More Than a Port:
Challenges in GPU-Based Monte Carlo and Method of Characteristics Particle Transport

ANS Annual Virtual Meeting 2021

Monday, June 14

Speaker: John Tramm

ECP EXASCALE COMPUTING PROJECT

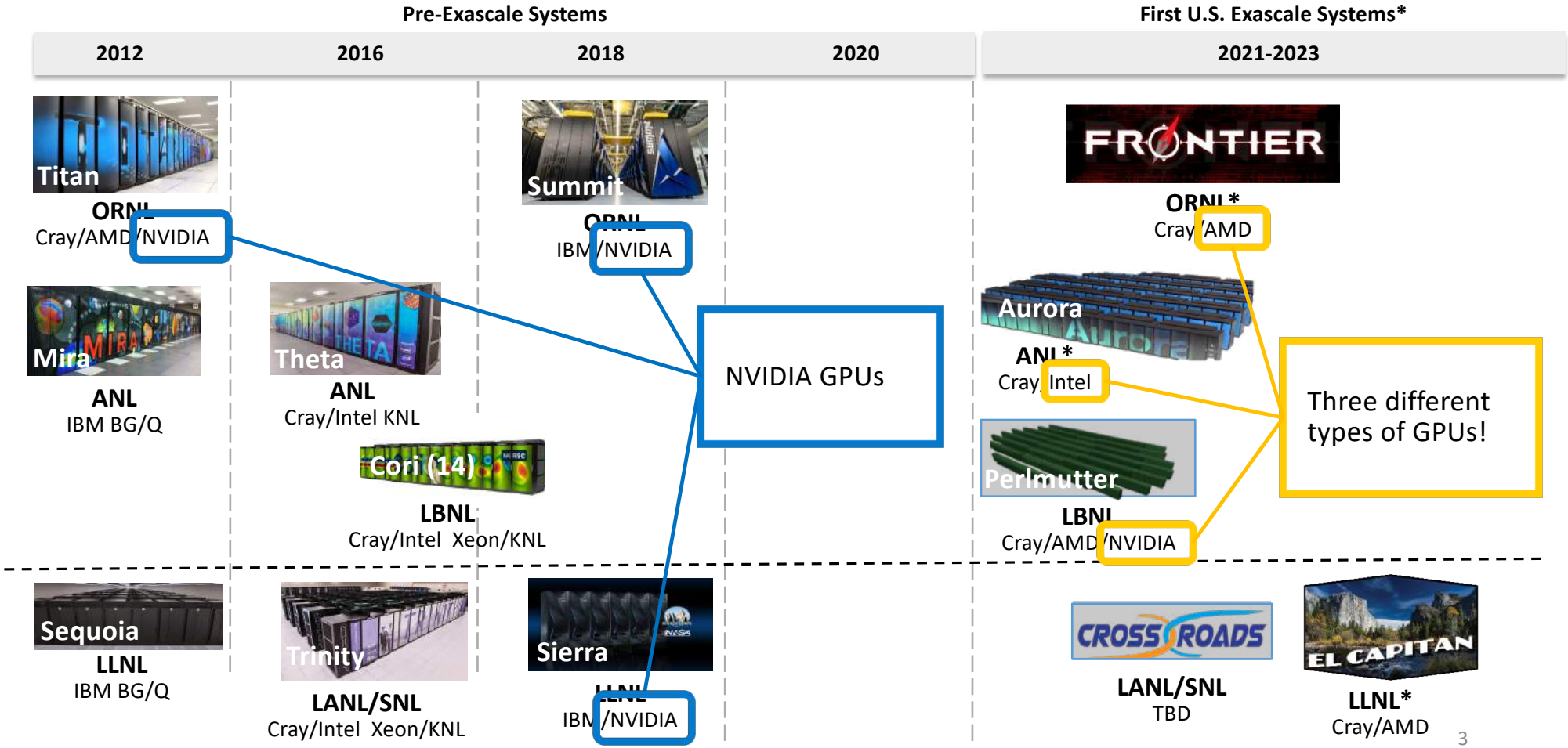
Argonne NATIONAL LABORATORY

Overview

- Why GPUs matter and impact on field of numerical simulation
- General porting and algorithmic challenges for GPU
- Case studies in porting real applications to GPU:
  - OpenMC (Monte Carlo) w/ OpenMP target offloading
  - ARRC (Random Ray Method of Characteristics) w/ CUDA

