VORPAL - A MULTIDIMENSIONAL CODE FOR SIMULATING ADVANCED ACCELERATION CONCEPTS

C. Nieter, J.R. Cary, University of Colorado at Boulder, Boulder, CO 80309, USA

Abstract

VORPAL is a new simulation code under development for simulation of advanced acceleration concepts such as Laser Wake Field Acceleration. VORPAL makes use of object oriented programming techniques to achieve greater flexibility and extensibility. For example, with VORPAL, one can set the physical dimension of the simulation at run time rather than compile time. This allows the user to simulate an advanced acceleration scenario rapidly in 2D to look for qualitative results, then move to 3D with the same code and nearly the same input file for more detailed simulations. The VORPAL framework can support multiple particle models (fluid, PIC), but at present only fluid representations are in place. VORPAL is designed to run on most flavors of UNIX and will run on both serial and parallel machines, including Beowulf clusters. VORPAL stores data in HDF5 files, allowing for subsequent visualization by a number of packages, including RSI's IDL and OpenDX. VORPAL will be applied to problems of laser-plasma interactions in the near future.

1 DIMENSION FREE

VORPAL is a dimension free code. What we mean by this is that the same code base will support simulations of one, two, and three dimensional systems. This is done by templating the code over dimension. The simultaneous implementation of of arbitrary dimensional prevents us from using nested loops to perform the updates.

We deal with the issue of indexing in multiple dimensions by using a generalization of an iterator. We define a class called VpFieldIter, which holds the index of a 1D array corresponding to the multiple indices of a multi-dimensional array. This class contains a bump method, which moves the index a given number of cells in a given direction. These iterators hold a pointer to a specific field whose data they can access for either reading or writing.

A computational method is then implemented by creating collections of these iterators that represent some finite differencing of an equation of motion. For example, suppose that we wanted to solve the following differential equation,

$$\frac{\partial \Phi}{\partial t} = \alpha \nabla^2 \Phi, \tag{1}$$

by finite differencing. The change in phi at the point (i, j) over one time step becomes

$$\Delta\Phi_{i,j} = \frac{\alpha \Delta t}{\Delta x^2}(\Phi_{i+1,j} + \Phi_{i-1,j} + \Phi_{i,j+1} + \Phi_{i,j-1} - 4\Phi_{i,j}). \tag{2}$$

A class which consists of iterators bumped to the correct location for all the terms in the equation including the left-hand side is constructed. This class has update method which evaluates the equation and a bump method which bumps all the iterators for the equation.

To create generic code for all dimensions, we create "walker" classes that use recursion and template specialization to walk the iterator collection objects through the grid. These classes are templated over dimension and they have an update method which recursively calls the update method for the lower dimension. We then specialize the first dimension to actually call the update method of the collection object.

An advantage of this methodology is that one has a single code base for all dimensionalities and all precisions. This reduces maintenance requirements. By using template specialization we can inline away all of the functions, so we do not have the normal loss of performance that accompanies recursion.

2 MULTIPLE MODELS

The typical plasma simulation code involves the self-consistent integration of a model for the charged particles along with a model for the electromagnetic field. Historically, codes have been written specifically to the model. One speaks of fluid codes, particle in cell codes, fully electromagnetic codes, codes that incorporate a reduced model for the electromagnetic fields, such as the electrostatic approximation, etc. VORPAL is designed to provide the capacity to incorporate different models for both the particles and the electromagnetic fields. The object oriented ideas of inheritance and polymorphism make this possible.

An example application is that of the electromagnetic field. We have a base class in VORPAL called VpEmField. This class defines the interface and hence all the behaviors one would expect from a electromagnetic field, the ability to give the electric or magnetic field at any grid point and the ability to update itself given a charge and current distribution. Any other component of the simulation that needs information from electromagnetic field, for example updating the particles, does not need to know the details of how the electromagnetic field is updated. The particles would communicate with the interface for the electromagnetic field defined in VpEmField, allowing us to put in different models for the electromagnetic field without having to change the code for particles.

