文章编号：1672-2892(2011)02-0169-06

基于 WinSock 和多线程技术的高性能并行 FDTD

段 鑫，陈 星

(四川大学 电子信息学院，四川 成都 610064)

摘 要：并行计算为时域有限差分(FDTD)方法仿真电大尺寸和复杂结构提供了强大的计算能力和内存资源。文章针对多核 PC 集群系统，提出了一种高性能并行 FDTD 算法，它采用 Windows Socket(WinSock)实现高效的进程间通信，同时采用多线程技术充分利用多核处理器资源。在集群系统上的实际测试表明：以 10 个处理器(30 个核)为例，该算法获得的加速比为 16.0，并行效率为 53.3%，优于单独使用消息传递接口(MPI)以及 MPI 结合 OpenMP 的传统 FDTD 并行算法，后两者在相同测试条件下仅分别获得 13.7,12.2 的加速比和 45.8%,40.7%的并行效率。

关键词：时域有限差分方法；并行计算；PC 集群；WinSock 编程接口；多线程

中图分类号：TN823.17 **文献标识码：**A

A high performance parallel FDTD based on WinSock and multi-threading

DUAN Xin，CHEN Xing

(College of Electronics and Information Engineering，Sichuan University，Chengdu Sichuan 610064，China)

Abstract: Parallel technology is a powerful tool to provide the necessary computing power and memory resources for the Finite Difference Time Domain(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 Windows Socket(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, finally resulting in the development of a FDTD parallelization strategy. An experiment is presented with its results demonstrating the proposed Winsock and multi-threading-based parallel FDTD achieving speedup of 16.0 and efficiency of 53.3% when 10 processors with 30 cores are employed. It outperforms traditional parallel FDTD which uses either MPI or MPI-OpenMP, gaining speedup of 13.7,12.2 and efficiency of 45.8%,40.7% respectively under the same circumstance.

Key words: Finite Difference Time Domain; Parallel Computation; PC Cluster; WinSock; Multi-threading

时域有限差分算法(FDTD)作为常用数值仿真算法，已被广泛用于求解各种电磁问题[1-3]。采用并行计算系统如 PC 集群是进行电大尺寸和复杂结构目标仿真计算的有效途径[4-6]。目前，绝大多数 FDTD 并行计算[4-5,7-9]采用消息传递接口(Message Passing Interface，MPI)[10-11]，但当今 PC 集群大量采用多核处理器，同一处理器中的多核可以通过创建线程实现共享内存的直接读取[1,12-14]，与此对比，MPI 基于进程之间消息传递的并行策略在多核并行上效率较低。并且，现有 MPI 的实现如 MPICH2，调试和运行都在 mpiexec.exe 下进行，不仅动态负载平衡不易实施，而且严重限制了图形化操作。已有的并行 FDTD 算法研究[4-9]很少考虑多核处理器特点，个别研究[15-16]采用共享内存并行编程接口(Open Multi-Processing，OpenMP)[17-18]来利用处理器的多核计算能力，但一般情况下，用户如果不熟悉多线程编程和 OpenMP 的优化技巧，则可能会造成额外线程开销过大。

为了充分发挥 PC 集群的多处理器多核结构带来的计算能力，本文提出了一种基于 WinSock 和多线程的并行 FDTD 算法：在 PC 集群中每只处理器上创建进程，进程之间的数据交换以消息传递方式，采用高效、底层的 WinSock 函数实现，运用网格重叠技术使只有子区域交界上的磁场数据需要传递；在多核处理器中的每一个核上，采用 Win32 线程应用程序编程接口(Application Programming Interface，API)函数创建一个线程，并与该核绑定；

收稿日期：2010-06-29；修回日期：2010-09-08

线程间的数据交换是由不同线程读写共享变量来实现的；采用同步机制对线程执行顺序加以限制。为了获得良好的负载平衡，FDTD 区域分割将考虑处理器之间和同一处理器的核之间在数据交换操作上的耗时差异，以及不同类型网格(例如普通网格和完全匹配层(Perfect Match Layer，PML)[19]网格)的计算量差异。