GPUs Are Taking Over

Department of Energy (DOE) Roadmap to Exascale Systems



Problem: New Programming Model Required

- Traditional CPU-based supercomputers:
  - MPI
  - MPI + OpenMP Threading
- GPU-based supercomputers:
  - MPI + X (where "X" is ***not*** OpenMP Threading)

Four basic choices of GPU programming model

GPU-specific kernels

- Isolate the computationally-intensive parts of the code into CUDA/HIP/SYCL kernels.
- Refactoring the code to work well with the GPU is the majority of effort.

Loop pragma models

- Offload loops to GPU with OpenMP target offloading or OpenACC.
- Most common portability strategy for Fortran codes.

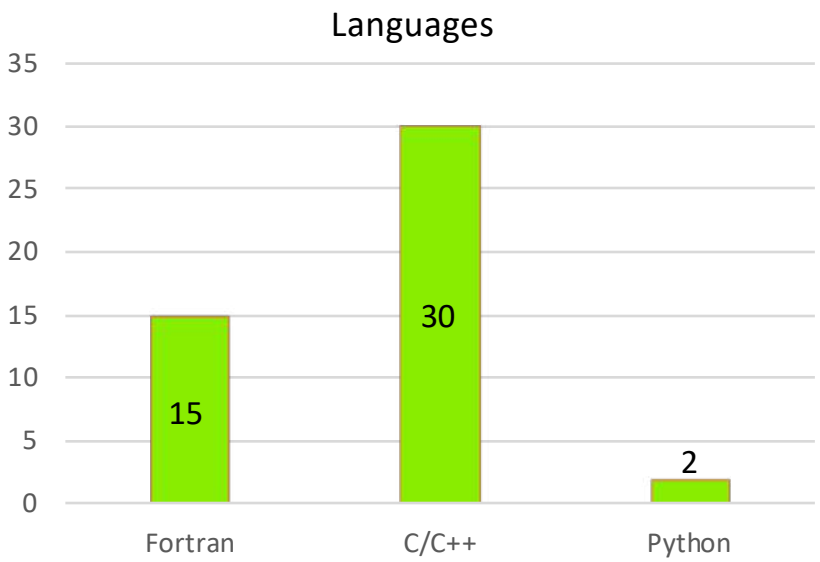
C++ abstractions

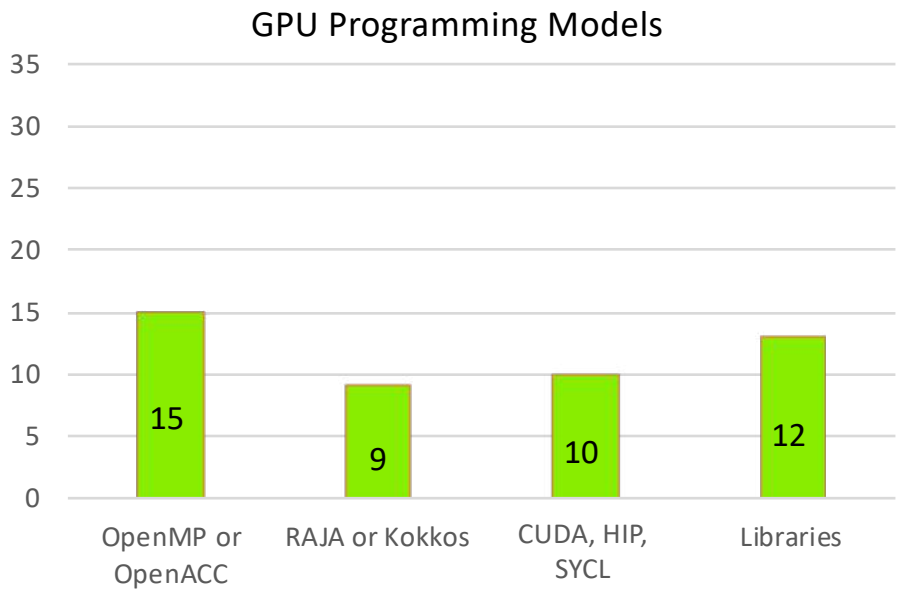
- Fully abstract loop execution and data management using advanced C++ features.
- Kokkos and RAJA libraries developed by NNSA in response to increasing hardware diversity.

