A Parallel Implementation of Particle Tracking with Space Charge Effects on an Intel iPSC/860

L. CHANG, G. BOURIANOFF, B. COLE, and S. MACHIDA

Superconducting Super Collider Laboratory, 2550 Beckleymeade Avenue, Dallas, TX 75237*

ABSTRACT

Particle-tracking simulation is one of the scientific applications that is well suited to parallel computations. At the Superconducting Super Collider, it has been theoretically and empirically demonstrated that particle tracking on a designed lattice can achieve very high parallel efficiency on a MIMD Intel iPSC/860 machine. The key to such success is the realization that the particles can be tracked independently without considering their interaction. The perfectly parallel nature of particle tracking is broken if the interaction effects between particles are included. The space charge introduces an electromagnetic force that will affect the motion of tracked particles in three-dimensional (3-D) space. For accurate modeling of the beam dynamics with space charge effects, one needs to solve 3-D Maxwell field equations, usually by a particle-in-cell (PIC) algorithm. This will require each particle to communicate with its neighbor grids to compute the momentum changes at each time step. It is expected that the 3-D PIC method will degrade parallel efficiency of particle-tracking implementation on any parallel computer. In this paper, we describe an efficient scheme for implementing particle tracking with space charge effects on an INTEL iPSC/860 machine. Experimental results show that a parallel efficiency of 75% can be obtained.

1 INTRODUCTION

The superconducting super collider (SSC) under design and construction near Waxahachie, Texas, will be the largest and most powerful scientific instrument ever built and will be used to investigate the fundamental origins of matter. The SSC complex consists of the linear accelerator (LINAC), low-energy booster (LEB), medium-energy booster (MEB), high-energy booster (HEB), and two collider rings, which will have a circumference of 54 miles (87 km) (Fig. 1). Each accelerator consists of an array of elements, mostly magnetic. Particles are first injected from the LINAC, then travel circularly in the LEB, the MEB, and the HEB, and finally in the collider for many turns while being accelerated. At the final stage, two bunches of particles at 20 TeV, traveling in the 87-km collider rings in opposite directions, are brought into head-on collision.

Because the construction cost of the SSC is ex-

* The Superconducting Super Collider Laboratory is operated by the Universities Research Association, Inc., for the U.S. Department of Energy under Contract No. DE-AC35-89ER40486.

Received November 1992
Revised May 1993

Scientific Programming, Vol. 2, pp. 37–47 (1993)
CCC 1058-9244/94/030037-11

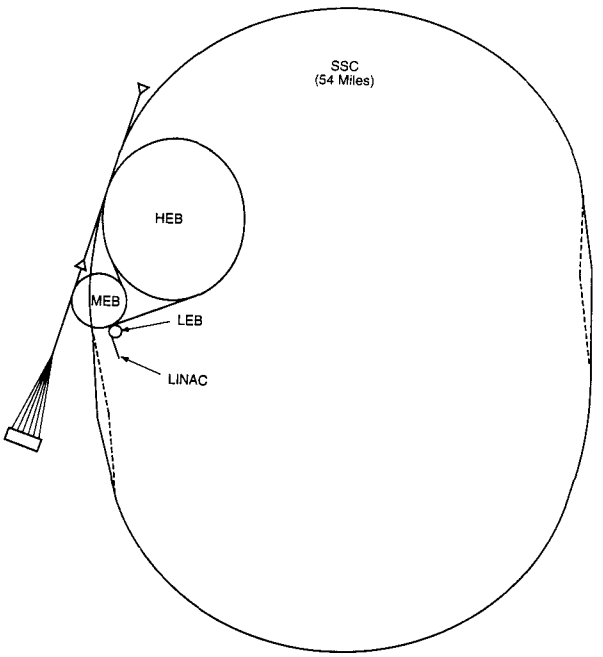

FIGURE 1 Schematic layout of the SSC complex. The 54-mile SSC will be the largest and most powerful particle accelerator ever built.

pensive (expected to exceed $8 billion), there is a demand to reduce the construction cost by design refinement. To reach such a goal, computer simulation of particle motion in each accelerator is essential. For a simulation to be useful, many particles must be tracked in a design lattice* for hundreds of thousands of turns [1]. If hundreds of thousands of particles are tracked in an SSC lattice with 14,000 magnet elements for 10^5 turns in a computer, then more than a thousand Tflops (10^{12} floating-point operations) will be performed. This requirement makes computer simulation difficult without the use of high-performance, scalable, parallel computers. Parallel programming for a large scientific application requires one to understand the characteristics of the problem and to redesign the program to take advantage of hardware features. At the SSC, thin-element accelerator program for optics and tracking (TEAPOT) [2] particle-tracking code has been implemented in a 64-node iPSC/860 machine. In the absence of space charge, it achieves about 98.3% parallel efficiency, because each node can track different groups of particles independently and only very little communication is required between nodes [3].

