AA545: Computational Methods for Plasmas
Computer Project 1.2
High-Performance 1D1V Electrostatic PIC

Matt Russell
Department of Aeronautics & Astronautics
University of Washington
(Dated: May 14, 2023)

I. Introduction

The complexity of modern plasma experiments, arising from the very large number of plasma particles, the presence of electromagnetic fields, and the multiscale nature of the dynamics, means that computer simulation is required in order for the physical system to be understood.

At a microscopic level, the exact description of an N-body system is intractable. There are simply too many interacting modes at play. It is also impossible to simulate this system directly. Leaving aside the electromagnetic field for the moment, the complete simulation of a mol of hydrogen plasma would require O(Y) (Y = Yottabyte) memory just to store the phase-space information, and even with cutting edge processing power, the simple act of just updating these values each timestep alone would take $O(10^{15})$ seconds, which is around $100,000,000$ years! Of course, these values are for a single CPU, but even with the most powerful GPU currently available on the market, the NVIDIA RTX 4090, boasting 16,384 CUDA cores, a pricetag of $ 1600 MSRP, and running code year-round that fully utilizes its capabilities (the most unbelievable part of this thought experiment), it would still take $\sim$ 6000 *years* just to update the positions and velocities of all the particles.

It must also be noted that this number is very much an underestimate. Underutilization of resources (an inevitabililty), weighting the charge, computing the fields, pushing the particles, and writing the data out have all been neglected in coming up with it. In reality, we are talking closer to an order of magnitude, or two, above this period. Barring a lengthy rerun of Moore's Law 2: GPU Boogaloo, this timescale is simply absurd to do more than entertain curiously. Furthermore, it is vain to think that a GPU with a ridiculous number of compute cores wouldn't also require a ridiculous amount of power, and generate a ridiculous amount of waste heat. Even if the number of compute cores on cutting-edge GPU hardware were to begin doubling every 18 months, it is likely these considerations would preclude a sufficient number of blades from being assembled to make such a problem manageable.

Therefore, even to just arrive at the most basic, "close-to-the-ground-truth", level that computational plasma physics has, we must average over these modes in order to obtain a reduced 6D+time phase-space description of the plasma. Then, if we draw a sample of "superparticles" from this distribution, each representing a large number of physical particles, and place them onto a grid, bestow them with a shape, accumulate their charge, and evolve their motion in a set of electromagnetic fields, we can obtain a glimpse into the plasma physics of interest. Simulating the kinetics of a plasma in this way is known as the Particle-in-Cell (PIC) method.

II. Background

The PIC algorithm is described below in Algorithm 1. Each of the simulation particles is referred to as a superparticle, that being its nature as a collection of physical particles. Over the course of a timestep, this phase fluid is evolved under the action of a set of electromagnetic fields.

Algorithm 1 The PIC Algorithm

```
Initialize superparticles, and grid
Particle Weighting
Field Solve
Force Weighting
Particle Push
goto Particle Weighting
```

This loop continues until a sufficient time has been reached that the simulation completes. Extra steps for initializing and writing to datafiles can be inserted at their appropriate places in the loop.

A. Initialization

The population is first initialized. N representative superparticles are created and their initial positions and velocities specified, $(\vec{x}_i, \vec{v}_i)$. Depending on the physics of interest, a choice is made here regarding the dimensionality of the phasespace. The project described here implements a 1D1V phasespace, the simplest kind of configuration. A mesh must additionally be initialized in order to store the structure of the electromagnetic field, and the particle locations. Furthermore, the PIC method requires that this mesh be a uniform, Cartesian grid. A nonuniform mesh will cause problems during the weighting steps because the process by which charge is accumulated to produce fields and forces assumes a regular spacing to the mesh. If the spacing were not uniform, then the charge density that each particle has to offer, as well as the impact it has, would vary based on where it is on the mesh, and this would violate their indistinguishable, atomic nature, producing inconsistent charges and currents, and rendering the simulation incoherent.

B. Particle Weighting

Given the state of each superparticle, their charge is then accumulated on the grid depending on the shape that the particles have. The simplest, sensible choice of shape that can be made is to treat the particles as a uniform charge cloud with a spatial expanse of a_0, and uniform height, $\frac{1}{a_0}$. This amplitude is specified by the requirement that charge be conserved. In general, every timestep, the particles are first located on the mesh. Then, based on their location, their charge is assigned appropriately to the gridpoints. To avoid disappearing charge, or double-counted charge, the particle expanse must be the same as the grid spacing, $a_0 = \Delta x$.

The simplest way of apportioning the charge is to find which gridpoint is nearest the particle's location, x_i, and place all the charge there. However, this produces noisy charge distributions as particles will suddenly 'leap' from one point to the other. To address this, we can refine the particle shape and consider a piecewise function in the shape of a triangle,

$$S_1(x) = \begin{cases} h_s(x + a_0), 0 < x < a_0 \\ h_s(x - a_0), -a_0 < x < 0 \end{cases}$$

This shape is not chosen arbitrarily. Higher-order shapes than 1st can be computed from convolutions of the 0th order shape. The height, $h_s = \frac{1}{3\Delta x^2}$, can be computed from normalizing the shape function to 1.

