AN OBJECT-ORIENTED PARALLEL PARTICLE-IN-CELL CODE FOR BEAM DYNAMICS SIMULATION IN LINEAR ACCELERATORS*

J. Qiang†, R. D. Ryne, S. Habib, LANL, Los Alamos, NM
V. Decyk, UCLA, Los Angeles, CA

Abstract

In this paper, we present an object-oriented three-dimensional parallel particle-in-cell code for beam dynamics simulation in linear accelerators. A two-dimensional parallel domain decomposition approach is employed within a message passing programming paradigm along with a dynamic load balancing. Implementing object-oriented software design provides the code with better maintainability, reusability, and extensibility compared with conventional structure based code. This also helps to encapsulate the details of communication syntax. Performance tests on SGI/Cray T3E-900 and SGI Origin 2000 machines show good scalability of the object-oriented code. Some important features of this code also include employing symplectic integration with linear maps of external focusing elements and using z as the independent variable, typical in accelerators. A successful application was done to simulate beam transport through three superconducting sections in the APT linac design.

1 INTRODUCTION

With increasing interest in high intensity beams, it is getting more important to accurately treat the space charge effects in linear accelerator. The particle-in-cell simulation method has been adopted to study high intensity beams [1, 2, 3, 4]. At present, most of these codes run only on serial computers. This restricts the number of numerical particles which can be used in the simulation due to the heavy computation time cost of particle-in-cell method. This restriction also degrades the accuracy of particle-in-cell simulation. Parallelism can significantly improve the statistical accuracy of particle-in-cell simulation by using a larger number of numerical particles and finer grid resolution. It also dramatically reduces the computation time. In this paper, we present an object-oriented parallel particle-in-cell code for beam dynamics simulation in linear accelerators. This gives the program better maintainability, reusability, extensibility, and high performance.

In this paper, the physical model is described in Section 2. The parallel numerical algorithm is discussed in Section 3. The object-oriented software design is given in Section 4. Performance tests are given in Section 5. An application is presented in Section 6. The conclusions are drawn in Section 7.

* Work supported in part by DOE Grand Challenge in Computational Accelerator Physics.

† Email: jiqiang@lanl.gov

2 PHYSICAL MODEL

The physical system for beam dynamics studies consists of the beam and the accelerating/transport system which in turn contains a number of accelerating and focusing elements. The forces acting on particles are due to externally applied fields and the inter-particle Coulomb field. The dynamics of particles is governed by the Poisson-Vlasov system of equations. In accelerator simulations, it is usual practice to take z to be the independent variable rather than the time t. The Vlasov equation is written as:

$$\frac{\partial f}{\partial z} + x'\frac{\partial f}{\partial x} + y'\frac{\partial f}{\partial y} + t'\frac{\partial f}{\partial t} + p'_x\frac{\partial f}{\partial p_x} + p'_y\frac{\partial f}{\partial p_y} + p'_t\frac{\partial f}{\partial p_t} = 0 \quad (1)$$

and the Poisson equation is

$$\nabla^2\phi = -\rho/\epsilon \quad (2)$$

where f is the particle distribution function in (x, p_x, y, p_y, t, p_t) phase space, a prime denotes $\frac{\partial}{\partial z}$, x, y, t are x, y position and time interval respectively, p_x and p_y are x and y dimension canonical momentum, $p_t = -H(x, y, z, p_x, p_y, p_z, t)$, where H is the Hamiltonian of system, ϕ is the space charge potential from Coulomb interaction, ρ is the charge density from the distribution function, and ϵ is the dielectric constant in vacuum. We define a new quantity K according to

$$K(x, p_x, y, p_y, t, p_t; z) = -p_z \quad (3)$$

The Vlasov equation can be rewritten as

$$\frac{\partial f}{\partial z} + [K, f] = 0 \quad (4)$$

where $[,]$ is the Poisson brackets. Given any two functions f and g, the Poisson bracket of f and g is given by

$$[f, g] = \sum_{i=1}^{3}\left(\frac{\partial f}{\partial q_i}\frac{\partial g}{\partial p_i} - \frac{\partial f}{\partial p_i}\frac{\partial g}{\partial q_i}\right) \quad (5)$$

where q specifies x, y, t, and p specifies p_x, p_y, p_t. The equations of motion for the particles using z as the independent variable are [5]

$$x' = \frac{\partial K}{\partial p_x}, \qquad p'_x = -\frac{\partial K}{\partial x} \quad (6)$$

$$y' = \frac{\partial K}{\partial p_y}, \qquad p'_y = -\frac{\partial K}{\partial y} \quad (7)$$

$$t' = \frac{\partial K}{\partial p_t}, \qquad p'_t = -\frac{\partial K}{\partial t} \quad (8)$$

