Parallelization of a Particle-in-Cell Algorithm

Kevin Howarth, Hari Raval, Aditi Memani, Taro Spirig

February 22, 2022

Application overview

The Particle-in-Cell (PIC) algorithm was first introduced in 1955 as a method for simulating fluid mechanics. In the following years, it became popular as a method for simulating the time-evolution of a plasma - a superheated gas whose atoms have split into positively charged ions and negatively charged electrons. The trajectories of these charged particles are affected by Coulombic interactions between the particles as well as externally imposed electromagnetic fields. One could naively attempt to simulate charged particle trajectories as an N-body problem, since Coulomb's law allows for exact calculation of the net force on each charged particle. However, the $\mathcal{O}(N^2)$ scaling of the N-body algorithm is incredibly restrictive. The particle density in interesting plasmas can easily exceed 10^{18} particles per cubic meter, so another algorithm is needed.

The PIC algorithm circumvents the $\mathcal{O}(N^2)$ scaling by solving the Poisson equation on a computational grid to determine the electric potential and force fields arising from a given particle arrangement. Interpolation is used both to assign particle charges to grid points and to transfer electric field strengths back to particle locations, allowing the particles to be time-stepped from their exact positions by means of a numerical ODE solver. Since these operations only do work once on each particle, the PIC algorithm's runtime scaling is $\mathcal{O}(N)$, making it a much more attractive option than a direct N-body algorithm.

Parallelization Goals

There are many opportunities for parallelizing the PIC algorithm to decrease runtime even further. Discretization of Poisson's equation via Finite Differencing leads to a system of linear equations. Direct solution of this system requires matrix inversion, which is $\mathcal{O}((N_g^d)^3)$, where N_g is the dimension of the computational grid and d is the number of spatial dimensions in the simulation. However, iterative methods for solving matrix equations (such as Jacobi, Gauss-Seidel, and SOR) can converge to a highly accurate solution at much better runtimes. These iterative methods lend themselves well to parallelization. One can also parallelize over the particle-to-mesh/mesh-to-particle interpolations as well as the time-stepping of particle locations. This increases efficiency when simulating with large numbers of particles.

Our team's goal is to explore these methods of parallelization. Two of the authors have already written a PIC code in Python as a previous course project (visible at https://github.com/KHowarth422/TinyPIC), so the aim of this semester's work will be to develop and validate parallelized code in C/C++ for performing individual steps of the PIC algorithm. An example of a possible final deliverable is a runtime comparison between the original Python code executed on a single laptop and the parallelized C/C++ code on the CS205 academic cluster.