After the cell the particle is located on is found, the charge in a 1st-order weighting method is apportioned to the neighboring grid points using inverse distances, where,

$$\rho_{c,j} = q_i \frac{a}{\Delta x} \tag{1}$$

$$\rho_{c,j+1} = q_i \frac{b}{\Delta x} \tag{2}$$

$$a = |x_{j+1} - x_i| \tag{3}$$

$$b = |x_i - x_j| \tag{4}$$

For an electrostatic simulation, this is all that is required. More complicated programs involving magnetic fields will require that the current be weighted as well in order to produce a current density that supports the magnetic field.

C. Field Solve

The purpose of the previous step is to obtain values for the charge and current density so that, in general, Maxwell's equations can be solved to obtain the electric and magnetic fields on the grid. In an electrostatic, 1D1V simulation this reduces to the task of solving Poisson's equation for the electric potential,

$$\frac{\mathrm{d}^2\phi}{\mathrm{d}x^2} = -\frac{\rho_c}{\epsilon_0} \tag{5}$$

to achieve this, the second derivative is discretized using a finite (central) difference, and the resulting system expressed according to,

$$\frac{1}{\Delta x^2}\left(\phi_{j+1} - 2\phi_j + \phi_{j+1}\right) = \frac{\rho_j}{\epsilon_0} \tag{6}$$

To solve this, the second derivative is expressed as a discrete operator containing the system of equations described by the above, and then solved in a suitable manner.

D. Force Weighting

After the particles have been accumulated on the grid, and the electromagnetic fields obtained, their impact on the particles must be computed. This is done by weighting the fields on the grid back to the particles, similar to how the charge and current density of the particles was originally weighted to the grid.

1. 0^{th}-order Weighting

At the simplest level, the nearest grid point (NGP) method selects the closest grid point to the given particle, and assigns its electric field to the particle.

2. 1^{st}-order Weighting

One level up is to use the method of inverse distances again. Inverse distances are important as computations using them result in the conservation of densities, e.g, charge or momentum.

$$E_i = q_i \frac{a}{\Delta x} E_j + q_i \frac{b}{\Delta x} E_{j+1} \tag{7}$$

the above expresses the force on the particle, i, that is located in cell j.

E. Particle Push

After the forces being felt by the particles have been computed, the particle motion must be advanced. In order to obtain second-order accuracy a Leapfrog method is employed.

1. Leapfrog Method

The Leapfrog method advances the position and velocity of a particle subject to an external force by calculating the velocity at 'half-steps', and then using this to advance

the position,

$$v_i^{n+1/2} = v_i^{n-1/2} + \Delta t * E_i \tag{8}$$

$$x_i^{n+1} = x_i^n + \Delta t * v_i^{n+1/2} \tag{9}$$

while being very accurate, the Leapfrog method also suffers from an instability that leads to phase-error if the condition $\omega \Delta t << 2$ is not satisfied.

III. Implementation

The PIC algorithm is written in C++, and the data is analyzed and graphed with Python. The code uses the Eigen library to provide a SparseMatrix class for the field solve step, and a native implementation of binary search to find the particles in $O(log(N_x))$ time. These decisions are made in light of performance considerations as they reduce the computational time to solve the fields and weight the particles from $\mathcal{O}(N_x^2)$, and $\mathcal{O}(N * N_x)$ to $\mathcal{O}(nnz)$, and $\mathcal{O}(N * log(N_x))$, respectively. This may not seem significant for $N_x = 33$ gridpoints, but even here the computational savings is at least an order of magnitude, and will only grow as the number of gridpoints increases.

To handle the boundaries, periodic boundary conditions (PBC) are implemented. A very puzzling bug was struggled with during the development whereby an issue with the PBCs was observed in the particle trajectories despite grid data showing their satisfaction. The solution was to eliminate the right boundary from the system of equations, and hard code the periodic boundary condition for the potential as well as charge density. The gauge, $\sum_j \phi_j = 0$, remained in the last row of the sparse matrix, and the periodicity, $\phi_0 = \phi_{N_x - 1}$ was expressed after the solution of the fields.

