33rd Annual International Conference of the IEEE EMBS
Boston, Massachusetts USA, August 30 - September 3, 2011

# Multi-GPU Accelerated Three-Dimensional FDTD Method for Electromagnetic Simulation

Tomoaki Nagaoka, *Member, IEEE,* and Soichi Watanabe, *Member, IEEE*

***Abstract*—Numerical simulation with a numerical human model using the finite-difference time domain (FDTD) method has recently been performed in a number of fields in biomedical engineering. To improve the method's calculation speed and realize large-scale computing with the numerical human model, we adapt three-dimensional FDTD code to a multi-GPU environment using Compute Unified Device Architecture (CUDA). In this study, we used NVIDIA Tesla C2070 as GPGPU boards. The performance of multi-GPU is evaluated in comparison with that of a single GPU and vector supercomputer. The calculation speed with four GPUs was approximately 3.5 times faster than with a single GPU, and was slightly (approx. 1.3 times) slower than with the supercomputer. Calculation speed of the three-dimensional FDTD method using GPUs can significantly improve with an expanding number of GPUs.**

## I. Introduction

NUMERICAL simulation with anatomically realistic human models has recently been performed for studies on medical applications and biological effects [1], [2]. The finite-difference time-domain (FDTD) method, one of the electromagnetic analysis methods, has mainly been used in these studies using human models. High-performance computer systems, such as supercomputers, were until a few years ago generally needed to perform FDTD calculation with human models, due to the problem of large memory and heavy central processing unit (CPU) time. Recently, general-purpose computing on a graphics processing unit (GPGPU) has received considerable attention in many scientific fields [3]-[5] because a GPGPU offers high computational performance at low cost.

We previously implemented the three-dimensional FDTD method on a GPU using Computer Unified Device Architecture (CUDA) [6], and also found that three-dimensional FDTD calculation using a single GPU can significantly reduce run time compared to when using a conventional CPU, even with a native GPU implementation of the three-dimensional FDTD method [7]. However, the available memory of a single GPU is limited.

In this paper, to realize large scale computing with a human model, we adapt three-dimensional FDTD code to multi-GPU environments using CUDA. FDTD calculation using multi-GPU is also expected to greatly reduce run time in comparison with that of a single GPU. The approach is adapted to simulate an electromagnetic field using a human model, and its performance is evaluated.

T. Nagaoka and S. Watanabe are with the Electromagnetic Compatibility Laboratory, Applied Electromagnetic Research Institute, National Institute of Information and Communications Technology, Tokyo 184-8795, Japan. (e-mail: nagaoka@nict.go.jp; wata@nict.go.jp).

## II. Multi-GPU Implementation of FDTD Method

### A. CUDA and GPU parallel computation

CUDA framework provides GPU parallel computation capability including a programming and execution environment. CUDA also supports multiple GPU subsystems (multi-GPU) in a single system. Figure 1 shows a high-level view of the multi-GPU system and CUDA framework.

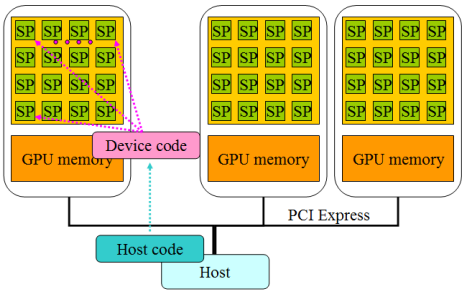


Fig. 1. CUDA and GPU parallel computing

A program should be converted to CUDA code by a CUDA compiler in order to be run on a GPU. CUDA code consists of device code and host code. The device code runs on the GPU while the host code runs on a CPU. The device code written as a kernel program executes a fragment of computation code, such as for-loop body code, that can be computed in parallel. CUDA generates thousands of GPU threads that simultaneously execute device code on stream processors. The host code allocates and frees GPU memory, transfer data between CPU and GPU, and controls execution of the device code.

A GPU has hundreds of processors, called stream processors (SPs). It is capable of executing thousands of GPU threads that CUDA generates, which are bundled into a thread block. Thread blocks are arranged into a grid [8]. CUDA maps parallelizable code fragment to these GPU threads.

