OpenMC

Experiences with OpenMP Target Offloading in the OpenMC Monte Carlo Particle Transport Application

John Tramm, PhD
Assistant Computational Scientist
Argonne National Laboratory

ECP EXASCALE COMPUTING PROJECT

OpenMP Monthly Telecon – Feb 25, 2022

Argonne NATIONAL LABORATORY

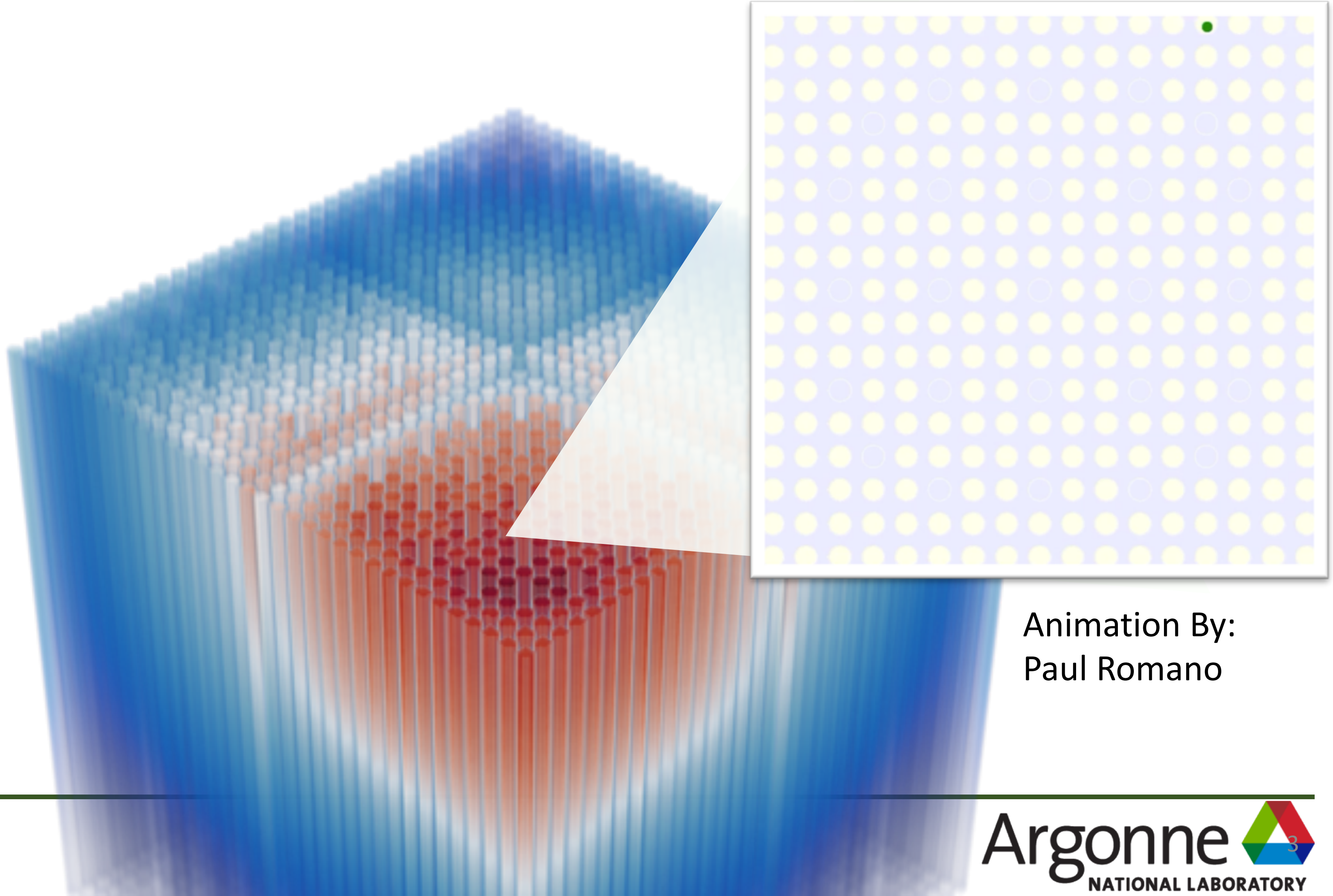
Acknowledgement

This work is a collaborative project with major contributions from **Paul Romano, Johannes Doerfert, Amanda Lund, Patrick Shriwise, Andrew Siegel, Gavin Ridley, and Andrew Pastrello.**

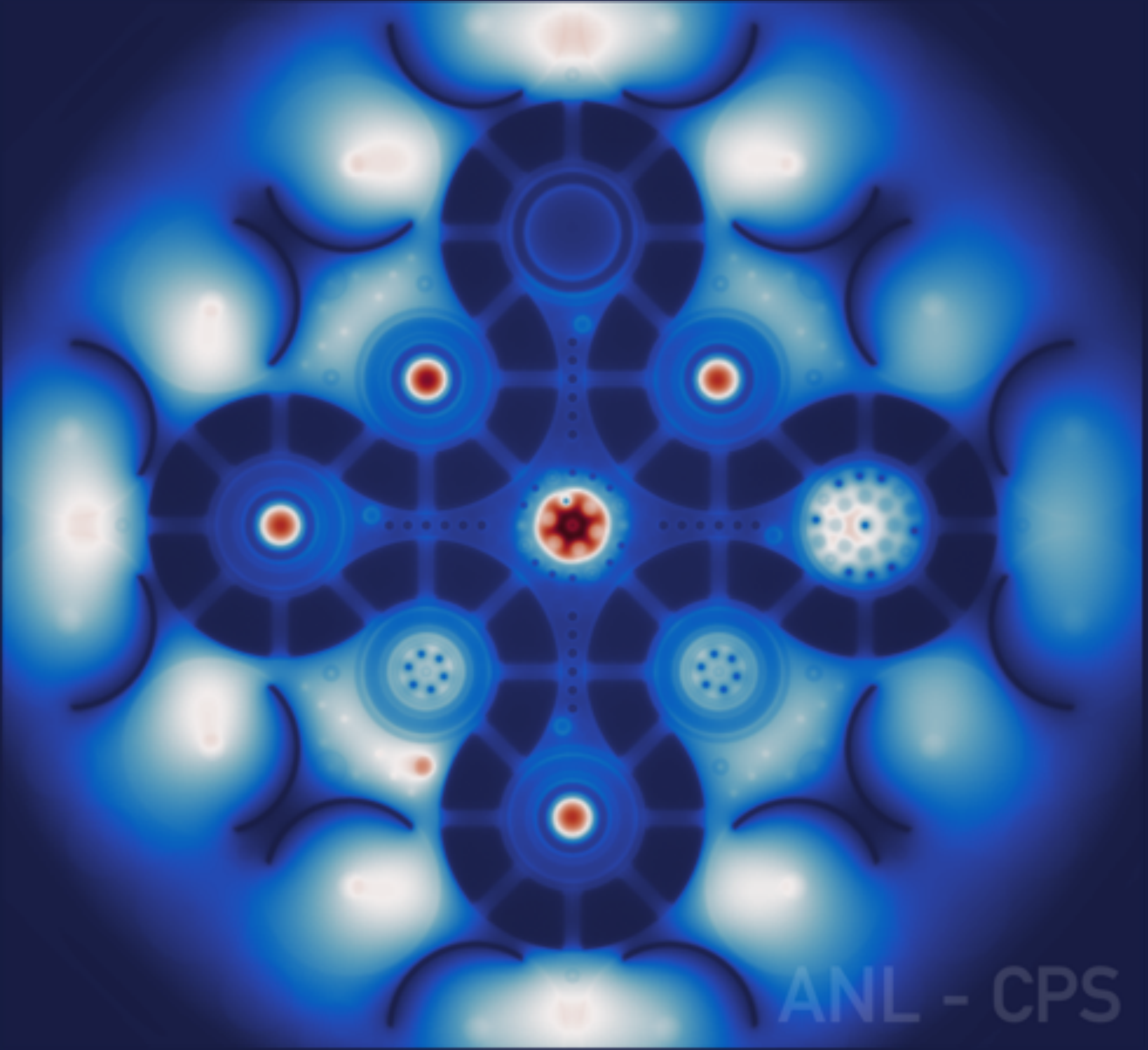
This presentation has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan. http://energy.gov/downloads/doe-public-access-plan.