本文将通过数值实验，在 PC 集群上测试所提出并行 FDTD 算法的并行效率，并与基于 MPI 和 MPI 结合 OpenMP 的 2 种并行 FDTD 算法进行比较。

1 FDTD 简要介绍

从 1966 年 Kane S Yee 的里程碑论文[1]开始，FDTD 得到迅猛发展，已经成为一类广泛使用的数值仿真方法。采用 Yee 网格，Maxwell 的 2 个旋度方程在空间和时间上被离散为 FDTD 差分方程[7]：

$$\boldsymbol{E}_x^{n+1}(i+\frac{1}{2},j,k)=\frac{\varepsilon_x-0.5\Delta t\sigma_x}{\varepsilon_x+0.5\Delta t\sigma_x}\boldsymbol{E}_x^{n}(i+\frac{1}{2},j,k)+\frac{\Delta t}{\varepsilon_x+0.5\Delta t\sigma_x}\left[\begin{array}{l}\dfrac{\boldsymbol{H}_z^{n+\frac{1}{2}}(i+\frac{1}{2},j+\frac{1}{2},k)-\boldsymbol{H}_z^{n+\frac{1}{2}}(i+\frac{1}{2},j-\frac{1}{2},k)}{\Delta y}\\ -\dfrac{\boldsymbol{H}_y^{n+\frac{1}{2}}(i+\frac{1}{2},j,k+\frac{1}{2})-\boldsymbol{H}_y^{n+\frac{1}{2}}(i+\frac{1}{2},j,k-\frac{1}{2})}{\Delta z}\end{array}\right] \tag{1}$$

$$\boldsymbol{H}_x^{n+\frac{1}{2}}(i,j+\frac{1}{2},k+\frac{1}{2})=\frac{\mu_x-0.5\Delta t\sigma_{Mx}}{\mu_x+0.5\Delta t\sigma_{Mx}}\boldsymbol{H}_x^{n-\frac{1}{2}}(i,j+\frac{1}{2},k+\frac{1}{2})+\frac{\mu_x-0.5\Delta t\sigma_{Mx}}{\mu_x+0.5\Delta t\sigma_{Mx}}\left[\begin{array}{l}\dfrac{\boldsymbol{E}_y^{n}(i,j+\frac{1}{2},k+1)-\boldsymbol{E}_y^{n}(i,j+\frac{1}{2},k)}{\Delta z}\\ -\dfrac{\boldsymbol{E}_z^{n}(i,j+1,k+\frac{1}{2})-\boldsymbol{E}_z^{n}(i,j,k+\frac{1}{2})}{\Delta y}\end{array}\right] \tag{2}$$

式中：$\boldsymbol{E}_x$ 和 $\boldsymbol{H}_x$ 分别是电场、磁场在直角坐标系中某点的 x 分量，其上标 n,n+1/2,n+1 分别表示当前时间步、下半个时间步、下一个时间步；ε_x 是介电常数；σ_x 是电导率；μ_x 是磁导率；σ_{Mx} 是导磁率；Δt 是时间步间隔。