CUDA supports multi-GPU by controlling multiple GPUs with a single program. Each GPU requires dedicated CPU threads and a multi-GPU system should be equipped with the same number of GPUs and CPUs. GPU memory is dedicated to a single GPU and cannot be addressed from either a CPU or another GPU. In this sense, multi-GPU can be regarded as a distributed memory multiprocessor.

978-1-4244-4122-8/11/$26.00 ©2011 IEEE

### B. *Three-dimensional FDTD computation on a single GPU*

FDTD is the most popular method of computational electromagnetic simulation because of its simple algorithm and very high computational efficiency using modern computer equipment [9]. The method also has application with an anatomically realistic human model with complex shape and internal structure [2].

Figure 2 shows the total flow of the three-dimensional FDTD method for a single GPU computation. There are four tasks within each time step for the GPU side in this figure: electric field computation (e_field), perfectly matched layers (PML) computation for electric field (e_pml) as the absorbing boundary condition [10], magnetic field computation (h_field) and PML for magnetic field (h_pml). All field updates in each time step can be parallelized and are offloaded to the GPU.

Our original three-dimensional FDTD program is written in C and converted to CUDA code to be executed on a single GPU. The three-dimensional FDTD process starts with initialization tasks such as memory allocation, parameters and GPU setup. Initial values for GPU computation are then transferred to GPU memory. Time step loop including e_field, e_pml, h_field and h_pml follows. After the time ends, results of GPU computation are transferred from GPU to Host (CPU). Finally, SAR computation is done by Host. Our GPU three-dimensional FDTD program requires no data transfer for field updates because all computations within each time step are purely executed by GPU.

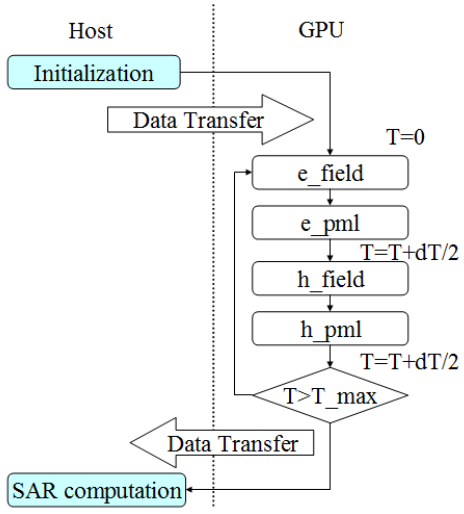


Fig. 2. FDTD process flow for single GPU processing

Electric field and magnetic field are three-dimensional vector fields. Six three-dimensional arrays are used for the field component; EX, EY, EZ, HX, HY and HZ. All the field computations, e_field, e_pml, h_field and h_pml, contain three-dimensional parallelizable for-loop and are converted to CUDA kernel codes. These kernel codes are called sequentially for each time step.

A typical for-loop would be:

```
for( z = 0; z < NZ; z ++)
  for( y = 0; y < NY; y ++)
    for( x = 0; x < NZ; x ++)
      EX[z][y][x] = c1 * (HZ[z][y][x] - HZ[z][y-1][x])
                    - c2 * (HY[z][y][x] - HY[z-1][y][x])
```

Since CUDA currently does not support a three-dimensional grid, up to two-dimensional for-loop can be parallelized in a straight way. We put a for-loop for the Z direction in a kernel program.

### C. *Multi-GPU parallelism*

For multi-GPU parallelism, it is necessary to partition data arrays such as electric fields among multiple GPUs. We divide three-dimensional arrays in the Z direction [11]. As noted above, each GPU is capable of addressing only local GPU memory and ghost nodes should be added to each instance of decomposed data. FDTD schema is the first-order partial differential equation, and it is necessary to add a single slice of ghost nodes at the boundary of the decomposed data. These ghost node values need to be updated for each time step by communicating with the neighboring GPU [8], [11].