Co-design frameworks

- Design application with a specific motif to use common software components
- Depend on co-design code (e.g. CEED, AMReX) to implement key functions on GPU.

Exascale Compute Project (ECP) applications use a mix of languages and programming models





Many codes are still in flux, with quite a few still deciding on a final programming model. A few Fortran codes are being rewritten in C++, but most are not.

GPU Programming & Porting Challenges

Memory Management

- Manual management of device memory space and host ←→ device communication
- Unified Shared Memory (USM) can help in some cases, but not in many others!

C++ Restrictions on Device

- No virtual functions or polymorphism
- No C++ STL usage
- No dynamic memory allocation

Lack of Portability

- Historically, GPU market dominated by NVIDIA and proprietary CUDA model
- Not all open standards supported at the same level by all vendors

Changes in Parallelism

- Some models require strict kernelization rather than "for loop" based parallelism
- Some algorithms may require management of shared resources (e.g., shared local memory)

GPU Algorithmic Challenges

Increased Parallelism

- Huge parallelism required to saturate a GPU
- Challenging for strong scaling

Memory Management

- Need to minimize host ←→ device data movement
- Typically less memory available on GPU

Thread Divergence

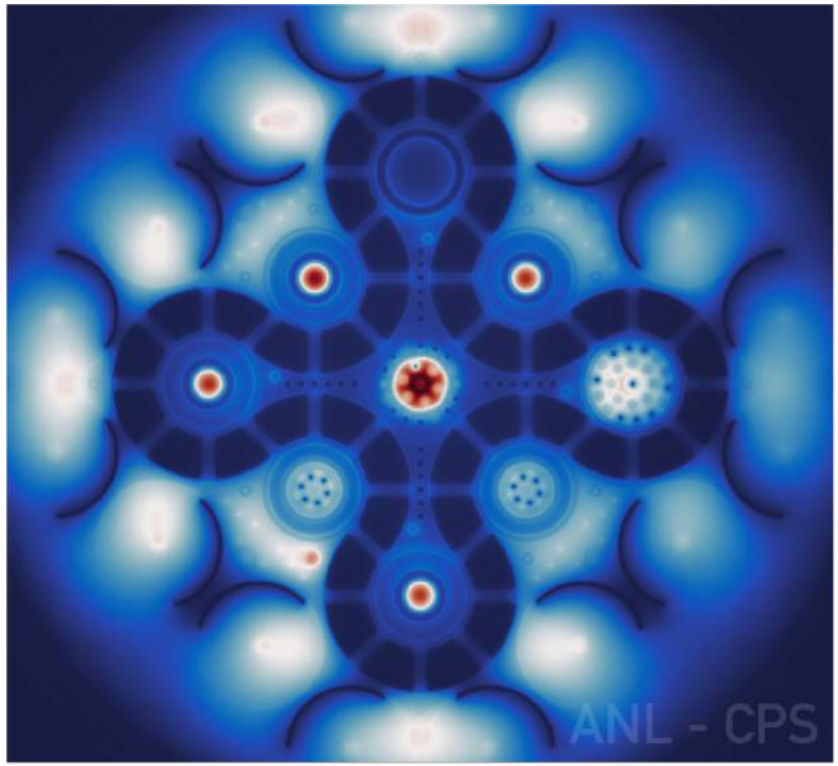
- SIMD
- Threads in a block/team/group should be doing same task

