A Novel Enhancing Technique for Parallel FDTD Method using Processor Affinity and NUMA Policy

Lei Zhao [1,2], Geng Chen [1], and Wenhua Yu [3]

[1] Center for Computational Science and Engineering, School of Mathematical Sciences
Jiangsu Normal University, Xuzhou, China
lzhaomax@163.com, gengchn@163.com

[2] State Key Laboratory of Millimeter Waves
Southeast University, Nanjing, China

[3] 2COMU, State College, PA 16803, USA
wenyu@2comu.com

Abstract ─ The traditional multiple CPUs mounted on one node in a high performance cluster is based on Symmetric Multi-Processing (SMP) architecture. The memory bandwidth is a major bottleneck in the high performance computing. Recently, Intel and AMD companies developed the (Non-uniform Memory Access (NUMA) architecture for the multi-CPU server that is an important extension of the SMP computer. In the NUMA architecture server, each CPU has its own memory and can also be access to the memory located the nearby of other CPUs through the onboard network. For a parallel code, we can allocate the data for each CPU inside its local memory to accelerate the memory access. In this paper, we investigate a way how to achieve the high performance of parallel FDTD code on a computer cluster that includes 21 nodes with 42 CPU and 168 cores. Numerical experiments have demonstrated that different job binding schemes can significantly affect the performance of parallel FDTD code.

Index Terms – NUMA, parallel FDTD, processor affinity, SMP.

I. INTRODUCTION

A high performance cluster has become a popular hardware platform today for the computational electromagnetic methods to solve the electrically large problems. In three popular computational electromagnetic methods, FDTD method [1] is parallel in nature, and hence, has high parallel performance than method of moment (MoM) [2], finite element method [3] due to a parallel FDTD method only requires the field exchange on the interface between the adjacent neighboring subdomains. For simulating electromagnetic (EM) problem from electrically-large and complex structures with FDTD method, parallel technology is a powerful tool to provide the necessary computing power and memory resources [4-8]. The parallel performance of FDTD code depends on not only the way how we develop the parallel FDTD code and problem type, but also on the hardware platform such as CPU type, network system, Input/Ouput (I/O) system, and the operating system as well. In this paper, the parallel FDTD code is developed based on the literature [7, 8] that uses the combination of Open Multiple Processing (OpenMP) [9] and Message passing Interface (MPI) library [10]. OpenMP is developed for the efficient use of multi-core processors and the MPI library is developed to use the distributed resource.

For the same parallel FDTD code, we investigate the performance of parallel FDTD code on an intermediate cluster that includes 21 nodes (42 CPUs with 168 cores) when we use the different binding techniques and the running environment variables. In all the numerical

Submitted On: Jan. 29, 2012
Accepted On: June 24, 2012

experiments, we do not modify the parallel FDTD code and keep the same hardware platform and operating system as well. Each node in the cluster includes two Intel Xeon X5520 2.67GHz processor, which support the NUMA architecture. Namely, it allows us to allocate the data for each CPU in one node to its local memory. If one job unit is assigned to one node, the communication between two CPUs in one node is realized through OpenMP. And the communication between the nodes is realized by the MPI functions. Otherwise, if one job unit is assigned to one core, all the communication between the cores is realized by the MPI functions. Furthermore, if one job unit is assigned to each CPU, the communication between the CPUs is realized by the MPI function but the communication between the cores inside each CPU is realized through OpenMP.

To achieve a good performance of the parallel FDTD code on the high performance cluster, the NUMA policy is used to extend the memory bandwidth and reduce memory access time by allocating the data for each CPU in its own local memory. The advantage of NUMA architecture is obvious from the numerical experiments. We also investigate the effect of processor affinity [11] on the parallel FDTD code performance by binding each rank to the node, CPU, or core. In this paper, all the test examples are carried out by using GEMS software [12].

II. THEORY AND METHOD

Both the electric and magnetic field updates in the FDTD method only require field information from their nearest neighboring cells, which requires much less communication information than other methods that require the 3-D communication data. Hence, the parallel FDTD method gives much less burden on the network system, and in turn, it generates the higher parallel efficiency. To achieve the better parallel performance, we install two sets of network systems in a regular cluster, one of them is design the data communication during the simulation and usually is fast. And the second one is designed the cluster management, namely, it allows simultaneously to check the cluster status without interrupting the data communication.

