Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 823426, 9 pages
http://dx.doi.org/10.1155/2015/823426

Research Article

Parallel Numerical Simulations of Three-Dimensional Electromagnetic Radiation with MPI-CUDA Paradigms

Bing He, Long Tang, Jiang Xie, XiaoWei Wang, and AnPing Song

High Performance Computing Centre, Shanghai University, Shanghai 200436, China

Correspondence should be addressed to Bing He; hebing@shu.edu.cn

Received 28 August 2014; Accepted 15 December 2014

Academic Editor: L. W. Zhang

Using parallel computation can enhance the performance of numerical simulation of electromagnetic radiation and get great runtime reduction. We simulate the electromagnetic radiation calculation based on the multicore CPU and GPU Parallel Architecture Clusters by using MPI-OpenMP and MPI-CUDA hybrid parallel algorithm. This is an effective solution comparing to the traditional finite-difference time-domain method which has a shortage in the calculation of the electromagnetic radiation on the problem of inadequate large data space and time. What is more, we use regional segmentation, subregional data communications, consolidation, and other methods to improve procedures nested parallelism and finally verify the correctness of the calculation results. Studying these two hybrid models of parallel algorithms run on the high-performance cluster computer, we draw the conclusion that both models are suitable for large-scale numerical calculations, and MPI-CUDA hybrid model can achieve higher speedup.

1. Introduction

Finite-difference time-domain (FDTD) method has become a common method for solving Maxwell's equations [1]. It is a full vector method and can be naturally given time-domain and frequency-domain information user need. This is the unique advantage in electromagnetic and photonic application. FDTD algorithm is discrete in terms of time and space. Therefore, the structure of the electromagnetic field must be described on the grid by the Yee cellular composition. Maxwell's equation is discrete in time factor; therefore, time step is closely related to the mesh size. When mesh size tends to zero in the limit case, the discrete model accurately describes Maxwell's equations.

Recently, general-purpose computing on a graphics processing unit (GPGPU) has received considerable attention in many scientific fields [2–4] because a GPGPU offers high computational performance at low cost. What is more, Intel Xeon Phi coprocessor, based on the Many Integrated Core (MIC) architecture, packs up to 1 TFLOP of double precision performance in one chip. It runs a Linux operations system and provides x86 compatibility and also supports several popular programming models including MPI, OpenMP, Thread Building Blocks, and others that are used on multicore architectures. High-performance computer architecture tends to hybrid system, and this corresponds to the software program design requirements mixed programming model. GPGPU and MIC accelerated computing components which appeared in recent years provide the opportunity to improve the performance of FDTD parallel algorithm. Therefore, we achieved the parallel three-dimensional FDTD algorithm based MPI-CUDA model.

The FDTD algorithm obtains a wide range of applications in many fields of electromagnetic radiation, such as radiation antenna analysis, scattering calculations, electronic packaging, and radar. With the development of high-performance computing, the MPI has solved a weakness that the computing time of the FDTD parallel algorithm [5, 6] is too long. However, increasing amount of computation, the MPI process in a single node increases computational burden. When we use two different hybrid models which are MPI-OpenMP model and MPI-CUDA model to solve this problem, we can use the distributed shared memory features to improve the parallel speedup and scalability [7, 8].

The rest of the paper is organized as follow. We present the FDTD algorithm with uniaxial perfectly matched layer

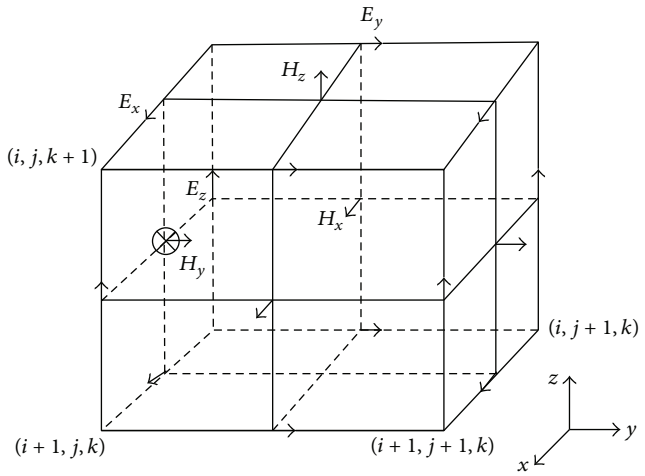


Figure 1: The structure of Yee cell.

(UPML) in Section 2. Then we describe the procedure and present basic steps for the method acceleration by means of MPI-OpenMP paradigms and MPI-CUDA paradigms in Sections 3 and 4. In Section 5, the performance of parallel computing of the two methods was compared and analyzed the factor that affects performance. The conclusions are given in Section 6 finally.

2. FDTD Algorithm

FDTD algorithm is a numerical method based on Maxwell's equations. The algorithm uses leapfrog calculation method and alternating electric field and magnetic field distribution in space within a half step sampling by Yee cellular composition [9].

