Regular Paper

Development of the Efficient Electromagnetic Particle Simulation Code with High Performance Fortran on a Vector-Parallel Supercomputer

Hiroki Hasegawa,[†,☆] Seiji Ishiguro[†,††] and Masao Okamoto[†,††]

A three-dimensional, relativistic, electromagnetic particle simulation code is developed using the "Exact Charge Conservation Scheme" on a vector-parallel supercomputer. This scheme is a method for calculating current densities. Applying this method, correction of electric fields is not needed. In this paper, some techniques to optimize the above scheme for a vector-parallel supercomputer are shown. The method for vectorization and parallelization in shared memory and in distributed memories are discussed. The code is written in Fortran90 and High Performance Fortran (HPF). Examination of this code is also made.

1. Introduction

The Particle-in-Cell (PIC) method is a common simulation scheme which has been used to study plasma physics [1] (mainly, laser plasmas, space plasmas, and so on). In this method, motions of individual particles and time evolutions of the self-consistent fields are computed alternately. In relativistic, electromagnetic (EM) PIC codes, first, charge and current densities on spatial grids (cells) are calculated from particle positions and velocities. Then, substituting these charge and current densities in Maxwell equations, the self-consistent electric and magnetic fields at the next time step are obtained. Finally, using these fields, particle velocities and positions are advanced by the relativistic equation of motion. This cycle is continued for simulation time.

Although there are several methods for calculating time evolutions of the fields, finite-difference Maxwell equations are often used. In this case, however, one has to correct the electric field by the Gauss law (Poisson equation) (Ref. 1) and the references therein). Also, this means that the continuity equation is not held rigorously.

Therefore, using the "Exact Charge Conservation Scheme [2),3)]," we develop the three-dimensional, relativistic EM-PIC code. In this scheme, current densities are computed with rigorous satisfaction of the finite-difference continuity equation. Then, from Ampère's law in finite differences, we find that the Gauss law is also held. Thus, applying this algorithm as scheme to calculate current densities, one needs no correction process. Further, because of the accuracy of current densities, in an open system simulation code [4)], consistent boundary conditions are given.

This paper describes a coding method which optimize the above scheme for a vector-parallel supercomputer. In actual calculation, we use the SX-7 supercomputer (NEC Corporation), which is used at the Theory and Computer Simulation Center, National Institute for Fusion Science (NIFS). Since this machine has five nodes (one node has 32 processors), we have to use not only automatic parallelization in shared memory on each node but also manual parallelization in distributed memories among nodes. For this purpose, we write simulation code with High Performance Fortran (HPF) which is a distributed parallel processing language for Fortran [5)].

In Section 2, we show an outline of our PIC code. Locations where field components are defined on a spatial grid and the flow of calculation are discussed. In Section 3, we review essential points of the "Exact Charge Conservation Scheme." It is explained how current densities are obtained in this scheme. In Section 4, we present the actual source code written in HPF. Further, methods for parallelization and vectorization are discussed. In Section 5, the performance of our code is examined. In Section 6, we summarize our work.

2. Overview of PIC Code

In our code, time evolutions of the fields are

† Theory and Computer Simulation Center, National Institute for Fusion Science
†† The Graduate University for Advanced Studies (Soken-dai)
☆ Presently with Earth Simulator Center, Japan Agency for Marine-Earth Science and Technology

calculated by Ampère's law and Faraday's law in finite differences. **Figure 1** shows locations where the electric field $\boldsymbol{E}$, the magnetic field $\boldsymbol{B}$, the current density $\boldsymbol{J}$, and the charge density ρ are defined on a cell. Using these definitions, we obtain Ampère's law as

$$\frac{1}{c}\frac{E^{x\,n+1}_{\alpha,\beta,\gamma}-E^{x\,n}_{\alpha,\beta,\gamma}}{\Delta t} = \left(\frac{B^{z\,n+1/2}_{\alpha,\beta+1,\gamma}-B^{z\,n+1/2}_{\alpha,\beta,\gamma}}{\Delta y} - \frac{B^{y\,n+1/2}_{\alpha,\beta,\gamma+1}-B^{y\,n+1/2}_{\alpha,\beta,\gamma}}{\Delta z}\right) - \frac{4\pi}{c}J^{x\,n+1/2}_{\alpha,\beta,\gamma}, \tag{1}$$

and Faraday's law as

$$\frac{1}{c}\frac{B^{x\,n+1/2}_{\alpha,\beta,\gamma}-B^{x\,n-1/2}_{\alpha,\beta,\gamma}}{\Delta t} = -\left(\frac{E^{z\,n}_{\alpha,\beta,\gamma}-E^{z\,n}_{\alpha,\beta-1,\gamma}}{\Delta y} - \frac{E^{y\,n}_{\alpha,\beta,\gamma}-E^{y\,n}_{\alpha,\beta,\gamma-1}}{\Delta z}\right), \tag{2}$$

where c is the speed of light, Δt is the time step, Δx, Δy, and Δz are the grid spacing, the subscripts α, β, and γ denote the integer indices of cells, and the index n is the integer time step. Also, Eqs. (1) and (2) represent only the x component of original equations. The other components are given by cyclic substitution of space coordinates.