In Yee's scheme [1], the computational domain is discretized by using a rectangular grid. The electric fields are located along the edges of the electric elements, while the magnetic fields are sampled at the centers of the electric element surfaces and are oriented normal to these surfaces, this being consistent with the duality property of the electric and magnetic fields in Maxwell's equations, as shown in Fig. 1.

If the computational domain is broken into two subdomains, and the interface coincides with the FDTD mesh. The electric fields on the interface can be counted into either subdomain 1 or 2. For instance, if it is belong to the subdomain 1, we need to borrow the magnetic field H_z^2 from the subdomain 2 when we calculate the electric field $E_y^{\text{interface}}$ on the interface.

$$E_y^{\text{interface, n+1}} = E_y^{\text{interface, n}} + \frac{\Delta t}{\varepsilon_y} \cdot \left[\frac{H_x^{2,n+1/2} - H_x^{1,n+1/2}}{\Delta z} - \frac{H_z^{2,n+1/2} - H_z^{1,n+1/2}}{\Delta x} \right]. \quad (1)$$

We need to borrow the electric field $E_y^{\text{interface}}$ on the interface when we calculate the magnetic field H_z^2 in the subdomain 2:

$$H_z^{2,n+1/2} = H_z^{2,n-1/2} + \frac{\Delta t}{\mu_z} \left[\frac{E_x^{2,n} - E_x^1}{\Delta y} - \frac{E_y^{2,n} - E_y^{1,\text{interface}}}{\Delta x} \right]. \quad (2)$$

In the MPI library, the communication of the electric and magnetic fields between the subdomains 1 and 2 are realized by the MPI functions MPI_Send and MPI_Recv. The information is changed through the high performance network system. OpenMP is based on the fine grid technique in the shared memory system, and its information exchange is through a shared memory. In the optimization of the parallel FDTD code, we need to achieve a balance between the minimum area of interface and performance of network. Internal consistency should be maintained

Uniform Memory Access (UMA) is a shared memory architecture used in parallel computers, as shown in Fig. 2. In the UMA model, all the processors share the physical memory uniformly, and access time to a memory location is independent of which processor makes the request or which memory chip contains the transferred

data. The UMA model is suitable for general purpose and time sharing applications by multiple users. Contrasted with UMA, NUMA is a shared memory architecture that describes the placement of main memory modules with respect to processors in a multiprocessor system, as shown in Fig. 3. Based on the idea, however, Intel and AMD use the different technical paths to realize the NUMA architecture.

To better understanding NUMA roles in parallel FDTD method, we do many simulations using GEMS with NUMA and UMA policy, respectively. For example, the command for running the GEMS project with NUMA is:

mpirun –np 9 -machinefile hosts nuamctl--physcpubind=0-8,9-15 /gpfsAPP/GEMS /GEMS_Solver test.gpv

and the command for running the GEMS project without NUMA is:

mpirun -np 9 -machinefile hosts /gpfsAPP /GEMS/GEMS_Solver test.gpv

In addition, job balancing plays an important role in determining performance of the parallel code. Proper job balancing can obtain good performance of the parallel FDTD code on the HPC system, while improper job balancing may reduce the performance of parallel code for most of the processors in the cluster to that of "waiting" during the simulation process. Another important factor that affects the parallel efficiency is the division of the sub-domains according to the allocation of the array in the computer's memory. Processor affinity is a modification of the native central queue scheduling algorithm in a symmetric multiprocessing operating system. Taking advantage of the fact that some remnants of a process may remain in one processor's state from the last time the process ran, we can enhance the performance of parallel FDTD code on a HPC cluster. For example, if we use two nodes (4 CPUS, 16 cores) to run GEMS with binding rank to nodes, CPUs and cores, respectively, we should first edit the rank files for banding nodes, CPUs and cores as following:

For binding nodes:

rank 0=host0 slot=0-7
rank 1=host1 slot=0-7

For binding CPUs:

rank 0=host0 slot=0-3
rank 1=host0 slot=4-7
rank 2=host1 slot=0-3
rank 3=host1 slot=4-7

For binding Cores:

rank 0=host0 slot=0
rank 1=host0 slot=1
rank 2=host0 slot=2
rank 3=host0 slot=3
rank 4=host0 slot=4
…….
rank 7=host0 slot=7
rank 8=host1 slot=0
… ….
rank 15=host1 slot=7

And the following commands will be used to run GEMS testing project.

