2010 International Conference on Computational and Information Sciences

The Parallelization of Three-Dimensional Electro-Magnetic Particle Model Using Both MPI and OPENMP

Xiaoyang Yan
School of Computer Science and Engineering
South China University of Technology
Guangzhou, China
yxycary@gmail.com

Weiwen Zhang
School of Computer Science and Engineering
South China University of Technology
Guangzhou, China
gavincheungcn@gmail.com

Huifang Deng
School of Computer Science and Engineering
South China University of Technology
Guangzhou, China
hdeng2008@gmail.com

Shehui Bu
School of Computer Science and Engineering
South China University of Technology
Guangzhou, China
bushehui@gmail.com

***Abstract*—Three-Dimensional Electro-Magnetic Particle Model (3DEMPM), based on the equations of Maxwell and Newton-Lorentz, takes advantages of the Finite-Difference Time-Domain (FDTD) and Particle-In-Cell (PIC) to trace a large quantity of particles in order to gain insight into the physics of them. Although MPI alone can be used to parallelize with 1D decomposition along *x*-direction, the efficiency decreases with increase of CPUs because there are more communications involved. We combine MPI and OPENMP to reduce the communications in the PC cluster, one node of which shares memory with duel-core. The whole domain is decomposed into several sub-domains, the same number as the nodes. Between the nodes we use MPI to realize the communications, and inside each node the OPENMP is applied to do the parallel computing with no communication. In this way, the higher speed-up is achieved while the communication is reduced.**

Keywords-3DEMPM; Parallel computing; MPI; OPENMP

I. INTRODUCTION

Computer simulation of plasma mainly comprises two numerical methods: magneto hydro-dynamic (MHD)[1] and particle-in-cell (PIC)[1]. MHD regards the plasma as fluid and makes a statistical analysis on the particles. Using MHD simulation, a good result could be achieved for the macro-scale, but the micro features of the plasma are neglected. Unlike MHD, the ions and electrons are treated as the relativistic and collision-less particles in particle simulation, thus we can get the micro features of plasma completely[1]. Hence it is an advanced model to study the micro phenomena of magnetic reconnection, magnetic storm, etc..

Based on PIC, the whole simulation grid is decomposed with the YEE lattice[2] and the electric and magnetic field both scatter discretely on the grids in 3DEMPM. Given the initial position and velocity of the particles as well as the fields, we could get the force on every particle using Lorentz equation before we calculate the new velocity and position of particles with Newton equations. Then we get the current density distribution by averaging statistic method and solve the Maxwell equations to update the electric and magnetic fields on every grid, so the particles move forward in the grid in turn. Thus a recursive procedure is formed. Usually, we run this code for thousands of loops which takes a long time and cannot be probably accomplished by just one computer. Parallel computing is an effective way to reduce the computing time [3].

3DEMPM can be parallelized with MPI and OPENMP. The OPENMP, a shared-memory standard of multithreads, allows large numbers of CPUs to have access to a single memory space. And MPI is based on message passing, running on a distributed memory system. It is essential to explore our parallelization of 3DEMPM with these two models. In addition, we can further parallelize 3DEMPM with the combination of MPI and OPENMP.

II. DESIGN OF PARALLEL ALGORITHM WITH MPI

A. Some Ideas of Parallelism

Basically, there are two solutions for the parallelized 3DEMPM.

The first one is to distribute the particles on average to each process, and magnetic and electric fields are assigned to all. In this way, the calculation of particles movement and current contribution can be parallelized. The current on each process are summed up to get the final update of electric field, and we can solve the Maxwell equations on each process respectively. The merit of this solution is load balance. But as the size of the grid becomes larger and larger, the communication between processes will increase dramatically, which is a great challenge of the switch and may lead to the problem of communication obstacles.

The second one is to decompose the domain into several sub-domains so that the physical value of fields and particles corresponding to the sub-domain are stored on each process.

978-0-7695-4270-6/10 $26.00 © 2010 IEEE
DOI 10.1109/ICCIS.2010.58

In this methodology, each process just needs to exchange the fields on the boundaries of sub-domain with its neighboring process for the calculation of electric and magnetic fields, and also the particles that will move to neighboring process. But as time goes by, particles are no longer equally distributed on the sub-domain, causing the problem of load imbalance, which means some sub-domains have to manipulate more particles while others do not.