Further, velocities and positions of particles are derived via the relativistic equation of motion,

$$\frac{\boldsymbol{p}^{n+1/2}_s-\boldsymbol{p}^{n-1/2}_s}{\Delta t} = q_s\sum_{\alpha,\beta,\gamma} S(x_\alpha-x_s)\,S(y_\beta-y_s)\,S(z_\gamma-z_s) \times\left(\boldsymbol{E}^n_{\alpha,\beta,\gamma} + \frac{(\boldsymbol{p}^{n+1/2}_s+\boldsymbol{p}^{n-1/2}_s)\times\boldsymbol{B}^n_{\alpha,\beta,\gamma}}{2m_s\Gamma^n_s c}\right), \tag{3}$$

and

$$\frac{\boldsymbol{x}^{n+1}_s-\boldsymbol{x}^n_s}{\Delta t} = \frac{\boldsymbol{p}^{n+1/2}_s}{m_s\Gamma^{n+1/2}_s}, \tag{4}$$

where m is the mass, $\boldsymbol{p}$ is the momentum, q is the charge, S is the form-factor of the finite-size particle [1], $\boldsymbol{x}_{\alpha,\beta,\gamma}$ is the position of grid, $\boldsymbol{x}_s$ is the particle position, Γ is the Lorentz factor, and the subscript s denotes the particle number including the particle species.

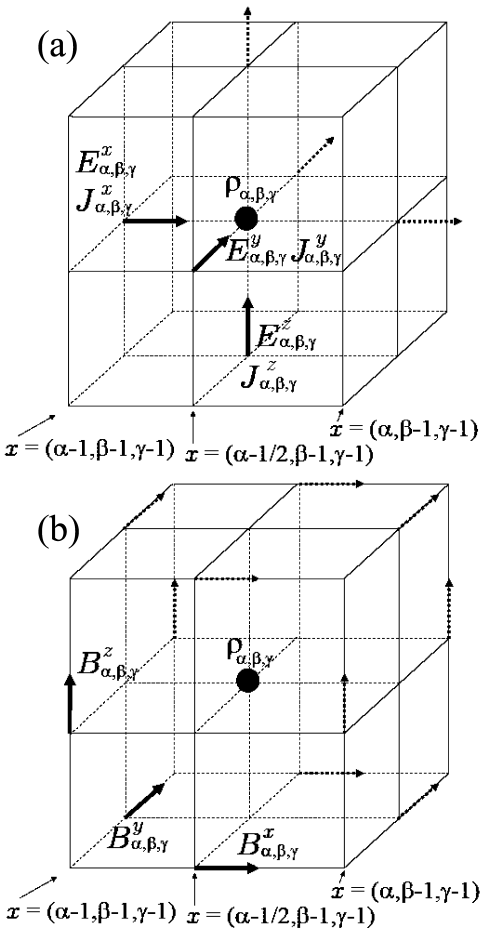

Fig. 1 (a) Locations where the electric field $\boldsymbol{E}$, the current density $\boldsymbol{J}$, and the charge density ρ are defined on a cell, and (b) those of the magnetic field $\boldsymbol{B}$.

The flow of calculation is presented in **Fig. 2**. First, using momenta, particle positions for the next time step are obtained. Then, from the old (time index n) and the new (time index $n+1$) positions, current densities are computed via the "Exact Charge Conservation Scheme." (This calculation is described in detail in Ref. 3) or the following section.) Next, substituting these current densities and magnetic fields in Eq. (1), we have the new electric fields. And then, using these electric fields, the new magnetic fields are given by Eq. (2). Finally, substituting the new fields into Eq. (3), particle momenta are advanced. Here, we note that par-

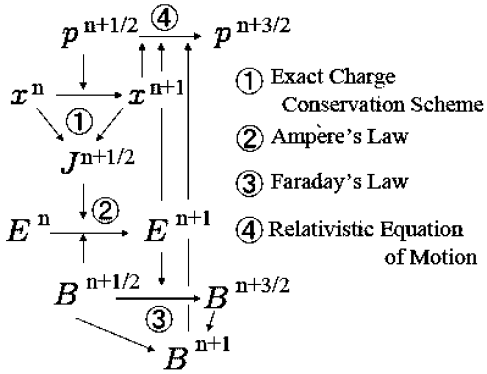

Fig. 2 Flow of calculation.

ticle positions and electric fields are defined at integer time step and that, on the other hand, the time steps for particle momenta, current densities, and magnetic fields are half-integer. Such a calculation scheme is called "the leap-frog method [1]."

