 12$.

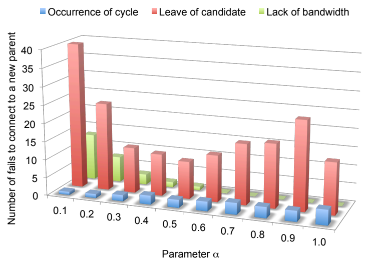
Fig. 1: Impact of parameters α and λ to the number of fails under the first churn model.



(a) $T_d = 600$.



(b) $T_d = 300$.



(c) $T_d = 100$.

Fig. 2: Impact of parameters α and T_d to the number of fails under the second churn model.

where k is called a shape parameter and t_m is called a scale parameter which is associated with the participation time of the peers, i.e., under this distribution, peers which participate in the system earlier will leave the system with a higher probability. Since those two parameters do not directly control the longevity of peers, in the simulation, we vary parameter λ from 4 to 12 [peers/sec] to virtually control the longevity of peers, since with a larger λ, many peers have an earlier participation time which causes a higher leaving probability within a short time. On the other hand, in the second model, all peers participate in the system at the beginning of the session, and a half of peers randomly leave the system during T_d time units, where parameter T_d is varied from 100 to 600 sec, i.e., a smaller T_d implies a shorter life time. In both models, once a peer leaves the system, it will not join the system again.

In the succeeding two subsections, we evaluate the impact of three proposed techniques to the churn tolerance of the underlying P2P live streaming systems. More concretely, as for the first technique, we evaluate the impact of parameter α and as for the second and the third techniques, we compare the performance with a scheme without such a loop avoidance/recovery mechanism. Metrics used in the evaluation are: 1) the number of fails to connect to the new parent, 2) the time spent for the reconnection after failing the connection to the new parent, and 3) the number of messages. The number of fails is itemized to the reason so that: 1a) fails due to the lack of upload bandwidth, 1b) fails due to the leave of the candidate, and 1c) fails due to the occurrence of a cyclic reference. Similarly, the time for reconnection is itemized to each of the above three reasons. Each value shown in the figures is an average over 100 trials.

Finally, parameters used in the simulation are as follows:

- The length of a live stream L is 600 sec.
- The total number of peers is 2000.
- The upload bandwidth of the source is 16 times of the full stream rate and the upload bandwidth of the other peers is varied from the fourth to the eighth of the full stream rate.

4.2 Proposal 1

This subsection evaluates the impact of parameter $\alpha(\in (0, 1])$ to the performance of the P2P live streaming systems with respect to the number of fails and the time required for the reconnection. We will also observe the impact of the churn rate to the performance.

Figure 1 illustrates the number of fails under the first churn model, where the horizontal axis indicates α and (a), (b) and (c) correspond to the case of $\lambda = 4$, 8 and 12, respectively. The number of fails due to the leave of candidates (indicated by red bars) rapidly decreases as α increases from 0.1 to 0.3, which is apparently because of the effect of the relaxation of the concentration of the selection of a specific peer as a candidate. When $\alpha = 0.1$, the number of fails due to the lack of the upload bandwidth and the number of fails due to the leave of candidates gradually increase as the value of λ increases from 4 to 8, whereas the former *decreases* as λ further increases from 8 to 12, which is because under such a high churn rate, the fail due to the leave of peers dominates the fail due to the lack of the bandwidth. The number of fails due to the lack of upload bandwidth (indicated by green bars) decreases as α increases and becomes (almost) zero for all values of λ by selecting α to be larger than 0.6. Thus we can conclude that Proposal 1 is effective even under a high churn rate.

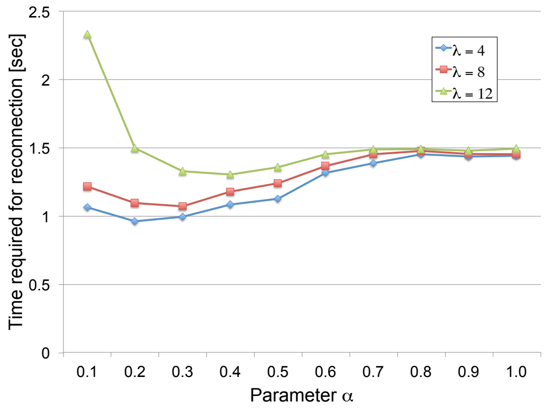
Figure 2 illustrates the number of fails under the second churn model, where (a), (b) and (c) correspond to the case of T_d = 600, 300 and 100, respectively. As for the number of fails due to the lack of the upload bandwidth and the occurrence of a cyclic reference is the same with the case of the first churn model. However, the number of fails due to the leave of candidates (indicated by red bars) significantly increases for large α's, which implies a side-effect of selecting a larger α. As a result, the number of fails takes the smallest value when α is around 0.4 to 0.5. When $\alpha = 0.1$, in contrast to the first churn model, the number of fails due to the lack of bandwidth takes almost the same value for all T_d's, while it becomes almost zero for $\alpha \geq 0.6$ as in the case of the first churn model. The analysis of such an interesting phenomenon is left as a future work.

Figure 3 illustrates the impact of α to the time required for the reconnection. Although it takes long time when α is small and the churn rate is high, the reconnection time converges to 1.5 sec as α increases under each churn model. In addition, the value of α which minimizes the reconnection time gradually increases as the churn rate increases; e.g., it changes as $\alpha = 0.2$, 0.3 and 0.4 as the value of parameter λ increases as 4, 8 and 12. By letting $\alpha_{\min}$ be the value of α minimizing the reconnection time and $\alpha_{\max}$ be the minimum value of α which stabilizes the reconnection time, the value of α should be determined to satisfy $\alpha_{\min} \leq \alpha \leq \alpha_{\max}$, since a too large α causes frequent fails as was described above. In summary, the result of simulation indicates that an appropriate value of α is around 0.5.

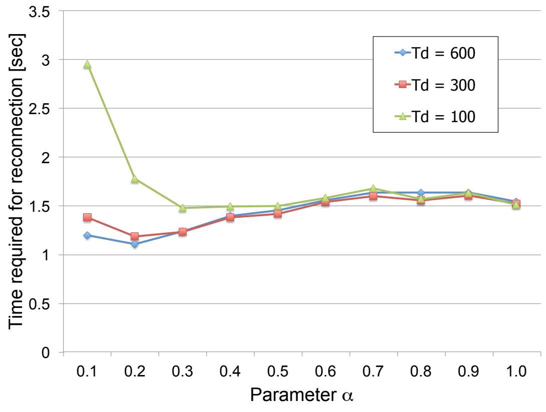
4.3 Proposals 2 and 3

In this subsection, we evaluate the impact of Proposals 2 and 3 to the performance by comparing it with the basic scheme which detects and resolves a cyclic reference in the following manner:

Basic scheme: Upon detecting the delay of the stream exceeding 1 sec, a peer transmits a probe message to the parent to verify the existence of a path to the root. If the peer receives the message transmitted by itself, it identifies the existence of a cycle, and initiates the reconnection procedure.



(a) The first churn model.

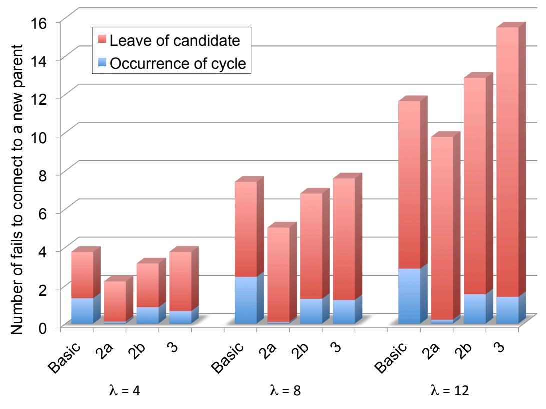


(b) The second churn model.

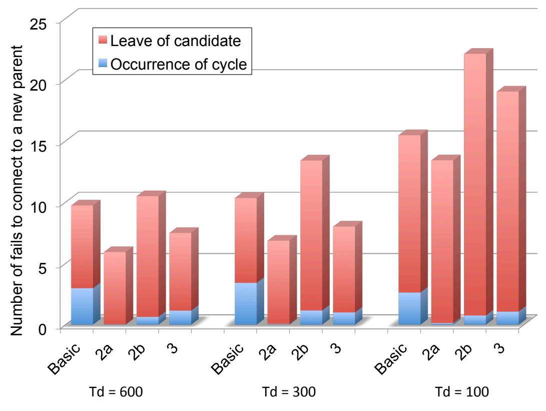
Fig. 3: Time required for the reconnection.

In the following, we fix the value of parameter α used in Proposal 1 to 0.6.

Figure 4 illustrates the impact of the churn rate to the number of fails to connect to the candidates, where four bars for each churn rate indicate the basic scheme, a scheme with Proposals 2a, 2b and 3, respectively. For each bar, red indicates the number of fails due to the leave of candidates and blue indicates the number of fails due to the occurrence of a cycle. Each technique certainly reduces the number of fails due to cycles compared with the basic scheme regardless of the churn model and the churn rate. In particular, Proposal 2a reduces it to almost zero. However, the number of fails due to the leave of candidates does not decrease by the proposed techniques, and the effect of those techniques slightly differs for each churn model; i.e., in the first model based on the Pareto distribution, the increase under Proposal 3 is the largest and in the second



(a) The first churn model.



(b) The second churn model.

Fig. 4: Impact of each proposal to the number of fails.

model, the increase under Proposal 2b is the largest. As for the time required for the reconnection, we found that Proposal 3 significantly reduces the reconnection time for each churn model. The reconnection time slightly increases by Proposal 2a under the first churn model, whereas there are no difference between Proposals 2a and 2b under the second churn model, regardless of the churn rate.

The above results indicate that the combination of Proposals 2a and 3 is the most effective. Thus finally, we evaluate the performance of the combined scheme of Proposals 2a and 3 in detail. Tables 1, 2 and Figure 5 summarize the results. From Table 1, we can observe that although the combined scheme reduces the number of fails as the churn rate increases from low to moderate, it becomes worse than the basic scheme when the churn rate is very high. As for the time required for the reconnection due to the occurrence of a cycle, the combined scheme significantly improves the basic scheme regardless of the churn model and the churn rate, as is shown in Table 2. In particular, although the reconnection

Table 1: Impact of the combined scheme to the number of fails.

Scheme	λ			T_d		
	4	8	12	600	300	100
Basic	3.74	7.41	11.63	9.72	10.33	15.46
Proposals 2a+3	2.5	5.04	14.63	6.38	6.53	19.16
Proposal 3	4.46	7.59	15.49	7.48	8.00	19.02

Table 2: Impact of the combined scheme to the time in second required for the reconnection.

Scheme	λ			T_d		
	4	8	12	600	300	100
Basic	1.46	1.46	1.48	1.64	1.67	1.74
Proposals 2a+3	0.90	0.96	1.18	1.21	1.27	1.30
Proposal 3	1.04	1.11	1.16	1.18	1.11	1.35

time of Proposal 2a is worse than the basic scheme under the first churn model, we can improve the basic scheme by combining it with Proposal 3. Figure 5 compares the number of messages issued in the combined scheme with the basic scheme. From the figure, we can see that the increase of the number of messages caused by the combined scheme is bounded by 20% of the basic scheme.

In summary, the combination of Proposals 2a and 3 could effectively bound the occurrence of cyclic references and reduces the time required for the reconnection by more than 0.4 sec compared with the basic scheme. However, as the churn rate becomes too high, it causes a significant number of fails due to the leave of the candidates, which causes the performance degradation such as the increase of the total number of fails and the number of messages.

5. Concluding Remarks

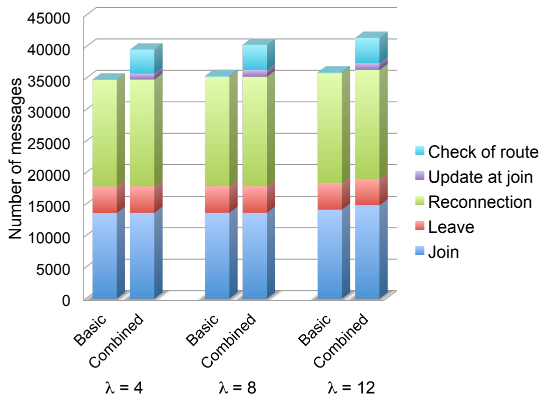
This paper proposes techniques to increase the resilience of P2P live streaming systems to the simultaneous leave of several peers. The result of simulation indicates that the proposed techniques reduce the number of fails to connect to a candidate (backup) peer and the time required for the reconnection after a fail even under a high churn rate. The refinement of the proposed techniques is left an important future work. The implementation in actual P2P live streaming systems is another crucial issue.

References

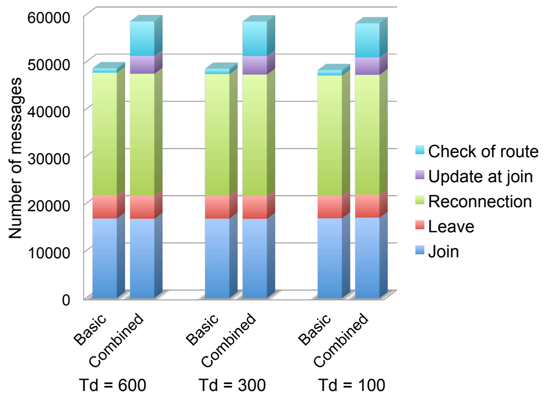
[1] K. C. Almeroth and M. H. Ammar. "Collecting and modeling the join/leave behavior of multicast group members in the MBone." In *Proc. of IEEE International Symposiun on High Performance Distributed Computing*, 1996, pages 209–216.

[2] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron and A. Singh. "SplitStream: High-bandwidth multicast in cooperative environments." In *Proc. of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, 2003, pages 298–313.

[3] URL: http//peersim.sourceforge.net/.



(a) The first churn model.



(b) The second churn model.

Fig. 5: Impact of the combined scheme to the number of messages.

[4] K. Sripanidkulchai, B. Maggs and H. Zhang. "An analysis of live streaming workloads on the Internet." In *Proc. of the 4th ACM SIGCOMM Conference on Internet Measurement (IMC '04)*, 2004, pages 41–54.

[5] D. A. Tran, K. A. Hua and D. Tai. "ZIGZAG: An efficient peer-to-peer scheme for media streaming." In *Proc. of the 22nd Annual Joint Conference of the IEEE Computer and Communications (INFOCOM 2003)*, 2003, pages 1283–1292.

[6] V. Venkataraman, K. Yoshida and P. Francis. "Chunkyspread: Heterogeneous Unstructured Tree-Based Peer-to-Peer Multicast." In *Proc. of the 14th IEEE International Conference on Network Protocols (ICNP '06)*, 2006, pages 2–11.

[7] F. Wang, Y.-Q. Xiong and J.-C. Liu. "mTreebone: A Hybrid Tree/Mesh Overlay for Application-Layer Live Video Multicast." In *Proc. of the 27th International Conference on Distributed Computing Systems (ICDCS '07)*, 2007, pages 49.

[8] X.-Y. Zhang and J.-C. Liu and B. Li and T.-S. Peter Yum. "CoolStreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming." In *Proc. of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2005)*, 2005, pages 2102–2111.

[9] M. Zhou and J.-C. Liu. "A Hybrid Overlay Network for Video-on-Demand." In *Proc. of IEEE International Conference on Communications (ICC 2005)*, 2005, pages 1309–1313.

Three Dimensional Free Space Optical Network Reconfiguration Heuristics Using Active Link Removal

Xuemei Sun, Danny Luong, and G. Young
Computer Science Department, California State Polytechnic University, Pomona, CA 91768

Abstract

Free Space Optical (FSO) communication is a type of optical communication that relying on beams of light to transmit data over long distances wirelessly. With recent increase in popularity of the technology in commercial applications as well as military and scientific applications, it is important to study new models and algorithms to improve the performance of this technology. One of the most important aspects of FSO network is the line-of-sight (LOS) requirement to sustain an uninterrupted data flow. In order to ensure a steady connection, auto-aligning transceivers can be implemented to overcome the LOS issue. In this paper, we propose several re-alignment algorithms to reconfigure a FSO network when there is a loss of line of sight. The main idea of our experiment is to expand up on the three dimensional FSO network model proposed by Kosumo et al. in 2013, addressing cases that have not been considered.

Keywords: *3D Mesh Network, Routing, Reconfigurable Network, Free Space Optics, Wireless Network.*

1. Introduction

Being a wireless technology, FSO has many advantages over its wired counterpart, fiber optic communication. Similar to fiber optic communication, FSO uses light sources and detectors to send and receive data. Nonetheless, instead of relying on extensive networks of fiber cables for transmission, FSO send data directly through the air, hence the name free-space. The main purpose for using FSO instead of fiber cable is to lower or even eliminate the costs of cable installation. FSO technology also does not require radio frequency spectrum licensing. This advantage allows FSO to be used more freely. Furthermore, it is possible to relocate an existing FSO network elsewhere, allowing the recycle of equipment. FSO technology may be the next prominent broadband network. High speed data rates, unlicensed spectrum, excellent security, low setup time, and inexpensive infrastructure are among its most attractive features [24].

Over the last few decades, FSO technology has been studied extensively. Different models and performance measurements have been proposed. There have also been many experiments done on this technology [7, 11, 14, 16- 18, 20, 21, 26]. In 2013, Kosumo et al. proposed a three dimensional (3D) model for FSO network. In their paper, several different heuristic algorithms for link reconfiguration were introduced and discussed [12]. For the sake of convenience, we will refer to their model and heuristics as KLWY (Kosumo, Luong, Wong, and Young) throughout our paper.

KLWY's 3D FSO network model provides a convenient way to study the effect that random broken links have on an auto-reconfigured FSO network over time. In their experiment, links between nodes in a simulated network are allowed to be randomly disconnected over time while the heuristic algorithms try to keep the network functional by reconnecting the nodes that have broken links. The experiment gives insight into how a network will transform after a period of time. It also shows that certain heuristics have better performance and are more effective than others in keeping the network connected and functional. However, the proposed heuristics do not consider the possibility of actively removing existing links in order to form new links [12].

In this paper, we propose several new link reconfiguration heuristics for 3D FSO networks. Our heuristics include the possibility of removing current working links in the network to form new links. We present analytical discussion and simulation results to determine the performance of the proposed heuristics. The overall performance of the heuristics is evaluated in terms of average node distance and network diameter.

The rest of the paper is organized as follows. Section 2 briefly discusses FSO technology and its different applications. Section 3 gives an overview of KLWY's 3D FSO network model, heuristics, and experimental result. The detail of our work is described in section 4, and discussion about future works is covered in section 5.

2. FSO Technology and Applications

Telecommunication technology has been evolving dramatically over the last few decades. Both the volume and the speed of information exchange have increased greatly. Current trends in multimedia communications such as voice, video, data, and images, are creating a demand for flexible networks with extremely high capacities [19]. Optical fiber with its enormous potential has established the ability to satisfy this demand. Telecommunications companies have been increasing the reach of their fiber optic networks to their customers. Besides being highly reliable, optical fiber has unlimited growth potential. It offers a transmission medium with Terabit per second (Tbps) bandwidth. Nonetheless, even with this potential, optical fiber has been costly in installations. Building infrastructure for an optical fiber network requires laying underground cable, which is usually very costly and time consuming. FSO technology has emerged because of this reason.

FSO uses beams of light to provide optical connections that can transmit images, videos, voice, and data. FSO has found applications in many different industries from commercial to scientific and military. For private corporate networks, wireless optics systems eliminate the recurring cost of leased lines while still providing a very high bandwidth link between sites. For temporary network connectivity needs, such as at exhibitions, conventions, sporting events, or disaster recovery, high bandwidth links can be easily and quickly provided using portable FSO systems. Furthermore, wireless optics systems can also be used as high-speed wireless backup for fiber optic cable and as "Last Mile" solutions, connecting customer sites to fiber backbones [2, 19].

Recent progress in space communication technology has proven the undisputable future of FSO. The Lunar Laser Communication Demonstration (LLCD) mission conducted by NASA in 2013 has revealed the possibility of expanding broadband capabilities in space using laser communication. LLCD demonstrated a record-breaking data download and upload speed to the moon at 622 Megabit per second (Mbps) and 20 Mbps respectively. NASA’s next mission for laser communication in space will be the Laser Communications Relay Demonstration (LRCD). LRCD is expected to demonstrate the ability to relay data at the rate of over one billion bits per second between two Earth stations using a satellite in geosynchronous orbit [1].

Despite the strengths and progress mentioned previously, FSO also has some weaknesses. FSO is essentially a LOS technology using free space as its medium. Because of the nature of its transmission medium, an FSO connection has potential disturbances such as rain, fog, physical obstructions, scintillation, beam wander, building’s movement, and seismic activities [7, 20, 26, 23]. In space communication and other long distance connections, the challenge involves pointing a very narrow laser beam accurately to the receiving device and keeping the LOS free of any physical obstruction [13]. When there is a physical obstruction in the path of the laser beam, the connection is completely lost.

3. Related Works

In 2013, Kosumo et al. proposed an approach to address link failures in a reconfigurable 3D FSO network, where transceivers are capable of realignment to create new connections with other transceivers in the network. This capability allows the network to be much more flexible. When there is a link failure in the network, transceivers can realign themselves to form new connections, keeping the network functional and connected. Following is an overview of KLWY’s 3D FSO network model and their reconfiguration heuristics.

3.1. KLWY’s 3D FSO Network Model

KLWY’s model is designed based on the well-studied mesh network topology model [5, 6, 10]. It is assumed that the network is made up of different nodes that are linked together. All the nodes are FSO devices. Thus the network model is classified as a homogeneous *n x n x n* mesh network topology. Each node in the network is limited to 6 connections, i.e. each node can connect up to 6 other nodes in the network.

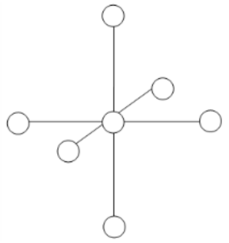


Figure 3.1 a node can have up to 6 connections [12]

The reconfiguration strategy for 2D n x n mesh networks was studied by Lee and Young in 2004 [17, 18]. In order to study the proposed 3D model, Kosumo et al. divided possible links into three different categories, type I, II, and III links. Type I links connect all the transceivers in the 3D FSO network before any reconfiguration take place. A node can connect with other node one hop away on any one of its axes through type I link. There can be up to 6 different type I links for a single node.

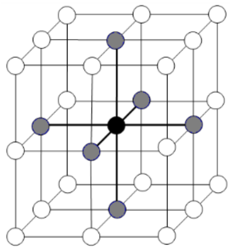


Figure 3.2 Type I link [12]

Type II link can be formed by connecting two diagonal nodes on a plane. In other words, a type II link connects a node with another node that is one hop away on two of its axes. A node has 12 different possible type II links.

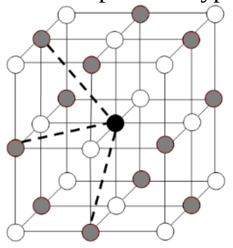


Figure 3.3 Type II link [12]

The last type is type III link. Type III link is a resulting diagonal link formed by two nodes that are located on different planes diagonally. Type III link connects a node with another node that is one hop away on all of its three axes. A node has 8 different possible type III links.

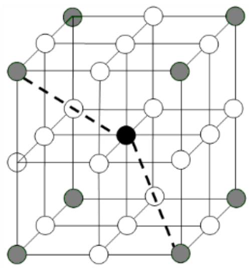


Figure 3.4 Type III link [12]

3.2. KLWY's Heuristics and Experimental Results

3.2.1. Link Reconfiguration Heuristics

Kosumo et al. offered 7 different possible heuristics to reconfigure their network after a link failure. Based on the model described in the previous section, when there is a link failure, a certain type of link is reformed based on the heuristic algorithm used. Every time a link is broken, there are two more nodes that have less than the maximum number of connection allowed. The heuristics then will try to reconnect either one or both of these nodes to other free nodes in the network. The 7 reconfiguration heuristics are summarized below,

H0: No reconnection after link failures for both nodes
H1: Attempt to reconnect only one node with a type II link
H2: Attempt to reconnect each node with a type II link
H3: Attempt to reconnect only one node with a type III link
H4: Attempt to reconnect each node with a type III link
H5: Attempt to reconnect only one node with a type II or type III link
H6: Attempt to reconnect each node with a type II or type III link
H7: Attempt to reconnect one node with a Type II link, and the other node with a type III link

For link reconfiguration using one node (H1, H3, and H5), the procedure checks all the neighboring nodes of the first node of the pair with broken link. If there is a diagonal node with less then maximum number of connections, then a new diagonal link is established. If there is no neighboring node with free connection, then the procedure checks the second node of the pair with broken link to see if any of its neighbors have less than maximum number of connections. Again if there is a node with free connection, a new link is established [12].

For link reconfiguration using two nodes (H2, H4, H6, and H7), both nodes of the pair with broken link are checked. If any of their neighbors have less than maximum number of connections, a new diagonal link is formed [12].

3.2.2. Experimental Results

Kosumo et al. measured the performance of their heuristics based on the network diameter and the average node distance of the network. Data gather from their experiment are tabulated in the following graphs,

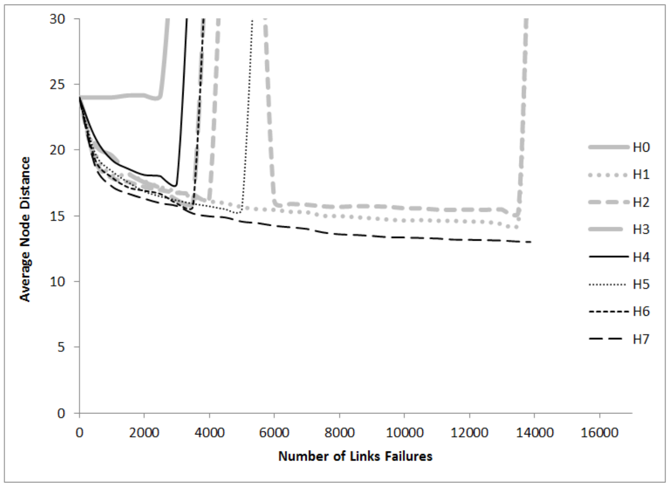


Figure 3.5 Average Node Distances vs. Link Failures Graph [12]

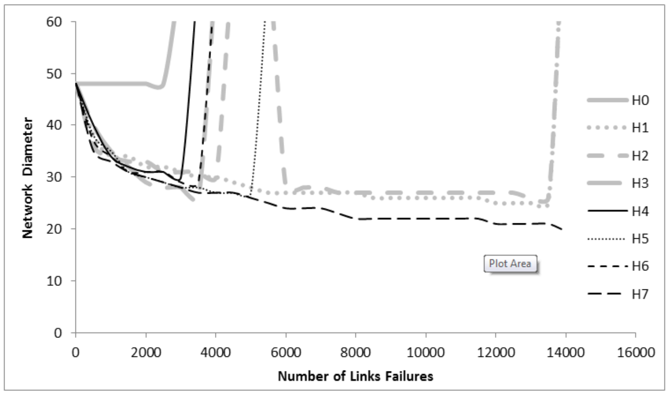


Figure 3.6 Network Diameters vs. Links Failures Graph [12]

The main conclusion that they drew is both the average node distance and network diameter decrease over time as more diagonal links are introduced into the network to replace the broken links. However, the type of diagonal link used to reconnect free nodes does provide different outcomes. As we can observe from the graphs, heuristics H1 and H2, which use type II links only, outperformed the other heuristics. H7, which uses both type of links at the same time, had the best performance. H7 was able to keep the whole network connected while decreasing the average node distance and network diameter as the number of link failures increased. Heuristics using type III link, on the other hand, did not perform well. H3 and H4 managed to decrease both the average node distance and the network diameter, but they allowed the network to become disconnected very early. In summary, the factors that contribute to the difference in performance between heuristics are the types of link used to reform broken links and the degree of connectivity of each node [12]

4. Reconfiguration Heuristics Using Active Link Removal

In this paper, we extend the work of Kosumo et al., which was done last year [12]. They introduced the possibility of reconfiguring FSO network after links failures, focused on a 3 dimensional *n* x *n x n* mesh network. Our research utilizes the same 3D FSO network model but focus on active link removal. Through the study of different shortest path algorithms and reconfigurable network models [4-6, 8, 9, 15, 22, 25], we come up with a different set of heuristics that take into account the possibility of removing existing links that are still working in the network to form new links.

4.1. Reconfiguration Heuristics

First, we introduce a notation system for our heuristic algorithms to it more convenience referring to them.

pi: Reconfiguration using pattern i. (1 ≤ i ≤6)
N: No removal of any existing links, use free transceivers only.
R: Remove an existing link for reconfiguration of certain pattern.

Based on this notation, the definition and description of the heuristic algorithms are given:

H0: No reconnection after link failures.
HNp1, HNp2, HNp3, HNp4, HNp5, HNp6: No removal of any existing links, using Pattern 1, 2, 3, 4, 5, 6 for reconfiguration.
HRp1, HRp2, HRp3, HRp4, HRp5, HRp6: Removing an existing link if necessary, using Pattern 1, 2, 3, 4, 5, 6 for reconfiguration.

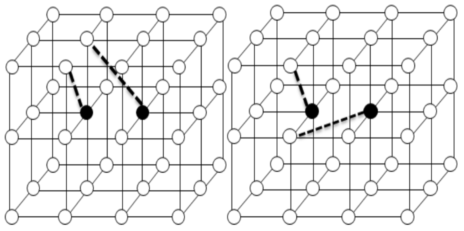


Figure 4.1 Examples of pattern 1

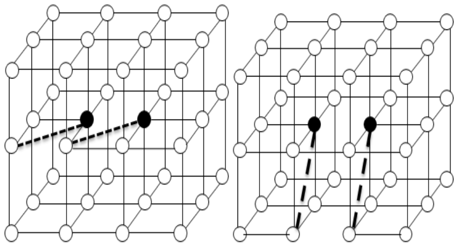


Figure 4.2 Example of patterns 2 & 3

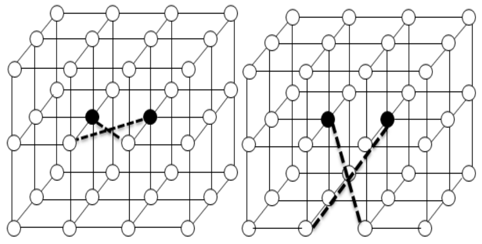


Figure 4.3 Example of patterns 4 & 5

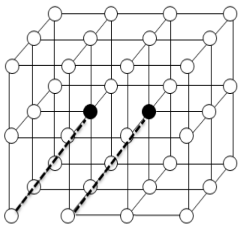


Figure 4.4 Example of pattern 6

These 6 different patterns show which link is removed based on the heuristics we use. In order to guarantee that both black nodes will be reconnected via a type I, II, or III link, we use these patterns to pick a suitable link for removal.

Heuristic H0 is the simplest among all above algorithms, and it also offers the worst results. Since H0 does nothing after link failures, the network very quickly becomes disconnected after a certain number of link failures. The only reason it is included in our study is because we can use it as a control group to compare the results of other reconfiguration algorithms.

We believe that with stronger constraints on the link patterns, better performance can be achieved. An important part of our study is to better understand and compare the impact of the proposed heuristic algorithms through real data. In our experiment, a multi-threaded simulation harness is designed and implemented. In the simulation a relatively large network (n = 10) is used and a given number of links are randomly chosen in the network to fail. All the algorithms described previously are implemented and applied to the network. The resultant average node distance and network diameter for each algorithm are recorded and compared.

4.2. Simulation Environment and Parameters

4.2.1. Simulation Environment

OS: Window 7 Ultimate
Processor: Intel Core i5-3210M CPU @ 2.5GHz
Programming language: Java
IDE: Eclipse Java EE IDE, Juno Release
Threads: Multi-threaded

4.2.2. Simulation parameters:

Mesh size	(n = 10): 10 * 10 * 10
Number of nodes	$V = n^3$: 1000
Initial number of links	$E = 3n^2(n-1)$

Statistical interval: 200 links failures
Network performance parameters: average node distance and network diameter

4.3. Simulation Results

The average node distance matrix for all algorithms and various numbers of failed links is shown in Table 4.1 and the network diameter in Table 4.2. The algorithm H7 in [12] and the baseline without any reconfiguration are also run for the sake of comparison. The overall best algorithms from the two groups of proposed algorithms are portrayed side by side with H0 and H7 in Figure 4.5 and Figure 4.6.

From Table 4.2, it can be observed that with increasing number of failed links, the network soon become un-connected, whereas several algorithms can maintain the connectivity throughout the whole simulation. In Table 4.1 and Table 4.2, we notice some non-removal algorithms. Although reduce the average node distance with a small fraction of broken links, these algorithms tend to lead to the network being disconnected when the number of link failures is higher. On the contrary, the removal-based algorithms, even with the same patterns, can defer the appearance of disconnection in the network.

The algorithms HNp1 and HRp1 are selected for their overall better performance and drawn against H0 and H7 in Figure 4.5 and Figure 4.6. In the figures, it is easy to observe the intersection of HNp1 and HRp1. In other words, it shows that as we have predicted, the removal based algorithms perform better when the link failures are relatively sparse. This can also be verified with the data of the other algorithms in the tables. In the same figures, it is also shown that the proposed algorithms outperform the existing algorithm H7 in terms of the average node distance.

Failure	H_0	$H7$	H_{Np1}	H_{Np2}	H_{Np3}	H_{Np4}	H_{Np5}	H_{Np6}	H_{Rp1}	H_{Rp2}	H_{Rp3}	H_{Rp4}	H_{Rp6}	H_{Rp6}
100	9.923	9.446	9.564	9.588	9.623	9.643	9.548	9.499	9.211	9.396	9.363	9.295	9.068	9.417
300	9.968	9.047	9.178	9.236	9.306	9.394	9.086	9.079	8.957	9.220	9.176	9.041	8.755	9.193
500	10.085	8.925	9.034	9.083	9.189	9.169	8.930	8.976	8.883	9.294	9.251	8.908	8.775	9.287
700	∞	8.866	8.949	9.023	9.125	9.132	∞	∞	8.948	9.413	9.424	8.906	8.961	9.466
900	∞	8.927	8.971	9.030	9.077	9.096	∞	∞	9.090	9.600	9.707	9.040	9.202	9.678
1100	∞	8.986	8.965	9.049	∞	9.193	∞	∞	9.321	9.638	9.809	9.190	9.481	9.870
1300	∞	9.090	9.010	∞	∞	∞	∞	∞	9.380	9.984	∞	9.364	∞	10.105
1500	∞	9.150	8.984	∞	∞	∞	∞	∞	9.483	9.892	∞	9.578	∞	10.347
1700	∞	9.230	9.042	∞	∞	∞	∞	∞	9.611	10.001	∞	9.755	∞	10.716
1900	∞	9.374	9.185	∞	∞	∞	∞	∞	9.762	10.054	∞	9.998	∞	∞
2100	∞	9.435	9.255	∞	∞	∞	∞	∞	9.860	10.277	∞	10.308	∞	∞
2300	∞	9.526	9.283	∞	∞	∞	∞	∞	10.079	10.362	∞	10.626	∞	∞
2500	∞	9.645	9.395	∞	∞	∞	∞	∞	10.204	11.645	∞	11.624	∞	∞
2700	∞	9.677	9.483	∞	∞	∞	∞	∞	10.273	∞	∞	∞	∞	∞

Table 4.1 Comparison of Average Node Distances under Link Failures for Different Heuristics

Failure	H_0	$H7$	H_{Np1}	H_{Np2}	H_{Np3}	H_{Np4}	H_{Np5}	H_{Np6}	H_{Rp1}	H_{Rp2}	H_{Rp3}	H_{Rp4}	H_{Rp6}	H_{Rp6}
100	27.0	24.561	25.828	25.243	25.828	25.828	25.828	25.732	23.485	25.414	24.657	24.071	23.196	25.196
300	27.0	22.707	23.485	24.071	24.071	24.071	23.464	23.196	22.899	24.071	23.485	23.485	21.928	23.196
500	27.0	21.610	22.899	22.899	23.485	22.899	21.392	21.928	22.314	23.485	23.485	21.728	20.660	22.196
700	∞	20.757	22.314	22.314	22.899	22.899	∞	∞	21.728	22.971	22.899	21.314	19.660	22.196
900	∞	20.757	22.314	22.314	22.314	22.314	∞	∞	21.314	23.485	22.899	21.314	19.856	21.856
1100	∞	20.364	21.899	21.730	∞	22.314	∞	∞	22.142	23.728	23.385	20.971	20.856	21.856
1300	∞	21.560	21.728	∞	∞	∞	∞	∞	21.142	24.213	∞	21.142	∞	20.856
1500	∞	20.535	21.142	∞	∞	∞	∞	∞	22.799	22.385	∞	21.314	∞	23.053
1700	∞	21.317	21.971	∞	∞	∞	∞	∞	21.385	22.799	∞	22.385	∞	23.517
1900	∞	20.828	20.799	∞	∞	∞	∞	∞	23.213	24.042	∞	22.385	∞	∞
2100	∞	19.924	21.971	∞	∞	∞	∞	∞	22.799	23.799	∞	22.213	∞	∞
2300	∞	20.364	21.799	∞	∞	∞	∞	∞	22.213	23.627	∞	22.799	∞	∞
2500	∞	20.560	20.385	∞	∞	∞	∞	∞	22.627	25.042	∞	23.627	∞	∞
2700	∞	20.292	20.799	∞	∞	∞	∞	∞	23.627	∞	∞	∞	∞	∞

Table 4.2 Comparison of Network Diameters under Link Failures for Different Heuristics

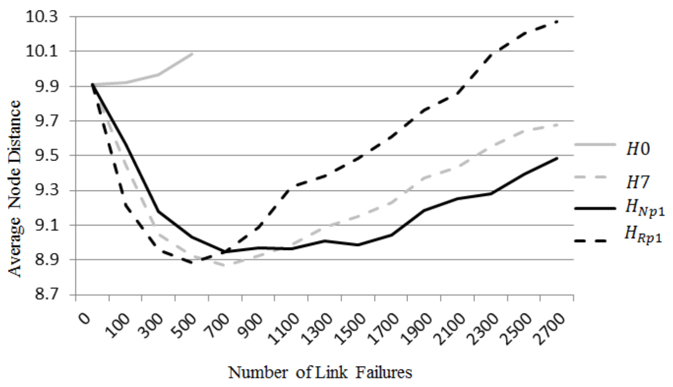


Figure 4.5 Avg. Node Distances vs. Links Failures Graph

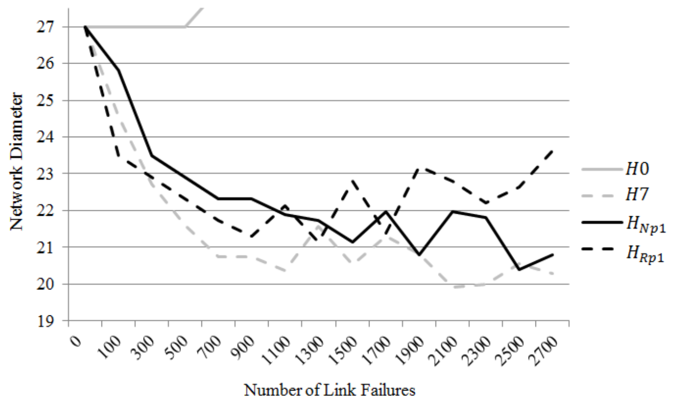


Figure 4.6 Network Diameters vs. Link Failures Graph

In our simulation, it should be noted that we choose the best algorithm in terms of average network distance when the connectivity of the network can be maintained over the entire range of link breakage. The reason we evaluate the algorithms in this way is because we would like a globally superior algorithm without the knowledge of actual number of link breakages. In a real FSO network, however, we may have a better estimation on the range of how many broken links. Thus, a more suitable criterion can be used to pick the optimal algorithm. For example, if we know the breakage ratio is less than 10%. Then from the simulation result, HRp2 instead of HRp1 can be the algorithm of choice, since HRp2 outperforms HRp1 in this range.

A point that may be confusing is the meaning of "trend" of data with varying number of failed links. The curves are drawn in the figures to enhance the visibility but the trend itself does not have a practical meaning. The network will be operated with some number of failed links and it is not practical to assume that the specific number is known at run time because our network can be of great scale and distributed in nature. The best approach to reconfigure the network is perhaps to come up with an estimation of the number of link failures and then pick the appropriate reconfiguration algorithms. It seems from Figure 4.5 that the reconfigured average node distance first "dropped" and then "rose" as more links in the network failed. Nonetheless, that does not necessarily mean we can pick the network operation point to achieve the best performance. This is simply because we do not have the global control of the whole network. We usually do not know exactly how many links in the network will fail or where these failures will be.

5. Conclusion and Future Work

Our research introduces the possibility of reconfiguring a 3D FSO network using active removal method after link failures. In this study, we proposed two groups of 3D FSO network reconfiguration algorithms based on patterns. One group of algorithms is opportunistic in a sense that they try to establish the predefined link pattern only if there are free nodes, which are available to form new link. This group of algorithms does not remove existing links even when it is necessary in order to form new links, i.e. there are not enough free nodes to actually form new connections. The other group of reconfiguration algorithms, however, is more aggressive in forming new links for the network. This group of algorithm actively removes existing link in the network in order to form new links and maximize the total number of connection in the network. A simulated environment was used to test the effectiveness of the proposed algorithms.

We collected the simulated results and computed the impact of different link reconfigurations on the overall network performance in terms of average node distance and network diameter. When the link breakage is expected to be sparse in the network, the removal-based algorithms in general work better than the non-removal ones. From the simulation result, HRp1 should be adopted in the long run, since it has the best over all performance. However, when the ratio of link failures exceeds a critical point, non-removal algorithm such as HNp1 might work better. In practice, it is FSO networks may operate on a large scale, and due to its distributed nature, the global link failures data may not readily available to a central control module. Thus it is not possible to pick the optimal algorithm for link reconfiguration. In this case, it would be desirable to first adopt a proper algorithm based on estimation of network variance. Perhaps when the network is in operation, local nodes can maintain a history of link failures. This data can then be aggregated and analyzed for a probability distribution function of number of link failures over time. The most suitable reconfiguration algorithm for the specific network condition can then be decided by using the knowledge from our simulated results.

Further study can be done on this topic. Under our current assumption of the network model, it is possible to add new patterns to the link removal algorithms. Existing patterns can also be combined, possibly forming a sequence of preferential patterns based on the number of free/available neighbor nodes. Instead of measuring performance of different heuristics using average node distance and network diameter, it is also feasible to use another parameter. Our current simulation model does not take into account the cost of each link in the network. All links in our network have their costs associated with the distances between nodes. Different costs for each link might be implemented in the model, and another performance measurement might be used.

Another potential direction for future researches is to analyze the performance intersection point of non-removal algorithms and removal-based algorithms. This intersection point, if can be found, may be independent of the network size (i.e. not a function of n) and can serve as an indication of sparseness of the link failure of the network. It is not only useful for selecting reconfiguration algorithms but can also provide directions on the design of the network topology in the first place. The case where links in the network fail progressively can also be investigated. If a global reconfiguration can be performed in a network, more comprehensive algorithms should be available to attain overall optimal performance. The ultimate goal would be to achieve some theoretical analysis of the network topology change under link failures and various reconfiguration algorithms. The analysis can be started on smaller networks and then be scaled up by composition.

REFERENCES

[1] "Laser Demonstration Reveals Bright Future for Space Communication." *Defense & Aerospace Week*, 38, 2014.

[2] J. S. Beasley, *Networking, 2nd Editon*. Upper Saddle River, NJ: Pearson Education, Inc. 2009.

[3] Y. Ben-Asher, D. Peleg, and A. Schuster, The complexity of reconfiguring networks models, *Information and Computation, 121,* pp. 41-58, 1992.

[4] Y. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster, The Power of Reconfiguration, *Journal of Parallel and Distributed Computing, 13*(2), pp.139-153, 1991.

[5] V. Bokka, H. Gurla, S. Olariu, and J. L. Schwing, Constant-Time Convexity Problems on Reconfigurable Meshes**,** *Journal of Parallel and Distributed Computing, 27*(1), pp. 86-99, 1995.

[6] K. Bondalapati and V. Prasanna, Reconfigurable Meshes: Theory and Practice**,** *Reconfigurable Architectures Workshop, International Paralllel Processing Symposium, 76,* pp.50-53, 1997.

[7] V. Brazda, V. Schejbal, and O. Fiser, "Rain impact on FSO link attenuation based on theory and measurement," *Antennas and Propagation (EUCAP), 2012 6th European Conference on,* pp.1239, 1243, 26-30, March 2012.

[8] E. W. Dijkstra, A note on two problems in connection with graphs, *Numerische Mathematik*, vol.1, pp.269-271, 1959.

[9] S. E. Dreyfus, An appraisal of some shortest-path algorithms, *Operations Research*, vol.17, no.3, pp.395-412, 1969.

[10] G. Ellinas, Routing and Restoration Architectures in Mesh Optical Networks. *SPIE Optical Networks Magazine,* vol.4, no.1, pp.91-106, 2003.

[11] J. M. Kahn, R. H. Katz, and K. S. J. Pister, Next century challenges: Mobile networking for "smartdust", *Proc.ACM/IEEE International Conference on Mobile Computing and Networking,* pp.271-278, 1999.

[12] J. Kosumo, D. Luong, J. Wong, and G. Young, "Heuristic for Reconfigurable Three Dimensional Free Space Optical Networks," *Proceedings of the 2004 International Conference on Parallel and Distributed Processing Techniques and Applications,* pp.423-429, vol. 2, July 24 2013.

[13] R. Lincks, "NASA develops and integrates first laser communications system," *Satellite Today* 12(55), 2013.

[14] J. Llorca, A. Desai, U. Vishkin, C. Davis, and S. Milner, Reconfigurable optical wireless sensor networks. *Optics in Atmospheric Propagation and Adaptive Systems VI, 5237, pp.*136-146, 2004.

[15] M. H. Macgregor and W. D. Grover, Optimized k-shortest-paths algorithm for facility restoration. *Software–Practice and Experience,* vol.24, no.9, pp.823-834, 1994.

[16] W. Mao and J. M. Kahn, Free-space Heterchronous Imaging Reception of Multiple Optical Signals. *IEEE Transactions on Communications,* vol.52, pp.269-279, 2004.

[17] T. Lee and G. Young, "Multipath Routing in Reconfigurable Free Space Optics Networks," Proceedings of the 2007 International Conference on Parallel and Distributed Processing Techniques and Applications, pp.817-821, June 2007.

[18] T. Lee and G. Young, "Routing in Reconfigurable Free Space Optics Network," Proceedings of the 2004 International Conference on Parallel and Distributed Processing Techniques and Applications, pp.946-952, 2004.

[19] A. S. Tanenbaum, *Computer Networks, 4th Edition*, Upper Saddle River, NJ: Prentice Hall, 2003.

[20] M. Tatarko, L. Ovsenik, and J. Turan, "Availability and reliability of FSO links estimation from measured fog parameters," *MIPRO, 2012 Proceedings of the 35th International Convention* , pp.192, 195, 21-25, May 2012.

[21] A. Vavoulas, H. G. Sandalidis, and D. Varoutas, "Weather effects on FSO network connectivity," *Optical Communications and Networking, IEEE/OSA Journal of* , vol.4, no.10, pp.734, 740, Oct. 2012.

[22] B. A. Warneke, M. D. Scott, B. S. Leibowitz, L. Zhou, C. L. Bellew, J. A. Chediak, J. M. Kahn, B. E. Boser, and K. S. J. Pister, Autonomous 16mm3 Solar-powered Node for distributed wireless sensor networks, *Proc. IEEE Sensors*, pp.1510-1515, 2002.

[23] H. A. Willebrand and S. G. Baksheesh, *Free-Space Optics: Enabling Optical Connectivity in Today's Networks,* First Edition, Sams, 2001.

[24] S. G. Wilson, M. Brandt-Pearce, Q. Cao, and J. Leveque, Free-space optical MIMO transmission with Q-ary PPM. *IEEE Trans. Communication,* vol.53, no.8, pp.1402-1412, 2005

[25] D. Xu, Y. Chen, Y. Xiong, C. Qiao, and X. He, "On finding disjoint paths in single and dual link cost networks," *In Proc. IEEE Infocom,* pp.53-54, 2004.

[26] S. A. Zabidi, W. A. Khateeb, M. R. Islam, A. W. Naji, "The effect of weather on free space optics communication (FSO) under tropical weather conditions and a proposed setup for measurement," *Computer and Communication Engineering (ICCCE), 2010 International Conference on* , pp.1,5, 11-12, May 2010.

SESSION

MATHEMATICAL MODELING AND PROBLEM SOLVING, MPS

Chair(s)

Prof. Minoru Ito
Nara Institute of Science and Technology
Japan

Accelaration of Poisson Corrupted Image Restoration with Loopy Belief Propagation

Hayaru SHOUNO[1]
[1] Graduate School of Informatics and Engineering, University of Electro-Communications, Chofugaoka 1-5-1, Chofu, JAPAN

Abstract— *We treat acceleration of an image restoration throughout Poisson noise channels. Previously, we proposed a image restoration method by use of Expectation Maximization (EM) algorithm[1]. The method requires calculation of inverse of the accuracy matrix, which requires* $O(M^3)$ *computational cost where* M *stands for the number of pixels, in order to obtain the posterior mean of some statistics value in each iteration. For reducing the calculation cost, we apply "loopy belief propagation(LBP)" algorithm into our method for the calculation of the marginal posterior means to substitute the posterior mean required in the EM algorithm. As the result, we can accelerate the previous algorithm over* 10 *times faster in the* 80×80 *size image.*

Keywords: Poisson image restoration, Latent variational method, Loopy Belief Propagation

1. Introduction

The image restoration problem in the field of digital image processing, is an important in the meaning of the pre-processing of image analysis instrumentation. The Poisson noise corruption process appears in the low contrast object observation such like night photograph, and some kind of computed tomography such like positron emission tomography (PET). We proposed a Poisson noised image restoration in the previous work[1]. In the method, we apply a Bayesian approach to the problem by use of the expectation maximization (EM) algorithm[2][3], which is an iterative type inference algorithm. In the algorithm, we confirmed the success of inference of the hyper-parameter, which controls the strength of the prior knowledge in the Bayesian method, as well as the pixel values. However, our algorithm requires the calculation of the inverse of the accuracy matrix in order to obtain some posterior mean of statistical values. The calculation of the inverse requires $O(M^3)$ computational cost, where M means the total number of the pixels. Thus, our algorithm is hard apply the large scale image restoration.

On the other hand, in the field of image/signal processing, the loopy belief propagation (LBP) algorithm is applied to infer some image restoration problems[4][5][6][7][8]. Even the LBP algorithm is an approximation inference method to obtain some marginal posterior mean, the restoration performance looks good with the appropriate hyper-parameter estimation.

In our previous method, the EM algorithm requires posterior mean. In our new method, we propose the approximation of the posterior mean as the marginal posterior mean. Applying the marginal posterior mean, we might reduce the calculation cost by use of the LBP algorithm. In this study, we investigate the calculation efficacy of the Poisson image restoration by use of the LBP algorithm for the approximation of the posterior mean.

The source code in this paper would be appeared in `http://bit.ly/1ktgEDM`.

2. Formulation

In the formulation, the basic idea is identical to our previous work[1]. Our method is based on the Bayesian approach, so that, at first, we formulate the Bayesian framework, which is consists of image observation process and prior probability.

2.1 Image Observation process

Let us consider λ_i as the statistical parameter of the Poisson process where i stands for the pixel position index, and z_i as its random variable, we can describe the observation probability as

$$p(z_i \mid \lambda_i) = \frac{(\lambda_i)^{z_i}}{z_i!} \exp(-\lambda_i). \tag{1}$$

Considering the Poisson process, Watanabe *et al.* treat the corruption process as a Bernoulli process, which counts the number of on-off event in the proper time bins[9]. Thus, we can translate eq.(1) as the binomial distribution form:

$$p(z_i \mid \rho_i) = \binom{K}{z_i} (\rho_i)^{z_i} (1-\rho_i)^{K-z_i}, \tag{2}$$

where $\lambda_i = K\rho_i$. In this formulation, we can confirm the eq.(2) converges to the Poisson distribution eq.(1) under the condition $K \to \infty$.

The parameter ρ_i in the eq.(2) is a non-negative parameter, which is just hard to treat for us. Thus, we introduce the logit transform into the parameter ρ_i, that is:

$$x_i = \frac{1}{2}\ln\frac{\rho_i}{1-\rho_i}, \tag{3}$$

and obtain the conditional probability for the condition x_m as

$$p(z_i|x_i) = \binom{K}{z_i}\exp((2z_i - K)x_i - K\ln 2\cosh x_i). \tag{4}$$

Hence, the image corruption process can be interpreted as observing the z_i under the condition of x_i.

2.2 Prior probability

Introducing the Bayesian inference requires several prior probability for the image. In this study, we assume some kinds of Gaussian Markov random field (GMRF)[6]. Usually, we define the GMRF as the sum of neighborhood differential square of parameters $\sum_{(i,j)}(x_i - x_j)^2$ where x_i and x_j are neighborhood parameters. The energy function and the prior probability for the GMRF can be described as following:

$$H_{\text{pri}}(\boldsymbol{x};\alpha,h) = \frac{\alpha}{2}\sum_{(i,j)}(x_i - x_j)^2 + \frac{h}{2}\sum_i x_i^2 \tag{5}$$

$$= \frac{1}{2}\boldsymbol{x}^{\text{t}}(\alpha\Lambda + hI)\boldsymbol{x} \tag{6}$$

$$p(\boldsymbol{x}|\alpha,h) = \frac{1}{Z(\alpha,h)}\exp\left(-H_{\text{pri}}(\boldsymbol{x};\alpha,h)\right), \tag{7}$$

$$Z(\alpha,h) = \int d\boldsymbol{x}\exp(-H_{\text{pri}}(\boldsymbol{x};\alpha,h)) \tag{8}$$

where (m,n) means the neighborhood pixel indices, and I means the identical matrix. In the eq.(5), the first term means the GMRF part and the second means the Gaussian prior for the zero-center value for stable calculation.

2.3 Posterior approximation

Introducing the latent variable approximation for the eq.(4), we can derive the upper limit of the observation process[10][9]. Introducing the variational parameter ξ_m, the term $\log 2\cosh x_m$ can be evaluated as

$$\ln 2\cosh x \le \frac{\tanh\xi}{2\xi}(x^2 - \xi^2) + \ln 2\cosh\xi. \tag{9}$$

Thus, the observation process can be evaluated as $p(\boldsymbol{z} \mid \boldsymbol{x}) \ge p_{\boldsymbol{\xi}}(\boldsymbol{z} \mid \boldsymbol{x})$ where

$$p_{\boldsymbol{\xi}}(\boldsymbol{z} \mid \boldsymbol{x}) = \prod_i \binom{K}{z_i}\exp\left(-\frac{1}{2}\boldsymbol{x}^{\text{T}}\Xi\boldsymbol{x} + \boldsymbol{z}^{\text{T}}\boldsymbol{x}\right)$$
$$\exp\left(\frac{1}{2}\boldsymbol{\xi}^{\text{T}}\Xi\boldsymbol{\xi} - K\sum_m \ln 2\cosh\xi_m\right), \tag{10}$$

where $\boldsymbol{z}$ means observation vector

$$\boldsymbol{z} = (2z_1 - K, \cdots, 2z_i - K, \cdots, 2z_M - K)^{\text{T}}, \tag{11}$$

$\boldsymbol{\xi}$ means the collection of latent parameter $\{\xi_m\}$, and matrix $\boldsymbol{\Xi}$ means a diagonal matrix whose components are $\{K\frac{\tanh\xi_m}{\xi_m}\}$. Thus, we approximate the $p_{\boldsymbol{\xi}}(\boldsymbol{z} \mid \boldsymbol{x})$ as the observation process, which is denoted as a Gaussian form.

From the observation (10) and the prior (7), we can derive posterior as

$$p_{\boldsymbol{\xi}}(\boldsymbol{x} \mid \boldsymbol{z},\alpha,h) \propto p_{\boldsymbol{\xi}}(\boldsymbol{z} \mid \boldsymbol{x})\, p(\boldsymbol{x} \mid \alpha,h), \tag{12}$$

and the observation can be approximated by the latent-valued form:

$$p_{\boldsymbol{\xi}}(\boldsymbol{x} \mid \boldsymbol{z},\alpha,h) \sim \mathcal{N}\big(\boldsymbol{x} \mid \boldsymbol{m}, (\Xi + \alpha\Lambda + hI)^{-1}\big), \tag{13}$$

$$\boldsymbol{m} = (\Xi + \alpha\Lambda + hI)^{-1}\boldsymbol{z}. \tag{14}$$

2.4 LBP for corrupted image restoration

In the previous work, we regarded the restoration parameters $\boldsymbol{x}^*$ as the posterior mean

$$\boldsymbol{x}^* = \langle\boldsymbol{x}\rangle = \int d\boldsymbol{x}\,\boldsymbol{x}\,p_{\boldsymbol{\xi}}(\boldsymbol{x} \mid \boldsymbol{z},\alpha,h) = \boldsymbol{m}. \tag{15}$$

In order to obtain appropriate restoration, the hyper-parameters $\theta = \{\alpha, h, \boldsymbol{\xi}\}$ should be adjusted properly. Thus, we applied EM algorithm for inferring the hyper-parameters. EM algorithm consists of two-step alternate iterations for the system that has hidden variables. Each time step of EM algorithm indicated by t consists of following two-steps:

- E-Step: Calculate Q-function that means the average of the likelihood function for the given parameter $\theta^{(t)}$:

$$\mathcal{Q}(\theta \mid \theta) = \langle\ln p(\boldsymbol{x},\boldsymbol{z} \mid \theta)\rangle_{\boldsymbol{x}|\theta^{(t)}} \tag{16}$$

- M-Step: Maximize the Q-function for θ, and the arguments are set to the next hyper-parameters $\theta^{(t+1)}$:

$$\theta^{(t+1)} = \underset{\theta}{\text{argmax}}\,\mathcal{Q}(\theta \mid \theta^{(t)}) \tag{17}$$

In each E-step, the inverse of accuracy matrix $(\Xi + \alpha^{(t)}\Lambda + h^{(t)}I)^{-1}$ is required to obtain the hyper-parameters. The computational cost for inverse of a matrix that size is $M \times M$ requires $O(M^3)$ order. Assuming the restoring image size is $L_x \times L_y$, the matrix size becomes $M = L_xL_y$.

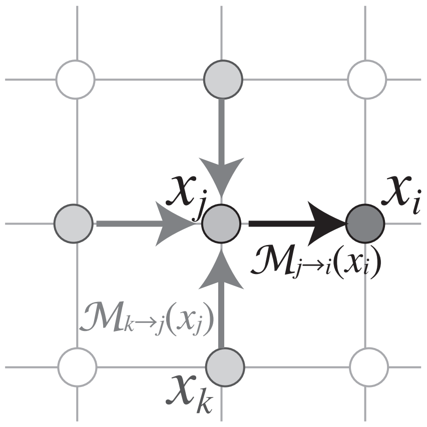


Fig. 1: Schematic diagram of message passing of the LBP: The LBP algorithm can be applied to infer the marginal posterior. Each circle shows the pixel, which has 4 nearest neighbors. For instance, considering the message from the jth unit to ith unit named $\mathcal{M}_{j\to i}(x_i)$, the message integrate the messages from the jth nearest neighbor except ith.

Thus, reduction of the calculation cost is important for the application.

In order to reduce the cost, we introduce the loopy belief propagation (LBP) in order to infer the parameters. In the manner of the Gaussian graphical model, the efficacy of the LBP were confirmed[6] [5]. Our approximated posterior, that is eq.(12), is denoted as a kind of Gaussian form, so that we can apply the LBP for the restoration. For applying LBP, we should change the evaluation of restoration value into the marginal posterior $p_{\boldsymbol{\xi}}(x_i \mid \boldsymbol{z}, \alpha, h)$ average:

$$x_i^* = \langle x_i \rangle = \int dx_i\, x_i\, p_{\boldsymbol{\xi}}(x_i \mid \boldsymbol{z}, \alpha, h). \tag{18}$$

Obtaining the marginal posterior mean, we apply a local message passing algorithm defined by LBP. For convenience, we introduce the following notations:

$$\beta_i = K\frac{\tanh \xi_i}{\xi_i}, \tag{19}$$

$$y_i = \frac{2z_i - K}{\beta_i}. \tag{20}$$

Then we obtain the observation likelihood for ith node as

$$p(y_i \mid x_i) \propto \exp\left(-\frac{\beta_i}{2}(y_i - x_i)^2\right). \tag{21}$$

The LBP algorithm is a kind of local message passing. Here, we denote the message from the jth node to the ith node as $\mathcal{M}_{j\to i}(x_i)$. Fig.1 shows the schematic diagram of the message passing. Here, considering the message $\mathcal{M}_{j\to i}(x_i)$, we should integrate the message of the jth connected units except ith. In each LBP iteration, this message passing is carried out for each connection. In the Gaussian MRF case, the message can be derived as

$$\mathcal{M}_{j\to i}(x_i) \propto \int dx_j\, p(y_j \mid x_j) \exp(-\frac{\alpha}{2}(x_i - x_j)^2 - \frac{h}{2}{x_j}^2) \prod_{k\in N(j)\backslash i} \mathcal{M}_{k\to j}(x_j), \tag{22}$$

where $N(j)$ means the collection of the connected units to the jth unit, and $N(j)\backslash i$ means the collection except ith unit. From the form of the integral in the eq.(22), we can regard the message from the jth node to the ith node as the following Gaussian

$$\mathcal{M}_{j\to i}(x_i) \propto \mathcal{N}\big(x_i \mid \mu_{j\to i}, {\gamma_{j\to i}}^{-1}\big). \tag{23}$$

Substituting the message form eq.(23) into the eq.(22), we can derive the message update rule as

$$\mu_{j\to i} = \frac{\beta_j y_j + \sum_{k\in N(j)\backslash i} \gamma_{k\to j}\mu_{k\to j}}{\beta_j + \sum_{k\in N(j)\backslash i} \gamma_{k\to j} + h} \tag{24}$$

$$\frac{1}{\gamma_{j\to i}} = \frac{1}{\alpha} + \frac{1}{\beta_j + \sum_{k\in N(j)\backslash i} \gamma_{k\to j} + h}. \tag{25}$$

The LBP requires iterations for convergence of the message values. After the convergence, the marginal posterior required for the EM algorithm can be evaluated as

$$p(x_i \mid \boldsymbol{y}, \alpha, h) \propto p(y_i \mid x_i) \prod_{j\in N(i)} \mathcal{M}_{j\to i}(x_i), \tag{26}$$

$$p(x_i, x_j \mid \boldsymbol{y}, \alpha, h) \propto p(y_i \mid x_i)p(y_j \mid x_j) \exp(-\frac{\alpha}{2}(x_i - x_j)^2 - \frac{h}{2}(x_i^2 + x_j^2)) \prod_{k\in N(i)\backslash j} \mathcal{M}_{k\to i}(x_i) \prod_{l\in N(j)\backslash i} \mathcal{M}_{l\to j}(x_j). \tag{27}$$

Thus, the Q-function for the proposing EM algorithm is

$$\begin{aligned}\mathcal{Q}(\theta \mid \theta^{(t)}) &= \langle \ln p(\boldsymbol{x}, \boldsymbol{y} \mid \theta) \rangle_{\mathrm{MP}} \\ &= \frac{1}{2}\sum_i \ln \beta_i - \sum_i \frac{\beta_i}{2} \left\langle (y_i - x_i)^2 \right\rangle_{\mathrm{MP}} \\ &+ \frac{M-1}{2}\ln\alpha - \frac{\alpha}{2}\sum_{(i,j)} \left\langle (x_i - x_j)^2 \right\rangle_{\mathrm{MP}},\end{aligned} \tag{28}$$

where $\langle\cdot\rangle_{\mathrm{MP}}$ means average over the marginal posterior eqs.(26) and (27). Deriving the eq.(28), we assume the hyper-parameter h is enough small $h/\alpha \ll 1$.

Let put them all together, the proposing EM algorithm is shown as the algorithm 1

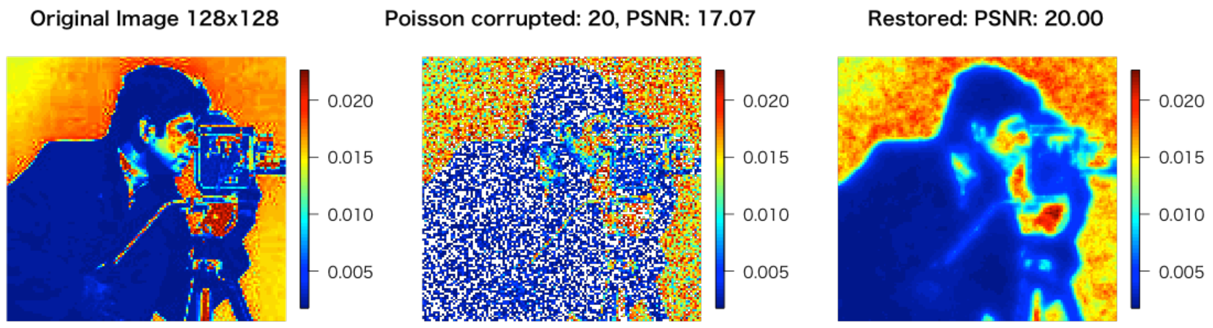


Fig. 2: Restoration sample: The left figure shows the original image cropped from the 'cameraman' with 128×128 size, and transformed linear gray-scale which range is $[2, 20]$ of Poisson parameters. The middle figure shows the Poison noise corrupted image following to the Poisson parameter assigned to the each pixel. The white pixel shows the out of the range of the original one. The right one shows the LBP restored image. The Massachusetts institute of technology has the copyright to the "cameraman" image.

3. Computer Simulation

In order to evaluate the acceleration efficacy, we measured the computational time for restoration both the LBP restoration and the previous algorithms. We adopt the following parameters in the simulations: $K = 1000, h = 10^{-5}$, and the initial hyper-parameters are $\alpha^{(0)} = 1.0$ and $\xi_i^{(0)} = 0$ respectively. The LBP convergence condition is relative error between current and previous state of the pixels $(\sum_i |m_i^{\text{LBPnew}} - m_i^{\text{LBPold}}|)/\sum |m_i^{\text{LBPold}}| < 10^{-9}$ in the meaning of the LBP iterations. Also the condition for the EM algorithm is set to $(\sum_i |m_i^{\text{EMnew}} - m_i^{\text{EMold}}|)/\sum |m_i^{\text{EMold}}| < 10^{-5}$. For these conditions, the typical number of convergence for the LBP requires below 100 iterations. And the typical number of iterations for the EM algorithm requires about 200 updates from the initial hyper-parameter state.

Controlling the image noise level of the Poisson corruption, we introduce the linear gray-level transformation from the original pixel values to the $[2, \lambda_{\text{Max}}]$. The larger the λ_{Max} is, the smaller corruption level becomes. We investigate following 5 cases of $\lambda_{\text{Max}} = \{10, 20, 40, 80, 160\}$.

The computer simulation is carried out on the Apple MacBook Pro, which has 2.7GHz Intel Core i7, 8 GBytes 1.6GHz DDR3 memory, with OS version is OS X 10.9.2. We implement the algorithm by R language which version is 3.0. For the comparison, we use the part of the image called "cameraman" which is extracted from the MATLAB R2013.

Fig.2 shows a LBP restoration sample with the size of 128×128 [pixels^2]. The left image shows the original image with gray-scale transformation of the range $[2, 20]$, that is the Poisson parameters assigned to the pixels. The maximum value of the range λ_{Max}, which is 20 in this case, controls the Poisson noise corruption level. The middle one shows the Poisson noise corrupted image. The gray pixel means that its pixel value is out of the range of the original image range $[2, 20]$. The right one shows the restored image from the middle one. We can see the restored image looks less impluse noise and smoother rather than the corrupted one.

4. Results

We compare the elapsed times for restoration between the previous our work and proposed one that is the LBP applying methods. Fig.3 shows the result. The horizontal axis shows the image scale of one side L_x, which equals to the other side L_y. Thus, the horizontal axis shows the square root of the total number of image pixels. The vertical one shows the elapsed time for restoring by use of the EM algorithm from the initial hyper-parameter state. Note that the vertical axis is applied the *log* scale. In the figure, the solid lines show the results for the LBPs, and the dashed one show the our previous work called "exact solution", which solve the inverse of accuracy matrix in each EM step. In the exact solution expressed as the dashed lines, the larger the image size is, the larger elapsed time becomes. We can also see the corruption level does not affect to the calculation cost for the previous work. On the contrary, in the LBP solutions, the calculation cost looks insensitive to the image size. Instead, the LBP solutions are affected to the corruption level, when

Algorithm 1 Poisson corrupted image restoration using EM algorithm with LBP

```
1: Set the initial hyper-parameters $\alpha^{(0)}$, $\boldsymbol{\xi}^{(0)}$, and $h$
2: Set the initial restoration image $x_i^{(0)}$
3: $t \leftarrow 0$
4: **repeat**
5:     Set $\beta_i^{(t)} = K\frac{\tanh \xi_i^{(t)}}{\xi_i^{(t)}}$, and $y_i^{(t)} = (2z_i - K)/\beta_i^{(t)}$.
6:     Carry out the LBP, where update eqs. are (24) and (25), under the given hyper-parameters $\alpha^{(t)}, \{\beta_i^{(t)}\}$.
7:     After convergence of the LBP, solve several statistics: the restoration pixel values $\{m_i\}$, those of variances $\{(\sigma_i)^2\}$, and the correlations $\{s_{ij}\}$:
       $$m_i = \frac{\beta_i^{(t)} y_i^{(t)} + \sum_{j\in N(i)} \gamma_{j\to i}\mu_{j\to i}}{\beta_i^{(t)} + h + \sum_{j\in N(i)} \gamma_{j\to i}} \quad (29)$$
       $$\sigma_i^2 = (\beta_i^{(t)} + h + \sum_{j\in N(i)} \gamma_{j\to i})^{-1}, \quad (30)$$
       $$s_{ij} = \frac{(\alpha^{(t)} - \gamma_{i\to j})(\alpha^{(t)} - \gamma_{j\to i})}{{\alpha^{(t)}}^3}. \quad (31)$$
8:     Update the hyper-parameters:
       $$\xi_i^{(t+1)} = \sqrt{{m_i}^2 + {\sigma_i}^2} \quad (32)$$
       $$\frac{1}{\alpha^{(t+1)}} = \frac{\left(\sum_{(i,j)} (m_i - m_j)^2 + \sigma_i^2 + \sigma_j^2 - 2s_{ij}\right)}{M-1}. \quad (33)$$
9:     $t \leftarrow t + 1$
10: **until** restoration image $\{m_i\}$ is converged.
11: $\boldsymbol{x}^* \leftarrow \boldsymbol{m}$
```

it becomes large, which is small λ_{Max}, the more calculation cost is required. However, in the large scale image, the LBP solutions has advantage to the exact solutions.

In order to evaluate restoration quantitatively, we introduce the peak signal noise to ratio (PSNR). The PSNR is defined as a kind of similarity between the reference image $\boldsymbol{q}^*$ and the test image $\boldsymbol{q}$ as:

$$\text{PSNR}(\boldsymbol{q}, \boldsymbol{q}^*) = 10 \log_{10} \left(\frac{\max \boldsymbol{q}^* - \min \boldsymbol{q}^*}{\text{MSE}(\boldsymbol{q}, \boldsymbol{q}^*)} \right)^2, \quad (34)$$

$$\text{MSE}(\boldsymbol{q}, \boldsymbol{q}^*) = \frac{1}{M} \sum_i (q_i - q_i^*)^2 \quad (35)$$

Fig.4 shows the PSNR between original image ρ_m and restored image with inverse logit transform. The horizontal axis shows the maximum number of the Poisson parameter λ_{Max} assigned to the original image. The large λ_{Max} means the original image shows high contrast, so that the noise corruption level is low. On the contrary, the small λ_{Max} means the noise corruption level is high. The evaluation is carried out with 10 times trials and plot with median with quantile deviation. Both the results of the LBP line and the Exact line are overlapped almost all, that is the restoration performance in the meaning of the PSNR are equivalent. In the figure, we can see the slightly improvement in high noise level around $\lambda_{\text{Max}} \sim 10$. On the contrary, in the low noise level around $\lambda_{\text{Max}} \sim 160$, all of the image qualities of LBP, exact, and corrupted one look equivalent. This result means that our restoration approach might not degrade the image quality.

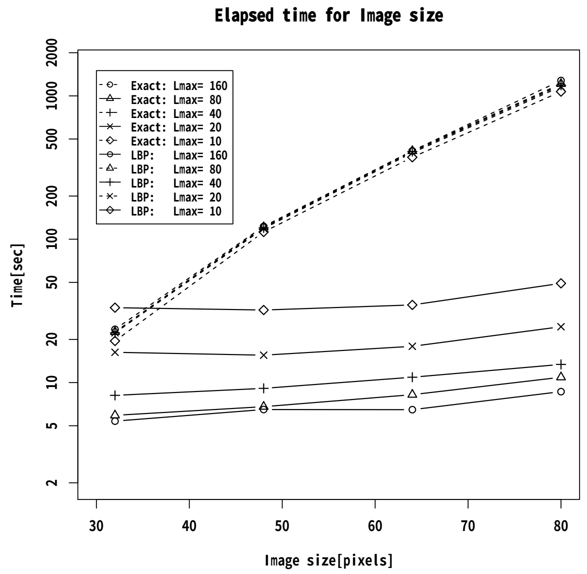


Fig. 3: Comparison of restoring times: The vertical axis shows the log-scaled elapsed time [sec] for the restoration, which means convergence of the EM algorithms. The horizontal one shows the image size $L_x (= L_y)$ [pixels], which means image size of one side. Each solid line shows the result for the LBP restoration, and dashed line line shows the result for the exact solution for the previous algorithm[1]. For both LBP and exact method, we investigate the convergence time for several noise level, which are $\lambda_{\text{Max}} = \{10, 20, 40, 80, 160\}$.

5. Summary & Conclusion

In this study, we propose an acceleration method for the Poisson corrupted image restoration with the LBP. In our image restoration framework that is based on the EM algorithm, the inverse of the accuracy matrix is required. The calculation cost of the inverse required $O(M^3)$ in general.

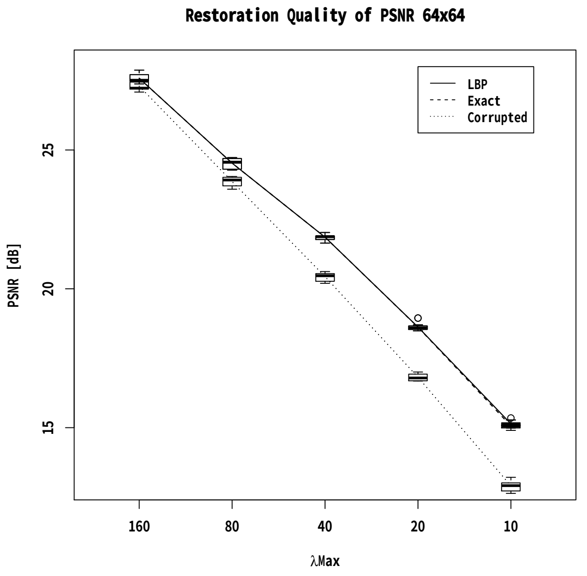


Fig. 4: Comparison of restoring quality for the 64×64 image: The vertical axis shows the peak signal to noise ration (PSNR) [dB], which means the similarity index to the original image. The horizontal one shows the Poisson parameter λ_{Max} for the image. The solid line shows the LBP result, and dashed one shows the exact solution. These two lines are almost all overlapped. The dot line shows the PSNR for the observed image.

Thus, we introduce the LBP to infer the statistics parameters. Our method approximate the Poisson corruption process as a Gaussian form, so that, we can easily derive the LBP update rule. To apply the LBP for the EM algorithm, we have to replace the posterior mean with the marginal posterior mean. Moreover, we should consider the effect of two-body interactions to infer the hyper-parameter α. Normally, the LBP only consider the single-body marginal posterior described as eq.(26). Only considering the single-body marginal posterior, the correlation of connected two units, which is denoted as s_{ij} in the eq.(31), becomes 0. This means same effect to the naive mean field approximation. Thus, the single-body marginal posterior occur the underestimation of the parameter of α. Avoiding the underestimation, we introduce the two-body marginal posterior described as eq.(27) in the hyper-parameter inference. The correlation s_{ij} update rule is derived as the eq.(31), which only requires the local message, so that the cost for the inference does not increase so much. Solving exact correlation between two units requires considering not only the connected bodies effect but also all the other bodies effect. This is the reason for the requiring the inverse of the accuracy matrix in the EM algorithm. We only consider the two-bodies effect, however, the hyper-parameter inference looks work well, and the restoration performance becomes same or more than the that of the exact solution in the previous work. Hence, we propose the LBP method is a good approximation for our Poisson corrupted image restoration framework.

Acknowledgment

This work is supported by MEXT/JSPS KAKENHI Grant number 25330285, and 26120515.

References

[1] H. Shouno and M. Okada, "Poisson observed image restoration using a latent variational approximation with gaussian mrf," in *PDPTA*. PDPTA 2013, 7 2013, pp. 201–208.

[2] A. Dempster, N. Larid, and D. B. Rubin, "Maximum likelihood estimation from incomplete data via the em algorithm," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 39, pp. 1–38, 1977.

[3] D. J. C. Mackay and C. Laboratory, "Hyperparameters: optimize, or integrate out," in *In Maximum Entropy and Bayesian Methods, Santa Barbara*. Kluwer, 1996, pp. 43–60.

[4] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[5] K. Murphy, Y. Weiss, and M. Jordan, "Loopy belief propagation for approximate inference: An empirical study," in *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, ser. UAI'99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 467–475. [Online]. Available: http://dl.acm.org/citation.cfm?id=2073796.2073849

[6] K. Tanaka, "Statistical-mechanical approach to image processing," *Journal of Physics A: Mathematical and General*, vol. 35, no. 37, pp. R81–R150, 2002. [Online]. Available: http://iopscience.iop.org/0305-4470/35/37/201/

[7] K. Tanaka, H. Shouno, M. Okada, and T. D. M., "Accuracy of the bethe approximation for hyperparameter estimation in probabilistic image processing," *Journal of Physics A: Mathematical and General*, vol. 37, pp. 8675–8695, 2004. [Online]. Available: http://iopscience.iop.org/0305-4470/37/36/007/

[8] P. Felzenszwalb and D. Huttenlocher, "Efficient belief propagation for early vision," *International journal of computer vision*, vol. 70, no. 1, pp. 41–54, 2006.

[9] K. Watanabe and M. Okada, "Estimation of varying binomial process using local variational approximation." *IEICE Transactions on Information and Systems*, p. to be appeared, 2011.

[10] J. A. Palmer, K. Kerutz-Delgado, D. P. Wipf, and B. Rao, "Variational em algorithm for non-gaussian latent variable models," in *Advances in Neural Information Processing Systems 18*, vol. 18. MIT Press, 2005, pp. 1059–1066. [Online]. Available: http://books.nips.cc/papers/files/nips18/NIPS2005_0803.pdf

Singularity Size Optimization in Data Deduplication Technique

Mie Ogiwara, Mizuki Takaya, Teruhisa Kasuya,
Itaru Koike, Toshiyuki Kinoshita

School of Computer Science, Tokyo University of Technology
1404-1 Katakura, Hachioji Tokyo, 192-0982, Japan

Abstract *Recently, massive data growth and data duplication in enterprise systems have led to the use of deduplication techniques. Since we keep multiple versions of files, there may be a large volume of mostly or exactly identical data. Deduplication is a powerful storage optimization technique that can be adopted to manage maintenance issues in data growth. We evaluated the effect of deduplication by analyzing how the singularity size affects the effect of deduplication for variable-length blocks. We clarify that the effect of deduplication is affected by the singularity size and the number of created blocks. We traced the change in the deduplication rate, which indicates the reduce ratio of file data volume, by changing the singularity size from 4 bits to 23 bits. The result shows that the optimum singularity size is 15 bits and that the deduplication rate is improved around 7 % at the optimum singularity size compared with at smaller or larger singularity size.*

Keywords *data deduplication, variable-length block, Rabin-Karp algorithm, optimum singularity size*

1. Introduction

In recent years, the volume of file data in enterprise systems has greatly increased due to the growing popularity in handling multimedia data including animation or video, etc. In these file systems, multiple versions of a file that are exactly or mostly identical might exist and deduplication techniques may be used to minimize the file data volume by eliminating redundant data. Deduplication is a file compaction technique that is commonly used in general enterprise systems by removing duplicates within and across a file. This general concept has been successfully applied to back up, virtual machine storage, and WAN replication.

Data deduplication is a process that calculates the similarity in record pairs and merges them if similarity is detected. It can reduce a huge amount of data by eliminating overlapping data (redundant data) in large-scale servers or data storage. Using deduplication, only one representative of two or more overlapping data files or the same areas of similar data is preserved, while the overlapping data is replaced with links that point to the representative data (as shown in Fig. 1.1). By replacing multiple overlapping data with links, data storage size can be largely reduced. As a result, the efficiency of data storage can be highly improved and cost in data maintenance and storage will be also decreased.

There are two types of deduplication techniques, file-level and block-level (Fig. 1.2 (a)(b)). In block-level deduplication, files are divided into several parts (each part is called a block) and any duplicate blocks are eliminated. File-level deduplication is simple and the deduplication execution time is short. High deduplication efficiency, however, cannot be achieved unless many files are exactly the same. On the other hand, by using block-level deduplication, high deduplication efficiency can be achieved since the files are not required to be strictly the same. The deduplication execution time, however, is longer due to the fact that more execution time is required for creating blocks.

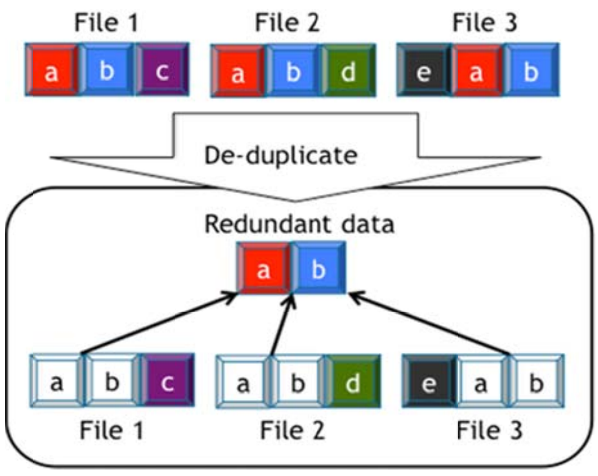
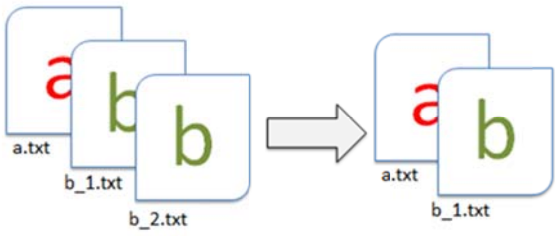
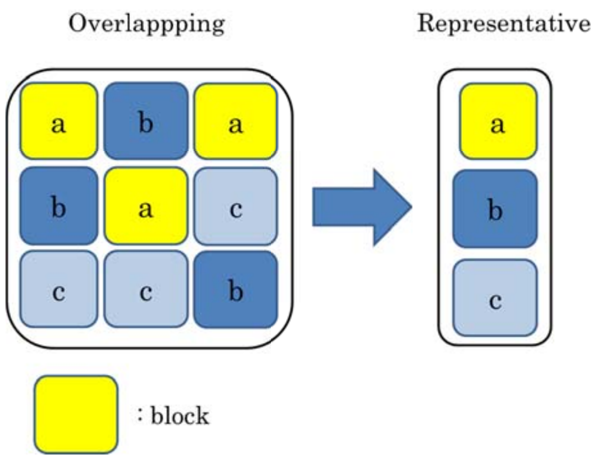


Fig. 1.1 Data deduplication



(a) File-level deduplication



(b) Block-level deduplication

Fig. 1.2 Two types of deduplication

There are two types of blocks in block-level deduplication, fixed-length block whose length is constant and variable-length block whose length can be changed. In order to efficiently generate variable-length blocks, a hash value is calculated for each candidate of the breakpoint of the block. If a particular bit-pattern (called singularity) is included in the hash value, the candidate becomes a real breakpoint of the block. If the singularity is not included in the hash value, the candidate does not become a breakpoint. The effect of deduplication can be changed by the singularity, especially the singularity size mainly affects the effect of deduplication. The purpose of this study is to analyze the optimum singularity size to maximize the effect of deduplication.

2. Related Works

In the variable-length block method, the block length can be changed, and the maximum and minimum block length is set not to generate an extremely large or small block. The effect of deduplication is also affected by this maximum / minimum block length.

The effect of deduplication in the fixed-length block method when the block length is set to 4 ~ 16 K byte is investigated in [1] and the efficiency of the variable-length method is discussed in [3]. The difference of the effect of deduplication between in the variable-length block and in the fixed-length block when the block length is larger than 4 K bytes is reported in [2] and when the block length is smaller than 4 K bytes is discussed in [4]. These studies have investigated the relationship between the block length and the effect of deduplication, however studies which have analyzed the relationship between the singularity size and the effect of deduplication are not available. In this study, we analyzed the optimum singularity size to maximize the effect of deduplication.

3. Block-level deduplication

3.1 Two types of block

We can consider two types of block, the fixed length block and the variable length block. The following example and Fig. 3.1 ~ 3.3 explains these two types. Consider an error in the sample sentence, "I am goin to Rome tomorrow." When "g" is inserted to correct the sentence, the sentence becomes "I am going to Rome tomorrow." In the fixed-length block method, since the characters after inserting "g" are shifted by one character, blocks after inserted "g" will not be recognized as duplicate blocks in the original sentence. On the other hand, in the variable-length block method, block length can be changed. Therefore, blocks after inserting "g" can

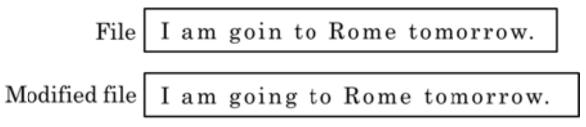


Fig. 3.1 Sample sentence

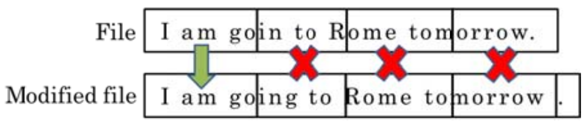


Fig. 3.2 Fixed-length block (seven- character block)

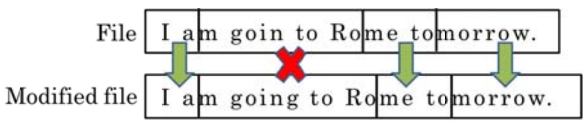


Fig. 3.3 Variable-length block (beginning at "m")

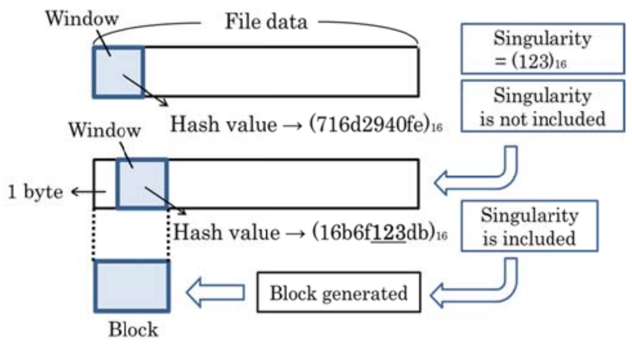


Fig. 3.4 Breakpoint search

be recognized as duplicate blocks and considered for deduplication since the block length can be adjusted to the insertion. Thus, in the variable-length block method, the effect of deduplication can be maintained even if a character has been inserted or deleted.

3.2 Variable-length block

The Rabin-Karp algorithm is used for generating a variable-length block. The following parameters are used in the algorithm.

(1) Minimum file size (default is 40 bytes)
Files that are smaller than this size will not be targeted for deduplication.
(2) Minimum block length (default is 4,000 bytes)
(3) Maximum block length (default is 16,000 bytes)
(4) Window size (default is 32 bytes)
Window is a unit that is changed to a hash value.
(5) Singularity (default is $(123)_{16}$)
When the singularity is included in the bit pattern of the hash value of the window, a block is generated at the position of the window.

For files that are larger than the minimum file size and smaller than the minimum block length, the whole file is generated as a block. However, if the files are larger than the minimum block length, a breakpoint will be determined. As shown in Fig. 3.4, in searching for the breakpoint, a hash value is first created for the window at the location of the minimum length block, and is checked up if it includes the singularity, or not. If the hash value includes the singularity, a breakpoint is found and a block is generated at the position. On the other hand, if the hash value does not include the singularity, the window is shifted one byte and the breakpoint search is repeated. If the breakpoint is not found until the maximum block length, a maximum length block is generated at this position.

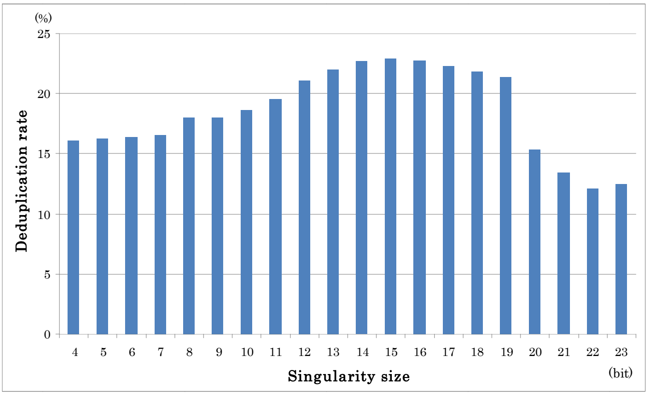


Fig. 4.1 Deduplication rate vs singularity size

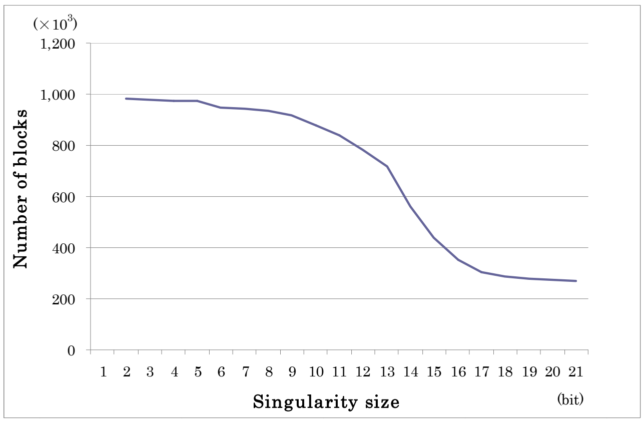


Fig. 4.2 Number of blocks vs singularity size

4. Verification of optimum singularity size

4.1 Verification experiment

As an indicator of the effect of deduplication, we use the "deduplication rate = the data size reduced by deduplication / original data size". The large deduplication rate shows better effect of deduplication. Experiment for verification is performed using a variable-length block method and testing was done on the file systems in our research lab in Tokyo University of Technology. The total size of the file system is 7G bytes and the target file system includes text data (data types are doc, xls, ppt), image data (pdf, gif, bmp, jpg), etc. The maximum / minimum block length is set to 4K bytes and 16K bytes, which are usually used in almost all deduplications [1].

4.2 Experimental Results

Fig. 4.1 and Fig. 4.2 show the relationship between the singularity size and the deduplication rate and the relationship between the singularity size and the number of blocks respectively. We can see that the highest deduplication rate achieves when the singularity size is 15 bits in Fig. 4.1 and the number of blocks decreases with the increase of singularity size in Fig. 4.2 since the singularity more hardly exists in the hash value of the window when the singularity size is larger.

When the singularity size is in range less than 15 bits, the deduplication rate increases with the increase of singularity size since more moderate length blocks that are applicable for the deduplication are generated and more effective deduplication can be performed. When the singularity size is too small, the singularity appears in almost all the hash value of the window and many blocks with around minimum block length were generated. As shown in Fig. 4.3 and Table 4.1, when the singularity size is 7 bits and 11 bits, the blocks between 4K and 5K bytes accounted for 98% and 97% of the total blocks respectively. This means that the deduplication is nearly same to the fixed-length block method and the benefit of variable-length method hardly appears.

On the other hand, when the singularity size is 15 bits, the highest deduplication rate is achieved, blocks with around minimum block length accounted for 63% of the total blocks. This ratio is considerably less than in the case of smaller singularity size. It can be considered that the more effective deduplication can be achieved by optimizing the singularity size.

Once the singularity size is in range larger than 15 bits, it is shown that the deduplication rate decreases with the increase of singularity size since longer size of singularity is hardly found in the hash value of the window and many blocks with around maximum length are possibly generated. When the singularity size is 15 bits, blocks between 15K and 16K bytes accounted for only 0.6% of the total blocks, nevertheless blocks between 15K and 16K bytes accounted for 48% and 90% of the total blocks when the singularity size is 19 bits and 23 bits respectively. The results can prove that the deduplication using the longer singularity size is almost nearer the fixed-length block method rather than the variable-length block method and the benefit of variable-length method also was hardly observed.

5. Conclusion

In this study, we clarified that the effect of the deduplication can be higher by optimizing the singularity size. It is recognized that the deduplication rate is improved around 7 % at the optimum singularity size compared with at smaller or larger singularity size.

As our future works, we will investigate the optimum singularity size for the various maximum / minimum block lengths not limiting to 4K / 16K bytes only.

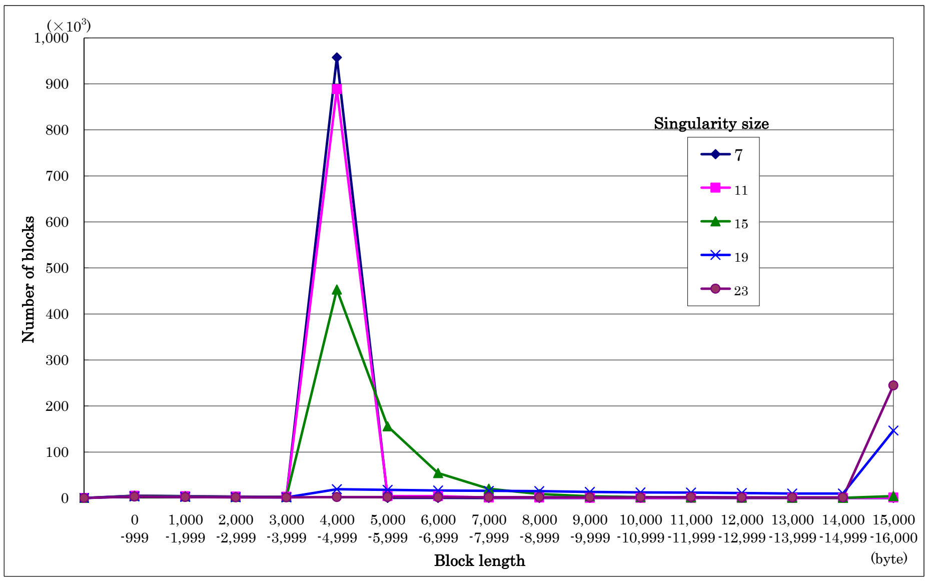


Fig. 4.3 Number of blocks vs block length

Table 4.1 Number of blocks vs singularity size

Range of block size \ Singularity size	7 (bits)	11	15	19	23
0 ～ 4,000 (byte)	14,829 (1.5%)	14,132 (1.5%)	12,306 (1.7%)	8,651 (2.8%)	7,972 (2.9%)
4,001 ～ 5,000	957,374 (98.4%)	888,619 (97.1%)	452,886 (63.1%)	19,077 (6.2%)	2,110 (0.8%)
5,001 ～15,000	1,335 (0.1%)	11,620 (1.3%)	248,069 (34.6%)	131,201 (43.0%)	15,744 (5.8%)
15,001 ～16,000	43 (0.0%)	1,079 (0.1%)	4,022 (0.6%)	146,528 (48.0%)	244,647 (90.5%)
Total	973,581	915,450	717,283	305,457	270,473

References

[1] Q. He, Z. Li, X. Zhang, “Data deduplication techniques,” Future Information Technology and Management Engineering (FITME) 2010, vol. 1, pp. 430-433, Oct. 2010

[2] C. Constantinescu, J. Glider, D. Chambliss , “Mixing Deduplication and Compression on Active Data Sets,” Data Compression Conference (DCC) 2011, pp. 393-402, March 2011

[3] A. N. Yasa, P. C. Nagesh, “Space savings and design considerations in variable length deduplication,” ACM SIGOPS Operating Systems Review, Vol. 46 Issue 3, pp. 57-64, Dec. 2012

[4] M. Noorafiza, I. Koike, H. Yamasaki, A. Rizalhasrin, T. Kinoshita, “Block Length Optimization in Data Deduplication Technique,” Proceedings of the 10th International Conference on Scientific Computing (CSC2013), pp.216-220, July 2013

Improved conformational search for protein-ligand docking based on optimal arrangement of multiple small search grids

Tomohiro Ban[1,2], Takashi Ishida[1], and Yutaka Akiyama[1,2]
[1]Department of Computer Science, Graduate School of Information Science,
Tokyo Institute of Technology, W8-76, 2-12-1 Ookayama, Meguro-ku, Tokyo 152-8550, JAPAN
[2]Education Academy of Computational Life Sciences,
Tokyo Institute of Technology, W8-93, 2-12-1 Ookayama, Meguro-ku, Tokyo 152-8550, JAPAN

Abstract— *The Glide protein-ligand docking algorithm often fails to find the correct binding mode. This is because the search process can easily fall into local minima when the search target area is widely distributed across the protein's surface and the search grid is relatively large. In this research, we propose a novel method that improves the search efficiency in such cases by dividing a single, large search grid into multiple small search grids. In addition, we propose a method to minimize the number of small grids by converting the problem into a set cover problem. We present experimental results to compare the performance of the proposed approach with that of the standard protocol under two different settings.*

Keywords: protein-ligand docking, Glide, set cover problem, conformation search

1. Introduction

The technique of protein-ligand docking aims to predict the binding mode of a protein and a small chemical compound (ligand) from their three-dimensional structures. This approach is now used in many fields, such as drug discovery and molecular biology [1] [2]. To date, various research groups from both commercial and academic organizations have developed protein-ligand docking software, such as AutoDock [3], GOLD [4], FlexX [5], and Glide[6]. In particular, Glide has demonstrated good accuracy using various benchmarks, and is recognized as one of the best docking software applications [7] [8] [9]. However, even Glide does not always return the correct binding mode. Therefore, an improvement in the accuracy of protein-ligand docking is highly desirable and would have a significant positive impact in various fields. The low prediction accuracies given by protein-ligand docking software are often caused by two substantial problems. One is the estimation of the binding free energy, and the other is the problem of searching the whole conformational space. The former problem is caused by the coarse model resolution and simplified potential energy function, which are intended to reduce the computational cost. The latter problem is a result of the huge number of conformations to be searched. In particular, this problem becomes more serious if the binding site of a target protein is unknown. This is because only a narrow region need be searched if the binding site is well known; if this is not the case, the entire protein surface must be searched. Thus, the conformational space search requires more computational resources, and this can become a serious problem.

To tackle this, several software packages, such as PocketFinder [10] and SiteMap [11], have been developed to predict the ligand binding sites. In the standard Glide docking protocol, multiple binding sites are predicted from the tertiary structure of a protein using SiteMap, and then a search grid is set to cover these predicted binding sites. Finally, only the region within the grid is searched in the Glide docking simulation process. However, even using this protocol, Glide sometimes fails to find the correct binding mode. The search easily falls into local minima if the predicted binding sites are widely distributed across the protein's surface and a large search grid is used. The search algorithm of Glide tends to intensively search narrow regions near positions that score highly in the initial search stage, and overlook good conformations far from such regions. Therefore, the search accuracy of Glide often becomes lower if a large number of binding sites are predicted over a widespread area.

In this study, we propose a method to improve the search efficiency of protein-ligand docking when many binding sites are predicted and the search grid is large. To avoid the problem of local minima, we use multiple small search grids instead of one large grid. Additionally, to minimize the number of small search grids, we translate this arrangement into a set cover problem, and successfully reduce the number of grids.

2. The Glide conformation search algorithm

The Glide search algorithm [6] is a four-part process that determines the conformation with the lowest binding free energy. In the first stage, the algorithm uses simple criteria to determine candidate positions on the protein that are likely to bind with a ligand. In the second stage, the algorithm arranges ligands at these points, and calculates their binding score using a rough score function. In the third

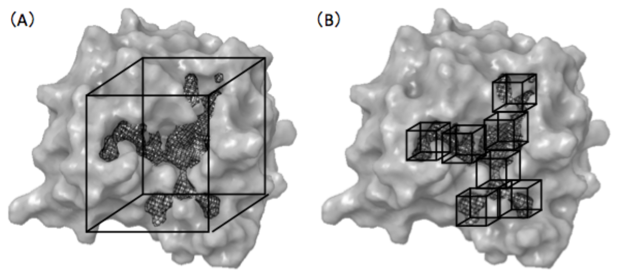


Figure 1: (A) A grid in the Glide standard protocol (B) Grids generated by the proposed method

stage, to minimize the binding free energy, the algorithm optimizes the structure of the ligand by dihedral angle rotation and rigid body transformation. In the final stage, the algorithm selects the best score conformation using a precise score function named GlideScore [6]. In particular, the second stage consists of two different processes. The first is the calculation of a GreedyScore, and the other is a refinement process. In the GreedyScore calculation process, ligands are arranged at the positions selected in the first stage, and the top 5000 conformations are selected according to their ChemScore [12]. In the refinement process, these 5000 conformations are refined by moving the center of the compound within $\pm$ 1Å and the top 400 conformations are finally selected.

The number of selected conformations is a fixed parameter, regardless of the size of a search grid. As a result, the algorithm often fails to find the correct conformation when the search target area is widely distributed over the protein surface and a large search grid is used. Of course, the parameter can be changed manually. However, the range is limited by the interface, and it is difficult to determine an appropriate value empirically.

3. The proposed method

Using the default Glide protein-ligand docking, the conformational search sometimes fails because of insufficient sampling. To solve this problem, we propose a method to improve the search efficiency by dividing a large search grid into multiple small search grids (Figure 1). For a search grid of optimal size, the Glide conformation search algorithm works well, even with the default settings, and we can generally obtain accurate conformations. Thus, in our proposed method, a large search grid is divided into multiple small grids, and then a conformational search is performed for each small grid. Finally, the output of all conformational searches is collated, and the final prediction results are selected according to the GlideScore.

Our proposed method has the clear disadvantage that the computational cost increases in proportion to the number of search grids, meaning that the cost of our method is larger

```
The algorithm of proposed method
Let $S$ be the site-points obtained by SiteMap;
Let $A$ and $B$ and $C$ be the empty sets;
While (Until $S$ is empty) do
  for (Until select the all elements of $S$) do
    Let $gc$ be the selected element of $S$;
    for (Until select the all elements of $S$) do
      Let $s$ be the selected element of $S$;
      if ($s$ is included in the grid whose $gc$ is the center) then
        Add $s$ into the set $A$;
      end if
    end for
    if (The number of $A$ is greater than the number of $B$) then
      Let $gc$ be a candidate of the center of a grid;
      Overwrite $A$ to $B$;
    end if
    Empty $A$;
  end for
  Add $gc$ into the set $C$;
  Remove the all elements of $B$ from $S$;
end while
Return $C$ as a set of the centers of grid.
```

Figure 2: Algorithm of the proposed method

than that of a standard protocol. To reduce this harmful influence, we also propose a grid arrangement method to minimize the number of search grids. We convert this grid arrangement problem into a set cover problem [13]. In the set cover problem, given a table set U made of n elements, a subset group of U expressed as S=$\{S_1,S_2, \dots ,S_l\}$, and a cost function $c : S \rightarrow Q_+$ (Q_+ is a set of positive rational numbers), we must identify the subset of S covering all elements of U with the lowest cost. In our optimal grid arrangement problem, we use the site-points obtained by SiteMap as the table set, and the site-points included on a grid whose center is one of the elements of the table set is the subset group. The cost is the number of elements of the table set included in each grid. In this way, we can convert the grid arrangement problem into a set cover problem. We use an approximate algorithm to solve this, because the set cover problem is known to be NP-hard [14]. The algorithm consists of seven steps: (i) Input the site-points obtained by SiteMap and (ii) prepare the empty set C. Next, (iii) select the highest-cost grid G and (iv) add the center of grid G to C. After that, (v) remove all of the site-points included in grid G, and (vi) repeat (iii)–(v) until S is empty. Finally, (vii) use the site-points in C as the centers of grids in the dispersion setting. Figure 2 shows the pseudo-code of this algorithm. The computational complexity is O(n^3), where n is the number of elements in the table set.

Figure 3 shows the behavior of the algorithm on a two-dimensional space. Both white and black dots represent site-points, and are elements of the table set. The black dots are selected as the center of a search grid by the proposed method, and squares represent each search grid. All of the dots are included in the union of these grids. In particular, the algorithm minimizes the number of grids. In this case, the algorithm successfully covers 30 dots with only 11 grids.

We implemented the proposed method by altering the XGlide.py python script in the Glide cross docking [15].

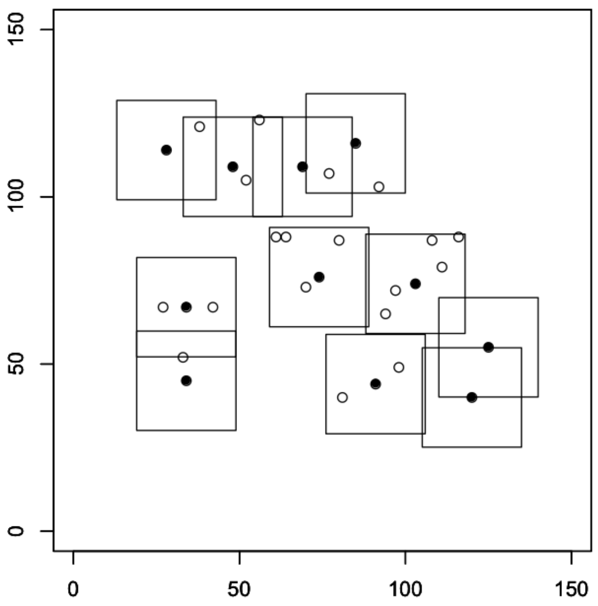


Figure 3: Example 2D grid arrangement given by the proposed method

4. Evaluation experiment

In this experiment, we confirm that the proposed method has better search efficiency than the Glide standard protocol under its default settings. We use the docking score and computation time to evaluate the search efficiency. We also compare the efficiency of the proposed method to that of the Glide standard protocol under the "heavier" setting, which makes the conformation search more onerous but more accurate. This is because a direct comparison of the proposed method and standard protocol with default settings is difficult, as the proposed method has a greater inherent computational complexity.

4.1 Dataset

We used a protein-ligand complex dataset called CCDC/Astex [16]. Because of limitations in computational power and the number of Glide software licenses, we randomly selected the following 20 proteins that did not cause errors in the docking process: 1A4G, 1AJ7, 1B9V, 1DBB, 1EJN, 1FAX, 1FKG, 1HDC, 1IBG, 1MMQ, 1QBR, 1RNE, 1TPH, 1XKB, 2DBL, 2H4N, 2TMN, 2TPI, 3ERD, 7CPA (complex structures 1GPY, 1RT2, and 4CTS were selected at first, but these were replaced by 1EJN, 1FAX, and 2TMN because of such errors). Before applying the docking calculation, the protein-ligand complexes were divided into a protein and a ligand using the Maestro software (Schrodinger, Inc.). The protein structure was optimized by the "Protein Preparation Wizard" within Maestro. This process includes five functions: "Remove cofactors", "Preprocess", "Optimize", "Remove waters", and "Minimize". The potential ligand conformations were generated by the "LigPrep" and "Epik" functions of Maestro.

4.2 Protocol to generate conformation search grids

The conformation search area for the docking simulations is determined based on the results of SiteMap. The SiteMap software predicts potential binding sites based on the protein's structural characteristics. In this experiment, we used SiteMap's default parameters and settings, except for the number of max reports, which was changed from 5 to 10 because the default value is too small for larger proteins.

Search grids were generated by the "Glide Grid Generation" function of Maestro. In the standard protocol, a search grid is located on the centroid of the site-points obtained by SiteMap. The edge size of the INNERBOX (the center of a ligand is restricted to this box through the docking process) is given by the ligand diameter, and the edge size of the OUTERBOX (all atoms of a ligand are restricted to this box) is set to the INNERBOX edge size + 16Å. In the proposed method, search grids are arranged at each of the selected site-points by our grid arrangement algorithm. The edge size of the INNERBOX and OUTERBOX are fixed to 10Å and 26Å, respectively. Therefore, the search grids generated by the proposed method are different from those in the Glide standard protocol. However, both methods satisfy the condition that all site-points given by SiteMap are included in any grid.

4.3 Protein-ligand docking using Glide

The docking results are highly dependent on the initial pose of the ligand. Thus, before the docking simulation, a sufficient number of initial ligand conformations were generated using "LigPrep" with its default settings. The protein-ligand dockings were performed using the "Ligand Docking" Glide function with default settings. Glide has two prediction modes, standard precision (SP) and extended precision (XP). Compared with SP mode, XP is slower but more accurate. In consideration of the computational cost, we used SP to predict the binding mode in this experiment.

As mentioned above, a direct comparison of the efficiency of the proposed method with that of the standard protocol under the default settings is difficult. Thus, we used the "heavier" setting in the standard protocol to enable a reasonable comparison. It is possible to improve the conformation search by increasing the number of searches, although this entails a heavier calculation. Under the heavier setting, the standard protocol forms one grid, as for the default setting. Therefore, we implemented the standard protocol with this heavier setting, and increased the number of conformation searches to that of the proposed method.

Table 1: Performance comparison of three methods

	Standard (default)			Proposed			Standard (heavier)		
PDB	Score [kcal/mol]	RMSD [Å]	time [sec]	Score [kcal/mol]	RMSD [Å]	time [sec]	Score [kcal/mol]	RMSD [Å]	time [sec]
1A4G	-7.34	23.9	5783	-8.08	23.5	9527	-7.34	23.6	25426
1AJ7	-7.33	1.8	1413	-8.02	2.3	2856	-7.71	2.2	1734
1B9V	-6.16	22.9	721	-7.15	23.6	1626	-5.43	23.7	997
1DBB	-8.74	0.5	1244	-9.13	0.5	2671	-8.73	0.5	1418
1EJN	-6.77	12.3	502	-8.84	1.0	712	-7.60	1.1	632
1FAX	-8.47	8.7	727	-8.78	11.1	1117	-9.16	4.4	1521
1FKG	-7.76	1.6	128	-6.81	5.1	120	-7.76	1.6	130
1HDC	-8.07	6.1	956	-7.96	6.1	2443	-8.05	6.1	1550
1IBG	-8.66	2.3	6705	-8.84	1.2	15601	-8.66	2.3	27230
1MMQ	-8.15	9.5	185	-7.58	9.7	310	-8.24	1.5	233
1QBR	-8.39	10.5	1108	-8.39	11.6	1427	-11.23	1.8	1278
1RNE	-13.64	1.5	43850	-15.57	0.6	75126	-13.64	1.5	68986
1TPH	-6.48	1.1	315	-6.26	1.2	460	-6.48	1.1	363
1XKB	-7.78	9.0	923	-11.52	2.1	1621	-11.51	2.0	1528
2DBL	-8.67	1.1	2082	-9.02	1.1	4682	-8.67	1.1	5865
2H4N	-5.02	6.3	526	-5.32	15.7	728	-5.03	6.3	809
2TMN	-5.23	2.6	469	-5.66	3.1	636	-5.97	4.2	664
2YPI	-8.16	0.8	513	-7.90	3.7	1083	-7.99	1.0	632
3ERD	-9.87	0.5	541	-9.95	0.6	804	-9.87	0.5	630
7CPA	-8.21	4.5	953	-9.21	4.5	1698	-8.71	4.2	1691
Average	-7.95	6.4	3482	-8.50	6.4	6262	-8.39	4.5	7166

4.4 Results of the evaluation experiment

Table 4.4 shows the docking scores, root mean square deviation (RMSD), and execution time for the proposed method and standard protocol with the default and heavier settings. The docking score is essentially the same as GlideScore, but is compensated by Epik state penalties [19]. Conformation searches are performed using GlideScore in the docking process, but the final output of Glide is a docking score. Therefore, we used the docking score as an evaluation metric in this experiment. This score represents the binding energy between a protein and a ligand, and so smaller values are better. The "Score" column shows the value of the lowest docking score. We also show the RMSD of all atoms superposed by a protein between the conformation of the crystal structure and the conformation of the complex with the best docking score. RMSD is often used to evaluate the accuracy of dockings. However, we did not use RMSD to measure the conformational search performance, because in many cases a better docking score has a larger RMSD. This is because the RMSD is highly dependent on the scoring function as well as the search performance. Therefore, we only used the docking score to evaluate the conformation search performance in this work.

From the results in Table 4.4, we can see that the proposed method exhibits the best search performance of the three methods considered. In addition, the docking score of the proposed method is better than that of the standard protocol with default settings for 15/20 complexes, and outperforms the standard protocol with the heavier setting in 13/20 cases.

The execution time of each method is shown in the "Time" column. This includes the time required by the proposed method to determine the optimal grid arrangement, as this is trivial compared with the overall execution time. From Table 4.4, we can see that the execution time of the proposed method is approximately twice that of the standard protocol with default settings. However, the proposed method is approximately 15% faster than the standard protocol with the heavier setting.

5. Discussion

5.1 Statistical significance of the improvement

The proposed method gives the best average docking score, and beats the docking score of the standard protocol with default settings in 75% of cases. Thus, we believe that the search performance of the proposed method is considerably better than that of standard protocols. To confirm this, we conducted a statistical test to check whether the difference is significant. Assuming non-parametric distributions, we applied a two-sample paired Wilcoxon signed rank test

[20] to the docking scores. This is a non-parametric statistical hypothesis test to assess the significance of differences between two related samples. We used the "wilcox.test" function of R 3.0.0 with the "pair" option.

First, we compared the standard protocol with default settings with the proposed method. The p-value of the test was 0.02, and the difference in performance was found to be statistically significant at the 0.05 level. Thus, the proposed method has significantly better conformational search performance, although its computational cost is greater.

We also compared the results from the standard protocol with the heavier setting with those given by the proposed method. The p-value of this test was 0.41, indicating that there is no significant difference in performance at the 0.05 level. Thus, from this experiment, it is impossible to conclude that the search performance of the proposed method is better than that of the standard protocol with the heavier setting. However, the proposed method is faster, and thus more efficient, than the standard protocol with the heavier setting.

5.2 The effect of optimal grid arrangement

Our grid arrangement algorithm is designed to minimize the number of search grids. In this experiment, arranging a grid for every site-point obtained by SiteMap would require an average of 4893.6 grids. However, using our optimal grid arrangement algorithm, this number decreased to only 14.2. Because the computational cost increases in proportion to the number of search grids, our grid arrangement method reduces the cost by a factor of approximately 350.

Of course, it is possible to employ other grid arrangement methods. To show the advantage of our method, we implemented another simple grid arrangement method that divides the large grid into small uniform grids at even intervals. Figure 4 shows an example arrangement given by this division method. In the figure, the white dots are site-points obtained by SiteMap, and the small crosses denote the centers of each search grid. We removed all grids that did not include any site-points. The union of the grids generated by the algorithm can also include all dots. We applied this arrangement method to the dataset used in the experiment. This algorithm generated an average of 22.7 search grids, which is more than in the proposed method. These results indicate that our proposed method can effectively decrease the number of search grids, and thus the computational cost.

6. Conclusion

In this study, we aimed to improve the conformation search of protein-ligand docking by avoiding local minima in large search areas. Thus, we proposed a method to improve the search efficiency by dividing one large search gird into multiple small search grids. In addition, we developed a technique that minimizes the number of such grids by converting the problem into a set cover problem. The results of an evaluation experiment show that the proposed method improves the docking score relative to the standard protocol. Unfortunately, however, statistical tests did not show a clear improvement over the standard protocol with the heavier setting. The computational cost of the proposed method was lower than that of the standard protocol with the heavier setting, which indicates that our method has better search efficiency than the standard protocol. In this research, the standard protocol with the heavier setting predicts the binding mode of a crystal structure better than the proposed method. We think this is due to the inaccuracy of the docking score. Thus, in future work, we will investigate the relationship between the docking score and the RMSD, and refine the score function to improve conformational searches.

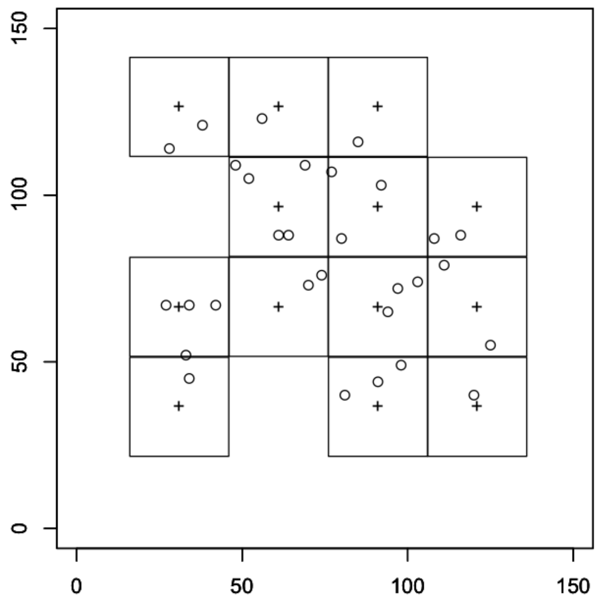


Figure 4: Example 2D grid arrangement given by the simple division method

7. ACKNOWLEDGMENTS

This work was supported in part by the Education Academy of Computational Life Sciences (ACLS) at Tokyo Institute of Technology. The authors would like to thank the TSUBAME supercomputer system at the Global Scientific Information and Computing Center, Tokyo Institute of Technology.

References

[1] Xie, Li, et al. "Drug discovery using chemical systems biology: identification of the protein-ligand binding network to explain the side effects of CETP inhibitors." PLoS Computational Biology 5.5 (2009): e1000387.
[2] Ramirez, Ursula D., et al. "Docking to large allosteric binding sites on protein surfaces." Advances in Computational Biology. Springer New York, 2010. 481-488.

[3] Goodsell, David S., Garrett M. Morris, and Arthur J. Olson. ”Automated docking of flexible ligands: applications of AutoDock.” Journal of Molecular Recognition 9.1 (1996): 1-5.
[4] Jones, Gareth, et al. ”Development and validation of a genetic algorithm for flexible docking.” Journal of Molecular Biology 267.3 (1997): 727-748.
[5] Rarey, Matthias, et al. ”A fast flexible docking method using an incremental construction algorithm.” Journal of Molecular Biology 261.3 (1996): 470-489.
[6] Friesner, Richard A., et al. ”Glide: a new approach for rapid, accurate docking and scoring. 1. Method and assessment of docking accuracy.” Journal of Medicinal Chemistry 47.7 (2004): 1739-1749.
[7] von Korff, Modest, Joel Freyss, and Thomas Sander. ”Comparison of ligand-and structure-based virtual screening on the DUD data set.” Journal of Chemical Information and Modeling 49.2 (2009): 209-231.
[8] Kellenberger, Esther, et al. ”Comparative evaluation of eight docking tools for docking and virtual screening accuracy.” Proteins: Structure, Function, and Bioinformatics 57.2 (2004): 225-242.
[9] Marcou, Gilles, and Didier Rognan. ”Optimizing fragment and scaffold docking by use of molecular interaction fingerprints.” Journal of Chemical Information and Modeling 47.1 (2007): 195-207.
[10] Laurie, Alasdair TR, and Richard M. Jackson. ”Q-SiteFinder: an energy-based method for the prediction of protein-ligand binding sites.” Bioinformatics 21.9 (2005): 1908-1916.
[11] Halgren, T. A. : ”New Method for Fast and Accurate Binding-site Identification and Analysis”, Chem. Biol. Drug Des., 2007, 69, 146.
[12] Eldridge, M. D.; Murray, C. W.; Auton, T. R.; Paolini, G. V.; Mee, R. P. Empirical scoring functions: I. The development of a fast empirical scoring function to estimate the binding affinity of ligands in receptor complexes. J. Comput.-Aided Mol. Des. 1997, 11, 425-445.
[13] Vijay V. Vazirani., ”Approximation algorithms” Springer p15-p26
[14] M.R.Garey and D.S. Johnson, ”Computers and Intractability: A Guide to the Theory of NP-Completeness”, W.H. Freeman and Company (1979)
[15] http://www.schrodinger.com/scriptcenter/#Docking
[16] Nissink, J. Willem M., et al. ”A new test set for validating predictions of protein-ligand interaction.” Proteins: Structure, Function, and Bioinformatics 49.4 (2002): 457-471.
[17] http://helixweb.nih.gov/schrodinger-2013.3-docs/glide/glide_user_manual.pdf, p58
[18] http://www.schrodinger.com/kb/348
[19] http://www.schrodinger.com/newsletter/12/68/
[20] Wilcoxon, Frank. ”Individual comparisons by ranking methods.” Biometrics bulletin 1.6 (1945): 80-83.

Validation of EEG Personal Authentication with Multi-channels and Multi-tasks

Yu Ishikawa, Chinami Yoshida, Masami Takata, Kazuki Joe
Nara Women's University, Nara, 630-8506, JAPAN

Abstract - *We investigate a feature extraction method that is effective for biometric identification using brain waves in this paper. We extract power spectrums of theta waves, alpha waves, beta waves and gamma waves for the quantity of personal characteristic. We measure brain waves with five tasks and multi-channel electroencephalograph to analyze each error rate. As a result, the error rate of the alpha wave is the lowest; the authentication rate by a single channel / a single task, sixteen channels / a single task, and sixteen channels and five tasks are 90%, 92%, and 97%, respectively.*

Keywords: biometric authentication, brain wave, image task, power spectrum

1 Introduction

As the spread of Internet infrastructure in recent years, various social networking services such as on-line shopping are provided for the information society. According to the increase of usability, crimes using the Internet are increasing rapidly, and the importance of authentication technologies to prevent such unauthorized access is required more than ever. While personal authentication by ID and password has been used mainly in the social networking services available at the time, it would have been forged easily against prying eyes or brute-force authentication attempts. As just described, conventional authentication technologies cannot be said to be reliable means necessarily in terms of safety, and biometric authentication is getting a lot more attention recently.

The biometric authentication refers to the personal authentication using biometric information. The biological information for the authentication includes fingerprint, iris, face, voiceprint, and handwriting, where plagiarism is difficult as compared with the traditional password-based authentication. In particular, iris or fingerprint based authentication does not provide high recognition rates but also be used in practical applications, but there are also reports that some authentication systems are forged [1]. Since the biological information is exposed to the outside at all times, it can be acquired for forgery on the basis of the biometric information. A vein based authentication system that uses in-vivo (not exposed to the outside) information is introduced while there is a report that spoofing is possible even for the vein authentication system. Conventional biometric authentication methods have a problem that it is impossible to change the biometric information right after succeeding the authentication once. It is considered that changeable biometric information is applicable to the biometric authentication so that it is to be updated if the biometric information is plagiarized. For these reasons, updatable in-vivo biological information is required.

In this paper, a biometric authentication using brain waves as the biometric information is proposed. Electroencephalogram (EEG) is in-vivo information and superior to confidentiality because it is measured neuronal activity of a number of cerebral cortices and not measurable without wearing electroencephalograph. Also it shows different characteristics depending on the individual, and can be used to intentionally change the own brain waves by changing what he/she images. The concept of personal authentication using the EEG and images has been proposed as pass-thoughts [2]. By making an efficient use of this feature, the biological information is possibly updated on a regular basis, and the safety is enhanced.

Studies utilizing brain waves for authentication are already underway by various researchers. For example, EEGs of forty examinees during open-eye-closed-eye are measured for personal identification to get an accuracy of 80% is reported in [3], and EEG rhythms of four examinees during closed-eye are analyzed to get an accuracy of 90% or more [4]. Other studies include personal authentication by visual evoked potential [5] and verbal recall problems and/or potential recall movement [6]. In such previous studies, auto-regressive (AR) models [3] [4] [7] or neural networks [5] [6] [8] for feature extraction have been proposed, but their computational costs are too expensive while the authentication performances are improved more than 90%. In this paper, we propose a personal authentication method with light computational cost as well as good authentication performance using frequency distribution of the target power spectrum as feature values.

The rest of the paper is organized as follows. In section 2, we introduce two types of related works. According to the existing research results, we propose an extension of the previous methods in section 3. The proposed method is validated with some experiments in section 4.

2 Related works

2.1 EEG personal authentication based on the

average power spectrum

We introduce a study for EEG personal authentication with light computational cost using average power spectrum as feature values [9]. In this study, a method of personal authentication by EEG brain waves during virtual driving operation, which means a simple driving simulator with tracing route, is proposed.

Electroencephalograph to be used is a single electrode of the frontal lobe Fp1 (International 10–20 system) with the sampling frequency of 128Hz. Spectral analysis by Fourier transform for the extraction of individual feature is adopted based on the fact that there are individual differences in brain wave spectrum in the α-β wave band. Exactly saying, the α and the β wave bands are divided into α1-α4 and β1-β4 regions, respectively. In each region, the average of power spectrums as individual feature is evaluated for authentication.

The flow of authentication process is as follows. First, the personal data is registered in advance to perform authentication. The brain waves during virtual driving operation are measured to calculate the power spectrum by FFT. The measured EEG spectrum is smoothed with five points moving average process. The process is performed L times for each examinee to generate L spectrums, which are spatially averaged. The α-β wave band parts of the obtained averaged spectrums are divided into several regions and the average value is calculated for each region. The average value is the template.

In collation process, examinee's brain waves in the virtual driving operation environment are measured once to calculate the spectrum as in the template and perform smoothing and normalization. Note that the normalization is a process to align the average spectrum used in the template calculation with the average spectrum for collation. To be compared with the template, if it is smaller than a predetermined threshold, it is authenticated as the right person.

For the authentication experiment, thirty examinees are employed. Measurement time is three minutes, and ten sets of brain waves data are taken from each examinee: five sets of data are used for generating templates, and the rest of five sets are for authentication. Evaluating experimental results by equal error rate (EER), the tracing route and driving simulator record 0.35 and 0.36, respectively. Furthermore, EERs are 0.51 and 0.22 in the case of B1 region and B3 region, respectively. In this way, a remarkable difference in each area is observed.

2.2 EEG personal authentication using multiple tasks

While the previous sub-section describes a study aiming at EEG measuring and its authentication during unconscious states, this sub-section describes another study for personal authentication of the EEGs that are measured with selecting own password thoughts (Pass-thoughts) [10].

They use the MindSet of NeuroSky Inc. as an electroencephalograph with 200Hz sampling frequency at frontal lobe Fp1 by a single channel. The following seven tasks are used for measuring brain waves: Deeply breathe (*breathing*), Image moving own fingers up and down (*finger*), Image doing a favorite sport (*sport*), Image singing a favorite song (*song*), Listen to a mechanical sound, then gaze at one point (*audio*), Select a color among red, green, blue and yellow, then count the number of the selected color on a displayed picture (*color*), Image a favorite password (*pass*). In *audio*, the examinee listens to a mechanical sound for five seconds, then gazes at one point for five seconds while his/her brain waves are measured for the total of ten seconds. In *color*, the examinee counts the number of selected color on a picture with measuring his/her brain waves for five seconds. Since the number of pictures is six, the total time for measuring examinee's brain waves is thirty seconds in total. In the rest of five tasks, the examinee imagines the tasks while his/her brain waves are measured for ten seconds.

In the authentication method, an STFT (short-time Fourier transform) is first applied to the recorded sample data as a pre-processing, and time-frequency analysis is performed. The frequency bands of the α and the β waves are cut out to calculate the median of the power spectrum for each frequency band. The resultant data is used as one-dimensional feature vector. Then, to calculate the degree of similarity between the one-dimensional vectors obtained from the sample data, the cosine similarity method is applied. From the above calculated self-similarity and cross-similarity, an optimal threshold value for authentication is estimated. The authentication is evaluated by the HTER (Half Total Error Rate), which is the average of the FAR (False Acceptance Rate) and the FRR (False Rejection Rate) calculated from each similarity.

In experiments with fifteen examinees, when evaluated using a common threshold to the examinees, the HTER is 0.32-0.43. However, when changing the task and the threshold for each examinee, the HTER decreased up to 0.011

3 Analysis of personal authentication using a multi-channel electroencephalograph

In this paper, we perform personal authentication using a multi-channel electroencephalograph based on the above related works, and analyze the results for several factors.

3.1 Measuring EEG

We use the BioSemi as the multi-channel electroencephalograph, of which the maximum sampling rate is 2,048Hz and the maximum electrode number is 256. A bipolar lead method is used for deriving the reference electrode. In this paper, we use a BioSemi with the maximum

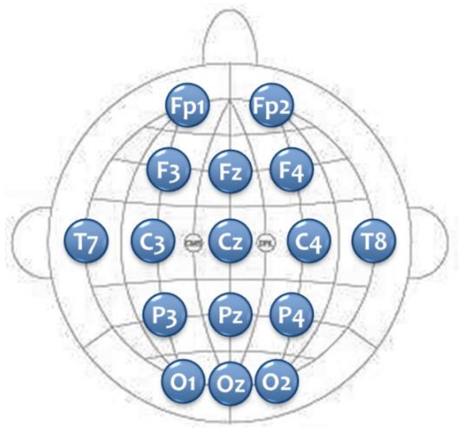


Figure 1 Electrode placement

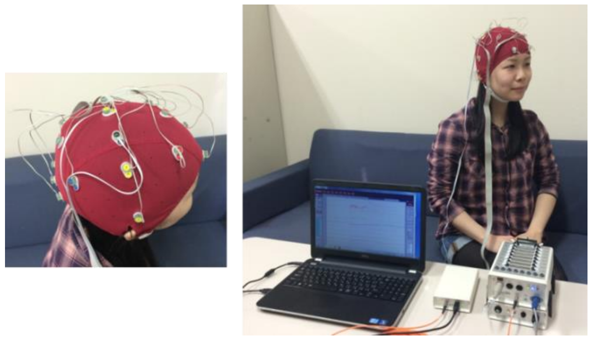


Figure 2 Mount a BioSemi and its measuring

sampling rate of 2,048Hz and 16 electrodes. The electrodes are placed as shown in Fig.1 based on International 10–20 system. Figure 2 shows mounting a BioSemi and measuring EEGs in the left and right, respectively.

3.2 Authentication method

We use an authentication method as described in subsection 2.1 and 2.2 to be extended. First, a template for individual is generated using *L* trial data sets. Each data set is processed as following. We apply STFT to the data to calculate the power spectrum by time, and perform a median filter on successive five values. In subsection 2.1, 8-29Hz brain waves (α and β waves) are partitioned into seven regions to calculate the average of each power spectrum as feature values. As a result, it is reported that the authentication rate at B3 β-wave region is good, but this is likely because some differences are observed in the β wave EEG that usually evokes concentration during virtual driving operations. In this paper, we use 4-40Hz brain waves (θ to γ waves) to get more information. Since the brain waves less than 3Hz contain considerable EOGs (electrooculography), we do not use them in this paper. Figure 3 shows an example θ-γ wave bands partition. The θ-γ wave bands are partitioned into two, three, five and three regions of θ1-θ2, α1-α3, β1-β5 and γ1-γ3, respectively. The average power spectrum for each region by time is calculated to be used as feature values. The

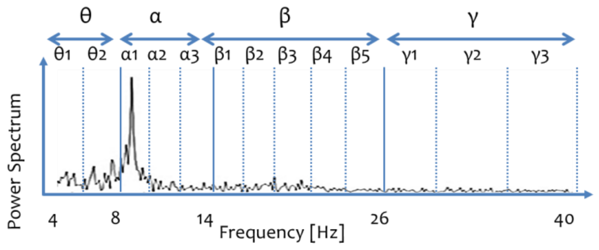


Figure 3 An example of θ-γ wave bands partition

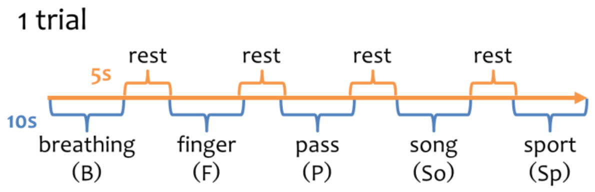


Figure 4 the flow of a trial

calculated feature values are averaged by each examinee to generate the template $S\begin{pmatrix}\theta1,1 & \theta2,1 & \cdots & \gamma3,1\\ \theta1,2 & \theta2,2 & \cdots & \gamma3,2\\ \vdots & & \ddots & \vdots\\ \theta1,t & \theta2,t & \cdots & \gamma3,t\end{pmatrix}$, where *t* represents time.

The authentication is performed using each feature value in the template and its cosine similarity. If the similarity is larger than a pre-defined threshold, the examinee is accepted. Otherwise, the examinee is rejected as a personator. The threshold is selected as the minimum EER that is the intersection of graphs FAR and FRR. The minimum EER is also used for evaluating the authentication accuracy performance.

4 Experiments

4.1 Experiment method

The examinees are ten healthy women in their 20s. To measure their brain waves, we adopt five tasks except *color* and *audio* described in subsection 2.2. Since tasks of *color* and *audio* require an open-eye state, we think they increase noises. We abbreviate tasks *breathing*, *finger*, *pass*, *song* and *sport* to **B**, **F**, **P**, **So** and **Sp**, respectively. Figure 4 shows the flow of a trial. We measure ten seconds for each of five tasks with taking a break of five seconds between each task, which is a trial. Each examinee performs ten trials. The learning data, namely templates are from the measured data of five examinees, and the rest of five examinees data are used for authentication tests. In other words, the number of authentication test data is fifty.

We perform the experiments for three different purposes. Experiment 1, 2 and 3 are to validate the equal error rate (EER) for each frequency band, each measuring position and

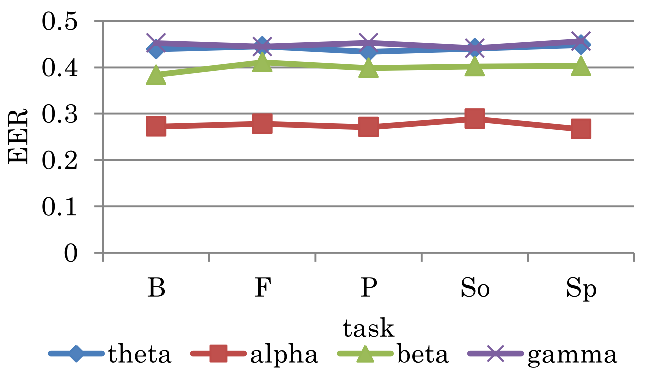


Figure 5 Results of Exp.1a

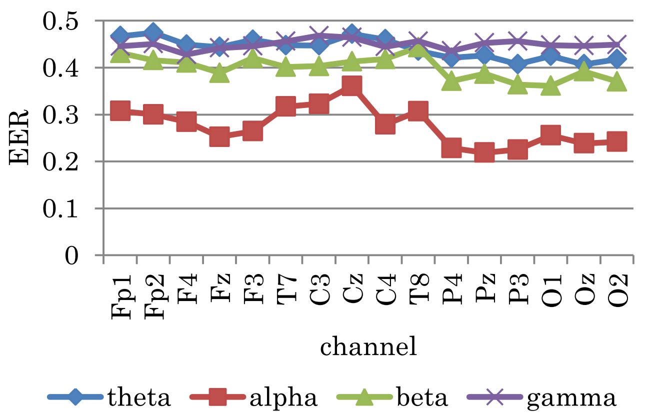


Figure 6 Results of Exp.1b

task combinations, respectively. We randomly select five learning data sets from the ten measured data sets and the rest of five data sets are used as authentication test data so that we have ten kinds of learning-test data set combination. The resultant ten EERs are averaged for the validation.

4.2 Experiment 1: Validate EER for each frequency band

We investigate the difference in EERs when changing the frequency band of the EEGs used for authentication.

In Experiment 1a, we get the EER of each frequency band in each task. The frequency bands consist of θ wave (4-8Hz), α wave (8-14Hz), β wave (14-26Hz) and γ wave (26-40Hz). From the EEGs for each task with changing the frequency band of the feature, we calculate the EER for each channel to obtain the average value of the 16 channels. Figure 5 shows the plots of the experiment results. The average EERs of θ, α, β and γ waves are 0.44, 0.28, 0.40 and 0.45, respectively. The authentication by α wave achieves the highest preciseness. It is also confirmed by Fig.5 that the authentication by α wave is more accurate for all tasks, and

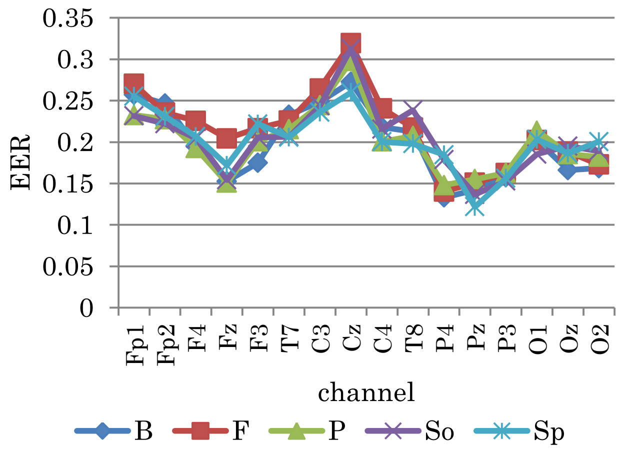


Figure 7 Results of Exp.2a

the difference to the second accurate authentication by β wave is more than 0.1. Investigating the results by task, *pass* and *sport* provide better accurate authentication while other tasks provide worse. This trend is true for all frequencies. Looking at α wave, *sport* is the best (0.27) and *song* is the worst (0.29).

In Experiment 1b, we get the EER of each frequency at each electrode position. Similar to Experiment 1a, the EER for each electrode position is calculated from the EEGs for each task with changing the frequency band of the feature. The obtained EERs are averaged by task for the validation as shown in Fig.6. As shown in the graph, the authentication accuracy by α wave is the best in all channels. In the case of α wave, the authentication performances from P4 to O2 are better than from Fp1 to T8. In the cases of θ wave and β wave, the performances from P4 to O2 are relatively good. The performances of θ-β waves on occipital area are better than frontal area. However, in the case of γ wave, no prominent differences for the EERs on any channels are confirmed.

From the results of Experiment 1a and 1b, the authentication accuracy is good in the order of α, β, θ and γ wave. Particularly, the authentication accuracy by α wave is outclassing. Although there are differences in the authentication accuracy by task, the performances of α and β waves are well for all tasks. Furthermore, focusing on the channels, the authentication accuracy of occipital α, β and θ waves is better than frontal. We think that the reason of the above trend is that the noises of eye movement are included in the frontal α, β and θ waves. Therefore, no influence is observed in the frontal γ wave that has a relatively high frequency.

4.3 Experiment 2: Validate EER for each measuring position

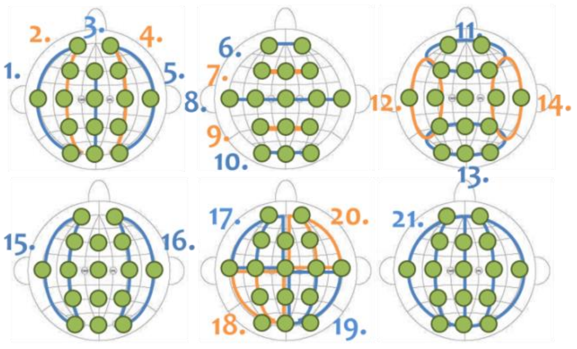


Figure 8 Electrode position pattern for Exp.2b

Table 1 Electrode position detail

pattern	channel
pattern1	Fp1, T7, O1
pattern2	Fp1, F3, C3, P3, O1
pattern3	Fz, Cz, Pz
pattern4	Fp2, F4, C4, P4, O2
pattern5	Fp2, T8, O2
pattern6	Fp1, Fp2
pattern7	F3, Fz, F4
pattern8	T7, C3, Cz, C4, T8
pattern9	P3, Pz, P4
pattern10	O1, Oz, O2
pattern11	Fp1, Fp2, F3, Fz, F4
pattern12	F3, T7, C3, P3
pattern13	P3, Pz, P4, O1, Oz, O2
pattern14	F4, C4, T8, P4
pattern15	Fp1, F3, T7, C3, P3, O1
pattern16	Fp2, F4, C4, T8, P4, O2
pattern17	Fp1, F3, Fz, T7, C3, Cz
pattern18	T7, C3, Cz, P3, Pz, O1, O2
pattern19	Cz, C4, T8, Pz, P4, Oz, O2
pattern20	Fp2, Fz, F4, Cz, C4, T8
pattern21	all channels

We investigate the difference in EERs due to the combination of electrode positions to be used for authentication. In Experiment 2a, we get the EER at each electrode position calculated from the EEGs for each task using the frequency band of 4-40Hz. Figure 7 shows the results. There is little difference between the authentication rates of all tasks. Although they are also affected by the kind of tasks, the channel with good authentication rates are Fz, P4, and Pz. The combination of a task and a channel with the best authentication rates is *sport*/Pz with the EER of 012. From the experiment results, we confirm that about 90% authentication

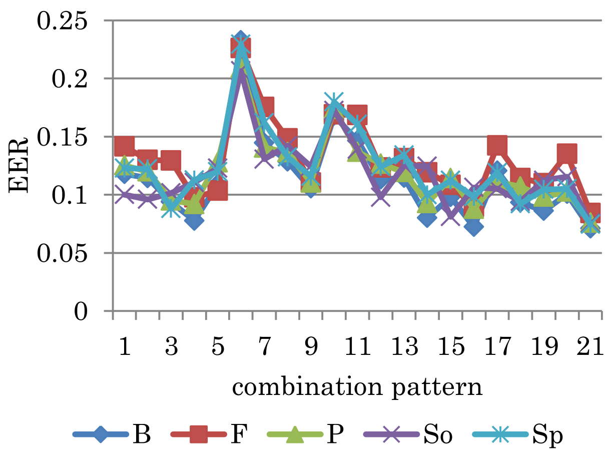


Figure 9 Results of Exp.2b

rate is obtained using all the 4-40Hz frequency with single channel and single task. Since *pass* is the task that is individual-changeable, EEGs with *pass* are expected as a pass-thought. The channel with bad authentication rates is Cz. The EER of the best channel P4 with *pass* is 0.15 while the EER of the worst channel Cz is 0.30. The results indicate that the authentication performance differs by electrode position so much.

In Experiment 2b, we generate a set of patterns by combining several channels, and use them to calculate EERs for each task on the electrode position to validate the patterns with good authentication performance. Figure 8 shows the patterns for the combined positions. The details of the patterns are shown in Tab.1. We investigate twenty-one combination patterns for each electrode position in this paper. Figure 9 shows the EER for the pattern 1-21 by task. Focusing on tasks, there is little difference between the authentication rates of all tasks. Focusing on patterns with accurate authentication, the pattern 21 for all tasks on all of 16 channels achieves the best EER. On the pattern 21, the task with best authentication performance is *finger* with EER 0.071. In Experiment 2a, while the authentication rate for the task *pass* on a single channel is about 85%, it increases up to about 92% on all of the 16 channels. The next best patterns include the pattern 16 occupying the left hemisphere for the tasks *breathing*, *finger* and *pass*, the pattern 15 occupying the left hemisphere and the pattern 3 passing through the center of the midline hemisphere for the task *song*. On the contrary, the patterns with poor authentication rates include the pattern 6 through the frontal pole and the pattern 10 through the occipital area. The experiment results indicate that different tasks give different electrode positions to each examinee. In particular, it turns out that the authentication performance using either hemisphere according to given tasks is better than using both hemispheres of the brain.

In Experiment 2c, we obtain the EERs on the electrode positions within the left, right and midline of hemisphere for

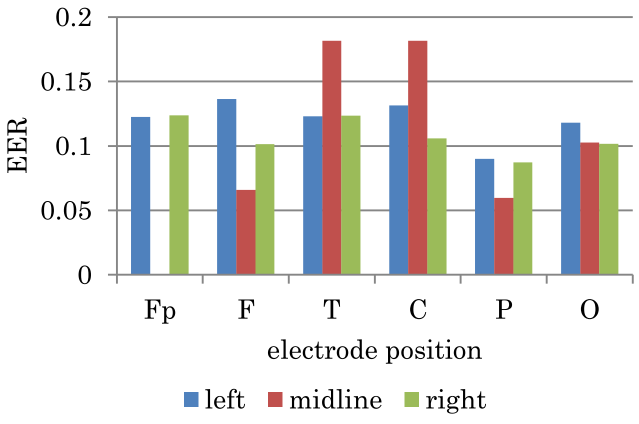


Figure 10 Results of Exp.2c

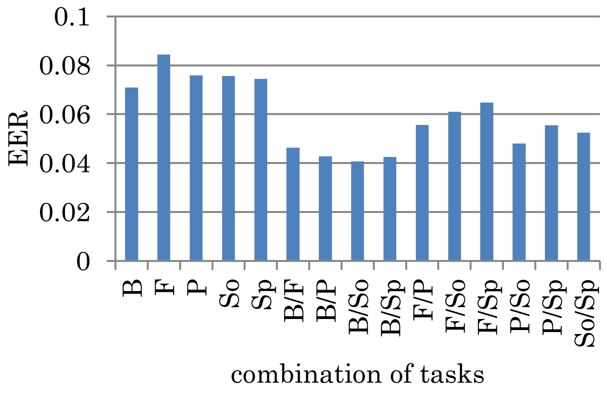


Figure 11 Results of Exp.3a (1-2 tasks)

all tasks as shown in Fig.10. Apparently, the electrode positions in the right of hemisphere give a little better authentication performance than the left. Especially, the electrode positions T and C within the right of hemisphere provide better authentication performance than the left/midline 0.03. The electrode positions of the midline hemisphere have a large difference between good authentication rates (T, C) and poor authentication rates (F, P). Furthermore, the electrode position P achieves the best authentication performance within any parts of hemisphere

4.4 Experiment 3: Validate EER for the combination of tasks

We investigate the difference in EERs due to the combination of tasks to be used for authentication. In Experiment 3a, we generate all combinations of one to five tasks to be used for the authentication and calculate each EER. In this paper, we present the pattern 21 (all channels) that gives the best authentication performance in Experiment 2b to calculate its EER. Figure 11 and 12 show the cases that the number of tasks to be combined is 1-2 and 3-5, respectively.

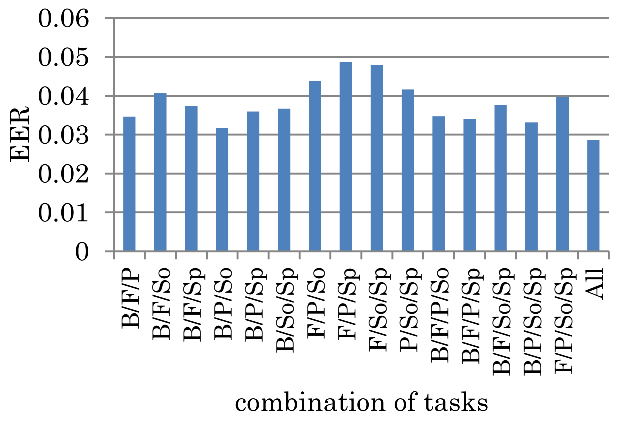


Figure 12 Results of Exp.3a (3-5 tasks)

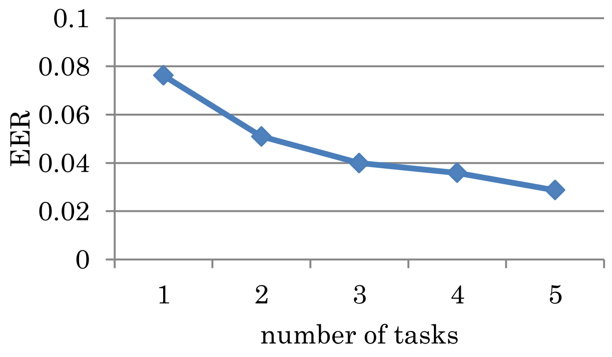


Figure 13 Results of Exp.3b

In the case of a single task, the task *breathing* gives good authentication performance. On the contrary, the task *song* gives the worst authentication performance of EER 0.084. In any case, the EER of each task is less than 0.09, which means a single task achieves good authentication of more than 92%.

The number of two task combinations is 10. The combinations of tasks including *breathing* which provide good authentication when used as a single task, achieve authentication performance of 95%. The combination of *breathing*/*song* provides the best authentication of EER 0.040.

The number of three task combination is 10, too. They all provide less than 0.05. Among the three task combinations, *breathing*/*pass*/*song* provides the best authentication of 0.031.

The number of four task combination is 5 and all combinations achieve good authentication of 0.03 to 0.04. Using all tasks, the authentication performance is the best 0.029.

In Experiment 3b, we investigate the difference of EERs with varying the number of tasks for the task combination. Figure 13 shows the average EERs by the number of tasks. The average EERs are 0.076, 0.051, 0.040, 0.036 and 0.029

as the number of tasks increases from 1 to 5. Namely, the larger the number of tasks for the task combination is, the better the average EERs are.

5 Conclusions

In this paper, we investigate EEG personal authentication with a 16-channel electroencephalograph. We adopt averaged power spectrums calculated by STFT for each frequency band of θ, α, β and γ waves as feature values for individuals. When EEGs are measured, the examinees perform five kinds of tasks; *breathing*, *finger*, *pass*, *song* and *sport*. Using the above data, we performed three types of experiments to validate EERs for each frequency band, EERs for electrode positions and EERs for the combination of tasks. From the experiment results of EER validation for each frequency band, we confirm that α waves provide the best authentication performance followed by β, θ and γ waves. It means that the α wave band is the most effective frequency of EEGs for personal authentication. From the experiment results of EER validation for electrode positions, about 85% authentication rates are observed with the task *pass* and the electrode position Pz, and we also observe that the authentication rates increase up to about 92% with all of 16 channels. Finally, from the experiment results of EER validation for the task combination with 16 channels, we confirm that the larger the number of task combinations is, the better the authentication performance is. Using all of five tasks, the EER achieves the best 0.029 that means the authentication rate of more than 97%.

Our future work includes investigating optimal tasks, electrode positions and frequency band with more examinees to get better authentication performance. We believe that power spectrum of EEGs is insufficient. So we should develop a new feature for individual authentication rather than power spectrum

6 References

[1] T. Matsumoto, H. Matsumoto, K. Yamada and S. Hoshino, Impact of artificial "Gummy" fingers on fingerprint systems, vol. 4677, Proc. SPIE, 2002, pp. 275-289.

[2] J. Thorpe, P. v. Oorschot and A. Somayaji, Pass-thoughts: Authenticating With Our Minds, NSPW '05 Proceedings of the 2005 workshop on New security paradigms, 2005, pp. 45-56.

[3] R. B. Paranjape, J. Mahovsky, L. Benedicent and Z. Koles, The Electroencephalogram as a Biometrics, vol. 2, Proc. of 2001 Canadian Conference on Electrical and, 2001, pp. 1363-1366.

[4] M. Poulos, M. Rangoussi, V. Chrissikopoulos and A. Evangelou, Parametric person identification from the EEG using computational geometry, vol. 2, Proc. of the 6th IEEE Int. Conf. on Electronics, Circuits and Systems, 1999, pp. 1005-1008.

[5] R. Palaniappan and D.P. Mandic, Biometrics from brain electrical activity: A machine learning approach, vol. 29 no.4, IEEE Trans. Pattrn Anal. Mach Intell, 2007, pp. 738-742.

[6] S. Marcel and J. R. Millan, Pearson Authentication Using Brainwaves (EEG) and Maximum A Posteriori Model Adaption, vol. 2, IEEE Trans. on Pattern Analysis and Machine Intelligence, 2007, pp. 743-748.

[7] A. Riera, A. Soria-Frish, M. Caparrini, C. Grau and G. Ruffini, Unobtrusive biometrics based on electroencephalogram analysis, vol. 2008, EURASHIP J.Advances in Signal Processing, 2008, pp. 1-8.

[8] K. Ravi and R. Palaniappan, Recognising Individuals Using Their Brain Patterns, vol. 2, Proc. 3rd International Conference on Information Technology and Applications, 2005, pp. 520-523.

[9] I. Nakanishi, H. Fukuda and S. Li, Biometric Verification Using Brain Waves toward On-Demand User Management Systems Performance differences between divided regions in α − β wave band, Proc. of the 6th International Conference on Security of Information and Networks, 2013, pp. 131-135.

[10] J. Chuang, H. Nguyen, C. Wang and B. Johnson, I Think, Therefore I Am: Usability and Security of Authentication Using Brainwaves, Workshop on Usable Security 2013 (USEC13), 2013, pp. 1-16.

Emotion Estimation of Comments on Web News by SVM and Naive Bayes Based Classifiers

Yasuhiro Tajima and Genichiro Kikui
Department of Systems Engineering, Okayama Prefectural University,
111, Kuboki, Soja, Okayama 719-1197, Japan

Abstract—*Social communication tools such as Twitter or Facebook spread the web service ability. Using their APIs, we can gather many users' comments easily. Such comments are usually short sentences but they also have many emotional comments. In this paper, we propose emotion estimation methods for multilabeled short comments of web news. Our methods can be applied to sentiment analysis and opinion mining. At first, we show the performance evaluation of a naive Bayes classifier and an SVM classifier. Then, we propose two improved methods. The first is an improved naive Bayes method which classifies each emotion label into two opposite emotions and uses their weights. We call this the weighting method. The second method consists of two stages of classifiers. The first stage distinguishes these oppositely classes, and the second stage selects one emotion from the opposite emotions. From our evaluation, we conclude that the weighting method is better among the naive Bayes classifiers and its performance is as good as SVM's.*

Keywords: emotion estimation, Twitter, naive Bayes, SVM

1. Introduction

In recent years, social networking tools have very important role on human communication such as Twitter or Facebook and so on. They usually have APIs for mash up with other web services. Especially, many web news sites use this function for gathering users' comments. Some TV programs also use these tools to make a bidirectional communication. These comments are useful for both of article writers and readers, but oftenly there is no retrieval system. Even though, there will be text base retrieval such as search engines and marker based systems such as "hash tag", but there is no system which responses to the request as "search funny comments."

In this paper, we propose an emotion labeling method to such comments. The emotions is comment writer's emotion. For example, if there is a news article about some crime and a comments such that "It will happen near my town.", then the comment writer may feel "fear" and "anticipation." Comments of web news articles have the following properties.

- They will be more emotional than other tweets. The comments to the news article are usually impressive.
- They will be short sentences. Twitter restricts the length of comments up to 140 characters, and other social tools have the same restriction.
- There will be no discussion. Some board systems have the comment tree making function, but many systems do not have.

Automatically emotion estimation of tweets is useful from these reasons.

In this paper, we propose two naive Bayes based classifier and performance evaluation with SVM and the simple naive Bayes classifier. Our new method uses the class of emotions which consists of two opposite emotions such as "joy" and "sadness". This is because, we use Plutchik's wheel of emotions[5], and there are eight emotions which can be classified into four classes. For the evaluation, we made experiments by Japanese news articles and their about 2000 tweets. The SVM and our proposed new method marked high performances comparing to the simple naive Bayes classifier.

There are some related studies about emotion estimation. In [1], a Japanese valency pattern dictionary for emotions has been made and emotion estimation for a sentence has been tried. An emotion corpus has also been made in [2]. Machine learning approach to emotion estimation has been tried in [3] and [4].

2. Vector models for emotion estimation

We denote a sentence of a tweet t which consists of n words by $t = w^{(1)}w^{(2)} \cdots w^{(n)}$. W_l and W_t denote vocabularies which appear in learning data and evaluation data, respectively. Let $W = W_l \cup W_t$, then $|W|$ denotes the size of the vocabulary of all data. Without loss of generality, assume an order on $W = \{w_1, w_2, \cdots, w_m\}$ and another order on $W_l = \{w_1, w_2, \cdots, w_l\}$. Now, $m = |W|$ and $l = |W_l|$ hold. On the naive vector modeling, we assume a map from a tweet t to m-dimensional vector $(u_1, u_2, \cdots, u_m)$. In this paper, u_i is the number of w_i which appears in t, i.e. $u_i = |\{j | w_i = w^{(j)}\}|$ where $t = w^{(1)}w^{(2)} \cdots w^{(n)}$. We denote the appearance of w_i by δ_i such that

$$\delta_i = \begin{cases} 1 & \exists j, w_i = w^{(j)} \\ 0 & otherwise \end{cases}$$

for $i = 1, 2, \cdots, m$. To avoid the zero frequency problem, we use additive smoothing for naive Bayes based methods.

We do not care the words which only exists in evaluation data such that $w \in (W_t - W_l)$ for SVM classification.

Target emotions are the following eight emotions.

- joy
- trust
- fear
- surprise
- sadness
- disgust
- anger
- anticipation

These are components of Plutchik's wheel of emotions[5]. In our setting, every tweet can have multilabels of emotions. A tweet t can be labeled by both of "joy" and "surprise" for example. Every tweet must have at least one label of the above emotions.

These eight emotions can be classified into four classes such that

- joy $\Longleftrightarrow$ sadness
- trust $\Longleftrightarrow$ disgust
- fear $\Longleftrightarrow$ anger
- surprise $\Longleftrightarrow$ anticipation

because of the pair of opposite emotions.

3. Estimation method

3.1 Simple naive Bayes

For probabilistic variables X, Y, it hols that

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

and this is called Bayes' theorem. Y denotes the target event. In our method, Y can take an event from {joy, trust, fear, surprise, sadness, disgust, anger, anticipation}. X denotes the vector which corresponds to a tweet t, i.e. X is an m-dimensional vector $(u_1, u_2, \cdots, u_m)$. If it holds that $P(Y|(u_1, u_2, \cdots, u_m)) \geq Th$ then the tweet t is labeled by the emotion Y. Here, Th is the threshold value and we define it $\frac{1}{8} = 0.125$ because there are eight emotions. If there exist more than two emotions, for example $P(Y = \text{“}joy''|(u_1, u_2, \cdots, u_m)) > Th$ and $P(Y = \text{“}trust''|(u_1, u_2, \cdots, u_m)) > Th$ holds, then t is a multilabeled tweet by "joy" and "trust". It holds that

$$P(Y|(u_1, u_2, \cdots, u_m)) = \frac{P((u_1, u_2, \cdots, u_m)|Y)P(Y)}{P((u_1, u_2, \cdots, u_m))}$$

from Bayes' theorem. In addition, $P((u_1, u_2, \cdots, u_m)|Y)$ and $P((u_1, u_2, \cdots, u_m))$ can be approximated by the followings.

$$P((u_1, u_2, \cdots, u_m)|Y) = \prod_{i=1,\cdots,m} P(w_i|Y)^{u_i}$$

$$P((u_1, u_2, \cdots, u_m)) = \prod_{i=1,\cdots,m} P(w_i)^{u_i}$$

Thus, $P(w|Y)$ and $P(w)$ for all $w \in W$ are needed to decide the labels of t. These values are estimated from learning data. $P(w|Y)$ is the probability that w appearance in all tweets with the emotion label of Y. $P(w)$ is the probability that w appearance in learning data.

3.2 Weighted naive Bayes

We can classify the set of emotions introduced by Plutchik[5]. That is four classes and each of them consists of opposite emotions: joy and sadness, trust and disgust, fear and anger, surprise and anticipation. Now, we assume that only one emotion on the each pair tends to be labeled. Let

$$\begin{array}{ll} y_1 = \text{“joy”}, & n_1 = \text{“sadness”}, \\ y_2 = \text{“trust”}, & n_2 = \text{“disgust”}, \\ y_3 = \text{“fear”}, & n_3 = \text{“anger”}, \\ y_4 = \text{“surprise”}, & n_4 = \text{“anticipation”} \end{array}$$

and $C_i = \{y_i, n_i\}$ for $i = 1, 2, 3, 4$. One emotion can be written by (C_i, m_i) where $m_i \in C_i$ for $i = 1, 2, 3, 4$. Let C and M are probabilistic variables of C_i and m_i, respectively. Then, $P(Y|w)$ can be written by the following for a word $w \in W$.

$$\begin{aligned} P(Y|w) &= P(C, M|w) \\ &= P(M|w, C)P(C|w) \\ &= \frac{P(w|C, M)P(M|C)}{P(w|C)} P(C|w) \end{aligned}$$

here, $P(M|w, C)$ means the emotion distribution when w and C are given. We approximate $P(C|w)$ by the probability that the emotion C is labeled to the tweet t which has w. For example, assume that there are x tweets in which w appears, and y tweets are labeled by C_i among these x tweets. Then, $P(C = C_i|w) = \frac{x}{y}$. $P(M|C)$ is also calculated from number of tweets. For example, if there are z tweets labeled by C_1 and x tweets labeled by "joy", then $P(M = \text{“}joy''|C_1) = \frac{x}{z}$.

For a tweet t which corresponds to $(u_1, u_2, \cdots, u_m)$, we approximates $p(Y|t)$ as follows.

$$\begin{aligned} P(Y|t) &= P(Y|(u_1, u_2, \cdots, u_m)) \\ &= \prod_{i=1,2,\cdots,m} P(Y|w_i)^{u_i} \end{aligned}$$

If $P(Y|t) > Th$ then t has the emotion label of Y.

We call this method "weighted naive Bayes" because $P(C|w)$ looks like a weight for $P(M|w, C)$.

3.3 Two stages naive Bayes

We use four classes of emotions C_i for $i = 1, 2, 3, 4$ which are defined in the previous section. In this method, two threshold value Th and Tc is used. At the first stage, $P(C_i|w)$ for every $i = 1, 2, 3, 4$ is calculated and check them whether $P(C_i|w) > Tc$ or not. If $P(C_i|w) \leq Tc$ then no label $m_j \in C_i$ is labeled to the target tweet. If $P(C_i|w) > Tc$ then select emotion m_i from C_i according to whether $P(m_i|C_i, t) > Th$ or not. The target tweet takes the label m_i When $P(m_i|C_i, t) > Th$. We call this step the second stage. Fig. 1 shows the flow of this method. In

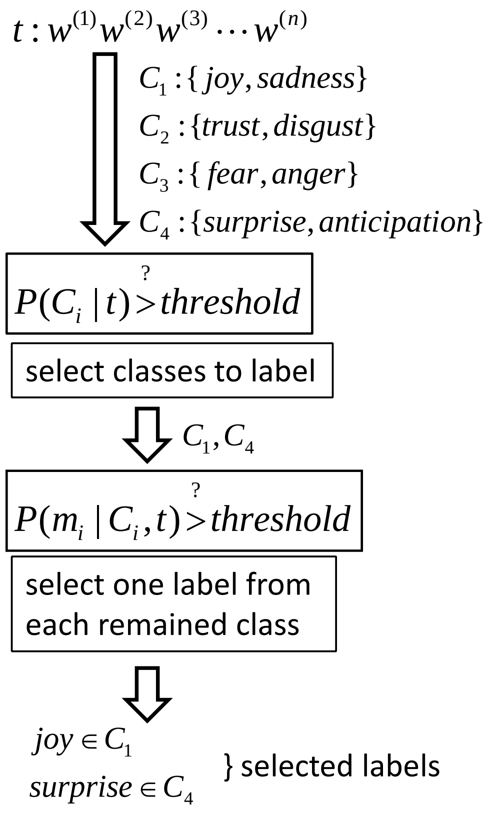

$t : w^{(1)}w^{(2)}w^{(3)} \cdots w^{(n)}$
$C_1 : \{joy, sadness\}$
$C_2 : \{trust, disgust\}$
$C_3 : \{fear, anger\}$
$C_4 : \{surprise, anticipation\}$
$P(C_i \mid t) \overset{?}{>} threshold$
select classes to label
C_1, C_4
$P(m_i \mid C_i, t) \overset{?}{>} threshold$
select one label from each remained class
$joy \in C_1$
$surprise \in C_4$
} selected labels

Fig. 1: The flow of two stage method

this paper, we use $Th = Tc = 0.1$ from some preliminary experiment.

When either Tc or Th is too low, the target tweet may have many labels and it increases the recall but decreases the precision.

3.4 SVM

SVM is a discriminative classifier which is based on margin maximization. In this paper, emotion labeling via SVM is processed as follows.

- Train an SVM for every emotion label which discriminates one emotion from the others. Then, there are eight SVMs and such SVMs are denoted by S_i for $i = 1, 2, \cdots, 8$. The input of each SVM S_i is a vector $(\delta_1, \delta_2, \cdots, \delta_l)$ of a tweet t which expresses the word appearance in t of the vocabulary of learning data. The output of S_i is whether the input tweet has the i-th emotion label or not.
- To predict that a tweet t has i-th emotion or not, make the input vector of t for S_i such that $(\delta_1, \delta_2, \cdots, \delta_l)$. Then, predict the emotion label according to the output of S_i for $i = 1, 2, \cdots, 8$.

SVM has a parameter C which is the weight of slack variables. We examine some values of C and their performances. In this paper, we use linear classifier with slack variables and L_2 norm. “liblinear” is one of the most effective implementation of linear SVM and we use this software for our experiments.

4. Evaluation

4.1 Data description

For experiment, web news and their comments are gathered. comments are tweets which attached to the news article. The news site is news.nicovideo.jp and tweets are processed as follows.

- Re-tweets are all deleted.
- All meaningless spaces and tabs are deleted.
- Comments for other tweets (re-tweets with original comment) are remained.

Then, our data descriptions are as follows.

- Total number of news article : 28
- Total number of tweets : 2075
- The average number of tweets per article : 78.04
- The maximum number of tweets per article : 100
- The minimum number of tweets per article : 12
- The average number of words per tweet : 19.34
- The maximum number of words per tweet : 65
- The minimum number of words per tweet : 1
- The size of vocabulary : 4987

We do not use news article body for learning and evaluation. Learning data only consist of tweets. All these news articles and tweets are in Japanese. Thus, we must do morphological analyze to all tweets. The morphological analyzer by which all tweets are processed is “mecab.” Every word consists of the pair of morpheme and whose tag.

Correct labels of emotions are made by hand. There are twelve persons to make the correct labels. One person can label one emotion per tweet. We call such a label “point.” Every tweet must be labeled by at least two persons to avoid bias. Thus, every tweet has at least two points. Learning data is a pair of a tweet and an emotion vector such that

$$(z_1, z_2, \cdots, z_8)$$

here,

$$z_i = \begin{cases} 0 & t's\ i-th\ emotion\ is\ 0\ point \\ 1 & otherwise \end{cases}$$

i.e. if tweet t is labeled on some emotion, the emotion has more than or equal to 1 point.

The average of points per tweet is 2.52. The maximum point is 9 and the minimum point is 2. The followings are point distribution of correct data.

- joy : 516
- sadness : 756
- trust : 147
- disgust : 1347
- fear : 291
- anger : 936
- surprise : 580
- anticipation : 656

The followings are the number of tweets whose emotion vector has more than 1 point.

- joy : 326 tweets
- sadness : 583 tweets
- trust : 130 tweets
- disgust : 954 tweets
- fear : 217 tweets
- anger : 621 tweets
- surprise : 405 tweets
- anticipation : 474 tweets

The average number of emotions whose point is more than or equals to 1 per tweet is 1.79. The maximum is 5 emotions and the minimum is 1 emotion.

It is expected that there are many points on “disgust” because no one has responsibility to comments of web news. Indeed, “disgust” is the most labeled emotion. We choice the base line that emotion vector is only labeled by “disgust.” Then, the performance of the base line is as follows.

- precision : 0.46
- recall : 0.25
- F value : 0.32

4.2 Experiment and results

For evaluation, experiments for each method with our learning data are executed and the performances are measured. In our all experiments, the cost C of slack variable on SVM is set to $C = 1.0$.

4.2.1 Simple cross validation

Table 1: Simple 5-fold cross validation

	simple	weighted	2stage	SVM
precision	0.4590	0.4718	0.4632	0.5610
recall	0.5970	0.6056	0.5818	0.5159
F value	0.5190	0.5304	0.5158	0.5375

Fig. 2 and Table 1 show the results of our methods using a 5-fold cross validation. The 5-fold is made by the followings.

1) For all tweets of one article are divided into 5 parts.
2) The evaluation data is the set of every one part of tweets from all articles. Thus, there are $\frac{1}{5}$ of all tweets.

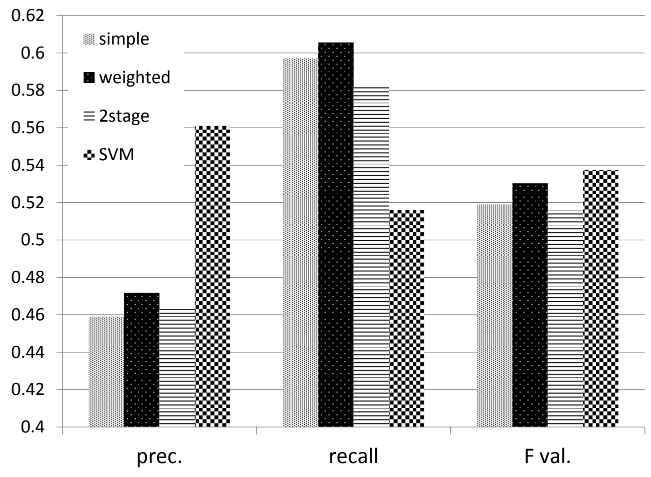


Fig. 2: Simple 5-fold cross validation (graph)

3) The learning data is the rest of them. Thus, there are $\frac{4}{5}$ of all tweets.

By this validation, there are at least $\frac{1}{5}$ tweets of one article in the learning data. Thus, any classifiers can obtain trends of every article. The recall is higher than the precision by the SVM, on the other hand, the precision is higher than the recall by naive Bayes based methods. The F value is almost 0.51 to 0.53 but the SVM has the highest performance and weighted naive Bayes has the second performance.

4.2.2 Cross validation among news articles

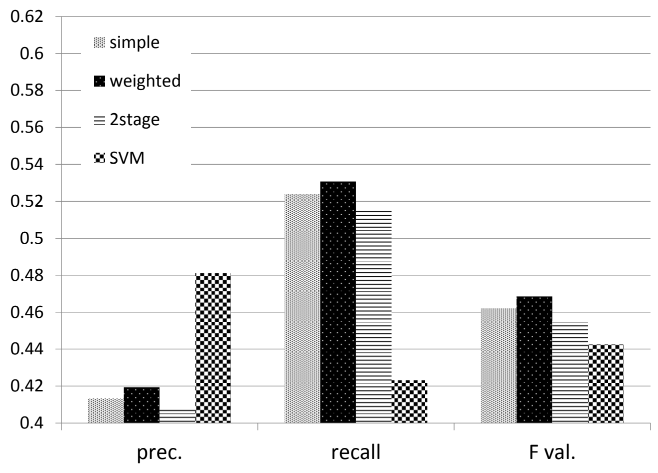


Fig. 3: Leave one file out (graph)

Fig. 3 and Table 2 show the results by leave one file out cross validation. The learning data and evaluation data are made by as follows.

Table 2: Leave one file out

	simple	weighted	2stage	SVM
precision	0.4133	0.4193	0.4079	0.4810
recall	0.5238	0.5307	0.5147	0.4231
F value	0.4620	0.4685	0.4551	0.4425

1) The evaluation data is all tweets of one article.
2) The learning data is all tweets of the all rest articles.

By the SVM, the precision is higher than its recall in this validation. The recall is higher than the precision by naive Bayes based methods. From these facts, the SVM tends to label less than naive Bayes methods. The F value of all naive Bayes based methods are higher than that of the SVM. We think this is caused that the SVM labels few emotions then one miss label decreases the F value comparing to naive Bayes based methods.

4.2.3 Closed data test

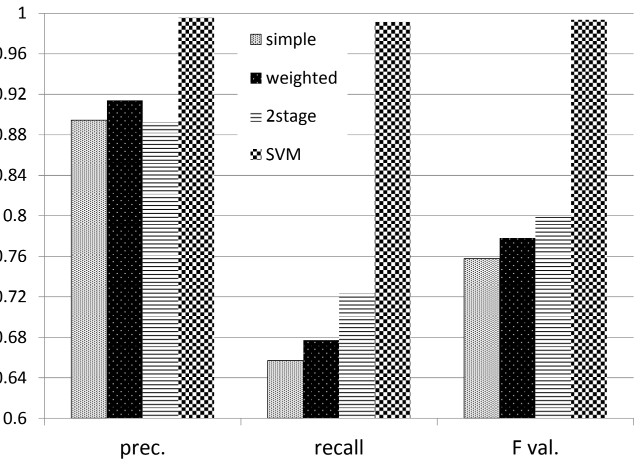


Fig. 4: Closed test (graph)

Table 3: Closed test

	simple	weighted	2stage	SVM
precision	0.8944	0.9139	0.8924	0.9957
recall	0.6569	0.6771	0.7231	0.9914
F value	0.7575	0.7779	0.7989	0.9936

Fig. 4 and Table 3 show the result of a closed test. In this test, all data are used for both learning and evaluation. The SVM scores almost 1.0 for the precision, the recall and the F value. From the previous two open data experiments, the F value of the SVM is higher than that of naive Bayes based method if learning data contain the trends of evaluation data. This trend is clear in the closed test.

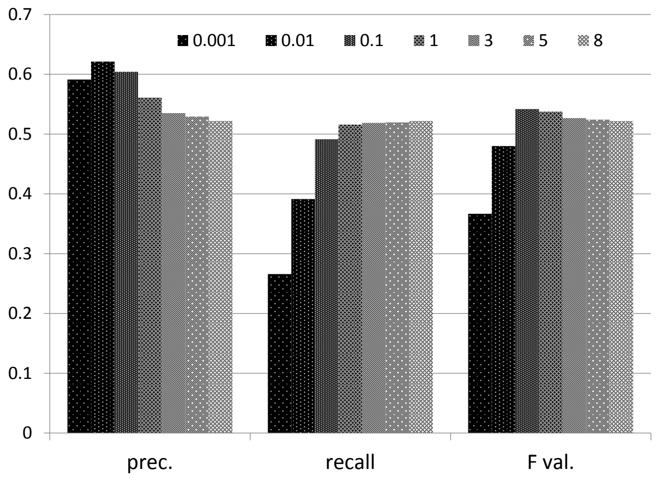


Fig. 5: Slack variable cost and performance (5-fold)

Naive Bayes based methods has different behavior against the previous two experiments. The precision is higher than the recall.

4.3 Soft margin cost on SVM

We show the difference between C values. SVM finds the hyperplane which has the maximum margin. This task is the optimization problem to maximize the function $L(w)$ where w is the normal vector of the hyperplane. Now, the total of slack variables is denoted by S. Including slack variables, the optimization function is $L(w) + C \cdot S$ where C is the cost of soft margins. We investigate the performance when C value is changed.

Table 4: Slack variable cost and performance (5-fold)

cost	0.001	0.01	0.1	
precision	0.5911	0.6212	0.6041	
recall	0.2659	0.3913	0.4914	
F value	0.3666	0.4800	0.5419	
cost	1.0	3.0	5.0	8.0
precision	0.5610	0.5350	0.5292	0.5218
recall	0.5159	0.5186	0.5194	0.5218
F value	0.5375	0.5266	0.5242	0.5217

Table 5: Slack variable cost and performance (leave one file out)

cost	0.001	0.01	0.1	
precision	0.5453	0.5457	0.5195	
recall	0.2189	0.2998	0.3905	
F value	0.3071	0.3767	0.4370	
cost	1.0	3.0	5.0	8.0
precision	0.4810	0.4620	0.4616	0.4462
recall	0.4231	0.4288	0.4354	0.4314
F value	0.4425	0.4367	0.4394	0.4300

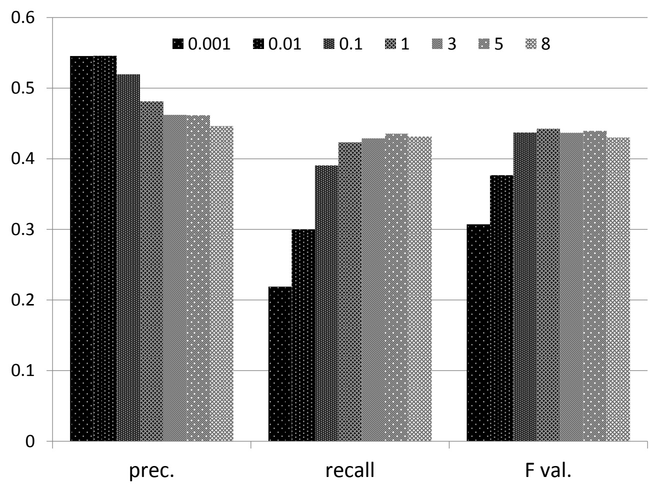


Fig. 6: Slack variable cost and performance (leave one file out)

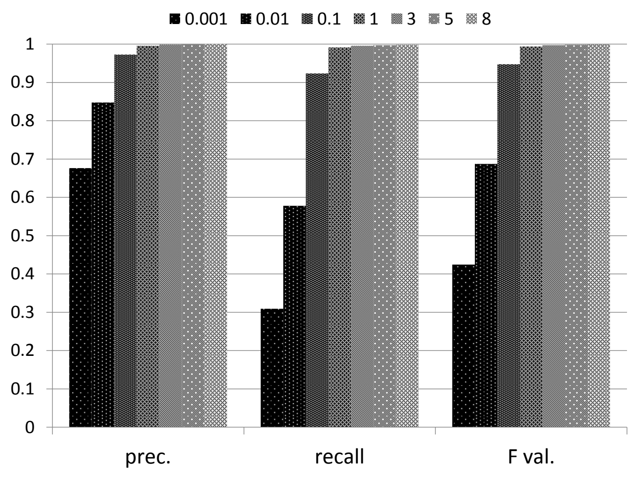


Fig. 7: Slack variable cost and performance (closed)

Fig. 5, 6 and 7 are the performances of the SVM by the 5-fold test, leave one file out and closed test, respectively. Table 4, 5 and 6 shows the values of the performance.

On the open data test (5-fold and leave one file out), the F value is the highest at $C = 1.0$. When the soft margin cost C is decreased, i.e. soft margins can be useable lightly, the recall value is rapidly decreased. On the other hand, the precision is not decreased so fast. This means that the hyperplane will be placed far from the center of learning vectors when the large soft margins are allowed. Then, the classifier tends to labels many emotions. From these figures and tables, $C = 1.0$ leads the best performance for our experiment.

Table 6: Slack variable cost and performance (closed)

cost	0.001	0.01	0.1	
precision	0.6763	0.8476	0.9726	
recall	0.3091	0.5779	0.9234	
F value	0.4242	0.6872	0.9474	
cost	1.0	3.0	5.0	8.0
precision	0.9957	0.9986	0.9996	0.9997
recall	0.9914	0.9955	0.9969	0.9979
F value	0.9936	0.9970	0.9983	0.9988

5. Conclusions

We introduced two naive Bayes based method for emotion estimation of tweets which are appended as comments to news articles. The new method uses the fact that the emotions can be classified into four classes and each of them consists of the two opposite emotions. The new methods are called the "weighted naive Bayes" and the "two stage naive Bayes".

Then, we compared their performances with the simple naive Bayes method and the SVM by the evaluation experiments. From these results, the SVM marks high performance when the learning data contains the trends of the evaluation data. Naive Bayes based methods have robustness to learning settings. The weighted naive Bayes is the best performance among naive Bayes based methods. The performance of this method marked about 5.5% more than that of the SVM in leave one file out test, but 1.3% less in the simple cross validation.

For the future study, decision of Th and Tc are important problem to use our methods. Since two stage naive Bayes uses both of Th and Tc, effective threshold decision method is more important for this classifier. Any other classifier can be applied at every stage of the two stage naive Bayes method. This problem is also remained for the future study.

In this paper, news article body has not been used. If we can make some bias of emotion distribution, it will contribute to the performance of our methods.

Acknowledgment

This study was partially supported by JSPS KAKENHI Grant Number 24500296.

References

[1] A. Kurozumi, Y. Murakami, M. Tokuhisa, J. Murakami, S. Ikebara, "Semantic analysis of emotional verbs in valency pattern dictionary", Proc. of the 2006 IEICE Society Conference, A-13-1, 2006.

[2] R. Tokuhisa, K. Inui, Y. Matsumoto, "Emotion classification using massive examples extracted from the Web", IPSJ Journal, vol.50, no.4, pp.1365–1374, 2009.

[3] T. Ogawa, K. Matsumoto, F. Ren, "About emotion estimation of "Emonyu" short sentence", IPSJ SIG Technical Report, vol.2010-NL0195, no.2, pp.1–6, 2010.

[4] Y. Tajima, "Emotion Estimation of multilabeled comments on web news sites", IPSJ SIG Technical Report, vol.2013-MPS-95, no.18, pp.1–6, 2013.

[5] R. Plutchik, "The emotions", University Press of America, Lanham MD, 1962.

Gröbner Basis of Non-Negative Matrix Factorization and Feature Extraction of Cross-Site Scripting Attacks

T. Matsuda
Department of Computer Science,
Shizuoka Institute of Science and Technology,
2200-2 Toyosawa Fukuroi, Shizuoka, Japan

Abstract—*Non negative matrix factorization (NMF) is the method to decompose a non negative matrix into two non negative matrices. NMF is used in many fields for extracting some features and the effectiveness had been recognized in the application for the analysis of sound signal or text mining. However, its theoretical verification is not easy because NMF is not an unique. In this paper, we investigate on an affine algebraic variety of NMF, and propose the feature extraction method of cross-site scripting attacks.*

Keywords: Non negative matrix factorization, Affine algebraic variety, Gröbner Basis, Feature extraction, Cross-Site scripting

1. Introduction

In many fields on information theory, a feature extraction plays an important role. NMF is known as one of the feature extraction methods, and it is used in many fields such as text mining [1], bioinformatics [2] [3], spectral data analysis [4] and image processing [5], and the effectiveness has been widely recognized. Let X, A and B be matrices whose elements are non negative rational number. The goal of NMF is to obtain the decomposition

$$X = AB.$$

We can see easily that this decomposition is not unique. A specific example is

$$\begin{aligned}\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} &= \begin{pmatrix} \frac{1}{2} & 1 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} \frac{1}{4} & 1 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 4 & 0 \\ 0 & 1 \end{pmatrix}.\end{aligned}$$

The derivation method of NMF had been studied (for example [6]), but the result of NMF depends strongly on the choice of initial numbers. Therefore, it is not easy to investigate why NMF is useful in real application.

In this paper, we investigate the property of an affine algebraic variety of NMF, and propose the feature extraction method of cross-site scripting attacks by using the property. We had already proposed the classification method of SQL injection attacks by using the algebraic property of NMF [7]. In [7], we considered the following decomposition, and classified SQL injection attack by using the information of $(a_{11}a_{12})$ and $(a_{21}a_{22})$.

$$\begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

On the other hand, we consider the decomposition

$$\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \end{pmatrix}\begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \end{pmatrix},$$

and propose the detection method by using the information of

$$\begin{pmatrix} a_{11} & a_{12} \end{pmatrix}$$

and

$$\begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \end{pmatrix}.$$

The rest of this paper is organized as follows. Section 2 summarizes on NMF. Section 3 prepares for algebraically consideration on NMF. Section 4 presents the main result of this study and proposes the detection method of cross-site scripting attacks. Section 5 detects cross-site scripting attacks and Section 6 concludes this study.

2. Non Negative Matrix Factorization

In this section, we will outline the algorithm concerning NMF. Firstly, let us prepare symbols as below.

$X : P \times S$ matrix
$A : P \times R$ matrix
$B : R \times S$ matrix
$x_{ps} : (p, s)$ element of X
$a_{pr} : (p, r)$ element of A
$b_{rs} : (r, s)$ element of B

Here, $1 \leq p \leq P$, $1 \leq r \leq R$ and $1 \leq s \leq S$. Let us consider the following decomposition.

$$X = AB.$$

Since the above decomposition of matrix has not uniqueness, some appropriate methods for extracting some feature of a given data are required. A lot of algorithms for realizing such decomposition had been studied [6] [8]. In this section, we will introduce the algorithm of Lee and Seung's multiplicative update rule [6]. The multiplicative update rule is given by

$$\begin{aligned} a_{pr} &= a_{pr}\frac{[XB^{\mathrm{T}}]_{pr}}{[ABB^{\mathrm{T}}]_{pr}} \\ b_{rq} &= b_{rq}\frac{[A^{\mathrm{T}}X]_{rq}}{[A^{\mathrm{T}}AB]_{rq}}. \end{aligned}$$

Here, $[\cdot]$ is matrix and $[\cdot]_{ij}$ is (i,j) element of the matrix $[\cdot]$. This multiplicative update rule is derived from the following optimization problem.

$$\begin{aligned} &\text{minimize} \quad ||X - AB||^2 \\ &\text{subject to} \quad a_{pr} \geq 0, b_{rq} \geq 0. \end{aligned}$$

Here, $||\cdot||$ is a Frobenius norm. The calculation result of the multiplicative update rule depends on an initial value of a_{pr} and b_{rq}.

From now on, we will give a calculation example of the multiplicative update rules. Let

$$X = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

and, we iterated 50 times the multiplicative update rules. If we set initial values as below.

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix},$$

$$B = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}.$$

Then, we have the following decomposition.

$$X = \begin{pmatrix} 0.30 & 0.83 \\ 0.64 & 0.00 \end{pmatrix}\begin{pmatrix} 0.01 & 1.55 \\ 1.20 & 0.64 \end{pmatrix}.$$

If we set initial values as below.

$$A = \begin{pmatrix} 1 & 0.1 \\ 0.1 & 10 \end{pmatrix},$$

$$B = \begin{pmatrix} 10 & 100 \\ 1 & 10 \end{pmatrix}.$$

Then, we have the following decomposition.

$$X = \begin{pmatrix} 0.01 & 0.04 \\ 0.00 & 0.11 \end{pmatrix}\begin{pmatrix} 84.24 & 52.63 \\ 0.02 & 9.25 \end{pmatrix}.$$

From the above two calculation example, we can see that the result of the multiplicative update rules show different trends by the choice of initial values. Although the NMF is not unique, NMF should have some rule. In this next section, we will investigate the affine algebraic variety of NMF.

3. Affine Algebraic Variety of NMF

In this paper, we investigate of the following NMF.

$$X = AB, \tag{1}$$

where

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \end{pmatrix},$$
$$A = \begin{pmatrix} a_{11} & a_{12} \end{pmatrix}$$

and

$$B = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \end{pmatrix}.$$

This decomposition is obtained from the following equations.

$$f_i = a_{11}b_{1i} + a_{12}b_{2i} - x_{1i}$$

for $i = 1, 2, \cdots, n$. Let $\mathbf{Q}_{\geq 0}$ be a set of non negative rational numbers, and

$$S = \mathbf{Q}_{\geq 0}[a_{11}, a_{12}, b_{11}, b_{12}, \cdots, b_{1n}, b_{21}, b_{22}, \cdots, b_{2n}]$$

be a polynomial ring with non negative rational number coefficients. Then, we see that the NMF of Eq. (1) corresponds to the affine algebraic variety

$$V = V(f_1, f_2, \cdots, f_n) = \{P \in \mathbf{Q}_{\geq 0}^{2n+2} \mid f_i(P) = 0, 1 \leq i \leq n\}.$$

Let us consider the defining ideal of V

$$I(V) = \{g \in S \mid g(P) = 0, P \in V\}$$

to investigate the property of V.

Let

$$a^{\alpha} = a_{11}^{\alpha_1}a_{12}^{\alpha_2}b_{11}^{\alpha_3}\cdots b_{2n}^{\alpha_{2n+2}}$$

be a monomial in the polynomial ring S. The multi-index $\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_{2n+2})$ corresponds to $\mathbf{Z}_{\geq 0}^{2n+2}$, where $\mathbf{Z}_{\geq 0}$ is a set of non negative integers. We define an order $\alpha > \beta$ if and only if $\alpha_1 = \beta_1, \cdots, \alpha_{i-1} = \beta_{i-1}$ and $\alpha_i > \beta_i$ for some i $(1 \leq i \leq 2n+2)$ and $\alpha, \beta \in \mathbf{Z}_{\geq 0}^{2n+2}$. This order is called as a monomial order, and it has the following properties.

- There exists a minimum element in an arbitrary subset of $\mathbf{Z}_{\geq 0}^{2n+2}$.
- It holds $\alpha > \beta, \alpha = \beta$ or $\alpha < \beta$.
- If $\alpha > \beta$, then $\alpha + \gamma > \beta + \gamma$ for any $\gamma \in \mathbf{Z}_{\geq 0}^{2n+2}$.

Here, we assumed

$$\begin{aligned} \alpha + \gamma &= (\alpha_1, \alpha_2, \cdots, \alpha_{2n+2}) + (\gamma_1, \gamma_2, \cdots, \gamma_{2n+2}) \\ &= (\alpha_1 + \gamma_1, \alpha_2 + \gamma_2, \cdots, \alpha_{2n+2} + \gamma_{2n+2}), \end{aligned}$$

and $a^{\alpha} = 0$ if $\alpha = (0, 0, \cdots, 0)$. In this study, we assume the following lexicographic order:

$$a_{11} > a_{12} > b_{11} > \cdots > b_{1n} > b_{21} > \cdots > b_{2n}.$$

Let $\mathrm{LT}(f)$ be the maximum term of the polynomial f in the lexicographic order. $\mathrm{LT}(f)$ is called as a leading term of f. Moreover, we define

$$\mathrm{LT}(I) = <\mathrm{LT}(g) \mid g \in I>$$

for any ideal $I \subset S$.

Definition 1: Let I be an ideal of S and $\{h_1, h_2, \cdots, h_k\}$ be generator of I. If

$$\mathrm{LT}(I) =< \mathrm{LT}(h_1), \mathrm{LT}(h_2), \cdots, \mathrm{LT}(h_k) >$$

holds, then $\{h_1, h_2, \cdots, h_k\}$ is called a Gröbner basis of I.

Definition 2: Let d be a non negative integer, and $S_{\leq d}$ be the set of total degree is less than or equal to d in S. We define

$$I_{\leq d} = I \cap S_{\leq d}.$$

Then, the affine Hilbert function of I is defined by

$$\mathrm{HF}_I(d) = \dim(S_{\leq d}/I_{\leq d}).$$

The affine Hilbert function is the function on d.

Definition 3: The polynomial which equals to $\mathrm{HF}_I(d)$ for sufficiently large d is called the affine Hilbert polynomial of I. We denote the affine Hilbert polynomial of I by $\mathrm{HP}_I(d)$.

Theorem 1: Let I be an ideal on S. The dimension of I is defined by the degree of $\mathrm{HP}_I(d)$.

We can see the proof of Theorem 1 in [9].

In general, there is no guarantee that the set $\{h_1, h_2, \cdots, h_k\}$ of the generator of I is the Gröbner basis of I. However, it is well known that Gröbner basis is obtained by using the Buchberger's Algorithm. The following $S - polynomial$ is fundamental to get Gröbner basis. The $S - polynomial$ of f and g is defined by

$$S(f, g) = \frac{\mathrm{LCM}(f, g)}{\mathrm{LT}(f)} f - \frac{\mathrm{LCM}(f, g)}{\mathrm{LT}(g)} g,$$

where $\mathrm{LC}(f)$ is the coefficient of $\mathrm{LT}(f)$ and $\mathrm{LCM}(f, g)$ is the least common multiple of $\mathrm{LC}(f)\mathrm{LT}(f)$ and $\mathrm{LC}(g)\mathrm{LT}(g)$.

4. Main Result

4.1 Main Theorem

In this section, we will prove the following theorem.

Theorem 2: The dimension of V is $2n - 1$.

(Outline of Proof of Theorem 2)

Firstly, we will compute the Gröbner basis of the ideal

$$I =< f_1, f_2, \cdots, f_n >,$$

where $f_i = a_{11}b_{1i} + a_{12}b_{2i} - x_{1i}$ and $i = 1, 2, \cdots, n$. For $1 \leq i < j \leq n$, let

$$f_{ij} = \frac{\mathrm{LCM}(f_i, f_j)}{\mathrm{LT}(f_i)} f_i - \frac{\mathrm{LCM}(f_i, f_j)}{\mathrm{LT}(f_j)} f_j,$$

and

$$J =< f_1, \cdots, f_n, f_{12}, \cdots, f_{n-1,n} > .$$

Then, we can see that

$$S(f_i, f_{ij}) = S(f_j, f_{ij}) = S(f_i, f_{kl}) = S(f_{ij}, f_{kl}) = 0$$

in S/J, for $1 \leq i < j < k < l \leq n$. Therefore,

$$\{f_i, f_{ij} | 1 \leq i < j \leq n\}$$

is the Gröbner basis of I. From this, we can see that the degree of the affine Hilbert polynomial of $I(V)$ is $2n + 1$. (Q. E. D).

Theorem 2 shows the freedom degree of NMF in Eq. (1). In order to decompose matrices uniquely, we define some equations on $a_{11}, a_{12}, b_{11}, \cdots, b_{2n}$ depending on $x_{11}, \cdots, x_{1n}$. In this study, we will consider the following NMF.

$$\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \end{pmatrix}.$$

4.2 Proposed Detection Algorithm

Firstly, let us define $x_{11}, x_{12}, \cdots, x_{1n}$. We define symbols as follows:

l_i	: input string $(j = 1, 2, \cdots)$
$\lvert l_j \rvert$	: strength of l_j
L	: set of l
s_i	: character in L $(i = 1, 2, \cdots)$
$\lvert s_i \rvert$	: total number of s_i in L

Let

$$x_i = \left[1000 \cdot \frac{|s_i|}{\sum_{j=1}^{J} |l_j|} \right].$$

Here, $[y]$ is the greatest integer that is less than or equal to y. We multiplied 1000 to get double digits integer. We collected 30 cross-site scripting attacks from [10] [11] and generated 50 normal sample. Then, we obtained the following Table 1. Here, normal sample are composed of the input of name, address, e-mail address, phone number, html grammar and Wiki grammar. Characters $s_1, \cdots, s_{11}$

Table 1: Characters in Cross Site Scripting Attacks

Variable	Characters	Value
x_1	" (double quotation mark)	36
x_2	< (less than sign)	25
x_3	> (grater than sign)	25
x_4	/ (slash)	24
x_5	' (single quotation mark)	22
x_6	space	21
x_7	) (right parenthesis)	19
x_8	((left parenthesis)	19
x_9	= (equal)	19
x_{10}	¥ (yen sign)	18
x_{11}	; (semicolon)	12
x_{12}	- (hyphen)	31
x_{13}	| (vertical bar)	20
x_{14}	% (percent)	18
x_{15}	+ (plus)	11
x_{16}	* (asterisk)	11
x_{17}	& (ampersand)	5
x_{18}	{ (left brace)	3
x_{19}	} (right brace)	3
x_{20}	[(left bracket)	3
x_{21}	] (right bracket)	3
x_{22}	? (question mark)	1

and $s_{12}, \cdots, s_{22}$ occurred frequently in our collected attack sample, respectively.

Secondly, let us define a_{11} and a_{12}. We define a_{11} and a_{12} as attack feature element and normal feature element, respectively. We compute a_{11} and a_{12} in the following way.

$$\begin{aligned} a_{11} &= \left[\frac{x_1 + \cdots + x_{11}}{10}\right] \\ a_{12} &= \left[\frac{x_{12} + \cdots + x_{22}}{10}\right] \end{aligned}$$

We multiplied $\frac{1}{10}$ to get double digits integer.

Finally, let us define $b_{11}, \cdots, b_{2,22}$ in the following way. Let L_A and L_N be the set of attack sample and the set of normal sample, respectively. For $i = 1, 2, \cdots, 22$, we compute

$$\begin{aligned} x_i^{(a)} &= \left[1000 \cdot \frac{|s_i|}{\sum_{l \in L_A} |l|}\right] + \epsilon \\ x_i^{(n)} &= \left[1000 \cdot \frac{|s_i|}{\sum_{l \in L_N} |l|}\right] + \epsilon, \end{aligned}$$

where $\epsilon = 0.001$. By adding ϵ, we got $x_i^{(a)} > 0$ and $x_i^{(n)} > 0$. From Theorem 2, we can see that the dimension of $I(V)$ becomes 0 by adding the following equations:

$$\frac{b_{1i}}{b_{2i}} = \frac{x_i^{(a)}}{x_i^{(n)}}.$$

Therefore, we can obtain unique decomposition by computing the above equations.

5. Detection of Cross-site Scripting

In this section, we will detect cross-site scripting attacks using the process in section 4.1.

5.1 Cross-site Scripting Attack

Cross-site scripting attacks are executed by injecting code such as JavaScript, VBScript, ActiveX or HTML, and attackers can gather individual information, change user settings and show a false advertising for user by using this attack. Cross-site scripting attacks come from the HTTP request, form fields on web page or cookies. If users get caught in a trap that attackers set up, then the data of the trap are sent to web application. And when the data come back to user, the attack is executed. The following are specific sample of cross-site scripting attacks.

l_1 <IMG SRC="javascript:alert('XSS');">
l_2 perl -e 'print "<SCRIPT>alert(ẌSS̈)</SCRIPT>";' > out
l_3 ̈;alert('XSS');//
l_4 '<STYLE>BODY-moz-binding:url("http://ha.ckers.org/xssmoz.xml#xss")</STYLE>

The above 4 sample will be used for the detection test later. Moreover, the following 5 normal sample will be also used for the detection test.

l_5 123-4567
l_6 '2010nen11gatsu10nichi
l_7 1-2-3RoppongiMinatoku
l_8 [graph:id:text:text(:image)]
l_9 |!text|

5.2 Learning Process

In the learning process of our proposed model, characters are extracted to detect attacks. From our collected sample, we extracted 11 characters $s_1, s_2, \cdots, s_{11}$ that well appear in attack sample, and 11 characters $s_{12}, s_{13}, \cdots, s_{22}$ that well appear in normal sample in the following way. We call $\{s_1, s_2, \cdots, s_{11}\}$ (resp. $\{s_{12}, s_{13}, \cdots, s_{22}\}$) attack feature (resp. normal feature) in this paper. The character s_i is classified attack feature (resp. normal feature) if $x_i^{(a)} \geq x_i^{(n)}$ (resp. $x_i^{(a)} < x_i^{(n)}$).

5.3 Detection Rule

By using the process of Section 4.1, we can compute the following two matrices from the given data $\{x_{11}, x_{12}, \cdots, x_{1n}\}$.

$$\begin{aligned} A &= \begin{pmatrix} a_{11} & a_{12} \end{pmatrix} \\ B &= \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \end{pmatrix}. \end{aligned}$$

Firstly, we explain on the role of the matrix A. The element a_{11} of A is determined by appearance frequency of attack feature $s_1, s_2, \cdots, s_{11}$. Similarly, a_{12} of A is determined by appearance frequency of normal feature $s_{12}, s_{13}, \cdots, s_{22}$. Therefore, it may be said that input l is attack (resp. normal) if $a_{11} > a_{12}$ (resp. $a_{11} < a_{12}$).

On the other hand, it would appear that the first row of B (resp. the second row of B) shows the feature of attack (resp. the feature of normal). If $a_{11} = a_{12}$, we detect in the following process. Let

$$d_A(b_{1i}, b_{2i}) = \begin{cases} 1 & (b_{1i} \geq b_{2i}) \\ 0 & (b_{1i} < b_{2i}), \end{cases}$$

for $i = 1, 2, \cdots, 11$, and

$$d_N(b_{1i}, b_{2i}) = \begin{cases} 1 & (b_{1i} \leq b_{2i}) \\ 0 & (b_{1i} > b_{2i}), \end{cases}$$

for $i = 12, 13, \cdots, 22$. Then, we detect l as attack (resp. normal) if $b_A \geq b_N$ (resp. $b_A < b_N$), where

$$\begin{aligned} b_A &= \frac{\sum_{i=1}^{11} d_A(b_{1i}, b_{2i})}{11} \\ b_N &= \frac{\sum_{i=12}^{22} d_N(b_{1i}, b_{2i})}{11} \end{aligned}$$

We will explain by using specific samples.

5.3.1 Example 1

For test sample l_1, we have the following decomposition.

$$X = (2,1,1,0,2,1,1,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0)$$

$$\begin{aligned} A &= \begin{pmatrix} 1 & 0 \end{pmatrix} \\ B_1 &= \begin{pmatrix} 2 & 1 & 1 & 0 & 2 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0.2 & 0 & 1.6 & 0 & 0.5 & 0.5 & 0.2 & 0 & 0 \end{pmatrix} \\ B_2 &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.1 & 1.1 & 0 & 0 & 0 & 0 & 0 & 0.1 \end{pmatrix}, \end{aligned}$$

where $B = (B_1 B_2)$. In this case, l_1 is detected as attack because $a_{11} > a_{12}$. From the result of B_1 and B_2, we can see that values of the element in the first row of B_1 are greater than values of the element of the second row of B_1.

5.3.2 Example 2

For test sample l_8, we have the following decomposition.

$$X = (0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,1,1,0)$$

$$\begin{aligned} A &= \begin{pmatrix} 0.001 & 0.001 \end{pmatrix} \\ B_1 &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 679 & 679 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 321 & 321 & 0 & 0 & 0 \end{pmatrix} \\ B_2 &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 250 & 250 & 0 \\ 0 & 0 & 0 & 1.1 & 1.1 & 0 & 0 & 0 & 750 & 750 & 0.1 \end{pmatrix}, \end{aligned}$$

where $B = (B_1 B_2)$. In this case, l_1 is detected as normal because $a_{11} = a_{12}$ but $b_A < b_N$. From the result of B_1 and B_2, we can see that values of the element in the second row of B_2 are greater than values of the element of the first row of B_2.

5.4 Simulation Result

Here, we summarize the result of the detection test. In this simulation, we extracted attack feature and normal feature from 30 attack sample and 50 normal sample, and we prepared 4 attack sample and 5 normal sample for the detection test shown in Section 5.1.

To show the effectiveness of our proposed detection method, we compared our proposed method with the detection method of Naive Bayes classifier. Let $P_A(s_i)$ and $P_N(s_i)$ be the probability that s_i appears in attack and normal, respectively. We compute $P_A(s_i)$ and $P_N(s_i)$ in the following way. For $i = 1, 2, \cdots, 22$, let

$$\begin{aligned} P_A(s_i) &= \frac{x_i^{(a)} + 0.001}{\sum_{j=1}^{J}(x_i^{(a)} + 0.001)} \\ P_N(s_i) &= \frac{x_i^{(n)} + 0.001}{\sum_{j=1}^{J}(x_i^{(n)} + 0.001)}. \end{aligned}$$

By adding 0.001, $P_A(s_i)$ and $P_N(s_i)$ become always positive value. Assume an input l including $\{s_{i_1}, s_{i_2}, \cdots, s_{i_K}\}$. We detect l as attack (resp. normal) if $\prod_{k=1}^{K} P_A(s_{i_k}) \geq \prod_{k=1}^{K} P_N(s_{i_k})$ (resp. otherwise).

Table 2 shows the detection result of our proposed method and Naive Bayes method. The detection results of our proposed method and Naive Bayes method were 100%. To investigate the point of difference between our proposed method and Naive Bayes method, we prepared two obfuscation attack sample l_{10} and l_{11}. The following the part of the obfuscation attack sample.

Table 2: Detection Result

Input	Proposed Method	Naive Bayes
l_1 (attack)	attack	attack
l_2 (attack)	attack	attack
l_3 (attack)	attack	attack
l_4 (attack)	attack	attack
l_5 (normal)	normal	normal
l_6 (normal)	normal	normal
l_7 (normal)	normal	normal
l_8 (normal)	normal	normal
l_9 (normal)	normal	normal

```
$=~[];$={___:++$,$$$$:(![]+"")[$],
```

Table 3: Detection Result of obfuscation attack sample

Input	Proposed Method	Naive Bayes
l_{10} (attack)	attack	normal
l_{11} (attack)	attack	normal

Table 3 shows the detection result of obfuscation attack sample. Our proposed method could detect two obfuscation attack sample, but Naive Bayes method judged these sample as normal. From the result of NMF, we could see that the characters of single quote and double quote have important role to detect obfuscation attack sample.

6. Conclusions

In this paper, we investigated the algebraic property of NMF, and proposed the detection method of cross-site scripting attacks by using the algebraic property of NMF. Moreover, by comparing to the method of Naive Bayes, we showed the effectiveness of our proposed method. However, we could not collect sufficient amounts of attack sample. So, we need to collect sufficient amounts of attack sample, and to do same simulation of this study. This is our important future work.

References

[1] B. Murray and B. Murray, *Email Surveillance Using Non-negative Matrix Factorization*, Computational and Mathematical Organization Theory 11 (3), pp. 249-264, 2005.

[2] H. Kim and H. Park, *Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis*, Bioinformatics 23 (12), pp.1495-1502, 2007.

[3] K. Devarajan, *Nonnegative Matrix Factorization: An Analytical and Interpretive Tool in Computational Biology*, PLoS Computational Biology 4 (7), 2008.

[4] M. W. Berry, M. Browne, A. N. Langville, V. P. Pauca and R. J. Plemmons, *Algorithms and applications for approximate nonnegative matrix factorization*, Computational Statistics and Data Analysis, 2006.

[5] D. D. Lee and H. S. Seung, *Learning the parts of objects by non-negative matrix factorization*, Nature 401(6755), pp.788-791, 1999.

[6] D. D. Lee and H. S. Seung, *Algorithms for Non-negative Matrix Factorization*, Advances in Neural Information Processing Systems 13: Proceedings of the 2000 Conference. MIT Press. pp. 556-562, 2001.

[7] T. Matsuda, *Solution Space of Non Negative Matrix Factorization and Consideration of Feature Extraction on Web Application Attacks*, 2014 International Conference on Information Science, Electronics and Electrical Engineering (Accepted).

[8] J. Kim and H. Park, *Fast Nonnegative Matrix Factorization: An Active-set-like Method and Comparisons*, SIAM Journal on Scientific Computing 33 (6), pp.3261-3281, 2011.

[9] D. Cox, J. Little and D. O'Shea, *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*, Springer, 1997.

[10] *An Autonomous Zone*, [Online]. Available: http://anautonomouszone.com/blog/xss-cheat-sheet

[11] *Cross-site-scripting (XSS) Tutorial*, [Online]. Available: http://www.veracode.com/security/xss

Accelerating the Numerical Computation of Positive Roots of Polynomials using Improved Bounds

Kinji Kimura[1], **Takuto Akiyama**[2], **Hiroyuki Ishigami**[3], **Masami Takata**[4], **and Yoshimasa Nakamura**[5]
[1,2,3,5]Graduate School of Informatics, Kyoto University,
Yoshida-honmachi, Sakyo-ku, Kyoto 606-8501, JAPAN
[4]Academic Group of Information and Computer Sciences, Nara Women's University,
Kita-Uoya-Nishi-Machi, Nara 630-8506, JAPAN
[1]kkimur@amp.i.kyoto-u.ac.jp, [2]akiyama@amp.i.kyoto-u.ac.jp, [3]hishigami@amp.i.kyoto-u.ac.jp,
[4]takata@ics.nara-wu.ac.jp, [5]ynaka@i.kyoto-u.ac.jp

Abstract— *The continued fraction method for isolating the positive roots of a univariate polynomial equation is based on Vincent's theorem, which computes all of the real roots of polynomial equations. In this paper, we propose two new lower bounds which accelerate the fraction method. The two proposed bounds are derived from a theorem stated by Akritas et al., and use different pairing strategies for the coefficients of the target polynomial equations from the bounds proposed by Akritas et al. Numerical experiments show that the proposed lower bounds are more effective than existing bounds for some special polynomial equations and random polynomial equations, and are competitive with them for other special polynomial equations.*

Keywords: continued fraction method, Vincent's theorem, local-max bound, first-λ bound

1. Introduction

The real roots of univariate polynomial equations are more useful than the imaginary roots for practical applications in various engineering fields. Thus, this paper focuses on the computation of all real roots of polynomial equations. For polynomial equations without multiple roots, we can isolate each root into a specific interval. The accuracy of the isolated real roots can be easily enhanced using the bisection method.

The continued fraction (CF) method for isolating the positive roots of univariate polynomial equations is based on Vincent's theorem [2], [11]. This method isolates each positive root using Descartes' rule of signs [3], which focuses on the coefficients of the polynomial equations, and can be accelerated by an origin shift. Thus, several coefficients of a polynomial equation are transformed into nonzero coefficients, even in the case of sparse polynomial equations that have many zero coefficients. The Krawczyk method [8], which is based on numerical verification, was developed to isolate the positive roots of polynomial equations which have many zero coefficients. In this paper, we focus on the CF method for isolating the positive roots of polynomial equations which have many nonzero coefficients.

To accelerate the CF method, the choice of the origin shift is important. For the shift value, we should use a lower bound of the smallest positive root of the target polynomial. In other words, we must compute this lower bound to accelerate the CF method. We can obtain the lower bound of positive roots of a polynomial equation from the upper bound of the replaced polynomial equation corresponding to the original equation. The Cauchy bound [9] and the Kioustelidis bound [7] are well-known upper bounds of the positive roots of polynomial equations, but these bounds are known to produce overestimates in some cases. Akritas *et al.* have given a generalized theorem including the Cauchy bound and the Kioustelidis bound [1]. Using pairing strategies derived from the generalized theorem, they proposed new upper bounds called the first-λ bound, the local-max bound, and the local-max quadratic bound.

In [10], a lower bound generated by Newton's method is proposed. In this paper, we propose two lower bounds that are more effective than that based on Newton's method: the "local-max2" bound and the "tail-pairing first-λ" bound. These are derived from the local-max bound and the first-λ bound, respectively, and use a different pairing strategy from the original bound. The local-max2 bound is always better than or equal to the local-max bound. The tail-pairing first-λ bound is expected to be more suitable for the CF method than the first-λ bound.

The remainder of this paper is organized as follows: Section 2 introduces the CF method based on Vincent's theorem, and Section 3 introduces the bounds proposed by Akritas et al. Section 4 proposes the new lower bounds, before Section 5 reports the results of performance evaluations of the proposed lower bounds. We end with a summary of our conclusions in Section 6.

2. Continued fraction method

In this paper, we discuss the computation of positive roots $x \in \mathbb{R}$ that satisfy the following polynomial equation:

$$f(x) = a_0x^n + a_1x^{n-1} + \cdots + a_{n-1}x + a_n = 0, \qquad (1)$$

where $a_i \in \mathbb{Z}$ and $a_n \neq 0$. Note that we need not consider the case $x = 0$ as one of the roots of $f(x) = 0$, since $a_n = 0$ is satisfied if any real root is equal to 0.

In addition, all the polynomial equations have rational coefficients and multiple roots in the interval $[u, v]$, $(-\infty, v]$, $(u, \infty]$, or $(-\infty, \infty)$ $(u, v \in \mathbb{R})$, and can be transformed into Eq. (1) in $x \in (0, \infty)$ using certain operations. For details, see [10].

2.1 Continued fraction method

The CF method aims to compute the positive roots of a polynomial equation $f(x) = 0$. It is based on Vincent's theorem [2], [11], and isolates the real roots in $(0, \infty)$ using Theorem 1, known as Descartes' rule of signs [3].

Theorem 1 (Descartes' rule of signs): For a polynomial equation

$$f(x) = a_0x^n + \cdots + a_{n-1}x + a_n = 0, \ x \in \mathbb{R}, \ a_i \in \mathbb{R},$$

let W be the number of "changes of sign" in the list of coefficients $\{a_0, \ a_1, \ \ldots, \ a_n\}$, except for $a_i = 0$, and let N be the number of positive roots in $(0, \infty)$. Under these definitions, the following relation holds:

$$N = W - 2h,$$

where h is a non-negative integer.

Using Theorem 1, the number of positive roots of the polynomial equation $f(x) = 0$ is determined as the following conditional branch:

- Case where $W = 0$: $f(x) = 0$ does not have any positive roots in the interval $x \in (0, \infty)$.
- Case where $W = 1$: $f(x) = 0$ has only one positive root in the interval $x \in (0, \infty)$.
- Case where $W \geq 2$: the number of positive roots of $f(x) = 0$ cannot be determined.

If $W = 1$, the isolated interval should be set to $(0, u\_b]$, where u_b denotes the upper bound of the positive roots of $f(x) = 0$. Computation methods for the upper bound of the positive roots of $f(x) = 0$ are described in Section 3.

In the case that $W \geq 2$, the interval $(0, \infty)$ should first be divided into two intervals. Then, Descartes' rule of signs can be applied to each interval. In the CF method, the interval $(0, \infty)$ is divided in $(0, 1)$ and $(1, \infty)$. This division is performed by the replacement $x \to x+1$ and $x \to 1/(x+1)$. Using the replacement $x \to x + 1$, the interval $(0, \infty)$ of the replaced polynomial equation corresponds to the interval $(1, \infty)$ of the original polynomial equation. Similarly, using the replacement $x \to 1/(x + 1)$, the interval $(0, \infty)$ of the replaced polynomial equation corresponds to the interval $(0, 1)$ of the original polynomial equation. The intervals $(1, \infty)$ and $(0, 1)$ do not include the case $x = 1$. To solve for this case, we must check that a constant term of the replaced polynomial equation vanishes after either replacement. In other words, if $a_n = 0$ in the replaced polynomial equation, then $x = 1$ is a root of the original polynomial equation.

Table 1: Synthetic division for $g_5(x)$.

a_0	a_1	a_2	a_3
	a_0	$a_0 + a_1$	$a_0 + a_1 + a_2$
a_0	$a_0 + a_1$	$a_0 + a_1 + a_2$	$a_0 + a_1 + a_2 + a_3$
	a_0	$2a_0 + a_1$	
a_0	$2a_0 + a_1$	$3a_0 + 2a_1 + a_2$	
	a_0		
a_0	$3a_0 + a_1$		

The replacements described above require the coefficients of the replaced polynomial equation to be calculated. This calculation can be performed by synthetic division. As an example, Table 1 shows the calculation of the coefficients of

$$g_5(x) = a_0(x + 1)^3 + a_1(x + 1)^2 + a_3(x + 1) + a_4. \quad (2)$$

As can be seen in Table 1, the coefficients of x^3, x^2, x^1, and x^0 in $g_5(x)$ are a_0, $3a_0 + a_1$, $3a_0 + 2a_1 + a_2$, and $a_0 + a_1 + a_2 + a_3$, respectively. Note that the computational complexity of the synthetic division for obtaining the coefficients of the replaced polynomial equation for a replacement $x \to x + 1$ is $O(n^2)$, where n is the highest order of the polynomial equation.

2.2 Acceleration using a lower bound

The CF method requires many replacement operations $x \to x + 1$ and $x \to 1/(x + 1)$. If the positive roots are much larger than 1, then the execution time increases, since we must repeat many replacement operations $x \to x + 1$. Thus, to decrease the execution time, the lower bound of the smallest positive root of a polynomial equation should be used as a shift.

The procedure for computing the lower bound l_b of $f(x) = 0$ is as follows:

1) Replace x with $1/x$ in $f(x)$.
2) Compute u_b, the upper bound of the positive roots of the replaced polynomial equation.
3) Obtain l_b as $l\_b = 1/u\_b$.

However, the replacement $x \to x + l\_b$ should not always be adopted, since l_b is not sufficiently large to reduce the execution time if $l\_b \leq 1$. Thus, the replacement $x \to x+l\_b$ is only adopted in $f(x)$ if $l\_b > 1$.

3. Computation of the upper bound of positive roots

The Cauchy rule [9] is a well-known idea for computing the upper bound of the positive roots of $f(x) = 0$. The Kioustelidis bound is related to the Cauchy rule [7]. However, both of these bounds are known to overestimate the upper bound in some cases.

To overcome this problem, Akritas et al. derived the following generalized theorem for computing the upper bound of the positive roots of $f(x) = 0$.

Theorem 2 (Akritas, 2006): Let $f(x)$ be a polynomial with real coefficients, and assume $a_0 > 0$. Let $d(f)$ and $t(f)$ denote its degree and number of terms, respectively. In addition, assume that $f(x)$ can be reshaped as follows:

$$f(x) = q_1(x) - q_2(x) + \cdots - q_{2m}(x) + g_6(x), \quad (3)$$

where the polynomials $q_i(x)$, $i = 1, \ldots, 2m$, and $g_6(x)$ have only positive coefficients. Moreover, assume that, for $i = 1, 2, \ldots, m$, we obtain

$$\begin{aligned} q_{2i-1}(x) = c_{2i-1,1}x^{e_{2i-1,1}} + \cdots \\ + c_{2i-1,t(q_{2i-1})}x^{e_{2i-1,t(q_{2i-1})}} \end{aligned} \quad (4)$$

and

$$q_{2i}(x) = b_{2i,1}x^{e_{2i,1}} + \cdots + b_{2i,t(q_{2i})}x^{e_{2i,t(q_{2i})}} \quad (5)$$

where $e_{2i-1,1} = d(q_{2i-1})$ and $e_{2i,1} = d(q_{2i})$, and the exponent of each term in $q_{2i-1}(x)$ is greater than the exponent of each term in $q_{2i}(x)$. If $t(q_{2i-1}) \geq t(q_{2i})$ for all indices $i = 1, 2, \cdots, m$, then the upper bound of the positive roots of $f(x) = 0$ is defined by

$$\begin{aligned} u\_b = \max_{i=1,2,\ldots,m} \Bigg\{ \left(\frac{b_{2i,1}}{c_{2i-1,1}} \right)^{\frac{1}{e_{2i-1,1} - e_{2i,1}}}, \ldots, \\ \left(\frac{b_{2i,t(q_{2i})}}{c_{2i-1,t(q_{2i})}} \right)^{\frac{1}{e_{2i-1,t(q_{2i})} - e_{2i,t(q_{2i})}}} \Bigg\}, \end{aligned} \quad (6)$$

for any permutation of the positive coefficients $c_{2i-1,j}$, $j = 1, 2, \cdots, t(q_{2i-1})$. Otherwise, for each of the indices i for which we obtain $t(q_{2i-1}) < t(q_{2i})$, we break up one of the coefficients of $q_{2i-1}(x)$ into $t(q_{2i}) - t(q_{2i-1}) + 1$ parts, so that $t(q_{2i}) = t(q_{2i-1})$. We can then apply the formula defined in Eq. (6).

Note that the ideas underlying both the Cauchy and Kioustelidis bounds are included in this theorem.

The sharpness of the upper bound is dependent on pairing coefficients from the non-adjacent polynomials $q_{2l-1}(x)$ and $q_{2i}(x)$ for $1 \leq l < i$.

For example, consider the polynomial

$$3x^3 - 5x^2 + 4x + 7. \quad (7)$$

In this case, we can create the pair

$$\left\{3x^3, -5x^2\right\}.$$

However, for the polynomial

$$3x^3 - 5x^2 - 4x + 7, \quad (8)$$

we cannot create the trivial pair, since the polynomial has only one positive coefficient. In this case, since

$$3x^3 = \frac{3}{2}x^3 + \frac{3}{2}x^3 = x^3 + 2x^3,$$

we can create the pair as

$$\left\{\frac{3}{2}x^3, -5x^2\right\}, \left\{\frac{3}{2}x^3, -4x\right\} \text{ or } \left\{x^3, -5x^2\right\}, \left\{2x^3, -4x\right\}.$$

Using Theorem 2, Akritas et al. proposed the "local-max" bound and the "first-λ" bound as follows:

Definition 1 ("local-max"): For a polynomial equation $f(x) = 0$ given by Eq. (1), the coefficient $-a_k$ of the term $-a_k x^{n-k}$ in $f(x) = 0$ is paired with the coefficient $a_m/2^t$ of the term $a_m x^{n-m}$, where a_m is the largest positive coefficient with $0 \leq m < k$ and t denotes the number of times the coefficient a_m has been used.

Definition 2 ("first-λ"): For a polynomial equation $f(x)$ given by Eq. (3) with λ negative coefficients, we first consider all cases for which $t(q_{2i}) > t(q_{2i-1})$ by breaking up the last coefficient $c_{2i-1,t(q_{2i})}$ of $q_{2i-1}(x)$ into $t(q_{2i}) - t(q_{2i-1}) + 1$ equal parts. We then pair each of the first λ positive coefficients of $f(x)$, encountered as we move in non-increasing order of exponents, with the first unmatched negative coefficient.

Note that the computational complexity of these bounds is $O(n)$.

Akritas et al. also proposed the "local-max quadratic" bound as follows:

Definition 3 ("local-max quadratic"): For a polynomial equation $f(x)$ given by Eq. (1), each negative coefficient $a_i < 0$ is "paired" with each of the preceding positive coefficients a_j divided by 2^{t_j}. That is, each positive coefficient a_j is "broken up" into unequal parts, as for the locally maximum coefficient in the local max bound. t_j is initially set to 1, and is incremented each time the positive coefficient a_j is used, and the minimum is taken over all j. Subsequently, the maximum is taken over all i.

From Definition 3, the local-max quadratic bound is computed as

$$u\_b_{\mathrm{LMQ}} = \max_{a_i<0} \min_{a_j>0:j>i} \sqrt[j-i]{-\frac{a_i}{\frac{a_j}{2^{t_j}}}}. \quad (9)$$

Note that the computational complexity of this bound is $O(n^2)$.

4. New upper bounds

In this section, we propose two new upper bounds for the positive roots of a polynomial equation. The first is the "local-max2" bound, and the second is the "tail-pairing first-λ" bound.

4.1 Local-max2 bound

The local-max2 bound is derived from the local-max bound. To compute the local-max bound, the largest positive coefficient a_m is broken up into unequal parts $a_m/2^t(t = 1, \cdots, s+1)$. For the local-max2 bound, we first break up the largest positive coefficient a_m into unequal parts $a_m/2^t(t = 1, \cdots, s)$. Then, since

$$a_m - \left(\frac{a_m}{2} + \cdots + \frac{a_m}{2^s}\right) = \frac{a_m}{2^s}, \tag{10}$$

we use $a_m/2^s$, which is the remaining part of a_m, as the last pair. It is obvious that the local-max2 bound is better than or equal to the local-max bound for all polynomials.

Algorithm 1 describes the implementation of the local-max2 bound. As for the local-max bound, the complexity of computing the local-max2 bound is $O(n)$.

For example, consider the polynomial

$$x^3 + 10^{100}x^2 - x - 10^{100}. \tag{11}$$

For the local-max bound, we pair the terms $\left\{\frac{10^{100}}{2}x^2, -x\right\}$ and $\left\{\frac{10^{100}}{2^2}x^2, -10^{100}\right\}$, and obtain a bound estimate of 2. For the local-max2 bound, we pair the terms $\left\{\frac{10^{100}}{2}x^2, -x\right\}$ and $\left\{\frac{10^{100}}{2}x^2, -10^{100}\right\}$, and obtain a bound estimate of $\sqrt{2}$. As a result, the upper bound of the local-max2 bound is better than that of the local-max bound for the polynomial (11).

4.2 Tail-pairing first-λ

The tail-pairing first-λ bound is derived from the first-λ bound. As for the first-λ bound, if there are more negative than positive coefficients, we first break up the last positive coefficient into several parts. In addition, we pair positive coefficients with unpaired tail negative coefficients when the number of positive coefficients is greater than that of negative coefficients.

Although it is not always better than or equal to the first-λ bound, we expect the tail-pairing first-λ bound to be better for the total number of shifts. The CF method performs the replacement $x \to x + l\_b$ many times. Thus, pairing negative coefficients of low degree with positive coefficients is an important task. In the tail-pairing first-λ bound, we pair high-degree coefficients with low-degree coefficients whenever possible.

There are two strategies for computing the tail-pairing first-λ bound. In the first strategy, we initially pair negative coefficients in the corresponding list, and then pair the tail

Algorithm 1 Implementation of the "local-max2" bound.

```
$cl \leftarrow \{a_n, a_{n-1}, \cdots, a_1, a_0\}$
**if** $n + 1 \leq 1$ **then**
    **return** $u\_b_{\mathrm{LM2}} = 0$
**end if**
$j = n + 1$
$negativeIndices = \{\}$
**for** $i = n$ to 1 step $-1$ **do**
    **if** $cl(i) < 0$ **then**
        $negativeIndices = negativeIndices \cup i$
    **else if** $cl(i) > cl(j)$ **then**
        **if** $count(negativeIndices) > 0$ **then**
            $t = 0$
            $l = count(negativeIndices)$
            **for** $k = 1$ to $l - 1$ **do**
                $t{+}{+}$
                $tempub = (2^t(-cl(negativeIndices(k))$
                        $/cl(j)))^{1/(j-negativeIndices(k))}$
                **if** $tempub > u\_b_{\mathrm{LM2}}$ **then**
                    $u\_b_{\mathrm{LM2}} = tempub$
                **end if**
            **end for**
            $tempub = (2^t(-cl(negativeIndices(l))$
                    $/cl(j)))^{1/(j-negativeIndices(l))}$
            **if** $tempub > u\_b_{\mathrm{LM2}}$ **then**
                $u\_b_{\mathrm{LM2}} = tempub$
            **end if**
        **end if**
        $j = i$
        $negativeIndices = \{\}$
    **end if**
**end for**
**if** $count(negativeIndices) > 0$ **then**
    $t = 0$
    $l = count(negativeIndices)$
    **for** $k = 1$ to $l - 1$ **do**
        $t{+}{+}$
        $tempub = (2^t(-cl(negativeIndices(k))$
                $/cl(j)))^{1/(j-negativeIndices(k))}$
        **if** $tempub > u\_b_{\mathrm{LM2}}$ **then**
            $u\_b_{\mathrm{LM2}} = tempub$
        **end if**
    **end for**
    $tempub = (2^t(-cl(negativeIndices(l))$
            $/cl(j)))^{1/(j-negativeIndices(l))}$
    **if** $tempub > u\_b_{\mathrm{LM2}}$ **then**
        $u\_b_{\mathrm{LM2}} = tempub$
    **end if**
**end if**
**return** $u\_b_{\mathrm{LM2}}$
```

negative coefficients. We call this the "tail-pairing first-λ type-I bound". The second strategy pairs the tail negative coefficients first, and then pairs the negative coefficients in the corresponding list. We call this the "tail-pairing first-λ type-II bound". Algorithm 2 describes the computation of the tail-pairing first-λ type-I bound. As for the first-λ bound, the computational complexity of both tail-pairing first-λ bounds is $O(n)$.

For example, consider the polynomial

$$x^5 + 2x^4 - 3x^3 + 4x^2 - 5x - 10^{10}. \tag{12}$$

For the first-λ bound, we pair the terms $\{x^5, -3x^3\}$, $\{2x^4, -5x\}$, and $\{2x^2, -10^{10}\}$, and obtain a bound estimate of $\sqrt{10^{10}/2} = 50000\sqrt{2}$. For the tail-pairing first-λ type-I bound, we pair the terms $\{x^5, -3x^3\}$, $\{2x^4, -10^{10}\}$, and $\{4x^2, -5x\}$, and find a bound estimate of $\sqrt[4]{10^{10}/2} = 100\sqrt[4]{50}$. For the tail-pairing first-λ type-II bound, we pair the terms $\{x^5, -10^{10}\}$, $\{2x^4, -3x^3\}$, and $\{4x^2, -5x\}$, which gives a bound estimate of $\sqrt[5]{10^{10}} = 100$. Thus, the tail-pairing first-λ bounds are better than the first-λ bound, and the tail-pairing first-λ type-II bound is better than the type-I bound for this polynomial.

5. Numerical experiment

In this section, we present numerical results that evaluate the effect of the proposed bounds.

5.1 Contents of the numerical experiment

To evaluate the effect of the proposed bounds, we implement the CF method with the following bounds:

- FL+LM: $(\max(\mathrm{FL}, \mathrm{LM}))$
- LMQ: local-max quadratic bound
- TPFL-I+LM2: $(\max(\mathrm{TPFL\text{-}I}, \mathrm{LM2}))$
- TPFL-II+LM2: $(\max(\mathrm{TPFL\text{-}II}, \mathrm{LM2}))$

Note that FL, LM, TPFL, and LM2 denote the first-λ bound, local-max bound, tail-pairing first-λ bound, and local-max2 bound, respectively.

As test polynomial equations, the following were used:

- Laguerre: $L_0(x) = 1$, $L_1(x) = 1 - x$, and $L_{n+1}(x) = \frac{1}{n+1}((2n+1-x)L_n(x) - nL_{n-1}(x))$
- Chebyshev-I: $T_0(x) = 1$, $T_1(x) = x$, and $T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$
- Chebyshev-II: $U_0(x) = 1$, $U_1(x) = 2x$, and $U_{n+1}(x) = 2xU_n(x) - U_{n-1}(x)$
- Wilkinson: $W_n(x) = \prod_{i=1}^{n}(x - i)$
- Mignotte: $M_n(x) = x^n - 2(5x - 1)^2$
- Randomized polynomial

The randomized polynomials are defined as

$$f(x) = \prod_{i=0}^{r}(x - x_i)\prod_{j=0}^{s}(x - \alpha_j + i\beta_j)(x - \alpha_j - i\beta_j), \tag{13}$$

Algorithm 2 Implementation of the "tail-pairing first-λ" bound.

```
$cl \leftarrow \{a_n, a_{n-1} \cdots, a_1, a_0\}$
$\lambda \leftarrow$ the number of negative elements of cl
if $n + 1 \le 1$ then
    return $u\_b_{\mathrm{TPFL}} = 0$
end if
$posStartIndex = n + 1$
$negTailIndex = 1$
while $negTailIndex \le n + 1$
     and $cl(negTailIndex) \ge 0$ do
    $negTailIndex + +$
end while
while $\lambda > 0$ do
    while $posStartIndex >= 0$
     and $cl(posStartIndex) \le 0$ do
        $posStartIndex - -$
    end while
    $posEndIndex = posStartIndex + 1$
    while $posEndIndex >= 0$
     and $cl(posEndIndex) \ge 0$ do
        $posEndIndex - -$
    end while
    $negHeadStartIndex = negHeadEndIndex$
    $negHeadStartIndex = posEndIndex$
    while $negHeadEndIndex >= 0$
     and $negHeadEndIndex \le negTailIndex$
     and $cl(negHeadEndIndex) \le 0$ do
        $negHeadEndIndex - -$
    end while
    $posCount = posEndIndex - posStartIndex$
    $negHeadCount = negHeadEndIndex$
     $-negHeadStartIndex$
    $j = posStartIndex$
    call Algorithm 3
    call Algorithm 4
end while
return $u\_b_{\mathrm{TPFL}}$
```

where $x_i,\ \alpha_j,\ \beta_j \in \mathbb{R}$. Note that the parameters x_i, α_j, and β_j were randomly set in the following range:

$$-10^9 \le x_i, \alpha_j, \beta_j \le 10^9. \tag{14}$$

The parameter s was set to 40, 490, 740, or 990, and r was set to 20. We then generated 100 test polynomial equations for each combination of parameters. All polynomials were preprocessed to have integer coefficients using the method introduced in [10].

The experiments were performed on an Intel Core i7 3770K CPU with 32 GB of RAM, with GCC 4.6.3 used as the C compiler. In addition, we used GMP [4], since the CF method needs multiple-precision arithmetic to compute the coefficients in the replaced polynomial equations.

Algorithm 3 Subroutine 1 of the "tail-pairing first-λ" bound.

```
if $negHeadCount > 0$ then
  $i = negHeadStartIndex$
  while $negHeadCount > 0$ do
    if $posCount == 1$
      and $negHeadCount > posCount$ then
      $k = negHeadCount - posCount + 1$
      for $v = 1$ to $k$ do
        $tempub = (-cl(i)/(cl(j)/k))^{1/(j-i)}$
        if $tempub > u\_b_{\mathrm{TPFL}}$ then
          $u\_b_{\mathrm{TPFL}} = tempub$
        end if
        $negHeadCount - -$
        $\lambda - -$
        $i - -$
        while $i \geq 0$ and $cl(i) == 0$ do
          $i - -$
        end while
      end for
    else
      $tempub = (-cl(i)/(cl(j))^{1/(j-i)}$
      if $tempub > u\_b_{\mathrm{TPFL}}$ then
        $u\_b_{\mathrm{TPFL}} = tempub$
      end if
      $negHeadCount - -$
      $\lambda - -$
      $i - -$
      while $i \geq 0$ and $cl(i) == 0$ do
        $i - -$
      end while
    end if
    $posCount - -$
    if $posCount > 0$ then
      $j - -$
      while $cl(j) == 0$ do
        $j - -$
      end while
    end if
  end while
end if
```

Algorithm 4 Subroutine 2 of the "tail-pairing first-λ" bound.

```
while $posCount > 0$
  and $negHeadEndIndex < negTailIndex$ do
  $i = negTailIndex$
  $tempub = (-cl(i)/(cl(j))^{1/(j-i)}$
  if $tempub > u\_b_{\mathrm{TPFL}}$ then
    $u\_b_{\mathrm{TPFL}} = tempub$
  end if
  $posCount - -$
  $\lambda - -$
  if $\lambda == 0$ then
    break
  end if
  $negTailIndex + +$
  while $negTailIndex >= 0$
   and $cl(negTailIndex) \geq 0$ do
    $negTailIndex + +$
  end while
  if $posCount > 0$ then
    $j - -$
    while $cl(j) == 0$ do
      $j - -$
    end while
  end if
end while
```

5.1.1 $\log_2$ optimization

$\log_2$ optimization is used in various open-source software [5] [6]. Assume that we wish to calculate bounds of the following form in multiple-precision integer:

$$\left(-\frac{b}{c}\right)^{\frac{1}{d-e}}, c > 0, b < 0, d > e > 0, \tag{15}$$

using division and root functions. It takes a considerable amount of time to calculate the bounds, and the execution time for each function depends on the bit-length of the arguments. Here, we can use $\log_2$ to find the bounds

$$\frac{1}{d-e}\left(\log_2(-b) - \log_2 c\right), \tag{16}$$

and the execution time of $\log_2$ for multiple-precision integer does not depend on the bit-length of the argument. Therefore, we can avoid division and root functions in multiple-precision integer by comparing $\log_2$ values. The bounds computed with $\log_2$ can be worse than those given by the division and root functions. However, this method saves a lot of time in computing the bounds, and is fast in terms of total execution time.

5.2 Results

Table 2 lists the execution time for special polynomial equations, and Table 3 lists that for random polynomial equations. Our proposed bounds are more effective than FL+LM and LMQ for the Laguerre polynomial and the Chebyshev polynomial, and are competitive with FL+LM for the Wilkinson and Mignotte polynomials. The maximum speed-up for the Laguerre polynomial is about 1.19, and for the Chebyshev-I and -II polynomials it is about 1.12 and 1.14 times, respectively. TPFL-II+LM2 is more effective than TPFL-I+LM2 for some special polynomial equations. We can see this tendency for random polynomial equations: both TPFL-I+LM2 and TPFL-II+LM2 are more effective than FL+LM and LMQ. We can also see that TPFL-II+LM2 is more effective than TPFL-I+LM2.

Table 2: Execution time for special polynomials.

Polynomial Class	Degree	Time (s) FL +LM	LMQ	TPFL-I +LM2	TPFL-II +LM2
Laguerre	100	0.01	0.01	0.01	0.01
Laguerre	1000	43.51	48.20	41.77	36.57
Laguerre	1500	221.10	242.69	217.21	189.34
Laguerre	2000	704.95	755.48	683.57	617.01
Chebyshev-I	100	0.01	0.01	0.01	0.01
Chebyshev-I	1000	40.22	41.11	36.30	36.48
Chebyshev-I	1500	206.87	210.86	184.45	185.61
Chebyshev-I	2000	650.85	638.67	590.36	590.36
Chebyshev-II	100	0.01	0.01	0.01	0.01
Chebyshev-II	1000	40.48	40.88	35.74	35.56
Chebyshev-II	1500	203.53	210.73	182.73	182.67
Chebyshev-II	2000	652.94	636.42	599.48	579.28
Wilkinson	100	0.00	0.00	0.00	0.00
Wilkinson	1000	4.53	4.92	4.52	4.54
Wilkinson	1500	22.45	23.82	22.46	22.46
Wilkinson	2000	70.46	73.97	70.59	70.60
Mignotte	100	0.00	0.00	0.00	0.00
Mignotte	1000	0.04	0.04	0.04	0.04
Mignotte	1500	0.12	0.12	0.12	0.12
Mignotte	2000	0.27	0.27	0.27	0.27

Table 3: Execution time for random polynomials.

Parameters	Degree	Time (s), Avg (Min/Max) FL+LM	LMQ
$s = 40$ $r = 20$	100	0.015(0.01/0.02)	0.0188(0.01/0.03)
$s = 490$ $r = 20$	1000	29.046(19.15/43.61)	30.161(17.47/49.39)
$s = 740$ $r = 20$	1500	135.59(94.78/203.07)	139.06(92.1/211.72)
$s = 990$ $r = 20$	2000	415.37(296.62/645.55)	425.47(270.36/835.35)

Parameters	Degree	Time (s), Avg (Min/Max) TPFL-I+LM2	TPFL-II+LM2
$s = 40$ $r = 20$	100	0.0145(0.01/0.02)	0.0127(0.01/0.02)
$s = 490$ $r = 20$	1000	27.325(19.05/38.39)	26.88(17.22/39.38)
$s = 740$ $r = 20$	1500	128.07(91.69/179.71)	123.84(86.17/176.16)
$s = 990$ $r = 20$	2000	384.11(266.41/617.17)	368.36(271.71/603.31)

6. Conclusions

In this study, we have proposed new lower bounds based on the local-max bound and the first-λ bound for accelerating the CF method. The local-max2 bound is sharper than or equal to the local-max bound. The tail-pairing first-λ bound is expected to be more suitable for the CF method than the first-λ bound, because of the need to replace $x \to x + l\_b$ many times in the CF method. The numerical results show that the average execution time of the CF method with both the local-max2 bound and the tail-pairing first-λ bound is faster than or nearly equal to that with the local-max bound, first-λ bound, and local-max quadratic bound for all polynomial equations.

References

[1] A. Akritas, A. Strzeboński, P. Vigklas, "Implementations of a new theorem for computing bounds for positive roots of polynomials", *C*omputing, 78, pp. 355–367, 2006.

[2] A. Akritas, A. Strzeboński, P. Vigklas, "Improving the performance of the continued fractions method using new bounds of positive roots", *N*onlinear Analysis: Modelling and Control, 13, pp. 265–279, 2008.

[3] G. Collins, A. Akritas, "Polynomial real root isolation using Descartes' rule of signs", *S*YMSAC '76, Proceedings of the Third ACM Symposium on Symbolic and Algebraic Computation, Yorktown Heights, NY, USA, ACM, pp. 272–275, 1976.

[4] (2013) The GNU MP Bignum Library. [Online]. Available: http://gmplib.org/

[5] (2013) Sage. [Online]. Available: http://sagemath.org/

[6] (2013) SymPy. [Online]. Available: http://sympy.org/en/index.html

[7] B. Kioustelidis, "Bounds for positive roots of polynomials", *J*. Comput. Appl. Math., 16(2), pp. 241–244, 1986.

[8] R.E. Moore, "Interval Analysis", Prentice Hall, Englewood Cliffs, N.J., 1966.

[9] N. Obreschkoff, "Verteilung und Berechnung der Nullstellen reeller Polynome", Berlin: VEB Deutscher Verlag der Wissenschaften 1963.

[10] Masami Takata, Takuto Akiyama, Sho Araki, Kinji Kimura, Yoshimasa Nakamura, "Improved Computation of Bounds for Positive Roots of Polynomials", In Proceedings of the 2013 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'13), Vol.II, pp. 168–174, 2013.

[11] A.J.H. Vincent, "Sur la resolution des équations numériques", J. Math. Pures Appl. 1, pp. 341–372, 1836. 2002.

Scalable 3D Modeling Using Multi-player Interactive Genetic Algorithm on Cloud Platform as a Service

Takahito Seyama[1], **Shintaro Bando**[1], **and Masaharu Munetomo**[2]
[1]Graduate School of Information, Hokkaido University, Sapporo, 060-0811, Japan
[2]Information Initiative Center, Hokkaido University, Sapporo, 060-0811, Japan

Abstract— *The simplification of 3D modeling in today's computer world remains challenging, and systems to support such modeling also need to be developed. It is further necessary in 3D modeling to incorporate subjective evaluation by users into optimization techniques. In this study, support for 3D modeling was investigated, and user fatigue (a disadvantage of interactive genetic algorithm usage) was reduced via a multi-player approach. A scalable system was also constructed to prevent overload-related delays from exacerbating user fatigue.*

1. Introduction

The rapid advancement of computers in recent years has enabled the creation of high-quality 3D computer graphics on PCs and game consoles. The proliferation of 3D printers and the availability of applications such as Google SketchUp and Metasequoia have also simplified and popularized 3D modeling. However, the simplification of such modeling in today's computer world remains challenging, and systems to support it also need to be developed. Optimization techniques related only to computer technology are not suited to 3D modeling, which is based on subjective user evaluation. Against such a background, this study involved the use of an interactive genetic algorithm for evolution calculation toward optimization based on user-computer interaction.

The interactive genetic algorithm enables adaptation to music, design, writing support and other areas where quantitative evaluation is difficult by allowing users themselves to perform evaluation, selection, termination and other conventional genetic algorithm operations. However, the algorithm has its own problems, such as user fatigue and small populations, as operations that are automated in conventional genetic algorithms must be performed manually. The burden on users increases with population size, but search efficiency decreases if the population size is reduced to ease this burden. In past studies, this burden was mitigated to a certain degree by changing the mutation ratio and using past genes to solve the above problems. However, as not all related issues had been resolved, it was necessary to find a solution from a different viewpoint based on these past studies. In the present study, an interactive genetic algorithm was implemented with a large number of users. Evaluation frequency and the burden on users were reduced by sharing the same gene pool and acquiring ideas from other users. Other factors to be considered are changes in people's attitudes and the formation of diverse cultures and values associated with the recent progress of globalization. The influence of such changes is evident in 3D modeling and design, which are based on human sensitivity. The application of a multi-player interactive genetic algorithm to 3D modeling alleviates the burden on the user by reducing the frequency with which evaluation must be performed and also supports the provision of diverse 3D models.

However, as the application of such a method may result in system overload, a scalable system was constructed in this study to prevent overload-related mental fatigue among users. Cloud computing technology, which has come to the fore as a new mode of operation to replace conventional server-client-type computing, was used to create the system. Open-source cloud computing software was used to provide extensibility and flexibility to the system in addition to other cloud benefits such as cost-effectiveness, availability and scalability. Scalability must also be taken into account for databases. As conventional SQL relational database management systems are not suited to a system involving distributed management using multiple computers or scale enlargement, this study involved the adoption of NoSQL, which can be used to process enormous amounts of data by having individual pieces of genetic hardware act as a single system. Flexible and enriched functionality was realized using MongoDB a document-oriented NoSQL data model.

To address the risk of possibly losing diversity of solutions and individual preferences due to the same gene pool being shared by many users, clustering based on the k-means method was performed to create new individuals and remove those with low evaluation values in each cluster.

A web application prototype satisfying these requirements was created and verified in the study. This paper gives an overview of the system constructed and the technology used for it.

2. Interactive genetic algorithm (IGA)

Interactive genetic algorithm usage is an optimization method based on user-computer interaction and users' subjective evaluation, and involves evolutionary calculation for genetic algorithms. Users evaluate the solution group subjectively and selection is based on this evaluation. Following such evaluation and selection, the genetic algorithm operations of crossing and mutation are performed to create a new solution group. In the majority of interactive genetic algorithms, the ending is also determined by users. The use of subjective evaluation allows the adaptation of such algorithms to art, music, design, construction, writing support and other areas where quantitative evaluation is difficult. As the adoption of user evaluation means a smaller population size and fewer generations than with conventional genetic algorithms, an algorithm for convergence at high speed is necessary. The mutation ratio is also high because diverse individuals are presented to users.

However, interactive genetic algorithms have their own problems, including user fatigue and a small population size due to the use of manual evaluation and selection. Such algorithms also involve a tradeoff relationship in which the burden on users increases if the population size is increased to improve search efficiency, and efficiency decreases if the population is made smaller to reduce the burden.

To alleviate the problem of user fatigue associated with interactive genetic algorithms, it is necessary to have solutions converge at high speed and reduce evaluation frequency. Although a variety of measures have been taken to address the issue (such as mutation ratio modification, use of past genes and user interface improvement) no fundamental solution has yet been found. In this study, user fatigue was reduced by allowing a large number of users to share the same gene pool and assimilate the ideas of others. This method also enables the presentation of diverse solutions to users.

However, diversity of solutions and individual preferences may be lost in multi-player interactive genetic algorithms if the convergence rate is inappropriate. It is also difficult to set an appropriate convergence rate in such algorithms featuring large numbers of uncertain elements. Accordingly, clustering via the k-means method (reference: “Some Methods for Classification and Analysis of Multivariate Observations, Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability”) was performed in this study to create new individuals based on crossing and mutation in each cluster and to update the population by removing individuals with low evaluation values in order to maintain the diversity of solutions and individual preferences.

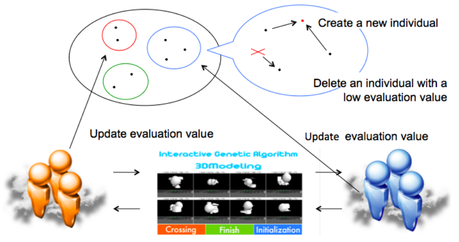


Fig. 1: Maintenance of the diversity of solutions and individual preferences via clustering

3. Cloud Foundry

3.1 Cloud Foundry overview

Cloud computing has advanced rapidly in recent years, and many companies and organizations benefit from the provision of services using related technology. However, technologies used in the field of cloud computing are many and varied, and individual companies and organizations provide services using different technologies. As a result, user convenience may be impaired by vendor lock-ins depending on specific cloud services and other adverse effects. Against this background, efforts to standardize cloud computing technology have become active, and open-source cloud computing software has advanced rapidly in recent years.

Cloud computing is divided into three types - Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) - depending on the type of service provided (reference: NIST Definition of Cloud Computing). Cloud Foundry is a type of open-source PaaS software developed and provided by VMware. PaaS provides databases and development environment platforms, and allows the construction of private PaaS environments. VMware also uses this to provide its cloudfoundry.com service.

Cloud Foundry also supports diverse languages, runtimes, frameworks and databases. It is easy to change application deployment, the number of instances and related performance, and databases using a command line tool called cf.

3.2 Cloud Foundry structure

Figure 2 shows the structure of Cloud Foundry (reference: Welcome to Cloud Foundry)
(reference: Welcome to Cloud Foundry)

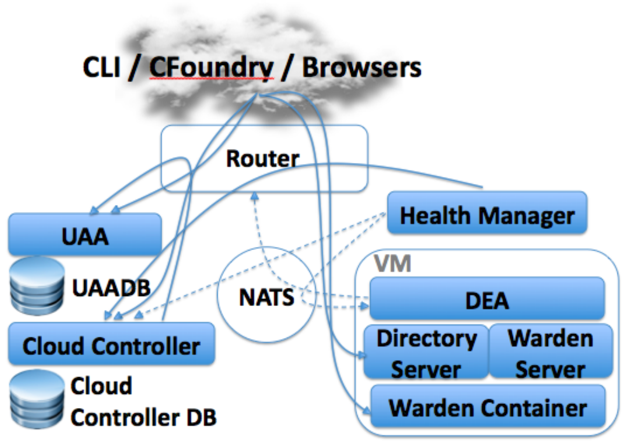


Fig. 2: Cloud Foundry Architecture

Cloud Controller

This provides the management function of Cloud Foundry. Application deployment, modification and other state changes for all components are performed via this application.

Router

This routes HTTP traffic from outside to appropriate components (usually applications running on Cloud Controller or a DEA node). It works in a way similar to that of the L7 load balancer in the OSI reference model.

DEA(Droplet Execution Agent)

This provides an agent function to receive commands from Cloud Controller and control the execution of applications.

Health Manager

This provides an application monitoring function on Cloud Foundry. It monitors the status of applications by sending Heartbeat data to the DEA and requesting repairs from Cloud Controller when an abnormality is detected.

Messaging (NATS)

This provides a messaging server function to mediate data exchange between components within Cloud Foundry, for which unique middleware called NATS is used.

Services

These provide deployment of databases and other services and bind to applications. Services can be integrated into Cloud Foundry via the Service Broker API.

Warden

This provides an application isolating function on the DEA and reduces influence between applications with a container system.

User Account and Authentication (UAA) Server

These provide the user authentication function for Cloud Foundry.

4. 3D modeling

Web 3D is a type of web data and a genre of 3D graphics based on a virtual 3D space that can be changed with the passage of time or operated freely by users.

Virtual reality modeling language (VRML) is a data format for Web3D and a medium that represents virtual 3D spaces. It was designed for use on the Web, and can be run online using a VRML browser or a regular browser with the corresponding plug-in. It comes in text file format with a .wrl extension. A VRML file consists of five elements - header, comments, nodes (fields), prototype and route.

Header

The first line of a VRML file is a header containing the VRML version number and character encoding, such as

```
#VRML V2.0 utf8
```

This indicates the use of VRML version 2.0 and UTF8 character encoding.

Comments

The second and subsequent lines starting with # are comment lines. Characters on these lines are for information only and have no function.

Nodes

A three-dimensional space created using VRML consists of various elements called nodes. Statements describing individual nodes are called node statements.
The basic form of a node statement is

```
Node type name {field ···. }.
```

Node type names are used to identify nodes, including those representing objective forms (e.g., spheres and cylinders), group nodes (e.g., transforms and groups) and appearance nodes representing the appearance of solid bodies as described below.

Fields

Each node has various attributes called attribute fields. Node names begin with an upper-case letter and field names begin with a lower-case letter to set them apart. Fields are identified by field names. The basic form of a field is

```
Field name Field value .
```

The field value is given as the value of the field designated by the field name.
Node statements can also be written as field values and nodes can have a nested structure.

Transform nodes

As with group nodes, transform nodes bring multiple nodes together and designate the transformation of the group. The center of the bounding box is designated by the bboxCenter field, the size of the bounding box by the bboxSize field, the node movement position by the translation field, node rotation by the rotation field and the node magnification factor by the scale field. The scaleOrientation field designates rotation against the node assigned in the children field before magnification/reduction in the scale field. The children field expresses multiple shape and group nodes as one node.

Transform node definition format

```
Transform {
    translation x y z
    rotation x y z r
    scale x y z
    scaleOrientation x y z r
    bboxCenter x y z
    bboxSize x y z
    children [child nodes  ···  ]
}
```

The rest of the node definition is omitted.

Shape nodes

Shape nodes define the shape and appearance of solid bodies. The shape (e.g., rectangular solid or sphere) can be designated in the geometry field, and the color, texture and other values are designated in the appearance field. The appearance node can be omitted.

Box nodes

Box nodes define the shape of rectangular solids. The n lengths in the x, y and z directions are designated in the size field.

Sphere nodes

Sphere nodes define the shape of spheres. The radius is designated in the radius field.

Cylinder nodes

Cylinder nodes define the shape of cylinders. The radius, height, bottom, side and top conditions are designated in the radius, height, bottom, side and top fields, respectively.

5. Structure and implementation of the proposed system

In this study, a web application prototype was created to realize 3D modeling using a multi-player interactive genetic algorithm. As extensive access (to systems? websites?) and prediction of the number of accesses have become difficult due to the proliferation of computers in recent years, flexible and scalable systems are necessary in large-scale web application development. With this in mind, a system featuring high flexibility, availability and scalability was constructed using cloud computing.

The same issue applies to databases. The relational database management systems that represent the mainstream in conventional database technology have abundant functions and outstanding consistency, but performance improvement based on scaling out is difficult. Accordingly, they are not suited to large-scale web applications that require flexibility and extensibility. This study involved the use of NoSQL, which enables performance improvement by scaling out based on distributed architecture. Enriched functionality was also realized using a document-oriented database (a NoSQL data model).

Figure 3 gives an overview of the entire system. Users access the application prototype web page via the Internet using PCs, smartphones, tablets and other digital devices to evaluate a 3D model. The evaluation data are sent as HTTP requests, and loads are distributed using a load balancer. The IGA program is run in the(each?) instance, and performance improvement can be easily achieved by scaling out. Data received and results are stored in the NoSQL database, which is a document-oriented data model with outstanding scalability and abundant functions.

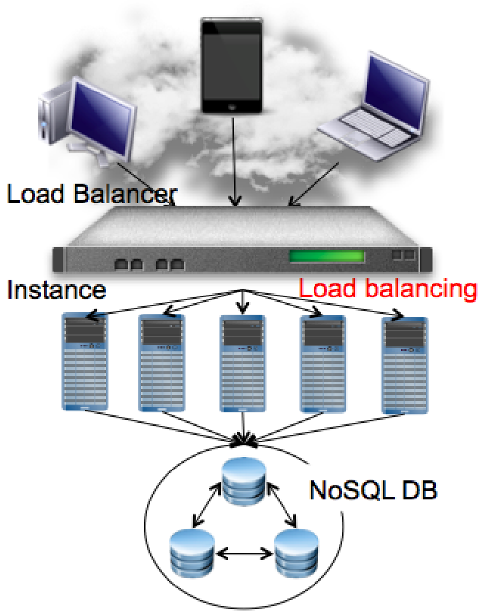
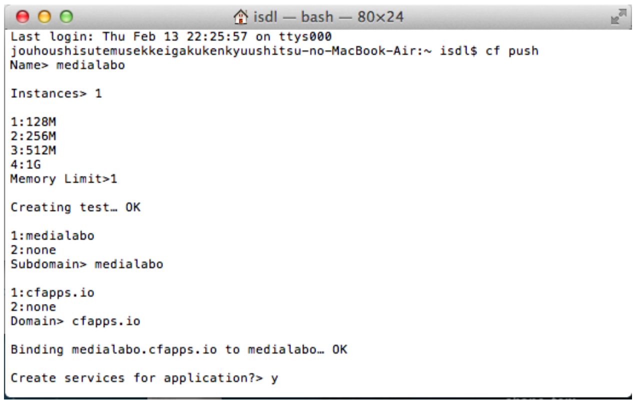


Fig. 3: Schematic diagram of the system

5.1 System implementation

In this study, the cloudfoundry.com service provided via VMWare on a trial basis was used (reference: Welcome to Cloud Foundry). Service users can set up an account free of charge, and PaaS implemented on Cloud Foundry can be accessed using a command line tool called cf. Although usable resources are limited, viability was confirmed using this service because it allows operation in the same way as when a Cloud Foundry environment is constructed in the operator's own environment. At cloudfoundry.com, operation can also be controlled via a graphical user interface (GUI) or a command line interface (CLI). Figure 4 shows application deployment via the CLI.



```
Last login: Thu Feb 13 22:25:57 on ttys000
jouhoushisutemusekkeigakukenkyuushitsu-no-MacBook-Air:~ isdl$ cf push
Name> medialabo

Instances> 1

1:128M
2:256M
3:512M
4:1G
Memory Limit>1

Creating test… OK

1:medialabo
2:none
Subdomain> medialabo

1:cfapps.io
2:none
Domain> cfapps.io

Binding medialabo.cfapps.io to medialabo… OK

Create services for application?> y
```

Fig. 4: Deployment status

1. Creation of initial population (population size: N)
2. Display of individuals presented (no. of individuals presented: M)
3. Evaluation by users/completion if the optimum solution exists
4. Update of evaluation values
5. Clustering using the k-means method based on genes
6. Selection of two individuals for crossing based on evaluation results
7. Creation of new individuals via crossing and mutation (evaluation value: average of the two selected individuals)
8. Removal of one individual with a low evaluation value
9. Repetition of 6 to 8 for each cluster
10. Selection of individuals for presentation based on evaluation results
11. Return to 2

Fig. 5: Program execution procedure

MongoDB, which is a document-oriented data model, was used as the NoSQL database. MongoDB is used to store and manage documents described by JSON as data models and to manage groups of documents as collections. The consistency of results is guaranteed and the model has a variety of applications, including MapReduce and an index function (reference: Basic Knowledge of NOSQL).

In the prototype of the application created in this study, the system was constructed with the following

No. of instances :3
Memory :512MB
Database :MongoDB
Database capacity :0.5GB

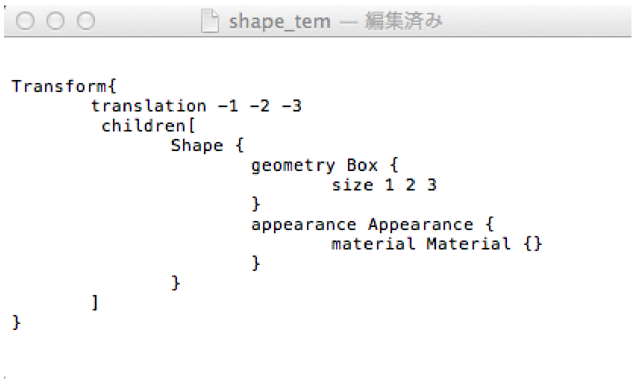
The application prototype was executed via the procedure shown in Fig. 5.

5.2 Implementation of 3D modeling

In this study, VRML object nodes were grouped and synthesized using a transform node. The object nodes used were box, sphere and cylinder types. As these nodes did not have fields defining coordinate movements, movement was made by grouping individual nodes using the transform node. The field value x representing the size was $1 \leq x \leq 4$, and y representing the central coordinate of the object was $-3 \leq y \leq 3$.

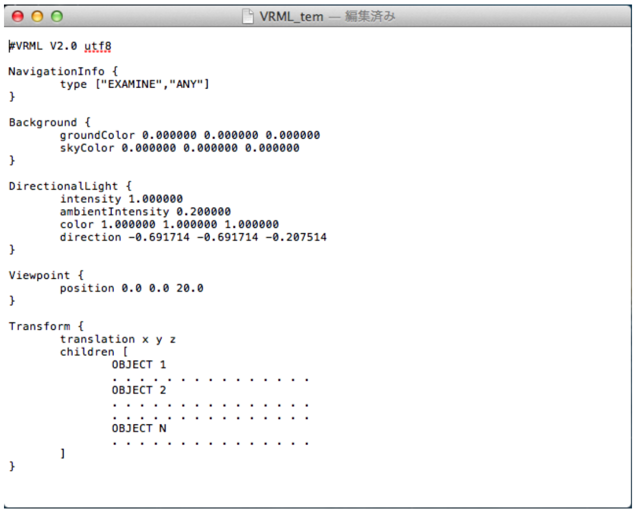
VRML files can be embedded in HTML using an embed tag.(Ex.: $<$ embed src = "VRML0.wrl" width = "300" height = "200" $>$)

Figure 6 shows an example of an object synthesized with a transform node, and Fig. 7 shows the VRML file template used. In Fig. 6 it can be seen that the center of a solid body with the size of 1, 2 and 3 in the x, y and z directions is in the coordinate (-1, -2, -3). In Fig. 7, N objects are synthesized and expressed as one by transforming genes in the “OBJECT” part as in the case shown in Fig. 6. In the translation field of the transform node, the center of gravity (a, b, c) of the synthesized object is calculated as x = -a, y = -b and z = -c to move the coordinate centering around the center of the object.



```
Transform{
        translation -1 -2 -3
         children[
                Shape {
                        geometry Box {
                                size 1 2 3
                        }
                        appearance Appearance {
                                material Material {}
                        }
                }
        ]
}
```

Fig. 6: Example of an object



```
#VRML V2.0 utf8

NavigationInfo {
        type ["EXAMINE","ANY"]
}

Background {
        groundColor 0.000000 0.000000 0.000000
        skyColor 0.000000 0.000000 0.000000
}

DirectionalLight {
        intensity 1.000000
        ambientIntensity 0.200000
        color 1.000000 1.000000 1.000000
        direction -0.691714 -0.691714 -0.207514
}

Viewpoint {
        position 0.0 0.0 20.0
}

Transform {
        translation x y z
        children [
                OBJECT 1
                . . . . . . . . . . . . . . .
                OBJECT 2
                . . . . . . . . . . . . . . .
                . . . . . . . . . . . . . . .
                OBJECT N
                . . . . . . . . . . . . . . .
        ]
}
```

Fig. 7: VRML file template

5.3 Interactive genetic algorithm implementation

A program was produced by setting the selection method, crossover method, mutation rate, population size, number of individuals presented and initial evaluation of the interactive genetic algorithm as follows:

Selection method :roulette wheel selection
Crossover method :two-point
Mutation ratio :3%
Population size :100
No. of individuals presented :8
Initial evaluation value :5
No. of clusters :10

In this study, genes had the shape, size and central coordinates of an object expressed in 7 bits. Genetic information details are given below.

Genetic information of the interactive genetic algorithm

1. Object shape (0: box; 1: sphere; 2: cylinder)
2. Object size a (box: center value x-coordinate; sphere: radius, cylinder: radius) ($1 \le a \le 4$)
3. Object size b (box: center value y-coordinate; cylinder: height) ($1 \le b \le 4$)
4. Object size c (box: center value z-coordinate) ($1 \le c \le 4$)
5. Object center value x-coordinate ($-3 \le x \le 3$)
6. Object center value y-coordinate ($-3 \le y \le 3$)
7. Object center value z-coordinate ($-3 \le z \le 3$)

5.4 Web application prototype

The application prototype is mainly expressed in JAVA. HTML page output and selected value transmission are achieved using JSP, and other processes are completed using a servlet.

3D modeling in the application prototype using an interactive genetic algorithm is performed mainly via the three steps outlined below. The Initialization button is used for individual and database initialization. In gene initialization, genes are created randomly within the range of the genetic information parameters shown in the overview of the interactive genetic algorithm implementation. The evaluation value is reset to 5. User evaluations of eight individuals presented are then sent to the IGA program using the Crossing button, and new individuals are presented to users after the evaluation values for individuals, clustering, crossing, mutation, removal of individuals and other processes have been updated. In the evaluation of individuals, Good, Normal and Bad are expressed by values of +1, ± 0 and -1, respectively, and the default is Normal. When a satisfactory individual is created, operation is ended by selecting the individual's Fin radio button and clicking the Finish button.

6. Operation confirmation

The web application prototype deployed on Cloud Foundry was actually operated in the study. The page shown in Fig. 8 was displayed by installing the browser plug-in and accessing the specified URL (http://medialabo.cfapps.io/). The Crossing button was clicked after evaluation to see newly created individuals.

The document in the database was stored as shown in Fig. 9. In this document, "name" is the individual number, "order" is the gene arrangement, "value" is the evaluation value and "cluster" is the relevant cluster number.

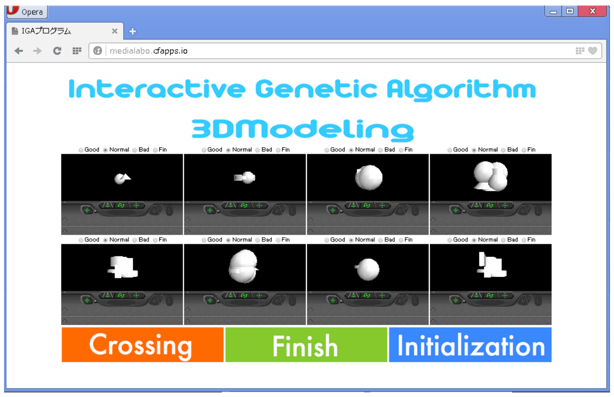
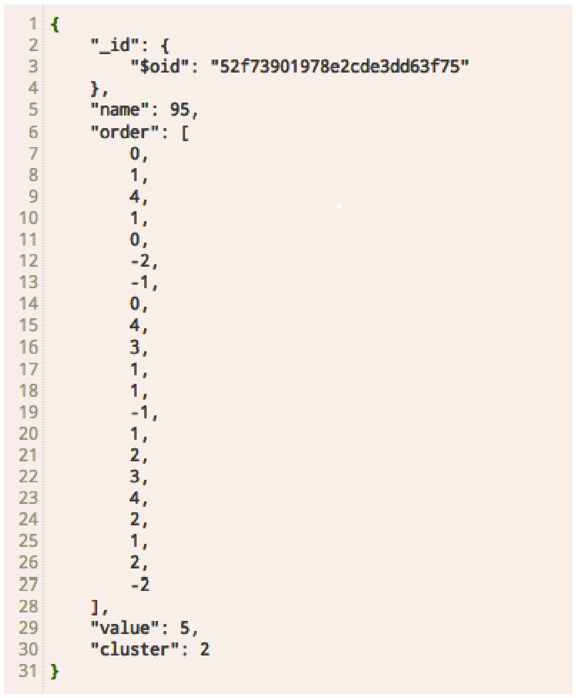


Fig. 8: Top page



```
{
    "_id": {
        "$oid": "52f73901978e2cde3dd63f75"
    },
    "name": 95,
    "order": [
        0,
        1,
        4,
        1,
        0,
        -2,
        -1,
        0,
        4,
        3,
        1,
        1,
        -1,
        1,
        2,
        3,
        4,
        2,
        1,
        2,
        -2
    ],
    "value": 5,
    "cluster": 2
}
```

Fig. 9: MongoDB document (data format)

7. Conclusion

In this study, a web application prototype in which 3D models are created on Cloud Foundry was created and verified. As a result of this verification, a scalable system was constructed and its operation was confirmed.

However, object production rules for matters such as formal grammar and production of initial individuals according to purpose still need to be established in order to realize a 3D modeling support system. The authors plan studies on more practical and new 3D modeling methods with focus on these issues.

In future work, it is first necessary to verify scalability in a large-scale system by constructing a Cloud Foundry environment and conducting loading tests on Hokkaido University's private cloud.

In regard to interactive genetic algorithms, comparison and verification need to be conducted by improving parameters, reusing optimum solutions found and extending application to interactive genetic programming (iGP) using tree-structured genes for interactive genetic algorithms.

To support coordination among large numbers of users, it is necessary to verify and compare methods to replace the k-means approach.

For 3D modeling, verification and comparison are needed concerning the production of objects via the establishment of formal grammar or other production rules and the transformation of initial individuals into models according to purpose.

Acknowledgments

This work is supported by KAKENHI (No.22500196), Japan Society for the Promotion of Science.

References

[1] Masayuki Hayashi, Introduction to the Open Cloud, Impress R&D (2012)
[2] Atsushi Nakada et al., Cloud Taizen (Complete Cloud Computing), Nikkei BP (2009)
[3] Makoto Shirota, Cloud no Shogeki (The Impact of the Cloud), Toyo Keizai, Inc. (2009)
[4] Hiroshi Ota, Shinya Motohashi, Tatsuya Kawano and Toshiaki Tsurumi, Basic Knowledge of NOSQL, Ric Telecom (2012)
[5] Japan CloudStack User Group et al., CloudStack Tettei Nyumon (Introduction to CloudStack), Shoeisha (2013)
[6] Shinji Miyamoto, Kiso kara no Servlet (Basics of Servlets)/JSP 3rd ed., Softbank Creative (2010)
[7] Nozomi Yamakawa, "Support System of Design with Interactive Genetic Algorithm", JSME 19th Computational Mechanics Division Conference, pp.193-194 (2006)
[8] Fuyuko Ito, Tomoyuki Hiroyasu, Mitsunori Miki and Hisatake Yokouchi, "Discussion of the Offspring Generation Method for Interactive Genetic Algorithms considering Multi-modal Preference", Journal of the Japanese Society for Artificial Intelligence, vol. 24, no. 1, pp. 127 - 135 (2009)
[9] Peter J.Bentley , David W.Corne. Creative Evolutionary Systems, Morgan Kaufmann (2001)
[10] Peter J.Bentley. Evolutionary Design by Computers, Morgan Kaufmann (1999)
[11] Rikk Carey, Gavin Bell, Chris Marrin. ISO/IEC 14772-1:1997 Virtual Reality Modeling Language (VRML97), VRML Consortium, Inc (1997)
[12] Peter Mell , Timothy Grance. The NIST Definition of Cloud Computing(NIST Special Publication 800-145)
[13] J. B. MacQueen. "Some Methods for classification and Analysis of Multivariate Observations, Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability", Berkeley, University of California Press, 1:281-297 (1967)
[14] Hideyuki Takagi. "Interactive Evolutionary Computation: Fusion of the Capabilities of EC Optimization and Human Evaluation", Proceedings of the IEEE, vol.89, no.9, pp.1275-1296 (2001)
[15] Alexandra Melike Brintrup , Jeremy Ramsden , Hideyuki Takagi. "Ergonomic Chair Design by Fusing Qualitative and Quantitative Criteria Using Interactive Genetic Algorithms", IEEE Transactions on Evolutionary Computation, vol. 12, no. 3 (2008)
[16] Hee-Su Kim and Sung-Bae Cho. "Application of interactive genetic algorithm to fashion design", Engineering Applications of Artificial Intelligence,vol. 13, no. 6, pp. 635 - 644 (2000)
[17] Jeff Clune , Anthony Chen , Hod Lipson. "Upload Any Object and Evolve it: Injecting Complex Geometric Patterns into CPPNs for Further Evolution", 2013 IEEE Congress on Evolutionary Computation (2013)
[18] http://www.cloudfoundry.com/ (2014/02/13)
[19] http://www.web3d.org/realtime-3d/(2014/02/16)

Selecting Strategies in Particle Swarm Optimization by Sampling-Based Landscape Modality Detection

Tetsuyuki Takahama[1] **and Setsuko Sakai**[2]
[1] Department of Intelligent Sciences, Hiroshima City University, Hiroshima, Japan
[2] Faculty of Commercial Sciences, Hiroshima Shudo University, Hiroshima, Japan

Abstract—*If the landscape of the objective function is unimodal, the efficiency of population-based optimization algorithms (POAs) can be improved by selecting strategies for local search around a best solution. If the landscape is multimodal, the robustness of the POAs can be improved by selecting strategies for global search in search space. We have proposed a method that estimates the landscape modality by sampling the objective values along a line and counting the number of changes in the objective values from increasing to decreasing and vice versa. In this study, in order to improve the performance of particle swarm optimization (PSO), we propose to select a proper strategy according to the landscape modality: The gbest model is selected in unimodal landscape and the lbest model is selected in multimodal landscape. The advantage of the proposed method is shown by solving unimodal and multimodal problems and by comparing it with standard PSOs.*

Keywords: Particle swarm optimization, Landscape modality, Landscape modality estimation, Lbest model, Gbest model

1. Introduction

There exist many studies on solving optimization problems using population-based optimization algorithms (POAs) in which a population or multiple search points are used to search for an optimal solution. For example, swarm intelligence algorithms inspired by collective animal behavior have been studied such as particle swarm optimization (PSO)[1], [2] and ant colony optimization. Also, evolutionary algorithms inspired by biological evolution have been studied such as genetic algorithm, evolution strategy and differential evolution[3], [4]. In general, POAs are stochastic direct search methods, which only need function values to be optimized, and are easy to implement. For this reason, POAs have been successfully applied to various optimization problems.

In this study, we paid attention to improve PSO. There are two models or movement strategies in PSO: the *gbest* model where each search point or a particle moves toward the best point in the population and the *lbest* model where each search point moves toward a best point in the neighbor points. It is known that the gbest model can solve unimodal problems efficiently but the strategy cannot solve multimodal problems stably and the search by the strategy is sometimes trapped at a local optimal solution. In contrast, it is known that the lbest strategy is robust to multimodal problems but the strategy cannot solve unimodal problems efficiently. However, the landscape of a problem to be optimized is often unknown and the landscape is changing dynamically while the search process proceeds. Thus, it is difficult to select a proper strategy.

We have proposed a simple method that detects the modality of landscape being searched: unimodal or not unimodal[5], [6], [7]. In the method, some points on the line connecting between the centroid of search points and the best search point are sampled. When the objective values of the sampled points are changed decreasingly and then increasingly, it is thought that one valley exists. If there exists only one valley or the landscape is unimodal, the gbest strategy is adopted. In this case, it is expected that the strategy can realize efficient search. If the number of valley is greater than one, the lbest strategy is adopted. In this case, it is expected that the strategy improves the divergence of the search and prevents premature convergence. The effect of the proposed method is shown by solving 13 benchmark problems including unimodal problems and multimodal problems.

In Section 2, related works are briefly reviewed. Detecting landscape modality is explained in Section 3. The optimization problem is defined and PSO is explained in Section 4. PSO with detecting landscape modality is proposed in Section 5. In Section 6, experimental results on some problems are shown. Finally, conclusions are described in Section 7.

2. Related Works

Many studies on strategy selection and parameter tuning have been done in order to improve the efficiency. The studies can be classified into two main categories: observation-based and success-based control[5], [6], [7].

1) observation-based control: The current search state is observed, proper strategies or parameter values are inferred according to the observation, and strategies and/or parameters are dynamically controlled. FADE(Fuzzy Adaptive DE)[8] observes the movement of search points and the change of function values between successive generations, and controls algorithm parameters. DESFC(DE with Speciation and Fuzzy

Clustering)[9] adopts fuzzy clustering, observes partition entropy of search points, and controls a parameter and the mutation strategies between the rand and the species-best strategy.

2) success-based control: It is recognized as a success case when a better search point than the parent is generated. The strategies and/or parameters are adjusted so that the values in the success cases are frequently used. It is thought that the self-adaptation, where strategies and/or parameters are contained in individuals and are evolved by applying evolutionary operators to the parameters, is included in this category. DESAP(Differential Evolution with Self-Adapting Populations)[10] controls algorithm parameters including population size self-adaptively. SaDE(Self-adaptive DE)[11] controls the selection probability of the mutation strategies according to the success rates and controls the mean value of a crossover rate for each strategy according to the mean value in success case. JADE(adaptive DE with optional external archive)[12] and MDE_pBX(modified DE with p-best crossover)[13] control the mean and power mean values of two parameters according to the mean values in success cases.

In the category 1), it is difficult to select proper type of observation which is independent of the optimization problem and its scale. In the category 2), when a new good search point is found near the parent, parameters are adjusted to the direction of convergence. In problems with ridge landscape or multimodal landscape, where good search points exist in small region, parameters are tuned for small success and big success will be missed. Thus, search process would be trapped at a local optimal solution.

In this study, we propose a new observation-based control in the category 1). As a problem independent observation, landscape modality is adopted and it is estimated whether the problem is unimodal or multimodal using sampling. It is thought that a proper strategy or algorithm parameters can be selected if the landscape modality can be identified.

3. Detecting Landscape Modality using Sampling

Search points in a current population or a set of search points $P = \{\boldsymbol{x}_i | i = 1, 2, \cdots, N\}$ are used to detect landscape modality using sampling[5], [6], where N is the number of search points or population size. The range of search points is determined, a line is drawn in the range, and equally spaced points are sampled along the line.

