Improved Performance of FDTD Computation Using a Thread Block Constructed as a Two-Dimensional Array with CUDA

Naoki Takada[1], Tomoyoshi Shimobaba[2], Nobuyuki Masuda[2], and Tomoyoshi Ito[2]

[1]Department of Informatics and Media Technology,
Sony Institute of Higher Education, Shohoku College,
428 Nurumizu, Atsugi, Kanagawa 243-8501, JAPAN
ntakada@shohoku.ac.jp

[2]Graduate School of Engineering, Chiba University,
1-33 Yayoi-cho, Inage-ku, Chiba, Chiba 263-8522, JAPAN
shimobaba@faculty.chiba-u.jp, masdanb@faculty.chiba-u.jp, itot@faculty.chiba-u.jp

Abstract ― In a previous study, the authors proposed an finite-difference time-domain (FDTD) implementation for a compute unified device architecture (CUDA) compatible graphics processing unit (GPU) using a thread block constructed as a two-dimensional (2-D) array. However, it was found that the larger the computational domain of the 2-D FDTD simulation using the GPU, the slower the computational speed.

In the present paper, the authors investigated the computational performance with respect to the size of a thread block constructed as a 2-D array, and improved the performance of the implementation. Finally, regardless of the size of computational domain, the computational speed using a single GPU (NVIDIA GeForce GTX 280) achieved approximately 30.0 Gflops, which was approximately 20 times faster than that of a single core of a central processing unit (Intel 3.0-GHz Core 2 Duo). The improved performance was approximately 65% of the theoretical peak performance (47.23 Gflops) obtained by the theoretical memory bandwidth (141.7 GB/s).

Index Terms ― Finite-difference time-domain method, GPU computing, graphics processing unit, high-performance computing.

I. INTRODUCTION

A graphics processing unit (GPU) is equipped with a large-memory graphics accelerator board for use in a personal computer (PC). The GPU has many processors for 32-bit floating-point calculations. The theoretical peak performance of recent GPUs is greater than 1 Tflops (floating point operations per second). High performance/cost has been reported for the hierarchical N-body simulation using a PC cluster equipped with 256 GPUs [1].

Programs can be developed that allow GPUs to perform general numerical calculations using a high-level shader language (HLSL) (Microsoft HLSL, NVIDIA Cg [2], etc.) or a programming environment (Brook [3], the NVIDIA compute unified device architecture (CUDA) programming environment [4], etc.). Implementation of the finite-difference time-domain (FDTD) method [5-7] on a GPU using various programming environments has been reported [8-16]. The development of the GPU code written in HLSL requires technical knowledge of computer graphics (CG) [8]. In the FDTD simulation using the GPU code written in NVIDIA Cg, the Euclidean normalized error increased monotonously with respect to the time step [9]. The GPU-FDTD code written in Brook has also been reported [10]. In three-dimensional (3-D) FDTD simulation, 3-D to 2-D translation has been reported [11]. Translation from 3-D to 2-D becomes very complicated because the 3-D computational domain of the FDTD simulation is allocated to 2-D texture as a CG technique. Programming tools for the GPU based on CUDA

have been available since 2007. In CUDA, the programmer does not need to be conscious of the CG technique. The advantages of CUDA over Cg and HLSL is that CUDA allows source code to be written in a C-like language and the memory on the GPU board can be used easily. The GPU implementation for LU decomposition solvers using CUDA for computational electromagnetics application has been reported [17]. More GPU implementations of the FDTD method using CUDA have been provided and the computational performances of these implementations have been discussed [12-16]. The FDTD computation using GPU has been implemented with data reuse of the electromagnetic field, and the computational performance has been reported [12]. In [13], a thread block [4] was constructed as a 2-D array. The performance of the 2-D FDTD implementation using GPU was investigated with respect to the four arrays considered (4 × 4, 8 × 8, 12 × 12, and 16 × 16). The FDTD computation was fastest for the 16 × 16 array among the four arrays. The GPU implementations of 3-D FDTD computation using CUDA have been reported [14, 15]. A thread block was constructed as a 2-D array [14], and the size of the 2-D array was 16 × 16. However, the performance of the implementation was not investigated the other 2-D arrays. In [15], a thread block was constructed as a 1-D array, and GPU implementations based on two thread-to-cell mapping algorithms were considered. The performances of the implementations were investigated with respect to the number of threads per thread block. Thus, a 1-D array or a 2-D array is used as a thread block.