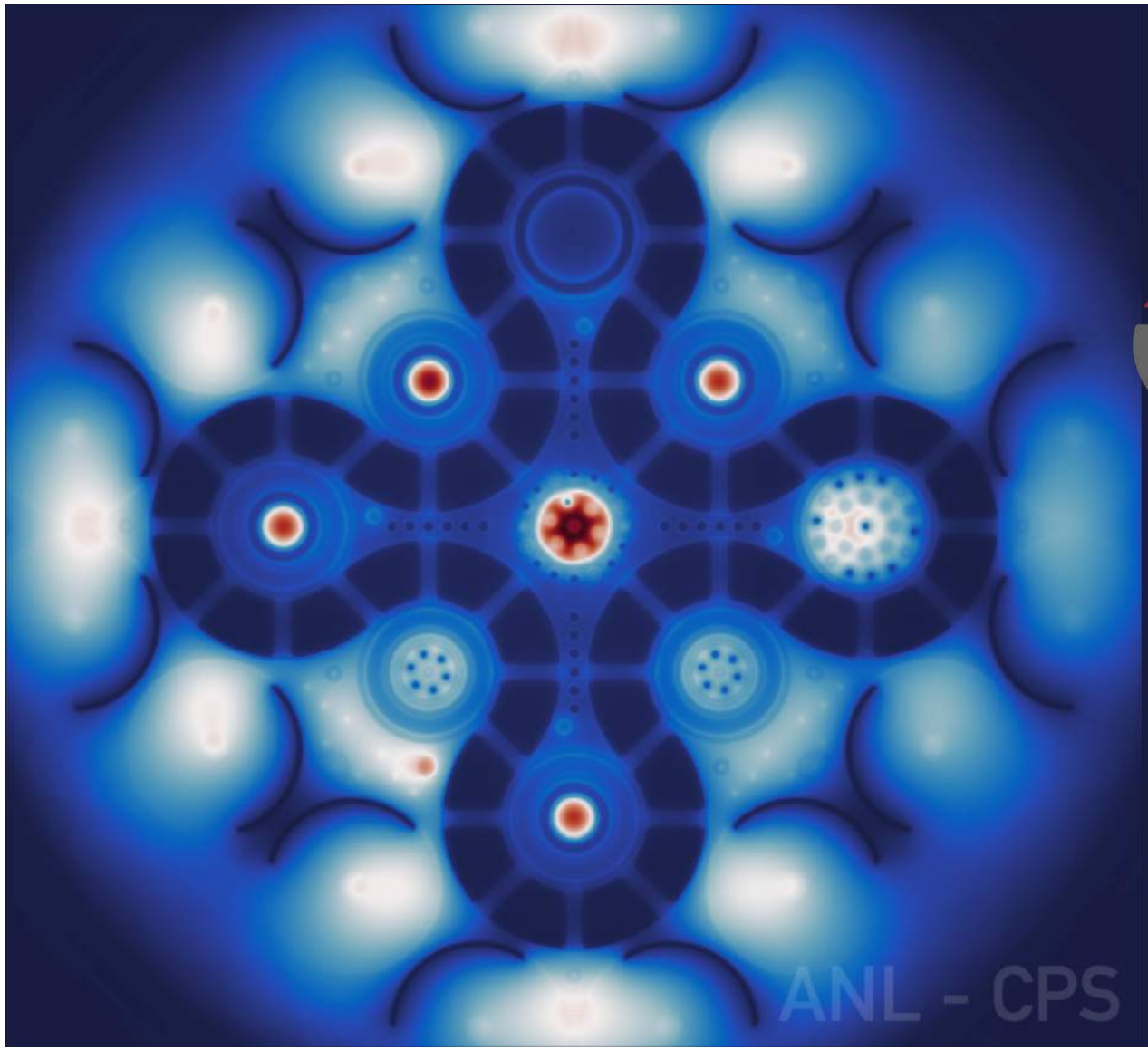
Mixed Precision

- FP32, FP16 offer performance gains on GPU
- Numerical innovations may be needed to ensure stability

Case Studies

So:
How bad was it to get
OpenMC
running on GPU?





OpenMC

- Monte Carlo continuous energy particle transport
- Language: Modern C++
- Parallelism: MPI + OpenMP Threading

