Computer Physics Communications 183 (2012) 2364–2369

Contents lists available at SciVerse ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Overlapping computation and communication of three-dimensional FDTD on a GPU cluster

Ki-Hwan Kim [a,b], Q-Han Park [b,c,*]

[a] Korea Institute of Atmospheric Prediction Systems, Seoul 156-710, Republic of Korea

[b] Department of Physics, Korea University, Seoul, 136-701, Republic of Korea

[c] School of Computational Sciences, Korea Institute for Advanced Study, Seoul, 130-722, Republic of Korea

ARTICLE INFO

Article history:
Received 23 December 2011
Received in revised form
4 June 2012
Accepted 7 June 2012
Available online 15 June 2012

Keywords:
FDTD
GPU cluster
CUDA
OpenCL

ABSTRACT

Large-scale electromagnetic field simulations using the FDTD (finite-difference time-domain) method require the use of GPU (graphics processing unit) clusters. However, the communication overhead caused by slow interconnections becomes a major performance bottleneck. In this paper, as a way to remove the bottleneck, we propose the 'kernel-split method' and the 'host-buffer method' which overlap computation and communication for the FDTD simulation on the GPU cluster. The host-buffer method in particular enables overlapping without any modifications to the update-kernels that are already in use. We also present theoretical formulas to predict the overlap threshold and the total throughput for each method. By using our overlap methods with 6 GPU nodes, we demonstrate that the total performance of 3D FDTD reaches 92% of a six-fold increase, which is the upper limit that would be reached if there were no communication overhead.

1. Introduction

The finite-difference time-domain (FDTD) method is a popular numerical method for electromagnetic field simulations [1]. The three-dimensional (3D) FDTD simulations that are used for practical applications involving complex and large scale structures require large allocations of memory and have significant run times. A graphics processing unit (GPU) enables hardware accelerations suitable for the 3D FDTD simulation [2–6]. In a previous paper, we have demonstrated the increase in computational power that is achievable through the performance analysis and optimization of the 3D FDTD method on a single GPU [7]. To further increase the computational power and overcome the limited memory size of a single GPU, clustering of GPUs is necessary [8–10]. A GPU cluster is an aggregation of computers in which each node is equipped with single or multiple GPUs.

The main issues affecting the performance of FDTD using the GPU cluster are the load balancing and the communication overhead. Load balancing is required when the processing powers of each node are different or when the update-functions executed in each node are not the same. The load balancing for FDTD on a heterogeneous GPU cluster has been discussed thoroughly in the literature [11]. The communication overhead occurs in FDTD because field arrays at the boundary of the grid space in each node should be exchanged between nearest nodes. To reduce the communication overhead, high-end networking systems such as InfiniBand, Myrinet and 10 Gigabit Ethernet are often used [12,13]. These high-end networking systems have the merits of very low latency and high bandwidth compared to common networking systems such as Gigabit Ethernet. Recently, the GPUDirect technology for NVIDIA GPUs with Infiniband and Myrinet has enabled direct communication between GPUs in different nodes [14,15].

The communication overhead can also be reduced by overlapping computation and communication without resort to high-end networking systems. There have been reports on overlapping computation and communication in a CPU-based cluster [16]. The communication overhead is a critical factor in a GPU cluster because the processing speed of a GPU for FDTD is much faster than that of a CPU. In this paper, we propose two methods, which we call the 'kernel-split method' and the 'host-buffer method' that overlap computation and communication for 3D FDTD on the GPU cluster. The host-buffer method in particular enables the overlapping of computation and communication without any modifications to the update-kernels executed in the GPU that are already in use. We also present theoretical formulas to predict the overlap threshold and the total throughput for each method. To test the methods, we have performed theoretical analyses and compared them with experiments on a GPU Cluster with NVIDIA Tesla C2075 GPUs. By using

* Corresponding author at: Department of Physics, Korea University, Seoul, 136-701, Republic of Korea. Tel.: +82 010 5336 3106; fax: +82 2 927 3292.
E-mail address: qpark@korea.ac.kr (Q.-H. Park).

