LA-UR-20-25948
Approved for public release; distribution is unlimited.

Title: Integration of Kokkos into MonteRay

Author(s): Kleedtke, Samantha
Tangtartharakul, Kavin
Burke, Timothy Patrick
Sweezy, Jeremy Ed

Intended for: Report

Issued: 2020-08-05

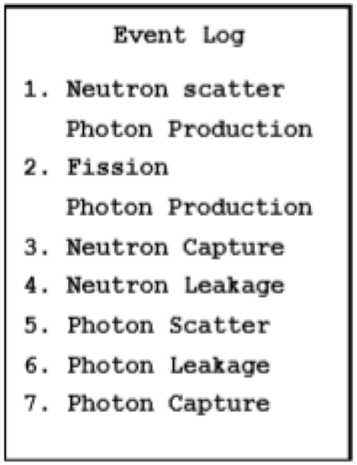
Integration of Kokkos into MonteRay

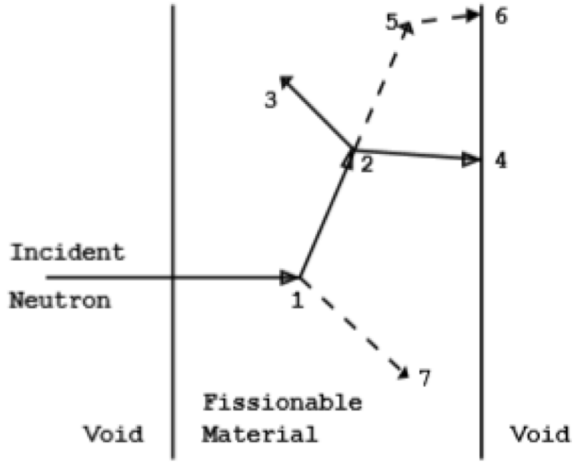
Samantha Kleedtke and Kavin Tangtartharakul

Mentors: Jeremy Sweezy and Timothy Burke

Monte Carlo Codes

- C++ based Monte Carlo Neutron Transport Code simulates the interaction of nuclear particles with materials
- follows the path of individual particles through a system
- events are simulated sequentially and then statistically sampled





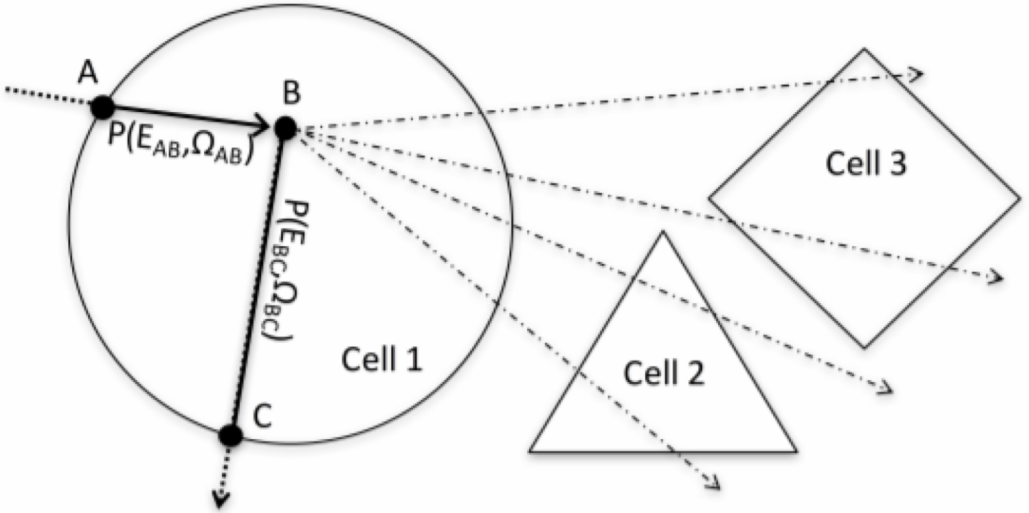
Briesmeister, Judith F. “MCNP -A General Monte Carlo N-Particle Transport Code.” 2000. Version 4C. LA-13709-M.

MonteRay Library

- MonteRay is a library for accelerating Monte Carlo ray-casting tallies with GPUs:
  1. Expected Path Length Estimator
  2. Next Event Estimators
  - written using Cuda and C++
  - the CPU performs the random walk while the GPU concurrently calculates tallies
  - coupled with Monte Carlo Application Toolkit (MCATK)

Expected Path Length

The Expected Path Length determines the attenuated path length of the emitted particle. Multiple outgoing rays can be sampled from each collision event and each ray can score to multiple cells.



Sweezy, Jeremy E. “A Monte Carlo Volumetric-Ray-Casting Estimator for Global Fluence Tallies on GPUs.” Journal of Computational Physics 372 (2018): 426–445. Crossref. Web.