mpirun -np n -machinefile hosts -rf ranks /opt/GEMS/bin64/GEMS_Solver test.gpv

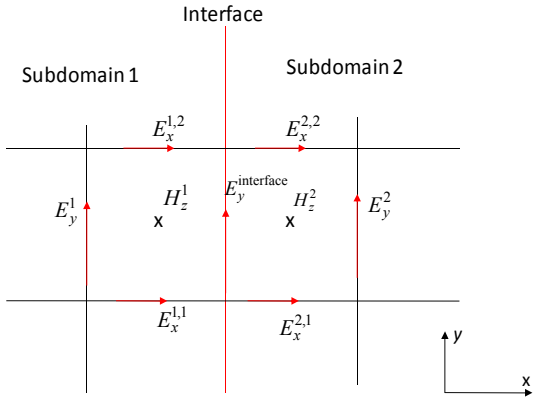


Fig. 1. Distributions of electric and magnetic fields near the subdomain interface.

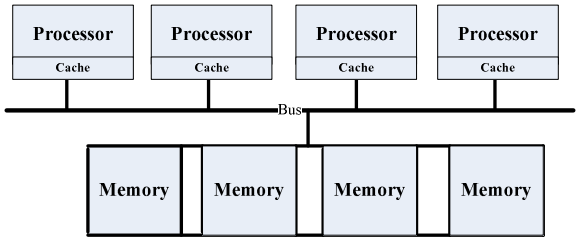


Fig. 2. UMA architecture.

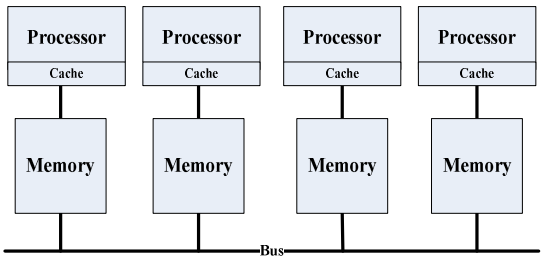


Fig. 3. NUMA architecture.

III. NUMERICAL EXPERIMENT RESULTS

In this section, we introduce a parallel processing platform installed with Linux operating system and investigate GEMS performance on the platform. The HPC cluster shown in Table 1 includes 23 nodes (21 computation nodes and 2 master nodes) and each node has two CPUs with Intel Xeon X5550 2.7GHz processor. The 10Gbps Ethernet is used to connect the computation nodes. To evaluate the performance of the FDTD code, we define the performance as follows:

$$Performanc\text{æ}\ (Mcells/s) = \frac{(N_x \times N_y \times N_z) \times Number\_of\_timesteps}{Simulation\_time\ (\text{sec}ond)}, \quad (3)$$

where N_x, N_y, N_z are the number of grids in *x, y* and *z* direction, respectively.

Firstly, NetPIPE [13] was used to test the performance of a network inside a node and internode. Table 2 gives NetPIPE results about the bandwidth, which shows that the network speed inside node is around 4 times of the internode. The NetPIPE results about the latency is described in Fig. 4, which shows the internode has a latency that is over 3 times than that inside-node. To test the job balancing role in determining performance of the parallel FDTD, a job with different processes has been run in one node of the HPC cluster. Fig. 5 shows the performance of the parallel code on one node with different processes, which indicates that job balancing plays an important role for performance of a parallel code.

An ideal case that is a hollow box with the simplest excitation and output, and its domain is truncated by using the Perfect Electric Conductor (PEC) boundary condition, was used as an example to study the impacts of processor affinity on parallel FDTD performance. The project settings including the number of unknowns, excitation type, output parameters and binging strategy (Binding each rank by node, by CPU and by core) are identical in the cluster simulations. Fig. 6 shows the performance of the parallel FDTD with different banding strategy, which indicates that parallel FDTD with banding rank to CPU give the best performance, and the worst case is banding rank to core. For example, the performances of the parallel FDTD code using 18 nodes are 5300 Mcells/sec, 4900 Mcells/sec, and 2700 Mcells/sec for binding rank to CPUs, nodes and cores, respectively. From the results shown in Fig. 6, we can also see that the job balancing between the internode and inside node play an important role to obtain good performance. If we bind each process to each core, we will suffer from the high latency of messages transmitting for there are more processes created between nodes. Binding each process to each node, we will not use the whole processor. However, if we choose to bind each process to CPU, all processors can be used and the latency of messages transmitting is less than that by banding to core.