0010-4655/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.cpc.2012.06.003

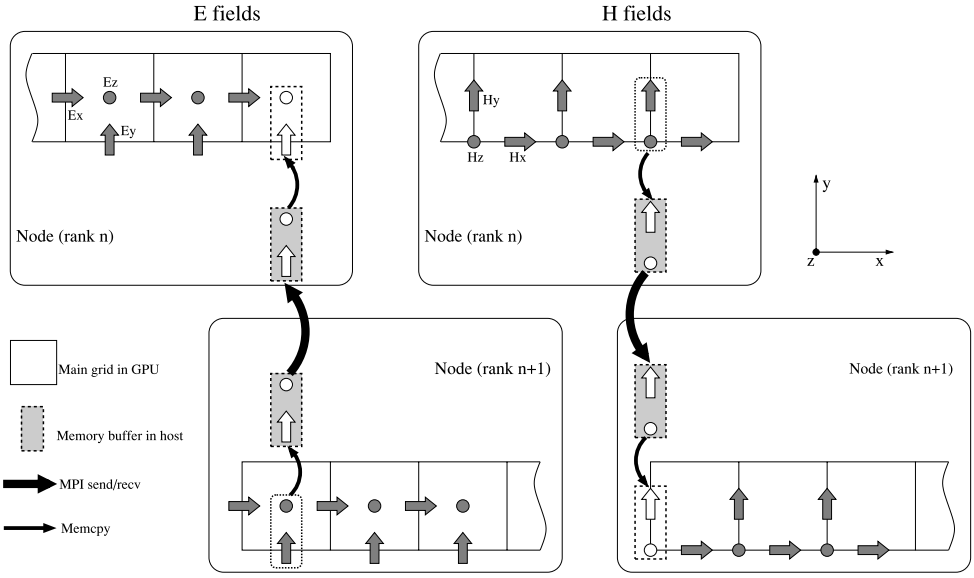

Fig. 1. Schematic diagram of communications between two nodes in the GPU cluster: two field arrays used for communication in the main grid are transmitted and received through memory copies and MPI communications. The main grid along the *y*-axis and *z*-axis is omitted for simplicity. The MPI nodes are arranged along the *x*-axis and each node communicates with its two nearest neighbors. Two nodes are placed above and below to emphasize the communication regions.

the overlap methods with 6 GPU nodes, we verify that the theoretical prediction agrees well with the experimental results. We also demonstrate that the total performance of 3D FDTD reaches 92% of the upper limit that would apply if there were no communication overhead.

2. Overlap methods

The process of communications for FDTD in the GPU cluster consists of three steps, as follows: (a) memory copy from GPU to host (b) MPI send and receive communications (c) memory copy from host to GPU. The communications between two nodes in the GPU cluster are shown schematically in Fig. 1. The main grid in Fig. 1 is a Yee grid where the electric fields are located at the face and the magnetic fields are located at the side in a cubic grid [1]. When the NVIDIA GPUDirect technology and specialized 3rd party network adaptors are used, the memory copy between the GPU and the host can be omitted.

In FDTD, the region used for communication is quite a lot smaller than the main region in most cases. If the communication region can be updated separately, the MPI communications can be overlapped with the update-kernels updating the main grid in the GPU. In this way, the MPI communication time is hidden by overlapping it with the execution time of the update-kernels. We propose the kernel-split method and the host-buffer method in order to overlap the computation and communication.

The kernel-split method directly splits the role of the update-kernels into two parts: (a) updating the region for communication and (b) updating the main region. First, the region used for communication is updated, then the updating of the main region and the MPI communication are executed simultaneously. Because the kernel of CUDA or OpenCL is originally executed asynchronously, user-level thread programming is not required. This method is intuitive. The main drawback, however, is that all update-kernels such as PML (Perfectly Matched Layer), PBC (Periodic Boundary Condition) and TFSF (Total Field/Scatter Field) must be split into two parts.

The host-buffer method utilizes the host-side buffer which is the two-layered FDTD region in the host. The procedure is as follows: When the updating of the main region in the GPU is started, the updating of the host-buffer is started simultaneously. Due to its small grid size, the updating of the host-buffer soon completes. The host-buffer immediately starts the MPI communication. Finally, the host-buffer copies the memory into or from the main region in the GPU. In the host-buffer method, Fig. 2 shows a schematic diagram of the data flow in two MPI nodes, while Fig. 3 shows a workflow and causal relationships between threaded processes in a MPI node. The greatest advantage of the host-buffer method is that no splitting of the kernel or function is required. This enables overlapping of the computation and the communication without any modifications to the update-kernels that are already in use. The buffers in the host often include additional update-functions such as PML, PBC and TFSF. Since the size of the buffer is much smaller than the size of the main grid in most cases, the total performance is little affected by the performance of the buffer. Thus the update-functions for the buffers can be implemented without painful performance optimization.

