OSIRIS: a three-dimensional, fully relativistic particle in cell code for modeling plasma based accelerators

R.A.Fonseca[1], L.O.Silva[1], F.S.Tsung[2], V.K.Decyk[2], W.Lu[2], C.Ren[2], W.B.Mori[2], S.Deng[3], S.Lee[3], T.Katsouleas[3], and J.C.Adam[4]

[1] GoLP/CFP, Instituto Superior Técnico, Lisboa, Portugal,
zamb@cfp.ist.utl.pt,
WWW home page: http://cfp.ist.utl.pt/golp/
[2] University of California, Los Angeles, USA
WWW home page: http://exodus.physics.ucla.edu
[3] University of Southern California, Los Angeles, USA
[4] École Polytechnique, Paris. France

Abstract. We describe OSIRIS, a three-dimensional, relativistic, massively parallel, object oriented particle-in-cell code for modeling plasma based accelerators. Developed in Fortran 90, the code runs on multiple platforms (Cray T3E, IBM SP, Mac clusters) and can be easily ported to new ones. Details on the code's capabilities are given. We discuss the object-oriented design of the code, the encapsulation of system dependent code and the parallelization of the algorithms involved. We also discuss the implementation of communications as a boundary condition problem and other key characteristics of the code, such as the moving window, open-space and thermal bath boundaries, arbitrary domain decomposition, 2D (cartesian and cylindric) and 3D simulation modes, electron sub-cycling, energy conservation and particle and field diagnostics. Finally results from three-dimensional simulations of particle and laser wakefield accelerators are presented, in connection with the data analysis and visualization infrastructure developed to post-process the scalar and vector results from PIC simulations.

1 Introduction

Based on the highly nonlinear and kinetic processes that occur during high-intensity particle and laser beam-plasma interactions, we use particle-in-cell (PIC) codes [1, 2], which are a subset of the particle-mesh techniques, for the modeling of these physical problems. In these codes the full set of Maxwell's equations are solved on a grid using currents and charge densities calculated by weighting discrete particles onto the grid. Each particle is pushed to a new position and momentum via self-consistently calculated fields. Therefore, to the extent that quantum mechanical effects can be neglected, these codes make no physics approximations and are ideally suited for studying complex systems with many degrees of freedom.

Achieving the goal of one to one, two and three dimensional modeling of laboratory experiments and astrophysical scenarios, requires state-of-the-art computing systems. The rapid increase in computing power and memory of these systems that has resulted from parallel computing has been at the expense of having to use more complicated computer architectures. In order to take full advantage of these developments it has become necessary to use more complex simulation codes. The added complexity arises for two reasons. One reason is that the realistic simulation of a problem requires a larger number of more complex algorithms interacting with each other than in a simulation of a rather simple model system. For example, initializing an arbitrary number of lasers or particle beams in 3D on a parallel computer is a much more difficult problem than initializing one beam in 1D or 2D on a single processor. The other reason is that the computer systems, e.g., memory management, threads, operating systems, are more complex and as a result the performance obtained from them can dramatically differ depending on the code strategy. Parallelized codes that handle the problems of parallel communications and parallel IO are examples of this. The best way to deal with this increased complexity is through an object-oriented programming style that divides the code and data structures into independent classes of objects. This programming style maximizes code reusability, reliability, and portability.

The goal of this code development project was to create a code that breaks up the large problem of a simulation into a set of essentially independent smaller problems that can be solved separately from each other. This allows individuals in a code development team to work independently. Object oriented programming achieves this by handling different aspects of the problem in different modules (classes) that communicate through well-defined interfaces.

This effort resulted in a new framework called OSIRIS, which is a fully parallelized, fully implicit, fully relativistic, and fully object-oriented PIC code, for modeling intense beam plasma interactions.

2 Development

The programming language chosen for this purpose was Fortran 90, mainly because it allows us to more easily integrate already available Fortran algorithms into this new framework that we call OSIRIS. We have also developed techniques where the Fortran 90 modules can interface to C and C++ libraries, allowing for the inclusion of other libraries that do not supply a Fortran interface. Although Fortran 90 is not an object-oriented language *per se*, object-oriented concepts can be easily implemented [3–5] by the use of polymorphic structures and function overloading.

