C++ Standard Parallelism for GPU Programming in a Particle-In-Cell Application

ESTER EL KHOURY, Université Paris-Saclay, UVSQ, CNRS, CEA, Maison de la Simulation, France and Université Paris-Saclay, CEA, Laboratoire en Informatique Haute Performance pour le Calcul et la simulation, France
MATHIEU LOBET, Université Paris-Saclay, UVSQ, CNRS, CEA, Maison de la Simulation, France
JULIEN BIGOT, Université Paris-Saclay, UVSQ, CNRS, CEA, Maison de la Simulation, France
LAURENT COLOMBET, CEA, DAM, DIF, France and Université Paris-Saclay, CEA, Laboratoire en Informatique Haute Performance pour le Calcul et la simulation, France

Performance portability remains a major challenge in high-performance computing (HPC) as applications increasingly target diverse GPU architectures. In this work, we evaluate C++ Standard Parallelism (stdpar) for a classical Particle-In-Cell (PIC) method on recent NVIDIA and AMD GPUs, and compare it against established parallel programming models such as Thrust, Kokkos, and SYCL, in terms of both runtime performance and programming productivity. Since stdpar relies on unified memory, we evaluate it against other models both with and without this feature enabled. The PIC implementation is dominated by the projection operator, which makes intensive use of atomic operations. Our analysis includes both a global performance assessment of the main loop and a more fine-grained examination of the projection kernel. On NVIDIA architectures, stdpar processes 1.7× fewer particles than Kokkos and 1.1× fewer than Thrust under equivalent conditions, despite its productivity benefits for developers. These findings underline the trade-offs between ease of development and peak performance when adopting stdpar for GPU-based HPC workloads.

CCS Concepts: • **Computing methodologies** → **Parallel programming languages**.

Additional Key Words and Phrases: C++, Parallelism, GPU, Unified Memory, Particle-in-Cell, stdpar

ACM Reference Format:
Ester El Khoury, Mathieu Lobet, Julien Bigot, and Laurent Colombet. 2025. C++ Standard Parallelism for GPU Programming in a Particle-In-Cell Application. In *Proceedings of The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '25)*. ACM, New York, NY, USA, 3 pages.

1 Introduction

Heterogeneous architectures combining CPUs and GPUs are widely used in HPC, especially for physics simulations. However, fully leveraging these architectures remains challenging. Many scientific applications are developed by physicists with limited programming expertise. An additional layer of difficulty arises from the diversity of GPU vendors (e.g., NVIDIA, AMD, Intel), each offering distinct programming models such as CUDA and HIP. This heterogeneity forces developers to maintain multiple versions of their codebases, thereby increasing both development and maintenance efforts. These challenges highlight a fundamental trade-off in HPC: balancing between Performance, Performance Portability and Productivity. To address these challenges, the C++17 standard introduced "stdpar", a high-level parallelism model aimed at simplifying parallel programming. NVIDIA was the first to extend this model for GPU execution in

Authors' Contact Information: Ester El Khoury, ester.elkhoury@cea.fr, Université Paris-Saclay, UVSQ, CNRS, CEA, Maison de la Simulation, Gif-sur-Yvette, France and Université Paris-Saclay, CEA, Laboratoire en Informatique Haute Performance pour le Calcul et la simulation, Bruyères-le-Châtel, France; Mathieu Lobet, mathieu.lobet@cea.fr, Université Paris-Saclay, UVSQ, CNRS, CEA, Maison de la Simulation, Gif-sur-Yvette, France; Julien Bigot, julien.bigot@cea.fr, Université Paris-Saclay, UVSQ, CNRS, CEA, Maison de la Simulation, Gif-sur-Yvette, France; Laurent Colombet, laurent.colombet@cea.fr, CEA, DAM, DIF, Arpajon, France and Université Paris-Saclay, CEA, Laboratoire en Informatique Haute Performance pour le Calcul et la simulation, Bruyères-le-Châtel, France.

2025. Manuscript submitted to ACM

heterogeneous computing environments, with AMD later adding support for stdpar as well. To further ease development, both vendors introduced memory management technologies that abstract away manual data transfers [1, 3].