SC '99, Portland, OR
(c) 1999 ACM 1-58113-091-8/99/0011 $3.50

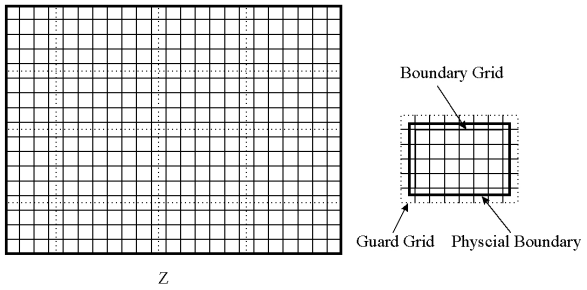

Figure 1: A schematic plot of two-dimensional decomposition on y-z domain.

Numerical solution of the particle equations of motion is carried out using a split-operator symplectic integration method and Lie algebraic maps.

We calculate the charge density by depositing the particles onto grids using a cloud-in-cell (CIC) scheme. The potential is expressed as

$$\phi_{p,q,r} = \sum G_{p-p',q-q',r-r'} \rho_{p-p',q-q',r-r'} \qquad (9)$$

where G is the Green's function on the grid, and ρ is the charge density on the grid. Often, the beam size is much smaller than the inside wall radius of the accelerator, in which case we may treat the beam as an isolated system. In such a case, the above convolution can be calculated using a Fast Fourier Transform (FFT) technique given by Hockney [6].

3 PARALLEL PARTICLE-IN-CELL ALGORITHM

A two-dimensional domain-decomposition approach is employed in the parallel particle simulation [7]. A schematic plot of the two-dimensional decomposition on the y-z plane is shown in Fig. 1. The solid grid lines define the computation domain grids. The dashed lines define the local computational domain on each processor. Here, the boundary grids are the outer most grids inside the physical boundary. The guard grids are used as temporary storage of grid quantities from the neighboring processors. The physical computational domain is defined as a 3-dimensional rectangular box with range $x_{min} \le x \le x_{max}$, $y_{min} \le y \le y_{max}$, and $z_{min} \le z \le z_{max}$. This domain is decomposed on the $y - z$ plane into a number of small rectangular blocks. These blocks are mapped to a logical two-dimensional Cartesian processor grid. Each processor contains one rectangular block domain. The range of a block on a single processor is defined as $x_{min} \le x \le x_{max}$, $y_{lcmin} \le y \le y_{lcmax}$, and $z_{lcmin} \le z \le z_{lcmax}$. Here, the subscript $lcmin$ and $lcmax$ specify local minimum and local maximum. The mesh grid is defined to store the field related quantities such as charge density and electric field. The number of grid points along three dimensions on a single processor is defined as:

$$\begin{aligned} Nx_{local} &= int[(x_{max} - x_{min})/hx] + 1 \qquad (10) \\ Ny_{local} &= int[(y_{lcmax} - y_{min})/hy] - \\ &\quad int[(y_{lcmin} - y_{min})/hy] + 2, 1 \qquad (11) \\ Nz_{local} &= int[(z_{lcmax} - z_{min})/hz] - \\ &\quad int[(z_{lcmin} - z_{min})/hz] + 2, 1 \qquad (12) \end{aligned}$$

where hx, hy, and hz are the mesh sizes along x, y and z direction respectively. Here, the number 2 in Ny_{local} and Nz_{local} are two guard grids on both sides of y and z dimensions outside the boundary grids if the number of processors in that dimension is greater than 1. If the number of processors in that dimension is 1 for one-dimensional processor partition case, this number is set to 1. For the processor containing the starting grid in the global mesh, there is one more grid point along the y and z directions. The particles with spatial positions within the local computational boundary are assigned to the processor containing that part of physical domain.