3. Method for Calculating the Current Density

Now, we briefly review the "Exact Charge Conservation Scheme [2),3)]." In this scheme, to avoid correction of electric fields by the Gauss law, current densities are calculated with rigorous satisfaction of the finite-difference continuity equation,

$$\frac{\rho^{n+1}_{\alpha,\beta,\gamma}-\rho^{n}_{\alpha,\beta,\gamma}}{\Delta t}+\frac{J^{x\,n+1/2}_{\alpha+1,\beta,\gamma}-J^{x\,n+1/2}_{\alpha,\beta,\gamma}}{\Delta x} +\frac{J^{y\,n+1/2}_{\alpha,\beta+1,\gamma}-J^{y\,n+1/2}_{\alpha,\beta,\gamma}}{\Delta y} +\frac{J^{z\,n+1/2}_{\alpha,\beta,\gamma+1}-J^{z\,n+1/2}_{\alpha,\beta,\gamma}}{\Delta z}=0. \quad (5)$$

For this purpose, we use a vector $\boldsymbol{W}$,

$$W^{x\,n+1/2}_{\alpha,\beta,\gamma} = -(J^{x\,n+1/2}_{\alpha+1,\beta,\gamma}-J^{x\,n+1/2}_{\alpha,\beta,\gamma})\frac{1}{q}\frac{\Delta t}{\Delta x}. \quad (6)$$

We now consider a contribution of one particle to $\boldsymbol{W}$. Assuming that S is the second-order form-factor, a vector $\boldsymbol{W}$ is given as

$$W^{x\,n+1/2}_{\alpha+i,\beta+j,\gamma+k} = DS^x_i \times\left(S^y_j S^z_k+\frac{1}{2}DS^y_j S^z_k +\frac{1}{2}S^y_j DS^z_k+\frac{1}{3}DS^y_j DS^z_k\right). \quad (7)$$

Here, α, β, and γ are the indices of cell which is the nearest to an old particle position, the subscripts i, j, and k are integral numbers from -2 to 2, S^y_j denotes $S(y_{\beta+j}-y^n_s)$, and DS is the difference between the old and the new S,

$$DS^x_i = S(x_{\alpha+i}-x^{n+1}_s)-S(x_{\alpha+i}-x^n_s). \quad (8)$$

Computing a vector $\boldsymbol{W}$ with the aid of Eq. (7), we have current densities by Eq. (6).

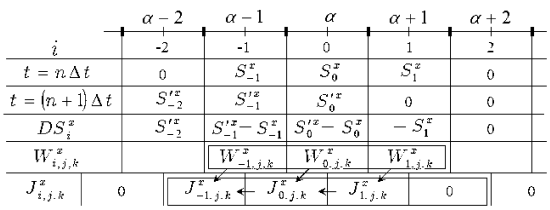

Fig. 3 Schematic diagram showing the order to calculate the current densities. Here, the new nearest index is $\alpha-1$, where α is the old one.

4. Subroutine of Calculating the Current Density

We then discuss a coding technique which optimize the "Exact Charge Conservation Scheme" for a vector-parallel supercomputer. In Section 4.1, we consider how many elements for each variable are necessary to the subroutine. In Section 4.2, we show a calculation flow in the subroutine. In Section 4.3, we explain methods for parallelization and vectorization.

4.1 Number of Elements for W and J

We assume that the index of the new nearest cell is $\alpha-1$. (Because $c\,\Delta t<\Delta x$, its decrease (or increase) is zero or one.) In this case, we obtain S and DS as shown in **Fig. 3**. Here, for the sake of simplicity, we omit the subscripts α, β, and γ. Although, from Eq. (7), we find that $W^x_{i,j,k}$ has a nonzero value at the point where DS is not zero, it is enough to calculate just three elements for $W^x_{i,j,k}$.

Since a part of the charge does not pass through the surface between $\alpha+1$ and $\alpha+2$ cells, we have $J^x_{2,j,k}=0$. Thus, from only $W^x_{1,j,k}$, we find $J^x_{1,j,k}$ by Eq. (6). Then, following the order as arrow in Fig. 3, $J^x_{i,j,k}$ is given via Eq. (6). Hence, it is sufficient to prepare three elements for $W^x_{i,j,k}$ $(i=-1,0,1)$.

Figures 4 and **5** show other situations; the index of the new nearest cell is α or $\alpha+1$. In all cases, it is enough to prepare only three elements for $W^x_{i,j,k}$ and four elements for $J^x_{i,j,k}$ $(i=-1,0,1,2)$ for calculating current densities. The order to compute $J^x_{i,j,k}$, however, is

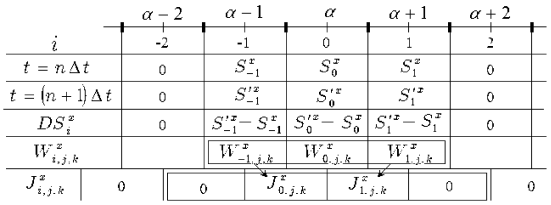

Fig. 4 Schematic diagram showing the order to calculate the current densities. Here, the new nearest index is α.

