A High Performance Parallel FDTD Based on Winsock and Multi-Threading on a PC-Cluster

X. Duan [1], X. Chen [1], K. Huang [1], and H. Zhou [2]

[1] College of Electronics and Information Engineering
Sichuan University, Chengdu, 610064, P. R. China
shawnduan@gmail.com, xingcsc@yahoo.com.cn, kmhuang@scu.edu.cn

[2] Institute of Applied Physics and Computational Mathematics of Beijing
Beijing, 100094, P. R. China
zhou-haijing@vip.sina.com

Abstract - Parallel technology is a powerful tool to provide the necessary computing power and memory resources for the FDTD method to simulate electrically-large and complex structures. In this paper, a high performance parallel FDTD is developed for multi-core cluster systems. It employs Winsock to achieve efficient inter-process communication as well as multi-threading to make full use of the hardware resources of multi-core processors on a PC-cluster. Key steps for parallel FDTD such as synchronization, data exchange, load balancing, etc., are investigated. An experiment simulating the scattering of an incident electromagnetic wave form of a computer case is presented which shows that the proposed parallel FDTD achieved speedup of 25.1 and parallel efficiency of 83.7% when 10 processors with 30 cores are utilized, and outperforms traditional parallel FDTD based on MPI or MPI-OpenMP, which gained speedup of 22.9, 24.9 and parallel efficiency of 76.3%, 83.1% respectively under the same circumstances.

Index Terms ─ FDTD, multi-threading, parallel computation, PC cluster, Winsock.

I. INTRODUCTION

As one of the most popular numerical methods, finite-difference time-domain (FDTD) has been widely used to solve various electromagnetic problems [1, 2]. However, the implementation of FDTD for simulating electrically-large and complex structures requires intensive computation and large amounts of memory resources, which is not possessed by a single machine. A highly efficient solution is to implement the FDTD algorithm in a parallel computer system, such as a PC cluster [3, 4, 5].

The FDTD method is conducive to parallel computation due to its structured mesh, regular data structures, and localized calculation [3, 6]. A common method to parallelize the FDTD is to divide the computation domain into many sub-domains that are calculated in different nodes of a cluster, and because the workload of each sub-domain is far less than that of the whole computation space, the memory, and time consumption is greatly reduced [4, 5, 7, 8]. Up till now, message passing interface (MPI), a library specification for message-passing [9, 10], is by far the most popular parallel programming environment for the FDTD to realize operations such as data exchange, synchronization, and etc. [3, 4, 6, 7, 8].

Today's PC clusters have employed multi-core processors, which integrate multiple execution cores on the same chip and thereby introduce a new level of parallelization. Many of the previous researches [3, 4, 5, 6, 7, 8] on the parallel FDTD have not taken multi-core processors into consideration. To take advantage of the computing capability of multi-core processors, it is necessary

for a parallel computation to carefully take both the intranode communication and the internode communication into account. Some implementations of MPI have made efforts to it. For instance, MPICH2-1.3.1 offers multi-core support by integrating a low-level communication subsystem called Nemesis to minimize the overhead for intranode communication by using lock-free queues in the processes and other optimizations such as a fastbox mechanism to bypass the queues [11,12]. However, in comparison with MPI that uses messages to perform intranode communication, creating threads to make data accessible through shared memory is a more natural method and supports greater bandwidth [13, 14].

A few studies [15,16] utilize open multi-processing (OpenMP), a shared memory parallel programming interface [17, 18], together with MPI to enhance shared-memory performance. However, a straight-forward integration of OpenMP constructs into the MPI program often does not give good speedup results [19]. Optimization techniques like minimizing OpenMP parallel overhead, aggregating messages, CPU affinity and cache-line alignment are usually necessary for a better performance. Thus, it requires programmers to have relatively substantial experience in tuning an OpenMP code.

