Object-Oriented Plasma PIC Simulation using Fortran 90 on Supercomputers

Charles D. Norton* Viktor K. Decyk† Boleslaw K. Szymanski‡

Abstract

One goal of the Numerical Turbulent Transport Project is to model a tokamak (fusion) plasma with $10^8 - 10^9$ particles, to explain anomalous transport of particles and energy. Since this ambitious high performance computing and communications (HPCC) project involves multiple institutions, and multidisciplinary collaborations, several project members have been investigating object-oriented techniques for designing particle-in-cell (PIC) codes. We summarize our experiences in this area using the modern constructs of the Fortran 90 programming language [2, 3, 4, 7, 8].

1 Introduction

Scientific application programming involves unifying abstract physical concepts and numerical models with sophisticated programming techniques that require patience and experience to master. Furthermore, codes typically written by scientists are constantly changing to model new physical effects. These factors can contribute to long development periods, unexpected errors, and software that is difficult to comprehend, particularly when multiple developers are involved.

The Fortran 90 programming language [5] addresses the needs of modern scientific programming by providing features that raise the level of abstraction, without sacrificing performance. Consider a 3D parallel plasma particle-in-cell program in Fortran 77 which will typically define the particles, charge density field, force field, and routines to push particles and deposit charge. This is a segment of the main program where many details have been omitted.

```
dimension part(idimp, npmax), q(nx, ny, nzpmx)
dimension fx(nx, ny, nzpmx), fy(nx, ny, nzpmx), fz(nx, ny, nzpmx)
data qme, dt /-1.,.2/
call push(part,fx,fy,fz,npp,noff,qtme,dt,wke,nx,ny,idimp,npmax,nzpmx)
call dpost(part,q,npp,noff,qme,nx,ny,idimp,npmax,nzpmx)
```

Note that the arrays must be dimensioned at compile-time. Also parameters must either be passed by reference, creating long argument lists, or kept in common and exposed to inadvertent modification. Such an organization is complex to maintain, especially as codes are modified for new experiments.

Using the new features of Fortran 90, abstractions can be introduced that clarify the organization of the code. The Fortran 90 version is more readable while designed for modification and extension.

---

*Jet Propulsion Laboratory, California Institute of Technology, MS 168-522, Pasadena, CA 91109-8099. The author's research was supported by a National Research Council Associateship, with additional resources from the Cornell Theory Center. Email: Charles.D.Norton@jpl.nasa.gov

†Department of Physics and Astronomy, University of California at Los Angeles, Los Angeles, CA 90024-1547 and Jet Propulsion Laboratory, California Institute of Technology, Pasadena CA 91109-8099. The author's research was carried out in part at UCLA and was sponsored by USDOE and NSF. It was also carried out in part at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. Email: decyk@physics.ucla.edu

‡Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180-3590. The author's research was supported under grants CCR-9216053 and CCR-9527151. Email: szymansk@cs.rpi.edu

```
use partition_module ; use plasma_module
type (species) :: electrons
type (scalarfield) :: charge_density
type (vectorfield) :: efield
type (slabpartition) :: edges
real :: dt = .2
call plasma_particle_push( electrons, efield, edges, dt )
call plasma_deposit_charge( electrons, charge_density, edges )
```

This style of object-oriented programming, where the basic data unit is an "object" that shields its internal data from misuse by providing public routines to manipulate it, allows such a code to be designed and written. Object-Oriented programming clarifies software while increasing safety and communication among developers, but its benefits are only useful for sufficiently large and complex programs.

While Fortran 90 is not an object-oriented language, the new features allow most of these concepts to be modeled directly. (Some concepts are more complex to emulate.) In the following, we will describe how object-oriented concepts can be modeled in Fortran 90, the application of these ideas to plasma PIC programming on supercomputers, and the future of Fortran programming (represented by Fortran 2000) that will contain explicit object-oriented features.

2 Modeling Object-Oriented Concepts in Fortran 90

Object-Oriented programming (OOP) has received wide acceptance, and great interest, throughout the computational science community as an attractive approach to address the needs of modern simulation. Proper use of OOP ensures that programs can be written safely, since the internal implementation details of the data objects are hidden. This allows the internal structure of objects and their operations to be modified (to improve efficiency perhaps), but the overall structure of the code using the objects can remain unchanged. In other words, objects are an encapsulation of data and routines.