In a previous study, the authors proposed a GPU implementation for FDTD computation using a thread block constructed as a 2-D array [16]. The computational domain of the FDTD simulation is divided into subdomains. The electromagnetic field data of a subdomain is stored in shared memory [4]. A subdomain is adjacent to four neighbor subdomains. In this case, a subdomain requires four overlapping areas that include the electromagnetic field data of four neighbor subdomains required to calculate the electromagnetic field on the boundaries of a subdomain. The proposed implementation uses two different subdomains for the calculation of the electric field and the magnetic field, and reduces the number of overlapping areas from four to two in order to reduce the number of branches in the CUDA program. In performance evaluation of the proposed implementation, NVIDIA GeForce GTX 280 was used as a GPU, and a 16 × 16 2-D array was used as a thread block. However, the larger the computational domain of the 2-D FDTD simulation using the proposed implementation, the slower the computational speed.

In the present paper, the authors investigated the performance of the proposed GPU implementation with respect to the size of a thread block constructed as a 2-D array and improved the performance of the proposed implementation. As a result, the computational speed of the implementation in a computational domain of 8,192 × 8,192 peaked when the size of the thread block was 32 × 4. Regardless of the size of the computational domain, the computational speed using a single GPU (NVIDIA GeForce GTX 280) was approximately 30.0 Gflops, which is approximately 20 times faster than that of a single core of a central processing unit (CPU) (Intel 3.0-GHz Core 2 Duo), where the Intel C compiler was used as C compiler.

The remainder of the present paper is organized as follows. The proposed GPU implementation for FDTD computation using a thread block constructed as a 2-D array in a previous study is described in Section II. In Section III, the performance of the proposed GPU-FDTD implementation is described in detail with respect to the size of a thread block constructed as a 2-D array, and the performance of the proposed implementation is improved. Finally, in Section IV, conclusions are presented and future research is described.

II. GPU-FDTD IMPLEMENTATION [16]

CUDA is a parallel computing architecture. NVIDIA GeForce GTX 280 has 30 streaming multiprocessors (SMs), each of which is composed of eight streaming processors (SPs) for 32-bit floating-point calculation, 16,384 registers, and 16 KB of on-chip memory. The CUDA program consists of the CPU code and the GPU code. The GPU code, which is written in a C-like language, includes data-parallel functions, referred to collectively as the *kernel*. A kernel is executed as a *grid* of *thread blocks*. A thread block is an array of threads that can cooperate. Threads within the same thread block are synchronized and share data

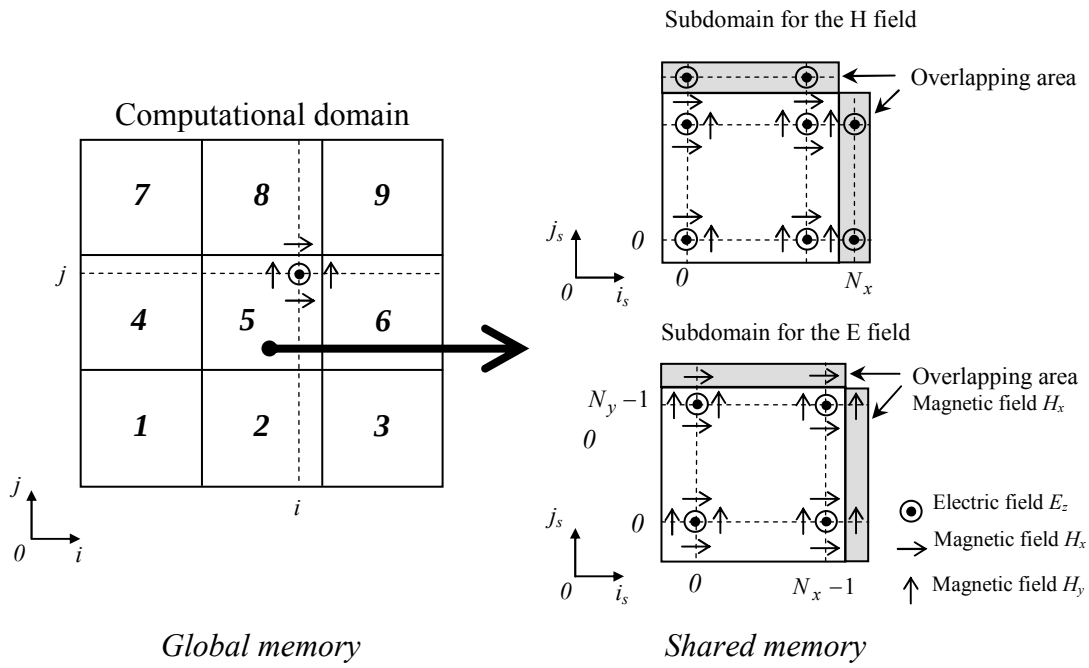

Fig. 1. Subdomains of the proposed GPU-FDTD implementation (TM case).

in the shared memory. The CPU code is written in the C language, and the CPU launches the GPU kernel.