The particles generated on each processor will advance following the maps defined in Section 2. If a particle moves outside the local computational domain, it will be sent to the corresponding processor where it is located. A particle manager function is defined to handle the explicit communication using MPI among two-dimensional processor grids. The y and z positions of every particle on each processor are checked. The particle is copied to one of its four buffers and sent to one of its four neighboring processors when its y or z position is outside the local computational domain. After a processor receives the particles from its neighboring processors, it will decide among those particles whether some of them will be further sent out or not. The outgoing particles are counted and copied into four temporary arrays. The remaining particles are copied into another temporary array. This process is repeated until there is no outgoing particle on all processors to be found. Then, the particles in the temporary storage along with the particles left in the original particle array are copied into a new particle array.

After each particle moves to its local computational domain, a linear CIC particle-deposition scheme is done for all processors to get the charge density on the grid. For the particles located between the boundary grid and computational domain boundary, these particles will also contribute to the charge density on the boundary grids of neighboring processors. Hence, explicit communication is required to send the charge density on the guard grids, which is from the local particle deposition, to the boundary grids of neighboring processors to sum up the total charge density on the boundary grids. With the charge density on the grids, Hockney's FFT algorithm is used to solve the Poisson's equation with open boundary conditions. Due to this algorithm, the original grid number is doubled in each dimension. The charge density on the original grids is kept the same. The charge density on the other grids is set to 0.

The Green's function is defined as

$$G_{p,q,r} = \frac{1}{\sqrt{(hx(p-1))^2 + (hy(q-1))^2 + (hz(r-1))^2}} \tag{13}$$

with

$$p = 1,\ Nx^*_{local}{+}1;\ q = 1,\ Ny^*_{local}{+}1;\ r = 1,\ Nz{*}_{local}{+}1 \tag{14}$$

Here, Nx^*_{local}, Ny^*_{local}, Nz^*_{local} are the local computation grid number without including guard grids in all three dimensions. For the grid points outside the above boundary, a symmetry is used to define the Green's function on these grids.

$$G_{p,q,r} = G_{2Nx-p+2,q,r},\quad p = Nx^*_{local} + 2,\ 2Nx \tag{15}$$

$$G_{p,q,r} = G_{p,2Ny-q+2,r},\quad q = Ny^*_{local} + 2,\ 2Ny \tag{16}$$

$$G_{p,q,r} = G_{p,q,2Nz-r+2},\quad r = Nz^*_{local} + 2,\ 2Nz \tag{17}$$

Communication is required to double the original distributed 3-dimensional grid explicitly. This can be avoided by including this process into the 3-dimensional FFT. In the 3-dimensional parallel FFT, we have taken advantage of the undistributed dimension along the x dimension, where a local serial FFT can be done in that dimension for all processors. A local temporary two-dimensional array with size $(2Nx, Ny_{local})$ is defined to contain part of the charge density at fixed z. The charge density on the original grid is copied into (Nx, Ny_{local}) part of the temporary array. The rest of the temporary array is filled with 0. In the case of the FFT of the Green's function, symmetry can be used to obtained the values of Green's function in the region $(Nx + 2, Ny_{local})$. After the local two-dimensional FFT along x is done, it is copied back to a slice of a new 3-dimensional array with size $(2Nx, Ny_{local}, Nz_{local})$. A loop through Nz_{local} gives the FFT along x for the three dimensional array. Then, a transpose is used to switch the x and y indices. Now, the 3-dimensional matrix has size $(Ny, Nx'_{local}, Nz_{local})$. Here, Nx'_{local} is the new local number of grids in x dimension along y dimension processor. A similar process is done to obtain the FFT along the y direction for double-size grids $(2Ny, Nx'_{local}, Nz_{local})$. Another transpose is used to switch y and z indices and a local FFT along z with a double-size grid is done on all processors to finish the 3-dimensional FFT for double-size grid in all three dimensions. During the inverse parallel FFT, a reverse process is employed to obtain the potential on the original grids. In the transpose of indices, global all-to-all communication is used.

