GPU-MPI Parallelization for QuickPIC, an Algorithm for Simulating Plasma Wake Field Acceleration

Yueran Tian [1] Yueluo Wang [1,2] Thamine Dalichaouch [3] Viktor Decyk [3] Warren Mori [3]
Weiming An [1,4]

[1]School of Physics and Astronomy, Beijing Normal University [2]Department of Astronomy, Peking University [3]Department of Physics and Astronomy, University of California, Los Angeles [4]Institute for Frontiers in Astronomy and Astrophysics, Beijing Normal University

Introduction

Plasma wakefield acceleration (PWFA) uses high-energy particle beams to generate oscillations in plasma, which in turn creates longitudinal electromagnetic fields that accelerate charged particles.
QuickPIC is a particle-in-cell program that employs quasi-static approximation to effectively simulate PWFA.

- **Quasi-static approximation:** since particle beams evolve at a much longer time scale than the plasma wavelength, QuickPIC separate the beam and plasma evolution time scales, and "pauses" the beam evolution while calculating multiple steps of plasma response.

QuickPIC supports parallel simulation on multiple CPUs via OpenMP and Message Passing Interface (MPI). We have now ported QuickPIC onto GPU platforms in a new version named QuickPIC-GPU, to support simulation with one or more GPUs.

Methods

QuickPIC separates beam and plasma computation into 3D (outer) and 2D (inner) loops. The 2D loops that calculate plasma response dominate computational load. A 2D loop is mainly made up of functions from two Fortran 2003 top layer classes: field2d (electromagnetic fields) and species2d (particles). These in turn rely on mid layer classes: fft2d (fast Fourier transforms), fpois2d (Poisson equation solver), ufield2d (field storage and arithmetic operations), part2d (particle related computations) and fdist2d (particle array initialization).

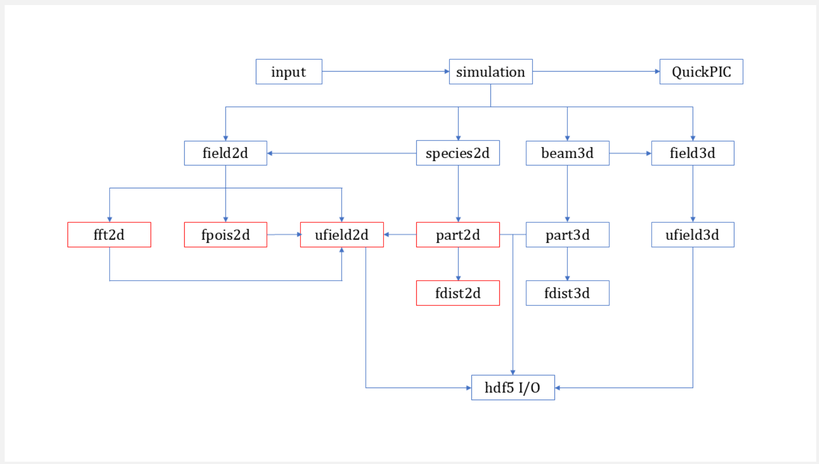

Figure 1. Class diagram of QuickPIC. Classes outlined in red are the main GPU revision targets.

In porting QuickPIC to GPU platforms, the Fortran77 functions called in these mid layer classes are replaced with CUDA C functions. In the original functions, the simulation box is divided equally among MPI nodes, and the particles (grouped by larger subregions called tiles)/discretized fields in each node are distributed to the CPU cores in the node via OpenMP. In the new CUDA C functions, each MPI node drives one GPU, with tiles (if needed) assigned to CUDA blocks and particles/grid points distributed across threads.

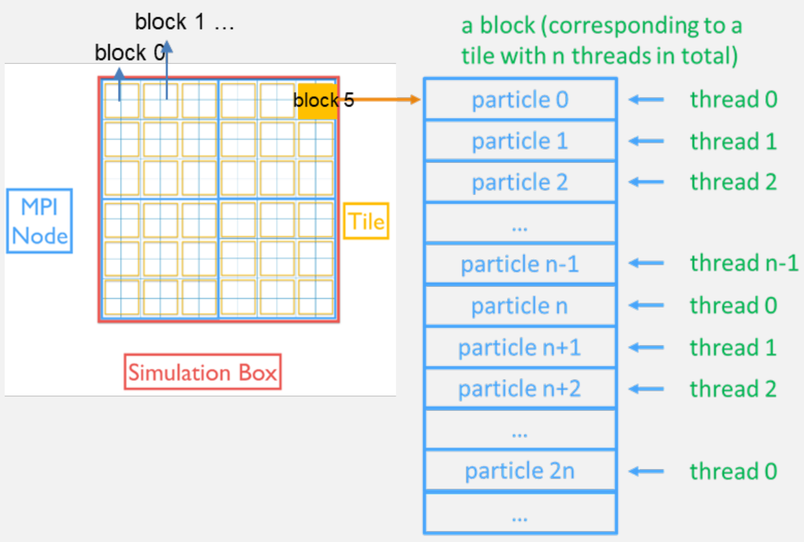

Figure 2. Mapping of CUDA architecture to simulation objects (withe particle simulation as an example).

