A CPU and GPU Heterogeneous Processing of Multimedia Data by using OpenCL[1]

Heegon Kim, Sungju Lee, Yongwha Chung, Daihee Park
Dept. of Computer and Information Science, Korea University, Sejong City, Republic of Korea

Abstract - *In recent times, it has become possible to parallelize many multimedia applications using multicore platforms such as CPUs and GPUs. In this paper, we propose a parallel processing approach for a multimedia application by using both the CPU and GPU. Instead of distributing the parallelizable workload to either the CPU or GPU, we distribute the workload simultaneously to both by using OpenCL. Based on our experimental results, using both the CPU and GPU, we confirm that the proposed parallel processing approach provides better performance than typical parallel processing approaches on account of maximal utilization of the given resources.*

Keywords: CPU, GPU, Heterogeneous Computing, OpenCL

1 Introduction

As multicore processors are now widely available, parallel processing approaches have been developed for many applications. In this paper, we focus on parallelizing multimedia applications by using both the CPU and GPU. Especially, OpenCL [1] has been defined as a standard for heterogeneous parallel computing. It provides a cross-platform framework for writing software able to run on different kinds of devices, from multicore CPUs to GPUs. That is, a parallel program written with OpenCL can be executed on either a CPU or GPU [2]. Generally, it is true that a GPU can provide better performance than a CPU for multimedia applications. However, current multicore CPUs are also powerful processors, and thus, when used together with a GPU, the total execution time can be reduced.

We propose a load balancing approach that can overcome the performance limits of either CPU-only or GPU-only execution. We first attempt to achieve parallelism in a typical multimedia application (i.e., photomosaic application [3]) using OpenCL, and measure its execution time on the CPU and GPU, respectively. Then, we partition the parallelized workload into two parts, based on the relative performance of the GPU over the CPU and some parallel overhead. Finally, we assign the GPU-portion of the workload to the GPU by using a non-blocking command, and then assign the remaining parallel portion to the CPU without waiting for a result from the GPU. By reducing the idle time on either the CPU or GPU, we maximally overlap the GPU execution with the CPU execution.

2 Heterogeneous processing of Photomosaic

A GPU has many cores with low clock frequency, whereas a CPU has few cores with high clock frequency. Because a GPU is based on a Single Instruction, Multiple Data (SIMD) architecture with many cores, it can efficiently compute the same operations over large images with no data dependency, as required by the photomosaic application. That is, the photomosaic application can perform better using a GPU than a CPU. A large number of studies of GPU-equipped environments using only GPU-based parallel processing have been published [4-5]. However, a current multicore CPU is also a powerful processor, and thus, when used together with a GPU, the total execution time can be reduced.

To generate the photomosaic based on a CPU-GPU heterogeneous computing environment, we employ OpenCL [1]. Figure 1 shows the OpenCL code for generating the photomosaic using both the CPU and GPU. After initializing the CPU and GPU, they are each assigned a workload (i.e., some row pixels of the image), respectively. Because of the speed discrepancy between the CPU and GPU, we assign the workload carefully such that the possible idle time on either the CPU or GPU should be minimized. Then, the OpenCL *clEnqueueReadBuffer* function with non-blocking mode is used for execution of the code on the GPU. Finally, the OpenCL *clFinish* function is used to synchronize between the CPU and GPU.

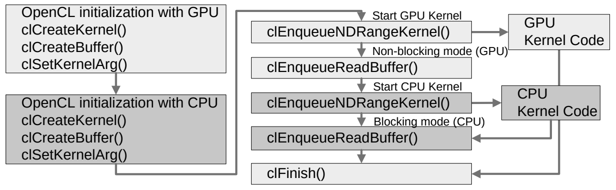

Figure 1. OpenCL code for the photomosaic

The amount of workload assigned to each processor is determined by the *clCreateBuffer* function and the *clSetKernelArg* function. In this paper, we propose a simple, but effective load balancing approach in order to reduce the possible idle time on either the CPU or GPU. We first execute the OpenCL code for CPU-only (i.e., assign 100% of the workload to CPU) and GPU-only (i.e., assign 100% of the workload to GPU) cases. Using the speed discrepancy between the CPU and GPU, we set the initial workload to each processor. Note that the non-blocking mode of the execution also incurs some overhead on the host (i.e., CPU).

[1] "This paper is being submitted as a poster".

Thus, we start searching for the optimal load distribution by comparing the execution time of the initial load distribution (i.e., determined by the speed discrepancy between the CPU and GPU) with that of the "CPU-less" load distribution (i.e., some of the initial CPU load is assigned to the GPU). If the CPU-less case provides faster execution time, this comparison is repeated until no more improvement can be found. As we will explain in the next Section, we can determine the optimal load distribution with very few comparisons.