The second solution is applied in this paper. On *x* axis, the domain is decomposed on average as shown in Fig. 1.

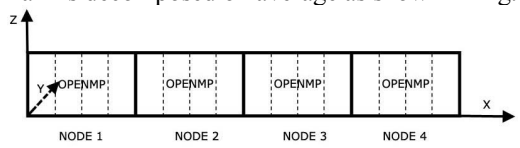

Figure 1 Decomposition with MPI plus OpenMP

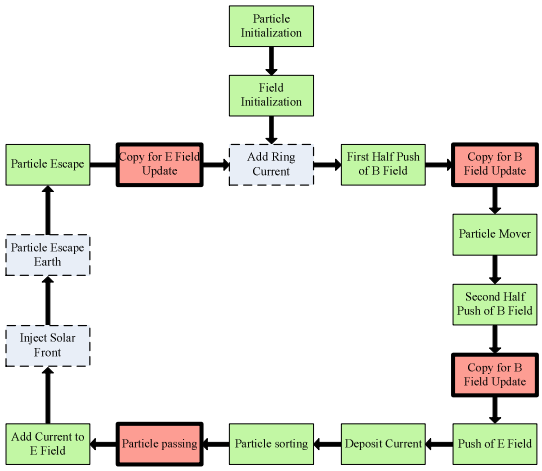

Figure 2 Flow chart for parallelized 3DEMPM

Fig. 2 shows the process of the parallelism, in which the subroutines with solid boarder are parallelized, while the ones with bold boarder are communications between processes, and the others with dashed lines work in the same way as the serial program.

B. The Parallel Computing of Electric and Magnetic Field

By YEE lattice and centre-difference, the magnetic field on the *(i,j,k)* grid depends on the electric field on (*i+1,j,k*), (*i,j+1,k*), (*i,j,k+1*); the electric field on the (*i,j,k*) grid is coupled with the magnetic field on (*i-1,j,k*), (*i,j-1,k*), (*i,j,k-1*).

In addition, when computing the position and velocity of particles, the particles on the *x* axis depends on the grid of *x=i-1* and *x=i+1*; when calculating the charge and current for the update of electric field, the electric field on the *(i,j,k)* grid relies on the grids *(i,j-1,k-1)*,*(i,j-1,k)*,*(i,j,k-1)*,*(i-1,j,k-1)*,*(i-1,j,k)*, *(i-1,j-1,k)* where particles are located at. To realize the data exchange between processes, the concept of Guard Cell[7] is introduced: for each process, on *x* axis, two grids are added to the front and three grids are added to the end. With such a scheme of expanding the grids, each process can obtain the field from its neighboring process.

Each process calculates the fields on [*FBDL*，*FBDR*], and [*FBDL*，*FBDR*] of all processes make up a whole domain, where *FBDL=3*, *FBDR=nFx+2* and *nFx=(mx-5)/Nproc* except for the last process with *FBDR = mx-2-(Nproc-1)*nFx*. And *PBDL* and *PBDR* referring to the boundaries for particles are also defined: *PBDL = (np-1)*nFx+3.0, PBDR = np*nFx+3.0*.

After the calculation of FIELD_PUSH, the boundaries of fields should be transferred to the neighboring process for communication. This is done by the subroutine FIELD_COPY. Process *N-1* delivers the field value on *FBDR* to process *N* on *FBDL-1*, meanwhile, process *N* passes the field value on *FBDL* to process *N-1* on *FBDR+1*, with the result that both processes get the field value on the boundaries from neighbors, as it is shown in Fig. 3(a).

$$f_{N-1}(FBDR, j, k) \rightarrow f_N(FBDL-1, j, k)$$
$$f_N(FBDL, j, k) \rightarrow f_{N-1}(FBDR+1, j, k)$$
$$1 \le j \le my, 1 \le k \le mz$$
$$f : ex, ey, ez, bx, by, bz$$

