PAPER *Special Section on Parallel and Distributed Computing and Networking*

Implementation of a GPU-Oriented Absorbing Boundary Condition for 3D-FDTD Electromagnetic Simulation

Keisuke DOHI[†a)], **Yuichiro SHIBATA**[††], **Kiyoshi OGURI**[††], *and* **Takafumi FUJIMOTO**[††], ***Members***

SUMMARY In this paper, we propose and discuss efficient GPU implementation techniques of absorbing boundary conditions (ABCs) for a 3D finite-difference time-domain (FDTD) electromagnetic field simulation for antenna design. In view of architectural nature of GPUs, the idea of a periodic boundary condition is introduced to implementation of perfect matched layers (PMLs) as well as a transformation technique of PML equations for partial boundaries. We also present efficient implementation method of a non-uniform grid. The evaluation results with a typical simulation model reveal that our proposed technique almost double the simulation performance and eventually achieve the 55.8% of the peak memory bandwidth of a target GPU.

key words: *absorbing boundary condition, perfectly matched layer, FDTD, GPU*

1. Introduction

In this paper, we discuss optimization techniques to implement 3D Finite-Difference Time-Domain (3D-FDTD) method with Absorbing Boundary Conditions (ABC) on a GPU, for accelerating analysis of antenna characteristics.

Characteristics analysis for antenna design consists of two steps: simulation of electromagnetic propagation in time domain and analysis of simulation results. Acceleration of simulation is needed since most of the execution time is occupied by the electromagnetic simulation. The FDTD method proposed by Yee discretizes the Maxwell's equation in spatial and time domain and calculates the electric and magnetic potential at each point on spatial grids or lattices [1], [2]. Although its computational costs, which are the number of floating point operations and memory accesses in this case, and memory usage increase linearly with the size of a simulation model, the method is widely used since performance of computers have been rapidly improved and the algorithm is simple and easy to understand.

Absorbing boundary conditions (ABCs) are important when using the FDTD method for unbounded problems. Since computers can not handle an infinite space or infinite number of elements on the grid, the method can only be used within a finite space. To resolve this limitation, several types of ABCs were proposed to attenuate reflection waves from boundaries of the computational space [3]–[5]. Among them, the perfectly matched layer (PML) proposed by Bérenger is widely used [6]. However, the PML needs additional floating point operations, memory accesses, and memory resources.

The FDTD method is known as a kind of stencil computation that has a high degree of parallelism but requires a large memory bandwidth. While GPU implementation is attractive as a cost-effective acceleration approach, our earlier work has shown that GPU implementation of Absorbing Boundary Conditions (ABCs) tends to be a bottleneck of the simulation [7]. We discuss this issue in this paper. In concrete terms, the main contributions of this paper are as follows:

- Efficient GPU implementation of a Non-Uniform grid method is presented to reduce memory requirements and its performance overhead is revealed.
- Novel implementation of ABCs is proposed and evaluated, which is coupled with periodic boundary conditions in view of a SIMD nature of GPUs.
- An implementation technique based on transformation of update equations of ABCs is introduced to reduce both of the computation amount and memory usage.

The proposed ideas are empirically evaluated by practical simulation of a micro strip antenna (MSA) in terms of performance and accuracy.

The rest of this paper is organized as follows. Section 2 introduces related works. Section 3 describes the 3D-FDTD method, the Split PML and basic implementation of 3D-FDTD method on a GPU. Our proposals are described in Sects. 4, 5 and 6. The memory usage, performance, accuracy and comparison with related works are evaluated in Sect. 7. Finally, Sect. 8 includes the conclusions and some directions for future work.

2. Related Works

So far, many researchers have reported the results of GPU acceleration of the stencil computation [8], [9], including 3D-FDTD method with ABCs [10]–[12]. Nagaoka *et al.* reported a performance comparison a 3D-FDTD method with the Split PML on a CPU, SX-8R super computer and Tesla C1060 GPU [11]. They used a human body model as a calculation target and pointed out that the performance var-

Manuscript received January 11, 2012.

Manuscript revised June 14, 2012.

[†] The author is with the Department of Science and Technology, Graduate School of Engineering, Nagasaki University, Nagasaki-shi, 852–8521 Japan.

[††] The authors are with the Division of Electrical Engineering and Computer Science, Graduate School of Engineering, Nagasaki University, Nagasaki-shi, 852–8521 Japan.

a) E-mail: dohi@pca.cis.nagasaki-u.ac.jp

DOI: 10.1587/transinf.E95.D.2787

ied with the calculation domain and CUDA thread block size. Implementation of a 3D-FDTD method was evaluated also on a variety of GPUs including CUDA incompatible GPUs [10]. Chu *et al.* achieved the performance of approximately 160 M elements updates per second using UPML [12]. While these works addressed parallelization and acceleration of a 3D-FDTD method for a normal computational space, there have been few reports that mainly discuss implementation techniques of ABCs and a Non-Uniform grid on GPUs.

3. Background

3.1 3D-FDTD Method

In this section, we briefly describe our 3D-FDTD simulation approach. Detailed mathematical background can found in other literature [1], [2]. We assumed that a simulation target is isotropy and non-dispersive having the conductivity of $\sigma = 0$. We also assumed permeability is uniform. We applied a typical substitution of central differences for the time and space derivatives for Maxwell's curl equations and thus our time-stepping expression can be written as for E_x for instance:

$$\begin{aligned} E_x^n\left(i+\frac{1}{2},j,k\right) &= E_x^{n-1}\left(i+\frac{1}{2},j,k\right) \\ &+ \frac{\Delta t}{\varepsilon_x\left(i+\frac{1}{2},j,k\right)\Delta y}\delta_y H_z^{n-\frac{1}{2}}\left(i+\frac{1}{2},j+\frac{1}{2},k\right) \\ &- \frac{\Delta t}{\varepsilon_x\left(i+\frac{1}{2},j,k\right)\Delta z}\delta_z H_y^{n-\frac{1}{2}}\left(i+\frac{1}{2},j,k+\frac{1}{2}\right) \end{aligned} \tag{1}$$

where Δy and Δz are cell sizes of each dimension, ε_x is a permittivity in x dimension, Δt denotes the time increment, and δ_u ($u \in \{x, y, z\}$) is a difference operator. For example, δ_y is defined as $\delta_y f(i, j, k) = f(i, j, k) - f(i, j-1, k)$.