What is Monte Carlo (MC) Particle Transport?

- Simulates individual particles as they move through and interact with material geometries
- High-fidelity and general purpose
- High computational cost
- Stochastic nature of simulation creates many challenges in terms of running efficiently on HPC architectures



Animation By:
Paul Romano



What is OpenMC?

- Monte Carlo (MC) neutral particle transport application
- Part of the ExaSMR ECP project
- Open source:
  - Started by Paul Romano
  - 52 contributors
  - Primarily developed at ANL
- Modern C++, with parallelism expressed via MPI + OpenMP

Argonne NATIONAL LABORATORY

Porting to OpenMP: Main **Programming** Challenges

Original CPU-Oriented Code		OpenMP Offloading GPU Port
Virtual functions	→	Tagged unions
STL containers (e.g., std::vector)	→	Pointers
Nested, complex data structures	→	Tons of mapping code

EXASCALE COMPUTING PROJECT

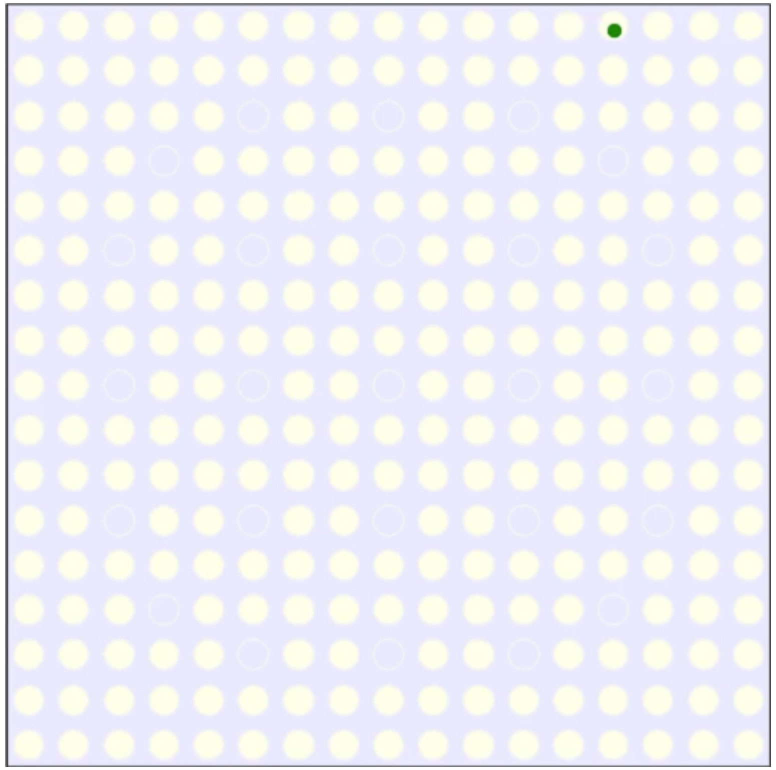
Argonne NATIONAL LABORATORY

Porting to GPU: Main **Algorithmic** Challenges

Original CPU-Oriented Code		OpenMP Offloading GPU Port
History-Based Algorithm	→	Event-Based Algorithm
Legacy CPU-Oriented Optimizations	→	New, GPU-Oriented Optimizations
Unsorted	→	Particle Sort in Energy

Algorithmic Challenges

History-Based Transport Example



Animation by Paul Romano

"History-Based" Parallelism

- Each particle undergoes random series of different events (collisions, movements, tallies, etc) from birth to death
- Parallelism expressed at high level over independent particles
- **Single monolithic GPU kernel**

"Event-Based" Parallelism

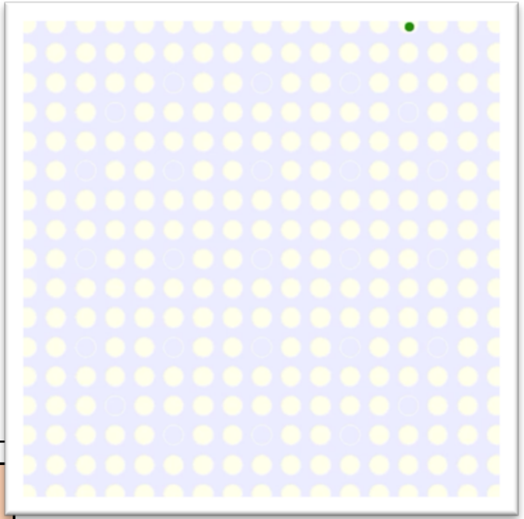
- Originally developed in the 80's for vector computers
- Only execute one low level event type at a time (**kernel splitting**)
- Parallelism expressed over particles requiring that event
- Greatly **reduces thread divergence**
- Opens the door to other GPU-centric optimizations
- **Many smaller GPU kernels**

History-Based Transport: Optimal for CPU

kernel

Algorithm 1 History-based algorithm in a full MC transport application

```
for each particle do                                          ▷ Independent
    while particle is alive do                                ▷ Dependent
        Event A: Compute macroscopic cross sections
        Event B: Sample distance to collision and collision type
        Event C: Move particle to collision site
        Event D: Process particle collision
        Event ...
    end while
end for
```



Animation Source: Paul Romano

The problem: particles will undergo different events in different order, resulting in very low (or zero) SIMD efficiency

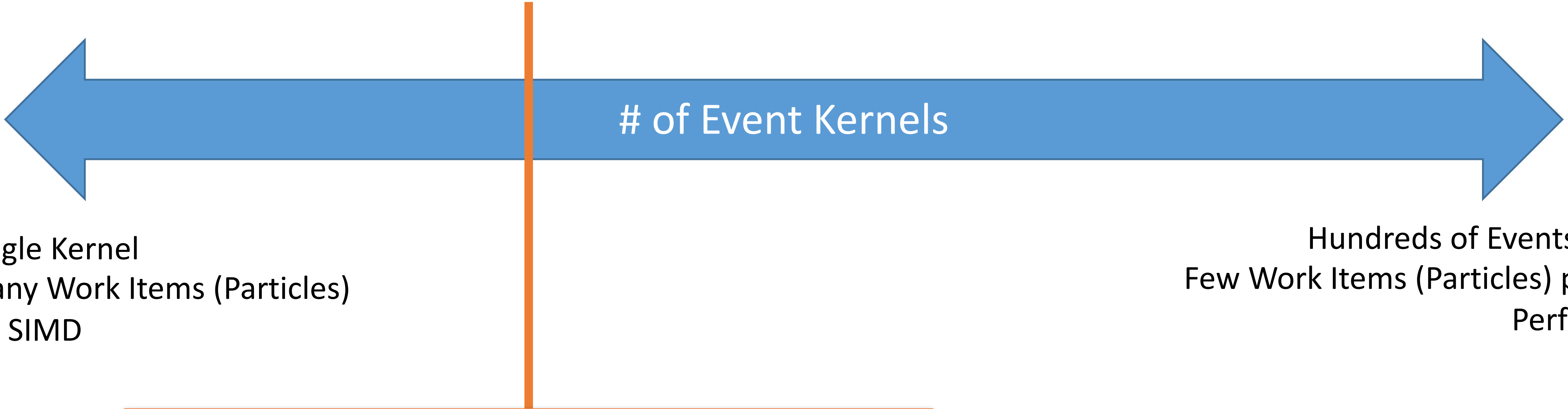
Event-Based Transport: Optimal for GPU

Algorithm 1 Event-based algorithm in a full MC transport application

```
       initialize buffer of particles
       while any particles are still alive do                        ▷ Dependent
kernel     for each alive particle do                                ▷ Independent
               Event A: Compute macroscopic cross sections
           end for
kernel     for each alive particle do                                ▷ Independent
               Event B: Sample distance to collision and collision type
           end for
kernel     for each alive particle do                                ▷ Independent
               Event C: Move particle to collision site
           end for
kernel     for each alive particle do                                ▷ Independent
               Event D: Process particle collision
           end for
kernel     for each alive particle do                                ▷ Independent
               Event ...
           end for
kernel     sort/consolidate surviving particles                      ▷ stream compaction
       end while
```

- **Solution: kernel splitting.** Parallelize over events instead, execute all particles that need that event in SIMD
- Host decides which event kernel to launch based on how many particles in that queue
- **Downside**: buffering of particles between events
- **Upside**: greatly reduced branching, potential for vectorization

Event Size Balance: How Many Kernels to Use?