3.1 Sampling

The objective values are examined along the following line, which connects the centroid of search points $\boldsymbol{x}^g$ and the best search point $\boldsymbol{x}^b$.

$$\boldsymbol{x} = \boldsymbol{x}^g + \lambda(\boldsymbol{x}^b - \boldsymbol{x}^g) \quad (1)$$

$$\boldsymbol{x}^g = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_i \quad (2)$$

$$\boldsymbol{x}^b = \arg\min_{\boldsymbol{x}_i \in P} f(\boldsymbol{x}_i) \quad (3)$$

where λ is a parameter for deciding the position of a point on the line. The range of the search points $[\boldsymbol{x}^{\min}, \boldsymbol{x}^{\max}]$ can be given as follows:

$$x_j^{\min} = \min_i x_{ij} \quad (4)$$

$$x_j^{\max} = \max_i x_{ij} \quad (5)$$

The range of the λ, $[\lambda^{\min}, \lambda^{\max}]$ satisfies the following condition:

$$x_j^{\min} \le x_j^g + \lambda(x_j^b - x_j^g) \le x_j^{\max} \quad (6)$$

Thus, if $(x_j^b - x_j^g)$ is positive, the range of the λ is given by:

$$\lambda^{\min} = \max_j \frac{x_j^{\min} - x_j^g}{x_j^b - x_j^g} \quad (7)$$

$$\lambda^{\max} = \min_j \frac{x_j^{\max} - x_j^g}{x_j^b - x_j^g} \quad (8)$$

If $(x_j^b - x_j^g)$ is negative, $x_j^{\min}$ and $x_j^{\max}$ in the equations are exchanged.

In order to decide M sampling points $\{\boldsymbol{x}_k | k = 1, 2, \cdots, M\}$, λ_k is given as follows:

$$\lambda_k = \lambda^{\min} + \frac{\lambda^{\max} - \lambda^{\min}}{M-1}(k-1) \quad (9)$$

$$\boldsymbol{z}_k = \boldsymbol{x}^g + \lambda_k(\boldsymbol{x}^b - \boldsymbol{x}^g) \quad (10)$$

Figure 1 shows an example of the sampling, where search points are shown by black circles, the centroid is shown by a white circle, sampling points are shown by triangles in case of $M = 6$.

3.2 Landscape Modality

In the obtained sequence $\{f(\boldsymbol{z}_k) | k = 1, 2, \cdots, M\}$, hill-valley relation is examined. For each point, the function $dir(\cdot)$ is introduced in order to judge whether the change is increasing or decreasing:

$$dir(\boldsymbol{z}_k) = \begin{cases} 1 & (f(\boldsymbol{z}_{k+1}) > f(\boldsymbol{z}_k)) \\ -1 & (f(\boldsymbol{z}_{k+1}) < f(\boldsymbol{z}_k)) \\ dir(\boldsymbol{z}_{k-1}) & (\text{otherwise}) \end{cases} \quad (11)$$

Figure 2 shows an example of detecting unimodal landscape, where the objective values are shown by the function of λ.

The landscape modality is identified using the number of changes in dir function. If the value of dir changed from -1 to 1 only once or there is no changes, it is thought that one valley exists and the landscape is unimodal. Otherwise, the landscape is not unimodal.

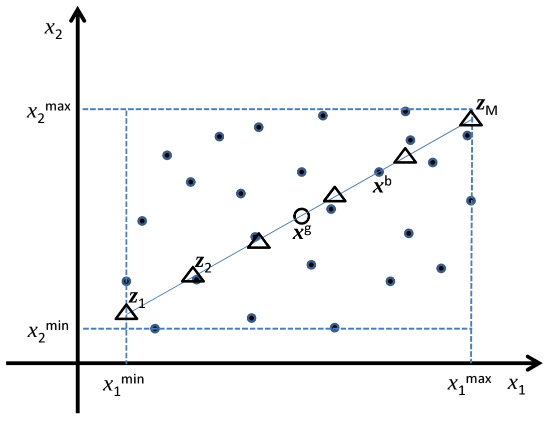


Fig. 1: An example of sampling for detecting landscape modality.

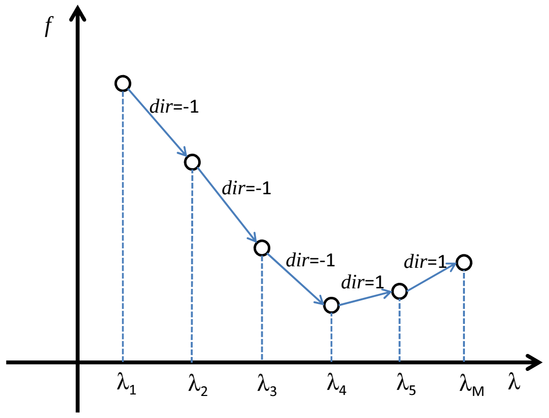


Fig. 2: An example of detecting unimodal landscape.

4. Optimization Problems and Particle Swarm Optimization

4.1 Optimization Problems

In this study, the following optimization problem (P) with lower bound and upper bound constraints will be discussed.

$$\text{(P)}\quad \begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) \\ \text{subject to} & l_i \le x_i \le u_i,\ i = 1, \dots, n, \end{array} \tag{12}$$

where $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$ is an n dimensional vector and $f(\boldsymbol{x})$ is an objective function. The function f is a nonlinear real-valued function. Values l_i and u_i are the lower bound and the upper bound of x_i, respectively. Let the search space in which every point satisfies the lower and upper bound constraints be denoted by $\mathcal{S}$.

4.2 Particle Swarm Optimization

An animal such as an ant, a fish, and a bird has limited memory and ability to perform simple actions. In contrast, a group of animals such as an ant swarm, a fish school, and a bird flock can take complex or intelligent actions such as avoiding predators and seeking foods efficiently. Swarm intelligence is defined as the collective actions of agents that act autonomously and communicate each other. PSO[2] is a swarm intelligence based optimization method which was inspired by the movement of a bird flock. PSO imitates the movement to solve optimization problems and is considered as a population-based stochastic search method or POA.

Searching procedures by PSO can be described as follows: A group of agents minimizes the objective function f. At any time t, each agent i knows its current position $\boldsymbol{x}_i^t$ and velocity $\boldsymbol{v}_i^t$. It also remembers its personal best visited position until now $\boldsymbol{x}_i^*$ and the objective value $pbest_i$.

$$\boldsymbol{x}_i^* = \arg \min_{\tau=0,1,\cdots,t} f(\boldsymbol{x}_i^\tau) \tag{13}$$

$$pbest_i = f(\boldsymbol{x}_i^*) \tag{14}$$

Two models, gbest model and lbest model have been proposed. In the gbest model, every agent knows the best visited position $\boldsymbol{x}_G^*$ in all agents and its objective value $gbest$.

$$\boldsymbol{x}_G^* = \arg\min_i f(\boldsymbol{x}_i^*) \tag{15}$$

$$gbest = f(\boldsymbol{x}_G^*) \tag{16}$$

In the lbest model, each agent knows the best visited position $\boldsymbol{x}_l^*$ in the neighbors and its objective value $lbest_i$, where l is the best visited position in the neighborhood.

$$\boldsymbol{x}_l^* = \arg \min_{k\in N_i} f(\boldsymbol{x}_k^*) \tag{17}$$

$$lbest_i = f(\boldsymbol{x}_l^*) \tag{18}$$

where N_i is the set of neighbor agents to i. The velocity of the agent i at time $t+1$ is defined as follows:

$$\begin{aligned} v_{ij}^{t+1} = wv_{ij}^t &+ c_1\, rand_{1ij}\, (x_{ij}^* - x_{ij}^t) \\ &+ c_2\, rand_{2ij}\, (x_{lj}^* - x_{ij}^t) \end{aligned} \tag{19}$$

where $l = G$ in the gbest model, w is an inertia weight and $rand_{kij}$ is a uniform random number in $[0,1]$ which is generated in each dimension. c_1 is a cognitive parameter, c_2 is a social parameter which represent the weight of the movement to the personal best and the group/neighbors best respectively.

The position of the agent i at time $t+1$ is given as follows:

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \boldsymbol{v}_i^{t+1} \tag{20}$$

4.3 Algorithm of PSO

The algorithm of PSO is defined as follows:

1) Initializing agents: Each agent i with a position $\boldsymbol{x}_i$ and a velocity $\boldsymbol{v}_i$ is created. $\boldsymbol{x}_i$ is randomly generated in the search space S, namely each element x_{ij} is a uniform random number in $[l_j, u_j]$. $\boldsymbol{v}_i$ is the zero vector where every element v_{ij}=0 in this study. The best visited position is set to the initial position, namely $\boldsymbol{x}_i^*$=$\boldsymbol{x}_i$.

2) Selecting the best agent: The id of the best agent G is decided.
3) Stopping if termination condition is satisfied: The algorithm is stopped when the number of function evaluations reaches the maximum number of evaluations FE_{max}.
4) Updating agents: The position and velocity of each agent i are updated according to Eq.(19) and Eq.(20), respectively. The each element of the velocity is truncated in $[-V_{max_j}, V_{max_j}]$. If the objective value of the new position is better than the personal best value, the personal best visited position is replaced with the new position. If the objective value of the new position is better than the group best value, the group's best visited position is replaced with the new position.
5) Go back to 3.

5. Proposed Method

In this section, a method of selecting a movement strategy dynamically is proposed for PSO.

5.1 Strategy selection

In general, if divergence of agents is kept to realize a global search, it can be avoided to be trapped at a local solution but the efficiency of the search will be reduced. If convergence of agents is enforced to realize local search around the best agent, the efficiency of the search is improved but the search will be trapped at a local solution.

In PSO, the gbest model can realize the local search and the lbest model can realize the global search. In the lbest model, the neighborhood of agents is defined as a topology such as star topology, ring topology, mesh topology, and so on. In this study, the ring topology is adopted, where agents are connected in the order of the agent numbers. The neighborhood size $N_{neighbor}$ is an important parameter in the lbest model. Small neighborhood size strengthens the global search and large neighborhood size strengthens the local search. When the size is same as the population size, the lbest model becomes the gbest model. In this study, the gbest model is selected for unimodal landscape and the lbest model with $N_{neighbor} = 5$ including the agent itself is selected for multimodal landscape.

5.2 Proposed algorithm

Figure 3 shows the proposed algorithm named LPSO(PSO with detecting Landscape modality), where T_L is the interval of iterations when landscape modality is estimated, N_{small} is the neighborhood size for the global search, and N_{large} is the neighborhood size for the local search. Lines with '+' at the first column are the modification to standard PSO.

If the number of direction changes from decreasing to increasing and vice versa is 1 or zero, the landscape modality is estimated as unimodal. However, the estimation should be done carefully because the sampling is done in a small region and the number of sampling points is small. Thus, the number of successive unimodal estimations is counted and if the number is equal to or greater than $N_{unimodal}$ the landscape is identified as unimodal.

```
 Initialize $P$;
 Evaluate all $\boldsymbol{x}$ in $P$;
 $G$=$\arg\min_{\{i|\boldsymbol{x}_i\in P\}} f(\boldsymbol{x}_i)$
+unimodal=0;
 for($t$=1;$t \leq T$;$t$++) {
+  if($t$%$T_L$==1) {
+    changed=landscape modality estimation in $P$;
+    if(changed==0 || changed==1) unimodal++;
+    else unimodal=0;
+  }
+  if(unimodal$\geq N_{unimodal}$) $N_{neighbor}$=$N_{large}$;
+  else $N_{neighbor}$=$N_{small}$;
   for(each agent $i$ in $P$) {
     $l$=best agent in $i$'s neighborhood
            of size $N_{neighbor}$;
     for(each dimension $j$) {
       $v_{ij}$=$wv_{ij}$+$c_1 rand_{1ij}$($x^*_{ij}$-$x_{ij}$)
                +$c_2 rand_{2ij}$($x^*_{lj}$-$x_{ij}$);
       if($v_{ij}$>$V_{\max_j}$) $v_{ij}$=$V_{\max_j}$;
       else if($v_{ij}$<$-V_{\max_j}$) $v_{ij}$=$-V_{\max_j}$;
       $x_{ij}$=$x_{ij}$+$v_{ij}$;
     }
     Evaluate $\boldsymbol{x}_i$;
     if($f(\boldsymbol{x}_i) < f(\boldsymbol{x}^*_i)$) {
       if($f(\boldsymbol{x}_i) < f(\boldsymbol{x}^*_G)$) $G$=$i$;
       $\boldsymbol{x}^*_i$=$\boldsymbol{x}_i$;
     }
   }
 }
 returns $\boldsymbol{x}^*_G$ as the best solution;
```

Fig. 3: Algorithm of LPSO.

6. Solving Optimization Problems

In this study, well-known thirteen benchmark problems are solved.

6.1 Test Problems and Experimental Conditions

The 13 scalable benchmark functions are shown in Table 1[12]. All functions have an optimal value 0. Some characteristics are briefly summarized as follows: Functions f_1 to f_4 are continuous unimodal functions. The function f_5 is Rosenbrock function which is unimodal for 2- and 3-dimensions but may have multiple minima in high dimension cases[14]. The function f_6 is a discontinuous step function, and f_7 is a noisy quartic function. Functions f_8 to f_{13} are multimodal functions and the number of their local minima increases exponentially with the problem dimension[15].

Independent 50 runs are performed for 13 problems. The dimension of problems is 30 (D=30). The maximum number of evaluations $FE_{\max}$ is 200,000. The parameters of PSO

Table 1: Test functions of dimension D. These are sphere, Schwefel 2.22, Schwefel 1.2, Schwefel 2.21, Rosenbrock, step, noisy quartic, Schwefel 2.26, Rastrigin, Ackley, Griewank, and two penalized functions, respectively[16].

Test functions	Domain
$f_1(\boldsymbol{x}) = \sum_{i=1}^{D} x_i^2$	$[-100, 100]^D$
$f_2(\boldsymbol{x}) = \sum_{i=1}^{D} \lvert x_i\rvert + \prod_{i=1}^{D} \lvert x_i\rvert$	$[-10, 10]^D$
$f_3(\boldsymbol{x}) = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_j\right)^2$	$[-100, 100]^D$
$f_4(\boldsymbol{x}) = \max_i\{\lvert x_i\rvert\}$	$[-100, 100]^D$
$f_5(\boldsymbol{x}) = \sum_{i=1}^{D-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$	$[-30, 30]^D$
$f_6(\boldsymbol{x}) = \sum_{i=1}^{D} \lfloor x_i + 0.5\rfloor^2$	$[-100, 100]^D$
$f_7(\boldsymbol{x}) = \sum_{i=1}^{D} ix_i^4 + rand[0, 1)$	$[-1.28, 1.28]^D$
$f_8(\boldsymbol{x}) = \sum_{i=1}^{D} -x_i \sin\sqrt{\lvert x_i\rvert} + D \cdot 418.98288727243369$	$[-500, 500]^D$
$f_9(\boldsymbol{x}) = \sum_{i=1}^{D} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$	$[-5.12, 5.12]^D$
$f_{10}(\boldsymbol{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)\right) + 20 + e$	$[-32, 32]^D$
$f_{11}(x) = \frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$[-600, 600]^D$
$f_{12}(x) = \frac{\pi}{D}[10\sin^2(\pi y_1) + \sum_{i=1}^{D-1}(y_i - 1)^2 \{1 + 10\sin^2(\pi y_{i+1})\} + (y_D - 1)^2] + \sum_{i=1}^{D} u(x_i, 10, 100, 4)$ where $y_i = 1 + \frac{1}{4}(x_i + 1)$ and $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m & x_i > a \\ 0 & -a \le x_i \le a \\ k(-x_i - a)^m & x_i < -a \end{cases}$	$[-50, 50]^D$
$f_{13}(x) = 0.1[\sin^2(3\pi x_1) + \sum_{i=1}^{D-1}(x_i - 1)^2 \{1 + \sin^2(3\pi x_{i+1})\} + (x_D - 1)^2 \{1 + \sin^2(2\pi x_D)\}] + \sum_{i=1}^{D} u(x_i, 5, 100, 4)$	$[-50, 50]^D$

are selected according to [17]: Number of agents $N = 30$, $w = 0.729$, $c_1 = c_2 = 0.729 \times 2.05 = 1.49455$ and $V_{\max_j} = 0.5(u_j - l_j)$. The parameters of LPSO are: The number of sampling points $M = N$, $T_L = 200$, $N_{unimodal} = 5$, $N_{small} = 5$ for the lbest model and $N_{large} = N$ for the gbest model.

6.2 Experimental Results

The performance of three algorithms, gbest model PSO, lbest model PSO, and LPSO are compared. Table 2 shows the experimental results. The mean value and standard deviation of best objective values over 50 runs are shown in the top row for each function. The number of success runs, where the algorithm can find the near optimal value less than 10^{-7}, is shown in the bottom row. The best results among all algorithms are highlighted using bold face fonts.

The gbest model PSO attained the best results in unimodal functions f_1, f_2, f_3 and f_4. Also, the gbest model attained the best results in multimodal functions f_5, f_8 and f_9. It is thought that the gbest model is suitable not only to unimodal functions but also to functions where search points need to move a fairly long distance such as f_5 and f_8. The lbest model PSO attained the best results in multimodal functions f_{12} and f_{13}, and in the step function f_6.

It is thought that LPSO will show the intermediate performance between the gbest model and the lbest model. Nevertheless, LPSO attained the best results in multimodal functions f_{10} and f_{11}, the step function f_6 and the noisy function f_7. Thus, it is shown that dynamic selection of the gbest model and the lbest model can attain better result than pure gbest or lbest model.

LPSO got the first and second rank among three algorithms and did not get the worst rank in all functions. The average ranks of the gbest model, the lbest model and LPSO are 1.85, 2.42 and 1.73, respectively. LPSO attained the best performance as for the average rank.

The average success runs over 13 functions in the gbest model, the lbest model and LPSO are 21.00, 20.46 and 30.62, respectively. LPSO attained the best performance as for the average success runs.

Therefore, it is thought that LPSO showed the most stable performance.

As a reference, convergence graphs of test functions are shown in Figure 4, where the mean best objective values of LPSO, the gbest model PSO and the lbest model PSO are plotted over the number of function evaluations.

7. Conclusions

It is difficult to select a proper optimization strategy, because the proper strategy depends on the optimization problem and also on landscape currently being searched. In this study, in order to select a proper strategy of PSO dynamically, a dynamic selection of strategies is proposed where the gbest model is selected in unimodal landscape and the lbest model is selected in multimodal landscape. Various 13 functions are solved and the results are compared with those of the gbest and lbest models of PSO. It was shown that the proposed method sometimes outperformed the pure models and attained the most stable performance.

In the future, we will apply the dynamic selection of strategies to various algorithms. Also, we will apply the dynamic selection of algorithms such as an algorithm in unimodal landscape and another algorithm in multimodal landscape.

Acknowledgment

This research is supported in part by Grant-in-Aid for Scientific Research (C) (No. 22510166,24500177) of Japan society for the promotion of science and Hiroshima City University Grant for Special Academic Research (General Studies).

Table 2: Experimental results on standard PSOs and the proposed method. Mean value ± standard deviation and the number of success runs in 50 runs are shown.

	gbest model PSO	lbest model PSO	LPSO
f_1	**7.650e-118 ± 2.779e-117** [50]	3.392e-46 ± 7.533e-46 [50]	3.562e-109 ± 2.449e-108 [50]
f_2	**1.306e-39 ± 9.139e-39** [50]	4.722e-29 ± 3.509e-29 [0]	1.378e-38 ± 7.001e-38 [50]
f_3	**1.451e-13 ± 2.707e-13** [50]	1.912e+03 ± 9.675e+02 [0]	7.385e-13 ± 1.811e-12 [50]
f_4	**1.058e-06 ± 2.506e-06** [11]	1.496e-01 ± 8.532e-02 [0]	1.476e-06 ± 2.992e-06 [11]
f_5	**1.139e+01 ± 1.731e+01** [0]	7.128e+01 ± 4.073e+01 [0]	3.450e+01 ± 3.424e+01 [0]
f_6	2.900e+00 ± 6.275e+00 [19]	**0.000e+00 ± 0.000e+00** [50]	**0.000e+00 ± 0.000e+00** [50]
f_7	5.543e-03 ± 2.994e-03 [0]	1.047e-02 ± 3.555e-03 [0]	**4.201e-03 ± 1.557e-03** [0]
f_8	**3.043e+03 ± 6.714e+02** [0]	4.394e+03 ± 5.930e+02 [0]	4.313e+03 ± 5.978e+02 [0]
f_9	**7.245e+01 ± 1.612e+01** [0]	1.030e+02 ± 1.701e+01 [0]	7.466e+01 ± 1.933e+01 [0]
f_{10}	1.626e+00 ± 1.053e+00 [10]	1.581e-14 ± 4.884e-15 [50]	**1.105e-14 ± 5.012e-15** [50]
f_{11}	2.472e-02 ± 3.357e-02 [19]	3.149e-03 ± 9.259e-03 [16]	**6.191e-04 ± 1.844e-03** [39]
f_{12}	1.826e-01 ± 3.576e-01 [28]	**1.135e-21 ± 7.942e-21** [50]	4.147e-03 ± 2.031e-02 [48]
f_{13}	8.743e-02 ± 3.750e-01 [36]	**1.350e-32 ± 0.000e+00** [50]	1.352e-32 ± 1.726e-34 [50]

References

[1] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proc. of IEEE International Conference on Neural Networks*, vol. IV, Perth, Australia, 1995, pp. 1942–1948.

[2] J. Kennedy and R. C. Eberhart, *Swarm Intelligence*. San Francisco: Morgan Kaufmann, 2001.

[3] R. Storn and K. Price, "Minimizing the real functions of the ICEC'96 contest by differential evolution," in *Proc. of the International Conference on Evolutionary Computation*, 1996, pp. 842–844.

[4] R. Storn and K. Price, "Differential evolution – A simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, pp. 341–359, 1997.

[5] T. Takahama and S. Sakai, "Differential evolution with dynamic strategy and parameter selection by detecting landscape modality," in *Proc. of the 2012 IEEE Congress on Evolutionary Computation*, June 2012, pp. 2114–2121.

[6] T. Takahama and S. Sakai, "Large scale optimization by differential evolution with landscape modality detection and a diversity archive," in *Proc. of the 2012 IEEE Congress on Evolutionary Computation*, June 2012, pp. 2842–2849.

[7] S. Sakai and T. Takahama, "Large scale optimization by adaptive differential evolution with landscape modality detection and a diversity archive," *Journal of Business Studies*, vol. 58, no. 3, pp. 55–77, Mar. 2012.

[8] J. Liu and J. Lampinen, "A fuzzy adaptive differential evolution algorithm," *Soft Comput.*, vol. 9, no. 6, pp. 448–462, 2005.

[9] T. Takahama and S. Sakai, "Fuzzy c-means clustering and partition entropy for species-best strategy and search mode selection in nonlinear optimization by differential evolution," in *Proc. of the 2011 IEEE International Conference on Fuzzy Systems*, June 2011, pp. 290–297.

[10] J. Teo, "Exploring dynamic self-adaptive populations in differential evolution," *Soft Comput.*, vol. 10, no. 8, pp. 673–686, 2006.

[11] A. Qin, V. Huang, and P. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 398–417, april 2009.

[12] J. Zhang and A. C. Sanderson, "JADE: Adaptive differential evolution with optional external archive," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 5, pp. 945–958, Oct. 2009.

[13] S. M. Islam, S. Das, S. Ghosh, S. Roy, and P. N. Suganthan, "An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 42, no. 2, pp. 482–500, april 2012.

[14] Y.-W. Shang and Y.-H. Qiu, "A note on the extended Rosenbrock function," *Evolutionary Computation*, vol. 14, no. 1, pp. 119–126, 2006.

[15] X. Yao, Y. Liu, , and G. Lin, "Evolutionary programming made faster," *IEEE Transactions on Evolutionary Computation*, vol. 3, pp. 82–102, 1999.

[16] X. Yao, Y. Liu, K.-H. Liang, and G. Lin, "Fast evolutionary algorithms," in *Advances in Evolutionary Computing: Theory and Applications*, A. Ghosh and S. Tsutsui, Eds. New York, NY, USA: Springer-Verlag New York, Inc., 2003, pp. 45–94.

[17] R. Eberhart and Y. Shi, "Particle swarm optimization: developments, applications and resources," in *Proceedings of the 2001 Congress on Evolutionary Computation*, vol. 1, 2001, pp. 81–86.

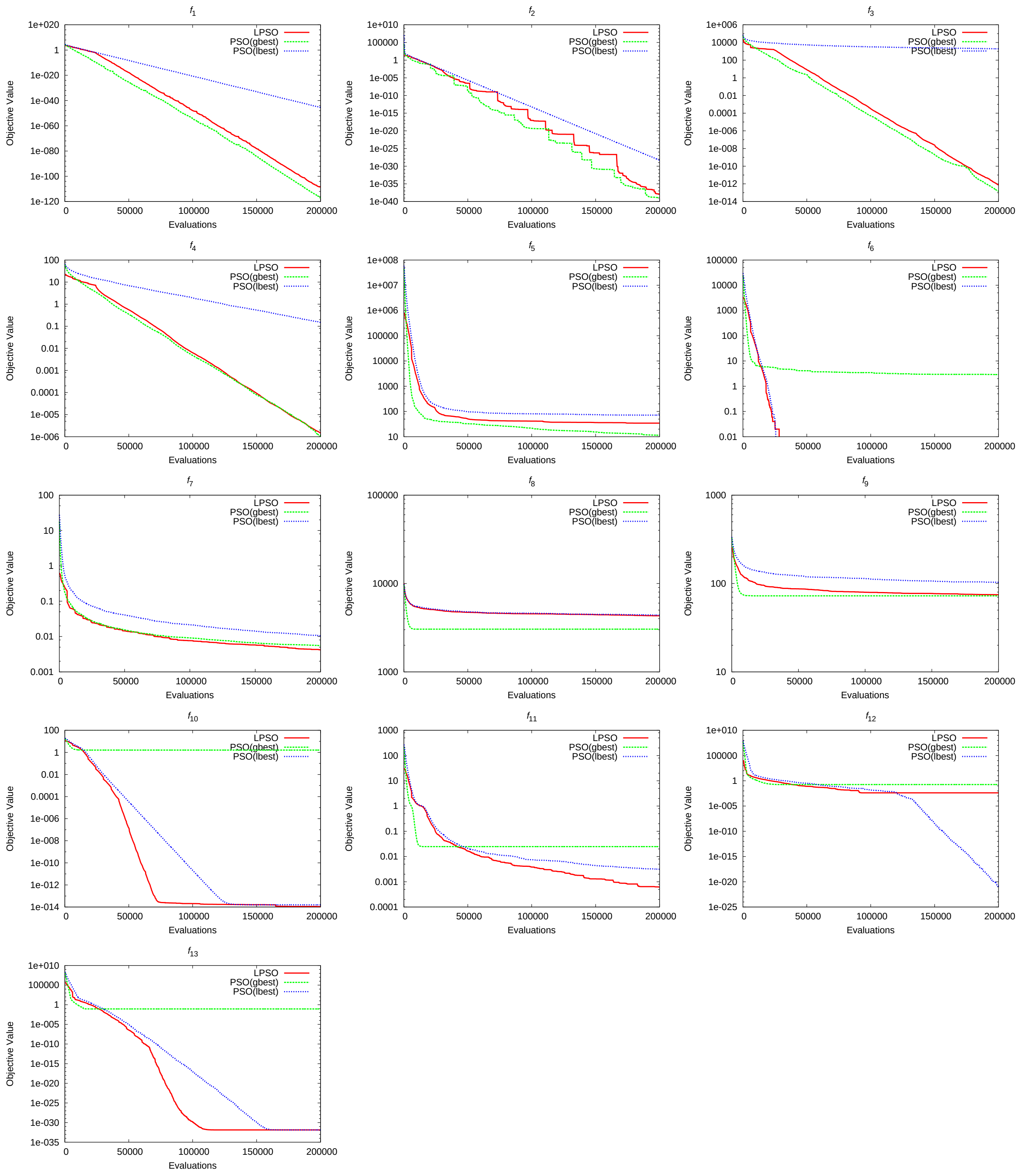


Fig. 4: Convergence graphs.

A Study on Non-Correspondence in Spread between Objective Space and Design Variable Space in Pareto Solutions

Tomohiro Yoshikawa, Toru Yoshida
Dept. of Computational Science and Engineering
Nagoya University
Furo-cho, Chikusa-ku, Nagoya 466-8603, Japan

Abstract— *Recently, a lot of studies on Multi-Objective Genetic Algorithm (MOGA), in which Genetic Algorithm is applied to Multi-objective Optimization Problems (MOPs), have been reported actively. MOGA has been also applied to engineering design fields, then it is important not only to obtain Pareto solutions having high performance but also to analyze the obtained Pareto solutions and extract the knowledge in the designing problem. In order to analyze Pareto solutions obtained by MOGA, it is required to consider both the objective space and the design variable space. In this paper, we define "Non-Correspondence in Spread" between the objective space and the design variable space. We also try to extract Non-Correspondence area in Spread with the index defined in this paper. This paper applies the proposed method to the trajectory designing optimization problem and extracts Non-Correspondence area in Spread in the acquired Pareto solutions.*

Keywords: Non-Correspondence, Objective Space, Design Variable Space, Distributed Area, Multi-objective Optimization Problem

1. Introduction

Genetic Algorithm (GA) is expected to be effective for solving Multi-objective Optimization Problems (MOPs), which maximizes or minimizes multiple objective functions at the same time. Recently, Multi-Objective Genetic Algorithm (MOGA), applying GA to MOPs, are getting much attention and a lot of studies have been reported[1]. Generally, it is difficult to obtain the optimized solution satisfying all objective functions because of their trade-offs. Then, it is necessary to obtain Pareto solutions which are not inferior to other solutions in at least one objective function.

In recent years, it is reported that MOGA is applied to engineering design problems in the real-world due to the improvement of computing performance[2][3][4]. In the engineering design problems, it is required not only to obtain high performance Pareto solutions using MOGA but also to analyze and extract design knowledge in the problem. And in order to analyze Pareto solutions obtained by MOGA, it is required to consider both the objective space and the design variable space.

Obayashi obtained Pareto solutions for aircraft configuration problem by MOGA and tried to analyze the obtained Pareto solutions through the visualization of the relationship between fitness values and design variables using Self Organizing Map (SOM)[2]. Kudo *et al.* proposed a visualization method that visualized the geometric distance between data in the design variable space based on their relationship in the objective space and analyzed the relationship between the fitness values and the design variables in the conceptual design optimization problem of hybrid rocket engine[5].

In this paper, we analyze obtained Pareto solutions considering the objective space and the design variable space, and we especially focus on "Non-Correspondence" between two spaces. In this study, we have introduced 3 patterns of Non-Correspondence between the objective space and the design variable space.

- Non-Correspondence in Sequencen
- Non-Correspondence in Spread
- Non-Correspondence in Linear Relationship

We have already reported on the Non-Correspondence in Sequence[6]. In this paper, we define "Non-Correspondence in Spread" and propose the quantitative index to extract Non-Correspondence area in Spread. Non-Correspondence area in Spread is the area where solutions are distributed densely in the objective space but are distributed widely in the design variables space, and vice versa. Moreover, this paper extends the index of non-correspondence to more practical index, which allows a designer to select the contributory design variables and fitness functions and to define the distance function.

This paper applies the proposed method to the trajectory designing optimization problem known as DESTINY (Demonstration and Experiment of Space Technology for INterplanetary voYage)[7] provided by Japan Aerospace Exploration Agency (JAXA). We apply NSGA-II (Non-dominated Sorting Genetic Algorithm-II)[8] to this problem and analyze the extracted Non-Correspondence area in Spread in the obtained Pareto solutions.

2. Non-Correspondence in Spread

2.1 Definition of Non-Correspondence in Spread

In this paper, we focus on Non-Correspondence in Spread. The area with Non-Correspondence in Spread, called Non-Correspondence area in Spread, is defined as the area where solutions are distributed densely in the objective space but are distributed widely in the design variables space, and vice versa. (Hereinafter we call simply " Non-Correspondence area"). Figure 1 shows an example of Non-Correspondence area. In Fig. 1, data 5-6-7-8 are distributed widely in the design variable space compared to the distribution of the objective space. It is important for designer to know this area in Pareto solutions because designer can select design variables from many design patterns in consideration of the cost of design or difficulty level of design.

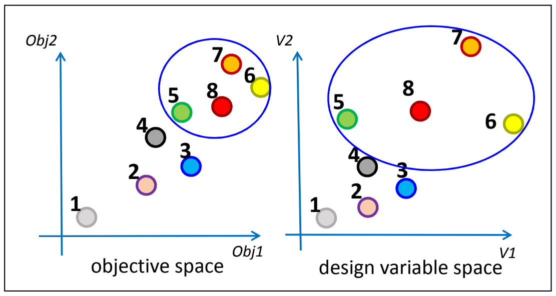


Fig. 1: Non-Correspondence are in Spread

2.2 Index for Non-Correspondence Area in Spread

Here, we define the quantitative index for Non-Correspondence in Spread to extract the Non-Correspondence area. The index is calculated in the following procedure.

1) Define the neighborhood radius ϵ (eq. (1)) in the objective space or the design variable space.
2) Extract the individuals as target individuals within radius ϵ from individual i.
3) Calculate the center of gravity of the target individuals.
4) Calculate the index for Non-Correspondence in Spread v_i according to eq. (2).

By the above procedure, the index v_i is calculated for each individual. The neighborhood radius ϵ is defined by eq. (1). In eq. (1), η denotes the parameter that defines the neighborhood radius, f_{lmax}, f_{lmin} mean the maximum and the minimum fitness values in the Pareto solutions for objective function l, and M_f is the number of objective functions. x_{lmax}, x_{lmin} mean the maximum range and the minimum range of design variables l, and M_d is the number of design variables. If the neighborhood is defined in the objective space, the upper equation in eq. (2) is employed and otherwise the lower equation is employed to calculate the value of index v_i. In eq. (2), d_{dik} is the normalized Euclidean Distance between target individual k and the center of gravity in the design variable space, d_{fik} is that in the objective space, N is the number of the target individuals and v_i is the index for individual i. Individuals with large indexes are distributed densely in the objective space / design variable space and distributed widely in the design variable space / objective space.

$$\epsilon = \begin{cases} \frac{\sqrt{\sum_{l=1}^{M_f}(f_{lmax}-f_{lmin})^2}}{\eta} \\ \text{(Neighborhood was defined in the objective space.)} \\ \frac{\sqrt{\sum_{l=1}^{M_d}(x_{lmax}-x_{lmin})^2}}{\eta} \\ \text{(Neighborhood was defined in the design variable space.)} \end{cases} \tag{1}$$

$$v_i = \begin{cases} \frac{1}{N}\sum_{k=1}^{N}(d_{dik})^2 \\ \text{(Neighborhood was defined in the objective space.)} \\ \frac{1}{N}\sum_{k=1}^{N}(d_{fik})^2 \\ \text{(Neighborhood was defined in the design variable space.)} \end{cases} \tag{2}$$

In the above index, the more the value of every design variable / fitness in the target individuals is different one another, the larger the index v_i becomes. However, a designer often want to analyze or focus on a certain design variable(s) / fitness function(s). Besides, there is often desirable difference value of design variable / fitness while fitness values / design variables are similar one another. For example, designers of rockets want to find the solutions that fitness values are similar, *i.e.* keeping the performances, but the launching date of the rocket have one month distance each other. Then, they can relaunch the rocket expecting the same performance when it had a trouble in the first launch.

The procedure to calculate the index v_i is extended by the selection of design variable / fitness function and the definition of the distance based on Gauss function. The extended index is calculated in the following procedure.

1) Define the Neighborhood radius ϵ (eq. (1)) in the objective space or design variable space.
2) Extract the individuals as target individuals within radius ϵ.
3) Select the desirable design variable or fitness value j.
4) Calculate the average $\overline{x_{ij}}$ of design variable / fitness value j in the target individuals .
5) Calculate the index v_{ij} according to eq. (3).

By the above procedure, the index v_{ij} $(j \in d, f)$ is calculated for each individual. In the following equations, N is the number of the target individuals, s_{ijk} denotes the degree of similarity between individual i and target individual k in j, μ_j is the desirable different value of the design variable / fitness value j, x_{jk} is the value of the design variable / fitness j of individual k, d_{ijk} denotes the difference between x_{jk} and $\overline{x_{ij}}$, and σ is the parameter of Gauss Function. The image of eq. (4) is shown in Fig. 2.

$$v_{ij} = \frac{1}{N}\sum_{k=1}^{N} s_{ijk} \tag{3}$$

$$s_{ijk} = \exp(-\frac{(d_{ijk} - \mu_j)^2}{\sigma^2}) \tag{4}$$

$$d_{ijk} = |x_{jk} - \overline{x_{ij}}| \tag{5}$$

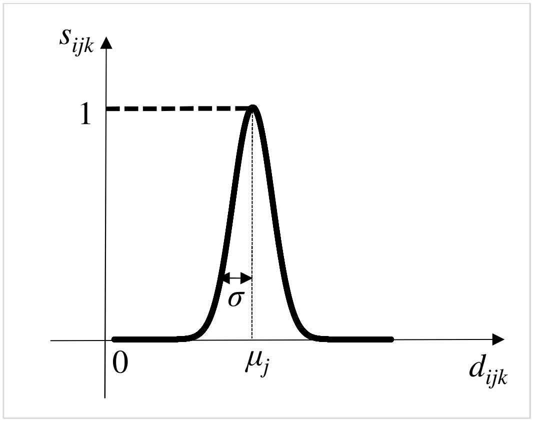


Fig. 2: Image of eq. (4)

When the index v_{ij} for individual i is close to 1, there are some individuals which have similar fitness values / design variables and have the design variable / fitness value j with the difference μ_j one another around the individual i. When eq. (6) is used in the calculation of index v_{ij} instead of eq. (5), what the index v_{ij} is close to 1 means that there are some individuals having the difference μ_j in j from the individual i.

$$d_{ijk} = |x_{jk} - x_{ji}| \tag{6}$$

3. Experiment

In this paper, we applied the above calculation to the trajectory designing optimization problem "DESTINY" provided by JAXA and analyzed the obtained Pareto solutions.

3.1 Trajectory Designing Optimization Problem

The aim of this problem is to reach the moon as early as possible with less fuel and to reduce the degradation of the solar array panel of the spacecraft due to the damage by the radiation of the Van Allen belt. As shown in Fig. 3, the spacecraft is launched by Epsilon Rocket and put elliptical orbits around the earth. Once being put in orbit, the spacecraft is released and accelerates with Ion Engine until it reaches the moon. The spacecraft firstly aims to gain the altitude of perigee and switches to gain the altitude of apogee on the way, then it gradually moves closer to the moon.

This paper tries to optimize of trajectory designing of the spacecraft until it reaches the moon ((1),(2) in Fig. 3). The objective functions, the design variables, and the range of each design variable in this problem are shown in TABLE 1, TABLE 2, and TABLE 3, respectively. $V6$ is used in the case of optimization for 6 objective functions. As shown in TABLE 1, this problem can be expanded to six objective optimization problem. This paper deals with 5 objective functions $Obj1$, $Obj2$, $Obj3$, $Obj4$, $Obj5$ in TABLE 1.

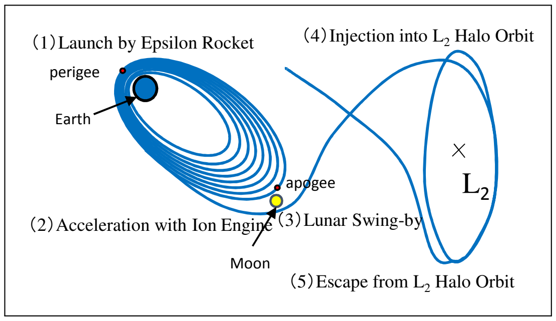

(1) Launch by Epsilon Rocket
(4) Injection into L_2 Halo Orbit
perigee
Earth
apogee
L_2
(2) Acceleration with Ion Engine
(3) Lunar Swing-by
Moon
(5) Escape from L_2 Halo Orbit

Fig. 3: Consept of DESTINY

3.2 Experimental Condition

NSGA-II was applied to the problem described above and 2000 Pareto solutions were obtained. We employed SBX[9] for the crossover and Polynomial Mutation[10]. Crossover rate was 1.0, mutation rate was 0.2, population size was 715, and generation was 100.

Figure 4 shows the visualization result of the distribution of obtained Pareto solutions in (a)the objective space and (b)the design variable space by Multi-Dimensional Scaling (MDS)[11].

Table 1: Objective Functions

$Obj1$	time to reach altitude of 20000km	Min
$Obj2$	IES (Ion Engine System) operation time	Min
$Obj3$	the time to reach the Moon	Min
$Obj4$	the maximum eclipse time	Min
$Obj5$	the time to reach an altitude of 5000km	Min
$Obj6$	Initial mass of the spacecraft	Max

3.3 Extraction of Non-Correspondence Area in Spread

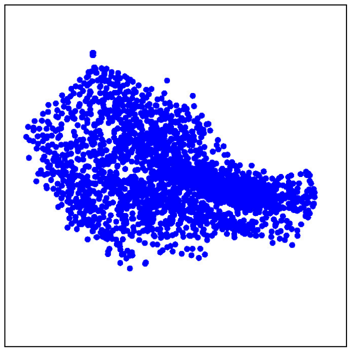
The result of the indexes for Non-Correspondence in Spread calculated by eq. (2) for obtained 2000 Pareto

Table 2: Design variables

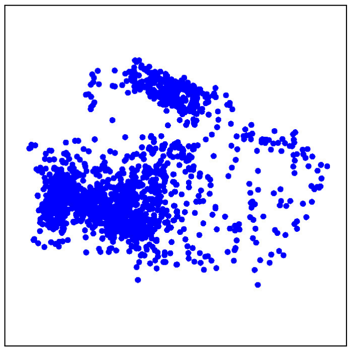
$V1$	: Launching date
$V2$	: Launching time
$V3$	: Switching apogee-perigee date
$V4$	: Range of IES operation time in perigee rise phase
$V5$	: Range of IES operation time in apogee rise phase
$V6$	: Initial mass of spacecraft

Table 3: Ranges of design variables

$V1$	2017/1/1～2018/1/1
$V2$	00:00:00～24:00:00
$V3$	90～365[days]
$V4$	0～180[degrees]
$V5$	0～180[degrees]
$V6$	350～450[kg]



(a) objective space



(b) design variable space

Fig. 4: Distribution of Pareto Solutions

Solutions, in which the neighborhood was defined in the objective space, are shown in Fig. 5. Neighborhood radius ϵ was set as $\eta = 8$ in eq. (1). The parameter of neighborhood radius ϵ was not sensitive and the results were not much changed by the difference of ϵ in the experiments of this paper. The individuals in Fig. 5 are sorted in descending order of the index v_i. The vertical axis shows the value of the index v_i and the horizontal axis shows the individual label.

We focused on the top 50 individuals with large indexes. Figure 6 shows the result of visualization of the distribution in which these 50 individuals are colored by red on the result of the objective space and the design variable space shown in Fig. 4. As shown in Fig. 6, the individuals with red color are distributed widely in the design variable space compared to the distribution in the objective space. We extracted 2 individuals in these 50 individuals and the fitness values and design variables of them are shown in TABLE 4.

In TABLE 4, each fitness value in the second and the third rows is normalized by the maximum and the minimum fitness values of the obtained Pareto solutions into the range of [0,1], and each design variable is normalized by the feasible ranges shown in TABLE 3 into [0,1]. In TABLE 4, though A and B have similar fitness values each other, the design variables are widely different. For example, the launching dates are March and December, the launching times are 1 in the midnight and 8 in the morning, and $V3$ and $V5$ are also different. In this area, there were some individuals that design variables are widely different with similar fitness values.

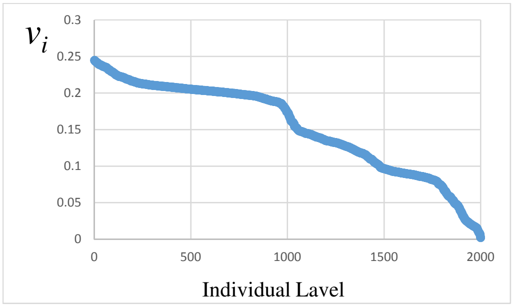
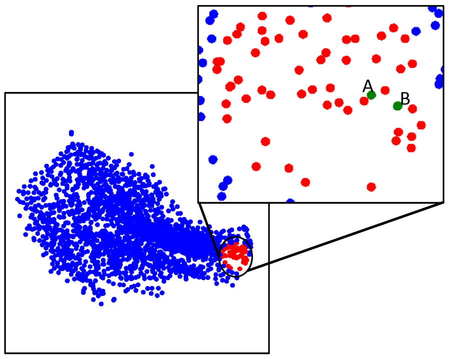


Fig. 5: Value of Index v_i in eq. (2) for each Individual (Neighborhood : Objective Space)

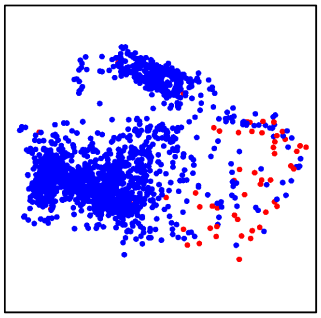
Table 4: fitness values and design variables of selected individuals (A, B)

	Normalized Value		Actual Value	
	A	B	A	B
$Obj1$	0.006	0.011	1434.70	1437.75
$Obj2$	0.846	0.910	8545.60	8713.77
$Obj3$	0.035	0.0005	401.08	395.65
$Obj4$	0.097	0.167	1.524	2.009
$Obj5$	0.018	0.085	217.71	221.07
$V1$	0.201	0.916	2017/3/15	2017/12/1
$V2$	0.051	0.336	01:13:47	08:4:14
$V3$	0.977	0.313	358	175
$V4$	0.999	1.000	179.94	180.00
$V5$	0.818	1.000	147.99	180.00

The result of the indexes in eq. (2), in which the neigh-



(a) objective space



(b) design variable space

Fig. 6: Distribution of Pareto Solutions for Non-Correspondence Area (Neighborhood : Objective Space)

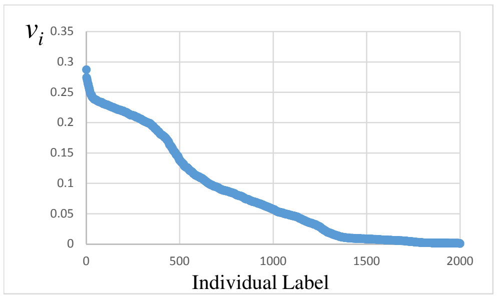
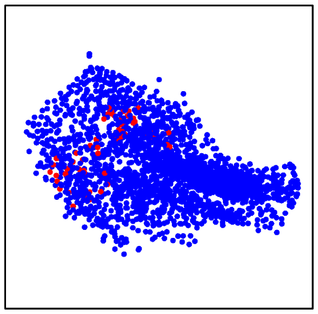
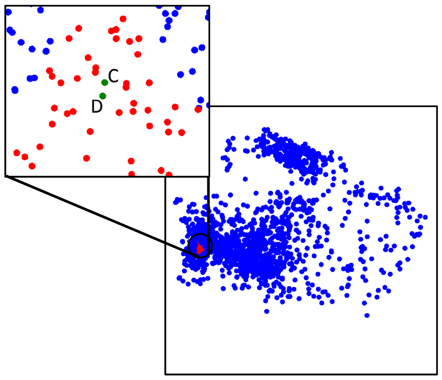


Fig. 7: Value of Index $v_i$in eq. (2) for each Individual (Neighborhood : Design Variable Space)



(a) objective space



(b) design variable space

Fig. 8: Distribution of Pareto Solutions for Non-Correspondence Area (Neighborhood : Design Variable Space)

borhood was defined in the design variable space are shown in Fig. 7. Neighborhood radius ϵ was set as $\eta = 8$ in eq. (1). Figure 7 shows the value of index v_i for each individuals same as Fig. 5.

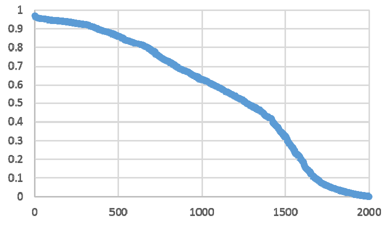
Figure 8 shows the result of the visualization of the distribution of the top 50 individuals with large indexes. As shown in Fig. 8, the individuals with red color are distributed widely in the objective space compared to the distribution in the design variable space. TABLE 5 shows the extracted 2 individuals C and D in Fig. 8 in the same way with TABLE 4. In TABLE 5, though C and D have similar design variables each other, the fitness values are widely different. In this area, there were some individuals that fitness values were very sensitive to the change of design variables. Thus it is required for the designer to choose or design very carefully a Pareto solution in this area.

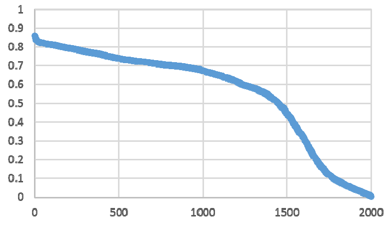
Figure 9 shows the result of the index v_{i1} in eq. (3) for the obtained 2000 Pareto solutions, in which the neighborhood was defined in the objective space and focused on the launching date $V1$ in the design variables. Figure 9(a) shows the result of $\mu_1 = 0.04$ (two weeks), Fig. 9(b) shows the result of $\mu_1 = 0.08$ (one months), and Fig. 9(c) shows the result of $\mu_1 = 0.33$ (four months). Neighborhood radius ϵ was set as $\eta = 8$ in eq. (1) and σ was 0.1. The visualization results of the top 50 individuals with large indexes in each case are shown in Fig. 10.

The fitness values and design variables of individual E and F, G and H, I and J in Fig. 10(a),(b),(c) are shown in TABLE 6(a),(b),(c), respectively. Note that $V1$ and $V2$ are cyclic, so the difference between 2017/12/31 and 2017/1/1 is 1 day

Table 5: fitness values and design variables of selected individuals (C, D)

	Normalized Value		Actual Value	
	C	D	C	D
$Obj1$	0.660	0.857	1801.06	1911.56
$Obj2$	0.226	0.061	6891.16	6452.99
$Obj3$	0.660	0.890	497.49	533.10
$Obj4$	0.156	0.694	1.938	5.689
$Obj5$	0.290	0.385	231.23	236.01
$V1$	0.750	0.750	2017/10/1	2017/10/1
$V2$	0.382	0.385	09:10:59	09:14:16
$V3$	0.038	0.038	100	100
$V4$	0.999	0.985	179.95	177.37
$V5$	0.864	0.841	155.53	151.43


(a) μ_1=0.04


(b) μ_1=0.08

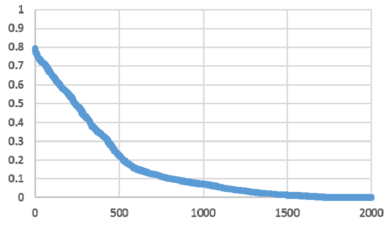
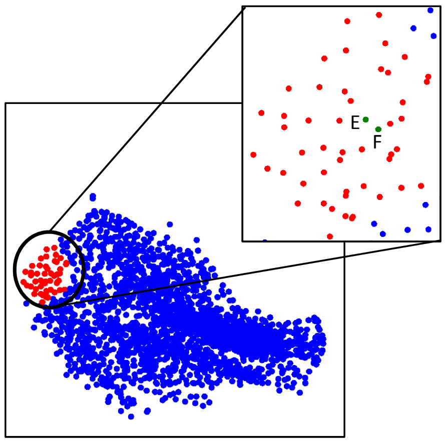
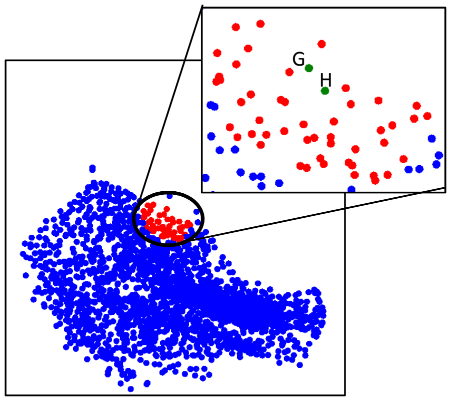

(c) μ_1=0.33

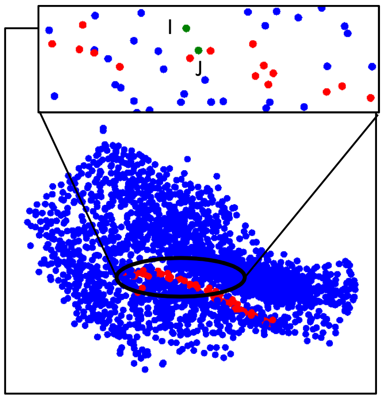
Fig. 9: Value of Index v_{i1} in eq. (3) for each Individual (Neighborhood : Objective Space)



(a) μ_1=0.04



(b) μ_1=0.08



(c) μ_1=0.33

Fig. 10: Distribution of Individuals in the Objective Space

and that between 00:00:00 and 23:59:59 is 1 second. We can see that the Pareto solutions having the desirable difference in $V1$ with similar fitness values could be extracted. In the launch of a Rocket, due to some troubles, the day of launch is often put off. Then, by the extraction of the area where the launching date is desirably different from other individuals having similar fitness values and the selection of a Pareto solution in this area, the launch of the rocket can be carried out on another date keeping the expecting performance (fitness values).

Table 6: fitness values and design variables of selected individuals (E, F, G, H, I, J)

(a)μ_1=0.04

	Normalized Value		Actual Value	
	E	F	E	F
$Obj1$	0.879	0.932	1923.56	1953.71
$Obj2$	0.060	0.025	6449.80	6355.44
$Obj3$	0.886	0.968	532.34	545.1
$Obj4$	0.694	0.674	5.689	5.547
$Obj5$	0.761	0.728	254.73	253.07
$V1$	0.767	0.807	2017/10/7	2017/10/22
$V2$	0.368	0.351	08:49:58	08:25:57
$V3$	0.003	0.009	90	92
$V4$	0.996	1.000	179.37	180.00
$V5$	0.848	0.842	152.68	151.64

(b)μ_1=0.08

	Normalized Value		Actual Value	
	G	H	G	H
$Obj1$	0.651	0.674	1796.35	1808.79
$Obj2$	0.283	0.282	7043.66	7040.11
$Obj3$	0.619	0.653	491.21	496.42
$Obj4$	0.128	0.102	1.742	1.556
$Obj5$	0.640	0.667	248.68	250.02
$V1$	0.786	0.709	2017/10/14	2017/9/16
$V2$	0.309	0.302	07:25:32	07:14:13
$V3$	0	0	90	90
$V4$	1.000	1.000	180.00	180.00
$V5$	0.877	0.876	157.78	157.61

(c)μ_1=0.33

	Normalized Value		Actual Value	
	I	J	I	J
$Obj1$	0.627	0.628	1782.43	1783.20
$Obj2$	0.231	0.177	6905.06	6761.41
$Obj3$	0.798	0.754	518.73	512.08
$Obj4$	0.337	0.368	3.200	3.413
$Obj5$	0.016	0.080	217.62	220.81
$V1$	0.211	0.904	2017/3/19	2017/11/27
$V2$	0.917	0.326	22:01:41	07:50:07
$V3$	0.147	0.150	130	131
$V4$	1.000	0.999	180.00	179.99
$V5$	0.839	0.842	151.02	151.64

4. Conclusion

In this paper, we defined Non-Correspondence in Spread between the objective space and the design variable space. We proposed the quantitative index to extract Non-Correspondence area in Spread. Moreover, this paper extended the index of non-correspondence to more practical index, which allowed a designer to select the contributory design variables or fitness functions and to define the distance function as the desirable difference. This paper applied the proposed method to the trajectory designing optimization problem known as DESTINY provided by JAXA and analyzed the extracted Non-Correspondence area in Spread in the obtained Pareto solutions. This paper showed that the Pareto solutions having the desirable difference in the launching date $V1$ with similar fitness values could be extracted. For the future work, we will apply to other problems with more objective functions and feedback the defined index and the extracted knowledge into the search and study Non-Correspondence in Linear Relationship.

References

[1] K. Deb, *Multi-objective optimization using evolutionary algorithms*. Wiley, 2001.

[2] S. Obayashi, “Multiobjective design optimization of aircraft configuration (in japanese),” *The Japanese Society for Artificial Intelligence*, vol. 18, pp. 495–501, 2003.

[3] K. Deb, “Unveiling innovative design principles by means of multiple conflicting objectives,” *Engineering Optimization*, vol. 35, no. 5, pp. 445–470, 2003.

[4] K. H. Akira Oyama, Yasuhiro Kawakatsu, “Application of multiobjective design exploration to trajectory design of the next-generation solar physics satellite,” *Japanese Society for Evolutionary Computation*, 2010.

[5] F. Kudo and T. Yoshikawa, “Knowledge extraction in multi-objective optimization problem based on visualization of pareto solutions,” *WCCI 2012 IEEE World Congress on Computational Intelligence*, pp. 860–865, 2012.

[6] T. Yoshida and T. Yoshikawa, “An extraction of non-correspondence area between objective space and design variable space based on order correlation of distance relation（in japanese）,” *The Japanese Society for Artificial Intelligence*, 2013.

[7] A. L ́ opez, A. Oyama, and K. Fujii, “Evaluating two evolutionary approaches to solve a many-objective space trajectory design problem,” *The Japanese Society for Evolutionary Computation*, 2012.

[8] K. Deb, *A Fast and Elitist Multiobjective Genetic Algorithm : NSGA-II*. 2002.

[9] K. Deb and R. Agrawal, “Simulated binary crossover for continuous search space,” *Complex Systems*, vol. 1, no. 9, pp. 115–148, 1994.

[10] K. Deb and M. Goyal, “A combined genetic adaptive search (geneas) for engineering design,” *Computer Science and Informatics*, vol. 26, pp. 30–45, 1996.

[11] J. W. Sammon, “A nonlinear mapping for data structure analysis,” *IEEE Transactions on computers*, vol. 18, no. 5, pp. 401–409, 1969.

Task Scheduling Algorithm for Multicore Processor Systems with Turbo Boost and Hyper-Threading

Yosuke Wakisaka, Naoki Shibata, Keiichi Yasumoto, Minoru Ito
Nara Institute of Science and Technology
Nara, Japan
{*yosuke-w, n-sibata, yasumoto, ito*}*@is.naist.jp*

Junji Kitamichi
The University of Aizu
Fukushima, Japan
kitamiti@u-aizu.ac.jp

***Abstract*—In this paper, we propose a task scheduling algorithm for multiprocessor systems with Turbo Boost and Hyper-Threading technologies. The proposed algorithm minimizes the total computation time taking account of dynamic changes of the processing speed by the two technologies, in addition to the network contention among the processors. We constructed a clock speed model with which the changes of processing speed with Turbo Boost and Hyper-threading can be estimated for various processor usage patterns. We then constructed a new scheduling algorithm that minimizes the total execution time of a task graph considering network contention and the two technologies. We evaluated the proposed algorithm by simulations and experiments with a multi-processor system consisting of 4 PCs. In the experiment, the proposed algorithm produced a schedule that reduces the total execution time by 36% compared to conventional methods which are straightforward extensions of an existing method.**

***Keywords*-Task scheduling algorithm, Multicore, Turbo Boost, Hyper-Threading**

I. Introduction

In recent years, multicore processors have been widely used in various computing environments including data centers and supercomputers. Since the produced heat by the processors is limiting their clock speed, technologies that change clock speed according to the temperature and power consumption of the processor are employed in the latest processors. Such technologies are used in the processors manufactured by Intel and AMD, and they are called Turbo Boost and Turbo Core[1]. We refer to both of the technologies by Turbo Boost, hereafter. Turbo Boost is a technique for increasing the clock speed of some processor cores within the thermal specification when other cores are inactive and the temperature of the processor die is low. Some processors also employ a technology called Hyper-Threading[2] that enables the physical resources of one physical processor to be shared between two or more logical cores to improve the overall throughput.

Task scheduling methods are methods for assigning a series of tasks to a parallel processing system. If we simply apply existing task scheduling methods such as [3], [4], [5] to a system consisting of multicore processors, many tasks are likely to be assigned to a same multicore processor because communication between cores on a same processor die is much faster than communication between dies. In this case, Turbo Boost cannot drastically increase the clock speed of the cores since almost all of the processor cores are active. In some cases, distributing tasks over different dies yields a better schedule because of the boosted clock speed. Thus, we need a scheduling algorithm that takes those technologies into account in order to derive the optimal schedule for systems with these technologies. There is difficulty for some existing scheduling algorithms to consider these technologies, since if tasks are scheduled by those methods that assigns tasks one by one to each processor core, the clock speed for executing the task can be slower than the estimation at the time of assignment, since the clock speed slows down as the subsequent tasks are assigned to the other processors on the same die.

In this paper, we propose a new task scheduling method that takes account of both Turbo Boost and Hyper-Threading technologies and minimizes the processing time. The proposed method takes a task graph specifying dependency among tasks by a directed acyclic graph (DAG) and a processor graph specifying the network topology among available processors, and outputs a schedule which is an assignment of a processor to each task. We constructed a clock speed model for estimating the change of effective processing speed of each core with Turbo Boost and Hyper-Threading. We then constructed a new scheduling algorithm that can more accurately estimate the effective clock speed of each core, utilizing the proposed model.

In order to evaluate the proposed method, we conducted simulations and experiments with actual processors. We compared the proposed algorithm with two algorithms which are extension of the Sinnen's scheduling algorithm[6] that takes account of network contention, and our clock speed model is integrated in a straightforward way. As a result, our method reduced the total processing time by up to 36% in the experiments with a real system. The difference between the scheduled processing time and the actual processing time was 5% in average, and thus we confirmed the task scheduling by our method is effective in the real environments.

II. Related Work

There are many kinds of task scheduling algorithms. In this paper, we assume that task scheduling is assigning a processor to each task, where the dependence of the tasks is represented by a directed acyclic graph(DAG). The problem to find the optimal schedule is NP-hard[6], and there are many heuristic algorithms for the problems.

List scheduling is a classical task scheduling method that assigns the processor that can finish each task to the task in order of a given priority of the tasks. The priority can be given by performing topological sorting on the dependence graph of the tasks, for example. Sinnen et al. extended the classical list scheduling algorithm, and proposed a new method that takes account of the communication delay and network contention[6]. This method assigns the input tasks to the processors while bandwidth in communication paths are reserved for each task so as to minimize the total processing time.

Song et al. proposed a dynamic task scheduling method that executes linear algebraic algorithms on multicore systems with shared or distributed memories. This method scales well, but only applicable to specific tasks.

Jongsoo et al. proposed a task scheduling program called Team scheduling that assigns stream programs to multicore processors [8]. Existing stream programs adjust data transmission timings depending on the data size in the given stream graph so as to efficiently utilize buffers of the processors. This technique is called Amortize. However, deadlock may occur when a large stream graph is input. Team scheduling achieves deadlock-freeness by applying Amortize to a part of the stream graph and suppressing buffer utilization. Moreover, this method achieves better throughput for the same buffer size as the existing methods.

Gotoda et al. proposed a task scheduling method which minimizes recovery time from a single processor failure in multicore processor environments[7]. This method is based on the algorithm [6] proposed by Sinnen et al., and assigns tasks to processors considering both network contentions and recovery time in case of failure of a multicore processor, and produces the optimal task schedule.

As far as we surveyed, there is no existing methods that consider the changes of clock speed by Turbo Boost or Hyper-Threading on a multicore processor system. Unlike these existing methods mentioned above, we propose a new method which targets the environments with a multicore processor system with Turbo Boost and Hyper-Threading. The proposed scheduling method minimizes the total execution time of the input task graph taking account of the two technologies and network contention.

III. Modeling Turbo Boost and Hyper-Threading

In this section, we briefly describe Turbo Boost and Hyper-Threading technologies. The, we describe our model for estimating effective clock speeds determined by the two technologies.

A. Turbo Boost and Hyper-Threading

Turbo Boost is a technology for boosting the clock speed for each core according to the computing load on the processor die. It monitors the temperature and the electric power consumed by the die and dynamically increase the clock speed of some cores if other cores are not used[1]. In this paper, we assume that it determines the clock speed only by the computing load of the all cores on the die.

Hyper Threading is a technology for sharing hardware resources of a physical core among multiple logical cores[2]. When more than one threads are executed on a physical core, the performance of the threads are lower than when only one thread is executed on the physical core. We model this change of execution speed by regarding the clock speed as the index of execution speed at each core, and lowering this speed index of each logical core according to the load on the other logical cores. Hereafter, we call this speed index *effective clock speed.*

B. Modeling

As mentioned above, Turbo Boost and Hyper-Threading technologies can be modeled so that it automatically changes the effective clock speed according to the kind of computational loads on each core. We also assume that the effective clock speed is instantly changed according to the change of core usage, in the course of task execution. It is also assumed that each processor is in one of the following states: (1) idle, (2) computation heavy, (3) memory access heavy, and (4) in-between of (2) and (3).

In order to construct the model, we developed a program that consists of two parts: the part that swaps two randomly selected elements of an 80MB array, and a part that iterates a simple loop staying in the L1 cache. The program repeats executing these two parts in turn. We adjusted the number of loops in the second part of the program, and measured the time to execute this program on multiple cores simultaneously. We specified the processor affinity to each thread so that all threads are executed on the specified cores. We calculated the effective clock speed from the measured processing time.

We used a PC with Intel Core i7 3770T (2.5GHz, 4 physical processors, 8 logical processors, single socket), 16GB memory, Windows 7 (64bit), Java SE (1.6.0 21, 64bit). We used Intel Turbo Boost monitor (Ver2.5) and CPU-Z (Ver1.61.3) to measure the physical clock speed. We first observed how the physical clock speed changes when the number of active physical cores is changed. We show the result of measurement in Table I. The left column shows the processor state, where the 4 pairs represent the usage of four physical processors and each pair like [2, 1] indicates the usage of logical processors within the corresponding physical processor. The right column shows the clock speed for the corresponding processor usage. The table shows that the clock speed does not depend on the ratio of memory access, but depends only on the number of active physical cores.

In our proposed scheduling method, Hyper-Threading is used only if tasks are already assigned to all physical cores. Thus, we assume that when two logical threads are running on a physical core, the effective clock speed only depends on the ratio of memory access at each logical core. We calculated the effective clock speed from the ratio of execution time by each logical processor to the execution time when one thread is executed on each physical core. The results are shown in Table II.

We constructed a model for effective clock speed from the results above, and we will determine the clock speed from the usage of the processor at which task nodes are assigned using this model.

IV. Problem Formulation

In this section, we formulate the problem of task scheduling taking account of Turbo Boost and Hyper-Threading technologies. The symbols used in this paper is summarized in Table.III.

The task scheduling is to find the schedule S that minimizes the total execution time $lt(S)$ from the given task graph G and processor graph N. A schedule is a tuple of an assignment of a processor to each task, the starting and finishing time of each task node, and the information of bandwidth reservation on each processor link.

A task graph G is a DAG in which each node represents a task to be performed. Each node in a task graph is called a task node. The amount of computation to finish task node v is denoted by $C_{comp}(v)$. A directed arc in the graph is called a task link, and a task link from node v_a to v_b indicates that task v_a must be completed before task v_b begins. A task link e also represents communication between two nodes, and the amount of data transfer for this link is denoted $C_{comm}(e)$. The set of all task nodes and the set of all task links are denoted **V** and **E**, respectively. Fig. 1 shows an example of a task graph consisting of 3 task nodes and 2 task links.

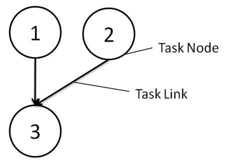


Figure 1: Example task graph

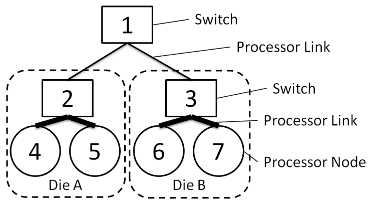


Figure 2: Example processor graph

A processor graph is a graph that represents the network topology between processors. A node with only one link is called a processor node, that corresponds to one processor core. A node with two or more links is called a switch. A switch is not capable of executing a task but only relays communication. An edge is called a processor link, and it represents a bidirectional communication link between processors and switches. One multicore processor is represented by multiple processor nodes, a switch and processor links connecting them. The set of all processor nodes and the set of all processor links are denoted **P** and **R**, respectively. $freq(p, s)$ denotes a function that gives the effective clock speed of processor p from the state s of all cores on the same die. Fig. 2 shows an example of a processor graph consisting of three processor cores and three switches, or two multicore processors and a switch.

In this paper, we use a network contention model based on the model proposed by Sinnen et al.[6], and we make the following assumptions. When data transfer is performed over network links between two processor nodes, due to bandwidth limitation these network links cannot perform

Table I: Effective clock speed with Turbo Boost

Processor states	Effective clock speed
[2, 1], [1, 1], [1, 1], [1, 1]	3.7
[2, 1], [2, 1], [1, 1], [1, 1]	3.5
[2, 1], [2, 1], [2, 1], [1, 1]	3.3
[2, 1], [2, 1], [2, 1], [2, 1]	3.1
[3, 1], [1, 1], [1, 1], [1, 1]	3.7
[3, 1], [3, 1], [1, 1], [1, 1]	3.5
[3, 1], [3, 1], [3, 1], [1, 1]	3.3
[3, 1], [3, 1], [3, 1], [3, 1]	3.1
[4, 1], [1, 1], [1, 1], [1, 1]	3.7
[4, 1], [4, 1], [1, 1], [1, 1]	3.5
[4, 1], [4, 1], [4, 1], [1, 1]	3.3
[4, 1], [4, 1], [4, 1], [4, 1]	3.1
[1, 1], [1, 1], [1, 1], [1, 1]	2.5
[2, 2], [2, 2], [2, 2], [2, 2]	2.6
[3, 3], [3, 3], [3, 3] [3, 3]	2.3
[4, 4], [4, 4], [4, 4], [4, 4]	2.5

Processor state: 1:idle, 2:computation heavy, 3:memory access heavy, 4:in-between of 2 and 3

Table II: Effective clock speed with Hyper-Threading

Processor states	Ratio of exec. times	Effective clock speed
[1 , 1]	1.0	2.5
[2 , 2]	0.84	2.6
[3 , 3]	0.76	2.3
[4 , 4]	0.79	2.5

Table III: Symbols used in this paper

Symbol	Meaning
V	Set of all task nodes
E	Set os all task links
P	Set of all processor nodes
R	Set of all processor links
$lt(S)$	Completion time of the last task node in schedule S
G	Task graph
N	Processor graph
$C_{comp}(v)$	Computation cost for task node $v \in V$
$C_{comm}(e)$	Communication cost for task link $e \in E$
$freq(s)$	Effective clock speed determined from processor state s
n_i	Task node for the i-th task
$w(v)$	Execution time for task node v
$c(e)$	Communication time at task link e
$proc(n)$	Processor assigned to task node $n \in\ V$
$pred(n_i)$	Set of all parent nodes of n_i

other data transfers. We also assume the following conditions are satisfied: if data are transferred through a series of processor links, downstream links cannot start data transfer before upstream links; communication inside a same die finishes instantly; all processors on a same die share a network interface that can be used to communicate with devices outside the die; all communication links outside dies have the same bandwidth. Data transfer corrensponding to task link e over a communication link outside dies requires $C_{comm}(e)$ length of time. One processor can execute only one task at a time. A task node cannot be executed until all execution of parent nodes and all corresponding data transfers are finished. It takes $C_{comp}(v)/freq(p,s)$ length of time for processor node p to finish execution of task node v, where s is the state of all cores on the same die as p.

V. Proposed Algorithm

In this section, we explain our scheduling algorithm. This scheduling problem is known as NP-Hard[6], and thus we propose a heuristic algorithm considering both network contention and change of clock speed with Turbo Boost and Hyper-Threading technologies by extending the scheduling algorithm proposed by Sinnen et al.[6]. We use the clock speed model described in Section 3 for this purpose.

Algorithm 1 List scheduling

```
   INPUT: Task graph $G = (\mathbf{V}, \mathbf{E}, w, c)$ and processor graph $H = (\mathbf{P}, \mathbf{R})$.
1: Sort nodes $n \in V$ into list $L$, according to priority scheme and precedence constraints.
2: for each $n \in L$ do do
3:     Find processor $p \in \mathbf{P}$ that allows earliest finish time of $n$.
4:     Schedule $n$ on $p$.
5: end for
```

Algorithm 2 Scheduling considering network contention

```
    INPUT: Task graph $G = (\mathbf{V}, \mathbf{E}, w, c)$ and processor graph $H = (\mathbf{P}, \mathbf{R})$.
 1: Sort nodes $n_j \in V$ into list $L$ in descending order of $bl$, according to precedence constraints.
 2: for each $n \in L$ do do
 3:     Find processor $p \in \mathbf{P}$ that allows earliest finish time of $n_j$, taking account of network bandwidth usage.
 4:     for each $n_i \in pred(n_j)$ in a definite order do
 5:         if $proc(n_i) \neq p$ then then
 6:             determine route $R = [L_1, L_2, ..., L_l]$ from $proc(n_i)$ to $p$.
 7:             schedule $e_{ij}$ on $R$.
 8:         end if
 9:     end for
10:     schedule $n_j$ on $p$.
11: end for
12: return the schedule.
```

Scheduling algorithms based on the list scheduling do not perform well with systems where clock speeds of the processors are controlled by Turbo Boost or Hyper-Threading. This is because the list scheduling assigns a processor to each task node in turn, and it cannot know the effective clock speed for each task during assignment, since the effective clock speed is influenced by the execution of succeeding tasks. The proposed method tentatively assigns processors to the all succeeding tasks assuming that these succeeding tasks are executed in a predetermined fixed clock speed. Then, it estimates the execution time of the tasks by applying the proposed model for the effective clock speed. Although this execution time is calculated using the tentative schedule, we regard this execution time as an approximation of the actual execution time and make the schedule based on it.

Hereafter, we first explain the traditional list scheduling algorithm, followed by the extension by Sinnen et al. for considering network contention. Then, we give the details of the proposed algorithm.

A. Existing Algorithms

The classical list scheduling algorithm is shown in Algorithm 1. In the list scheduling, each task is assigned to the processor that allows the earliest finish time of the task, in descending order of bl, that is the length of remaining schedule.

The algorithm proposed by Sinnen, et al. is shown in Algorithm 2. Below, we give explanation for the pseudocode.

Line 2 to 11: Each task node $n_j \in L$ is assigned a processor in order of the position in L.
Line 3: The processor assigned to n_j is determined taking account of network bandwidth usage. Reserved bandwidth in line 7 is referred here.

Algorithm 3 The proposed scheduling algorithm

```
    INPUT: Task graph $G = (V, E, v_{start}, C_{comp}, C_{comm})$, processor graph $N = (P, R)$ and frequency model $freq$
1:  $S_{prev}$ = an empty schedule
2:  Sort nodes in $V$ into list $L$ in descending order of the length of succeeding tasks, according to precedence constraints.
3:  for $n_i \in L$ $n_i$ is the first element in $L$ do
4:      $S_{cur}$ = an empty schedule, $T_{cur} = \infty$
5:      for each $p_i \in P$ do
6:          $S_{cand} = S_{prev}$
7:          for each preceding task $n_j$ of $n_i$ do
8:              if $p_i$ is not assigned to $n_j$ on $S_{cand}$ then
9:                  Determine route $r = [L_1, L_2, ..., L_l]$ from the processor assigned to $n_j$ to $p_i$
10:                 Reserve bandwidth $C_{comm}$(the task link from $n_j$ to $n_i$) on route $r$ in $S_{cand}$
11:             end if
12:         end for
13:         Calculate finishing time of $n_i$ including communication time assuming that $n_i$ is executed on $p_i$ with the fixed clock speed, and add the information of finishing time to $S_{cand}$
14:         Schedule all unassigned tasks in $S_{cand}$ using Algorithm 2 and substitute the resulting schedule for $S'_{cand}$
15:         Calculate execution time of each task node in $S'_{cand}$ with the proposed model for effective clock speed
16:         if the total execution time of $S'_{cand} < T_{cur}$ then
17:             $S_{cur} = S_{cand}, T_{cur}$ = the total execution time of $S'_{cand}$
18:         end if
19:     end for
20:     Remove $n_i$ from $L$
21:     $S_{prev} = S_{cur}$
22: end for
23: return $S_{cur}$
```

Line 4 to 9: Bandwidth of e_{ij} is reserved for the network route between the processor assigned to n_i (which is the parent node of n_j) to the processor assigned to n_j.

B. Scheduling Considering Frequency Change

The pseudo code for the proposed algorithm is shown in Algorithm 3. In the algorithm, S_{prev}, S_{cur} and S_{cand} retain portions of schedules in which only a part of the all assignment is specified. The total execution time for these incomplete schedules can be calculated by assigning processors to the all unassigned tasks using algorithm 2, and then applying our clock speed model. S_{prev} retains the best incomplete schedule in which the all tasks prior to n_i are assigned, and other tasks are not assigned. S_{cur} and T_{cur} retain the current best incomplete schedule in which n_i and the prior tasks are assigned, and the corresponding execution time, respectively. Below, we give explanation for the pseudocode.

Line 3 to 22: A processor is assigned to each task node.
Line 5 to 19: Each processor p_i is tentatively assigned to the first task n_i in list L so that the processor that achieves the earliest finish time of the all tasks is found.
Line 7 to 13: Processor p_i is assigned to task link n_i.
Line 14: The all succeeding tasks after n_i are scheduled assuming that they are executed in a fixed clock speed.
Line 15: The total execution time for this schedule is calculated using the proposed clock speed model.
Line 16 to 18: The best processor to be assigned to n_i is determined by the execution time.

VI. EVALUATION

In order to evaluate the efficiency of the schedule generated by the proposed method and the accuracy of the proposed model for effective clock speed, we conducted experiments using a real system and simulation-based comparisons.

A. Compared Methods

As we described in Section 2, we could not find an existing task scheduling method considering Turbo Boost or Hyper-Threading. In order to make fair comparisons, we integrated our clock speed model into the Sinnen's scheduling algorithm in a straightforward way and made two methods: **SinnenPhysical** that is a scheduling algorithm that tries to assign only physical processors to the tasks, and **SinnenLogical** that tries to assign all logical processors to the tasks. These two methods are extended so that they utilize the clock speed model when choosing the best processor that allows the earliest finishing time of each task[1].

As a preliminary experiment, we compared **SinnenPhysical** and **SinnenLogical** with the original method proposed by Sinnen et al. that does not consider the changes of clock speed at all, and confirmed that **SinnenPhysical** and **SinnenLogical** generate better schedules than the original algorithm for our system configuration.

B. Configuration

We used a PC with Intel Core i7 3770T (2.5GHz, 4 physical processors, 8 logical processors, single socket), 16GB memory, Windows 7 (64bit), and Intel 82579V Gigabit Ethernet Controller as a computing node. The system consists of four of these PCs connected with Gigabit Ethernet. We implemented the programs to execute the scheduled tasks using the standard TCP socket with Java SE (1.6.0 21, 64bit). In order to eliminate the influence of the operating system, we stopped the background tasks except the ones required for continuing the minimum operations of the OS. We set the threads' affinities to each of processor cores so that each task node is executed on the core specified by the schedule. We tested the two real network topologies shown in Fig. 3. For the simulation, we also tested a fully-connected network topology. We used 420Mbps as the bandwidth of processor

[1]At Line 3 in Algorithm 2, the processor assigned to n_j is determined taking account of the two technologies. Only already assigned tasks are considered to estimate the effective clock speed.

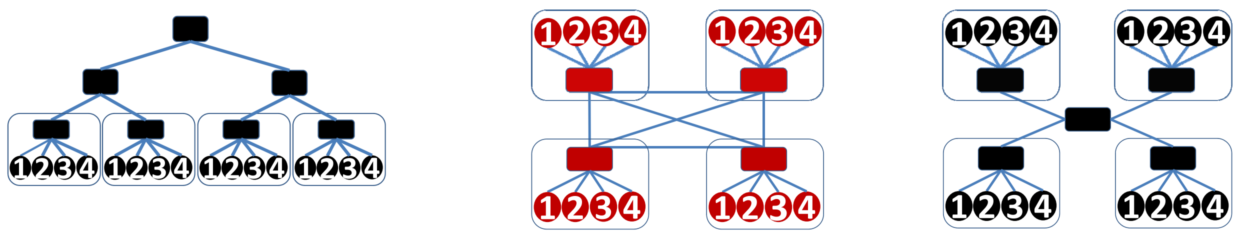


Figure 3: Processor graphs used in evaluation

links outside the dies, that is obtained by measuring the network bandwidth on the above system.

We used task graphs for Robot Control and Sparse Matrix Solver from the Standard Task Graph Set[9], [10] in our evaluation. The Sparse Matrix Solver has 98 nodes and 177 links and represents a sparse matrix solver of an electronic circuit simulation generated by the OSCAR FORTRAN compiler. This graph has relatively high level of parallelism. The Robot Control has 90 nodes and 135 links. The Robot Control task graph represents a parallel task for Newton-Euler dynamic control calculation for the 6-degrees-of-freedom Stanford manipulator. The Robot Control task has lower level of parallelism compared to the Sparse Matrix Solver. Since the ratio of computation and memory access is not specified in these task graphs, we used the 4th state of the processor load, which is in-between of computation heavy and memory-access heavy states described in Section 3, for the all task nodes.

C. *Efficiency of Generated Schedules*

We evaluated the efficiency of generated schedules by comparing the generated schedules with the proposed method and the two comparison methods. We calculated the execution time of generated schedules with simulation, and measured the execution time on the real system by assigning and executing tasks on the processors in the real system. We performed simulations with the combinations of the two task graphs and the three processor graphs. In the experiments, we combined the two task graphs and the two processor graphs except the fully-connected topology.

We compared the total execution time of the schedules generated by the proposed method to the schedules generated by the compared methods. The simulation results and the experimental results are shown in Fig. 4 and 5, respectively. These results show that the proposed method reduced the total execution time by up to 43% in the simulation, and up to 36% with the real system. We can see that the proposed method has greater effect on the Sparse Matrix Solver task than on the Robot Control task. This is because the Robot Control task has less parallelism, and this limits the freedom for scheduler to choose a processor for each task. Thus, the algorithm has smaller freedom for controlling the generated schedule. The results also show that our method has greater effect on the tree-shaped or the star-shaped network topology than the fully-connected topology. This is because the fully-connected topology requires less communication time than other two toplogies.

D. *Accuracy of Effective Clock Speed Model*

In order to evaluate the accuracy of the proposed model for effective clock speed, we compared the total execution time of the task graphs on the real system with the simulated results. Fig. 6 and 7 show the results. In the experiment, the error of the estimated execution time was no more than 7%, and the average error was 4% . We also chose 20 random task nodes from the graphs and compared the distribution of the execution time for each of the nodes with the simulated results. Fig. 7 shows the 90%-tiles of the measured execution time with the simulated time. The maximum error was 16% and the average error was 8.5%.

The difference between the results in the simulation and the experiments is probably coming from the fluctuation of network bandwidth and the processor load by the background tasks in the OS. However, the errors in the results are not significant, and the proposed clock speed model is sufficient for estimating the execution time of each task with Turbo Boost and Hyper-Threading.

VII. Conclusion

In this paper, we formulated the problem for generating task schedules minimizing the total execution time of task graphs considering network contention and multicore processors with Turbo Boost and Hyper-Threading technologies. We also modeled the two technologies so that the effective processing speed of each core can be estimated. Then, we developed a new task scheduling algorithm for the problem. In the experiments for evaluation, the proposed algorithm produced a schedule that is 36% faster than the compared methods. Since the proposed method makes the system finish execution of the tasks earlier, it also contributes for saving power consumption of the whole system. As a part of our future work, we are going to make our algorithm capable of accepting multiple task graphs in real time.

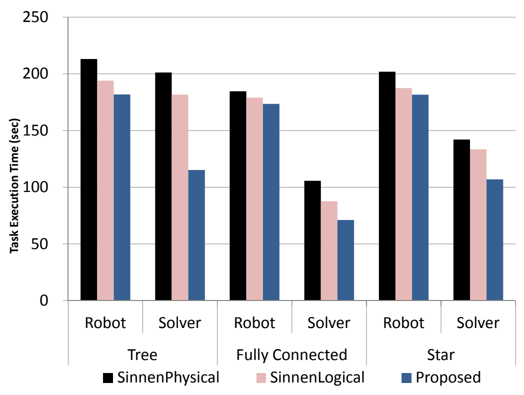


Figure 4: Simulation result

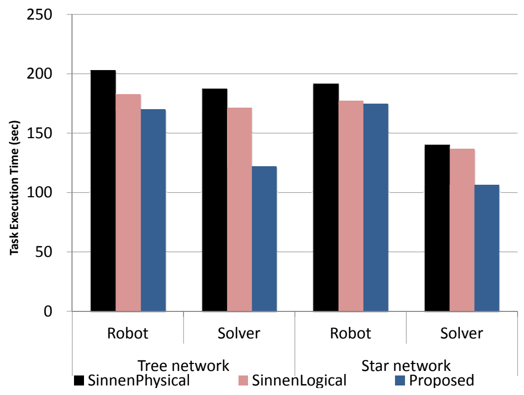


Figure 5: Results with real devices

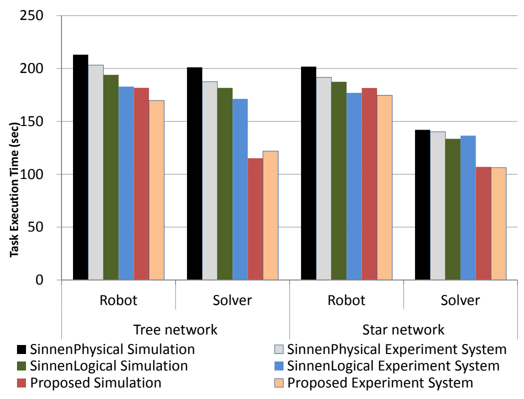


Figure 6: Comparison between estimated and real execution time of the whole task graph

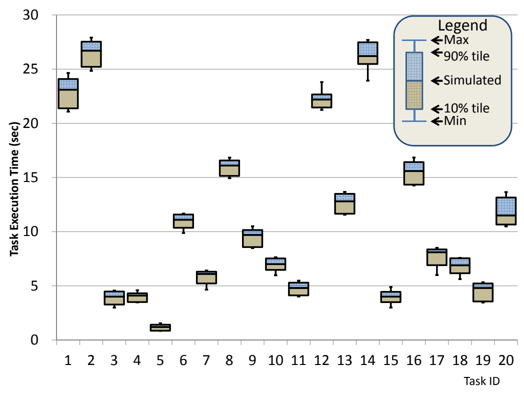


Figure 7: Comparison between estimated and real execution time of of each task node

References

[1] Intel: Intel turbo boost technology in intel core microarchitecture(nehalem) based processors. Technical report, Intel (2008)

[2] Marr, D.T., Group, D.P., Corp, I.: Hyper-threading technology architecture and microarchitecture. Intel Technology Journal **6**(1) (2002) 4–15

[3] Kwok, Y., Ahamad, I.: Static scheduling algorithms for allocating directed task graphs to multprocessors. ACM Computing Surveys(CSUR) **31**(4) (dec 1999) 406–471

[4] Sinnen, O., To, A., Kaur, M.: Contention-aware scheduling with task duplication. Journal of Parallel and Distributed Computing **71**(1) (oct 2011) 77–86

[5] Sinnen, O., Sousa, L., Sandnes, F.: Toward a realistic task scheduling model. Parallel and Distributed Systems, IEEE Transactions on **17**(3) (mar 2006) 263–275

[6] Sinnen, O., Sousa, L.: Communication contention in task scheduling. Parallel and Distributed Systems, IEEE Transactions on **16**(6) (jun 2005) 503–515

[7] Gotoda, S., Ito, M., Shibata, N.: Task scheduling algorithm for multicore processor system for minimizing recovery time in case of single node fault. In: In Proceedings of Cluster, Cloud and Grid Computing (CCGrid), 12th IEEE/ACM International Symposium on. (may 2012) 260–267

[8] Park, J., Dally, W.J.: Buffer-space efficient and deadlock-free scheduling of stream applications on multi-core architectures. In: Proceedings of the 22nd ACM symposium on Parallelism in algorithms and architectures. (2010) 1–10

[9] Tobita, T., Kasahara, H.: A standard task graph set for fair evaluation of multiprocessor scheduling algorithms. Journal of Scheduling **5**(5) (2002) 379–394

[10] Tobita, T., Kasahara, H.: Standard task graph set Home Page @ONELINE (2012) http://www.kasahara.elec.waseda.ac.jp/schedule/.

Multiple Precision Integer Multiplication on GPUs

Koji Kitano and Noriyuki Fujimoto
Graduate School of Science, Osaka Prefecture University, Sakai-shi, Osaka, Japan

Abstract— *This paper addresses multiple precision integer multiplication on GPUs. In this paper, we propose a novel data-structure named a product digit table and present a GPU algorithm to perform the multiplication with the product digit table. Experimental results on a 3.10 GHz Intel Core i3-2100 CPU and an NVIDIA GeForce GTX480 GPU show that the proposed GPU algorithm respectively runs over 71.4 times and 12.8 times faster than NTL library and GMP library, two of common libraries for single thread multiple precision arithmetic on CPUs. Another experiments show also that the proposed GPU algorithm is faster than the fastest existing GPU algorithm based on FFT multiplication if bit lengths of given two multiple precision integers are different.*

Keywords: multiple precision integer, parallel multiplication, GPGPU, CUDA

1. Introduction

Multiple precision integer arithmetic finds several applications, for example in primality test of a large number which is important in public-key cryptography. There are many works on multiplication that is of wide application of all other multiple precision arithmetics. These representatives include Karatsuba method [7], Toom-Cook method [19], 4-way method [22], 5-way method [22], and Strassen FFT multiplication [16]. Time complexities of these methods are $O(n^{1.585})$,$O(n^{1.465})$, $O(n^{1.404})$, and $O(n^{1.365})$ respectively where n is the number of bits of a multiplicand and a multiplier, but in the study [21], Zuras compares implementation of these methods in C language and assembly language on HP-9000/720, and reports that the naive $O(n^2)$ method is the best for small numbers and that all naive methods are faster than FFT multiplication which has advantage in time complexity for large numbers. Finally, Zuras concludes that FFT multiplication is not always the fastest even for extremely large numbers ($>$ 37,000,000 bits).

On the other hand, research on GPGPU (General-Purpose computation on Graphics Processing Units) has much attention in recent years, and several works on multiple precision integer multiplication with a GPU are known. The fastest work of these existing works is the implementation [2] of FFT multiplication on GPUs. This method puts multiple precision integers into a 2^{393216} -ary number and computes multiple precision integer multiplication by Karatsuba method except that one digit $\times$ one digit is performed by FFT multiplication. This method is fast if the bit lengths of a multiplier and a multiplicand are the same and multiples of 393216. However, it is significantly slowed down if the bit lengths of a multiplier and a multiplicand are different or not multiples of 393216, because in such a case given two numbers are promoted to numbers of the bit length that is the smallest multiple of 393216 not smaller than the bit length of the larger number of the two numbers [2]. Moreover, currently required bit length in cryptology is a few thousands.

In this paper, we propose a novel data-structure named a product digit table and presents a GPU algorithm to perform the multiplication with the product digit table. The proposed method is different from the FFT multiplication in that there is no need to promote given two numbers even if the bit lengths of the two numbers are not the same. In addition, since the proposed method represents numbers in 2^{32} -ary, efficiency loss is relatively low even for numbers of a few thousand bits.

The remainder of this paper is organized as follows. In Section 2, we briefly review existing algorithms for GPUs. In Section 3, we present the proposed GPU algorithm in detail. In Section 4, we show some experimental results. In Section 5, we give some concluding remarks and future works. Due to the limited space, we illustrate neither the architecture nor the programming of GPUs in this paper. Readers unfamiliar with these are recommended the studies [4], [8], [13], [14], [15].

2. Related Works

As far as we know, except below, there is no existing research on multiple precision integer multiplication for GPUs.

In the study [2], the authors improved their result in the study [1], and implemented a method on CUDA such that Karatsuba method divides the whole multiplication into multiplications of smaller numbers which are performed by Strassen FFT multiplication. Their experiments show up to 4.29 times speedup with a single core of a 2.93 GHz Intel Core i7 870 and an NVIDIA GeForce GTX480 in integer multiplication of length 255 Kbits to 24.512 Mbits.

In the study [6], the authors reported 0.8 to 2.9 times speedup relative to CPU library mpFq [5] with SSE2 instructions in integer multiplication of length 160 to 384 bits for a 3 GHz Intel Core2 Duo E8400 and an NVIDIA GeForce 9800GX2.

In the study [21], under the assumption that many independent four arithmetic operations in multiple precision are given, the authors proposed a multiple precision arithmetic library which executes each operation with a single thread, and report about 4 times speedup relative to CPU library GNU MP [3] in 30720 integer multiplications of 2048 bits $\times$ 2048 bits with a single core of a 2.80 GHz Intel Core i7 and an NVIDIA GeForce GTX280.

In the study [18], the authors implemented the modular algorithm [9] which can execute in $O(n)$ time a multiplication of two numbers represented in the residue number system, while transformation of an n word integer from and to the residue number requires $O(n^2)$ time.

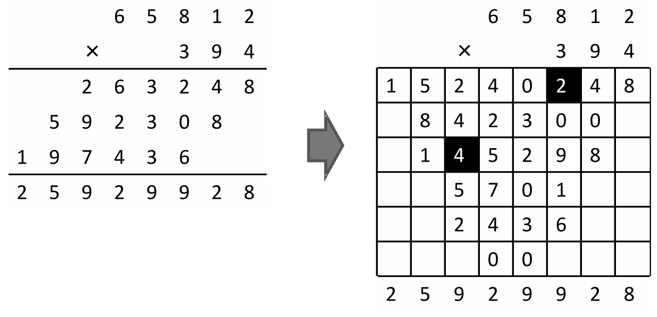
In the study [20], the authors reported about an optimization of multiple precision integer multiplication of 1024 bits $\times$ 1024 bits with Karatsuba method on CUDA.

In the study [12], the authors implemented a multiple precision multiplication with FFT on CUDA and reported 10 times speedup, but the used CPU and GPU are not shown.

3. The Proposed Method

3.1 A Product Digit Table

We can compute an integer multiplication by manual calculation as shown on the left side of Fig. 1. In the example, the base is 10. We consider computing the multiplication with a table as shown on the right side of Fig. 1. An element of the table is a quotient or remainder when dividing by the base a product of one digit of a multiplier and one digit of a multiplicand. Therefore, we can independently compute each element. For example, a white 2 on a black ground in the table is a remainder of division of 8 $\times$ 4 by base 10, and a white 4 on a black ground is a quotient of division of 5 $\times$ 9 by base 10. We can get the product by computing the summation column by column in the table and propagating the carries. We call such a table a *product digit table*. Fig. 2 shows a typical shape of a product digit table for a product of a multiple precision integer A of a digits and a B of b digits ($a \geq b$). Each element only in gray-scaled area A_1, A_2, or A_3 has a value. Fig. 2 implies that a product digit table has the following properties.



```
          6 5 8 1 2
        ×     3 9 4
      -------------
        2 6 3 2 4 8
      5 9 2 3 0 8
    1 9 7 4 3 6
    ---------------
    2 5 9 2 9 9 2 8
```

			6	5	8	1	2
		×			3	9	4
1	5	2	4	0	2	4	8
	8	4	2	3	0	0	
	1	4	5	2	9	8	
		5	7	0	1		
		2	4	3	6		
			0	0			
2	5	9	2	9	9	2	8

Fig. 1: Manual calculation of a multiplication and the corresponding product digit table

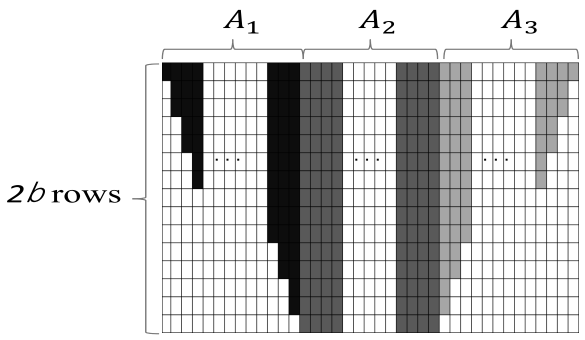


Fig. 2: Shape of the product digit table in case of a digits $\times$ b digits

- The number of the columns in $A_1(A_3)$ is equal to b.
- The number of the rows of a column in $A_1(A_3)$ is monotonically increasing (decreasing) two by two if the columns are seen from left to right.
- The number of the columns in A_2 equals $a - b$, and the number of the rows in A_2 is exactly $2b$.

When A and B in a base BASE are stored respectively in an array A[0.. a-1] and an array B[0.. b-1] (A[0] and B[0] are the least significant digits), the algorithm in Fig. 3 generates a product digit table of A $\times$ B on a 2D array T. Each element of T that is not assigned a value is don't care. Fig. 4 shows an example of a product digit table that is made up by the algorithm if A is a five digit number and B is a three digit number.

3.2 Data Structure

Each element of a product digit table is computed based on a product of one digit of A and one digit of B. We set BASE=2^{32} to execute the computation of the product as efficiently as possible. This setting makes every product fit in 64 bits (unsigned long long type on GPUs). Therefore, we can compute each quotient by 32 bits shift to the right for a product of length 64 bits and each remainder by computing one digit $\times$ one digit as in 32 bits (unsigned int type).

In the proposed GPU algorithm, we basically allocate a thread to each column of a product digit table so that both summation computation for each column and carry propagation can be done in parallel. However, if we directly use a product digit table in Fig. 2, the amount of computation and memory access would be imbalanced among threads allocated to the columns in A_1 or A_3. Hence, we represent a product digit table as a 2D array of size $a \times (2b)$ as shown in Fig. 5, rather than a 2D array of size $(a + b) \times (2b)$ as shown in Fig. 2. In the rest of this paper, for a product digit table in the load balanced form, R_1 is defined as the rectangle that corresponds to A_2 and R_2 is defined as the rectangle that corresponds to the two triangles A_1 and A_3 (See Fig. 2 and Fig. 5). Fig. 6 shows an example of a product digit table in the load balanced form. Notice that the

```
unsigned int T[b*2][a+b];
 /* make the area A1*/
for(j=0; j<b; j++){
  for(i=0; i<j; i++){ T[i*2][j]=A[j-i]*B[i]%BASE; T[i*2+1][j]=A[j-i-1]*B[i]/BASE;}
  T[j*2][j]=A[0]*B[j]%BASE;
}
 /* make the area A2*/
for(; j<a; j++) for(i=0; i<b; i++){ T[i*2][j]=A[j-i]*B[i]%BASE; T[i*2+1][j]=A[j-i-1]*B[i]/BASE;}
 /* make the area A3*/
for(; j<a+b; j++){
  T[0][j]=A[a-1]*B[j-a]/BASE;
  for(i=1; i<a+b-j; i++){ T[i*2-1][j]=A[a-i]*B[j-a+i]%BASE; T[i*2][j]=A[a-i-1]*B[j-a+i]/BASE;}
}
```

Fig. 3: An algorithm for constructing a product digit table

A[4] ×B[2] /BASE	*A[4] ×B[1] /BASE*	*A[4] ×B[0] /BASE*	*A[4] ×B[0] %BASE*	*A[3] ×B[0] %BASE*	*A[2] ×B[0] %BASE*	*A[1] ×B[0] %BASE*	*A[0] ×B[0] %BASE*
	A[4] ×B[2] %BASE	*A[4] ×B[1] %BASE*	*A[3] ×B[0] /BASE*	*A[2] ×B[0] /BASE*	*A[1] ×B[0] /BASE*	*A[0] ×B[0] /BASE*	
	A[3] ×B[2] %BASE	*A[3] ×B[1] /BASE*	*A[3] ×B[1] %BASE*	*A[2] ×B[1] %BASE*	*A[1] ×B[1] %BASE*	*A[0] ×B[1] %BASE*	
		A[3] ×B[2] %BASE	*A[2] ×B[1] /BASE*	*A[1] ×B[1] /BASE*	*A[0] ×B[1] /BASE*		
		A[2] ×B[2] /BASE	*A[2] ×B[2] %BASE*	*A[1] ×B[2] %BASE*	*A[0] ×B[2] %BASE*		
			A[1] ×B[2] /BASE	*A[0] ×B[2] /BASE*			

Fig. 4: An example of a product digit table generated by the algorithm in Fig. 3 (in case that A is a five digit number and B is a three digit number)

order of the elements in each column never affects the result of a multiplication in question. Hence, we can arbitrarily arrange the elements within their column. R_2 in Fig. 5 has the following properties.

- The horizontal length is b, and the vertical length is $2b$.
- A border line that splits R_2 into A_1 and A_3 lies between the 2α-th row and the $(2\alpha + 1)$-th row in the α-th column from the right.
- In the β-th row, B[$\beta/2$] is referred in all columns, and in the α-th column, A[α] is referred at the top row and the indices of the referred elements of A cyclically increase one by one every two rows.

3.3 Algorithm

We divide R_2 into tiles of width BLOCK_SIZE elements and height (BLOCK_SIZE $\times$ 2) elements as shown in Fig. 7. In the proposed algorithm, each thread block (block for short) has BLOCK_SIZE threads and is allocated to a tile so that each thread computes a summation of a column of the tile.

The tile can be categorized into three types: tiles with A_1 elements only, A_3 elements only, and the both elements. We respectively refer these types as T_1, T_3, and T_{13}. In a tile of the type T_{13}, a conditional branch is needed and therefore the execution efficiency is lower than that of the type T_1 or the type T_3. However, the slow down for a whole program is negligible because the number of tiles of T_{13} is much smaller than the total number of tiles of T_1 and tiles of T_3. In fact, the parallelism of a block is sustained in a tile of T_1 and a tile of T_3.

Each block accesses BLOCK_SIZE $\times$ 2 elements of A and BLOCK_SIZE elements of B. This enables each thread to compute the column summation only with values on shared memory. Device memory accesses to load the elements into shared memory can be coalesced, because each block refers contiguous elements of A and B.

3.4 Detail of Implementation

We illustrate what computation is to be performed for R_2. Regardless of the type of a tile, the first thing for a block to perform is loading necessary elements on device memory

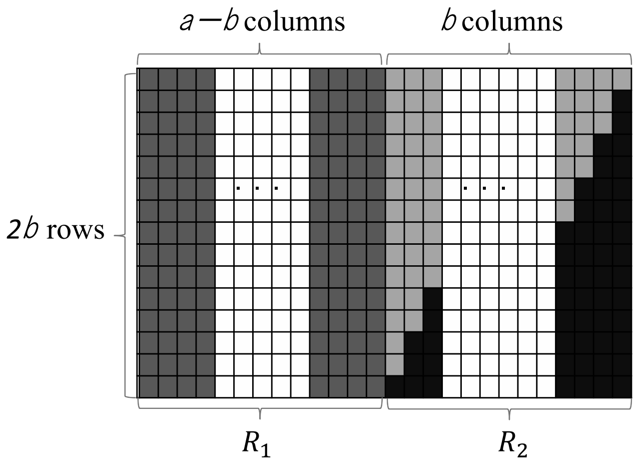


Fig. 5: A product digit table in load balanced form (general case)

into shared memory. Then, each thread of a block computes a summation of elements in the allocated column col. Each thread of a block that computes T_{13} separately computes two summations of A_1 elements and A_3 elements. Then, each thread adds the summation of A_1 elements to index col of an array C to store the answer, and adds the summation of A_3 elements to index $col+a$ of C. We call a CUDA C kernel function for rectangle R_2 mul_Bint_a.

Next, we explain a kernel function for rectangle R_1. This function is identical to the function for rectangle R_2 except that indices to refer A, B, and C are changed. We call a CUDA C kernel function for rectangle R_1 mul_Bint_b.

If the number of digits is enough large, almost all elements of C are likely to exceed the base 2^{32} at this stage with high probability. Thus, we perform carry propagation to make all elements smaller than the base. We divide this processing into two steps for efficiency. The first step is to separate each sum in C into the least significant digit and the carry. We call a CUDA C kernel function for the first step mul_Bint_c. Note that the used base 2^{32} is large enough to limit each carry at most one digit. Therefore, we can regard all the least significant digits form a multiple precision integer with base 2^{32}. Similarly, all the carries form a multiple precision integer. Hence, in the second step, we add these two multiple precision integers.

We explain the detailed implementation of the second step. In general, a multiple precision integer sum involves carry propagation, which is essentially sequential. In the worst case, carry propagates from the least significant digit to the most significant digit, which requires the computation time such that a single thread computes the whole sum. Our implementation is essentially the same as "carry skip adder" [10] that is one of implementations of a full adder in hardware. However, we devised our implementation as below so that addition is quickly done if only short propagations are occurred. Our implementation utilizes a fact that the possibility that at least one carry arises is very small because the base is 2^{32}. Fig. 8 illustrates the idea behind our implementation, but here the used base is 10 for simplicity.

At first, from given two arrays A and B we generate in parallel an array I to store carry information as shown in Fig. 8. The *carry information* is 1 if the sum of the corresponding two elements is larger than the base, 2 if it is equivalent to the base-1 (it is 9 in Fig. 8), or 0 if it is neither of the above two. Next, for every element whose carry information is 2, in parallel we change 2 to 1 (0) if its right neighbor is 1 (0). Notice that 2 such that its right neighbor is also 2 is not changed. We repeat this parallel processing until all 2s are disappeared. Then, we compute and store the sum total of three elements: the corresponding two elements of given two multiple precision integers and the final carry information of the two elements. Finally, we change each element of the array to a remainder of it divided by the base. The changed array is the multiple precision integer sum. This process needs barrier synchronization among blocks, so we achieve this by dividing the whole process into three kernel functions mul_Bint_d, mul_Bint_e and mul_Bint_f. These functions are executed only once in the order of mul_Bint_d, mul_Bint_e and mul_Bint_f.

4. Experiments

In this section, we compare the proposed method with NTL [17] and GMP [3], two representatives of existing multiple precision arithmetic libraries for a single CPU thread, and also with Emmart et al.'s FFT multiplication on GPUs [2] that is the fastest of existing studies on multiple precision integer multiplication on GPUs.

For each test, 3.10 GHz Intel Core i3-2100 and an NVIDIA GeForce GTX480 was used. Basically, we used 64 bit Windows7 Professional SP1 as an OS and Visual Studio 2008 Professional as a compiler, but only for GMP programs we used 64 bit Linux (ubuntu 11.04) as an OS and g++ 4.5.2 as a compiler because GMP does not officially support Windows. We used CUDA Ver 3.2 and display driver Ver 285.62. We used 48KB shared memory. The used versions of NTL and GMP are respectively 5.5.2 and 5.0.4. The NTL does not use SIMD instructions such as SSE instructions, but the GMP uses SSE2 instructions and MMX instructions. Both NTL and GMP are single-threaded libraries. Multi-threaded versions of them do not exist yet. Parallelizing them seems to be hard work. Therefore, the CPU programs use a single core only.

At first, we show a comparison between the proposed method and NTL. Table 1 (Table 2) summarizes execution times of a multiplication with the proposed method (NTL) for two multiple precision integers of length 8 Kbits to 256 Kbits. Table 3 shows the corresponding speedup ratios. These tables indicate that the proposed method is faster than NTL in all cases and the maximum speedup is over 70 times.

A[4] ×B[0]%BASE	*A[3] ×B[0]%BASE*	*A[2] ×B[0]%BASE*	*A[1] ×B[0]%BASE*	*A[0] ×B[0]%BASE*
A[3] ×B[0]/BASE	*A[2] ×B[0]/BASE*	*A[1] ×B[0]/BASE*	*A[0] ×B[0]/BASE*	*A[4] ×B[0]/BASE*
A[3] ×B[1]%BASE	*A[2] ×B[1]%BASE*	*A[1] ×B[1]%BASE*	*A[0] ×B[1]%BASE*	*A[4] ×B[1]%BASE*
A[2] ×B[1]/BASE	*A[1] ×B[1]/BASE*	*A[0] ×B[1]/BASE*	*A[4] ×B[1]/BASE*	*A[3] ×B[1]/BASE*
A[2] ×B[2]%BASE	*A[1] ×B[2]%BASE*	*A[0] ×B[2]%BASE*	*A[4] ×B[2]%BASE*	*A[3] ×B[2]%BASE*
A[1] ×B[2]/BASE	*A[0] ×B[2]/BASE*	*A[4] ×B[2]/BASE*	*A[3] ×B[2]%BASE*	*A[2] ×B[2]/BASE*

Fig. 6: An example of a product digit table in the load balanced form (reconstructed from the table in Fig. 4)

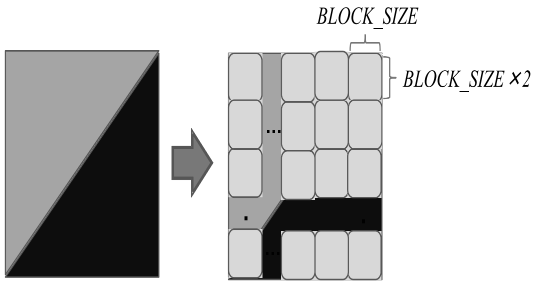


Fig. 7: Thread block allocation for a product digit table in the balanced form

Next, we show a comparison with GMP that is known to be faster than NTL in general. Table 4 summarizes execution times of a multiplication for the same multiple precision integers as the integers in Table 1 and Table 2. Table 5 shows the corresponding speedup ratios. Although these results are inferior to the results for NTL, we can see over 10 times speedup.

In addition, we conducted a speed comparison with Emmart et al.'s FFT multiplication on GPUs. Emmart et al. reported 255 Kbits × 255 Kbits with GTX480 takes 0.207 msec. Under the almost same condition of 256 Kbits × 256 Kbits, the proposed method takes 0.5922 msec as shown in Table 1. Hence, if bit lengths of given two multiple precision integers are the same, Emmart et al.'s FFT multiplication is about three times faster than the proposed method. However, if bit lengths of given two multiple precision integers are different, the proposed method is faster than Emmart et al.'s FFT multiplication. For example, consider the case of 256 Kbits × 8 Kbits. As noted in Section 1, Emmart et al.'s implementation is based on FFT of 383 Kbits. Thus, values smaller than 383 Kbits must be promoted to 383 Kbits to be multiplied as 383 Kbits × 383 Kbits. In fact, they reported 383 Kbits × 383 Kbits with GTX480 takes 0.200 msec, but 255 Kbits × 255 Kbits takes 0.207 msec due to additional time for promotion. So in their implementation, the speed in 256 Kbits × 8 Kbits is at least the speed in 383 Kbits × 383 Kbits. In contrast to this, as shown in Table 1, the execution time of the proposed algorithm is shortened if either a multiplier or a multiplicand is smaller than the other.

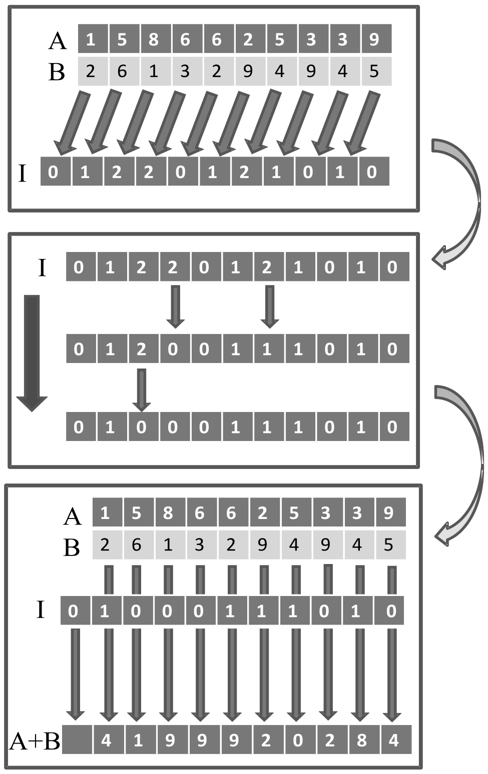


Fig. 8: An algorithm for computing a multiple precision integer sum

For example, the execution time of a multiplication of 256 Kbits × 8 Kbits is about 13 times faster than a multiplication of 256 Kbits × 256 Kbits. In summary, in case of 256 Kbits × 8 Kbits, the proposed method is about 4.57 times faster than Emmart et al.'s GPU implementation (0.207 msec / 0.0453 msec).

5. Conclusion

We have proposed a novel data structure named a product digit table, and we have presented an algorithm that fast executes a multiple precision integer multiplication on GPUs based on the product digit table. The proposed method is based on manual calculation, so in case of a multiple

Table 1: Execution times of A × B with the proposed algorithm on a GPU (msec) (Data transfer time between a GPU and a CPU is not included)

B \ A	8Kbits	16Kbits	32Kbits	64Kbits	128Kbits	256Kbits
8Kbits	0.0211	0.0290	0.0292	0.0311	0.0364	0.0453
16Kbits	0.0290	0.0212	0.0301	0.0334	0.0426	0.0607
32Kbits	0.0292	0.0301	0.0248	0.0395	0.0570	0.0934
64Kbits	0.0311	0.0334	0.0395	0.0530	0.0946	0.1671
128Kbits	0.0364	0.0426	0.0570	0.0946	0.1610	0.3110
256Kbits	0.0453	0.0607	0.0934	0.1671	0.3110	0.5922

Table 2: Execution times of A×B with NTL library on a CPU (msec)

B \ A	8Kbits	16Kbits	32Kbits	64Kbits	128Kbits	256Kbits
8Kbits	0.094	0.179	0.357	0.695	1.388	2.799
16Kbits	0.179	0.273	0.560	1.031	2.049	4.059
32Kbits	0.357	0.560	0.777	1.574	3.068	6.666
64Kbits	0.695	1.031	1.574	2.464	4.590	9.455
128Kbits	1.388	2.049	3.068	4.590	6.989	13.972
256Kbits	2.799	4.059	6.666	9.455	13.972	20.766

Table 3: Speedup ratios of the proposed algorithm to NTL

B \ A	8Kbits	16Kbits	32Kbits	64Kbits	128Kbits	256Kbits
8Kbits	4.5	6.2	12.2	22.3	38.1	61.8
16Kbits	6.2	12.9	18.6	30.9	48.1	66.9
32Kbits	12.2	18.6	31.3	39.8	53.8	71.4
64Kbits	22.3	30.9	39.8	46.5	48.5	56.6
128Kbits	38.1	48.1	53.8	48.5	43.4	44.9
256Kbits	61.8	66.9	71.4	56.6	44.9	35.1

Table 4: Execution times of A×B with GMP on a CPU (msec)

B \ A	8Kbits	16Kbits	32Kbits	64Kbits	128Kbits	256Kbits
8Kbits	0.035	0.053	0.089	0.158	0.284	0.552
16Kbits	0.053	0.074	0.117	0.215	0.396	0.779
32Kbits	0.089	0.117	0.165	0.286	0.543	1.072
64Kbits	0.158	0.215	0.286	0.385	0.748	1.486
128Kbits	0.284	0.396	0.543	0.748	0.997	1.912
256Kbits	0.552	0.779	1.072	1.486	1.912	2.602

precision integer multiplication of the same bit length, our algorithm runs slower than FFT multiplication. However, if bit lengths of given two multiple precision integers are different and their bit numbers are not extremely large, our algorithm is better than FFT multiplication.

Future works include further optimization of our implementation, in particular for integers of length a few thousand bits intended for the real problem including public-key cryptography.

References

[1] Emmart, N. and Weems, C., "High Precision Integer Addition, Subtraction and Multiplication with a Graphics Processing Unit," Parallel Processing Letters, Vol.20, No.4, pp.293-306, 2010.

[2] Emmart, N. and Weems, C., "High Precision Integer Multiplication with a GPU," IEEE Int'l Parallel & Distributed Processing Symposium (IPDPS), pp.1781-1787, May 2011.

Table 5: Speedup ratios of the proposed algorithm to GMP

B \ A	8Kbits	16Kbits	32Kbits	64Kbits	128Kbits	256Kbits
8Kbits	1.7	1.8	3.0	5.1	7.8	12.2
16Kbits	1.8	3.5	3.9	6.4	9.3	12.8
32Kbits	3.0	3.9	6.7	7.2	9.5	11.5
64Kbits	5.1	6.4	7.2	7.3	7.9	8.9
128Kbits	7.8	9.3	9.5	7.9	6.2	6.1
256Kbits	12.2	12.8	11.5	8.9	6.1	4.4

[3] Free Software Foundation, "The GNU Multiple Precision Arithmetic Library, ",
http://gmplib.org/ , 2011.

[4] Garland, M. and Kirk, D. B., "Understanding Throughput-Oriented Architectures," Communications of the ACM, Vol.53, No.11, pp.58-66, 2010.

[5] Gaudry, P. and ThomÃş, E., "The mpFq Library and Implementing Curve-based Key Exchanges," Software Performance Enhancement for Encryption and Decryption Workshop, pp. 49âĂŞ64, 2007.

[6] Giorgi, P., Izard, T., and Tisserand, A., "Comparison of Modular Arithmetic Algorithms on GPUs," Int'l Conf. on Parallel Computing (ParCo),
http://hal-lirmm.ccsd.cnrs.fr/docs/00/43/06/89/PDF/article-parco09.pdf , Sept. 2009.

[7] Karatsuba, A. and Ofman, Y., "Multiplication of Multidigit Numbers on Automata," Doklady Akademii Nauk SSSR, Vol.145, No.2, pp.293-294, 1962. (in Russian). English translation in Soviet Physics-Doklady, Vol.7, pp.595-596, 1963.

[8] Kirk, D. B. and Hwu, W. W., "Programming Massively Parallel Processors: A Hands-on Approach," Morgan Kaufmann, 2010.

[9] Knuth, D. E., "The Art of Computer Programming," Vol.2, Seminumerical Algorithms, 3rd Edition, Addison-Wesley, 1997.

[10] Lehman, M. and Burla, N., "Skip Techniques for High-Speed Carry Propagation in Binary Arithmetic Units". IRE Trans. Elec. Comput., Vol.EC-10, pp.691-698, 1961.

[11] Lindholm, E., Nickolls, J., Oberman, S., and Montrym, J., "NVIDIA Tesla: A Unified Graphics and Computing Architecture,"

[12] Liu, H. J. and Tong, C., "GMP Implementation on CUDA : A Backward Compatible Design with Performance Tuning,"

[13] NVIDIA, "CUDA Programming Guide Version 5.5.,"
http://www.nvidia.com/object/cuda_develop.html , 2013.

[14] NVIDIA, "CUDA Best Practice Guide 5.5. ,"
http://www.nvidia.com/object/cuda_develop.html , 2013.

[15] Sanders, J. and Kandrot, E., "CUDA by Example: An Introduction to General-Purpose GPU Programming," Addison-Wesley Professional, 2010.

[16] SchÃűnhage, A. and Strassen, V., "Schnelle Multiplikation grosser Zahlen," Computing, Vol.7, pp.281-292, 1971.

[17] Shoup, V., "NTL: A Library for doing Number Theory,"
http://www.shoup.net/ntl/, 2009.

[18] Tanaka, T. and Murao, H., "An Efficient Method for Multiple-Precision Integer Arithmetics Using GPU, " Information Processing Society of Japan SIG Technical Report, Vol.2010-HPC-124, No.2, pp.1-7, 2010 (in Japanese).

[19] Toom, A. L., "The Complexity of a Scheme of Functional Elements Realizing the Multiplication of Integers," Soviet Math., Vol. 3, pp.714-716, 1963.

[20] Zhao, K., "Implementation of Multiple-precision Modular Multiplication on GPU",
http://www.comp.hkbu.edu.hk/ pgday/2009/10th_papers/kzhao.pdf , 2009.

[21] Zhao, K. and Chu, X., "GPUMP: A Multiple-Precision Integer Library for GPUs," IEEE Int'l Conf. on Computer and Information Technology (CIT), pp.1164-1168, June 2010.

[22] Zuras, D., "More on Squaring and Multiplying Large Integer," IEEE Trans. Computers, Vol.43, No 8, pp.899-908, Aug. 1994.

Combining the Phoenix Flash Code with the Binary Index Flash Code for Low Write Deficiency

G. N. Corneby[1], L. K. Sanchez[1], P. Fernandez[1], M. J. Tan[2], and K. Kaji[2]
[1]Department of Information Systems and Computer Science, Ateneo de Manila University, Philippines
[2]Graduate School of Information Science, Nara Institute of Science and Technology, Japan

Abstract—*In the framework of floating codes, a block of flash cells stores data in the form of binary numbers. The fundamental approach in constructing a coding scheme is by assigning cells to bits. However, the way to assign cells to bits is not simple, as the frequency of changes of the value of the bits is not known. This makes it difficult to partition the cells in a block in proportion to the frequency of the changes of bit value where the most updated bit has the most number of cells assigned to it. In this study, we discuss a novel coding scheme to dynamically assign cells to bits. At the beginning, there is a pre-determined assignment of cells to bits, but the coding scheme allows reassignment of cells if needed. The proposed coding scheme gives a very low write deficiency. A novel idea is discussed in this manuscript.*

Keywords: flash code, flash memory, phoenix flash code, binary-indexed, absorb, revive

1. Introduction

Flash memory devices are currently constrained by the write asymmetry property. It is easy to increase the charge in one flash cell, i.e., perform a *cell write*, but decreasing a charge in a cell is not possible except by emptying the charges simultaneously in all cells of a block. This operation is referred to as a *block erasure*.

A block erasure is not only time-consuming but also causes some damage to the device. It has been estimated that a block of cells can only accommodate about 10^4 to 10^5 block erasures before it becomes unreliable [6]. It is therefore desirable to delay block erasures as much as possible, by designing good coding schemes, in order to extend the lifespan of flash memory devices.

A *flash code* is used for decoding and encoding digital information in a flash memory. The performance of a flash code is normally evaluated by measuring its *write deficiency*. This is computed by taking the difference between the maximum possible and the actual number of cell writes. A lower write deficiency is clearly preferred.

One of the most popular flash codes in literature is the Index-less Indexed Flash Code (ILIFC). This flash code partitions a block of cells into sub-blocks, called *slices*. Each ILIFC slice has exactly k cells, where k is the number of bits of the data represented by a block. Encoding is designed so that it is possible to infer both the bit index (that a slice represents) and the bit-value by just reading the cell-values within the given slice [5].

Binary Index Flash Code (BIFC) introduced the partitioning of a flash code into smaller slices. Unlike ILIFC which uses k cells per slice, BIFC uses slices of size $s = O(\log k)$. The drawback in BIFC is that there is an overhead write deficiency of $s - 2$ for every slice. Generally, however, the BIFC flash code has a better write deficiency than the ILIFC when k is sufficiently large [9].

More recently, the Dual Mode Flash Code (DMFC) was introduced, combining the BIFC with a Simple Segmentation coding scheme. Experimental results using DMFC show that it has a significantly lower write deficiency than any of the previously designed flash codes in the average case [11].

In this study, we improve the DMFC further by using Phoenix Flash Code (PFC), a recently developed coding scheme that allows reassignment of cells to bits.[2] In the original version of PFC, the reassignment of cells contains a slight overhead cost which contribute to $O(n)$ to the write deficiency. We introduce a modified version of the reassignment to remove such overhead. The detailed discussion of PFC is in Sect. 3. Afterwards, we combine PFC, with the modified reassignment, with the BIFC. We refer to this latter flash code as PFCB and will be discussed in Sect. 4. Computer simulations show that this flash code is superior, even when compared to the DMFC and this is shown in Sect. 5.

2. Preliminaries

A *block* of flash memory is a sequence of n cells. Each cell stores an integer value from $A_q = \{0, \ldots, q-1\}$, and this value is referred to as the *cell-level*. A cell has three type states. A cell with a value of 0 or $q-1$ is said to be *empty* or *full*, respectively. If a cell is neither empty nor full, then that cell is said to be *active*. All cells within a block are ordered. The value of the i-th cell is denoted by c_i where $0 \leq i < n$. A tuple $(c_0, \ldots, c_{n-1}) \in A_q^n$ represents a possible state of a block. For two states $C = (c_0, \ldots, c_{n-1})$ and $C' = (c'_0, \ldots, c'_{n-1})$, we write $C \preceq C'$ if $c_i \preceq c'_i$ for all $0 \leq i < n$, and $C \prec C'$ if $C \preceq C'$ and $C \neq C'$. A state can transit from C to C' if and only if $C \prec C'$, as state transition is accomplished through cell writes. Similar to individual cells, a block has three type of states. A block is *empty* if all cells within the block are empty. A block is *full* if all the cells within a block are full. Otherwise, the block is *active*. The notion of "states", "$\prec$" and "type of states" are extended to subsets of cells in a natural manner [3], [10]. A block of flash memory cells stores a k-bit data $D = (d_0, \ldots, d_{k-1})$. The data D is updated through a *write operation* which flips the value of a single bit in D.

A flash code $F = (\mathcal{E}, \mathcal{D})$ contains two functions which are used to update and retrieve the data stored within a block. The decode function $\mathcal{D} : A_q^n \to (d_0, \ldots, d_{k-1})$ retrieves the value of the data stored in the block of flash memory cells. The encode function $\mathcal{E} : \{0, \ldots, k-1\} \times A_q^n \to A_q^n \cup \{E\}$ is applied to the block for every write operation. The encode function first attempts to accommodate the write operation by applying cell writes to the block following a pre-specified procedure. If successful, this operation produces a new block state $C' = \mathcal{E}(i, C)$ where $\mathcal{D}(C)$ and $\mathcal{D}(C')$ only differ in the i-th bit. Otherwise, the encode function returns a block erasure E.

Since a block has n cells and each cell has a maximum of $q - 1$ levels, then in the ideal case a flash code can accommodate each write operation with one cell write. Hence, the maximum number of write operations by the ideal flash code is $n(q-1)$. This expression is used in a metric for the performance of flash codes. Specifically, the *write deficiency* of a flash code F, denoted by $\delta(F)$, is computed using the formula $\delta(F) = n(q-1) - t$, where t is the actual number of write operations accommodated by the flash code F. A write deficiency of zero is the ideal case.

3. Phoenix Flash Code

Phoenix Flash Code (PFC) is a very recently developed flash code. Its encoding process is somewhat similar to *stacked segment encoding (SS encoding)*[11]. This is especially true when there is an equal distribution of write operations among the k bits of data. PFC starts with dividing the block into smaller groups called *segments*. A segment $S_i = (c_{i,0}, \ldots, c_{i,k-1})$ where $c_{i,j} = c_{ik+j}$, $0 \leq i < \frac{n}{k}$. Each segment has k cells and is cyclic in the context of the cell adjacency. This implies that the left adjacent cell of $c_{i,0}$ is $c_{i,k-1}$ and the reverse also holds true. For each active segment S_i, cell $c_{i,j}$ is initially assigned to d_j where $0 \leq j < k$. The bit-value of d_j is computed using the parity of the cell assigned to it. An example is shown in Fig. **??** using segment S_1.

Following the encoding process of the SS encoding, it can be observed that the assigning of cells to bits is not flexible enough to accommodate non-uniform distribution of write operations. PFC solves this problem by incorporating two operations, called *absorption* and *revival*. These two operations are used only when one cell is about to become full in the current write operation. In all the other scenarios, PFC acts similar to SS encoding.

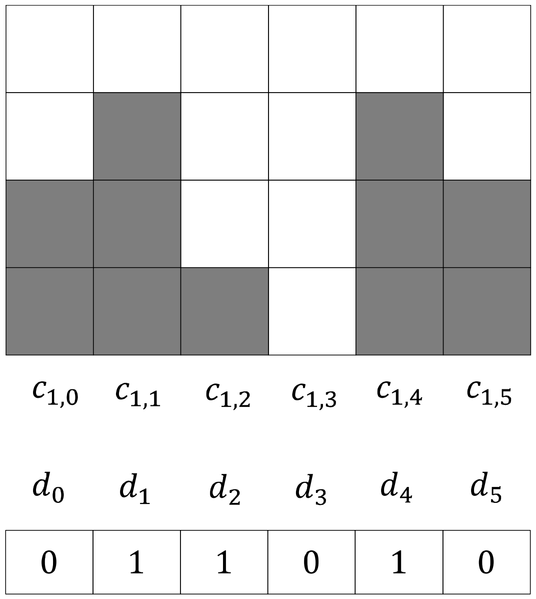


Fig. 1: Mapping a PFC segment to data bit-values.

The first operation, absorption, is used to allow some cell to take over the adjacent cell within a segment. Originally, cell $c_{i,j}$ in segment S_i is assigned to d_j. If the absorption operation is applied to $c_{i,j}$, then the right-adjacent cell $c_{i,j'}$, where $j' = (j+1) \bmod k$, is reassigned from $d_{j'}$ to d_j. We can view the result of the operation as $d_{j'}$ has either been reassigned to a cell in the nearest available segment S_{i+x}(x is some positive integer) or has become unassigned. Both are acceptable if we assume that an unassigned bit has a value of 0. Consider the segment in Fig. 1. Observe that in the next write operation for d_4, the encoding process will increase the cell-value of $c_{1,4}$ which makes the cell full. This will invoke the absorption operation and reassign $c_{1,5}$ to d_4 as shown in Fig. 2.

The absorption operation is incomplete on its own. Although we cannot observe any error in the previous example, this is only because cell $c_{i,5}$, referred to as the adsorbate, has an even cell-value. This causes no problem because absorbing a cell with even parity does not affect the value of the bit assigned to the absorber. In the case when the adsorbate has an odd parity, the absorption operation by itself will cause problems with data integrity. To remedy this conundrum, we also invoke the revival operation. This operation is performed to preserve data integrity by applying a cell write to "revive" the bit previously assigned to the adsorbate.

We will discuss two approaches for the revival operation. We will follow the same symbols and notation as used in the discussion of the absorption operation. The first is a simple and straightforward approach and does not require any change to the absorption operation. In the event when the revival operation is invoked, the operation simply increases

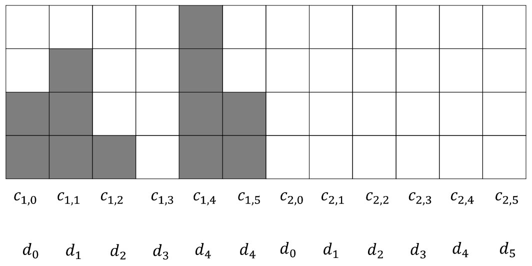

$c_{1,0}$ $c_{1,1}$ $c_{1,2}$ $c_{1,3}$ $c_{1,4}$ $c_{1,5}$ $c_{2,0}$ $c_{2,1}$ $c_{2,2}$ $c_{2,3}$ $c_{2,4}$ $c_{2,5}$
d_0 d_1 d_2 d_3 d_4 d_4 d_0 d_1 d_2 d_3 d_4 d_5

Fig. 2: Updating d_4 causes the (even parity) d_5 to be absorbed.

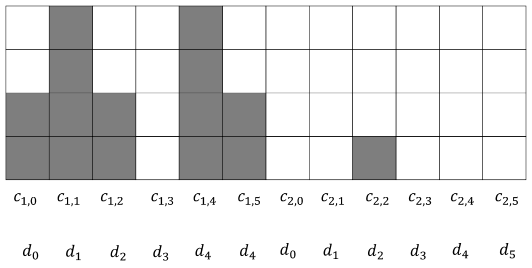

$c_{1,0}$ $c_{1,1}$ $c_{1,2}$ $c_{1,3}$ $c_{1,4}$ $c_{1,5}$ $c_{2,0}$ $c_{2,1}$ $c_{2,2}$ $c_{2,3}$ $c_{2,4}$ $c_{2,5}$
d_0 d_1 d_2 d_3 d_4 d_4 d_0 d_1 d_2 d_3 d_4 d_5

Fig. 3: Applying the first approach of the revival operation to d_2.

the cell-value of $c_{i,j'}$ and $c_{i+x,j'}$ by 1 each. Take note that cell $c_{i+x,j'}$ of segment S_{i+x} is the cell to which $d_{j'}$ is newly assigned. This approach is the same as the one discussed in [2]. An example can be seen in Fig. 3.

We can observe that if a segment needs only one more cell write to make it full, then only one bit is assigned to all of the cells of that segment. We can further observe that in a full segment, the bit assigned to the cells of the full segment has a bit-value of 0, assuming k is even. This implies that from the perspective of the decoding function, we can ignore full segments as they have zero significance to the value of the data. The same goes for empty segments. Thus, we will only focus on active segments. Furthermore, a bit can only be assigned in one segment at a time. The core of the decoding function is to determine which cells are absorbed and which are not. This will enable us to know to which bit the cells are assigned. With the descriptions of the absorption and revival operations mentioned above, we can state that a cell is *independent*, i.e. not absorbed, if its left adjacent cell is not full. Once assignment of cells to bits is known, computing the bit-value is simply done by computing the parity of the appropriate cell-values.

There are three factors to the write deficiency. The first is the unused cells that were not enough to form one segment. There will be exactly $n \bmod k$ such cells, which is at most $k-1$. The second factor is the extra two cell writes used for each revival operation. In a segment, we can apply at most $k-1$ revival operations. Overall, this contributes to at most $O(n)$ to the write deficiency. The third factor is from

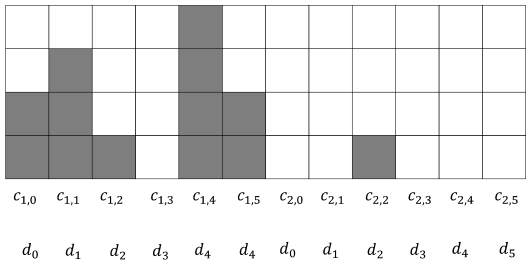

$c_{1,0}$ $c_{1,1}$ $c_{1,2}$ $c_{1,3}$ $c_{1,4}$ $c_{1,5}$ $c_{2,0}$ $c_{2,1}$ $c_{2,2}$ $c_{2,3}$ $c_{2,4}$ $c_{2,5}$
d_0 d_1 d_2 d_3 d_4 d_4 d_0 d_1 d_2 d_3 d_4 d_5

Fig. 4: Applying the second approach of the revival operation to d_2.

the active segments left upon block erasure. There can be at most $k-1$ active segments. A loose upper bound to its contribution to write deficiency is $O(k^2q)$. Hence the total write deficiency is at most $O(n+k^2q+kq) = O(n+k^2q)$.

As we can see from the above discussion, the first approach for the revival operation comes with an overhead cost that is at most $O(n)$. Similar with BIFC, this is an issue when n is significantly larger than k and q. The second approach was designed to remove this overhead. Some modifications on the absorption operation and decode procedure are needed to successfully replace the previous version of the revival operation.

As mentioned before, the revival operation is only invoked when the adsorbate cell, i.e., $c_{i,j'}$ as stated in the previous discussion, has an odd parity. The second approach starts with delaying the cell write to $c_{i,j}$, the absorber cell, that would have rendered it full. Instead, it directly applies a cell write to $c_{i+x,j'}$. In doing so, it transfers the data information of $d_{j'}$ from $c_{i,j'}$ to $c_{i+x,j'}$ and causes $c_{i,j'}$ to be implicitly absorbed by $c_{i,j}$. This saves the flash code from applying the extra two cell writes as stated in the first approach. Hence removing the overhead cost. The second approach of the revival operation is illustrated in Fig. 4.

The second approach of the revival operation requires some modification to the decoding procedure. Specifically, the way to determine if a cell is either independent or absorbed. It is not enough to merely check if the adjacent cell to the right is full because with the second approach, there is now a possibility to have cells that are absorbed that is adjacent to non-full cell on the former's right. However, a cell can only still be assigned to one segment at a time. Let the set $T = \{S_{a_1}, \ldots, S_{a_r}\}$ be the set of active segments in state of the block C. We first process the rightmost segment S_{a_r} using the same rule to determine between absorbed and independent cells as the one in the first approach. This is allowed because no revival operation has been applied to S_{a_r} otherwise there should exist an active segment $S_{a_{r+1}}$, which in turn negates that S_r is the rightmost active segment in block C.

Starting from $S_{a_{r-1}}$ to S_{a_1}, we still follow the same rule to determine between absorbed and independent cells, except

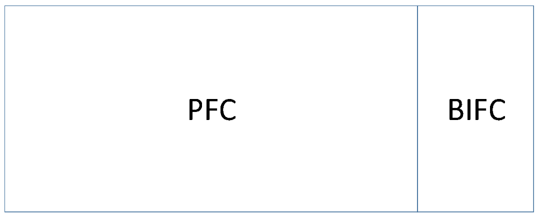


Fig. 5: Visual representation of PFCB and BIFC partitions.

we amend one more rule. For every active cell $c_{a_i,j}$ in segment S_{a_i} that is labelled as independent by the previous rule, the amended rule dictates that if d_j is already assigned to a cell or cells in S_{a_x}, $x > i$, then $c_{a_i,j}$ is labelled as absorbed. Afterwards, we can proceed similarly as the decoding procedure in the first approach and compute of the bit-value of each bit of the data.

By removing the overhead cost of using the revival operation, we are able to reduce the upper bound write deficiency of PFC from $O(n + k^2q)$ to $O(k^2q)$. This performance is comparable to the first phase encoding of ILIFC [5].

4. Combining PFC and BIFC

In the previous section, the implementation of the second approach removed one of the factors of write deficiency of PFC. In this section, we will discuss a method to diminish the effect of the first factor, the cells that was never used as a segment, to the write deficiency. In our modified flash code we combine PFC and BIFC [11], and refer to this as the Phoenix Flash Code with BIFC (PFCB). We first divide the block into two major partitions. In the left partition, we fit as many PFC segments as possible, i.e., $m = \lfloor n/k \rfloor$ segments. The (possibly empty) right partition is then allocated for the BIFC slices. See Fig. 5 for this. This implies that BIFC only uses the cells which are not enough to form a segment, thus addressing the write deficiency due to the $n \mod k$ cells in the original Phoenix Flash Code. Note that if there are further remainder cells that cannot form a BIFC slice, these cells are incorporated in the PFC partition.

Whenever it is possible to update some i-th bit using the PFC partition, the appropriate PFC segment(s) is updated. Otherwise an appropriate BIFC slice is updated. Only when both options are not possible does a block erasure occur. When n is significantly larger than k, the portion for BIFC may seem insignificant in relation to the entire block. However, there are still more than enough cells to form many slices to continue the encoding process than it would have been without the BIFC partition.

5. Results

We simulated the performance of different flash codes and compared the resulting write deficiencies. In the simulations, the block size was fixed at 2048 cells and the maximum charge of a cell was set to 7. The bit size k was varied, and

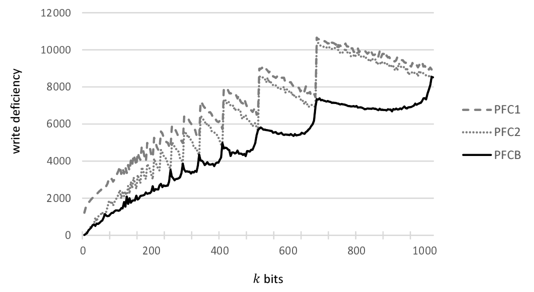


Fig. 6: Simulation results using uniform distribution.

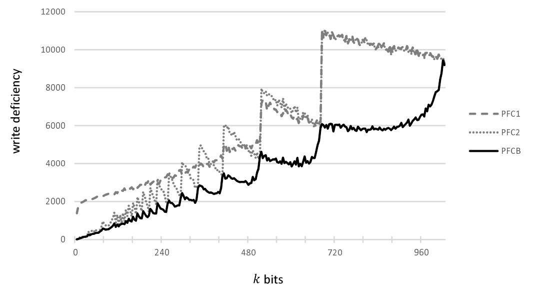


Fig. 7: Simulation results using one bit dominating 50% of the updates.

for each k value, 30 experiments were run, and the average write deficiencies were reported.

We tested the performance of the flash codes in two different distributions. In the uniform distribution, each bit has an equal probability of $1/k$ of being updated. We first investigate the improvements brought about by modifying the revival operation of the original PFC to incorporate the second approach and also the improvements brought about by incorporating the BIFC partition. Fig. 6 and 7 clearly illustrate the significant improvements to the original PFC. We then compare the PFCB with other flash codes and the results shown in Fig. 8 indicates its superiority, even against the previous best DMFC. As a side note, the write deficiency graphs of ILIFC and BIFC each shows a sharp increase at some k value. This can be attributed to the fact that at the specific k value, the number of bits in the data is larger than the maximum number of possible slices that may be assigned. Thus, there is at least one bit which cannot be assigned to a slice (after the slices have been assigned to the other bits), and this causes a block erasure to be called upon first update for that bit.

We also test the performance of the PFCB against other flash codes in a non-uniform distribution. Here 1 bit has 0.50 probability of being updated, while the rest of the other bits have a uniform $0.5/(k - 1)$ probability of being updated. In this distribution, the proposed flash code is even more significantly superior as shown in Fig. 9.

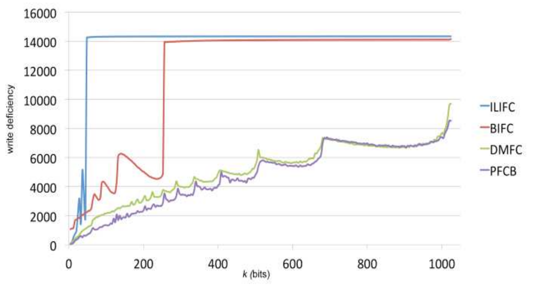


Fig. 8: Simulation results using uniform distribution.

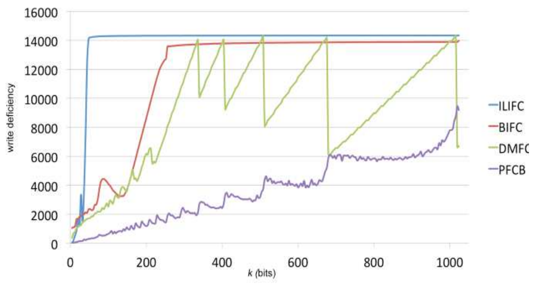


Fig. 9: Simulation results using one bit dominating 50% of the updates.

6. Conclusion

In this study, we propose a new flash code that combines the Phoenix Flash Code (PFC) and the Binary Indexed Flash Code (BIFC). Simulation results indicate that it has a significantly better write deficiency than existing flash codes in literature. Generally, a better write deficiency leads to a longer lifespan for flash memory devices.

It would be interesting to investigate other techniques for lowering the write deficiency of flash codes, by perhaps developing new operations. Future studies can also explore other ways of combining two or more flash codes, and evaluating the resulting performance.

References

[1] V. Bohossian, A. Jiang, and J. Bruck, “Buffer coding for asymmetric multi-level memory,” in *Proc. ISIT’07*, 2007 , pp. 1186–1190.

[2] G. Corneby, L. Sanchez, M. J. Tan, P. Fernandez, and Y. Kaji, “Phoenix flash code: introducing the absorption and revival operations for reducing flash memory write deficiency,” in *Proc. (NCITE 2013)*, 2013, pp. 209–215.

[3] A. Jiang, V. Bohossian, and J. Bruck, “Rewriting codes for joint information storage in flash memories,” *IEEE Transactions on Information Theory*, vol. 56, pp.5300–5313, Oct. 2010.

[4] Y. Kaji, “The expected write deficiency of index-less flash codes and their improvement,” *IEICE Trans. on Fundamentals*, vol. E95-A, no. 12, pp. 2130–2138, Dec. 2012.

[5] H. Mahdavifar, P. Siegel, A. Vardy, J. Wolf, and E. Yaakobi, “A nearly optimal construction of flash codes,” in *Proc. ISIT’09*, 2009, pp. 1239–1243.

[6] A. Olson, and D. Langlois, “Solid state drives data reliability and lifetime,” In Imation Corporation White Pater, 2008.

[7] R. Rivest, and A. Shamir, “How to reuse a “write-once” memory,” Information and Control, vol. 55, pp.1–19, 1982.

[8] R. Suzuki, and T. Wadayama. “Layered index-less indexed flash codes for improving average performance,” in *Proc. ISIT’11*, 2011, pp. 2138–2142.

[9] M. J. Tan, and Y. Kaji, “Uniform-compartment flash code and binary-indexed flash code,” in *IEICE Technical Report*, 2012, pp. 25–30.

[10] M. J. Tan, and Y. Kaji, “Flash code utilizing binary-indexed slice encoding and resizable-clusters,” *IEICE Trans. on Fundamentals*, vol. E96-A, no. 12, pp. 2360–2367, Dec. 2013.

[11] M. J. Tan, P. Fernandez, N. Salazar, J. Ty, and Y. Kaji, “Flash code with dual modes of encoding,”. in *Pre-Proc. WCTP’13*, 2013.

A CIL Virtual Machine for Wireless Sensor Network Applications

Yutaka Yanagisawa[1], **Yasue Kishino**[1], **Takayuki Suyama**[2], **Tsutomu Terada**[3]
Masahiko Tsukamoto[2], **and Futoshi Naya**[3]
[1]NTT Communication Science laboratories, NTT Corporation
[2]Cognitive Mechanisms Laboratories, Advanced Telecommunications Research Institute International
[3]Guraduate School of Engineering, Kobe University

Abstract— *This paper describes CILIX, a compact and powerful implementation of a CIL virtual machine working on resource-poor wireless sensor nodes. CILIX can process CIL programs on a device that has such limited computational resources as an 8-bit/16-bit CPU, 32-KB program memory, and 4-KB RAM. It provides many useful functions for a sensor node, including an I/O manager with UDP, FAT 32, thread control, and dynamic program replacement. For developing software on sensor nodes using CILIX, developers can chose programming languages from C#, C++/CLI, Visual Basic, J++, F#, and the many other languages supported by the .NET Framework.*

1. Introduction

A process virtual machine, which enables portability by abstracting each device and operating system, also presents a standard programming interface across a range of target platforms [1]. This mechanism reduces the cost of developing a program that works on various sensor devices with different platforms. Currently, Java Virtual Machine (JVM) and Virtual Execution System (VES) for Common Intermediate Language (CIL) are the two most popular process virtual machines used on personal computers. In this paper, we denote an implementation of VES as CIL-VM. Several JVMs have been implemented on resource-poor sensor devices. For example, SimpleRTJ [2] and Darjeeling [3] provide the execution environment for Java code on small sensor devices that have an 8-bit/16-bit CPU, 2- to 4-KB RAM, and 32- to 128-KB program memory. Due to these existing JVMs, we can develop software for small sensor nodes in the powerful Java development environment. JVM only supports Java, so developers cannot choose any other programming languages.

On the other hand, CIL-VM can execute programs developed in various programming languages, for instance, J++, C#, Visual Basic, and C++/CLI, F#. In other words, a developer can choose her favorite language in which to develop software on a device with CIL-VM. Two implementations of CIL-VM are available to execute programs on small computer devices. The more popular one is Common Language Runtime (CLR) by Microsoft, which provides CLR as a runtime system of CIL included in the .NET Micro Framework (NMF/SPOT) for such small computer devices as mobile phones, smartphones, and industrial embedded computers that have a 32-bit CPU and 64-KB RAM. The other is presented in the Mono open source project. Mono's CIL-VM works on various small computer devices that have Linux, a 32-bit CPU, large RAM, and large program memory. Each implementation requires a 32-bit CPU and over 64-KB RAM to work; however, most sensor devices have only an 8- to 16-bit CPU, 2- to 4-KB RAM, and 32- to 64-KB program memory. It is important to reduce the RAM size for sensor devices because it increases both the cost to implement hardware devices and their physical size.

Thus, to provide an executable CIL system for small sensor devices, we designed CILIX, which requires only an 8- to 16-bit CPU, 4-KB RAM, and 32-KB program memory. In this paper, we describe the detailed technical issues for designing and implementing such small devices.

In the design of CILIX architecture, we focus on the following three requirements:

1) Compatibility: CILIX must have high compatibility with the existing implementations of CLR and Mono virtual machines.
2) Functionality: CILIX must provide the necessary functions for wireless sensor devices.
3) Memory-Saving: For porting on small memory devices, CILIX's program size should be much smaller than existing CIL-VMs. Moreover, we introduce a mechanism to reduce the size of the CIL program stored in the memory.

In general, strong compatibility and functionality increase the program size of the runtime system. Our main challenge is to develop techniques to reduce the required program memory without any deterioration of compatibility and functionality.

The fully compatible CIL-VM described in ECMA 335 [4] has many functions not used in CLR, which is included in the .NET Framework.[1] For example, *sleeping* functions increase the program size. In our first approach to reducing program memory, we omit them in our designed VM after we investigate the necessity of every function described

[1]These functions may be supported in the future or perhaps Microsoft just added as many functions as possible.

in ECMA 355. In the second approach, we identify the functions that require large program size, but that are used only for limited purposes. It is impossible for small resource-poor devices to provide all the functions of the full version CIL-VM described in ECMA 355 [4] because the full version of CIL-VM is designed for rich computer devices that have a 32-bit CPU, large RAM, program memory, and various I/O devices. To develop a CIL-VM with reasonable compatibility, we checked all the functions in CIL-VM presented by ECMA 355 and the number of program codes generated by both the compiler `csc.exe` in the .NET Framework and the `gmcs` command provided by the Mono project. We carefully chose functions that are not implemented on CILIX from the viewpoint that they are very rarely used for small sensor devices. For example, functions to execute unmanaged code are available for using existing native libraries such as `Win32API.dll`, but small sensor devices have no such libraries. Therefore, we do not support any functions that execute unmanaged code on a sensor device. The details and the reasons for our choices of functions are described in Section 2.4. Section 3 presents information to implement a CIL-VM that has substantial compatibility to the existing CIL-VMs.

We also introduce a mechanism called a Metadata Pre-Processor (MPP) to reduce the size of the CIL program to be executed on a sensor derive. The large CIL program also consumes memory space on a device, because the CIL program code must be stored on the program memory. The PE (`.exe`) file that includes the CIL program code has some redundant and unused data in the runtime. Our implemented MPP removes such unused data and compresses the redundant data.

We implemented CILIX on ATMega128L (8-bit CPU, 4-KB RAM, 128-KB program memory), MSP430, (16-bit CPU, 4-KB RAM, 32-KB program memory), and TWE-001 (32-bit CPU, 128-KB RAM, 128-KB program memory). To check both the size of the program memory and the compression ratio of the program code by MPP, we developed several programs for encoding, data compression, numerical treatment, sorting, and so on. Moreover, we compared the processing time and the size of the used memory on our implemented CILIX with existing virtual machines in CLR and Mono. As a result of these experiments, we show that CILIX can execute CIL program code for practical usage on several small sensor devices that have limited computational resources.

2. Requirements and Design

As mentioned in Section 1, we designed CILIX to meet three requirements: having high compatibility with existing CIL-VM; providing available functions to work on a wireless sensor device; and reducing both the size of CILIX and the CIL program stored on ROM or flash memory. In this section, we show the CILIX architecture after explaining our three requirements.

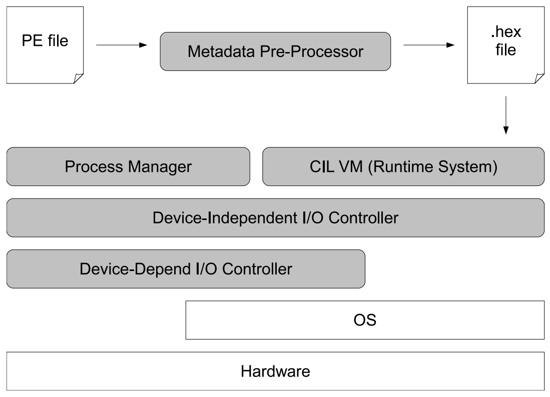


Fig. 1: CILIX Architecture

2.1 Compatibility

The virtual machine brings a standardization of environments to develop software by abstracting the diverse platforms of sensor devices. In other words, compatibility with existing virtual machines is one of the most important requirements in porting a virtual machine to a new platform. Therefore, we designed CILIX as a highly compatible VM that can execute a CIL program (.exe file) compiled by existing compilers, such as `csc.exe` provided by Microsoft and `gmcs` provided in the Mono project. Even though CILIX has high compatibility with existing VMs, it is difficult to implement a fully compatible VM on a small sensor device that has only limited computational resources. To achieve both high compatibility and porting onto a resource-poor device, we carefully removed the unsuitable functions that are too expensive to implement on a sensor device.

2.2 Functionality

The CIL-VM defined in ECMA 335 is designed as a simple virtual 32-bit stack-based processing unit, similar to JVM. The CIL-VM only has such essential functions as number calculation, transferring data on the memory, and controlling the executed program. In general, to support practical functions, for example, I/O management, threading, and file systems, developers implement these functions as a class library. To archive both the reduction of the size of CILIX and to provide useful functions for developing program code on a sensor device, we implemented the following three significant functions as an embedded class library in CILIX:

1) Dynamic program relocator
2) Interfaces for typical I/O devices (sensors and wireless communication devices)
3) Multi-threading controller

These functions are supported by most existing middleware for small sensor devices. We implemented them as a class library that has compatibility with the .NET Framework class library. For example, we implemented the embedded `Thread` class to support the multi-threading mechanism. Our implemented `Thread` class also has methods `run()`, `stop()`, `wait()`, and so on.

Each method provides the same function as the method implemented in the `Thread` class, which is included in the .NET Framework class library.

2.3 Memory-Saving

Reducing memory is the most significant technical issue to introduce virtual machines into limited-resource devices. Increasing the size of the logical memory, such as EEPROM, flash memory, and RAM, increases the physical size of the device and its price. In other words, we can use a small, low-price sensor device to reduce the program size.

We determined the minimum hardware requirements for the device, which has 4-KB RAM and 32-KB program memory (EEPROM and/or flash memory). As mentioned in Section 1, our required minimum hardware is smaller than most existing sensor devices. The price of the minimum 8-16-bit device, which has 4-KB RAM and 32-KB program memory, is lower than \$5. *Richer* 8-/16-bit devices than our minimum requirements provide little price advantage for 32-bit devices. Therefore, we chose the above minimum hardware requirements.

2.4 CILIX Architecture

The CLI specifications are defined in ECMA-335 [4][5].[2] ECMA-335 has four partitions, I–IV, each of which has independent page numbers. In this paper, the following notation, "ECMA P-X Y P," means page Y in partition X of ECMA-335. For example, ECMA P-II 183 P means page 183 in partition II, and ECMA P-X S.Y means section Y in partition X.

The CILIX runtime system has the following four runtime modules:

- Executer: loads and executes CIL program data from EEPROM or flash memory.
- Process Manager: controls the start-up and the stopping of the virtual module and also has a function to dynamically replace the program data on the memory.
- Platform-independent I/O Manager: provides a method to access I/O devices and only includes program code that is independent from device architecture, such as processing string data, conversion of data types, and so on.

[2]We recommend referring to *The Common Language Infrastructure Annotated Standard*[5] as a technical document for CIL-VM. It has many helpful annotations to the original ECMA-335.

- Platform-dependent I/O Manager: provides a method to access physical I/O devices and includes device-dependent program code.

Figure 1 shows the runtime system architecture of our designed CILIX. CILIX is composed by runtime modules and a Metadata Pre-Processor (MPP), which compresses the size of the CIL program data (`exe*␣file)` MPP is an independent module from the runtime system that can reduce the total size of the CIL program data by removing unused program code from a `exe*`

3. Implementation

This section describes the implementation of CILIX. We show the information for the implementation of CIL-VM, which has substantial compatibility with existing runtime systems, before we explain the non-CLI modules to provide convenient functions for wireless sensor devices. For the I/O control module, we only describe the essential ideas to implement it.

3.1 Substantial Subset of CLI

As mentioned in the previous section, ECMA defines the CLI specifications, but existing compilers `csc.exe` and `gmcs` do not generate all the CIL operations, the metadata tables, and the signatures described in ECMA. To reduce the program size of the runtime system, we implemented CILIX as a substantially compatible CIL virtual machine without functions for supporting such unused data. We extracted the *minimum indispensable* information from ECMA to implement CILIX, which can execute any CIL program code generated by `csc.exe` and `gmcs` without unsupported functions, as explained in the previous section. To obtain information, we investigated a number of `.exe` files generated by existing compilers for C#, C++/CLI, and Visual Basic.

In the rest of this subsection, we describe the information to implement CILIX.This information is available for developers who want to implement another CIL virtual machine.

3.2 Process Management Module

This module, which has several important functions for controlling a process on a sensor device like a small embedded operating system, provides these functions: process initialization, multi-thread control, dynamic program relocation, and restoring from an exception. For the initializing process, the module allocates memory for the program and loads the data used in the program onto the runtime memory from the program memory. Next we describe the other functions.

3.2.1 Dynamic Program Relocator

CILIX provides a function to change a CIL program to execute by relocating the program code on the program memory. The Program Relocator can read program code from a MicroSD card or a remote server by wireless communication to put the read data into the program memory (flash memory).

The relocation process is very simple. When we use the wireless communication method for relocation, we must send a special packet to inform the next packet including the new program code. If the I/O manager of the wireless device finds the spatial packet, the manager informs the Process Manager who stops the program's execution before the Relocator starts to work. The program code is transferred as a set of UDP packets next to the special packet. We describe UDP-based communication in the next subsection. After the Relocator retrieves the program code from the buffer in the I/O manager, the Relocator puts the program code into the program memory. Finally, the Process Manager initializes and starts the new program.

For MicroSD cards, we put them into a MicroSD card slot. When the I/O Manager of the SD Card (SPI) finds a new MicroSD card, the Manager checks a file named `/program.hex` based on the FAT32 format. If the Manager finds the program, it informs the Process Manager who performs the same processes as for using a wireless device.

3.2.2 Thread Controller

A typical process on a sensor node is a combination of a program to read a value from a sensor in an interval and a program to send a set of read values to the server. In this case, we want to concurrently execute two programs on a sensor device. CILIX supports a multi-thread control mechanism, which is available for such uses. The Thread Controller of CILIX provides simple concurrent processing in a sensor device. Each thread has an independent heap area and a buffer to back up the data in the managed area. CILIX has a memory space for the managed area of the current thread. This memory space stores the global variables used in the runtime.

To exchange the current thread with a suspended thread, CILIX moves all the data in the managed area into the current thread's buffer after CILIX stops to execute the current thread. Next, it moves all the data in the buffer in the suspended thread and switches to the heap area. Finally, it executes a new current thread with the heap area and manages all the thread's data.

The context switching interval can be given by the developer. As a default setting parameter, CILIX switches the thread every 160 opcodes.[3] In our implementation, CILIX can manage any number of threads as long as the device has memory space.

To maintain compatibility with the .NET Framework, we implement three embedded classes: System.Threading.-Monitor, System.Threading.Thread, and System.Threading.-ThreadStat.

[3]Note that we only give this value through our experiments to execute a number of practical programs on sensor devices.

3.2.3 Restoring from Exception

When an exception occurs and no `try/catch` block catches it, CILIX must process the restoring from the exception. If the device has a process management system such as an operating system, the system recovers the uncaught exception. On small devices without such a recovery system, CILIX recovers the exception. CILIX provides two alternative methods; the runtime system restarts the program in the first method, and the runtime system halts the process and waits for the program code sent from the server.

3.3 I/O Control Module

A small sensor device has various types of I/O devices, for example, thermometers, acceleration sensors, UART, SD cards, and radio frequency devices for wireless communication, LEDs, and LCDs. In general, developers must write specific program code for each individual device. To abstract I/O devices, we introduce a UDP-based interface in the design of an I/O control module. CILIX allows us to access each I/O device with UDP packets.

Our design offers the following three benefits:

1) Selecting a device with a port number: to change the device to the access mode, a developer only changes the port number related to the device.
2) Emulation of a device as a UDP program on a PC: we can build an I/O device as a program with a UDP port on a PC for debugging.
3) Concentration of device-dependent code in `send` and `recv`: any CIL program can only access an I/O device through the `send` and `recv` methods. In other words, all the program code, which depends on each I/O device, is gathered into these methods.

CILIX supports reading the `/program.hex` file from FAT32 sectors on SPI devices. We only suppose the usage of SPI devices to transfer a CIL program file from a PC to a small device. CILIX does not currently support reading other files or writing data onto an SPI device because we must add too much code to support those functions. For reading a program file, CILIX does not use UDP packets to improve the time to load program code in the memory. The runtime system automatically checks the SPI device to determine whether it has a program file to load during the runtime system's initial process.

4. ICT Application for Agriculture

In this section, we show an application systems using our designed virtual machine.

To improve flower yields, we built a remote environmental monitoring system using our 17 sensor nodes in two greenhouses from July to December in 2012. This application system consists of the following three sub-systems (Fig. 2):

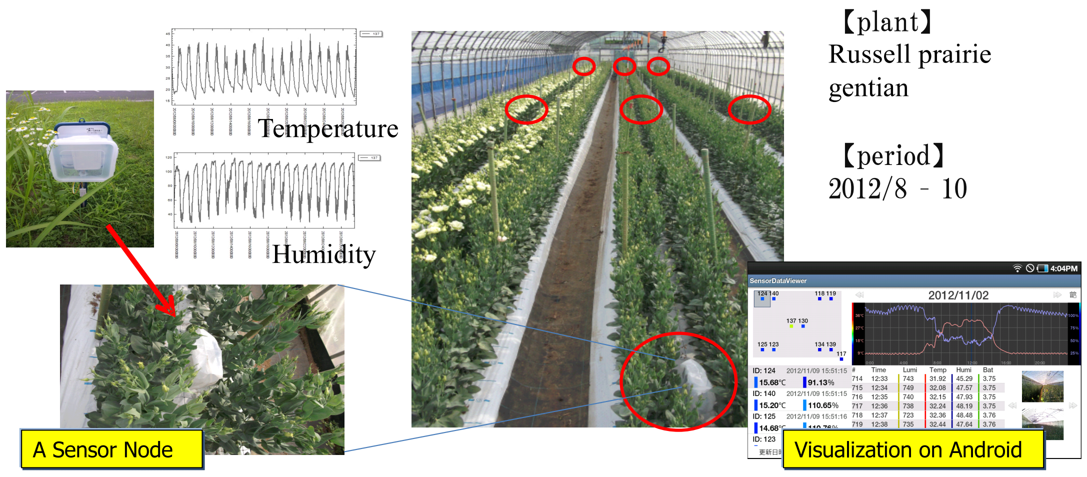


Fig. 2: Environmental monitoring system deployed in greenhouses

Table 1: Middleware specifications for wireless sensor devices

Name	Type	Language	Program relocatable
TinyOS	OS	nesC, TinyScript (Java, C#)	yes
Smart-Its	Library	C	yes
PAVENET	OS	C	yes (IrDA)
Ubiquitous Chip	OS	ECA rules	yes
BTnodes	OS	C/C++	yes (Bluetooth)
Nano-RK	OS	C	no
SimpleRTJ	VM	Java	no
Darjeeling	VM	Java	yes
Sun Spot	VM	Java	yes
Mate	VM	bytecode	yes
ASVM	VM	TinyScript, motlle, and TinySQL	yes
VM*	VM	Java	yes
NMF	VM+OS	C#, VB.NET, Managed C++	yes
Cilix	VM	C#, VB.NET, Managed C++	yes

Table 2: Hardware requirements of middleware for wireless sensor devices (minimum hardware requirements of each middleware.)

Name	CPU	Bit of CPU	ROM	RAM
TinyOS	Atmega128L (MICA Mote)	8bit	128KB+EEPROM 4KB	4K
Smart-Its	PIC18F6720	16bit	128KB + EEPROM 1KB	4KB
PAVENET	PIC18F452 (U-cube)	16bit	32KB + EEPROM 256B	4KB
Ubiquitous Chip	PIC16F873	14bit	4KB + EEPROM 128B	192B
BTnodes	Atmega128	8bit	128KB + EEPROM 4KB	4KB
Nano-RK	Atmega128 (Fire Fly)	8bit	128KB + EEPROM 4BKB + SROM (64KB+180K)	4KB
SimpleRTJ	Atmega128, ARM, H8	8/16/32 bit	14-36KB	0.2KB
Darjeeling	Atmega128, MPS430	8/16 bit	128KB	4-10KB
Sun Spot	ARM920T	32bit	4MB	512KB
Mate	Atmega128L (Mote)	8bit	128KB+EEPROM 4KB	4KB
ASVM	Atmega128L (Mote)	8bit	128KB+EEPROM 4KB	4KB
VM*	Atmega128L (Mote)	8bit	128KB+EEPROM 4KB + extra FLASH 512KB	4KB
NMF	ARM7/9/Cortex M3, ADI BlackFin	32bit	256KB	64K
Cilix	ATmega128, MSP430	8/16/32bit	32KB	4KB

1) Wireless sensor network: gathers sensor data in greenhouses.
2) Translator: receives sensor data from the sensor network and sends them to the server by a 3G/4G network.
3) Sensor data server: stores and retrieves sensor data.

The key system is the wireless sensor network that consists of TWE-001-based wireless sensor nodes. Each sensor node, which has the CIL program of our implemented virtual machine, obtains data from each sensor by the CIL program. The size of the compiled CIL program (exe file) is 16 KB, but our MPP compresses it to 4 KB. At the beginning of this experience, the measurement period was one minute on all the sensor nodes, but later we often changed the period (1-10 minutes) to optimize the balance between energy consumption and the value of the data on each sensor. Several times we also rewrote the program to change the routing protocol and the data compression algorithm to fit the sensor network. In the modification process, we used a function for a dynamic program relocation with wireless communication to reduce the cost of rewriting the program on each sensor node deployed in a large greenhouse. Our CIL virtual machine reduces the cost of program development because many existing Windows programmers have mastered how to develop applications for sensors network for a short period.

5. Related Work

In this section, we describe such previous proposed middleware as operating systems and virtual machines and explain the position of our implemented CILIX in them. Table 1 and 2 show the existing middleware systems and virtual machines.

The middleware for small sensor nodes can be classified into two types of software. The first is operating systems, such as TinyOS[6], Smart-Its[7], PAVENET[8], Ubiquitous Chip[9], BTnodes[10], and Nano-RK[11]. The second is virtual machines, such as Darjeeling[3], SimpleRTJ[2], Sun Spot[12], Maté[13], ASVM[14], and VM*[15]. Several systems present both functions; for example, the .NET Micro Framework (NMF) [16] includes both a virtual machine and an operating system. We focus on the cost of developing programs on a sensor node for each proposed system.

TinyOS is one of the most common operating systems for developing a program on a wireless sensor device. It provides necC and TinyScript as the programming languages, even though they are not so popular for developing programs for resource-rich computer devices. The programs developed in the Ubiquitous Chip are described as a small set of simple ECA rules. However, the developers must be familiar with the techniques to write a program with event-driven rule-based languages. Smart-Its presents libraries to develop a program on a PIC device with C language. Several operating systems only allow C language for program development. For instance, BTnode provides an environment to develop a program on a sensor device with Bluetooth, PAVENET supports a function for real-time hardware processing on a small device, and Nano-RK can be used for developing a program to process data on a sensor node in real time. To develop a program on a sensor device, we must use a specific programming language for each operating system.

Virtual machines can be classified into two types of systems: a subset system of Java Virtual Machine (JVM) and one with an original language (not Java). Maté[13] has a non-Java virtual machine, which can execute a program on TinyOS. The internal language of this virtual machine has a set of 24 *convenient* operation codes for wireless sensor devices. For example, it has a function to retrieve data from a sensor device with only one-byte operation code, which enables us to reduce the size of the program code. ASVM, another non-Java virtual machine for TinyOS, allows us to customize the set of operation codes. The original virtual machine achieves both powerful functions and a method to reduce the size of the program code; however, we do not choose a programming language to develop software on a sensor device.

On the other hand, there are several subsets of JVM, such as SimpleRTJ, VM*, Sun Spot, and Darjeeling. To execute a Java program on a limited-resource device, SimpleRTJ provides a mechanism to pack and compress both the runtime system and the Java program code into a small program module. Darjeeling presents a program converter from original Java byte code to compressed byte code. Since Darjeeling also provides a virtual machine to execute compressed Java byte code, it enables us to execute a Java program on a resource-poor device. VM*, which is a virtual machine implemented on a MOTE device, reduces the program code by the following two mechanisms. The first removes unused code from the `.class` file, and the second selects a minimum set of modules to execute Java byte code. Sun Spot presents a virtual machine to execute Java byte code without any conversion; however, it requires a *rich* device to execute a program. Every virtual machine has several implementations for 8-bit, 16-bit, and/or 32-bit CPUs. Each virtual machine allows us to develop a program only with Java programming language.

As mentioned, most existing middlewares limit the environment to develop a program on a sensor device. On the other hand, CIL-VM can execute programs written in several programming languages, including C#, C++/CLI, J++, Visual Basic, and F#. The .NET Micro Framework (NMF) provides a runtime system for a 32-bit CPU, but it cannot support 16-/8-bit CPUs. Because small sensor devices usually have an 8-/16-bit CPU, a developer cannot use it to develop programs on them. Since our CILIX can execute a CIL program on such a limited-resource device with an 8-/16-bit CPU, we can develop a program with many different programming languages.

6. Conclusions

This paper presented the design and implementation of CILIX, which can work on an 8-/16-bit CPU, 4-KB RAM, and 32-KB program memory with reasonable compatibility to existing CIL-VMs. We implemented CILIX on three devices, ATmega128L, MSP430, and TWE-001, and experimentally evaluated the performance of our implemented CILIX with them. We showed that CILIX can execute CIL program code with practical processing times on each device with limited resources. We hope our research leads to the development of another CIL-VM.

References

[1] M. Kuorilehto, M. Hännikäinen, and T. D. Hämäläinen, "A survey of application distribution in wireless sensor networks," *EURASIP J. Wirel. Commun. Netw.*, vol. 2005, pp. 774–788, October 2005.

[2] RTJ-Computing, "The Simple Real Time JAVA," http://www.rtjcom.com/.

[3] N. Brouwers, K. Langendoen, and P. Corke, "Darjeeling, a feature-rich vm for the resource poor," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '09. New York, NY, USA: ACM, 2009, pp. 169–182. [Online]. Available: http://doi.acm.org/10.1145/1644038.1644056

[4] ECMA, "Standard ecma-335: Common language infrastructure (cli)," http://www.ecma-international.org/publications/standards/Ecma-335.htm.

[5] J. S. Miller, *The Common Language Infrastructure Annotated Standard*. Addison-Wesley Professional, 2003.

[6] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," *SIGPLAN Not.*, vol. 35, pp. 93–104, November 2000. [Online]. Available: http://doi.acm.org/10.1145/356989.356998

[7] M. Beigl and H. Gellersen, "Smart-its: An embedded platform for smart objects," in *In Proc. Smart Objects Conference (SOC 2003*, 2003, pp. 15–17.

[8] S. Saruwatari, T. Kashima, M. Minami, H. Morikawa, and T. Aoyama, "Pavenet: A hardware and software framework for wireless sensor networks," *Transaction of the Society of Instrument and Control Engineers*, vol. E-S-1, no. 1, pp. 74–84, 2005.

[9] T. Terada, M. Tsukamoto, K. Hayakawa, T. Yoshihisa, Y. Kishino, A. Kashitani, and S. Nishio, "Ubiquitous chip: A rule-based i/o control device for ubiquitous computing," in *Pervasive Computing*, ser. Lecture Notes in Computer Science, A. Ferscha and F. Mattern, Eds. Springer Berlin / Heidelberg, 2004, vol. 3001, pp. 238–253.

[10] B. Project, "Btnodes - a distributed environment for prototyping ad hoc networks," http://www.btnode.ethz.ch/.

[11] A. Eswaran, A. Rowe, and R. Rajkumar, "Nano-rk: An energy-aware resource-centric rtos for sensor networks," *Real-Time Systems Symposium, IEEE International*, vol. 0, pp. 256–265, 2005.

[12] Oracle, "Sun spot," http://jp.sun.com/products/software/sunspot/.

[13] P. Levis and D. Culler, "Maté: a tiny virtual machine for sensor networks," *SIGPLAN Not.*, vol. 37, pp. 85–95, October 2002. [Online]. Available: http://doi.acm.org/10.1145/605432.605407

[14] P. Levis, D. Gay, and D. Culler, "Active sensor networks," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2*, ser. NSDI'05. Berkeley, CA, USA: USENIX Association, 2005, pp. 343–356. [Online]. Available: http://portal.acm.org/citation.cfm?id=1251203.1251228

[15] J. Koshy and R. Pandey, "Vmstar: synthesizing scalable runtime environments for sensor networks," in *Proceedings of the 3rd international conference on Embedded networked sensor systems*, ser. SenSys '05. New York, NY, USA: ACM, 2005, pp. 243–254. [Online]. Available: http://doi.acm.org/10.1145/1098918.1098945

[16] M. Corporation, ".net micro framework," http://www.microsoft.com/en-us/netmf/default.aspx.

Quantifying Sentiment for the Japanese Economy and Stock Price Prediction

Hiroshi Ishijima[1], **Takuro Kazumi**[2], **and Akira Maeda**[3]
[1]Graduate School of International Accounting, Chuo University, Shinjuku, Tokyo, Japan
and Center on Japanese Economy and Business, Columbia Business School, New York, NY, USA
[2]CyberAgent, Inc., Chiyoda, Tokyo, Japan
[3]Graduate School of Arts and Sciences, The University of Tokyo, Meguro, Tokyo, Japan

Abstract - *The purpose of this paper is to re-examine the analysis of Ishijima et al. [2], providing a more comprehensive analysis on the sentiment of the Japanese economy that appears on the Nikkei articles. We extend their analysis in two dimensions: One is to expand data coverage, and the other is to create variations of the original sentiment index. Granger causality test results indicate the usefulness of our sentiment indexes.*

Keywords: Big data, Text mining, Nikkei, Sentiment, Stock market

1 Introduction

Sentiment analysis is gaining increasing interest in both academia and businesses. As the sentiment invisibly reflects the atmosphere of economic activities and the psychology of economic agents, analyzing the sentiment helps us understand the economy and security markets in a more sophisticated way. With such a conception, Ishijima et al. [2] analyzed the sentiment towards the Japanese economy that might appear in daily news articles. In fact, they created an index that accounts for the frequency of occurrence of words that affirmatively or negatively describe the current economic situation. Articles were taken from the *Nikkei*, a popular business newspaper in Japan. They then performed statistical analysis to examine correlations between the sentiment index and Tokyo Stock Exchange prices. Interestingly, they concluded that the index significantly predicts stock prices of three days in advance.

The purpose of this paper is to re-examine the analysis of Ishijima et al. [2], providing a more comprehensive analysis on the sentiment of the Japanese economy that appears on the *Nikkei* articles. To this end, we extend their analysis in two dimensions: One is to expand data coverage, and the other is to create variations of the original sentiment index.

(1) Data that covers 29-year-horizon:

In their study, Ishijima et al. [2] only covered a period of five years that ranges from January 2007 to September 2012. In contrast, we work on a longer sample period that covers years from March 1984 to September 2012. On a yearly basis, we examine the predictability of stock prices by our sentiment indexes.

(2) Variations of sentiment indexes:

We reconsider the methodology of creating index and newly propose four indexes. The detail of our methodology is the following:

Scoring process

We quantify the sentiment along one-dimensional semantic axis; that is from negative to positive feelings. For every single word that appears in the Nikkei, we match it to the prescribed semantic dictionary developed by Takamura [3]. If it matches, we record the score paired with the word that represents how much it associates the negative feeling with the Japanese people. In this scoring process, there are two aspects that we can deal with. One is about how to score on each of matched words and the other is about how far we will cover the Nikkei pages – just headlines or entire set of articles. We will elaborate on each of these aspects.

Scoring method

The semantic dictionary (Takamura [3]) provides, to each of contained words, the score that ranges from -1 to 1. The score shows the closer the score becomes to -1, the more negative feeling people associates the word with, and vice versa. We then exploit the score in two ways: Using the raw score or rounding to the nearest integer score that is either +1 or -1. We call the former scoring "real-valued" and the latter "integer-valued." In the latter integer-valued case, we round to -1 if the raw score ranges between -1 and 0, and otherwise +1.

Coverage of source in the Nikkei

Then we summed up these scores over the following two sources: One is limited to headlines, the other covers the entire set of articles. We call the former coverage "Headlines Only" and the latter "Entire Set of Articles." These two ways of counting scores make us understand how important the sentiment exhibited in headlines is in predicting the stock prices, as comparing the stock price predictability of sentiment in the entire set of articles.

With the above-mentioned methodology, we can have two scoring methods and two extents of coverage in the Nikkei. It results in four ways to create the sentiment index. While, in Ishijima et al. [2], they only created and examined one of four sentiment indexes; that is the integer-valued article sentiment index in our category.

The rest of this paper is organized as follows. Section 2 elaborates on how to create sentiment indexes. Section 3 builds our models. Section 4 implements an empirical analyzes in the Japanese stock markets. Section 5 concludes the paper.

2 Creating Sentiment Indexes

Source: the coverage of pages to pick words

Every page in newspapers is comprised of pairs of headlines and articles. The place where the word appears either in headlines or in articles might affect the impact of how much the reader will invoke positive or negative sentiment on that word. In this aspect, we strictly distinguish the words in articles from the words in headlines. We clarify on this by introducing some notations.

In the newspapers delivered at day t, we have $n_t^{(H)}$ headlines and $n_t^{(A)}$ articles. Each headline and article are denoted by $H_{i,t}\ (i = 1,\cdots,n_t^{(H)})$ and $A_{l,t}\ (l = 1,\cdots,n_t^{(A)})$. Each headline and article have $n_{i,t}^{(H)}$ and $n_{l,t}^{(A)}$ words, respectively. The words in headline $H_{i,t}$ and article $A_{l,t}$ are denoted by $W_{ij,t}^{(H)}\ (j = 1,\cdots,n_{i,t}^{(H)})$ and $W_{lm,t}^{(A)}\ (m = 1,\cdots,n_{l,t}^{(A)})$, respectively.

At this point, we introduce the aggregate notation to represent whichever the word that comprises either headlines or articles. This enables us easier to articulate how to quantify the sentiment in the discussion that will follow. The coverage of pages to pick words is limited to either headlines or articles and is denoted by $\mathcal{G} := \{H, A\}$. We simply call $\mathcal{G}$ as the "*source*." We then let $G \in \mathcal{G}$ to show either headline H or article A.Then in the newspaper delivered at day t, $W_{ij,t}^{(G)}$ denotes the j-th word $(j = 1,\cdots,n_{i,t}^{(G)})$ that comprises the i-th source $G_{i,t}\ (i = 1,\cdots,n_t^{(G)})$.

Semantic dictionary

The semantic dictionary (Takamura, [3]) is denoted by $\mathcal{D} := \{(D_k, S(D_k)) | k = 1 \cdots K\}$. That is, the dictionary comprises pairs of word and its semantic score that ranges from -1 to +1. Regarding the semantic score, the closer to -1 (or +1) the score becomes, the more negative (or positive) feeling the word invokes to the Japanese people.

Semantic index: two methods to quantify the positive or negative feelings

We define the indicator function to count if the word that appears in the source matches one of listed words in the dictionary.

$$I_{ij,t}^{(G)}(k) := \begin{cases} 1 & \left(\text{if } W_{ij,t}^{(G)} \text{ matches } D_k\right) \\ 0 & (\text{otherwise}) \end{cases} \tag{1}$$

In the aspects of how to score the positive or negative feelings, we introduce two methods to create the sentiment indexes.

(1) Real-valued sentiment index

The first way is to exploit the semantic score $S_k = S(D_k)$ that is assigned to the listed word D_k. We define this first way as the "*real-valued sentiment index:*"

$$x_t^{(G,R)} := \sum_{i=1}^{n_t^{(G)}} \sum_{j=1}^{n_{i,t}^{(G)}} \sum_{k=1}^{K} I_{ij,t}^{(G)}(k) \cdot S_k \tag{2}$$

For the source $G = H$ that the coverage of pages to pick words is limited to headlines, the real-valued sentiment index is given by $x_t^{(G,R)} = x_t^{(H,R)}$. In the same way, for the source of $G = A$ (articles), the real-valued sentiment index is given by $x_t^{(G,R)} = x_t^{(A,R)}$.

(2) Integer-valued sentiment index

The second way is to round the semantic score S_k to the nearest binary integer that is either -1 or +1. Introducing the integer variable for each of semantics scores:

$$J_k := \begin{cases} +1 & (\text{if } 0 < S_k \leq 1) \\ -1 & (\text{if} -1 \leq S_k < 0) \end{cases} \tag{3}$$

We then define the second way as "integer-valued sentiment index":

$$x_t^{(G,I)} := \sum_{i=1}^{n_t^{(G)}} \sum_{j=1}^{n_{i,t}^{(G)}} \sum_{k=1}^{K} I_{ij,t}^{(G)}(k) \cdot J_k \tag{4}$$

Reminding that each of two sentiment indexes has the option in picking the source, – either headlines $(G = H)$ or entire set of articles $(G = A)$ – we thus have four types of sentiment indexes in the analysis.

As a summary, we use the following notation to represent these four sentiment indexes (s.i.).

$$x^{(G,\#)} := \begin{cases} x^{(H,I)} & (\text{integer} - \text{valued headline s.i.}) \\ x^{(H,R)} & (\text{real} - \text{valued headline s.i.}) \\ x^{(A,I)} & (\text{integer} - \text{valued article s.i.}) \\ x^{(A,R)} & (\text{real} - \text{valued article s.i.}) \end{cases} \tag{5}$$

where G denotes one of the sources H or A and # denotes the scoring method that is either integer-valued "I" scoring or real-valued "R" scoring.

Along the procedures that we described above, we created 29-year daily time-series of four sentiment indexes based on headlines and articles from the Nikkei. We remark that these sentiment indexes are normalized so that they have zero means and unit standard deviations. Due to space limitation, we omitted reporting summary statistics and time-series charts of our sentiment indexes.

3 Model

To explore whether or not our sentiment indexes are able to predict stock prices, we will exploit the vector auto-regression (VAR) model that is conventional one in the econometrics literature. For each of two cases in terms of the scoring method, we estimated three VAR(p) models which comprise either (1) Model H: headline sentiment index, (2) Model A: entire article sentiment index, (3) Model H&A: both headline and article sentiment indexes, as well as stock log-returns. Namely, each of three models is specified as follows:

Model H
$$y_t = \sum_{i=1}^{p} \alpha_i y_{t-i} + \beta_i x_{t-i}^{(H,R\ or\ I)} + \varepsilon_t \qquad (6)$$

Model A
$$y_t = \sum_{i=1}^{p} \alpha_i y_{t-i} + \gamma_i x_{t-i}^{(A,R\ or\ I)} + \varepsilon_t \qquad (7)$$

Model H&A
$$y_t = \sum_{i=1}^{p} \alpha_i y_{t-i} + \beta_i x_{t-i}^{(H,R\ or\ I)} + \gamma_i x_{t-i}^{(A,R\ or\ I)} + \varepsilon_t \qquad (8)$$

Within these VAR(p) model specifications, the *Granger causality* can be stated if the sentiment indexes ($x^{(G,\#)}$) *Granger-cause* (*G-cause*) stock log-returns (y), the past sentiment indexes should help predict the stock log-returns, beyond prediction by past stock log-returns alone. Using these three VAR(p) models, we will implement the three Granger causality tests (G-tests).

(1) "*G-test for Model H*" tests whether or not the headline sentiment index G-causes stock log-returns by exploiting Equation (6). The null hypothesis is $\beta_i = 0\ (i = 1 \cdots p)$.

(2) "*G-test for Model A*" tests whether or not the entire article sentiment index G-causes stock log-returns by exploiting Equation (7). The null hypothesis is $\gamma_i = 0\ (i = 1 \cdots p)$.

(3-1) "*G-test 1 for Model H&A*" tests whether or not the headline sentiment index G-causes stock log-returns by exploiting Equation (8). The null hypothesis is $\beta_i = 0\ (i = 1 \cdots p)$.

(3-2) "*G-test 2 for Model H&A*" tests whether or not the entire article sentiment index G-causes stock log-returns by exploiting Equation (8). The null hypothesis is $\gamma_i = 0\ (i = 1 \cdots p)$.

4 Empirical Analysis in the Japanese Stock Market

While the Nikkei is daily published and delivered with a few no-issue days, the Japanese stock market is closed every weekend. To handle such daily data set that is partially missing and hence inconsistent in frequency, we follow the approach of Bollen et al. [1]. That is, we eliminated every Saturday and Sunday from complete data set before implementing analysis.

Also, before estimating VAR models Eqs. (6)–(8), we implemented augmented Dickey–Fuller tests. For each year, we verified that all the time series of stock log-returns, headline and article sentiment indexes do not have unit root with 1% significance.

Table 1 and Table 2 show Granger test results with real- and integer-valued sentiment indexes, respectively. For each year, three models of Model H, A and H&A or relevant Eqs. (6), (7) and (8) are estimated and tested. On each model estimation, we searched the lag p from 1 to 7 to identify the best (i.e. lowest) p in terms of AIC. For each model estimation with the best p, the relevant test statistics ("Granger"-labeled columns) and AIC values are reported in Tables.

Table 1: Predictability of "real-valued sentiment index" in terms of Granger tests and AICs. Test statistics for Granger causality are given in the columns titled "Granger." *, ** and *** mark the test statistics that are 10%, 5% and 1% significant, respectively.

	Headline Eq. (6)			Article Eq. (7)			Headline & Article Eq. (8)			
Year	Lag (p)	Granger	AIC	Lag (p)	Granger	AIC	Lag (p)	Granger H	Granger A	AIC
1984	3	1.12	-10.17	6	1.51	-10.10	6	4.99***	5.20***	-11.19
1985	7	1.54	-10.46	6	1.78	-10.65	6	3.60***	3.67***	-11.45
1986	2	1.69	-10.18	6	1.00	-10.44	6	1.91**	1.17	-11.60
1987	2	0.02	-9.21	6	1.78	-9.77	6	0.71	1.64*	-11.17
1988	2	1.12	-11.07	6	0.66	-11.38	6	1.88**	1.28	-12.88
1989	2	2.89*	-11.53	5	1.39	-12.01	5	1.98**	1.86**	-13.70
1990	2	0.14	-8.73	7	1.96*	-9.63	5	3.30***	4.75***	-11.21
1991	2	0.40	-9.37	5	0.52	-10.35	5	3.48***	4.93***	-11.72
1992	5	1.29	-8.69	5	0.83	-9.48	5	2.06**	2.69***	-10.79
1993	1	1.67	-9.18	5	0.21	-10.03	5	3.17***	3.69***	-11.29
1994	1	0.41	-9.46	5	0.37	-10.27	5	1.77*	5.12***	-11.52
1995	5	0.94	-9.19	5	0.37	-9.98	5	3.91***	4.71***	-11.48
1996	5	1.08	-10.05	5	0.53	-11.08	5	1.70*	2.24**	-12.61
1997	2	0.07	-8.78	5	1.18	-9.72	5	2.31**	2.49***	-11.21
1998	5	3.01**	-8.80	5	2.22*	-9.94	5	1.43	3.23***	-11.28
1999	1	0.64	-9.29	5	0.29	-9.99	5	1.60	2.20**	-11.22
2000	1	4.35**	-8.99	5	1.37	-10.04	6	1.77**	3.74***	-11.36
2001	2	0.32	-8.59	1	0.36	-9.72	1	2.19	2.31*	-11.17
2002	2	2.03	-8.90	5	0.94	-10.10	5	0.77	1.42	-11.74
2003	1	0.12	-8.89	5	0.25	-10.30	1	0.54	2.59*	-11.99
2004	1	0.16	-9.50	5	1.30	-10.70	1	0.19	2.74*	-12.26
2005	5	0.18	-10.10	5	0.30	-11.24	1	4.29**	8.65***	-12.81
2006	5	0.70	-9.37	5	1.74	-10.53	5	1.07	3.57***	-12.27
2007	6	0.42	-9.59	6	0.15	-10.70	5	2.27**	4.40***	-12.32
2008	6	0.37	-7.68	5	0.58	-8.59	5	1.92**	4.48***	-10.15
2009	6	0.95	-8.87	6	0.83	-9.88	6	1.51	1.62*	-11.36
2010	5	2.35**	-9.24	5	2.26**	-10.36	5	1.51	2.72***	-12.02
2011	1	0.19	-9.18	5	1.13	-10.00	5	1.07	1.45	-11.44
2012	2	3.88*	-9.71	5	1.13	-10.91	5	1.20	1.04	-12.32

Table 2: Predictability of "integer-valued sentiment index" in terms of Granger tests and AICs. Test statistics for Granger causality are given in the columns titled "Granger." *, ** and *** mark the test statistics that are 10%, 5% and 1% significant, respectively.

	Headline Eq. (6)			Article Eq. (7)			Headline & Article Eq. (8)			
Year	Lag (p)	Granger	AIC	Lag (p)	Granger	AIC	Lag (p)	Granger H	Granger A	AIC
1984	1	0.35	-9.76	1	1.06	-10.04	1	2.77*	2.72*	-10.19
1985	2	3.03**	-10.39	3	3.39**	-10.72	1	1.16	0.69	-10.83
1986	1	2.58	-9.62	1	2.21	-9.49	1	3.56**	0.49	-9.79
1987	1	0.20	-8.40	6	1.29	-8.70	1	0.28	2.11	-9.08
1988	2	0.60	-10.37	5	1.87*	-10.77	1	0.18	0.98	-11.15
1989	6	1.92*	-10.42	1	2.49	-10.86	1	0.12	1.50	-11.11
1990	2	0.04	-7.65	2	1.01	-8.16	2	1.16	0.89	-8.28
1991	1	0.37	-8.69	1	1.36	-9.03	2	2.86**	1.76	-9.39
1992	1	2.07	-8.28	5	1.04	-8.29	1	0.95	0.23	-8.80
1993	4	3.51***	-8.81	1	0.03	-9.06	1	0.87	0.03	-9.34
1994	5	1.32	-9.55	1	0.01	-9.59	1	0.14	0.42	-10.31
1995	1	0.21	-8.86	1	0.11	-9.28	1	0.19	0.52	-9.74
1996	1	2.12	-9.33	1	0.12	-9.73	1	1.70	0.44	-9.91
1997	2	1.04	-8.67	2	0.42	-8.82	2	0.53	0.57	-9.48
1998	2	0.78	-8.49	5	2.05*	-8.78	1	1.43	2.12	-9.40
1999	1	0.10	-9.35	1	0.05	-9.57	1	0.11	7.06***	-10.39
2000	1	0.31	-8.41	1	0.65	-9.11	1	0.86	0.67	-9.23
2001	1	1.43	-8.28	1	0.02	-8.48	1	3.02**	0.59	-9.13
2002	1	0.10	-8.82	1	0.91	-9.01	1	0.02	1.70	-9.78
2003	5	2.88*	-8.70	4	1.71	-8.63	1	2.73*	2.29	-9.10
2004	1	0.01	-9.38	1	0.00	-9.69	1	2.96*	0.23	-10.38
2005	1	0.48	-9.74	5	0.41	-9.67	5	1.13	1.45	-10.13
2006	1	0.17	-9.24	2	0.13	-9.70	2	0.86	1.02	-10.30
2007	1	1.28	-9.06	6	2.21**	-9.27	1	2.41*	3.40**	-9.66
2008	1	0.32	-7.62	3	0.41	-7.68	1	1.04	2.16	-8.31
2009	3	1.73	-8.38	6	1.57	-8.02	1	6.26***	0.91	-8.60
2010	3	1.55	-8.89	5	2.18*	-9.22	3	2.64**	1.35**	-9.56
2011	1	0.10	-8.61	4	2.43**	-8.95	1	1.34	3.48**	-9.21
2012	1	0.87	-9.56	3	0.95	-9.86	1	1.95	0.20	-10.36

Goodness of fit:

For both real- and integer-valued cases, Model H&A fits best in terms of AICs throughout 29 years. When comparing the real- and integer-valued cases, the former performs better in the aspects of AICs. Hence, in the following discussion, we will mainly focus on the results for estimating Model H&A on the basis of real-valued sentiment index.

Predictability of stock prices:

From the right most panel in Table 1, Model H&A of Eq. (8) persistently shows the predictability of stock prices on the basis of real-valued sentiment index. More specifically, the article index persistently and significantly Granger-causes the stock log-returns in conjunction with the headline index ("Granger A" column in that right most panel). Also, the headline index Granger-causes the stock log-returns in conjunction with the article index ("Granger H" column in the same panel). On the flipside, Models H and A (Eqs. (6) and (7)) do not seem to provide the persistent Granger causalities. These results imply that it is important for our VAR modeling to incorporate both headline and article sentiment indexes in order to predict stock prices; and that it is insufficient to separately introduce either headline or article sentiment indexes[1].

It should be noted that the Granger causality seems to be weakened during some periods. We will elaborate on each of these. During the period from 1986 to 1988 that was right after the Plaza Accord on 1985, the Japanese market has been a bull market and on the way to its peak. In this period, the headline sentiment index has stronger Granger causality than the article's except 1987 that has brought Black Monday.

In contrast, during the period from 2001 to 2004 that was right after the burst of the Internet bubble, in the year of 2009 that was right after the 2008 financial crisis, and during two years from 2011 to 2012 in which we experienced the Japan quake and had been trying to recover, the Japanese economy suffered from these unusual events. In those periods, the article index has stronger Granger causality than the headline index.

Significant lagged variables:

Table 3 exhibits the year-on-year estimates of Model H&A of Eq. (8) on the basis of real-valued sentiment index. Interestingly enough, we found the seven cyclical patterns in our estimation results.

Cycle 1 was significant: For the first 10 years from 1984 to 1993 except 1986 – that correspond to the 11th business cycle defined by Cabinet Office in Japan –, two sentiment indexes with shorter lags (1 to 2) and longer lags (5 to 6) served as significant variables.

Cycle 2 was not significant: In the next 5 years from 1994 to 1998 that correspond to the 12th business cycle in Japan, we couldn't find any significant lagged variables on neither headline nor article indexes.

Cycle 3 was significant: In the next 2 years from 1999 to 2000 that correspond to the Internet bubble, two sentiment indexes with lag-of-one and middle lags (3 to 4) were significant.

Cycle 4 was not significant: During the period from 2001 to 2005 that was after the burst of the Internet bubble and before 2008 financial crisis, we couldn't find any significant lagged variables in sentiment indexes, again.

Cycle 5 was significant: In the 3 years from 2006 to 2008 that brought the financial crisis, two sentiment indexes with lag-of-one and middle lags (3 to 4) were significant.

Cycle 6 was not significant: During the period from 2009 to 2010 that was right after the crisis and corresponded to the very first two years of the 15th business cycle in Japan, we couldn't find any significant lagged variables in sentiment indexes.

Cycle 7 was significant: In recent two years after the Japan quake, two sentiment indexes with lag-of-two were significant.

Comparison with the relevant work:

Ishijima et al. [2] reported that during the period after the 2008 financial crisis, the integer-valued article sentiment index alone significantly predicts stock prices three-days-ahead. This can be found in the middle panel titled "Article Eq. (7)" on Table 2. Indeed, we can see the significant Granger causalities around 2008. Unfortunately, this finding does not seem to be persistent when we review this from 29-year-horizontal results that we have shown in this paper.

5 Conclusions

We created the 29-year daily time-series of four sentiment indexes that reflect the positive or negative feelings represented in the Nikkei newspaper. The analysis is based on Ishijima et al. [2], but is a sophisticated version of their analysis. We showed the persistent predictability of Japanese stock prices on the basis of two sentiment indexes that quantified the sentiment over headlines and entire articles, respectively.

6 References

[1] J. Bollen, H. Mao and X. Zeng, "Twitter Mood Predicts the Stock market," *Journal of Computational Science*, vol. 2, no. 1, pp. 1–8, 2011.

[2] H. Ishijima, T. Kazumi and A. Maeda, "Sentiment Analysis for the Japanese Stock Market," *forthcoming in Global Business and Economics Review*, 2014.

[3] H. Takamura. (2007). Tango Kanjyo Kyokusei Taiou Hyou (Semantic Orientations Dictionary). [Online]. Available URL: http://www.lr.pi.titech.ac.jp/~takamura/pndic_ja.html (accessed 2014-05-31)

[1] Results shown here are those of Granger causality tests on the basis of OLS estimators. We remark that the data we handled may contain sample biases. Hence, to consider the hetero-scedasticity due to such possible sample biases, we double-checked to conduct tests with robust covariance-matrix estimators as well. The results were mostly the same in case of real-valued sentiment index, but there were found small differences in case of integer-valued sentiment index. The detail is omitted here due to space limitation.

Table 3: Estimated parameters of Model H&A Eq. (8) on the basis of "real-valued sentiment index": Estimates from 1984 to 2011. Figures in parentheses show the p-values. *, ** and *** mark 10%, 5% and 1% significant variables, respectively.

				Lag 1			Lag 2			Lag 3			Lag 4			Lag 5			Lag 6	
Year	Lag (p)	Const	Ret	Head	Art	Ret	Head	Art	Ret	Head	Art	Ret	Head	Art	Ret	Head	Art	Ret	Head	Art
1984	6	-0.0011	0.2275***	0.0014*	-0.0006	0.0244	0.0016**	-0.0019***	-0.0894	-0.0011	0.0009	0.2067***	-0.0014*	0.0002	-0.1650**	0.0007	-0.0011	-0.0470	-0.0005	0.0008
		(0.5962)	(0.0013)	(0.0699)	(0.3393)	(0.7278)	(0.0388)	(0.0078)	(0.1960)	(0.1588)	(0.2099)	(0.0031)	(0.0874)	(0.8205)	(0.0205)	(0.3535)	(0.1006)	(0.4882)	(0.5192)	(0.2590)
1985	6	0.0004	0.0693	-0.0005	0.0007	-0.0619	0.0004	-0.0011**	-0.0962	-0.0003	0.0001	-0.0169	-0.0003	0.0003	-0.0118	0.0006	-0.0004	-0.0755	0.0011**	-0.0004
		(0.5817)	(0.2604)	(0.3683)	(0.1773)	(0.3202)	(0.4617)	(0.0211)	(0.1235)	(0.5829)	(0.8982)	(0.7859)	(0.5911)	(0.5659)	(0.8489)	(0.2517)	(0.4320)	(0.2232)	(0.0217)	(0.3514)
1986	6	0.0012**	0.2369***	0.0000	0.0010	-0.0375	-0.0007	-0.0006	-0.0633	-0.0004	-0.0002	0.0927	0.0007	-0.0010	0.0019	-0.0003	-0.0004	-0.1089*	-0.0002	0.0007
		(0.0302)	(0.0002)	(0.9741)	(0.3474)	(0.5688)	(0.4963)	(0.5621)	(0.3353)	(0.6666)	(0.8704)	(0.1586)	(0.5046)	(0.2996)	(0.9771)	(0.7704)	(0.6808)	(0.0904)	(0.8309)	(0.5102)
1987	6	-0.0017	-0.0254	-0.0030	0.0056**	-0.1594**	0.0017	-0.0020	0.0669	0.0011	-0.0018	-0.1119*	0.0000	0.0015	0.1661***	0.0001	0.0002	-0.1318**	-0.0013	0.0049**
		(0.4925)	(0.6876)	(0.1543)	(0.0362)	(0.0118)	(0.4245)	(0.4570)	(0.2936)	(0.6203)	(0.4950)	(0.0805)	(0.9817)	(0.5756)	(0.0091)	(0.9676)	(0.9503)	(0.0407)	(0.5098)	(0.0478)
1988	6	0.0005	0.0502	-0.0021**	0.0014	-0.0463	0.0002	-0.0005	-0.0188	0.0001	0.0005	0.0119	0.0009	0.0003	-0.0601	-0.0015*	0.0017*	0.0311	0.0012	-0.0005
		(0.5698)	(0.4276)	(0.0153)	(0.1627)	(0.4620)	(0.7766)	(0.6257)	(0.7659)	(0.9308)	(0.5971)	(0.8327)	(0.2739)	(0.7567)	(0.2800)	(0.0726)	(0.0816)	(0.5704)	(0.1559)	(0.6136)
1989	5	0.0022**	0.0200	-0.0013	0.0002	0.0672	0.0000	-0.0009	0.0405	-0.0019**	0.0020**	-0.0771	0.0005	0.0004	-0.0049	-0.0007	0.0005			
		(0.0230)	(0.7612)	(0.1137)	(0.8145)	(0.3010)	(0.9675)	(0.3362)	(0.5321)	(0.0283)	(0.0397)	(0.2289)	(0.5839)	(0.7003)	(0.9382)	(0.3809)	(0.5943)			
1990	5	0.0021	0.1275*	0.0025	-0.0074*	-0.2147***	-0.0015	0.0006	-0.0098	0.0036	-0.0055	0.0369	-0.0005	-0.0029	-0.1191*	-0.0006	-0.0005			
		(0.5849)	(0.0566)	(0.4152)	(0.0633)	(0.0015)	(0.6301)	(0.8865)	(0.8872)	(0.2445)	(0.1721)	(0.5831)	(0.8614)	(0.4681)	(0.0745)	(0.8525)	(0.9076)			
1991	5	-0.0006	0.0331	0.0030*	-0.0022	-0.1459**	-0.0008	0.0022	-0.0459	0.0007	-0.0005	-0.0589	-0.0020	0.0003	0.0375	-0.0035**	0.0047**			
		(0.7449)	(0.6228)	(0.0861)	(0.3424)	(0.0297)	(0.6464)	(0.3471)	(0.4889)	(0.6917)	(0.8331)	(0.3792)	(0.2579)	(0.8941)	(0.5753)	(0.0448)	(0.0404)			
1992	5	-0.0044	-0.0328	-0.0009	0.0031	-0.0157	0.0031	0.0000	0.0349	-0.0012	-0.0008	0.0487	-0.0038	0.0018	0.0908	-0.0039	0.0061*			
		(0.2150)	(0.6239)	(0.7116)	(0.3393)	(0.8130)	(0.2131)	(0.9944)	(0.5957)	(0.6280)	(0.8029)	(0.4570)	(0.1341)	(0.5856)	(0.1642)	(0.1147)	(0.0622)			
1993	5	0.0001	-0.0191	0.0014	-0.0008	0.0131	0.0015	-0.0011	-0.0616	0.0019	-0.0026	0.1002	-0.0006	0.0007	-0.0555	-0.0034**	0.0035			
		(0.9665)	(0.7766)	(0.4069)	(0.7460)	(0.8480)	(0.3863)	(0.6709)	(0.3659)	(0.2527)	(0.2948)	(0.1461)	(0.7287)	(0.7906)	(0.4239)	(0.0458)	(0.1523)			
1994	5	0.0004	-0.0804	0.0001	-0.0006	-0.0495	-0.0004	0.0014	0.0439	0.0019	-0.0020	0.0638	-0.0003	0.0001	-0.0950	-0.0006	0.0006			
		(0.8255)	(0.2246)	(0.9593)	(0.7481)	(0.4534)	(0.8045)	(0.4440)	(0.5054)	(0.1878)	(0.2830)	(0.3283)	(0.8186)	(0.9611)	(0.1464)	(0.6863)	(0.7511)			
1995	5	-0.0018	0.0335	-0.0020	0.0019	0.0183	-0.0010	0.0024	-0.0258	-0.0014	0.0021	-0.0561	-0.0028	0.0030	-0.0935	0.0004	0.0016			
		(0.3189)	(0.6124)	(0.3160)	(0.4632)	(0.7811)	(0.6096)	(0.3752)	(0.6978)	(0.5042)	(0.4187)	(0.4000)	(0.1728)	(0.2785)	(0.1645)	(0.8479)	(0.5618)			
1996	5	-0.0011	-0.1412**	-0.0019	0.0001	0.0622	-0.0013	0.0010	0.0482	0.0017	-0.0012	0.0873	-0.0011	0.0022	-0.0088	0.0001	0.0003			
		(0.7406)	(0.0356)	(0.1905)	(0.9741)	(0.3572)	(0.3886)	(0.6320)	(0.4774)	(0.2388)	(0.5624)	(0.2023)	(0.4368)	(0.2901)	(0.8950)	(0.9550)	(0.8888)			
1997	5	-0.0014	-0.2031***	0.0016	-0.0038	-0.2213***	0.0029	-0.0049	-0.0044	-0.0018	0.0056	-0.0900	-0.0025	0.0042	0.0155	0.0016	-0.0005			
		(0.8060)	(0.0024)	(0.4889)	(0.2854)	(0.0010)	(0.2330)	(0.1803)	(0.9477)	(0.4399)	(0.1200)	(0.1682)	(0.2793)	(0.2509)	(0.8101)	(0.4859)	(0.8910)			
1998	5	0.0019	-0.0045	-0.0029	-0.0014	-0.1297**	-0.0021	-0.0019	0.1273*	0.0022	0.0008	-0.1350**	0.0016	0.0021	0.0749	0.0025	-0.0039			
		(0.7170)	(0.9458)	(0.2056)	(0.6893)	(0.0487)	(0.3691)	(0.6163)	(0.0528)	(0.3533)	(0.8232)	(0.0387)	(0.4859)	(0.5625)	(0.2486)	(0.2752)	(0.2953)			
1999	5	0.0029	-0.0912	0.0009	-0.0005	-0.0499	0.0004	0.0006	0.0308	0.0029*	-0.0031	0.0515	-0.0035**	0.0035	0.0010	0.0021	-0.0035			
		(0.3241)	(0.1750)	(0.5987)	(0.8207)	(0.4547)	(0.8187)	(0.7911)	(0.6447)	(0.0886)	(0.1806)	(0.4393)	(0.0374)	(0.1227)	(0.9879)	(0.2117)	(0.1275)			
2000	6	0.0009	0.0275	0.0042**	-0.0021	0.0333	0.0002	0.0001	-0.1037	-0.0018	0.0054*	0.0573	0.0017	-0.0023	-0.0502	0.0006	-0.0024	-0.1265*	0.0027	-0.0043
		(0.7869)	(0.6840)	(0.0292)	(0.4806)	(0.6175)	(0.9071)	(0.9722)	(0.1193)	(0.3445)	(0.0591)	(0.3802)	(0.3816)	(0.4371)	(0.4489)	(0.7556)	(0.4201)	(0.0543)	(0.1638)	(0.1708)
2001	1	-0.0005	-0.1038	0.0003	-0.0021															
		(0.8336)	(0.1080)	(0.9087)	(0.6284)															
2002	5	0.0007	-0.0126	0.0008	0.0041	-0.0305	0.0013	-0.0031	0.0164	-0.0005	-0.0023	-0.0168	0.0017	-0.0015	0.0303	0.0013	-0.0008			
		(0.8395)	(0.8517)	(0.7687)	(0.3760)	(0.6504)	(0.6261)	(0.5006)	(0.8081)	(0.8399)	(0.6146)	(0.8036)	(0.5311)	(0.7367)	(0.6518)	(0.6244)	(0.8510)			
2003	1	0.0027	0.0302	0.0021	-0.0041															
		(0.2300)	(0.6399)	(0.3246)	(0.3399)															
2004	1	0.0003	-0.0131	0.0005	-0.0003															
		(0.8451)	(0.8392)	(0.7365)	(0.9136)															
2005	1	0.0008	0.0512	0.0000	0.0010															
		(0.3565)	(0.4308)	(0.9943)	(0.6425)															
2006	5	0.0076***	-0.0786	0.0029	-0.0040	-0.0405	-0.0011	-0.0003	0.0177	0.0023	-0.0070**	-0.0581	0.0007	-0.0064*	0.0759	0.0009	-0.0034			
		(0.0085)	(0.2349)	(0.1551)	(0.2178)	(0.5335)	(0.6233)	(0.9405)	(0.7887)	(0.2890)	(0.0383)	(0.3783)	(0.7324)	(0.0531)	(0.2483)	(0.6349)	(0.2837)			
2007	5	0.0014	0.0284	0.0045**	-0.0063**	0.0083	-0.0012	0.0013	0.0983	-0.0016	0.0002	-0.1323*	0.0006	-0.0013	-0.0476	0.0014	-0.0012			
		(0.5948)	(0.6737)	(0.0152)	(0.0301)	(0.9016)	(0.5323)	(0.6547)	(0.1393)	(0.4023)	(0.9358)	(0.0503)	(0.7358)	(0.6675)	(0.4796)	(0.4450)	(0.6778)			
2008	5	-0.0069	-0.0749	-0.0014	0.0062	-0.0861	0.0037	-0.0018	-0.0992	-0.0002	-0.0012	-0.0079	-0.0085*	0.0127*	-0.0594	0.0024	-0.0036			
		(0.2724)	(0.2650)	(0.7583)	(0.3388)	(0.1994)	(0.4140)	(0.7847)	(0.1390)	(0.9729)	(0.8610)	(0.9062)	(0.0634)	(0.0572)	(0.3710)	(0.5969)	(0.5878)			
2009	6	-0.0011	-0.0271	0.0021	-0.0028	0.0563	-0.0035	0.0037	-0.0607	0.0003	-0.0006	0.1019	-0.0014	0.0003	-0.0966	0.0013	0.0040	0.0990	0.0017	0.0006
		(0.7698)	(0.6853)	(0.3970)	(0.5007)	(0.3903)	(0.1583)	(0.3547)	(0.3503)	(0.9058)	(0.8721)	(0.1190)	(0.5845)	(0.9337)	(0.1414)	(0.6198)	(0.3056)	(0.1349)	(0.5015)	(0.8764)
2010	5	0.0010	0.0336	0.0021	0.0001	-0.1084	0.0018	-0.0036	0.1085	-0.0024	-0.0010	-0.0404	-0.0008	0.0031	-0.1221*	0.0000	-0.0028			
		(0.6896)	(0.6145)	(0.3300)	(0.9696)	(0.1059)	(0.4134)	(0.3153)	(0.1002)	(0.2624)	(0.7855)	(0.5402)	(0.7165)	(0.3894)	(0.0683)	(0.9826)	(0.4134)			
2011	5	0.0014	0.0058	-0.0006	0.0020	-0.0082	0.0024	-0.0065**	0.0198	-0.0009	-0.0020	-0.2052***	0.0016	-0.0038	0.0046	0.0014	-0.0018			
		(0.5065)	(0.9317)	(0.7770)	(0.5220)	(0.9023)	(0.2670)	(0.0404)	(0.7677)	(0.6670)	(0.5191)	(0.0023)	(0.4540)	(0.2370)	(0.9466)	(0.5187)	(0.5666)			
2012	5	0.0016	0.0214	-0.0026	0.0007	0.0724	0.0044**	-0.0050*	0.0973	0.0000	-0.0036	-0.0410	-0.0001	0.0019	0.0206	0.0000	-0.0008			
		(0.3338)	(0.7861)	(0.1195)	(0.8024)	(0.3592)	(0.0108)	(0.0822)	(0.2150)	(0.9874)	(0.2138)	(0.5966)	(0.9404)	(0.5090)	(0.7885)	(0.9948)	(0.7793)			

SESSION

PARALLEL PROGRAMMING AND NOVEL FRAMEWORKS + PARALLEL ARCHITECTURES AND SYSTEMS

Chair(s)

TBA

Hybrid CPU-GPU Pipeline Framework PDPTA'14

Fahad Khalid*, Frank Feinbube, Andreas Polze
Hasso Plattner Institute for Software Systems Engineering
14482 Potsdam, Germany
fahad.khalid, frank.feinbube, andreas.polze@hpi.uni-potsdam.de

Abstract— *The pipeline pattern for parallel programs is utilized in a wide array of scientific applications designed for execution on hybrid CPU-GPU architectures. However, there is a dearth of tools and libraries to support implementation of pipeline parallelism for hybrid architectures.*
We present the Hybrid Pipeline Framework (HyPi) that is intended to fill this gap. HyPi provides high level abstractions in C++ for implementation of pipelines on hybrid CPU-GPU architectures. It is a generic framework intended to support a wide range of applications. The complexities characteristic of such implementations, e.g., partitioning of input/output data structures, asynchronous memory transfer, communication between CPU and GPU etc., are handled by the framework and are therefore hidden from the developer. HyPi exposes certain degrees of freedom that can be tuned to optimize the performance of a simulation based on application specific requirements. We present a detailed account of the framework design, and evaluate the framework performance using a real-world application from the domain of computational biology. Results show that HyPi performs on par with a custom-tailored, hand-tuned pipeline implementation for the given application.

Keywords: Hybrid pipeline, heterogeneous computing, pipeline parallelism, Intel Threading Building Blocks, CUDA

1. Introduction

The advent of the CUDA programming model marked the emergence of mainstream application of accelerator programming to scientific computing. Over the years, the said programming model has evolved significantly; supporting a substantial number of advanced features. A large number of algorithms have been ported to the GPU architecture; several of which are available in CUDA based libraries [1], [2]. Yet, even today, programming a GPU costs much more in terms of productivity than programming a CPU for the same problem. Therefore, even though applications capable of execution on a GPU may benefit in terms of speedup, the effort required to engineer such applications reduces programmer productivity. In short, the decision of whether or not to port an application to the GPU must balance the trade-off between application performance and developer productivity. We term this trade-off the *productivity vs. performance trade-off.*
The *pipeline* pattern [3], [4] for parallel programming covers a broad range of applications for which the *productivity vs. performance trade-off* is evident. A commonly recurring application in hybrid CPU-GPU architectures is the overlap of computation and transfer of memory from GPU to CPU (and vice versa). Other applications include the execution of different stages of a program on either CPU or GPU, in order to properly utilize all processing resources [5], [6].
Even though the CUDA programming model provides features that make it possible to implement a cross-device pipeline, e.g., using CUDA *streams* for asynchronous memory transfer, the implementation details render the process rather cumbersome. It is conceivable that if the process of implementing cross-device pipelines is simplified, researchers will be incentivized to explore the potential of implementing new algorithms using the hybrid CPU-GPU pipelining approach.
In this paper we present our *Hybrid Pipeline Framework (HyPi)* that is designed to provide high level abstractions for implementing the pipeline pattern on hybrid architectures. It is a generic framework intended to support a wide range of applications that can benefit from a hybrid pipeline. *HyPi* hides much of the intricacies inherent in implementing such a pipeline, while exposing enough degrees of freedom so that the pipeline performance can be optimized for each individual application. Moreover, we believe that *HyPi* can serve as a test bed for assessing the feasibility of implementing certain algorithms using the hybrid pipelining approach. It is important to note that *HyPi* is available as an external C++/CUDA library, and does not require any additional compiler support. The library has been tested with both GCC and the Intel C++ compiler.
The paper is organized as follows: Section 1.1 presents an overview of related work, where different applications and available frameworks for hybrid pipelining are discussed. Section 1.2 highlights the research gap and our contribution. A detailed account of framework design is presented in Section 2. Application of *HyPi* to a computational biology simulation is presented in Section 3. This is followed by comparative evaluation of *HyPi* against a custom-tailored pipeline in Section 4. Finally, Section 5 concludes the paper with a discussion of future work.

1.1 Related Work

Software pipelining has been around as a concept in computer science for a long time. A rigorous survey of various methods for software pipelining is presented in [7]. In parallel computing, pipelining has been identified as a commonly occurring pattern [8], and has therefore been the subject of study for many a research project.
A thorough literature review of the use of pipelining in hybrid computing has revealed that pipelining is commonly utilized in three different situations: 1) Overlapping computation and transfer of data between CPU memory and GPU memory; 2) execution of different stages of a program on either CPU or GPU based on which architecture is better suited to the computation; and, 3) execution of different stages of a program on either CPU or GPU for load balancing and optimal resource utilization. Following is a selection of works that utilize pipelining in one or more of the above mentioned situations:
A *Pipelined Multi-GPU MapReduce (PMGMR)* implementation is presented in [9] where GPU acceleration is extended to multiple GPUs. In this work, the primary application of pipelining is to overlap computation and communication in order to reduce the communication overheard. The implementation also makes it possible to process datasets that exceed both CPU and GPU memory capacity.
In order to harness the power of GPU clusters for *MapReduce*, a library has been developed [10]. This work focuses on tackling the challenges of data movement between GPUs, managing out-of-core data on GPUs, as well as modifying *MapReduce* in order to leverage the GPU cluster architecture. The pipelining concept is utilized in terms of overlap of computation and communication.
The problem of efficient scheduling of *MapReduce* tasks on a coupled CPU-GPU chip is dealt with in [5]. Two different approaches are presented; 1) dynamically dividing *Map* tasks onto both CPU and GPU, and 2) pipelining *Map* and *Reduce* tasks between GPU and CPU. Empirical evidence is provided to show that a pipelining solution where *Map* is implemented on the GPU and *Reduce* is implemented on the CPU justifies the use of pipelining for *MapReduce*.
Moim [11] is a *MapReduce* framework for Multi-GPU systems that implements a 3-stage pipeline consisting of *input split*, *map* and *merge* phases. The *input split* and *merge* phases are executed on the CPU, while the *map* phase is executed on the GPU. The *reduce* phase can be executed simultaneously on GPU and CPU depending on whether or not the GPU memory can hold the entire partition to be reduced.
To summarize, in order to improve the performance of the *MapReduce* model, two types of GPU acceleration methods have been employed. In the first approach, both *Map* and *Reduce* are implemented as GPU kernels. In the second approach, *Map* is implemented on the GPU, while *Reduce* is implemented on the CPU. Justification for the second approach lies in the fact that the modern GPU architecture is particularly suitable for data parallel applications that employ the *Map* pattern [4]. In such an application, multiple instances of the *Map* function can be processed in parallel by many processing elements; thereby maximizing the utilization of processing resources on the GPU. However, a parallel implementation of the *Reduce* pattern processes data in such a way that the resource utilization (in terms of processing elements) follows a tree like pattern, i.e., the number of required processing elements (or the *degree of parallelism*) decreases over time. Therefore, utilization of the processing resources is not consistent throughout the function. Even though *dynamic parallelism* [12] can be used to improve resource utilization [13], kernel design becomes exceedingly complicated.
Hybrid CPU-GPU pipelining is not limited to *MapReduce*. A 3-stage CPU-GPU pipeline for eigenvector and eigenvalue determination is presented in [6]. The work establishes the significance of utilizing a hybrid pipeline for optimal resource utilization and presents a stochastic queue monitoring strategy for parallel applications based on the pipeline pattern.
The concept of hybrid CPU-GPU pipelining has also been applied in computational biology, where the problem of enumeration of elementary flux modes in metabolic networks was parallelized using an OpenMP and CUDA based solution [14].
The capability to implement a hybrid CPU-GPU pipeline has been introduced within the context of the FastFlow [15] framework. The FastFlow framework was extended by introducing an abstraction for creating and executing a pipeline with user-defined stages. A GPU-enabled linear algebra library can be called from within a stage, making it a GPU execution stage. Therefore, different stages of the pipeline can execute on either the CPU or the GPU.

1.2 Research Gap and Our Contribution

Most of the above mentioned related work concentrates on either utilizing pipelining for a specific programming model such as *MapReduce*, or a particular application. Therefore, most of these works lack generalizability. In contrast, the framework described in [15] is much more general and useful for a larger number of applications.
The framework we present in this paper is not limited to any specific application or a programming model such as *MapReduce*. It is a generalized framework similar to the FastFlow pipeline presented in [15], and provides features that overcome certain limitations of the FastFlow pipeline. Our *Hybrid Pipeline Framework (HyPi)* is built on the pipeline functionality provided by a widely used and robust parallel programing model, i.e., *Intel Threading Building Blocks* [16]. *HyPi* provides the following features that stand out in comparison to other frameworks:

- Automated management of CUDA *streams* and *events* for asynchronous *CPU-to-GPU* and *GPU-to-CPU* memory transfer.
- Automated management of communication between GPU and CPU using the *callback* functionality introduced in CUDA 5.0.
- The possibility of multi-threaded execution of CPU stages of the pipeline.
- Automated partitioning of input/output data structures.

In addition to presenting a detailed account of the features mentioned above, this paper provides a thorough analysis of the major issues in implementing such a generic framework with CUDA. To the best of our knowledge, no other framework provides all the features available in our *Hybrid Pipeline Framework (HyPi)*.

2. Framework Design

In the rest of the document, we refer to the CPU as *Host*, and the GPU as *Device*.

2.1 Overall Pipeline Design

One of the major *HyPi* design objectives is to provide the programmer with a familiar interface for pipeline implementation. Based on our positive experience with *Intel Threading Building Blocks (TBB)* [16], we decided to use TBB as the foundation for the pipeline. TBB is a free and open source multithreading library from Intel, which provides a reliable and efficient abstraction (*tbb::pipeline*) for pipeline implementation on multicore CPUs. Each stage of a pipeline is implemented as a C++ class that inherits *tbb::filter*.

There are two possible ways in which *HyPi* can be used to facilitate the implementation of a hybrid pipeline: 1) using *HyPi* stages, and 2) using custom-tailored stages.

When using *HyPi* stages, the programmer does not need to implement the pipeline stages. Instead, classes predefined in the framework are used. The following types of stages are pre-implemented in the framework:

- *DeviceFilter:* This class represents a pipeline stage that executes a CUDA *Device* kernel. It automatically handles partitioning of input/output data structures, asynchronous memory transfer to and from the *Device*, and *callback* mechanism required to inform the following pipeline stage of the completion of *Device-to-Host* result data transfers. Details of these features are covered in Section 2.2.
- *CallbackFilter:* This stage usually follows the *DeviceFilter* and encapsulates the *Device-to-Host* communication that takes place using *callbacks*. In TBB, a typical pipeline stage proceeds with executing its function as soon as it receives a token from the previous stage. The token generation function in *DeviceFilter* and initiation of memory transfer work asynchronously. Therefore, the stage that follows *DeviceFilter* must not just wait for the token, but also the corresponding memory transfer confirmation. The *CallbackFilter* hides this mechanism from the programmer and passes the token to the next stage only once the corresponding memory transfer is complete.
- *PostProcessFilter:* This class represents a stage that is required to receive result data from the *Device* and process it on the *Host*.

For the above mentioned automation procedure to work, the programmer is required to register the signature of the *Device* kernel to be used, and a post-processing function to be called from within the *PostProcessFilter*. This process is similar in principle to the *MapReduce* programming model where the programmer specifies that *Map* and *Reduce* functions, and the rest is taken care of by the framework.

One or more custom-tailored pipeline stages can also be provided by the programmer as C++ classes that extend *tbb::filter*. Either the entire pipeline can be constructed in this fashion, or a pipeline with *HyPi* stages can be extended to include more stages. For a pipeline that does not use any of the *HyPi* stages, *Device* programming can be simplified by using a C++/CUDA library provided in *HyPi* that exposes functions for automated data partitioning, automated kernel launch for each partition, and simplified abstractions for asynchronous data transfer from *Host* to *Device* and *Device* to *Host*. In fact, *HyPi* stages are primarily C++ classes that encapsulate these functions in an organized manner. Example of a custom-tailored pipeline initialization is presented in Figure 1.

Any stage intended for execution on the *Device*, e.g., *DeviceFilter* (as well as *CallbackFilter*), must always operate in the *serial in-order* mode. This is because parallelism is achieved by the *Device* kernel itself, and does not require multiple *Host* threads. *Host* bound stages however, can operate in *parallel* mode as well.

```
tbb::pipeline pipeline;

CallbackFilter myCbFilter;
DeviceFilter myDevFilter(... params ...,
                         myCbFilter);
PostProcessFilter myPProcFilter(... params ...,
                         myDevFilter);

numPartitions = myDevFilter.getNumPartitions();
myCbFilter.initTrasferredFlags(numPartitions);

pipeline.add_filter(myDevFilter);
pipeline.add_filter(myCbFilter);
pipeline.add_filter(myPProcFilter);

pipeline.run(numTokensInFlight);
pipeline.clear();
```

Fig. 1: A custom 3-stage pipeline initialization using *HyPi* stages (backend code).

2.2 Automation Design Challenges

The following subsections describe design considerations for each of the major features provided by *HyPi*.

2.2.1 Partitioning

In hybrid architectures, *Device* memory is generally much smaller than the *Host* memory. Given this limitation, one important design assumption in *HyPi* is that the input/output data structures must be partitioned to ensure that they fit into the *Device* memory. Since *HyPi* is a library, the information about which data structures are required as arguments by the user-defined kernel, along with the corresponding sizes of these data structures, must be provided by the programmer. Similar to OpenCL [17], each kernel argument is registered with the framework as part of the initialization code. If necessary, the size of the output data structure can be specified as a function of input size. This function must also be provided by the programmer.

This information is then used by the framework to determine the maximum partition size for each data structure. In order to determine the partition size, three factors are evaluated: 1) total *Device* memory required by all data structures, as well as the total *Device* memory available; 2) impact of partition size on pipeline performance due to memory transfer between *Host* and *Device*; and 3) efficient utilization of *page-locked* [12] *Host* memory. The user can configure factors 2 and 3 by specifying upper limits for partition size and *page-locked Host* memory. Since a small upper limit (i.e., a value which is much smaller than the total *Device* memory available) on partition size may allow multiple partitions to reside simultaneously in the *Device* memory, each set of such partitions is bundled together as a *Segment*. The segment size is controlled by the upper limit on *page-locked* memory. The use of *page-locked* memory is necessitated by the fact that the framework uses asynchronous CUDA calls for all *Host-to-Device* and *Device-to-Host* memory transfers.

Note: The first prototype supports only equal sized linear data structures for automated partitioning. However, at the time of this writing, sophisticated algorithms are being implemented to support a wider range of possibilities.

It is important to note that *page-locked Host* memory is a scarce resource. Therefore, in order to keep a good number of tokens in flight, it is necessary to employ *multiple-buffering* [18] for data structures that use *page-locked* memory. This way, processing of a single partition results in the occupation of only part of the *page-locked* memory. The next partitions in line can use the remaining portions. In the meantime, the *PostProcessFilter* can release the *page-locked* memory occupied by the first partition and make it available for use by further partitions. The *multiple-buffering* solution is implemented in *HyPi*.

2.2.2 Kernel

For *HyPi* to be able to automatically call the user-defined CUDA kernel, the kernel must be registered with the framework. As mentioned in Section 2.2.1, kernel arguments must also be registered. However, CUDA kernel configuration parameters – such as maximum grid dimension and threads per block – can be configured by the user.

2.2.3 Stream and Event Management

Once all the required information is available to the framework, for each segment, the kernel processes all partitions. Depending on the problem size, processing each partition may require launching multiple grids. For each grid, separate streams and events are used to ensure efficient asynchronous memory transfer between *Host* and *Device*. All these steps are handled by the framework.

2.2.4 Device-Host Event Communication

Once a partition has been processed by the *Device*, output generated by the kernel is transferred from *Device* to *Host* asynchronously. This means that the *Device* can start processing the next partition without waiting for the *Device-to-Host* transfer to complete. Therefore, the *DeviceFilter* may issue tokens for which the *Host* has not yet received the result data. In order to make sure that the *Host* is aware of when the data transfer is complete, *callback* functionality introduced in CUDA 5.0 is used. As soon as the data transfer is complete, a *callback* function is executed that updates the state of the *CallbackFilter* stage. This update is correlated with the token received from the *DeviceFilter*. Therefore, at this point, the *CallbackFilter* knows that the process is complete and sends out a token to the next stage.

3. Use Case

In this section, we present the application of *HyPi* to the problem of enumerating elementary flux modes in metabolic networks.

The process of metabolism comprises of chemical compounds, called *metabolites*, transformed into other chemical compounds through *reactions* catalyzed by enzymes. A group of such related *metabolites* and *reactions* can be viewed as a network, modeled mathematically as a node-weighted directed hypergraph. A node in such a hypergraph stands for the number of molecules of a particular metabolite; while a directed hyper-edge represents a reaction. *Elementary Flux Modes (EFMs)* are minimal subnetworks that operate at equilibrium. Removal of any component results in the EFM being unusable. In order to use EFMs to characterize the behavior of a metabolic system, it is required to enumerate all EFMs in the system. A commonly used algorithm for EFM enumeration is the *Nullspace* algorithm [19]. For a detailed description of the algorithm, we refer the reader to [20]. Here we summarize the main steps of the algorithm:

1) The *Nullspace* algorithm operates on the *Stoichiometic matrix*. Rows in the stoichiometric matrix correspond to *metabolites*, and the columns correspond to *reactions*. The matrix can be viewed as the *incidence matrix* corresponding to the hypergraph that represents the metabolic network to be analyzed. The stoichiometric matrix is compressed [21] in order to improve performance.
2) An underdetermined system of homogeneous linear equations is solved to obtain the *nullspace*, where the stoichiometric matrix is the coefficient matrix. The *nullspace* is permuted to simplify further operations. The permutation results in two parts of the matrix: $R^{(1)}$ and $R^{(2)}$.
3) For each row in $R^{(2)}$:
   a) Algebraic combinations are generated for selected columns in $R^{(2)}$ and bitwise combinations are generated for the corresponding parts in $R^{(1)}$
   b) Duplicate candidates are removed
   c) Each candidate is verified for elementarity
   d) The verified candidates are appended as column vectors to the *nullspace*

A row in $R^{(2)}$ that has been processed is converted into a binary representation, and moved to $R^{(1)}$.

Due to its combinatorial nature, the candidate generation phase is extremely expensive both in terms of computation and memory. Even for small to medium sized networks, parallel computation is a necessity.

3.1 Combinatorial Candidate Generation

The *combinatorial candidate generation* algorithm refers to the generation of bitwise combinations in $R^{(1)}$. Figure 2 describes the pseudocode. $R^{(1)}$ is split into two bit matrices, **X** and **Y**. A candidate vector is generated by performing a *bitwise OR* operation between a column in **X** and a column in **Y**, and then performing a *popcount* operation on the result vector. Indices corresponding to the operand columns in **X** and **Y** are stored, marking the result vector as a candidate for elementarity testing if the *popcount* is greater than a certain threshold value λ (determined elsewhere in the *Nullspace* algorithm).

In previous work [14], we developed a hybrid pipeline based parallel solution to the combinatorial candidate generation problem. There we implemented a pipeline using OpenMP and CUDA that was tightly coupled with, and optimized for the given problem. In our current work, we present an implementation based on *HyPi* stages and compare the performance of the two implementations. There are three reasons for choosing this application: 1) The pipeline pattern is not obviously applicable to the parallelization problem at hand, and represents a class of problems where an efficient pipeline implementation is not straightforward; 2) The nature of this algorithm is different from those for which pipeline implementations are typically used (such as those mentioned in Section 1.1), and therefore presents additional challenges; and 3) We can compare *HyPi* performance against a custom-tailored and optimized pipeline for a complex algorithm.

```
Input : Matrices: $X$, $Y$ – Vectors: $indX$, $indY$
        Integer: $\lambda$
Output: Ordered pairs of column indices:
        $\{(x, y) \mid x \in indX \text{ and } y \in indY\}$
1 foreach colX: column in X do
2     foreach colY: column in Y do
3         candidate = $colX \vee colY$;
4         numNonZeros = popcount($candidate$);
5         if $numNonZeros \leq \lambda$ then
6             store pair ($indX[colX]$, $indB[colY]$);
7         end
8     end
9 end
```

Fig. 2: Combinatorial candidate generation algorithm. $indX$, $indY$ contain column indices of **X** and **Y** respectively; $\vee$ is the bitwise OR operation; λ is a threshold value (as described in Section 3.1).

The hybrid pipeline implementation of combinatorial candidate generation (as described in [14]) divides the algorithm into two phases: 1) *Generate*; and 2) *Map*. The *Generate* phase is implemented as a CUDA *Device* kernel. It is responsible for generating all candidates, computing the *popcount* values, and processing the condition to verify if the result vector should be kept for elementariy testing. Results from the *Generate* phase are stored in a bit array. The *Map* phase takes this bit array and maps the results to the corresponding column indices in **X** and **Y**. The *Map* phase is implemented as a post-processing step computed on the *Host*.

3.2 HyPi Implementation of the Candidate Generation Pipeline

The *HyPi* implementation of the combinatorial candidate generation algorithm is done using *HyPi* stages. Just the three pre-defined *HyPi* stages are used, i.e., *DeviceFilter*, *CallbackFilter* and *PostProcessFilter*. First, the kernel signature is registered with the framework, so that the *DeviceFilter* can automatically call the kernel. Then the *Map* phase is registered with the framework as a post-processing function. In this case, we chose to implement the *Map* phase as a multithreaded function parallelized using OpenMP. This is to show that even though *HyPi* uses Intel TBB for pipeline parallelism, it does not imply that the entire program must be dependent on Intel tools only.

The custom-tailored pipeline from the previous implementation [14] was encapsulated in what we call the *Maximum Resource Utilization Framework (MRU)*. *MRU* is designed to utilize all available *Host* threads when running a pipeline. This is done by having (in addition to the pipeline implementation) a *Host*-only multithreaded version of the given algorithm. We have an OpenMP parallel version of the combinatorial candidate generation algorithm. *MRU* divides the input into two parts. One part is processed by the *Host*-only parallel implementation, and the other part is processed by the pipeline implementation. This way, no *Host* threads are idle while the algorithm is executed. The *HyPi* implementation of the candidate generation algorithm also utilizes *MRU* as a harness.

4. Evaluation

We compare *HyPi* performance against: 1) a serial implementation of the candidate generation algorithm as available in *ElMo-Comp* [22], 2) an OpenMP based *Host*-only parallel implementation, and 3) the custom-tailored pipeline executed inside *MRU*. All our implementations are based on the *ElMo-Comp* code base.

4.1 Test Environment

The machine used for performance comparison consists of an Intel Nehalem EX architecture based quad-core Xeon E5520 CPU with 4 cores, where each core supports 2 hardware threads. The machine is equipped with 17GB of RAM. In addition to the CPU, the machine has an NVIDIA GTX 680 GPU with 4GB of device memory, supporting compute capability 3.0. All tests were carried out with CUDA driver version 5.5. However, it is possible to use the framework with CUDA 5.0, which is the earliest version supporting the *callback* functionality. The operating system used for the experiments is Ubuntu 12.04 LTS. The code was compiled using GCC 4.6.3 and NVCC 5.5.

4.2 Results

Given the machine available (as described in Section 4.1), it was not possible to conduct performance comparisons using real biological networks. This is because even for smaller networks, memory requirements are too high, and only three or four networks can be evaluated. This results in a very small sample against which performance comparisons can be done. In order to have a larger number of network samples, datasets of controlled sizes were artificially generated. Moreover, since we are concerned only with the combinatorial candidate generation part of the *Nullspace* algorithm, our measure of network size is the number of candidate vectors generated during the execution of the candidate generation algorithm.
Figure 3 presents a comparison of the serial, *Host*-only parallel and *HyPi* versions of the code. The plot indicates that as the number of candidates increases, the margin with which *HyPi* outperforms the other implementations gets wider. This behavior is expected, since it was shown in [14] that a pipeline implementation coupled with *MRU* is superior to the other implementations.

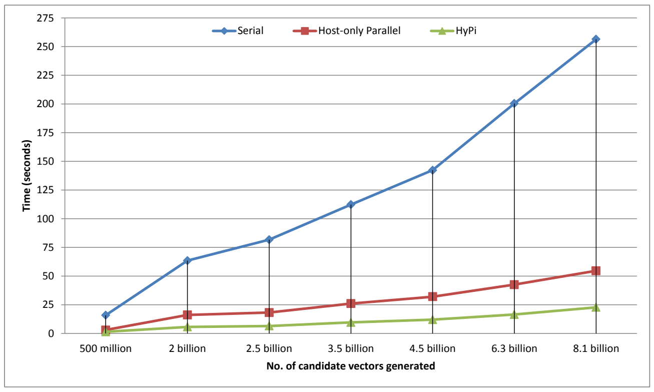


Fig. 3: Performance comparison between serial, *Host*-only parallel and *HyPi* implementations.

Figure 4 plots the performance of the custom-tailored OpenMP based pipeline implementation (designed during previous work [14]) with the *HyPi* implementation. As the plot depicts, *HyPi* performance is on par with the custom tailored pipeline. In fact, *HyPi* performance is slightly better. Although the difference is only marginal, we thought it necessary to investigate the cause.
Even though most of the pipeline design features are shared among the two implementations, management of *tokens in flight* is different. For *HyPi*, this is handled by the underlying TBB framework. In *tbb::pipeline*, the number of tokens in flight is specified by the programmer. The *HyPi* results plotted in Figure 4 are based on pipeline execution with two tokens in flight (if we change the number of tokens in flight to one, the resulting *HyPi* performance is poorer than the custom-tailored pipeline, which is expected because a single token essentially serializes the pipeline). The custom-tailored pipeline on the other hand has tightly-coupled stages without an explicit notion of tokens in flight. We believe that the performance improvement seen in *HyPi* is due to a superior token management strategy implemented in Intel TBB.

5. Conclusion and Future Work

5.1 Summary

We have presented the *Hybrid Pipeline Framework (HyPi)* intended to simplify the process of implementing pipeline parallelism in hybrid CPU-GPU architectures. The framework is implemented in C++ using Intel TBB and NVIDIA CUDA, and is suitable for pipeline applications where some stages execute on the CPU, while others execute on the GPU. *HyPi* exposes pre-developed stages as well as library routines that automate the processes of data partitioning, asynchronous data transfer from CPU-to-GPU and GPU-to-CPU, *callback* mechanism for communication between GPU and CPU, as well as automated execution of the CUDA

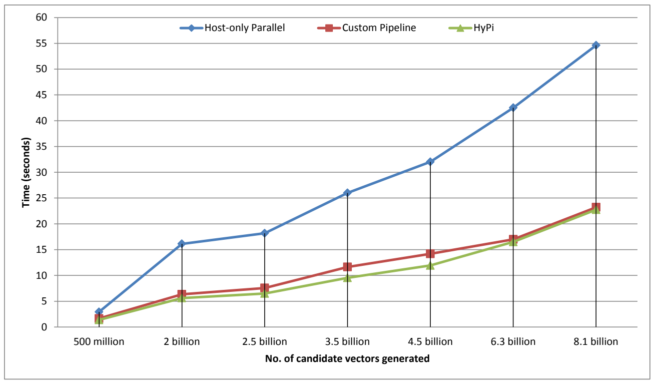


Fig. 4: Performance comparison between *Host*-only parallel, custom-tailored pipeline and *HyPi* implementations.

kernel over multiple data partitions. We have evaluated the performance of the framework against a real-world application from computational biology, and shown that it performs on par with a custom-tailored pipeline for the same problem.

5.2 Discussion

In the future, we intend to combine *HyPi* and *MRU* (mentioned in Section 3.2). At the moment, static load balancing provided by *MRU* cannot be used within the pipeline, i.e., an individual pipeline stage cannot be replicated on both the CPU and GPU for load balancing. Instead, the entire pipeline has to be replicated as CPU code and executed in parallel using OpenMP threads. Moreover, we believe it is important to introduce *dynamic load balancing*, since not all workloads can be dealt with using static schemes.

Similarly, the automated partitioning algorithms will be extended to support more complex data structures. At the moment only linear data structures are supported. Once these features have been implemented, we intend to test the framework with other scientific simulations that present different challenges in terms of hybrid pipelining.

In Section 1, we introduced the term *productivity vs. performance trade-off*. This term encapsulates our long-term research objectives. Our hypothesis is that various characteristics of a computational kernel such as *Degree of Parallelism*, *Arithmetic Intensity*, *Degree of Control Divergence* etc., make the kernel suitable for execution on either the CPU or an accelerator such as the GPU. We conjecture that a scientific simulation can be broken down into multiple computational kernels, where each kernel constitutes a stage in a hybrid pipeline. The assignment of stages to different execution architectures will depend on the above mentioned and certain other characteristics. Therefore a scientific simulation could be executed as a hybrid pipeline. Work is in progress to investigate this hypothesis, and results will be presented in a later publication.

References

[1] J. R. Humphrey, D. K. Price, K. E. Spagnoli, A. L. Paolini, and E. J. Kelmelis, "CULA: hybrid GPU accelerated linear algebra routines," pp. 770 502–770 502–7, 2010.

[2] NVIDIA, "CUBLAS Library," User Guide DU-06702-001_v5.5, July 2013.

[3] T. Mattson, B. Sanders, and B. Massingill, *Patterns for Parallel Programming*. Addison-Wesley Professional, 2004.

[4] M. McCool, J. Reinders, and A. Robison, *Structured Parallel Programming: Patterns for Efficient Computation*. Elsevier, 2012.

[5] L. Chen, X. Huo, and G. Agrawal, "Accelerating MapReduce on a coupled CPU-GPU architecture," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 2012, p. 25.

[6] M. T. Garba and H. González-Vélez, "Asymptotic peak utilisation in heterogeneous parallel CPU/GPU pipelines: A decentralised queue monitoring strategy," *Parallel Processing Letters*, vol. 22, no. 02, p. 1240008, 2012.

[7] V. H. Allan, R. B. Jones, R. M. Lee, and S. J. Allan, "Software pipelining," *ACM Comput. Surv.*, vol. 27, no. 3, pp. 367–432, Sept. 1995.

[8] J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, and A. White, Eds., *Sourcebook of Parallel Computing*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[9] Y. Chen, Z. Qiao, S. Davis, H. Jiang, and K.-C. Li, "Pipelined multi-GPU MapReduce for Big-Data processing," in *Computer and Information Science*, ser. Studies in Computational Intelligence, R. Lee, Ed. Springer International Publishing, 2013, vol. 493, pp. 231–246.

[10] J. Stuart and J. Owens, "Multi-GPU MapReduce on GPU clusters," in *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, May 2011, pp. 1068–1079.

[11] M. Xie, K.-D. Kang, and C. Basaran, "Moim: A multi-GPU MapReduce framework."

[12] NVIDIA, "CUDA C programming guide," Design Guide PG-02829-001_v6.0, February 2014.

[13] D. Merrill and A. Grimshaw, "High performance and scalable radix sorting: A case study of implementing dynamic parallelism for GPU computing," *Parallel Processing Letters*, vol. 21, no. 02, pp. 245–272, 2011.

[14] F. Khalid, Z. Nikoloski, P. Tröger, and A. Polze, "Heterogeneous combinatorial candidate generation," in *Euro-Par 2013 Parallel Processing*, ser. Lecture Notes in Computer Science, F. Wolf, B. Mohr, and D. Mey, Eds. Springer Berlin Heidelberg, 2013, vol. 8097, pp. 751–762.

[15] M. Goli, M. Garba, and H. González-Vélez, "Streaming dynamic coarse-grained CPU/GPU workloads with heterogeneous pipelines in FastFlow," in *High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*, June 2012, pp. 445–452.

[16] J. Reinders, *Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism*. O'Reilly Media, Inc., 2007.

[17] K. O. W. Group, "The OpenCL specification, Standard Specification," December 2011.

[18] D. E. Knuth, *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1997.

[19] C. Wagner, "Nullspace approach to determine the elementary modes of chemical reaction systems," *The Journal of Physical Chemistry B*, vol. 108, no. 7, pp. 2425–2431, 2004.

[20] D. Jevremović, C. T. Trinh, F. Srienc, and D. Boley, "On algebraic properties of extreme pathways in metabolic networks," *Journal of Computational Biology*, vol. 17, no. 2, pp. 107–119, 2010.

[21] J. Gagneur and S. Klamt, "Computation of elementary modes: a unifying framework and the new binary approach," *BMC bioinformatics*, vol. 5, no. 1, p. 175, 2004.

[22] D. Jevremović, C. T. Trinh, F. Srienc, C. P. Sosa, and D. Boley, "Parallelization of nullspace algorithm for the computation of metabolic pathways," *Parallel Computing*, vol. 37, no. 6-7, pp. 261–278, 2011.

Implementing MPI_Barrier with the NetFPGA

O. Arap[1], G. Brown[2], B. Himebaugh[2], and M. Swany[1]
[1]Center for Research in Extreme Scale Technologies, Indiana University, Bloomington, IN, USA
[2]School of Informatics and Computing, Indiana University, Bloomington, IN, USA

Abstract— *Parallel programs written using the standard Message Passing Interface (MPI) frequently depend upon the ability to synchronize execution using a barrier. Barrier synchronization operations can be very time consuming. As a consequence, there have been investigations of custom interconnects and protocols for accelerating this operation and other collective operations in parallel MPI programs.*

In this paper, we explore the use of hardware programmable network interface cards utilizing standard media access protocols as an alternative to fully custom synchronization networks. Our work is based upon the NetFPGA – a programmable network interface with an on-board Virtex FPGA and four Ethernet interfaces. We have implemented a network-level barrier operation using the NetFPGA for use in MPI environments. This paper compares the performance of this implementation with MPI over Ethernet for a small configuration.

Keywords: NetFPGA, MPI, MPI_Barrier, Synchronization, Collective Operations

1. Introduction

Barrier synchronization can have a significant performance impact on programs running on large parallel processors. A barrier is a logical delimiter for participating processes to ensure that all the processes are at the barrier point in their execution sequence [6]. A participating process may continue with its execution after it receives a release notification either from one of its peers or after an appropriate set of peer message exchanges indicates that all the participating processes have called the barrier. Regardless of the task parallelization, the barrier is a sequential blocking call for all the processes. It introduces a latency completely depending on the execution sequences of other processes, underlying communication infrastructure and the logic of the barrier implementation.

In the past years, many proposals have been presented to reduce the latency of barrier synchronizations. They are classified as software solutions, hardware solutions and hybrid solutions that involve both hardware and software aspects. Software barrier proposals are largely independent of underlying hardware technology [12] [19]. They tend to be implemented using generic solutions that can be applied to different platforms by just changing the calls in the user level library implementation. Software solutions lack the performance of hardware and hybrid solutions due to the fact that software solutions are inherently limited by the hardware, which is not necessarily optimized to implement barrier logic.

Hardware based and hybrid solutions are typically proposed for specific target platforms such as parallel machines with custom interconnects, clusters of FPGAs with a specific communication medium, parallel machines with specific target topologies, etc [11] [14] [5] [20] [15] [7] [18] [1] [8]. However, not all researchers have access to special purpose parallel machines. As a result, many researchers build their own cluster using Commodity Off-the-Shelf (COTS) hardware. This is an active area of research focused on clusters of workstations, which can be constructed using Commodity Off-the-Shelf (COTS) processors and hardware to achieve high performance parallel execution.

This work is focused on investigating how programmable hardware platforms such as the NetFPGA [13] can be utilized to implement barriers. The NetFPGA has become a standard platform for learning and implementing networking hardware in academic research. To the best of our knowledge, this is the first attempt to utilize the NetFPGA in the implementation of barrier synchronization. The NetFPGA platform has been widely used to prototype networking hardware with the goal of reducing the performance costs by offloading some specific tasks to the hardware level. It has standardized interfaces between hardware modules and software level access to the hardware modules.

It is difficult to claim that our hardware barrier implementation using the NetFPGA bests all the other hardware barrier solutions in terms of performance since we do not have access to all the competitive technologies, and it is not our goal with this work. However, lowering the barrier logic into the hardware provides significant performance benefits compared to software based implementations, and we will show that our implementation using the NetFPGA does not conflict with this assumption. There are several proposals that target different FPGA platforms, which either implement the entire system on chip, or utilize single FPGA as a separate networking device such as NIC or switch. However, our design is different since the NetFPGA provides implementation standards with a specific development suite. We completely utilize the NetFPGA development environment, and thus leverage its extensibility for future functionality.

The remainder of this paper is organized as follows: Section 2 summarizes the design goals. Section 3 outlines the implementation details and architectural design. Section

4 presents performance evaluation of our design. Section 5 provides some background and discusses related work. Section 6 offers discussion about our work and how it could be extended in the future, and finally Section 7 concludes the paper.

2. Design Goals

In this paper, we propose a barrier synchronization framework utilizing the standard infrastructure from the NetFPGA platform and using standard protocols such as UDP, IP and Ethernet. The unique contributions of our work are as follows:

- The design relies on the standard NetFPGA driver and there is no need to change anything in the OS. We incorporate some simple changes in the user-level code, utilizing the Open MPI [2] library to generate the packets that the NetFPGA recognizes and processes.
- All of our additional hardware modules live in the user-data-path [4], as recommended by the NetFPGA user community. Therefore, it is self-describing and could be extended by someone who is familiar with the NetFPGA environment.
- We enable a flexible topology that could be created by connecting different ports of the NetFPGA directly to each other. The current implementation supports four distinct physical topologies.
- We are providing a framework that can be easily extended to other types of MPI collective operations. We began with the barrier implementation as our base.
- Our work does not require a separate control network for barrier synchronization as it can perform the synchronization on the network where the data also flows.

3. Implementation

Our FPGA node design is derived from the *reference NIC* implementation distributed with the NetFPGA package. The host communicates with our synchronization engine through a UDP socket – operating system support for such sockets is part of the standard package. The NF_Barrier implementation consists of sending a specially crafted UDP message, and then blocking until a barrier release message is received. An added feature of building our implementation upon the NetFPGA reference NIC is that our node maintains the ability to forward standard IP packets.

The simplicity of the host interface belies the complex task that the barrier node must perform. The barrier tracks outstanding requests by storing the various MAC, IP addresses, checksum and UDP header fields. These are later used to generate a message to release the host from the barrier. The generated release packet must arrive user-space travelling up to the protocol stack. Therefore, it must be properly formed, so that none of the layers prevent packet to be processed by the application layer.

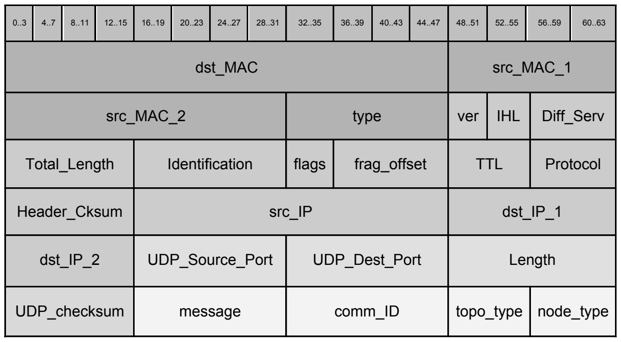


Fig. 1: Fields and structure of an actual NF_Barrier packet

3.1 Packet Format

Our design is intended to support a variety of topologies. We use the packet format presented in Figure 1 to inform the underlying synchronization hardware about the current topology. The synchronization hardware state-machine is customized to support each specific topology.

The *message* field denotes the packet type. Host processes handle only two types of message – a barrier start and a barrier release message. The NetFPGA updates the message type based on its current state in the state machine. It may handle other message types based upon the current topology. For example, with the tree topology, there is an additional message to indicate that there are children at a barrier, and to notify the parent NetFPGA that all of its children have called the barrier. The *topo_type* field is to specify the network topology. Currently, we support ring, binary tree, butterfly and star topologies. The *node_type* field denotes the node type in a specified topology.

3.1.1 Life of Barrier Packet in the NetFPGA

Once a packet arrives at the NetFPGA, it is placed in the appropriate receive queue and is passed to the user data path. The receive queues attach a module header to inform subsequent modules about the packet source and length. From the input queue, the packet arrives at the *output_port_lookup* module which examines whether it is a barrier packet. If it is a barrier packet, based on the state it is in, the *output_port_lookup* module determines which ports the packet is going to be injected. If the packet is going to be forwarded to multiple ports, the packet is duplicated to the multiple transmit queues at the same time. If the packet is not a barrier packet, the packet is handled as a regular Ethernet traffic, and is forwarded based on the receive queue number it is received in.

3.2 Supported Topologies

We explain in detail how the organization of NetFPGA nodes in specific topologies and the communication required to achieve synchronization.

3.2.1 Ring Topology

The ring topology has two types of nodes: head and regular nodes. Head nodes wait for their host to call the barrier, and then for the rest of the ring to call the barrier. After the head node learns that all previous nodes are at the barrier, it initiates a release message to the next node and to the host. Non-head nodes wait for the nodes preceding them and their hosts to call the barrier before sending the *MSG_PREV_AT_BR* message; they then wait to receive a release message, initiated by the head node, and subsequently forward the release message to their host and successor in the ring.

In the ring topology, $port0$ is used to connect to the successor in the ring and $port1$ is used to connect to the predecessor node in the ring. On the wire, the synchronization packet flow runs in only one direction, which is from $port0$ of the current node to the $port1$ of the next node.

3.2.2 Tree Topology

The tree topology is implemented with three node types: root, internal, and leaf. As expected, a leaf node has no children and a root node has no parent. Internal nodes have both children and a parent. A leaf node waits for the host to call the barrier, sends a *MSG_CHILD_AT_BR* message, waits to receive a release message from its parent, and finally releases its host. The root node waits for its children and host to call the barrier (the order is irrelevant), then it sends a release message to its children and the host. An internal node is a combination of leaf and root nodes. It initially waits for its children and host to call the barrier. Then, it notifies its parent that its host and children have called the barrier. Finally, it waits for a release message from its parent. When an internal node receives a release message from its parent, it forwards the message to its children and host.

In the tree topology, $port0$ and 1 are used to connect to children (if any) and $port2$ is used to connect to the parent node (if any). In this topology, the packet flow is on both directions on the link.

3.2.3 Star Topology

The star topology is implemented with 2 node types: center and regular nodes. Center node will wait for all the other nodes connected to it to call the barrier. After all regular nodes send the *MSG_AT_BR* messages; they wait for center node to send the release message. The center node will craft the release message when all the other nodes and its host call the barrier. It sends the release message to the regular nodes and its host. In the star topology, if the node is a regular node, only $port0$ is used to connect to the center node. All of the ports of center node could be used to establish the star topology which is up to 5 nodes because of the NetFPGA port limitation.

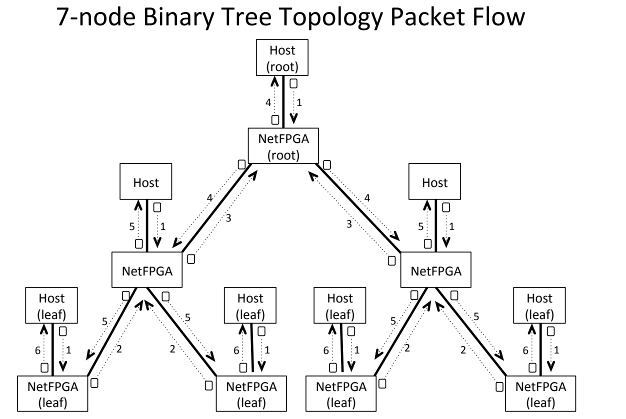


Fig. 2: NF_Barrier packet flow for 7 nodes in a binary tree

3.3 Sample Packet Flow

Figure 2 depicts a sample packet flow scenario for a 7-node complete binary tree.

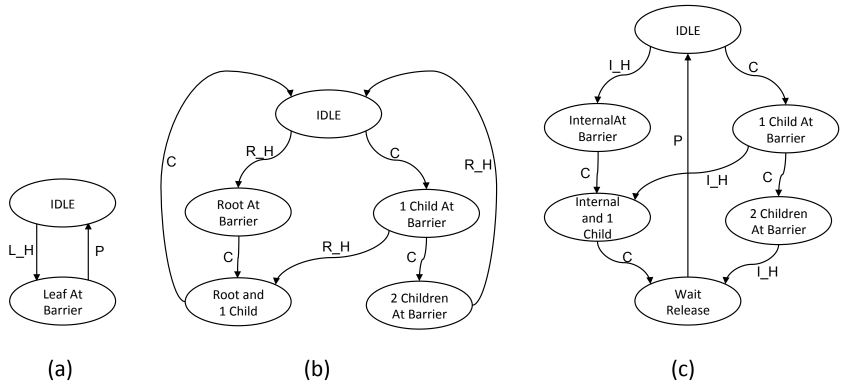
1) All the hosts invoke the barrier and the NetFPGAs receive the *MSG_AT_BR* messages. The NetFPGA stores necessary header fields for constructing a release message when the time comes.
2) Leaf NetFPGAs update the message received from the host and tell their parents that the node and its children are all at the barrier even when they have no children.
3) The NetFPGAs that are in between root NetFPGA and leaf NetFPGAs receive the messages from their children and since their hosts are also at the barrier, they forward the message to their parent which is the root NetFPGA.
4) Since all of its children and the host itself are at the barrier, the root NetFPGA crafts a release message with the remembered header fields and sends it to its children and the host at the same time.
5) Internal NetFPGAs also perform necessary header field updates, and forward the release message both to the host and children.
6) The leaf NetFPGAs receive the release message and, after updating the header fields, they release their host processes from the barrier.

The preceding scenario demonstrates the packet flow in our tree design. We are going to present a sample packet flow for our ring topology in the next section, while we describe our performance measurement model.

3.4 State Machines

To describe the designs, we provide the protocol state machines for the various nodes in the tree topology. Figure 3 presents the state machines employed in a full binary tree topology. Figure 3.d details the meaning of the state names and transitions between them.

We did not provide figures for the state machines for ring and star topologies because they are very simple. On



L_H	Leaf Host offloads barrier
R_H	Root Host offloads barrier
I_H	Internal Host offloads barrier
C	At Barrier packet from a child
P	Release packet from the parent

(d)

Fig. 3: State Machines for the binary tree topology (a) leaf nodes (b) root node (c) internal nodes (c) Legend for state transitions

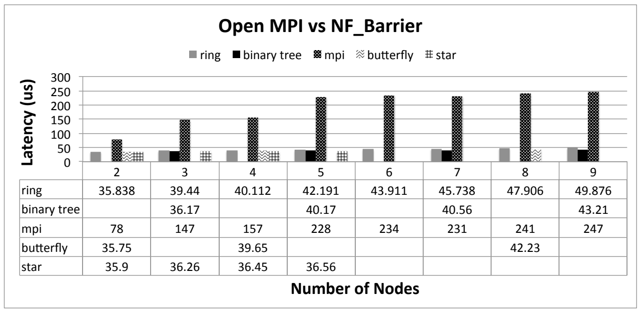
the other hand, the butterfly topology state machine is not presented but briefly discussed because it is so complex compared to the other cases. The non-central node in the star topology should employ a state machine like a leaf node in a tree case because its role is to wait for the host to call the barrier, notify the central node about the barrier call and wait for the central node to broadcast the release message. The central node is like the root node in the tree topology and it waits for regular nodes notifications to generate the release message. The order of message arrival from the neighbor hosts does not matter.

The most complex topology is the butterfly. The number of states is a lot more than other cases because the order of packet arrival matters in this implementation. There is no specific node role in this implementation and every node employs the same state machine unlike other cases discussed so far. In an 8-node butterfly topology, each NetFPGA must connect three other NetFPGAs. In addition, it will also interface to the host. Therefore, there are total of four ways to receive barrier packets. Since the order of the packet arrival matters, there are 4!=24 different sequences these packets can arrive. The order is important, and the ports, which packets are received from, represent different states. For example, if it is received from the $port1$, it means 2 nodes in the whole topology called the barrier. The states somewhat employ a logic to keep track of who have called the barrier until then in the whole topology from a single node's perspective. Since the butterfly algorithm is a 1-phase barrier algorithm, there is no release message circulating between the NetFPGAs and the only release message is sent from NetFPGA to the host.

4. Evaluation

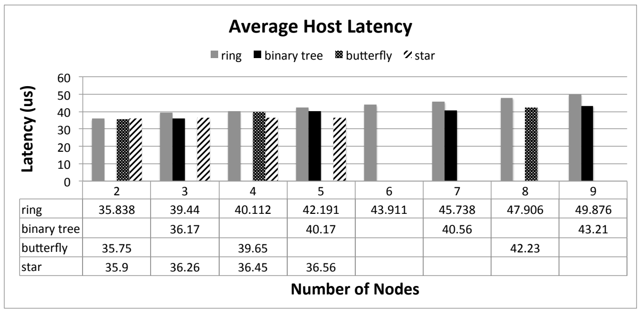
4.1 Experimental setup and results

Our experimental setup consists of 8 NetFPGAs in hosts with Intel(R) Core i5-2400 at 3.10GHz CPUs, 4GB RAM, and a dual Gigabit Ethernet NIC. The NetFPGA ports were directly connected to the each other establishing a tested topology. In this paper, we present micro-benchmark results obtained running OSU Micro-Benchmark Suite [3] for MPI_Barrier. In addition, we are going to describe how we can precisely time the NetFPGA operations after we offload the collective to the NetFPGA network.



	2	3	4	5	6	7	8	9
ring	35.838	39.44	40.112	42.191	43.911	45.738	47.906	49.876
binary tree		36.17		40.17		40.56		43.21
mpi	78	147	157	228	234	231	241	247
butterfly	35.75		39.65				42.23	
star	35.9	36.26	36.45	36.56				

Fig. 4: Performance comparison of NF_Barrier for implemented topologies to generic MPI_Barrier for Open MPI

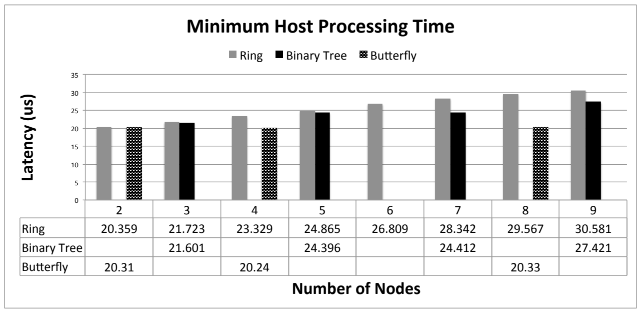


	2	3	4	5	6	7	8	9
ring	35.838	39.44	40.112	42.191	43.911	45.738	47.906	49.876
binary tree		36.17		40.17		40.56		43.21
butterfly	35.75		39.65				42.23	
star	35.9	36.26	36.45	36.56				

Fig. 5: Performance comparison of different topologies which are currently implemented for NF_Barrier

The benchmark is configured to run 10 million barrier calls and averaged latency results are recorded. Figure 4 shows the latency of a single barrier operation for different numbers of hosts and various topologies.

Even though averaged results give us significantly bet-



	2	3	4	5	6	7	8	9
Ring	20.359	21.723	23.329	24.865	26.809	28.342	29.567	30.581
Binary Tree		21.601		24.396		24.412		27.421
Butterfly	20.31		20.24				20.33	

Fig. 6: Minimum latency experienced by different topologies which are currently implemented for NF_Barrier

ter performance compared to the point-to-point Open MPI implementation, it does not precisely demonstrate how our design contributes to the overall barrier latency. According to the results presented in Figure 5, if the number of nodes is increased by one in ring topology, it introduces approximately 2μ latency. If the height of the tree increased by one, it introduces additional 3.5μ. It is also the same for the butterfly topology, if we increase the number of nodes by the power of 2. However, because of the node parallelism, these numbers are expected to be close to each other since we are introducing a single parallel NetFPGA processing to the overall processing time. We run our benchmarks to find out what the minimum latency of a barrier would be for various hosts in various topologies. The minimum latencies experienced are presented in Figure 6. The purpose of presenting the minimum results are to show that the host itself introduces a huge variance to the overall performance of our implementation. Therefore, it is not fair to evaluate our design based on average results unlike some other previous work [8]. As observed in Figure 6, when the host involvement in barrier latency is minimal, it provides more precise data for understanding how our design really contributes to the overall performance. Hence, we can extrapolate valuable information about processing time of the NetFPGAs. According to these results, an additional node to the ring introduces an average of 1.46μs latency, an increase in the height of the tree introduces 2.95μs of latency to the leaf nodes, and there is no latency introduced for the case of butterfly implementation.

We define p as the NetFPGA's single packet processing time. In the ring case, if a node is the last one to arrive barrier call, it will wait for its packet to circulate through the ring once. Therefore, an increase in the number of nodes in the ring would introduce extra latency of p to the last node arrived at the barrier. In a full binary tree, if a leaf node is the last one to arrive barrier, its notification packet goes up to the root, and then it is sent back to all the children as a release packet. Therefore, an increase in the height of a tree would introduce latency of $2p$ to the leaf nodes. In butterfly topology, we expect latency to increase p amount when the number of nodes increase by the powers of two. However, we do not see consistant results for this case in Figure 6. The presented numbers are a lot more consistent than averaging overall latencies and give us an idea about how fast the NetFPGA processes the packet. So, based on these results, p is around 1.46μs. However this is still not a precise measurement.

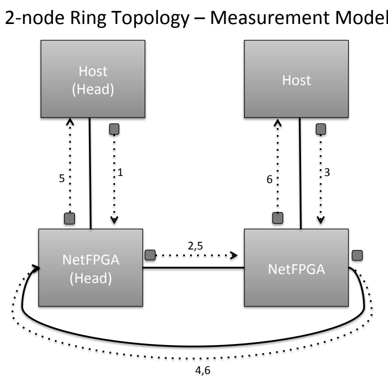
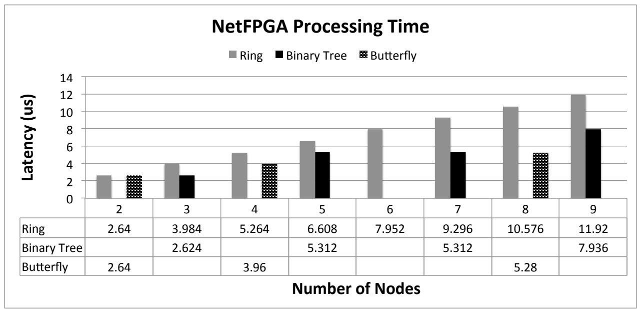


Fig. 7: Example packet-flow scenario that describes our precise performance measurement model

To precisely measure the NetFPGA processing time, p, we developed the model pictured in Figure 7. NetFPGA has a 125Mhz clock and we created a 64-bit timer which increments on each clock cycle. The steps to measure the NetFPGA's single packet processing time in 2-node ring topology are listed below.

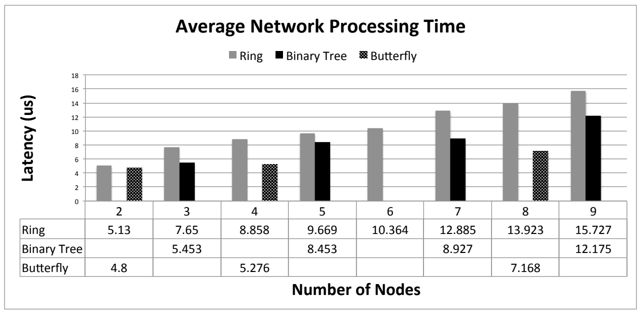
1) The host of the head node manually sends an MPI_Barrier message to the NetFPGA.
2) The NetFPGA forwards this packet to the second NetFPGA on the ring. Second NetFPGA then waits for its host to call MPI_Barrier.
3) The host of the second NetFPGA sends the MPI_Barrier message to its NetFPGA. The NetFPGA records the time at a certain place through the data-path.
4) The second NetFPGA now forwards the packet to the head NetFPGA, which is the head-node.
5) The head NetFPGA now knows that everyone has called MPI_Barrier. It generates a release message and forwards it to the host and the second NetFPGA at the same time.
6) When the second NetFPGA receives the release message from the head NetFPGA, it records the time again at the same place on the data-path. The difference between the two recorded timestamps provides the NetFPGA processing time for both nodes. This data is written into a packet which is sent to the host as a release message.

The time measured in this model includes the propagation delay. However, the propagation delay is negligible since we



	2	3	4	5	6	7	8	9
Ring	2.64	3.984	5.264	6.608	7.952	9.296	10.576	11.92
Binary Tree		2.624		5.312		5.312		7.936
Butterfly	2.64		3.96				5.28	

Fig. 8: Precise processing time of NetFPGA for different topologies and various number of nodes



	2	3	4	5	6	7	8	9
Ring	5.13	7.65	8.858	9.669	10.364	12.885	13.923	15.727
Binary Tree		5.453		8.453		8.927		12.175
Butterfly	4.8		5.276				7.168	

Fig. 9: Average latency introduced after MPI_Barrier is offloaded to the NetFPGA network for various topologies

used short cables to connect the hosts. Based on our precise measurements, p is 1.32μ. An increase in the number of nodes in the ring introduces 1.32μ delay, and the results are presented in Figure 8 that prove the consistency of our precise measurement. Similar model is used for the tree topology for the leaf nodes and an increase in the height of a tree introduces $2p$ latency. Based on the number of processing time, we put the estimate results for the butterfly algorithm, however they are not measured, since it is very hard to inject packets to the NetFPGAs at the same time because of the system noise of the different arrival times.

Network's average performance results after the host offloads the barrier operation to the NetFPGA still present valuable information especially for the butterfly topology. For these measurements we used a similar approach as we did for the precise measurements. However, in this case we recorded the timestamp when NetFPGA receives offload request from the host. The second timestamp is recorded when the NetFPGA sends the release message to the host. The difference is attached to the release packet. Measurements for the ring and butterfly are averaged for each host. However, for the tree case, only the results for the leaf nodes are averaged since the upper nodes are released quite earlier than the leaf nodes. Non-leaf nodes can introduce a huge bias and do not offer how the network processing time is related with the height of the tree. The results are presented in Figure 9.

5. Related Work

Zotov [20] proposes a hardware mechanism to synchronize n-dimensional mesh-connected MIMD computers. This work is one of the most comprehensive works in the literature about barrier synchronization and maps out the limitations of different synchronization frameworks. The work itself proposes a separate control networks for mesh-connected MIMD computers, and it is different from our work in three key aspects. This work proposes a separate control network for barrier synchronization. Instead, our work implements synchronization on the data network. Almasi et al. [5] are also another example claiming that it is better to have separate barrier logic and build a separate network to handle the synchronization.

Even though they are not considered as clusters of workstations, FPGA based network on chip (NoC) architectures are also related to our work. Mahr at al. [14] implement an MPI library for multiprocessor systems on a single chip. They connect the processing elements on a single chip in different ways such as a ring topology, star topology and shared bus. [15] also similarly proposes a centralized synchronization solution for 8 cores on a single chip. [7] is another example to achieve barrier synchronization on a NoC environment. According to [7] the defining feature is that the barrier release messages are broadcasted to facilitate the job of storing the source node information. Our work differs in that sense since we store the source node protocol information until the end of barrier release message. [18] discusses scalability and effect of different barrier algorithms on a NoC based platform. The algorithms investigated in this paper are central counter, combining tree and dissemination algorithm. Huang et al. [11] also focus on optimizing MPI primitives on a NoC system.

Moreover, there are some other proposals for different platforms. TMD- MPI [16] focuses on MPI_Send and MPI_Recv implementation in multi-FGPA platforms and [17] is the extension of their work to unite their design with a specialized x86 platform. Our work provides both a distributed barrier implementation and has the potential to support a variety of network topologies. Previous work [8] is the most similar to our work, but with some caveats. It is applicable only for a specific FPGA cluster architecture and topology - a tree. In contrast, we support a variety of topologies - all ring, tree and star are discussed in this paper. In addition, the communication between the FPGA systems is not using standard protocols as we do in our work. In another FPGA implementation [10], a single FPGA is used to collect barrier messages from connected hosts and to distribute them a release message when all nodes call barrier. It implements a centralized barrier algorithm employing a simple state machine. Fabric Collective Accelerator (FCA) [1] offloads the collective communication burden to Mellanox InfiniBand adapters and switches. Along with that [9] describes the implementation of a non-blocking barrier call with CORE-Direct hardware capabilities introduced in the InfiniBand NIC ConnectX-2. They provide a list of tasks that achieves the barrier utilizing recursive-doubling algorithm. However, unlike our work, this implementation does not totally implement the barrier collective in the hardware but defines the routine that employs the primitive tasks provided by the hardware.

6. Discussion and Future Work

Our design has obvious limitations, including manual configuration. We leave these to be addressed in future work. Moreover, even though we integrated our design into the Open MPI via simply replacing the included MPI_Barrier, a more significant integration effort is necessary to preserve the architecture and semantics of Open MPI.

In our packet format we defined a field called *comm_ID*. However, it is not used in this design; the goal is to distinguish active barrier operations, which may run on simultaneously for different MPI communicators. Each of the simultaneous barrier operation will require a separate state machine. Therefore, in order to distinguish the states of active barrier synchronizations, we are planning to investigate the best way to store the *comm_ID* with their associated barrier states. We are currently investigating approaches to store the (*comm_ID*, *barrier_state*) tuples since the read and write operations for those tuples are going to be almost equal.

Moreover, we are planning to put hardware logic into the NetFPGA to learn the topology of the NetFPGA collective network and configure node roles as appropriate. This information will be propagated to the MPI environment, eliminating the hardcoding that comes with the current design and making it portable to other NetFPGA network configurations. We are also planning to achieve the self-configurability without changing any system level driver, and implementing the logic at the hardware and user-level.

7. Conclusion

In this paper, we have presented preliminary results using NetFPGAs to implement MPI_Barrier synchronization. While the hardware designs presented have some limitations, the results provide strong evidence that this is likely to be a fruitful research domain. Limitations in our initial design include lack of mechanisms for failure recovery and the need for a pre-assigned root node. Our plans include better and more robust implementations of barriers and other synchronization mechanisms, performance evaluation on real parallel code, and integration with MPI libraries.

References

[1] Fabric collective accelerator. `www.mellanox.com/products/fca/`.

[2] Open mpi: Open source high performance computing. `http://www.open-mpi.org`.

[3] Osu micro-benchmarks 4.0. `http://mvapich.cse.ohio-state.edu/benchmarks/`.

[4] Reference router walkthrough. `http://wiki.netfpga.org/foswiki/bin/view/NetFPGA/OneGig/ReferenceRouterWalkthrough`.

[5] G. Almási, P. Heidelberger, C. J. Archer, X. Martorell, C. C. Erway, J. E. Moreira, B. Steinmacher-Burow, and Y. Zheng. Optimization of MPI collective communication on BlueGene/L systems. In *ICS '05: Proceedings of the 19th annual international conference on Supercomputing*, pages 1–10. ACM Request Permissions, June 2005.

[6] T. S. Axelrod. Effects of synchronization barriers on multiprocessor performance. *Parallel Comput.*, 3(2):129–140, May 1986.

[7] X. Chen, S. Chen, Z. Lu, A. Jantsch, B. Xu, and H. Luo. Multi-FPGA implementation of a Network-on-Chip based many-core architecture with fast barrier synchronization mechanism. *NORCHIP, 2010*, 2010.

[8] S. Gao, A. G. Schmidt, and R. Sass. Hardware Implementation Of MPI_Barrier On An FPGA Cluster. In *International Conference on Field Programmable Logic and Applications (FPL 2009)*, pages 12–17. IEEE, 2009.

[9] R. Graham, S. Poole, P. Shamis, G. Bloch, N. Bloch, H. Chapman, M. Kagan, A. Shahar, I. Rabinovitz, and G. Shainer. Overlapping computation and communication: Barrier algorithms and connectx-2 core-direct capabilities. In *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–8, 2010.

[10] T. Hoefler, J. M. Squyres, T. Mehlan, F. Mietke, and W. Rehm. Implementing a Hardware-Based Barrier in Open MPI. *Proceedings of KiCC*, 2005.