3.2 Equations for the Split PML

Since the 3D-FDTD method handles the a finite computational space, the computational space is generally surrounded by zero values that correspond the Perfect Electric Conductor (PEC). The PEC reflects waves as a result of computation, and thus ABC are required to attenuate the reflection waves.

As previously mentioned, we used Bérenger's Split PML [13] as an ABC. Each component of $\boldsymbol{E}$ and $\boldsymbol{H}$ is split to two components called subcomponents, and the time-stepping equations of each subcomponent are defined as:

$$\begin{aligned} E_{xy}^n\left(i+\frac{1}{2},j,k\right) &= C_{E1y}(j)E_{xy}^{n-1}\left(i+\frac{1}{2},j,k\right) \\ &+ C_{E2y}(j)\delta_y H_z^{n-\frac{1}{2}}\left(i+\frac{1}{2},j+\frac{1}{2},k\right) \end{aligned} \tag{2}$$

where $C_{E1u}(p)$ and $C_{E2u}(p)$ ($u \in \{x, y, z\}$) are introduced for ease of describing the equations. $C_{E1u}(p)$ and $C_{E2u}(p)$ are defined as

$$C_{E1u}(p) = \frac{2\varepsilon_0 - \sigma_u(p)\Delta t}{2\varepsilon_0 + \sigma_u(p)\Delta t}, \quad u \in \{x, y, z\} \tag{3}$$

$$C_{E2u}(p) = \frac{2\Delta t}{2\varepsilon_0 + \sigma_u(p)\Delta t} \cdot \frac{1}{\Delta u}, \quad u \in \{x, y, z\} \tag{4}$$

where ε_0 shows permittivity of free space and $\sigma_u(p)$ shows conductivity at the coordinate p on the dimension u. Note that $\sigma_u(p)$ is 0 regardless of the value of p in non PML regions. Since $C_{E1u}(p)$ and $C_{E2u}(p)$ are constants in the time domain, these values can be calculated in advance of the simulation process. The Split PML needs two subcomponents for each component of $\boldsymbol{E}$ and $\boldsymbol{H}$. Thus, the Split PML needs up to 12 additional memory elements compared to normal FDTD calculation.

3.3 Basic Implementation of 3D-FDTD Method on CUDA-Compatible GPU

A general approach to implementation of the 3D-FDTD method on CUDA-compatible GPUs divides the whole simulation space to fixed size *blocks* and makes CUDA thread blocks process each *block* [8], [9]. Our implementation also divides the whole simulation target of size (S_x, S_y, S_z) into small *blocks* of size (B_x, B_y, B_z) and makes CUDA thread blocks process each *block*. Each component of $\boldsymbol{E}$ and $\boldsymbol{H}$ is stored memory as 3D-array so that dimension x has a unit-stride, dimension y has a larger stride, and dimension z has the largest stride. Figure 1 shows the placement of CUDA threads within a *block* and direction of processes. $B_x \times B_y$ CUDA threads within a CUDA thread block are placed on a 2D plane and each CUDA thread moves the process along the line with z direction. Therefore, the CUDA thread on the coordinate (x, y) processes a total of B_z cells from $(x, y, 0)$ to $(x, y, B_z - 1)$. When CUDA threads that have the same y coordinate in a block access to components of $\boldsymbol{E}$ and $\boldsymbol{H}$ at the same time, these accesses can be coalesced because these components are continuous on the GPU memory. Since the indices expressed in fractional numbers as like in Eq. (1) are inconvinient for straightforward implementation, we introduced a set of offset values to make the coordinates integer numbers as shown in Table 1. Among possible candidates of offset values, we chose the ones in Table 1 so that accesses

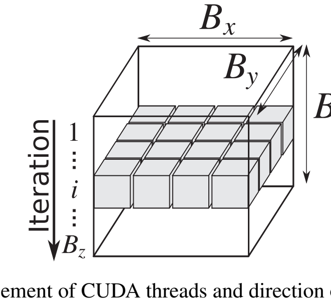


Fig. 1 Placement of CUDA threads and direction of process within a *block*. The gray boxes show CUDA threads.

Table 1 Offsets for components of $\boldsymbol{E}$ and $\boldsymbol{H}$.

	Components		
Field	x	y	z
$\boldsymbol{E}$	$(-1/2, 0, 0)$	$(0, -1/2, 0)$	$(0, 0, -1/2)$
$\boldsymbol{H}$	$(0, -1/2, -1/2)$	$(-1/2, 0, -1/2)$	$(-1/2, -1/2, 0)$

to the $\boldsymbol{E}$ and $\boldsymbol{H}$ components are coalesced as much as possible. We describe coordinates with the offsets using '[ ]' instead of '( )'. Thus, Eq. (1) can be rewritten as

$$\begin{aligned} E_x^n[i,j,k] = E_x^{n-1}[i,j,k] &+ \frac{C_{E2y}[j]}{\varepsilon_{xr}[i,j,k]}\delta_y H_z^{n-\frac{1}{2}}[i,j,k] \\ &- \frac{C_{E2z}[k]}{\varepsilon_{xr}[i,j,k]}\delta_z H_y^{n-\frac{1}{2}}[i,j,k] \end{aligned} \quad (5)$$

and the update equation of the Split PML for E_x can also be rewritten as

$$\begin{aligned} E_x^n[i,j,k] = C_{E1y}[j]\,E_{xy}^{n-1}[i,j,k] &+ C_{E2y}[j]\,\sigma_y H_z^{n-\frac{1}{2}}[i,j,k] \\ &+ C_{E1z}[k]\,E_{xz}^{n-1}[i,j,k] \\ &- C_{E2z}[k]\,\sigma_z H_y^{n-\frac{1}{2}}[i,j,k]. \end{aligned} \quad (6)$$

Hereafter, we discuss our implementation in the converted coordinates as shown in Eqs. (5) and (6).