Single Kernel
Many Work Items (Particles)
No SIMD

Hundreds of Events/Kernels
Few Work Items (Particles) per Event
Perfect SIMD

For MC, previous work suggests having O(5-10) events is optimal balance

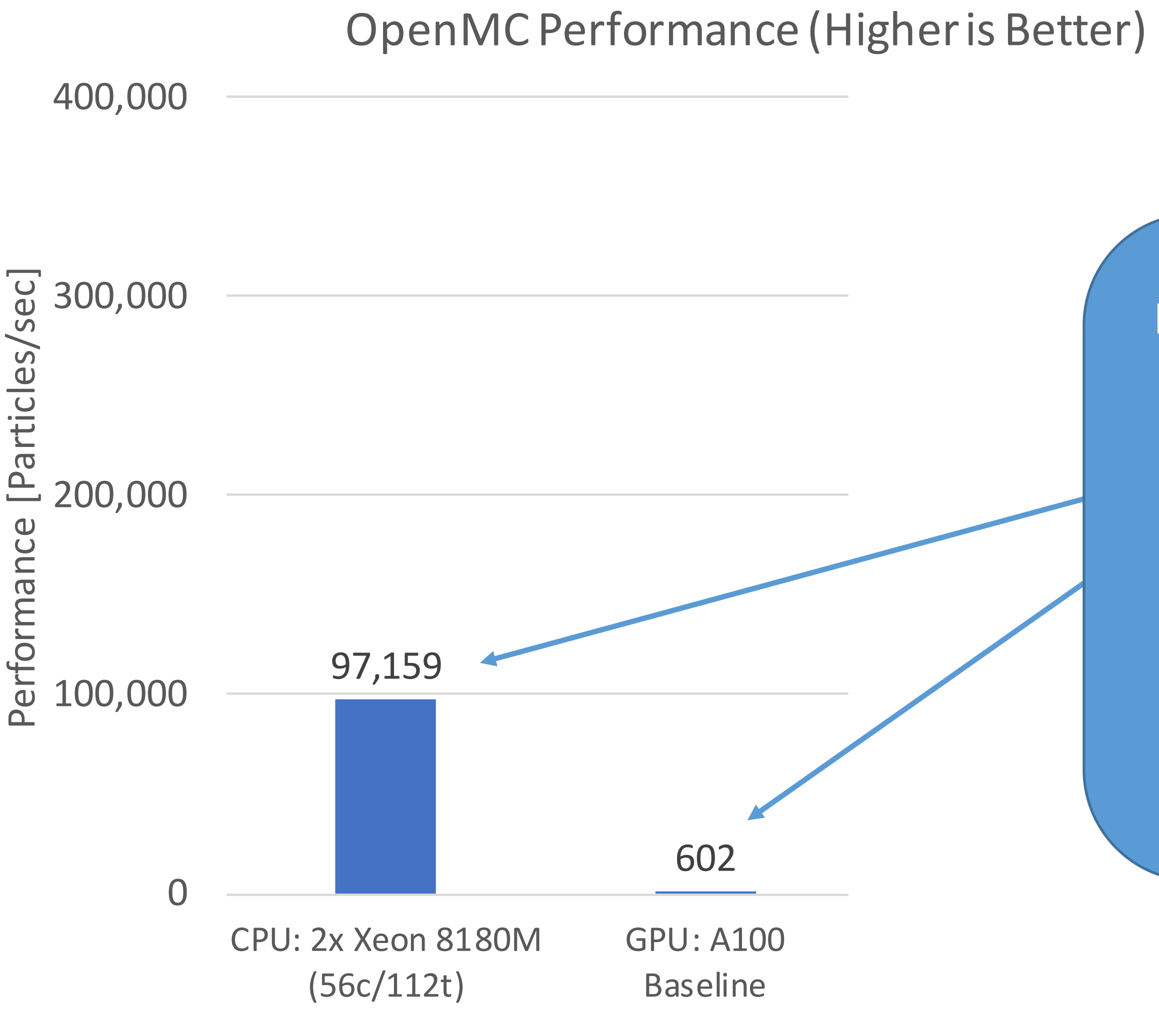
OpenMC Events

OpenMC Event Kernels

- Particle Initialization
- Calculate Cross Sections (Fuel)
- Calculate Cross Sections (non-Fuel)
- Advance Particle
- Cross Surface
- Collision
- Particle Death

- All main **event kernels** in OpenMC have been offloaded to device
- Some kernels are very large:
  - Deep call stacks
  - Functions scattered over many files
  - O(1000's) lines of code per kernel

Initial GPU Results



First results with LLVM compiler on A100 GPU were obtained in mid 2021.

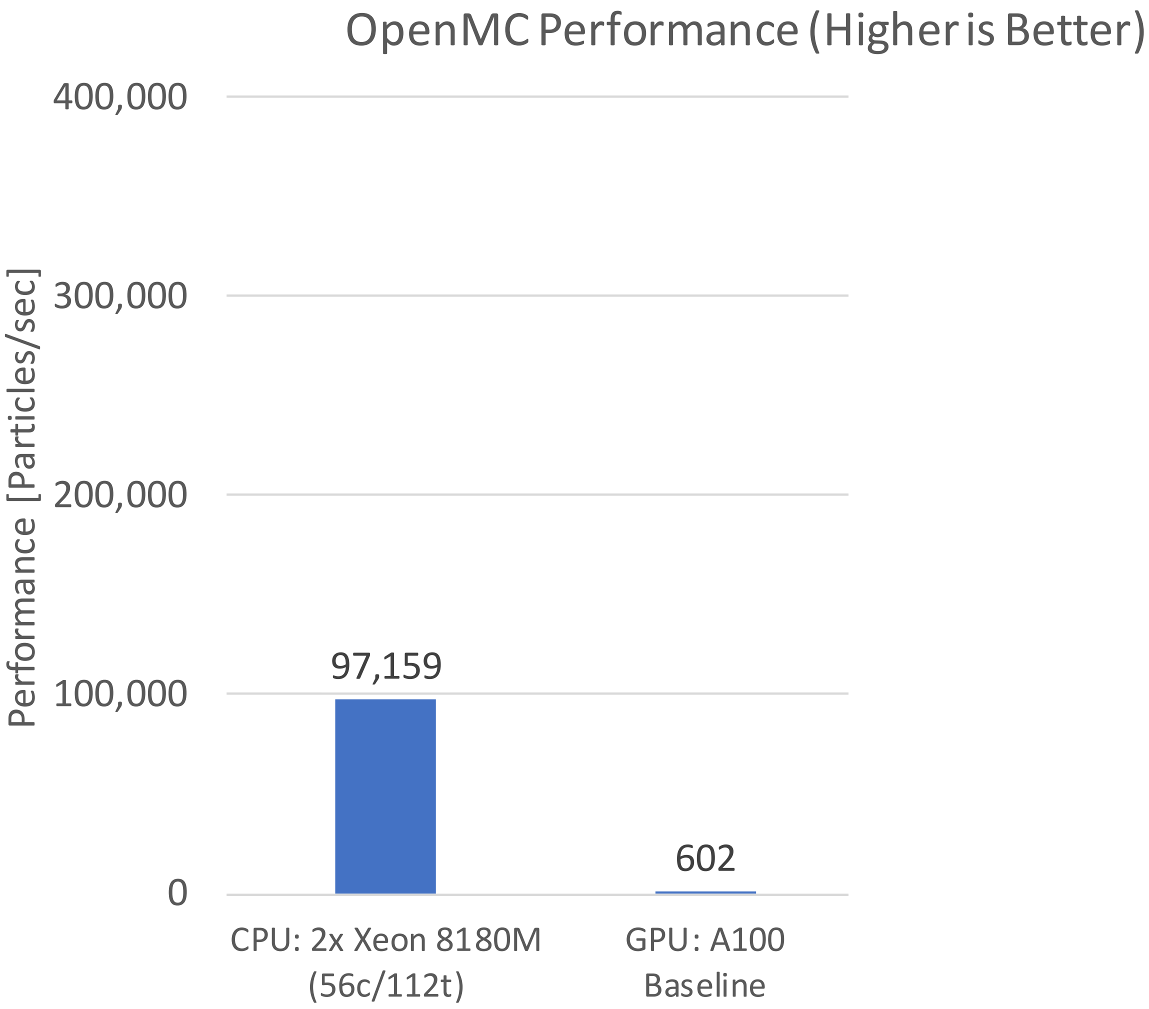
Performance of A100 was equivalent to less than a single CPU core!