Fig. 4 schematically shows a comparison of the execution times for one time-step between non-overlap and overlap methods. The non-overlap method has understandably the longest execution time because computation and communication are executed sequentially. The overlap methods, on the other hand, can drastically reduce the execution time due to the simultaneous execution of computation and communication. The kernel-split method and the host-buffer method show little difference in terms of performance. Both of these overlap methods can hide the communication time behind the computation time when the update-kernel's execution time is long enough (which is the same as requiring that the main grid size is large enough).

3. Theoretical analysis

We provide the theoretical formula to determine the overlap condition through quantification of the execution times of each operation in Fig. 4. From this, we can quantitatively predict the total throughput in the GPU cluster as well as the threshold of the overlap condition. For simplicity, we assume that the MPI nodes are distributed along one axis and that a node only communicates with its two nearest-neighbor nodes. We use the abbreviations that are shown in Table 1.

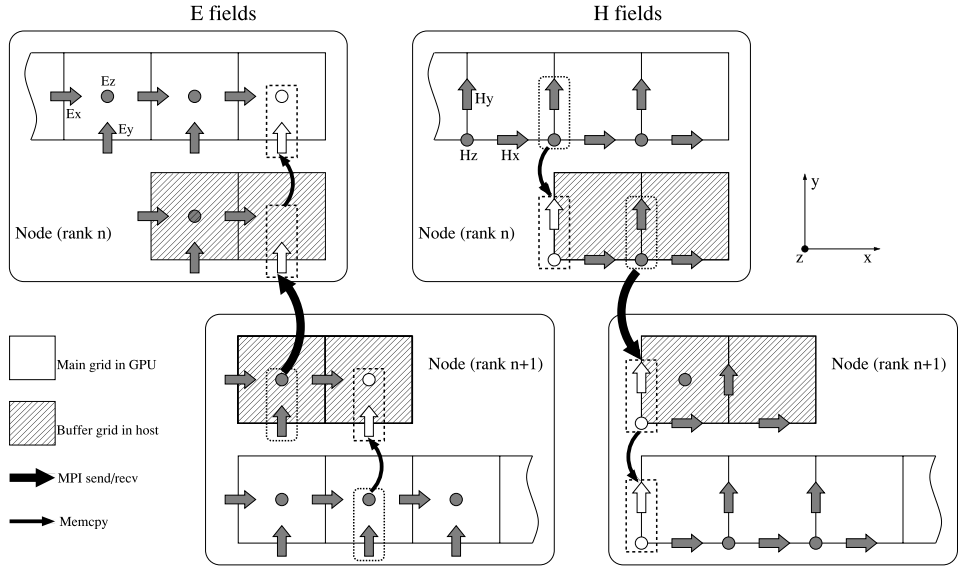

Fig. 2. Schematic diagram of communications using the host-buffer method: the host-buffer is wholly responsible for the MPI communications independently of updating the main region in the GPU.