The perfectly parallel nature of particle tracking is due to the fact that the interaction effects between particles can be ignored when the beam energy is high. In practice, when the beam energy is low, the space charge effects dominate the dynamics as in the LEB and the MEB (at transition) at the SSC. Readers who are interested in the space charge effects are referred to Machida et al. [4]. From the computational point of view, the computational model and characteristics of space charge should be of interest to computational scientists. To model the beam dynamics at low energy correctly, the momentum changes of each particle caused by the space charge force must be calculated. This requires one to solve three-dimensional (3-D) Maxwell field equations by a particle-in-cell (PIC)* fashion, an approach widely used in the simulation of plasma physics. A characteristic of the PIC method is that each particle needs to allocate charge to its nearest neighboring 3-D grids and to retrieve field information from its nearest grids by an interpolating method at each time step. Because particles will change their relative spatial position during tracking, communication between particles and their surrounding grids will be costly for parallel implementation. An efficient and reliable scheme is needed to reduce the communication cost of the space charge calculation and to keep the load balance of particle tracking and space charge computations among nodes as even as possible.

In this paper, we will show a reliable and efficient scheme to implement parallel particle tracking under the influence of space charge on an INTEL iPSC/860 MIMD computer.

The rest of the paper is organized as follows: Section 2 describes the computational model and algorithm of particle tracking under the influence of space charge. Section 3 addresses our parallel implementation techniques. Section 4 presents empirical results on the iPSC/860 and CRAY. Section 5 concludes the paper and provides future research direction.

2 SIMULATION MODEL

Particle simulation based on a single-particle model is perfectly matched for scalable parallel

* A lattice is a detailed description of how an array of elements such as dipole (bending), quadrupole (focus/defocus), sextupole, and rf cavities are arranged to form an accelerator ring at SSC.

* The computation of the electromagnetic field is dependent on the charge density to which all particles must contribute in a nonlocal manner.

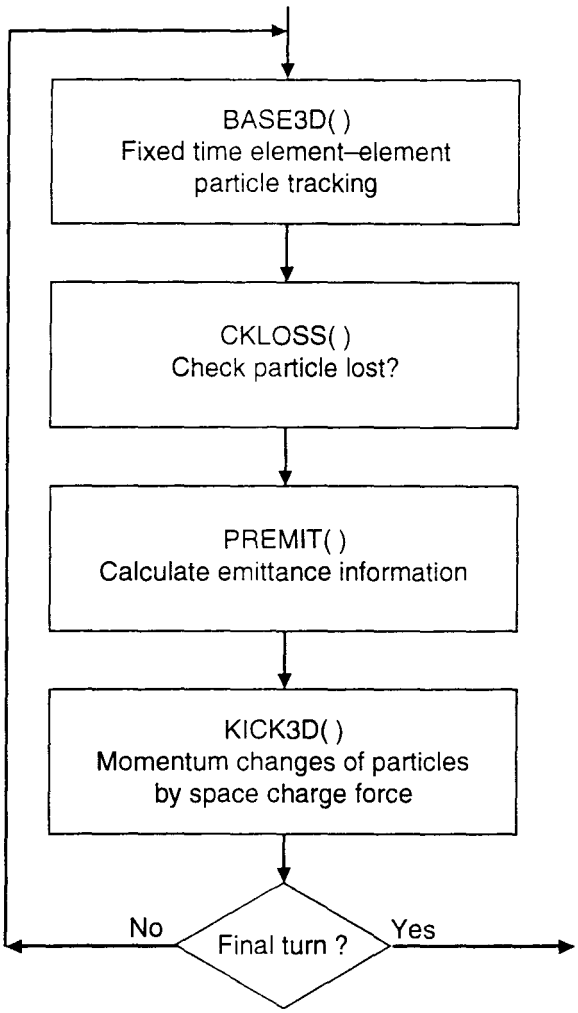

FIGURE 2 Computational model.

architectures. An example is the ZTRACK code by Yan [1]. However, it cannot simulate the operation of the LEB and the MEB because their dynamics are dominated by space charge. To integrate space charge into an existing tracking code such as TEAPOT, it is necessary to calculate charge densities and fields consistent with numerical stability requirements.

For computer simulation, we will use an explicit, fixed-time advance. The motion of particles is tracked for $k(t)$ magnetic elements followed by a kick (space charge force), where $k(t)$ is the number of elements tracked by a particle within the time t and $t + \Delta t$, and may be a fraction less than 1. At each element, the motion of particles is calculated in 6-D phase space. At each kick, we compute a 3-D space charge force on each particle and update its position and momentum. The choice of Δt is determined by numerical stability and strongly affects the speed of the calculation. Figure 2 shows the computational flow of the fixed-time-step sampling method.

In the following, we briefly discuss the models of space charge calculation. Our intention is to provide the minimum accelerator background needed for understanding the parallelization methods used for particle tracking with a space charge code. The tracking algorithm used is based on a 4-D symplectic procedure [2] and later extended to 6-D by Machida et al. [5].

2.1 3-D PIC Formulas for Space Charge

To calculate a 3-D space charge field, we employ the PIC method. At each time step, a 3-D cylindrical grid is first constructed, then the electromagnetic field for particles is computed using the discretized version of 3-D Maxwell equations.

We assume that the 3-D cylindrical grid has coordinates (r, θ, z), and the beam pipe has a circular cross section with radius b, the perfect conductivity $\sigma = \infty$, and the scalar potential $\phi \equiv 0$ at $r = b$. Let n and $\vec{J}$ be the charge density and the current, respectively, and let $\vec{A}$ be the vector potential.

