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

CExA Kokkos Tea Time

May 21, 2025

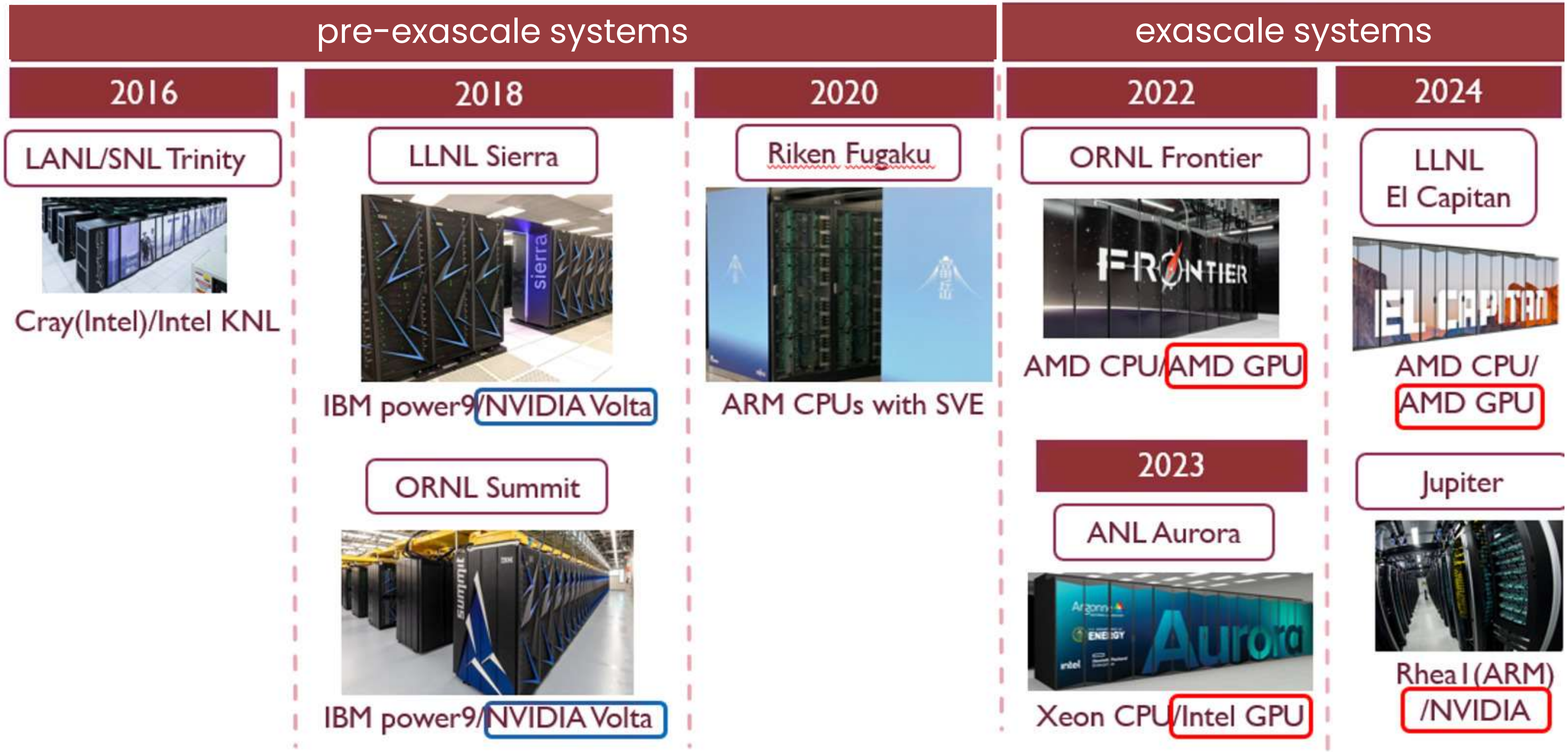
Ester El Khoury

ester.elkhoury@cea.fr

Maison de la Simulation, CEA Saclay, 91191 Gif-sur-Yvette CEDEX

Outline

1. Performance Portability



1. Performance Portability

Constructors

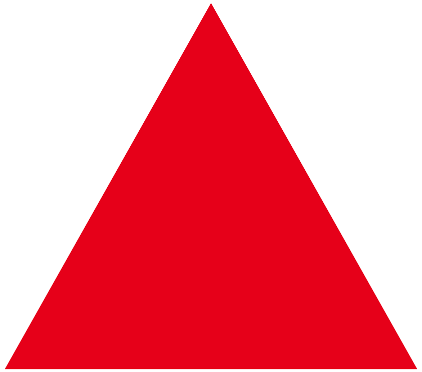
Low-Level Language
- Advanced optimizations with fine-grained control
- Very high performance
- Limited portability

