Acceleration of Computational Fluid Dynamics Analysis by using Multiple GPUs

Hyungdo Lee[1], Bongjae Kim[2], Kyounghak Lee[3], Hyedong Jung[1]

[1]Embedded and Software Research Center, Korea Electronics Technology Institute, Korea
[2]Department of Computer Science and Engineering, Sun Moon University, Korea
[3]IACF Namseoul University, Korea
joytop88@keti.re.kr[1], bjkim@sunmoon.ac.kr[2], khlee@nsu.ac.kr[3], hudson@keti.re.kr[1]

Abstract - *GPU-based computing is widely used in various computing fields. In case of Computational Fluid Dynamics (CFD), there are computation intensive iterative solvers. Iterative solvers are bottlenecks of CFD. Recently, CFDs require high-accuracy and high-resolution than before. By above reason, the problem size of CFDs continues to grow and the performance of CFDs is also falling in terms of execution time. One of the solutions is to use GPU which support many cores than typical CPU. GPU can be used to accelerate the computation of CFDs like matrix multiplication. The improvement of GPU depends on how to use GPU due to the complexity of its architecture. In this paper, we propose a scheme to improve the performance of CFD applications based on multi-GPUs. In our approaches, we adjust GPU-based SpMV (Sparse Matrix Vector multiplication) and use multi-GPUs by considering characteristics of input matrix. We have changed the matrix multiplication method from scalar-based scheme to enhanced vector-based scheme. In addition, we used direct memory access (DMA) scheme among multi-GPUs to reduce the latency. Based on the performance evaluation result, the overall performance was improved 4.6 times when compare to previous CPU-based scheme.*

Keywords: Computational Fluid Dynamics, Multi-GPUs, CUDA

1 Introduction

Computational Fluid Dynamics (CFD) is a computer-based numerical analysis or simulation such as fluid flow and heat transfer [1]. Recently, CFDs requires high-accuracy and high-resolution. By above requirements, the size of problems is also increased continuously. For example, mesh structures for CFD analysis or simulation are getting fine-grained to obtain a more accurate result.

HPC (High Performance Computing) systems are essential and a good choice to deal with this problem. Because, HPC system supports massive computing power. In addition, GPUs can be applied to HPC systems to accelerate the computation. Typically, a GPU support more cores than typical CPU. For example, there are 2496 cores on each Tesla K20M GPU card. For these reasons, many studies have been performed in an effort to realize a high-performance computing environment based on GPU. Molecular dynamics [2][3], quantum chemistry [4], financial engineering [5][6], data mining [7] are some representative fields which use GPU-based HPC system.

In this paper, we focus on CFD application. Typically, there are many iterative solvers in CFD applications. iterative solvers is a dominant part of CFD application in terms of execution time. Iterative solvers are bottleneck of CFD simulation. Therefore, iterative solver is key point to increase the performance of CFDs. An iterative solver is mainly consisted of SpMV (Sparse Matrix Vector multiplication). GPU is one of the best solutions to accelerate SpMV computation. However, GPU architecture is very complex and it is hard to obtain relatively good performance based on GPU. In this paper, we propose some scheme and approaches to increase the performance in terms of execution time and latency. To increase the performance of CFDs, we adjust GPU-based SpMV method and use multi-GPUs by considering characteristics of input matrix. In our SpMV scheme, a warp (32 GPU threads) are assigned to multiple rows of a sparse matrix to calculate matrix vector multiplication. Because, a warp is a scheduling unit of GPU. By using this manner, we can minimize the memory access of GPUs when doing SpMV. In addition, we use direct memory access scheme to reduce the data transfer latency among multi-GPUs. Our enhanced SpMV scheme is applied to BiCGStab and CG solver. BiCGStab and CG solver are two representative iterative solver algorithms which are widely used in CFDs. Based on the performance evaluation result, the overall performance was improved 4.6 times when compare to previous CPU-based approach.

The rest of this paper as follows. In section 2, some related works are discussed. In section 3, we will explain our approach to improve the performance based on the computing environment with multi-GPUs. Performance evaluation. Finally, we conclude this paper with future works in section 5.