[11] L. Huang, Z. Wang, and N. Xiao. Accelerating NoC-Based MPI Primitives via Communication Architecture Customization. In *2012 IEEE 23rd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 141–148. IEEE, 2012.

[12] I. Jung, J. Hyun, J. Lee, and J. Ma. Two-phase barrier: A synchronization primitive for improving the processor utilization. *Int. J. Parallel Program.*, 29(6):607–627, Dec. 2001.

[13] J. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo. Netfpga–an open platform for gigabit-rate network switching and routing. In *Microelectronic Systems Education, 2007. MSE'07. IEEE International Conference on*, pages 160–161. IEEE, 2007.

[14] P. Mahr, C. Lörchner, H. Ishebabi, and C. Bobda. SoC-MPI: A Flexible Message Passing Library for Multiprocessor Systems-on-Chips. In *2008 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, pages 187–192. IEEE, 2008.

[15] M. Monchiero, G. Palermo, C. Silvano, and O. Villa. Efficient Synchronization for Embedded On-Chip Multiprocessors. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(10):1049–1062, 2006.

[16] M. Saldana and P. Chow. TMD-MPI: An MPI Implementation for Multiple Processors Across Multiple FPGAs. In *International Conference on Field Programmable Logic and Applications (FPL 2006)*, pages 1–6, 2006.

[17] M. Saldana, A. Patel, C. Madill, D. Nunes, D. Wang, P. Chow, R. Wittig, H. Styles, and A. Putnam. MPI as a programming model for high-performance reconfigurable computers. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 3(4):22, 2010.

[18] V. F. Silva, C. de Oliveira Fontes, F. R. V. Wagner, and S. on-Chip VLSI-SoC 2012 IEEE IFIP 20th International Conference on. The impact of synchronization in message passing while scaling multi-core MPSoC systems. In *VLSI and System-on-Chip (VLSI-SoC), 2012 IEEE/IFIP 20th International Conference on*, 2012.

[19] D. Tsafrir and D. Feitelson. Barrier synchronization on a loaded smp using two-phase waiting algorithms. In *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*, pages 8 pp–, 2002.

[20] I. Zotov. Distributed virtual bit-slice synchronizer: A scalable hardware barrier mechanism for n-dimensional meshes. *Computers, IEEE Transactions on*, 59(9):1187–1199, 2010.

Method of Extracting Parallelization in Very Large Applications through Automated Tool and Iterative Manual Intervention

Smitha K.P, Aditi Sahasrabudhe, Vinay Vaidya
Center for Research in Engineering Sciences and Technology (CREST), KPIT Technologies, Pune, India

Abstract

Program parallelization involves multiple considerations. These include methods for data or control parallelization, target architecture, and performance scalability. Due to number of such factors, best parallelization strategy for a given sequential application often evolves iteratively. Researchers are confronted with choices of parallelization methods to achieve the best possible performance. In this paper, we share our experience in parallelizing a very large application (250K LOC) on shared memory processors. We iteratively parallelized the application by leveraging selective benefits from automatic as well as manual parallelization. We used YUCCA, an automatic parallelization tool, to generate parallelized code. Using the information generated by YUCCA, we improved the performance by modifying the parallelized code. This iterative process was continued until no further improvement was possible. We observed performance improvement of 17% compared to 5% improvement reported in the literature. The performance improvement was gained in very short time and despite the constraint of having to use only SMPs for parallelization.

Keywords: Parallelization, Optimization, Performance, Share memory processors

1 Introduction

Parallelization at software as well as hardware level is extremely important to improve performance of software significantly. Program parallelization is most fruitful when opted from start of the design phase of an application. However, in order to port legacy applications onto parallel hardware and reap its benefits, parallelization is needed after an application is designed and implemented. In order to parallelize the application code manually, high proficiency in domain of the application as well as parallelization techniques is needed. Automatic parallelization tools are a natural choice when parallelization is considered after development of application. Automatic parallelization typically focuses on select techniques and looks for specific patterns in the application that have the potential to execute in parallel. Choice of appropriate automatic parallelization tools is important to parallelize applications for specific target platforms.

In this paper, we present our work in parallelizing a very large code set consisting of more than 250K LOC (Lines of Code) for shared memory processors. In order to bring in improvement in the performance, manually parallelizing the application at algorithm level was one option. However, considering the massive size of given application, it was not feasible to use manual parallelization at algorithmic level. YUCCA (User Code Conversion Application), an automatic parallelization tool developed at our research center, supports parallelization for SMPs [1]. We used it to squeeze first level of parallelization benefits from the code. By using the intermediate results generated by YUCCA, we manually improved performance of the application on a shared memory processor. In this paper, we present the work in exploiting parallelization with respect to a given gigantic code and a thorough analysis of success and failure of all parallelization strategies that we used. We also present a unique literature review by discussing various important factors, which are considered crucial for parallelization.

2 Literature Review

In this section, we highlight some work in the area of automatic parallelization and its relevance in case of large data sets or large-scale applications.

2.1. Data Dependence Analysis

Automatic parallelization tools, which make use of static information of the code, ultimately come up with code partitions that can be dispatched on multiple processors or cores [2, 3, 4]. In order to determine partitions of the program, data dependence analysis is a must [3, 5]. Programming languages allow access to variables in various fashions including arrays, pointers, pointers of arrays, arguments to functions, parameters of functions etc. Over and above, their scope and visibility, different type qualifiers, also play an important role in program behavior. Dependency analysis also needs to understand side effects of functions called at various places in the code. Real life applications do have multiple global variables, hierarchical calls to functions, iterative and recursive functions etc. As mentioned in [6, 7], the analysis becomes crucial when the programs are irregular and unstructured. In case of irregular programs, pointers, their level of indirection and their ability to contain more than one data points is a crucial factor [7]. Random and intuitive nature of program needs extensive data dependence analysis in unstructured programs [6]. Basis of dependency analysis is the read/update access to the variables. In [8], it is claimed that though necessary, a thorough end-to-end dependency checking may not be required for parallelization. Hence, the analysis is done on the basis of values, rather than memory.

2.2. Large Applications

Large program code sets typically have code distributed across multiple source and header files. Probabilistically, large number of LOC tends to cover variety of complex program syntaxes. The automatic parallelizing tools should be capable of handling and mapping of all these syntactical patterns accurately. The efficient use of system resources by the parallelization tools becomes an important factor especially in case of large applications [9, 10]. Applications having very large number of lines are difficult to parallelize, as they require a thorough understanding of the application as well as the domain [11, 12, 13].

Irrespective of the code size, performance benefits of parallelization depend on the inherent degree of parallelization the code supports. If the chunks into which the application is partitioned for parallelization are of small size, then switching time between these partitions increases. This in turn leads to less performance benefit.

2.3. Overhead of Parallelization

All parallelization techniques including use of APIs (like Pthreads, OpenMP, MPI, Intel TBB, etc.) add overhead on parallelized application. In all of these cases, the overhead arises from the synchronization and context switching time of these APIs. The absolute overhead time does not depend on the data size or the execution time. It depends on the number of parallel threads or processes created and how often the control switches from one to another [14, 15]. In [8], since the execution is broadly split into two paths, one is speculative execution and other is actual execution, most of the overheads are placed on the speculative path. However, as compared to the multithreaded/multiprocessing application, behavior oriented parallelization incurs additional overhead for protecting the data from unauthorized accesses by speculative paths. In order to keep the overhead minimum, the parallelization techniques believe on parallelizing the calls to the most time consuming functions, rather than parallelizing the body of that function [16].

2.4. Avenues of Parallelization

When parallelizing the source code of an application, it can be looked at from many perspectives. The most promising way is to find parallelization at the algorithm level. For example, an mp3 decoder applies same steps on left and right channel of the audio stream. These steps can be executed in parallel. Similarly, for image processing applications, if the same video stream is being used for two different applications, the applications can be run in parallel after preprocessing on the common video is done. The parallelization efforts at this level, though time consuming and difficult, typically derive more benefits than the parallelization efforts local to application code [17]. Automatic parallelization techniques focus on code sections which consume most of the application time, loops [18, 3], control paths [19, 20], etc.

2.5. Performance Improvement Depends on Size of Input Data

Parallelization tools, which make use of static analysis for dependency checking, cannot determine the size of input data. Moreover, parallelization tools focus only some part of the code / selected control paths in the code for parallelization. If the parallelized code and the input data are tightly coupled and size of input is large, then performance improvement resulting from parallelization of the code is more.

2.6. Target Architecture of Hardware on which Parallelized Code is Executed

The performance improvement because of parallelization, needless to say, is dependent on the hardware platform on which the parallelized code is executed. Number of cores/processors, processing speed, memory architecture play important roles in deciding performance gain of parallelized applications. As described in [6, 10], the shared memory architectures typically have multilevel caches, and typically have one or two levels of global memory accessible to all processors. When multiple applications execute on such hardware, with only one thread dedicated to each application, processor spends more time on memory access and gives less throughput. Data parallel applications tend to give more performance on shared memory architectures, such as RISC, GPGPUs etc. [21].

2.7. Static and Dynamic Analysis Methods

For parallelization of an application, behavior of program and the data needs to be modeled. This can be done in by analyzing the application at compile time or run time. Static analysis methods [2, 22, 14] collect information about the behavior of data and program at compile time. In order to complete the analysis for all possible behaviors of data and code, these methods tend to be more conservative and time consuming [22]. Such tools need to follow safer approach if the worst-case conditions or if the behavior of the program cannot be predicted [8]. In case of dynamic methods, the analysis is done speculatively based on the partial execution or based on the partial visibility to the input data [22]. Few analysis methods combine benefits of static and dynamic analysis, and analyze the applications in hybrid manner.

2.8. Handling I/O

In most of the algorithms, I/O operations need to be executed in the same order as they were in the sequential version. Hence, their presence poses a big challenge for parallelizing tools [7].

2.9. Library Code

Many applications use third party libraries in the applications. Absence of their source code can be a bottleneck for source code parallelizing tools. In such cases, depending on the input and output of the library APIs, worst cases need to be considered by parallelization tools. This problem gets aggravated if the libraries are using difference software languages for their implementation [7, 23].

2.10. Automatic Generation of Parallelized Code

APIs generated by the parallelization tool has to support target architecture [24]. APIs like OpenMP, Pthreads are used for parallel application executing on shared memory architectures, MPI are used for distributed memory architectures. OpenCL and CUDA provide APIs for GPGPUs. Inserting parallelization constructs at proper places in the code without altering its semantics and functional behavior is a tough task for parallelization tools [25].

3 Overview of Automatic Parallelization Tool

YUCCA, 'User Code Conversion Application,' is an automatic parallelization tool, which parallelizes complete projects/code sets, written in C language. YUCCA tool (earlier named as S2P tool [1]) is a source-to-source conversion tool; i.e. when a C application is given as input, YUCCA generates parallelized C application as output. The parallel code is a multithreaded code with Pthreads and OpenMP constructs inserted at relevant places. Throughout this text, words 'code', 'program' and 'application' are used interchangeably and they refer to the inputs and outputs of the YUCCA tool. YUCCA inserts Pthreads APIs in case of task parallelization and OpenMP APIs in case of loop parallelization. YUCCA tool consists of a compiler-like front-end that can preprocess, scan and parse application code and an intelligent back-end that performs static dependency analysis to identify parallelizable sections of code [1, 26]. YUCCA's front end synthesizes information about application in an XML schema. YUCCA's back end performs rigorous dependency analysis on this information. Results of these analysis methods are released to the user. The end result of dependency analysis includes a task dependency matrix (TDM) similar to static task graph in [2]. TDM is nothing but a matrix in which every code section is checked against each other for control as well as data dependencies [27]. Empty values in the matrix denote that there is no dependency in between two code sections. Non-empty values denote the dependency in the form of line number(s) on which the dependency exists between two code sections. Every such code section is called as a 'task'. According to [1], criteria for defining boundaries of tasks is 'first level programming constructs' in 'main' function of the C application. Expressions, selection statements, control statements, iterative statements, function call sites that are immediately contained by 'main' function form tasks. Nested programming constructs are analyzed for dependencies; however TDM reports these dependencies only at the 'first hierarchical level' of main function. The information presented in TDM is further used for partitioning the code. The code insertion module in YUCCA then inserts parallelization constructs around and inside these partitions. Scheduler executing on a multicore processor [28] schedules the multithreaded application generated by YUCCA.

4 OpenMX - Application for Nano-Scale Material Simulations

The code set, which we have considered as case study is OpenMX version 3.3. The software package OpenMX (Open source package for Material eXplorer) is designed for large number of nano-scale material simulations [29]. The algorithms used in OpenMX enable researchers to study electronic, magnetic, and geometrical structures of variety of materials. Hence, the package finds applications in areas of biomaterials, magnetic materials, nano-scale conductors, carbon nanotubes etc. Using this package, researchers working in these areas can have deep understanding of various complicated and useful materials [29]. Since the package is computationally intensive, improving its performance by parallelization helps to quickly simulate properties of above mentioned materials. The application has earlier been parallelized using MPI (Message Passing Interface) on distributed memory systems using three methods. Results of parallelization can be found on [29].

The OpenMX source code consists of 250 LOC spread across 192 C files and 10 header files. Work described in [30], denotes cost for developing parallel programs in terms of Person Minutes per LOC. According to the metric specified, mean cost of developing an OpenMP application is 24.8 person minutes/LOC. To develop an application of size 250K, it would take more than 12,916 person days (assuming 8 hours per person day). The numbers indicated denote the time to develop the application rather than parallelize it. Assuming that 25% of the effort would be spent in parallelizing the application, the effort reduces to 3,229 person days. Since the metrics mentioned in the literature mentions code development by novice programmers, we would still like to reduce it by 50% to consider parallelization efforts by parallelization experts. This consideration further brings down the efforts to 1614 days. This number gives a sense of huge efforts required for manually parallelizing an application of size of 250LOC.

5 Parallelization of OpenMX using YUCCA Tool

OpenMX code contained varied syntaxes, including six level of pointer, macro definitions containing approximately 10000 words on a single statement, numerous conditional operators etc. As per YUCCA's architecture, a binary expression is formed for each expression in the code and hence conditional operators are not handled. In order to overcome this, we changed the conditional statements to selection statements and completed the parsing of the OpenMX application through YUCCA. We could successfully parse the entire application through YUCCA with all the code complexities mentioned above.

The next step to automatic parallelization was the execution of YUCCA back-end, which actually does the conversion of sequential code to parallel code using Pthreads, based on thorough analysis of information present in the XML schema. YUCCA tool is capable of parallelizing loops and tasks selectively as well as simultaneously. In order to generate parallelized C code, YUCCA first generated Task Dependency Matrix (TDM) [1] showing information about the created tasks and the dependencies between all these tasks. After TDM generation, YUCCA released parallelized code for the entire OpenMX package. Table 1 shows comparison of execution time of sequential and YUCCA parallelized version of OpenMX application for various input sizes.

Table 1: Benchmarking of parallel execution time of OpenMX code

Input File	Size (KB)	Execution Time of OpenMX application (mm:ss)		Percentage Improvement (%)
		Sequential	Parallel	
Methane.dat	4	00:12	00:12	0%
C60.dat	6	01:00	01:00	0%
DIA216_DC.dat	17.8	09:57	10:02	-4.7%
DIA512_DC.dat	36.9	36:24	Memory allocation error	

The results mentioned in this section as well as subsequent sections are recorded on experimental set up mentioned in Table 2.

Table 2: Testing environment

Processor	Intel i3 processor-dual core (HT)
Operating System	32-bit Ubuntu Linux 10.04
Speed	3.2 GHz
RAM Size	4 GB

The first and second column of table 1 shows the name and size of the files. The third column shows the execution time of the input file with OpenMX sequential version. Fourth column shows execution time of OpenMX parallel version generated by YUCCA. Fifth column shows the percentage of performance improvement of parallel version over sequential version.

By comparing results of execution of parallelized version of OpenMX with the sequential one, we could verify that both the versions are functionally equivalent. By looking at result table 1, we can see that there is no performance improvement after the parallelization. However, there is performance degradation for the input file 'DIA216_DC.dat' with large size among the first three files. A thorough inspection of the parallelized code leads to following crucial observations:

1. The core computation of the application is handled by only one thread, and the computation varied for different input files according to the settings we are specifying the input.dat file.
2. By default, YUCCA partitions tasks in the main function and inserts parallelization constructs around the boundaries of these tasks. In case of OpenMX code, the core computation happens at inner code level. Hence the code section which performed this core time-consuming computation is not parallelized.
3. There was an overhead because of large number of threads created.
4. For one of the input files (DIA512_DC.dat), we were not able to execute the parallelized version successfully, because the system ran out of memory. As the file size is large, memory required for the computation is more. In case of parallelized code, as there was more number of parallel tasks, there is a limitation on memory available to each thread. This limited thread-stack memory did not suffice for the computations in case of large files.

5.1. Customization of Parallelized OpenMX Code

5.1.1. Parallelization of DFT Function

To get an insight of performance of parallelized application, we profiled the OpenMX sequential code using Valgrind and found that the function 'DFT ()' takes around 85% of the total execution time. Since, 'DFT' function was at 2 levels inside main function, YUCCA created only one thread for the whole 'DFT ()' function. Hence, the core time consuming computation did not get parallelized. All the threads, except the thread executing DFT function, were idle most of the time. Therefore, we realized that parallelizing 'main' function will not give any performance benefit. Hence, we modified the YUCCA tool in such a way that the tool will parallelize the function according to user inputs. With a modified version of YUCCA tool, we got another parallelized version of OpenMX code.

5.1.2. Cosmetic Code Changes to Reduce the Size of TDM

The parallelized code was once again checked for correctness by comparing its results with the results of serial version. Once the functional equivalence was tested, we observed that 45 tasks are created, in the parallelized code. YUCCA creates one thread per task. Because of 45 threads, we started facing the issue related to memory exhaustion. In order to fix the memory related issues resulting from more number of tasks (and thus threads), we made some cosmetic changes to the code. As shown in figure 1, we modified the application code, such that only limited numbers of tasks were created. As per YUCCA design, 5 tasks (and hence 5 threads) will be created for code in figure 1.a, and only 3 tasks (and hence 3 threads) would be created by making some superficial changes as shown in figure 1.b.

```
{
        Selection statement#1;
        Loop#2;
        Loop#3;
        Control statement#4;
        FunctionCall#5;
}
```

Figure 1.a. Sample code snippet before changes

```
{
        {
                Selection statement#1;
                Loop#2;
        }
        {
                Loop#3;
                Control statement#4;
        }
        {
                FunctionCall#5;
        }
}
```

Figure 1.b. Sample code snippet after superficial changes

By making similar change in the OpenMX code, less number of tasks, and hence threads, were created by YUCCA. After making these changes, the parallel version executed without any memory allocation errors. However, it did not lead to any performance improvement as shown in table 3.

Table 3: Benchmarking of parallel execution time after DFT parallelization and cosmetic code changes

Input File	Size (KB)	Execution Time (mm:ss)		Percentage Improvement (%)
		Sequential	Parallel	
Methane .dat	4	00:12	00:12	0%
C60.dat	6	01:00	01:01	0%
DIA216_ DC.dat	17.8	09:57	09:58	0%
DIA512_ DC.dat	36.9	36:24	36:24	0%

5.1.3. Merging Data Dependency Analysis with Value Analysis

A deeper look at the TDM and the side effect analysis report generated by YUCCA gave us the list of variables, which were causing the dependency. In one case, only one variable was creating interdependencies between two tasks. In the dependent task, a function was invoked four times. Out of four call sites, first call site used a particular literal value of the argument and the remaining three function call sites passed another value of the argument. Inside the body of function definition, a selection statement bifurcated use of these literals to different computations. We used value analysis within the function definition and gathered that only the first call site would pose dependency problems instead of all the four call sites. Hence, in the parallelized code, we removed the synchronization constructs that were placed due to last three call sites. Removal of these constructs opened few more opportunities for parallelization.

5.1.4. Data Privatization

In another case, the dependency was there due to update of an array variable. Even if two different memory locations/indices of array are accessed or updated by two tasks, YUCCA would treat it like dependency pertaining to the entire array variable, when executed in task parallelization mode. The array update was happening in all parallel tasks and on different indices. Moreover, the values were not being utilized for any functional calculation. Since the array update occurred in the core computation loop of all tasks, each following task needed to wait until the completion of previous task. To avoid this situation, we applied data privatization for the array variable in all the tasks, and added code to transfer the values from temporary array variable to original array variable after the loop iteration. Also the wait and post synchronization signals were placed accordingly. Because of this modification, the computations in the parallel tasks were happening until the point where the data transfer from temporary variables to actual array variables happened. Only the tasks were waiting for some fraction of time. Therefore, we could achieve performance improvement for the parallelized version.

By doing manual optimization in YUCCA parallelized code, we achieved a performance benefit of 11 – 17%. The increment factor depends on the size of the input file. From the result table, we could see that the performance improvement is less for DIA512_DC.dat file (size: 36.9KB). This is because of two reasons. Based on the contents of the input file, different computations are performed in the application. Other reason is the size of the file. If we select files with same input settings, then the percentage of performance benefit decreased for files with larger size. In the graph shown in figure 2, performance benefit obtained for DIA512_DC.dat is less than the benefit obtained for DIA216_DC.dat. But the benefit obtained for C60.dat is more than what we obtained for DIA512_DC.dat. Here the files, DIA216_DC.dat and

DIA512_DC.dat have same settings in the input file. Hence, when there are files with same settings, then performance benefit is less for file with larger size.

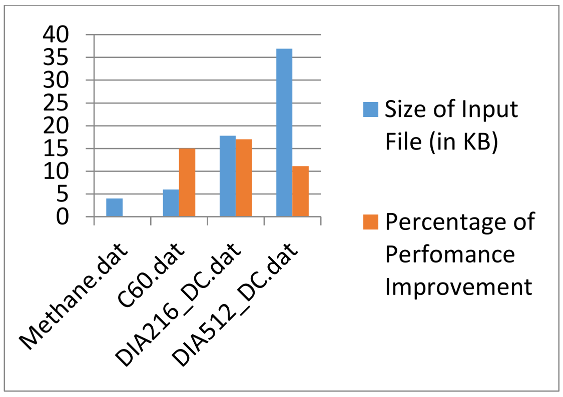


Figure 2: Size of OpenMX input files versus percentage of performance improvement

5.2. Role of Automatic Parallelization Tool for Obtaining Parallelized OpenMX Application with Improved Performance

Even if we did some manual modifications in the code for optimization, all the modifications were based on the results produced by YUCCA tool. There were no manual interventions in dependency analysis or task creation. As far as task synchronization is considered, we changed positions for some of the wait/post signals to resolve some dependencies. However, we did not add/remove any synchronization signals added by YUCCA tool. In addition, there were no dead locks/data races in the parallelized code produced by YUCCA tool. We reached this conclusion based on testing done on the parallelized code regarding functional integrity.

Table 4: Benchmarking of parallel execution time after DFT parallelization, cosmetic code changes, value analysis, and data privatization

Input File	**Size (KB)**	**Execution Time of OpenMX (mm:ss)**		**Percentage Improvement (%)**
		Sequential	**Parallel**	
Methane .dat	4	00:12	00:12	0%
C60.dat	6	01:00	00:51	15%
DIA216 _DC.dat	17.8	09:57	08:14	17%
DIA512 _DC.dat	36.9	36:24	32:21	11.13%

As mentioned in this paper, there had been past efforts of parallelization of OpenMX code. Parallelization of the code using MPI was carried out by 3 different ways [29]. For 2 processors, the performance improvement was always less than or close to 5%. By leveraging the strength of automatic parallelization tool and manual comprehensive efforts, we could get performance gain up to 17% as shown in table 4. Apart from the gain in performance improvement, we would also like to highlight the speed of parallelization. As mentioned in section II, it takes almost 12,916 person days to develop the parallel application of the size of OpenMX. As against this number, first round of parallelization using YUCCA merely took 48 hours. Further superficial changes to the application and customization of YUCCA code was completed in less than 50 days.

6 Conclusion and Future Work

In this paper, we have shown how to combine usage of automated tool along with selective changes to the code iteratively to achieve best possible parallelization. For manual customization, we made use of information generated by the tool itself. These efforts involved reducing numbers of threads, data privatization, and exploiting parallelization at multiple first level blocks in the code. By making such changes to the code and without even understanding the functionality of the code, we could fetch 17% improvement in the performance of the code, when executed on shared memory processor. We completed parallelization of 250 KLOC code in about 2 person months. This is estimated to 3% of the manual effort required for parallelizing such a large code.

The next challenge is to convert the lessons learnt from manual parallelization into an algorithm. This algorithm can further be used to enhance the automatic tool.

7 References

[1] Aditi Athavale, Priti Ranadive, M. N. Babu, Prasad Pawar, Sudhakar Sah, Vinay Vaidya, and Chaitanya Rajguru, "Automatic Sequential to Parallel Code Conversion The S2P Tool and Performance Analysis", *Global Science & Technology Forum (GSTF) Journal on Computing*, 2012, vol. 1, no. 4, pp. 128-137.

[2] Vikram Adve, Rizos Sakellariou, "Application Representations for Multiparadigm Performance Modeling of Large-scale Parallel Scientific Codes", In *The International Journal of High Performance Computing Applications*,2000, vol. 14, no.4, pp. 304-316.

[3] Banerjee, Utpal, Rudolf Eigenmann, Alexandru Nicolau, and David A. Padua, "Automatic program parallelization", *Proceedings of the IEEE* 81, no. 2, 1993, pp. 211-243.

[4] Vivek Sarkar, "Partitioning and scheduling parallel programs for multiprocessors", *PhD dissertation*, MIT press, 1989.

[5] Anish Sane, Priti Ranadive, and Sudhakar Sah, "Data dependency analysis using data-write detection techniques", In *International Conference on Software Technology and Engineering (ICSTE) 2010*, vol. 1, pp. 1-9.

[6] Lumsdaine, Andrew, Douglas Gregor, Bruce Hendrickson, and Jonathan Berry, "Challenges in parallel

graph processing" *Parallel Processing Letters* 17, no. 1, 2007, pp. 5-20.

[7] Brian Armstrong, and Rudolf Eigenmann, "Challenges in the Automatic Parallelization of Large-scale Computational Applications", *In Proceedings of* International Symposium on the Convergence of IT and Communications, 2001, pp. 50-60.

[8] Chen Ding, Shen Xipeng, Kelsey Kirk, Tice Chris, Huang Ruke, and Zhang Chengliang, "Software behavior oriented parallelization", *In ACM SIGPLAN Notices*, 2007, vol. 42, no. 6, pp. 223-234.

[9] Krepska, Elzbieta, Thilo Kielmann, Wan Fokkink, and Henri Bal. "Hipg: parallel processing of large-scale graphs", *ACM SIGOPS Operating Systems Review* 45, no. 2, 2011, pp. 3-13.

[10] Zanni, Luca, Thomas Serafini, and Gaetano Zanghirati, "Parallel software for training large scale support vector machines on multiprocessor systems", *The Journal of Machine Learning Research* 7, 2006, pp. 1467-1492.

[11] Edward Carmona, "Parallelizing a large scientific code-methods, issues, and concerns", *In Proceedings of the 1989 ACM/IEEE conference on Supercomputing*, 1989, pp. 21-31.

[12] Gonina, Ekaterina, Anitha Kannan, John Shafer, and Mihai Budiu, "Parallelizing large-scale data processing applications with data skew: a case study in product-offer matching", In *Proceedings of the second international workshop on MapReduce and its applications*, 2011, pp. 35-42.

[13] Wilson, M. R. "Parallel molecular dynamics techniques for the simulation of anisotropic systems", In *Advances in the Computer Simulations of Liquid Crystals*, 2000, pp. 389-415.

[14] Wu, Peng, Arun Kejariwal, and Călin Caşcaval, "Compiler-driven dependence profiling to guide program parallelization", In *Languages and Compilers for Parallel Computing*, 2008, pp. 232-248.

[15] Grama Ananth, Anshul Gupta, and Vipin Kumar, "Isoefficiency function: A scalability metric for parallel algorithms and architectures", *IEEE Parallel and Distributed Technology, Special Issue on Parallel and Distributed Systems: From Theory to Practice* 1, no. 3, 1993, pp. 12-21.

[16] Boby George, Pooja Nagpal, "Optimizing Parallel Applications Using Concurrency Visualizer: A case study", Parallel Computing Platform Group, Microsoft Corporation, 2010.

[17] Pankratius, Victor, Ali Jannesari, and Walter F. Tichy, "Parallelizing bzip2: A case study in multicore software engineering", *Software, IEEE* 26, no. 6, 2009, pp. 70-77.

[18] Carl D Offner, "Modern Dependence Testing", *HP Labs, Technical Report HPL-2005-177*, September 1996.

[19] Vinay Vaidya, Pushpraj Agrawal, Aditi Athavale, Anish Sane, Sudhakar Sah, and Priti Ranadive, "Increasing Parallelism on multicore processors using Induced Parallelism", In *Software Technology and Engineering (ICSTE) 2010*, vol. 1, pp. V1-5, IEEE, 2010.

[20] Aleen, Farhana, and Nathan Clark. "Commutativity analysis for software parallelization: letting program transformations see the big picture", *ACM Sigplan Notices* 44, no. 3, 2009, pp. 241-252.

[21] Slogsnat, David, Markus Fischer, Andrés Bruhn, Joachim Weickert, and Ulrich Brüning, "Low level parallelization of nonlinear diffusion filtering algorithms for cluster computing environments", In *Euro-Par 2003 Parallel Processing*, 2003, pp. 481-490.

[22] Michael Ernst, "Static and Dynamic Analysis: Synergy and Duality", In International Conference on Software Engineering, 2003, pp-25-28.

[23] Brian Armstrong, and Rudolf Eigenmann, "Application of automatic parallelization to modern challenges of scientific computing industries", *In* 37th International Conference on *Parallel Processing*, 2008, pp. 279-286.

[24] Gonzalez-Alvarez, Cecilia, Youhei Kanehagi, Kosei Takemoto, Yohei Kishimoto, Kohei Muto, Hiroki Mikami, Akihiro Hayashi, Keiji Kimura, and Hironori Kasahara, "Automatic parallelization with OSCAR API Analyzer: a cross-platform performance evaluation", *IPSJ SIG Technical Report*, vol.2012-HPC-137 no.10.

[25] Vandierendonck, Hans, Sean Rul, and Koen De Bosschere. "The Paralax infrastructure: automatic parallelization with a helping hand", In *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, 2010, pp. 389-400.

[26] Vinay Vaidya, Priti Ranadive, and Sudhakar Sah, "Method and system for speeding execution of software code", PCT/IN2009/000697, 2009.

[27] Vinay Vaidya, Priti Ranadive, Sudhakar Sah and Jaydeep Vipradas, "Method of Reorganizing Tasks to Achieve Resource Optimization", PCT/IN2009/000701, 2009.

[28] Vinay Vaidya, Sudhakar Sah, Priti Ranadive, "Optimal Task Scheduler For Multicore Processor", In *International Conference on Software Technology and Engineering (ICSTE), 2010,* vol. 1, pp. V1-1 – V1-4.

[29] J. OpenMX. (2013, May 23). Welcome to OpenMX [Online]. Available: http://www.openmx-square.org/

[30] Hochstein, Lorin, Jeffrey Carver, Forrest Shull, Sima Asgari, Victor Basili, Jeffrey K. Hollingsworth, and Marvin V. Zelkowitz, "Parallel programmer productivity: A case study of novice parallel programmers", In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, IEEE, 2005, pp. 35-35.

Wait-less Parallel (MPI) Programming: A Disciplined Approach

Ralph Butler, Chrisila Pettey, and Nathan Reale
Department of Computer Science
Box 48
Middle Tennessee State University
Murfreesboro, Tennessee, USA

ralph.butler, chrisila.pettey(@mtsu.edu), and ncr2g@mtmail.mtsu.edu

Abstract - *Throughout the years the introductions of new programming paradigms have been greeted by many with skepticism. From goto-less (or structured) programming [1,2] to lock-free programming [3] there was an initial feeling of "how can this possibly work, and if it does will it be better than what we already have?" The idea of wait-less message passing can also have a slightly magical feel. The notion of receive carries with it the idea of waiting for a message. However, there is no magic, there are technologies that permit you to do it. But, using those kinds of technologies can lead to problems with debugging and maintenance. So we are proposing a discipline on the use of wait-less message passing that makes it feasible even for applications that have random communication patterns.*

Keywords: message passing; MPI; wait-less programming.

1 Introduction

In the past when people were advocating goto-less programming [1], it was realized that to encourage people to adopt the model, a more positive view was to think in terms of a disciplined approach that became known as structured programming [2]. Similarly, in our work, non-blocking message passing [6] (including pseudo out-of-band) has certain negative aspects such as difficulty of development and debugging. Because of these difficulties, the model is less widely used even though it has potential for significant performance gains. So we are proposing the more positive terminology of a discipline we call the wait-less model. Obviously the goals differ somewhat in that structured programming was mostly about ease of maintenance whereas our approach emphasizes performance gains. Nonetheless they both require a disciplined approach.

Anyone who has done much programming with MPI [5] knows that there are advantages such as performance gains to be had by using constructs such as *isend* and *irecv/rsend*, but people do not normally write code that *never* waits for anything, i.e. wait-less programming. It is clear that performance gains of that nature depend on non-blocking functions. So the new MPI-3 non-blocking collectives have made it possible to take advantage of non-blocking communication in situations where you might like to mimic the ability to deliver out-of-band data much like in the TCP/IP protocol suite. So, for example, in a branch-and-bound algorithm, a message containing a new bound needs to reach all participants ahead of most other message types. By using a non-blocking *ibcast* you can accomplish that if all the ranks are using this proposed wait-less model. A related, but in our view slightly different problem that can be tackled with non-blocking collectives, is the need to solicit help from one or more other ranks - for example, in a branch-and-bound algorithm where one of the ranks has depleted its work and needs to request work from any rank that can provide it. In the case where the ranks have been sharing status with non-blocking collectives, then a non-blocking *send* is probably sufficient to make the request. Otherwise a non-blocking *ibcast* or other collective would prove more useful.

Wait-less programming is often regarded as difficult, so you probably only want to do it for applications that involve non-trivial synchronization associated with the parallelism. Parallel libraries [4] already exploit this kind of model. They sometimes operate in a manner much like MPI does internally. In other words, they implement the notion of a progress engine. This means that on every invocation to the library, they check to see what things need to be done and try to progress each one of them at least a little bit. They have the disadvantage that they have to wait until the user invokes some library function. The application has the advantage that, if it never waits, it can simply *test* to see which outstanding operations need attention.

In the following sections we will describe the wait-less model and the discipline that constrains the manner in which it is implemented. The discipline is the key because it helps you to obtain the benefits of the model and yet alleviate many of the problems associated with maintenance and debugging.

2 The Model

The model we are proposing is best used for applications where random message exchanges have to occur at unpredictable times, so in order to make the best use of resources we need to be able to compute until additional data becomes available and also be able to deliver pseudo out-of-band messages that relay that data to other processes. In order to motivate our desire for the proposed model, we need to mention other parallel paradigms that we see frequently implemented in applications. Each of these paradigms has its place, and can be of use (and should be used) in some applications.

The first commonly used model is perfect for the lock-step interactions of something like a client-server. For this situation *send* and *recv* are usually fine. The second scenario would be to use a *probe* (or an *iprobe* to alleviate waiting) for any source - any tag. However, the sender in this situation cannot do an *rsend* in circumstances where the user's protocol would permit it, so the sending processes will have to wait. Perhaps more importantly, you cannot probe a collective - which is something that is needed for the pseudo out-of-band messages.

When programmers begin their foray into non-blocking message passing with MPI, they may use the *irecv, isend*/*rsend* combination in an algorithm like the following:

```
        process 1            process 2
        irecv                compute
        isend                  ...
        compute              rsend (matched to the irecv)
        wait(irecv)            ...
```

This does take you into the realm where you are starting to get better overlap of function. Unfortunately this is still a somewhat lock-step algorithm. The algorithm must be enhanced with *test* operations on both the *irecv* and the *isend* with a loop back to the compute portion of the algorithm in order to have wait-less message passing.

The enhanced algorithm still does not handle out-of-band message passing. Out-of-band message passing permits the rank to get a message quickly to others, perhaps even leap-frogging existing messages. I.e. high priority messages as in the example mentioned above of a rank needing to solicit new work. For such out-of-band message passing you could use one of the non-blocking collectives such as *ibcast* and ensure that the other ranks are checking for completion of the operation at the top of each iteration of their compute loop.

To unify the two models of point-to-point and collective operations, you need some common technique for checking for completion. Fortunately both of them use the *MPI_Request*, which can always be checked with any of the test family, i.e. *test, testall, testany*.

In the next section, we describe our disciplined approach for using the *irecv*, *isend*/*rsend* combination with the non-blocking collectives to implement the model.

3 The Disciplined Approach

In this approach we make use of the fact that all the non-blocking operations, both point-to-point and collective, produce a *request* that we can use a *test* to determine completion. We have chosen the word *reap* to describe our total process of completing a request. The situation is analogous to that of reaping a process from the operating system so it does not become a zombie. However, in this case, we not only want to allow MPI to reclaim internal resources about the request, but we also need to perform necessary cleanup on our side as well – for example freeing buffers. The disciplined approach consists of an initialization phase and a loop over four parts as follows:

```
initialization: The initialization phase is where we
    set up any non-blocking calls (either point-to-point or collective such as irecv's and
    ibcast's) that create a request. A good example of why this needs to be done might
    be to do an ibcast with the master rank as the root to establish our presence in the
    collective in case the master needs to get a new bound to every process. It is important
    to register every request so it can be handled by a testany in the loop below.
loop until done:
  testany for all registered requests
  if any was reaped
    if this request type has an associated buffer
        free the buffer
    if this request type requires handling by the application
        handle the request. This is likely to involve posting additional requests.
        For example, if it was a non-blocking ibcast then you need to go back into it
        and register it again.
  if some local work is available
    perform a portion which may generate more local work and may cause more
    isend/irecv/ibcast's that need to be registered. If more work was generated,
    and we have an outstanding request from another rank, then one of the non-blocking
    messages will be a reply to that. On the other hand, all local work may have been
    depleted here. In which case, the issue
```

```
        will be dealt with in the following *if*
        statement.
    if no local work is available and there is no
        outstanding request for work
        request some which will likely result in
        additional *isend/irecv/ibcast's* that have to
        be registered.
```

4 Experiences and Conclusions

Having initially explored this approach in the construction of parallel libraries, we decided to further investigate it by presenting it as a student research project in a graduate classroom with both masters and PhD level students. We motivated the approach by discussing a variety of alternatives. The only serious alternative that arose from these discussions (other than those described earlier in the paper) was remote memory access (so-called one-sided operations). The most compelling argument against the RMA approach was that the holder of a piece of data that needs to be broadcast must do a *put* operation into every other rank. We concluded the conversation of the disciplined approach with a branch and bound version of the traveling salesman problem as an application that could benefit from this approach. In that discussion we described an algorithm that used rank 0 as a server that could act as a clearinghouse for available work and distribute new bounds as they become available and check for termination. Since the client ranks were where almost all the work gets done, we focused on applying the disciplined approach to them. One student expressed particular interest in the project and went on to implement the traveling salesman project that was discussed in class. He is currently testing his implementation on our Beowulf cluster. After further testing we will put his work into our class resources.

Wait-less programming is very difficult - both in terms of development and debugging. Because of these difficulties, few programmers choose to use this paradigm even though it could give them performance gains. But if a programmer has a template, a discipline, to follow, this makes it more likely that they will program using the model and program correctly. Our experiences with our students have taught us that the disciplined approach seems to alleviate the fear of this kind of programming, and we plan to continue using it in the future.

5 References

[1] Dijkstra, E., "Go To Statement Considered Harmful," *Communications of the ACM* 11,3 (March 1968), pp. 147-148.

[2] Dijkstra, E. *A Discipline of Programming*, Prentice-Hall, Inc., 1976.

[3] Herlihy, Maurice, and J. Eliot B. Moss. Transactional memory: Architectural support for lock-free data structures. Vol. 21. No. 2. ACM, 1993.

[4] Lusk, Ewing L., Pieper, Steven C., Butler, Ralph M., “More Scalability, Less Pain,” SciDAC Review 2010 http://www.scidacreview.org/1002/html/adlb.html Accessed March 2014.

[5] MPI-3 http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf Accessed March 2014.

[6] Nonblocking Communication. Section 3.7 of the MPI Forum. http://www.mpi-forum.org/docs/mpi-1.1/mpi-11-html/node44.html Accessed May 2014.

Study of single-electron DOMINO logic circuit

Hiroyuki Otake*, and Takahide Oya
Graduate School of Engineering, Yokohama National University,
Tokiwadai 79-5, Hodogaya-ku, Yokohama 240-8501, Japan
*E-mail: otake-hiroyuki-zg@ynu.jp

Tetsuya Asai
Graduate School of Information Science and Technology, Hokkaido University,
Kita 14, Nishi 9, Sapporo 060-0814, Japan

Abstract—**A single-electron (SE) circuit, which is just one type of nanodevice, has been attracting attention in the nanotechnology research area. However, we have yet to determine the most appropriate information processing architecture for the SE circuit. So, as a candidate for the architecture, we are proposing the application of DOMINO logic theory to the SE circuit. DOMINO logic circuit is a logic circuit that is based on the behavior of a domino. To make sure that the DOMINO logic circuit is the most appropriate information processing architecture for the SE circuit, we designed and evaluated an actual SE calculator that is based on the DOMINO logic circuit. To do this, we designed basic logic circuits, e.g., OR, AND, and XOR, using the SE DOMINO logic circuit, and evaluated them by first conducting a Monte Carlo simulation. As a result, we confirmed that our circuit performs correctly.**

Keywords: single-electron circuit, domino logic circuit, single-electron oscillator

1. BACKGROUND AND MOTIVATION

Nanodevices have recently been attracting attention for use as novel devices having unique nonlinear phenomena in nanotechnology research area. Single-electron (SE) circuits, which are just one type of nanodevice, have also been attracting attention because of their unique behavior, i.e., the Coulomb blockade phenomenon [1, 2]. We can control individual electrons by harnessing the phenomenon in the SE circuit. Various useful applications of SE circuits have already been proposed by many researchers. However, we have yet to determine the most appropriate information processing architecture for SE devices. In this study, we propose a new logic circuit design for the SE circuit for consideration as a candidate for the most appropriate architecture. In particular, we apply the "DOMINO logic [3]" theory to the SE circuit. The DOMINO logic circuit is based on the behavior of dominoes. When a domino falls in the DOMINO logic circuit, we can assume it represents a logical "1." In contrast, when a piece does not fall, it represents a logical "0." It was previously clarified that basic logic circuits, e.g., OR, AND, and XOR, can be designed using the DOMINO logic circuit as described in Ref. [3]. Mimicking the behaviors of dominoes, i.e., "speeds of falling dominoes are constant" and "when two falling lines of dominoes collide head-on, both lines will stop falling," is important when using DOMINO logic circuits in electrical devices. We use the SE circuit in this study to mimic the behaviors of the dominos. We appropriately named this circuit a SE DOMINO logic circuit. We propose our SE DOMINO logic circuit as a candidate for consideration as a new SE circuit in this study. Moreover, we aim to design a practical SE calculator, e.g., a full adder. The SE calculator will be a novel information processing device. We will clarify here that the DOMINO logic circuit is the most appropriate information processing architecture for an SE device. We will present our basic SE DOMINO logic circuit, i.e., OR, AND, and XOR, and that by conducting a Monte Carlo simulation as a first step in this study, we confirmed that their operations are feasible.

2. THEORY

We use a SE oscillator (SEO) that consists of a bias voltage V_d, a resistance, and a tunneling junction in series to mimic the behaviors of dominos in this study. The tunneling junction has a threshold value for the electrons to tunnel through it, i.e., we can control the flow of electrons by changing the bias voltage. Our SEO also has a threshold value for changing the voltage at both ends of the tunneling junction caused by the electron tunneling. Figure 1 shows a schematic of the SEO. Figure 2 shows an example of its operation when V_d is bigger than the threshold value of the tunneling junction. We can see in Fig. 2 that the voltage at both ends of the tunneling junction in a SEO rapidly changes at regular time intervals, because electron tunneling occurs.

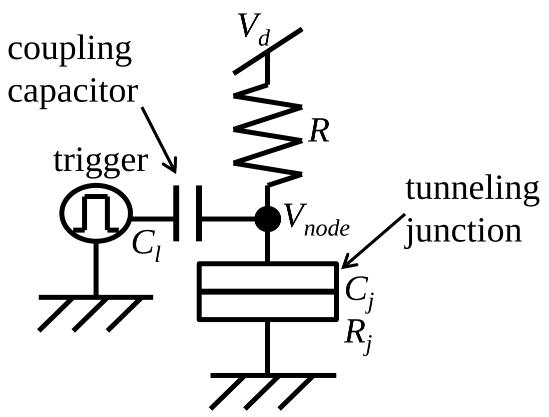


Fig.1: Schematic of SEO
(R=77[MΩ], C_l=4[aF], C_j=10[aF], R_j=0.2[MΩ])

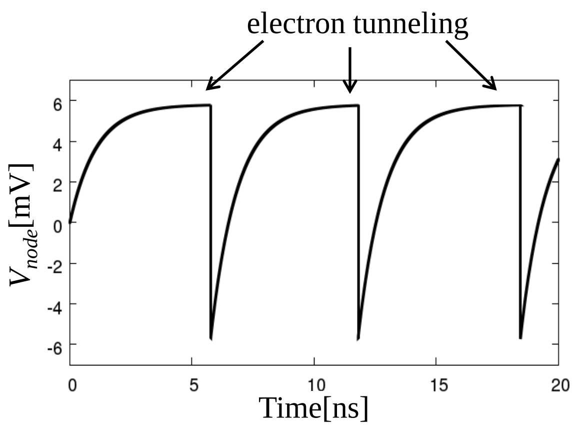


Fig. 2: Sample operation of Fig. 1
(V_d=5.7[mV])

We use arrayed SEOs connected together using coupling capacitors to mimic "falling domino lines" on the SE circuit. Figure 3 shows a schematic of a one-dimensional chain of six SEOs as an example. Each polar character of the bias voltage of the SEOs is inverted to alternating. In Fig. 3, when the signal is inputted at SEO V1 as a trigger, the electron tunnels in V1, and the node voltage V_{node} of the tunneling junction in V1 rapidly changes. In addition, the drastic voltage change of V1 becomes the input trigger for V2 through the coupling capacitor. In this way, the electron tunneling occurs one after another. Figure 4 shows the sample operation of Fig. 3. We can assume this operation is the "falling DOMINO lines" one. However, the electron tunneling occurred due to a quantum phenomenon with randomness. Therefore, the spreading speed of the mimicked falling is not constant, as shown in Fig. 4. We propose the use of multiple tunneling junctions (multiple-junction SEO (Fig. 5)) instead of a single tunneling junction in a SEO [4].

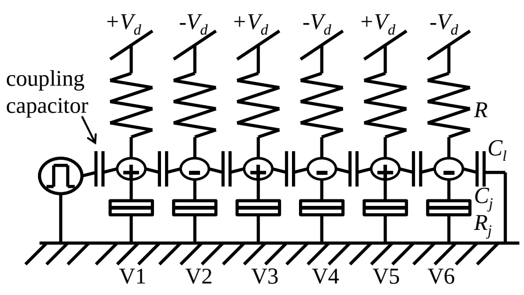


Fig.3: Schematic of one-dimensional chain of six SEOs
(R=77[MΩ], C_l=4[aF], C_j=10[aF], R_j=0.2[MΩ])

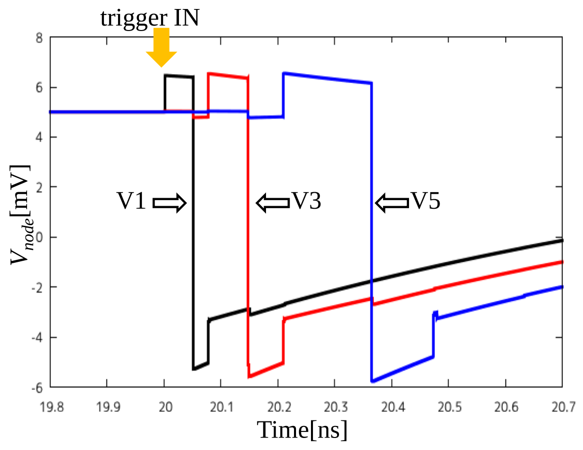


Fig. 4: Sample operation of Fig. 3
(V_d=5.0[mV])

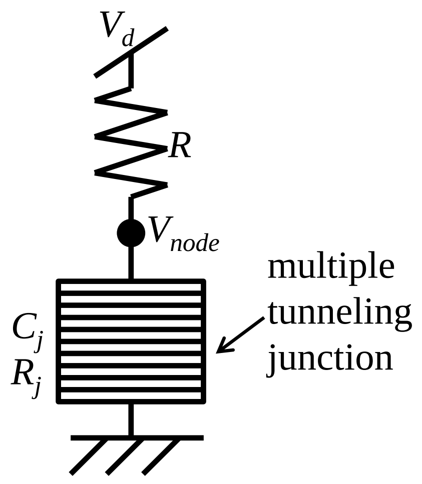


Fig.5: Schematic of multiple tunneling junctions SEO
(R=77[MΩ], C_j=500[aF], R_j=0.2[MΩ], 50 layers)

We can set the falling speed to almost constant in the lines by using the multiple-junction SEO. Figure 6 shows the sample operation of Fig. 5. We can confirm from Fig. 6 that the voltage change occurs due to multiple electron tunnelings. If multiple tunneling junction SEOs are arrayed with coupling capacitors just like in Fig. 3, the operating principle does not change. However, we determined that the spreading speed of mimicked falling was almost constant.

The SEO cannot generate the second electron right after the first tunneling occurred because the first drew the change in V_{node}. As a result, the second cannot occur until the V_{node} is recharged by V_d. Therefore, we can also mimic the "stopping falling dominoes by collision." Figure 7 shows the sample operation of Fig. 3, when the input trigger provides SEOs from both ends. For an easy explanation of Fig. 7, the node voltages of negative biased SEOs were multiplied by -1. We can see from Fig. 7 that the two signals from both ends of Fig. 3 collide at V3 and V4, and the propagation of the electron tunneling stops. These techniques give us a hint for how to apply the DOMINO logic to the SE circuit.

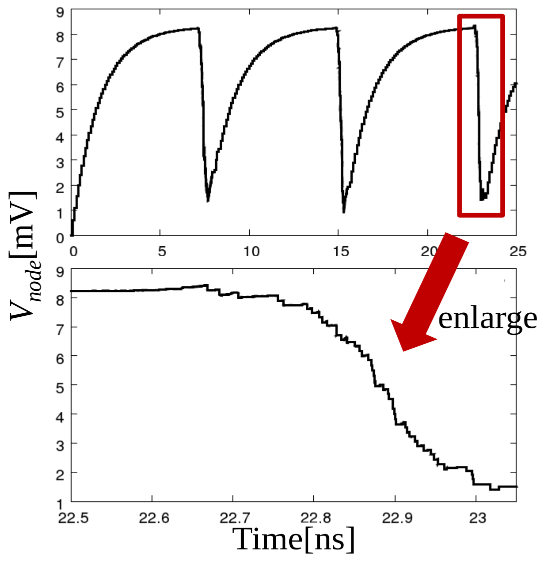


Fig. 6: Sample operation of Fig. 5 (V_d=8.2[mV])

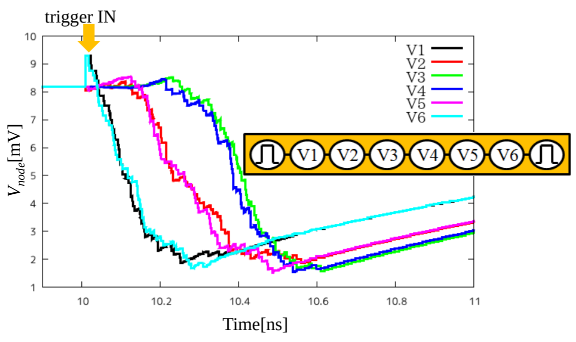


Fig. 7: Sample operation of Fig. 3 when two input triggers are provided from both ends (R=77[MΩ], C_l=2[aF], C_j=500[aF], R_j=0.2[MΩ], 50 layers, V_d=±8.1[mV])

3. SIMULATION

We designed the basic logic circuits, e.g., OR, AND, and XOR, using our SE DOMINO logic circuit to demonstrate the SE calculator. We used multiple tunneling junction SEOs (containing 50 tunneling junctions in series) to design a basic logic circuit. We designed a multi-function logic circuit that can operate as an OR, AND, and XOR circuit, and changed its function by changing the bias voltage on one circuit in this study. By using the multi-function logic circuits, the structure of the circuit could be simplified to design the SE calculator.

Figure 8 shows a schematic of the multi-function SE DOMINO logic circuit. In Fig. 8, each blue circle represents one SEO, and the coupling capacitor between each SEO is omitted. For easy explanation of Fig. 8, we assigned a number to each SEO as based on X-Y coordinate (X,Y). For example, the SEO connected to trigger A_{in} was numbered (2,4). In the multi-function SE DOMINO logic circuit, the input parts were (2,4) and (6,4) and the output part was (4,1). We used the Monte Carlo simulation to confirm the validity of our multi-function SE DOMINO logic circuit.

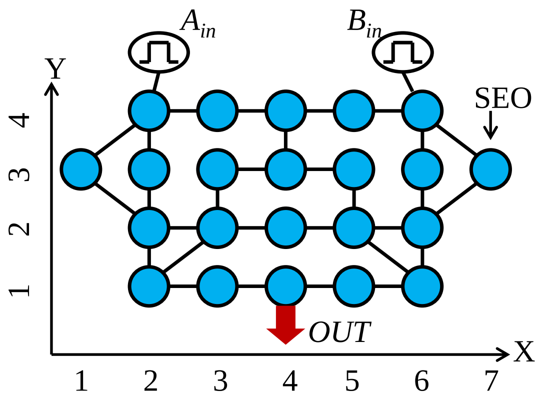


Fig. 8: Schematic of multi-function SE DOMINO logic circuit (R=77[MΩ], C_l=2[aF], C_j=500[aF], R_j=0.2[MΩ], 50 layers)

First, we confirmed when our SE DOMINO logic circuit performed as an OR. Figure 8 describes how to set the bias voltage of each SEO for the OR operation on the SE DOMINO logic circuit. In Fig. 9, a plus or a minus in the circle represents the polarity of the bias voltage of each SEO, and a blank circle is a zero biased SEO that is a power-off. For an example of the signal flow in the OR mode, an input signal from A_{in} follows along the path (2,4)→(3,4)→(4,4)→(4,3)→(5,3)→(5,2)→(4,2)→(3,2)→(2,1)→(3,1)→(4,1) and finally arrives at the output part. On the other hand, a signal from B_{in} follows the path (6,4)→(5,4)→(4,4)→(4,3)→(5,3)→(5,2)→(4,2)→(3,2)→(2,1)→(3,1)→(4,1) and finally arrives at the output part. Figure 10 shows the simulated operation if A_{in} was set to logical "1" and *Bin* was set to logical "0", and also if A_{in} and B_{in} were both logical "1." We could confirm from the results that the OR mode of our SE DOMINO logic circuit operated correctly.

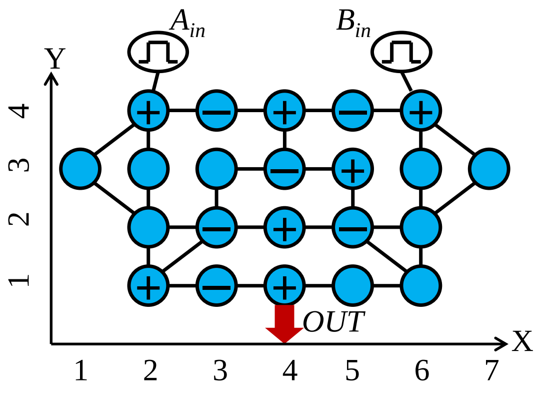


Fig. 9: OR mode of SE DOMINO logic circuit (R=77[MΩ], C_l=2[aF], C_j=500[aF], R_j=0.2[MΩ], 50 layers)

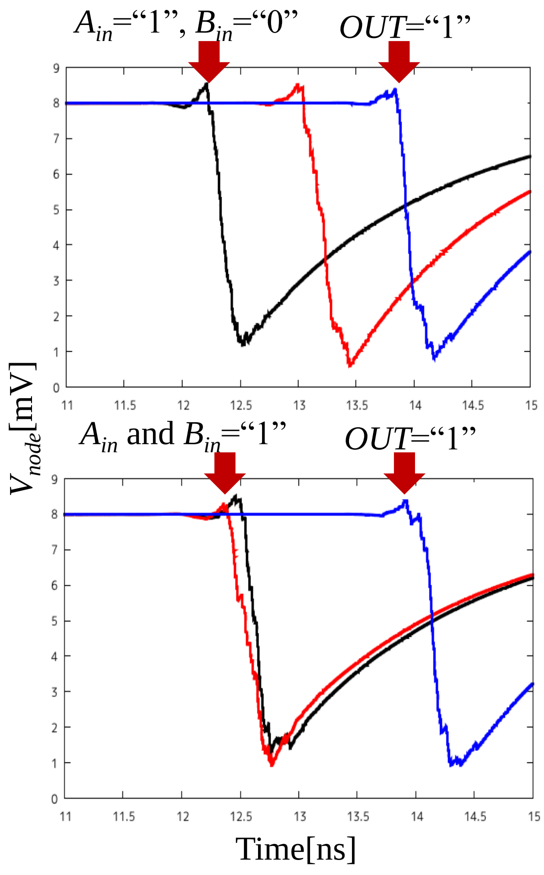


Fig.10: Simulated operation of OR mode (V_d=±8.0[mV])

Next, we confirmed the AND-mode operation of our SE DOMINO logic circuit. Figure 11 describes how to set the bias voltage of each SEO for an AND. In Fig. 11, a plus or a minus in the circle represents the polarity of the bias voltage of each SEO, and a blank circle represents a zero biased SEO that is a power-off just like shown in Fig. 9. Moreover, the bias voltage of only (4,4) was set lower than the others. In the lower biased SEO, two trigger signals from neighboring SEOs were required to produce electron tunneling at the SEO. In the AND mode, the path for the signal flows that the input signals from A_{in} and B_{in} followed was the same as that in the OR mode. Figure 12 shows the simulated operation if A_{in} was a logical "1" and B_{in} was a logical "0," and also if A_{in} and B_{in} were both logical "1." We confirmed from the results that the AND mode of our circuit operated correctly.

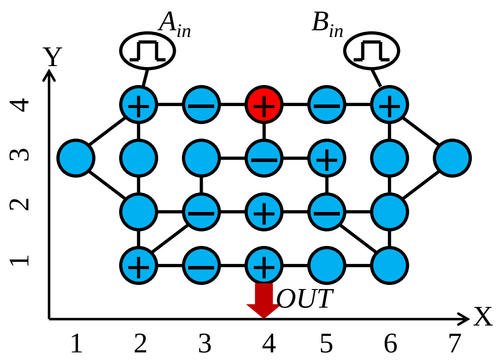


Fig. 11: AND mode of SE DOMINO logic circuit (R=77[MΩ], C_l=2[aF], C_j=500[aF], R_j=0.2[MΩ], 50 layers)

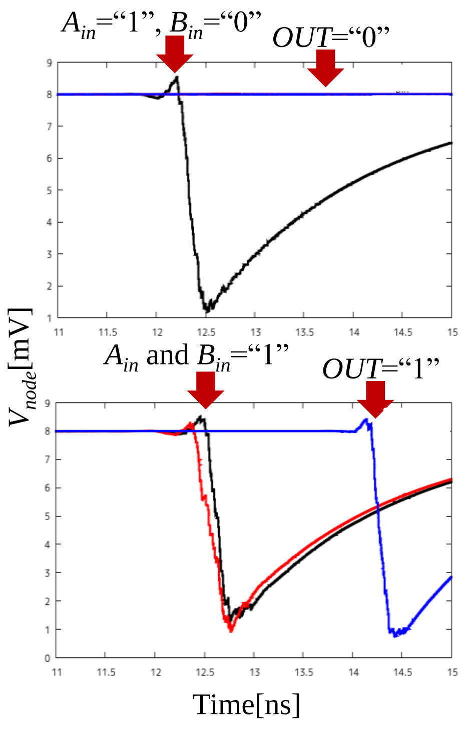


Fig.12: Simulated operation of AND mode (V_d=±8.0[mV], V_d at (4,4)=7.4[mV])

Finally, we confirmed the XOR-mode operation. Figure 13 describes how to set the bias voltage of each SEO for XOR. For the XOR-mode operation, the bias voltage of (2,2) and (6,2) was set lower than the others. In an XOR operation, a signal from A_{in} follows the paths (2,4)→(1,3) and (2,3)→(2,2)→(3,2)→(3,3)→(4,3)→(5,3)→(5,2)→(6,1)→(5,1)→(4,1), and finally arrives at the output part. On the other hand, a signal from B_{in} follows the paths (6,4)→(6,3) and (7,3)→(6,2)→(5,2)→(5,3)→(4,3)→(3,3)→(3,2)→(2,1)→(3,1)→(4, 1), and finally arrives at the output part. Figure 14 shows the simulated operation if A_{in} is a logical "1" and B_{in} is a logical "0," and if A_{in} and B_{in} were both logical "1." If both input signals from A_{in} and B_{in} were a logical "1," the two signals would collide at (4,3) and stop propagating as a result of the collision. We could confirm from the results that the XOR mode of our circuit operated correctly.

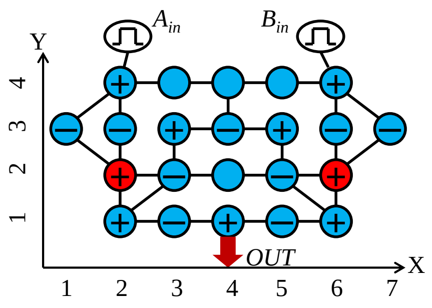


Fig. 13: XOR mode of SE DOMINO logic circuit (R=77[MΩ], C_l=2[aF], C_j=500[aF], R_j=0.2[MΩ], 50 layers)

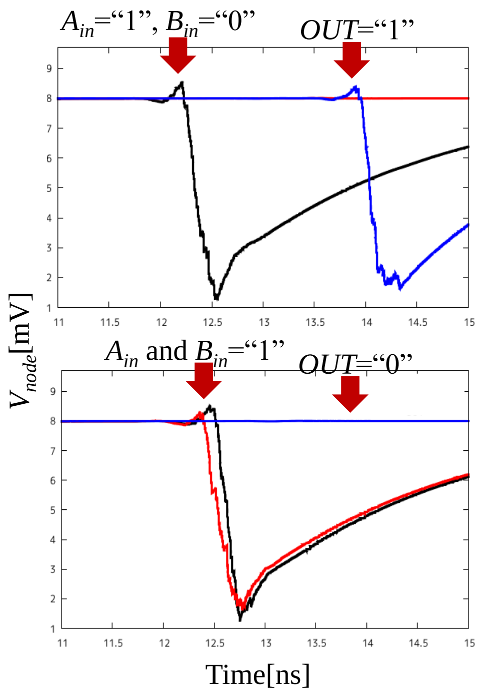


Fig.14: Simulated operation of XOR mode (V_d=±8.0[mV] , V_d at (2,2) and (6,2)=7.4[mV])

We confirmed the multi-function SE DOMINO logic circuit can operate as OR, AND, and XOR circuits. However, the multi-function SE DOMINO logic circuit was based on the premise that there was no time interval between two input signals of A_{in} and B_{in}. In other words, if there was time interval, the multi-function SE DOMINO logic circuit was not able to operate as OR, AND, and XOR circuits correctly. For example, when there was time interval between the two input signals of A_{in}= "1" and B_{in}= "1" in XOR mode, two input signals should not collide at (4,3) and we may fail to get correct output signal "0". To solve this problem, we have to design the circuit which adjusts the time interval of two signals in future work.

4. CONCLUSION AND FUTURE WORK

We aimed at designing a new type of SE information processing circuit in this study. For this propose, we proposed the application of DOMINO logic theory to the SE circuit. Mimicking the behaviors of dominos is required in the circuit in order to use the DOMINO logic in the circuitry. For this, we used a SEO chain because we can mimic the DOMINO behavior by using SEO chains. To make sure that the DOMINO logic circuit was a suitable information processing architecture for the SE circuit, we designed and demonstrated the basic SE DOMINO logic circuits. In particular, we designed a multi-function SE DOMINO logic circuit and confirmed its correct operation. The multi-function SE DOMINO logic circuit can act as OR, AND, or XOR by changing the bias voltage. In our future work, we will design the circuit which adjusts the time interval of two signals. Then, we will combine the adjusting circuit and the multi-functions of SE DOMINO logic circuits, and design and demonstrate the SE calculator, e.g., a full adder by using a simulator.

We must consider the influence of noise for effective operation of our SE circuit. We did not simulate noise in this study, because we aimed to design a new type of SE information processing circuit. However, we believe we can overcome the influence of noise because previous studies [5,6] on the relation between the SE circuit and the influence of noise have been ongoing. Therefore, the proposals and demonstrations in this study showing that the DOMINO logic circuit is a viable candidate for use as the most appropriate information processing architecture for the SE circuit should be significant.

5. ACKNOWLEDGMENT

This work was partly supported by JSPS KAKENHI Grant Numbers 25110015 and 24710149.

6. REFERENCE

[1] H. Gravert, and M.H. Devoret, "Single Charge Tunneling–Coulomb Blockade Phenomena in Nanostructures," New York: Plenum, (1992).

[2] T. Oya, T. Asai, T. Fukui, and Y. Amemiya "Reaction-Diffusion Systems Consisting of Single-Electron Oscillator," International Journal of Unconventional Computing, vol. 1, pp. 115-128, (2004).

[3] S. O'Keefe, "Implementation of Logical Operations on a Domino Substrate," International Journal of Unconventional Computing, vol. 5, pp. 115-128, (2009).

[4] T. Oya, T. Asai, and Y. Amemiya, "A Single-Electron Reaction-Diffusion Device for Computation of a Voronoi Diagram," International Journal of Unconventional Computing, vol.3, pp. 271-284, (2005).

[5] Y. Murakami and T. Oya, "Study of two-dimensional device-error-redundant single-electron oscillator system," SPIE Proceedings, vol. 8463, 84631E, (2012).

[6] H. Fujino and T. Oya, "Study of stochastic resonance in a quantum dot network," SPIE Proceedings, vol. 8463, 84631D, (2012).

Impact of Thread Synchronization and Data Parallelism on Multicore Game Programming

Abu Asaduzzaman, Hin Y. Lee, and Deepthi Gummadi
Department of Electrical Engineering and Computer Science
Wichita State University, Wichita, Kansas, USA

Abstract—*Xbox-360 has three cores with six logical threads and the PlayStation-3 has one master core and six independent worker cores. According to the current design trends, multicore processors will be ubiquitous in every game computer. A game engine has many 'components' and multithreading is an important technique to parallelize the execution of these components. However, effective programming of multiple threads in multicore systems has challenges including concurrent processing, thread synchronization, data and task level parallelism, and load balancing. In this paper, we investigate the challenges and benefits of thread synchronization and data level parallelism on multicore game engine programming. We implement a multi-object interactive game engine in an 8-core workstation using single-threaded model (STM), multithreaded asynchronous model (MAM), multithreaded synchronous model (MSM), and multithreaded synchronous model with data parallelism (MSMDP). Experimental results show that MSMDP is the best and it reduces the execution time up to 50%.*

Keywords: Data level parallelism; game engine; multicore architecture; multithreaded programming; thread synchronization;

1. Introduction

There are many components in a simple modern game engine. According to the flow of operations, important components in a single threaded game engine are: Input, Game Logic, Artificial Intelligence (AI), Physics (engine for collision detection/response), Audio (for sound), and 3D Graphics. A rendering engine called "renderer" is required for 2D or 3D graphics. A graphics package may include scene graph, culling and sorting, skeletal animation and rendering. Inside a component, there may be many subcomponents that 'glue' together to form a complete package. Some of these components can be middleware to make programming easier. An operation from start to finish is known as a clock cycle.

In addition to standalone game machines, game engines are nowadays being used for educational, engineering, and scientific applications [1]. To fulfill the high performance requirements, game engines are adopting new hardware technologies like multicore CPUs [2] and software technologies like multithreaded parallel programming [3], [4].

Many parallel programming techniques are available; one or more of them can be used in game engine programming. When components in a game engine are originated from many different middleware, the design of the library will most likely dictate which one is more suitable to be used. Some middleware such as Bullet Physics library includes multithreading in their application programming interface (API). Depending on the type of multithreading model used, some level of data redundancy is required to improve performance. Therefore, a mechanism to ensure data/cache coherency is needed in the implementation.

As the number of cores in a processor increases but the speed of the core has not changed much in the recent years, multithreading can be very helpful to get as much performance out of a system as possible to the advancement of video game technologies. Currently available middleware used in high-level API, like Open MPI, make the parallel implementation a challenge. Therefore, various methods should be evaluated when implementing multithreading in a game because the components usually never work the same way. One multithreading technique might not be suitable for a particular API of a component because of the way it is built; optimization of multithreaded game engines requires a lot of experimentations.

In this work, we implement a multi-object video game engine using middleware from different vendors. Various multithreaded asynchronous and synchronous models, with and without data parallelism, are implemented to study their effectiveness to improve the performance of multicore game engines.

The rest of the paper is organized as follow: Section 2 reviews some related published articles. Section 3 explains the impact of data/task parallelism, and synchronization. A multi-object multithreaded video game engine implementation is presented in Section 4. Some simulation results are discussed in Section 5. Finally, this paper is concluded in Section 6.

2. Background Study

The game industry has surpassed the movie and music industry in U.S. in 2005 and 2007, respectively. In 2008, the game industry surpassed the music industry in the U.K. and is expected to surpass DVD sales in the future. The desire for more complex game elements is driving the game industry forward. Multithreaded parallel programming has potential to implement complex game engine. However, multicore CPU (not manycore GPU) is a relatively new technology, especially in the game development world.

Multicore architecture is a recent design trend and most vendors are adopting multicore processors to their products. Multilevel cache memories are common in multicore processors [5]. The cache memory hierarchy normally has level-1 cache (CL1), level-2 cache (CL2), and main memory. In most cases, CL1 is split into instruction cache (I1) and data cache (D1) and CL2 is a unified cache [2]. Performance and power consumption are impacted by cache misses, increased usage of main memory, and poor cache memory arrangement. Using communication that is too fine grained can cause the cache to be underutilized [6]. A thread reading the data can receive the set of multiple data objects first and then process all of them. The effective size of the set of data objects can be calculated by the size of each line and the size of the cache line size of the cores. When communication is too coarse grained, capacity misses could happen when there are large amount of objects being copied that are larger than the cache size. Some processors use shared cache (like shared CL2) that can be accessed by multiple cores. When more cores access the same cache, there will be overhead of managing the use of the cache by multiple processors. Multicore systems are very suitable for multithreaded processing as multiple threads can be executed on multiple cores at the same time [5].

Task level parallelism is a popular method for game engine multithreading, where components run asynchronously in their own loop or synchronously in a single loop with multiple forks and joins. An asynchronous model of game engines has been introduced in [7]. In this model, as soon as a task is done, it will run immediately from the beginning. Data sharing could limit the effectiveness of this model depending on the amount of synchronization required. The multithreaded game engine introduced in [7] is an asynchronous model that uses multiple render states to buffer data. In this implementation, there is one ‘world’ state and three ‘render’ states. For a game engine, data parallelism is where the same type of data in a component is parallelized in multiple threads. The use of this in a game engine is when a component spawns multiple worker threads to process one type of data. To properly scale a multicore game engine, task parallelism and data parallelism have to be employed as introduced in [8].

When the cores are used more evenly, it gives more opportunities for developers to implement more distributed and parallel game play elements [9]. Intel Corporation has used a ‘thread pool’ mechanism to manage tasks as discussed in [10]. In a thread pool, each component has one or more tasks that will be queued and threads that are idle or have finished a task will retrieve a task from the queue to run next. This ensures that there will be a maximum amount of threads that can run at the same time. A multicore architecture is integrated to expose multithreading concept to game programmers to different number of cores without recompilation of code.

RedLynx has implemented multithreading in their game Trials HD [11]. It uses the Bullet Physics Engine for physics simulation. The library is optimized in-house for the Xbox 360 CPU and the vector units. One of the new features is the threaded asynchronous resource loading [12]. Developers can load rendering resources in a thread-safe way and use them concurrently with the rendering operation.

In this work, we use thread pooling technique and graphics rendering API to develop a multithreaded game engine to evaluate the impact of synchronization and data parallelism on performance of the game engine.

3. Important Techniques for Game Engine

Some important techniques used in game engines to improve performance are briefly explain in the following subsections.

3.1 Data Level Parallelism

Data parallelism is the distribution of the same type of data to process across different threads. For a game engine, data parallelism is where the same type of data in a component is parallelized in multiple threads [13]. This is used in a game engine when a component spawns multiple worker threads to process one type of data. If only data parallelism is employed, the series of different types of operations are sequential, only the data of a type of operation are processed concurrently at one stage. If the type of data requires communication among themselves, a thread safe communication system has to be implemented. This method scales well for many number of processors because the size of the data for each thread can be divided equally. It may also be easy to balance the load among multiple cores because there is only one type of object being processed concurrently. When the data type does not share data with each other, this method of parallelism can easily be implemented to scale on any number of threads. Communication among the threads can be reduced by grouping the objects that are most likely to interact with each other in the same thread [14].

3.2 Task Level Parallelism

Task parallelism is the distribution of different task across different threads. The use of task parallelism in game engine is by running each component task in its own thread [15], [16]. There are two model of execution for this method: the synchronous model and asynchronous model. The synchronized model is where all the tasks of the components must finish in a single clock cycle. At the end of the clock cycle, the application will loop to the beginning to start the operations in the same order every time. The asynchronous model is where the tasks of the components can run and finish at their own time. To share data among the threads, a synchronization stage can be used in between the clock cycle in this model. In an asynchronous model all the components run in their own loop. Some components that do not always have new data for a single frame are usually implemented asynchronously such as resource loading, player input, and networking.

3.3 Data and Task Level Parallelism

To properly implement a game engine that will scale properly and fully utilize parallelism for various numbers of cores, both data parallelism and task parallelism have to be employed. A mixture of task and data parallelism is an optimum approach to exploit multithreading in game engines [17]. In this combination, each task can run in parallel with another task and may spawn several worker threads. A system may have number of cores less or more than the number of parallelizable components. In task parallelism, if there are more cores than the number of types of component to be parallelized, then if each of the types of component runs in a single core, there will be cores that are not used. Therefore, to maximize parallelism, data parallelism should also be employed to maximize the use of all cores. A highly data-parallelism design would make it easier to manage tasks that are sequential as there may only be race condition among the same type of data being parallelized but may not fully utilize the concurrency advantage for some components that are decoupled from each other. A highly task-parallelism design would cause some cores to be unused as there may be more cores than the number of different types of tasks that can run at the same time but having a synchronization stage with no mutex locking can be easily implemented if it is the synchronous model. Mixing task and data parallelism takes advantage of the fact that not all components and data objects of a game engine are completely dependent.

3.4 Synchronization

Synchronization with respect to multithreading is basically data synchronization. Synchronization is used to make sure that the same data are not executed at the same time by two threads. One method for synchronization is with the use of mutex (i.e., mutual exclusion). Use of a mutex locking in a game engine depends on the multithreading model. The main drawbacks with mutex locks are overhead, deadlocks, contention, and priority inversion [18].

In some cases, lockless algorithm can be used. In those cases, a game engine is designed so that mutex locking is entirely avoided. The easiest method is to have a synchronization stage where all processes must run in sequence. Another method is to use a message passing system between threads. This avoids the use of mutex locking when passing data. Some other synchronization techniques include reader-writer lock and read-copy-update [9].

4. Test Game Engine

We implement a multi-object game engine: Tower Defense Game (TDG) in our laboratory to investigate the impact of thread synchronization and data parallelism on multicore game engine performance.

4.1 Game Policy

The objective of the game is to defend a main base structure. The player has to build defensive structures that will destroy waves of enemies trying to destroy the main base structure. The player will try to survive as many waves as possible. Enemies will get harder after every wave. For every enemy destroyed, the player gains credits which could be used to build more defenses. Currently, there are three possibilities to terminate the game - (i) the player defends the main base structure for 3 minutes (the player wins the game), (ii) the main base structure is destroyed (the player loses the game), and (iii) abnormal termination.

4.2 Different Implementations

The video game engine is implemented using single-threaded model and various multithreaded models (with and without synchronization). We briefly explain different implementations below.

- *Single Threaded Model (STM):*
  The first operation is to capture input events and put it in a buffer; this is an input / output (I/O) operation with the operating system. The order of operations in the single threaded implementation is:
  1) Capture input
  2) Update input operation
  3) Update game logic
  4) Update AI
  5) Update physics
  6) Process navigational mesh updates
  7) Simulate physics
  8) Render graphics

- *Multithreaded Asynchronous Model (MAM):*
  There are two threads; each thread has independent clock cycle. The update graphics stage in this model reads the data buffered from the updates of the other thread. The order of operations is:
  Thread 1:
  1) Capture input
  2) Update game logic
  3) Update AI
  4) Update physics
  5) Process navigational mesh updates
  6) Simulate physics

Thread 2:

1) Update graphics
2) Render graphics

- *Multithreaded Synchronous Model (MSM):*
  In this lockless implementation, data synchronization is done in the serial stage only. In the parallel stage, all the operations run in parallel, each on a different thread. The order of operations is:
  Serial Stage:
  1) Capture input
  2) Update logic
  3) Update AI
  4) Update physics
  5) Update graphics

  Parallel stage: (One possible order)
  1) Process navigational mesh updates
  2) Simulate physics
  3) Render graphics

- *Multithreaded Synchronous Model with Data Parallelism (MSMDP):*
  This implementation is a combination of task and data parallelism using the multithreaded synchronous model. It is similar to the synchronous model but the physics will have 2 worker threads to process collision detection. In the physics simulation thread, two more threads are spawned during the collision detection stage. This is considered parallelism within a component. The order of the operations is:
  Serial stage:
  1) Capture input
  2) Update logic
  3) Update AI
  4) Update physics
  5) Update graphics

  Parallel stage: (One possible order)
  1) Update navigational mesh
  2) Simulate physics
  3) Perform collision detection on object batch 1
  4) Perform collision detection on object batch 2
  5) Render graphics

5. Experimental Results

In this work, we develop a multi-object game engine using C++ in a multicore computer to explore how the performance of a multicore game engine is influenced due to data parallelism and thread synchronization.

5.1 Important System Parameters

The workstation used in this experiment is an 8-core (dual-processor, quad-core per processor) system from Intel, runs at 2.13 GHz, and has 6 GB of RAM. The operating system used is the Linux Debian 6.0. Output parameters include: the number of frames generated per minute, processing time for each frame, processing time for each component, and speedup factor, S(P) = T(S)/T(P). Where, T(S) is the best sequential time and T(P) is the run time due to the parallel implementation.

5.2 Results and Discussion

We start with exploring the generation of diffirent frames by a single-threaded multi-object game engine. Then we study various multithreaded models of the game engine. We compare the performance and speedup due to various implementations: STM, MAM, MSM, and MSMDP.

First, we examine the number of frames generated due to various implementations for a 3-minute simulation of the game. As illustrated in Figure 1, the multithreaded synchronous model with data parallelism game console generates more frames per minute than any other consoles. Single-threaded game console generates about 4,750 frames per minute, multithreaded asynchronous and synchronous both generate about 5,000 frames per minute, and multithreaded synchronous with data parallelism generates about 5,050 frames per minute.

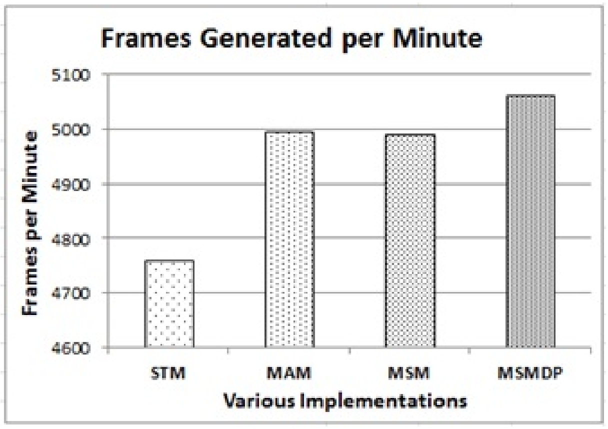


Fig. 1: Number of frames generated in one minute simulation of the Tower Defense Game.

Second, we investigate the maximum time required to process different frames. As shown in Figure 2, the single-threaded game console takes the maximum amount of time to process a frame when compared with other multithreaded models. MAM shows improvement over STM. Building a proper asynchronous multithreaded engine requires a lot of time and it will be more complex than a synchronized model in most cases. The asynchronous model of execution will most likely require more memory to implement and as such it is only recommended in cases where user experience can be improved by components running at their own clock cycle. It is also observed that multithreaded synchronous model with data parallelism console takes the minimum amount of time to process a frame.

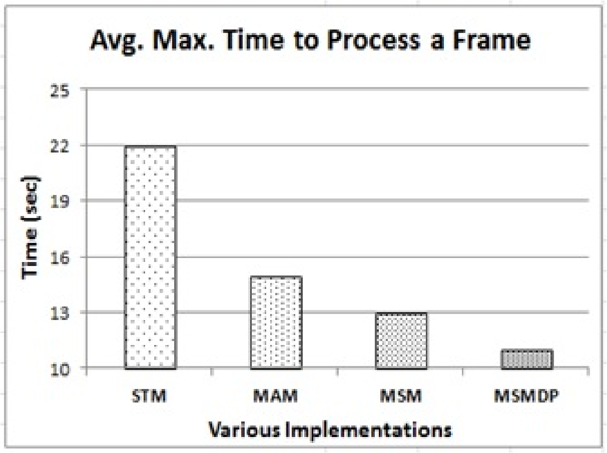


Fig. 2: The average maximum processing time for each frame.

Third, we study the maximum time required to process different components. As shown in Figure 3, the components are found to have different amount of time for processing. The physics component takes up most of the processing time; this is because the game uses a lot of physics objects and operations.

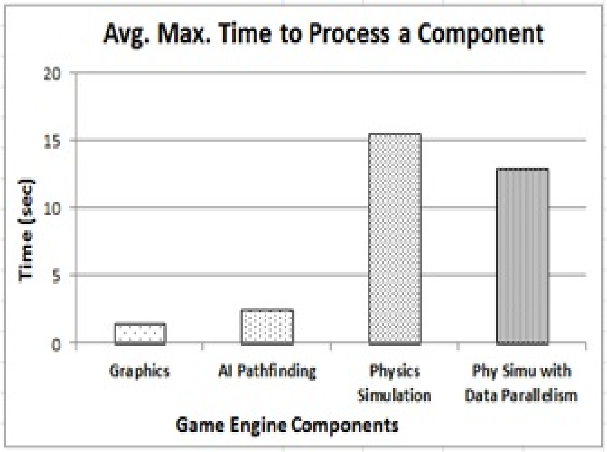


Fig. 3: The average maximum processing time for each component.

Fourth and finally, we observe the speedup due to the multithreaded implementations over the single-threaded implementation of the test game engine. According to the experimental results, the speedup increases as the number of threads increases (see Figure 4). More than 14x speedup is achieved due to the multithreaded synchronous model with data parallelism for 512 threads. This speedup is impressive. Future video games are expected to be much more complex than what we have today; speedup of such complex systems can be significantly increased by applying MSMDP like GPU computing. In GPU technology, thousands of threads

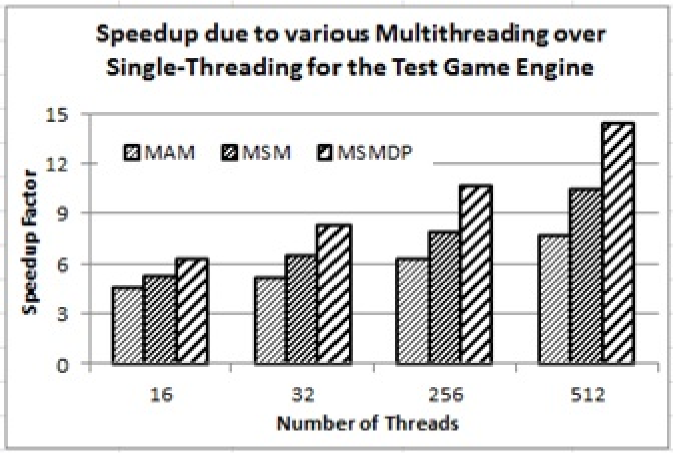


Fig. 4: Speedup due to multithreaded over single-threaded implementation of the test game engine.

can be generated for computation extensive systems; the threads are then run concurrently in parallel on a multicore CPU with manycore GPU system.

6. Conclusion

Single-processor multithreaded game engines struggle to improve performance due to the lack of hardware support. Recently introduced multicore systems have the potential to improve the performance of multithreaded game engines. However, programming multithreaded game engines for multicore architectures introduces various challenges including data parallelism, thread synchronization, task parallelism, and load balancing. In this work, we explore the challenges due to thread synchronization and data parallelism by developing a multicore video game console (called Tower Defense Game). We develop C++ programs for single-threaded and several multithreaded models with and without data parallelism. Experimental results support the fact that the multithreaded models outperform the single threaded model. Although different game components take different amount of processing times (see Figure 3), the multithreaded synchronous model with data parallelism generates more frames and takes less amount of time to process the frames (see Figures 1 and 2).

On an 8-core system, the speedup due to multithreaded synchronous program with data parallelism is about 14 (see Figure 4) with respect to the single-threaded implementation. It is expected that a much higher speedup will be needed for future game engines, which can be achieved by applying concurrent/parallel programming techniques like GPU computing.

We plan to implement the entire test game engine (i.e., Tower Defense Game) on a CPU/GPU platform to explore the performance and power consumption in our next endeavor.

References

[1] S. Berberich, "Video games starting to get serious," http://ww2.gazette.net/stories/083107/businew11739-32356.shtml, 2007.

[2] A. Asaduzzaman and I. Mahgoub, "Cache modeling and optimization for portable devices running mpeg-4 video decoder," pp. 239–256, 2006.

[3] F. Feinbube, P. Troger, and A. Polze, "Joint forces: From multi-threaded programming to gpu computing," *IEEE Software Journal*, vol. 28, no. 1, p. 51, 2011.

[4] J. Tulip, J. Bekkema, and K. Nesbitt, "Multi-threaded game engine design," *IE '06 Proceedings of the 3rd Australasian conference on Interactive entertainment*, pp. 9–14, 2006.

[5] A. Asaduzzaman, F. N. Sibai, and H. Elsayed, "Performance and power comparisons of mpi vs pthread implementations on multicore systems," *9th International Conference on Innovations in Information Technology (IIT)*, 2013.

[6] "Multithreading problems," http://www.roguewave.com/portals/0 /products/threadspotter/docs/2011.2/manual-html-linux/manual-html/multithreading-problems.html, 2011, rogue Wave Software.

[7] A. D. Vries, "Multithreaded renderloop," http://blog.slapware.eu/game-engine/programming/multithreaded-renderloop-part1/, 2013.

[8] A. E. Rhalibi, D. England, and S. Costa, "Game engineering for a multiprocessor architecture," School of Computing and Mathematical Sciences, Liverpool John Moores University, 2005.

[9] L. Davies, "Practical examples of multi-threading in games," pp. 4–30, 2006, intel.

[10] "Threading building blocks (TBB)," http://www.threadingbuildingblocks.org, 2011.

[11] K. Gadd, "Threading and your game loop," http://www.altdevblogaday.com/2011/07/03/threading-and-your-game-loop/, 2011.

[12] D. Andrade, B. B. Fraguela, J. Brodman, and D. Padua, "Task-parallel versus data-parallel library-based programming in multicore systems," *International Conference on Parallel, Distributed and Network-based Processing*, pp. 101–110, 2009.

[13] V. Monkkonen, "Multithreaded game engine architectures," http://www.gamasutra.com/view/feature/130247/multithreaded-game-engine.php?, p. 3, 2006.

[14] R. Kriemann, "Implementation and usage of a thread pool based on POSIX threads," *In Max-Planck-Institute for Mathematics in the Sciences*, pp. 22–26, 2004.

[15] M. Guevara, C. Gregg, K. Hazelwood, and K. Skadron, "Enabling task parallelism in the."

[16] L. Baumstark and L. Wills, "Exposing data-level parallelism in sequential image processing algorithms," in *In Proceedings of the Ninth Working Conference on Reverse Engineering*, 2002, pp. 1095–1350.

[17] J. S. Harbour, *Multi-Threaded Game Engine Design*. Course Technology PTR (1st ed.), 2010.

[18] E. Cronin, A. R. Kurc, B. Filstrup, and S.Jamin, *An Efficient Synchronization Mechanism for Mirrored Game Architectures*. Kluwer Academic Publishers, 2003, extended Version.

Multi-Gbps Fano Decoding Algorithm on GPGPU

Ozgur Ates, Selcuk Keskin and Taskin Kocak
Department of Computer Engineering, Bahcesehir University, Istanbul 34353, Turkey

Abstract— *The bandwidth requirements for the next-generation wireless applications are increasing. The newest standards such as the WirelessHD aim to transmit signals at high speed in the range of multi-Gigabit per second (Gbps). At this rate, the processing effort of the baseband signals becomes challenging. In this paper, we propose to use GPGPU for parallel processing to offer multi-Gbps throughput for a sequential convolutional decoding algorithm; namely, the Fano algorithm. NVIDIA's latest Kepler architecture based K20c GPU and their CUDA programming platform are used. Some algorithmic and CUDA-based optimizations are developed to achieve a throughput of 4.6 Gbps.*

Keywords: Fano algorithm, CUDA, high throughput decoding

1. Introduction

Convolutional coding is a subject that can be said to have started with P. Elias [1]. Inspired by Shannon's mathematical material in communication [2] and Hamming's paper on error-correcting code [3], Elias introduced the concept of convolutional coding. This well-known mechanism in telecommunication is used in signal transmission over a noisy channel. Decoding a signal is one of the most time consuming tasks done at the baseband along with the fast Fourier transform (FFT).

Elias's algorithm is characterized by its relative simplicity compared to Fano algorithm, another sequential decoding algorithm, or compared to Viterbi which can be visualized as a search for the shortest path through a trellis diagram. Zigangirov [4] and independently by Jelinek [5], they proposed their implementations of the stack algorithm. Viterbi introduced the convolutional decoding known as the Viterbi algorithm [6]. This is an algorithm which finds the shortest path for a weighted graph [7]. The method used by the algorithm is proven to obtain the maximum-likelihood decoding (MLD) of a transmission with inter-symbol interferences [8]. However, the algorithm is notorious for having an exponential complexity growth in correlation with a long constraint length.

Due to the availability of several GHz of unlicensed bandwidth around 60GHz, multi-gigabit per second wireless communications is drawing a lot of research attention [9]–[11]. Several international standard organizations and associations of industrial partners are working to define specifications for millimeter-waves systems operating in the 60 GHz band. The WirelessHD consortium is an industry-led effort to define a worldwide standard specification or the next-generation wireless digital network interface specification for consumer electronics and personal computing products. Due to the increase in bandwidth, multi-gigabit decoding in wireless communications has become a challenging task. ASIC/FPGA based sequential decoders have been proposed; however, a hardware solution is not flexible and easy to change with new standards. In this paper, we present design and implementation of the Fano decoding algorithm as a software solution on Graphics Processing Units (GPU) that can support the high-throughput requirements. A GPU provides a parallel architecture, which combines raw computation power with programmability [12].

GPUs have only been used for 3D graphics rendering in the first years of their evolution. With the advent of technology of GPUs offer high performance of general purpose processing by executing thousands of threads simultaneously. GPU provides extremely high computational throughput by employing many cores working on a large set of data in parallel. Compute Unified Device Architecture (CUDA), developed by NVIDIA, is a widely used programming approach in massively parallel computing applications [13].

This paper is organized as follows: Section 2 gives a brief overview of Fano decoding and introduces communication environments. Section 3 gives the basic design of the GPGPU-based decoding algorithm. Section 4 explains the optimizations taken to achieve multi-Gbps decoding speed. The experimental results are given in Section 5. Finally, conclusions are drawn in Section 6.

2. Fano algorithm

The Fano algorithm is a tree searching algorithm characterized by a good performance with low average complexity at reasonably high signal-to-noise ratio (SNR) [14]. The algorithm is a sequential decoding algorithm. Fano does not claim to propose the maximum likelihood decoding schema as in the Viterbi algorithm. However, for only slight decoding accuracy deterioration, the proposed algorithm obtains a nearly optimal decoding performance with significantly less decoding effort. The tree is composed of branches and nodes. Each branch of the tree has a weight that is also called branch metric. Paths are sequences of branches. The weight of a path is simply the sum of all of the metrics of its branches.

This is a search algorithm in the sense that the search for the minimum weight for a path is conducted from the root to a leaf. The search is sequential and is done from one node to its neighbouring nodes and so on. The algorithm is a depth-first tree-searching algorithm. The search goes on as long as the current node is not a leaf node. The Fano algorithm moves forward if a branch metric is above a certain threshold T. On the other hand, it moves backward if there is no possible forward move and searches for other branch candidates. In case neither is possible, T is tightened. The threshold T is updated based on the branch metrics statistics. T is initially selected as a multiple of delta which is the threshold increment. The workflow of the algorithm can be seen in Fig.1.

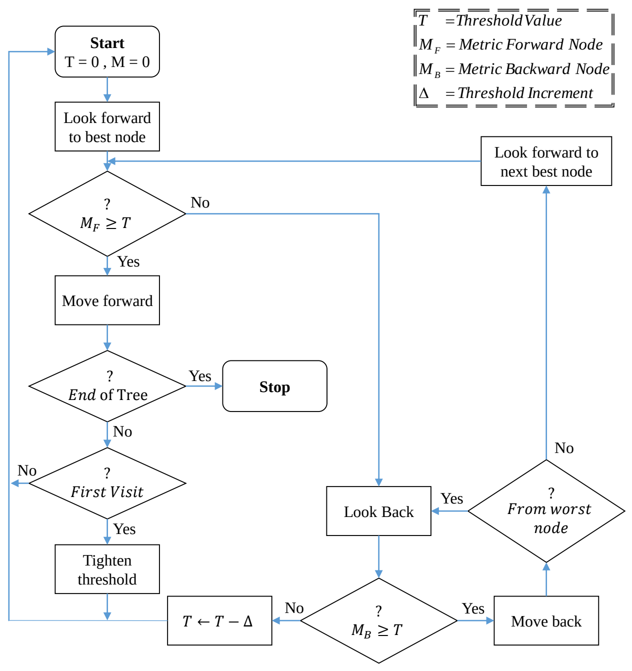


Fig. 1: Flow chart of the Fano algorithm

We use the Fano bit metric, introduced in [15] and [16]:

$$M(y|x) = log_2 p(y|x) - log_2 p(y) - R, \qquad (1)$$

where $p(y|x)$ is the conditional probability density function(pdf) of the received symbol y given the transmitted symbol x, $p(y)$ is the marginal pdf of y, and R is the code rate.

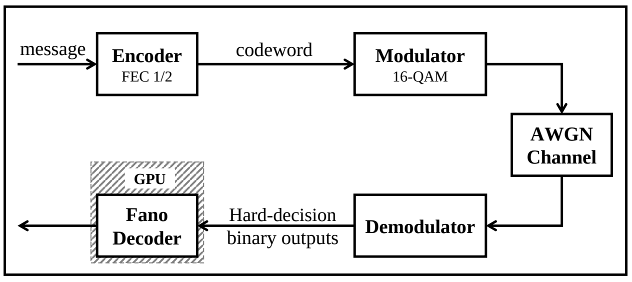


Fig. 2: Communication System Architecture

We implemented the communication system shown in Fig. 2 in Matlab environment. We focus on modulation and coding schemes for the WirelessHD specifications [17]. The code rate is $R = 1/2$. The modulation is 16-QAM, and an AWGN channel was assumed, and the output of the demodulator was 1-bit hard-decision.

A binary convolutional encoder is conveniently structured as a mechanism of shift registers and modulo-2 adders, where the output bits are modular-2 additions of selective shift register contents and present input bits. Selected shift registers are chosen according to generator polynomials which are $g0 = \{155\}_8$ and $g1 = \{117\}_8$, and the constraint length is $K = 7$.

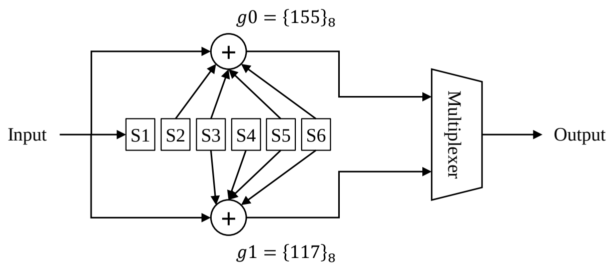


Fig. 3: Convolutional encoder FEC rate 1/2

3. Fano algorithm on NVIDIA GPGPU

3.1 GPGPU Architecture

A GPGPU based on Kepler architecture (GK-codenamed chip) consists of next generation streaming multiprocessors (SMX), and each of them has stream processors (cores). A Kepler GK110 in Tesla K20c implementation includes 13 SMX units and six 64-bit memory controllers. The SMX processing core architecture of the K20c can be seen in Fig. 4. As seen in the figure GPU has 192 single-precision CUDA cores, 64 double-precision units, 32 special function units (SFU), and 32 load/store units (LD/ST).

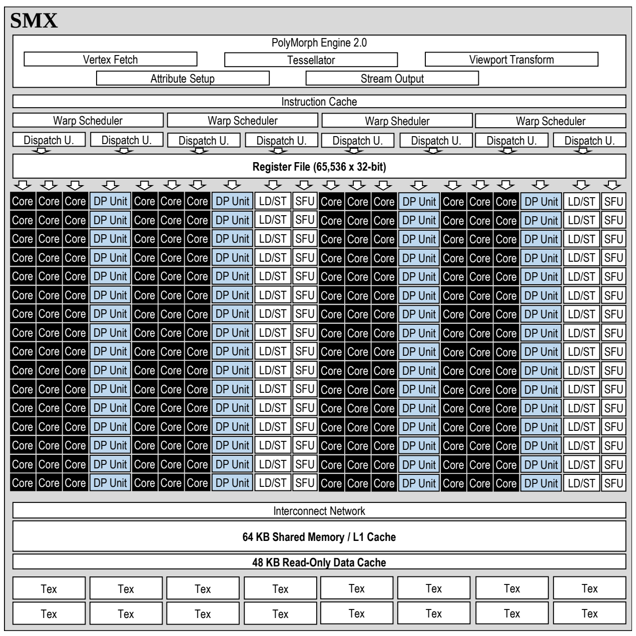


Fig. 4: Streaming Multiprocessor (SMX)

Each SMX has 64 KB L1 Cache which can also be used as shared memory. Global memory of GPU is an off-chip memory. The SMX can access the global memory, but access time is slow.

3.2 GPGPU Coding

In November 2006, NVIDIA introduced CUDA, a general purpose parallel computing platform and programming model that leverages the parallel compute engine in NVIDIA GPUs to solve many complex computational problems in a more efficient way than on a CPU. CUDA comes with a software environment that allows developers to use C as a high-level programming language. CUDA provides a simple path for users familiar with the C programming language to easily write programs and execute them on their devices. It consists of a set of extensions to the C language to allow direct access to the board and runtime libraries. CUDA extends C by allowing programmers to define C functions, called kernels, which, when called, are executed N times in parallel by N different CUDA threads, as opposed to only once like regular C.

CUDA threads may access data from multiple memory spaces during their execution. Each thread has private local memory. Threads gathered as blocks having shared memory visible to all threads of their block and with the same lifetime as those blocks. All threads have access to the global memory. There are also two additional read-only memory spaces accessible by all threads: the constant and texture memory spaces. The multiprocessor creates, manages, schedules, and executes threads in groups of 32 parallel threads called warps. An instruction that accesses addressable memory (i.e., global, local, shared, constant, or texture memory) might need to be re-issued multiple times depending on the distribution of the memory addresses across the threads within the warp.

Fano implementation in CUDA was selected because it allows programmers to use the board as a parallel execution platform. Also, we no longer depend on the processing power of the CPU which is heavily controlled by the OS. Kepler GK110 supports the new CUDA Compute Capability 3.5. The following Table 1 summarizes parameters of Compute Capability for Kepler GPU architecture.

Table 1: Compute Capability of Kepler GPU

	KEPLER GK110
Compute Capability	3.5
Threads / Warp	32
Max Warps / Multiprocessor	64
Max Threads / Multiprocessor	2048
Max Thread Blocks / Multiprocessor	16
32-bit Registers / Multiprocessor	65536
Max Registers / Thread	255
Max Threads / Thread Block	1024
Shared Memory Size Configurations (KB)	16/32/48

CUDA is dependent on the architecture of the GPU to be used in. Knowing that architecture, it may allow further optimizations. Memory access is the limiting factor of parallel programming. We may have as many threads as we want, they may be as quick as possible but processing power is useless if the data transfer is slow. In regular processing, we have to wait for the memory to be initialized, and then we have to wait for the result to be calculated. In parallel processing, we may multiply the number of cores working and make the processing strength virtually unmatched. However, we still have to send the data to be processed to the memory and get the result from that memory. Since the processes all run at the same time, their synchronization may also become a problem.

3.3 GPGPU Implementation

The Fano algorithm is a sequential algorithm, thus it is not suitable for data-level parallel implementation. In other words, multiple threads cannot cooperate to work on a common codeword in parallel. Only task-level parallelism in which threads work independently in different set of codewords is possible in the Fano algorithm. Threads in GPU calculate the codewords individually as shown in Fig. 5. One demodulated codeword is (200+6)*2=412 $bits$ where 200 input $bits$, 6 trailing $bits$ and 2 comes from the code rate. After calculation, the output is 200 $bits$ which is the same as the input. After some experiments, 128 threads on each SMX were chosen in order to occupy the most of GPU's resources. Consequently, the most effective decoding could be achieved at high occupancy.

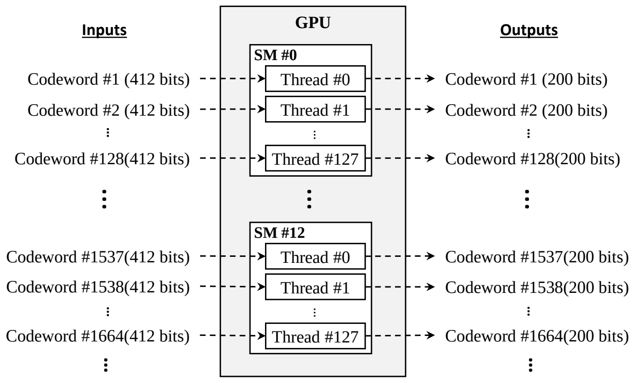


Fig. 5: Parallel calculation of codewords

The Fano algorithm is implemented on the GPU as depicted in Fig. 6. The demodulated bits are bound to the texture memory of the board. This was motivated by the fact that we need a big memory space to hold inputs and texture memory has an access pattern adequate for the algorithm. The memory accesses have spatial locality as such the immediate adjacent memories are cached. The output is directed to global memory. Each iteration of the algorithm is characterized by determining the metric, state and output corresponding to the current step. This part was calculated before running the kernel since the number of states and inputs are well known and common throughout the algorithm. These common values are put in constant memory.

Once these precondition variables are found, their values are stored in registers. Depending on those values, the algorithm decides on making a move forward or backward. Current steps and previous ones are also stored in registers. The algorithm continues until all inputs are decoded or an overflow is encountered.

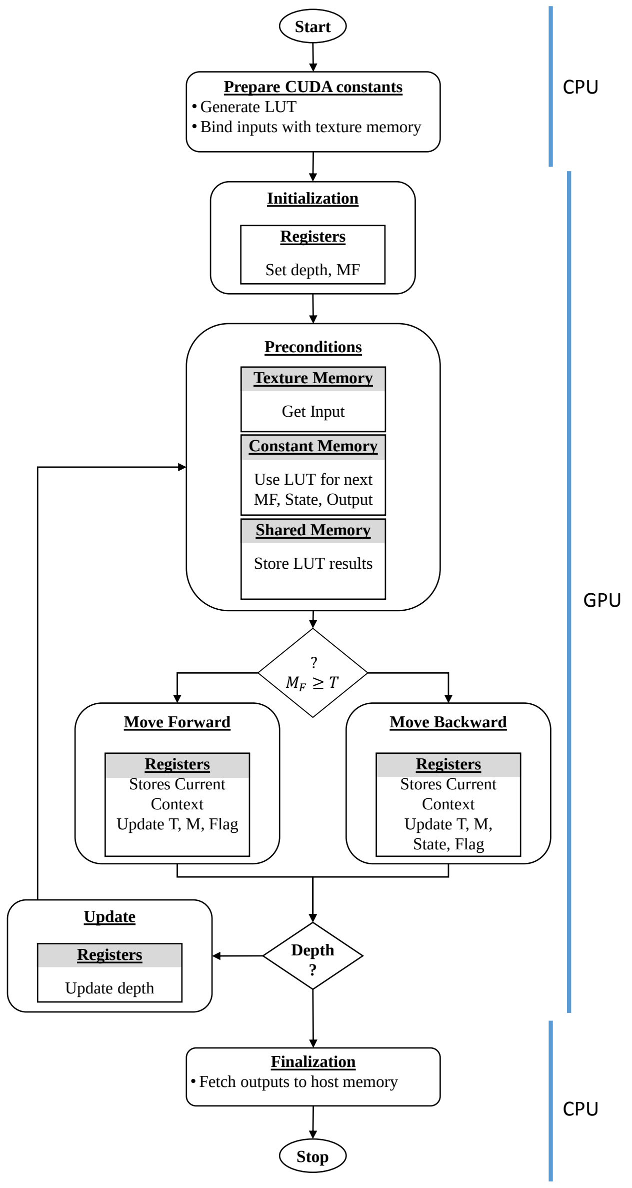


Fig. 6: Flow chart of GPU Implementation for the Fano algorithm

4. Optimizations

4.1 Queue usage optimization

In memory bound applications such as the one in this work, memory access is a major bottleneck of parallel computing. Fano algorithm stores previously visited depths with their respective contexts, that is, metric, state, LFNB flag. All these three parameters occupy 16 bits. While the operations to be done at each step are trivial, the spatial-temporal requirement is huge. Considering the WirelessHD frame has a depth of 206 in total (200 input and 6 trailing bits), it required four historic arrays of 206 elements. Tests were carried on to find a correlation between memory usage and its actual implementation. This statistical analysis was conducted on testing back tracings to find the their impact and mechanism. This revealed that back tracing is very local, generally one or two steps back. The analysis also showed that the all back traces for successfully decoded vectors were 16 or less. While the array implementation fills 206*16=3296 *bits*, the queue implementation only requires sixteen contexts at a time meaning that it only takes 16*16=256 *bits* in the memory.

4.2 Memory optimization

Global memory has a high latency, it is a slow access memory. Hence, the use of registers and shared memory were prioritized whenever possible. However, such memory types are small, respectively 255 registers per thread and 48, 32 or 16 kilobytes per block defined at compile time by the programmer. In this manner, the biggest problem is working with inputs and outputs. The Fano algorithm is a sequential algorithm that advances from one depth to one of its neighbouring depths until the codeword is completely decoded or an overflow condition is met. We used this feature to fetch a new chunk of inputs at each new sixteen depth which the Fano algorithm is designed to generate one output bit per iteration. Its CUDA implementation needed a variant such that we also used partial outputs as depicted in Fig. 8. This allowed the algorithm to work on registers rather than global memory.

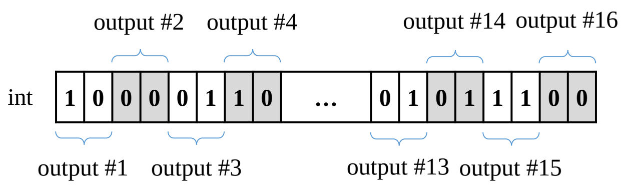


Fig. 7: Input/Output representation

Fig. 7 illustrates the *int* data type being used as a block of 16 outputs generated by 32 inputs. Each demodulated codeword is composed of 206*2=412 *bits*, thus we are able to represent a codeword with $\lceil 412/32 \rceil$=13 *int* variables. Once the decoding process was completed, the output was again 206 *bits* with the last 6 *bits* being its trailing bits. The output was represented with $\lceil 206/32 \rceil$=7 *int* variables. Storing 20 *int* variables during decoding caused CUDA to use a lot of registers. Accordingly, the occupancy became low and decoding process took long. The memory was partitioned as described in Fig. 8. Thanks to this method, only two *int* variables, one for the current input block and another one for the output block, were used at the same time per kernel. Processed inputs and outputs were dispatched into shared memory that was used as an extra storage space. Input and output blocks were connected in such a way that a new output block was taken every two times a new input was requested due to $coderate = 1/2$. Therefore, an *int* would hold 1*16 inputs and 2*16 outputs. Updating mechanism is done when the algorithm needs to increment of decrement the current depth.

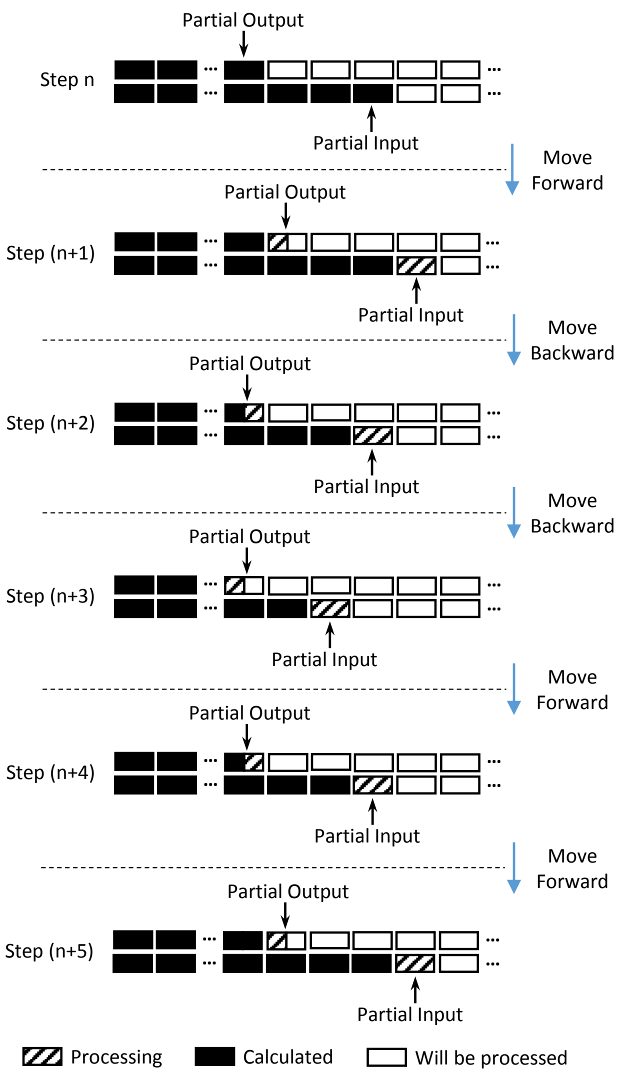


Fig. 8: Partial input/output representation

4.3 Look-up tables

The Fano algorithm systematicaly computes at each iteration the next state, metric values and the look for next bit (LFNB) flag bit. The total effort is about 30 assignments and 35 ALU operations. The input to this operation is the current state and input bits. To avoid any unnecessary calculation, a look-up table (LUT) was prepared. Since there were 64 states and 4 possible inputs bit combinations at each iterations, next state, output, metric, values, LFNB bit were pre-calculated and were stored in this LUT. This redundant and computationally heavy step was changed with a mere memory read. A somehow similar idea is also found in another paper about Bidirectional Fano Algorithm in which, two decoders process in pair [18]. In there, the authors used LUT for deciding whether a merged state was detected. In the current paper, we have six successive stages constituted by the logical bit output from the shift registers of the encoder. Hence, each stage constitutes a 6-bit string. The total number of combinations is $2^6 = 64$; thus, the total size of the look-up table is 64*4*32=8192 $bits$.

The occupancy analysis of the Fano decoder algorithm with the mentioned optimizations can be observed on Table 2. The end result is that SMX fully occupied with these three changes.

Table 2: CUDA analysis

	Theoretical	Device Limit
Occupancy per SMX		
Active Blocks	16	16
Active Warps	64	64
Active Threads	2048	2048
Occupancy	100.00	100.00%

5. Experimental results

The Fano algorithm for GPGPU was implemented on CUDA 5.0 environment [19] and executed on NVIDIA's Tesla K20c board. All the experiments were conducted on a 3.2 GHz Intel Core i7-960 processor with 12 GB DDR3 of memory.

The testings revealed that GPGPU based implementation of the decoder gave a throughput 1.8 Gbps at the lowest Eb/No, 2 dB, and 4.6 Gbps for 8 dB. The WirelessHD specification requires a speed of 2.8 Gbps a modulation of 16-QAM modulation and 1/2 code rate. The throughput analysis in Fig.9 reveals that the algorithm can support the required throughput from Eb/No value of 4.5 dB and onwards.

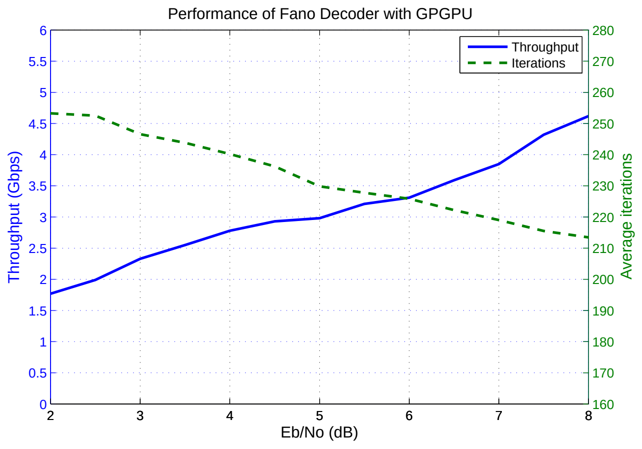


Fig. 9: Throughput and average number of iterations vs Eb/No

In the same figure we may note that the most noisy signals, 2 dB and 2.5 dB require fairly similar decoding effort with 260 iterations. This is explained by the fact that, at those Eb/No, like all decoders, Fano algorithn have a lot of trouble decoding. This translates in the algorithm to return back into previous steps on move backs. This makes decoding use more iterations for lower Eb/No compared to higher Eb/No decoding. On those higher Eb/No values, the algorithm decodes at an average of 213 to 220 iterations for 8 dB and 7 dB, respectively. As the Eb/No value increases, less number of iterations are required, therefore less back tracing occurs. Note that there is a negative correlation between the number of iterations and the throughput.

WirelessHD works at 60 GHz spectrum, For this very reason, the signal will have low probability to travel through walls and obstructing objects. Therefore, we can assume the transmitter to be at relatively close proximity of the decoder. This allows the receiver to work at a relatively low SNR range (2-8 dB) rather than (12-20 dB) since the information will receive low or negligeable distortion. Therefore, we can mainly operate at Eb/No=8 dB resulting on low bit error rate, that is, between 10^{-5} and 10^{-6}.

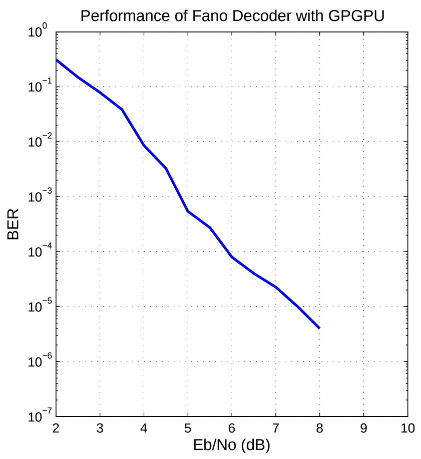


Fig. 10: BER vs Eb/No

The Fig. 11 shows how the number of codewords (batch size) influences the throughput of the algorithm. In GPGPU programming, the kernel initialization has a computing overhead. To mitigate this initialization problem, we need to process as many codewords as possible at the same time. As expected, the highest speed was reached at Eb/No with 4.6 Gbps for 131072 codewords.

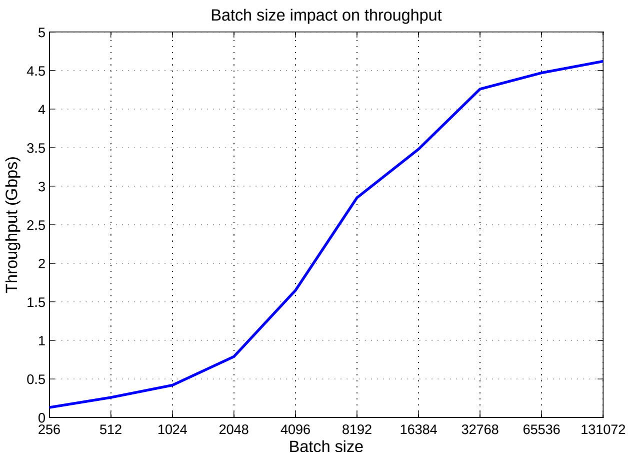


Fig. 11: Throughput vs the number of codewords

6. Conclusions

In this paper, a GPGPU based implementation of the Fano algorithm is explored to provide multi-Gbps. Look-up tables were used to avoid redundant calculations. Also, instead of using the classical Fano algorithm implementation with a complete historic record, its sub-section with previous 16 steps records were used in a queue. This drastically reduces the memory requirement of the algorithm. Experimental results showed a throughput of 4.6 Gbps and an error rate of less than 10^{-5} at relatively low SNR values.

References

[1] P. Elias, "Coding for noisy channels," *IRE Convention Record*, vol. 4, pp. 37–46, 1955.
[2] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, pp. 379–423,623–656, 1948.
[3] R. Hamming, "Error Detecting and Error Correcting Codes," *The Bell System Technical Journal*, vol. 29, pp. 147–160, 1950.
[4] K. Zigangirov, "Some sequential decoding procedures," *Problemy Peredachi Informatsii*, vol. 2, no. 4, pp. 13–25, 1966.
[5] F. Jelinek, "Fast Sequential Decoding Algorithm Using a Stack," *IBM J. Res. Dev.*, vol. 13, no. 6, pp. 675–685, 1969.
[6] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.
[7] J. Omura, "On the Viterbi decoding algorithm," *IEEE Trans. on Information Theory*, vol. 15, no. 1, pp. 177–179, 1969.
[8] G. Forney, "The Viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, 1973.
[9] (2013) Status of Project IEEE 802.11 VHT Study Group. [Online]. Available: http://www.ieee802.org/15/pub/TG3c.html
[10] (2013) IEEE 802.15 WPAN Task Group 3c (TG3c) Millimeter Wave Alternative PHY. [Online]. Available: http://www.ieee802.org/15/pub/TG3c.html
[11] H. Singh, J. Oh, C. Kweon, X. Qin, H.-R. Shao, and C. Ngo, "A 60 GHz wireless network for enabling uncompressed video communication," *IEEE Communications Magazine*, vol. 46, no. 12, pp. 71–78, December 2008.
[12] J. B. Srivastava, R. Pandey, and J. Jain, "Implementation of Digital Signal Processing Algorithm in General Purpose Graphics Processing Unit (GPGPU)," *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 1, no. 4, pp. 1006–1012, 2013.
[13] D. B. Kirk and W.-M. W.Hwu, *Programming Massively Parallel Processors*, 2nd ed. Morgan Kaufmann, 2012.
[14] R. Fano, "A heuristic discussion of probabilistic decoding," *IEEE Trans. on Information Theory*, vol. 9, no. 2, pp. 64–74, 1973.
[15] S. Lin and D. Costello, *Error Control Coding*. Prentice Hall, 2004.
[16] Y. S. Han and P.-N. Chen, *Sequential Decoding of Convolutional Codes*. John Wiley & Sons, Inc., 2003. [Online]. Available: http://dx.doi.org/10.1002/0471219282.eot348
[17] (2013) Wireless High-Definition (WirelessHD). [Online]. Available: http://www.wirelesshd.org
[18] R. Xu, T. Kocak, G. Woodward, and K. Morris, "Throughput Improvement on Bidirectional Fano Algorithm," in *Proceedings of the 6th International Wireless Communications and Mobile Computing Conference*, 2010, pp. 276–280.
[19] (2013) The CUDA Programming Guide. [Online]. Available: https://developer.nvidia.com/category/zone/cuda-zone

Radiation Therapy Optimization with GPU Programming

David Allen[1], **Ovidiu Daescu**[1]
[1]School of Engineering and Computer Science, The University of Texas at Dallas
Richardson, TX, United States

Abstract— *Parallel programming techniques lend themselves well to a variety of biological and medical related tasks. In this paper, we show an application of GPU programming to the optimization of treatment plans for cancer. We are specifically focused on a class of radiotherapy called Intensity Modulated Radiation Therapy or IMRT. Radiation therapy works by exposing cancer cells to a clinically prescribed dose of radiation, causing them to die out. However, care must be taken to avoid applying too high of a dose to the healthy tissues. Optimization of treatment plans involves reducing the amount of radiation absorbed by healthy tissues while ensuring that the tumor receives a high enough dose for the treatment to be effective. We demonstrate how to apply parallel programming techniques to improve the efficiency of an algorithm we previously developed to improve the quality of radiation treatments by reducing healthy tissue exposure to the radiation.*

Keywords: Parallel Programming, GPU Programming, Computational Geometry, Kinetic Data Structures, Radiation Therapy

1. Introduction

Developing efficient and optimal strategies for the construction of medical treatment plans is a rising concern given recent advancements of modern medicine. As the demand for timely medical services increases with a growing and longer lived population, the need for automated and semi-automated methods to quickly formulate high quality treatment plans has grown to relieve the added burden on medical professionals. Many of these computational solutions involve algorithms that lend themselves well to parallel programming optimizations. We describe one such optimization to a technique we have previously developed for use in developing improved radiation therapy treatment plans for cancer patients.

1.1 Radiation Therapy Treatment

Radiation therapy is one of the primary treatments used by doctors to combat cancer. The intervention works by applying a dose of radiation to cancerous cells, which are unable to repair the damage the radiation does to them and subsequently die out. To cause enough damage to the diseased cell a clinically determined radiation dose threshold must be reached during the course of treatment. Healthy tissues and organs that surround the tumor or are in the path of the radiation will also receive some radiation does as well. Although healthy cells are better able to repair themselves from the damage caused by the radiation exposure, it is ideal to keep the dose received by these healthy tissues as low as possible to avoid causing the patient other health complications. Clinicians determine the maximum dose that each healthy tissue and organ can receive before the cost of damaging the patient's vital organs outweighs the benefit of the treatment. Some organs are more susceptible to radiation damage than others, or are more highly critical to the patient's overall health. These are called **Organs at Risk** or OARs. When developing treatment plans clinicians must take care to ensure that the tumor receives the amount of radiation needed for the treatment to be effective while protecting the OARs and other healthy patient tissues from absorbing too high of a dose. This is often complicated by the fact that these targets are often moving within the patient's body as the patient experiences involuntary motions such as breathing.

Although radiation therapies come in many forms, we focus on a specific technique throughout the rest of this paper. **Intensity Modulated Radiation Therapy** or IMRT is a type of radiotherapy that involves using a linear accelerator (or linac) to fire a beam of ionizing radiation at the target tumor. The beam is mounted on a movable gantry and can be rotated around the patient to fire at the target from several different angles, which ensures that the tumor receives a conformal dose of radiation but the healthy tissues in the path from the beam source to the target are spared. Another way in which the healthy tissues are protected is by using a multi-leaf collimator or MLC to shape the beam to the tumor shape. The MLC consists of several pairs of metal plates or "leaves" that can open and close to block sections of the source beam. By moving the leaves to approximate the shape of the target tumor in the beam's eye view (BEV), the healthy tissues surrounding the tumor are shielded and thus the quality of treatment is improved. Work related to the optimization of IMRT treatment plans can be found in: [1], [2], [3], [4], [5], [6], [7], [8], [9].

One of the challenges in optimizing IMRT treatment plans is handling the previously mentioned motion of the patient. As the patient breathes the tumor may shift in position, or the organs around the tumor may shift. Also, the patient may move unexpectedly during treatment. Methods that have been developed to handle the patient's motion involve using respiratory gating [10], [11], [12], which involves finding the time that the tumor moves into the radiation beam's range.

These methods develop a "strike zone" used to determine when to turn the radiation on or off. Having the patient perform breathing exercises [12] is another method designed to create a breathing pattern that can be reproduced by the patient during treatment, improving predictability of the breathing motion.

We have developed what is to the best of our knowledge a novel method optimizing treatment plans with patient motion. Our method involves finding the point in time during which the target tumor is the least obstructed by other organs in the beam's eye view. We can project the 3D shapes of the patient's organs and the tumor onto the 2 dimensional field of the BEV as the 2D cross sectional areas (see Figure 1). By observing how these shapes move in time, we can find the time interval during which the cross sectional contour of the tumor overlaps least with the cross sections of the other healthy tissues. This would be an optimal time to turn on the MLC shaped radiation beam, as the beam would penetrate fewer healthy tissues to reach the target.

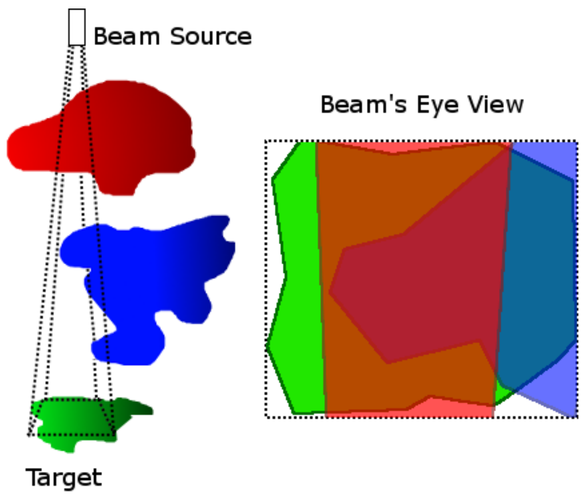


Fig. 1: Viewing the target and surrounding tissues and organs from the radiation source beam's viewpoint. The 3D objects are projected as 2D cross sections onto the 2D viewing plane

1.2 Previous Work

Our previous work on GPU programming involved the design of algorithm development models [13] and applying GPU programming to the image template matching problem [14]. We have also used GPU parallel programming techniques to optimize treatment simulations for IMRT [15]. This work only focused on improving the run time efficiency of an IMRT treatment plan simulation by parallelizing the distribution of radiation dosages to each voxel in the CT volume (obtained from imaging the patient's organs and the tumor) of the area of the patient's body being treated. The change in dose for each voxel must be recalculated at each time step for many voxels, but the dose increase received by each voxel at each step of the simulation is independent of the others. This made parallelization the obvious choice for speeding up dose calculations. The contribution of this paper is to improve the speed of a technique used to optimize the quality of IMRT treatments (which we first proposed in [16]), using similar GPU parallel programming techniques. Contrary to our previous work with IMRT GPU programming optimizations, we are not simply improving the efficiency of a simulation, but improving an algorithm designed to directly optimize IMRT treatment plans.

2. Materials and Methods

2.1 Kinetic Data Structures

In order find the time of minimal overlap, we developed a data structure and algorithm based on **Kinetic Data Structures** (KDS) [17], [18], [19]. A KDS is a data structure that maintains a set of relations called *certificates* that provide a proof of correctness for some underlying property (called the *configuration function*) of a system of moving geometric objects. A KDS can be used to maintain the convex hull of two moving polygons [20], or the set of nearest neighbors of a set of moving points [21], and polygon collision [22], [23], [24]. In our case, we used a KDS to maintain the structure of the intersection or overlap regions of multiple convex polygons, which are used to represent an approximation of the cross sectional slices of the target tumor, OARs and other regions of interest in the beam's eye view of the radiation source in an IMRT treatment.

We are able to make the assumption that these polygons are convex because if one of these cross sectional areas possesses a non-convex shape, it can be easily preprocessed to decompose it into a set of convex shapes. As these polygons move in time, the structure of the intersection between the target (the polygons representing the tumor being treated) and the polygons representing other organs and tissues ("non targets") will change. For clarity, we will only consider one target T and non-target S when describing the KDS data structure. There may be many targets and non-targets, but ultimately we are only interested in the regions of intersection between any individual target, non-target pair. These regions can be computed independently, and their areas are computed and summed to obtain the total area of the intersecting regions, which corresponds to the area of overlap between the cross sections of the tumor and other organs, which is the function we wish to minimize.

Let S and T be two convex polygons that intersect, and let this intersection region be R. The intersection of S and T is itself a convex polygon. This polygon is defined by a set of vertices that are one of three types: vertices on the boundary of S but inside of T, vertices on the boundary of T that are inside of S, and vertices representing the intersection of the boundaries of S and T. It follows that whenever a polygon vertex from S or T enters or leaves the interior of the opposing polygon, the set of vertices that make up R is changed.

This transition from inside to outside can only occur at times when a polygon vertex overlaps the boundary of the opposing polygon. For example, a polygon vertex on S could "collide" with an edge on T as S moves in time. At this point of time, t, the vertex could be either entering or leaving the interior of T depending on the directions and magnitudes of the velocities of S and T and the previous positions of the vertices and edges of the system. A vertex and edge are colliding if the vertex's position lies on the line segment of the edge, or in other words when the shortest distance between the vertex position and the edge line segment is zero. The certificates that make up our KDS are the set of distances between all S vertices and T edges, as well as all T vertices and S edges. Whenever one of these distances becomes zero as the polygons move in time, a collision has occurred and the configuration function must be updated accordingly to maintain the correctness of the data structure representing the intersection region R. In order

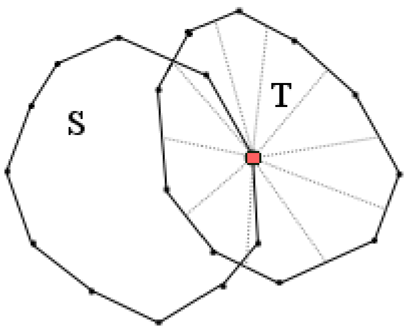


Fig. 2: In order to find all collisions, every vertex in S must have a certificate with every edge of T, and vice versa. The dotted lines represent the distance functions between the circled vertex on S and the edges of T that must be solved for time t

to determine when these collisions occur we must know how the polygons will move in time. The **flight plan** of a polygon describes it's motion throughout the time interval that is under consideration. We represent this flight plan as a list of velocity and duration time pairs. The polygon moves according to the first velocity in the list for an amount of time equal to the first duration, and so forth until the final duration is completed. Knowing this flight plan ahead of time allows us to predict when collisions (and thus changes in the configuration function) will occur. It is possible for us to obtain the flight plan beforehand by using 4D medical imaging techniques to record the motion of the patient's organs (and the tumor) over the desired time interval. Given that we have the flight plan for each of the polygons we know whenever the velocity of one of them changes. Between the times that these velocity change events occur, the velocity of all polygons and thus all vertices and edges is constant. During these *stable intervals* we can compute if and when a certificate fails and this only happens when a vertex/edge pair collides. As we are dealing with straight line edges only we know that any vertex can cross an edge at most once during the stable interval as the velocity of each vertex is constant. We can calculate these failure times for all certificates and throw out any that occur at times that are outside of the stable interval to obtain a complete list of collision event times for the interval. If we order these times from earliest to latest, we can process them in order of occurrence to maintain the validity of the configuration function as time passes.

Let n be the number of vertices in S and m be the number of vertices in T. Since the number of edges on the polygon is the same as the number of vertices, n and m are also the number of edges in S and T respectively. It only takes $O(1)$ constant time to update the configuration function upon certificate failure. This is because at each event we only need to add or remove a constant number of vertices from the intersection region whenever a collision event occurs. The intersection area R is represented as a circular doubly linked list that allows for constant time insertion and deletion of elements (vertices). To find the position in R at which we must insert/delete vertices, we maintain a lookup table of all polygon vertices of S that contains a flag that is true if the S vertex is inside of T and false otherwise. If it is true, then that vertex must be in R and the lookup table entry has a pointer to the position of the vertex in R. We have an equivalent table for T. We also need a lookup table for all S / T edge pairs that contains a flag that is true if the two edges intersect and false otherwise. If an S edge and a T edge intersect then there must be a vertex in R corresponding to this intersection vertex, and the lookup table contains a pointer to this vertex in R. We know the vertex and edge that are colliding from the certificate itself and can find these in the lookup tables and follow the pointers in the table entries to the position in R. Thus we can process each collision in $O(1)$ time with $O(n^2)$ space for the lookup tables.

The collision event times calculated above are only valid during the stable interval, and whenever a velocity of one of the polygons changes, the failure times must be recomputed as the change in velocity may change the direction of motion of any of the vertices in the system. Each vertex in S participates in m certificates, (see Figure 2) one for each edge in T, so S contributes a total of $O(nm)$ certificates. There are likewise $O(mn)$ certificates from T. This is an $O(nm)$ number of certificates that must be updated upon any change in velocity in the system. This quadratic time is not ideal for fast computation, and we observe a considerable reduction in performance of the algorithm as the number of vertices in the system increases. High numbers of vertices may be required if there are many targets or non-targets, especially if the cross sectional areas of these are complex in shape and require many vertices to approximate correctly. To approximate the smooth shape of many of the tissues and organs involved, high vertex counts are ideal. However, the certificate structure is such that during a stable interval the collision times of each vertex/edge pair is independent of all the others. This suggests a parallel programming solution.

In order to find the area of intersection between the two polygons we must triangulate R and then sum the areas of each triangle. Even with our KDS, which we can use to update the triangulation in a piecemeal fashion as we do the structure of R, the time this process takes is linear with respect to the number of vertices in the system. Between collision events the vertices of R and their ordering is stable. The area of a single triangle of the triangulation of R can be represented by a function that is dependent on time t. However, in order to find the total area we must compute the sum of these individual triangle functions:

$$AREA = \sum_{i=0}^{\#triangles} w_i \sqrt{A_i t^4 + B_i t^3 + C_i t^2 + D_i t + E_i} \tag{1}$$

Where A_i, B_i, C_i, D_i, and E_i are constants unique to each triangle. The constant w_i is a scaling weight used as a multiplier for each triangle area and is unique to each polygon in the system. This weight is used by clinicians to adjust the importance of each non-target in the system. Targets that are highly critical (such as OARs) will be given higher weights and their areas of overlap with the target contribute more to the weight function than lower weighted structures. This weighing process ensures that we take into account the clinical importance of all organs when optimizing the IMRT treatment plan.

The form of these triangle area functions is that of a square root of a quartic function. Due to this complex structure the sum cannot be minimized by setting the derivative to zero and solving for the roots of t. This prevents us from finding an exact solution to the minimization problem. In order to find the minimum, we must strategically sample the area at different times and find the minimum of the sampled time. This involves computing the triangulation of R and summing the areas of the triangles. Another opportunity for parallel programming presents itself in the both the triangulation of R and calculation of the individual triangle areas and their sum.

2.2 Comparison to Brute Force Methods

Before we discuss the parallel programming solution to the $O(nm)$ certificate update problem, it is instructive to see how our KDS, combined with parallel programming optimizations, improves upon a naive or brute force method of maintaining the intersection region between the target and non-target polygons.

Consider our two convex polygons S and T. S and T are continuously moving in time according to their respective flight plans, and as they move the combinatorial structure of R, the intersection region, may change at critical times. In a truly brute force method, we would have to recompute R at each time step of the motion, which would be determined by the minimum time step that can be captured by the medical imaging technique used to obtain the flight plan. Computing R involves finding the vertices of T that are inside S and vice versa, as well as the set of all intersections between S edges and T edges. At best, each of these steps is an $O(nlogn)$ procedure, ($O(nlogn)$ time for the point in polygon tests, and another $O(nlogn)$ for the edge intersection, if the edges are first presorted). To perform this recalculation at each time step would be computationally very expensive.

However, one might consider a simple optimization to this algorithm. Given the structure of R from the previous time step and that the velocities of S and T are known, we could in some cases consider only those vertices that are endpoints of the line segments that are intersecting. Assume that the boundaries of S and T intersect the minimum amount of times, which for convex polygons is two. As the polygons move according to their respective velocities, only a constant portion of vertices enter or leave R, namely the points that are near the intersection points. These intersection points are the result of some edge e_T in T intersecting with some edge e_S in S. The nearest vertices on S to the intersection point should be the endpoints of e_S, again because S is convex polygons. The same holds for T and e_T. Moreover, for each intersecting edge, one of the edge's endpoints must be inside the opposing polygon, and the other must be outside. Now suppose that S and T begin moving horizontally in opposite directions with constant velocities, as in Figure 3 (a). Eventually, one of the intersection points will have the same location as the endpoints, as the endpoint will collide with the boundary of its opposing polygon. We can use the velocity of the polygons to predict when this will occur, and we only need check the four endpoint vertices for any one intersection point for a total of eight vertices (with two intersection points).

However, this only works for a specific case and does not generalize to all situations. Convexity alone does not guarantee this property, and in the worst case all of the edges of S could be intersecting all of the edges of T, which requires we check all vertices. Consider if S and T began moving "towards" each other horizontally as shown above in Figure 3 (b). The circled vertex in the figure will collide with the T's boundary, and is not one of the intersecting edge endpoints. This collision which will create new intersection points, requiring us to consider even more vertices. There can be a total of $O(n+m)$ intersection vertices in the system, as in Figure 3 (c).

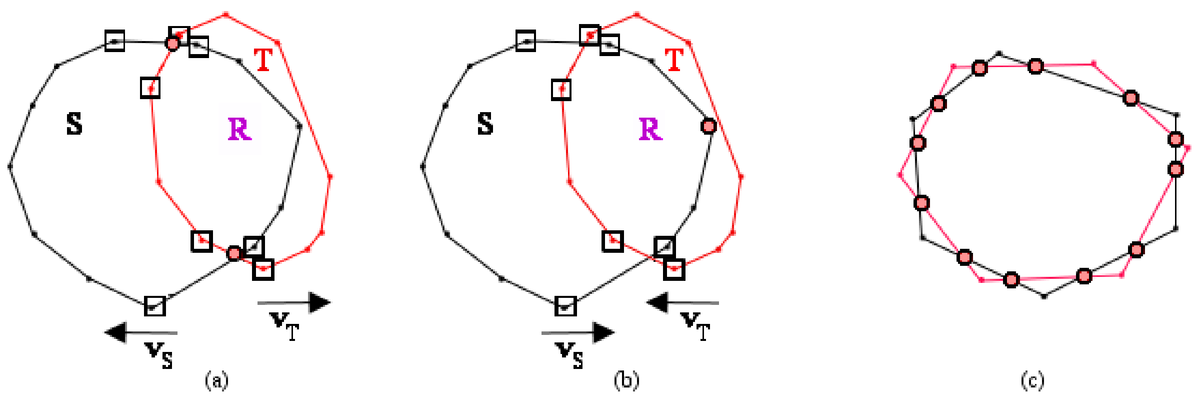


Fig. 3: (a) In this case, S and T are moving away from each other, and the vertices surrounded by boxes (around the circled intersection points) are those that may exit/enter the intersection region, suggesting that we only need to consider these vertices. (b) However, when S and T move towards each other, the circled S vertex will collide with the boundary of T before the vertices around the intersection points. (c) In this case, all edge pairs intersect. In the general case, we cannot guarantee any particular structure of R, so we must check all vertex/edge pairs.

Ultimately we must check all vertex/edge pairs between S and T for when each of these collisions occur, if they do at all, every time the velocity of one of the polygons changes. We must do this because we cannot guarantee that R is moving according to one of the previously mentioned situations throughout the entire motion of the polygons. Checking all vertex/polygon pairs at every velocity change is exactly our KDS algorithm as described above. There may be some way to detect intervals when these special cases hold and optimize the algorithm accordingly, but this is to our knowledge an open problem.

2.3 Exploiting Parallelism

We can use parallel programming techniques to improve the performance of our KDS based algorithm at two of the algorithm's steps: (1) computation of the certificate failure times and (2) calculation of the areas of the triangles making up the triangulation of the intersection region. We desire improved performance so that the IMRT treatment plans can be updated quickly. Ideally this update should happen in real time during the actual treatment as it is impossible to predict every way the patient can move beforehand. Assuming we have some way of detecting or predicting a sudden patient motion during treatment, we could recompute values for collision times quickly to maintain an optimal treatment when given the new motion.

The computation of each certificate failure time involves calculating the distance between a vertex and an edge, where the vertex is from one polygon of a certain class and the edge is from a different polygon of the opposite class. One of these polygons must be a target polygon - a polygon representing part or all of the target tumor. The other must be some non-target, which can be a OAR or other organ or tissue. If the minimal distance is zero, the vertex lies on the line supporting the line segment of the edge, and it can be said that the vertex collides with the edge if the vertex position is within the horizontal and vertical ranges of the edge line segment.

However, this computation is complicated by the fact that both the vertex and edge may be moving in time. To find the time of a collision, if a collision occurs, we must represent the location of the point and the endpoints of the edge as functions that vary with time, which can be derived by standard kinematic equations. The velocities and initial positions are known and constant during stable intervals. By substituting these equations for the vertex and edge endpoints into an equation that projects the vertex onto the line supporting the edge and calculating the distance between the projected point and the vertex, we can obtain the minimal distance as a function that varies with time. Setting this equation equal to zero and solving for time gives the time at which the minimal distance between the vertex and the edge is zero. For polygons S and T, let $p = (p_x, p_y)$ be some vertex on S colliding with T edge $e_T = (q_1, q_2)$ where $q_1 = (q_{1x}, q_{1y})$ and $q_2 = (q_{2x}, q_{2y})$ are endpoints of an edge of T. Let $V_S = (V_{Sx}, V_{Sy})$ and $V_T = (V_{Tx}, V_{Ty})$ be the velocities of S and T respectively during the current stable interval. We define for convenience $\Delta q_x = (q_{2x} - q_{1x})$ and $\Delta q_y = (q_{2y} - q_{1y})$. Let t_f be the time when p and e_T collide. We have solved for t_f in a concurrent work:

$$t_f = -\frac{-p_x(\Delta q_y) + p_y \Delta q_x - q_{2x} q_{1y} + q_{1x} q_{1y}}{-V_{Sx}\Delta q_y + V_{Sy}\Delta q_x + V_{Tx}\Delta q_y - V_{Ty}\Delta q_x} \quad (2)$$

This equation must be computed for all $O(nm)$ certificates in the system. However, the result for one certificate is

Table 1: GPU vs CPU Performance

Subdivision Level	#Vertices	# Certificates	GeForce® 460 GTX GPU	CPU	Radeon® HD 6630M GPU	CPU	GeForce® 9800 GTX+ GPU	CPU
0	20	160	0.475	0	2.755	0.05	1.925	0
1	42	640	0.45	0.025	2.85	.25	1.95	1.15
2	84	2560	0.55	0.125	3.05	0.875	1.2	2.725
3	168	10240	0.625	0.225	5.825	3.65	2.325	9.75
4	336	40960	0.625	1.075	8.625	13.975	4.275	39.4
5	672	163840	1.65	4.25	9.475	55.75	5.85	153.625
6	1344	655360	3.15	16.525	10.775	222.625	16.775	613.475
7	2688	2621440	9.70	63.850	31.1	893.5	50.3	2458.2
8	5376	10485760	35.05	253.325	177.34	3561.5	461.375	9835.50

independent of the others, so these computations can be done in parallel. We implemented a simple OpenCL kernel that allows for the parallel computation of the certificate failure times on a GPU graphics card. We used OpenCL so that our implementation can be used with graphics cards from different manufacturers. We pass memory buffers for the x and y initial coordinates for the vertex and both endpoints of the edge, as well as buffers indicating the polygon ids the vertex and the edge come from. We also pass buffers containing the velocities of the polygons, which can be accessed by the polygon ids. The kernel calculates the times when the distance function is zero and returns a buffer with the results. These results are then tested for failure times that are within the time range of the stable interval.

We can also apply parallel programming to the second problem: the area computation. This has not yet been implemented in our current work. Finding the minimum area involves sampling the area of the intersection region R at regular time intervals. The higher the sample rate, the more often we must compute the area and the slower the program will be. The fact that the polygons of the system are convex allows us to compute the area in $O(n)$ time, as the intersection of two convex polygon is itself a convex polygon and can thus be triangulated in linear (with respect to the number of vertices in R) time.

We are assured that the number of vertices defining the R is linear with respect to the number of vertices in S and T. To see why, recall that the intersection region R of S and T is composed of three different types of vertices: intersection vertices or polygon vertices from either S or T. The maximum number of intersection vertices occurs in R when all edges of S intersect with at least one edge of T. In this case the number of edges for S and T must be equal, and the number of intersection vertices is twice this number of edges (as seen in Figure 3 (c)). The maximum number of polygon vertices occurs when one of either S or T is completely inside the other, and is determined by the number of vertices in the enveloped polygon. Thus the number of vertices in R is linear with respect to the number of vertices in S or T.

In order to compute the area, we must triangulate the convex region of R by simply picking any vertex on R and forming triangles with the edges of R on which this vertex is not incident. We are assured that the number of vertices is $O(n)$, so we only need to form $O(n)$ triangles. If we have buffers maintaining x and y coordinates of the initial positions of the vertices in R as well as a buffer containing the horizontal and vertical components of the velocities of each vertex, we can construct a kernel that computes the areas of the triangles at some time t by forming triangles with the first vertex in the location buffer and pairs of the remaining vertices. These pairs must appear in order on the boundary of R, so the buffer must maintain the ordering of the vertices of R. Whenever a collision event occurs R is changed and must be recalculated, which takes $O(n)$ time. However, if the sampling rate is high enough this linear time recomputation may be worth the cost, as the number of sampling times may vastly outnumber the number of collision events.

The individual triangle areas are computed during the triangulation by using Heron's formula, which computes the area from the length of the three sides of the triangle. This formula allows us to easily define the triangle area in terms of the location of each vertex (with each vertex moving in time). In order to get the three side lengths, the distances between each pair of vertices on the triangle must be computed and then entered into the formula. The resulting triangle areas are returned in another buffer.

However, the goal is to find the *total* area, so we must also sum the areas of each triangle. When finding the sum we must examine each triangle area, and to improve the efficiency of the summation process a divide and conquer strategy can be used. This can be done by pairing up all of the triangle areas, computing the sum of the pairs in parallel, and repeating this on the resulting sums until a single sum is returned.

3. Results and Conclusions

We implemented the certificate failure calculation kernel using OpenCL and tested it in a prior C++ implementation of our KDS algorithm. We tested performance on two desktop computers and one laptop. The performance results (in milliseconds, averaged over 40 runs) are shown in Table 1 and compared to the KDS algorithm's performance without GPU optimization. One desktop had an Intel® i7 2600K 3.40 GHz CPU and was equipped with an NVIDIA GeForce® 460 GTX graphics card. The other desktop contained a 3.00 GHz Intel® Pentium D CPU and an NVIDIA GeForce® 9800 GTX+ card, and the laptop had an Intel® i7-2640M 2.8GHz CPU with an integrated AMD Radeon® HD 6630M graphics device. The KDS program was given a toy example consisting of 1 target polygon and 4 non-targets, each of which had no more than 5 vertices initially. We used a simplistic subdivision algorithm to quickly increase the vertex count in these polygons, and each subdivision level doubles the number of vertices from the previous level.

Our results indicated an overall improvement of roughly an order of magnitude in processing time. As Table 1 shows, the GPU parallel programming algorithm performs more poorly than the non-parallel version when the vertex count is low. The reason for this may be the added overhead of transferring the memory buffers from the CPU to the GPU and back, which for insignificant vertex sizes may cost more than the benefits of parallelization. Once the vertex count reaches the hundreds, the GPU algorithm performs up to an order of magnitude better than the CPU only algorithm.

Future work will include the application of this GPU programming technique to improve the speed of the triangulation of the intersection region and the computation of the total area of intersection. The results currently obtained were derived from a very simplistic kernel design, and it remains an open task to see if a different design could produce more efficient results. Overall we feel that the results in Table 1 show that applying parallel programming models with GPUs is a cost effective way to improve the efficiency of optimization algorithms for medical treatment planning.

4. Acknowledgments

This work has been partially sponsored by NSF award CNS-1035460.

References

[1] D. Rangaraj, G. Palaniswaamy, and L. Papiez, “Dmlc imrt delivery to targets moving in 2d in beam's eye view,” *Medical physics*, vol. 35, p. 3765, 2008.

[2] R. McMahon, R. Berbeco, S. Nishioka, M. Ishikawa, and L. Papiez, “A real-time dynamic-mlc control algorithm for delivering imrt to targets undergoing 2d rigid motion in the beam's eye view,” *Medical physics*, vol. 35, p. 3875, 2008.

[3] D. McQuaid and S. Webb, “Imrt delivery to a moving target by dynamic mlc tracking: delivery for targets moving in two dimensions in the beam's eye view,” *Physics in Medicine and Biology*, vol. 51, no. 19, p. 4819, 2006.

[4] X. Wu and J. Abraham, “The intensity level reduction in radiation therapy,” in *Proceedings of the 2005 ACM symposium on Applied computing*. ACM, 2005, pp. 242–246.

[5] L. Papiez, “Dmlc leaf-pair optimal control of imrt delivery for a moving rigid target,” *Medical Physics*, vol. 31, no. 10, pp. 2742–2754, 2004.

[6] M. Langer, V. Thai, and L. Papiez, “Improved leaf sequencing reduces segments or monitor units needed to deliver imrt using multileaf collimators,” *Medical Physics*, vol. 28, no. 12, pp. 2450–2458, 2001.

[7] B. Sun, D. Rangaraj, L. Papiez, S. Oddiraju, D. Yang, and H. H. Li, “Target tracking using dmlc for volumetric modulated arc therapy: A simulation study,” *Medical Physics*, vol. 37, no. 12, pp. 6116–6124, 2010.

[8] N. Boland, H. W. Hamacher, and F. Lenzen, “Minimizing beam-on time in cancer radiation treatment using multileaf collimators,” *Networks*, vol. 43, no. 4, pp. 226–240, 2004.

[9] X. Wu and J. Abraham, “The intensity level reduction in radiation therapy,” in *Proceedings of the 2005 ACM symposium on Applied computing*, ser. SAC '05. New York, NY, USA: ACM, 2005, pp. 242–246.

[10] Z. Wang, C. Willett, and F. Yin, “Reduction of organ motion by combined cardiac gating and respiratory gating,” *International Journal of Radiation Oncology* Biology* Physics*, vol. 68, no. 1, pp. 259–266, 2007.

[11] R. Wagman, E. Yorke, E. Ford, P. Giraud, G. Mageras, B. Minsky, and K. Rosenzweig, “Respiratory gating for liver tumors: use in dose escalation,” *International Journal of Radiation Oncology* Biology* Physics*, vol. 55, no. 3, pp. 659–668, 2003.

[12] P. Keall, G. Mageras, J. Balter, R. Emery, K. Forster, S. Jiang, J. Kapatoes, D. Low, M. Murphy, B. Murray, *et al.*, “The management of respiratory motion in radiation oncology report of aapm task group 76,” *Medical physics*, vol. 33, p. 3874, 2006.

[13] J. S. Kirtzic and O. Daescu, “A parallel algorithm development model for the GPU architecture,” *International Conference on Parallel and Distributed Processing Techniques and Applications (accepted for publication)*, 2012.

[14] R. Anderson, J. Kirtzic, and O. Daescu, “Applying parallel design techniques to template matching with GPUs,” in *High Performance Computing for Computational Science–VECPAR 2010: 9th International Conference, Berkeley, CA, USA, June 22-25, 2010, Revised, Selected Papers*, vol. 6449. Springer-Verlag New York Inc, 2011, p. 456.

[15] J. S. Kirtzic, D. Allen, O. Daescu, and T. Richardson, “Applying the parallel gpu model to radiation therapy treatment,” 2013.

[16] D. Allen and O. Daescu, “Radiation therapy simulation and optimization using kinetic polygon modeling,” in *Information Reuse and Integration (IRI), 2013 IEEE 14th International Conference on*, Aug 2013, pp. 239–246.

[17] L. Guibas, “Kinetic data structures,” in *Handbook of Data Structures and Applications*. CRC Press, 2004.

[18] L. J. Guibas, “Kinetic data structures – a state of the art report,” 1998.

[19] J. Basch, L. J. Guibas, and J. Hershberger, “Data structures for mobile data,” in *JOURNAL OF ALGORITHMS*, 1997, pp. 747–756.

[20] G. Alexandron, H. Kaplan, and M. Sharir, “Kinetic and dynamic data structures for convex hulls and upper envelopes,” *Computational Geometry*, vol. 36, no. 2, pp. 144 – 158, 2007.

[21] P. K. Agarwal, H. Kaplan, and M. Sharir, “Kinetic and dynamic data structures for closest pairs and all nearest neighbors,” Tech. Rep., 2008.

[22] J. Basch, J. Erickson, L. J. Guibas, J. Hershberger, and L. Zhang, “Kinetic collision detection between two simple polygons,” 1999, pp. 102–111.

[23] D. Kirkpatrick, J. Snoeyink, and B. Speckmann, “Kinetic collision detection for simple polygons,” in *Proceedings of the sixteenth annual symposium on Computational geometry*, ser. SCG '00. New York, NY, USA: ACM, 2000, pp. 322–330.

[24] J. Hershberger, “Kinetic collision detection with fast flight plan changes,” *Information Processing Letters*, vol. 92, no. 6, pp. 287 – 291, 2004.

Enhanced Automated Data Dependency Analysis for Functionally Correct Parallel Code

Prasad Pawar[1], **Pramit Mehta**[1], **Naveen Boggarapu**[1], **and Léo Grange**[1]
[1]Center for Research of Engineering Sciences and Technology (CREST), KPIT Technologies Ltd., Pune, India

Abstract—*There is a growing interest in the migration of legacy sequential applications to multicore hardware while ensuring functional correctness powered by automatic parallelization tools. OpenMP eases the loop parallelization process, but the functional correctness of parallelized code is not ensured. We present a methodology to automatically analyze and prepare OpenMP constructs for automatic parallelization, guaranteeing functional correctness while benefiting from multicore hardware capabilities. We also present a framework for procedural analysis, and emphasize the implementation aspects of this methodology. Additionally, we cover some of the imperative enhancements to existing dependency analysis tests, like handling of unknown loop bounds. This method was used to parallelize an Advance Driver Assistance System (ADAS) module for Lane Departure Warning System (LDWS), which resulted in a 300% performance increase while maintaining functional equivalence on an ARM™ based SOC.*

Keywords: Automatic Loop Parallelization, OpenMP, Range Test, Conditional Parallelization, Loop Normalization, Loop Dependency Analysis

1. Introduction

Modern computer architecture ethos favors a multicore desgin paradigm for both servers and emedded soultions. Multicore platforms add a significant overhead to program development cycles due to added complexities, and strict verification of compatibility issues. The standard procedure to increase the throughput of a sequential program, is to optimize or parallelize the application. Parallelizability of an application is based on the inter-dependency of the tasks within that application. Detailed code analysis is required to decide the dependency and parallelizability of the tasks in code. Sometimes, the time overhead induced by multithreading using the OpenMP API may nullify the parallelization benefits. Thus, availability of multiple cores does not always guarantee a performance improvement. The majority of current as well as legacy applications, especially in safety critical domains like automotive, aerospace, etc. are sequential. When these applications are parallelized, their functional correctness for all possible set of inputs is a major concern. The application performance is essential but is of lower priority compared to functional correctness. Migration of these applications to parallel or multicore hardware requires thorough verification as parallelization may induce complex errors. For e.g. synchronization errors, deadlocks, resource contention, etc. using dependency analysis which plays a major role in parallelization process [13]. Manual analysis of code for parallelizability is a complex and time consuming [15] activity depending upon number of lines, inter-dependent functions, and file count. In these efforts, human error may still creep in, especially in large code sets as it is humanly impossible to keep track of multiple synchronizations across files. This paper focuses on the mechanism developed to automatically convert sequential code into parallelized code using OpenMP constructs. This approach comprises of:

1) automatically analyzing a loop block for dependencies, and
2) checking functional correctness.

The programmer can then fine-tune the parallelized code for further optimal behavior. In the upcoming section we will discuss in detail the methodologies used in the loop parallelization module of YUCCA tool, which was earlier known as S2P [3].

2. Literature survey

Automatic parallelization includes analysis of local and global variables, array indices, inter-procedural dependencies, alias analysis, control flow analysis, etc. Runtime preprocessing to efficiently reduce the cost of communication along with non-linear array sub-script analysis for irregular parallelizable code is handled by automatic parallelization techniques for irregular scientific applications [11]. Compiler guide lines to programmer and semi-automatic loop parallelization based on iteration analysis, variable induction analysis is handled by the technique of compiler guided refactoring [12]. There has been work done in the context of adaptive work load distribution across OpenMP loops using machine learning [8]. Machine learning based thread versioning results in optimum work-share distribution over the threads. Run-time comparison comes up with optimum OpenMP combinations in regards of chunk and number of threads. Fine coarse grain parallelization techniques such as tiling of a given data set, and performing distinct tile operation on different processing cores are also being taken care of by automatic parallelization tools like PLUTO [7]. A polyhedral model is used in PLUTO to analyze the irregular dependencies in nested loops to find opportunities for parallelization. There are tools for automatic parallelization like Intel Compilers [1], Par4All [2], Cetus [4] etc. described in the automatic parallelization YUCCA tool [3], which do not cover implementation in detail.

3. Prerequisites and limitations for loop parallelization using OpenMP

- Loops to be detected by static analysis for parallelization needs to be normalized before processing.
- Loop indices should always be integers.
- OpenMP does not check for data dependencies, data conflicts, race conditions, and deadlocks.
- OpenMP does not allow static extent scope expanse to multiple code files.
- The functional correctness of a parallelized block is not verified by OpenMP.
- Jump statements should not directly or indirectly break the loop considered for parallelization[1].
- The types of variables under consideration are primitives, arrays, and pointers or array references.
- Loop stride and loop conditions are loop invariant.

4. Dependency analysis

4.1 Simple variable analysis

Variable analysis is an essential part of static analysis to find out the parallelizability of a loop. This section covers all possible types of variables and scenarios where variables are used. The typical flow for the proposed methodology is as shown in figure 1.

The following sections cover some methodologies for variable analysis.

4.1.1 Prediction of variable values

The dependency analysis gets changed based on the scope of the variable (i.e. whether the variable is local or global with respect to the loop). The values of the variables are identified based on their location.

In example 1, on line 4, variable a is assigned a constant, 10, and in line 5 b is assigned 23. Within the for loop on line 8, variable c is an expression in the form of the abstract syntax tree (AST) as shown in figure 2.

The same AST is modified as shown in figure 3

In the same way all the variable are reduced and possible values of that variables are stored in a list.

4.1.2 Handling of branches

If branching is detected, all the possible paths are noted. A list comprising of all the possible values for each variable in each branch is then generated. Branches that cannot be reached based on variable values are ignored for dependency analysis. For example,

```
if (b==12){
```

[1]Note that jump statements can be used to break nested child blocks, but not the parent block which is considered for parallelization.

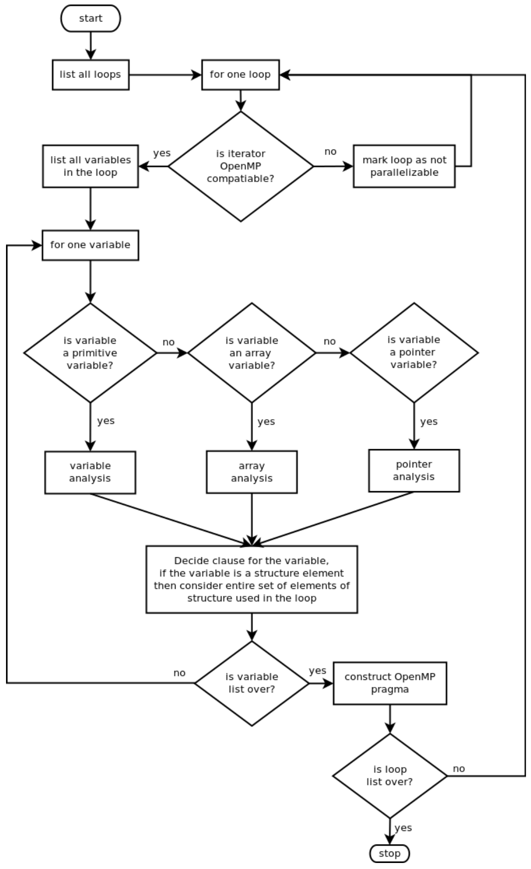

start
list all loops
for one loop
is iterator OpenMP compatiable?
yes
no
list all variables in the loop
mark loop as not parallelizable
for one variable
is variable a primitive variable?
no
is variable an array variable?
no
is variable a pointer variable?
yes
yes
yes
variable analysis
array analysis
pointer analysis
Decide clause for the variable, if the variable is a structure element then consider entire set of elements of structure used in the loop
no
is variable list over?
yes
construct OpenMP pragma
is loop list over?
no
yes
stop

Fig. 1: Flowchart of dependency analysis

Example 1 Sample source code for variable value prediction

```
 1 main () {
 2     int a, b, c, i, j;
 3     ...
 4     a = 10;
 5     b = 23;
 6     for ( i = 0 ; i < 10; i++){
 7         ...
 8         c = a + b;
 9         b = c + 1;
10         a = a + b + c + 43;
11     }
12 }
```

```
   c
   +
  / \
 a   b
```

Fig. 2: AST for expression c = a + b

```
      c
      +
     / \
   10   23
```

Fig. 3: Modified AST based on figure 2

```
    ...
};
```

If it is known that 12 is not a possible value of b, then the *then* path of *if* statement is ignored.

4.1.3 Read write test

The Read Write Test identifies if there is any loop carried dependency. It verifies whether a variable written in a loop is used in the next iteration. We do this by determining if there is a read access on the variable before a write access. In case such dependencies exist, the loop is marked non-parallelizable.

In example 2, $x2$ is read on line 7 before it is updated. As the value of $x2$ written in current iteration is used in the next iteration, this loop is considered non-parallelizable.

Example 2 Sample code for read write test

```
 1 main (){
 2     int i,k,x1,x2;
 3     int a[20];
 4     ...
 5     for (i = 0; i < k; i++){
 6         ...
 7         x1 = x2 + 4;
 8         x2 = x1 + k + i;
 9         ...
10     }
11     ...
12 }
```

In example 3, $l2$ is read before it is written. Hence, we mark this loop non-parallelizable as well. However, this scenario can be handled using the *Reduction* construct found in OpenMP.

Another case where the *read write test* requires some assistance from other tests is if there is a conditional control flow in the loop, and the variables are read or modified as a part of the conditional control flow.

Arrays and pointers in a loop are analysed using other tests which are explained in the sections below.

4.1.4 XNOR test

As discussed, the *read write test* considers the position of a variable, but sometimes it is also needs to consider the control flow of the loop where the variable is being used. The *XNOR test* is used for variables which are accessed first

Example 3 Sample code for reduction

```
main (){
    int i,k,l1, l2;
    int a[20];
    ...
    for(i = 0; i < k; i++){
       ...
       l2 += i;
       ...
   }
   ...
}
```

in a conditional block. If the variable is accessed before the conditional block, the *read write test* is considered for each branch.

The following scenarios illustrate the cases where the *read write test* needs the assistance of the XNOR Test.

A variable first accessed in a conditional block is,

(a) only accessed inside the conditional block,
(b) only written in all the branches,
(c) only read in all the branches, or,
(d) read in at least one branch as well as written in another branch

In scenario *a*, if variable x is accessed only in the conditional block, then the *read write test* is applied on that variable with respect to each branch separately. If there is at least one branch that the *read write test* marks as non-parallelizable, then the entire loop is marked non-parallelizable.

In scenario *b*, if the variable x is first accessed in the conditional block and all branches have write access to that variable, then that loop is marked as parallelizable with respect to variable x.

In scenario *c*, if the variable x is first accessed in the conditional block and all branches have read access from that variable, then the loop is marked non-parallelizable with respect to that variable. However, if there is a write after the branch statements, the loop is marked parallelizable.

Static analysis of scenario *d* cannot determine which branch of the control flow graph will occur during any iteration, hence we mark this case as non-parallelizable as well.

In the example 4, variable b is only accessed in the conditional block. In one branch it is written first and read later, and in the other branch, it is read first and written later. This will cause the *read write test* to fail in one branch, so this loop is marked non-parallelizable.

4.2 Variable array analysis

Variable array analysis is different from typical variable analysis as we need to look through a range of memory

Example 4 Example 4. Sample code for XNOR test case 1

```
for (i = 0; i < 20; i++){
    if (k == i){
        a = b + c;
        b = a + k;
    } else {
        b = k;
        r = b;
    }
}
```

locations, and determine memory contention of write as well as read across this range. Hence, in order to parallelize loops containing array references, we use the following tests:

4.2.1 Loop normalization

Loop normalization is carried out to ensure that correct conditions are achieved by the means of analysis. A loop is assumed to be normal when its stride is 1. For example, a regular loop in which the loop index i iterates from lower bound l to stride s in steps of Δ, the normalized iteration is written as: $(i - l + s)/\Delta$. The loop body condition will also change accordingly.

Further, to ease the analysis via the *range test*[5] for array references, several pre-processing steps need to be followed:

1) Multi-dimensional arrays need to be converted to single dimensional array. For e.g. $a[640][480]$, can be expressed as $a[640 * 480]$, and all the references to $a[i][j]$ can be expressed as $a[480 * i + j]$.
2) Array references in a loop should be a function of the iterators of the current loop and the inner loops, and the loop invariants. This can be achieved by back tracing the abstract syntax tree (AST) of array reference.

If the array references are the return values of a defined function, then the AST of the function definition is back-traced to its dependency with respect to the iterators. This converts the array reference into a function of iterators and invariants.

4.2.2 GCD test

The GCD test [14] is used to identify loop carried dependencies. It works on the basis of solutions to Diophantine equations. In a loop iteration where we have two or more array references, and where at least one reference is a writing operation, the GCD test verifies whether the integral solutions of the equations are possible.

In example 5, there exists a dependency *if and only if* we can find i_1 and i_2 such that $2 * i_1 + K1 = 2 * i_2 + K2$. The GCD test ignores the bounds of the loop in a simple case like this example. Given a bound, we can parameterize the equations to obtain the range in which there is a possibility of finding integral solutions.

Example 5 GCD Test example

```
for (i = 0 ; i < 100 ; i ++){
   a[2*i + K1] = ...;
   ... = a[2*i + K2];
}
```

4.2.3 Range test

GCD based analysis has the following shortcomings while trying to analyze and identify a loop carried dependency:

- both the upper and the lower bounds of the loop need to be known beforehand, and
- arrays inside the loop need to be indexed linearly.

In order to overcome the above shortcomings, the *range test* was proposed by Blume et al [5]. This test works for nonlinear as well as unknown bound conditions.

We have extended the work of [5] to include singular array references inside a loop. By doing so, we can generate an OpenMP *if* clause which can determine, at runtime, whether a loop can be parallelized.

In example 6, $f(i, j, \text{loop invariants})$ is a mathematical function of iterator variables i and j, and the loop invariants. We will go through the process of generating the conditions under which the outer loop, with iterator i, is parallelizable. Although this process is explained for an array access inside an inner loop, it can be generalized for any number of inner loops [5] as well.

Example 6 Sample Code with loop_invariants

```
for(i= LB_i; i < UB_i; i++){
  for(j = LB_j; j < UB_j; j++){
     a[f(i,j,loop invariants)] = ...;
  }
}
```

Generating conditions: Firstly we need to determine whether the funciton $f(...)$ is either strictly increasing or decreasing in the range of i. For that, let

$$P(i, j, \text{loop\_invariants}) = f(i + 1, j, \text{loop\_invariants}) - f(i, j, \text{loop\_invariants}) \quad (1)$$

In order to identify the conditions, we use the symbolic range propagation methods as described in [6]. Since the bounds are unknown, we shall get an interval $[a, b]$ where a is derived from values of i,j where the curve $P(i, j, \text{loop\_invariants})$ is minimum, and b where $P(i, j, \text{loop\_invariants})$ is maximum. As i and j are increasing in nature, the condition thus derived will be $a > 0$.

Once it has been determined that the function has a strict positive/negative slope, we need to assess whether for any loop iteration i, the range of memory locations accessed are different from that accessed by the loop iteration $i+1$. Using methods similar to the *range test*, we find the lowest memory location accessed by the index function $f(...)$ during iteration $i+1$ by $f_{min}(i+1, j, \text{loop\_invariants})$, and the highest memory location accessed by the index function $f(...)$ during iteration i. This method applies irrespective of whether the index function is increasing or decreasing.

If the difference between these two memory locations is greater than zero, we can identify that the concerned loop is parallelizable with respect to this array variable.

Example 7 Code Snippet 1:

```
#pragma parallel for shared(a)\
if( (N > 0) && (N >  M2) )
for(i = 0 ; i < M1; i++){
    for(j= 0 ; j < M2 ; j++){
        a[ N * i + j] = ...;
    }
}
```

Explanation using example 7: In this example, we have two conditions in an OpenMP construct, which are resolved only during runtime. The following can be concluded by examining example 7:

$$f(i,j) = N * I + j$$
$$Lowerbound(i) = 0, Upperbound(i) = M1$$
$$Lowerbound(j) = 0, Upperbound(j) = M2$$
$$f(i+1,j) - f(i,j) = N*(i+1)+j - N*(i)+j = N$$

$f(i,j)$ is strictly increasing given N>0; establishes condition 1. This establishes condition 1.

$f_{min}(i+1,j)$ is lowest memory access in loop iteration i+1. Thus we have,

$$\begin{aligned} f_{min}(i+1,j) &= N*(i+1) + \text{lower bound of } j \\ &= N*(i+1)+0 \\ &= N*i+N \end{aligned} \tag{2}$$

Similarly we have

$$\begin{aligned} f_{max}(i,j) &= N*(i) + \text{upper bound of } j \\ &= N*i+M2 \end{aligned} \tag{3}$$

Difference between equations 2 and 3 needs to be greater than zero to determine memory independence. This establishes condition 2 as shown in equation 4.

$$\begin{aligned} N*i+N-N*iM2 &> 0 \\ N-M2 &> 0 \\ N &> M2 \end{aligned} \tag{4}$$

Similarly conditions for N < 0 where the function is decreasing can be determined.

Example 8 Code Snippet 2:

```
#pragma parallel for shared(A)\
     if ( 2*L1 +1-a > 0 && \
          *L1 +1 + L2-a-M2 > 0 )
for( i =L1 ; i < M1 ; i++){
  for(j=L2 ; j< M2 ; j++){
      a[i*i - a*i5 + j] = ...;
  }
}
```

Explanation using example 8: This example uses a more complicated array accessing function which uses loop invariants. The following can be concluded by examining example 8:

$$f(i,j) = i^2 - a*i+j \tag{5}$$

where i and j are loop iterators and a is a loop invariant which is a constant.

$$f(i+1,j) = i^2 + 2*i+1-a*i-a+j \tag{6}$$

Taking a difference of equation 6 and 5 we have

$$f(i+1,j) - f(i,j) = 2*i+1-a \tag{7}$$

For $f(...)$ to be an increasing function, the condition thus becomes

$$2*i+1a > 0 \tag{8}$$

thus

$$i > (a-1)/2 \tag{9}$$

$$\begin{aligned} f_{min}(i+1,j) &= i^2 + 2*i+1-a*(i+1)+L2 \\ &= i^2+2*i+1-a*i-a+L2 \end{aligned} \tag{10}$$

$$f_{max}(i,j) = i^2 - a*i + M2 \tag{11}$$

Difference between equations 10 and 11.

$$2*i+1+L2-a-M2 > 0 \tag{12}$$

thus for all values of i,

$$\begin{aligned} M2 &< 2*i+1-a+L2 \\ M2 &< 2*L1-a+1+L2 \end{aligned} \tag{13}$$

4.2.4 Arrays as loop private variables

The dependency analysis is performed by considering an array variable as *private* or *shared*. If the modified values of an array are used after the completion of the loop block, and no dependency exists between any of the array subscripts, then the array is used as a shared variable using the *shared* OpenMP construct. If array values are not required after loop block, then the array can be used as a private variable. So the parallelization of a loop is still possible if there is a dependency existing within an array, but the array is not required after the loop block execution by the privatization of the array [13]. By assigning a separate copy of the array to each thread.

So, the parallelization of a loop is still possible if there is a dependency existing within an array, but it is subject to the array not being required after the loop block execution. The parallelization is acheived through privatization of the array [13][2].

4.3 Variable pointer analysis

Pointers of variables need to be analyzed separately as the methodology to understand the memory contention is different. Our methodology supports a single level of indirection while considering pointer analysis. Memory maps of each OpenMP thread are considered to determine the correct OpenMP constructs. There are three major possibilities while considering pointer access in a loop body which may result in a loop carried dependency:

1) *The value stored at the address to which a pointer points is altered.* Alias analysis [10][9] is done to capture all the variables which are affected by the pointer and are used in the loop. Further variable analysis [16] is done over these variables to identify loop carried dependencies, if any exist. As a pointer cannot be kept private with the memory still remaining the same, the write and read access of that address is kept atomic.
2) *The pointer is not modified, but an arithmetic expression of the pointer is used to modify values different addresses.* Pointers of this form can be converted into array form as shown below:

$$p = \&A;$$

$$*(p + i) \ = A[i];$$

After this conversion, array analysis, as mentioned in section 4.2, is used to check for parallelizability.
3) *The pointer is made to point at a different address.* The first check is to analyze how many times the pointer has been changed. Separate alias analysis has to be done for each write operation on that pointer. Variable analysis is run over all the aliased variables for identification of loop carried dependencies. Precautions taken for case (1) are also to be considered here.

[2]Array privatization is implemented by assigning a separate copy of the array to each thread.

4.4 OpenMP constructs usage

In this section we look at the OpenMP clauses used for automatic parallelization.

4.4.1 Identification of OpenMP clause of a variable

The previous sections covered the analysis of different types of variables. During these analyses, we identified the appropriate OpenMP clause for each variable. These clauses are subject to the parallelizability of the loop.

Based on its functionality and location, a single variable may be assigned multiple OpenMP clauses. In case a conflict arises due to multiple clauses, the highest priority clause would be used. Clause priority is based on table 1.

Table 1: Conflict Resolution Table.

Clause 1	Clause 2	Result Clause
Private	First-private	First-private
Private	Last-private	Last-private
Private	Shared	First-private
First-private	Last-private	Last-private & First-private
First-private	Shared	First-private
Last-private	Shared	Last-private & First-private

4.4.2 Structure and union individual element analysis – OpenMP construct

In case of structures and unions, each element is analysed separately, and for every element, an appropriate clause is used. Once the clauses are finalized for each element, a single clause is decided for the structure based on the priorities found in table 1.

4.4.3 Static and dynamic clause

Based on the analysis of the code, a corresponding scheduling clause of OpenMP is used. Here, the focus of analysis is to identify the conditional code existing within a parallelized loop. Based on the analysis of conditions, we determine whether the entire code gets executed. If the code present within a loop is definite[3], then we use a static scheduling clause along with OpenMP's standard clause in order to parallelize the loop. If there are dependencies on conditional statements like if, while, switch etc. or function calls within a loop block, then we use OpenMP's dynamic scheduling policy.

[3]That is, not having any dependency during execution on any type of conditional statements like if, while, switch etc. or function calls within the loop block

4.4.4 If clause

During code analysis , if we come across a condition which needs to be satisfied before the parallelization of code (as mentioned in range test , section 4.2.3), the condition is used inside OpenMP's *if* clause to increase the probability of parallelization without losing the correctness of the application.

4.5 Experiments and results

We implemented this methodology on an lane departure warning system. The source code used is ≈2000 lines of code, and was chosen as a representation of computer vision algorithms used currently in automotive systems. As these algorithms pertain to the safety of a driver, it is extremely important to maintain functional correctness while improving its performance through parallelization.

The output obtained was cross-compiled for an i.MX 6 Sabrelite board running embedded Linux kernel 3.0.15. The output obtained was functionally the same for all inputs as compared to the output of sequential code while giving algorithm performance benefit of ≈300%.

5. Conclusion

Multiple automatic parallelization techniques exist, but each one has its own limitations, thus limiting their use to specific applications. We present the YUCCA tool, which addresses lexical analysis in order to find opportunities for parallelization. This paper focuses on:

1) complex loop analysis for optimal utilization of multi-core technology, and
2) ensuring functional correctness, in order to address the primary concerns of safety critical applications.

Using the XNOR test (section 4.1.4), the enhanced range test (section 4.2.3), and the pointer analysis methodologies presented in section 4.3, we analyze variables within conditional branches. Additionally, we handle unknown loop bounds as well as variable pointers within a loop. This paper also covers the usage of OpenMP constructs in order to parallelize loops, maximize resource utilization, and achieve improved throughput.

5.1 Moving forward

We are in the process of optimizing our variable pointer analysis module in order to make it more robust. Additionally, our ability to address function calls within loops is still in its nascency. Development in this field would benefit if we could have a standardized test set as well as corresponding benchmarks (both in terms of performance, as well as in terms of source code).

References

[1] Automatic Parallelization with Intel Compilers. [Online]. Available: https://software.intel.com/en-us/articles/automatic-parallelization-with-intel-compilers

[2] The PIPS Workbench Project. [Online]. Available: http://www.cri.ensmp.fr/pips/

[3] A. Athavale, P. Ranadive, M. Babu, P. Pawar, S. Sah, V. Vaidya, and C. Rajguru, "Automatic sequential to parallel code conversion: The s 2 p tool and performance analysis," *GSTF Journal on Computing(JoC)*, vol. 1, no. 4, 2012.

[4] U. K. Banerjee, *Dependence analysis for supercomputing*. Kluwer Academic Publishers, 1988.

[5] W. Blume and R. Eigenmann, "The range test: a dependence test for symbolic, non-linear expressions," in *Supercomputing Conference*, 1994, pp. 528–537.

[6] W. Blume and R. Eigenmann, "Symbolic range propagation," in *International Parallel and Distributed Processing Symposium/International Parallel Processing Symposium*, 1995, pp. 357–363.

[7] U. Bondhugula, M. Baskaran, S. Krishnamoorthy, J. Ramanujam, A. Rountev, and P. Sadayappan, "Automatic transformations for communication-minimized parallelization and locality optimization in the polyhedral model," in *Compiler Construction*. Springer, 2008, pp. 132–146.

[8] X. Chen and S. Long, "Adaptive Multi-versioning for OpenMP Parallelization via Machine Learning," in *International Conference on Parallel and Distributed Systems*, 2009, pp. 907–912.

[9] K. D. Cooper and K. Kennedy, "Fast interprocedual alias analysis," in *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '89. New York, NY, USA: ACM, 1989, pp. 49–59. [Online]. Available: http://doi.acm.org/10.1145/75277.75282

[10] A. Deutsch, "Interprocedural May-Alias Analysis for Pointers: Beyond k-limiting," in *SIGPLAN Conference on Programming Language Design and Implementation*, 1994, pp. 230–241.

[11] M. Guo, "Automatic Parallelization and Optimization for Irregular Scientific Applications," in *International Parallel and Distributed Processing Symposium/International Parallel Processing Symposium*, 2004.

[12] P. Larsen, R. Ladelsky, J. Lidman, S. McKee, S. Karlsson, and A. Zaks, "Parallelizing more loops with compiler guided refactoring," in *Parallel Processing (ICPP), 2012 41st International Conference on*, Sept 2012, pp. 410–419.

[13] D. E. Maydan, S. P. Amarasinghe, and M. S. Lam, "Array-data flow analysis and its use in array privatization," in *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 1993, pp. 2–15.

[14] C. D. Offner, "Modern dependence testing," DEC Internal Report, Tech. Rep., 1996.

[15] P. Randive and V. G. Vaidya, "Parllelization tools," in *Second International Conference on MetaComputing*, 2011.

[16] A. Sane, P. Ranadive, and S. Sah, "Data dependency analysis using data-write detection techniques," in *International Conference on Software Technology and Engineering*, 2010.

Acknowledgements

We would like to thank Subodh Pachghare, Shiva Ghose, Priti Ranadive, Aditi Sahasrabudhe, Smitha K P, Chaitanya Rajguru, Dr. Vinay Vaidya from CREST at KPIT Technologies for their contribution towards review and valuable inputs.

A Brief Survey on Parallel Programming Applications for Image Processing

Leticia Flores-Pulido, Esther Ortega-Mejía, Eduardo Vega-Alvarado, Juan Manuel-Álvarez
Universidad Autónoma de Tlaxcala
Facultad de Ciencias Básicas, Ingeniería y Tecnología
Calzada Apizaquito s/n. C.P. 90300 Apizaco, Tlaxcala, Mexico
aicitel.flores@gmail.com, eortega_m@hotmail.com, evega@ipn.mx, jumaalmx@gmail.com

***Abstract*—In this work a series of applications for parallel programming is exposed, all of them related to image processing. A comparison of related works is included, showing the features and scope of these applications. The third section of this paper presents a proposal based on hypercubes, as a mechanism for establishing arrays of processors for parallel computing in addition to the approaches exposed in the included works, with the objective of exposing another kind of solution for image processing in huge data collections.**

I. Introduction

Parallel computing has different approaches and techniques, and uses multiple resources and platforms to solve complex problems, in those cases when computer power is required. All this diversity is related with the type of problem to be solved and the quantity and classification of the data involved. Once that the parallel section of an algorithm is identified, usually there are several alternatives for implementing a solution that takes advantage of this parallelism.

A. Automatic Road Extraction From Satellite Images Using Extended Kalman Filtering And Efficient Particle Filtering

There is an extended necessity for using geospatial information, in order to make precise visualizations of roads and terrain, in areas where projects are to be developed, making this a complex problem to be solved. The analysis of this information is made by processing satellite images, and the platform required has a considerably complexity. There are different methods to solve this problem. Automatic tracking starts with an automatic seeding of a road segment that indicates the road centreline, and then the computer learns relevant information of the road, such as initial direction, step size, width, and so on. A Road Extraction Method is shown in the flowchart of the Figure 1. It begins with a given satellite image, with two filters applied after the automatic seeding stage. The reference profile and seed point are extracted automatically by an automatic seeding technique. The edge of the road is detected using Canny edge detection. The road network is tracked by a filter until it reaches a road junction or an obstacle, then it is passed to another filter to initialize the seed point of the road branch.

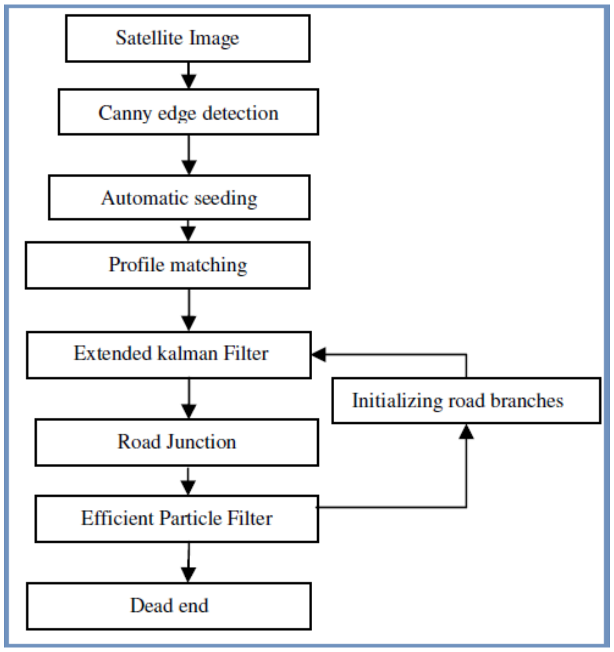


Fig. 1. Workflow of Road Extraction Method.

Two algorithms are addressed for automatic road extraction: Extended Kalman Filtering (EKF) and Particle Filtering(PF). EKF serves to identify a hidden vari- able that represents the degree of freedom in the state vector of the dynamic system, considering the distance along the road as time variable. PF is a model- based estimation simulation used for Bayesian models. Since it is necessary to know the current road segment for the propagation of dynamic models, a pointer is assigned to the destination road segment. All information of the current seg- ment of the road, such as directions and neighbours, is indexed in the database by means of the pointer. The main approximation of the filter is the Gaussian as- sumption about the conditional target state distribution given a mode sequence and observations. The efficient particle filter with 80 particles yields satisfactory simulation results [1].

Extended Kalman Filter has many applications in computing and is generally used to minimize error in Non lineal Dynamical Systems. As a future work a neural network processed by parallel computing is proposed in order to improve the performance.

B. Parallel and SIMD Optimization of Image Feature Extraction

There are two main methods for image retrieval in large data bases: Description- Based Image Indexing and Content-Based Image Retrieval (CBIR). The first method uses a search on metadata describing the images, such as tags, key- words, subject headings or even text; however, it is difficult to capture the ap- propriate words to describe an image, and in some applications like surveillance cameras there is no description. In the second method, the search is conducted by an analysis of image contents such as colors, shapes or textures, or any other information derived from the image itself; so, image feature extraction is the base for CBIR. Image feature extraction has many other applications, especially when it is done in real time, such as image filtering, automatic image processing, automatic monitoring and capturing, and automatic computer vision [2].

In the paper, authors introduce some optimized methods of image feature extraction, including both thread level parallelism (TLP) and SIMD (single in- struction multiple data) instruction level parallelism (ILP). Images are consid- ered as a main theme that is usually what we want to detect. The scenes above and below, and on the left and right sides are usually different. So, the frame is divided in five sections, each of them 1/2 x 1/2 the size of the origin frame, and with a central section that overlaps the other ones; Figure 2 shows the division scheme. Four optimization methods are proposed:

1) Adaptive Coarse-grained TLP where each working CPU deals with one images feature extraction.
2) SIMD optimization on image smoothing and gradient generating eliminates the high-frequency noise of the image and prepares it for gradient generating.
3) Eliminate conditional branch and SIMD optimization on Non-maxima suppression.
4) SIMD optimization on image color feature extraction. Conditional branches are substituted, this time using SIMD logical shift-operation instructions.

Results show a speed-up ratio for the TLP optimization of 8, nearly linear speed-up; this is because every image is processed in one thread, and there is no dependency between threads. The SIMD speed-up is not remarkable for relatively small images, but as the size gets larger, SIMD optimization shows a good improvement. This is a good example of an application using more than one parallelism technique simultaneously. The first part of the proposal involves techniques related with the algorithm implementation and scheduling, while the second stage is based on SIMD characteristics from the hardware platform, that is, the instruction set of the processor.

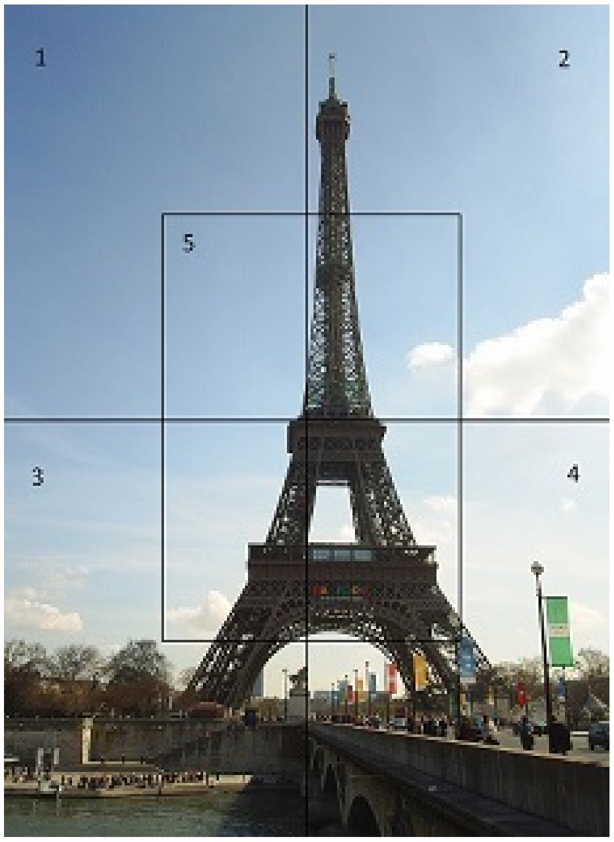


Fig. 2. Division of an image frame

C. GPU Accelerated Automated Feature Extraction From Satellite Images

This work shows the implementation of computational power by a parallel architecture based on a GPU (graphic processing unit), to match computing requirements for processing huge data quantum like satellite images. This kind of analysis for automated feature extraction is very time consuming by traditional sequential processors, even with multi-cores. This work focuses on an algorithm for automated feature extraction, from panchromatic satellite images used for detection of urban areas and water bodies. The strategy for this solution starts by copying an image from the CPU memory to GPU memory, computing all parallel calculations on GPU and then returning the results to CPU memory.

The solution can be obtained by two approaches. The first approach consists on dividing the image in rows of pixels, and then an entire row is assigned to one processing thread, so the number of threads equals the number of rows. This solution is faster than the one for a multi-core CPU. The second approach process in parallel every pixel, assigning one thread to each pixel. The configuration for threads and blocks is selected to minimize execution time, making this solution faster than the first approach.

Synchronization was needed at the end of the whole parallel computing [3]. The automated feature extraction algorithm is formed by four techniques applied to an image for identifying regions of interest. The main contribution in this work is the translation of the algorithm from CPU execution to parallel kernel execution by multiple threads inside a GPU, optimizing execution time. The best result was obtained using parallel pixel approach, with a maximum speed up of 1900. Further extensions of this work include feature extraction from objects like buildings and roads with a higher accuracy.

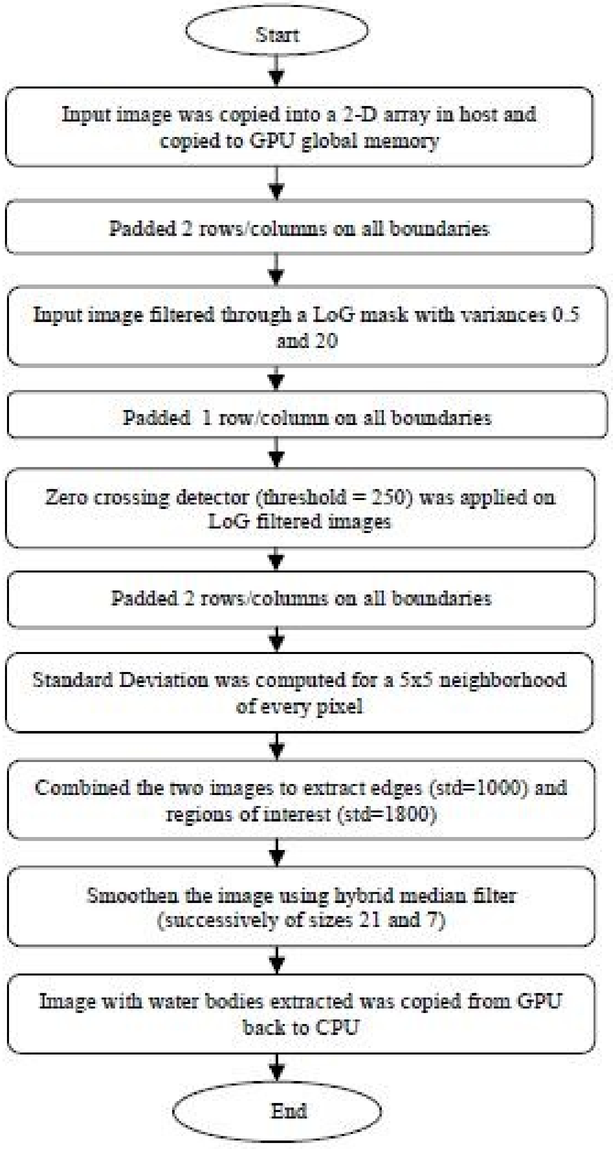

Start
Input image was copied into a 2-D array in host and copied to GPU global memory
Padded 2 rows/columns on all boundaries
Input image filtered through a LoG mask with variances 0.5 and 20
Padded 1 row/column on all boundaries
Zero crossing detector (threshold = 250) was applied on LoG filtered images
Padded 2 rows/columns on all boundaries
Standard Deviation was computed for a 5x5 neighborhood of every pixel
Combined the two images to extract edges (std=1000) and regions of interest (std=1800)
Smoothen the image using hybrid median filter (successively of sizes 21 and 7)
Image with water bodies extracted was copied from GPU back to CPU
End

Fig. 3. Flow Chart for Urban Area Detection [3].

Author	Parallel Approach	Image Processing Method	Data Images
[1]	parallel	Kalman Filtering edge detection	Satellite Images
[2]	TLP, SMID	Canny Operation Edge	Image and Video
[3]	Row-wise Threads	Median Filter	Image and Video

II. Basic Concepts

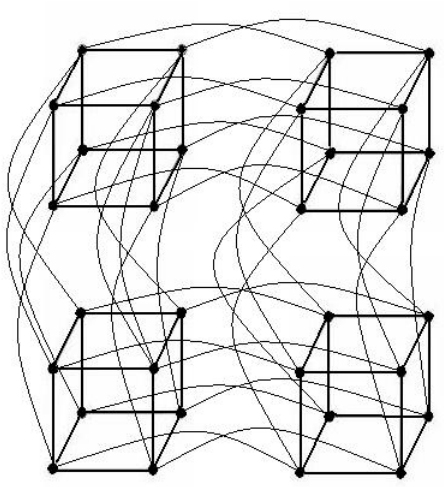


Fig. 4. Architecture proposed based on Hypercube approach for parallel solutions in image processing problems.

A n-dimensional hypercube can be defined by a set of vertices labelled with a binary representation of the numbers 0 to 2n-1. Two vertices are connected if their labels differ in only one bit. Its degree and its diameter is equal to n. This hypercube can be constructed recursively from lower dimensional hypercubes [4], [5]. Consider two n-1 dimension cubes whose vertices are labelled from 0 to 2n-1-1, a n-dimensional hypercube is obtained by uniting the vertices of the same label, from one cube to the other. This presents advantages from the point of view of modularity, since it is possible to gradually increase the size of the network, but this modularity is limited by the maximum number of connections originally planned. The hypercube has an optimal routing algorithm that is simply adapted to an embodiment wired 1. If determining the order of route, for example by decreasing bit numbers, the algorithm is free of blockage. The hypercube topology is much more effective than the mesh topology; for a network of size N, the diameter and the average distance of the hypercube evolve according to $\log^2 N$, whereas in the case of a 2D grid the order is $\sqrt{N}$.

A. Basic Concepts and Hypercube Network Fundamentals

N-cube graph. Is an undirected graph that consists of $k = 2^n$ vertices labelled from 0 to $2^n - 1$, such that there is an edge between any two vertices if and only if the binary representations of their labels differ by only one bit[4].

# nodes. Total number of nodes in a topology for a given dimension [6].

# links. Total number of links in a topology for a given dimension [6].

Cost. A network with a larger diameter has nodes with small degree but suffers from long delay in interprocessor communication [6].

Degree. Is directly linked to the number of ports at a node, and is also an indicator of the cost of the node. [5].

# edges. Number of edges per node; it is best if this number is a constant independent of the network size, because the processor organization scales more easily to systems with large number of nodes [6].

Hamming Distance. The minimum distance between the nodes A and B is equal to the number of bits that differ between the labels for A and B,i.e., to the Hamming distance H(A,B) [4].

Diameter. Is the largest distance between two nodes; low diameter is better, since diameter puts a lower bound on the complexity of parallel algorithms requiring communication between arbitrary pairs of nodes [6].

Average Distance($\bar{d}$). Is the summation of distances of all nodes from a given node (source) over the total number of nodes. Average distance plays a key role in determining the overall delay in a computer network [6].

Bisection Width. The minimum number of edges that must be removed in order to divide the network into two halves (within one) [6].

Message Traffic Density (ρ). This factor is defined as $\rho \equiv \bar{d}N/L$, where L is the total number of links and $\bar{d}$ is the average distance.

B. Approach Proposed: Configuring Hypercube Networks

We propose a novel approach to process image astronomical data using hypercube networks for stellar spectra of 9000. It is proposed a 6-dimensional architecture, with 64 nodes (2^8). Distribution is performed based on the number of galaxies (1400) and quasars (7600): ten nodes will process 1400 galaxies, 140 spectra per node. Fifty nodes are dedicated to process 7600 quasars, with 152 quasars spectra per node. Two nodes collect data and one more node stores the availability of these data. There is only a node without operation. A 64-nodes six-dimensional architecture is proposed (see Figure 5) .

The process begins with the addressing of each node (Figure 6.) The second stage computes the paths between nodes that can be achieved (Figure 7.) The shortest path is computed in order to find the faster solution for image stellar data that can be distributed in our hypercube architecture (Figure 8.) The shortest path and the longest path can be compared for statistical data (Figure 8 versus Figure 9.) For a n-dimensional cube, each CPU has n connections to the other CPUs so wiring complexity increases logarithmically in proportion to the size, and the largest path behaves the same way.

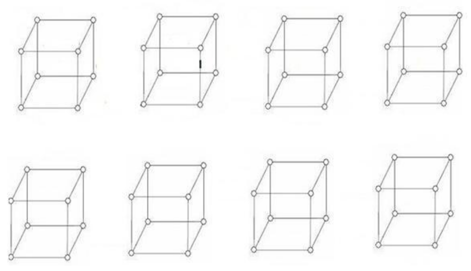


Fig. 5. Hypercube networks.

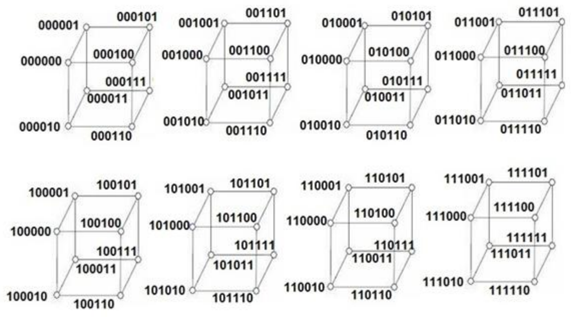


Fig. 6. Binary codes for each dimension and its distribution.

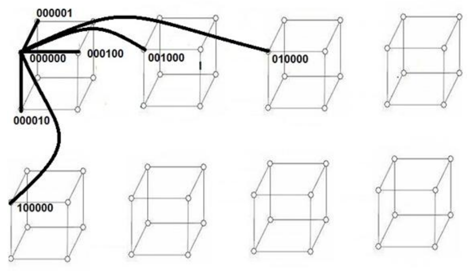


Fig. 7. An n-dimensional network has n possible connections.

III. Conclusions

Parallel computing is a promising approach for solving problems associated with image processing. The quantity of data, the availability of development platforms, but especially the response times involved are details that finally decide the way a solution is developed. All the applications analysed in this work present a valid point of view as solutions, and even some of them can be interchanged. However, the most important conclusion from this analysis is the fact that normally more than one solution can be applied simultaneously when solving the problem. Furthermore, is necessary to determine if the use of resources and time considered for that implementation is worth the expected results.

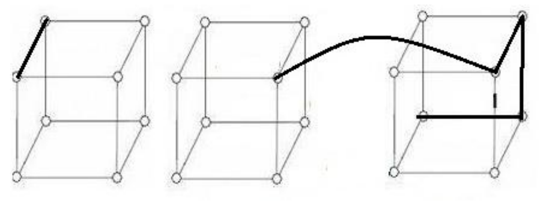


Fig. 8. The shortest route is one movement, while the shortest path between neighbours is four movements.

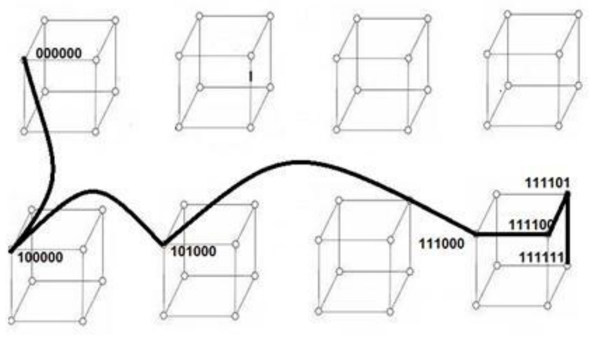


Fig. 9. The longer route is six connections.

References

[1] J. Subash; *Automatic Road Extraction From Satellite Images Using Extended Kalman Filtering And Efficient Particle Filtering* , International Journal of Distributed and Parallel Systems (IJDPS) Vol.2, No.6, Bangalore, India; 2011.

[2] Ming Qi, Guangzhong Sun, Guoliang Chen, *Parallel and SIMD Optimization of Image Feature Extraction*, Procedia Computer Science 4, 2011, pp. 489–498.

[3] K.P. Tejaswi, D. Shanmukha Rao, T. Nair, A. V. Prasad; *GPU Accelerated Automated Feature Extraction From Satellite Images* International Journal of Distributed and Parallel Systems (IJDPS) Vol.4, No.2, March 2013

[4] Y. Saad , M. H. Schults; *Topological Properties of Hypercubes* Search Report YALEU 1985

[5] J. Degila, B. Sanso; *A Survey Of Topologies And Performance Measures For Large-Scale Networks* IEEE Communication Surveys, Fourth Quarter, Volume 6, No.4, 2004

[6] M. V. Rao, N. Chalamaiah; *Hypercube and Its Variant Networks: A Topological Evaluation* Proceedings of the Eighth International Conference on High-Performance Computing in Asia-Pacific Region(HPCASIA0́5), 2005

SESSION

SYSTEMS SOFTWARE + PARALLEL PROGRAMMING MODELS + PETRI NETS + CACHE COHERENCE

Chair(s)

TBA

Automatic Performance Tuning of Pipeline Patterns for Heterogeneous Parallel Architectures

E. Bajrovic[1] **and S. Benkner**[1]
[1]Research Group Scientific Computing, Faculty of Computer Science, University of Vienna, Vienna, Austria

Abstract— *Heterogeneous parallel architectures combining conventional multicore CPUs with GPUs and other types of accelerators promise significant performance gains compared to homogeneous systems. However, exploiting the full potential of such systems is becoming more and more challenging often forcing programmers to combine different programming models and parallelization strategies. A promising approach to coping with the increased programming complexity is the use of parallel patterns for expressing certain types of computations at a high-level of abstraction while relying on the compiler and runtime system to map such patterns onto a heterogeneous system. In this paper we present an approach for automatic performance tuning of high-level pipeline patterns for heterogeneous parallel systems in the context of a task-parallel component-based programming model. Our automatic performance tuning approach attempts to automatically determine the best combination of pattern-specific parameters, parameters exposed by the runtime system, and machine-specific parameters such that execution is optimized for a given workload and target architecture. Experimental results on two state-of-the-art heterogeneous systems demonstrate the effectiveness of our approach.*

Keywords: parallel programming, patterns, autotuning, heterogeneous manycore architectures

1. Introduction

Over the last decade we have seen dramatic changes in the architecture of parallel systems due to the introduction of multicore processors and the shift to heterogeneous parallel systems that comprise different types of execution units specialized for efficiently processing different types of computational workloads. Typically, heterogeneous parallel architectures combine conventional multicore CPUs with GPUs and other types of accelerators. Such systems have become increasingly important since they promise significant performance gains compared to homogeneous systems. However, exploiting the full potential of such systems often requires combining different programming models and parallelization strategies, which significantly increases the complexity of application development.

A promising approach for coping with the complexity of programming heterogeneous parallel architectures is the use of parallel patterns or skeletons [1], [2], [3], [4], for expressing certain types of computations at a high-level of abstraction while relying on the compiler and runtime system to map such patterns onto a heterogeneous system. However, mapping high-level parallel patterns efficiently to different types of heterogeneous target architectures often requires fine-tuning of various parameters at the application level (e.g. replication factors of pipeline stages) or runtime level (e.g. the scheduling strategy), which usually is a time-consuming tasks and requires detailed knowledge of the involved compiler(s) and runtime system(s) as well as of the target architecture. As a consequence, automatic performance tuning techniques (also referred to as *autotuning*) to automatically search for the best combination of such parameters have become of growing interest.

In this paper we present an approach for automatic performance tuning of high-level pipeline patterns for accelerated parallel systems. Our work builds on a component-based task-parallel programming framework that has been developed in the context of the European PEPPHER project [5], which addressed programmability and performance portability for single-node heterogeneous manycore systems. Within the PEPPHER framework, pipeline patterns are realized based on while-loops with source-code annotations [6]. Pipeline stages usually correspond to calls to multi-architectural components, for which multiple implementation variants may be provided. Such component implementation variants may be optimized for different execution units of a heterogeneous target architecture, e.g., for a homogeneous multicore CPU, for a GPU, or for other types of accelerators. At runtime, for each call to a component a task is generated, yielding a dynamic task graph. It is then up to the runtime system to schedule the task graph for efficient parallel execution on the different execution units of a heterogeneous target system. The runtime system chooses for each task a suitable component implementation variant and dynamically scheduling its execution onto a free execution unit of the target architecture such that all available execution units are utilized and overall performance is optimized.

Within the European Autotune project [11] we have integrated the PEPPHER high-level programming framework with the Periscope Tuning Framework [11] for online performance analysis and tuning. Our automatic performance tuning approach takes into account pattern-specific parameters, parameters exposed by the runtime system, as well as machine-specific characteristics in order to optimize the execution of applications with pipeline patterns on heterogeneous parallel systems equipped with CPUs and GPUs.

The remainder of this paper is organized as follows: In Section 2 we provide an overview of the PEPPHER framework and describe the support for high-level pipeline patterns. In Section 3 we describe our pipeline coordination layer which manages the execution of pipeline patterns at runtime and which exhibits different parameters amenable to autotuning. Section 4 provides an overview of the Persicope tuning framework and outlines the tuning of pipeline patterns. In Section 5 we present experimental results using a real-world face detection application. Section 6 discusses related work followed by a conclusion in Section 7.

2. High-Level Programming Framework

The European research project PEPPHER [5] developed a methodology for improving programmability and performance portability for single-node heterogeneous many-core systems. The PEPPHER methodology is characterized by a component-based programming approach in combination with an asynchronous task-parallel execution model.

2.1 Multiarchitectural Components

The central idea is to provide performance-critical parts (typically functions) of applications as components with multiple implementation variants, called multi-architectural components. Each such variant is tailored for a different type of target architecture (CPU, GPU, accelerator) that may be utilized within a heterogeneous many-core system. Component implementation variants may be sequential or parallel and may be implemented with different programming APIs including C/C++, OpenMP, CUDA and OpenCL. All implementation variants of a specific component must adhere to the same component interface. Components and implementation variants are accompanied with meta-data, supplied via external XML descriptors. Such descriptors specify the data read and/or written by a component and provide information about the target platform(s) [7] and about specific resource requirements or constraints.

For constructing applications from components a set of coordination primitives has been developed. Programmers may construct applications at a high level of abstraction by invoking component functionality from C/C++ codes via their interfaces and by using source code annotations (pragmas) to delineate asynchronous (or synchronous) component calls. With this approach, a sequential program spawns component calls, which are then scheduled for task-parallel execution by the runtime system. A source-to-source compiler transforms annotated component calls such that they are registered with the runtime system and generates corresponding glue-code.

2.2 Pipeline Patterns

In addition to the basic coordination primitives for designating asynchronous (or synchronous) component calls we have developed high-level language support for pipeline patterns. Pipeline patterns are expressed using annotated while loops where the loop body comprises calls to multi-architectural components.

An example of a high-level C++ pipeline code for face detection in a stream of images is shown in Figure 1. The first pipeline stage reads images from an input file, the middle stages perform image transformation and face detection, and the last stage outputs the result images where all detected faces are marked with rectangles. For the `detectFace` stage, two different component implementation variants are provided within the PEPPHER framework, one optimized for execution on a conventional CPU core and one optimized for GPUs. These implementation variants have been re-engineered from the OpenCV image processing library [10]. By means of annotations, the user can specify what kind and size of buffers should be generated for passing data between pipeline stages. Moreover, the user can specify a replication factor for individual pipeline stages in order to influence the degree of parallelism during execution. By changing the replication factors and buffer sizes the user can quickly experiment with different configurations of the pipeline. However, the goal of our autotuning approach is to automatically determine the best values for these tuning parameters such that overall execution time is minimized.

```
N = get_max_execution_units ();

#pragma pph pipeline with buffer ( PRIORITY , N*2 )
while ( inputstream >> file ) {
        readImage ( file , image );
       #pragma pph stage replicate (N)
        {
            resizeAndColorConvert ( image );
            detectFace ( image , outimage );
        }
        writeFaceDetectedImage ( file , outimage );
}
```

Fig. 1: A pipeline pattern for face detection in a stream of images.

2.3 Transformation System

A source-to-source compiler transforms pipeline patterns into a C++ code that utilizes a coordination layer for managing parallel execution on heterogeneous many-core architectures at runtime. Pipeline constructs are analyzed in order to determine the structure of the pipeline (stage interconnection) by analyzing the data types of objects passed between pipeline stages. For each stage interconnection corresponding buffer data structures are generated.

The generated target code contains calls to the pipeline coordination layer which comprises various classes for coordinating the execution of pipeline stages on top of the StarPU runtime system [8]. At run-time, component invocations result in tasks that are managed by the StarPU runtime system and executed non-preemptively.

3. Pipeline Coordination Layer

The pipeline coordination layer manages all aspects of execution on a heterogeneous many-core architecture, including the automatic management of buffers for data passed between pipeline stages, the replication of individual stages, and the coordination of task-parallel execution of pipeline stages. Internally, the pipeline coordination layer utilizes the StarPU [8] heterogeneous runtime system.

StarPU is responsible for dynamically selecting suitable component implementation variants for pipeline stages and for scheduling their execution to the different execution units of a heterogeneous many-core system in a performance- and resource-efficient way. StarPU also manages data transfers between execution units and provides support for different scheduling strategies, with the goal of utilizing all execution units of the target architecture. Data transfers for tasks are determined based on the XML meta-data provided in component descriptors, and their costs are taken into account during scheduling by StarPU. In the following we outline the major aspects of the pipeline coordination layer and its interaction with the underlying StarPU runtime system.

The pipeline coordination layer provides several classes for coordinating the execution of pipeline stages. The *PipelineManager* class is used to control multiple pipeline patterns within an application. The *Pipeline* class provides methods for starting, pausing and resuming the execution of a pipeline pattern, and for dynamically reconfiguring the tuning parameters of a pipeline (i.e., changing the replication factor and buffer sizes). The *Stage* class encapsulates information on the stage functionality (i.e., the component that is invoked), connected buffers, predecessor and successor stages, and the stage replication factor. The *Stage* class provides two methods for coordinating the execution of a pipelined application: the method *execute_async()* for posting a stage for execution to the runtime system and the method *callback()* for transferring control back to a stage object after its associated component has finished execution. Every stage instance is executed in an asynchronous fashion.

Figure 2 illustrates how a pipeline pattern is being executed on top of the StarPU runtime system. The *PipelineManager* creates a pipeline object and the corresponding stage and buffer objects. It then starts the pipeline by invoking the *run_pipeline()* method of the pipeline object. The pipeline object then calls the *execute_async()* method for each stage object, which initiates the execution of stages, each within its own thread. Each instance of a stage execution pops input data from the corresponding stage input buffer, and then delegates the actual execution of the stage to StarPU by calling the *post()* method. From this point on StarPU is responsible for executing the stage instance by selecting an appropriate stage implementation variant and scheduling its execution on a suitable execution unit (CPU or GPU). When the stage instance has finished execution, StarPU calls the method *callback()* to pass control back to the stage object which then initiates execution of the next stage instance.

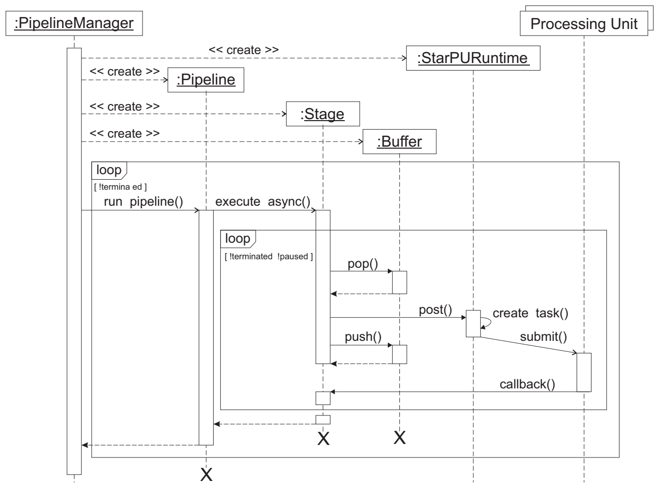


Fig. 2: Pipeline Execution Model.

StarPU relies on a representation of the program as a directed acyclic graph (DAG) where nodes represent component calls (tasks) and edges represent data dependences. The runtime system dynamically schedules component calls to the available execution units of a heterogeneous many-core architecture such that (1) independent component calls execute in parallel on different execution units and (2) the "best" implementation variants for a given architecture are selected based on historical performance information captured in performance models. StarPU also manages data transfers between CPUs and GPUs, ensures memory coherency, and provides support for different scheduling strategies. Besides the well-known EAGER scheduling policy, StarPU also features the Heterogeneous Earliest Finish Time (HEFT) [9] policy. The HEFT policy considers inter-component data dependencies and schedules components to workers taking into account the current system load, available component implementation variants, and historical execution profiles, with the goal of minimizing overall execution time by favoring implementations variants with the lowest expected execution time.

3.1 Tuning Parameters

The pipeline coordination layer enables dynamic reconfiguration by exposing a set of tuning parameters, thus allowing users or external tools to tune the execution of the pipeline in order to achieve a desired goal (e.g., to maximize pipeline throughput). The following tuning parameters are provided: (1) the stage replication factor, which determines the number of stage instances that may be executed in parallel, (2) the sizes of buffers to hold data packets passed between pipeline stages, (3) the number of CPU cores to be used, (4) the number of GPUs to be used, and (5) the scheduling strategy used by StarPU for scheduling component calls to free execution units of the target system.

All these parameters have a profound influence on the performance of applications that rely on pipeline patterns. Finding the best parameter combination for a given application, problem size, and machine configuration is an elaborate and time-consuming task for users and thus should be automated as far as possible.

3.2 Performance Metrics

In order to support automatic performance tuning the coordination layer provides integrated support for measuring the following performance metrics of pipeline patterns:

- Stage execution time - the execution time of an individual instance of a pipeline stage.
- Buffer input processing time - the time to process the input objects of one buffer.
- Buffer output processing time - the time to process the output objects of one buffer.
- Buffer size - the size of individual buffers.
- Pipeline execution time - the overall execution time of one pipeline pattern.

4. The PTF Tuning Framework

The Periscope Tuning Framework (PTF) [11] is an extension of the Periscope online performance analysis tool [12]. PTF aims at providing an integrated infrastructure for performance analysis and automatic performance tuning that can incorporate expert knowledge to guide the search for performance problems and tuned code versions.

PTF facilitates the development of tuning plugins that include codified expert knowledge about the performance characteristics and computational patterns of the target applications and the specific tuning problem. Besides the pipeline tuning plugin presented in this paper, several other tuning plugins (e.g. for tuning of MPI parameters, for master/-worker patterns, and for energy tuning via dynamic voltage and frequency scaing) have been developed in the context of the AutoTune project [11].

Based on performance analysis, tuning plugins identify tuning alternatives, so-called tuning scenarios (i.e. different configurations of tuning parameters), and then proceed to evaluate them. The evaluation of tuning scenarios may be performed online, i.e. during a single application run, which reduces the time required to find the best tuning scenario dramatically.

Automatic performance analysis is based on formalized performance properties, e.g., load imbalance or slow pipeline stages (limiter stages). One or more analysis agents may be used to search for performance properties in the program execution under investigation. Analysis agents communicate with the monitor via the monitor request interface (MRI) linked with the application process(es) to be tuned. The MRI monitor performs the measurements of performance data requested by the analysis agent and transfers the measured performance data to the PTF.

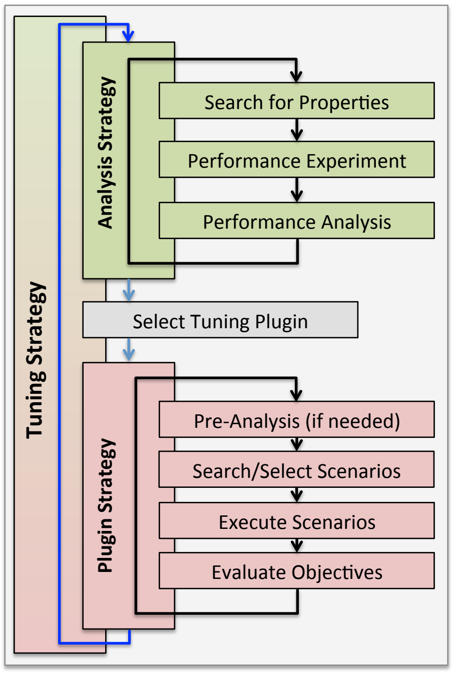


Fig. 3: The PTF Tuning Model

4.1 The PTF Tuning Model

Figure 3 illustrates the PTF tuning model. As shown in the figure, a tuning strategy is comprised of an analysis strategy and a plugin strategy, which may be performed iteratively, depending on the concrete nature of the tuning problem. The analysis strategy guides performance analysis and the search for performance properties, while the plugin strategy guides the search for optimized tuning scenarios. Once the tuning process is finished, a tuning report will be generated that documents the tuning actions recommended by PTF.

The tuning process is usually proceeded with a preprocessing step of the application source files (not shown in Fig. 3). Preprocessing comprises code instrumentation required for performance analysis and static analysis and is either performed by PTF, which includes and integrated instrumenter for C/C++ and FORTRAN, or by external tools. In the case of pipeline patterns, instrumentation is performed by the PEPPHER source-to-source transformation system. During the instrumentation phase, also a SIR file (Standard Intermediate Representation) is generated, which includes static information about the instrumented code regions to be utilized by PTF for performance analysis and tuning.

The analysis strategy guides the search for certain (predefined) performance properties, by performing one or more performance experiments and analyzing the corresponding performance measurements. In case of a pipeline pattern, the analysis strategy searches for a limiter stage, which takes much more time than other stages. If a limiter stage is found, the pipeline tuning plugin is triggered and attempts to

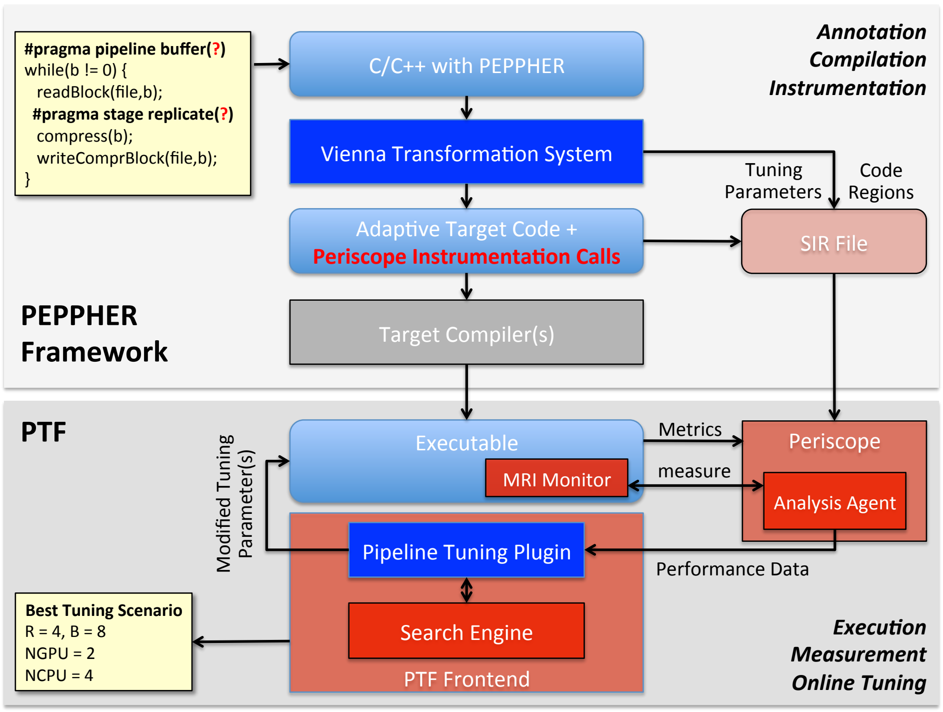


Fig. 4: Integration of the PEPPHER framework for high-level pipeline patterns with the Periscope Tuning Framework. Blue components are specific to the PEPPHER framework, while red components are specific to PTF.

increase the replication factor of the limiter stage such that overall execution time is reduced. In addition to this specific plugin strategy, we have realized a general plugin-strategy finding a configuration of the five pipeline tuning parameters that minimizes overall execution time.

As shown in Figure 3, a plugin strategy is comprised of an optional pre-analysis phase, a phase for searching, selecting and analyzing tuning scenarios within the set of overall tuning scenarios, and phases for executing tuning scenarios and evaluating the tuning objectives such that the best tuning scenarios is identified. For preparation and creation of new tuning scenarios, plugins can access a search interface, which enables to apply different search strategies for finding promising tuning scenarios. In our current implementation of the pipeline tuning plugin we have used exhaustive search. In the future we plan to integrate alternative search strategies.

4.2 Pipeline Tuning Workflow

In the following we describe the major steps of the pipeline tuning workflow according to Figure 4.

First, the application source code is processed by the PEPPHER transformations system, which translates high-level pipeline patterns into a representation that uses the pipeline coordination layer and inserts monitoring calls for obtaining the pipeline-specific performance metrics. In addition, a SIR file (XML intermediate program representation) with information about the relevant tuning parameters and code regions is created. The generated code is then compiled with target specific compilers and linked with the PTF performance monitoring (MRI) libraries. During program execution, the linked MRI monitor is used for communicating measured performance metrics to the PTF and for (re-)setting the values of tuning parameters. The tuning plugin decides the measurements that will be considered for application tuning, and the modified tuning parameters. The pipeline tuning plugin constructs the set of tuning scenarios, executes them by dynamically reconfiguring the pipeline tuning parameters, and reports the best tuning scenario.

5. Experimental Evaluation

For evaluation we use the OpenCV face detection application outlined previously in Figure 1. The application processes a set of 350 images of nHD (640x360) resolution, each containing an arbitrary number of human faces. For the computationally most demanding component `detectFace()`, which is called in the middle stage, two different implementation variants, one for a single CPU core and one for a GPU, were utilized. These implementation variants have been re-engineered from the OpenCV library, which includes both a sequential C++ version and a CUDA version, but which had to be slightly adapted to the PEPPHER component model.

We present speedup measurements and autotuning results on two different CPU/GPU systems. The first machine is equipped with two quad-core Intel Xeon X5550 CPUs

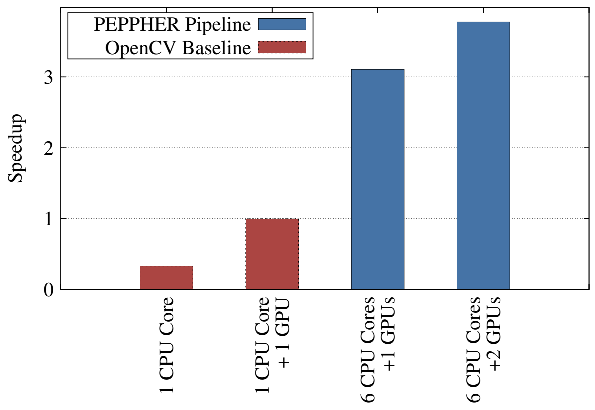


Fig. 5: Speedup results for face detection application on a machine with two quad-core CPUs and two Tesla GPUs relative to the OpenCV baseline version.

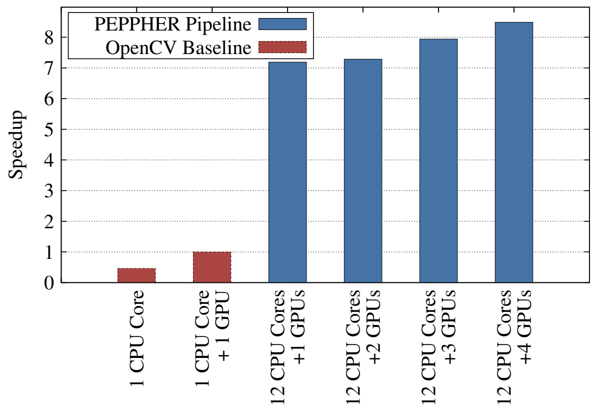


Fig. 6: Speedup results for face detection application on a machine with two octa-core CPUs and four Kepler GPUs relative to the OpenCV baseline version.

(2.66GHz, 24GB RAM) and NVIDIA Tesla C2050 and C1060 GPUs, respectively. The second machine is equipped with two octa-core Intel Xeon E5-2650 CPUs (2.0 GHz, 128GB RAM) and 4 NVIDIA Kepler K20 GPUs. As shown in Figures 5 and 6, we executed the face detection pipeline on different machine configurations and utilized PTF to automatically determine the best combination of tuning parameters such that execution time was minimized.

Figure 5 shows speedup results on the first machine equipped with Tesla GPUs. The second red bar in the figure is the OpenCV baseline version, i.e. using the original OpenCV library which supports using just 1 CPU and 1 GPU. The two blue bars show the results of the autotuned PEPPHER pipeline versions using 6 CPU cores and one or two GPUs, respectively. These results clearly demonstrate that our high-level component-based approach can effectively utilize all execution units of the system. Note also that no source code changes were necessary to run on the two different machine configurations with one and two GPUS, respectively. Using the whole machine a speedup of about 4 has been obtained compared to the OpenCV base version and a speedup of about 12 compared to the single core version.

Using the PTF tuning plugin, we used exhaustive search to find the best configuration for the available tuning parameters. As described in Section 3.1, we considered the following five tuning parameters: (1) stage replication factor of the `detectFace()` stage, (2) input buffer size of the `detectFace()` stage, (3) number of CPU cores, (4) number of GPUs, and (5) the scheduling policy - EAGER (simple greedy scheduler) versus HEFT (Heterogeneous Earliest Finish Time) [8]. In total PTF explored 360 possible configurations, spending about 6 hours in doing so. Finding the best parameter configuration manually would require significantly more time, usually several days of reconfiguration and performance measurement.

In Table 1, we summarize the explored values for each tuning parameter when tuning the face detection application on the first machine. With the best parameter configuration using the whole system (i.e., all CPU cores and all GPUs) the execution time of the face detection application over the complete data set was 8.2 seconds. It used a replication factor of 8, 6 CPU cores, 2 GPUs, buffer size of 32 and HEFT scheduling policy. The slowest configuration that utilized the whole system resulted in an execution time of 19.6 seconds.

Tuning parameter	**Possible values**	**Best configuration**
Replication factor	1, 2, 4, 8	8
Number of CPU cores	1, 2, 4, 6, 8	6
Number of GPUs	0, 1, 2	2
Scheduling policy	"EAGER", "HEFT"	"HEFT"
Buffer size	8, 16, 32	32

Table 1: Possible values of tuning parameters (first machine).

Figure 6 shows speedup results on the second machine equipped with Kepler GPUs. Again, the second red bar in the figure is the OpenCV baseline version. The four blue bars show the results of the PEPPHER pipeline version using one up to four GPUs as well as 12 CPU cores. Again no code changes were required to run the application on these different machine configurations. Using 12 CPU cores and 1 GPU delivers a speedup of about 7 over the OpenCV baseline version that uses one CPU and one GPU. Adding more GPUs results in only modest speedup increases, which can be mainly attributed to the rather low resolution of the images. We expect that for higher resolutions greater speedups with multiple GPUs would be possible due to the increased computational complexity.

Also on the second machine we used to PTF to find the best configuration of tuning parameters by exploring 1470 different combinations. The best parameter configuration using all CPU cores and all GPUs of the second machine resulted in an execution time of 4.6 seconds, with a replication factor of 16, 12 CPU cores, 4 GPUs, buffer size of 32 and HEFT scheduling policy. The slowest configuration that utilized the whole system resulted in an execution time of 15.3 seconds.

6. Related Work

Due to the increasing complexity and diversity of parallel architectures, there is a growing interest in automatic performance tuning techniques. Existing autotuning efforts include self-tuning specialized libraries (e.g., linear algebra or signal processing) like ATLAS[14] or FFTW[15], tools that automatically search for best combination of compiler optimization parameters [16], [17], and tools that search for best values of application-level parameters [18], [19].

Our approach mainly deals with tuning of parameters exhibited by a runtime library (i.e. our coordination layer) and thus our work is more akin to the third group (application tuning) than the first one (self-tuning libraries), because we execute computational components that are not part of the library and can behave very differently from each other. Our efforts are close to the emerging area of (possibly automated) tuning of OpenCL or CUDA parameters [20], [21].

7. Conclusion

In this paper we presented our work on autotuning support for high-level pipeline patterns for heterogeneous many-core architectures. We have developed a pipeline tuning plugin for the Periscope Tuning Framework, in order to automatically determine the best combination of performance relevant tuning parameters exhibited by the pipeline coordination layer.

In our future work we will experiment with different search strategies and investigate methods for continuous online tuning, such that pipeline patterns are automatically adapted to changing work loads or varying target machine configurations. Moreover, we will extend our work to other common parallel patterns and other architectures [22].

Acknowledgment

This work was supported by the European Commission's FP7 project AutoTune under grant no. 288038 (see http://www.autotune-project.eu).

References

[1] T. Mattson, B. Sanders, and B. Massingill, "Patterns for Parallel Programming," Addison-Wesley, 2005.

[2] N. Bell and J. Hoberock, "Thrust: A Productivity-Oriented Library for CUDA," in *GPU Computing Gems, Jade Edition* (W. mei Hwu, ed.), Morgan Kaufmann, 2011.

[3] J. Enmyren and C. W. Kessler, "Skepu: a multi-backend skeleton programming library for multi-gpu systems," in *Proceedings of the Fourth International Workshop on High-Level Parallel Programming and Applications*, HLPP '10, (New York, USA), ACM, 2010.

[4] J. Dokulil, E. Bajrovic, S. Benkner, M. Sandrieser, and B. Bachmayer, "HyPHI – Task Based Hybrid Execution C++ Library for the Intel Xeon Phi Coprocessor," in *42nd International Conference on Parallel Processing (ICPP-2013)*, Lyon, France, 2013.

[5] S. Benkner, S. Pllana, J. L. Träff, P. Tsigas, U. Dolinsky, C. Augonnet, B. Bachmayer, C. Kessler, D. Moloney, and V. Osipov, "PEPPHER: Efficient and Productive Usage of Hybrid Computing Systems," *IEEE Micro*, vol. 31, no. 5, pp. 28–41, 2011.

[6] S. Benkner, E. Bajrovic, E. Marth, M. Sandrieser, R. Namyst, and S. Thibault, "High-Level Support for Pipeline Parallelism on Manycore Architectures," in *Euro-Par 2012 Parallel Processing - 18th International Conference*, vol. 7484, pp. 614–625, 2012.

[7] M. Sandrieser, S. Benkner and S. Pllana, "Using Explicit Platform Descriptions to Support Programming of Heterogeneous Many-Core Systems," *Parallel Computing*, Volume 38, Issues 1-2, Pages 52-56, January-February 2012.

[8] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, "StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures," *Concurrency and Computation: Practice and Experience, Special Issue: Euro-Par 2009*, vol. 23, pp. 187–198, Feb. 2011.

[9] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," Parallel and Distributed Systems, IEEE Transactions on, vol. 13, no. 3, 2002.

[10] B. Gary, *Learning openCV: Computer Vision with the openCV Library*. O'Reilly USA, 2008.

[11] R. Miceli, G. Civario, A. Sikora, E. Cesar, M. Gerndt, H. Haitof, C. Navarrete, S. Benkner, M. Sandrieser, L. Morin, and F. Bodin, "AutoTune: A Plugin-Driven Approach to the Automatic Tuning of Parallel Applications," in *Applied Parallel and Scientific Computing*, vol. 7782 of *LNCS*, pp. 328–342, Springer, 2013.

[12] S. Benedict, V. Petkov, and M. Gerndt, "PERISCOPE: An Online-Based Distributed Performance Analysis Tool," in *Tools for High Performance Computing 2009* (M. S. Mueller, M. M. Resch, A. Schulz, and W. E. Nagel, eds.), pp. 1–16, Springer, 2010.

[13] M. Gerndt and E. Kereku, "Selective Instrumentation and Monitoring," in *Proceedings of 11th Workshop on Compilers for Parallel Computers (CPC 04), Kloster Seeon*, pp. 61–74, 2004.

[14] C. Whaley, A. Petitet, and J. J. Dongarra, "Automated empirical optimization of software and the ATLAS Project," *Parallel Computing*, vol. 27, 2001.

[15] M. Frigo and S. Johnson, "FFTW: an adaptive software architecture for the FFT," in *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, vol. 3, pp. 1381–1384, 1998.

[16] S. Triantafyllis, M. Vachharajani, N. Vachharajani, and D. August, "Compiler optimization-space exploration," in *Code Generation and Optimization, 2003. CGO 2003. International Symposium on*, 2003.

[17] M. Haneda, P. M. W. Knijnenburg, and H. A. G. Wijshoff, "Automatic selection of compiler options using non-parametric inferential statistics," in *Parallel Architectures and Compilation Techniques, 2005. PACT 2005. 14th International Conference on*, pp. 123–132, 2005.

[18] I.-H. Chung and J. K. Hollingsworth, "Using information from prior runs to improve automated tuning systems," in *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*.

[19] Y. L. Nelson, B. Bansal, M. Hall, A. Nakano, and K. Lerman, "Model-guided performance tuning of parameter values: A case study with molecular dynamics visualization," in *International Parallel and Distributed Processing Symposium*, pp. 1–8, 2008.

[20] S. Ryoo, C. I. Rodrigues, S. S. Stone, S. S. Baghsorkhi, S.-Z. Ueng, J. A. Stratton, and W.-m. W. Hwu, "Program optimization space pruning for a multithreaded GPU," in *Proceedings of the 6th annual IEEE/ACM international symposium on Code generation and optimization*, pp. 195–204, 2008.

[21] Y. Liu, E. Zhang, and X. Shen, "A cross-input adaptive framework for GPU program optimizations," in *International Parallel and Distributed Processing Symposium*, 2009.

[22] J. Dokulil and S. Benkner. "Automatic Tuning of a Parallel Pattern Library for Heterogeneous Systems with Intel Xeon Phi," in *12th International Symposium on Parallel and Distributed Processing with Applications (ISPA 2014)*, Milan, Italy, 2014.

Exploiting Reuse-Frequency with Speculative and Dynamic Updates in an Enhanced Directory Based Coherence Protocol

Nilufar Ferdous[1], **Monobrata Debnath**[1], **Byeong Kil Lee**[2], **and Eugene John**[1]
[1] Department of Electrical and Computer Engineering , The University of Texas at San Antonio,
San Antonio, TX-78249, USA
[2] Samsung Research, Seoul, South Korea

Abstract - *As the number of cores increases on chip multiprocessors, cache coherence is fast becoming a major impediment in improving the performance of the multi-cores. This is exacerbated by the fact that the interconnection speeds does not scale well enough with the speed of the processors. To ameliorate these limitations, several mechanisms were augmented to the cache coherence protocols to enhance the performance of the multiprocessors. These mechanisms include policies such as write-update policy, write-invalidate policy etc. However, it has been previously shown that pure write-update protocol is highly undesirable because of the heavy traffic caused by the updates. On the other hand, write-invalidation protocol is not the optimal solution as many of the sharers of the cache blocks may be reused in the near future. In order to increase the efficiency, we introduce a novel update mechanism which uses reuse frequency and last touch time of each cache block as metrics to take the decision dynamically whether it will retain its write-invalidate strategy or will update the sharers as in write-update protocol. Our research enhances the speedup of MOESI cache coherence protocol by 12 % (average) and reduces L1 cache miss rate by 17%. It also reduces the network power consumption by as much as 26% (5% in average) and the network latency as much as 10%.*

Keywords: MOESI; Cache Coherence Protocol; last touch time; reuse frequency; core locality;

1 Introduction

The current trend in chip multiprocessor (CMP) is to incorporate large number of cores. The earlier implementation of multi-core technology integrated small number of cores (two to four) in to the CMP. But now the industry is heading towards larger number of cores on each processor to enhance the throughput. For example, IBM Cell Processor has eight cores [14], and Tilera TILE64 has [15], 64 cores incorporated in a single chip. Along with these technical trends, the cache coherence complexity has also increased significantly.

Cache coherence is the mechanism that allows maintaining the state and value of a cache block in the caches of a chip multiprocessor (CMP). It ensures that no data is lost or overwritten before the data is transferred from a cache to the target memory. Since cache coherence involves significant amount of communication over wires, wire speed and bandwidth are the major concerns for the improvement of the performance and scalability of the cache coherence protocol. Due to the fact that, in a CMP the interconnection speed does not scale well enough with the increase in core speed, smart mechanisms and novel polices are necessary for accelerating the performance of the coherence protocols. Chip multiprocessors with a small number of cores can use a snoop-based coherence protocol which relies on a broadcast-based interconnect [1]. However, broadcast based interconnect uses the bus which degrades the performance as the number of cores increases. As snoop-based protocol lacks scalability, and therefore directory based protocol [2] is widely used in a large scale CMP. The basic idea is to keep a directory entry for every memory line. This entry consists of its state and a sharing code indicating the caches that contain a copy of the line, which are termed as sharers of a cache block. On a cache miss, each coherence transaction is sent to a directory controller which, in turn, using its corresponding entry, redirects it to the cores caching the line, e.g., redirects to the sharers of the cache line.

This paper examines the design of effective coherence mechanisms for a directory based multi-core architecture that has a shared last level cache (L2 cache as LLC) which maintains MOESI (*Modified, Owner, Exclusive, Shared, Invalid*) [5] cache coherence protocol and ensures the correctness of the cache coherence substrate via token counting. In this work, the directory-based MOESI token protocol is used instead of the broadcast–based token protocol (TokenB) [1] to enhance the scalability of the protocol. We begin by exploring the directory-based MOESI token cache coherence protocol(*MOESI_CMP_Token*)for multi-core processors in a CMP and adapt certain mechanisms to attune the directory based MOESI protocol for boosting the performance in terms of execution time, power consumption, and network latency.

The conventional MOESI coherence protocol is a write-invalidation based protocol and has no mechanism for updating the sharers. In a pure *write-update* cache coherence protocol, all the sharers are updated according to the new modified value whereas in a *write-invalidation* based coherence protocol the existing sharers of the cache block are not updated according to the new value. Previous researches [13] [22] have already shown that *pure write-update* is highly undesirable because of the heavy traffic caused by the updates. It has also been shown that in a pure write-update protocol the network traffic increases with the increase of the CMP size [22]. In this paper we show that simply implementing write-invalidate based directory protocol does not provide the most effective solution. In our proposed mechanism, we keep record of the access frequencies of the cache blocks, e.g., the number of times a cache block is reused in the past, and term this metric as *reuse frequency* (RF). We also consider the time-based *aging* of the accesses of the cache blocks for better speculation of reuse in the *near-immediate future* and have termed the last access time as *last touch time*. In our work, we have incorporated the concept of *temporal locality* of a cache block by means of its *reuse frequency* and *last touch time*. Many of the cache blocks in the shared LLC may show very high *temporal locality* while the others may not. If we can update the sharers of the cache blocks which show high temporal locality, then many of the requests of the local cores can be satisfied by the updated sharers' content. This will eventually improve the performance of the cache coherence protocol in terms of execution time and power consumption.

To the best of our knowledge, our proposed work, *Exploiting Reuse-Frequency with Speculative and Dynamic Updates in an Enhanced Directory Based Coherence Protocol (RFU-Dir)*, is the first one that applies a smart mechanism policy to dynamically update the sharers of only those cache blocks who are highly reused in the recent past. The updates are dynamic because after each and every write back of an individual cache block in the LLC, the decision to update is taken dynamically by the *Dynamic Speculative Update Mechanism Policy (DSP)*. In this work, we show that *RFU-Dir* enhances the speed up of the directory based MOESI cache coherence protocol by 12%, decreases L1 miss rate by 17% (average) and reduces the L1 miss latency by 3.2% (average). Our proposed work also decreases network latency and L1 replacement as much as 10% and 90% respectively. The network power consumption and the directory access are also decreased by 5 % (average) and 13% (average) respectively.

The remainder of the paper is structured as follows. Section 2 identifies the motivation of our work, Section 3 presents the dynamic speculative update mechanism and Section 4 presents the methodology with the experimental results. Related work is summarized in Section 5. Finally, Section 6 outlines the main conclusions of this work.

2 Motivation

Previous researches have shown that, if a cache block is used by a core in a CMP, then there is a high possibility that the particular cache block will be reused by that core in the *near-immediate* future. In other words, if at time=t_0, if core1 issues an address for a cache block '*A*' then within a *near-immediate* future time interval, $t_0 \leq t \leq t_0+k$, where k is an integer, there is a high possibility that core1 will again issue request for that cache block '*A*'. In our work, we term this phenomenon as *core locality* for a cache block residing in the LLC. The conventional pure *update–based* protocol updates the sharers of each and every cache blocks [13]. Updating the sharers of each and every cache blocks demands excessive bandwidth which has already led the update-based protocol to virtual extinction, especially in a broadcast-based snoopy protocol [13]. This situation gets worse as large number of cores is incorporated in a CMP. On the other hand, a pure write-invalidate protocol does not take advantage of the updates of the potential sharers which may show very high temporal locality. In order to overcome these limitations, we make use of the *reuse frequency* and *last touch time* of each cache block as criteria for the sharers to be updated or not.

To understand the motivation of this work, let us now assume a scenario where a READ (L1_GETS) request is issued for a cache block 'A' by core1in a 64-core CMP. Due to a read miss in the local core, the request (L1_GETS) is sent to the shared LLC and the desired data is sent back to the local cache of core1. Let us assume that within a short interval of time, in the same way core3, core5, and core19 also have issued L1_GETS request for the same cache block 'A' and eventually all these cores have cached a local copy of the block 'A'. So core1, core3, core5 and core 19 are now the sharers of the cache block 'A' residing in the LLC.

Now, let us assume another scenario that, at time t=t_0, core17 has issued a WRITE request (GETX) for 'A' in the LLC and at time t= t_0+n, where n is an integer, the modified cache block is written back to the LLC. In the write-invalidate protocols, none of the sharers are updated after the write back to the LLC. However, due to the existing tendency of *core locality*, there is a high possibility of a core to reissue request for the same cache block in the *near-immediate future.* So in the *near–immediate future*, if core 1, core 3, core 5 and core 19 again reissues request for cache block 'A' then all these requests will result in coherence misses. This will trigger many message passing transactions. This will increase the power consumption, network latency, bandwidth contention and the execution time. The situation may get worse as the size of the CMP gets larger. A novel and smart update mechanism policy can improve the performance of the CMP by considering the limiting factors of the pure update protocol. In our work, we have updated the sharers of only those cache blocks which have more potential to be reused in the near future and ignore the updates of those sharers which are less probably to be used in the future.

3 Dynamic speculative update mechanism

3.1 Speculation of high reuse of cache blocks via reuse frequency and last touch time

High *reuse frequency* and *last touch time* in the *near-immediate* past enhance the possibility of a cache block to be reused in the *near-immediate* future. So, in our work we have considered these two metrics to dynamically update the sharers. To increase scalability we have considered the directory based coherence protocol instead of the snoopy based protocol, to minimize the bandwidth overhead with the increase in the number of cores. When a cache block is not present in their local caches then the request (*L1_GETS*) is forwarded to the shared LLC. This situation is depicted in Figure 1. In our work, each cache block in the LLC is associated with a *reuse frequency* counter which is initialized to zero and is reset when the cache block is evicted from the LLC. The *reuse frequency* counter is incremented with each request that comes to the LLC for that cache block. A *reuse frequency* counter achieves high value when several cores have issued requests for the same cache block. The high reuse frequency of a cache block in the shared LLC also depicts the fact that a particular LLC cache block has high possibility of having multiple sharers. Again, due to *core locality*, there is a high possibility that the same core reissues requests for a cache block within a short interval of time. For this reason, if the sharers are updated based on the *last touch time* then the performance can also be improved due to the tendency of temporal locality of the cache blocks.

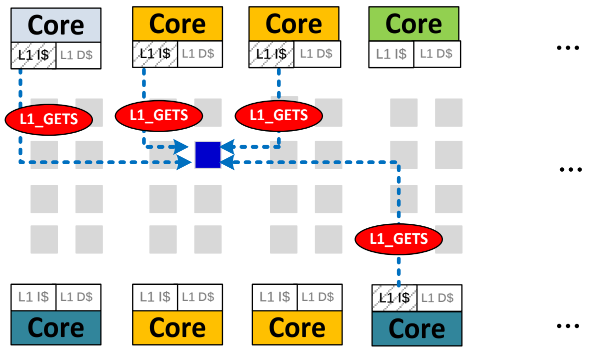


Figure 1. Several *READ* Requests (*L1_GETS*) is coming from different L1 caches to the same cache block in the LLC because cache miss occurs in their corresponding L1 caches.Thus increasing the *reuse frequency* of the cache block in the LLC.

3.2 Implementation of dynamic speculative update mechanism policy

In this work, we introduce an intelligent and novel scheme of speculative update of the sharers which enhances performance of the write-invalidation based directory protocol in terms of execution time, network latency and power consumption. In our proposed work, wherever a write back occurs in the LLC, the sharers are not updated for every cache block. Instead the Dynamic Speculative Update Mechanism Policy (DSP) is being consulted. The sharers are updated only when the cache block is being reused again and again in the past and has a last touch time in the near immediate past. Figure 2(a) depicts the scenario of consulting the DSP and 2 (b) depicts that after the condition checking of the reuse frequency and the last touch time, all the sharers are being updated. The constraints of the Speculative Dynamic Update Mechanism Policy are described below:

Speculation of reuse via high reuse frequency: The reuse frequency of the cache block, whose sharers are to be updated, should be greater than the Threshold value (experimental value of Threshold=13) to assure the higher possibility of being reused in the future, i.e. Reuse Frequency>Threshold.

Speculation of reuse via last touch time: A cache block may be reused again and again long before in the *distant past*, but may not be reused in the *near immediate future*. Updating the sharers of such a cache block with high *reuse frequency* may cause a hazard to performance of the CMP. Our work ensures the reusability of a cache block in the *near immediate future* by considering its *last touch time*. A cache block which is touched in the *near immediate past* has a high possibility of reuse in the *near immediate future* due to its temporal locality. In this work, the life span time of a cache block is divided into many parts, such as, *near immediate* past, *near immediate* future, *distant* past. We define the *near immediate* past as, $t_0 \leq t \leq t_0+k$, where k is a integer (experimental value of *near immediate past* is time range of 2000 microsecond in the past from the current *touch time* of a cache block) and *distant* past as, $t<t_0$, (experimental value of *distant past* is more than 2000 microsecond time span in the past from the current touch time of a cache block).

3.3 Space conflict for the speculative update

In our proposed work, when a L1 cache receives a speculative update and it does not have enough space to hold the line then the LRU line of that L1 cache is evicted. The LRU line is evicted instead of cancellation of the update request because the cache blocks with high *reuse frequency* has a higher possibility of reuse in the near future compared to the least recently used block.

3.4 Extra hardware

Reuse frequency counter is used to keep record of the reuse frequency. Each cache line is associated with a reuse counter and a 32 bit buffer is used to keep track of the *last touch time*. The *reuse frequency* counter is assigned an initial value of zero and is incremented when a request to that cache block is received. For calculating the *reuse frequency*, we used a 4 bit counter per cache line. Therefore, extra hardware cost for counters:

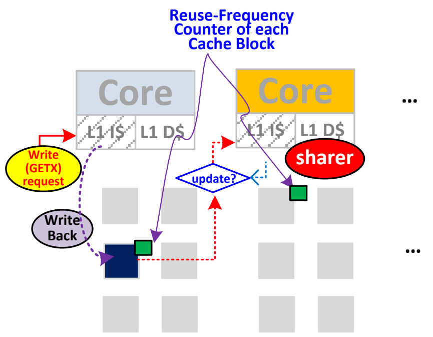
= (#of cache lines *counter size)/LLC size
= [{(1 bank size)*(# of banks)*(counter size)}/ (line size)] / [256*1024*8*# of banks]
= [{(256*1024)*64*8*4}/ (64*8)] / [256*1024*8*64]
= .007 (0.07% of total LLC size).

4 Evaluation methodology and results

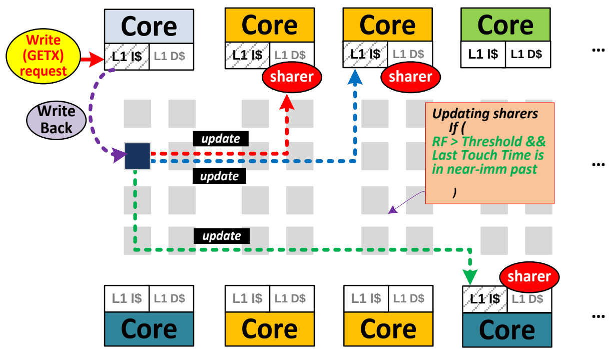
In this section, we first describe our experimental setup. Then we present the experimental results for our proposed model validation. We contrast the proposed work with the existing invalidation based MOESI token coherence protocol (*MOESI_CMP_Token*) to verify the impact of our proposed work in terms of performance evaluation.

4.1 Experimental setup

In order to evaluate the proposed *RFU-Dir* scheme, we set up a NUCA-based L2 cache model and the parameters are listed in TABLE I. We evaluate our system using the Virtutech Simics full system execution-driven simulator along with GEMS [4] timing model. The network latency and power is measured by GARNET simulator [6] and Orion [7] simulator respectively. To simulate real applications' L1 and L2 cache behavior, we ran PARSEC 2.1 benchmark suit.



2(a). The Cache Controller of the L2 bank in the LLC is consulting the *Speculative Update Mechanism Policy* whether the sharers should be updated or not.



2(b). Mechanism of updating the sharers in *RFU-Dir*.
Figure 2. Sharers Update Mechanism in *RFU-Dir*.

We assume a MOESI token based cache coherence protocol (*MOESI_CMP_Token*) among the L1 caches and L2 caches. We also assume a two level cache hierarchy where L2 is the shared Last Level Cache (LLC) shared by 64 cores in a Chip Multiprocessor (CMP). All the L1 caches are local to the cores. So there are 64 L1 caches. L1 caches are split into L1-I (instruction) and L1-D (data) caches. In the protocol, the L2 cache is a Non Uniform Cache Architecture (NUCA) based model [17] and has 64 banks. We assume the protocol to be directory based which has 64 directories. Since our primary focus is to reduce the coherence miss rate, improve the execution time and reduce the power consumption, this processor model is sufficient to evaluate our scheme's fundamental performance. We consider the applications from the PARSEC 2.1[8] benchmark suites. We used the default input set and sim-large configuration for PARSEC 2.1 applications. PARSEC 2.1 applications were run to completion starting from the beginning of their parallel sections and the applications were run for a billion instructions starting from their region of interest (ROI).

4.2 Experimental results

In this section, the results of the comparative evaluation between the conventional directory based MOESI coherence protocol and our proposed work is presented. We considered the benchmarks from the PARSEC benchmark suit that has a certain number of sharers [8].We did not consider the benchmarks that have no sharing patterns. In our proposed scheme, the threshold value of *reuse frequency* (RF) and the *near-immediate past* are varied along a range of values but in our experiment, we observe that *reuse frequency=13* gives the most optimal result in most of the cases. The value of the most effective *reuse frequency* depends upon the benchmarks that are used. In our case, when the *reuse frequency* is a very small value, e.g. *reuse frequency* =3, then the result is not enhanced. This is because of the fact that when a cache block is touched only a fewer number of times, e.g. 3 times, then the possibility of reuse of the cache block is much lesser in the future. So, if we update sharers of the cache block with low *reuse frequency* then the performance will not be improved. Again, when the *reuse frequency* increases then the possibility also increases that the cache block will be reused in the future. In our experiment, when *reuse frequency* is selected within a range 10 to 13 the performance is improved. However, increasing the *reuse frequency* value greater than 13 degrades the performance because in this case the cache blocks are overly utilized and has a very poor possibility of reuse in the future. So updating the sharers increases the execution time and the network power consumption. *T*he *near-immediate past* is considered as an approximate time range of 2000 microsecond in the past.

TABLE I. PARAMETERS FOR THE SIMULATED PROCESSORS

PARAMETER	***VALUE***			
Number of Chips	1			
Number of Cores per chip	64			
Processor Model	In-order			
OS	Solaris 10			
CPU	Ultrasparc-iii-plus			
Main Memory Latency	200 cycle			
Directories per chip	64			
Network Link latency	1 cycle			
L1 Cache (ICache,DCache)	**CacheSize**	**SetAssociativity**	**Latency**	**BlockSize**
	64KB	4	3 cycle	64B
L2 Cache Bank	**CacheSize**	**SetAssociativity**	**Latency**	**BlockSize**
	256KB	4	12 cycle	64B

4.2.1 Analyzing cache miss rate and its consequences

Our proposed work decreases the L1 cache miss rate by 17 % (average) by dynamically updating the local copies which have higher possibilities to be reused in the *near* *immediate future*. So the possibility is high that when the requests are issued for those data blocks they could be satisfied by their local caches. Thus it decreases the L1 cache miss rate. Figure 3 shows the comparison of the L1 cache miss rates (MPKI -Misses per Kilo Instructions) of the conventional *MOESI* protocol and our proposed work, *RFU-Dir*. From Figure 3, we can see that for the benchmarks bodytrack and fluidanimate, the L1 cache miss rate decrease by 30% and 35% respectively. Ferret, freqmine, raytrace, swaptions and vips show a cache miss rate reduction of 8%, 4%, 5.5%, 2 % and 2.5% respectively. In this work, we have updated the sharers of the highly utilized cache blocks so that they may be reused. Here, we can observe that the reduction of the L1cachemiss rate is larger for fluidanimate than for vips since fluidanimate has larger degree of sharing patterns than that of vips. The number of sharers of the cache blocks varies from one application to another. Some applications have a higher degree of sharing pattern while others have a lower degree. In our work, those applications whose cache blocks have larger number of sharers contribute more cache hit rate. When a data block is requested by the core and it is not present in the local cache then the request in sent to the shared LLC. After finding the data from the lower level cache (LLC) or from the main memory the data block is sent to the local cache. If the local cache does not have enough space to hold the data the Least Recently Used (LRU) block is then replaced by the incoming data block. It can be observed that our work results in increased L1 cache hit rate. This means that many of the issued requests are satisfied by the L1 caches. This minimizes the cache block requests to the LLC. As a result, the total numbers of L1 replacements are decreased. Figure 4 shows the percentage of the change in L1 replacements in our proposed *RFU-Dir*. Our proposed work decreases the replacement of the cache lines of the L1 cache by 44% in bodytrack, by 4% in fluidanimate, by 3% in freqmine, by 90%, raytrace 29%, and by 6% in swaptions. However, vips does not show any improvement as the cache blocks of vips exhibits less temporal locality.

We also notice that the READ (L1_*GETS)* requests are issued to the LLC as a result of the cache misses in the local caches. In our proposed work, many of the requests issued by the cores are satisfied by the updated values of the sharers residing in the L1 cache. The decrease in the number of the READ (L1_GETS) requests, coming from the L1caches to the shared LLC, is one of the consequences of the increase in the hit rate of the L1 caches. Figure 5 shows the decrease of the total number of *READ (L1_GETS)* requests coming from L1caches to the shared LLC.

4.2.2 Analyzing execution time

Speculative and dynamic updates of the sharers increases the L1 cache hit rate which eventually decreases the directory accesses and network latency. Directory accesses occur on the consequences of cache misses which cause long latencies and extra performance overhead. The increased L1 cache hit rate is able to minimize many unnecessary transactions such as ACK, SEARCH_DATA, SEND_DATA, and NACK which also minimizes the average network latency. DSP also decreases the L1 miss latency. The decreases of these parameters enhance the speedup of our work. Figure 6 shows the increase in speedup of each benchmark. From the graph, we can observe that our proposed work maximizes the speedup of fluidanimate and bodytrack. We have observed that the LLC cache blocks of bodytrack have a large number of sharers, than many other benchmarks [8]. So, constraints of the DSP cause a significant improvement in the speedup of the benchmark. Ferret, freqmine, raytrace, swaptions and vips show a certain degree of cache blocks' sharing pattern [8]. For this reason, our work increases the speed up of these benchmarks to a certain degree in compare to fluidanimate. The execution time is also proportional to the total latency of the network. If the network latency of a CMP can be minimized then it also influences the total execution time. Figure 7 shows the comparison of the average network latency.

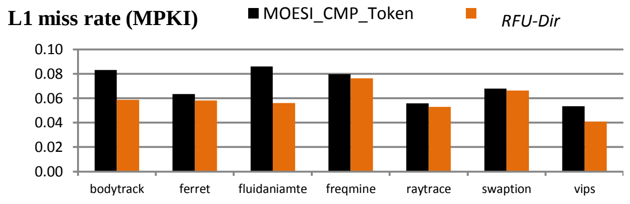


Figure 3. Comparison of the decrease in L1cache miss rate (MPKI).

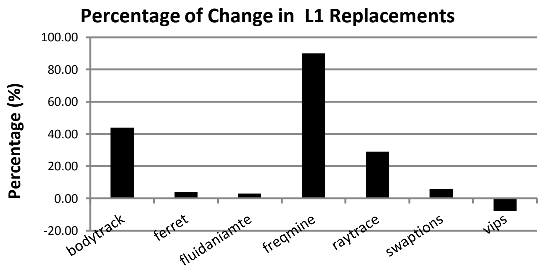


Figure 4. Percentage of change of L1 Replacements in *RFU-Dir* in compare to the Convectional directory based MOESI Cache Coherence.

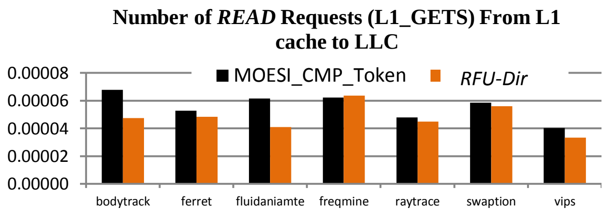


Figure 5. Comparison of the number of *READ* requests (normalized to total number of instructions) coming to the shared LLC as results of L1 misses.

Figure 8 shows the comparison of the normalized number (normalized to the total number of the instructions)) of directory access for tag searching. From the analysis, we can see that the increased L1 cache hit rate of *RFU-Dir* decreases the total directory accesses. From Figure 9, we can also see the percentage of decrease in L1 miss latency in *RFU-Dir*. All these factors contribute to the improvement in the speed up of *RFU-Dir*.

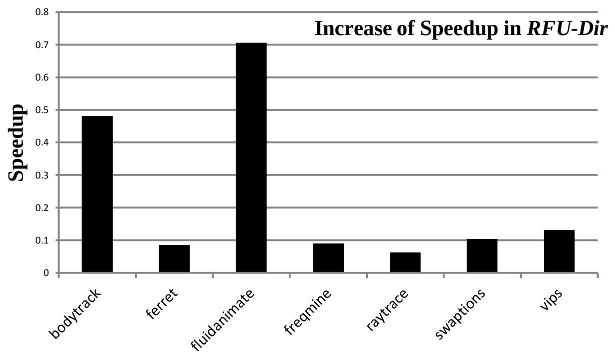


Figure 6. Increase in Speed up of RFU-Dir in compare to the conventional Directory based cache coherence (MOESI_CMP_Token).

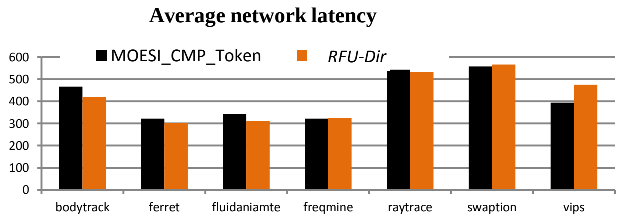


Figure 7. Comparison of the average network latency (microsecond).

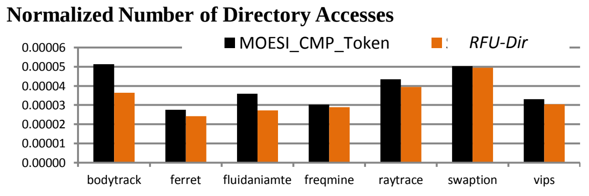


Figure 8. Comparison of the Normalized number of Directory Accesses (normalized to total number of instructions) in Directory based cache coherence (MOESI_CMP_Token) and *RFU-Dir*.

4.2.3 Analyzing power consumption

Updating the sharers speculatively and dynamically improves the hit rate which in turns decreases the total number of message passing mechanisms which would require extra clock cycles. All these activities may include the searching of the tag array in the L1 cache, LLC and also in directory, the initiation of the requests of the transactions, sending DATA to the requestor, acknowledgements (NACK, ACK) etc. All these activities not only increase the network latencies and occupy the precious on-chip bandwidth but also increase the network power consumption. Figure10 shows the change of the network power consumption in *RFU-Dir* compared to the conventional MOESI_CMP_Token. It can be observed that as the number of sharers in vips is less the locality of the reused cache blocks is not dominant.

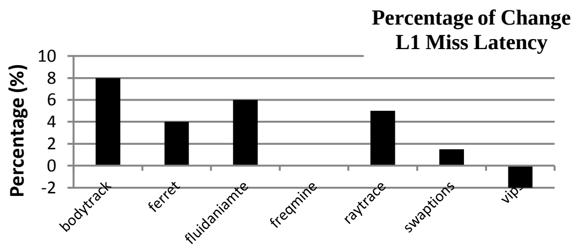


Figure 9. Percentage decrease of L1 cache miss latency in *RFU-Dir* in compare to the conventional Directory based cache coherence (MOESI_CMP_Token).

5 Related works

Previous works in cache coherence have included prediction, which have covered a broad spectrum. In [18], Eggers and Katz compare write-invalidate and write-update snoopy-cache protocols. Eggers and Katz also evaluate two extensions to pure write-invalidate and write-update called protocol read-broadcast and competitive snooping [19]. Eggers and Katz's competitive snooping protocol is an implementation of Karlin's Snoopy-Reading protocol [20]. Grahn *et al*. [22] proposes *competitive-update* and showed that *pure write-update* is highly undesirable in the general case because of the heavy traffic caused by the updates. Their work focus on setting a competitive-threshold value to the counter associated to each cache block in the local cache and the counter is decremented upon getting an update message from a remote processor. When the counter reaches to zero the copy of the cache block is invalidated and further updates to that cache block are ceased. There are significant differences between Grahn *et al*.'s work and our proposed work. In our work we assign the *reuse frequency* of each cache block in the shared LLC as zero and increment the value upon each access to that cache block. In our work we have considered the *reuse frequency* as well as the *last touch time* of the cache blocks to take the decision dynamically whether the sharers should be updated or not. In [22], Archibald proposed an adaptive write-invalidate/write-update snoopy-cache protocol. In their work they invalidated all other copies of the block when a single processor has issued three consecutive writes to the same block without any intervening access by another processor. Some of these techniques [9] [11] also added states to the coherence protocol. Some of the protocols focused on table based predictors [3] [10] that predicted messages before their arrival and took necessary steps at proper time.

In the previous works several coherence protocols [9][12] are proposed but an intelligent and novel scheme for a directory based protocol has not been studied where the *reuse frequency* and *last touch time* of a cache block are used to speculatively update the sharers. The dynamic and speculative updates of the sharers have improved the performance of *RFU-Dir*. We implement *RFU-Dir* on the directory based protocol to enhance scalability and ensure low traffic overhead .We use simple counters and buffers for keeping track of the *reuse frequency* and *last touch time* for each cache block respectively.

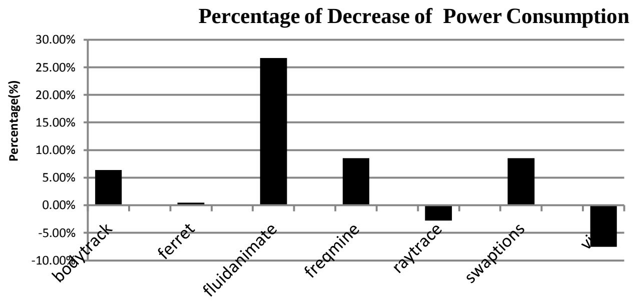


Figure 10. Decrease of power consumption in *RFU-Dir* (in percentage).

6 Conclusions

We proposed a speculative-based dynamic update mechanism to update the sharers dynamically in the conventional directory based MOESI coherence protocol. The proposed dynamic update mechanism is not limited to MOESI cache coherence protocol. It can also be applied to cache coherence protocol like MOSI *(Modified, Owner, Shared, Invalid)*, and MSI *(Modified, Shared, Invalid)*. Our dynamic and speculative update mechanism can be applied in any invalidation directory-based protocol. By using low overhead counters and buffers we keep track of the *reuse frequency* and the *last touch time* of the cache blocks respectively and instead of updating the sharers of all the cache blocks in the LLC we update the sharers of only those cache blocks which are highly reused in the *near-immediate* past.

We showed that simple counters and buffers can effectively identify a substantial portion of the sharers which may show temporal locality in the *near-immediate* future. Thus, updating those sharers result in improvement in the execution time, power consumption and hit rate. Our work also reduced the network latency, the L1 miss latency, the total number of L1 cache replacements and the directory accesses for searching the tag arrays. In the future we aim to further improve the coherence protocol in terms of performance, power, protocol complexity and scalability.

7 References

[1] M. K. Martin, M. D. Hill, D. A. Wood. "*Token Coherence: Decoupling Performance and Correctness*". In Proc. 30th Int'l Symp.on Computer Architecture, 2003.

[2] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy. "*The directory-based cache coherence protocol for the DASH multiprocessor*". In Proceedings of the 17th Annual International Symposium on Computer Architecture, pages 148–159,May 1990.

[3] M. K. Martin, P. J. Harper, D. J. Sorin, M. D. Hill, and D. A.Wood. "*Using Destination-Set Prediction to Improve the Latency/Bandwidth Tradeoff in Shared-Memory Multiprocessors*." Proc. 30th Int'l Symp.on Computer Architecture, 2003.

[4] M.K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A .R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood., "*Multifacet's General Execution-driven Multiprocessor Simulator (GEMS)Toolset*".Computer Architecture News (CAN) 2005.

[5] D. E. Culler, J. P. Singh, and A. Gupta. "*Parallel Computer Architecture: A Hardware/Software Approach*". Morgan 1998.

[6] NiketAgarwal, Tushar Krishna, Li-ShiuanPeh and Niraj K. Jha, *"GARNET: A Detailed On-Chip Network Model inside a Full-System Simulator"*. In Proceedings of IEEE International Symposium On Performance Analysis of Systems and Software (ISPASS), Boston, Massachusetts, April 2009.

[7] Andrew B. Kahng, Bin Li, Li-Shiuan Peh and Kambiz Samadi,"*ORION 2.0: A Power-Area Simulatorfor Interconnection Networks"*, to appear In IEEE Transactions on Very Large Scale Integration(TVLSI).

[8] C. Bienia, S. Kumar, J. P. Singh, and K. Li. "*The PARSEC Benchmark Suite: Characterization and Architectural Implications"*.In Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, October 2008.

[9] A. R. Lebeck and D. A. Wood. "*Dynamic self-invalidation: Reducing coherence overhead in shared-memory multiprocessors*". Proc. 22nd Int'lSymp.on Computer Architecture, 1995.

[10] S. S. Mukherjee and M. D. Hill. "*Using Prediction to Accelerate Coherence Protocols.*" Proc. 25th Int'l Symp. on Computer Architecture 1998.

[11] L. Cheng, J. Carter, and D. Dai. "*An adaptive cache coherence protocol optimized for producer-consumer sharing*." In International Symposiumon High-Performance Computer Architecture, 2007.

[12] A. Lai and B. Falsafi. "*Selective, accurate, and timely self-Invalidationusing last-touch prediction*." In Proc.6th Int'l Symp.onHighPerformanceComputer Architecture, 2000.

[13] John Shen and Mikko Lipasti. *"Processor Design: Fundamentals of Superscalar Processors"* First Edition, mcgraw-Hill 2005

[14] IBM Cell Architecure Project.http*://www.research.ibm.com/cell/*

[15] Tilera's Tile64 Multi-core processor – *http://www.tilera.com/Pdf/ProBrief_Tile64_Web.pdf*

[16] K.Li and P.Hudak *"Memory Coherence in Shared Virtual Memory Systems",* ACM Transactions on Computer Systems, 1989.

[17] Changkyu Kim, Doug Burger, Stephen W. *Keckler "An Adaptive Non uniform cache structure for wire-delay dominated on-chip Caches"* In Proceeding of the International Conference Architectural support for Programming Languages and OperatingSystems-2002.

[18] S.J. Eggers and R.H. Katz*,"A characterization of sharing in Parallel programs and its application to coherency protocol Evaluation",* Proc.15th Internat.Symp.on Computer Architecture Honolulu, (May1988) 373-382.

[19] S.J. Eggers and R.H. Katz, "*Evaluating the performance of four Snooping cache coherency protocols"*, Proc. 16th Inter.Symp.on Computer Architecture, Jerusalem, Israel (May 1989) 2-15.

[20] A. R. Karlin, M.S. Manasse, L. Rudolph and D.D Sleator, "*Competitive snoopy caching"*, Proc. 27th Annual Symp on Foundations of Computer Science (Oct. 1986) 244-254.

[21] J.K. Archibald, "A *Cache Coherence approach for large Multi Processor systems"*, Proc. International. Conf. on Supercomputing St. Malo, France (Jul. 1988) 337-345.

[22] H. Grahn, P. Stenström, M. Dubois *"Implementation and Evaluation of Update-Based Cache Protocols Relaxed Memory Consistency Models"* (1995) Future Generation SystemsVolume11.

Miss-rate Based Dynamic Phase Tracking on Multicore Processors

Sourav Dutta
Department of Electrical and
Computer Engineering
Southern Illinois University
Carbondale, Illinois 62901–6899
Email: sourav@siu.edu

Nazeih M. Botros
Department of Electrical and
Computer Engineering
Southern Illinois University
Carbondale, Illinois 62901–6899
Email: botrosn@siu.edu
Corresponding Author

***Abstract*—In this paper we propose a method to detect phases of an application with similar L2 cache miss behavior. Our proposed algorithm can predict per phase cache miss rate and able to detect stable phase transitions dynamically. In order to predict the L2 cache miss behavior of an application for different phases, We exploit the fractal behavior of cache and propose a miss rate adaption technique which allows to detect phase transitions. In this approach we use performance counters already available in modern processors. Adaptive behavior of the presented work makes it useful for real time applications. We performed different applications from modern benchmark to evaluate our proposed method. We use sniper multi-core simulator to model a multi-core out-of-order processor resembling the modern Intel Nehalem processor. Experimental results show that the proposed method has about 95% accuracy with minimum 2.1% prediction error and significantly low overhead.**

I. Introduction

Designers always attempt to optimize micro-architectural parameters such as cache size , window size etc to achieve the average performance needs of general microprocessors which are desgined to provide overall good performance while executing a large variety of workloads. However this overall optimization approach can cause poor performance and power dissipation during specific execution phases of certain programs.In recent days architects have proposed re-configurable hardware that can adapt dynamically to improve power, performance criteria [1],[5].

In a reconfigurable environment it is necessary to know the proper timing to initiate reconfiguration. Prior work shows that phase boundaries are relatively good choice to perform reconfiguration [6] ,[7]. Hence accurate detection of stable phase transition is an important property of dynamically reconfigurable systems. Moreover hardware assisted phase detection techniques generally incurs high overhead . Several researchers proposed reuse of configuration information of repeating phases to reduce overhead by using various phase classification techniques [4], [5], [8].

In early works [4], researchers proposed various methods to track working sets to model memory demands of a given application effectively. One common approach is to create an LRU stack and imitate the LRU replacement policy over collected memory traces[9]. This approach estimates miss rate curve by counting the number of hits to each stack location and predict the behavior of page misses against memory allocation. The run-time generation of miss rate curve using the mentioned approach is not negligible due to the increasing data size along with complexity of modern applications.

In this paper we propose an adaptive low overhead method to detect phases based on their memory demands. Our proposed on-line method can predict last level cache(L2) miss behavior per phase on modern commodity processors. In comparison to main memory and disks , there are several challenges associate to cache memory optimization[9]. Reducing main memory misses are more advantageous than reducing cache misses.In comparison to the cost of miss event , tracking cost of cache misses and based on that estimating cache miss rate is considerably higher. Its challenging to reduce the cost of tracking cache misses than miss penalty. On the other hand processor gets sufficient amount of time to track misses and perform calculations before the data from the disk appears when a misses in main memory occurs. Usual overhead of on-line phase tracking is high since it needs external hardware support. The primary challenge is to distinguish phase changes from random fluctuations.

Our proposed low overhead on-line method can estimate phase based cache miss rate which can help the designer to profile per phase cache requirement of the running application on a modern commodity processor. We exploit existing performance monitoring units of modern commercial processors to collect memory traces of the running application. Our algorithm can process these information on-line to support optimizations at different system levels such as virtual machine , operating system and various run-time programming environments.

We propose a fractal based miss rate estimation approach for on-line phase detection. Our presented algorithm doesn't need any external hardware. Maximum likelihood method has been used over a small number of sampled cache misses and cache access data pair to find cache miss rate for the detected phase.

In order to detect stable phase transition we propose a counter based approach which will help to differentiate phase transition from random fluctuations by collecting number

of consecutive deviations of instantaneous miss rate from predicted miss rate.
The remaining part of the paper is organized as follows. Background works and motivation are discussed in Section 2. Section 3 presents the methodology of the proposed on-line phase detection technique followed by experimental results and performance evolution in Section 4. Finally, Section 5 concludes the paper.