From Figure 1, we can see that the Yee cell has the following characteristics: each magnetic field component was surrounded by four electric field components and each electric field component was surrounded by the four components of the magnetic field, and these field components placement relative position in the Yee cell and automatically satisfy the continuity conditions in the interface. This sampling method not only meets Maxwell's equations difference calculation but also meets Faraday's law of electromagnetic induction and the natural Ampere's law [10]. Therefore, this method gradually completes recursive entire electromagnetic fields. First, the explicit equations for the E_x and H_x are given by

$$\begin{aligned} &E_x^{n+1}\left(i+\frac{1}{2}, j, k\right) \\ &\quad = CA\left(i+\frac{1}{2}, j, k\right) + CD \cdot CB\left(i+\frac{1}{2}, j, k\right) \\ &\qquad \cdot \left[H_z^{n+1/2}\left(i+\frac{1}{2}, j+\frac{1}{2}, k\right) - H_z^{n+1/2}\left(i+\frac{1}{2}, j-\frac{1}{2}\right)\right. \\ &\qquad - H_y^{n+1/2}\left(i+\frac{1}{2}, j, k+\frac{1}{2}\right) \\ &\qquad \left. + H_y^{n+1/2}\left(i+\frac{1}{2}, j, k-\frac{1}{2}\right)\right] \end{aligned} \tag{1}$$

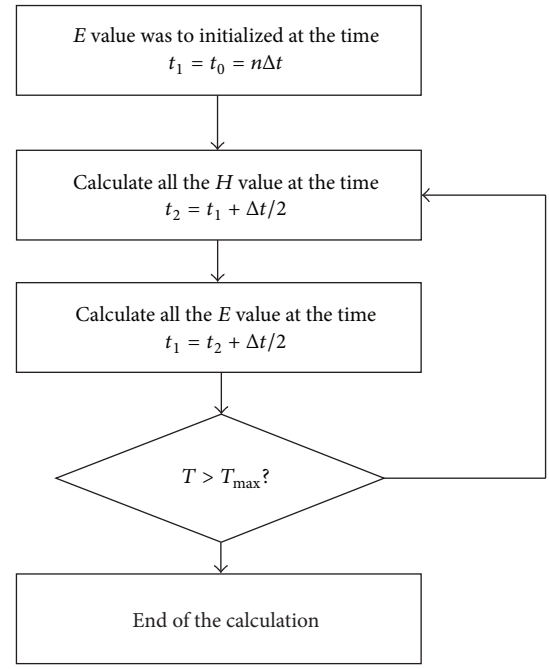


Figure 2: The workflow of the FDTD method.

$$\begin{aligned} &H_x^{n+1/2}\left(i, j+\frac{1}{2}, k+\frac{1}{2}\right) \\ &\quad = H_x^{n-1/2}\left(i, j+\frac{1}{2}, k+\frac{1}{2}\right) + CD \\ &\qquad \cdot \left[E_y^n\left(i, j+\frac{1}{2}, k+1\right) - E_y^n\left(i, j-\frac{1}{2}, k\right)\right. \\ &\qquad \left. - E_z^n\left(i, j+1, k+\frac{1}{2}\right) + E_z^n\left(i, j, k-\frac{1}{2}\right)\right], \end{aligned} \tag{2}$$

where

$$\begin{aligned} CA(i, j, k) &= \frac{(1-\sigma(i, j, k)\,\Delta t/2\varepsilon(i, j, k))}{(1+\sigma(i, j, k)\,\Delta t/2\varepsilon(i, j, k))}; \\ CB(i, j, k) &= \frac{\varepsilon_0}{\varepsilon(i, j, k)} + \frac{\sigma(i, j, k)\,\Delta t}{2\varepsilon(i, j, k)}; \\ CD &= \frac{\Delta t}{\delta\sqrt{\varepsilon_0\mu_0}}, \end{aligned} \tag{3}$$

where ε is the relative permittivity, σ is the conductivity of the tissue [S/m], δ is the mesh size, and Δt is the time step. Figure 2 shows the workflow of the FDTD method.

The explicitly iterative process of FDTD algorithm requires initial field values and boundary conditions. The traditional definition of initial field values put all the space field values which are defined as 0, then the field values of two successive time steps are stored as the initial field value of the next step. There are a variety of the boundary conditions, such as Mur absorbing boundary, perfectly matched layer (PML), and uniaxial perfectly matched layer [11] (UMPL). With the improvement of FDTD algorithm these three matching layers have good effect on absorption.

In our experiments, we use uniaxial perfectly matched layer (UPML) as absorbing boundary condition. UPML inherited the PML absorbing layers good absorption characteristics, but the UPML absorbing layer is directly based on Maxwell's equations which is different from PML absorbing layer. This ensures that the form of FDTD algorithm calculation model has good consistency with Maxwell's equations, which makes it easier to understand and program.