Kokkos and Performance Portability

- Kokkos is commonly used to write performance portable applications for HPC platforms
- programming model in C++
- provides abstractions for parallel execution of code and data management
- goal is to implement Kokkos into the Expected Path Length file that uses Cuda as a backend

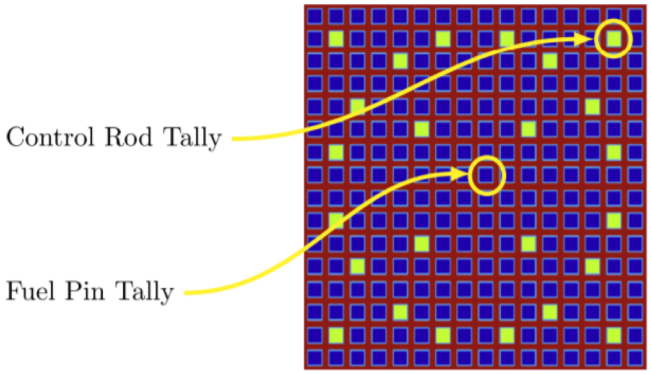
https://github.com/kokkos/kokkos

Edwards, H. Carter, Trott, Christian R., Sunderland, Danial. “Kokkos:Enabling manycore performance portability through polymorphic memory access patterns.” Journal of Parallel and Distributed Computing 74 (2014):3202–3216.

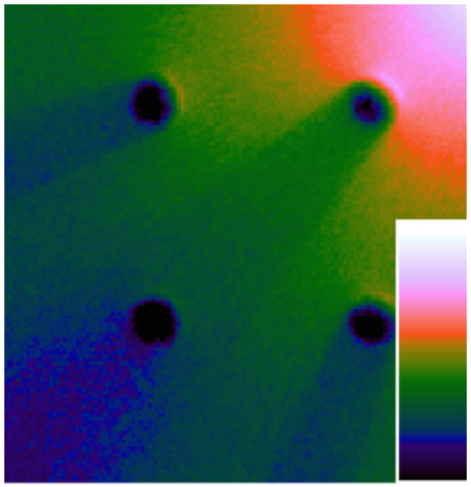
Methodology

1. Analyzed the times building and running the “nightly” tests without Kokkos (solely used the CPU)
2. Integrated Kokkos and its parallel patterns into the Expected Path Length tally
3. Debugged our pattern’s execution policy to ensure tests passed
4. Compared the performance and determined the benefits and drawbacks of portability via Kokkos

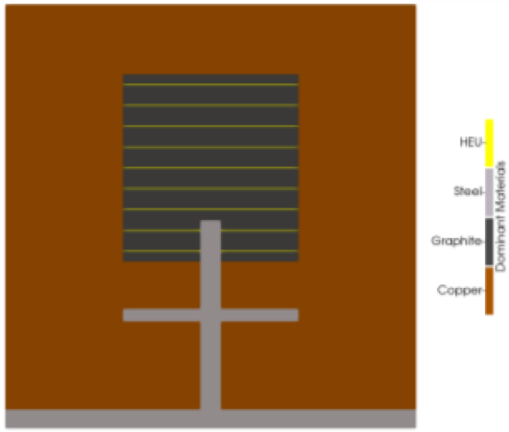
“nightly” Test Suite



The Assembly test simulates a nuclear reactor with fuel and control rods.



The Criticality Accident test simulates the top right corner of a reactor reaching super-critical reactivity.



The Zeus test simulates platters of highly enriched uranium, graphite, and copper reflectors stacked.

Sweezy, Jeremy E. “A Monte Carlo Volumetric-Ray-Casting Estimator for Global Fluence Tallies on GPUs.” Journal of Computational Physics 372 (2018): 426–445. Crossref. Web.

Results: Total Times

- after we implemented Kokkos into the Expected Path Length estimator

	Serial	Original (just CUDA)	Kokkos (Added)
Assembly test	497.36s	24.23s	47.06s
Criticality test	593.89s	52.19s	128.03s
Zeus test	60.19s	11.50s	24.93s

Results: Assembly Test

- GPU times are similar after Kokkos was added
- CPU times are slower after Kokkos was added

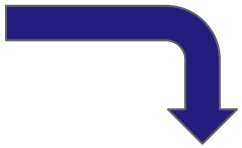
	Serial	Original (just CUDA)	Kokkos Added
CPU Time	148.88 s	5.05 s	8.03 s
GPU Time	0 s	1.13 s	1.15 s

- criticality and zeus tests show similar results
  - GPU times are similar, CPU times have increased

Obstacles Encountered

- initial attempt: Kokkos was unable to find CUDA and was running the Expected Path Length in serial

```
cmake ../../monteray -Denable_cuda=ON -
DCMAKE_BUILD_TYPE=Release -Dcuda_arch=Volta -
DCMAKE_INSTALL_PREFIX=$PWD/install
```



```
cmake ../../monteray -Denable_cuda=ON -DCMAKE_BUILD_TYPE=Release -
DCMAKE_INSTALL_PREFIX=$PWD/SandBoxRelease -DKokkos_ENABLE_CUDA=ON -
DCMAKE_CXX_COMPILER=nvcc_wrapper -DKokkos_ARCH_VOLTA70=ON -
DKokkos_ENABLE_CUDA_LAMBDA=ON -DKokkos_ENABLE_CUDA_CONSTEXPR=ON -
DKokkos_ENABLE_CUDA_RELOCATABLE_DEVICE_CODE=ON -DKokkos_ENABLE_CUDA_UVM=ON
```

- Kokkos::finalize() caused segmentation faults

Conclusion

- deleted Kokkos::finalize() in order to fix the segmentation faults
- with Kokkos added:
  - GPU times were about the same
  - CPU times were slower
- integrating Kokkos into the Expected Path Length file was not too difficult
- converting the build system to use Kokkos was more work than intended
  - bugs in Kokkos' build system, both master and devel branches
  - if the code was initially written with Kokkos in mind, things would have went more smoothly

Questions?