II. Background and motivation

A. Miss rate prediction

Multilevel caches are important feature of modern processor systems [11],[12] and this trend will continue in future due to its significance in speeding up processors. A lot of work has been done to predict cache miss behavior by creating an analytical model of cache[13],[18]. However ,only a few papers studied the theoretical approach to understand the source of cache miss behavior on cache sizes. Based on empirical observations Chow [13],[14] anticipate that the cache miss rate is governed by a power law function of cache's capacity. Based on this assumption he proposed an analytical model of cache to calculate the optimum cache hierarchy which can maximize the performance and reduce the cost. Przybylski et al.[15],[16] proposed a method to optimize the cache size at various levels by observing the cache miss behavior with respect to cache capacity . It is certain that if the size of a small workload fits the cache then the miss rate will be considerably small in comparison to large workload. In case of a large workload it is observed that the miss rate decrease as a power law of the cache size.
Hartstein et al.[18] formulates an analytical model of cache access behavior to establish the previously mentioned power law of cache miss rate theoretically. They observed the cache re-reference pattern with respect to time and combined that with a statistical model of cache replacement algorithm to estimate the cache miss rate in terms of cache size. Their work reveals that the cache miss rate should vary with cache size as an inverse square root power law.
In this paper we employ the fractal cache model proposed by Thiebaut[10] in order to detect phases based on the miss rate. Gillis and Weiss[19] observed that the memory access of a program can be seen as random walks. They established an asymptotic relationship between the range of a random walk $R(n)$ and the number of steps n. The predicted value of $R(n)$ is as follows :

$$R(n) \rightarrow A.n^{1/\theta} \;\; for \;\; 1 < \theta < 2 \tag{1}$$

In the above equation A and θ are constants. Theta , the inverse of the slope of the asymptote, is the fractal dimension of the walk. Thibaut employs the observation made by Gillis and Weiss to test the pattern of the accumulated number of misses with respect to the number of cache references. If the range of the random walk tends asymptotically towards a straight line , the walk is a fractal. Thibaut consider the range of the random walk $R(n)$ is equivalent to the number of misses since both of them represent the number of unique cells visited and propose the following model to describe the asymptotic nature of the accumulated misses.

$$Numberofmisses = An^{1/\theta} \;\; for \;\; n >> 1 \tag{2}$$

Where n is the number of cache references. The behavior of the accumulated number of misses can be described into two parts. Considering all cache references are misses (cold cache misses) at the starting stage of the program , the miss behavior can be described as

$$y = n \tag{3}$$

For the rest of the program , the cache miss behavior is governed by the fractal model described above and can be written as

$$y = A.n^{1/\theta} \tag{4}$$

Since the cold cache period is a small fraction of the full simulation time , the fractal model can successfully describe the overall cache miss behavior of a program. Thiebaut include cache capacity into the fractal model to describe practical cache miss behavior of a program and proposed the following model.

$$C = A.n_c^{1/\theta} \tag{5}$$

Here C denotes the total number of cache lines. Nc denotes the number of cache references which brings a finite-size cache into its full capacity. n_c can be computed as

$$n_c = (C/A)^{\theta} \tag{6}$$

Instantaneous miss rate MR can be estimated as the gradient of the curve at the point n_c. Miss can be obtained by deriving eq (6) with respect to the cache capacity

$$MR = \frac{A^{\theta}.C^{1-\theta}}{\theta} \tag{7}$$

It is important to estimate A and θ on-line to make eq (7) relevant for real time applications.Since cache shows fractal behavior, the number of misses in the cache of capacity C follows the straight line of equation

$$y = MR.n + D \;\; for \;\; n \geq n_c, \tag{8}$$

This conforms with Laha, Patel, and Iyer's observation [20] that the steady-state number of misses varies linearly with the number of accesses to the cache. We exploit eq (8) to keep track of the detected phase based on the previously calculated miss rate.
We employ maximum likelihood method to estimate these two parameters based on few samples of number of cache references and accumulated number of misses.Maximum likelihood estimation (MLE) is a powerful method developed by famous statistician R.A Fisher in early 1920s . The principal idea is that the underlies probability distribution of the collected or observed data makes it predictable using the process of statistical inference. The resulting parameter vector maximize the likelihood function as well as minimize the error between

predicted and observed data. In this paper we define the likelihood function as follows

$$y(n|\mathbf{x}) = A.n^{(1/\theta)} \tag{9}$$

Where the parameter vector **x** consists of A and θ. We employ the following chi-square error to measure the goodness of the predicted number of cache misses.

$$\chi^2 = \sum_i \left(\frac{y_i - y(n_i|\mathbf{x})}{\sigma_i} \right)^2 \tag{10}$$

III. Methodology

A. Memory traces per interval

In order to implement our methodology for detecting phases , we need to record cache access and cache misses in a fixed interval of time. In our case study environment we collect number of load and store events associated to L2 cache and from that information we calculate the number of L2 cache access as a summation of these two events. We take the sum of L2 load misses and store misses to collect the number of L2 cache misses.

B. Interval method for phase discovery

Pioneer work has been done by Allen and Cocke [21] to convert program control flow into a hierarchy of regions. For scientific programs, most computation and data access are in loop nests. Previous studies reveal that phase based analysis accurately summarize the overall program behavior.
In this work, we divide program execution into fixed-length sampling intervals (measured in terms of time quanta). Information related to Memory access is collected over the fixed sampling interval which is necessary to compute miss rate on-line. The miss rate collected over the previous interval compared to the calculated miss rate of the present phase. A phase change is indicated if both of them differ by more than a threshold (th). In this way we locate phases and classify them with their miss rate through on-line memory access behavior analysis.

C. On-line phase detection

Most scientific workloads exhibit typical memory demands for different phases. Within a phase, the miss rate remains nearly constant[9]. This inspired us to compute miss rate based on few memory access data to predict the miss behavior when the monitored program enters a phase and re-calculate it when a new phase is encountered. Through this approach, the overhead can be substantially lowered since we need to compute only A and θ in the eq (7) using the previously mentioned maximum likelihood method. However, in order to predict miss rate of different phases an adaptive mechanism is needed which can sense the phase transition behavior and set new miss rate to predict the miss behavior of the new phase.The key challenge is to differentiate phase changes from random fluctuations.
We propose an effective and simple algorithm to detect changes for memory access behavior by predicting cache miss rate. First, we sampled accumulated number of cache access and accumulated number of misses. Let A^i_{acc} and M^i_{acc} denote the sampled value of accumulated cache access and number of accumulated misses during ith time interval respectively. We can pick K sampled value of accumulated miss by $(Miss^i + Miss^{i-1} + ... + Miss^{i-k+1})$ and similarly accumulated cache access can be calculated by $(access^i + access^{i+1} + ... + access^{i-k+1})$.
After K data pair have been sampled we can calculate A^i and θ^i by using the maximum likelihood method over $(A^i_{acc}, M^i_{acc}), (A^{i-1}_{acc}, M^{i-1}_{acc}), ..., (A^{i-k+1}_{acc}, M^{i-k+1}_{acc})$ and then Miss rate can be calculated as $MR^i = f(A^i, \theta^i)$.Once MR^i is calculated, let (A^j_{acc}, M^j_{acc}) and $(A^{j+1}_{acc}, M^{j+1}_{acc})$ be the current sampled value and let $MR = MR(X)|X \in (j-k, j)$. We can calculate instantaneous miss rate $\triangle MR = (M^{j+1}_{acc} - M^j_{acc})/(A^{j+1}_{acc} - A^j_{acc})$. We define $err_a = | \triangle MR - MR|$. Where err_a is the absolute error. If $err_a \in [1-T, 1+T]$, where T a small threshold of choice. We assume the input signal is in stable phase. Otherwise , we assume that a new phase is encountered. In this case all the K sampled data is cleared so the MR based on all those data will not be used.

D. Miss rate adaption

To be consistent on phase detection and corresponding miss rate prediction , our method needs to detect phase transition and should correct the miss rate on-line. In order to do so we propose a method which can adjust A and θ dynamically and based on the new values it can predict the miss rate for the detected phase.
As we mentioned before the main challenge is to differentiate stable phase transitions from random fluctuations , to overcome this we propose a consecutive miss rate error counter C . After computation of miss-rate based on K sampled values , we calculate instantaneous miss rate $\triangle MR$. If absolute error err_a is more than the threshold value we increment the error counter. If C reached a maximum error count (100 in our evolution) then new miss rate need to be calculated based on new K samples. In that case we delete all previous K sampled values and start collecting new samples of accumulated number of cache access and corresponding accumulated number of misses till we collect K pair of data. Once we collect enough data , we recalculate A , θ and miss-rate based on the method mentioned in the previous section. However , at any time if the absolute error is less than or equal to the threshold value , the counter has to be reset. Basically the error counter counts the number of consecutive deviation of instantaneous miss rate from the predicted miss rate when the deviation is beyond a certain threshold value.

IV. Experimenal Results and analysis

In this section, we will evaluate the proposed phased detection based on L2 cache miss rate prediction methodology . We first plot the collected number of cache misses against corresponding number of cache access. Then we plot the

Algorithm 1 $MLE(M_{acc}[K], A_{acc}[K], C)$

```
Input:$M_{acc}[K], A_{acc}[K], C$
Output : $MR$
$\hat{M}_{acc}[K] \leftarrow A.A_{acc}[K]^{1/\theta}$
$\chi^2 \leftarrow \sum \left(\frac{\hat{M}_{acc}[K] - M_{acc}[K]}{\sigma}\right)^2$
Search $A\&\theta$ which minimize $\chi^2$
$MR \leftarrow \frac{A^{\theta}(1-C)^{1-\theta}}{\theta}$
Return
```

Algorithm 2 On-line Miss Rate based phase tracking

```
Initialize:$K \leftarrow 0$ ; $Count \leftarrow 0$; $M_{new} \leftarrow 0$ ; $A_{new} \leftarrow 0$;
for all $ith$ interval do
  $M_i \leftarrow$ Miss at $ith$ interval
  $A_i \leftarrow$ Cache Access at $ith$ interval
  $M_{old} \leftarrow M_{new}$
  $A_{old} \leftarrow A_{new}$
  $M_{new} \leftarrow M_{new} + M_i$
  $A_{new} \leftarrow A_{new} + A_i$

  if $K < SampleNum$ then
    $M_{acc}[K] \leftarrow M_{new}$
    $A_{acc}[K] \leftarrow A_{new}$
    $K \leftarrow K + 1$
    if $K == SampleNum$ then
      $MR \leftarrow MLE(M_{acc}[K], A_{acc}[K], C)$
    end if
  else
    $\triangle M \leftarrow M_{new} - M_{old}$
    $\triangle A \leftarrow A_{new} - A_{old}$
    $\triangle MR \leftarrow \triangle M / \triangle A$
    if $|MR - \triangle MR| > \epsilon$ then
      $Count \leftarrow Count + 1$
      if $Count > \tau$ then
        $Count \leftarrow 0$
        $K \leftarrow 0$
      end if
    end if
  end if
end for
```

TABLE I: Simuated system specifications

Component	Parameters
Processor	2 sockets , 4 cores per socket
Core	2.66 GHz, 4-way issue , 128 entry ROB
L1-I	32 KB, 4 way, 4 cycle access time
L1-D	32 KB, 8 way, 4 cycle access time
L2 cache	256 KB per core, 8 way, 8 cycle
L3 cache	8 MB per 4 cores, 16 way, 30 cycle
Main memory	65 ns access time, 8 GB/s per socket

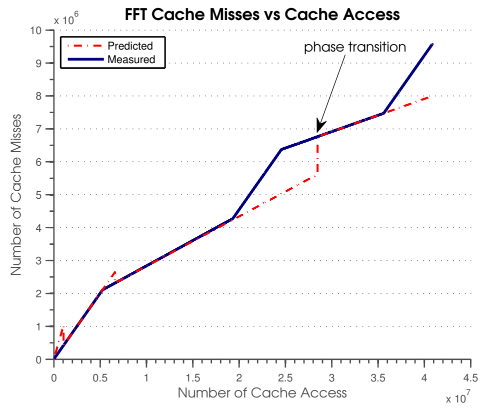


Fig. 1: Number of L2 Cache Misses vs Number of L2 Cache Access for FFT

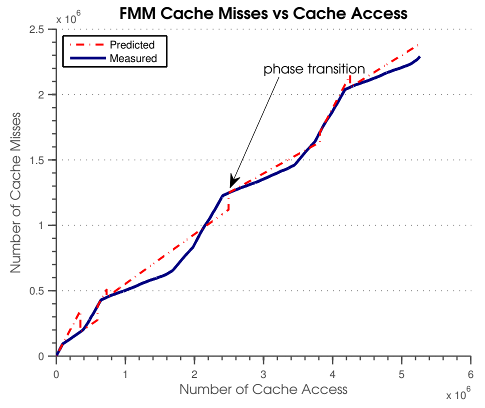


Fig. 2: Number of L2 Cache Misses vs Number of L2 Cache Access for FMM

predicted value of cache misses calculated while running the applications. The results validate our idea of tracking phase based on the miss rate of individual phases.Next, we review the prediction accuracy as well as run-time overhead of the proposed method.

For the results shown in this paper we used the Sniper multi-core simulation infrastructure [22] , which has been validated against real hardware. We configured it to model a multi-core out-of-order processor resembling the Intel Nehalem processor (Table 1) . Our workload consists of 8 different applications from the benchmark *SPLASH2* (*fft* , *fmm* , *barnes,ocean* , *raytrace* , *radiosity* , *water-nsquared* , *water-spatial*). All these applications represents a broad cache usage characteristics. We simulated the full applications without fast forwarding in order to detect various phases.

A. Accuracy and Runtime Overhead

In this section we first evaluate the prediction accuracy of our proposed method by comparing with the measured values. Then we evaluate runtime overhead of our method.
Figure 1 illustrates the measured L2 cache misses and the predicted number of L2 cache misses based on the presented method. It shows the cache access behavior of 8 different applications taken from *SPLASH2* benchmark. The measured values of cache access and cache misses are obtained through the counters available in the performance monitoring unit

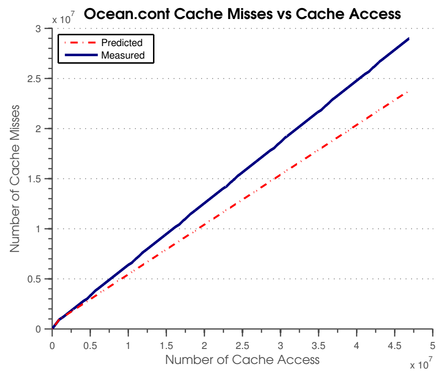


Fig. 3: Number of L2 Cache Misses vs Number of L2 Cache Access for Ocean(Continuous)

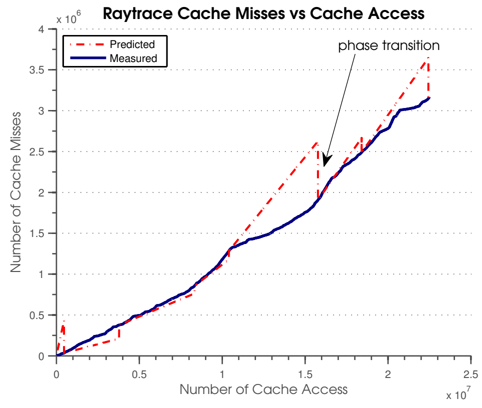


Fig. 4: Number of L2 Cache Misses vs Number of L2 Cache Access for Raytrace.

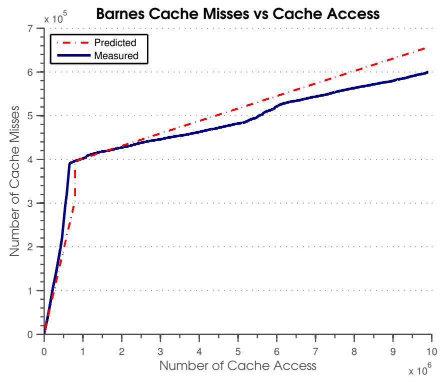


Fig. 5: Number of L2 Cache Misses vs Number of L2 Cache Access Barnes.

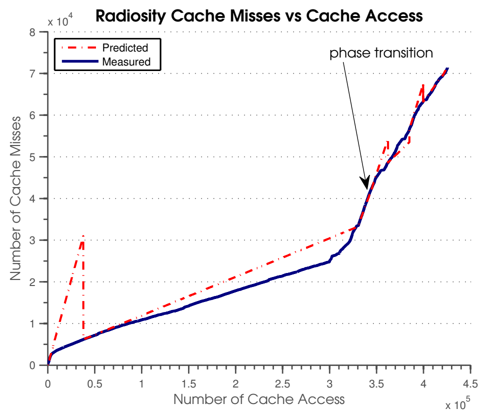


Fig. 6: Number of L2 Cache Misses vs Number of L2 Cache Access for Radiosity.

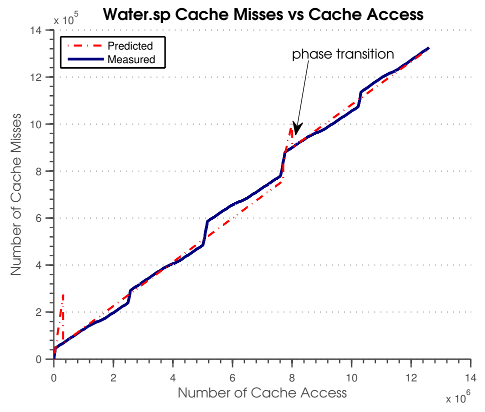


Fig. 7: Number of L2 Cache Misses vs Number of L2 Cache Access for Water(sp).

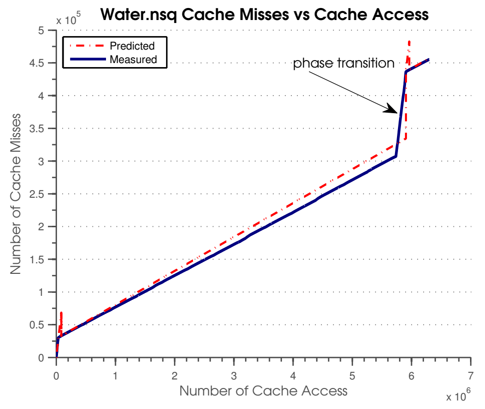


Fig. 8: Number of L2 Cache Misses vs Number of L2 Cache Access for Water(nsq).

while the applications are running. For each application we calculate the miss rate dynamically in-order to keep track of the running phase. We set 10 milliseconds interval size to collect data.

Form Figure 1 we can compare the measured value and the on-line predicted value of L2 cache misses for a given number of L2 cache access. The fit of our prediction is fairly good and it can successfully track program phases based on their memory access behavior. We use formal relative error method to calculate the prediction error of the proposed approach. Relative error is calculated for each interval and then prediction error is measured by taking the average on it. Table 2 demonstrates the prediction errors for different work-

TABLE II: Statistics

Workload	Overhead($\times 10^3$cycles)	prediction error(%)
FFT	522	2.48
FMM	187	3.18
Ocean.cont	81	11.40
Raytrace	103	9.81
Barnes	173	3.8
Radiosity	2760	4
Water Spatial	1399	2.9
Water nsquared	1787	2.1

loads. The error varies from 2.1% to 11.4% . Most of them are less than 4% and average error is 4.95% . Prediction error for two applications are comparably more. Error of *Ocean* with *contiguous partition allocation* (*Ocean.cont*) and *Raytrace* are 11.4% and 9.81% respectively. Experiment shows that miss-rate for *Barnes* and *Ocean.cont* remain same through out the full simulation. We can conclude that these two applications have constant memory access behavior through the full simulation.

The runtime overhead comes from the computation time of miss-rate based on the collected number of cache misses and cache access. Table 2 exhibits the miss-rate computation time for different workloads . Average computation time is 876 thousand cycles. *Radiosity* , *Water Special* (*Water.sp*) and *Water nsquared* (*Water.nsq*) have higher miss-rate calculation time than other workloads. The proposed method has lower runtime overhead than hardware directed phase tracking approaches since it needs to calculate only A and θ in order to find miss-rate. Usage of maximum likelihood method makes it simple to find the optimal values of A and θ based on the collected sampled values of number of cache misses and cache access.

V. Conclusion

We have successfully classified different phases of a complex workload based on their L2 cache miss-rate behavior on modern commodity processors by employing the proposed fractal behavior of cache. The predicted number of cache misses is close enough to the measured value which is collected while running the application. Accuracy of our method validate the fractal behavior of L2 cache of modern multi-core processors. We have implemented our method using performance monitoring unit commonly available in almost all modern processors. The low overhead of the proposed approach makes it useful for real-time applications. We achieved about 95% accuracy when compared to the measured value of cache misses. In addition to accuracy and low run-time overhead , the practical advantage of our proposed method is simplicity, since it needs to calculate only two parameters A and θ. The proposed method is cost effective since it does not require external hardware. Our proposed method would be beneficial for dynamic cache management, runtime scheduling and profiling techniques for power aware high performance computation.

References

[1] D. H. Albonesi, Dynamic IPC/clock rate optimization, in Proc. of the 25th Annual Intl. Sym. on Computer Architec- ture, pp. 282-292,Jun. 1998.

[2] M. Huang, J. Reneau, S.-M. Yoo, and J. Torrellas, A framework for dynamic energy efficiency and temperature management, in Proc. of the 33rd Annual Intl. Sym. on Mi- croarchitecture, pp. 202-213,Dec. 2000.

[3] J. E. Smith, and A. S. Dhodapkar, Dynamic microarchitec- ture adaptation via co-designed virtual machines, in 2002 Intl. Solid State Circuits Conference, Digest of Technical Papers, pp. 198-199, Feb. 2002.

[4] A. S. Dhodapkar, and J. E. Smith, Managing multi- configuration hardware via dynamic working set analysis, in Proc. of the 29th Annual Intl. Sym. on Computer Architec- ture, pp. 233-244,May 2002.

[5] M. Huang, J. Renau, and J. Torrellas, Positional adaptation of processors: application to energy reduction, in Proc. of the 30th Annual Intl. Sym. on Computer Architecture, pp. 157-168,Jun. 2003.

[6] Timothy Sherwood, and Brad Calder, Time varying behav- ior of programs, UC San Diego Technical Report UCSD- CS99-630, Aug. 1999.

[7] J. Henning, SPEC CPU2000 memory footprint, online at http://www.spec.org/ cpu2000/analysis/memory

[8] T. Sherwood, S. Sair, and B. Calder, Phase tracking and prediction, in Proc. of the 30th Annual Intl. Sym. on Com- puter Architecture, pp. 336-347, Jun. 2003.

[9] D. K. Tam, R. Azimi, L. B. Soares, and M. Stumm. "Rapidmrc: Approximating l2 miss rate curves on commodity systems for online optimizations". ASPLOS 09 , pages 121132, 2009.

[10] Thiebaut,D. On the fractal dimension of computer programs and its application to the prediction of the cache miss ratio. IEEE Transactions on Computers, July 1989,1012-1026.

[11] R. Kalla, B.Sinharoy, and J. Tendler. IBM Power5 Chip: A dual-core multi-threaded processor, IEEE Micro, vol. 24(2), pp. 40-47, 2004.

[12] http://www.sun.com/processors/manuals/USIV_v1.0.pdf. Ultra-SPARC IV Processor,User's Manual Supplement, Version 1.0, 2004.

[13] C. K. Chow. On Optimization of Storage Hierarchies, IBM Journal of R&D, vol. 18, pp.194 - 203, 1974.

[14] C. K. Chow. Determination of Cache's Capacity and its Matching Storage Hierarchy, IEEE Transactions on Computers, vol. c-25, pp. 157 - 164, 1976.

[15] S. Przybylski, M. Horowitz and J. Hennessy. Performance Tradeoffs in Cache Design, Proceedings of the 15th Annual International Symposium on Computer Architecture, pp. 290 - 298, 1988.

[16] S. Przybylski, M. Horowitz and J. Hennessy. Characteristics of Performance-Optimal Multi-Level Cache Hie rarchies, Proceedings of the 16th Annual International Symposium on Computer Architecture, pp. 114 - 121, 1989.

[17] M. H. Macdougall. Instruction-level Program and Processor Modeling, Computer, vol. 7, pp. 14 - 24, 1984.

[18] A. Hartstein, V. Srinivasan, T. Puzak, and P. Emma. On the Nature of Cache Miss Behavior: Is It $\sqrt{2}$? In The Journal of Instruction-Level Parallelism, volume 10, 2008.

[19] J. E. Gillis and G. H. Weiss, Expected number of distinct sites visited by a random walk with an infinite variance, Journal of Mathamatical Physics, Vol.11, No.4, pp.1307-1312, April. 1970.

[20] S. Laha, J. H. Patel, and R. K. Iyer, Accurate low-cost methods for performance evaluation of cache memory systems, Tech. Rep., Coordinated Sci. Lab., Univ. of Illinois at Urbana-Champaign, Urbana, IL 61801, pp. 1-26, 1986; IEEE Trans. Comput., to be published.

[21] F. Allen and J. Cocke. "A proram data flow analysis procedure". Communications of the ACM, 19:137147, 1976.

[22] Carlson, T. E.; Heirman, W.; Eeckhout, L. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simu- lations. International Conference for High Performance Computing, Networking, Storage and Analysis (SC), Nov. 2011.

Petri Net Based Algorithm Modelization and Parallel Execution on Symmetric Multiprocessors

Gustavo Wolfmann
Laboratorio de Computación
Fac. Cs. Exactas Físicas y Naturales
Universidad Nacional de Córdoba
Av. Vélez Sársfield 1611 - Córdoba - Argentina
gwolfmann@efn.uncor.edu

Armando De Giusti
III LIDI
Fac. Informática
Universidad Nacional de La Plata
50 y 120 - La Plata - Argentina
degiusti@lidi.info.unlp.edu.ar

Abstract—**PDPTA 2014 - The Symmetric Multiprocessors architecture is composed by a complex set of cores, chips and memory channels that make it difficult to implement a parallel program that efficiently uses all resources. Another obstacle for achieving a performance according the resources is added by algorithms with hard data dependency. Asynchronicity is a key to get all processors running. Petri Nets have been used for a long time to model algorithms, but not as a tool to parallel execution. In this paper we introduce an asynchronous Parallel Execution Model based on Petri Nets and the process to go from a high level model to an executable parallel program. The Cholesky Factorization algorithm is used as a testbed. Tests results yield values that are near the theoretical peak and open good prospects to expand the model to other environments and algorithms.**

Keywords*—*Petri Net Modelization - Symmetric Multiprocessor - Parallel Execution Model - Cholesky Factorization Algorithm.

I. INTRODUCTION

Tiled algorithms emerge as a solution to the problem of load balance for dense linear algebra algorithms on multicore processors [1]. This type of algorithms divides data in square blocks that are used in subsequent stages of processing until the final result is reached. This strategy of data division allows increasing the number of tasks that can be run in parallel when there is no data dependency among data tiles.

To improve processing performance, the parallel algorithm has to drive two key concepts: granularity and asynchronicity. Granularity plays an important role in parallel performance because the larger the number of blocks, the more tasks that can be run in parallel. Furthermore, without asynchronicity, performance decreases due the existence of blocking points in the algorithm that causes processors to become idle until they all reach each point.

The combination of a large number of tasks with asynchronicity generates an execution problem: the selection of the task to be launched and the processor that execute it. Hundreds or even thousands of tasks running in a machine with a large number of parallel processors will result in an overload problem.

In a previous work, [2] we have shown that Petri Net is a good tool to model and control the execution of a complex parallel algorithm. A high-level modelization based on Coloured Petri Nets (CPN) [3] has been presented. Transitions represent the tasks of the algorithm and Places represent the data parameters used by each task. For example, the Cholesky Factorization algorithm can be resolved with four BLAS/LAPACK [4], [5] routines; thus, the CPN that model that algorithm uses only four transitions with one, two or three input places according the data blocks used as parameters. No additional transitions nor places are needed.

The Coloured Petri Net model is good to understand and analyze the parallel algorithm, but it is too complex to be the basis for parallel execution. Unfolding a CPN to a Token Petri Net (TPN) [6] allows working with an equivalent but simpler net. This unfolding defines a net with the exact number of tasks the algorithm must execute. In addition, as Petri Nets are good to model a parallel process, the resulting unfolded net allows analyzing parallel execution restrictions of the algorithm.

On the other hand, there is a tendency to increase the number of cores in a Symmetric Multiprocessor machine (SMP). This feature is given by assembling multiple CPU chips into the motherboard, normally with two or four slots to add up to sixteen cores in each one. In order to maintain data locality, two problems arise with this hardware architecture: efficient memory management and process affinity [7].

The multiplicity of processors in the SMP machine results in difficulties with cache memory. Thus, the higher level of cache memory is shared by one or more cores, based on chip design. To avoid cache misses, a parallel process that uses a block of data, should not have more threads than the number of cores that share the higher cache memory.

Not only the number of threads must be present to minimize cache misses, but also the position of the thread in the pool of cores, namely, core affinity must be taken into account. If the threads of a parallel process are distributed in different chips of the CPU, cache consistence for all threads will have to copy data between chips, reducing performance.

Both facts, the number and placement of threads, have an impact on parallel algorithm design. In advance, a double core division is recommended to have all processors working with an acceptable level of cache faults. A first level consists in a series of logical processors, whose quantity must match the number of higher cache memory partitions. The second level divides each logical processor into as many threads as physical cores shares the higher level of cache memory. For example, if the higher cache level is shared by four cores and the machine has 32 cores, it should be divided into eight parallel logical processors with four contiguous cores each.

Therefore, an efficient SMP machine utilization must not only use a data block size that minimizes cache misses, but also the number of cores that share one block of higher cache memory must be considered. If we add algorithm structure, the complexity to get an efficient parallel execution is high.

In this paper we present the design and execution results of a parallel version of Cholesky factorization algorithm, modeled with Petri Nets and executed on two different SMPs. This work is the continuation of the one cited above, using the same algorithm modelization. In this paper, an execution model that takes into account the hardware variables of an SMP machine is introduced. The rest of the paper is organized as follows: the next section presents a brief summary of the Petri Net model developed before. Section three introduces the execution model. Results are discussed in Section four and finally, conclusions and future research are presented.

II. The Petri Net Model

In a previous work [2], the model used to analyze and simulate runnings of a parallel algorithm was introduced. It is based on Coloured Petri Nets. This high level model is then unfolded into a Simple Place / Transition net (TPN), which is used to run the parallel algorithm. A summary is presented.

Figure 1 shows the CPN that represents Cholesky's algorithm. It has only four Transitions and eight Places; each Transition represents one routine and each Place represents one of its parameters. The name of the places follows the number of the block used in each operation. Color tokens are represented by $< x{,}y >$, multiset repetitions by braces $\{x\}$, and functions arcs are only Booleans of the form $if(cond)$.

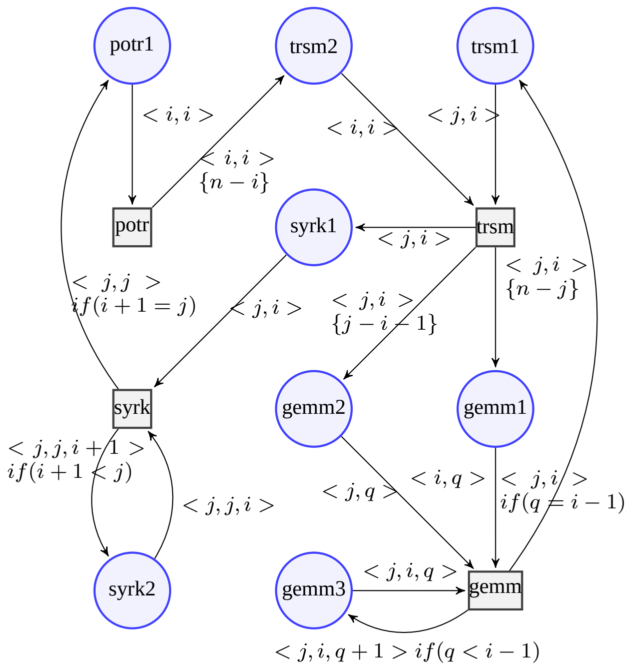


Fig. 1. Coloured Petri Net that represents Cholesky's factorization algorithm.

Place in CPN	Domain in CPN
potr1 trsm2	$< i,i >, i = 1 \dots n$
trsm1 syrk1 gemm1	$< j,i > j = 2 \dots n, i = 1 \dots j-1, j > i$
gemm2	$< j,i >, j = 3 \dots n, i = 1 \dots j-2, j > i$
syrk2	$< j,j,i >, j = 2 \dots n \wedge i = 1 \dots j-1 \wedge j > i$
gemm3	$< j,i,q >, j = 3 \dots n, i = 2 \dots n-1, q = 1 \dots i-1 \wedge j > i \wedge i > q$

Fig. 2. Domains of the Places for the Coloured Petri Net in Fig.1 .

Arcs domains are shown in the table of Fig. 2. The algorithm is generically defined by the CPN, and the number of tiles in which the matrix is divided is provided as a parameter in the domain definition. The high level of expressivity of a simple model can be remarked.

In contrast, the overhead required to represent CPN domains and function arcs in an executable way is expensive in terms of high performance computing, and it is impractical to use it directly. Nevertheless, the CPN developed like this, meets the definition of well-formed CPNs [6]. This type of nets is easily transformed into a TPN, which has a simpler computational implementation and is light to execute.

The unfolding of a CPN is defined in Diaz et.al. [6]. Each Place P_j in a CPN has an associated Domain $D(P_j)$; thus, its unfolding produces as many Places in the TPN as the cardinality of $D(P_j)$ in the colored Place, preserving the repetitions of the multiset. Hence, each Place in the TPN has an association with a unique value from the pairs (color, place) in CPN and only one token can live on it.

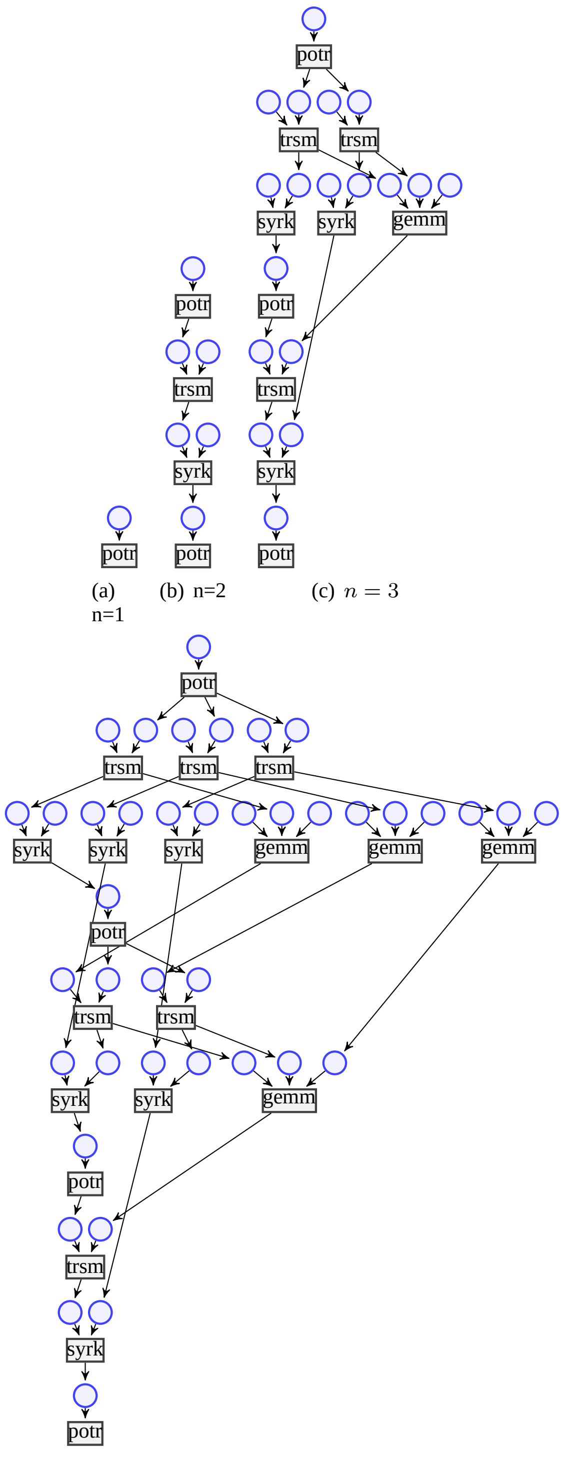
Transitions are unfolded by generating as many Transitions in TPN as the cardinality of the Cartesian Product of all the elements of its domain in the CPN. The cardinality of the multiset in each Place must be preserved. Hence, each Transition in TPN is associated with a unique combination from the Cartesian Product, preserving repetitions of the multisets of each input Place. Only guards with true values produce results. By construction, each unfolded Transition represents an individual event that will be associated with a single task.

Figure 3 shows four examples of unfoldings from CPN to TPN, for $n = 1, 2, 3, 4$ square tiles divisions[1]. Places are shown in the same order they have in CPN. Their names are not shown due to space limitations. The order in which each unfolding is shown is not random: each unfolding of n divisions has the same graphic as $n-1$ divisions, adding to the top, the tasks due to larger number of tiles. In this way, the Transitions in Fig. 3 at the same horizontal level represent the same task in all figures, ordered from end to start.

The chart in Fig. 3 shows two important aspects of algorithm parallel execution. First, there is a critical path of tasks execution derived of data dependency, that can not be exceeded [8]. Each increment in the number of tiles generates a sequence of *potr*, *trsm* and *syrk* tasks that must be done serially.

[1]A tile division of n represents $n \times n$ square blocks of data

Secondly, in the hypothetical case of having an unlimited number of parallel processors with the same execution time for all tasks, the minimum time required for the parallel execution is a function of the number of tile divisions, and clearly, there is an upper limit of the number of processors that can run in parallel.



(a) n=1 (b) n=2 (c) $n = 3$

(d) $n = 4$

Fig. 3. Token Petri Net unfolded from the Coloured Petri Net in Fig.1 using different numbers of square tiles divisions (n).

op $\backslash n$	1	2	3	4	5	6	8	10	12	15	20
potr	1	2	3	4	5	6	8	10	12	15	20
syrk	0	1	3	6	10	15	28	45	66	105	190
trsm	0	1	3	6	10	15	28	45	66	105	190
gemm	0	0	1	4	10	20	56	120	220	455	1140
total	1	4	10	20	35	56	120	220	364	680	1540
seq. tasks	1	4	7	10	13	16	19	22	25	28	31

Fig. 4. Number of each task according the tile division.

Figure 3 shows that, when the number of tile divisions is small, the idleness of all parallel processors is high, due the strong data dependency. The table in Fig. 4 shows the number of algorithm tasks based on the number of tile divisions. The *potr* task has a linear growth, *syrk* and *trsm* have quadratic growth, and *gemm*, cubic growth. As stated above, each stage of processing introduces a group of three serial tasks, except for the first one has only one. This implies that, for example, a tile division of five has 13 sequential tasks over the total of 35.

The parallelism of the algorithm with few tile divisions is poor. Due the cubic growth of the *gemm* task, more divisions generate more tasks and a better capacity for parallel execution. By contrast, a tile division of 10 results in a large number of parallel tasks, and an increment of the overload cost. A tile division with low number of divisions in a SMP machine with few cores, will be enough to make an acceptable use of its computational power, but, in machines with 32 or more cores, a bigger number of divisions and a complex management tool of tasks to exploit their capacity will be needed.

In the next section, the parallel execution model is described. It was developed to handle the combined complexity of the algorithm and the SMP architecture, and it will be used as a basis for the execution.

III. The Execution Model

The previous section shows how to model the algorithm with Coloured Petri Nets (CPN). Unfolding the CPN to a simple Token/Place Petri Net (TPN) transform a compact net into a bigger but simpler one to execute. This section shows how to execute a parallel algorithm based on TPN.

The Parallel Execution Model (PEM) is defined as a tuple:

$$PEM = (P, T, I^-, I^+, M, \Pi, \chi) \qquad (1)$$

where:

- P is a finite set of Places P_i, with cardinality $|P| = p$, $i = 1 \ldots p$.
- T is a finite set of Transitions T_j, with cardinality $|T| = t$, $j = 1 \ldots t$.
- I^- and I^+ are the negative and positive incidence matrixes of the TPN, with dimension $p \times t$ (I^- and $I^+ \in \mathbb{N}^{p \times t}$).
- M, is the Mark Vector for Places, $p \times 1$ ($M \in \mathbb{N}^p$).
- M_f, is the Final Mark Vector, $p \times 1$ ($M_f \in \mathbb{N}^p$).
- Π is a finite set of Processors Π_i, with cardinality $|\Pi| = \pi, i = 1 \ldots \pi$. Each processor Π_i has a boolean

variable e ($\Pi_i.e$), which is set as either true or false to indicate if it is running or if it is idle.

- χ is a Boolean variable that implements a mutual exclusion mechanism over M that allows each Π_i to update M securely.

The initial state is:

- $M = M_0$, the initial mark of the TPN.
- $\chi = true$, the exclusion is free.
- $\Pi_i.e = true$, $i = 1 \dots \pi$, because all processors are idle.

The PEM is very close to Timed Petri Nets [9]. Both share the concept that firing a Transition is not instantaneous because there is a time elapsed between the start and the end of the firing. The same as in PEM, the firing action represents the execution of a task, but the difference is that in PEM firing is not done autonomously once the transition is enabled as it is in Timed Petri Nets. An idle Processor is responsible to fire the Transition selected among all the enabled ones.

The number of enabled Transitions can be lower or higher than the number of Processors. As a result of this, we can have idle Processors with no Transitions to fire or enabled Transitions waiting for a free Processor, depending on the number of enabled Transitions in relation to the number of idle Processors. In the first case, the execution speedup of the will be poor and this situation must be avoided. In the second case, the Processor must select the most appropriate Transition to fire. To do this, we have adopted the general criteria of selection based on the priority of the Transitions that are in the “Critical Path” to finish the algorithm.

The implementation of this execution model needs one Mutual Exclusion (mutex) mechanism to avoid concurrent reading and writing operations over vector M, which is the one that defines the algorithm state. In this sense, the Processors act serially when selecting the next Transition to fire, waiting for the mutex enabled.

The Pseudo-code of the PEM execution algorithm is presented in Fig. 5. In round-robin format, each logical processor with the enabled flag set on, searches for a task to execute based on the Petri Net modelization of the algorithm. To determine which Transitions are enabled, only simple linear algebra operations are needed. In effect, if we call I_j^- and I_j^+ the j-th column (transition) in I^- and I^+ respectively, the j-transition is enabled if the vectorial subtraction $M - I_j^-$ does not have any negative value. This is defined by function h, which has arity $h : \mathbb{N}^{p \times t}, \mathbb{N}^{p \times 1} \to \{0, 1\}$, with parameters M and I^-, and their result values are:

$$h(j) = \begin{cases} 0 & \text{if } (M - I_j^-) \text{ has negative/s value/s} \\ 1 & \text{if } (M - I_j^-) \text{ else} \end{cases} \qquad j = 1 \dots t$$

Computing h for all the columns determines all the enabled transitions ready to be fired at one point of the execution. By design, each Place of the unfolded TPN is the input Place of only one Transition. This guarantees no competition between enabled transitions for input tokens, and that all enabled transitions can be fired simultaneously.

```
 1 While main algorithm not finished
 2   If can hold the mutual exclusion
 3      Compute h function
 4      Select one task to execute
 5      Update M by absorbing tokens
 6      Free the exclusion
 7      Task execution
 8      Inject tokens in M
 9   Else
10      Delay
11   Endif
12 End
```

Fig. 5. Pseudo-code of the task selection algorithm.

To determine the task to be executed, a dynamic scheduler was developed. It uses a valuation function that is applied to the set of enabled Transitions, selecting the transition with highest valuation, T_k. The valuation function is the key for the parallel processing performance, because when the set of enabled Transitions has more than one element, it must select the one that will enable more Transitions in the future, namely, it keeps the larger number of parallel tasks enabled. This scheduler is related to Quark [10], which prioritizes data locality instead of availability of parallel tasks.

Additionally, there is a mapping between each Transition and each task to be executed, and also a mapping between each Place and the data block which is used as parameter in the task. To execute a task, the Processor determines which task and which parameters are needed from the T_k selected and then runs it.

Steps 5 and 8 of the pseudo-code algorithm represents the evolution of the execution. Similar to Timed Petri Nets, tokens are absorbed and injected at two times. In step 5 the tokens from the input Places of T_k are absorbed, and in step 8, they are injected to their output Places. Both steps are made with simple linear algebra operations:

$$M' = M - I_k^- \qquad \text{in 5 at } t_0 \qquad (2a)$$
$$M''' = M'' + I_k^+ \qquad \text{in 8 at } t_0 + \Delta_k \qquad (2b)$$

and after step 8, potentially new Transitions become enabled. M and M'' are the markings at time t_0 and $t_0 + \Delta_k$, where Δ_k is the elapsed time of the T_k task execution. The cycle is repeated until the end of the algorithm is reached, which occurs when $M = M_f$.

The overhead introduced by the parallel execution is defined by three factors. First, the mutual exclusion mechanism, which uses few cycles of clock. Second, matrix and vector operations, which are highly optimized to run in milliseconds with current processors. Third, the selection policy must be guided by a balancing among selection load and overall algorithm performance. In fact, the sum of the time of three factors is several orders of magnitude smaller than the routine execution, which means a minimum overhead.

IV. Experiments

A FORTRAN program was developed to read, interpret and execute the model. OpenMP was used as shared memory model of execution and tests were run over two machines, the first with four AMD 6344 processors, which conform a 48 cores machine, and a second, with two Intel Xeon E5-2680 chips. Single precision floating point was used for all tests.

There are two requirements of the PEM that limit the possible stack of compiler / libraries to be used for coding: nested parallelism and core affinity. Both requirements are the basis of the PEM and are features that must be present jointly in the compiler. This fact left only one possibility for each machine: gfortran with ACML for the AMD-based machine, and Ifortran with MKL for the Intel-based one.

The configuration of the parallel hardware may vary in number of cores and chips, so the implementation of the PEM must be configurable to adapt to the hardware used. Thus, an XML-style file that contains the settings of the real machine in which the program runs is used. It contains the first-and second-level processor division, the type of physical processors and the mapping between Places with data and Transitions with tasks. In the software, this feature acts as an intermediate layer between hardware and algorithm and make tunning easier.

A. The AMD-based machine

The architecture of the AMD-based is as follows: four dies with twelve cores each. Each die has two blocks of L3 cache memory associated to a block of six cores. For floating point operations, the die has one Fused Multiplication Addition unit (FMA) shared by two cores. Each FMA unit can perform concurrently one addition and one multiplication of 256 bits, improving the processing power of the cores that share it. Since there is an ACML version that is optimized for these FMA units, it was used but restricting the number of processors to the model in the PEM model to 24.

The logical hardware division used in the tests was $n{\times}1$ processors, where n is the number of first level processors, i.e., the second level of divisions has only one core. This decision was made based on two factors: the first is that the research focus is over task synchronization, thus, the higher the number of processors to synchronize, the better the test to our objectives. Second, the implementation of ACML for the routines used, has a poor speedup. In effect, the parallel implementation that uses the FMA units, when using several threads (six,eight,etc), does not scale properly for the routines *ssyrk*, *strsm* and *spotrf*. In consequence, the serial version of ACML was used, running each routine in one FMA unit. Nevertheless, the affinity concept remains important, because, as the FMA unit is shared by two cores, the logical processor division needs to take two consecutives cores to make exclusive use of one FMA.

Several test results are shown in Table 6. The body presents time and flops of various combinations of matrix range, number of parameters and tile divisions.

The analysis of the results brings some conclusions:

- The low number of data divisions has poor results: it is the effect of poor parallelism when the number of tile divisions is fewer than six.

procs		8		16		24	
range	dvs	secs	flops	secs	flops	secs	flops
12000	8	3.14	183	2.89	199	3.12	185
	12	2.91	198	2.21	260	2.38	242
	15	4.05	142	4.07	141	4.42	130
24000	8	22.11	208	18.98	243	20.33	227
	12	19.28	239	13.49	342	14.98	308
	15	19.21	240	12.06	382	13.73	336
30000	8	43.05	209	36.99	243	40.90	220
	12	36.02	250	25.11	358	25.74	350
	15	35.50	254	23.11	389	23.42	384

Fig. 6. Time in seconds and flops in GFlops from tests with matrix ranges of 12000, 24000 and 36000; 8, 16 and 24 processors, and data divided in 8, 12 and 15 tiles with the AMD-based machine.

- A tile division of 15 generates 680 tasks with 1800 parameters, which is the range of the corresponding incidence matrix. This results in a heavy overload for the matrix operations when updating the Marking Vector M. However, this tile division produces a large number of parallel tasks. A balance must be achieved between the number of parallel tasks and data tile range in order to keep the processing/overload ratio convenient for throughput.
- The best result is for range of 30000, 16 processors and 15 tiles, which brings 389 Gflops. Since AMD-based machine has a theoretical processing peak of 998 Gflops[2], it represents a processing utilization that is close to 40% of its peak. Better yet, if we only consider the sixteen processors used, the effective processing ratio increases to 58%. These are very good values considering they are negatively affected by cache misses, the serial part of the algorithm and overload.
- Using the 24-processor configuration has no speedup improvements versus using the 16-processor one. This is due to a problem in physical memory configuration because the bank one is the only one used in this machine, so the memory channel becomes saturated when all the 24 processors are running.

B. The Intel-based machine

The Intel-based machine used has two Xeon E5-2680 chips, each of them with eight cores. Thus, the operating system has a total of sixteen threads available.

Each of the cores of the Intel processor has one AVX unit (Advanced Vector Extensions) that uses 256 bits registers. It can perform addition and multiplication operations simultaneously over these registers. This feature determines a theoretical processing power per core similar to that of the FMA unit available in the AMD-machine.

In the Intel-based machine, tests were executed using Intel Composer 2013 suite, which includes the MKL BLAS/LAPACK implementation. In Fig. 7, a summary of the most representative results are shown. As with the AMD-based machine, the best performance is reached when tile division is twelve.

The analysis of the results brings some conclusions:

[2]998 Gflops = 2.6 Ghz x 24 fma units x 2 ops x 8 single precision values

procs		8x1		16x1	
range	dvs	secs	flops	secs	flops
24000	8	17.08	270	13.25	348
	12	14.35	321	9.52	484
48000	8	140.59	262	101.73	342
	12	109.24	337	70.99	519

Fig. 7. Time in seconds and flops in GFlops from tests with matrix ranges of 24000 and 48000, 8 and 16 processors, and data divided into 8 and 12 tiles, with the Intel-based machine.

- Similar to the results obtained with the AMD-based machine, the impact of having more tile divisions is strong: in all cases, division by 12 tiles increases performance goes up to 30%.
- Going from 8 to 16 logical processors does not scale properly due to the effect of memory channel saturation. However, performance goes up to 50%.
- Processing a big matrix with 16 logical processors and 12 tile divisions yields a result of 519 Gflop. Considering the processing power available with sixteen cores, the rate of use of the physical processors gets near 75% of the theoretical peak, which evidences a very good management of cache misses and synchronizations.

A final test was done by fixing the number of tile divisions and modifying core divisions. The table in Fig. 8 shows time and flops for a range of 24000 and 48000, 12 tile divisions and all the remaining combinations for sixteen cores into two levels. This test was done on the AMD-based machine, but the scalability was so poor that the results were useless. This is the opposite with the Intel-based machine. They show a similar performance regardless of how the cores are divided. The best performance is achieved with a division of four logical processors with four cores each, which brings 616 gflops. Considering that the theoretical peak of the machine is 691.2 gflops, a near-optimum result was obtained.

procs		1x16		2x8		4x4		8x2	
range	dvs	secs	flps	secs	flps	secs	flps	secs	flps
24000	12	11.69	392	10.17	453	8.98	513	8.40	549
48000	12	69.73	529	62.72	588	59.81	616	60.33	611

Fig. 8. Time in seconds and flops in GFlops for tests with matrix ranges 24000 and 48000, using 16 threads with double level division, virtual processors x internal threads (1x16,2x8,4x4,8x2), and data divided into 12 tiles, with the Intel-based machine.

Finally, the Fig. 9 shows a timeline for the execution of one of the tests, with a range of 24000, a division of 12 tiles and 16 processors. Processor idleness can be observed both at the initial and final stages of the execution, as mentioned above. Also, there is no idle time while processing and the low impact of the overload is evident from the absence of idle areas around the tasks.

V. Conclusions

The research has several points to highlight:

- It has been shown that Petri Net not only has good properties to model concurrent systems, but that it is also a good basis for a parallel execution model. It is not easy to manage the parallel execution of hundreds or even thousands of tasks, but we found how to do it with this tool.
- The execution tool developed can be adopted to any SMP machine and optimized libraries thanks to the combination of TPN with the virtual processor definition, two levels of hardware division and processor affinity.
- The parallel execution environment developed was able to reach a real utilization of the processors very close to its theoretical limit. Asynchronicity and affinity were the key to this achievement.
- Modeling an algorithm with CPN allows analyzing its parallel capabilities and brings information about its possibilities and limitations in the search for better parallel performance. In particular, we conclude that applying a tiled division of data to the Cholesky algorithm does not result in a good performance if tile division is less than six.
- The XML-style of the model's parameter file allows not only adapting it to different machines, but it also leaves open the capability to switch the algorithm by only changing the incidence matrix and task mapping. Thus, any parallel algorithm designed following a well formed CPN can be executed with a high level of performance by only changing the incidence matrix and tuning its virtual processors.

Future work will focus on researching the advantages and obstacles when using the technique developed with various algorithms and also using heterogeneous systems, such as hybrid CPU / GPU systems. An implementation in a distributed memory architecture will also be studied.

References

[1] A. Buttari, J. Langou, J. Kurzak, and J. J. Dongarra, “A class of parallel tiled linear algebra algorithms for multicore architectures,” LAPACK Working Note, Tech. Rep. 191, Sep. 2007.

[2] G. Wolfmann and A. De Giusti, “Parallel asynchronous modelization and execution of cholesky algorithm using petri nets,” in *The 2013 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA13)*, 2013.

[3] K. Jensen and L. M. Kristensen, *Coloured Petri Nets - Modelling and Validation of Concurrent Systems*. Springer, 2009.

[4] “Basic Linear Algebra Subprograms Technical Forum Standard,” University of Tennessee, Tech. Rep., 2001. [Online]. Available: http://www.netlib.org/blas/

[5] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. J. Dongarra, J. Du Croz, S. Hammarling, A. Greenbaum, A. McKenney, and D. Sorensen, *LAPACK Users' guide (third ed.)*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1999.

[6] M. Diaz, *Petri Nets: Fundamental Models, Verification and Applications*. London, Hoboken: ISTE Ltd - John Wiley & Sons, Inc., 2009.

[7] M. McCool, J. Reinders, and A. Robison, *Structured Parallel Programming: Patterns for Efficient Computation*. Elsevier Science, 2012.

[8] S. Tomov, J. Dongarra, and M. Baboulin, “Towards dense linear algebra for hybrid gpu accelerated manycore systems,” *Parallel Computing*, vol. 36, no. 5-6, pp. 232–240, 2010.

[9] J. Wang, *Timed Petri Nets: Theory and Application*, ser. The International Series on Discrete Event Dynamic Systems. Springer US, 1998.

[10] A. YarKhan, J. Kurzak, and J. Dongarra, “Quark users' guide: Queueing and runtime for kernels,” Innovative Computing Laboratory, University of Tennessee, Tech. Rep., 2011.

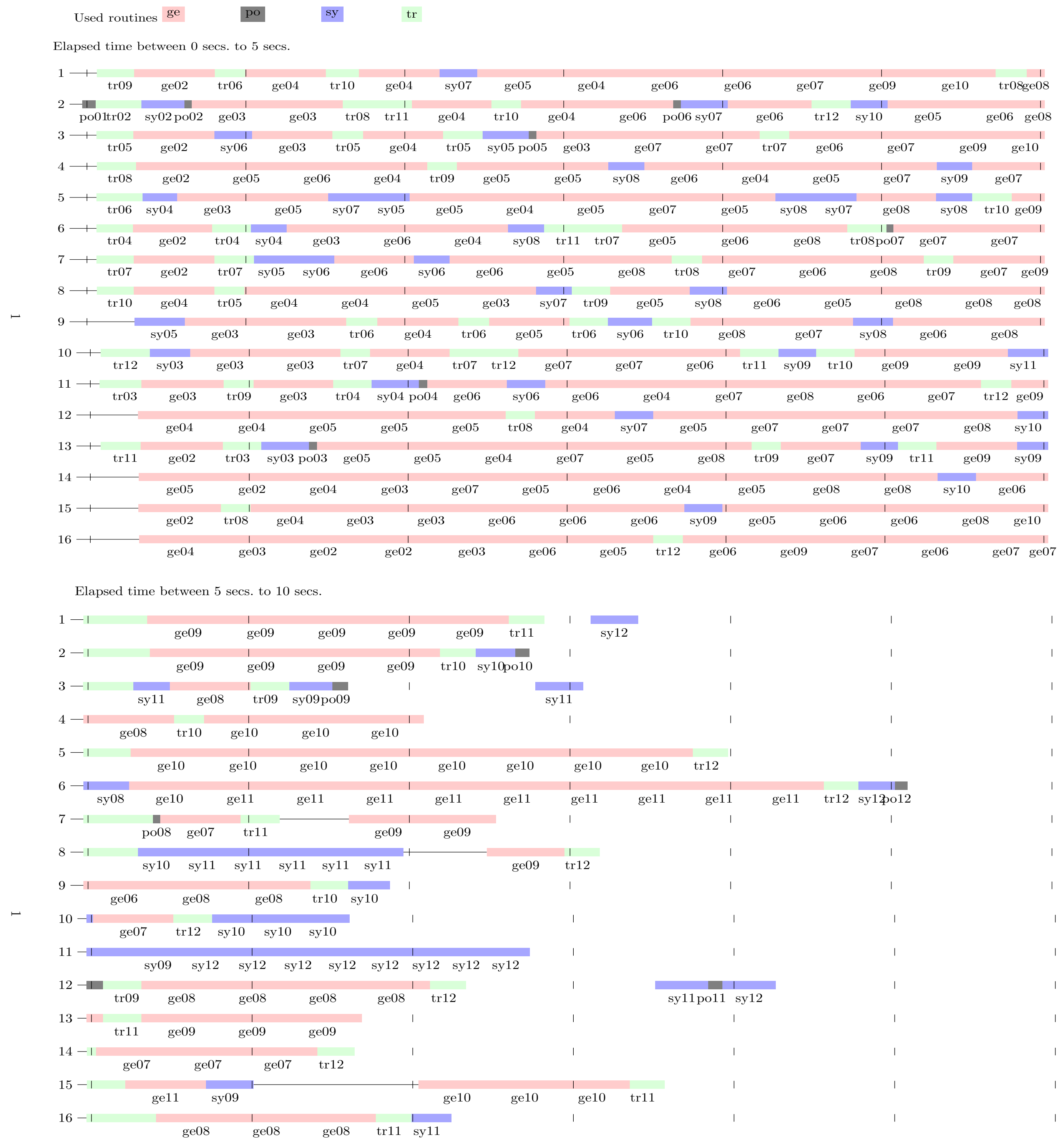


Fig. 9. Execution timeline, divided in two sections, Intel-based machine, 24000 range, 12 tiles, 16 processors

SESSION

HIGH-PERFORMANCE COMPUTING + DISTRIBUTED ALGORITHMS AND APPLICATIONS

Chair(s)

TBA

Real-Time Token-Based Mutual Exclusion Algorithms

Mitchell L. Neilsen
Department of Computing and Information Sciences
Kansas State University
Manhattan, KS, USA
{neilsen@ksu.edu}

Abstract

Many token-based, distributed mutual exclusion algorithms can be generalized by a single algorithm. The algorithm's performance is dependent upon the logical topology imposed on the nodes and the policy used to forward requests.

This paper extends the generalized algorithm to support prioritized, real-time requests in a generalized fashion and presents models that can be used to analyze the performance and verify the correctness of the generalized algorithm. Both safety and liveness properties are verified. Model checking is also used to analyze performance. Using the best topology, the generalized algorithm attains the same worst-case performance as a centralized algorithm; i.e., three messages per critical section. In the average case, the generalized algorithm performs better than a centralized one when the star topology is used. Finally, requests by nodes at each priority level are processed in order, resulting in bounded, predictable worst-case response times.

Keywords: distributed algorithm, model checking, mutual exclusion, real-time, token-based

1 Introduction

Many distributed mutual exclusion algorithms have been proposed [1, 4, 6, 8, 12, 13, 16, 17]. These algorithms can be classified into two groups [13]. The algorithms in the first group are called *permission-based* [1, 4, 6]. A node enters its critical section only after receiving permission from a quorum of nodes. The algorithms in the second group are called *token-based* [8, 12, 16, 17]. The possession of a system-wide unique token gives a node the right to enter its critical section.

Lamport proposed one of the first distributed mutual exclusion algorithms [4]. Lamport's algorithm is permission-based and requires $3 * (N - 1)$ messages to provide mutual exclusion. Another permission-based algorithm, proposed by Ricart and Agrawala, reduces the number of required messages to $2 * (N - 1)$ messages per critical section entry [14]. Maekawa proposed a permission-based algorithm in which the number of messages required is $O(\sqrt{N})$ [6].

Ricart and Agrawala proposed a token-based algorithm which is essentially the same as Suzuki and Kasami's algorithm [16]. The maximum number of messages required by these algorithms is N because request messages are sent to all other nodes, and the token is passed in a single message.

By imposing a tree-based logical structure on the nodes, another class of token-based algorithms has been obtained. All of the nodes, except for the root node, are on a path to the root node (a sink node) in the logical structure. The logical structure determines the path along which a request message travels. There are two different types of logical structures: *dynamic* and *static*.

An algorithm, based on a dynamic logical structure, was proposed by Trehel and Naimi [17]. The basic notion underlying this algorithm is path reversal. *Path reversal* at each node is performed as a request from node x travels along the path from node x to the root node. As the request travels, node x becomes the new parent of each node on the path, except for node x. Thus, node x becomes the new root node. A complete analysis of path reversal has been given by Ginat [3]. The average number of messages required per critical section is $O(log(N))$.

If a static logical structure is used, the basic notion underlying the algorithm is what we call edge reversal [11]. *Edge reversal* at each node is performed as the request from node x travels along the path from node x to the root node. At each node, the direction of each edge on the path is changed to point towards node x; that is, to the neighboring node who sent the request on behalf of node x. However, the shape of the logical structure never changes. Suprisingly, this small change results in algorithms which have a small fixed upper bound on the number of messages required per critical section, and the upper bound only depends on the logical structure. Algorithms based on edge-reversal were proposed by Neilsen and Mizuno [11] and Raymond [12]. Raymond's algorithm assumes that the static logical structure is an unrooted tree. If the radiating star topology is used, the average number of messages required is $O(logN)$. However, this is not optimal, using a simple star topology with one root node only requires 4 messages per critical section, two messages to pass the request, and two for the token.

Neilsen and Mizuno introduced a token-based algorithm that achieves optimal performance with respect to worst-case message complexity [11]; i.e., 3 messages per critical section. They also proposed another algorithm that generalizes all existing token-based algorithms that impose a logical structure on the nodes [9]. Both algorithms assume a fully-connected physical network and a directed acyclic graph (dag) structured logical network. A node or a token does not need to maintain a queue of outstanding requests for mutual exclusion. Instead, the queue is maintained implicitly in a distributed manner and may be deduced by observing the states of the nodes. The algorithms require very simple data structures; each node maintains a few simple variables, and the token carries no (or a very simple) data structure. Furthermore, the algorithms can adapt to changes in the network.

More recently, the notion of prioritized mutual exclusion algorithms have been proposed [5, 7, 10, 15]. Mueller's algorithm extends Trehel and Naimi's algorithm by incorporating a priority queue at each node. Our generalized algorithm can also be easily adapted to operate much like a priority-based scheduler which schedules tasks based on their priority, and schedules tasks at the same priority level using round robin scheduling [10]. The basic single-priority, generalized algorithm can be used to provide round robin scheduling. The other algorithms include dynamic priority adjustment to prevent starvation and ensure fairness, at the expense of losing real-time guarantees [5, 15].

To support prioritized, hard real-time requests, the generalized algorithm is easily extended to pass the token between different priority levels. Not only can the topology be generalized, but the algorithms used to forward requests and preserve real-time behaviour can be generalized as shown in this paper.

Section 2 introduces the generalized algorithm. Section 3 presents models that can be used to verify correctness with respect to guaranteed mutual exclusion, deadlock freedom, and real-time properties for the highest priority tasks. Section 4 analyzes the performance of the algorithm using these models. Finally, Section 5 summarizes the results.

2 Generalized Algorithm

We assume that the system consists of N nodes, which are uniquely numbered from 0 to $N-1$. At any time, each node can have at most one outstanding request to enter its critical section. Physically, the nodes are fully connected by a reliable network, but logically, the nodes at each priority level are organized in a directed acyclic graph (dag). Nodes in different levels are assigned different priorities.

Two types of messages, called REQUEST and TOKEN, are exchanged among nodes. When a node wants to obtain the token to enter its critical section, it sends a REQUEST message. A TOKEN message represents the token; when a node receives a TOKEN message, it may enter its critical section.

Each node maintains three simple variables: integer variables LAST and NEXT, and a boolean variable HOLDING or SINK. The logical directed acyclic graph (dag) structure indicates the path along which a REQUEST message travels and is imposed by the LAST variables in the nodes. When a node initiates or receives a REQUEST message, the node forwards the request to the neighboring node pointed at by its LAST variable (unless the node is a sink, in which case its LAST variable is -1).

The NEXT variable indicates the node which will be granted mutual exclusion after this node. If the node is currently the last node to be granted mutual exclusion, its NEXT variable is -1. Thus, by following the NEXT variables from the token holder to the node whose NEXT variable is -1, the implicit waiting queue of pending requests can be deduced. When a node leaves its critical section, it forwards the token to the node at the front of the waiting queue and also performs a *dequeue* operation. That is, it sends a TOKEN message to the node indicated by its NEXT variable and sets NEXT to -1 (this corresponds to the dequeue operation), unless NEXT is already -1. If NEXT is -1, the node continues to hold the token if it is at the highest priority level by setting HOLDING to true, otherwise the token is returned to the highest priority level as described below.

Semantically, a sink node in the system is (1) the last node in the implicit waiting queue (i.e., its NEXT variable is -1), and (2) the last node on the path along which a request travels within a given priority level (i.e., its LAST variable is -1). When a sink node receives a REQUEST message, it *enqueues* the request into the implicit waiting queue and becomes a non-sink. The node initiating the request becomes the new sink since it is now the last node in the queue. Each edge in the path must change direction to point in the direction of the new sink. This is done by the nodes along the path in a distributed manner as follows.

When a node initiates a new REQUEST message, it forwards the message to its neighbor indicated by its LAST variable and sets its LAST variable to -1 to become a new sink. It remains a sink until it receives a subsequent request.

When an intermediate (non-sink) node receives a REQUEST message from a neighboring node X, it passes the message to the neighboring node indicated by its LAST variable. Then, the node sets its LAST variable to *any* node on the path traveled by the REQUEST message. If a node receives a subsequent request, it forwards the request in the direction of the new sink. In Trehel and Naimi's algorithm, the LAST variable is set to the node that initiated the request; this is called *path reversal*. In Neilsen and Mizuno's algorithm, the LAST variable is set to point to the neighboring node from which it received the REQUEST message; this is called *edge reversal*.

When a sink node receives a REQUEST message, it sets its NEXT variable to the identifier of the node initiating the request. This corresponds to an *enqueue* operation. The node also sets its LAST variable to *any* node on the path traveled by the REQUEST message. Note that if a sink node holds the token, but is not in its critical section (indicated by a boolean variable HOLDING) when it receives a request, it immediately forwards the token to the node initiating the request.

```
const
    $I$ = node identifier
var
    HOLDING          : boolean;
    LAST, NEXT       : integer;

proc ProcessWork;
    begin
        if (not HOLDING) then
            begin
                send REQUEST($I$) to LAST;
                LAST := -1;
                wait until a TOKEN message is received;
            end;
        HOLDING := false;

                critical section (CS)

        if (NEXT $\neq$ -1) then
            begin
                send TOKEN message to NEXT;
                NEXT := -1;
            end;
        else HOLDING := true;
    end;

proc ProcessRequest; (receive REQUEST($X_1, \cdots, X_k$))
    begin
        if (LAST = -1) then
            begin
                if HOLDING then
                    begin
                        send TOKEN message to $X_1$;
                        HOLDING := false;
                    end;
                else NEXT := $X_1$;
            end;
        else send REQUEST($X_1, X_2, \cdots, X_k, I$)
            to LAST;
        LAST := $X_i$ for some $1 \leq i \leq k$;
    end;
```

Figure 1. Generalized algorithm

Because of message delay, there may be several sink nodes in the system while requests are in transit. The system is initialized so that only one node at the highest priority level possesses the token. Initially, there is only one sink node at each priority level, and its LAST variable is initialized to -1. In all other nodes, LAST is set to point to the neighbor which is on a path to a sink node. When a request reaches a sink node at a lower priority level, the request is forwarded as a proxy request to a node acting as the lower level's proxy at the highest priority level. Proxy requests are forwarded just like ordinary requests, but no edges or paths are reversed as the proxy requests travel to a sink node (or some node with a pending request).

The complete generalized algorithm for a single priority level is shown in Figure 1. There are two procedures at each high priority node: ProcessWork and ProcessRequest. Procedure ProcessWork is executed when a high priority node I requests for entry into its critical section, and procedure ProcessRequest is executed when a high priority node I receives a request from some other high priority node. In the algorithm, REQUEST messages are of form REQUEST($X_1, X_2, \cdots, X_k$) where $X_1, X_2, \cdots, X_k$ denotes the path on which the request traveled and X_1 denotes the node where the request originated. Each node executes procedures ProcessWork and ProcessRequest in local mutual exclusion. The only exception is that a node does not have to execute in local mutual exclusion while waiting for a TOKEN message to arrive or while in its critical section.

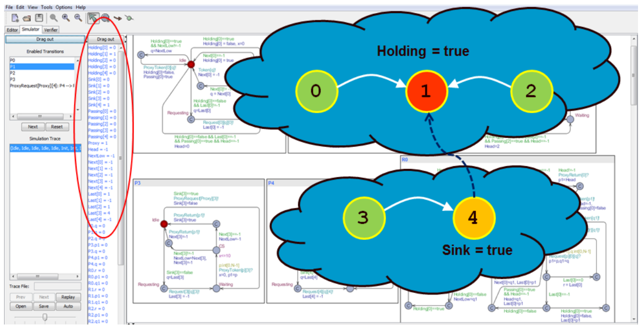


Figure 2. Two priority levels

The prioritized algorithm for low priorty nodes is similar, except that the boolean variable HOLDING is replaced with SINK as shown in Figure 2. Initially, a node in the highest priority level holds the token, and HOLDING for that node is set to true. Likewise, the root node at each lower priorty level has its SINK variable set to true. When a request from a low priority node reaches the sink node, a PROXY REQUEST is sent up to a node at the highest priority level, called the *proxy*, and eventually the PROXY REQUEST reaches a node that holds the token or will receive the token in the future. To further generalize the algorithm, requests or proxy requests from lower priority nodes only need to be passed to the point where they reach a node that is currently requesting the token. The key point is that all low priority requests are eventually enqueued in the token. Finally, if a proxy request reaches the highest priority level, then it is forwarded just like a regular request, except that the edges are not reversed and in the generalized case the request only needs to reach a node that is currently requesting the token. Note that higher priority nodes never request the token from lower priority nodes so there is no need to dynamically adjust the edges. When a node that is holding the token passes the token to a lower priority node, a PROXY TOKEN message is used to pass the token, and the token is returned back to the high priority node who sent the token using a PROXY RETURN message.

3 Verification Model

In this section we provide models that can be used to verify the correctness of the generalized algorithm with respect to guaranteed mutual exclusion, deadlock freedom, and starvation freedom using UPPAAL [2]. The model for a single priority level consists of two templates, **ProcessWork** and **ProcessRequest**, corresponding to the procedures shown above in Figure 1. Channels are used to model the exchange of request messages and the token. Global arrays are used to model the state at each node using the arrays `Holding`, `Sink`, `Next`, and `Last` as defined above. Initially, one node will hold the token, so `Holding[i] = true` at that node. Also, the topology is defined by the initial values assigned to `Last`.

The first UPPAAL template, **ProcessWork**, is shown below in Figure 3. It models the work performed at each node. Initially, all nodes are in the Idle state. The template is parameterized using `id` to identify each node where `id` $\in \{0, 1, \cdots, N-1\}$. At node 0, `id = 0`, etc. Also, `Holding[id]` is set to true at the node currently holding, but not using, the token. This node can enter its critical section immediately after setting `Holding[id]` to false to indicate that the token is in use.

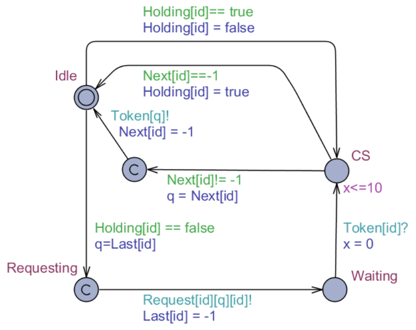


Figure 3. ProcessWork template

All other nodes must send a request message to the node identified by `LAST[id]`, and set `LAST[id] = -1`. Upon receipt of the token via a `Token` message, the requesting node may enter its critical section. A local clock, x, is used to prevent a node from remaining in it's critical section forever. The model limits critical sections to be at most 10 time units through the location invariant $x \leq 10$.

To process requests, the **ProcessRequest** template is used as shown in Figure 4. When a request message is received, at node `id` from node `p`, using `Request[p][id][s]?`, the source node requesting to enter its critical section is node `s`.

If the node receiving the request is a sink node, `Last[id] == -1`, the request can be satisfied immediately if the node receiving the request is holding, but not using, the token; that is, if `Holding[id] == true`. In this case, the `Token` message can be sent immediately. On the other hand, if the token is currently in use, then the request is simply enqueued by setting `Next[id] = t` which is a local meta variable assigned to `s` when the request is received.

If the node receiving the request is not a sink node, `Last[id]`$\geq$`0`, then the request is forwarded on to the node indicated by `Last[id]`.

In all cases, `Last[id]` is set to the identifier of the neighboring node which sent the request; that is, *edge reversal* is used in Figures 3 and 4. The generalized algorithm is slightly more complex because each node can set it's `Last[id]` value to be any node visited from the requesting node to the sink. Consequently, a list or queue of visited node numbers must be carried with the `Request` message. This is modeled with a set of global queues that get updated as the `Request` travels. One queue is assigned to each node and used to enqueue the nodes on the path from the given requesting node to a sink.

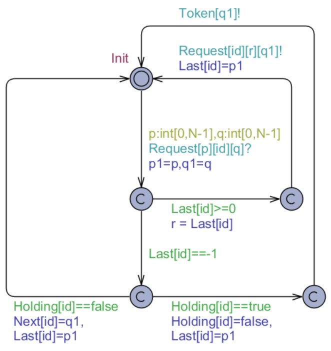


Figure 4. ProcessRequest template

The nodes can be initialized to impose any logical topology. For example, to impose the star topology, with node 0 as the root, simply set `Holding`$[$`i`$] = false$ and `Last`$[$`i`$] = 0$ for all `i` $\neq 0$, and `Holding`$[0] = true$ and `Last`$[0] = -1$.

The generalized **ProcessWork** template is shown in Figure 5. By making Init a committed state, all nodes will enter the Ready state before any nodes start requesting. The only other change required is to add a function, `initRequest(id)`, that is used to initialize the queue passed with the `Request` message to contain a single element `id`.

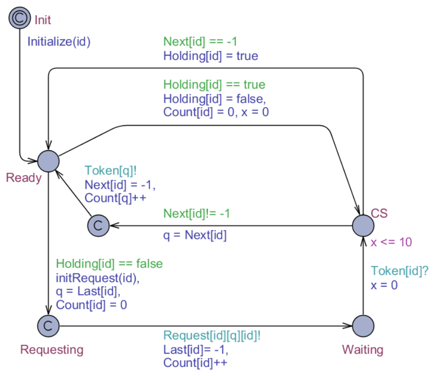


Figure 5. Generalized ProcessWork

The generalized **ProcessRequest** template is shown in Figure 6. As the `Request` message travels from a requesting node to a sink node, the identifier of each node receiving the request must be enqueued in the request message. Since UPPAAL messages have zero capacity, this is modeled using a set of global queues and a function call `enQueue(t,id)` to enqueue `id` on the queue for the requesting node `t`. Also, each node on the path sets `Last[id]` to be some element in the queue.

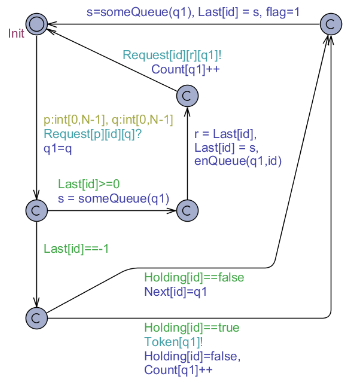


Figure 6. Generalized ProcessRequest

The function, `someQueue()`, relies on a random number generator, modeled by the RandomValue template shown below, to randomly select a random element from the queue carried with the request. Finally, to support different priority levels, we can add the notion of a proxy node at the highest priority level, and use algorithms similar to the above to request the token.

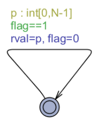


Figure 7. RandomValue

Due to space constraints, we only include the prioritized models for edge reversal (extending the templates shown in Figures 3 and 4), but the generalized case is similar.

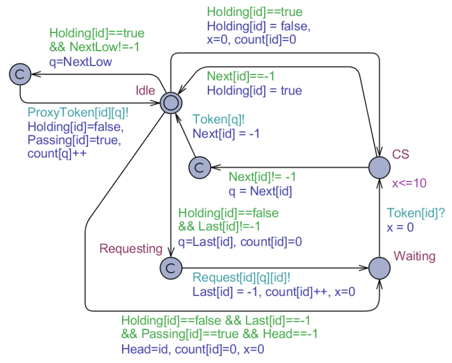


Figure 8. Prioritized ProcessWork

When no requests are pending at the highest priority level, and a request is pending at some lower priority level, the token is passed down to allow a node at a lower priority level to obtain the token using a `ProxyToken` message. The first pending request at each lower priority level is enqueued in priority order in the token. Of course, this may lead to starvation; if there is always a pending request at the highest priority level, then lower priority requests will not be satisfied

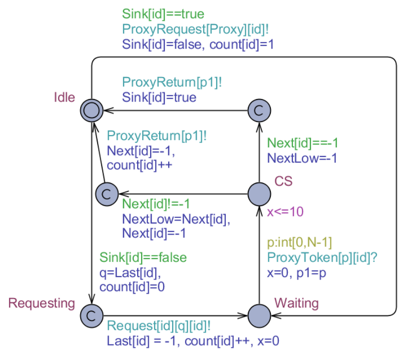


Figure 9. Prioritized ProcessWorkLow

For the prioritized case, four templates are used, nodes at the highest priority level use **ProcessWork** and **ProcessRequest**, and lower priority nodes use **ProcessWorkLow** and **ProcessRequestLow**, as shown in Figures 8-11.

4 Performance Analysis

To verify the correctness of the algorithm we have developed analytical proofs of correctness. We have also verified both safety and liveness properties using UPPAAL [2] for models such as those shown in Figure 2. To initialize the system for that model, abbreviate names by setting `P(i) = ProcessWork(i)` and `R(i) = ProcessRequest(i)` for `i=0,1,2`; `P(i) = ProcessWorkLow(i)` and `R(i) = ProcessRequestLow(i)` for `i=3,4`. It is easy to verify that the algorithm satisfies mutual exclusion. The property `A[](forall (i:node) forall (j:node)` `((i==j) or not (P(i).CS and P(j).CS)))` is satisfied, where `node` is defined as a new type `int[0,4]`; that is, only one process can be in its critical section (state `CS`) at any time. To verify liveness properties, we first limit the amount of time each process can remain in its critical section; otherwise, one node could stay in its critical section forever. The `CS` state has a location invariant of $x \leq 10$ for a real-valued clock x which is initialized to 0 upon entry to the critical section state. The choice of ten time units is arbitrary. To verify that a node that wants to enter its critical section can enter, we verify the property `P(i).Requesting --> P(i).CS`; that is, requesting the critical section "leads to" entry. Formally, a process, at the highest priority level, in the `Requesting` state eventually reaches the `CS` state.

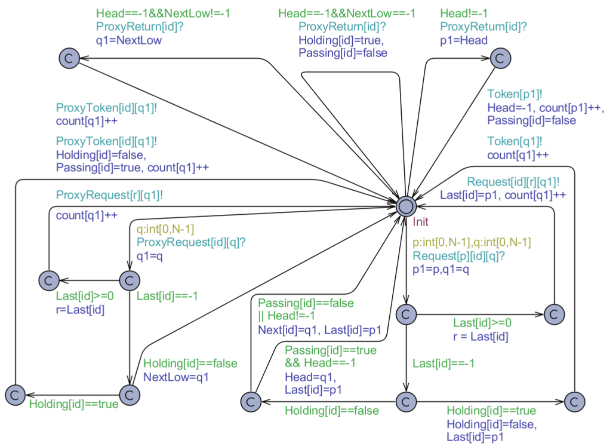


Figure 10. Prioritized ProcessRequest

To verify real-time behaviour, real-valued clocks can be used. For the example shown in Figure 2, each higher priority node only needs to wait at most 30 time units before entering its critical section. This can be verified using the models in Figures 8-11 using the query E<>(P(1).Waiting and P(1).x >= 30) which is satisfied, and the query E<>(P(1).Waiting and P(1).x > 30) which is not. Note that clock x is reset to 0 when entering the Waiting state. This case occurs when a low priority node is in its critical section when all three higher priority nodes request to enter. In general, starvation freedom is not satisfied by lower priority processes, so no such bound exists for low priority nodes unless there is sufficient delay between subsequent requests. Using simple worst-case response time analyses, it is easy to show that the worst-case response time before a given node is allowed to enter its critical section is found to be the worst-case blocking time caused by a lower priority node being in its critical section, followed by the maximum possible interference that results from all equal or higher priority nodes requesting at the same time. The models can be modified to include a minimum delay between subsequent requests. The results match those found using analytical models and simple response time analysis.

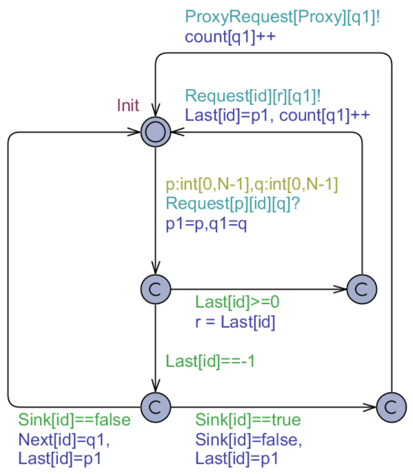


Figure 11. Prioritized ProcessRequestLow

The performance of the algorithm depends on the topology of the logical structure. The best topology with respect to message complexity is the star topology, with one node in the center and all other nodes as leaf nodes. For the analysis, we define the *diameter* D of a logical structure to be the length of the longest path in the structure. As the logical structure evolves, the value of D may also change.

With a single priority level, the *upper bound* is equal to $(D+1)$ messages per critical section entry: D messages for the request to travel to the sink node and one message for the token to be sent back to the requesting node. Thus, using the straight line topology, the upper bound is N, the number of nodes in the system. For the best topology, a star, the upper bound is 3, which is the same as a centralized mutual exclusion algorithm. To verify this upper bound, counters can be used, as shown above. Once a request is satisfied, the counter, count[id], is set back to zero. The model can be verified to determine the maximum number of messages required; e.g., E<>(count[2]==3) – on some path, eventually, does the count reach 3. For the star topology with *edge reversal*, each leaf node can require up to 3 messages per critical section; thus, the property E<>(count[2]==3) is satisfied if node 2 is a leaf node, and the property E<>(count[2]==4) is never satisfied. Likewise, the central node requires at most two messages per critical section, and lower priority node 3 requires at most 5 messages. Not suprisingly, for the worst topology – a straight line – the worst-case for path reversal and the generalized algorithm is N messages, $N-1$ `REQUEST` messages and one `TOKEN` message.

If there is more than one priority level, then the number of messages required at the highest priority level is only dependent on the number of nodes in the highest level with the same analysis as above. For lower priority messages, after the request reaches a sink node, by traversing at most the diameter of the topology used for the given priority level, a proxy request may need to be forwarded up to the highest priority level, and to a sink node – again, at most the diameter of the nodes at the highest priority level. The token is passed to the lower priority node with a single message, and returned with a single message. Thus, the worst-case message complexity is $D_H + D_L + 2$ for low priority requests, where D_H is the diameter of high priority nodes, and D_L is the diameter of low priority nodes. If the low priority level consists of a single node, then $D_L = 0$. For the star topology, and the *proxy* node set as the root, at most 3 messages are required per critical section for both low and high priority nodes.

5 Summary

This paper presented a generalized, prioritized, token-based algorithm for distributed mutual exclusion. In the generalized algorithm, requests from lower priority nodes can be processed in different ways – either by passing each low priority request up to a sink at the highest priority level or by only passing low priority requests up to some other requesting node.

The algorithm imposes very little storage overhead on each node and in each message. Furthermore, the algorithm generalizes several existing token-based algorithms and can be extended for prioritized, real-time systems. Using the best topology and edge reversal, the algorithm attains comparable performance to a centralized mutual exclusion algorithm; that is, three messages per critical section entry. In the average case, the algorithm attains the best performance of any known algorithm. Real-time performance can be determined using verification of the models and analytically using response time analysis.

References

[1] D. Agrawal and A. El Abbadi. An efficient solution to the distributed mutual exclusion problem. In *Proc. 8th ACM Symposium on Principles of Distributed Computing*, pages 193–200, 1989.

[2] G. Behrmann, A. David, K.G. Larsen, J. Håkansson, P. Pettersson, W. Yi, and M. Hendriks. UPPAAL 4.0. In *Proceedings of the 3rd International Conference on the Quantitative Evaluation of SysTems (QEST) 2006*, IEEE Computer Society, pages 125–126, 2006.

[3] D. Ginat, D.D. Sleator, and R.E. Tarjan. A tight amortized bound for path reversal. *Information Processing Letters*, 31:3–5, 1989.

[4] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[5] J. Lejeune, L. Arantes, J. Sopena, and P. Sens. A prioritized distributed mutual exclusion algorithm balancing priority inversions and response time. In *42nd IEEE Internaltional Conference on Parallel Processing (ICPP13)*, October 2013.

[6] M. Maekawa. A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems. *ACM Transactions on Computer Systems*, 3(2):145–159, 1985.

[7] F. Mueller. Prioritized token-based mutual exclusion for distributed systems. In *International Parallel and Processing Symposium (IPPS)*, pages 791–795, 1998.

[8] M. Naimi and M. Trehel. How to detect a failure and regenerate the token in the log(n) distributed algorithm for mutual exclusion. *Lecture Notes in Computer Science*, 312:155–166, 1987.

[9] M.L. Neilsen. A generalized token-based mutual exclusion algorithm for wireless networks. In *Proceedings of the 20th Int'l Conference on Parallel and Distributed Computing Systems (PDCS-2007), Las Vegas, Nevada, USA*, ISCA, 2007.

[10] M.L. Neilsen. Model checking prioritized token-based mutual exclusion algorithms. In *19th International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, pages PDPTA–3940, 2013.

[11] M.L. Neilsen and M. Mizuno. A dag-based algorithm for distributed mutual exclusion. In *IEEE 11th International Conference on Distributed Computing Systems*, pages 354–360, 1991.

[12] K. Raymond. A tree-based algorithm for distributed mutual exclusion. *ACM Transactions on Computer Systems*, 7(1):61–77, 1989.

[13] M. Raynal. *Algorithms for Mutual Exclusion.* MIT Press, 1986.

[14] G. Ricart and A. K. Agrawala. An optimal algorithm for mutual exclusion in computer networks. *Communications of the ACM*, 24(1):9–17, 1981.

[15] M. Singh, D. Mishra, and D. Pandey. Designing an efficient algorithm in distributed system for mutual exclusion. In *2012 International Conference on Computer Technology and Science (ICCTS 2012)*, pages 161–164, 2012.

[16] I. Suzuki and T. Kasami. A distributed mutual exclusion algorithm. *ACM Transactions on Computer Systems*, 3(4):344–349, 1985.

[17] M. Trehel and M. Naimi. A distributed algorithm for mutual exclusion based on data structures and fault tolerance. In *Proc. IEEE 6th International Conference on Computers and Communications*, pages 35–39, 1987.

Cost-aware Short-term Load Forecasting of Power System

Kai-Chao Yang, Chung-Chieh Huang, and Jia-Shung Wang
Department of Computer Science, National Tsing-Hua University, Hsinchu, Taiwan
National Chip Implementation Center, Hsinchu, Taiwan

Abstract - *To precisely forecast the load in the power system, numerous training data have to be collected from sensors. As time goes by, the continuously increasing data cause more and more storage and bandwidth requirements. In this article, we present a cost-aware short-term load forecasting method which uses zonal prediction and multi-resolution data compression to reduce data size without significantly influence the forecasting accuracy. The user can collect just partial data from sensors for forecasting under a predefined tolerable prediction error, such that the system is robust subject to the precision degradation due to storage or bandwidth limitation. The experimental results demonstrate that 92.72% bandwidth can be saved with the prediction errors slightly increasing from 1.08% to 1.64%. Moreover, we also propose a similar-hour selection approach which helps the neural network to predict the next hour load. By integrating the proposed zonal prediction, similar-hour selection, and three-layer neural network together, the simulation results show that the prediction errors can be reduced to 0.95% ~ 1.18%.*

Keywords: Power system analysis computing; smart grid; prediction methods; data compression; neural networks

1 Introduction

The purpose of short-term load forecasting is to predict electricity loads in a short time interval, ranging from minutes to several days. It is an essential issue in the power system safety, management, and electric transportation planning. It is also a challenge to predict short-term loads accurately. The load variation is dependent not only on the load at the previous hour, but also on the load at the same hour on the previous day, the load at the same time in the previous week, or other similar days. The electricity load at a given time period is also affected by weather conditions, seasonal effects, and some anomalous events. There have been many methods used in load forecasting, such as multiplicative autoregressive models, dynamic linear or nonlinear models, autoregressive models, Kalman filtering, optimization techniques, and nonparametric regression. The most popular method is regression [1], which finds the relationship between variables by given data. In [2], a regression based daily peak load forecasting method with a transformation technique was presented. Recently, it has been discovered that the relationship between loads and related factors is distinctly non-linear, so regression methods are not satisfactory for load forecasting problems. Therefore, several methods based on Artificial Intelligence techniques such as fuzzy theory [3], expert systems [4], and neural networks [5]-[7][10]-[15] on load forecasting have been proposed. Among various methods on load forecasting, Artificial Neural Network (ANN) has received much attention due to its ability to learn complex and nonlinear relations. Most ANN methods for load forecasting adopt back propagation algorithms [5]: The desire outputs are considered as a part of inputs and the training process will adjust weights and bias in the network to match the real targets. The ANN approach in [6] was used to get the relationship between various load, temperature, and dew point values. Lately, methods based on similarity also get much attention.

Similar day method in [7] uses the information of the days which have similar weather condition including wind-chill temperature [8] humidex [9], and weekday index, to the forecasted day to estimate the load curve. The selected day is required to have the same weekday index and similar weather to that of tomorrow. Sometimes the wind-chill temperature is used in spring, fall, and winter, and the humidex measurement is only used in summer. Another similar day method is presented in [10]. The similar day selection is based on history load in the same season using Euclidean norm with weighted factors. These methods can deal not only with the nonlinear part of load curve, but also with weekends and anomalous events.

The methods we mentioned above focus on the forecasting model, but few of them have discussed the data used for load forecasting. Load forecasting usually covers an area with several zones. Different weather conditions and life styles in the area both significantly influence the electricity load demand. Theoretically, in order to increase the accuracy of forecasting results, more source data should be collected from sensors. However, continuously increasing population leads to rapid growth of data size. Hence how to store and transmit these big data has become an important issue. To reduce storage and bandwidth requirements without significantly reducing forecasting accuracy, we propose a cost-aware short-term load forecasting framework in this article. The proposed framework consists of zonal prediction, multi-resolution data compression, similar-hour selection, and neural networks. We divide an area to several zones. In each zone, a similar-hour neural network is applied to forecast the load. Then the forecasting results from all zones are aggregated to get the final result. Besides, the proposed multi-resolution data compression is applied to reduce the data size if a prediction error bound is assigned.

The idea of zonal prediction is inspired from [16], where the authors presented an optimal partition technique that performs prediction in each zone individually and then sum up predicted values as the result of the entire area. The main advantage of zonal prediction is reducing bandwidth requirements because the server collects only zonal forecasting results instead of all source data of sensors. Another reason to use zonal prediction is because each zone has its own climate and habits of load usage. If we use a universal model with a single load and weather information to predict the load of the whole area, it may ignore the diversity between regions in the area and increase the error of prediction. Besides, the load of the entire area is influenced by load variations of each zone. It is hard to observe the load trend or regularity of the entire area because of the complex interaction between influential factors of different zones. Compared with the load summation of all zones, it is easier to forecast the load of an individual zone due to fewer influential factors, so the aggregated zonal prediction is more precise than the entire area prediction.

Similar-hour selection is to find out the time that has the most similar load to the requested hour. We use hours instead of days because, in our experience, it is more likely to precisely forecast the load if more detailed information can be provided. An accurate estimated similar hour can increase the accuracy of ANN output. The experimental results show that each step of the proposed method can effectively reduce prediction error. Note that instead of focusing on the ANN structure, we mainly focus on the inputs fed into the neural networks, so different ANN structures can apply the proposed method to improve the estimation accuracy.

To further reduce transmission and storage overhead of source data, the idea of multi-resolution data compression is proposed. Here “multi-resolution” means the compression rate is adjustable. This idea is inspired by the observation that some queries may not need exact forecasting results. For example, a user wants to know if the predicted load is in a range. In this situation, it is not necessary to collect full-resolution data from sensors. Thus we apply EZW algorithm on source data, and build up the Rate-Distortion relationship to estimate the relationship between prediction errors and compressed sizes. By applying EZW algorithm, the bandwidth and storage requirements can be significantly reduced with slightly increasing of errors. The experimental results demonstrate that 92.5% bandwidth can be saved with the prediction errors increasing from 1.08% to 1.6%. The system can save even more required bandwidth if higher prediction errors are tolerable. Thus the proposed method is suitable for low bandwidth environments or unstable networks, such as wireless sensor networks.

The rest of the article is organized as follows. The four stages of the proposed cost-aware short-term load forecasting method are presented in Section 2.1 to 2.4. In Section 3, we use experiments to demonstrate the performance of each stage of the proposed method. The conclusions and the future works are drawn in Section 4.

2 Cost-Aware Short-Term Load Forecasting

A zone-based forecasting method using similar-hour approach is developed to predict the electricity load at next hour. As shown in Fig. 1, we divide the forecasted area into several zones and perform compression and prediction in each zone in order to reduce the amount of data and improve the prediction accuracy of zonal loads. Inside each zone, we first use historical data (include historical weather and loads) and the weather forecast of the forecasted time to find the similar hour of the forecasted time in the history. Next, we send the data of selected similar hour and some other load influential factors to a three-layer ANN. The output of the neural network is the zonal forecasting load result. Then we sum up the prediction values of each zone as the forecasting load of the whole area.

In the following sub-sections, the proposed zonal prediction and multi-resolution data compression are described in Section 2.1 and 2.2, respectively. Then the similar hour selection is discussed in Section 2.3. Finally, in Section 2.4 the training and forecasting process of our three-layer neural network is presented.

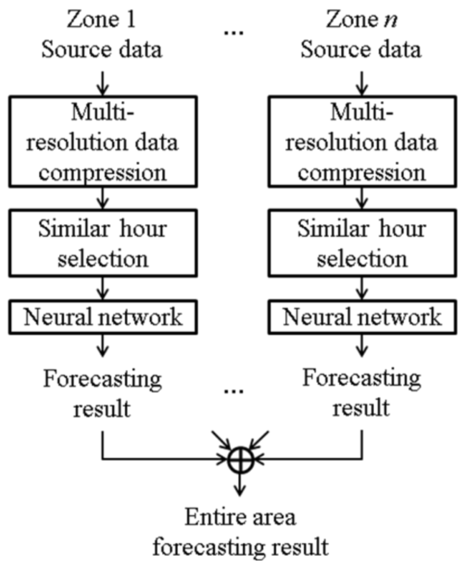


Fig. 1. Overview of the proposed forecasting method.

2.1 Zonal Prediction

Load forecasting usually covers a range of area with several zones. Each zone has its own climate and habits of load usage. TABLE I shows the load and temperature data of eight zones of New England in 2008 [17]. It can be seen that the electricity load and temperature vary in the entire area. Different weather conditions and life styles both significantly influence the load demand.

TABLE I. LOAD AND TEMPERATURE OF EIGHT ZONES OF NEW ENGLAND IN 2008

Zones	Temperature (F)		Load (MW)	
	Max	Min	Average	Peak
ME	88	-4	1321	1724
NH	97	-11	1323	2309
VT	92	-10	687	1040
CT	98	-2	3705	6962
RI	97	8	954	1835
SEMASS	97	5	1738	3284
WCMASS	94	0	2072	3635
NEMASSBOST	94	7	2988	5302
Range	88~97	-11~8	687~3705	1040~6962

If the forecasting result simply applies the mean of temperature and load in the entire area for prediction, it may ignore the diversity between zones in the area and increase the error of prediction. Therefore, we divide the entire area into several zones and use zonal information for prediction. Then these forecasted loads are aggregated (i.e., summed up) as the final result of the entire area. Let p be the "precise forecast" probability of a zone. Then we use the following equation to describe the probability that at least i of N zones are precisely forecasted. Here "precise forecast" is defined as the event that forecasting error is small enough.

$$P(X \geq i) = \sum_{j=i}^{N} C_j^N p^j (1-p)^{N-j} \tag{1}$$

From (1), it can be seen that as long as the forecasting error of each zone is small enough, there is high probability that most zones can be precisely forecasted. Thus the aggregated result will be accurate as well. For example, if p = 95% and N = 8, the probability that at least 6 zones are precisely forecasted is over 99%, i.e., it is very likely that the load of the requested area can be also precisely forecasted.

2.2 Multi-resolution Data Compression

Each load forecasting method requires historical data for prediction. As time goes by, continuously increasing historical data will occupy a lot of storage. Thus data compression is necessary to reduce storage requirements in a large system. However, after decompression from highly compressed data, recovered data will cause resolution degradation, which reduces the forecasting accuracy. It is difficult to balance between the compression rate and forecasting accuracy. For this reason, we apply multi-resolution data compression, so that the user can decide the compression rate according to requirements. Another advantage of multi-resolution data compression is reduction of transmission requirements since some queries do not request exact results. For example, a user wants to know if the predicted load is in a certain range. In this situation, it is not necessary to collect full-resolution data from sensors. To reduce transmission rate and save storage, we apply the Embedded Zerotree Wavelet algorithm (EZW) [18] on source data. The main advantages of EZW algorithm are simple, effective, and arbitrary coding rate. Thus it is very suitable for low-power sensing or computing devices. Fig. 2 shows the R-D (Rate vs. Distortion) relationship of source data of different zones in New England. Rate means the compressed size, the ratio of compressed data size and source data size. Distortion represents the MSE (Mean Squared Error) between source data and compressed data, formulated as follows,

$$MSE = \frac{1}{j-i+1} \sum_{t=i}^{j} (L_{d,t} - L_{d,i,j}^{c})^2 \tag{2}$$

In (2), $L_{d,t}$ is the load at the forecasting hour t on day d, and $L^c_{d,i,j}$ is the compressed load that represents the source (i.e., sensed) loads from $L_{d,i}$ to $L_{d,j}$. It can be seen that all curves in Fig. 2 are smooth and have similar downward trends, so the server can build up the R-D relationship in the training procedure. When a tolerable MSE is given, the required rate can be calculated according to the requested distortion. Then each zone collects source data according to the required rate to save transmission bandwidth. Besides, when a sensing device is running out of the storage, the EZW algorithm also helps reducing storage requirements by downgrading the resolution level of source data. In the next section, we will show the relationship between the prediction error and compressed size, so that the server can directly calculate the required rate from a tolerable prediction error assigned by the user.

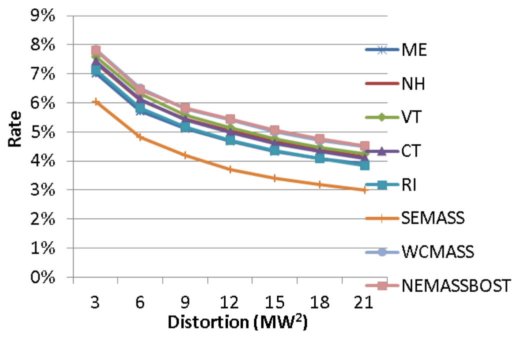


Fig. 2. R-D relationship of the EZW algorithm in different zones of New England.

2.3 Similar Hour Selection

Historical loads similar to the forecasted time are usually used as inputs for neural networks in load forecasting. The question is how to find the most suitable "similar historical load." To solve this problem, some previous researches such as [7] and [10] proposed similar day approaches, where the load curve is forecasted using information of the days with similar weather conditions of the forecasted day. Instead of similar day approaches, we propose the similar hour method using a similar hour in history data to enhance the accuracy of prediction as mentioned before. The similar hour is extracted based on the difference of temperature, dew point temperature, previous hour load, and load trend in historical load.

Our similar hour selection is inspired by two observations, illustrated by Fig. 3(a) and Fig. 3(b), respectively. In Fig. 3(a), suppose the forecasting hour is the forth hour, denoted as the red vertical line, and the actual load is represented as the solid curve. We use the actual load of the forecasting hour to exhaustively search for the most similar load in the same hour from historical data. The best result and its surrounding loads are shown by the dot curve. The exhaustive search is to find the hour in history which has the minimal load difference to the actual load, so the result can be treated as “best match.” In addition to the load of current forecasted hour, it can be seen that other parts of the similar curve are also very close to the actual one. However it is difficult to find out the most suitable similar curve like Fig. 3(a) because the actual load of the current forecasting time is unknown. Fortunately, we observe that if two curves are similar in the current hour, there is high probability that they will be also similar in the next hour, as demonstrated in Fig. 3(a). Therefore, we can use the previous hour of the forecasting hour to search for the most similar hour.

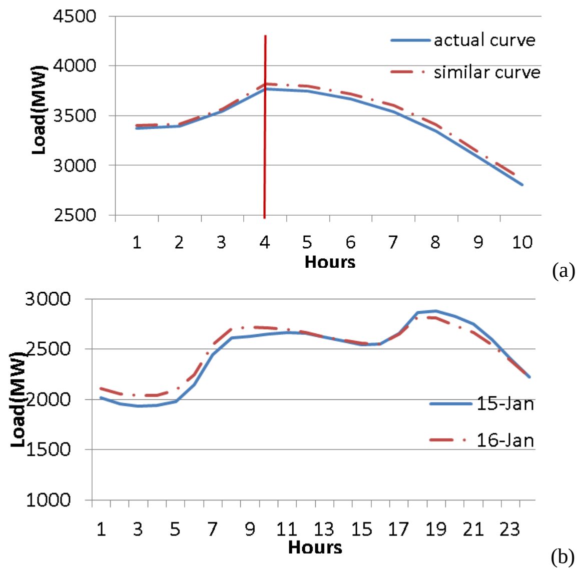


Fig. 3. (a) The actual load curve of the forecasted hour and its most similar load curve. (b) The load curves of two consecutive days.

The second observation is that two consecutive days have similar load trends. Here “trend” means the load curve tends to increase or decrease. For example, in Fig. 3(b) the load curves of 15-Jan and 16-Jan both decrease in the 20th hour. During searching for the similar hour, we can utilize this feature to filter out the candidates that have the opposite load trend to the hour on the day before the forecasted day. This can decrease the number of candidates in the search range. According to our experiments, on average 25% candidates can be filtered out.

Let $L_{d,t}$ be the load at the forecasting hour t on day d. Before searching for similar hours, two kinds of search ranges are defined, the hour search range and day search range. We search for similar hours in the hour search range of the day search range. The hour search range is set as [t-1, t+1], and the day search range is [d-45, d-1] (45 days before the current forecasted hour) $\cup$ [d-410, d-320] (45 days before and after the current forecasted hour in the previous year). It is unlikely to find out the most similar hour in days far away from the forecasting time point because load curves have different characteristics in different seasons. For example, suppose 15:00 on April 17, 2010 is the forecasting time point. Then the search range is every 14:00 to 16:00 from March 3, 2009 to May 1, 2009 and from March 3, 2010 to April 16, 2010.

The observation from Fig. 3(a) shows that if two load curves are similar in an hour, they will be similar in the next hour as well. Hence, the load of the hour previous to the forecasting hour can be viewed as the anchor in order to find similar hours in the search range. Thus we apply $L_{d,t\text{-}1}$ to find similar hours in the candidate set {$L_{d',t'}$ | d' $\in$ [d-45, d-1] $\cup$ [d-410, d-320] and t' $\in$ [t-1, t+1]}. Before starting the searching process, we can assume that $L_{d\text{-}1,t}$ has the same load trend as $L_{d,t}$ from the observation in Fig. 3(b), so candidates that have the opposite load trend to $L_{d\text{-}1,t}$ can be discarded first. Then we calculate the load difference between $L_{d,t\text{-}1}$ and candidates, and find out three candidate hours with smallest load difference. Among the three candidates, the one that has the most similar weather condition is selected. Here “the weather condition” means temperature and dew point temperature.

Consequently, the procedure to search for similar hours can be described as the following steps:

Step 1. Let $L_{d,t}$ be the load at the forecasting hour t on day d. Check the trend of $L_{d\text{-}1,t}$, and set the search range to R = {(d', t') | d' $\in$ [d-45, d-1] $\cup$ [d-410, d-320], t' $\in$ [t-1, t+1], and $L_{d',t'}$ has the same trend as $L_{d\text{-}1,t}$}

Step 2. Search for $L_{d1,t1}$, $L_{d2,t2}$, and $L_{d3,t3}$ such that

$$\begin{aligned} L_{d_1,t_1-1} &= \underset{L_{d',t'},\forall(d',t')\in R}{\arg\min}\left(\{\left|L_{d,t-1}-L_{d',t'-1}\right|\}\right) \\ L_{d_2,t_2-1} &= \underset{L_{d',t'},\forall(d',t')\in(R-(d_1,t_1))}{\arg\min}\left(\{\left|L_{d,t-1}-L_{d',t'-1}\right|\}\right) \\ L_{d_3,t_3-1} &= \underset{L_{d',t'},\forall(d',t')\in(R-(d_1,t_1)-(d_2,t_2))}{\arg\min}\left(\{\left|L_{d,t-1}-L_{d',t'-1}\right|\}\right) \end{aligned} \tag{3}$$

Step 3. Let $T_{d,t}$ and $D_{d,t}$ be the temperature and dew point temperature of the forecasting hour, respectively. Search for the most similar load $L_{d^*,t-1}$, such that

$$L_{d^*,t^*} = \underset{L_{d_i,t_i} \forall i=1,2,3}{\arg\min} \left(\left\{ \left| T_{d,t-1} - T_{d_i,t_i-1} \right| + \left| D_{d,t-1} - D_{d_i,t_i-1} \right| \right\} \right) \quad (4)$$

Step 4. (d^*, t^*) is the most similar hour at time (d, t).

2.4 Artificial Neural Network

The estimated similar load L_{d^*,t^*} is used for load forecasting in this sub-section. Load forecasting needs historical statistics to predict the current load demand, but the analysis of correlations among historical data is usually complex and nonlinear. The Artificial Neural Network (ANN) is suitable for load forecasting because it can learn complex and nonlinear relations among variables. In this sub-section, we use a three-layered Perceptron neural network composed of one input layer, one hidden layer, and one output layer. The number of hidden neurons is decided by the approach in [15]. The ANN starts by setting the estimated optimal number of hidden neurons as the square root of the product of the number of inputs and the number of outputs, and then the number is increased by one. For each hidden neuron number, we perform forecasting and record the result for comparison. The inputs of ANN are temperature, dew point temperature, listed as follows: 1. Temperature: $T_{d,t-1}$, T_{d^*,t^*}, and $T_{d,t}$. 2. Dew point temperature: $D_{d,t-1}$, D_{d^*,t^*}, and $D_{d,t}$. 3. Load: $L_{d,t-1}$, $L_{d-1,t}$, and L_{d^*,t^*}.

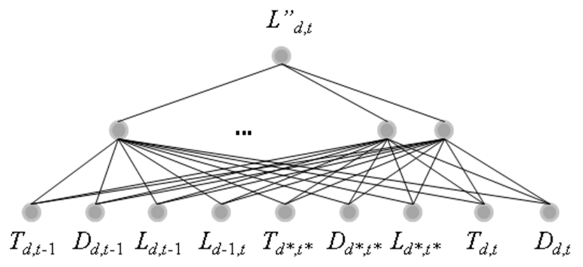


Fig. 4. The ANN used in the proposed method

Note that in addition to the hour previous to the forecasted hour and the day before the forecasted day, the similar hour information we found in the previous step is also treated as the input. The structure of ANN is shown in Fig. 4. The final output is the forecasting load $L''_{d,t}$ at hour t on day d.

The neural network is trained by historical data. During the training procedure, the ANN will automatically adjust the weights in the network based on the difference of the neural network output and the actual result. To avoid over-fitting, we examine the difference between predicted output $L''_{d,t}$ and actual result $L_{d,t}$.

$$\left| L''_{d,t} - L_{d,t} \right| < \varepsilon \quad (5)$$

If the value is lower than a assigned threshold ε, we stopped training. Otherwise, the network will continue training for N times, where N is a given number. The neural network model is trained by using the data of past two weeks before the forecasted day and past two weeks before and after the forecasted day in the previous year at the forecasting hour. For example, if the forecasting time is at 15:00, April 17, 2009, the training data is at every 15:00 from April 3, 2009 to April 16, 2009 and from April 3, 2008 to May 1, 2008.

3 Simulational Results

The performance of the proposed method for short-term load forecasting is demonstrated using hourly weather and load data of New England Independent System Operator in 2008-2010. Data of 2008 and 2009 are treated as historical data to forecast electricity loads in 2010. The electricity system in New England is divided into eight zones. Each zone has individual temperature, dew point, and load demand data. A year is separated into four seasons. We select April 15-21, July 15-21, October 15-21, and January 15-21 in spring, summer, fall, and winter of 2010 for prediction, respectively. In our ANN, the learning rate, learning times, and termination threshold are set to 0.7, 10000, and 0.01, respectively. Unless the absolute error of network output is lower than the termination threshold, the ANN will continue training until the learning times reach.

In the following sub-sections, we first show the accuracy of the proposed similar hour selection along with ANN. Then the benefit of aggregated zonal prediction is presented. Finally, multi-resolution data compression is applied to the predicted results, and the R-D relationship is illustrated.

3.1 Proposed Similar Hour Selection and ANN

TABLE II shows MAPE values of forecasting results produced by the proposed method and effects with/without the proposed similar hour selection. Note that here we do not apply aggregated zonal prediction, so the input of ANN includes data in all eight zones. In TABLE II, the proposed similar hour selection has a significant improvement in summer. The reason might be that the usage of air conditioners causes huge electricity consumption, but they do not work in fixed time periods every day. Thus the ANN is hard to predict the load if it does not apply similar hour selection.

For the purpose of performance evaluation, we exhaustively search for similar hours and apply these search results in our ANN, shown in TABLE II as well. The exhaustive search is to find the best similar hour, whose load value is closest to the actual load of forecasted hour in historical data by brute force within the same search range as in the proposed similar hour selection method. Thus a similar hour obtained by the exhaustive search can be treated as the optimal similar hour. From TABLE II, we can see the proposed similar selection method is close to the optimal results.

TABLE II. MAPE OF THE FORECASTING RESULTS USING PROPOSED ANN WITH SIMILAR-HOUR SELECTION

Season	MAPE of the proposed ANN (%)		
	without similar hour selection	with similar hour selection	with the best similar hour
Spring	1.87	1.40	1.00
Summer	4.23	1.47	1.26
Fall	2.81	1.60	0.93
Winter	3.30	1.11	0.93
Average	3.05	1.40	1.03

Fig. 5 compares forecasting curves with and without the proposed similar hour selection in summer. It can be seen that without the similar hour as input, the ANN predicts loads basically following the trend of previous cycle (i.e., loads in the previous day) because the input of ANN includes only the load of previous hour and the same hour in previous day. Since actual loads (the solid curve) of each cycle have different trends, the ANN without using similar hours produces wrong results. This phenomenon can be especially observed in the second and third cycles from 33th-73th hours, where the peak value of solid curve significantly varies in each cycle. Thanks to the similar hour selection, the ANN can predict much more accurate results.

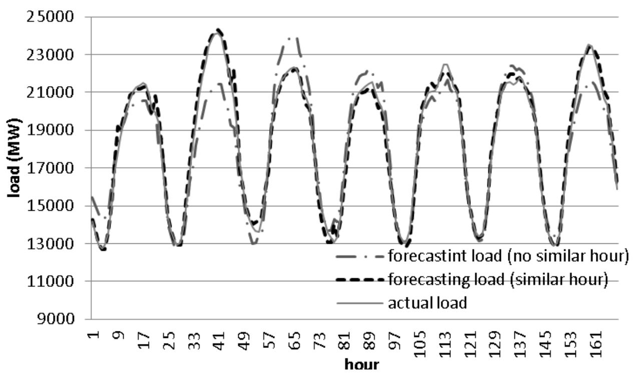


Fig. 5. The actual load curve and forecasting load curves in July 15-21, 2010.

3.2 Aggregated Zonal Prediction

In the previous experiments, data in all eight zones are fed into the ANN. In this sub-section, the ANN produces the forecasting load for each zone instead of the whole New England. Then the eight forecasting loads are aggregated to the final forecasting load. The second column in TABLE III shows MAPE results after performing the proposed similar hour selection, ANN, and aggregated zonal prediction. In comparison with TABLE II, there is a 23% improvement on average. Besides, the aggregated zonal prediction can also be performed with the optimal similar hour mentioned in the previous sub-section. The improvement is about 17% on average.

Furthermore, the proposed method is also compared with the load forecasting approach proposed by Mandel [10]. Mandel et al use Euclidean norm to evaluate the day similarity by load slope, load value, and temperature to select similar days, and then feed the weather forecast, weekday information of the forecasted day, and the load data of similar days into the ANN to perform several hours ahead load forecasting. Here we only take one-hour-ahead result of the Mandel method to compare with the proposed method. For fairness, both methods use data of New England for prediction. The results are shown in the rightmost column in TABLE III.

TABLE III. MAPE OF THE FORECASTING RESULTS USING PROPOSED ANN WITH SIMILAR-HOUR SELECTION

Season	MAPE of the aggregated zonal prediction (%)		MAPE of Mandel method [10]
	with similar-hour selection	with the best similar hour	
Spring	0.95	0.75	1.17
Summer	1.18	1.12	1.62
Fall	1.01	0.84	1.09
Winter	1.05	0.69	1.31
Average	1.08	0.85	1.30

3.3 Multi-resolution Data Compression

The above experiments have shown that the proposed method can accurately forecast electricity loads. In some situations, a user may tolerate a certain degree of errors. Thus the EZW algorithm is applied on source data to reduce bandwidth and storage requirements. In Fig. 2, we have shown the R-D relationship between the source data and compressed data. In TABLE IV, the first two columns illustrate the average compressed size of source data in all eight zones after applying the EZW algorithm. It can be seen that 92.72% to 95.97% transmission bandwidth and storage are saved as the distortion of source data ranges from 3MW^2 to 21MW^2. The side effect of multi-resolution compression is somewhat inaccurate forecasting results because imprecision source data are applied for prediction. Hence, we also show the MAPE of forecasting results in TABLE IV. When the average distortion is 3MW^2 and the compressed size is 7.28%, the MAPE is 1.64%. This means 92.72% bandwidth and storage can be saved and the prediction error only increases about 0.5% MAPE. According to TABLE IV, given target prediction accuracy (MAPE), we can compress data at the corresponding distortion or compressed size. And without sacrificing much prediction accuracy, we can reduce considerable bandwidth and storage requirements.

TABLE IV. RELATIONSHIP AMONG MAPE, COMPRESSED SIZE, AND DISTORTION

Distortion of source data (MW^2)	Compressed size of source data (%)	MAPE of forecasting results (%)
3	7.28	1.64
6	5.98	2.07
9	5.32	2.30
12	4.89	2.74
15	4.53	3.17
18	4.25	3.39
21	4.03	3.53

TABLE IV can be further divided into two relationships, MAPE-D (MAPE vs. Distortion) and R-D, as illustrated in Fig. 6. The MAPE-D relationship is roughly a linear curve, and the R-D relationship is a standard inverse proportional curve. From the above two observations, it is possible to formulate the relationship in TABLE IV. The discussions of this part are left for our future works.

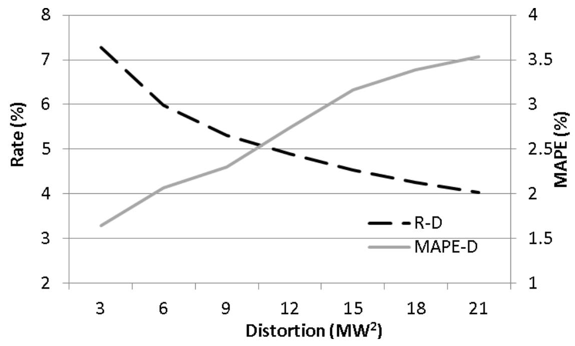


Fig. 6. The R-D relationship (dot curve) and MAPE-D (solid curve) relationship of forecasting loads

The two solid curves in Fig. 7 describe the MAPE-D relationships of forecasting data using aggregated zonal prediction and the entire New England area, respectively. The dot curve is the MAPE average of eight zone MAPE values. It can be observed that the slope of the proposed method is smaller than the two other curves. This means the forecasting loads of aggregated zonal prediction are less sensitive to the inaccurate source data, such that more bandwidth and storage can be saved due to smaller compressed size.

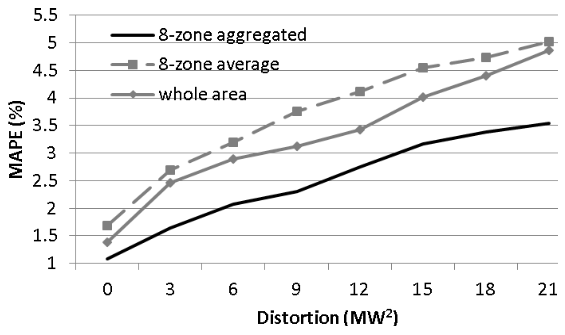


Fig. 7. The MAPE-D relationship of forecasting results using aggregated zonal prediction, 8-zone average, and whole area, respectively.

4 Conclusions

In this article, a zone based neural network using similar hour selection method is proposed. The errors (MAPE) of the proposed method in four seasons range from 0.95% to 1.18%. Besides, we also proposed multi-resolution data compression. By applying EZW algorithm, the bandwidth and storage requirements are significantly reduced with slightly increasing of errors. Thus the proposed method is suitable for wireless sensor networks, where transmission rates are limited.

There are other factors that could be taken into account to improve the accuracy of proposed similar hour selection and ANN, such as weather and humidity. However, too many factors may also interfere with the forecasting precision. This will be one of our future works. Besides, electricity loads are usually periodic. This feature may be useful for compression. We will try to let the forecast method more tolerant to data distortion through improving the compression techniques.

5 References

[1] M. Negnevitsky, P. Mandal, and A.K. Srivastava, "An overview of forecasting problems and techniques in power systems," *IEEE Power & Energy Society General Meeting*, pp.1-4, 26-30 July 2009.

[2] T. Haida and S. Muto, "Regression based peak load forecasting using a transformation technique," *IEEE Trans Power Syst*, vol.9, no.4, pp.1788-1794, 1994.

[3] K. -B. Song, Y. -S. Baek, D. -H. Hong, and G. Jang, "Short-term load forecasting for the holidays using fuzzy linear regression method," *IEEE Trans Power Syst*, vol.20, no.1, pp. 96- 101, Feb. 2005.

[4] K. -L. Ho, Y. -Y. Hsu, C. -F. Chen, T, -E. Lee, C. -C. Liang, T. -S. Lai, and K. -K. Chen, "Short term load forecasting of Taiwan power system using a knowledge-based expert system," *IEEE Trans Power Syst*, vol.5, no.4, pp.1214-1221, 1990.

[5] Yong Wang; Dawu Gu; Jianping Xu; Jing Li; , "Back propagation neural network for short-term electricity load forecasting with weather features," *International Conference on Computational Intelligence and Natural Computing*, vol.1, pp.58-61, 6-7 June 2009.

[6] Z.H. Osman, M.L. Awad, and T.K. Mahmoud, "Neural network based approach for short-term load forecasting," *IEEE Power Systems Conference and Exposition*, pp.1-8, 15-18 March 2009.

[7] Y. Chen, P.B. Luh, and S.J. Rourke, "Short-term load forecasting: Similar day-based wavelet neural networks," *World Congress on Intelligent Control and Automation*, pp.3353-3358, 25-27 June 2008.

[8] R. Osczevski and B. Maurice, "The new wind chill equivalent temperature chart," *Bull. Amer. Meteorol. Soc*., vol. 10, pp. 453–1458, 2005.

[9] J. M. Masterton and F. A. Richardson, "Humidex, a Method of Quantifying Human Discomfort Due to Excessive Heat and Humidity," *Environment Canada, Atmospheric Environment Service*, CLI 1-79, 1979.

[10] P. Mandal, T. Senjyu, N. Urasaki, and Toshihisa Funabashi, "A neural network based several-hour-ahead electric load forecasting using similar days approach," *International Journal of Electrical Power & Energy Systems, vol. 28*, no. 6, pp. 367-373, 2006.

[11] T. Senjyu, H. Takara, K. Uezato, and T. Funabashi, "One-hour-ahead load forecasting using neural network," *IEEE Trans Power Syst*, vol. 17, no. 1, pp. 113-118, 2002.

[12] R. Lamedica, A. Prudenzi, M. Sforna, M. Caciotta, and VO. Cencellli, "A neural network based technique for short-term forecasting of anomalous load periods," *IEEE Trans Power Syst*, vol. 11, no. 4, pp. 1749-1755, 1996.

[13] H. S. Hippert, C.E. Pedreira, and R. C. Souza, "Neural networks for short-term load forecasting: A Review and Evaluation," *IEEE Trans. Power Systems*, vol. 16, no. 1, pp. 44-55, 2001.

[14] 14 W. Sun, "Ant colony based feed forward NN short-term load forecasting model with input selection and DA clustering," *International Conference on Natural Computation*, vol.2, pp.300-304, 18-20 Oct. 2008.

[15] S. E. Papadakis, J. B. Theocharis, S. J. Kiartzis, and A. G. Bakirtzis, "A novel approach to short-term load forecasting using fuzzy neural networks," *IEEE Trans. Power Systems*, vol. 13, no. 2, pp. 480–492, 1998.

[16] S. Fan, K. Methaprayoon, and W.J. Lee, "Multi-area load forecasting for system with large geographical area," *IEEE Industrial and Commercial Power Systems Technical Conference*, pp.1-8, 4-8 May 2008.

[17] ISO New England. Available: http://www.iso-ne.com/index.html

[18] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Trans. Signal Processing*, vol. 41, no. 12, pp. 3445-3462, 1993.

Design of an FPGA-Based FDTD Accelerator Using OpenCL

Yasuhiro Takei, Hasitha Muthumala Waidyasooriya, Masanori Hariyama and Michitaka Kameyama
Graduate School of Information Sciences, Tohoku University
Aoba 6-6-05, Aramaki, Aoba, Sendai, Miyagi, 980-8579, Japan
Email: {takei, hasitha, hariyama, kameyama}@ecei.tohoku.ac.jp

Abstract— *High-performance computing systems with dedicated hardware on FPGAs can achieve power efficient computations compared with CPUs and GPUs. However, the hardware design on FPGAs needs more time than the software design on CPUs and GPUs. We designed an FDTD hardware accelerator using the OpenCL compiler for FPGAs in this paper. Since it is possible to design a hardware automatically from an OpenCL code, we can implement applications on FPGAs in a short time compared with the design by using a hardware description language. According to the result of the implementation of the FDTD accelerator on the FPGA, the processing speed is faster than a CPU. Moreover, its power consumption is about one-tenth of a GPU.*

Keywords: OpenCL, FPGA, FDTD method, Hardware accelerator

1. Introduction

In the field of the high performance computing such as three-dimensional image processing, electromagnetic simulation, fluid dynamics and DNA sequence, a very large scale computing system is required. However, the power consumption of high performance computer systems becomes a serious problem. The FPGA is attracting attention as the accelerator for such high-performance computing systems. A very large scale architecture for high performance computings can be implemented on a FPGA because of the advancement of the process technology. The power consumption of FPGAs is about one tenth as much as that of GPUs. However, very long time is required for the implementation of the FPGA-based accelerator. The software-based design on CPUs and GPUs needs only a software code by using C language or CUDA. On the other hand, the hardware-based design on FPGAs needs circuit modules for calculations, controls and connecting to the host PC by using a hardware design language(HDL).

To solve this problem, Altera Corporation released Altera SDK for OpenCL [1] which is the OpenCL compiler for FPGAs. OpenCL is the programming language for parallelized heterogeneous multicore architectures. OpenCL is standardized by the Khronos group [2]. The source code of the OpenCL is constituted by the host code and kernels. The initialization, the data-transfer from the host PC to the accelerator and running the kernels are described in the host code. The parallelized computation on the accelerator is described in the kernel code. As a feature of the OpenCL, the common source code can be run on the different architectures by using compilers corresponding to architectures such as multi-core CPUs, GPUs, the CELL processors and so on. In order to implement the OpenCL code on the FPGA board, Altera SDK for OpenCL can be used. This compiler does not require the HDL design for the calculation and connecting to the host PC by PCI express as shown in Fig.1, and the design time can be reduced. In the recent studies of the FPGA-based accelerator by using OpenCL, fractal image processing [6] and AES encryption encoding [7] have been reported. These studies achieve a low power and high performance computing compared with GPUs. In this article, we implement the FDTD (Finite-Difference Time-Domain) method accelerator by using the OpenCL compiler for FPGAs. We compare the performance of the FPGA-based accelerator with a CPU and a GPU in order to research the utility of the OpenCL compiler.

2. Implementation of an FPGA-Based FDTD accelerator by using OpenCL

The FDTD method[3] has been widely used in an electromagnetic simulation. Since the FDTD method

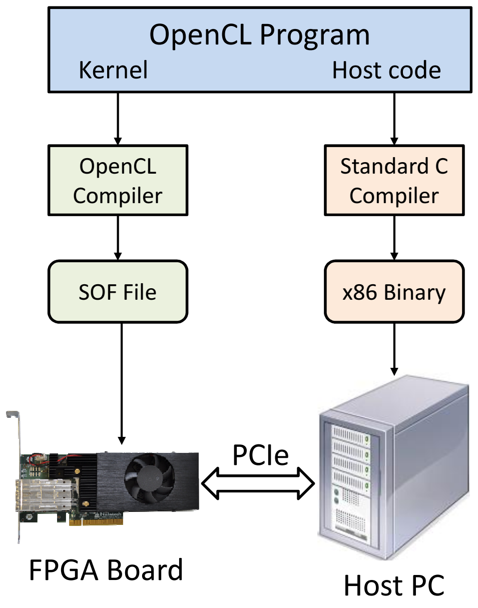


Fig. 1: OpenCL implementation on the FPGA

has the high degree of parallelism, there are many studies which use computer clusters, GPUs [4], [5] and FPGAs [8], [9] to accelerate the FDTD method. Figure 2 shows the flowchart of the FDTD method. It starts with transferring initial data of the electric and magnetic fields. Then the initial data are processed to obtain the electric field information for the first time step. After that, the boundary conditions are applied. Then the magnetic field information are obtained and the boundary conditions for the magnetic field are applied. These steps are repeated for a given number of time steps. Equation (1) shows the electric field computation. Equations (2) and (3) show the magnetic field computation. Electric and magnetic fields in x, y, z directions are denoted by E and H respectively. The time step is denoted by n and the coordinates of the 2D fields are denoted by i and j. Note that the boundaries of the electric and magnetic fields are calculated differently. Parameters Px, Py, Qx, Qy are determined by the permittivity, the permeability, the size of grids and the length of the time step. A detailed description of the FDTD is given in [3].

$$\begin{aligned} E_z^{n+1}(i,j) &= E_z^n(i,j) \\ -P_y(i,j)&\left\{H_x^{n+\frac{1}{2}}(i,j+1/2) - H_x^{n+\frac{1}{2}}(i,j-1/2)\right\} \\ +P_x(i,j)&\left\{H_y^{n+\frac{1}{2}}(i+1/2,j) - H_y^{n+\frac{1}{2}}(i-1/2,j)\right\} \end{aligned} \tag{1}$$

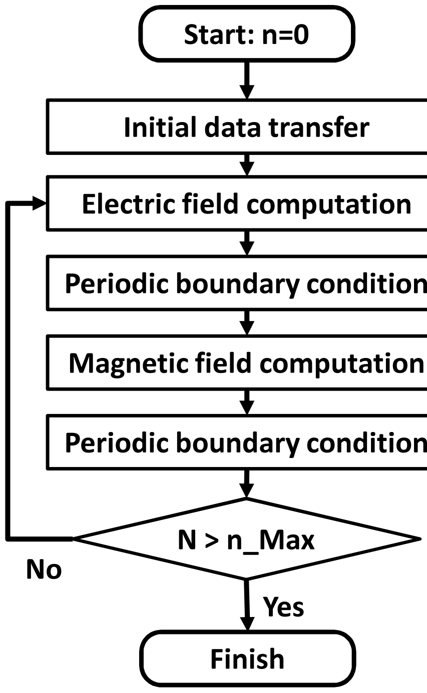

Start: n=0
Initial data transfer
Electric field computation
Periodic boundary condition
Magnetic field computation
Periodic boundary condition
N > n_Max
No
Yes
Finish

Fig. 2: The flowchart of the FDTD method

$$\begin{aligned} H_x^{n+\frac{1}{2}}(i,j+1/2) &= H_x^{n-\frac{1}{2}}(i,j+1/2) \\ &-Q_y(i,j)\left\{E_z^n(i,j+1) - E_z^n(i,j)\right\} \end{aligned} \tag{2}$$

$$\begin{aligned} H_y^{n+\frac{1}{2}}(i+1/2,j) &= H_y^{n-\frac{1}{2}}(i+1/2,j) \\ &-Q_x(i,j)\left\{E_z^n(i+1,j) - E_z^n(i,j)\right\} \end{aligned} \tag{3}$$

Figure 3 shows the example of the OpenCL code for computing the electric field by Eq.(1). Thanks to the descriptions of global_id(0) and global_id(1) in line 4 and 5 respectively, the electric and magnetic fields in the coordinates of the two-dimensional fields are accessed in parallel. In this implementation, we make four kernel codes for updating electric fields, updating magnetic fields, applying boundary conditions and exciting electric fields. Counting time steps and running kernels are controlled by the host code. Let us explain about data-transfers between a host PC to accelerators. At the first of the FDTD computation, initial values of electric fields and magnetic fields, and the values of Px, Py, Qx, Qy are transferred from a host PC to accelerators. The values of excited electric fields are also transferred in every time step. When the value of time steps reaches the given value, the results

```
__kernel void  efield (__global float *Ez, __global float *Hx, __global float *Hy,
                              __global float *Px, __global float *Py)
{
       int i =get_global_id(0); // 2D Thread ID x
       int j =get_global_id(1); // 2D Thread ID y

       if((i>=1)&&(j>=1)){
          Ez[j*N+i] = Ez[j*N+i]-Py[j*N+i]*(Hx[j*N+i]-Hx[(j-1)*N+i])+Px[j*N+i]*(Hy[j*N+i]-Hy[j*N+(i-1)]);
       }
}
```

Fig. 3: An example of the OpenCL code

of the simulation are transferred from accelerators to a host PC.

3. Evaluation

We implement the FDTD method by C language on "Intel Core i7 920", and by OpenCL on "nVidia Geforce GTX 580" and "Nallatech P385-A7 FPGA board"[10]. This FPGA board has the Altera StratixV GX A7, a DDR3-SDRAM(8GB) and a PCI-Express. We use Visual Studio 2010(64bit) and nvidia GPU computing SDK 4.2 for the compilation on the CPU and the GPU. We use Altera SDK for OpenCL 13.0 for the compilation on the FPGA. Figure 4(a) shows the simulation model. This model has N $\times$ N grids (N=128,256,512). The electric field at (N/2,N/2) is excited as shown in Fig.4(b). The boundary area is a perfect conductor ($Ez = 0$). The single-precision floating-point is used for the simulation.

Table 1 shows the resource usage on the FPGA. The number of processing units and the degree of the kernel vectorization can be changed [11]. When the kernel vectorization is used, each scalar operation in the kernel, such as addition or multiplication, is translated to an SIMD operation by the compiler. The processing units becomes smaller, and the throughput becomes higher.

Figures 5(a),5(b) and 5(c) show the results of the simulation of the electric field on the CPU, the GPU and the FPGA, respectively. As shown in these figures, the simulation results are almost consistent with each other. However, there are very small errors since the rounding of a floating point are different depending on the platforms.

Table 2 shows the processing time of the FDTD method. The processing time of the FPGA is about half of that of the CPU. However, the processing time of the FPGA is about 22 times longer than that of the GPU. One of the main reasons is the bandwidth of the global memory. The bandwidth on the FPGA board is further narrower than that of the GPU board. In order to increase the performance on the FPGA, the memory access to the global memory should be reduced by improving the OpenCL code. The power consumption of the FPGA is 25W [6]. This power consumption is about one tenth of that of GPU board. Based on these results, the FPGA accelerator can achieve very low power and high performance computing if the OpenCL code is improved to reduce the memory access.

4. Conclusion

In this article, we implement the FPGA-based accelerator for the electromagnetic simulation by using the OpenCL compiler. The processing time of FPGA is about half of that of CPU. However, the processing time is longer than that of GPU. The power consumption of the FPGA is about one tenth of that of GPU. For the future work, we improve the specialized OpenCL code for the FPGA. For example, the resource usage of processing units becomes small if the fixed-point calculation is used. Moreover, we are now designing the FPGA-based accelerator for the electromagnetic simulation on antennas and optical devices.

Acknowledgement

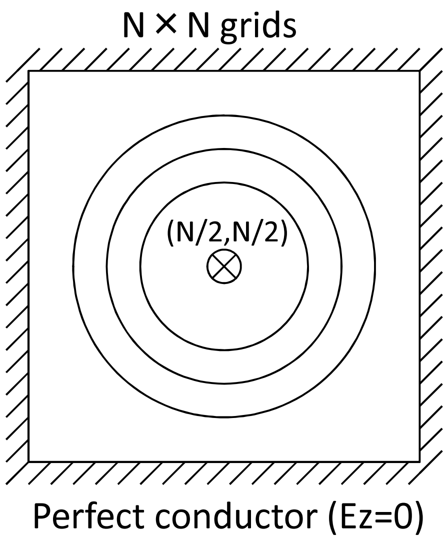
This work is supported by JSPS KAKENHI grant number 24300013.

Table 1: Resource usage

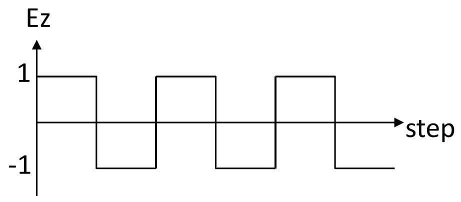
PEs	LEs	FFs	DSPs	RAMs
1	76965(16%)	127326(14%)	4(2%)	650(25%)
4	144072(31%)	302233(32%)	16(6%)	1411(55%)
16(Vectorization)	204092(44%)	435734(46%)	64(25%)	2042(80%)

Table 2: Processing time (s) (Time steps=1000)

Grids	CPU(Corei7 920)	CPU(Corei7 920) +GPU(GTX 580)	CPU(Xeon E5 2069) +FPGA(Nallatech P385-A7)
128×128	0.249	0.156	2.030
256×256	1.294	0.203	2.780
512×512	11.232	0.249	5.400



(a) Simulation model



(b) Excitation of the electric field

Fig. 4: Set up of the simulation

References

[1] Altera corpolation, “Altera SDK for OpenCL Programming Guide”, http://www.altera.co.jp/literature/hb/opencl-sdk/aocl_programming_guide.pdf

[2] Khronos group, http://www.khronos.org/opencl/

[3] H. S. Yee, “Numerical Solution of Initial Boundary Value Problems Involving Maxwell’s Equations in Isotropic Media”, IEEE Transactions on Antennas and Propagation, Vol.14, No.3, pp.302-307, 1966.

[4] Z. Bo, X. Zheng-hui, R. Wu, L. Wei-ming, S. Xin-qing, “Accelerating FDTD algorithm using GPU computing”, International Conference on Microwave Technology & Computational Electromagnetics (ICMTCE), pp.410-413, 2011.

[5] T. Nagaoka and S. Watanabe, “A GPU-based calculation using the three-dimensional FDTD method for electromagnetic field analysis”, International Conference on Engineering in Medicine and Biology Society (EMBC), pp.327-330, 2010.

[6] D. Chen and D. Singh, “Fractal Video Compression in OpenCL:An Evaluation of CPUs, GPUs, and FPGAs as Acceleration Platforms”, Design Automation Conference (ASP-DAC) 18th Asia and South Pacific, pp.297-304, 2013.

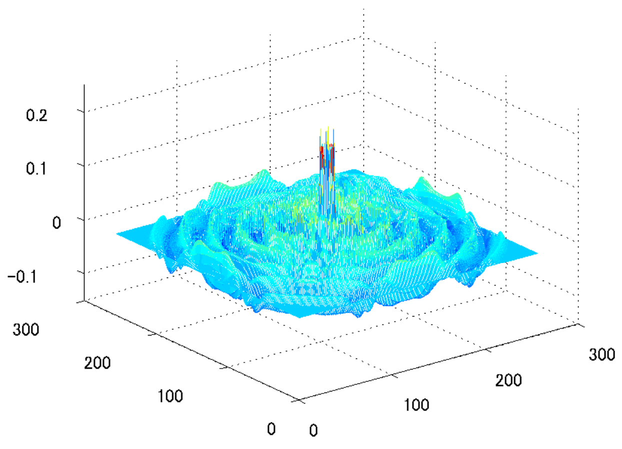
[7] Nallatec, “40Gbit AES Encryption Using OpenCL and FPGAs”, http://www.nallatech.com/images/stories/technical _library/white-papers/40_gbit_aes_encryption_using_opencl _and_fpgas_final.pdf

[8] W. Chen, P. Kosmas, M. Lesser and C. Rappaport, “An FPGA Implementation of the Two Dimensional Finite Difference Time Domain (FDTD) Algorithm”, ACM/SIGDA International Symposium on Field-Programmable Gate Arrays(FPGA), pp.213-222, 2004.

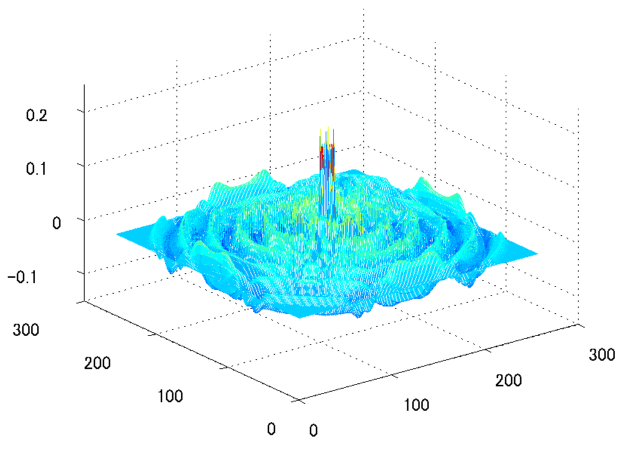
[9] K. Sano, Y. Hatsuda, W. Luzhou and S. Yamamoto, “Performance Evaluation of Finite-Difference Time-Domain (FDTD) Computation Accelerated by FPGA-based Custom Computing Machine”, Interdisciplinary Information Sciences, Vol.15, No.1, pp.67-78, 2009.

[10] Nallatec, “OpenCL FPGA Accelerator Cards”, http://www.nallatech.com/opencl-fpga-accelerator-cards.html

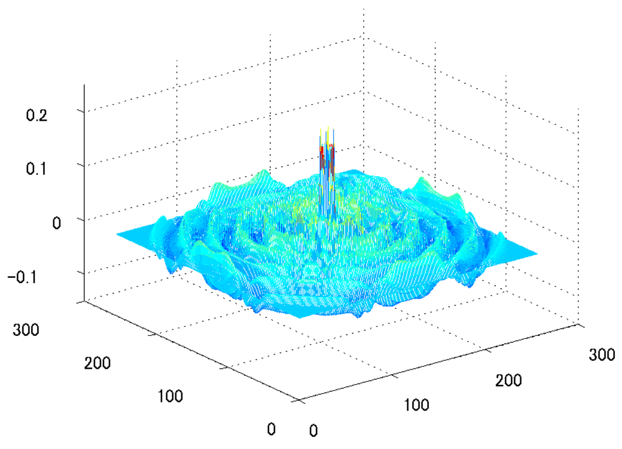
[11] Altera corpolation, “Altera SDK for OpenCL Optimization Guide”, http://www.altera.co.jp/literature/hb/opencl-sdk/aocl_optimization_guide.pdf



(a) Result on CPU



(b) Result on GPU



(c) Result on FPGA

Fig. 5: Results of the simulation (N=256, Time steps=250)

A Practical Election Protocol Based on an Unreliable Failure Detector in Distributed Systems

Yong Hwan Cho, Sung-Hoon Park and Seon-Hyong Lee
School of Electrical and Computer Engineering, Chungbuk National Unvi. Cheongju
ChungBuk 361-763
E-mail: [yhcho,spark]@chungbuk.ac.kr

Abstract

A Leader is a Coordinator that supports a set of processes to cooperate a given task. This concept is used in several domains such as distributed systems, parallelism and cooperative support for cooperative work. In completely asynchronous systems, there is no solution for the election problem satisfying both of safety and liveness properties in asynchronous distributed systems. Therefore, to solve the election problem in those systems, one property should be weaker than the other property. If an election algorithm strengthens the safety property in sacrifice of liveness property, it would not nearly progress. But on the contrary, an election algorithm strengthening the liveness property in sacrifice of the safety property would have the high probability of violating the safety property. In this paper, we presents a safety strengthened Leader Election protocol with an unreliable failure detector and analyses it in terms of safety and liveness properties in asynchronous distributed systems.

Keywords : Distributed Computing, Leader Election, Asynchronous Distributed Systems, Failure Detectors

1. Introduction

Distributed systems consist of groups of processes that cooperate in order to complete specific tasks. A *Leader* is a Coordinator that supports a set of processes to cooperate a given task. This concept is used in several domains such as distributed systems, parallelism and cooperative support for cooperative work.

To elect a *Leader* (or Coordinator) in a distributed system, an *agreement* problem must be solved among a set of participating processes. This problem, called the *Election* problem, requires the participants to agree on only one leader in the system [1]. The problem has been widely studied in the research community [2,3,4,5,6]. One reason for this wide interest is that many distributed protocols need an election protocol.

The Election problem is described as follows. At any time, there is at most one process that considers itself a *leader* and all other processes consider it as to be their only leader. If there is no leader, a leader is eventually elected.

The so-called FLP impossibility result, which states that it is impossible to solve any non-trivial agreement in an asynchronous system even with a single crash failure, also applies to the election problem [7]. That means that there is no solution for the election problem satisfying both of safety and liveness properties in completely asynchronous distributed systems.

It must be pointed out, however, that the impossibility result really means “not always possible,” as opposed to “never possible.” As a matter of fact, any algorithm that tries to solve the Election Problem cannot always make progress without violating safety; there exist cases in which the algorithm violating safety, although it is very unlikely.

Therefore, to solve the election problem in those systems, one property should be weaker than the other property. If an election algorithm strengthens the safety property in sacrifice of liveness property, it would be difficult to progress. But on the contrary, an election algorithm strengthening the liveness property in sacrifice of the safety property would have the high probability of violating the safety property. There exists a trade-off between safety property and liveness property.

A stable election protocol, which implies the safety strengthened election protocol, is needed in a practical distributed computing environment. Consider a mission critical distributed system such as an electronic commerce system that runs multiple servers in which one of them roles a master (leader) and others are slaves.

To have data consistency among the servers in the system, this system should not violate safety property, which means that all processes connected the system never disagree on a *leader*. In those systems the safety property is more important property than the liveness property.

As a classic paper, there is Garcia-Molina's Invitation algorithm to solve election problem in asynchronous distributed systems. The algorithm strengthens the progress property rather than safety and it allows more than two leaders in the systems.

Our idea is based upon the Garcia-Molina's Invitation algorithm for solving the election problem in asynchronous distributed systems [2]. He redesigns the Bully algorithm for synchronous distributed systems into the Invitation algorithm for asynchronous distributed systems by using a specification that is weak enough to be solvable, allowing the algorithm to progress even in completely asynchronous distributed systems.

His specification uses a strong progress requirement, allowing executions in which even a single process suspicion of the current leader's crash and its attempted leader election from the members may lead a progress to elect a new leader from all processes.

We propose an election algorithm that requires processes to elect a new leader only when they agree with the current leader's crash. This requirement is strong because, if no set of processes agrees on the current leader's crash, no progress is made. The requirement is, however, much more stronger than the one proposed by Garcia-Molina's Invitation algorithm in that it implicitly states that the leader election of any process be allowed only on the basis of only it's own knowledge.

In this paper, we presents a safety strengthened Leader Election protocol with an unreliable failure detector and analyses it in terms of safety and liveness properties in asynchronous distributed systems.

Our algorithm, based on a standard three phases commit protocol, is fully distributed. It does not extend the asynchronous model of concurrent computation to include global failure detectors. Progress of the algorithm can be guaranteed only in case of minimal violating a safety property.

The rest of the paper is organized as follows. In Section 2, we describe our system model and definitions. In Section 3, this paper relates the election specification to other ways to solve the election problem. In Section 4, this paper provides a stable algorithm that solves the Leader Election problem. In Section 5, we ensure the correctness of the algorithm by proving that it satisfies the two properties of the specification given in Section 4. Finally, Section 6 summarizes the main contributions of this paper and discusses related and future works.

2. Model and Definitions

Our model of asynchronous computation with failure detection is the one described in [9,10]. In the following, we only recall some informal definitions and results that are needed in this paper.

2.1 Processes

We consider a distributed system composed of a finite set of processes $\Omega=\{p_1,p_2,..,p_n\}$ where processes are identified by unique id's. Communication is by message passing, *asynchronous* and *reliable*. Processes fail by crashing; Byzantine failures are not considered.

Every pair of processes is connected by a communication channel. That is, every process can send messages to and can receive messages from any other. We assume processes are able to probe a communication channel for incoming messages. Communication channels are considered to be reliable, FIFO, and to have an infinite buffer capacity. A reliable channel ensures that a message, sent by a process p_i to a process p_j, is eventually received by p_j if p_i and p_j are correct (i.e. do not crash).

Asynchrony means that there is no bound on communication delays or process relative speeds. A process that has been infinitely slow for some time and has been unresponsive to other processes may become responsive again at any time. Therefore, processes can only suspect other processes to have crashed, using local failure detectors.

A failure detector is a distributed oracle which gives hints on failed processes. We consider algorithms that use failure detectors. Local failure detectors are assumed to be inaccurate and incomplete. That is, local failure detectors may erroneously suspect that other, operational processes have crashed or that crashed processes are operational. Since local failure detectors run independently at each process, one local failure detector may perceive a failure, but other detectors may perceive it at a different time or not at all. The failure model allows processes to crash, silently halting their execution. Because of the unpredictable delays experienced by the system, it is impossible to use time-outs to accurately detect a process crash.

We assume that a process communicates with its local failure detector through a special receive-only channel on which the local failure detector may place a new list of id's of processes not suspected to have crashed. We call this list the local connectivity view of the process. Each process considers the last local connectivity view received from its local failure detector as the current one.

2.2 Election Specifications

The Election problem is described as follows: At any time, as most one process considers itself the leader, and at any time, if there is no leader, a leader is eventually elected. More formally, the Election Problem is specified by the following two properties:

- *Safety*: All processes in the local connectivity view of the process never disagree on a *leader*.
- *Liveness*: All processes should eventually progress to be in a state in which all processes connected to the system agree to the *only one* leader.

3. Circumventing The Impossibility Result

In this section, we relate the election specification to other ways to solve the election problem.

- In an asynchronous model augmented by global failure detectors, processes have access to modules that (by definition) eventually reflect the state of the system. Therefore, progress and safety can be guaranteed unconditionally.
- In a timed asynchronous model, processes must react to an input, producing the corresponding output or changing state, within a known time bound. Under this model, progress and safety can be guaranteed if no failures and recoveries occur for a known time needed to communicate in a timely manner.
- In a completely asynchronous model, progress cannot always be guaranteed without violating safety and failure detectors in practice eventually reflect the system state, but they must be considered arbitrary. Correct processes react in practice within finite time, but this time cannot be quantified. Therefore, in order to guarantee a solution, we need a weaker specification of the problem.

Our approach falls into the last category that originated with Garcia-Molina's work [2]. Our election algorithm, however, differs from Garcia-Molina's in several ways.

- Processes in Garcia-Molina's model do not need to wait to get consensus about the current leader's crash. If one process suspects that the leader failed, it may attempt to elect the new leader. Garcia-Molina's specification says that, if one process attempts to be a new leader, it eventually should be elected as a leader. Our specification requires all processes in a set to agree on the current leader crash before changing their new leader.

- Garcia-Molina's specification allows a solution in which the attempted change of a leader divides all processes into several sub-groups. Our specification does not allow such a sub-group because it states that if all processes in a system agree on a new leader, they must eventually accept such a leader.

In our model stability is also required for progress, but, at variance of the above case, it is not necessarily related to the state of the system. In other words, eventual progress is required when there is agreement among a set of the local failure detectors, even if failures and recoveries continue to occur in the system.

4. Election Algorithm

We provide a robust algorithm that solves the Leader Election problem given in Section 2. The algorithm is based on the three asynchronous phases.

- A prepare phase, in which a process propose a new leader that the other processes agree with.
- A ready phase, in which all process that agree on the new leader acknowledge the reservation of the potential leader.
- A commit phase, in which the new leader is finally elected, and all process accept it their only leader.

4.1 Solution Sketch

The main idea for the algorithm is as follows. A process *p* that is informed by its local failure detector of a leader's crash and that has the smallest id among processes in its new local connectivity view sends a message to all processes in its view proposing to update the current leader with the new leader.

Each process received the message records this proposal until the newly elected leader is the same as the proposed new leader in its local view. At which point, it responds by sending back an Accept or Retry message to the process that proposed the leader update. The Accept message is sent if the process agrees on the proposed leader in its local current view.

Upon sending the Accept message, the process reserves the prospective leader, so that no other proposal is accepted for that system. Upon receiving a Retry message, the proposing process restarts the first phase of the algorithm, sending a new Propose message to all processes in its view.

When the proposing process has collected Accept messages from all processes in its view, it starts the commit phase by sending commit messages, ordering other processes in its view to commit the leader update. Upon receiving a commit message, the processes accept the reserved prospective leader as a their new leader.

4.2 Code Description

The code is shown in Fig. 1. The first guarded command in Fig. 1 shows how a process *p*, when informed of a change in its local connectivity view, set its view to be current and checks if the current leader has crashed. If the leader has crashed, it set the variable *LeaderStatus* to be false. When *leaderStatus* is false in the third guarded command in Fig. 1, the process *p* checks it is the minimum id among the processes in v_p. If *p* is the minimum id, it increases the round and proposes itself as a new prospective leader and initializes its *ack* array to zero.

The second guarded command in Fig. 1 checks for incoming messages from other processes. These may be proposals for a new leader (Propose), rejections to propose a new leader (Rej), acceptances of a proposed new leader (Acc), orders to commit a new leader (Commit) or orders to abort a proposed new leader (Abort).

Upon receiving a proposal message from process q, process p stores the new leader's id proposed by q at position q of the array NewLeader and stores the proposed round at position q of the array RoundIn, then sets position q of the array Prop to true to record the receipt of the proposal from q and sets the CurView to false to refresh the current view of the system.

```
Upon received $v_p$ from FD:
   $CurView := true$;
   If $CurLeader \notin v_p$ then $LeaderStatus := 0$;
   end-if
   If $p = min(v_p)$ then $Round := Round + 1$;
            Call $start\_election()$;
   end-if

Upon received ($Propose, PropLeader, k$) from $q$:
    $Prop := true$; $CurView := false$;
    $NewLeader := PropLeader$ ; $RoundIn := k$;
    Call reply_election();

Upon received ($Reject, k$) from $q$:
    If $Round = k$ then
       Send ($Abort, Round$) to $\forall j \in v_p$;
       For $\forall j \in v_p$, $ack[j] := 0$;
    end-if

Upon received ($Accept, k$) from $j$:
   If $Round = k$ then $ack[j] := 1$;
      If for $\forall q \in v_p$, $ack[q] = 1$ then
         Send ($Commit, PropLeader, Round$) to
                    $\forall q \in v_p$;
         For $\forall q \in v_p$, $ack[q] := 0$;
      end-if end-if

Upon received ($Commit, PropLeader, k$) from $j$:
   If $RoundIn = k$ then
            $CurLeader := PropLeader$;
            $LeaderStatus := 1$;
    end-if

Procedure $Start\_election()$:
   $PropLeader := p$;
   Send ($Propose, PropLeader, Round$) to
         $\forall q \in v_p$;
   For $\forall q \in v_p$, $ack[q] := 0$;

Procedure $Reply\_election()$;
   If ( $CurView \wedge Prop$ ) then $Prop := false$;
      If ($Newleader \leq min(v_p) \wedge RoundIn > Round$)
         then Send ( $Accept, PropIn$) to $q$;
               $Next = RoundIn + 1$;
      end-if
    else Send ($Reject, PropIn$) to $q$;
   end-if
```

Fig. 1. The Algorithm.

Upon receiving a proposal message from process q, process p stores the new leader's id proposed by q at position q of the array NewLeader and stores the proposed round at position q of the array RoundIn, then sets position q of the array Prop to true to record the receipt of the proposal from q and sets the CurView to false to refresh the current view of the system.

If process p later agrees on the proposed new leader, it sends a response to process q (see last guarded command in Fig. 1). The response is either an acceptance of the new leader at position NewLeader[q] if the minimum id among the process in v_p is greater or equal than the id of proposed NewLeader[q] and the proposed round greater than the current round; or it is an rejection to the proposed new leader if the minimum id among the process in v_p is less than the id of proposed NewLeader[q] or the proposed round less or equal than the current round.

A rejection to the proposed new leader consists of sending back to q the proposed round. An acceptance consists of acknowledging the proposed new leader at position NewLeader[q].

We now examine the guarded commands of the remaining message types. A process p that receives a rejection to the its proposal sends all processes in v_p a message to abort the proposed round and reinitializes the ack array to zero.

A process p that receives an acceptance regarding its proposed new leader receives the proposed round. If the received round is equal to the round

of the most recent proposal sent, process p sets the element at position q in the array ack to 1 to record the acceptance.
Then, it inspects the ack array to check if all entries are 1. If so, p starts the commit phase by broadcasting its previously proposed new leader and the corresponding proposed round PropRound to all processes in v_p and reinitializes the ack array to zero.
A process p that receives an order to commit a new leader at position q from process q, simply sets the current leader to the proposed new leader and sets the current round to the proposed round.

5. Correctness

We can ensure the correctness of the algorithm by proving that it satisfies the two properties of the specification given in Section 4.

5.1 Safety

Theorem 1. *The algorithm described in Section 4 satisfies the safety condition of the specification (Property 1, Section 2): At any point in time, all processes connected the system never disagree on a leader.*

Proof. Either all processes remain in the start state or some process *p* receives the proposed leader as its leader. In the start state, the safety property holds since all processes are in the state in which a leader has not been elected. If some process *p* receives its leader by committing a proposed leader at a given position *q*, it must have received a Commit message from some process *q*; therefore, *q* must have received Accept messages regarding its proposal of a new leader from all processes in v_p including *p*. It follows from the last guarded command in Fig. 1 that, if process *p* has accepted the proposal of process *q*, it will not accept any other proposal for new leader, making it possible to commit at most single proposed leader. Therefore, process *p* either commits the process at position *q* as a new leader or ends up with position *q* by aborting the proposed new leader. Therefore safety property holds.

5.2 Liveness

Theorem 2. *The algorithm described in Section 4 satisfies the liveness condition of the specification (Property 2, Section 4): All processes should eventually progress to be in a state in which all processes connected to the system agree to the only one leader.*

Proof. By contradiction, a non-progress means that the new leader is not elected forever even though there is no leader; therefore, no Commit messages must be sent. Since the number of processes is finite, there must be at least one process whose id is the minimum value in v_p and that process eventually sends a Propose message. Call this process *p*. By the code in Fig. 1, we see that, to have no Commit message, each time *p* sends a Propose message, it should be rejected by other process. It follows that, in order to abort infinitely many Propose messages, other process *q* must reject the proposed messages infinitely often. Propose messages are rejected either when the minimum id of v_p is greater than the id of the proposed leader or because of a Propose message already has been received (see Fig. 1).
The first case is ruled out because it implies that some process always considers that there is a process that is alive and whose id is less that the id of proposed new leader. But by strong completeness of a failure detector it is contradiction.
The second case is also ruled out, because it implies that other process q sends infinitely many proposals of the other leader. But by eventual strong accuracy of a failure detector, the process q knows that there is a process whose id is less that its id. Therefore it is contradiction.

6. Concluding Remarks

We have presented a robust election protocol with a reliable failure detector in completely asynchronous systems. We have assumed our local failure detectors to be inaccurate and incomplete. With this approach, the leader election specification states explicitly that progress cannot always be guaranteed. In practice, our requirement for progress is weaker than that stated in the original specification of having a set of processes sharing the same leader.

In fact, if the rate of perceived a leader failures in the system is lower than the time it takes the protocol to make progress and accept a new leader, then it is possible for the algorithm to make progress every time there is a leader failure in the system. This depends on the actual rate of a leader failures and on the capacity of the failure detectors to track such failures.

In [10], Chandra and Toueg note that failure detectors defined in terms of global system properties cannot be implemented. This result gives strength to the approach of relaxing the specification and of having a robust election protocol. In real world systems, where process crashes actually lead a connected cluster of processes to share the same connectivity view of the network, convergence on a new leader can be easily reached in practice.

References

[1] G. LeLann, “Distributed Systems–towards a Formal Approach,” in Information Processing 77, B. Gilchrist, Ed. North–Holland, 1977.

[2] H.Garcia-Molian, “Elections in a Distributed Computing System,” IEEE Transactions on Computers, vol. C-31, no. 1, pp. 49-59, Jan. 1982.

[3] H. Abu-Amara and J. Lokre, “Election in Asynchronous Complete Networks with Intermittent Link Failures.” IEEE Transactions on Computers, vol. 43, no. 7, pp.778-788, 1994.

[4] H.M. Sayeed, M. Abu-Amara, and H. Abu-Avara, “Optimal Asynchronous Agreement and Leader Election Algorithm for Complete Networks with Byzantine Faulty Links.,” Distributed Computing, vol. 9, no. 3, pp.147-156, 1995.

[5] J. Brunekreef, J.-P. Katoen, R. Koymans, and S. Mauw, “Design and Analysis of Dynamic Leader Election Protocols in Broadcast Networks,” Distributed Computing, vol. 9, no. 4, pp.157-171, 1996.

[6] G. Singh, “Leader Election in the Presence of Link Failures,” IEEE Transactions on Parallel and Distributed Systems, vol. 7, no. 3, pp.231-236, March 1996.

[7] M. Fischer, N. Lynch, and M. Paterson, “Impossibility of Distributed Consensus with One Faulty Process,” Journal of the ACM, pp. 374-382. (32) 1985.

[8] T. Chandra and S.Toueg, “Unreliable Failure Detectors for Reliable Distributed Systems,” Journal of ACM, Vol.43 No.2, pp. 225-267, 1996.

[9] D. Dolev and R Strong, “A Simple Model For Agreement in Distributed Systems,” In Fault-Tolerant Distributed Computing, pp. 42-50. B. Simons and A. Spector ed, Springer Verlag (LNCS 448), 1987.

[10]T. Chandra, V. Hadzilacos and S. Toueg, “The Weakest Failure Detector for Solving Consensus,” Journal of ACM, Vol.43 No.4, pp. 685-722, 1996.

Autonomic Management for Availability And Performance in SaaS Model

Nadir K.Salih, Tianyi Zang
School of Computer Science and Engineering, Harbin Institute of Technology, Harbin, Heilongjiang, China

Abstract - *SaaS application gave many aspects of the business management. For that it became available and using in many domains. It will be needed to realize customer's requirements from design to runtime. In this paper we have described the service availability for business process to perform the necessity of this kind of application. Depending on our SaaS model in three levels of the management we applied dynamic analysis by PRISM for availability and performance requirement. We followed the probability description because it the best way to define QoS requirements. It help us to use Markov chain which it described transition can occur from any state to other state and represented a simple or a compound event. For explain our research work we have taken the online booking as a running example. The result is we have adapted the SaaS online booking system by detecting or predicting violation in availability and performance requirements. Finally we present a conclusion and future work.*

Keywords: SaaS application, availability, Markov chain, probability

1 Introduction

In this paper we have showed the benefits of the autonomic management for our new model in SaaS application. Exactly we have defined the proportion of time for the system is functional and working to represented by availability in our model levels. We started at first by describing probability of the availability in the user level when the customer requires services from tenant level. It clearly obtained some scenarios for user activities will have an effect on the system availability. In the same description we introduced availability in the tenant level and the provider level. And the important thing is the availability of the services introduced from our SaaS model depends on availability of the resources. Secondly we used formal language probabilistic computation tree logic (PCTL), and continuous stochastic logic (CSL) which they have defined QoS requirements obviously. With the quality evaluation Markov chain we described discrete and continuous states of our model [1]. All processes have explained and verification models by used online booking example. We have summarized our contributions of this paper as follow:

- A novelty in availability requirement for our three levels in SaaS model.
- The hierarchy of availability in our model, availability of the provider level means availability of the services, availability of the services level depends on availability of the resources level.
- Used formal language is probabilistic computation tree logic (PCTL) and continuous stochastic logic (CSL) which they defined QoS requirements obviously.
- Quality evaluation model by DTMCs, CTMCs
- Approximation of availability and performance metrics it has adapted SaaS application and shows what the user and the tenant expecting from the system.
- Empirical evaluation by using PRISM as a model checker. It made dynamic analysis for detecting or predicting violation of the SaaS application requirements.

We organized this paper by beginning with the description for availability of the SaaS model in section 2. In section 3 we defined the formal definition of QoS requirement. By the sequence we introduced the type of the quality evolution model in section 4. In section 5 we have described running example for verification model. We made the empirical evaluation for availability and performance requirements in section 6. In section 7 we mentioned the related work. Finally, we present the conclusion and point to future work.

2 Availability of SaaS Model

Availability of the user level depends on availability of the tenant level and availability of tenant level depends on availability of the provider level. Therefore, a detailed of the availability and analysis for this architecture can be carried out to support design architectural decisions. Different variants of this architecture can be modeling and comparing with respect to the availability objectives to be fulfilled. For example the hierarchy of the availability requirement it depicted in figure 1. As we have described new model for the SaaS application in our last research. Here we extended it to manage availability and performance requirements. First of all we show the availability of three levels user, tenant, and provider as follow:

2.1 Availability Model in User Level

We have taken online booking hotel as example. The user level can access through web start with browse, booking, payment, and exit. We can describe many scenarios for user level by different probabilities as appeared in figure 2

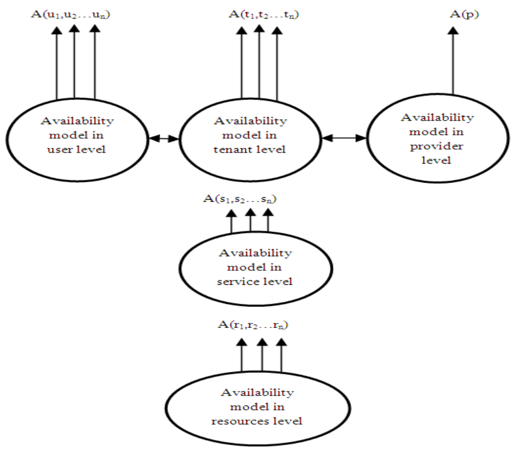

A(u1,u2...un)
A(t1,t2...tn)
A(p)
Availability model in user level
Availability model in tenant level
Availability model in provider level
A(s1,s2...sn)
Availability model in service level
A(r1,r2...rn)
Availability model in resources level

Fig 1 Availability of SaaS Model

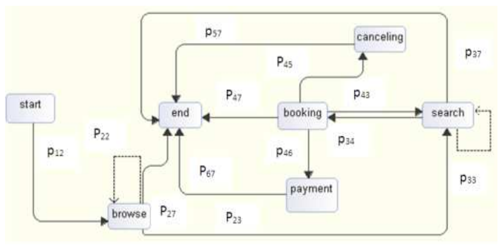


Fig 2 Availability Model in User Level

Here in user level just the customer want to receive services from online booking hotel system. But in this level the user can do some processes without benefit travel agencies like just browse and searching without booking. This will effect to the resources availability for other they want to book. That can be appeared in scenarios in table 1 below: show different type of users by their probabilities in every scenario.

Table 1 User Level Scenarios

Scenarios	type1	type2	type3
Start-browse-end	10%	6%	2%
Start-browse-browse-end	20%	10%	12%
Start-browse-search-end	15%	15%	15%
Start-browse-search-booking-end	20%	40%	20%
Start-browse-search-booking-cancelling-end	60%	50%	30%
Start-browse-search-booking-payment-end	20%	50%	70%
Start-browse-search-booking-search-payment-end	30%	45%	65%
Start-browse-search-booking-search-cancelling-end	40%	20%	10%

For example in scenario 1 (start-browse-end) the probability of user type 1 is 10 percent, user type 2 is 6 percent, and user type 3 is 2 percent. The user type has big value of the probability that can more effect in availability. In other scenario (Start-browse-search-booking-search-payment-end) this completes the booking service and gave good income by user type 3 for travel agency.

2.2 Availability Model in Tenant Level

We look in tenant or travel agencies level through the web of monitoring customer's booking activity as depict in figure 3. Travel agency it can divide the period of booking in a normal season or low season this mean resources availability in low season will be high. *Pij* it shows the probabilities between states of process in travel agencies level.

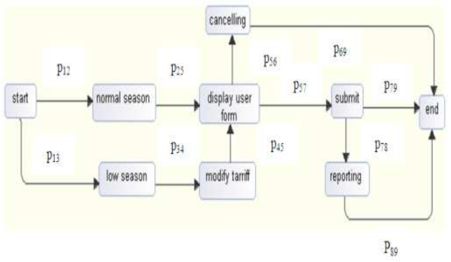


Fig 3 Availability Model in Tenant Level

Probability variable from travel agency to travel agency depend on the services introduce and resources availability. Availability model in this level is very important to serve many users in fewer resources to minimize the costing.

2.3 Availability Model in Provider Level

The provider level need availability model to manage all travel agencies. Begin from monitoring all events of travel agencies that serve all customers. And modification information of travel agencies by adds new agencies or deletes any redundancy data for them. After that the system in this level can show the costing or payment of the services provide from provider to travel agencies.

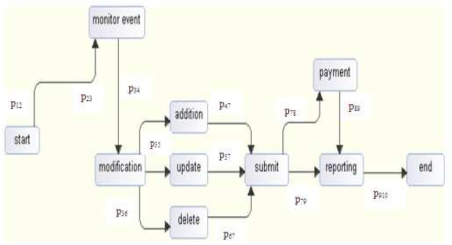


Fig 4 Availability Model in Provider Level

By this description in provider level we showed the availability model for services that can be introduce in this level. We represented this availability by probabilities between states inside the model as depicted in figure 4.

2.4 Availability Model in Service Level

Availability model for service level can depend on web server, application server, and data base server that will be interact between any level of our model. Any service in figure 5 it has need to availability of web server at first to browse the service and the next probability for application server availability to process the functionality of service

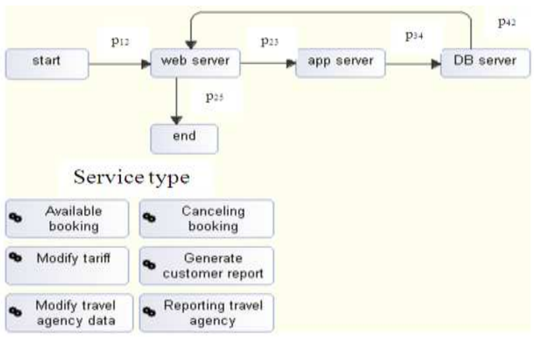


Fig 5 Availability Model in Service Level

Then it need to availability of database server to modify and save data. In the last it need to the web server again to show the result or the request of the demand.

3 Formal Definition of QoS Requirements

The estimation of the QoS requirements [2] is very important for optimization SaaS application in our model. We used two formal languages are probabilistic computation tree logic (PCTL) and continuous stochastic logic (CSL), which they defined QoS requirements obviously. PCTL is defined by the following syntax:

$\Phi ::= \text{true}\ |a|\ \Phi \wedge \Phi|\ \neg\ \Phi\ |\ \rho_{\bowtie p}\ (\Psi)$

$\Psi ::= X\ \Phi\ |\ \Phi\ U^{\leq t}\ \Phi$

Where $p \in [0, 1]$, $\bowtie \in \{<, \leq, >, \geq\}$, $t \in N \cup \{\infty\}$, and a represents an atomic proposition. The temporal operator X is called next and U is called until. Formula generated from Φ is referred to as state formula and they can be evaluated to either true or false in every single state. While the formula generated from Ψ are named path formula and their truth is to be evaluated for each execution path. PCTL can naturally represent availability-related properties for a DTMC model of the application. For example, we may easily express constraints that must be satisfied concerning the probability of reaching absorbing failure or success states from a given initial state. These properties belong to the general class of reachability properties. Reachability properties are expressed as $\rho_{\bowtie p}(\diamond\Phi)$ which expresses the fact that the probability of reaching any state satisfying Φ has to be in the interval defined by constraint $\bowtie p$. In most cases, Φ just corresponds to the atomic proposition that is true only in an absorbing state of the DTMC. In the case of a failure state, the probability bound is expressed as $\leq x$, where x represents the upper bound for the failure probability; for a success state it would be instead expressed as $\geq x$, where x is the lower bound for success. PCTL is an expressive language through which more complex properties than plain reachability may be expressed. Such properties would be typically domain-dependent, and their definition is delegated to system designers.

We have considered the continuous stochastic logic to state properties on CTMC models. For CTMC there are two main types of properties relevant for analysis: steady-state properties where the system is considered when equilibrium has been reached. And it is the transient properties where the system is considered at specific time points or intervals. CSL is able to express both steady-state and transient properties by means of the S and ρ operators, respectively. The syntax of CSL is recursively defined as follows:

$\Phi ::= \text{true}\ |a|\ \Phi \wedge \Phi|\ \neg\ \Phi\ |\ S_{\bowtie p}|\quad \rho_{\bowtie p}\ (\Psi)$

$\Psi ::= X^{\leq t}\ \Phi|\ \Phi\ U^{\leq t}\ \Phi$

Where $p \in [0, 1]$, $t \in R\geq 0 \cup \{\infty\}$, $\bowtie \in \{<, \leq, >, \geq\}$, and a represents an atomic proposition. The semantics of Boolean operators is the same defined for PTCL. As well as the possibility it defines derived operators, both boolean and temporal. To address the semantics of S andρ, let us first introduce the notation $\pi@t$, which denotes the state in which the execution described by trace π is at time t. As before, $\pi[i]$ denotes the i-th state the execution passed through, thus $\pi@t = \pi[i]$ with $i = \min_i\{t \leq \sum_{j=0}^{i} tj\ \}$.

4 Quality Evaluation Model

The uses of models extend beyond the initial design of an application support both the evolution of the software architecture. And it is devises suitable for reconfiguration strategies in dynamic contexts. To model complex interactions between components, use other kinds of models like Markov chains or more generally state space models. Many examples of dependencies among system components have been observed in practice and captured by Markov models. State-space based model which states represent various conditions of the system. Transitions between states indicate occurrences of events. State can keep track of number in functioning resources of each type, states of recovery for each failed resource, number of tasks of each type waiting at each resource, and allocation of resources to tasks. A transition can occur from any state to any other state and can represent a simple or a compound event. For that we looked in the Markov chain has two types in discrete and continuous time:

4.1 Discrete SaaS User-Tenant-Provider Model

State-transition systems augmented with probabilities formalizing path-based on properties of Discrete SaaS User-Tenant-Provider Model (DSUTP) model [3]. We turn to the problem of constructing a discrete time Markov chain with a given initial distribution, α, and transition matrix, P. More explicitly, for the discrete set S = {1, 2, . . . } (the finite state space is handled similarly), we assume the existence of:

(i) An initial distribution $\alpha = \{\alpha_k\}$ giving the associated probabilities for the random variable X_0. That is, for $k \in S$,

$$\alpha_k = P\{X_0 = k\}\ldots\ldots\ldots\ldots\ldots..(1)$$

(ii) Transition probabilities, p_{ij} , giving the desired probability of transitioning from state $i \in S$ to state $j \in S$:

$$pij = P\{X_n+1 = j \mid X_n = i\}.......(2)$$

Note that we will require that α is a probability vector in that $\alpha_k \geq 0$ for each k and $\sum_{k\in s} \alpha_k = 1$ We further require that for all $i \in S \sum_{j\in s} p_{ij} = 1$.

We simply says that the chain will transition somewhere from state i (including the possibility that the chain transitions back to state i). The problem is to now construct a discrete time Markov chain for a given choice of α and $\{p_{ij}\}$ using more elementary building blocks: uniform random variables. Implicit in the construction is a natural simulation method.

4.2 Continuous SaaS User-Tenant-Provider Model

A Continuous SaaS User-Tenant-Provider Model (CSUTP) is characterized by state changes that can occur at any arbitrary time the Index space is continuous and the state space is discrete valued [4]. A CSUTP can be completely described by: Initial state probability vector for X (*t0*):

$P(X(t_0) = k), k = 0,1,2, \dots \dots$ …………….(3)

Transition probability functions (over an interval):

$p_{ij}(v,t) = P(X(t) = j | X(v) = i), for\ 0 \le v \le t\ and\ i,j = 0,1,2 \dots$

$p_{ij}(t,t) = \begin{cases} 1, & if\ i = j \\ 0 & otherwise \end{cases}$ $and\ , \sum_{j \in I} p_{ij}(v,t) = 1\ \forall\ i; ,0 \le v \le t$…………………………….....(4)

A CSUTP is said to be irreducible if every state can be reached from every other state, with a non-zero probability. A state is said to be absorbing if no other state can be reached from it with non-zero probability. Notion of transient, recurrent non-null, recurrent null are the same as in a DSUTP. There is no notion of periodicity in a CSUTP.

5 Running Example for Verification SaaS Model

From our activity diagram in previous research we have described the workflow representing online booking system to customer. We have defined SaaS application in three management levels. We have taken samples of the services for every level. For example provider level has: (1) modify travel level data, (2) monitor events, and (3) reporting, for tenant level has: (1) check user form, (2) modify tariff, and (3) customer reporting, and user level has: (1) available booking, (2) cancelling booking, and (3) searching. The costing or payment process for this system classified in two types: first payment from customer to travel agencies for giving services like booking. Second payment from travel agencies to provider for prepares resources and data management for them. In our system need to monitor the availability of the services, there are some properties will be very important like failure rate (*FR*) of services, the costing(*C*) of the services, execution time (*ET*) inter-arrival rate (μ), and maximum request of services (λ). From property we can define some requirements for suitable availability and performance metrics for our model:

R₁: The probability P_1 of the service booking failure from user web site P_1=0.13

R₂: The probability P_2 of the service cancelling failure from user web site P_2=0.14

R₃: The probability P_3 of the service modify tariff failure from tenant web site P_3=0.01.

R₄: The probability P_4 of the service monitor event failure from provider web site P_4=0. 05.

R₅: The probability P_5 the service demand of booking during high season will drop if it take more than 5 second, P_5 more than or equal 0.5.

R₆: The probability P_6 the service request of booking during high season will drop if it take more than 10 second, P_6 more than or equal 0.8.

R₇: The probability P_7 the service request of monitor event from provider to tenants will drop if it take less than 7 second, P_7 more than or equal 0.4.

SaaS application administrators require suitable availability in QoS requirements [5]. Approximation of metrics [6] that will adapt SaaS application and what the user and tenant expecting from system will help administrators. The availability services of our three levels model it has depicted in figure 6 by DSUTP model.

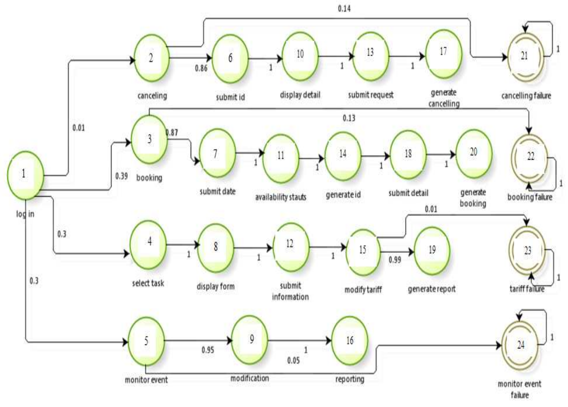


Fig 6 Online Booking DSUTP Model

Requirements R_1–R_4 can be translated in PCTL as follows:

$P_{\ge 0.13} = [\diamond\ s = 22]$ The probability of eventually reaching state 22 booking failure, which corresponds to the successful completion of the session (see Fig. 6) is greater than or equal 0.13

$P_{\ge 0.14} = [\diamond\ s = 21]$ The probability of eventually reaching state 21 cancelling failure, which corresponds to the successful completion of the session (see Fig. 6) is greater than or equal 0.14

$P_{\ge 0.01} = [\diamond\ s = 23]$ The probability of eventually reaching state 23 modify tariff failure, which corresponds to the successful completion of the session (see Fig. 6) is greater than or equal 0.01

$P_{\ge 0.05} = [\diamond\ s = 24]$ The probability of eventually reaching state 24 monitor event failure, which corresponds to the successful completion of the session (see Fig. 6) is greater than or equal 0.05

Performance requirement for our model in three levels it can realize the continuity of SaaS system activity. We have defined the state rate in a time for three levels in our running example see table 2. To show these continuous states and their requests we have depicted CSUTP model in figure 7.

Table 2 State Rate in A time

	State rate(λ)	Value req\sec
1	Login-cancel	10
2	Login-booking	15
3	Cancel process	20
4	Generate cancel	5
5	Booking process	30
6	Generate booking	8
7	Login-travel agency	5
8	Receive job	10
9	Done job	5
10	Login-provider	3
11	Monitor event	5
12	Modification by provider	20

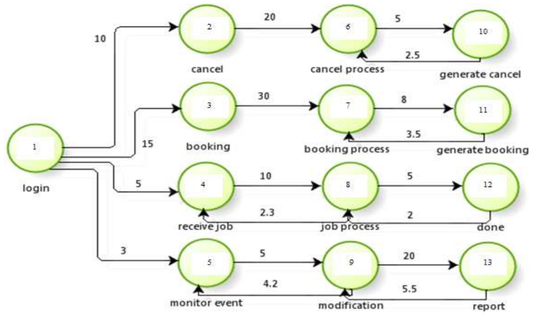


Fig 7 Online Booking CSUTP Model

Performance requirements have been mapped into the following CSL formula (notice that the notion of time here is continuous and not discrete in time steps.

$P_{\geq 0.5} = [\diamond^{\leq 5}\ s = 11]$ The probability of eventually to reach state 11 (generate booking) within 5 s is greater than or equal 0. 5

$P_{\geq 0.8} = [\diamond^{\leq 10}\ s = 12]$ The probability of eventually to reach state 12 (request booking done) within 10 s is greater than or equal 0. 8

$P_{\geq 0.4} = [\diamond^{\leq 7}\ s = 5]$ The probability of eventually to reach state 13 (request of monitor event report) less than 0.5 s is greater than or equal 0.4

6 Empirical Evaluation

We have adapted the SaaS application automatically identified by detecting a validation of requirement for our model in three levels. The possible verification of requirements can be achieved by using model checking techniques. We used DSUTP model and connected with environment which can effect in SaaS availability may be change over a time. The requirements from *R1* to *R4* are depicted by DSUTP for analysis availability requirements and represented the failure of services. By CSUTP model we have defined R5 to R7 for analysis performance requirements [7].

6.1 Availability Requirement

We used PRISM as model checker to dynamic analysis temporal logic formula for all possible value. For availability we have an assign to configurable SaaS parameters by depict the probability variation of the requirements failure to consider into valid and invalid range. From figure 6 we have supposed some failures for three levels of our model. Here we used PRISM coding to DSUTP model failure of online booking system services see the listing 1.

Table 3 The Pseudo-code of DSUTP for R1 to R4

A pseudo-code of DSUTP for R_1 to R_4

```
dtmc
module failure
// local state
s: [1..24] init 1;
// state failure
d: [21..24] init 21;
[ ] s=1->0.01:(s'=2)+0.39: (s'=3)+0.3: (s'=4) + 0.3:(s' =5) ;
[ ] s=2 -> 0.86 : (s' =6) + 0.14 : (s' =21)  &  (d' =21) ;
[ ] s=3 -> 0.87 : (s' =7) + 0.13 : (s' =22) &  (d' =22) ;
[ ] s=4 -> 1 : (s' =8);
[ ] s=5 -> 0.95 : (s' =9) + 0.05 : (s' =24) &  (d' =24) ;
[ ] s=6 -> 1 : (s' =10);
[ ] s=7 -> 1 : (s' =11);
[ ] s=8 -> 1 : (s' =12);
[ ] s=9 -> 1 : (s' =16);
[ ] s=10 -> 1 : (s' =13);
[ ] s=11 -> 1 : (s' =14);
[ ] s=12 -> 1 : (s' =15);
[ ] s=13 -> 1 : (s' =17);
[ ] s=14 -> 1 : (s' =18);
[ ] s=15 -> 0.99 : (s' =19) +0.01 : (s' =23) &  (d' =23) ;
[ ] s=16 ->  (s' =16);
[ ] s=17 ->  (s' =17);
[ ] s=18 -> 1 : (s' =20);
[ ] s=19 ->  (s' =19);
[ ] s=20 ->  (s' =20);
[ ] s=21 -> 1 : (s' =21);
[ ] s=22 -> 1 : (s' =22);
[ ] s=23 -> 1 : (s' =23);
[ ] s=24 -> 1 : (s' =24);
Endmodule
```

Experimental results about our method we used the probabilistic model checker PRISM on the example to verify the satisfaction of the previous properties, we got the following probabilities:

• "Probability of cancelling failure is equal to 0.0014"

• "Probability of booking failure for a user is equal to 0.0507"

• "Probability of tariff failure for travel agencies equal to 0.003"

• "Probability of monitor event failure for provider level equal to 0.015"

We supposed input from outside environment to our model that has modeling different services. Two services from user level cancelling failure and booking failure. The output of PRISM indicated to validation of input because the probability 0.0014, 0.0507 for cancelling failure, and booking failure respectively inside the range of requirements (cancelling failure from user web site P_2=0.14, booking failure from user web site P_1=0.13). In the tenant level the probability of tariff failure is 0.01 no violated in model checker because it 0.003. And so for provider level the failure of monitor event for travel agencies activities is 0.05 more than the output of PRISM 0.015 it is valid value. The output of this analysis is in figure 8 bellow.

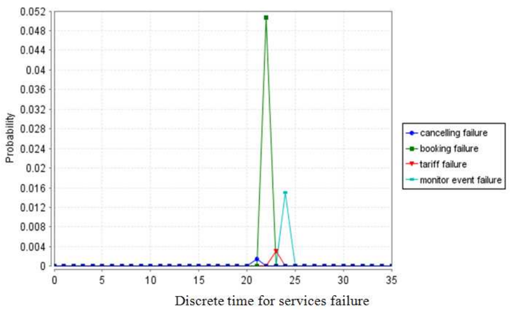


Fig 8 Analysis of Availability Requirements

Model checker help online SaaS application to detect the violation that will lead to failure it look like prognostics strategies [8] to what will happen. And we understand the effect of services failure for availability of application. Definitely we increased availability of the SaaS application to be functional and working.

6.2 Performance Requirements

Here we used PRISM coding to CSUTP model of the online booking system services for performance requirement [9]. See the listing 2.

Table 5 The Pseudo-code of CSUTP for R_5 to R_7

A pseudo-code of CSUTP for R_5 to R_7

```
ctmc
 module GCbooking
 // local state
 s : [1..11] init 1;
 // value of the drop
 d :[10.. 11] init 10;
      [] s=1 -> 15 : (s'=3) ;
      [] s=3 -> 30 : (s'=7) ;
      [] s=7 -> 8 : (s'=11) &(d' =11);
      [] s=11 -> 3.5 : (s'=7) ;
 endmodule
module Gdonebooking
  // local state
  s : [1..12] init 1;
  value of the drop
  d :[11.. 12] init 11;
      [] s=1 -> 5 : (s'=4) ;
      [] s=4 -> 10 : (s'=8)   ;
      [] s=8 -> 2.3 : (s'=4) + 5 : (s'=12) & (d' =12)  ;
      [] s=12 -> 3.5 : (s'=8) ;
endmodule
module Gdonereportevent
  //local state
  s : [1..13] init 1;
  // value of the drop
     d: [12..13] init 12;
     [] s=1 -> 3 : (s'=5) ;
     [] s=5 -> 5 : (s'=9)   ;
     [] s=9 -> 4.2 : (s'=5) + 20 : (s'=13) & (d'  =13)  ;
     [] s=13 -> 5.5 : (s'=9) ;
endmodule
```

Again by means of PRISM, we get the following probabilities:

• "Probability that demand of booking during high season will drop if it take more than 5 second, P5=1. "

• "Probability P_6 the request of booking during high season will drop if it take more than 10 second, P_6=1"

• "Probability the request of monitor event from provider to tenants will drop if it take less than 7 second, P_7=1"

We observed that PRISM output not satisfies the requirements under the stated assumptions on the behavior of the environment because it violated the value input for system control. Can see figure 9 analysis the different probability services request for the model levels drop *generatebooking*, drop *requesbooking*, and drop *monitorevent* for user level, tenant level and provider level respectively.

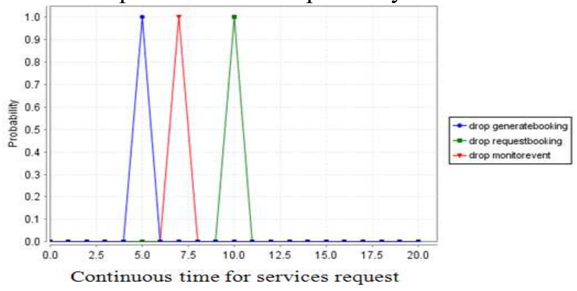


Fig 9 Analysis of Performance Requirements

Continuous detection of this violation will increase the SaaS system performance and define the range of time request for every level service of our model. In addition we can predict for what will happen for any service level. This analysis depend on monitoring specification of system environment and it choices for plan state [10].

7 Related Work

The SaaS application is hot research in recent years in [11] they Configurable business process, and isolation database to Leverage SMEs. Researchers in [12] they innovated multi-layered customization framework to Manage the variability of SaaS applications and tenants-specific requirements. Ontology is used to derive customization and deployment information for tenants cross layers. In [13] authors they used Context-oriented Programming to achieves a higher customization flexibility by single-version application code base. In [14] they Identified requirements for such a runtime architecture addressing the individual interests of all involved stakeholders. SaaS reference architecture must support at design time as well as at runtime. They Self-optimize the runtime environment according to a tenant's configuration. Authors in [15] they designed Three architectural patterns that support variability in multi-tenant SaaS environments. In [16] they depicted the design space and represent the common and variant parts of SaaS architectures. Feature model enhances the understanding of SaaS systems, and supports the architect in designing the

SaaS application architectures. The research in [17] it defined architecture by SaaS application layers for flow customization business process. Integration requirements and roadmap between SaaS application and on-premise applications are introduced in [18] by frame work architecture in SBM services. In [19] they designed Systematic approach and corresponding tool support for guiding the design of SaaS application architectures. Selected features are related to design decisions and a SaaS application architecture design is derived. They separated data servers for Tenants, separated application server, and one distribution server. They implemented architecture in [20] to define a SaaS application as an organization-based service composition and helps SaaS vendors, tenants in modeling and managing their applications. All researches didn't take autonomic management for SaaS application.

8 Conclusions

This paper has explained the way for autonomic management of QoS requirements for SaaS application. We have described the availability of SaaS application because it is a big issue in business process. To grantee the system continuous working we used the probability modeling. That it has gave our model high power in dynamic analysis for QoS requirements. By PRISM we have insured the model checker can give indicator of violation of requirements to detect and predict for what will happen. The future work will be following the autonomic management of the SaaS application for other QoS requirements in our new model.

9 Acknowledgement

This work has been developed with the support under the project with number: 2012AA02A604, 863 Program key projects in China: The Technology and the System Development for Smart Acquirement of Personal Healthcare Information. And so the Key Project of NSF in China: Methodology of Value-oriented Software Services: Theory, Method and Application with number: 61033005.

10 References

[1] N. Poggi , T. Moreno , J. L. Berral , R. Gavaldà , J. Torres. Self-adaptive utility-based web session management. Elsevier, 2008, pp. 1712–1721.

[2] N. Pogg, T. Moreno, J. L. Berral, R. Gavald, Jordi Torres. Web Customer Modeling for Automated Session Prioritization on High Traffic Sites. Springer-Verlag Berlin Heidelberg, 2007, pp. 450–454.

[3] N.Tcholtchev, A. Betkowska Cavalcante, R. Chaparadza. Scalable Markov Chain based Algorithm for Fault-Isolation in Autonomic Networks. Communications Society subject matter experts for publication, IEEE, 2010.

[4] R. Calinescu , M. Kwiatkowska. Using Quantitative Analysis to Implement Autonomic IT Systems.IEEE, 2009, pp.100-110.

[5] R. Ashok Kumar, K. Hareesh, K. Ganesan, D. H. Manjaiah. Maximizing the Availability and Reliability of Videos in VoD System Using Markov Chain. International Conference on Communication Theory, Reliability, and Quality of Service. IEEE, 2009, pp. 15-19.

[6] G. Casale, N. Mi, L. Cherkasova, E. Smirni. Dealing with Burstiness in Multi-Tier Applications: Models and Their Parameterization. IEEE Transactions oN Software Engineering, vol. 38, no 5. 2012, pp.1040-1053.

[7] A. Shrestha, L.Xing Y. Dai. Decision Diagram Based Methods and Complexity Analysis for Multi-State Systems. IEEE Transactions ON reliability, vol 59, no. 1, 2010, pp. 145-161.

[8] B. Sun, S. Zeng, R.Kang, M. G. Pecht. Benefits and Challenges of System Prognostics, IEEE Transactions on Reliability, vol. 61, NO. 2, 2012, pp. 323-335.

[9] D. Krishnamurthy, J. Rolia, M. Xu. WAM—The Weighted Average Method for Predicting the Performance of Systems with Bursts of Customer Sessions, IEEE Transactions ON Software Engineering, vol. 37, no. 5,2011, pp.718-735.

[10] M. S. Bouguerra, D. Trystram, F. ric Wagner. Brief Contributions: Complexity Analysis of Checkpoint Scheduling with Variable Costs. IEEE Transactions on Computers, vol. 62, no. 6, 2013.pp 1269-1275.

[11] Irene S. Harris, Z. Ahmed. An Open Multi-Tenant Architecture to Leverage SMEs. European Journal of Scientific Research. Vol.65 No.4, pp. 601-610, 2011.

[12] W.Tek Tsai, Q. Shao, W. Li. OIC: Ontology-based Intelligent Customization Framework for SaaS. International Conference on Service-Oriented Computing and Applications (SOCA) IEEE,2010.

[13] E. Truyen, N. Cardozo,S. Walraven, J. Vallejos, Bainomugisha, S. Gunther, T. Hondt, W. Joosen. Context-oriented Programming for Customizable SaaS Applications. Proceedings of the 27th Annual Symposium on Applied Computing ACM, 2011, pp. 418-425.

[14] Julia Schroeter, Sebastian Cech, Sebastian Götz, Claas Wilke, Uwe Aßmann. Towards Modeling a Variable Architecture for Multi-Tenant SaaS-Applications. Sixth International Workshop on Variability Modeling of Software-Intensive Systems.ACM, 2012.

[15] Jaap Kabbedijk, Slinger Jansen. Variability in Multi-tenant Environments: Architectural Design Patterns from Industry, Springer, 2011.

[16] K. Öztürk, B. Tekinerdogan, Feature Modeling of Software as a Service Domain to Support Application Architecture Design. International Conference on Software Engineering Advances. IARIA, 2011.

[17] Ralph Mietzner, Frank Leymann. Generation of BPEL Customization Processes for SaaS Applications from Variability Descriptors, International Conference on Services Computing.IEEE, 2008.

[18] A. J. Elmore, S. Das, D. Agrawal, A. El Abbadi. Towards an Elastic and Autonomic Multitenant Database. ACM, 2011.

[19] B. Tekinerdogan, K. Ö., A.Doğru, Modeling and Reasoning about Design Alternatives of Software as a Service Architectures. Ninth Working IEEE/IFIP Conference on Software Architecture.IEEE,2011.

[20] D. Concha, J. Espadas, D. Romero, A. Molina. The e-HUB evolution: From a Custom Software Architecture to a Software-as-a-Service implementation. Elsevier,2010, pp 145–151.

SESSION

BIG DATA AND DATA ANALYTICS + RELATED TECHNOLOGIES AND METHODS

Chair(s)

TBA

Hadoop MapReduce Configuration Parameters and System Performance: a Systematic Review

Ailton S Bonifacio[1], **Andre Menolli**[2], **and Fabiano Silva**[1]
[1]Department of Informatics, Federal University of Paraná, Curitiba, Paraná, Brazil
[2]Computer Science Departament, Universidade Estadual do Norte do Paraná, Bandeirantes, Paraná, Brazil

Abstract—*Hadoop MapReduce assists companies and researchers to deal with processing large volumes of data. Hadoop has a lot of configuration parameters that must be tuned in order to obtain a better application performance. However, the best tuning of the parameters is not easily obtained by inexperienced users. Furthermore, most of studies that address this issue are superficial and cover only some parameters. This paper presents a systematic review by identifying current research papers, which addresses the correlation between Hadoop configuration settings and performance. In this sense, this systematic review identified 743 papers in 5 researched databases. Applying the steps for selecting and a few exclusion criteria, the number of papers was reduced to 40. These papers were analyzed and classified according to Hadoop configuration parameters and the goals of the studies.*

Keywords: Configuration Parameters, Hadoop, MapReduce, Performance, Systematic Review

1. Introduction

MapReduce, a framework introduced by Google Inc., is a programming model designed to process large data volumes in parallel in a distributed environment (clusters) [1].

Hadoop [2] is an open-source implementation of MapReduce and currently have been one of the most widely used implementations for processing large amounts of data. Its programming model is divided into two main phases: the Map and Reduce phases. The reduce phase is divided between phases shuffle, sort and reduce. This is the intermediate phase.

A large number of companies, research centers and researchers have used Hadoop MapReduce implementations in applications such as data mining, production of reports, indexing Web pages, analysis of log files, machine learning, financial analysis, scientific simulation, statistics, research in bioinformatics and others [3]. Therefore, Hadoop has been studied to identify several aspects involving the tuning and performance.

Current studies show that the performance of Hadoop MapReduce jobs depends on the cluster configuration, input data type and job configuration settings [1], [3], [4]. Furthermore, some studies point out that most of the applications running on Hadoop, show a large difference between the behavior of the map and reduce phases. For instance, in some cases the Map phase is computationally intensive, in other is Reduce and others both [4].

This paper presents an overview of these studies focused on Hadoop configuration parameters that has influence on the job performance. The paper explores how the tuning parameter of Hadoop impacts on the entire system performance. More specifically, this research might answer the following questions:

(1) Which Hadoop configuration parameters has influences and impact on system performance?

(2) Which parameters are influenced by Hadoop phases?

(3) Which parameters are influenced by workloads characteristics?

This paper is organized as follows. In Section 2, a background on MapReduce and Hadoop framework, concepts and theories are presented. Section 3 describes the research method applied to systematic review. Results are presented in Section 4 and discussed in Section 5. Section 6 presents the final considerations this paper.

2. Background

This section presents the main concepts related to this systematic review.

2.1 MapReduce

MapReduce is a framework of distributed functional programming. Its processing occurs in two phases: Map and Reduce. MapReduce framework divides the work into a set of independent tasks and manage communications and data transfers between nodes in the cluster and the related parts of the system.

Apache Hadoop [2] is the well known and most widely used implementation of MapReduce. However, the improvement of application performance is directly related to the setting of the Hadoop's parameters. To ascertain the relationship between parameter values and a good performance is not a simple task, even for experienced users.

2.2 Hadoop

The Apache Hadoop software library allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage.

Hadoop is a free, Java-based programming framework that supports the processing of large data sets in a distributed computing environment. It is part of the Apache project sponsored by the Apache Software Foundation.

2.2.1 Execution Parameters

In order to run a program such as a job in Hadoop, an object configuration job is created and the parameters of the job must be specified. That is, since the job was created for MapReduce, it must enabled to run in scale on a cluster [4]. Therefore, in order to avoid some problems such as under-performance, for example, it is necessary to tuning some parameters.

Hadoop provides many configurable parameters that lead to better or worse performance on a cluster. Most of these parameters take effect on execution of the job and cluster. In the case of large clusters of machines, these parameters become quite important and provide an indication of the performance of this cluster in relation to the submitted job. Thus, for better system performance, the framework must be further optimized as is possible.

Hadoop has over 190 parameters that can be specified to control the behavior of a MapReduce job. Of those, more than 25 parameters can impact the performance of the task [5]. Some configuration parameters aim to control various aspects of the behavior of the task at runtime. Some of these aspects include the allocation and use of memory, competition, optimize I/O and network bandwidth usage. If the parameters are not specified, default values are assumed.

3. Research Method

One method that has gained significant popularity in the various area of research is the Systematic Literature Review (SLR or systematic review). Specifically in Big Data, few systematic reviews are found. Therefore, it is a field to be explored. More specifically, we found no systematic review about performance of Hadoop systems related to their configuration parameters and tuning. In this sense, a review on this subject in a systematic and rigorous manner, will be valuable for research and practice.

SLR is a rigorous methodological review of the research results. Besides adding the existing evidence on a specific research topic, an SLR is also intended to support the development of guidelines and consensus for future studies, based on evidence from several previous studies. Therefore, a systematic review aims to identify, evaluate and interpret relevant research about a particular issue, a thematic area or phenomena of interests using an explicit and rigorous method [6].

Among the several grounds for make a systematic review[6], we highlight for this work:

- Identify the directions that current research on the subject in a matter are taking;
- Provide a background to the new researchers in the field in order to properly position them regarding what has been done and the consensus of the community.

A systematic review involves several distinct activities, which according to [7], summarizes the stages of the systematic review in three main phases: planning, conducting and reporting the review. The next subsections detail the steps taken to produce this study.

3.1 Planning the review

The planning of the systematic review started by developing a protocol. This protocol specified the process and methods that would be applied. The protocol specified the research questions, the research strategy and the criteria for inclusion and exclusion. This systematic review aims to provide an overview of studies which apply to performance of Hadoop systems in regard to their configuration parameters, answering the research questions mentioned in Section 1.

3.2 Conducting the review

Once the protocol has been agreed, the steps for conducting the review are presented below.

3.2.1 Research Identification

A systematic review aims to find the largest possible number of primary studies related to the research topic. Thus, the first step is the identification of keywords and search terms. In this study, the keywords and terms have been used in order to get the relevant work in a broader way, in relation to the universe of this study [8]. All possible permutations of the words "Hadoop" and "parameters" were considered. Moreover, we varied the terms relating to "performance" (see Table 1). For example, the first search string was: "Hadoop", "Performance Tuning" and "Parameter". All search terms were searched in full-text, title, and keywords of all databases listed in Table 2. These databases were chosen because of their relevance in the scientific community [6], and for the indexing of the most important conferences of parallel and distributed computing, workshops and journals. As the objective of this study is to understand the performance of Hadoop framework by setting the its parameters, and this is a relatively new research topic, only studies of 2008 onwards have been searched. The research was carried out in October 2013, which means that the publications of the end of 2013 until now may not have been indexed by the databases. The identification process produced 1181 papers. This formed the basis for the next step.

3.2.2 Selection of Primary Papers

The first step after the preliminary identification of papers was the elimination of duplicate titles. The execution of this

Table 1: Terms for the search

Terms about Performance
Performance Tuning
Performance Analysis
Performance Prediction
Performance of MapReduce
Performance Model
Hadoop Performance
Job Performance
Self-Tuning

Table 2: Databases researched in the systematic review

Researched Databases
ACM Digital Library
IEEE Xplore Digital Library
Science Direct
Springer Link
Scopus (Elsevier)

step reduced the set for 743 papers. The next step was to remove the titles clearly not related to the review. This step reduced the set for 296 papers. Then the abstracts were read and evaluated, with the following exclusion criteria:

- exclude papers that were clearly not of the hadoop system area or not applied to hadoop performance or job performance,
- exclude papers that were not related to the terms of performance tuning, analysis, prediction, model or self-tuning,
- exclude papers that were not related to the tuning parameters or configuration as well as their implications on performance,
- exclude papers that were just of literature review, as example, systematic review, and,
- exclude papers without validation method, if it were not clearly a practical study.

After the execution of this process, the set of papers was reduced to 124. The full text for all 124 papers was obtained and read with the same exclusion criteria. The final number of papers selected for the review was 40, which clearly fitted to the criteria defined for accomplishment of the systematic research, which were then analyzed.

3.2.3 Study quality assessment and classification

After reading and analyzing the selected papers, the next step was to classify them according to parameters studied for each paper. Due to lack of specific literature on the topic discussed in this systematic review, the classification of parameters was based on the definitions and classifications adopted in the book Pro Hadoop [4], which is one of the most cited references in the analyzed studies and based on our own understanding since there are no classifications ratified by the scientific community. In this systematic review, the purpose of the classification parameters, is answer the questions raised in Section 1 and show which parameters have been identified and studied in the works that deal with the performance of Hadoop.

3.2.4 Data Synthesis

For the synthesis, were extracted the Hadoop configuration parameters that has influence on the performance, the central objectives and the experimentation methods.

4. Results

The results of this study are divided into three main sections, aiming to answer the initial questions. Section 4.1 focuses on identifying parameters which were used in the detected studies, Section 4.2 focuses on which parameters are affected by each Hadoop phase and Section 4.3 focuses on identifying which parameters are related to the workloads characteristics.

The Table 3 summarizes the parameters. The classification used in the table answers the first three questions of this systematic review. Discussions about it follow in the next sections.

4.1 Configuration Parameters Results

According to the classification adopted, we identified all parameters that somehow influences system performance and which have been studied in the papers that we have identified.

About 29 configuration parameters, which according to studies analyzed impact on system performance, were identified. Figure 1 shows a graph that points out the number of papers by parameters. More details in Section 5.

4.2 Hadoop Phases x Parameters Results

The execution of a MapReduce job is divided into a Map phase and a Reduce phase. The data output by each map task is written into a circular memory buffer. When this buffer reaches a threshold, its content is sorted by key and flushed to a temporary file. These files are then served via HTTP to machines running reduce tasks. Reduce tasks are divided into three sub-phases: shuffle, sort and reduce. The shuffle sub-phase copies the data output from the map nodes to the reducer's nodes. The sort sub-phase sorts the intermediate data by key. Finally, the reduce sub-phase, which runs the job's reduce() function and the final result is then written to the distributed file system [47].

Studies point out that most of the applications running on Hadoop, show a large difference between the behavior of the map and reduce phases. Therefore, it is very important know which parameters must be tuned according to the processing phases of an application on the Hadoop. In Table 3, in the column *Parameter Phase*, we classify the parameters which influence the phases of Hadoop: Map, Reduce and intermediate sub-phases of the Reduce phase. The sub-phases are classified as Merge/Shuffle phase. In

Table 3: Configuration Parameters

Parameter Category	Parameter Level	Parameter Phase	Workload Characteristics	Parameter	Scientific Papers
Hadoop	Cluster Level	Merge/ Shuffle	IO	dfs.replication	[9], [10], [11], [5], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21]
				dfs.replication.interval	[9], [10], [11], [14], [15], [16]
				mapred.min.split.size	[22], [23]
			Number of maps	dfs.block.size	[9], [10], [11], [5], [24], [12], [25], [13], [14], [15], [16], [26], [27], [28], [29], [30], [31], [32], [17], [18], [19], [33], [20], [34], [22], [35]
	Job Level	Core Job	CPU	mapred.compress.map.output	[12], [29], [17], [22], [36], [37], [3], [23]
				mapred.job.reuse.jvm.num.tasks	[20], [22], [23]
				mapred.output.compression.type	[17], [22], [23]
				mapred.reduce.slowstart.completed.maps	[18], [21], [36], [37], [3], [23]
			Memory	mapred.child.java.opts	[5], [13], [16], [17], [22], [23]
			Network	mapred.reduce.parallel.copies	[17], [22], [38], [19]
		Map	CPU	mapred.map.tasks.speculative.execution	[18], [23]
				mapred.tasktracker.map.tasks.maximum	[9], [10], [11], [12], [13], [14], [15], [16], [29], [17], [18], [34], [22], [35], [23], [39], [40]
		Merge/ Shuffle	CPU	mapred.map.output.compression.codec	[17], [22], [23]
			IO	io.file.buffer.size	[9], [10], [11], [5], [14], [15], [16], [29], [20]
				io.sort.factor	[9], [10], [11], [5], [13], [14], [15], [16], [32], [17], [18], [19], [20], [36], [37], [3], [38], [23]
				io.sort.mb	[9], [10], [11], [5], [13], [14], [15], [16], [17], [22], [36], [37], [3], [38], [23], [41]
				io.sort.record.percent	[9], [10], [11], [5], [13], [14], [15], [16], [22], [36], [37], [3], [38], [23], [41]
				io.sort.spill.percent	[9], [10], [11], [14], [15], [16], [29], [20], [34], [22], [36], [37], [3], [38], [23]
			Memory	mapred.inmem.merge.threshold	[5], [16], [18], [22], [36], [37], [3], [23]
				mapred.job.reduce.input.buffer.percent	[5], [16], [22], [36], [37], [3], [38], [23]
				mapred.job.shuffle.input.buffer.percent	[5], [16], [29], [22], [36], [37], [3], [38], [23]
				mapred.job.shuffle.merge.percent	[5], [16], [18], [36], [37], [3], [38], [23]
		Reduce	CPU	mapred.reduce.tasks.speculative.execution	[23]
				mapred.tasktracker.reduce.tasks.maximum	[9], [10], [11], [12], [13], [14], [15], [16], [29], [17], [18], [34], [22], [35], [23], [39], [40]
			Memory	mapred.tasktracker.tasks.maxmemory	[42]
Workload	Job Level	Map	CPU	mapred.map.tasks	[9], [10], [11], [24], [12], [14], [15], [16], [26], [27], [30], [31], [19], [33], [20], [34], [22], [35], [43], [44], [42], [45], [46]
			IO	mapreduce.combine.class	[29], [36], [37], [3]
				min.num.spills.for.combine	[36], [37], [3], [23]
		Reduce	CPU; IO	mapred.reduce.tasks	[9], [10], [11], [5], [24], [12], [25], [14], [15], [16], [26], [27], [28], [30], [31], [32], [19], [33], [20], [34], [22], [35], [36], [37], [3], [38], [23], [43], [44], [42], [46]

addition, Core Job was listed as a phase that represents those parameters that directly control essential functions of the job. The parameter *dfs.block.size* was classified by phase *Number of maps* just to demonstrate that the value this parameter is what defines the number of map tasks.

4.3 Workloads Characteristics x Parameters Results

Optimizing Hadoop Performance through the characteristics of workloads are important in order to make full use of that cluster resources (CPU, memory, IO and network - see Table 3, in the column *Workload Characteristics*).

Among other examples, set certain parameters can reduce the IO cost and network transmission, but can cause a CPU overhead. This is an example if we configure the *mapred.compress.map.output* to parameter to true (default is false), it will decrease the size of data transferred from the Mapper to the Reducers, but will add more processing in the data compression and decompression process [4]. Furthermore, some parameters are correlated with each other. The relation between the *io.sort.mb* parameter (the amount of buffer space in megabytes to use when sorting streams) and *mapred.child.java.opts* parameter (Java control options for all mapper and reducer tasks) is an example. The upper limit of the former is smaller than the second size. Configure sufficient Map and Reduce slots to maximize CPU utilization and configure the Java heap size (for Map and Reduce JVM processes) so that ample memory is still available for the OS kernel, buffers, and cache subsystems are other examples [23].

Workload characterization by CPU, memory, IO and network (via benchmarks) is essential before performing

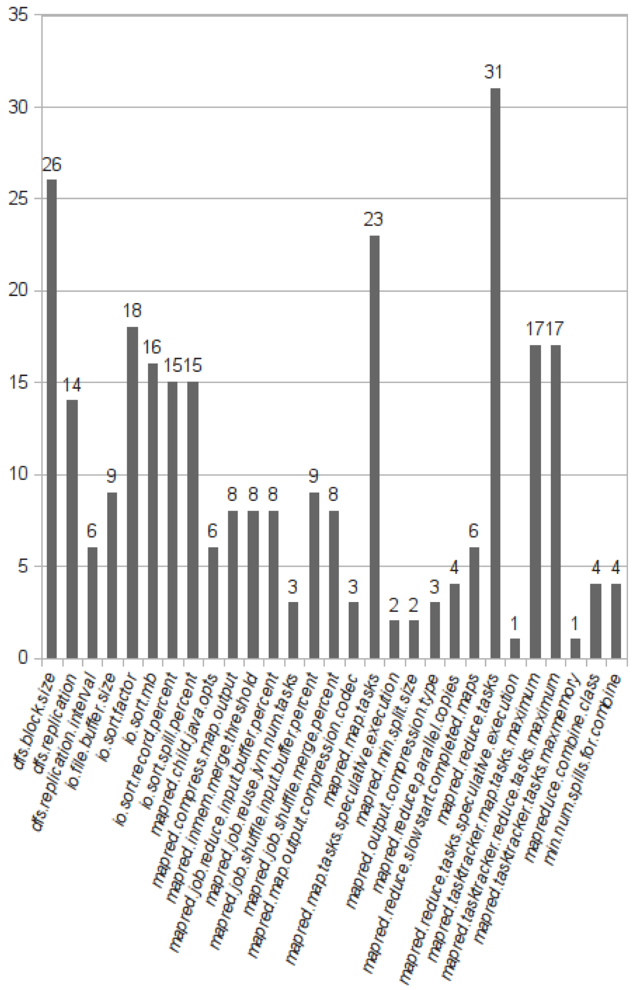


Fig. 1: Number of Scientific Papers per Parameter

any tuning parameters. Furthermore, it is important that we deploy the parameters are related to the characteristics of workloads. This allows identifying the parameters that can be set to obtain best performance.

5. Results Analysis

This section discusses the results of this systematic review and based on the answers from 3 initial questions. Furthermore, the results obtained from our study identified the main purpose of the studies analyzed, and the methods of experimentation and validation used in each study.

5.1 Questions Considerations

Answering the first question, it is clear that about 29 Hadoop configuration parameters are those which more impact on system performance. From these 29 parameters, 10 parameters were covered over 65% of the papers. We observed that the most discussed parameters on these works are *mapred.reduce.tasks* (The suggested number of reduce tasks for a job) with 31 papers of the 40 papers surveyed, mapred.map.tasks (The suggested number of map tasks for a job) with 23 papers, *dfs.block.size* (The basic block size for the file system) with 26 papers and *io.sort.factor* (The number of map output partitions to merge at a time) with 18 papers. These parameters are those that allow the framework to attempt to provide data locally for the task that processes the split. Although some parameters have not been widely exploited by most papers, studies have shown their importance in relation to performance.

Answering the second question, we identify the parameters that are affected by each Hadoop phase. This will also allow the targeting of tuning the parameters identified at each stage of Hadoop if this is required. This approach would be rather useful if we know the application characteristics and at what phase it is CPU, IO or network bound, for example. Thus, the tuning of the parameters would be directed by this prior knowledge.

Answering the third question, we can observe the parameters are impacted according to workloads characteristics. Thus, it is possible, in future work, observe the workloads characteristics and working with the tuning of Hadoop parameters targeted at them.

5.2 Purpose and Experimentation Approach Results

Related work in this systematic review have had in common the exploration of configuration parameters and performance of Hadoop. However, we note that the studies have different core purposes, as well as different methods of experimentation. Thus, for better understanding, we classified the papers according to the main purpose and the methods applied.

For classifying the experimentation method, it was used the taxonomy proposed by Zelkowitz and Wallace [48]. In this taxonomy there are four approaches toward experimentation: (1) Scientific Method, (2) Engineering Method, (3) Empirical Method and (4) Analytical Method.

Table 4 shows the main purpose of each work and the main experimentation methods found in the analyzed papers. Importantly, the identification of the method used in the analyzed studies are not always as clear to identify. Some studies show a twofold interpretation. Thus, even with dubious features, in some cases, we classify according to the predominant method.

We observed that many studies focused between the purpose of obtaining performance models (about 22.5%) and performance prediction (37.5%). Other were divided into performance analysis and tuning performance, and only one job to prediction energy and performance penalties. Furthermore, we observed that 67.5% of the analyzed papers have used the empirical approach as experimentation method, and 32.5% have used the analytical method.

Most studies have adopted an empirical approach in experimental methods. This means that proposed a hypothesis and

Table 4: Purpose and Experimentation Approach Results

Purpose	**Approach**	**Scientific Papers**
Energy Prediciton	Analytical	[13]
Performance Analysis	Empirical	[32], [43], [12], [26], [21], [15]
Performance Model	Empirical	[18], [40], [33]
	Analytical	[44], [19], [16], [35], [30], [25]
Performance Penalties	Empirical	[14]
Performance Prediction	Empirical	[24], [11], [42], [37], [3], [28], [38], [9], [10], [29]
	Analytical	[36], [41], [46], [31], [17]
Performance Tuning	Empirical	[5], [22], [23], [20], [34], [27], [39]
	Analytical	[45]

validated by a method of statistical analysis. In other words, data were collected and statistically verified the hypothesis. On the other hand, some studies have utilized the analytical method. These studies have developed a formal theory to a problem, and results derived from this theory were compared with empirical observations.

5.3 Validation Methods

A further analysis was achieved through the researched information by this systematic review is about the validation methods used by the studies. Validation methods were analyzed in the same manner as the experimentation methods and were classified according to the taxonomy for software engineering experimentation [49].

The studies may be classified into three of the twelve methods considered in the taxonomy. It is noteworthy that once again, here, papers with dubious interpretations. Thus, the validation methods with predominant features were classified. It is possible to observe in Table 5 that most of the works used mainly dynamic analysis (benchmark) and simulation to validate their researches. As can be seen, the controlled dynamic analysis method is the method more used in researches projects and formal theoretical analysis method is the least used method.

Table 5: Validation Results

Validation Type	**Validation Method**	**Scientific Papers**
Controlled	Dynamic Analysys (Benchmark)	[18], [24], [5], [45], [22], [36], [42], [37], [3], [19], [28], [40], [32], [43], [12], [26], [41], [38], [13], [23], [21], [34], [27], [46], [31], [30], [39], [25], [29], [17], [33]
	Simulation	[11], [14], [20], [35], [9], [10], [15]
Formal	Theoretical Analysis	[44], [16]

6. Final Considerations

A systematic review on the relationship of the Hadoop configuration parameters and system performance was presented. The goal was a study to review the most recent research into this context and answer the following questions: (1) Which Hadoop configuration parameters has influences and impacts on system performance? (2) Which parameters are influenced by workloads characteristics? (3) Which parameters are influenced by Hadoop phases?

The results achieved were able to answer these questions and provide references for future works directed to its subject matter. Besides the expected answers, the results analysis identified the main purpose and the experimentation and validation methods approached by the examined papers. Once again, with great importance for reference in future works.

References

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," in *Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation - Volume 6*, ser. OSDI'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 10–10.

[2] Apache, "Apache hadoop," Oct 2012, october 24, 2012. [Online]. Available: {http://hadoop.apache.org/}

[3] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu, "Starfish: A self-tuning system for big data analytics," in *In CIDR*, 2011, pp. 261–272.

[4] J. Venner, "Tuning your mapreduce jobs," in *Pro Hadoop*. Apress, 2009, pp. 177–206.

[5] S. Babu, "Towards automatic optimization of mapreduce programs," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 137–142.

[6] B. Kitchenham, "Procedures for performing systematic reviews," Department of Computer Science, Keele University, UK, Technical Report TR/SE-0401, 2004.

[7] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering - a systematic literature review," *Information and Software Technology*, vol. 51, no. 1, pp. 7 – 15, 2009.

[8] A. Menolli, S. Reinehr, and A. Malucelli, "Organizational learning applied to software engineering: A systematic review," *International Journal of Software Engineering and Knowledge Engineering*, vol. 23, no. 08, pp. 1153–1175, 2013.

[9] G. Wang, A. Butt, P. Pandey, and K. Gupta, "A simulation approach to evaluating design decisions in mapreduce setups," in *MASCOTS '09. IEEE International Symposium on*, Sept 2009, pp. 1–11.

[10] G. Wang, A. R. Butt, P. Pandey, and K. Gupta, "Using realistic simulation for performance analysis of mapreduce setups," in *Proceedings of the 1st ACM Workshop on Large-Scale System and Application Performance*, ser. LSAP '09. New York, NY, USA: ACM, 2009, pp. 19–26.

[11] S. Hammoud, M. Li, Y. Liu, N. Alham, and Z. Liu, "MRSim: A discrete event based mapreduce simulator," in *Fuzzy Systems and Knowledge Discovery (FSKD), 2010 Seventh International Conference on*, vol. 6, Aug 2010, pp. 2993–2997.

[12] M. Koehler, Y. Kaniovskyi, and S. Benkner, "An adaptive framework for the execution of data-intensive mapreduce applications in the cloud," in *Proceedings of the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, ser. IPDPSW '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 1122–1131.

[13] W. Li, H. Yang, Z. Luan, and D. Qian, "Energy prediction for mapreduce workloads," in *Proceedings of the 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, ser. DASC '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 443–448.

[14] S. Kadirvel and J. A. B. Fortes, "Towards self-caring mapreduce: Proactively reducing fault-induced execution-time penalties," in *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, July 2011, pp. 63–71.

[15] G. Wang, A. R. Butt, H. Monti, and K. Gupta, "Towards synthesizing realistic workload traces for studying the hadoop ecosystem," in *19th IEEE Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. Raffles Hotel, Singapore: IEEE Computer Society, Jul. 2011, pp. 400–408.

[16] X. Lin, Z. Meng, C. Xu, and M. Wang, "A practical performance model for hadoop mapreduce," in *Proceedings of the 2012 IEEE International Conference on Cluster Computing Workshops*, ser. CLUSTERW '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 231–239.

[17] H. Yang, Z. Luan, W. Li, D. Qian, and G. Guan, "Statistics-based workload modeling for mapreduce," in *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, ser. IPDPSW '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 2043–2051.

[18] F. Ahmad, S. T. Chakradhar, A. Raghunathan, and T. N. Vijaykumar, "Tarazu: Optimizing mapreduce on heterogeneous clusters," *SIGARCH Comput. Archit. News*, vol. 40, no. 1, pp. 61–74, Mar. 2012.

[19] J. Han, M. Ishii, and H. Makino, "A hadoop performance model for multi-rack clusters," in *Computer Science and Information Technology (CSIT), 2013 5th International Conference on*, March 2013, pp. 265–274.

[20] Y. Liu, M. Li, N. K. Alham, and S. Hammoud, "HSim: A mapreduce simulator in enabling cloud computing," *Future Gener. Comput. Syst.*, vol. 29, no. 1, pp. 300–308, Jan. 2013.

[21] T. D. Plantenga, Y. R. Choe, and A. Yoshimura, "Using performance measurements to improve mapreduce algorithms," *Procedia Computer Science*, vol. 9, no. 0, pp. 1920 – 1929, 2012.

[22] D. Heger, "Hadoop performance tuning - a pragmatic & iterative approach," *CMG Journal*, 2013.

[23] X. Lin, W. Tang, and K. Wang, "Predator - an experience guided configuration optimizer for hadoop mapreduce," in *CLOUDCOM '12*. Washington, DC, USA: IEEE Computer Society, 2012, pp. 419–426.

[24] M. An, Y. Wang, and W. Wang, "Using index in the mapreduce framework," in *Proceedings of the 2010 12th International Asia-Pacific Web Conference*, ser. APWEB '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 52–58.

[25] X. Yang and J. Sun, "An analytical performance model of mapreduce," in *Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on*, Sept 2011, pp. 306–310.

[26] M. Koehler and S. Benkner, "Design of an adaptive framework for utility-based optimization of scientific applications in the cloud," in *Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on*, Nov 2012, pp. 303–308.

[27] A. Raj, K. Kaur, U. Dutta, V. Sandeep, and S. Rao, "Enhancement of hadoop clusters with virtualization using the capacity scheduler," in *Services in Emerging Markets (ICSEM), 2012 Third International Conference on*, Dec 2012, pp. 50–57.

[28] S. Kadirvel and J. A. B. Fortes, "Grey-box approach for performance prediction in map-reduce based platforms," in *Computer Communications and Networks (ICCCN), 2012 21st International Conference on*, July 2012, pp. 1–9.

[29] H. Yang, Z. Luan, W. Li, and D. Qian, "Mapreduce workload modeling with statistical approach," *J. Grid Comput.*, vol. 10, no. 2, pp. 279–310, Jun. 2012.

[30] D. Tiwari and D. Solihin, "Modeling and analyzing key performance factors of shared memory mapreduce," in *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, May 2012, pp. 1306–1317.

[31] N. B. Rizvandi, J. Taheri, R. Moraveji, and A. Y. Zomaya, "On modelling and prediction of total CPU usage for applications in mapreduce environments," in *ICA3PP'12*. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 414–427.

[32] N. Khoussainova, M. Balazinska, and D. Suciu, "PerfXplain: debugging mapreduce job performance," *Proc. VLDB Endow.*, vol. 5, no. 7, pp. 598–609, Mar. 2012.

[33] Z. Zhang, L. Cherkasova, and B. T. Loo, "Benchmarking approach for designing a mapreduce performance model," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '13. New York, NY, USA: ACM, 2013, pp. 253–258.

[34] W. Premchaiswadi and W. Romsaiyud, "Optimizing and tuning mapreduce jobs to improve the large-scale data analysis process," *Int. J. Intell. Syst.*, vol. 28, no. 2, pp. 185–200, Feb. 2013.

[35] A. Nunez and M. G. Merayo, "A formal framework to analyze cost and performance in map-reduce based applications," *Journal of Computational Science*, no. 0, pp. –, 2013.

[36] H. Herodotou, F. Dong, and S. Babu, "Mapreduce programming and cost-based optimization? Crossing this chasm with starfish." *PVLDB*, vol. 4, no. 12, pp. 1446–1449, 2011.

[37] H. Herodotou and S. Babu, "Profiling, What-if Analysis, and Cost-based Optimization of MapReduce Programs," *PVLDB: Proceedings of the VLDB Endowment*, vol. 4, no. 11, pp. 1111–1122, 2011.

[38] P. Lama and X. Zhou, "AROMA: Automated resource allocation and configuration of mapreduce environment in the cloud," in *Proceedings of the 9th International Conference on Autonomic Computing*, ser. ICAC '12. New York, NY, USA: ACM, 2012, pp. 63–72.

[39] K. Wang, B. Tan, J. Shi, and B. Yang, "Automatic task slots assignment in hadoop mapreduce," in *Proceedings of the 1st Workshop on Architectures and Systems for Big Data*, ser. ASBD '11. New York, NY, USA: ACM, 2011, pp. 24–29.

[40] K. Kambatla, A. Pathak, and H. Pucha, "Towards optimizing hadoop provisioning in the cloud," in *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing*, ser. HotCloud'09. Berkeley, CA, USA: USENIX Association, 2009.

[41] R. R. Kompella, Y. C. Hu, and D. Xie, "On the performance projectability of mapreduce," in *Proceedings of the 2012 IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*, ser. CLOUDCOM '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 301–308.

[42] H. Herodotou, F. Dong, and S. Babu, "No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics," in *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, ser. SOCC '11. New York, NY, USA: ACM, 2011, pp. 18:1–18:14.

[43] K. Kim, K. Jeon, H. Han, S.-g. Kim, H. Jung, and H. Y. Yeom, "MRBench: A benchmark for mapreduce framework," in *Proceedings of the 2008 14th IEEE International Conference on Parallel and Distributed Systems*, ser. ICPADS '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 11–18.

[44] M. Elteir, H. Lin, and W. chun Feng, "Enhancing mapreduce via asynchronous data processing." in *ICPADS*. IEEE, 2010, pp. 397–405.

[45] Z. Guo, G. Fox, M. Zhou, and Y. Ruan, "Improving resource utilization in mapreduce." in *CLUSTER*. IEEE, 2012, pp. 402–410.

[46] N. B. Rizvandi, J. Taheri, R. Moraveji, and A. Y. Zomaya, "Network load analysis and provisioning of mapreduce applications," in *PDCAT '12*. Washington, DC, USA: IEEE Computer Society, 2012, pp. 161–166.

[47] F. Tian and K. Chen, "Towards optimal resource provisioning for running mapreduce programs in public clouds," in *Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing*, ser. CLOUD '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 155–162.

[48] M. V. Zelkowitz and D. R. Wallace, "Experimental models for validating technology," *Computer*, vol. 31, no. 5, pp. 23–31, May 1998.

[49] M. V. Zelkowitz, D. R. Wallace, and D. W. Binkley, "Lecture notes on empirical software engineering," N. Juristo and A. M. Moreno, Eds. River Edge, NJ, USA: World Scientific Publishing Co., Inc., 2003, ch. Experimental Validation of New Software Technology, pp. 229–263.

PDTSSE: A Scalable Parallel Decision Tree Algorithm Based on MapReduce

Yan Cui*†, Yuanyang Yang*, Shizhong Liao*‡
*School of Computer Science and Technology
Tianjin University, Tianjin 300072, P. R. China,
†Department of Common Required Courses
Tianjin University of Traditional Chinese Medicine, Tianjin 300193, P. R. China
‡Corresponding author: szliao@tju.edu.cn

***Abstract*—Parallel decision tree learning is an effective and efficient approach to scaling the decision tree to large data mining application. Aiming at large scale decision tree learning, we present a novel parallel decision tree learning algorithm in MapReduce framework, called PDTSSE (Parallel Decision Tree via Sampling Splitting points with Estimation). We first propose an estimation method for sampling splitting points, which can effectively handle both categorical and numeric attributes over large scale data. We also derive an error bound for the algorithm, and analyze the computational complexities of the algorithm. Finally, we describe the implementation procedures in MapReduce framework. Theoretical analysis and experimental results show that PDTSSE has low computational cost compared to state of the art classifiers while maintaining the quality of the generated trees in terms of accuracy, and can scale to large scale data mining application.**

Keywords-Classification; Decision Tree; Sampling; Parallel, MapReduce

I. Introduction

Decision trees are elementary and effective classifiers. They have been used widely for data mining because of their high efficiency and accuracy, good readability and robustness when compared with other classification methods [1], [2]. Decision tree algorithms retain integral dataset in memory and sort all numerical attributes to decide which is used to split child nodes. Memory-resident and sorting result in high requirements on running time and memory storage. These requirements are two of most significant drawbacks of decision tree algorithms.

Techniques improving these drawbacks include discretization, sampling [3] and parallelization [4]. Catlett [3] proposed a method that samples at each node of the classification tree, but this study didn't consider datasets that were too large to fit in main memory. Even more unfortunately, discretization and sampling cause significant losses in accuracy and lack of reliable [5]. Chan and Stolfo [6] proposed a method that partitioned the dataset such that each subset fitted in main memory. However, their results showed that the performance of resulting decision tree was decreased. Alsabti [7] presented a decision tree classifier for large datasets called CLOUDS. CLOUDS reduced the loss of accuracy and narrowed the search space of the best split through sampling the splitting points and an estimation step for numeric attributes. Since most of the decision tree algorithms are designed only for memory-resident data, much work has been dedicated to improve them. SLIQ [8], SPRINT [9] and ScalParC [10] performed pre-sorting to increase efficiency when computing the best numeric split for each numeric attribute at the beginning of tree growth. BOAT [11], CLOUDS [7] and SPIES [12] replaced sorting with approximate representations of the training data. Although pre-sorting and approximate techniques resulted in more accurate, they could not accommodate very large data sets. RainForest [2] significantly improved performance over the SPRINT classification algorithm by adapting the size of dataset to the amount of main memory. Parallel decision tree algorithms were studied over these years [4], [13] as well. SPDT [14] was a parallel decision tree algorithm which was scalable for streaming data. Most parallel decision trees were constructed in a breath-first mode [8], [10], [14], [15] because it balanced loads better.

In this work, we propose a novel parallel decision tree algorithm namely PDTSSE based on MapRecude framework to improve these drawbacks. The rest of the paper is structured as follows. Section II gives a detailed description of the PDTSSE and its scalable algorithms implemented based on Hadoop platform. We give the complexity analysis and error bound of PDTSSE in the Section III. In Section IV, we present experimental results from a detailed performance evaluation of PDTSSE algorithm and dwell upon the advantages of PDTSSE over existing methods. Finally, we conclude in Section V.

II. PDTSSE Algorithm

In this section, we initially give detailed description and design of the PDTSSE algorithm. Two critical procedures under the framework of MapReduce that handle splitting nodes and growing subtrees are given as well. We also describe the entire tree induction process in the Hadoop environment, which is a successful implementation of MapReduce [16].

A. Algorithm Design

Consider the following problem: given a very large training set D which is much larger than main memory, each instance in D has d dimensions (i.e. $\mathbf{x_i} \in R^d, i \in \{1, 2, \dots, |D|\}$) with label $y_i \in \{1, \dots, c\}$. Our goal is to construct a decision tree

Submission to PDPTA'14, Email to first author: cuiyan_tju@126.com, second author: yyyang@tju.edu.cn

based on the training set with multiple processors in a parallel distributed environment. In parallel decision tree algorithms, the primary problems remain finding good splitting points and partitioning the training set to generate new nodes. Considering the characteristics of various parallel methods, PDTSSE uses a breath-first strategy with hybrid parallelism to build the decision tree. Moreover, in order to reduce running time and computation cost while achieving high predictive accuracy and scalability, the following improvements are made: First, SSE method is employed to derive the best splitters among all alive intervals for each internal node of the tree. After SSE, domain sizes of the numeric attributes don't have any impact on the performance of our algorithm. Second, for the reason that the quality of split for a numeric attribute can be computed independently, we evaluate it in parallel environment based on gini index after one scan over the data set. Last, the main data structures used in PDTSSE are the distributed count matrices which contain summary statistics computed from training set.

Before tree induction, same as SPRINT and ScalParC algorithms, PDTSSE sorts the values of numeric attributes only once. The difference is that only numeric attribute lists are split separately in the following step. Every attribute list contains attribute value and class value associated with every record, and then we run a MapReduce on training data and compute approximate equi-depth histograms for every numerical attribute to get all SS split points. For each categorical attribute, there is a count matrix associated with it, for each numeric attribute, a count matrix is associated with every sampling split. Every count matrix contain the frequency of each class in each partition, the value of gini index for each split can be calculated from them. To minimize bookkeeping and communication cost, PDTSSE passes a hash table of entire training dataset.

A controller lies at the kernel of PDTSSE, The Controller manages the entire tree induction process using a set of MapReduce jobs, each of which builds different parts of the tree at the same time. The controller maintains the components as following:

- **ModelFile(M):** In initial state, it's empty. At any point, the model file contains the complete classification tree constructed so far.
- **MRExpandQueue(MRQ):** This queue contains nodes to be extended to which the input D is greater than a given threshold or too large to fit in memory.
- **InMemoryQueue(IMQ):** This queue contains nodes to be extended to which the input D is less than a given threshold or fits in memory completely.

As tree induction proceeds, the Controller dequeues nodes off MRQ and schedules jobs to find split predicates for the expanding nodes. Each job takes as input a set of nodes S, the training set D^* and the current state of the model M. Once the split predicates are determined, the nodes in S are expanded, and then the Controller will update model M with S and their split predicates, MRQ and IMQ are updated with new child nodes according to the node size at the end of this step.

The PDTSSE algorithm breaks up the process of constructing a decision tree into a set of MapReduce tasks which can be divided into two different type of MapReduce jobs according to the different stages of tree building. Dependencies exist between the two different tasks.

- **MR_ExpandNodes :** This job is responsible for collecting summary statistics and computing a set of candidate splits for nodes in MRQ during tree building, after then, it will expand tree in parallel.
- **MR_InMemBuildNodes :** This job completes tree induction for nodes in IMQ with task parallelism.

B. Controller Design

The Controller schedules two types of jobs containing ScheduleMR_ExpandNode and ScheduleMR_InMemory to run repeatedly until all the queues are all empty and none of the jobs it schedules remain running.

Algorithm 1 CONTROLLER

```
Input: Training Data $D$, Model M=$\emptyset$, IMQ=$\emptyset$, MRQ=$\emptyset$
 1: MRQ.append(root)
 2: while MRQ not empty do
 3:     ExpandNodeSet $S$ = getExpandNodes(MRQ,M)
 4:     ScheduleMR_ExpandNode(S)
 5: end while
 6: while IMQ not empty do
 7:     ScheduleMR_InMemory(IMQ,M)
 8: end while
 9: if MRQ empty and IMQ empty and jobs finished then
10:     Exit
11: end if
```

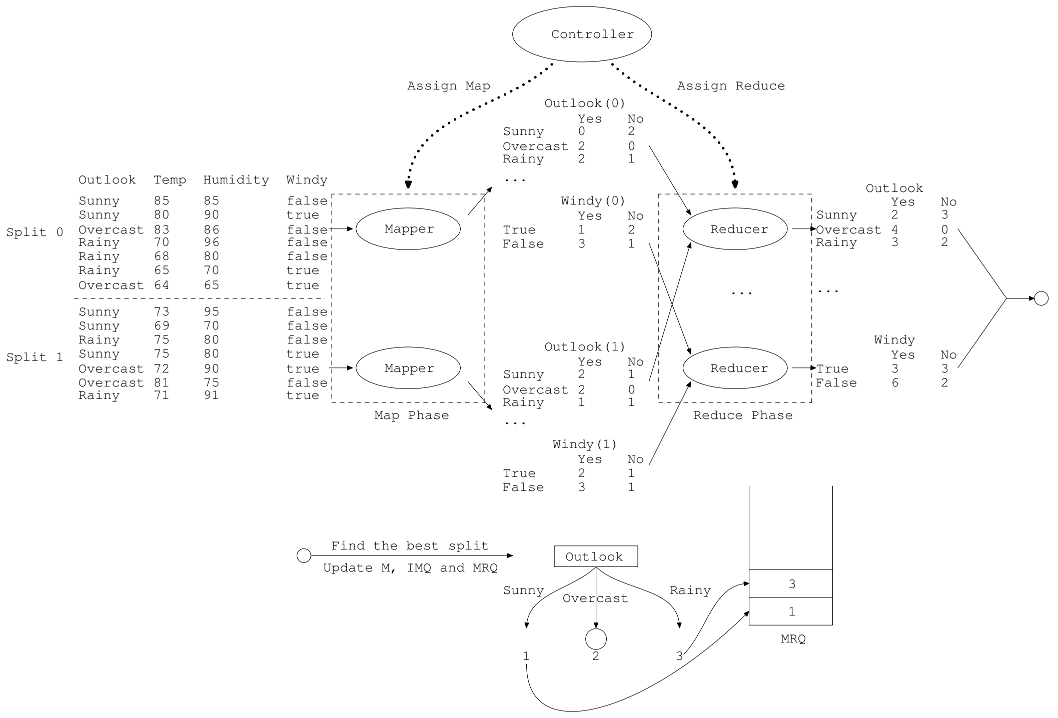
C. MR_ExpandNodes: Expanding Nodes In Parallel

MR_ExpandNodes is the kernel component that allows PDTSSE to train on datasets that are too large to fit in memory. Given a set of nodes S at one level of the tree, the training dataset D^*, and the current tree model M, this job computes a set of good splits for each node in S and generates new child nodes. It mainly has two important steps in processing: Summary Statistics Process and SubTree Building Process.

1) Summary Statistics Process: In the Summary Statistics process, Each mapper maintains two tables $T_{\{n,a,v\}}[c]$, $T_{\{n,a,s,l\}}[c]$ for categorical and numeric attributes respectively. We also denoted them as $T_{categorical}$, $T_{numerical}$ for simplification, where n, a denotes the node and corresponding split attribute respectively, s is a split point of numeric attribute and l is a indicator variable set to 0 to indicate leaf child, otherwise set to 1 to indicate right child.

In the Map phase, the training dataset D is partitioned across a set of mappers. Each mapper loads the current model M and the input nodes S into memory, and goes through the assigned subset D^* and applies a Map function to each record in D^*. Local summary statistics can be collected on subsets of the training data and later aggregated.

The Algorithms executed by each mapper is outlined in Algorithm 2 and Algorithm 3. Given a training record $(\mathbf{x}, y)$, a mapper will first determine whether the record current reading is part of the input dataset for a node in S by traversing the current model M. Once the node is determined, the next step is to update count matrix $T_{\{n,a\}}$ or $T_{\{n,a,s\}}$ associated with the current splitter. After all mappers have processed its



	Outlook	Temp	Humidity	Windy
Split 0	Sunny	85	85	false
	Sunny	80	90	true
	Overcast	83	86	false
	Rainy	70	96	false
	Rainy	68	80	false
	Rainy	65	70	true
	Overcast	64	65	true
Split 1	Sunny	73	95	false
	Sunny	69	70	false
	Rainy	75	80	false
	Sunny	75	80	true
	Overcast	72	90	true
	Overcast	81	75	false
	Rainy	71	91	true

Outlook(0)

	Yes	No
Sunny	0	2
Overcast	2	0
Rainy	2	1

...

Windy(0)

	Yes	No
True	1	2
False	3	1

Outlook(1)

	Yes	No
Sunny	2	1
Overcast	2	0
Rainy	1	1

...

Windy(1)

	Yes	No
True	2	1
False	3	1

Outlook

	Yes	No
Sunny	2	3
Overcast	4	0
Rainy	3	2

...

Windy

	Yes	No
True	3	3
False	6	2

Fig. 1. The overview of PDTSSE algorithm. Decision tree is expanded at the root node after one iteration.

Algorithm 2 MR_EXPANDNODESMAP::MAP

```
Input: ExpandNodeSet $S$, TreeModelFile $M$, Training
    record $(\mathbf{x}, y) \in D^*$
Output: Table $T_{\{n,a,v\}}[c]$, $T_{\{n,a,s,l\}}[c]$
 1: $n$ = TraverseTree ($M$,$\mathbf{x}$)
 2: if $n \in S$ then
 3:     candidate attributes $A_{candidate} \leftarrow$
        getCandidateAttrs($M$,$n$)
 4:     for all attribute $a \in A_{candidate}$ do
 5:         $v \leftarrow$ attribute value on $a$ in $\mathbf{x}$
 6:         if $a$ is categorical then
 7:             countMatrix $T_{\{n,a\}}$ =
                getCountMatrix($n, a$)
 8:             addToTable($(\mathbf{x}, y), T_{\{n,a\}}$ )
 9:         else
10:             for all split point $s$ of attribute $a$ do
11:                 addToTable($(\mathbf{x}, y), T_{\{n,a,s\}}$ )
12:             end for
13:         end if
14:     end for
15: end if
```

input records, they output local summary statistics to master node. If the candidate attribute is categorical, the key will be a tuple of the form $\{n, a\}$ and the corresponding count matrix $T_{\{n,a\}}$ as values. Otherwise, the tuple will be of the form $\{n, a, s\}$ and $T_{\{n,a,s\}}$ as values. Subsequently, a combine function will aggregate the values with the same key to reduce communication and I/O cost. In the Reduce phase, each reducer takes the values associated with the particular key and aggregates values for every key. Finally, the output of reducers gives the global count matrix corresponding with each candidate split. Thus we can evaluate possible splits for nodes S in the next step.

Algorithm 3 MR_EXPANDNODESMAP::MAP_FINALIZE

```
Input: Table $T_{numerical}, T_{categorical}$
Output: countMatrix $T_{\{n,a\}}$, $T_{\{n,a,s\}}$ on each node $n$ with
    attribute $a$
1: for all countMatrix $T_{\{n,a\}}$ in $T_{categorical}$ do
2:     Output $((n, a), T_{\{n,a\}})$
3: end for
4: for all countMatrix $T_{\{n,a,s\}}$ in $T_{numerical}$ do
5:     Output $((n, a, s), T_{\{n,a,s\}})$
6: end for
```

2) SubTree Building Process: In the SubTree Building process, every mapper works on the output from the summary statistics job, by aggregating the local count matrices with the same key, we can get expanding nodes S and evaluate the quality of all possible sampling splits for each node in S at the same time, then expand tree and generate new child nodes concurrently in the same way as serial decision tree algorithm. The gini index of categorical attributes can be calculated from count matrix $T_{\{n,a\}}$. For numeric attributes, $gini_{min}$ can be computed from all sampling split points. All the intervals with $gini_{est} \geq gini_{min}$ are eliminate to derive alive intervals. For each alive interval, we begin to scan attribute list L associated with this attributer from the boundary, and update count matrix simultaneously to evaluate the gini index at every split point.

The algorithm executed on each mapper is outlined in Algorithm 4 and Algorithm 5. Each mapper processes two types of keys. The first is of the form $\{n, a\}$ with a value list V of all the output by the mappers. These partial statistical matrices are aggregated to get a single count matrix with the values for all local statistics information about node n with categorical attribute a. The other type of key that a reducer processes belongs to numerical attributes. The keys corresponding to numeric attribute are of the form $\{n, a, s\}$. Here the set V associated with each key is a set of count matrices consisting of statistics information about SS split point s corresponding with node n, attribute a. Candidate attributes on each node can be derived from model M, for every node being expanded in S, if the node's size falls below a threshold then labeled with majority label. Otherwise, the best splitter will be chosen from candidate attributes.

Algorithm 4 MR_EXPANDTREENODESMAP::MAP

```
Input: Key $k$, Value Set $V$, TreeModelFile $M$, Table $T_{categorical}$, $T_{numerical}$
 1: if $k == n, a$ then
 2:     add $V$ to $T_{categorical}$
 3: else
 4:     add $V$ to $T_{numerical}$
 5: end if
```

Algorithm 5 MR_EXPANDTREENODESMAP::MAP_FINALIZE

```
Input: Key $k$, Value Set $V$, TreeModelFile $M$, Table $T_{categorical}$, $T_{numerical}$ and Attribute Lists $L$
 1: expandNodeSet $S \leftarrow$ getExpandNodes($T_{categorical}$, $T_{numerical}$)
 2: for all expand Node $n$ in $S$ do
 3:     $y_{freq} \leftarrow$ getMajorityLabelOfData($n$, $T_{\{n,a,v\}}[c]$, $T_{\{n,a,s,l\}}[c]$)
 4:     $A_{candidate}$ = getCandidateAttrs($n$,$T_{\{n,a,v\}}[c]$, $T_{\{n,a,s,l\}}[c]$)
 5:     Node size $|n| \leftarrow$ SizeOf($n$, $T_{\{n,a,v\}}[c]$, $T_{\{n,a,s,l\}}[c]$)
 6:     if $|n| \leq$ num_threhold then
 7:         labelNode($n, y_{freq}$)
 8:     else if identicalLabel($n$, $T_{\{n,a,v\}}[c]$, $T_{\{n,a,s,l\}}[c]$) or isIdentical($n, T_{\{n,a,v\}}[c], T_{\{n,a,s,l\}}[c]$) then
 9:         label $y_{identical} \leftarrow$ identicalLabel($n$)
10:         labelNode($n, y_{identical}$)
11:     else
12:         Split $split_{best} \leftarrow$ findBestSplit($T_{\{n,a,v\}}[c]$, $T_{\{n,a,s,l\}}[c], L$)
13:         for all Node $d$ in $n_c$ do
14:             UpdateQueues($|d|, M, d, A_{candidate}, L$)
15:             UpdateTreeModel($M, n$)
16:             UpdateNodeCandidateAttrs($n, A_{candidate}$)
17:         end for
18:     end if
19: end for
```

Figure 1 gives an overview of PDTSSE algorithm. In each iteration, a new level of nodes is appended to the tree, that is, the tree's depth is incremented by 1. There needs one scan of entire dataset, once the data scan is complete, local statistical information is merged and send to the controller, which makes the splitting decision for each expanding node of the tree and build child nodes where needed. If the node is already pure enough, the splitting is stopped and the node is assigned a label. This building procedure is efficient because only the count matrices, which are fixed in their size, are kept in main memory.

D. MR_InMemBuildNodes: Tree Induction In Memory

As tree induction progresses, the node size becomes small enough to fit in memory at lower levels of trees. At such point, rather than continuing tree induction using MR_ExpandNodes task, the Controller completes tree induction in memory using a different MapReduce job namely MR_InMemBuildNodes with task parallelism. This job partitions D^* across a set of mappers. The map function then processes every record $(\mathbf{x}, y)$ and traverses the tree in M to see whether the record is input to some node n, where $n \in S$. If such a node is found, the map function outputs the node n as the key and a tuple of the form $(\mathbf{x}, y)$ as the value. The reduce function receives as input a node n (as key) and the set of training records that are input to the node (as values). At last, The reducer loads the training records for n into memory and completes subtree construction rooted at n using in-core algorithm.

III. THEORETICAL ANALYSIS

In this section, we will give the complexity of splitting and training error bound of PDTSSE.

A. Complexity of Splitting

Every iteration consists of two steps: an updating step performed simultaneously by all the processors and a merging step performed by the master processor. Whenever MR_ExpandNodes job is executed, it needs a scan over dataset and attribute lists once for each numeric attribute during the procedure of summary statistics. After getting the global information about class distribution for all possible SS splits, we can choose a split with minimum gini value denoted as $gini_{min}$ among all of the SS splits. If the test attribute a is numeric, then go to scan through all the records in the attribute list corresponding with attribute a in the selected sampling split, and meanwhile use hill-climbing method to evaluate the lower bound of each interval to derive survival intervals. The time complexity of this process is $O(c \times q_{alive})$, which is relevant to the number of label values instead of the size of the interval. Finally, the best splits is obtained using attribute lists L and count matrix relevant to the selected split is updated for each record current read. Thus the computation complexity of the selection of the best split $gini_{best}$ depends only on the number of classes c, dataset size $|D|$ and the number of survival interval q_{alive}. Since $q_{alive} \ll q$ where q denotes the number of intervals, compared with SLIQ and SPRINT, PDTSSE significantly decrease the computation cost in selecting the best split. For large datasets that can't be loaded into memory completely once, the number of candidate attributes in each internal node decreases, so the time complexity of choosing splitter and communication cost decreases. The only memory allocation is for the count matrices. As the number of intervals is constant, operations on count matrices take a constant amount of time. Every processor performs at most $|D|/W$ matrix updates. There are totally $W \times |S| \times (A_c + A_n \times q)$ count matrices at most, where $|S|$ is the number of leaves that are to be expanded in the current iteration, W is the number of processors and A_c, A_n are the number of categorical attributes and numeric attributes respectively. Assuming that $W, |S|, c,$ and d are all independent of the dataset size $|D|$, thus space and communication complexities is constant.

B. Error Bound of PDTSSE

In this section, we investigate the training error of PDTSSE. Assume that impurity function G, the gap in the impurity function before and after splitting is denoted as $\triangle$, suppose that s is the best split point of splitting attribute a_i for a tree node n, so that node n is split according to the rule $x^{(i)} \leq s$. Denote by τ the probability that a record reaching n is directed to its left child node. Denote further by $p_{n,j}, p_{L,j}$ and $p_{R,j}$ the probability of label j in this node n and its left and right child nodes respectively. Define the function for the change in value of impurity before and after splitting $\triangle(\tau, \{p_j\}, \{p_{L,j}\}, \{p_{R,j}\}) = \triangle(n, i, s)$ as

$$\triangle = G(\{p_j\}) - \tau G(\{p_{L,j}\}) - (1-\tau) G(\{p_{R,j}\}) \quad (1)$$

$\triangle$ can be calculated precisely at every candidate split using (1). However, for PDTSSE, it initially divided the values of a numeric attribute into B roughly equal-size intervals with $B-1$ split points $s_1, s_2, \ldots, s_{B-1}$ where $s_1 \leq s_2 \ldots \leq s_{B-1}$. Suppose that the best split point is s, and $s_k \leq s \leq s_{k+1}$. Since the number of points in the interval $[s_k, s]$ is bounded, there is a bound on the degree of change in $\triangle$ if one node is split at s_k instead of the best split point s.

For Decision tree T, the training error rate of T is

$$e_T = \frac{1}{N} \sum_{\text{leaf } n \text{ in } T} |n| (1 - \max\{p_{n,j}\})$$

Assume that the impurity function G is continuous and satisfies $G(\{p_j\}) \geq 1 - \max_j \{p_j\}$. Thus the inequality implies that $e_T \leq G_T$ where G_T is defined as follows

$$G_T = \frac{1}{N} \sum_{\text{leaf } n \text{ in } T} |n| G(\{p_{n,j}\}) \quad (2)$$

For our analysis, only one new child node is generated in each expanding. As the split of categorical attribute is identical to serial algorithm, here we only consider the numeric attributes. Let T_t be the tree after the tth iteration for the decision tree only one node is expanded per iteration, and n_L, n_R denote its left and right child nodes respectively. Then

$$G_{T_{t-1}} - G_{T_t} = \frac{|n|}{N} \triangle (n, i, s) \quad (3)$$

Definition 1. *An internal node n split by a rule $\mathbf{x}^{(i)} \leq s$ is said to perform locally well with respect to a function $f(\{p_{n,j}\})$ if it satisfies $\triangle(n, i, s) \geq f(\{p_{n,j}\})$. A tree T is said to perform locally well if every interval node n in it performs locally well. Finally, a decision tree building algorithm performs locally well if for every training set, the output tree performs locally well.*

Proof: From (3), we can get that the lower bound of $\triangle(n, i, s)$ is $f(\{p_{n,j}\})$, also the upper bound of G_{T_t} and e_{T_t} is related to $f(\{p_{n,j}\})$. Suppose that T_{t-1} has a leaf for which $\frac{|n|}{N} f(\{p_{n,j}\})$ can be lower-bounded by a quantity $h(t, G_{T_{t-1}})$ which depends only on t and $G_{T_{t-1}}$. Then from the recurrence $G_{T_t} \leq G_{T_{t-1}} - h(t, G_{T_{t-1}})$, we can derive a lower bound on the training error rate of an algorithm that performs locally well. As a simple example $f(\{p_{n,j}\} = \alpha G(\{p_j\}), \alpha \geq 0$. By (2), and since the number of leaves in T_{t-1} is t, there exists a leaf n in T_{t-1} for which $\frac{|n|}{N} G(\{p_{n,j}\}) \geq G_{T_{t-1}}/t$. Let $\tilde{n}$ be the node which is split in the tth iteration, thus $\frac{|n|}{N} f(\{p_{n,j}\}) \geq \frac{\alpha}{t} G_{T_{t-1}}$, where n, $\tilde{n}$ represents the best splits for $\triangle_n$, $\triangle_{\tilde{n}}$ respectively. Then we get

$$\begin{aligned} G_{T_{t-1}} - h(t, G_{T_{t-1}}) &= \frac{|\tilde{n}|}{N} \triangle_{\tilde{n}} \geq \frac{|n|}{N} \triangle_n \\ &\geq \frac{|n|}{N} f(\{p_{n,j}\}) \geq \frac{\alpha}{t} G_{T_{t-1}} \end{aligned} \quad (4)$$

Let G_0 be an upper bound on G_{T_o}, we obtain $G_{T_t} \leq G_0(t-1)^{-\alpha/2}$ by (4), therefore

$$e_{T_t} \leq G_0(t-1)^{-\alpha/2}$$

Kearns and Mansour [17] give a proof that top-down decision tree learning algorithms are boosting algorithms, that is to say, if the functions that label the internal nodes of the decision tree can weakly approximate the unknown target function, the algorithm will amplify this weak advantages to build a tree achieving any desired level of accuracy, so the performance of resulting tree depends on the local performance of internal tree nodes. ■

Theorem 1. *Assume that the intervals which PDTSSE operate on are equal-size. Let $\mathbf{x}^{(i)} < s$ is the best split predicate for a leaf n, then $\forall \delta \geq 0$. There exists B that only depends on $\tau, \{p_j\}, \{p_{L,j}\}, \{p_{R,j}\}$ and δ, such that the split $\mathbf{x}^{(\tilde{i})} \leq \tilde{s}$ chosen by PDTSSE algorithm with B equal-size intervals satisfies $\triangle(n, \tilde{i}, \tilde{s}) \geq \triangle(n, i, s) - \delta$.*

Proof: Assume that B is fixed, and consider the split $\mathbf{x}^{(i)} < u_k$, $u_k \leq s \leq u_{k+1}$ and $\tilde{\tau}, \tilde{p_L}, \tilde{p_R}$ denote the quantities relevant to this split. Let ρ_j denote the probability that a training record $\mathbf{x}$ with label j that satisfies $u_k \leq \mathbf{x}^{(i)} \leq s$. Then $\tilde{\tau} = \tau - \rho_0 - \rho_1, \tilde{p}_{L,j} = (\tau \cdot p_{L,j} - \rho_j)/\tilde{\tau}, \tilde{p}_{R,j} = ((1-\tau)p_{R,j} + \rho_j)/(1-\tilde{\tau})$.

By the continuity of $\triangle(\tau, \{p_j\}, \{p_{L,j}\}, \{p_{R,j}\})$, for every $\delta > 0$, there exists ϵ such that

$$\triangle(\tau, \{p_j\}, \{p_{L,j}\}, \{p_{R,j}\}) - \triangle(\tilde{\tau}, \{p_j\}, \{\tilde{p}_{L,j}\}, \{\tilde{p}_{R,j}\}) \leq \delta$$

$\forall \rho_j \leq \epsilon$. Since $\rho_j \leq \frac{1}{B+1}$, we can guarantee that $\rho_j \leq \epsilon$ by setting $B = 1/\epsilon$. We thus have $\triangle(n, \tilde{i}, \tilde{s}) \geq \triangle(n, i, u_k) \geq \triangle(n, i, s) - \delta$ ■

Corollary 1. *Assume that the standard decision tree algorithm performs locally well with respect to a function $f(\{p_j\})$, and that the functions operating on equal-size intervals. Then $\forall \delta(p_j) \geq 0$, the PDTSSE algorithm performs locally well with respect to $f(\{p_j\}) - \delta(\{p_j\})$, in the sense that for every training set there exists B such that the tree constructed by the PDTSSE algorithm with B bins performs locally well. Moreover, B doesn't depend on the size of the training set.*

We derive an upper bound on the error rate of PDTSSE from the theorems and corollary above. Set $f(\{p_j\}) = \alpha G(\{p_j\}), \alpha \geq 0$. We get from Definition 1 that $e_{T_t} \leq G_0(t-1)^{-\alpha/2}$. Applying Corollary 1 with $\delta(\{p_j\}) = \frac{\alpha}{2} G(\{p_j\}) = f(\{p_j\})/2$, we deduce that the PDTSSE's error rate is guaranteed to be no more than $G_0(t-1)^{-\alpha/4}$ when using enough intervals.

TABLE I. DATA SET SUMMARY.

Dataset	#Attributes	#Classes	#Instances
Segment	22	2	8124
Letter	16	26	20000
Abalone	8	29	4177
Adult	14	2	48842
ISOLET	617	26	7797
Satimage	36	6	6435
Shuttle	9	7	58000
Connect	42	3	67557
Covertype	54	7	581012
Poker hand	10	10	1000000

IV. EXPERIMENTAL EVALUATION

In this section, we show our experiments and present a detailed performance evaluation of PDTSSE.

A. Experimental Setup and Datasets

We run our experiments on ten datasets from the UCI repository, the characteristics of these datasets are shown in Table I.

All of our experiments were performed on a MapReduce equipped cluster that consists of 5 processors, each of which was configured to use 2GB of RAM and 320GB of hard drive space. One as the master node and the others work as slave nodes. To mitigate the effects of varying cluster conditions, all the running times have been averaged over multiple runs.

B. Experimental Results

To verify the effectiveness and efficiency of PDTSSE in the parallel distributed setting, in our first experiment, we compare the values of gini index of the best splitters generated by SSE methods to those using direct method (DM), dataset sampling (DS) and sampling the splitting points (SS) for different number of intervals. We have presented the results using 10 and 30 intervals for each numeric attribute. Table II shows the results of exact and estimated gini index using different methods.

These results show that the SS method missed the minimum gini index most of the time, the same holds true for the DS method. The accuracy of the SS method is more sensitive to the number of intervals, the estimated gini may be not good when there are not enough intervals. The SSE method we applied in PDTSSE missed the exact gini index only for a few cases and the estimated gini generated by the SSE method is more accurate than those of the SS method. These results demonstrate that SSE can achieve a very good estimate of the splitter with minimum gini index.

Our next experiment presents a comparison of time taken in selecting the best split predicate on the Poker Hand dataset by PDTSSE, SPRINT and SLIQ under the condition of different number of processors.

The results of this experiment are shown in Figure 2. It is apparent that SPRINT needs less of time than SLIQ, and PDTSSE outperforms other methods when using the same number of workers, in addition, the time for selecting best splits significantly decreases as the number of workers is increased.

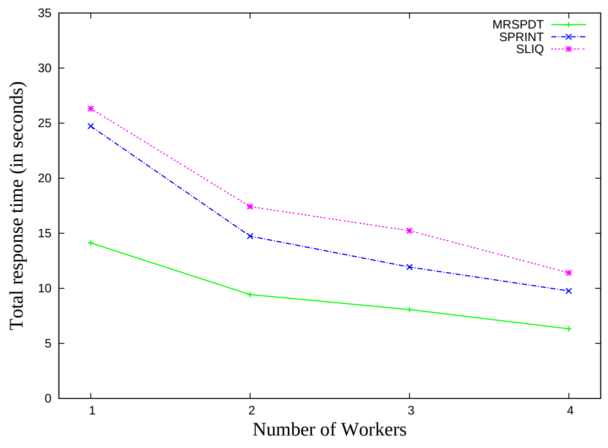


Fig. 2. Comparison of running time for selecting best splits using different parallel decision tree algorithms.

TABLE III. THE ACCURACY OBTAINED FOR DIFFERENT DATASETS.

Dataset	C4.5	CART	SPRINT	PDTSSE	
				q=20	q=50
Segment	94.45%	94.38%	94.51%	94.57%	94.57%
Letter	84.51%	83.67%	84.58%	84.12%	84.12%
Abalone	20.13%	21.23%	-	20.32%	20.49%
Adult	86.20%	86.17%	-	85.86%	86.32%
ISOLET	83.81%	82.89%	84.86%	84.21%	85.31%
Satimage	86.40%	85.75%	86.46%	86.12%	86.54%
Shuttle	99.93%	99.89%	99.87%	99.97%	99.97%
Connect	80.28%	80.32%	-	79.96%	80.20%
Covertype	-	-	-	68.18%	68.21%
Poker hand	-	-	-	54.82%	55.38%

In next experiment, we give a detailed comparison of PDTSSE's accuracy to C4.5, CART and SRPINT using different number of intervals based on 5-fold cross validation technique. Results in Table III show that the accuracy obtained by PDTSSE is quite similar or comparable to those of C4.5, CART and SPRINT. The greater the number of intervals, the higher accuracy will PDTSSE achieve.

The last set of experiments present the scalability performance of PDTSSE in terms of the parallel running time obtained for various training set sizes. For these experiments, we randomly split the dataset Covertype into 5 roughly equal-size groups, first on a single group, then two groups, and so on up to five groups. These increasingly larger training datasets are named D_1, D_2, D_3, D_4 and D_5 respectively. In these experiments, the size of training set is kept constant while the number of workers changes from 1 to 4.

Figure 3 shows the runtime scalability of PDTSSE. As expected, training time increases as the amount of training data is increased. The time, which is related to speedup tends,

TABLE II. EXACT AND ESTIMATED GINI USING DIFFERENT METHODS.

Dataset	DM	DS		SS		SSE	
		10%	30%	q=10	q=30	q=10	q=30
Segment	0.7143	0.7313	0.7221	0.7219	0.7162	0.7161	0.7143
Letter	0.9403	0.9421	0.9403	0.9403	0.9418	0.9403	0.9403
Abalone	0.8620	0.8649	0.8625	0.8632	0.8624	0.8620	0.8620
Adult	0.3228	0.3412	0.3356	0.3285	0.3234	0.3259	0.3228
ISOLET	0.9263	0.9475	0.9362	0.9421	0.9412	0.9414	0.9272
Satimage	0.6532	0.6591	0.6546	0.6541	0.6532	0.6532	0.6532
Shuttle	0.1758	0.1758	0.1810	0.1758	0.1758	0.1758	0.1758

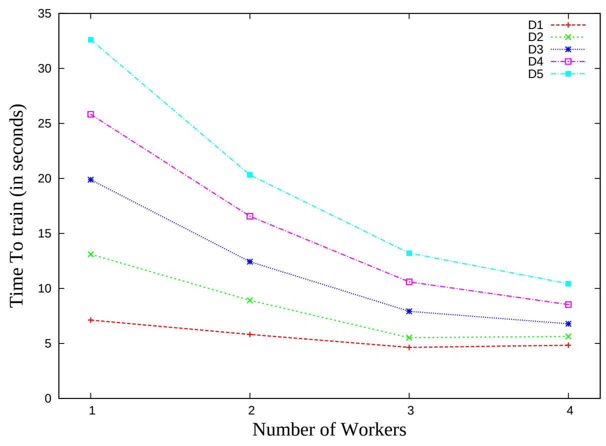


Fig. 3. Parallel Running time versus data size and number of workers.

decreases as the number of processors is increased. In addition, relative speedups are improved for larger training set sizes. The results fit the theoretical analysis. For large datasets, computation becomes a significant factor of the overall running time, therefore, PDTSSE can be used to classify large scale datasets.

V. CONCLUSION

In this work, we have presented PDTSSE, a new algorithm for large-scale decision tree learning in MapReduce framework, which has low memory requirement, high efficiency and good scalability. We show that sampling splitting points with estimation is central to PDTSSE. Future works include extending this algorithm to larger datasets and clusterings. We will also utilize PDTSSE in practical applications.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China under grant No.61170019 and the Tianjin Key Laboratory of Cognitive Computing and Application.

REFERENCES

[1] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, “Machine learning, neural and statistical classification,” 1994.

[2] J. Gehrke, R. Ramakrishnan, and V. Ganti, “Rainforest-a framework for fast decision tree construction of large datasets,” in *VLDB*, vol. 98, 1998, pp. 416–427.

[3] J. Catlett, “Megainduction: Machine learning on very large databases,” Ph.D. dissertation, Basser Department of Computer Science, University of Sydney, 1991.

[4] N. Amado, J. Gama, and F. Silva, “Parallel implementation of decision tree learning algorithms,” in *Progress in Artificial Intelligence*. Springer, 2001, pp. 6–13.

[5] D. D. Lewis and J. Catlett, “Heterogenous uncertainty sampling for supervised learning.” in *ICML*, vol. 94, 1994, pp. 148–156.

[6] P. K. Chan and S. J. Stolfo, “Meta-learning for multistrategy and parallel learning,” in *Proc. Second Intl. Workshop on Multistrategy Learning*, 1993, pp. 150–165.

[7] S. Ranka and V. Singh, “Clouds: A decision tree classifier for large datasets,” *Knowledge Discovery and Data Mining*, pp. 2–8, 1998.

[8] M. Mehta, R. Agrawal, and J. Rissanen, “Sliq: A fast scalable classifier for data mining,” in *Advances in Database Technology—EDBT’96*. Springer, 1996, pp. 18–32.

[9] J. Shafer, R. Agrawal, and M. Mehta, “Sprint: A scalable parallel classi er for data mining,” in *Proc. 1996 Int. Conf. Very Large Data Bases*. Citeseer, 1996, pp. 544–555.

[10] M. V. Joshi, G. Karypis, and V. Kumar, “Scalparc: A new scalable and efficient parallel classification algorithm for mining large datasets,” in *Parallel processing symposium, 1998. IPPS/SPDP 1998. proceedings of the first merged international... and symposium on parallel and distributed processing 1998*. IEEE, 1998, pp. 573–579.

[11] J. Gehrke, V. Ganti, R. Ramakrishnan, and W.-Y. Loh, “Boat-optimistic decision tree construction,” in *SIGMOD Conference*, vol. 12, 1999.

[12] R. Jin and G. Agrawal, “Communication and memory efficient parallel decision tree construction,” in *SIAM Conference on Data Mining (SDM)*, 2003, pp. 119–129.

[13] N. Amado, J. Gama, and F. Silva, “Exploiting parallelism in decision tree induction,” in *Proceedings from the ECML/PKDD Workshop on Parallel and Distributed computing for Machine Learning*, 2003, pp. 13–22.

[14] Y. Ben-Haim and E. Tom-Tov, “A streaming parallel decision tree algorithm,” *The Journal of Machine Learning Research*, vol. 11, pp. 849–872, 2010.

[15] B. Panda, J. S. Herbach, S. Basu, and R. J. Bayardo, “Planet: Massively parallel learning of tree ensembles with mapreduce,” *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1426–1437, 2009.

[16] J. Dean and S. Ghemawat, “Mapreduce: Simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[17] M. Kearns and Y. Mansour, “On the boosting ability of top-down decision tree learning algorithms,” in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. ACM, 1996, pp. 459–468.

An Adaptive Data Resourcing Model to Relieve the Pain of Searching from Hybrid Clouds for Big Data

Yilang Wu
Grd. School of Comp. Sci. & Engineering
University of Aizu
Aizu-wakamatsu, Japan, 965-8580
Email: d8152103@u-aizu.ac.jp

Junbo Wang
University Business Innovation Center
University of Aizu
Aizu-wakamatsu, Japan, 965-8580
Email: j-wang@u-aizu.ac.jp

Zixue Cheng
School of Comp. Sci. & Engineering
University of Aizu
Aizu-wakamatsu, Japan, 965-8580
Email: z-cheng@u-aizu.ac.jp

***Abstract*— The pain comes together with the prevalence of searching from hybrid clouds for big data, which is induced by the ambiguity of user expectation, the heterogeneity of data resources, the complexity of data computation, and the perceptibility of returned data. In this study, we propose an adaptive data resourcing model as a pain reflliever (or analgesia) to the aforementioned problems; specifically by the means of: granular data fusion to the data schema of open source or open data, elastic computing platform base on networked computers, fitness oriented query base on semantic situation awareness, and data visualization via (temporal/spacial) responsive interface. We achieve good performance by implementing this adaptive method into a demo system for teamwork, for instance the utilization of the idle resources, accomplishment of the tracked issues, and reusability of knowledge experience, etc. And we judge the conclusion that our adaptive data resourcing method is effective analgesia to the pain of searching for big data.**

***Index Terms*—Big Data, Hybrid Clouds, Situation Awareness, Adaptiveness, Teamwork**

I. Introduction

A. Motivations

The large pools of data that can be captured, communicated, aggregated, stored, and analyzed —is now part of every sector and function of the global economy. In 2011, a McKinsey report [1] advocated that the big data is the next frontier for innovation, competition, and productivity. And it is now making revolutionary breakthroughs in commerce, science and society.

However, it is not easy for citizens to keep pace with such an accelerated society that driven by the big data, much less being productive. With the exploding amount of data, individuals or teams, no matter the genius or think tank, all suffer from the chaos of VUCA[2] (volatility, uncertainty, complexity and ambiguity) situations. Furthermore, it is apt to come up with emergent and sophisticated puzzles, which could hardly been illustrated in a clear description.

B. Purpose and Issues

We name such phenomenon as the pain of searching in big data, which is caused by

- the ambiguity of user expectation,
- the heterogeneity of data resources,
- the complexity of data computation,
- and the perceptibility of returned data.

Thus, we are aiming to take advantage our limited resources to invent the analgesia for such a pain, or at least make ourselves more stress-free.

We came up with both technical and administrative issues:

1) One authenticated hybrid-clouds service is required: there are many cloud services available in big data, but most of them are enclosed eco-system from each other, making it annoying for users switching between the services.
2) We need a consistent data overview: the data inconsistency in the current big data, whether in format or meanings, makes it very inconvenient for viewing or processing.
3) An automatic awareness is expected: currently, most of the search is done base on use input queries, far from understanding user's ambiguous expectations
4) A pervasive easy going interface is in need.

C. Contributions

In this study, we propose an adaptive data resourcing model to relieve the pain of searching in big data, and we decided to build our own hybrid clouds. Here are main contributions:

1) A granular data fusion base on the data schema of open source platforms: it involves the data of productivity management, relation management, integrated library, and the content management such as blog and wiki.
2) An elastic computing platform base on networked computers: using network setup and Hadoop DFS to build a computer cluster, and utilize the idle computing resources, such as the retired computers or idle PCs (this was done via nested virtual machines) of lab mates'.
3) A set of fitness oriented queries and semantic situation awareness: by the corporation with U.B.I.C. of the University of Aizu, we implement semantic situation awareness using MapReduce functions; and also a set of queries that could easily fetch metadata from the big data.
4) A perceptible data visualization via (temporal/spacial) responsive interface: we take advantage of the responsive web UI design and asynchronously visualize the returned data within tolerated time delay.

D. Paper Structure

In the fellowing sections, we first illustrate background and related work in section II. Then we introduce an adaptive resourcing model in section III to regulate the fitness, responsiveness, granularity and complexity of the analgesia process. In section IV, we the take teamwork management for a case study to our data resourcing model, and deploy an integrated system base on hybrid open source tools, involving productivity management, relation management, integrated library, and content management for efficient teamwork with less stress or pains. In section V, we evaluate the case study and method by analyzing the improvement of teamwork performance. The upgraded teamwork by using our service is getting much more smooth and easier, in terms of utilization of the idle resources, accomplishment of the tracked issues, and reusability of knowledge and experience, etc; Finally we judge our conclusion in section V that our adaptive data resourcing method is effective analgesia to the pain of big data.

II. Related Work

A. Various Information Searching Models

For information retrieval, the most popular model is the keyword based one, in which a keyword list is input by a user to describe request for the content of information. The keyword based searching is equipped with statistical methods [3], e.g., occurrence frequency of a keyword. Keyword based information retrieval model has shown great power, e.g.. Google. However, the keyword based information retrieval model cannot represent the semantic relations between keywords. To overcome it, a concept-based model was proposed to show semantic relation among keywords [4]. The model is built base on the consideration that the meaning of a word is related with concepts in the real world. Therefore, a content of information can be summarized by a set of concepts. If a keyword is inputted, several concepts will be mapped with it in the model, and information will be retrieved based on the concepts.

In addition to the concept-based models, more effective model to understand the user input are semantic based approaches [5][6] and meta search engine [7][8][9]. Among them, sentences are indexed instead of words in [6], and the inputted keywords will be extended based on the public database, e.g., Wikipedia or WordNet, in order to search closer results for users in [7][8][9].

B. Cloud based Searching Model

Recently, new information retrieval models become necessary and important with the advancement of cloud computing technology. When more and more information have been stored in Cloud, e.g. emails, sensor data, life logs, health records, and so on, it has become a big challenge to effectively search information from clouds. Data encryption technology [10][11] makes data utilization more reasonable. In [11], a fuzzy based keyword search was proposed over encrypted data in cloud computing. When inputting a keyword, an edit distance between the inputted and stored keywords will be computed and the corresponding files will be shown to users based on fuzzy theory.

C. Our Unique Focus

However, most of the current services for search from clouds are enclosed from each other, users could hardly achieve seamless connection among clouds, unless taking several jumps even though they have the access permission to each of them. Meanwhile, the data in different clouds is generally inconsistent in format or physical meanings, causing unnecessary and costive data conversion steps before processing and searching. Furthermore, users are confused with the various interfaces of different clouds, especially for those none-professional users. In this research, we are trying to propose a novel data-resourcing model for a set of data sources on hybrid clouds.

III. The Novel Searching Method

In this section, we illustrate a novel searching method for big data. We first describe the assumptions in section III-A, and them define the requirements and properties in section III-B, and finally we explained a simple algorithm and evaluation method in section III-C.

A. Assumptions

Normal human's searching behavior is based on cognitives, rather than unconscious work. There mainly three types of cognitive processes, see to Table I. Situation awareness is the perception of environmental elements with respect to time and/or space.

TABLE I
Relationships among Human Cognitive Process [12][13][14]

		Phase	
		Process (the process to achieve knowledge	Outcome (a state of knowledge
Objective	Tactical (short-term)	Situation Assessment	Situation Awareness
	Strategic (long-term)	Sensemaking	Understanding
	Scientific (longer-term)	Understand	Predict

Based on the cognitive processing model, we made assumptions as fellows:

1) Given the big data, situations of the targeted issues could be sensible through data computation.
2) Lacking the data or information is kind of pain for human beings, and pain is one of the fundamental motivations causing human to search for information. Taking researchers for example, it feels really painful when one urgent document is missing; thus the expected information or data is a reliever or analgesia to such a pain.
3) On the other hand, the pain might be transmitted to the searching system if the computation cost is too expensive.

Given the limitation to the current cloud based searching model that we mentioned in subsection II-C, we summarize the types of pain of searching in big data:

1) the ambiguity of user expectation,
2) the heterogeneity of data resources,
3) the complexity of data computation,
4) and the perceptibility of returned data.

And it is necessary to introduce an adaptive balance into the system aided searching, therefore, we illustrate the adaptiveness in Figure 1, making it as the ultimate goal for this study. It will be further explained in Section III-B.