In developing OSIRIS we followed a number of general principles in order to assure that we were building a framework that would achieve the goals stated above. In this sense all real physical quantities have a corresponding object in the code making the physics being modeled clear and therefore easier to maintain, modify and extend. Also, the code is written in a way such that it is largely

independent from the dimensionality or the coordinate system used, with much of the code reused in all simulation modes.

Regarding the parallelization issues, the overall structure allows for an arbitrary domain decomposition in any of the spatial coordinates of the simulation, with an effective load balancing of the problems in study. The input file defines only the global physical problem to be simulated and the domain decomposition desired, so that the user can focus on the actual physical problem and does not need to worry about parallelization details. Furthermore, all classes and objects refer to a single node (with the obvious exception of the object responsible for maintaining the global parallel information), which can be realized by treating all communication between physical objects as a boundary value problem, as described below. This allows for new algorithms to be incorporated into the code, without a deep understanding of the underlying communication structure.

3 Design

3.1 Object-Oriented Hierarchy

Figure 1 shows the class hierarchy of OSIRIS. The main physical objects used are particle objects, electromagnetic field objects, and current field objects. The particle object is an aggregate of an arbitrary number of particle species objects. The most important support classes are the variable-dimensionality-field class, which is used by the electromagnetic and current field classes and encapsulates many aspects of the dimensionality of a simulation, and the domain-decomposition class, which handles all communication between nodes.

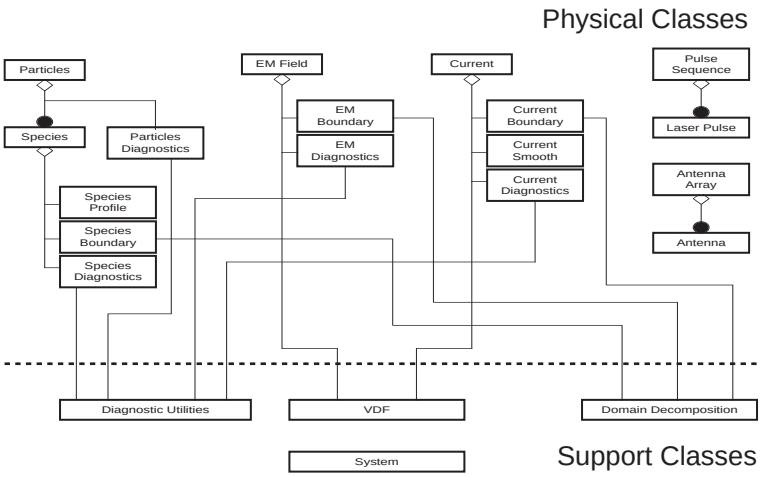

Fig. 1. Osiris main class hierarchy

Benchmarking of the code has indicated that the additional overhead from using an object oriented framework in Fortran 90 leads to only a 12% slowdown in speed.

3.2 Parallelization

The parallelization of the code is done for distributed memory systems, and it is based on the MPI message-passing interface [10]. We parallelize our algorithms by decomposing the simulation space evenly across the available computational nodes. This decomposition is done by dividing each spatial direction of the simulation into a fixed number of segments (N_1, N_2, N_3). The total number of nodes being used is therefore the product of these three quantities (or two quantities for the 2D simulations).

The communication pattern follows the usual procedure for a particle-mesh code [11]. The grid quantities are updated by exchanging (electric and magnetic fields) or adding (currents) the ghost cells between neighboring nodes. As for the particles, those crossing the node boundary are counted and copied to a temporary buffer. Two messages are then sent, the first with the number of particles, and the second with the actual particle data. This strategy allows for not setting an *a priori* limit on the number of particles being sent to another node, while maintaining a reduced number of messages. Because most of the message are small, we are generally limited by the latency of the network being used. To overcome this, and whenever possible, the messages being sent are packed into a single one, achieving in many cases twice the performance.