In this work, a novel parallel FDTD algorithm based on Winsock and multi-threading is proposed. To fully exploit the computation capacity of a PC cluster with both multi-processor and multi-core features, it utilizes the efficient, low-level Winsock programming rather than MPI to realize the message passing between processors, creates threads, and maps them to cores by using Win32 thread application program interface (API), so the sub-domain computation is carried out in each core. Data exchange between threads is performed by shared variables being directly written by one thread and read by another. Meanwhile in this parallel FDTD, only magnetic field values on the sub-domains' interfaces need to be transmitted by using an overlapping scheme, an efficient synchronization mechanism is used to impose constraints on the execution order of threads, and the different workload of various cells (such as ordinary and perfectly matched layer (PML) [20]) are taken into consideration in the domain decomposition phase to achieve better load balancing.

A numerical experiment has been conducted to estimate the efficiency of the proposed FDTD parallelization strategy. In the experiment, three parallel FDTD codes based on the proposed method, MPI, and MPI-OpenMP, respectively, are developed to simulate the same electromagnetic model in parallel on a PC cluster, and their run time, speedup, and parallel efficiency are compared and analyzed.

The remainder of this paper is organized as follows: Section II briefly introduces the FDTD method, while Section III describes the essential elements of the parallel FDTD based on Winsock and multi-threading. The experimental results are presented in Section IV and finally conclusions in Section V.

II. A BRIEF INTRODUCTION TO THE FDTD METHOD

Since Kane S. Yee's paper in 1966 [1], the FDTD has developed into a widely-used numerical simulation method. In Yee's difference scheme, the computation domain is discretized to space grids in Cartesian Coordinates. The FDTD update equations are then obtained by discretizing Maxwell's two curl equations using central-difference approximations to the space and time partial derivatives. The updated equations for Ex and Hx [6] are as follows:

$$E_x^{n+1}(i+\frac{1}{2},j,k)=\frac{\varepsilon_x-0.5\Delta t\sigma_x}{\varepsilon_x+0.5\Delta t\sigma_x}E_x^{n}(i+\frac{1}{2},j,k)+\frac{\Delta t}{\varepsilon_x+0.5\Delta t\sigma_x}\left[\begin{array}{l}\dfrac{H_z^{n+\frac{1}{2}}(i+\frac{1}{2},j+\frac{1}{2},k)-H_z^{n+\frac{1}{2}}(i+\frac{1}{2},j-\frac{1}{2},k)}{\Delta y}\\ -\dfrac{H_y^{n+\frac{1}{2}}(i+\frac{1}{2},j,k+\frac{1}{2})-H_y^{n+\frac{1}{2}}(i+\frac{1}{2},j,k-\frac{1}{2})}{\Delta z}\end{array}\right], \quad (1)$$

$$H_x^{n+\frac{1}{2}}(i,j+\frac{1}{2},k+\frac{1}{2})=\frac{\mu_x-0.5\Delta t\sigma_{Mx}}{\mu_x+0.5\Delta t\sigma_{Mx}}H_x^{n-\frac{1}{2}}(i,j+\frac{1}{2},k+\frac{1}{2})+\frac{\Delta t}{\mu_x+0.5\Delta t\sigma_{Mx}}\left[\begin{array}{l}\dfrac{E_y^{n}(i,j+\frac{1}{2},k+1)-E_y^{n}(i,j+\frac{1}{2},k)}{\Delta z}\\ -\dfrac{E_z^{n}(i,j+1,k+\frac{1}{2})-E_z^{n}(i,j,k+\frac{1}{2})}{\Delta y}\end{array}\right], \quad (2)$$

thus only adjacent cells are needed for the computation. Similar equations can be written for the other electric and magnetic field components.

III. A PARALLEL FDTD BASED ON WINSOCK AND MULTI-THREADING

A. Parallel programming model

In a PC cluster, processors are in a distributed address space while cores in the same processor are in a shared address space. The address space has a significant influence on the data exchange [21]. To accommodate the hybrid address space of a PC cluster, the parallel FDTD algorithm in this work employs a two-level programming model, as shown in Fig. 1. The upper level consists of processes created by the main console and assigned in processors while the bottom level is composed of threads created by each process for cores in a processor.