4. The Non-Uniform Grid

4.1 Introduction of the Non-Uniform Grid

Since the size of the FDTD simulation is limited by available size of memory on a GPU device in effect, we introduce the liner interpolated Non-Uniform grid [14], [15] to reduce memory usage for simulation. By using multiple sizes of simulation cells, the Non-Uniform grid approach aims at effective reduction of the memory usage, floating point operations and data memory accesses while sustaining the simulation accuracy. The computation error caused by the use of Non-Uniform grid is discussed in [15]. Applying a linear interpolation to Eq. (4), we got:

$$C_{E2u}[p] = \frac{2\Delta t}{2\varepsilon_0 + \sigma_u[p]\Delta t} \cdot \frac{2}{\Delta u[p-1] + \Delta u[p]}, \quad u \in \{x, y, z\} \quad (7)$$

where $\Delta u[p]$ is a cell size of coordinate p in dimension u.

4.2 Implementation

Important things when introducing the Non-Uniform grid are that how much memory usage can be reduced and how many additional costs, such as floating point operations and data memory accesses, we have to pay. Here, we discuss the additional costs to use the method. In the case of the Uniform grid, we can use $\Delta t/(\varepsilon_0 \Delta u)$ instead of $C_{E2u}[p]$ in Eq. (1). Because $\Delta t/(\varepsilon_0 \Delta u)$ is a constant, it need not to be fetched from the global memory. Therefore, the additional memory access cost for the Non-Uniform grid is made by fetches of six float values; $C_{E2u}[p]$ and $C_{H2u}[p]$ $(u \in \{x, y, z\})$, that is $6 \times 4 = 24$ Bytes, per cell.

Since $C_{E2u}[p]$ and $C_{H2u}[p]$ are constant values through the simulation, we can place these values on the constant memory or texture memory instead of the global memory to mitigate the access penalty. In addition, appropriate thread placement and blocking can make further reduction of the accesses. As described in Sect. 3.3, each CUDA thread has the invariant coordinate (x, y) in our implementation. Therefore, each CUDA thread needs to fetch the values for $C_{E2x}[i]$, $C_{H2x}[i]$, $C_{E2y}[j]$ and $C_{H2y}[j]$ only once when the CUDA kernel is launched. Moreover, the values of $C_{E2z}[k]$ and $C_{H2z}[k]$ can be shared by all of the threads within the same CUDA thread block, so that a total of B_z fetches are required per CUDA thread block while processing a *block* of (B_x, B_y, B_z).

To summarize, the Non-Uniform grid method needs additional $4{\times}(2B_x{+}2B_y{+}2B_z)$ data fetches from the arrays of $C_{E2u}[p]$ and $C_{H2u}[p]$ $(u \in \{x, y, z\})$ per *block*. Therefore, the additional memory access amount per grid point becomes

$$\frac{4 \times (2B_x + 2B_y + 2B_z)}{B_x \times B_y \times B_z} \text{ Bytes}, \quad (8)$$

which can be mitigated by an appropriate blocking size.

5. Novel Absorbing Boundary Condition

5.1 Motivation

As already shown, the Split PML as an ABC increases the number of memory accesses and arithmetic operations per cell due to the subcomponents. Generally, the ABC forms thin regions that is smaller than the size of *block*. When we allocate computation for the ABC regions and other regions into the same *block*, instruction and data flows make divergences which results in severe performance degradation due to a SIMD property of GPU architectures. To avoid this situation, the ABC regions and other regions should be placed in different *blocks*. However, in turn, since the ABC regions are thin, the ABC *blocks* need a lot of padding cells to align the blocks especially in the dimensions with a large block size. In addition, we need at least two ABC *blocks* per dimension. While the padding cells consume resources, they do not make any contributions to the computation. However, the threads assigned for the padding cells execute the same instruction flow as the ones for the ABC regions to keep the SIMD property and thus to avoid the thread divergence in the *block*.

5.2 Introduction of the Periodic Boundary Condition

In order to reduce the excess padding cells, we propose a novel implementation of an ABC introducing the idea of

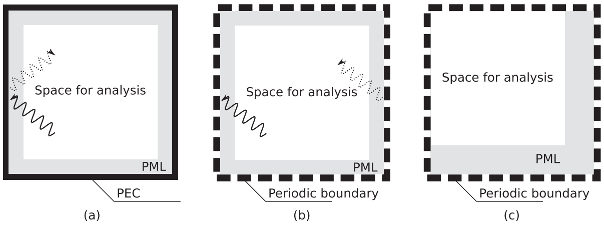


Fig. 2 (a) The model that has the Split PML and PEC. A dashed line shows reflected wave before attenuation. (b) The model that has the periodic boundary condition instead of the PEC. (c) The model that we use.

the periodic boundary condition. In the periodic boundary condition, one side of the simulation space continues to the other side. Thus, using this periodicity, we can gather the ABC regions in one side of each dimension.

The periodic boundary condition can be introduced by changing the definition of the difference operator δ_u in Eq. (1). For example, δ_y is defined as

$$\delta_y f(i, j, k) = f(i, j, k) - f(i, (j + S_y - 1) \bmod S_y, k) \quad (9)$$

where S_y is the size of dimension y.

In the original configuration as shown as Fig. 2 (a), emitted waves are finally reflected by the PEC, but before that, they are sufficiently attenuated by Split PMLs so as not to affect the simulation results. Therefore, we can use the periodic boundary condition instead of the PEC as show in Fig. 2 (b), since waves that leap the boundary of the simulation space should be also enough weak not to affect the simulation. Additionally, as shown in Fig. 2 (c), we can gather the Split PMLs on one side in each dimension for efficient blocking.