We also took great care in encapsulating all parallelization as boundary value routines. In this sense, the boundary conditions that each physical object has can either be some numerical implementation of the usual boundary conditions in these problems or simply a boundary to another node. The base classes that define grid and particle quantities already include the necessary routines to handle the later case, greatly simplifying the implementation of new quantities and algorithms.

3.3 Encapsulation of System Dependent Code

For ease in porting the code to different architectures, all code that is machine dependent is encapsulated in the system module. At present we have different versions of this module for running on the Cray T3E, the IBM SP, and for Macintosh clusters, running on both MacOS 9 (MacMPI [8]) and MacOS X (LAM/MPI [9]) clusters. The later is actually a fortran module that interfaces with a POSIX compliant C module and should therefore compile on most UNIX systems, allowing the code to run on PC-based (Beowulf) clusters. The MPI library has also been implemented on all these systems requiring no additional effort.

3.4 Code Flow

Figure 2 shows the flow of a single time step on a typical OSIRIS run. It closely follows the typical PIC cycle [2]. The loop begins by executing the diagnostic routines selected (diagnostics). It follows by pushing the particles using the updated values for the fields and depositing the current (advance deposit). After this step, the code updates the boundaries for particles and currents, communicating with neighboring nodes if necessary. A smoothing of the deposited currents, according to the specified input file, follows this step. Finally, the new values of the Electric and Magnetic field are calculated using the smoothed current values, and its boundaries are updated, again communicating with neighboring nodes, if necessary.

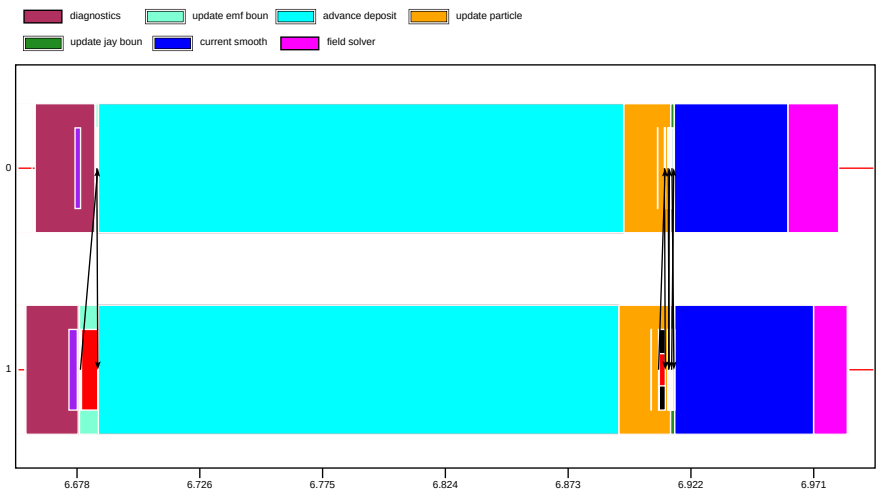

Fig. 2. A typical cycle, one time step, in an OSIRIS 2 node run. The arrows show the direction of communication between nodes.

If requested, at the end of each loop, the code will write restart information, allowing the simulation to be restarted later on at this time step.

4 OSIRIS Framework

The code is fully relativistic and it presently uses either the charge-conserving current deposition schemes from ISIS [6] or TRISTAN [7]. We have primarily adopted the charge-conserving current deposition algorithms because they allow the field solve to be done locally, i.e., there is no need for a Poisson solve. The code uses the Boris scheme to push the particles, and the field solve is done locally using a finite difference solver for the electric and magnetic fields in both space and time.

In its present state the code contains algorithms for 2D and 3D simulations in Cartesian coordinates and for 2D simulations in azimuthally symmetric cylindrical coordinates, all of which with 3 components in velocity (i.e. both 2D modes are indeed $2\frac{1}{2}$D or 2D3V algorithms). The loading of particles is done by distributing the particles evenly on the cell, and varying the individual charge of each particle according to the density profile stipulated. Below a given threshold no particles are loaded. The required profile can be specified by a set of multiplying piecewise linear functions and/or by specifying Gaussian profiles. The initial velocities of the particles are set according to the specified thermal distribution and fluid velocity. The code also allows for the definition of constant external electric and magnetic fields.