Dynamic load balance is employed with adjustable frequency to keep the number of particles on each processor about equal. A density function is defined to find the local computation domain boundary so that the number of particles on each processor is roughly balanced. This number depends on the local integration of charge density distribution function on each processor. To determine the local boundary, first, the three-dimensional charge density is summed up along x direction on each processor to reduce to a two-dimensional density function. This function is distributed locally among all processors. Then, the two-dimensional density function is summed up along the y direction to get the local one-dimensional charge density function along z. This density function is broadcast to the processors along y direction. The local charge density function is gathered along z and broadcast to processors along the z direction to get a global z direction charge density distribution function on each processor. Using this global z direction density distribution, the local computational boundary in the z dimension can be determined assuming that each processor contains about about $1/nproc_z$ fraction of total number of particles. Here, $nproc_z$ is the number of processors along the z direction in the two-dimensional Cartesian processor grid. A similar process is used to determine the local computation boundary in the y direction. Strictly speaking, the above algorithm will work correctly for a two-dimensional density distribution function which can be separated as a product of two one-dimensional function along each direction. However, from our experience, this algorithm works reasonablely well in beam dynamics simulation in the linear accelerator.

4 OBJECT-ORIENTED SOFTWARE DESIGN

The above parallel particle-in-cell algorithm is implemented in an object-oriented framework for the accelerator simulation. Object-oriented software design is a method of design encompassing the process of object-oriented decomposition [8]. After analysis of the (complex) physical system, the system is first decomposed into simpler physical modules. Next, classes and objects are identified inside each module. The class is an abstract representation of objects. Each class has interfaces to communicate with the outside environment. Relationships are then built up among different classes and objects. These classes and objects are implemented in a concrete language representation. The implemented classes and objects are tested separately and then put into the physical module. Each module is tested separately before it is assembled into the whole program. Finally, the whole program is tested to meet the requirements of problem.

An application of the object-oriented design methodology outlined above to beam dynamics studies in accelerators results in the decomposition of the physical system into five modules. The first module handles the particle information consisting of the *Beam*, *BeamBC*, and the *Distribution* classes. The second module handles information regarding quantities defined on the field grid containing *Field* and *FieldBC* classes. The third module handles the external focusing and accelerating elements containing the *BeamLineElem* base class and its derived classes, the drift tube class, the quadrupole classes, and the rf gap class. The fourth module handles the computational domain geometry containing the *Geometry* class. The last module provides

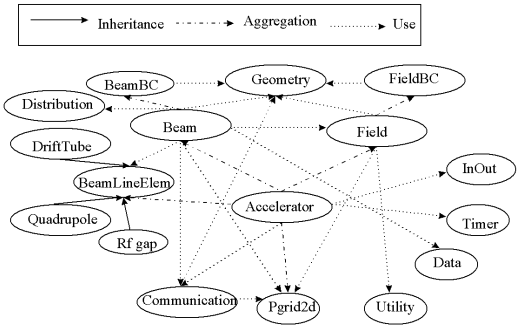

Figure 2: Class diagram of accelerator beam dynamics system.

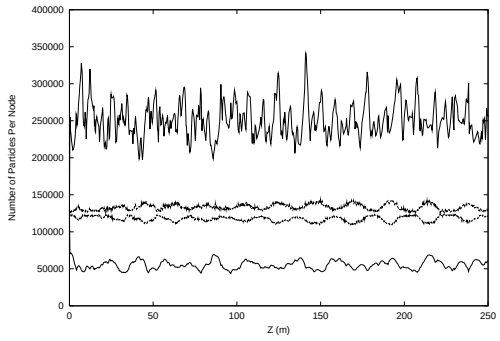

Figure 3: The maximum number of particles and minimum number of particles on one processor without and with dynamic load balance.

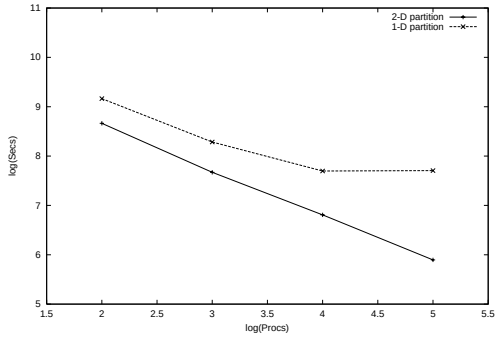

Figure 4: The time cost as function of number of processors on T3E using one-dimensional and two-dimensional parallel partition.