To summarize, by introducing the periodic boundary condition, the number of the ABC *blocks* for each dimension can be reduced from 2 to 1, and thus the number of padding elements can also be reduced. Note that the number of cells used for the Split PMLs is not changed by introducing the periodicity.

6. Transformation of Update Equations of the Split PML

6.1 Motivation

It is independent in terms of dimension if a cell is in an ABC region or not. For example, there can be a cell that is in an ABC region in dimension z but not in dimension y. For such cells, we propose transformation of update equation of the Split PML Eq. (6) to reduce the required memory size and access amount at the cost of slight increase in the operation count, considering that the simulation size is limited by the available memory size on the GPU device.

6.2 Transformation

Given that a cell that is inside the normal space region in dimension y, we get $\sigma_y [j] = 0$ and the following equation

Table 2 Comparison of memory access requirements of each update equation for E_x. NON_ABC shows the equation that both dimensions are in the normal region, ORIGINAL shows the original equation of the Split PML and TRANSFORMED shows the transformed equation, respectively.

	NON_ABC	ORIGINAL	TRANSFORMED
Read	5	6	6
Write	1	3	2
Total	6	9	8
Ratio	1	1.5	1.333

$$\begin{aligned} E_x^n [i, j, k] = E_{xy}^{n-1} [i, j, k] & \\ + C_{E2y} [j] \sigma_y H_z^{n+\frac{1}{2}} [i, j, k] & \\ + C_{E1z} [k] E_{xz}^{n-1} [i, j, k] & \\ - C_{E2z} [k] \sigma_z H_y^{n+\frac{1}{2}} [i, j, k] & \end{aligned} \quad (10)$$

by substituting $\sigma_y [j] = 0$ to Eq. (6). From the definition of subcomponents [13], we get

$$E_{xy}^{n-1} [i, j, k] = E_x^{n-1} [i, j, k] - E_{xz}^{n-1} [i, j, k] . \quad (11)$$

By substituting Eq. (11) into Eq. (10), we get

$$\begin{aligned} E_x^n [i, j, k] = E_x^{n-1} [i, j, k] & \\ + C_{E2y} [j] \sigma_y H_z^{n+\frac{1}{2}} [i, j, k] & \\ + (C_{E1z} [k] - 1) E_{xz}^{n-1} [i, j, k] & \\ - C_{E2z} [k] \sigma_z H_y^{n+\frac{1}{2}} [i, j, k] . & \end{aligned} \quad (12)$$

Comparing Eqs. (6) and (12), it is clear that we can reduce the number of accesses to the subcomponent E_{xy} as well as the total memory usage. On the other hand, while original Eq. (10) includes the intermediate term which can be reused for $E_{xz}^n [i, j, k]$, the transformed Eq. (12) does not, which means this techniques required one additional operation for each cell. In this sense, this transformation approach is oriented for memory constrained devices like GPUs.

Table 2 shows the comparison of memory access of update equations for E_x. Since the transformed equation saves one write access compared to the original equation, this technique can improve the performance for the region that only one dimension is in the Split PML. We obtained a total of $2^3 = 8$ time-stepping equations for $\boldsymbol{E}$ by applying the same transformation to E_y and E_z. In addition, we also got similar 8 equations for $\boldsymbol{H}$. Note that these transfomed equations are mathematically equivalent to the original ones, so that simulation accuracy is not affected.

7. Performance Evaluation and Analysis

7.1 Running Example

We chose a stacked rectangular microstrip antenna (MSA) with a shorting plate, which has been proposed for dual band operation in [16], as a target of numerical calculation. MSAs are widely used in mobile communications due to their lower profile, weight, and manufacturing costs, as

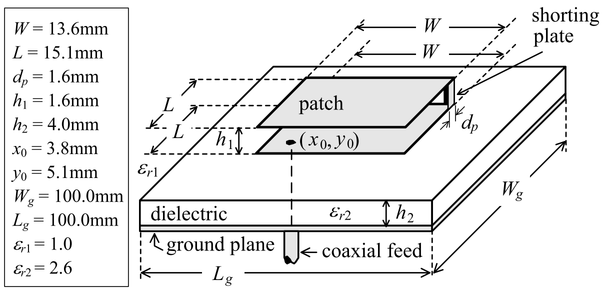


Fig. 3 The geometry of a stacked rectangular MSA.

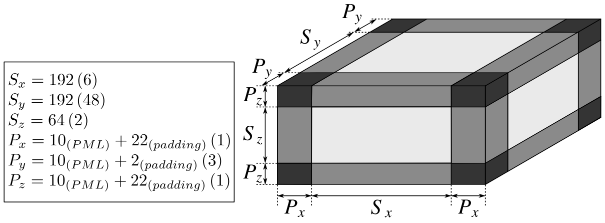

Sx = 192 (6)
Sy = 192 (48)
Sz = 64 (2)
Px = 10(PML) + 22(padding) (1)
Py = 10(PML) + 2(padding) (3)
Pz = 10(PML) + 22(padding) (1)
Py
Sy
Py
Pz
Sz
Pz
Px
Sx
Px

Fig. 4 Numbers that provided in parentheses show the number of *blocks*. The color of each region shows the group of update equations and darker colors show higher computational costs, that is floating point operations and memory accesses. 10-cell thick PMLs are used for the boundaries of each dimension. Since the *block* size is 32 in both x and z dimension, 22 padding cells are required for both P_x and P_z. On the other hand, since the *block* size is four in y dimension, three *blocks* are used for P_y to cover the 10-cell thick PMLs, introducing two padding cells.

well as their compatibility with integrated circuit technologies. The MSA proposed in [16] radiates a linearly polarized wave at the lower frequency band and a circularly polarized wave at the higher frequency band.

Figure 3 shows the geometry of a stacked rectangular MSA with a shorting plate. The antenna consists of a dielectric substrate and a layer of air with a rectangular patch. The upper and lower patches are the same size. The upper patch is shorted to the lower patch at the apex by a conducting plate. The relative dielectric constant the upper and lower layer are $\varepsilon_{r1} = 1.0$ and $\varepsilon_{r2} = 2.6$. The antenna is excited at the lower patch by a coaxial feed through the lower dielectric substrate at point which lays around the diagonal.