At present we have implemented and tested a finite differencing method for the electromagnetic fields based on the Yee Mesh. The electric and magnetic fields are offset from the nodal point of a cell on a Yee Mesh. The electric

0-7803-7191-7/01/$10.00

field lives on the edges of the cells with the direction of the field corresponding to the vector running along the edge. The magnetic fields live on the faces of the cells, with the direction of the field corresponding to the perpendicular to the face. By arranging the fields in the cell in this manner, the finite differencing of Maxwell's equations is second order. The Yee Mesh solver has been implemented and tested in VORPAL.

A cold relativistic fluid model for the plasma particles has been implemented. Cold fluid means that we assume the pressure term does play an important role in the dynamics. The fluid density obeys the continuity equation:

$$\frac{\partial n}{\partial t} + \nabla \cdot n\mathbf{v} = 0. \tag{3}$$

The relativistic fluid momentum, $\mathbf{p}$ obeys the Lorentz force equation.

$$\frac{\partial \mathbf{p}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{p} = q(\mathbf{E} + \frac{\mathbf{v}}{c} \times \mathbf{B}). \tag{4}$$

The equation that governs the density is of a flux conservative form. The equation can be understood from a numerical point of view as the conservation of flux entering or leaving a cell. To update the density within a cell one just determines the fluxes passing through each cell boundary and then add or remove that from the density in the cell.

This gives us a conservative algorithm, however since we are interpolating the density to calculate the flux, there is a possibility of creating negative densities. This negative density can lead to an instablity. We are working on a method to correct the flux such that flow out of cell can not be greater than the density in cell for any one time step.

Combining the density equation with the momentum equation allows us to cast the momentum update in a flux conservative form as well. However, this does not allow for regions of zero density, as in conservative form one finds the local velocity by dividing out the density - a numerically ill defined procedure. The fluid momentum in a region of zero density still has meaning, it defines the trajectories of test particles within that region. Solving equation 4 directly by doing an advective update rather than a conservative one yields an algorithm that works when one has regions of zero density.

3 LOAD BALANCING AND DOMAIN DECOMPOSITION

There are two reasons for load balancing. The first is that different domains may have different amounts of computation. For example, in the propagation of a beam through an accelerating structure, the advance of the particles (the most computationally intensive part of the problem) takes place only where there are particles. The second reason is that different domains may be computed on processors of different capability. This is especially true of Beowolf clusters, as they become upgraded over time. If new processors are added to a Beowolf cluster over time, Moore's law tells that they will be able to carry out computations in less time.

The usual 2D domain decomposition of a 3D cubic region is shown in Fig. 1. Such a decomposition cannot achieve complete load balancing, as is needed is needed when there are a larger number of particles in one domain, as happens in beam simulations. The reason complete load balancing is not obtainable is that there are four domains, hence three conditions for equality of the computing time used by each domain. However, there are only two movable planes, thus insufficient freedom for satisfying three conditions.

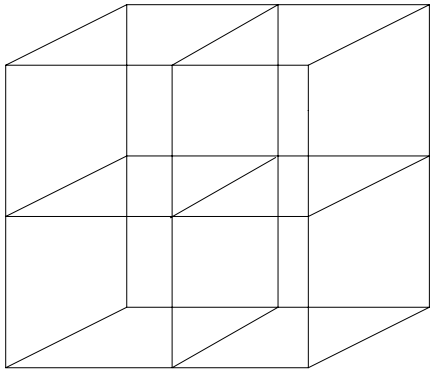

Figure 1: Standard 2D decomposition of a 3D cubic region

We do the domain decomposition as illustrated in Fig. 2. The decomposition planes perpendicular to one direction cut the entire region, but in the second direction they break at each of the planes of the first direction. Having three movable planes to satisfy three conditions makes load balancing possible.