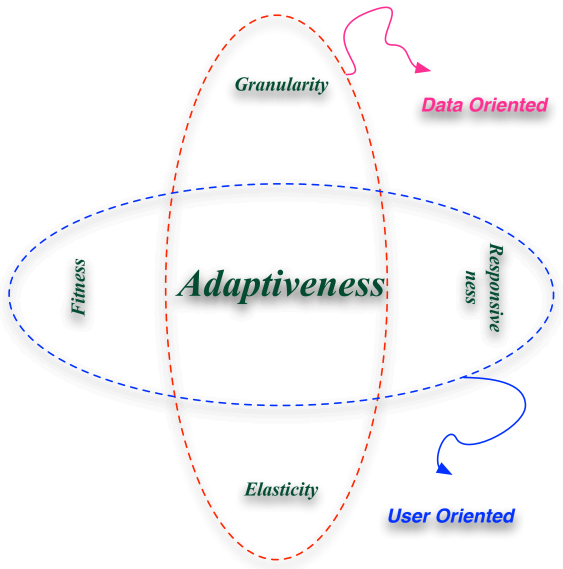


Fig. 1. Adaptiveness of Data Resourcing in Big Data

B. Requirements and Properties of the Model

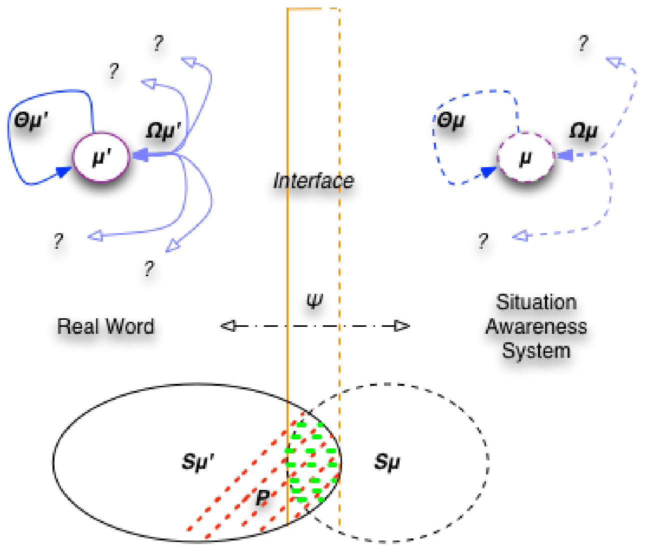


Fig. 2. Modeling of Situation Awareness Process

In this subsection, we define the requirements and properties of the adaptive model, a set of parameters is defined in Table II. We illustrate the process of situation awareness in Figure 3. The left side is the real world that involves the situations and issues of the targeted entity μ' and respectively it is the cyber world on the right side. And there is the interface Ψ in the middle, standing for the system interface.

See to Table II for detailed information. The situations of an entity μ' (whether a human or object) in the real world contain the inclusive situations $\Omega\mu'$ that are rationally depending on other entities, and also the exclusive situations $\Theta\mu'$ that inside the entity μ' itself. The state of the situation is migrating along with the time, for instance $\Delta S(\mu, \Delta T)$ is the situation migration from time t to $(t + \Delta T)$. We have mentioned the existence of pain in the previous sections, there are pain factors P as the subset of situation $S\mu'$. And the main mission of the cyber system is trying to relive the pain, which means returning the analgesia A through the interface Ψ.

TABLE II
PARAMETERS DEFINITION FOR SITUATION AWARENESS PROCESS

PARAM.	DESCRIPTION
μ & μ'	The target entity (and the abstract one) in the situation awareness process;
$\Theta\mu'$ & $\Theta\mu$	The exclusive real-world situation of entity μ' and the abstract one μ respectively;
$\Omega\mu'$ & $\Omega\mu$	The inclusive (dependent) real-world situation of entity μ' and the abstract one μ respectively;
t	The initial time for situation awareness process;
Δt	A period of time while the real world situations $S\mu'$ is changing;
$S(\mu', t)$	The real-world situation $S\mu'$ of entity μ' at time t;
$\Delta S(\mu', \Delta t)$	The transformed real-world situation form $S(\mu', t)$ after Δt;
ΔT	A period of time for the response of Situation Awareness System;
$S(\mu, t)$	The assessed situation to the abstract entity μ at initial time t;
$\Delta S(\mu, \Delta T)$	The situation awareness to the abstract entity μ at time $t + \Delta T$;
Ψ	The interface between real world and the Situation Awareness System;
P	The pains in the real-world situations;
A	The analgesia (pains killer) set by situation awareness;
η	The element of assessed situation S' in format of data;

1) The adaptiveness is defined in Eq. 1.

$$Adaptiveness = \alpha F + \beta R + \gamma G + \lambda E, \quad \text{where } \alpha + \beta + \gamma + \lambda = 1 \qquad (1)$$

2) Fitness: the output of situation awareness is just-matched or being more implicit than that of demanded, see to Eq. 2;

$$F(\mu, t) = (A(\mu, t) \bigcap P(\mu, t))/P(\mu, t) \qquad (2)$$

3) Responsiveness: the output of situation awareness is just-in-time or with less delay, see to Eq. 3;

$$R(\mu, t) = \Delta T/\Delta t \qquad (3)$$

4) Granularity: the data schema of the targeted entity is at least just enough to represent the real information, see

to Eq. 4;

$$G = StructureComplexity(S(\mu,t)) = StructureComplexity(\Theta(\mu,t) \bigcup \Omega(\mu,t)) \quad (4)$$

5) Elasticity: the computation (transformation) complexity of situation assessment, see to Eq. 5

$$E = ComputationComplexity(S(\mu,\Delta T)) = TransformationComplexity(S(\mu,t), S(\mu, t{+}\Delta T)) \quad (5)$$

6) The situation of entity μ' at time t in real world is defined in Eq. 6, and the situation migration of entity μ' from time t to time $t + \Delta T$ is defined in Eq. 7

$$S(\mu',t) = \Theta\mu' \bigcup \Omega\mu' \quad (6)$$

$$S(\mu',t) \xrightarrow{\Delta S(\mu',t)} S(\mu', t+\Delta t) \quad (7)$$

7) Respectively, in the cyber world, the situation of entity $S(\mu,t)$ of μ at time t is defined in Eq. 8, and the situation migration $S(\mu,t)$ is defined in Eq. 9

$$S(\mu,t) = \Theta\mu \bigcup \Omega\mu \quad (8)$$

$$S(\mu,t) \xrightarrow{\Delta S(\mu,t)} S(\mu, t+\Delta t) \quad (9)$$

8) The pain is included in real situation $S\mu'$, see to Eq. 10; and the analgesia would be the join of real situation $S\mu'$ and the cyber situation $S\mu$, see to 11

$$P \subseteq S\mu' \quad (10)$$

$$A = S\mu \bigcap S\mu' = S\mu \bigcap P \quad (11)$$

The main purpose of the adaptive resourcing is to return the metadata of analgesia A via the optimization of data processing; making the data schema just enough to represent the targeted issues, the volume of the computation is just executable by the allocated computation resource, and the returned analgesia being just fit and just in time to to the pain,

C. The Searching Approach

It is obvious that MapReduce[15] programming model highly matches the requirement of our adaptive data resourcing, taking the Hadoop[16] as the computation framework for example, the Distributed File System supports the granularity, the distributed processing in multi nodes supports the elasticity.

The computation flow is illustrated in Figure 3, the system take users' pain (basically users' situations) or users' direct input of queries as system input, and then process the input in the MapReduce function by considering the constrains of granularity and elasticity, and finally try to return the data in fitness and responsiveness. The pseudocode is documented in Alg. 1, there are a variety of ways optimize the Map and Reduce function to improve the fitness and responsiveness, we will explained in detail in future work.

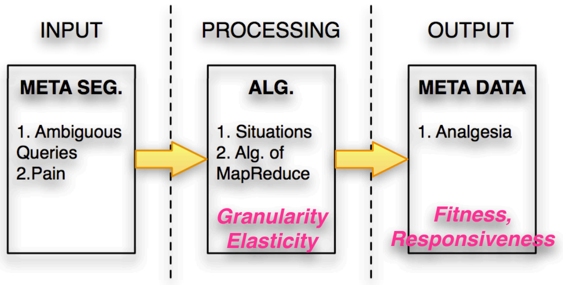


Fig. 3. Flow of Adaptive Resourcing

Algorithm 1 Pseudocode for Adaptive Data Resourcing

```
1: TRY
2: TRY
3: $S \leftarrow Map(\Omega, \Theta)$
4: CATCH ERROR "Error caught in Map function, the volume of the data exceeds the granularity."
5: END TRY
6: $A \leftarrow Reduce(S)$
7: CATCH
8: ERROR "Error caught in Reduce function, the computation of the data exceeds the elasticity."
9: END TRY
```

IV. Implementation

A. Implementation Environment and Tools

TABLE III
System Architecture

Technology for Interface	HTML, CSS, JavaScript (jQuery), PHP, Ruby		
Open Source Platform	Project Management		Redmine
	Relation Management		Vtiger
	Integrated Library		Koha
	Content Management	Wiki	MediaWiki
		Blog	Wordpress
Authentication Server	Oauth2		
Database	MySQL		
Web Server	Apache2		
Distributed Framework	Hadoop		
Operation System	Linux, OS X		
Network	Ethernet, WiFi		
Machine	Cluster of 6 Macintosh, 5 PCs and over 20 Virtual Machines running on PCs		

The system is implemented and deployed by fellowing the adaptiveness requirements. We turn to the most popular technologies, the web programming. The system architecture is enumerated in Table III.

We deployed a 3 clusters to utilize the idle computation resources in lab:

- a cluster of PCs for Web service;
- a cluster of physical and virtual machines that running on lab mates' PCs for Hadoop, everyone can share his/her own computation resources depending on the elastic demand;

- and a cluster of 6 Macintosh, working as pervasive (inside lab) display and also the node of sensor networks which can aware users' indoor locations.

B. Functions and Services

Base on the Oauth server[17], we fuse the teamwork data according to the data schema of 5 open source tools, and there are also many extensions or plugins to fuse extra data source.

a) Blog for Content Management: We need a blogging system to promote the demos or ideas of team's, in format of rich contents such as videos and images. We turn to the Wordpress since it is the most popular blogging system in use on the Web[18] at more than 60 million websites[19]. The Wordpress itself is also a hub of extensions that seamlessly interconnects with other cloud services, e.g. Facebook, Twitter.

b) Wiki for Content Management: A manual documenting tool is also required for teamwork, here we turn to the MediaWiki, since it is supported by the Wikimedia Foundation, advanced in compatibility and extensions.

c) Integrated Library: The integrated library helps to management the books, CDs, DVDs in library. There are more than three options, and we finally choose Koha, for its advance in licensing, community support, and functionaries (routing periodicals, inventory control, authorities, generation of notices to customers, order tracking, etc.), according to [20].

d) Relation Management: The relation management (RM) platform would provide the service of 1) Contact, Calendar, and Inventory, etc. There are two potential options, and we choose the Vtiger rather than the Compierem, mainly for the reason of extensiveness and compatibility. With some extensions, the Vtiger is able be seamlessly fused with Google contact and calendar.

e) Productivity Management: Getting things done is the most wanted output of teamwork, David Allen's wide spreading book[21], shared the breakthrough method for stress-free performance. Allen's premise is simple: our productivity is directly proportional to our ability to relax. Here we introduced an issues tracking open source tool Redmine to: granularly break down the issues and human resource,

C. Interface of the System

We implement the main page, a responsive web UI, to encapsulate the aforementioned tools, see to Figure 4. The data resourcing model in section III generates a set of metadata that customized for each user, and visualized in the main page.

Taking a specific use case, the issues in backing up research students's demo project, for example, we will show how the data is interlinked via the team working flow, see to Fig. 5:

1) Login to the interface with a global authentication;
2) Break the project down into different sub tasks, and create issues for each tasks, by setting the issue title, assignee, time stamp, priority, notes, etc;
3) The back up the technical details, system specifications, source codes, or other documentations inside the project management (professional contents) and also the wiki (general contents);
4) The integrated library records users' access to library items;
5) Before the project is accomplished, make the demonstration page in Blog.

In such a case, massive data that related with one project is unconsciously interlinked; it would be possible for a new user to review the whole story in the main interface, see to Fig. 4.

D. Features of the System

The system achieves several features that we expected:

1) Granularity: the Oauth server consistently fused the standard data schema in the selected open source tools;
2) Elasticity: the Hadoop on the computer clusters can support complex computation in different scale
3) Fitness: the situation data that we collected are mainly related with teamwork, without much noise; and the Reduce function in MapReduce can gradually improves the fitness of adaptive data resourcing;
4) Responsiveness: the UI is responsive in layout and also time due to asynchronous data visualization.

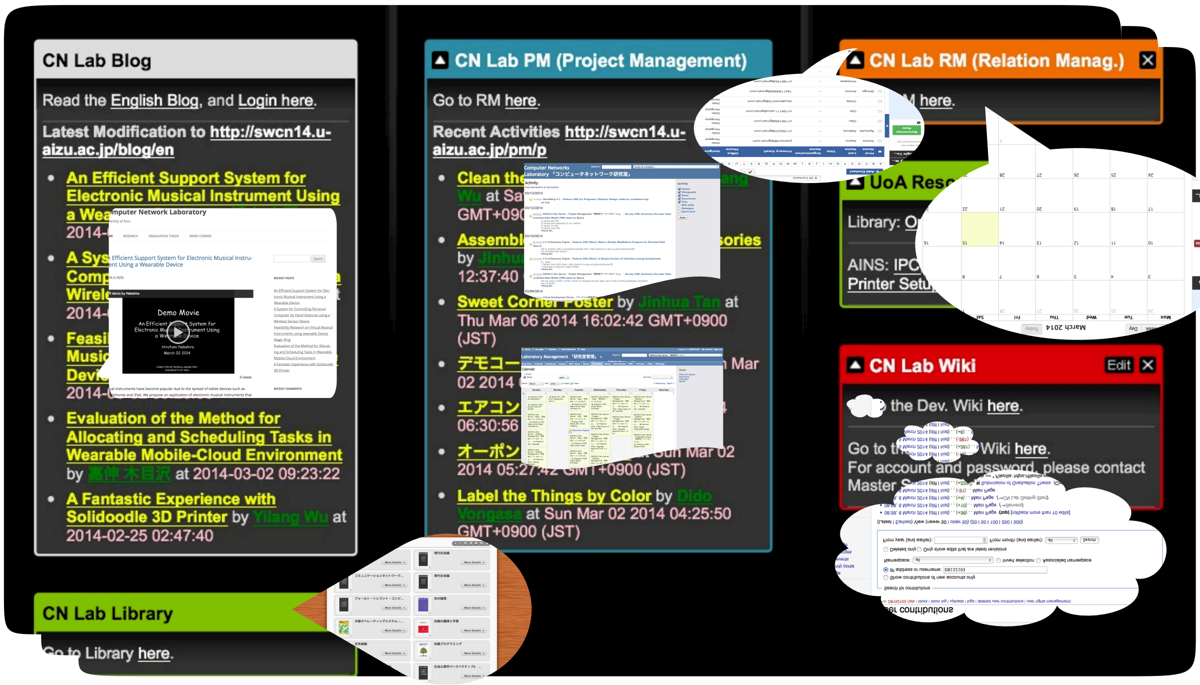
V. Conclusion

In this paper, we first proposed a novel method for searching useful information for a team from big data collected from hybrid clouds. Our method can deal with the inconsistence of the data in different clouds and find relations between data in different clouds based on teams situation. We have made a prototype based on the method, as a research laboratory management system, consisting of project management, relation (including contacts, calendar, and inventory) management, library, and content (blog with research demos and wiki with a lot of technical details) management.

The research is significant, since the era of big data is coming and the current search engines cannot well find the unclear and ambiguous searching request by multiple members of a team. It can also be used for recommendation of events, which should be perfect.

In the future, we will ask more research mates in the laboratories to use the system and get the feedback from them, so as to better improve the system. We will also evaluate our system from the following aspects. (a) response time vs. quality of the results, (b) the granularity vs. elasticity.

1) Deep Fusion of Hybrid Clouds, Our current demo system is still a local cloud service for the first stage. But with the consistent extensions and standard data schema that already prepared, we are sooner or later going to upgrade it as a hybrid-clouds service.
2) Optimize the MapReduce to improve the fitness of the adaptive data resourcing;
3) With the indoor positioning system based Situation Awareness, by using RSSI[22] of BLE (such as iBeacon[23]), we can visualize the activities over the indoor map, see to Figure 6. Since everyone is using smartphone with BLE 4.0+ enabled, it is also possible to introduce automatic checkin checkout service base on RSSI.


CN Lab Blog

Read the English Blog, and Login here.

Latest Modification to http://swcn14.u-aizu.ac.jp/blog/en

- An Efficient Support System for Electronic Musical Instrument Using a Wea 2014-
- A Sys Comp Wirel 2014-
- Feasi Music Devi 2014-
- Evaluation of the Method for Allocating and Scheduling Tasks in Wearable Mobile-Cloud Environment by 嘉倖 木日爪 at 2014-03-02 09:23:22
- A Fantastic Experience with Solidoodle 3D Printer by Yilang Wu at 2014-02-25 02:47:40

CN Lab PM (Project Management)

Go to RM here.

Recent Activities http://swcn14.u-aizu.ac.jp/pm/p

- Clean the Wu at Sa GMT+090
- Assembl by Jinhu 12:37:40
- Sweet Corner Poster by Jinhua Tan at Thu Mar 06 2014 16:02:42 GMT+0900 (JST)
- デモコー 02 2014
- エアコン 06:30:56
- オーボン 2014 05:27:42 GMT+0900 (JST)
- Label the Things by Color by Dido Vongasa at Sun Mar 02 2014 04:25:50 GMT+0900 (JST)

CN Lab RM (Relation Manag.)

here.

UoA Reso

Library: O

AINS: IPC

Printer Setu

CN Lab Wiki

the Dev. Wiki here.

Go to t Wiki here.

For account and password, please contact Master

CN Lab Library

Go to Library here.

Computer Network Laboratory
Demo Movie
An Efficient Support System for Electronic Musical Instrument Using a Wearable Device
Hirofumi Nakahira
March 02 2014
March 2014
Edit

Fig. 4. System User Interface

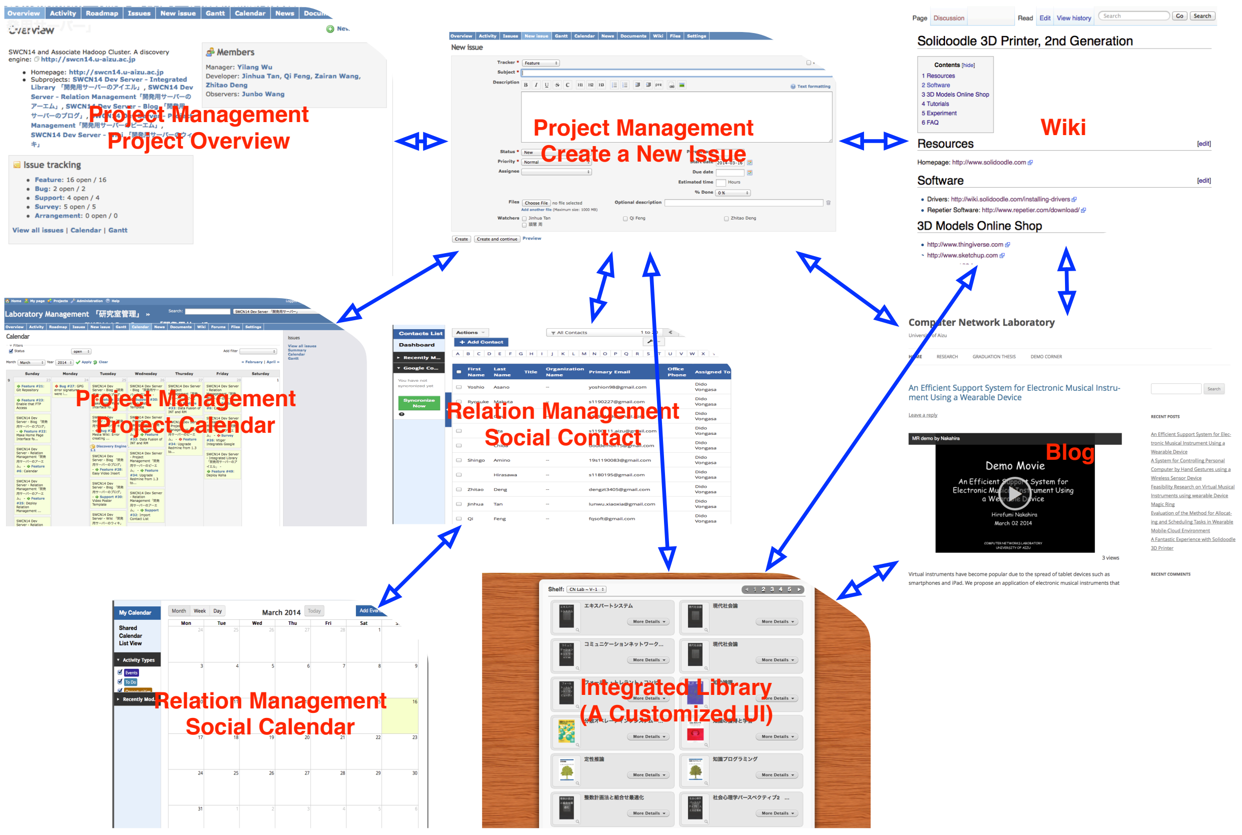


Fig. 5. Data Generation during Teamwork

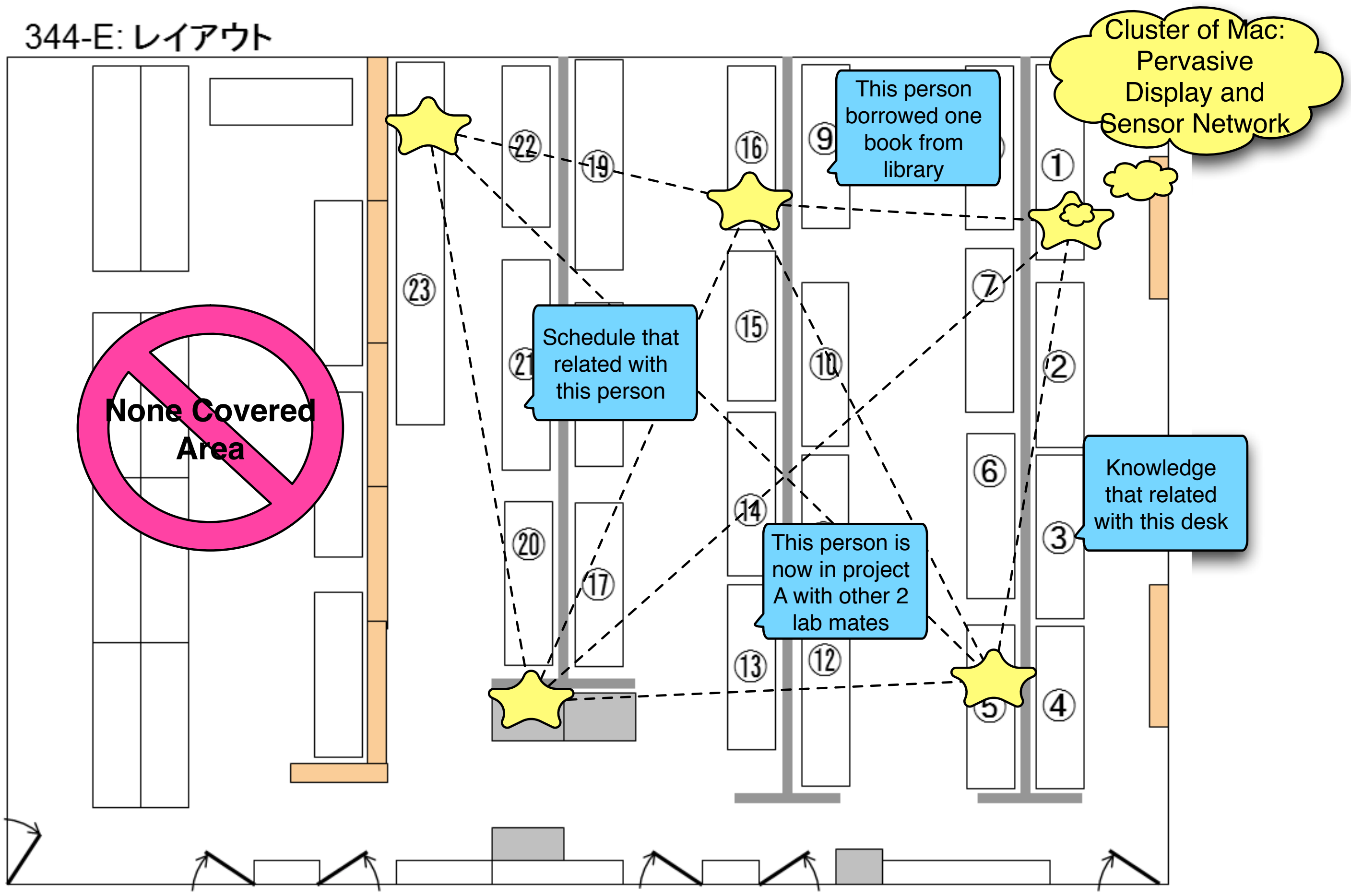

344-E: レイアウト
Cluster of Mac: Pervasive Display and Sensor Network
This person borrowed one book from library
Schedule that related with this person
None Covered Area
Knowledge that related with this desk
This person is now in project A with other 2 lab mates
1
2
3
4
5
6
7
9
10
12
13
14
15
16
17
19
20
21
22
23

Fig. 6. Indoor Map based Situation Awareness

REFERENCES

[1] V. Mayer-Schonberger and K. Cukier, *Big Data: A Revolution That Will Transform How We Live, Work, and Think*. Eamon Dolan/Houghton Mifflin Harcourt, 1 ed., Mar. 2013.

[2] Daniel Wolf, *Prepared and Resolved: The Strategic Agenda for Growth, Performance and Change*. dsb Publishing, 2006.

[3] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer Networks and ISDN Systems*, vol. 30, pp. 107–117, Apr. 1998.

[4] H.-m. Haav, "A Survey of Concept-based Information Retrieval Tools on the Web,"

[5] C. Mangold, "A survey and classification of semantic search approaches," *International Journal of Metadata, Semantics and Ontologies*, vol. 2, no. 1, p. 23, 2007.

[6] Hakia, "Semantic Search Technology, Citing Internet Resource http://company.hakia.com/documents/White %20Paper_Semantic_Search_Technology.pdf," 2010.

[7] D. N. Milne, I. H. Witten, and D. M. Nichols, "A knowledge-based search engine powered by wikipedia," *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management - CIKM '07*, p. 445, 2007.

[8] A. Avetisyan, "LOOK4 : Enhancement of Web Search Results with Universal Words and WordNet," no. 4.

[9] B. V. Keong and P. Anthony, "Meta Search Engine Powered by DBpedia," no. June, pp. 89–93, 2011.

[10] T. Mamachan and R. M. Thanka, "Survey on Keyword Searching in Cloud Storages," vol. 2, no. 11, 2012.

[11] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Enabling Efficient Fuzzy Keyword Search over Encrypted Data in Cloud Computing," pp. 1–16.

[12] S. Fiore, "Personal Communication," 2007.

[13] S. K. Boddhu, R. L. Williams, E. Wasser, and N. Kode, "Increasing situational awareness using smartphones," 2012.

[14] S. K. Boddhu, M. McCartney, O. Ceccopieri, and R. L. Williams, "A collaborative smartphone sensing platform for detecting and tracking hostile drones," 2013.

[15] A. Pavlo, E. Paulson, and A. Rasin, "A comparison of approaches to large-scale data analysis," *. . . on Management of data*, pp. 165–178, 2009.

[16] T. Execution and J. Submission, "Hadoop Map / Reduce Tutorial," pp. 1–33, 2007.

[17] G. Brail and S. Ramji, "OAuth - The Big Picture,"

[18] BuiltWith, "CMS Usage Statistics, Retrieved at March 15th, 2014, http://trends.builtwith.com/cms."

[19] J. Coalo, "With 60 Million Websites, WordPress Rules The Web. So Where's The Money?," 2012.

[20] T. Müller, "How to choose a free and open source integrated library system," *OCLC Systems & Services*, vol. 27, no. 1, pp. 57–78, 2011.

[21] D. Allen, *Getting Things Done: The Art of Stress-Free Productivity*. Penguin Books, first thus ed., Dec. 2002.

[22] I. Engineering, "ENHANCED RSSI-BASED HIGH ACCURACY REAL-TIME USER LOCATION TRACKING SYSTEM FOR INDOOR AND," vol. 1, no. 2, 2008.

[23] J. Manyika, M. Chui, J. Bughin, and R. Dobbs, "Disruptive technologies: Advances that will transform life, business, and the global economy," *McKinsey Global Institute, . . .* , no. May, 2013.

Task Allocation and Node Activation for Energy-Proportional MapReduce Clusters

Min-Ki Kim and Haengrae Cho
Department of Computer Engineering, Yeungnam University, Republic of Korea
E-mail: `hrcho@yu.ac.kr`

Abstract— *MapReduce has emerged as an important paradigm for building large-scale data intensive applications. In this paper, we propose algorithms for task allocation and node activation in MapReduce clusters. They consolidate the system load to a minimum set of active groups, and thus can save energy significantly. Furthermore, they are rack-aware and thus can reduce energy consumption of power-hungry rack components, such as cooling, power distribution units, and power backup equipment. Experiment results indicate a reduction in energy consumption up to 30% when compared to previous algorithms.*

Keywords: Big data, MapReduce, Energy proportionality, Cluster, Node activation

1. Introduction

Large data center operating costs have prompted the consideration of energy management policies. Current solutions can be divided into two categories: *cluster-wide* and *local* [4]. Cluster-wide policies achieve *energy proportionality* by reconfiguring the cluster dynamically. When the cluster load is very high, most nodes should turn on and share the load. However, if the cluster load is significantly lower than the peak load, cluster-wide policies consolidate the load on a subset of nodes and turning off the remaining nodes. As a result, the entire cluster can consume energy in proportion to its load level. Local policies put unused resources in a low-power state. Typical examples include dynamic voltage scaling and power-aware storage management.

This paper focuses on cluster-wide energy management for MapReduce clusters. MapReduce has emerged as an important paradigm for building large-scale data intensive applications [13]. It provides a simple and powerful programming model that enables easy development of distributed applications to process vast amounts of data on large clusters of commodity machines. Recent empirical studies show that MapReduce clusters have to support highly diverse workloads consisting of short interactive jobs and long-running batch jobs [3], [12]. Furthermore, some workload has high peak-to-average ratios [2]. A cluster provisioned for the peak load would be often underutilized and waste a great deal of energy. This means that MapReduce clusters have high potential to save energy by cluster-wide energy management.

In this paper, we assume that the cluster is partitioned into several groups. Each group includes one replica of every data item. To save energy, we exploit both *inter-group strategy* and *intra-group strategy*. The inter-group strategy consolidates system load to a minimum set of active groups so that nodes in other groups can stay at low-power mode for a long time. The intra-group strategy leverages rack-aware node activation to reduce the number of active racks for each group. A recent study on power management for data center shows that rack-based power management can lead to several times more energy savings than other solutions that focus only on the power consumed by node [5].

The contribution of this paper can be summarized as follows.

- We first propose *task allocation algorithms* that select target nodes to execute input tasks. The goal is to consolidate the system load to a minimum set of active groups. We present two alternatives on node selection, group-based and locality-based, which shows an interesting trade-off between energy consumption and performance.
- Then we propose *node activation algorithms* that determines a node to be activated when the system load increases. Considering the activation relationship of group, rack, and node, we propose three node activation algorithms, entire activation, rack-based activation, and locality-based activation. They also exploit a trade-off between energy consumption and performance.
- We develop an experiment model of MapReduce cluster and evaluate the proposed algorithms with respect to energy consumption and performance.

The rest of this paper is organized as follows. Section 2 presents related work and explains its limitations. Section 3 describes the cluster architecture that we assume in this paper. Section 4 proposes algorithms for task allocation and node activation. Section 5 describes the experiment model and Section 6 analyzes the experiment results. Finally, the concluding remarks appear at Section 7.

2. Related Work

A number of studies have reported cluster-wide energy management methods for MapReduce clusters. The *covering set* (CS) method [9] is the initial work. It keeps one replica

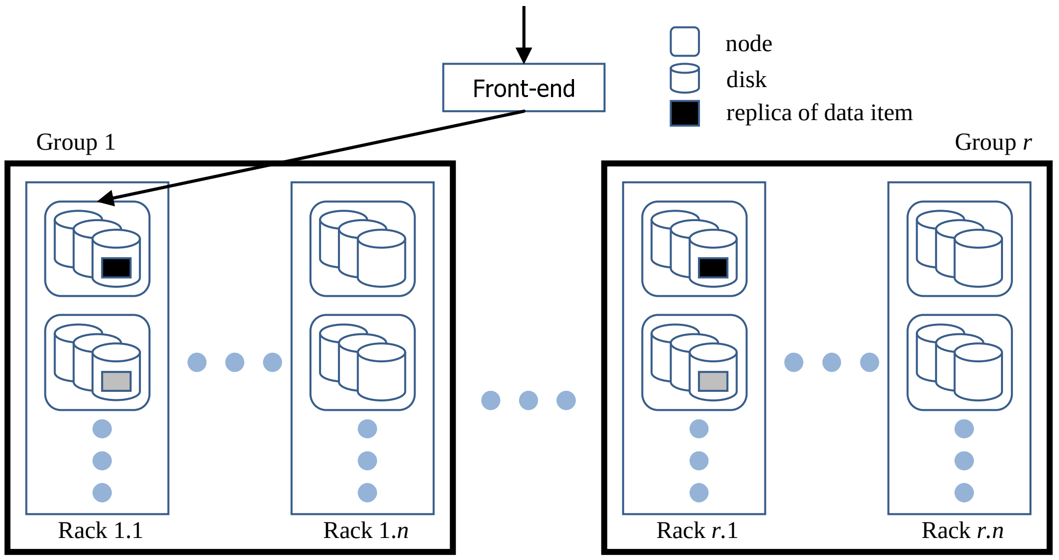


Fig. 1: Cluster architecture

of every data item within a small subset of nodes called CS nodes. CS nodes remain fully powered to preserve data availability while the rest is powered down. iPACS [7] presents a CS node set discovery algorithm that finds an energy-optimized node set with any required degree of data availability. When the system load changes significantly, it activates the new CS node set according to the current load.

Sierra [14] motivates our work the most. It extends the CS method by partitioning the cluster into r groups, where r is a replication factor. Each node belongs to one of the groups so that one replica of each data is placed to each group. Similar to our work, Sierra reconfigures the cluster by activating or deactivating each group according to the system load. However, it may suffer from two limitations. First, it lacks an elaborate task allocation scheme that determines which nodes are going to execute incoming tasks. As we will describe at Section 4, the number of idle nodes can be different according to the task allocation algorithm, especially when the variance of cluster load is significant. By maximizing the number of idle nodes, we can increase the probability of deactivating racks and groups. The next limitation of Sierra is that it does not consider the order of node activation for each group. We will show that activating nodes at random order causes unnecessary energy consumption and longer execution time for each task.

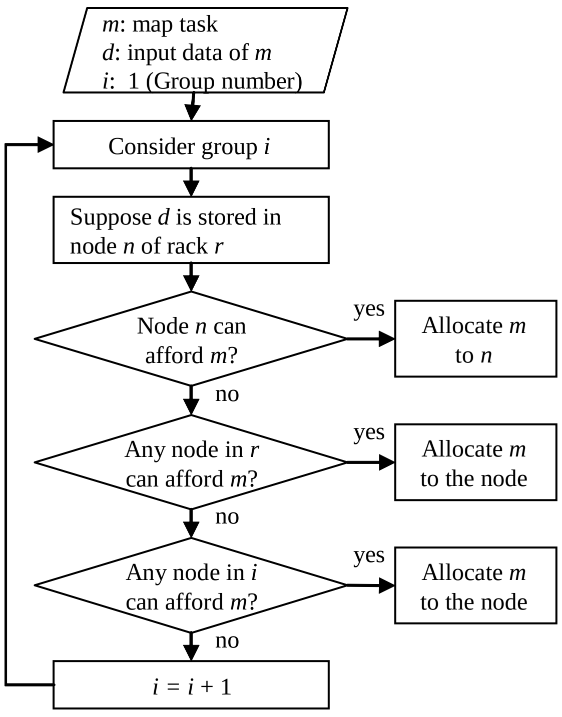
GreenHDFS [6] and BEEMR [2] also divide the cluster into several groups. Unlike the CS method, each group includes different set of data. This causes a data availability problem when a node in some active group requires any data stored in deactivated groups. AIS (All-in Strategy) [8] is a totally different approach. It runs the given jobs with the entire set of nodes in the cluster to complete them as quickly as possible. Upon completion of the jobs, every node is deactivated to save energy until the next run. One potential drawback is that even with small jobs, AIS still needs to wake up the entire cluster, possibly wasting energy. Furthermore, it cannot support real-time jobs that require fast response time and deadline constraint [1].

3. Cluster Architecture

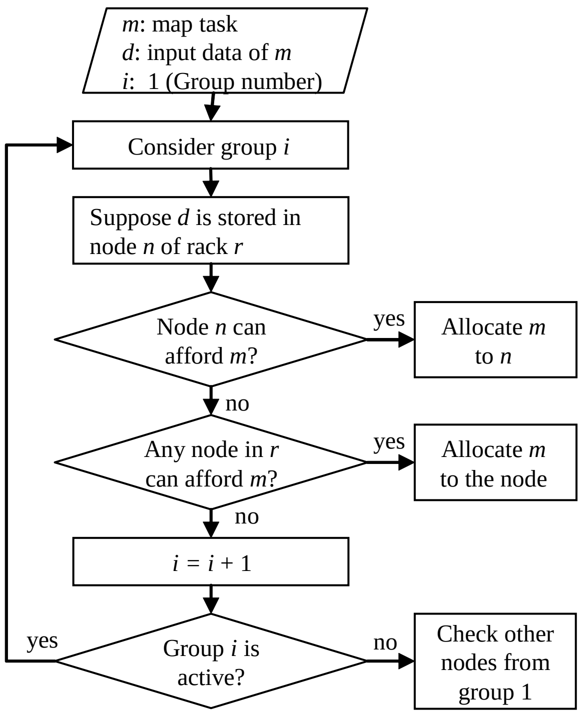
Figure 1 shows the baseline cluster architecture. The cluster is partitioned into r groups, and each group stores one replica of every data item. This means that there are r replicas for each data item. Group 1 is a *primary group*. It has a role to CS nodes [9] and thus always powered on. As a result, the primary group can ensure the immediate availability of data items required for real-time jobs. Other groups may be activated or deactivated according to the system load. This way we can support the principle of *energy proportionality* such that the amount of energy consumed by the cluster is proportional to the amount of work performed.

We assume that each group consists of n distinct racks and all nodes in a rack are in the same group. This permits the entire rack to be turned off when the owner group is deactivated, allowing additional power savings by turning off rack-wide equipment such as switches [5], [14]. We also assume that there is more bandwidth available between nodes on the same rack compared to nodes on different rack. Note that this assumption is effective for most configurations of current data centers. For example, Hadoop prefers within-rack transfers to off-rack transfers when placing MapReduce tasks on nodes [15].

The front-end is a single point of contact for a client wishing to execute a MapReduce job. Typically, a MapReduce job is divided into a number of map tasks and reduce tasks. The front-end allocates each task to a node in some



(a) Group-based allocation (GBA)



(b) Locality-based allocation (LBA)

Fig. 2: Task allocation algorithms for map task

active group. The task allocation has to be performed in an energy-efficient manner not to activate excessive groups compared to the current load. Furthermore, it tries to take advantage of data locality to reduce the amount of inter-node or inter-rack communications. We will describe the proposed task allocation algorithms at Section 4.1. The front-end also monitors the system load and compares it to the processing capacity of active groups. If the system load increases, the current set of active nodes would not afford additional tasks. Then the front-end has to turn on new nodes. There are some alternatives to select new nodes to be activated. We will discuss this issue at Section 4.2.

4. Proposed Algorithms

In this section, we first describe task allocation algorithms that determine a node to execute a task. Then we propose node activation algorithms that determine a node to be activated to process excessive load.

4.1 Task Allocation Algorithms

We propose two task allocation algorithms, named *group-based allocation* (GBA) and *locality-based allocation* (LBA). Figure 2 presents the details of both algorithms for allocating a map task. They take advantage of data locality to reduce inter-node and inter-rack communications. Specifically, they first check if the task is *data-local*, that is, running on the same node that the required data resides on. If the node has available slots for the task, the task can be allocated to the node. Otherwise, both algorithms check if the task is *rack-local*: on the same rack, but not the same node, as the input data. Some task is neither data-local nor rack local. For the task, GBA checks if any other node in the current group is available to execute it. On the other hand, LBA checks if the task is data-local or rack-local for the next group. If it is not data-local and rack-local for every active group, then LBA checks the availability of other nodes from group 1 again.

Note that GBA allocates a task to any node in group 2 only if no node in group 1 can afford new tasks. This means that GBA can consolidate the system load to the limited number of groups. As a result, more groups can be deactivated and the amount of energy reduction would be significant. LBA adopts a different approach. It allocates the task to group 2 in case of data-local or rack-local even though some nodes in group 1 are available. Since LBA takes advantage of data locality more significantly, it could outperform GBA with respect to the execution time. However, more groups would be activated at the same time and thus the amount of energy reduction is limited.

Allocating reduce tasks are rather simple. To exploit data locality, they can be allocated to the same node or same rack of preceding map tasks. If the node or the rack is not available, the front-end will simply take any available node from group 1.

4.2 Node Activation Algorithms

If the set of active groups cannot afford current system load, the front-end has to activate a new group. Considering to the activation relationship of group, rack, and node, we propose the following node activation algorithms.

- **Entire activation (EA)**: When a group is activated, every rack and node in the group is activated at the same time.
- **Fractional activation (FA)**: When a group is activated, only part of entire racks and nodes of the group are activated according to the system load. To select racks and nodes to be activated, the front-end considers the required data of yet-to-be-run tasks. There are two algorithms of the fractional activation.
  - **Rack based activation (FA-RBA)**: It first turns on a rack that includes the most required data of yet-to-be-run tasks. Within the rack, nodes are activated in order according to the system load. Only if every node in the active racks is activated, FA-RBA turns on a new rack.
  - **Locality based activation (FA-LBA)**: It selects an inactive node that includes the most required data of yet-to-be-run tasks. If the node is included in inactive rack, then FA-LBA turns on the rack first and then the node.

Note that those algorithms show an interesting trade-off between energy consumption and performance. First, EA should spend energy the most since all members of the group are activated regardless of the system load. However, it can cope with the following increase of system load without any turn-on delay. FA activates only part of the group according to the system load. If the system load increases thereafter, the rest should be activated with considerable turn-on delay. This means that FA has some limitation to adapt dramatic load variation. However, it can save energy significantly compared to EA.

The trade-off between energy consumption and performance also exists between FA-RBA and FA-LBA. FA-RBA tries to reduce the number of active racks. Since 70% of data center power goes toward rack-related components, such as cooling, power distribution units, the switch gear, and power backup [5], reducing the number of active racks must contribute to save energy significantly. On the other hand, FA-LBA just activates nodes that store the required data the most regardless of rack activation. As a result, it is possible that many racks of a group are activated while there are only a few active nodes for each rack. FA-LBA increases the probability of data-local at the cost of large energy consumption.

5. Experiment model

To evaluate the performance of proposed algorithms, we developed a simulation model of the MapReduce cluster using CSIM discrete-event simulation package [10]. Table 1 shows the simulation parameters. The parameter values will be used for most experiments unless otherwise noted. Many of their values are adopted from [5].

Table 1: Simulation parameters

Parameters	Values
Number of groups	3
Number of racks	150
Number of nodes	3000
Number of Map slots per node	10
Number of Reduce slots per node	5
Rack transition time	300 sec
Node transition time	30 sec
Maximum duration of staying at idle state	15 min
Energy consumption of idle node	100 w/h
Energy consumption of active node	500 w/h
Rack power overhead for each node	50%
Fixed energy consumption of active rack	5 kw/h
Intra-rack network bandwidth	64 Gbps
Inter-rack network bandwidth	128 Gbps
Data size of each group	80 TB
Load skew factor (θ)	0.0 $\sim$ 1.0
Portion of hot data	4%
Probability of accessing hot data	80%

The cluster consists of three groups and each group has 50 racks. The first group is a primary group and is always powered on. The other groups can be deactivated according to the system load. We assume a homogeneous computing environment where the hardware specification of each device type is same. Specifically, a rack includes 20 nodes, and every node has 10 map slots and 5 reduce slots. That means a node can execute 10 map tasks and 5 reduce tasks at the same time. Each node can stay at one of three execution states: *active*, *idle*, and *off*. An off node does not consume any energy. A node is in active state when it executes some tasks. When the node completes every assigned task, it moves to the idle state. We assume that the energy consumption of idle node is much lower than that of active node. Furthermore, if any node in non-primary group stays at the idle state for longer than the threshold (15 min), the front-end turns off the node. The time taken by the node to go between idle and off states is set to 30 seconds.

The state of rack is modeled similarly. If any node in a rack is active, the rack is also active. If every node in the rack is in off state, it moves to the off state. The transition delay taken by the rack is set to 300 seconds same to [5]. An active rack consumes energy in proportion to the number of active nodes. Similar to [5], we configure rack overhead to be 50%; i.e. the support-infrastructure like cooling system on each rack consumes 50% as much energy as the nodes on the rack. Since the rack may also have equipments like interconnect bay and power backup that spend energy independent of node states, we also model that there is significant fixed energy consumption for each rack [11].

A group contains one replica for each data item. We configure that the data size of each group is 80 TB, and it is evenly distributed to the nodes. To model the access skew,

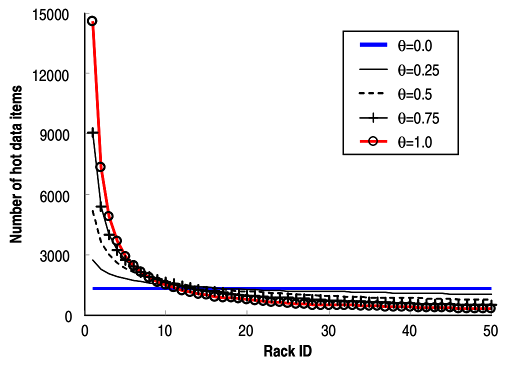


Fig. 3: Distribution of hot data items on various θ

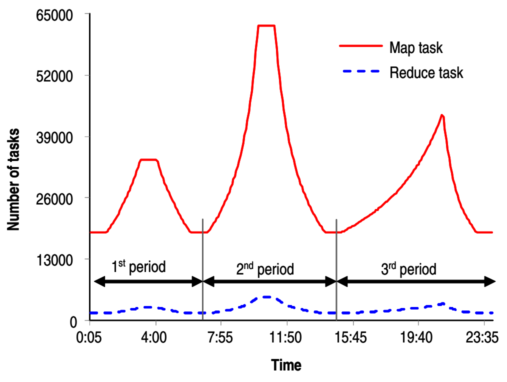


Fig. 4: Load variation model

we categorize the entire data items into *hot set* and *cold set*. A data item in hot set has a high probability of being accessed by tasks. The load skew factor, θ, determines how much portion of hot set is assigned to each rack. Suppose that D_{hot} represents the size of entire hot set. We set D_{hot} to 3.2 TB which corresponds to 4% of the entire data set (= 80 TB). If each group includes N racks, the size of hot set in k-th rack is determined by the following Zipf-like distribution expression, $D_{hot} * \frac{1/k^{\theta}}{\sum_{i=1}^{N}(1/i^{\theta})}$. If θ is set to 0, every rack has the same number of hot data items. On the other hand, if θ is set to 1, the first few racks store most of hot data items. Figure 3 shows the number of hot data items assigned to every rack of a group on different settings of θ. The size of a data item is set to 64 MB.

Figure 4 depicts a load variation model used throughout the experiments. It consists of three periods, each of which has a load increasing stage and a load decreasing stage. The first period lasts for the first seven hours. It is intended to model the initial situation of the cluster where only the primary group is active and no node of the primary group caches anything. The second period starts after the first period. In this period, the load increases up to the maximum cluster capacity and then decreases dramatically. The last period starts near fifteen hours. Unlike the previous periods, the load increases slowly and thus we can compare the performance of proposed algorithms under different load increasing speed. We believe that our load variation model can capture most of real workload characteristics related to MapReduce clusters [2], [6].

6. Experiment Results

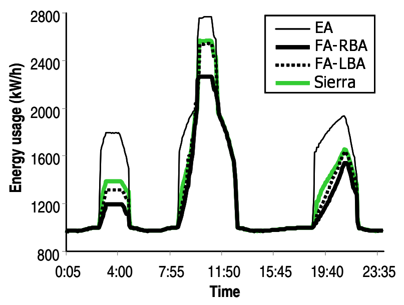
We implement two versions of task allocation algorithms, GBA and LBA, and four versions of node activation algorithms, EA, FA-RBA, FA-LBA, and Sierra [14] that chooses the activated node at random. Performance metrics are *energy usage* and *response time* (turnaround time). The energy usage is the aggregate power consumed by every rack and node. We use the number of active racks and active nodes as a secondary metric to explain the variation of the energy usage. The response time in seconds is measured as the difference between when a task is submitted and when the task successfully commits.

6.1 Energy Usage

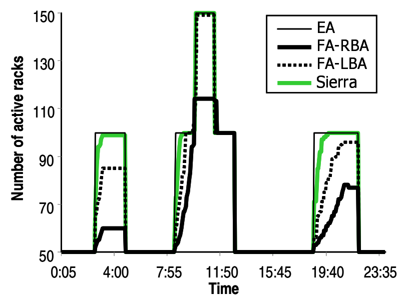
We first compare the energy usage. Figure 5 shows the experiment results. As expected, EA consumes energy the most because it turns on every rack and node in the activated group without regard to the system load. On the other hand, the fractional activation algorithms can save energy by turning on part of racks and nodes of the group according to the system load.

Among the fractional activation algorithms, FA-RBA performs best. It can save energy up to 30% compared to EA as Figure 5.a and 5.b show. This is because FA-RBA turns on a new rack only if all nodes in active racks are active. Figure 5.c and 5.d show that the number of active racks is the smallest at FA-RBA. Both FA-LBA and Sierra consume more energy due to large number of active racks. The difference of the number of active nodes is not significant among the fractional activation algorithms as Figure 5.e and 5.f show. Since Sierra selects a new node at random manner, the selected node may be any rack in the group. This means that Sierra would turn on most racks for each activated group. FA-LBA also turns on more racks compared to FA-RBA. This is because FA-LBA selects a new node that stores the requested data items the most, even though the node is in inactive rack. It prefers to support fast response time rather than reduce energy consumption.

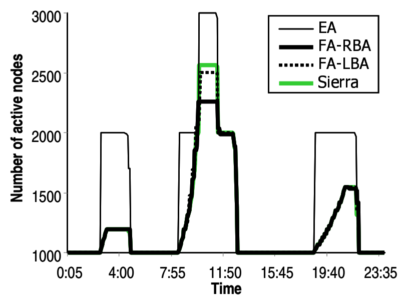
Task allocation algorithms also affect the energy usage. Specifically, LBA consumes more energy than GBA. The difference is significant between the second period and the third period. Since GBA tries to distribute the system load to a minimal number of groups, it has more chances to turn off any group compared to LBA. From Figure 5.c and 5.e, we can guess that GBA activates the primary group only between the second period and the third period since the number of active racks and nodes correspond to the size of a group. On the other hand, as Figure 5.d and 5.f illustrate,



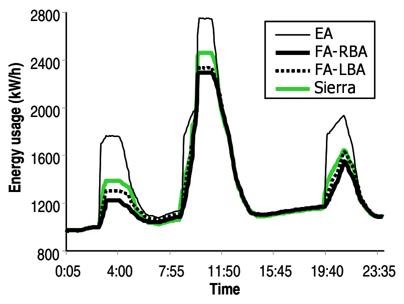
(a) Energy usage (GBA)



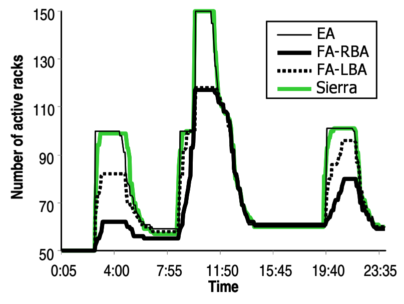
(c) Number of active racks (GBA)



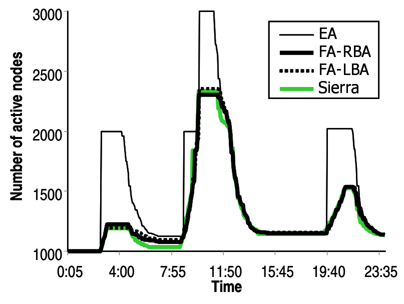
(e) Number of active nodes (GBA)



(b) Energy usage (LBA)



(d) Number of active racks (LBA)



(f) Number of active nodes (LBA)

Fig. 5: Energy usage ($\theta = 0.5$)

LBA turns on more racks and nodes at that duration, which means that LBA activates additional groups.

6.2 Response Time

We also compare the response time of proposed algorithms. Figure 6 shows the difference of response times of proposed algorithms from EA. We select EA as a baseline since EA turns on every rack and node immediately for an active group. As a result, it may not experience any transition delay for any tasks allocated to the group.

For the most part, the differences of response time between node activation algorithms are not significant. This is especially true for GBA as Figure 6.a shows. In LBA, the difference increases at the first period, and Sierra performs worst among them. At the second period, every fractional algorithm performs worse dramatically for some point where the load increases up to the maximum cluster capacity. Since every rack and node in off state should be activated, many tasks experience transition delay waiting for the rack and node will be active. Similar phenomenon also occurs once for FA-RBA at the third period. FA-RBA tries to reduce the number of active racks, and thus it may suffer from large transition delay when the system load increases suddenly and requires many racks being activated.

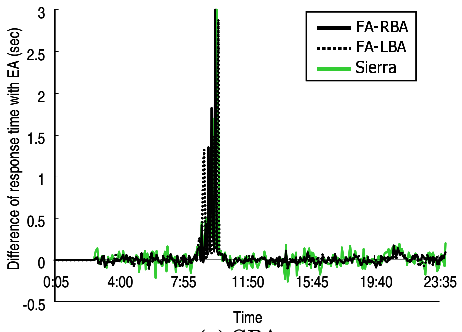
6.3 Access Skew

The last experiment compares the energy usage on various settings of θ. Figure 7 shows the average energy usage at the first period. When θ is 0, FA-RBA performs best and it can save energy about 20% of EA. FA-LBA and Sierra consume much energy compared to FA-RBA. Since every rack of a group has the same number of hot data items when θ is 0, the access set of tasks distribute the entire racks. This means that FA-LBA turns on most racks while very few nodes would be activated at each rack. As a result, the fixed energy consumption of active rack takes significant part of aggregate energy consumption. FA-RBA can reduce the fixed energy consumption since it tries to minimize the number of active racks.

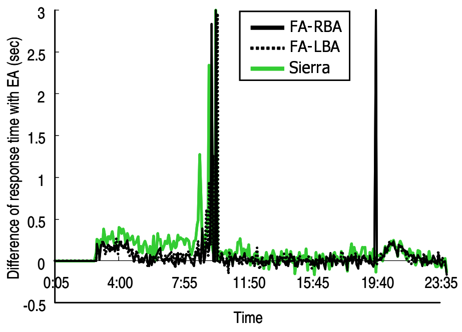
The energy consumption of FA-RBA increases in proportion to θ, and it consumes more energy than FA-LBA when θ is 1. Note that the first few racks store most of hot data items when θ is 1. Let us suppose that 'hot rack' is one that stores hot data items the most. FA-LBA assigns a task to the hot rack only if it accesses hot data items. The execution time of the task should be relatively short since it does not incur inter-rack communication. On the other hand, in FA-RBA, the hot rack may be assigned to tasks accessing other data items. This causes to make the execution time of the tasks longer. Since many tasks have to access data items in hot rack, the execution time of the tasks may be also prolonged. As a result, more racks should be required to provide map/reduce slots for incoming tasks and thus FA-RBA may consume much energy.

7. Concluding Remarks

In this paper, we consider cluster-wide energy management for MapReduce clusters. We first propose task allocation algorithms that select target nodes to execute incoming tasks. Then we propose node activation algorithms that

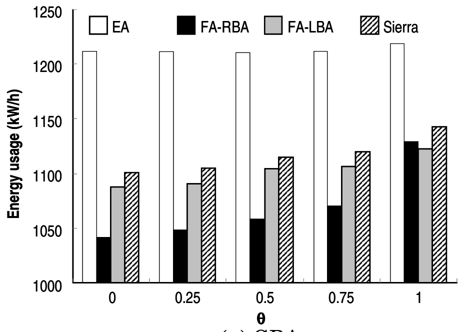


(a) GBA

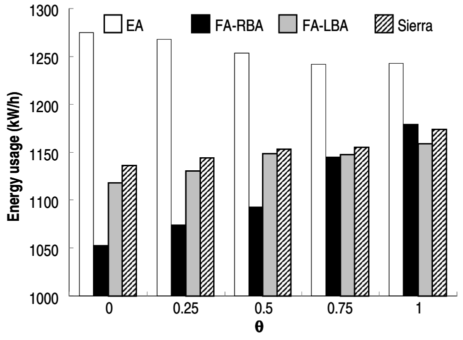


(b) LBA

Fig. 6: Difference of response time from EA ($\theta = 0.5$)



(a) GBA



(b) LBA

Fig. 7: Average energy usage on various θ

determine a new node to be activated when the system load increases. The proposed algorithms consolidate the system load to a minimum set of active groups, and thus can save energy significantly. Furthermore, they are rack-aware and thus can reduce energy consumption of power-hungry rack components, such as cooling, power distribution units, and power backup equipment.

To evaluate the performance of proposed algorithms, we develop a simulation model of MapReduce clusters. The important experiment results are summarized as follows. First, the group-based allocation can reduce the number of active racks and thus can save energy considerably compared to the locality-based allocation. Next, the rack-aware fractional activation outperforms other node activation algorithms with regard to the energy usage. The performance improvement is up to 30% when compared to entire activation algorithm. Furthermore, its response time is comparable to other algorithms in most cases.

Acknowledgement

This work is supported by BK21 Plus program of Korea Ministry of Education and National Research Foundation of Korea.

References

[1] Borthakur D, et al (2011) Apache hadoop goes realtime at Facebook. In: Proc 2011 ACM SIGMOD. pp 1071-1080.
[2] Chen Y, Alspaugh S, Borthakur D, Katz R (2012) Energy efficiency for large-scale MapReduce workloads with significant interactive analysis. In: Proc 7th ACM European Conf Computer Syst (EuroSys'12). pp 43-56.
[3] Chen Y, Alspaugh S, Katz R (2012) Interactive analytical processing in big data systems: a cross-industry study of MapReduce workloads. J Proc of VLDB Endowment 5(12):1802-1813.
[4] Cho H (2012), Energy management for a real-time shared disk cluster. J Supercomputing 62:1338-1361.
[5] Ganesh L, Weatherspoon H, Matian T, Birman K (2013) Integrated approach to data center power management. IEEE Trans Computers 62(6):1086-1096.
[6] Kaushik R, Bhandarkar M, Najrstedt K (2010), Evaluation and analysis of GreenHDFS: a self-adaptive, energy-conserving variant of the Hadoop distributed file system. In: Proc 2nd IEEE Int Conf Cloud Comp Tech and Sci (CloudCom'10). pp 274-287.
[7] Kim J, Chou J, Rotem D (2014) iPACS: Power-aware covering sets for energy proportionality and performance in data parallel computing clusters. J Parallel Distrib Comput 74:1762-1774.
[8] Lang W, Patel JM (2010) Energy management for MapReduce clusters. J Proc of VLDB Endowment 3(1-2):129-139.
[9] Leverich J, Kozyrakis C (2010) On the energy (in)efficiency of hadoop clusters. Operating Syst Review 44(1):61-65.
[10] Mesquite Software, Inc. (2009) User's guide of CSIM20 simulation engine.
[11] Patil V, Chaudhary V (2013) Rack aware scheduling in HPC data centers: an energy conservation strategy. Cluster Comput 16:559-573.
[12] Ren K, Kwon Y, Balazinska M, Howe B (2013) Hadoop's adolescence - an analysis of Hadoop usage in scientific workloads. J Proc of VLDB Endowment 6(10):129-139.
[13] Sakr S, Liu A, Fayoumi A (2013), The family of MapReduce and large-scale data processing systems. ACM Comp Surveys 46(1):1-44.
[14] Thereska E, Donnelly A, Narayanan D (2011), Sierra: practical power-proportionality for data center storage. In: Proc 6th ACM European Conf Computer Syst (EuroSys'11). pp 169-182.
[15] White T (2012), Hadoop - The definitive guide (3rd Edition). O'Reilly.

SESSION

SCIENTIFIC COMPUTING, FFT, MATRIX PROBLEMS, LINEAR SYSTEMS, FINITE ELEMENT METHODS

Chair(s)

TBA

Parallel Processing of Irregular Workloads on the GPGPU: Adaptive Quadrature

Derek Kern and Gita Alaghband
Department of Computer Science and Engineering, University of Colorado Denver, Denver, CO, USA
{derek.kern, gita.alaghband}@ucdenver.edu

Abstract—*This paper presents a parallel (GPGPU) approach for dealing with the turbid workload of adaptive quadrature, called 'parallel block-cutting adaptive quadrature' (PBCAQ). PBCAQ provides speedups as high as 211 times the performance of its sequential competitors. In addition, it has two intertwined and desirable properties: (1) its speedups increase as the size of the workloads being processed increase; and (2) it performs best over definite integrals requiring larger workloads. These two properties together make PBCAQ a valuable example of computing an inequitable, turbid workload on the GPGPU, devices that require workload simplicity.*

Keywords: Parallel processing, GPU, CUDA, SIMT, adaptive quadrature, numerical integration

1. Introduction

In this paper we explore an efficient implementation of adaptive quadrature (AQ) on GPGPU architectures. This problem is selected as an example of an algorithm that exhibits an unpredictable workload and poses challenges in equitably dividing work. Problems with these characteristics are generally difficult to parallelize effectively for the SIMT (Single Instruction Multiple Thread) parallel model of GPGPUs. This computing model offers significant performance benefits for applications with predictable, regular patterns of parallelism and computation, where a single instruction can be applied to many data items at the same time (within GPGPU *threads*).

On GPGPUs, the instruction sequences, called *kernels*, are launched by the CPU onto the GPGPU forming groups of parallel threads, called *warps*, that will execute concurrently on GPGPU *streaming multiprocessors* [1], [2]. Losses of GPGPU computing efficiency occur when: (1) computing units sit idle, which happens when loads are not properly balanced; (2) threads within warps diverge; and (3) warps sit idle during memory accesses. The additional flexibility that comes from the SIMT model and GPGPU architectures cannot easily be exploited without detailed knowledge of such facets [3].

Some workloads, as that of AQ, are intrinsically difficult to conform with processing workloads suitable for the GPGPUs. Often these workloads are resistant to simple or equitable division. In some instances, it may be because the elements of the workload are not uniform and cannot be further divided into uniform elements. In other instances, it may be because dependences that cannot be discovered statically make workload division difficult; this type of workload is known as an *amorphous* workload [4], [5].[1]

Another reason that a workload may not easily conform to GPGPU processing is because the amount of work left to be done cannot be simply circumscribed. In these cases, the task of deciding upon equitable work divisions, needed for parallel load balancing, is either too costly or not possible in principle. In this paper, we introduce the term *turbid workload* to refer to this type of unpredictable workload. Breadth-first search is a good example of an algorithm with such a workload [6].

Adaptive quadrature (AQ) is another example of an algorithm with a turbid workload. AQ is a divide and conquer process that is used to refine the approximation of definite integrals, the area under the curve of a function over a specific interval [10]. It works by first estimating an approximation of the area under the curve for the given interval; this approximation is checked for accuracy (within a given tolerance); if the integral is not within the tolerance, the interval is divided in half and each subinterval is approximated recursively; accurate integrals calculated for subintervals are accumulated. During this process, integrals are approximated using methods like the trapezoidal rule or Simpson's rule. Each time the interval is divided and work preserved for later processing, it is unclear how much work is left within each division. Thus, throughout the processing of AQ, divisions of workload are unlikely to be equal. These types of problems pose a serious challenge to parallel speedup and efficiency for SIMT-type parallelism.

AQ has many applications. Among them are holographic interferometry [7], multilevel regression models [8], and free-surface motion in liquids [9]. Any of these applications and many others would greatly benefit from a GPGPU accelerated form of adaptive quadrature.

In this paper, a parallel algorithm for GPGPU processing of the turbid workload of adaptive quadrature will be shown. It is called 'parallel block-cutting adaptive quadrature' (PBCAQ). PBCAQ is an effective approach for parallelizing adaptive quadrature. It provides significant speedups over sequential competitors; on some definite integrals, these speedups can reach as much as 211 times.

This paper is organized as follows. In Section 2, the basics

[1]The concept of an amorphous workload is closely tied to the concept of *amorphous data-parallelism* put forward by Kulkarni et al [4], [5]. Whereas amorphous data-parallelism refers to the pattern of parallelism exhibited by an algorithm, an amorphous workload refers to the workload resulting from an algorithm whose pattern of parallelism is amorphous data-parallelism.

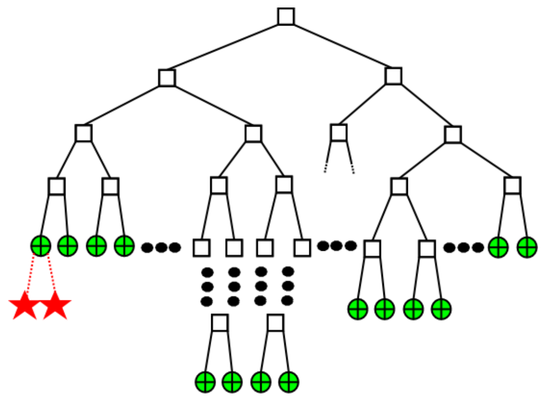


Fig. 1: Computational tree for adaptive quadrature after values of $f(x)$, a, b, and τ are fixed. White boxes represent *mediate computations*; green circles represent *immediate computations*; red stars represent *unnecessarily fine immediate computations*.

of adaptive quadrature are described. Section 3 introduces the continuity assumption, which is the animating principle behind PBCAQ. Section 4 provides a detailed explication of the PBCAQ algorithm. In Section 5, the results for PBCAQ are presented in comparison to sequential AQ versions as well as modified forms of the PBCAQ algorithm. This paper concludes with Section 6.

2. Adaptive quadrature

As stated in Section 1, adaptive quadrature (AQ) is a divide and conquer process that is used to refine the approximation of definite integrals. Let $\int_a^b f(x)\,dx$ be the integral being evaluated using AQ over interval $i = [a, b]$ for a given approximation tolerance τ. During the integration process, subintervals of i, defined as $i_m = [a_m, b_m]$, will be approximated within tolerance τ_m, where τ_m is τ divided as many times as the subinterval of i and $a \le a_m < b_m \le b$. For each interval i_m, AQ evaluates the interval using a coarse approximation method and a fine approximation method.[2] Let s_c be the approximation returned by the coarse method and let s_f be the approximation returned by the fine method. If $|s_c - s_f| \le \tau_m$, then i_m is approximated within tolerance and added to the overall result; if $|s_c - s_f| > \tau_m$, then i_m is divided, typically into two halves, and AQ approximates each of the divisions. Let the divide and conquer approach to AQ be known as 'common adaptive quadrature' (CAQ).

In order to ease discussion, some new terms are needed. Let a *mediate computation* be the computation of subinterval i_m resulting in an approximation not within tolerance; mediate computations are the empty, white boxes in Figure 1. Note that a mediate computation results in its interval, i_m, being further divided into smaller subintervals $i_{m1}, i_{m2}, ...i_{mn}$ for approximation. Let an *immediate computation* be the computation of subinterval i_m resulting in an approximation within tolerance; immediate computations are the green circles in Figure 1. Let an *unnecessarily fine immediate computation* be an immediate computation of a subinterval i_{m1} within tolerance at length ℓ_1 when a subsuming subinterval i_{m2} of length ℓ_2 (where $\ell_2 = \ell_1 * 2^x, x \ge 1$) exists and i_{m2} can be approximated within tolerance; unnecessarily fine immediate computations are the red stars in Figure 1.

[2]All of the algorithms tested in this paper were implemented using the trapezoidal rule for coarse approximation and Simpson's rule for fine approximation.

3. The Continuity Assumption

The design of PBCAQ rests upon a key assumption. This assumption is formally stated below:

Continuity Assumption: Given some continuous function $f(x)$, an integral approximation method M and a tolerance τ, if $\int_a^b f(x)\,dx$ is being approximated using adaptive quadrature and if subinterval i of length l is approximated by M within τ, then the intervals adjacent to i of length l, $i+1$ and $i-1$, will likely also be approximated by M within τ. Similarly, given some continuous function $f(x)$, an integral approximation method M and a tolerance τ, if $\int_a^b f(x)\,dx$ is being approximated using adaptive quadrature and if subinterval i of length l is *not* approximated by M within τ, then the intervals adjacent to i of length l, $i+1$ and $i-1$, will likely *not* be approximated by M within τ.

The continuity assumption is a useful guide to avoiding some of the mediate computations that are normally visited within the AQ computational tree. With the size of adjacent intervals as starting points, much of the mediate work of repeatedly finding correct interval sizes can be skipped. This means that groups of adjacent intervals can be quickly approximated.

Within Figure 2, the mediate computations that this assumption eliminates can be seen. PBCAQ finds an initial interval size (depth first) and then, by assuming continuity, traverses the leaves of the computational tree (horizontally) until the interval size no longer applies; at which point, it either slightly enlarges or shrinks the interval size (and tolerance); it then continues traverse the leaves of the computational tree at the new interval size.[3] While mediate computations are not eliminated, their number can be mitigated by the continuity assumption.

4. Parallel, block-cutting adaptive quadrature for the GPGPU

4.1 Basic algorithm

PBCAQ is implemented in NVIDIA's Compute Unit Device Architecture (CUDA). As such, CUDA nomenclature will be used throughout. Physically, an NVIDIA GPGPU consists of some number of *streaming multiprocessors* (SM), each of which has some number of *cores*, usually 32.[4] Thus, if an NVIDIA GPGPU has 14 SMs, then it has 448 cores. Logically, the primary unit of computation in CUDA is the

[3]As will be discussed, PBCAQ works on the level of regions (of intervals). However, the continuity assumption applies just the same.

[4]At times, cores will also be referred to as *compute units*.

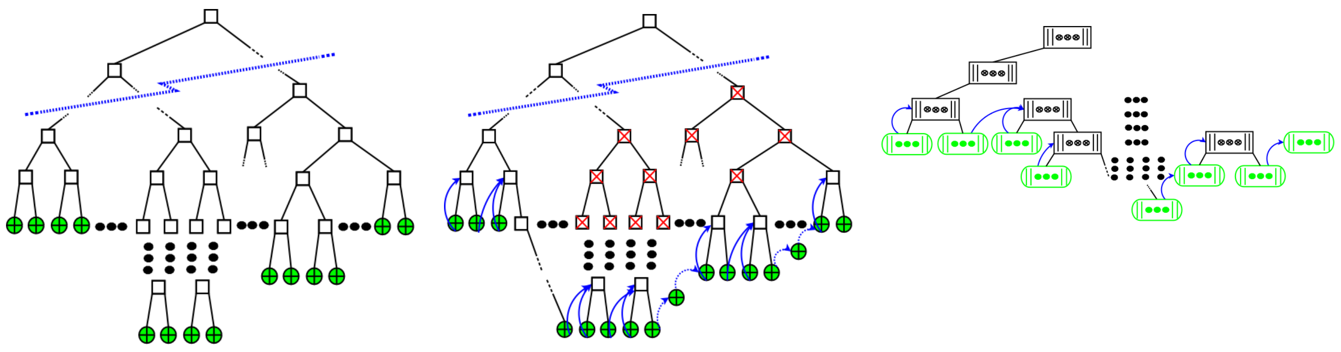


Fig. 2: Comparison of computational trees for CAQ (left), SCAQ* and PBCAQ (right). In CAQ and SCAQ, squares and circles, respectively, represent mediate and immediate computations of intervals; squares with X's represent mediate computations not performed by SCAQ. In PBCAQ, rectangles and ovals, respectively, represent mediate and immediate computations of (intervals within) regions. * Note that SCAQ is described below in Section 4.3

thread such that a single thread will be run on a single core of a single SM. Threads are organized into *blocks* and blocks are organized into *grids*. While any thread can utilize global memory on the device, threads within the same block will execute on the same SM and, thus, are able to utilize shared local memory. Finally, since the number of threads that can be launched on a GPGPU vastly outnumber the number of available cores, threads that are part of the same block are organized into *warps*, which are groups of threads that execute simultaneously on a SM.

AQ algorithms, like CAQ, often work on the level of individual intervals (and subintervals). PBCAQ, on the other hand, works by simultaneously approximating huge numbers of intervals. Let the intervals simultaneously processed on the GPGPU by PBCAQ be known as a *region*. When CAQ approximates an interval i, it compares the error for that approximation to the tolerance to determine whether it is within tolerance. When PBCAQ approximates a region r, it simultaneously approximates all intervals within r on the GPGPU; it then compares the sum of the errors for all intervals in r to the tolerance to determine whether the approximation of r is within tolerance. Thus, PBCAQ never considers whether the approximation of an individual interval of a region is within tolerance; instead, PBCAQ is only concerned with whether a region, as a whole, has been approximated within tolerance. The distinction between the individual intervals processed by CAQ and regions (of intervals) processed by PBCAQ can be seen in Figure 2.

PBCAQ begins by setting $length$, which is the length of the region being approximated, to be the entire length from $lower$ to $upper$. It approximates the first region at level $\log_2(\alpha * \beta)$ (where $\alpha * \beta = 2^x, x \geq 1$) such that α is the number of blocks executed on the GPGPU and β is the number of threads per block. It continues (depth first) to shrink $length$ by half, summing all of the errors from all of the blocks (in the region) approximated on the GPGPU, until it achieves an approximation within tolerance for the entire region. After such an approximation, it adds the region approximation to the total approximation, doubles $length$ and then, moving from $upper$ to $lower$, proceeds to approximate the adjacent region, if there is one, which will continue to be the case until

```
function PBCAQ($f, lower, upper, \tau$)
    Let $\alpha$ be the number of thread blocks;
    Let $\beta$ be the number of threads per block;
    $I \leftarrow 0.0$; $length \leftarrow upper - lower$;
    Let $S$ and $E$ be apprxs and errs;
    while $upper > lower$ do
        ▷ Apprxs and errs are generated by thrds in next step;
        Approximate $length$ on GPGPU using $(\alpha * \beta)$ thrds;
        barrier;
        Reduce apprxs and errs, $S$ and $E$, on GPGPU;
        barrier;
        Transfer apprxs and errs, $S$ and $E$, to host mem;
        ▷ Compare error for region to tolerance
        if $E \leq \tau$ then
            $I \leftarrow I + S$;
            $upper \leftarrow lower$;
            $length \leftarrow 2 * length$;  } ▷ Expansion step
            $\tau \leftarrow \tau * 2$;      }
        else
            $length \leftarrow length/2$;
            $\tau \leftarrow \tau/2$;
        end if
        $lower \leftarrow max(upper - length, lower)$;
    end while
    return $I$;
end function
```

Fig. 3: Parallel, block-cutting adaptive quadrature (PBCAQ)

it breaches $lower$.

Considering the approximation of regions by PBCAQ, let the region being approximated be r_m. PBCAQ (displayed in Figure 3) breaks r_m into $\alpha*\beta$ intervals, where α is the number of thread blocks and β is the number of threads per block. It submits the region to the GPGPU where each thread is assigned its own interval to approximate. Each thread will find both an approximation and an error for its assigned interval.

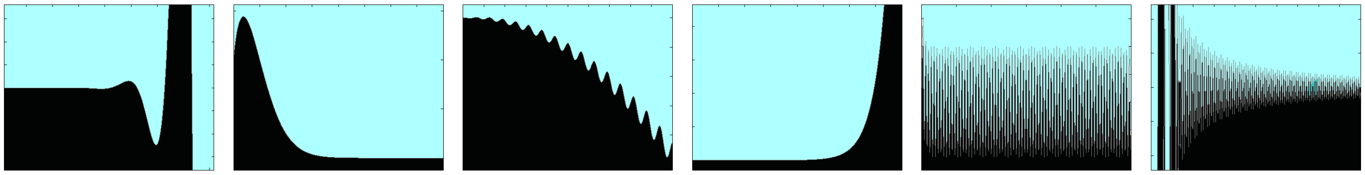
After each of the $\alpha * \beta$ intervals are approximated within the region, the approximations and the corresponding errors

Table 1: Adaptive quadrature versions tested

Name	Description
CAQ	Sequential, queue-based, divide and conquer
SCAQ	Sequential, side-cutting
BCAQ	Sequential version of PBCAQ
PSCAQ	Parallel, side-cutting
PBCAQ	Parallel, block-cutting
PBCAQr	PBCAQ with coin-flip interval doubling
PBCAQs	PBCAQ with error-slope-based interval doubling
PBCAQg	PBCAQ with log of tolerance over error-based interval doubling

Table 2: Functions and intervals integrated over for testing

#	f(x)	Intervals
1	$e^{2x}sin(3x)$	**0.0–5.0**, **0.0–6.0**, **0.0–7.0**
2	$(x^x)^{-1}$	**0.0–1000.0**, **0.0–2000.0**, **0.0–3000.0**
3	$(4x)cos(2x) - (x-2)^2$	**0.0–30.0**, **0.0–50.0**, **0.0–70.0**
4	$e^{4x}(\sqrt{1+e^{4x}})^r - 1$	**0.0–5.0**, **0.0–6.0**, **0.0–6.75**
5	$e^{-3x}cos(5\pi x)$	**0.0–10.0**, **0.0–20.0**, **0.0–30.0**
6	$sin(10\pi x)(\pi x)^{-1}$	**1e-6–10.0**, **1e-6–15.0**, **1e-6–20.0**



Function #1 Function #2 Function #3 Function #4 Function #5 Function #6

Fig. 4: Graphs of functions #1–#6

for these intervals must be reduced (summed) into a single approximation and error bound for the region. This reduction is done primarily on the GPGPU. After it is complete, two highly reduced arrays, one containing errors and the other containing approximations, are transferred back into main memory for final reduction by the CPU.

The performance benefits of PBCAQ are due to several factors. First, no mediate computation takes place for the top part of the computational tree up to level $\log_2(\alpha * \beta)$. This means that approximation of the integral will start for the first region at approximately level 19.[5] Second, using the continuity assumption, we evaluate adjacent equally divided intervals within regions. If a region is approximated accurately, the integral of the entire region is achieved and no further evaluation is needed for this region. Adjacent regions are then approximated accordingly and only divided further if the desired accuracy is not achieved for the region. Third, the abundance of dedicated parallelism in the GPGPU results in small enough intervals within each region to achieve the result within the desired accuracy very fast. In fact, this method would work very well for many core MIMD platforms. Finally, PBCAQ is able to simultaneously approximate entire regions, made up of hundreds of thousands or more intervals. It can do this efficiently because it can spread the approximation of a region across the many threads available on the GPGPU.

[5]This assumes $\alpha = 2048$ and $\beta = 256$. These block and thread allocations values were used for most of the test functions (see Table 2) in this paper.

4.2 Improving the algorithm

Aside from the speedups that come from the simultaneous region approximation of PBCAQ, there is a means of further improving its performance. In Figure 3, note the two bracketed lines colored blue and labeled “Expansion step”. The expansion step executes each time a region is approximated within tolerance. If the next, adjacent region of $length * 2$ can be approximated within tolerance, then the expansion step diminishes (unnecessarily fine) immediate computations. However, if this is not the case, then it increases the number of mediate computations.

What is needed is a way to execute the expansion step when it diminishes the number of immediate computations and skip it when it increases the number of mediate computations. This need gives rise to three further versions of PBCAQ: (1) PBCAQr, which executes the expansion step based upon a random coin-flip; (2) PBCAQs, which executes the expansion step based upon the slope of the errors returned from the GPGPU, i.e. when errors decreased over the region; and (3) PBCAQg, which executes the expansion step based upon the (natural) log of the tolerance over the sum of the errors returned from the GPGPU.

While PBCAQs and PBCAQg have obvious value in that they provide an inductive prediction upon whether the next region of $length$ will be approximated within tolerance, PBCAQr, on the surface seems to be useful only as a point of comparison. However, it has value independent of its comparative value. PBCAQr makes no prediction. Yet, it does provide the means to skip the expansion step half of the time, while still providing the means to 'climb out' of unnecessarily small values of $length$.

4.3 Other algorithms

Before moving to the results, there are three additional AQ algorithms needing description. They were tested along with CAQ and PBCAQ and help to add context to the results. The first is 'side-cutting adaptive quadrature' (SCAQ) [10]. It is sequential and, like PBCAQ, it utilizes the continuity assumption from Section 3. It operates on the level of individual intervals while PBCAQ operates on regions.

The second is 'block cutting adaptive quadrature' (BCAQ). BCAQ is a straightforward sequential version of PBCAQ. The parallel speedups, presented in Section 5, are calculated against the best running time of the three sequential versions, i.e. SpeedUp(P) = min(runtime(CAQ), runtime(SCAQ), runtime(BCAQ)) $\div$ runtime(P). BCAQ was included so that parallel speedups could also be calculated against the sequential version of PBCAQ.

The final additional AQ algorithm to mention is PSCAQ. It is the GPGPU implementation of SCAQ and works on the level of individual intervals, unlike PBCAQ. It provides another basis for comparison.

5. Results

Eight different AQ versions were tested. Three of these versions were sequential while five were for the GPGPU. The complete list of versions tested appears in Table 1. They were all compiled with level three (`-O3`) and fast-math optimizations.

All of the AQ versions were tested against six functions. These functions where chosen because of their varying shapes and the amount of work that their integration engenders. Given that this paper focuses upon workload issues associated with AQ, definite integrals containing singularities were avoided.[6] The size of the workloads associated with each function varied in two ways: (1) by varying the tolerance; and (2) by varying the interval length. For (1), three different tolerances were used. In ascending order of precision, they were: `1e-7`, `1e-8`, and `1e-9`. For (2), each of the functions and their respective interval lengths, in ascending order by interval length, are displayed in Table 2. Graphs of these functions can be seen in Figure 4.

By varying tolerance and interval length, two datasets were generated. In the first, for each function, the tolerance was fixed at `1e-9`, while the interval length was increased through the three values available to each function (shown in Table 2). In the second dataset, the interval length was fixed to the largest value available to each function, while the tolerance was decreased through the values of `1e-7`, `1e-8`, and `1e-9`.[7] Runtimes for each version encompass the entire time of execution, including the data transfer time between GPGPU and CPU memories. All of the runtimes for each dataset were calculated as an average over ten runs.

For the PBCAQ versions, 2048 GPGPU blocks, each with 256 threads, were used with one exception; for function #2, 256 GPGPU blocks, each with 64 threads, were used. These block and thread counts were chosen by running an analysis in which the number of blocks and threads in use was varied, as powers of 2. The block and thread counts that most frequently yielded the best performance for each function were chosen.

The versions were tested on a single core of an `AMD` Opteron 2427 12-core CPU, with 2 `NVIDIA` Tesla Fermi S2050 GPUs (only one GPU was used) running CentOS 5.8. The CPU had 24 Gb of RAM; it also had an $6 * 64$ kb L1 instruction cache and $6 * 64$ kb L1 data cache, $6 * 512$ kb L2 cache and a 6 Mb L3 cache. The `NVIDIA` Tesla Fermi S2050 GPU had 14 streaming multiprocessors each with 32 cores (for a total of 448 cores); it also had 3 Gb of RAM.

From Tables 3 and 4, it can be seen that PBCAQ outperforms all sequential versions and its parallel competitor, PSCAQ. For function #1, its speedup climbs from 49 to 100 to 160 as the interval being integrated over goes from 0.0 to 5.0, 6.0, and finally 7.0. Thus, its speedup climbs as the AQ workload increases (with the length of the interval). The same is true if the workload size is increased through the tolerance (Table 4). In fact, this is the case for all functions #1–#6. For each function, as the length of the interval increases or the tolerance decreases, and, *ipso facto*, the workload size increases, the speedup yielded by PBCAQ increases.

Even though the results in Table 3 appear straightforward, there are few anomalies to discuss. While the speedups for PBCAQ are impressive for functions #1 and #3–#5, the speedups for functions #2 and #6 are less so. The functions themselves are the cause of this behavior. Focusing on function #2, after $x = 2$, it smoothly converges towards zero as x increases. Of course, the relatively flat shape of the curve beyond $x = 2$ means that it can be approximated with few refinements. This fact can be seen in the results for the sequential BCAQ for function #2 in Table 3; even though the interval doubles from 1000.0 to 2000.0, the runtime increases a modest 7 percent from 625.77ms to 665.70ms. Clearly, the workload must not have increased significantly even though the length of the interval most certainly has.[8] This means that PBCAQ is best applied when a definite integral has a sizable workload; some integrands, like function #2 over 0.0–3000.0, do not have sizable workloads, which washes out the benefits of the massive parallelism available on GPGPUs.

As for the versions of PBCAQ that control the execution of the expansion step, PBCAQg outperforms PBCAQ, PBCAQr, and PBCAQs. This can be seen in Figure 5. The idea behind PBCAQg is to skip the expansion step until magnitude of the ratio of the tolerance over the sum of the errors returned from the GPGPU reaches a certain threshold. The prediction

[6]MATLAB was used to determine whether or not definite integrals contained singularities.

[7]Note that decreasing the tolerance increases approximation precision and the workload size.

[8]This fact can also be seen in Table 5. Notice, for PBCAQ, the total regions processed goes from 46, at 0.0–1000.0, to 52, at 0.0–3000.0. Again, this is a very modest increase in the number of regions needed to approximate an interval that is 3 times larger than the original.

Table 3: Results for sequential versions, PSCAQ, and PBCAQ, varying workload by interval length (Tolerance used: `1e-9`)

Function #	Intervals	CAQ	SCAQ	BCAQ	PSCAQ		PBCAQ	
		Time (ms)	*Time (ms)*	*Time (ms)*	*Time (ms)*	*Speedup*	*Time (ms)*	*Speedup*
1	**0.0–5.0**	11,288.87	9,177.48	8,821.56	202.49	43.57	178.98	49.29
	0.0–6.0	34,486.96	26,601.91	22,220.24	612.28	36.29	221.48	100.13
	0.0–7.0	96,409.17	72,395.07	62,780.25	10,338.96	6.07	390.15	160.91
2	**0.0–1000.0**	1,549.33	1,257.47	625.77	1,333.77	0.47	138.85	4.51
	0.0–2000.0	2,336.04	1,869.56	665.70	1,928.50	0.35	136.52	4.88
	0.0–3000.0	2,862.15	2,273.70	737.75	2,331.90	0.32	141.17	5.23
3	**0.0–30.0**	14,972.00	11,075.19	4,584.55	258.03	17.77	160.38	28.58
	0.0–50.0	45,909.05	32,493.22	10,478.06	541.54	19.35	183.58	57.08
	0.0–70.0	89,595.28	63,430.98	16,670.30	1,257.69	13.25	226.93	73.46
4	**0.0–5.0**	4,996.91	3,460.79	3,969.29	189.73	18.24	166.16	20.83
	0.0–6.0	15,268.57	10,316.79	11,405.98	428.92	24.05	229.54	44.95
	0.0–6.75	34,507.96	22,662.23	25,945.51	3,811.10	5.95	333.11	68.03
5	**0.0–10.0**	5,104.17	4,100.58	3,259.67	179.97	18.11	149.08	21.87
	0.0–20.0	6,746.47	5,424.12	6,457.79	5,350.85	1.01	158.97	34.12
	0.0–30.0	13,004.99	10,439.36	9,649.70	10,206.69	0.95	174.25	55.38
6	**1e-6–10.0**	2,186.26	1,602.69	1,434.52	248.41	5.77	143.21	10.02
	1e-6–15.0	3,408.58	2,478.98	1,720.21	440.34	3.91	144.61	11.90
	1e-6–20.0	4,545.15	3,263.59	2,008.42	863.41	2.33	143.49	14.00

Table 4: Results for sequential versions, PSCAQ, and PBCAQ, varying workload by tolerance (Interval used: Largest for each function; see Table 2)

Function #	τ	CAQ	SCAQ	BCAQ	PSCAQ		PBCAQ	
		Time (ms)	*Time (ms)*	*Time (ms)*	*Time (ms)*	*Speedup*	*Time (ms)*	*Speedup*
1	**1e-7**	9,053.00	7,375.65	6,834.82	185.69	36.81	165.01	41.42
	1e-8	31,309.94	25,311.32	20,167.66	526.90	38.28	209.98	96.05
	1e-9	96,233.51	76,043.28	66,083.14	10,244.84	6.45	392.35	168.43
2	**1e-7**	252.41	223.77	360.50	338.45	0.66	136.09	1.64
	1e-8	874.60	734.23	489.66	810.97	0.60	139.20	3.52
	1e-9	2,845.57	2,368.34	743.82	2,329.40	0.32	141.05	5.27
3	**1e-7**	8,426.96	6,353.87	1,772.02	213.07	8.32	141.65	12.51
	1e-8	26,754.30	20,793.99	5,213.23	358.08	14.56	162.72	32.04
	1e-9	89,466.24	65,804.82	17,285.78	1,258.73	13.73	226.60	76.28
4	**1e-7**	3,395.06	2,832.19	2,170.78	173.19	12.53	153.13	14.18
	1e-8	11,004.69	8,751.51	6,877.60	381.23	18.04	200.75	34.26
	1e-9	34,473.11	26,058.89	22,837.65	3,798.32	6.01	334.07	68.36
5	**1e-7**	1,349.30	1,129.17	627.42	1,162.58	0.54	138.52	4.53
	1e-8	3,573.15	2,970.11	2,517.04	2,866.51	0.88	145.83	17.26
	1e-9	12,916.33	10,803.44	9,966.99	10,189.25	0.98	171.30	58.18
6	**1e-7**	449.60	342.45	214.35	301.76	0.71	140.75	1.52
	1e-8	1,440.29	1,093.14	660.54	534.15	1.24	138.98	4.75
	1e-9	4,496.55	3,402.93	2,059.48	859.14	2.40	144.86	14.22

made by PBCAQg has the virtue of being easy to compute and largely accurate at deciding whether the next region would be better approximated at a larger size. It outperforms the other versions of PBCAQ because it processes fewer total regions. This is evident in Table 5 and is the result of avoiding the execution of the expansion step when the next interval cannot be processed at a larger size.

On the flip side, PBCAQs seems less effective at predicting whether the expansion step should be executed for function #2. This can be seen in Figure 5, where the speedup yielded by PBCAQs decreases as the interval size increases. In fact, it is outperformed by PBCAQr, which makes its choice concerning the expansion step randomly. An explanation can be found in Table 5, where the total regions processed by PBCAQs for function #2 is significantly higher than any other version. Comparing regions rejected and total region counts

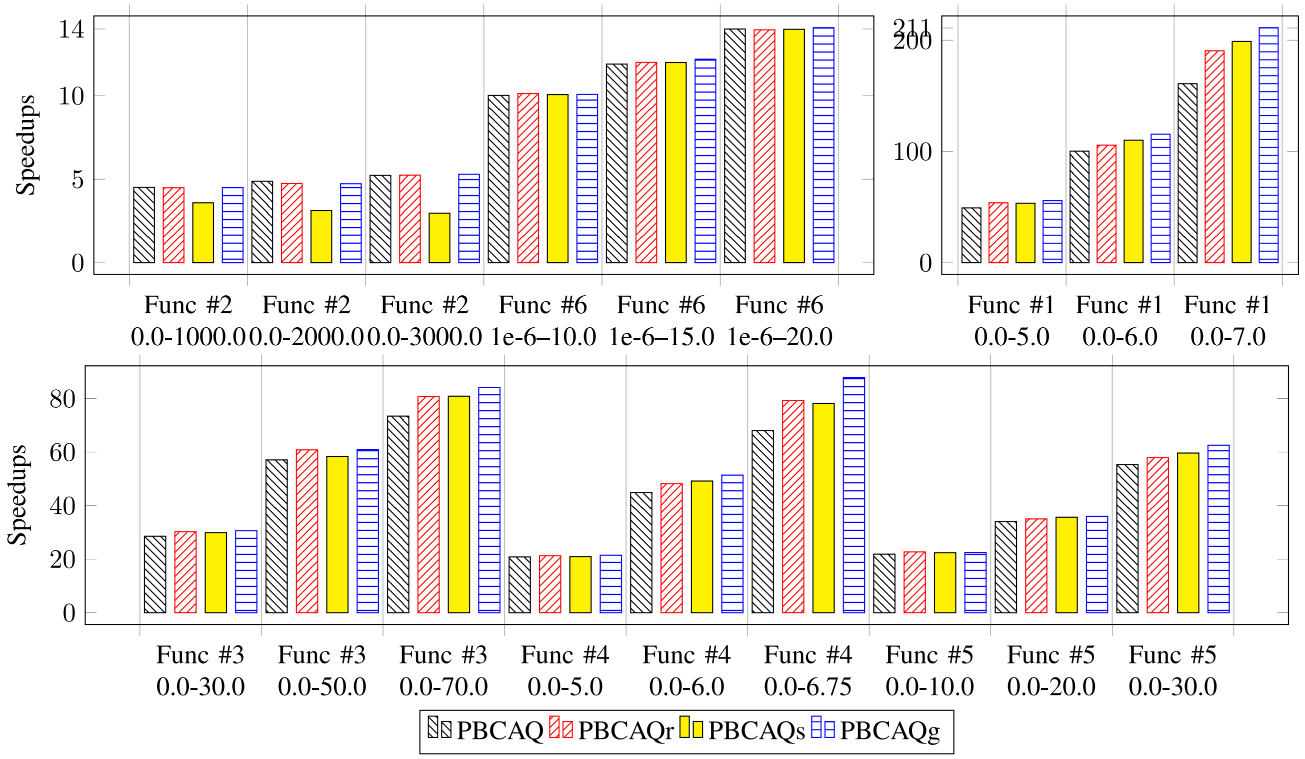


Fig. 5: Parallel speedups of PBCAQ versions, varying workload by interval length (Tolerance used: `1e-9`)

Table 5: Rejected regions and total regions processed for PBCAQ versions, over functions #1 - #4, varying workload by interval length (Tolerance used: `1e-9`)

F#	Intervals	PBCAQ		PBCAQr		PBCAQs		PBCAQg	
		Rej	*Tot*	*Rej*	*Tot*	*Rej*	*Tot*	*Rej*	*Tot*
1	**0.0–5.0**	22	44	13	37	12	38	8	34
	0.0–6.0	56	112	33	99	25	86	15	76
	0.0–7.0	166	332	84	255	66	236	24	210
2	**0.0–1000.0**	25	46	21	51	8	415	21	42
	0.0–2000.0	27	50	21	54	8	815	23	46
	0.0–3000.0	28	52	21	53	8	1215	24	48

of PBCAQs to the other versions, it can be deduced that PBCAQs is opting to execute the expansion step only on rare occasions, forcing it to perform many wasteful unnecessarily fine immediate computations.

6. Conclusion

Parallel, block-cutting adaptive quadrature (PBCAQ) offers significant speedups over the sequential versions of adaptive quadrature. It is also an excellent example of parallelizing turbid workloads for the GPGPU. In the case of adaptive quadrature, the continuity assumption provides needed illumination for how to decompose its workload. While the continuity assumption is not a panacea, it is just enough to help PBCAQ compute adaptive quadrature quickly and efficiently.

There are two next steps for this research. First, the concepts and lessons learned from processing AQ on a single GPGPU will be used to scale PBCAQ across multiple GPGPUs on multiple machines. The *many-task* parallelism offered by NVIDIA's Kepler architecture also presents an interesting target. Second, a more detailed study of turbid workloads will be done by finding other algorithms sharing turbidity similar to adaptive quadrature.

References

[1] N. Wilt, *The CUDA Handbook: A Comprehensive Guide to GPU Programming*. Pearson Education, 2013.
[2] NVidia, *NVIDIA's Next Generation CUDA Compute Architecture: Fermi*, 2011.
[3] S. Hong, S. K. Kim, T. Oguntebi, and K. Olukotun, "Accelerating cuda graph algorithms at maximum warp," *SIGPLAN Not.*, vol. 46, no. 8, pp. 267–276, Feb. 2011.
[4] K. Pingali, M. Kulkarni, D. Nguyen, M. Burtscher, M. Mendez-lojo, D. Prountzos, X. Sui, and Z. Zhong, "Amorphous data-parallelism in irregular algorithms," 2009.
[5] M. Kulkarni, M. Burtscher, R. Inkulu, K. Pingali, and C. Casçaval, "How much parallelism is there in irregular applications?" *SIGPLAN Not.*, vol. 44, no. 4, pp. 3–14, Feb. 2009.
[6] D. Merrill, M. Garland, and A. Grimshaw, "Scalable gpu graph traversal," *SIGPLAN Not.*, vol. 47, no. 8, pp. 117–128, Feb. 2012.
[7] M. Ragulskis and L. Saunoriene, "Order adaptive quadrature rule for real time holography applications." in *Numerical Methods and Applications*, ser. Lecture Notes in Computer Science, T. Boyanov, S. Dimova, K. Georgiev, and G. P. Nikolov, Eds., vol. 4310. Springer, 2006, pp. 685–692.
[8] S. Rabe-Hesketh, "Reliable estimation of generalized linear mixed models using adaptive quadrature," *Stata Journal*, vol. 2, no. 1, pp. 1–21(21), 2002.
[9] Q. Nie and G. Baker, "Application of adaptive quadrature to axi-symmetric vortex sheet motion," 1998. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?rep=rep1&type=pdf&doi=10.1.1.56.4181
[10] P. Gonnet, *Adaptive Quadrature Re-revisited*. Lulu Enterprises Incorporated, 2009.

Embarrassingly Parallel Butterflies Solve Diagonally Dominant Tridiagonal Toeplitz Systems

Brian J. Murphy
Department of Mathematics and Computer Science
Lehman College of the City University of New York, Bronx, NY 10468 USA
brian.murphy@lehman.cuny.edu

Abstract— *We present a parallel tridiagonal Toeplitz solver for diagonally dominant systems. Our solver utilizes a two tiered data decomposition. The first is a course grained and communication free matrix block partitioning. The second is fine-grained and depends on a fast Fourier transform (FFT) kernel. Our algorithm is designed to benefit from the use of parallel FFT processors. Such devices vastly outperform the general architectures on which other tridiagonal solvers are most likely to be implemented in terms of both speed of execution and conservation of energy. Furthermore, ramping up processor counts to limit or even eliminate agglomeration is embarrassingly parallel.*

Keywords: Tridiagonal Toeplitz, Diagonally Dominant, Linear System, FFT, Butterfly.

1. Introduction

The ability to solve a diagonally dominant tridiagonal Toeplitz (DDTT) system of linear equations quickly is central to many computational tasks in the sciences, engineering, signal and image processing, as well as for interactive and real-time systems in graphics and video. These systems are at the heart of problems as diverse as cubic spline and B-spline curve fitting [19], [7], preconditioning for iterative linear solvers [5], [20], computation of photon statistics in lasers [14], computational fluid dynamics [41], solving neuron models by domain decomposition [21], solving second order differential equations and numerical solutions to integral equations [40], [18], and more.

We propose a parallel algorithm to solve a DDTT system of linear equations

$$\begin{pmatrix} d & 1 & & \\ c & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & c & d \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{pmatrix}, \qquad (1)$$

where $|d| > |c| + 1$. Here and hereafter we express (1) as the matrix equation $T\mathbf{x} = \mathbf{b}$, where $n \times n$ tridiagonal Toeplitz matrix $T = (t_{i,j})_{i,j=0}^{n-1}$, such that $t_{i,i-1} = c$, $t_{i,i} = d$, $t_{i,i+1} = 1$, and $t_{i,j} = 0$ otherwise, and n-vectors $\mathbf{x} = (x_i)_{i=0}^{n-1}$ and $\mathbf{b} = (b_i)_{i=0}^{n-1}$. Our algorithm determines a solution vector by carrying out its computation on carefully chosen substitute systems that approximate the original system in question. Its accuracy compares quite well with other methods. It achieves extreme scalability with the finest of granularity by utilizing a two tiered data decomposition. The first is a course grained and communication free matrix block partitioning. The second is fine-grained and depends on a fast Fourier transform (FFT) kernel.

FFTs are the underlying mechanism found in a great many computations [4], [3]. The FFT and inverse FFT (IFFT) compute the discrete Fourier transform (DFT) and inverse DFT (IDFT) respectively [9], both in $O(n \log_2 n)$ arithmetic operations (ops). The FFT lends itself quite naturally to parallel implementations requiring $O(\log_2 n)$ parallel steps. Application specific integrated circuits (ASICs) and Field-Programmable Gate Arrays (FPGAs) incorporating Butterfly circuits allow for efficient parallel computation of FFTs that essentially avoid many of the bottlenecks one must expect to encounter when relying on a general purpose architecture. Our algorithm allows a large number of such devices to perform entirely independent computation in parallel to tackle large linear systems. For a given degree of diagonal dominance it allows throughput to increase in lock step with input size. Competitive parallel solvers lack either the fine granularity of the FFT or require agglomeration for effective utilization of hardware.

The rest of our paper is organized as follows: Background information appears in the next section. We present original work that forms the mathematical basis for our algorithm in Section 3. We present and analyze our algorithm in Section 4. Empirical results are available in section 5. We discuss application of our algorithm to ill-conditioned systems in section 6. We conclude the paper with some final thoughts in Section 7.

2. Background

2.1 Tridiagonal Solvers

The most work efficient known tridiagonal solver is the Thomas Algorithm. It is essentially Gaussian elimination in the tridiagonal case and requires $8n - 4$ arithmetic operations (ops). Unfortunately, the Thomas Algorithm is inherently serial due to data dependencies between successive iterations. Recursive doubling (RD) [38] and cyclic reduction (CR) [17] are examples of fine-grained parallel tridiagonal solvers. Both require $O(\log_2 n)$ parallel steps. CR is, however, the more work efficient of the two requiring $O(n)$ ops compared to RD's $O(n \log_2 n)$ ops. The course-grained SPIKE Algorithm [33] for solving banded systems is another $O(n)$ op parallel solver that has been applied to tridiagonal systems [6].

Recent interest in unleashing the parallel processing power of Graphics Processing Units (GPUs) has lead to the implementation on GPUs of several well known algorithms and hybrid variants thereof for solving tridiagonal systems. CR has a work efficiency advantage over many such parallel solvers. However, GPU idiosyncrasies can undo this advantage [44] unless they are dispensed with effectively [10]. One of the obstacles to implementing an efficient GPU based CR is the fact that its stride doubles each step leading to bank conflicts [44]. Improvements were realized using register packing, which significantly outperformed previous implementations [10]. A CPU based LU decomposition overlapping a necessary data transfer from host to device eliminated much of the communication on the GPU required by CR resulting in a significant speedup [29]. Chang et al. [6] took another approach aimed at distributing independent work to the thread processors of the GPU. They applied the SPIKE Algorithm to a tridiagonal matrix.

2.2 Toeplitz Solvers

Morf [26] and Bitmead and Anderson [2] inverted Toeplitz matrices in $n \log_2^2 n$ ops. In essence they extended the complete recursive triangular factorization of a matrix, that Strassen [39] used to bound the arithmetic complexity of matrix inversion. In doing so they took advantage of the FFT based matrix-vector-multiplication available for structured matrices of Toeplitz type, which is essentially a truncated polynomial-polynomial-product. A nonsingular triangular Toeplitz matrix is invertible in $O(M(n))$ ops [37] based on Newton's iteration, where $M(n)$ is the number of ops required to multiply an $n \times n$ triangular Toeplitz matrix by an n-vector. Over the fields supporting FFT this bound is $O(n \lg n)$. Some minor improvement in the implementation was proposed by Commenges and Monsionin [8] and related work [1], [42], [15], and [36], for which the cost of computation is $(1.\bar{6} + o(1))M(n)$ ops. Additional progress was made by Schönhage [35] and Murphy [27] both presenting algorithms requiring $(1.5 + o(1))M(n)$ ops. Harvey's [16] further improvement requires $(1.\bar{4} + o(1))M(n)$ ops.

2.3 Tridiagonal Toeplitz Solvers

A DDTT matrix T, from (1), can be split into the product of two bidiagonal Toeplitz matrices plus a perturbation term via application of the quadratic formula. The two possible equalities are

$$T = \begin{pmatrix} 1 & & & \\ r_1 & \ddots & & \\ & \ddots & \ddots & \\ & & r_1 & 1 \end{pmatrix} \begin{pmatrix} r_2 & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & r_2 \end{pmatrix} + \begin{pmatrix} (d - r_2) & & & \\ & & & \\ & & & \\ & & & \end{pmatrix} \quad (2)$$

and

$$T = \begin{pmatrix} r_2 & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & r_2 \end{pmatrix} \begin{pmatrix} 1 & & & \\ r_1 & \ddots & & \\ & \ddots & \ddots & \\ & & r_1 & 1 \end{pmatrix} + \begin{pmatrix} (d - r_2) & & & \\ & & & \\ & & & \\ & & & \end{pmatrix}, \quad (3)$$

where $r_1 = \frac{d \pm \sqrt{d^2 - 4c}}{2}$, $r_2 = d - r_1$. Hereafter, we write the matrix equations $T = LU + P$ and $T = UL + P'$ to represent (2) and (3) respectively. Because we focus on diagonally dominant T we can always find r_1 and r_2 such that $|r_1| < 1 < |r_2|$ [31] and so it can be guaranteed that L and U are diagonally dominant as well.

Several algorithms that solve DDTT systems start by solving $LU\mathbf{y} = \mathbf{b}$ via backward and forward substitution, which requires $4n + O(1)$ ops. Most then apply a correction step to $\mathbf{y}$ to find either an exact solution or a good approximate solution to $\mathbf{x} = T^{-1}\mathbf{b} = (LU + P)^{-1}\mathbf{b}$ [34], [43], [22], [24], [23], [31]. The parallel algorithms based on this split/correct technique [22], [24], [23], [31], all require some amount of communication during the correction stage. McNally et al. [25] did away with the correction step and in so doing eliminated communication between partitions of the decomposition. They built on the work by Yan and Chung [43] who recognized that the need for significant corrections was limited to a small leading segment of the approximated solution vector $\mathbf{y}$. McNally et al. [25] noted that in solving $UL\mathbf{z} = \mathbf{b}$, significant differences between $\mathbf{x}$ and $\mathbf{z}$ were limited to a small number of trailing elements. They showed that concatenating the appropriate segments of $\mathbf{y}$ and $\mathbf{z}$ results in a good approximation of $\mathbf{x}$.

The aforementioned parallel solvers perform a course-grained decomposition. Murphy [28] deployed the fine-grained FFT to solve bidiagonal Toeplitz systems. For the diagonally dominant case a higher level course-grained decomposition was performed using a banded approximation to the inverse. This allowed multiple smaller FFTs to stand in for a single larger FFT and obviated the need for communication between partitions. Combining the Toeplitz LU splitting technique with the FFT based bidiagonal Toeplitz solver results in a solver for DDTT systems that will very significantly outperform the corresponding non-FFT versions when executed on FFT processors. Solving a tridiagonal Toeplitz system by solving consecutive bidiagonal Toeplitz system each via FFT/IFFT in this manner, however, leads one to wonder whether a single round of FFT/IFFT can produce the desired result.

2.4 Decay of the Inverse

The inverse of a DDTT matrix, T, decays exponentially at a rate dependent on the values c and d as the distance from its main diagonal increases [11], [30]. Recall $r_1 = \frac{d \pm \sqrt{d^2-4c}}{2}$ and $r_2 = d - r_1$ such that $|r_1| < 1 < |r_2|$ and let ϵ be the machine epsilon defined as the smallest floating point value such that $1+\epsilon > 1$ and $1-\epsilon < 1$ in floating point arithmetic at a given precision. If $\alpha r_1^{\lceil k/2 \rceil} < \epsilon$ and $\alpha r_2^{\lceil -k/2 \rceil} < \epsilon$ where $\alpha = \sum_{i=0}^{\lceil k/2 \rceil} (r_1/r_2)^i$ then beyond a bandwidth k, where k is odd and centered on the main diagonal, the values of T^{-1} are too small to contribute to a numerically computed matrix-vector product. In such computations a k-banded approximation of T^{-1} that matches T^{-1} within the centralized k-band is a sufficient substitute for T^{-1}.

2.5 Circulant Substitution

Let $n \times n$ matrix $Z_1 = (z_{i,j})$ such that $z_{i,i-1} = 1$, for $i = 1, \ldots, n-1$, $z_{0,n-1} = 1$, and $z_{i,j} = 0$ for all other pairs (i,j). Then $Z_1(\mathbf{v}) = \sum_{i=0}^{n-1} v_i Z_1^i$ is a circulant matrix defined by its first column n-vector $\mathbf{v} = (v_i)_{i=0}^{n-1}$. The product of an $n \times n$ circulant matrix $Z_1(\mathbf{v})$ and an n-vector $\mathbf{x} = (x_i)_{i=0}^{n-1}$ can be obtained in $O(n \log_2 n)$ ops and $O(\log_2 n)$ parallel steps via

$$Z_1(\mathbf{v})\mathbf{x} = F_n^{-1}(F_n(\mathbf{v}) \ .* F_n(\mathbf{x})), \tag{4}$$

(see Pan [32], Chapter 2), here and hereafter $F_n(\mathbf{u})$ is the order n DFT of $\mathbf{u}$, $F_n^{-1}(\mathbf{u})$ is the order n IDFT of $\mathbf{u}$, and $\mathbf{u} \ .* \mathbf{w} = (u_i w_i)_{i=0}^{n-1}$ is an element-wise product of $\mathbf{u}$ and $\mathbf{w}$, where n-vector $\mathbf{u} = (u_i)_{i=0}^{n-1}$ and n-vector $\mathbf{w} = (w_i)_{i=0}^{n-1}$.

It is common practice in engineering circles to derive a quick rough estimate for the solution to a Toeplitz linear system by substituting a related circulant matrix for its Toeplitz counterpart [13]. The motivation for this substitution is two fold. Associated Toeplitz and circulant matrices exhibit asymptotic convergence and circulant matrices are diagonalized via the DFT. The latter tells us that the inverse of a circulant matrix can be computed via FFT. This is done efficiently by applying the equality

$$Z_1^{-1}(\mathbf{v}) = Z_1(F_n^{-1}(1 \ ./F_n(\mathbf{v}))), \tag{5}$$

(see Pan [32], Chapter 2), here and hereafter $1 \ ./\mathbf{u} = (1/u_i)_{i=0}^{n-1}$ is the element-wise reciprocal of $\mathbf{u}$. (5) indicates that the set of invertible circulant matrices is closed under matrix inversion so that together with (4) it is clear that $Z_1(\mathbf{v})\mathbf{x} = \mathbf{b}$ can be solved by computing

$$\mathbf{x} = F_n^{-1}((1 \ ./F_n(\mathbf{v})) \ .* F_n(\mathbf{b})). \tag{6}$$

3. The basis for our Algorithm

We exploit a k-banded approximation to T^{-1}. Due to the documented decay in T^{-1}, we can always increase the precision of our approximation by choosing a larger k. For convenience then and without loss of generality we will here and hereafter assume $k = 2^i - 1$ for an integer $i > 0$ and that k is centered on the main diagonal.

We now introduce three matrices depicted in Fig. 1. Here and hereafter, let $U = (u_{i,j})$, where $u_{n,1} = 1$ and $u_{i,j} = t_{i,j}$ otherwise, let $W = (w_{i,j})$, where $w_{1,n} = c$ and $w_{i,j} = t_{i,j}$ otherwise, and let circulant matrix $C = (c_{i,j})$, where $c_{1,n} = c$, $c_{n,1} = 1$, and $c_{i,j} = t_{i,j}$ otherwise. We are motivated to consider these matrices based upon the following lemma and its two corollaries.

Lemma 1: $\mathbf{x}' = U^{-1}\mathbf{b}$ provides a good approximation for all but the trailing $\lceil k/2 \rceil$ elements of $\mathbf{x} = T^{-1}\mathbf{b}$, wherever a k-banded approximation of T^{-1} will suffice.

Proof:

Clearly the first $n - \lceil k/2 \rceil$ elements in the last column of T^{-1} can be treated as zero. We can investigate the values of the corresponding elements of U^{-1} by applying the Sherman Morrison formula. We have

$$U^{-1} = (T + \mathbf{u}\mathbf{v}^{\mathsf{T}})^{-1} =$$
$$T^{-1} - (T^{-1}\mathbf{u}\mathbf{v}^{\mathsf{T}}T^{-1})/(1 + \mathbf{v}^{\mathsf{T}}T^{-1}\mathbf{u})$$

where $\mathbf{u} = \mathbf{e}_{n-1}$, $\mathbf{v} = \mathbf{e}_0$, and $\mathbf{e}_i$ is the ith column of the appropriately sized identity matrix. Clearly $1 + \mathbf{v}^{\mathsf{T}}T^{-1}\mathbf{u}$ is a scalar. $\mathbf{v}^{\mathsf{T}}T^{-1}\mathbf{u}$ is the element in the first row and last column of T^{-1}. This is the smallest value above the main diagonal of T^{-1} and as such can be treated as zero. Therefore $T^{-1}\mathbf{u}\mathbf{v}^{\mathsf{T}}T^{-1}$ is a very good approximation for $T^{-1} - U^{-1} = (T^{-1}\mathbf{u}\mathbf{v}^{\mathsf{T}}T^{-1})/(1 + \mathbf{v}^{\mathsf{T}}T^{-1}\mathbf{u})$. $T^{-1}\mathbf{u}$ is the last column of T^{-1} and it has already been noted that its leading $n - \lceil k/2 \rceil$ elements approach zero. $\mathbf{v}^{\mathsf{T}}T^{-1}$ is the first row of T^{-1}. Clearly its trailing $n - \lceil k/2 \rceil$ elements approach zero. In fact $T^{-1}\mathbf{u}$ and $\mathbf{v}^{\mathsf{T}}T^{-1}$ contain the same elements but in reverse order, due to the persymmetry of a Toeplitz inverse. The largest value of outer product $(T^{-1}\mathbf{u}) \cdot (\mathbf{v}^{\mathsf{T}}T^{-1})$ is in its southwest corner where the perturbation or non-zero element of $T - U$ occurs and its exponential decay radiates from there so that no more than $\lceil k/2 \rceil$ of the southwestern most diagonals of $T^{-1} - U^{-1}$ are significantly different from zero. We see that the first $n - \lceil k/2 \rceil$ elements in the last columns of $T^{-1} - U^{-1}$ as well as T^{-1} approach zero and can be treated as such and therefore the same is true of U^{-1}.

Now, left multiply $\mathbf{x} = T^{-1}\mathbf{b}$ by U. We have $U\mathbf{x} = UT^{-1}\mathbf{b}$. Obviously UT^{-1} differs from the identity matrix in only its last row. Let $\mathbf{b}' = UT^{-1}\mathbf{b} = (b_i')_{i=0}^{n-1}$. Clearly then

$$T = \begin{pmatrix} d & 1 & & \\ c & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & c & d \end{pmatrix} \quad U = \begin{pmatrix} d & 1 & & \\ c & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ 1 & & c & d \end{pmatrix}$$

$$W = \begin{pmatrix} d & 1 & & c \\ c & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & c & d \end{pmatrix} \quad C = \begin{pmatrix} d & 1 & & c \\ c & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ 1 & & c & d \end{pmatrix}$$

Fig. 1: Four related matrices: C is a circulant matrix sharing its three non-zero parameters with U, W and tridiagonal Toeplitz T. U and W both differ from T and from C by a single but different element.

$b'_{n-1} \neq b_{n-1}$ and $b'_i = b_i$ otherwise. Inspection confirms that $b'_{n-1} = b_{n-1} + x_0$. So we have $\mathbf{x} = U^{-1}\mathbf{b}'$, but b'_{n-1} is unknown to us since x_0 is unknown. Fortunately, we will not need to know b'_{n-1} when calculating the first $n - \lceil k/2 \rceil$ elements of $\mathbf{x}$ via $\mathbf{x} = U^{-1}\mathbf{b}'$. The first $n - \lceil k/2 \rceil$ elements of the last column of U^{-1} will form products with b'_{n-1} in each case producing a term approaching zero which will not contribute to the corresponding entry for $\mathbf{x}$. Clearly then, it is only computation of the trailing $\lceil k/2 \rceil$ elements of $\mathbf{x}$ that depend significantly on the value of b'_{n-1} and for which our computation might not provide a good approximation. ■

Corollary 1: $\mathbf{x}'' = W^{-1}\mathbf{b}$ provides a good approximation for all but the leading $\lceil k/2 \rceil$ elements of $\mathbf{x} = T^{-1}\mathbf{b}$, wherever a k-banded approximation of T^{-1} will suffice.

Proof: Corollary 1 can be proven much the same as Lemma 1. ■

Corollary 2: $\mathbf{x}''' = C^{-1}\mathbf{b}$ provides a good approximation for the central $n - k$ elements of $\mathbf{x} = T^{-1}\mathbf{b}$, wherever a k-banded approximation of T^{-1} will suffice.

Proof: Corollary 2 follows from Lemma 1 and Corollary 1. ■

4. Our Algorithm

We reveal our algorithm in two stages. The first is a special case of the second. It solves $T\mathbf{x} = \mathbf{b}$ by solving both $U\mathbf{x}' = \mathbf{b}$ and $W\mathbf{x}'' = \mathbf{b}$. According to Lemma 1 and Corollary 2, as long as $\lceil k/2 \rceil < n$ the top half of $\mathbf{x}'$ and the bottom half of $\mathbf{x}''$ will be good approximations for the corresponding halves of $\mathbf{x}$. Solutions to $U\mathbf{x}' = \mathbf{b}$ and $W\mathbf{x}'' = \mathbf{b}$ are obtained by first solving $C\mathbf{x}''' = \mathbf{b}$ and then adjusting the leading and trailing $\lceil k/2 \rceil$ elements of $\mathbf{x}'''$ respectively via two partial applications of the Sherman Morrison formula. Of course, we choose to solve $C\mathbf{x}''' = \mathbf{b}$ because we know from (6) that the solution can be determined fast via parallel FFT.

Algorithm 1: FFT Based Diagonally Dominant Tridiagonal Toeplitz Solver

INPUT: Scalars c, d, k, and n such that $|d| > |c| + 1$ and $\lceil k/2 \rceil < n$ defining $n \times n$ DDTT matrix T and a k-band for its inverse, n-vector $\mathbf{b}$.

COMPUTE: $\mathbf{x}''' = (x'''_i)_{i=0}^{n-1} = F_n^{-1}((1 \ ./F_n(\mathbf{v}))\ .*\ F_n(\mathbf{b}))$,
$\overline{\mathbf{v}} = F_n^{-1}(1\ ./F_n(\mathbf{v})) = (\overline{v}_i)_{i=0}^{n-1}$,
where $\mathbf{v} = (d, c, 0, ..., 1)^{\mathsf{T}} = (v_i)_{i=0}^{n-1}$,
$\mathbf{x} = (x_i)_{i=0}^{n-1}$, where
$(x_i)_{i=0}^{\lfloor k/2 \rfloor} = (x'''_i)_{i=0}^{\lfloor k/2 \rfloor} - c x'''_{n-1}/(1 + c\overline{v}_{n-1})[(\overline{v}_i)_{i=0}^{\lfloor k/2 \rfloor}]$,
$(x_i)_{i=\lceil k/2 \rceil}^{n-\lceil k/2 \rceil - 1} = (x'''_i)_{i=\lceil k/2 \rceil}^{n-\lceil k/2 \rceil - 1}$,
$(x_i)_{i=n-\lceil k/2 \rceil}^{n-1} = (x'''_i)_{i=n-\lceil k/2 \rceil}^{n-1} - x'''_0/(1 + \overline{v}_1) [(\overline{v}_{(i+1) \bmod n})_{i=n-\lceil k/2 \rceil}^{n-1}]$.

OUTPUT: $\mathbf{x} \approx T^{-1}\mathbf{b}$.

Algorithm 1 requires $O(\log_2 n)$ parallel steps. Each FFT/IFFT can be handled on separate FFT processors, though a single processor will do. Computing $F_n(\mathbf{v})$ and $F_n(\mathbf{b})$ requires $O(\log_2 n)$ steps. Element-wise reciprocals and products are computed in $O(1)$ steps. $O(\log_2 n)$ steps complete the computation of $\mathbf{x}'''$ and $\overline{\mathbf{v}}$ via IFFT. Two independent applications of the Sherman Morrison Formula account for all other computation. They are performed simultaneously for $k+1$ elements in $O(1)$ steps. Note that due to the sparsity of $\mathbf{v}$, $F_n(\mathbf{v})$ can be computed in $O(1)$ steps if desired.

Algorithm 1 inherits its fine-grained parallelism as well as its scalability from its underlying FFTs. The FFT is theoretically infinitely scalable. Realistically, scaling FFT hardware has its obstacles. We overcome these impediments by employing a block decomposition of T. The non-zero blocks of T centered on the main diagonal are DDTT matrices defined by c and d. They are in essence smaller versions of T. Here and hereafter let $h = k + 1$ and let $2h \times 2h$ DDTT matrix T_{2h} be such a block of T. Let $\hat{\mathbf{b}}_i = T_{2h}\mathbf{x}_i$ where $\mathbf{x}_i = (x_j)_{j=i}^{i+2h-1}$ and $\hat{\mathbf{b}}_i = (\hat{b}_{i,j})_{j=0}^{2h-1}$. Clearly then $\hat{b}_{i,j} = b_{j+i}$ for $j = 1, 2, \ldots, 2h-2$, whereas $\hat{b}_{i,0} = b_i$ and $\hat{b}_{i,2h-1} = b_{2h-1+i}$ are guaranteed only when $i = 0$ and when $i = n - 2h$, respectively. Obviously, $\mathbf{x}_i = T_{2h}^{-1}\hat{\mathbf{b}}_i$. Let $\hat{\mathbf{x}}_i = T_{2h}^{-1}\mathbf{b}_i$, where $\mathbf{b}_i = (b_j)_{j=i}^{i+2h-1}$. Clearly, if it is true of T^{-1}, then a k-banded approximation of T_{2h}^{-1} will also suffice for numerical computation of matrix-vector products. Therefore, since $\hat{\mathbf{b}}_i$ and $\mathbf{b}_i$ differ in at most their first and last positions, $\hat{\mathbf{x}}_i$ and $\mathbf{x}_i$ can differ in at most their first and last $\lceil k \rceil$ positions. We can solve $T\mathbf{x} = \mathbf{b}$ then by solving multiple overlapping instances of $T_{2h}^{-1}\hat{\mathbf{x}}_i = \mathbf{b}_i$. Let C_{2h}, U_{2h}, and W_{2h} be $2h \times 2h$ matrices exhibiting the same structures as C, U, and W respectively. We substitute C_{2h} for T_{2h} and solve via FFT for each partition. Each such block matrix computation produces h elements of $\mathbf{x}$. The first and last partitions are adjusted via the Sherman Morrison formula to reflect substitutions of U_{2h} and W_{2h} to determine the first and last $h/2$ elements of $\mathbf{x}$ respectively.

Algorithm 2: Partitioned FFT Based Diagonally Dominant Tridiagonal Toeplitz Solver

```
INPUT:    Scalars $c$, $d$, $h$, and $n$ such that
          $|d| > |c| + 1$ and $h/2 < n$
          defining $n \times n$ DDTT matrix $T$
          and a $k$-band for its inverse,
          $n$-vector $\mathbf{b} = (b_j)_{j=0}^{n-1}$.

COMPUTE:  $\mathbf{x}_i''' = (x_{i,j}''')_{j=0}^{2h-1} =$
          $F_{2h}^{-1}((1 \ ./F_{2h}(\mathbf{v})) \ .* F_{2h}(\mathbf{b}_i))$,
          for $i = 0, 1, \ldots, n/h - 2$, and
          $\overline{\mathbf{v}} = (\overline{v}_i)_{i=0}^{2h-1} = F_{2h}^{-1}(1 \ ./F_{2h}(\mathbf{v}))$,
          where
          $\mathbf{v} = (d, c, 0, ..., 1)^{\mathsf{T}} = (v_j)_{j=0}^{2h-1}$
          and where
          $\mathbf{b}_i = (b_j)_{j=h(i)}^{h(i+2)-1}$,
          $\mathbf{x}' = (x_{0,j}''')_{j=0}^{h/2-1} -$
            $cx_{0,2h-1}'''/(1 + c\overline{v}_{2h-1})[(\overline{v}_j)_{j=0}^{h/2-1}]$,
          $\mathbf{x}_i = (x_{i,j}''')_{j=h/2}^{3h/2-1}$,
          for $i = 0, 1, \ldots, n/h - 2$, and
          $\mathbf{x}'' = (x_{n/h-2,j}''')_{j=3h/2}^{2h-1} -$
            $x_{n/h-2,0}'''/(1 + \overline{v}_1)$
            $[(\overline{v}_{(j+1) \bmod 2h})_{j=3h/2}^{2h-1}]$.

OUTPUT:   $\mathbf{x} = (\mathbf{x}'^{\mathsf{T}}, \mathbf{x}_0^{\mathsf{T}}, \mathbf{x}_1^{\mathsf{T}}, \ldots, \mathbf{x}_{n/h-2}^{\mathsf{T}}, \mathbf{x}''^{\mathsf{T}})^{\mathsf{T}}$
            $\approx T^{-1}\mathbf{b}$.
```

Algorithm 2 requires $O(\log_2 h)$ parallel steps. $F_{2h}(\mathbf{v})$ and each of the $F_{2h}(\mathbf{b}_i)$ can be computed in $O(\log_2 h)$ steps. Element-wise reciprocals and products are computed in $O(1)$ steps. $O(\log_2 h)$ steps complete computation of the $\mathbf{x}_i'''$ and $\overline{\mathbf{v}}$ via IFFT. The remaining computation consists of two independent applications of the Sherman Morrison Formula to correct h elements in $O(1)$ steps.

5. Experimental Results

For proof of concept and to verify numerical stability, we have tested our algorithm in MATLAB and CUDA implementations. Table 1 presents representatives of the many tests we performed to compare our algorithm to Octave's sparse system solver, and Octave's Gaussian Elimination (G.E.) for numerical stability. Our metric is relative error which we calculate as $||b - T\hat{x}||_2/||b||_2$. Tests were performed for a number of DDTT systems as well as ill-conditioned tridiagonal Toeplitz systems chosen to challenge the numerical stability of our algorithm. System size for all tests is 1024×1024. The relative error calculation does not differentiate between a localized error and an error distributed throughout the solution vector. For ill-conditioned systems both our algorithm and G.E. produce a small cluster of poor approximations at one end of the solution vector, but very good results for the rest. For this reason Table 1 fails to provide the full picture for both our algorithm's and G.E.'s ill-conditioned system solving. Excluding the trailing end of each solution vector, the results generated have similar accuracy for ill-conditioned and diagonally dominant systems as suggested in Section 5. Although not included in the table, tests reveal that when $|d| < |c| + 1$ and the matrix is symmetric or close to symmetric, in which case the matrix is well-conditioned, our algorithm performs poorly.

Our CUDA implementation was executed on a machine sporting a Core i7 Q740 running at 1.73GHz and an NVIDIA GeForce 460M GTX Fermi device. Due to CR's work efficiency advantage over the FFT we tested for $k = 32$ to match with the 32 thread processors of our GPU's symmetric multiprocessor so that both algorithms would enjoy a processor per datum. After distributing such partitions to all 7 TPs throughput is approximately 0.11 GSPS for CR and 0.14 GSPS for our algorithm when transfer from host to device is eliminated from consideration. Clearly, GPUs with their very limited processor counts, their latency riddled and severely bandwidth limited memory hierarchies, and their overhead for fetch execute are not the ideal architecture for our algorithm nor CR. For our algorithm, an architecture designed to process FFTs will certainly prevail.

6. Discussion

Both fine grained algorithms, CR and the FFT, require $\log_2 n$ stages. Each stage of CR generally requires 17 steps. 11 of these 17 steps are floating point multiplications or divisions and the other 5 are floating point subtractions. Each stage of a straight forward FFT requires a complex multiplication followed by a complex addition or subtraction.

c / d	Condition#	Sparse	G.E.	Alg. 1	Alg. 2
1 / 1.0E+08	1.0E+00	1.1E-16	1.1E-16	2.1E-16	2.0E-16
1 / 64	1.1E+00	8.4E-17	1.5E-16	2.1E-16	2.1E-16
1 / 4	3.0E+00	9.2E-17	1.1E-16	2.1E-16	2.1E-16
4 / 0	5.8E+23	6.9E-01	1.1E-02	1.1E-02	1.1E-02
4 / 1	3.9E+16	1.4E290	3.1E-02	2.4E-02	2.5E-02
1.0E+08 / 1	1.6E+17	NAN	4.0E-02	2.8E-02	2.8E-02
1.0E+08 / 0	5.5E+29	NAN	4.0E-02	3.1E-02	3.2E-02

Table 1: Relative Errors $||b - T\hat{x}||_2/||b||_2$ by Method. For all tests $n = 1024$ and $h = 64$.

This can be performed in 3 steps given the necessary number of floating point units. Complex multiplication requires 2 steps. Given complex numbers $a+bi$ and $c+di$, where a, b, c, and d are floating point values their product $(a+bi)(c+di) = (ac - bd) + (ad + bc)i$. Floating point products ac, bd, ad, and bc can all be computed in parallel in a single step by four floating point units. Thereafter $ac - bd$ and $ad + bc$ require a single step on two of those four floating point units. Our algorithm requires a sequence of two FFTs and two element-wise multiplications. The first of these two element-wise multiplications involves complex operands. The second has real operands. These multiplications require no more than 3 steps. Our algorithm then requires 6 steps (2 floating point multiplications and 4 floating point additions) for each of $\log_2 n$ stages, plus 3 additional steps (2 floating point multiplications and 1 floating point addition). So while CR is more work efficient than the FFT, when the processor count matches the degree of parallelism, the FFT's lower workload per processor tilts expected latency in it's favor. On many architectures multiplication/division has a significantly higher latency than addition/subtraction. Fermi devices however pipeline their operations in a way that results in an invariant latency across the instruction set. This works in favor of CR versus FFT on the GPU since CR has many more multiplications to perform. Additionally, FFTs can be reformulated to replace each multiplication with multiple additions. Where addition has a lower latency than multiplication this can provide a boost to throughput as well as lower the latency of FFT based computations.

While we lack the expertise to bring together the hardware components best suited for our algorithm, it is clear that the main component is readily available both as ASICs and FPGAs. For instance Dillon Engineering [12] produces a FPGA capable of producing 25.6 GSPS when configured as a pipelined 64 point FFT processor. The supporting circuitry around the FFT processor need only perform a handful of predetermined floating point operations in an embarrassingly parallel fashion. Our FFT based algorithm would allow an array of small FFT processors and their supporting circuitry embedded on FPGAs to tackle large linear systems by processing system blocks in complete isolation.

Clearly our algorithm could be implemented on a 64 point device to solve any tridiagonal Toeplitz system where $|c/d| \leq 1/4$. For a real system where $|c/d| \leq 1/2$ a few extra steps can be added due to symmetry in the Fourier image and the same 64 point device can be applied. At one end of the spectrum, a single such FPGA could handle all partitions on its own at a pipelined 25.6 GSPS for real systems. This is orders of magnitude faster than the GPU implementations. At the other end of the spectrum two FPGAs could be assigned to each partition. In addition to the embarrassingly parallel O(1) step computations, one FPGA would apply the FFT to the input as it arrives for processing and the other the IFFT that produces the final output. The FPGAs in separate partitions work independently in an embarrassingly parallel manner. Samples per second theoretically approach 25.6 GSPS times the number of partitions. Providing each processor with its input at a rate to keep the pipeline saturated is perhaps the biggest challenge. But even the memory can be partitioned into banks, where the number of banks equals the number of FFT processors. Each FFT processor would need to read data from only two banks. It's own and that of it's immediate neighbor. Fortunately, if a hardware package including the FFT processor(s) can handle a single partition, scaling up could be almost as simple as dropping in additional hardware.

7. Concluding Remarks

One of the arguments in favor of the great effort to harness GPU processing power has been their ready availability and amortization of their R&D costs due to their applicability to the video game market. The FFT and therefore FFT processors find use in myriad different applications. This also leads to ready availability and cost control, so that FFT processors are a viable building block on which to base design of parallel algorithms. In all, our algorithm when implemented on FFT processors with minimal supporting circuitry has many advantages over other algorithms. Not only does it exhibit the finest of granularity, but is designed for implementation on hardware that can actually take full advantage of that granularity thereby maximizing parallelism even for the largest of linear systems. For a fixed k its op count is $O(n)$ and it requires $O(1)$ parallel steps. That's $O(n \log_2 k)$ and $O(\log_2 k)$ respectively when k is not held constant. Combine the low step count of our algorithm with hardware designed to carry out these specific computations and it becomes clear that the algorithm will compete successfully.

A MATLAB implementation of our algorithm can be found at:

$http:\backslash\backslash comet.lehman.cuny.edu/bmurphy/tridiagonal$

References

[1] Daniel J. Bernstein. Removing redundancy in high-precision newton iteration. Technical report, 2004.

[2] R. R. Bitmead and B. D. O. Anderson. Asymptotically Fast Solution of Toeplitz and Related Systems of Linear Equations. *Linear Algebra and Its Applications*, 34:103Ű–116, 1980.

[3] Ronald Bracewell. *The fast Fourier transform and its applications*. McGraw-Hill Science, New York, NY, 1999.

[4] E. O. Brigham. *The fast Fourier transform and its applications*. Prentice Hall, Upper Saddle River, NJ, 1988.

[5] Raymond H. Chan. Toeplitz preconditioners for Toeplitz systems with nonnegative generating functions. *IMA Journal of Numerical Analysis*, 11(3):333–345, 1991.

[6] L.W. Chang, J.A. Stratton, H.S. Kim, and W.M.W. Hwu. A scalable, numerically stable, high-performance tridiagonal solver using gpus. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 27. IEEE Computer Society Press, 2012.

[7] K.-L. Chung and L.-J. Shen. Vectorized algorithm for b-spline curve fitting on Cray X-MP EA/16se. In *Proceedings of the 1992 ACM/IEEE conference on Supercomputing*, Supercomputing'92, pages 166–169, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[8] D. Commenges and M. Monsion. Fast inversion of triangular toeplitz matrices. *IEEE Transactions on Automatic Control*, 29:250–251, 1984.

[9] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex fourier series. *Math, Comp.*, 19:297Ű–301, 1965.

[10] Andrew Davidson and John D. Owens. Register packing for cyclic reduction: A case study. In *Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units*, pages 4:1–4:6, March 2011.

[11] S. Demko, W. F. Moss, and P. W. Smith. Decay rates for inverses of band matrice. *Mathematics of Computation*, 43(168):491–499, 1986.

[12] Dillon Engineering. http://www.dilloneng.com/fft_ip/parallel-fft, 2013.

[13] Sun Feng-Wen, Jiang Yimin, and Baras John S. On the convergence of the inverses of toeplitz matrices and its applications. *IEEE Transactions on Info. Theory*, 49(1):180–190, 2003.

[14] J. Guitart and S. Ruiz-Moreno. Strict calculation of the light statistics at the output of a traveling wave optical amplifier. *Electronic Letters*, 29:1589–1590, 1993.

[15] G. Hanrot and P. Zimmermann. Newton iteration revisited, 2002.

[16] David Harvey. Faster algorithms for the square root and reciprocal of power series. *Computing Research Repository*, 2009.

[17] R. W. Hockney. A fast direct solution of PoissonŠs equation using Fourier analysis. *Journal of the ACM*, 12(1):95–113, Jan. 1965.

[18] Li. Kuiyuan. A fully parallel method for tri-diagonal eigenvalue problem. *Internat. J. Math. Math. Sci.*, 17:741–Ű752, 1994.

[19] Joseplluis Larriba-Pey, Juan J. Navarro, and Angel Jorba. Vectorized algorithms for natural cubic spline and b-spline curve fitting. In *Proceedings of PDP'96*, pages 385–392, 1996.

[20] Fu-Rong Lin and Wai-Ki Ching. Inverse Toeplitz preconditioners for Hermitian Toeplitz systems. *Numerical Linear Algebra with Applications*, 12(2–3):221–229, March/April 2005.

[21] M. Mascagni. A parallelizing algorithm for computing solutions to arbitrarily branched cable neurons. *Journal of Neuroscience Methods*, (36):105–114, 1991.

[22] Jeffrey M. McNally. Fast parallel algorithms for tri-diagonal symmetric toeplitz systems. Master's thesis, University of New Brunswick (Canada), St. John, 1999.

[23] Jeffrey M. McNally. *A scalable communicationless parallel algorithm for tri-diagonal Toeplitz systems*. PhD thesis, University of New Brunswick (Canada), St. John, 2003.

[24] Jeffrey M. McNally, L.E. Garey, and R.E. Shaw. A split-correct parallel algorithm for solving tri-diagonal symmetric Toeplitz systems. *Inter. J. of Comp. and Math.*, 75:303–313, 2000.

[25] Jeffrey M. McNally, L.E. Garey, and R.E. Shaw. A communicationless parallel algorithm for tridiagonal Toeplitz systems. *J. of Comp. and Applied Math.*, (212):260–271, 2008.

[26] M Morf. Doubling Algorithms for Toeplitz and Related Equations. pages 954Ű–959, 1980.

[27] Brian J. Murphy. Acceleration of the inversion of triangular toeplitz matrices and polynomial division. *Computer Algebra in Scientific Computing: Lecture Notes in Computer Science*, 6885:321–332, 2011.

[28] Brian J. Murphy. Butterflies Solve Bidiagonal Toeplitz Systems. *http://comet.lehman.cuny.edu/bmurphy/bidiagonal*, 2013.

[29] Brian J. Murphy. Solving Tridiagonal Systems on a GPU. In *Proceedings of the 20th International Conference on High Performance Computing (HiPC, 2013)*, pages 159–168, 2013.

[30] Reinhard Nabben. Decay rates of the inverse of nonsymmetric tridiagonal and band matrices. *SIAM J. Matrix Anal. Appl*, 20:820–837, 1999.

[31] S. S. Nemani. *Perturbation methods for circulant-banded systems and their parallel implementation*. PhD thesis, University of New Brunswick (Canada), St. John, 2001.

[32] Victor Y. Pan. *Structured Matrices and Polynomials: Unified Superfast Algorithms*. Birkhäuser, Boston, MA, USA, 2001.

[33] Eric Polizzi and Ahmed H. Sameh. A parallel hybrid banded system solver: the spike algorithm. *Parallel Comput.*, 32(2):177–194, February 2006.

[34] O. Rojo. A new method for solving symmetric circulant tri-diagonal system of linear equations. *Comput. Math. Appl.*, (20):61–67, 1990.

[35] Arnold Schönhage. Variations on computing reciprocals of power series. *Information Processing Letters*, 74:41–46, 2000.

[36] Arnold Schönhage and Ekkehart Vetter. A new approach to resultant computations and other algorithms with exact division. In *European Symposium on Algorithms*, pages 448–459, 1994.

[37] M. Sieveking. An algorithm for division of power series. *Computing*, 10(1–2):153–156, March 1972.

[38] Harold S. Stone. An efficient parallel algorithm for the solution of a tridiagonal linear system of equations. *J. ACM*, 20(1):27–38, January 1973.

[39] Volker Strassen. Gaussian Elimination is not Optimal. *Numer. Math*, 13:354Ű–356, 1969.

[40] X.-H. Sun. Application and accuracy of the parallel diagonal dominant algorithm. *Parallel Comput.*, pages 1241Ű–1267, 1995.

[41] Xian-He Sun and Stuti Moitra. A fast parallel tridiagonal algorithm for a class of CFD applications. Technical Report 3585, NASA, 1996.

[42] Joris van der Hoeven. Newton's method and fft trading. *Journal of Symbolic Computation*, 45:857–878, 2010.

[43] W. M. Yan and K. L. Chung. A fast algorithm for solving special tri-diagonal systems. *Computing*, 52:203–211, 1994.

[44] Yao Zhang, Jonathan Cohen, and John D. Owens. Fast tridiagonal solvers on the GPU. In *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP 2010)*, pages 127–136, January 2010.

Resiliency in IO Tools for Scientific Computing

William W. Dai
Los Alamos National Laboratory
Mail Stop D413, Los Alamos, NM 87545

Keywords: resiliency, data structure, data management, IO.

Abstract
IO tools in numerical simulations often serve two functions, one for writing and reading files to restart calculations and the other for writing and processing diagnostic files including files for graphics post-processing. For diagnostic files, tools directly working for high-level data structures are desired, such meshes with adaptive mesh refinement, unstructured meshes, and association between meshes and variables. Typically, one file in numerical simulations involves several or many writings. One of questions about the resiliency of IO tools is the following. If a simulation was crashed during writing a file, could an IO tool read the high-level data structures, meshes and their associated variables in the file that was incompletely written? This paper describes how a parallel IO library, HIO, is developed and modified toward resiliency. The goal of the library is to provide sustainable, interoperable, efficient, scalable, and convenient tools for parallel IO and data management for high-level data structures in numerical simulations. The high-level data structures include multi-dimensional arrays, structured meshes, unstructured meshes, the meshes generated through adaptive mesh refinement, variables associated with these meshes, and data defined on particles in particle simulations. The library is based on MPI-IO. Compared with MPI-IO, the overhead to write the explicit users' data structures are very small. To further improve IO performance, in addition to meta data, the library could buffer problem-size data while keeping users' explicit high-level data structures. The buffering mechanism improves IO performance by a factor 10 to 20 in simulations of multi-physics. For resiliency, we have designed and implemented additional procedure so that the library could read all the high-level data within a file even if the file was incompletely written due to an interruption of a simulation. The cost for the resiliency of the library is very small in either performance or space.

1. Introduction

Parallel IO and scientific data management have played an important role since the beginning of large scale scientific computing, and are getting more important due to the increase of the scale of the computing. Existing products, which have partially addressed the issue, include HDF5 [1], SAF [2], CGNS [3], NetCDF [4,7], Silo [5], UDM [6], and others. Each of the existing products has certain advantages and disadvantages. Some of the products have good functionalities for unstructured meshes, but they either don't have efficient capabilities for running on parallel computer environments or lack for good parallel I/O performance. Some of them are designed for parallel environments, but do not have the capabilities to deal with unstructured meshes, or they get only a fraction of MPI I/O performance. Some have good IO performance but lack the functionality to query data sets for their relationship. Some have a decent IO performance for large data sets, while they failed to deliver the similar performance for small data sets.

One of issues many researchers are looking into is resilience of IO tools, particularly IO tools with high-level data structures. This paper will describe how an IO library, HIO, is structured toward the resilience for high-level data structures used in simulations of multi-physics. If a simulation is interrupted during writing a file, whether or not this file, which contains high-level data structures, could be read through the library.

The HIO library is for parallel IO and data management for high-level data structures used in numerical simulations. It has been developed under Department of Energy (DoE) Advanced Simulation and Computing (ASC) program for ASC code projects. The HIO library is the further development of the UDM library [6]. The development include the direct dependent on MPI-IO without middle layer between HIO and MPI-IO, buffering mechanism, the N-to-M mode, i.e, writing simulation data on N computer processors to M files, and a resilient feature. The library consists of functionalities for IO and data management in numerical multi-physics simulations, such hydrodynamics, magnetohydrodynamics, radiation diffusion and transport, material mixing, and mechanics. The high level data structures include particles in particle simulations, structured meshes, unstructured meshes, meshes generated from adaptive mesh refinement (AMR), and their associated variables defined on cell centers, vertices, edges, or faces of the meshes. The library is built on the top of MPI I/O, and its I/O performance is very close to that of MPI I/O. Therefore, the cost to bookkeeping users' explicit high-level data structures is negligible. To further improve IO performance, we buffer data together in the library before calling MPI-IO. We have implemented additional procedure for resilience so that a file interrupted during writing could be read for the high-level data structured contained in the file. The resilient feature is controlled by an input parameter. The cost for the resilient feature is small.

To our knowledge, the functionality and performance of the library are superior to existing products for these data structures in simulations of multi-physics.

2. Functionality of the Library

The HIO library provides IO and data management tools for data in particle simulations, single and multi-dimensional arrays, structured meshes, unstructured meshes, and the mesh generated through block-, patch-, and cell-based AMR, and variables defined on these meshes in numerical simulations on parallel computer environments. It also provides a hierarchical data structure within a data file. The files generated through the library are self-described.

One of the important and powerful functionality in the HIO library is the management of unstructured meshes and their associated variables. The library supports a broad range of unstructured meshes, which include meshes with fixed shapes, arbitrary polygons, and arbitrary polyhedrons. The mesh elements with a fixed shape may be triangles, quadrangles, pentagons, tetrahedrons, pyramids, wedges, pentagon prisms, and points. A mesh element may be a zone, or face, or edge, i.e., a mesh may be a zone-mesh, face-mesh, edge-mesh, and points. An edge-mesh may be one-, or two-, or three-dimensional, and a face-mesh may be either two- or three-dimensional. Mesh elements of a zone-mesh may be made directly from nodes, or the elements may be made from edges, or the elements may be made from faces and the faces are then made from either edges or nodes. The HIO library also supports ghost mesh elements in unstructured meshes, boundary faces, boundary edges, boundary nodes, slip faces, slip edges, slip nodes, etc. The variables associated with unstructured meshes may be node-variables, or edge-variables, or face-variables, or zone-variables, and variables may be scalars, or vectors, or tensors.

Examples for mesh elements include triangles, quadrangles, pentagons, arbitrary polygons, hexahedrons, tetrahedrons, wedges, pyramids, pentagon-prisms, and arbitrary polyhedrons. The mesh elements may be made from any lower level entities, such as faces, edges, and nodes.

Although the HIO library covers a broad range of unstructured meshes, a user only has to set up his/her own mesh definition, and all other mesh definitions are hidden from the user. For example, for an unstructured zone-mesh made from nodes, only the list of nodes for each element is needed if the elements are of a fixed shape, such as prisms. If mesh elements are arbitrary polyhedrons made from nodes, two arrays are needed, one for the numbers of nodes for elements, and the other for the list of nodes for each element. Like the capability for structured meshes, the association between a mesh and a set of variables is automatically built into the library and a file.

The following is an example to write an unstructured mesh with general polyhedrons. To specify the mesh, each computer processor has a number of elements, nzone. The set of elements have a number of faces and nodes, nface and nnode. The arrays, num_faces_for_zone and facelist_for_zone, are to specify the elements, and the arrays, num_nodes_for_face and nodelist_for_face, are to define the faces. The arrays, x, y, and z, are the locations of these nnode nodes. All these arrays and sizes are local to the processor. Then code to write this unstructured mesh is as follows.

```
meshio_unstructured_mesh m;
meshio_coord *c = &m->coord;
meshio_init(meshio_umesh, -1, &m);
m.dims = 3;
m.type = meshio_general_mesh;
m.sizes[0] = nzone; m.sizes[1] = nface; m.sizes[3] = nnode;
m.num_nodes_for_face = num_nodes_for_face;
m.nodelist_for_face     = nodelist_for_face;
m.num_faces_for_zone = num_faces_for_zone;
m.facelist_for_zone     = facelist_for_zone;
c->coord[0] = x;  c->coord[1] = y; c->coord[2] = z;
c->datatype = meshio_double;
m.datatype = meshio_int;
meshio_write(meshio_umesh, fileid, &m);
```

Although a mesh generated through AMR may be considered as an unstructured mesh in IO, but it involve unnessary memory copies and additional working memory requrement. For example, a typical cell-based structured mesh is represented by cell center and cell width of each cell in each dimension. To convert ths AMR mesh to a typical unstructured mesh require some work, and will often involve unnecessary (often troubling) redundant nodes. With the HIO library, the AMR mesh could be naturally handles without any transformation.

As stated before, for cell-based structured AMR meshes, we store the center and width of each element in each dimension. Scalar variables associated with the meshes are one-dimensional and associated vector variables are two-dimensional arrays. The association between variables and a mesh is automatically built and stored in the file. For block- or patch-based AMR structured meshes, each block or patch is considered as a standard structured mesh.

The library also support ghost cells in AMR meshes. Some post data anaylizers need variables on ghost cells to derive information around interfaces between processor interfaces, such as a visualization tool calculating isosurfaces.

Users can add any description to any object, such as a file itself, or an array, or a mesh, or a variable, as long as the description is not of the problem-size, and each processor has the same description. More importantly, writing all descriptions almost doesn't have any IO cost, since all the descriptions will be buffered together with all the meta data and are written at the end of a file when the file is closed.

The number of the descriptions and each description can be automatically queried. As writing the description, reading any description does not involve any additional IO cost since all the descriptions together with all the meta data are read into the memory when a file is open.

Writing small data sets into a file on a parallel environment will typically result in very low IO performance. The HIO library provides an automatic buffering mechanism so that a large number of small data sets will automatically buffered together before they, together with their names and descriptions, are written into a file. Writing small data sets, users will get the same IO performance as they get for large data sets, and users don't have to keep track of the locations of each individual small data set in the combined buffer and a disk file.

To read a small data set, the HIO library actually only copies small data set from a buffer to the user's memory. If the buffer is not available yet, the library will automatically read the buffer first, and then copy the data. Therefore, the library doesn't involve reading from a disk with a small set of data.

To users, all the tedious operations necessary for writing and reading the small data sets are behind the scene. Writing small data sets is the same as writing big data sets, and even the names and arguments of the functions to be called are the same.

As stated before, the HIO library buffered all meta together, i.e., data used to describe real data, and write the buffer into a file when a file is closed. The library also copies small data sets into a buffer and then automatically writes the buffer into a file when the full becomes full. The buffer includes the description of each small data set.

There are at lease two cases in which we have to broad the buffering capability. The first situation is encountered for data in block- and patch-based AMR. Sometimes, users want write each individual block and patch into a files. Since there are a large number of blocks and patches, writing a complete mesh will involve a large number of IO operation if users don't want to map blocks and patches to a full mesh. Each of IO operation will be expensive.

The second case is to further improve IO performance for users' high-level data structures. For example, there are about hundred IO operations in a typical simulation of Roxane project, which come from the cell-based AMR mesh and associated variables, meshes occupied by each material and variables defined on the meshes, the unstructured mesh used for radiation diffusion and variables defined on the mesh, etc. Figure 1 shows unstructured cells used in three-dimensional radiation diffusion solvers. Even if there is only a single mesh in each file, and associated with this mesh could be many variables. Normal, a mesh will constitute a few IO operations, and each variable will do one IO operation. Of course, we could buffer them all together and execute only one IO operation, but it will typically loss the high level structure of mesh and the association between mesh and variables. The HIO library provides a mechanism to reduce the hundred IO operations into a few without losing users' high level structures.

To initiate the mechanism, users only have to call the following function with file id and the size of buffer specified, *meshio_init_buffer(file_id, buffer_size),* The parameter file_id is the id of the file created. All the data in the subsequent calls of writing will be buffered until the call of the following function or the file is closed, *meshio_finalize_buffer(file_id).* All the calls between the call meshio_init_buffer and the call meshio_finalize_buffer (or closing the file) are the same no matter whether or not the buffering mechanism is used. Also users use exactly same way to read buffered data as that for un-buffered data.

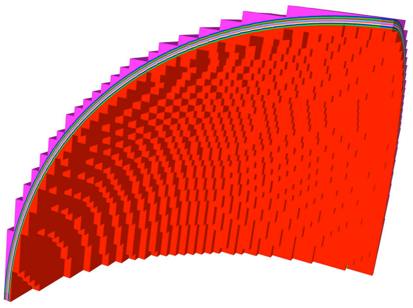


Figure 1. Unstructured cells in three dimensions formed through interface reconstrution in a AMR mesh. The unstructure cells are used in the radiation diffusion solver in Roxane.

After data, for example, all the meshes and the variables defined on the meshes, are written into a file through buffering, internally there are two approaches to read these data in the library mainly for two different purposes to read the file. The first is to restart a simulation. The most efficient way to read the data is to read each whole buffer and then copy the data from each buffer to users' memory. For example, after open a file, a user reads a particular mesh. Internally the HIO library first find the buffer containing this mesh, and read the entire buffer into memory from the file. The buffer is kept in memory until another buffer is requested. After reading the mesh, the user reads a particular variable. Internally the library copy the variable from the buffer to the user's memory without touching the file. This mechanism to read buffered data is extremely efficient for restarting.

Another case is to visualize a particular variable in a large number of files that correspond to different instants in a simulation, for example, in the format of movie through a visualization tool. In this case, we could not afford to read a whole buffer in each file just for ONE variable. The HIO library actually reads only one variable in each of files for this case. Users could choose either of the two approaches to read buffered data through the function, *meshio_buffer_mode(flag_for_buffered_read)* with the input 1 or 0 for flag_for_buffered_read. The typical use to set to buffered reading for restarting, and to non-buffered reading for visualization.

A file written through the HIO library is self-described. All the information in the file may be queried through a function in the library. For example, for a given file, users may find the number of arrays, meshes, and variables, the description of each array, mesh, and variable, and any association between meshes and variables. Through the querying function in the library, meshes and variables may be directly viewed through parallel graphics tools.

After a data object, such as array, mesh, and variable, is written into a file, users may read any part of the data object in terms of, for example, global ids if global ids are defined, or a processor rank, or a space domain.

As stated beore, one of usages of the library is to connect data files to visualization tools. All the images presented in this paper are produced through the visualization tool Ensight. Simulations generate a set of files for visualtion through the HIO library, and the set of files could be read by either Ensight and VISIT without direct transformation.

We woule like to mention that the connection bwteen HIO and post-processor tools (such as visualiation) is different from the connections in many other tools. Since HIO has the concept of mesh and associated variables, no matter where users write their mesh and varaibles in a file, and no matter how to use the library, the HIO reader, which connects tp post processor tools, will always be able to find all the meshes and their variables. Many IO tools require users to write, for example, coordinates and connectivity arrays of a mesh to specific places for their readers to identify, while variables of a mesh have to write to another special place.

3. STRUCTURE OF THE LIBRARY

The HIO library is built on the top of MPI-IO. Actually the library consists of two parts. The first part is called bio, the second part is called meshio, and either of them is also a parallel IO library. The library bio is directly built on the top of MPI-IO, and meshio is doesn't directly related to MPI-IO.

The bio libaray support the hierarchical data structure and one dimensional arrarys.To use the buffering capability in bio, any arrays written through bio_write between the following two functions will be written through buffering.

- bio_buffer_init(fileid, name, size, buffer_id)
- bio_buffer_finalize(buffer_id)

The bio library itself doesn't have concepts of high level data structures used in simulations, such as meshes and variables, but it could be directly used, for example, for retarting.

The functions in bio-layer is tyoically used for restart files, which the functions in meshio-layer typically for files used in viusalization and connection. Although files generated from both layers could be visualized through HIO readers and visualization tool (such as Ensight), the structures in the two kinds of file are different. To visualize the files generated from bio-functions only, users have to follow certain assumtions to write files, or users have to write a specific reader for their own bio files. In this sense, many IO tools and libraries in our community are like this. But, meshio-functions are different, and through meshio functions users don't have to follow any assumtions. A reader based on meshio-functions could read any meshes and variables written through meshio-functions.

4. BASIC FUNCTIONS AND UAGE

One of the design principles of the HIO library was a small number of functions. The following is the list of main functions of the library.

- meshio_set_mcomm(mfiles)
- meshio_open(filename, mode, fid)
- meshio_close(fid)
- meshio_init(type, fileid, obj)
- meshio_write(type, fileid, obj)
- meshio_query(type, fileid, filter, nobjs, objs)
- meshio_clean(type, nobjs, objs)
- meshio_get_size(type, domain, fileid, obj)
- meshio_read(type, domain, fileid, objs, nobjs)
- meshio_init_buffer(fileid, name, buffersize)

The first function is to set the number (M) of files to be written in the subsequent files. By default, M is 1 without the call of the function. The second and third functions are for opening and closing files. The function meshio_init is to initialize any object before it is being used, and the object includes array, structured and unstructured mesh, AMR mesh, and variable defined on any mesh. The function meshio_query is for querying, which include querying files, querying variables, querying relationship between variables and meshes, querying attributes, etc. The function meshio_clean is to release the memory allocated in the call of the function meshio_query. The function meshio_get_size to determine the sizes of grid zones, faces, edges, and nodes for a given part of a mesh, for example, a spatial domain, a part associated with a specific processor, or a number of elements. The function meshio_read is to read attributes and any data for a given part of a mesh, which include coordinate, mesh, variable, etc.

The function, meshio_init_buffer, is to create a buffer of size buffersize. If this function is called, all the subsequent data to be written through meshio_write will be buffered into the buffer, and are then written into the file until the buffer is full or the file is closed.

A cell_based AMR structured mesh is defined through arrays for the centers of elements, x, y, z, and arrays for widths of the elements, dx, dy, and dz. The following segment of codes shows the usage to write a cell_based AMR mesh.

```
meshio_Structured_Element_AMR m;
meshio_init(meshio_smesh_element_amr, -1, &m);
m.name = meshname;
m.dims  = 2;
m.datatype_coord = meshio_double;
m.size         = nelement;
m.coord[1]   = x; m.coord[0]   = y;
m.dcoord[1] = dx; m.dcoord[0] = dy;
meshio_write(meshio_smesh_cell_amr, fileid, &m);
```

After a mesh is written, a set of variables can be written into the file, and the relationship between the mesh and variables is automatically built. The following codes show the usage to write a scalar variable defined on elements, zone_density, and a vector defined on nodes, node_velocity_x and node_velcity_y.

```
meshio_Mesh_Var   var;
meshio_init(meshio_mesh_var, -1, &var);
var.name          = varname1;
var.mesh_ids[0] = m.id;
var.type            = meshio_zone;
var.datatype      = meshio_double;
var.rank            = 0;
var.comps[0].buffer = zone_density;
meshio_write(meshio_mesh_var, fileid, &var);

meshio_init(meshio_mesh_var, -1, &var);
var.name          = varname2;
var.mesh_ids[0] = m.id;
var.type            = meshio_node;
var.datatype      = meshio_double;
var.rank              = 1;
var.comp_sizes[0]   = 2;
var.comps[0].buffer = node_velocity_x;
var.comps[1].buffer = node_velocity_y;
meshio_write(meshio_mesh_var, fileid, &var);
```

After a mesh and a set of variables are written into a set of files. Any part of the mesh and the variables associated with this part of the mesh may be easily read. The following segment of codes shows the usage to read a part of mesh defined through domain and variables associated with this domain.

```
int                              nvar;
meshio_Domain                    domain;
meshio_Unstructured_Mesh  m;
meshio_Mesh_Var                  *vars;
```
specify mesh and domain
```
meshio_get_size(type, domain, fileid, &m);
```
allocate space for the part of mesh, and variables
```
meshio_read(meshio_umesh, domain, fileid, &m, 1);
meshio_read(meshio_mesh_var, domain, fileid, &vars,
nvar);
```

5. Resiliency and Performance

As the first try to address the resilience of the library, we would like to be able to read the existing data of a file if IO operations are interrupted during writing. Particularly, the high-level data structures supported by the library could be read through the library. As stated before, the file structure and high-level data structures are kept through meta data in the library, and the meta data are written into a file as the header and tail. The header is of fixed size solely for the necessary information about hardware, and the size of the tail depends of the usage of the library. The tail is written at the end of the file when the file is closed.

This structure of header and tail is specially designed for resilient IO operations. If IO operations were interrupted before the file is closed, the tail would not be in the file, and therefore the structure of the file and high-level data structures, which were already written into the file, would not be recognized. To overcome this deficiency for resilient operations, we provide an additional option for the redundancy for the header and tail. In the library, immediately after every operation of writing, we write the header and tail into a separate file, called the backup file, through one processor. The size of the backup file is exactly the size of the header and tail.

For normal IO operations, the backup file is an overhead. If a file of our library is successfully closed, the backup file will not be used. But, if IO operations are interrupted before the file is closed, the library will read the backup file to determine the structure of the data file first, and then read the data in the data file.

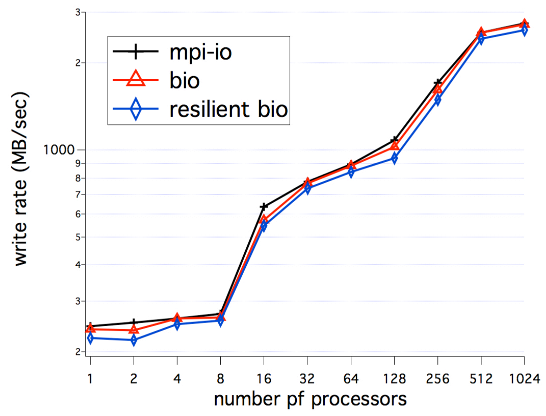


Figure 2. Write rates of MPI-IO, bio, and resilient bio for large data sets.

The library has an option to generate one backup file for each file. But, since the library is developed for numerical simulations in which many files will be generated at different instants during one simulation, we have an option to generate only one backup file for one simulation. The backup file of a new data file at a new instant will overwrite the backup file of the data file at a previous instant.

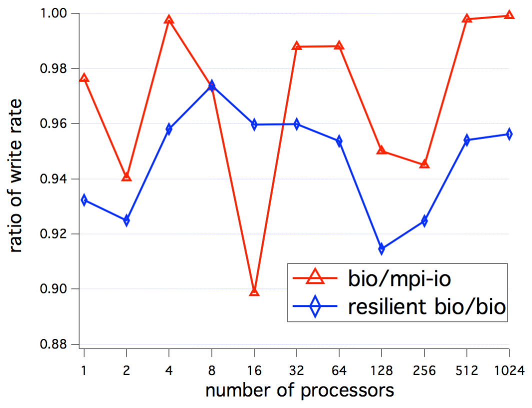


Figure 3. Write rates of bio and resilient bio for large data sets measured by MPI-IO and bio library respectively.

To show the overhead of the resilience feature of the library, we implemented a series tests for both bio and meshio libraries on the cluster, Moonlight, at the Los Alamos National Laboratory. Moonlight has 302 nodes, and each of which is comprised of 2 x Eight-Core Intel Xeon model E5-2670@2.6GHz. The two Xeon sockets (ie. all 16 cores) share 32 GBytes of RAM on each node. The scratch space is accessed through Panasas parallel file system.

The first set of tests is for IO of large data sets through bio library. Each processor has 100 arrays, each of which has 1,000,000 values of double precision (8 bytes). Therefore, the size of file generated through MPI-IO is 800 MByte for one processor and 819.2 GByte for 1024 processors. The files generated through bio library are slightly larger due to meta data included in the files, and they are 9196 Byte more for one processor and 9206 Byte for 1024 processors. These numbers of 9196 or 9206 Byte are also the size of backup file for resilience. Therefore, the cost of bio library or the resilience feature in disk space is 0.001% for one processor and 0.000001% for 1024 processors. Figure 2 shows the performance of writing through MPI-IO, bio without resilience, and bio with resilience. In Fig.3, we give the write rate measured by MPI-IO and bio respectively. We should point out that we have not made any optimization for underneath MPI-IO and file systems.

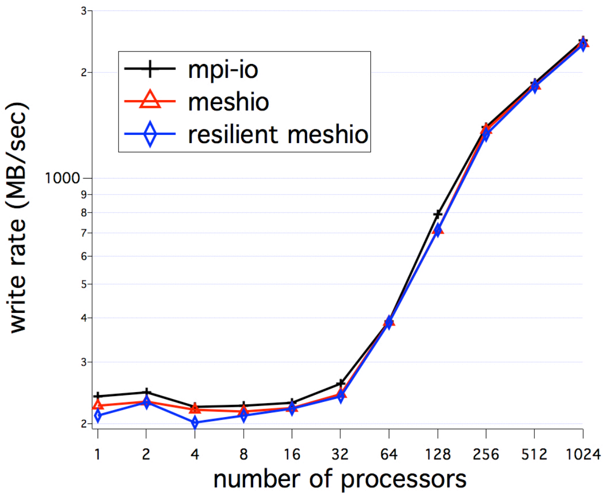


Figure 4. Write rates of MPI-IO, meshio, and resilient meshio for large data sets.

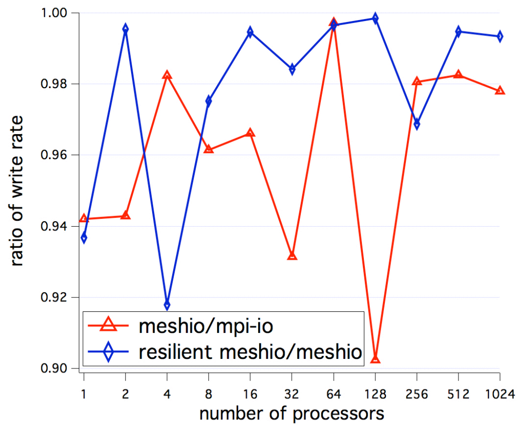


Figure 5. Write rates of meshio and resilient meshio for large data sets measured by MIO-IO and meshio.

The second set of tests is for a large unstructured mesh and its associated variables, in which each processor writes 786432 three-dimensional elements and 274,625 nodes, a connectivity array, and 100 variables of double precision (6 bytes) defined on the elements. The sizes of files generated from MPI-IO, which do not contain any meta data, are 651,465,240 bytes for one processor, and 667,100,405,760 bytes for 1024 processors. The sizes of files generated from meshio library, which include all the meta data describing the unstructured mesh and parallel environment, are 651,483,994 for one processor and 667,100,629,115 for 1024 processors. These additional 17,854 and 223,355 bytes are the overhead of meshio in disk, and are also the sizes of the backup files for resilience. The percentages of the overhead are 0.003% for one processor and 0.000003% for 1024 processors. Figure 4 displays the writing performance of MPI-IO, the meshio library, and the meshio library with resilience. Figure 5 gives the writing rates measured by MPI-IO and meshio respectively.

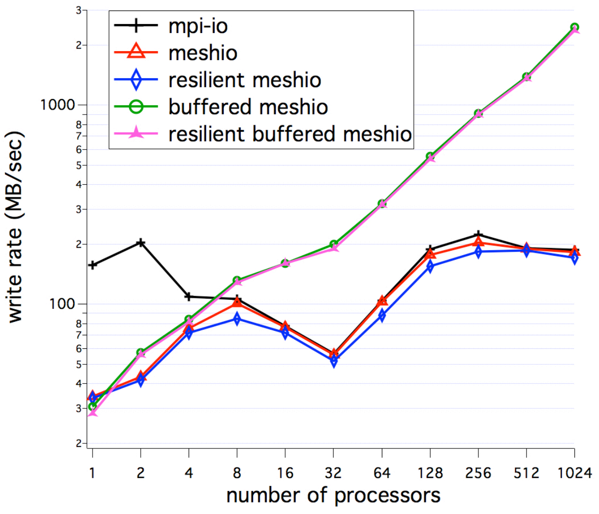


Figure 6. Write rates of MPI-IO, meshio, resilient meshio, buffered meshio, and resilient buffered meshio for small data sets.

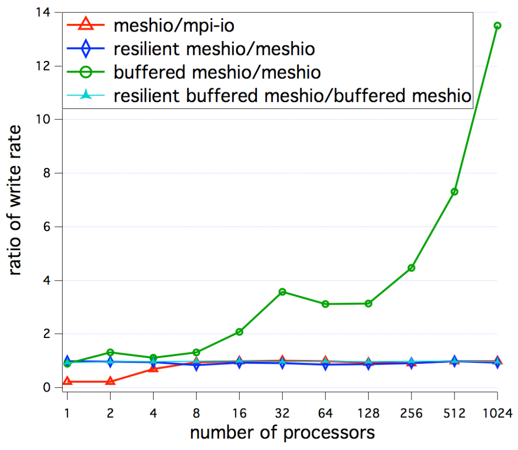


Figure 7. Write rates of meshio, resilient meshio, buffered meshio, and resilient buffered meshio for small data sets measured by MPI-IO, meshio, meshio, and buffered meshio respectively.

The final set of tests is for writing a smaller unstructured mesh together with its 1,000 associated variables. Each processor has 41,472 three-dimensional elements and 15,625 nodes, a connectivity array describing the relation between each element and nodes. The variables are double precision (8 bytes) defined on elements. The sizes of the files generated from MESHIO-IO, which do not have any meta data, such as dimensionality, number of processors, and data type, are 332,980,440 bytes for one processor and 340,971,970,560 bytes for 1024 processors. The sizes of files generated from the meshio library are 34,400,795 bytes for one processor and 35,207,432,315 bytes for 1024 processors. Figure 6 shows the write rates of MPI-IO, meshio, resilient meshio, buffered meshio, and resilient buffered meshio. Figure 7 shows the write rates of meshio, resilient meshio, buffered meshio, and resilient buffered meshio measured by MPI-IO, meshio, and buffered meshio respectively.

6. Acknowledgments

The work presented here has been supported by the United States Department of Energy through the Advanced Simulation and Computing Program. LA-UR-13-28525.

References

[1] Brown, S., Folk, M., Goucher, G., Rew, R., “Software for Portable Scientific Data Management”, Computers in Physics, vol. 7, no. 3, pp.304-308 (1993).

[2] Miller, M. C., Reus, J. F., Matzke, R. P., Arrighi, W. J., Schoof. L. A., Hitt, R. T., Espen, P. K., “Enable Integration of High Performance, Scientific Computing Applications: Modeling Scientific Data with the Sets and Fields (SAF) Modeling System”, in Computational Science- ICCS 2001, Alexandrov et al. (Eds.), Springer-Verlag Berlin Heidelberg 2001, pp.158-167, 2001.

[3] Thauser Th., “Parallel I/O for the CGNS System”, 42nd AIAA Aerospace Sciences Meeting and Exhibit, AIAA 2004-1088, Reno, Nevada, January, 2004.

[4] Rew, R. K., Davis, G., “NetCDF: An Interface for Scientific Data Access”, IEEE Computer Graphics and Applications, vol.4, pp 72-82, July (1990).

[5] Roberts, L., J., “Silo User’s Guide”, University of California Research Lab Report, Lawrence Livermore National Laboratory, UCRL-MA-118751-REV-1 (2000).

[6] William W. Dai, Rob Aulwes, Michael Gaeta, and Ron Pfaff, Unified Data Model (UDM): A Library for Parallel IO and Data Management, The Proceedings of the 2007 International Conference on Parallel amd Distributed Processing Techniquies and Appliciation,Arabnia et al (Eds.), CSREA Press 2007, Vol.II, pp.697-702, 2007.

[7] K. Guo, W.-K. Liao, A. Nisar, A. Choudhary, R. Ross, R. Latham, Using Subfiling to Improve Programming Flexibility and Performance if Parallel Shared-file I/O, The Proceedings of the International Conference on Parallel Processing, Vienna, Austria, 2009.

Parallelizing Matrix Exponential based Solver on Shared Memory Systems using Cilk

Abdul Jabbar Saeed Tipu, Ammar Hasan, Mohsan Jameel, and Aamir Shafi
School of Electrical Engineering and Computer Science,
National University of Sciences and Technology, Islamabad, Pakistan

Abstract— *Matrix Exponential based algorithm (MEXP) is a recently developed method for solving a positive definite system of linear equations. MEXP already outperforms other state of the art algorithms, such as the Preconditioned Conjugate Gradient method (PCG), in most cases, on customizable hardware platforms such as FPGAs or ASICs. In this paper we have analyzed the performance of MEXP on multicore hardware using a shared-memory model called Cilk and compare it with the Conjugate Gradient method (CG) and PCG. Our multithreaded MEXP outperforms the Cilk based PCG and CG methods in terms of parallelism and execution time as we increase numbers of cores. The comparison of the performance for the tested benchmark problems shows that parallel MEXP relatively gives almost 2 to 3 times more speedup than parallel PCG and 5 to 8 times more speedup than parallel CG. Thus, MEXP is more parallelizable and scalable than both PCG and CG.*

Keywords: MEXP, PCG, CG, Cilk, parallelism and speedup

1. Introduction

Solving a system of linear equations is one of the core problems in numerical and scientific computing. A large number of direct and iterative methods exist for solving problem of a linear system [1], [2]. The iterative methods are preferred over direct methods in case of large and sparse matrices as they are often computationally efficient in comparison. Iterative solvers approximate the solution in successive iterations and can be terminated when a desirable accuracy is achieved. However, the iterative methods may take a lot of iterations if the problem is ill-conditioned and could result in a higher computational time. In iterative methods preconditioners are often used to improve the condition number of the problem. Preconditioned Conjugate Gradient method (PCG) [3] is one of the popular iterative methods for solving systems of linear equations with symmetric positive definite matrices. One of the co-authors, Hasan [4] has recently invented a new method called the Matrix Exponential based algorithm (MEXP), for solving a positive-definite system of linear equations. It is also an iterative method. It was developed to run on the customizable hardware platforms i.e. FPGAs (Field-Programmable Gate Arrays), ASICs (Application-Specific Integrated Circuit), etc. On such platforms MEXP can exploit its parallelism and outperforms the state of the art Preconditioned Conjugate Gradient (PCG) in most of the cases. There are two main reasons for this. The MEXP method involves computations that are easily parallelizable on such architectures. The second reason is that the convergence of MEXP is logarithmically proportional to the condition number of a linear system while the convergence of conjugate gradient is proportional to the square root of the condition number [10], which gives MEXP an advantage over PCG.

In this paper we have analyzed the performance of the MEXP algorithm on a shared-memory (multicore) system to discover its potential of parallelism. The core component of MEXP which brings the linear system to its solution is matrix multiplication that has high computational intensity. The parallel approach of matrix multiplication exists in [9] which has high parallelism and gives almost linear speedups due to which MEXP is expected to be highly parallelizable on multicore machine.

In this research all the parallelization is done with Intel Cilk Plus [7] and tested on multicore processor. Intel Cilk Plus is the one of the multicore programming models and technologies in the landscape of parallel shared-memory architectures. The other competing technologies are PThreads, OpenMP, TBB, etc. The reason for choosing Cilk is its powerful work-stealing runtime system [8] which does not let any thread to remain idle if it has completed the task assigned to it and it also takes care of the underlying details like load balancing, synchronization and communication protocols.

The Florida Matrices [11] have been used as testing benchmarks in our performance evaluation. The Florida Matrix collection includes problems arising in the areas of power network, structural engineering and computational fluid dynamics. The PCG and CG methods do not contain the ample amount of parallelism to be exploited on multicore hardware due to which their computational cost and execution time does not decrease significantly on increasing threads. The MEXP method exploits full parallelism which significantly reduces its computational cost and execution time on increasing worker count. It shows speedups near ideal case unlike PCG and CG. The parallel MEXP yields almost 3 times more speedup relatively than parallel PCG on 8-cores whereas it yields almost 5 to 8 times more speedup than parallel CG.

In rest of the paper, Section 2 explains design and implementation of multi-threaded MEXP, Section 3 explains the Cilk based parallel PCG, Section 4 shows the performance evaluation results on a multicore system and Section 5 tells the conclusion and future work.

2. Multithreaded Matrix Exponential based Algorithm

Matrix Exponential based algorithm (MEXP) is a new iterative method for solving a system of linear equations. Some of the advantages of MEXP are: 1) the number of iterations taken by MEXP vary logarithmically with the condition number of the problem as compared to the square root of condition number for most other iterative methods, 2) when parallelism is exploited in customizable hardware platforms, the computational complexity of MEXP is lower than most other iterative methods, and 3) unlike other iterative methods, MEXP does not require the use of preconditioners.

Consider a system of linear equations $Fx=g$, where $F \in R^{n\times n}$ is a symmetric and positive definite matrix, and $x \in R^n$ and $g \in R^n$ are vectors. The MEXP algorithm for solving the above system of linear equations is as follows:

```
MATRIX-EXPONENTIAL-METHOD($F$, $g$)
1) Find $\lambda_{max} = ||F||_\infty$
2) $Y = \begin{bmatrix} I_n - F(1/\lambda_{max}) & g(1/\lambda_{max}) \\ 0 & 1 \end{bmatrix}$
3) while(1)
4)     if($||Y_{11}||_\infty / ||Y_{12}||_\infty <$ tolerance)
5)     else $Y = Y^2$
6) $x = Y_{12}$
```
Algorithm 1:Matrix Exponential based method.

In Algorithm 1, λ_{max} is an upper bound of the largest eigenvalue of F. It can be easily computed by finding the infinity norm of F, $||F||_\infty$. Y is matrix of order n+1 with four sub matrices. Y_{11} is initialized with $I_n - F(1 / \lambda_{max})$ where I is identity matrix, Y_{12} with $g(1 / \lambda_{\max})$, Y_{21} with zero and Y_{22} with 1. The algorithm is executed until a desired tolerance is achieved.

2.1 Parallelizing Matrix Exponential based Algorithm

The parallelizing strategy is to parallelize those parts of MEXP that are compute-intensive such as computing the infinity-norm, initialization of matrix Y and matrix-multiplication. All these matrix operations consist of independent sub tasks which can be assigned to different threads and can be executed in parallel to complete their main tasks using reducers [5] in case of data race and ultimately to exploit MEXP‘s parallelism on multicore system . This makes MEXP naturally parallelizable. Following is the Cilk based implementation of MEXP:

```
PARALLEL-MEXP($F$, $g$)
1) $n = F.rows$
2) $Y =$ create matrix of order $(n+1)$
3) $R =$ create matrix of order $(n+1)$
4) $lambda\_max =$NORM-INF-MAT($F$, $n$)
5) INIT-Y-MATRIX($Y$, $F$, $n+1$, $lambda\_max$)
6) while(1)
7)     $a =$NORM-INF-MAT($Y_{11}$, $n$)
8)     $b =$NORM-INF-VECT($Y_{12}$, $n$)
9)     if($a/b$ <tolerance) break
10)    else PARALLEL-GEMM($Y$, $Y$, $R$)
11)    $Y = R$
12) $x = Y_{12}$
```
Algorithm 2: Multithreaded MEXP.

```
NORM-INF-MAT($A$, $n$)
1) $temp =$ create array of size $n$
2) $reducer\_max\ maximum$;
3) cilk_for $i = 0$ to $n-1$
4)     for $j = 0$ to $n-1$
5)         $temp_i += $ abs($A_{ij}$)
6)     $maximum$.calc_max($temp_i$)
7) return $maximum$.get_value()
```
Algorithm 3: Infinity-norm of a matrix.

```
INIT-Y-MATRIX($Y$, $F$, $n$, $lambda\_max$)
1) cilk_for $i = 0$ to $n-1$
2)     cilk_for $j = 0$ to $n-1$
3)         if($i == j$)
4)             $Y_{ij} = 1 - F_{ij} * 1/lambda\_max$
5)         else
6)             $Y_{ij} = -F_{ij} * 1/lambda\_max$
7) cilk_for $i = 0$ to $n-1$
8)     $Y_{in} = g_i * 1/lambda\_max$
9) cilk_for $j = 0$ to $n-1$
10)    $Y_{nj} = 0$
11) $Y_{nn} = 1$
```
Algorithm 4: Initializing matrix Y.

```
NORM-INF-VECT($V$, $n$)
1) $temp =$ create array of size $n$
2) $reducer\_max\ maximum$;
3) cilk_for $i = 0$ to $n-1$
4)     $temp_i =$abs($V_i$)
5)     $maximum$.calc_max($temp_i$)
6) return $maximum$.get_value()
```
Algorithm 5: Infinity-norm of a vector.

Algorithm 3, 4 and 5 are supporting the parallel implementation of parallel MEXP (Algorithm 2). The Algorithm 3 finds absolute sum of each row of matrix as an independent task in parallel and then it finds their maximum using *reducer_max* object [12] which is applied instead of synchronization locks to avoid race conditions. By using locks the execution becomes sequential while using reducers the execution remains parallel. Algorithm 4 initializes each cell independently in parallel to set matrix *Y* to its starting condition. As the tasks generated to fill matrix *Y* are independent therefore, there is no chance of race condition. Algorithm 5 finds the absolute value of each cell of vector as an independent task in parallel and then it finds their maximum using a *reducer_max* object because race conditions can lead to wrong computational results with simple variable.

In MEXP, the square matrix-multiplication takes longer than any other parts of this algorithm. The time-complexity of this part is n^3. It involves n^2 dot-products. For this purpose we have used the parallel Cilk based implementation of BLAS `gemm()` routine in PARALLEL-MEXP which is PARALLEL-GEMM and can be found in [6]. The grain size used in this implementation is 50. Since this is a parallel-recursive (divide and conquer) approach, at each recursive call it divides the problem into a sub-problem of order $n/2$ and executes each call in parallel until the problem size is reduced to order of 50. There is also another technique called the Strassen‘s multiplication [9] which has complexity of O($n^{2.81}$) and its Cilk based implementation is also available but it is numerically unstable therefore, we could not have used it here.

3. Cilk based Preconditioned Conjugate Gradient

In this section we present our parallel implementation of PCG using Cilk which is compared with PARALLEL-MEXP in terms of speedups and execution time. Below is the layout of the algorithm:

```
PARALLEL-PCG(F, g, M, n)
1)  $x, r, p, z, t, q$ = create array of size $n$
2)  $iteration = 0; rho\_new = 0; beta = 0;$
3)  $r = g$
4)  while(1)
5)      PARALLEL-GEMM($M, r, z$)
6)      $rho\_old = rho\_new$
7)      $rho\_new$ = DOT-PRODUCT($r, z, n$)
8)      $iteration++$
9)      if($iteration == 1$) $p = z$
10)     else $beta = rho\_new/rho\_old$
11)     cilk_for($i = 0$ to $n-1$) $p_i = z_i + beta \times p_i$
12)     PARALLEL-GEMM($A, p, q$)
13)     $alpha$ = DOT-PRODUCT($r, z, n$)/ DOT-PRODUCT($p, q, n$)
14)     cilk_for($i = 0$ to $n-1$) $x_i = x_i + alpha \times p_i$
15)     cilk_for($i = 0$ to $n-1$) $r_i = r_i - alpha \times q_i$
16)     PARALLEL-GEMM($A, x, t$)
17)     cilk_for($i = 0$ to $n-1$) $t_i = t_i - g_i$
18)     if(NORM($t, n$) / NORM($g, n$) < tolerance) break
19)     if($iteration == n \times 4$) break
```
Algorithm 6: Cilk based Precondition Conjugate Gradient (PCG).

In Algorithm 6, line 1 *x* is the initial solution or condition of the problem *F*, which is zero by default. In line 14 *x* is being updated repeatedly in main loop of this method. In every next iteration *x* gets closer to the solution. From line 16 to 19, it computes residual value and checks whether it is upto certain level of tolerance or not. If tolerance level gets achieved after some iterations then last updated value of *x* is finally the solution of problem $F.x = g$.

In this algorithm, *M* is a Chebyshev preconditioner. The preconditioning is also parallelized and its computational cost is also included in performance evaluation which takes O(n^3) operations of PARALLEL-GEMM routine. In performance evaluation we have compared MEXP with both PCG and CG. Although the performance of PCG varies depending upon the preconditioner being used. The reason for choosing Chebyshev preconditioner is that it is a general preconditioner and its computations are more parallelizable than the other preconditioners. This is why in the original paper of MEXP [4], authors have used this preconditioner while comparing PCG and MEXP, and for the same reason we have used this preconditioner in this paper. Following are the parallel algorithms that support the implementation of PARALLEL-PCG:

```
DOT-PRODUCT(a, b, n)
1) $reducer\_opadd\ sum;$
2) $sum$.set_value(0)
3) cilk_for $i = 0$ to $n-1$
4)     $sum = sum + a_i \times b_i$
5) return $sum$.get_value()
```
Algorithm 7: parallel dot-product.

```
NORM(V, n)
1) $reducer\_opadd\ sum;$
2) $sum$.set_value(0)
3) cilk_for $i = 0$ to $n-1$
4)     $sum = sum + V_i^2$
5) return sqrt($sum$.get_value())
```
Algorithm 8: parallel norm-2.

In PARALLEL-PCG, Algorithm 7 computes the dot-product of two vectors a and b of size n in parallel and returns the result. Algorithm 8 computes the norm-2 of a vector V of size n in parallel and returns the result. In both of the above algorithms the reducer object for addition is used so that at the end of parallel iterations their results will be merged in a single result, otherwise, it can introduce the data-race conditions on a simple variable.

4. Performance Evaluation

In this section, the performance of all the algorithms is compared in terms of speedups and execution time. The performance evaluation is done on 8-cores Intel (R) Xeon (R) E5520 2.27 GHz and 8 MB of cache with Linux platform using Intel Cilk Plus version 14.0.1 (Intel Composer XE 2013). Compiler optimization switch used is âĂŞO2 by default and the tolerance level is set to 10^{-8} for parallel MEXP and 10^{-6} for parallel PCG and CG. Florida Matrices [11] have been used as our testing benchmarks. These results are gathered by using Cilk utility called "Cilkview scalability analyzer" which tells us the speedups and execution time for different number of CPU cores (workers or threads). First we show the speedups among MEXP, PCG and CG. From Figure 1 to 8, n denotes problem size, $\kappa(F)$ denotes condition number of problem matrix F, s denotes iterations of MEXP, k denotes iterations of PCG and v denotes iterations of CG.

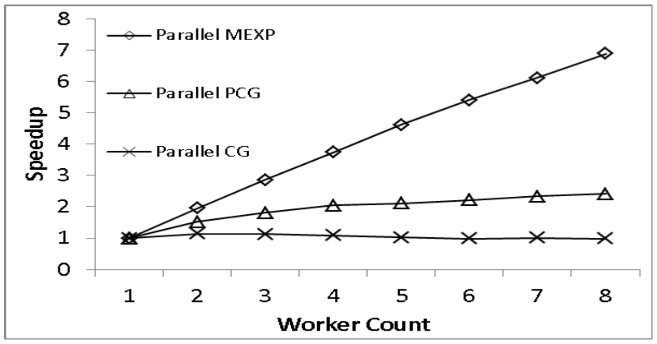


Fig. 1: Speedups on 494_bus (power network) matrix with $n = 494$, $\kappa(F) = 3.9 \times 10^6$, $s = 24$, $k = 133$ and $v = 1313$.

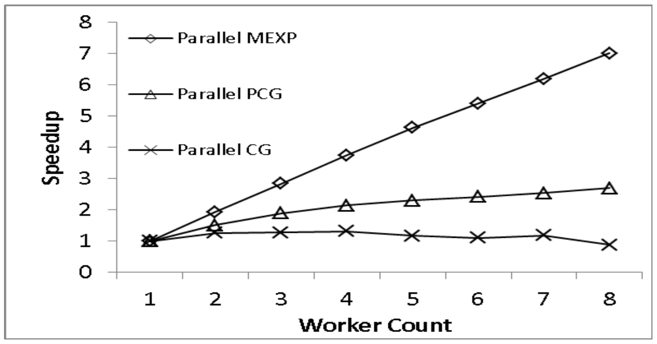


Fig. 2: Speedups on tomography_500 (computer vision) matrix with $n = 500$, $\kappa(F) = 6.26 \times 10^7$, $s = 29$, $k = 3$ and $v = 408$.

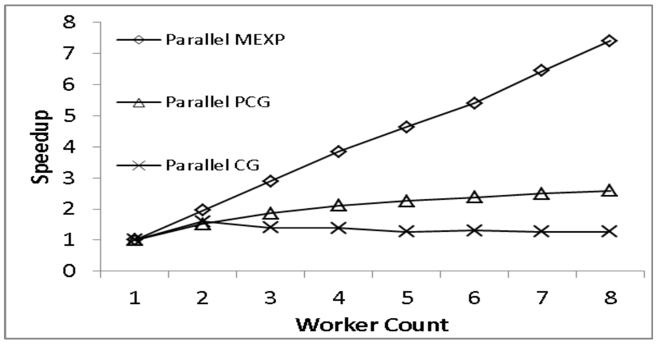


Fig. 3: Speedups on 685_bus (power network) matrix with $n = 685$, $\kappa(F) = 5.31 \times 10^5$, $s = 22$, $k = 71$ and $v = 555$.

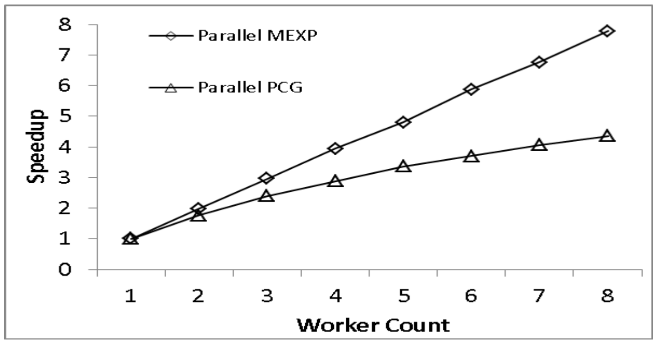


Fig. 4: Speedups on msc01050 (structural problem) matrix with $n = 1050$, $\kappa(F) = 9.51 \times 10^6$, $s = 30$ and $k = 2$.

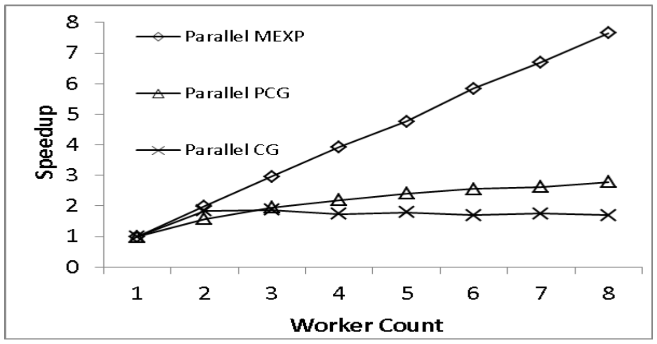


Fig. 5: Speedups on bcsstk27 (structural problem) matrix with $n = 1224$, $\kappa(F) = 7.71 \times 10^4$, $s = 19$, $k = 70$ and $v = 913$.

In Figure 1 to 8, MEXP shows more speedups than PCG and CG which actually proves the point that it has more parallelism than CG. The cause of this result is repeated use of parallel-recursive matrix multiplication technique which has high performance in terms of parallelism and scalability. In terms of relative speed of MEXP over PCG on 8 cores ($\text{Speedup}_{\text{MEXP}}/\text{Speedup}_{\text{PCG}}$), it has approximate values from 2 to 3 in these cases and relative speedup of MEXP over CG ($\text{Speedup}_{\text{MEXP}}/\text{Speedup}_{\text{CG}}$) is approximately 5 to 8. MEXP

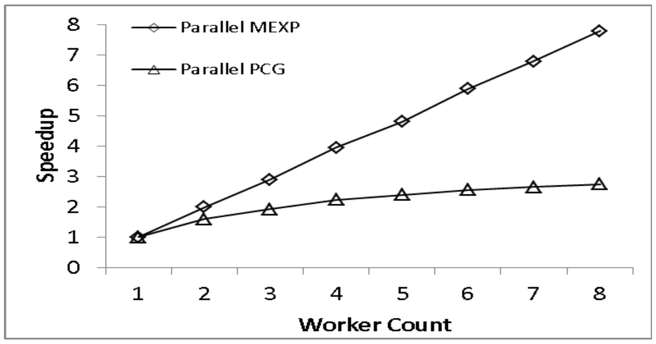


Fig. 6: Speedups on bcsstk13 (computational fluid dynamics) matrix with $n = 2003$, $\kappa(F) = 4.57 \times 10^{10}$, $s = 38$ and $k = 459$.

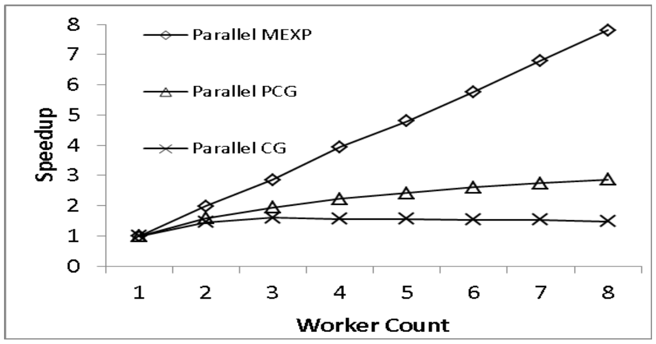


Fig. 7: Speedups on cage9 (directed weighted graph) matrix with $n = 3534$, $\kappa(F) = 72.3718$, $s = 7$, $k = 4$ and $v = 74$.

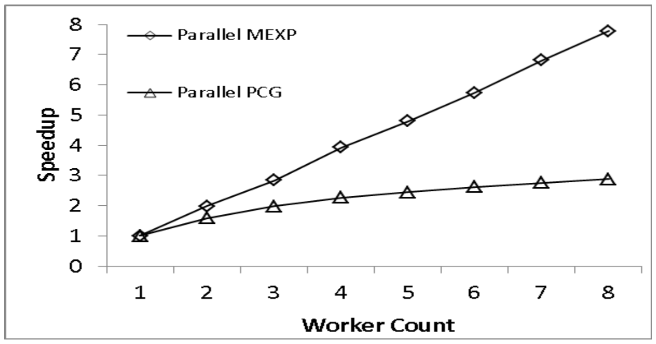


Fig. 8: Speedups on bcsstk15 (structural problem) matrix with $n = 3948$, $\kappa(F) = 7.97 \times 10^{9}$, $s = 36$ and $k = 158$.

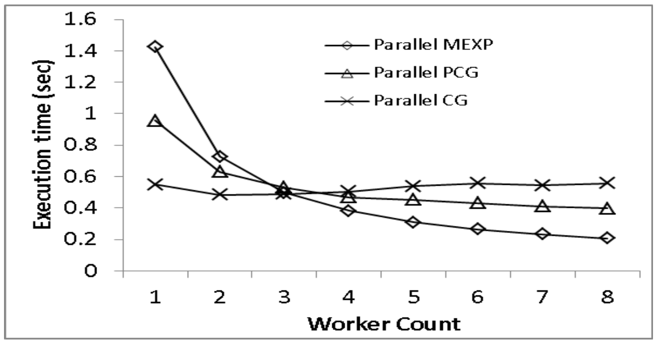


Fig. 9: Execution time on 494_bus matrix.

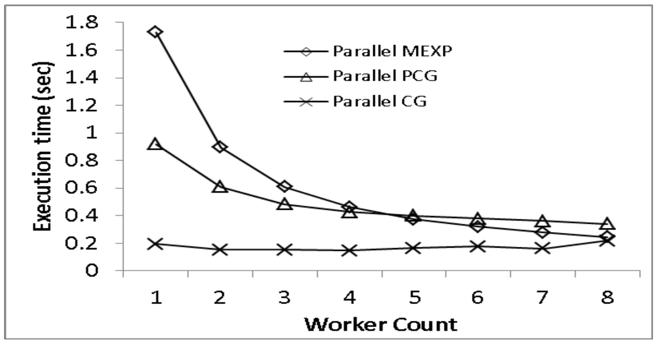


Fig. 10: Execution time on tomography_500 matrix.

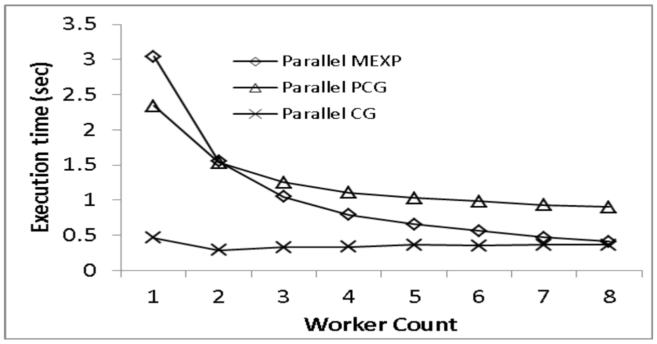


Fig. 11: Execution time on 685_bus matrix.

exhibits more parallelism than CG because it has n times more work than CG. When this ample amount of work is divided on physical number of cores, it shows almost linear speedup and high parallelism. It does not mean that MEXP beats PCG in terms execution time in every case but in most cases. From Figure 9 to 16 respectively shows the execution time graph of MEXP, PCG and CG for 8-cores on same data.

In Figure 9 to 16, parallel MEXP beats parallel PCG in most cases, in terms of execution time, as we increase the number of cores. As far as simple parallel CG is concerned, in some cases MEXP beats CG from 3 to 8 cores and in other cases it is expected to beat CG for more than 8 cores because technique used to parallelize MEXP is much more scalable (i.e. parallel recursive divide-and-conquer matrix-multiplication (PARALLEL-GEMM) [6], [9]) than parallel CG. These results could have been more clear if we would have more processing cores but due to limited resources we were just restricted to 8 number of cores. In Figure 15 the case is different because matrix problem is not ill-conditioned. It has very low condition number value as compare to other problems in this performance evaluation thus it produces very less workload and outperforms MEXP while MEXP performs better on ill-conditioned problems. In Figure 12, 14 and 16, CG is not shown because it fails

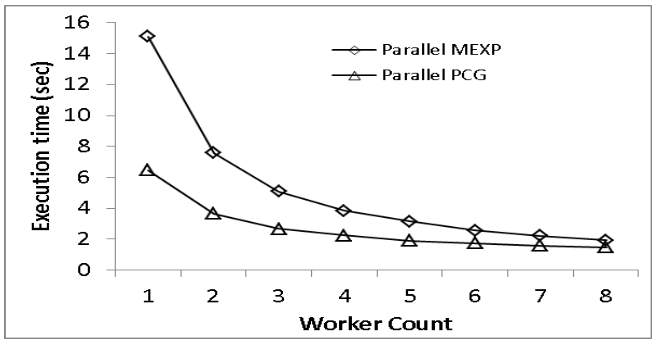


Fig. 12: Execution time on msc01050 matrix.

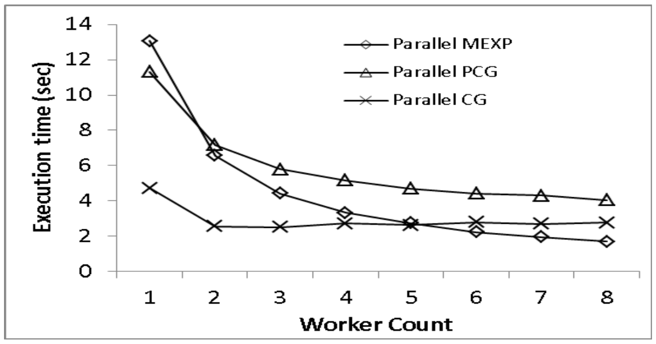


Fig. 13: Execution time on bcsstk27 matrix.

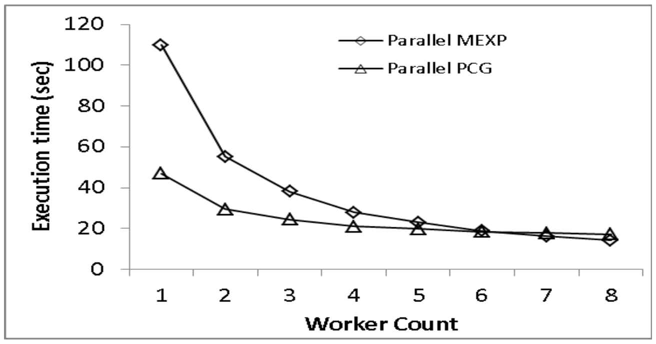


Fig. 14: Execution time on bcsstk13 matrix.

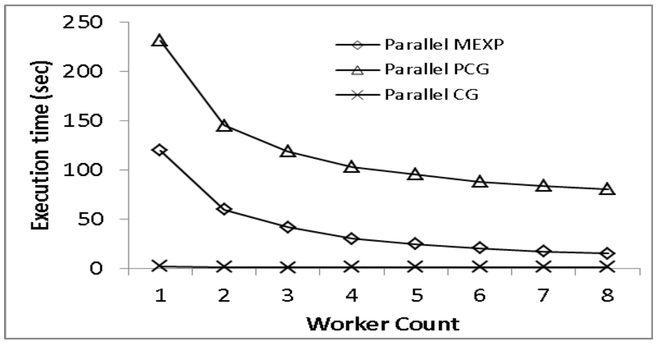


Fig. 15: Execution time on cage9 matrix.

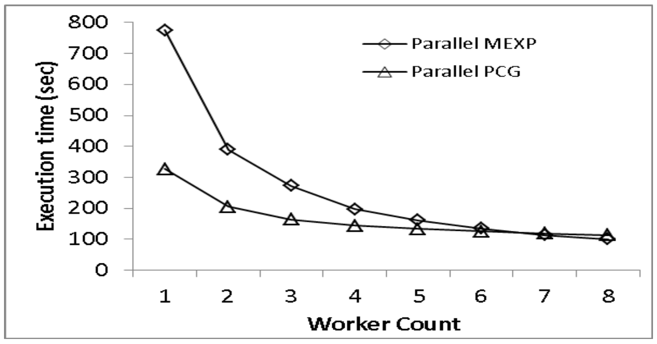


Fig. 16: Execution time on bcsstk15 matrix.

to converge to solution on these problems. The reason is distribution of eigenvalues are clustered together. In that case people use specific preconditioners for specific problems but that adds the preconditioning overhead whereas MEXP does not require preconditioners for any sort of problem. Therefore, it is evident from these results and analysis that for shared-memory hardware architecture (on increasing the number of processing cores) MEXP performs better than PCG and CG in most cases for ill-conditioned and clustered eigenvalues problems.

5. Conclusions

In this research, our motivation was to discover the parallelism of the matrix exponential based algorithm (MEXP) on multicore hardware platform. We parallelized MEXP using one of the shared-memory platforms called Intel Cilk Plus and exploit it on multicore system. Parallel MEXP shows higher parallelism and scalability than parallel PCG and parallel CG and it outperforms both of these methods in terms of both speedups and execution time (in most cases). Parallel MEXP shows approximately 2 to 3 times more relative speedup than parallel PCG and 5 to 8 times more relative speedup than parallel CG.

As far as future work is concerned MEXP can be parallelized with other shared-memory models and can be compared to Cilk based MEXP. The performance of MEXP can also be explore in other various applications of numerical scientific computing and control system theory where solving a system of positive-definite linear equations over large data sets is required.

References

[1] M. T. Chu, "Linear algebra algorithms as dynamical systems", *ActaNumerica*, 2008, 17:187.

[2] James M. Ortega, "Stability of difference equations and convergence of iterative processes", *SIAM Journal on Numerical Analysis*, 10(2):268282, 1973.

[3] Magnus R. Hestenes and Eduard Stiefel, "Methods of Conjugate Gradients for Solving Linear Systems", *Journal of Research of the National Bureau of Standards*, Research Paper 2379, Vol. 49, No. 6, Dec. 1952.

[4] Ammar Hasan, Eric C. Kerrigan and George A. Constantinides, "Solving a positive definite system of linear equations via the matrix exponential", *Decision and Control and European Control Conference*

(CDC-ECC), 50th IEEE Conference, dated 12−15 Dec. 2011, Orlando, FL: pp.2299−2304.

[5] Matteo Frigo, Pablo Halpern, Charles E. Leiserson and Stephen Lewin-Berlin, "Reducers and Other Cilk++ Hyperobjects", *SPAA'09*, Aug. 11−13, 2009, Calgary, Alberta, Canada.

[6] Michael McCool, Arch Robison and James Reinders, *Structured Parallel Programming*, Version 3: November 21, 2013. Code examples available at: http://parallelbook.com/sites/parallelbook.com/files/code20131121.zip.

[7] (2009) Intel Cilk Plus. [Online] . Available at: http://software.intel.com/en-us/articles/intel-cilk-plus/.

[8] Robert D. Blumofe, Christopher F. Joerg, Bradley C. Kuszmaul, Charles E. Leiserson, Keith H. Randall and Yuli Zhou, "Cilk: An Efficient Multithreaded Runtime System", *In the Proceedings of the Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '95)*, Jul. 19−21 1995, Santa Barbara, California.

[9] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, *Introduction to Algorithms*, 3^{rd} ed., The MIT Press, London, England, 2009.

[10] James Demmel, *Applied Numerical Linear Algebra*, MIT, Sep. 1996.

[11] T. A. Davis and Y. Hu., "The University of Florida Sparse Matrix Collection", *ACM Transactions on Mathematical Software*, Vol 38, Issue 1, pp 1:1 - 1:25, 2011. Available at: http://www.cise.ufl.edu/research/sparse/matrices.

[12] (2009) Demo of reducer_max. [Online]. Available at: https://www.cilkplus.org/tutorial-reducer-max.

ANALYZING THE EFFECT OF CONTEXTUAL INFORMATION IN NOISE BASED CLASSIFIER ON SATELLITE IMAGES

Rakesh Dwivedi[a,], *S. K. Ghosh[a] , Anil Kumar[b],

[a]Indian Institute of Technology Roorkee, India
[b]Indian Institute of Remote Sensing, Dehradun, India

KEYWORDS: **Fuzzy c-means clustering, Noise clustering, contextual, Markov Random field(MRF), Discontinuity Adaptive, Edge preservation**

ABSTRACT:
At present a large number of earth observation satellite at different spatial resolution are provide remote sensing data for land use classification. Due to the continuous nature of real world phenomena, mapping of, land cover classes is a challenge. Further presence of mixed pixels also decreases the accuracy of image classification. Fuzzy classification technique such as Fuzzy *c*-Means (FCM), Possibilistic *c*-Means(PCM) and Noise Clustering (NC) can be used to handle mixed pixels. Although these classifiers have the advantage of classifying mixed pixels by assigning membership value, yet these non contextual based classifiers do not incorporate spatial contextual information of a pixel. Use of context eliminates the problem of isolated pixels and improves the classification accuracy. In this paper a hybrid algorithm, based upon contextual and NC classifier have been developed using MRF models. Smoothness prior and four discontinuity adaptive prior has been used to incorporate contextual information with NC. The developed discontinuity adaptive contextual NC classifier has been tested both on coarse and fine resolution dataset i.e. AWFIS and LISS-III having spatial resolution of 56 m and 23.5m respectively. It is expected that discontinuity adaptive prior models, improves the overall classification accuracy by preserving the edges at boundaries and the classified output is spectrally and spatially consistent.

I. INTRODUCTION

A hard classification technique of satellite data does not take into account gradual spatial variation in land cover classes. To incorporate the gradual boundary change problem researchers had been proposed the ‘soft’ classification techniques that decompose the pixel into class proportions, minimizes the mixed pixel problem using fuzzy logic (Fisher, 1997, Maselli *et al.*, 1994). Fuzzy classification is a soft classification technique (Binaghi and Rampini, 1993), which deals with vagueness in class definition (Foody *et al.*, 1996). Therefore, it can model the gradual spatial transition between land cover classes.

Fuzzy c-Means (FCM) (Bezdek, *et al.*, 1980; Ehrlich *et al.*, 1984., Bezdek *et al.*, 1987) is an unsupervised clustering algorithm which has been widely used to find fuzzy membership grades between 0 and 1. The aim of FCM is to find cluster centres in the feature space such that it minimizes the intra-class variation and maximizes the inter-class distances using an objective function. Standard FCM algorithm considers the spectral characteristics.

Fuzzy *c*-Means supervised classification algorithm has been widely used to classify satellite images with ambiguous land cover classes. It is a popular fuzzy set theory based soft classifier, which handles the vagueness of a pixel at sub-pixel level. FCM has been successful in assigning the membership (u_{ij}) of a pixel to multiple classes but this assignment is relative to total number of classes defined and not absolute (Krishnapuram and Keller, 1993, Foody 2000). This is due to the constraint imposed on the membership values as given by the Eq. (1)

$$\sum_{i=1}^{c} u_{ij} = 1 \qquad for\, all\, i \tag{1}$$

The main motivation behind Possibilistic c-Means (PCM) relates to the relaxaction of the constraint on membership value in (1) and gives absolute membership value, as stated by Eq. (2)

$$\max_i u_{ij} > 0 \qquad for\, all\, j \tag{2}$$

In case of PCM, this membership value represents the “degree of belongingness or compatibility or typicality”, contrary to that represented by FCM, where it is, “degree of sharing”.

An important aspect in classification is the presence of noise, which may have been introduced at any stage of data collection and transmission. This affects the performance of any classification algorithm. Literature reveals that a good solution to this problem does not exist. An ideal solution would be one where the noise points get automatically identified and removed from the data. A concept of "Noise Cluster” can be introduced such that noisy data points may be assigned to the noise class. The approach is developed for an objective functional type (K-means or fuzzy K-means) algorithm, and its ability to detect 'good' clusters amongst noisy data has been aptly demonstrated by (Dave, 1991). The approach is applicable to both fuzzy supervised classification algorithms as well as regression based methods.

In supervised classification, validity plays a pivotal role in achieving a robust classification because without the concept of validity, it is neither possible to separate the good points from the noise points and outliers nor access the quality of the solution. The solution to the robust clustering problem requires that the algorithm reject noise data before it computes the parameter estimates (Dave and Krishnapuram, 1997).

The purpose of study of noise clustering without entropy is not only to establish a connection between fuzzy set theory and robust statistics, but also to discuss and compare several popular clustering methods from the point of view of robustness (Dave, 1990) and (Foody *et al.*, 1995).

The aim of this paper is to study the behaviour of associated learning parameters of noise clustering with contextual for optimization estimation, using different fuzzy based functions, which is used for classification of multi-spectral remote sensing data in sub-pixel mode. Very often land cover classes changes gradually from one to another, therefore in such condition it is difficult to define sharp boundaries between two land cover classes and NC with contextual based classification techniques can be used to represent such conditions. In the next section, the details of parameters considered in FCM, PCM and noise clustering with contextual are provided.

1. CLASSIFIERS AND ACCURACY ASSESSMENT APPROACHES

1.1 Fuzzy *c*-Means Approach (FCM)

Fuzzy c-Means (FCM) was originally introduced by Bezdek (1981). In this supervised classification technique each data point belongs to a cluster to some degree that is specified by a membership grade, and the sum of the memberships for each pixel must be unity. This can be achieved by minimizing the generalized least - square error objective function,

$$J_m(U,V)=\sum_{i=1}^{N}\sum_{j=1}^{c}\mu_{ij}{}^{m}\left\|X_i-x_j\right\|_A^2 \qquad (3)$$

Subject to constraints,

$$\begin{aligned}&\sum_{j=1}^{c}\mu_{ij}=1 \; for\ all\ i\\ &\sum_{i=1}^{N}\mu_{ij}>0 \qquad for\ all\ j\\ &0\le\mu_{ij}\le 1 \qquad for\ all\ i,j\end{aligned} \qquad (4)$$

where X_i is the vector denoting spectral response of a pixel i, x is the collection of vector of cluster centers x_j, μ_{ij} are class membership values of a pixel, c and N are number of clusters and pixels respectively, m is a weighting exponent ($1<m<\infty$), which controls the degree of fuzziness, $\left\|X_i-x_j\right\|_A^2$ is the squared distance (d_{ij}) between X_i and x_j, and is given by,

$$d_{ij}^2=\left\|X_i-x_j\right\|_A^2= X_i-X_j{}^{T} A\; X_i-X_j \qquad (5)$$

where A is the weight matrix. Amongst a number of A-norms, three namely Euclidean, Diagonal and Mahalonobis norm, each induced by specific weight matrix, are widely used. The formulations of each norm are given as (Bezdek, 1981),

$$\begin{aligned}&A=I \quad Euclidean\ Norm\\ &A=D_j^{-1} \quad Diagonal\ Norm\\ &A=C_j^{-1} \quad Mahalonbis\ Norm\end{aligned} \qquad (6)$$

Where I is the identity matrix, D_j is the diagonal matrix having diagonal elements as the eigen values of the variance covariance matrix, C_j given by,

$$C_j=\sum_{i=1}^{N} X_i-x_j \quad X_i-x_j{}^{T} \qquad (7)$$

The class membership matrix μ_{ij} is obtained by;

$$\mu_{ij}=\frac{1}{\sum_{k=1}^{c}\left(\frac{d_{ij}^2}{d_{ik}^2}\right)^{1/m-1}} \qquad (8)$$

$$where \quad d_{ik}^2=\sum_{j=1}^{c}d_{ij}^2 \qquad (9)$$

1.2 Possibilistic *c*-Means Approach (PCM)

In PCM, for a good classification is it expected that actual feature classes will high membership value, while unrepresentative features will have low membership values (Krishnapuram and Keller, 1993). The objective function, which satisfies this requirement, may be formulated as;

$$J_m(U,V)=\sum_{i=1}^{N}\sum_{j=1}^{c}\mu_{ij}{}^{m}\left\|X_i-v_j\right\|_A^2+\sum_{j=1}^{c}\eta_j\sum_{i=1}^{N}1-\mu_{ij}{}^{m} \qquad (10)$$

Subject to constraints;

$$\begin{aligned}&\max_j u_{ij}>0 \qquad for\ all\ i\\ &\sum_{i=1}^{N}\mu_{ij}>0 \qquad for\ all\ j\\ &0\le\mu_{ij}\le 1 \qquad for\ all\ i,j\end{aligned}$$

μ_{ij} is calculated from Eq. (8).
In Eq. (10) where η_j is the suitable positive number, first term demands that the distances from the feature vectors to the prototypes be as low as possible, whereas the second term forces the μ_{ij} to be as large as possible, thus avoiding the trivial solution. Generally, η_j depends on the shape and average size of the cluster j and its value may be computed as;

$$\eta_j=K\frac{\sum_{i=1}^{N}\mu_{ij}^m d_{ij}^2}{\sum_{i=1}^{N}\mu_{ij}^m} \qquad (11)$$

Where K is a constant and is generally kept as one. After this, class memberships, μ_{ij} are obtained as;

$$\mu_{ij}=\frac{1}{1+\left(\frac{d_{ij}^2}{\eta_j}\right)^{1/m-1}} \qquad (12)$$

1.3 Noise Clustering with Contextual Algorithm

The idea of noise clustering is based on the introduction of an additional cluster that is supposed to contain all outliers (Dave and Keller., 1997; Tuia and Camps-Valls, 2011; Upadhyay et al., 2013). Feature vectors that are about the noise distance 'δ' or further away from any other prototype vector get high membership degrees to this noise cluster. The noise prototype is such that the distance d_{cj} distance of feature vector x_j from v_c is the fixed constant value

$$d_{cj}=\delta^2, \quad \forall j \qquad (13)$$

The specification of the noise distance depends on several factors, i.e. maximum percentage of the data set to be classified as noise, distance measure, number of assumed clusters and the expansion of the feature space(Klawonn., 2004).

The noise distance proposed in (Krishna.,1998) is a simplified statistical average over the non-weighted distances of all feature vectors to all prototype vectors.

$$\delta^2 = \lambda \frac{\sum_{i=1}^{c-1}\sum_{j=1}^{n} d_{ij}}{n(c-1)} \qquad (14)$$

Where λ is the value of the multiplier used to obtain δ from the average of distances. The memberships of the vectors in the data set to the noise cluster are defined as

$$u_{*j} = 1 - \sum_{i=1}^{c} x_{ij} \qquad (15)$$

Objective function is given by (Krishna.,1998)

$$J_{NC} = \sum_{i=1}^{c}\sum_{k=1}^{n} u_{ik}^{m} d_{ik}^{2} + \sum_{i=1}^{n} \delta^2 (1 - \sum_{i=1}^{c} u_{ik})^{m} \qquad (16)$$

And u_{ik} may be computed as

$$u_{ik} = 1 \Big/ \sum_{j=1}^{c} \left(\frac{d_{ik}}{d_{jk}}\right)^{2/(m-1)} \qquad (17)$$

Where i= 1,……c, k=1,…….n , resolution parameter δ> 0 and weighting exponent m>1.

n= row*column (image size). The distances are defined as

$$(d_{ik})^2 = (x_k - v_i)^T A_i (x_k - v_i) \qquad (18)$$

for all 'k' and i=1to (c-1) .The v_i denotes the mean vector of each class and can be defined as

$$v_i = \sum_{k=1}^{n} u_{ik}^{m} x_k \Big/ \sum_{i=1}^{n} u_{ik}^{m} \qquad (19)$$

Geman and Geman (1984) did maximum a priori (MAP) estimation as statistical criterion using simulated annealing and Gibbs Sampler for MRF based image restorations. It was found improved restorations at low signal-to-noise ratio. This paper explains the equivalence between Gibbs distribution and Markov random field (MRF). The restoration was performed using the simulated annealing theorem which convergence to the global maxima of the posterior distribution. Similar simulated annealing algorithm was used in this research work to find out the global posterior energy without sticking in a local minimum.

Solberg *et al., (*1996*)* used MRF to include context for multisource satellite images. It was found that MRF can model spatial class dependencies as well as temporal class dependencies. MRF model achieved 2% higher classification accuracy when same set of image used for the two different models. Finally it was conclude that MRF model provide better results for classification of multisource satellite images.

The idea of using this hybrid approach of soft classification noise clustering with contextual is a new which helps significantly to eliminate noise pixels and improves the classification accuracy.

The sequence of discussion starts with the objective function of NC, followed by two sub-sections, each corresponding to one of the two objective functions. The objective function of NC discussed in Eq.(16). The main concept of the NC algorithm is the introduction of a single noise cluster that will hopefully contain all noise data points (Cimino *et al.*, 2005). Data points whose distances to all clusters exceed a certain threshold are considered as outlier. This distance is called the noise distance. The basic objective function of NC is given in Eq. (16), includes the information about the distance of the feature vector (that forms the pixel) from the cluster mean in the feature space but it does not include information on spatial context. The spatial context here includes the influence of the neighbouring pixel on the target pixel in the image space.

The MAP-MRF (Maximum A Posterior Solution-Markov Random Field) framework works by maximizing the posterior probability which is related to prior and conditional energy (Eq.(20) to Eq.(24)).

Eq.(20) states the NC objective function formulated using smoothness prior. From now onwards the objective function in Eq.(20) will be referred as NC-S.

$$\mathrm{U}(\mathrm{u}_{\mathrm{ij}}/d) = (1-\lambda)\left[\sum_{i=1}^{c}\sum_{j=1}^{n} u_{ij}^{m} d_{ij}^{2} + \sum_{i=1}^{n}\delta^2 (1-\sum_{i=1}^{c} u_{ij})^{m}\right] + \lambda\left[\sum_{i=1}^{n}\sum_{j=1}^{c}\sum_{j'\in n_i} \beta \, u_{ij} - u_{ij'}{}^{2}\right] \qquad (20)$$

where,

$\mathrm{U}(\mathrm{u}_{\mathrm{ij}}/d)$ = Posterior energy of image μ, given image d.

λ= Weight for spectral and contextual information (smoothness strength).

u_{ij}= Membership value of pixel i of class j.

n= Number of pixels.

m= weighing exponent

$(d_{ij})^2 = (x_j - v_i)^T A_i (x_j - v_i)$

β= weight for neighbors.

n_i= Neighborhood window around pixel i.

δ= resolution parameter

NC objective functions formulated with discontinuity adaptive priors (DA prior) are given in (Eq.(21) to (24)).

$$\mathrm{U}(\mathrm{u}_{\mathrm{ij}}/d) = (1-\lambda)\left[\sum_{i=1}^{c}\sum_{j=1}^{n} u_{ij}^{m} d_{ij}^{2} + \sum_{i=1}^{n}\delta^2 (1-\sum_{i=1}^{c} u_{ij})^{m}\right] + \lambda\left[\sum_{i=1}^{n}\sum_{j=1}^{c}\sum_{j'\in n_i} \left(-\gamma e^{-\frac{\eta^2}{\gamma}}\right)\right] \qquad (21)$$

$$\mathrm{U}(\mathrm{u}_{\mathrm{ij}}/d) = (1-\lambda)\left[\sum_{i=1}^{c}\sum_{j=1}^{n} u_{ij}^{m} d_{ij}^{2} + \sum_{i=1}^{n}\delta^2 (1-\sum_{i=1}^{c} u_{ij})^{m}\right] + \lambda\left[\sum_{i=1}^{n}\sum_{j=1}^{c}\sum_{j'\in n_i} \left(-\frac{\gamma}{1+\frac{\eta^2}{\gamma}}\right)\right] \qquad (22)$$

$$\mathrm{U}(\mathrm{u}_{\mathrm{ij}}/d) = (1-\lambda)\left[\sum_{i=1}^{c}\sum_{j=1}^{n} u_{ij}^{m} d_{ij}^{2} + \sum_{i=1}^{n}\delta^2 (1-\sum_{i=1}^{c} u_{ij})^{m}\right] + \lambda\left[\sum_{i=1}^{n}\sum_{j=1}^{c}\sum_{j'\in n_i} \left(\gamma \ln(1+\frac{\eta^2}{\gamma})\right)\right] \qquad (23)$$

$$\mathrm{U}(\mathrm{u}_{\mathrm{ij}}/d) = (1-\lambda)\left[\sum_{i=1}^{c}\sum_{j=1}^{n} u_{ij}^{m} d_{ij}^{2} + \sum_{i=1}^{n}\delta^2 (1-\sum_{i=1}^{c} u_{ij})^{m}\right] + \lambda\left[\sum_{i=1}^{n}\sum_{j=1}^{c}\sum_{j'\in n_i} \left(\gamma|\eta| - \gamma^2 \ln(1+\frac{|\eta|}{\gamma})\right)\right] \qquad (24)$$

where, all the symbols have common meaning as described above. In addition, γ is the *adaptive interaction function* (AIF) with a value varying between 0 and 1.

1.4 Accuracy Assessment Approach

Silván-Cárdenas and Wang, (2008) developed theoretical grounds, for a more general accuracy assessment of soft classifications, which account for the soft class distribution uncertainty.
In formal grounds, one requires the agreement-disagreement measure to conform Eq. (25), where A and D denote agreement and disagreement operators respectively, where s'_{nk} and r'_{nl} denote the over and underestimation errors at pixel n.

$$C\ s_{nk}, r_{nl} = \begin{cases} A\ s_{nk}, r_{nl} & if\ k = 1 \\ D\ s'_{nk}, r'_{nl} & if\ k \neq 1 \end{cases} \tag{25}$$

$$s'_{nk} = s_{nk} - \min\ s_{nk}, r_{nk}$$
$$r'_{nl} = r_{nl} - \min\ s_{nl}, r_{nl}$$

Practically, it is convenient to express each confusion interval in the form $P_{kl} \pm U_{kl}$ where P_{kl} and U_{kl} are the interval center and the interval half-width, respectively. These are computed as indicated by Eq.(26) and (27), respectively. The general structure of SCM is provided in Silvan-Cardenes and Wang(2008).

$$P_{kl} = \frac{P_{kl}^{\min-\min} + P_{kl}^{\min-least}}{2} \tag{26}$$

$$U_{kl} = \frac{P_{kl}^{\min-\min} - P_{kl}^{\min-least}}{2} \tag{27}$$

With the availability of IRS-P6 satellite data it is possible to acquire spectrally same and spatial different data sets of same area with same acquisition time. Due to the uniqueness of availability of these data sets, soft fraction images generated from coarser resolution data set (e.g. AWIFS, IRS-P6) can be evaluated from fraction images generated from finer resolution data sets (e.g. LISS-III, IRS-P6) as reference data set acquired at same time.

For the uncertainty visualization and evaluation of the classification results, the entropy criterion is proposed. This measure expresses by equation (28).

$$\text{Entropy(x)} = \sum_{i=1}^{j} -\mu(w_i/x)\log_2(\mu(w_i/x)) \tag{28}$$

For high uncertainty, the calculated entropy (Eq. (28)) is high and inverse. Therefore this criterion can visualize the pure uncertainty of the classification results.

2. STUDY AREA AND DATA USED

The study area for the present research work belongs to Sitarganj Tehsil, Udham Singh Nagar District, Uttarakhand, India. It is located in the southern part of the state. In terms of geographic latitude/longitude, the area extends from 28°52’29”N to 28°54’20”N and 79°34’25”E to 79°36’34”E. The area consists of agricultural farms with sugarcane and paddy as one of the few major crops with two reservoirs namely, Dhora and Bhagul reservoir. The images for this research work have been taken from two different sensors namely AWIFS and LISS-III belonging to satellite IRS-P6. The AWIFS dataset used here for classification and LISS III for referencing purposes. This image of Resourcesat-1 acquired on 15th October, 2007.

3. METHODOLOGY

Two datasets namely (AWIFS), and (LISS-III) were geometrically corrected with RMSE less than 1/3 of a pixel and re-sampled using Nearest Neighbour method at 60m, and 20m so that 9 LISS-III pixels corresponds to 1 AWiFS pixels. Fig. 1 shows the flowchart of the methodology adopted to fulfil the objectives of this paper. The six classes of interest, namely deciduous forest, eucalyptus plantation, water bodies, crop land , non-crop land , and non crop moist land have been taken for this study work. Training data was collected with the help of field data and testing was conducted while taking 100 samples per class and total 600 samples randomly selected.

In order to achieve good identification of information high classification accuracy and certain parameters, such as resolution parameter ‘δ’, smoothness controller ‘β’ , Weight for spectral and contextual information (smoothness strength) ‘λ’, and ‘γ’ is the *adaptive interaction function* (AIF) have to be either determined or optimized for NC with Contextual classifier for assessment of optimized values of ‘δ’, ‘λ’, ‘β’ and ‘γ’. The range of resolution parameter δ has been taken from 1 to 10^6 with the interval of 10, while values of smoothness controller ‘β’ is varying from 1 to 10 and contextual information (smoothness strength) ‘λ’ and *adaptive interaction function* (AIF)’ γ, is varying from 0 to 1 respectively. Fuzzy overall accuracy, fuzzy kappa coefficient and uncertainty in accuracy parameters have been estimated for different LISS-III, and AWIFS data sets. It has been observed that for fixed resolution parameter (δ) i, e 10^6, and β=2.0, λ=0.7 and γ=0.2. The fuzzy overall accuracy as well as fuzzy kappa coefficient is optimum as shown in Fig. 3 and 5. It has also observed that uncertainty in fuzzy overall also minimum in a given Fig. 4 and 6. So, it is important to decide what should be the appropriate optimized parameter values to be used in noise clustering with contextual classifier for image classification.

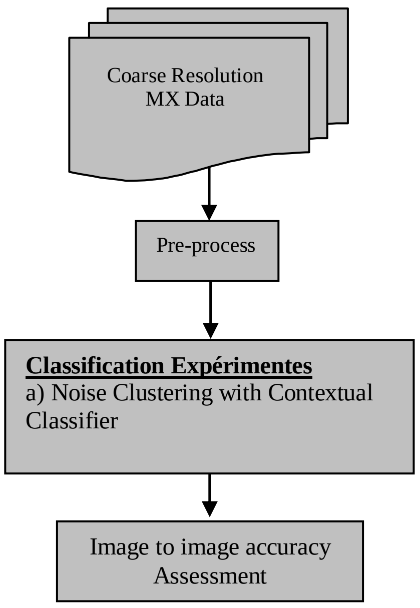

Coarse Resolution MX Data
Pre-process
Classification Expérimentes
a) Noise Clustering with Contextual Classifier
Image to image accuracy Assessment

Fig. 1: Methodology adopted

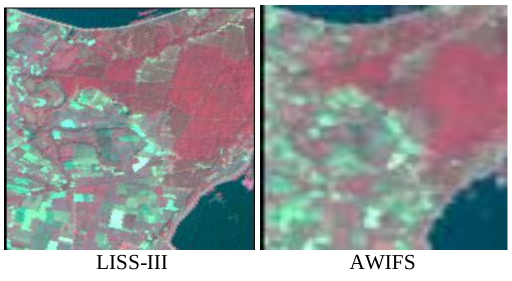


Fig. 2: Location of study area

4. RESULTS AND DISCUSSIONS

In Contextual classification, λ is the smoothness parameter that controls the balance between spectral and spatial information. This parameter is involved in Noise Clustering with Smoothing (NC-S) and in Noise Clustering Discontinuing Adoptive Prior (NC-DA) models. To optimize this parameter entropy was calculated and edge preservation was verified in parallel. It is important to verify edge preservation in this work because here the developed contextual classifier is for discontinuity adaptive image classification. 'β' is the weight given to the neighboring pixel in a window in NC-S model. Similarly like lambda optimization the entropy and edge preservation have been checked to determine the optimized value for β. γ is involved in DA models, it determines the rate at which AIF reaches zero and controls the interaction between two pixels (Li, 1995). Estimation of γ has been done by calculating entropy values as well as by verifying edge preservation. In Fig. 3 optimization of λ,β and γ, for LISS-III image is provided. In Table1 optimized parameter values for all the models for LISS-IV, LISS-III and AWiFS are given. According to Table 1 the overall classification accuracy of NC- contextual discontinuity adaptive classification was improved and it was more than 93% for NC(DA-H3) model . As compare to NC-S the discontinuity adaptive contextual NC classifier achieved 0.5% to 5% improvement in classification accuracy. It has been found that the NC-DA (H3) has achieved the highest overall accuracy with 93.09 % with minimum entropy value0.021. In this work edge verification has been checked for the fraction output images. After comparing the overall accuracy for NC-S and NC-DA it has been found that NC-DA (H3) provides best classification results with highest overall accuracy. So NC-DA (H3) has been taken as a best classifier and edge preservation is verified for this classifier output as given in Table 1. In Fig. 3 optimization of λ, β and γ for AWiFS image is provided. The errors normally occurs at the edges (Tso and Olsen, 2005), thus on preserving the edges, the overall accuracy is improved as it has been noticed in Fig. 3 and Table 1. Thus it is observed that the accuracy of NC-DA (H3) is higher than the NC-S and, other NC-DA models, so it preserves the edges accurately. From Table 1 it is observed that the entropy values are less in NC-DA in comparison to NC-S. So it can be concluded that the classified output from NC-DA has less uncertainty as compared to NC-S.

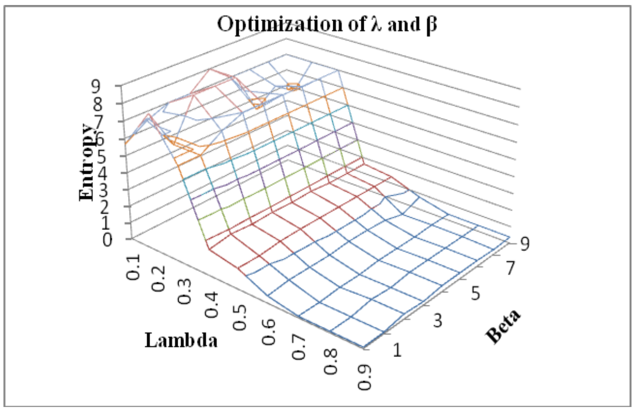


Fig.3: Optimization of λ and β

Table 1: Overall maximum fuzzy accuracy NC with Contextual classifier

Classifier	Optimized parameters	Fuzzy Overall accuracy (SCM)	Uncertainty in fuzzy Overall accuracy	Fuzzy Kappa coefficient	Uncertainty in fuzzy Kappa coefficient	AWiFS Entropy
NC-S	λ = 0.7 and β=2.0	87.5 8	9.82	0.80	0.23	0.6
NC(DA-H1)	λ = 0.7 and λ=2.0	89.0 8	11.05	0.98	0.13	1.25
NC(DA-H2)	λ = 0.7 and λ =2.0	91.7 6	13.98	0.87	0.15	3.54
NC(DA-H3)	λ = 0.7 and λ =2.0	93.0 9	8.54	0.82	0.09	0.021
NC(DA-H4)	λ = 0.7 and λ =2.0	88.0 2	9.06	0.67	0.26	1.34

5. CONCLUSION

In this research work performance of each classifier was estimated based on overall accuracy, fuzzy kappa coefficient, uncertainty in overall accuracy and fuzzy kappa coefficient and entropy. It has been tried to generate fraction outputs from NC-S, and NC-DA classifier. These outputs have been generated from AWIFS as well as LISS-III images of IRS-P6 data. Fuzzy overall accuracy and fuzzy kappa coefficient are relative accuracy assessment but entropy is an absolute uncertainty indicator. From resultant Table 1 and Fig. 3, while monitoring entropy of fraction images for different parameter values, optimum parameter has been obtained for NC-DA(H3) model which gives highest accuracy (SCM) i.e. 93.097%.While using NC-DA(H3) classifier to generate fraction images.The main objective of this research work is to develop a sub-pixel classifier for classifying moderate and coarse spatial resolution multi-spectral dataset using NC-DA MRF models. Another objective is to study the four DA models for NC. The efficiency for DA models is shown in Table1. In this research work a method has been developed to incorporate spatial contextual information in NC using DA MRF models which preserves the edges. The optimized value of λ for LISS-III image is 0.7. Thus it can be concluded that the role of spatial contextual information is less in coarser resolution image whereas for finer resolution image the role of spatial contextual information increases. From the results of accuracy assessment it is found that NC-DA perform better as compare to NC-S. It means that as spatial resolution of the image becomes finer while spatial context increases. It also observed that using DA prior the accuracy was further improved for LISS-III image as compare to AWiFS image and DA priors were used to preserve the edges. Thus it can be concluded that if spatial resolution becomes finer the discontinuities increases and it becomes essential to preserve the edge. It has been found from the accuracy assessment of AWiFS and LISS-III image that NC-DA (H3) performs better in case of coarse and moderate resolution images among all other DA models. It also can preserve the discontinuities for the coarse and moderate spatial resolution image. The entropy values were calculated for the classified output of NC-S and NC-DA, and it was found that the uncertainty in the classified results are less in NC-DA models as compared to NC-S.

6. REFERENCES

Binaghi, E. and Rampini, A., 1993. Fuzzy decision making in the classification of multisource remote sensing data. Optical Engineering 6, 1193-1203.

Bezdek, J.C., 1980. A convergence theorem for the fuzzy ISODATA clustering algorithms. IEEE Trans. on Pattern Anal. Machine Intell., PAMI-2(1): 1-8.

Bezdek, J.C., Ehrlich, R., and Full, W., 1984. FCM: the fuzzy c-means clustering algorithm.Computers & Geosciences, 10(2-3): 191-203.

Bezdek, J.C., Hathaway, R.J., Sabin, M.J., and Tucker, W.T., 1987. Convergence theory for fuzzy c-means: counterexamples and repairs. IEEE Trans. on Syst., Man, Cybern., SMC-17(5):873-877.

Bezdek, J. C. 1981. Pattern Recognition with Fuzzy Objective Function Algorithms, Plenum, New York, USA.

Dave, R. N. 1990. Fuzzy-shell clustering and applications to circle detection in digital images, Int. J. General Syst., vol. 16, pp. 343–355.

Dave, R.N., 1991. Characterization and detection of noise in clustering, Pattern Recognition Letters 12, 657-664.

Dave R. N. and Krishnapuram, R., 1997. Robust Clustering Methods: A Unified View. IEEE Transactions on Fuzzy Systems, VOL. 5, NO. 2, May.

Fisher P, 1997. The pixel: a snare and a delusion, International Journal of Remote Sensing, vol 18, no.3, pp 679-685.

Foody G. M., 1995, Cross-entropy for the evaluation of the accuracy of a fuzzy land cover classification with fuzzy ground data. ISPRS Journal of Photogrammetry and remote sensing, 50, 2-12.

Foody, G. M., 1996, Approaches for the production and evaluation of fuzzy land cover classifications from remotely sensed data. International Journal of Remote Sensing, vol. 17, no. 7, pp. 1317-1340.

Foody, G M 2000. Estimation of sub-pixel land cover composition in the presence of untrained classes, Computers and Geosciences, vol 26,pp 469-478.

Krishnapuram, R and Keller, J M 1993. A possibilistic approach to clustering, IEEE Transactions on Fuzzy System, vol 1,no. 2,pp 98-110.

Kumar, A., Ghosh, S. K. and Dadhwal, V. K., 2006. Study of Sub-Pixel Classification Algorithms for High Dimensionality Data Set, IEEE International Geoscience And Remote Sensing Symposium and 27th Canadian Symposium on Remote Sensing, Denver, Colorado, USA, 31 July – 04 August.

Silván-Cárdenas, J. L., and Wang L., 2008. Sub-pixel confusion–uncertainty matrix for assessing soft classifications, Remote Sensing of Environment (2007), doi: 10.1016/j.rse.2007.07.017.

Tuia, D. and Camps-Valls, G., 2011. Urban Image Classification With Semisupervised Multiscale Cluster Kernels. IEEE journal of selected topics in applied earth observations and remote sensing, vol. 4, no. 1. 1401-1404.

Upadhyay, P., Ghosh, S.K., and Kumar, A., 2013. Moist deciduous forest identification using temporal MODIS data — A comparative study using fuzzy based classifiers, Ecological Informatics, 18, 117–130

SESSION

CLOUD AND GRID COMPUTING

Chair(s)

TBA

Virtual Machine Instance Scheduling in IaaS Clouds

Naylor G. Bachiega, Henrique P. Martins, Roberta Spolon, Marcos A. Cavenaghi
Departamento de Ciência da Computação
UNESP - Univ Estadual Paulista
Bauru, Brazil

Renata S. Lobato, Aleardo Manacero
Departamento de Ciência da Computação e Estatística
UNESP - Univ Estadual Paulista
São José do Rio Preto, Brazil

Abstract— With steady increase in the use of computers, problems such as energy demand and space in data centers are occurring worldwide. Many solutions are being designed to solve these situations, among them is Cloud Computing, which uses existing technologies, such as virtualization, trying to solve problems like energy consumption and space allocation in data centers or large companies. The cloud is shared by multiple customers and allows an elastic growth, where new resources such as hardware or software, can be hired and added anytime in the platform. In this model, customers pay for the resources they use and not for all the architecture involved. Therefore, it is important to determine how efficiently those resources are distributed in the cloud. This paper aims to propose and develop a scheduling algorithm for the cloud that could efficiently define the distribution of resources within the architecture.

Keywords—Cloud Computing; Scheduling Algorith; Virtualization

I. INTRODUCTION

Cloud Computing is seen as a trend in the current scenario in almost all organizations. The advantages of using Cloud Computing are the reduction hardware and maintenance cost, accessibility, flexibility and a highly automated process in which the client does not need to concern about software upgrading [1].

Sabahi [2] defines Cloud Computing as a network environment based on computational resource sharing. In fact, clouds are based on the Internet and try to disguise their complexity for the customers.

II. CLOUD COMPUTING

The Cloud refers to the hardware and software delivered as services over the Internet by data centers. Companies that provide clouds make use of virtualization technologies, combined with their ability to provide computing resources through their network infrastructure.

Cloud Computing uses virtualization to create an environment (cloud), which allocates instances (virtualized operating systems) according to the available resources (physical machines). These virtual machine instances are allocated in accordance with the physical machines that are part of the cloud environment.

A. Classes of Services

According to Buyya [3], Cloud Computing is divided into three service classes according to the type of services offered by providers: Infrastructure as a Service (IaaS), Software as a Service (SaaS) and Platform as a Service (Paas):

- **Software as a Service (SaaS):** in this class, the applications reside on top of the model, offering "software on demand". The applications are accessible from various devices such as a Web browser (e.g.: webmail). The customer does not manage or control the cloud infrastructure, such as network, servers, operating systems, storage, or even the application. The collection for the service, in this case, can be based on the number of users [4].
- **Platform as a Service (PaaS):** provides an environment for developers to build, test and deploy their applications, not caring about the infrastructure, amount of memory or hardware requirements. One example of this class is Google Apps service, where it is offered a scalable environment for developing and hosting Web applications or Microsoft Azure [4].
- **Infrastructure as a Service (IaaS):** in this class of service, the customer has the availability of cloud processing, networking, storage, and other computing resources, where he can install operating systems or any other system. The customer does not manage or control the underlying cloud infrastructure and pay only for the structure used. As examples, it can be mentioned IaaS services such as Amazon Elastic Compute Cloud (Amazon EC2), Amazon Simple Storage Service (Amazon S3), Eucalyptus, OpenNebula and OpenStack [4].

Besides the three types of services mentioned above, other authors also consider: CaaS (Communications as a Service), DaaS (Datacenter as a Service), Kaas (Knowledge as a Service) and HAAS (Hardware as a Service) [5].

B. Benefits from Cloud Computing

The main benefit brought with the use of Cloud Computing is scalability. Servers that are not being used represent problems with management and energy consumption. Full load and low load servers use almost the same amount of electricity, so servers which are not being used are not viable. With the resource provisioning provided by the cloud, based on demand, it is easier to scale the system, introducing more resources when they are needed. This allows for reduction in power consumption and management effort, optimizing the use of servers, network and storage space. The economics of clouds involve the following aspects [6]:

- **Economy of scale from the providers view:** it is achieved from big datacenters, minimizing operating costs related to power consumption, personnel, and management. The minimization is a direct result of the assembly of multiple resources in a single domain.
- **Economy of scale from the demand view:** occurs due to the demand aggregation, reducing inefficiencies resulting from load variations, increasing server's load.
- **Economy of scale from the multitenancy view:** since the degree of sharing can be increased, it is possible to reduce the cost of management of servers.

III. Scheduling on the Cloud

One of the most challenging problems in parallel and distributed computing is known as the scheduling problem. The goal of scheduling is to determine an assignment of tasks to processing units in order to optimize certain performance indices [9]. It should be noted that there are two kinds of scheduling within the architecture:

- **Scheduler Manager:** the scheduling algorithms work in order to scale virtual machine instances for computing nodes, responsible for the processing.
- **Scheduler hypervisor within the computing node:** the scheduling algorithm is present in the operating system of the physical machine, sharing the processing of their cases.

The cloud has an infrastructure that includes a scheduler resource. Differently from an operating system that generally works with processes of low granularity, the manager of the cloud works with virtual machine instances, if compared to processes, it can be said they would be of high granularity. Thus, the scheduler of the IaaS cloud application works for allocation of virtual machine instances, which must determine which node allocates this instance.

Differently from a common process scheduling, cloud virtual machine instance remains active, consuming resources or not, until an action (user request or failure hardware/software) interrupts processing. Some items must be evaluated before the cloud scheduler makes its decision on which node should allocate a new request for resources such as:

- Free processing capacity of the node;
- Amount of total memory available;
- Amount of secondary memory available;
- Free ability to read/write secondary memory;
- Free upstream and downstream capacity of the network.

A major problem of scheduling is to determine the cost of a task. The cloud has the same problem of how to determine the cost of processing, disk, memory and a virtual machine instance network before information can be staggered. In such cases it is necessary to use an adaptive scheme, in which the algorithms and parameters used for scheduling decisions dynamically change according to the state of the previous, current and/or future feature [7].

As it can be seen in Figure 1, the grid computing which is, in a way, similar to the one of the cloud, the adaptive scheduling is realized with a focus on resource heterogeneity of candidates, the dynamic performance of resources and the diversity of applications [7]:

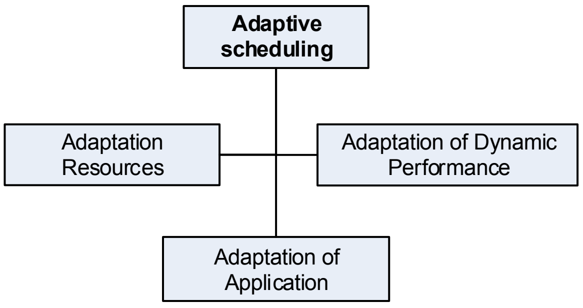


Fig. 1. Taxonomy of scheduling in Grid [7].

According to Casavant [8], an assumption for a good solution in scheduling can be recognized in cases where a metric is available for evaluating a solution, this technique can be used to decrease the time required to find an acceptable solution (schedule). The factors that determine this approach include:

- Availability of a function to evaluate a solution.
- Time needed to evaluate a solution.
- Ability to judge according to some metrics the value of an optimal solution.
- Provision of an intelligence mechanism to limit the space solutions.

As discussed, to a cloud environment, the scheduler needs to evaluate the conditions of computing nodes (approximate or heuristic) before allocating the next virtual machine instance (static scheduling), must select the node with more resources available, whereas it is not possible to measure accurately the amount of resources that need new instance (suboptimal), periodically measure (adaptive) and relocate instances if necessary (load balancing), to not harm the performance of other instances present on the node in question.

A. *Scheduling Algorithm of OpenSource Clouds*

There are several scheduling algorithms used to determine a better balancing of processing and distribution of resources. In open-source clouds, the main algorithms are deterministic, using the scores to determine the node that will be used for processing. This score does not take into account the condition of resources available in the cloud and it often affects its performance, as well as the services delivered to the customers by service providers.

Considering that the current scheduling algorithms of open-source cloud to determine static mode cloud resources, this study aimed to create a dynamic scheduling algorithm to determine which computing nodes within a cloud, have the resources to efficiently host new virtual machine instances.

IV. Methodology and Testing Environment

A cloud was built to have this work developed. As it can be seen from Figure 2, four computers were used, one of them as a manager and the others as computing nodes. The management system OpenStack cloud was used, and it was chosen because it is open-source, belongs to an active community and has extensive documentation.

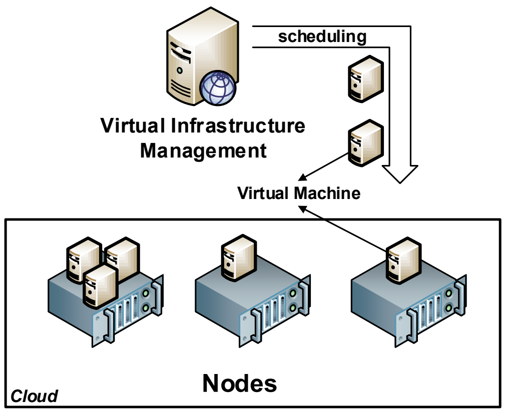


Fig. 2. Cloud infrastructure.

Table 1 describes the initial state of each node, amount of memory, network, CPU, number of cores and cache processor on each physical node.

TABLE I
NODE RESOURCES

Nodes	Network	Memory	CPU Mhz	Cores
01	100 Mbps	4 GB	2000	4
02	100 Mbps	4 GB	2000	4
03	100 Mbps	4 GB	2000	4

Table 2 shows the initial state of the nodes in relation to the resources available before scheduling testing.

TABLE II
AVAILABLE RESOURCES

Nodes	CPU	Network	Memory	Disk
01	99.7 %	100 %	91 %	100 %
02	99.8 %	100 %	91.2 %	100 %
03	99.8 %	100 %	90 %	100 %

For better observation of the results and to provide more detailed comparison in the development phase, the algorithm was tested with two behaviors of virtual machines in an attempt to simulate a real production environment.

- **Virtual machines with constant consumption:** three virtual machine images were created with Ubuntu Server 12.04 operating system that runs in its startup script that provides a different constant consumption of resources (processing, memory, network and disk) for each operating system as in Figure 3.
- **Virtual machines with variable resource consumption:** in this model, the startup script provides a varying consumption with the use of threads that are initiated randomly.

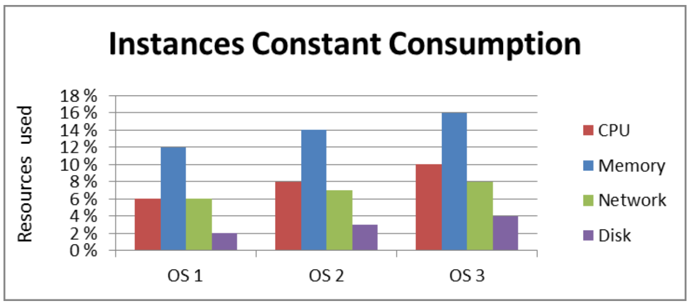


Fig. 3. Instances of constant consumption.

Assuming that virtual machines can adopt two behaviors within a cloud: constant resource consumption and variable resource consumption, an algorithm was developed, as shown in Figure 4, to create this scenario.

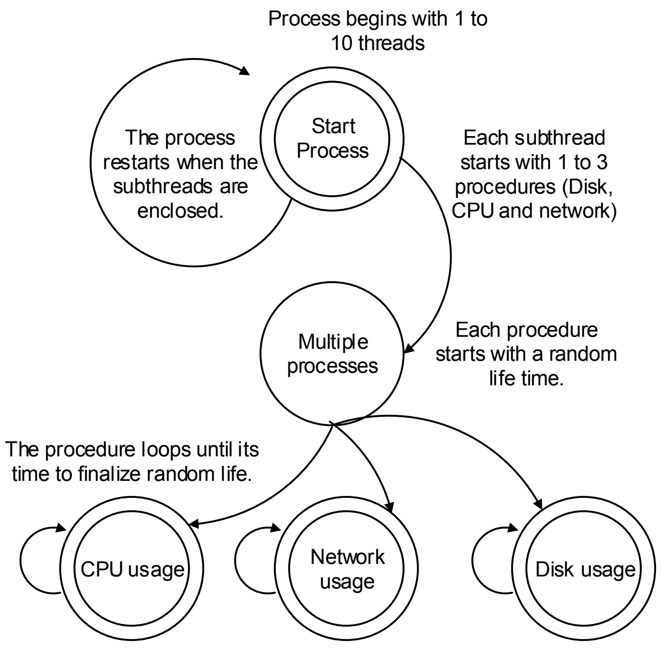

Process begins with 1 to 10 threads
Start Process
The process restarts when the subthreads are enclosed.
Each subthread starts with 1 to 3 procedures (Disk, CPU and network)
Multiple processes
Each procedure starts with a random life time.
The procedure loops until its time to finalize random life.
CPU usage
Network usage
Disk usage

Fig. 4. Simulation consumption script.

Initially the script starts 1-10 threads and each of these threads start 1-3 new threads containing a specific consumption: CPU, disk and network. Note that the memory is not inserted in the script, as when a virtual machine instance is loaded, the KVM removes the portion of memory available and allocates the physical machine to the virtual machine.

V. Results and Discussion

A. *Tests with standard scheduling algorithm of OpenStack*

To start the test, thirty virtual machine instances were launched using a script that selects instances randomly from 1 to 3 with different loads, but constant.

As it can be seen from Figure 5, node 1 has more resources available, but node 2 received 10 instances, overloading it more than the others.

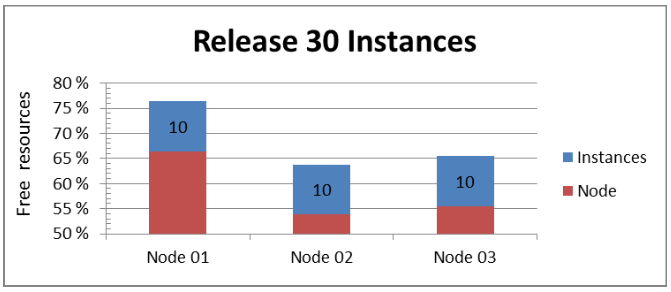


Fig. 5. Test Release: instances of constant consumption.

After that, eight variable instances consumption launches were made and measurements were carried out for the first ten minutes, twenty minutes, thirty minutes and one hour, as it is shown in Figure 6:

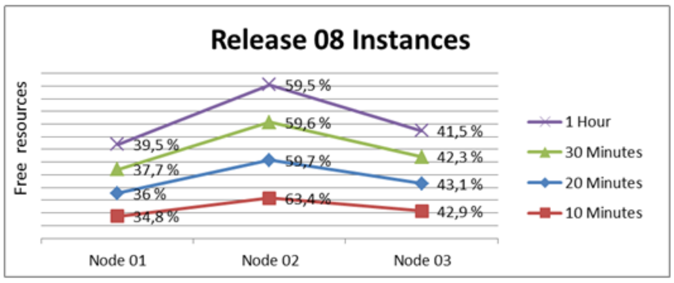


Fig. 6. Test Release: instances of variable consumption.

There were variations in resource consumption, but these changes followed a pattern. This happened because the hypervisor present in the node scheduler has its own system of load balancing, balancing resources among active VMs, making it possible to specify by an average, the quantity of available node resources.

B. *Prototype*

The algorithm prototype consists of two parts:

- The first must constantly evaluate the free resources of each node and save the information in a database.
- The second should select the node with more resources at the time of instantiation of any virtual machine.

Thus, the algorithm should monitor the amount of free resources on each node of the private cloud, should create an index for the manager, which may contain the nodes with more resources available and should also scale the nodes with more resources, as well as new requests of virtual machine instances. It should be noted that, in open-source managers, this selection is done manually by the user.

1) *Node Resources*

A program that monitors, in predefined time intervals, the amount of resources of each node was created. The program is written in Python, to be the same language used by OpenStack and is present in each node of the cloud.

To monitor the amount of available resources on the node, a library of Python, called PSUTIL was used. This library contains functions that evaluate the use of resources such as CPU, memory, network, disk. In the specific case of the disk, to determine its speed, a test is made to read and write to calculate transfer rates.

The algorithm was scheduled to collect machine information and store it in a database. It was scheduled to run every five minutes, and that time was based on *uptime* command, found in Unix and Linux systems, which checks the load processes in these systems. This algorithm has three important functions:

- **resources:** this function is performed only once and through performance analysis, which determines the maximum data transfer capacity of the disk and the network, number of cores in the processor, memory and processing available.
- **check_node:** this function is executed every five minutes, storing on a database the amount of resources available for that node.
- **check_instance:** this function is executed every five minutes, storing on a database the amount of resources consumed by each instance.

2) *Choice Node*

The second algorithm is present in the manager. This should capture all the records present in the database with the information of resources for each node, analyses them and score them. For this, the algorithm evaluates the records of the conditions for the last 24 hours of an active node, taking into consideration that this is only a prototype and future revisions may take into account the behavior parameterization of the nodes using neural networks or other dynamic algorithms.

To determine the node with more features, it is taken into consideration a simple average of the records and weights are applied on resources that may influence the performance of a virtual machine. In this test, higher weights were prioritized for CPU and memory.

Based on the results, the algorithm chooses the node with more resources available before launching the virtual machine instance. As it is shown in Figure 7, the prototype could distribute the load among the nodes participating in the network more evenly than the current algorithm of the OpenStack cloud.

So, it is extremely necessary to receive feedback from each participant from the cloud, demonstrating their real capacity of available resources. For instances of constant consumption, the prototype achieved its goal when it distributed the resources in an equated way among the instances.

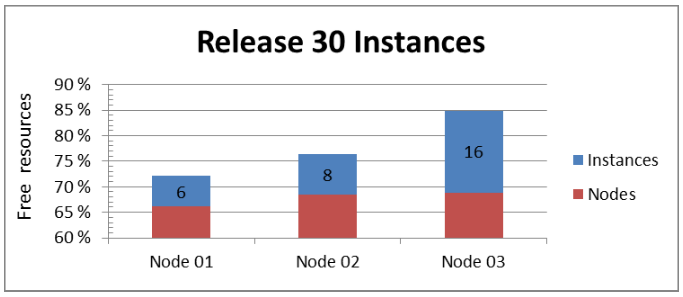


Fig. 7. Results of instances of constant consumption.

As one of the approaches to the use of Cloud Computing is the reduction in energy consumption, the prototype could also be used to allocate the maximum possible instances on a single node, allowing the manager to hibernate nodes not used, without compromising the performance of the virtualized operating system.

As in the constant consumption, the prototype worked with instances of variable consumption (Figure 8), because it evaluates the condition of the nodes before scheduling application for VM. In the case of instances of variable consumption, future improvements of this algorithm could migrate instances of overloaded nodes and monitor these nodes after migration.

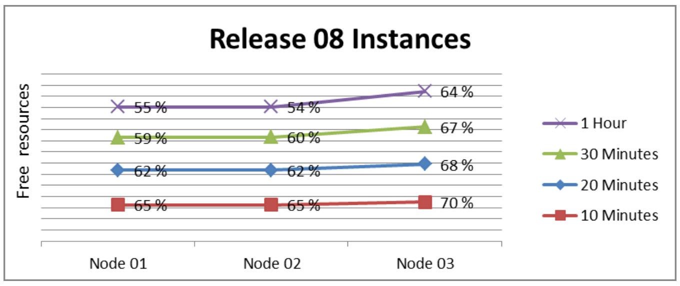


Fig. 8. Results of instances of variable consumption.

In open-source clouds, the main algorithms are deterministic, using the scores to determine the node that will be used for processing. This score does not take into account the condition of resources available in the cloud and it can hinder the performance of the architecture and affect the services delivered to the customers by service providers.

The results in this paper show that instances that behave like processes in the operational system, such as the KVM hypervisor, allow the analysis and record of consumed resources, as well as the calculation of the amount of resources available for each node of the cloud. With this information, it is possible to determine at least two policies for scheduling requests for virtual machine instances that justify the use of Cloud Computing:

- To distribute the load among the nodes of the cloud, thereby improving the quality of service provided;
- To allocate the maximum instances for a node until its exhaustion of resources by turning off unused nodes.

Therefore, this prototype has shown that feedback processes that perform the VM instances is essential to determine with some precision the resource capacity of each node participating in the cloud, making it possible for a manager to decide which policy for operation of the architecture will be used, to be chosen among quality of service and/or energy savings:

- Quality of service: to ensure the availability of the service to a client, allocating the resources between nodes without overloading a particular specific node;
- Energy savings: it makes possible the large data centers, to monitor and to allocate the correct amount of instances per node, disabling nodes that are inactive, reducing energy consumption and the amount of carbon dioxide released into the atmosphere.

Thus, it is concluded that Cloud Computing provides great benefits, among the major energy consumption, physical space savings in data centers, easy sizing provisions, APIs for external interface, among others. However, determining how resources will be provisioned into the cloud is of utmost importance to ensure its success and adoption by large companies.

References

[1] BHADAURIA, Rohit; CHAKI, Rituparna. A Survey on Security Issues in Cloud Computing. CORR, P. abs/1109.5388, 2011.

[2] SABAHI, F. Cloud computing security threats and responses. Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on May 2011.

[3] BUYYA, Rajkumar; BROBERG, James; GOSCISNSKI, Andrzej M. Cloud Computing: Principles and Paradigms. John Wiley and Sons: San Francisco, 2011.

[4] SASIKALA, P. Cloud computing: present status and future implications. Available at: <http://www.inderscience.com/storage/f123101246118579.pdf>. Last access: 29 apr. 2013.

[5] HE, Zhonglin; HE, Yuhua. Analysis on the security of cloud computing. Proc. Spie, Qingdao, China, n., p.7752-775204, 2011.

[6] BACHIEGA, Naylor G.; et al. Open Source Cloud Computing: Characteristics and an Overview. Available at: <http://world-comp.org/p2013/PDP3537.pdf>. Last access: 10 jan. 2014.

[7] DONG, F.; AKL, S. G. Scheduling Algorithms for Grid Computing: State of the Art and Open Problems. Kingston, Ontario, Canada, Janeiro 2006.

[8] CASAVANT, Thomas L.; KUHL, Jon G. A taxonomy of scheduling in general-purpose distributed computing systems. IEEE Transactions on Software Engineering, New York, v. 14 n. 2, p.141-154, Feb. 1988.

Performance Analysis of Initialization Methods for Optimizing Artificial Bee Colony Grid Scheduling

T.Vigneswari[1], **Dr.M.A.Maluk Mohamed** [2]
1CSE Dept.Kings College of Engineering, Research scholar- System Software group, India
[2] Research Guide System Software group &Principal M. A. M College of Engineering, India

Abstract *Efficient scheduling is critical to achieve high performance in grid computing. Scheduling issues are NP-complete and were extensively studied with various algorithms being proposed in literature. To improve application performance using grid environment, resource use and job scheduling must be efficient. This work proposes Artificial Bee colony (ABC) algorithm to locate an optimum schedule for dynamic jobs arrival. The performance of the proposed methods is evaluated based on make span. In this work three initialization techniques namely random initialization, orthogonal initialization and chaotic initialization is studied and the output presented. Simulations were performed with 25 jobs, and resources are grouped into 5 clusters*

Keywords: Grid Scheduling, Optimization, Artificial Bee Colony (ABC), Initialization.

1 Introduction

Task scheduling algorithms are either Heuristic based or Guided random search based. The former is divided into [2]: list scheduling, clustering and task duplication. List scheduling Heuristics maintains priority based tasks list with 2 phases: Task prioritizing – to prioritize each task according to criteria and processor selection- to select a suitable processor to minimize a predefined cost function. Examples of heterogeneous system algorithm are Heterogeneous Earliest Finish Time (HEFT) and Critical-Path-On-a-Processor (CPOP). Clustering Heuristics map tasks in a graph to unlimited clusters. At every step, selected tasks to be clustered can be any task, and not necessarily a ready task. Each iteration refines previous clustering by merging clusters. When two tasks are assigned to same cluster, they are executed on same processor. Task duplication Heuristics schedules a task graph by mapping some tasks redundantly, reducing inter-process communication overhead. These algorithms performance is dependent on heuristics effectiveness. Hence, they are unlikely to ensure consistent results on a range of problems [3], specially when DAG task scheduling problem complexity increases.

Guided random search based algorithms: These use random choice to guide themselves through a problem. Guided random search based algorithm have robust performance on various scheduling problems, but are less efficient, generating higher computational cost than heuristic based algorithms. Genetic algorithms (GA) evolve solutions for task scheduling problems. Some examples are Tabu search (TS), Particle Swarm Optimization (PSO), Simulated Annealing (SA) and Ant Colony System (ACS) [4].Heuristic algorithms guarantee locating near optimal solution in lower polynomial time. Heuristic based list scheduling algorithms include Heterogeneous EarliestFinish Time (HEFT) and Critical-Path-On-a-Processor (CPOP) while Guided random search based scheduling algorithms have robust performance on scheduling issues.

Regardless of the EA method under consideration, one stage is always common: population initialization. Population-based optimization algorithms rely on initial population of potential solutions. Conventionally, basic random number generators (RNGs) initialize population. For years, researchers were not interested in the influence of population initialization on Evolutionary Algorithms (EA) performance. Recent studies suggest the possibility of significantly improving EAs performance using different initialization methods [5]. Studies show that improving initial population increases probability of finding optimum solutions, decreases computational cost [6], reduces variance of results [7] and improves EAs solution quality [8]

Initialization procedures are stochastic or deterministic methods. Generally the aim of stochastic methods is producing random numbers [9]. From a designers' point of view, more random sequence (and less predictable and reproducible set) ensures better initial population for stochastic algorithms like EAs. Thus, stochastic methods most important attribute is cycle time (period length) and degree of unpredictability. Though these methods do not produce "true random" numbers, they are EAs most used initialization methods. Recently, stochastic methods subcategory called Chaotic number generators attracted interest [9]. Chaotic methods mimic behavior of dynamical systems yielding unpredictable point sequences [10].

In contrast to first group, Deterministic methods focus on uniformity rather than randomness [7]. Researchers believe a more uniform initial population enhances EA's exploration ability in early iterations [6] in the absence of prior knowledge about a problem. In other words, increasing initial population uniformity reduces probability of missing large part of search space saving much computational budget. Orthogonality is a deterministic method.

This study aims to minimize overall job completion time or application makespan. Makespan represents time lapsedfrom start of first task to end of last scheduled task. This study proposes to investigate relationship between initial population and final outcome of ABC for grid scheduling with optimal makespan.

2 Related works

Genetic Algorithms (GA) are widely used for optimization and it is commonly used for grid scheduling too. Gao et al [11] proposed 2 models to predict job completion time in a service grid. Two algorithms using predictive models were proposed to schedule jobs at both system and application levels. GA minimizes average job completion time through optimal job allocation on every node in application level scheduling. Experiment showed that scheduling system using adaptive scheduling algorithms efficiently allocated service jobs. A new Space Time GA (STGA) based on faster searching and protected chromosome formation was proposed by Song, et al [12]. Jobs were subjected to system failures caused by infected hardware, software problems or lack of proper security policies, in an open-resource Grid system's parallel execution. The process models risk and unsecured conditions in Grid job scheduling resulting in 3 risk-resilient strategies being developed. A Reliable Job Scheduler using Resource Fault Occurrence History (RFOH) in Grid Computing was proposed by Khanli et al [13].To improve grid scheduling reliability, the strategy used Resource Fault Occurrence History (RFOH) details. RFOH stored faults and number of jobs in execution using this resource. FOHT was updated and either resource was unable to complete the job within deadline, or a new job was assigned to resource. Based on RFOH information GA was used to find an optimum schedule. A two level scheduling system, with first level being formed by a computing nodes set – each having a local scheduling policy – and a second level of super scheduler was proposed by Martino [14]. Local scheduler accepts one job at a time allocating it on local hardware regarding current (local) information. GRID jobs allocation simulation results were presented. Search strategy for input case did not converge to optimal case inside limited trials undertaken compared to earlier work on 24 jobs. GA benefits include improving scheduling quality which is discussed. GGAS modular structure allows expansion of its functionalities to include other first level scheduling policy regarding FCFS being considered. This paper proposes local search strategy to improve convergence when jobs to be considered are big as in real world operations.

To improve the efficiency of the GAs it was hybridized with other optimization techniques as follows: Adaptive job scheduling for Computational Grid based on Ant Colony Optimization with Genetic Parameter Selection [15] was presented by Mandloi and Gupta where GA was combined with ACO algorithm for efficient grid environment task scheduling. Jobs arrival, resource request and availability entries were simulated, and resource brokers designed. Resources were clustered into many groups, and ACO algorithm used to find optimal schedule. GA controlled ACO algorithm parameters. Job execution failure and job completion were taken to evaluate performance of new scheduling algorithm. Results proved that optimization algorithm using GA and PSO were better than FCFS algorithm. presented A Comparative Study of GA and ABC for Job Scheduling [16] was presented by Selvi and Umarani. For efficient grid environment job scheduling, functions similar to Genetic operations and bee behavior are combined. Population was initialized by processors and jobs number in GA based job scheduling. During selection, best individuals were selected. In cross over, two different tasks were selected, and assignment of machines exchanged randomly. For this, exchange point was needed. During mutation, a task was selected randomly and reassigned to new processor. Steps from selection to mutation were repeated until make span was less than pre-specified value. To select best source positions in cross over and mutation, ABC optimization algorithm was used. Simulations were undertaken with five different job sets. Results were compared with GA based job scheduling and ABC job scheduling. Numerical results showed that hybrid GA-ABC job scheduling gave high accuracy, efficiency with minimum job completion time when comparing to GA and ABC scheduling algorithms.Ant Colony Optimization (ACO) is an EA which is popularly used for scheduling. To offset new grid environment job scheduling challenges, Fidanova et al [17] presented a heuristic scheduling algorithm to achieve high throughput computing in a grid environment. This is a NP-problem requiring exponential time to solve. Hence heuristic algorithm aimed to achieve solutions in reasonable time. This paper discusses ACO method based heuristic algorithm and formulates strategies for grid scheduling. The algorithm guarantees load balancing. Chen et al [18] proposed an ACO algorithm to schedule large-scale work-flows with various QoS parameters to enable users to specify QoS preferences and define minimum QoS thresholds for applications. The algorithm's objective is finding a solution meeting all QoS constraints and optimizing user-preferred QoS parameter., Seven new heuristics for ACO approach were designed based on workflow scheduling characteristics and an adaptive scheme allowing artificial ants to select pheromone value based heuristics was suggested. Experiments on ten workflow applications with 120 tasks demonstrated the new algorithm's effectiveness.

For scheduling the tasks in the grid environment, Particle Swarm Optimization (PSO) algorithm is used. The design/implementation of Hierarchical Discrete Particle Swarm Optimization (H-DPSO) for grid environment's dependent task scheduling was proposed by Garg and Singh [19]. In HDPSO, particles are dynamic and hierarchically arranged with good particles lying above and influencing the swarm. The problem's bi-objective version is minimizing makespan and total cost simultaneously when optimization criteria are considered. The H-DPSO based scheduler was evaluated via application task graphs. Simulation analysis showed that H-DPSO based scheduling for grid computing is viable and effective. Tasks Scheduling in Computational Grid using a Hybrid Discrete Particle Swarm Optimization (HDPSO) [20] was presented by Karimi and Motameni. Particles were initialized by Min-Min algorithm. For every job, a set of minimum completion time on all available processors was found. Then for every task, processor which ensured minimum expected completion time was selected. Simulations were conducted with specific jobs, and processing time for each processor was pre-assumed. Makespan and throughput evaluated scheduling performance. Experiments were conducted 10 times using varied random values. Results were compared with algorithms like Min-Min and Max-Min algorithms. The comparison revealed that HDPSO algorithm ensured minimum makespan and maximum throughput. Hybrid PSO Algorithm (HPSOA) to resolve dynamic Web services selection with QoS global optimal in grid workflow was presented by Hu,et al [21]. The algorithm's essence was that dynamic Web Service selection problem with QoS global optimal was transformed into a multi-objective services composition optimization with QoS constraints. Crossover and mutation in GA were brought to PSO algorithm to form a mix algorithm called HPSOA to solve QoS global optimal problem. Theoretical analysis and experiments indicate the algorithm's feasibility and efficiency.

Some of the other optimization methods used for optimizing the scheduling in grid computing are as follows: Kim et al [22] developed a binary artificial bee colony (BABC) algorithm for grid computing binary integer job scheduling problems. A binary artificial bee colony extension of BABC incorporating a flexible ranking strategy (FRS) to up balance between exploration and exploitation was proposed. FRS generates and uses new solutions for diversified search in early generations speeding up convergence in latter generations. Two variants were introduced to minimize makespan. A specific number of best solutions are used with FRS initially , while in the second run, best solutions number is reduced with every new generation. Simulation for benchmark job scheduling issues reveals that new method's performance is better than alternatives like simulated annealing, GA, and PSO. A grid computing tasks scheduling algorithm based on simulated annealing (SA) was introduced by Fidanova et al [23]. SA is initial solution generation creating a set of neighbours. Tasks are collected in a set and scheduled. Sop task arrival time is unimportant. Tasks scheduling is on the same machine and form a local queue related to the machine. Tasks queue is sent onto the machine when running tasks from previous queue. So send time does not influence makespan time. When the grid has a cluster like part, it is considered one machine, and hence the issue is converted to a sequential tasks scheduling problem. GLOA- A new Job Scheduling Algorithm for Grid Computing [24] was presented by Pooranian et al. Group Leader Optimization Algorithm (GLOA) was inspired by leaders role in social groups. Problem space was separated into many small parts and each processed separately to locate an optimal solution in parallel. Convergence in a short time is important in optimization problems. GLOA finds an optimum schedule for jobs arrival with available resources and reached optimal solution quickly. Makespan evaluated grid scheduling performance. Results showed that makespan was very small compared to other scheduling algorithms.

Some works available in literature are reviewed in this section. Literature proposes many GA based heuristics and swarm based optimizations for grid scheduling. Algorithms try to overcome the problem by fitness function changes. Hybridization among different meta-heuristics was effective for problems and outperformed single methods. However, it is seen that none of the works consider the quality of the initial population chosen for the optimization problem. The research goal of this study is to investigate relationship between initial population and final outcome of ABC.

3 Methodology

In this study, Random initialization, chaotic initialization method based on stochastic methods and orthogonal initialization method based on deterministic method is investigated. Figure 1 shows the flowchart for the ABC algorithm. The steps are explained in the following sections.

3.1 Artificial Bee Colony (ABC) Optimization

Karaboga proposed ABC in 2005 where those involved share food sources information with each other. ABC algorithm is advantages due to its robustness, fast convergence, high flexibility and fewer control parameters [25]. Some tasks are performed by specialized individuals (bees) in a real colony optimization and they maximize nectar in the hive. ABC algorithm adopts three kinds of bees like employed bees, onlooker bees, and scout bees. Half the colony includes employed bees and the other half includes onlooker bees. Bees functions are

1. Employed bees exploit nectar sources explored before providing information to other onlooker bees in the hive about food source site quality which they exploit.
2. Onlooker bees wait in the hive and decide the food source to exploit based on information shared by employed bees [26].
3. Scouts search environment randomly to locate a new food source based on an internal motivation or possible external clues or randomly.

To simulate these behaviors, some ABC algorithm steps are given below:

1. Initialize food source position.
2. Each employed bee produces new food source in the site exploiting the better source.
3. Every onlooker bee selects source based on solution quality, produces a new food source in chosen food source site exploiting the better source.
4. To determine source to be abandoned and allocate employed bee as scout to search for new food sources.
5. To memorize best food source found.
6. Repeat steps 2-5 till stopping criterion is met.

First algorithm randomly produces $\vec{x}_i (i = 1, ...SN)$ solutions in parameters where SN is number of food sources. Second, for every employed bee, when its total equals half the food sources, a new source is produced by the equation (1):

$$v_{i,j} = x_{ij} + \varphi_{ij}(x_{ij} - x_{kj}) \qquad (1)$$

where φ_{ij} is a uniformly distributed real random number in the range [-1,1], k is index of randomly chosen solution from colony (k = int(rand ∗SN) + 1), j = 1, . . .,D and D is the dimension of problem. After producing $\vec{v}_i$, the new solution is compared to $\vec{x}_i$ solution and employed bee exploit better source. Third, an onlooker bee selects a food source with probability (equation 2) producing a new source in chosen site by equation (1). For employed bee, the better source is exploited.

$$p_i = \frac{fit_i}{\sum_{j=1}^{SN} fit_j} \qquad (2)$$

where fit_i is fitness of solution $\vec{x}_i$. After all onlookers are distributed to sources, they are checked to ensure their abandonment. If the cycles a source cannot improve is greater than a predetermined limit, then source is exhausted. The employed bee is associated with exhausted source and becomes a scout making random search in problem domain by equation (3).

$$x_{ij} = x_j^{\min} + (x_j^{\max} - x_j^{\min}) * rand \qquad (3)$$

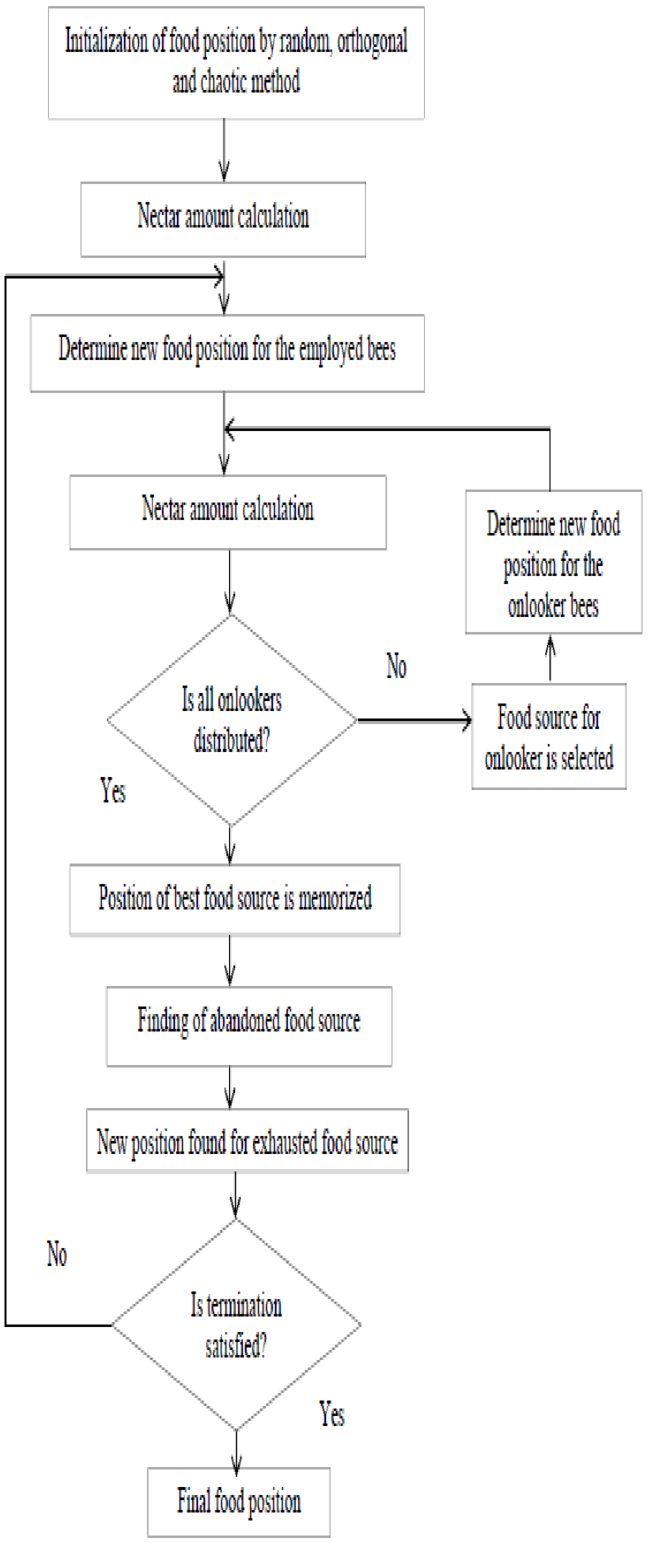

Initialization of food position by random, orthogonal and chaotic method
Nectar amount calculation
Determine new food position for the employed bees
Nectar amount calculation
Determine new food position for the onlooker bees
Is all onlookers distributed?
No
Food source for onlooker is selected
Yes
Position of best food source is memorized
Finding of abandoned food source
New position found for exhausted food source
No
Is termination satisfied?
Yes
Final food position

Fig 1 Flowchart for Artificial Bee Colony (ABC) Optimization

Usually initializations of bee's food position are done randomly. But in this study, it is proposed that the initialization of food source is performed using random, orthogonal and chaotic methods.