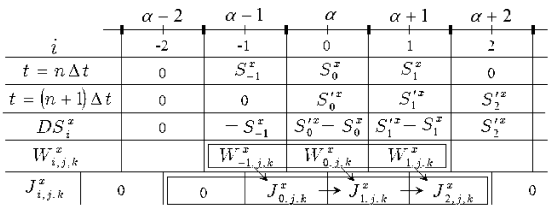

Fig. 5 Schematic diagram showing the order to calculate the current densities. Here, the new nearest index is $\alpha + 1$.

different in each case. Then, decisions which situations individual particles produce are usually taken with the IF statement.

On the other hand, with respect to j (or k), we find that only four elements for $W^x_{i,j,k}$ and $J^x_{i,j,k}$ are needed. Assuming that the index of the new nearest cell for the y component is $\beta-1$, we have $S^y_2 = 0$ and $DS^y_2 = 0$. Hence, from Eq. (7), we obtain $W^x_{i,2,k} = 0$ and $J^x_{i,2,k} = 0$. It is thus concluded that the elements for $j = 2$ are not necessary. We, however, note that elements used in calculation are different in situations. Therefore, in the loop for $j = -1, 2$ (or $k = -1, 2$), one has to modify the index j (or k) about all particles.

By the above methods, the total of calculations can be reduced.

4.2 Calculation Flow in Subroutine

In the subroutine, calculation is done along the following arrangement;

Large Loop 1

Loop 1

Step 1.1
Find the index number of the nearest cell to $\boldsymbol{x}_s^n$.

Step 1.2
Compute $\boldsymbol{x}_s^{n+1}$.

Step 1.3
Find the index number of the nearest cell to $\boldsymbol{x}_s^{n+1}$.

Step 1.4
Difference between the old and the new index number is given.

Step 1.5
Check the value of the index difference.

Step 1.6
Calculate S and DS.

Loop 2

Sub Loop A

Step A.1
$W_{-1,j,k} \sim W_{1,j,k}$ are obtained from S and DS.

Step A.2
Using $W_{i,j,k}$, $J_{i,j,k}$ is computed.

Sub Loop B

Step B.1
$J_{i,j,k}$ is added to work arrays for current densities.

Large Loop 2
Obtain current densities on grids by summation of work arrays.

Large Loop 1 is a double loop which has lengths of the total number of processors (`np`) and blocks (`nb`). Loop 1 is a single loop with a maximum vector register length (`veclen`) of 256. Here, the total number of simulation particles is taken to be `np` × `nb` × `veclen`. Further, Loop 2 is a double loop, $j, k = -1, 2$. Sub Loop A and B are single loop which has a maximum vector register length. The actual source code in Fortran90 and HPF is shown in Appendix.

4.3 Optimization

4.3.1 Parallelization

This code is parallelized in distributed memories with HPF and in shared memory with an automatic parallelizing function of sxf90 compiler, which is a Fortran90 compiler for the SX supercomputer. In PIC simulation, the total calculation for particles is more than 100 times as large as the one for field data.

In our code, subroutines for calculating current densities (subroutine CURRENT) and advancing particle momenta (subroutine PUSH) include the computations about particles. Hence, we now especially optimize these subroutines.

First, arrays for particle data (positions, momenta) are distributed to each HPF process. The extent of last rank for these arrays is taken to be `np`. Then, inserting DISTRIBUTE directive lines (`!HPF$ distribute`) to the code as shown in Appendix, they are distributed with respect to the last rank.

Further, in an HPF process, calculations are

automatically parallelized in shared memory by the sxf90 compiler. We take the total number of generated tasks to be `np` / `npro`, where `npro` is the total number of HPF processes.

In the subroutine, we write the INDEPENDENT directive line (`!HPF$ independent`) and the sxf90 compiler directive to force parallelization (`!cdir parallel do`)[6] immediately before the DO statement for Large Loop 1. Then, this loop is parallelized in both distributed memories and shared memory. Also, Large Loop 2 is parallelized in distributed memories.

4.3.2 Vectorization

Aiming at effective vectorization, we set the loop count of the innermost loop at a maximum vector register length and use no IF statement; instead, we prepare the array `cx` to check the particle moving direction in Step 1.5. If the nearest index difference is `l` ($= -1, 0$, or 1), `cx(iv,l)` $= 1.0$, and other components are zero. Further, we add some directive lines above these loops, to promote vectorization. Applying this way, we vectorize Loop 1 and Sub Loop A. (Directive lines shown in Appendix are explained in detail in Ref. 7).)

On the other hand, Sub Loop B is generally not vectorized, due to uncertainty of dependency between a particle number and an index of its nearest cell. Then, although work arrays, whose first rank extent is equal to the loop count, have been used to vectorize this loop, this vectorization scheme needs large memories. In this subroutine, however, using the LISTVEC compiler directive (`!cdir listvec`), we can vectorize Sub Loop B without these work arrays. (This directive option is described in detail in Refs. 7) and 8).)