2 Related Works

2.1 3D Coronary Artery Blood Flow Dynamics

CFD is commonly used in scientific and engineering fields to investigate fluid flow and its interactions in a particular domain. In order to applying our multi-GPU accelerating scheme, we use 3D coronary blood fluid-dynamics simulation application which investigates flow of blood and its interactions to diagnose cardiovascular disease. This application is constructed using 3D unsteady Navier-Stokes equations which describe how the velocity, pressure of a moving fluid is related. And our 3D unsteady Navier-Stokes equations use Finite Element Method (FEM). FEM subdivides a large problem into smaller, simpler, parts, called finite element. So it can make discretized domain from physical space. Many studies have shown that finite element methods can be successfully applied to the analysis of the unsteady Navier-Stokes equations [8]. and this application use Uzawa iteration to solve the Navier-Stokes equation by using FEM. Uzawa Iteration consist of outer iteration to update the pressure and an elliptic inner iteration for velocity. In computing 3D unsteady Navier-Stokes equations, we should consider time as a fourth coordinate direction. As space coordinates are discretized, time must be discretized. And explicit method is used for time integration. It is computed before executing Uzawa iteration [9]. In this way, the 3D coronary blood fluid-dynamics is analyzed. Uzawa iteration and explicit method has linear system problems. So Conjugate Gradient (CG) and Bi-Conjugate Gradient Stabilized (BiCGStab) solvers can solve the problems.

2.2 Preconditioned Iterative Solver

The size of modeling of engineering, physics and economics is increasing. So solving the large-scale linear systems is essential. For solving the large-scale linear systems, direct method is no effective. Because it has a heavy memory load and the computing overhead. From the past, iterative solver like CG, GMERS, BiCGStab were designed to overcome direct method problems in terms of performance. And preconditioning techniques also were designed to improve accuracy and performance of iterative solver [10]. As other studies to improve the iterative solver in progress, many solvers were proposed like Bi-Conjugates Gradient (BiCG), Conjugate Gradients-Squared (CG-S), Bi-Conjugate Gradients Stabilized (BiCGStab) solver [11]. CG-S is a variant of the BiCG solver. But, it has been observed that CG-S may lead to a rather irregular convergence behaviour, so that in some cases rounding errors can even result in severe cancellation effects in the solution. So, another variant of BiCG or BiCGStab which is more stabilized and efficient. In this paper, explicit method uses BiCGStab solver and Uzawa iteration use CG solver to solve linear system at unsteady Navier-Stokes equation.

When solving a linear equation of the form $A \times x = b$ for x, where matrix A is large and b is a vector, time of obtaining x should be a long time. Because computational complexity of matrix inversion. In this case, the iterative solver is used. Furthermore, after substitute the form $A \times x = b$ to $r_i = b - A \times x$, the iterative solver repeat until r_i becomes sufficiently small. This allows to obtain the similar approximations as x. and preconditioner make it fast and accurate.

In recent years, the iterative solver has been applied to HPC with GPU [12]. also the preconditioner has been applied [13]. Furthermore, study of improving CG solver using texture memory and shader function of GPU are in progress [14].

2.3 Compute Unified Device Architecture

Single core of CPU has some limitation in terms of it performance like computing power. So multi-core architecture like dual or octa cores has been applied to CPU architecture to improve the performance. However, there is also a limit that increasing the number of CPU cores in a single chip. In order to overcome this problem and achieve HPC, many-core architecture with GPU can used to general purposes. The reason why GPU has many cores is that common graphics applications handle large 3D rendering and multi-textures and these require huge computing power. Formerly, general purpose programming in the GPU-based computing environment was not possible. However, the needs of HPC equipped with a lot of GPU devices has been increasing continuously. So GPU vendors are developing GPU platform including GPU programming language and programming tools. The latest released GPU's performance has the minimum 1 TFLOPS/s of computation performance and 160GB/s of off-chip memory bandwidth. By above reasons, many GPU-based HPC system are widely used for various computing fields. There are two major GPU platforms. First one is OpenCL(Open Computing Language) which is used universally today, Second one is CUDA (Compute Unified Device Architecture) which can be used over only NVIDA GPUs or architecture. In this paper, we use NVIDIA GPUs and CUDA programming model to improve the performance of CFD applications.