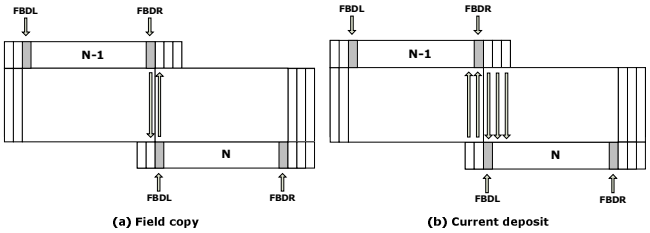

Figure 3 Field communication

Here comes the pseudo code of MPI in FORTRAN that sends *x*-component of the field to the process on the right. The similar approach is followed for *y* and *z*.

```
if(myid.ne.0)then
      call MPI_IRECV(fxr,my*mz, MPI_REAL,myid-1)
else
      call MPI_IRECV(fxr,my*mz,MPI_REAL,Nproc-1)
end if
……
if(myid.ne.(Nproc-1))then
      call MPI_SEND(fxs,my*mz, MPI_REAL, myid+1)
else
      call MPI_SEND(fxs,my*mz,MPI_REAL,0)
end if
call MPI_WAIT
```

Upon updating the electric field, SPLIT calculates not only the particles on [*PBDL*, *PBDR*), but also the ones on the Guard Cell. Firstly, we set the electric fields on the Guard Cell of each process to be zero before computing the current contributed by the particles in the sub-domain of each process. Then, after SPLIT, the electric fields on the Guard Cell of each process are the current contribution to the boundaries of the neighboring processes. Finally, these electric fields will be passed to neighboring processes and then be added to the originally computed values to obtain the corresponding electric fields on the boundaries, as it is shown in Fig. 3(b).

$$f_{N-1}(FBDR+m-2, j, k) = f_{N-1}(FBDR+m-2, j, k) + f_N(FBDL+m-3, j, k)$$
$$f_N(FBDL+n-1, j, k) = f_N(FBDL+n-1, j, k) + f_{N-1}(FBDR+n-1, j, k)$$
$$1 \le j \le my, 1 \le k \le mz$$
$$f : ex, ey, ez, m = 1,2; n = 1,2,3$$

C. The Parallel Procession of Particles

The calculation of particles' velocity and displacement is done by the subroutine MOVER. Each process manipulates the particles whose *x-coordinates* are within [*PBDL*, *PBDR*).

DHD =PBDL-3.0

i= xi(n)-DHD

With the definition of *DHD*, *i* is the relative coordinate value of *n*th particle for the sub-domain of a process, hence the field position where the particle stands is found.

Besides, PARTICLE_SORTING uses some middle variables to store the position and velocity of particles whose *x* position is beyond the range of [*PBDL, PBDR*). And then these variables of one process will be passed to the neighboring process by PARTICLE_PASSING so that each process receives the corresponding particles.

D. The Performance Analysis

In each loop, the computation functions include BPUSH, MOVER, EPUSH, SPLIT and PARTICLE_ESCAPE, all of which constitute the computing part. On the other hand, BCOPY, ECOPY and PARTICLE_SORTING as well as PARTICLE_PASSING are responsible for the communicating. It is necessary to examine both computing time and communicating time. Table I shows a comparison between them in 10 loops for 245x145x145 with 8 pairs of particles per cell.

TABLE I TIME FOR MPI IN 10 LOOPS

	Comp (s)	Comm (s)	Total (s)	Ratio of Comp to Comm	Speedup (times)
1	282	-	282	-	1
2	161.52	6.48	168	24.93	1.68
4	83.5	8.5	92	9.82	3.07
8	43.61	9.39	53	4.64	5.32
16	25.25	9.15	34.4	2.76	8.20

With more processes involved, the computation time decreases to a certain extent and meanwhile the communication increases as there are more data exchanges between processes. And the ratio of the computation time to the communication time drops greatly. To sum up, MPI does not work well with the increasing number of processes. A shared memory model, such as OPENMP, may offer a more efficient parallelization strategy.