Examples:
- CUDA (for NVIDIA GPUs)
- HIP (for AMD GPUs)

Portability in a Scientific Context

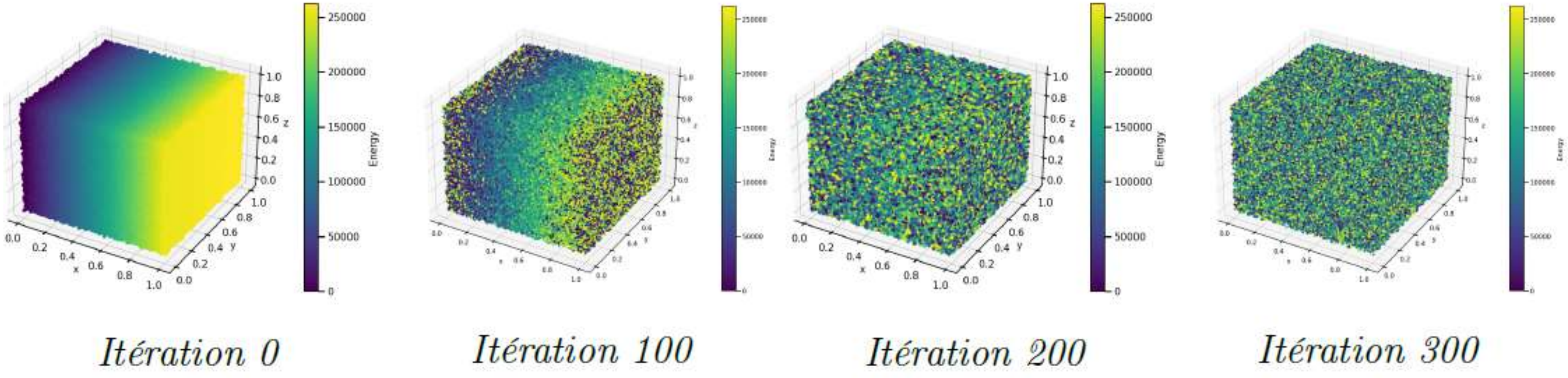
High-Level Language
- Good performance
- Less manual optimization
- Better portability
- Increased productivity

Examples:
- Directive-based models: OpenMP, OpenACC
- C++-based libraries : Thrust, Kokkos
- Evolution of C++



1. Performance Portability

Does the C++17 Standard Parallelism model offer a good compromise between Performance, Portability, and Productivity especially when running large-scale plasma physics simulations on a single GPU?



2. C++ Standard Parallelism
General Overview

- **No language extensions**
  Example: CUDA : __host__, __device__, __global__
- **No special directives**
  Example: OpenACC : #pragma
- **No new libraries**
  Example: Thrust
- **Uses familiar C++ objects**
- **Same abstraction model for memory**
  Example: std::vector
- **Implicit memory management**