These objects represent abstractions. Another important concept is the notion of inheritance, which allows new abstractions to be created by preserving features of existing abstractions. This allows objects to gain new features through some form of code reuse. Additionally, polymorphism allows routines to be applied to a variety of objects that share some relationship, but the specific action taken varies dynamically based on the object's type. These ideas are mechanisms for writing applications that more closely represent the problem at hand. As a result, a number of programming languages support OOP concepts in some manner.

Fortran 90 is well-known for introducing array-syntax operations and dynamic memory management. While useful, this represents a small subset of the powerful new features available for scientific programming. Fortran 90 is backward compatible with Fortran 77 and, since it is a subset of High Performance Fortran (HPF), it provides a migration path for data-parallel programming. Fortran 90 type-checks parameters to routines, so passing the wrong arguments to a function will generate a compile-time error. Additionally, the automatic creation of implicit variables can be suppressed reducing unexpected results.

However, more powerful features include derived-types, which allow user-defined types to be created from existing intrinsic types and previously defined derived-types. Many forms of dynamic memory management operations are now available, including dynamic arrays and pointers. These new Fortran 90 constructs are objects that know information such as their size, whether they have been allocated, and if they refer to valid data. Fortran 90 modules allow routines to be associated with types and data defined within the module. These modules can be used in various ways, to bring new functionality to program units. Components of the module can be private and/or public allowing interfaces to be constructed that control the accessibility of module components. Additionally, operator and routine overloading are supported (name reuse), allowing the proper routine to be called automatically based on the number and types of the arguments. Optional arguments are supported, as well as generic procedures that allow a single routine name to be used while the action taken differs based on the type of the parameter. All of these features can be used to support an object-oriented programming methodology [2, 3, 6].

Table 1: 3D Parallel Plasma PIC Experiments on Cornell's IBM SP2 (32PEs, 8M Particles).

Language	Compiler	P2SC Super Chips	P2SC Optimized
Fortran 90	IBM xlf90	622.60s	488.88s
Fortran 77	IBM xlf	668.03s	537.95s
C++	KAI KCC	1316.20s	1173.31s

3 Application: Plasma PIC Programming on Supercomputers

In the introduction, we illustrated how Fortran 77 features could be modeled using Fortran 90 constructs. In designing the PIC programs, basic constructs like particles (individually and collectively), fields (scalar and vector, real and complex), distribution operations, diagnostics, and partitioning schemes were created as abstractions using Fortran 90 modules. Fortran 90 objects are defined by derived types within modules where the public routines that operate on these objects are visible whenever the object is "used". (The private components of the module are only accessible within module defined routines.)

A portion of the species module, shown below, illustrates how data and routines can be encapsulated using object-oriented concepts. This module defines the particle collection, where the interface to the particle Maxwellian distribution routine is included.

```
module species_module
    use distribution_module ; use partition_module
    implicit none
    type particle
        private
        real :: x, y, z, vx, vy, vz           ! position & velocity components
    end type particle
    type species
        real :: qm, qbm, ek                   ! charge, charge/mass, kinetic energy
        integer :: nop, npp                   ! # of particles, # of particles on PE
        type (particle), dimension(:), pointer :: p   ! particle collection (dynamic)
    end type species
contains
    subroutine species_distribution(this, edges, distf)
        type (species), intent (out) :: this
        type (slabpartition), intent (in) :: edges
        type (distfcn), intent (in) :: distf
        ! subroutine body
    end subroutine species_distribution
    ! additional member routines
end module species_module
```

Some OOP concepts, such as inheritance, had limited usefulness while run-time polymorphism was used infrequently. Our experience has shown that these features, while sometimes appropriate for general purpose programming, do not seem to be as useful in scientific programming. Well-defined interfaces, that support manipulation of abstractions, were more important. More details on the overall structure of the code can be found in [7].

The wall-clock execution times for the 3D parallel PIC code written in Fortran 90, Fortran 77, and C++ are illustrated in Table 1. Although our experience has been that Fortran 90 continually outperforms C++ on complete programs, generally by a factor of two, others have performance results that indicate that C++ can sometimes outperform Fortran 90 on some computational kernels [1]. (In these cases, "expression templates" are introduced as a compile-time optimization to speed up complicated array operations.)