3 CFD Acceleration by using Multi-GPUs

3.1 Basic Iterative Solver based on CUDA Programming Model

First of all, in this sub-section, we explain an approach that accelerates BiCGStab and CG solver by using single GPU. GPU-based computing acceleration is a kind of data parallelization. Therefore, if there are data dependency in algorithms, it is hard to parallelize the algorithm with GPU. In case of BiCGStab and CG solver are mainly consisted of SpMV operation, addition and subtraction between vectors,

and inner product operation between vectors. These operations are very suitable to apply GPU-based computing acceleration because these operations do not have any data dependency. In addition, some preconditioner can be used to converge rapidly for BiCGStab and CG solver. There are many preconditioners such as diagonal, incomplete factorization, approximate inverse preconditioner. In our scheme, we choose the Jacobi preconditioner because it is very suitable to parallelize its operation. We use CUDA programming model like Figure 1 to parallelize those operations and those operations can be executed on the GPU in parallel.

$$\text{for } i = 1, 2, 3, \ldots, n \quad v_{(i)} = a_{(i)} + b_{(i)}$$

$$gId = blockIdx * blockDim + threadIdx$$
$$v_{gId} = a_{(gId)} + b_{(gid)}$$

Figure 1. An Example of Vector Addition Code (Left Side: Sequential(CPU), Right Size: Parallel(GPU))

Figure 1 shows an example of vector addition code. Left side code is a sequential version for CPU. Right side code is a parallel version which is followed CUDA programming model for GPU. In case of the sequential version, as shown in Figure 1, the addition process sequentially proceeds *for* statement from first to last elements of the vector to calculate vector *v* [15]. In case for GPU, on the other hand, the addition process creates many threads as much as the size of vector, *n* by using CUDA C like *blockIdx*, *blockDim* and *threadIdx*. And the thread of the size of the vector is assigned to concurrently *gId*. A thread to process the addition take each element from two vectors. After proceeding the addition, a thread stores the added value in the vector *v*. This progress can be executed concurrently at the GPU.

```
1.  gId = blockIdx * blockDim + threadIdx
2.  for (k = rowPtr[gId]; k < rowPrt[gId+1]; k++)
3.      r[gId] += val[k] * v[culm[k]]
```

Figure 2. GPU-based SpMV Code Using CSR Format

Figure 2 is an accelerated SpMV code using CUDA programming model. Similar to Figure 1, SpMV operation can be accelerated with GPU and CUDA programming model. A matrix used in the our SpMV is a form of sparse matrix. It needs to be compressed to use memory efficiently and compute its related operation fast. There are many methods to compress a sparse matrix to a compressed form. For examples, there are Coordinate (COO), ELLPACK (ELL), Hybrid (HYD) and CSR (Compressed Storage Row) format. Those format is classified according to the nature of the sparse matrix. In short, the proper compress algorithm is different depending on the characteristic of a sparse matrix. DIA, ELL, CSR, HYB, COO format is sorted by usage of the nature of the matrix. The order is from structured matrix to unstructured matrix. For example, DIA format is very suitable for structure matrix form. COO format is very suitable for unstructured matrix form. The nature of sparse matrix of our blood fluid dynamics is middle of structured and unstructured matrix. Therefore, we use CSR matrix format when compress an original sparse matrix.

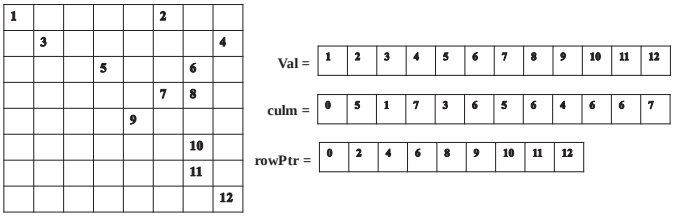


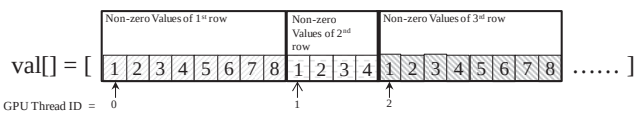
Figure 3. An Example of Sparse Matrix and its CSR Format

Figure 3 is an example of sparse matrix and its CSR format. As shown in Figure 3, the size of sparse matrix is 10 × 10. First, non-zero values of sparse matrix sequentially store in the *val*. Second, *val*'s column position at the sparse matrix is stored in the *culm* per each non-zero value. At last, *rowPtr* stores the starting index of each row *in Val*.

