PIC Code with Adaptive Grid

Scott Luedtke, Max Porter, Joel Iventosch, Mark Sholte

March 3, 2015

1 Introduction

Particle in Cell (PIC) codes are commonly used to simulate plasma physics. Existing codes, such as EPOCH [1], allow physicists to simulate many experiments, including laser experiments. Larger problems, such as 3D simulations of high-intensity laser experiments like those performed on the Texas Petawatt Laser (TPW), are too computationally expensive to run on modern supercomputers. One way to bring these problems down to size is to adaptively change the grid size. Simulations can require a very high resolution at certain sites (for example, where a laser hits a target), and orders of magnitude coarser resolution throughout the rest of the simulation.

Our goal is not to accurately simulate physics, but rather to develop part of a PIC scheme that will allow very large problems to be run on modern supercomputers. To do this, we propose to build a 2D PIC code with a standard Boris particle pusher [2], implement data passing routines among nodes, develop an adaptive grid, and load balance the work at each time step based on the changing grid. The size of the full simulations will make much of this difficult. The grid will have to adapt on a node by node basis based on its neighbors only, without knowledge of the entire grid. We expect this work to fully occupy all group members. If some parts prove easier than we expect, the project can be expanded by implementing an adaptive time step for certain particles [3], or by checking the physical results of our code.

2 Problem Description

We consider a 2D region of space divided into a rectangular grid. At each grid point we store the electric and magnetic fields. Traveling across this grid (non-discretely) are many particles. These particles both influence and are influenced by the field values at their nearby grid points.

The central problem of our project is how to apply an adaptive grid algorithm that will define the field values with constant accuracy, but not constant resolution. This can be accomplished through an adaptively changing grid discretization which determines at each time step whether it should zoom in, i.e., split a single grid division into four (or more) smaller grid divisions, possibly splitting repeatedly several levels down to wind up with orders of magnitude more grid divisions in place of the previous single division, or zoom out, i.e., combine previously split divisions which no longer require their high resolution.

The parallelization problem contained within this is determining how to efficiently assign processors to calculating particles and fields in a way that minimizes the communication time between processors and the idling time due to some processors being given larger tasks than others. Since two areas of the same size will not always have similar numbers of particles in them as the particles move around, it must be determined how much area each processor is responsible for in order to give each processor roughly the same number of particles to update (particle updates being more time-intensive than field updates). We will divide up tasks by physical areas containing particles. It will be hard to achieve optimal load balancing because of non-uniform constantly changing grid resolution. As areas become more or less fine and particles move to different cells, the decision of which processor to assign to a given area must be constantly re-evaluated.

3 Proposed Parallelization

We will use MPI for communication between nodes on Lonestar and OMP for communication within each node. At each time step in the simulation we will have two dependent updates to our model: the particle push, and the field update. Our algorithm will parallelize these computations by dividing the cells among the processors in contiguous blocks each with a similar number of total particles and having each processor update only the particles and fields inside its block. Care will be taken to adjust the cell distribution in an attempt to keep the particles divided evenly among the processors.

On each node, the particle push is trivially parallelized with OMP. The field update for grid points within a node's area is similarly parallelizable. The update for boundary grid points will require MPI communication with neighboring nodes. The complexity of our proposed algorithm arises from the communication scheme that we will have to implement in between time steps in order to enable continued efficient parallelization at the next time step. Due to the nature of our problem, particles will travel across the grid and there will be instances where a disproportionately large fraction of the particles is located in a block being updated by a single processor. Such a load imbalance will leave many processors idle while waiting for the overworked processor to finish the time step iteration. Therefore, after each time step we will reassign cells from one processor to another in an effort to keep the workload balanced. The size of full simulations forbids this load-balancing process to be done by a control node. Instead, it will be done on each node, communicating with its neighbors only. Implementing the load balancing is the primary challenge of this project.

4 Problem Size and Resources Needed

The size of the full 3D simulations is enormous. They will need at least $(10^3)^3$ base grid points and at least 10 times as many particles. That size forbids any central control and restricts the design possibilities for load balancing. To develop the adaptive grid for this project, we propose a much smaller 2D problem.

We expect approximately 20 million particles keeping track of momentum, position, and a few other variables for a total of about 1 GB of memory. We will likely use a 100x100 basis grid that would adapt to not more than 10 million points. Keeping track of E and B fields, this will require less than 1 GB.

Together, the memory requirements for the grid and particles is about 2 GB. In addition, each processor needs to keep track of which grid points it is responsible for as well as knowing which processors are responsible for neighboring grids (a stencil over the grid needs to be applied at each time step to update the E and B fields). However, this only requires roughly an additional int (4 bytes) per grid point for a negligible amount of extra memory.

Assuming there are n particles and m grid points both divided fairly evenly among the p processors with latency l and bandwidth b, then the computational complexity at each time step is O(n/p) to update the particles positions plus O(m/p) + $O(l+b^*(m/p)^{0.5})$ to update the fields at the grid points. Thus, the expected computational complexity is O((n+m)/p). In general, if the particles and grid points are significantly unbalance among the processors, then this complexity is increased. In the worst case all particles in the simulation are in 1 cell and thus 1 processor is responsible for updating all of them, thereby increasing the time to O(n+m/p).

[1] The EPOCH code was developed as part of the UK EPSRC funded projects EP/G054940/1

[2] Birdsall, C.K. and Langdon, A.B. (1985). *Plasma Physics via Computer Simulation.* McGraw-Hill.

[3] Arefiev, Alexey V., et al. "Temporal resolution criterion for correctly simulating relativistic electron motion in a high-intensity laser field." Phys. Plasmas 22, 013103 (2015)