PARALLEL OBJECT-ORIENTED DESIGN IN FORTRAN FOR BEAM DYNAMICS SIMULATIONS*

J. Qiang†,R. D. Ryne, S. Habib, LANL, Los Alamos, NM

Abstract

In this paper we describe an object-oriented software design approach, using Fortran 90 (F90) and the Message Passing Interface (MPI), for modeling the transport of intense charged particle beams. The object-oriented approach improves the maintainability, resuability, and extensibility of the software, while the use of explicit message passing provides the freedom necessary to achieve high performance. Furthermore, an approach to object-oriented design based on Fortran will help those physicists familiar with procedure-oriented programming to make the transition to object-oriented design. In this paper we will describe the implementation of this approach and our success in developing two-dimensional and three-dimensional parallel beam dynamics codes that achieve high performance with only a small overhead associated with the object-oriented design.

1 INTRODUCTION

Object-oriented design is being widely applied in computer software engineering to implement complex codes which possess good maintainability, reusability, and extensibility. This technique also enables the encapsulation of detailed communication syntax in parallel computing, thereby reducing the extent of difficulty of parallel programming using MPI. In the parallel computing environment, such efforts have mostly been directed to the design of object-oriented frameworks using explicit message passing and C++ [1]. However, in the the accelerator physics community, Fortran still remains a popular language for demanding numerical simulations. Most popular used accelerator codes were programmed using Fortran based on procedure-oriented software design. It will be beneficial to the accelerator community to be able to to take advantage the object-oriented software design using Fortran language.

In this paper, we present an effort to implement object-oriented software design using F90 with MPI in the simulation of charged particle transport in accelerators. The paper is organized as follows: The physical system is described in Section 2, the implementation of object-oriented software design is presented in Section 3, parallel domain decomposition is discussed in Section 4, and performance tests are described in Section 5. We conclude by presenting an application to the simulation of high intensity beam transport through a superconducting linac.

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

† Email: jiqiang@lanl.gov

2 PHYSICAL SYSTEM

The physical system addressed in this paper consists of an intense charged particle beam and a linear accelerating system. The accelerating system contains three types of beam line elements: drift spaces, quadrupole magnets and rf gaps. The forces acting on the beam 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 a usual practice to take z to be the independent variable rather than the time t. The Vlasov equation is written as:

$$\frac{\partial f}{\partial z} + [H, f] \quad = \quad 0 \tag{1}$$

and the Poisson equation is

$$\nabla^2 \phi = -\rho/\epsilon \tag{2}$$

where f is the particle distribution function in phase space, $[,]$ is the Poisson bracket, H is the Hamiltonian of system with z as the independent variable, ϕ is the space charge potential from the Coulomb interaction, ρ is the charge density associated with the distribution function, and ϵ is the dielectric constant in vacuum. This system of equations is solved using a particle-in-cell method.

3 OBJECT-ORIENTED SOFTWARE DESIGN IN FORTRAN 90 FOR ACCELERATOR SIMULATION

In this study, parallel object-oriented software design is implemented using F90 and MPI. Object-oriented design is an approach encompassing the process of object-oriented decomposition [2]. In an objected-oriented design, after analysis of the (complex) physical system, the system is first decomposed into simpler physical modules. Next, objects are identified inside each module. Then, classes are abstracted from these objects. Each class has interfaces to communicate with the outside environment. Then relationships are 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 modules. 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.

Our implementation of the object-oriented software design methodology 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 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. 1.

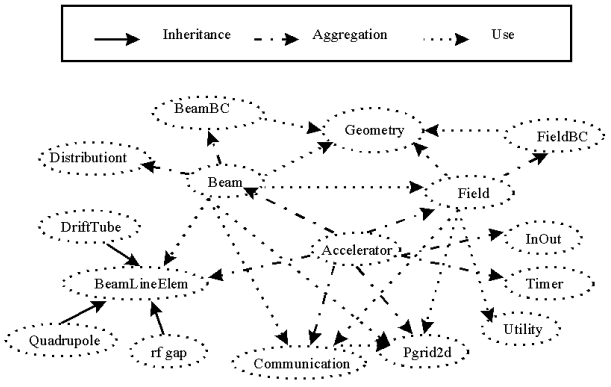

Figure 1: Class diagram of the accelerator beam dynamics system

The detailed method of writing objected-oriented programs using F90 was discussed by Decyk et al.[3]. In the following, we will give an example of using run-time polymorphism to implement the beam line elements in our simulation. In the beam dynamics simulation with F90, we define a base class *BeamLineElem*, and three derived classes for the drift, quadrupole magnet, and rf gap beam line elements. The scaled down sketch of this class is below:

```
module BeamLineElemclass
  use DriftTubeclass
  use Quadrupoleclass
  use Rfclass
  type BeamLineElem
    private
    type (DriftTube), pointer :: pdrift
    type (Quadrupole), pointer :: pquad
    type (Rf), pointer :: prf
  end type BeamLineElem
  interface assign_BeamLineElem
    module procedure assign_quad,&
           assign_rf,assign_drift
  end interface
contains
  function assign_drift(pdrift) &
           result(ppdrift)
  function assign_quadf(pquad) &
           result(ppquad)
  function assign_rf(prf) result(pprf)
  subroutine update_BeamLineElem(this,&
             z0,z1)
end module BeamLineElemclass
```

Since there is no direct support of inheritance in F90, we define a derived type in the *BeamLineElem* base class which contains three pointers to the derived classes as private data members. An overloaded function *assign_BeamLineElem* which includes three assignment functions is used to initialize the base *BeamLineElem* class object with different derived class object addresses. In each assignment function, only one pointer is initialized and the other two pointers are set to null. In the public function *update_BeamLineElem* of the base class, updating operations from derived classes are selected according to the different actual object association of pointers in the base class data member. The polymorphism is achieved by calling this subroutine with a constructed base *BeamLineElem* object in the application.

