34th Annual International Conference of the IEEE EMBS
San Diego, California USA, 28 August - 1 September, 2012

Accelerating Three-Dimensional FDTD Calculations on GPU Clusters for Electromagnetic Field Simulation

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

Abstract—**Electromagnetic simulation with anatomically realistic computational 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 computational human model, we adapt three-dimensional FDTD code to a multi-GPU cluster environment with Compute Unified Device Architecture and Message Passing Interface. Our multi-GPU cluster system consists of three nodes. The seven GPU boards (NVIDIA Tesla C2070) are mounted on each node. We examined the performance of the FDTD calculation on multi-GPU cluster environment. We confirmed that the FDTD calculation on the multi-GPU clusters is faster than that on a multi-GPU (a single workstation), and we also found that the GPU cluster system calculate faster than a vector supercomputer. In addition, our GPU cluster system allowed us to perform the large-scale FDTD calculation because were able to use GPU memory of over 100 GB.**

I. Introduction

NUMERICAL simulation with computational human models in anatomy has recently been performed for studies on medical applications and biological effects [1], [2]. The finite-difference time-domain (FDTD) method [3], one of the electromagnetic analysis methods, has mainly been used in these studies using computational human models. High-performance computer (HPC) systems, such as supercomputers, were until a few years ago generally needed to perform the FDTD calculation with computational human models, because the FDTD calculation runs very slowly and requires a large amount of computational memory.

Recently, general-purpose computing on a graphics processing unit (GPGPU) has received considerable attention in many scientific fields [4]-[6] because a GPGPU offers high computational performance at low cost. We previously implemented the three-dimensional FDTD method on a single GPU using Computer Unified Device Architecture (CUDA) [7], and also found that three-dimensional FDTD calculation using a single GPU (NVIDIA Tesla C1060) can significantly reduce run time compared to when using a conventional CPU (Intel Xeon X5450) , even with a native GPU implementation of the three-dimensional FDTD method [8]. However, the available memory of a single GPU is very small to perform the three-dimensional FDTD calculation. Therefore, in order to improve the calculation speed and the available memory, we adapt the FDTD code to multi-GPU environments [8].

In this paper, to more efficiently perform the large-scale FDTD computation with real computational human model, we attempt to implement the three-dimensional FDTD code on multi-GPU cluster environment. The approach is adapted to simulate an electromagnetic field using a human model, and its performance is evaluated.

II. Converting CPU FDTD Code To GPU-Cluster Code

A. Parallel Computation on Multi GPU Cluster

GPU Cluster consists of multiple PCs, which is called "cluster nodes", connected with high speed network such as "Infiniband". Each cluster node is equipped with single or multiple GPU subsystem that is a board with GPU and dedicated GPU memory. Figure 1 shows a GPU Cluster system architecture.

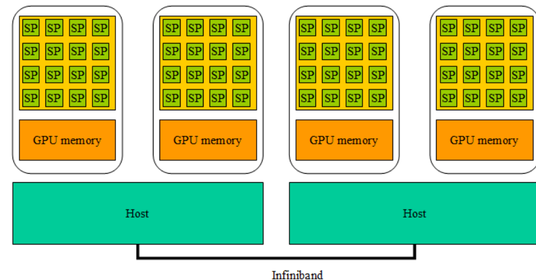

Fig. 1. GPU Cluster system

There are multiple level of parallelism in a GPU Cluster System; inner GPU level, intra GPU level.

Inner GPU parallelism is based on Single Program Multi Datastream (SPMD) model. A GPU has hundreds of processors called stream processors (SPs). In SPMD model, all SPs execute the same program but for different data elements. 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 [10]. CUDA maps parallelizable code fragment to these GPU threads.

Intra GPU level parallelism in multi GPU subsystems is based on distributed memory parallel model. Each GPU is allowed to access only to the dedicated GPU memory. Therefore, it is required to divide the computation domain into subsets that fit to GPU memory and loaded before GPU computation. GPU-GPU communication and synchronization control are required for this level parallelism.

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).