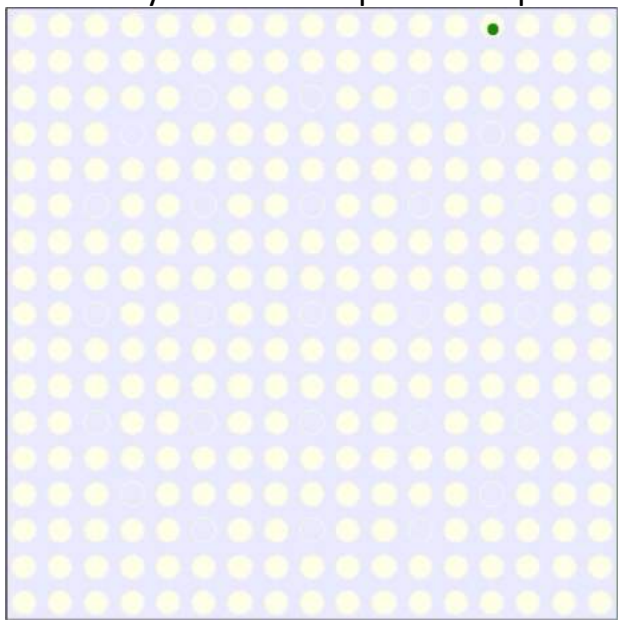
Argonne NATIONAL LABORATORY

Porting Challenges

- GPU port uses MPI + **OpenMP Target Offloading**
- Personally ran into, identified, and reported **at least 23 compiler bugs!**
- Modern C++ problems:
  - virtual functions not allowed* on device
  - standard template library usage (STL) not allowed on device
- Laborious to manually map deep hierarchical data structures to device memory
  - Easy to make mistakes and hard to debug

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

"Event-Based" Parallelism

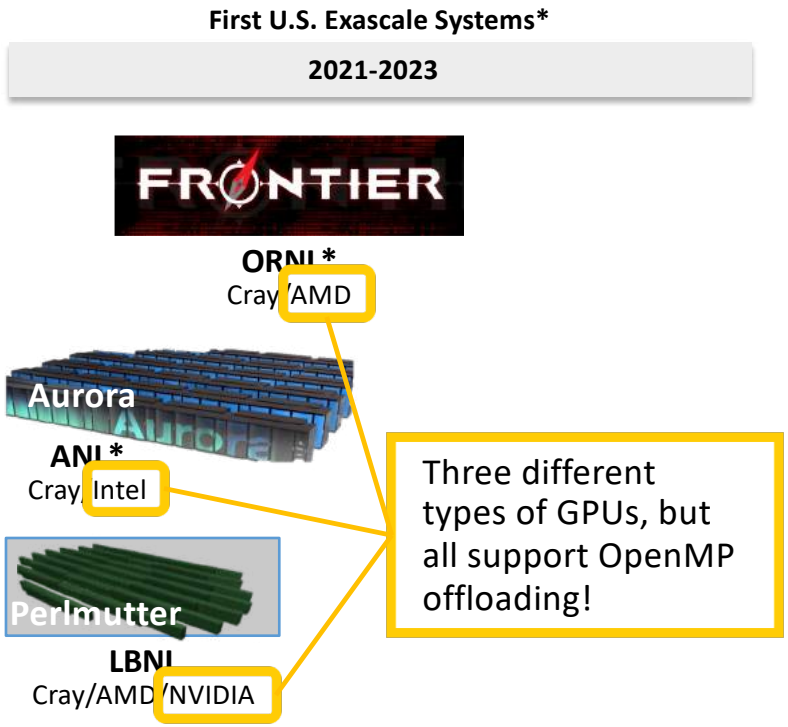
- Originally developed in the 80's for vector computers
- Only execute one low level event type at a time (kernel splitting)
- Parallelism expressed over particles requiring that event
- Greatly reduces thread divergence