从差分方程(电场和磁场其他分量的差分方程形式类似)可以看出，FDTD 采用迭代计算过程，每个网格电磁场分量计算只需要相邻网格数据，这极大地简化了并行计算。

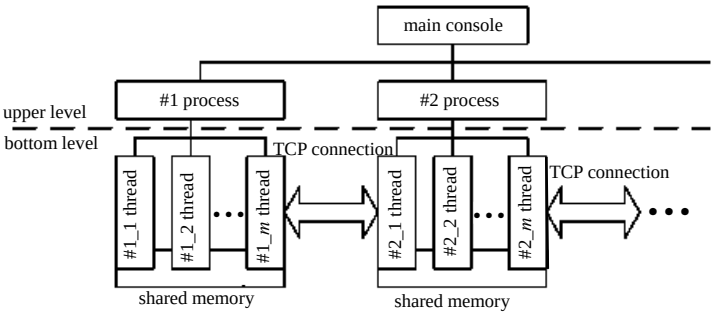

Fig.1 Two-level model for the parallel FDTD

图 1 并行 FDTD 的双层模型

2 基于 WinSock 和多线程的并行 FDTD 算法

2.1 并行模型

在 PC 集群上，多个处理器采用分布式地址空间，而同一处理器的多个核为共享地址空间，不同类型地址空间对并行计算中数据交换有着重大影响[20]。为了适应这种混合的地址空间，本文提出的并行 FDTD 算法采用了双层模型，见图 1。在模型上层，主进程将计算区域分割为若干子区域，并在每个处理器创建计算进程，而模型下层为各计算进程在该处理器多个核中创建相应的计算线程。

为避免前述 MPI 缺陷，采用 WinSock 实现进程间的消息传递。WinSock 在 Windows TCP/IP 用户应用和下层 TCP/IP 协议栈直接定义了 1 组标准接口[21-22]。如图 2 所示，利用传输控制协议(TCP)在 2 个进程的套接字口之间建立连接。一旦连接建立，它们就可以调

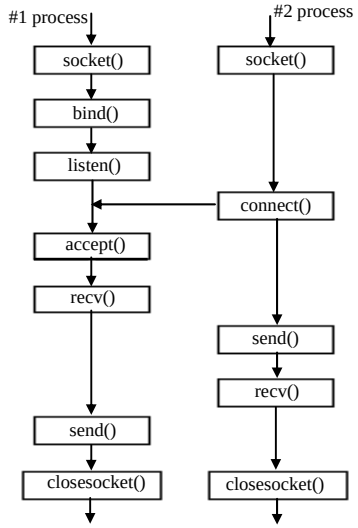

Fig.2 Flow chart of WinSock programming

图 2 WinSock 实现进程间消息传递的编程流程

用 send()和 recv()传递消息。

对双层模型的下层采用多线程技术。使用 Win32 线程 API 实现多线程的关键步骤如下:

a) 每个处理器上的计算进程以单线程开始，也称作主线程;

b) 调用_beginthreadex()创建从线程，并赋予相应的计算任务;

c) 计算过程中，线程通过对共享变量的读写完成数据交换。同时，同步机制用于协调线程的执行和共享数据的管理;

d) 完成计算任务后，从线程调用_endthreadex()终止。

为了得到更高的并行效率，调用 SetThreadIdealProcessor()将从线程绑定到各核，从而阻止线程在程序执行阶段被调度到其他核。

2.2 区域分割

FDTD 的计算区域分割如图 3 所示。每个进程被分配 1 个子区域，子区域被进一步分割为更小计算区域后再分配给核绑定的计算线程。每个子区域的两端定义扩展页用来存储相邻子区域交界面的场值。对同一处理器不同核中的计算线程，场值存储在共享变量中，可以直接相互读写，因此不用扩展页。