7.2 Modeling

Many work including ours have reported the *block* size has a strong effect on the performance [7]. Although the best size may depend on the presence or absence of each optimization technique, we used the *block* size of $(B_x, B_y, B_z) = (32, 4, 32)$ based our primary evaluation results.

Figure 4 shows the initial simulation model that is based on the MSA shown in Fig. 3 with 10 layers of the Split PML. A total of (256×216×128) cells were used to make the Non-Uniform grid, where 5 kinds of cell sizes were used; 1 mm to 0.1 mm. The size of the *block* is (32, 4, 32) as aforementioned. S_x, S_y and S_z in Fig. 4 show the number of cells in each dimension and P_x, P_y and P_z are the number of cells in each dimension of the ABC with padding.

Table 3 The number of *block*, cells and memory usage.

region	# of *blocks*	# of cells	rate	memory	ratio
NON_ABC	576	2,359,296	.333	54.0 MB	.143
PML1	840	3,440,640	.486	236.3 MB	.625
PML2	288	1,179,658	.167	81.0 MB	.214
PML3	24	98,304	.014	6.8 MB	.018
Total	1,728	7,077,888	1	378.1 MB	1

Table 4 Comparison of memory usage between the Non-Uniform grid and Uniform grid in the running example.

	# of cells	memory usage
Uniform grid	(1,120 × 1,100 × 224) + PML	13,584.6 MB
Non-Uniform grid	(192 × 192 × 64) + PML	378.1 MB

Table 5 Execution time for the Non-Uniform grid and the Uniform grid. Both methods processed the same number of cells. (256 × 212 × 128)

	Execution time (sec)
Uniform	155.81
Non-Uniform	155.86

Table 3 shows the number of *blocks*, the number of cells and memory usage for each region of the model. We classified cells into the four categories according to presence or absence of the ABC; all dimensions are not in the ABC (NON_ABC), any one of three dimensions is in the ABC (PML1), any two of the three are in the ABC (PML2), and all the dimensions are in the ABC (PML3). Approximately 67% of the cells including padding is in the ABC.

As an implementation platform, we used the GeForce GTX 295, which has 30 Streaming Processors and 896 MB GDDR3 memory per GPU core. While the GPU has two GT200b GPU cores, we used only one GPU core in this implementation. Single precision floating point operations were utilized through the simulation.

7.3 Effect of Non-Uniform Grid

First, we evaluated the Non-Uniform Grid method in terms of memory usage and execution time. Table 4 shows comparison results of the Global memory usage on the GPU. We used the finest cells in the case of the Uniform Grid. Although it depends on a simulation model how much memory usage can be reduced, the reduction of 1/36 was achieved in this running example.

Next, we evaluated the penalty of the Non-Uniform grid. As already shown in Sect. 4.2, additional costs of using the Non-Uniform grid arisen from additional fetches of $C_{E2u}[p]$ and $C_{H2u}[p]$ in non ABC regions. To evaluate this penalty, we modified our implementation not to fetch $C_{E2u}[p]$ and $C_{H2u}[p]$, and to use the constant values of $(\Delta t/\varepsilon_0)$ and $(\Delta t/\mu_0)$ instead of them. Table 5 shows the result using the simulation model shown in Fig. 4. Note that we focus only on revealing the performance difference here and thus this modification does not preserve the simulation results.

The results show that the Non-Uniform grid degrades the simulation performance per cell by about 0.03% due to additional fetches of $C_{E2u}[p]$ and $C_{H2u}[p]$. In view of the 1/36 reduction of required number of cells, our implementation approach of the Non-Uniform method is efficient for for both reduction of memory usage and performance improvement.

7.4 The Periodic Boundary Condition with ABC

Figure 5 shows the simulation model with the periodic boundary condition. Compared to the model shown in Fig. 4, the number of *blocks* of ABC is smaller in all dimension. Table 6 shows the number of *blocks*, cells and memory usage. Clearly, the periodic boundary condition reduces memory usage. Figs. 6 (a) and (b) illustrate the execution time of these models. By reducing cells for the ABCs, the periodic ABC achieved 1.8 times faster performance than the Non-Periodic ABC.

The periodic boundary condition significantly reduces the computational region, thus both memory usage and computational performance are improved. We concerned that costly memory access with a long stride would decrease the performance. However, as experiment results show, the method enabled efficient implementation on a GPU.

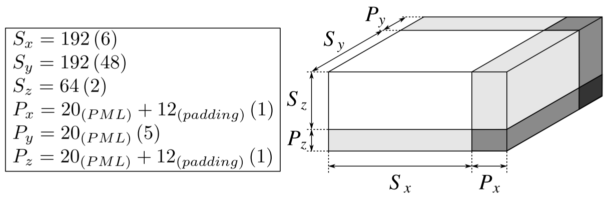


Fig. 5 The simulation model with the periodic boundary condition to the model shown in Fig. 5. Using the periodic boundary condition with ABC, 20 PML cells in total are assigned to P_x and P_z, reducing the nubmer of padding cells 22 to 12. In y dimension, five *blocks* are required for P_y to cover the 20 PML cells.