Random initialization

Random initialization attempts to generate statistically uniform random numbers in a given range. In reality, RNGs cannot produce uniform distribution of points [27]. This shortcoming is worse when search space dimensionality grows or number of points diminishes.

Orthogonal Initialization

An orthogonal array states limited combinations are scattered uniformly all over space for all combinations. So orthogonal design is a potential method to generate a good initial population. Also Orthogonal Design (OD) was applied to produce evenly scattered points over search space [28]. OD produces an orthogonal 2D array like $L_M(Q^N)$ where M is population size, N is number of factors, Q levels and L denotes Latin square. Q is an even integer and $M = Q^J$ while J is a positive integer that satisfies $N = \frac{Q^J - 1}{Q - 1}$. The attributes of orthogonal array are:

1. For factor present in a column, level occurs exactly $\frac{M}{Q}$ times. This attribute makes resulting population uniform.
2. Array orthogonality is not sensitive to order of columns. Hence, reorder is performed on columns or removed as the resulting array is orthogonal [29].

It is defined as x_i to be ith factor, and they continuous, but orthogonal design is applicable to discrete factors alone. To overcome this, every factor is quantified into a finite number of values. Specifically, domain $[l_i, u_i]$ is quantized into Q_1 levels and $\alpha_{i,1}, \alpha_{i,2}, ...\alpha_{i,Q}$ levels, where design parameter Q_1 is odd and $\alpha_{i,j}$ is given by

$$\alpha_{i,j} = \begin{cases} l_i & \text{j=1} \\ l_i + (j-1)\left(\frac{u_i - l_i}{Q_1 - 1}\right) & 2 \le \text{j} \le Q_1 - 1 \\ u_i & \text{j=}Q_1 \end{cases} \tag{4}$$

Chaotic Initialization

Chaotic motion traverses all states by regularity in all state and so chaotic initialization methods ensure better distribution in search space due to chaos's randomness and non-repetitive ergodicity [30].

$$x_{i,j}^{(k+1)} = \mu(1 - 2\,|\,x_{i,j}^{(k)} - 0.5\,|),\ 0 \le x_{i,j}^{(0)} \le 1 \tag{5}$$

Where $x_{i,j}^{(k)}$ is jth variable of ith individual in kth iteration and μ is bifurcation factor.Equation (5) shows chaotic methods being deterministic. But, resulting chaotic sequences are sensitive to initial condition and outputs are unpredictable.

Chaos is the well-known logistic mapping defined by

$$z^{j+1} = \mu z^j (1 - z^j),\ \ z^j \in [0,1]\ \text{j=1,2...} \tag{6}$$

where z^j is value of variable z at jth iteration, and l is chaotic attractor. The chaos system includes special characteristics like ergodicity, randomicity and extreme sensitivity to initial conditions.

This solves the following continuous parameter optimization problem:

$$\text{(P)=}\begin{cases} \max\ \ \text{f}(\text{X}_{1,}.....\text{X}_r) \\ s.t \quad \text{X}_i \in [a_i, b_i],\ \ \text{i=1,2....,r} \end{cases} \tag{7}$$

where $[a_i, b_i] \subset R$; f is real-valued continuous function; r the number of optimization variables.

r chaotic variables are generated by Logistic mappings:

$$z_i^{j+1} = \mu_i z_i^{\,j}(1 - z_i^{\,j}),\ \ \text{i=1,2,....r, j=1,2...} \tag{8}$$

where i is serial number of chaotic variables, and μ_i = 4. Let j = 0, and given the r chaotic variables different initial values $z_i^0 = (i = 1, 2, ...r)$, then values of r chaotic variables $z_i^1 = (i = 1, 2, ...r)$ are produced by Logistic equation and encoded into a real-coded antibody. Let j = 1,2,. . . ,N-1, and then other N-1 antibodies are produced by same method [31]

.

4 RESULTS AND DISCUSSION

Simulations are conducted with 25 jobs, and the resources are grouped into 5 clusters. Three initializations such as random, orthogonal and chaotic are performed. The proposed scheduling algorithm is executed for 3 times. During every run Makespan value is calculated. Totally 100 iterations are performed. Figure shows the Average Makespan.

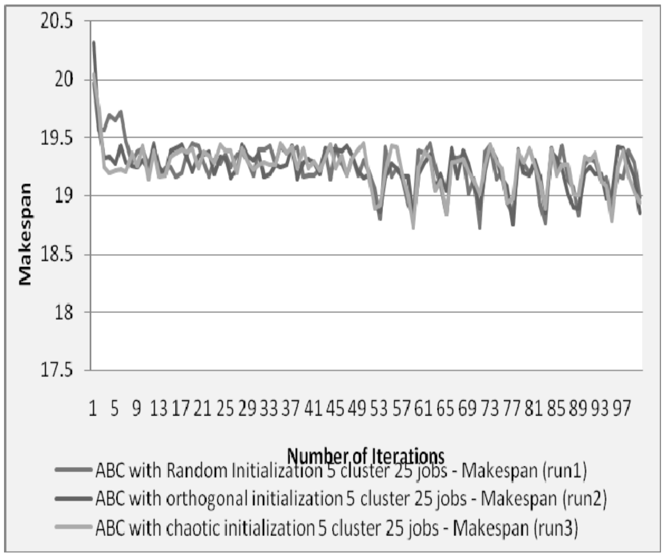


Fig 2 . Convergence with different initialization scheme

It is observed from figure 2 that Average Makespan of 3 runs is achieved in the simulation with varying number of iterations. The average Makespan of ABC with orthogonal initialization with 5 clusters, 25 jobs performs better.

5. CONCLUSION

A Grid is meant for large scale distributed and parallel computing systems where each node shares resources dynamically during application execution. Usually resources are heterogeneous and distributed geographically. Resource selection depends on applications availability, cost and Quality of Service (QoS) requirement. Jobs assignment to resources should be optimal to reduce makespan, minimize allocated resources cost and maximize throughput. It is difficult to locate optimal resource allocation for specific jobs that lower jobs schedule length. This work proposes an ABC algorithm to locate an optimum schedule for dynamic jobs arrival. Here, three initialization techniques ie; random initialization, orthogonal initialization and chaotic initialization are studied and output presented. Simulation reveals that Average Makespan of 3 runs is achieved with varied iterations. ABC average makespan with orthogonal initialization with 5 clusters, 25 jobs performs better than random initialization and chaotic initialization

.

References

[1] Lorpunmanee, S., Sap, M. N., Abdullah, A. H., &Chompoo-inwai, C. (2007). An ant colony optimization for dynamic job scheduling in grid environment.International Journal of Computer and Information Science and Engineering, 1(4), 207-214

[2] TarekHagras, Jan Janecek, "A Near Lower-Bound Complexity Algorithm for Compile-Time Task-Scheduling in Heterogeneous Computing Systems", Proceedings of the ISPDC/HeteroPar'04 IEEE, 2004

[3] YumingXu, Kenli Li, Tung Truong Khac, MeikangQiu, "A Multiple Priority Queueing Genetic Algorithm for Task Scheduling on Heterogeneous Computing Systems", IEEE 14th International Conference on High Performance Computing and Communications, pp. 639-646, 2012

[4] Manudhane, K. A., &Wadhe, A. (2013). Comparative Study of Static Task Scheduling Algorithms for Heterogeneous Systems. International Journal on Computer Science & Engineering, 5(3).

[5] M. Clerc, "Initialisations for particle swarm optimisation," Tech. Rep., 2008.

[6] S. Kimura and K. Matsumura, "Genetic algorithms using lowdiscrepancysequences," in Proceedings of the 2005 conference onGenetic and evolutionary computation. ACM, 2005, pp. 1341–1346.

[7] R. Morrison, "Dispersion-based population initialization," in Geneticand Evolutionary ComputationGECCO 2003. Springer, 2003, pp. 204–204.

[8] Z. Ma and G. A. Vandenbosch, "Impact of random number generatorson the performance of particle swarm optimization in antenna design,"in Antennas and Propagation (EUCAP), 2012 6th European Conferenceon. IEEE, 2012, pp. 925–929.

[9] C. Yanguang, M. Zhang, and C. Hao, "A hybrid chaotic quantum evolutionary algorithm," in Intelligent Computing and Intelligent Systems (ICIS), 2010 IEEE International Conference on, vol. 2. IEEE, 2010, pp. 771–776.

[10] N. Dong, C.-H. Wu, W.-H. Ip, Z.-Q. Chen, C.-Y. Chan, and K.-L. Yung, "An opposition-based chaotic ga/pso hybrid algorithm and its application in circle detection," Computers & Mathematics with Applications, 2012.

[11] Gao, Y., Rong, H., & Huang, J. Z. (2005). Adaptive grid job scheduling with genetic algorithms. Future Generation Computer Systems, 21(1), 151-161.

[12] Song, S., Hwang, K., & Kwok, Y. K. (2006). Risk-resilient heuristics and genetic algorithms for security-assured grid

job scheduling. Computers, IEEE Transactions on, 55(6), 703-719.

[13] Leyli Mohammad Khanli, Maryam Etminan Far and Ali Ghaffari presented " Reliable Job Scheduler using RFOH in Grid Computing ",Journal of Emerging Trends in Computing and Information Sciences, VOL. 1, NO. 1, JULY 2010.

[14] Di Martino, V., &Mililotti, M. (2004). Sub optimal scheduling in a grid using genetic algorithms. Parallel computing, 30(5), 553-565.

[15] MandloiSaurabh and Hitesh Gupta, " Adaptive job Scheduling for Computational Grid based on Ant Colony Optimization with Genetic Parameter Selection ", International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970) Volume-3 Number-1 Issue-9 March-2013.

[16] V.Selvi and R.Umarani, "Comparative Study of GA and ABC for Job Scheduling ", International Journal of Soft Computing and Engineering (IJSCE), ISSN: 2231-2307, Volume-2, Issue-6, January 2013.

[17] Fidanova, S., &Durchova, M. (2006). Ant algorithm for grid scheduling problem. In Large-Scale Scientific Computing (pp. 405-412). Springer Berlin Heidelberg.

[18] Chen, W. N., & Zhang, J. (2009). An ant colony optimization approach to a grid workflow scheduling problem with various QoS requirements. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 39(1), 29-43.

[19] Garg, R., & Singh, A. K. (2013). Enhancing the Discrete Particle Swarm Optimization based Workflow Grid Scheduling using Hierarchical Structure. International Journal of Computer Network and Information Security (IJCNIS), 5(6), 18.

[20] Maryam Karimi and HomayoonMotameni, " Tasks Scheduling in Computational Grid using a Hybrid Discrete Particle Swarm Optimization " , International Journal of Grid and Distributed Computing Vol. 6, No. 2, April, 2013.

[21] Hu, C., Wu, M., Liu, G., &Xie, W. (2007, August). QoS scheduling algorithm based on hybrid particle swarm optimization strategy for grid workflow. In Grid and Cooperative Computing, 2007. GCC 2007. Sixth International Conference on (pp. 330-337). IEEE.

[22] Kim, S. S., Byeon, J. H., Liu, H., Abraham, A., &McLoone, S. (2012). Optimal job scheduling in grid computing using efficient binary artificial bee colony optimization. Soft Computing, 1-16.

[23] Fidanova, S. (2006, October). Simulated annealing for grid scheduling problem. In Modern Computing, 2006. JVA'06. IEEE John Vincent Atanasoff 2006 International Symposium on (pp. 41-45). IEEE.

[24] Zahra Pooranian, Mohammad Shojafar, Jemal H. Abawajy, and MukeshSinghal ," GLOA- A new Job Scheduling Algorithm for Grid Computing " , International Journal of Artificial Intelligence and Interactive Multimedia, Vol. 2, Nº 1, 2013.

[25] Yan, G., & Li, C. (2011). An effective refinement artificial bee colony optimization algorithm based on chaotic search and application for pid control tuning. Journal of Computational Information Systems, 7(9), 3309-3316.

[26] Karaboga, D., &Akay, B. (2009). Artificial bee colony (ABC), harmony search and bees algorithms on numerical optimization. In Innovative Production Machines and Systems Virtual Conference.

[27] H. Maaranen, K. Miettinen, and M. M. M¨akel¨a, "Quasi-random initial population for genetic algorithms," Computers & Mathematics with Applications, vol. 47, no. 12, pp. 1885–1895, 2004.

[28] Peng, L., Wang, Y., Dai, G., & Cao, Z. (2012). A Novel Differential Evolution with Uniform Design for Continuous Global Optimization. Journal of Computers, 7(1).

[29] Kazimipour, B., Li, X., & Qin, A. K. (2013, June). Initialization methods for large scale global optimization. In Evolutionary Computation (CEC), 2013 IEEE Congress on (pp. 2750-2757).

[30] Dong, N., Wu, C. H., Ip, W. H., Chen, Z. Q., Chan, C. Y., & Yung, K. L. (2012). An opposition-based chaotic GA/PSO hybrid algorithm and its application in circle detection. Computers & Mathematics with Applications, 64(6), 1886-1902.

[31] Zuo, X. Q., & Fan, Y. S. (2006). A chaos search immune algorithm with its application to neuro-fuzzy controller design. Chaos, Solitons& Fractals, 30(1), 94-109.

Framework to detect and repair distributed intrusions based on mobile agent in hybrid cloud

Abir KHALDI[1], Kamel KAROUI[1], Henda BEN GHEZALA[1]
[1] RIADI Laboratory ENSI, University of Manouba, Manouba, Tunisia

Abstract- *Cloud computing is an emerging paradigm based on distributed services. It is deployed in virtual resources to provide services to public customers and private organizations. Generally, without security measures, distributed cloud services are vulnerable. In this paper, we will propose a framework for detecting and repairing distributed intrusions in hybrid cloud. Our framework is based on secure mobile agents.*

Keywords: Cloud computing, security, IDS, Mobile Agent.

1. Introduction

Cloud providers offer the customers' services requirements. There are some security issues associated with cloud services. These issues fall into two broad categories: Security issues faced by cloud providers and security issues faced by customers. In most cases, the providers must ensure their infrastructure security and their clients' data integrity while the customer must ensure that the provider has taken the proper security measures to protect his information.

Because of its distributed nature, cloud computing environments are easy targets for intruders looking for exploring possible vulnerabilities. The first defense line to face attackers is to deploy a firewall to filter unauthorized access then an IDS (Intrusion detection system) in order to detect coming attacks.

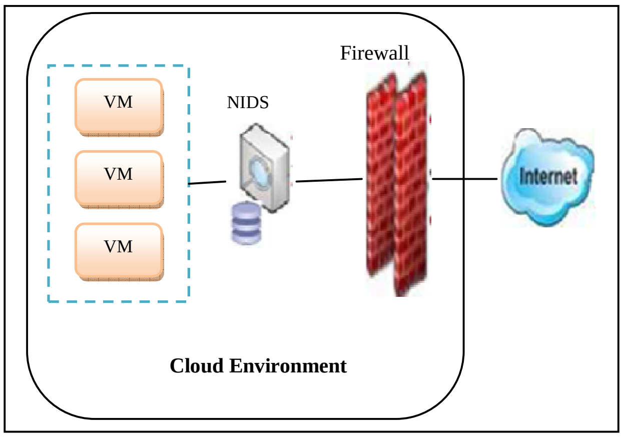


Figure 1. Firewall and NIDS in the Cloud architecture

In figure 1, we have an NIDS (Network IDS) to monitor all cloud network traffics. When an attack occurs, NIDS alerts cloud administrator.

In [1], we proposed a secure cloud architecture based on an NIDS as a second line of defense after the firewall. The NIDS performance is really approved for detecting attacks.

But, attacks can be distributed between cloud nodes and be hidden for the NIDS. So to detect them, we will propose a framework implementing :

- A HIDS (Host IDS) in every virtual machine (VM)
- An intelligent process to correlate between HIDS alerts.
- Secure Agents to execute the correlating process

We will focus on deploying this framework on hybrid cloud environment to:

- Phase 1 : Detect distributed attacks
- Phase 2 : Evaluate the attacks risks
- Phase 3 : Repair attacks

In this paper, we will propose a framework based on secure mobile agents to detect distributed intrusions and repair the vulnerabilities in hybrid cloud. The repairing phase consists on adding a new security policy in the firewall.

The reminder of this paper is organized as follows. The section 2 discusses some related works in the area of mobile agent based IDS. In the next sections, we will describe our proposed framework using mobile agents to detect and repair intrusions. Then we will explain implementation prototype in section 5 to evaluate results in section 6. Finally, we will give conclusions in section 7.

2. Related work

The IDS is based on two simple components architecture: collection component and analyzer component. While this architecture is effective just for small collections of monitored hosts. In fact, centralized analysis limits the ability to scale up to handle larger collections. Therefore, Mobile Agent-based intrusion detection system, such as Autonomous Agents for Intrusion Detection (AAFID) [2], follows a hierarchical structure. So, if any part of the internal nodes is disabled, the functioning of that branch of IDS will be disqualified.

In addition, those architectures are not flexible, not completely distributed and are not able to respond to attacks against intrusion detection system itself.

The IDS performance using mobile agents is considerably important to reduce the network load. In this case, agents communications should be secured. This issue, which has been neglected by most of related works, will be one of the main concern when we design our framework . It will be based on secure mobile agents to detect distributed intrusions in hybrid cloud . Table I illustrates a comparative study on related works.

Table 1. Comparing properties of previous related work

RELATED WORK	*ARCHITECTURE*	NETWORK LOAD	SCALABILITY	*RESISTIBILITY*	*AGENT SECURITY*	*DESCRIPTION*
[2]	Hierarchical	Distributed towards root node	Low	low single point of failure		Increasing the resistance to the failure of a specific component by Using data and function redundancy
[3][4]	Hierarchical	Distributed towards root node	Low	low single point of failure		Using Mobile Agents to trace intruders among the various hosts involved in an intrusion
[5][6]	Centralized	Distributed towards root node	Moderate	low single point of failure	No security approach	Agents are composed dynamically using a genetic algorithm, which continually attempts to maximize the likelihood of discovering existing vulnerabilities.
[7]	Hierarchical	Distributed towards a gateway Agent	Moderate	Moderate No single point of failure		Approach was proposed to detect distributed intrusion among the network by various Agents.
[8]	Centralized	Symmetrical	High	Moderate No single point of failure		The presented intrusion detection system, DIDMA is designed by keeping in mind the notion of flexibility, scalability, platform independence
[9]	Hierarchical	Distributed towards upper level	Low	low single point of failure		They show how dynamic aggregation provides a mechanism for extending existing objects and allows us to quickly add new features to the system.
[10]	Peer to peer	Symmetrical	High	Moderate No single point of failure		A virtual neighborhood is created where neighbors take on the task of looking out for each other.
[11]	Peer to peer	Symmetrical	High	Moderate No single point of failure		Applying Mobile Agents technology to provide intrusion detection for Cloud applications regardless of their locations.

3. Proposal mobile agent IDS framework in hybrid cloud

In this section, we will define the cloud environment of our framework, its objectives, components and functions.

3.1 Cloud environment

The deployment models of cloud computing are [12]:

• Public cloud : The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider.

• Private cloud: The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises.

• Hybrid cloud: The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds).

• Community cloud : The cloud infrastructure is provisioned for exclusive use by a specific community of consumers

from organizations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be owned, managed, and operated by one or more of the organizations in the community, a third party, or some combination of them, and it may exist on or off premises.
In our proposed framework, we focus on detecting intrusions in hybrid cloud.

3.2 Framework objectives

The main framework objectives are :

- Distributing correlation and decreasing network load : To supervise all the network nodes, a central node should query them and collect detected intrusions information to analyze it. So the network traffic will increase. To do, we try to adopt a distributed correlated system based on mobile agents to reduce network load due to the migration of agents from one node to another.
- Reducing CPU load for each Cloud node: We try to distribute the work load of detecting intrusions between nodes instead of centralized it on one principal node.
- Securing communication: We want to adopt a secure mobile agents platform within an encrypted communication between agents in order to avoid any intrusion.
- Detecting distributed intrusions: An attack against a cloud computing system can be silent and not detected just in only one node,. In fact, cloud-specific attacks don't necessarily leave traces in one node. In this way, we propose to analyze IDS traces using data mining to detect new attacks.

3.3 Framework components

Our framework is based on six actors described as follows (see figure2):

1. An IDS : An IDS is deployed in each node (VM) in the hybrid cloud (private and public). The IDS monitors the traffics, detects intrusions and saves it in its database.
2. Correlated Mobile Agent (CMA): it is a mobile agent dispatched to each node in the cloud area. The CMA contains the rules to verify in each node using the alerts saved in IDS database. In the same time, the framework supports two CMA every one in each cloud area (public, private) to have rapidly a hole idea about the hybrid cloud intrusions.
3. a Public Cloud Agent (PbCA) : It is a static agent implemented in the administrator node in public cloud. This agent dispatch a CMA to detect intrusions and go back with all the results of the correlation process.
4. A Private Cloud Agent (PvCA) : It is a static agent implemented in the administrator node in private cloud. This agent dispatch a CMA to detect intrusions and go back with all the results of the correlation process.
5. An Hybrid Cloud Agent (HCA): It is a static agent implemented in the administrator node in hybrid cloud. This agent query the PbCA and the PvCA to start with the detection process in order to evaluate the security level in the hybrid cloud.
6. A Static Agent (SA) : The static agent is implemented in each VM to receive the CMA.

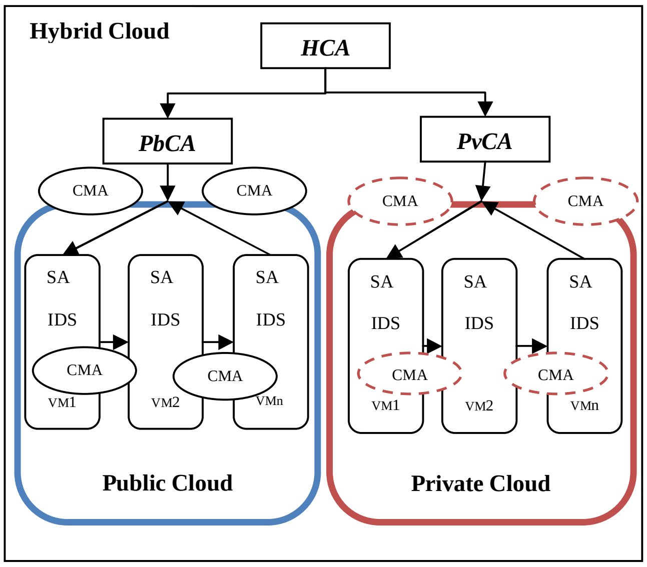


Figure 2. Mobile Agent IDS Framework in Hybrid Cloud

3.4 Framework Functions

We will describe the different functions and interactions between agents to detect distributed intrusions in the figure 3.

The HCA can manage all the hybrid cloud towards its cloud area: public cloud and private cloud. The management can only be done for one cloud area or the two area in the same time depending on the Cloud status.

Therefore, HCA asks the PbCA (11) or the PvCA (21) or both of them in the same time to report it the distributed intrusions detection in their cloud to audit the hybrid cloud.

The PbCA and the PcCA create a CMA with all the rules implemented in its code and dispatch it to the Cloud VM (12,22).

The CMA migrates to the SA (13,23). The SA receives The CMA and asks password. This step is very important because AS reject CMA if it is not authenticated.

After receiving CMA (14,24), CMA asks information stored in the IDS database. It hasn't permissions to access directly to IDS database so SA is the middleware. CMA applies all the rules in its data base to detect distributed intrusions in the VM. When finished, CMA moves to the next VM to repeat the same steps done in the first VM.

After finishing the detection in all the cloud VM, CMA reports the results to its Cloud Agent (PbCA (15) or

PvCA(25))). The Cloud Agent gives report to the HCA (16,26) to supervise the hybrid cloud.

If any VM is not connected or broken down, the CMA discovers the VM status and migrate to the next VM to continue its work.

The distributed detection process can be launched by the HCA , the PbCA or the PVCA.

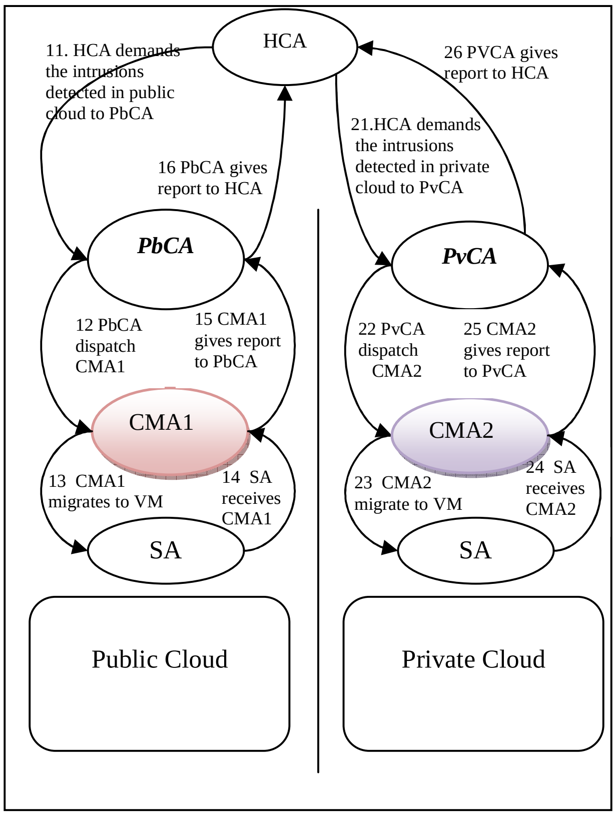

HCA
11. HCA demands the intrusions detected in public cloud to PbCA
26 PVCA gives report to HCA
21.HCA demands the intrusions detected in private cloud to PvCA
16 PbCA gives report to HCA
PbCA
PvCA
12 PbCA dispatch CMA1
15 CMA1 gives report to PbCA
22 PvCA dispatch CMA2
25 CMA2 gives report to PvCA
CMA1
CMA2
13 CMA1 migrates to VM
14 SA receives CMA1
23 CMA2 migrate to VM
24 SA receives CMA2
SA
SA
Public Cloud
Private Cloud

Figure 3. Agent IDS Framework intercommunication

4. Mobile agent based framework fixing vulnerabilities in hybrid cloud

When the HCA detects distributed intrusion, the cloud network administrator should take the necessary security measures and apply it immediately. For that, we propose to extend the IDS Framework in section III to fix vulnerabilities and avoid intrusions (see figure 4) .

If intrusions occurs, it means that there is a vulnerability in the VM or a missing policy security in the firewall. So HCA could dispatch a Reparation mobile agent (RMA) to:

- the vulnerable VM to repair it if there is any service to close or to reject any established communication with a malicious user.
- the firewall to apply new security rules to avoid intrusions detected. In this way, firewall should implement a Static Agent to receive the RMA in order to get rules and apply them.

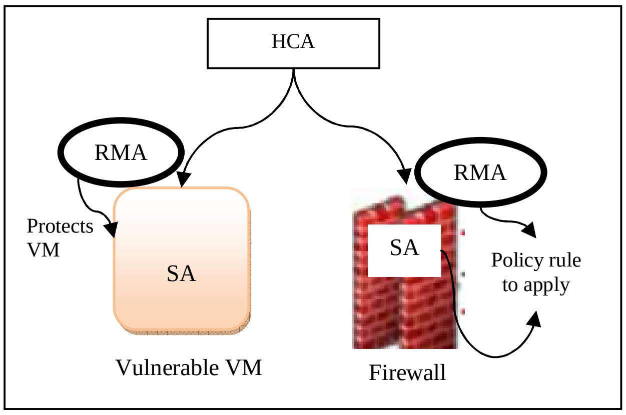


Figure 4. Repairing of vulnerability in cloud environment

5. Prototype Implementation

The proposed framework in the previous section is illustrating how specific features of the Mobile Agents can increase the efficiency of the system and decrease the network load as well (see figure 5).

Bee-Gent Mobile Agent has been used for implementation. Bee-Gent technology was first released in 1999 by Toshiba [13], as a new type of pure agent development framework for the advanced network society. Its communication framework is based on the multi-agents model. The Bee-gent framework is comprised of two types of agents: agent wrappers and mediation agents.

• Agent Wrappers are used to 'agentify' existing applications. The agent wrappers manage the states of the applications, which are wrapped around, and invoke the applications when necessary.

• Mediation Agents support inter-application co-ordination by handling all communications among applications. The mediation agents move from the site of an application to another where they interact with the remote agent wrappers.

For The IDS, we deployed SNORT[14] in each VM to monitor the system and the network intrusions. We configured snort to save alerts in its mysql database to deal with analyzed phase by CMA.

We choose iptables as a firewall in a linux machine to manage the repairing of vulnerability and the application of new security rules.

To implement our architecture, we've chosen the VMware vSphere Hypervisor 5 composed of an ESXi and vSphereClient. The choice of VMWare ESXI was made based on following reasons :

- Freeware version
- Solution qualified by the internet community as 'stable' and portable
- Fully managed through vSphere.
- Supports hot migration.

The figure 5 shows all the framework components to detect and avoid distributed intrusions in the cloud area.

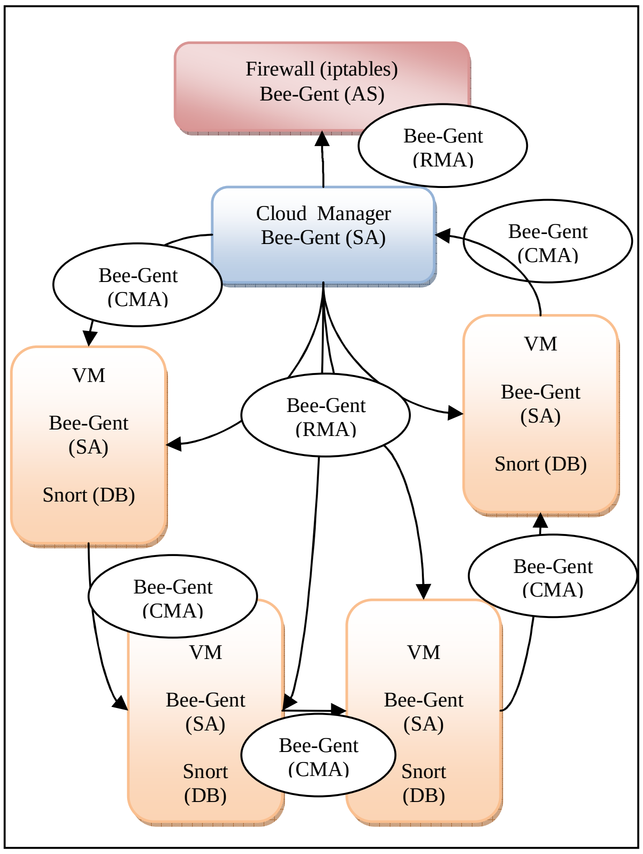


Figure 5. IDS Framework components in hybrid cloud

6. Framework Evaluation

In this section, our mobile agent IDS framework performance will be challenged while we are comparing it with the performance of client/server IDSs approach. Our aim is to verify our IDS features and effectiveness. The IDS with Mobile Agent approach claims the less network load compared to the client/server approach, by shipping code to data instead of shipping data to code .

In figure 6, we compare the network load (number of request exchanged in the network) for the client-server approach and the mobile agent approach according to the number of machines. So the mobile agent (CMA) is dispatched from the cloud manager (PbCM or PvCM) to each VM in the cloud to detect distributed intrusion and return back results. The number of request in this case is:

RequestNumber = VMNumber + 1

But when we use the client/server approach, the cloud manager should query each VM to receive response so:

RequestNumber = VMNumber *2

Consequently, the mobile agent concept becomes relatively interesting especially when the count of VMs increases.

the Bee-Gent mobile agent approach offers two important agent features:

- When the mobile agent migrates to a broken VM, it moves to an another to continue its work. So due to this property, we avoid the single point of failure.
- The mobile agent intercommunication should be authenticated and encrypted. This property avoid any attempted attack aiming to intercept agent communication.

Using mobile agents allows to fix vulnerabilities either in the VM or in the firewall by adding new security rules.

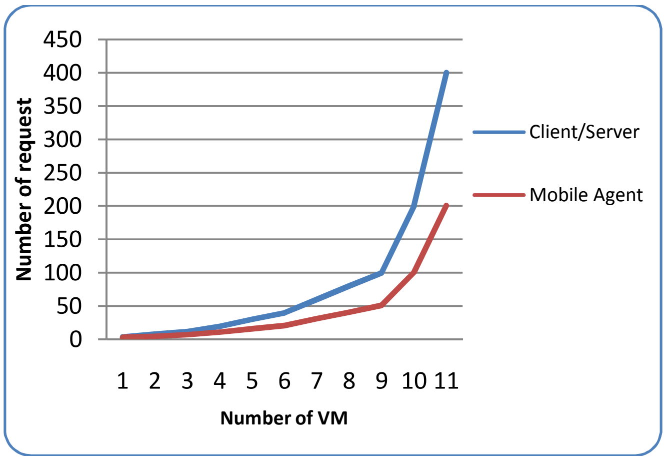


Figure 6. Evaluation of mobile agent versus client/server in IDS Framework

7. Conclusion

Cloud computing takes the essence of both Mobile agents and virtualization in a way to combine their key benefits. The VMs are the ideal platforms for agents to execute safely, based on the fact that virtual machine can be used to provide secure, isolated sand boxes for the Mobile Agents. In our framework, Clouds and Virtualization can benefit from IDS approach which mobile agents makes it scalable, flexible and cost effective.

In our future work, we will test this framework for detecting DDOS attacks in the cloud environment.

8. References

[1] A. khaldi, k. Karoui, N. Tanbène. H. Ben ghezala, “Secure cloud architecture design”, 2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, 2014 April, Oxford..

[2] J.Balasubramainyan, J.O. Garcia-Fernandez, D.Isacoff, E.H. Spafford, D.Zamboni, “An architecture of intrusion detection using autonomous Agents”, Department of Computer Science, Purdue University coast TR 98-05, 1998.

[3] Wayne Jansen, Peter Mell, Tom Karygiannis, Don Marks, “Applying Mobile Agents to Intrusion Detection and Response”, NIST Interim Report (IR) – 6416 October 1999.

[4] M.Asaka, S.Okazawa, A.Taguchi, and S.Goto, "A Method of Tracing Intruders by Use of Mobile Agents," INET'99, June 1999.

[5] Michael Conner, Chirag Patel, Mike Little, “Genetic Algorithm/Artificial Life Evolution of Security Vulnerability Agents,” Army Research Laboratory Federal Laboratory 3rd Annual Symposium on Advanced Telecommunications & Information Distribution Research Program (ATIRP), February 1999.

[6] Barrett, Michael, W. Booth, M. Conner, D. Dumas, M. Gaughan, S, Jacobs, M. Little, “Intelligent Agents System Requirements and Architecture,” Report to ATIRP, p. 5, October 1998.

[7] P. C. Chan and Victor K. Wei, “Preemptive Distributed Intrusion Detection using Mobile Agents”, Department of Information.

[8] Pradeep Kannadiga and Mohammad Zulkernine , “A Distributed Intrusion Detection System Using Mobile Agents”, School of Computing Queen’s University, Kingston Ontario, Canada K7L 3N, DIDMA:, 2005 IEEE.

[9] G. Helmer et al., Lightweight ,”Agents for intrusion detection”, The Journal of Systems and Software 67 (2003) 109–122.

[10] Geetha Ramachandran and Delbert Hart, “A P2P Intrusion Detection System based on Mobile Agents”, 2004 ACM 1-58113-870-9/04/04.

[11] Dastjerdi, Amir Vahid, Kamalrulnizam Abu Bakar, and Sayed Gholam Hassan Tabatabaei. “Distributed intrusion detection in clouds using mobile agents.” Advanced Engineering Computing and Applications in Sciences, 2009. ADVCOMP'09. Third International Conference on. IEEE, 2009.

[12] Mell, P. &Grance, T., 2011, “The NIST Definition of Cloud Computing”, NIST Special Publication 800-145 (Draft). Retrieved 2013-10-11)

[13] Bee-Gent, Online: http://flylib.com/books/en/4.4.1.92/1/ (January 2014)

[14] Snort, Online: http://www.snort.org/, (December 2013).

Distributed Fast Fourier Transform (DFFT) on MapReduce Model for Arabic Handwriting Feature Extraction Technique via Cloud Computing Technologies

Hamdi Hassen[1], **Khemakhem Maher**[1,2]
[1] Mir@cl Lab, FSEGS University of Sfax, Tunisia
[2] Computer Science Department, Faculty of Computing and Informnation Technology, King Abulaziz University, KSA

Abstract *- The choice of relevant features is very important and decisive step when building a handwriting recognition system. Indeed, a good choice can lead to a powerful system and vice-versa. Fast Fourier Transform (FFT) is amongst adequate feature extraction technique to achieve such an objective. Typical Arabic handwriting recognition tasks based on FFT and especially when dealing with a big and massive amount of Arabic handwriting documents require enough processing power that could not be provided by current state-of-the-art workstations. Distributed computing architectures and infrastructures appear to be a solution to afford such a mission.*
Our aim is indeed to distribute the FFT feature extraction techniques using the MapReduce programming model for Arabic handwriting feature extraction using Cloud Computing architecture.
Experiments were conducted on the MapReduce model via the Amazon Web service (AWS) Cloud Computing architecture, with a real large scaled dataset from the IFN/ENIT database.
Performance analysis revealed the viability of our investigation; moreover, it confirms also that such infrastructures can speed up substantially the entire pattern recognition system.

Keywords: Pattern Recognition, Mapreduce , FFT, Cloud computing

1. Introduction

The main task achieved by any OCR (optical Character Recognition) system is to convert a scanned text (offline) or handwriting on writing device (online) into a text document. Text recognition is a sub field of the pattern recognition area which has been the subject of so much research in the past three decades.

The Recognition of Arabic handwriting characters is a challenge in the last few years. This challenge is due to many factors such as: first, the great similarity between some characters. Second, the shape variation of most of Arabic characters according to their position in a given word or sub word (morphological problem). Third, the complexity and richness of the Arabic calligraphy especially for the Ancient documents. And fourth, the existence of words and sub words in a given text. These factors become much more difficult when dealing with large amount of documents which is our main objective through this work.

However, we believe that a good selection of feature extraction technique remains one of the most important steps for achieving good recognition accuracy. Such a selection requires the knowledge of both, the robust and efficient feature extraction technique that can lead to relevant features and the adequate hardware architecture that can handle such robust feature extraction technique in order to computerize large amount of documents in a reasonable time.

High performances of FFT is a key issue in Arabic handwriting recognition system. Nevertheless, it represents a complex techniques when processing a massive database of Arabic handwriting text.

Distributed computing architectures such as Cloud Computing technologies provide enough computing capacities to process this kind of complex algorithm.

Our idea consists on using the programming model MapReduce for processing large Arabic handwriting data sets with a distributed FFT algorithm on a real Cloud Computing platform. The aim of this work is to demonstrate that the performances of FFT on the feature extraction step can be achieved by the distribution of FFT using MapReduce programming model on Cloud architecture .

The rest of the paper is organized as follows: Section 2, describes the feature extraction difficulties of the Arabic handwriting Characters. Section 3 presents a general description of the FFT as feature extraction techniques and its complexity. Our approach is presented in the fourth section. Section 5 provides some performance evaluation and investigation of our approach. Finally, a conclusion with some remarks and future work are presented in Section 6.

2. Arabic handwriting Features

The Arabic handwritten language presents many challenges to the Optical Character Recognition (OCR) developer [1] that can be presented as follows:

In Arabic language, a text is composed of cursive words and sub words where each of which is formed of consecutive letters linked one to another sometimes with some overlaps [2]. Most of Arabic alphabet letters are presented by four forms according to their position in a given word: isolated form, initial form, medial form, and final form. In addition, the corresponding shape of each of these forms can be completely different from the others for the same character. Moreover, some of Arabic characters have similar shape and only diacritic dots make them different but unfortunately more complicated. This is why the recognition rate of Arabic characters is lower compared to other languages especially Latin. These properties will cause a high level of difficulty in the choice of the pertinent and relevant features [3] during the design of an Arabic OCR system.

The analysis of Arabic handwriting script is further complicated compared to Latin script due to obligatory dots/stokes that are placed above or below most letters. Some of the dots and strokes can be missed in the preprocessing step where the text should be cleaned up to be used directly and efficiently by the feature extraction technique in the OCR process[1]

Another difficulty of Arabic handwriting recognition due to the different writing styles, in fact Arabic handwriting can be in different style such as Ruqqah and some others usually used for decorative calligraphy such as Kofi, Thuluth and Diwani. This feature will cause more difficulties for recognition and make the learning database of the recognition system even larger[1].

The choice of the robust and efficient algorithm to deal with the relevant and pertinent features is very important and decisive in handwriting recognition rate.

Results of surveys conducted by Arabic handwriting OCR researchers confirm that high performance of the fast Fourier transform (FFT) is a key issue for Arabic handwriting recognition system, in fact it represents a robust and efficient features extraction technique for Arabic handwriting recognition process [4].

3. Fast Fourrier transform as a features extraction technique

The FFT is a fast algorithm to implement the Discrete Fourier Transform (DFT) [5].

A FFT is an algorithm that computes the DFT and its inverse. A Fourier transform converts time (or space) to frequency and vice versa; an FFT rapidly computes such transformations by factorizing the DFT matrix into a product of sparse (mostly zero) factors [6].

The main principle of FFT is "Divide and Conquer to break down a big problem to a number of smaller problems and tackle them individually". The FFT needs to satisfy the flowing condition [7] .

$\sum$ cost (sub problem) + cost (overhead) < cost (original problem).

The purpose of our system is to recognize large amount of Arabic handwriting text using FFT as a feature extraction technique implemented on distributed architecture .

The recognition of the Arabic handwriting by FFT consists previously on applying operating the Fourier Transform on the contour of the character [8]. First, we start with the detection of the contour. Second , the code of Freeman of the contour is generated on which one operates the calculation of Fourier Transform. Figure 1 depicts the FFT process.

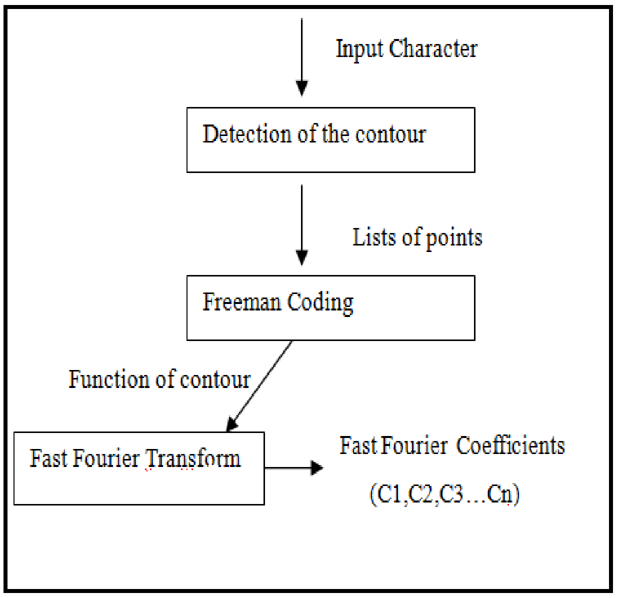


Figure 1. The FFT process

Mathematically, the process of Fourier Transform is represented by the following equation:

$$X_N(K) = A_0 + \sum_{n=1}^{N} a_n \cos\frac{2\pi nK}{N} + b_n \sin\frac{2\pi nK}{N} \quad [1]$$

$$Y_N(K) = C_0 + \sum_{n=1}^{N} c_n \cos\frac{2\pi nK}{N} + d_n \sin\frac{2\pi nK}{N} \quad [2]$$

Where

K: the k points of the contour,

N: is the necessary number in the approximation of the contour by the coefficients of Fourier

a_n, b_n, c_n et d_n: The coefficients of Fourier corresponding to the harmonic n.

a_0 et c_0: The continuous components that correspond to the initial points where the frequency is equal to 0.

The FFT algorithm computes the DFT and produces exactly the same result as evaluating the DFT definition directly. The most important difference is that an FFT is much faster.

FFT represents an efficient and robust feature extraction technique that uses Fourier Descriptors (FD). In fact, it can be normalized in order to be invariant to the position of the characters (translation or rotation), the size and the starting point [9]. But, FFT is characterized with a complexity order of O(n log n) [10] [11].

Another weakness of FFT applied to Arabic handwriting features extraction is that the extracted primitives need a high level of memory consuming [12].

It appears that the optimization of FFT is as equally demanding as the design of an efficient feature extraction technique. Our optimization of FFT consists on using the MapReduce model to distribute the FFT algorithm via the cloud computing architecture.

4. MapReduce model for distributed FFT on cloud computing architecture

MapReduce technique is a programming model for processing massive data sets with a parallel and distributed manner on a distributed architecture. A MapReduce program is composed of two main function: Map() and Reduce() [13].

In the Map function, the master node takes the input, divides it into smaller sub-problems and distributes them to worker nodes. In the reduce function, the worker node may do this again in turn, leading to a multi-level tree structure. The worker node processes the smaller problem, and passes the answer back to its master node. Figure 2 explains the MapReduce Model.

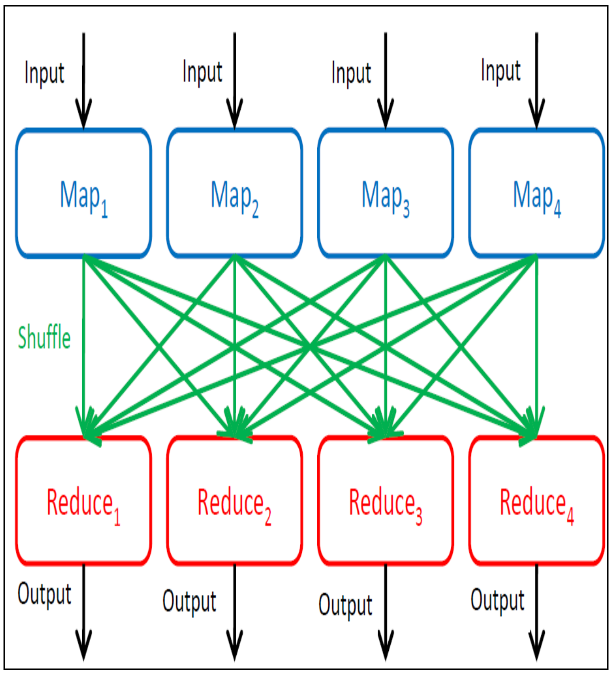


Figure 2: Map reduce model

Our idea consists of distributing the computational FFT as features extraction techniques across a cluster of virtual servers running in the Cloud Computing architecture using the MapReduce model to analyze and process a large and massive amounts of arabic handwriting texts. The figure below describes the design of our approach.

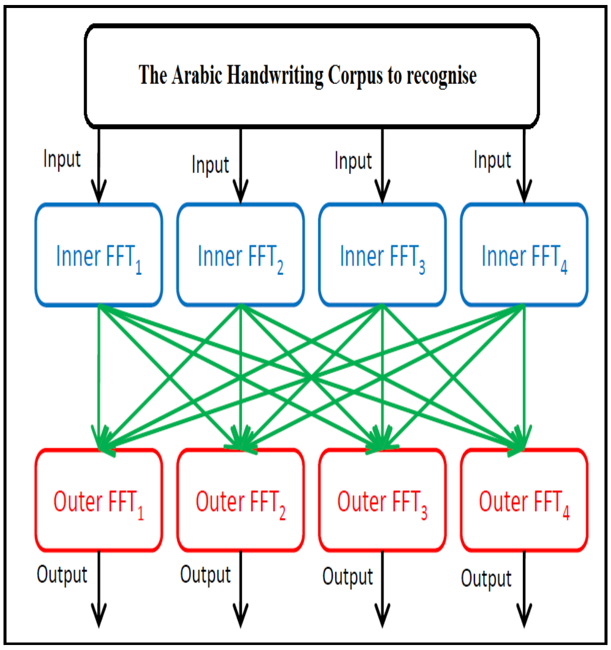


Figure 3: FFT via MapReduce model

The open-source framework Hadoop[14] is used as a tool to manage clusters in cloud architectures. The distributed processing model MapReduce is used by Hadoop on the Amazon web service cloud architecture[15] in which the FFT task is mapped to a set of servers for processing then the results of the map function is performed by the reduce function to achieve a single output set. The distribution and control of FFT algorithm is done by a node called the master node. Figure 4 illustrates the AWS MapReduce mechanism.

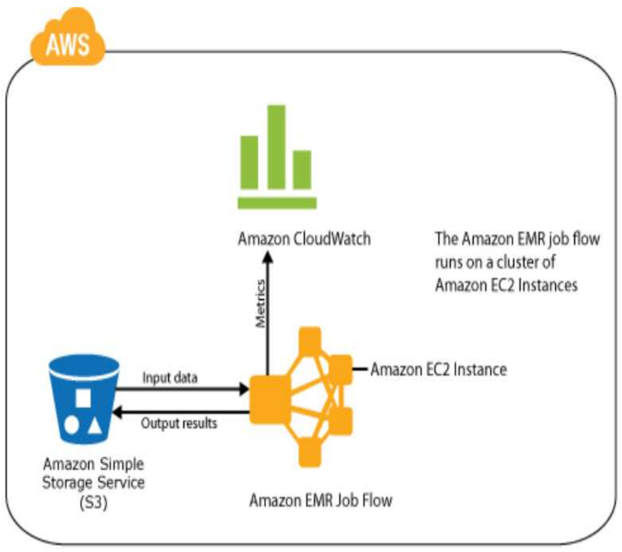

AWS
Amazon CloudWatch
Metrics
The Amazon EMR job flow runs on a cluster of Amazon EC2 Instances
Amazon EC2 Instance
Input data
Output results
Amazon Simple Storage Service (S3)
Amazon EMR Job Flow

Figure 4: AWS MapReduce model

5. The experimental study

5.1. Datasets

To evaluate the performance of DFFT as features extraction techniques based on MapReduce model via the cloud architecture, a database with 16000 pages (370 characters/page) is used.

The reference database is formed of 345 shapes representing approximately the different Arabic alphabet randomly chosen from the Arabic handwritten word images dataset. This base contains samples of 945 names of Tunisian Towns retrieved from the already normalized and preprocessing database IFN/ENIT Institut of Communications Technology (IFN),Technical University Braunschweig, Germany , Ecole Nationale d'Ingénieurs de Tunis (ENIT), Tunisia [16]. Figure 5 represents un sample of the studied corpus.

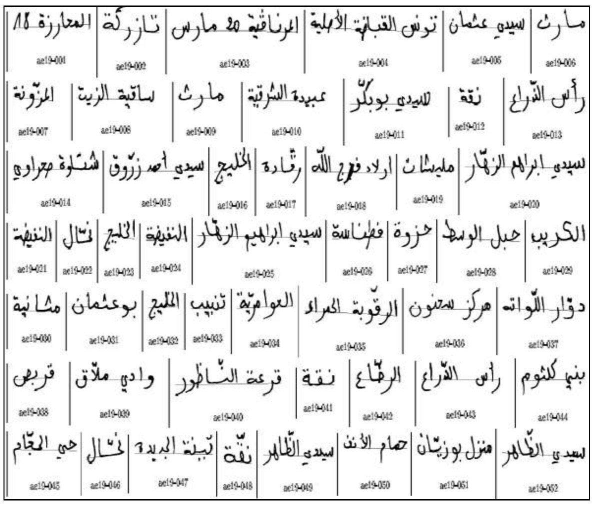


Figure 5: A sample of the studied corpus

5.2. Experimental environment

To set up an experimental environment that would allow us to objectively study the effectiveness of Distributed FFT across a cluster of virtual servers running in a cloud architecture via the MapReduce model in the entire Arabic handwriting recognition process , we used the following tools:

- A local Intel Core 2 Duo desktop having the configuration: 3.00 GHz *2, 2 GB of RAM running a Windows XP operating system,
- The Cygwin shell to run Linux command [17].
- The java programming language and JDK 1.6 was istalled.
- The Eclipse 3.4 tool was used to program and build our OCR application based on FFT as a feature extraction technique.
- The cascading framework [18] was used to easily and quickly develop the Data Analytics and Data management
- 100 cores using the three Standard Amazon EC2 Instances.
  - ✓ The small instances each with 1.7 GB of memory, 160 GB of instance storage, and 32-bit platform.
  - ✓ The Large Instance 7.5 GB of memory, 850 GB of instance storage, 64-bit platform
  - ✓ The Extra Large Instance 15 GB of memory, 1690 GB of instance storage and 64-bit platform.
- The Amazon S3 Bucket [19] is used to store and receive the input and the output into and from the cloud clusters.

The execution of FFT in AWS MapReduce model should respect some steps. First, we start by developing and executing our OCR application based on FFT as a feature extraction technique in the same local host using Java programming language. Second we should Sign Up to amazon web service to Upload our application and data to Amazon S3. The database to upload and process is very massive, we have using the AWS Import/Export option based on the use of physical storage devices. The configuration and the launch of clusters is the third step in which we use the AWS management console to specify the number of EC2 instances that will be used in cluster, and the types of instances to use. Finally, when Amazon EMR will automatically terminate the cluster when processing is complete, we retrieve the output from Amazon S3 on the cluster. Many tools are used to visualize the output like MicroStrategy[20].

5.3. Results and analysis

This section presents the results and analysis of the experimental study conducted in the real IFN/ENIT database using distributed FFT as feature extraction technique on Mapreduce model on the AWS cloud computing architecture.

3 running jobs flows are created in cascading AWS MapReduce respectively for 3 different transform size (N= 64, N= 128, N= 256).

The execution time, the speedup factor and the efficiency factor are chosen to evaluate our experimental [21] . The execution time is the maximum length of time that FFT could take to extract features from characters on a specific Amazon EC2 instances for a definite transform size (N). The speedup refers to how much the distributed FFT is faster than its corresponding in a sequential manner and finally the efficiency factor that refer to the pour cent of useful of resources

The experimentally derived values of the execution time, the FFT speed up factor and the efficiency factor on AWS MapReduce model for different instances of Amazon EC2 and different transform size are given in Table 1.

TABLE I EXECUTION TIME, SPEEDUP FACTOR AND EFFCIENCY FACTOR OF FFT ON AWS MAPREDUCE MODEL

		The execution time(h)			**The speedup factor(%)**			**The efficiency factors(%)**		
Trans-form size	**Number of cores**	**Amazon EC2 Instances**			**Amazon EC2 Instances**			**Amazon EC2 Instances**		
		Small	**Large**	**Extra large**	**Small**	**Large**	**Extra large**	**Small**	**Large**	**Extra large**
N= 64	25	0.671	0.611	0.551	14.158	15.548	17.241	0.566	0.622	0.690
	50	0.400	0.384	0.368	23.750	24.740	25.815	0.475	0.495	0.516
	75	0.316	0.301	0.280	30.063	31.561	33.929	0.401	0.421	0.452
	100	**0.261**	**0.254**	**0.245**	**36.398**	**37.402**	**38.776**	**0.364**	**0.374**	**0.388**
N= 128	25	0.551	0.485	0.432	17.241	19.588	21.991	0.690	0.784	0.880
	50	0.280	0.265	0.243	33.929	35.849	39.095	0.679	0.717	0.782
	75	0.180	0.185	0.175	52.778	51.351	54.286	0.704	0.685	0.724
	100	**0.140**	**0.135**	**0.130**	**67.857**	**70.370**	**73.077**	**0.679**	**0.704**	**0.731**
N= 256	25	0.530	0.470	0.420	17.925	20.213	22.619	0.717	0.809	0.905
	50	0.260	0.240	0.223	36.538	39.583	42.601	0.731	0.792	0.852
	75	0.160	0.170	0.165	59.375	55.882	57.576	0.792	0.745	0.768
	100	**0.150**	**0.130**	**0.120**	**63.333**	**73.077**	**79.167**	**0.633**	**0.731**	**0.792**

The major observations are as follows:

- The average test time of FFT in a sequential mode is 9.5 hours and on a distributed architecture with 100 computers the execution time is 0.150 hour, 0.130 hour and 0.120 hours respectively for the three Amazon EC2 Instances.
- The execution time, The speedup and the efficiency of FFT increase linearly as FFT size increases and as Amazon EC2 Instances integrate more memory capacities.

Consequently, obtained results confirm that:

- AWS MapReduce is an adequate framework to speed up the FFT feature extraction technique applied in a greedy process (the Arabic handwriting recognition system). In fact if we use 100 cores with an extra large AWS instance with 256 FFT size, we can extract the inerrant and pertinent features of 1370 characters in a second with a speedup factor equal to 79%.
- AWS MapReduce is an efficient tool to develop a distributed FFT as a large scale Arabic handwriting feature extraction technique . In fact for FFT size equals to 256 with an extra large AWS instance , the system is used for 79%.

6. Conclusion and perspective

In this paper, we have proposed an approach to distributed FFT as a feature extraction technique for Arabic handwriting recognition system using MapReduce Model via cloud computing architecture.

Experimental results of DFFT on AWS MapReduce are presented and confirmed the viability of our investigation. Performance analysis confirmed indeed that FFT on AWS MapReduce can provide an effective framework to speed up the feature extraction process.

Further investigations are under study and could extend the development of a powerful Arabic handwriting recognition system based on MapReduce model via the cloud computing architecture which constitutes indeed our main objective.

ACKNOWLEDGMENT

The authors wish to thank the reviewers for their fruitful Comments. The authors also wish to acknowledge the members of the Miracl Laboratory, Sfax, Tunisia, the members of Department of Mathematics and Computer Science, University of Marburg, Germany and the members of computer science department, College Of Science And Arts at Al-Ola, Taibah University, KSA

7. References

[1] Aburas, A.A. Gumah, M.E.Arabic Handwriting Recognition: Challenges and Solutions, International Symposium on Information Technology, 2008. ITSim, Kuala Lumpur, Malaysia Page(s):1 - 6, 2008.

[2] Raid S, Jihad. E, Hierarchical On-line Arabic Handwriting Recognition, 10th International Conference on Document Analysis and Recognition, Barcelona, pages 86- 871, 2009.

[3] John M, Thad S, Richard S, and George C. On-line cursive handwriting recognition using speech recognition methods. In Proceeding of IEEE ICASSP’94 Adelaide, pages v125–v128, Adelaide, Australia, April 1994.

[4] F. Kuhl. Elliptic fourier features of a closed contour. Computer Graphics and Image processing 18, pages 236–258, 1982.

[5] K. Arbter. Affine-invariant fourier descriptors. from pixel to features. Elsevier Science Publisher B.V (North-Holland), 1989

[6] Snoussi Maddouri, S., Reconnaissance de l’Ecriture Arabe Manuscrite par Réseau de Neurones Transparent et Transformée de Fourier, Journée des jeunes chercheurs sur l’Ecrit et le Document, Colloque International Francophone sur l’Ecrit et le Document (JEDCIFED), Lyon, France, 2000.

[7] Sabri A, Achraf S, Arabic Character Recognition using Modified Fourier Spectrum (MFS), GMAI '06 Proceedings of the conference on Geometric Modeling and Imaging: New Trends, Washington, Pages 155 - 159, 2006.

[8] M.Szmulo, «Boundary normalization for recognition of non touching non-degraded characters », ICDAR, IEEE, 1997, pp 463-466.

[9] K. Arbter. Affine-invariant fourier descriptors. from pixel to features. Elsevier Science Publisher B.V (North-Holland).

[10] Granlund G H (1972). Fourier Preprocessing for Hand Print Character Recognition. In IEEE Transactions on Computers, February 1972.

[11] Kauppinen, et al An experimental comparison of autoregressive and Fourier-based descriptors in 2D shape classification. In IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.17, No.2, 1995.

[12] Haikal El-Abed, Sofiene Haboubi Samia Maddouri Noureddine Ellouze, Invariant Primitives for Handwritten Arabic Script: A Contrastive study of four feature sets, 10th International Conference on Document Analysis and Recognition, pp 691-697, 2009.

[13] Dean J. and Ghemawat S., \Mapreduce: Simpli_ed data processing on large clusters", Communications of the ACM - 50 th anniversary issue: 1958 - 2008 2008, vol. 51, Jan. pp. 107-113.

[14] Varia J., Mathew S., Overview of Amazon Web Services, 2013.

[15] Available at: http://hadoop.apache.org/

[16] M. Pechwitz, S. S. Maddouri, V. Mrgner, N Ellouze, and H. Amiri. \Ifn/enit - database of handwritten arabic words"In In Proc. of CIFED, Tunisie, 2002, pages. 129 - 136, 2002.

[17] Available at: http://www.cygwin.com/

[18] Available at: http://www.cascading.org/

[19] Available at: http://aws.amazon.com/s3/

[20] Available at: http://www.microstrategy.com/

[21] V. Kumar, A. Grama, A. Gupta, and G. Karypis.\ Introduction to Parallel Algorithm Design and Analysis", Benjamin Cummings, Redwood City - USA, 1994.

SESSION

ENERGY EFFICIENT NETWORKING SYSTEMS APPLICABLE TO EMERGENCY

Chair(s)

Prof. Hiroaki Nishikawa
University of Tsukuba
Japan

An optimization study on Broadcast Based Information Sharing System (BBISS)

Sayuri Wada[1], Hiroshi Ishii[2], Hiroaki Nishikawa[3], and Keisuke Utsu[2]
[1]Graduate School of Information and Telecommunication Engineering,
Tokai University, Minato, Tokyo, Japan
[2]School of Information and Telecommunication Engineering, Tokai University, Minato, Tokyo, Japan
[3]Graduate School of Systems and Information Engineering, University of Tsukuba, Tsukuba, Ibaraki, Japan

Abstract - *We have proposed the broadcast based information sharing system: BBISS which deliveries and shares the information such as text data and image data by broadcast communication in the surrounding area when the communication infrastructure cannot be used by large-scale disasters. We have already executed a preliminary evaluation of the system performance but most appropriate parameter values have not been clarified yet. Hence, this paper aims at clarifying the optimal values of parameters for BBISS by use of network simulators.*

Keywords: Broadcast, Ad hoc communication, Information sharing

1 Introduction

Large-scale disasters disable communication infrastructures such as base stations and servers. In the situations, people wish to obtain disaster information and safety information by versatile applications such as text, voice, image, or video data, in the infrastructures unavailable areas. To enable those application without using the infrastructures, a novel communication method, to deliver the information in the area with only ad hoc communication fuctions on users' mobile communication terminals, is required.

To tackle with the above issue, we have proposed a novel communication system named Broadcast Based Information Sharing System, BBISS [1]. The effectiveness of the system has been shown preliminarily in [2] and [3]. The system uses the redancancy of radio broadcast communication to deliver the information efficiently. In addition, the system has a function that can complement packet loss to supplment unstableness of the radio communication. The study [2] and [3] has been shown that BBISS perform the better performance than existing methods in term of the information reachability and reduction of redundant packet reception. Althogh the study [2] [3]show that the performance of BBISS depends on its parameters settings, the optimal parameter settings have not been studied yet. Therefore, this study optimizes the parameters of BBISS through the network simulations.

Section 2 explains the outline of BBISS. Section 3 shows the optimization of several parameters used in BBISS through the newtork simulations. Lastly, Section 5 concludes the study.

2 Parameters to be optimized inthe proposed system, BBISS

As stated in Introduction, we have already proposed the BBISS for information sharing without the infrastructure in disaster situation. The details of the system architecture and its operation are well described in [2][3].

Here, we only show the parmeters to be optimized.

(1) *req_threshold*: the maximum number of times the "Sending state" transits to "Retrans req state".

(2) *relay_threshold*: in the relay decision state, if the number of relaying nodes does not reach the relay threshold, the state transits to "Sending state".

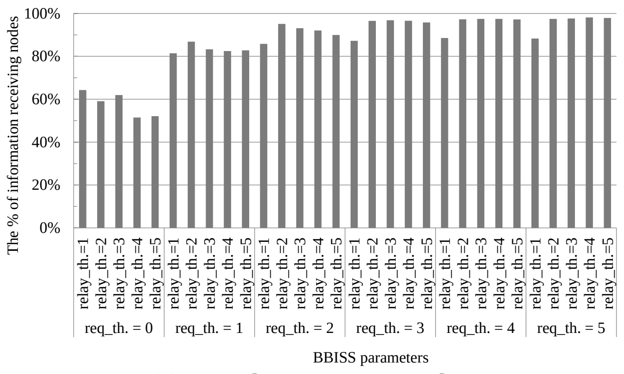
As described in [2][3], the selection of these parameters are very important to optimize the BBISS operation. In the next chapters below, we study the optimization of those parameters.

3 The evaluation by the simulation

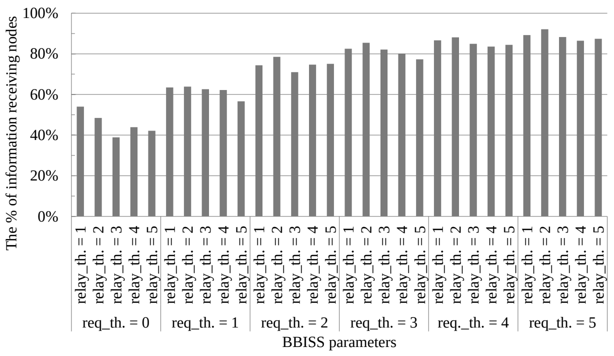
In this chapter, we optimize the parameter for the proposed scheme BBISS by using the network simulator OPNET [4].

3.1 The simulation environment

The simulation environment is as follows. The area size is 1000m x 600m. The numbers of nodes in the area are 200 nodes (Simulation A) and 400 nodes (Simulation B). The initial position of nodes is random. All nodes move at a speed of (0.00, 4.00) m/s according to the Random Waypoint model. The MAC layer of the nodes is IEEE802.11b, the data rate is 11Mbps. The radius of communication area (one hop radio area) is 150m. The *payload_size* of packet is 1024Byte. The number of initiator nodes which generate information is 10% of all nodes in the area. The generated information size is 100kByte, which is constituted by 100 packets. *send_interval*

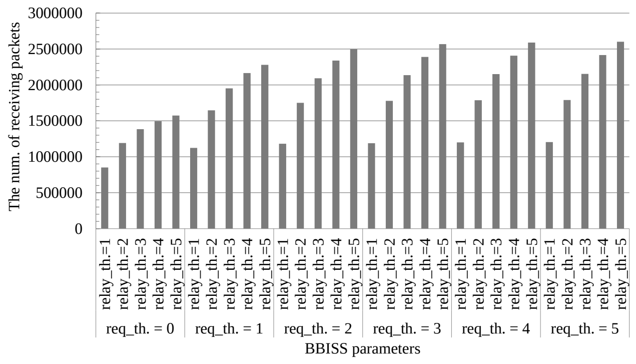


(a) Simulation A : 200 nodes

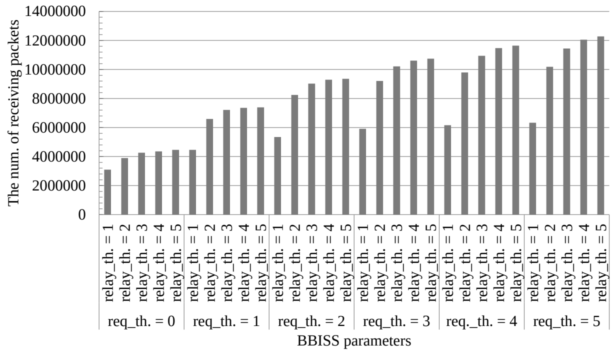


(b) Simulation B : 400 nodes

Fig. 1 Simulation result for (i) the percentage of information receiving nodes in the area



(a) Simulation A : 200 nodes



(b) Simulation B : 400 nodes

Fig. 2 Simulation result for (b) the num. of receiving packets in the area

between successively sent packets is 33ms. The parameter values of BBISS are set as follows. *relay_threshold* is set as an integer in the range of 1 ~ 5, and the *req_threshold* is set as an integer in the range of 0 ~ 5.

3.2 Evaluation items

The average values of the following items (i)-(v) are found for 100 simulation runs at every value of the random seed.

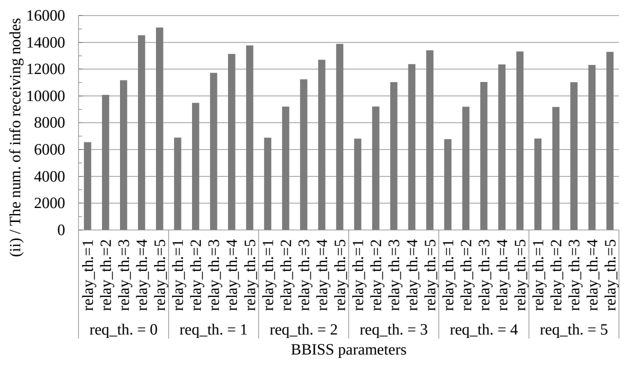
(i) The % of information receiving nodes [%]
Among all nodes except the initiator nodes, the percentage of the nodes which received the information is shown. The higher the percentage is, the better performance is.

(ii) The number of receiving packets in the area
The total number of packets that are received by the nodes in the area is shown. The smaller the value is, the better the performance is.

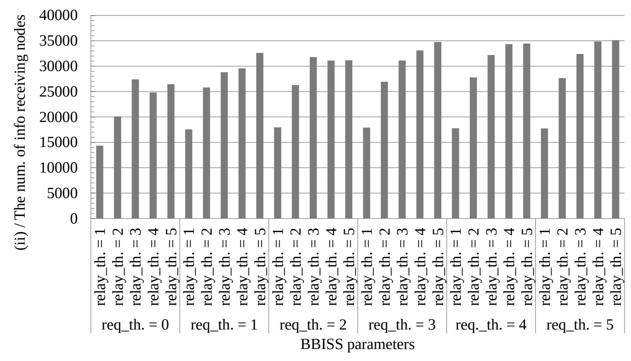
(iii) The number of receiving packets in the area, (ii) / the number of nodes received the information successfully,(i) x (the number of all the nodes)/100.
To normalize (ii), (ii) is divided by the number of the information receiving nodes. The smaller the value is, the better the performance is.

(iv) The number of retransmission request packets
The total number of retransmission request packets which the nodes sent in the area is shown.

(v) The average time of information delivery [s]
The delivery time (from the instant when information is generated in the application layer of the initiator node to the instant when the packet has been received by every node and reaches the application level) is averaged across all the receiving nodes. The smaller the value is, the better the performance is.

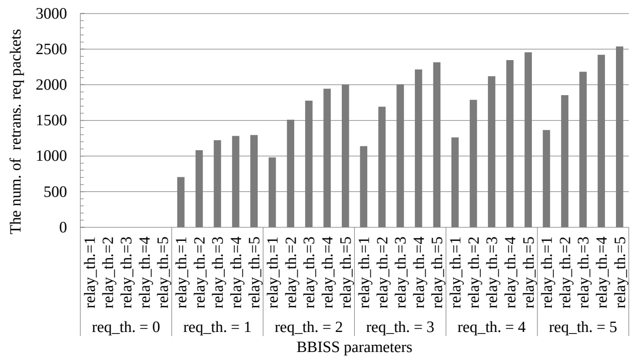


(a) Simulation A : 200 nodes

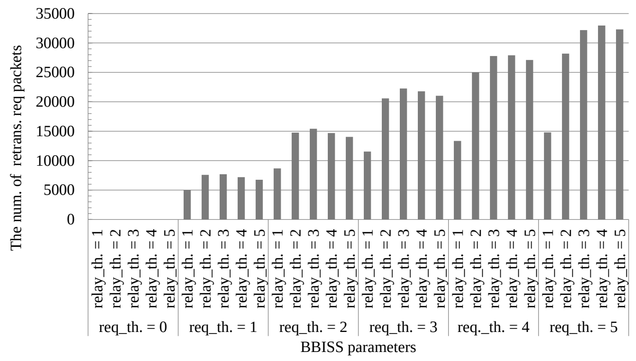


(b) Simulation B : 400 nodes

Fig. 3 Simulation result for (iii) the num. of receiving packets / the num. of information receiving nodes in the area

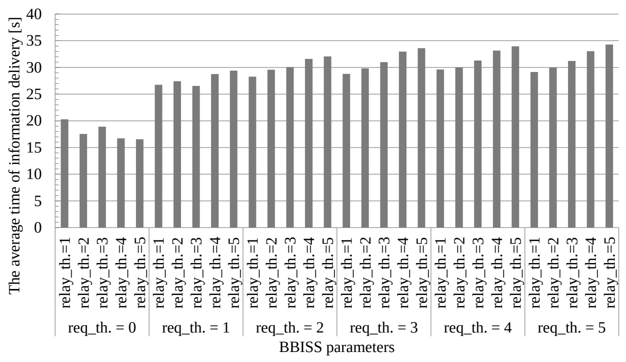


(a) Simulation A : 200 nodes

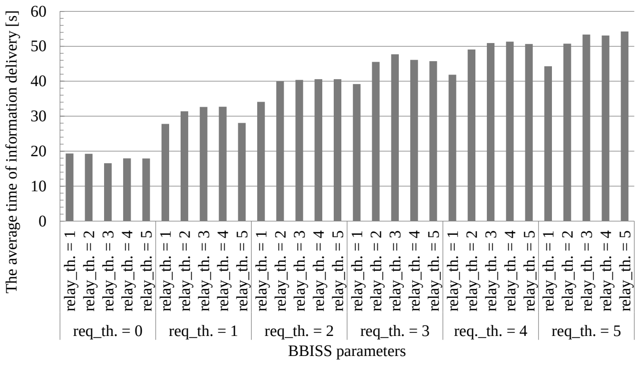


(b) Simulation B : 400 nodes

Fig.4 Simulation result for (iv) the num. of transmitted retransmission req pakcets in the area



(a) Simulation A : 200 nodes



(b) Simulation B : 400 nodes

Fig. 5 Simulation result for (v) the average time of information delivary

3.3 Simulation Results and Consideration

The results of the simulations are shown in Fig. 1 ~ Fig. 5.

(i) The % of information receiving nodes [%]

The results of Simulation A and Simulation B were shown in Fig. 1 (a) and (b), respectively.

In Simulation A, the case where *req_threshold* was more than or equal to 3 and *relay_threshold* was more than or equal to 2, the % of information receiving nodes was more than 95%.

In Simulation B, when *req_threshold* was 2, 3, 4, or 5 and when for each *req_threshold, relay_threshold* was 2, the % of information receiving nodes were the highest among the cases where other values of *relay_threshold* was set.

Furthermore, comparison of simulations A and B showed that the % of information receiving nodes was less in Simulation B than that in Simulation A. This can be explained as follows. Since the number of nodes in the area in Simulation B was larger than that in A, the number of relaying nodes was larger and the data flame collisions was increased.

(ii) The number of receiving packets in the area

The results of Simulation A and Simulation B are shown in Fig. 2 (a) and (b), respectively.

In Simulations A, and B, in all the *req_threshold* cases, the larger *relay_threshold* was, the more the number of receiving packets was. Here, in the Simulation A, if the *req_ threshold* was larger than or equal to 3 and *relay_threshold* was the same value for each *req_threshold*, no major difference in the number of receiving packets appears even if the value of *req_threshold* was made larger.

(iii) The number of receiving packets in the area / the number of nodes that received the information successfully

The results of Simulation A and Simulation B are shown in Fig. 3 (a) and (b), respectively.

In Simulation A for all *req_threshold*, the larger *relay_threshold* was set, the larger the value was shown. In Simulation B, the case where *req_threshold* was more than or equal to 3, the larger *relay_threshold* was set, the larger the value was shown.

(iv) The number of retransmission request packets

The results of Simulation A and Simulation B are shown in Fig. 4 (a) and (b), respectively

When *req_threshold* was set to 0, retransmission request packet was not sent and so the value was 0. In Simulation A, regardless of *req_threshold*, the larger *relay_threshold* was, the more retransmission request packets were transmitted. In Simulation B, regardless of *req_threshold*, the case where *relay_threshold* was more than or equal to 3 or 4, the total number of retransmission request packets were maximum.

(v) The average time of information delivery [s]

The results of Simulation A and Simulation B are shown in Fig. 5 (a) and (b), respectively

In Simulation A, the cases where *req_threshold* were more than or equal to 2, no significant difference were shown for the same value of *relay_threshold*. In Simulation B, the larger *req_threshold* was, the larger average time of information delivery was performed.

The case where *req_threshold* was 0 both in Simulation A and B, the average time of information delivery was shorter than the cases where *req_threshold* was greater than or equal to 1.

The results can be explained as follows. The case where *req_threshold* was 0, the time was short, because the nodes do not send retransmission request packets.

3.4 The optimum value of the parameter

The optimum value of the parameters in BBISS is discussed according to the simulation result.

In terms of (i) the percentage of nodes that received information successfully, the optimum value of *req_threshold* is more than or equal to 3, and that of *relay_threshold* is more than or equal to 2 in Simulation A and for Simulation B, the cases where *req_threshold* is 2 and over and *relay_threshold* is 2 is considered to be optimum.

Taking into account of (iii) the number of receiving packets in the area / the number of nodes that received the information successfully, the optimum values of *req_threshold* is greater or equal to 3 and the value of *relay_threshold* is 2 for Simulation A and for Simuation B, *req_threshold* is larger than and equal to 2 and *relay_threshold* is equal to 2 is optimum.

Lastly, taking into account of (v), for Simulation A, same as (iii), the case where *req_threshold* is greater than or equal to 3 and *relay_threshold* is 2 is optimum. For Simulation B, considering that the smaller the req_threshold is the smaller the value(v) is, the case where *req_threshold* is equal to 2 and *relay_threshold* is 2 is optimum Thus, we can conclude that the optimum values of the BBISS parameters are 2 or 3 for req *req_threshold* and 2 for *relay_threshold* .

4 Conclusions

We have proposed Broadcast Based Information Sharing System (BBISS) which can deliver and share image and text data size infomration in the infrastructures unavailable area by using a broadcast communication. In this paper, we studied the optimum parameters for BBISS by the network simulations.. Acording to the simulation results, we can conclude that the optimum value of *req_threshold* is 2 or 3 and that of *relay_threshold* is 2. The future issueto be tackled is whether the optimum values got in this paer will be suitabe for other simulation conditions such as node density and mobiity seed.

5 Acknowledgments

This study has been supported by the following foundations.

The Telecommunications Advancement Foundation, Japan

The Science Research Promotion Fund, The Promotion and Muual Aid Corporation for Private Schools Japan

6 References

[1] Keisuke Utsu, Hiroaki Nishikawa and Hiroshi Ishii, "A Proposal on Broadcast based Information Sharing System over Disaster and Congestion Tolerant Ad Hoc Network", the 2013 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'13) , pp.599-603, (2013)

[2] Keisuke Utsu, Hiroshi Sano, Hiroshi Ishii. "A proposal on Broadcast Based Information Sharing System for Infrastructure Unavailable Situations", CMN-14-001, The papers of Technical Meeting on "Communications", IEE Japan (p. 1-6), Jan 2014

[3] Keisuke Utsu, Sayuri Wada, Hiroshi Ishii. "Performance Evaluation on Infrastructure-less Broadcast Based Information Sharing System, BBISS", CMN-14-002, The papers of Technical Meeting on "Communications", IEE Japan (p. 7-12), Jan 2014

[4] The network simulator "OPNET", http://www.opnet.com/

A Safety Information Sharing Application on BBISS

Tomomi Itoh[1], Ayami Manaka[1], Yuuka Sugawara[1], Hiroshi Ishii[1], Hiroaki Nishikawa[2], and Keisuke Utsu[1]
[1]School of Information and Telecommunication Engineering, Tokai University, Minato, Tokyo, Japan
[2]Graduate School of Systems and Information Engineering, University of Tsukuba, Tsukuba, Ibaraki, Japan

Abstract – *Large-scale disasters frequently happen in Japan. People in disaster areas may try to send and get safety information of themselves, their family or friends. However, since communication infrastructure including servers are often unavailable in the situations, it may be difficult for the people to share the information using the infrastructure. To solve this problem, we have proposed Broadcast Based Information Sharing System, BBISS, which can communicate in peer-to-peer (P2P) manner without servers and the communication infrastructures. On the top of BBISS, we develop a safety information sharing application to actually enable users to utilize the service. The paper reports the prototype design and the implementation of the application and future study issues.*

Keywords: Ad hoc communication, Safety information, Application

1 Introduction

After a large-scale disaster happens, we wish to confirm the safety of their family and friends. In the case where we are in a remote area from the disaster areas, we may try to obtain the safety information by telephones or e-mails that enable us to connect to people in the disaster areas. In addition, the services: Message dial 171 [1] or Web 171 [2] are provided in Japan. The services require communication infrastructures, such as public telephone networks, mobile networks, internet connections, and servers, to be available. In the disaster areas, since the communication infrastructures may be unavailable due to physical damages, congestions, or power failures, it is difficult for us to obtain the safety information from the services. In the case, we have no other ways except for posting at disaster shelters to share the information [3]. Since the way cannot share the information quickly and correctly, it may cause a spread of false rumors in the areas.

We have proposed Broadcast Based Information Sharing System, BBISS, which does not require the existing infrastructure [4]. The system requires only broadcast communication functions of IEEE802.11 wireless LAN in ad hoc mode at mobile terminals. To actually make people to make the most of BBISS, we develop a safety information sharing application on the top of BBISS.

This paper is organized as follows. Section 2 describes existing safety information sharing ways and their problems. Section 3 and Section 4 describe outlines and a way of implementation of our safety information sharing application, respectively. Section 5 explains user tests of the application. Lastly, Section 6 concludes the study.

2 Current safety information sharing ways and problems

Today, we usually confirm the safety of our family and friends through the telephone. In Japan, telephone operators recommend to use message dial services or internet message board services because telephony is often out-of-service due to traffic congestion. In addition, internet applications: twitter [4] and SNS, telephone applications: LINE [5] and Skype [6] are coming to be used. Although the systems are useful outside the disaster areas, it may be difficult to use the systems in the disaster areas damaged seriously. It is because the communication infrastructure may be out of order due to power failure or physical damage. Even if we have radios or handy televisions instead of cell phones, we cannot obtain local information of the disaster areas and safety information on our family and our friends.

To share the safety information in the disaster areas seriously damaged, we have only one way to put up posters on the bulletin boards at disaster shelters. In the posters, names and pictures of the victims, contact information, names of shelters, and others are shown.

Below, advantages and disadvantages of the safety information sharing using the message dial services, using the internet message board services for the disasters, and using the bulletin boards at the disaster shelters.

The message dials/the internet message board services for the disasters:

- Though television programs and advertisements encourage using them, many people do not use them [3].
- We cannot use them when the communication infrastructures are unavailable.

The bulletin boards at the disaster shelters:

- No difficulties. We can describe freely and easily.
- It is difficult for us to find the necessary information for us among many patches of paper.
- It takes much time if we have to share information at other shelters.
- If we cannot get to the evacuation shelters, we cannot post and share the safety information. Therefore, it may not possible to share the information by immobile victims.

3 Safety information sharing application

Considering the problems mentioned in Section 2, we develop the safety information sharing application on BBISS.

3.1 Assumed environment

The application assumes a following environment. A Large-scale disaster happens and disables public telephone networks, mobile networks and internet accesses. Although the users in the area may have smartphones, tablet PCs, or laptop PCs, they have no available IP addresses and gateway information after the internet connectivity is lost. In addition, existing ad hoc network routing protocols are usually not installed and unavailable. Only broadcast communication of IEEE802.11 wireless LAN ad hoc mode is available in the terminals. Hence, the application must assume that the users share the safety information by mobile devices without communication infrastructures and servers.

3.2 BBISS and safety information sharing application

To distribute information to all terminals in the area without ad hoc network routing protocols, Simple Flooding (SF) and other improved flooding methods have been proposed. Since the methods are usually used to transfer control messages in the routing protocols, the information that should be sent must be containable in just one packet. When the information cannot be contained in one packet, the information must be divided into multiple packets. Therefore, in order for the information to be reassembled correctly, all the divided packets must be received successfully at an information receiving node.

To satisfy the above requirements, we have already proposed a novel information delivery method, BBISS and shown its effectiveness. As shown in Fig.1, to enable the people to actually take advantage of the BBISS, we develop a safety information sharing application on the top of BBISS. The application makes it possible for the users to transmit safety information to neighboring nodes (users). By repetition of the operation, the information is delivered all over the area.

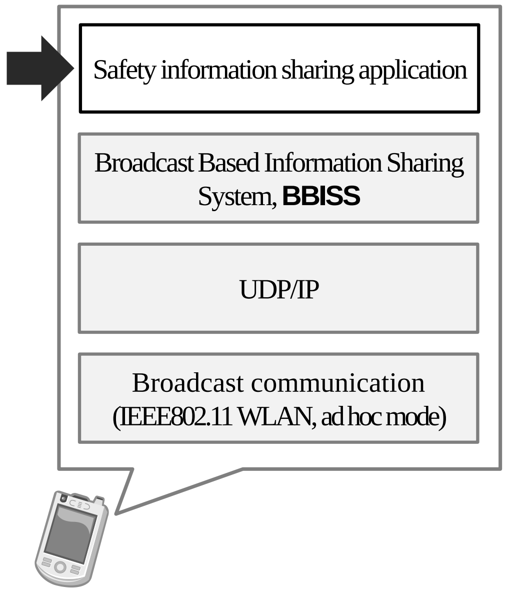

Safety information sharing application
Broadcast Based Information Sharing System, BBISS
UDP/IP
Broadcast communication (IEEE802.11 WLAN, ad hoc mode)

Fig. 1 Implementation of the safety information sharing application

3.3 Expected effect of the application

As mentioned in Sections 1 and 2, at this point in time, we have no other ways except for the posting at the disaster shelters to share the safety information so that we have to suffer inconvenient and slow information sharing. To make matters worse, people outside the disaster area cannot recognize the information submitted in the infrastructure unavailable areas. Our application is expected to have the following effects compared with the information sharing by the posters on the bulletin boards.

a. Making it possible to share the information rapidly

In the disaster area, using the bulletin boards, it is difficult to gather together in a large number of safety information. Since it is necessary to gather the information and paste them on the bulletin boards by someone’s hand work, it is not possible to share the information rapidly.

On the other hand, the application makes it possible to share the information rapidly, because the safety information is digitized, delivered, and shared rapidly in the area.

b. Improving the accessibility

To submit and share the safety information using the bulletin boards, the users have to write the information on paper, get to the bulletin board and paste or confirm the information. Therefore, it is difficult for aged people and injured people to submit and share the information. On the other hand, since the application enables them to submit and share the information at the place where they are, the application can contribute to the improvement of usability.

In Japan, the other problem is that, people from abroad, who are not familiar with Japanese, tend to be information refugees [10][11]. They cannot understand the safety information which is written in Japanese on the bulletin board. Furthermore, it is difficult for them to submit their safety information as Japanese understandable. Therefore, to enable them to share the information easily, the application should support multilingual navigation.

c. Making it possible to cooperate with the area outside the disaster area

In the case where internet connection is recovered or the case where internet accessible terminals appear in the area, the information should be submitted to the Internet. The function is our future issue to be tackled.

3.4 Functions and requirements of the application

The application is composed of the submission function and the view function. The submission function needs to be inputted user's name, day of birth, and safety state as required information, and some message as optional information. The view function is required for users to be able to view the received information from other terminals (users). Assuming the disaster happened and the infrastructures are unavailable, the application should be usable for every smartphone or tablet PC user, regardless of the age or the native language. The following explains the requirements to the application.

a. Simple design

Assuming to be used by various people in confusion after a disaster, the application should have a simple and clear design. Moreover, the application should able to be operated without instructions or helps.

b. Multilingual navigation

The displayed language should be switchable to English, Chinese, or Japanese, for the user needs.

c. Multiple submissions

Assuming the case where there is a person who submit the information, we may have to submit his information on the behalf of him. Therefore, the application has to be able to submit the information in addition to oneself.

4 Implementation of the safety information sharing application

4.1 Information sharing system

BBISS is an information sharing system that runs on top of the broadcast communication on the ad-hoc mode using IEEE802.11 series wireless LAN. In this paper, we implement an information sharing system that simulated the BBISS on the UDP / IP by socket programming on Windows PCs. The specifications of the PCs are shown in Table 1 and 2. The suppressive function of redundant relay of information and the retransmission function of unreached packets are omitted to simplify the implementation. Then, we implement the safety information sharing application on the above environment.

Table 1 Spec for the PCs in the implementation (laptop PCs)

Type	ASUS X301A-RX
CPU	Intel Pentium B970, 2.3GHz, 2 cores
RAM	4GB
OS	Windows 7 Home Premium 64bit
WLAN	Logitec LAN-W150N/U2 (IEEE802.11b)

Table 2 Spec fot the PCs in the implentation (tablet PCs)

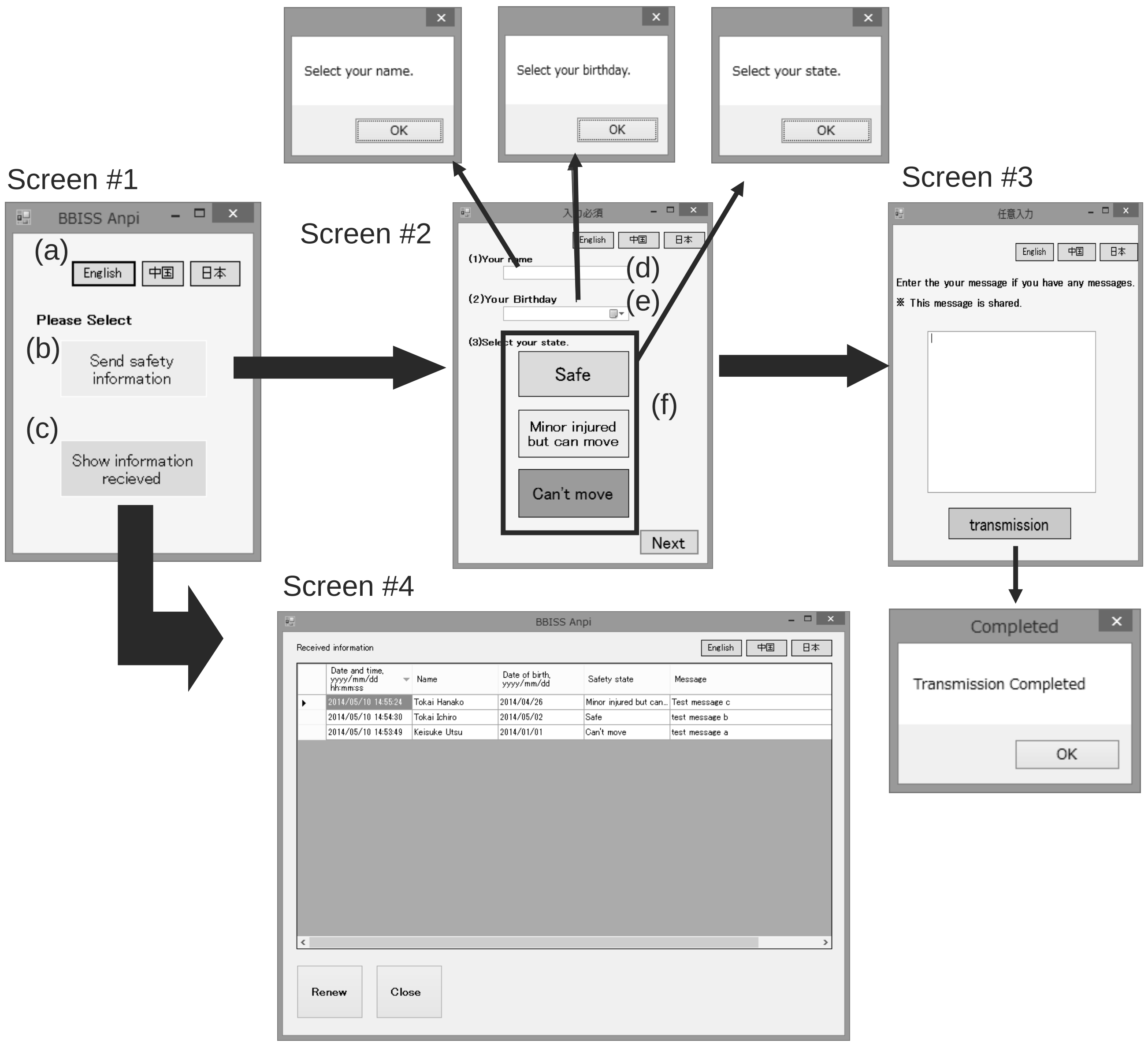
Type	ASUS VivoTab Smart ME400C
CPU	Intel Atom Z2760, 1.8GHz, 2 cores
RAM	2GB
OS	Windows 8
WLAN	Broadcom 802.11bgn SDIO (IEEE802.11b)

4.2 Screen design of application

The screen design of the application is shown in Fig.2. The application is composed mainly of four screens (Screens #1~#4). The contents of each screen, the principal screen components, and the operating procedure are shown below.

Screen #1：Initial screen

(a) Language selection buttons



Select your name.

Select your birthday.

Select your state.

Enter the your message if you have any messages.
※ This message is shared.

Transmission Completed

BBISS Anpi

Received information

	Date and time, yyyy/mm/dd hh:mm:ss	Name	Date of birth, yyyy/mm/dd	Safety state	Message
▶	2014/05/10 14:55:24	Tokai Hanako	2014/04/26	Minor injured but can...	Test message c
	2014/05/10 14:54:30	Tokai Ichiro	2014/05/02	Safe	test message b
	2014/05/10 14:53:49	Keisuke Utsu	2014/01/01	Can't move	test message a

Fig.2 Screen transition of safety information sharing application

The buttons are for switching the displayed language to Japanese, English, or Chinese.

(b) "Send the safety information" button

The button is for sending the safety information at Screen #2.

(c) “Display the received information” button

The button is for displaying the safety information received from the surrounding terminals at Screen #4.

Screen #2 : Screen to send safety information (Required inputs)

(d) Name input box

The user’s name is input to the box. The information is used for user identification.

(e) Date of birth selection box

The date of birth of the user is selected from the calendar. The information is used for user identification.

(f) Safety state selection buttons

The user's safety state is selected from the three buttons. The user’s safety state to be selected from three: "Safe", "Minor injured but can move", and "Can’t move", where each of the state has each color as red, yellow, and green(blue) , respectively which is same as traffic signals. If “Next” button is pressed, the screen transits to Screen #3. However, if the user’s name is not input, or either safety state or date of birth is not selected, the screen transits to the error screen when the "Next" button is pressed.

Screen #3 : screen to send safety information (Arbitrary input)

If necessary, the user can input any message in the input box. It is possible to enter his location, phone numbers, email addresses, and the like. After “Submit” button is pressed, input information is transmitted by BBISS to the surrounding terminals, and is displayed "transmission completed".

5 Evaluation of the safety information sharing application

To show the effectiveness of the application, we took a questionnaire survey. The candidates for the survey were 15 students who were not concerned the development of the application at Tokai University. The PCs showed in Table 1 are used. The survey date was Wednesday, April 30, 2014.

Impression about the application:

Q.1: Is the screen of the application clear?

Yes: 15 of 15

No: 0 of 15

Q.2: Do you operate it intuitively?

Yes: 10 of 15

No: 5 of 15

Q.3: For Q.2: what is the difficulty in the use?

- To input the date of birth is difficult: 9 of 15
- To recognize whether the state button is pressed or not: 1 of 15
- To understand what the date should be input (the date of today or the user’s birthday), at the calendar input box: 1 of 15
- No answer: 4 of 15

The function should be installed in future:

The survey gave the following opinions about the function that should be installed in future.

- The date of birth selection box should be simplified.
- An entry example of the name should be displayed.
- If the rescue list with only users who selected "can’t move" was outputted, it would be more useful.
- The safety information to be displayed should be updated in real time.

To select the safety state was intuitive and easy to understand for the users because the state buttons were simple color: green, yellow, and red. However, it was difficult for the users to select their date of birth because it was selected from the calendar box, which is implemented for the simplification. The box should be pull-down menus in the future.

In future, we plan to install the following functions. The application assumed to be used by general people. Assuming the local government’s use, the function to output a list of users who selected "can’t move" may be useful. Furthermore, in addition to safety state and text information, the function to submit pictures may be helpful in rescue activities and recognizing damage situations.

Fig.3 Implementation of the application (on laptop PCs)

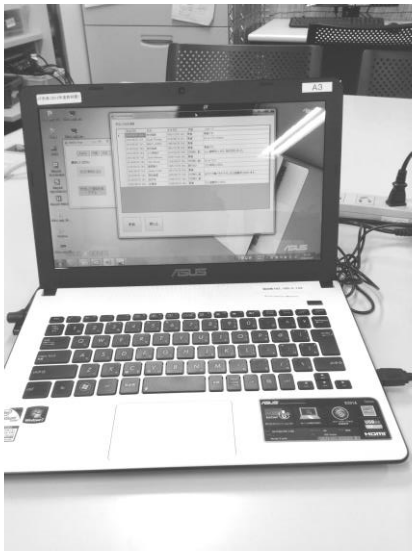


Fig.4 Implementation of the application (on laptop PCs)

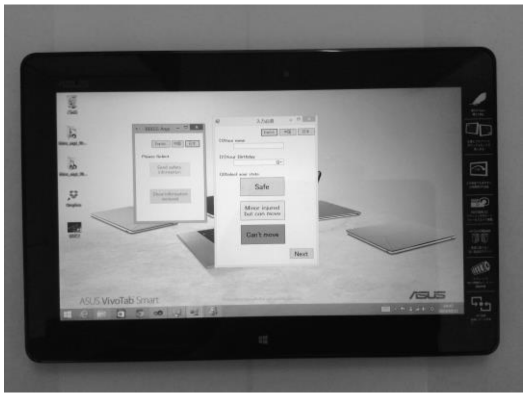


Fig.5 Implementation of the application (on tablet PCs)

6 Conclusions

We have studied the safety information sharing application without using existing communication infrastructure for the case where the infrastructures are out of order due to the large-scale disaster happenings. In this paper, we designed and developed the safety information sharing application on top of BBISS, and implemented it on the laptop PCs. Then we experienced the application and took the questionnaire survey to test subjects. The evaluation results showed the effectiveness of the application and the functions should be installed in future.

In future, we plan to improve the application based on the obtained comments on the evaluation. In addition, in this study, the application was implemented on the laptop PCs, therefore we plan to implement the application on general portable information terminals, and carry out a practical test cooperating with local governments.

Acknowledgments

This study has been supported by the following foundations.

- The Telecommunications Advancement Foundation, Japan
- The Science Research Promotion Fund, The Promotion and Mutual Aid Corporation for Private Schools, Japan

References

[1] Message Dial 171, NTT East Corporation, https://www.ntt-east.co.jp/saigai/voice171/

[2] Web 171, NTT East Corporation, http://www.ntt-east.co.jp/saigai/web171/

[3] Mobile Society Research Institute, NTT DOCOMO, "Mobile Communication 2012-13", 2012

[4] Keisuke Utsu, Hiroshi Sano, and Hiroshi Ishii, "A Proposal on Broadcast Information Sharing System for Infrastructure Unavailable Situations", The Papers of Technical Meeting on "Communications", pp.1-6, CMN-14-001, IEE Japan, in Japanese

[5] Keisuke Utsu, Sayuri Wada, and Hiroshi Ishii, "Performance Evaluation on Infrastructure-less Broadcast Based Information Sharing System, BBISS", The Papers of Technical Meeting on "Communications", pp.7-12, CMN-14-002, IEE Japan

[6] Yuta Hirano, Hiroaki Matsumoto, Keisuke Utsu, Hiroshi Sano, and Hiroshi Ishii, "A Proposal of Message Sharing Application in Disaster Situation on BBISS", The Papers of Technical Meeting on "Communications", pp.13-17, CMN-14-003, IEE Japan

[7] Twitter, Twitter Inc., http://twitter.com

[8] LINE, Line Corporation, http://line.me/

[9] Skype, Microsoft, http://www.skype.com

[10] Ministry of Load, infrastructure, Transport and Tourism, Japan, http://www.mlit.go.jp/common/001000495.pdf

[11] Bureau of Citizens and Cultural Affairs, http://www.seikatubunka.metro.tokyo.jp/index3files/hokokuH240424.pdf

Location Data Supplementing Information Transfer Method over MANET

Kei Kobayashi[1], **Yosuke Totani**[2], **Hiroshi Sano**[2], **Keisuke Utsu**[2], **and Hiroshi Ishii**[2]
[1]Graduate School of Information and Telecommunication Engineering, Tokai University, Minato, Tokyo, Japan
[2]School of Information and Telecommunication Engineering, Tokai University, Minato, Tokyo, Japan

Abstract - *A Location aided Ad Hoc Network is one of the effective information transfer methods in the disaster situation. This method requires the GPS (Global Positioning System) to obtain location information. However, nodes cannot necessarily obtain their own location information in the disaster situations because GPS function of mobile devices are usually fully supported by infrastructure (network provider) and may not work in such situation. In addition, there may be many nodes that do not have GPS devices. Hence, the number of nodes are limited that can participate in above mentioned location aided MANET (Mobile Ad Hoc Network). In this paper, we propose a method enabling GPS unavailable nodes to join the network by supplementing their own location data from GPS available nodes. In addition, we make preliminary evaluation of the method and show its effectiveness.*

Keywords: Ad hoc network, Greedy forwarding, Global positioning system

1 Introduction

MANET (Mobile Ad-hoc Network) has been studied as a technology for building infrastructure-independent and autonomously distributed controlled network in the disaster situation. Some kinds of MANET have there are methods to efficient communication by using the location information obtained by GPS.

However, since this location information aided method assumes that all nodes must be able to obtain the location information by use of GPS, nodes that cannot obtain the location information cannot participate in the network. GPS that runs on the popular device can be classified into two types. One type is supported by positioning satellites and the other is supported by mobile communication infrastructure. The latter may become unusable if the infrastructure becomes out of order in the disaster situation. In addition, there are too many nodes that do not have GPS. Namely, assuming the disaster situation, nodes that can obtain the location information and those that are considered to be mixed in the network area.

We propose a method to make nodes that cannot obtain location information participate in the location-aided network by giving location information to those nodes from nodes that can obtain location information.

Section 2 describes the greedy forwarding method that is a basis of our proposal and our proposal itself. In section 3, we evaluate the proposed method through network simulation and show its effectiveness. Section 4 concludes this paper.

2 Delivery method

This section firstly shows the greedy forwarding method as a respresentative of conventional location information aided approach. Then it proposes a location data supplementing information transfer method

2.1 Greedy forwarding

The greedy forwarding method is one of the methods used in location aided MANETs. In the case of general information transfer methods of MANET without location data, a source node broadcasts data packets to neighboring nodes and then all nodes that receive the packet relay the packet by broadcasting. This method is often called "simple flooding". In contrast, greedy forwarding enables directional communication to the destination by selecting one of the neighbor nodes as a next hop node by using the location information. In this greedy forwarding method, nodes periodically exchange their own location information got by GPS and IDs among neighbor nodes by HELLO packets. By use of exchanged location information, each data transmission node before sending can select the next hop node according to the criteria whether the node is the closest to the destination or the transmission node can forward the packet by the biggest distance to the node to be selected The node selected as the next hop node performs the same procedure as the prior transmission node that results in realization of efficient communication.

2.2 Proposed method

The location-aided MANET assumes that all the nodes can obtain their own location information by GPS. However, GPS that runs on the popular device can be classified into two types. Some positioning satellites support one type and the other is supported by mobile communication infrastructure.

The latter may become unusable in the disaster situation because infrastructure becomes out of order. In addition, there are a number of nodes that do not have GPS. For these reasons, it may not be effective in case of using the conventional location aided approach based on, e.g., greedy forwarding.

Therefore, we propose a method that enables nodes cannot obtain location information to participate in the location-aided MANET by giving location information from the nodes that can obtain location information. Same as the greedy forwarding method, our proposal utilizes HELLO packets from GPS available neighbors and collects location information in the Hello packets. By the collected information, GPS unavailable node can guess its own location and participate in the network with guessed location.

We describe the method below. In our proposed method, we call the node LS-node (Location information Server node) that can obtain location information and the NL-node (Non Location information node) that cannot obtain location information. The method consists of two phases that are the preparation phase and the transmission phase.

2.2.1 Preparation phase

In this phase, each node exchanges information by HELLO packets and calculate evaluation values of themselves. This self-evaluation value is used for whether NL-nodes can participate in the location-aided MANET based on the proposed method or not, and it is also used as materials for determining next hop node after nodes participate in the network. This self-evaluation consists of the evaluations of location information and movement.

The evaluation of location information consists of the calculation of the location coordinate and assessment of the accuracy of the location coordinate. And the evaluation of movement the calculation of the velocity of each node, which is used to avoid selecting the node that moves fast as a next hop node. The details of the preparation phase are as follows.

i. Obtaining location information

Each node obtains location information. LS-node obtains the location information by GPS that is equipped in the nodes. NL-node collects the location information in the HELLO packets from the neighboring LS-nodes, estimates its own location and then treats the estimated location information as its own.

Each node judges the location coordinate and assesses its. In this paper, the location information contains two factors. One is the position coordinate (hereinafter referred to as o^i) itself and its accuracy (hereinafter referred to as s^i). o^i is an essential element to participate in location-aided MANET because each node selects the next hop node based on this information. On the other hand, s^i is important for NL-nodes. In the case of NL-nodes, the positional error may occur because NL-nodes estimate the location information based on the location coordinates collected from neighbor LS-nodes. Therefore, it is necessary to assess the accuracy of the estimated location coordinate. And s^i is the value that assesses the accuracy. Calculation methods of assessment are given below. Each LS-node uses the position coordinates that obtain from its own GPS as o^i. In addition, the value of s^i is set to the highest value because the positional error hardly arises. NL-node by use of the location coordinates got from the neighbor LS-nodes draws circles that imitate the transmission range whose center is the location of each LS-node by and treats the center of the overlapping area in circles as the position coordinates. Moreover, each NL-node calculates the size of the overlapped area and treats it as s^i to estimate the accuracy of the coordinates. The calculation method is changed depending on how many numbers of circles make the overlap. However, the number of circles to make the overlap may become enormous. Hence, in this paper, we restrict the number of circles in the case of 1 - 3. Each NL-node calculates only three recent location information if the numbers of circles becomes more than four. The number of circles is decided by how many HELLO packets are received. In the case that s^i is 0 (there is no overlapping) or πr^2 (multiple circles coincide together) we give the case the lowest values of evaluation because it is assumed that it doesn't fulfill as the accuracy of the positional coordinates. It is described below how to calculate the overlap area for the cases where two circles form the overlap and more than three circles do.

When there are two LS nodes nearby a NL node, there is an overlapped area in the two circles. In this case, the center of the straight line that links the intersection points of two circles is set as a provisional position coordinate. And then the overlapped area is calculated and it is treated as the accuracy of the position coordinates.

When three nearby nodes form the duplicated area, the area that three circles make can divide into three cases.

Case 1 In the case that two circles duplicate perfectly among three circles, it is regarded that two circles duplicated are a single circle and then calculation of overlapped area with other circle just like the case of two circle overlapping. Fig.1 shows an example of this case 1.

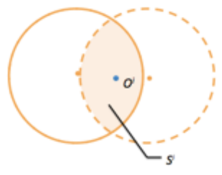


Fig 1. An image of case 1

Case 2 In the case that one of three circles contains the overlapped area that other two circles make, the circle containing the overlapped area is not required to be considered, therefore calculation of the overlapped area of remaining two is made. Fig.2 is an example of this case 2.

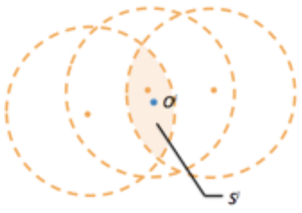


Fig 2. An image of case 2

Case3 In the case that any circle does not completely contain the overlapped area that other two circles make. The provisional coordinate is that of the circumventer of the triangle that is made by lines between intersection points of three. See Fig.3.

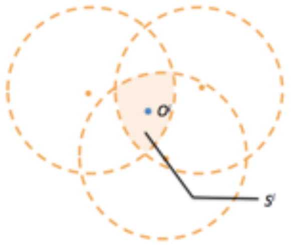


Fig 3. An image of case 3

ii. Judgment of location and its assessment on the accuracy

Each node calculates its own velocity, m^i. Each node, whichever it is LS or NL, obtains own location information multiple times and then calculates own velocity by using those differences of coordinates and that of time when calculation is made.

iii. Calculation of node velocity, m^i

Each node calculate E_t^i by using s^i and mi. E_t^i to be used for each node to make a decision to participate in the network and a judgment of the appropriateness to be selected as a next hop node. The formula to calculate E_t^i is following.

【The formula to evaluate each node】

$$E_t^i = f(s^i, m^i)$$

The value of E_t^i is decided according to Table 1. The smaller s^i and m^i are, the higher the value of E_t^i is. It is because the smaller the movement of each node is, the higher the evaluation of each node is.

Table 1. The value of E_t^i

$s^i \diagdown m^i$	$0 < m^i \leq 1$	$1 < m^i \leq 5$	$5 < m^i \leq 10$
$s^i = 0$	0	0	0
$0 < s^i \leq \pi r^2/4$	9	7	5
$0 < s^i \leq \pi r^2/2$	8	6	4
$0 < s^i \leq \pi r^2$	7	5	3
$\pi r^2 \leq s^i$	0	0	0

r : the radius of the radio coverage of each node

iv. Calculation of E_t^i

After each NL-node is permitted to participate in the network in the step iv, each NL-node judges whether it can treat itself as a LS-node or not. This judgment is made by use of severer condition. We assume the positional error with GPS should not exceed 50m and so decide the condition to be a LS-node accordingly. The positional error must be less than $\pi r^2/25$ (radius = 250m).

v. Judgment to be an LS node

Each node notifications the values calculated in above steps (position coordinate, node ID, and E_t^i) by using HELLO packet. The payload of the HELLO packet is following.

- Node ID
- Location information
- Evaluation of the nodes (E_t^i)

vi. Notification of node ID, (provisional) coordinate and E_t^i by Hello

Each node performs above steps at a certain interval in the preparation phase. LS-nodes perform above processing when each LS-node just obtained the location information by GPS. And NL-nodes perform it when each NL-node just obtains more than two HELLO packets from LS-nodes within last 2 seconds.

2.2.2 Transmission phase

When each node transfers data packets, it transits to the transmission phase. While each node does not need data transfer, it remains in a preparation phase. The transmission node considers value of E_t^i of neighbor node i notified by HELLO packets and di that is a distance from the neighbor node i to the destination node, and determines whether the node i to be selected as a feasible next hop node. After determining the next hop node, the transmission node sends a data packet to the node. Repeating this procedure, packets can be reached to the destination node. The detail of transmission phase processing is as follows.

i. The transmission node receives E_t^i and location data of neighbor node i by HELLO packets. The HELLO packet is the same as packets that each node broadcasts in step vi of preparation phase. The value of di, distance from node i to the destination is calculated here. Here di is not an actual distance but a normalized value form 0 to 10 according to Fig.4.

ii. The transmission node adds E_t^i and di and calculates E^i of the transmission evaluation of node i.

【The transmission evaluation formula】

$$E^i = E_t^i + d^i$$

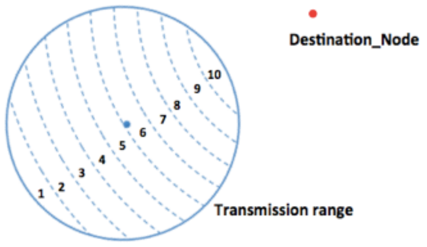


Fig 4. Evaluation value of d^i

iii. The transmission (or relay) node selects the node with the best E_t^i value in the communication area as a next hop node.

iv. The relay node broadcasts the packet containing a node ID of the next hop node.

v. The nodes that received the data packet check the node ID contained in the data packet. And then, if the node ID coincides with its own node ID, it finds it is selected as a next hop relay node. And it performs same procedure shown above, selects the next hop node, and transmits the data packet to the next hop node. If the node ID is not congruous with its own node ID, it discards the packet. These operations are repeated until the data reaches the destination.

3 Evaluation

We evaluated the proposal method by comparison with existing greedy forwarding algorithm using the simulator created by the script language. The simulation conditions are shown in Table 2.

Table 2. Simulation condition

Configuration	
Simulated area	1000×1000m
Setup of nodes	
Coverage	250m
Number of nodes	100
Default LS-node ratio (Nodes other than LS-nodes are NL-nodes)	1~100%
Moving velocity (Random way point *)	1 to 10 [m/s]
The participating conditions to a network	
Qualification to participate	$E_t^i \neq 0$
Threshold to be LS-node	$0 < s^i <= \pi r^2/25$
The selection condition of a node	
Select the best result of the following calculation $E^i = E_t^i + d^i$	

*【**Random way point model**】

i. The destination coordinate is defined at random.

ii. The speed is decided at random within a range from 1 to 10 [m/s].

iii. A node moves at a fixed speed selected in ii to the destination.

iv. It stops for random time at the destination coordinate..

v. Back to i

3.1 Contents of evaluation

The items evaluated in the simulation are shown below.

I. Node participation ratio in the network and number of NL-nodes treated as LS-nodes

How many NL-nodes can participate in the network and how many NL-nodes can be treated LS-node

II. Reachability

Among randomly positioned nodes, a source node and a destination node are selected, between which a multi-hop communication is made. The reachability is defined as the ratio of packets that reach the destination successfully to total packets sent averaged over all the trials of simulation.

III. Average number of hops

How many hops are required to reach destination node averaged for all the packets. It is calculated to know whether we can reduce the number of hops by introducing our idea. The evaluation method for average number of hops is defined as the total number of hops divided by the total number of times of reaching.

3.2 Results of the simulation

I. Node participation ratio in the network and number of NL-nodes treated as LS-nodes.

Figure 5 shows the number of nodes that is able to participate in the network and the number of nodes that become to be LS-nodes for the case where the number of LS-nodes is 15 among 100 nodes in the network. Vertical axis indicates the number of nodes and horizontal axis indicates the simulated time. The top most line shows the summation of the numbers of LS-nodes and nodes that could participate in the network. According to the figure 5, at an average of 59.5% of NL-nodes during simulated times (60 seconds) becomes possible to participate in the network. And we show that it is 1.23% NL-nodes can be treated as LS-nodes at an average during simulated time.

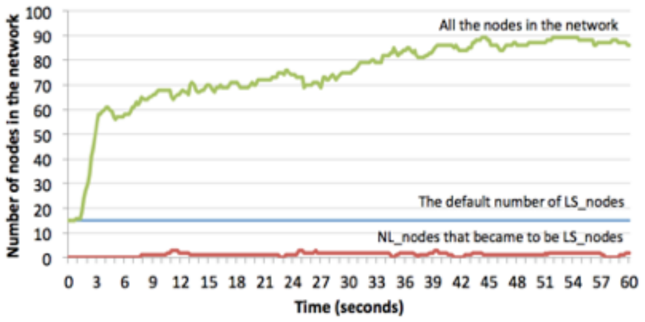


Fig 5. Participation ratio of nodes

According to the results, it is clear that the number of nodes that can participate in the network increases by applying the proposal method. Therefore, we can expect to raise the reachability and to reduce the average number of hops, which are shown below.

II. Reachability

Figure 6 shows the reachability. The vertical axis indicates the reachability and horizontal axis indicates the number of LS-nodes. As a result, our proposal can achieve high reachability (successful communication) even if the number of LS-node is small. When the number of LS-node is only 20, our proposal can achieve 80% reachability but on the contrary the conventional greedy method cannot at all. It shows our proposal is highly effective. This is because our proposal can make the NL-nodes participate in the network that results in high reachability.

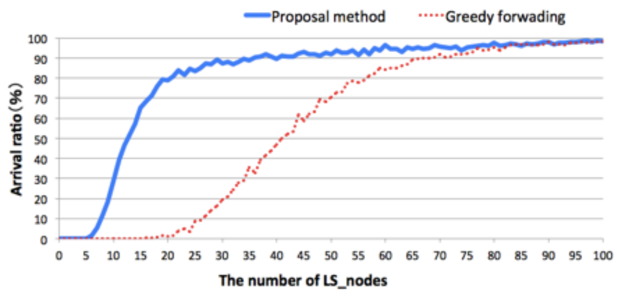


Fig 6. Reachability

III. Average number of hops

Fig.7 shows the average number of hops when the packet reaches the destination. The vertical axis indicates average number of hops and the horizontal axis indicates the original number of the LS-nodes in the network area. Our proposal achieves that it is less hops than that of greedy forwarding. Below 10 LS-nodes, greedy cannot realize any communication. This result means that the transmission node became able to select the best next hop node from among the many candidates by applying the proposed method.

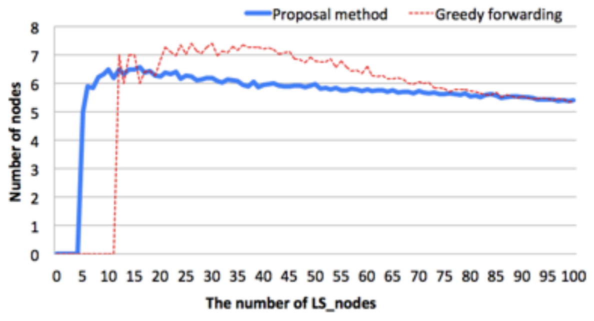


Fig 7. Average number of hops

4 Conclusions

In this paper, we have proposed a new algorithm to enable nodes that do not obtain the location information by themselves to participate in the location-aided MANET using the location information given by other nodes that can obtain location. Through the simulation study, we have shown that our proposal can improve packet reachability drastically compared with existing greedy algorithm. Further study is needed on evaluation when node mobility is considered.

5 Acknowledgement

This research is partly conducted as JSPS Grants-in-Aid for Scientific Research 26430372.

6 References

[1] Kei kobayashi, Yosuke Totani, Hiroshi Sano, Keisuke Utsu, Hiroshi Ishii: "Location Data Supplementing Information Transfer Method over MANET", CMN-14-005, pp.25-28 (2014)

[2] B.Karp and H.Kung: "GPSR: Greedy Perimeter Stateless Routing for Wireless Networks" (2000)

[3] Kenichi Mase, Shiro Sakata: "Ad-Hoc Networks and Meth Networks", Vol. 1 pp.43-52 (2007)

[4] Hiroshi Nakagawa, Kazuyuki Nakamaru, Tomoyuki Ohta, Yoshiaki Kakuda: "A Hybrid Greedy Routing with Location Information for Mobile Ad Hoc Networks" (2008)

[5] Naohide Fukushi, Keisuke Utsu, Hiroshi Ishii: A study on the Method of Information Discovery using GPS over Mobile Ad-hoc Network

A Proposal on Location-aided Route Discovery based on Two-hop Neighbor Information over Ad Hoc Network and its Preliminary Evaluation

Phonepadith Phoummavong[†], **Keisuke Utsu**[‡] **and Hiroshi Ishii**[*]
[†] Graduate School of Science and Technology, Tokai University, Tokyo, Japan
[‡,*] Sch. of Information and Telecommunication Engineering, Tokai University, Tokyo, Japan
[†]phonepadith_pp@live.com, [‡]utsu@utsuken.net, *ishii@ishiilab.net

Abstract - *How to efficiently reduce the number of packets of broadcasting in Mobile ad hoc network (MANET) is very important inevitability for saving energy and prolonging life time. Most of previous researches based on broadcast approaches do not welll consider avoiding redundant packets. However route discovery from a source to specific destination based on broadcasting causes large increase of unnecessary packets and affects the scalability. According to this research we propose two approaches, the first approach is the use of the connected dominating set (CDS) utilizing information of two-hop neighbors in ad hoc network, where searching minimum number of neighbors. The Second is based on Location-Aided Routing Protocol (LAR) or Geocast that is powerful to find specific area where destination node is. Eventually based on two base algorithms above, in this research we implement our algorithm (Location-aided Route Discovery based on two-hop neighbor information) and analysis via relationship between CDS and LAR. Finally, we find the effective factors to reduce the number of overhead packets and number of redundant rebroadcasting by our algorithm to precisely select the forwarding node. We compare our algorithm with the Direct Flooding and Geocast through computer simulation. The simulation result shows that our proposal decrease the number of overhead packets in Ad Hoc network on broadcast compared with existing algorithms.*

Keywords: Direct-Flooding, Geocast, Tow-hop, Route Discovery.

1 Introduction

A mobile ad hoc network (MANET) has been widely used to support communication for such as terms of military, education, medical and emergency. Normally a mobile ad hoc network is easily deployed because it is combined by a group of independent mobile nodes without infrastructure. Consequently a mobile ad hoc network can normally operates in an emergency situation via broadcasting a message or information to the other user for requesting collaboration.

The simplest flooding algorithm has been already known to cause the broadcast storm problem [1], [2]. Basically, in the flooding operation, source (*S*) sends the message to all the neighbors and then every neighbor will forward the packets until reaching the destination (*D*). But the problem is a time for holding the communication, that is, a batteries of nodes are consumed much and lifetimes of nodes and network are very limited. Hence, reducing number of packets is significant factor where decelerating the lifetime of node. Recently there are various algorithms of broadcast operations to reduce unnecessary packets which are show below.

First category, the CDS [3],[4] is utilizing information of two-hop neighbor nodes, in short *S* exchanges information among itself and neighbors. The neighbor who has minimum number of neighbors is selected as the next sender before packets are transmitted. Basically, this research, is calculating connected dominating set by using marking process. However, since the CDS algorithm just finds the minimum connected dominating set for each node, it is possible for *S* cannot find position of *D*.

Second category is a local broadcast algorithm in wireless ad hoc network reducing the number of transmission [5] and broadcast redundancy in ad hoc wireless network [6]. In additional, both researches [5] and [6] are extended from the *Dominant Pruning* (DP) and *Distributed Dominant Pruning* [4] which are local broadcast algorithms that can eliminates redundant transmission based on two-hop information for neighbors and then find minimum number of forwarding nodes. However [5] and [6] are different objective from our own proposal. Because they just show that they decrease the number of retransmission node by coverage transmission range without specified location of *D* so the consumption of overhead packets will also increase.

For third category is the direct flooding described in [7] Beacon-less routing (BLR). The author proposes the routing of packet to reduce routing overhead. BLR does not require neighbor nodes to broadcast Hello-message (Beacon Message) and avoids drawback such as extensive uses of scarce battery-power. BLR has good performance to guarantee that packets will reach to *D.* In contrast the consumption of overhead packets stills increases because *S* selects all neighbor nodes located in determined area neighbor nodes are possible to forward packets together.

As for fourth category, several geographical routing protocols (LAR) by use of global positioning system (GPS) or we call *Geocast*, are proposed in [8], [9]. In this algorithm, the source *S* is using distance determine by coordinate *cov*(*x*,*y*) and compare distance between each neighbor node to *D* in the communication area. And therefore *S* can selects a neighbor node to forward packet which has distance closest to *D*. The operation of *Geocast* is better than BLR in a sense that *S* specifies a location of *D* and can reduce the number of transmission node. But the consumption of packets still increases because *S* selects neighbor node without considering the number of two-hop of neighbor information.

In this paper, we first follow the CDS method to calculate the minimum number of neighbor nodes. Second, we follow the fourth category above to estimate distance between neighbor nodes and *D*. In this regard, we can establish the probability by relationship between CDS and Geocast. By use of the probability that is the main mechanism of our proposal, *S* precisely selects neighbor node who forwards the packet.

This paper will be organized as follows. Section 2 reviews related works. In section 3, we describe our own proposal. In section 4 we present an evaluation. Finally conclude this paper.

2 Related Work

In this section, we describe the previous works of the broadcast solution and problem.

2.1 Flooding algorithm

The flooding algorithm [1] is a simplest operation by directly forwarding packets to all the neighbor nodes in the network. So this approach has many problems such as the consumption of packets, the redundancy of transmission node and the collision of packets.

2.2 Direct flooding algorithm (Beacon-less routing)

Basic principle of this approach BLR [7] is fixable than Flooding algorithm. See Fig. 1. When *S* sends a packet to *D*, as first step, *S* determines the position of *D* and stores its current position (coordinates *X*, *Y*), and then *S* broadcasts a packet by attaching information (coordinates of *S* itself and *D*) into the header of the packet. Identically each node that receives the packet replaces the previous position by their current its own position in to the header of the packet before forwarding packet to the next hop. Subsequently if a packet reaches neighbor node, the position information of the previous node will be extracted from the header of packet. Therefore neighbor of *S* easily decides to relay the packet or not. The node will relay the packet if the node is located in the specific area (forwarding area shown in gray). Since the BLR avoids any beaconing mechanism such as Hellos, it is not necessary to know the neighbor node information. But the number of packets is increasing because multiple nodes relay the packets and redundant packets exists in the forwarding area.

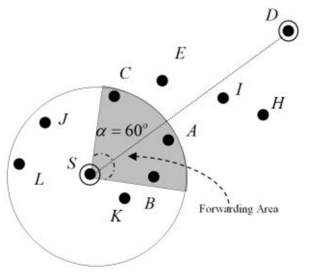


Fig. 1. Direct flooding Route Dicovery Scheme.

2.3 Geocast(Location-Base Multicast Algorithms)

Y. Ko, and H. Vaidya proposed Geocast LBM [10] similar to LAR [11] by considering the GPS (global positioning system). Fig. 2 shows Geocast. Geocast assumes each node broadcasts his own position on Hello packets periodically so each node can know its neighbors' position (coordinate). When *S* initially forwards a packet, this mechanism follows three steps: first step, the multicast region is specified by *S*. Second step, *S* calculates $DIST_S$ (distance between *S* and *D*). Third step, *S* it compares distance from each neighbor node to *D*. For example, node *S* has neighbors *I*, *N* and *K*. If $DIST_s \geq DIST_i$, node *i* (*i*=*I*, *N* or *K*) is a candidate forward packet. Else if $DIST_D < DIST_i$, node *i* (*I*, *N*, *K*) is not a candidate to forward packet. *S* selects a node (e.g., node I) as next forwarding node, if the node has the shortest *DIST* to the destination (or multicast region). Thus node *I* replaces the coordinate (X_S,Y_S) by its own coordinate (X_i,Y_i). Likewise, the operation is iterated until reaching the region.

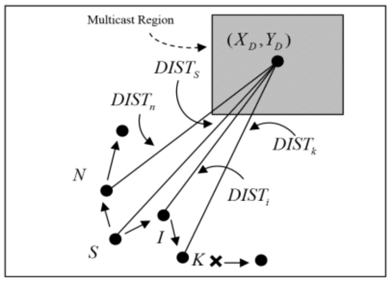


Fig. 2. Location-Base Multicast Scheme.

3 Our Proposal Methods

In this section, we propose a broadcast scheme algorithms to reduce the number of redundant packets by improving the efficiency of CDS and Geocast ILAR [12].

The main advantage of CDS is that it centralizes the whole network into small connected dominating set sub network, which means only a gateway keeps routing information, Hence as long as network topology change does not affect this sub network, there is no need to recalculate routing table.

An improved location-aided routing (ILAR) is improved from LAR. In this scheme, the author first decides a baseline, which is the line between node *S* and *D*, for route discovery. When the request packet is broadcasted, a node in request zone based on baseline is chosen as the next broadcasting node.

According to existing algorithms above, we propose three route discovery methods to find relay node. First method considers the minimum average distance between two-hop neighbor of *S* and *D*. The second method considers the probability by relation between distance of neighbor of *S* to *D* and the number of neighbor node of *S*. The third method considers the probability by relation between distance of two-hop neighbor of *S* and the number of two-hop neighbor of *S*.

3.1 Method 1

Based on communication by Hello message exchange, CDS can determine an information of one-hop (v_i) and two-hop neighbor (v'_i) of *S*. Ore specifically, while node *S* is crossing message to exchange information and coordinate with neighbors, *S* also collects *id* of neighbor node by receiving short packet message ("hello" message) and position of neighbor nodes including two-hop neighbor information. It can selects the best of neighbor node by processing the header of packet before broadcasting initial packet. By this way we can calculate the average distance from two-hop neighbor of *S* to *D*.

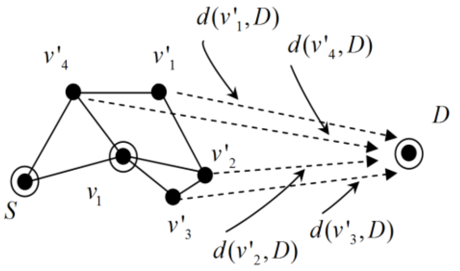


Fig. 3. Method 1 minimum average distance two-hop

To calculate the average distance from two-hop neighbor of *S*, we define $d(v'_1, D)\dots d(v'_4, D)$ that are distances between node v'_i and *D*. Then we can find the average distance from neighbors of node *S* to *D* in the equation below.

$$d_{avg}(v_i) = \frac{d(v'_1,D)+d(v'_2,D)+\ldots+d(v'_i,D)}{n(v_i)} \qquad (1)$$

Eq. 1. where $d_{avg}(v_i)$ represents the average distance for neighbors of sender, then $n(v_i)$ is the number of neighbors of neighbor (v_i) but except the sender (previous node) as shown in Fig. 3. (e.g. node v_1 does not include node *S* for calculating average distance because node *S* is sender). Finally *S* decides a neighbor node as a next sender and sent RREQ (Route REQuest) which has minimum average distance in Eq. 2.

$$Select = \min_{\forall i} \{d_{avg}(v_i)\} \qquad (2)$$

3.2 Method 2

Second method, we consider the probability by relation between distance of neighbors of *S* to *D* and the number of neighbor node of *S*. Same as first method, a sender collects two-hop neighbor information but it is different from first method in the fact *S* collects distances from one-hop neighbors (v_i) to *D* shown as Fig. 4.

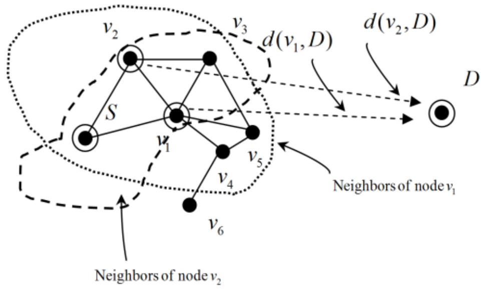


Fig. 4. Method 2 finding probability from distance two-hop

Fig. 4. shows both of node v_1 and v_2 are neighbor of *S* and they have neighbor nodes such as $N(v_1)=\{S, v_2, v_3, v_4, v_5\}$, $N(v_2)=\{S, v_1, v_3\}$ and $d(v_1,D)$, $d(v_2,D)$ are distances between node v_1, v_2 to *D*. Therefore in Eq. 3, we find the probability (distance between node v_i and *D*) divided by the summation of $d(v_1,D)$, $d(v_2,D) \dots d(v_i,D)$ (total of distance neighbor nodes of *S* to *D*). Absolutely if a neighbor node of *S* is closest to *D*, it will get high probability.

$$p_{avg}(v_i) = 1 - \frac{d(v_i,D)}{d(v_1,D)+d(v_2,D)+\ldots+d(v_i,D)} \qquad (3)$$

As a next step we find the probability by considering the number of neighbors of each neighbor node of v_i. In Eq. 4, $n(v_i)$ is the number of neighbors of neighbor v_i that is two-hop neighbors of the sender (In Fig. 4, i.e. $n(v_1) = 5$ and $n(v_2) = 3$). $n(v_i)$ is divided by summation of all number of two-hop neighbors of *S*. So if neighbor of *S* (e.g., v_i) has minimum neighbors, it will get the high probability.

$$p_n(v_i) = 1 - \frac{n(v_i)}{n(v_1)+n(v_2)+\ldots+n(v_i)} \qquad (4)$$

Finally, from Eq. 3 and 4 we get the probability that is a combination of the probabilities of distance and the probability regarding the neighbor node v_i.

$$\Pr(v_i) = \begin{cases} p_{avg}(v_i) + p_n(v_i) - p_{avg}(v_i)p_n(v_i); \text{ if } p_{avg}(v_i) > p_n(v_i) \\ p_{avg}(v_i)p_n(v_i) \qquad ; \text{ if } p_{avg}(v_i) < p_n(v_i) \end{cases} \quad (5)$$

Here, the probability Pr(v_i) is relationship between distance node v_i to D and number of neighbor node v_i. Subsequently node S selects next neighbor node given by probability Pr(v_i). This method can avoid the consumption of overhead packet because S can precisely select a neighbor node to forward packet.

3.3 Method 3

As for the third method, we consider the probability by relation between distance of two-hop neighbor of S and the number of two-hop neighbor of S, In Fig.5. Like operation in [12], we assume that how S selects two-hop neighbor for calculating the average distance. For node v_1, we define the baseline 1 that connects node v_1 and D. Next the baseline 2 crosses over on baseline 1 by angle 90^o degree. Subsequently, two-hop neighbor located on the right hand side of baseline 2 is selected by S for calculation $d_{avg}(N(v_i))$ such as v'_3, v'_4, v'_5 but for node v_2 is not selected because it is located on the left hand side, where $d_{avg}(N(v_i))$ means average distance from neighbors of v_i, i.e., two hop neighbors of S to D.

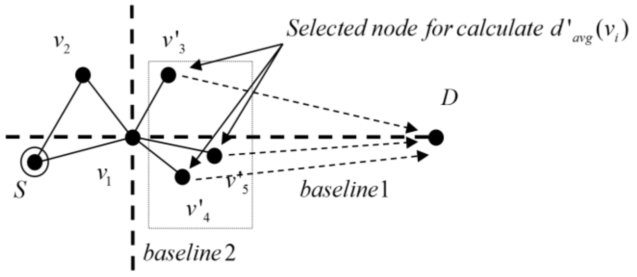


Fig. 5. Method 3 Sender collects information from node A and B

Before S initially broadcasts a packet, it collects neighbor node information from node A and B same as first method and second method i.e. in figure 6 (a) $N(A)$ ={ E, I, F}, in Fig. 6. (b) $N(B)$={F,G,H}. Then we denote $d_{avg}(N(A))$ and $d_{avg}(N(B))$ represent average distance from two-hop neighbors to D.

Next we will find the average distance of two-hop neighbors to D assuming we already know the coordinates of two-hop neighbor nodes and destination node i.e. node C: $d(C, D)$, E: $d(E, D)$, K: $d(K, D)$ and I: $d(I, D)$. Then we determine the average distance of two- hop neighbors by Eq.6.

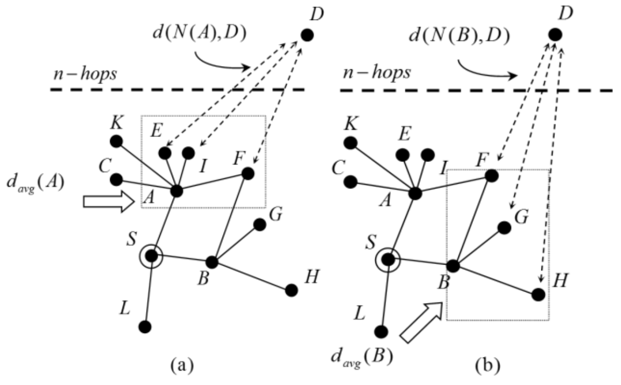


Fig. 6. Method 3 Sender collects information from node A and B

$$d'_{avg}(v_i) = \frac{d(v'_1,D) + d(v'_2,D) + ... + d(v'_i,D)}{n_{select}(v_i)} \quad (6)$$

From Eq.6 we find average distance, where $d(v'_1, D)$, $d(v'_2, D)$.. $d(v'_i, D)$ are distance from two-hop neighbor of S to D and $n_{select}(v_i)$ is number of neighbor nodes of v_i that is selected as neighbor nodes same as Fig.4. Next step we define Δd_{avg} in Eq.7 that represents the reference value or the total average distance of neighbor node of v_i. Here, $n(S)$ is a number of neighbor nodes S.

$$\Delta d_{avg} = \frac{d'_{avg}(v_1) + d'_{avg}(v_2) + ... + d'_{avg}(v_i)}{n(S)} \quad (7)$$

In Eq.8 we calculate Δn_{avg} is mean reference value or total average number of two-hop neighbors of S. where $n(v_i)$ is the number of neighbor nodes v_i , and $n(S)$ is the number of neighbor nodes S. Then S estimates the total average neighbor nodes by the relation distance two-hop neighbor to D and the number of neighbor node.

$$\Delta n_{avg} = \frac{n(v_1) + n(v_2) + ... + n(v_i)}{n(S)} \quad (8)$$

Then we apply the exponential probability function [13] to each method by use of Δd_{avg}, $d'_{avg}(v_i)$, $n_N(v_i)$ and Δn_{avg} to select the possible neighbor node to forward packet.

- Case 1: If $d'_{avg}(v_i) > \Delta d_{avg}$ and $n_N(v_i) > \Delta n_{avg}$

$$\Pr(v_i) = 1 - \frac{d'_{avg}(v_i)}{Max(d'_{avg}(v_i))} e^{-\left(\frac{\Delta n_{avg}}{n_N(v_i)}\right)} \quad (9)$$

- Case 2: If $d'_{avg}(v_i) < \Delta d_{avg}$ and $n_N(v_i) > \Delta n_{avg}$

$$\Pr(v_i) = 1 - \left(\frac{d'_{avg}(v_i)}{Max(d'_{avg}(v_i))} \right) \left(\frac{d'_{avg}(v_i)}{\Delta d_{avg}} \right) e^{-\left(\frac{\Delta n_{avg}}{n_N(v_i)} \right)} \quad (10)$$

- Case 3: If $d'_{avg}(v_i) > \Delta d_{avg}$ and $n_N(v_i) < \Delta n_{avg}$

$$\Pr(v_i) = 1 - \left(\frac{d'_{avg}(v_i)}{Max(d'_{avg}(v_i))} \right) \left(\frac{\Delta d_{avg}}{d'_{avg}(v_i)} \right) e^{-\left(\frac{n_N(v_i)}{\Delta n_{avg}} \right)} \quad (11)$$

- Case 4: If $d'_{avg}(v_i) < \Delta d_{avg}$ and $n_N(v_i) < \Delta n_{avg}$

$$\Pr(v_i) = 1 - \left(\frac{d'_{avg}(v_i)}{\Delta d_{avg}} \right) e^{-\left(\frac{n_N(v_i)}{\Delta n_{avg}} \right)} \quad (12)$$

Equation from Eq.9 to Eq.12 represent the four cases by function of probability, where Pr (v_i) is combined by relation distance and number of neighbor node. First Case $d'_{avg}(v_i) > \Delta d_{avg}$, $n_N(v_i) > \Delta n_{avg}$ means that node v_i has average distance larger than total average distance and number of neighbor has larger than total number average of two-hop neighbor. So for this case, the probability is very low. Therewith for second case, $d'_{avg}(v_i) < \Delta d_{avg}$, $n_N(v_i) > \Delta n_{avg}$ means that node v_i has average distance smaller than total average distance and number of neighbor has larger than total average number of neighbor two-hop. So for this case, the chance of probability has higher than case 1. Similarly third case, $d'_{avg}(v_i) > \Delta d_{avg}$, $n_N(v_i) < \Delta n_{avg}$ is same as case 1 but the number of neighbor has smaller than total number average of two-hop neighbor wherefore the chance of probability in case 3 also has value higher than case 1. Thereupon fourth case $d'_{avg}(v_i) < \Delta d_{avg}$, $n_N(v_i) < \Delta n_{avg}$ means that when both of a average distance and number of neighbor have smaller value than total average distance and total number average of two-hop neighbor. So the probability is highest. Therefore S selects node v_i which has the highest probability to broadcast packet.

4 Evaluation

4.1 Assumption and scenario geographic

We propose two assumptions. The first is that node S transmits packet to D by maximum distance for 1 time slot via broadcast. In addition we assume S is located in around south west corner and D is located in around north east corner. Then position of each node is set according to random uniform distribution in area 600x600 m^2 and the radio transmission range is 150 m. Similarly the Second case is that we assume S transmits packet to D but the initial position of S is randomly selected for 1000 time slots. Based on two assumptions above, we set four patterns of network topology for simulation such as pattern 1 for 50 nodes in Fig. 7. pattern 2 for 100 nodes in Fig. 8. pattern 3 for 150 nodes in Fig. 9. pattern 4 for 200 nodes in Fig. 10.

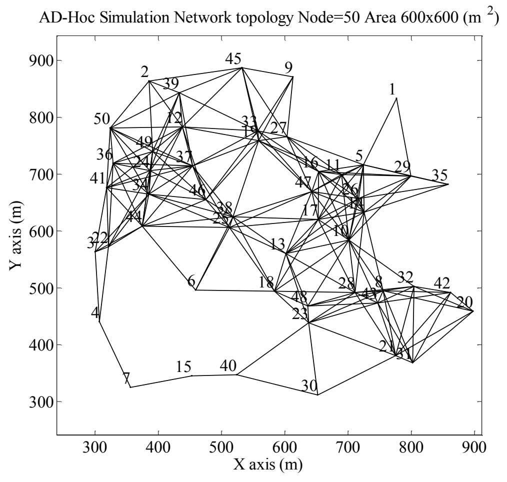

AD-Hoc Simulation Network topology Node=50 Area 600x600 (m 2)
Y axis (m)
X axis (m)

Fig. 7. pattern 1 for 50 nodes

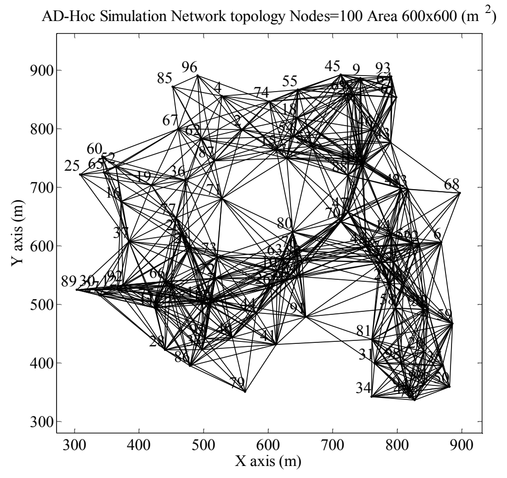

AD-Hoc Simulation Network topology Nodes=100 Area 600x600 (m 2)
Y axis (m)
X axis (m)

Fig. 8. pattern 2 for 100 nodes

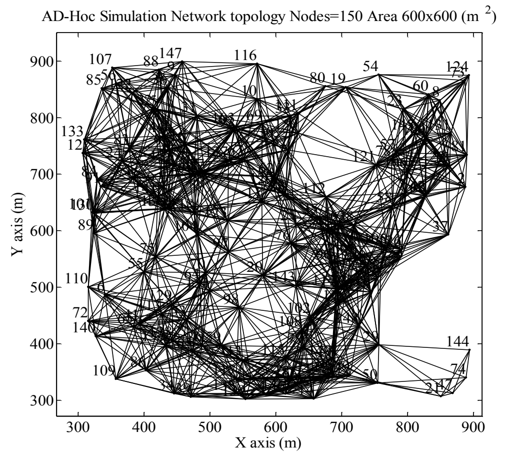

AD-Hoc Simulation Network topology Nodes=150 Area 600x600 (m 2)
Y axis (m)
X axis (m)

Fig. 9. pattern 3 for 150 nodes

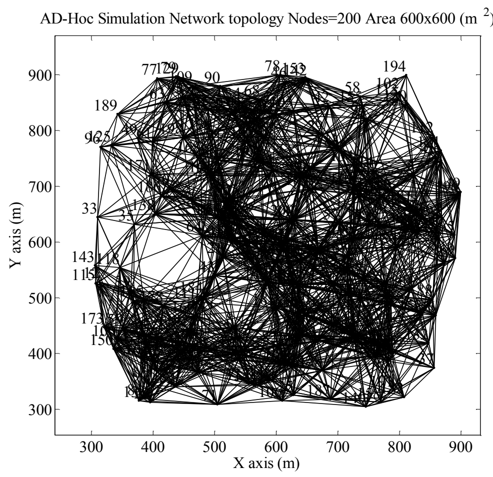

AD-Hoc Simulation Network topology Nodes=200 Area 600x600 (m 2)
Y axis (m)
X axis (m)

Fig. 10. pattern 4 for 200 nodes

4.2 Results

In this path we present some result of this research, first path shows the number of overhead packets in one time slot and 1000 time slots by varying the number of nodes from 50 to 200. Second path shows the average number of nodes rebroadcasting in one time slot and 1000 time slots by varying the number of nodes from 50 to 200.

Fig. 11. shows the average overhead of direct-flooding (α=60^{o} in Fig.1.), Geocast, our own proposed method 1, 2 and 3 within different density of mobile nodes when *S* and *D* are maximum distance in 1 time slot. The number of overhead (packets) of direct-flooding is very large because the consumption is made via large number of transmission node. Next the Geocast is better than Direct-flooding. In contrast for our proposal method 1, 2 and 3 the overhead of packets are increasing very slowly and the control of overhead packet is better than two previous algorithms. When we are varying the number of total nodes to 200, in our proposal method 1, the number of overhead packets is larger than Geocast because method 1 does not precisely selects the next node. Anyway for method 2 and 3, overhead are smaller in any node density than method 1 and the other approaches.

Fig. 12. shows the average overhead of direct-flooding, Geocast, Own propose method 1, 2 and 3 with different density of mobile node when *S* is chosen randomly and it sends RREQ packet (random sender) to *D* in 1000 time slots. Then the average overhead of direct-flooding is larger because *S* selects all neighbors in forwarding area for sending RREQ packet that is traditional flooding algorithm. Likely the reason from above, Geocast is better than direct-flooding but the number of overhead packets are large than our proposal method 1, 2 and 3. Since we are varying the number of mobile nodes to 200 thereupon the method 1 has the consumption of overhead packets larger than proposal method 2 and 3.

Fig. 13. shows the number of transmission node or hop count with different density of mobile node for 1 time slot (maximum distance). The number of transmission nodes of Direct-flooding is larger than Geocast because direct-flooding selects all neighbors to in forwarding area to send RREQ packet. Since Geocast reduces the number of nodes compared with direct-flooding but the number of relay nodes for Geocast is large because packets are broadcasted within multicast region. Anyway for our proposal, three methods are very small compared with two previous algorithms because *S* initially broadcasts packet only one node selected to forward RREQ packet.

Fig. 14. Shown that average number of transmission node or hop counts with different density of mobile node for 1000 times slot (Random Sender). The reason is same as that given above, the number transmission node of Direct-flooding and Geocast are larger than our proposals.

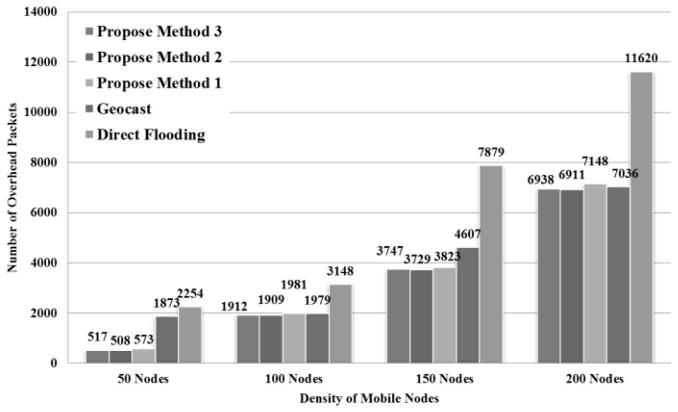


Fig. 11. Number of Packet VS Density of mobile nodes (Maximum distance)

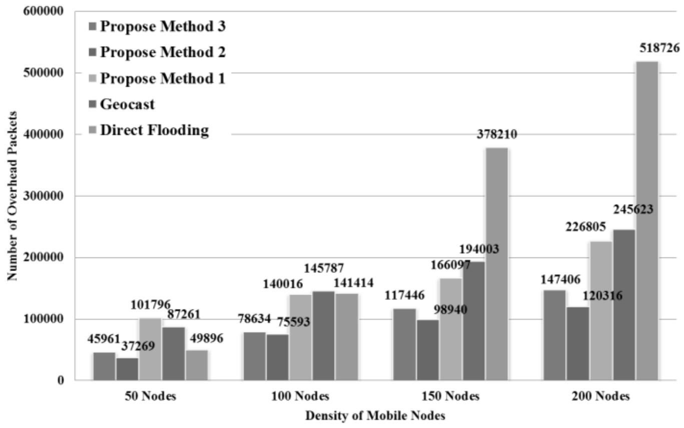


Fig. 12. Number of Packet VS Density of mobile nodes (Random sender)

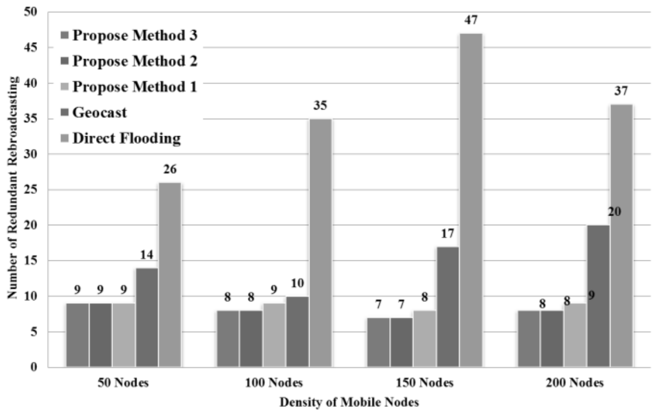


Fig. 13. Number of redundant rebroadcasting VS Density of mobile nodes (Maximum distance)

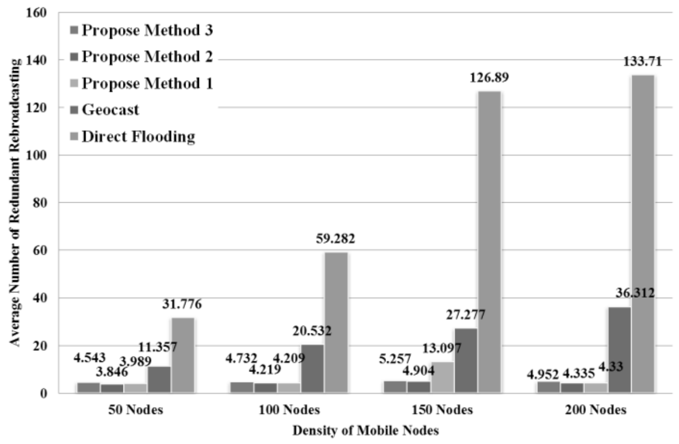


Fig. 14. Average Number of redundant rebroadcasting VS Density of mobile nodes (Random Sender)

5 Conclusion

In this paper, we address the problem of efficient data transmission in mobile ad hoc network. Constantly selecting the best node according to the probability makes conventional routing discovery have lowest overhead packet. Facing the fact that selection of the best next hop is very difficult in existing geographic broadcasting mechanism, we propose Location-aided route discovery based on two-hop neighbor information over Ad Hoc Network inspired by geographic routing. For all the simulation result, it shows how precisely our proposals select the forward nodes and reduces redundant packets which has good results even in case of varying density of node and network topology. So our proposal method offers adaptability which improves the performance by simulation in term the number of overhead packet and number of transmission node comparing with direct-flooding and Geocast. Finally we confirm the performance and efficiency of our proposal is better than existing methods. We need future study on evaluation more in detail by varing network topology or considering mobility.

6 Acknowledgement

This work was partly supported by JSPS KAKENHI Grant Number 26420372.

7 References

[1] S. Y. Ni, Y. C. Tseng, Y. S. Chen and J. P. Sheu, “The Broadcast Strom in a Mobile Ad Hoc Network,” Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking, ACM New York, USA, pp. 151–162, 1999.

[2] S. Misra, I. Woungang and S. C. Misra, “Guide to Wireless ad Hoc Networks”, British Library Cataloguing in Publication Data, Springer-Verlag London Limited, 2009.

[3] J. Wu and H. Li, “On Calculating Connected Dominating Set for Efficient Routing in Ad Hoc Wireless Networks,” Proceedings of the 3rd international workshop on Discrete algorithms and methods for mobile computing and communications, ACM New York, USA, pp. 7-14 1999.

[4] F. Dai and J. Wu, “Distributed Dominant Pruning In Ad Hoc Networks,” Communications, ICC '03. IEEE International Conference Vol. 1, pp. 353-357, May. 2003.

[5] M. Khabbazian, L. F. Blake and V. K. Bhargava, “Local Broadcast Algorithm in Wireless Ad Hoc Networks: Reducing the Number of Transmissions,” Mobile Computing, IEEE Transactions, Vol. 11, Issue. 3, pp. 402-413, March. 2012.

[6] W. Lou and J. Wu, “On Reducing Broadcast Redundancy in Ad Hoc Wireless Network,” System Sciences, Proceedings of the 36th Annual Hawaii International Conference, January. 2003.

[7] M. Heissenbuttel, T. Braun, T. Bernoulli and M. Walchli, “BLR: beacon-less routing algorithm for mobile ad hoc networks,” Elsevier’s Computer Communications Journal, pp. 1076-1086, 2004.

[8] C. Maihofer, “A Survey of Geocast Routing Protocols,” IEEE Communication Surveys and Tutorial, Vol. 6, Issue. 2, pp. 32-42, 2004.

[9] H. Frey, “Scalable Geographic Routing Algorithm for Wireless Ad Hoc Networks,” Network, IEEE, Vol. 18, Issue. 4, pp. 18-22, 2004.

[10] Y. B. Ko, and N. H. Vaidya, “Geocasting in Mobile Ad hoc Networks: location-based Multicast Algorithm,” Mobile Computing Systems and Applications, WMCSA '99. Second IEEE, pp. 101-110, 1999.

[11] M. A. Zayene and N. Tabbane “Performance Evaluation of Location-Aided Routing Protocols in Ad Hoc Networks,” Information Infrastructure Symposium, GIIS, pp. 1-6, June. 2009.

[12] N. C. Wang and S. M. Wang “An Efficient Location-Aided Routing Protocol for Mobile Ad Hoc Networks,” Parallel and Distributed Systems, Proceedings. 11th International Conference, Vol. 1, pp. 335-341, July. 2005.

[13] M. L. Shooman “Reliability of Computer Systems and Networks: Fault Tolerance Analysis, and Design” John Wiley & Sons, 2002.

A Proposal on Efficient Broadcast Based Information Transfer Method Using Location Data over MANET

Yosuke Totani[1], **Keisuke Utsu**[1], **and Hiroshi Ishii**[1]
[1]School of Information and Telecommunication Engineering, Tokai University, Minato, Tokyo, Japan

Abstract – *In the disaster situation, mobile ad-hoc networks can be considered as one of the communication means when communication infrastructure is unavailable. Under the situation, a number of redundant packets with power consumption should be suppressed to save battery and prolong the network life. In this paper, we propose an efficient broadcast based information transfer method using location data to reduce the redundant packets with achieving high data reachability.*

Keywords: Ad Hoc Network, Broadcast

1 Introduction

Information technology has rapidly been spread in Japanese society. The Internet, a portable information device and something like that are the necessaries for our lives. However, communication infrastructure must be essential to these technologies. In case of emergency, the communication infrastructure may become unable to handle a large amount of access and base stations may be physically disrupted. When communication infrastructure is unavailable, we cannot transmit information if you know where you want to send to, because telecommunication cannot be used without IP address in such a case. Urgent information such as situation of damage and confirmation of someone's safety should be distributed without waiting for the recovery of communication infrastructure.

Recently, wireless ad hoc network has been actively studied as a communication means when communication infrastructure is unavailable. The wireless ad hoc network can be immediately constructed by only gathering information terminals in some region. In addition, the network constructed with only mobile information terminal that can communicate with each other is called mobile ad hoc network (MANET).

When communication infrastructure is unavailable, studies on simple flooding (SF) [1] and directional flooding [2] have been proposed as delivery systems of information. However, both of which have problem of power consumption due to redundant packets and that of information loss due to packet collision before packets reach the destination.

In this paper, we propose information transfer method over MANET using location data without depending on communication infrastructures. Proposed method is aiming to reduce redundant packets and to achieve high data reachability at the stage of not being able to use the IP address just after the disaster. We show that proposed method is more efficient than existing methods through evaluation using a simple network simulator.

Section 2 describes algorithms and problems of existing methods. Section 3 explains the proposed method. And then, section 4 shows effectiveness by results of simple network simulator. Finally, section 5 concludes this study.

2 Existing methods

Routing protocol over MANET is mainly classified into two types. One is topological-based routing. The other is location-based routing. The latter routing is widely proposed to solve a problem of scalability against topological-based routing such as AODV (Ad hoc On-Demand Distance Vector) [3] or OLSR (Optimized Link State Routing Protocol) [4]. And furthermore, location-based routing has two approaches of next-hop forwarding method and directional flooding method.

In the next-hop forwarding method, each node periodically sends ID and location data of itself in a packet to neighborhood nodes. Referring to the neighborhood location data, a node that relays the packet selects a next relaying node which has the biggest forwarding distance from the relay node to the end node or which is the closest to the end node. This method to select a next relaying node is called greedy forwarding [5].

In the directional flooding method, neighborhood nodes broadcast one after another toward the end node. At this time, each node rebroadcasts the packets without specifying next hop nodes. A node that receives the packet, however, does not rebroadcast it without any qualification but does only when needed based on the location data of a start node, a node just before or the end node. Moreover in directional flooding, a node that receives the packet independently of other nodes decides whether to rebroadcast or not just based on geographical conditions. More specifically, a "transmission zone" between a starting node (or a relaying node just before) and the end node is defined, and simple flooding is executed only inside the zone. When a node receives a packet, it does or does not rebroadcast the packet depending on whether it is inside the zone or not.

However, these location-based routings generate a large number of packets because each node informs its own location data to each other among neighbors that results in large power consumption. As for the directional flooding, it is not clear how to define appropriate transmission zone. On the other hand, when communication infrastructure is unavailable in the disaster situation, it is concerned that topological-based routing cannot be used due to presupposition that IP address must be assigned to each node. Therefore, we aims at realizing broadcast-based data transmission with less or without HELLOs and without definition of transmission zone. Below, we firstly consider broadcast by simple flooding as the easiest communication means over MANET to compare with our proposal.

2.1 Simple Flooding

Simple flooding (SF) is widely used as a classic information delivery method of the broadcast. A basic algorithm of SF is as follows.

1. An initiator node creates a message and broadcasts a packet containing the message around each nodes within one-hop radio area.

2. Each node that receives a packet rebroadcasts it in a same manner as step 1 if it receives the packet for the first time. While the packet has been already, the node discards the packet.

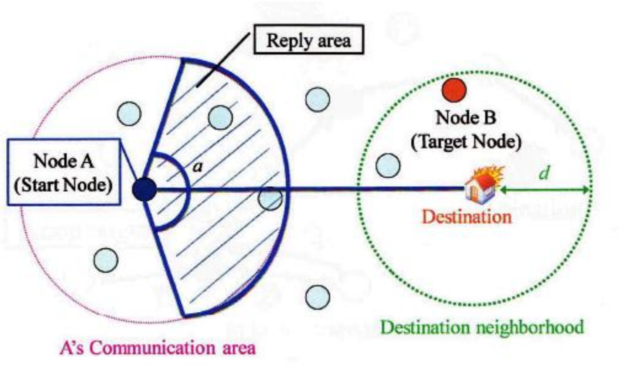
SF does not require assigning IP address to each node, and is an information delivery method in order to distribute information to all nodes in the network. Therefore, if SF is used to send information to a specific node or an area, it generates many redundant packets in the directions other than that to a destination and causes more power consumption.

A next part explains an information discovery method related to a specific node based on the next-hop forwarding algorithm by applying greedy forwarding without using IP but with use of location data provided by GPS. And this method does not use Hello message explicitly to soften the problem of next-hop forwarding method as described below.

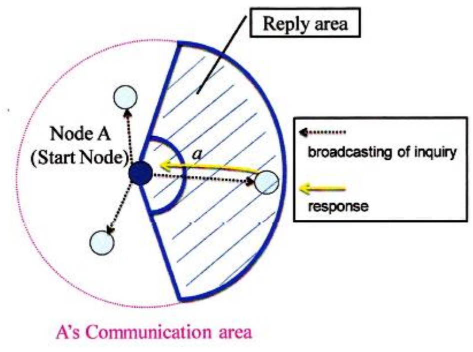
2.2 A method of information discovery using GPS over MANET [2]

As an existing method of information discovery using GPS over MANET, a method has been proposed that aims to reduce a redundant packets and power consumption in a case of not using IP address. The algorithm is as follows.

1 As shown in Fig. 1(a), the start node identifies the destination and the neighborhood of the destination. It finds its own location using GPS, and determines an angle "*a*" on either side of the line that connects itself with the destination. Within the one-hop radio, the area bound by angle "*a*" is called the reply area, and angle "*a*" is called the reply area angle.



(a) Step 1
(a) Outline of the method (Step 1)



(b) Step 2
(b) Outline of the method (Step 2)

Fig. 1 Outline of the method

2 As shown in Fig. 1(b), the start node broadcasts a packet that queries whether the recipient of the packet is the "target node" that holds information about the destination.

3 A node that has received a query packet, recognizes its own location using GPS, and

3.1 if it is the target node, it broadcasts a reply packet declaring that it is the target node;

3.2 if it is not the target node and is located within the reply area, it broadcasts a reply packet declaring that it is not the target node and also indicating its location coordinates.

4 If the start node receives a reply packet declaring that the sender is the target node, the start node requests it to send the information about the destination. The node sends the target information using an existing routing protocol, such as DSR (The Dynamic Source Routing) [6], with its location as the origin if IP addresses are already available.

In case IP address are unavailable yet, some other methods such as location aided routing or broadcast will be used. If the start node receives no reply packet declaring that the sender is the target node, it calculates the distance between each node from which it received a reply and the destination based on the location coordinates included in the reply packet, and sends a request packet (RQP) to the node nearest to the destination.

5 The node that has received the RQP repeats steps 2 through 4 successively, just as the start node.

This algorithm of method realizes suppressing redundant packets because of not broadcasting to each nodes except the destination node. In addition, it realize reducing the send and receive times of each nodes because of determination of the angle.

2.3 Problems of existing methods

In case that information is distributed not using IP address when communication infrastructure is unavailable, SF generates a lot of redundant packets in broadcasting to reach a particular node. Furthermore, routings such as a method of information discovery using GPS over MANET also generate a lot of redundant packets because each nodes exchange location data to each other. Moreover, local maximum problem may occur and cause the loss of packets due to selection of a node with the nearest distance to destination node as shown in Fig. 2 [5].

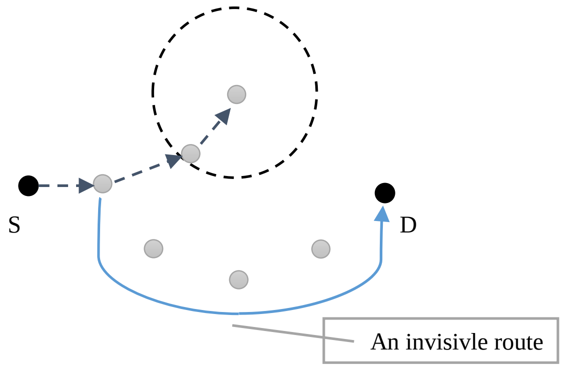


Fig. 2 Local maximum

3 Proposed method

Based on the existing methods and background above, we propose an efficient transfer method without using IP address. In the supposed environment, it is assumed that each node can obtain its own location data by using GPS and coordinate of destination area. In addition, it is also assumed that it is difficult to assign IP address to each node due to the physical disruption of communication infrastructure. Each node is a mobile information device and can broadcast data on the IEEE802.11 wireless LAN.

3.1 Requirements

The requirements for the method are as follows:

- It does not require routing methods based on IP address but only broadcasting. The idea comes from the problem that IP address assignment is difficult in case of infrastructure disruption including DNS service due to disaster.

- Each node transfers information to destination area with a light processing burden to suppress redundant flooding and to reduce redundant packets. A mobile information device should suppress power consumption as much as possible and reducing redundant packets directly lead to suppressing power consumption.

- Moreover, proposed method achieves high data reachability.

3.2 Algorithm for forwarding

1 Initiator node broadcasts a packet including location data of both itself and destination in addition to data.

2 A node that receives the packet calculates two distances by referring to location data of initiator node, destination node and itself. One is the distance between itself and the destination node. The other is that between the initiator node and the destination node. At this time, the node rebroadcasts if it is nearer to the destination than the initiator node, or the node does not rebroadcast if not. Furthermore, even if it is the case above, the node does not rebroadcast either in the case that the node receives multiple number of the same information packets within random time, before sending.

3 The step 1 and 2 are repeated until the destination node receives the packet.

4 If a destination node receives the same packets more than twice, the packet from second is discarded.

3.3 How to realize the requirements

Our proposal executes information transmission by only broadcasting, IP address is not needed. The number of redundant packets can be reduced because all nodes does not rebroadcast but some limited nodes rebroadcast. In addition, unlike the method that a node determines a relay node one by one such as next-hop forwarding method, high data reachability is expected to be achieved with reducing redundant packets because only limited number of node relays packets.

4 Evaluation

This section evaluates the proposed method in comparison with the existing method by use of simple network simulator that is written in a script language.

4.1 Simulation environment and conditions

In location-based routing, data reachability changes significantly depending on the number of nodes in the area.

The simulation conditions are shown in Table 1.

Number of nodes (*N*)	50, or 50～100
Trials	1000
Simulated area (map)	1000m × 600m
Radio coverage (radius)	200m
Reply area angle (*a*)	150°
Distance of destination	Middle, or long range

Therefore, the reachability evaluation is made for various cases of the number of nodes, i.e., from 50 to 100 nodes. And in the case of evaluation of total number of sent and received packets, number of nodes is fixed to 50 nodes. Each node is set in the area randomly and there is 1000 map patterns. The simulated area is 1000m x 600m and the radius of the radio coverage of each node is 200m. Reply area angle in Fig. 1 is 150 degrees because the existing study shows 150 degrees is optimum value. We prepare two patterns of distance from the initiator to the destination: middle range distance and long range. The middle range sets the initiator node on the center of the simulated area and the destination node on the edge of northeast of the simulated area. The long range sets the initiator node on the southwest and the destination node on the northeast. If each node receives the same packets less than 3 times within a given random time, each nodes can rebroadcast.

4.2 The contents of evaluation

The evaluation is made by comparison among SF (section 2.1), discovery (section 2.2) and our proposal for (a) the total number of sent and received packets and (b) packet reachability.

As for the comparison (a), all broadcasted packets and received packets in the network are counted in SF. The total number of packets including query packets, reply packets, request packets and data packets is counted for the discovery. All the broadcasted packets and received packets are counted for our proposal. It is desired to reduce as much as possible with keeping high data reachability. As for (b), we count probability of reaching the destination, i.e., how many times in 1000 times simulation sent packet is reached to the destination, by varying the density of the node (that is number of nodes) in the simulated area. It is desired that the probability of reaching the destination comes up to 100% as much as possible.

4.3 Simulation results and consideration

The Evaluation results are shown in Fig. 4 to Fig. 6. Fig. 4 and Fig. 5 show the evaluation results for (a) total number of sent and received packets averaged for 1000 trials of the map patterns. Fig. 6 shows probabilities of reaching the destination by varying node density from 50 nodes to 100 nodes averaged among 1000 times trials for each density.

As shown in Fig. 4, comparing the proposed method with the existing discovery and SF, we succeeded in reducing a number of redundant packets for middle range distance case, since it shows that total number of sent and received packets can be decreased. As shown in Fig. 5, we succeeded in reducing for the long range distance case as well. However, it must be avoided to decrease reachability by reducing total number of packets. As shown in Fig. 6, the proposed method maintains more high data reachability than the existing methods. From the above, it can be seen that proposed method is efficient.

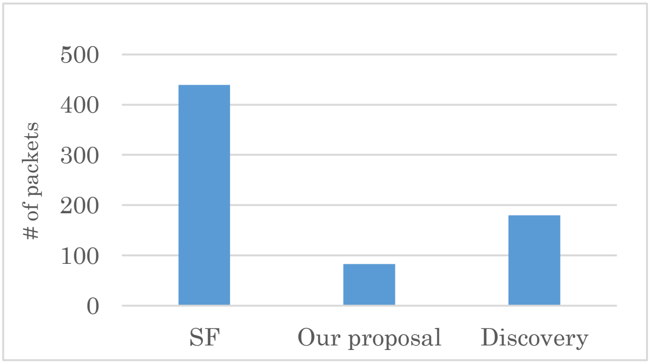


Fig. 4 Total num. of sent and received packets (middle)

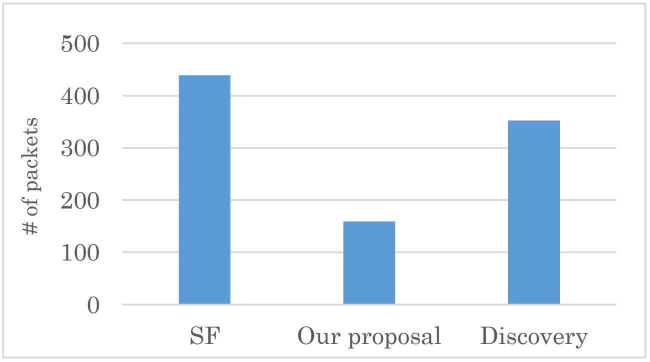


Fig. 5 Total num. of sent and received packets (long)

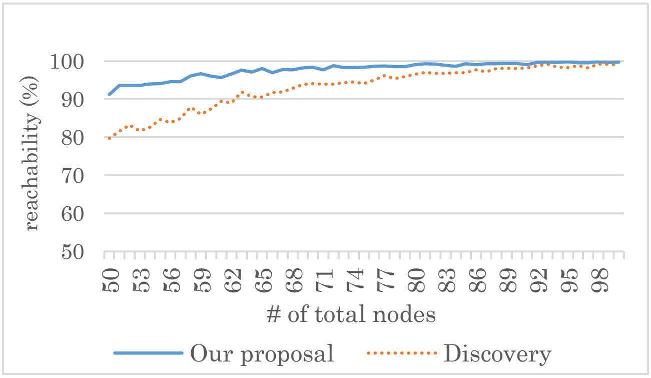


Fig. 6 Data reachability

5 Conclusion

In this paper, we propose an efficient broadcasting information transfer method using location data without using IP address when communication infrastructure is unavailable in such a case as disaster situation. And this proposal is evaluated by simple network simulator. The results of this evaluation shows we have succeeded in reducing a number of redundant packets and achieving high data reachability compared with existing methods.

This proposal is on the presupposition that each nodes are fixed. Therefore, we should plan to study that it is expected to improve the results if each nodes move on.

6 Acknowledgments

This research is conducted as a part of JSPS Grants-in-Aid for Scientific Research 26430372.

7 References

[1] Jorjeta G. Jectcheva, David A. Malts. “A Simple protocol for Multicast and Broadcast in Mobile Ad Hoc Networks”, IETF MANET Working Group Internet-Draft, <draft-ietf-manet-simple-mbcast.txt>, (Jul 2001)

[2] Naohide Fukushi, Keisuke Utsu, Hiroshi Ishii. “A study on the Method of Information Discovery using GPS over Mobile Ad-hoc Network”; (2008)

[3] C. perkins, E. Belding-Royer, S. Das. “Ad hoc On-Demand Distance Vector (AODV) Routing”, Request for Comments: 3561, http://www.ietf.org/rfc/rfc3561.txt, (Jul 2003)

[4] T. Clausen, Ed., P. Jacquet, Ed.. “Optimized Link State Routing Protocol (OLSR)”, Request for Comments: 3626, http://www.ietf.org/rfc/rfc3626.txt, (Oct 2003)

[5] Kenichi Mase, Shiro Sakata. “Ad-Hoc Networks and Meth Networks”, Vol. 1 pp.43-52 (2007)

[6] D.Johnson, Y. Hu, D. Maltz. “The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4”, Request for Comments: 4728, http://tools.ietf.org/rfc/rfc4728.txt, (Feb 2007)

Energy Efficient Data-Driven Networking Processor with Autonomous Load Distribution Capability

Shuji SANNOMIYA and Hiroaki NISHIKAWA
Graduate School of Systems and Information Engineering, University of Tsukuba,
Tsukuba Science City, Ibaraki, Japan

Abstract— *Energy efficiency is one of the crucial issues for the processors realizing battery-operated devices to lengthen the lifetime of wireless network systems. CUE, a data-driven processor, is one of promising energy-efficient processors because of its real-time multiprocessing with essential power consumption. The CUE is realized by an elastic circular pipeline whose elastic capability enables to preserve processing time naturally even when the number of valid data exceeds a given design target temporarily due to the fluctuation of input traffic.*

In this paper, a data-driven networking processor with autonomous load distribution capability is proposed to provide higher-level elastic capability against the larger input traffic fluctuation without any additional controls resulting in the degradation of the energy efficiency. In the processor proposed, the CUE's in chip multiprocessor or many-core architectures are deconstructed and every processor module is placed on a parallelized circular pipeline in order to integrate the distributed elastic capabilities.

Keywords: data-driven processor, real-time multiprocessing, self-timed pipeline, many-core processor, load distribution

1. Introduction

Wireless network systems such as wireless M2M (machine-to-machine) system and wireless sensor network system are one of promising technologies to realize not only convenience but also safety for both human life and social infrastructure. Since they are mainly composed of battery-operated devices and the power budget of the battery is strictly limited, energy efficiency of processors is one of crucial issues to lengthen their lifetime.

Each processor in such battery-operated devices executes both signal processing and networking protocol handling programs. Although the processing load of the processor varies depending on the input traffic fluctuating dynamically, the execution of those programs is imposed strict time constraints. That is, the processor should provide not only thrifty power consumption but also real-time multiprocessing against the load fluctuation.

To realize real-time multiprocessing with essential power consumption, the authors have already studied a series of data-driven processor, named CUE [1], [2]. The CUE realizes data-driven processing scheme in which operation execution is initiated on the arrival of input data as long as computational resources (i.e. pipeline stages) are available, thus it realizes the real-time multiprocessing without extrinsic program-execution overheads such as context switching and interrupt handling resulting in power dissipation. Moreover, the CUE is realized as a circular pipeline structured by self-timed elastic pipeline in which each pipeline stage autonomously transfers valid data based on local negotiation between adjacent pipeline stages. As a result of this local data transfer, the valid data in the self-timed elastic pipeline can be transferred between the pipeline stages even when the other valid data is hold at a following pipeline stage. Therefore, the CUE provides elastic capability against instantaneous load fluctuation without any additional controls or circuits.

In order to improve the energy efficiency, chip multiprocessors or many-core processors are commonly used to convert the parallelism inherent in target programs into core-level parallel execution. The core-level parallel execution increases the throughput and thus makes it possible to lower the supply voltage and decrease power consumption while retaining the throughput, i.e. the energy efficiency can be improved compared with the single-core processor. In the chip multiprocessor or many-core processors, the processing load in each core may vary depending on the input traffic, and the instantaneous increase of the input traffic increases the number of valid data in the pipeline of the core. Generally, the maximum number of acceptable valid data is given as a design target, and clock-synchronized pipelines can deal with the valid data only when the number of valid data is within the design target.

In contrast, CUE-based chip multiprocessor or many-core processor can deal with the load fluctuation beyond the design target. Already the chip multiprocessor implementation of the CUE has been studied and its ultra-low-power feature has also been proven [2], [3], [4]. Existing CUE-based chip multiprocessors are realized by interconnecting the CUE's by using a token router which is a switch-based network and realized by self-timed elastic pipeline [4]. By virtue of the elastic capability, the circular elastic pipeline can accept the valid data when the number of valid data is over the design target. In such crowded circular elastic pipeline, data processed in a pipeline stage may wait to be transferred until the following pipeline stage becomes available. This transfer wait time can be absorbed within the elastic capability. However, the transfer wait time increases the processing time of the CUE when it exceeds the elastic capability. Moreover, the continual occurrence of the processing time increase may result in the violation of the real-time multiprocessing. Such situation where the processing time increase continually occurs is called overload. Consequently, the processing time increase should be avoided in order to avoid the overload situation.

In this paper, a data-driven networking processor with autonomous load distribution capability is proposed to avoid the processing time increase by extracting the elastic capability exhaustively instead of additional controls or circuits resulting in the degradation of the energy efficiency. As long as the processing time increase is absorbed by the elastic capability, the processing time of the CUE is retained and only essential power is consumed. That is, the maximum number of valid data in the circular elastic pipeline absorbing the processing time increase fully is the design target to guarantee the real-time multiprocessing with essential power consumption. As for the chip multiprocessors or many-core processors, the elastic capability is distributed over the cores and thus the transfer wait time in a core can be absorbed only by using the elastic capability of the core even when the elastic capabilities over the other cores are leftover. Therefore, the design target can be enhanced if the leftover elastic capabilities are utilized and the enhanced design target leads to the improvement of the energy efficiency. To realize the enhancement of the design target of the chip multiprocessor or many-core processors, the cores are deconstructed and every processor module is placed on a circular elastic pipeline to integrate the elastic capabilities distributed over the cores. The effectiveness of the processor structure proposed is discussed with the object of the design target.

2. Data-driven networking processor

To guarantee the real-time multiprocessing with essential power consumption, in the circular elastic pipeline of the data-driven networking processor CUE, the number of the valid data should be within the design target. The design target is determined depending on the elastic capability.

In this section, the operating principle of the circular elastic pipeline is explained, and the elastic capability is discussed focused on the signal propagation time. In addition, the leftover elastic capability of the chip multiprocessor configuration of the CUE is discussed.

2.1 Circular elastic pipeline

The CUE's circular elastic pipeline is realized by self-timed elastic pipeline (STP). In the STP, only pipeline stages with valid data are driven exclusively as a consequence of the localized data transfer called handshake. The valid data is called token.

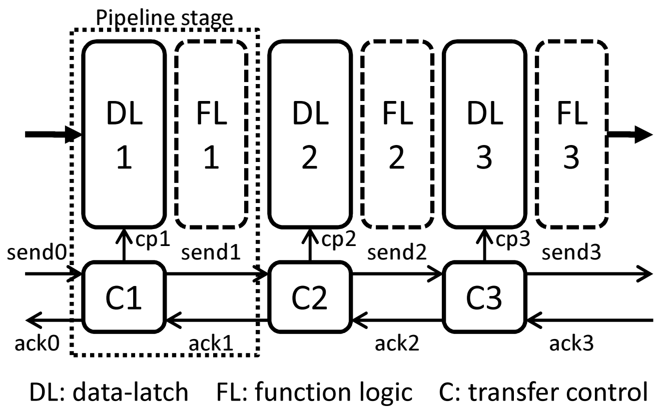


Fig. 1: Self-timed elastic pipeline.

Figure 1 shows the basic structure of the STP in which each stage consists of a data-latch (DL), functional logic (FL) and transfer control unit (C). The STP employs four-phased handshake [5]. Based on the four-phased handshake, the tokens in the STP are transferred between adjacent stages by using transfer request signal (send signal) and acknowledge signal (ack signal) which are based on negative logic, as follows.

- (0) Reset: After the assertion of the reset signal, each C negates both its send signal and ack signal.
- (1) The C asserts its ack signal after its send signal is asserted.
- (2) After the assertion of the ack signal, the preceding C negates its send signal.
- (3) After the negation of the send signal, the C asserts both its gate open signal (cp) and its send signal and concurrently it negates its ack signal, only if the ack signal from the succeeding C is negated. As a result, the token is latched in the stage to which the C belongs.
- The succeeding C repeats the above steps similarly to the C.

This handshake not only concentrates dynamic power consumption into the pipeline stages with valid data but also provides the CUE with elastic capability.

The elastic capability can be defined by the propagation time of the signals in the STP. As shown in figure 2, the minimum time for handshake at a pipeline stage is $(T_f + T_r)$, where T_f and T_r denotes the send signal propagation time and the ack signal propagation time respectively. When the temporal distance $D_{(t)}$ which means the signal propagation time on the critical path of the circuit between two adjacent tokens is equal to or greater than $(T_f + T_r)$, the token can be transferred at T_f because the ack signal arrives before send signal and T_r is overlapped. On the other hand, when $D_{(t)}$ is temporarily less than $(T_f + T_r)$, the transfer of the token is postponed until $D_{(t)}$ becomes equal to or greater than $(T_f + T_r)$. This postponed time is called transfer wait time in this paper.

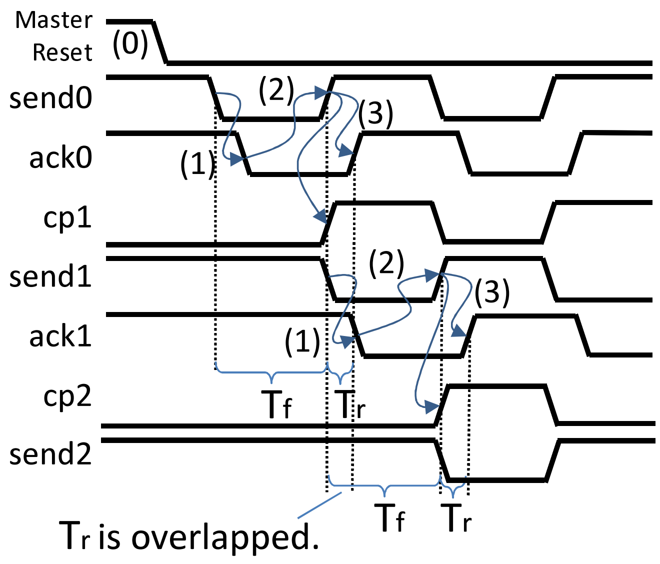


Fig. 2: Timing chart of handshake.

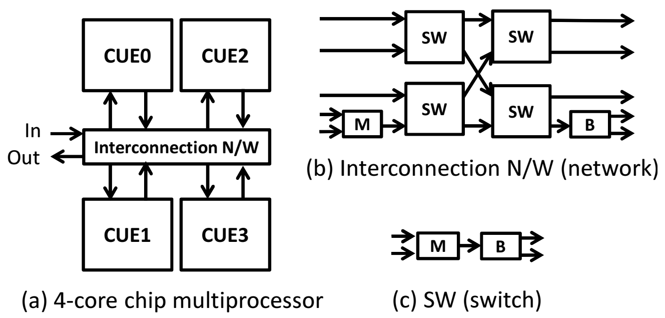

CUE0
CUE2
CUE1
CUE3
In
Out
Interconnection N/W
(a) 4-core chip multiprocessor
SW
M
B
(b) Interconnection N/W (network)
(c) SW (switch)

Fig. 3: Data-driven networking chip multiprocessor (4-core).

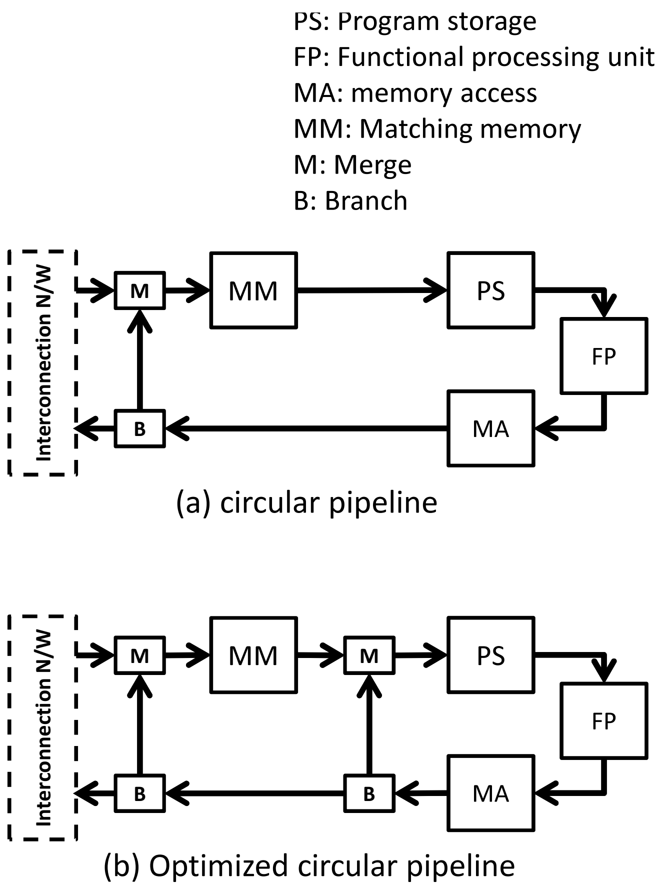


Fig. 4: Data-driven networking processor: CUE.

The transfer wait time can be absorbed if it is equal to or less than the temporal margin $D_{(t)} - (T_f + T_r)$ of the following token. That is, the sum of the $D_{(t)}$ in the circular elastic pipeline determines the elastic capability to absorb the transfer wait time, and thus it should be exhaustively utilized. However, the current chip multiprocessor configuration of the CUE may leave a part of the elastic capability.

2.2 Elastic capability in existing chip multiprocessor

The chip multiprocessor version of the CUE can be easily realized by interconnecting the CUE's by using a token router, as shown in figure 3. The token router is a switch-based multi-stage interconnection network whose switch is realized by the merge (M) and branch (B) stages.

The circular elastic pipeline realizes the circular datapath indispensable to execute programs directly. Figure 4 shows the functional block diagram of the CUE. The CUE consists of matching memory (MM), program storage (PS), functional processing unit (FP) and memory access (MA). As shown in the figure 4(a), the CUE processors are realized by a circular pipeline connecting the MM, PS, FP and MA by using merge and branch stages. The merge stage accepts tokens from two preceding stages in order of arrival and transfers the tokens to a succeeding stage while the branch stage transfers each token to one of two succeeding stages selectively. With this structure, the concurrent operations of the target programs can be naturally exploited over the circular pipeline as long as the pipeline stage is available.

In the CUE processors, each operand is packetized with information required to execute operations into token in order to execute operations independently from each other. The information of the token is called tag. The tag consists of operation code, destination node number and generation. The generation is the number used to identify the stream to which the data belongs and specify the order of the data in a stream. On the other hand, every operation is assigned unique number which is called destination node number, and the destination node number is used to identify an operation to which the data is input.

The MM, PS, FP and MA are used to execute an operation according to the tag. The MM provides temporal storage to keep tokens whose operation code represents binary operation, until the arrival of the paired tokens, and it outputs either a token containing two operands for binary operation or a token containing single operand for unary operation. To realize the pairing of tokens, the MM is realized by a content-addressable memory (CAM) whose key consists of the generation and destination node number. The operand or a pair of operands in the token output from the MM is processed according to the operation code in the FP, and the FP outputs a token whose data is the result of the operation. After that, the operation code of the token is replaced with that of an operation specified by the destination node number of the token in the PS storing operations of the target programs.

The circular elastic pipeline is shared among the operations in order to reduce the circuit area which is rather limited in early sub-micron process technology era and is becoming inconsequential compared to power consumption in deep sub-micron and beyond era. As for the power consumption, the MM may occupy more than a half of the power consumption required to execute an operation because the detection of the arrival of the tokens in the MM is typically realized by using the CAM in which the keys stored are thoroughly compared to the input key for every incoming token.

Fortunately, more than a half of the operations of the protocol handling can be executed without driving the MM. Such operations are classified into two types: single-operand operations and single-operand with constant operations. An example of the former is an operation to realize increment/decrement operation and that of the latter is an operation to read/write memory with absolute address. These two types of operations are collectively called single-token operations, because they can be executed after an input token arrives.

To realize the essential power consumption, an optimized circular pipeline has been proposed [2]. In the optimized circular pipeline, a bypass route is realized to execute the single-token operations without driving the MM, as shown in the figure 4(b).

Figure 5 shows the UDP/IP handling program's processing time measured by using the test chip of the latest CUE, named ULP-CUE [2]. Although the processing time is kept to be approximately constant regardless of multiplicity which means the number of streams processed concurrently, the processing time slightly increases when the multiplicity is 4. This is because the number of tokens temporarily exceeds the design target. As a result, the $D_{(t)}$ becomes temporarily less than $(T_f + T_r)$ and thus the transfer wait time appears on the processing time.

The problem is that the transfer wait time in a CUE can be absorbed only within the sum of $D_{(t)} - (T_f + T_r)$ in the CUE and thus it may appears on the program execution time even though the sum of $D_{(t)} - (T_f + T_r)$ is left in the other CUE's in the chip multiprocessor. Generally, enhancing the design target requires finer pipelining of the

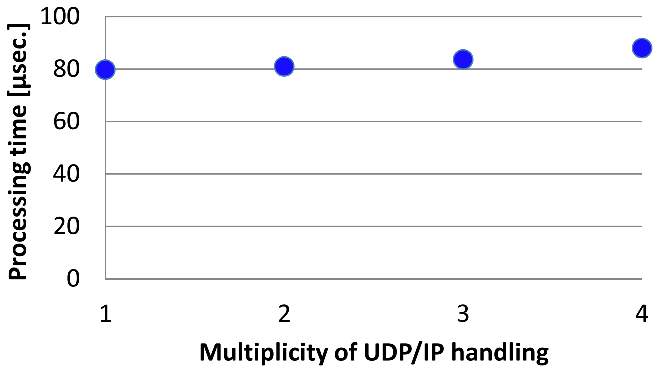


Fig. 5: Processing time of CUE.

processor modules and thus it results in the increase of the power consumption. Consequently, the leftover elastic capability should be exhaustively utilized to improve the energy efficiency.

3. Energy efficient data-driven networking processor architecture

In the CUE, the concurrency of the operations is naturally deployed over the circular elastic pipeline by virtue of the self-timed elastic pipeline (STP). As described in the previous section, during the deployment of the concurrency, transfer wait time is absorbed as long as the number of tokens is within the design target. However, when the number of the tokens exceeds the design target, the transfer wait time appears and increases the processing time of the CUE.

In this section, the design target is modeled by focusing on the elastic capability and a circular elastic pipeline to enhance the design target temporarily is discussed.

3.1 Model of design target

In order to keep the number of tokens in the circular elastic pipeline within the design target, target programs should be designed to suppress the fluctuation of the number of tokens. With such programs, the number of tokens may be constant while a token laps the circular elastic pipeline [6]. Based on this fact, an assumption that the number of tokens remains while a token laps the circular elastic pipeline is introduced, and the design target is modeled based on the assumption.

In the circular elastic pipeline, $(T_f + T_r)$ of a pipeline stage is usually different from that of the other pipeline stages. For example, the arbitration at a merge stage may postpone the arrival of the ack signal and thus the T_r may be increased during the arbitration. Moreover, it is difficult to strictly retain the designed signal propagation time through circuit implementation phase in which unexpected delays are produced by design tools and fabrication environment.

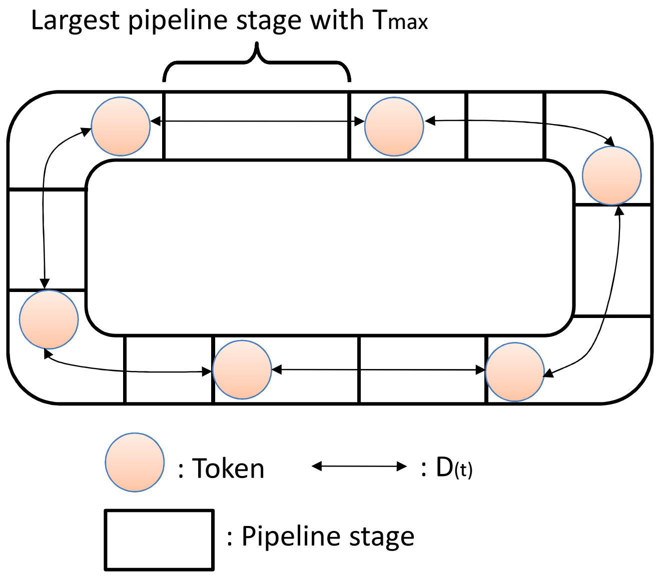


Fig. 6: Temporal distance between tokens.

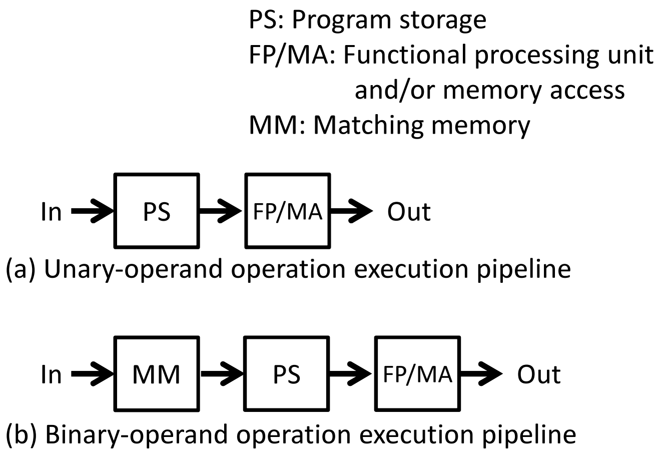

PS: Program storage
FP/MA: Functional processing unit and/or memory access
MM: Matching memory
In
PS
FP/MA
Out
(a) Unary-operand operation execution pipeline
In
MM
PS
FP/MA
Out
(b) Binary-operand operation execution pipeline

Fig. 7: Minimum operation execution pipelines.

As shown in figure 6, by virtue of the handshake of the STP, tokens autonomously attempt to keep $D_{(t)} \geq T_{max}$, where T_{max} denotes the largetst $(T_f + T_r)$ in the circular elastic pipeline. The tokens with $D_{(t)} \geq T_{max}$ are transferred between stages at T_f, and thus the sum of $D_{(t)}$ can be $\sum T_f$ at maximum. Based on these facts, the number of tokens, which is denoted by P_{total}, should satisfy equation (1) to assure $D_{(t)} \geq T_{max}$ for each token.

$$\sum T_f \geq T_{max} \times P_{total} \qquad (1)$$

Based on the equation (1), the design target, which is denoted by P_{DT}, is defined by equation (2).

$$P_{DT} = \lfloor \frac{\sum T_f}{T_{max}} \rfloor \qquad (2)$$

3.2 Circularization of operation execution pipelines

In the chip multiprocessors, it is difficult to forecast the runtime fluctuation of the number of tokens (P_{total}) for each core, and thus it is impossible to share the leftover elastic capability without additional control or circuit which enables dynamic load control. However, the tokens in the circular elastic pipeline autonomously retains their $D_{(t)} \geq T_{max}$ as a result of the handshake, as shown in the figure 6. That is, the temporal margin $D_{(t)} - T_{max}$ can be shared among tokens without no additional control or circuit. This fast leads to the circularization of the operation execution pipelines.

As shown in the figure 4(b), a single token operation can be executed with only PS, FP and MA in principle and the binary-operand operations are executed with PS, FP, MA and MM. Based on these facts, the operation execution pipelines can be categorized into two types: unary-operand operation type and binary-operand operation type. Figure 7 shows these pipelines. To realize the execution of the unary-operand operation, PS and FP/MA are used to fetch operation and execute a given operation, as shown in the figure 7(a). On the other hand, to actualize the operation between two operands, the operand arrived first should be temporally stored until the other operand arrives, and thus the MM is introduced in front of the PS, as shown in the figure 7(b).

To circularize the operation execution pipelines, the unary-operand operation execution pipeline and the binary-operand operation execution pipeline is unified to realize a building unit corresponding to a CUE, and the output of the building unit is connected to the input of the other building unit. Figure 8 shows a circularized operation execution pipeline corresponding to a 2-core chip multiprocessor. In the circularization, the transfer control of the PS and FP/MA is modified to provide bypass routes to drive the MM only for the binary-operand operations. Such transfer control modification can be realized by using already-proposed bi-directional transfer control circuit [7]. The bi-directional transfer control circuit can provide 2-input and 2-output for every pipeline stage. It is true that the structure proposed requires the bi-directional transfer control circuits but they are less than the interconnection network in the chip multi-processor.

In the circularized operation execution pipeline, target programs can be easily assigned to the PS's. As described in the section 2, a unique destination number is assigned to each operation, and the operation is stored to the PS. The tokens in the circularized operation execution pipeline flow circularly, and thus the operations are also assigned circularly. This can be easily realized by modulo operation, as shown in figure 9. In the data-driven programs, a sequence number named rank is commonly used to identify the location on the critical path of the program. The figure 9 shows an example of the case where the number of building units is 2. In this

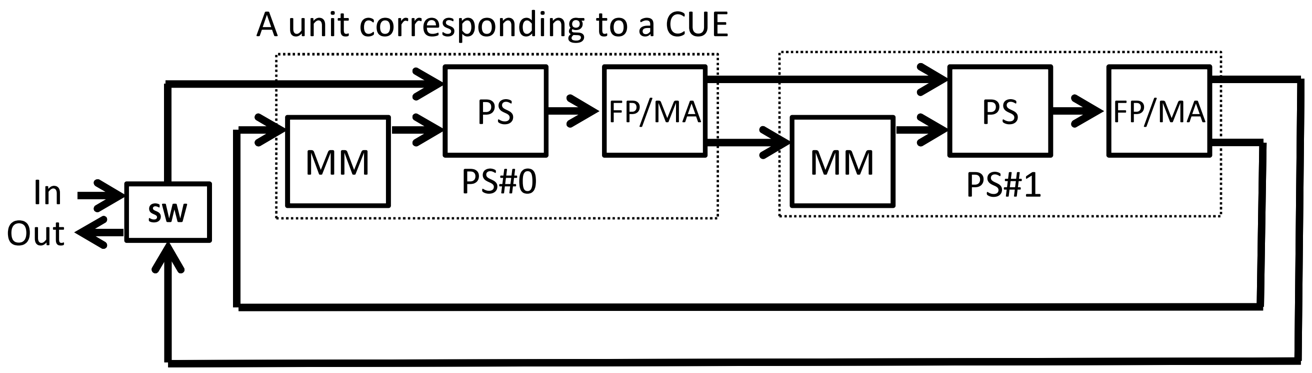


Fig. 8: Circularized operation execution pipeline.

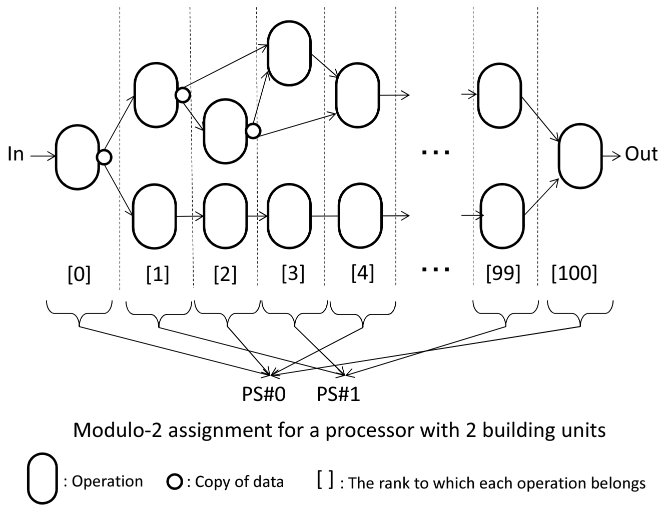

In
Out
[0]
[1]
[2]
[3]
[4]
[99]
[100]
PS#0
PS#1
Modulo-2 assignment for a processor with 2 building units
: Operation
: Copy of data
[ ] : The rank to which each operation belongs

Fig. 9: An example of program assignment to circularized operation execution pipeline.

case, the reminder of the modulo-2 operation against the rank indicates the PS to which the operations are stored.

With the circularization discussed above, the temporal margin $D_{(t)} - T_{max}$ can be shared among all of the tokens in the circularized operation execution pipeline without any additional control or circuit.

3.3 Parallelization of circular pipeline

The circularized operation execution pipeline can deploy the concurrent operations of the target programs over itself naturally as a result of the handshake of the self-timed elastic pipeline. However, the instantaneous increase of the concurrency of the operations may result in the increase of T_{max} of the largest pipeline stage, and thus the design target P_{DT} may decrease resulting in the fear to increase processing time of the target programs. Therefore T_{max} should be kept constant even when the concurrency of the operations increases instantaneously.

The instantaneous increase of the concurrency is brought in the circular operation execution pipeline by the copy of tokens. For instance, the output tokens of the copy at rank=1 program shown in the figure 9 are applied discrete operations at rank=2. The copy is realized by making the handshake twice for a token, and thus $(T_f + T_r)$ of the pipeline stage realizing the copy doubles instantaneously during the copy. That is, T_{max} may double due to the copy.

To absorb such instantaneous increase of the T_{max} and retain the P_{DT}, the circularized operation execution pipeline is parallelized. Figure 10 shows the parallelized circular operation execution pipeline. The parallelization provides two pipelines, and thus the pipeline stages which are busy or handshaking can be bypassed by transferring the tokens to the parallel pipeline stages. In this figure, it is assumed that the copy is realized in the FP/MA. To assure the operation fetch in the two pipelines, the parallel PS's store the same operations. The number of the FP/MA is 4 and thus the throughput can be the same as that of a circularized operation execution pipeline with 4 building units. Moreover, one MM is shared between the parallel two pipelines. This is because the number of binary-operand operations is expected to be less than that of the unary-operand operations.

With the parallelized circular operation execution pipeline, the elastic capabilities over the chip multiprocessor can be integrated by virtue of the circularization of the operation execution pipelines, and the P_{DT} can be retained against the instantaneous increase of the concurrency as a result of the parallelization of the circular pipeline.

3.4 Estimation of elastic capability

The pipeline proposed provides the CUE with the integrated elastic capability against input traffic fluctuation. To show the effectiveness of the pipeline proposed, the integrated elastic capability is estimated quantitatively based on the ULP-CUE and UDP/IP program which are explained in section 2.

The T_{max} and $\sum T_f$ of the ULP-CUE are measured by the post-layout circuit simulation of the ULP-CUE, and they are approximately $3nsec.$ and $14nsec.$ respectively. According to these measured results and the eq. (2), the P_{DT} of the

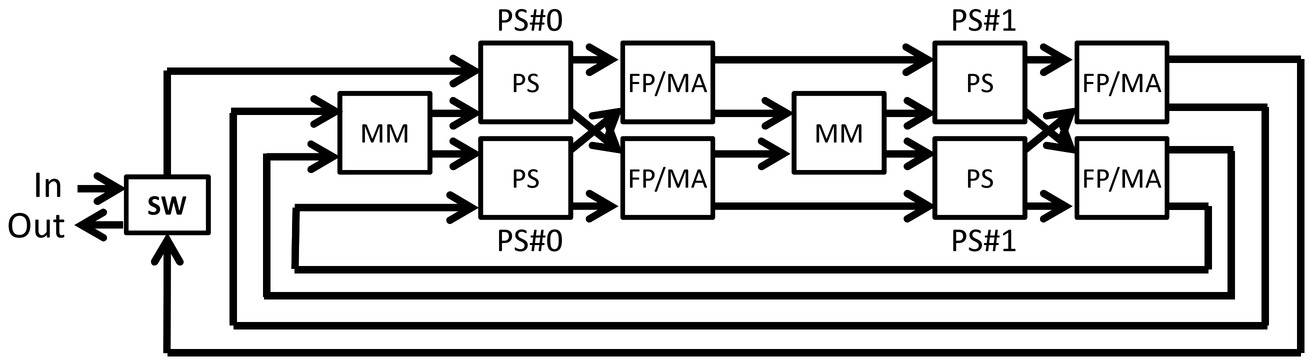


Fig. 10: Parallelized circular operation execution pipeline.

ULP-CUE is 4. Moreover, the P_{total} during the execution of the UDP/IP with one input packet is 1. That is, it may be expected that 4 packets can be processed simultaneously without the processing time increase. However the measured processing time indicates that only two packets (i.e. multiplicity=2) can be processed without the processing time increase, as shown in the figure 5. This is because the P_{DT} decreases instantaneously at runtime.

On the other hand, the parallelized circular operation execution pipeline makes it possible to not only keep the P_{DT} but also integrate the elastic capabilities distributed over cores. If two ULP-CUE's are deployed over the parallelized circular operation execution pipeline, the P_{DT} becomes $16 = 2 \times 2 \times 4$ because the parallelized circular operation execution pipeline realizes two parallel pipelines for two ULP-CUE. As shown in the figure 5, the processing time of the UDP/IP program is approximately 80 μsec., and thus it is expected that the real-time multiprocessing can be guaranteed without any additional controls even when the number of input packets becomes 16 from 1 during approximately 80 μsec.

The circuit size of the parallelized circular operation execution pipeline deploying two ULP-CUE's is comparable with that of a chip multiprocessor with four ULP-CUE's. However, in the chip multiprocessor, it is impossible to distribute the input packets with different data length (i.e. different processing load) to cores properly, and thus the P_{DT} of the chip multiprocessor with four ULP-CUE's is determined by that of one ULP-CUE. Moreover, the processing time in the chip multiprocessor increases due to the interconnection network, and thus the energy efficiency degrades. In contrast, the processor proposed enables to increase the P_{DT} in proportion to the number of building units.

4. Conclusions

In this paper, a data-driven networking processor with autonomous load distribution capability is proposed to avoid the processing time increase by exhaustively extracting the elastic capability of the self-timed elastic pipeline. In the processor proposed, cores are deconstructed and every processor module is placed on a circular pipeline in order to integrate the elastic capabilities distributed over the cores. Moreover, the circular pipeline is parallelized to prevent the concurrent execution of the operations from being a bottleneck of the circular pipeline. With this parallelized circular pipeline, the amount of the acceptable load fluctuation can be scaled along with the number of original cores, and thus the real-time multiprocessing can be preserved against the input traffic fluctuating dynamically without any additional controls. The result of the quantitative evaluation on an actual application program will be presented at conference.

Acknowledgement

Although it is impossible to give credit individually to all those who organized and supported our project, the authors would like to express their sincere appreciation to all the colleagues in the project.

This research work was supported in part by IS program of Semiconductor Technology Academic Research Center (STARC).

References

[1] Hiroaki Nishikawa, “Design Philosophy of a Networking-Oriented Data-Driven Processor: CUE,” IEICE Transactions on Electronics, Vol.E89-C No.3, pp.221-229, Mar. 2006.
[2] Shuji Sannomiya, Kazuhiro Aoki, Makoto Iwata, and Hiroaki Nishikawa, “Power-Performance Verification of Ultra-Low-Power Data-Driven Networking Processor: ULP-CUE,” in Proc. of PDPTA, pp.465-471, July 2012.
[3] Kazuhiro Aoki, Hiroshi Ishii, Makoto Iwata, and Hiroaki Nishikawa, “A Comprehensive Evaluation of ULP-DDNS by Platform Simulator,” in Proc. of PDPTA, pp.445-451, July 2012.
[4] Kazuhiro Aoki, Shuji Sannomiya, Makoto Iwata, Hiroshi Ishii and Hiroaki Nishikawa, “An Implementation of Platform Simulator for Congestion-Free Ultra-Low-Power Data-Driven Networking System,” in Proc. of PDPTA, pp.611-617, July 2013.
[5] C. J. Myers, “Asynchronous circuit design,” Univ. of Utah John Wiley & Sons, Inc., 2001.
[6] Shuji Sannomiya, Naoya Kagawa, Keiichi Sakai, Makoto Iwata, “A Data-Driven On-Chip Simulation Module and Its FPGA Implementation,” in Proc. of PDPTA, pp.710-716, July 2008.
[7] Shuji Sannomiya, Kazuhiro Komatsu, Makoto Iwata, “A Self-Timed Pipeline Circuit for Low-Power Surrounding LSI Chips,” in Proc. of PDPTA, pp.613-619, June 2007.

Spatial Parallelization of Self-Timed FFT Circuit

Norifumi UNO, Ryuichi TAGUCHI, and Makoto IWATA
Graduate School of Engineering, Kochi University of Technology,
Kami, Kochi, 782-8502 Japan

Abstract— *This paper presents a multimode and multichannel FFT (MM-FFT) circuit for mobile broadband wireless access (MBWA), wireless local area network (WLAN) and wireless personal area network (WPAN) applications. Using the proposed MM-FFT architecture based on the self-timed pipeline (STP) circuit, variable-point and multiple-streams FFTs are capable of achieving a high throughput even if each FFT point and sampling rate are different from others. The proposed architecture improves its performance by utilizing both spatial and pipeline parallelism inherent in FFT calculation.*

The proposed MM-FFT circuit was designed and synthesized using a 65 nm CMOS standard cell library. The post-synthesis simulation results indicated that the proposed circuit could achieve 4.0 G sample/s at maximum, which is 5 times faster than the original MM-FFT circuit.

Keywords: heterogeneous wireless communication, multimode and multichannel, FFT, self-timed pipeline, spatial parallelization

1. Introduction

The amount of data traffic on wireless networks has been increasing exponentially in recent years. Meanwhile, usage of smart wireless devices has widely spread. To accommodate such huge traffic and provide uniform user experience in ubiquitous environment, heterogeneous wireless network (HetNet) has been investigated [1].

Our research project aims to establish a self-timed pipeline (STP) implementation for the dependable wireless systems (DWS) [2] supporting multimode and multiband interfaces like HetNet. Since the STP circuit inherently has a clockless passive operation mode [3], [4], it can flexibly process any combination of signal streams even if they are sampled at different frequencies.

Digital modulation techniques employed in typical HetNets are based on orthogonal frequency-division multiplexing (OFDM) or single-carrier frequency-division multiplexing (SC-FDM), and fast Fourier transform (FFT) circuit is one of key components in DWS stations and user terminals. This is because FFT is utilized not only for digital modulation but also for channel estimation [5] realizing seamless handover among heterogeneous cells. That is, a multimode and multichannel FFT module adaptive to different characteristics enables us to select optimum modulation, channel, and network dynamically depending on individual wireless communication condition. This sort of seamless handover in DWS will also contribute to energy efficiency.

There have been many studies on high-speed FFT circuits [6], [7], [8], [9] which can be applied to 2.4 G sample/s wireless personal area network (WPAN) standard. Flexible-radix-configuration multipath-delay-feedback (FRCMDF) FFT circuit [6] supports triple modes for WPAN, wireless local area network (WLAN), and mobile broadband wireless access (MBWA) applications. However, this circuit operates only in a single mode at the same time. Therefore, a basic architecture of multimode and multichannel FFT (MM-FFT) circuit based on the STP circuit was proposed in [7], its performance could not reach to 2.4 G sample/s required for WPAN. This paper proposes spatial parallelization technique of the previous MM-FFT circuit and evaluates it by 65 nm CMOS circuit design. Post-synthesis simulation results show the proposed MM-FFT circuit achieves 4.0 G sample/s in 512-point single mode and 2.0 G sample/s in 512-point dual mode.

2. Self-Timed Pipeline

Each pipeline stage of the STP consists of a data latch as a pipeline register, function logic, and transfer control unit named C-element. The basic structure of the STP is shown in Figure 1. The data latch, function logic, and C-element are denoted by DL, Logic, and C, respectively. The data is packed with tag into packet form, and the packet is transferred between the pipeline stages as a result of the communication between the C's in the adjacent stages. The communication is performed stage-by-stage according to the 4-phase handshake protocol [10] by using transfer request and acknowledge signals which are called send signal and ack signal respectively. The stage-by-stage transfer unit holds a state of each pipeline stage independently, and the states of the stages are defined below according to the handshake protocol. Here, the C-element in the i-th stage is denoted by C_i.

- Reset state: The send and ack signals are negated after the assertion of the reset signal.
- Idle state: The C_i waits until the $send_i$ is asserted.
- Busy state: The $send_i$ is asserted at the beginning of the transfer of the packet from the precedent $(i-1)$-th stage. After the assertion of the $send_i$, the C_i asserts its ack signal ack_i. In response to the assertion, the C_{i-1} negates the $send_i$. After that, if and only when both

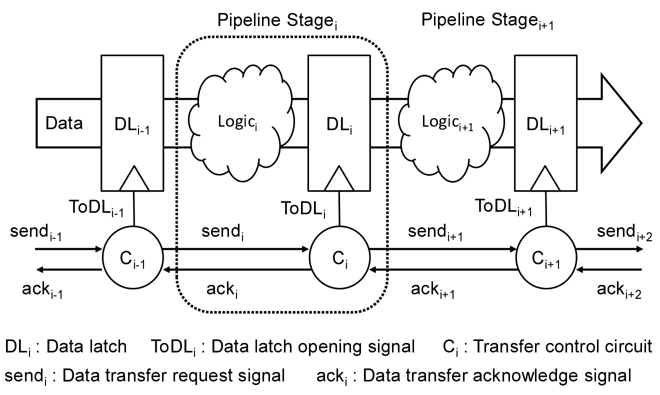


Fig. 1: Self-timed pipeline.

the $send_i$ and ack_{i+1} are negated, the C_i asserts the $ToDL_i$ to open the DL_i and it asserts $send_{i+1}$ at the same time. As a consequence, the packet is latched in the $(i+1)$-th stage, and the i-th stage goes to idle state. Otherwise, the C_i waits until the ack_{i+1} is negated while it keeps its send and ack signals.

The successive stages receiving the assertion of the send signal go to busy state and their C's repeat the same transfer control sequence individually. Forwarding latency of a C-element for transferring the data from a DL to the succeeding DL is adjusted to delay time of a critical path in the stage, including the response time and setup time of the DL. Its backward latency is adjusted to the hold time of the DL.

This stage-by-stage transfer control of the STP suggests the timing of the power controls. That is, in the idle stages, the circuit of the DL and combinational logic can be powered-off, i.e., the supply-voltage can be cut, while that of the C and sequential logic can be powered-down, i.e., the supply voltage can be lowered enough to keep the circuit's states [11]. Moreover, in the busy stages, those circuits should be powered-down enough to assure the switching of the transistors, i.e., the supply-voltage can be lowered as long as the required switching speed is achieved [12].

3. Parallelism in FFT

The single carrier FDE module performs on the receiver side after the FFT calculation to combat frequency-selective fading and phase distortion [5]. To equalize the transmitted data in frequency domain, a pilot signal is used for estimating the transfer function and the noise power in the air channel. Therefore, after the received data are transformed from time domain to frequency domain by FFT, they are equalized based on estimated results and then retransformed to time domain by IFFT.

In the case of OFDM, FFT is also used for modulating data onto each subcarrier and IFFT is for demodulating data on each subcarrier. Furthermore, in multiple-input multiple-output (MIMO) antenna configuration, a multichannel FFT/IFFT circuit is necessary in a transmitter/receiver.

Therefore, we aim to implement an MM-FFT circuit in which multiple FFT operations of variable sizes are simultaneously performed for multiple input signal sequences sampled in different frequencies.

Originally FFT is a fast version of discrete Fourier transform (DFT). N point DFT is defined by the equation (1).

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}$$
$$W_N^{kn} = e^{-j2\pi kn/N}, \quad k = 0, 1, ..., N-1 \tag{1}$$

where the input sequence of N complex data $x(0)$, $x(1)$, $\cdots$, $x(N-1)$ is transformed into an N-periodic sequence of complex data.

In the Cooley-Tukey FFT algorithm, radix-r butterfly operations are recursively applied to N input signals, and the depth of recursion is $log_r N$. In each recursion, the number of butterfly operations (i.e., r-point FFT) is N/r and they can be calculated in parallel because there is no data dependency among them. An example of a decimation-in-frequency FFT (N=8, r=2) is shown in Figure 2. As seen in this dataflow diagram, four butterfly operations can be concurrently executed in each recursion step.

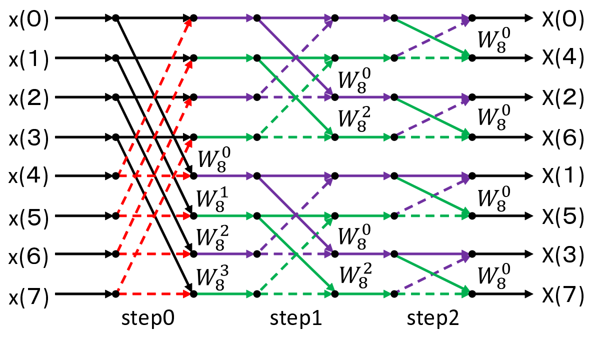


Fig. 2: Dataflow diagram of radix-2 decimation-in-frequency FFT (N=8).

If the dataflow diagram of FFT shown in Figure 2 is interpreted based on the dynamic dataflow model [3], multiple instances of the same FFT diagram can be allowed to be executed by introducing a channel identifier, which differentiates an instance among them. In the same way, the dataflow diagram of a butterfly operation can be interpreted for multiprocessing of the butterfly if every data flowing the diagram have a set of identifiers ID, which is composed of channel identifier ch, step identifier $step$, and butterfly instance identifier within the step btf. In this case, it is necessary to provide a function supplying an appropriate set of operands with those identifiers and storing intermediate data in the memory buffer. This parallel execution scheme in case of radix-2 butterfly is illustrated in Figure 3. In the figure, every operand and result of butterfly are identified by $ID(ch, step, btf)$ and multiple sets of operands are issued from the commutator consecutively. The commutator manages the number of operand sets for executable butterfly instances, which represents the degree of parallelism P_{ch}. At the same time, the commutator attaches appropriate identifiers to those issued operands.

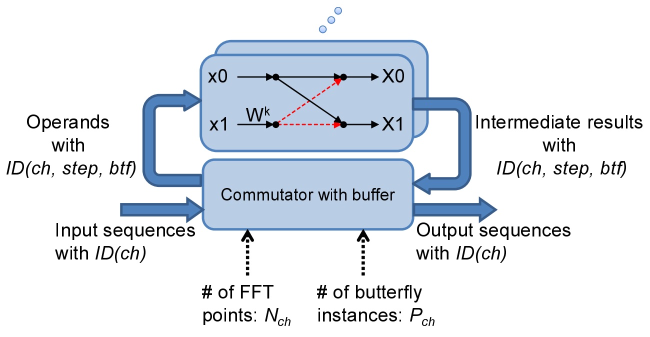

x0
X0
x1
W^k
X1
Operands with ID(ch, step, btf)
Intermediate results with ID(ch, step, btf)
Commutator with buffer
Input sequences with ID(ch)
Output sequences with ID(ch)
# of FFT points: N_ch
# of butterfly instances: P_ch

Fig. 3: Parallel execution scheme of multiple butterfly instances.

As long as the butterfly operation with a correct set of identifiers is executed under dynamic dataflow model, valid execution of multiple FFT calculations is guaranteed even if the size of an FFT N_{ch} and the sampling frequency of its input data sequence are different from others.

```
do {
        btf += $P_{ch}$;
        if (btf >= ($N_{ch}$/r)) {
                step++;
                btf %= ($N_{ch}$/r);
        }
} while ( step < $log_r N_{ch}$);
```

Fig. 4: ID handling function in commutator.

Furthermore, it is noted that the topological connectivity of signal flow graph of FFT shown in Figure 2 is invariant but this graph can be modified to a uniform structure [7]. This uniform type of FFT structure allows the commutator to fetch r operands from the buffer memory in parallel. Because the memory access pattern is uniform at all steps, the buffer memory can be simply composed of r single-port memory banks.

Figure 5 shows a dataflow graph representation of the proposed parallel execution scheme of MM-FFT. Firstly, the input complex data is stored in multi-bank buffer memory consecutively. If a set of operands for the first butterfly is ready to be computed, an instance of FFT is initiated and P_{ch} sets of identifiers IDs are issued at ID handling module according to input data arrival. After that, r operands are read from the buffer memory in parallel based on the issued ID. Similarly, twiddle factors are read from TF lookup table in parallel and radix-r butterfly is calculated. The r resultant data from the butterfly are written into the buffer memory in parallel. Then, preparation of a continuous butterfly operation is conducted as defined in Figure 4. After executing the last butterfly in the FFT instance, the output data read from the buffer memory are reordered.

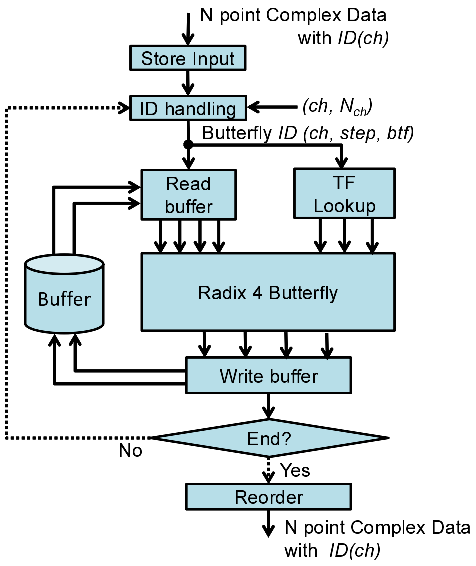
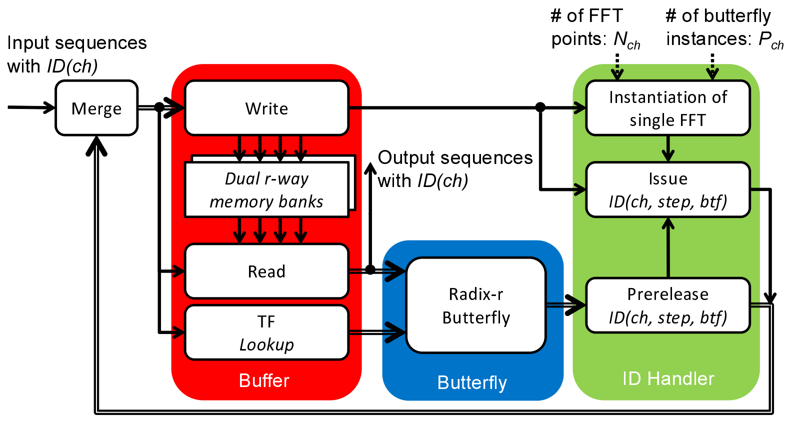


Fig. 5: Dataflow diagram of the radix-4 multichannel scheme.

4. MM-FFT Architecture

The MM-FFT architecture was designed to process multiple streams in parallel, where every stream can be calculated in different mode, i.e., the number of FFT points or sampling rate.

In order to realize the dataflow shown in Figure 5, it is essential to maintain stable dataflow in the STP without any pipeline bottleneck as well as to guarantee atomic (i.e., read-after-write) accesses of intermediate data stored in the buffer memory during execution. Therefore, the buffer memory accesses must be integrated at the single STP stage. Moreover, intermediate resultant data of radix-r butterfly are written in different buffer addresses from that of operands when the uniform type of FFT structure is employed. Thus, in our design, we adopt dual buffer memory modules each of which is used for butterfly operations in either even step or odd step of the FFT. It requires $2N_{ch}$ words SRAM for N_{ch}-point FFT. As a result, MM-FFT engine is designed as shown in Figure 6 to utilize the passive operation mode of STP. This FFT engine operates as follows.



Input sequences with ID(ch)
Merge
Write
Dual r-way memory banks
Read
TF Lookup
Buffer
Output sequences with ID(ch)
Radix-r Butterfly
Butterfly
# of FFT points: N_{ch}
# of butterfly instances: P_{ch}
Instantiation of single FFT
Issue ID(ch, step, btf)
Prerelease ID(ch, step, btf)
ID Handler

Fig. 6: STP implementation of the MM-FFT.

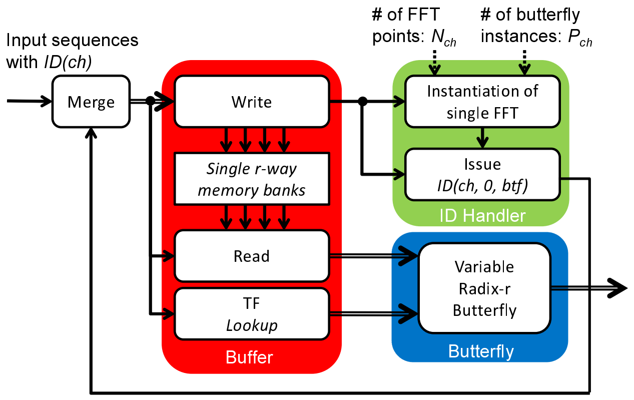
All data flowing in the pipeline has an operation code op as well as $ID(ch, step, btf)$. The op is assigned one of operations, i.e., in, $read$, $write$, or out. Every stage in the STP-based MM-FFT engine changes its operation depending on op of the packet.

- **input phase**: N_{ch} input data from a channel ch consecutively arrive at one of input ports of the merge stage. At that time, each input data is composed as a packet form including a complex number, ch identifier, index i(=0,...,N_{ch}-1) and $op(= in)$. The input data reaching to the buffer memory stage is written in a place associated with index i of the packet.
- **instantiation phase**: If op of a packet arriving at the ID handler stage is in, an FFT for the channel ch may be instantiated. Only when r operands necessary for the first butterfly are stored in the buffer memory, an FFT instance for the channel ch is initiated. ID(ch, 0, 0) for the first butterfly is issued with $op(= read)$. After that, succeeded butterfly instances are instantiated with ID(ch, 0, 1),..., ID(ch, 0, P_{ch}-1) within the allowable degree of parallelism P_{ch}.
- **read phase**: If op of a packet arriving at the buffer memory stage is $read$, r operands for the btf-th butterfly instance are read out from the buffer memory in parallel. Their addresses can be calculated from $ID(ch, step, btf)$. To allow those parallel accesses, the buffer memory is composed of dual r-way memory banks. If the $step$ is even, the operands are read from the first set of r-way memory banks. If odd, the second set is accessed for operand fetches. At the same time, $(r-1)$ twiddle factors necessary for the butterfly are fetched from the twiddle factor lookup table in parallel. Since the lookup table holds twiddle factors only in the fourth quadrant, each lookup data needs the change of quadrant by swapping the real and imaginary number, or changing on one(or both) sign(s) of the number(s).
- **butterfly phase**: If op of a packet arriving at the butterfly stage is $read$, a butterfly instance is executed using r operands and $r-1$ twiddle factors.
- **prerelease ID phase**: If op of a packet arriving at the ID handler stage is $read$, this stage prepares to write r resultant data to the buffer memory. In the uniform type of FFT, all results of a butterfly should be stored to one of memory banks. Therefore, word length of each memory bank is expanded to $r \times$ (*length of a complex word*). By this expansion, all results are written at the same time. In this prerelease ID phase, r results are packed into one word with $op(= write)$ to prepare for the next writing phase.
- **write phase**: If op of a packet arriving at the buffer memory stage is $write$, a packed result of the btf-th butterfly instance is written in the buffer memory. Its address can be calculated from $ID(ch, step, btf)$. If the $step$ is even, the intermediate result of the FFT is written in the second set of r-way memory banks. If odd, the first one is accessed for the result storing. After writing the result, the ID packet is transferred to the ID handler stage. In this stage, $ID(ch, step, btf)$ is updated based on the function defined in Figure 4 and then op is changed to $read$. If $step$ exceeds $log_r N_{ch}$, the FFT operation is finished and buffered data are output. In this case, op is changed to out.

5. Spatial Parallelization of MM-FFT

The degree of pipeline parallelism realized by the MM-FFT architecture is limited by the number of STP stages. That is, in order to improve the pipelined parallel processing performance, it is necessary to divide every STP stage finer. However, some STP stages cannot be divided more, e.g., the buffer memory stage. Therefore, we focus on the spatial parallelism in FFT in place of the pipeline parallelism to further improve the FFT performance.

As illustrated in Figure 2, the N-point FFT calculation is recursively divided into r (N/r)-point FFT's in each step. Since there is no data-dependency among (N/r)-point FFT's, each of them can be independently calculated on an corresponding MM-FFT circuit. Therefore, in addition to multiple MM-FFT modules, we introduce a single-step MM-FFT module to realize the first step radix-r butterfly operations. This module can be implemented to simplify the MM-FFT architecture as shown in Figure 7. In this simplified single-step MM-FFT architecture, $step$ and P_{ch} are fixed to zero and N_{ch}/r respectively, the prerelease ID module and dual r-way memory bank can be reduced, and the resultant data of the butterfly stage are directly output. Figure 8 illustrates the spatially-parallelized MM-FFT structure, where r_f denotes radix in the first step, r_l denotes radix in latter recursive steps of FFT, and P_s denotes the degree of spatial parallelism.



Input sequences with ID(ch)
Merge
Write
Single r-way memory banks
Read
TF Lookup
Buffer
# of FFT points: N_{ch}
# of butterfly instances: P_{ch}
Instantiation of single FFT
Issue ID(ch, 0, btf)
ID Handler
Variable Radix-r Butterfly
Butterfly

Fig. 7: Simplified architecture for single-step MM-FFT.

The first-step MM-FFT module is similar to multi-path delay feedback configuration [13]. After input data from 0 to $(r_f - 1/r_f)N_{ch}$ are stored in multi-path FIFO buffer, radix-r_f butterfly operations in the first step are begun to be executed. Resultant data are consecutively distributed to the latter MM-FFT modules and they are processed there in parallel. The destination module of resultant data is decided based on $(btf\ mod\ P_s)$.

The radix r_f in the first-step MM-FFT module can be flexibly altered from 2 to 8 according to $N_{ch}(= r_f \times r_l^{integer})$, while r_l is fixed. That is, the proposed architecture can process multiple streams in parallel even if the sampling rate or the number of points N_{ch} is different each other.

Here, it can be noted that the number of the latter MM-FFT modules may be less than $log_{r_l}(N_{ch} - 2^{r_f})$ because the MM-FFT module itself can process multiple streams as long as its processing resource is enough to do those. In other words, spatial parallelism P_s, i.e., the number of the latter MM-FFT modules, can be optimized at the circuit design phase, depending on the cost-performance requirement of target wireless applications.

Since the total number of butterfly operations for N_{ch}-point FFT can be calculated by N_{ch}/r times $\log_r N_{ch}$ and P_{ch} butterflies of those are simultaneously processed in pipeline, the total processing time of the original MM-FFT, T_{MM-FFT}, can be estimated by equation (2).

$$T_{MM-FFT} = \frac{N_{ch}\log_r N_{ch}}{rP_{ch}} ST_f \tag{2}$$

where S denotes the number of STP stages and T_f denotes the average data-forwarding time in a STP stage, i.e., the circulation time of the STP ring is ST_f.

Based on the equation (2), the total processing time of the proposed MM-FFT, T_{FFT}, can be estimated by equation (3).

$$T_{FFT} = S_f T_{ff} + \frac{N_{ch}\log_{r_l}\frac{N_{ch}}{r_f}}{P_s r_l P_{ch}} S_l T_{fl} \tag{3}$$

where S_f and S_l denotes the number of STP stages in the first single-step MM-FFT module and the latter MM-FFT modules respectively, and T_{ff} and T_{fl} denotes the average data-forwarding time. Because the total number of butterfly operations in each latter MM-FFT module is reduced by P_s and r_f, T_{FFT} is shortened except for the small overhead time of the first step MM-FFT module, $S_f T_{ff}$.

6. Evaluation

In order to evaluate the performance and area of the proposed MM-FFT circuit, its STP circuit is designed using a 65 nm CMOS standard cell library. The specifications of this circuit are as follows. The word length of a complex data is 32 bits; 16 bits for real part or imaginary part. 2 bits are assigned for integer and 14 bits are assigned for fraction part. The number of FFT points can be altered from 64 to 1024 points. The radix r_f and r_l are four. The degree of pipeline parallelism P_{ch} for a single channel can be altered from 1 to 16 and that of spatial parallelism P_s is fixed to four. The number of STP stages, S_f and S_l, are respectively 4 and 30. The designed STP circuit was described by Verilog-HDL and synthesized by Design Compiler, Synopsys Inc.

Table 1 shows total cell area of the proposed circuit and the original MM-FFT circuit. In those circuit implementations, the signal data and twiddle factors are stored in SRAM modules, i.e., data mem. and TF mem. The cell area of memory accounts for 70 % of the total cell area.

Table 1: Total cell area of the synthesized circuit.

MM-FFT		Logic	data mem.	TF mem.
Proposed	cell/bit	118464 cells	96K bit	30K bit
	area $[mm^2]$	0.67	1.51	0.27
Previous [7]	cell/bit	25865 cells	64K bit	24K bit
	area $[mm^2]$	0.15	0.37	0.05

Although hardware cost of the proposed circuit is 4.3 times larger than the MM-FFT, throughput of the proposed circuit is 5 times faster than the MM-FFT in the case of $N_{ch} = 512$, $r = 4$, $r_f = 4$, and $r_l = 4$.

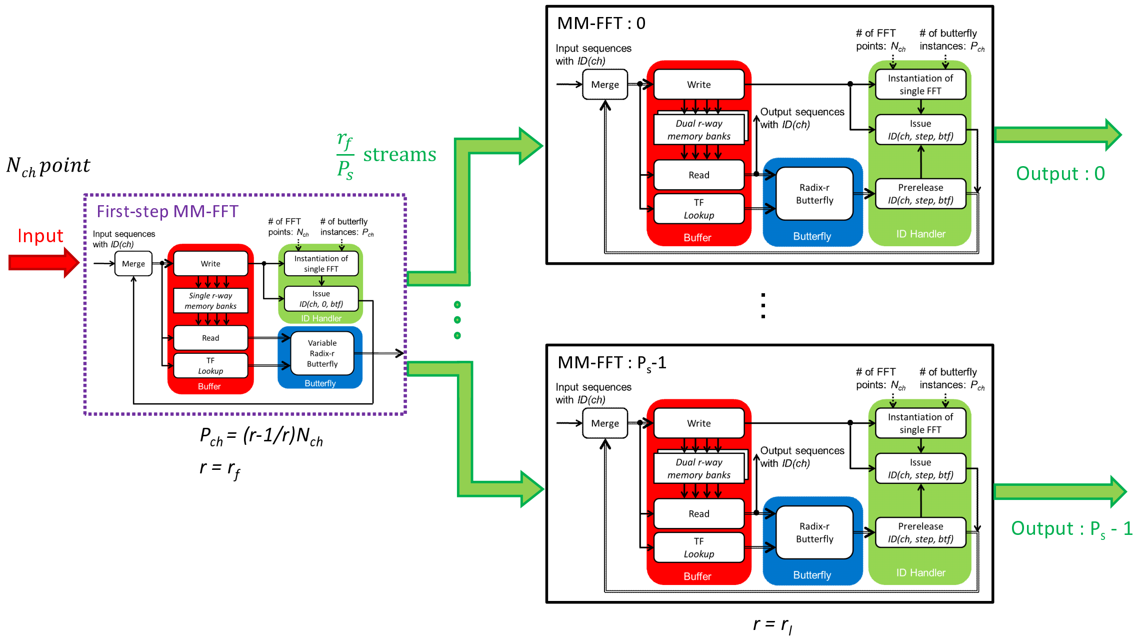


Fig. 8: Spatially-parallelized MM-FFT architecture.

As for the performance of the designed circuit, post-synthesis simulation results revealed that the equation (3) nearly estimated the performance. However, actual T_f could not be optimized in our design work. We thus recalculated post-synthesis simulation results by assuming that T_f is 1 ns in average. This assumption is realistic according to our LSI fabrication experiences of STP-based data-driven chip-multiprocessor [14], [15]. Based on this assumption, the potential performance of the designed circuit is shown in Figure 9 in the case of single input stream.

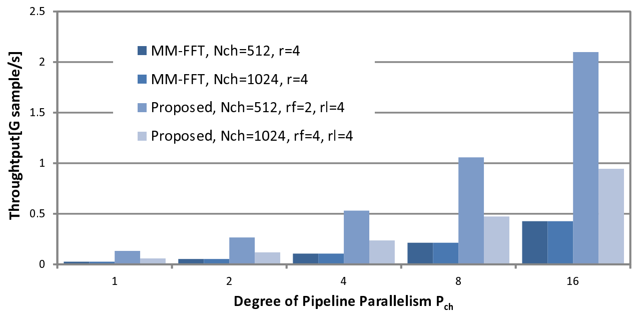


Fig. 9: FFT performance comparison of the designed circuit (single stream).

In this figure, the horizontal axis denotes the degree of pipeline parallelism P_{ch} and the vertical axis indicates the maximum sampling rate (throughput) which can be achieved by the designed circuit. In the case of double input stream such as multiple-input multiple-output (MIMO) scheme, the performance for each stream's FFT is falls in half but it is scalable without any parallel processing overhead.

In the case of 4 pipeline stages in the first-step MM-FFT module, radix-8 in the MM-FFT and 30 pipeline stages in the MM-FFT is shown Figure 10. According to the equation (2), the throughput is basically irrelevant to the number of streams as long as processing resource is enough for those.

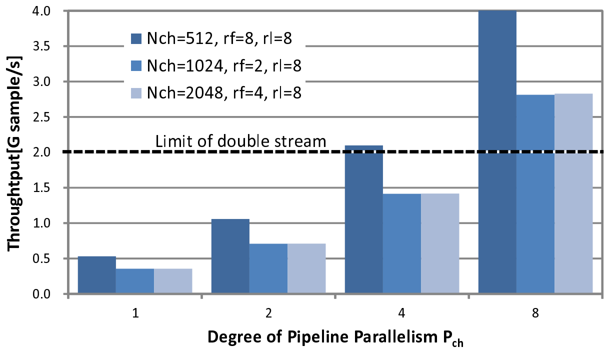


Fig. 10: FFT performance estimation ($r_s = 8$).

Since this circuit is designed to assume data transfer time is 1 ns and four data is simultaneously input, maximum input rate is limited up to 4.0 G sample/s in the case of single input stream. In the case of two input streams with same sampling rate, this upper boundary is decreased to half, i.e., 2.0 G sample/s. This boundary is drawn by a dotted line in the figure. Thus, the performance is limited at 4.0 G sample/s in the case of $N_{ch} = 512, P_{ch} = 8$, and single input stream but it is sufficient to apply the proposed circuit to WPAN standard (IEEE802.11ad) application ($N_{ch} = 512$, 2.4 G sample/s). This indicates that a HetNet application shown in Table 2 can be accommodated by the proposed circuit.

Table 2: An applicable set of air interfaces.

Air interface (Example)	WPAN (IEEE802.11ad)	WLAN (IEEE802.11n)	MBWA (LTE)
# of points N_{ch}	512	2048	128
Sampling rate (G sample/s)	2.4	0.04	0.02
# of streams	1	2	2
MIMO channels	—	4	—

If two streams of the 2.4 G sample/s WPAN application are accommodated, the designed circuit cannot process them because of the upper limitation of the performance 4.0 G sample/s. For the further performance improvement, the number of parallel input data should be expanded from four to eight.

7. Conclusion

In order to establish dependable and heterogeneous wireless network systems, mobile devices should be equipped with a multimode and multiband interface for stable wireless communication. Such mobile terminals would be useful and sustainable so as to provide uniform experience for users.

This paper focuses on the multimode and multichannel FFT (MM-FFT) used for MIMO OFDM and SC-FDE and proposes a spatial parallelization technique of the MM-FFT circuit based on STP circuit. Evaluation results indicated that the proposed circuit could achieve throughput higher than the original MM-FFT circuit [7]. Furthermore, the proposed STP implementation will be feasible to the required performance for current heterogeneous wireless communication, e.g., 2.4 G sample/s for WPAN, 2-stream 20 M sample/s for WLAN, 2-stream 40 M sample/s for MBWA. This evaluated condition is assumed for a typical case of HetNet so that benchmark evaluation in case of other usage scenario should be further studied.

Since mobile devices and terminals have to operate at low power consumption, the proposed circuit must cooperate with typical low power techniques, e.g., dynamic voltage scaling (DVS) [16]. Since the degree of pipeline parallelism P_{ch} in the proposed circuit can be adjusted along with dynamically-scaled supply voltage, the circuit could contribute to lower its power consumption. Quantitative evaluation on such energy efficiency will be reported in other article.

Acknowledgement

Although it is impossible to give credit individually to all those who organized and supported our project, the authors would like to express their sincere appreciation to all the colleagues in the project.

This research work was supported in part by Core Research for Evolutional Science and Technology (CREST), Japan Science and Technology Agency (JST). The circuit design work was supported by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Synopsys, Inc. and Cadence Design Systems, Inc.

References

[1] R. Q. Hu, Y. Qian, S. Kota, and G. Giambene, “HetNets - A New Paradigm for Increasing Cellular Capacity and Coverage,” IEEE Commun. Mag., Vol. 18, No. 3, pp. 8–9, June 2011.

[2] K. Tsubouchi, S. Kameda, and N. Suematsu, “Dependable Air,” IEICE Trans. Electron., Vol. J95-C, No. 12, pp. 450–459, Dec. 2012 (in Japanese).

[3] H. Terada, M. Iwata, and S. Miyata, “DDMP’s: Self-Timed Super-Pipelined Data-Driven Multimedia Processors,” Proc. IEEE, Vol. 87, No. 2, pp. 282–296, Feb. 1999.

[4] K. Komatsu, S. Sannomiya, M. Iwata, H. Terada, S. Kameda, and K. Tsubouchi, “Interacting Self-Timed Pipelines and Elementary Coupling Control Modules,” IEICE Trans. Fundamentals, Vol. E92-A, No. 7, pp. 1642–1651, Jul. 2009.

[5] D. Falconer, S. L. Ariyavisitakul, A. Benyamin-Seeyar, and B. Eidson, “Frequency Domain Equalization for Single-Carrier Broadband Wireless Systems,” IEEE Commun. Mag., Vol. 40, No, 4, pp. 58–66, Apr. 2002.

[6] S. Tang, C. Liao, and T. Chang, “An Area- and Energy-Efficient Multimode FFT Processor for WPAN/WLAN/WMAN Systems,” IEEE J. Solid-State Circuits, Vol. 47, No. 6, pp. 1419-1435, June 2012.

[7] R. Taguchi, H. Ohiso, K. Mendori, K. Miyagi, and M. Iwata, “Self-Timed Single Circular Pipeline for Multiple FFTs," Proc. The 2013 Int’l Conf. on Parallel and Distributed Processing Techniques and Applications, pp. 625–630, July 2013.

[8] G. Zhong, F. Xu, and A. N. Willson, Jr., “A Power-Scalable Reconfigurable FFT/IFFT IC Based on a Multi-Processor Ring,” IEEE J. Solid-State Circuits, Vol. 41, No. 2, pp. 483–495, Feb. 2006.

[9] P. Tsai, C. Chen, and M. Huang, “Automatic IP Generation of FFT/IFFT Processors with Word-Length Optimization for MIMO-OFDM Systems,” EURASIP Journal on Advances in Signal Processing, Vol. 2011, No.1, Jan. 2011.

[10] C. J. Myers, “Asynchronous Circuit Design,” John Wiley & Sons, Inc., July 2001.

[11] K. Miyagi, S. Sannomiya, M. Iwata, and H. Nishikawa, “Autonomous Power-Supply Control for Ultra-Low-Power Self-Timed Pipeline,” Proc. The 2008 Int’l Conf. on Parallel and Distributed Processing Techniques and Applications, pp. 704–709, July 2008.

[12] K. Miyagi, M. Iwata, S. Sannomiya, and H. Nishikawa, “Low-Powered Self-Timed Pipeline with Runtime Fine-Grain Power Supply,” Proc. The 2012 Int’l Conf. on Parallel and Distributed Processing Techniques and Applications, pp. 472-478, July 2012.

[13] Y. Lin, H. Liu, and C. Lee, “A 1-GS/s FFT/IFFT Processor for UWB Applications,” IEEE J. Solid-State Circuits, Vol. 40, No. 8, pp. 1726–1735, Aug. 2005.

[14] S. Sannomiya, K. Aoki, M. Iwata, and H. Nishikawa, “Power-Performance Verification of Ultra-Low-Power Data-Driven Networking Processor: ULP-CUE,” Proc. The 2012 Int’l Conf. on Parallel and Distributed Processing Techniques and Applications, pp. 465–471, July 2012.

[15] K. Miyagi, S. Sannomiya, M. Iwata, and H. Nishikawa, “Self-Timed Power-Aware Pipeline Chip and Its Evaluation, ” Proc. The 2011 Int’l Conf. on Parallel and Distributed Processing Techniques and Applications, pp. 442–448, July 2011.

[16] K. Miyagi, S. Sannomiya, M. Iwata, and H. Nishikawa, “Low-Powered Self-Timed Pipeline with Variable-Grain Power Gating and Suspend-Free Voltage Scaling,” Proc. The 2013 Int’l Conf. on Parallel and Distributed Processing Techniques and Applications, pp. 618–624, July 2013.

DL Timing-Error Detection and Recovery Circuit for Self-Timed Pipeline

Shohei OKAMUNE[1], **Kei MIYAGI**[2], **and Makoto IWATA**[1]
[1]Graduate School of Engineering, Kochi University of Technology, Kami, Kochi, 782-8502 Japan
[2]Department of Information and Communication Systems Engineering,
Okinawa National College of Technology, Nago, Okinawa, 905-2192 Japan

Abstract— *With advanced semiconductor technology, higher dependability and low power consumption of LSI chip is required more and more. That is, a kind of resilient capability to allow partial failure but to work with a self-recovery mechanism by the remaining part of LSI chip must be provided. In this study, we focus on the timing-error which is one of the transient fault of LSI's and we then try to improve dependability of the self-timed pipeline (STP) circuit which is one of necessary technologies in future heterogeneous many-core LSI chips. This paper proposes a timing-error detection and recovery circuit applicable to data latches in STP without impairing its autonomous behavior. The proposed timing-error recovery circuit is localized within a stage to guarantee the STP's handshake protocol. For a preliminary evaluation, we applied the proposed circuit to five-stage 32-bit integer multiplier module by using 65nm CMOS standard cell library.*

Keywords: self-timed pipeline, dependability, timing-error, error detection, error recovery

1. Introduction

Steady performance improvement of modern computer systems due to the advanced semiconductor integration technology for LSI chips is indispensable for the development of information society. According to Moore's law, over several ten-billions of transistors are estimated to be integrated on a LSI chip in 2020. However, more integrated transistors must cause more failure rate of the LSI chip. This is because the failure rate of the chip could be approximately proportional to the number of transistors. Furthermore, the transistor characteristics will vary widely due to the downscaled process variations [1]. Therefore, the dependability of the LSI chips will become more important in the future, especially in case of emergent situations.

There has been studied about dependable and low-power clock-synchronous pipeline circuits such as Razor flip-flop [2], Canary flip-flop [3], and so on. Those circuits aimed to improve the dependability as well as to reduce the design margin provided for dynamic voltage and frequency scaling. By introducing those flip-flops, average supply voltage can be reduced, i.e., energy efficiency of the circuit will be improved.

However, there is little study on dependable self-timed circuits required for globally-asynchronous locally-synchronous (GALS) like network-on-chip module of heterogeneous many-core chips. This paper thus focuses on dependable and energy-efficient self-timed pipeline (STP) circuits [4], [5], especially data-latch timing-error detection and recovery circuit for the STP. In the following section, the basic structure and behavior of the STP circuit are briefly introduced and its timing-error tolerance issues are discussed. A timing-error detection and recovery circuit for the STP is proposed in section 3. In section 4, its timing constraints in the circuit design phase are described and preliminary circuit design of a 32-bit multiplier based on the proposed circuit is reported.

2. Timing-error tolerance of STP

In the self-timed pipeline (STP) circuit, every pipeline stage intercommunicates (hand-shakes) with its neighbor stages and processes a stream of data at stage-by-stage. The "hand-shake" protocol enables the stage to process and transfer the data autonomously. This section introduced basic structure and behavior of the STP circuit and then addresses its timing-error tolerance issues in comparison with related works.

2.1 Self-timed pipeline

STP circuit is configured as shown in Figure 1. Each pipeline stage is composed of a data latch DL_i operating as a pipeline resister, a Logic, and an autonomously driven element C_i which is used for controlling data transfer between adjacent pipeline stages. Every data is transferred at stage-by-stage based on localized control signals (send and ack signals) between adjacent pipeline stages as follows.

1) (Master reset) After resetting the whole STP, all $send$ and ack signals are asserted.
2) (Phase 1: sending) After completing a send process at stage$_{i-1}$, C_i at $stage_i$ opens the data latch DL_i when both $send_i$ and ack_{i+1} signals are asserted. At the same time, $send_{i+1}$ signal is negated for its succeeded $stage_{i+1}$.
3) (Phase 2: receiving) After that, ack_{i+1} signal is negated by C_{i+1}. This indicates that $stage_{i+1}$ is receiving the data transferred from $stage_i$.

4) (Phase 3: completion of send process) C_i then intends to complete the data transfer. At that time, $send_{i+1}$ is asserted again.
5) (Phase 4: completion of receive process) After that, ack_{i+1} signal is asserted again by C_{i+1}. This indicates that $stage_{i+1}$ has completed to receive the data from $stage_i$.
6) The above steps are iterated as long as there are data in the pipeline.

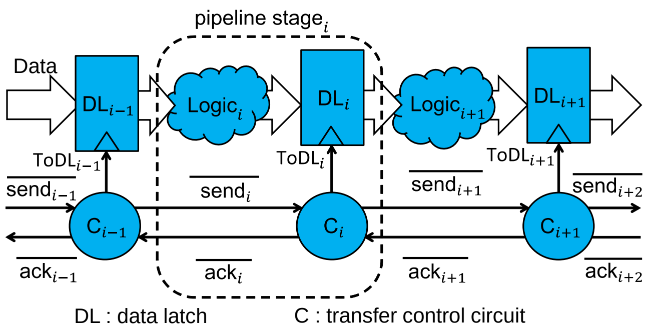

pipeline stage$_i$
Data
DL$_{i-1}$
Logic$_i$
DL$_i$
Logic$_{i+1}$
DL$_{i+1}$
ToDL$_{i-1}$
ToDL$_i$
ToDL$_{i+1}$
$\overline{\text{send}_{i-1}}$
$\overline{\text{send}_i}$
$\overline{\text{send}_{i+1}}$
$\overline{\text{send}_{i+2}}$
C$_{i-1}$
C$_i$
C$_{i+1}$
$\overline{\text{ack}_{i-1}}$
$\overline{\text{ack}_i}$
$\overline{\text{ack}_{i+1}}$
$\overline{\text{ack}_{i+2}}$
DL : data latch
C : transfer control circuit

Fig. 1: Basic structure of self-timed pipeline.

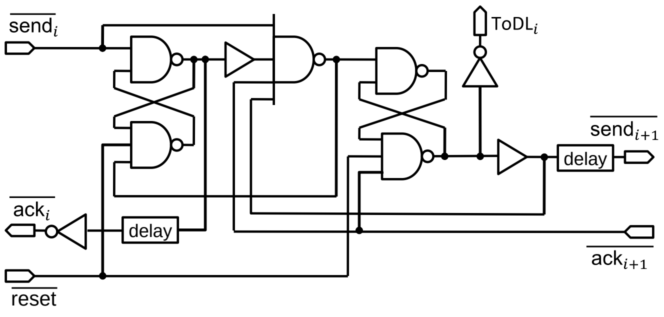

$\overline{\text{send}_i}$
ToDL$_i$
$\overline{\text{send}_{i+1}}$
delay
$\overline{\text{ack}_i}$
delay
$\overline{\text{ack}_{i+1}}$
reset

Fig. 2: Data-transfer control circuit for STP.

For those data-transfer steps, all send and ack signals are designed in negative logic in this paper. Our STP circuit operates based on the 4-phase hand-shake protocol with bundled data. In order to implement quasi-Muller's C element by using standard CMOS cell libraries, a circuit proposed in [4] is employed in this paper. As shown in Figure 2, this circuit is composed of two SR latches, and 4-input nand gate, and two delay cells. The former SR latch keeps a receiving state whether the stage completes to receive data from its preceded stage or not (*completion* or *receiving*). The latter SR latch keeps a sending state whether the stage completes to send data to its succeeded stage or not (*completion* or *sending*). Both latches thus hold a *completion* state when the C element is initiated by the master reset signal. The output of the nand gate is negated when $send_i$ and ack_{i+1} are asserted, the state in the former SR latch is changed from *receiving* to *completion*, and the state in the latter SR latch is *sending*. The delay time of two delay cells should be adjusted depending on the latency time of critical path and setup/hold time of the DL within the $Logic_i$ of the pipeline stage.

Since all wires within the STP are localized in their corresponding pipeline stage, its timing optimization efforts on signal integrity such as clock skew problem are not severe compared with that of the globally synchronous pipeline circuit. STP can operate robustly even if tact time of a certain pipeline stage is longer than that of other stages. However, in such case, the pipeline throughput is degraded so that the pipeline tact of every stage should be optimized and shortened in its circuit design phase.

2.2 DL timing-error in STP

DL timing-error detection and recovery in STP is important not only to improve STP's resiliency against process and environmental variability but also to eliminate design margin effective for STP's dynamic voltage scaling (DVS) technique [6] in terms of low power consumption.

As for the clock synchronous circuits, *in-situ* timing-error monitoring is very effective for dependable low-power circuit with DVS so that many circuits have been proposed [2], [3], [7]. Most of them employ duplicated logic circuits and detect timing-error by comparing their results. For example, a Razor flip-flop [2] shown in Figure 3 can detect timing-error by comparing an output of a main flip-flop with that of a shadow latch controlled by a delayed clock. If a timing-error occurs on the main flip-flop, the valid data kept in the shadow latch is written back to the main flip-flop for correcting the error in the next clock cycle. By using the Razor flip-flop and controlling supply voltage along with error frequency, the design margin of supply voltage and frequency can be reduced. However, the Razor flip-flop requires the complex hardware such as a counterflow pipeline [8] to recover the timing-error.

In place of such error recovery hardware, a Canary flip-flop proposed in [3] aims to predict timing-error for con-

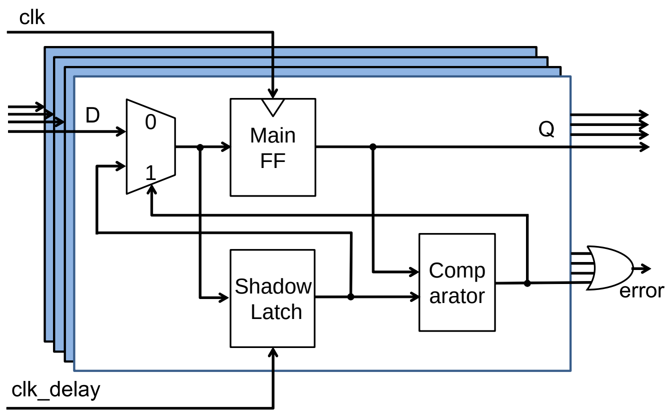


Fig. 3: Block diagram of Razor flip-flop.

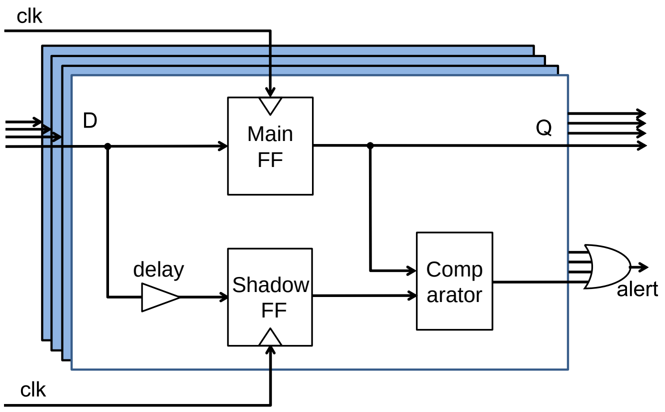


Fig. 4: Block diagram of Canary flip-flop.

trolling the supply voltage. In the Canary flip-flop shown in Figure 4, an input signal to the shadow latch is delayed and the shadow latch is driven by the same clock signal of the main flip-flop. Thus, it does not need the complex hardware but there is a severe assumption that any timing-error does not occur at the main flip-flop. As a result, the reducible design margin of the Canary flip-flop is smaller than that of the Razor flip-flop.

In order to minimize the design margin, detection and recovery of the timing-error is also essential in the case of the STP circuit. Although the error recovery hardware such as the counterflow pipeline can be introduced in the STP, autonomous behavior by virtue of the localized handshake in the STP will be lost. Therefore, a DL timing-error detection and recovery circuit for the STP is proposed in the following section.

3. DL timing-error recovery for STP

If the Razor flip-flop is applied to the STP for DL timing-error recovery, a ToDL signal in a STP stage has to be asserted again when a timing-error occurs at the stage. As described in the previous section, the ToDL is asserted as a result of handshake with neighbor stages. In order to assert the ToDL again, the handshake process with neighbor stages must be reset and restart. This means that the complex recovery hardware such as the counterflow pipeline is also required even in the STP, which might increase circuit and performance overhead beyond the single stage.

Therefore, we propose a timing-error recovery circuit localized in a single STP stage. The proposed circuit is illustrated in Figure 5. As shown in this figure, a timing-error monitoring function is realized in the same way as the Razor flip-flop, i.e., it is composed of a main flip-flop driven by the ToDL, a shadow latch driven by the delayed ToDL, and a comparator of both output data. When a DL timing-error occurs at an STP stage, a multiplexer selects the valid data stored in the shadow latch. After the valid data is successfully processed and transferred to the succeeded pipeline stage, the control signal of the multiplexer must be reset to 0. This is because a timing-error might not occur at the next pipeline phase. Since the completion of pipelined process at the STP stage can be detected by the rising edge of the ack_{i+1} signal, this signal is utilized as an enable signal (en_i) to the multiplexer. This sort of an extended flip-flop for DL is named OK flip-flop in this paper.

The control signals from all OK flip-flops within a DL_i are aggregated to a single representative signal ($error_i$). That is, when a DL timing-error occurs at any OK flip-flop in the DL_i, the $error_i$ signal is asserted. In this case, the valid data is delayed and transferred along with the delayed $ToDL_i$ signal. If the assertion of data-transfer start signal $send_{i+1}$ is delayed for the same time as the delayed ToDL, the valid data can propagate through the (i+1)-th logic circuit ($Logic_{i+1}$). This selective delay circuit for the OK flip-flops is shown in Figure 6. As a result, the timing-error can be recovered and the wrong data can be successfully corrected. In the case of Razor flip-flop, the error recovery process additionally needs one or more clock cycle(s). On the contrary, the recovery time of the proposed circuit can be adjustable to the delay time for the ToDL. If the process or environmental variation of the circuit is small, the recovery time can be shortened. Furthermore, since the proposed timing-error recovery circuit is localized within a stage to guarantee the STP's handshake protocol, the recovery time might be buffered because of the elastic buffering capability inherent in the STP circuit. This kind of feature cannot be found in the clock-synchronous pipeline circuit so that it is very unique to the STP circuit.

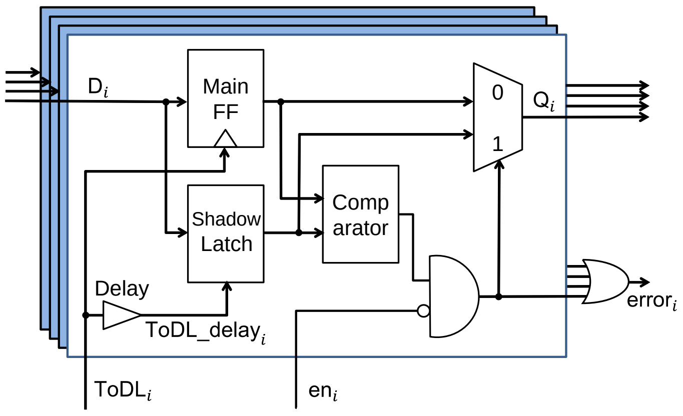


Fig. 5: Block diagram of OK flip-flop.

4. Signal timing

4.1 Signal timing of STP

In general, forwarding latency required to transfer valid data from one set of data-latches to a set of data-latches in the succeeding stage is calculated by the sum of response

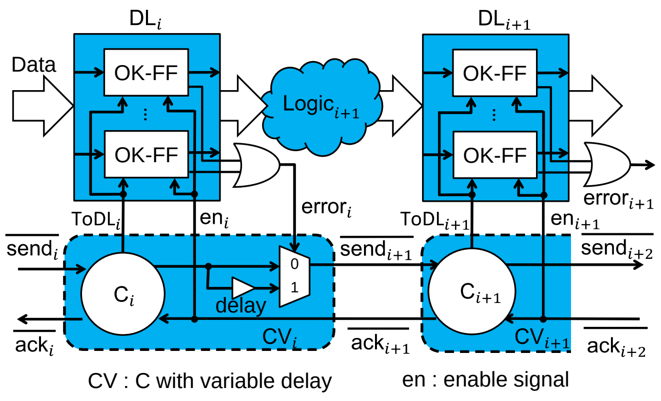


Fig. 6: STP implementation capable of DL timing-error recovery.

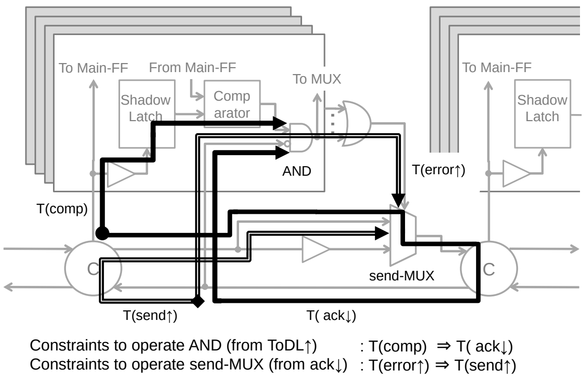

To Main-FF
From Main-FF
To MUX
Shadow Latch
Comparator
AND
T(error↑)
T(comp)
send-MUX
C
T(send↑)
T(ack↓)
Constraints to operate AND (from ToDL↑) : T(comp) ⇒ T(ack↓)
Constraints to operate send-MUX (from ack↓) : T(error↑) ⇒ T(send↑)

Fig. 7: Timing constraints of STP with OK flip-flop.

time of the data latch τ_q, delay time of a critical path in the logic τ_{cp}, and setup time of the data latch τ_{setup}. Thus, handshake time of STP must be adjusted to the forwarding latency. As explained in Section 2, the original STP circuit operates based on the 4-phase handshake protocol so that latency time T_f required from the first to the third phase of the protocol has to exceed the forwarding latency.

After completing the data-transfer, the data latch has to receive the next data correctly. Therefore, backward latency time T_r required for the fourth phase must exceed hold time of the data latch τ_{hold}.

$$T_f \geq \tau_q + \tau_{cp} + \tau_{setup} \quad (1)$$
$$T_r \geq \tau_{hold} \quad (2)$$

Using the two parameters and, it is then possible to define pipeline throughput as $1/(T_f+T_r)$ and pipeline efficiency as $T_f/(T_f+T_r)$. Pipeline throughput is a measure of packet flow rate through the pipeline. Pipeline efficiency is the proportion of net processing time spent on packet processing in terms of pipeline throughput.

If the equation (1) is not satisfied in runtime, i.e., a timing-error occurs, the proposed OK flip-flop detects the timing-error and the third phase involved in T_f is prolonged by the proposed circuit to process the valid data in the logic circuit. Although the equation (2) might be not satisfied in runtime, this issue is not covered in this paper.

4.2 Signal timing related to OK flip-flop

When any timing-error does not occur at all, the output of the main flip-flop propagates to the next logic circuit ($Logic_{i+1}$) in the same way as the original STP circuit. In this case, the control signal of the multiplexer in OK flip-flop is negated. This means that the third handshake phase is not prolonged, i.e., asserted $send_{i+1}$ is not delayed. In terms of the pipeline throughput and power consumption, there is small overhead related to the multiplexer in OK flip-flop and the multiplexer to select the send signal.

If a timing-error occurs, some constraints on signal timing within the proposed circuit are satisfied. Significant timing constraints are related to two multiplexers. Figure 7 illustrates critical paths concerning on those constraints.

The first one of those constraints is related to the control signal of the multiplexer in OK flip-flop. This control signal is generated by a comparison result of the comparator and the negated ack_{i+1}. In the case of timing-error, the comparison result must be settled before the negated ack_{i+1} arrives in order to prevent any glitch at the control signal of the multiplexer. If this constraint is guaranteed, both input signals of the multiplexer are apparently settled before the control signal is asserted to indicate the occurrence of a timing-error. Signal transitions of the error$_i$ are shown in Figure 8. Figure 8(a) shows the case of no timing-error and the figure (b) shows the case of the occurrence of a timing-error. As seen in this figure, error$_i$ signal is originated from asserted ToDL$_i$ and derived from two paths. One is from ToDL$_i$ to error$_i$ via ToDL_delay$_i$ and en$_i$ and the other is via $send_{i+1}$ and ack_{i+1}.

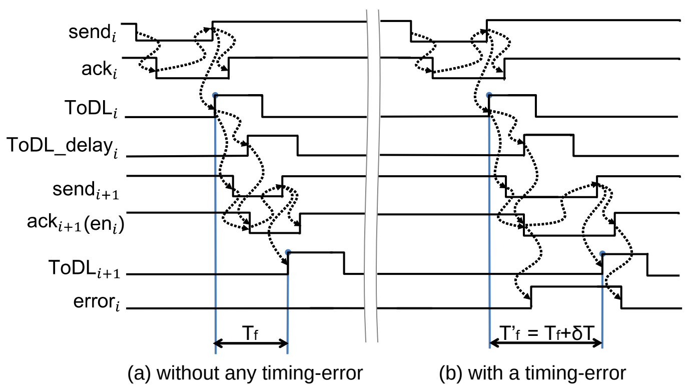


Fig. 8: A timing chart of STP with OK flip-flop.

The second constraint is related to the multiplexer selecting an original send signal or a delayed send signal. As indicated by two double lines in Figure 7, a control signal $error_i$ and two input signals of the multiplexer are originated from ack_{i+1} signal. On the one hand, during the first handshake phase, $error_i$ must be negated to select an original send signal that is not a delayed send signal. On the other hand, during the third handshake phase, an appropriate send signal must be selected depending on whether a timing-error occurs or not. When the timing-error occurs, the asserted send signal must be delayed and the original send signal must not propagate to the next stage through the multiplexer. In this case, $error_i$ signal must be asserted before the original send signal is asserted. If this constraint is violated, $send_{i+1}$ signal wave will have a short glitch and the proposed circuit will fail to recover the error.

As seen in this timing chart, if the $send_{i+1}$ signal is delayed as same as the delay time δT_f of ToDL_delay$_i$, the error can be recovered by the proposed circuit. In this case, the forwarding latency will be prolonged to $T_f + \delta T_f$. This overhead time for recovering the timing-error is flexibly optimized along with the delay variability of the logic circuit in stage-by-stage at the circuit design phase. Since the proposed circuit is still faced with the short path problem like the Razor flip-flop, reasonable δT_f might be about a half of T_f.

Therefore, the proposed circuit will work well when the variability of the critical path delay is less than 50 % and the shortest path delay in the logic circuit is longer than the half of the critical path delay. Furthermore, the recover latency is localized within the single stage so that it might be buffered because of the elastic buffering capability inherent in the STP circuit.

For the feasibility study on the proposed circuit, a 32-bit integer multiplier circuit was designed by using a 65nm CMOS standard cell library. As shown in Figure 9, the designed multiplier is divided into five pipeline stages and it is composed of four 16-bit multipliers and a 64-bit adder circuit. The proposed OK flip-flop is applied to every pipeline stage and it is controlled by the C element with variable delay.

The designed multiplier was described by Verilog HDL and synthesized by Design Compiler of Synopsys Inc. Table 1 summarizes the synthesized results in comparison with the original STP implementation of the same multiplier. In the case of no timing-error, the forwarding latency of the proposed circuit was slightly prolonged because there is latency of the multiplexer to select the output of the main flip-flop and the shadow latch. In this design, the delay time for ToDL is set to 5 ns. When the timing-error occurs at a pipeline stage, the forwarding latency of the stage becomes 1.5 times longer than the normal case. As for the cell area overhead, the proposed circuit is 1.7 times larger than the original STP implementation. In general, the logic circuit part is more dominant in the pipeline stages compared with DL and C and it is possible to apply the proposed circuit to limited stages so that it is expected that the area overhead would decrease in the case of actual STP circuits such as data-driven processor [4], [9].

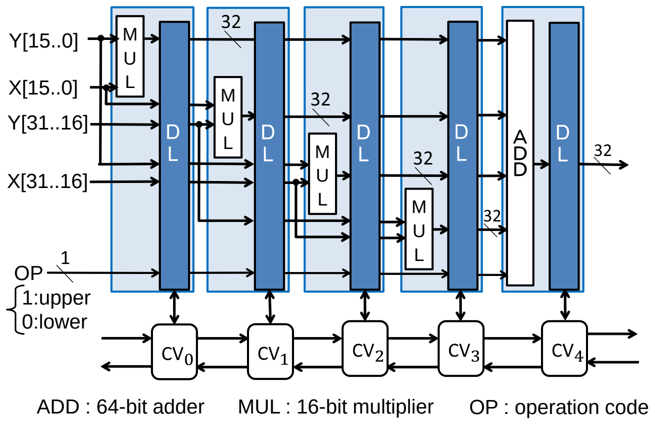


Fig. 9: STP implementation of 32bit multiplier with the OK flip-flop.

Table 1: Latency and area of the designed multiplier.

STP implementation		OK-FF	D-FF	ratio
normal case: T_f	[ns]	9.58	8.9	1.08
error case: T'_f	[ns]	14.49	-	-
cell area	[mm^2]	0.036	0.021	1.71

5. Conclusion

This paper discussed a data-latch timing-error detection and recovery for the self-timed pipeline (STP) circuit to improve its dependability and the energy efficiency. The proposed circuit including OK flip-flop and variable delay can recover the timing-error within the single pipeline stage. Additional latency for the error recovery could be autonomously absorbed because of STP's elastic buffering capability as long as the pipeline occupancy does not reach the maximum pipeline efficiency.

Since the paper reported only preliminary evaluation results on post-synthesis design, post-layout design for an actual LSI application must be evaluated in terms of latency, die area, and power consumption with dynamic voltage scaling (DVS).

Acknowledgement

Although it is impossible to give credit individually to all those who organized and supported our project, the authors would like to express their sincere appreciation to all the colleagues in the project.

This research work was supported in part by Core Research for Evolutional Science and Technology (CREST), Japan Science and Technology Agency (JST). The circuit design work was supported by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Synopsys, Inc. and Cadence Design Systems, Inc.

References

[1] K.J. Kuhn, M.D. Giles, D. Becher, P. Kolar, A. Kornfeld, R. Kotlyar, S.T. Ma, A. Maheshwari, and S. Mudanai, "Process Technology Variation," IEEE Trans. Electron Devices, vol.58, no.8, pp. 2197–2208, Aug. 2011.

[2] D, Ernst, N. Sung Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation," Proc. the 36th Annual Symp. on Microarchitecture, pp. 7–18, Dec. 2003.

[3] Y. Kunitake, T. Sato, and H. Yasuura, "A Replacement Strategy for Canary Flip-Flops," Pacific Rim Int'l Symp.on Dependable Computing, pp. 227–228, Dec. 2010.

[4] H. Terada, S. Miyata, and M. Iwata, "DDMP's: Self-Timed Super-Pipelined Data-Driven Multimedia Processors," Proc. of the IEEE, Vol. 87, No. 2, pp. 282–295, Feb. 1999.

[5] K. Komatsu, S. Sannomiya, M. Iwata, H. Terada, S. Kameda, and K. Tsubouchi, "Interacting Self-Timed Pipelines and Elementary Coupling Control Modules," IEICE Trans. Fundamentals, Vol. E92-A, No. 7, pp. 1642–1651, July. 2009.

[6] K. Miyagi, S. Sannomiya, M. Iwata, and H. Nishikawa, "Low-Powered Self-Timed Pipeline with Variable-Grain Power Gating and Suspend-Free Voltage Scaling," Proc. The 2013 Int'l Conf. on Parallel and Distributed Processing Techniques and Applications, pp. 618–624, July. 2013.

[7] K. Hirose, Y. Manzawa, M. Goshima, and S. Sakai, "Delay-Compensation Flip-Flop with *In-situ* Error Monitoring for Low-Power and Timing-Error-Tolerant Circuit Design," Japanese J. of Applied Physics, Vol. 47, No. 4, pp. 2779–2787, Apr. 2008.

[8] R.F. Sproull, I.E. Sutherland, and C.E. Molnar, "The Counterflow Pipeline Processor Architecture," IEEE Design & Test of Computers, Vol. 11, No. 3, pp. 48–59, July. 1994.

[9] S. Sannomiya, K. Aoki, M. Iwata, and H. Nishikawa, "Power-Performance Verification of Ultra-Low-Power Data-Driven Networking Processor: ULP-CUE," Proc. The 2012 Int'l Conf. on Parallel and Distributed Processing Techniques and Applications, pp. 465–471, July. 2012.