Under the Lorentz gauge $\partial\phi/\partial t + c\vec{\nabla} \cdot \vec{A} = 0$, the Maxwell equations can be rewritten as

$$\left(\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2}{\partial \theta^2}\right)\phi = -4\pi n(r, \theta, z, t), \quad (1)$$

$$\left(\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2}{\partial \theta^2}\right)\vec{A} = -\frac{4\pi}{c}\vec{J}(r, \theta, z, t), \quad (2)$$

where we invoked the ordering

$$\frac{1}{c^2}\frac{\partial^2}{\partial t^2} \sim \frac{\partial^2}{\partial z^2} \ll \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2}{\partial \theta^2}. \quad (3)$$

This can be justified by the typical dimension of the beam, whose transverse size is the order of 1 mm, whereas the longitudinal size is the order of 1 m.

We further simplify the current density,

$$\vec{J} = J_z\vec{e}_z = \bar{v}_z n\vec{e}_z, \quad (4)$$

where $\bar{v}_z$ is the average velocity of the beam. This is commonly referred to as the ultrarelativistic approximation, and it means that particles translate as a rigid body. From the Lorentz force equation $\vec{F} = q(\vec{E} + \vec{v}/c \times \vec{B})$, the electromagnetic force is $\vec{F} = -q/\gamma^2\,\vec{\nabla}\phi$, and we need only solve the scalar potential Eq. (1). The charge density and scalar potential are Fourier transformed in θ:

$$n(r, \theta, z, t) = \sum_m n_m(r, z, t) \exp(im\theta), \quad (5)$$

with the inverse transforms

$$n_m(r, z, t) = \frac{1}{2\pi} \int_0^{2\pi} n(r, \theta, z, t) \exp(-im\theta) d\theta, \quad (6)$$

and the same for ϕ.

For each m (referred to as mode number), Eq. (1) assumes the form

$$\frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial \phi_m}{\partial r} \right) - \frac{m^2}{r^2} \phi_m = -4\pi n_m. \quad (7)$$

The general solution for the equation for $m \geq 0$ is

$$\phi_m = W_m(r) - \left(\frac{r}{b} \right)^m W_m(b), \quad (8)$$

where $W_m(r)$ is

$$W_{m=0}(r) = -4\pi \int_0^r n_0(r, z, t) r' \ln \frac{r}{r'} dr', \quad (9)$$

$$W_{m\geq 1} = -\frac{2\pi r^m}{m} \int_0^r n_m(r, z, t) r'^{(1-m)} dr' - \frac{2\pi r^{-m}}{m} \int_0^r n_m(r, z, t) r'^{(1+m)} dr'. \quad (10)$$

2.2 Algorithm

The space charge algorithm proceeds as follows:

1. Construct the bounding cylinder of particles. The cylinder is then decomposed into 3-D grids [see Fig. 3(a)]. Each grid has index (r, θ, z), which corresponds to cylindrical coordinates $(rdr, \theta d\theta, zdz)$.
2. For each particle, we allocate charge to each grid nearest the particle by the trilinear interpolation method based on the relevant volume ratio [see Fig. 3(b)]. For example, the grid point at index (r_1, θ_1, z_1) has volume ratio $(r_2 dr - r)(\theta_2 d\theta - \theta)(z_2 dz - z)/(drd\theta dz)$, where (r, θ, z) are the cylindrical coordinates of the particle and (r_2, θ_2, z_2) is the index of the opposite grid point of (r_1, θ_1, z_1).
3. Compute the Fourier decomposition of charge density in θ using Eq. (6).

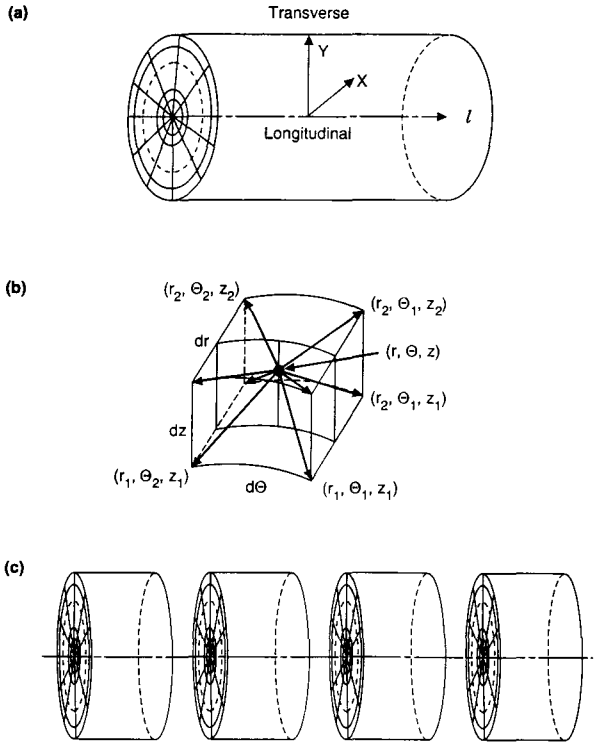

FIGURE 3 3-D grid decomposition. (a) 3-D grids of bounding cylinder of particles; (b) communication between a particle and its neighbor grids; (c) grid decomposition in longitudinal direction.