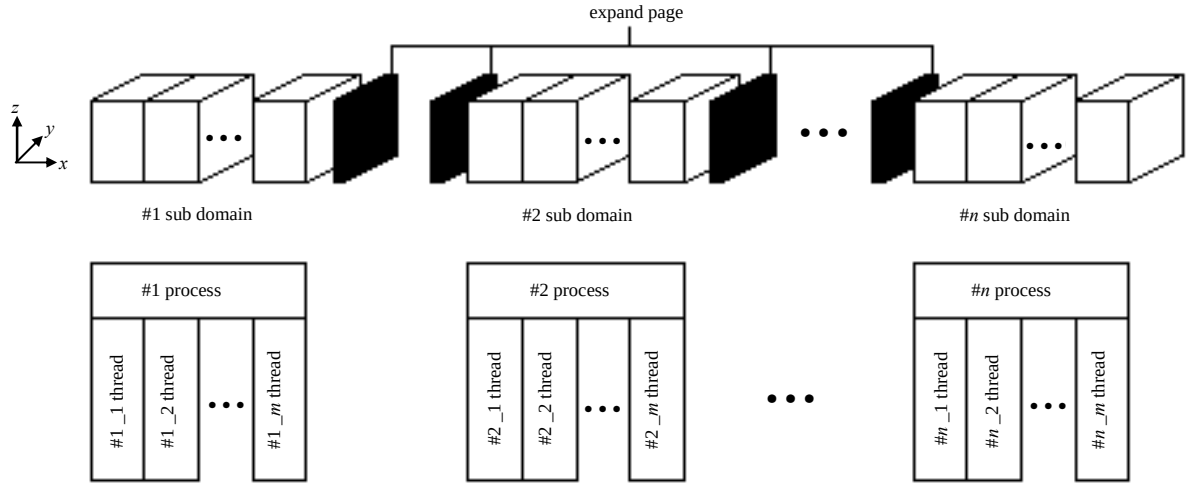

Fig.3 Domain decomposition

图 3 区域分割

2.3 同步

FDTD 的迭代计算对电场和磁场值的更新顺序有着严格要求，否则将导致错误结果。因此，为控制各计算线程的执行顺序，同步机制对并行 FDTD 是必不可少的。

图 4 为本文的同步机制流程。SetEvent()将事件对象的状态设置为信号状态，WaitForMultipleObjects()用于等待所有事件对象都设置成信号状态。

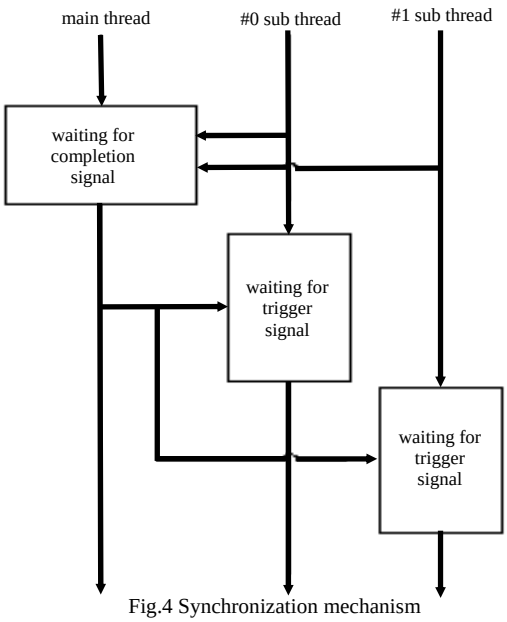

Fig.4 Synchronization mechanism

图 4 同步机制

2.4 数据交换

根据 FDTD 算法递推公式和区域分割原理，FDTD 并行计算过程中的每一个迭代步，相邻子区域需要交换区域交界面的场值。对同一个处理器中的计算线程，数据交换通过线程读写共享的内存变量完成；对分布于不同处理器中计算线程，则需要显式调用 WinSock 的消息传递函数实现数据交换。

数据交换是影响并行效率的主要因素之一。为最大限度地减少数据交换次数和交换量，采用了如图 5 所示的网格重叠技术[8]，即把分配给相邻进程的子区域交界面重叠起来，这样其电场量会在对应的由不同进程产生的相邻线程上更新，对每个

时间步来说，虽然计算了 2 次交界面上的电场量，但是只有磁场量需要被传递。不过，网格重叠技术不适用于同一个进程产生的线程，否则会导致共享的电场数组被更新 2 次，从而计算错误。