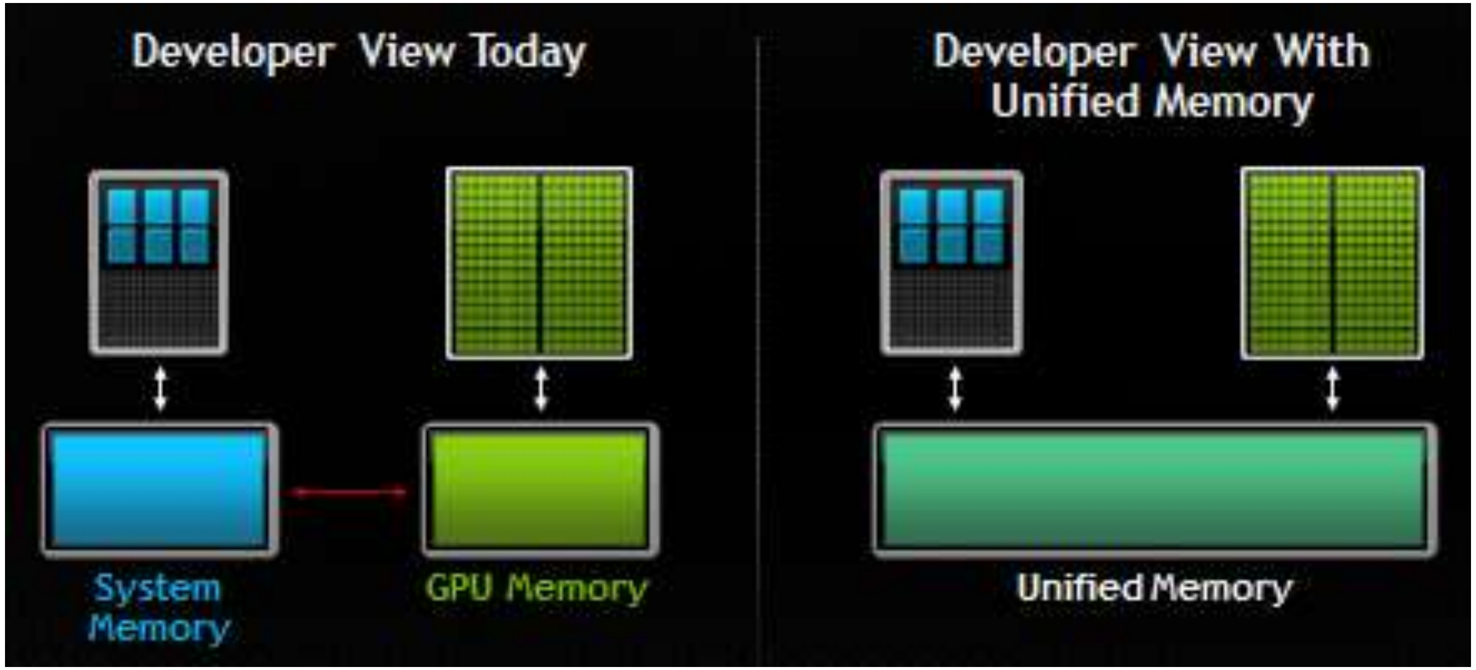
2. C++ Standard Parallelism
Execution Policies

Serial (C++98)	Parallel (C++17)
`std::vector<T> x{…};` `std::transform(x.begin(), x.end(), x.begin(), [](int x) { return x + 1; });`	`std::vector<T> x{…};` `std::transform(std::execution::par_unseq, x.begin(), x.end(), x.begin(), [](int x) { return x + 1; });`

- **C++ defines four execution policies:**
  - **std::execution::seq:** Sequential execution. No parallelism is allowed.
  - **std::execution::unseq:** Vectorized execution on the calling thread, but not parallel
  - **std::execution::par:** Parallel execution on one or more threads
  - **std::execution::par_unseq:** Parallel execution on one or more threads, with each thread potentially vectorized

2. C++ Standard Parallelism
NVIDIA Stdpar implementation

Memory management



Still physically separate memory spaces,
BUT a unified virtual address space

nvc++ compiler flags

CPU

nvc++ -stdpar=multicore

GPU

nvc++ -stdpar=gpu
nvc++ -stdpar

Clang compiler flags

clang++ --hipstdpar

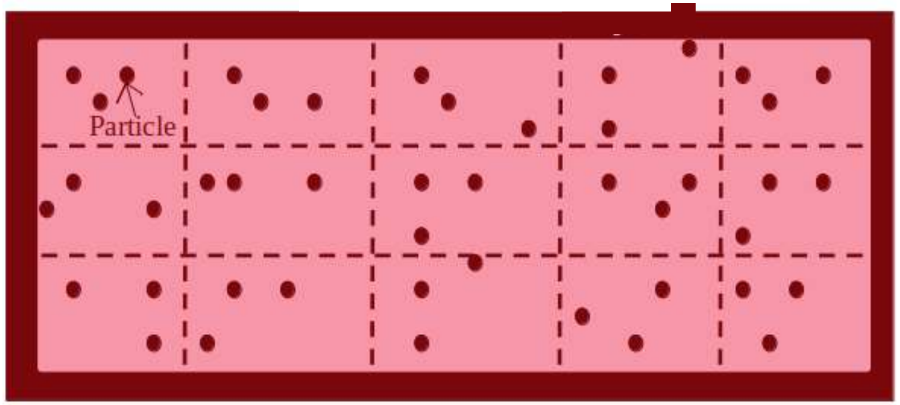
3. MiniPIC
Description

- MiniPIC: A Mini Application
- Based on Smilei, a simulation code for plasma physics
- Uses the **Particle-in-Cell (PIC)** model
- **Goal**: Explore new programming models