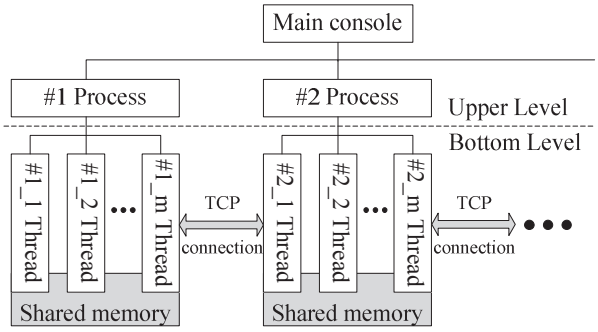

Fig. 1. The two-level programming model for the parallel FDTD.

This work adopts Winsock rather than MPI to realize an inter-process message passing. Winsock defines a standard interface between a Windows TCP/IP client application and the underlying TCP/IP protocol stack [22, 23]. As shown in Fig. 2, a connection using the transmission control protocol (TCP) is established to a specific socket separately on both sides. Once the connection is set up, they can pass messages by calling *send*() and *recv*() .

Explicit threading is employed in the bottom level of the programming model, directly splitting up the computation. The key steps using the Win32 thread API [13] to multi-thread are as follows:

a. A process starts with a single thread of execution. This is called the main thread.
b. This is followed by *_beginthreadex*() being used to create new sub-threads. Those sub-threads can now start their own computation tasks.
c. During the computation, shared variables are used and data exchange is performed by write or read accesses of the threads. Meanwhile, synchronization both coordinates thread execution and manages shared data.
d. When the computation task is finished, each sub-thread calls *_endthreadex*() to terminate.

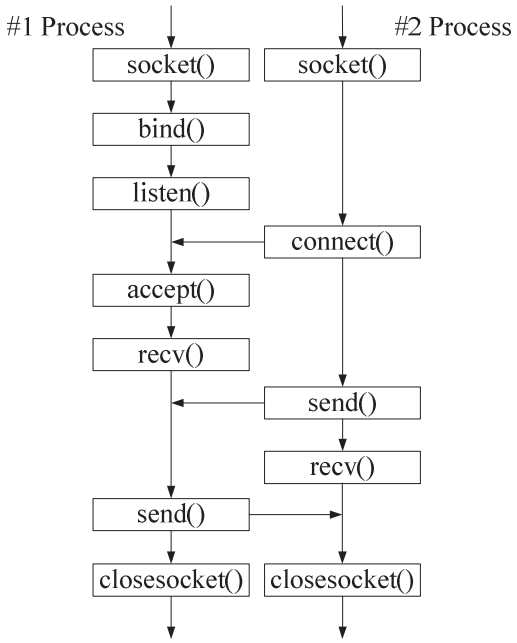

Fig. 2. The flow chart of Winsock programming.

For a better efficiency, a fixed mapping from the threads to the execution cores is employed by calling *SetThreadAffinityMask*() . This prevents threads from migrating to other cores during program execution.

B. Domain decomposition

Figure 3 shows the spatial parallelism used in this work that divides the whole computation space into sub-domains. Each sub-domain is assigned to one process for parallel computation. In each sub-domain, additional pages, called expand pages, are introduced to store field values from neighbors' interface for data exchange. Every process creates threads according to the

number of execution cores employed in the parallel computation, splits a sub-domain further into smaller sub-domains, and then assigns them to each thread. For threads in the same process, field values stored in shared variables are mutually directly accessible, thus eliminating the need for expanded pages.