4. Compute the electrical field (E_r, E_θ, E_z) for each grid. This can be done by computing ϕ using Eqs. (7–10).
5. Compute the momentum changes for each particle from its surrounding fields and update its coordinates and other attributes.

3 PARALLEL IMPLEMENTATION

The key to a parallel implementation of a computational model into a MIMD hypercube parallel computer is to distribute the computation and data into the separate nodes such that each node has an equal share of computations, while communication between nodes is minimized. Although the principle is simple, the practice is more complicated, and a given implementation must take advantage of the available hardware features and take care of subtle issues with each parallel scheme—I/O and memory problems, numerical stability, and hardware failures—to achieve high performance for a large scientific application.

To take advantage of scalable parallel computers, it is necessary to understand the characteristics of the problem so that the program can be implemented correctly and efficiently (with performance scalable to the number of computing nodes available). The characteristics of particle tracking with space charge effects are summarized below.

3.1 Characteristics

- Particles are tracked in 6-D phase space, and they can be tracked independently in BASE3D( ).
- Particles are lost when they collide with the wall of the machine.
- The 3-D PIC method requires a large number of field quantities defined on a 3-D mesh. The memory requirements of a complete problem exceed the limit of 8 Mbytes of memory of a single node, so domain decomposition is required.
- Communication between a particle and its eight nearest grid points is required to allocate charge and to interpolate the fields.
- A large data set must be produced for the visualization of tracking results and for restart capability (primarily to permit recovery from hardware failure).

3.2 Implementation Schemes and Techniques

For reasons that will be explained in the next section, the main task of parallel implementation is to ensure that each computing node has the same number of particles in order to achieve load balancing. This is very important, because tracking the noninteracting particles in the magnetic lattice occupies most of the computational time. The second primary task is to decompose the 3-D grids in KICK3D( ). The constraints are (1) the space charge code should fit into an 8-Mbyte node; (2) each node should have the same amount of workload in computing the loops over particles (for allocating charge to grids and retrieving field information from grids) as well as the loops over 3-D grids (for computing electromagnetic force); and (3) communication among nodes should be minimized.

Below we consider partition schemes for particles in the tracking phase and 3-D grids in space charge.

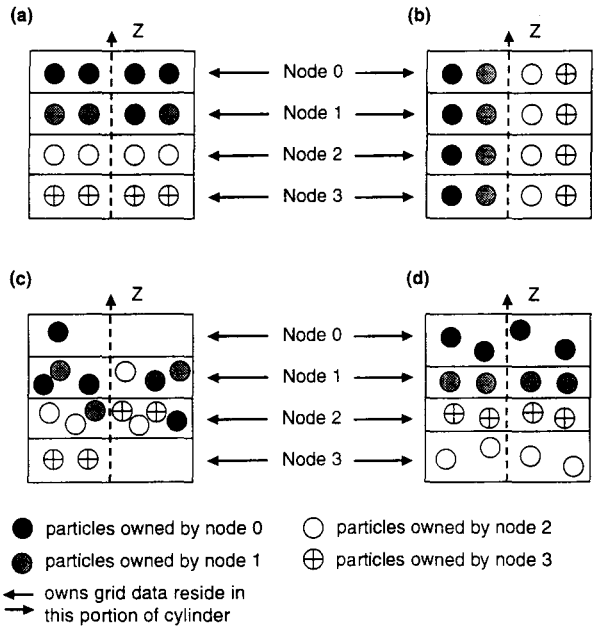

FIGURE 4 Possible spatial position of particles and subcylinder with equal-grid and equal-particle approach. Each rectangle is the side view of the cylinder. (a) equal size subcylinder, uniform particle distribution; (b) equal size subcylinder, uniform particle distribution, cyclic particle mapping; (c) equal size subcylinder, nonuniform particle distribution; (d) nonequal size subcylinder, nonuniform particle distribution.

3.2.1 Partition Scheme for Particle Tracking

So that each computing node has the same workload, particles are assigned equally into computing nodes by the block or cyclic method.* Because particles are frequently lost during tracking when they run into a wall, a load imbalance situation will develop. That is, some nodes might have many more particles to track than the others. The cyclic method is usually a better approach to deal with a load-imbalance situation. However, such an approach is not adequate when space charge is introduced. Figure 4(b) demonstrates a case using a cyclic approach, which will produce busy communications among all nodes because it violates the data locality principle. In practice, we found that a block decomposition is a proper way to deal

* Assume that the index set of all particles is [0, 1, . . . , 15] and the index set of all nodes is [0, 1, 2, 3]. In the block method, the particle index set [4, 5, 6, 7] is assigned to node 1, whereas in the cyclic method, the particle index set [1, 5, 9, 13] is assigned to node 1.

with our problem as long as particles are not lost dramatically.

3.2.2 3-D Grid Decomposition Schemes for Space Charge

There are several ways to map the 3-D grids into computing nodes. It depends on how the hypercube is connected as a ring, a 2-D mesh, or a 3-D hypercube. For programming simplicity, we will use block mapping from 3-D grids into a 1-D array of computing nodes. The cylinder is partitioned in the longitudinal direction [see Fig. 3(c)]. The partition method can be based on the equal number of grids (equal-grid) strategy or equal number of particles (equal-particle) in a subcylinder strategy.