- C++ Code
- No distributed-memory parallelism

- Several Programming Models Implemented: SYCL, Thrust, Kokkos, OpenMP, OpenMP Target, OpenACC

GPU
Domain

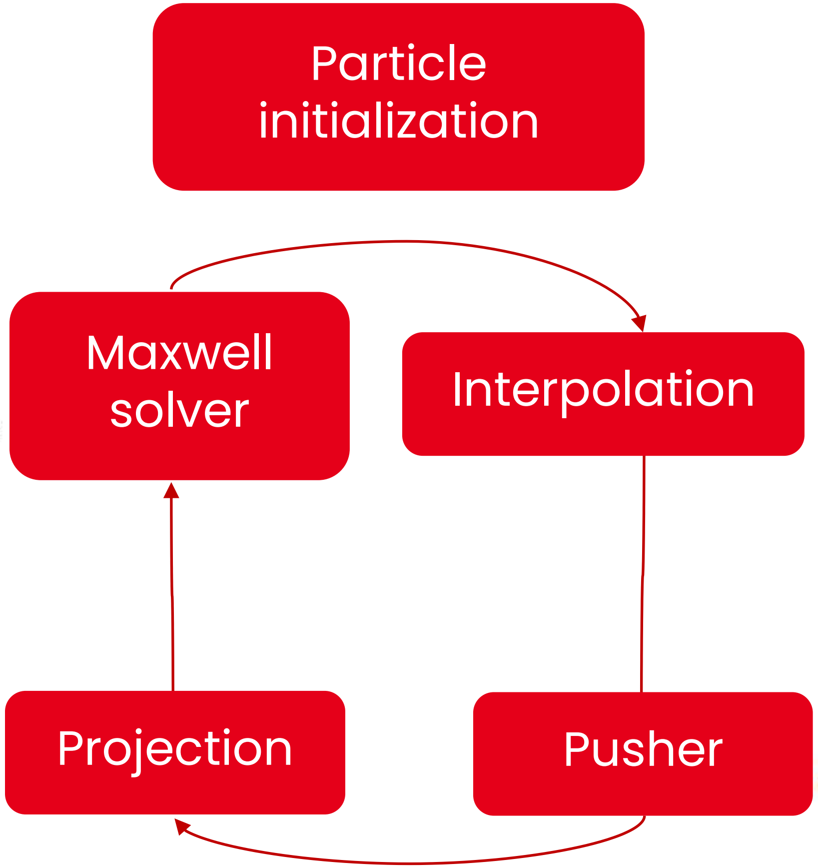


- A Domain Consists of Two Data Structures:
  - **Particles:** Represent matter
  - **Grid:** Represents electromagnetic fields