5. Examination of Code

We then examine the accuracy and the performance of this code. The test simulation parameters are as follows. The total number of cells in system is $64 \times 64 \times 64$. Both the numbers of electrons and ions per cell are 216; that is, the total number of simulation particles is 113,246,208. The test calculations have been done for 100 time steps.

Figure 6 presents a time variation of the quantity dQ, where dQ is defined as

$$dQ = \sum_{\alpha,\beta,\gamma} \left(\frac{\rho^{n+1}_{\alpha,\beta,\gamma} - \rho^{n}_{\alpha,\beta,\gamma}}{\Delta t} + \frac{J^{x\,n+1/2}_{\alpha+1,\beta,\gamma} - J^{x\,n+1/2}_{\alpha,\beta,\gamma}}{\Delta x} + \frac{J^{y\,n+1/2}_{\alpha,\beta+1,\gamma} - J^{y\,n+1/2}_{\alpha,\beta,\gamma}}{\Delta y} + \frac{J^{z\,n+1/2}_{\alpha,\beta,\gamma+1} - J^{z\,n+1/2}_{\alpha,\beta,\gamma}}{\Delta z} \right)^2 . \quad (9)$$

It is found that the continuity equation have been held over simulation time in a range of round-off error.

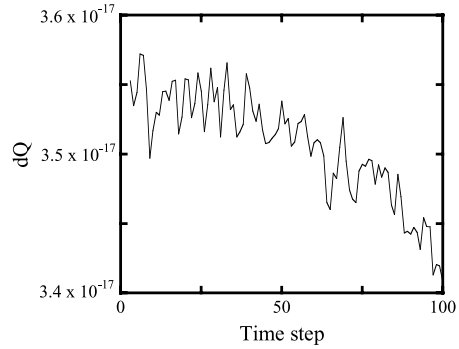

Fig. 6 Time variation of the quantity dQ.

Table 1 Results of test runs with the LISTVEC directive. Here, the total number of generated tasks is 32.

	RUN1	RUN2	RUN3
Total PE (`np`)	32	64	96
node	1	2	3
HPF process (`npro`)	1	2	3
Real Time (sec.)	915.24	470.10	328.72
nsec/step/particle	80.8	41.5	29.0
Conc. Time (PE$\geq$ 1)	888.30	455.32	318.38
Conc. Time (PE= 32)	834.01	431.83	294.25
Memory Size (GB)	10.9	14.6	18.2
Vector Length	255.4	254.8	254.2
Vector Ratio (%)	99.92	99.91	99.89
Execution Time for subroutines			
CURRENT	666.22	342.52	239.70
PUSH	221.51	110.55	74.42
For Large Loop 1 in subroutine CURRENT			
Vector Length	256.0	256.0	256.0
Vector Ratio (%)	99.93	99.93	99.93

Table 1 shows the results of test runs in which the total numbers of tasks are `np` / `npro` = 32, while **Table 2** displays those of other runs in which `np` / `npro` = 16. In these tables, we find that the calculation time is in inverse proportion to the total number of HPF processes. This fact indicates that the distributed parallelization with HPF is very successful.

We show, in **Fig. 7**, the dependence of the

Table 2 Results of test runs with the LISTVEC directive. Here, the total number of generated tasks is 16.

	RUN4	RUN5	RUN6
Total PE (np)	16	48	144
node	1	3	5
HPF process (npro)	1	3	9
Real Time (sec.)	1860.32	582.87	224.06
nsec/step/particle	164.3	51.5	19.8
Conc. Time (PE≥ 1)	1834.86	572.59	216.30
Conc. Time (PE= 16)	1817.12	557.67	202.67
Memory Size (GB)	10.8	14.8	26.9
Vector Length	255.7	255.0	253.1
Vector Ratio (%)	99.93	99.92	99.88
Execution Time for subroutines			
CURRENT	1373.79	425.39	163.46
PUSH	460.54	135.25	49.16
For Large Loop 1 in subroutine CURRENT			
Vector Length	256.0	256.0	256.0
Vector Ratio (%)	99.94	99.93	99.93

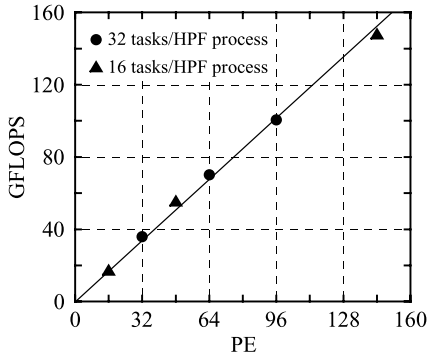

Fig. 7 Dependence of the computing performance (GFLOPS) on the total number of processors (PE). Here, the circles are the observed values in RUN1∼3 which appear in Table 1, the triangles represent those in RUN4∼6 which are presented in Table 2, and the solid line denotes the value of 12% of theoretical peak performance.

computing performance (GFLOPS) on the total number of processors. The value of FLOPS linearly increases with the total number of processors.