In the equal-grid approach, each node gets a nearly equal size of subcylinder (or the same number of mesh points). If particle distribution is uniform in the longitudinal direction, then each subcylinder will contain the same amount of particles. Therefore, there is little or no communication between nodes. Figure 4(a) shows the best-case situation, in which all the particles and their grid neighbors belong to the same node; therefore, little or no communication is needed. In practice, the distribution of particles tends to be nonuniform during simulation. Figure 4(c) shows a nonuniform particle distribution case in which not only is communication necessary, but some nodes have to update grid information for the other nodes as well. As a result, load balancing among nodes is uneven.

In the equal-particle approach, the cylinder is partitioned into subcylinders, each of which contains a nearly equal number of particles. When the particle distribution is nonuniform, each node will have an unequal subcylinder [see Fig. 4(d)]. This strategy gains a performance advantage by keeping communication minimal at the expense of uneven grids in each node's domain. To achieve high parallel efficiency, an effective mechanism is necessary to maintain the "equal-particle" structures and to minimize the load-imbalance effect of uneven grids.

Both approaches have their advantages and drawbacks (see Table 1 for a comparison). In general, there is a trade-off between speed and memory. Because a memory upgrade for iPSC/860 is relatively expensive, it would be very desirable to combine both approaches to compromise parallel efficiency and memory to achieve high parallel efficiency within the limit of node memory.

3.2.3 First Try

Because we want to keep the code size as small as possible to fit into 8 Mbytes of available memory, we have chosen the simplest strategy to implement particle tracking with space charge code. That is, particles are partitioned using the block method, and 3-D grids are decomposed using block mapping with the equal-grid approach in the z direction. This approach can be implemented more quickly than alternative methods. We made no assumption about spatial relationships between particles and their surrounding grids. Particles can move anywhere (e.g., across several domains [subgrids]) between calculations. This approach is very general and could be implemented with moderate effort should a parallel compiler, which can effectively solve irregular communication within a parallel loop, become available in the future.

3.2.4 Communication Patterns and Programming Techniques

In the following, we discuss briefly the techniques used to solve irregular communications in space charge. We consider the case where a particle needs to allocate charge to its eight nearest neighbor grid points (referred to as allocating process). The inverse process of interpolating field information to the particle location (referred to as interpolating process) can be treated similarly. For clarity, only one grid point with index (0, 0, 0) is shown in the following sequential code. Note that

Table 1. Comparison of Equal-Grid and Equal-Particle Partition

	Equal-Grid	Equal-Particle
Load balance (loops over particles)	No (nonuniform)	Yes
Load balance (Loops over Grids)	Yes	No (nonuniform)
Communication overhead	Large (nonuniform)	Small
Memory (grid) size scalability	Yes	No (nonuniform)
Programming effort	Easy	Difficult

Note: Nonuniform refers to the distribution of particles.

```
vol = dr*dz*dth
do ipart=mpar, npar
     ir0 = ri(ipart)/dr
     ith0 = thi(ipart)/dth
     iz0 = zi(ipart)/dz
     ratio(ipart,0,0,0)=(((ir0+1)*dr-ri(ipart))
         +((ith0+1)*dth-thi(ipart))+((iz0+1)*dz-zi(ipart)))/vol
     density(ir0,ith0,iz0) = density(ir0,ith0,iz0)+ratio(ipart,0,0,0)
enddo
```

particles and 3-D grids are partitioned based on the strategies described above. A particle and its nearest neighbors might not belong to the same node. That is, `zi(ipart)` and `density(ipart,*,*)` do not necessarily belong to the same node.

The current parallel programming tool available to us is the Mimdizer (Pacific Sierra, Inc.), which has an automatic decomposition tool at the loop level. However, the performance we obtained with this tool has not been acceptable. Another tool reported by Hiranandani et al. [6] as being able to transform this kind of code into explicit message-passing routines without much programming effort is Fortran D by Rice University. However, Fortran D is still under development and was not available to us when we developed the code. The communication strategy, therefore, had to be developed by hand.

The strategy that we use is similar to the approach proposed by Saltz et al. [7]. The idea is based on block I/O transfer to minimize the communication between nodes. That is, all information that a node needs to communicate with other nodes is accumulated into a buffer. A global communication table that describes how a pair of nodes should communicate with one another is computed first. Each node then sends out self-descriptive information to the other nodes. The information received by a node includes the position and fractional density for each grid that should be updated by this node. An advantage of this approach is that the global communication table needs to be computed only once in KICK3D( ) at each time step. Therefore, it results in a reduction of communication time that would be very difficult to achieve even using future automatic parallel compilers, because such an automatic parallelizer will not be able to plan ahead and collect operations as a human programmer can. This strategy provides a reliable and effective communication mechanism for Fortran implementation.

3.2.5 Performance Tune-Up

A particle can move from one grid to another grid between space charge calculations and is in fact unlikely to keep the perfect spatial position seen in Figure 4(a) and (d) all the time. It is probable that a situation like that in Figure 4(c) will happen during a long run. One way to keep the particles and their associated grid points in the same memory is to sort the particles in the z direction and to remap into computing nodes. The best sorting algorithm requires order of $(n \log n)/p$ operations [$O(n)/p$ if bucket sort is used], where p is the number of nodes. When n is large, the overhead will be exceptionally high. Because particles change their relative position and surrounding grids in the longitudinal direction gradually, it is necessary to sort these particles only occasionally (about every 50 turns).