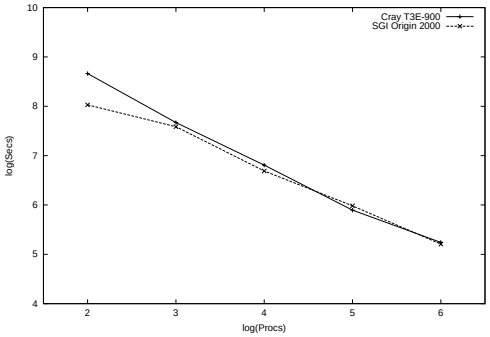

Figure 5: The time cost as function of number of processors on Cray T3E and SGI Origin.

auxiliary and low level classes to handle explicit communication and input-output containing the *Pgrid2d*, *Communication*, *Utility*, *InOut* and *Timer* classes. The class diagram of the object-oriented model for a beam dynamics system is presented in Fig. 2. Here, run-time polymorphism is used to implement different external beam line elements. A single operation using the function of the beam-line-element base class can automatically select appropriate function from different concrete beam-line-element class object to execute. The inheritance relation defines a 'is' kind relationship among classes. The aggregation defines a relation that a class has an object of another class in its data member. The use defines a relation that a class uses an object of another class in its member function. The above object-oriented design is implemented using both Fortran 90 and the POOMA C++ framework [9, 10]. In this paper, we only show the simulation results using the F90/MPI code.

5 PERFORMANCE TESTS

The performance of the object-oriented code was tested on both the SGI/Cray T3E-900 and the SGI Origin 2000. Fig. 3 shows the largest number of particles on one processor and the least number of particles on one processor with and without dynamic load balance as a function of time steps on 16 processors. The total numerical particle number is 2 million with $64 \times 64 \times 64$ grids. We see here that with the dynamic load balance the difference between the maximum number of particles and the minimum number of particles has been drastically reduced. This suggests that the dynamic load balance algorithm in our paper works well. In Fig. 4, we also give a comparison of time cost on the Cray T3E as a function of number of processors using one-dimensional and two-dimensional parallel processor partitions. In this simulation, we have used 2.6 million particles and $64 \times 64 \times 64$ grids. The two-dimensional partition shows better scalability and is faster than the one-dimensional partition. This is because the two-dimensional processor partition has a more favorable surface-to-volume ratio. Communication cost is proportional to the surface area of the subdomain, whereas computation is proportional to its volume. Fig. 5 shows the time costs as a function of processor number on the SGI/Cray T3E-900 and on the SGI Origin 2000 for the same problem in Fig. 4. Good scalability of our object-oriented parallel particle-in-cell code has been achieved. The initial less time cost on the SGI Origin using 4 processors may be due to the larger cache size of this machine.

6 APPLICATION

As an application, we simulated the beam transport through three superconducting sections in a design of the APT

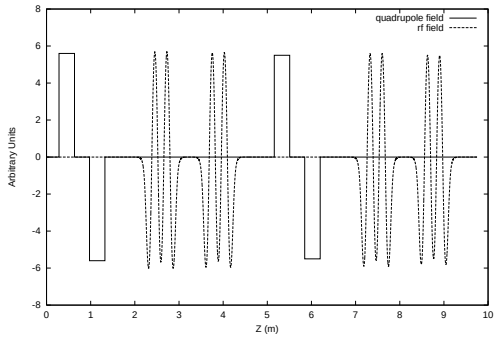

Figure 6: The external focusing and accelerating field in the superconducting linac.

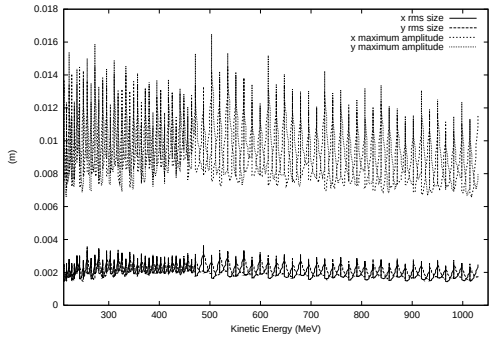

Figure 7: The transverse rms size and maximum amplitudes of beam as a function of its kinetic energy.

linac [11]. The major physical parameters in the design are listed in Table 1.