Why?

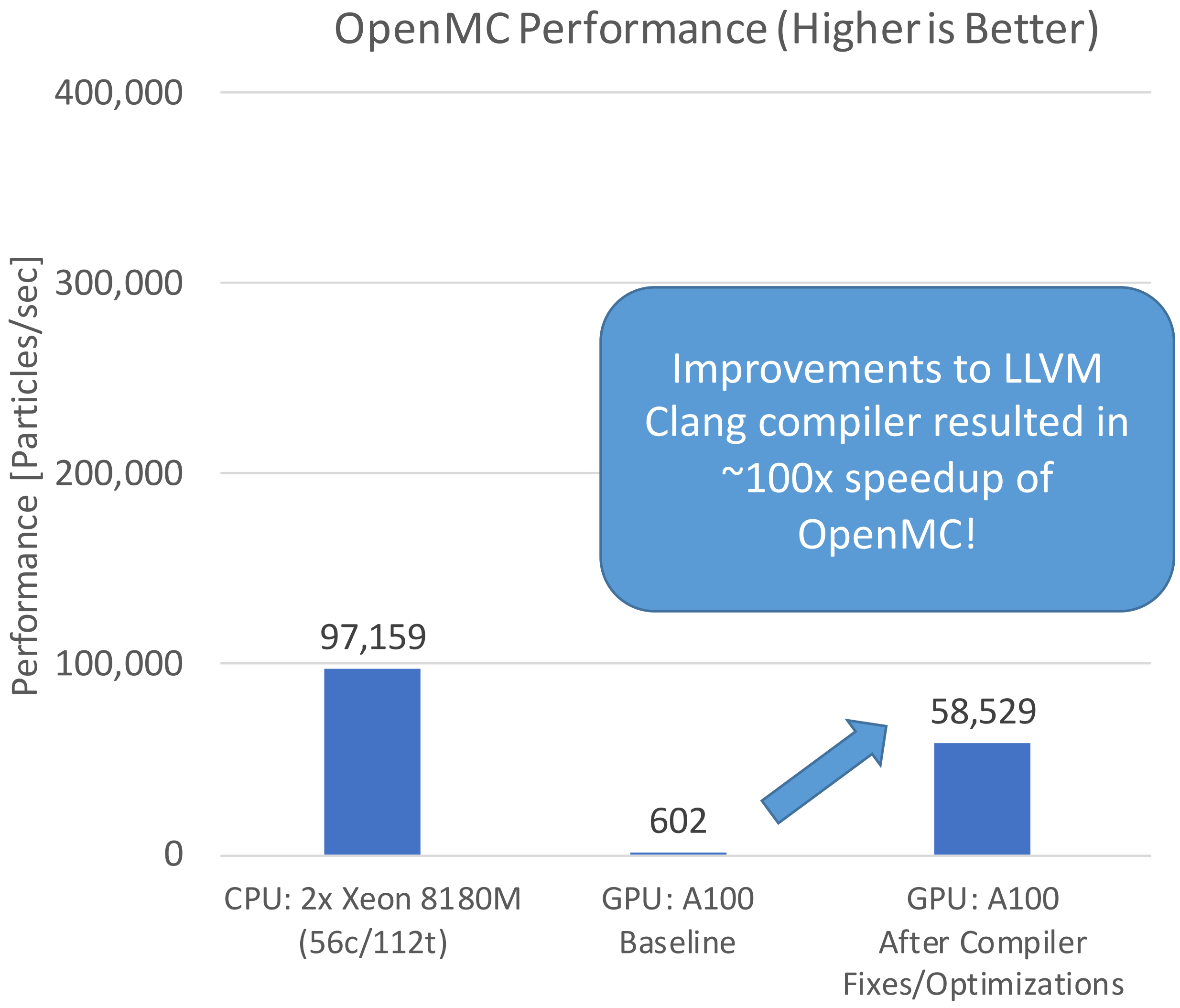
Compiler Issues

- LLVM was the first compiler that allowed us to at least get correct results
- However, performance at first was very poor...
- Close collaboration with LLVM compiler team (particular Johannes Doerfert) resulted in a several issues being identified (and promptly remedied!) in compiler
  - Extremely high costs for OpenMP `#pragma omp target update` clauses
  - Unnecessary globalization of stack variables
- `-fopenmp-cuda-mode` flag and use of a **cmake unity build** was very useful for improving performance as well
  - Although upcoming device link time optimization (LTO) capabilities in LLVM will make these steps unnecessary

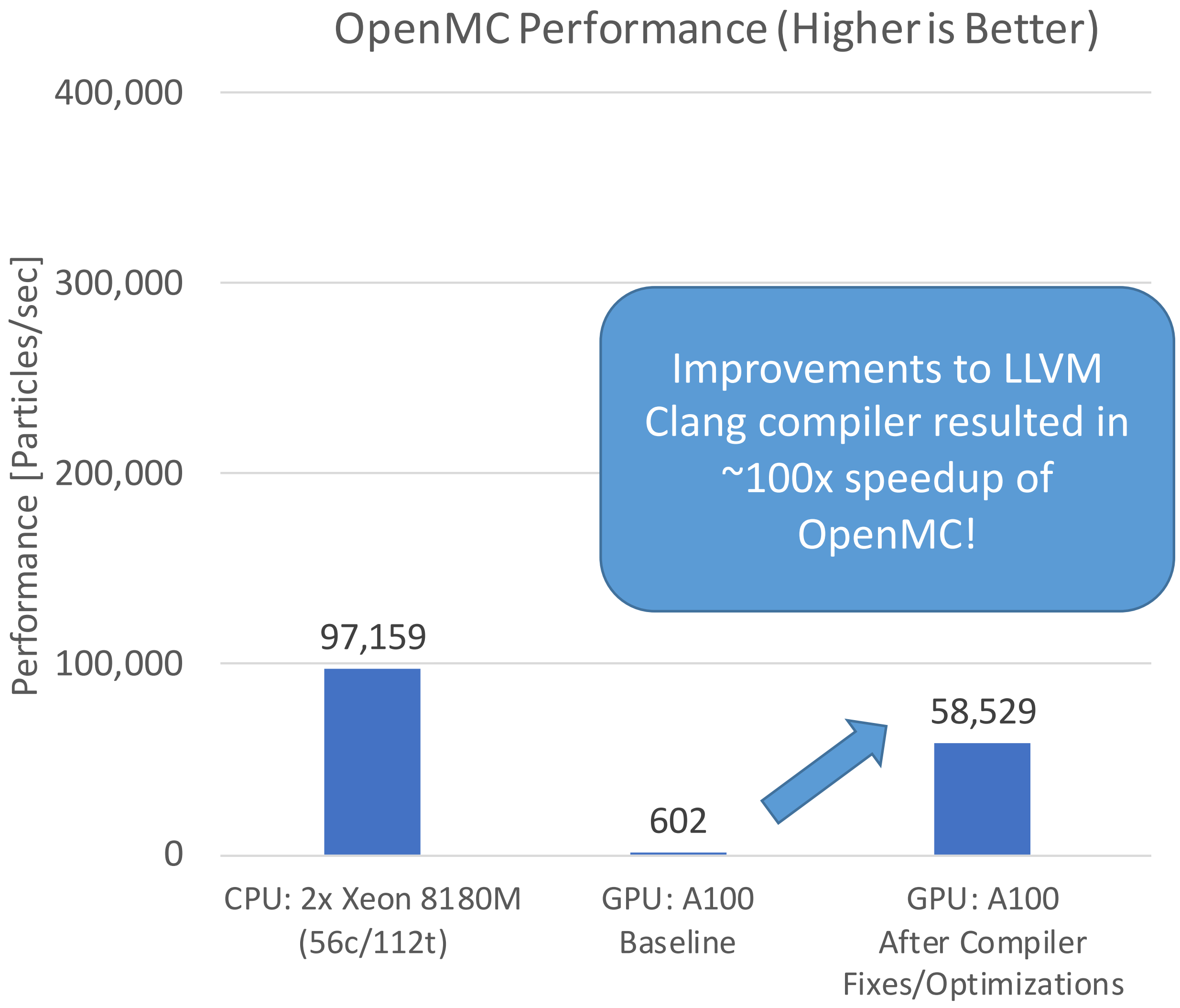
Results of Compiler Optimizations



Results of Compiler Optimizations



Results of Compiler Optimizations



GPU performance was now reasonable enough to begin real performance optimization work of the code.