- Interaction between particles and the grid

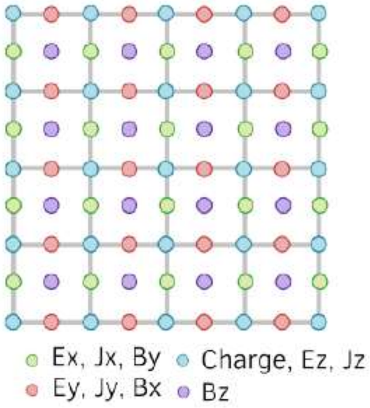
3. MiniPIC

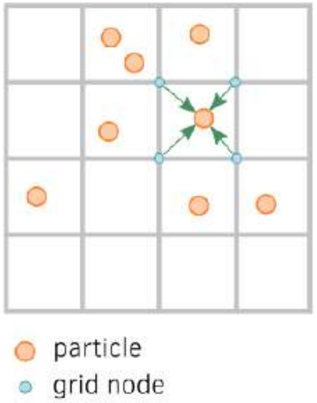
PIC loop: The four main operators



- Stencil problem
- Less computationally expensive

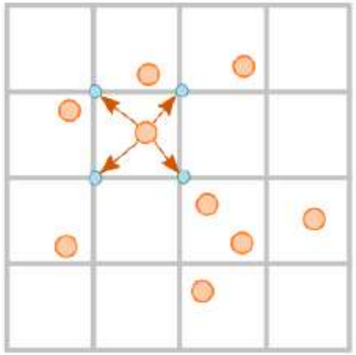
⇒ Parallelizes efficiently on GPU

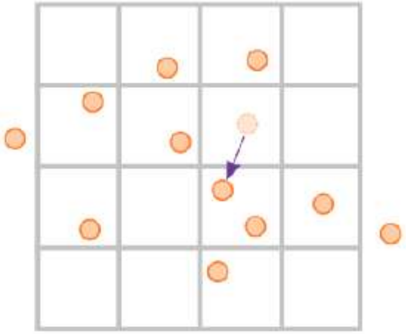




- Random memory access on the grid
- Problem: Cache miss

Random access
+
Memory contention





- Vector problem
- Indices are independent
- Contiguous memory access

⇒ Easy to port

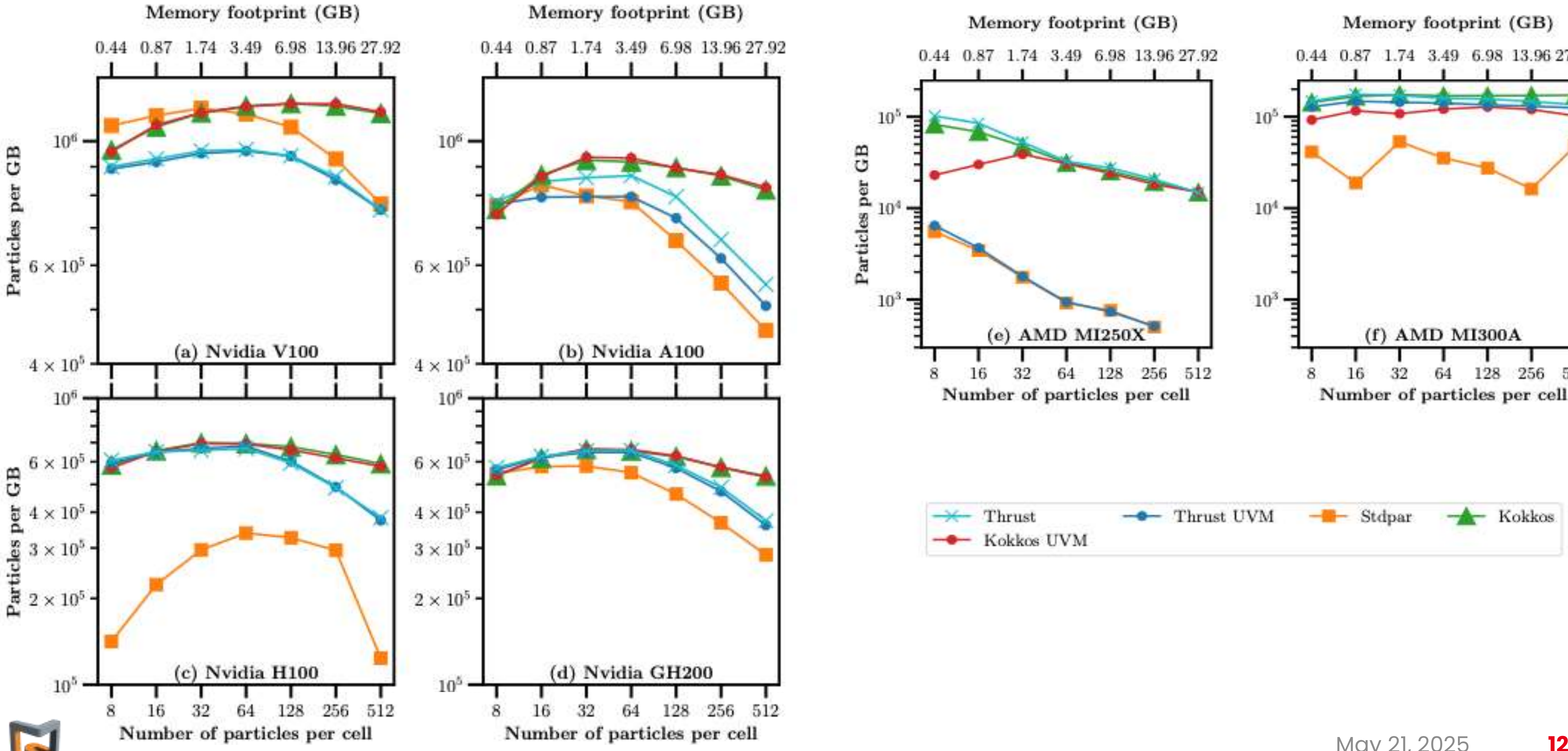
4. Methodology

- The case studied: Thermal plasma