Table 6 The number of *block*, cells and memory usage with the Non-Uniform grid and the periodic boundary condition.

region	# of *blocks*	# of cells	ratio	memory	ratio
NON_ABC	576	2,359,296	.518	54.0 MB	.263
PML1	444	1,818,624	.399	124.9 MB	.609
PML2	88	360,448	.079	24.8 MB	.121
PML3	5	20,480	.004	1.4 MB	.007
Total	1,033	4,558,848	1	205.1 MB	1

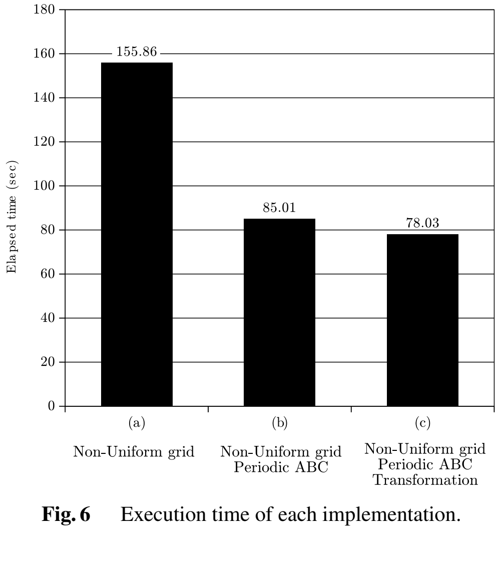


Fig. 6 Execution time of each implementation.

7.5 Effect of Transformation

Table 7 shows the number of *blocks*, the number of cells and memory usage after the transformation of the update expressions. The transformation reduces memory usage of PML1 and PML2 so that we only needs 70% of memory usage in the case of the model shown in Table 6.

Figures 6 (b) and (c) summarize the performance comparison results for the transformation of the expression. In the running example, the transformation of the Split PML improved the execution performance by 8.9%.

7.6 Simulation Accuracy

In order to evaluate the simulation accuracy of our proposed implementation techniques, we compared the characteristics measured by the real antenna and our simulation results.

In analysis of antennas, radiation and feed point characteristics are discussed generally. We calculated axial ratio of a circularly polarized wave as radiation characteristic and return loss as feed point characteristic at the higher frequency band in the dual band MSA. Figures 7 and 8 show the calculated axial ratio and return loss, respectively. The measured results are also shown for comparison. The relative error of the frequency at the minimum axial ratio between the calculated and measured results is 0.5%. The calculated and measured minimum axial ratio are 0.82 dB and 0.43 dB, respectively. In the return loss, the double peaking behaviors are observed in both of the calculated and measured results. The relative errors of the frequencies at the two peaks between the calculated and measured results are 0.3% and 0.7%. The calculated return losses at the two peaks are −20.6 dB and −27.6 dB and the measured ones are −19.6 dB and −26.9 dB. In both of the axial ratio and the return loss, the calculated results agree well with the measured ones.

Table 7 The number of *block*, cells and percentage of total with the Non-Uniform grid, the periodic boundary condition and transformation of update equations.

region	# of *blocks*	# of cells	ratio	memory	ratio
NON_ABC	576	2,359,296	.518	54.0 MB	.375
PML1	444	1,818,624	.399	69.4 MB	.482
PML2	88	360,448	.079	19.3 MB	.134
PML3	5	20,480	.004	1.4 MB	.010
Total	1,033	4,558,848	1	144.1 MB	1

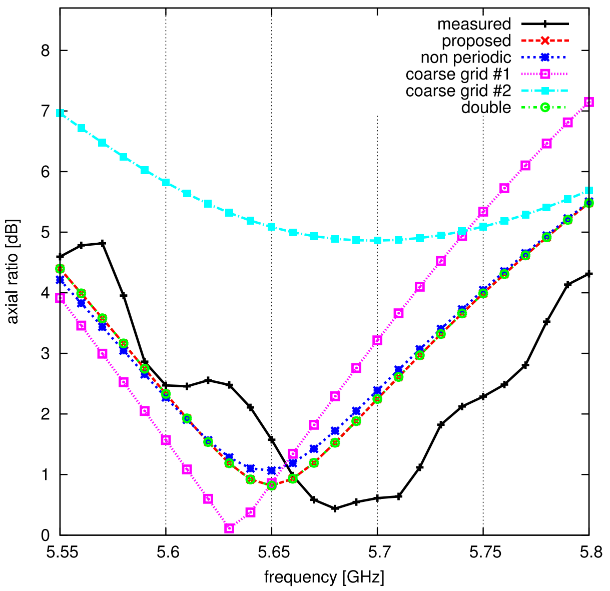


Fig. 7 Comparison of minimum axial ratio between the calculated and measured results. 'measured': Measured data from actual antenna. 'proposed': Simulation results with our proposed method. 'non periodic': Simulation results with non-periodic ABC. 'coarse grid #1': Simulation results with double sized cells. 'coarse grid #2': Simulation results with quadruple sized cells. 'double': Simulation results with double precision floating operations. Note that the two lines, 'proposed' and 'double', are almost overlapped.

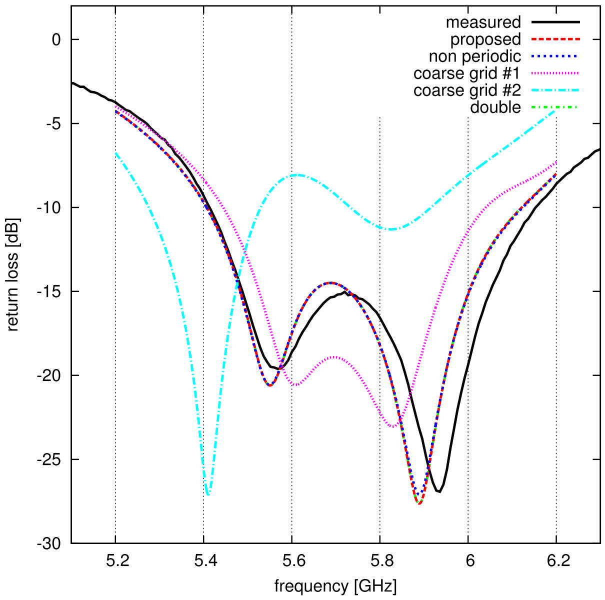


Fig. 8 Comparison of the return loss between the calculated and measured results. Note that the three lines, 'proposed', 'non periodic' and 'double', are almost overlapped.