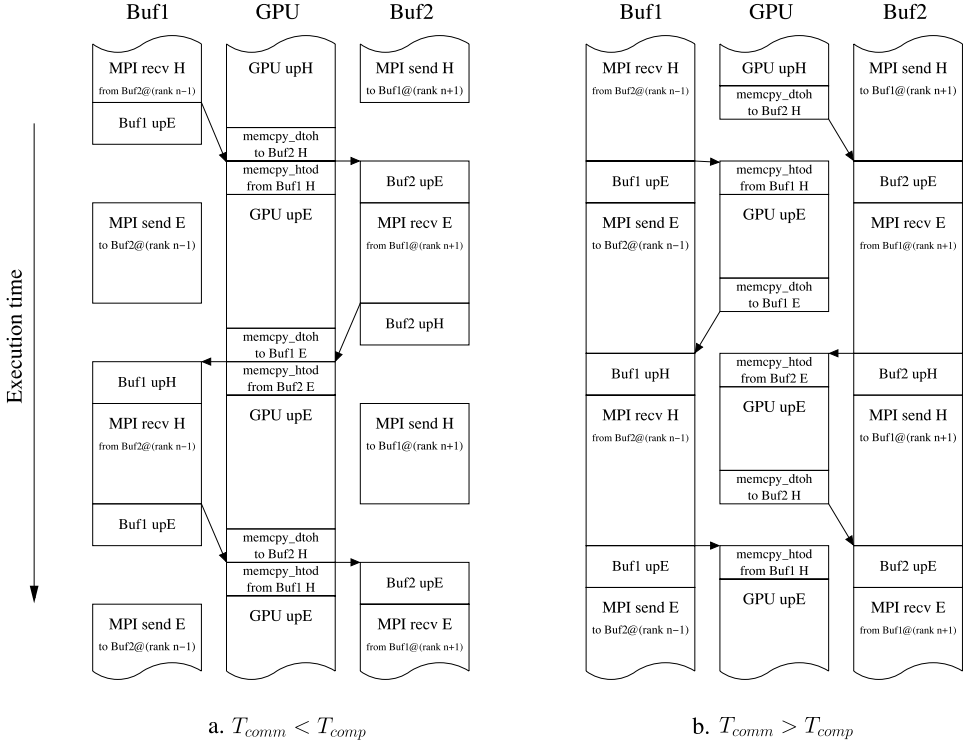

Fig. 3. Workflow and causal relationships of the host-buffer method in a MPI node (rank n): arrows indicate causal relationships between two threaded processes. The job indicated by the end point of the arrow must be executed after the job indicated by the start point of the arrow is completed. Two cases are presented where (a) the communication time is smaller than the computation time, or (b) the communication time is larger than the computation time.

Table 1
Abbreviations.

Abbreviation	Meaning	Unit
T	Execution time for one time-step	Second
GS	Grid size	Point
THP	Throughput of the 3D FDTD	Point/s
BW	Bandwidth	Byte/s
LT	Latency time	Second
n byte	Size of the floating-point data (Single precision: 4, double precision: 8)	Byte

The execution times of memory copy and MPI communication are given by

$$T_{memcpy} = 4 \times \left(\frac{2 \times \text{nbyte} \times \text{GS}_{comm}}{\text{BW}_{memcpy}} + \text{LT}_{memcpy} \right) \quad (1)$$

$$T_{mpi} = 4 \times \left(\frac{2 \times \text{nbyte} \times \text{GS}_{comm}}{\text{BW}_{mpi}} + \text{LT}_{mpi} \right) \quad (2)$$

where GS_{comm} is a grid size of the communication region. The BW_{memcpy} and LT_{memcpy} are the bandwidth and the latency of the memory copy between the GPU and the host. The BW_{mpi} and LT_{mpi}

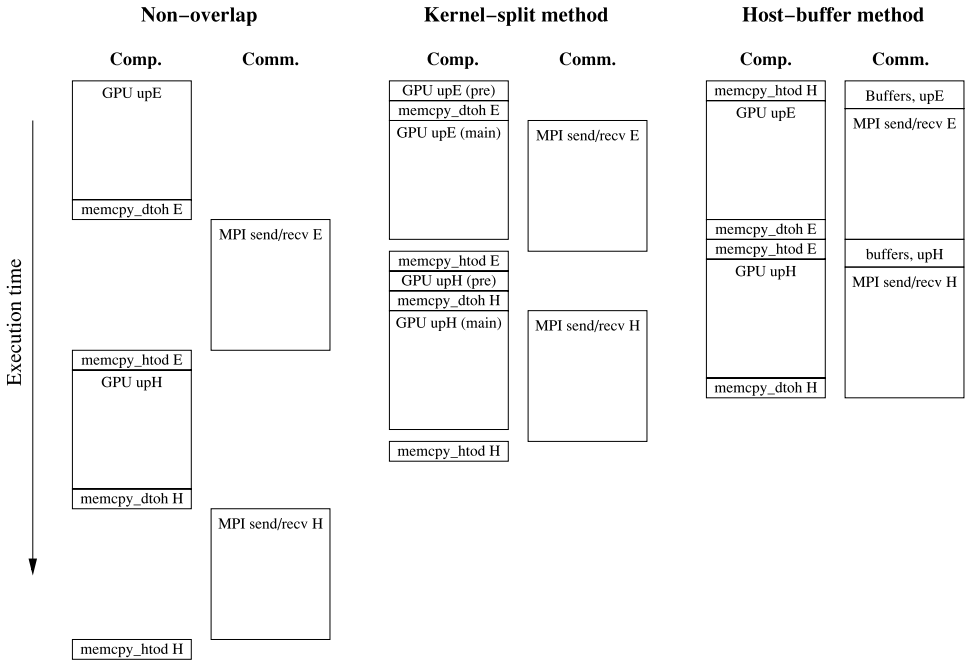

Fig. 4. Comparison of execution times for one time-step between non-overlap and overlap methods: the execution time of each operation is proportional to the height of the box. The label memcpy_d2h means the memory copy from the GPU to the host, while memcpy_h2d means the opposite.