4 PARALLEL NUMERICAL ALGORITHM

In the following, we will describe the parallel numerical algorithm used in this study. A domain-decomposition approach is employed in the algorithm. For example, in the three-dimensional beam dynamics simulation, 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 each 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, respectively, of computation domain. The particles with spatial positions within the local computational boundary are assigned to the processor containing that part of physical domain.

The explicit communication in the parallel computation is encapsulated in the communication class. A particle manager function is defined to move the particles from one computation domain to another computation domain. A field manager function is defined to resize the grid number which is needed in solving Poisson's equation with open boundary condition. The grid exchange functions are defined to get the grid information from neighboring processors.

For the particles local to the computational domain, a symplectic integration scheme is employed to advance the particles[4]. The charge density is obtained using a cloud-

in-cell charge deposition scheme. The potential in Poisson's equation is obtained using Hockey's algorithm for open boundary condition[5]. The electrical field is calculated using a central finite difference scheme and interpolated onto the particles.

5 PERFORMANCE TEST

The performance of the object-oriented F90/MPI parallel codes was tested on both SGI/Cray T3E-900 and SGI Origin 2000. As a test of the overhead in object-oriented F90, which might be due to the use of pointers and dynamically allocated arrays, we give a comparison of the time costs on SGI/Cray T3E between the object-oriented code and the conventional procedure-based code in Fig. 2. We

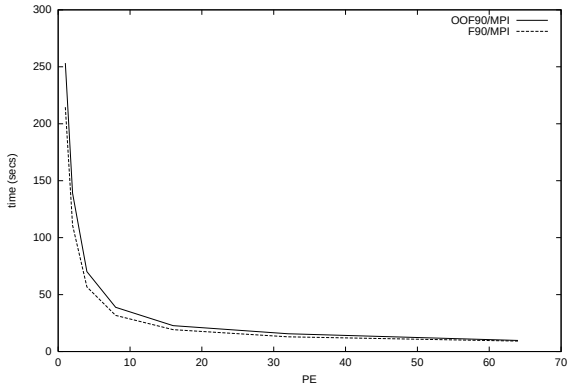

Figure 2: Time costs of object-oriented and procedure based F90/MPI codes as a function of PEs on T3E

note that even on a small number of processors, the overhead from object-oriented code is about 10%. This overhead decreases with increasing number of processors. As an example, in Fig. 3 we also give the time costs of the three-dimensional object-oriented F90/MPI code on Cray T3E and SGI Origin as a function of the number of processors. Good scalability is achieved on both machines.

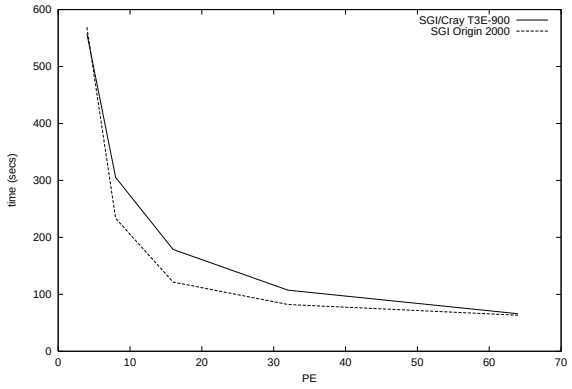

Figure 3: Time costs of 3-dimensional object-oriented F90/MPI code as a function of PEs on T3E and SGI Origin

6 APPLICATION

As an application, we simulate the beam transport through three super-conducting sections in a design of the APT linac[6]. Fig. 4 gives the transverse maximum amplitudes as a function of kinetic energy. These maximum amplitudes set the lower bound of the minimum aperture that can be achieved in the design.

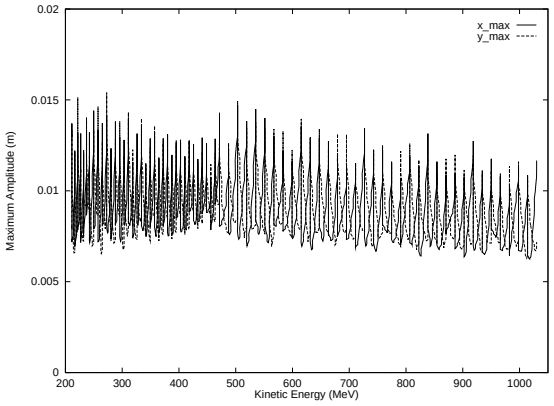

Figure 4: Transverse maximum amplitudes x and y of beam as a function of kinetic energy

7 CONCLUSIONS

In this paper we have described parallel object-oriented design in Fortran for beam dynamics simulations. As previously stated, our implementation with F90/MPI encapsulates the details of communication in low level auxiliary classes. This also provides the benefits of better maintainability, reusability and extensibility of software with small performance overhead.

8 ACKNOWLEDGMENTS

We thank Dr. Viktor Decyk for helpful discussions. This research used resources of the National Energy Research Scientific Computing Center and resources of the Advanced Computing Laboratory at Los Alamos National Laboratory.

9 REFERENCES

[1] G. Wilson, L. Paul, (ed.), Parallel Programming Using C++, MIT Press, Cambridge (1996).

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

[3] V. K. Decyk, C. D. Norton, and B. K. Szymanski, Computer Physics Communications 115, p.9 (1998).

[4] R. D. Ryne and S. Habib, "High Performance Computing for Beam Physics Applications", LA-UR-94-2904 (1994).

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

[6] T. P. Wangler, private communication (1998).