A Novel Information Sharing Architecture Constructed by Broadcast Based Information Sharing System (BBISS)

Keisuke Utsu[1], **Chee Onn Chow**[2], **Hiroaki Nishikawa**[3], **and Hiroshi Ishii**[1]
[1]School of Information and Telecommunication Engineering, Tokai University, Minato, Tokyo, Japan
[2]Faculty of Engineering, University of Malaya, Kuala Lumpur, Malaysia
[3]Graduate School of Systems and Information Engineering, University of Tsukuba, Tsukuba, Ibaraki, Japan

Abstract - *Existing communication infrastructure may be unavailable in disaster situations. Under the situations, it is difficult to share information composed of multiple packets, such as text, image, and audio data in the communication infrastructure unavailable areas. To enable information sharing without using existing communication infrastructure in the areas, we have proposed a novel system "Broadcast-Based Information Sharing System (BBISS)". The paper evaluates the performance of BBISS by the network simulations. The simulation results can conclude that the proposed method achieves the high information reachability without significantly increasing of the number of packet exchanges.*

Keywords: Broadcast, Ad hoc communication, Information sharing

1 Introduction

Communication carriers equip backup systems and batteries to prevent the communication infrastructure disruptions due to disasters. However, the above approaches taken by communication carriers are not sufficient to endure traffic congestions by confusion owing to the disasters, it is difficult to share the information among victims and to execute the rescue and recovery activities. In the infrastructure unavailable situations, damage and safety information are necessary to be shared by text, image, voice, or video data in the area. The information to be shared is as follows:

1. Announcement of damage information and evacuation instructions: The announcement may be broadcast by voices using microphones and speakers by local governments, polices, or firefighters in usual.

2. Sharing information such as safety information, searching, buzzes in the area: The information may be shared by notices and posters among the victims, the local governments, the police, or the firefighters. However, there is no convenient and efficient alternative way using the notices and posters.

Applications that enable the above information sharing are to disseminate generated information to the whole area. For the objective, we have proposed Broadcast-Based Information Sharing System (BBISS). That is a novel information sharing system which uses ad hoc communication in [1].

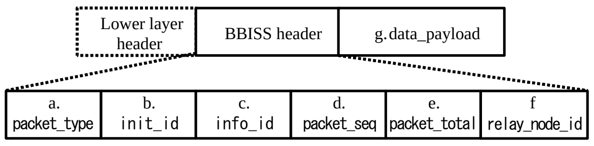
The paper evaluates the performance of BBISS through the network simulations and shows the information sharing architecture constructed by BBISS. The paper is organized as follows. Section 2 explains outlines and problems of existing information delivery methods using broadcast based communication. Section 3 revisits BBISS proposed in [1]. Section 4 shows the effectiveness of the proposed methods through the performance evaluation by network simulations. Lastly, Section 5 concludes the study.

2 Existing information delivery methods using broadcast communication

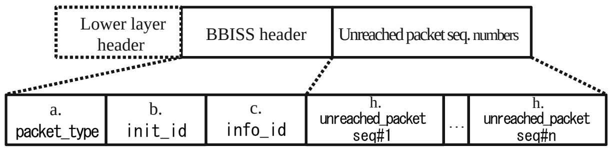
The ad hoc network architecture has been studied as a networking technique in the infrastructure unavailable situations, and many routing protocols have been proposed. In general, available IP addresses must be assigned at nodes in the network to communicate using routing protocols. However, the IP addresses are often not available in the infrastructure unavailable situations owing to the large-scale disaster. Moreover, some servers must be required to collect and disseminate the information to adopt the existing client-server applications. Considering the above problems, the existing ad hoc network architecture cannot be applied to the applications discussed in the Section 1.

2.1 Simple flooding

The Simple Flooding (SF) has been implemented in ad hoc network routing protocols to deliver routing messages in the broadcasting manner [2]. Although SF is one solution to disseminate the information to the area, it has following problems. In this method, when a node receives a packet, the packet is broadcast if it has never received before (non-identical packet). Therefore, since many nodes broadcast the packets and data frame collisions occur, the packet reachability is degraded. Probabilistic scheme and Counter-based scheme have been proposed as the methods without HELLO packet exchanges and dedicated devices such as GPS



(a) Normal and Retransmission packet


Lower layer header
BBISS header
Unreached packet seq. numbers
a. packet_type
b. init_id
c. info_id
h. unreached_packet seq#1
…
h. unreached_packet seq#n

(b) Retransmission Req Packet

Fig.1 Packet format

(Global Positioning System) [3][4]. However, since the above methods do not assume to deliver the information consisting of multiple packets such as data files, they do not take into account the communication reliability of information made from multiple packets.

2.2 Counter-based scheme

In this method, whether to relay or not is determined by how many times an identical packet has been received. The basic procedure is explained below.

1. On receiving a packet, a node sets its counter at 1 if the packet is non-identical to ever received packet. Identical packets are rejected.
2. The counter value is incremented by 1 if an identical packet has been received during an arbitrary period of time (*decision_time*).
3. If the counter reaches a threshold value (*c_threshold*) after expiration of *decision_time*, broadcasting is canceled.

The assumed information types for applications shown in Section 1 among the nodes are file download type information such as text, image, voice, and video, which are composed of multiple packets, in addition to single-packet-messages. Since, the above delivery methods shown in Sections 2.1 and 2.2 cannot complement unreached packets, they are not suitable to the applications.

3 Proposed method, BBISS

Here, we revisit and show the idea of BBISS which we have already proposed in [1] for facilitating understanding.

3.1 Assumed environment

The nodes are wireless communication devices such as smartphones, tablet PCs, or Laptop PCs. The nodes in the area have no available IP addresses and gateway information, and they can communicate only by broadcast communication of IEEE802.11 series wireless LAN. Since the proposed system can be implemented over the general UDP/IP platform by socket programing, the proposed system is easily implemented with flexibility on the existing terminals. The shared information is assumed text information (in several Kbytes size) and image information (in several hundred Kbytes size), both of which consist of multiple packets.

3.2 Requirements

We consider the system needs to meet the following requirements #1 to #3.

- Requirement #1: IP addresses, gateway information, and servers at the nodes are not used.
- Requirement #2: Unreached packets must be complemented without TCP (Transmission Control Protocol) and unicast transmission to assure the communication reliability.
- Requirement #3: The battery charge consumption of the communication terminals should be saved.

3.3 Design of the proposed system

3.3.1 Packet format

Three types of packet formats: Normal packet, Retransmission packet, Retransmission request packet are defined in this method as shown in Fig.1. The packet type is recognized by the "a. Packet type field". The packet formats for the Normal packet and for the Retransmission packet are same except for the "a. Packet type field". The roles of the each field are explained below:

a. Packet type (packet_type), 2Byte: The packet type (Normal packet, Retransmission packet, Retransmission Req packet) is recognized by this field.

b. Initiator node ID (init_id), 8Byte: The ID of the information initiator node, i.e. the information originator, is recognized by this field. Each node must be assigned a unique ID such as a MAC address of a wireless communication interface of the communication terminal.

c. Information ID (info_id), 2Byte: The field shows the ID of the information, which is given by the initiator node, is unique for each initiator node, and may be overlapped with that of other initiator nodes.

d. Packet sequence number (packet_seq), 4Byte: The sequence number of the packet in the information is recognized by the field, which is unique for each information.

e. Packet total (packet_total), 4Byte: The number of packets of which the information is recognized by the field.

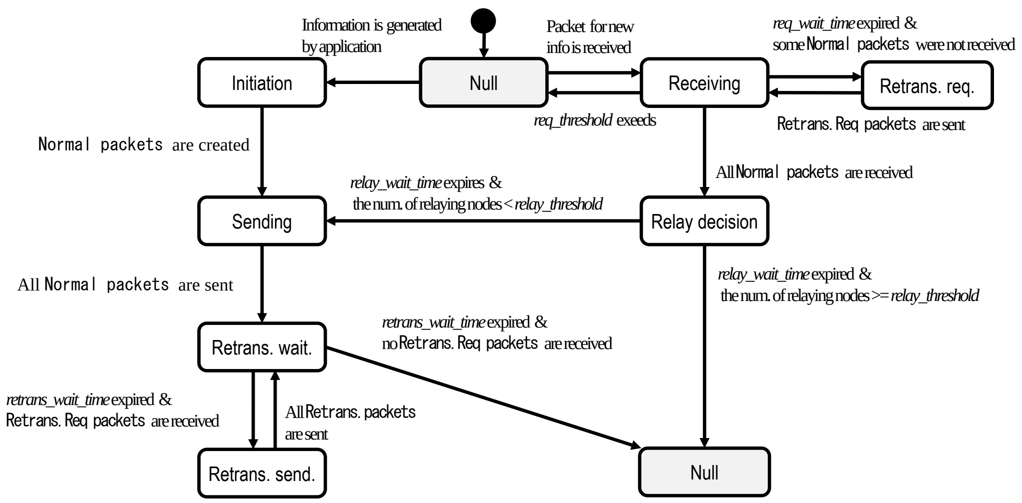

Information is generated by application
Null
Initiation
Packet for new info is received
Receiving
req_wait_time expired & some Normal packets were not received
Retrans. req.
req_threshold exeeds
Retrans. Req packets are sent
Normal packets are created
Sending
relay_wait_time expires & the num. of relaying nodes < relay_threshold
Relay decision
All Normal packets are received
All Normal packets are sent
relay_wait_time expired & the num. of relaying nodes >= relay_threshold
Retrans. wait.
retrans_wait_time expired & no Retrans. Req packets are received
retrans_wait_time expired & Retrans. Req packets are received
All Retrans. packets are sent
Retrans. send.
Null

Fig.2 State flow diagram for BBISS

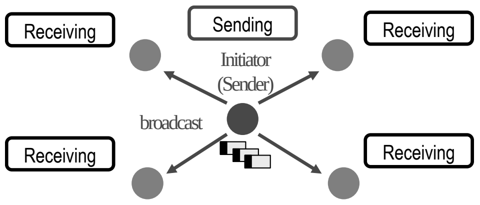


Fig.3 Sending and Receiving states

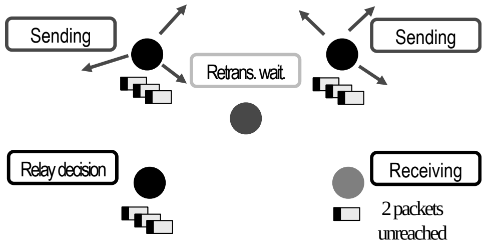


Fig.5 Relay decision and Receiving states

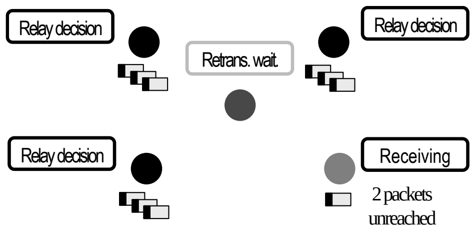


Fig.4 Relay decision and Receiving states

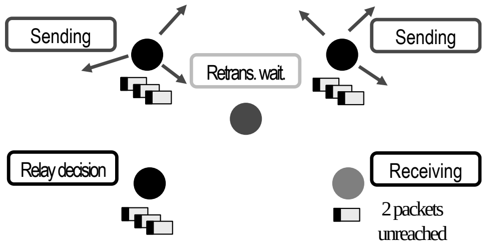


Fig.6 Relay decision and Sending states

f. Relaying node ID (relay_node_id), 8Byte: When a relaying node (that is a node except for the initiator node) relays the packet, the field is overwritten with the node ID of the relaying node. The field is used when a relaying node counts how many other nodes are relaying the information on the Relay decision state described later. Here, the field is empty when the packet is generated at an initiator node.

g. Data (data_payload): Divided data is contained in the field. The size of the field is determined by the parameter *payload_size*.

In addition, Retransmission Req packets contain the following.

h. Unreached packet sequence numbers: If a node finds that some packets are missing for received information, all the sequence numbers of unreached packets are described here, and informed to the neighboring nodes.

3.3.2 Operation of the proposed method

The operation of the proposed method is explained below. The proposed method operates according to the state transition diagram shown in Fig. 2. In the figure, the transition conditions are shown at the side of the arrows. Here, the "transmission" means broadcast transmission.

i, Initiation state: information generation: When a node generates information, the state transits from "Null state" to "Initiation state". Initiator divides information into multiple packets according to *payload_size*. Each packet contains Initiator ID and Information ID. Then, the state transits to "Sending state" after the waiting time which is determined by Random(min, max).

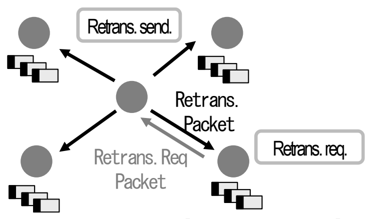


Fig.7 Retrans. req. and Retrans. send. States

Table 1 Parameters for each delivery method

	c_threshold	*relay_threshold*	*send_interval*	Range of Random(min, max)
SF	-	-	0.064 s	(0.064, 0.640) s
C4	4	-		
C3	3	-		
C2	2	-		
Ba4	-	4	0.064 s	(0.064, 0.640) s
Ba3	-	3		
Ba2	-	2		
Ba1	-	1		
Bb4	-	4	0.032 s	(0.032, 0.320) s
Bb3	-	3		
Bb2	-	2		
Bb1	-	1		

ii, Sending state: information sending and relaying: The initiator node sends the packets sequentially with a fixed time interval which is determined by *send_interval*. After sending all the packets, the state transits to "Retrans. wait. state".

iii, Packet receiving state : As shown in Fig.3, when a node received a packet which is a part of information which has not been received yet (That is distinguished by the combination of init_ID and info_ID.), the state transits from "Null state" to "Receiving state". Then the node waits for *req_wait_time* which is determined by the expression (1). Here, *num_of_packets* means the total number of packets for the information, which is described on *packet_total* field in the each packet. Random_max means the maximum value of Random(min, max).

$$req\_wait\_time = 2 \times (send\_interval \times num\_of\_packets) + \text{Random\_max} \quad (1)$$

During the period, if the node received all the packets of the information, the node break *req_wait_time* immediately. Then the state transits to "Relay decision state" shown in Fig.4. If the node does not receive the packets of the information (owing to data frame collisions or other problems) after expiration of *req_wait_time*, the state transits to "Retrans req state" shown in Fig.5 to request to retransmit the unreached packets. Here, the number of times the state transits to "Retrans req state" is limited to *req_threshold*. If the number of times reaches the threshold, the state transits to "Null state". This means that the retransmission requests are not transmitted anymore, and the information receiving is failed.

iv, Relay decision (the node decides whether to relay the information or not): The node waits for *relay_wait_time* which is determined by Random(min, max). During the period, the node counts the number of nodes relaying the same information (relaying node), which is distinguished by the combination of init_ID, info_ID, and relay_node_ID. After the period, if the number of relaying nodes is not reached to *relay_threshold*, the state transits to "Sending state". As shown in Fig.6, the node relays the information.

v, Retrans. req. state: The node waits a period which is determined by Random(min, max). Then the node transmits Retransmission Req packet(s) in which sequence number(s) of unreached packet(s) shown in Fig.7, and the state transits "Receiving state".

vi, Retrans. wait. State: The node waits for *retrans_wait_time* which is determined by the expression (2). If the node receives Retransmission Req packet(s) during the period, the state transits to "Retrans. send. state" after the period. If the node does not receive any Retransmission Req packets during the period, the state transits to "Null state".

$$retrans\_wait\_time = 2 \times num\_of\_pakcets \times send\_interval + 2 \times \text{Random\_max} \quad (2)$$

vii, Retrans. send. State: The node transmits retransmission packet(s) are indicated in retransmit by Retransmission Req pakcet(s) with the fixed interval time *send_interval*. Here, the node retransmits the packet(s) only once evan if it receives multiple Retransmission Req Packet(s), during *retrans_wait_time* period at the previous state "Retrans. wait. state". When the node finishes retransmitting the retransmission packets, the state transits to "Retrans. wait. state".

4 Evaluation by the simulations

Although we have shown the possible effectiveness of our proposal in [1], the evaluation made was preliminary, to make the evaluation mare in detail, this section shows the performance comparison of BBISS with the existing methods through the network simulator OPNET [5].

4.1 Operation of the proposed method

The simulation conditions are described below. The simulation area is defined as 1000m×600m. We assume networks with 100, 200, 400, and 800 nodes. The initial positions of the nodes are set to be random. All nodes move at a speed of [0.00, 4.00] m/s according to the random waypoint model. This model is based on the human walking and running speed. The configuration of each node is described below. The node Mac layer is IEEE802.11b with the data rate of 11 Mbps. The communication range is 150m radius. The information generated by initiator node is delivered by Simple Flooding (SF), Counter-based scheme (CF), or BBISS, using the parameter shown in Table 1. Here, in CF, 3 sets of parameters, C4-C2 are determined, and in BBISS, 8 sets of parameters, Ba4-Ba1 and Bb4-Bb1 are determined. In SF and C2, the interval time of Random(mix, max) is inserted between packets when the node relays a packet. *packet_size* is 1024Byte.

In addition to the above, the following 2 conditions, Simulations #1 and #2, are assumed. In Simulation #1, 5% nodes of all nodes are information initiator nodes. Each initiator node generates one information data of 20kByte (i.e. 20 packets). In Simulation #2, 1% nodes of all nodes are information initiator nodes. Each initiator node generates one information data of 100kByte (i.e. 100 packets). Total number of packets initiated is same between Simulations #1 and #2.

4.2 Evaluated items

The average values of the following items (i)-(v) are found for 100 simulation runs at every value of the random seed. Here, the information data generated by only one initiator node is evaluated, so the information data generated by the other initiator nodes are as background traffic.

(i) The % of the information receiving nodes: The % of nodes that received the information to the total nodes successfully except for the initiator node of the information is shown. The higher the percentage is, the better the performance is.

(ii) The num. of receiving packets in the area: The total number of packets that are received by the nodes in the area is shown. Here "receiving" means that a data frame is received and transferred to upper layer. The smaller the value is, the better performance is.

(iii) (ii) / The number of information receiving nodes: To normalize (ii), (ii) is divided by the number of the information receiving nodes, { (i) x (total num. of nodes) / 100}. The smaller the value is, the better the performance is.

(iv) the num. of transmitted Retransmission Req packets in the area

(v) The average time of information delivery: The time between the information is generated at the initiator node and received at the receiving node (all packets that are composing the information are received) is calculated. The nodes could not receive the information are eliminated. The times at all receiving nodes are averaged. The smaller the value is, the better the performance is.

4.3 Simulation results and discussion

The simulation results are shown in Figs. 8~12.

(i) The % of the information receiving nodes

The simulation results for Simulation #1 and #2 are shown in the Figs. 8 (a) and (b), respectively. The % for BBISS was higher than that for SF and CF regardless of the number of nodes and simulation cases.

The results can be explained as follows. In SF and CF, a node decides whether to relay a receiving packets or not at each time when the node receives the packet. Since the operation caused many redundant relay-transmissions, many packets were lost due to data frame collisions. To make the matter worse, these methods did not equip the function to complement the lost packets. Therefore, these methods proved to have low percentage. On the other hand, BBISS outperformed SF and CF. In BBISS, the node determines on the performance or nonperformance of the information relaying at the time when the node completes the information receiving. The operation made the packet transmissions dispersive. In addition, BBISS performed the retransmission attempts. Therefore, BBISS proved to have the higher percentage than those of SF and CF.

(ii) the num. of receiving packets in the area

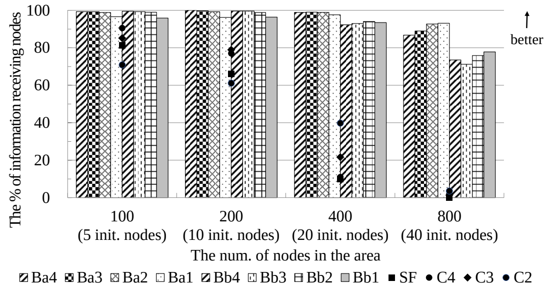
The simulation results for Simulation #1 and #2 are shown in the Figs. 9 (a) and (b), respectively. Although the results in BBISS depended on the parameter setting, some cases for BBISS proved to have the larger number than that for SF and CF. The reason seemed that the (i) for BBISS was increased. The more specific discussion will be shown in the evaluation of (iii).

(iii) (ii) / The number of information receiving nodes

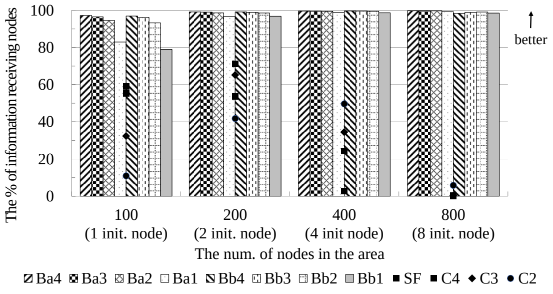
The simulation results for Simulation #1 and #2 are shown in the Figs. 10 (a) and (b), respectively. Ba2, Ba1, Bb3, Bb2, and Bb1 for BBISS performed the same value as the best existing method C2 in both Simulations #1 and #2. Considering the results of (i), Ba2, Ba1, Bb3, Bb2, and Bb1 for BBISS performed high percentage of information receiving nodes without increasing the packet reception (traffic).

(iv) The num. of transmitted Retransmission Req packets in the area

The simulation results for Simulation #1 and #2 are shown in the Figs. 11 (a) and (b), respectively. In the cases where the numbers of nodes in the area were 100 and 200; those were small numbers, or low node density, the smaller the values of *relay_threshold* were, the more the Retransmission Req packets transmitted. The result can be

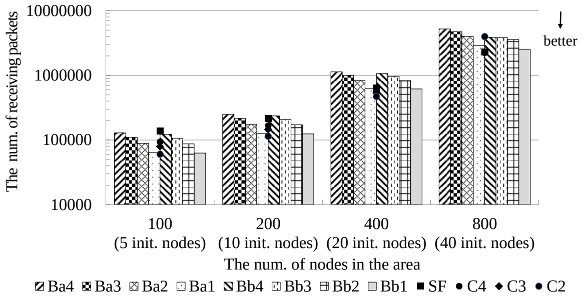


(a) Simulation #1

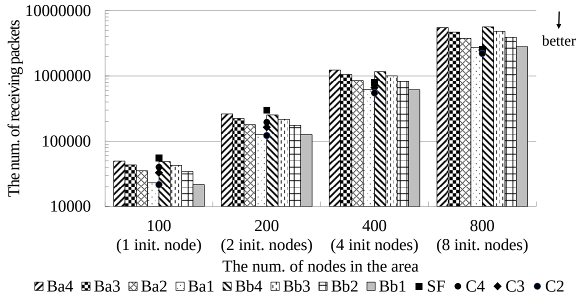


(b) Simulation #2

Fig.8 Simulation result for (i) the percentage of information receiving nodes in the area

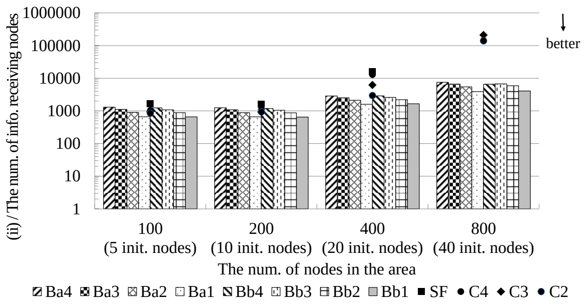


(a) Simulation #1

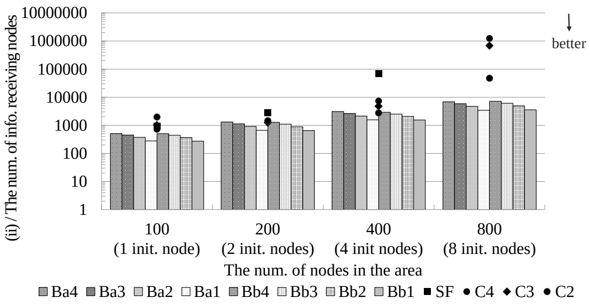


(b) Simulation #2

Fig.9 Simulation result for (ii) the num. of receiving packets in the area



(a) Simulation #1



(b) Simulation #2

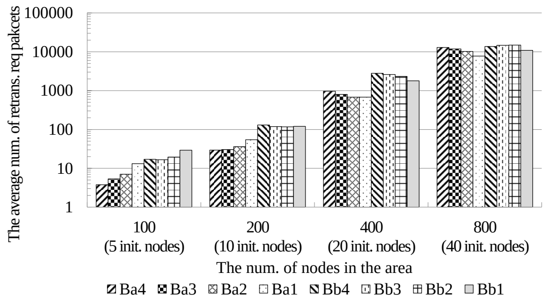
Fig.10 Simulation result for (iii) the num. of receiving packets / the num. of information receiving nodes in the area

explained as follows. Since the node density was low, the cases with the small *relay_threshold* saved relaying, and then the packets were not delivered to the whole area. As the result, the number of Retransmission Req packets was increased. On the other hand, in the cases where the numbers of nodes in the area were 400 and 800; those were large numbers, or high node density, an opposite result to the above was given. The larger values of *relay_threshold* were, the more the Retransmission Req packets were transmitted. The result can be explained as an opposite reason to the above. Since the node density was high, the cases with the large *relay_threshold* did not save relaying, and the relay transmissions performed, and the packet losses due to data frame collisions were increased. As the result, to complement the packet losses, the number of Retransmission Req packets was increased.

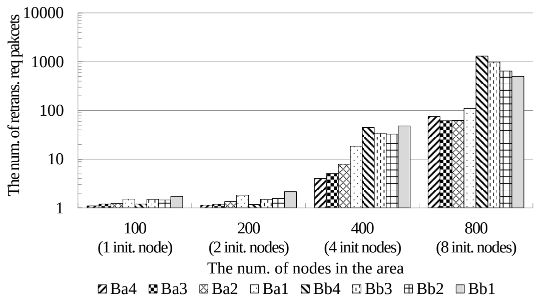
As mentioned in the evaluation (i), since BBISS outperformed the existing method, we can conclude that the retransmission operation on BBISS can contribute to the information reachability. Although we have not discussed the number of Retransmission Req packets specifically, we have to optimize the parameters considering the number of Retransmission Req packets sent in the companion paper. That is because parameters which perform the smaller number of Retransmission Req packet may perform the less unreached packets or the higher information reachability and low traffic load.

(v) The average time of information delivery:

The simulation results for Simulation #1 and #2 are shown in the Figs. 12 (a) and (b), respectively. In Simulation #1 where the generated information size was small (20kByte), the results for Bb4-Bb1 showed the smaller results than the results for Ba4-Ba1 because Bb4-Bb1 set smaller values of Random (min, max) than Ba4-Ba1. Although the results for

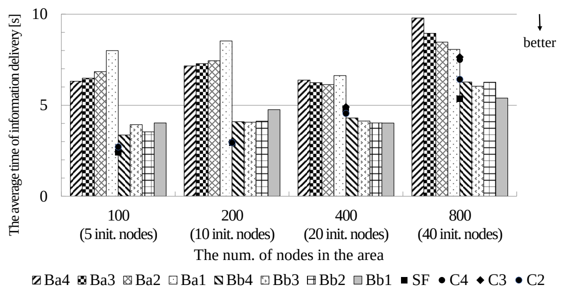


(a) Simulation #1

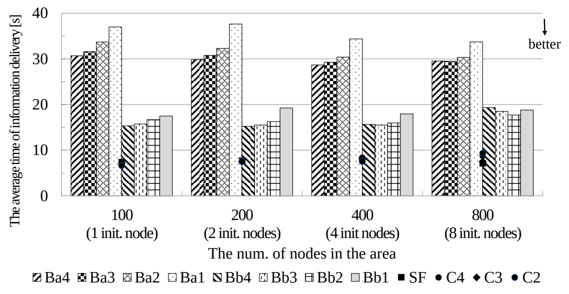


(b) Simulation #2

Fig.11 Simulation result for (iv) the num. of transmitted retransmission req pakcets in the area



(a) Simulation #1



(b) Simulation #2

Fig.12 Simulation result for (v) the average time of information delivary

BBISS were the same as or 1-2 seconds longer than the results for the best existing method

C2, the differences were small, so that can be negligible. On the other hand, in Simulation #2, where the generated information size was large (100kByte), the results for BBISS showed larger than the results for the existing methods. Bb4-Bb1 for BBISS showed 5-10s longer results than the best existing method C2.

Considering the evaluation (i), we can conclude that BBISS showed the longer delivery times than the existing methods in return for the improvement of the information reachability. The shortening the delivery time for BBISS by adjustment of the parameters is our future issue to be tackled.

5 Conclusions

This study evaluates the performance of BBISS through the network simulations. The simulation results can conclude that BBISS outperform the existing flooding methods in terms of the information reachability without increasing on traffic. The parameter optimizations are our future issue to be tackled, some of them are shown in the companion paper.

6 Acknowledgments

This study has been supported by the Telecommunications Advancement Foundation, Japan.

7 References

[1] Keisuke Utsu, Hiroaki Nishikawa and Hiroshi Ishii, "A Proposal on Broadcast based Information Sharing System over Disaster and Congestion Tolerant Ad Hoc Network", the 2013 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'13) , pp.599-603, Jul. 2013

[2] Jorjeta G. Jetcheva, David A. Malts: "A Simple protocol for Multicast and Broadcast in Mobile Ad Hoc Networks", IETF MANET Working Group Internet-Draft, <draft-ietf-manet-simple-mbcast. txt>, 2001

[3] Brad Williams, Tracy Camp: "Comparison of Broadcasting Techniques for Mobile Ad Hoc Networks", Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking and Computing, pp. 194-205 ,2002

[4] Yu-Chee Tseng, Sze-Yao Ni, Yuh-Shyan Chen, Jang-Pinig-Sheu: "The Broadcast Problem in a Mobile Ad Hoc Network", Wireless Networks Volume 8, Springer, pp. 153-167, Kluwer Academic Publishers, 2002

[5] The network simulator "OPNET", http://www.opnet.com

SESSION

SIXTH COMPLEX SYSTEMS, THEORY AND APPLICATIONS WORKSHOP, CSTAW 2014

Chair(s)

Dr. Lou D-Alotto
USA
Dr. Georgios Sirakoulis
Greece
Dr. William Spataro
Italy
Dr. Giuseppe A. Trunfio
Italy

Traffic regime and 1/f noise for a specific approach to a city

R. Thieberger
Department of Physics, Ben Gurion University, Beer Sheva 84105, Israel

(Dated: May 5, 2014)

We examine the traffic lights regime to enable the fastest overall approach to a city for a specific case. The case involves a traffic light where one continues on the main road, into which additional cars are entering at the light. At this intersection an alternative route begins , which is longer but into which no additional cars are entering. To keep the total number of vehicles constant, we subtract on the main road,far from the intersection, the same number of cars as were added at the intersection. We calculate the Fourier transform of the average on each traffic light cycles of the velocity on the main road and bypass. We obtained different results for different cases. All the cases can be written as $1/f^{\alpha}$. *We check by least squares the value of* α. *As changes in acceleration will also influence the noise, we check also the alpha for the change in acceleration.*

1. INTRODUCTION

In a previous study,[3] , we examined a specific traffic problem. We wish in this study to change somewhat the previous assumptions and add the possibility of changing the duration of a certain traffic light. To make our exposition clearer we describe again the procedure given in our previous study. This traffic problem mimics to a certain degree a real situation. We did not try to obtain the actual values as we wish here just to show the feasibility of our approach. The real situation we encounter when entering the city of Beer Sheva, Israel, from the North-East. The specific light we are considering here is governed by the local council of the last suburb. The council decided not to let the through light to be longer than the one going to the suburb. Therefore the main question posed is whether by prolonging the period of going through the suburb(called here "the bypass") one may gain in the overall amount of cars entering the city, although those specific cars going on the bypass may lose time. The main purpose of this paper is to point out the method. We will use elementary cellular automata for our purpose.

Empirical observations of traffic show that at high enough densities the behaviour of traffic becomes quite complex. Therefore, Cellular automata is one of the most used methods for evaluating traffic and that is because of their speed and complex dynamic behaviour. Cellular automata were first studied by Ulam and von Neumann ([2]). An important contribution to the field was in the work of S. Wolfram [1] who introduced classifications, used in the present study. The elementary cellular automaton is a collection of cells arranged on a one dimensional array. Each cell can obtain just two possible numbers: one and zero. The "time" is discreet and at each time step all the cell values are updated synchronously. The value of each cell depends just on the values in the previous step of that cell and its two neighbours. Wolfram names each elementary cellular automaton with a binary numeral, which he calls: "rule". This value results from reading the output when the inputs are lexicongraphically ordered. This will become clearer when we will explain the rules which we use. The rules we used are taken from the cellular automata model as proposed by Gershenson and Rosenblueth[4] .

In addition to velocities and fluxes we are also interested in the power spectrum of the average velocities over a cycle. This value gives us the main contribution to the noise. All the cases can be written as $1/f^{\alpha}$. We obtain three regions for the value of α, before the jammed region, during the onset of the jam and for the denser region. We check by least squares the value of α. We will consider this expression in the section dedicated to calculating the noise.

TABLE I: Wolfram rules used in this model

$t-1$	t_{184}	t_{252}	t_{136}
000	0	0	0
001	0	0	0
010	0	1	0
011	1	1	1
100	1	1	0
101	1	1	0
110	0	1	0
111	1	1	1

2. THE MODEL

We will deal here only with the "microscopic " models were we consider each individual vehicle. Our highways are represented by an array of cells, each cell has the values zero or one. One represents a vehicle and zero an empty portion of the highway. We assume that the magnitude of a cell corresponds to the average length of a vehicle. In figure 1, we show the layout of our model. At a certain point we have a bifurcation where there are two different ways to proceed and at a later point where they merge again. This model represents in a simplistic way the posibility of using two alternative routes (the main route and the "bypass") when approaching a city from a certain direction of suburbs. We add the possibility that additional cars are coming into the main road and are removed when approaching the city. So that overall the number of vehicles is preserved. The rules, which are the same as used by Gershenson and Rosenblueth [4] , are given in Table 1. In figure 2 we give the rules at different locations along our array.

In our analysis we distinguish between four regions:

i. The "bypass region"(denoted by iq).

ii. The region on the main road between the entrance and exit of the "bypass"(denoted by ipe).

iii. The whole of the main road(denoted by ip).

iv. The part of the main road from the second traffic light and on(denoted by t).

2.1. Measures.

The density, ρ, is given by the number of 'ones' (i.e. vehicles) devided by the general number of cells. Initially we take this value to be the same for the three sections. We check how this value changes in the different regions. Here we are interested only in the equilibrium values. The velocities, v, denoted by vp, vq, vpe and vt, are given by the number of cells which change in one step from 0 to 1.

Another measure which interests us in this study is the ratio beween the average time it takes to traverse the "bypass" to the average time it takes on the main road between the two merging points. We will denote this value by 'qdpe'.

In our calculation, space and time are just abstract quantities. Still if concrete numbers are desired, one can quote[4] were one cell represents five meters, and a time step represents a third of a second, which gives us about 50 km/hour, roughly the speed limit within a city.

2.2. The grid

The schematic picture of our specific problem is given in fig 1. A general view of the grid is given in fig 2. The schematic car movement is given in fig 3. We denote the cells on the main route by ip and the cells on the bypass by iq. The cells between $ip = istop$ and $ip = istop1$ we denote by ipe. The cells after $ip = istop1$ we denote by ipt. At $ip = istop$ the vehicles move on the main road or on the bypass according to the 'lights', the time going on the bypass may be longer than the one going straight on the main road. We will check how this influences the overall speed of travel.

In fig3 we show schematically the movements of the vehicles. We have two stop lights (denoted by '1' and '2' on the diagram). When the movement is on the "main road" diagram 'a' gives us the movement. When we enter or exit the "by pass" then 'b' gives us the rules.

We have a parameter telling us the amount of

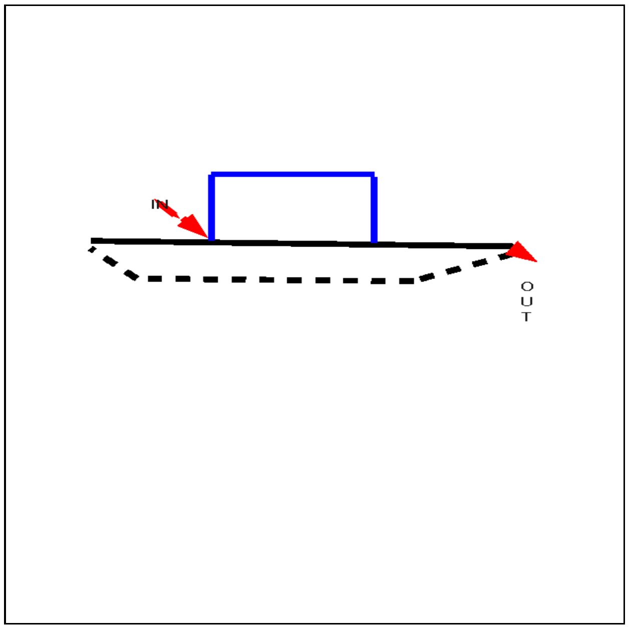


FIG. 1: The movement of vehicles

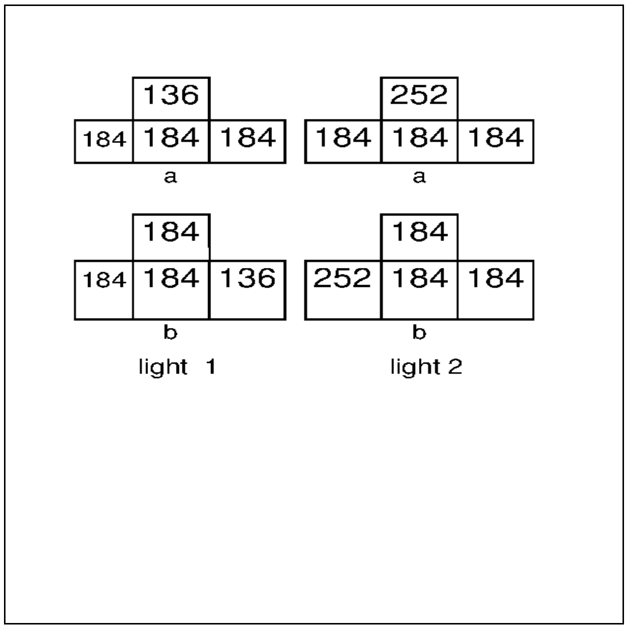


FIG. 3: The movement of vehicles

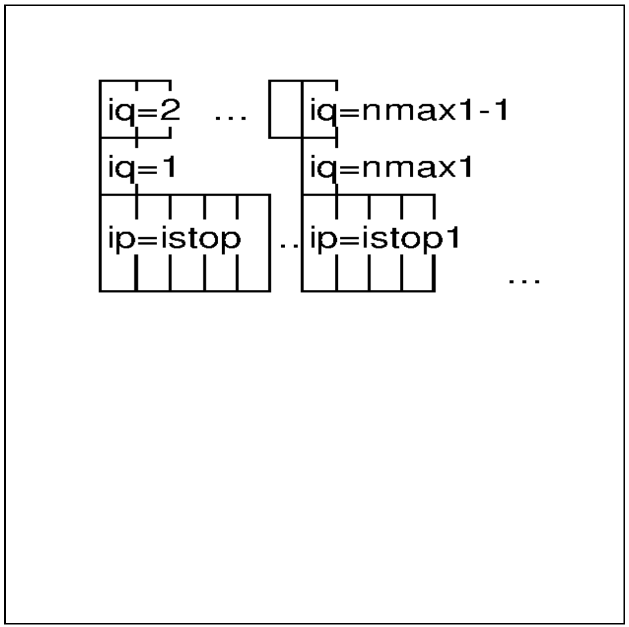


FIG. 2: The grid

"cars" added to the main road at the junction of the bypass. This same amount is deducted from the "main road" farther away and is done in order to preserve the total number of vehicles. The actual addition of cars is governed by a random number which depends on the parameter(i.e. the percentage of cycles when a car is added).

3. NOISE

Traffic noise is one of the most important souces of noise polution. It is well known that this is a health hazard. In this study we wish to check the frequency distribution of the noise. It was shown by Takayasu and Takayasu[7], that we obtain $1/f$ noise. Let us explain here this term: "$1/fnoise$" refers to the phenomenon of the spectral density, $S(f)$, of a stochastic process having the form:

$$S(f) = const./f^{\alpha}$$

When $\alpha = 0$ we say that we have white noise. If $\alpha = 1$ we say we have pink noise. If $\alpha = 2$ we say we have brown noise. To understand better this term see Procaccia and Schuster[5] and Erland et al.[6]. An Indian group[8] made measurements in a number of locations and obtained a mostly pink noise in a large range of frequencies. To obtain $S(f)$ we make the Fourier transform of the velocities. To perform our Fourier transform we take the averages over each light cicle and study the frequencies of these averages over all the cycles taken in our calculations. We compare the results to the $1/f^\alpha$ by a least square test.

4. RESULTS AND DISCUSSION

We used a fixed grid: The main road was comprised of 1200 cells, the "by pass" 300 cells and the distance between the two lights was 120 cells. We used the "green wave" regime. As we have just two lights it was shown by Gershenson and Rosenblueth [4] ,that in this case one does not get different results using the "self-organizing" regime.

We introduce a vehicle on the first intersection for 40% of the steps and we eliminate the same number of vehicles on the last point of our main route, again per unit time.

In Fig 4 we show the change in velocity of the main road, after the second traffic light (vt), as function of the car density. In this case we assume the same duration of the red and green lights at the first intersection.

We see in this figure that the average velocity changes from free flow to the jammed region at about $\rho = 0.6$. In the next figure (Fig5), we show the change of the appropriate flux as function of the density.

We denote by jw-1 the number of additional green light at the first intersection enabling cars to go by the bypass. That means that in Fig4 it is assumed that jw is 1. The maximum value for jw will be 12, as that is the cycle between green and red lights in our calculation. In the next three figures we wish to show the changes of different

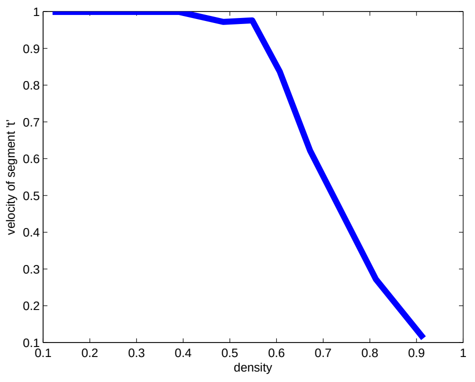


FIG. 4: The change in velocity as function of the density

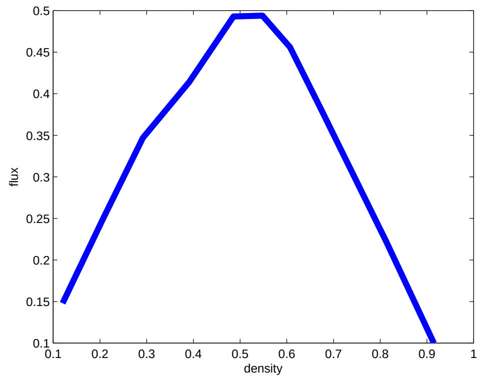


FIG. 5: The change in flux as function of the density

parameters as a function of jw for a specific density, which is at the beginning of the transition from free flow to the jammed region. We chose $\rho = 0.486$. The purpose is mainly ilustrative, but it is similar in other regions.

In figures 6 we show the change in velocities of the by pass(vq) and the velocity between the two traffic lights on the main road(vpe).

In all the cases, we see a strong change at jw=7. Quite clearly, the average velocity on the bypass increases as more cars are going that way and, at the same time, the average velocity on the main

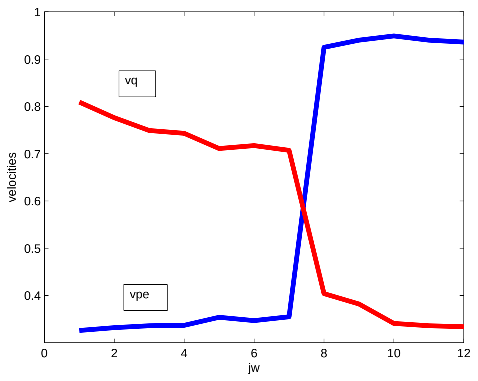


FIG. 6: The velocity on the bypass and betweenthe two traffic lights on main road

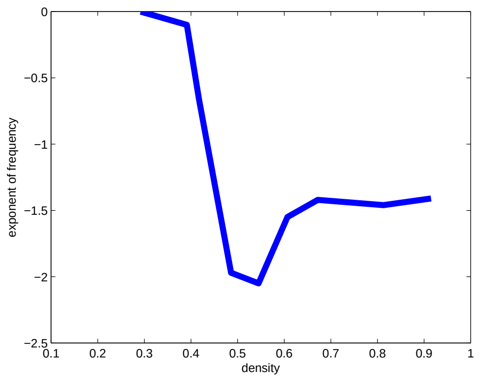


FIG. 8: The values of α as a function of density.

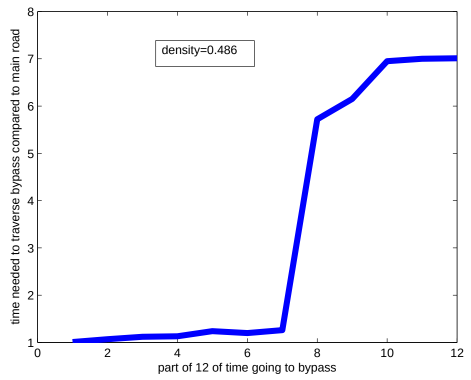


FIG. 7: The relation between the time needed to go by the two routes.

road decreses.

To get a better understanding of the traffic situation we have to check the relation between the times a car needs to arrive at the second traffic light on the bypass and on the main road. This relation is denoted by

$$qdpe = time(q)/time(pe)$$

. In Fig7 we show this relation, and we see again that at jw=7 we obtain a large change.

We averaged the velocities over a traffic light cycle and studied the power spectrum. The value we are interested in is α, in the expression $1/f^{\alpha}$. We give this value in Fig8.

This is an interesting result. When we increase the density so that we reach the transtion from free flow to the jammed region the noise shoots up from white noise to brown noise and then setles in the region of pink noise.

In conclusion, we can say that our calculations give us a wide range of information which can be applied for specific cases so that the traffic light regime can be chosen with much less trial and error than without such a guided approach.

[1] S. Wolphram, 1986, *Theory and Applicationof Cellular Automata., World Scientific.*

[2] *J. von Neumann, 1966, Theory of Self-Reproducing Automata (Edited and completed by Arthur Burks), University of Illinois Press.*

[3] *Reuben Thieberger, 2012, Computer Tech. and Application, Vol. 3, 744-748.*

[4] *Carlos Gershenson and David A. Rosenblueth, 2009, arXiv:0907.1925v1 .*

[5] *I.Procaccia and H.G. Schuster, 1983, Phys. Rev. 28A, 1210-12.*

[6] *S.Erland, P.E.Greenwood and L.M.Ward, 2011, Europhysics Letters, 95,60006.*

[7] *Misako Takayasu and Hideki Takayasu, 1993, Fractals, vol 1, 860-866.*

[8] *K.B.Patange, A.R. Khan, S.H. Behere and Y.H. Shaikh, 2011, Int. Journal of Artificial Life Research, Vol.2, 1-11 .*

Synthetic Earthquakes Obtained with Two Cellular Automata Models and Comparison with Real Seismicity

Alejandro Muñoz-Diosdado
Unidad Profesional Interdisciplinaria de Biotecnología, National Polytechnic Institute, Mexico City, Mexico

Abstract - *The most useful models in the earthquake description are supposed to have a stress distribution, which is usually modeled by means of a cellular automaton with homogeneous distribution. The geological evidence has shown that the Earth crust during an earthquake is broken into fragments in a scale range that goes from millimeters to hundreds of kilometers and it has in its structure a fractal distribution. In this work two cellular automata have been used to describe a seismic fault; properties of the models observed in real seismicity have been obtained, especially the Gutenberg-Richter law.*

Keywords: Cellular automata, earthquakes, seismicity, fractals, Gutenberg-Richter law

1 Introduction

A lot of geological phenomena are scale free. The invariability of the scale is equivalent to a fractal distribution, which requires a power law dependece between the the number of objects of a specific size with the size. The concept of self-organized criticality (SOC) was introduced by Bak et al. [1] as a general organizing principle governing the behavior of spatially extended dynamical systems with both temporal and spatial degrees of freedom. Composite open systems having many interacting elements organize themselves into a stationary critical state with no length or time scales other than those imposed by the finite size of the system. This statistical self-similar behavior is reflected through several empirical power-laws in geology and geophysics [2]. In such a state, a smaller event often begins a chain reaction that can lead to a catastrophe. According to Bak et. al. [1, 3], the temporal "finger print" of the SOC state is the presence of $1/f^{\alpha}$ noise and its spatial signature is the emergence of scale invariant (fractal) structure. The scale invariance is a well-known property of a lot of geological structures and phenomena. This statistical self-similar behavior is reflected through several empirical power-laws in geology and geophysics [2, 4]. The earth's crust can be seen as a hierarchical set of shapes and sizes suitable for a fractal description [5]. The so called Gutenberg- Richter (GR) law for the size distribution of earthquakes is a typical power law of the seismology [7]. In fact, as Bak [7] has asserted, any theoretical or numerical SOC-earthquake model has to reproduce the GR-law as a first proof of its seismic consistency.

In the model of Burridge-Knopoff (BK) [8], the behavior of a real seismic fault is modeled by describing the dynamics of a linear spring-block array using differential equations. Similar BK type models have been extended to 2 and 3− D versions and they have been very successful in reproducing not only the GR law but several other properties of real seismicity [9,10]. Here, several interesting properties of models which are concomitant with properties of real seismicity are shown. A comparison has been made between the model of Olami, Feder and Christensen (OFC) [11, 12] this is a two dimensional non conservative cellular automaton model, and the model proposed by Barriere and Turcotte (BT) [5] that introduced a fractal structure fractal inside the cellular automaton.

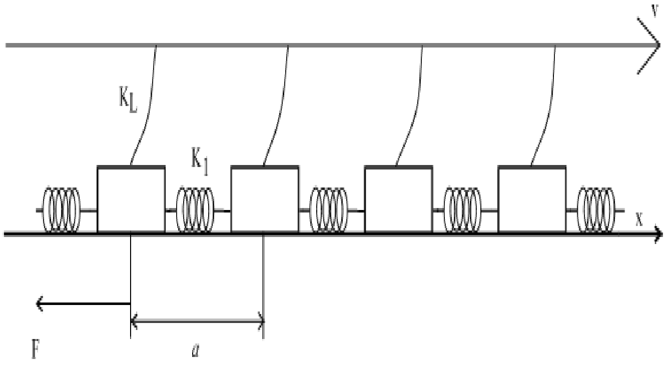


Fig. 1. The one dimensional spring-block model of Burridge and Knopoff (BK), see reference [8].

The earth lithosphere is broken into about dozen mayor rigid plates and several minor ones. These plates slowly grind against each other, building up stress and creating faults. Seismic slip is associated with earthquakes and refers to motion due to a frictional instability between the two sides of a fault. After undergoing seismic slip, the formerly sliding rock experiences an interval of little or no motion during which the stress on the rock recharges. The elastic strain increases monotonically on a fault, resulting in an increase of stress. Once the stress accumulates to the breaking strength, this region becomes unstable and rapidly rebounds or slips to a lower, more stable stress state. After the earthquake subsides, tectonic forces renew the gradual buildup of stress on the fault, eventually culminating in another earthquake [13] and so on.

The Gutenberg-Richter law establishes that the earthquake occurrence frequency is related to the magnitude *m* by means of the relationship

$$log_{10}N(m) = a - bm, \tag{1}$$

where *a* and *b* are constants and *N(m)* is the number of earthquakes larger than *m* in a specific time interval. Although the relationship (1) is universal, the values of *a* and *b* depend on each region. The constant *a* specifies a regional level of seismicity. The values of *b* are approximately between 0.75 and 1.54. A power-law means that a quantity *M* can be expressed as a power of other quantity *s*:

$$M(s) = s^{-\tau} \tag{2}$$

As fractals are characterized by power-law distributions, when *M(s)* versus *log s* is plotted a straight line is obtained. The exponent τ is the straight-line slope. Besides the Gutenberg-Richter law, seismologists have obtained empirically a lot of power-laws in seismology. Aki [14] showed that the Gutenberg-Richter law is equivalent to

$$N \approx A^{-D/2} \tag{3}$$

where *A* is the rupture surface and *D* is the fractal dimension of the seismic fault. A relation between *b* and *D* is obtained as

$$D = 2b \tag{4}$$

Therefore the fractal dimension *D* of the seismic region is two times the *b* value.

2 The OFC and the BT models

The OFC earthquake model is the first example for a supposedly self-organized critical yet non conservative model. The model consists of equal blocks located on a plate, which is supposed to be in the fault (Fig. 2). Each block is connected to its neighbors by harmonic springs and they are lugged individually by other springs subject to other plate that moves with a constant speed. When the upper plate is moved slowly, it causes that the force (or stress) linearly increases in each block, to the point where the force equals a threshold (the force for the fault breaking), after that, the block slips to a state of residual force. A sliding block transfers force to its nearest neighbors, if these neighbors receive sufficient additional force to cause the slipping and so on, it can generate a chain reaction or a synthetic earthquake that is stopped once all the blocks are down of the threshold.

It should be emphasized that the representation of the faults as objects of two dimensions does not imply that the faults are smooth, as objects without structure. However, a lot of structure is included upon discretizing the fault plane. Olami et al. [9, 10] assumed that the block that is moved will slip to the zero force position. An *LxL* arrangement of blocks is defined by *(i, j)*, where *i* and *j* are integers whose values are between *1* and *L* and if the displacement of each block from its relaxed position on the lattice is $x_{i,j}$, then the total force exerted by the springs on a given block *(i, j)* is expressed by [7, 8]

$$F_{i,j} = k_1[2x_{i,j} - x_{i-1,j} - x_{i+1,j}] + k_2[2x_{i,j} - x_{i,j-1} - x_{i,j+1}] + k_L[x_{i,j}] \tag{5}$$

where K_1, K_2 and K_L are the elastic constants. The force redistribution in the position *(i, j)* is given by the following relationship,

$$F_{i\pm1,j} \rightarrow F_{i\pm1,j} + \delta F_{i\pm1,j};\ F_{i,j\pm1} \rightarrow F_{i,j\pm1} + \delta F_{i,j\pm1};\ F_{i,j} \rightarrow 0 \tag{6}$$

where the force increment in the nearest neighbors is given by,

$$\delta F_{i\pm1,j} = \frac{k_1}{2k_1+2k_2+k_L}F_{i,j} = \alpha_1 F_{i,j} \tag{7}$$

where α_1 and α_2, are the elastic ratios. As can be observed the force redistribution is not conservative. The model is homogeneous because the same values of α_1 and α_2 are considered in the entire grid. If $\alpha_1 = \alpha_2$ the model is isotropic. Olami et al. made the mapping of the spring-block model into a continuous, non conservative cellular automaton. The most approximate values to real seismicity are produced for α-values around 0.2. This is reasonable, because if it is assumed that all the elastic constants are in the same scale ($K_1 \approx K_2 \approx K_L$) then $\alpha \approx 0.20$ (see Eqs. 7 and 8). In Fig. 3 it is shown a time series of synthetic earthquakes obtained with this model.

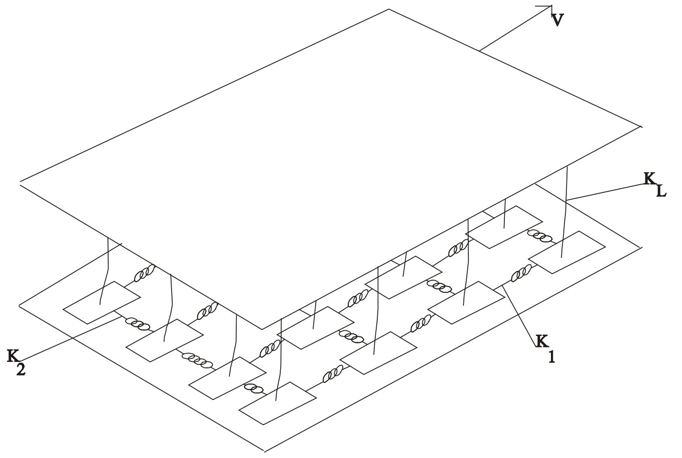


Fig. 2. Geometry of the spring-block model of Olami, Feder and Christensen (OFC).

It has been proposed that a seismic fault model must be able to produce power laws of the type of the GR-law. However, the ability to produce a power law does not mean that the model will be useful, because it also must be able to reproduce other phenomena and lead to features that the seismologists could observe in real faults [13]. The cumulative seismicity was calculated, which was obtained by adding the number of blocks that are relaxed in each one of the events, and then it was plotted it in function of time. Such graphics are stair-shaped plots similar to those of real seismicity. Angulo-Brown and Muñoz-Diosdado [9, 10] have reported that these stair shaped graphics are a characteristic of the model. They found that the synthetic cumulative seismicity in the long-term situation could be bounded by a straight line, whose slope depends on the system size and cannot be arbitrarily large.

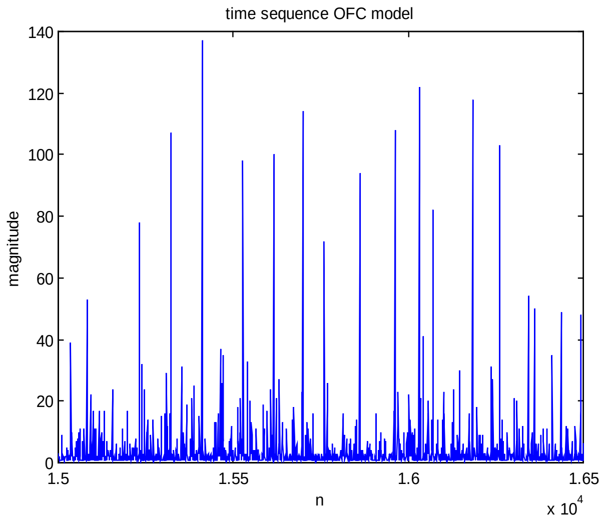


Fig. 3. A time series with 16,500 synthetic earthquakes obtained from dee OFC model with $\alpha = 0.2$.

The basic BT model is a bi-dimensional sand pile model. The standard cellular-automata model has a grid of boxes of equal size. Particles are randomly dropped into these boxes. Barriere and Turcotte [5] considered a grid with a fractal distribution of sizes, each box representing a fault. The random addition of particles to the boxes is analogous to the addition of stress to a zone of crustal deformation. A redistribution of particles from a box is the analog of an earthquake. The number of particles redistributed from a box is a measure of the strength of the synthetic earthquake. Big boxes have big earthquakes; small boxes have small earthquakes. Some of the redistributed particles are lost from the grid and the remainders are transferred to other boxes. This transfer is analogous to the transfer of stress during an earthquake from the fault on which the earthquake occurred to adjacent faults. When the redistribution from a small box results in instability in a big box the instability in the small box is the analog of a foreshock. When redistribution from large boxes triggers instabilities in the smaller boxes, these are the equivalent of aftershocks.

Barriere and Turcotte [5] considered the four models illustrated in the Figs. 4-7. In model 1 a square box is divided into four equal sized boxes at first order. At second order two diagonally opposite boxes are further divided into four boxes. This construction can be extended to any desired order. The Fig. 4 shows the construction of this model at 5th order, the smallest boxes are considered of size one, so at 6th order there is a 32x32 grid. For this case $N_1=64$ boxes of characteristic size $r_1=1$, $N_2=32$ for $r_1=2$, $N_3=16$ for $r_3=4$, $N_4=8$ for $r_4=8$, $N_5=4$ for $r_5=16$ and finally $N_6=2$ for $r=32$. The following equation can be used to calculate the fractal dimension

$$D = \frac{\ln\left(N_{n+1} \big/ N_n \right)}{\ln\left(r_n \big/ r_{n+1} \right)} \tag{9}$$

where N_n is a number of objets (or fragments) with a characteristic linear dimension r_n, using $n=3$ it is obtained $D = \ln(8/16)/\ln(4/8) = 1$.

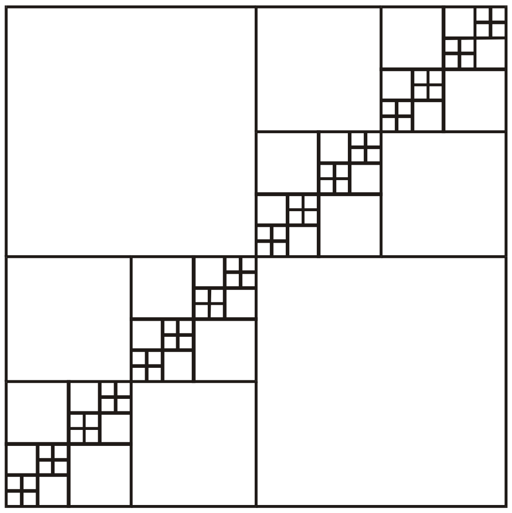


Fig. 4. Illustration of fractal cellular automata model numer one.

Model 2 illustrated in the Fig. 5 is a variation of model 1 with the same fractal dimension. In model 3 the square box is again divided into four equal sized boxes at first order. But at second order only one box is retained and three are further divided into four boxes. It is shown at 5th order the third model in Fig. 6, in this case, $N_1=108$ for $r_1=1$, $N_2=27$ for $r_2=2$, $N_3=9$ for $r_3=4$, $N_4=3$ for $r_4=8$, and $N_5=1$ for $r_5=16$. Therefore with $n=3$ the fractal dimension of third model is $D = \ln(3/9)/\ln(4/8) = \ln 3/\ln 2 = 1.585$. In model 4 a square box is divided into nine equal sized boxes at first order. At second order, three boxes along a diagonal are retained and the other six boxes are further divided into nine boxes. This model is showed in Fig. 7 at 3th order, in this case $N_1=324$ for $r_1=1$, $N_2=18$ for $r_1=3$, and $N_3=3$ for $r_3=9$. The fractal dimension of the fourth model is $D=\ln(3/18)/\ln(3/9)=\ln 6/\ln 3= 1.6309$.

Barriere and Turcotte applied the following three cellular-automata rules to their four models: (i) Particles are added one at a time to randomly selected boxes. The probability that a particle is added to a box is proportional to the area A of the box. (ii) A box becomes unstable when it contains 4A particles. (iii) They considered two alternative rules for redistribution. In the first, particles are redistributed to immediately adjacent boxes or are lost from the grid. The number of particles redistributed to an adjacent box is proportional to the linear dimension of the box. In the second rule, particles are redistributed into the four adjacent regions that have the same size as the unstable box. The number of particles redistributed to a box is proportional to the area of the box. (iv) If after a redistribution of particles from a box any of the adjacent boxes are unstable, one or further redistributions are carried out. In any redistribution the critical number of particles is redistributed. Redistribution is continued until all boxes are stable.

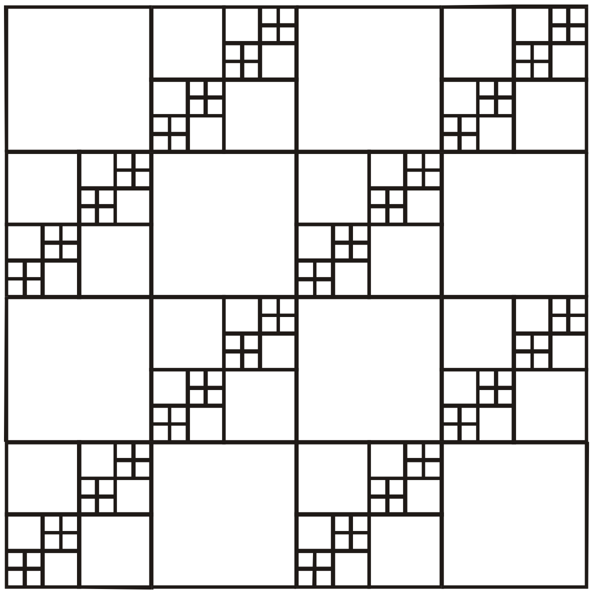


Fig. 5. Illustration of fractal cellular automata model numer two.

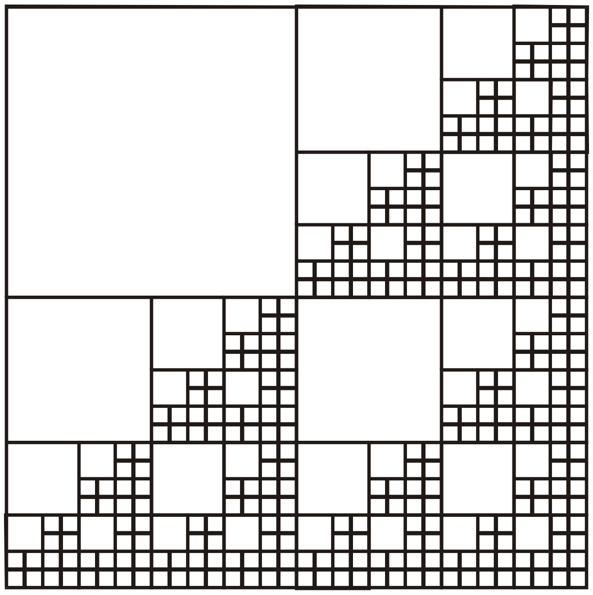


Fig. 6. Illustration of fractal cellular automata model numer three.

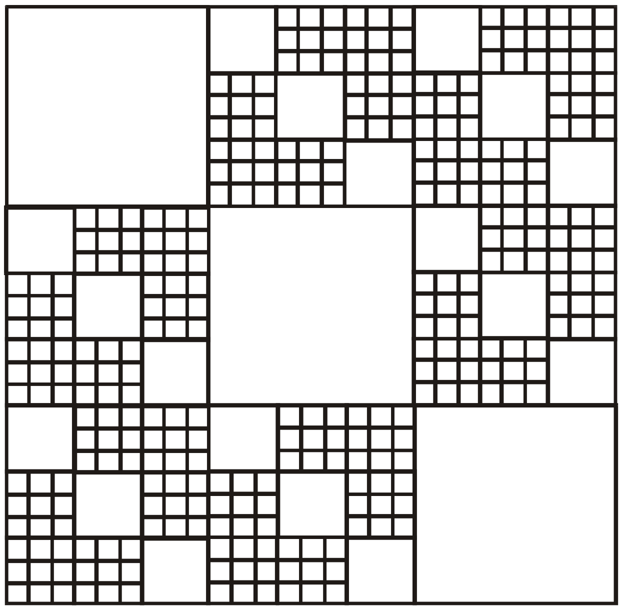


Fig. 7. Illustration of fractal cellular automata model numer four.

The size of the total event is defined as,

$$E = \sum_{i=1}^{N} f_i A_i \qquad (1)$$

where *Ai* is the area of the region that becomes unstable, *fi* is the frequency or number of times that a region of area *Ai* participates during the event, the smallest box is supposed to have size one. The magnitude *M* of the synthetic event is defined as *M* = log (*E*).

3 Results

In the OFC model the probability distribution of earthquake magnitude frequency was obtained and with these graphics the values of the *b* exponent of the GR law were obtained. The distribution forms a straight line that is extended in several orders of magnitude, before the line is curved downward due to the finite size effect (see Fig. 8). This is similar to the real GR law as it is shown for instance in Fig. 9, the GR law for the region in the Mexican Pacific south coast that goes from Colima and Jalisco states to the Oaxaca state passing by Michoacán and Guerrero states. After that, the cumulative seismicity was calculated to investigate if in this case a straight line bounds the stair-shaped graphics in the long-term situation. Actually the staircase-shaped plots were obtained (Fig. 10).

The stability for the cumulative seismicity stair-shaped graphs in the long-term situation was obtained and this means that there are straight line slopes that are superior bounds of the staircases. Actually, we have observed these kinds of plots in the seismological zones of Oaxaca, Guerrero, Michoacán and Jalisco-Colima in Mexico [15]; all of them have been characterized by one value of the *b* exponent of the GR-law.

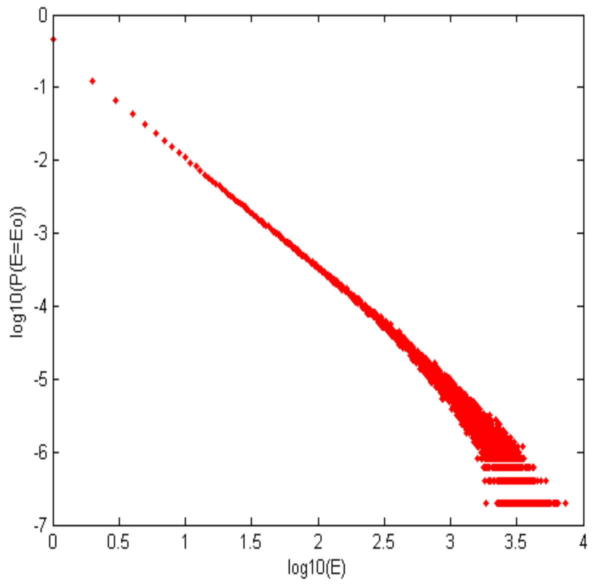


Fig. 8. Distribution of synthetic earthquake magnitude in a 100x100 system with open boundary conditions. The network is divided into two parallel regions with α -values, 0.225 and 0.175, respectively, 1000,000 events.

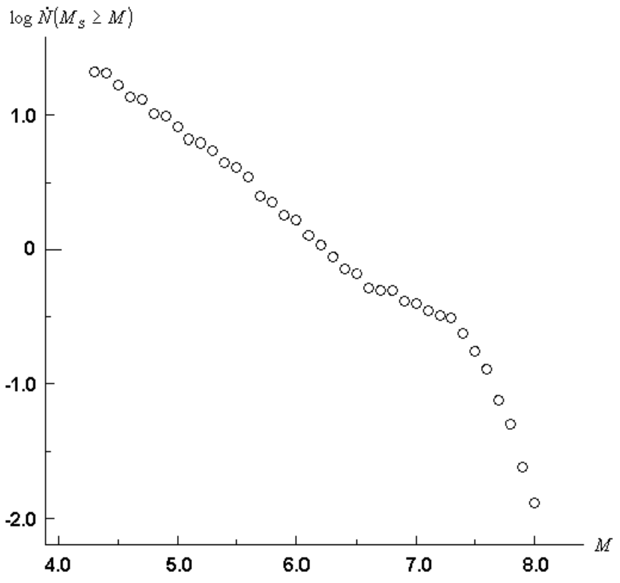


Fig. 9. Real GR law for the seismogenic region of the south of the Mexican Pacific coast, note at the end the effect of finite size

As can be seen in the plots as the one shown in Figs. 10 (Colima- Jalisco region), 11 (Michoacán region), 12 (Guerrero region) and 13 (Oaxaca region) it seems that they can be characterized also by the value of the slope m_f of a long-term straight line. It means that, for real earthquakes, as in the synthetic seismicity, in the long-term behavior the envelope of the stair-shaped plots seems to tend to a straight line, so when quiescence is produced the plot tends to return to the historical slope of the seismological region.

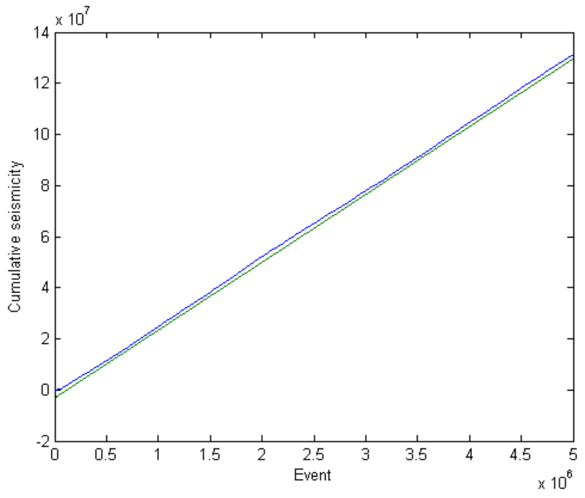


Fig. 10. Cumulative synthetic seismicity. The structure of the stair-shaped plots can be seen when less events are plotted.

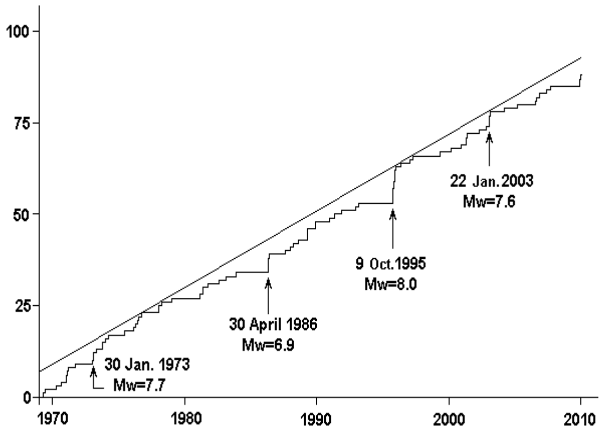


Fig. 11. Number of earthquakes with $Ms \geq 4.3$, for the Colima-Jalisco region in the Mexican Pacific coast, between 17.8 and 19.8º lat. N and 103 and 105.8º long. W, from January 1, 1969 to April 30, 2010, with depths less or equal to 60 km. Note the two precursory seismic quietude.

The cellular automata with fractal structure were programmed for the four models proposed by Barriere and Turcotte using the rules described in the previous section, we considered fourth to eighth order. In these models the patterns of synthetic seismicity are similar to the patterns of real seismicity. The cumulative seismicity has also a staircase shape that is bounded by a straight line. In table I the m_f slope values obtained for each one of the fractal models with different orders are summarized.

For this model it was also carried out the frequency and magnitude statistics and it was found that these fractal models reproduce the Gutenberg-Richter law. For instance, the analysis results for the second fractal, sixth order are shown in Fig. 8, the slope of the straigth line b =0.49± 0.04, this result is in agreement with the equation (1).

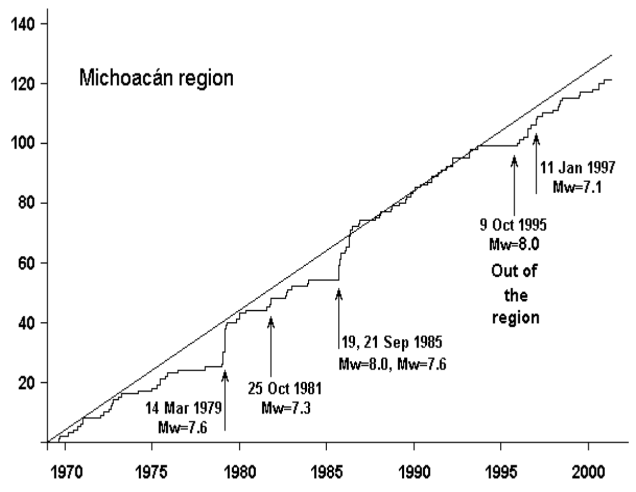


Fig. 12. Number of earthquakes with $Ms \geq 4.3$, for the Michoacán region in the Mexican Pacific coast.

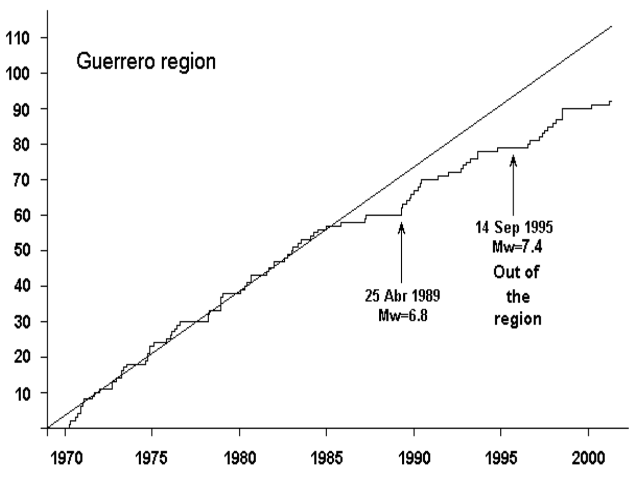


Fig. 13. Number of earthquakes with $Ms \geq 4.3$, for the Guerrero region in the Mexican Pacific coast.

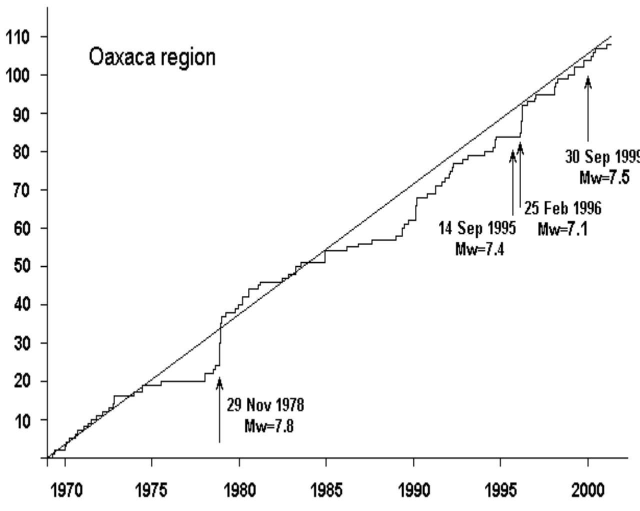


Fig. 14. Number of earthquakes with $Ms \geq 4.3$, for the Oaxaca region in the Mexican Pacific coast.

TABLE I. SLOPE M_F OF THE STRAIGHT LINE ASSOCIATED TO THE STAIR GRAPHICS OF THE ACCUMULATED SEISMICITY FOR EACH ONE OF THE FRACTAL AUTOMATON MODELS WITH DIFFERENT ORDERS, D IS THE FRACTAL DIMENSION

Model	**D**	**Order**						
		2	3	4	5	6	7	8
1	1	-	1.10	1.39	1.68	1.93	2.23	2.49
2	1	-	2.94	3.86	4.76	5.36	6.63	7.58
3	1.58	-	1.93	3.73	7.29	13.8	26.0	49.0
4	1.63	2.7	8.28	23.8	67.5	-	-	-

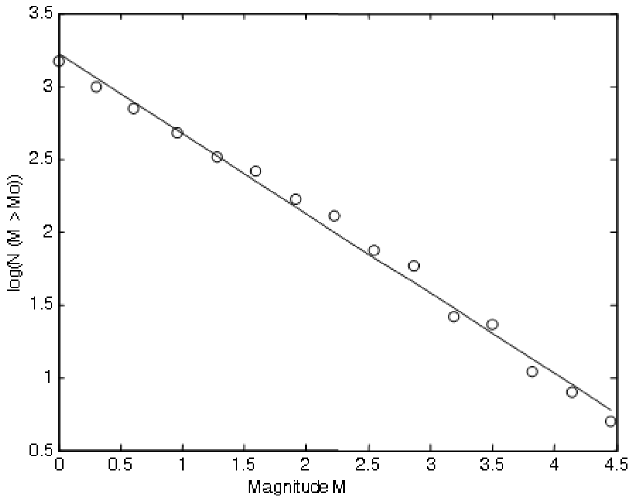


Fig. 15. Statistical distribution of the frequency logarithm and the magnitude for a synthetic seismicity pattern obtained with the second model, sixth order. Note the agreement with the Gutenberg-Richter law with $b = 0.49 \pm 0.04$.

In Table II, the values of the *b* parameter for each one of the used models are shown. It can be observed in Table II that for the models 1 and 2 the values of *b* are close to the value 0.5, which approximately is one half of their respective fractal dimension *D=1*, therefore for the fractal models 1 and 2 they fulfill the Aki [12] relationship *D=2b* quite well. However, for the fractal models 4 and 5 where the respective dimensions are *D =1.585* and *D =1.631* a relationship *D< 2b* is observed.

TABLE II. VALUES OF THE COEFFICIENT *B* OF THE GUTENBERG-RICHTER LAW

Model	**D**	**Order**						
		2	3	4	5	6	7	8
1	1	-	0.54	0.54	0.53	0.50	0.50	0.50
2	1	-	0.49	0.49	0.48	0.48	0.48	0.47
3	1.58	-	0.67	0.67	0.55	0.55	0.54	0.54
4	1.63	0.66	0.59	0.56	0.81	-	-	-

4 Conclusions

The OFC and BT models qualitatively reproduce many of the properties observed in real seismicity, so they are good models to model seismic faults. Catalogues of synthetic seismicity have been obtained with similar properties to the catalogues of real seismicity, based in the characteristics of this fractal model it can be proposed that this catalogues can be used to study aftershocks and foreshocks. The results are similar to the ones obtained with real data, so this research is in the correct way to have in the future a more appropriate model for a seismic fault based on a spring block model.

5 References

[1] P. Bak, C. Tang, and K. Weisenfeld, “Self organized criticality: An explanation of 1/f noise,” Phys. Rev. Lett. 59, 381-384, 1987.

[2] K. Ito, and M. Matsuzaki, “Earthquakes as a self organized critical phenomena,” J. Geophys. Res. 95(B5), 6853-6860, 1990.

[3] P. Bak, C. Tang, and K. Weisenfeld, “Self-organized criticality”, Phys. Rev. A, 38(1), 364-374, doi: 10.1103/PhysRevA.38.364, 1988.

[4] D. L. Turcotte, “Fractals and chaos in geology and geophysics”, Cambridge University Press, Cambridge, 1997.

[5] B. Barriere, and D. L. Turcotte, “Seismicity and self-organized criticality,” Phys. Rev. E 49(2), 1151-1160, 1994.

[6] B. Gutenberg, and C. F. Richter, “Frequency of earthquakes in California,” Bull. Seismol. Soc. Am. 34, 185-188, 1944.

[7] P. Bak, How Nature Works, Springer Verlag, New York, 1996.

[8] R. Burridge, and L. Knopoff, “Model and theoretical seismicity,” Bull. Seism. Soc. Am. 57, 341-371, 1967.

[9] F. Angulo-Brown, and A. Muñoz-Diosdado, “Further seismic properties of a spring-block earthquake model”, Geophys. J. Int. 139(2), 410- 418, 1999.

[10] A. Muñoz-Diosdado, and F. Angulo-Brown, “Patterns of synthetic seismicity and recurrence times in a spring-block earthquake model,” Rev. Mex. Fís. 45(4), 393-400, 1999.

[11] K. Christensen, and Z. Olami, “Scaling, phase transitions, and nonuniversality in a self organized critical cellular automaton model,” Phys. Rev. A 46, 1829-1838, 1992.

[12] Z. Olami, H. J. S. Feder, and K. Christensen, “Self organized criticality in a continuous, nonconservative cellular automaton model,” Phys. Rev. Lett. 68, 1244-1247, 1992.

[13] C. D. Ferguson, W. Klein, and J. B. Rundle, “Long-range earthquake fault models,” Comp. Phys. 12, 34-40, 1999.

[14] K. Aki, “A probabilistic synthesis of precursor phenomena” *in Earthquake Prediction, Maurice Ewing series*, 4, eds Simpson, , D. W. & Richard, P. G., AGU, Washington. , pp 566-575, 1981.

[15] A. H. Rudolf-Navarro, A. Muñoz-Diosdado, and F. Angulo-Brown, “Seismic quiescence patterns as possible precursors of great earthquakes in Mexico,” International Journal of Physical Sciences 5(6), 651-670, 2010.

Parallelization of an Iterative Method for Solving Large and Sparse Linear Systems using the CUDA-Matlab Integration

Lauro Cássio Martins de Paula,
Anderson da Silva Soares
Institute of Informatics
Federal University of Goiás
Goiânia, Brazil
{lauropaula, anderson}@inf.ufg.br

***Abstract*—This paper presents a parallel implementation of the Hybrid Bi-Conjugate Gradient Stabilized (BiCGStab(2)) iterative method in a Graphics Processing Unit (GPU) for solution of large and sparse linear systems. This implementation uses the CUDA-Matlab integration, in which the method operations are performed in a GPU cores using Matlab built-in functions. The goal is to show that the exploitation of parallelism by using this new technology can provide a significant computational performance. For the validation of the work we compared the proposed implementation with a BiCGStab(2) sequential and parallelized implementation in the C and CUDA-C languages. The results showed that the proposed implementation is more efficient and can be viable for simulations being carried out with quality and in a timely manner. The gains in computational efficiency were 76x and 6x compared to the implementation in C and CUDA-C, respectively.**

Keywords: Matlab, GPU, CUDA, BiCGStab(2).

I. Introduction

A linear system is a linear equations finite set applied in a variable finite set. Sparse and large linear systems may appear as result of the modeling of various computer science and engineer problems [18]. To solve such systems, iterative methods are more indicated and efficient than exact methods [20]. Iterative methods use less memory space and reduce rouding errors in computer operations [15]. Such methods perform successive approximations in each iteration to obtain a more precise solution for the system.

Classical iterative methods such as Jacobi and Gauss-Seidel are considered easy to deploy and use [17]. Nevertheless, despite this feature both may have a slow convergence or even not converge for large systems [20]. Another disadvantage is that when the coefficient matrix is not square (number of rows equal to the number of columns), these two methods can not guarantee the linear system convergence. As a consequence, the research and implementation of computational methods are considered important tasks in various areas of science, particularly those that involve the solution of large linear equations systems [6].

There are several methods for solution of linear systems. Some of them are considered good in relation to the computational cost. However, the computational performance may be affected if the size of the system is large. In some cases in which the linear systems to be solved are very large, the computational processing may last too many days and the methods solution speed difference are significant. Consequently, the implementation of efficient and robust methods such as the Hybrid Bi-Conjugate Gradient Stabilized (BiCGStab(2)) becomes important and often necessary for the simulations are performed with quality and in a short time [2]. The BiCGStab(2) is an iterative method developed for solving large and sparse linear systems and is considered a good one [6].

Several studies have used the computational resources of Graphics Processing Units (GPU) to solve large and sparse linear systems. For instance, Bowins [2] presented a comparison of computational performance between the Jacobi method and the Bi-Conjugate Gradient Stabilized (BiCGStab) method. In that work, both methods were implemented in two versions: sequential and parallelized. Based on the results obtained, he showed that as the size of the system increases, the parallel implementation outperforms the sequential in terms of computational efficiency.

Weber *et al.* [21] presented graphics processing unit (GPU) data structures and algorithms to efficiently solve sparse linear systems that are typically required in simulations of multibody systems and deformable bodies. Their solving method results in a speedup factor of up to 13 in comparison to other sequential and GPU methods.

More recently, Paula *et al.* [6] proposed a parallelization of the BiCGStab(2) method for solving linear systems using Compute Unified Device Architecture (CUDA) and compared the computational performance between the sequential and parallelized versions of the method. They showed that from the computational point of view, the parallel version of BiCGStab(2) method is more efficient.

In this context, this paper presents a parallel implementation of the BiCGstab(2) method, which uses the CUDA-Matlab technology in a GPU for solving linear systems. The goal was showing that the proposed implementation can be more appropriate and, through its use, it is possible to enable the efficient solution of large and sparse linear systems for increasingly complex (larger) systems can be solved in a timely manner. To achieve this goal, we performed a comparison with the implementation of the BiCGStab(2) method proposed by Paula *et al.* [6] in the solution of linear systems of varying sizes. The results showed that the computation time can be significantly reduced with the implementation proposed in this paper. It was possible to obtain speedup gains of 76x and 6x compared with the sequential and parallelized implementation proposed in [6], respectively.

The remainder of this paper is organized as follows. It is detailed in Section II the BiCGStab(2) iterative method. Section III describes the CUDA and its integration with Matlab. The materials and methods used to achieve the objective of the work are described in Section IV. The results are presented and discussed in Section V. Finally, Section VI contains the conclusions.

II. BiCGStab(2) Method

The solution of a linear equations system $Ax = b$, where $A_{n\times n}$ is the coefficient matrix and $b_{n\times 1}$ the vector of independent terms, may require a huge computational effort especially when A is very large. For example, to solve a linear system one can use an iterative method. Iterative methods perform successive approximations in each iteration to obtain a more accurate solution and are recommended for large linear systems with sparse matrices [1].

Iterative methods are classified into two groups: stationary and non-stationary methods [6]. The stationary methods use the same information at each iteration, $i.e.$, the results of one iteration are used for the next iterations [18]. In non-stationary methods, the information used may change with each iteration. The non-stationary methods are difficult to implement but may provide a faster convergence for the system and are more suitable even when the coefficient matrix is dense (non-sparse) [20].

The BiCGStab(2) is a non-stationary iterative method developed by van der Vorst and Sleijpen [18]. This method combines the advantages of BiCGStab and Generalized Minimum Residual (GMRES) method [14]. Consequently, the BiCGStab(2) is considered a robust method and with convergence guarantee superior to BiCGStab, suitable for solution of linear systems generated in the solution of differential equations of fluid flow [18].

Algorithm 1 shows a snippet of pseudocode for the algorithm of BiCGStab(2) method. A full pseudocode can be obtained in [6]. Some adjustments were made naming comparing with the original algorithm. In the Algorithm 1, the Greek letters represent scalars, lowercase letters represent vectors expressed in matrix form, capital letters represent matrices, and parentheses with comma separated vectors represent scalar products between vectors.

In step 38 of the method, so that the vector x_{i+2} is sufficiently precise, the higher value corresponding to the difference between the results of each term of the vector x in two consecutive iterations, divided by the result of the term in the current iteration, should be less than a given accuracy as, for example, $max(\frac{x_i-(x_{i-1})}{x_i}) < 10^{-5}$.

Algorithm 1: Snippet of pseudocode for the algorithm of BiCGStab(2) method.

```
 1. $r_0$ = b - A$x_0$
 2. $\hat{r}_0$ = $r_0$
 3. $\rho$ = $\alpha$ = $\omega_1$ = $\omega_2$ = 1
 4. v = w = p = 0
 5. for $i$ = 0, 2, 4, ... do
 6.    $\hat{\rho}$ = -$\omega_2\rho$
       Even BiCGStab step: from step 7 to 16
 7.      $\rho$ = $r_i^T\hat{r}_0$ ...
16.      $x_i$ = $x_i$ + $\alpha$p
       Odd BiCGStab step: from step 17 to 27
17.      $\rho$ = $s^T\hat{r}_0$ ...
27.      t = As
       GMRES(2)-part: from step 28 to 38
28.      $\omega_1$ = $r^T$s ...
36.      $x_{i+2}$ = $x_i$ + $\alpha$p + $\omega_1$r + $\omega_2$s
37.      $r_{i+2}$ = $r_i$ - $\omega_1$s - $\omega_2$t
38.      If $x_{i+2}$ is accurate, stop.
39. end for
```

III. CUDA

Compute Unified Device Architecture (CUDA) was the first Application Programming Interface (API), created by $NVIDIA^{®}$ in 2006, to allow the GPU could be used for a wide variety of applications [4]. CUDA is supported by all graphics cards from $NVIDIA^{®}$, which are extremely parallel, having many cores with many memories and a memory cache shared by all cores. The CUDA code is an extension of the C computer language (CUDA-C), where a few keywords are used to label the parallel functions (kernels) and their data structures [3].

Since its inception, several studies have used CUDA to parallelization of various types of problems. For instance, Yldirim and Ozdogan [22] presented an algorithm as a clustering approach based on wavelet transform for parallelization on GPU using CUDA-C. Fabris and Krohling [9] proposed an algorithm of evolution implemented in CUDA-C for solving optimization problems. Atasoy *et al.* [1] presented a eliminating method implemented in CUDA-C using Gauss-Jordan to solve systems of linear equations. Paula *et al.* [6] used CUDA-C to parallelize the BiCGStab(2) method for solving linear systems of varying sizes. Finally, Paula *et al.* [8] proposed a parallelization strategy for phase 2 of the Successive Projections Algorithm using CUDA-C.

In order to help programmers, the $MathWorks^{®}$ has developed a plugin able to do integration between CUDA and Matlab. Make use of Matlab to GPU computing can enable applications to be more easily accelerated. GPUs can be used with Matlab using the Parallel Computing Toolbox (PCT). The PCT provides an efficient way to speedup codes in Matlab language, running them on a GPU [11], [7]. For this, the programmer must change the data type to input a function to use the commands (functions) in Matlab that were overloaded ($GPUArray$). Through $GPUArray$ function one can allocate memory in the GPU and make calls to various functions of Matlab, which are performed on the GPU's processing cores. Additionally, developers can make use of the PCT $CUDAKernel$ interface to integrate their code in CUDA-C with Matlab [13].

The development of applications running on the GPU using the PCT is usually easier and faster than using CUDA-C language [12]. According to Little and Moler [11], this is because aspects of exploitation of parallelism are implicitly performed by the PCT itself, freeing the programmer from many inconveniences. However, the organization and the number of threads to be executed on the GPU cores can not be managed manually by the programmer. Still, it is important to emphasize that in order to be used, the PCT requires a graphics card from $NVIDIA^{®}$.

After CUDA-Matlab integration, few studies have used this technology. For example, the $NVIDIA^{®}$ [5] released a book that demonstrates how programs developed in Matlab can be accelerated using GPUs. Simek and Asn [16] presented an implementation in MATLAB with CUDA for compression of medical images. Kong *et al.* [10] accelerated some functions in Matlab for image processing on GPUs. Reese and Zaranek [13] developed a manual programming GPUs using Matlab. More recently, Paula *et al.* [7] proposed a parallel implementation of the Firefly Algoritm using CUDA-Matlab for variable selection in a multivariate calibration problem. Based on the results of these works, we note that, in future, the PCT may be more used due to the fact of allowing a code in Matlab can be easily parallelized. Therefore, instead of implementing a kernel function and set the amount and organization of threads blocks, the programmer must only identify which parts of your code are parallelizable and make use of the built-in Matlab functions.

IV. Experimental

The GPU was initially developed as a flow-oriented technology, optimized for calculations of data-intensive applications , where many identical operations can be performed in parallel on different data [4]. Unlike a Central Processing Unit (CPU), which executes only a few threads in parallel, the GPU was designed to run thousands of them [8].

As previously mentioned, one can explore parallelism in GPUs using the PCT plugin, which provides an efficient way to speedup codes in Matlab language invoking functions that are overloaded to run in the cores of a GPU from $NVIDIA^{®}$. Thus, this paper presents an implementation of the BiCGStab(2) method in Matlab, which uses this technology. The proposed implementation is analogous to Algorithm 1. Initially, the data are transferred to the GPU memory. Soon after, the method begins execution and all operations are performed in the GPU's processing cores for threads that are created and managed implicitly by the PCT.

All the linear systems used in this paper were generated using Matlab (version R2013a) built-in functions. The coefficient matrix (A) of each system was generated randomly using the function $gallery('dorr', n)$, which returns a square matrix of dimension n, sparse and diagonally dominant. The diagonal dominant characteristic indicates that the sum of all elements in a row is not greater than the main diagonal element of the matrix. The vector of unknowns (x) was randomly generated by $randn(n, 1)$ function, which returns a vector of n rows and 1 column. The vector of independent terms (b) was generated by multiplying the matrix A and vector x. For each system generated was passed to BiCGStab(2) only the matrix A and the vector b which, after attempting convergence system, returned vector x.

To evaluate the computational gain obtained by implementing the parallelized method, it was recorded the time spent on each iteration of the BiCGStab(2) algorithm.

The purpose of this paper was not to compare the differences between Matlab and solution methods, but only use Matlab to generate the random systems and compare the speed of calculation of the methods in the solution of several linear systems.

A. Computational setup

All calculations were carried out by using a desktop computer with an Intel Core i7 2600 (3.40 GHz), 8 GB of RAM memory and a $NVIDIA^{®}$ GeForce GTX 550Ti graphics card with 192 CUDA cores and 2 GB of memory config. The Matlab R2013a software platform was employed throughout.

V. Results and Discussion

The results obtained with the BiCGStab(2) parallelized method were compared with its sequential implementation, in order to verify the computational gain obtained with parallelized implementation. Additionally, a comparison was made with implementations (sequential and parallelized) of BiCGStab(2) proposed by Paula *et al.* [6]. The comparative graphs of processing time (in seconds) of different linear systems solved with the BiCGStab(2) method (sequential and parallelized) in Matlab are shown in Figures 1 and 2.

Figure 1 shows that the sequential implementation may be more efficient for linear systems with dimensions ranging from 10 to 1000. This is due to the fact the algorithm of the method contain inherently sequential operations. For example, the scalar products running sequentially on the CPU, depending on the size of the system, may have a significantly reduced computational time compared to the same time of execution in cores of the GPU. Likewise, the operations between scalars (steps 8, 13 and 34, for example) can not be divided between multiple threads and, consequently, this may result in poor performance when executed by a single GPU thread. Furthermore, due to the existence of an overhead associated with the parallelization of tasks in GPU, the size of the system to be solved must be taken into consideration [3], [6], [8].

On the other hand, Figure 2 shows that for systems with dimension greater than 1500, the parallelized BiCGStab(2)

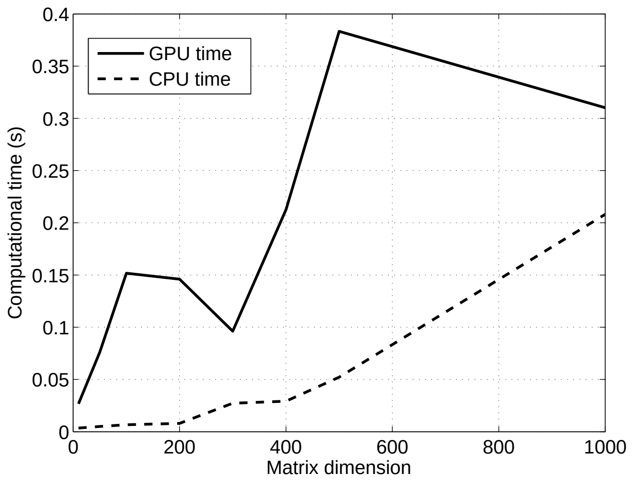


Fig. 1. Comparison of calculation speed for systems with dimension between 10 and 1000.

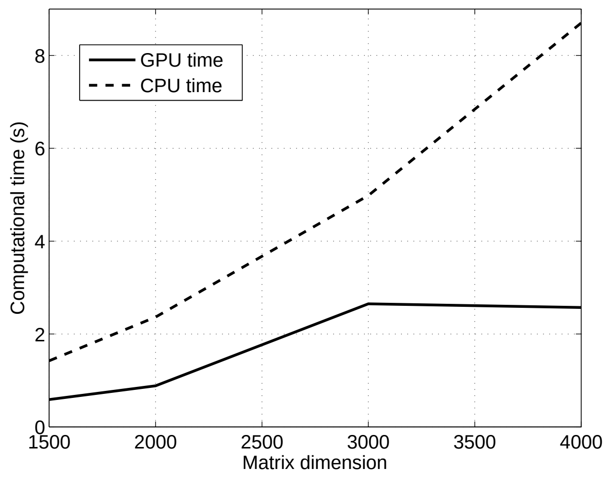


Fig. 2. Comparison of calculation speed for systems with dimension between 1500 and 4000.

exceeds the sequential implementation. In this case, in comparison of computational efficiency, the speedup gain obtained was approximately 2.59x. Therefore, the implementation that uses the GPU would be more appropriate since the size of the system used is greater than 1500×1500.

Figure 3 shows a comparison between the proposed sequential implementation and the sequential implementation proposed by Paula *et al.* [6]. The BiCGStab(2) implemented in Matlab is much higher compared to the same implementation in C language. It is observed that the time for implementation proposed by Paula *et al.* [6] requires a computational effort which increases approximately exponentially with the size of the system, while the time for implementation in Matlab is less pronounced. The speedup gain provided by the sequential implementation in Matlab was approximately 76.75x. Consequently, the use of the method implementation in Matlab can provide a more significant gain of computational performance.

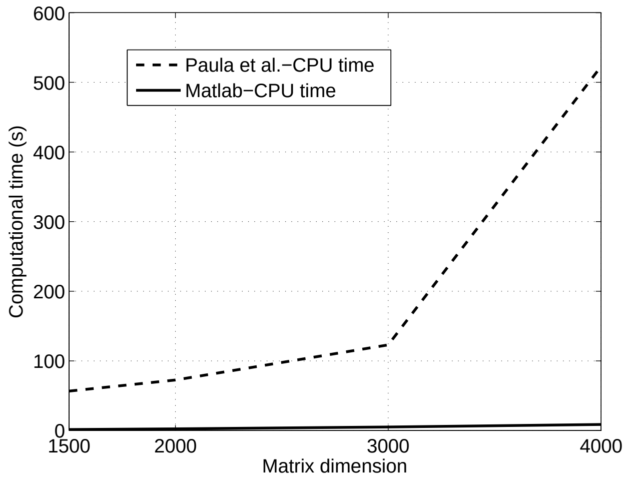


Fig. 3. Comparison of calculation speed for systems with dimension between 1500 and 4000 between sequential implementations of the BiCGStab(2) in Matlab and C.

Figure 4 shows a comparison between the proposed parallelized implementation and the parallelized implementation proposed by Paula *et al.* [6]. As in the previous case, it is possible to note the superiority of the parallelized BiCGStab(2) using CUDA-Matlab integration in the solution of the treated systems. It can be seen that the time for implementation in CUDA-C also requires a computational effort approximately exponentially in that the size of the system increases. In this case, the speedup obtained was approximately 6.12x. Therefore, compared to the parallelized implementation proposed by in [6], the parallelized BiCGStab(2) in Matlab can be a more appropriate choice of the computational point of view.

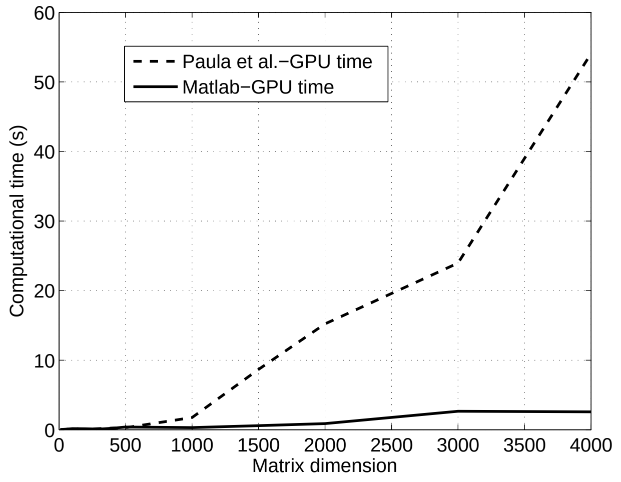


Fig. 4. Comparison of calculation speed for systems with dimension between 1500 and 4000 between parallelized implementations of the BiCGStab(2) in CUDA-Matlab and CUDA-C.

VI. Conclusion

We have implemented and used in this work a computer code in Matlab of the BiCGStab(2) iterative method for solution of large and sparse linear systems. The method was implemented on a fully sequential version as well as in a parallelized version using a GPU with CUDA-Matlab integration. The purpose of this paper was to present a new implementation of BiCGStab(2) to enable the rapid solution of linear systems and compare the computational performance with the sequential implementation. Additionally, a comparison was made with the sequential and parallelized implementation proposed in [6].

For the systems evaluated here, it was found a superiority of the parallelized implementation with CUDA-Matlab regarding the computational time spent in the calculation of each system. It was possible to obtain a speedup gain of around 76x and 6x compared to the sequential and parallelized implementation presented in [6], respectively. Compared to the sequential implementation in Matlab, the parallelized BiCGstab(2) was faster only for systems with dimension greater than 1500, and the speedup was approximately 2.5x. Therefore, it was concluded that the implementation of the method that performs in the GPU, compared to implementations proposed by Paula *et al.* [6], would be a more suitable and appropriate implementation to obtain a significant computational performance.

Future works in this same line of research may solve linear systems with larger dimensions than this paper. The systems generated in the simulations of fluid flow problems studied in the Computational Fluid Dynamics may be solved. Techniques for efficient exploitation of parallelism in scalar product between vectors operations can also be applied in an attempt to further increase the computational performance. Furthermore, alternatives to CUDA-Matlab integration such as OpenCL [19] may be investigated for comparative studies.

Acknowledgment

The authors thank the research agencies CAPES and FAPEG for the support provided to this research.

References

[1] Nesrin Aydin Atasoy, Baha Sen, and Burhan Selcuk, *Using gauss-jordan elimination method with cuda for linear circuit equation systems*, Procedia Technology **1** (2012), no. 0, 31–35.

[2] Elise Cormie Bowins, *A comparison of sequential and gpu implementations of iterative methods to compute reachability probabilities*, Proceedings First Workshop on GRAPH Inspection and Traversal Engineering (2012), 20–34.

[3] N. $CUDA^{TM}$, *Nvidia cuda c programming best practices guide*, NVIDIA Corporation, 2701 San Tomas Expressway Santa Clara, CA 95050, 2009.

[4] NVIDIA $CUDA^{TM}$, *Nvidia cuda c programming guide*, 5.0 ed., NVIDIA Corporation, 2701 San Tomas Expressway Santa Clara, CA 95050, 2013.

[5] NVIDIA Corp. $CUDA^{TM}$, *Accelerating matlab with cuda*, vol. 1, NVIDIA Corporation, 2007.

[6] Lauro Cássio Martins de Paula, Leonardo Barra Santana de Souza, Leandro Barra Santana de Souza, and Wellington Santos Martins, *Aplicação de processamento paralelo em método iterativo para solução de sistemas lineares*, X Encontro Anual de Computação, 2013, pp. 129–136.

[7] Lauro Cássio Martins de Paula, Anderson S. Soares, Telma W. Soares, Alexandre C. B. Delbem, Clarimar J. Coelho, and Arlindo R. G. Filho, *Parallelization of a modified firefly algorithm using gpu for variable selection in a multivariate calibration problem*, International Journal of Natural Computing Research **4** (2014), no. 1, 31–42.

[8] Lauro Cássio Martins de Paula, Anderson Silva Soares, Telma W. Soares, Wellington Santos Martins, Arlindo Rodrigues Galvo Filho, and Clarimar Jos Coelho, *Partial parallelization of the successive projections algorithm using compute unified device architecture*, International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), 2013, pp. 737–741.

[9] Fabio Fabris and Renato A. Krohling, *A co-evolutionary differential evolution algorithm for solving min-max optimization problems implemented on gpu using c-cuda*, Expert Systems with Applications **39** (2012), no. 12, 10324–10333.

[10] Jingfei Kong, Martin Dimitrov, Yi Yang, Janaka Liyanage, Lin Cao, Jacob Staples, Mike Mantor, and Huiyang Zhou, *Accelerating matlab image processing toolbox functions on gpus*, Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, ACM, 2010, pp. 75–85.

[11] J. Little and C. Moler, *Matlab gpu computing support for nvidia cuda-enabled gpus*, http://www.mathworks.com/discovery/matlab-gpu.html, 2013.

[12] Xiongwei Liu, Lizhi Cheng, and Qun Zhou, *Research and comparison of cuda gpu programming in matlab and mathematica*, Proceedings of 2013 Chinese Intelligent Automation Conference, Springer, 2013, pp. 251–257.

[13] J. Reese and S. Zaranek, *Gpu programming in matlab*, http://www.mathworks.com/company/newsletters/articles/gpu-programming-in-matlab.html, 2011.

[14] Youcef Saad and Martin Schultz, *Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM Journal on scientific and statistical computing **7** (1986), no. 3, 856–869.

[15] Yousef Saad, *Iterative methods for sparse linear systems*, Siam, 2003.

[16] Vaclav Simek and Ram Rakesh Asn, *Gpu acceleration of 2d-dwt image compression in matlab with cuda*, Computer Modeling and Simulation, 2008. EMS'08. Second UKSIM European Symposium on, IEEE, 2008, pp. 274–277.

[17] Gerard Sleijpen and Henk A. Van der Vorst, *A jacobi-davidson iteration method for linear eigenvalue problems*, SIAM Review **42** (2000), no. 2, 267–293.

[18] Gerard L. G. Sleijpen and Henk A. Van Vorst, *Hybrid bi-conjugate gradient methods for cfd problems*, Computational Fluid Dynamics REVIEW (1995), no. 902.

[19] Ryoji Tsuchiyama, Takashi Nakamura, Takuro Iizuka, Akihiro Asahara, Jeongdo Son, and Satoshi Miki, *The opencl programming book*, Fixstars, 2010.

[20] Henk A. Van Vorst, *Bi-cgstab: A fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems*, SIAM Journal of Scientific and Statistical Computing **13** (1992), no. 2, 631–644.

[21] Daniel Weber, Jan Bender, and Markus Schnoes, *Efficient gpu data structures and methods to solve sparse linear systems in dynamics applications*, Computer Graphics Forum, Wiley Online Library, 2012.

[22] Ahmet Artu Yldirim and Cem Ozdogan, *Parallel wavelet-based clustering algorithm on gpus using cuda*, Procedia Computer Science **3** (2011), no. 0, 396–400.

Cellular Automata as Acceleration Kernel of Interconnection Network Simulation

Takashi Yokota, Kanemitsu Ootsu, and Takeshi Ohkawa
Department of Information Systems Science, Graduate School of Engineering, Utsunomiya University
7–2–1 Yoto, Utsunomiya, Tochigi, 321-8585 Japan.

Abstract—*Drastic growth in the number of processor cores in state-of-the-art supercomputers is increasing difficulties in discussing and designing efficient architectures. One of the toughest problems is interconnection network (ICN) architecture that is severely responsible to both performance and costs. For the coming exa-scale supercomputers and beyond, a significant breakthrough is expected in discussing ICN methods. This paper presents a novel ICN simulation method by introducing cellular automaton (CA) principle. Through a three-step discussion in introducing CA, the proposed method can offer a powerful simulation engine. Evaluation results reveal the significant speedup of simulation time with precise simulation results. Furthermore, our preliminary implementation in GPGPU shows about sixteen times acceleration from commercial four-core processors.*

Keywords: interconnection networks, simulation, large-scale parallel systems, cellular automata

1. Introduction

State-of-the-art supercomputers are employing a huge number of processors. The Top-500 list of supercomputers[1] indicates that the numbers of processor cores in top-five supercomputers are drastically increasing as Fig. 1 depicts, where the number of processor cores increases about 10^3 times in two decades.

Development of high-end supercomputers involves a wide spectrum of technological challenges, and one of the most important challenges is interconnection methodology. Interconnection network (ICN) is one of the most important key issues. Thus, many research institutes discuss ICN technologies from various points of view; physical layer technologies, e.g., optical interconnection, topologies, routing algorithms, fault-tolerance, and so on.

This paper addresses simulation issues for large-scale ICNs. So many researches have reported their ICN methods, e.g., topology and routing algorithm. In many cases, effectiveness of their novel method is depicted as software simulation results, where a *reasonable* system size is assumed. Conversely speaking, although we expect a high-level of effectiveness of the novel method by extrapolating the scale of system, actual effectiveness is not proven.

Large-scale ICN simulations face serious difficulties. Most of ICN simulators are not ready for extreme-scale simulations. Although parallelizing ICN simulator is not a difficult task, large-scale simulation requires a lot of resources and long computing time. Some simulation infrastructures offer large-scale simulations, although, actual simulation sizes are limited as opposed to skyrocketing supercomputers.

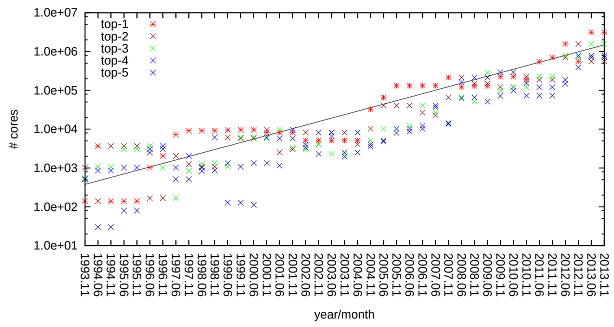


Fig. 1: The number of processor cores in top-five supercomputers. The linear line approximates the top-1 dots by the least-square method.

An alternative approach is approximated method that estimates total communication flows by approximation. Although the method is useful in offering a rough estimation, we have to consider preciseness of the estimation results. Thus, the approach is out of our scope.

According to the discussions above, this paper aims at light-weight ICN simulation fundamentals so that we can quickly estimate new ideas in large-scale situations. For this purpose, we introduce a cellular automata principle in modeling an ICN router. The major contribution of this paper is to offer a new idea based on cellular automata for accelerating simulation speed of large-scale interconnection networks.

The rest of this paper is organized as follows. Section 2 overviews related work in ICN simulation methods from large-scale simulation point of view and also from cellular automaton application. Section 3 provides fundamentals of ICN simulation and offers baseline assumptions in this paper. Section 4 discusses modeling of router functions by means of cellular automaton principle, followed by experimental results in Section 5. Finally, Section 6 concludes this paper.

2. Related Work

Many researches require their own simulators and some of them are presented and discussed. Dally et al. present BookSim[2] that was originally developed for writing their textbook of interconnection networks[3]. Recent simulators include OMNET++ [4] and TOPAZ [5]. They have remarkable features in flexibility and ease of configuration. These simulators are fully functional, but, they are not suitable for extremely large-scale systems simulation.

As shown in Fig. 1, rapid growth in the scale of supercomputers attracts computer architects to serious fear in designing extremely large-scale parallel systems. This crisis motivates system architects to discuss appropriate methods in estimating interconnection costs and performance. Thus, several important simulation infrastructures are proposed: SMART [6], Pose [7], [8], BigSim [9], and OpenNSIM [10].

Although large-scale simulators offer affordable environment, there still exist essential problems in extremely large-scale simulations, i.e., they require a lot of resources and computing times. From this point of view, approximate methods are given as an alternative approach. Atzori and Isola[11], Choudhury et al.[12], and Yazaki et al.[13] present their specific characteristics. Yokota et al.[14] have also investigated scaling issues by means of parallelized ICN simulator.

Fired by Wolfram's significant milestone[15], cellular automata have been discussed in wide variety of application fields[16]. For example, lattice gas methods present quick, accurate and powerful simulation methodologies[17]. Simply speaking, a car is regarded as a particle in the conventional CA model and this naturally extends traditional CA models to car-traffic. For example, cars running on a road are modeled by one-dimensional cellular automata. As CA models of car-traffic are actively discussed, e.g. [18], analogies of car-traffic and information network are recognized[19]. The essential ideas of car-traffic model of CA is flow simulation, thus, with regard to information network, message packets instead of cars are modeled[20]. Lawniczak et al. [21] present cellular automata model of OSI network layer and discuss its preciseness. Brooks et al.[22] present a practical cellular automata model of UDP and TCP traffics, which can quickly approximates a well-known realistic simulator NS-2.

3. ICN Simulation

3.1 Router Organization

This subsection presents the baseline architecture on which this paper discusses simulation issues.

This paper assumes two-dimensional torus networks for simplification of discussions. While other topological options are also possible for discussion, two-dimensional torus has general characteristics that are applicable to the other topologies, i.e., not a small number of input/output ports and multiple virtual channels.

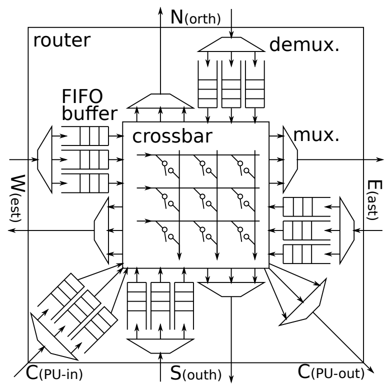


Fig. 2: Router organization for two-dimensional torus.

A typical organization of router for two-dimensional torus network is depicted in Fig. 2. A router has five sets of input and output ports, as Fig. 2 illustrates N(orth), E(ast), S(outh), W(est), and C(pu) ports. As this figure suggests, N, E, S, and W ports are used to connect to the four neighboring routers, and C port connects the corresponding processor core.

Each input port employs a necessary number of FIFO buffers that correspond to virtual channels. Virtual channels offer logical communication paths for enhancing performance and preventing deadlocks. But, a physical connection between two neighboring routers, and also between router and processor, is simply a *wire*. Thus, simultaneous transmissions along multiple VCs are prohibited at any point of time. This physical limitation implements a demultiplexor before the VC buffers, and also implements a multiplexor at each output port.

After being buffered by the VC buffer, each head packet requests a connection circuit to an appropriate output port and channel. A crossbar switch implements the connection circuit as shown in Fig. 2. It is usually explained as collection of cross-point switches. A crossbar never accepts conflicting requests for an identical output. Thus, from a logic-functional point of view, a crossbar is regarded as a set of selectors.

3.2 Synchronous Simulation

Many ICN simulators are designed under the cycle-accurate principle, where all components operate clock by clock. Usually, an ICN simulator assumes a global clock signal and all components operate synchronously by means of the global clock.

To ensure the completely-synchronous operations, an ICN simulator follows two-phase update fashion at every clock cycle. In the first phase, each router firstly checks incoming packets in its five input ports. Since a packet belongs to one of the virtual channels, the packet is stored in the corresponding VC buffer.

After storing incoming packets, the router determines output direction of each of FIFO-top packets. Based on the output request information, the router resolves conflicting requests to properly control the crossbar switch that connects input packets to output ports. Thus, when some of requests conflict to an identical output port, only one of them is selected to pass through the crossbar switch, and other ones are kept in their input buffer. A virtual channel supports a logical link and multiple (three in this paper) VCs share a physical channel (link). Furthermore, this paper assumes that each output port has no memory elements. Thus, at most one virtual channel wins to transfer a packet and other channels are suppressed.

By the end of the first phase, each router determines the next states of its subordinate components. The next state is a local variable that is invisible to neighboring routers and processors. The router updates its (current) state with the next one in the second phase.

As the ICN simulator should answer the cycle-accurate results, it has to follow appropriate handling of packets from generation to consumption. This means that the simulator should follow processor behaviors to fulfill the necessary communication situations to simulate. For example, in a simulation process of continuous communication, each processor generates appropriate packets following a given traffic pattern in given intervals.

Furthermore, statistics functions are required for ICN simulation, that is, the ICN simulator should measure the number of received packets and average (and sometimes maximum) latency of the received packets. In unsteady communication situations, the simulator should measure duration time of collective communication.

4. Cellular Automata Model

4.1 Cellular Automata for Flow Simulation

As we mentioned in Section 2, cellular automaton models are widely used for flow simulation. The most typical example is car-traffic simulation[18]. A simple example situation is highway where a lane is represented as a string of cells. Each cell represents discretized position of cars and a white/black cell represents occupied/unoccupied by a car.

The primitive idea in car-traffic models is based on so-called *box-ball* models. A car can move to the neighboring cell if and only if the neighboring cell is not occupied by another car. While the update rule of cell is quite simple, the car-traffic system sometimes shows jamming phenomena that we sometimes encounter in real life.

4.2 Naive Model

Our first idea is to apply a cellular automaton principle to ICN simulation. Contrary to the car-traffic models, we have to make further discussions for modeling router functions. After being transmitted from an output port, a packet immediately runs to the next router. An input port has multiple VC buffers, and VCs share a physical link.

In our previous work in [23], [24], we have simplified the cellular automata model of router at an essential level. The center idea of the CA model is that a set of CA cells represent router functions where the binary state of each cell represents whether the cell is occupied by a message packet or not. Only uni-directional traffics, i.e., rightward and downward, are supported. Virtual channels are represented as parallel lanes that do not share an identical physical link. The crossbar switch is simply controlled by *traffic signal* where blue(red) signal allows rightward(downward) traffic.

The essentially simplified model shows phase transition phenomena at congested situation, and qualitative characteristic that emerging speed of congestion is $O(1)$ and the congestion sustains $O(N^2)$ time. The results clarify qualitative knowledge, but, the results are not sufficient for quantitative discussions.

Thus, we extend the simplified model to match the realistic router organization by means of cellular automata principles[25]. The first point is bidirectional communication in each direction and this extension is straightforward. The second point is an appropriate model of a crossbar switch. As we stated in Section 3.1, a crossbar switch is a collection of selectors. We model the crossbar behavior with a single cell that is connected from the FIFO top cells in input ports in a many-to-one fashion. The update rule functions as a selector. The third point is the multiplexing function that is required for multiple virtual channels to share a physical link. Fig. 4 presents the organization of our CA model of a router.

We introduce following three components of cellular automata to properly model the router organization:

- buffering function that implements the first-in-first-out buffer function (Fig. 5(a)),
- crossbar function that implements the crossbar switch function (Fig. 5(b)), and
- channel selection function that implements channel-sharing function of virtual channels (Fig. 5(c)).

In CA principle, the next state of each cell is determined by its neighboring cell. Our three CA components also follow the principle, although, update rules differ so that each rule corresponds to a specific function.

In the buffering function, each cell updates its own state simply according to the preceding cell. Other two functions have slightly complicated rules. The center cell in Fig. 5(b) is the crossbar cell, which represents an output request to a specific direction (e.g., east) via the channel selection function. Surrounding cells are the FIFO top cells in input ports, and each cell requests the center cell iff its occupying packet are going to the corresponding direction of the center cell. The channel selection function allows at most one message packet to traverse the corresponding physical link.

We further introduce a table-based update method to accelerate CA computation. For example in Fig. 5(a), since each cell has binary state, i.e., 0/1, both current and next states of the whole cells are summarized in a five-bit binary number.

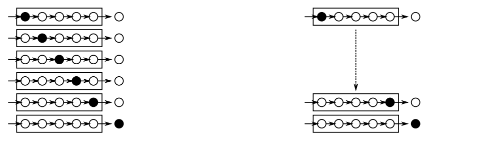

(a) Naive update. (b) Macro CA update.

Fig. 3: Update sequence in the naive and Macro CA models.

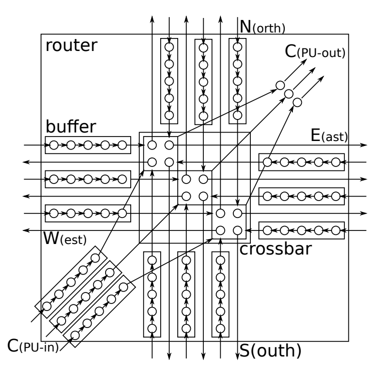


Fig. 4: CA representation of a router.

We can prepare a state transition table based on the binary representation, and this reduces computation significantly.

4.3 Macro CA Model

By introducing the CA principle and table-based update method, we can expect drastic speedup in simulating ICNs in which a vast number of routers are connected to route packets. But, the naive model has a serious drawback: the buffering function does not sufficiently reflect actual FIFO behaviors. The naive CA behaves a shift-register function and it consumes unnecessary clock cycles to pass through the buffer even when the buffer contains no other packets.

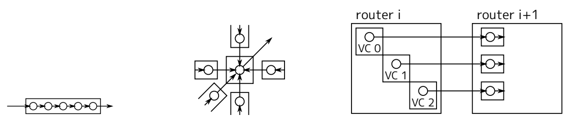
Accurate simulation needs accurate FIFO behavior in the buffering function. To satisfy the requirement, we extend the CA principle by modifying the update table that is introduced in the previous section. Fig. 3 shows an example. The Macro CA update method reduces intermittent steps as depicted in Fig. 3(b), whereas the naive method requires unnecessary steps (Fig. 3(a)).

4.4 Asynchronous Model

The proposed Macro CA model follows the practical behaviors of router functions, however, it does not yet earn satisfactory preciseness in simulation results. The reason comes from (1) the simple *box-ball* system that restricts the packet flow rate to 1/2 and (2) the three separate CA components that correspond to the individual functions of buffering, crossbar and channel selection.

To ensure consistent results, simulations are requested to be operated fully synchronous. Since the Macro CA model also consists of three CA components, a message packet requires at least three steps in time to go through a router, i.e., one step in buffer, one step in crossbar, and further one step in channel selection. One straightforward solution is representing all possible transitions in a single update table, but, this method requires too large table in size since many cells are used in each router.

The alternative solution is asynchronous update in the three CA components with the router model, while synchronous update principle is guaranteed at intra-router level. The CA-based router organization (in Fig. 4) forms a uni-directional graph from input ports to output. By starting from the buffering function in input port, asynchronous update even keeps consistency of cells states. Practically, the buffering function is applied followed by the crossbar and channel-selection functions in order. This improves packet transfer latency, e.g., it requires only one step to traverse a router, while the Macro CA requires three steps.



(a) bufer (b) crossbar (c) channel selection

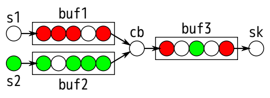
Fig. 5: Three CA components of router model.

5. Evaluation

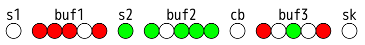
5.1 Space-Time Behaviors

Fig. 6(a) shows a simple merging flow followed by a FIFO buffer. Packet sources *S1* and *S2* produce packets continuously and the generated packets are injected into the corresponding buffers *buf1* and *buf2*. Output packets from the two buffers are merged at the crossbar *cb* and the selected packet is fed to the succeeding buffer *bufn* to a sink node *sk*. To represent the cells states, cells are aligned in a row as depicted in Fig. 6(b). Space-time chart of each CA model described in the previous section is shown in Figs. 6(c) to (e).

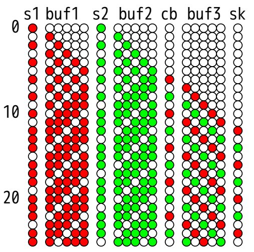
Fig. 6(e) shows preferable behaviors of the asynchronous model. It is clear that the asynchronous update with a router outperforms other CA models in the two aspects of performance metrics. One is throughput and the other one is latency. With respect to the former, we can see the string of colored cells at the *sk* cell, the naive and Macro CA models only feed one packet in every two cycles, while the asynchronous model feeds consecutively. Latency issues are not directly shown in the figure, however, we can discuss on the figure. In Fig. 6, buffers are saturated at about 30th cycle in either model. In the saturated situation, the source nodes inject a packet in every four cycles in the naive and Macro CA models, while the asynchronous model allows generating a packet in every two cycles. This reflects drastic difference in latency.



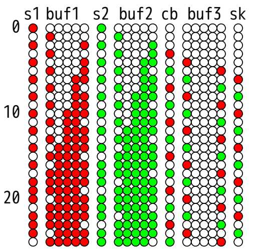
(a) A merging flow followed by a buffer.



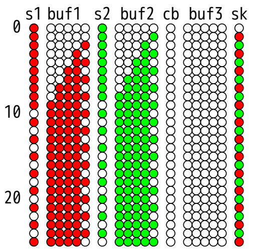
(b) One-line representation of (a).



(c) Naive model.



(d) Macro CA model.



(e) Asynchronous model.

Fig. 6: Space-time snapshots in ramp-way simulation. Time goes downward. Organization is depicted in Fig. 6(a) and all of cell states are represented in a row as explained in Fig. 6(b).

5.2 Simulation Preciseness

The asynchronous update method significantly improves packet-movement behaviors and we can expect precise simulation results. We built three versions of CA-based ICN simulator that are based on the naive model, macro CA model, and asynchronous model, respectively. These simulators perform both steady and unsteady communication situations.

Firstly, we compare steady communication results of the CA-based simulator to those of our conventional (i.e., non-CA based) simulator as a reference. Fig. 7 shows *transpose* traffic pattern results. Horizontal axis shows traffic load given in a relative representation to $8/N$ where N is the network size[14]. Curves with X symbols show throughput that is measured as the number of received packet and their values are given as the left vertical axis. The right vertical axis shows average latency of received packets whose curves are drawn with box-shaped symbols.

When the traffic load is low, throughput is proportional to the load and average latency is constantly low. After the load exceeds a threshold, throughput is saturated and average latency is drastically increased around the saturation point. We can find the following points in Fig. 7. As discussed in the previous section, difference in (average) latency is remarkable, but, the asynchronous model follows the conventional model that is represented as *sim* in the diagram. Saturation points of the naive and macro CA models are low, whereas the asynchronous model is well suited to *sim*.

The major reason of the significant gaps of the naive and macro CA models is inter-router update rule. As shown in the previous subsection, these two models cannot support continuous packet sending without any gaps and this prevents precise simulation. On the other hand, the asynchronous model resolves the gap problem, thus supports precise simulation.

We further discuss preciseness of the asynchronous model in other traffic patterns. Fig. 8 shows the results. Diagrams in Fig. 8 show that the asynchronous model follows the conventional simulator at high level in various traffic patterns. Important issues in ICN performance are saturation point in throughput and average latency level. The asynchronous model matches the reference performance at the both points.

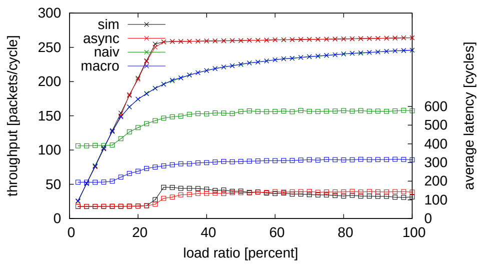


Fig. 7: ICN performance simulation results of 128×128 torus network by the presented CA models with the results of the conventional simulator.

Table 1: Duration times measured in the CA models and conventional simulator.

traffic pattern	naive model	macro model	async model	conv. sim.
trns	779	455	195	260
shfl	779	515	303	401
bcmp	767	510	192	194
brev	773	577	318	328
brot	779	666	315	396
rand	776.8	443.7	176.8	187.1
torn	401	581	258	260

(unit: clock cycles)

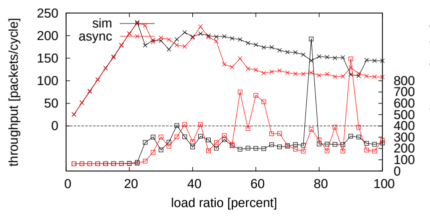
Note that an intermittent congestion drastically degrades performance at near- and after saturation point and the turbulence appear as a peak in the performance diagram. Other differences come from the crossbar and channel selection functions that are not clearly separated in the conventional simulator.

Another preciseness discussion is for unsteady communication. Table 1 summarizes duration times that are measured in our various simulators. The asynchronous model performs closely to the conventional simulator.

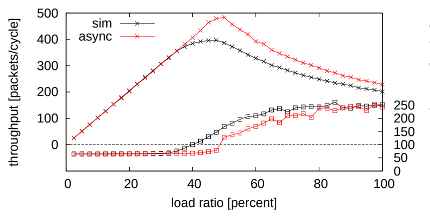
5.3 Simulation Speed

Fig. 9 compares elapsed time of simulation of each simulator, i.e., the conventional simulator (*sim*), CA-based simulators (*naive*, *macro*, and *async*).

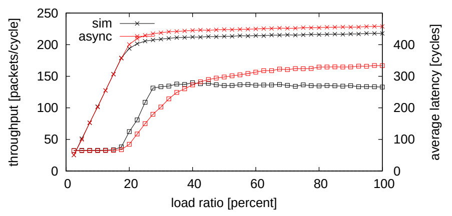
Each simulator runs on Core i7 3770S 3.1GHz clock.



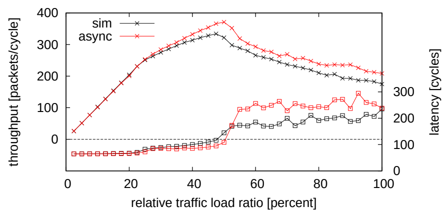
(a) perfect shuffle (*shfl*)



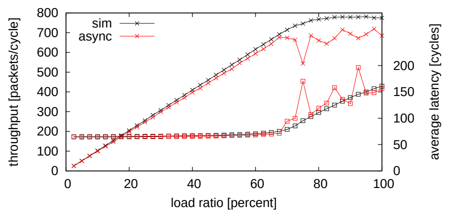
(b) bit-complement (*bcmp*)



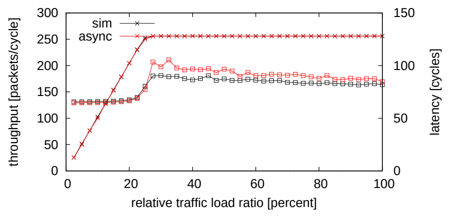
(c) bit-reversal (*brev*)



(d) bit-rotation (*brot*)



(e) uniform random (*rand*)



(f) tornado (*torn*)

Fig. 8: Simulation results of the asynchronous model compared to the conventional simulator results. 128×128 torus network results.

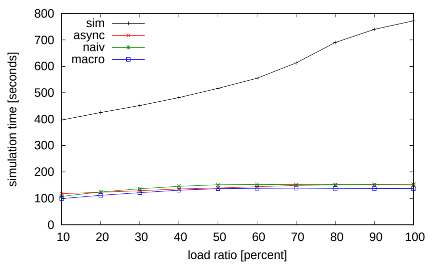


Fig. 9: Simulation speed comparison. 128×128 two-dimensional torus network in uniform random traffic pattern for 20,000 clock cycles.

The conventional simulator is parallelized by means of MPI (message passing interface, [26]). The CA-based simulators are also parallelized but by *pthreads* library. In both cases, eight processes (threads) are used.

In Fig. 9, we can find remarkable speedup by means of the CA model. Specifically, the asynchronous model accelerates simulation speed from 3.5 to 5.0 times faster than the conventional one.

Another remarkable feature is that simulation time is stable for traffic loads. In Fig. 9, the conventional simulator requires long simulation time as traffic load increases. But, the CA-based ones are stable.

5.4 GPGPU Application

Since each router independently operates, ICN simulation inherently involves large-scale parallelism. Furthermore, operations in each router are not so complicated tasks. Thus, this nature well matches many-core architectures and GPGPU is a natural option.

We preliminarily port our CA-based ICN simulator onto the CUDA environment that is supplied by NVIDIA. The porting task is basically straightforward, but some modifications are necessary to extract the GPU's full power. In our implementation to GPGPU, memory access scheduling is the most sensitive to performance.

As a preliminary evaluation, we use a unsteady communication condition to compare simulation performance of GPGPU and multicore processor. We use a consumer electronics device GTX-780 from NVIDIA and Core i7 4770 processor from Intel. Our original CA simulator is parallelized by means of *pthread* library and we can run the simulator with a specified number of processors (threads). In this performance evaluation, we use maximum of eight threads for the 4770 processor.

The preliminary evaluation uses a simple situation of unsteady communication of a perfect-shuffle traffic pattern in which each node generates ten packets. The system size is 1024×1024 connected in the two-dimensional torus topology. Performance is measured by CUDA-supplied functions *cudaEventElapsedTime()* in GPGPU, and *gettimeofday()* function in the pthread execution. The former runs in 212.5 seconds whereas the latter requires 3388.6 seconds, i.e., the commercial GPU runs about 16 times faster than the commercial multicore processor.

5.5 Discussion

Results shown in the previous subsections reveal preferable features in the proposed CA-based model for ICN simulation.

The table-based update principle reduces complexity class considerably. Process in the conventional simulator is composed of buffering, routing, and selection functions. These processes are not so complicated, although, net complexity becomes large because of large number of packets. On the other hand, the CA-based model reduces intermittent computation by means of table-lookup operations and thus reduces redundant complexity.

Furthermore, the table-based method eliminates conditional branches that prevent full power of GPU capability since GPU runs in the SIMD (single-instruction, multiple data stream) fashion.

6. Conclusions

Rapid growth in the scale of parallel supercomputers drives computer architects to discuss extremely large-scale simulation. Interconnection network (ICN) is one of the most serious problems and it is clear that some breakthrough is required for the future exa-scale systems.

This paper contributes the cellular automaton principle in large-scale ICN simulation. The cellular automata approach simplifies the router model, but a naive CA implementation is too simplified to represent accurate simulation. Thus, we extended the naive model to match realistic one to reach the asynchronous model that well fits to the conventional simulator.

This paper firstly showed the microscopic behaviors in the proposed CA models. Then, we discussed preciseness of simulation results and speed of simulation runs. Evaluation results reveal that the proposed CA model well matches the conventional simulator while the model runs 3.5 to 5.0 times faster.

This paper further discussed GPGPU implementation of the CA model. The table-based update method that is proposed in this paper well matches GPU architectures, since the method reduces redundant computation in router functions and eliminates conditional branches. Our preliminary evaluation results show about sixteen times acceleration by a commercial GPU device.

Acknowledgment

This research was supported in part by Grant-in-Aid for Scientific Research ((C)24500055, (C)24500054) and Young Scientists ((B)25730026) of Japan Society for the Promotion of Science (JSPS).

References

[1] "Top500 supercomputer sites," http://www.top500.org/.

[2] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. Shaw, J. Kim, and W. J. Dally, "A detailed and flexible cycle-accurate network-on-chip simulator," in *Proc. IEEE Int. Symp. Performance Analysis of Systems and Software (ISPASS)*, 2013, pp. 86–96.

[3] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[4] A. Varga and R. Hornig, "An overview of the OMNET++ simulation environment," in *Proc. 1st Int. Conf. Simulation Tools and Techniques for Communications, Networks and Systems & Workshops (Simutools '08)*, 2008, pp. 60:1–60:10.

[5] P. Abad, P. Prieto, L. Menezo, and J.-A. G. Adrian Colaso, Valentin Puente, "TOPAZ: An open-source interconnection network simulator for chip multiprocessors and supercomputers," in *Proc. 6th IEEE/ACM Int. Symp. Networks on Chip (NoCS)*, 2012, pp. 9–11.

[6] F. Petrini and M. Vanneschi, "SMART: a Simulator of Massive ARchitectures and Topologies," in *Proc. Int. Conf. Parallel and Distributed Systems (Euro-PDS'97)*, 1997, pp. 185–191.

[7] T. L. Wilmarth and L. V. Kalé, "POSE: Getting over grainsize in parallel discrete event simulation," in *Proc. Int. Conf. Parallel Processing (ICPP 2004)*, vol. 1, 2004, pp. 12–19.

[8] T. L. Wilmarth, G. Zheng, E. J. Bohm, Y. Mehta, N. Choudhury, P. Jagadishprasad, and L. V. Kalé, "Performance prediction using simulation of large-scale interconnection networks in POSE," in *Proc. 19th Workshop on Principles of Advanced and Distributed Simulation (PADS'05)*, 2005, pp. 109–118.

[9] G. Zheng, G. Kakulapati, and L. V. Kalé, "BigSim: a parallel simulator for performance prediction of extremely large parallel machines," in *Proc. 18th Int. Parallel and Distributed Processing Symp.*, vol. 1, 2004, p. 78b.

[10] H. Miwa, R. Susukita, H. Shibamura, T. Hirao, J. Maki, M. Yoshida, T. Kando, Y. Ajima, I. Miyoshi, T. Shimizu, Y. Oinaga, H. Ando, Y. Inadomi, K. Inoue, M. Aoyagi, and K. Murakami, "NSIM: An interconnection network simulator for extreme-scale parallel computers," *IEICE Trans. Information and Systems*, vol. E94-D, no. 12, pp. 2298–2308, December 2011.

[11] L. Atzori and M. Isola, "A traffic scaling approach to speed up network simulations," in *Proc. Global Telecommunications Conference (GLOBECOM)*, vol. 7, 2003, pp. 3862–3866.

[12] N. Choudhury, Y. Mehta, T. L. Wilmarth, E. J. Bohm, and L. V. Kalé, "Scaling an optimistic parallel simulation of large-scale interconnection networks," in *Proc. 37th Conf. Winter Simulation (WSC'05)*, 2005, pp. 59–1–600.

[13] S. Yazaki and H. Ishihata, "An evaluation of message-flow-based network simulator," *IPSJ Trans. Advanced Computing Systems*, vol. 4, no. 3, pp. 47–55, May 2011, (in Japanese).

[14] T. Yokota, K. Ootsu, and T. Baba, "Are uniform networks scalable?" in *Proc. 9th Int. Conf. Parallel and Distributed Computing, Applications and Technologies (PDCAT 2008)*, Dec. 2008, pp. 137–140.

[15] S. Wolfram, Ed., *Theory and Applications of Cellular Automata*. World Scientific Publishing, 1986.

[16] H. Gutowitz, Ed., *Cellular Automata: Theory and Experiment*. MIT Press, 1991.

[17] G. D. Doolen, Ed., *Lattice Gas Methods: Theory, Applications, and Hardware*. MIT Press, 1991.

[18] J. A. Cuesta, F. C. Martínez, J. M. Molera, and A. Sánchez, "Phase transitions in two-dimensional traffic-flow models," *Physical Review E*, vol. 48, no. 6, pp. R4175–R4179, December 1993.

[19] S. Gábor and I. Csabai, "The analogies of highway and computer network traffic," *Physica A*, vol. 307, pp. 516–526, 2002.

[20] Z. Ren, Z. Deng, and Z. Sun, "Cellular automaton modeling of computer network," *Computer Physics Communications*, vol. 144, pp. 243–251, 2002.

[21] A. Lawniczak, A. Gerisch, and B. D. Stefano, "Development and performance of cellular automaton model of OSI network layer of packet-switching networks," in *Proc. Canadian Conference on Electrical and Computer Engineering (CCECE 2003)*, vol. 2. IEEE, May 2003, pp. 1409–1412.

[22] R. R. Brooks, C. Griffin, and T. A. Payne, "A cellular automata model can quickly approximate UDP and TCP network traffic," *Complexity*, vol. 9, no. 3, pp. 32–40, 2004.

[23] T. Yokota, K. Ootsu, F. Furukawa, and T. Baba, "A cellular automata approach for understanding congestion in interconnection networks," *IPSJ Trans. Advanced Computing Systems*, vol. 47, no. SIG 7(ACS–14), pp. 21–42, May 2006, (in Japanese).

[24] ——, "Phase transition phenomena in interconnection networks of massively parallel computers," *Journal of Physical Society of Japan*, vol. 75, no. 7, p. 078401, June 2006.

[25] T. Yokota, K. Ootsu, and T. Ohkawa, "A cellular automata approach for large-scale interconnection network simulation," in *Proc. 1st Int. Symp. Computing and Networking (CANDAR)*, December 2013, pp. 545–551.

[26] "MPICH: High-performance portable MPI," http://www.mpich.org/.

A Memory-Efficient Algorithm for Large-Scale Symmetric Tridiagonal Eigenvalue Problem on Multi-GPU Systems

Hyunsu Cho and Peter A. Yoon
Department of Computer Science, Trinity College, Hartford, CT, USA

Abstract—*Divide-and-conquer algorithm is a numerically stable and efficient algorithm that computes the eigenvalues and eigenvectors of a symmetric tridiagonal matrix. We often face the situation where the input matrix fits into the main memory but not into the on-chip memory of a GPU device. We present an out-of-core implementation where only part of the input matrix is resident in GPU memory at any point in time. It works independently of the physical size of GPU memory, handling any size of input as long as it fits into the main memory. Work is dynamically allocated to multiple GPUs and CPU cores, taking account of available workspaces and progress of the algorithm. In addition, it delivers a performance comparable to that of conventional multi-GPU implementations for cases where workspaces fit into the GPU memory.*

Keywords: Symmetric eigenvalue problem, parallel computation, general-purpose GPU computing, CUDA

1. Introduction

Divide-and-conquer algorithm is a widely used algorithm that computes the eigenvalues and eigenvectors of a symmetric tridiagonal matrix. The algorithm is known to be numerically stable and efficient when computing the full spectrum of eigenvalues [1]. Furthermore, any general symmetric eigenvalue problem can be reduced to tridiagonal form via a series of orthogonal similarity transformations. When combined with a deflation step, the algorithm delivers a good overall performance: it takes about $O(n^{2.3})$ flops to compute all the eigenvalues and eigenvectors of an $n \times n$ matrix [2].

The idea of using multiple GPUs to handle large matrices is not new. In particular, MAGMA library [3], [4] features a hybrid implementation of divide-and-conquer that uses both multiple GPUs and multicore CPUs. It off-loads the most costly portion of the algorithm, matrix multiplication, to the GPUs. Each GPU memory stores a part of the workspace, which is periodically synchronized with its counterpart in the main memory. This approach works well most of the time on multi-GPU systems, as intermediate workspaces do not grow beyond the total memory of all the GPU devices installed. Unfortunately, for very large input matrices, intermediate matrices may fit into the main memory but still exceed the total size of GPU memory. This situation may arise because GPU memory is limited in size compared to main memory. For instance, one NVIDIA® Tesla® K20c supports only about 5 GB of memory. Intermediate workspaces still have to be loaded to GPU memory, so that GPU cores can make high-bandwidth accesses.

We overcome this limitation by fixing the size of GPU workspaces to be less than available GPU memory. Depending on the size of GPU memory and that of the main memory, we dynamically compute the partition for block matrix multiplication. With fixed GPU workspaces, we are free to deal with any large input matrices, as long as the input matrix fits into the main memory. We confirmed that our implementation could handle input size as large as 50,000×50,000.

The overhead required by dynamic partition can be problematic for small subproblems. A general criterion is whether a subproblem fits entirely into a single GPU's memory. For small problems, it is better to avoid block matrix multiplication entirely. Instead, we let GPU devices solve multiple subproblems in parallel. This has an additional benefit of hiding latency in memory transfer, which is relatively costly compared to the small computational work involved.

This paper is organized as follows: Section 2 presents a brief overview of divide-and-conquer algorithm for symmetric tridiagonal eigenvalue problem.

Section 3 discusses how tasks should be organized in modules. Section 4 discusses important details regarding our out-of-core implementation on multi-GPU systems. Finally, Section 5 presents performance results and analysis.

2. Divide-and-conquer algorithm

Let A be an $n \times n$ symmetric tridiagonal matrix where the diagonal and subdiagonal entries are given by a_i's and b_i's respectively. The idea is to transform A into a sum of two smaller tridiagonal systems:

$$A = \left[\begin{array}{c|c} \tilde{A}_1 & \\ \hline & \tilde{A}_2 \end{array}\right] + H = \tilde{A} + H \tag{1}$$

where

$$\tilde{A}_1 = \begin{bmatrix} a_1 & b_1 & & \\ b_1 & \ddots & \ddots & \\ & \ddots & a_{m-1} & b_{m-1} \\ & & b_{m-1} & a_m - b_m \end{bmatrix}$$

$$\tilde{A}_2 = \begin{bmatrix} a_{m+1} - b_m & b_{m+1} & & & \\ b_{m+1} & \ddots & \ddots & & \\ & \ddots & & a_{n-1} & b_{n-1} \\ & & & b_{n-1} & a_n \end{bmatrix}$$

and

$$H = \left[\begin{array}{cc|cc} & & & \\ & b_m & b_m & \\ \hline & b_m & b_m & \\ & & & \end{array}\right].$$

Now that we managed to divide the given eigenvalue problem into two problems of smaller size, we can merge the eigendecompositions of $\tilde{A}_1$ and $\tilde{A}_2$ to get the eigendecomposition of $\tilde{A}$.

Suppose we have obtained the eigendecomposition of $\tilde{A}_1$ and $\tilde{A}_2$, that is, we compute orthogonal matrices $\tilde{Q}_1, \tilde{Q}_2$ and diagonal matrices $\tilde{D}_1, \tilde{D}_2$ such that

$$\tilde{A}_1 = \tilde{Q}_1 \tilde{D}_1 \tilde{Q}_1^T \text{ and } \tilde{A}_2 = \tilde{Q}_2 \tilde{D}_2 \tilde{Q}_2^T.$$

Then the eigendecomposition of $\tilde{A}$ is given by

$$\tilde{A} = \left[\begin{array}{c|c} \tilde{A}_1 & \\ \hline & \tilde{A}_2 \end{array}\right] = \tilde{Q}\tilde{D}\tilde{Q}^T$$

where

$$\tilde{Q} = \left[\begin{array}{c|c} \tilde{Q}_1 & \\ \hline & \tilde{Q}_2 \end{array}\right] \quad \text{and} \quad \tilde{D} = \left[\begin{array}{c|c} \tilde{D}_1 & \\ \hline & \tilde{D}_2 \end{array}\right].$$

The remainder of the algorithm involves transforming the eigenvalues and eigenvectors to take account of the matrix H being added on the right-hand side. To compute the eigendecomposition of A from that of $\tilde{A}$, we perform a process known as *rank-one update* [1].

The matrix H, also known as the *rank-one modifier*, is a product of form

$$H = \rho \mathbf{w}\mathbf{w}^T$$

where

$$\rho = b_m \text{ and } \mathbf{w} = \left[\begin{array}{c} \mathbf{e}_m \\ \hline \mathbf{e}_1 \end{array}\right].$$

Here, $\mathbf{e}_i$ is the i^{th} elementary unit vector. It follows that

$$\begin{aligned} A &= \tilde{Q}\tilde{D}\tilde{Q}^T + \rho \mathbf{w}\mathbf{w}^T \\ &= \tilde{Q}(\tilde{D} + \tilde{Q}^T \rho \mathbf{w}\mathbf{w}^T \tilde{Q})\tilde{Q}^T \\ &= \tilde{Q}(\tilde{D} + \rho \mathbf{z}\mathbf{z}^T)\tilde{Q}^T \end{aligned} \qquad (2)$$

where

$$\mathbf{z} = \tilde{Q}^T \mathbf{w} = \begin{bmatrix} \text{last column of } \tilde{Q}_1^T \\ \text{first column of } \tilde{Q}_2^T \end{bmatrix}.$$

Thus, it suffices to compute the eigendecomposition of the matrix $\tilde{D} + \rho \mathbf{z}\mathbf{z}^T$. If $\tilde{D} + \rho \mathbf{z}\mathbf{z}^T = \hat{Q} D \hat{Q}^T$, then the eigendecomposition of A is given by

$$\begin{aligned} A = \tilde{Q}(\tilde{D} + \rho \mathbf{z}\mathbf{z}^T)\tilde{Q}^T &= \tilde{Q}\hat{Q} D \hat{Q}^T \tilde{Q}^T \\ &= Q D Q^T \end{aligned} \qquad (3)$$

where $Q = \tilde{Q}\hat{Q}$.

It only remains to find the eigenvalues and eigenvectors of $\tilde{D} + \rho \mathbf{z}\mathbf{z}^T$. This task consists of three distinct subtasks:

2.1 Perform deflations

Let d_i's be the entries of the diagonal matrix D and the z_i's be the entries of the vector $\mathbf{z}$:

$$\tilde{D} = \text{diag}(d_1, d_2, \cdots, d_n), \quad \mathbf{z} = [z_1 \quad z_2 \quad \cdots \quad z_n]^T.$$

It turns out that, whenever $d_i = d_{i+1}$ or $z_i = 0$ for some i, we get an eigenvalue for free: d_i itself is an eigenvalue of $\tilde{D} + \rho \mathbf{z}\mathbf{z}^T$. Furthermore, the corresponding eigenvector is either $\mathbf{e}_i$ (if $z_i = 0$) or some rotation of it (if $d_i = d_{i+1}$). This phenomenon is called *deflation*. In practice, deflations occur frequently, when $|d_i - d_{i+1}|$ or $|z_i|$ is small enough.

The major saving occurs in the matrix multiplication step in (3): we can leave out the i-th eigenvalue and eigenvector from the computation of $\hat{Q}$ [2]. Instead, we infer their values directly from $\tilde{Q}$, which is already available. Hence, we skip the corresponding rows and columns when we compute $Q = \tilde{Q}\hat{Q}$. In this way, matrix multiplication in (3) can be accelerated so that the whole algorithm costs only $O(n^{2.3})$ in time instead of $O(n^3)$.

Let $T + \rho \mathbf{u}\mathbf{u}^T$ be the submatrix that is the result of deflating the matrix $\tilde{D} + \rho \mathbf{z}\mathbf{z}^T$:

$$T = \text{diag}(\delta_1, \delta_2, \cdots, \delta_k) \text{ and } \mathbf{u} = [\zeta_1 \quad \zeta_2 \quad \cdots \quad \zeta_k]^T$$

where $\delta_1 < \delta_2 < \cdots < \delta_k$ and $\zeta_i \neq 0$ for all i.

2.2 Computing the eigenvalues via the secular equation

Let λ be an eigenvalue of $T + \rho \mathbf{u}\mathbf{u}^T$ with an associated eigenvector $\mathbf{q}$. Then by definition,

$$(T + \rho \mathbf{u}\mathbf{u}^T)\mathbf{q} = \lambda \mathbf{q}, \qquad (4)$$

so that

$$T\mathbf{q} + \rho(\mathbf{u}^T\mathbf{q})\mathbf{u} = \lambda \mathbf{q} \qquad (5)$$

It turns out that $\mathbf{u}^T\mathbf{q} \neq 0$; otherwise, $\lambda = \delta_i$ for some i and $\zeta_i = 0$, which contradicts the conditions of $T + \rho \mathbf{u}\mathbf{u}^T$. Similarly, we conclude that $\lambda \neq \delta_i$ for all $i = 1, \cdots, k$.

Since $\lambda \neq \delta_i$ for all i, the diagonal matrix $T - \lambda I$ has no zero entry and its inverse is well defined. With some algebra, it is possible to show that (5) is equivalent to

$$1 + \rho \mathbf{u}^T (T - \lambda I)^{-1} \mathbf{u} = 0 \qquad (6)$$

This equation is equivalent to a rational equation known as *the secular equation*:

$$1 + \rho \sum_{i=1}^{n} \frac{\zeta_i^2}{\delta_i - \lambda} = 0 \qquad (7)$$

The k solutions of the secular equation give the eigenvalues of $T + \rho \mathbf{u}\mathbf{u}^T$. The equation can be solved by a variant of the Newton-Raphson method in which approximating lines are replaced with approximating rational asymptotes. Li [5] lays out the full details of a secular equation solver and offer solutions to common issues in numerical stability of the algorithm.

2.3 Computing the eigenvectors

Once we obtain the eigenvalues λ_i of $T + \rho \mathbf{u}\mathbf{u}^T$, we compute the corresponding eigenvectors $\mathbf{q}_i$. In theory, $(T - \lambda_i I)^{-1}\mathbf{u}$ gives an eigenvector of λ_i:

$$\begin{aligned}&(T + \rho \mathbf{u}\mathbf{u}^T)[(T - \lambda I)^{-1}\mathbf{u}]\\&= ((T - \lambda I) + \lambda I + \rho \mathbf{u}\mathbf{u}^T)[(T - \lambda I)^{-1}\mathbf{u}]\\&= \lambda[(T - \lambda I)^{-1}\mathbf{u}]\end{aligned} \tag{8}$$

Unfortunately, two computed eigenvectors are not numerically orthogonal whenever their associated eigenvalues are close to each other. Gu and Eisenstat [6] proposed a more stable way to compute numerically orthogonal eigenvectors of $T + \rho \mathbf{u}\mathbf{u}^T$. In a nutshell, their approach amounts to solving an *inverse eigenvalue problem*: Let $\lambda_1, \lambda_2, \cdots, \lambda_k$ be the roots of the secular equation (7). Let $\hat{\mathbf{u}}$ be the vector whose k entries are given by

$$\hat{u}_i = \sqrt{\frac{\prod_{j=1}^{k}(\lambda_j - \delta_i)}{\rho \prod_{\substack{j=1\\ j\neq i}}^{k}(\delta_j - \delta_i)}} \tag{9}$$

Also, let

$$\Lambda = \operatorname{diag}(\lambda_1, \lambda_2, \cdots, \lambda_k).$$

Then the matrix $\Lambda + \hat{\mathbf{u}}\hat{\mathbf{u}}^T$ has $\lambda_1, \lambda_2, \cdots, \lambda_k$ as its eigenvalues. Furthermore, $(\Lambda - \lambda_i I)^{-1}\hat{\mathbf{u}}$ gives a numerically stable eigenvector of each eigenvalue λ_i.

3. Task organization

Using LAPACK routine `dstedc` as a guide [7], we organize divide-and-conquer algorithm in the following modules:

- `dlaed0`: Split the given problem into 128×128 subproblems, performing appropriate rank-one cuts. Then compute the eigendecomposition of each 128×128 subproblem by calling the QR routine `dsteqr`. Finally, let `dlaed1` merge the eigendecompositions of adjacent submatrices until we have the eigendecomposition of the original matrix.
- `dlaed1`: Coordinate subtasks necessary to merge the eigendecompositions of two adjacent submatrices. Specifically,
  - Produce $\tilde{D} + \rho \mathbf{z}\mathbf{z}^T$ from A via (2).
  - Call `dlaed2` to carry out Subtask 2.1
  - Call `dlaed3` to carry out Subtasks 2.2 and 2.3.
  - *Back-transform* the eigenvector collection $\hat{Q}$ of $\tilde{D} + \rho \mathbf{z}\mathbf{z}^T$ by multiplying with $\tilde{Q}$, as described in (3).
- `dlaed2`: Perform deflation as given by Subtask 2.1. To differentiate between deflated eigenvalues and eigenvectors from non-deflated ones, we maintain an ordered list of eigenvalues [7]. Each time we deflate an eigenvalue, we remove it from the ordered list and put it at the end of the list; in other words, we permute the list. Let σ be the permutation that results from deflation.
- `dlaed3`: Compute the eigendecomposition $\tilde{D} + \rho \mathbf{z}\mathbf{z}^T = \hat{Q}D\hat{Q}^T$ by carrying out Subtasks 2.2 and 2.3. Now that all the deflated eigenvalues are at the end of the list, we can focus on the non-deflated portion, i.e. $T + \rho \mathbf{u}\mathbf{u}^T$. The $\mathbf{u}$ vector is given by $\mathbf{z}$ with σ applied. More specifically, `dlaed3` does the following steps:
  - Call `dlaed4` to compute each root λ_i of the secular equation.
  - Solve the inverse eigenvalue problem to compute numerically orthogonal eigenvectors that correspond to λ_i's.

  After `dlaed3` returns, `dlaed1` should re-merge the deflated eigenvalues back into the middle of the list.
- `dlaed4`: Compute the i-th root λ_i of the secular equation. We use an iteration scheme known as the Middle Way, where we create a series of approximating rational functions whose asymptotic poles match those of the secular equation near λ_i [5].

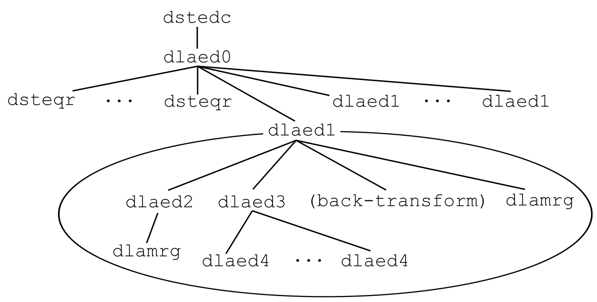


Fig. 1: A call graph of `dstedc`.

4. Parallel Implementation on Multiple GPUs

The key idea is to fix the size of GPU workspaces so that we do not run out of GPU memory regardless of the size of input matrices. Now the only limiting factor is the main memory, which in many systems is in plentiful supply. Since the input can be of any size but GPU workspaces are not, it is crucial to build a dynamic partition of tasks. More importantly, the nature of work changes as the algorithm progresses.

Like other algorithms of its kind, divide-and-conquer algorithm starts with many small base cases (cf. Fig. 2).

As small systems are merged into larger ones, there would be fewer and fewer subproblems left. In other words, the work at hand gradually becomes more coarse-grained. Thus, we need to adapt the way we allocate tasks depending on the current size of subproblems.

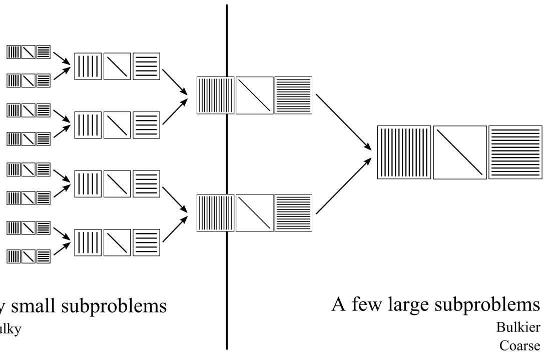


Fig. 2: A schematic of divide-and-conquer algorithm

Before we discuss dynamic allocation of tasks, let us briefly look at CUDA™, a general-purpose GPU computing platform.

4.1 CUDA programming environment

Graphical processing units (GPUs) are commodity hardware that were originally designed to accelerate graphics applications. In recent years, a number of non-graphical, computationally expensive algorithms have been implemented on GPUs [8]. In particular, NVIDIA offers a general-purpose API called CUDA™. All recent NVIDIA graphics cards support this interface.

GPUs are massively parallel processors in which many small worker threads execute in parallel. While each thread may not be as powerful as a typical CPU core, the collaboration of many threads helps achieve a high throughput. GPUs follow a data parallelism paradigm in which each worker thread executes a similar set of instructions but processes its own portion of data.

In typical circumstances, a GPU does not launch its own work. Instead, a CPU thread launches a *kernel*, or a subroutine, to be executed on a selected GPU. The CUDA runtime launches multiple instances of the kernel to be run by the GPU threads. Kernel launch parameters determine how many GPU threads are launched and how they are organized.

A defining characteristic of GPU programming is that GPUs have memory spaces separate from the main memory. GPUs cannot access the main memory directly; instead, content has to be copied from the main memory to the GPU memory first. This step is essential in supporting a large degree of parallelism, as the GPU processing cores require a dedicated memory designed for high bandwidth. The data transfer passes through the PCI Express channel, making the operation relatively costly. Furthermore, the size of GPU memory is also a limit; even high-end models carry only a few gigabytes of dedicated memory. To make matters worse, on systems with multiple graphics cards installed, the GPUs have memory spaces separate from one another. Thus, a CPU thread that copies a buffer into a GPU memory needs to designate a specific target GPU.

4.2 Dynamic block partition of back-transform

A computational bottleneck in divide-and-conquer algorithm is the back-transformation step at the end of `dlaed1`. When only a few eigenvalues deflate, its cost approaches $O(n^3)$ flops, where n is the number of eigenvalues. Fortunately, this step is a BLAS 3 operation and scales well on GPUs. It is where MAGMA makes most use of GPUs [3], and we intend to do so as well.

Given an $n \times k$ transformation matrix $\tilde{Q}$ and a $k \times k$ collection $\hat{Q}$ of eigenvectors, the transformed eigenvectors are given by the product $\tilde{Q}\hat{Q}$. Both n and k change over time, n being the size of subproblems at the current level and k being the number of non-deflated eigenvalues. Let G be the greatest integer such that three $G \times G$ matrices fit into a single GPU device's memory. Let D be the number of GPU devices installed. The idea is to pick a multiple of D that is large enough so that

$$\frac{n}{aD} \leq G.$$

Then a block matrix of dimension $n/aD \times k/aD$ will certainly fit into a single GPU device. Let A_{ij} and B_{ij} be block matrices of $\tilde{Q}$ and $\hat{Q}$, respectively, where each A_{ij} is $n/aD \times k/aD$ and each B_{ij} is $k/aD \times k/aD$. We now have a conformable partition of matrix multiplication.

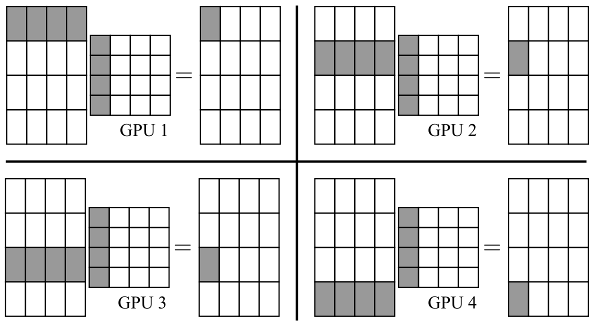


Fig. 3: Out-of-core block multiplication using 4 GPUs

Since we partitioned the matrix product in multiples of D, it is straightforward to assign block multiplications to the D GPU devices (cf. Fig. 3). Notice that no more than one A block and one B block need to be resident in each GPU device at any moment. Once each partial product $A_{ip}B_{pj}$ $(1 \leq i,\ j,\ p \leq aD)$ is computed, the corresponding block C_{ij} can be incremented by that amount. Matrix multiplication is supported by cuBLAS [9], a fast GPU implementation of BLAS interface.

4.3 Fine-grained parallelism

Out-of-core matrix multiplication is fairly inefficient when the operands are small — there is a constant overhead of moving block matrices back and forth between the GPU and CPU memories. In addition, we have to compute the matrix partition for each subproblem. If we could keep everything in one place, we would be able to eliminate all the overhead.

To hide the overhead, we let the GPUs solve multiple subproblems in parallel, each GPU solving one subproblem. The benefit of such approach is two-fold. First, we avoid performing out-of-core matrix multiplication when it is not necessary. Second, we hide the latency of data transfer by overlaying multiple subproblems on top of each other. For instance, at the moment when GPU 1 is fetching a workspace from the main memory, GPU 2 may be decomposing another matrix. Fig. 4 shows a visual representation of overlapping merge tasks. The CUDA toolkit incorporates a visual profiler capable of drawing a timeline of kernel launches [10].

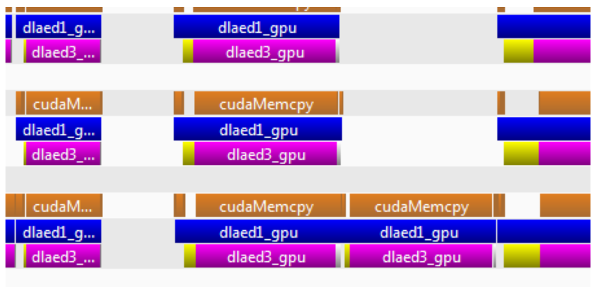


Fig. 4: Overlapping of multiple merge tasks

Unfortunately, certain parts of divide-and-conquer do not scale well on GPUs. Especially, deflation process involves construction of permutations and has to be done serially. We solve this problem by paring each GPU device with a host thread and forming a compute group. On the other hand, both secular equations and inverse eigenvalue problems can be solved efficiently in bulk parallel fashion by GPUs: each λ_i can be computed independent of other λ_j's, and similarly with the eigenvectors.

4.4 Profiling

Extending the idea of simultaneous merging, we also make use of idle CPU cores and form compute groups as well. An added difficulty is that performance scales at different rates on GPU-CPU groups and on CPU-only groups. We overcome this difficulty by constructing linear regression models of respective compute groups.

Consider GPU-CPU groups first. We define two independent variables that affect performance: let X_1 be the subproblem size and X_2 be the number of GPU-CPU groups. The dependent variable is Y, the time it takes to solve a subproblem of size X_1 using X_2 groups. We model their relationship by a power function of form

$$Y = X_1^{\alpha_1} X_2^{\alpha_2} 2^{\alpha_3}.$$

where $\alpha_1, \alpha_2, \alpha_3$ are parameters to be fitted. Similarly, let X_3 be the number of CPU-only groups and Z be the time it takes for X_3 CPU groups to solve subproblem of size X_1:

$$Z = X_1^{\beta_1} X_3^{\beta_2} 2^{\beta_3}.$$

The models reflect our intuition to some degree: for instance, if performance were to scale linearly with respect to the number of groups, Y would be proportional to X_2^{-1}, suggesting $\alpha_2 \approx -1$. In addition, the $O(n^{2.3})$ work complexity of the algorithm suggests that the scaling of Y is some multiple of that of X_1.

Each of the models is nonlinear on its own, but we can easily transform it into a linear model. Taking the logarithm of both sides gives

$$\log Y = \alpha_1 \log X_1 + \alpha_2 \log X_2 + \alpha_3$$
$$\log Z = \beta_1 \log X_1 + \beta_3 \log X_3 + \beta_3.$$

Now the parameters can be fitted using the method of least squares. Given the parameters, we estimate the ratio R between performance of GPU-CPU groups and that of CPU-only groups:

$$R = \frac{Z}{Y} = X_1^{\beta_1 - \alpha_1} X_3^{\beta_2} X_2^{-\alpha_2} 2^{\beta_3 - \alpha_3}$$

The ratio enables our implementation to balance loads by allocating the right number of subproblems to each kind of compute groups. Our code package incorporates a separate profiler that runs test matrices and computes the parameters. It saves the parameters to a configuration file so that the main subroutine could load them at startup.

5. Performance

Our machine comprises a dual 2.0 GHz Intel® Xeon® E5-2620 CPU and four NVIDIA® Tesla® K20c graphics cards. The machine was configured with 64 GB main memory and 5 GB memory for each GPU. Our experiments used double-precision floating point arithmetic. A package containing the full source code and the performance profiler is available at `https://github.com/hcho3/dstedc_mgpu`.

Prior work such as [3] show that the empirical complexity of divide-and-conquer algorithm depends on the characteristics of the input matrix. If a significant portion of the eigenvalues of a subsystem deflate out, the cost is closer to $O(n^2)$ rather than $O(n^3)$. For the purpose of this experiment, we choose a simple random sample $\lambda_1, \cdots, \lambda_n$ from the standard normal distribution and generate a symmetric tridiagonal matrix whose eigenvalues are λ_i's. For all the test matrices we generated this way, 8-12% of eigenvalues deflate.

Despite the limited amount of memory available on GPU devices, our implementation was able to handle up to 50,000×50,000 input matrices, for which outputs and workspaces combined occupied 85% of the main memory. On the GPU side, only 3.9 GB out of 5 GB was used. However, MAGMA's implementation could not handle input matrices larger than 36,000. Fig. 5 illustrates how our implementation handles large matrices stably even in the face of limited GPU memory.

Table 1: Performance for various test matrices

	Matrix dimension	Performance (sec)		
		Hybrid	CPUs only	Speedup
In-core	1024	0.98	0.34	0.35
	2048	1.80	0.93	0.52
	4096	3.84	3.83	1.00
	8192	9.28	17.43	1.88
Out-of-core	16384	26.91	99.80	3.71
	32768	103.00	681.56	6.62
	36000	117.38	867.70	7.39
	50000	239.36	2278.90	9.52

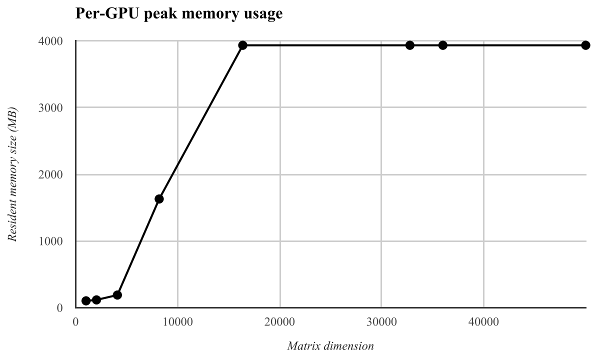


Fig. 5: Average GPU memory consumption for different input sizes

To put our implementation's performance in context, we created a version that exclusively uses CPU cores (cf. Table 1). For smallest input matrices, the CPU-only version shows better performance. One significant factor is that, unlike GPUs, CPU cores share the same memory space. So when the algorithm progresses from one level to next, it is possible to re-group the CPU cores to form fewer compute groups. Also, the overhead of setting up multiple CUDA contexts is absent.

On the other hand, the hybrid version does better for 8192×8192 input matrices and larger, as the back-transformation step takes a growing share of flops. At the same time, matrix multiplication is a bulk parallel task and scales well on GPUs. The performance profile is illuminating in that regard: the values of α_i's and β_i's were respectively

$$\alpha_1 = 0.978, \alpha_2 = -0.916, \alpha_3 = -11.884$$
$$\beta_1 = 2.401, \beta_2 = -0.529, \beta_3 = -26.788.$$

This means that each time the subproblem size was doubled, GPU-CPU groups spent only twice as much time as it had, whereas CPU-only groups had to spend 5.3 times as much. The model produced a good fit for the data points of the profile, giving R-squared coefficients of 0.984 and 0.996 for GPU-CPU groups and CPU-only groups, respectively.

6. Conclusion

In this paper, we presented a memory-efficient implementation of divide-and-conquer algorithm on multi-GPU systems. Our implementation made use of both multiple GPUs and multicore CPUs. We overcame the limitations in GPU memory by fixing GPU workspaces to a size independent of subproblem size. This approach allowed our implementation to handle input matrices as large as 50,000×50,000.

Furthermore, despite the added complexity caused by the fixed size of GPU workspaces, our implementation exhibited a significant speedup for large input matrices compared to a version that used multicore CPUs exclusively. At the same time, we allocated tasks for the fine-grained portion of the algorithm. By solving multiple subproblems simultaneously, some on GPUs and some on CPUs, our implementation solve small problems at a rate comparable to the case where only CPUs are used.

References

[1] D. S. Watkins, *Fundamentals of Matrix Computations* (New York: John Wiley and Sons Incs., 1991).
[2] J. Demmel, *Applied Numerical Linear Algebra* (Philadelphia: SIAM, 1997).
[3] C. Vomel, S. Tomov and J. Dongarra, Divide and conquer on hybrid GPU-accelerated multicore systems, *SIAM Journal on Scientific Computing*, 34(2), 2012, C70-C82
[4] MAGMA Library, version 1.4.1. URL: http://icl.cs.utk.edu/magma/
[5] R-C. Li, Solving secular equations stably and efficiently, LAPACK Working Notes, 1994.
[6] M. Gu, S.C. Eisenstat, A stable and efficient algorithm for the rank-one modification of the symmetric eigenproblem, *SIAM Journal on Matrix Analysis and Applications*, 15(4), 1994, 1266-1276
[7] J. Rutter, A serial implementation of Cuppen's divide and conquer algorithm for the symmetric eigenvalue problem, LAPACK Working Notes, 1994.
[8] D. B. Kirk and W. W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach* (Burlington, MA : Morgan Kaufmann Publishers, 2010).
[9] NVIDIA, cuBLAS Library User Guide. http://docs.nvidia.com/cuda/pdf/ CUBLAS_Library.pdf.
[10] NVIDIA, Profiler User's Guide. http://docs.nvidia.com/cuda/pdf/ CUDA_Profiler_Users_Guide.pdf
[11] J. Dongarra, T. Dong, M. Gates, A. Haidar, S. Tomov, I. Yamazaki, MAGMA: a new generation of linear algebra library for GPU and multicore architectures, *Supercomputing*, Salt Lake City, Utah, 2012.
[12] I. Yamazaki, T. Dong, R. Solca, S. Tomov, J. Dongarra and T. Schulthess, Tridiagonalization of a dense symmetric matrix on multiple GPUs and its application to symmetric eigenvalue problems, *Concurrency and Computation: Practice and Experience*, published online, 2013, DOI: 10.1002/cpe.3152
[13] NVIDIA, CUDA C Programming Guide.
[14] E. Agullo, J. Demmel, J. Dongarra, B. Hadri, J. Kurzak, J. Langou, H. Ltaief, P. Luszczek and S. Tomov, Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects, *Journal of Physics: Conference Series*, 180(1), 2009, 12-37.

Computation of Control Related States of Top Left K-net System (with a Nonsharing Resource Place) of Petri Nets

Daniel Yuh Chao, Tsung Hsien Yu, and Ding Chun Huang
Department of Management and Information Science
National Cheng Chi University, Taipei, Taiwan, ROC

Abstract - *Earlier, Chao pioneered the very first closed-form solution of the number of reachable and other states for marked graphs (MG) and k-th order system which is the simplest class of S^3PR (Systems of Simple Sequential Processes with Resources). This paper progresses one step further on enumerating reachable (forbidden, live and deadlock) states for top Left k-net systems (one non-sharing resource place in the top position of the left-side process, below denoted as Top-Left-k-net) with a formula depending on parameter k for a subclass of nets with k sharing resources.*

Keywords: Control systems, discrete event systems, flexible manufacturing systems, Petri nets

1 Introduction

PETRI nets (PN) have been used for modeling and analyzing concurrent systems, such as flexible manufacturing (or resource allocation) systems (FMS) (or RAS) [1-9]. Reachability [11-16] can be used to verify system properties of liveness, boundedness, reversibility, and so on. However, the persistent problem of using PN for modeling various systems is the large number of states generated (called the state explosion problem). It has been shown that the complexity of the reachability problem of a Petri net is EXPSPACE-hard in [12]. Lee *et al.* [12] show that the reachability problem (whether a marking is reachable) is NP-complete for even a live and safe Free Choice net (LSFC).

To efficiently break the ever exponential time plight of Petri nets, Chao [20, 21, 22] pioneered the very first solution for S^3PR using closed-form methodology by applying simple graph theory and combinatorial mathematics. Here, we extend one step further by constructing a closed form formula of Left k-net systems (*Top-Left-k-net*) with r^* on the top position of the left-side process.

The approach is explained as follows. The only one token at each r_i initially can stay at r_i, p_i (left holder place), or p'_i (right holder place); i.e., 3 possibilities. All together, there are 3^k possibilities or states. But some states are not reachable from initial marking via a certain firing sequence. It is easy to find and enumerate the patterns of token distribution for reachable ($\check{R}$) and live (L) states. From that, one can infer the number F of forbidden states as their difference ($F=\check{R} - L$). The number of nonreachable states is $3^k - \check{R}$. Reachability problem becomes trivial as one can check whether the marking fits the reachable pattern and hence whether it is reachable.

The rest of the paper is organized as follows. We first list important contributions of our series papers in section II. Based on the results obtained and the methodology of [22], we will then enumerate reachable, forbidden, and live states of *Top-Left-k-net* system in section III. Finally, Section VI concludes the paper.

2 Contributions of our series papers [20, 22, 24, 25]

Here we define the *k*-th order system with one non-sharing resource place.

Definition 1: *A k-th order system is a subclass of S^3PR with k resource places $r_1, r_2, ..., r_k$ shared between two processes N_1 and N_2 and one non-sharing resource place r'_h ($=r^*$) used by an operation place p^* in P_1 or P_2*

1. *$M_0(r'_h) = 1$ and $\forall r \in P_R$, $M_0(r) = 1$.*
2. *N_1 (resp. N_2) uses $r_1, r_2, ..., r_k$ (resp. $r_k, r_{k-1}, ...r_2, r_1$) in that order.*
3. *$M_0(p_0) = k+1$, $M_0(p'_0) = k$, where p_0 and p'_0 are the idle places in processes N_1 and N_2, respectively.*
4. *Holder places of r_j in N_1 and N_2 are denoted as p_j and p'_j respectively.*
5. *The compound circuit containing $r_i, r_{i+1}, ..., r_{j-1}, r_j$ is called (r_i-r_j)-region.*
6. *If r'_h does not exist, then it is called a k-th order system.*
7. *There are 3 possibilities for the token initially at r_i to sit at: p_i (N_1), p'_i, (N_2),and r_i. The corresponding token or r_i state is denoted by 1, -1 and 0, respectively.*
8. *x^y means r_h is at x state (x=1,0,-1) and r^* is at y state (y=1, 0,-1), where h is the location of non-sharing resource being used by an operation place p^*. The system is denoted as Top-Left k-th order system when h=1 and p^* in P_1; Top-Right k-th order system when h=1 and p^* in P_2; Bottom-Left k-th order system*

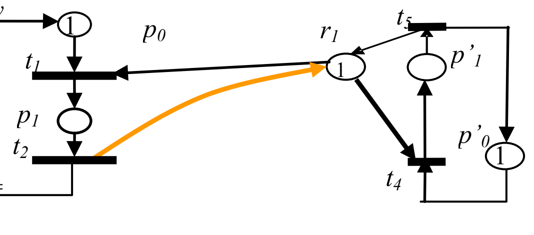


Fig. 1. 1st-order system.

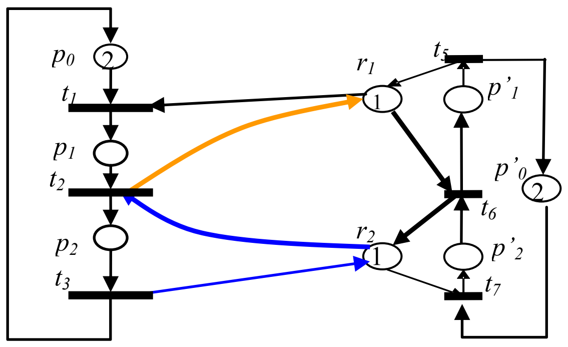


Fig. 2. 2nd-order system.

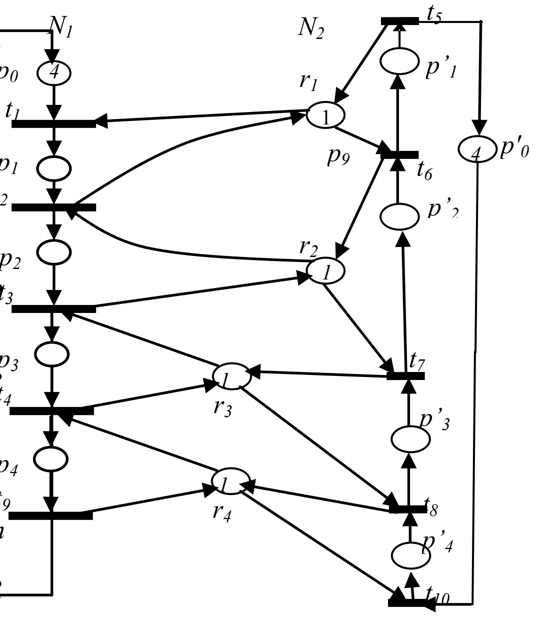


Fig. 4 4-th order system.

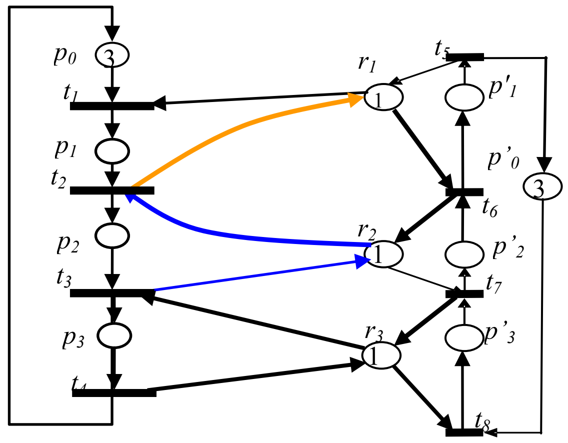


Fig. 3. 3rd-order system.

when h=k and p* in P1; Bottom-Ri

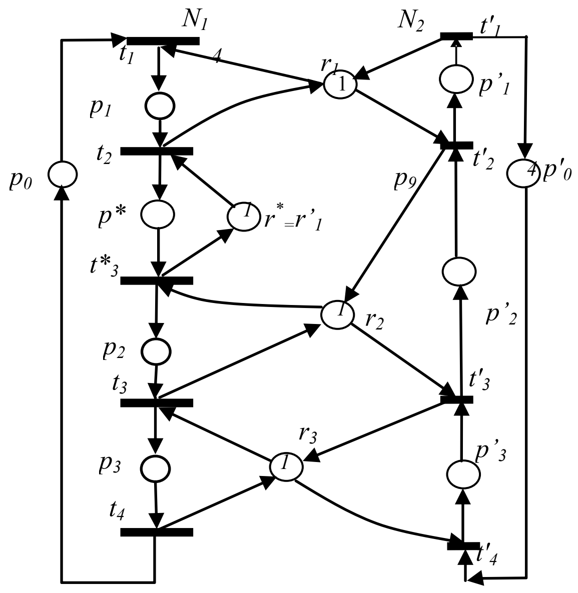


Fig. 5(a) *Top-Left* 3-th order system *N*.

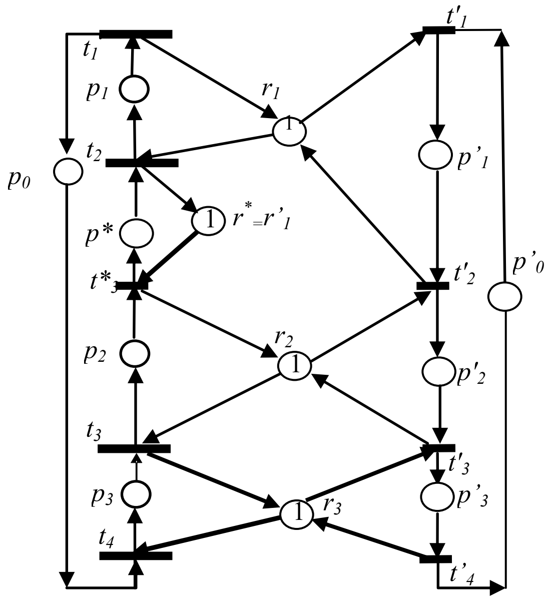


Fig. 5(b) *Top-Left* 3-th order top-system reverse N^r.

ght k-th order system when h=k and p in P_2.*

Examples are shown in Figs. 1-5.

2.1 *K-th order system [20]*

Let *N* be a Petri net; N^r is the reverse net of *N*. By the concept of complete reachability graph (Fig. 6) that contains live, forbidden and nonreachable states, we have *Lemma 1, Lemma 2 and Theorem 1.*

Lemma 1: Any forbidden state in N is nonreachable in N^r.

Lemma 2: Any nonreachable state s in N is a forbidden one or a nonreachable one in N^r.

Theorem 1: $\vartheta(k)=¥(k)-B(k)$, *where* $\vartheta($ ***k), ¥(k), and B(k)*** *are the number of* ***forbidden, nonreachable, and nonreachable +empty-siphon states*** *in a k-th order system, respectively.*

In Fig. 5, Deadlock (resp. forbidden but not deadlock, live) states are the nodes that are pointed by a dashed line from *D(k)*. Pointed from *¥(k)* and *B(k)* are nonreachable states. States nonreachable in both *N* and N^r are (*1,-1,1*) and (*-1 1 -1*). Note that there are no directed paths 1) from a forbidden state to live states, and 2) from reachable states to nonreachable ones.

For the 3rd order system, there are 3 kinds of unmarked (resp. nonreachable) siphon states: (*1 -1 x*), (*x 1 -1*), and (*1 0 -1*) [resp. (*-1 1 x*), (*x -1 1*), and (*-1 0 1*)], where *x=-1, 0, 1*.

Definition 2: $s = (x_1\ \ x_2\ \ldots\ x_k)$, $x_i=1, 0, -1$, $k\geq i\geq 1$ *is a state for a k-th order system N.* $(x_i\ \ x_{i+1}\ \ldots\ x_q\ \ x_{q+1})$, $k\geq q\geq i\geq 1$ *(embedded in s) is a substate of s.*

By Definition 2, we have some characteristics of nonreachable and forbidden states of a *k*-th order system.

A substate of (-1 x x ... x 1) (x=1 0 -1) corresponds to a nonreachable state.

A substate of (1 x x ... x -1) (x=1 0 -1) corresponds to a forbidden or a nonreachable state.

State s=(x x ... x 1 x x ... x -1 x x ...x 1 x x ... x -1 x x ... x) cannot be a reachable state. It means that a reachable state cannot have two substates of (*1 x x ... x -1*).

If $s = (x_1\ \ x_2 \ldots x_{i-1}\ \ 1_i\ \ x_{i+1}\ \ x_{i+2}\ \ldots x_k)$, *does not carry a substate of* $(1_g\ \ x_{g+1}\ \ x_{g+2}\ \ldots x_k)$, $g>i$, *then s with* $x_m=0$ *or 1,* $m=1$ *to i-1 and* $x_j=0$ *or -1,* $j=i+1$ *to k are the only reachable states.*

A deadlock state has the pattern: $(1_1\ 1_2\ \ldots\ 1_m\ -1_{m+1}\ -1_{m+2}\ \ldots\ -1_k)$, $1\leq m<k$.

Finally , shown below is the total number of each type of states in a *k*-th order system that we proved in [20].

The total number of states is 3^k.

The total number of live states $L(k) = 2^{k+1}-1$.

The total number of reachable states $R(k) = (k+2)2^{(k-1)}$.

The number of forbidden states $\vartheta(k) =(k-2)2^{(k-1)}+1$.

The number of nonreachable states $¥(k) = 3^k - (k+2)2^{(k-1)}$.

The number of nonreachable +empty-siphon states $B(k) = 3^k - k2^k - 1$.

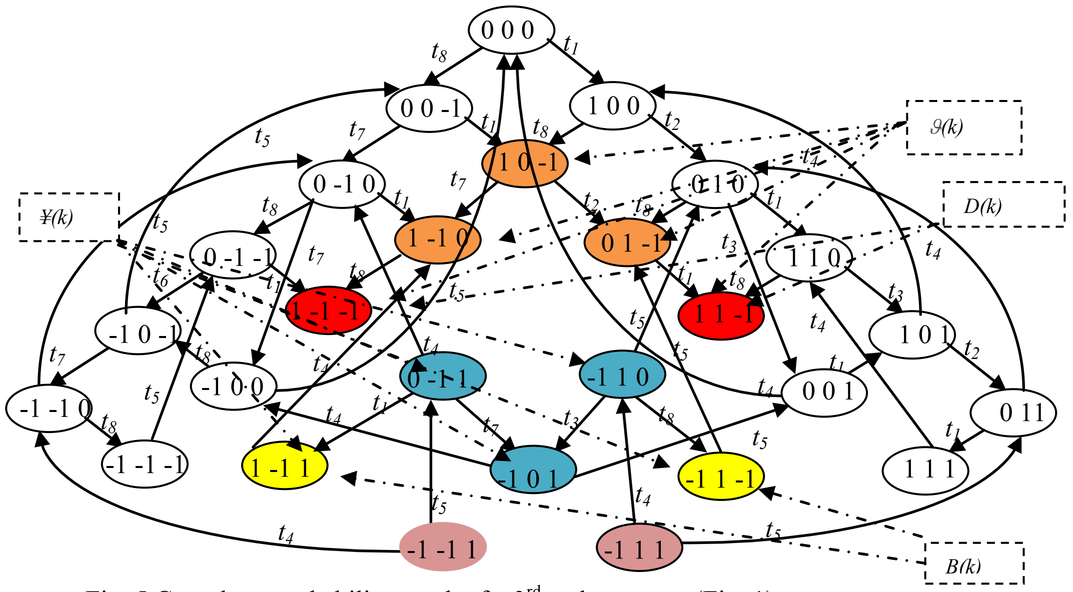


Fig. 5 Complete reachability graph of a 3rd-order system (Fig. 1).

The total number of deadlock states D(k)=k-1.

2.2 *Top-Left, Bottom-Left, and Middle-Left system* [22, 24, 25]

Definition3: *The equivalent* $N^e = (P^e \cup P^e{}_R, T^e, F^e)$ *of a net* $N = (P \cup P_R, T, F)$ *(*P_{NR} *is the set of non-sharing places) is defined as*

1. $P^e{}_R = P_R \setminus P_{NR}$;
2. $P^e = P \setminus \bigcup_{r \in P_{NR}} H(r)$;
3. $T^e = T \setminus \bigcup_{r \in P_{NR}} r^\bullet$;
4. $F^e = (F \bigcup_{r \in P_{NR}} ({}^\bullet r, r^{\bullet\bullet}) \cup ({}^\bullet(r^\bullet), {}^\bullet r) \setminus \bigcup_{r \in P_{NR}} [(H(r), H(r)^\bullet) \cup ({}^\bullet H(r), H(r)) \cup ({}^\bullet r, r) \cup (r, r^\bullet) \cup (r^\bullet, r^{\bullet\bullet}) \cup ({}^\bullet(r^\bullet), r^\bullet)]$;

Definition 4: The reverse net of N^e *is denoted as* N^{er}.

We say the net in Fig. 3 (*k*-th order system) is the equivalent of the net in Fig. 5(a), since the Fig. 3 net is exactly the same as the net in Fig. 5(a), except that the nets has one non-sharing resource place *r**.

Let *N* be a net that contains a nonsharing resource in the left process side (for example *Top-Left, Bottom-Left, and Middle-Left*). Since there are forbidden states in N^e (due to empty-siphon), but live (due to marked siphon) or reachable in *N* since an empty siphon in N^e may become marked in *N*. We have shown that in *N* the number of reachable states (*R'*) is *>2R* and the number of live states (*L'*) *>2L*. To compute *R'* and *L'* we need to know how many forbidden and nonreachable states in N^e, become reachable or live in *N*.

Because of a nonsharing resource, we have shown that: 1) markings nonreachable in N^e, may become reachable in *N*(denoted the number of which as *Θ(k)*); 2) forbidden markings in N^e may be live in *N* (denoted the number of which as *C(k)*); 3) nonreachable markings in N^e may be live in *N* (denoted the number of which as *A(k)*). We have

$$R' = 2R + \Theta(k). \qquad (1)$$

$$L' = 2L + A(k) + C(k). \qquad (2)$$

The phenomenon is explained as follows:

C(k): In *Top-Left* structure, by holding at p_1 left-side process can wait for right-side process to go through their own work flow. The set of current states belongs to the set of forbidden states in N^e. While after firing t_2 in *Top-Left,* it may be live in *N* [22].

A(k): In *Bottom-Left* structure, by holding at p^* left-side process can wait for right-side process to go through their own work flow. While after firing t^*_{k-1} in *Bottom-Left*, set of succeeding states belongs to set of unreachable states in N^e [24].

Shown below is methodology of *Top-Left* [22].

We will derive *R'(ϑ', L', ...etc)* in terms of *R(ϑ, L, ...etc)* based on the concept of equivalent *k*-th order system of a *Top-Left k*-th order system.

For the 3rd order system, there are 3 kinds of unmarked (resp. nonreachable) siphon states: ($1\ {-1}^1\ x$), ($x\ 1\ {-1}$), and ($1\ 0^1\ {-1}$) [resp. ($-1\ 1^1\ x$), ($x\ {-1}\ 1$), and ($-1\ 0^1\ 1$)], where *x=-1, 0, 1.*

A substate of $(-1\ x^1\ x \ldots x\ 1)$ $(x=1,0,-1)$ *corresponds to a nonreachable state.*

A substate of $(1\ x^1\ x \ldots x\ {-1})$ $(x=1,0,-1)$ *corresponds to a forbidden or a nonreachable state.*

Let M be a reachable marking in N^e, *then both M*=M+r* and M'= M+p* are reachable in N.*

Both $s=(1\ {-1}^0\ {-x_3}\ {-x_4} \ldots x_{j-1}\ x_j)$ *and* $s'=(-1\ 0^1\ {-1_3}\ x_4 \ldots x_{j-1}\ x_j)$ *where* $x_i = 0$, *or -1, correspond to two legal markings M.*

Let M be such that only the top r_1-r_2 *region in* N^{er} *is unmarked.*
1) *M is nonreachable in* N^e.
2) *M*=M+r* is reachable in N.*

Let $s=(-1\ 0^0\ 0_3\ 0_4 \ldots 0_{j-1}\ 1_j\ x_{j+1}\ x_{j+2} \ldots x_k)$ *be such that only the top r1-rj siphon in* N^{er} *is unmarked.*
1) *M is nonreachable in* N^e.
2) *M*=M+r* is reachable in N.*
3) *The total number of such M* is* $2^{(k-j)}$.

The total number of reachable states in N is $R'=2R+ 2^{(k-1)}-1 =(2k+5)2^{(k-1)}-1$.

Let $s=(1\ 0^0\ 0_3\ 0_4 \ldots 0_{j-1}\ {-1_j}\ x_{j+1}\ x_{j+2} \ldots x_k)$ *correspond to Marking M such that there are unmarked siphons in only the top* r_1-r_2 *region in* N^{er}. *The total number of possible live markings under M is* 2^{k-j}.

The total number of forbidden markings in N^e *that may be live in N is* $C(k)= 2^{(k-1)}-1$.

Let $s=(-1\ 1^0\ x_3\ x_4 \ldots x_k)$ *correspond to Marking M such that there are unmarked siphons in only the top* r_1-r_2 *siphon in* N^{er}. *The total number of possible live markings under M is* 1^{k-j}.

The total number of nonreachable markings in N^e *that may be live in is N A(k)= k-1.*

$L'(k) = 2L+A(k)+C(k)=18\times 2^{k-2}+k-4.$

$\vartheta'(k) = R' - L'(k) = (k-2)\,2^k - (k-3)$.

$¥'(n) = 2\times 3^k - R' = 2\times 3^k - (2k+5)\,2^{(k-1)} + 1$

3 Computation of *Top-Left-k-net* system reachable forbidden and non-reachable states

Definition 7: A k-net system (Top-Left-k-net) is a subclass of S^3PR with k resource places $r_1, r_2, \ldots, r_k$ shared between two processes N_1, N_2 ...and N_u and one non-sharing resource place r'_{gen} (=r) used by an operation place p^* in P_1*

1. *$M_0(r'_{gen}) = 1$ and $\forall r \in P_R$, $M_0(r) = 1$.*
2. *N_1 (resp. N_2...and N_u) uses $r_1, r_2, \ldots, r_k$ (resp. $r_k, r_{k-1}, \ldots r_2, r_1$) in that order.*
3. *$M_0(p^0_1)=k+1$, $M_0(p^0_i)=k$, $i > 1$, where p^0_1 and p^0_i are the idle places in processes N_1 and N_i, respectively.*
4. *Holder places of r_j in N_1 and N_i are denoted as p_j and p'_j respectively.*
5. *The compound circuit containing $r_i, r_{i+1}, \ldots, r_{j-1}, r_j$ is called $(r_i$-$r_j)$-region.*
6. *If r'_{gen} does not exist, then it is called a k-net system.*
7. *x^y means r_{gen+1} is at x state (x=1,0,-1) and r* is at y state (y= 0,-1), where gen is the location of non-sharing resource being used by an operation place p^*. The system is denoted as Top-Left-k-net system when gen=1 and p* in P_1.*

In *k*-net and *Top Top-Left-k-net,* let y_i^j deote the *i*-th token state at Process *j* (>1). $y_i^j=-1$ means the *i*-th token is at operation place p_i of Process *j*, and not at operation place p_i of other processes. Hence, $y_i^2 + y_i^3 + \ldots + y_i^{\mu} = y_i = -1$ and there are (μ-1) possibilities; i.e., exactly one of $y_i^2, y_i^3, \ldots, y_i^{\mu}$ equals -1; the rest are 0. $y_i^j=0$ means the *i*-th token is at resource place r_i. Thus, $y_i \leq 0$.

Chao [20] has constructed the formula of L_k and $Ŗ_k$, for *k*-net in theorem 2, and *3*, as extracted respectively blow:

Theorem 2 [20]*: For a k-net with μ processes, the total number of live states is* $L_k = 2^k + (\mu)^k - 1$ [20].

Theorem 3 [20]*: For a k-net with μ processes, the total number of reachable states is* $R(k) = 2^k + (\mu-1)y(1-x^k)/(1-x)$, where $x = \mu/2$ and $y = 2^{(k-1)}$ [20].

Here we extend to construct the formula of $Ł'_k$, and $Ŗ'_k$ for top k-net based on above results. The presence of the non-sharing resource place increases the number of states by a factor of 2. By formula (1) and (2), we can extend to $Ł'_k = 2L_k + A'(k) + C'(k)$. where *A'(k) and C'(k) are defined below*:

Theorem 4]: For *a k-net with μ processes,*

1. *The total number of forbidden markings that may be live, $C'(k) = (\mu)^{k-1} - 1$.*
2. *The total number of nonreachable markings that may be live, $A'(k) = (\mu-1)(k-1)$.*

Theorem 5: For a k-net with μ processes the total number of live markings $Ł'_k = 2L_k + (\mu-1)(k-1) + (\mu)^{k-1} - 1$.

Theorem 6: For a k-net with μ processes the total number of reachable markings $Ŗ'_k = 2Ŗ_k + ((\mu)^{k-1} - 1)$.

Examples: Table 3 lists results when *k=4* , μ=3 and *k=4,* μ=4 .These results have been validated experimentally.

Table 3.

k	4	4
μ	3	4
	k –net	
R	146	376
L	96	271
F	50	105
D	14	30
Total states	256	625
	Top-Left-k-net	
C'	6	9
A'	26	64
R'	318	815
Ł'	224	614
F'	94	201
Total states	512	1250

4 Conclusions

We report the very first method to compute in closed form the number of reachable states of *Top-Left-k-net* system without constructing a reachability graph. This helps to estimate the percentage of deadlocks and legal-state losses due to the addition of a monitor, and avoid the dire situation of mid-run abortion of reachability analysis due to exhausted memory. The formal result is important even if specific since manufacturing systems correspond generally to higher order (k much larger than 3) S^3PR systems, which can model concurrent programs where a locked data item can be represented by a single resource place with one token [27]. Current tools may not be able to handle such high order S^3PR systems due to the state explosion problem.

5 References

[1] Chao, D. Y., "Reachability of Nonsynchronized Choice Petri Nets and Its Applications," *IEEE Trans. Syst., Man, Cybern. A.*, vol. 35, no. 6, 2005, pp. 1203-1213.

[2] J. Ezpeleta, J. M. Colom, and J. Martinez, “A Petri net based deadlock prevention policy for flexible manufacturing systems,” *IEEE Trans. Robot. Automat.*, vol. 11, pp. 173–184, Apr. 1995.

[3] Armin Zimmermann, http://www.tu-ilmenau.de/ TimeNeT, Technische Universitat Ilmenau, System and Software Engineering, D-98684 Ilmenau, Germany.

[4] Daniel Yuh Chao, "Computation of elementary siphons in Petri nets for deadlock control," *Comp. J.*, (British Computer Society), vol. 49, no. 4, pp. 470–479, 2006.

[5] Shih, Y.Y. and D.Y. Chao, "Sequence of control in S^3PMR," *Comp. J.*, doi:10.1093/comjnl/bxp081.

[6] Uzam M and Zhou, MengChu, "An improved iterative synthesis approach for liveness enforcing supervisors of flexible manufacturing systems," *Int J Prod Res.*, vol. 44, no. 10, pp. 1987-2030, 2006.

[7] Chao, D.Y., "Improvement of Suboptimal Siphon- and FBM-Based Control Model of a Well-Known S^3PR," *IEEE Transactions on Automation Science and Engineering*, vol. 8, no. 2, pp. 404-411, 2011.

[8] D.Y. Chao, "Enumeration of Lost States of A Suboptimal Control Model of A Well-Known S3PR," *IET Control Theory Appl.,* 2011, vol. 5, no. 11, pp. 1277–1286.

[9] J. S. Lee, M. C. Zhou, and P. L. Hsu, "An application of Petri nets to supervisory control for human-computer interactive systems," *IEEE Transactions on Industrial Electronics*, vol. 52, no. 5, pp. 1220-1226, Oct. 2005.

[10] Mizuno, N., Ohta, A., and Tsuji, K., "Reachability Problem of Marked Graphs with Batch Processing Arcs," Industrial Electronics Society, 2007. *IECON 2007*. 33rd *Annual Conference of the IEEE* , pp. 70-75.

[11] Miyamoto, T. and Horiguchi, K. "Modular reachability analysis in fundamental class of multi-agent nets ," *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society*, pp. 3782-3787.

[12] Lee, D. I., Kumagai, S. and Kodama, S. "Reachability of LSFC nets," IEEE International Symposium on Circuits and Systems, vol. 4, pp. 2666 - 2669, May 1990.

[13] Ferrarini, L., "On the reachability and reversibility problems in a class of Petri nets," *IEEE Tran on Systems, Man and Cybernetics*, vol. 24, no. 10, pp. 1474 - 1482.

[14] K. Hiraishi and A. Ichikawa, "A Class of Petri Nets that a Necessary and Sufficient Condition for Reachability is Obtainable," *Trans. of SICE*, vo1. 24, no. 6, 1988, pp.635-640 (in Japanese).

[15] A. Ichikawa, K. Yokoyama, and S. Kurogi, "Control of event-driven systems: reachability and control of conflict-free Petri nets," *Trans. of SICE*, vol. 21, no.4, 1985, pp.324-330.

[16] Kostin, A. E., "Reachability analysis in T-invariant-less Petri nets," *IEEE Tran on Automatic Control*, vol. 48, no. 6, pp. 1019-1024.

[17] P. H. Starke, *INA: Integrated Net Analyzer*, 1992. Handbuch, Germany.

[18] Hong Liang and Daniel Yuh Chao, "Enumeration of Reachable States for Arbitrary Marked Graphs," *IET Control Theory and Applications*, vol. 6, no. 10, pp. 1536 – 1543, 2012, 07.

[19] Chao, Daniel Yuh, "Recursive Solution of Number of Reachable States of A simple subclass of FMS," *International Journal of Systems Science*, Jan. 2013, DOI: 10.1080/00207721.2012.745240.

[20] Daniel Yuh Chao, Jiun-Ting Chen, Fang Yu, "Enumeration of Reachable and Other States of Simple Version of Systems of Simple Sequential Processes with Resources (S3PR)," Industrial Electronics Society, 2012 ISIE 2012. The 21th IEEE International Symposium on Industrial Electronics.

[21] Daniel Yuh Chao, Tsung Hsien Yu, "Enumeration of Reachable (forbidden, live, and deadlock) States of Top k-th Order System (with a non-sharing resource place) of Petri Nets," *IECON 2013 - 39th Annual Conference on IEEE Industrial Electronics Society*, pp. 3515-3521.

[22] Daniel Yuh Chao, Tsung Hsien Yu, Sou Chein Wu, "Closed Form Formula Construction to Enumerate Control Related States of K-th Order S3PR System (with a Top Left side non-sharing resource place) of Petri Nets," IEEE Systems, Man, and Cybernetics Society, 2014, *ICNSC14, The 11th IEEE International Conference on Networking, Sensing and Control*.

[23] Daniel Yuh Chao, Tsung Hsien Yu, "Enumeration Of Reachable, Forbidden, Live And Deadlock States Of K-Th Order System (With A Bottom Non-Sharing Resource Place) Of Petri Nets," *ATISR 2013. The International Conference on Applied and Theoretical Information Systems Research*.

[24] Daniel Yuh Chao, Tsung Hsien Yu, Chia Chang Liou, "Enumeration of Reachable, forbidden, live, and deadlock States of Bottom k-th Order System (with a left side non-sharing resource place) of Petri Nets," IEEE Systems, Man, and Cybernetics Society, 2014, *ICNSC14, The 11th IEEE International Conference on Networking, Sensing and Control*.

[25] Daniel Yuh Chao, Tsung Hsien Yu, Tsung Fu Lin "Closed Form Formula Construction to Enumerate Control Related States of Middle-Left K-th Order S3PR System of Petri Nets," The American Council on Science and Education, *2014, The 2014 International Conference on Computational Science and Computational Intelligence (CSCI '14)*.

[26] Nazeem, A., Reveliotis, S.A., Wang, Y., and Lafortune, S.: "Designing Compact and Maximally Permissive Deadlock Avoidance Policies for Complex Resource Allocation Systems Through Classification Theory: The Linear Case," *IEEE Trans. Automat. Contr.* 56, (8), 2011, pp. 1818-1833

Efficient Anonymization of the SocioNet with the Aid of Rumor Riding

Hiroki IIZUKA[1] **and Satoshi FUJITA**[1]
[1]Department of Information Engineering, Hiroshima University
Higashi-Hiroshima, 739-8527, Japan

Abstract— *In this paper, we propose an anonymized object search scheme for the SocioNet which is an unstructured P2P based on the notion of the similarity of interests. The proposed scheme is an application of a randomized object search scheme proposed by Liu et al. called Rumor Riding (RR, for short). We propose two techniques to overcome the inefficiency of a simple application of the RR to the SocioNet. The performance of the proposed scheme is evaluated by simulation. The simulation result indicates that the proposed scheme reduces the number of messages to a half of a simple combination, and additionally, shows that the number of delegates selected in the RR severely affects the success rate of the overall scheme particularly when the TTL is not large.*

Keywords: Peer-to-Peer content sharing, anonymity of users, object search, Rumor Riding, SocioNet.

1. Introduction

Recent advancement of network technologies enables us to easily share various contents over the Internet. For example, YouTube attracts more than 1 billion unique user visits per month and the upload of 100 hours of video every minute in 2014. A key issue to realize such a content sharing over a large network is *how to find the location of a requested object*. In particular, the support of an efficient object search is a crucial issue for Peer-to-Peer (P2P) applications since in those systems, objects are generally stored in the local storage of each peer without being collected to a specific server as in classical content sharing services.

Flooding of queries with a designated TTL (time to live) is a simple but commonly used technique to realize an efficient object search in P2P networks. There are many proposals concerned with the variations of the query flooding, which includes LightFlood [3], Diff-Flooding [2] and UMPS [10]. Among them, we are interested in the object search based on an unstructured overlay reflecting the interest of the users. SocioNet [4] and UIM [1] are representatives of such approaches. The key idea of such similarity-based overlays is to connect peers to have similar interests by a link so that the peer which issues a query, called **questioner**, can be connected with a peer which has an object matching the query, called **respondent**, through a path consisting of a small number of links. By adopting such an overlay, the efficiency of query flooding can be significantly improved compared with random overlays [4]. However, although it certainly improves the efficiency, it causes a serious risk for each user so that the fact of issuing a query, the fact of responding to the query and the content of the query and the reply are disclosed to all peers to have similar interests. In other words, such a simple flooding could not preserve the privacy of users which is a crucial drawback of the most of existing flooding-based object search schemes.

In this paper, we focus on the SocioNet as the underlying similarity-based P2P, and propose a scheme to preserve the anonymity of users in the network. The proposed scheme is an application of a randomized method proposed by Liu *et al.* called Rumor Riding (RR, for short) [5]. The key idea of the RR is to select delegates through random walk and to make those delegates to conduct actual query flooding and the response to the query (see Section 3 for the details). It is evaluated by simulation that such a randomized approach could certainly preserve the anonymity of users while keeping the cost reasonably low. However, a direct application of the RR to the SocioNet is not efficient since the RR was originally proposed for random overlays and the application of the RR loses the benefit of the SocioNet so that the distance between the questioner and the respondent is short. To overcome such an issue, this paper proposes two techniques to improve the efficiency of the object search in the SocioNet in terms of the number of messages which is necessary to keep a high success rate.

The performance of the proposed scheme is evaluated by simulation. The result of simulation indicates that it reduces the number of messages to a half of a simple combination of the RR and the SocioNet, and additionally, it shows that the number of delegates severely affects the success rate when the given TTL is not large. More precisely, we found that the number of delegates, which can be controlled by tuning parameters used in the RR, should be at least three to attain a high success rate while keeping the number of messages sufficiently low.

The remainder of this paper is organized as follows. Sections 2 and 3 describe an overview of the SocioNet and the basic flow of the RR, respectively. Section 4 describes the proposed scheme. Section 5 describes the simulation result. Finally, Section 6 concludes the paper with future work.

2. SocioNet

2.1 Overview

The SocioNet is an unstructured P2P based on the notion of *similarity of interests*. Each link in the SocioNet is either a similarity link or a random link. The former is intended to connect peers to have similar interests so that a query issued by a peer easily hits a target object with high probability *which is expected to be held by a peer to have similar interest to the questioner*, and the latter is intended to connect a pair of remote peers so that the resulting network has a short diameter. Random links are established by *rewiring* similarity links with a certain probability β through random walk, as in the Watts and Strogatz's scheme to construct small-world networks [9] (the value of parameter β is set to around 0.2 to 0.3 in the SocioNet).

The search of a target object is done through the flooding of a query as in conventional P2Ps, while the existence of similarity links could significantly reduce the number of message transmissions required for attaining a given hit rate compared with random overlays such as Gnutella [8].

2.2 Similarity of Peers

The similarity of peers is defined as follows. Let O_i be the set of objects held by peer i. Assume that each object is attached **tags** representing the attributes of the object, e.g., a music file of the performance of Benny Goodman will be attached tags Jazz, Clarinet and Swing. Let $\mathcal{T} = \{t_1, t_2, \ldots, t_j, \ldots\}$ denote the (universal) set of tags. For each peer i and tag $t_j \in \mathcal{T}$, let O_{i,t_j} denote the set of objects attached tag t_j in set O_i. Then, the relevance of tag t_j with peer i is defined as

$$w_{i,t_j} \stackrel{\text{def}}{=} \frac{|O_{i,t_j}|}{|O_i|}.$$

For example, if peer i has 100 objects and 50 of them are attached tag Jazz, then $w_{i,\text{Jazz}} = \frac{50}{100} = 0.5$. The profile of peer i, denoted by $\vec{w_i}$, is a vector of relevances, i.e.,

$$\vec{w_i} \stackrel{\text{def}}{=} (w_{i,t_1}, w_{i,t_2}, \ldots, w_{i,t_j}, \ldots).$$

With the above notions, the similarity of peer j for peer i is defined as follows

$$sim(i,j) \stackrel{\text{def}}{=} \frac{|O_i|}{|O_j|} \times \frac{1}{cos(\vec{w_i}, \vec{w_j})}, \quad (1)$$

where peer j with a smaller $sim(i,j)$ is more favorable for peer i as an adjacent peer connected by a similarity link. The reader should note that the above notion of similarity is not symmetrical. If fact, even if two peers a and b have the same profile $\vec{w} = \vec{w_a} = \vec{w_b}$, when $|O_a| < |O_b|$, we have

$$sim(a,b) < 1 < sim(b,a),$$

that is, b would be favorable for a but the reserve is not true.

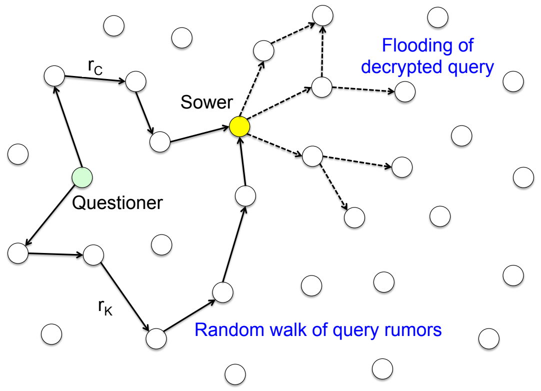


Fig. 1: Steps 1 and 2 in the Rumor Riding.

2.3 Update Procedure

With the above notions, the SocioNet establishes similarity links in two different ways. The first way is to use a server which keeps the similarity for all pairs of peers to select pairs to have high similarity in a centralized manner. The second way, which will be adopted in the proposed scheme, is to use random walk. More concretely, each peer i which wishes to update its similarity links first conducts x independent random walks, where x is the (maximum) degree of the peer in the overlay. At any peer in the random walk, it stops with probability $c/\log N$ for some constant c so that the expected length becomes $\mathcal{O}(\log N)$, where N is the number of peers in the network, and the peer at the stopped point is regarded as the candidate for new neighbors. Among those x candidates and the currently adjacent x peers, peer i selects x peers to have highest similarity to peer i, and updates neighbors so that it is connected to the selected x peers.

3. Rumor Riding

Rumor Riding (RR) is a scheme to realize an anonymous object search in unstructured P2Ps. The basic idea of the RR is to delegate the roles of the *flooding of a query* and the *reply to the query* to randomly selected peers called sowers. With such a randomized mechanism, we can keep the anonymity of the questioner and the respondent. In addition, to keep the security of message transmissions, each message is encrypted by the sender of the message using the public key of the receiver.

The protocol for the object search in RR consists of five steps. In the following, we explain each step in detail.

Step 1: Generation of Query Rumors

Let i be the questioner. At first, peer i generates a public key K_i^+ and inserts it to the content of the query, where K_i^+ will be used to encrypt the reply to the query by the respondent. Let q be the plain text of the resulting message including K_i^+. Peer i then encrypts q with a symmetric key K into a cipher text C, then organizes two **query rumors** r_K

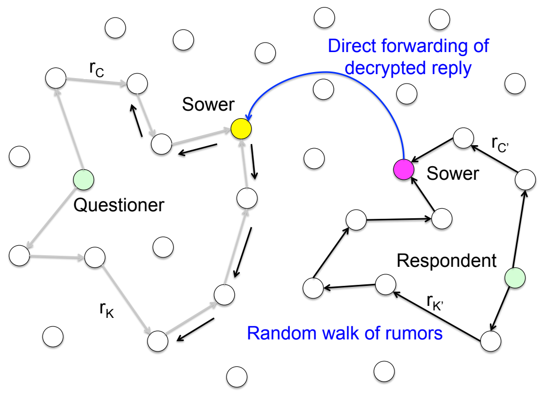


Fig. 2: Step 3 in the Rumor Riding.

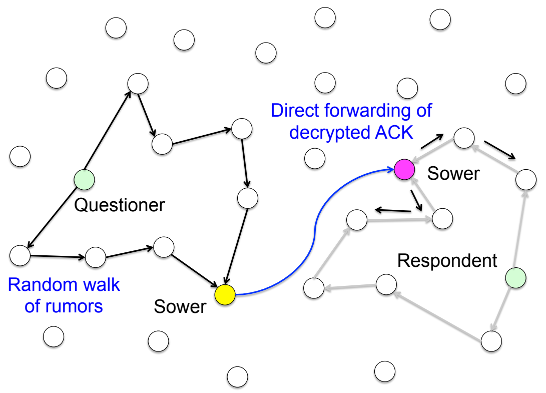


Fig. 3: Step 4 in the Rumor Riding.

and r_C, where r_K and r_C are messages containing K and C, respectively. Those rumors are sent out to different neighbors of peer i and start an (independent) random walk with an appropriate TTL (more precisely, peer i generates k such pairs of rumors to increase the probability of those rumors "meeting" at a peer, where k is an appropriate parameter; it is experimentally verified that k and the TTL should be determined so that their product is from 100 to 200, i.e., if k is four then the TTL should be from 25 to 50 [5]). See Figure 1 for illustration.

Step 2: Sowers Concerned with the Questioner

In the RR, a peer which receives both r_K and r_C serves as a delegate of the questioner called **sower**. More concretely, after decrypting message q from K and C, each sower starts the flooding of q to its neighbors and waits for the reply to the query from an appropriate respondent. After receiving a reply message from the respondent, which is encrypted with the public key K_i^+ of the questioner i and is separated into two rumors similar to the separation of q into r_K and r_C, it sends back those rumors to the questioner along the paths traveled by r_K and r_C, respectively, in the reverse direction. The reader should note that to enable such a behavior of the sower and the other intermediate peers, the RR should force every peer to cache all rumors passing through the peer for a certain time so that it is expired after the reply message is successively received by the questioner.

Step 3: Reply from the Respondent

Suppose that query q transmitted by a sower s is received by a peer j holding an object matching the query. After receiving q, peer j generates a reply message and encrypts it with the public key K_i^+ of the questioner i. Let R be the resulting cipher text. Peer j then encrypts R and the IP address of s with a symmetry key K' into a cipher text C', then organizes two **reply rumors** $r_{K'}$ and $r_{C'}$ similar to Step 1. Those rumors are sent out to different neighbors and start an (independent) random walk, as before. If a peer receives both $r_{K'}$ and $r_{C'}$ from its neighbors, then the peer serves as the sower concerned with the respondent as follows: 1) it decrypts R and the IP address of s from reply rumors, and 2) it directly forwards reply rumors to sower s. See Figure 2 for illustration.

Step 4: ACK Message

After receiving reply rumors $r_{K'}$ and $r_{C'}$, the questioner i decrypts R from C' with symmetry key K' and then decrypts the reply message from R with the secret key of peer i. Then peer i sends an ACK message to the respondent j in the following manner: 1) it encrypts the ACK message into a cipher text with the public key of j (which should be contained in the reply message); 2) it organizes two rumors from the cipher text as in previous steps; and 3) it sends out those rumors to different neighbors, as before. The sower conceded with the ACK message directly forwards the received rumors to the sower concerned with the reply message described in Step 3, which will be delivered to the respondent j by traveling the path used in the random walk in the reverse direction. See Figure 3 for illustration.

Step 5: Transmission of Object

After receiving the (encrypted) ACK message, the respondent j decrypts it into plain text with the symmetry key contained in a rumor and the secret key of j. After that, it moves to the actual transmission of the requested object using digital envelope. More concretely, after encrypting the object into a cipher text F, peer j transfers it to the questioner through random walk of two rumors, direct forwarding of the rumors to the sower concerned with the ACK message, and the delivery of rumors by traveling the path used in the random walk of Step 4 in the reverse direction.

4. Proposed Method

4.1 Design Issues

This section describes the details of the proposed scheme. The goal of the scheme is to realize an anonymous object

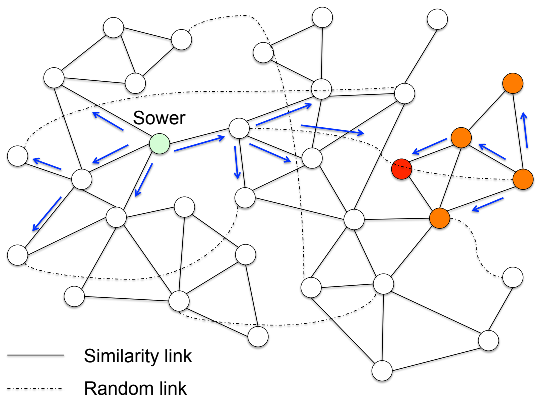


Fig. 4: Dynamic switch of the mode of flooding during the query propagation.

search in the SocioNet using the notion of the RR described in Section 3. However, if we directly apply the techniques used in the RR to the SocioNet, we will face to the following issues: 1) As was described in Section 2, the SocioNet is designed in such a way that the questioner is located in the neighborhood of the respondent. However, the direct application of the RR to the SocioNet loses such a benefit of the SocioNet, since in the RR, the actual flooding is conducted by a sower which is randomly selected from all peers in the network, i.e., we cannot guarantee that the sower is in the neighborhood of the respondent. 2) The search in the RR is based on a simple flooding, i.e., it repeats the forwarding of a received query to *all* neighbors until the TTL given to the query exhausts. However, such a simple scheme does not fully utilize the structure of the SocioNet so that two types of links play different roles in the overlay, i.e., random link connects remote peers and similarity link connects peers to have similar interests. This means that to improve the efficiency of the object search, the propagation of a query from the selected sower should be conducted by carefully considering the difference of the role of links.

In the following subsections, we propose two techniques to overcome those issues.

4.2 Dynamic Switch of the Mode of Flooding

The first technique is to take into account the difference of the role of links during the propagation of query messages. More concretely, we devolve the role of *diversification* to random links in an early phase of the query propagation and the role of *intensification* to similarity links in the remaining steps of the query propagation.

The concrete operation proceeds as follows. Let s be a sower concerned with the questioner which received two query rumors r_K and r_C from its different neighbors. After decrypting message q from C with K, s starts the flooding of q to its neighbors by setting TTL to a *small* value, e.g., two to five, using both of random and similarity links. Each copy of the query stops the propagation when: 1) the TTL exhausts or 2) it arrives at a peer holding an object matching the query. In addition, if it arrives at a peer which has a *similar interest to the query*, then it switches the mode of flooding so that it merely uses similarity links to realize an efficient intensification of the exploration.

The similarity of a peer j with a query q is calculated as follows. Recall that in the SocioNet, each peer is associated with a profile representing its interests in the form of a vector of relevances to the tags in $\mathcal{T}$. The idea is to associate a set of tags to each query issued by the questioners[1]. If query q is associated with a single tag t drawn from set $\mathcal{T}$, the similarity of the query with a peer j is calculated in the following three steps: 1) extract the relevance $w_{j,t}$ of j with tag t from the profile $\vec{w_j}$; 2) extract top α elements from the profile with the maximum relevance; and 3) if $w_{j,t}$ is contained in the extracted α elements, then we judge that the similarity between peer j and query q is high. If q is associated with two or more tags, we extend the above scheme so as to check whether the majority of tags associated with the query are contained in the top α elements in the profile vector.

Figure 4 illustrates a running example of the scheme. In this figure, the peer holding an object matching the query issued by the questioner is painted red, and peers which has a similar interest with the query is painted orange. After decrypting the query from two query rumors received from different neighbors, the sower, which is painted green, initiates a flooding of the query by setting the TTL to a small value. The flooded message uses all links within the TTL, and after arriving at an orange peer, which has a similar interest to the query, it switches the mode to the flooding without random links.

4.3 Similarity-Based Filtering

The second technique is to filter queries at each similarity link by the similarity of the receiver to the query. Suppose that peer j receives a query q associated with a set of tags. In the first technique, *all* similarity edges outgoing from j are used for the propagation of the query unless the TTL is exhausted. However, since the similarity of peers is defined by the cosign similarity of profiles and the number of objects held by each peer (see Equation (1) for the details), a neighbor ℓ of j connected by a similarity edge (j, ℓ) might *not* be relevant to q even if peer j is relevant to q and the value of $sim(j, \ell)$ is small. For example, consider the case in which peers j and ℓ have 200 objects attached tag Jazz, peer j has 20 objects attached tag Clarinet and peer ℓ has

[1]The simplest way to realize such a situation is to ask questioners to designate tags associated with the query. Another possible way is to adopt the technique of automatic tag attachment which has been proposed in the literature [7]. In the evaluation described in Section 5, we assume that each query is attached a single tag by the questioner.

Table 1: Parameters used in the simulation.

The number of peers	10000
The number of objects	1000
The number of peers holding matching object	100
Average degree of peers	6
Rewiring probability	0.3
TTL of the first phase	2
Threshold θ	0.8

no object attached tag Clarinet. In such a case, a query q with tag Clarinet received by peer j should not be forwarded to peer ℓ, since ℓ has no object attached tag Clarinet and such a fact can be detected by analyzing the relevance of the receiver ℓ to the query.

The filtering of queries is conducted by using the cosign similarity. More concretely, each query q is associated with a binary vector $\vec{q}$ so that the i^{th} element in the vector takes value 1 if and only if the i^{th} tag (in set $\mathcal{T}$) is associated with q. Then, the similarity $\sigma(q, \ell)$ between peer ℓ and query q is calculated as $\sigma(q, j) = \cos(\vec{q}, \vec{w_\ell})$, and the similarity link connecting to ℓ stops the forwarding of q if the value of $\sigma(q, j)$ is smaller than a predetermined threshold θ.

5. Evaluation

5.1 Setup

We evaluate the performance of the proposed scheme by simulation. The simulation is conducted by using PeerSim simulator [6], and as the competitor, we use a simple combination of the RR and the SocioNet in which each sower concerned with the questioner initiates a flooding of the decrypted query with a designated TTL. In the following, we denote the above combined scheme as COMB and the proposed scheme with two techniques PROP, where for the reader's reference, we also show the result for the scheme merely with the first technique denoted as TECH1 and that with the second technique denoted as TECH2. The metric for the evaluation is the number of messages and the success rate, which are averaged over 30 runs.

Parameters used in the simulation are given as follows. The number of peers and the number of objects are fixed to 10000 and 1000, respectively, where each object can have several copies in the overlay. The number of copies held by each peer follows a Poisson distribution with mean $\lambda = 6$. The popularity of the object matching a query is set to 1%, i.e., we consider a situation in which among 10000 peers, only 100 peers hold the object matching the query. The overlay network consisting of similarity links is generated by the Barbási-Albert (BA) model so that the average degree of each peer is six and the probability of rewiring a similarity link into a random link is set to 0.3. TTL of the first phase of the query forwarding used in the first technique is set to two. Finally, we fix threshold θ used in the second technique to 0.8. Those parameters as summarized in Table 1.

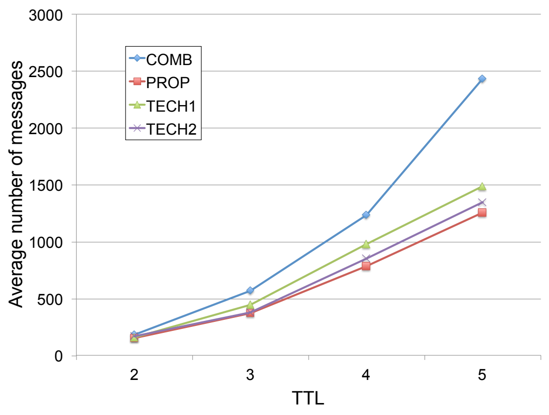


Fig. 5: The average number of messages issued in four schemes.

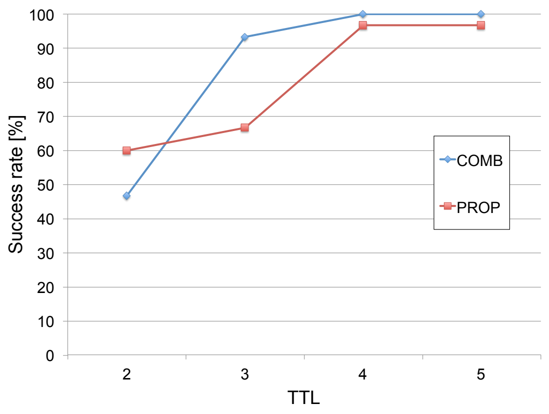


Fig. 6: The success rate of two schemes obtained by dividing the number of successful runs by the total number of runs.

5.2 Number of Messages

Figure 5 illustrates the result on the number of messages. The horizontal axis is the TTL of the flooding and four curves correspond to the result for COMB, PROP, TECH1 and TECH2, respectively. Although there is no big difference among four schemes when TTL is two, we could find a significant reduction of the number of messages as the TTL becomes large. In particular, the amount of improvement of COMB by PROP is about 50% when TTL is five.

5.3 Success Rate

Figure 6 compares the success rate of COMB and PROP, which is calculated by dividing the number of successful runs by the total number of runs in the simulation, where the horizontal axis is the TTL of query flooding, as before. The success rate of COMB monotonically grows as the TTL increases, which reaches 100% when TTL is four. However, the success rate of PROP is not stable with respect to the monotonic change of the TTL; e.g., the success rate when

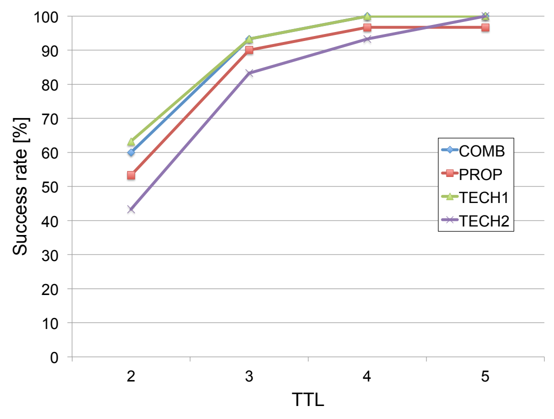


Fig. 7: Comparison of the success rate of four schemes which is calculated by excluding runs with two or less sowers.

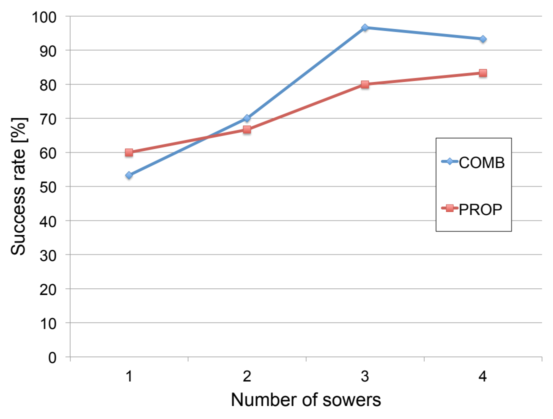


Fig. 8: Impact of the number of sowers to the success rate (TTL is fixed to three).

TTL is three seems to be too small compared with the success rate for other TTLs.

A reason of such an instability of the success rate is due to the small number of sowers generated by the RR. See Figure 7 for illustration. This figure redraws the curves of the success rate after excluding simulation runs in which the number of sowers generated by the random walks is two or less. As shown in the figure, by excluding such runs, we have a reasonable grow of the success rate, and can make the following observations: 1) the use of the second technique reduces the success rate (recall that the second technique stops the forwarding of the query to a peer to have a profile which is not similar to the query); and 2) COMB is better than TECH2, i.e., the simultaneous use of the first technique with the second technique relaxes the badness of the second scheme. The conjecture such that the number of sowers affects the success rate is confirmed by Figure 8, which illustrates the impact of the number of sowers to the success rate by fixing the TTL to three.

6. Concluding Remarks

This paper proposes an anonymized object search scheme for the SocioNet. More precisely, we propose two techniques to overcome the inefficiency of a simple application of the Rumor Riding to the SocioNet, where the first technique is to dynamically switch the kind of links used for the query propagation and the second technique is to filter queries at each similarity link by the similarity of the receiver to the query. The performance of the scheme is evaluated by simulation. The simulation result indicates that the proposed scheme reduces the number of messages of a simple combination of the SocioNet and the Rumor Riding to a half without significantly reducing the success rate.

A future work is to verify the effect of the popularity of the searched object to the performance, which was fixed to 1% in the current simulation. Another key issue is to conduct a detailed analysis of the behavior of the proposed scheme, since in the current work, we merely evaluate the average number of messages and the success rate.

References

[1] G. Chen, C.-P. Low, and Z.-H. Yang. “Enhancing Search Performance in Unstructured P2P Networks Based on Users’ Common Interest.” *IEEE Trans. on Parallel and Distributed Systems*, 19(6): 821–836, 2008.

[2] Y.-D. Gong, J. Hu, Z.-J. Dong, S.-N. Wang, and S.-J. Hu. “Improved Flooding-Based Resource Discovery.” In *Proc. of the 2nd International Workshop on Intelligent Systems and Applications (ISA)*, 2010, pages 1–4.

[3] S. Jiang, L. Guo, and X.-D. Zhang. “LightFlood: an efficient flooding scheme for file search in unstructured peer-to-peer systems.” In *Proc. of International Conference on Parallel Processing*, 2003, pages 627–635.

[4] K. C.-J. Lin, C.-P. Wang, C.-F. Chou, and L. Golubchik. “SocioNet: A Social-Based Multimedia Access System for Unstructured P2P Networks.” *IEEE Trans. on Parallel and Distributed Systems*, 21(7): 1027–1041, 2010.

[5] Y.-H. Liu, J.-S. Han, J.-L. Wang. “Rumor Riding: Anonymizing Unstructured Peer-to-Peer Systems.” *IEEE Trans. on Parallel and Distributed Systems*, 22(3): 464–475, 2011.

[6] A. Montresor and M. Jelasity. “PeerSim: A scalable P2P simulator.” In *Proc. of the 9th International Conference on Peer-to-Peer Computing (P2P’09)*, 2009, pages 99–100.

[7] T.-T. Qin and S. Fujita. “Automatic Tag Attachment Scheme for Efficient File Search in Peer-To-Peer File Sharing Systems.” In *Proc. International Conference on Advances in Social Network Analysis and Mining (ASONAM 2011)*, 2011, pages 507–511.

[8] Y. Wang, X.-C. Yun, and Y.-F. Li. “Analyzing the Characteristics of Gnutella Overlays.” In *Proc. of the 4th International Conference on Information Technology*, 2007, pages 1095–1100.

[9] D. J. Watts and S. H. Strogatz. “Collective dynamics of ’small-world’ networks.” *Nature*, 393(6684): 440–442, 1998.

[10] A. Wu, X.-S. Liu, and K.-J. Liu. “Efficient flooding in peer-to-peer networks.” In *Proc. of the 7th International Conference on Computer-Aided Industrial Design and Conceptual Design (CAIDCD ’06)*, 2006, pages 1–6.

Parallelization of an Analytical Method for the Evaluation of Voltage Sags in Electric Power Systems

A. Antunez, A. Ramos, E. Espinosa and C. Ceja
Faculty of Electrical Engineering
Universidad Michoacana de San Nicolás de Hidalgo
Morelia, México

Abstract - *In this paper, parallel processing was applied to the parallelization of an analytical method for estimation of voltage sags in electric power systems. The parallelization of this analytical method is made by using multi-thread programming (POSIX threads). POSIX is a standard for UNIX-like operating systems which specifies an application programming interface for multithreaded programming. The algorithm was written in C programming language. All tests were performed using the GNU/Linux operating system. The proposed parallelized analytical method was applied to the 57-bus and 118-bus IEEE test systems, in order to demonstrate its proper operation. A comparative analysis of the parallelized method with respect to the traditional method is presented, and the reduction in computational time is shown.*

Keywords: voltage sags; analytical method; parallel processing; multi-threading.

1 Introduction

In the first years of the 1980s, the term *power quality* started to gain importance when analyzing the performance of an electric system, due to the faults that occurred in electric equipment associated with voltage disturbances. This demonstrated that power systems weren't as reliable as they were thought to be, giving an even bigger relevance to aspects of power quality, in an electric industry where fulfilling the quality demands of clients has become a priority [1].

Voltage sags which are defined as a reduction in RMS voltage with durations of half a cycle to one minute, caused by short-circuits, overloads or starting of large motors are one of the power quality disturbances which affect clients most quickly [2]. During a voltage sag, the voltage magnitude is not equal to zero, however, it is significantly lower than the voltage level under normal operation conditions, which makes voltage sags one of the main causes of undesired equipment trip [3] [4].

Some methods for the stochastic estimation of voltage sags systems have been proposed, such as the presented in [3]-[6]. Voltage sags in a specific location of the system are calculated by means of sag probability density functions on transmission lines, or using elements of the bus impedance matrix and their relation with a fictitious bus, whose position changes along the transmission lines.

The analytical method proposed by E. Espinosa-Juárez and A. Hernández [7] (henceforth abbreviated as AMEH) has significant advantages with respect to some of the referred methods in [3]-[6]; one of these advantages, for example, is the possibility of application in case of unbalanced faults, as well as the traditional case of balanced faults [6], and also the accuracy achieved with these method.

The AMEH requires analyzing all the system lines in order to calculate the voltage sags for each bus of the electrical network; this means that applying this method to large networks significantly increases the number of required calculations.

On the other hand, in modern electrical system analysis, it has become necessary to use efficient computing techniques, such as parallel processing, amongst others [8][9]. Parallel processing is defined as a type of data processing in which two or more processing elements perform calculations in order to solve a problem simultaneously [10][11].

In the area of the electrical system parallel processing has been applied to analyze and solve several problems, as well as in the solution of several industrial applications [12]-[16]. Parallel processing has produced a significant reduction in the time necessary to perform studies and consequently, makes it possible to carry out a power system analysis in less time

By using computational techniques, such as parallel programming, a much more efficient implementation of the AMEH method can be achieved.

In this paper, the AMEH analytical method for stochastic voltage sag estimation in electric power systems is implemented using parallel processing techniques based on multi-thread schemes. The implemented method is applied to analyze two IEEE test systems in order to show its functionality.

2 Parallelized AMEH Analytical Method

In order to clearly illustrate the parallelization of the AMEH method, a brief description of the method in [7] is presented, followed by a detailed explanation of how the parallelization is achieved.

2.1 Fundaments of the AMEH Analytical Method

Consider an electrical system represented by the following nodal formulation [7]:

$$V_{bus} = Z_{bus} I_{bus} \tag{1}$$

where V_{bus} is the bus voltage vector
Z_{bus} is the bus impedance matrix
I_{bus} is the bus current vector

When a fault occurs at bus *i*, the fault current at said bus is calculated as:

$$I_i = \frac{V_i^0}{Z_{ii}} \tag{2}$$

where: I_i is the fault current at bus *i*
V_i^0 is the pre-fault voltage at bus *i*
Z_{ii} is the *ii* element of the bus impedance matrix

The voltage at a specific bus of the system, for example, bus *m*, when a fault at bus *i* occurs, is calculated using the following expression:

$$V_m = V_m^0 - Z_{mi} I_i \tag{3}$$

where: V_m is the voltage at bus *m* when a fault occurs at bus *i*
V_m^0 is the pre-fault voltage at bus *m*
Z_{mi} is the *mi* element of the bus impedance matrix

Assuming a pre-fault voltage of 1 pu,

$$V_m = 1 - \frac{Z_{mi}}{Z_{ii}} \tag{4}$$

This means that a residual voltage magnitude *Vm* is present at bus *m* when a fault occurs at bus *i*.
On the other hand, voltage at bus *m* is affected by faults that can occur at any point of the system, faults can appear at buses and any position along the transmission lines. Each bus and each line have an associated value of faults per year (λ), which is obtained from statistics.
Taking this into account, equation (4) is generalized in order to being able to determine the voltage at bus *m* when the fault occurs at any part of a transmission line [7]. Consider a transmission line of the analyzed system, which connects buses *k* and *j*, as shown in Fig. 1.

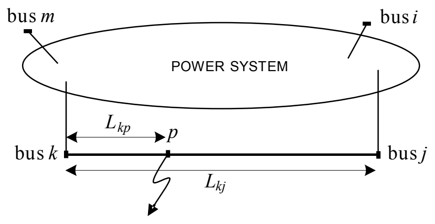


Fig.1. Transmission line connecting buses *k* and *j*.

Then, when a fault occurs at *p*, the voltage at bus *m* will be [7]:

$$V_m = 1 - \frac{(1-\psi)Z_{mk} + \psi Z_{mj}}{(1-\psi)^2 Z_{kk} + \psi^2 Z_{jj} + 2\psi(1-\psi)Z_{kj} + \psi(1-\psi)z_{kj}} \tag{5}$$

where:
$Z_{mk}, Z_{mj}, Z_{kk}, Z_{jj}, Z_{kj}$ are elements of the bus impedance matrix
z_{kj} is the transmission line impedance
In (5) ψ is the variable that defines the fault position:

$$\psi = \frac{L_{kp}}{L_{kj}} \tag{6}$$

where:
L_{kp} is the distance between buses *k* and *p*
L_{kj} is the distance between buses *k* and *j*
The value of ψ varies from 0 to 1.
The probability of fault occurrence in the specified position ψ between ψ_{low} and ψ_{up} corresponding to the remaining voltage between V_{low} and V_{up} varies a value of ψ varies is given by [7]:

$$P(\psi_{low} \le \psi \le \psi_{up}) = \int_{\psi_{low}}^{\psi_{up}} g(\psi)\,d\psi \tag{7}$$

where:
$P(\psi_{low} \le \psi \le \psi_{up})$ is the probability of $\psi_{low} \le \psi \le \psi_{up}$ and $g(\psi)$ is the probability distribution function of faults along the considered line.
The number of voltage sags at bus *m* caused by faults at line *k-j* can be calculated as [7]:

$$N_q^a(V_{low} \le |V_m^a| \le V_{up}) = \psi_q \int_{\psi_{low}}^{\psi_{up}} g(\psi)\,d\psi \tag{8}$$

Therefore, if uniform distribution of faults along the line is considered, the total number of voltage sags at bus *m* is obtained by considering the voltage sags calculated in each line of the system [7]:

$$N_m(V_{low} \le |V_m^a| \le V_{up}) = \sum_q N_q^a(V_{low} \le |V_m^a| \le V_{up}) \tag{9}$$

2.2 Parallelization of AMEH method

The AMEH analytical method has a very good level of precision in voltage sag estimation. However, it requires to analyze all the system lines in order to calculate the voltage sags for each bus of the electrical network, this means that applying this method to large networks significantly increases the complexity of the problem and performing the corresponding calculations.

For example, for the IEEE 118-bus test system [17], which has 177 transmission lines, fault calculations must be performed for each line in order to evaluate the voltage sags at each bus in the system, which translate into a total of 20,886 calculations of voltage remaining curves, each with its respective operations.

In the designed algorithm for the parallelization of the AMEH method, a fault distribution function associated to transmission lines is defined. Then, the impedance matrix is calculated and equation (7) is evaluated considering values in the range $0 \leq \psi \leq 1$, in order to obtain the curve of remaining voltage corresponding to the specified node. This result helps determine whether the voltage has a monotonically increasing or decreasing behavior (or neither), which, in turn, allows to define the integration limits for equation (8), according to the desired voltage ranges. Afterwards, the number of voltage sages is calculated with equation (8), and finally, using equation (9), the total number of voltage sags at bus *m*, originated by faults at every line of the system, is obtained.

There are different programming environments which allow implementing parallel processing schemes. A computer with a multicore CPU is capable of performing operations in parallel, dividing the total amount of work among all the available cores. However, a control scheme is necessary in order to avoid overlapping between the cores. If all the memory locations of the system can be accessed by all cores, it is possible for a specific variable, which has been modified by several cores, to contain an incorrect or unpredictable value, producing unexpected results in a program. A simple solution to this problem is specifying which memory locations are shared between the cores, that is, which parts of the memory can be accessed by all cores. A program which uses this particular type of memory allocation is generally known as a shared-memory program.

In this work, the analytical method previously described will be optimized using POSIX threads. POSIX is a standard for UNIX-like operating systems which specifies an application programming interface for multithreaded programming. [18]

POSIX threads or, more commonly, Pthreads, is not a programming language, but rather a library which can be linked to programs written in C language in order to implement shared-memory schemes.

As the name indicates, Pthreads is based on the use of threads, which can be considered as individual processes capable of running simultaneously. Each thread has private stacks and program counters, but access to determined variables may be common to certain threads.

The main purpose of applying parallel computing to the analytic method is to significantly reduce the time required to analyze large electric networks.

When using traditional or sequential programming to apply the analytical method, the calculation of voltage sags must be performed one transmission line at a time. This process becomes especially long if a large network is to be analyzed.

On the contrary, when applying parallel computing the problem can be divided into smaller parts, that is, each line is analyzed by an individual processing element, running in parallel. Considering this, the ideal case would be when the number of lines in the system equals the number of available threads. Under these conditions, the execution time reduction would be maximal.

However, an electrical system is usually formed by far more lines than the total number of threads that can be launched in parallel in current multicore computers. In this case, a certain number of lines are assigned to each thread, in such a way that the total amount of calculations is divided as evenly as possible. If, for example, a small system consists of 10 lines, and the computer used to perform the analysis has 4 threads available, two threads would be used to calculate voltage sags along 3 lines each, and the remaining two would be used to calculate voltage sags along 2 lines each, covering the totality of calculations required with a load division as even as possible.

The efficiency of the parallel algorithm is measured in terms of the time it takes to complete the calculations with one processing element in comparison to the time it takes to complete the calculations with *p* processing elements; this relation is known as *Speed-Up* (S) [19].

$$S = \frac{T_1}{T_p} \tag{12}$$

where:

T_1 is the execution time with one processing element;

T_p is the execution time with *p* processing elements.

Using this metric of performance ensures that the reduction of execution time is independent of the computer characteristics.

3 Case studies

Two case studies are presented. The results obtained verify the correct performance and improved efficiency of the implemented method using multi-thread programming.

3.1 Studies in the IEEE 57-bus test system

The IEEE 57-bus test system (Fig. 2) consists of 63 lines, 15 transformers and 7 generators [20]. Three-phase balanced faults with unitary fault ranges are considered, and bus fault ranges are neglected.

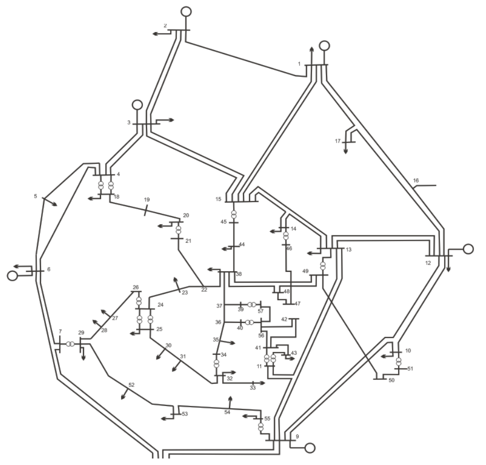


Fig. 2. IEEE 57-bus test system.

Fig. 3, Fig. 4 and Fig. 5 show the results of sags per year considering ranges from 0.6 to 0.7 p.u., 0.7 to 0.8 p.u. and 0.8 to 0.9 pu, respectively, for faults at lines, using λ=1. Fig. 6 shows the difference between the sags per year values of ranges from 0 to 0.7 p.u., 0 to 0.8 p.u. and 0 to 0.9 p.u. In Fig. 7 the total sags per year are shown, that is, the sum of sags per year at lines and the sags per year at nodes, along with the sags per year with a range from 0 to 0.9 pu.

In Table 1 the computing times for the 57-bus system are shown, and it can be observed that as the number of threads increases, the execution time decreases proportionally. However, when using more than 3 threads the results do not show any further improvement. This is due to the small dimensions of the system. A larger system would have to be analyzed in order to perceive a significant time reduction.

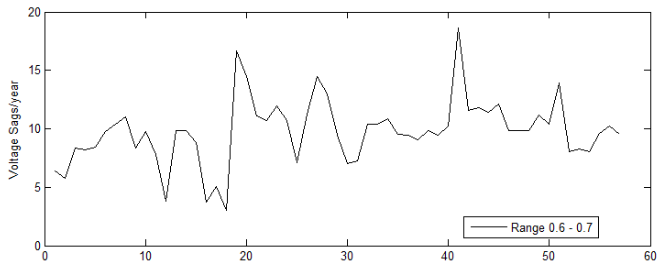


Fig.3. Voltage sags/year considering a voltage sags range from 0.6 to 0.7 p.u.

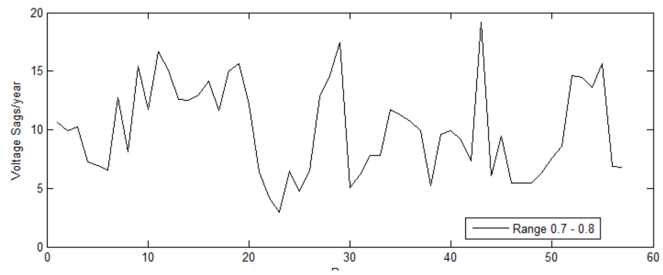


Fig. 4. Voltage sags/year considering a voltage sags range from 0.7 to 0.8 p.u.

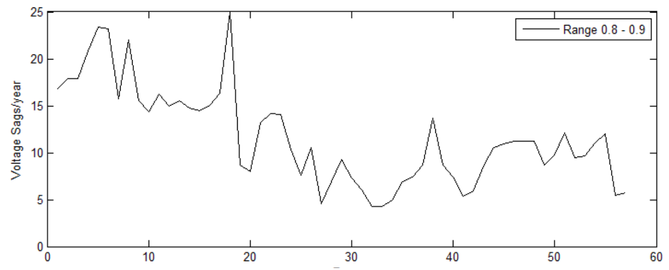


Fig. 5. Voltage sags/year considering a voltage range from 0.8 to 0.9 p.u.

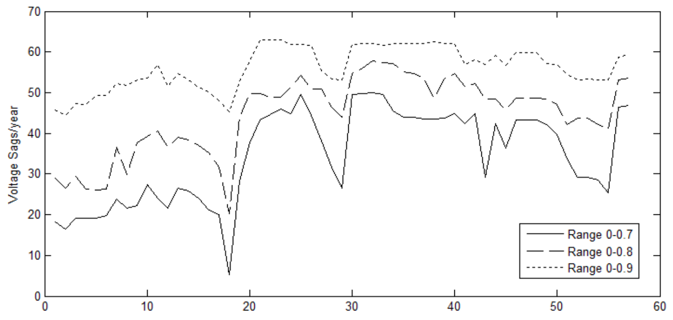


Fig. 6. Voltage sags/year considering voltage ranges from 0 to 0.7, 0 to 0.8 and 0 to 0.9 p.u.

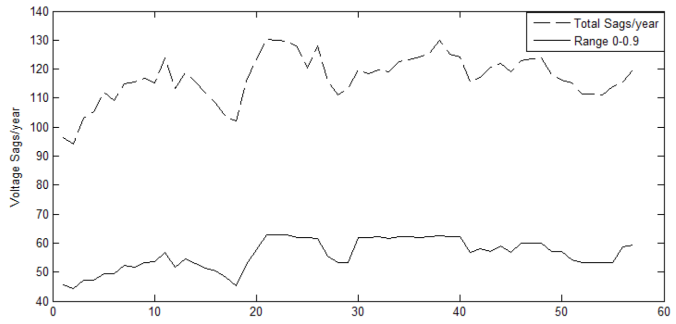


Fig. 7. Voltage sags/year at lines plus voltage sags/year at nodes, considering a voltage range from 0 to 0.9 p.u.

TABLE I. RESULTS OF THE IEEE 57-BUS TEST SYSTEM, SHOWING THE SPEED UP OBTAINED USING DIFFERENT NUMBER OF THREADS

Threads	Time (s)	Speed Up
1	0.024	1.00
2	0.019	1.26
3	0.017	1.41
4	0.017	1.41

3.2 Studies in the IEEE-118 bus test system

The IEEE 118-bus test system (Fig. 8) consists of 177 lines, 9 transformers and 33 generators [17]. Three-phase balanced faults with unitary fault ranges are considered, and bus fault ranges are neglected.

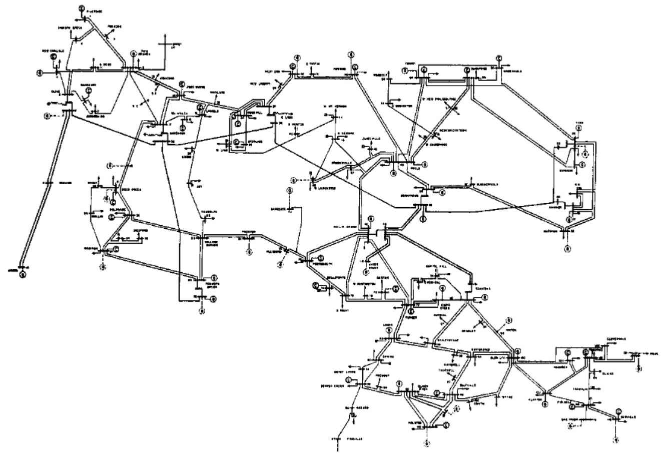


Fig. 8. IEEE 118-bus test system.

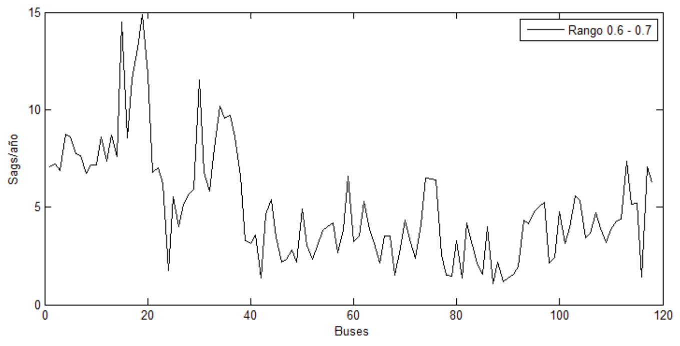


Fig. 9. Voltage sags/year considering a voltage sags range from 0.6 to 0.7 p.u.

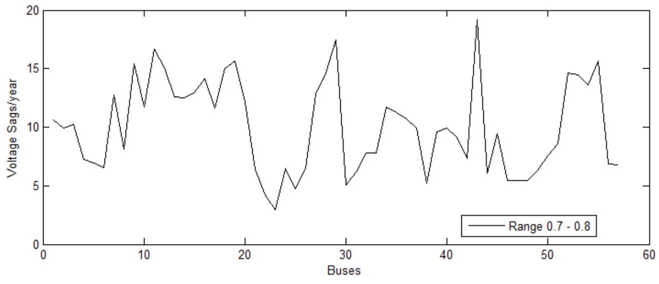


Fig. 10. Voltage sags/year considering a voltage sags range from 0.7 to 0.8 p.u.

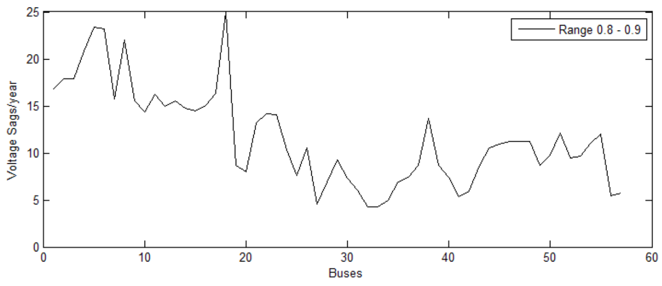


Fig. 11. Voltage sags/year considering a voltage range from 0.8 to 0.9 p.u.

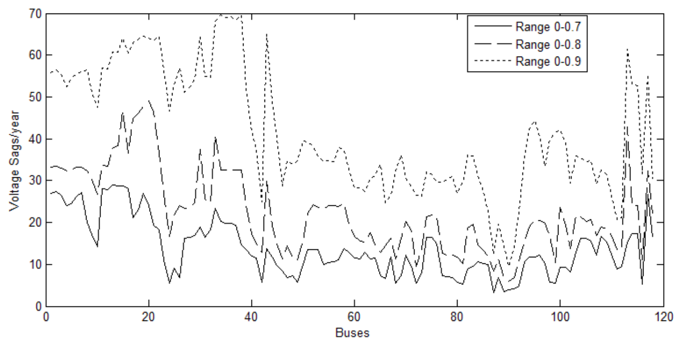


Fig. 12. Voltage sags/year considering voltage ranges from 0 to 0.7, 0 to 0.8 and 0 to 0.9 p.u.

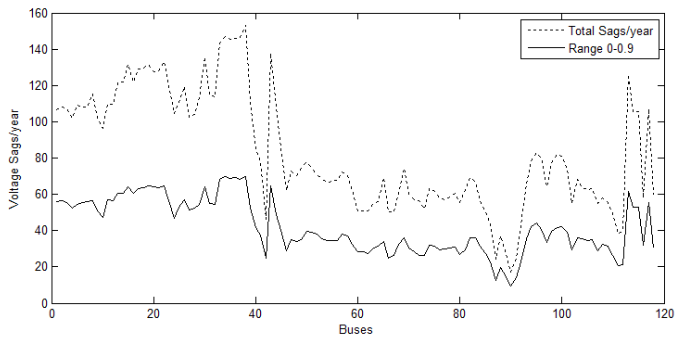


Fig. 13. Voltage sags/year at lines plus voltage sags/year at nodes, considering a voltage range from 0 to 0.9 p.u.

Fig. 9, Fig. 10 and Fig. 11 show the results of sags per year considering ranges from 0.6 to 0.7 p.u., 0.7 to 0.8 p.u and 0.8 to 0.9 p.u., respectively, for faults at lines, using λ=1. Fig. 12 shows the difference between the sags per year values of ranges from 0 to 0.7 p.u., 0 to 0.8 p.u. and 0 to 0.9 p.u. In Fig. 13 the total sags per year are shown, that is, the sum of sags per year at lines and the sags per year at nodes, along with the sags per year with a range from 0 to 0.9 pu.

In Table II the computing times for the IEEE 118-bus test system are shown, and it can be observed that the execution time decreases as the number of processing elements (threads) used increases.

TABLE II. RESULTS OF THE IEEE 118-BUS TEST SYSTEM, SHOWING THE SPEED UP OBTAINED USING DIFFERENT NUMBER OF THREADS

Threads	Time (s)	Speed Up
1	0.171	1.00
2	0.136	1.26
3	0.125	1.37
4	0.12	1.43
5	0.118	1.45

4 Conclusions

In this paper, parallel processing was applied to the parallelization of the AMEH analytical method for estimation of voltage sags in electric power systems. The parallelization was done using a multi-thread scheme, which allows to divide the work of performing all the required calculations for faults at lines and buses of the analyzed system.

The proposed parallelized analytical method was applied to the IEEE 57-bus and IEEE 118-bus test systems. Based on the results obtained, it has been demonstrated that the parallelized method represents a highly useful tool in the analysis of voltage sags, which greatly improves the efficiency of the sequential AMEH analytical method.

References

[1] J. Mason, R. Targosz, “European power quality survey report”, Leonardo Energy Initiative, Nov. 2008. Available in: http://www.leonardo-energy.org/ european-power-quality-survey-report

[2] IEEE Recommended Practice for Monitoring Electric Power Quality, IEEE Std. 1159-1995, Nov. 2, 1195

[3] M. H. J. Bollen, “Understanding power quality problems. Voltage sags and interruptions”, IEEE PRESS Series on Power Engineering, 2000.

[4] Y. S. Lim, G. Strbac, “Analytical approach to probabilistic prediction of voltage sags on transmission networks”, IEE Proc. Generation, Transmission and Distribution, vol. 149, no. 1, pp. 7-14, Jan. 2002.

[5] S. Quaia, F. Tosato, "A method for analytical voltage sags prediction", IEEE Bologna Power Tech 2003, Jun. 2003.

[6] M. H. J. Bollen, “Method of critical distances for stochastic assessment of voltage sags”, IEEE Proc. Generation, Transmission and Distribution, vol. 145, no. 1, pp. 70-76, Jan. 1998.

[7] E. Espinosa-Juárez, A. Hernández., “An analytical approach for stochastic assessment of balanced and unbalanced voltage sags in large systems”, IEEE Trans. Power Delivery, vol. 21, no. 3, pp. 1493-1500, Jul. 2006.

[8] A. F. Vojdani, “Smart Integration”, IEEE Power and Energy Magazine, IEEE, vol. 6, Issue 6, pp. 71-79.

[9] United States Department of Energy, “2010 Smart Grid System Report”, U. S. Department of Energy, February 2012.

[10] G.S. Stavrakakis, C. Lefas, A. Pouliezos, "Parallel processing computer implementation of a real time DC motor drive fault detection algorithm", IEE Proceedings Electric Power Applications, Vol. 137, No. 5 September 1990, pp 309-313.

[11] K. Werler, H. Glavitsch, "Computation of transients by parallel processing", IEEE Transactions on Power Delivery, Vol. 8, No. 3, July 1993.

[12] G.S. Stavrakakis, C. Lefas, A. Pouliezos, "Parallel processing computer implementation of a real time DC motor drive fault detection algorithm", IEE Proceedings Electric Power Applications, Vol. 137, No. 5 September 1990, pp 309-313.

[13] K. Werler, H. Glavitsch, "Computation of transients by parallel processing", IEEE Transactions on Power Delivery, Vol. 8, No. 3, July 1993.

[14] Z.A. Mariños, J.L.R. Pereira, Jr. Carneiro, "Fast harmonic power flow calculation using parallel processing", IEE Proc.-Gener. Transm. Distrib., Vol. 141, No. 1 January 1994, pp 27-32.

[15] N. Garcia, A. Medina, "Swift time domain solution of electric systems including SVSs". IEEE Transactions on Power Delivery, Vol. 18, No. 3, July 2003, pp 921-927.

[16] O.A. Rico-Hernández, A. Ramos-Paz, "Analysis of electrical networks using fine-grained techniques of parallel processing based on OpenMP", Proceedings of the 2011 8th International Conference on Electrical Engineering Computing Science and Automatic Control (CCE).

[17] R. Christie, "Power Flow Test Cases, 57 Bus Power Flow Test Case", University of Washington, College of Engineering, Electrical Engineering, May 1993. Available: http://www.ee.washington.edu/research/pstca/.

[18] Peter S. Pacheco, “An introduction to parallell programming”, Ed. Elsevier Morgan Kaufmann, 2011

[19] I. Foster, "Designing and building parallel programs", Addison Wesley, 1994.

[20] R. Christie, "Power Flow Test Cases, 57 Bus Power Flow Test Case", University of Washington, College of Engineering, Electrical Engineering, May 1993. Available: http://www.ee.washington.edu/research/pstca/.

The Solution of a FEM Equation in Frequency Domain Using a Parallel Computing with CUBLAS

R. Dominguez[1], **A. Medina**[1], **and A. Ramos-Paz**[1]
[1]Facultad de Ingeniería Eléctrica, División de Estudios de Posgrado, U.M.S.N.H., Ciudad Universitaria, C.P. 58030, Morelia, Michoacán, MEXICO.

Abstract - *The recent technological computer advances have allowed the use of the Finite Element Method (FEM), to calculate the solution of the Maxwell field equations of electrical machines or devices. In some cases, an axisymmetric or a plane symmetry can be assumed to reduce the complexity of the finite element analysis to be performed. Nevertheless, the large size of the matrix equations derived, could imply a significant computing effort. In this paper, a parallel method of solution in frequency domain of a FEM equation with currents known is proposed. It consists on implementing the LU method using a parallel computing with CUBLAS. A normal and a reduced type of FEM equation proposed by the authors have been solved in the frequency domain using this parallel computing platform. It is shown that a significant reduction in the computing time to solve these FEM equations in the frequency domain is achieved.*

Keywords: Finite element method, frequency domain analysis, parallel processing

1 Introduction

The Finite Element Method (FEM) is a very powerful tool to solve the electric and magnetic equations of electrical machines or devices. The method has been widely used, since the computational technological advances have allowed the application of the method on the modeling and simulation of electrical machines or devices with complex geometries of configurations [1]-[3].

Nevertheless, the method can be difficult to use in devices with 3D geometries or in those which need a detailed geometry model; the reason is the large matrix equations derived by the finite element analysis, which in turn can be difficult to solve in the frequency domain or in the time domain. However, the finite element analysis can be simplified if a planar or axisymmetric assumption is taking into account [2], [3].

In an earlier paper, the authors proposed a new form to solve a FEM equation with currents or voltages known [4]. The method consists on deriving a lesser order equation from a normal FEM equation. The *reduced* equivalent equation obtained is expressed in terms of the time varying variables, and it can be easily solved in time domain or in the frequency domain [3]. The *reduced* equation can be calculated from a *normal* FEM equation derived from of a finite element analysis performed on a device with a planar or an axisymmetric symmetry [4]. The *reduced* equation is easy to derive and solve, since it implies the use of simple matrix operations [4]. These matrix operations can be derived by a parallel computing. Moreover, the *normal* and the *reduced* FEM equations can be solved in the frequency domain by a parallel solution. Thus, it is possible to obtain a significant computation time reduction. The FEM equations to be solved correspond to equations that model a device with a planar or axisymmetric symmetry, and whose conductor currents are known.

In this paper, the *LU* method has been implemented in the *CUBLAS* parallel platform, in order to solve *normal* and *reduced* FEM equations in the frequency domain. Specifically, an *LU* decomposition process was implemented using parallel processing using routines of the *CUBLAS* library. The proposed parallel solution has been tested in two devices: a planar conductor and a series reactor with an axisymmetric symmetry assumption.

The rest of the paper is organized as follows: Section 2 explains the features of the partial differential equations of devices modelled by planar or the axisymmetric symmetries. Section 3 explains the features of the FEM matrix equations, derived from a finite element analysis performed with the partial differential equations shown in Section 2. Section 4 explains how the *normal* and the *reduced* FEM matrix are solved in the frequency domain; Section 5 describes how these equations are solved using the CUBLAS computing platform; Section 6 describes a case study which consists of two devices in which the parallel solution has been tested: the first device is a “T” conductor modelled by a planar symmetry and the second device is an air series reactor modelled by an axisymmetric symmetry. Finally, Section 7 contains the main conclusion drawn from this investigation.

2 Partial Differential Equations of a Device with Planar or Axisymmetric Symmetries

This investigation is based on the following assumptions: the frequency of the voltage source of the device to be modelled is low enough to neglect the displacement current in the Maxwell field equations [2], [3], [5]. The permeability and the conductivity of the device are assumed to be constant. Finally, there are no voltage difference at different conductor points [5].

In some cases, the modelling of a device can be simplified by a planar or axisymmetric symmetries [2], [3]. If it is considered that a skin effect exists on the conductors of the devices, and that these conductors are excited by voltage sources, then the partial differential equations for a device with a planar or an axisymmetric symmetries are given by [5]. [6],

$$\frac{\partial}{\partial x}\left[v\left(\frac{\partial A_z}{\partial x}\right)\right]-\frac{\partial}{\partial y}\left[v\left(\frac{\partial A_z}{\partial y}\right)\right]+\sigma\frac{\partial A_z}{\partial t}=\frac{\sigma\{U_c\}}{l} \tag{1}$$

$$-\frac{v}{r}\frac{\partial}{\partial r}\left[r\left(\frac{\partial A_\phi}{\partial r}\right)\right]-\frac{v}{r}\frac{\partial}{\partial z}\left[r\left(\frac{\partial A_\phi}{\partial z}\right)\right]+\frac{v}{r^2}A_\phi+\sigma\frac{\partial A_\phi}{\partial t}=\sigma\frac{\{U_c\}}{2\pi r} \tag{2}$$

Where A_z and A_ϕ are the magnetic vector potential of a device with a planar or an axisymmetric symmetry assumption, respectively; σ and v are the conductivity and reluctivity of the materials, respectively. $\{U_c\}$ is a vector which contains the voltages applied at the conductors of the device. If it is considered that the voltages along the z-axis are constant for a planar symmetry; and that the voltages along the ϕ-axis are constant for an axisymmetric symmetry; then it is possible to derive an equation to relate the voltage, current and the magnetic vector potentials at the conductors of the device [5], [6]. The equation is given by [5], [6],

$$[\varDelta_x]^{-1}\{U_c\}-\iint_{S_c}\sigma\frac{\partial A}{\partial t}dS=\{I\} \tag{3}$$

Where $\{I\}$ is a vector that contains the conductors' current. The matrix $[\varDelta_x]$ for the planar and the axisymmetric symmetry is defined by the equations (4) and (5), respectively.

$$[\varDelta_x]\ =\left[\frac{2\pi}{\sigma}\left(\iint_{S_c}\frac{dS}{r}\right)^{-1}\right]_{diag} \tag{4}$$

s

$$[\varDelta_x]\ =[R_c]=\left[\frac{l}{\sigma}\left(\iint_{S_c}dS\right)^{-1}\right]_{diag} \tag{5}$$

Where $[R_c]$ is the conductor matrix resistance if the device has a planar symmetry assumption. The surface area S_c of the equations (4) and (5) varies if the device is modeled by a planar or an axisymmetric symmetry. For the case of the planar symmetry, the surface area S_c involves the plane x-y [5]. For an axisymmetric symmetry, the surface area involves the plane r-z [6].

3 Finite Element Analysis of the Device

It is possible to perform a finite element analysis on the partial differential equations defined on (1) and (2). At the same time, a Newton Cotes analysis can be performed on the expression defined in (3). It yields [5], [6],

$$[S]\{A_x\}+[T]\frac{d\{A_x\}}{dt}=\{f\}\{U_c\} \tag{6}$$

$$[\varDelta_x]^{-1}\{U_c\}-[M_c]\frac{d\{A_x\}}{dt}=\{I\} \tag{7}$$

Where the matrices $[S]$, $[T]$, $[M_c]$ and the vector $\{f\}$ are obtained from the finite element analysis performed for a planar or axisymmetric symmetry [5], [6]. The vector $\{I\}$ contains the currents in the conductors of the device, A_x is defined in the z-axis and the ϕ-axis for the planar and the axisymmetric symmetry, respectively.

If the conductor currents in $\{I\}$ are known, it is possible to calculate the magnetic vector potentials $\{A_x\}$ and the conductor voltages $\{U_c\}$. This can be achieved by coupling the equations (6) and (7) in a unique equation that can be easily solved in the frequency domain [3], [7]. It gives,

$$\left(\begin{bmatrix}[S] & -\{f\}\\ 0 & [\varDelta_x]^{-1}\end{bmatrix}+j(2\pi f)\begin{bmatrix}[T] & 0\\ -[M_c] & 0\end{bmatrix}\right)\begin{Bmatrix}\{\tilde{A}_x\}\\ \{\tilde{U}_c\}\end{Bmatrix}=\begin{Bmatrix}0\\ \{\tilde{I}\}\end{Bmatrix} \tag{8}$$

Where the vector of magnetic potentials $\{\tilde{A}_x\}$, the conductor voltages $\{\tilde{U}_c\}$ and the conductor currents $\{\tilde{I}\}$ are all harmonic variables defined for frequency f. The equation (8) can be represented as,

$$([K]+j(2\pi f)[G])\{\tilde{X}\}=\{\tilde{f}\} \tag{9}$$

Moreover, (9) can be represented in a simpler way, i.e.

$$[A]\{\tilde{X}\}=\{\tilde{b}\} \tag{10}$$

The equation (10) is a *normal* FEM matrix expression. It is possible to derive a simpler equation from (10) [4]. This *reduced* equation allows to express (10) in terms of its time varying variables, e.g. the vector of magnetic potentials of the conductors [4]. The equation is of lesser order than (9) and can be also solved in the frequency domain. The *reduced* equation can be represented by,

$$[A_T]\{\tilde{X}_T\}=\{\tilde{b}_T\} \tag{11}$$

The equations shown in (10) and (11) have a *preprocessing* step, where their matrices are formed by a finite element analysis, and by a *calculating* step in which their solution in the frequency domain is derived. These stages will be discussed next.

4 Solution of the Normal and the Reduced FEM Equations in the Frequency Domain

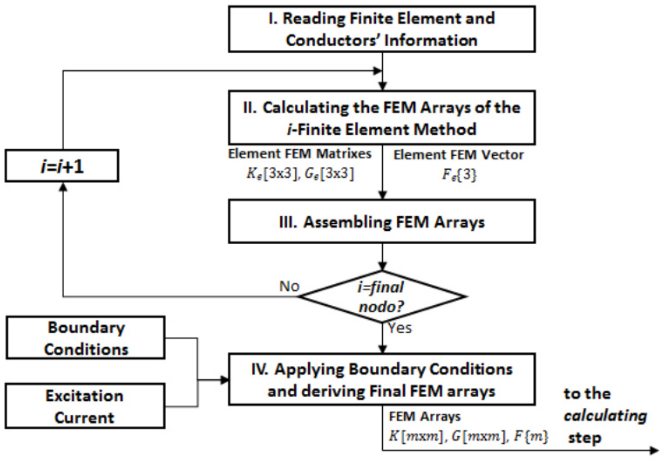
The FEM matrix equation to be solved are the *normal* (10) and the *reduced* types (11). For both equations can be recognize two specific steps in the process of calculating their solution in the frequency domain, i.e. a *preprocessing* and a *calculating* steps, respectively. These stages will be explained next.

4.1 Preprocessing Step of the FEM Equations

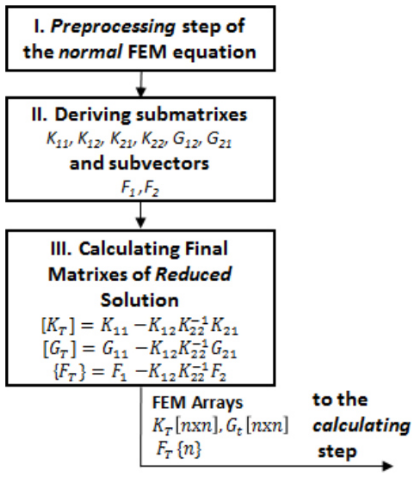
The *preprocessing* step of the *normal* FEM method consists on deriving the final matrices $[K]$ and $[G]$ and the vector $\{f\}$ of the (10). The process consists on first calculating the FEM matrices and vectors of one finite element, integrate them into the global matrices and vectors that model the device [2], [3] and apply the required boundary conditions.

The *preprocessing* step of the *reduced* FEM method consists on deriving sub-matrices and sub-vectors from the final matrices and vectors obtained from the preprocessing step of the *normal* FEM equation, in order to calculate matrices of lesser order [4]. These FEM matrices permit to formulate a FEM equation of lesser order, which allows to directly solve the time varying variables of the device. The *preprocessing* step of a *normal* and a *reduced* FEM equations can be seen in Fig. 1.

Fig. 1 *Preprocessing* steps of the FEM equations


I. Reading Finite Element and Conductors' Information
II. Calculating the FEM Arrays of the i-Finite Element Method
Element FEM Matrixes $K_e[3x3]$, $G_e[3x3]$
Element FEM Vector $F_e\{3\}$
i=i+1
III. Assembling FEM Arrays
No
i=final nodo?
Yes
Boundary Conditions
Excitation Current
IV. Applying Boundary Conditions and deriving Final FEM arrays
FEM Arrays $K[mxm]$, $G[mxm]$, $F\{m\}$
to the calculating step

a) Preprocessing step of the *normal* FEM equation


I. Preprocessing step of the normal FEM equation
II. Deriving submatrixes $K_{11}, K_{12}, K_{21}, K_{22}, G_{12}, G_{21}$ and subvectors F_1, F_2
III. Calculating Final Matrixes of Reduced Solution
$[K_T] = K_{11} - K_{12}K_{22}^{-1}K_{21}$
$[G_T] = G_{11} - K_{12}K_{22}^{-1}G_{21}$
$\{F_T\} = F_1 - K_{12}K_{22}^{-1}F_2$
FEM Arrays $K_T[nxn]$, $G_t[nxn]$ $F_T\{n\}$
to the calculating step

b) Preprocessing step of the *reduced* FEM equation

4.2 Calculating Step of the FEM Equations

Once the matrices and vector of the *normal* and the *reduced* FEM equations are calculated, it is possible to derive their solution in the frequency domain. The *normal* and the *reduced* equations have the form of the expressions previously defined in (10) and (11), respectively.
It can be seen that these FEM equations have the form of the expression $[A_g]\{\tilde{x}_g\} = \{\tilde{b}_g\}$. This matrix equation can be solved by using the *LU* method.