are the bandwidth and the latency of the MPI communications. The coefficient 2 is the number of field arrays copied at each grid point. The coefficient 4 in Eq. (1) is the number of memory copies between the GPU and the host. The coefficient 4 in Eq. (2) is the number of MPI send/recv communications.

We define the computation time as the sum of update-kernel's execution time and the memcpy time:

$$T_{comp} = T_{memcpy} + \frac{\mathrm{GS}_{comp}}{\mathrm{THP}_{gpu}} \tag{3}$$

where GS_{comp} is the grid size of the main region and THP_{gpu} is the throughput of the FDTD in the GPU. The THP_{gpu} is the ratio of the grid size of the main region to the sum of the execution times of the update-kernels of FDTD, such as PML, PBC and TFSF. In a previous paper, we showed that if we know the operational intensity of each update-kernel and the memory bandwidth of the GPU, then we can predict the THP_{gpu} [7]. We do not consider the latency time of the kernel execution because it is negligible. The kernel-split and the host-buffer methods, therefore, have the same computation time. Similarly, the communication time is defined by

$$T_{comm} = \begin{cases} T_{mpi} + T_{memcpy} + \dfrac{\mathrm{GS}_{comm}}{\mathrm{THP}_{gpu}} & \text{(kernel-split method)} \\ T_{mpi} + \dfrac{\mathrm{GS}_{buffer}}{\mathrm{THP}_{buffer}} & \text{(host-buffer method)} \end{cases} \tag{4}$$

where GS_{buffer} is the grid size of the host-buffer and THP_{buffer} is the throughput of the single-thread FDTD in the CPU.

We can define the overlap condition from the computation time and the communication time as

$$T_{comp} \geq T_{comm}. \tag{5}$$

By substituting Eqs. (3) and (4) into Eq. (5), we obtain the conditions as

$$\frac{\mathrm{GS}_{comp}}{\mathrm{GS}_{comm}} \geq \begin{cases} 1 + \dfrac{\mathrm{THP}_{gpu}}{\mathrm{GS}_{comm}} \times T_{mpi} & \text{(kernel-split method)} \\ \mathrm{THP}_{gpu} \times \left(\dfrac{2}{\mathrm{THP}_{buffer}} + \dfrac{T_{mpi} - T_{memcpy}}{\mathrm{GS}_{comm}} \right) & \text{(host-buffer method)}. \end{cases} \tag{6}$$

We define the threshold of the overlap condition as the value satisfying the equality in Eq. (6). This equation shows that the communication time can be overlapped by the computation time when the ratio of grid sizes ($\mathrm{GS}_{gpu}/\mathrm{GS}_{comm}$) is greater than the threshold.

We can also estimate the total throughput in the GPU cluster through combinations of the computation time, the communication time and the overlap condition. We assume that all nodes have the same grid size as the main region, GS_{comp}. The total throughput is defined by

$$\mathrm{THP}_{total} = \begin{cases} \dfrac{\mathrm{GS}_{comp}}{T_{comp}} \times \mathrm{N}_{node} & \text{at } R \geq R_0 \\ \dfrac{\mathrm{GS}_{comp}}{T_{comm}} \times \mathrm{N}_{node} & \text{at } R < R_0 \end{cases} \tag{7}$$

where

$$R = \frac{\mathrm{GS}_{comp}}{\mathrm{GS}_{comm}} \tag{8}$$

and R is the ratio of grid sizes, R_0 is the threshold and N_{node} is the number of MPI nodes. Substituting Eqs. (3), (4) and (6) into Eq. (7) yields

$$\mathrm{THP}_{total}|_{R \geq R_0} = \left(\frac{1}{\mathrm{THP}_{gpu}} + \frac{1}{R} \times \frac{T_{memcpy}}{\mathrm{GS}_{comm}} \right)^{-1} \times \mathrm{N}_{node} \tag{9}$$