Note that, determining the actual execution time on either the CPU or GPU analytically is a very difficult problem. Because the OpenCL code can be run on both the CPU and GPU, we can easily measure, with very few comparisons and fine tune the workload distribution, in addition to handling the speed discrepancy between the CPU and GPU (i.e., CPU-only and GPU-only cases).

3 Experimental Result

For evaluating the proposed approach, we used two platforms. *Platform 1* was composed of an AMD Phenom II X4 955 CPU (4 cores) and a GeForce GTX 285 GPU (336 cores). *Platform 2* was composed of an Intel Core i5-2500 CPU (4 cores) and a GeForce GTX 560 GPU (336 cores). The target image size was 3072×2048, and the image was partitioned into 64×64 blocks of data for the photomosaic. That is, we need to assign each of the 32 (= 2048/64) row blocks to each processor. In order to obtain the speed discrepancy between the CPU and GPU, we first executed the photomosaic for CPU-only and GPU-only cases. For platform 1, the GPU performs 2.3 times better than the CPU, whereas on platform 2, CPU performs 1.7 times better than GPU. Therefore, from the total of 32 row blocks, we set the initial load distribution CPU:GPU = 10:22 on platform 1 and CPU:GPU = 20:12 on platform 2.

Then, we compared the execution time of this initial distribution with that of CPU-less distribution. In our experiment for the minor tuning of the load distribution, we decreased the CPU workload in each comparison (i.e., the first iteration compared the workload ratio 10:22 with 9:23 for platform 1, and compared the workload ratio 20:12 with 19:13 for platform 2). Because the workload ratio 9:23 was faster than 10:22 on platform 1, we compared the workload ratio 9:23 with 8:24 in the next iteration. Similarly, since the workload ratio 19:13 was faster than 20:12 on platform 2, we compared the workload ratio 19:13 with 18:14 in the next iteration. The comparison was terminated when improvements cannot be obtained. We derived the optimal load distribution for this example (i.e., workload ratio of CPU:GPU = 7:25 for platform 1 and 18:14 for platform 2).

Note again that our approach does not need to measure all the cases. In addition to CPU-only and GPU-only (for the initial load distribution), only the cases of workload ratio from 10:22 to 6:26 (i.e., five workload ratio cases) were measured on platform 1, and only the cases of workload ratio from 20:12 to 17:15 (i.e., four workload ratio cases) were measured on platform 2, in order to derive the optimal load distribution. Finally, Table 1 compares the speedup of CPU-only, GPU-only, and the proposed approach.

Table 1. Speedup comparison

	Speedup	
	Platform 1	Platform 2
CPU-only (CPU:GPU = 32:0)	16.8	19.5
GPU-only (CPU:GPU = 0:32)	38.5	11.7
Proposed approach	45.0 (CPU:GPU = 7:25)	26.0 (CPU:GPU = 18:14)

4 Conclusions

We have proposed an efficient heterogeneous parallel processing approach to reduce the CPU idle time in a multimedia application. The approach using OpenCL, involves using both the CPU and GPU, and decreases total execution time resulting in better performance.

Experiments with the use of both the CPU and GPU for parallel processing have demonstrated that our parallel processing approach can provide a speedup of 45 (17% better performance than the typical parallel processing approach using GPU-only) on platform 1 and a speedup of 26 (222% better performance than using GPU-only) on platform 2. Our approach exploits the main advantage of OpenCL (i.e., portability across platforms) and derives the optimal load distribution without using complicated scheduling techniques.

5 Acknowledgement

This research was supported by Basic Science Research Program(through the NRF funded by the Ministry of Education, Science and Technology, No. 2012R1A1A2043679).

6 References

[1] J. Stone, D. Gohara, and G. Shi, "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems," *Computing in Science and Engineering*, Vol. 12, No. 3, pp. 66-73, 2010.

[2] R. Gaetano and B. Pesquet-Popescu, "OpenCL Implementation of Motion Estimation for Cloud Video Processing," in *Proc. of International Symposium on Multimedia Signal Processing*, pp. 1-6, 2011.

[3] R. Silvers and M. Hawley, *Photomosaics*, Henry Holt, New York, 1997.

[4] J. Cao, X. Xie, J. Liang, and D. Li, "GPU Accelerated Target Tracking Method," *Advances in Intelligent and Soft Computing*, vol. 128, pp. 251-257, 2012.

[5] D. Davendra and I. Zelinka, "GPU Based Enhanced Differential Evolution Algorithm: A Comparison between CUDA and OpenCL," *Intelligent Systems Reference Library*, vol. 38, pp. 845-867, 2013.