3. FDTD Algorithm Based on MPI-OpenMP

3.1. MPI-OpenMP Hybrid Model. MPI programming model is a parallel programming interface for developing standards based messaging, acting on a heterogeneous network environment. It provides a reliable transport mechanism, and it uses security channel to achieve the communication between all the tasks within a process group, data exchange and processing. However, MPI is calculated by parallel interprocess communication, which results in a lower efficiency, the parallel memory overhead, programming problems, and other shortcomings. The OpenMP programming model utilizes fork-join execution mode and shared memory model where a process can be divided into several parallel task execution threads. In a single node, when the main thread is running into OpenMP parallel region, it will produce different thread queue to achieve a parallel effect. However, since OpenMP parallel programming model can only be a single node, computing power of the CPU has been greatly restricted.

MPI-OpenMP programming model is the combination of these two models, which is between multiple machines using MPI distributed memory and each MPI process uses multithreaded OpenMP shared memory model to parallel computing [12]. This model can reduce the number of MPI processes in parallel, thereby reducing the number of messages passing. What is more, OpenMP parallel on each node can save memory overhead. So this parallel hybrid model has certain advantage over a single model.

3.2. The Division of MPI Model. According to the different regional calculation scale, each division way has different characteristics. Common area includes one-dimensional, two-dimensional, and three-dimensional division. In the cube model, each section of the communication direction is all the sane, which means that the communication traffic is proportionate to the number of communication surfaces. Three-dimensional method produces the minimal communication surface. Therefore, we should select the three-dimensional division in the balanced scales calculating. But when the area which you calculated is dominant in one direction or in both directions, the communication traffic is not proportional to the number of communication surfaces. The one-dimensional or two-dimensional division can get better results. For example, long waveguide will be divided one-dimensional along the waveguide direction. In addition wing aircraft is usually divided two-dimensional alone its width direction.

In this paper, the computational model which is similar to the phone box is an extreme rectangular model. We performed a one-dimensional division by the meshing and each process is assigned to one node on average to obtain high efficiency. This division can be reduced communication time between nodes in the rectangular model (Figure 3).

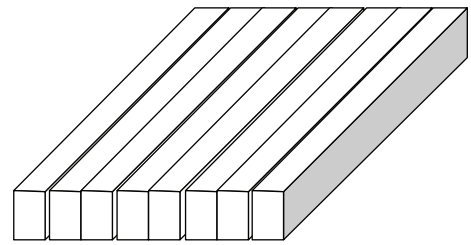


Figure 3: The division of rectangular computational model.

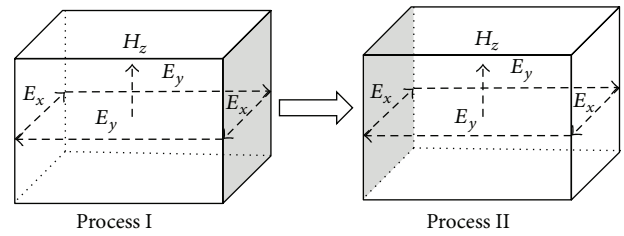


Figure 4: Data communication between processes.

3.3. MPI Parallel Communication Design. As shown in (1), FDTD field value for each step of the algorithm depends only on the value of magnetic field intensity and the circumference value of the grid, which means that it has significantly localized. Therefore, MPI algorithm just calculates the area into a plurality of subregions, and each subregion is allocated MPI node in the mesh to ensure the exchange data in the each time step. Finally, we synchronize data of each node at end of the step. In order to obtain high efficiency of parallel speedup, we partition the process by X direction and put the processes average assigned to each node. This partitioning can reduce the communication between nodes time in the X direction dominant of the rectangular parallelepiped model.

After the calculate area is divided into a number of subregions. The data communication is shown in Figure 4. If we want to calculate H_z which is in process II, we should require the value of electric fields E_x and E_y which is in process I. Then process I transfers the values to process II. H_z can begin calculating when data exchange is completed. Obviously, the data in the other direction can communicate with each other in this way.

3.4. OpenMP Threaded Design and Optimization. The electric field and magnetic field strength only depends on the electromagnetic field data of the former step in the electromagnetic computing. Therefore, it has the natural parallelism. In the multicore computing, the program uses OpenMP model to increase program parallelism.