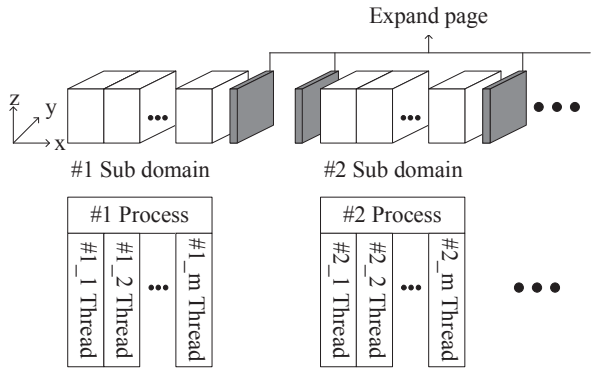

Fig. 3. Domain decomposition.

C. Synchronization

At each time step of FDTD, the update of the electric and magnetic fields are executed sequentially. Execution of threads in the wrong order may produce unwanted outputs. Figure 4 illustrates the synchronization mechanism adopted in this work, where N is the number of sub-threads, and *count* is defined as a volatile integer. *InterlockedIncrement*64(&*count*) increases by 1 the value of variable *count* as an atomic operation, and *InterlockedExchange*64(&*count*, 0) sets variable *count* to 0 as an atomic operation as well.

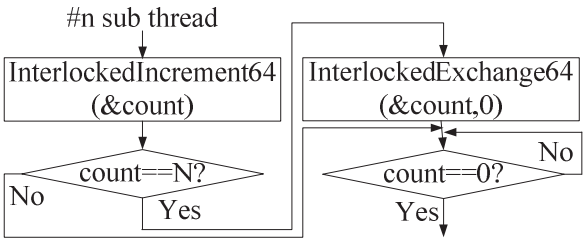

Fig. 4. The synchronization mechanism.

D. Data exchange

According to the FDTD updated equations, for cells on the interfaces between two adjacent sub-domains, the field data on the neighboring threads are required, necessitating the execution of data exchange. For threads on the same processor, data exchange is done by shared E and H variables directly written by one thread and read by another. As for threads on different processors, messages containing E and H values are passed by explicitly calling the Winsock API: *send*() and *recv*() [23].

Data exchange is one of the main factors affecting the parallel efficiency, along with synchronization and load balancing. This work utilizes an overlapping scheme [7], i.e. the interface of adjacent sub-domains assigned to two processes is overlapped, on which the electric field components are updated in both neighboring threads created by separate processes, shown in Fig. 5. Although the electric field components on the interface are calculated twice, only magnetic field components are transmitted at each time step. However, the overlapping scheme does not apply to the threads created in the same process due to the use of shared variables; otherwise data race occurs, leading to calculation errors.

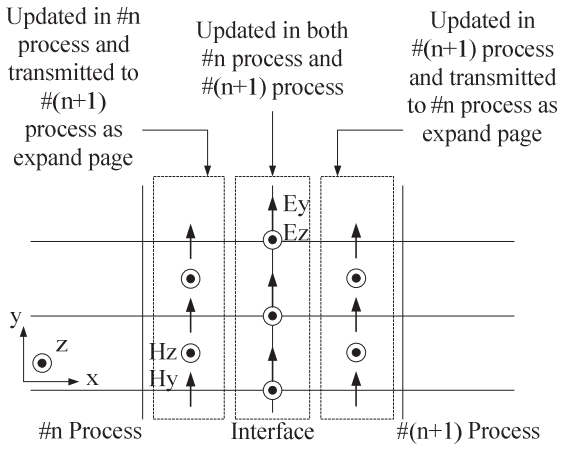

Fig. 5. The field exchange configuration employing the overlapping scheme.

E. Load balancing

Load balancing is a critical issue for a parallel computation to achieve high efficiency [24]. For the parallel FDTD, a good load balancing strategy should suitably divide computation space so that every thread is equal in actual execution duration between any two succeeding operations of synchronization, avoiding idle conditions owing

to mutual waiting. One of crucial factors affecting load balancing is various cells such as ordinary cells and PML cells, due to the different workloads that they possess. However, no specific rule exists for evaluating this factor in the domain decomposition, just experience acquired from repeated experiments. Our experiments show that a PML cell requires approximately 2.6 times more computing time than that of an ordinary cell. So a PML cell is evaluated as 2.6 ordinary cells when the whole space is divided into sub-domains.