Other Algorithmic Optimizations

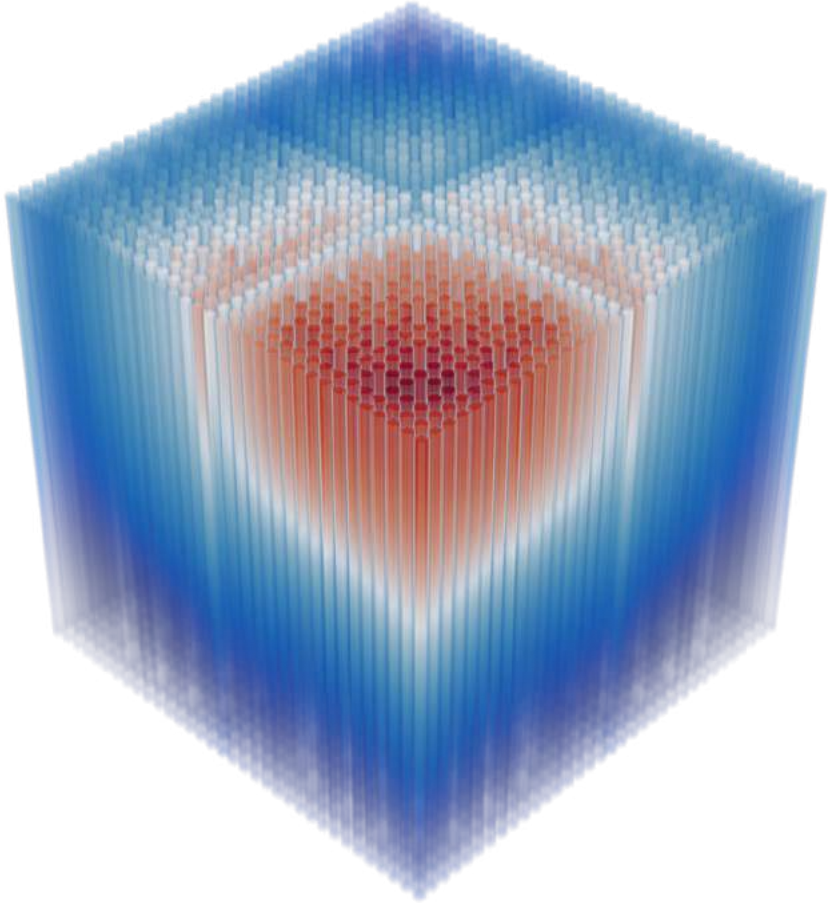
- Sorting (by material and/or particle energy)
- SOA vs. AOS
- Efficient shared memory cell neighbor lists

OpenMC Port Epilogue

- OpenMP 5.0 target offloading is an extremely new programming model → Lots of compiler bugs!
- However, all exascale systems will support this model
- A lot of prior work on Monte Carlo for GPU, so we had a good roadmap for what optimizations are needed



So:
How bad was it to get
ARRC
running on GPU?