Also, Table 1 shows that the difference between concurrent times (Conc. Time) with one processor or over, and with 32 processors, is quite small. This means that the code achieves the highly parallelized state in shared memory. Since subroutines CURRENT and PUSH occupy most of the calculation time as shown in the table of execution time, the performance of this code is highly improved by only parallelization of these subroutines.

Further, the observed vector length is almost equal to a maximum vector register length, and vector ratios are close to 100 percents.

Table 3 Results of test runs with the work arrays.

	RUN7	RUN8	RUN9
Total PE (np)	32	64	96
node	1	2	3
HPF process (npro)	1	2	3
Real Time (sec.)	756.54	445.15	344.92
nsec/step/particle	66.8	39.3	30.5
Conc. Time (PE≥ 1)	729.06	428.72	331.80
Conc. Time (PE= 32)	672.41	405.53	308.30
Memory Size (GB)	68.6	130.0	191.4
Vector Length	228.0	207.8	192.5
Vector Ratio (%)	99.74	99.57	99.41
Execution Time for subroutines			
CURRENT	506.85	317.64	255.49
PUSH	223.46	110.77	74.03
For Large Loop 1 in subroutine CURRENT			
Vector Length	256.0	256.0	256.0
Vector Ratio (%)	99.94	99.94	99.94

Table 3 shows the results for the case using work arrays instead of the LISTVEC directive line. From comparison between Tables 1 and 3, it is found that the required memory size decreases with LISTVEC directive.

6. Summary

We have developed a three-dimensional, relativistic, electromagnetic particle simulation code that uses the "Exact Charge Conservation Scheme" on a vector-parallel supercomputer. Although PIC codes with this scheme is more useful than those with conventional EM-PIC methods due to the accuracy of current densities (especially, in an open system code), we further have succeeded in optimization of this scheme for vector-parallel supercomputers.

In parallelization, in order to make the code adequate at multi-node jobs, we have used not only an automatic parallelizing function of the sxf90 compiler in shared memory but also HPF for distributed parallelization. In vectorization, inserting the LISTVEC directive line, the required memory size is saved.

Test calculations indicate that parallelization and vectorization are successful. Thus, our code will make plasma particle simulation more efficient and fast.

Acknowledgments This work is performed with the support and under the auspices of the NIFS Collaborative Research Program (NIFS04KDAT007) and supported in part by a Grant-in-Aid for Scientific Research from the Ministry of Education, Culture, Sports, Science

and Technology.

The authors are grateful to Prof. R. Horiuchi (NIFS), Dr. H. Ohtani (NIFS), and Mr. N. Horiuchi (NEC) for stimulating discussions.

References

1) Birdsall, C. K. and Langdon, A. B.: *Plasma Physics via Computer Simulation*, Adam Hilger (1991).
2) Villasenor, J. and Buneman, O.: Rigorous charge conservation for local electromagnetic field solvers, *Comput. Phys. Comm.*, Vol.69, pp.306–316 (1992).
3) Esirkepov, T. Zh.: Exact charge conservation scheme for Particle-in-Cell simulation with an arbitrary form-factor, *Comput. Phys. Comm.*, Vol.135, pp.144–153 (2001).
4) Takamaru, H., et al.: A Self-Consistent Open Boundary Model for Particle Simulation in Plasmas, *J. Phys. Soc. Jpn.*, Vol.66, pp.3826–3836 (1997).
5) High Performance Fortran Forum: *High Performance Fortran Language Specification Version 2.0* (1997); translation to Japanese: Springer-Verlag Tokyo (1999).
6) NEC Corporation: *FORTRAN90/SX Multitasking User's Guide*, NEC Corporation, Revision No.11 (2002).
7) NEC Corporation: *FORTRAN90/SX Programmer's Guide*, NEC Corporation, Revision No.13 (2002).
8) Sugiyama, T., et al.: Vectorized Particle Simulation Using "LISTVEC" Compile-directive on SX Super-computer, *IPSJ Trans. on Advanced Computing Systems* (in Japanese), Vol.45, No.SIG6 (ACS6), pp.171–175 (2004).

Appendix

We here show the actual source code of subroutine of calculating current densities. This is written in Fortran90 and HPF. The variables and arrays in the code denote as follows. The arrays `x`, `px`, and `lorentz` are the particle position, momentum, and Lorentz factor, respectively. The array `curx` represents the current density on a grid. The arrays `jxm`, `jx0`, `jx1`, and `jx2` are work arrays for the current densities. The array `alpha0` denotes the old nearest index. The variable `x1` is the new particle position. The variable `alpha1` is the new nearest index. The array `ldx` represents the nearest index difference. The arrays `sx0(iv,i)` and `dsx(iv,i)` are S_i^x and DS_i^x, respectively. The array `wx(i)` denotes $W_{i,j,k}^x$. The array `jx(iv,i)` is $J_{i,j,k}^x$.