Parallelizing CPU serial loops into GPU kernel functions risks concurrent thread access to shared memory addresses, leading to race conditions. For example, in calculating charege density distribution, multiple particles contribute to the same grid point, so threads may simultaneously read, modify, and write to the same memory location. Without synchronization, threads could read unmodified values, resulting in incomplete updates (e.g., only one particle's contribution recorded instead of the cumulative sum). To resolve this, CUDA-specific atomic operations (e.g., atomicAdd(), atomicMax()) were employed. These operations serialize access to shared/global memory locations, ensuring that only one thread modifies a target address at a time. In the CPU version, some functions (mainly functions that handle particle migration across tiles) include multiple OpenMP loops with inter-dependent results. We convert these loops into separate CUDA kernels called by the same host function, so that they are executed sequentially, and inter-dependent results are calculated before the are read by a different thread in a different block.
Particle migration, fast Fourier transform (FFT), and some other operations include MPI transfers across different nodes. We replace the original CPU-to-CPU MPI transfers with CUDA-aware GPU-to-GPU transfers, to avoid copying data to CPU memory before the transfer, and from CPU memory after the transfer.

Performace Evaluation

- **Accuracy**

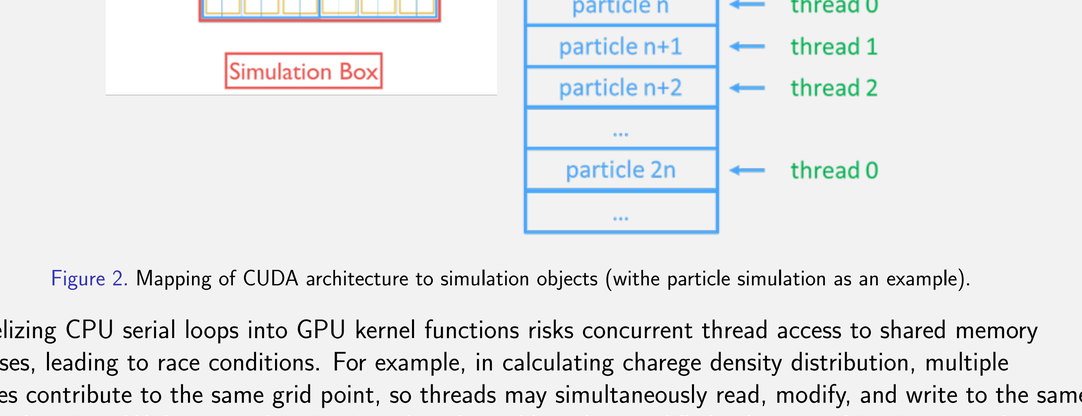

Figure 3. Simulated with the program before(left)/after(right) revision, electric field in direction z (ξ), distributed across the x-ξ plane.

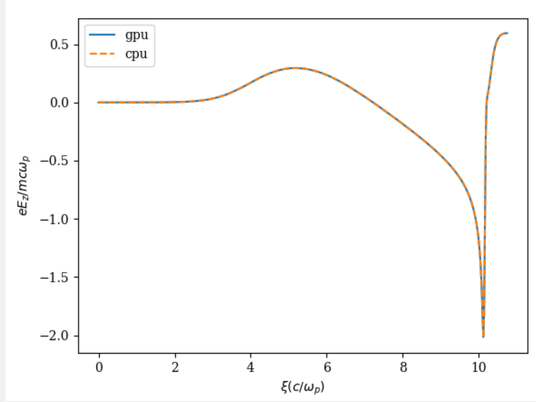

Figure 4. Comparison of simulation result before/after revision, electric field in direction z (ξ), distributed along the z axis. Simulations were conducted with 64 CPU cores (for the old program) and 1, 2 and 4GPUs (for the new program).