Another approach is to have subcylinder guards for each node. Here, sorting is reduced to subcylinder guard communication between two neighbor nodes. This approach usually assumes that particles can move only from a subcylinder to the next neighbor subcylinder between space charge calculations. Such a constraint is imposed by numerical stability considerations for any explicit time advance algorithm. Therefore, communication is performed only between neighboring nodes. A combination of the above tune-up strategies with our current scheme makes it possible to provide better performance than either of the above approaches with only a little extra memory expense.

3.3 Code Development

The particle tracking with space charge code was first written for the CRAY-YMP by Machida et al. [5]. The CRAY code that can handle 10,000 particles in an LEB lattice utilized about 19 Mbytes of memory, which includes 11 Mbytes space charge (KICK3D) code. This code was analyzed with

FORGE (Pacific Sierra, Inc. [8]) to identify the most computationally intensive portion. The most time-consuming routine is the particle-tracking code (BASE3D) subroutine. The code is very complex and not well suited for pipelining. It uses about 61% total time when the time step $\Delta t = 10$ ns. The next most time-consuming routine is the KICK3D code, which takes about 25%. The code is tuned using the optimization option of Fortran compiler only, because the code is very complex and not well suited to pipelining. Parallel implementation was based on the strategies addressed in Section 3.2.3.

3.4 Hardware Platform

The SSC iPSC/860 has 64 computing nodes, 62 of which have 8 Mbytes of memory and 2 of which have 32 Mbytes. The MIMD architecture allows one to run different programs on different nodes simultaneously, although the programming paradigm at this machine tends to favor the single program multiple data (SPMD) style. For particle tracking with space charge we combine both paradigms, wherein the master node (node 0) runs a different program from the other nodes (workers). Because the worker nodes are utilized for tracking and kicking, they can execute in 8-Mbyte memory nodes. The master node, which has 32 Mbytes, is also utilized to deal with I/O and to input data.

4 EMPIRICAL RESULTS AND DISCUSSION

We have tested the program on our iPSC/860 machine, running 10,000 particles for 500 turns in an LEB lattice that contains 693 elements. For our applications, a cylindrical grid size of 40 × 20 × 32 is appropriate. This suggests the maximum number of nodes used in this study is 32. Additional nodes will provide redundant computation using our domain decomposition strategy. As mentioned previously, the fields are Fourier decomposed in the azimuthal direction, with a maximum mode number of 2 utilized in this case. Using 32 nodes, the simulation took about 26.3 hours to finish. The results of the parallel implementation were checked against the CRAY version by starting a run on the CRAY and tracking it for six turns. The two codes were exercised in tandem from this point and tracked for 500 turns. Differences in the random number generator required this type of start-up procedure. Figure 5(a)–(c)

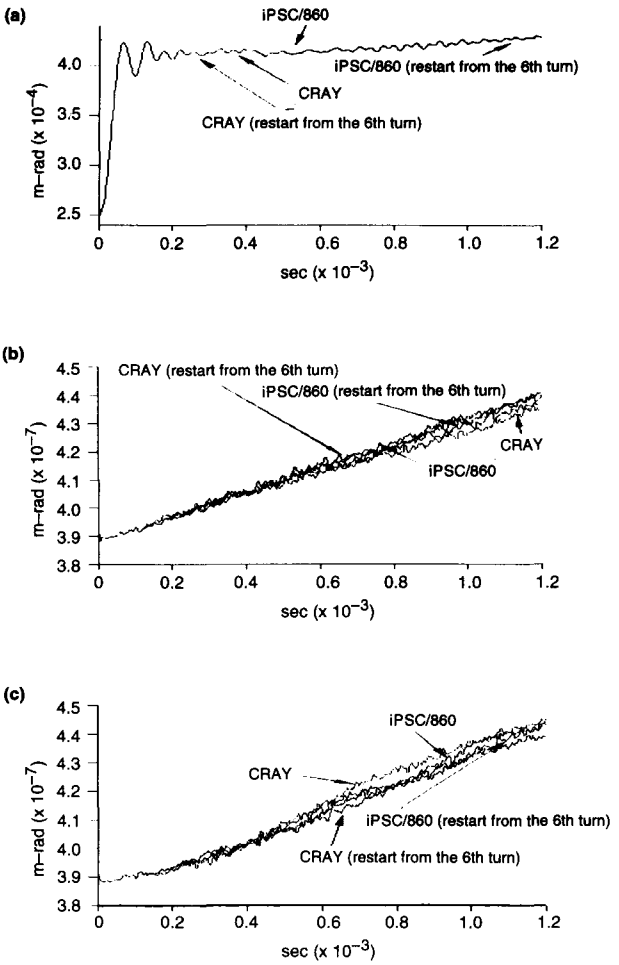

FIGURE 5 Emittance growth (a) in the S direction; (b) in the X direction; (c) in the Y direction.

shows the emittance growth in the longitudinal S, X, and Y directions, respectively. Readers can see that the emittance evolution in the S direction is identical, but it differs slightly in both the X and Y direction after about 0.2 msec (100 turns). However, they almost converge at 1.2 msec (500 turns) in both the X and Y direction. The differences in numerical results between the two supercomputers are small and are probably due to differences in word length.