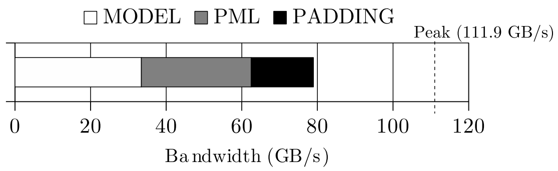


Fig. 9 Required memory bandwidth for each part of the antenna model. Bandwidth of MODEL, PML and PADDING are 33.4 GB/s, 29.0 GB/s and 16.5 GB/s, respectively.

First, in order to assess how periodic boundary condition affected the simulation accuracy, we also implemented simulation code with the original non-periodic boundary condition and compared the simulation results. As Figs. 7 and 8 show, slight differences were observed in the trends of axial ratios, but the peak frequency found was identical. On the other hand, the difference in the boundary condition hardly affected the results in terms of the return loss.

Next, aiming at examining numerical error in the simulation, we compared our simulation results with two different grid models with coarse cells: cells with double size (coarse grid #1) and cells with quadruple size (coarse grid #2) in each dimension. That is, these coarse grid models consist of 1/8 and 1/64 cells in total compared to the original model, respectively. As Figs. 7 and 8 show, the finer the cell size, the more accurate results were obtained for both the axial ratios and the return losses, suggesting the reasonability of our simulation scheme.

Finally, we explored how arithmetic precision affected the simulation accuracy, by executing the simulation with double precision arithmetic. As shown in Figs. 7 and 8, almost no difference were observed between the simulation results for the single precision and double precession; the maximum relative error was less than 0.007%. This suggests single precision arithmetic offers enough simulation accuracy as far as our application field is concerned.

7.7 Memory Bandwidth

Next, we evaluated achieved memory bandwidth of our implementation, since this often tends to become a performance bottleneck for GPU implementation of stencil computation. Our simulation model consists of three parts as follows:

- MODEL: A main part including the MSA
- PML: The Split PML part
- PADDING: Padding part.

Figure 9 shows achieved memory bandwidth of whole application and ratios of each part.

MODEL, PML and PADDING accounted 42.3%, 36.8% and 20.9% of the total achieved memory bandwidth, respectively. These ratios differ a little from the ratios of the number of cells shown in Table 7. This should come from the fact that the process for PADDING cells is the same as that of the PML cells, and that the PML needs 1.4~1.9 times greater memory access than the normal cells.

Compared with the peak memory bandwidth of the target GPU, our implementation achieved about 55.8% of the peak bandwidth when the accesses for PADDING region are not taken account. If we include PADDING accesses, the achieved bandwidth becomes 70.5% of the peak. As far as our knowledge goes, there is no other report of 3D-FDTD

Table 8 Performance comparison with previous works. The peformance parameters are from announced values [17]–[20] unless otherwise noted. *Performance per GPU core. **Calculated by the values shown in [12]. †Exact GPU model number is not mentioned in [10]. ‡Not reported in the literature.

	Our Proposal	Our Proposal (double)	CPU	[11]	[12]	[10]
GPU	GeForce GTX295	GeForce GTX295	Core i7 920	Tesla C1060	ATI HD4850	ATI X800†
Peak Bandwidth (GB/s)	111.9*	111.9*	25.6	102	63.6**	28.8~32
Size	192 × 192 × 64	192 × 192 × 64	182 × 186 × 69	204 × 145 × 499	180 × 180 × 180	160,000 points
ABC	Split PML	Split PML	Split PML	Split PML	UPML	CPML
	10 layers	10 layers	10 layers	8 layers	-‡	-‡
Grid	Non-Uniform	Non-Uniform	Non-Uniform	Uniform	Uniform	Uniform
Model	MSA	MSA	MSA	Human Body	Vacuum	Vacuum
Precision	Single	Double	Single	-‡	Single	-‡
Throughput (Mpoints/s)	453	140	2.66	45	160	30
Throughput/Bandwidth (Mpoints/GB)	4.05	1.25	0.1	0.43	2.51	0.96~1.04

implementation with ABCs that achieves the same degree of high memory bandwidth or efficiency for the practical model. This suggests the effectiveness of our proposed techniques.

7.8 Performance Comparison

Table 8 shows performance comparison with related works. Note that Table 8 shows peak bandwidth for one of two GPU cores on the GeForce GTX295 because we only used only one GPU core. Perfect equalization of comparison conditions is difficult of course, but we tried to be fair in choosing comparison metrics. We compared the simulation throughputs in million-point updates per second (Mpoints/s) and the simulation throughputs per peak bandwidth in million-point updates per giga bytes (Mpoints/GB) in order to alleviate differences in the size of simulation models and utilized GPU architectures. While ABC regions are excluded from the model size, the execution time used for the throughput calculation includes those of the ABC regions and Padding regions. We also presented the performance for the double precision version of our simulation code, since arithmetic precision is not explicitly mentioned in the literature for some comparison targets. Our implementation shows good results for both metrics among the compared implementation, suggesting effectiveness of our implementation approach for GPU architectures.

We also compared the performance with that of our CPU-based implementation. While parallel processing with multi-core, multi-thread and SIMD instructions were not utilized in this implementation, the code was optimized considering the cache structure. Table 8 shows the performance of the GPU implementation achieved about 170 times throughput than the CPU implementation.

8. Conclusion

In this paper, we addressed the efficient implementation of absorbing boundary conditions of 3D-FDTD method for antenna designing on CUDA-compatible GPU. To reduce memory usage and to improve the simulation performance, a Non-Uniform grid method and the periodic boundary conditions were applied for Split PML implementation. The transformation technique of update-equations for partial ABC cells was also proposed. The empirical experiment showed that the proposed methods almost doubled the simulation performance and eventually achieved the memory bandwidth of 62.5 GB/s which corresponds to 55.8% of the peak of the target GPU. Our interest future work includes expansion of our method to treat dispersive materials which requires more complex update-equations.

References

[1] K. Yee, "Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media," IEEE Trans. Antennas Propag., vol.AP-14, no.3, pp.302–307, 1966.

