UPIC: A Framework for Massively Parallel Particle-in-cell Codes

Viktor K. Decyk [*]

Department of Physics and Astronomy, University of California, Los Angeles Los Angeles, CA 90095-1547 USA

Abstract

The UCLA Parallel Particle-in-Cell (UPIC) Framework, is designed to provide trusted components for building a variety of parallel Particle-in-Cell (PIC) codes. It is based on the idea that most PIC codes share common algorithms, such as particle pushing and deposit subroutines, solvers for electromagnetic fields, as well as parallel data management, such as passing particles and fields between processors. The framework provides the optimized common algorithms, as well as various skeleton (template) main programs. The user provides specialized subroutines that are specific to his or her problem. It is based on Fortran95, but low level subroutines can easily be called from other languages. Codes based on the framework can run on as few as one processor to as many as 2000, and examples will be presented.

Key words: particle-in-cell codes; parallel computing; design patterns

1. Introduction

Computer processing power has increased inexorably for several decades. It is now possible to perform calculations scarcely imaginable not many years ago. But the increased capability comes at a price: the computer architecture is also increasingly complex and more difficult to program. Furthermore, as more computer power is available, scientists desire to increase the complexity of their computer software as well. The result is that high performance software increasingly requires a team effort, not only for the physics but for the software development. The situation is especially difficult for graduate students.

In the past, graduate students in computational plasma physics would inherit a code from a previous student, then modify it in some fashion for his or her thesis. This process often took a year or more, because it required considerable understanding to know what could be safely changed. Furthermore the inherited code was often obscurely written and poorly documented, partly because graduate students in physics are rarely trained in software development and partly because funding agencies and reviewers are primarily interested in scientific results, not software development. This problem is even worse for parallel programs, which require a special expertise of its own. Particle-in-Cell (PIC) codes are particularly challenging because they are very computational demanding, and high performance is very important: an order of magnitude difference in performance can make a substantial difference in the kind of calculation which can be done.

For these reasons, we are developing a framework at UCLA for the rapid construction of new parallel PIC codes [1]. A framework here means a collection of trusted components along with a collection of skeleton main programs which can be augmented by individual scientists for their specific problem. The framework is designed to support a variety of problems by providing “Lego” blocks which are common to many different PIC codes. The user provides the specialized procedures that are unique for his or her

[*] Corresponding author.
Email address: decyk@physics.ucla.edu (Viktor K. Decyk).

problem.

2. Software Design

The framework is designed in layers that can be used with different programming styles. Because high performance is so important in PIC codes, the lowest layers are written in Fortran77. This is partly due to the 30 year legacy of PIC codes that already existed at UCLA. It is also based on the observation that Fortran77 tends to give the highest performance of any language, because it is a relatively primitive language that compilers can transform and optimize well. Pointers and pointer aliasing are not permitted and the array structures are easier for compilers to analyze than pointer arithmetic. Furthermore, Fortran77 compilers have been in existence for more than 50 years and are very mature.

Another reason for the use of Fortran77 at the lowest layer is that it can be called by many other languages, and thus serves well as a high performing, lowest common denominator. A few simple main skeleton programs in Fortran77 are available for those who prefer to work at this layer. However, Fortran77 is an inflexible, unsafe language in many ways, and this layer is not intended to be used directly except for simple codes.

On top of this layer, there exists a library of Fortran90 wrapper functions, which hide some of the complexity of the Fortran77 layer. These wrapper functions call one or more Fortran77 functions, but offer simpler arguments and enable strict type checking. Fortran90 supports dynamic memory, so small work or scratch arrays needed at the lowest layer are declared and hidden here, and are not visible at higher layers. Fortran90 arrays are actually objects that know their size, so that this information can also be hidden here. The Fortran90 arrays also know if they have been allocated, so that the wrapper functions can provide safety checks. Finally, the wrapper functions can make use of pointer arrays, which would degrade performance if pointers had been used in the Fortran77 layer. The sample skeleton programs available for this layer still use a procedural style of programming, familiar to many scientists, but they are much easier to use than Fortran77.

For complex software projects, there is an object-oriented layer, which makes use of Design Patterns [2]. Although Fortran90 is not an object-oriented language, the missing features of the language can be emulated [3-4]. The major piece missing is inheritance, and this feature can be largely avoided by use of Design Patterns, which emphasizes object composition over inheritance [5]. The major advantage of object-oriented design is that larger blocks of code can be encapsulated and used as black boxes without understanding its implementation. It is particularly useful when multiple authors are contributing to a project, where each author has a well defined area of responsibility. Further details about the object-oriented layer are given in [1]. The separation of the framework into layers, allows the wrapper layer and the object-oriented layer to be written in another language such as C++, or Fortran2003, which is a fully object-oriented language.