The *calculating process* of the *normal* and the *reduced* FEM equations is performed using the *LU* method. Thus, the first step consists on performing a decomposition of the matrix $[A_g]$ into two matrices $[L_g]$ and $[U_g]$, respectively. It yields,

$$[A_g] = [L_g][U_g] \tag{12}$$

After having the matrices $[L_g]$ and $[U_g]$, the solution of $[A_g]\{x_g\}=\{b_g\}$ can be achieved by triangular decomposition *LU*; and the *normal* and *reduced* FEM equations can be solved. The difference between these equations is the *preprocessing* step and the order of the FEM matrix equation to be solved by the *calculating* step.

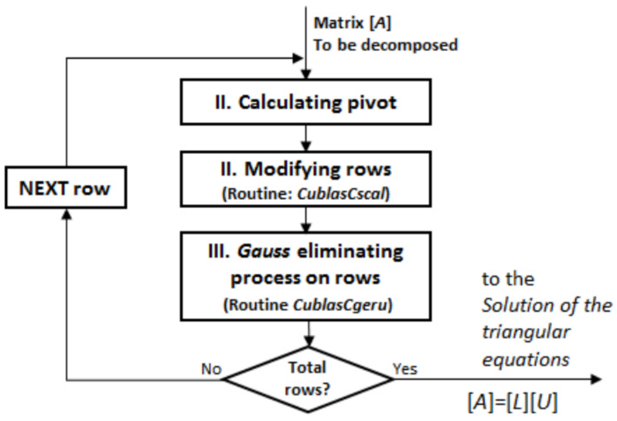
5 Calculating Process implemented by a Parallel Computing in CUBLAS

The *calculating process* for the *normal* and the *reduced* FEM matrix equations are implemented in the *CUBLAS* computing platform. Some steps of the *preprocessing* process of the *reduced* FEM equation can also be implemented by parallel computing. This will be explained next.

5.1 Decomposition LU implemented in CUBLAS

Once the complex matrix equation $[A]\{\tilde{x}\} = \{\tilde{b}\}$, that corresponds to the *normal* or the *reduced* FEM equation, has been formulated, the matrix $[A]$ will be decomposed into the product of matrices $[L]$ and $[U]$. This can be achieved by using the standard *LU* decomposition process. This process implies to calculate a pivot located in the main diagonal of $[A]$, performing a modification of the next rows and, finally, eliminating the rows using the Gauss eliminating process. The decomposition process was implemented by a parallel computing in *CUBLAS*. This process is shown in Fig. 2.

Fig. 2 Decomposition process implemented in *CUBLAS*



The *CUBLAS* routines used for the parallel computation ot the LU decomposition, correspond to matrices and vectors composed of single precission complex numbers [8]. Once the matrix $[A_g]$ is decomposed int the product of $[L_g]$ and $[U_g]$, the equation $[A_g]\{\tilde{x}_g\} = \{\tilde{b}_g\}$ can be easily solved. This will be explained next.

5.2 Final Solution achieved by CUBLAS

After having the matrixes $[L_g]$ and $[U_g]$, the solution $\{\tilde{x}_g\}$ can be calculated by solving the next equations in the *CUBLAS* computing platform,

$$[L_g]\{\tilde{y}_g\} = \{\tilde{b}_g\} \tag{13}$$

$$[U_g]\{\tilde{x}_g\} = \{\tilde{y}_g\} \tag{14}$$

Equation (13) is solved by using the routine *cublasCstrv*, and specifying that the equation to be solved corresponds to a triangular matrix stored in lower mode [8]; while (14) is also solved using the routine, but specifying that the equation to be solved corresponds to a triangular matrix stored in upper mode [8]. It can be seen that the solution of the complex equation $[A_g]\{\tilde{x}_g\} = \{\tilde{b}_g\}$ can be easily derived by implementing the *LU* method by a parallel computing in *CUBLAS*. The results and the performance of this method of solution were tested for the case study described next.

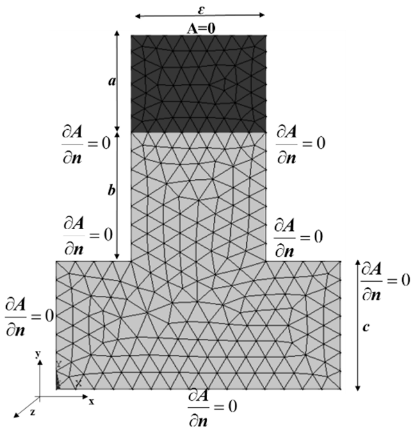
6 Case Study

It consists on analyzing in the frequency domain two devices modelled by a planar and the axisymmetric symmetry assumption. The first device to be analyzed is a "T" planar conductor. The second device is an air series reactor that can be modelled by an axisymmetric symmetry assumption. The finite element analysis to be performed on these devices involves the solution of the *normal* and the *reduced* FEM equations, which have the form of the expressions shown in (10) and (11), respectively. These equations will be solved in a sequential and a parallel computing platform.

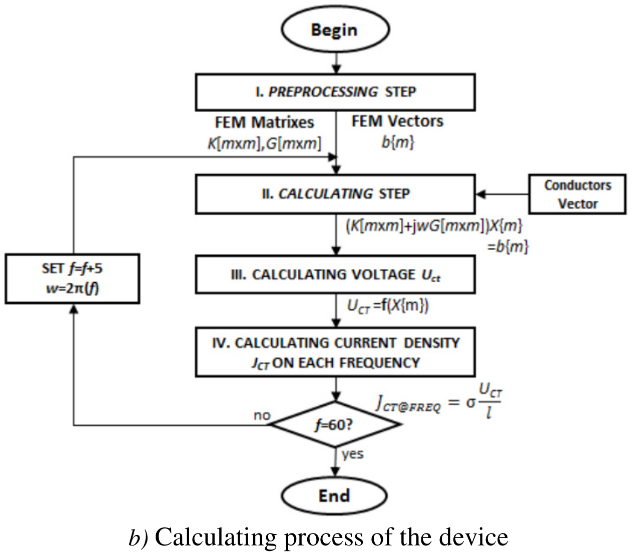
6.1 Device modelled by a Planar Symmetry Assumption

It consists on analyzing a "T" slot-embedded conductor with a copper conductor and an air region in a frequency range of 5Hz to 60Hz with a frequency step of 5Hz. The objective of the example is to analyze how the total source current density J_{ct} of the conductor varies in this frequency range [5]. The source density J_{ct} will be obtained via the *calculating* process shown in Fig. 3 (b). The FEM model and the geometry of the "T" conductor is shown in Fig. 3(a).

Fig. 3 Device with a planar symmetry assumption



a) Geometry and FEM model

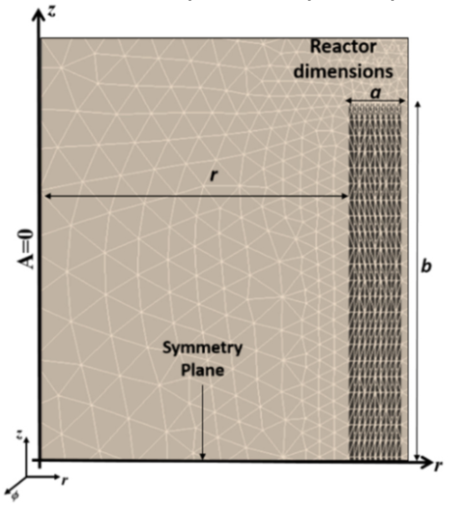

Begin
I. PREPROCESSING STEP
FEM Matrixes
K[mxm],G[mxm]
FEM Vectors
b{m}
II. CALCULATING STEP
Conductors Vector
(K[mxm]+jwG[mxm])X{m} =b{m}
SET f=f+5
w=2π(f)
III. CALCULATING VOLTAGE U_{ct}
U_{CT}=f(X{m})
IV. CALCULATING CURRENT DENSITY J_{CT} ON EACH FREQUENCY
$J_{CT@FREQ} = \sigma\frac{U_{CT}}{l}$
no
f=60?
yes
End

b) Calculating process of the device

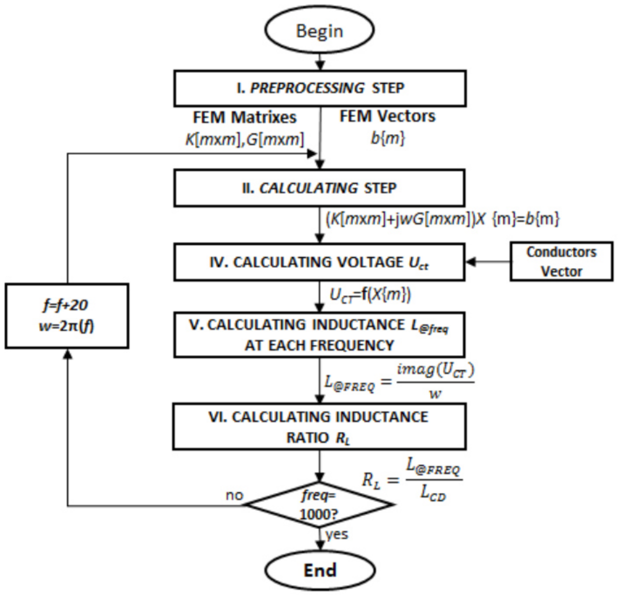
6.2 Device modelled by an Axisymmetric Symmetry Assumption

It consists on analyzing in the frequency domain, a small air-cored reactor [6]. The example consists on finding how the reactor inductance ratio (R_L=L_{ca}/L_{cd}) varies within a frequency range [6], defined from 20Hz to 1000Hz with a frequency step of 20Hz. L_{ca} is defined as the inductance obtained at a specific frequency; and L_{cd} is the inductance in a near to zero frequency. Here, the inductance ratio will be obtained via the *calculating* process shown in Fig. 4(b). The FEM model and the geometry of the series reactor is shown in Fig. 4(a).

Fig. 4 Device with an axisymmetric symmetry assumption



a) Geometry and FEM model


Begin
I. PREPROCESSING STEP
FEM Matrixes K[mxm],G[mxm]
FEM Vectors b{m}
II. CALCULATING STEP
(K[mxm]+jwG[mxm])X {m}=b{m}
IV. CALCULATING VOLTAGE U_{ct}
Conductors Vector
U_{CT}=f(X{m})
V. CALCULATING INDUCTANCE $L_{@freq}$ AT EACH FREQUENCY
$L_{@FREQ} = \frac{imag(U_{CT})}{w}$
VI. CALCULATING INDUCTANCE RATIO R_L
$R_L = \frac{L_{@FREQ}}{L_{CD}}$
f=f+20
w=2π(f)
no
freq= 1000?
yes
End

b) Calculating process of the device

6.3 Methods of Solution of the FEM equations

The two devices will be solved by the *normal* and the *reduced* FEM equations, which have the form of the expressions defined in (10) and (11), respectively. The dimensions and features of both, *normal* and *reduced* FEM equations, are listed in Table I. Please notice that the FEM equations of each device are required to be solved several times for the respective frequency range.

Table I. FEM equations to be solved in a frequency range

Device analyzed	No. FEM Eqs	*Normal* FEM equation	*Reduced* FEM equation
Planar symmetry	14	$[A_{266}]\{\tilde{x}_{266}\} = \{\tilde{b}_{266}\}$	$[A_{T,205}]\{\tilde{x}_{T,205}\} = \{\tilde{b}_{T,205}\}$
Axisym. symmetry	51	$[A_{3520}]\{\tilde{x}_{3520}\} = \{\tilde{b}_{3520}\}$	$[A_{T,1270}]\{\tilde{x}_{T,1270}\} = \{\tilde{b}_{T,1270}\}$

In order to measure the performance of the method implemented in *CUBLAS*, the *normal* and the *reduced* FEM equations were also solved in a sequential computing platform. Specifically, the *LU* routines included in the *GSL* computing platform [9]. In the sequential form of the solution, the *preprocessing* and the *calculating* steps were entirely implemented in the *GSL* platform [9]. For the parallel solution, some stages of the *preprocessing* step were calculated by a sequential computing in *GSL* [9], while the *calculating* steps were completely implemented in the *CUBLAS* computing platform [8]. Thus, the *calculating* step of the *normal* and the *reduced* FEM equation will be solved for each frequency by the *LU* method implemented in the *CUBLAS*.

Table II and III describe the specific routines that are used for the sequential and the parallel solutions of the *normal* and the *reduced* FEM equations, respectively.

Table II. Routines used in the *sequential* form of solution of the FEM Equations

Stage	*Normal* FEM Equation	*Reduced* FEM Equation
Preprocessing Step		
Normal preprocessing step	*C* routines, *GSL* matrix routines	
Deriving Submatrixes for the *reduced* equation	Not applied	*GSL* matrix routines
Calculating final matrixes for the *reduced* equation	Not applied	*gsl_blas_dgemm* *gsl_blas_dgmev*
Calculating Step		
Forming equation $[A]\{\tilde{x}\} = \{\tilde{b}\}$	*GSL* matrix routines	
LU Decomposition $[A] = [L][U]$	*gsl_linalg_complex_LU_decomp*	
Solving equation $[A]\{\tilde{x}\} = \{\tilde{b}\}$	*gsl_linalg_complex_LU_solve*	

Table III. Routines used in the *parallel* form of solution of the FEM equations

Stage	*Normal* FEM Equation	*Reduced* FEM Equation
Preprocessing Step		
Normal preprocessing step	*C* routines, *GSL* matrix routines	
Deriving Submatrixes for the *reduced* equation	Not applied	*GSL* matrix routines
Calculating final matrixes for the *reduced* equation	Not applied	(Matrix inverse calculated using routine defined in [10]) *cublasSgemm* *cublasSgemv*
Calculating Step		
Forming equation $[A]\{\tilde{x}\} = \{\tilde{b}\}$	*CUBLAS* matrix routines	
LU Decomposition $[A] = [L][U]$	See Fig. 2	
Solving equation $[A]\{\tilde{x}\} = \{\tilde{b}\}$	*cublasCtsv:* $([L]\{\tilde{y}\} = \{\tilde{b}\})$ $([U]\{\tilde{x}\} = \{\tilde{y}\})$	

The computing times obtained from solving the *normal* and the *reduced* FEM equations in the sequential and the parallel form of solution, it will be shown in the next section.

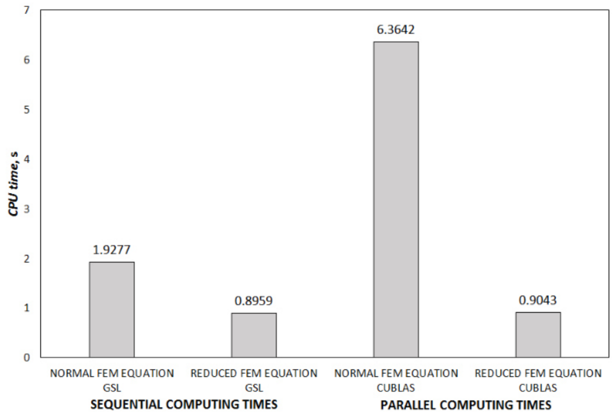
6.4 Results and Performance Comparison

It is important to mention that the results obtained from the solution of the planar and axisymmetric problems, were validated and compared against simulations performed with ANSYS in the frequency domain. The results derived by the *normal* and the *reduced* FEM equations are accurate and validate the proposed parallel form of solution of both equations.

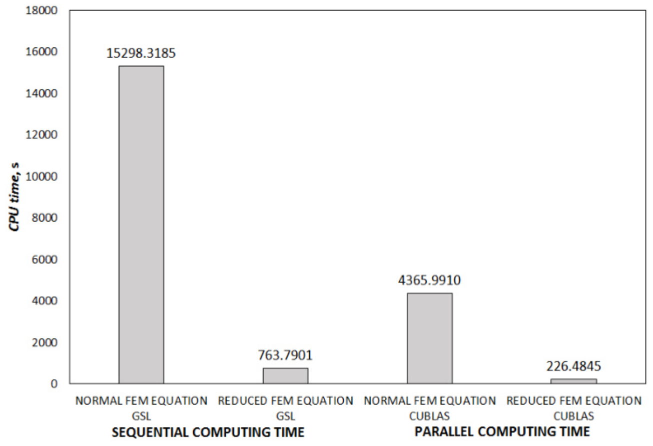
The *normal* and the *reduced* FEM equations were solved in the computing platforms *GSL* and *CUBLAS*. The programs were implemented in the same computer and operative system. A Dell Precision R5500 Rack Workstation, GPU NVIDIA® Quadro® 600, 1 GB RAM and an Ubuntu Operative System were used.

The total computation time (*CPU time*) required to solve the devices with planar and axisymmetric symmetries in the correspondent frequency range was measured. Fig. 5 illustrates the *CPU times* needed to solve these equations using the sequential and the parallel computing platforms.

Fig. 5. *CPU times* derived for the FEM equations solutions



a) *CPU time* derived for the *planar* device



b) *CPU time* derived for the *axysimmetric* device

For the case of the device with a planar symmetry, it can be observed that the *reduced* FEM equation allows to derive a faster solution compared to the *normal* FEM equation solution. Specifically, when the sequential computing was used, the *CPU time* of the *normal* and the *reduced* equation are 1.92sec and 0.89sec, respectively. Moreover, when the parallel computing was used, the *CPU time* of the *normal* and the *reduced* equation are 6.36sec and 0.90sec, respectively. Although the *reduced* FEM equation allows a faster solution with both computing platforms to be achieved, a reduction of *CPU time* was not obtained when parallel computing with *CUBLAS* was used. The reason being is that the *reduced* and the *normal* equations of the planar device are of low order, i.e. 205 and 266, respectively. A *CPU time* reduction cannot be achieved, since the advantage of using the parallel platform is only evident when the size of the equations to be solved is really huge.

For the specific case of the device with an axisymmetric symmetry, it can be observed that the *reduced* FEM equation also permits to derive a faster solution compared to the *normal* FEM equation solution. For example, for sequential processing, the *CPU time* of the *normal* and the *reduced* equation are 15298.31sec and

763.89sec, respectively. Moreover, when parallel computation was used, the *CPU time* of the *normal* and the *reduced* equation were 4365.99sec and 226.48sec, respectively. It can be seen that the parallel processing of the *reduced* FEM equation requires of only 226.48sec. The sequential computation of a normal FEM equation requires a CPU time of 15298.3sec. The difference between these *CPU times* is really significant, nearly 6760%. The reason is that the *reduced* and the *normal* equations of the planar device are of higher order, i.e. 3520 and 1270, respectively.

7 Conclusions

A method of solution of a FEM equation, using the *LU* method implemented in the *CUBLAS* computing platform has been proposed. It has the following advantages:

1) It can be used to solve a *normal* and a *reduced* FEM equation that models devices that can be simplified by a planar or an axisymmetric symmetry assumption.
2) Its solution has been compared against a sequential computing platform. It has allowed a significant reduction of computer effort, as compared to the sequential solution, which was implemented by using the LU routines included in the *GSL* platform.
3) It allows a significant time reduction when the *reduced* FEM equation is solved. A significant reduction of *CPU time* to solve larger order FEM equations sets in the frequency domain has been obtained. The *CPU time* for solving this equation using *CUBLAS* is 67.54 times lesser, than the time required for solving the *normal* FEM equation with *GSL.*

The parallel solutions of the *normal* and the *reduced* FEM equations have been successfully tested for a case study where a finite element analysis has been used to analyze planar and axisymmetric devices. The results derived by the parallel and the sequential solutions of these FEM equations have been against those obtained by finite element simulations performed in ANSYS in the frequency domain. An excellent agreement between the results obtained with both approaches has been achieved.

A significant time reduction has been achieved with the application of *CUBLAS* platform for solving the FEM equations in the frequency domain. For the specific case of the device modelled by an axisymmetric symmetry assumption, it has been obtained a *CPU time* of 226.48s which is a significant small time, compared with the *CPU time* of 15298.31s, which was derived by the sequential solution with *GSL.*

8 References

[1] X. Wang, D. Xie, "Analysis of Induction Motor Using Field-Circuit Coupled Time-Periodic Finite Element Method Taking Account of Hysteresis", IEEE Transactions on Magnetics, Vol. 45, No.3, pp. 1740-1741, March 2009.

[2] Bianchi N., Electrical Machine Analysis Using Finite Element Method, 1st ed. Taylor & Francis Group, Boca Raton, 2005.

[3] A. Arkkio, "Analysis of Induction motors based on the numerical solution of the magnetic field and circuit equations", Doctoral thesis, Acta Polytechnica Scandinavica, Electrical Engineering Series, no. 59, September 1987.

[4] R. Dominguez, A. Medina, "A Novel Method for the Solution of a Finite Element-Circuit Coupled Equation using a Reduced Equivalent Equation", Proceedings of the 2013 IEEE International Electric Machines and Drives Conference, pp. 1061-1064, May 2013.

[5] A. Konrad, "Integrodifferential Finite Element Formulation of Two-Dimensional Steady-State Skin Effect Problems", *IEEE Transactions On Magnetics,* vol. mag-18, No.1, pp. 284-286, January 1982.

[6] K. Preis, "A Contribution to Eddy Current Calculations in Plane and Axisymmetric Multiconductor Systems", IEEE Transactions On Magnetics, Vol. 19, No.6, pp. 2397-2399, November 1983.

[7] S.L. Ho, H.L. Li, W.N. Fu, "Inclusion of Interbar Currents in a Network-Field Coupled Time-Stepping Finite-Element Model of Skewed-Rotor Induction Motors", IEEE Transactions on Magnetics, vol. 35, No.5, pp. 4218-4219, September 1999.

[8] CUDA Toolkit 5.0 CUBLAS Library, Nvidia Corporation, 2701 San Tomas Express Way Santa Clara CA, pp. 59-60, 2014.

[9] GNU Scientific Library, GNU Project, 2013.

[10] R. Cisneros-Magaña, A. Medina, V. Dinavahi, "Parallel Kalman Filter Based Time-Domain Harmonic State Estimation", Proceedings of the North American Power Symposium (NAPS), pp. 1-6, September 2013.

SESSION

POSTERS

Chair(s)

TBA

A Novel Method to Minimize Side Effects by Cache Contention on the Inclusive Multi-Core Shared Cache

Hyo-Joong Suh[1]
[1] School of Computer Science and Information Engineering,
The Catholic University of Korea, Bucheon-si, Gyeonggi-do, Korea

Abstract - *Cache contention is annoying issue of the multi-core processors with private L1 cache and inclusive shared L2 cache. If core P1 runs with small working set which fits in its L1 cache while core P2 runs cache consuming process, loaded blocks in the shared L2 cache by P1 are extinguished by frequent access from P2. Furthermore, it incurs an inevitable invalidate to the L1 cache of the core P1 by the multi-level cache inclusion property. This study focused on this problem, and solves by access grouping with extension of cache status. Simulation results show the proposed method minimizes the side effects by the cache contention on the inclusive shared cache.*

Keywords: cache contention, multi-level cache, inclusion property, multi-core, shared cache

1 Introduction

Shared memory multiprocessor systems with multi-core processors are widely used from the mid-sized to the high performance computer systems. Generally, multi-core processors have higher level private cache for each core, and lower level cache shared by some cores. Fig. 1 shows typical memory hierarchy of a quad-core processor.

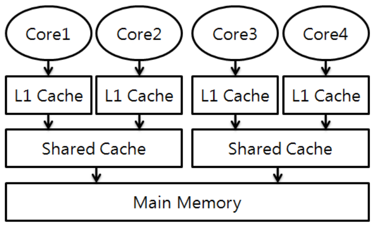


Figure 1. Memory hierarchy of a quad-core processor

In case of the hierarchical cache structure, there are two issues which this study addressed; cache coherency and inclusion property [1]. The cache coherency protocol guarantees that the validity of the cache block with latest updated content, and the inclusion property enforces the lower level cache to maintain a superset of its higher level caches such as Intel Core i7. The benefit of this inclusion is coherency exclusion of higher level caches from other cores' memory accesses.
The multi-level cache inclusion property has precious benefits of transaction complexity, latency, and bandwidth saving, but the inclusion wastes valuable cache blocks by duplication between L1 and L2 caches. Furthermore, if unbalanced workloads are dispatched on the each core with shared L2 cache, lightweight core is disturbed by invalidation requests to maintain the inclusion property between caches. Fig. 2 shows the cache contention [2] and invalidation by inclusion which this study addressed to resolve.

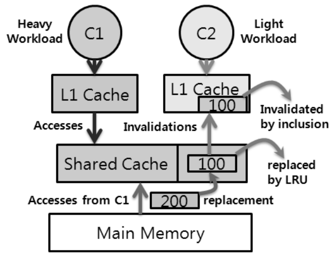


Figure 2. Side effects of the cache contention

2 Limited Inclusion by the Property

To remove the side effects by the inclusion property, first step is finding cache blocks which can be writeable among cores. Executable program contains several types of binary data. Writeable shared data are keys of the coherency transactions because the other data do not incur major coherency problems. Thus the inclusion property can be limited to the writeable shared data only. Tab. 1 shows type classifications by the data, and propose of the inclusion control methods.

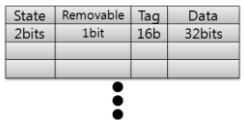
Table 1. Data type and inclusion control

Data types	Inclusion property control
executable codes	Minor control by OS (invalidate when page replacements, etc.)
read only data	
writeable private data	
writeable shared data	Controlled by H/W
run-time allocated data	

According to the data types, every memory block can be grouped as described types when the program loaded into main memory. The type distinction can be process by the compiler except some run-time allocated data. For simplified processing, the run-time data can be handled same as the writeable shared data.

3 Page Table and Shared Cache

To minimize of the inclusion property overhead, every memory access need to check whether it keeps the inclusion or not. To accomplish of this, I propose the *inclusion bit* which has the information related to the address range of memory. Thus the inclusion bits can be attached to the page table/TLB, and it is controlled by the OS. Every virtual address access from the core has to be translated to the physical address with the inclusion attribute. If a translated address need not maintain the coherency, it can be placed in the shared L2 cache with additional status bit that can be removable. Fig. 3 shows the shared L2 cache blocks with the *removable bit*.



State	Removable	Tag	Data
2bits	1bit	16b	32bits

Figure 3. Shared cache with removable bit (32KB, 8-way)

Every memory access from the core may invoke a block replacement in the shared L2 cache according to the LRU-like algorithm. If the most aged block is removable, it can be removed without invalidation of the higher level L1 cache. Fig. 4 exhibits this scenario which does not keep the inclusion property.

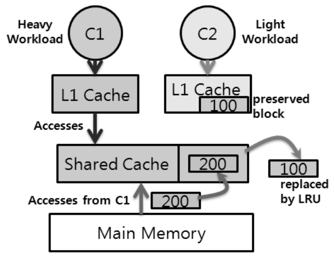


Figure 4. Shared cache replacement without inclusion

4 Simulation Results

For the evaluation of the proposed method, The DineroIV [3] cache simulator was used to test the impact of the L1 and shared L2 caches. The DineroIV is memory trace driven simulator, but does not support the hierarchical shared cache structure of multi-core processor. Thus the simulator was modified through *trace control program* which handles two memory traces, two L1 caches, and shared L2 cache.
For the memory access traces, CEXP was selected as a small workload. Other 8 programs and 507 writes (writeable shared data) from CEXP are mixed into another trace which as a memory consuming workload. Tab. 2 shows the property of detailed workloads from the SPEC'92 benchmark [4].

Table 2. Property of the simulated traces

Program	Instruction Fetch	Data Read	Data Write	Sum
CEXP	18041	1452	507	20000
Mixed trace*	42240	4722	7469	54431

*Mixed trace: COMP, EAR, HYDRO, MDLJD, NASA7, SWM, UCOMP, WAVE, 507 writes from CEXP (writeable shared data)

Generally, size of the shared L2 cache must be bigger than the sum of its higher level L1 caches for efficiency, thus the simulated sizes of the shared L2 caches are tuned to quadruple and octuple of sum of its L1 caches sizes. Figure 5 shows simulation results of conflict miss ratios originated from the non-inclusive types of data that according to the proposed method and the traditional method.

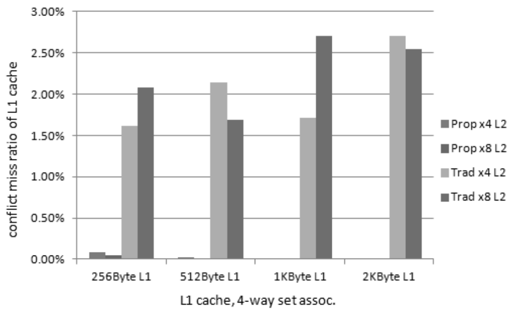


Figure 5. L1 cache conflict miss ratios by the non-inclusion types of data, with the CEXP trace

The simulation results show the inclusion property incurs most of the conflict misses in the L1 cache, and the proposed method removes these unnecessary cache misses to almost zero percent.

5 Conclusion

Generally, the inclusion property alleviates the complexity of the cache coherence, bus congestion, and memory access latency. But if unbalanced workloads dispatched into the cores with shared cache, it incurs the cache contention in the shared cache, and generates unintended invalidation to the higher level cache. The proposed method solved this issue by limiting the inclusion property to the writeable shared data, adding the inclusion bit in the page table/TLB and removable bit in the shared L2 cache.
By the simulation results, proposed method can remove most of the conflict misses of the higher level cache despite the cache contention occurred in the lower level shared cache.

Acknowledgment

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2013R1A1A2057967)

References

[1] Baer, J-L., and W-H. Wang. "On the inclusion properties for multi-level cache hierarchies"; Proc. 15th Annual International Symposium on Computer Architecture, 73-80, 1988.
[2] Y. Xie, and G. H. Loh. "Scalable shared-cache management by containing thrashing workloads"; High Performance Embedded Architectures and Compilers, 262-276, Springer Berlin Heidelberg, 2010.
[3] J. Edler and M. D. Hill. "Dinero IV Trace-Driven Uniprocessor Cache Simulator"; http://www.cs.wisc.edu/~markhill/DineroIV/.
[4] K. M. Dixit. "New CPU benchmark suites from SPEC"; In Compcon Spring'92. 37th IEEE Computer Society International Conference, 305-310, 1992.

Study of thermal-noise-assisted signal propagation of neuromorphic single-electron circuit

Ryo Hirashima[1], Tetsuya Asai[2], and Takahide Oya[1]
[1]Graduate School of Engineering, Yokohama National University
Tokiwadai 79-5, Hodogaya-ku, Yokohama 240-8501, Japan
[2]Graduate School of Information Science and Technology, Hokkaido University
Kita 14, Nishi 9, Sapporo 060-0814, Japan

Abstract - *We propose a new neuromorphic single-electron circuit that can improve speed of signal propagation by harnessing thermal noise energy. In recent years, the single-electron circuit has attracted attention in the field of nanotechnology. However, it is known that the circuit is sensitive to noise. In addition, it takes time to signal propagation. In contrast, recently, very unique neuronal behavior that an axon in a neuron improves the speed of signal propagation by harnessing noise energy has been reported. It is considered that the application of this behavior to the circuit is expected to provide a new high-speed single-electron circuit. In this study, we aim to design such the circuit to improve the signal propagation speed by harnessing the thermal noise by using a Monte Carlo simulation. As results of the simulations, it has been clarified that the signal propagation time was shortened with the increase of the thermal noise energy.*

Keywords: Single-electron circuit; Neurons; Signal propagation; Thermal noise

1 BACKGROUND AND MOTIVATION

In recent years, with the development of nanotechnology, nano-scaled elements or devices have been able to be fabricated. A single-electron device that is one of nano-scaled devices has attracted attention because it can control an individual electron by using a quantum effect [1]. It has tunneling junctions and quantum dots as main components. Since it can control a few electrons in operation, the single-electron circuit should show extremely low power consumption. However, there are some problems in the use of the single-electron circuit. Especially, the circuit is very sensitive to noise such as heat or light. These noises cause the circuit malfunction. In addition, with the increase scale of the circuit, the signal propagation will take a long time.

To solve the problems, we focus on neurons in this study. It has been already reported that a structure of an axon (an output wire) in neuron can improve the signal propagation speed of the system by harnessing noise energy efficiently [2]. Here, we imitate a behavior of a special neuron called myelinated neuron to solve the problems as mentioned above. The myelinated neuron has a special axon that is covered with an insulated layer. This insulating layer is called the myelin. In the myelin, there are some gaps called nodes of Ranvier. In the myelinated neuron, it is known that the gaps and their intervals make the signal propagation be faster. This phenomenon of the myelinated neuron characteristic is called saltatory conduction.

An operation of a single-electron oscillator that is one of the single-electron circuits we use, and that consists of a resistance, a quantum dot, a tunneling junction and a power supply [3] is similar to neurons [4]. A cell membrane structure of the axons can be represented by an equivalent circuit using single-electron oscillators. Thus, the structure of the neuron can be imitated by the single-electron circuit. Here, we aim to design a neuromorphic single-electron circuit based on the myelinated neuron to improve the signal propagation speed by harnessing thermal noise.

2 SIMULATION

Firstly, we designed a neuromorphic single-electron circuit we proposed, and evaluated its operation by using Monte Carlo simulation. As a test circuit, we prepared one-dimensional single-electron oscillators (Fig. 1) that was the equivalent circuit of a part of the axon in the neuron in the simulator. Then, we measured the required time for the signal propagation under a thermal noise environment. As results of the simulations, we found interesting operation of the proposed circuit. For example, when four tunneling junctions were connected in parallel and used as the coupling element of the one-dimensional single-electron oscillators, a signal propagation time was shortened with an increase of thermal noise (Fig. 2). Therefore, it can be said that it was possible to improve the signal propagation speed by harnessing a thermal noise. The reason we considered is that the improvement of the speed was caused by an electron tunneling at the coupling element under the thermal noise environment. The parallel structure of tunneling junctions in the coupling element area in Fig. 1 provided the some paths for tunneling electrons. Thus, we considered the tunneling rate could be improved by the proposed circuit.

3 ACKNOWLEDGMENT

This work was partly supported by JSPS KAKENHI Grant Numbers 25110015 and 24710149.

4 References

[1] H. Gravert, et al., “Single Charge Tunneling–Coulomb Blockade Phenomena in Nanostructures,” New York: Plenum, (1992).

[2] O. Marcinek, et al., “Noise-assisted spike propagation in myelinated neurons,” Physical Review E, vol. 79, 011904, (2009).

[3] T. Oya, et al., “Reaction-Diffusion Systems Consisting of Single-Electron Oscillators,” Int. Journ. of Unconventonal Computing, vol. 1, pp. 177-194, (2005).

[4] T. Oya, et al., "Neuronal synchrony detection on signle-electron neural network," Chaos, Solitons & Fractals, vol. 27, no. 4, pp. 887-894 (2006).

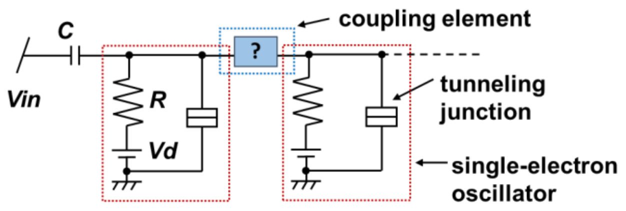


Fig. 1 one-dimensional single-electron oscillators

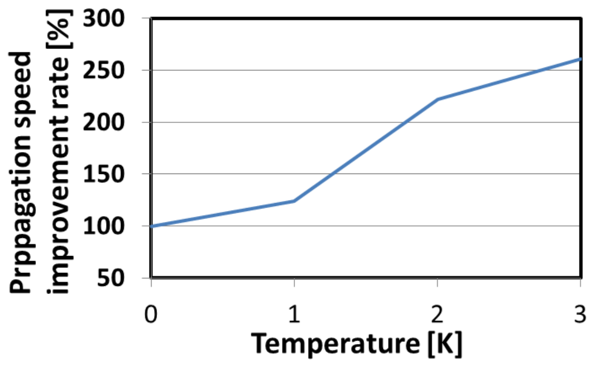


Fig. 2 simulation result

SESSION

LATE BREAKING PAPERS AND POSITION PAPERS: PARALLEL AND DISTRIBUTED PROCESSING TECHNIQUES AND APPLICATIONS

Chair(s)

Prof. Hamid R. Arabnia

Efficient Classification of Hyperspectral Images on Commodity GPUs using ELM-based Techniques

Javier López-Fandiño, Dora B. Heras, Francisco Argüello

Abstract—*Hyperspectral image processing algorithms are computationally very costly, which makes them good candidates for parallel and, specifically, GPU processing. Extreme Learning Machine (ELM) is a recently proposed classification algorithm very suitable for its implementation on GPU platforms. In this paper we propose an efficient GPU implementation of an ELM-based classification strategy for hyperspectral images. ELM can be expressed in terms of matrix operations that can take maximum advantage of the GPU architecture. Regarding the classification accuracy, the proposed algorithm achieves competitive results as compared to a traditional SVM strategy with significantly lower running times. Additionally, the use of a voting mechanism to improve the accuracy results is also considered.*

Index Terms— **Hyperspectral images, ELM, SVM, GPU, CUDA.**

1. Introduction

Nowadays, hyperspectral datasets are readily available thanks to the recent advances in sensor technology. These images provide information on hundreds of spectral bands at different wavelengths for each pixel, allowing to discriminate between different physical materials and objects. Different kind of problems can be addressed when dealing with the processing of hyperspectral datasets: classification, segmentation, and target detection, among others.

Traditional methods like RBF (radial basis function) neural networks or KNN (K-nearest neighbour) have been used for classification of hyperspectral images. However, other algorithms like SVM [1] are more suitable to process all the information stored in the spectral channels of this kind of images.

Nevertheless, to achieve real-time processing, faster methods than SVM are needed, even if high-performance computing systems are used. Extreme Learning Machines (ELM) are a brand new method that slightly improves SVM running times and that has been previously used in remote sensing [2], [3]. Additionally, ELM is a suitable algorithm to be implemented on commodity GPUs because the required operations are mostly matrix operations that can be computed in blocks without data dependencies among them.

Different ELM-based techniques have been proposed [4]: online sequential ELM, which is adequate when data is received in chunks; incremental ELM, where hidden nodes are added one by one; or pruned ELM, that starts with a large network and then eliminates the hidden nodes that have low relevance to the class labels. A well-known technique to easily improve the results of a classification method consists in the use of ensembles, namely, the combination of the results obtained through different classification techniques or the same technique with different training datasets [5]. A simple mechanism in order to combine the results of the ensemble is majority vote, that assigns the most repeated value on the ensemble to the final value of a sample. Examples of this technique applied to ELM are Ensemble Based Extreme Learning Machine (EN-ELM) [6], and Voting-based Extreme Learning Machine (V-ELM) [7].

In this paper we present an optimized CUDA (Compute Unified Device Architecture) GPU implementation of an ELM-based algorithm to classify hyperspectral datasets in real time. We also discuss different approaches in order to combine the classifiers through a majority vote mechanism, in order to improve accuracy results. The implementation keys are the exploitation of the thousands of threads available in the GPU architecture and the adequate use of the memory hierarchy. The GPU algorithm is formulated in terms of matrix operations that are efficiently executed in blocks by an optimized linear algebra library.

The next sections of this paper are organized as follows: Sect. 2.1 explains the ELM mechanism to classify hyperspectral images, while the majority vote mechanism is presented in Sect. 2.2. We introduce some GPU and CUDA fundamentals in Sect. 3. The GPU implementations are described in Sect. 4. Sect. 5 is devoted to the discussion of the experimental results. Finally, Sect. 6 summarizes the conclusions of this work.

2. Hyperspectral Image Classification Using ELM-based Techniques

In this section, we present the ELM and V-ELM algorithms, whose GPU implementation will be studied in Sect. 4.

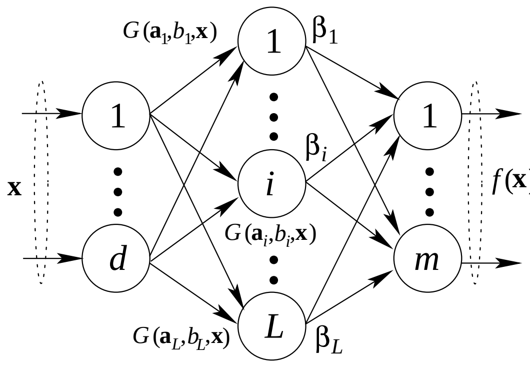
2.1 ELM-based Classification

The raw pixel-wise ELM algorithm was proposed as an efficient learning algorithm for single-hidden layer feedforward neural networks (SLFNs) [8].

The output function of a SLFN with L hidden nodes and m output nodes, and being $\boldsymbol{x}$ the input vector (see Fig.1) can be written as

$$f_L(\boldsymbol{x}) = \sum_{i=1}^{L} \boldsymbol{\beta}_i G(\boldsymbol{a}_i, b_i, \boldsymbol{x}), \quad \boldsymbol{x} \in \mathbb{R}^d, \ \boldsymbol{\beta}_i \in \mathbb{R}^m, \qquad (1)$$

Centro Singular de Investigación en Tecnoloxías da Información (CITIUS), Universidade de Santiago de Compostela. Rúa de Jenaro de la Fuente Domínguez, 15782 - Santiago de Compostela. (E-mail: {javier.lopez.fandino, dora.blanco, francisco.arguello}@usc.es)



$G(\mathbf{a}_1,b_1,\mathbf{x})$
β_1
β_i
$G(\mathbf{a}_i,b_i,\mathbf{x})$
$G(\mathbf{a}_L,b_L,\mathbf{x})$
β_L
x
$f(\mathbf{x})$
1
d
1
i
L
1
m

Fig. 1
SLFN AS USED BY ELM.

where $G(\boldsymbol{a}_i, b_i, \boldsymbol{x})$ denotes the output function of the ith hidden node, being $\boldsymbol{a}_i$, b_i the hidden node parameters and $\boldsymbol{\beta}_i$ the weight vector connecting the ith hidden node to the output nodes. For the case of additive nodes with activation function g, it can be expressed as

$$G(\boldsymbol{a}_i, b_i, \boldsymbol{x}) = g(\boldsymbol{a}_i \cdot \boldsymbol{x} + b_i), \quad \boldsymbol{a}_i \in \mathbb{R}^d, \ b_i \in \mathbb{R}, \tag{2}$$

A SLFN with L hidden nodes can approximate N arbitrary distinct samples and targets $(\boldsymbol{x}_i, \boldsymbol{t}_i) \in \mathbb{R}^d \times \mathbb{R}^m$, if the following equation system can be solved:

$$\mathbf{H}\boldsymbol{\beta} = \mathbf{T} \tag{3}$$

where

$$\mathbf{H} = \begin{bmatrix} \boldsymbol{h}(\boldsymbol{x}_1) \\ \vdots \\ \boldsymbol{h}(\boldsymbol{x}_N) \end{bmatrix} = \begin{bmatrix} G(\boldsymbol{a}_1, b_1, \boldsymbol{x}_1) & \dots & G(\boldsymbol{a}_L, b_L, \boldsymbol{x}_1) \\ \vdots & \dots & \vdots \\ G(\boldsymbol{a}_1, b_1, \boldsymbol{x}_N) & \dots & G(\boldsymbol{a}_L, b_L, \boldsymbol{x}_N) \end{bmatrix}_{N \times L}, \tag{4}$$

$$\boldsymbol{\beta} = \begin{bmatrix} \boldsymbol{\beta}_1^T \\ \vdots \\ \boldsymbol{\beta}_L^T \end{bmatrix}_{L \times m}, \quad \text{and} \quad \mathbf{T} = \begin{bmatrix} \boldsymbol{t}_1{}^T \\ \vdots \\ \boldsymbol{t}_N{}^T \end{bmatrix}_{N \times m}. \tag{5}$$

$\mathbf{H}$ is called the hidden layer output matrix of the neural network. Huang et al. [9], [4] have proven that a SLFN with randomly generated additive or RBF nodes in the hidden layer can universally approximate any continuous target function over any compact subset $\chi \subset \mathbb{R}^d$. For the case of additive nodes, the activation function g can be any infinitely differentiable function, including sigmoidal functions, and also the radial basis, sine, cosine and exponential functions among others.

Once they are randomly generated, hidden node parameters $(\boldsymbol{a}_i, b_i)$ remain fixed and training a SLFN is equivalent to finding a least-squares solution $\hat{\boldsymbol{\beta}}$ of the linear system $\mathbf{H}\boldsymbol{\beta} = \mathbf{T}$, i.e.:

$$\hat{\boldsymbol{\beta}} = \mathbf{H}^{\dagger}\mathbf{T}, \tag{6}$$

where $\mathbf{H}^{\dagger}$ is the *Moore-Penrose generalized inverse* of matrix $\mathbf{H}$ [10].

So, ELM can be summarized as follows [9], [4]:

Algorithm ELM: Given a training set $\{(\boldsymbol{x}_i, \boldsymbol{t}_i) | \boldsymbol{x}_i \in \mathbb{R}^d, \boldsymbol{t}_i \in \mathbb{R}^m, i = 1, \dots, N\}$, hidden node output function $G(\boldsymbol{a}_i, b_i, \mathbf{x})$, and hidden node number L,

1) Randomly generate hidden node parameters $(\boldsymbol{a}_i, b_i)$, $i = 1, \dots, L$ where $\boldsymbol{a}_i$ and b_i are the input weight and bias values.
2) Calculate the hidden layer output matrix $\mathbf{H}$.
3) Calculate the output weight vector, $\boldsymbol{\beta} = \mathbf{H}^{\dagger}\mathbf{T}$.

As it was stated in [11], ELM requires less human intervention than SVM because a single parameter, the number of neurons in the hidden layer, needs to be optimized, since all the other parameters are randomly initialized. It also achieves similar or better generalization performance for binary and multiclass classification cases. In addition, ELM has better scalability and it runs at much faster learning speed than SVM.

In our hyperspectral image case, each training sample represents a random selected pixel on the image and each component of the sample a spectral band of the pixel. The output that we obtain after the classification phase is the predicted class for each pixel of the image.

2.2 Voting-based ELM

In this work we follow an approach based on ensembles [7], [12], i.e., computing a number of independent ELMs with the same number of hidden nodes and the same activation function in each hidden node and combining the results obtained by the different classifiers. The individual ELMs are trained with the same dataset and the learning parameters of each ELM are randomly initialized independently.

Majority vote is the simplest method to implement among all the combination methods because it does not assume prior knowledge of the behaviour of the individual classifiers and it does not require training [13]. We use a democratic majority vote method where each classifier vote counts equal to the others and the final decision for each sample is the most repeated vote for the sample.

3. CUDA GPU Programming Model

Nowadays, commodity GPUs provide massively parallel processing capabilities based on their data parallel architecture. CUDA is a hardware/software platform that enables NVIDIA GPUs to execute programs invoking parallel functions called kernels that execute across many parallel threads [14]. These threads are organized into blocks so that each thread executes an instance of the kernel following a SIMD programming model. The blocks are arranged in a grid that is mapped to a hierarchy of CUDA cores in the GPU.

A streaming multiprocessor (called SM) contains plenty of CUDA cores and executes one or more thread blocks. The threads are executed in groups of 32 threads called warps. If all the threads in a warp execute the same code and access memory with nearby addresses the performance will be greatly improved.

Threads can access data from multiple memory spaces. First, each thread has a private local memory and registers. Each block of threads has a shared memory visible exclusively to the threads within the block with a lifetime that is equal to the block lifetime. Finally, all threads access the same global memory space (DRAM) which is persistent across kernel launches by the same application. The lower the memory level, the faster the read/write access to the data. Shared memory lifetime makes it difficult to share data among thread blocks because it implies the use of global memory whose access is slower than shared memory access.

The GPU architecture named Kepler includes a two level cache hierarchy. There are 64 KB of on-chip memory for each SM, which can be configured as half each for the shared memory and the L1 cache, 48 KB of shared memory and 16 KB of L1 cache or viceversa. There is also a unified L2 cache of 1536 KB that is shared among all the SM units. In the most recent Maxwell architecture [14] there are 64 KB of dedicated shared memory since the L1 cache is placed together with the texture memory.

Different performance optimization strategies have been applied in this work in order to optimize the computational performance:

- *Maximize the parallel execution.* It is important to organize the algorithm in computational blocks that can be independently executed minimizing communications among them.
- *Improve the efficiency in the use of the memory hierarchy.* Trying to maximize both the spatial and temporal locality in the data access, thus reducing data movement among different memory levels. It is essential to perform the maximum number of computations on data already stored in shared memory, therefore minimizing the data transfer between global and shared memory.
- *Exploit the available optimized CUDA libraries.* There are different efficient CUDA libraries for FFT, image processing, or linear algebra, among others, that can improve the performance of the code.

4. ELM GPU Implementation

In this section we present the CUDA implementation of the ELM algorithm used for the classification of hyperspectral datasets. We use the MAGMA library [15] to achieve optimal GPU linear algebra operations. This library has proven to be more efficient than others like CUBLAS [16] [15].

4.1 ELM-based Classification

In this section we briefly describe the GPU implementation of the V-ELM algorithm introduced in Sect. 2 for K independent ELMs. The algorithm consists of three main phases: preprocessing, training and test. The pseudocode in Fig. 2 shows the algorithm that has been implemented, including host and device codes. The kernels executed in GPU are placed between $<>$ symbols. The pseudocode also includes the GM and SM acronyms to indicate kernels executed only in global memory and kernels that only use shared memory, respectively.

First, all the data are scaled in the range [0:1] (line 1 in the pseudocode). Given that ELM is a supervised learning algorithm and ground truths of the datasets are available, the pixels (pixel vectors) of each dataset are randomly distributed between two non overlapping sets: training and test. These two sets are stored in the matrices $\mathbf{X}_{train}$ and $\mathbf{X}_{test}$, respectively, where each row represents a sample and each column a spectral band (lines 3 and 4 in the pseudocode). Data matrices are converted to column major format in order to be used by the MAGMA library (line 5 in the pseudocode). The ground truth labels are also split into two target matrices, $\mathbf{T}_{train}$, which is used during the learning phase, and $\mathbf{T}_{test}$, which will be used to check the accuracy results. Finally, the training and test target matrices are processed so that each row represents a sample and each column a class, where a value of 1 indicates membership to a class and a value of -1 is assigned otherwise (line 6 in the pseudocode). The preprocessing phase is computed in CPU and the results are stored in the global memory of the GPU. All the remaining steps will be computed in GPU.

The training phase starts by generating random weights and biases (line 7 in the pseudocode). The weights matrix has a number of rows equal to the number of neurons in the hidden layer. The number of columns is the same as the number of neurons in the input layer (equal to the spectral band number). This matrix takes values in the range [-1:1]. The biases vector takes values in the range [0:1] and its size is equal to the number of neurons in the hidden layer. These two matrices are then stored in the global memory of the GPU. Then, we have to calculate the hidden layer output matrix $\mathbf{H}$. To do this, we first multiply the transpose of the weights matrix by the training matrix, then we add the biases vector to the result and, finally, we apply an activation function to each element of the matrix (lines 8, 9 and 10 in the pseudocode corresponding to (2)). In our case, a sigmoid function ($f(x) = 1/(1 + e^{-x})$) is applied through a CUDA kernel with each thread operating over a single element of the matrix. The final training step consists in calculating the output weights multiplying the transpose of the pseudoinverse of matrix $\mathbf{H}$ by the training targets matrix (line 12 in the pseudocode corresponding to (6)). In the next section we will detail how to calculate the pseudoinverse of a matrix.

The test phase (lines 13 to 17 in the pseudocode) starts by multiplying the transpose of the previous generated weights matrix by the data matrix $\mathbf{X}_{test}$. Then, the biases vector is added and the activation function is applied just like in the training phase to obtain the test hidden layer matrix $\mathbf{H}$. These operations are analogous to those of lines 8 to 10 in the training phase. Then, the output matrix $\mathbf{Y}$ is calculated multiplying $\mathbf{H}^T$ by the output weights matrix β obtained in the training phase. Finally, the estimated output label $\mathbf{T}_i$ is calculated as the argument that maximizes $\mathbf{Y}_i$ for each sample,

$$\hat{\mathbf{T}}_i = \underset{c=1,\ldots,C}{\arg\max}\, \mathbf{Y}_{i,c} \tag{7}$$

```
Require: hyperspectral dataset $\mathbf{X}$, label set $\mathbf{T}$, $K$: number of ELMs, $L$: number of neurons in the hidden layer, $C$: number of classes, $N$: number
    of samples
 1: Scale hyperspectral dataset in [0:1]                                          ▷ Preprocessing phase
 2: for each ELM $k$ in V-ELM ($k = 1, \dots, K$) do
 3:     Randomly choose the training points
 4:     Take the remaining points for test
 5:     Store data in column major order matrices $\mathbf{X}_{train}$ and $\mathbf{X}_{test}$
 6:     Process target matrices $\mathbf{T}_{train}$ and $\mathbf{T}_{test}$
                                                                                  ▷ Training phase
 7:     <Generate random weights ($\mathbf{a}_i$) and biases ($b_i$), $i$=1,...,$L$>               ▷ GM
 8:     <Transpose the weight matrix and multiply by $\mathbf{X}_{train}$ >                ▷ SM
 9:     <Add the biases>                                                          ▷ SM
10:     <Apply activation (sigmoid) function to obtain $\mathbf{H}$ >                       ▷ GM
11:     <Calculate $\mathbf{H}^\dagger$ as he Moore-Penrose pseudoinverse of $\mathbf{H}$ >               ▷ GM-SM
12:     <Calculate output as $\boldsymbol{\beta} = \mathbf{H}^\dagger \times \mathbf{T}_{train}$ >                    ▷ SM
                                                                                  ▷ Test phase
13:     <Transpose weight matrix and multiply by $\mathbf{X}_{test}$ >                     ▷ SM
14:     <Add the biases>                                                          ▷ SM
15:     <Apply activation (sigmoid) function to obtain $\mathbf{H}$ >                       ▷ GM
16:     <Calculate output as $\mathbf{Y} = (\mathbf{H})^T \times \boldsymbol{\beta}$ >                          ▷ SM
17:     <Calculate the estimated output label>                                    ▷ GM
18: end for
                                                                                  ▷ Majority vote phase
19: <Calculate the output as the majority vote of the $K$ estimated labels for each sample>    ▷ GM
                                                          ▷ GM: Global Memory, SM: Shared Memory
```

Fig. 2
PSEUDOCODE FOR THE V-ELM ALGORITHM IN GPU.

4.2 Moore-Penrose Inverse of a Matrix

In this section we explain how to efficiently calculate the pseudoinverse of a matrix with CUDA using the MAGMA library. It is the most computationally costly operation in the training phase and the one that involves more steps (line 11 in the pseudocode). We implement it as described in [10].

We first check the dimensions of the input $\mathbf{H}$ matrix in order to know if the number of rows is lower than the number of columns. If this is the case, we use MAGMA to compute a matrix $\mathbf{A}$ like the multiplication of the original matrix by its transpose, otherwise we compute $\mathbf{A}$ as the multiplication of the transpose matrix by the original matrix. This operation ensures that $\mathbf{A}$ is a symmetric positive definite matrix.

The next step consists in calculating the Cholesky factorization of $\mathbf{A}$ with the MAGMA *dpotrf* function and then applying a kernel to nullify the upper triangle of the factorized matrix obtaining the $\mathbf{L}$ matrix. Unlike the other steps, this last kernel is launched in global memory.

Afterwards, an $\mathbf{M}$ matrix is calculated multiplying $\mathbf{L}^T$ by $\mathbf{L}$ and then we compute the inverse of this matrix with the MAGMA *dgetrf* and *dgetri* functions.

Finally, once all of these matrices have been calculated, we obtain the inverse of the original $\mathbf{H}$ matrix with a set of consecutive multiplications computed using the MAGMA *dgemm* function. If the dimension check of the $\mathbf{H}$ matrix at start resulted in that the row number is lower than the column number, we compute $\mathbf{H}^\dagger$ as,

$$\mathbf{H}^\dagger = \mathbf{H}^T \times \mathbf{L} \times \mathbf{M} \times \mathbf{M} \times \mathbf{L}^T,$$

otherwise we compute

$$\mathbf{H}^\dagger = \mathbf{L} \times \mathbf{M} \times \mathbf{M} \times \mathbf{L}^T \times \mathbf{H}^T.$$

4.3 Voting-based ELM

The voting algorithm used in this work, as it was explained in Sect. 2.2, comprises a set of independent ELMs whose outputs are stored in a matrix. After all the computations we will obtain an *N* by *C* matrix (being *N* the number of pixels and *C* the number of classes in the dataset) containing the vote of each ELM for every pixel of the image. This phase of the algorithm is not needed if a single ELM is launched.

```
1: for each sample $i$ ($i = 1, \dots, N$) do
2:     for each ELM $k$ ($k = 1, \dots, K$) do
3:         $\mathbf{S}_i(\hat{\mathbf{T}}_i^k) = \mathbf{S}_i(\hat{\mathbf{T}}_i^k) + 1$
4:     end for
5: $\mathbf{mvT}_i = \arg\max_{c=1,\dots,C} \mathbf{S}_{i,c}$
6: end for
```

Fig. 3
VOTING PHASE OF THE V-ELM ALGORITHM.

The majority vote phase is shown in Fig. 3. For each sample, a vector $\mathbf{S}_i$ is used to store the vote of the *K* ELMs and then, the final label ($\mathbf{mvT}_i$) is calculated as the most repeated output value produced by the different ELMs for the sample. A CUDA kernel is launched and computed in global memory where each thread computes the majority vote for one pixel of the image.

5. Results

We have evaluated the proposed algorithm on a PC with a quad-core Intel Core i5-3470 at 3.20 GHz and 8 GB of RAM. The code has been compiled using the gcc 4.6.3 version with OpenMP 3.0 support under Linux using 4 threads. Regarding the GPU implementation, CUDA codes run on an NVIDIA GeForce GTX Titan with 14 SMXs and 192 CUDA cores each. The CUDA code has been compiled using nvcc with version 5.5 of the toolkit under Linux.

The accuracy results are expressed in terms of overall accuracy (OA) average accuracy (AA) and kappa coefficient [17]. The performance results are expressed in terms of running times and speedups compared to an OpenMP CPU optimized version of the ELM parallelized using 4 threads and whose algebra operations are accelerated with the LAPACK library [18] together with GoToBLAS2 [19] . For comparison purposes, speedups of the ELM implementations are also calculated comparing to an optimized SVM implementation with the parameter values and number of training samples taken from [20]. The running times include the training and test phases, completely executed in GPU, therefore they do not include CPU-GPU data transfers.

The test were run on two hyperspectral airborne datasets [21]: A 103-band ROSIS image of the University of Pavia (Pavia Univ.) with a spatial dimension of 610 x 340 pixels and a 220-band AVIRIS image of 145 x 145 pixels taken over Northwest Indiana (Indian Pines).

We compare three different configurations using ELM:

1) A single ELM trained with 200 samples for each class (ELM).
2) A V-ELM comprising 8 ELMs trained with a total of 200 samples for each class equally spread (with bootstrap) among the ELMs. This way the number of training points used by the 8 ELMs is the same as those used by the single ELM of the previous configuration (V-ELM-1).
3) A V-ELM comprising 8 ELMs trained with 200 samples for each class for each one of the ELMs, so that each ELM is the same as in the first configuration (V-ELM-2).

The number of training samples employed are 200 per class, or half the number of samples in the class if there are not enough samples. These samples are randomly chosen and all the remaining samples are used for test. The number of hidden layer neurons employed are 500 for Pavia Univ. and 950 for Indian Pines in all the cases [2]. These configurations were chosen in order to achieve the best accuracy [3].

Table 1 shows accuracy results for the Pavia Univ. and Indian Pines images in terms of OA, AA, and kappa.

The first thing to highlight in the results is that both the ELM and V-ELM-2 configurations obtain acceptable accuracy results, being best result slightly better than the SVM for both images. Regarding the V-ELM-1 configuration, as was expected, it offers in all the cases a lower accuracy than a single ELM. This is due to the very low number of samples to train each class in each ELM, so an overfitting is produced resulting in poor generalization capabilities. This is supported by the fact that every ELM in this configuration obtains 100% accuracy in the training phase but much lower accuracy in the later test phase.

For the Pavia Univ. image, the V-ELM-2 configuration clearly improves the ELM configuration while for the Indian Pines image both configurations obtain similar results, being the ELM configuration only slightly better. Figures 4 and 5 help in understanding the results. They represent the ground truth and false color classification maps for both datasets. The class specific accuracies for the SVM and the best ELM classification method applied to each image are shown in Tables 2 and 3. It can be observed that ELM clearly improves SVM in certain classes. In the case of Pavia Univ., the best improvements are achieved for the biggest class called

Table 2

Classification accuracies per class as percentages for the SVM (data taken from [20]) and the best ELM in terms of accuracy (V-ELM-2) for the Pavia Univ. image.

	No. of available pixels	SVM	V-ELM-2
Asphalt	6631	84.93	80.86
Meadows	18649	70.79	92.78
Gravel	2099	67.16	84.57
Trees	3064	97.77	96.07
Metal sheets	1345	99.46	99.60
Bare soil	5029	92.83	92.98
Bitumen	1330	90.42	94.32
Bricks	3682	92.78	84.93
Shadows	947	98.11	99.59

Table 3

Classification accuracies per class as percentages for the SVM (data taken from [20]) and the best ELM in terms of accuracy (ELM) for the Indian Pines image.

	No. of available pixels	SVM	ELM
Alfalfa	54	74.36	78.89
Corn-notill	1434	78.18	81.48
Corn-mintill	834	69.64	76.72
Corn	234	91.85	91.47
Grass-pasture	497	92.17	95.99
Grass-trees	747	91.68	96.93
Grass-pasture-mowed	26	100	77.69
Hay-windrowed	489	97.72	99.34
Oats	20	100	78.00
Soybean-notill	968	82.03	83.15
Soybean-mintill	2468	58.95	64.47
Soybean-clean	614	87.94	90.99
Wheat	212	98.77	99.17
Woods	1294	93.01	90.56
Bldg-grass-trees-drives	380	61.52	89.00
Stone-steel-towers	95	97.78	69.38

meadows with an accuracy of 92.78% obtained by ELM while SVM obtains 70.79%. For the case of the Indian Pines dataset, the best ELM improvements are achieved for the classes *corn-mintill* (76.72% obtained by ELM and 69.64% by SVM) and *bldg-grass-tree-drives* (89.00% obtained by ELM and 61.52% by SVM). For these classes, more homogeneous areas are observed in the ELM classification maps of the Figures 4 and 5.

Table 4 shows performance results in terms of running times and speedups calculated over the OpenMP multicore implementations for the Pavia Univ. and Indian Pines datasets. It can be observed that the CUDA GPU implementation is faster than the OpenMP CPU one for all the configurations in both datasets (up to 10.1× for SVM and 8.6× for V-ELM-1 for the Pavia Univ. image). It also can be checked that the single ELM configuration is the fastest one for both datasets. Regarding the comparison to the SVM implementation, Table 5 shows that, for both images, the single ELM configuration is faster than SVM, 8.6 times faster for the Pavia Univ. image in CPU and 6.5 times for the same image in GPU.

Running times also indicate that V-ELM-2 is better than V-ELM-1 because V-ELM-1 only saves time against V-ELM-2 in the training phase. This represents a small part of the total

Table 1
CLASSIFICATION ACCURACY AS PERCENTAGES FOR THE PAVIA UNIV. AND INDIAN PINES IMAGES. THE ELMS CONTAINED 500 AND 950 NODES IN THE HIDDEN LAYER, RESPECTIVELY.

	Pavia Univ.			Indian Pines		
	OA	AA	kappa	OA	AA	kappa
SVM	81.01	88.25	75.86	78.17	85.97	75.33
ELM	86.68	89.46	82.52	80.81	86.20	77.81
V-ELM-1	79.20	77.29	72.62	63.14	75.66	58.67
V-ELM-2	90.31	91.78	85.78	79.59	80.42	72.13

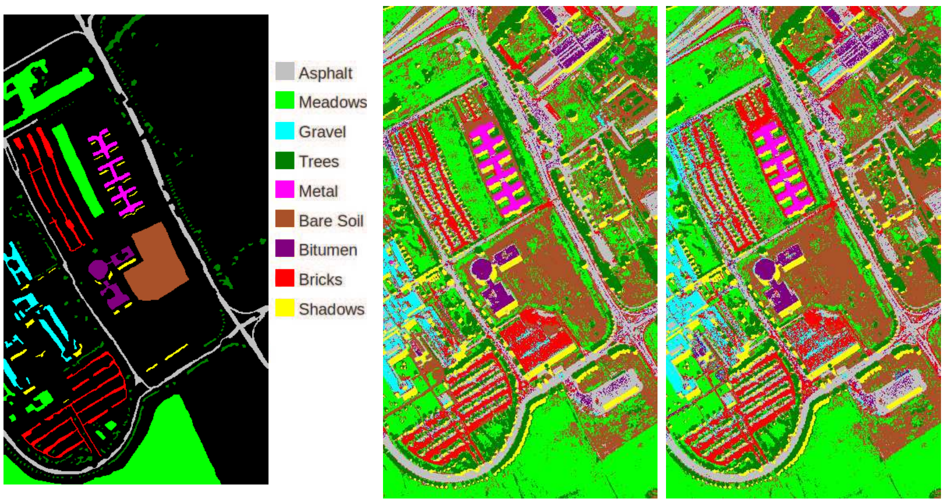


Fig. 4
FROM LEFT TO RIGHT, PAVIA UNIV. GROUND TRUTH, SVM CLASSIFICATION MAP, AND BEST ELM CLASSIFICATION MAP IN TERMS OF ACCURACY (V-ELM-2).

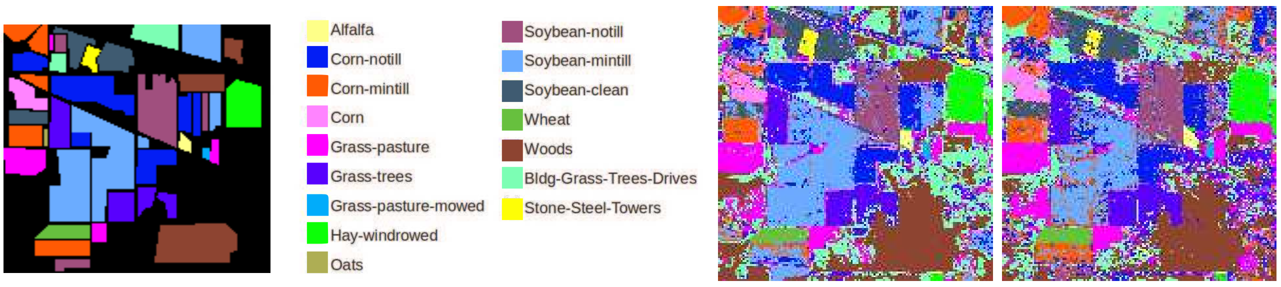


Fig. 5
FROM LEFT TO RIGHT, INDIAN PINES GROUND TRUTH, SVM CLASSIFICATION MAP, AND BEST ELM CLASSIFICATION MAP IN TERMS OF ACCURACY (ELM).

Table 4
PERFORMANCE RESULTS FOR THE PAVIA UNIV. AND INDIAN PINES IMAGES.

Pavia Univ.	SVM	ELM	V-ELM-1	V-ELM-2
OpenMP CPU	19.9844s	2.3304s	17.2394s	18.9022s
CUDA GPU	1.9802s	**0.3063s**	1.9960s	2.4501s
speedup CPU-GPU	10.1×	7.6×	8.6×	7.7×
Indian Pines	SVM	ELM	V-ELM-1	V-ELM-2
OpenMP CPU	2.6980s	1.1653s	4.5749s	9.6903s
CUDA GPU	0.4548s	**0.3096s**	0.8032s	2.6058s
speedup CPU-GPU	5.9×	3.8×	5.7×	3.7×

Table 5
SPEEDUPS AGAINST SVM FOR PAVIA UNIV. AND INDIAN PINES IMAGES.

	Pavia Univ.			Indian Pines		
	ELM	V-ELM-1	V-ELM-2	ELM	V-ELM-1	V-ELM-2
OpenMP CPU	**8.6**×	1.2×	1.1×	2.3×	0.6×	0.3×
CUDA GPU	**6.5**×	1.0×	0.8×	1.5×	0.6×	0.2×

time, resulting in a final time slightly higher for V-ELM-2 while its accuracy is better as we explained before.

The V-ELM-2 configuration provides more stable results than a single ELM in exchange for a higher execution time. The V-ELM-2 configuration is more interesting when the dataset size is large because if the dataset is too small (as in the case of Indian Pines) there are not enough samples to take advantage of the voting. Besides, in big datasets, the ELM algorithm has larger speedups against SVM allowing the voting configurations to be executed in almost the same time as a single SVM, as shown in Table 5.

Summarizing, on the one hand, the raw ELM algorithm described in this paper is significantly faster than SVM and, on the other hand, the V-ELM-2 algorithm always approaches or improves the raw ELM accuracy.

6. Conclusions

In this paper we have presented an ELM-based GPU implementation to efficiently classify hyperspectral datasets exploiting efficiently the hundreds of threads available, using shared memory to make an effective use of the memory hierarchy, and exploiting a linear algebra library. Different ensemble configurations were also considered to achieve better classification accuracies.

Results have shown that commodity GPUs like the GTX Titan used in this work are good candidates to reduce computation times in order to achieve real-time hyperspectral processing. For the raw ELM and the Pavia Univ. and Indian Pines datasets, speedups of 7.6× and 3.8× are achieved, respectively, compared to the ELM CPU classification. Results also show that the hyperspectral dataset classification using ELM is faster than the SVM one, up to 8.6× faster in CPU and 6.5× in GPU.

Acknowledgements

This work was supported in part by the Ministry of Science and Innovation, Government of Spain, cofounded by the FEDER funds of European Union, under contract TIN 2010-17541, and by Xunta de Galicia, Program for Consolidation of Competitive Research Groups ref. 2010/28. Javier acknowledges financial support from the Xunta de Galicia, under a predoctoral grant.

References

[1] F. Melgani and L. Bruzzone, "Classification of hyperspectral remote sensing images with Support Vector Machines," *IEEE Trans. Geosci. Remote Sens.*, vol. 42, no. 8, pp. 1778–1790, 2004.
[2] M. Pal, "Extreme-Learning-Machine-based land cover classification," *International Journal of Remote Sensing*, vol. 30, no. 14, pp. 3835–3841, 2009.
[3] D. B. Heras, F. Argüello, and P. Quesada-Barriuso, "Exploring ELM-based spatial–spectral classification of hyperspectral images," *International Journal of Remote Sensing*, vol. 35, no. 2, pp. 401–423, 2014.
[4] G.-B. Huang, D. H. Wang, and Y. Lan, "Extreme Learning Machines: a survey," *International Journal of Machine Learning and Cybernetics*, vol. 2, no. 2, pp. 107–122, 2011.
[5] Y. Lan, Y. C. Soh, and G.-B. Huang, "Ensemble of online sequential Extreme Learning Machine," *Neurocomputing*, vol. 72, no. 13, pp. 3391–3395, 2009.
[6] N. Liu and H. Wang, "Ensemble based Extreme Learning Machine," *IEEE Signal Process. Lett.*, vol. 17, no. 8, pp. 754–757, 2010.
[7] J. Cao, Z. Lin, G.-B. Huang, and N. Liu, "Voting based Extreme Learning Machine," *Information Sciences*, vol. 185, no. 1, pp. 66–77, 2012.
[8] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme Learning Machine: theory and applications," *Neurocomputing*, vol. 70, no. 1, pp. 489–501, 2006.
[9] G.-B. Huang, L. Chen, and C.-K. Siew, "Universal approximation using incremental constructive feedforward networks with random hidden nodes," *IEEE Trans. Neural Netw.*, vol. 17, no. 4, pp. 879–892, 2006.
[10] P. Courrieu, "Fast computation of Moore-Penrose inverse matrices," *arXiv preprint arXiv:0804.4809*, 2008.
[11] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme Learning Machine for regression and multiclass classification," *IEEE Trans. Syst., Man, Cybern. B*, vol. 42, no. 2, pp. 513–529, 2012.
[12] M. van Heeswijk, Y. Miche, E. Oja, and A. Lendasse, "GPU-accelerated and parallelized ELM ensembles for large-scale regression," *Neurocomputing*, vol. 74, no. 16, pp. 2430–2437, 2011.
[13] L. Lam and C. Y. Suen, "Application of majority voting to pattern recognition: an analysis of its behavior and performance," *IEEE Trans. Syst., Man, Cybern. A*, vol. 27, no. 5, pp. 553–568, 1997.
[14] Nvidia, *NVIDIA GeForce GTX 750 Ti Whitepaper*, 2012.
[15] S. Tomov, R. Nath, P. Du, and J. Dongarra, "MAGMA Users' Guide," *ICL, UTK (November 2009)*, 2011.
[16] Nvidia, *CUBLAS Library User Guide*, 2013.
[17] J. A. Richards and X. Jia, *Remote sensing digital image analysis*. Springer, 1999, vol. 3.
[18] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed., ISBN 0-89871-447-8, Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999.
[19] T. A. C. Center, "Gotoblas2," 2014. [Online]. Available: https://www.tacc.utexas.edu/tacc-projects/gotoblas2
[20] Y. Tarabalka, J. Chanussot, and J. A. Benediktsson, "Segmentation and classification of hyperspectral images using Minimum Spanning Forest grown from automatically selected markers," *IEEE Trans. Syst., Man, Cybern. B*, vol. 40, no. 5, pp. 1267–1279, 2010.
[21] C. I. G. from the Basque University (UPV/EHU), "Hyperspectral remote sensing scenes," 2014. [Online]. Available: http://www.ehu.es

A second generation of DEFG: Declarative Framework for GPUs

Robert Senser and Tom Altman
Department of Computer Science and Engineering, University of Colorado Denver, Denver, CO

Abstract - *DEFG is our declarative language and framework for the efficient generation of OpenCL GPU applications. Using our new DEFG implementation, run-time and lines-of-code comparisons are provided for three well-known algorithms: Sobel image filtering, breadth-first search and all-pairs shortest path. The DEFG declarative language and corresponding OpenCL kernels provide complete OpenCL applications. The lines-of-code comparison demonstrates that the C/C++ DEFG applications require significantly less coding than hand-written CPU-side OpenCL applications. The run-time results demonstrate equivalent, or better, performance characteristics compared to the hand-written applications.*

Keywords: OpenCL, algorithms, declarative language, C++

1 Introduction

This paper is a continuation of our previous work [1], where a description of the DEFG prototype and its associated performance results were introduced. This paper describes our completed DEFG Version 2 and reports on the early promising tests results showing significant improvement in performance and functionality.

Producing high performance computing (HPC) software for use on graphical processing units (GPUs) is often a difficult and daunting task. This type of software tends to require the use of specialized, parallel algorithms and requires the use of low-level application programming interfaces (APIs), in the context of a thorough understanding of the GPU architecture. The *Declarative Framework for GPUs* (DEFG) provides a domain-specific computer language (DSL) designed to assist the software developer. It mitigates the need for a deep understanding of the full CPU-side OpenCL API, therefore allowing the developer to focus on the algorithms being used and on the most efficient usage of the overall GPU architecture.

Our research in processing large, sparse graphs on GPUs has, out of necessity, led to the direct development of DEFG. As these large graphs tend to lack locality of reference, the parallel algorithms needed to process them efficiently tend to be complex. Sample problem domains range from graph problems such as the Breadth-First Search (BFS), Single-Source Shortest Path (SSSP), and All-Points Shortest Path (APSP) to iterative matrix inversion, parallel prefix computation, image processing, and parallel sorting. Using DEFG permits us to focus on the algorithms, which are coded mainly in the GPU kernels, and to spend less time focusing on the CPU-side code. In this full implementation of DEFG, we have implemented and measured, in terms of lines-of-code and run-time performance, three well-known algorithms: Sobel image filtering for edge detection [2] and from the graph theory: BFS and APSP [3].

Common GPU environments in use today, such as OpenCL [4] and NVIDIA's proprietary CUDA [5], tend to provide low-level, very specialized APIs. Their usage requires an understanding of complex, CPU-side APIs [6]. DEFG provides several higher-level design patterns that abstract the CPU-side coding to a declarative level. Much as the now-ubiquitous relational databases accept database requests as declarative SQL statements and quickly return the requested data, DEFG uses design patterns and declarative statements to produce high performance CPU-side code, which performs the desired computations. Once the developer has produced the kernel code to be executed on the GPU, DEFG simplifies the task of executing this kernel code. Complex CPU-side operations outside the context of the DEFG design patterns can be utilized within DEFG as callable functions.

This DEFG implementation consists of a parser written in Java, using ANTLR 3 [7], a Java-based optimizer, and a code generator, which is written in C++. The parser handles syntax checking and results in an abstract syntax tree, expressed as an XML document. This tree is then optimized for run-time performance and decorated with cross-reference information needed for code generation. The tree is then processed by a code generator, which uses the TinyXML2 library [8] to accept the XML-based tree. For example, the twelve lines of DEFG code expressed in Figure 1 result in approximately 460 lines of C/C++ code, a snippet of which is shown in Figure 2. All sample code is shown in Section 6, at the end of this paper. The OpenCL kernel executed by this code is shown in Figure 3. Note that this generated OpenCL code is designed to execute on any OpenCL-supported device, including the CPU.

OpenCL is an open and cross-platform standard for developing high performance applications on parallel hardware. This standard is supported by major vendors including NVIDIA, AMD, and Intel. There are two major components defined by the standard: the OpenCL C programming language used on the parallel device and the CPU-side APIs for C/C++ that provide access to the device's OpenCL kernels. The CPU manages the execution of the kernels on the OpenCL parallel device.

The CPU-side code obtains the kernel source code and then calls the appropriate OpenCL APIs to compile this kernel source code. In addition, the OpenCL CPU-side code acquires and manages the low-level buffers accessed by the device kernel. These required actions tend to make the CPU-side code quite verbose and often complex; additional API complexity is added by the OpenCL requirement to support many different types of parallel platforms and devices, examples being CPUs, GPUs, and even specialized FPGA [9] and DSP [10] hardware. This flexibility unfortunately adds numerous specialized API parameters to the OpenCL API. It can be argued that the OpenCL API is unnecessarily complex, not easily learned, and somewhat hard to use and debug. DEFG takes over much of the burden of writing the OpenCL CPU-side code, thus permitting the developer to focus on the device kernels and the actual parallel algorithms.

We approached our work as follows: using three existing OpenCL applications and using their existing OpenCL kernels without any changes, we replaced the existing CPU-side code with the DEFG-generated code. The DEFG source modules needed approximately 90% fewer lines of code. We then compared the computational performance of the three applications over two different OpenCL platforms. Performance variations between the DEFG results and the reference results were identified and analyzed.

Section 2 describes related work and includes a description of the three existing OpenCL applications, which we used as reference/benchmark applications and converted to DEFG. The DEFG language is briefly described in Section 3. We then present our experimental results in terms of lines-of-code counts and run times in Section 4. A summary of ongoing and future work is presented in the last section.

2 Related Work

Numerous attempts have been made to construct languages, compilers, and tools to make the production of high performance parallel solutions easier. In 2005, Shen et al. [11] talked about the "holy grail" of parallelization, which is the automated parallelization of serial programs, being out of reach. However, progress is being made. One approach towards the efficient production of GPU-based parallel solutions is the use of domain-specific languages (DSL). DEFG is a DSL, a language and associated tools that facilitate the production of OpenCL applications. Martin Fowler defines a DSL as a "computer programming language of limited expressiveness focused on a particular domain," and suggests that DSLs can be broken into two categories: *interna*l DSLs and *external* DSLs [12]. DSLs of both varieties have been produced for GPU-based HPC.

Internal DSLs for GPU-based HPC include extensions to Python such as: PyGPU [13], PyCUDA [14], and PyOpenCL [15]. These DSLs tend to consist of Python wrappers placed around a particular GPU API. There are also C/C++ extensions, such as Bacon [16]. Aside from DEFG, other GPU external DSLs include the SPL digital signal processing language [17] and the MATLAB Parallel Computing Toolbox (which supports CUDA and permits passing some MATLAB functions to the GPU and permits GPU kernel execution [18]). Both MATLAB and DEFG require that the GPU kernel be provided.

The BFS and APSP implementations we chose for our DEFG testing are existing implementations, easily obtained from software development kits (SDKs) and benchmarks [19-20]. Obviously, there exist other published algorithms and implementations that may provide better overall run-time performance but that is not the primary goal of this research. We have implemented a subset of these algorithms in DEFG and will present our results in a future paper. For example, Merrill, et al. suggest a much faster BFS solution which uses *prefix sum* to help distribute the work among GPU threads without locking [21]. For APSP, Katz and Kider provide a method for using *tiling* with the Floyd-Warshall APSP algorithm to minimize GPU global memory access times [22].

3 DEFG Framework Language

The DEFG *declarative language* consists of a number of `declare`, `execute and call` statements, and some *optional statements* such as `sequence/times` and `loop/while`. An example DEFG source file is shown in Figure 1. The `declare` statement is used to name the DEFG application, to define and name the GPU kernels to be executed, to define any required scalar variables, such as a graph's node count, and to define the buffers to be given to the GPU. Lines 1 to 8, in the DEFG sample, express `declare` statements. The syntax on line 6, enclosed in "[["and"]]" symbols, is our method for setting the global grid size. The `call` statement is used to invoke C/C++ functions, e.g., to obtain the input data; the sample has `call` statements on lines 9 and 11. The `execute` statement on line 10 is used to execute the kernel. The flow of control is a design pattern built into DEFG.

The optional DEFG statements can be used to provide support for more complex design patterns where the kernels may have to be executed a variable number of times. Figure 4 contains a DEFG example which executes the kernel once for each graph node. Figure 4, line 9, shows the `sequence` statement application. DEFG contains statements to process scalar values returned by kernels. This capability was used in the DEFG BFS solution to conditionally stop the parallel device execution. DEFG Version 2 generates OpenCL 1.1 code in keeping within the limits of NVIDIA's current OpenCL support [23].

Table 1: Test Configurations

Name	**Configuration Data**
CPU	Windows 7, Intel I3 Processor, 1.33 GHz, 4 GB RAM, using AMD OpenCL SDK 2.8 (no GPU)
GPU-Tesla T20	Penguin Computing Cluster, Linux Cent OS 5.3, AMD Opteron 2427 Processor, 2.2 GHz, 24 GB RAM, using NVIDIA OpenCL SDK 4.0, NVIDIA Tesla T20 with 14 Compute Units, 1147 MHz and 2687M RAM

Table 2: Lines of Code

	DEFG	**DEFG**	
	Declarative	**Generated**	**Reference**
BFS	42	620	364
FW	12	481	478
SOBEL	12	467	442

Table 3: Run-time Performance, in milliseconds

	CPU		**GPU-Tesla T20**	
	DEFG	**Reference**	**DEFG**	**Reference**
BFS-4096	1.5	2.6	4.3	5.8
BFS-65536	12.3	14.2	8.0	11.3
FW	111.8	152.0	6.0	51.2
SOBEL	23.0	24.8	3.7	4.1

4 Discussion of Results

To test the viability of DEFG, we selected three existing OpenCL solutions based on well-known algorithms: Sobel image filtering and Floyd-Warshall APSP, both from the AMD APP SDK [17], and breadth-first search from the OpenDwarfs benchmark [18]. We will refer to these solutions as SOBEL, FW, and BFS, respectively. SOBEL was chosen because it represents the class of simpler GPU problems, where a single kernel is called once and because it has significant RAM locality of reference. DEFG can support several concurrent GPU devices, in a declarative manner, and SOBEL provides a good test case for this added capability. This capability will be more fully covered in a future paper.

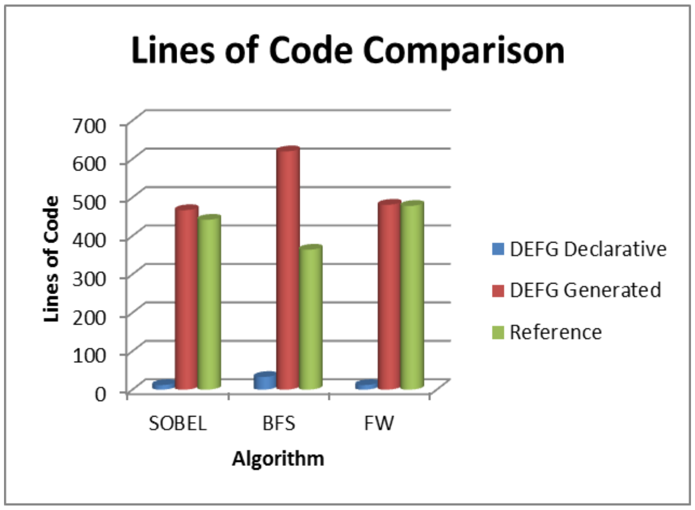
FW and BFS were selected because they represent two different classes of graph-oriented GPU problems, with the BFS solution requiring multiple GPU kernels. The FW algorithm simply requires that a common operation be repeated for each graph node. In this FW implementation, the OpenCL kernel is called once for each node. This call-for-each-node behavior must be managed from the CPU-side. The OpenDwarfs BFS implementation is based on the work by Harish [24] and uses a version of Dijkstra’s algorithm [3]. The actual OpenDwarfs code is an OpenCL port of the BFS CUDA code from the Rodinia benchmark [25]. This BFS implementation requires that a pair of kernels be repeated until success is indicated by the second kernel. This repetition is managed by the CPU-side code.

All three of these were converted to DEFG, keeping the unmodified OpenCL kernels. The conversions to DEFG produce exactly the same results as the corresponding reference version. Before discussing the performance results, we summarize the hardware and software used. The tests were run on two configurations, which we call CPU and GPU-Tesla T20, which are listed in Table 1.

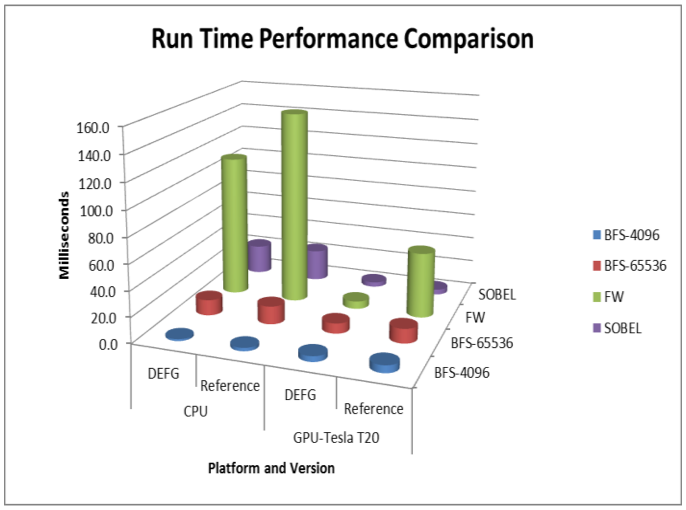
In terms of developer-written module line count results, the three DEFG versions were much smaller than their reference counterparts. Table 2 shows the line counts for SOBEL, BFS, and FW. Shown are the number of lines of DEFG declarative code, the number of lines of generated code, and the estimated number of non-comment lines in the reference version. This data is shown graphically in Plot 1. On average, the DEFG code is 4.2 percent of the generated code, and 5.1 percent of the reference code. It should be noted that the reference code tended to include additional functionality and that the DEFG generated-code counts include an additional 150 lines of template code used to identify and select the requested GPU devices.

The run-time performance comparison turned out to be very interesting. The raw run times, in milliseconds, are presented in Table 3. Plot 2 shows this data presented in 3D form. The results shown are the average of ten runs done for each case. Where we encountered unexpected results, we often reran the tests with manual code changes to isolate the underlying technical causes. We made these code changes to both the DEFG and reference OpenCL code. However, the numbers shown here are only the original times, i.e., those prior to any manual code modifications.

SOBEL is the simplest application and the run-time performance results obtained are comparable, as expected. The SOBEL results are shown on the graph in purple. The DEFG performance was slightly faster on the CPU and GPU-Tesla T20. DEFG needed 23.0 ms and 3.7 ms, respectively, while the reference case needed 24.8 ms and 4.1 ms.



Plot 1: Size Comparison of Module Sizes



Plot 2: Performance Comparison of Run Times

The run-time results of the FW tests, which are shown in green, surprised us. We saw no obvious explanation for why DEFG should be substantially faster. We reviewed the OpenCL code for both DEFG and the AMD SDK-supplied reference case, and did not find any significant differences in buffer usage or the OpenCL API functions used. We did notice that the reference case was using asynchronous events (when not required) and we temporarily disabled them and reran the reference case. The FW T20 reference case run times dropped three-fold from an average 51.2 ms to 17 ms. This difference was later traced to what we identified as an error in the reference case's OpenCL event handling.

The BFS run-time comparisons used two different graphs. The first graph has 4,096 nodes, shown in blue on the graph, and the second has 65,536 nodes, shown in red. Our earlier prototype version of DEFG was substantially slower than the reference BFS; prototype DEFG needed 59.4 ms to perform what the reference case BFS did in 11.3 ms. The DEFG Version 2 buffer-use optimization reduced the average BFS T20 run time from 59.4 ms to 8 ms! This drastic improvement in performance is due to the optimizer's removal of unneeded buffer transfer operations between the CPU and GPU.

We cannot leave the BFS performance topic without noting that the OpenCL CPU configuration's performance was better than the GPU performance for the Tesla 4,096 node case. We postulate that this is explained by the BFS implementation being used. This graph algorithm implementation is based on the work by Harish [24], which does not compensate for the lack of RAM cache found in many GPU designs. The CPU version most likely fared so well due to the multiple levels of memory caching provided by the Intel I3; it is likely that the 4,096 node test case fit entirely into the Intel I3's cache.

In summary, these four comparison tests have shown that, at least in these cases, the declarative approach used in DEFG can be used to produce OpenCL applications with fewer lines of code and comparable, or better, performance levels.

5 Ongoing and Future Work

This full DEFG implementation has shown that our declarative approach is able to produce good results with less code written while maintaining similar run-time performance, at least for this family of test cases. The addition of buffer optimization has greatly benefited the DEFG buffer management performance and, hence, the overall run times. DEFG also will significantly benefit from the addition of high-performance data loaders and result displays, as well as simple debugging aids such as logging and formatted buffer dumps. We have already enhanced the DEFG toolkit to support the use of multiple GPUs and to generate callable C/C++ modules. We have implemented the generation of human-readable OpenCL C/C++ code, which is a starting point for the creation of customized GPU applications.

DEFG was developed as a result of a specific need; that need being the rapid and efficient production of CPU-side code for use in GPU-based parallel algorithms research. Our DEFG results continue to be very promising. DEFG provides a tool to achieve the quick utilization of new OpenCL kernels and algorithms. Given this success, we anticipate enhancing DEFG further and eventually making it publicly available. The DEFG toolkit should be a useful asset in future GPU high-performance algorithms research.

6 Sample Code Figures

```
declare application  sobel
 declare integer Xdim (0)
            integer Ydim (0)
            integer BUF_SIZE (0)
 declare gpu gpuone ( * )
 declare kernel  sobel_filter SobelFilter_Kernels  ([[2D,Xdim,Ydim ]] )
 declare integer buffer image1 ( Xdim Ydim ) halo (1)
            integer buffer image2 ( Xdim Ydim ) halo (1)
 call init_input (image1(in) Xdim (out) Ydim (out) BUF_SIZE(out))
 execute run1 sobel_filter ( image1(in) image2(out) )
 call disp_output (image2(in) Xdim (in) Ydim (in) )
end
```

Figure 1: Sample DEFG Code

```
// *** buffers in
cl_mem buffer_image1 = clCreateBuffer(context, CL_MEM_READ_ONLY|CL_MEM_COPY_HOST_PTR, (BUF_SIZE * sizeof(int)),(void *) image1, &status);
if (status != CL_SUCCESS) { handle error }
status = clSetKernelArg(sobel_filter, 0, sizeof(cl_mem), (void *)&buffer_image1);
if (status != CL_SUCCESS) { handle error }
cl_mem buffer_image2 = clCreateBuffer(context, CL_MEM_WRITE_ONLY, (BUF_SIZE * sizeof(int)),(void *) NULL, &status);
if (status != CL_SUCCESS) { handle error }
status = clSetKernelArg(sobel_filter, 1, sizeof(cl_mem), (void *)&buffer_image2);
if (status != CL_SUCCESS) { handle error }
// *** execution
size_t global_work_size[2]; global_work_size[0] = Xdim ; global_work_size[1] = Ydim ;
status = clEnqueueNDRangeKernel(commandQueue, sobel_filter, 2, NULL, global_work_size, NULL, 0, NULL, NULL);
if (status != CL_SUCCESS) { handle error }
// *** result buffers
status = clEnqueueReadBuffer(commandQueue, buffer_image2, CL_TRUE, 0, BUF_SIZE * sizeof(int), image2, 0, NULL, NULL);
if (status != CL_SUCCESS) { handle error }
```

Figure 2: Snippet of Generated OpenCL Code

```
 __kernel void sobel_filter(__global uchar4* inputImage, __global uchar4* outputImage) {
        uint x = get_global_id(0);  uint y = get_global_id(1);
        uint width = get_global_size(0);  uint height = get_global_size(1);
        float4 Gx = (float4)(0);  float4 Gy = Gx;
        int c = x + y * width;
        /* Read each texel component and calculate ..*/
        if( x >= 1 && x < (width-1) && y >= 1 && y < height - 1)
        {
                float4 i00 = convert_float4(inputImage[c - 1 - width]);
                // similar lines omitted
                float4 i22 = convert_float4(inputImage[c + 1 + width]);
                Gx =   i00 + (float4)(2) * i10 + i20 - i02  - (float4)(2) * i12 - i22;
                Gy =   i00 - i20  + (float4)(2)*i01 - (float4)(2)*i21 + i02  -  i22;
                /* taking root of sums of squares of Gx and Gy */
                outputImage[c] = convert_uchar4(hypot(Gx, Gy)/(float4)(2));
        }
 }
```

Figure 3: Snippet of Sobel OpenCL Kernel Code (from AMD APP SDK 2.8) [19]

```
declare application  floydwarshall
  declare integer NODE_CNT (0)
                integer BUF_SIZE (0)
  declare gpu gpuone ( any )
  declare kernel  floydWarshallPass FloydWarshall_Kernels  ( [[ 2D,NODE_CNT ]] )
  declare integer buffer buffer1 ( BUF_SIZE )
                integer buffer buffer2 ( BUF_SIZE )
  call init_input (buffer1(in) buffer2(in) NODE_CNT(out) $BUF_SIZE(out))
  sequence NODE_CNT times
    execute run1 floydWarshallPass ( buffer1(inout) buffer2(inout) NODE_CNT(in) DEFG_CNT(in) )
  call disp_output (buffer1(in) buffer2(in) NODE_CNT(in))
end
```
Figure 4: Sample DEFG Code Showing a Sequence

7 References

[1] Senser, R. and Altman, T. "DEF-G: Declarative Framework for GPU Environment." *Proceedings of The 2013 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'13), vol II*, pp. 490-496, World-comp.org, 2013.

[2] Vincent, O. and Folorunso, O. "A descriptive algorithm for sobel image edge detection." *Proceedings of Informing Science & IT Education Conference (InSITE)*, pp. 97-107, 2009.

[3] Cormen, T. et al. Introduction to Algorithms (3rd ed.). MIT Press and McGraw-Hill, 2009.

[4] The OpenCL Specification 1.1, [Online]. Available: http://www.khronos.org/opencl/

[5] CUDA 5 Programming Guide, [Online]. Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide

[6] OpenCL Reference Pages, [Online]. Available: http://www.khronos.org/registry/cl/sdk/1.1/docs/man/xhtml/

[7] ANTLR 3, [Online]. Available: http://www.antlr3.org/

[8] Tiny XML2, [Online]. Available: http://www.grinninglizard.com/tinyxml2/index.html

[9] Altera Corporation OpenCL, [Online]. Available: http: www.altera.com/opencl

[10] Texas Instruments OpenCL, [Online]. Available: http://e2e.ti.com/support/dsp/omap_applications_processors/f/447/t/132798.aspx

[11] Shen, J. and Lipasti, M. Modern Processor Design: Fundamentals of Superscalar Processors. Boston: McGraw-Hill Higher Education, 2005.

[12] Fowler, M. Domain-specific Languages, Addison-Wesley Professional, 2010.

[13] PyGPU, [Online]. Available: http://fileadmin.cs.lth.se/cs/Personal/Calle_Lejdfors/pygpu/

[14] PyCUDA, [Online]. Available: http://mathema.tician.de/software/pycuda

[15] PyOpenCL, [Online]. Available: http://mathema.tician.de/software/pyopencl

[16] Tuck, N. "Bacon: A GPU Programming Language With Just in Time Specialization (Draft)." University of Massachusetts Lowel, Lowel MA 01854.

[17] Xiong, J. et al. "SPL: a language and compiler for DSP algorithms." *Proceedings of the ACM SIGPLAN 2001 conference on Programming language design and implementation (PLDI '01)*, pp. 298-308, ACM, New York, NY, USA, 2001.

[18] Matlab Parallel Computing Toolbox, [Online]. Available: http://www.mathworks.com/products/parallel-computing/

[19] AMD APP SDK 2.8, [Online]. Available: http://developer.amd.com/tools/heterogeneous-computing/amd-accelerated-parallel-processing-app-sdk/

[20] OpenDwarfs Benchmark, [Online]. Available: https://github.com/opendwarfs/OpenDwarfs

[21] Merrill D. et al. "Scalable GPU graph traversal." *Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming (PPoPP '12)*, pp. 117-128, ACM, New York, NY, USA, 2012.

[22] Katz, G. and Kider J. "All-pairs shortest-paths for large graphs on the GPU." *Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, pp. 47-55. Eurographics Association, 2008.

[23] NVIDIA Cuda Zone – OpenCL, [Online] Available: https://developer.nvidia.com/opencl

[24] Harish, P. and Narayanan, P. "Accelerating large graph algorithms on the GPU using CUDA." *High Performance Computing–HiPC 2007*, pp. 197-208, 2007.

[25] Rodinia GPU Benchmark, [Online]. Available: http://lava.cs.virginia.edu/Rodinia/

System Level Comparative Performance Analysis of H.264 Encoder by Network-on-Chip Topologies

Suk Ki Lee[1], **Jong Kang Park**[2], **and Jong Tae Kim**[1, 2]
[1]Department of IT Convergence, Sungkyunkwan University, Suwon-si, Gyeonggi-do, Korea
[2]Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon-si, Gyeonggi-do, Korea

Abstract – *This paper describes comparative analysis results of system performance by network-on-chip topologies on system level. Network-on-chip systems are implemented by typical topologies, mesh topology, crossbar topology, folded torus topology and point-to-point topology on system level. Running the x264/AVC encoding application on each system and analyze performance from simulation results. We can find the fact that performance differences exist when running the benchmark application, by network-on-chip topologies in advance. On each implemented system, there are performance differences ranged from -5.45% to +4.75% based on mesh topology. For analyzing the effect of topologies, we use the CPUs total cycles, which reflect communication latency. Types of topologies affect the number of routers on the paths related to latency and performance. More components of on-chip network reinforce such trends. We analyze the reason why each topology has different result. Used approach helps to find optimal topology by changing various on-chip-network topologies on system level*

Keywords: H.264 encoder, Network-on-chip (NoCs), Performance analysis, System level, Topology

1 Introduction

The interests of traditional system-on-chip (SoC) design space exploration focuses on improvement of processing element's (PE's) computational ability. However, recently, as the performance of each single on-chip processing element is improved and the number of components increases, communication architecture between PEs become more important on design area, performance, and energy consumption of overall system. As a result, the on-chip communication architecture becomes one of the major SoC design factor to be considered. As scale grows, the global interconnection brings many problems such as critical on-chip synchronization errors, unforeseeable delays, and high power consumption. In order to alleviate these problems, network-on-chip (NoC) approach has been suggested as the alternative to traditional bus-based communication architecture [1], [4], [5].

On classical bus architecture, multiple on-chip components cannot communicate with each other simultaneously than only one entity can transmit the signal at a time. This characteristic brings the limitation on utilizing the communication channel. However, on NoC architecture, impose the notion of network communication upon interconnection of components inside the chip. Therefore, NoCs improve the utilization of communication. Such efficiency of communication of NoCs can be different from on-chip-network architectures. Interconnection architecture of on-chip components is network topology. Typical network topologies are mesh, crossbar, torus, and point-to-point approach (see Fig. 1).

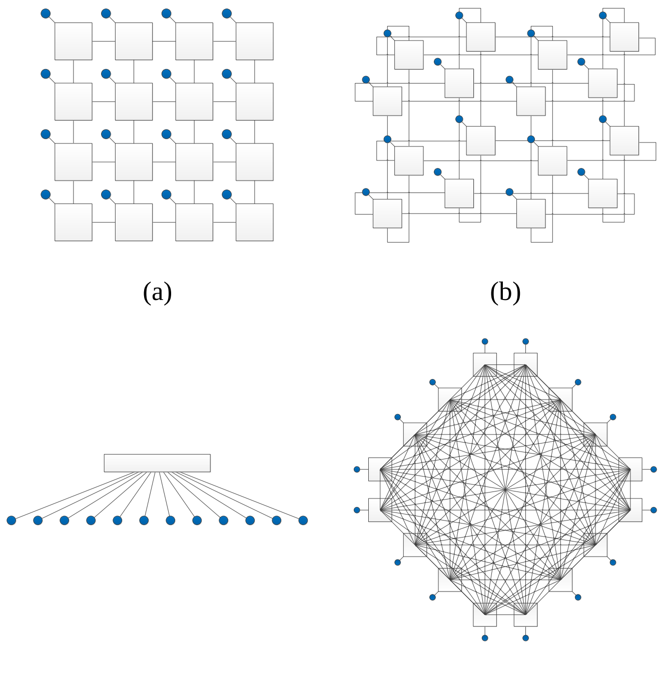


Fig. 1 Types of network-on-chip topologies.
(a) Mesh, (b) Folded torus, (c) Crossbar, (d) Point-to-point.

In this paper, we simulate the same application on NoC systems implemented by different typical network topologies (mesh, crossbar, torus, and point-to-point). Furthermore, we analyze and compare the results from different conditions. Section 2 introduces implemented NoC systems and used target application. Simulation results are analyzed and compared in section 3. Finally, summary will be discussed in section 4.

2 Implemented Systems and Target Applications

In this study, we follow the system-level NoC design flow shown Fig.2. We set the system specification, only changing number of CPUs for confirming the impact of topology in our research. With same spec (same number of CPUs), we implement the system, changing the NoC topology then check the performance. Through that design flow, we can find that which topology shows the best performance.

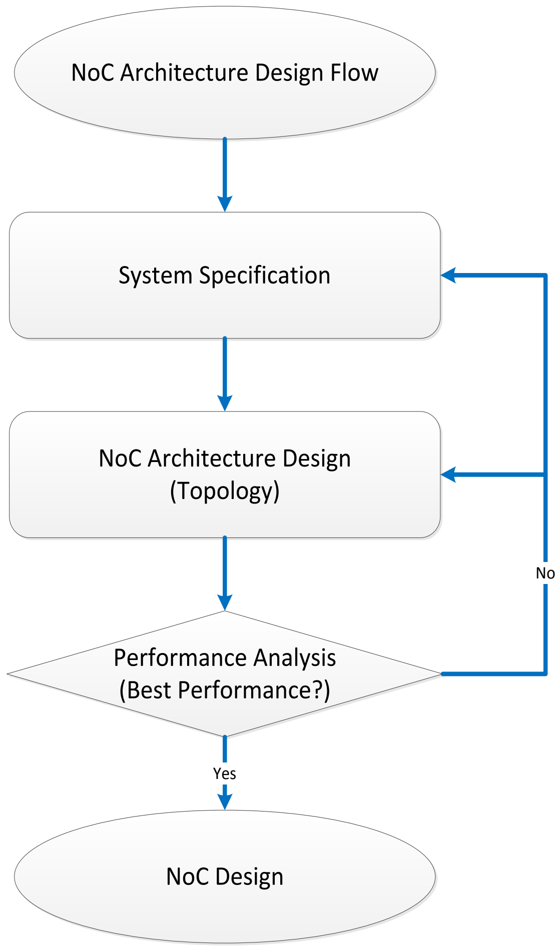

NoC Architecture Design Flow
System Specification
NoC Architecture Design (Topology)
No
Performance Analysis (Best Performance?)
Yes
NoC Design

Fig. 2 System-level NoC design flow

2.1 Implemented System

In this paper, we use the gem5 simulator, full-system simulation framework, to implement NoC systems. The gem5 simulator helps users to simulate on environment as same as real system, compose and arrange inside of chip what they want and also run operating system (OS) on implemented system [2]. This simulator makes users can conduct correct simulations on system level without any low level consideration. We utilize basic topologies that the gem5 simulator offers; mesh, crossbar, torus, and point-to-point to build the 12 types of systems by each case using 4 CPUs, 8 CPUs and 16 CPUs.

Firstly, mesh topology is the most simple and rudimentary NoC topology. This topology requires the number of processing elements (CPUs) to be equal to the number of routers. In this network architecture, the routers, which are connected to the corresponding components, are interconnected literally “mesh” form. Secondly, crossbar topology is network architecture that each on-chip component is connected to every other component through crossbar switch. Multiple components can transmit the packets simultaneously to any of the components as long as the multiple packets do not compete for the same source or same destination. Thirdly, torus topology is similar to mesh topology, but it includes wrap-around links that connects two end nodes on the same row or column. In this paper, our implementation uses folded torus architecture, whose lengths of the links are the same, as the basic torus structure. Lastly, point-to-point topology is the network architecture that every component has dedicated links connecting to every other component [7].

2.2 Target Application

Target application, which is used in our implementation, is x264 offered from a representative shared memory chip-multiprocessors (CMPs) benchmark suite Princeton Application Repository for Shared-Memory Computers (PARSEC). x264 is H.264/AVC (Advanced Video Coding) encoding application which are well used for compressing of high-definition (HD) videos. Pipeline approach is used in parallel algorithm of x264. There are virtual pipeline stages as many as the number of parallelized encoding threads. There are three methods for encoding the compressed frames, I-Frame, P-Frame and B-Frame. Among these, P-Frame has only information about changed parts compared with previous I-Frame or P-Frame. B-Frame is compressed with previous and next frame information using inter prediction. Both P-Frame and B-Frame encoding require that the encoder needs data of images and motion vectors, which come from reference frames. For obtaining and calculating this information, each pipeline stage processes the required tasks. Especially, fast-upward movements encoding process is one of the typical reasons, which brings about performance degradation in practical use. To mitigate these problems, x264 uses parallelization model that has more number of threads than the number of cores to improve its performance [3].

3 Simulation Results and Analysis

With uncompressed 640×360 pixels and 32 frames video from short film “Elephants Dream” [6] as input sets, we perform the encoding process through x264 on implemented twelve systems, which have different topology and number of components. We measured the encoding frame rate and execution time on the basis of frame per second and second respectively through experiments. Simulation results are shown in Table Ⅰ.

TABLE I
AVERAGE FRAME RATE ON DIFFERENT TOPOLOGIES

Num. of CPUs	Threads	Mesh (fps)	P2P (fps)	Crossbar (fps)	Torus (fps)
4	32	29.057	28.677	28.103	28.717
	40	30.930	30.537	29.963	30.653
	64	31.343	30.225	29.883	30.335
8	32	35.453	37.097	35.610	34.097
	40	38.910	40.760	40.470	36.753
	64	39.703	41.453	40.203	37.850
16	32	35.297	36.370	36.200	33.810
	40	38.897	39.823	39.730	37.687
	64	39.800	40.820	40.405	36.620

On systems, which have four CPUs, based on 40 threads, average encoding frame rate of mesh topology is 30.93 frames per seconds (fps), point-to-point topology system can process 30.537 fps, 29.963 fps on crossbar topology, and torus architecture has 30.653 fps. Mesh topology shows the best performance, followed by torus, point-to-point and crossbar on 4-CPUs systems.

Mesh topology system with 8 CPUs shows 39.703 fps based on 64 threads. In same condition, point-to-point topology process with 41.453 fps, crossbar topology has 40.203 fps, torus topology records 37.85 fps. Point-to-point, crossbar, mesh and torus are listed in the order of better performance. With 16 CPUs, point-to-point topology has 39.823 fps to process 40 parallelized threads. Crossbar topology shows 39.730 fps, mesh topology records 38.897, torus has 37.687 fps frame rate.

In Fig.3, crossbar topology shows maximum performance degradation of 4.658% based on mesh topology with 4-CPU, 64 threads. Torus topology shows 3.214%, point-to-point displays 3.565% less performance than mesh topology with same condition.

Fig.4 shows frame rate growth 4.755% on point-to-point topology and 5.543% decrease on torus topology compare with 8-CPUs mesh topology system. Based on 16-CPUs mesh topology system, crossbar topology shows 2.916% performance improvements with 64 threads. On the other hands, 5.213% degradation exists on torus topology (see Fig. 5).

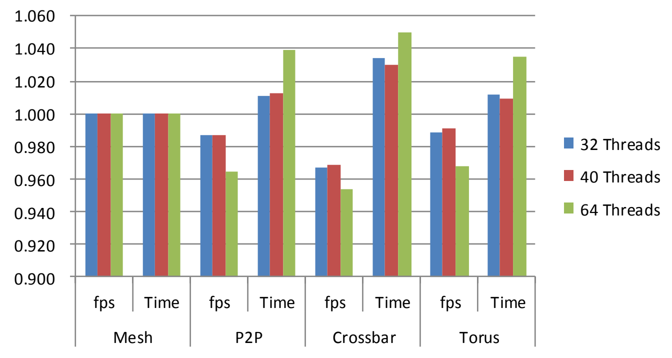


Fig. 3 Performances of each network-on-chip topologies comparison to the mesh topology, which is composed of 4CPUs

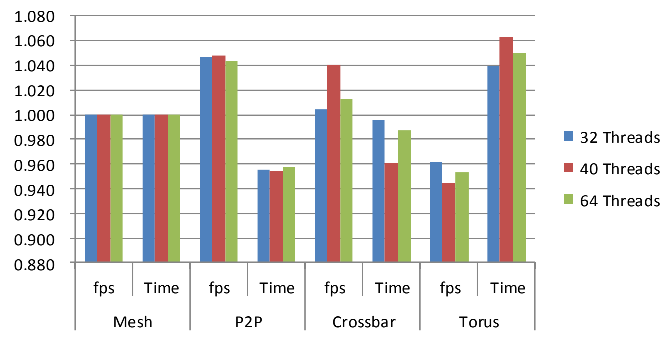


Fig. 4 Performances of each network-on-chip topologies comparison to the mesh topology, which is composed of 8CPUs

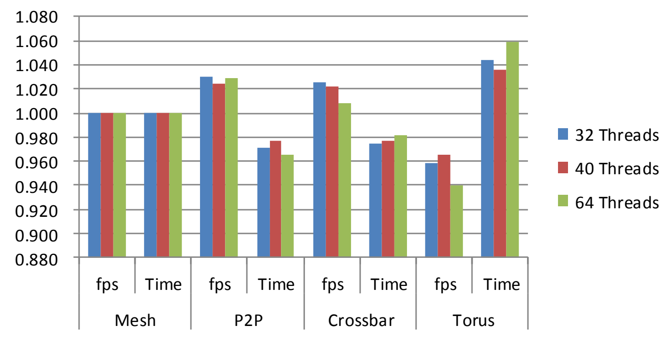


Fig. 5 Performances of each network-on-chip topologies comparison to the mesh topology, which is composed of 16CPUs

TABLE Ⅱ

TOTAL CPUS CYCLES ON DIFFERENT TOPOLOGIES

Num. of CPUs	Threads	Topology	Num. of Total Busy Cycles	Num. of Total Idle Cycles	Num. of Total Cycles
8	40	Mesh	7.5884E+09	5.4543E+09	1.3043E+10
		Torus	7.9932E+09	5.6779E+09	1.3671E+10
		P2P	8.3664E+09	3.7730E+09	1.2139E+10
		Crossbar	7.6613E+09	5.0784E+09	1.2740E+10
	64	Mesh	7.5382E+09	5.3359E+09	1.2874E+10
		Torus	7.8214E+09	5.4792E+09	1.3301E+10
		P2P	7.3946E+09	4.9550E+09	1.2350E+10
		Crossbar	7.6133E+09	5.0534E+09	1.2667E+10
16	40	Mesh	7.8071E+09	1.8700E+10	2.6507E+10
		Torus	8.2336E+09	1.9415E+10	2.7649E+10
		P2P	7.5563E+09	1.8570E+10	2.6126E+10
		Crossbar	7.6908E+09	1.8632E+10	2.6323E+10
	64	Mesh	7.7949E+09	1.8720E+10	2.6514E+10
		Torus	8.2964E+09	1.8357E+10	2.6654E+10
		P2P	7.5258E+09	1.7543E+10	2.5069E+10
		Crossbar	7.6649E+09	1.8181E+10	2.5845E+10

Even though we perform simulation with same application x264 and identical components, we can confirm the fact that results present a great contrast from topology to topology. It is due to application mapping can be carried out by on-chip-network topology differently.

This result also shows a tendency that 8-CPUs and 16-CPUs systems have the same performance order, point-to-point, crossbar, mesh and torus. Differently, at 4-CPUs system, mesh, torus, point-to-point and crossbar topology records better performance in the order named, compare to 8-CPUs and 16-CPUs systems.

The number of CPUs growing means the components of topology are increasing. More number of topology components causes that packet data path have more junctions, routers or switches, to get to the destination nodes. The number of routers that a packet has to pass to reach its destination is different from topology to topology. As for topology, it affects communication delays first, and consequently that effect has influence on performance of application.

At 4-CPUs system, there are four CPUs, which are composed of network and generate small path diversity. However, 8-CPUs system or 16-CPUs system generate many path diversities, many intermediate routers and are influenced by topologies. Point-to-point topology is connecting all the nodes by dedicated communication paths. Therefore, if one node wants to send data packet to another node, there is no intermediate router. Because of that reason, point-to-point topology records the fastest performance on systems with 8 CPUs and 16 CPUs. Crossbar switch sets path up between any two processing elements, and it allows another concurrent connection among other arbitrary input and output nodes except using same source node or same destination node. This characteristic of crossbar topology makes the system performs secondly. Mesh and torus topology have more intermediate routers than above topologies when they connect nodes and all the links have same length. Both of them have similar network structure and only difference is that torus has wrap-around path. This difference makes performance difference. In addition, folded torus network has twice-longer links than mesh topology. Because of that, torus topology system shows lower performance than system, which has mesh topology.

For analyzing the effect of topologies, we use the CPUs total cycles. CPUs total cycles reflect communication latency in a roundabout way, which are caused by intermediate switches of topologies. Total busy cycles, total idle cycles and total cycles of all the CPUs on the system are shown in Table Ⅱ. On 8-CPUs system with 40 threads, point-to-point topology shows the lowest total cycles, in other words, it has low communication latency. Compare to the point-to-point topology system, crossbar system has 4.945% more cycles, 7.441% more for mesh topology and 12.617% more for torus system. In addition, crossbar topology system with 64 threads on 16 CPUs runs 3.097% more than point-to-point system. Mesh and torus system operates 5.766% and 6.322% more respectively.

To see the effects of topology better, we use average deviation rate (ADR). ADR factor shows the average amount of communication latency deviation of each topology compare to the topology that has the best performance and the lowest latency, in this study, point-to-point topology. The formula for ADR calculation is shown in formula (1).

$$\text{ADR} = \frac{\text{Total CPU Cycles of each topology} - \text{Best topology's Total CPU Cycles}}{\text{Best topology's Total CPU Cycles}} \quad (1)$$

On average, crossbar topology shows 2.8405%, Mesh shows 4.7276%, Torus shows 8.1168% more latency than point-to-point(see Table Ⅲ).

TABLE Ⅲ

TOTAL CPUS CYCLES AND AVERAGE DEVIATION RATE ON DIFFERENT TOPOLOGIES

Num. of CPUs	Threads	P2P (cycles)	Crossbar (cycles)	Mesh (cycles)	Torus (cycles)
8	40	1.21E+10	1.27E+10	1.30E+10	1.37E+10
	64	1.24E+10	1.27E+10	1.29E+10	1.33E+10
16	40	2.61E+10	2.63E+10	2.65E+10	2.76E+10
	64	2.51E+10	2.58E+10	2.65E+10	2.67E+10
ADR		0%	2.84%	4.73%	8.12%

These results support our trends analysis that on more number of CPUs, point-to-point has the lowest communication latency because of dedicated communication links. Crossbar switch makes the system has lower latency than mesh, torus topology system. Mesh and torus have more intermediate routers on data paths than point-to-point and crossbar topologies. Especially, torus network is connected with longer link distance, so it has the highest communication latency, which causes degradation performance.

Additionally, from the CPUs cycles simulation result, we can also find tendency related to the number of CPUs. When the number of CPUs increases two times from eight to sixteen, total idle cycles also increase irrespective of topologies, while total busy cycles does not shows much change. Total busy cycles changes in the range of 0.903~1.061 times. The number of total idle cycles of torus system with 40 threads records the least increase by 3.419 times. Increasing by 4.922 times on point-to-point topology system with 40 threads is the highest growth in total idle cycles. As the number of CPUs is growing, total length of the systems' links that connect components and the degree of distribution of application's task on to multiple CPUs also increase. These reasons makes more idle cycles.

4 Conclusions

This paper describes comparative analysis results of system performance by network-on-chip topologies on system level. When design the system, we conduct simulation with the gem5 full-system simulator to build a system, which has same condition like actual system. Before constructing real system, using our approach, we can find the fact that performance differences exist when running the x264, H.264/AVC encoding application, by network-on-chip topologies in advance. A tendency can be found from our simulation result that 8-CPUs and 16-CPUs system have the best performance on point-to-point topology followed by crossbar, mesh and torus topologies. We use the CPUs total cycles that reflect the communication latency indirectly for analyzing the effect of topologies. We can see the same trend for latency that came from topology. This result can be obtained because of topologies' characteristics. Especially, more CPUs stands for more components of on-chip network topology that is routers. Types of topologies affect the number of routers on the paths related to latency and performance. Therefore, point-to-point topology, which has dedicated path among all CPUs without intermediate routers, has low communication latency and best performance. Crossbar, mesh and torus have more intermediate routers than point-to-point. Our study finds the effect of topology on performance and analyzes the reason why each topology has a different result by using total CPU cycles. Topologies affect on decision process which part of application mapped into which processing elements. Moreover, the used approach in this study helps to find optimal topology by changing various on-chip-network topologies on system level.

5 References

[1] Benini, L, and De Micheli, G. "Networks on chips: A new SoC paradigm." IEEE Computer, vol. 35, pp. 70-78, Jan. 2002.

[2] Binkert, N. et al. "The gem5 simulator." ACM SIGARCH Computer Architecture News, vol. 39, pp. 1-7, May 2011.

[3] Bienia, C. et al. "The PARSEC benchmark suite: characterization and architectural implications." Proc. Parallel architectures and compilation techniques. ACM, 2008.

[4] Marculescu, R. et al. "Outstanding research problems in NoC design: system, microarchitecture, and circuit perspectives." Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions, vol. 28, pp. 3-21, Jan. 2009.

[5] Ho, Ron, Kenneth W. Mai, and Mark A. Horowitz. "The future of wires." Proceedings of the IEEE, 2001, pp. 490-504.

[6] Elephants Dream. http://www.elephantsdream.org/, 2006.

[7] Pasricha, Sudeep, and Nikil Dutt. On-chip communication architectures: system on chip interconnect. Morgan Kaufmann, 2010, pp. 443-448.

On the Optimization of HDA* for Multicore Machines. Performance Analysis.

Victoria Sanz[1], Armando De Giusti[2], Marcelo Naiouf
III- LIDI, School of Computer Sciences, UNLP, La Plata, Buenos Aires, Argentina

Abstract – *Combinatorial optimization problems are interesting due to their complexity and applications, particularly in robotics.*
This paper deals with a parallel algorithm suitable for shared memory architectures, based on the HDA algorithm (Hash Distributed A*), which allows finding solutions to combinatorial optimization problems. The implementation was carried out using the shared memory programming tools provided by the Pthreads library, the Jemalloc memory allocator and taking the N^2-1 Puzzle as study case.*
The experimental work focuses on analyzing the speedup and efficiency achieved by the parallel algorithm when running on a computer with multi-core processors, for different instances of the problem and varying the amount of threads/cores used. Finally, the scalability obtained with increasing workload and number of threads/cores used is analyzed.

Keywords: Parallel Heuristic Search; HDA*; Multicore; Combinatorial Problems; Scalability.

1 Introduction

In the area of Artificial Intelligence, heuristic search algorithms are used as the basis to solve combinatorial optimization problems that require a sequence of actions that minimize a goal function and allow transforming an initial configuration (which represents the problem to be solved) into a final configuration (which represents the solution).

One of the most used search algorithms for that purpose is known as *Best First Search* (*BFS*) [1], which browses the graph that represents the state space of the problem using a cost function $\hat{f}$ to value the nodes, which is in part composed of some heuristic information, that will guide the search faster to the solution and will reduce the nodes to be considered. The algorithm is different from the conventional methods because the graph is implicit and generated dynamically, i.e. nodes are created as the search progresses. During the process, it keeps two data structures: one for the unexplored nodes ordered by the function $\hat{f}$ (*open list*), and the other for the already explored nodes (*closed list*) used to avoid processing the same state repeatedly. In each iteration, the most promising node available on the open list is removed (according to function $\hat{f}$), it is included on the closed list and legal actions are applied to it to generate successor nodes which will be added to the open list under certain conditions. The search continues until a node that represents the solution is removed from the open list.

The A* algorithm [2] is one of the most commonly used *BFS* variants because it guarantees finding optimal cost solutions. To that end, the cost function $\hat{f}$ contains known cost information of the path from the initial node to the current node and heuristic information to estimate the unknown cost of the path from the current node to the solution node, which can never overestimate the actual cost; in this way, the search is guided to firstly process the most promising paths.

On the other hand, over the last years the development of parallel heuristic search algorithms has been promoted because the high requirement of computing power and memory, as a consequence of the exponential or factorial graph growth, makes its resolution on a single-core processor difficult. Moreover, it is common to find multi-core machines today, so the sequential applications should be adapted to take advantage of the computing power that this architecture provides.

So far, different authors have presented several techniques to parallelize *BFS* algorithms, which vary according to how they manipulate the open and closed list and in the load balancing strategy used among processors during the execution. The chosen technique will depend on the architecture and the problem to solve [3].

On a shared memory architecture, the simplest strategy is to keep only one open list and only one closed list shared by all the threads (*centralized strategy*). This implies a thread synchronization process to ensure data structure consistency, which will limit performance [3][4]. Although the open and closed lists can be implemented through data structures that allow concurrent access to different portions in order to reduce resource contention, several authors have shown that this technique will only bring improvements for problems with high heuristic computation time [3], and especially current studies have shown that it does not get a competitive performance on multi-core machines [5].

In order to solve the previous problem, each process/thread is equipped with its own local open and closed lists (*decentralized strategy*) and performs a quasi-independent search. This strategy is suitable either for shared memory or distributed memory architectures. However, communication among the processes/threads is needed due to the following reasons:

- As only one process/thread has the initial node on its open list at the beginning and the graph is generated at run time, the workload should be distributed dynamically.
- The nodes located on the processor's open list might not be the global best ones, so it will be necessary to equalize the nodes quality between processors.
- Duplicate nodes (nodes representing the same state) can be generated by different processes/threads. If the duplicate

[1] Fellow, CONICET.
[2] Principal Researcher, CONICET.

detection procedure is only performed by the process/thread which has generated the node and/or by that which has received the node owing to load balancing, the detection and pruning of duplicate nodes will be *partial* because another process/thread may have a node representing the same state on its open or closed list. However, if *absolute* detection and pruning is required, strategies that assign each state to a particular processor will be needed.

- The termination criterion should be modified, as there are multiple inconsistent open lists and, as a consequence of dynamic load balancing, there may be some graph nodes that are being communicated between processes/threads.
- The costs of the partial solutions found should be communicated in order to use them to prune the paths that lead to suboptimal cost solutions.

In this sense, the HDA* algorithm (*Hash Distributed A**) [6] parallelizes A* using the *decentralized strategy* and it applies *Zobrist´s hash function* to assign each state to a unique process; in this way, when a process generates a node, the owner process can be identified and the node is transferred to it. This mechanism allows balancing the workload, leveling node quality, and pruning duplicates in an *absolute* way, as the nodes representing a same state are always sent to the same process. The algorithm was implemented using the *MPI* message passing library and asynchronous communication, so the algorithm can be executed either on distributed or shared memory architectures.

On the other hand, the research carried out by [5] presents an adaptation of the HDA* algorithm developed using the shared memory programming tools provided by the *Pthreads* library; in this way, it is possible to eliminate some inefficiencies that arise when the original HDA* algorithm is run on a shared memory machine. The Jemalloc library [7] is used to avoid performance degradation due to contention in the access to the data structures managed by the dynamic memory allocator, caused by the frequent alloc/free operations. Algorithm performance is analyzed on a multicore machine. Moreover, a technique to create a state space abstraction that allows assigning state blocks to the threads, instead of individual states as it occurs with *Zobrist´s hash function*, is included in the algorithm. Then, the *PBNF* algorithm that allows threads to work during synchronization free periods is presented. The experimental work is done in part considering 250 easy instances of the 15-Puzzle, using the Sum of the Manhattan Distances heuristic, and an analysis of the speedup obtained as the architecture scales is carried out. Although a better performance is obtained with the *PBNF* algorithm, the algorithm is complex and does not use the same approach as the serial A*, so a *superlinear speedup* is obtained in some cases.

A common problem that causes performance degradation in multi-threaded applications, which frequently perform allocation and deallocation operations, is the producer-consumer relation that arises due to alloc-free operations carried out by different threads, which creates a need for synchronization to keep the consistency of the structures assigned to each thread by the memory allocator. In order to improve this, it is suggested to incorporate a pool of pointers to node in every thread (*Memory Pool*) to prevent thread A from freeing memory that is allocated by thread B; instead, thread A will store those pointers for future reuse.

Based on the above, the HDA* algorithm is still interesting due to its simplicity. The focus of this paper is the incorporation of techniques to optimize the HDA* algorithm for its execution on multicore, which may lead to a better performance, and to carry out a scalability analysis when the workload and the amount of processors are increased.

2 Contribution

This paper presents a parallel algorithm suitable for shared memory architectures, based on the HDA* algorithm, which allows finding optimal solutions to combinatorial optimization problems, in this case to the N^2-1 Puzzle. In this sense, the algorithm is similar to the one proposed in [5] but with the following differences: it incorporates an algorithm to detect termination in a decentralized way, which is an adaptation of the algorithm proposed by *Dijkstra and Safra* [8] [9]; threads accumulate a customizable quantity of nodes addressed to another thread before attempting their transfer, i.e. there are no transfers after each node generation; and a technique called *Memory Pool* is used to avoid performance degradation caused by alloc-free operations in a producer-consumer relation among different threads.

The contributions are:

- Carrying out experimental work running the HDA* parallel algorithm proposed, suitable for shared memory, on a multicore processor machine, using different initial configurations of the problem and varying the amount of threads/cores used, analyzing the performance obtained (*speedup*, *efficiency*) in each case.
- Carrying out a comparison between the performance obtained by the parallel algorithm when active waiting or passive waiting is used while the thread is idle.
- Documenting the benefits obtained when using the *Memory Pool* technique.
- Carrying out a scalability analysis of the algorithm when the workload and amount of threads/cores used are increased.

3 Characterization of the N^2-1 Puzzle

The N^2-1 Puzzle problem consists in N^2-1 pieces numbered from 1 to N^2-1 placed on an N^2 sized board [10]. Each square of the board contains one piece, so there is only one empty square.

- A legal movement implies moving the empty square to an adjacent position, either horizontally or vertically, by moving the piece that was in the newly emptied square to the previous position of the empty square.
- The objective of the puzzle is applying legal movements until the initial board becomes the selected final board. The solution to the problem should be the one that

minimizes the number of movements required to achieve the final configuration from the initial given configuration.

1.1. Heuristics

Heuristic search algorithms use information about the problem to guide the search process, so they value the nodes based on the application of a *heuristic function*. Thus, they process first the node that looks more promising. The heuristic value of a node is an estimate and indicates how close it is to the solution node.

A more polished heuristic will carry out estimates that are closer to the real cost; therefore, the algorithms that use it will need to process less nodes [1].

The heuristic used by the algorithms presented for the resolution of the Puzzle problem is a variation of the sum of the Manhattan distance of the pieces with the addition of linear conflict detection among pieces, the detection of the last movements applied, and an analysis of corner pieces. The definition can be found in [11].

4 Sequential A* algorithm

The A* algorithm [2] is a variation of the *Best First Search* technique where each node *n* is valuated in accordance to the cost of reaching it from the root of the search tree $\hat{g}(n)$ and a heuristic that estimates the cost to go from *n* to a solution node $\hat{h}(n)$. Thus, the cost function will be $\hat{f}(n) = \hat{g}(n) + \hat{h}(n)$. If the heuristic is *admissible* (i.e., it never overestimates the real cost), the algorithm will always find an optimal solution.

The algorithm keeps a list of unexplored nodes (*open list*) ordered by the value of function $\hat{f}$, and another list of already explored nodes (*closed list*) used to avoid loops in the search graph. Initially, the open list contains only one element, the initial node, and the closed list is empty.

In each step, the node with the lowest $\hat{f}$ value (the most promising node) is removed from the open list and examined. If the node is the solution, the algorithm ends. Otherwise, the node is expanded (generating the children nodes by applying legal movements) and added to the closed list. Each successor node is added to the open list if it does not appear on either list, or if it does but its cost value improves that of the previous node (this verification is known as *duplicate detection*).

Once the node that represents the final state has been found, the sequence of actions taken on the optimal path can be obtained by following the sequence of pointers to each parent node.

5 HDA* algorithm for shared memory architectures

The HDA* algorithm suitable for shared memory architectures proposed in [5] is based on the following:

- Each thread has its own open and closed lists.
- Each thread has an input queue known globally where the rest of the threads will deposit nodes that must be processed by this thread. The input queue must be protected by a lock to keep its consistency.
- Each thread has a local output queue for each peer thread, which does not need to be protected since it will be for thread's own use to avoid obstructions.
- When a thread t_i generates a node that belongs to another thread t_j, it must be communicated by adding it to t_j's input queue at some point. In order to do this, the thread tries to take the lock associated to t_j's input queue. When the lock is obtained immediately, node transfer is done by copying the pointer, and then the lock is released (this enables subsequent access to the queue by another thread). Otherwise, the pointer is added to the local output queue for t_j (there is no waiting time associated with this operation).
- After thread t_i carries out a certain number of node expansions from its open list:

- For each non-empty local *output queue*, the thread tries to communicate the stored nodes on it to its owner thread. In order to do this, the thread tries to take the lock associated to the input queue of the corresponding thread. If the lock is obtained, all the pointers to node are transferred, leaving the local output queue empty. Otherwise, it is not forced to wait.
- The thread tries to consume the nodes left by other threads on its own input queue. To do this, the thread must take the lock but it is only forced to wait if its open list is empty (in this case, it does not have any nodes to keep on working).

Fig. 1 shows the communication scheme of HDA* algorithm for shared memory. Here the thread's main local structures can be seen (open list, closed list, output queues) and also the global input queues. We can observe that thread 0 and thread 3 have generated a node that corresponds to thread 2; both of them attempt to take the lock associated to target thread's input queue. On the one hand, thread 3 gets the lock immediately, copies the pointer and releases the lock. On the other hand, thread 2 does not get the lock immediately, so it adds the pointer to its local output queue for the target thread.

The implementation was carried out with the tools provided by the Pthreads Library and Jemalloc, the dynamic memory allocator. The allocation of states to threads was done through the *Zobrist Function*. The input and output queues were implemented as a dynamic array that contain pointers to node.

6 Implementations

The implementations of the following algorithms were carried out in C Language. The compilation was done through Gcc, phase in which the memory allocator that is going to be used can be selected (in this case, it was *ptmalloc* [12] or *Jemalloc* [7]).

6.1 Sequential A*

The selected structure to implement the open list is a *MinHeap* [13] whose content is indexed by an *Extensible*

Hash Table [14]. This structure allows nodes to be ordered according to the $\hat{f}$ function, so the operations of inserting a node, removing the node with the lowest $\hat{f}$ value and decreasing the priority of a node can be carried out in logarithmic order; at the same time, it enables carrying out searches to determine the existence of a node that represents a particular state in constant order. Additionally, an *Extensible Hash Table*[14] was used to implement the closed list, which allows the operations of insertion/ removal of a node and searching to determine the existence of a node that represents a particular state occur in constant order.

The keys associated to the elements (nodes) stored in the *Hash Tables* are obtained by calculating the *Zobrist Function* [15] over the representation of the state. The *Zobrist Table* that was used is loaded from file and it will be the same for all the runs and instances of the problem selected as a case of study. The key will be represented with a 64-bit integer, which leads the function to assign the same key to different states of the search space (this happens because the number of possible problem states can be much higher than 2^{64}). This case is not frequent during the run of the search algorithm.

The kind of heuristic functions used is that which calculates the estimation of the cost directly, taking as input the representation of the state. Consequently, the heuristic is problem-dependent and can be selected before the compilation phase; this enables experimenting with different heuristic functions and analyzing the performance obtained.

6.2 HDA* for shared memory

A version of the HDA* algorithm suitable for shared memory similar to the one studied in Section V was implemented. Threads and synchronization mechanisms provided by the *Pthread* library were used. The assignment of states to threads was carried out through the *Zobrist Function*.

Each *thread* will perform an A* search locally, keeping its local open and closed lists. The node communication strategy is based on the use of input and output queues.

To avoid making an only thread detect the termination state by checking the state of the other threads and the state of their input queues, an adaptation of the *Dijkstra and Safra's* termination detection algorithm was carried out, allowing all the threads to cooperate with such purpose. Each thread keeps a *state* (or *color)* and a *counter* of sent and received nodes - instead of the number of "communications" that were done[1]. The *termination token* will be represented with a shared variable with the following information: a counter of nodes in transit, a *state* (or *color*), and the identifier of the thread that owns the token at the moment; the data that corresponds to the token are not protected since only one thread will be able to modify them at a given point in time. The end of the computation will be communicated through a shared variable *end*.

[1] The input queues do not count how many times a deposit was done over them, but the total amount of nodes that they store (logical dimension). Because of that, the amount of nodes in transit is calculated instead of the amount of "communications" or "deposits" that have not been received yet. This is a modification of the *Safra and Dijsksträ* algorithm.

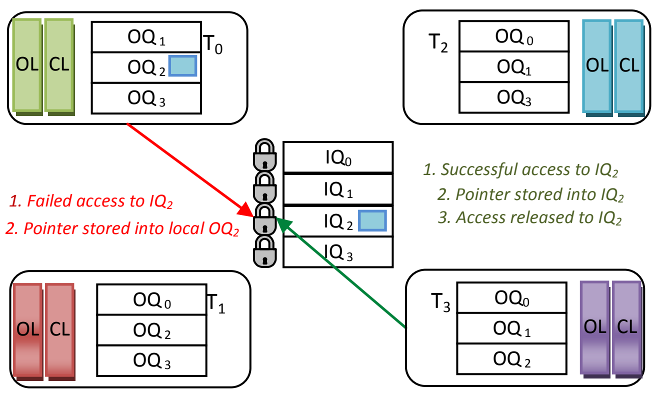

OL
CL
OQ_1
OQ_2
OQ_3
T_0
T_2
OQ_0
OQ_1
OQ_3
OL
CL
IQ_0
IQ_1
IQ_2
IQ_3
1. Failed access to IQ_2
2. Pointer stored into local OQ_2
1. Successful access to IQ_2
2. Pointer stored into IQ_2
3. Access released to IQ_2
OL
CL
OQ_0
OQ_2
OQ_3
T_1
T_3
OQ_0
OQ_1
OQ_2
OL
CL

Fig. 1. Communication scheme of HDA* algorithm suitable for shared memory

With the aim of making possible the pruning of nodes that will lead to suboptimal solutions, the threads share a pointer to the best solution found so far by all the threads (*best_solution*) and its cost (*best_solution_cost*). Both variables must be protected, since two threads can find two different solutions and try to update these values at the same time.

The code that all the threads will run is identical, only thread 0 will be in charge of the additional tasks of generating the initial node and adding it to the input queue of its owner thread, initializing the common structures, detecting the termination state, and recovering from the shared memory the steps sequence that represents the solution to the problem once computation is finished.

Each thread will carry out a series of iterations until it detects the end of the computation (through a change in the value of the variable *end*). In each iteration, the following phases are performed:

- *Phase of node consumption from input* queue: the thread checks whether its own input queue is not empty. In that case, it tries to take the lock associated to the queue. When it obtains the access immediately, it takes all the pointers to nodes that were deposited on the queue, releases the lock, and then for each node whose cost is lower than *best_solution_cost* the thread performs the duplicate detection process adding them to the open list as appropriate.
- *Processing phase*: the thread processes at least *LNPI* (*Limit of Nodes per Iteration*) nodes from its open list. When the thread removes a node, it verifies if its cost is at least *best_solution_cost*. If it is so, the thread empties the open list since the nodes on it will lead to suboptimal solutions. Otherwise, it checks if the node represents the solution and in that case it updates *best_solution* and *best_solution_cost,* after having taken the lock that protects them and having consulted again if the node cost is less than *best_solution_cost*[2]. When the removed node is

[2] This is necessary because two threads can find two solutions with different cost at the same time. If a solution had not been found yet or if the two solutions that have just been found improve the current partial solution, when the threads try to obtain the lock to update the shared data, the thread with the best solution could update the data first, and then the second thread could

not the solution, it is inserted into the closed list, it is expanded (in this way successors are generated), and then for each successor the *Zobrist Function* is calculated so as to know which thread has to process it. When the node belongs to the thread that generated it, the thread carries out the duplicate detection and adds it on the open list as appropriate. Otherwise, the thread places the node in the local output queue for the target thread; when the amount of stored nodes on the output queue is higher than the limit *LNPT* (*Limit of Nodes per Transference*), the thread tries to take the lock of the target thread's input queue and, if it obtains the lock immediately, it transfers the stored nodes leaving the output queue empty[3]. A node transfer simply means a pointer copy. When a thread is the first one that deposits nodes on the input queue of another thread (i.e., at that moment the input queue was empty) it must inform the action, just in case the other thread was idle waiting for work.

- *Idle phase*: after the processing phase, if the thread stands idle because its open list is empty, it will send nodes stored on each non-empty output queue, and it will wait for any of the following events:

- *End of calculation*: thread 0 detected the *termination state* and it changed the value of the *end* variable, so that this state can be known.
- *Termination token arrival*: the thread must update the shared variables that correspond to the token, based on the termination algorithm, and pass it to the following thread, which means that the thread must change the value of the *token owner* field (informing the successor thread that it is the new owner). On the other hand, thread 0 verifies if the termination conditions are given. If this is so, the value of the *end* variable changes. Otherwise, it starts a new round to detect termination.
- *Work deposit on its own input* queue: the thread must obtain the lock associated to its input queue, it must take all the pointers to nodes that were deposited on the queue (leaving the input queue empty), and release the lock. For each node whose cost is lower than *best_solution_cost* the duplicate detection is carried out, adding the node to the open list as appropriate.

The termination detection algorithm involves updating the variable *state* (or *color*) and the *counter* of the thread every time it adds nodes to the input queue of another thread or every time it removes nodes from its own input queue, either increasing or decreasing the local counter of nodes that were deposited and received respectively.

To resolve performance degradation when an alloc-free relation between threads happens, a *pool* of pointers to node (*Memory Pool*) was incorporated to each thread, where the pointers to node that the thread wishes to "set free" for a further use are stored. This technique prevents access by thread A to the structures assigned to another thread B by the dynamic memory allocator, when the former wants to "free" a pointer allocated by the latter, situation that would produce contention.

Finally, the possibility to compile the algorithm to carry out a wait in a *passive* or *active* way when the thread stands idle was incorporated.

7 Experimental results

For the experimental work, a machine with two Intel ® Xeon ® E5620 [16] processors was used. Each processor has *four* 2.4 Ghz physical *cores*. Each *core* has two L1 caches of 64 KB for data and instructions respectively and one L2 cache of 256 KB. At the same time, all processor cores share a L3 cache of 12 MB. Each processor has a memory controller, therefore, the machine's memory design is NUMA and it uses a *QuickPath Interconnect (QPI)* interconnection of 5.86 GT/s. The machine has 32 GB of RAM, DDR3 1066 Mhz.

The tests were carried out taking into account the 100 initial configurations of 15-Puzzle used by [17], numbered from 1 to 100. Ten of the configurations with more steps for their solution [18] were also taken into account in the parallel algorithm scalability analysis, since for some of them resolution time is considerable. These were numbered from 101 to 110.

7.1 Sequential A*

7.1.1 Effect in the use of Jemalloc

The sequential algorithm was run with the 100 initial configurations and the final configuration suggested by [17] using the heuristic function presented in Section II.A and varying the dynamic memory allocator between *ptmalloc* and *Jemalloc*. *Jemalloc* was configured to work with 256 arenas, as this configuration will be used during the parallel algorithm tests.

For each initial configuration, 10 runs were performed and the runtime in seconds for each test was calculated. From the results, it can be observed that the average runtime of each configuration obtained from the samples that use *Jemalloc* presents a reduction ranging between 0.9% and 15.8% with respect to the average runtime for the same configurations using *ptmalloc*.

As it has been proved that when *Jemalloc* (configured to work with 256 arenas) is used with sequential A*, algorithm performance improves, the above mentioned allocator will be used in the tests from now on.

7.1.2 Effects in the use of Memory Pool technique

The sequential algorithm was run with the 100 initial configurations and the final configuration proposed by [17], using the heuristic function presented in Section II.A, *Jemalloc* (configured to use 256 arenas) and the *pool* of pointers to node "*Memory Pool*". For each initial configuration, 10 tests were run. Then, the average runtime in seconds obtained for each configuration was compared with

obtain the lock to carry out its update. If the condition about the cost is not verified again, the second thread could make effective the update and a suboptimal solution would be stored.

[3] This is different to the version proposed by Burns in which after each node generation that belongs to another thread t_j, the thread tries to take the lock associated to t_j's input queue. Moreover, here when the lock is obtained all the nodes stored on the output queue for the target thread are communicated, this is another difference with Burns' version.

the results presented in the previous section that do not use the *"Memory Pool"* technique.

From the comparison, it can be observed that the "*Memory Pool" technique* does not bring any advantage for the sequential application: 17 configurations suffered an increment of their average runtime of about 2% and 9%; 15 configurations increased their average runtime between 1% and 2%; 43 configurations achieved a modest increase in its average runtime which goes between 0% and 1%; finally, 25 configurations reduced their average runtime between 0% and 6.45%.

Generally, relevant variations in performance achieved by instances with a significant runtime are not observed. Thus, the sequential results obtained without using the "*Memory Pool*" technique will be used for the performance analysis of the parallel algorithm.

7.2 HDA* for shared memory

The HDA* parallel algorithm is nondeterministic, i.e. when different experimental samples are taken for the same initial/ final configuration and the same parameters, the results obtained by the algorithm may be different. That is possible because an initial configuration can have multiple optimal solutions and, as threads distribute the space of states dynamically among themselves, the nodes processed by a thread will vary depending on how asynchronous events occur in the system.

In the tests, affinity was used to allocate each thread to an exclusive *core* using the function *sched_setaffinity*() [19]. In those tests with 4 threads 1 pair of threads was allocated to each machine processor, and in those tests with 8 threads 1 thread was allocated to each physical *core* of the machine.

The selected initial configurations are those used in Section VII.A whose sequential runtime is of at least 5 seconds[4]. For performance analysis, the configurations numbered from 101 to 106 were also taken into consideration; sequential and parallel tests for configurations 107, 108, 109 and 110 exhausted available RAM memory and were therefore aborted.

The Jemalloc memory allocator, configured to work with 256 arenas, and the heuristic function presented in Section II.A were used. For each initial configuration and each parameter group, 100 samples were obtained. The parameters are: the amount of cores/threads, whose values vary between 4 and 8; *LNPI* between 1, 5, 50 and 500; *LNPT* was set in 26 nodes. Then, the *average runtimes* resulting from the 100 runs for the same configuration and set of parameters, which will be called *average sample,* were obtained.

7.2.1 Passive waiting vs. active waiting

Two test sets were run using *active waiting* and *passive waiting* respectively, *LNPT was* limited to 26 and the *Memory Pool* technique was used.

Average runtimes brought by the test sets do not show an apparent benefit for any particular waiting technique. This may be due to algorithm asynchronism, as most times threads perform attempts to take locks and in case they do not get it they keep on working. Additionally, each thread is run on an exclusive core. Therefore, either waiting actively or resorting to the Operating System to perform a passive wait does not cause drastic changes in performance.

7.2.2 Effects in the use of Memory Pool technique

Two test sets were run including the *Memory Pool* technique or not; *LNPT* was limited to 26, and *active waiting was used*.

Average runtimes of the test set that uses *Memory Pool* reduced the average runtimes of the test set that does not use that technique between 4.5% and 12.82%. Generally, the reduction in the average runtime for the samples with 4 threads is between 4.5% and 8.64%, while the reduction in tests with 8 threads is between 6.43% and 12.82%.

Therefore, the advantage of this technique for reducing contention in the access to structures assigned to each thread by the dynamic memory allocator, in cases with an existing producer-consumer relation between threads by alloc-free operations, is shown.

7.2.3 Performance Analysis

The experimental tests discussed in the previous section, which optimized the results, were considered to assess parallel algorithm performance[5]. Moreover, tests were carried out for configurations 101 to 106 following the same strategy. Then, for each configuration and number of threads, the *average sample* that minimizes average runtime, i.e. the sample whose *LNPI* parameter value optimizes performance, was selected.

To assess algorithm scalability, the average samples selected for each configuration were organized according to their sequential workload (sequential time). In this sense, escalating the problem means increasing the number of processed or generated nodes. On the other hand, the architecture is escalated by increasing the number of cores used to solve the problem.

Fig. 2 shows the Speedup obtained by the average sample selected for each configuration using 4 cores and 8 cores, while Fig. 3 shows the Efficiency obtained. For tests with 4 cores, the Speedup obtained varies from 2.95 to 4.01, while Efficiency ranges from 0.73 to 1.0034. Tests with 8 cores show a Speedup between 5.14 and 8.15, and Efficiency between 0.64 and 1.018.

Both average samples that obtained a superlinear Speedup present a negative Search Overhead[6] (-2.92 for the average sample with higher Speedup with 4 cores and -9.86 for the average sample with higher Speedup with 8 cores). Therefore, the parallel algorithm processes fewer nodes than the sequential algorithm. This situation is possible for this class of algorithms due to the causes explained in [20].

[4] Configurations are as follows: 3, 15, 17, 21, 26, 32, 33, 49, 53, 56, 59, 60, 66, 82, 88, 100

[5] The tests of interest are those that use active waiting, limiting LNPT in 26 and using the *Memory Pool* technique.

[6] The Search Overhead represents the percentage of increment in the number of nodes expanded by the parallel algorithm against the sequential algorithm and it is calculated with the formula 100x(NP/NS -1), where NP= number of nodes processed by the parallel algorithm and NS = number of nodes processed by the sequential algorithm.

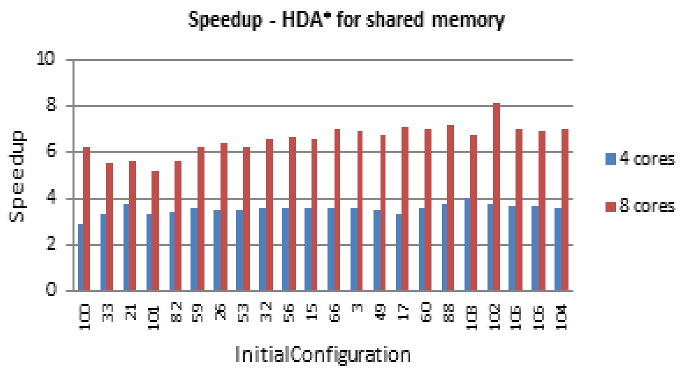


Fig. 2. Speedup achieved by the HDA* algorithm for shared memory, by configuration

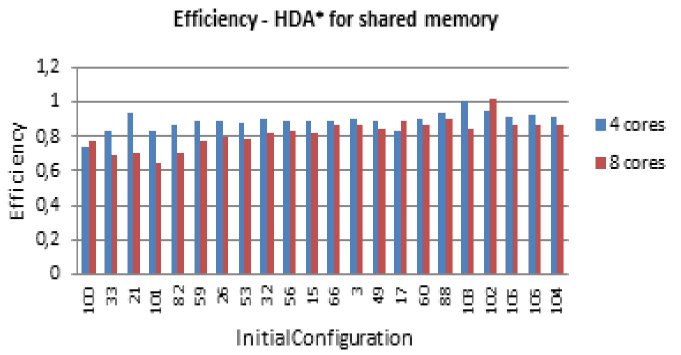


Fig. 3. Efficiency achieved by the HDA* algorithm for shared memory, by configuration

After analyzing the results shown in Fig. 2 and 3, it can be concluded that, for the same workload (initial configuration), if the number of cores is increased, the Speedup obtained is better. This proves that the problem is solved faster as more cores are used. However, efficiency does not normally remain constant. This decrease in efficiency is due to different factors, such as sequential parts especially at the beginning and at the end of computation, synchronization, idle time, load unbalance, search overhead increase, among other factors.

It is observed that when the problem is escalated maintaining the same number of processors, efficiency generally improves or remains constant as *overhead* is less significant on total processing time.

8 Conclusions and future lines of work

A version of the HDA* algorithm that is suitable for shared memory architectures and incorporates an effective technique to avoid performance degradation when there is a producer-consumer relation between various threads due to alloc-free operations was presented. The algorithm was run taking the Puzzle problem as study case and a more polished heuristic with respect to the classical one. On the other hand, it was proved that using active or passive waiting when the thread becomes idle is irrelevant, as there are no significant variations in performance.

This paper shows a scalability analysis of the parallel algorithm on a machine with multicore processors. From the results obtained, it can be concluded that the behavior exhibited is typical of a scalable parallel system, where efficiency can be kept constant when workload and architecture are escalated.

Future lines of work focus on contrasting the algorithm presented in this paper against HDA* for distributed memory (implemented exclusively with MPI), comparing the performance achieved and the amount of memory used.

9 References

[1] S. Russel and P. Norvig, Artificial Intelligence: A Modern Approach, Segunda edición ed., New Jersey: Prentice Hall, 2003.

[2] P. Hart, N. Nilsson and B. Raphael, “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”, IEEE Transactions on Systems Science and Cybernetics, vol. 4, nº 1, pp. 100-107, 1968.

[3] V. Kumar, K. Ramesh and V. N. Rao, “Parallel Best-First Search of State-Space Graphs: A Summary of Results”, de Proceedings of the 10th Nat. Conf. AI, AAAI, 1988.

[4] V.-D. Cung and B. Le Cun, “An efficient implementation of parallel A*”, de Proceedings of the First Canada-France Conference on Parallel and Distributed Computing , Montréal, 1994.

[5] E. Burns, S. Lemons, W. Ruml and R. Zhou, “Best-First Heuristic Search for Multicore Machines.”, Journal of Artificial Intelligence Research, vol. 39, pp. 689-743 , 2010.

[6] Kishimoto, A. Fukunaga and A. Botea, “Evaluation of a simple, scalable, parallel best-first search strategy”, Artificial Intelligence, pp. 222–248, 2013.

[7] J. Evans, “A Scalable Concurrent malloc(3) Implementation for FreeBSD”, de Proceedings of the 3rd annual Technical BSD Conference, Ottawa, 2006.

[8] E. W. Dijkstra, “Shmuel Safra’s version of termination detection EWD-Note 998”, 1987.

[9] E. W. Dijkstra, W. H. J. Feijen and A. J. M. Van Gasteren, “Derivation of a termination detection algorithm for distributed computations”, Information Processing Letters, vol. 16, pp. 217-219, 1983.

[10] D. Ratner and M. Warmuth, “The (n2−1)-puzzle and related relocation problems”, Journal of Symbolic Computation, vol. 10, nº 2, pp. 111-137, 1990.

[11] R. Korf and L. Taylor, “Finding Optimal Solutions to the Twenty-Four Puzzle”, Proceedings of the Thirteenth National Conference on Artificial Intelligence, 1996.

[12] W. Gloger, “Wolfram Gloger's malloc homepage”, 2014. http://www.malloc.de/en/.

[13] Aho, J. Ullman and J. Hopcroft, Data Structures and Algorithms, First Edition, Boston, MA: Addison-Wesley Longman Publishing Co, 1983.

[14] R. Ramakrishnan and J. Gehrke, Database Management Systems, Second Edition, McGraw-Hill Education, 1999.

[15] A. L. Zobrist, “A New Hashing Method with Application for Game Playing”, University of Wisconsin, Madison, 1968.

[16] Intel Ark, 2010. http://ark.intel.com/products/47925.

[17] R. Korf, “Depth-first Iterative-Deepening: An Optimal Admissible Tree Search”, Artificial Intelligence, pp. 97-109, 1985.

[18] A. Brüngger, Solving Hard Combinatorial Optimization Problems in Parallel: Two Cases Studies, Zurich, 1998.

[19] D. Gove, Multicore Application Programming. For Windows, Linux and Oracle (c) Solaris. Developers' Library, Boston, MA: Pearson Education, 2011.

[20] Grama, A. Gupta, G. Karypis and V. Kumar, Introduction to Parallel Computing (Second Edition), Edinburgh Gate, Harlow, Essex: Pearson, 2003.

Compiler-Level Explicit Cache for a GPGPU Programming Framework

Tomoharu Kamiya[1], **Takanori Maruyama**[1], **Kazuhiko Ohno**[1], **and Masaki Matsumoto**[2]
[1]Department of Information Engineering, Mie University, Tsu, Mie, Japan
[2]Medical Engineering Institute, Inc., Tsu, Mie, Japan

Abstract— *GPU is widely used for high-performance computing. However, standard programming framework such as CUDA and OpenCL requires low-level specifications, thus programming is difficult and the performance is not portable. Therefore, we are developing a new framework named MESI-CUDA. Providing virtual shared variables accessible from both CPU and GPU, MESI-CUDA hides complex memory architecture and eliminates low-level API function calls. However, the performance of current implementation is not sufficient because of the large memory access latency. Therefore, we propose a code-optimization scheme that utilizes fast on-chip shared memories as a compiler-level explicit cache of the off-chip device memory. The compiler estimates access count/range of arrays using static analysis. For mostly reused variables, code is modified to make copy on the shared memory and access the copy, using small shared memories efficiently. As the result of evaluation, our scheme achieved 13%–192% speedup in two of three programs.*

Keywords: GPGPU, CUDA, parallel programming, compiler, optimization

1. Introduction

The performance of Graphics Processing Unit (GPU) has been improved rapidly [1]. Therefore, recent GPUs are used as generic high-performance computing resources. Such GPU usage is called General Purpose computation on Graphics Processing Unit (GPGPU) [2]. However, current de facto GPGPU programming frameworks such as CUDA [3] and OpenCL [4] are still difficult to use. They provide APIs for low-level specifications such as memory allocation and data transfer. Although they enable the user to hand-optimize the performance of the program, it requires deep knowledge of GPU architecture. Furthermore, such optimization may not be portable to the different GPU models.

Therefore, we are developing a new framework named *MESI-CUDA* [5], [6] for easier GPGPU programming. MESI-CUDA is a CUDA variation which hides low-level GPU features. It provides virtual shared variables which can be accessed from both CPU and GPU. Explicit memory management or data transfer are not needed. The user can write MESI-CUDA program without low-level specifications expecting automatic optimization by the compiler.

However, current optimization is not sufficient. One reason is that current implementation uses only off-chip device memory. Fast on-chip GPU memories called shared memories are not used. Thus we propose an optimization scheme that automatically utilize the shared memories as compiler-managed caches of the device memory. Based on the result of static analysis, our scheme determines variables to cache so that device memory accesses are minimized. Then copying/writing back code is inserted and the code accessing the variables is modified. Thus the target program is optimized to use shared memories as explicit caches.

This paper is organized as follows: Section 2 gives a brief introduction of GPU/CUDA/MESI-CUDA and points out the current issue. In Section 3 we discuss the related works. Section 4 details the proposed scheme and Section 5 shows the evaluation results. In Section 6, we state the conclusion.

2. Background

2.1 GPU Architecture

GPU is a collection of streaming multiprocessors (SM), which have certain number of CUDA cores. Although CUDA cores are simpler than typical CPU cores, a GPU has hundreds or thousands of CUDA cores. Thus the potential performance of a GPU is much higher than a CPU.

Fig. 1 shows a typical architecture of a GPU card installed on a PC. Similarly as the CPU cores share the main memory (called *host memory* in CUDA programming), all CUDA cores share a large off-chip *device memory*. Furthermore, each SM has a small on-chip memory called *shared memory*, which is shared by all CUDA cores in the SM. We do not discuss other memories, such as constant and texture memories, because the proposed scheme does not use them.

NVIDIA GPU architecture have been evolved in each generations Tesla/Fermi/Kepler changing specifications and introducing new features. Different models often have different specifications even if they belong to the same generation.

2.2 CUDA

CUDA (Compute Unified Device Architecture) [3], [7], [8] is a GPGPU programming framework using extended C/C++ or Fortran. Fig. 2 shows a (non-optimized) matrix multiplication program using CUDA. The additional code required for parallel programming is shown in bold font.

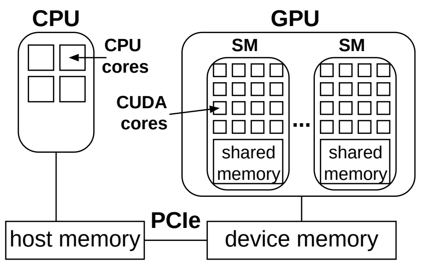


Figure 1: GPU Architecture

```
 1 #define N 1024
 2 #define BX 128
 3 #define S (N*N*sizeof(int))
 4 int ha[N][N], hb[N][N], hc[N][N];
 5 __global__
   void transpose(int a[][N], int b[][N], int c[][N]){
 6   int k;
 7   int row = blockDim.y*blockIdx.y+threadIdx.y;
 8   int col = blockDim.x*blockIdx.x+threadIdx.x;
 9   c[row][col] = 0;
10   for(k = 0 ; k < N ; k++){
11     c[row][col] += a[row][k] * b[k][col];
12   }
13 }
14 void init_array(int d[N][N]){...}
15 void output_array(int d[N][N]){...}
16 int main(int argc, char *argv[]){
17   int *da, *db, *dc;
18   dim3 dimGrid(N/BX, N);
19   cudaMalloc(&da, S);
20   cudaMalloc(&db, S);
21   cudaMalloc(&dc, S);
22   init_array(ha);
23   init_array(hb);
24   cudaMemcpy(da, (int*)ha, S, cudaMemcpyHostToDevice);
25   cudaMemcpy(db, (int*)hb, S, cudaMemcpyHostToDevice);
26   transpose<<<dimGrid, BX>>>
         ((int(*)[N])da,(int(*)[N])db,(int(*)[N])dc);
27   cudaMemcpy((int*)hc, dc, S, cudaMemcpyDeviceToHost);
28   output_array(hc);
29   cudaFree(da);
30   cudaFree(db);
31   cudaFree(dc);
32 }
```

Figure 2: CUDA Matrix Multiplication

In CUDA, CPU and GPU are called *host* and *device*, respectively. Functions, declared with the `__device__` or `__global__` qualifier, are called *kernel functions* and executed on the device (Fig. 2 l. 5–13). The other functions (called *host functions* in this paper) are executed on the host (l. 14–32). To start computation on the GPU, any host function invokes a `__global__` kernel function (called *kernel invocation*) specifying the number of threads (l. 26). Then, the created GPU threads execute the kernel function. In this paper, we simply call GPU threads as *threads*.

CUDA uses *grids* and *blocks* for controlling thread mapping to data and physical resources. A block is a group of threads executed on the same SM, and a grid is a group of blocks of the same size. A kernel invocation creates a grid with the specified grid/block sizes, which are the numbers of total blocks and threads per block, respectively.

Table 1: CUDA Built-in variables

`gridDim.x`, `gridDim.y`, `gridDim.z`	grid size (# of blocks)
`blockIdx.x`, `blockIdx.y`, `blockIdx.z`	block index (in the grid)
`blockDim.x`, `blockDim.y`, `blockDim.z`	block size (# of threads)
`threadIdx.x`, `threadIdx.y`, `threadIdx.z`	thread index (in the block)

The grid/block sizes can be specified as integer values or 3D vectors using a built-in type `dim3`. Fig. 2 program creates a grid of `N/BX`×`N` blocks and each block consists of `BX` threads (Fig. 2 l. 18, 26). The grid/block sizes are not limited by the numbers of SMs and CUDA cores; blocks and threads are automatically mapped to the physical resources.

Grid/block sizes and block/thread indices can be obtained using built-in variables shown in Table 1. Using the variables in the index expressions of arrays, each thread can make the same computation on the different array element. In the kernel function `transpose()` of Fig. 2 program, `row` and `col` are computed using block/thread indices so each thread computes different element of the array `c` (l. 7–12).

The host/device memories are only accessible from CPU/CUDA cores, respectively. To share data between CPU and GPU, memory allocations on both memories and data transfers between them are required. In CUDA programming, the user must explicitly describe such low-level behaviors calling API functions: memory allocation/deallocation calling `cudaMalloc()`/`cudaFree()` (l. 19–21, 29–31) and data transfer calling `cudaMemcpy()` (l. 24–25, 27).

2.3 CUDA Optimization Techniques

In CUDA programming, hand-optimization considering the architecture-level features of the target GPU often largely contributes achieving high-performance.

2.3.1 Controlling Concurrency/Parallelism

Because each SM executes a *warp* of 32 threads in a SIMD manner, the thread block size should be an integral multiple of the warp size 32. It is better to have multiple warps in a block because the execution can be switched to hide the latency when the active warp is stalled on memory accesses. The number of blocks also should be large enough so that concurrent blocks run on each SM.

2.3.2 Optimizing Memory Usage

The parallel device memory accesses in a warp are coalesced if the requested data are in the same L2 cache line of 128 bytes, but otherwise they are serialized. Thus, the threads in a warp are better to access the neighboring data on the memory at the same time. [8].

Another memory-usage optimization is allocating frequently used data on the shared memories. Because their access latency is much smaller, they can be used as caches of the device memory. Although CUDA enables shared memories to be configured as L1 caches, the performance is often not sufficient. Their size of 48KB [1] is too small and cached data tend to be not reused when threads are scanning large arrays. Thus explicit caching on the application layer is used as an optimization.

A local variable of a kernel function can be allocated on the shared memory using `__shared__` qualifier. However, explicit copy from/to the device memory is needed in the function because direct copying between the host and shared memories is not possible. Such variable is shared among all threads in the block, thus explicit synchronization calling `__syncthreads()` may be needed.

2.3.3 Reducing Data Transfer Overhead

The data transfer between host/device memories can be another bottleneck of performance. Avoiding fine-grained transfers or overlapping transfers/kernel executions using asynchronous transfer functions improves the performance.

2.4 MESI-CUDA

CUDA programming API is based on the complex GPU architecture. Although such low-level API enables hand-tuning considering hardware specifications, it is difficult and may not be efficient on other GPU models. Therefore we are developing an easier GPGPU programming framework *MESI-CUDA* [5], [6], hiding low-level features from the user.

In MESI-CUDA, basic parallelization scheme is same as CUDA: writing host/kernel functions for CPU/CUDA cores and invoking the latter from the former. We do not hide this explicit parallelization because the characteristics of CUDA cores are quite different from the CPU cores. For example, CUDA cores can run fine-grained threads with small overhead, but branch divergent code is inefficient. It would be unpractical to ignore such differences in high-performance computing using GPU.

On the other hand, we adopted a virtual shared memory model that all CPU/CUDA core share a single global memory (Fig. 3). Actually, only global variables defined with `__global__` qualifier are shared. To avoid confusing with the CUDA variables defined with `__shared__` qualifier, we call our shared variables as *virtual shared variables* or *VS variables*. This design is due to the following reasons:

1) GPU has no hardware/OS support to implement generic virtual shared memory. Virtual-sharing of the specified variables can be implemented at compiler-level, using static analysis and inserting appropriate data transfer code between host/device memories.
2) Because the cores are heterogeneous, the roles of host/device are clear. Many working variables are accessed only on either of the host/device. Thus explicit sharing of minimum variables is safe and also efficient.

Generally, shared memory based parallel programming requires synchronization and mutual exclusion. However,

[1]User-available size is currently 48KB of physical 64KB memory.

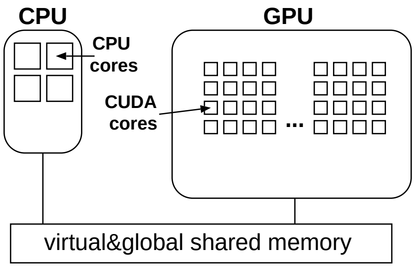


Figure 3: MESI-CUDA Programming Model

```
 1 #define N 1024
 2 #define BX 128
 3 __global__int ga[N][N], gb[N][N], gc[N][N];
 4 __global__
   void transpose(int a[][N], int b[][N], int c[][N]){
 5   int k;
 6   int row = blockDim.y*blockIdx.y+threadIdx.y;
 7   int col = blockDim.x*blockIdx.x+threadIdx.x;
 8   c[row][col] = 0;
 9   for(k = 0 ; k < N ; k++){
10     c[row][col] += a[row][k] * b[k][col];
11   }
12 }
13 void init_array(int d[N][N]){...}
14 void output_array(int d[N][N]){...}
15 int main(){
16   init_array(ga);
17   init_array(gb);
18   transpose<<<dimGrid, BX>>>
       ((int(*)[N]))ga,(int(*)[N]))gb,(int(*)[N]))gc);
19   output_array(gc);
20 }
```

Figure 4: MESI-CUDA Matrix Multiplication

data access races are usually avoided in GPU programming because of the poor synchronization mechanism. Thus we adopt implicit synchronization that the shared values are made logically consistent on each kernel invocation.

Without low-level description of memory management and data transfer [2], the user can concentrate on device-independent parallel algorithm. For example, the matrix multiplication program in Fig. 2 can be simplified using MESI-CUDA as shown in Fig. 4. The additional code required for parallel programming in MESI-CUDA is shown in bold font. The arrays for 2D matrices are defined as VS variables and can be accessed from both host/kernel functions (Fig. 4 *l*. 3) [3]. We support variable-length array and dynamic allocation of VS variables [6], but in this paper we only discuss VS variables of static sizes.

The MESI-CUDA compiler is a translator to CUDA and generates low-level code for memory management and data transfer. Our research goal is to automatically apply the

[2]To make MESI-CUDA upper-compatible to CUDA, we did not remove low-level API functions. If the optimization of MESI-CUDA compiler is not sufficient, the user can hand-optimize like CUDA.

[3]If the input/output variables of multiplication is fixed to `ga`, `gb`, and `gc`, they can be directly accessed in kernel functions and passing as function arguments is not needed.

optimizations described in Section 2.3 and achieve high performance like hand-optimized CUDA programs.

3. Related Works

The latest CUDA 6 [7] and Kepler GPUs implemented *Unified Memory*, which enables to allocate *managed memory* by either statically defining a variable with `__managed__` qualifier or dynamically calling `cudaMallocManaged()`. Such memory can be accessed from both CPU and GPU.

The features are almost same with MESI-CUDA's VS variables; only user-specified data is logically shared and they are automatically copied between host/device memories. The large difference is that VS variables are implemented in compiler-level, while the managed memory is implemented in hardware/driver-level. Our advantage is that compile-time optimization is possible using static analysis. For example, asynchronous data copying code can be inserted where the data transfer and kernel executions are overlapped. Another example is the synchronizations between host and device. MESI-CUDA automatically inserts synchronization code to maximize their parallel execution, while CUDA 6 requires explicit synchronization calling `cudaDeviceSynchronize()` or setting a environment variable `CUDA_LAUNCH_BLOCKING` as 1 to automatically synchronize for every kernel invocation.

The main purpose of Unified Memory is easier GPGPU programming and hand-optimization using conventional low-level API is encouraged for high-performance. The goal of MESI-CUDA is to hide optimization under the compiler. However, generating code using new CUDA features may help to utilize hardware/driver supports for such features.

OpenACC [9] or OpenMP-to-CUDA translation [10], [11] are another GPGPU approach without low-level specifications. In these programming frameworks, a sequential program with some parallelizing directives is compiled into a parallel program executable on GPU. They have advantages on usability; abstract directives are easier than low-level API functions, sequential programs can be parallelized easily, and the program is portable to different GPU models or other heterogeneous multi-cores. However, their performance depends to the compiler optimization, which is usually worse compared with hand-optimized CUDA code [12]. As mentioned in Section 2.4, we consider explicit and heterogeneous parallel programming is necessary for high-performance.

For various input languages, schemes for automatic generation and optimization of CUDA low-level code are developed. CUDA-Lite [13] automatically generates memory access code from user specified annotations, optimizing accesses using shared memories. Yang, et al. [14] optimize memory accesses in CUDA kernel functions using shared memories for coalescing accesses to the device memory. Although our scheme is similar to these approaches, we do not assume additional annotations by the user and it is supposed to be a part of global optimization including mapping and scheduling in future.

4. Proposed Scheme

Current implementation of MESI-CUDA allocates area for virtual shared (VS) variables on host/device memories. Therefore, every access to the VS variables on GPU is a access to the device memory. By caching VS variables on shared memories, the memory access latency is largely reduced and the performance will be improved. Therefore, we propose a new scheme that the compiler automatically makes the optimization of explicit caching mentioned in Section 2.3.2.

For simplicity, we discuss the case that a kernel function f is invoked by the following statement, where S_g, S_b are integer values:

$$f\texttt{<<<}S_g\texttt{, }S_b\texttt{>>>(}\ldots\texttt{);}$$

The values of `GridDim.x` and `BlockDim.x` will be S_g, S_b, respectively. If f calls other device functions, the analysis and code generation are extent to cover such functions. We also denote the size of available shared memory per block as C. On current GPU models, C=48KB but it may be changed in the future models. Furthermore, using a smaller value as C suppresses the shared memory usage per block, which can increase concurrent blocks per SM.

4.1 Caching Strategy

The scope of variables defined with `__shared__` qualifier is within the defined kernel function f. Thus the caching candidates are the variables which are on the device memory and accessed in f. Assuming that enough registers are available for local variables in f, VS variables and dereferences of pointer arguments will be the device memory accesses. We denote the list of caching candidate as V and include all such variables in V as the initial value.

Because each access latency is reduced for the cached variables, caching is more effective if the number of accessing the variable is larger. However, copying from/to the device memory causes another overhead which increases according to the variable size.

If the variable is an array, not all elements may be accessed in a block. So caching the set of accessed array elements is enough. However, static analysis may not obtain strict set of accessed elements. Furthermore, generating efficient access code is difficult for irregular access patterns. Therefore, we make static analysis to obtain the range of accesses on each dimension of the array. Instead of caching the whole array, the subarray of the obtained range is cached. For multi-dimensional arrays, the obtained range may not be a single continuous area on the device memory. In such cases, the required areas are packed into a continuous area on the shared memory, forming the subarray.

Our scheme gives higher priority of caching if a candidate variable has higher access count per byte. We first make static analysis to obtain the access counts, select variables to be cached, and finally generate code to cache the variables.

4.2 Static Analysis

We make static analysis on each kernel function and obtain access count of each variable in V. We also obtain required bytes for caching each variable. Here we expect that kernel functions satisfy the following assumptions:

1) All loop iteration numbers are fixed and known at compile time.
2) All array index expressions are first degree polynomials on all loop and built-in index variables (`threadIdx.x`, `blockIdx.x`, etc.). For example, `a[i*N+j]` or `a[threadIdx.x/N+i]` are acceptable but `a[i*j]` or `a[threadIdx.x/i]` are not.
3) Kernel functions may have conditional statements, but the branch probability is regard as 1/2; the access counts of `if`/`else` blocks are averaged and the access ranges are merged.

While most practical sequential programs will not satisfy these assumptions, many CUDA programs will satisfy. To prevent inefficient branch divergence, using `if`/`while` statements is tend to be avoided. Index expressions are commonly simple and linear because data elements should be divided equally to the threads preventing access races and balancing the load statically. Even if the assumptions are not satisfied for some candidate variables, we can just remove such variables from the candidate list V and apply our scheme to other variables.

Considering the assumption 1, 3, the access count of a candidate variable v, denoted as $\mathit{access}(v)$, is obtained as the sum of each access count of the variable occurrences. An access count of a variable occurrence is a product of loop iterations which include the occurrence. Considering the assumption 2, the access range of array elements is obtained by computing the index expression value with the minimum/maximum values of loop variables.

Suppose that a candidate variable v is a m-dimensional array and accessed in a kernel function f. The access range of v in the thread $t_{p,q}$ ($q = 0, \ldots, S_b - 1$) belonging to a block b_p ($p = 0, \ldots, S_g - 1$) is obtained as follows.

We denote occurrences of v in f as $v^1, \ldots, v^k$ and s-th index expression of v^r as $e_s(v^r)$. We compute the values of $e_s(v^r)$ on every combination of minimum/maximum values of loop variables. Considering the assumption 2, minimum/maximum of the computed values are the minimum/maximum values of $e_s(v^r)$, denoted as $\mathit{min}(e_s(v^r))$ and $\mathit{max}(e_s(v^r))$ [4]. We denote the range of an index expression value as $R_s(e_s(v^r)) = [\mathit{min}(e_s(v^r)), \mathit{max}(e_s(v^r))]$.

[4]It may not be true if the modulo operator `%` is used because the operation is not monotonic. We simply regard the minimum/maximum value of the term $e\%M$ in the expressions as 0 and $M - 1$, respectively.

```
Set target variable set $V_c$ empty
Sort candidate list $V$ in the descending order of $\overline{access}(v)$
while ($V$ is not empty){
  Select first variable $v_t$ in $V$ and remove $v_t$ from $V$
  if ($byte(R(v_t, b_p) \le C$){
    $V_c \leftarrow V_c \cup \{v_t\}$
    $C \leftarrow C - byte(R(v_t, b_p))$
  }
}
```

Figure 5: Algorithm Obtaining Variables to Cache

The access range of v^r is a m-dimensional range denoted as follows [5]:

$$R(v^r) \quad = \quad R_1(e_1(v^r)) \times \ldots R_m(e_m(v^r))$$

The access range of v in the thread $t_{p,q}$ and in the block b_p is obtained as follows:

$$\begin{aligned} R(v, t_{p,q}) &= \bigcup_{r=1}^{k} R(v^r, t_{p,q}) \\ R(v, b_p) &= \bigcup_{q=0}^{S_b-1} R(v, t_{p,q}) \end{aligned}$$

We define the union of two ranges $R' \cup R''$ as a minimum range including R' and R''.

The required size (number of array elements) and memory bytes for caching v are computed as follows:

$$\begin{aligned} \mathit{size}(R_s(v, b_p)) &= \mathit{emax}(v, b_p, s) - \mathit{emin}(v, b_p, s) + 1 \\ \mathit{size}(R(v, b_p)) &= \mathit{size}(R_1(v, b_p)) \times \ldots \mathit{size}(R_m(v, b_p)) \\ \mathit{byte}(R(v, b_p)) &= \mathit{size}(R(v, b_p)) \times \texttt{sizeof(}\textit{type of } v\texttt{)} \end{aligned}$$

where

$$\begin{aligned} R(v, b_p) &= R_1(v, b_p) \times \ldots R_m(v, b_p) \\ R_s(v, b_p) &= [\mathit{emin}(v, b_p, s), \mathit{emax}(v, b_p, s)] \end{aligned}$$

and $\mathit{emin}(v, b_p, s)$, $\mathit{emax}(v, b_p, s)$ are respectively minimum/maximum value of $e_s(v^r)$ for all q, r.

4.3 Optimization

Fig. 5 shows the algorithm of obtaining a set of variables to cache: V_c. The average access count per byte of a variable v_t is denoted as $\overline{\mathit{access}}(v_t)$, which is computed as follows:

$$\overline{\mathit{access}}(v_t) \quad = \quad \mathit{access}(v_t)/byte(R(v_t, b_p))$$

4.4 Code Generation

For each caching target $v_t \in V_c$, we apply the following code generation/modification in the kernel function f.

[5]This definition of range assumes that possible values of each index expression is continuous and the expressions are independent each other. The range is redundant in the cases of non-unit stride access patterns or dependent expressions like `a[i][i]`. Introducing more accurate range is the future work.

```
__shared__ type _s_v_t[S_m(v_t)]...[S_1(v_t)];
int _ix1, ..., _ixm;
for (_ixm = 0 ; _ixm < S_m(v_t) ; _ixm++){
    :
    :
for (_ix2 = 0 ; _ix2 < S_2(v_t) ; _ix2++){
for (_ix1 = 0 ; _ix1 < S_1(v_t) ; _ix1 += T ){
  _s_v_t[_ixm]...[_ix1]
     = v_t[_ixm+O_m(v_t)]...[_ix1+O_1(v_t)];
}}...}
__syncthreads();
```

Figure 6: Caching code for a variable v_t

4.4.1 Caching Variables

First, we insert the definition of a variable `_s_`v_t with `__shared__` qualifier. if v_t is an array, the size of the s-th dimension is *size*$(R_s(v, b_p))$. Next, we insert code for copying the initial values from the device memory and writing back the final values to the device memory to the head and tail of f, respectively.

The pseudo code defining `_s_`v_t and copying the initial values is shown in Fig. 6. For simplicity, we assume that the size of the first dimension $S_1(v_t)$ is an integral multiple of the number of copying threads $T =$ `blockDim.x`. We also use the following notations in Fig. 6.

$$\begin{aligned} S_s(v_t) &= \textit{size}(R_s(v_t, b_p)) \\ S(v_t) &= \textit{size}(R(v_t, b_p)) \\ O_s(v_t) &= \textit{emin}(v, b_p, s) \end{aligned}$$

Note that $S_s(v_t)$ and $S(v_t)$ are constant but $O_s(v_t)$ will be not. In most cases, it includes block indices such as `blockIdx.x` thus different on each block.

To copy the initial cache values, each array element of v_t within the caching range is assigned to the corresponding element of `_s_`v_t. Because the elements consecutive on the first dimension are consecutive in the device memory, coalesced accesses are expected on the parallel assignment of such elements. If later accesses in the block are not consecutive, they will cause non-coalesced device accesses without our cache. Using our scheme, the array is cached using coalesced accesses then shared memories are accessed later. Therefore, the access latency will be largely reduced. After copying, `__syncthreads()` must be called to ensure copying is completed before starting computations on them. If v_t is write-only in f, code for copying and synchronization is omitted.

The write-back code will be reverse copy of Fig. 6. The code can be omitted if v_t is read-only in f. Synchronization is not needed after the write-back, because the threads end immediately after that.

4.4.2 Accessing Cache

Each occurrence of v_t in f is replaced with `_s_`v_t. If v_t is an array and only its subset is cached, the index expressions must be modified as follows:

$$v_t\texttt{[}e_m\texttt{]}\ldots\texttt{[}e_1\texttt{]} \quad \rightarrow \quad \texttt{\_s\_}v_t\texttt{[}e_m\texttt{-}O_m(v_t)\texttt{]}\ldots\texttt{[}e_1\texttt{-}O_1(v_t)\texttt{]}$$

```
 4 __global__
   void transpose(int a[][N], int b[][N], int c[][N]){
 5   int k, _ix1;
 6   int row = blockDim.y*blockIdx.y+threadIdx.y;
 7   int col = blockDim.x*blockIdx.x+threadIdx.x;
 *   __shared__int _s_a[1][N];
 *   __shared__int _s_c[BX];
 *   for (_ix1 = threadIdx.x ; _ix1 < N ; _ix1 += BX){
 *     _s_a[0][_ix1] = a[row][_ix1];
 *   }
 *   __syncthreads();
 8   _s_c[threadIdx.x] = 0;
 9   for(k = 0 ; k < N ; k++){
10     _s_c[threadIdx.x] += _s_a[0][k] * b[k][col];
11   }
 *   c[row][col] = _s_c[threadIdx.x];
12 }
```

Figure 7: Optimized Kernel Function of Fig. 4 Program

Table 2: Evaluated Programs

name	description
`matmul`	matrix multiplication shown in Fig. 4
`dif`	single dimension diffusion equation solver using difference method
`ep`	EP (Embarrassingly Parallel) in NAS Parallel Benchmarks [15]

Fig. 7 is the result of applying our scheme to kernel function `transpose()` in Fig. 4. Modifications are shown in bold font.

5. Evaluation

To evaluate our scheme, we compared the execution time of MESI-CUDA programs shown in Table 2, applying/not applying the proposed optimization. The result is shown in Table 3. The columns 'normal' and 'opt' are the execution time of programs applying and not applying our optimization, respectively. The column 'speedup' is the inverse of the execution time ratio of 'opt' to 'normal'.

Our optimization achieved speedup on all GPU models for `matmul` and `dif`. As shown in Fig. 7, matrices A and C of $C = A \times B$ are cached in `matmul` and achieved 13% to 192% speedup when S_b is optimized to be the best. In `dif`, the required size for caching is only $S_b \times 4$ bytes and each array elements are shared between adjacent threads. Large block size is possible without reducing concurrent blocks.

As for the result of `ep`, our optimization achieved 23% to 98% speedup on C2050. When $S_b = 32$, it also slightly improved performance on other GPU models. However, the optimization caused slowdown for Kepler GPUs for larger S_b. Applied to `ep`, our optimization caches small arrays for random-accessed histogram but the main array for storing random numbers is too large to be cached. Therefore the contribution of reducing access latency is limited. In addition, large S_b reduces concurrent blocks because the required size of the histogram is $S_b \times 80$ bytes.

Table 3: Execution Time and Speedup using Proposed Scheme

Data Size	Block Size	Tesla C2050 (Fermi)			GeForce GTX 680 (Kepler)			GeForce Titan (Kepler)			Tesla K20 (Kepler)		
	S_b	normal(s)	opt(s)	speedup	normal(s)	opt(s)	speedup	normal(s)	opt(s)	speedup	normal(s)	opt(s)	speedup
`matmul`													
1024^2	32	0.162	0.046	3.56	0.091	0.049	1.86	0.065	0.032	2.04	0.082	0.040	2.03
	64	0.089	0.026	3.47	0.051	0.026	1.95	0.036	0.018	2.00	0.046	0.022	2.12
	128	0.055	0.029	1.93	0.034	0.029	1.16	0.024	0.015	1.58	0.033	0.019	1.80
	256	0.045	0.038	1.18	0.034	0.030	1.13	0.024	0.015	1.62	0.033	0.018	1.89
	512	0.045	0.036	1.26	0.034	0.024	1.42	0.024	0.015	1.62	0.034	0.011	3.18
`dif`													
256K	32	6.12	4.13	1.48	4.03	2.47	1.63	2.85	1.66	1.72	4.15	2.39	1.73
	64	3.37	2.20	1.53	2.25	1.35	1.66	1.64	0.97	1.69	2.34	1.35	1.73
	128	2.17	1.35	1.60	1.71	0.91	1.88	1.26	0.75	1.68	1.72	1.00	1.73
	256	1.90	1.26	1.50	1.75	0.96	1.82	1.34	0.78	1.73	1.81	1.04	1.75
	512	2.06	1.43	1.45	1.88	1.09	1.73	1.45	0.87	1.66	1.96	1.14	1.71
512K	32	12.21	8.20	1.49	7.97	4.86	1.64	5.60	3.24	1.73	8.23	4.70	1.75
	64	6.70	4.37	1.53	4.41	2.63	1.68	3.17	1.84	1.72	4.62	2.64	1.75
	128	4.27	2.66	1.61	3.35	1.75	1.92	2.38	1.38	1.72	3.42	1.95	1.75
	256	3.74	2.48	1.50	3.42	1.86	1.84	2.51	1.44	1.74	3.60	2.04	1.77
	512	4.05	2.81	1.44	3.70	2.10	1.76	2.72	1.61	1.69	3.90	2.27	1.71
`ep`													
class B	32	1.60	1.25	1.29	2.04	1.88	1.09	1.36	1.30	1.05	1.17	0.99	1.18
	64	1.05	0.83	1.27	1.84	1.84	1.00	1.29	1.27	1.02	0.74	0.90	0.82
	128	1.65	0.83	1.98	1.81	1.92	0.94	1.30	1.30	1.00	0.94	0.99	0.95
	256	1.30	0.84	1.55	1.80	1.95	0.92	1.29	1.30	1.00	0.92	0.99	0.93
	512	1.03	0.83	1.23	1.79	1.90	0.95	1.33	1.32	1.01	0.93	1.01	0.92

6. Conclusion

Although GPGPU is widely used for high-performance computing, major programming frameworks like CUDA are difficult and the performance is not portable. Therefore, we are developing an easier programming framework MESI-CUDA. However, access latency of virtual shared variables is large, thus we proposed an automatic optimization scheme using on-chip shared memories as explicit cache.

To select variables of higher reused rate as the caching targets, we make static analysis to obtain the average access counts and accessed range in a block for each variable. The target variables are determined at compile time and code for explicit caching is automatically generated. Therefore, no support in hardware/driver-level is required and the dynamic overhead of cache management does not occur.

As the result of evaluations, our scheme achieved 13% to 192% speedup for `matmul`/`dif` programs but slowdown for `ep` program running on Kepler GPUs. Using shared memories reduces concurrent blocks on a SM thus the trade-off should be considered for applying our optimization.

As a future work, the result of current range analysis may be redundant and should be improved. Recognizing non-unit stride access patterns and packing required elements on caching will save the capacity of shared memories. Another issue is that our scheme tries to utilize the shared memories under the restriction of user-specified grids and blocks. The optimization may be far from the best. For example, specifying large block size may increase the access range of arrays and prevent their caching due to the lack of capacity. Another example is that the threads of common accessing range are distributed into different blocks, which prevents to share the cache value. Our next challenge is to develop optimization scheme of threads/data mapping which automatically controls block size and improve efficiency of data accesses and caching.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number 24500060.

References

[1] J. D. Owens *et al.*, “A survey of general-purpose computation on graphics hardware,” *Computer Graphics Forum*, vol. 26, no. 1, pp. 80–113, 2007.

[2] “Gpgpu.org,” http://www.gpgpu.org/.

[3] “CUDA Zone,” http://developer.nvidia.com/category/zone/cuda-zone.

[4] “OpenCL,” http://www.khronos.org/opencl/.

[5] K. Ohno, D. Michiura, M. Matsumoto, T. Sasaki, and T. Kondo, “A GPGPU programming framework based on a shared-memory model,” *Parallel and Distributed Computing and Networks*, vol. 3, pp. 1–14, 2013.

[6] K. Ohno, M. Matsumoto, T. Kamiya, and T. Maruyama, “Supporting dynamic data structures in a shared-memory based GPGPU programming framework,” in *Proc. 24th IASTED Intl. Conf. on Parallel and Distributed Computing and Systems*, 2012, pp. 122–131.

[7] *NVIDIA CUDA C Programming Guide*, 6th ed., NVIDIA Corporation, February 2014.

[8] *CUDA C Best Practices Guide*, NVIDIA Corporation, January 2012.

[9] “OpenACC Home,” http://www.openacc-standard.org/.

[10] S. Lee, S. Min, and R. Eigenmann, “OpenMP to GPGPU: a compiler framework for automatic translation and optimization,” *SIGPLAN Not.*, vol. 44, pp. 101–110, 2009.

[11] “OpenMP,” http://openmp.org/.

[12] T. Hoshino, N. Maruyama, S. Matsuoka, and R. Takaki, “CUDA vs OpenACC: Performance case studies with kernel benchmarks and a memory-bound CFD application.” in *CCGRID*. IEEE Computer Society, 2013, pp. 136–143.

[13] S. Ueng, M. Lathara, S. S. Baghsorkhi, and W. W. Hwu, “CUDA-Lite: Reducing GPU programming complexity,” in *Languages and Compilers for Parallel Computing*, 2008, pp. 1–15.

[14] Y. Yang, P. Xiang, J. Kong, and H. Zhou, “A GPGPU compiler for memory optimization and parallelism management,” *SIGPLAN Not.*, vol. 45, pp. 86–97, 2010.

[15] “NAS parallel benchmarks,” https://www.nas.nasa.gov/publications/npb.html.

High Performance Embedded Software Design for Protective Relay Algorithm in Digital Signal Processors

Heung Sun Yoon, Myungha Kim, Jong Kang Park, and Jong Tae Kim
Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon, South Korea

Abstract - *Most measurements and protective algorithms for power systems have been implemented by embedded software in more than a single digital signal processor (DSP). As more complex functions required for intelligent electronic devices, we need to improve the existing software solutions for DSPs in terms of execution time, cost and reliability. This paper suggests a fixed-point design technique for several mathematical functions in relay algorithms. By porting it to a real target system, we also evaluated that the proposed design can give higher speed of operations, higher sample rate and more increasing concurrent channel support capability than the existing solution.*

Keywords: protective relay, digital signal processor, fixed-point design

1 Introduction

Protective relays have been used for preventing and minimizing damage by detecting electric power system faults such as short circuit and ground fault in a timely manner. In today, key features of protective relay are supporting multi-channel sources and high precision protection control. Furthermore, there is a trend that equipment monitoring and control, communication and prevention diagnosis are required. Requests of operation performance have been extremely enlarged in order to implement these demands [1,2]. Therefore, more than two processors have been used to implement the protective relay in the existing design [3,4,5].

In this paper, we proposed embedded software optimized fixed point design for digital filter and protective algorithms. We presented expected processing speed of processor which has floating-point operations contrast with processer which doesn't have. And we verified that low-cost single DSP is possible to provide high-quality protective relaying by optimizing operations.

Our design provide following advantages.

- Single DSP can provide high sample rate which is 128 samples/period or more in real-time by improving processing speed so that we get more information from the electric power system.

- High sample rate improves noise cancellation performance of digital filter, as the result high quality protective relay is possible.

- The number of channels are increased with single DSP, since processing speed is improved.

- Coordination accuracy is always guaranteed within the designated fixed-point range, since simplification is not applied. It is easy to change parameters to handle various requirements.

By the result of this paper, the fixed-point implementation in TMS320C 6416 requires additional 128 bytes non-volatile memory compared to the software of floating-point version. However, the operation speed is improved by 53 times and the entire code memory usage is reduced by 25%. Furthermore, our code is running 27 times faster than the DSP embedding the floating point unit.

2 Fixed-point design of relay algorithm

In this section, the optimization of protective relay algorithm, hardware specifications and each details of the fixed-point design will be covered.

2.1 Software of protective relay

The main difference between the digital and traditional analog protective relay is that the digital protective relay is implemented by software which is executed by the instruction set architecture. Most of digital protective relay features such as protection, auto-monitoring, and man machine interface (MMI) are implemented by software.

The digital protective relay provides a number of features. As shown in Fig. 1, the embedded software divided into three categories such as protective relay software, interface and diagnosis software. Protective relay software include digital filter, protective algorithm, control sequence. Diagnosis software perform regular monitoring, auto- and self-diagnosis and MMI software which processes correction value and display. Protective relay software is being processed as a top priority, normal human machine interface software and diagnosis software will perform in the free time.

Fig 2 illustrates basic structure of protective relay software which is main subject of this paper. Voltage and current signals are came from external Potential Transformer (PT), Current Transformer (CT) and then these signals pass low-pass analog filter. After converting these signals by analog-to-

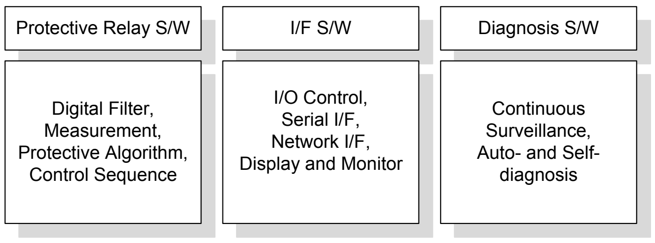

Protective Relay S/W
I/F S/W
Diagnosis S/W
Digital Filter, Measurement, Protective Algorithm, Control Sequence
I/O Control, Serial I/F, Network I/F, Display and Monitor
Continuous Surveillance, Auto- and Self-diagnosis

Fig. 1 Embedded S/W for Protective Relay

digital converter (ADC), timer interrupt service routines store data in the internal memory buffer at specified period. Unnecessary harmonic components are removed by digital filter at each reading voltage and current samples. Essential information to monitor and relay are extracted by measuring root-mean-square (RMS) value, phase and frequency. Instantaneous and inverse time protective algorithms are executed to detect variety faults through relay co-ordination property.

2.2 Target hardware specification

In this paper, we employed TMS320C6416 processor based on Von Neumann architecture and only supporting fixed-point operations by the hardware components. Fig 3 presents the detailed internal memory structure of the processor. TMS320C6416 processor is a VLIW architecture. Internal memory of processor consists of two-level cache hierarchical structures (Level 1 program cache memory 4KB (L1P), level 1 data cache memory 4KB (L1D) and 64KB RAM (L2)). Level 2 64KB RAM can be used in various ways such as program cache memory or data memory or mixture. As shown in Fig. 3, DSP Core load 256 bits program code at a clock from L1P, and 32 bits data from L1D. In case that

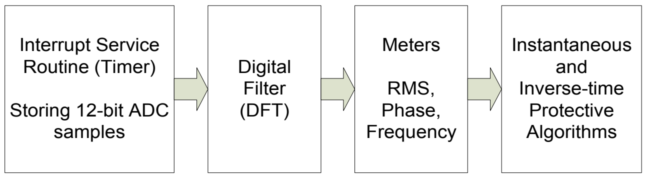

Interrupt Service Routine (Timer)
Storing 12-bit ADC samples
Digital Filter (DFT)
Meters
RMS, Phase, Frequency
Instantaneous and Inverse-time Protective Algorithms

Fig. 2 Block Diagram for Digital Protective Relay S/W

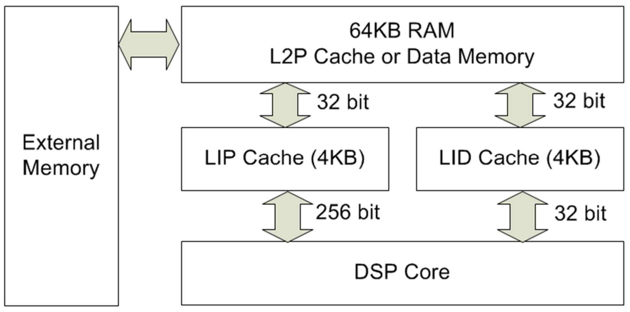


Fig. 3 Memory Structure for Target Signal Processor

64KB RAM is used for level 2 program cache (L2P), 32 bits data are transferred from L2P to L1P in a clock. In case that 64KB RAM is used for data memory, 32 bits data are transferred to L1D in a clock.

2.3 Fixed-point design of digital filter

High harmonic components of voltage and current signals which come from external analog circuit are eliminated by the digital filter. And then, the protective algorithm can be applied. In general, speed and reliability of digital protective relay are primarily determined by designing digital filter. We used DFT (Discrete Fourier Transform) filter for the relay. As shown in Fig. 4, $X_{re}(k)$ and $X_{im}(k)$ which are real number and imaginary number of X(n) respectively are calculated to obtain k harmonic component by convolution in fixed-point. N and r represent the number of samples and filter tap respectively. A fixed-point format (type, W, F) in Fig.4 is composed of the data type, the number of total bits and the number of bits for fractional part, respectively. For example, (short, 12, 0) represents the corresponding data is a short integer type with 12-bit length and has no fractional bits. 12-bit ADC samples and previously stored DFT filter coefficients are calculated by convolution. Final outputs come out into 32 bits fixed-point.

2.4 CORDIC operations

Embedded software of the relay was implemented by applying COordinate Rotation DIgital Computing (CORDIC) algorithm [6] to compute main operations of RMS, phase and protective algorithm based on [3]. Including the trigonometric functions and various arithmetic operations can be calculated by CORDIC. It is not an approximated method to reduce amount of computation. CORDIC algorithm has advantages in terms of efficiency and accuracy. Table 1 shows the CORDIC operations for required arithmetic operations in this study. Each arithmetic operation can be calculated by using specific CORDIC operational modes and it might need additional constant multiplication or division based on fixed-point depending on the type of operation. In addition, normalization and correction operations are required

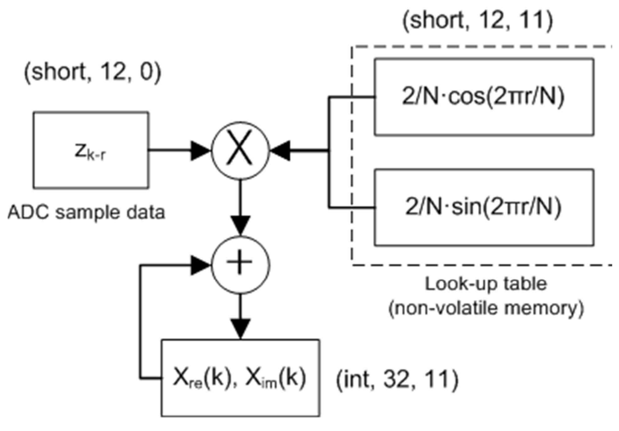


Fig. 4 Fixed-point design for convolution in DFT filter

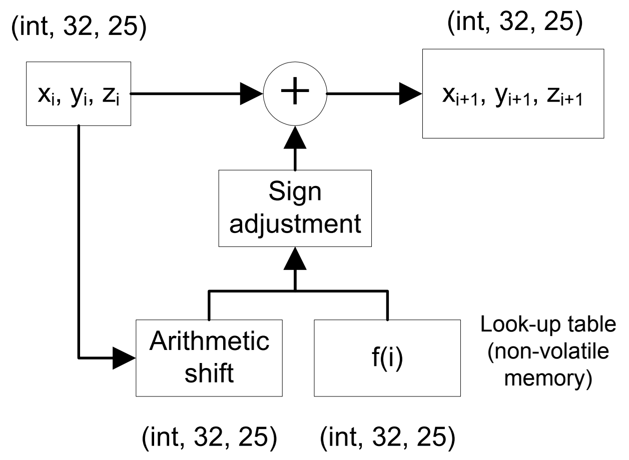


Fig. 5 Fixed-point design for CORDIC operations

to guarantee the convergence of CORDIC iteration depending on the type of operation [3]. Fig. 5 illustrates the schematic of fixed-point design for main CORDIC operations. Normalized x, y, z for the convergence, are the inputs of the CORDIC algorithm. Its precision ranges up to 2^{-25} and the number of the maximum iterations is 32.

2.5 Measurement for RMS value and phase

Fig. 6 shows the RMS value and phase measurement process. Circular-vectoring mode of CORDIC operation is used for the measurement of RMS value as listed in Table 1. L-bit shift operation will be added to imply proper correction after output z has been obtained from normalized input x and y. The phase measurement gets the output after the similar process of RMS measurement. However, the phase measurement does not need normalization and correction of the inputs and outputs. As a result, RMS value and phas can be obtained after executing CORDIC algorithm only once.

2.6 Fixed-point design for protective algorithm

Table 1 CORDIC operational modes for protective relay

Arithmetic Operation	Block	CORDIC operation[3,6]
$\cos(z)$	Filter	Circular-Rotation
$\sin(z)$	Filter	Circular-Rotation
$\sqrt{(\mathrm{x}^2+\mathrm{y}^2)}$	RMS meter	Circular-Vector
$\tan^{-1}(\frac{y}{x})$	Phase meter	Circular-Vector
e^z	protective algorithm	Hyperbolic-Rotation
ln(w)	protective algorithm	Hyperbolic-Vector

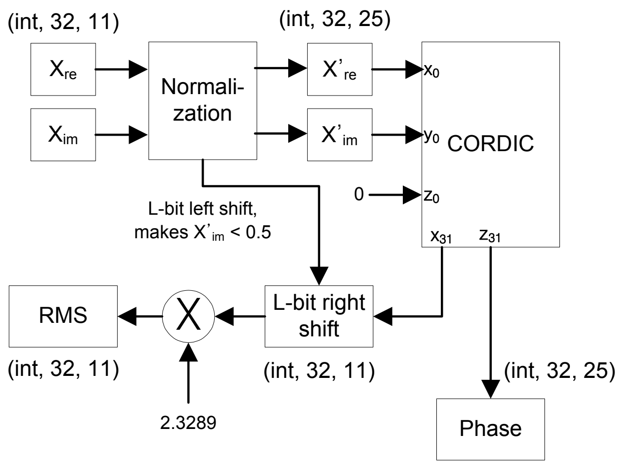


Fig. 6 Fixed-point design for measuring RMS and Phase

We adopted general inverse-time over current protection algorithm. Eq. (1) is a characteristic equation of the time-current curve.

$$\mathrm{T} = \frac{K}{I_r^p - 1} \tag{1}$$

K and *p* are the constant representing the cut-off characteristics of the relay, I_r is the ratio of reference current to the fault current. Since *p* is a real number including fractional number, the key point of the optimization is reduction in calculation time for exponentiation. There have been an approximated method to estimate quickly by referencing lookup table (LUT) which implies the characteristic equation. However, it causes a trade-off relation between the errors of operation result and the degree of flexibility in selecting curves [3,7,8]. In this paper, the calculation of the exponentiation are implemented by using CORDIC algorithm based on the result of the study [3].

$$I_r^p = e^{\ln I_r^p} = e^{p \cdot \ln I_r} \tag{2}$$

Eq. (2) can be calculated by Hyperbolic-Vector and Hyperbolic-Rotation mode. Fig. 7 depicts the fixed-point design for protective algorithm including exponentiation. After the calculating ratio of reference current (I_{ref}) to RMS value (I_{RMS}), algorithm mainly consists of logarithm and exponential function. Exponentiation requires the pre-processing and the post-processing of CORDIC operation to guarantee the convergence in similar way as mentioned in section 2.5. Pre-process executes the normalization including L-bit shifts and extracting fractional part, and the post-processing provides applying correction. As shown in Fig. 7, two CORDIC operations, five fixed-point multiplication, a division and three addition/subtraction are executed during exponentiation. Since the exponentiation is calculated in run-time without using previously stored information for time-current curve of Eq. (1), curve selection can be easily

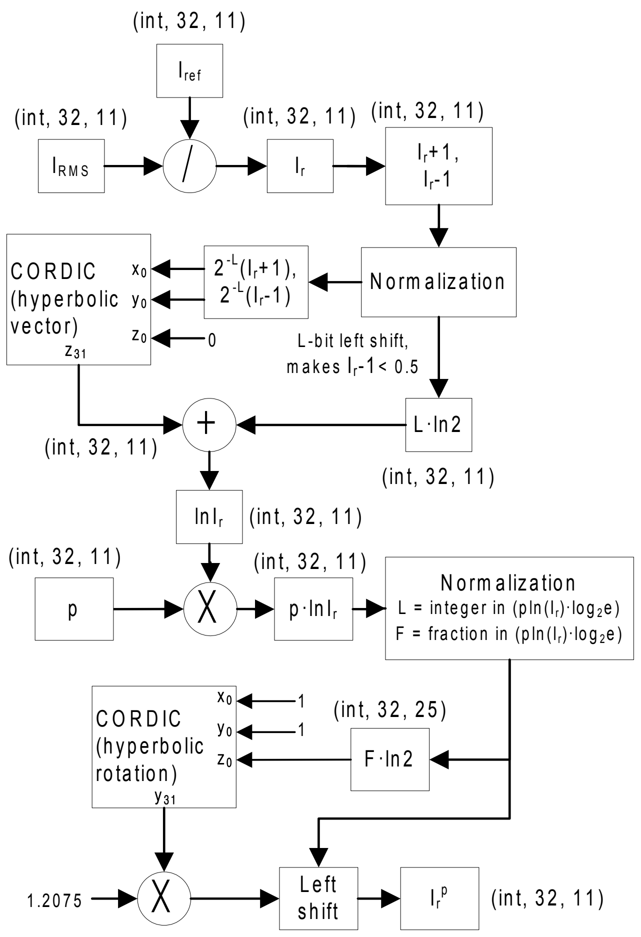


Fig. 7 Fixed-point design for exponentiation in inverse-time protection

configured by only setting the parameters *K* and *p* within the designated precision.

3 Experimental results

In this section, we discuss the complexity of operations. The proposed fixed-point software design in section 2 was ported into TMS320C6416 DSP. We also compared the both implementation results for fixed-point and floating-point DSPs. We will discuss the multi-channel supporting capability for the proposed design method.

3.1 Comparison of execution time and area complexity

Table 2 shows summarized results of relay software that have been implemented by the fixed-point and floating-point codes, running on two different DSPs with or without floating-point hardware support. Case 1 represents our work proposed in this paper. Case 2 implements the floating-point operation by using standard libraries provided in the vendor provided cross-compiler where the target processor is TMS320C6416. Case 3 implements floating-point operation by using TMS320C6727 which supports floating-point hardware unit. Consequently, case 1 is 53 times faster than case2 in terms of total execution time and four times better than case 2 in the code memory size. Specifically the execution time of measurement and protective algorithms is decreased by 97% due to the use of CORDIC algorithm. In addition, Case 1 is faster than case 3 about 20 times in execution time and saving 50% code size. CORDIC algorithm requires two LUTs which have 32 entries based on 16 bits, but it requires only 128 bytes in addition. It can be neglected for the overall code size.

3.2 Expandability of sample rate and channels

In Table 2, the input data rate is 128 samples/period. Fig. 8 illustrates the maximum samples per period without loss of data in real-time when the number of simultaneous input channel is increased as 4, 8, 12 and 16. FXP_6416, FLP_6416 and FLP_6727 are case 1, case 2 and case 3 respectively. We assumed that the digital filter of FLP_6416 and FLP_6727 use the same structure as in FXP_6416.

Fig. 9 illustrates the maximum samples per period with same conditions of Fig. 8 when the number of simultaneous input channel increases as 24, 32, 64 and 128. From the results of Fig. 8 and Fig. 9, we can see the proposed software in fixed-point design enlarges the number of multi-channel source and samples per period drastically. It also leads more cost reduction in the system integration.

Table 2 Execution time result of protective relay

case	1	2	3	4
Protective relay Software	Fixed-point	Floating-point	Floating-point	[5]
DSP	TMS320C 6416	TMS320C 6416	TMS320C 6727	TMS320(32
Floating-point support	X	X	O	O
frequency [MHz]	500	500	300	50
Samples per period	128	128	128	24
Digital filter [us]	0.7	140.0	54.3	10.3
Measurement [us]	2.0	200.6	69.1	
Protective relay Operation [us]	4.1	22.8	7.0	5.6
Total execution [us]	6.9	363.1	130.6	-
Code size [KB]	4.1	16.0	9.4	-

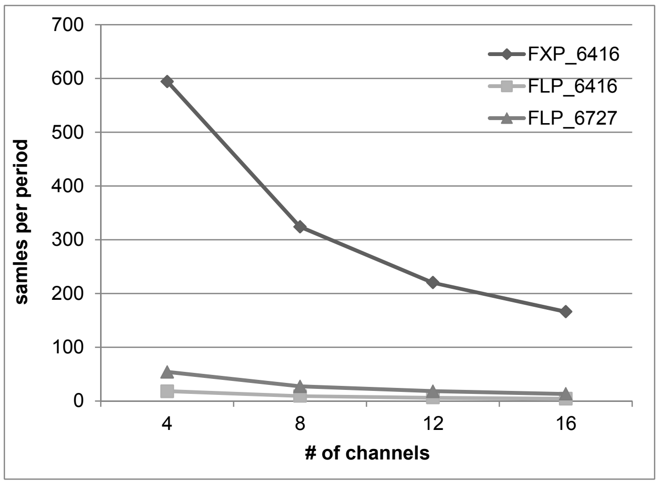


Fig. 8 Max. number of samples per period according to concurrent input channels numbers

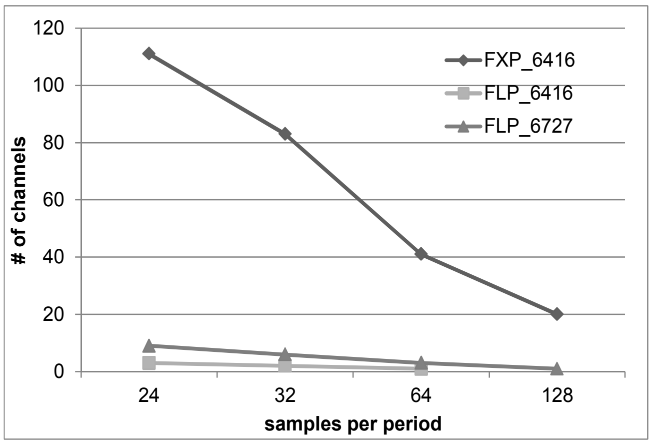


Fig. 9 Max. number of input channels according to the given samples per period

4 Conclusions

We reduced the amount of computation and memory usage by implementing fixed-point design for the protective relay in commercial DSP against conventional floating-point software. In particular, we optimized the measurement and protective algorithms which have high complexity by applying CORDIC algorithm. As a consequence, the execution time of software was improved significantly. Accuracy of the proposed software is guaranteed in designed precision, without using approximated curve models and extrapolation. The proposed software can provide more stable protective relaying by enhancing pre-processing operations such as digital filter. Further, our work is possible to reduce the number of processors. It is very advantageous in terms of cost, since it requires small amount of computation.

5 References

[1] Korea Electric Association, "2011 year book," 2011.

[2] P.E.K. Tang, "Design and implementation of IEC61850 for power generating plant protection, control and automation," Proc. of POWERCON, pp.1-7, 2010.

[3] J.K. Park, J.T. Kim and Myong-Chul Shin, A CORDIC-based digital protective relay and its architecture, Microelectronics Reliability J., Vol. 49, No.4, pp.438-447, 2009.

[4] AREVA T&D, "Network Protection & Automation Guide", www.areva-td.com, 2002.

[5] H-S Suh and G-B Kweon, "A Study on Design of Digital Protective Relay for Transformer Using a DSP," J. of KIIEE, Vol.17, No.6, pp.39-46, 2003.

[6] B. Parhami, "Computer Arithmetic: Algorithms and Hardware Designs," Oxford University Press New York, 1999.

[7] G. Benmouyal, "Design of a Digital Multi-curve Time-overcurrent Relay," IEEE Trans. on Power Delivery, Vol.6, No.2, pp.656-665, 1991.

[8] J.C. Tan, P.G. McLaren, R.P. Jayasinghe and P.L. Wilson, "Software model for inverse time overcurrent relays incorporating IEC and IEEE standard curves," IEEE CCECE 2002, pp.37-41, 2002.

Heuristics for Conversion Process of GPU's Kernels for Multiples Kernels with Concurrent Optimization Divergence

José Ritomar Carneiro Torquato[1], **Esteban Walter Gonzalez Clua**[1]
[1] Institute of Computing, Federal Fluminense University, Niteroi, RJ, Brazil

Abstract - *Graphics Processing Units have been created with the objective of accelerating the construction and processing of graphic images. In its historical evolution line, concerned with the large computational capacity inherent, these devices started to be used for general purposes. However, the design of the GPUs don´t work well with divergent algorithms, mainly conditionals and repetitions. In this work we present a strategy for finding the divergence root of the kernels and try to deduce alternative solutions, decomposing them into concurrent kernels. We developed mechanisms for the user in order to easily readapt his code and take advantages of architectures that support concurrent kernels*.

Keywords: Divergence; Concurrents Kernels; Warps; GPGPU.

1 Introduction

GPUs (Graphics Processing Units) were designed to make to process polygons, and they have a peculiar feature: the same sequence of operations to different data. Following its historical evolution, current GPUs keep following this paradigm in its architectural models. In this style of execution, all the hardware involved executes the same instruction, before moving on to the next one. In fact the model brings benefits by reducing the cost of production and offering an optimized memory access. The new architecture Kepler GK110, is called by the NVidia "The next generation of GPUs" and still uses the same concepts of multiprocessor streams [1]. We believe that this architecture remains in awhile because, in practice, this restriction is what makes technologically feasible to massively parallel architecture.

Diverging code is defined as the fact that a stream of code executed in a parallel environment can take different directions in each of its instances. In Single Instruction Multiple Thread (SIMT) architecture, occurring divergence, all statements that do not follow the same path are forced to wait at the point of divergence. It is noteworthy that this is not a limitation of the solution, but the hardware architecture.

In this work we identify strategies that can minimize the effects of divergence in execution time of parallel applications. The optimization algorithm is currently the main and most efficient way to reduce the impact of divergence, forcing the implementation to follow a single path. A commonly adopted technique consists in separating the code into two parts, running a first leg and then the other. This was the only way to deal with this problem on GPUs until a little time ago. Although it is shown effective, in many cases the time dependence of data makes this solution inappropriate. With the Fermi GPUs series, Nvidia started implementing concurrent kernels. We present a new technique to divide a code divergence by using this technology. Preliminary tests showed that we can reduce the divergence by creating concurrent kernels.

In this paper we identify the mechanisms used to reduce the impact of the difference in execution time of parallel algorithms. Furthermore, we propose the use of concurrent kernels based on new generations of GPUs, such as Kepler, as an alternative in treating the problem.

The remainder of the paper is organized as follows. Section 2 provides background on GPU's evolution and Unified Architecture. Section 3 describes the divergence problem. Section 4 presents the optimizations of divergence, evaluation methodology and results. Section 5 discusses related work, and gives directions for future work.

2 Unified architecture

The first video cards created were simple and the severe hardware limitations made unimaginable graphics processing by them. Following the chronological evolution emerged raster, fixed function and programmable devices. These last one brought pixels and vertex processors, able to treat, only and respectively, pixels and vertices. At that time, there were not multi-core CPUs so the GPU was seen as an alternative to increase the processing power in specific tasks. Thus, researchers from different areas began to "consider" data input of mathematical calculations as vertices and pixels, making the use of these processors in solving mathematical equations possible. For the first time a GPU was used with general-purpose, giving rise to GPGPU (General Purpose GPU)

Processors of vertices and pixels did nothing beyond their specific tasks, increasing the interest in the computational power of devices, as well as the inconvenience of having to map all that was wanted in vertices and pixels. In addition, the processors were built only to the treat their structures, and an application that performed more vertices or pixel would leave the other processors idle.

Nvidia proposed a unified architecture in their cores [2], creating a new architecture called CUDA (Compute Unified Device Architecture). Some advantages over previous architectures CUDA GPUs are:

- Memory random access: access to any region of memory to read and write;
- Manageable user- Cache: threads can cooperate reading and writing data in shared memory and any thread can access the shared memory of its block ;
- Low learning curve: simple extensions of C language, without requiring knowledge of graphics or graphics APIs.

Programming models for GPU (as CUDA and OpenCL) are designed to allow legacy programs to take advantage of new features in a transparent way. In other words, programs originally written for a particular architecture are scalable to the following architecture. Also, allow the use of heterogeneous systems, thus CPUs and GPUs are distinct and separate memory devices. Each of them performs the function for which they are best prepared.

CUDA facilitates programming since it allows developers to focus on developing their algorithms without the need to learn language specific mechanisms. Instead, it provides a minimum length of the C / C + + to construct parallel applications.

3 Divergence problem

During the execution of the code by the GPU, each decoded instruction is sent to the scheduler. They remain queued until despatch in execution units, often called warps. This approach reduces the time for loading and decoding of instructions by N execution units, however, it does not require instructions to follow the same path. If there would be a piece of code in which some instruction keep on processing, they execute while the others wait for a different point of divergence [3][4]. Thus, a conditional statement can result in divergence when it is based on values that are particular to the specific thread [5].

For example, one if instruction may cause the thread to follow different paths, or, similarly, a loop may cause divergence whether the conditions are based on the thread's own values.

To demonstrate the impact of the divergence, we must consider the following code, similar to what occurs in problems of reducing vectors:

```
01 if (threadIdx.x < 32)
02 {
03     if (threadIdx.x < 16)
04     {
05         if (threadIdx.x < 8)
06             func_a1();
07         else
08             funca2;
09     }
10     else
11     {
12         func_b();
13     }
14 }
```

Listing 1: Divergence problem demonstration.

We will use the code in Listing 1 to illustrate how the divergence can affect the efficiency. Its execution results in data that are displayed in Figure 1.

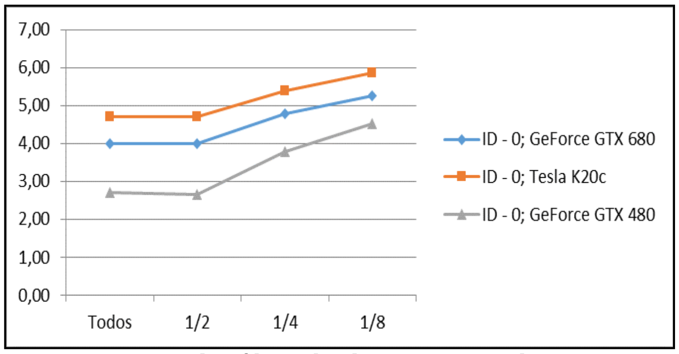


Figure 1: Sample of how the divergence may have strong impact on performance

The first line of code in Listing 1 eliminates all threads of the block except the first 32 threads (first warp), the one we will use for our analysis. This does not result in any difference within a specific warp. The other warps of the block simply do not scale to this session and wait.

Analyzing only the first warp, we observed that in line 3 the test `threadIdx.x < 16` is done, what breaks the warp is carried out exactly in half. In the graph first transition is noticed, this operation does not result in actual divergence since the CUDA kernels are organized in banks of 16 cores, not 32. Thus, the scheduler cyclely sends instructions to two or more sets of 16 cores and the paths of true and false conditional statement run on cores from different banks.

In the subsequent step, the threads 16 to 31 call `func_b` function (line 12), however, threads 0 to 15 have another condition associated (line 05). Therefore, this time is not based on half of the warp, but in a quarter of it. So, we need a minimum of 16 threads for scheduling. Thus, the first eight threads will proceed to the function `func_a1` while the remaining eight (8.. 15) await.

The functions `func_b` and `func_a1` will continue their instructions independently and shoot the second half of the warps. This is less efficient than the search for a single statement, but nevertheless, better than a sequential execution. Eventually `func_a1` will finish and `func_a2` will start the threads 0-7. Meanwhile, `func_b` might also have been completed.

Analyzing the best result different levels of divergence are perceived. The first one is great, without divergence. The second one differs based on half of the warp but does not result in real divergence, since they run in parallel. Dividing the first half of warps into two groups, these should run in series, as they will expect a stretch to be finished and only then the next starts. Once again, dividing the first group in a total of four paths they will also result in a serial execution case.

4 Optimization of divergence

4.1 *Naïve* Test

An example of simple demonstration was created in order to highlight the importance of separating different kernels and create separate concurrent Kernels. Considering that the program will receive, as input, a vector of k positions filled with N numbers, which alternate between large and small values, as shown in Figure 2 below:

i =	*0*	*1*	*2*	*3*	*4*	*5*	*...*	*k-4*	*k-3*	*k-2*	*k-1*	*k*
N =	5	5000	5	5000	5	5000	5	5000	5	5000	5	5000

Figure 2: Input of the First Demonstration Kernel.

In the next step, our test program will run on a kernel, shown in Listing 2, a repetition by N times (with N being the value of the position i of the input vector) and the input vector is stored in global memory.

In our first test, we have a *Naïve* approach, which reads data sequentially. We will have half the cores using a small value and the other half using a large value (in the same block) and it is hoped that the cores running the repetition with the highest number of iterations dictate the overall runtime.

Next, we used an index thread strategy, forcing a block to take the odd and another the even numbers. Our objective is to allow two kernels to perform the same function concurrently. Thus, we come to the result shown in Listing 3.

The kernels shown in listing 2 and 3 are equal in function, however, we put some "intelligence" in the while loop within lines 5 and 15 in Listing 3 in order to force these kernels specifically deal with values from the same class (all small or all large). Thus, the `kernel02a` will only treat small values of Figure 2 while the `kernel02b` treats the others.

```
__global__ void kernel01(int *a)
{
    int i = a[threadIdx.x];
    __shared__ int k;
    while (i > 0){
        i--;
        k+=i;
    }
}
```

Listing 2: Initial kernel.

```
01  __global__ void kernel02a(int *a)
02  {
03      int i = a[threadIdx.x];
04      __shared__ int k;
05      while (((i % 2 == 0) && i > 0)){
06          i--;
07          k+=i;
08      }
09  }
10
11  __global__ void kernel02b(int *a)
12  {
13      int i = a[threadIdx.x];
14      __shared__ int k;
15      while ((i % 2 != 0) && (i > 0)){
16          i--;
17          k+=i;
18      }
19  }
```

Listing 3: Concurrents Kernels.

The Table 1 summarizes the execution times, and Figure 3 shows these results graphically comparing them:

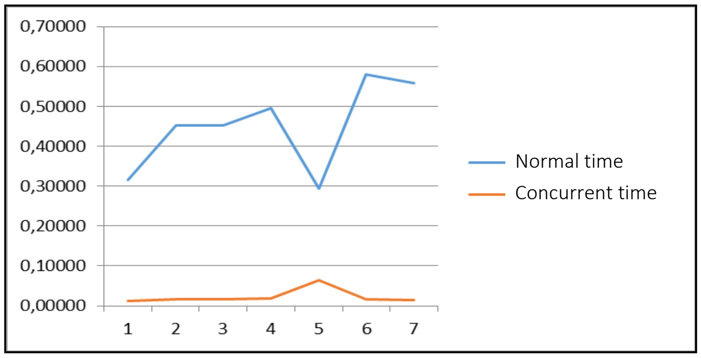


Figura 3: Initial x Concurrents Kernels

Host	Device	Normal Time	Concurrent Time
K10 Motorhead	GeForce GTX 680	0,31466	0,01200
K10 Motorhead	Tesla K10.G1.8GB	0,45114	0,01722
K10 Motorhead	Tesla K10.G1.8GB	0,45142	0,01702
K20 - Clash	Tesla K20c	0,49501	0,01869
K20 - Clash	GeForce GTX 680	0,29312	0,06470
Orange Lab Pos	GeForce GTX 480	0,57942	0,01533
Orange Lab Pos	GeForce GTX 480	0,55824	0,01523

Table 1: Comparison of execution times in the first kernel demonstration

4.2 *Sum Reduction*

A reduction algorithm extracts a single value from a matrix, calculated by comparing every element of it. The reduction may be to sum, to the maximum or minimum values, of the components. These algorithms share the same structure. A reduction may be performed sequentially stepping through each element of the array. When an element is visited, the action to be taken depends on the desired reduction. To sum reduction, the current value is accumulated [1].

Listing 4 shows a CUDA kernel for reduction of sum. The input matrix data were placed in main memory, the array was divided so that each block CUDA reduce a portion of the original matrix. The reduction will be made in device, using the shared memory, in other words, there will be a shared variant where the partial sums will be saved. Each iteration of the line 6 loop is a round of reduction. The syncthreads () statement in the for loop ensures the necessary timing for the performace of the previous iteration and to prepare the threads for the next iteration. Each round of implementation of even elements will contain the partial sums of each pair after iteration until all sums are performed.

The kernel of Listing 4 has caused divergence of the iteration loop of line 6. In this place only threads with even threadIdx.x values perform the sum due to the condition imposed on line 9. Such divergence can be reduced with a change in the algorithm.

```
01  __global__ void sumReduceD(const Utype *a, Utype *sum)
02  {
03      __shared__ int partialSum[arraySize];
04      unsigned int t = threadIdx.x;
05      partialSum[t] = a[t];
06      for(int stride = 1; stride < blockDim.x; stride *= 2)
07      {
08          __syncthreads();
09          if(t % (2*stride) == 0)
10          {
11              partialSum[t] += partialSum[t+stride];
12              sum[0] = partialSum[t];
17          }
18      }
19  }
```

Listing 4: Divergent Reduction Sum

```
__global__ void sumReduceN(const Utype *a, Utype *sum)
{
  __shared__ int partialSum[arraySize];
  unsigned int t = threadIdx.x;
  partialSum[t] = a[t];
  for(int stride = blockDim.x >> 1; stride > 0; stride >>= 1)
  {
    __syncthreads();
    if(t < stride)
    {
      partialSum[t] += partialSum[t+stride];
      sum[0] = partialSum[t];
    }
  }
}
```

Listing 5: Optimized Sum Reduction.

The modified kernel in Listing 5 adds elements that are in the middle of a section, rather than adding neighboring elements. At the end of the first iteration, the sum is stored in the first half of the array. At each iteration of the loop the overall operation is divided by 2 by shifting step by one bit to the right, an economical way to perform division by 2. Note that the kernel in Listing 5 also has an `IF` (line 9) which means that it will still have divergence, however, the amount of threads that execute this instruction is minimal compared to the previous case.

To verify the efficiency of concurrence we have unbundled the kernel of Listing 5 in two others, each of them responsible for performing half the reduction of sum.

```
__global__ void sumReduceC1(const Utype *a, Utype *sum, long offset)
{
    if (threadIdx.x < offset)
    {
        __shared__ int partialSum[arraySize];
        unsigned int t = threadIdx.x;
        partialSum[t] = a[t];
        for(int stride = blockDim.x>>1; stride > 0; stride >>= 1)
        {
            __syncthreads();
            if(t < stride)
            {
                partialSum[t] += partialSum[t+stride];
                sum[0] = partialSum[t];
            }
        }
    }
}

__global__ void sumReduceC2(const Utype *a, Utype *sum, long offset)
{
    if (threadIdx.x >= offset)
    {
        __shared__ int partialSum[arraySize];
        unsigned int t = threadIdx.x;
        partialSum[t] = a[t];
        for(int stride = blockDim.x>>1; stride > 0; stride >>= 1)
        {
            __syncthreads();
            if(t < stride)
            {
                partialSum[t] += partialSum[t+stride];
                sum[0] = partialSum[t];
            }
        }
    }
}
```

Listing 6: Concurrent Sum Reduction

Host	Device	Normal Time	Divergent Time	Concurrent Time
K10 Motorhead	ID - 0; GeForce GTX 680	0,01210	0,01523	0,01411
K10 Motorhead	ID - 1; Tesla K10.G1.8GB	0,01834	0,02387	0,02256
K10 Motorhead	ID - 2; Tesla K10.G1.8GB	0,01846	0,02390	0,02214
K20 - Clash	ID - 0; Tesla K20c	0,02458	0,04464	0,03117
K20 - Clash	ID - 1; GeForce GTX 680	0,01168	0,01533	0,01440

Table 2: Comparison of execution times on Sum Reduction Algorithm

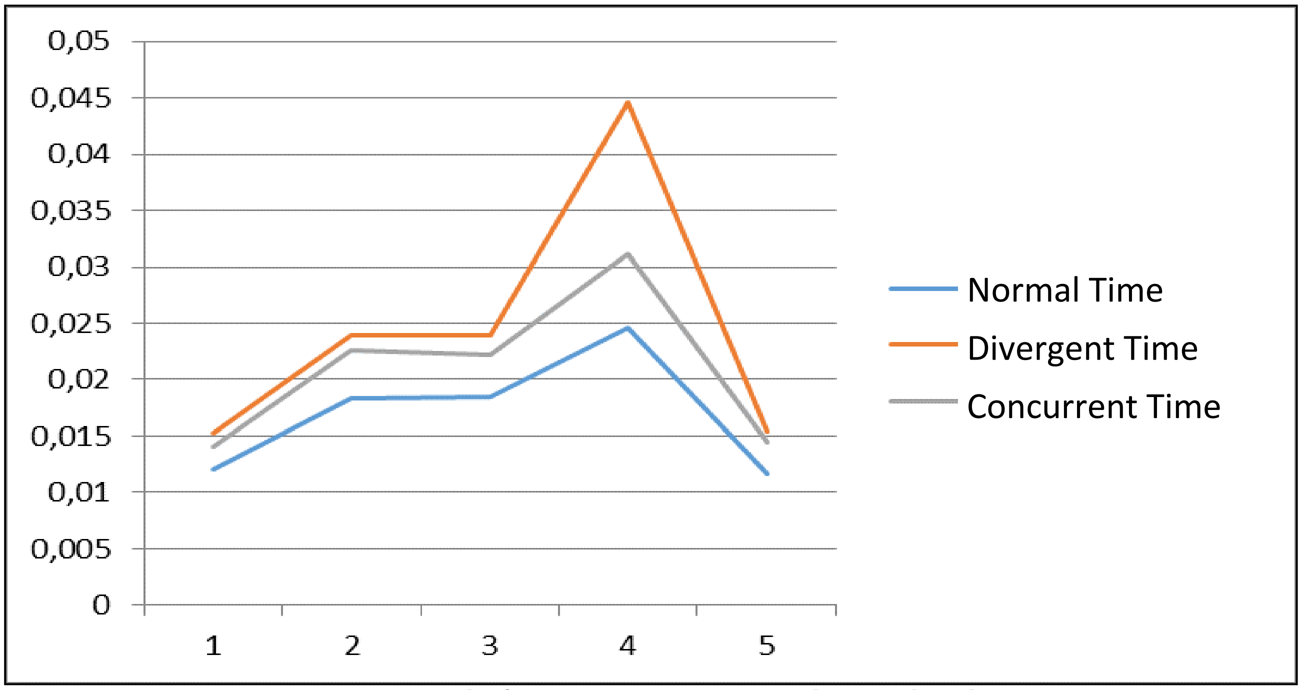


Figure 4: Graph of execution times on Sum Reduction Algorithm

The `sumReduceC1` and `sumReduceC2` kernels in Listing 6 run concurrently, each being responsible for elements of the two halves of the array, delimited by the offset variant. The Table 2 below shows the times taken in the implementation to reduce the sum of an array with 1024 elements.

5 Conclusions and Futurework

Here we present the problem of disparity in kernels, ie, the divergence that is the result of the emergence of distinct branches of implementation due to conditional or repetitions present in algorithms.

In this paper we propose a new approach to minimize the effects of them through the use of concurrent kernels and found satisfactory results that justify further study on the topic.

As future work, we propose algorithms to analyze patterns in two suites used for investigation of parallel applications. The Rodinia [6] suite is often used to measure multi / many core and parallel data applications, covering a wide range of parallel communication patterns, among them applications of medical imaging, bioinformatics, physical simulation, image processing, etc.. The Parboil [7] suite brings a suite of applications useful to study the performance of the architecture and compilers.

It is also intended to analyze the Cetus [8] which is a source code translator for multicolored infrastructure that fosters research in architecture for compiler optimizations with automatic parallelization.

Thus, we seek to list a series of strategies to map different types of problems, enabling transform a single kernel into n concurrent kernels. We hope to contribute to a set of heuristics that migth assist in mapping, preferably in an automatic way and with less divergence as possible.

6 References

[1] D. B. Kirk and W. W. Hwu, Programming massively parallel processors: a hands-on approach. Morgan Kaufmann, 2010.

[2] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, “NVIDIA Tesla: A unified graphics and computing architecture,” Micro, IEEE, vol. 28, no. 2, pp. 39–55, 2008.

[3] S. Cook, CUDA Programming: A Developer’s Guide to Parallel Computing with GPUs. Newnes, 2012.

[4] B. Coutinho, D. Sampaio, F. M. Q. Pereira, and W. Meira Jr., “Divergence Analysis and Optimizations,” in Proceedings of the 2011 International Conference on Parallel Architectures and Compilation Techniques, 2011, pp. 320–329.

[5] T. D. Han and T. S. Abdelrahman, “Reducing branch divergence in GPU programs,” in Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units, 2011, pp. 3:1–3:8.

[6] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, “Rodinia: A benchmark suite for

heterogeneous computing," in Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on, 2009, pp. 44–54.

[7] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W. M. W. Hwu, "Parboil: A revised benchmark suite for scientific and commercial throughput computing," Cent. Reliab. High-Performance Comput., 2012.

[8] H. Bae, D. Mustafa, J.-W. Lee, Aurangzeb, H. Lin, C. Dave, R. Eigenmann, and S. Midkiff, "The Cetus Source-to-Source Compiler Infrastructure: Overview and Evaluation," Int. J. Parallel Program., vol. 41, no. 6, pp. 753–767, 2013.

Fault-Tolerant Databases: Assessing the Risks and Costs

Berkay Surmeli[1], Martin Maskarinec[1], and Kathleen Neumann[1]
[1] School of Computer Sciences, Western Illinois University, Macomb, IL

Abstract - *This paper presents the position that a formula can be used to assess the risks of a distributed datacenter. This formula may be compared against benchmarks to assist an administrator in making decisions concerning server upgrades, datacenter placement, etc. It can also be used to help the administrator decide what data to replicate and where to place the copies.*

Keywords: Distributed Databases. Fault-tolerance, Fault-tolerance measurement.

1. Introduction

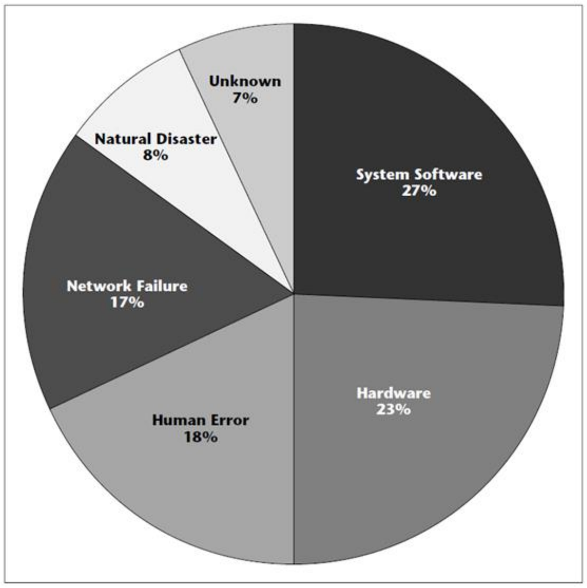
In the globalized world, companies, based on the proximity to the resources, markets and transportation hubs, consider opening branches and data centers in different locations over the world. These datacenters usually house large databases with data needed by their other branch locations. Hence, establishing the optimal distributed database system and the optimal configuration of redundancy and replication presents significant challenges.

Today, datacenters and computer systems are expected to be available all the time. The potential loss of revenue, customers and reputation is more and more dependent of the reliability of your datacenters. For some businesses, even small amounts of downtime may cost millions of dollars. As we now have global enterprises which support users, clients or customers in multiple time zones, it is no longer possible to bring systems down at night for extended periods of time to do maintenance and upgrades. Likewise we need to plan for ways to compensate for unplanned outages. In distributed database systems, when one site experiences an unplanned outage, there will be no way to access the data from that site, unless we have made previsions ahead of time for data replication. When one site or server is down, the replica of the data is used in another server or location, so that the system continues to function and respond in such a way that the users do not notice any problems.

In this paper we are proposing a formula that analyzes the risk to the servers. This analysis is calculated based on a variety of weighted factors such as an analysis of error logs for the servers, assessing reliability of the hardware components based on age and environmental factors, the size of the data, the frequently of data usage, and the data distribution model employed in the system. This formula may then be used to determine the risk of the system and to aid in the decision as to whether or not to replicate data, and if so, what to replicate and where.

2. Risks

The goal of any server or datacenter is to have high availability. The common measurement of “nines” has been debated over the years. Nonetheless, the closer to 100% availability that a server or datacenters obtains, the more satisfied the users become. The risks of server or datacenter outages come from a broad range of possibilities. The following graph shows the breakdown of the cause of outages [1]:



3. Costs of Not Achieving High Availability

The cost and impact of not achieving an acceptably high amount of availability varies greatly depending on the type of business or entity involved. While unavailability of data may cost millions of dollars for some businesses, other outages may have zero cost to the entity – it all depends on the type of entity impacted. For example, if the entity impacted is a firm that trades stocks or a large retailer during heavy holiday shopping times, the measurable monetary loss would be great. However, if the entity is a repository for information and the availability was lost during a non-peak time, then the loss was negligible. There have been a variety of surveys and studies done that study this type of impact, such as those done by the Ponemon Institute [2]. However, even in the case where there was no monetary loss due to data not being available, there are unmeasureable consequences, such as reputation, that are still at stake.

4. Minimization of Risks and Costs through Database Replication

Distributed databases, where all or portions of the data are replicated on other servers or locations, have long been viewed essential in achieving an acceptable level of data availability. The costs associated with managing the entire system are often outweighed by the potential loss associated by unavailability. Synchronous replication and asynchronous replication are options that exist and are considered in the overall calculation of risk.

4.1 Synchronous Replication

This replication enables zero data loss disaster recovery by ensuring that the data stored at the secondary storage site is the exact mirror image of the data at the primary data site. In this method of replication, each update must be recognized and confirmed at both the primary site and secondary site before the application can continue production. Thus, the system ensures that the secondary site is always in sync with primary data center. This enables the secondary site to take over production immediately in case there is any disruption at the primary site. However this method of replication is very expensive in terms of time. Distance is also a major drawback of synchronous replication. The distance between primary centers and backup centers is restricted due to use of fiber channels which can only extend to 200km [4].

4.2 Asynchronous Replication

With asynchronous replication, updates are propagated to the copies periodically, not as they are made. One major advantage of this is that it is much more efficient in terms of transmission time – periodic updates mean less network overhead. In addition to this, asynchronous copies can extend to any distance without affecting the propagation. This means secondary sites can be located thousands of miles away from the primary site, ensuring that secondary data is away from any likely disaster region.

The greatest disadvantage of asynchronous replication is the time lag between data being stored at the primary site and the remote site. This may mean that transactions and data not replicated at the time of disaster will be lost. In the event of any unplanned outage, data on the secondary storage may not be current. [3]

The next subsections detail two approaches to permitting asynchronous replication. Which is best would be somewhat system dependent; the choice would be left to the individual systems administrator.

4.2.1 Primary Copy Asynchronous Replication

In this approach, one copy of the data is deemed the “primary copy” and always must be current. The other copies will not be updated immediately; the primary copy server will push out the updates to the remote copies periodically. This has the advantage of using a simple communication model. In a case when a server (either a server holding a primary copy or one holding a secondary copy) has a failure, the updates needed may be held in stable queues at the other sites until this server comes back up. This allows an element of fault tolerance to be built into the basic idea of replication [3].

4.2.2 Update everywhere

In the Update everywhere technique, any copy may be updated at any time. Periodically, the servers will exchange their updates with each other. This has the advantage of lower communication cost during updates (as compared with primary copy replication). However, it requires a much more complicated synchronizing algorithm, since different copies may have updated the same data.

5. How to Determine Where to Replicate Data

Determining where to replicate all, or a portion, of a distributed database can be based on a variety of factors. Depending on the entity, the factors will include such items as: the proximity to primary/heavy users, the reliability/capacity of communication networks between sites, environmental factors such a typical seasonal weather related concerns, age/reliability of equipment at replicated site, comfort level of potential success of disaster recovery efforts, etc.

5.1 Hardware Risk Assessment

The follow are the factors that will help determine the hardware risk level of the server or data center in the distributed environment. These factors also help in determine the site for data replication.

a) Determine the access/query and update frequency for each table in the database. Likewise the number of applications that access each table, and the frequency with which those applications runs.

   Analyzing this data will give an indication of which portions of the database have the highest need to be replicated.

b) Determine the amount of time, and the number of times, a server or datacenter had an unplanned outage within a given interval.

c) Determine the age and usage of the most vital hardware like CPUs, hard drives, fans, memory, etc.

5.2 Environmental Risk Assessment

Environmental risks should also be considered when placing datacenters and/or replicated parts of databases. Areas that are prone to hurricanes, earthquakes, floods, extreme hot or cold temperatures, etc. are less desirable locations for the placement of servers or data centers that need to have high availability.

While some of the above-mentioned environmental risks are hard to predict, and therefore assess, there are some environmental risks that we can assess. For example, we know that there is a high probability that at some point there will be an interruption to the power supply to a particular server or an entire data center. The cause of the interruption could be the result of a weather related event, or it could be accidental (for example a miscalculation of where a backhoe should dig during a construction project), or the interruption could be part of scheduled maintenance or upgrade to the electrical system. Hence, we also need to take into consideration other factors such as:

a) The presence and reliability of backup systems such as Uninterrupted Power Supply (UPS) devices. Likewise a determination on how frequently UPS devices are tested for functionality should be considered. For example, a datacenter has no protection against data loss nor the opportunity for graceful shutdown if the UPS devices also fails due to dead batteries or faulty failover to a fuel powered generator.

b) The presence of appropriate fire protection systems, water protection systems, cooling systems, alert notification systems, etc. The frequency of testing to make sure that these systems are fully functional also needs to be considered.

As an example, while we may not be able to assess the frequency with which ice sheets, created by an unusually bad winter storm, will fall from the roof of building and damage air conditioning units on the ground, we can minimize the likelihood that UPS unit will also fail due do a short in the failover system. Likewise, we can minimize the likelihood that the alert system will also fail due to an error in the script that indicates who should receive a text message or a pager alert.

5.3 Formula to Calculate Risk Level

A weighted formula is being developed to help determine the risk level for either an individual server or an entire datacenter. The formula takes to account the following:

1. Availability/Response time of the server/ datacenter. This would include the percent of time the server/datacenter has been available in the last twelve months. Also, this would need to factor in size and frequency of data access as well as the replication scheme used.

2. Age of equipment. The value used in the calculation is the following on a sliding scale:
    a. Less than 1 year old: 100 pts
    b. Less than 2 years old: 80 pts
    c. Less than 4 years old: 60 pts
    d. Older than 4 years: 0 pts

3. Assessment of adequate cooling facilities, water protection systems and fire protection systems. The maximum points are 100 when everything is found to be adequate.

4. Assessment of reliability adequacy of UPS systems. The maximum points are 100 when it is found the UPS is completely adequate.

5. Assessment of other environmental risks as discussed in section 5.2. The maximums points are 100 when it is found that there are virtually no other environmental risks.

    The weights of these four variables are:
    1. Availability: 25%
    2. Age of Equipment: 10%
    3. Assessment of cooling, water and fire systems: 20%
    4. Assessment of UPS: 40%
    5. Assessment of other Environmental Risks: 5%

Example:

Availability/Response time in the last year: 90 * 25% = 22.50
Age of Equipment: 15 month old server: 80 pts * 10% = 8
Assessment of Cooling/Fire/Water protection: 95 pts * 20% = 19
Assessment Electrical/UPS protection: 95 pts * 40% = 38
Assessment of other Environmental Risks: 90 pts * 5% = 4.5
Total = 92.00

As an example as to how this data may be used, suppose it has been determined that the minimum acceptable risk value is 93. The administrator may then recalculate with a replication schema deployed for the most vulnerable data. Suppose this increases the first metric to 95%; if so, the overall score would be increased to 93.25, which exceeds the minimum value. Thus, the administrator would be advised to replicate the most vulnerable data to achieve the desired score.

6. Conclusion

This paper has presented a position that the risk of which server/datacenter to replicate data to can be calculated by using a formula. Additional testing and simulation will allow the authors to continue to develop and fine tune the formula and the weights associated with each risk factor.

7. Bibliography

[1] Marcus, Evan and Hal Stern. *Blueprints for High Availability, Scond Edition*. Indianapolis: Wiley Publishing, Inc, 2003. Book.

[2] Ponemon Institute. "2013 Cost of Data Center Outages." 2013.

[3] Paul, Sujoy. *Pro SQL Server 2008 Replication*. Berkeley: Apress Publishers, 2009.

[4] Rahimi, Saeed K. and Frank S. Haug . *Distributed Database Management Systems: A Practical Approach*. John Wiley & Sons, 2011.

Viability of the Parallel Prefix Algorithm for Sequence Alignment on Massively Parallel GPUs

A. Christian Dicker[1], B. John Sibandze[1], C. Samuel Kelly[1], and D. Jens Mache[1]
[1]Computer Science, Lewis & Clark College, Portland, Oregon, USA

Abstract—*In this paper, we compare two different methods for parallelizing the Needleman–Wunsch dynamic programming algorithm for finding the optimal alignment of two sequences: (1) computing the elements of each diagonal of the table in parallel and (2) computing the elements of each row in parallel using the parallel prefix algorithm. In 2003, the latter algorithm was shown to decrease communication between processors and provide a more uniform work distribution [1]. With the increasing prevalence of general purpose programming on graphics processing units (GPUs), there is a need to reassess the viability of the parallel prefix-based algorithm. We discuss our implementation of both algorithms on a massively parallel GPU and compare the runtimes by thread count as well as by sequence size. We find that the parallel diagonal algorithm runs faster for large sequence lengths.*

Keywords: Bioinformatics, GPU, CUDA

1. Background

1.1 The Needleman–Wunsch Algorithm

The Needleman–Wunsch algorithm uses a dynamic programming table to find an optimal alignment of two sequences (which might represent DNA sequences, English words, etc.), where an alignment is found by inserting any number of gaps into either sequence so that the lengths end up the same. The score of an alignment is found by considering the pair of symbols in each column. If the symbols match (and are not both gaps), that column receives a score of c_1; if they don't match (and neither is a gap), it receives a score of c_2; and if either of the symbols is a gap, the column receives a score of c_3. The score of the alignment is the sum of the scores of all of the columns. The optimal alignment is the one with the highest score [3]. Following related work, we use $c_1 = 1$, $c_2 = 0$, and $c_3 = -1$ in our examples and experiments.

If the sequences are $a_1 a_2 \ldots a_n$ and $b_1 b_2 \ldots b_m$, and our table is T, then $T[i][j]$ (for $i = 0, 1, \ldots, n$ and $j = 0, 1, \ldots, m$) is the score of an optimal alignment of the substrings $a_1 a_2 \ldots a_i$ and $b_1 b_2 \ldots b_j$. (If $i = 0$ or $j = 0$, then the corresponding string is the empty string.) Using this scheme, $T[i][j]$ is the maximum of $T[i-1][j]$, $T[i-1][j-1]$, and $T[i][j-1]$, each plus the cost of moving to the current cell. That is,

$$T[i][j] = \max \begin{cases} T[i-1][j] - 1 \\ T[i-1][j-1] + f(a_i, b_j) \\ T[i][j-1] - 1, \end{cases}$$

where

$$f(a, b) = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{if } a \neq b. \end{cases}$$

1.2 The Parallel Diagonal Algorithm

Notice that each entry in T depends only on entries in the previous two diagonals of the table—and not on any of the entries in the same diagonal. Therefore all of the entries in a diagonal can be computed in parallel.

This observation naturally leads to the parallel-diagonal method of parallelizing the Needleman–Wunsch algorithm, where the diagonals of T are computed in sequence, with each element potentially computed by a different processor [2].

1.3 The Parallel Prefix-Based Algorithm

Given an sequence C of n values and a binary associative operation $\oplus$, the prefixes S of C are given by

$$S[i] = C[1] \oplus C[2] \oplus \cdots \oplus C[i],$$

for $1 \leq i \leq n$. In the case of the Needleman–Wunsch algorithm, we use as our binary operation the maximum function, so that the ith prefix maximum is the maximum of the first i elements of the original sequence.

The sequence of prefixes S can be computed in logarithmic time with multiple processors.

This algorithm was used as the basis of a different method of parallelizing the Needleman–Wunsch algorithm by Aluru *et al.* [1] In this method, the table is computed row-by-row. Since the entries in each row depend on entries in the previous row and the same row, it is done in multiple steps.

First, a partial solution is obtained by assigning to each entry in the row the maximum of the north and northwest entries, each plus the cost of moving to the current cell. Second, the column number is added to each entry to facilitate the computation of the parallel prefix maxima. Finally, the parallel prefix maxima are computed, and the column numbers are subtracted again from each entry, yielding the final values. More detail is given in [1].

The authors showed that this algorithm was time-optimal like the parallel diagonal algorithm while decreasing the amount of communication required between processors.

The parallel prefix algorithm allows p processors to find the prefixes of an array of n numbers in $\mathcal{O}(\frac{n}{p} + \log p)$ time, while also distributing work among the processors uniformly. [1] Therefore the time of filling out the entire table is $\mathcal{O}(\frac{n^2}{p} + n \log p)$.

1.4 Graphics Processing Units

Graphics processing units (GPUs) are accelerators that use data parallel computation to perform hundreds or thousands of operations in parallel. Due to their low power consumption and relatively low cost, they are increasing in prevalence, including being an integral part of many of the fastest supercomputers in the world.

It is for this reason that it is important to re-examine parallel algorithms invented without GPUs in mind within this new paradigm. CUDA (Compute Unified Device Architecture) is a language developed by NVIDIA that allows programmers to use NVIDIA GPUs for general purpose programming. Below we describe our attempt to re-evaluate the benefits of the parallel-prefix algorithm on an NVIDIA GPU.

2. Implementation on GPU

We implemented the two algorithms for an NVIDIA GPU in order to compare their runtimes. We will give pseudocode for each implementation below.

We looked at two implementations of each algorithm: using a single block and multiple blocks. On a single block, we only have access to a single streaming multiprocessor which contains 48 CUDA cores. When we utilize all blocks, we have access to the full 336 CUDA cores on our NVIDIA GTX 460. However, there is some added overhead due to the fact that we have to leave a kernel to synchronize between blocks. Forty-eight CUDA cores already surpasses the maximum number of cores on which Aluru was able to test the parallel prefix algorithm [1].

2.1 Parallel Diagonal

Here we start off with $A = [0]$ and $B = [-1, -1]$, the first two diagonals of T. These are copied to the GPU, along with the two sequences we are comparing (S_1 and S_2), and then the following is executed on the GPU in many different threads, each with a unique thread index (t in the code below).

Note that by using global memory on CUDA, we can execute threads in several blocks, which enables us to use more parallelism.

Algorithm 1 Pseudocode for parallel diagonal GPU kernel

```
Require: $n$, the length of the sequences
Require: $t$, the thread index numbered from 0 to $n$
Require: $diag\#$, the diagonal number from 0 to the length
    of the diagonal
Require: $C$, the diagonal to be computed
Require: $B$, the previous diagonal
Require: $A$, the diagonal prior to $B$ Every diagonal $D \in \{A, B, C\}$ is constructed so that it contains every element
    of the table for which $row + column = diag\#$ and with
    the element in column $column$ accessible at $D[column]$.
  if $column = 0$ or $row = 0$ then
      $C[t] \leftarrow -diag\#$
  else if $i < diaglen$ then
      $C[t] \leftarrow \max \begin{cases} B[t] - 1 \\ B[t-1] - 1 \\ A[t-1] + f(S_1[column-1], S_2[row-1]) \end{cases}$
  end if
  $A \leftarrow B$
  $B \leftarrow C$
```

2.2 Parallel Prefix

We begin with the first row being filled with values 0 to $-n$ where n is the length of the sequence. This row is copied to the GPU, along with the two sequences we are comparing (S_1 and S_2), and then the following is executed on the GPU in many different threads, each with a unique thread index (t in the code below).

3. Experiments

We conducted two main experiments on our NVIDIA GTX 460. The first one compares the diagonal and parallel prefix algorithms on one block with threads equal to 2^i where $0 \leq i \leq 10$ and sequence sizes of 2^j where $8 \leq j \leq 16$. The second experiment uses multiple blocks. For testing the diagonal, we ensure that there is a thread for every element. For testing the parallel prefix, we test a number of blocks equal to 2^k where $0 \leq k \leq 12$, threads per block varying as before, and the sequence size varying as before. The purpose of varying the number of blocks and threads per block is to ensure that we are achieving the optimal configuration so that we may, later, compare the algorithms at their best.

4. Results

We compared the fastest time for each sequence size. With one block, parallel prefix is faster for sequences up to and including 4096 characters. For multiple blocks, the diagonal algorithm is faster for all sequence lengths. Overall, comparing one block to multiple blocks, the single block parallel prefix is fastest for sequences up to and including

4096 characters. From sequence sizes of 8192 and up, the diagonal algorithm on multiple blocks is fastest.

In Figure 2, you can see the speedup of the parallel prefix and parallel diagonal algorithms for one block. You will notice that the speedup does not always increase as the number of threads increases. We tested multiple configurations to find the ideal number of threads. Since the runtime increases with the number of processors, this is not necessarily equal to the largest number of simultaneously running threads on the GPU.

5. Discussion

According to our results, computing elements of a row in parallel using the parallel prefix algorithm can be faster than computing elements of a diagonal in parallel for small enough sequences.

It may seem strange that the parallel prefix algorithm would be faster on one block, when it uses fewer processors, than it is on multiple blocks. This is due to added overhead that occurs when synchronizing threads between multiple blocks, which requires leaving the kernel completely. It makes sense that for larger sequences, this added overhead becomes negligible, and in fact, for sequences of 16384 and larger, the algorithm that uses multiple blocks is faster.

It may also seem strange that the parallel prefix algorithm on a single block is faster than the diagonal algorithm on multiple blocks. This is, most likely, due to some inherent overhead from the diagonal algorithm and once again, becomes negligible at longer sequence lengths. In this case, as mentioned in our results, this sequence length is 8192.

6. Conclusions & Future Work

In this GPU age, we re-examined two different methods for parallelizing the Needleman–Wunsch algorithm for finding the optimal alignment of two sequences: parallel diagonal and parallel-prefix. Our main result is that the parallel prefix-based algorithm on an NVIDIA GTX 460, running on a single block is fastest for short sequences up to and including 4096 characters. Beyond that, the diagonal algorithm on multiple blocks is fastest.

For future work, we would like to rerun our experiments on a better GPU. We have access to an NVIDIA Tesla K20, that we hope to utilize. Also, all of our experiments were done comparing two strings of the same length, which in terms of work distribution, is the worst case for the diagonal algorithm. Therefore, experiments should be done with sequences of very different lengths.

7. Acknowledgments

This research was supported by NSF grant DUE 1044932, by the James F. and Marion L. Miller Foundation and by the John S. Rogers Science Research Program. We would like to thank Kyle Barton, Samuel Dodson, Danielle Fenske, Adam Smith, Ben Whitehead.

References

[1] Srinivas Aluru, Natsuhiko Futamura, and Kishan Mehrotra. Parallel biological sequence comparison using prefix computations. *J. Parallel Distrib. Comput.*, 63(3):264–272, 2003.

[2] David Kirk and Wen mei Hwu. Programming massively parallel processors. Elsevier, 2013.

[3] Saul Needleman and Christian Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mal. Biol.*, 48:443–453, 1970.

Algorithm 2 Pseudocode for parallel prefix GPU kernels

```
Require: $S_1, S_2$, the sequences we are comparing
Require: $n$, the length of the sequences
Require: $p$, the total number of threads (analogous to processors)
Require: $t$, the thread index numbered from 0 to $p-1$
Require: $B$, the row we are computing
Require: $A$, the previous row of the table
Require: $r$, the row number of $B$
Require: $X$, the array of partial solutions for $B$
Require: $Y$, the array that the parallel prefix max is being computed on
Require: $Maxima$,
  $X[\frac{nt}{p}] \leftarrow \frac{nt}{p} + \max \begin{cases} A[\frac{nt}{p}] - 1 \\ A[\frac{nt}{p} - 1] + f(S_1[\frac{nt}{p} - 1], S_2[r-1]) \end{cases}$
  for $i \leftarrow \frac{nt}{p} + 1, \frac{n(t+1)}{p} - 1$ do
      $X[i] \leftarrow i + \max \begin{cases} A[i] - 1 \\ A[i-1] + f(S_1[i-1], S_2[r-1]) \end{cases}$
      $X[i] \leftarrow \max(X[i], X[i-1])$
  end for
  $Y[t] \leftarrow X[\frac{n(t+1)}{p} - 1]$
  $Maxima[t] \leftarrow X[\frac{n(t+1)}{p} - 1]$
  SYNCHRONIZE
  for $i \leftarrow 0, \lceil \log_2 p \rceil - 1$ do
      $partner\_index \leftarrow (t-1)$ XOR $2^i$
      $new\_max[t] \leftarrow \max(Maxima[t], Maxima[partner\_inc$
      if $t > partner\_index$ then
          $Y[t] \leftarrow \max(Y[t], Maxima[partner\_index])$
      end if
      SYNCHRONIZE
      $Maxima[t] \leftarrow new\_max[t]$
      SYNCHRONIZE
  end for
  $tmp \leftarrow Y[t]$
  for $i \leftarrow \frac{nt}{p} + 1, \frac{n(t+1)}{p} - 1$ do
      if $X[i] < tmp$ then
          $X[i] \leftarrow tmp$
      end if
      $B[col] = x[col] - col$
  end for
```

Fig. 1: Runtime for Different Algorithms

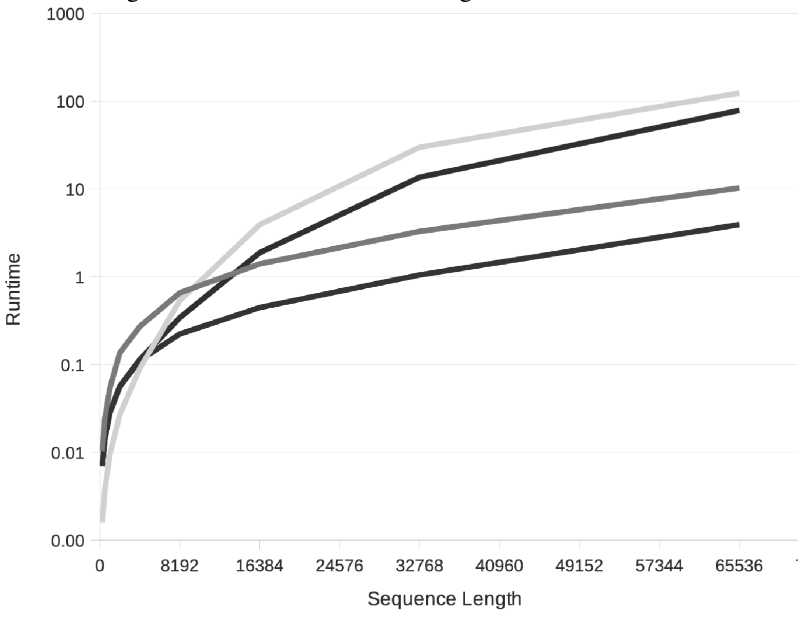
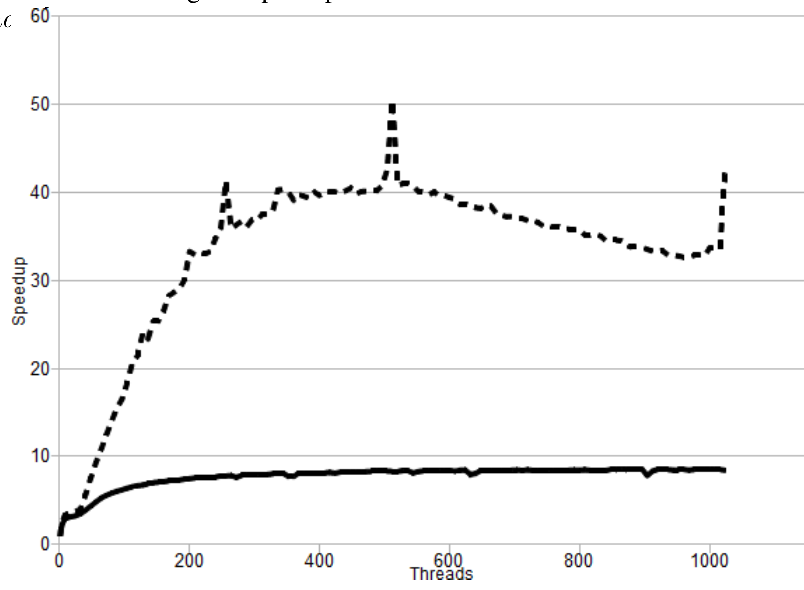


Fig. 2: Speedup for One Block



Resocialization and uses of Social Networks for elderly

A. Cecile Treton[1], **B. Christian Bourret**[2]
[1]Department Culture & society, Paris Est University, Paris, France
[2]Dicen, Paris Ouest University Marne la Vallée, Paris, France

Abstract - *The subject of this publication focuses on the relational commitment problematic. The objective of the research is to identify the relational patterns of the elderly. The research method questions about relationship dynamics which include interactions with for example alliances and desalliances, common points and affinities, meeting places which include the use of social networks on the web. We study the new relational technologies around the notion of situation, describe by Alex Mucchielli. The proposed work will mainly rely on qualitative methods: field observations, field interviews with stakeholders, mainly in situation. For complete organizational and technological approaches, with the biographical interview, we want to know how the person creates its social network.*

Keywords: ICT, social networks, elderly, relational technologies, social ties.

1 Introduction

Like many countries worldwide, the French government is now facing a problem related to the increase in solitudes. A study of the Foundation de France “[1]”, points out a decline in the integrative capacity of family, friends and neighborhood networks. It focuses on increasing situations of loneliness, which affect the entire population, especially the elderly. Between 2010 and 2013, the share of individual isolation increased from 20% to 23%. 5 million of persons, young and older people express a feeling of loneliness.

In France, the term "elderly" concerns a population from 60 years. It corresponds to the reference age at which health problems can appear and justify, according to the French labor law, the payment of an assisted living service.

The situation of loneliness of the elderly is special. Its causes, its consequences and its forms are not the same as for young people.

A reduced ability of mobility and the progressive loss of the emotional environment with the disappearance of its friends, its husband or its wife, not necessarily compensated by the presence of children, explain the loneliness of elderly.

The loneliness for the elderly has consequences on the deterioration of their health status. Seniors who express a distress of the isolation mainly belong to two groups “[2]”. The first is composed of very old people. They feel isolated because of their physical abilities reduced or their disabilities. The second is composed of younger seniors who feel isolated for socio-economic reasons. Often, a residential isolation adds at the emotional and social isolation. Investigations assess loneliness objectively based on the number of daily contacts.

For increase contacts, associations organize visits by volunteers or neighbors and can also integrate technologies that facilitate exchanges at home. These technical devices have the advantage of contributing to the security of the elderly living at home. They detect a lack of movement and can broadcast many messages. These messages generate appropriate behaviors such as taking drugs, the door opening, remembering a visit, etc...

However, it lacks an element in this scheme. The loneliness is considered only by the number of contacts but social actors don’t take into account the subjective nature of loneliness.

The study of the Foundation de France indicates that people surrounded (4.1%) may also feel isolated, even if that this feeling is felt mainly by people objectively alone (11.5%).

The feeling of loneliness does not depend on the number of contacts, but of the value and quality of interpersonal relationships. The sociology of relational dynamics has highlighted the necessary distinction between weak ties and strong ties “[3]”. The feeling of loneliness questions about the relationship between several persons and the mutation of the social network depending life situations.

We do not study the causes and consequences of isolation but the way the person establishes a relational process.

2 Objective

The research objective is to formalize a model of intervention that allows the person to doing progress his network of relationships. The idea is to develop an individualized approach that allows the person a reappropriation of its relation to other.

Currently, external actors (family, neighbors, professionals, medical community, etc.) communicate in the direction of the elderly. We wish to reformulate this model by giving the person the ability to boost the relationship on its own initiative.

In this sense, we consider it necessary to rethink the use of communication technologies. The telecare creates an exchange of standardized way. It can be only a simple reproduction, an artifact of intersubjective communication. It doesn’t take into account the socio-affective dimension of social connections so necessary to the elderly. We think that our research findings can provide useful information to improve communication platforms and their use.

2.1 The old age, a phase of transformation

The situation of the elderly is particularly instructive. The transformations of the social network during its life can be studied. From the observation of the evolution of social ties, we will able to determine the socialization process and the value of links. We are going to identify the strong and friendly ties whose C. Bidart "[4]" highlights the "thickness".
We also believe that aging is accompanied by periods of transition destabilizing. By V. Caradec "[5]", old age brings changes in physical capacities, social status, with many friend losses, difficult transitions, etc… The professionals identify that changes are more intense between 79 and 83 years than over the entire life.

2.2 Socialization and feeling of security

We would like to highlight in particular the identity dimension of the relationship and the sense of security that generates the socialization that reduces the sense of vulnerability.
For C. Deloro "[6]" the other, "Alter Ego", is a "mirror" of myself. The other is seen through personal projections. The other also allows me to know myself better. Merleau Ponty [7] writes : " *The world is not a subject, which I have the law of the constitution, it is the natural place and scope of all my thoughts and all my explicit perceptions (…) the man lives in the world it is in the world he knows himself* ". The other becomes a « link" remarks C. Deloro, and its perception produces an « echo".
C. Audibert "[8]" highlights how emotional and elective relationship with a chosen person allows a "serene solitude" that is to say, freedom to "grow its own solitary garden" Friendship raises the relationship to oneself, through the other, as a participant in self-respect. The self-awareness with that part of intimacy, never revealed, is developed through the other one. It allows us to experience ourselves as existing in our uniqueness.
Socialization allows people to survive through exchanges of gestures, activities and also conversations. S. Tisseron "[9]" in reference to the studies of the anthropologue Robin Dunbar notes that the civilization of the hunter-gatherers spends during its activities, 25% of the day to "chat". According to him, the chatter is not simply a way to transmit information but also a way to develop secure attachments

2.3 Use social networks

We believe that social networks and relational technologies have a role to play in our thinking as they contribute to the highlighting of the report to another, even if they are not currently used by the elderly. B. Stiegler "[10]" with reference to philosopher G. Simondon "[11]" considers that through social networks, each individual part is connected to a "collective individuation," because" The unity of life is the whole group and not the isolated individual." He is agreeing with that Mr. McLuhan wrote "[12]": "Any extension of human faculties is the reaction to irritation caused by the environment and comes in the form of requirements (...) the new medium is a drug for the save in the social balance." J. Rifkin "[13]" writes: "The empathic approach is the existential awareness of the vulnerability we all share."

3 Method and axis of research

The hypothesis of the research is to explore ways to reinstate the individual in the relational process. The research could lead to the formalization of an intervention model. This model could guide the technological choices.
The objective of this research is to analyze the ways of socialization used by the Elderly in terms of relational network. We consider like C. Bidart "[4]" that "relations always have a story." We strive to highlight individual perceptions and the intersubjective nature of the relationship through the social network. This research must do appear expectations, needs and values of the person when it creates a relationship.

3.1 Statistical selection

We selected a population corresponding to three groups. The first group is composed of sick elderly or disabled. They are for those reasons far from social life. The second group consists of people between 60 and 65 isolated for socio-economic reasons. Older people without specific problems are the third group. We could identify 9 persons: 2 couples, 3 single women and 2 single men. They mainly live in Paris. They are between 60 and 95 years old.
We use an investigative work in two parts. Fist, we have collected biographical interviews on the affinities, stories of friendship. Then, we are going to study a communications device with elderly people and neighbors in a French Department. Our interviews last 45mns. Data have been analyzed according to specific criterias.

3.2 Theoric reference

This work is in the field of Science of Information and Communication (SIC), as proposed by Françoise Bernard "[14]" which revolves around questions of meaning, relationships, knowledge and action. We have adopted a constructivist approach that considers social reality as constructed by the actors involved, with questions about social representations and interactions.

3.2.1 Sociology of relational dynamics

We borrowed theories in sociology and education science. The work of collection of interviews and the analysis of materials (life stories) was mainly inspired by the sociology of relational dynamics and the approach of the biographical interviews.
Indeed, these approaches seem most appropriate to account for the complex nature of the relationship to another. The

relationship to the other depends on a variety of cultural and social contexts and the life, emotions and motivations of each individual.
The sociology of relational dynamics of C. Bidart “[4]” studies the nature and inherent relational systems to each individual with his/her environment. The reticular perception can reveal underlying sociological data: community process, identity projection and social recognition, etc.
C. Bidart writes: “What is the relationship to another? (…), this expression means there is a link that goes beyond the simple interaction, which has been registered in time, and has been crystallized beyond occasional exchanges.” The study of networks can highlight individual perceptions, needs and elective choices. It can also highlight changes associated with the stages of life, particularly in the context of advancing age. The level and the process of socialization for each person could be formalised by graphs, with peer relationships, breakups, interconnections, weak or strong links, etc**.**

3.2.2 Method of biographical interviews

To complement this approach, we use the method of biographical interviews defined by C. Delory-Momberger “[15]”. This approach gives the possibility to consider the action as well as the meaning given by the narrator. It reflects his/her interpretation of his/her experiences through social space. The narrative highlights events and breaks in time and space.
C. Delory-Momberger explains: “The intended object of biographical research (…) would be the study of methods of constitution of the individual as a social being singular”.
On the side of listening, we would study by categories such as forms of discourse, the action plan, the recurring patterns or “Topoi”, the biographical management.

3.2.3 Situational semiotic

Finally, in the field of Computer Science and Communication, we rely on Situational semiotic as formulated by A. Mucchielli based on a constructivist approach which study communications in a specific situation.
A. Mucchielli “[16]” raises the question of relation to each other about social identifications, he writes: “Identify the other, it is the judge to define, and this judgment comes from the contexts. Identify the other, is giving meaning to his being and locate him in a set of contexts”. The orientation for the action of an individual is built around different settings and contexts.
The establishment of the reading grid of the interviews owes much to the work of the researcher, which offers a panoramic formalization table for facilitate the analysis of the situation. The table is divided into “frames”. The frames are determined based on the representations of the actor with the identification of its standards, challenges and temporal, spatial, physical and sensory aspects, etc…
The reducing side of the model does not escape at the author. He notes : “The modeling, is not defined here as the operation of model building, but in a constructionist perspective, it is defined as the development of a schematic representation of the operation of the studied phenomena, a representation obtained from a theory and a model”.

3.3 First results and table

Using this method we have established a first approach that has yet to be finalized.
We notice that each person has his specific relational schema.
We have classified interviews into several categories. Here is an extract of the first analysis from three biographical interviews:

	Dorothée	**Audrey**	**Sylviane**
Biographical frame events	Pension Work Marriage Death	Childhood Studies Work Travels Marriage Death	War Marriage Accident (vision loss)
characterization of the relationship with the other	Compensatory mode selective	Adaptive mode	Selective mode
Type of links	Favors direct links	Direct links and interconnection	No interconnection
Methods activation	Favors the common activity	Friendly and ritualized	Search help, support, taking account of disability

Some elements are convergent, others are specific.
We see as similarities, weak interconnections between members, transformation of the relational network at every stage of rupture (events, changes …), consolidation of the relationship through mutual sharing of a situation, search moments of shared pleasures.
The specific elements are level of involvement, exchange modes, regularity of meetings, expectations and needs satisfaction.
We confirm that age changes the relational schema of the person. A reorganization of the network is necessary with loss of familiar. Changes concern the link density, the strengthening the local network, an increasing of distance relationship management (letters, telephones, etc.), relationships having less impact on the intimate sphere.

3.4 The use of ICT to mitigate loneliness

The use of social digital networks allowed emergence of many research about social relations by sociologists and experts of information systems management. Studies show that users of social digital networks, especially young people and

older people, referred to as “silver surfers" are at first interested by exchanges with their friends or members of their family. The daily conversations contribute at the socialization.
Serge Proulx “[17]” emphasizes specific aspects of uses of social and digital networks like: Complete its profile and increasing its visibility, use specific modalities of exchanges, alternate the private and public communications, participate to collective contributions, expand weak and strong ties. But, the biographical interviews collected from elderly people show that these modes of communication existed before the digital relations like the self control described by Norbert Elias “[18]” or the reference to a public event like pretext to conversation.
Nevertheless, some needs are specific to the elderly and isolated people. They need to understand the structuration of their emotional relationships and we think the digital network can be use in this case. Also, the design of the social networks interfaces seems far of habits of elderly people. The presentation of information cannot be treated by some users such as the elderly. They have difficulties to integrate certain technical functions like: the mailing lists, communities of contacts or the scrolling of messages.

4 Conclusion

In a context marked by a technological and sanitary dominant approach and "top down” exchange, it is necessary to consider the formulated needs of users and to bet on the human in its ability to give meaning to its action.
The initial findings show the desire and ability of elderly people to create a network of relationships, based on a singular relational process and redundant. Age rather than the socio-professional category reveals common specificities in the way to live relations (density, modalities, temporal and spatial forms, etc.).
The analyze of social digital networks or traditional relationships help to understand impact of daily exchanges in the process of socialization. To allow people to build their relational network it seems necessary to use an accompaniment model which promotes creation of situations of communications. These situations could integrate ICT and their value of “conversational media”. Serge Proulx “[17]” writes: “the interpretation of activities on-line and off-line is a subject of interrogations for researchers, a challenge, and requires news methods for reconciling the technical and social universe”.

5 References

[1] Fondation de France. “Loneliness in France”, Study, Paris, 2012.
[2] Françoise Souêtre-Rollin . “Rapport Isolement et vie relationnelle”, Collectif Combattre la solitude, Paris, 2006.
[3] Marc Granovetter. “Le marché autrement, le marché de l’économie”, Desclée de Brouwer, 2000.
[4] Claire Bidart, Alain Degenne, M. Grosseti. “La vie en réseau dynamique des relations sociales”, Paris, PUF, 2011.
[5] Vincent Caradec. “Sociologie de la vieillesse et du vieillissement”, Paris, collection Armand Colin, 2012.
[6] Cyrille Deloro. “L’Autre, Traité de narcissisme intelligent”, Paris, Edition Larousse, 2009.
[7] M. Merleau Ponty. “Phenomenologie de la perception”, Ed. Gallimard, Paris, 1945.
[8] Catherine Audibert., “The inability to be alone, essay on love, loneliness and addiction”, Ed. Payot, Paris, 2008.
[9] Serge Tisseron. “Empathy in the heart of the social game”, Ed. Albin Michel, Paris, 2010.
[10] Bernard Stiegler. “The most valuable at the time of sociotechnologies”, Social networks, Ed. FYP, 2012.
[11] George Simondon. “The individual and his hysico-biological genesis”, Ed. Jérôme Milliam, Paris, 1997.
[12] Marshal Mac Luhan. “From eye to ear, the New galaxy” Ed. Hartubise, Montréal, 1977.
[13] Jéremy Rifkin. “A new conscience for a world in crisis, to a civilization of empathy”, Ed. Actes Sud, 2011.
[14] Françoise Bernard. “ICT, discipline openness and decompartmentalization”, in A. Bouzon: “Organizational communication debate. Fields, concepts, perspectives”, Paris, L’Harmattan, 2006.
[15] Christine Delory-Momberger. “Approche méthodologique et recherche biographique”, EXPERICE, Université Paris 13, mars 2014.
[16] Alain Muchielli. “Situation et Communication”, Ed. Ovadia, Nice, 2010.
[17] Serge Proulx. “Media sociaux, enjeux pour la commincation”, Ed. Presse Universitaire du Québec, 2012.
[18] Norbert Elias. “La Société des individus”, Pocket, 2004.