$$\mathrm{THP}_{total}|_{R < R_0} = \begin{cases} \left(\dfrac{1}{\mathrm{THP}_{gpu}} + \dfrac{T_{mpi} + T_{memcpy}}{\mathrm{GS}_{comm}} \right)^{-1} \\ \quad \times R \times \mathrm{N}_{node} \quad \text{(kernel} - \text{split)} \\ \left(\dfrac{2}{\mathrm{THP}_{buffer}} + \dfrac{T_{mpi}}{\mathrm{GS}_{comm}} \right)^{-1} \times R \times \mathrm{N}_{node} \\ \quad \text{(host} - \text{buffer)}. \end{cases} \tag{10}$$

We consider the GS_{comm} as a constant because in most cases the ratio of grid sizes (R) is only changed upon domain decomposition in the GPU cluster. In Eq. (9), if the ratio of grid sizes is large enough or the memcpy time is small enough, the total throughput comes close to the upper limit of $\mathrm{THP}_{gpu} \times \mathrm{N}_{node}$. The ratio of grid sizes

Table 2
Experimental throughput: the update-kernel executed in the GPU was written by OpenCL-C. A single-threaded FDTD program without optimizations was used for the update-function executed in the CPU. We used single precision floating-point numbers.

Processor	GPU	CPU
Model	NVIDIA Tesla C2075	Intel i5-2500
Throughput (Mpoint/s)	1188.02	38.26

Table 3
Experimental bandwidth and latency: the memcpy function included the memory copy via the PCI-Express bus as well as combining and splitting the two fields arrays. The network system used for MPI communications was Gigabit Ethernet. The full-duplex mode of a Gigabit Ethernet switch was used.

	Memory copy	MPI (Gigabit Ethernet)
Bandwidth (MB/s)	1690.72	193.87
Latency (ms)	0.081	0.63

can be increased up to the limit of the GPU memory size. The memcpy time can be decreased by using technologies such as zero-copy, page-locked memory and PCI-Express v3.0 [17,18].

On the whole, the total throughput tends to be decreased as the ratio of grid sizes is decreased. Especially, when the ratio of grid sizes is smaller than the threshold, the total throughput is decreased linearly. Contrary to the behavior under the threshold, the total throughput is relatively constant above the threshold. In the practical sense, consequently, the optimum ratio of grid sizes should be just above the threshold in order to achieve efficient overlapping.

4. Experimental results

We compare the theoretical predictions for the overlap threshold and the total throughput with experimental results obtained using NVIDIA Tesla C2075 GPUs. We measure experimentally the throughput, bandwidth and latency as shown in Tables 2 and 3 to obtain the theoretical predictions of the threshold from Eqs. (6) and the total throughput from Eqs. (9) and (10). We first measure the execution times for the update-kernel, update-function, memory copy and MPI communication when the grid size is varied from 1 Mpoints to 25 Mpoints. We also determine the throughput, bandwidth and latency by using the gradients and y-intercepts of the first degree polynomials that were linearly fitted to the execution times. The throughput is measured by only the fundamental methods of FDTD without any additional methods such as PML, PBC or TFSF. Note that the throughput can be decreased if these methods are included in the update-kernel and update-function. The update-kernels executed in the GPU were fully-optimized by using optimization methods which are summarized in the literature [7]. The update-functions executed in the CPU were not optimized. We did not optimize for the buffer region in the host to implement easily the additional update-functions such as PML, PBC and TFSF. The calculation using CPU was performed with a single core and without the SIMD vectorization. The total performance, however, is little affected by the non-optimized CPU codes.

Fig. 5 shows that the theoretical predictions of the overlap threshold and the total throughput are in agreement with the experimental results. The theoretical predictions are obtained by substituting the quantities in Tables 2 and 3 into Eqs. (6), (9) and (10). This confirms that the theoretical formulas can be a useful guideline for achieving high performance in the GPU cluster as follows: (a) The memcpy time is dominant for the performance enhancement from Eq. (9) when the ratio of grid sizes is larger than the threshold. (b) The MPI communication time is also dominant for the performance enhancement form Eq. (10) when the ratio of grid sizes is smaller than the threshold. (c) The ratio of grid sizes