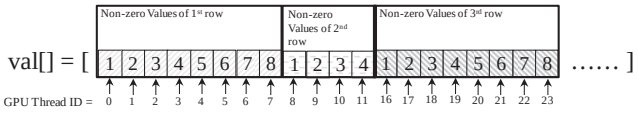
3.2 Advanced Iterative Solver based on Memory Coalescing

In our basic iterative solver, we just use a scalar-based SpMV. One GPU thread is assigned per row of sparse matrix in case of the scalar-based SpMV. Typically, a warp is a scheduling unit of GPU. A warp is consisted of 32 GPU threads. In basic iterative solver, a warp cannot access to contiguous memory space in which non-zero elements are stored. By above reason, scalar-based SpMV could not utilize memory coalescing when access to memory. If we use memory coalescing, we can reduce the number of memory access operation for a warp to only one memory access operation.

In the advanced iterative solver, we used enhanced SpMV method to improve the performance more. In the advance iterative solver, a warp (32 GPU threads) are assigned to multiple rows of a sparse matrix to calculate matrix vector related operations. A warp can access to contiguous memory space by this manner. Therefore, multiple memory access operations can be minimized. Figure 4 shows an example of GPU thread assignment comparison between scalar-based SpMV and enhaced SpMV.



(a) GPU Thread Assignment of Scalar-based SpMV (Basic Iterative Solver)



(b) GPU Thread Assignment of Enhanced SpMV (Advance Iterative Solver)

Figure 4. An Example of GPU Thread Assignment Comparison Between Scalar-based SpMV(Basic Iterative Mode) and Enhanced SpMV (Advanced Iterative Solver)

3.3 Using Multi-GPUs with Domain Decomposition and MPI Programming Model

If we use domain decomposition scheme, we can obtain further acceleration with multi-GPUs. In our scheme, we device main domain into multiple sub-domains. Each sub-domain for CFDs is assigned to one GPU to simulate or analyze the result [16]. Figure 5 show the concept of using multi-GPUs with domain decomposition.

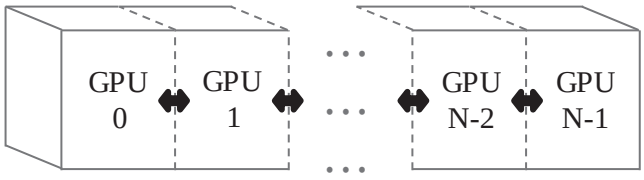


Figure 5. A Concept of Using Multi-GPUs with Domain Decomposition

If we use domain decomposition scheme, data exchanged between two adjacent domains or GPUs because intermediate result affect each adjacent domain. In our approaches, we use MPI (Message Passing Interface) programming model to communicate and exchange the intermediate result between two adjacent domains or GPUs.

In this situation, there are 2 step data copy operations. For example, first step is one GPU memory to main memory, and second step copy is required from main memory to another GPU memory. Data communication overhead is considerable. This communication is also bottleneck of GPU-based computation. To reduce the data communication overhead, we use DMA (Direct Memory Access) scheme between two GPU memory. By using DMA, we can communicate directly among multi-GPUs. In short, we can improve the performance more by using DMA.

4 Performance Evaluation

4.1 Evaluation Environment

Table 1 shows the computing environment used in the performance evaluation. GPU and CUDA libraries are necessary must be installed in the system to apply CUDA-based programming model onto preconditioned iterative solver for CFD simulation or analysis. As shown in Table 1, CUDA 7.5 was installed in the computing system. GeForce Titan Black and Tesla K20 are equipped. In case of MPI library, MVAPICH2 2.2b was used to evaluate the performance.