[2] A. Taflove and M.E. Brodwin, "Numerical solution of steady-state electromagnetic scattering problems using the time-dependent Maxwell's equations," IEEE Trans. Microw. Theory Tech., vol.MTT-23, no.8, pp.623–630, 1975.

[3] S. Gedney, "An anisotropic perfectly matched layer-absorbing medium for the truncation of FDTD lattices," IEEE Trans. Antennas Propag., vol.44, no.12, pp.1630–1639, 1996.

[4] J. Roden and S. Gedney, "Convolution PML (CPML): An efficient FDTD implementation of the CFS-PML for arbitrary media," Microw. Opt. Technol. Lett., vol.27, no.5, pp.334–339, 2000.

[5] S. Cummer, "A simple, nearly perfectly matched layer for general electromagnetic media," IEEE Microwave and Wireless Components Lett., vol.13, no.3, pp.128–130, 2003.

[6] J. Bérenger, Perfectly matched layer (PML) for computational electromagnetics, Morgan & Claypool Publishers, 2007.

[7] K. Dohi, Y. Shibata, K. Oguri, and T. Fujimoto, "GPU implementation and optimization of electromagnetic simulation using the FDTD method for antenna designing," SIGARCH Comput. Archit. News, vol.39, pp.26–31, Dec. 2011.

[8] K. Datta, M. Murphy, V. Volkov, S. Williams, J. Carter, L. Oliker, D. Patterson, J. Shalf, and K. Yelick, "Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures," Proc. 2008 ACM/IEEE Conference on Supercomputing, SC '08, pp.4:1–4:12, IEEE Press, Piscataway, NJ, USA, 2008.

[9] P. Micikevicius, "3D finite difference computation on GPUs using CUDA," Proc. 2nd Workshop on General Purpose Processing on Graphics Processing Units, GPGPU-2, pp.79–84, ACM, New York, NY, USA, 2009.

[10] M. Inman, A. Elsherbeni, J. Maloney, and B. Baker, "GPU based FDTD solver with CPML boundaries," IEEE Antennas and Propa-

gation Society Int'l Symp., pp.5255–5258, IEEE, 2007.

[11] T. Nagaoka and S. Watanabe, "A GPU-based calculation using the three-dimensional FDTD method for electromagnetic field analysis," Annu. Int'l Conf. of the IEEE Engineering in Medicine and Biology Society (EMBC), pp.327–330, IEEE, 2010.

[12] T. Chu, J. Dai, D. Qian, W. Fang, and Y. Liu, "A novel scheme for high performance Finite-Difference Time-Domain (FDTD) computations based on GPU," Algorithms and Architectures for Parallel Processing, pp.441–453, 2010.

[13] J. Bérenger, "A perfectly matched layer for the absorption of electromagnetic waves," J. Comput. Phys., vol.114, no.2, pp.185–200, 1994.

[14] C. Railton and J. McGeehan, "Analysis of microstrip discontinuities using the finite difference time domain technique," IEEE MTT-S Int'l Microwave Symp. Digest, pp.1009–1012, IEEE, 1989.

[15] H. Jiang and H. Arai, "Analysis of computation error in antenna's simulation by using non-uniform mesh FDTD," IEICE Trans. Commun., vol.E83-B, no.7, pp.1544–1553, July 2000.

[16] T. Fujimoto and K. Tanaka, "Stacked rectangular microstrip antenna with a shorting plate for dual band (vics/etc) operation in ITS," IEICE Trans. Commun., vol.E90-B, no.11, pp.3307–3310, Nov. 2007.

[17] NVIDIA, "GeForce GTX 295 Overview," http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-295

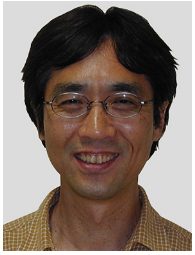
[18] NVIDIA, "Tesla C1060 computing processor board," http://www.nvidia.com/docs/IO/43395/BD-04111-001_v06.pdf

[19] Intel, "Intel Core i7-900 desktop processor extreme edition series and Intel Core i7-900 desktop processor series datasheet, Vol 1."

[20] ATi, "ATi RADEON X800 3D architecture white paper."

Keisuke Dohi received B.E. and M.E. degrees from Nagasaki University, Japan, in 2009 and 2011, respectively. His research interests include reconfigurable system and GPGPU computing.

Yuichiro Shibata received the B.E. degree in electrical engineering, the M.E. and Ph.D. degrees in computer science from Keio University, Japan, in 1996, 1998 and 2001, respectively. Currently, he is an associate professor at Department of Computer and Information Sciences, Nagasaki University. He was a Visiting Scholar at University of South Carolina in 2006. His research interests include reconfigurable systems and parallel processing. He won the Best Paper Award of IEICE in 2004.



Kiyoshi Oguri received B.S. and M.S. degrees in physics from Kyushu University, Japan, in 1974 and 1976, respectively. He also received the Ph.D. degree in information engineering from the same university in 1997. Since joining NTT in 1976, he have been engaged in the research, design and development of high-end general purpose computer, high-level logic synthesis system and a wired logic-based dynamic computing architecture. Currently, he is a professor of Nagasaki University, Japan. His research interests are in hardware modeling, high-level synthesis, FPGA-related systems, and Plastic Cell Architecture. Prof. Oguri received the Motooka Prize in 1987, the Best Paper Award of IPSJ in 1990, the Okochi Memorial Technology Prize in 1992, the Achievement Award of IEICE in 2000, and the ACM Gordon Bell Prize in 2009.

Takafumi Fujimoto received the B.E and M.E. degrees from Nagasaki University in 1992 and 1994, respectively, and a Dr. Eng. degree from Kyushu University in 2003. He is currently an Associate Professor at Nagasaki University. From Nov. 2004 to Sept. 2005, he was a Visiting Scholar in the department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign. His main interests are the analytical method and design of printed antennas in antenna engineering and the diffraction-free beams in optical engineering. Dr. Fujimoto is a member of the IEEE and Optical Society of America (OSA).