- **Speed**

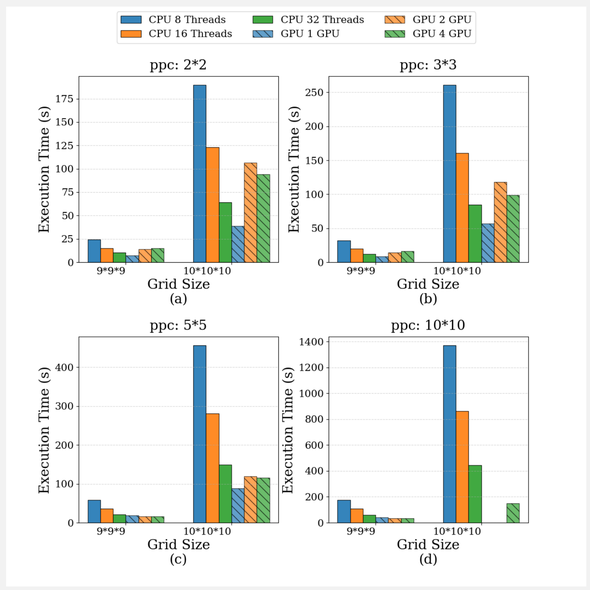

Figure 5. Measurement of simulation 2D time in seconds. Subplots (a), (b), (c) and (d) shows setups with 4/9/25/100 particles per cell, respectively. Execution time is measured with 8(blue, plain)/16(orange, plain)/32(green, plain) CPU threads and 1(blue, slashed)/2(orange, slashed)/4(green, slashed)GPUs. In each subplot, results with a simulation resolution of 512^3 are shown on the left side, and results with a simulation resolution of 1024^3 are shown on the right side.

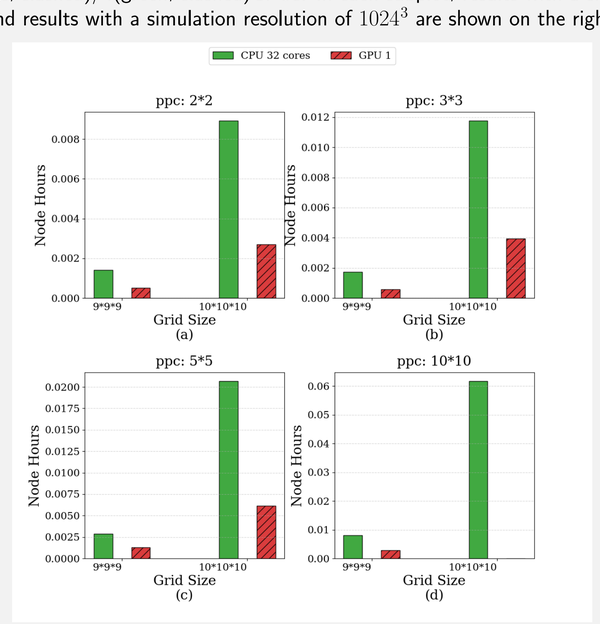

Figure 6. Number of node hours spent in 2D simulation. Subplots (a), (b), (c) and (d) shows setups with 4/9/25/100 particles per cell, respectively. Execution time is measured with 32(green, plain) CPU threads and 1(red, slashed) GPU. In each subplot, results with a simulation resolution of 512^3 are shown on the left side, and results with a simulation resolution of 1024^3 are shown on the right side.

Discussions

- **FFT transpose cost** In FFT transpose, using multiple GPUs lead to a high computational cost in data transfer across MPI nodes. This in turn causes the total simulation time of 2 and 4 GPUs to be longer than that of 1 GPU. In future revisions, we shall attempt to use finite difference field solvers to avoid this transpose across the whole simulation box.
- **3D loop revision** We plan to also move 3D particle and field solvers to GPUs, to reduce data transfer and optimize the 3D loop.

References

Joshi, C., S. Corde, and W. B. Mori (July 2020). "Perspectives on the generation of electron beams from plasma-based accelerators and their near and long term applications". In: *Physics of Plasmas* 27.7, p. 070602. ISSN: 1070-664X. DOI: 10.1063/5.0004039. eprint: https://pubs.aip.org/aip/pop/article-pdf/doi/10.1063/5.0004039/20006480/070602_1_5.0004039.pdf. URL: https://doi.org/10.1063/5.0004039.

Decyk, V. K. and T. V. Singh (2014). "Particle-in-Cell algorithms for emerging computer architectures". In: *Computer Physics Communications* 185.3, pp. 708–719. ISSN: 0010-4655. DOI: https://doi.org/10.1016/j.cpc.2013.10.013. URL: https://www.sciencedirect.com/science/article/pii/S001046551300341X.

An, W., V. K. Decyk, W. B. Mori, and T. M. Antonsen (2013). "An improved iteration loop for the three dimensional quasi-static particle-in-cell algorithm: QuickPIC". In: *Journal of Computational Physics* 250, pp. 165–177. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2013.05.020. URL: https://www.sciencedirect.com/science/article/pii/S0021999113003525.

Acknowledgements

Work supported by Strategic Priority Research Program of the Chinese Academy of Sciences, Grant No. XDB0530000, NSFC Grant 12075030, and Beijing Normal University Scientific Research Initiation Fund for Introducing Talents 310432104.