Then, NUMA policy is used to reduce memory access time in the average case through the fast introduction of local memory. For NUMA providing each node with its own local memory, memory accesses, parallel code with NUMA policy can avoid throughput limitations. The performances of GEMS with NUMA policy are plotted in Fig. 7, where the numactl command is used as a plugin of GEMS software. As a compared date, the performances of GEMS without NUMA policy are also shown in Fig. 7. Comparing the results shown in Fig. 7, we can obtain that the performance of GEMS with NUMA is around 1.5 times than that without NUMA. For example, the performances of GEMS using 18 nodes of the HPC system are 3400 Mcells/sec and 5400 Mcells/sec for without NUMA policy and with NUMA policy, respectively.

Finally, we use the different options described above to simulate a reflector antenna fed by a dual mode circular horn, as shown in Fig. 8. The thinner horn part is excited by TE11 mode. The transit will generate the TM10 mode and have the same magnitude and out of phase with the TE11 mode at the end of the thicker horn. This horn will

generate a very low slob by cancelling the fields generated by the TE11 and TM10 modes.

Table 1: HPC cluster information

Computation Nodes (21) Master Nodes(2)	CPU type	Intel Xeon E5520
	Clock speed	2.67GHz
	Number of nodes	23
	Available memory	12GB (DDR3 1067MHz)
	Operating system	Cent OS (Linux)
	Network system	BNT 10Gbps Ethernet

Table 2: NetPIPE testing results: bandwidth

Netpipe Testing	Internode	8822.02 Mbps
	Inside node	33462.93 Mbps

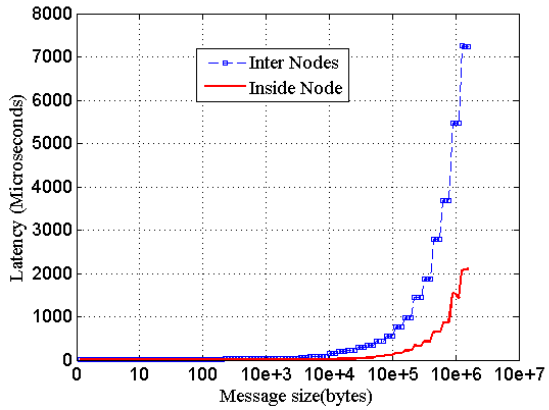


Fig. 4. NetPIPE testing results: latency.

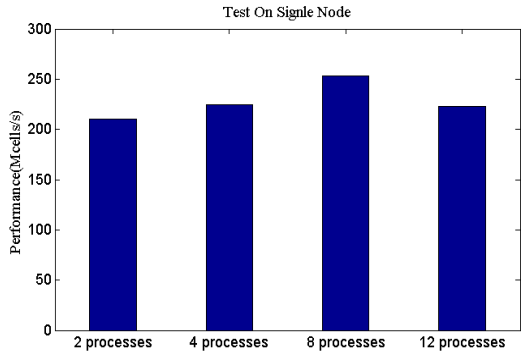


Fig. 5. Testing results about job balancing problem.

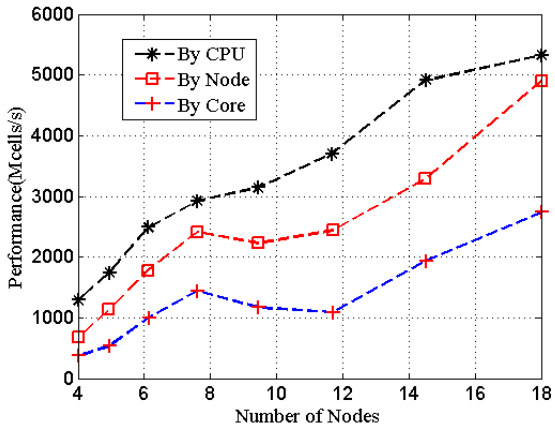


Fig. 6. Parallel FDTD performance with different binding strategy.