F. The thread execution

A thread carries out the actual FDTD calculation, as illustrated in Fig. 6. Electric and magnetic fields are updated sequentially, each followed by a synchronization operation. If a thread possesses a sub-domain interface between neighboring nodes, exchange of the magnetic field components with neighboring processes, through sending and receiving operations, is performed upon update of the magnetic field.

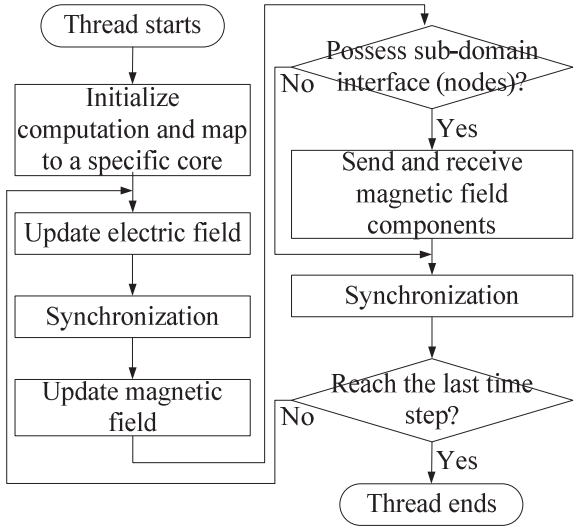

Fig. 6. The flow chart of the thread execution.

optimizations for multi-core processors. The MPI based code creates a process binding to each execution core, and the data exchange between two cores is realized by calling MPI_Sendrecv(), just like that between two processors. Channel Nemesis is chosen to optimize intranode and internode communication. The MPI-OpenMP based code is similar to the pure MPI code. It creates a process in each processor, and employs OpenMP to automatically parallelize to the computing loops [15, 19], shown in Fig. 7. The *parallel* pragma is hoisted outside the loop of time steps to minimize threading overhead. When synchronization is not a necessity at the end of a parallel loop, *nowait* is specified with the *for* directive. Thread binding is achieved by calling $SetThreadAffinityMask()$.

```
Do initialization work;
#pragma omp parallel private (…)
{
for timestep=0 to timestep_max
   {
   …
   #pragma omp for private (…) nowait
   computation loops;
   …
   #pragma omp master
   {
   MPI_Sendrecv();
   }
   #pragma omp barrier
   }
}
```

Fig. 7. MPI-OpenMP algorithm.

IV. RESULTS AND ANALYSIS

To estimate the parallel performance of the proposed method, three parallel FDTD codes based on the proposed method, the MPI library, and MPI-OpenMP, respectively, are developed in Visual Studio 2010. The implementation of MPI used in this paper is MPICH2-1.3.1, the latest version supporting the MPI 2.0 standard and

As depicted in Fig. 8, an electromagnetic plane wave with a frequency of 1GHz propagates to a computer case, and the three parallel FDTD codes are employed to simulate the scattered field of the computer case. For this scattered field problem, its whole computation space is discreted into $720*170*330$ cells along x, y, and z axis, respectively, with $dx = dy = dz = 0.001m$.

The experiment is carried out on a PC cluster comprising Intel Q6600 processors interconnected by a fast 1000Mb/s Ethernet. Each processor possesses 4 cores; in the experiment, one core is left to process system work and up to three cores can be used for the parallel FDTD computation.

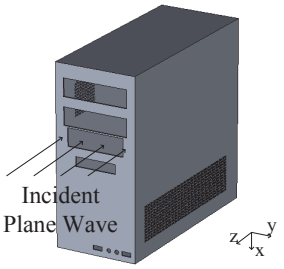

Fig. 8. The computation model.

The scattered field simulated by the three parallel FDTD codes agrees well with the simulation by computer simulation technology (CST) Microwave Studio. Table 1 demonstrates the tested run time and memory consumption of the three parallel FDTD codes, which drop quickly upon introduction of more threads/processes. One can see that the proposed method always costs less run time in comparison with both the MPI based and the MPI-OpenMP based parallel FDTD, and three codes consume the same amount of memory.

