# CS 267 Final Project Proposal: High-Order Hybrid Particle-in-Cell Method

Yanhe Huang, Zixi Hu, Colin Wahl

## I. PROBLEM

We will solve the Vlasov-Poisson system using a Particle-in-Cell (PIC) method. The system can be written as

$$\frac{\partial f}{\partial t} + \boldsymbol{v} \cdot \frac{\partial f}{\partial \boldsymbol{x}} - \boldsymbol{E} \cdot \frac{\partial f}{\partial \boldsymbol{v}} = 0$$

and

$$\nabla^2 \phi = -\rho$$

with $\mathbf{f}(\boldsymbol{x}, \boldsymbol{v}, t)$ is the phase-space distribution, $\boldsymbol{E}(\boldsymbol{x}, t)$ is the electric field satisfying $\boldsymbol{E} = -\nabla\phi(\boldsymbol{x}, t)$, $\phi(\boldsymbol{x}, t)$ is the potential, and $\rho(\boldsymbol{x}, t)$ is the charge density. Under the assumptions that our distribution contains negatively charged particles and the ions form a fixed, neutralizing background field, the charge density and phase-space distribution can be related by

$$\rho(\boldsymbol{x}, t) = 1 - \int f(\boldsymbol{x}, \boldsymbol{v}, t) d\boldsymbol{v}.$$

By representing the distribution as a set of Lagrangian particles with charge

$$q_p = f(\boldsymbol{x}_p^i, \boldsymbol{v}_p^i, t = 0) h_x^D h_v^D$$

we can express the PDE above as a set of ODEs. Four steps are required to advance the Lagrangian particles: deposit charge, solve Poisson's equation, interpolate the fields, and integrate the equations of motion. We will implement this method with high-order kernels in a way that meets the requirements of existing convergence theory. Specifically, we will respect requirements on the number of particles per cell that are often ignored by state of the art PIC codes. If time permits, we will also implement phase-space remapping - which was shown in [4] to be important for long time convergence.

## II. IMPLEMENTATION

We propose a hybrid MPI-OpenMP implementation and plan to investigate the performance of the high-order hybrid PIC method. To prescribe work to different nodes, we will decompose the physical domain. Previous work has implemented a physical space domain decomposition and found this to have good weak scaling results [4]. We plan to implement a Z-order domain decomposition but hope to investigate other domain decompositions strategies if time permits. The OpenMP threads will then divide the work up in the phase-space slab of the physical space region associated with the node. Since all of the steps including

particles will be implemented by looping over the particles, we will allow each thread to work on a contiguous chunk of particles. Previous work has focused solely on the particle-mesh steps (the deposition and interpolation) for shared-memory architectures [2] and GPUs [1, 3] since these steps often are the most difficult to implement since they involve nonuniform memory access. To avoid race conditions between different threads we plan to give each a local copy of the mesh and then aggregate the results once each is done. Since the mesh is only two dimensional, it is relatively inexpensive for each thread to have a portion of this mesh. Once the deposition step has been completed, we will implement a 4th-order finite difference scheme with ghost regions to solve for the potential. We plan to use a finite difference method since it is simple, local, and generally this stage accounts for $\sim 5\%$ of the total run time. Once each node has solved for the region in space necessary for it to do the force interpolation, it will loop over the particles and interpolate the forces to each particle. This will again be a threaded operation; however, unlike the deposition stage, race conditions will not be an issue since each particle is only owned by a single thread. Finally, we will advance the particles in time with an explicit 4th-order Runge-Kutta method. This will not require communication beforehand; however, if a particle leaves a processor's region of control, we will need to communicate this before moving on. Since the particles will be stored in `std::vectors` we will initially allocate more space than needed and each particle will have a flag determining whether it exists. Each node will add particles to the end of the list and change the flag on particles that leave its control. Periodically, the `std::vector` will be copied and all the flagged particles will be deleted.

To test our implementation, we will examine the weak scaling performance for linear Landau damping - a standard test problem in this field. We also hope to test the performance on a more difficult problem such as the two-stream instability. For linear Linear landau damping, the distribution of particles will stay uniform through out the simulation. The two-stream instability may provide insight into drawbacks of our domain decomposition strategy.

Finally, we hope to perform some form of remapping although implementing this in an efficient way is still an active area of research. We are currently investigating using local remapping done adaptively in time; however, implementing such a strategy in this project should be considered a reach goal that, if not completed, will be investigated during the summer.

---


[1] Ferit Büyükkeçeci, Omar Awile, and Ivo F Sbalzarini. A portable opencl implementation of generic particle–mesh and mesh–particle interpolation in 2d and 3d. *Parallel Computing*, 39(2):94–111, 2013.

[2] Kamesh Madduri, Jimmy Su, Samuel Williams, Leonid Oliker, Stéphane Ethier, and Katherine Yelick. Optimization of parallel particle-to-grid interpolation on leading multicore platforms. *IEEE Transactions on Parallel and Distributed Systems*, 23(10):1915–1922, 2012.

[3] George Stantchev, William Dorland, and Nail Gumerov. Fast parallel particle-to-grid interpolation for plasma pic simulations on the gpu. *Journal of Parallel and Distributed Computing*, 68(10):1339–1349, 2008.

[4] Bei Wang, Gregory H Miller, and Phillip Colella. A particle-in-cell method with adaptive phase-space remapping for kinetic plasmas. *SIAM Journal on Scientific Computing*, 33(6):3509–3537, 2011.