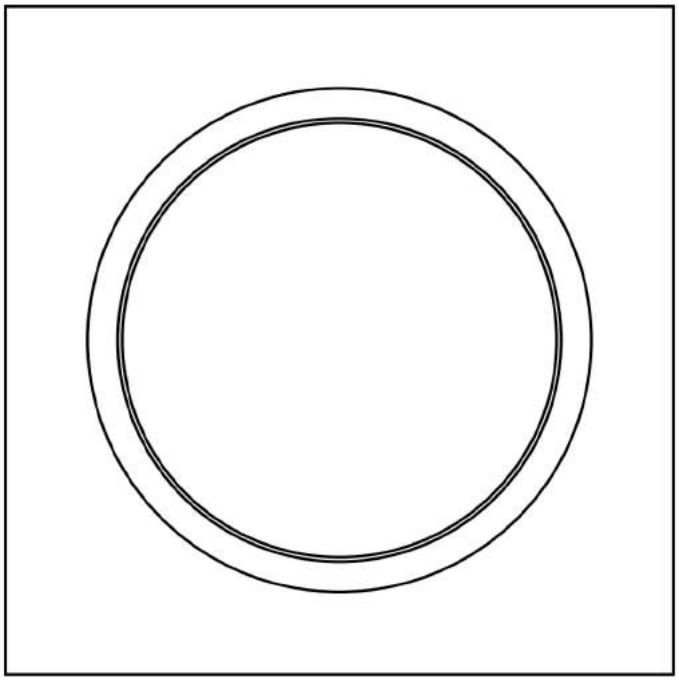


EXASCALE COMPUTING PROJECT

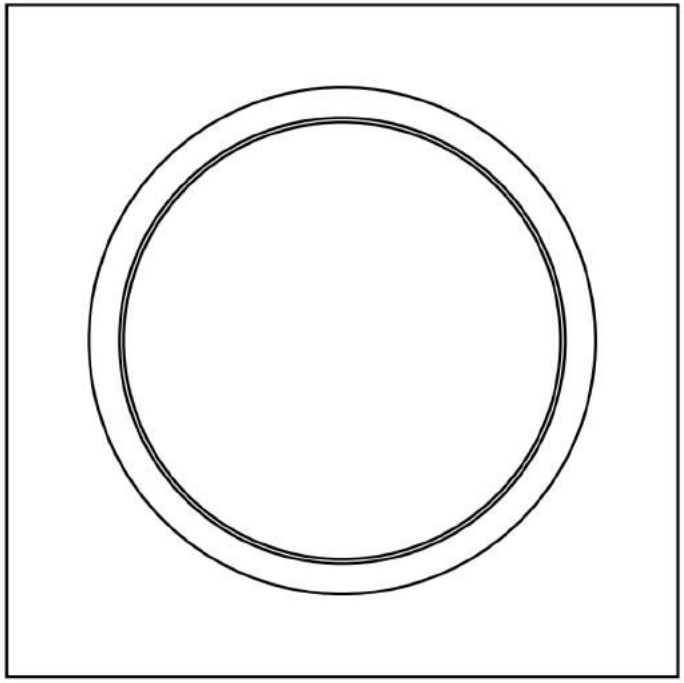
Argonne NATIONAL LABORATORY

Brief Background:
ARRC is an application that uses the random ray method

Traditional Deterministic MOC



Random Ray

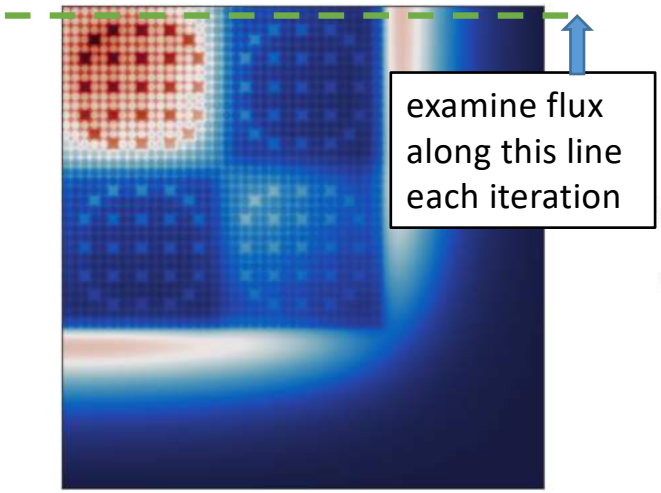


Advantages compared to traditional deterministic MOC:

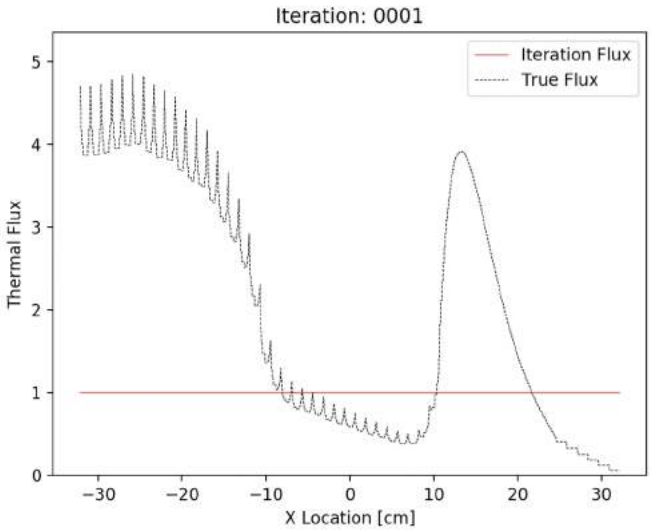
- No storage of angular fluxes at boundaries:
  - 50-1000x less memory needed
- Low ray density required to reach stationary source distribution:
  - 50-500x less work per iteration

Deterministic vs. Random Ray: Real Time Comparison

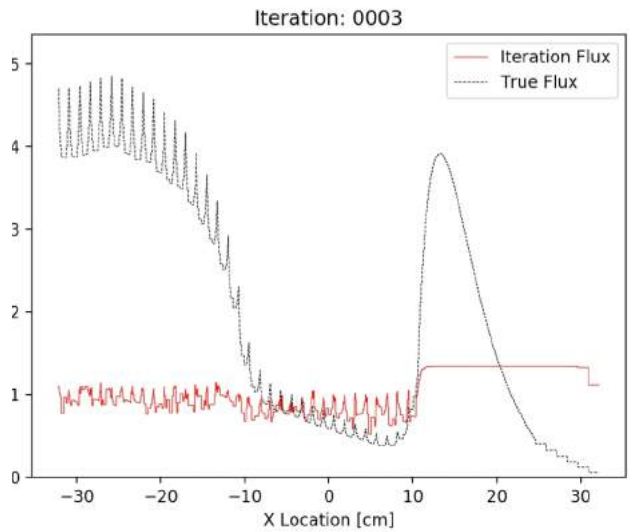
Example: 2D C5G7



Traditional Deterministic



Random Ray



- Random ray method can generate noisy, but unbiased stationary source distribution using very few rays
- Once stationarity reached, statistics can be accumulated (just like in MC)