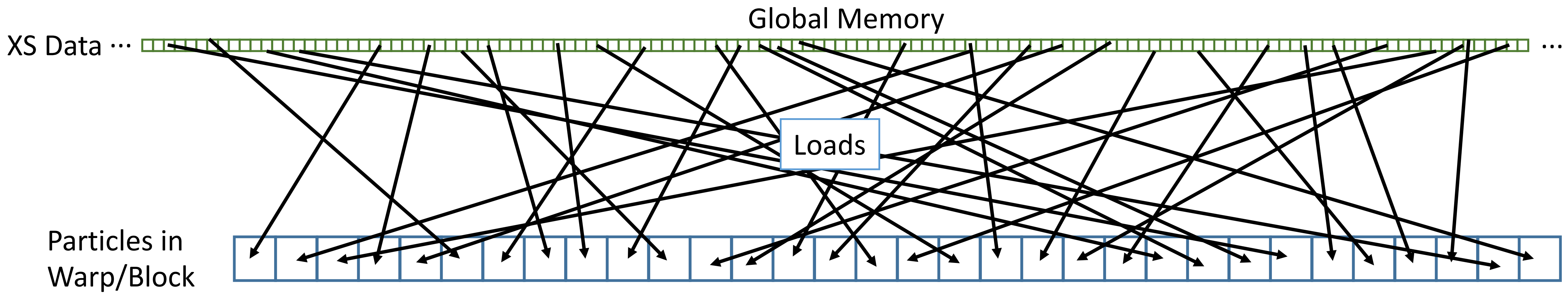
Application Optimization Highlights

- There were a variety of smaller optimizations each netting 5-10% that were all helpful
- The two biggest changes however were:
  - the ***removal*** of a legacy CPU-oriented optimization
  - sorting of kernel work items
- The above two changes worked together to massively boost performance!

Application Performance Breakthrough

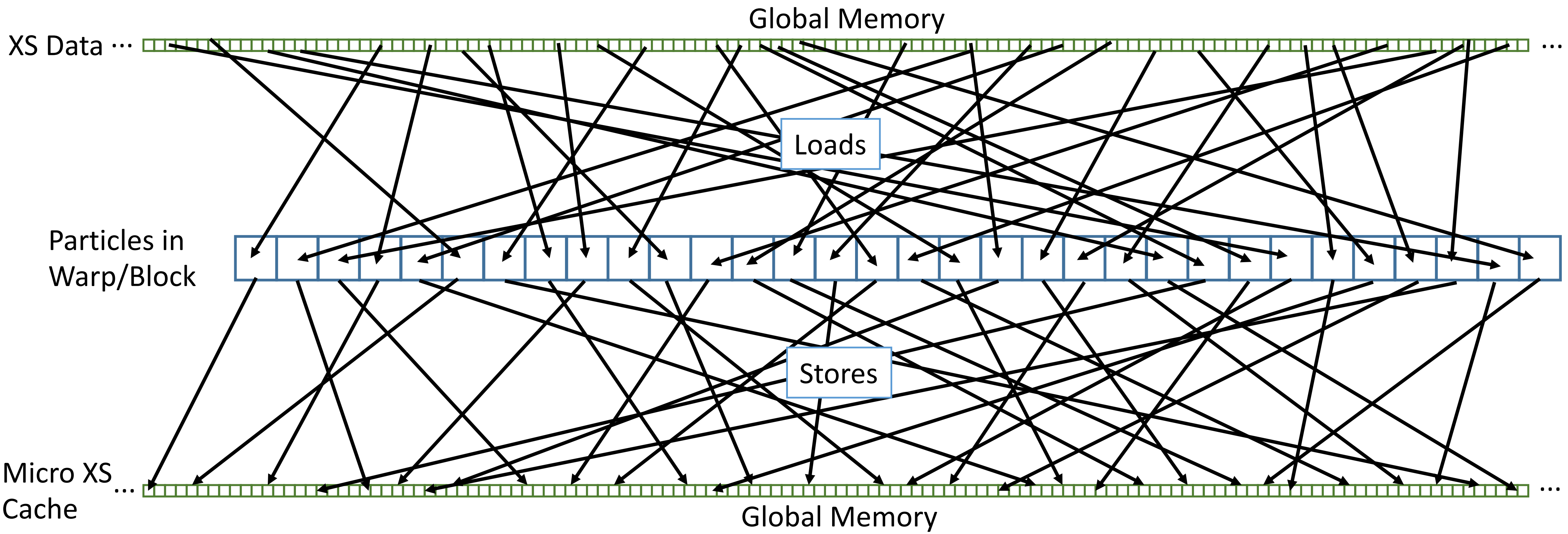
Removal of a legacy CPU-oriented optimization

XS Lookup Kernel



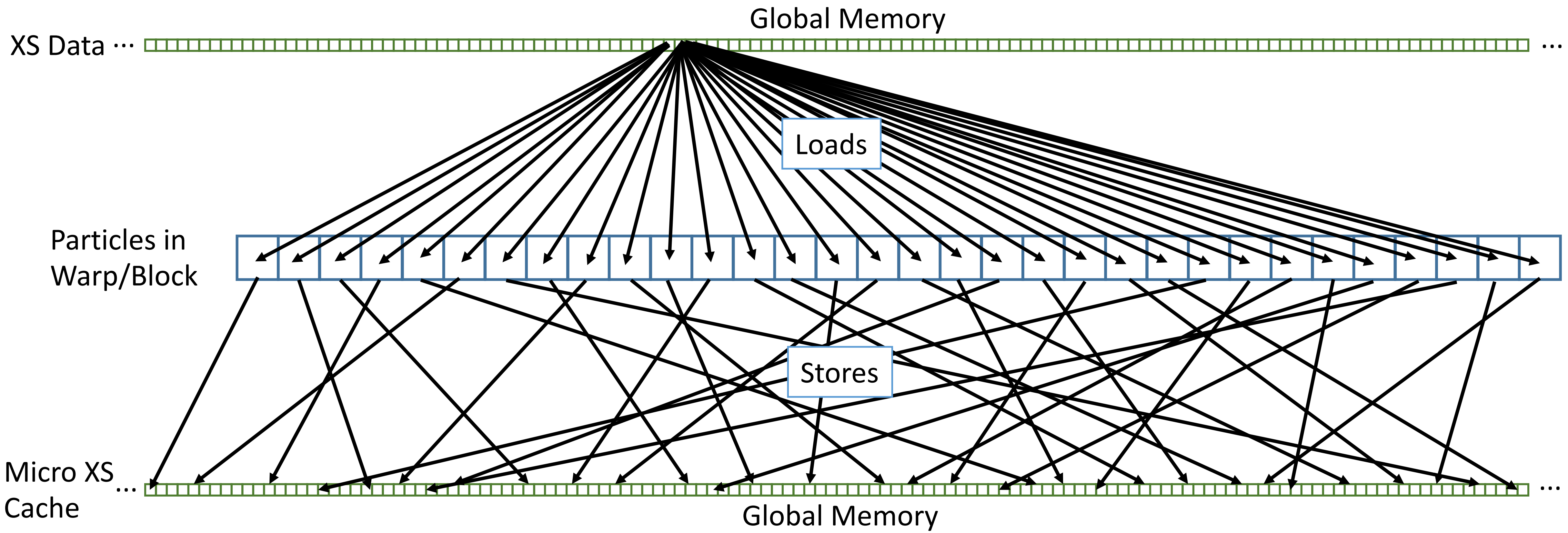
On CPU, addition of a software cache removes need to perform redundant search operations in XS Data grid, but this ends up being awkward on GPU