978-1-4577-1787-1/12/$26.00 ©2012 IEEE

B. Existing C Code to CUDA Code

In FDTD method, electric and magnetic fields are discretized with regular grid and allocated as three dimensional arrays. Field values are updated with parallelizable for-loop. A typical example of the field update for-loop shows in Fig.2. (Not actual code) EX is an array for x-element of electric field. HY and HZ are y and z element of magnetic fields, respectively. The for-loop is converted to a pair of CUDA kernel code (lower) and kernel call (upper). (Fig.3) CUDA generates thousands of GPU threads that simultaneously execute kernel program on SPs.

```
for( int k = 0; k < NZ; k ++)
    for( int j = 0; j < NY; j ++)
        for( int i = 0; i < NX; i ++){
             int kji = NX*NY*k+NX*j+i ;
           EX[kji] = c0*EX[kji]
                      +c1*(HZ[kji] - HZ[kji-NX])
                      -c2*(HY[kji] - HY[kji-NX*NY])
}
```

Fig. 2. A typical field computation for-loop

```
// kernel call
exKernel<<< grid, threads >>>(  dEX, dHY, AHZ) ;
// CUDA kernel program
__global__  void
exKernel(  float* dEX,  float* dHY,  float* dHZ)
{
    int ii = blockDim.x*blockIdx.x + threadIdx.x ;
    int jj = blockDim.y*blockIdx.y + threadIdx.y ;
    int kk = blockDim.z*blockIdx.z + threadIdx.z ;
    int kji = NX*NY*kk+NX*jj+ii;
    dEX[kji] = c0*dEX[kji]
             +c1*( dHZ[kji]- dHZ[kji-NX    ] )
             -c2*( dHY[kji]- dHY[kji-NX*NY ] );
};
```

Fig. 3. A pair of CUDA kernel program (lower) and kernel call (upper)

All the data used in CUDA kernel is required to transfer GPU memory before CUDA kernel call. After the computation is completed, the results may be transferred to host memory. Process flow of three-dimensional FDTD program by GPU follows:

1. Initialize GPU
2. Allocate GPU memory and copy initial data to GPU
3. Time step for-loop{
   execute electric field computation on GPU
   execute magnetic field computation on GPU
   }
4. Transfer GPU data to host memory
5. Release GPU memory and Finish

C. Implementation of GPU-GPU Communication on GPU Cluster

As FDTD is based on finite difference schema, GPU computes an array element by referring adjacent array element. When a GPU updates an element at the boundary of the decomposed data array, it is going to refer to the adjacent data elements that reside another GPU subsystem. GPU does not access another GPU memory but copies of referenced data elements are allocated. These reference data elements are not updated by GPU. Therefore to keep coherency between the original and copied data elements, the updated values of the original elements are copied from another GPU memory and replace the value of copy elements after the original elements are successfully updated.

Figure 4 shows an example of referenced data update 1D data decomposition (see also Fig.2). k-th GPU computes up to index "n" of EX (x component of electric field array) and HY (y component of magnetic field array), and k+1 th GPU computes greater than index "n" of E and H (elements of E and H are depicted by green small rectangles). From finite difference schema of FDTD, k-th GPU computes EX[n], by referring HY[n], and HY[n+1]. Unfortunately HY[n+1] resides on (k+1)-th GPU memory where k-th GPU cannot access directly. Similarly, (k+1)-th GPU computes HY[n+1] by referring EX[n] and EX[n+1], where EX[n] resides on k-th GPU memory, where (k+1)-th GPU cannot access to.

These reference data elements are copied on GPU that requires these elements for computation (depicted in yellow small rectangles). k-th GPU just refers the copy of HY[n+1] and does not updates its value. To keep the original HY[n+1] on (k+1)-th GPU and the copied element on k-th GPU coherent, the newly updated original value are transferred from (k+1)-th GPU to k-th GPU .

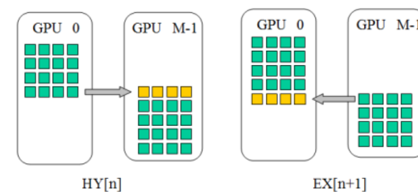

Fig. 4. Coherency between original and reference copy data

Figure 5 shows synchronization and GPU-GPU communication flow inside the FDTD time step. Each rectangle represents a task of computation or control. There are N cluster nodes, and each cluster node is equipped with M GPUs. At the end of electric field computation, all the GPUs with in a cluster node wait for completion of other GPU's computation. This type of synchronization is achieved by "barrier synchronization". Within a cluster node, OpenMP barrier synchronization function is used in combination with cudaThreadSync. Barrier synchronization among all the GPUs is implicitly achieved by MPI (Message Passing Interface) send and receive functionality. At this point, all the (N*M) GPUs in this cluster complete the electric field computation, then "GPU-GPU communication" that updates the reference-copy data elements with transfer the newly updated original data values starts. At the end of GPU-GPU communication, all the GPUs waits for data transfer is completed. After all the GPU-GPU communication is finished, GPUs start to compute magnetic field with magnetic field computation. At the end point of magnetic field computation, synchronization is taken as the end of electric field computation.