For better memory utilization, the load balancing may not be very good; therefore, when considering performance optimization, we should have a compromise between the

```
#pragma omp parallel for schedule(static)
for (i = 1; i < ie_tot; i++)
   for (j = 0; j < je_tot; j++)
      for (k = 0; k < ke_tot; k++)
         ...
```

In order to increase OpenMP load balance and get better scalability, we merge the nested loops and reduce from 3 nest layers to 2 layers.

```
_end = je_tot;
ij_end = (ie_tot) * (je_tot);
ij_start = j_end + 1;
#pragma omp parallel for schedule(static)
for (ij = ij_start; ij < ij_end; ij++)
{
   i = ij/j_end;
   j = ij;
   if (!i) continue;
   for (k = 0; k < ke_tot; k++)
      ...
}
```

Algorithm 1

need to optimize memory utilization and load balancing. In OpenMP model, there are four kinds of commonly used scheduling, such as static, dynamic, guided, and runtime. The commonly used scheduling is static where iterations are divided into chunks of size chunk_size while chunks are assigned to threads in the team in round-robin fashion in order of thread number and dynamic where each thread executes a chunk of iterations and then requests another chunk until no chunks remain to be distributed.

From the definitions of the schedule, we can see that static scheduling applies to the situation that all CPU functions are similar due to the complexity of balancing reasons, while the dynamic scheduling applies to the situation of the ability to run large differences between the CPUs. Due to the experimental environment that each CPU computing power is not very different, we use static scheduling so that each thread can average computing tasks [13] (see Algorithm 1).

Its obvious that x, y, and z directions have similar calculation.

3.5. MPI-OpenMP Hybrid Programming. With the increasing scale of operation, the shared storage processor overhead of process limits parallel performance. We can solve this problem by OpenMP and MPI hybrid model where each node is calculated by MPI processes and each node uses the optimal number of threads. Thus the amount of computation is shared by multiple threads, thereby reducing communication overhead and easing the process workload on each node.

When FDTD algorithm based on MPI-OpenMP model allocates a process for each compute node, each node is assigned a process. Passing MPI initialization and calculation parameters initialization, the process starts OpenMP parallel computing. In this phase, the program should update cycle electric field value and the electric field value with adjacent nodes exchanges the date. After the communication is completed, CPUs use OpenMP programming model to update the value of the magnetic field. When the programs achieve the maximum time step, MPI processes and the main procedures will end. As for the border absorption treatment, process calculation is carried out in a single process in every time step. Therefore, it can also improve program parallelism through OpenMP model.

The pseudocode is as shown in Algorithm 2.

4. MPI + GPU Programming Model Design

4.1. FDTD Parallel Algorithm Based on GPU Model. FDTD is the most popular method of computational electromagnetic simulation because of its simple algorithm and high computational efficiency. Figure 5 shows the flow of the three-dimensional FDTD method for a single GPU computation [14]. There are four tasks within each time step for the GPU side in this figure: electric field computation (e_field), uniaxial perfectly matched layers (UPML) computation for electric field (e_upml) as the absorbing boundary condition, magnetic field computation (h_field), and PML for magnetic field (h_upml). All field updates in each time step can be parallelized and are offloaded to the GPU [15].

Our GPU implementation of the FDTD method is based on the C++ code that runs on the ZiQiang4000 high-performance computer clusters of Shanghai University. Since all the magnetic and electric field computations can be vectored and parallelized, these computations are also candidates for GPU computation with a CUDA kernel. The CUDA kernel including e_field, e_pml, h_field, and h_pml follows. And our GPU three-dimensional FDTD program requires no data transfer for field updates because of all computations within each time. The CUDA kernel function [16] would be in Algorithm 3.

4.2. The Optimized Memory of Access Patterns. The kernel code which can be called by the GPU has role similar to

```
MPI_Init(&argc, &argv); //MPI parallel environment initialization
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Get_processor_name(processor_name, &namelen);
MPI_Barrier(MPI_COMM_WORLD);
node.init(x_size/size, y_szie, z_size); //Grid data initialization
UPMLcompute(); //Set up the absorbing boundary
for (int i = 0; i < max_step; i++) //Iterating over the time step
{
#pragma omp parallel for schedule(static)
  for (i = 1; i < ie_tot; i++)
  {
    ...Compute; //Calculating the electric field and magnetic field values
  }
MPI_Send(datasend_right, data_size, MPI_DOUBLE, NodeRank + 1, 1,
MPI_COMM_WORLD); //Send data to the neighboring node
MPI_Recv(datarecv_right, data_size, MPI_DOUBLE, NodeRank + 1, 1, MPI_COMM_WORLD,
&status); //Accept the adjacent node data
MPI_Barrier(MPI_COMM_WORLD);
}
MPI_Finalize();
Out(EHdate); //Output the results
```