III. THE OPTIMIZATION OF PARALLIZED MPI 3DEMPM WITH OPENMP

Unlike MPI, the shared memory model of OPENMP does not have to consider the communication any more. $OMP PARALLEL and $OMP DO can be added outside the loop in order to do the computation concurrently by threads. The subroutines of MOVER and SPLIT are at the core of the computation. Both of them have to access all of the ions and electrons in order to update their new positions or compute the current. So it is essential to parallelize these two subroutines for optimization. The arrays of particles are defined as the SHARED directives while other local variables are defined as FIRSTPRIVATE or PRIVATE. Each thread is responsible for manipulating the same number of ions on average in terms of the total.

```
SUBROUTINE MOVER
!$OMP PARALLEL SHARED (ui,vi,wi,xi,yi,zi)
!$OMP+ FIRSTPRIVATE (ex,ey,ez,bx,by,bz ,ions,……)
!$OMP+ PRIVATE (n,ex0,ey0,ez0,bx0,by0,bz0,……)
!$OMP DO
      do n=1,ions
      …….
      end do
!$OMP END DO
!$OMP END PARALLE
END SUBROUTINE
```

In the SPLIT, the electric field is determined by the surrounding particles not limited to the location of the field. Hence, we need to make a reduction to sum up the current contributed by particles for updating the electric field.

```
SUBROUTINE SPLIT
!$OMP PARALLEL SHARED (ui,vi,wi,xi,yi,zi)
!$OMP+FIRSTPRIVATE (mh, mx,my,mz ,ions, ……)
!$OMP+ PRIVATE (i,j,k,dx,dy,dz,cx,cy,cz,……)
!$OMP+ REDUCTION (+:ex,ey,ez)
!$OMP DO
      do m=1,ions
      …….
      end do
!$OMP END DO
!$OMP END PARALLE
END SUBROUTINE
```

TABLE II TIME FOR OPENMP IN 10 LOOPS

Threads	Total time (s)	Speed-up (times)
1	216	1.00
2	119	1.82
4	71	3.04
8	53	4.08
16	96	2.25

The OPENMP program runs on the hp server with 8 CPUs and can also be accelerated. The performance falls when there are 16 threads as there are only 8 CPUs in each machine. It is known that due to the limited memory of a single machine, OPENMP works poorly when the grid grows in size and more particles are involved. On the other hand, MPI is an explicit control parallelism, with its distributed memory on different nodes in the cluster, solving problems with bigger scale. Hence, a better approach is to take advantages of both MPI and OPENMP. Between the nodes, MPI is applied to realize the communication while OPENMP is responsible for conducting the parallel computation with no communication inside the node.

Again, the MPI domain decomposition is carried out as before with the 3D grid divided between each process on *x* axis. The OPENMP can be added to the MPI within the process as follows:

```
MPI_INIT
!$OMP PARALLEL
……
!$OMP END PARALLEL
```

MPI_FINALIZE

MPI uses the MPI_INIT and MPI_FINALIZE calls to initialize and finalize the process context as usual. In addition, OPENMP PARALLEL DO directives are placed around the loops, generating threads to calculate the position and velocity of particles concurrently in each of the process. The number of threads can be determined by the call of OMP_SET_NUM_THREADS command.

TABLE III TIME FOR MIXED MPI AND OPENMP IN 10 LOOPS

	Comp (s)	Comm (s)	Total (s)	Ratio of Comp to Comm	Speedup (times)
1	282	-	282	-	1
2p1t	164.3	6.48	170.8	25.36	1.65
2p2t	81.52	6.48	88	12.58	3.2
2p4t	43.52	6.48	50	6.72	5.64
4p2t	41	8.5	49.5	4.82	5.7
2p8t	32.32	6.48	38.8	4.99	7.27
4p4t	26.7	8.5	35.2	3.14	8.01
8p2t	22.81	9.39	32.2	2.43	8.76

It is high time to make a comparison of MPI with the mixed mode of MPI and OPENMP from TABLE I and TABLE III.

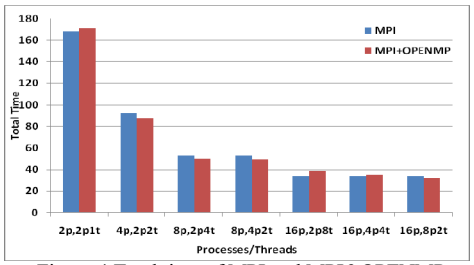

Figure 4 Total time of MPI and MPI&OPENMP

As shown in Fig. 4, generally, the total time of mixed mode drops slightly compared with that of pure MPI, except for running 2p(p: process used by MPI) and 2p1t (t: thread used by OPENMP) when we cannot draw benefits from communication reduction and get overhead of thread context. What's more, when running 16p, we find another three alternatives 2p8t, 4p4t and 8p2t working differently. It seems that the performance is lower when there are more threads involved, and 8p2t is the best among the three. We select the best results, 4p2t and 8p2t, to get Fig. 5.