Table 1: Run time (in seconds) and memory consumption (in MB)

Number of threads/processes		1	6	12	18	24	30
Run time	The proposed method	7092	1257	648	444	344	282
	MPI	7092	1292	661	468	377	309
	MPI-OpenMP	7092	1268	655	449	346	284
Memory Consumption		2950	1500	775	412	231	140

Figure 9 gives the measured speedup and parallel efficiency for the three parallel FDTD codes. Here, speedup is the ratio of sequential implementation execution time and parallel

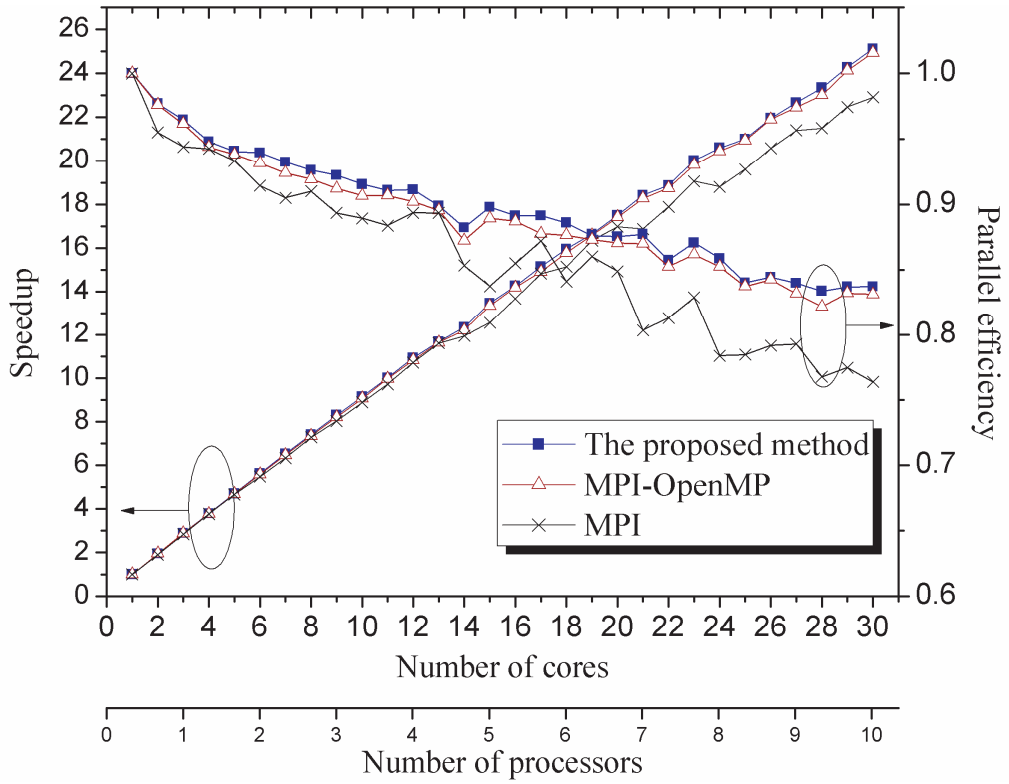

Fig. 9. Speedup and parallel efficiency.

execution time, while parallel efficiency is equal to speedup divided by the number of cores employed. One can observe that the speedup and parallel efficiency of the proposed method is considerably higher than that of the MPI based FDTD and the discrepancy becomes even larger with the increase of the number of cores. Compared with the MPI-OpenMP version of FDTD, the proposed method's speedup and parallel efficiency is slightly bigger, but it is worth noting that the former is more difficult to optimize because it needs a careful fine tuning to achieve good performances. When 30 cores are utilized, the proposed method's speedup rises to 25.1 with parallel efficiency of 83.7%, while the MPI application's speedup is 22.9 with parallel efficiency of 76.3% and the MPI-OpenMP application's speedup is 24.9 with parallel efficiency of 83.1%.