Algorithm 2

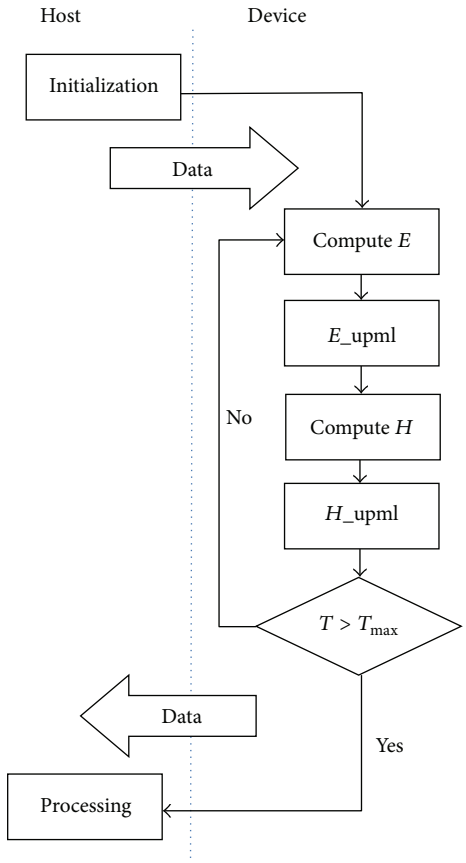


Figure 5: The workflow of the three-dimensional FDTD method for a single GPU computation.

caluH C function. The kernel variable threadIdx, blockIdx, and blockDim can calculate the array indexes ix, iy, and iz. Host-side code then uses caluH ⋘grid, block⋙ (arguments) function to call the kernel code, where grid and block specify the division of thread blocks and the number of threads in the each block.

As the internal memory of GPU, registers and shared memory have a high access speed and a small data latency. Therefore, the data is loaded into the global memory so that shared memory can improve the efficiency of GPU parallel. Consider

shared double new_hx[];
new_hx [hx_size] = _hx [hx_size].

The first line of code is carried in the shared memory array definition. Since the shared memory space is very limited, programmer should choose dynamically allocated arrays or static allocation based on the data size. The second line was to load the data into the global memory shared memory which can increase memory access speed by data sharing. Obviously, other data should also do the same optimization.

4.3. MPI-CUDA Hybrid Programming. In the MPI programming model, a problem will be divided into multiple subtasks so that each process to execute a task. But with the growth of the scale, MPI communication overhead increases at the same time; thereby this will reduce the parallel efficiency. However, the model is just coarse-grained parallelism between the nodes, and computing capacity of CPU is not fully utilized. Then the CUDA model is just to make up the deficiency of this section. The use of multithreaded shared memory

```
__global__ void caluH(arguments)
{
   ix = threadIdx.x + blockIdx.x * blockDim.x;
   iy = threadIdx.y + blockIdx.y * blockDim.y;
   iz = threadIdx.z + blockIdx.z * blockDim.z;
   if (ix < ex_size && iy < ey_size && iz < ez_size)
   {
      int i = ix * ez_size * ey_size + iy * ez_size + iz;
      double tmp = _bx[i];
      _bx[i] = D1 * _bx[i] −D2 * ((_ez[i + ez_size] − _ez[i]) − (_ey[i + 1] − _ey[i]))/delta;
      _hx[i] = D3 * _hx[i] + D4 * (D5 * _bx[i] −D6 * tmp)
   }
}
The host:
dim3 block(BlockDim[0], BlockDim[1], BlockDim[2]);
Grid[0] = nx/blockDim[0];
Grid[1] = ny/blockDim[1];
Grid[2] = nz/blockDim[2];
dim3 grid (Grid[0], Grid[1], Grid[2]);
caluH <<<grid, block>>> (arguments);
```

Algorithm 3

mechanism on the GPU achieves data sharing fine-grained parallelism, which can achieve higher speedup [17].

In order to reduce the computational load of the GPU, we use a single node in a multi-GPU programming model [18], which means that each process is calculated by calling the function cudaSetDevice() to get two GPUs in a single node.

Workflow of three-dimensional FDTD program based MPI-CUDA model is as follows.

(1) Initialize calculation parameter GPU and MPI environment.

(2) Divide the regional calculation and allocate GPU memory and copy initial data to GPU.

(3) Time step for-loop Use MPI model.

   (a) execute electric field computation on GPU

   (b) execute magnetic field computation on GPU.

(5) Process output results.

5. Experiments and Analysis

5.1. Experimental Environment. ZiQiang4000 normal cluster is constituted of 40-unit IBM X240 server blade, which contains two intel E5-2680, 64 G shared memory, and 16 core components in each blade. The GPU computing cluster node contains 11 sets of IBM DX360, which has two intel E5-2680, 64 G shared memory, and plus two Nvidia M2090 GPU constitution. The software environment of the system includes Centos Linux 6.3 operating system, NVCC compiler supports OpenMP guided C/C++ compiler, and MPICH2 parallel environment.