Also, in this code, the form-factor is given as

$$S(x_\alpha - x_s) = \frac{3}{4} - (x_\alpha - x_s)^2, \tag{10}$$

$$S(x_{\alpha\pm1} - x_s) = \frac{1}{2}\left(\frac{1}{2} \mp (x_\alpha - x_s)\right)^2. \tag{11}$$

```
!-------------------------------------
 subroutine current(x,px,...,curx,...)
!-------------------------------------
!hpf$ processors pro(npro)
        :
        :
        :
!hpf$ distribute (*,block) onto pro   &
!hpf$& :: x,px,y,py,z,pz,lorentz
!hpf$ distribute (*,*,*,block)        &
!hpf$& onto pro :: jxm,jx0,jx1,jx2
!-------------------------------------
! Large Loop 1
!-------------------------------------
!hpf$ independent, new(j,k,ii,...)
!cdir parallel do
!cdir& private(j,k,ii,...)
 do ip = 1, np
 do ii = 1, nb
!-------------------------------------
! Loop 1
!-------------------------------------
!cdir nodep
!cdir noinner
!cdir shortloop
 do iv = 1, veclen
   m  = iv + (ii-1)*veclen
!-------------------------------------
! Step 1.1
!-------------------------------------
   alpha0(iv) = int( x(m,ip) + 1.0d0 )
   beta0(iv)  = int( y(m,ip) + 1.0d0 )
   gamma0(iv) = int( z(m,ip) + 1.0d0 )
!-------------------------------------
! Step 1.2
!-------------------------------------
   x1 = x0+dt*px(m,ip)/lorentz(m,ip)
   y1 = y0+dt*py(m,ip)/lorentz(m,ip)
   z1 = z0+dt*pz(m,ip)/lorentz(m,ip)
!-------------------------------------
! Step 1.3
!-------------------------------------
   alpha1 = int( x1 + 1.0d0 )
   beta1  = int( y1 + 1.0d0 )
   gamma1 = int( z1 + 1.0d0 )
```

```
!-------------------------------------
! Step 1.4
!-------------------------------------
   ldx(iv) = alpha1 - alpha0(iv)
   ldy(iv) = beta1  - beta0(iv)
   ldz(iv) = gamma1 - gamma0(iv)
!-------------------------------------
! Step 1.5 (write only the x component)
!-------------------------------------
   cx(iv,-1) =                          &
        dble( ldx(iv)*(ldx(iv)-1)/2 )
   cx(iv, 0) =                          &
        dble((1-ldx(iv))*(1+ldx(iv)))
   cx(iv, 1) =                          &
        dble( ldx(iv)*(ldx(iv)+1)/2 )
         :
         :
         :
!-------------------------------------
! Step 1.6 (write only the x component)
!-------------------------------------
   dx0 = dble(alpha0(iv)) - 0.5d0 - x0
   dx1 = dble(alpha1)     - 0.5d0 - x1
   sx0(iv,-2) = 0.0d0
   sx0(iv,-1) = 0.5d0*(0.5d0+dx0)**2
   sx0(iv, 0) = 0.75d0 - dx0*dx0
   sx0(iv, 1) = 0.5d0*(0.5d0-dx0)**2
   sx0(iv, 2) = 0.0d0
   sx1(-1) = 0.5d0*(0.5d0+dx1)**2
   sx1( 0) = 0.75d0 - dx1*dx1
   sx1( 1) = 0.5d0*(0.5d0-dx1)**2
   dsx(iv,-2) =  cx(iv,-1)*sx1(-1)
   dsx(iv,-1) =  cx(iv,-1)*sx1( 0)    &
               + cx(iv, 0)*sx1(-1)    &
               - sx0(iv,-1)
   dsx(iv, 0) =  cx(iv,-1)*sx1( 1)    &
               + cx(iv, 0)*sx1( 0)    &
               + cx(iv, 1)*sx1(-1)    &
               - sx0(iv, 0)
   dsx(iv, 1) =  cx(iv, 0)*sx1( 1)    &
               + cx(iv, 1)*sx1( 0)    &
               - sx0(iv, 1)
   dsx(iv, 2) =  cx(iv, 1)*sx1( 1)
         :
         :
         :
 end do
!-------------------------------------
! Loop 2
!-------------------------------------
!cdir novector
!cdir noconcur
 do k = -1, 2
!cdir novector
!cdir noconcur
 do j = -1, 2
!-------------------------------------
! Sub Loop A
!-------------------------------------
!cdir nodep
!cdir noinner
!cdir shortloop
 do iv = 1, veclen
!-------------------------------------
!  Modify j and k
!  with the aid of the index difference
!-------------------------------------
   jj = j - ldy(iv)*(ldy(iv)-1)/2
   kk = k - ldz(iv)*(ldz(iv)-1)/2
!-------------------------------------
! Step A.1
!-------------------------------------
   wx(-1) = dsx(iv,-1)*                 &
       ( sy0(iv,jj)*sz0(iv,kk)          &
       + dsy(iv,jj)*sz0(iv,kk)*0.5d0    &
       + sy0(iv,jj)*dsz(iv,kk)*0.5d0    &
       + dsy(iv,jj)*dsz(iv,kk)/3.0d0 )
   wx( 0) = dsx(iv, 0)*                 &
       ( sy0(iv,jj)*sz0(iv,kk)          &
       + dsy(iv,jj)*sz0(iv,kk)*0.5d0    &
       + sy0(iv,jj)*dsz(iv,kk)*0.5d0    &
       + dsy(iv,jj)*dsz(iv,kk)/3.0d0 )
   wx( 1) = dsx(iv, 1)*                 &
       ( sy0(iv,jj)*sz0(iv,kk)          &
       + dsy(iv,jj)*sz0(iv,kk)*0.5d0    &
       + sy0(iv,jj)*dsz(iv,kk)*0.5d0    &
       + dsy(iv,jj)*dsz(iv,kk)/3.0d0 )
!-------------------------------------
! Step A.2
!-------------------------------------
   jx(iv,-1) = (1/dt)*                  &
       cx(iv,-1)*(wx(-1)+wx(0)+wx(1))
   jx(iv, 0) = (1/dt)*                  &
      (cx(iv,-1)*(       wx(0)+wx(1)) &
      -cx(iv, 0)*(wx(-1)             ) &
      -cx(iv, 1)*(wx(-1)             ))
   jx(iv, 1) = (1/dt)*                  &
      (cx(iv,-1)*(              wx(1)) &
      +cx(iv, 0)*(              wx(1)) &
      -cx(iv, 1)*(wx(-1)+wx(0)       ))
   jx(iv, 2) = (1/dt)*                  &
      -cx(iv, 1)*(wx(-1)+wx(0)+wx(1))
 end do
!-------------------------------------
! Sub Loop B
!-------------------------------------
!cdir listvec
 do iv = 1, veclen
```

```
   a0 = alpha0(iv)
   b0 = beta0(iv)
   c0 = gamma0(iv)
!-------------------------------------
! Modify j and k
! with the aid of the index difference
!-------------------------------------
   jj = j -ldy(iv)*(ldy(iv)-1)/2
   kk = k -ldz(iv)*(ldz(iv)-1)/2
!-------------------------------------
! Step B.1
!-------------------------------------
   jxm(a0-1,b0+jj,c0+kk,ip) =         &
            jxm(a0-1,b0+jj,c0+kk,ip) &
                       + jx(iv,-1)
   jx0(a0,  b0+jj,c0+kk,ip) =         &
            jx0(a0,  b0+jj,c0+kk,ip) &
                       + jx(iv, 0)
   jx1(a0+1,b0+jj,c0+kk,ip) =         &
            jx1(a0+1,b0+jj,c0+kk,ip) &
                       + jx(iv, 1)
   jx2(a0+2,b0+jj,c0+kk,ip) =         &
            jx2(a0+2,b0+jj,c0+kk,ip) &
                       + jx(iv, 2)
 end do
 end do
 end do
 end do
 end do
!-------------------------------------
! Large Loop 2
!-------------------------------------
!hpf$ independent, new(i,j,k),        &
!hpf$& reduction(curx)
 do ip = 1, np
 do k = -1, nz+2
 do j = -1, ny+2
!cdir nodep
 do i =  0, nx+2
   curx(i,j,k) = curx(i,j,k)          &
      + jxm(i,j,k,ip) + jx0(i,j,k,ip) &
      + jx1(i,j,k,ip) + jx2(i,j,k,ip)
 end do
 end do
 end do
 end do
```

