Performance-portable GPU acceleration of the hybrid PIC code `dHybridR`

Bricker Ostler*, Miha Cernetic[†‡], and Damiano Caprioli[†§]
* Department of Physics, The University of Chicago, Chicago, IL 60637, USA
bostler@uchicago.edu
† Department of Astronomy and Astrophysics, The University of Chicago, Chicago, IL 60637, USA
‡ Cerebras Systems, Sunnyvale, CA 94085, USA
§ Enrico Fermi Institute, The University of Chicago, Chicago, IL 60637, USA

Many astrophysical plasmas are effectively collisionless: particles evolve according to self-consistent collective electromagnetic fields and wave-particle interactions rather than binary Coulomb collisions. Such physical systems are best modeled by the Vlasov–Maxwell system of equations. However, Eulerian Vlasov methods become particularly costly for distributions with extended power-law tails requiring large momentum-space domains. Instead, one often employs the so-called particle-in-cell (PIC) method, which samples the distribution function using a set of "macroparticles". For many problems in high-energy astrophysics, space physics, and laboratory astrophysics, only the ions need to be described kinetically; electrons can instead be approximated as an inertialess, charge-neutralizing fluid. This is the so-called hybrid PIC method that our relativistic MPI+Fortran code, `dHybridR`, employs.

Before the GPU port, we refactored the CPU-only code to take advantage of cache locality and compiler auto-vectorization. The species property arrays were split into per-property arrays and the electromagnetic field arrays reorganized so that consecutive iterations of a loop over macroparticles or grid cells read adjacent elements in memory, making efficient use of the cache hierarchy and allowing the compiler to auto-vectorize such loops. Additionally, macroparticles are sorted by cell every timestep so that the gather, push, and deposit routines loop over grid cells and, within each cell, over its macroparticles.

The theme of these optimizations was directly translatable to the subsequent GPU port: individual particle arrays (position, velocity, ...) allow work-items in a subgroup to make coalesced device-memory accesses, while pre-sorting these arrays and looping in a cell-centric way allow a one-to-one correspondence between subgroups and cells, where each subgroup can efficiently operate using subgroup reduction operations.

We aimed to minimize changes to the existing CPU-only codebase and avoid tying GPU support to a particular vendor. We therefore chose OpenMP target offloading for broad codebase coverage with specialized SYCL kernels written for the most intensive subroutines. Fortran `bind(C)` interfaces then connect these kernels to the Fortran codebase.

We first offloaded the time-stepping loop, including the field solve, boundary conditions, moment deposition, and field and particle packing. After code correctness was verified, we moved the particle and field data on-device and systematically eliminated all routine host–device synchronizations. The only functionalities not offloaded are either minimal in compute cost (particle injection), or amortized across many timesteps (diagnostic output and load balancing).

Subsequent profiling revealed ample room for optimization, particularly in the charge / current deposition kernel due to atomic contention. Since the code uses explicit timestepping, all particles are subject to the speed-of-light CFL condition: any particle in the simulation may move at most one cell per timestep, even if it moves at the speed of light. Because typical ion speeds are much smaller than c, we find that most particles remain in the same cell over a single timestep. Our charge / current deposition kernel therefore accumulates non-crossing particles in private register-resident stencils, reduces them across the subgroup, and assigns one work-item to update the global grid. Cell crossers instead deposit their shifted stencils directly into the global grid with atomics, avoiding the increased register pressure and reduced occupancy of a larger stencil for all particles. This improved routine requires $4\times3^D(1+N_{\rm cross})$ global atomic additions per cell and species compared with $4\times3^D N_{\rm ppc}$ in the naive implementation, for spatial dimension D and $N_{\rm cross}\ll N_{\rm ppc}$ crossing particles.

The only section requiring a complete refactor was inter-rank particle communication at the end of each time step. The original code serially classified and packed outgoing particles, appended incoming particles, and compacted the particle arrays. These operations now execute via multiple data-parallel passes on the device.

Thermal-plasma benchmarks in 2D and 3D demonstrate weak scaling efficiencies of 86–97% reaching 49,152 accelerators on each of the exascale systems Aurora (Intel) and Frontier (AMD) and 1,536 GPUs on Polaris (NVIDIA). At 256 particles per cell (ppc), full-node GPU execution is 14–32× faster than the refactored CPU implementation and uses approximately 90% less energy per particle-update on Aurora and Frontier. To our knowledge, no other hybrid PIC code has reported GPU performance at this scale.

To demonstrate this capability, we performed twin 2D and 3D simulations of a benchmark parallel collisionless shock.

Even at this early stage, the 2D run holds about 50% of the magnetic energy in the out-of-plane component whereas the 3D run is near equipartition. The 3D simulation took 15.6 h on 512 Frontier nodes; on the same nodes' CPUs it would require an order of magnitude more node-hours, and 40–$50\times$ more with the original implementation.

These advances substantially lower the computational barrier to large-scale 3D hybrid-kinetic simulations of collisionless plasmas, and our approach requires only OpenMP offload, SYCL, and their interoperability, presenting a practical GPU-porting strategy for existing MPI-parallelized Fortran plasma codes.

ACKNOWLEDGMENT

OpenAI Codex and Anthropic Claude assisted with organization and language editing throughout this summary.