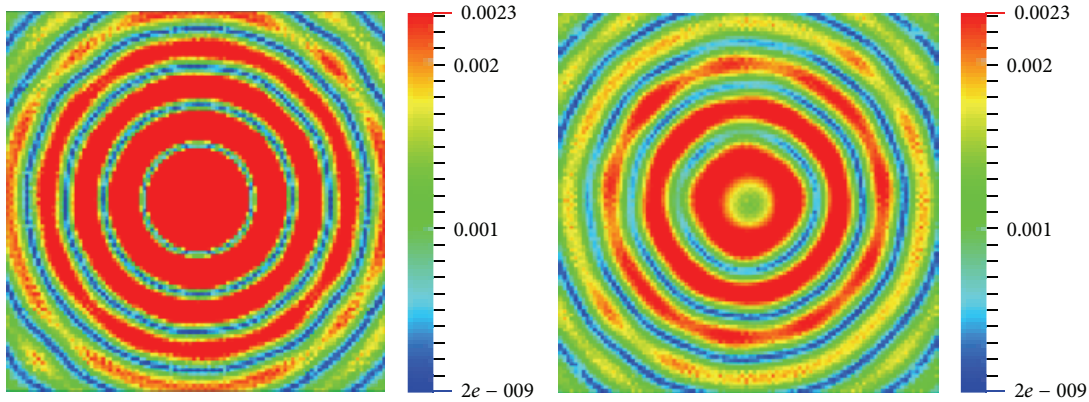
5.2. Model Validation and Analysis. Verifying the correctness of parallel programs is a prerequisite for its performance analysis. In order to verify the correctness of the program and UMPL absorbing boundary absorption effect, this paper makes use of three-dimensional point source radiation to have a situation analysis. we compared the run time between GPUs and CPUs in the case of Gaussian incidence at 900 MHz. What is more, the grid size is 0.02 mm and the calculation scale is of 100×100×100. We use UPML absorbing layer which is set to 10-mesh size and the maximum time step 1000. Figure 6 shows the 50 layers and 65 layers of XY section of the electric field strength calculations at the 1000 steps. As we can see in the figure, electromagnetic waves shape was diffusion, and the further the cross-sectional layer of electric field away from the excitation source, the smaller the value you get. The excitation source is at the center of the field map and gradually spreads to the electromagnetic field borders, which fits the propagation of electromagnetic waves.

Table 1: The result of FDTD algorithm based on MPI.

Process	Overall time(s)	Speedup ratio	Communication area ($\times 10^3$)	Efficiency (%)
1	17146	1	0	100
2	8956	1.9144	32	95.72
4	4632	3.7012	99	92.59
8	2306	7.4353	230	92.9
16	1188	14.4327	482	90.21
32	623	27.5217	1017	86.01
48	422	40.6303	1540	84.64
64	338	50.7221	2066	79.25

5.3. Parallel Analysis. In order to obtain intuitive results, we expand the calculated scale to 1024 × 256 × 128, and the absorbing boundary UMPL mesh size is set to 16. The results were as follows after 1000 steps calculation using MPI model.

Table 1 shows the result of point source radiation calculation time based on the MPI model, such as speedup



(a) The 50 layers (b) The 65 layers

Figure 6: The electric field strength calculations of sectional x-y at 1000 steps.

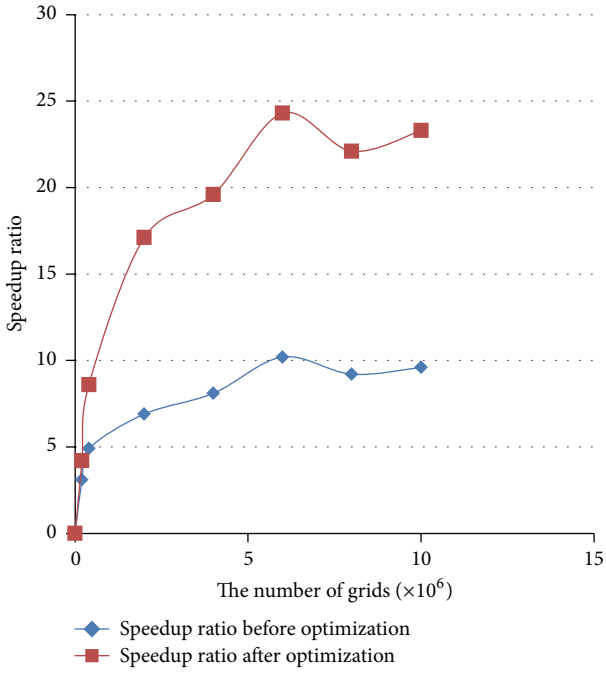


Figure 7: GPU/CPU speed ratio.

and efficiency of the communication area. In this paper, we performed 16 processes on each node. It can be seen from the table that, as the number of processes increases, the computing time significantly reduced for linear speedup. And the electromagnetic radiation program has good scalability and parallelism in a distributed process parallel environment. But with the increase in the number of processes, the parallel efficiency becomes more and more low. This is because the increase in the number of processes leading to the communication area is increasing. It will increase the communication overhead and cause the load between nodes to be not balanced.