A. Normalization

The plasma frequency is,

$$\omega_p = \sqrt{\frac{ne^2}{m\epsilon_0}} \tag{10}$$

It is not desirable for a numerical simulation to work with very large numbers as repeated computation with such a value amplifies roundoff error. Typically, they are signs of instability. Small numbers are preferred as they reduce the magnitude of roundoff error. To remain coherent, numerical simulations are done using dimensionless units, here these are,

$$\tilde{\omega}_p = \sqrt{\frac{N}{L}} \tag{11}$$

$$\epsilon_0 = 1.0 \tag{12}$$

$$q_i = -1.0 \tag{13}$$

IV. Results

The first task of the assignment was to initialize two particles at rest on a grid spanning $[-\pi, \pi]$, with 33 gridpoints. The particle states are initially specified as,

$$(x, v)_0 = (-\frac{\pi}{4}, 0) \tag{14}$$

$$(x, v)_1 = (\frac{\pi}{4}, 0) \tag{15}$$

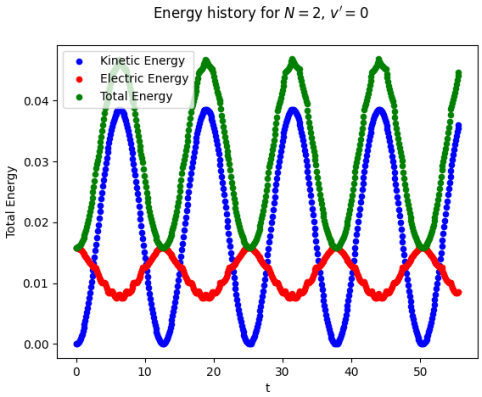

FIG. 1. Energy history for Task 1. The plasma particles are subject to first-order weighting, and the system energy completes approximately 4.5 cycles in a time of 55.6 to yield $\omega_{p,comp.} \approx .508$. The theoretically expected frequency is $\omega_{p,theo.} = .564$.

The second task of the assignment was to initialize two particles with an initial velocity,

$$(x, v)_0 = (-\frac{\pi}{2}, v') \tag{16}$$

$$(x, v)_1 = (\frac{\pi}{2}, -v') \tag{17}$$

$$v' = 0.1 \tag{18}$$

As the magnitude of the perturbation and the energy

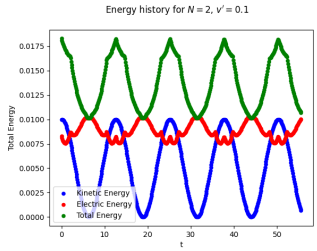

FIG. 2. Energy history for two superparticles initialized with a velocity perturbation. The plasma energy here also oscillates with a period of approximately 4.5 cycles per 55.6 units of time, the difference between this and the system in Figure 1 being just a relative change in the phase and scale of the energy due to the velocity perturbation.

of the system increases, the amplitude of the oscillator's motion also increases. Eventually, the amplitude will be large enough that the particle trajectories cross.

$\omega\Delta t$	ω	$\omega_{meas.}$
0.06283	3.1915	3.1979
0.6283	3.1915	2.8781
1.2566	3.1915	2.8781
3.1416	3.1915	1.1512

TABLE I. Tabulation of the theoretical and measured plasma frequency given $\omega\Delta t$ for the simulation.

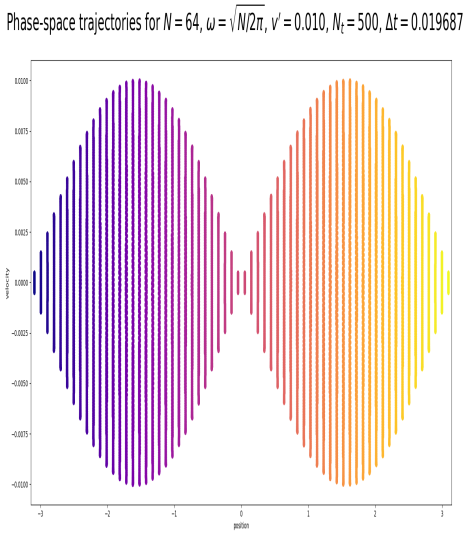

FIG. 3. Phasespace for $N = 64$ particles that were initialized with a sinusoidal velocity perturbation $v' = 0.01 * sin(x)$. The oscillating trajectories of the particles as the motion evolves gives the graph a bowtie character as the initial perturbation modulates the state of the system.

The final task of the assignment is to initialize $N = 64$ particles with a sinusoidal velocity perturbation that supports a single spatial period and study its energy over a number of cycles in order to measure the plasma frequency and observe as the Leapfrog instability introduces phase error into the simulation when $\omega_p\Delta t << 2$ is violated. The computed values are tabulated in Table

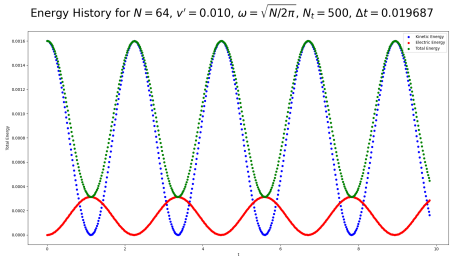

FIG. 4. Energy history for $\omega\Delta t = 0.06283$

I. They were obtained by running simulations for four different values of the characteristic angle, $\omega\Delta t$ to obtain energy historys. The average of this data was then computed, and used to shift the dataset downwards so that the number of zero-crossings in the total energy could be used to measure the frequency, and the phase error studied as $\omega\Delta t << 2$ is violated.

From the presented data, it is apparent that error in the frequency of the simulation is introduced as the characteristic angle, $\omega\Delta t$ reaches the scale of 2, and beyond.