In order to convert the code to GPU cluster, implementing GPU-GPU communication and synchronization is required. Within a cluster node, OpenMP and CUDA function is used to GPU-GPU communication,

whereas communication between cluster nodes is implemented with MPI.

Field data arrays are decomposed along Z-direction, which is 1D decomposition. Assume there are N cluster nodes with M GPUs, and there exists totally M*N GPUs. Each GPU is identified with the cluster node id, k, and GPU ID in k-th cluster node. In MPI environment, the cluster node id is equivalent to MPI rank. Then domain is partitioned into M*N subsets. We relate (k,l) th GPU to k*M+l th subset of the domain. Each GPU communicates just two contiguous indexed GPUs. If GPU ID "l" is neither 0 nor M-1, then GPU communicates with GPUs within the same rank. In such case, GPU-GPU communication is achieved by combination of two cudaMemcpy, DeviceToHost and HostToDevice. If GPU ID is equal to 0 or M-1, it communicates with GPU resides in adjacent MPI rank node. In this case, MPI_Send and MPI_Recv are used in addition to cudaMemcpy.

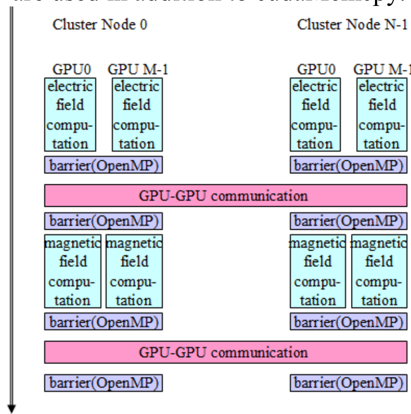

Fig. 5. GPU Cluster task flow

Figure 6 illustrates GPU-GPU communication in pseudo code. This code is executed in all OpenMP threads on all cluster nodes.

```
// MPI rank = k
#pragma omp barrier
cudaMemcpy(boundarydata,DeviceToHost);
#pragma omp barrier
#pragma omp master
// if GPU id l==0 or M-1
     MPI_Send(boundarydata, rank=k-1,k+1 );
     MPI_Recv(boundarydata, rank=k+1,k-1 );
#pragma omp barrier
cudaMemcpy(boundarydata,HostToDevice);
#pragma omp barrier
```

Fig. 6. Pseudo code for GPU-GPU communication

Assume the current thread is related to (k,l)th GPU, that is l-th OpenMP thread with MPI rank k. The array elements on the boundary of the decomposed array are copied back to host memory with cudaMemcpy function. All the GPU wait for completion of these data transfer by "#omp pragma barrier" OpenMP barrier synchronization functionality. If this OpenMP thread id , that is GPU ID, is equal to 0 or M-1, the boundary data should be send to (k+1) or (k-1) th MPI rank with MPI_Send function and the boundary data should be received from MPI rank (k-1) or (k+1). Finally, updated boundary values are transferred to GPU. MPI_Send and MPI_Recv are synchronized function that wait for completion of the process. This means that all GPUs with GPU ID is 0 or M-1 wait for completion of MPI communication. Therefore, MPI send-receive pair in combination with OpenMP barrier functionality realizes barrier synchronization for all GPUs.

III. GPU Cluster Environment

Figure 7 shows our multi-GPU cluster system and Table 1 shows the specification of the system. The GPU cluster system consists of three nodes. The seven GPU boards (NVIDIA Tesla C2070) are mounted on each node. The each GPU board has 6 GB GDDR5 RAM. GPU memory of 42 GB per node is available. Therefore, the memory of 126 GB (42GB × 3 nodes) is available in the cluster system.

Fig. 7. Multi-GPU cluster system

TABLE I. Specifications

CPU	[Intel Xeon X5677 (3.46 GHz, 12MB) × 2] × 3 nodes
Main memory	96 GB (DDR3 1066 MHz Reg.ECC 16 GB × 6) × 3 nodes
GPU	[NVIDIA Tesla C2070 × 7] × 3 nodes
CUDA cores (GPU)	[448 core × 7] × 3 nodes
GPU memory	[6 GB × 7 = 42 GB] × 3 nodes = 126 GB
Memory Interface (GPU)	GDDR5 SDRAM 384 bit
Memory Bandwidth (GPU)	144 GB/sec
Internode communication	InfiniBand QRD
Operating System	Linux (CentOS 5.5)
CUDA version	CUDA 4.0

IV. Performance Tests

A. *Performance of multi-GPU cluster system*

To examine the performance of the FDTD calculation on multi-GPU cluster system, we conducted a test using the calculation model shown in Fig. 8. The model was a cube domain, and its center was allocated a sphere (25 cells in radius). The incident wave was an assumed plane wave. The calculation domain was 600 × 600 × 600 cell size.