2.5 负载平衡

负载平衡是实现高效率并行计算的关键因素之一[23]。对并行 FDTD 而言，一个良好的负载平衡策略应该合理地划分计算空间，使各计算线程在每个时间步中的执行时间相等，避免线程因相互等待造成空闲状态。这其中有很多因素影响负载平衡。首先，不同的网格，如普通网格和 PML 网格拥有不同的计算量。其次，1 次消息传递操作远比 1 次直接读写共享变量操作耗时。但是，如何权衡这些因素实现合理的区域分割还没有明确、有效的规则供遵循，只能借鉴经验和反复尝试。

本文测试表明：对 FDTD，1 个 PML 网格的计算耗时大概是普通网格的 2.6 倍。因此，在计算区域分割为子区域时，1 个 PML 网格被换算为 2.6 个普通网格。另 1 个简单有效的方法是，当 1 个计算线程需要以消息传递方式执行数据交换操作时，则将 1 层网格分给相邻的不需要执行消息传递操作的线程，以补偿消息传递操作较多的耗时。

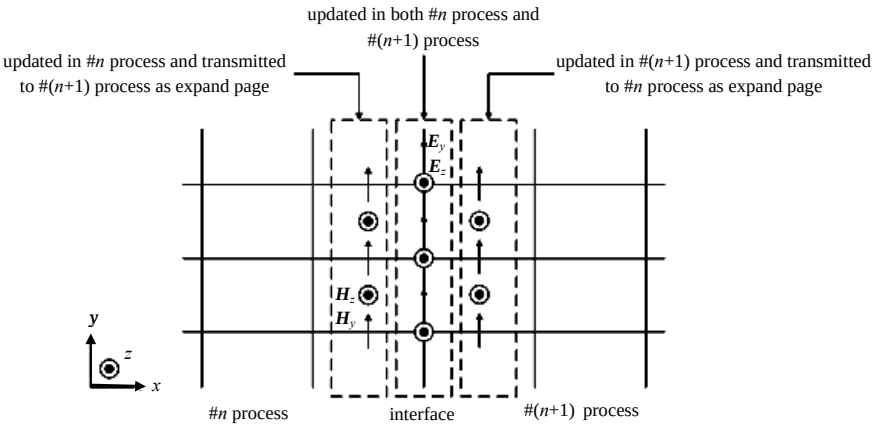

Fig.5 Field exchange configuration employing the overlapping scheme
图 5 采用网格重叠技术的场值交换

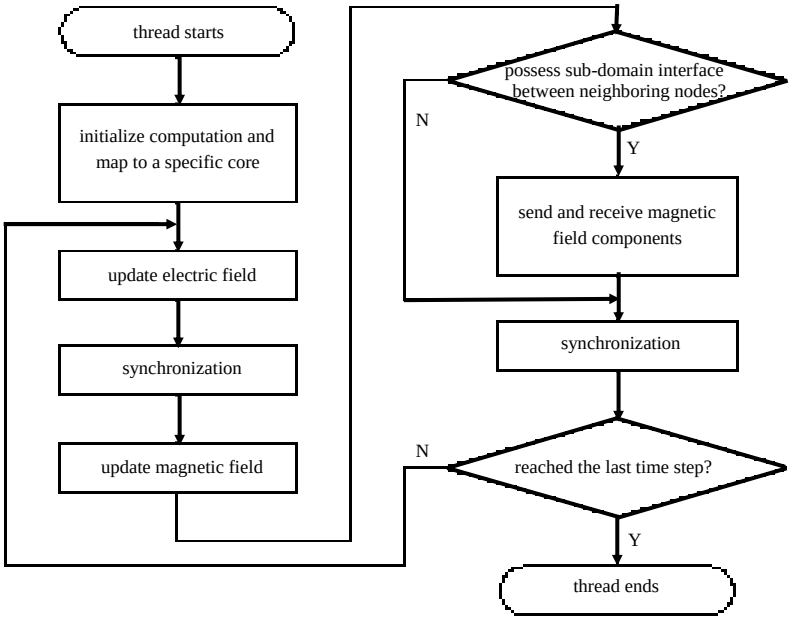

Fig.6 Flow chart of the thread execution
图 6 计算线程流程图

2.6 计算线程流程

计算线程是本文 FDTD 并行计算任务的最终和实际执行者，其计算流程如图 6 所示。电场和磁场量在各自更新后都会有一个同步操作。如果一个线程包含相邻节点的子区域交界面，则在磁场更新后会有发送和接收操作来交换磁场量。