- It consists of electrons and ions with a uniform charge distribution
- 64 cells per direction

GPU Name	CUDA/ ROCM Version	Compiler Version	Memory Size/Type	Memory Bandwidth (MB/s)	Double-Precision (TFLOPS)
NVIDIA V100 (PCIe 32GB)	12.2.1	nvc++ 23.7	32GB/HMB2	900	7
NVIDIA A100 (SXM 40GB)	12.2.1	nvc++ 23.7	40GB/HBM2	1555	9.7
NVIDIA H100 80GB	12.2.1	nvc++ 23.7	80GB/HBM3	3350	34
NVIDIA GH200 480GB	12.3	nvc++ 24.1	96GB/HBM3	4000	34
AMD MI250X	rocm-6.1.2	clang++ 17.0.0	128GB/HBM2e	3200	47.9
AMD MI300A	rocm-6.1.2	clang++ 17.0.0	128GB/HBM3	5300	61.3

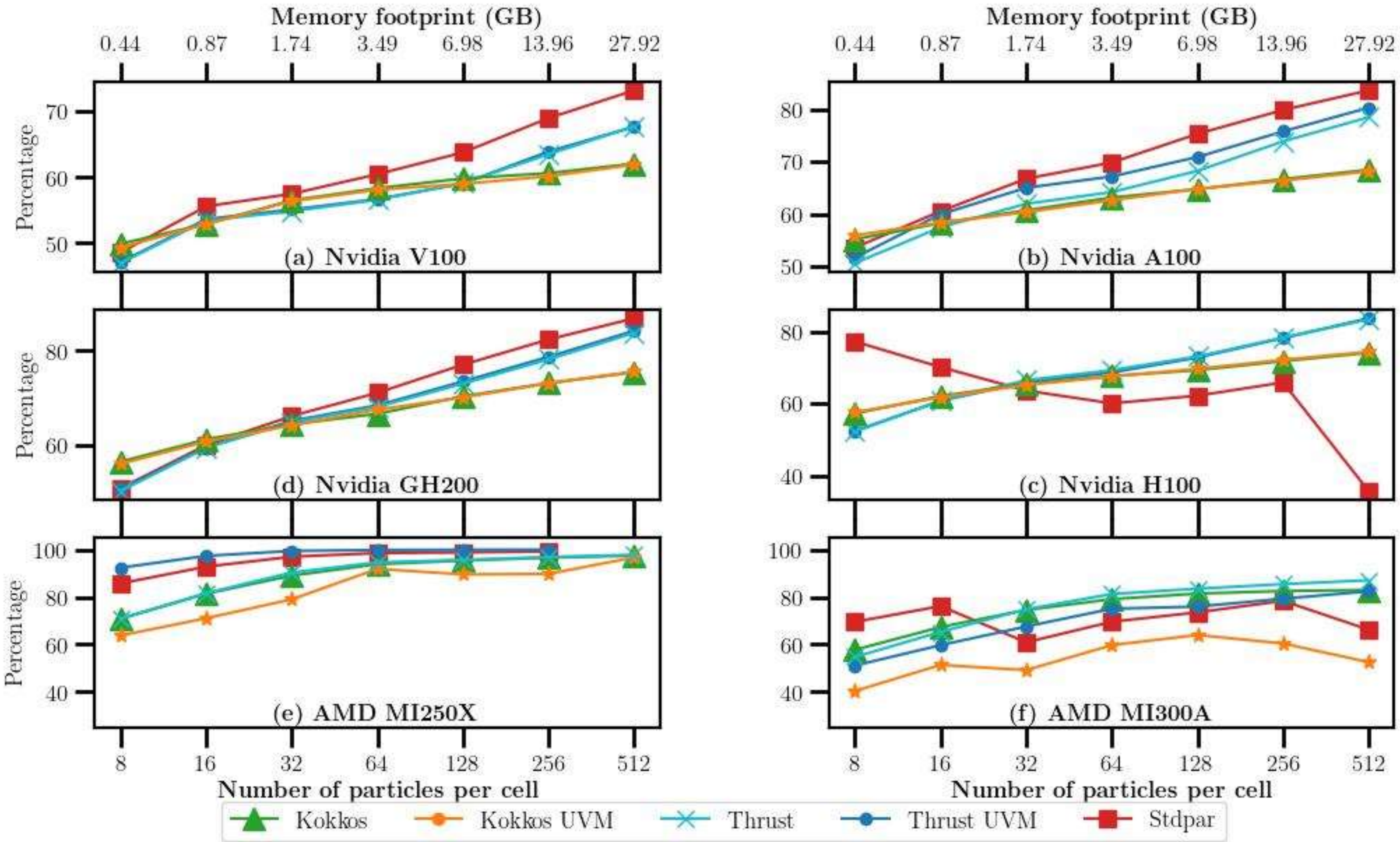
- Comparison with :
  - Thrust
  - Kokkos
  - Thrust with UVM
  - Kokkos with UVM