Figure 9 shows the calculation time of our multi-GPU cluster system. Computation time is calculated for part of the electromagnetic field update. The thread block size is 32 × 16. Calculation speed with two GPUs in one node was approximately 2.3 times faster than with seven GPUs in the same node. In the cluster system, the communication times between cluster nodes that do not occur in one node can significantly affect the overall time of three-dimensional

FDTD calculation. However, the speeds with three nodes (21 GPUs) were approximately 1.4 times faster than with one node (7 GPUs).

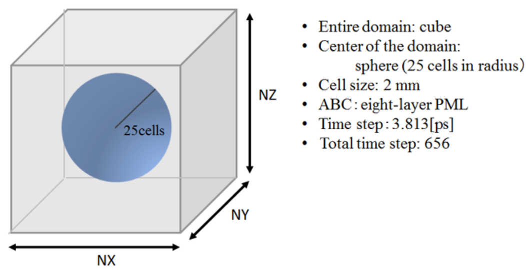

Fig. 8. Calculation model for performance test

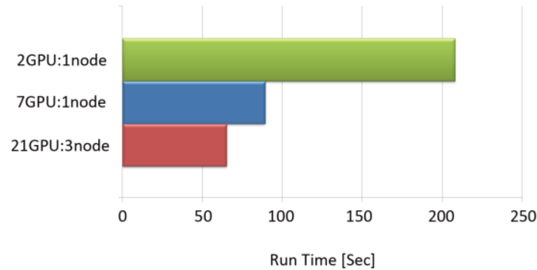

Fig. 9. Computational performance of multi-GPU cluster system

B. 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 cluster system and a supercomputer in the case of a plane wave incidence for a numerical human model. In this performance test, we used an anatomical model of three-year-old child developed by Nagaoka *et al* [11]. Figure 10 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 two nodes (14 GPUs) of GPU cluster system and one node (8 CPUs) of the supercomputer were 4354 sec and 6618 sec, respectively. The thread block size for the GPU calculation was 32 × 16 in this test. Run time on the GPU cluster system was approx. 1.5 times faster than that using the vector supercomputer. We found that multi-GPU cluster system is able to significantly accelerate three-dimensional FDTD calculation with positive implications for practical application.

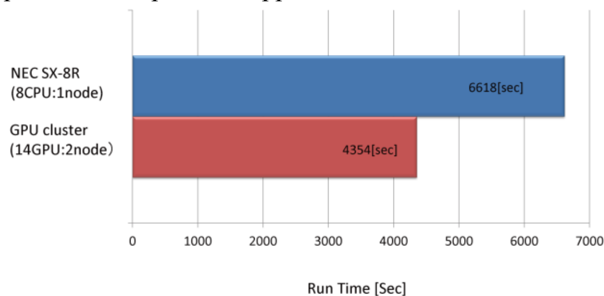

Fig. 10. Comparison run time of FDTD calculation between multi-GPU cluster and supercomputer

V. CONCLUSION

We implemented the three-dimensional FDTD method on multi-GPU cluster environment with CUDA and MPI. In this study, we used the NVIDIA Tesla C2070 as GPGPU boards and examined the performance of the FDTD calculation on multi-GPU cluster environment. We confirmed that the FDTD calculation on the multi-GPU clusters is faster than that on a multi-GPU (a single workstation), and we also found that the GPU cluster system calculate faster than a vector supercomputer. In addition, our GPU cluster system allowed us to perform the large-scale FDTD calculation because were able to use GPU memory of over 100 GB.

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

[4] 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.

[5] 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.

[6] 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.

[7] NVIDIA, NVIDIA *CUDA programming guide version* 4.0, NVIDIA Corporation, 2011.

[8] 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.

[9] T. Nagaoka and S. Watanabe, "Multi-GPU accelerated three-dimensional FDTD method for electromagnetic simulation," In *Proceedings of IEEE EMBC 2011*, pp. 401-404, Boston, MA, 2011.

[10] 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.

[11] T. Nagaoka, E. Kunieda, and S. Watanabe, "Proportion corrected scaled voxel models for Japanese children and their application to the numerical dosimetry of specific absorption rate for frequencies from 30 MHz to 3 GHz," *Phys. Med. Biol.*, vol. 53, pp. 6695-6711, Nov. 2008.