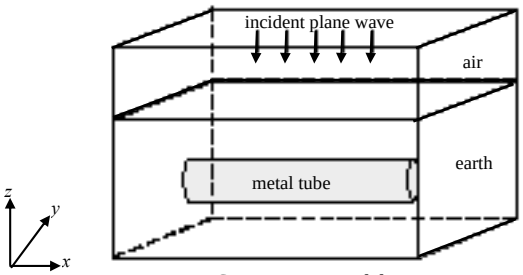

Fig.7 Computation model
图 7 计算模型

3 并行计算实验和结果分析

为了验证本文提出的并行 FDTD 算法的可行性和测试其并行效率，分别采用本文算法、基于 MPI 的并行 FDTD 算法、MPI 和 OpenMP 的混合并行 FDTD 算法对图 8 散射场问题进行仿真计算。1 根金属管埋在相对介电常数为 4.0 和电导率为 0.001 的泥土里，如图 7 所示。1 GHz 的平面波垂直从空气中入射到泥土。金属管长为 0.66 m，半径为 0.015 m。计算空间在 x,y 和 z 轴上被离散成 720×110×150 个网格，其中 dx=dy=dz=0.001。

基于 MPI 的并行 FDTD 算法是在每个核上创建 1 个计算进程，所有计算进程之间都采用 MPI，依靠消息传递机制实现数据交换。MPI 和 OpenMP 的混合并行 FDTD 算法是在每个处理器上创建 1 个计算进程，然后使用 OpenMP 利用多核处理器的计算能力。测试所用的 PC 集群的每个计算节点拥有一个 Intel Q6600 四核处理器和 8 G 内存，用 1 000 Mb/s 以太网连接。测试中，每个处理器中最多有 3 个核用于并行 FDTD 计算，剩下 1 个核用于操作系统等常规任务。

表 1 并行 FDTD 算法的运行时间(s)和内存耗用(MB)测试值
Table1 Run time(in seconds) and memory consumption(in MB)

number of threads/processes	run time			memory consumption
	the proposed method	MPI	MPI-OpenMP	
1	2 390	2 390	2 390	910
6	450	460	519	460
12	258	272	303	235
18	198	221	243	160
24	171	191	204	122
30	149	173	195	100

表 1 列举了测试得到的 3 种并行 FDTD 算法的运行时间和单只处理器内存消耗测试数据(3 种算法的内存消耗相同)。可以看出，并行计算能够大幅度地减少 FDTD 计算时间和单只处理器内存消耗，本文提出的并行 FDTD 算法在 3 种算法中速度最快，例如：在线程/进程数为 30 时，分别比基于 MPI 的并行 FDTD 算法、MPI 和 OpenMP 混合并行 FDTD 算法快 24 s 和 46 s。图 8 显示了 3 种并行 FDTD 算法的加速比和并行效率，其中加速比等于串行计算时间和并行计算时间的比值，并行效率等于加速比除以参与并行计算的核的数量。可以看出，本文提出的并行 FDTD 算法有更高的加速比和并行效率，并且随着线程/进程数的增加，该算法的优势更加明显。在线程数为 30 时，其加速比达到 16.00，并行效率为 53.3%；而基于 MPI 的并行 FDTD 算法的加速比为 13.75，并行效率为 45.8%；MPI 和 OpenMP 混合并行 FDTD 算法的加速比为 12.23，并行效率为 40.7%。本文提出的算法使用 WinSock 替代 MPI 进行处理器间的通信，不仅效率更高，还摆脱了 mpiexec.exe 的限制，从而简化了调试和后续编程。在单个处理器内部，多线程的使用使进程间的消息传递次数显著减少，而线程间的共享内存读写速度远远快于进程间消息传递，因此本文提出的算法具有更好的并行性能。同时，这种清晰的并行模型使得动态负载平衡的加入更加方便。文献[14–15]表明 MPI 混合 OpenMP 的并行 FDTD 算法的并行性能通常优于单纯的 MPI，但本文的测试结果却相反。这主要是因为：第一，本文算例的子区域划分力度不够小；第二，为了体现 OpenMP 的快速部署和使用简易的特性，文中 OpenMP 只是用来自动并行化循环，没有做过多的优化。需要指出的是，加速比和并行效率的测试值与计算对象密切相关，如果计算量加大，并行管理开销和数据通信延时占整个并行计算时间比例会减小，可以测试得到更高的加速比和并行效率。本例中计算模型的规模较小，因此测试得到的加速比和并行效率值较低。