XS Lookup Kernel: With Micro XS Cache



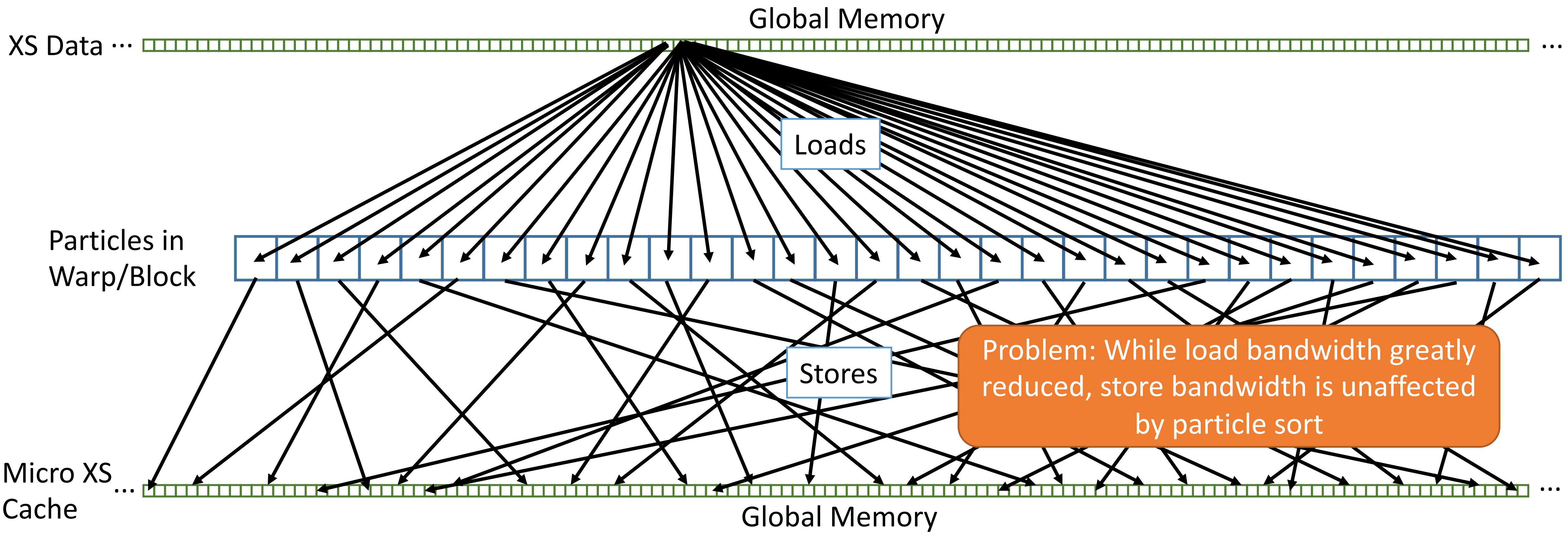
Cache may be re-used and will remove the need to search XS data arrays

XS Lookup Kernel: With Micro XS Cache, **Sorted**



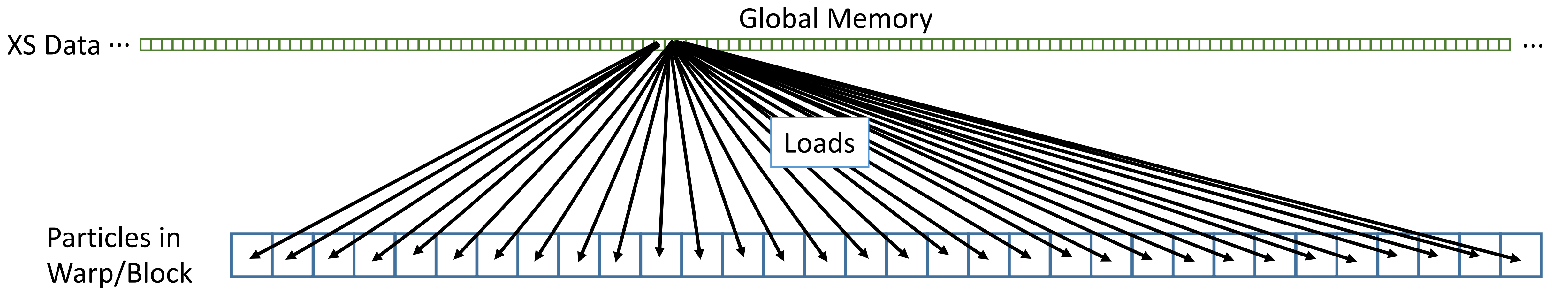
If we sort particles by energy before calling the kernel, all particles in a warp of 32 will access the same data

XS Lookup Kernel: With Micro XS Cache, **Sorted**



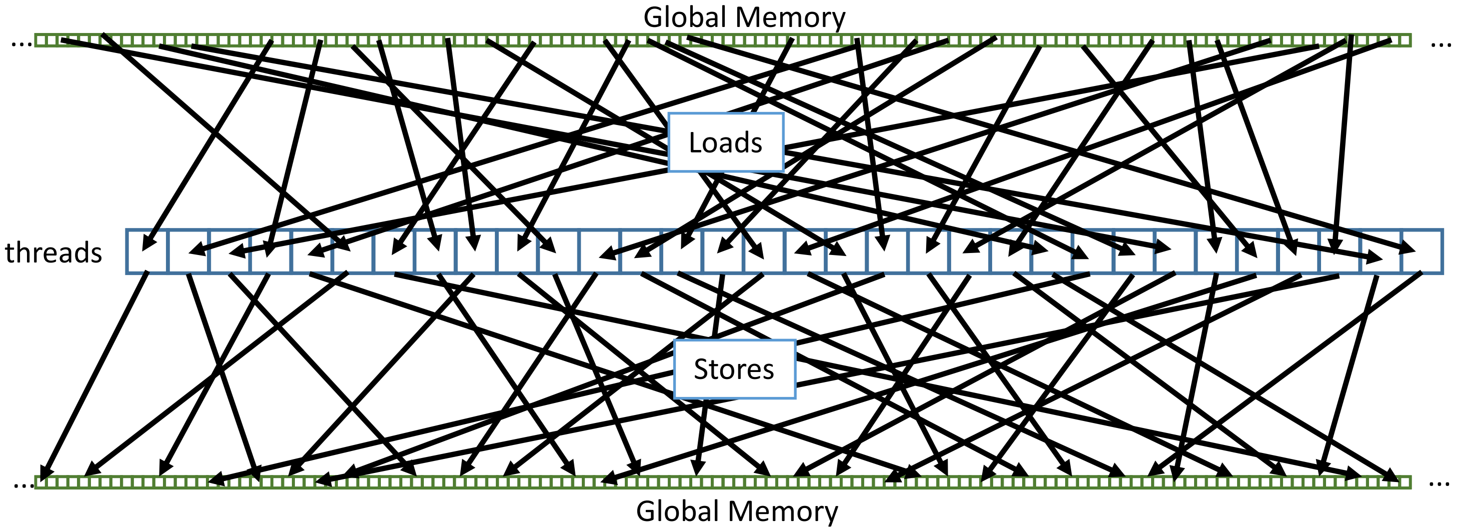
If we sort particles by energy before calling the kernel, all particles in a warp of 32 will access the same data

XS Lookup Kernel: **Without** Micro XS Cache, **Sorted**

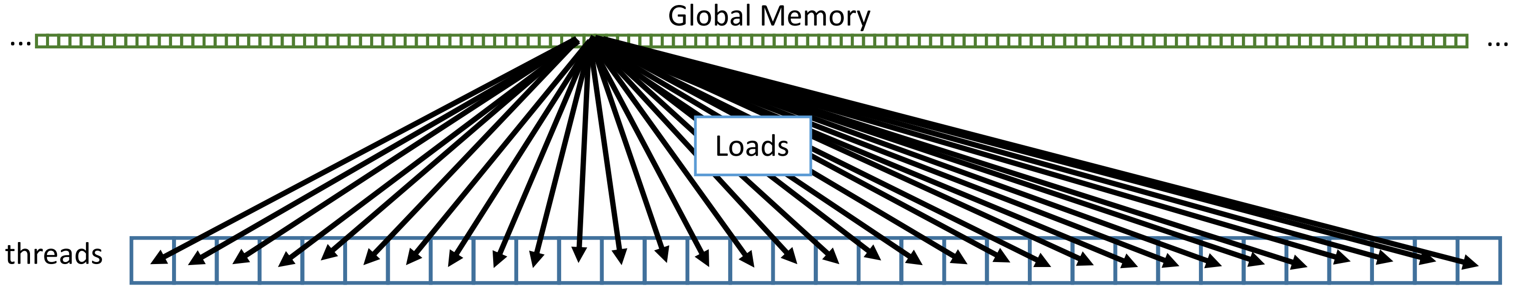


If we simply remove the XS cache, we have to perform more searching operations, but as all operations are shared between adjacent threads in a warp overall bandwidth is greatly reduced!