(Received April 28, 2005)
(Accepted August 13, 2005)

Hiroki Hasegawa received his Ph.D. degree from Nagoya University in 2004. Since 2004 until 2005 he had been in National Institute for Fusion Science (NIFS) as a COE researcher. He is now a research scientist of the Earth Simulator Center of Japan Agency for Marine-Earth Science and Technology (JAMSTEC). He has engaged in the research of particle acceleration in plasmas. Also, his current research interest is computer simulation on a vector-parallel supercomputer. He is a member of JPS.

Seiji Ishiguro received his Ph.D. degree from Nagoya University in 1987. He has worked for Tohoku University and National Institute for Fusion Science since 1987. He is now a Professor of National Institute for Fusion Science. His current research interests are potential formation in plasmas, laser-plasma interaction, and simulation science. He is a member of JPS and JSPF.

Masao Okamoto received his Ph.D. degree from Kyoto University in 1971. Since then, he has worked for Osaka University, Japan Atomic Energy Research Institute (JAERI), Institute of Plasma Physics of Nagoya University, and National Institute for Fusion Science (NIFS). He is now Professor and the Director of Theory and Computer Simulation Center at NIFS. His special fields are plasma physics, fusion science, and simulation science. His recent interest is multi-scale simulations for toroidal plasmas. He is a member of APS, JPS, and JSPF.