The boundary conditions we have implemented in OSIRIS are: conducting, and Lindmann open-space boundaries for the fields [17], and absorbing, reflective, and thermal bath boundaries for the particles (the later consists of reinjecting any particle leaving the box with a velocity taken from a thermal distribution). Furthermore, periodic boundary conditions for fields and particles are also implemented.

This code also has a moving window, which makes it ideal for modeling high-intensity beam plasma interactions where the beam is typically much shorter than the interaction length. In this situation simulation is done in the laboratory reference frame, and simulation data is shifted in the direction opposite to the window motion defined whenever an integer number of cells corresponds to this motion in the number of time steps elapsed. Since this window moves at the speed of light in vacuum no other operations are required. The shifting of data is done locally on each node, and boundaries are updated using the standard routines developed for handling boundaries, thus taking care of moving data between adjacent nodes. The particles leaving the box from the back are removed from the simulation and the new clean cells in the front of the box are initialized as described above.

OSIRIS also incorporates the ability to launch EM waves into the simulation, either by initializing the EM field of the simulation box accordingly, or by injecting them from the simulation boundaries (e.g. antennas). Moreover, a subcycling scheme [18] for heavier particles has been implemented, where the heavier species are only pushed after a number of time steps using the averaged fields over these time steps, thus significantly decreasing the total loop time.

A great deal of effort was also devoted to the development of diagnostics for this code that goes beyond the simple dumps of simulation quantities. For all the grid quantities envelope and boxcar averaged diagnostics are implemented; for the EM fields we implemented energy diagnostics, both spatially integrated and resolved; and for the particles phase space diagnostics, total energy and energy distribution function, and accelerated particle selection are available. The output data uses the HDF [12] file format. This is a standard, platform independent, self-contained file format, which gives us the possibility of adding extra information to the file, like data units and iteration number, greatly simplifying the data analysis process.

5 Visualization and Data-Analysis Infrastructure

It is not an exaggeration to say that visualization is a major part of a parallel computing lab. The data sets from current simulations are both large and complex. These sets can have up to five free parameters for field data: three spatial dimensions, time and the different components (i.e., E_x, E_y, and E_z). For particles, phase space has seven dimensions: three for space, three for momentum and one for time. Plots of y versus x are simply not enough. Sophisticated graphics are needed to present so much data in a manner that is easily accessible and understandable.

We developed a visualization and analysis infrastructure [13] based on IDL (Interactive Data Language). IDL is a 4GL language, with sophisticated graphics capabilities, and it is widely used in areas such as Atmospheric Sciences and Astronomy. It is also available on several platforms and supported in a number of systems, ranging from Solaris to the MacOS.

While developing this infrastructure we tried simplifying the visualization and data analysis as much as possible, making it user-friendly, automating as much of the process as possible, developing routines to batch process large sets of data and minimizing the effort of creating presentation quality graphics. We implemented a full set of visualization routines for one, two and three-dimensional scalar data and for two and three dimensional vector data. These include automatic scaling, dynamic zooming and axis scaling, integration of analysis tools, animation tools, and can be used either in batch mode or in interactive mode. We have also developed a comprehensive set of analysis routines that include scalar and vector algebra for single or multiple datasets, boxcar averaging, spectral analysis and spectral filtering, k-space distribution function, envelope analysis, mass centroid analysis and local peak tools.

6 Results

The code has been successfully used in the modeling of several problems in the field of plasma based accelerators, and has been run on a number of architectures. Table 1 shows the typical push times on two machines, one supercomputer and one computer cluster.