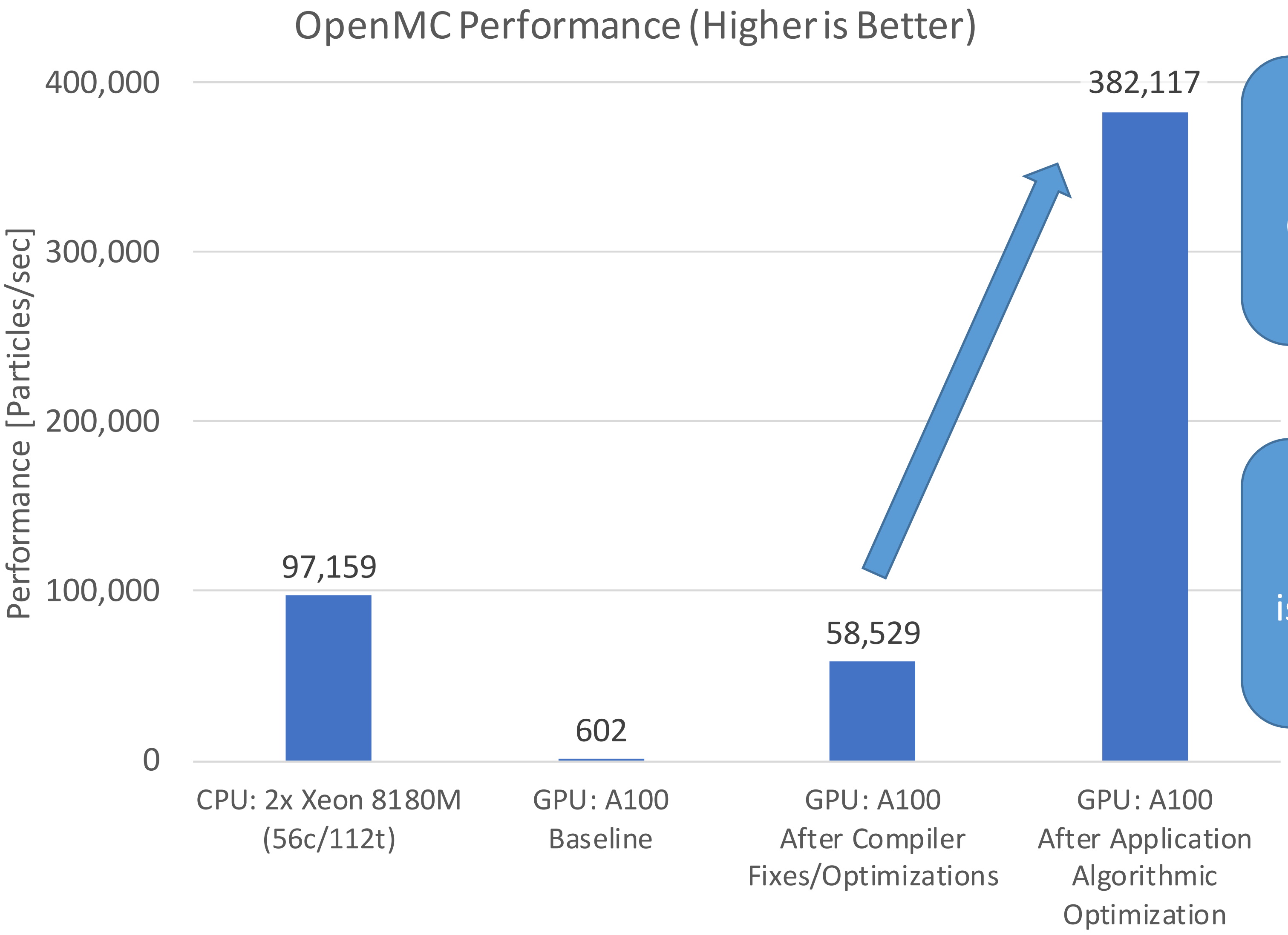
Original XS Kernel



Optimized XS Kernel



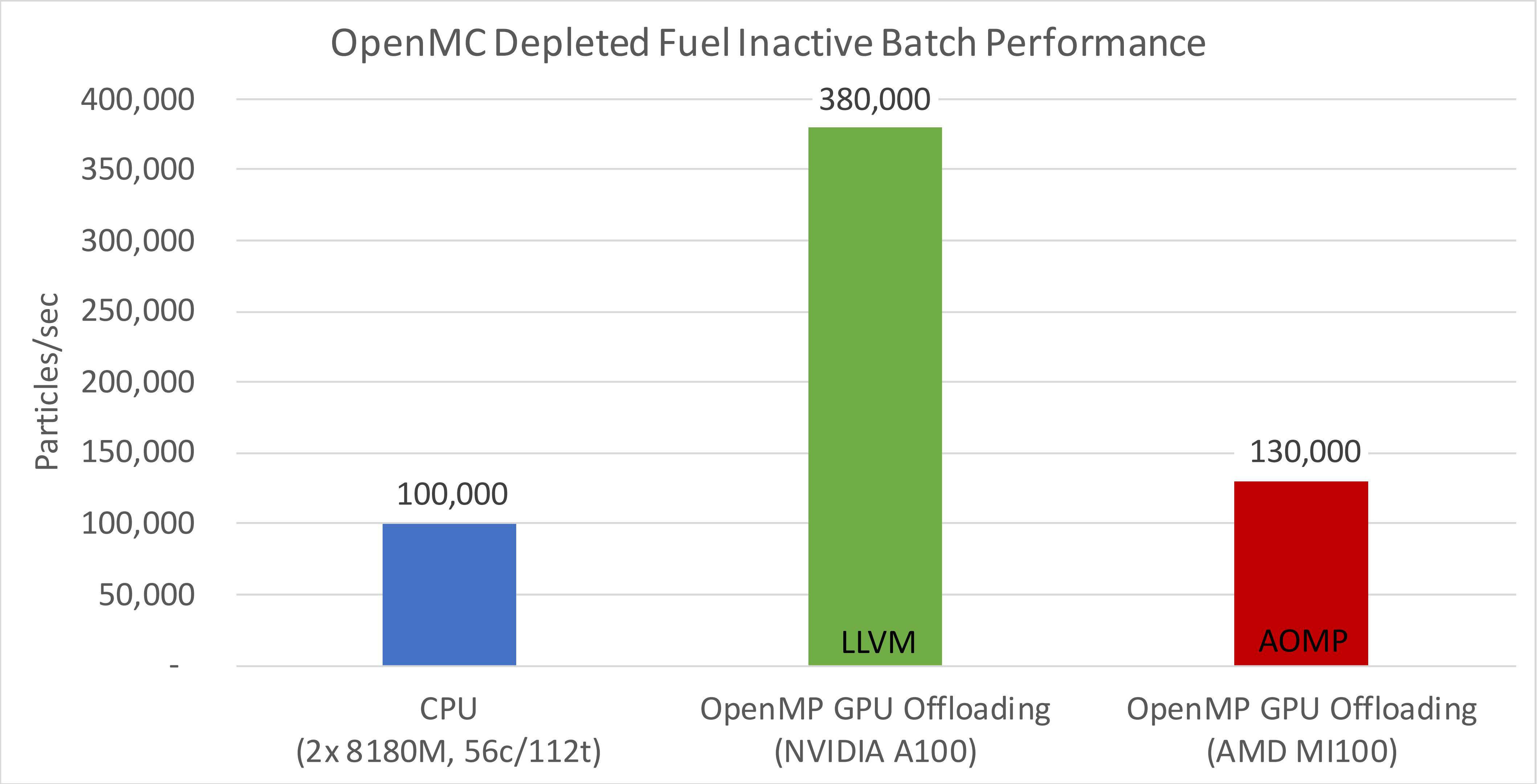
Final GPU Results



Algorithmic optimizations to OpenMC combined for a ~7x speedup!