In the case of the 2-D FDTD method, the equations in the transverse magnetic (TM) case are as follows:

$$H_x^{n+1/2}(i, j+1/2) = H_x^{n-1/2}(i, j+1/2) - \Delta t / \mu \Delta y \left\{ E_z^n(i, j+1) - E_z^n(i, j) \right\}, \quad (1)$$

$$H_y^{n+1/2}(i+1/2, j) = H_y^{n-1/2}(i+1/2, j) - \Delta t / \mu \Delta x \left\{ E_z^n(i+1, j) - E_z^n(i, j) \right\}, \quad (2)$$

$$E_z^{n+1}(i, j) = E_z^n(i, j) - \Delta t / \varepsilon \Delta y \left\{ H_x^{n+1/2}(i, j+1/2) - H_x^{n+1/2}(i, j-1/2) \right\} + \Delta t / \varepsilon \Delta x \left\{ H_y^{n+1/2}(i+1/2, j) - H_y^{n+1/2}(i-1/2, j) \right\}, \quad (3)$$

where $E_z^{n+1}(i, j)$ is the required value of the electric field at grid point (i, j) and the (n+1)-th time step, Δx and Δy are the sizes of the spatial division in the x and y directions, respectively, and Δt is the time increment. Parameters ε and μ are the electric permittivity and the magnetic permeability in the medium, respectively. A large quantity of electromagnetic field data in the computational domain for FDTD simulation is stored in the global memory as off-chip device memory on a CUDA-compatible graphics accelerator board. The CPU allocates the data of the electromagnetic fields to a global memory on the GPU board. The memory size of each electromagnetic field array in the program must be an integer multiple of 16 for coalesced global memory access [4]. If the memory size of each required electromagnetic field array in the computational domain is not an integer multiple of 16, the memory size, which is larger than that of each required array, is allocated in order to be equal to an integer multiple of 16. Shared memory enables faster data access than global memory and accounts for 16 KB of on-chip memory in the role of CPU cache memory. In the proposed implementation, a thread block is constructed as a 2-D array, and the computational domain of the FDTD simulation is divided into a small subdomain. The electromagnetic field data in each subdomain are stored in each shared memory as shown Fig. 1. Calculating the electric field data $E_z^{n+1}(i, j)$ in Region 5 requires the magnetic field data $H_y^{n+1/2}(i+1/2, j)$ in Region 6 and the magnetic field data $H_x^{n+1/2}(i, j+1/2)$ in Region 8. The required data of the magnetic field, which overlap neighboring subdomains as shown in Fig. 1, are also stored in each shared memory. Each subdomain of the proposed implementation includes two overlapping areas. In the CUDA

```
__global__ void maxwH( float *HX, float *HY, float *EZ)
{
  int tx=threadIdx.x;
  int ty=threadIdx.y;
  unsigned int x = blockIdx.x*blockDim.x + threadIdx.x;
  unsigned int y = blockIdx.y*blockDim.y + threadIdx.y;
  __shared__ float SH_EZ[NY+1][NX+1];
  float xx;
  if(ty==(NY-1))SH_EZ[ty+1][tx]=EZ[(y+1)*w+x];
  if(tx==(NX-1))SH_EZ[ty][tx+1]=EZ[y*w+x+1];
  SH_EZ[ty][tx]=EZ[y*w+x];
  xx=SH_EZ[ty][tx];
  HX[(y+1)*w+x]=HX[(y+1)*w+x]-dtdy*(SH_EZ[ty+1][tx]-xx);
  HY[y*w+x+1]=HY[y*w+x+1]+dtdx*(SH_EZ[ty][tx+1]-xx);
}
```