We describe ghost node updates with a two GPU system, but extending to n-GPU is straight-forward. Let GPU#0 holds E and H arrays from Z=1 to Z=100 and GPU#1 holds Z=101 to Z=200. EX[z][y][x] depends on HY[z][y][x] and HY[z-1][y][x]. We note the dependency as:

EX[z][y][x] ← HY[z][y][x] , HY[z-1][y][x]

In this notation, HY depends on EX as:

HY[z][y][x] ← EX[z][y][x] , EX[z+1][y][x]

At the boundary of data for each GPU, z=100:

For GPU #0

EX[100][y][x] ← HY[100][y][x], HY[99][y][x]

HY[100][y][x] ← EX[100][y][x], EX[101][y][x]

For GPU #1

EX[101][y][x] ← HY[101][y][x], HY[100][y][x]

HY[101][y][x] ← EX[101][y][x], EX[102][y][x]

From these dependencies, it is necessary to add EX[101][y][x] for all x and y to data of GPU#0 as ghost nodes. At the same time, HY[100][y][x] for any x and y are added to the data of GPU#1 as ghost nodes.

EX, HY and other variables are updated at each time step, but ghost node values are only referenced and not updated. Updated values are copied from the neighboring GPU. For example HY[100][y][x] are updated on GPU#0, and ghost nodes HY[100][y][x] on GPU#1 are updated by copying the value on GPU#0.

Figure 3 shows ghost nodes and updates via GPU communication. The left figure shows ghost nodes of EX [101][y][x] for GPU#0 and the right figure HY[101][y][x] for GPU#1. GPU#1 is responsible for updating EX[101][y][x]. After GPU#1 updates EX[101][y][x], the data value is transferred to GPU#0. Ghost nodes of HY[101][y][x] are managed in the same way as EX. Other variables such as EY, EZ, HX and HZ are managed in the same way as EX and HY.

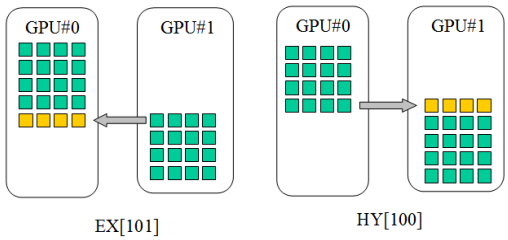


Fig. 3. Ghost nodes updates and GPU communication for each step

### D. Implementation of the multi-GPU approach with Open MP

Our target multi-GPU system is a single system with multiple GPUs. GPU-GPU communication is achieved via host memory on this system, which is a combination of GPU-Host and Host-GPU data transfer. Each GPU requires dedicated CPU threads. OpenMP is used to create CPU threads and control their execution including thread synchronization. Figure 4 shows a two-GPU case of multi-GPUed FDTD time step. A single-GPU version of the FDTD time step is replaced with that of the multi-GPU version.

The first OpenMP task is to create a parallel region with the number of GPUs threads. After electric field computation (e_field, e_pml) is completed on two GPUs, ghost nodes for the electric field are updated with GPU#1 values (right-left arrow). CPU threads are synchronized before starting magnetic field computation (h_field, h_pml) with OpenMP barrier synchronization at GPU-GPU communication. After magnetic field computation is completed, ghost nodes for the magnetic field are updated with GPU#0 values (left-right arrow).

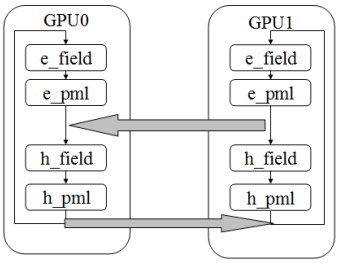


Fig. 4. Multi-GPU version of FDTD time step loop

## III. Performance Tests

### A. Computational environment

In this study, we used an NVIDIA Tesla C2070 as the GPGPU board. It has 448 scalar processer (SP) cores and 6GB GDDR5 RAM. The core clock and memory bandwidth of the board are also 1.15 GHz and 144 GB/sec, respectively. We used a workstation equipped with Intel 4-Core Xeon X5620 (2.4GB) as the CPU and 48 GB memory (6 × 8 GB DDR3-1066 with ECC), and Ubuntu 10.04 64-bit Linux as the operating system. The four GPU boards (Tesla C2070) were mounted on a workstation. Therefore, GPU memory of 24 GB is available in the environment. We used the NVIDIA Linux driver (190.18) for the GPU, and gcc 4.3.3 and CUDA 3.2 as the compilers of the CPU and GPU, respectively.

