DynoGrid

Dynamically Load-Balanced PIC Code with an Adaptive Grid

Scott Luedtke, Max Porter, Joel Iventosch, Mark Sholte

May 9, 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.

We have developed a 3D PIC code with a standard Boris particle pusher [2], added an adaptive grid, implemented data passing routines among nodes, and load balanced the work based on the changing grid. Our code is not physically accurate, ommiting a field solver and ignoring many physically relevant effects, but we believe it is an accurate representation of a production code in performance characteristics and expect our load balancing to behave similarly on a production code.

2 Code Description

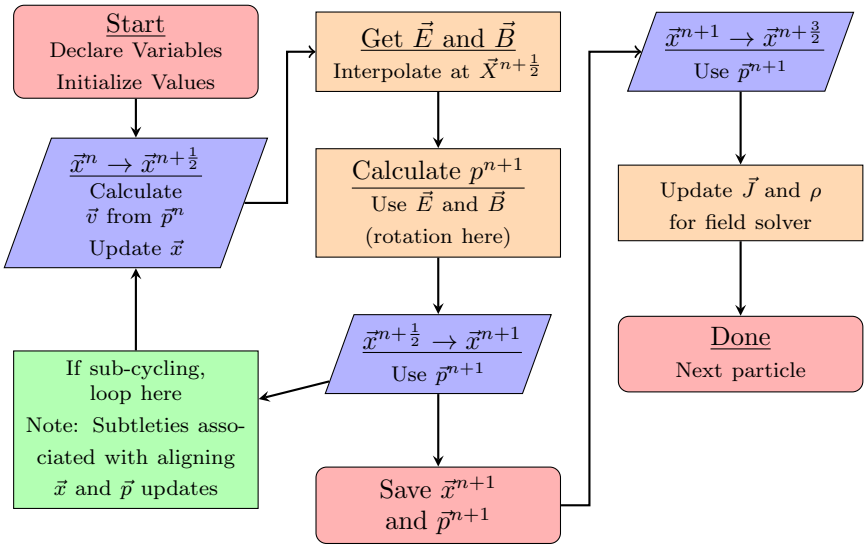
PIC codes are commonly used in computational physics. A PIC code typically consists of an array of grid points that define electromagnetic field values, and particles that move continuously throughout the grid. We created a fairly simple 3D laser-plasma code with externally defined fields (the laser), and physical effects such as particle collision and photon pair production omitted. We did however add an adaptive grid which makes load balancing more complex

2.1 Particle Pusher

Particle-in-cell (PIC) codes used to simulate laser-plasma interactions commonly use the Boris pusher [2], a second order accurate pusher for charged particles in electromagnetic fields. The pusher is outlined in Fig. 1. Essentially, the particle is advanced a half time-step, the momentum is updated with $\vec{E}$ and rotated with $\vec{B}$, and the particle is moved a final half time-step. With a properly set time step, a particle will never move more than one cell.

Our pusher follows the implementation in EPOCH [1], but with some modifications. We use our own linear interpolation method (for simplicity), and omit the steps necessary for the field solver, which we did not implement. We have different logic for finding the field points nearest a particle, and recursive logic to find the finest grid cell a particle is in. In addition, we have a particle list for each base cell, and implemented logic to pass particles between lists (this makes the load balancing easier).

Figure 1: Schematic of the Boris pusher. The sub-cycling and current and density calculations were not implemented.



2.2 Particle Lists

2.3 Grid Trees

2.4 Grid Refinement

3 Parallelization

3.1 List Passer

3.2 Cell Passer

3.3 Load Balancer

4 Theoretical Performance

5 Performance Results

6 Discussion

7 Whodunit

Scott Luedtke was primarily responsible for the particle pusher and load balancer. Since this project is closely related to Scott's research, he oversaw the physics and provided general guidance for what the code should accomplish.

Max Porter worked on the grid refinement and stuff.

Joel Iventosch wrote the output and visualization routines and helped with MPI passing. He also helped with the grid refinement.

Mark Sholte wrote much of the list functions and tree structure. He also managed any git repository issues that arose and proved a masterful debugger.

8 Conclusions

[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.