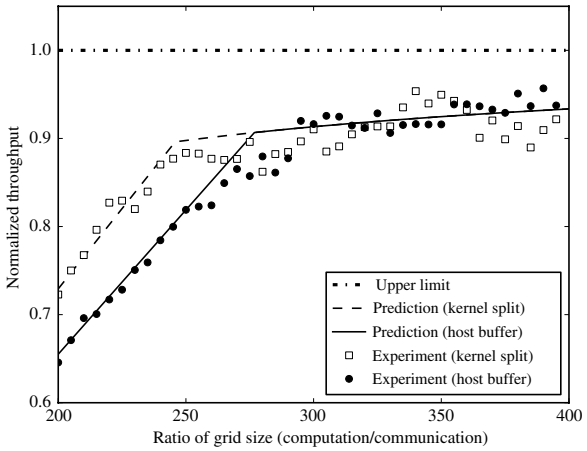

Fig. 5. Comparison between theoretical predictions and experimental results in terms of the total throughput: the dash–dot line on the top represents the upper limit which is the throughput of a single GPU. The dashed line and the solid line represent the theoretical predictions. The square and circle symbols represent the experimental results.

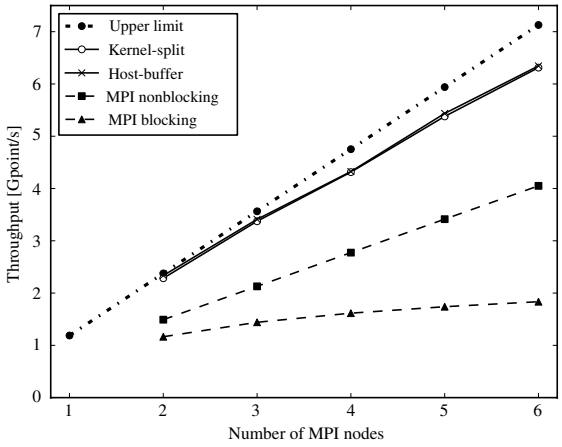

Fig. 6. Comparison of the total throughput between the non-overlap methods and the overlap methods: the dash–dot line on the top represents the upper limit which is a multiple of the single GPU throughput. The solid lines represent the kernel-split and the host-buffer overlap methods. The dashed lines represent the non-overlap methods using MPI non-blocking and MPI blocking communications. All symbols represent the total throughput obtained experimentally in each MPI node. Each node had a single NVIDIA Tesla C2075 GPU. The network system is Gigabit Ethernet and it uses commodity hardwares.

should be above the threshold for overlapping the computation and the communication.

Fig. 6 shows a comparison of the total throughput between the non-overlap methods and the overlap methods in the GPU cluster according to the size of the MPI nodes. We choose the main grid size to exceed the overlap threshold. The same main grid size of $360 \times 250 \times 256$ was allocated to each node, of which the memory size was about 800 MBytes.

The total throughput of each overlap method increased linearly with the size of the MPI nodes, as described by Eq. (9). The total throughput of the MPI non-blocking communication also increased linearly with the size of the MPI nodes. On the other hand, the total throughput of the MPI blocking communication showed little increase with the size of the MPI nodes. This is because only two nodes can communicate simultaneously when the MPI blocking communication occurs.

The kernel-split and host-buffer overlap methods show performances that are much higher than those of the non-overlap methods. The total throughput of the overlap methods reaches 92% of the upper limit. It is noteworthy that the high performance was

Table 4
Experimental equipments.

	Model/version
GPU	NVIDIA tesla C2075
CPU	Intel core i5-2500 3.3 GHz
Mainboard	Gigabyte GA-EP45-DS5
Network adaptor	Realtek RTL-8111C integrated chipset
Switch hub	Netgear GS748T Gigabit smart switch
Operating system	Debian GNU/linux testing (wheezy) version
NVIDIA driver	v275.28
Compilers	GCC v4.6.1 and NVCC v4.0
MPI library	OpenMPI v1.4.3

obtained in a Gigabit Ethernet environment without a high-end networking system such as InfiniBand, Myrinet or 10 G Ethernet. If optimizing methods to reduce the memcpy time such as zero-copy are applied, the total throughput comes close to the upper limit. The experimental results show that our overlap methods are very practical even in a GPU cluster with Gigabit Ethernet using commodity hardware. Table 4 shows the experimental equipment. All of the cluster nodes are identical. For the FDTD simulations, we used the periodic boundary condition in all directions, The sine-wave line source was excited to generate a cylindrical wave propagation.