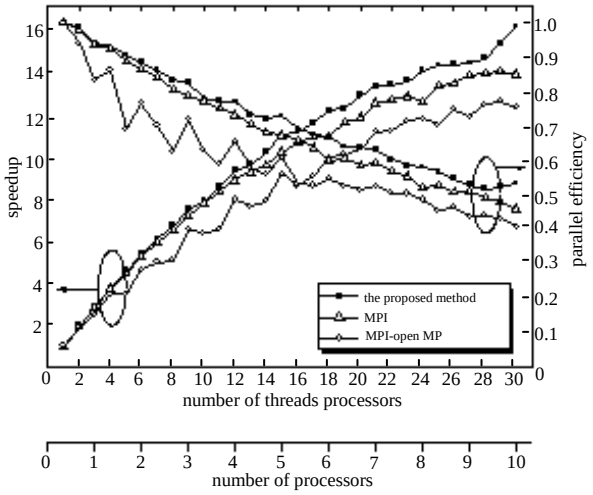

Fig.8 Speedup and parallel efficiency
图 8 加速比和并行效率

4　结论

针对当前 PC 集群普遍采用多核处理器的特点，以及并行计算普遍采用的 MPI 和 OpenMP 的缺点，提出了一种新的高性能并行 FDTD 算法，它采用双层模型和多线程技术，在 PC 集群的处理器中创建计算进程，并进一步在处理器的核中创建计算线程，不同处理器中线程之间利用 WinSock 建立 TCP 连接以传递消息，而同一处理器中线程采用直接读写共享内存来完成数据交换。实验测试结果表明，本文提出的并行 FDTD 算法比基于 MPI 的并行 FDTD 算法，以及 MPI 和 OpenMP 混合并行 FDTD 算法在相同条件下有更高的加速比和并行效率。

参考文献：

[1] Kane Yee. Numerical Solution of Initial Boundary Value Problems Involving Maxwell's Equations in Isotropic Media[J]. IEEE Transactions on Antennas and Propagation, 1966,14(3):302–307.

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

[3] 张文涛,杨向华,周邦华. FDTD 分析准八木天线的算法实现[J]. 信息与电子工程, 2010,8(3):273–275. (ZHANG Wentao,

YANG Xianghua,ZHOU Banghua. Programming analysis of the quasi-Yagi antenna by using FDTD method[J]. Information and Electronic Engineering, 2010,8(3):273-275.)

[4] Guiffaut C,Mahdjoubi K. A Parallel FDTD Algorithm using the MPI Library[J]. IEEE Antennas and Propagation Magazine, 2001,43(2):94-103.

[5] Guy A,Schiavone,Iulian Codreanu,et al. FDTD Speedups Obtained in Distributed Computing on a Linux Workstation Cluster[C]// Antennas and Propagation Society International Symposium. Salt Lake City:[s.n.], 2000:1336-1339.