Although our goal in using parallel computers is to reduce the computation time in tracking study, readers are often interested in the scalability issues such as whether the performance of implementation is scalable to the number of processors. From what we learned in using massively parallel computers, such issues can be observed in the following ways. First, in the absence of space charge force, our problems have a natural granularity

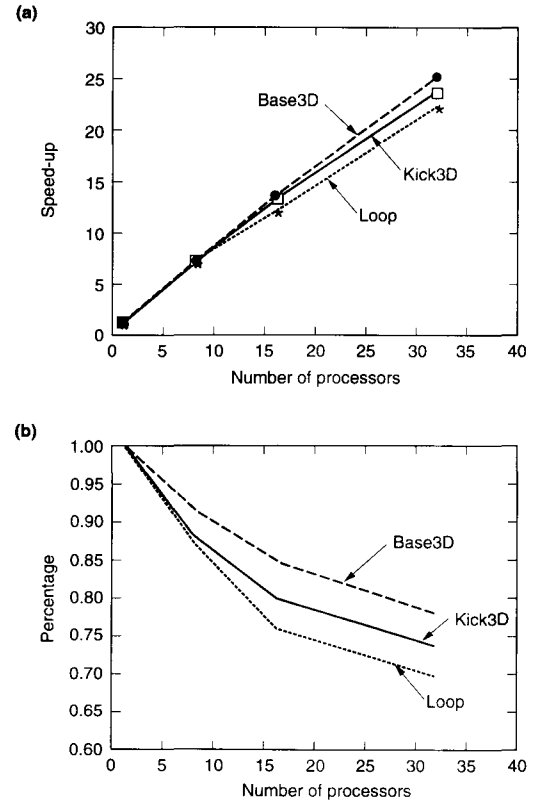

FIGURE 6 (a) Speed-up performance and (b) parallel efficiency.

that makes it "embarrassingly parallel," one simply distributes the particles over available nodes and track. The number of nodes should not exceed the number of particles tracked and the overall performance of the calculation is dominated by single node performance. In this context, obviously, the scalability is limited by the number of particles studied. Second, in the space charge case, there is also a natural granularity of the grid size (32 in our test case) that limits its scalability to the number of processors as explained previously. It is also obvious that the space charge is communication code intensive, which will be the bottleneck eventually as the number of processors increases. To show the performance of our parallel implementation, a speed-up performance graph and a parallel efficiency figure are shown in Figure 6a and 6b, respectively. The speed-up comparison is based on the performance of the original code (nonparallelized version) running on a big (32 Mbytes) node versus 8-node, 16-node, and 32-node parallel version. Both figures represent the performance of the overall loop, the performance of the BASE3D routine for particle tracking, and the performance of the KICK3D routine for space charge at a time step (Fig. 2). The parallel efficiency of the tracking code is 80–92%, whereas the parallel efficiency of the space charge code is about 75–88%. The overall loop performance is slightly lower than the performance of tracking and of space charge code because we need to check the particle loss situation and collect emittance information at each time step. The above facts indicate that our parallel algorithm does not provide the optimal solution, but it does a fairly reasonable job. Using 32 nodes, we are able to obtain a speed-up in overall performance by a factor of 22. A more significant fact is that the 32-node performance makes space charge simulation feasible, which otherwise would be impossible using SUN-SPARC II, and eliminates a month of computations.

The use of advanced visualization techniques as an aid to understanding coherent wave motions in plasma simulation is well accepted. Part of the parallel space charge simulation effort is to develop high bandwidth visualization techniques capable of displaying simulation results from the hypercube. To this end, we have integrated a Silicon Graphics Crimson/VGX to the hypercube and have begun the software development task.

Figure 7 shows the motion of 6000 particles in the first six consecutive time steps. The transverse dimension is scaled by a factor of 100 relative to the longitudinal direction. Eight different particles are shown in the figure; particles with the same number are assigned into the same nodes. An optimal algorithm must maintain the same identification number of particles in the same contiguous slices of cylinders to minimize communication among nodes and to maintain workload balance.

5 CONCLUSION AND FUTURE RESEARCH DIRECTION

We have successfully implemented particle tracking with space charge effects using the 3-D PIC method with an explicit time advance on our iPSC/860 parallel computer. The numerical results are compared with CRAY. We show that the new version of iPSC/860 code does the right physics and is very effective and scalable for our applications. The current implementation is very effective and can be implemented quickly to suit our operational needs.