Table 1. Computing Environment

Features	Descriptions
CPU	2 × Intel Xeon CPU E5-2650 2.6 GHz v2
Memory	64 GB
OS	CentOS 7.2
Kernel Version	3.10.0
CUDA Version	CUDA 7.5
OFED Version	MLNX_OFED_LINUX 3.2
MVAPICH2 Version	MVAPICH2-2.2b
GPU	GeForce Titan Black Tesla K20
InfiniBand Host Channel Adapter	Mellanox ConnectX-3 VPI Adapter Dual-Port QSFP, FDR IB(56 Gb/s)

4.2 Evaluation Results

4.2.1 Acceleration of Basic Iterative Solver based on CUDA Programming Model

Table 2 shows the result of acceleration of basic iterative solver per one iteration. The row size of sparse matrix for CG is 219,725 and non-zero elements are 2,710,418. In case of the BiCGStab, the row size is 219.725 and its non-zero elements are 5,201,235. As shown in Table 2, the maximum speed up is 7.349 by GPU-based acceleration. Overall, GPU-based acceleration is better than CPU-based solver. Titan Black is better than Tesla K20 because the base clock speeds of Titan Black and Tesla K20 are 889 MHz and 706 MHz, respectively.

Table 2. Basic Iterative Solver Results

Features		CPU (s)	GPU (s)	Speed Up
CG	Tesla K20	0.716	0.195	3.672
	Titan		0.115	6.226

	Black			
BiCGStab	Tesla K20	0.801	0.167	4.811
	Titan Black		0.109	7.349

4.2.2 Acceleration of Advanced Iterative Solver based on CUDA Programming Model

Table 3 shows the result of acceleration of advanced iterative solver per one iteration. As shown in Table 3, the maximum speed up is 18.372 seconds. Maximum speed up is increased from 7.347 seconds to 18.372 seconds due to memory coalescing. Similar to Table 2, Titan Black is better than Tesla K20 in terms of the execution time. Based on the Table 2 and Table 3, we can improve the performance of CG and BiCGStab solver. As the result, we can reduce the overall execution time of CFD simulation or analysis application.

Table 3. Advanced Iterative Solver Results

Features		CPU (s)	GPU (s)	Speed Up
CG	Tesla K20	0.716	0.107	6.692
	Titan Black		0.073	9.808
BiCGStab	Tesla K20	0.801	0.062	12.919
	Titan Black		0.044	18.372

4.2.3 Intra-node Data Transfer Latency between GPUs

Table 4 shows the result of intra-node data transfer latency between two GPUs. In the performance evaluation, we used Mellanox OFED 3.2, MVAPICH2 2.2b, and NVIDIA GeForce Titan Black. As shows in Table 4, In case of 32 KB data transfer, DMA is about 1.2 times better than No DMA. Similarly, in case of 64 KB data transfer, DMA is about 5.9 times faster than No DMA. As we explained in Section 3, The overhead of data exchange or transfer is considerable between two intra GPUs. Therefore, if we use DMA, we can reduce the data transfer delay effectively.

Table 4. Intra-node Data Transfer Latency Between GPUs

Size (Byte)	No DMA (us)	DMA (us)
32 K	75.8	59.0
64 K	124.9	59.4
128 K	197.9	59.4
256 K	366.9	61.5

Acceleration of Blood Fluid Dynamics by using Multi-GPUs and Advanced Iterative Solver

To evaluate the performance of our scheme, we used blood fluid simulation application. Advance iterative solver and DMA scheme were applied to our blood fluid simulation application. BiCGStab and CG solver were used.

Figure 6 shows the result of total execution time of blood fluid dynamics. In the performance evaluation, we used GeForce Titan Black GPU because Titan Black showed better performance than K20 GPU. Simulation areas are divided into 4 areas. Each area is assign to one CPU core or one GPU device. As shown in Figure 6. We can reduce the total execution time of blood fluid dynamics by using multi-GPUs. The total execution time was reduced from 7113 seconds to 1527 seconds. In short, the performance was improved about 4.6 times in terms of speed up.