OpenMC performance is now excellent on GPU

Performance Portability

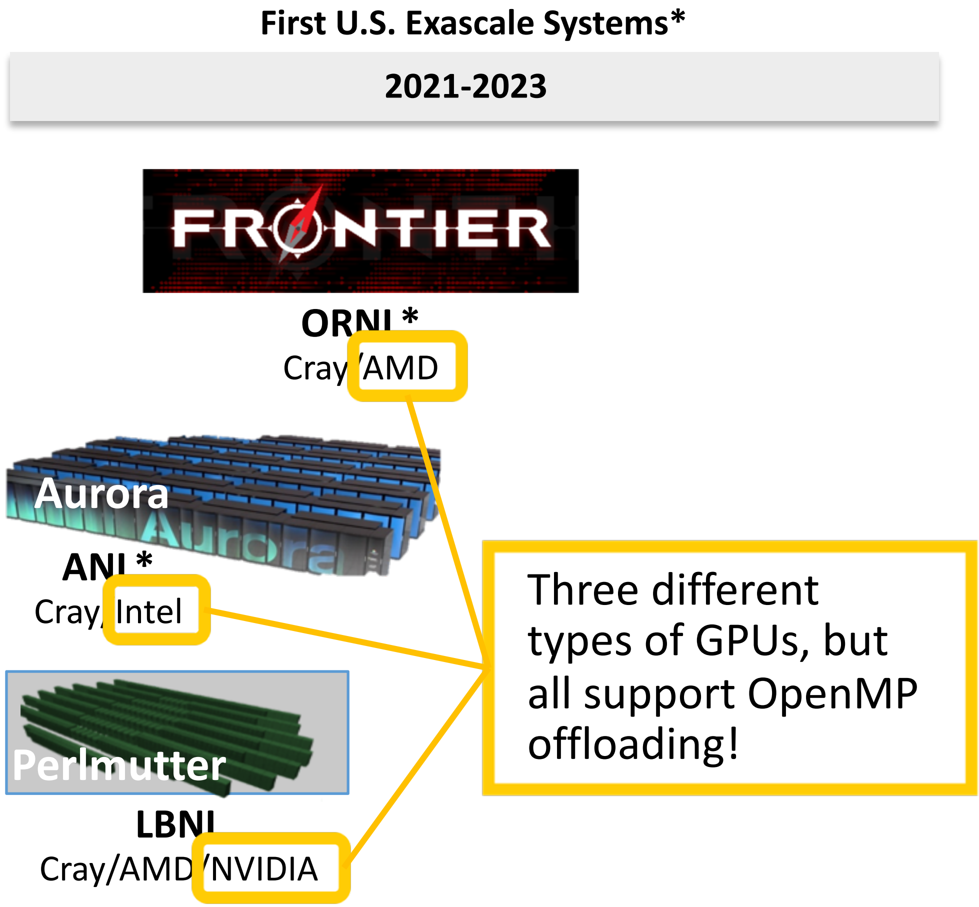


EXASCALE COMPUTING PROJECT

Argonne NATIONAL LABORATORY

Things I Really Like About OpenMP

- Ease of mapping data to device
  - Especially OpenMP 5.0 custom mappers
- Familiarity
  - Our development team already used to OpenMP threading model
- **<u>Portability</u>**
  - First party compiler support on NVIDIA, AMD, and Intel GPUs
  - LLVM Clang support for many GPU architectures
  - Highly unified CPU + GPU codebase with minimal/no "siloing"
- LLVM compiler team (particularly **Johannes Doerfert**) has been very responsive and able to make fixes quickly → allowed us to make rapid progress on performance optimization
- **<u>Excellent Performance</u>**



Things I Don't Like About OpenMP

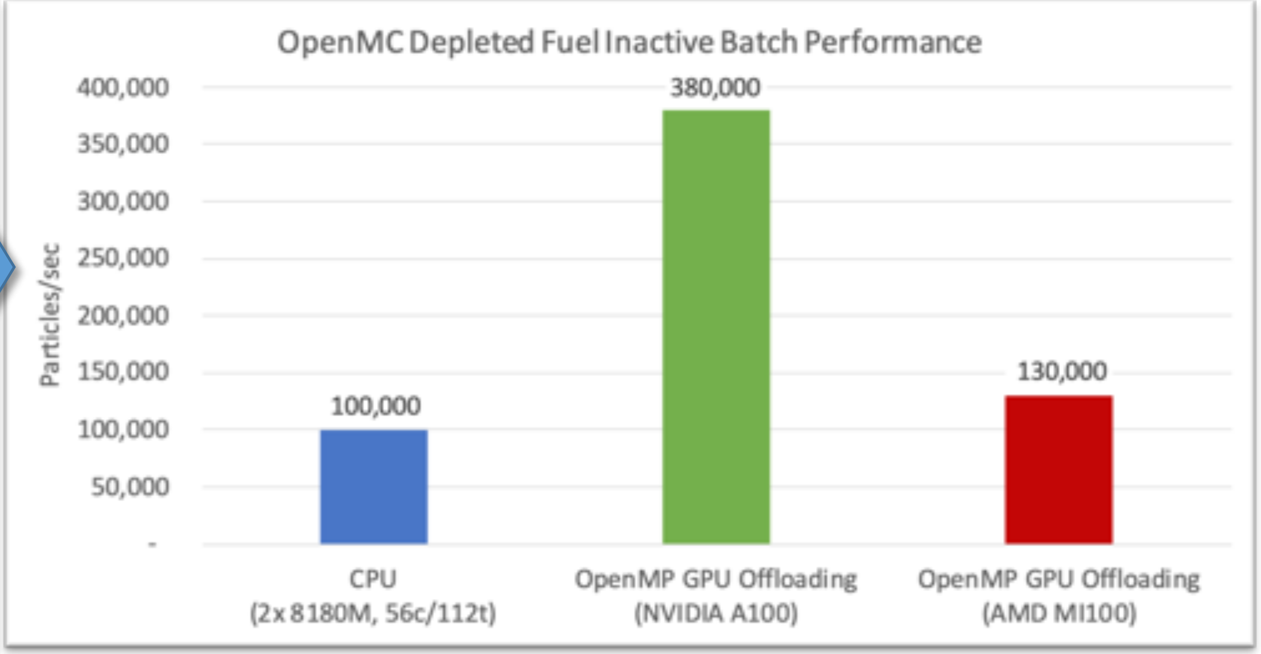
- New programming model, so in 2020 and 2021 compilers still had many bugs / unsupported features
  - In 2022, several compilers are rapidly approaching maturity!
  - Our Compiler ←→ Application codesign work will hopefully result in a smoother path for future apps teams

- No "baked-in" way to do on-device parallel sorts & scans
  - CUDA/HIP Thrust is sorely missed...
  - Possible route forward via `ompx` library?

Overall:
I find the pros of using OpenMP far outweighed the cons, and would highly recommend it to other GPU application teams.

Takeaways

- Despite initial struggles with compilers, LLVM Clang and AMD AOMP are now able to compile and run OpenMC.
- OpenMC showing impressive performance on both A100 and MI100.
- Co-maturation of OpenMC and LLVM Clang helped both teams make rapid progress
- We recommend OpenMP offloading model to other apps teams



Speaker: John Tramm (jtramm@anl.gov)

ECP EXASCALE COMPUTING PROJECT

Argonne NATIONAL LABORATORY

Acknowledgement

This presentation has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory (“Argonne”). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan. http://energy.gov/downloads/doe-public-access-plan.

30