### B. Performance of multi-GPU FDTD calculation

To examine the performance of the FDTD calculation on multi-GPU, we conducted a test using the calculation model shown in Fig. 5. The model was a cube domain, and its center was allocated a sphere (25 cells in radius). Cell size was 2 mm. The absorbing boundary condition (ABC) is the eight-layer PML. The time step was 3.813 *ps* and the total number of steps was 656. The incident wave was an assumed plane wave.

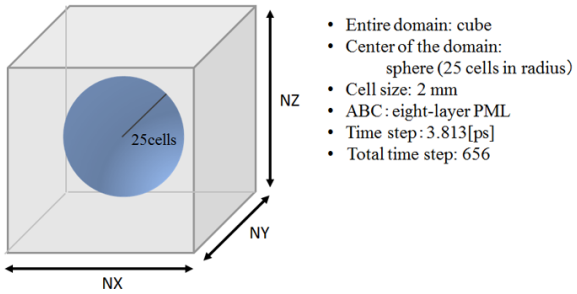


Fig. 5. Calculation model for performance test

Figure 6 shows the calculation time of single and multi-GPUs. The calculation domain was 550 × 550 × 550 cell size. Computation time is calculated for part of the electromagnetic field update. The thread block size is 32 × 16. Calculation speeds with two, three and four GPUs were approximately 2, 2.9 and 3.5 times faster, respectively, than with a single GPU. Calculation time with four GPUs was also approximately 235 times faster than with a CPU. In contrast, GPU-GPU communication time is much shorter than total calculation time. The result shows that the number of GPUs has a marked influence on calculation speed and also that calculation with multi-GPU has high scalability.

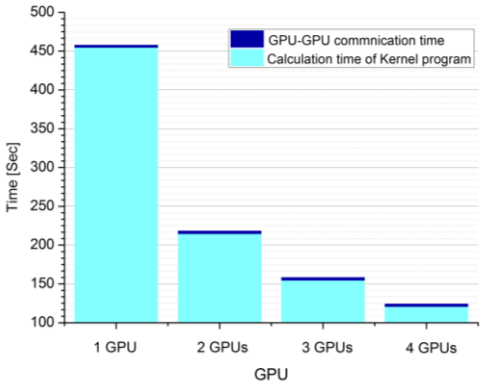


Fig. 6. GPU/CPU speed ratio

### C. Performance test for a specific example using a human model

We usually use a vector supercomputer SX-8R (NEC) for the three-dimensional FDTD calculation using anatomically realistic human models because the CPU takes a very long time to run a FDTD calculation. We therefore

compared the run time between multi-GPU (four GPUs) and a supercomputer in the case of a plane wave incidence for a numerical human model. In this performance test, we used an anatomically realistic pregnant female model at 26 weeks gestation developed, by Nagaoka et. al [12]. The model consists of $2 \times 2 \times 2$ mm$^3$ voxels and has 56 different tissues. The human model was assumed to be in free space. The cell size of the calculation domain was $2 \times 2 \times 2$ mm$^3$. The absorbing boundary condition was set by eight-layer PML. The PML boundaries were set 30 cells (60 mm) away from all parts of the human model, as shown in Fig. 7. The overall calculation domain was thus $211 \times 329 \times 878$ cells.

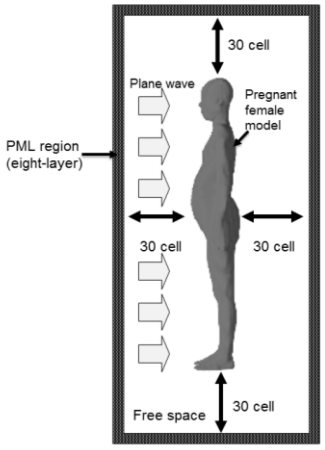


Fig. 7. Calculation condition with a human model

Figure 8 shows the results of the comparison. Run times in this figure are continuously computed times from 30 MHz to 3 GHz (a total of 22 frequencies). The run times with the multi-GPU and one node (eight CPUs) of the supercomputer were 30,372 sec and 23,665 sec, respectively. The thread block size for the GPU calculation was $32 \times 16$ in this test. Run time on the multi-GPU was slightly (approx. 1.3 times) slower than that using the supercomputer, while run time on a single GPU (NVIDIA Tesla C1060) was more than five times slower than those using one node of the supercomputer in a previous study [7]. We found that multi-GPU is able to significantly accelerate three-dimensional FDTD calculation with positive implications for practical application. The result also indicated that multi-GPU FDTD calculation may be faster than the supercomputer if we use a workstation equipped with more than four GPUs.