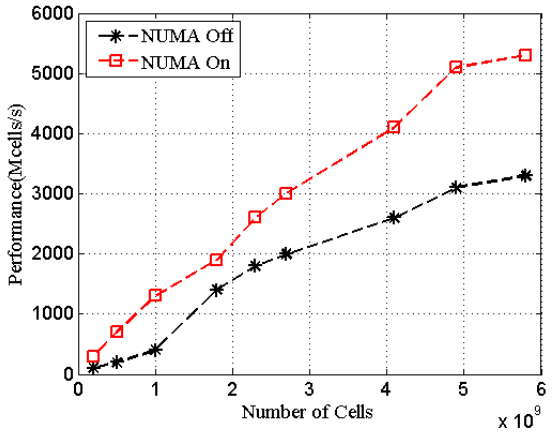


Fig. 7. Parallel PDTD performance with NUMA and without NUMA.

Finally, we use the different options described above to simulate a reflector antenna fed by a dual mode circular horn, as shown in Fig. 8. The thinner horn part is excited by TE11 mode. The transit will generate the TM10 mode and have the same magnitude and out of phase with the TE11 mode at the end of the thicker horn. This horn will generate a very low slob by cancelling the fields generated by the TE11 and TM10 modes.

Due to the symmetric property, we need only to simulate one quart of the original problem. The original domain size is 770 mm× 770 mm× 670 mm, and the one quart domain size is 385 mm × 385 mm × 670 mm, which is discretized into 569 × 569 × 1144 non-uniform cells. Output parameters include the far field pattern and return

loss. This is a very large problem, which cannot be solved by two nodes of the HPC directly even using parallel FDTD method. The return loss of the reflector antenna is plotted in Fig.9, and Fig.10 gives directivity of the parabolic reflector antenna at working frequency 12GHz.

The comparison between with and without NUMA option is shown in Fig. 11, where 4 computation nodes are used to simulate the problem. Fig.11 shows that parallel FDTD with banding rank to CPU give the best performance, and the performance of GEMS with NUMA is around 1.5 times than that without NUMA. To investigate the parallel efficiency of the parallel FDTD, 18 computation nodes are used to simulate the parabolic reflector antenna fed by a dual mode circular horn. The performance and consumed time of parallel FDTD using 18 computation nodes are illustrated in Fig. 12. Comparing the results in Fig. 11 and Fig. 12, we can obtain that the parallel efficiency of parallel FDTD is almost 90%. For example, when we run the parallel FDTD code by binding each rank to CPU, the consumed time of parallel FDTD with NUMA using 4 nodes is 2 hours, and that using 18 nodes is 23 minutes.

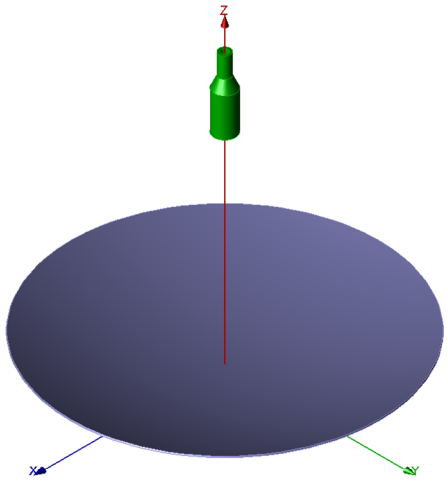


Fig. 8. Parabolic reflector antenna fed by a dual mode circular horn.

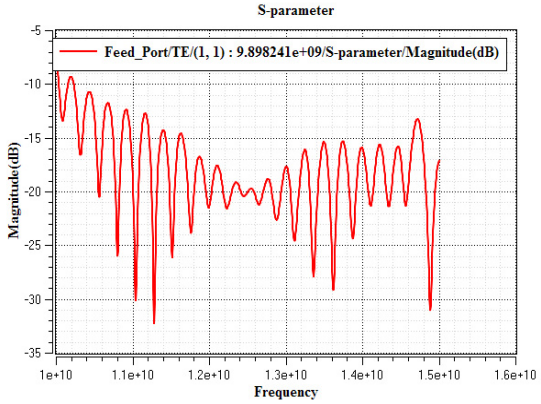
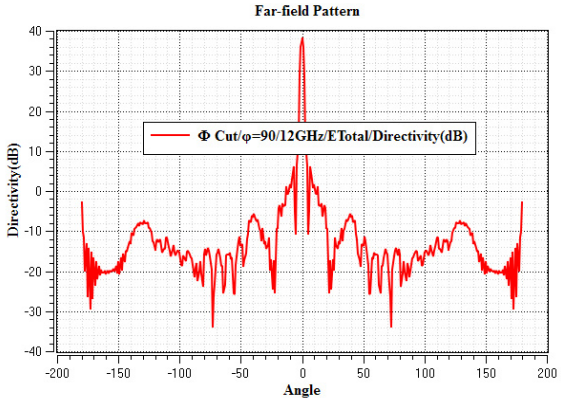
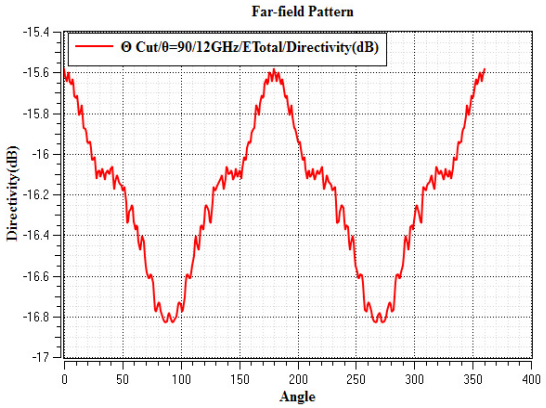