From Figure 7, we can see that, when calculating small scale, the acceleration effect of GPU is not obvious. This is due to the number of threads too little to create enough block to make SM concurrent execution, reducing GPU occupancy rate. With the increasing of the calculation size, GPU computing resources gradually are called and the speed of computation is improved. Therefore, when calculating a smaller scale, GPU speed increase is limited, and in the large-scale electromagnetic calculations, GPU-accelerated effects model is very impressive, which is reaching about 9 times the CPU. After optimizing memory access patterns by shared storage, the max speed ratio achieves 23 when the number of grids is 6×10^6. The reason is segmentation of data blocks fit in GPU memory.

MPI-OpenMP hybrid programming model combines two kinds of models, which can take advantage of shared memory and message passing model and improve FDTD algorithm parallelism. FDTD program is mainly calculated by the parameter initialization, the time step, absorbing boundary calculated, updating the electric field, and data processing. However, over 98% of the total time is conducted in two stages which are absorbed electromagnetic boundaries update phase and calculating electromagnetic fields. We can utilize OpenMP programming model to improve program parallelism at this two stages. In this paper, each node runs two processes, and the number of parallel threads is 8. Similarly, MPI-CUDA models use shared memory and message passing mechanism to calculate the electric field value performed on the GPU. Each block is divided into $16 \times 16 \times 16$, and the calculation results are shown in Table 2.

From Columns 2 and 3 of Table 2, it can be seen that pure MPI program is better than MPI-OpenMP model when the calculating cores is small. This is because MPI communication is very small when the number of processes is small, and OpenMP program will be a corresponding increase in the thread overhead. But with the increase of the number of the GPUs, the growth rate of MPI model will decline as MPI process communication and synchronization overhead increases, while MPI-OpenMP program obtains a better parallelism for saving offset cost of the thread. By comparing the two parallel methods, we can see MPI application performance superior to MPI-OpenMP program

Table 2: The result of FDTD algorithm based on all models.

Process	MPI model		MPI-OpenMP model		MPI-CUDA model	
	Overall time(s)	Speedup ratio	Overall time(s)	Speedup ratio	Overall time(s)	Speedup ratio
1	17146	1	17146	1	803	21.3524
4	4632	3.7012	4697	3.6504	217	79.0138
8	2306	7.4353	2376	7.2163	109	157.3028
16	1188	14.4327	1210	14.1702	79	217.0378
32	623	27.5217	615	27.8797	68	252.1471
48	422	40.6303	407	42.7581	61	281.082

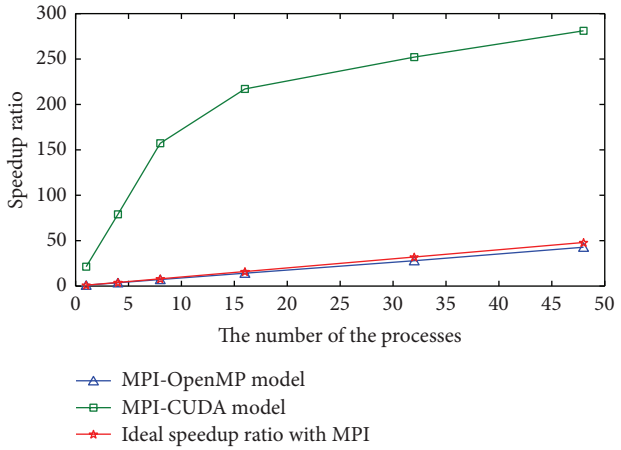


Figure 8: The result of FDTD algorithm based on two hybrid models.

when calculating the cores is less than 32. But with the increase in the number of cores, the advantage of MPI-OpenMP program will gradually be revealed and get better speedup.

From Columns 2 and 4 of Table 2, it can be seen that pure MPI-CUDA program is better than pure MPI model, and it has a linear increase when the number of processes is small. But with the increase in the number of processes, the efficiency of the program decreased very significantly. MPI communication overhead is increased in the proportion of the total time. As the overall calculation region is divided into a plurality of subregions, the size of each process significantly reduced to GPU parallel efficiency. We can find that the speedup of MPI-CUDA programming model is about 20 times the pure MPI program, which effectively improved the parallelism of the program.

Figure 8 shows the result of FDTD algorithm based on two hybrid models. Due to the GPU's high-speed floating-point operations, MPI-CUDA parallel model clearly has a better effect. However, this model is limited by memory and is device-dependent. MPI-OpenMP program can save communication costs and have better scalability. These two kinds of parallel models are suitable for mass-scale electromagnetic calculations which conform the actual electromagnetic calculations. Therefore, the programmer should be selected by different hybrid models according to the different circumstances.