For future research directions, we are investigating the use of 3-D visualization techniques in

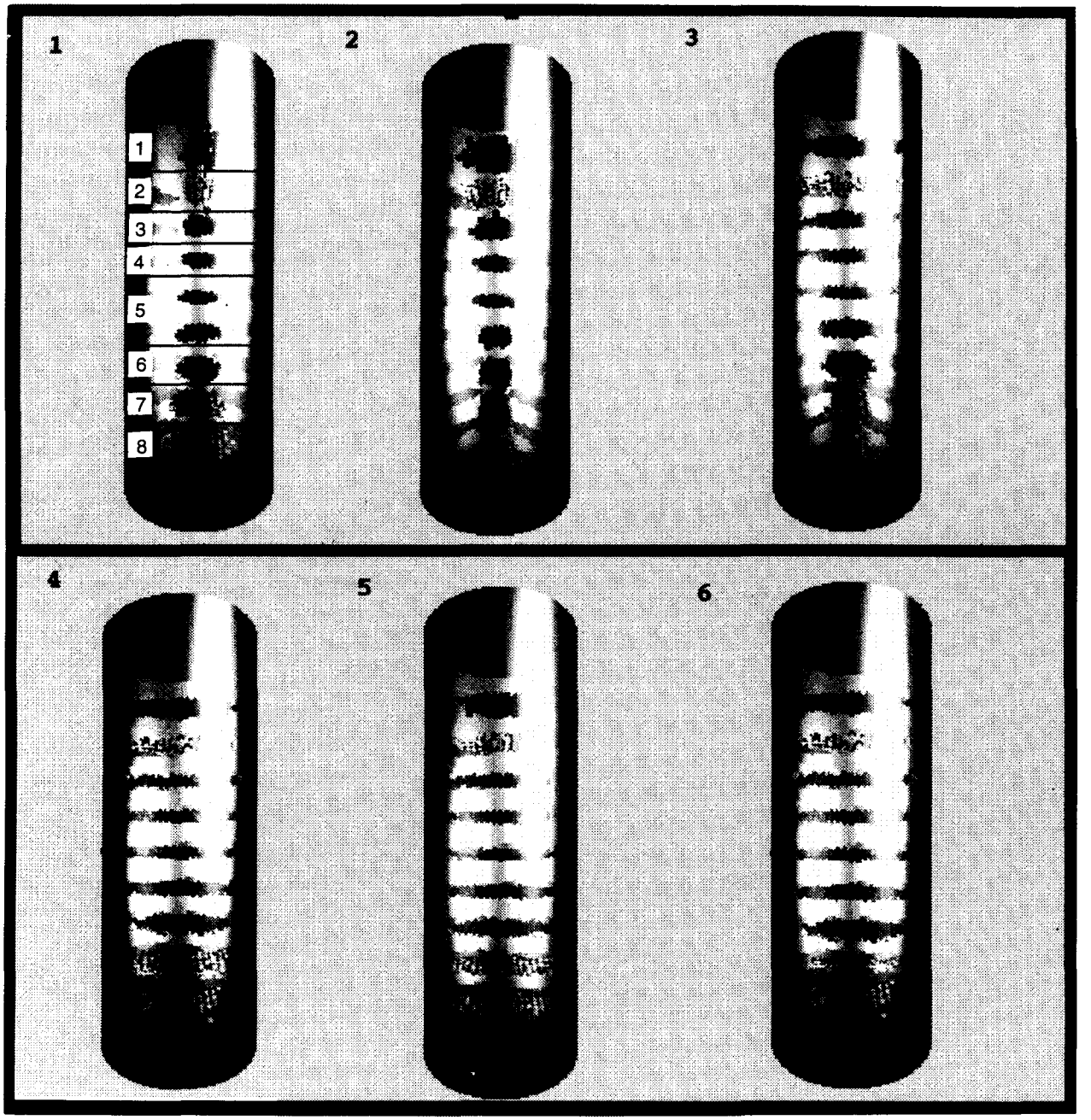

FIGURE 7 The motion of 6000 particles at the first six time frames.

order to visualize collective phenomena. This requires normalization of the motion with respect to the local β function ratio.

REFERENCES

[1] Y. Yan, Supercomputing for the Superconducting Super Collider. Energy Science, *Supercomputing* 1990, National Energy Research Supercomputer Center, 9-13.

[2] L. Schachinger, and R. Talman, "TEAPOT: A thin-element accelerator program for optics and tracking," *Particle Accelerators*, vol. 22, pp. 35–56, 1987.

[3] G. Bourianoff, and R. Talman, "Acclerator progress relies on computation simulations," *Comput. Phys.* vol. 6, pp. 14–23, 1992.

[4] S. Machida, G. Bourianoff, N. K. Mahale, N. Mehta, F. Pilat, R. Talman, and R. C. York, "Space charge effects in the SSC low energy booster," *IEEE Particle Accelerator Phys. Conf.*, vol. 1, pp. 383–385, 1991.

[5] S. Machida, G. Bourianoff, Y. Huang, and N. K. Mahale, "Tracking study of hadron collider boosters." HEACC '92, pp. 1058–1060. *Int. J. Mod. Phys.* A (proc. suppl.) 2B, 1993, World Scientific Publishing Co.

[6] S. Hiranandani, K. Kennedy, and C. W. Tseng, "Compiler optimizations for Fortran D on MIMD distributed-memory machines," *Supercomputing 1991*, Albuquerque, pp. 86–100.

[7] J. Saltz, K. Crowley, R. Mirchandaney, and H. Berryman, "Run-time scheduling and execution of loops on message passing machines," *J. Parallel Distrib. Comput.* vol. 8, pp. 303–312, 1990.

[8] *FORGE/MIMDIZER User's Guide*, Pacific Sierra, Inc.