Fig. 9. Return loss of the parabolic reflector antenna.

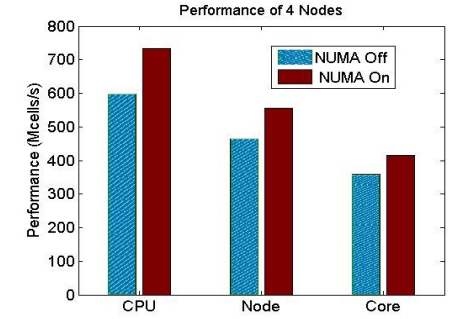


(A)

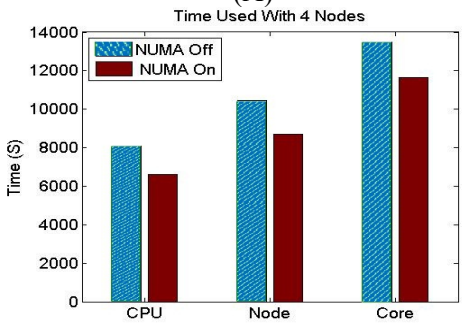


(B)

Fig. 10. Directivity of the parabolic reflector antenna at working frequency 12GHz. (A) ϕ cut-plane with $\phi = 90^0$ (B) θ cut-plane with $\theta = 90^0$.

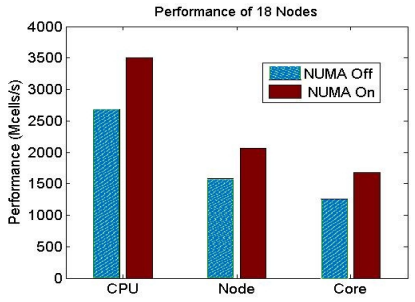


(A)

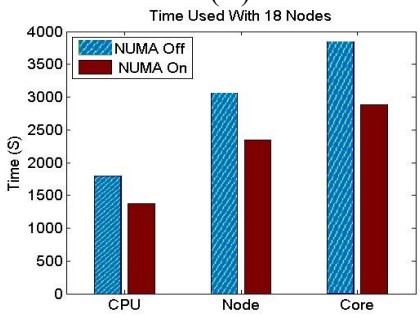


(B)

Fig. 11. Parallel FDTD with NUMA and without NUMA using 4 computation nodes. (A) Performance of parallel FDTD. (B) Consumed time of parallel FDTD.



(A)



(B)

Fig. 12. Parallel FDTD with NUMA and without NUMA using 18 computation nodes. (A) Performance of parallel FDTD. (B) Consumed time of parallel FDTD.

IV. CONCLUSION

In this paper, the processor affinity and NUMA policy are used to enhance the performance of a parallel FDTD code on a HPC cluster. By binding each rank to the node, CPU and core, we investigate the effect of processor affinity on parallel FDTD code performance and find that the processor affinity has significant impacts on the performance. With the advantage of NUMA policy that can reduce memory access time, the parallel FDTD code using NUMA policy can obtain better performance than that without NUMA policy. The proposed methods for optimizing the performance of parallel FDTD code are suite for other parallel code, which is very useful enhance the performance of a HPC cluster.

ACKNOWLEDGMENT

This work was supported in part by the Natural Science Foundation of Jiangsu Province under Grant No. BK2010174, in part by Natural Science Foundation of the Jiangsu Higher Education Institutions under Grant No. 10KJB510025, in part by the Open Project of State Key Laboratory of Millimeter Waves under Grant No. K201008, and in part by Postgraduate Innovation Project of Jiangsu Province under Grant No. CXZZ11_0899.