There is no performance penalty in using the wrapper or object-oriented layer, because the procedures in these layers do not perform any substantial calculations. They merely make decisions and delegate the work to some underlying optimized Fortran77 subroutine.

3. Multiple Models, Levels of Accuracy, and Architectures

The framework supports different kinds of electromagnetic models. The most basic is the electrostatic model, where the electric field is obtained from solving Poisson's equation. The Darwin model, which neglects the transverse displacement field in Maxwell's equation, uses Poisson solvers for the induced electric and magnetic fields. The most complete model is the electromagnetic model obtained from solving Maxwell's equation. Currently, all the solvers are spectral. In additional to periodic boundary conditions using FFTs, combinations of Dirichlet and Neumann boundary conditions are also supported using various Discrete Sine and Cosine transforms. Finally, vacuum boundary conditions for isolated systems are supported for the electrostatic case using FFT-based convolutions. Particle pushing and deposit subroutines are available which correspond to the various electromagnetic models, both with and without relativistic effects.

It is important to allow multiple levels of accuracy in order to know when the level of accuracy is sufficient for a given problem. Therefore the framework is designed to provide both high and medium accuracy algorithms. One important part of this is support for quadratic interpolation. Quadratic interpolation reduces non-physical aliasing arising from the use of the grid, but its cost is typically twice the cost of the commonly used linear interpolation.

One of the most important feature the framework is designed to support is parallel processing. For distributed memory computers, a domain decomposition is used [7]. Regions of space which contain approximately equal number of particles but possibly unequal volume, are defined. Particles are moved into or out of these regions as the calculation proceeds. This ensures that the particle calculations are load balanced. The spectral field solvers, however, perform better if the regions are of equal volume, and so a uniform partition is used to solve the fields.

The Message-Passing Interface (MPI) is the primary mechanism for parallel processing. The philosophy used is to separate the physics procedures from the communication. A library of MPI routines, called PLIB, is used to move data. This includes a particle manager to move particles to the proper processor, a field manager which handles guard cells and transposition of data, and a partition manager which moves data between different partitions, and I/O subroutines. The reason for the separation of data movement from physics was originally to allow the physicist to write or modify the physics modules without having to know about the intricacies of message-passing. It turned out that performance did not suffer, and the separation meant that the same communication pattern could be reused in different contexts.

For shared memory computers, threads can be used to run the most time-consuming subroutines in parallel. A simple multi-tasking library, similar to the Cray Multi-tasking library, was implemented, based on Posix threads. The particle push, charge and current deposit, and FFT procedures can be run in parallel as tasks. The framework can also run with mixed multi-tasking and MPI messaging, where multi-tasking is used on a multiple cpu shared memory node, and message-passing is used between such nodes.

4. Codes based on the Framework

The most important code based on the UPIC Framework is QuickPIC [6]. This code is used to study plasma based accelerators. The code makes use of the quasi-static approximation, where Maxwell's equation in the transverse direction can be approximated by Poisson equations for the electric and magnetic field. The transverse equations are solved by a 2D PIC code which calculates the plasma wake, which is then used by a 3D code to advance the driver. QuickPIC has produced agreement with experiment and runs more than 100 times faster than a conventional PIC code for this problem.

The framework has also been used to construct the parallel DRACO code for modeling ion propulsion for spacecraft [8]. The ion propulsion code made use of the particle subroutines and the wrapper layer, but added a more realistic field solver using an immersed finite-element technique. This code was one of the first simulations ever performed for a realistic spacecraft including the entire solar array panel.

Graduate students in plasma physics have used the framework to model their experiments. One such thesis involved the study of symmetric neutralized ion beams [9]. This is an exploratory concept which may be of use as a compact alternative to conventional neutral beam injection in magnetic confinement fusion devices.

Acknowledgements

This work was supported by the U. S. Department of Energy under the SCIDAC program.

References

[1] V. K. Decyk and C. D. Norton, Computer Phys. Comm. 164 (2004) 80.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, Reading, MA, 1995.

[3] V. K. Decyk, C. D. Norton, and B. K. Szymanski, Scientific Programming 6(4) (1997) 363.

[4] V. K. Decyk and C. D. Norton, Scientific Programming 12 (2004) 45.

[5] V. K. Decyk and H. Gardner, submitted to Computer Physics Communications, 2006.

[6] C. Huang, V. K. Decyk, C. Ren, M. Zhou, W. Lu, W. B. Mori, J. H. Cooley, T. M. Antonsen, Jr., and T. Katsouleas, to be published in J. Computational Phys., 2006.

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

[8] J. Wang, Y. Cao, R. Kafafy, J. Pierru, and V. K. Decyk, IEEE Transactions on Plasma Science 34 (2006) 2148.

[9] Nathaniel K. Hicks and Alfred Y. Wong, to be published in J. Fusion Energy, 2006.