2 MiniPIC Study

We port the PIC method to stdpar using MiniPIC [4], a mini-application serving as a research testbed for evaluating parallel programming models. To the best of our knowledge, this is the first study of a PIC implementation using stdpar.

The plasma is modeled using macro-particles stored in memory using a Structure of Arrays layout [2]. Our experiments target single-GPU execution and covers four main operators: interpolation, pusher, projection, and the Maxwell solver. Among these, current projection is the most computationally intensive. As particles move, they induce electric currents that must be accumulated on a spatial grid. This process introduces parallelization challenges due to concurrent updates: independently processed particles may access the same grid nodes simultaneously. To ensure correctness, atomic operations are employed. While more efficient techniques exist to reduce atomic usage, they require fine-grained parallel control unavailable in stdpar.

3 Benchmarking Results

Evaluating stdpar is timely given the growing adoption of standard C++ in HPC. Since stdpar relies on Thrust as its backend in NVIDIA's NVHPC SDK and on rocThrust for AMD GPUs since ROCm 6.1, benchmarking Thrust directly provides valuable insight into the performance and portability of standard parallel algorithms. Kokkos is included as a widely used, performance-portable model, and both Kokkos and Thrust are tested with Unified Virtual Memory (UVM) enabled to mirror stdpar's implicit memory model. Our evaluation also extend to SYCL for broader cross-vendor coverage. Experiments were conducted on six recent GPU architectures: NVIDIA V100, A100, H100, GH200, and AMD MI250X and MI300A. Our results indicate that stdpar greatly enhances developer productivity through its high-level abstraction, yet its performance is highly workload-dependent and often trails behind established models. On NVIDIA and AMD GPUs, Kokkos perform similarly with or without UVM, showing UVM isn't the main reason for stdpar's gap. On AMD GPUs, early MI250X tests revealed performance limitations in both stdpar and Thrust due to inefficient UVM handling, an issue that appears resolved on the MI300A. The observed gaps between stdpar and other models, likely stem from the API's high abstraction level, which constrains compiler optimizations, and the lack of hierarchical parallelism, which limits low-level control compared to Kokkos. In the projection operator, which dominates the PIC workload, stdpar achieves performance gains of approximately +19% on H100 and +32% on GH200 relative to the V100 and MI250 baselines. These results indicate that improvements remain modest for scoped atomics in standard C++. The SYCL performance on MI300A is worst than stdpar and this is due to the poor performance of the atomic reductions in the projection kernel.

4 Future works

Going forward, we will investigate performance gaps between stdpar and Thrust, optimize critical kernels such as the Projection operator, and rewrite the mini-application in CUDA. We also plan to explore asynchronous programming and propose enhancements to C++ libraries for better GPU support.

5 Acknowledgments

We sincerely thank all members of the Maison de la Simulation laboratory for valuable discussions and support. Additionally, this work has received support from the CExA[1] Moonshot project of the CEA. The work has been supported by AIDAS - AI, Data Analytics and Scalable Simulation - which is a Joint Virtual Laboratory gathering the Forschungszentrum Jülich (FZJ) and the French Alternative Energies and Atomic Energy Commission (CEA).

References

[1] Tyler Allen and Rong Ge. 2021. In-Depth Analyses of Unified Virtual Memory System for GPU Accelerated Computing. In *SC21: International Conference for High Performance Computing, Networking, Storage and Analysis* (St. Louis, MO, USA). 1–14. doi:10.1145/3458817.3480855

[2] A. Beck, J. Derouillat, M. Lobet, and et al. 2019. Adaptive SIMD optimizations in particle-in-cell codes with fine-grain particle sorting. *Computer Physics Communications* 244 (2019), 246–263. doi:10.1016/j.cpc.2019.05.001

[3] NVIDIA Corporation. 2020. Unified Memory in CUDA: A Beginner's Guide. [Online]. Available: https://developer.nvidia.com/blog/unified-memory-cuda-beginners/. Accessed: May 27, 2024..

[4] J.J. Silva-Cuevas, M. Zych, K. Peyen, I. Kabadshow, and M. Lobet. 2025. Towards a complete task-based implementation of a 3D particle-in-cell code: Performance studies and benchmarks. *Computer Physics Communications* 313 (2025), 109647. doi:10.1016/j.cpc.2025.109647

[1] CExA Moonshot Project: https://cexa-project.org