The implementation of this decomposition is more difficult than that of the usual decomposition. We have prototyped a methodology for doing so. Using set theoretical ideas, we have defined the concept of a VpGridRgn, which is a logically cubical (bounded by six planes - orthogonal parallelepiped). Each domain is described by two VpGridRgn's, its physical region, physRgn, plus a layer of surrounding guard cells, xtndRgn. The physRgn is the region over which the domain must solve the dynamics. To do this it mush know the field at beginning of the time step in its xtndRgn. Thus the required communication is that each domain the values in its xtndRgn from the processors that have calculated those cells. We can see that all that needs to be done is to figure out the intersection of the domains xtndRgn with a neighboring domain's physRgn.

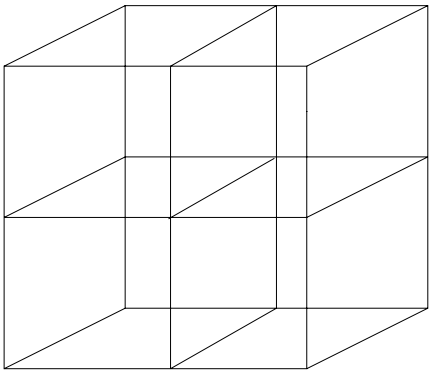

Figure 2: Fully general 2D decomposition of a 3D cubic region

This intersection is itself a VpGridRgn and it contains the cells which the neighboring domain needs to pass to the domain in question. In Fig. 3 the black rectangles are the physRgn's of the various domains and the red rectangle is the xtndRgn of domain 3. The blue rectangle is the intersection of domain 3's xtndRgn with domain 2's physRgn. This is the region that domain 2 must pass to domain 3.

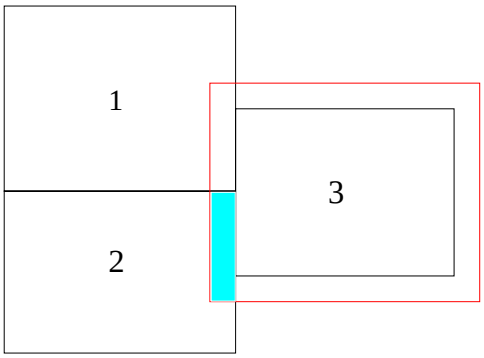

Figure 3: The intersection of an extened region with a neighboring domain's physical region

4 VISUALIZATION

The visualization in VORPAL is by post-processing the output files, which are written in the self-describing HDF5 format. The older HDF4 format can be read directly by Research Systems' Interactive Data Language and OpenDX, a open source visualization package based on IBM's Data Explorer. There is a command line utility that will convert HDF4 files into HDF5 which comes with HDF5. The next release of IDL will support HDF5 directly. Both IDL and OpenDX allow us to write visualization scripts and more elaborate GUI driven applications. We have found that OpenDX is easier to use since it has a very intuitive visual programming environment. In Fig. 4 we show the results of a 3D parallel simulation of an electromagnetic wave launched into a region of vacuum using VORPAL.

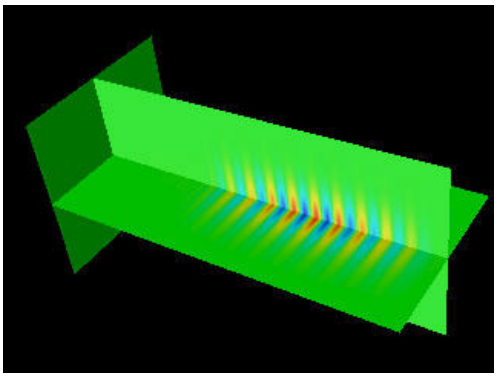

Figure 4: Transverse component of a electromagnetic wave in vacuum

5 CONCLUSIONS

We have produced a parallel plasma physics code using object oriented programming techniques and other advanced features of C++ that allows for greater flexibility compared to current codes. Multiple representations of the both the particles and electromagnetic fields are possible, and we have implemented a Yee mesh finite differencing for the electromagnetic fields and a fluid model for the particles which we will use to study problems in laser-plasma interactions. Our code runs in any dimension and with selectable precision without loss of performance. We have the ability to support load balancing using a set-theory based message passing.