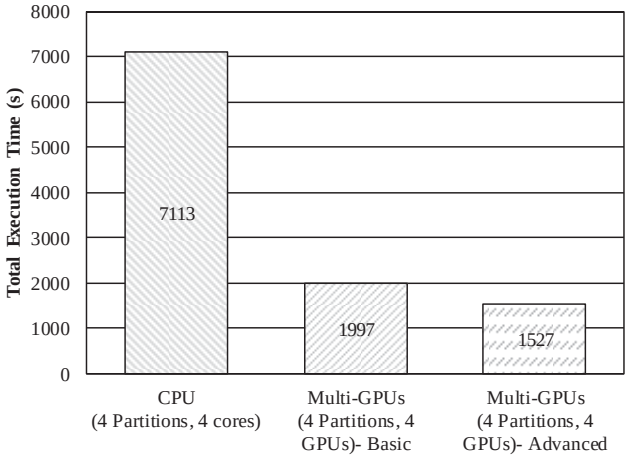


Figure 6. Total Execution Time of Blood Fluid Dynamics

5 Conclusions and Future Works

In this paper, we proposed a scheme to improve the performance of CFD application based on multi-GPUs. Our scheme used enhanced vector-based SpMV method with domain decomposition in order to reduce the SpMV time. SpMV is a dominant part of CFD applications in terms of execution time. We reduced the SpMV time by using multi-GPUs. In addition, DMA scheme are also used to reduce the latency among multi-GPUs. As the result, overall execution time decreased efficiently. Based on the performance evaluation results, the performance was improved 4.6 times when compared to the previous scheme which uses 4 CPU cores. In the future works, we will apply GPU-based computing to other computation-sensitive fields.

6 Acknowledgement

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No.R0101-15-0171, Development of Multi-modality Imaging and 3D Simulation-

Based Integrative Diagnosis-Treatment Support Software System for Cardiovascular Diseases)

7 References

[1] Ferziger, Joel H., and Milovan Peric. Computational methods for fluid dynamics. Springer Science & Business Media, 2012.

[2] Anderson, Joshua A., Chris D. Lorenz, and Alex Travesset. "General purpose molecular dynamics simulations fully implemented on graphics processing units." Journal of Computational Physics 227.10 (2008): 5342-5359.

[3] Anderson, Joshua A., Chris D. Lorenz, and Alex Travesset. "General purpose molecular dynamics simulations fully implemented on graphics processing units." Journal of Computational Physics 227.10 (2008): 5342-5359.

[4] Olivares-Amaya, Roberto, et al. "Accelerating correlated quantum chemistry calculations using graphical processing units and a mixed precision matrix multiplication library." Journal of chemical theory and computation 6.1 (2009): 135-144.

[5] Fatone, Lorella, et al. "Parallel option pricing on GPU: barrier options and realized variance options." The Journal of Supercomputing 62.3 (2012): 1480-1501.

[6] Surkov, Vladimir. "Parallel option pricing with Fourier space time-stepping method on graphics processing units." Parallel Computing 36.7 (2010): 372-380.

[7] Jian, Liheng, et al. "Parallel data mining techniques on graphics processing unit with compute unified device architecture (CUDA)." The Journal of Supercomputing 64.3 (2013): 942-967.

[8] Taylor, Cedric, and P. Hood. "A numerical solution of the Navier-Stokes equations using the finite element technique." Computers & Fluids 1.1 (1973): 73-100.

[9] Ferziger, Joel H., and Milovan Peric. Computational methods for fluid dynamics. Springer Science & Business Media, 2012.

[10] Elman, Howard C. Iterative methods for large, sparse, nonsymmetric systems of linear equations. Diss. Yale University, 1982.

[11] Van der Vorst, Henk A. "Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems." SIAM Journal on scientific and Statistical Computing 13.2 (1992): 631-644.

[12] Oancea, Bogdan, Tudorel Andrei, and Andreea Iluzia Iacob. "CUDA based iterative methods for linear systems." Computer Science 1 (2012): 228-232.

[13] Dehnavi, Maryam Mehri, et al. "Parallel sparse approximate inverse preconditioning on graphic processing units." Parallel and Distributed Systems, IEEE Transactions on 24.9 (2013): 1852-1862.

[14] Bolz, Jeff, et al. "Sparse matrix solvers on the GPU: conjugate gradients and multigrid." ACM Transactions on Graphics (TOG). Vol. 22. No. 3. ACM, 2003.

[15] Cormie-Bowins, Elise. "A comparison of sequential and GPU implementations of iterative methods to compute reachability probabilities." arXiv preprint arXiv:1210.6412 (2012).

[16] Jacobsen, Dana A., Julien C. Thibault, and Inanc Senocak. "An MPI-CUDA implementation for massively parallel incompressible flow computations on multi-GPU clusters." 48th AIAA aerospace sciences meeting and exhibit. Vol. 16. 2010.