(a)

```
__global__ void maxwEZ( float *HX, float *HY, float *EZ, float *T)
{
    int tx=threadIdx.x;
    int ty=threadIdx.y;
    unsigned int x = blockIdx.x*blockDim.x + threadIdx.x;
    unsigned int y = blockIdx.y*blockDim.y + threadIdx.y;
    __shared__ float SH_HX[NY+1][NX+1];
    __shared__ float SH_HY[NY+1][NX+1];
    if(ty==(NY-1))SH_HX[ty+1][tx]=HX[(y+1)*w+x];
    if(tx==(NX-1))SH_HY[ty][tx+1]=HY[y*w+x+1];
    SH_HX[ty][tx]=HX[y*w+x];
    SH_HY[ty][tx]=HY[y*w+x];
    __syncthreads();
   if((x==511)&&(y==511)){
       T[0]+=dt;
      EZ[y*w+x]=am*sin(omega*T[0]);
   }else{
      EZ[y*w+x]=EZ[(y)*w+x]+dtdx*(SH_HY[ty][tx+1]-SH_HY[ty][tx])
                            -dtdy*(SH_HX[ty+1][tx]-SH_HX[ty][tx]);
   }
    if(x==0)EZ[(y)*w+x]=0;
    if(x==1023)EZ[(y)*w+x]=0;
    if(y==0)EZ[(y)*w+x]=0;
    if(y==1023)EZ[(y)*w+x]=0;
}
```

(b)

Fig. 2. GPU-FDTD code, (a) kernel for calculating the magnetic field, (b) kernel for calculating the electric field.

program, the size of each subdomain, excluding overlapping areas, is $N_x \times N_y$ when the size of the thread block constructed as a 2-D array is $N_x \times N_y$. In calculating the magnetic field data H_x and H_y (Equations (1) and (2)), all of the threads in each $N_x \times N_y$ thread block first store the data of electric field E_z of each subdomain in the shared memory, whereas no data of magnetic fields H_x and H_y of each subdomain are stored in the shared memory. Next, the same electric field data $E_z^n(i,j)$ required in Equations (1) and (2) are stored in the register only once [12]. After these procedures, all of the threads in a thread block are used to calculate Equations (1) and (2) in each subdomain. Finally, the calculated data of magnetic fields H_x and H_y are stored in the global memory, while the calculation of electric field E_z by Equation (3) at the following time step is performed in the same manner. The subdomain used to calculate the electric field E_z (Subdomain for the E field shown in Fig. 1) differs from that used to calculate the magnetic fields H_x and H_y (Subdomain for the H field shown in Fig. 1) in order to use the shared memory efficiently. Therefore, two kernels for the electric field and magnetic field calculations are required in the CUDA program. The number of time steps is counted and stored in the global memory by a particular SP in each kernel if the calculation of the electric field E_z or the magnetic field H_x or H_y requires the number of time steps for the boundary condition. The kernel codes of the proposed GPU-FDTD implementation are shown in Fig. 2.

III. PERFORMANCE

In the present paper, the authors used the NVIDIA CUDA programming environment for the GPU and a NVIDIA GeForce GTX 280 as the GPU board and timed the calculations required for a simple 2-D model, excluding for the absorbing boundaries, in order to investigate the basic performance of the proposed GPU-FDTD implementation. The propagation of electromagnetic waves from the line source in the TM case was used as the calculation model. The line source was located in the center of the 2-D computational domain. The authors compared the GPU implementation with the conventional CPU implementation. In the GPU implementation, the authors developed a GPU-FDTD code written in the C language and a kernel written in a C-like language for the instruction set of the GPU using the CUDA programming environment. A kernel can be embedded in the code written in the C language for the CPU. Two kernels in the FDTD code were used: a kernel to calculate the magnetic fields H_x and H_y and a kernel to calculate the electric field E_z. A CUDA driver (180.22) was used. The GPU-FDTD code was compiled using NVIDIA CUDA 2.1. In the CPU implementation, the conventional FDTD code was written in the C language. Here, FDTD computation was performed using a single core in the CPU. The code for the CPU was compiled using the Intel C-compiler (ver. 11.1) with "-msse –O3" as an optimized compiler option. The CPU-only computation used SSE instructions. In the CPU and GPU implementations, the authors used the

same PC equipped with an Intel Core 2 Duo E8400 (3.0 GHz) as the CPU, 2.0 GB of memory (DDR3-1333), and Fedora 9 as the Linux operating system and timed 1,000 iterations of the calculation by Equations (1) through (3) for the GPU and CPU-only computation.

In Equations (1) through (3), the authors replace $\Delta t/\mu\Delta x$, $\Delta t/\varepsilon\Delta x$, and $\Delta t/\varepsilon\Delta y$ with constants. As a result, the number of operations in Equations (1) through (3) is estimated to be 12. The theoretical peak performance of the FDTD simulation using the NVIDIA GeForce GTX 280 as a GPU is obtained as 47.23 Gflops = 141.7 GB/s ÷ 4 byte/word × 12 operations ÷ nine words, where the theoretical memory bandwidth is 141.7 GB/s, and the number of load/store data in Equations (1) through (3) is estimated to be nine words. On the other hand, the theoretical peak performance of the GPU is obtained as 933.12 Gflops = three operations/SP × 240 SP × 1.296 GHz, and the theoretical peak performance of the FDTD simulation using the GPU (47.23 Gflops) is smaller than in the latter example (933.12 Gflops). Therefore, the bottleneck of 2-D FDTD computation using the GPU is the memory bandwidth.