Table 1. Typical push time for two machines, in two and three dimensions. Values are in μs/particle $\times$ node

Machine	2D push time	3D push time
Cray T3E-900	4.16	7.56
EP^2 Cluster	4.96	9.82

We have also established the energy conservation of the code to be better than 1 part in 10^5. This test was done in a simulation where we simply let a warm plasma evolve in time; in conditions where we inject high energy fluxes into the simulation (laser or beam plasma interaction runs) the results are better. Regarding the parallelization of the code, extensive testing was done on the EP^2 cluster [19] at the IST in Lisbon, Portugal. We get very high efficiency, (above 91% in any condition), proving that the parallelization strategy is appropriate. Also note that this is a computer cluster running a 100 Mbit/s network, and that the efficiency on machines such as the Cray T3E is even better

One example of a three-dimensional modeling of a plasma accelerator is presented on figure 3. This is a one-to-one modeling of the E-157 Experiment [14] done at the Stanford Linear Accelerator Center, where a 30 GeV beam is accelerated by 1 GeV. The figure shows the Lorentz forces acting on the laser beam e.g. $\mathbf{E} + \mathbf{z} \times \mathbf{B}$, where $\mathbf{z}$ is the beam propagation direction, and we can clearly identify the focusing /defocusing and accelerating/decelerating regions

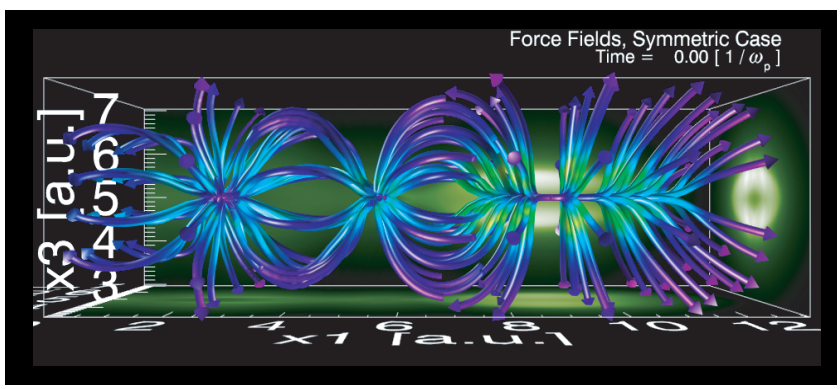

Fig. 3. Force field acting on the 30 GeV SLAC beam inside a plasma column.

Another example of the code capabilities is the modeling of the Laser Wakefield Accelerator (LWFA). In the LWFA a short ultrahigh intensity laser pulse drives a relativistic electron plasma wave. The wakefield driven most efficiently when the laser pulse length $L = c\tau$ is approximately the plasma wavelength $\lambda_p = 2\pi c/\omega_p$ - Tajima-Dawson mechanism [15]. Figure 4 shows the plasma wave produced by a 800 nm laser pulse with a normalized vector potential of 2.16, corresponding to an intensity of 10^{19}W/cm^2 on focus, and a duration of 30 fs, propagating in an underdense plasma.

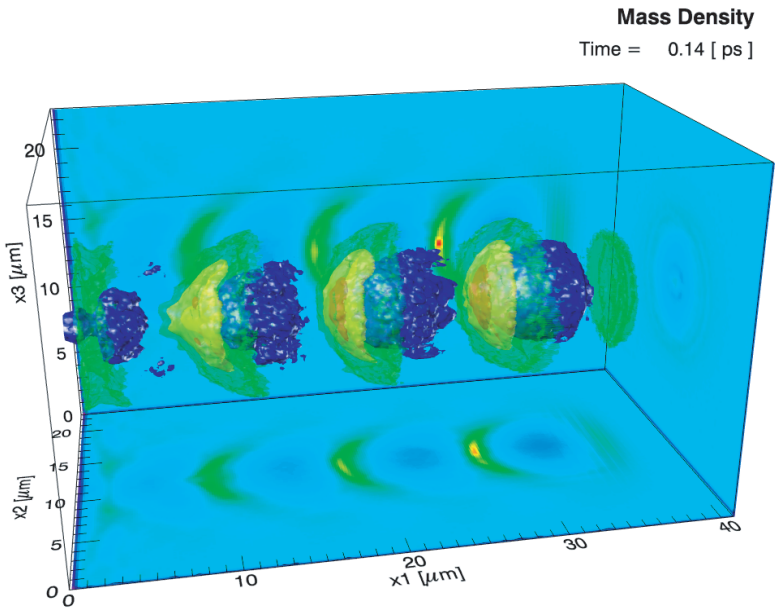

Fig. 4. Plasma Wave produced in the LWFA. Isosurfaces shown for values of 0.5, 1.2, 2.0 and 5.0 normalized to the plasma background density.