REFERENCES

[1] A. Taflove and S. Hagness, *Computational Electromagnetics: The Finite-Difference Time-Domain Method*, 3rd ed., Artech House, Norwood, MA, 2005.

[2] Harrington, R. F., *Field Computation by Moment Methods*, MacMillan, New York, 1968.

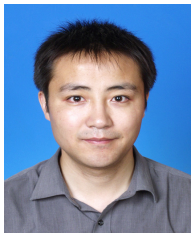
[3] J. M. Jin, *The Finite Element Method in Electromagnetics* (2nd Edition), New York: John Wiley & Sons, 2002.

[4] F. L. Teixeira, "A Summary Review on 25 Years of Progress and Future Challenges in FDTD and FETD Techniques," *Applied Computational Electromagnetics Society (ACES) Journal*, vol. 25, no. 1, pp. 1-14, 2010.

[5] V. Demir, "A Stacking Scheme to Improve the Efficiency of Finite-Difference Time-Domain Solutions on Graphics Processing Units," *Applied Computational Electromagnetics Society (ACES) Journal*, vol. 25, no. 4, pp. 323 - 330, 2010.

[6] X. Duan, X. Chen, K. Huang, H. Zhou, "A High Performance Parallel FDTD Based on Winsock and Multi-Threading on a PC-Cluster," *Applied Computational Electromagnetics Society (ACES) Journal*, vol. 26, no. 3, pp. 241 - 249, 2011.

[7] W. Yu, R. Mittra, T. Su, Y. Liu, and X. Yang, *Parallel Finite Difference Time Domain Method*, Artech House, Massachusetts, June, 2006.
[8] W. Yu, R. Mittra, X. Yang, and Y. Liu, *Electromagnetic Simulation Techniques Based FDTD Method*, John Wiley and Sons, 2009.
[9] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, 2nd ed., MIT Press, Cambridge, MA, 1999.
[10] Optimizing software applications for NUMA Site: http://software.intel.com.
[11] C. Zhang, X. Yuan, and A. Srinivasan, "Processor Affinity and MPI Performance on SMP-CMP Clusters," *IEEE International Symposium on Parallel & Distributed Processing*, Workshops and PHD. Forum (IPDPSW), pp. 1-8, 2010.
[12] GEMS-A 3D Parallel EM simulation software package, www.2comu.com, State College, PA, 16801, USA.
[13] http://www.scl.ameslab.gov/netpipe/.



Lei Zhao received a BS in Mathematics from Jiangsu Normal University, Xuzhou, China, in 1997, and an MS in Computational Mathematics and a PhD in Electromagnetic Fields and Microwave Technology from Southeast University, Nanjing, China, in 2004 and 2007, respectively. From August 2007 to August 2009, he worked in the Department of Electronics Engineering, the Chinese University of Hong Kong, as a research Associate. Since September 2009, he has worked in the School of Mathematical Sciences, Jiangsu Normal University. He is also the Director of the Center of Computational Science and Engineering, which is affiliated with the Jiangsu Normal University. He has published over 20 technical papers. His current research interests include accurate numerical modeling, biomedical EM compatibility, and parallel computing in computational EM.

Geng Chen was born in Suqian, China, in February 1989. He received the BS degree in Computational Mathematics from Jiangsu Normal University in 2012, and is currently working toward the MS degree at Jiangsu Normal University, Xuzhou, China. His research interests include parallel-processing techniques, numerical methods, and software development.

Wenhua Yu joined the Department of Electrical Engineering of the Pennsylvania State University and has been a group leader of the Electromagnetic Communication Lab since 1996. He received his PhD in Electrical Engineering from the Southwest Jiaotong University in 1994. He worked at the Beijing Institute of Technology as a Postdoctoral Research Associate from February 1995 to August 1996. He has published five books related to the FDTD method, parallel-processing techniques, software-development techniques, and simulation techniques, from 2003 to 2009. He has published over 150 technical papers and four book chapters. He founded the Computer and Communication Unlimited company, and serves as President and CEO. He is a Senior Member of the I EEE. His research interests include computational electromagnetic methods, software-development techniques, parallel-processing techniques, simulation and design of antennas, antenna arrays, and microwave circuits.