Here, T_{GPU} is the GPU computation time (s) in the computational domain of $L \times L$, and N_{itr} is the number of time steps of the FDTD simulation. Subsequently, the actual computational speed (flops) can be obtained as 12 operations $\times L \times L \times N_{itr}/T_{GPU}$ (N_{itr} = 1,000). When the size of the subdomain is 16 × 16, the two computational speeds of the GPU-FDTD simulation using shared and non-shared memory are shown in Fig. 3. In Fig. 3, the 'non-shared memory' indicates the basic GPU-FDTD computation without the subdomain for using shared memory, while 'shared memory' indicates the proposed GPU-FDTD implementation using shared memory. The larger the computational domain of the 2-D GPU-FDTD simulation, the slower the computational speed. In the computational domain of 8,192 × 8,192, the authors investigated the performances of two GPU-FDTD implementations with respect to the size of a thread block constructed as a 2-D array (Table 1). In 'shared memory', the thread block of size 32 × 4 achieved a peak speed (Table 1(a)). In the 'non-shared memory', the thread block of size 64 × 4 achieved a peak speed (Table 1(b)). In Table 1, the computation time using the

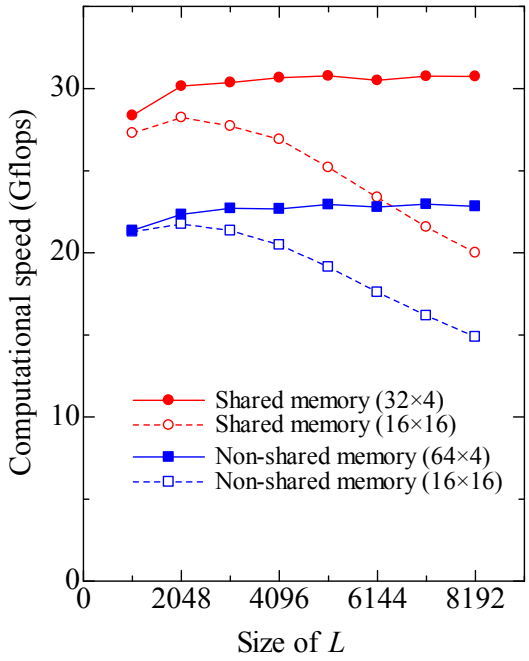

Fig. 3. Computation speed versus computational domain $L \times L$.

proposed GPU-FDTD implementation was very long when N_y was larger than or equal to N_x. The authors analyzed the performance of the proposed GPU-FDTD implementation using the NVIDIA CUDA Visual Profiler. Bank conflicts of shared memory occurred when $N_x \leq 8$ for all cases of the total number of threads per thread block considered herein. Therefore, the performance of the global memory overall throughputs, which is the sum of the global memory write throughput and the global memory read throughput, decreased markedly in the GPU computation of the electric field and the magnetic field. For all cases of the total number of threads per thread block, the number of divergent branches within a warp increased in the GPU computation of the electric field when $N_x \leq 16$. In the 'shared memory', the performance of the global memory overall throughputs was the best for the case in which the total number of threads per thread block is 128. In Table 1(a), a thread block of size 32 × 4 achieved a peak speed. In the case of the 32 × 4 thread block, the global memory overall throughput of the GPU computation of the electric field was 99.12 GB/s, while the global memory overall throughput of the GPU computation of the magnetic field was 126.70 GB/s. The

Table 1: Computation time for the GPU-FDTD implementations with respect to the size of the thread block constructed as a 2D-array ($N_x \times N_y$) in the computational domain: 8,192 × 8,192. (a) GPU-FDTD implementation with shared memory, (b) basic GPU-FDTD implementation without a subdomain for shared memory.