5. Benchmarking Results
PIC Loop



5. Benchmarking Results

Projection Loop



6. Conclusion
Strengths

- Accessible to non-experts
- No major rewrite of existing C++ code
- No explicit memory management
- Compatible with NVIDIA and AMD
- Classic dynamic memory allocation
- High performance for simple loops

6. Conclusion
Limitations

- Limited optimization for experts
- CPU-GPU switching without explicit management → inefficient data transfers
- Execution policy implementation depends on compiler developers
- Model still in development
- Potential for further improvement

7. Looking Ahead

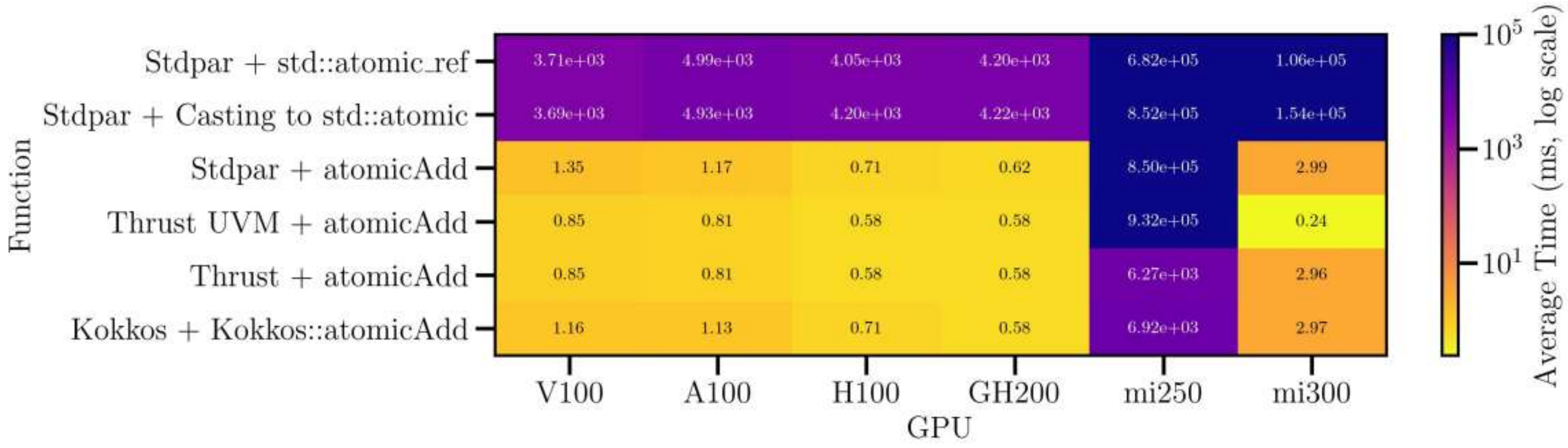
Untapped Potential

- Investigate the performance differences between different programming models, particularly in comparison with Thrust.
- Analyze performance variations between the H100 and GH200.

Future Direction

- Focus on the exploration of asynchronous programming.

8. GPU Atomicity: Early Insights



Thank you