Application of Metaprogramming for Computational Code Generation in Parallel Particle Simulations

Wiktor Łodyga
Faculty of Electrical Engineering
Warsaw University of Technology
Warsaw, Poland
wiktor.lodyga@pw.edu.pl

I. Introduction

Particle-in-Cell coupled with Monte Carlo Collisions (PIC/MCC) is a standard kinetic method for low-temperature plasma simulations [1], [2]. Its timestep consists of particle motion, charge deposition, field solution, boundary handling, collision sampling, and particle management. Many of these operations are particle-local and therefore well suited to GPU execution [3].

The main difficulty addressed in this work is not only performance, but also code maintainability. Extending a PIC/MCC model with new species, collision channels, boundary conditions, or diagnostics usually requires manual changes in GPU kernels. The proposed approach replaces such manual variants with automatically generated, specialized CUDA solvers.

II. Method

The workflow is based on two Julia libraries. PlasmaModelingToolkit.jl defines the simulation case: domain, species, collisions, boundary conditions, and numerical parameters. Hennel.jl transforms this description into a self-contained Julia/CUDA solver module generated through metaprogramming [4], [5].

The generated code contains variable declarations, memory allocation, CUDA kernels, launch configuration, precomputed data, and the timestep schedule. Model choices known at generation time are resolved statically, which removes generic runtime branches, dead paths, and unnecessary control logic from GPU kernels.

The generator also records read and write accesses of metakernels. These dependency annotations are used to detect particle-local kernels that can be fused without changing the semantics of the timestep. Fusion reduces selected global-memory reads and writes and decreases the number of kernel launches.

III. Results and Conclusions

The generated electrostatic PIC/MCC solvers were tested using the capacitively coupled plasma benchmark of Turner et al. [6]. The generated code reproduced the expected time-averaged plasma profiles within the statistical uncertainty of the PIC/MCC method.

Performance was evaluated against the hand-written GPU implementation by Juhász et al. [3]. On an RTX 2080 Ti, the generated solvers achieved execution times of the same order of magnitude as the reference implementation, with runtime ratios of 1.05–1.86 across the tested Turner cases. Profiling confirmed that the dominant cost remains in particle-local stages, especially collision handling.

Kernel fusion was evaluated for compatible stages, such as particle motion followed by boundary handling. The fused kernels reduced intermediate global-memory traffic and produced a measurable reduction in the execution time of the fused part of the timestep.

These results show that metaprogramming can generate specialized CUDA solvers for PIC/MCC simulations from declarative model descriptions. The approach reduces manual kernel maintenance, preserves numerical and statistical correctness, gives performance comparable in order of magnitude to hand-written CUDA code, and enables automatic optimization through dependency-aware kernel fusion.

References

[1] C. Birdsall, "Particle-in-cell charged-particle simulations, plus Monte Carlo collisions with neutral atoms, PIC-MCC," *IEEE Transactions on Plasma Science*, vol. 19, no. 2, pp. 65–85, Apr. 1991. [Online]. Available: http://ieeexplore.ieee.org/document/106800/

[2] Z. Donkó, "Particle simulation methods for studies of low-pressure plasma sources," *Plasma Sources Science and Technology*, vol. 20, no. 2, p. 024001, Apr. 2011. [Online]. Available: https://iopscience.iop.org/article/10.1088/0963-0252/20/2/024001

[3] Z. Juhasz, J. Ďurian, A. Derzsi, Matejčík, Z. Donkó, and P. Hartmann, "Efficient GPU implementation of the Particle-in-Cell/Monte-Carlo collisions method for 1D simulation of low-pressure capacitively coupled plasmas," *Computer Physics Communications*, vol. 263, p. 107913, Jun. 2021. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0010465521000503

[4] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A Fresh Approach to Numerical Computing," *SIAM Review*, vol. 59, no. 1, pp. 65–98, Jan. 2017. [Online]. Available: https://epubs.siam.org/doi/10.1137/141000671

[5] T. Besard, C. Foket, and B. De Sutter, "Effective Extensible Programming: Unleashing Julia on GPUs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 4, pp. 827–841, Apr. 2019, arXiv:1712.03112 [cs]. [Online]. Available: http://arxiv.org/abs/1712.03112

[6] M. M. Turner, A. Derzsi, Z. Donkó, D. Eremin, S. J. Kelly, T. Lafleur, and T. Mussenbrock, "Simulation benchmarks for low-pressure plasmas: Capacitive discharges," *Physics of Plasmas*, vol. 20, no. 1, p. 013507, Jan. 2013. [Online]. Available: http://aip.scitation.org/doi/10.1063/1.4775084