(a)

Total number of threads per thread block							
512		256		128		64	
$N_x \times N_y$	Time (ms)	$N_x \times N_y$	Time (ms)	$N_x \times N_y$	Time (ms)	$N_x \times N_y$	Time (ms)
512 × 1	34,910.91	256 × 1	30,332.19	128 × 1	29,208.40	64 × 1	31,378.07
256 × 2	27,860.34	128 × 2	26,828.33	64 × 2	26,546.84	32 × 2	31,021.64
128 × 4	27,023.15	64 × 4	26,404.10	**32 × 4**	**26,193.54**	16 × 4	32,172.59
64 × 8	28,329.08	32 × 8	27,766.32	16 × 8	31,641.55	8 × 8	49,528.76
32 × 16	31,408.21	16 × 16	40,315.71	8 × 16	81,558.07	4 × 16	151,044.05
16 × 32	43,464.29	8 × 32	92,666.19	4 × 32	179,097.02	2 × 32	334,621.00
8 × 64	99,550.48	4 × 64	201,960.70	2 × 64	462,255.13	1 × 64	1,156,390.13
4 × 128	240,642.34	2 × 128	513,940.38	1 ×128	1,135,948.50		
2 × 256	531,257.31	1 × 256	1,137,908.38				
1 × 512	1,142,198.88						

(b)

Total number of threads per thread block							
512		256		128		64	
$N_x \times N_y$	Time (ms)	$N_x \times N_y$	Time (ms)	$N_x \times N_y$	Time (ms)	$N_x \times N_y$	Time (ms)
512 × 1	36,148.74	256 × 1	35,712.13	128 × 1	35,765.40	64 × 1	36,315.55
256 × 2	35,523.61	128 × 2	35,412.05	64 × 2	35,463.31	32 × 2	35,510.76
128 × 4	35,283.57	**64 × 4**	**35,283.14**	32 × 4	35,446.44	16 × 4	36,006.95
64 × 8	36,126.11	32 × 8	36,836.52	16 × 8	41,499.72	8 × 8	72,708.84
32 × 16	41,103.40	16 × 16	54,172.49	8 × 16	103,208.16	4 × 16	188,353.81
16 × 32	60,904.04	8 × 32	117,666.61	4 × 32	219,635.84	2 × 32	434,315.56
8 × 64	133,583.14	4 × 64	259,646.30	2 × 64	571,017.06	1 × 64	1,445,463.25
4 × 128	299,888.72	2 × 128	632,672.88	1 × 128	1,410,071.50		
2 × 256	661,946.06	1 × 256	1,422,834.38				
1 × 512	1,442,981.13						

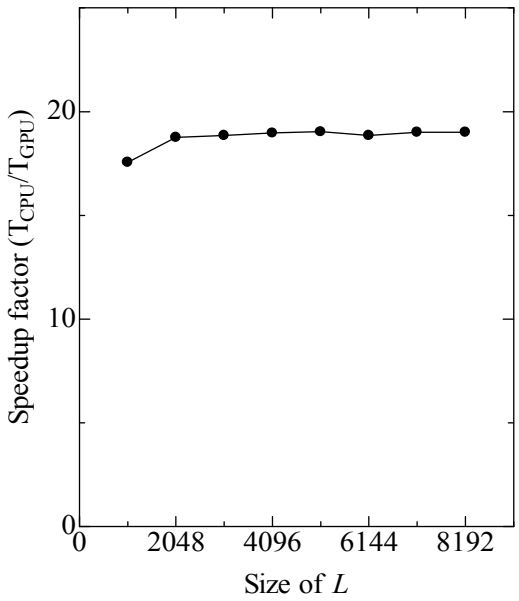

Fig. 4. Speedup factor (T_{CPU} / T_{GPU}) versus computational domain $L \times L$.

computational speeds using the 32 × 4 thread block in the 'shared memory' and the 64 × 4 thread block in the 'non-shared memory' are shown in Fig. 3. Regardless of the size of the computational domain, the computational speed of the proposed GPU-FDTD implementation achieved approximately 30.0 Gflops. The authors improved the performance of the GPU-FDTD simulation by using the optimum size of a thread block constructed as a 2-D array and compared the computation time of the GPU-FDTD simulation with that of CPU-only simulation. In Fig. 4, the speedup factor shows the ratio of the computation time of the CPU only (T_{CPU}) to that of GPU (T_{GPU}). The FDTD simulation using a single GPU was approximately 20 times faster than that using a single CPU core.