Energy gain	211.4 - 1.03 GeV
Beam current	0.1 A
Accelerator length	513.58 m
Quadrupole gradient	5.6-5.1, 5.5-6.05, 5.0-7.25 T/m
Accelerating gradient	4.3-4.54, 4.3-5.01, 5.25 MV/m
Synchronous phase	-30 to -35, -30 to -42, -30 degree

The external focusing and accelerating fields for the first two cyromodules are given in Fig. 6. A quadrupole-doublet FODO focusing lattice is used to provide transverse strong focusing and to reduce the focusing period comparing with singlet lattice. The external longitudinal rf field is from a MAFIA calculation of the 5 cell superconducting cavity. For the above physical parameters and external field, we have performed the simulation using 20 million numerical particles on $128 \times 128 \times 128$ grids. The initial distribution used here is a Gaussian distribution. Fig.7 gives the transverse beam rms size and maximum amplitudes as function of kinetic energy of beam. Initial increase of the transverse rms size of the beam is due to the decrease of transverse focusing in the transition section in order to achieve a smooth transition from the normal conducting linac focusing lattice (singlet) to the superconducting focusing lattice (doublet). A jump around 480 Mev is due to the jump of external focusing between the second section and the third section. The maximum transverse amplitudes set the lower bound of the minimum aperture that can be achieved in the design.

7 CONCLUSIONS

In the above sections, we presented an object-oriented three-dimensional parallel particle-in-cell program for beam dynamics simulation inside linear accelerators. This program employs domain decomposition method with MPI. A dynamic load balance scheme is implemented in the code. It also has better maintainability, reusability, and extensibility compared with conventional structure based code. Performance tests on the SGI/Cray T3E-900 and the SGI Origin 2000 show good scalability of the code. Using a symplectic integrator, we can also take advantage of the linear maps of external beam line elements obtained from modern Lie algebraic techniques. This code was successfully applied to the simulation of beam transport through three superconducting sections in the APT linac design.

8 ACKNOWLEDGMENTS

We would like to thank Drs. Babara Blind, Frank Krawczyk, and Thomas Wangler for RF superconducting data. This work was performed on the SGI/Cray T3E at NERSC and SGI Origin2000 at the ACL.

9 REFERENCES

[1] B. B. Godfrey, in Computer Applications in Plasma Science and Engineering, ed. by A. T. Drobot, Springer-Verlag, New York, 1991.

[2] A. Friedman, D. P. Grote and I. Haber, Phys. Fluids, B 4, 2203 (1992).

[3] R. Ryne and S. Habib, Part. Accl. 55, 365 (1996).

[4] T. Wrangler, Principles of RF Linear Accelerators, John Wiley & Sons, New York, 1998.

[5] A. J. Dragt, in: AIP Conference Proceedings 87, ed. R. A. Carrigan, F. R. Huson and M. Month (1982).

[6] R. W. Hockney and J. W. Eastwood, Computer Simulation Using Particles, Adam Hilger, New York, (1988).

[7] P. C. Liewer and V. K. Decyk, J. Comp. Phys., 85, 302 (1989).

[8] G. Booch, Object-Oriented Analysis and Design with Applications, Benjamin/Cummings, Menlo Park, CA, (1994).

[9] V. K. Decyk, C. D. Norton, and B. K. Szymanski, Scientific Programming, vol. 6, no. 4, IOS Press, Winter 1997, p. 363.

[10] W. Humphrey, R. Ryne, T. Cleland, et. al., in Computing in Object-Oriented Parallel Environments, ed. by D. Caromel, R. R. Oldehoeft, and M. Tholburn, Lecture Notes in Computer Science, vol. 1505, 1998.

[11] G. P. Lawrence, High-Power Proton Linac For APT: Status of Design and Development, in Proceeding of Linac98, Chicago, IL, 1998.