V. CONCLUSION

Today's PC clusters have improved remarkably in their computation ability by utilizing powerful new hardware, such as processors integrated with multiple cores. Parallel FDTD should be carefully designed to make full use of PC clusters' multi-processor and multi-core features. This paper presents a high performance parallel FDTD method using Winsock and multi-threading to parallelize the FDTD computation. Threads are created to do the sub-domain computation in each execution core. Neighboring threads that are on the same nodes directly access each others' memory during data exchange. Winsock sets up TCP connections between neighboring threads that run on different nodes to pass messages. Meanwhile, some techniques have been utilized to improve the parallel FDTD's synchronization, data exchange, load balancing, and etc.

A numerical experiment of simulating a scattered field problem on a PC cluster with multi-core processors has been conducted to verify the validation and efficiency of the proposed method. In the experiment, the proposed method exhibits higher speed up and parallel efficiency than the FDTD using MPI or MPI-OpenMP. Moreover, the proposed method is natural and easy to implement for a good performance due to its explicit and direct threading. Those features make the method widely usable in numerical simulation problems.

ACKNOWLEDGEMENT

This work was supported by NCET of China (No. NCET-08-0369) and the National Natural Science Foundation of China (No. 10876020).

REFERENCES

[1] K. Yee, "Numerical Solution of Initial Boundary Value Problems Involving Maxwell's Equations in Isotropic Media," *IEEE Transactions on Antennas and Propagation*, vol. 14, no.3, pp. 302-307, 1966.

[2] A. Taflove, *Computational Electromagnetics: The Finite-Difference Time-Domain Method*, Artech House, Norwood, 2000.

[3] 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.

[4] G. A. Schiavone, I. Codreanu, R. Palaniappan, and P. Wahid, "FDTD Speedups Obtained in Distributed Computing on a Linux Workstation Cluster," *Antennas and Propagation Society International Symposium*, vol. 3, pp. 1336-1339, 2000.

[5] V. Varadarajan and R. Mittra, "Finite-Difference Time-Domain (FDTD) Analysis using Distributed Computing," *IEEE Microwave and Guided Wave Letters*, vol. 4, no.5, pp. 144-145, 1994.

[6] W. Yu, R. Mittra, T. Su, Y. Liu, and X. Yang, *Parallel Finite-Difference Time-Domain Method*, Artech House, 2006.

[7] W. Yu, Y. Liu, T. Su, N.-T. Hunag, and R. Mittra, "A Robust Parallel Conformal Finite-Difference Time-Domain Processing Package using the MPI Library," *IEEE Antennas and Propagation Magazine*, vol. 47, no. 3, pp. 39-59, 2005.

[8] W. Yu, M. R. Hashemi, R. Mittra, D. N. de Araujo, M. Cases, N. Pham, E. Matoglu, P. Patel, and B. Herrman, "Massively Parallel Conformal FDTD on a BlueGene Supercomputer," *IEEE Transactions on Advanced Packaging*, vol. 30, no. 2, pp. 335-341, 2007.

[9] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI: The Complete Reference,* The MIT Press, 1996.

[10] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface, second edition*, The MIT Press, 1999.

[11] D. Buntinas, G. Mercier, and W. Gropp, "Implementation and Shared-Memory Evaluation of MPICH2 over the Nemesis Communication Subsystem," *Proc. of the 13th European PVM/MPI Users' Group Meeting (Euro PVM/MPI 2006)*, September 2006.

[12] D. Buntinas, G. Mercier, and W. Gropp, "Design and Evaluation of Nemesis, A Scalable Low-Latency Message-Passing Communication Subsystem," *Proceedings of International Symposium on Cluster Computing and the Grid 2006 (CCGRID '06)*, 2006.

[13] S. Akhter and J. Roberts, *Multi-Core Programming: Increasing Performance through Software Multi-threading*, Intel Press, 2006.