The authors estimated the effective performance (Fig. 5). As a result, the effective performance achieved approximately 65% of the

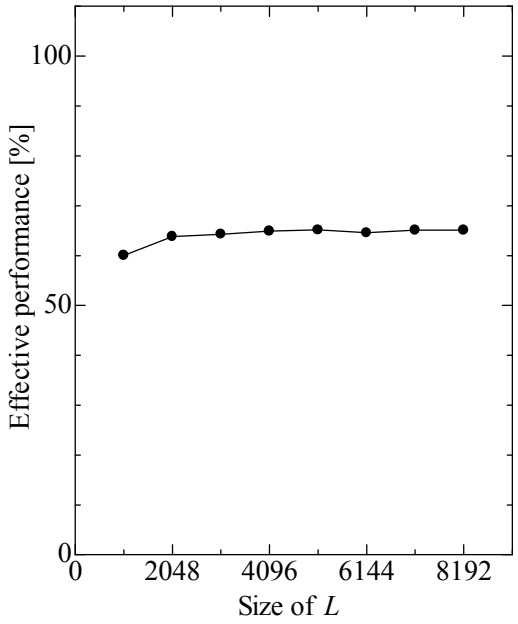

Fig. 5. Effective performance of the improved GPU-FDTD computation versus computational domain $L \times L$.

theoretical peak performance (47.23 Gflops) obtained using the theoretical memory bandwidth.

IV. Conclusion

The authors proposed GPU-FDTD implementation using a thread block constructed as a 2-D array in a previous study. However, in a 16 × 16 thread block, the larger the computational domain of 2-D GPU-FDTD simulation, the slower the computational speed. In the present paper, the authors investigated the computational performance with respect to the size of a thread block constructed as a 2-D array. As a result, the computational speed of the GPU-FDTD simulation peaked when the thread block size was 32 × 4. Regardless of the size of the computational domain, the computational speed of the GPU (NVIDIA GeForce GTX 280) was approximately 30.0 Gflops, which is approximately 20 times faster than that using a single core of the central processing unit (Intel 3.0-GHz Core 2 Duo). Finally, after improving the performance of the proposed GPU-FDTD implementation, the effective performance was approximately 65% of the theoretical peak performance of GPU-FDTD computation using an NVIDIA GeForce GTX 280 as a GPU.

In the future, the authors intend to apply the proposed method to 3-D FDTD simulation.

ACKNOWLEDGMENT

The present study was supported in part by a Grant-in-Aid for Young Scientists (B), 22700060, from the Ministry of Education, Culture, Sports, Science and Technology, Japan.

REFERENCES

[1] T. Hamada, T. Narumi, R. Yokota, K. Yasuoka, K. Nitadori, and M. Taiji, "42 TFlops hierarchical *N*-body simulations on GPUs with applications in both astrophysics and turbulence," *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, 2009.

[2] W. R. Mark, R. S. Glanville, K. Akeley, and M. J. Kilgard, "Cg: A system for programming graphics hardware in a C-like language," *ACM SIGGRAPH*, pp. 896-907, 2003.

[3] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, "Brook for GPUs: stream computing on graphics hardware," *ACM SIGGRAPH*, pp. 777-786, 2004.

[4] NVIDIA, *NVIDIA CUDA Programming Guide version 2.1*, NVIDIA, 2008.

[5] K. S. Yee, "Numerical solution of initial boundary value problems involving Maxwell´s Equations in isotropic media," *IEEE Trans. Antennas Propagat*., vol. AP-14, pp. 302-307, 1966.

[6] A. Taflove, "Computational electrodynamics: the finite difference time domain method," *Artech House*, Inc., 1995.

[7] K. S. Kunz and R. J. Luebbers, "The finite difference time domain method for electromagnetics," *CRC Press*, Inc., 1993.

[8] N. Takada, N. Masuda, T. Tanaka, Y. Abe, and T. Ito, "A GPU implementation of the 2-D finite-difference time-domain code using high level shader Language," *ACES Journal*, vol. 23, no. 4, pp. 309-316, 2008.

[9] G. S. Baron, C. D. Sarris, and E. Fiume, "Fast and accurate time-domain simulations with commodity graphics hardware," *Proceedings of the Antennas and Propagation Society International Symposium*, July 2005.

[10] M. J. Inman and A. Z. Elsherbeni, "Programming video cards for computational electromagnetics application," *IEEE Antennas and Propagation Magazine*, vol. 47, no. 6, pp. 71-78, 2005.

[11] M. J. Inman, A. Z. Elsherbeni, J. G. Maloney, and B. N. Baker, "Practical implementation of a CPML absorbing boundary for GPU accelerated FDTD technique," *ACES Journal*, vol. 23, no. 1, pp. 16-22, 2008.

[12] N. Takada, T. Takizawa, Z. Gong, N. Masuda, T. Ito, and T. Shimobaba, "Fast computation of 2-D finite-difference time-domain method using graphics processing unit with unified shader," *IEICE Trans. Inf. Syst.*, vol. J91-D, no. 10, pp. 2562-2564, 2008.