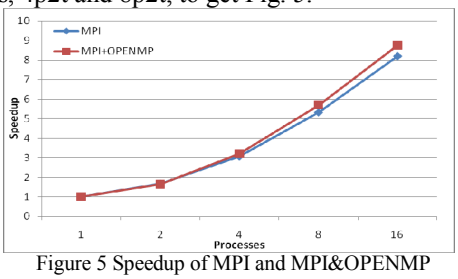

Figure 5 Speedup of MPI and MPI&OPENMP

It can be clearly seen from Fig. 5 that the combination of MPI and OPENMP offers a higher speedup than the MPI alone. The reason can be explained as below. When the program runs with *N* processes on MPI and *N/2* processes with *2* threads on each for mixed MPI and OPENMP, there are *N-1* communications between processes in MPI while only *N/2-1* for the mixed, and meanwhile the computation can also be accelerated by the 2 threads with OPENMP. Therefore, in the mixed mode of MPI and OPENMP, the communication time is reduced. It can also be estimated that when the number of processes *N* increases, the benefits of the combination are more obvious since there will be *(N-1)-(N/2-1)=N/2* of communication time that can be saved.

IV. THE EXPERIMENTAL RESULTS

All of the experiments are carried out on the Linux cluster of 8 personal computers with 8G memory connected by the network. Although MPI can accelerate the execution of 3DEMPM, it involves large communication between each node when many computing machines are added. And OPENMP does not suffer from the load imbalance and there are no explicit communications involved. As a result, when an application needs good scalability, a mixed mode, with MPI between the nodes and OPENMP within each node, can have a better performance since we can reduce the number of processes for less communication time and the load imbalance is the only issue between the nodes.

V. CONCLUSIONS

Based on FDTD and PIC, we propose that the domain is decomposed into several sub-domains and each process calculates on its own sub-domain and communicates with its neighbors to acquire the boundary data. With MPI, 3DEMPM is successfully parallelized on the distributed cluster. Also, OPENMP and the mixed mode of MPI and OPENMP are presented. This combination can reduce the communication time and achieve a better efficiency and speedup than the MPI alone. Further optimization, such as load balancing, and communication reducing, should be taken into account so as to achieve a better performance.

REFERENCES

[1] C K Birdsall, Plasma Physics Via Computer Simulation

[2] R.W Hockney, J.W Eastwood ,Computer Simulation Using Particles (Taylor & Francis,1989)(ISBN 0852743920)

[3] C. Guiffaut and K. Mahdjoubi, France,A Parallel FDTD Algorithm Using the MPI Library,2001.

[4] Dr Peter S. Pacheco,Woo Chat Ming,MPI User Guide in FORTRAN

[5] Louarn, P., A. Fedorov, E. Budnik, et al, Cluster observations of complex 3D magnetic structures at the magnetopause, Geophys. Res. Lett., 31, L19805, doi:10.1029/2004GL020625. 2004.

[6] WeiFeng Tao, DongSheng Cai, XiaoYang Yan, et al, Scalability analysis of parallel PIC codes on computational grids, Computer Physics Communications, 179, 2008

[7] Cai, D. S., Yaoting Li, Ken-Ichi Nishikawa, Chijie Xiao, Xiaoyang Yan and Zuying Pu, Parallel 3D Electromagnetic Particle code using High Performance Fortran: Parallel TRISTAN, Space Plasma Simulation, Springer, 25-53, 2003.

[8] Cai D., Li Y.T., Nishikawa K.. Xiao c.J. and Yan X.Y.: Three-dimensional Electro-magnetic Particle-in-Cell code using High Performance Fortran on PC Cluster, High Performance Computing, Lecture Notes in Computer Science, LNCS 2327, 515-525, 2002.