[14] F. Cappello and D. Etiemble, "MPI Versus MPI+OpenMP on the IBM SP for the NAS Benchmarks," *Supercomputing ACM/IEEE 2000 Conference*, 2000.

[15] M. F. Su, I. El-Kady, D. A. Bader, and S.-Y. Lin, "A Novel FDTD Application Featuring OpenMP-MPI Hybrid Parallelization," *Parallel Processing, 2004 International Conference*, pp. 373-379, 2004.

[16] R. Rosenberg, G. Norton, J. C. Novarini, W. Aderson, and M. Lanzagorta, "Modeling Pulse Propagation and Scattering in a Dispersive Medium: Performance of MPI/OpenMP Hybrid Code," *SC 2006 Conference, Proceeding of the 2006 ACM/IEEE*, pp. 47-47, 2006.

[17] B. Chapman, G. Jost, and R. Van Der Pas, *Using OpenMP: Portable Shared Memory Parallel Programming*, The MIT Press, 2008.

[18] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, and R. Menon, *Parallel Programming in OpenMP*, Academic Press, 2001.

[19] A. Rane and D. Stanzione, "Experiences in Tuning Performance of Hybrid MPI/OpenMP Applications on Quad-Core Systems," *Proc. of 10th LCI Int'l Conference on High-Performance Clustered Computing*, 2009.

[20] J. Berenger, "A Perfectly Matched Layer Medium for the Absorption of Electromagnetic Waves," *J. Comput.*, vol. 114, 1994, pp. 185-200.

[21] T. Rauber and G. Rünger, *Parallel Programming for Multicore and Cluster Systems*, Springer, 2010.

[22] A. Jones and J. Ohlund, *Network Programming for Microsoft Windows*, Microsoft Press, 2002.

[23] B. Quinn and D. Shute, *Windows Sockets Network Programming*, Addison-Wesley Professional, 2009.

[24] J. Watts and S. Taylor, "A Practical Approach to Dynamic Load Balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 3, pp. 235-248, 1998.

Xin Duan was born on May 23, 1986 in Sichuan, China. He received his B.S. degree in Information and Communication Engineering in 2008 from Sichuan University. He is now working toward his M.S. degree in Electromagnetics and Microwave in Sichuan University. His research is mainly focused on computational electromagnetics and antenna design.

Xing Chen received his M.S. degree in Radio Physics in 1999 and the Ph.D. degrees in Biomedical Engineering in 2004, both from Sichuan University, China. He joined the teaching staff in 1991, and is now a Professor in the College of Electronics and Information Engineering of Sichuan University. His main research interests are in the fields of antenna design, optimization algorithm, numerical methods, and parallel computation. He is a senior member of the Chinese Institute of Electronics.

Kama Huang received his M.S. degree in 1988 and his Ph.D. degree in 1991 in Microwave Theory and Technology both from the University of Electronic Science and Technology, China. He has been a professor of the College of Electronics and Information Engineering of Sichuan University, China since 1994, and the director of the College since 1997. In 1996, 1997, 1999, and 2001, he was a visiting scientist at the Scientific Research Center "Vidhuk" in the Ukraine, Institute of Biophysics CNR in Italy, Technical University Vienna in Austria, and Clemson University in USA, respectively. At these institutions, he cooperated with the scientists to study the interaction between electromagnetic fields and complex media in biological structure and reaction systems. He has published over one hundred papers.

Haijing Zhou was born in Beijing in 1970. He received his B.S., M.S., and Ph.D. degrees in Microwave engineering in 1991, 1994, and 1997, respectively, from UESTC (University of Electronics Science and Technology of China). Since 1998, he has been working at IAPCM (Institute of Applied Physics and Computational Mathematics of Beijing), as an Associated Professor (1999) and a Professor (2005), where his research is mainly in the areas of high power microwave, ultra-wideband electromagnetics, and computational electromagnetics. His current interests include classical electromagnetic field theory, transient electromagnetics, computational electromagnetics, microwave technology, antenna technology, and wave propagation, etc.