[13] S. Ryoo, C. Rodrigues, S. Baghsorkhi, S. Stone, D. Kirk, and W. Hwu, "Optimization principles and application performance evaluation of a multithreaded GPU using CUDA," *Proc. of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp.73–82, 2008.

[14] P. Sypek, A. Dziekonski, and M. Mrozowski, "How to render FDTD computations more effective using graphics accelerator," *IEEE Trans. Magn.*, vol. 45, no. 3, pp. 1324-1327, 2009.

[15] V. Demir and A. Z. Elsherbeni, "Compute Unified Device Architecture (CUDA) based finite-difference time-domain (FDTD) implementation," *ACES Journal*, vol. 25, no. 4, pp. 303-314, 2010.

[16] N. Takada, T. Shimobaba, N. Masuda, and T. Ito, "High-speed FDTD simulation algorithm for GPU with compute unified device architecture," *Proc. 2009 IEEE AP-S Int. Symposium and USNC/URSI National Radio Science Meeting*, session 126, 126.9, 2009.

[17] M. J. Inman, A. Z. Elsherbeni and C. J. Reddy, "CUDA based LU decomposition solvers for CEM applications," *ACES Journal*, vol. 25, no. 4, pp. 339-347, 2010.

Naoki Takada received a B.E. degree and an M.S. degree in Electrical Engineering from Gunma University, Gunma, Japan in 1994 and 1996, respectively and a Ph.D. in Electrical Engineering from Gunma University in 2000. From 1996 to June 2001, he was a research associate at Oyama National College of Technology, Tochigi, Japan. From July 2001 to March 2005, he was a research scientist with the High-Performance Biocomputing Research Team, Bioinformatics Group, Genomic Science Center (GSC), Institute of Physical and Chemical Research (RIKEN), Yokohama, Japan and joined the "Protein Explorer Project" for a petaflops special-purpose computer (MDGRAPE-3) system for molecular dynamics simulation of proteins. This project was the Protein 3000 project supported by the Ministry of Education, Culture, Sports, Science, and Technology of Japan. He was a lecturer from April 2005 to 2009 and an associate professor from 2010, at Shohoku College, Atsugi, Japan.

His research interests are GPGPU, distributed and parallel computation including the FDTD method, a special-purpose computer for the FDTD method, numerical simulation including the FDTD method, the CIP method, and molecular dynamics, and electromagnetic theory. He is a member of ACES and IEICE.

Tomoyoshi Shimobaba received B.E. and M.E. degrees from Gunma University in 1997 and 1999. And he received a Ph.D. from Chiba University in 2002. From 2002 to 2005, he was a special postdoctoral researcher at RIKEN. From 2005 to 2009, he was an associate professor at the Graduate School of Science and Engineering, Yamagata University. He is currently an associate professor at the Graduate School of Engineering, Chiba University.

His research interests include 3D display, digital holography, and special-purpose computing using FPGA and GPU. He is a member of OSA, IEICE, and ITE.

Nobuyuki Masuda received a bachelor degree and a master degree in System Science from the University of Tokyo, Tokyo, Japan in 1993 and 1995, respectively, and a Ph.D. in System Science from the University of Tokyo in 1998. From 2000 to March 2004, he was a research associate at Gunma University, Gunma, Japan. Since April 2004, he has been a research associate at Chiba University, Chiba, Japan.

His research interests include a special-purpose computer for digital-holographic particle-tracking velocimetry and computer-generated holograms on GPU. He is a member of IEICE, IPSJ, and ASJ.

Tomoyoshi Ito received a B.E. degree, M.S. degree, and Ph.D. from the University of Tokyo, Tokyo, Japan in 1989, 1991, and 1994, respectively. He was a research associate from 1992 to 1994 and an associate professor from 1994 to 1999 at Gunma University, Gunma, Japan. From 1999 to 2005, he was an associate professor at Chiba University, Chiba, Japan, and since 2005, he has been a professor.

His research interests are high-performance computing and its applications. He was an initial member of the GRAPE project, which has produced special-purpose computers for astrophysics. He developed the first machine, GRAPE-1, in 1989 and the second machine with high accuracy, GRAPE-2, in 1990 and the third machine for protein simulation, GRAPE-2A, in 1991. From 1992, he has also designed and built special-purpose computers for holography named HORN. Using HORN computers, he is trying to develop a three-dimensional television. He is a member of OSA and IEICE.