7 Future Work

In summary, we have presented the OSIRIS framework for modeling plasma based accelerators. This is an ongoing effort; future developments will concentrate on the implementation of true open-space boundaries [16] and ionization routines. Regarding the visualization and data analysis infrastructure, a Web-Driven visualization portal will be implemented on the near future, allowing for efficient remote data analysis on clusters.

8 Acknowledgements

This work was supported by DOE, NSF (USA), FLAD, GULBENKIAN, and by FCT (Portugal) under grants PESO/P/PRO/40144/2000, PESO/P/INF/40146 /2000, CERN/P/FIS/40132/2000, and POCTI/33605/FIS/2000.

References

1. Dawson, J.M.: Particle simulation of plasmas. Rev. Mod. Phys., vol.55, no. 2, April 1983, p. 403-47.

2. Birdsall, C.K., Langdon, A.B.: Plasma physics via computer simulation. Bristol, UK: Adam Hilger, 1991, xxvi+479 pp.
3. Decyk, V. K., Norton, C. D., Szymanski, B. K.: How to express C++ concepts in Fortran 90. Scientific Programming, Vol. 6, no. 4, 1998, p. 363.
4. Decyk, V. K., Norton, C. D., Szymanski, B. K.: How to support inheritance and run-time polymorphism in Fortran 90. Comp. Phys. Com., no. 115, 1998, pp. 9-17.
5. Gray, M. G., Roberts, R. M.: Object-Based Programming in Fortran 90. Computers in Physics, vol. 11, no. 4, 1997, pp. 355-361.
6. Morse, R.L., Nielson, C.W.: Numerical simulation of the Weibel instability in one and two dimensions. Phys. Fluids, vol.14, no.4, April 1971. pp.830-40.
7. Villasenor, J.; Buneman, O.: Rigorous charge conservation for local electromagnetic field solvers. Computer Physics Communications, vol.69, no. 2-3, March-April 1992.
8. Decyk, V.K., Dauger, D.E.: How to Build an AppleSeed: A Parallel Macintosh Cluster for Numerically Intensive Computing. Presented at the International School for Space Simulation ISSS-6, Garching, Germany September 2001; also at `http://exodus.physics.ucla.edu/appleseed/appleseed.html`
9. `http://www.lam-mpi.org/`
10. Message Passing Interface Forum.: MPI: A message-passing interface standard. International Journal of Supercomputer Applications, vol. 8, no. 3-4, 1994.
11. Gropp, W., Lusk, E., Skjellum, A.: Using MPI. MIT Press, 1999. xxii+371 pp.
12. `http://hdf.ncsa.uiuc.edu/`
13. Fonseca, R. *et al.*: Three-dimensional particle-in-cell simulations of the Weibel instability in electron-positron plasmas. IEEE transactions in plasma science Special Issues on Images in Plasma Science, 2002
14. Muggli, P. *et al.*: Nature, vol. 411, 3 May 2001
15. Tajima, T., Dawson, J. M.: Laser Electron Accelerator, Phys. Rev. Lett., vol. 43, 1979, pp. 267-270
16. Vay, J.L.: A new Absorbing Layer Boundary Condition for the Wave Equation. J. Comp. Phys., no. 165, 2000, pp. 511-521.
17. Lindmann, E. L.: Free-space boundary conditions for the time dependent wave equation. J. Comp. Phys., no. 18, 1975, pp. 66-78.
18. Adam, J. C.; A. Gourdin Serveniere, and A. B. Langdon: Electron sub-cycling in particle simulation of Plasmas J. Comp. Phys., no. 47, 1982, pp. 229-244.
19. `http://cfp.ist.utl.pt/golp/epp/`