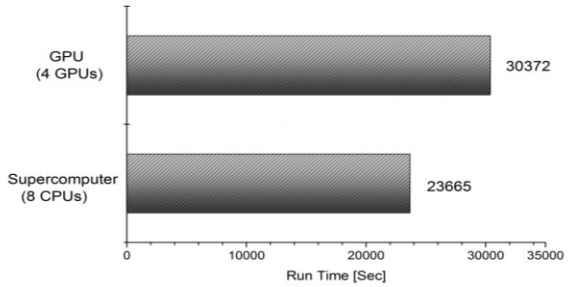


Fig. 8. Comparison run time of FDTD calculation between GPU and supercomputer

## IV. Conclusion

We implemented the three-dimensional FDTD method on multi-GPU using CUDA. This study used the NVIDIA Tesla C2070 as GPGPU boards. The GPU memory of 24 GB became available for the FDTD calculation in the computational environment. We tested the performance of an FDTD calculation on multi-GPU. The results indicated the calculation speed of three-dimensional FDTD using GPUs can significantly improve with an expanding number of GPUs.

## Acknowledgment

Parts of this work were carried out using the vector supercomputer SX-8R (NEC) at the National Institute of Information and Communications Technology.

## References

[1] J. Kim and Y. Rahmat-Samii, "Implanted antennas inside a human body: simulation, designs, and characterizations," IEEE Trans. Microwave Theory Tech., vol. 52, pp. 1934-1943, Mar. 2004.

[2] J. W. Hand, "Modeling the interaction of electromagnetic fields (10MHz-10GHz) with the human body: methods and applications," *Phys. Med. Biol*., vol. 52, pp. R243-286, Jul. 2008.

[3] M. de Greef, J. Crezee, J. C. van Eijk, R. Pool and A. Bel, "Accelerated ray tracing for radiotherapy dose calculations on a GPU," *Med. Phys.*, vol. 36, pp. 4095-4102, Sep. 2009.

[4] S. S. Samant, J. Xia, P. Muyan-Ozcelik and J. D. Owens, "High performance computing for deformable image registration: towards a new paradigm in adaptive radiotherapy," *Med. Phys.*, vol. 35, pp. 3546-3553, Aug. 2008.

[5] N. Takada, T. Shimobaba, N. Masuda and T. Ito, "High-speed FDTD simulation algorithm for GPU with compute unified device architecture," in *Proc. IEEE-APS/URSI Int. Symp*., pp. 1-4, 2009.

[6] NVIDIA, NVIDIA *CUDA programming guide version* 3.2, NVIDIA Corporation, 2010.

[7] T. Nagaoka and S. Watanabe, "A GPU-based calculation using the three-dimensional FDTD method for electromagnetic field analysis," In *Proceedings of IEEE EMBC 2010*, pp. 327-330, Buenos Aires, 2010.

[8] P. Micikevicius, "3D finite difference computation on GPUs using CUDA," In *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units (GPGPU-2)*, pp. 79-84, 2009.

[9] A. Taflove and S. C. Hagness, *Computational Electromagnetics: The Finite-Difference Time-Domain Method, 3rd ed.*, London: Artech House Publishers, 2005.

[10] J. P. Berenger, "A perfectly matched layer for the absorption of electromagnetic waves," *Journal of Computational Physics*, pp. 185-200, 1994.

[11] D. A Jacobsen, J. C. Thibault, and I. Senocak. "An MPI-CUDA Implementation for Massively Parallel Incompressible Flow Computations on Multi-GPU Clusters," *48th AIAA Aerospace Sciences Meeting and Exhibit*, 2010.

[12] T. Nagaoka, T. Tagashi, T. Saito, K. Takahashi, K. Ito and S. Watanabe, "An anatomically realistic whole-body pregnant-woman model and specific absorption rates for pregnant-woman exposure to electromagnetic plane waves from 10 MHz to 2 GHz," *Phys. Med. Biol.*, vol. 52, pp. 6731-674, Nov. 2007.