5. Conclusion

We proposed the 'kernel-split method' and the 'host-buffer method', which overlap computation and communication of the 3D FDTD on the GPU cluster. The kernel-split method directly splits the update-kernels into two parts. The host-buffer method utilizes the host-side FDTD buffer. There is no difference between the two methods in terms of performance. The host-buffer method in particular has a great merit in that it does not require any splitting of the kernel or function. This enables overlapping without any modifications to the update-kernels that are already in use.

We have also presented the theoretical formula to predict the threshold of the overlap condition and the total throughput of the GPU cluster. The theoretical formula can also be a useful guideline for optimizations to achieve high performance. We have confirmed that the theoretical predictions for the overlap threshold and the total throughput are in reasonable agreement with the experimental results by using NVIDIA Tesla C2075 GPUs. In addition, we demonstrated that the proposed methods are very practical even in a GPU cluster with a Gigabit Ethernet environment, which is the most common networking system.

Furthermore, we point out that our overlapping methods will be useful for stencil (nearest-neighbor) computations, which are widely used to numerically solve various partial differential equations. Since the region for communication is much smaller than the main region in most stencil computations, our methods can be well applied.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2011-0020205, 2011-0019170 and 2011-0029807).

References

[1] A. Taflove, S.C. Hagness, Computational Electrodynamics: The Finite-Difference Time-Domain Method, third ed., Artech House, Norwood, MA, 2005.
[2] S. Adams, J. Payne, R. Boppana, Finite difference time domain (FDTD) simulations using graphics processors, in: 2007 DoD High Performance Computing Modernization Program Users Group Conference, Vol. 2007, IEEE, 2007, pp. 334–338.
[3] M. Inman, A. Elsherbeni, IEEE Antennas and Propagation Magazine 47 (2005) 71.
[4] S. Krakiwsky, L. Turner, M. Okoniewski, Acceleration of finite-difference time-domain (FDTD) using graphics processor units (GPU), in: 2004 IEEE MTT-S International Microwave Symposium Digest (IEEE Cat. No. 04CH37535), Vol. 2, IEEE, 2004, pp. 1031–1034.
[5] P. Sypek, A. Dziekonski, M. Mrozowski, IEEE Transactions on Magnetics 45 (2009) 1324.
[6] M.R. Zunoubi, J. Payne, W.P. Roach, IEEE Antennas and Wireless Propagation Letters 9 (2010) 756.
[7] K.-H. Kim, K. Kim, Q.-H. Park, Computer Physics Communications 182 (2011) 1201.
[8] A. Kaufman, S. Yoakum-Stover, GPU cluster for high performance computing, in: Proceedings of the ACM/IEEE SC2004 Conference, IEEE, 2004, pp. 47–47.
[9] V.V. Kindratenko, et al., GPU clusters for high-performance computing, in: 2009 IEEE International Conference on Cluster Computing and Workshops, IEEE, 2009, pp. 1–8.
[10] S. Rostrup, H. De Sterck, Computer Physics Communications 181 (2010) 2164.
[11] R. Shams, P. Sadeghi, Journal of Parallel and Distributed Computing 71 (2011) 584.
[12] W. Feng, P. Balaji, C. Baron, L. Bhuyan, D. Panda, Performance characterization of a 10-Gigabit ethernet TOE, in: 13th Symposium on High Performance Interconnects, HOTI'05, IEEE, 2005, pp. 58–63.
[13] B. Chandrasekaran, D. Buntinas, S. Kini, D. Panda, P. Wyckoff, IEEE Micro 24 (2004) 42.
[14] H. Wang, et al., Computer Science — Research and Development 26 (2011) 257.
[15] M.J. Rashti, A. Afsahi, 10-Gigabit iWARP ethernet: comparative performance analysis with infiniband and myrinet-10G, in: 2007 IEEE International Parallel and Distributed Processing Symposium, IEEE, 2007, pp. 1–8.
[16] D.P. Rodohan, S.R. Saunders, R.J. Glover, International Journal of Numerical Modelling: Electronic Networks, Devices and Fields 8 (1995) 283.
[17] D.P. Playne, K.A. Hawick, Asynchronous communication schemes for finite difference methods on multiple GPUs, in: 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, IEEE, 2010, pp. 763–768.
[18] NVIDIA, NVIDIA CUDA Best Practice Guide, NVIDIA Corporation, Santa Clara, CA, 2011, p. 95050.