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

John Tramm, PhD
Assistant Computational Scientist
Argonne National Laboratory

OpenMP Monthly Telecon – Feb 25, 2022
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
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
Porting to OpenMP: Main **Programming** Challenges

Original CPU-Oriented Code

- Virtual functions
- STL containers (e.g., std::vector)
- Nested, complex data structures

OpenMP Offloading GPU Port

- Tagged unions
- Pointers
- Tons of mapping code
Porting to GPU: Main *Algorithmic* Challenges

**Original CPU-Oriented Code**
- History-Based Algorithm
- Legacy CPU-Oriented Optimizations
- Unsorted

**OpenMP Offloading GPU Port**
- Event-Based Algorithm
- New, GPU-Oriented Optimizations
- Particle Sort in Energy

---

Argonne National Laboratory

ECP EXASCALE COMPUTING PROJECT
**Algorithmic Challenges**

**History-Based Transport Example**

- 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**

*Animation by Paul Romano*
History-Based Transport: Optimal for CPU

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

1: for each particle do  
   ▷ Independent
2:   while particle is alive do  
   ▷ Dependent
3:     Event A: Compute macroscopic cross sections
4:     Event B: Sample distance to collision and collision type
5:     Event C: Move particle to collision site
6:     Event D: Process particle collision
7:     Event ...
8:   end while
9: end for

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

1: initialize buffer of particles
2: while any particles are still alive do ▷ Dependent
3: for each alive particle do ▷ Independent
4: Event A: Compute macroscopic cross sections
5: end for
6: for each alive particle do ▷ Independent
7: Event B: Sample distance to collision and collision type
8: end for
9: for each alive particle do ▷ Independent
10: Event C: Move particle to collision site
11: end for
12: for each alive particle do ▷ Independent
13: Event D: Process particle collision
14: end for
15: for each alive particle do ▷ Independent
16: Event ...
17: end for
18: sort/consolidate surviving particles ▷ stream compaction
19: 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?

For MC, previous work suggests having $O(5\text{--}10)$ 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
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

OpenMC Performance (Higher is Better)

Performance [Particles/sec]

400,000
300,000
200,000
100,000
0

CPU: 2x Xeon 8180M (56c/112t)
GPU: A100 Baseline

Baseline
After Compiler Fixes/Optimizations
After Application Algorithmic Optimization

97,159
602
Results of Compiler Optimizations

Improvements to LLVM Clang compiler resulted in ~100x speedup of OpenMC!

Performance [Particles/sec]

OpenMC Performance (Higher is Better)

- CPU: 2x Xeon 8180M (56c/112t)
- GPU: A100 Baseline
- GPU: A100 After Compiler Fixes/Optimizations

97,159
602
58,529
Results of Compiler Optimizations

Improvements to LLVM Clang compiler resulted in ~100x speedup of OpenMC!

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

OpenMC Performance (Higher is Better)

- **CPU:** 2x Xeon 8180M (56c/112t)
- **GPU:** A100

Baseline

- **Performance [Particles/sec]:** 602

After Compiler Fixes/Optimizations

- **Performance [Particles/sec]:** 58,529
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
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
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.

Problem: While load bandwidth greatly reduced, store bandwidth is unaffected by particle sort.

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

OpenMC Performance (Higher is Better)

- CPU: 2x Xeon 8180M (56c/112t)
- GPU: A100 Baseline
- GPU: A100 After Compiler Fixes/Optimizations
- GPU: A100 After Application Algorithmic Optimization

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

OpenMC performance is now excellent on GPU
Performance Portability

OpenMC Depleted Fuel Inactive Batch Performance

- CPU (2x 8180M, 56c/112t) 100,000
- OpenMP GPU Offloading (NVIDIA A100) 380,000
- OpenMP GPU Offloading (AMD MI100) 130,000

Particles/sec
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

- **Portability**
  - 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

- **Excellent Performance**
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.
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)
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.