[6] VaradarajanV,Mittra Raj. Finite Difference Time Domain(FDTD) Analysis Using Distributed Computing[J]. IEEE Microwave and Guided Wave Letters, 1994,4(5):144-145.

[7] YU Wenhua,Mittra Raj,SU Tao,et al. Parallel Finite Difference Time Domain Method[M]. Boston:Artech House, 2006.

[8] YU Wenhua,LIU Yongjun,SU Tao,et al. A Robust Parallel Conformal Finite Difference Time Domain Processing Package Using the MPI Library[J]. IEEE Antennas and Propagation Magazine, 2005,47(3):39-59.

[9] YU Wenhua,Hashemi M R,Raj Mittra,et al. Massively Parallel Conformal FDTD on a BlueGene Supercomputer[J]. IEEE Transactions on Advanced Packaging, 2007,30(2):335-341.

[10] Marc Snir,Steve Otto,Steven Huss-Lederman,et al. MPI:The Complete Reference[M]. Cambridge:The MIT Press, 1996.

[11] William Gropp,Ewing Lusk,Anthony Skjellum. MPI:Portable Parallel Programming with the Message-Passing Interface[M]. 2nd edition. Cambridge:The MIT Press, 1999.

[12] Shameem Akhter,Jason Roberts. Multi-Core Programming:Increasing Performance through Software Multi-threading[M]. Hillsboro,OR:Intel Press, 2006.

[13] Mohammad J,Rashti Ahmad Afsahi. 10-Gigabit iWARP Ethernet:Comparative Performance Analysis with InfiniBand and Myrinet-10G[C]// Parallel and Distributed Processing Symposium,USA:[s.n.], 2007:1-8.

[14] Brian Hausauer. Performance Study of Winsock Direct with 10 Gb Ethernet RDMA-Enabled NIC[C]// Cluster Computing, Cluster Computing. USA:IEEE International Conference, 2006:1-6.

[15] Mehmet F Su,Ihab El-Kady,David A. A Novel FDTD Application Featuring Open MP-MPI Hybrid Parallelization[C]// Parallel Processing Atlanta. GA:[s.n.], 2004:373-379.

[16] Robert Rosenberg,Guy Norton,Jorge C,et al. Modeling Pulse Propagation and Scattering in a Dispersive Medium:Performance of MPI/OpenMP Hybrid Code[C]// SC 2006 Conference, Proceeding of the 2006 ACM/IEEE. Tampa:[s.n.], 2006:47-47.

[17] Barbara Chapman,Gabriele Jost,Ruud Van Der Pas. Using OpenMP:Portable Shared Memory Parallel Programming[M]. Mass.:MIT Press, 2008.

[18] Rohit Chandra,Leonardo Dagum,Dave Kohr,et al. Parallel Programming in OpenMP[M]. New York:Academic Press, 2001.

[19] Berenger J. A Perfectly Matched Layer Medium for the Absorption of Electromagnetic Waves[J]. Comput.J, 1994,114(2): 185-200.

[20] Thomas Rauber,Gudula Rnger. Parallel Programming for Multicore and Cluster Systems[M]. Berlin:Springer, 2010.

[21] Anthony Jones,Jim Ohlund. Network Programming for Microsoft Windows[M]. Washington:Microsoft Press, 2002.

[22] Bob Quinn,David Shute. Windows Sockets Network Programming[M]. New Jersey:Addison-Wesley Professional, 2009.

[23] Jerrell Watts,Stephen Taylor. A practical approach to dynamic load balancing[J]. IEEE Transactions on Parallel and Distributed Systems, 1998,9(3):235-248.

作者简介:

段 鑫(1986-),男,四川省泸州市人,在读硕士研究生,主要研究方向为计算电磁学、天线设计.email:shawnduan@gmail.com.

陈 星(1970-),男,四川省通江县人,教授,博士,主要研究方向为天线设计、优化算法、数值方法、并行计算.