6. Conclusions

With the development of high-performance computing, the new technology provides a better use of space for FDTD algorithm. The various parallel models make hybrid programming model become the mainstream of high-performance computing. In this study, we implemented the three-dimensional FDTD method by MPI-OpenMP model. What is more, we also implemented FDTD method on multi-GPU cluster environment with CUDA and MPI and fulfilled the simulation of three-dimensional numerical electromagnetic radiation. The results indicate that hybrid programming model can better take advantage of distributed shared memory in order to improve parallel performance.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported by the Major Research Plan of NSFC [no. 91330116], the National Science Foundation of China [no. 71203135], Key Project of Science and Technology Commission of Shanghai Municipality [no. 11510500300], and the Research Fund for the Doctoral Program of Higher Education [no. 20113108120022].

References

[1] R. F. Harrington, *Time-Harmonic Electromagnetic Fields*, Mcgraw-Hill Electrical and Electronic Engineering Series, McGraw-Hill, 1961.

[2] M. de Greef, J. Crezee, J. C. van Eijk, R. Pool, and A. Bel, "Accelerated ray tracing for radio-therapy dose calculations on a gpu," *Medical Physics*, vol. 36, no. 9, pp. 4095–4102, 2009.

[3] S. Adams, J. Payne, and R. Boppana, "Finite difference time domain (FDTD) simulations using graphics processors," in *Proceedings of the DoD High Performance Computing Modernization Program Users Group Conference*, pp. 334–338, June 2007.

[4] N. Takada, T. Shimobaba, N. Masuda, and T. Ito, "High-speed FDTD simulation algorithm for GPU with compute unified device architecture," in *Proceedings of the IEEE Antennas and Propagation Society International Symposium (APSURSI '09)*, pp. 1–4, Charleston, SC, USA, June 2009.

[5] C. Guiffaut and K. Mahdjoubi, "A parallel FDTD algorithm using the MPI library," *IEEE Antennas and Propagation Magazine*, vol. 43, no. 2, pp. 94–103, 2001.

[6] Z. Yu, D. Wei, and L. Changhong, "Analysis of parallel performance of MPI based parallel FDTD on PC clusters," in *Proceedings of the Asia-Pacific Conference Proceedings and Microwave Conference Proceedings (APMC '05)*, vol. 4, p. 3, IEEE, December 2005.

[7] P. Micikevicius, "3D finite difference computation on GPUs using CUDA," in *Proceedings of the 2nd Workshop on General Purpose Processing on Graphics Processing Units (GPGPU '09)*, pp. 79–84, ACM, 2009.

[8] T. Nagaoka and S. Watanabe, "A GPU-based calculation using the three-dimensional FDTD method for electromagnetic field analysis," in *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC '10)*, pp. 327–330, IEEE, September 2010.

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

[10] W. Yu, "A novel hardware acceleration technique for high performance parallel fdtd method," in *Proceedings of the 2nd IEEE International Conference on Microwave Technology and Computational Electromagnetics (ICMTCE '11)*, pp. 441–444, May 2011.

[11] S. D. Gedney, "An anisotropie perfectly matched layer-absorbing medium for the truncation of FDTD lattices," *IEEE Transactions on Antennas and Propagation*, vol. 44, no. 12, pp. 1630–1639, 1996.

[12] M. F. Su, I. El-Kady, D. A. Bader, and S.-Y. Lin, "A novel FDTD application featuring OpenMP-MPI hybrid parallelization," in *Proceedings of the International Conference on Parallel Processing (ICPP '04)*, pp. 373–379, August 2004.

[13] J. Li, J. Shu, Y. Chen, D. Wang, and W. Zheng, "Analysis of factors affecting execution performance of openMP programs," *Tsinghua Science & Technology*, vol. 10, no. 3, pp. 304–308, 2005.

[14] Nvidia Corporation Technical Staff, *Nvidia Cuda C Programming Guide*, Chapter 12, NVIDIA Corporation, 2012.

[15] T. Nagaoka and S. Watanabe, "Multi-GPU accelerated three-dimensional FDTD method for electromagnetic simulation," in *Proceedings of the 33rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS '11)*, pp. 401–404, IEEE, September 2011.

[16] D. S. Cai, Y. Li, K.-I. Nishikawa, C. Xiao, and X. Yan, "Three-dimensional electromagnetic particle-in-cell code using high performance fortran on PC cluster," in *High Performance Computing*, vol. 2327 of *Lecture Notes in Computer Science*, pp. 515–525, Springer, Berlin, Germany, 2002.

[17] V. Demir and A. Z. Elsherbeni, "Programming finite-difference time-domain for graphics processor units using compute unified device architecture," in *Proceedings of the IEEE Antennas and Propagation Society International Symposium (APSURSI '10)*, pp. 1–4, Toronto, Canada, July 2010.

[18] D. A. Jacobsen, J. C. Thibault, and I. Senocak, "An MPI-CUDA implementation for massively parallel incompressible flow computations on multi-GPU clusters," in *Proceedings of the 48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, vol. 16, January 2010.