The most aggressive compiler options produced the fastest timings and are represented in the table. The KAI C++ compiler with +K3 -O3 -abstract_pointer spent over 2 hours in the compilation process. The IBM

F90 compiler with -O3 -qlanglvl=90std -qstrict -qalias=noaryovrlp used 5 minutes for compilation. (The KAI compiler is generally considered the most efficient C++ compiler when objects are used. This compiler generated slightly faster executables than the IBM C++ compiler.) Applying hardware optimization switches (-qarch=pwr2 -qtune=pwr2) introduced additional performance improvements specific to the P2SC processors.

We have found Fortran 90 very useful, and generally safer with higher performance than C++ and sometimes Fortran 77, for large problems on supercomputers. Fortran 90 derived-type objects improved cache utilization, for large problems, over Fortran 77. (The C++ and Fortran 90 objects had the same storage organization.) Fortran 90 is less powerful than C++, since it has fewer features and those available can be restricted to enhance performance, but many of the advanced features of C++ have not been required in scientific computing. Nevertheless, advanced C++ features may be more appropriate for other problem domains [4, 7].

4 Conclusions

The use of object-oriented concepts for Fortran 90 programming is very beneficial. The new features add clarity and safety to Fortran programming allowing computational scientists to advance their research, while preserving their investment in existing codes.

Our web site provides many additional examples of how object-oriented concepts can be modeled in Fortran 90 [6]. Many concepts, like encapsulation of data and routines can be represented directly. Other features, such as inheritance and polymorphism, must be emulated with a combination of Fortran 90's existing features and user-defined constructs. (Procedures for doing this are also included at the web site.) Additionally, an evaluation of compilers is included to provide users with an impartial comparison of products from different vendors.

The Fortran 2000 standard has been defined to include explicit object-oriented features including single inheritance, polymorphic objects, parameterized derived-types, constructors, and destructors. Other features, such as interoperability with C will simplify support for advanced graphics within Fortran 2000.

Parallel programming with MPI and supercomputers is possible with Fortran 90. However, MPI does not explicitly support Fortran 90 style arrays, so structures such as array subsections cannot be passed to MPI routines. The Fortran 90 programs were longer than the Fortran 77 versions (but more readable), and much shorter than the C++ programs because features useful for scientific programming are not automatically available in C++.

References

[1] J. R. Cary, S. G. Shasharina, J. C. Cummings, J. V. W. Reynders, and P. J. Hinker. Comparison of C++ and Fortran 90 for Object-Oriented Scientific Programming. *Computer Physics Communications*, 105:20–36, 1997.

[2] V. K. Decyk, C. D. Norton, and B. K. Szymanski. Introduction to Object-Oriented Concepts Using Fortran 90. Technical Report PPG–1560, Institute of Plasma and Fusion Research, UCLA Dept. of Physics and Astronomy, Los Angeles, CA 90095-1547, July 1996.

[3] V. K. Decyk, C. D. Norton, and B. K. Szymanski. Expressing Object-Oriented Concepts in Fortran 90. *ACM Fortran Forum*, 16(1):13–18, April 1997.

[4] V. K. Decyk, C. D. Norton, and B. K. Szymanski. How to Express C++ Concepts in Fortran 90. Technical Report PPG–1569, Institute of Plasma and Fusion Research, UCLA Dept. of Physics and Astronomy, Los Angeles, CA 90095-1547, Feb. 1997.

[5] T. M. R. Ellis, I. R. Philips, and T. M. Lahey. *Fortran 90 Programming.* Addison–Wesley, Reading, MA, 1994.

[6] C. D Norton, V. K. Decyk, and B. K. Szymanski. High Performance Object-Oriented Programming in Fortran 90. Internet Web Pages, October 1996. http://www.cs.rpi.edu/~szymansk/oof90.html.

[7] C. D. Norton, V. K. Decyk, and B. K. Szymanski. High Performance Object-Oriented Scientific Programming in Fortran 90. In M. Heath, et. al, editor, *Proc. Eighth SIAM Conference on Parallel Processing for Scientific Computing*, Minneapolis, MN, March 14–17 1997. CD-ROM.

[8] C. D. Norton, B. K. Szymanski, and V. K. Decyk. Object Oriented Parallel Computation for Plasma Simulation. *Communications of the ACM*, 38(10):88–100, October 1995.