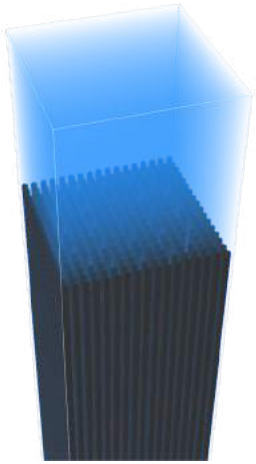
ARRC Porting Challenges

- Port from C89 using MPI + OpenMP Threading → MPI + CUDA
- Actually not too bad!
- CUDA compiler has been around since 2007: reliable and performant!
- Original app being in C89 meant:
  - No STL usage to remove
  - No virtual functions to flatten

ARRC Algorithmic Challenges

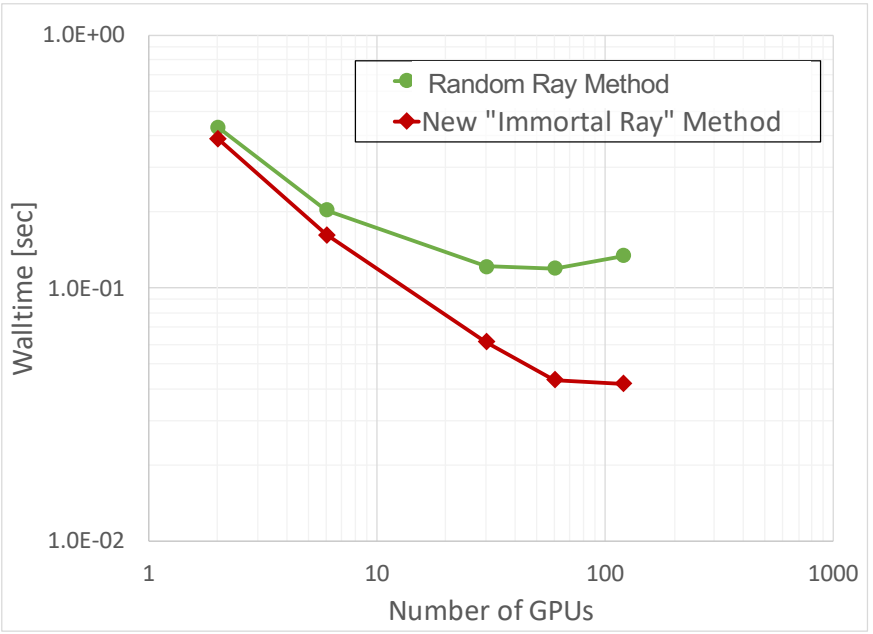
1. Adaption of event-based mode from Monte Carlo
2. Making domain decomposition performant on GPU
3. Increasing amount of parallel work exposed by method to enable strong scaling on GPU ("Immortal Ray" variant)

All of these things required significant research and/or implementation effort!



Simulation: 3D PWR Assembly

ARRC Strong Scaling on Summit Supercomputer
[Time per Power Iteration]



ARRC Port Epilogue

- CUDA not supported on exascale machines!
  - Aurora (Intel GPUs)
  - Frontier (AMD GPUs)

→ Thus, ARRC may need a *2nd* GPU port to a more portable model

- Significant research required into GPU-specific optimizations for this method
  - GPU optimizations for deterministic MOC not always applicable to random ray

Takeaways

- Many choices for GPU programming model – **no "best" choice in most cases**
- Porting to an exascale programming model can be easy/difficult **depending on starting point**
- Performance gains on GPU as compared to CPU are not usually found out of the box**, but can usually be achieved via algorithmic and/or numerical innovations!**
- **Event-based algorithms** appear to be very helpful for performance in **both** Monte Carlo and MOC/random ray transport

Speaker: John Tramm (jtramm@anl.gov)


EXASCALE
COMPUTING
PROJECT


Argonne
NATIONAL LABORATORY

Takeaways

Immortal Rays: Rethinking Random Ray Neutron Transport on GPU Architectures

Scientific Achievement

Developed a new numerical variant of The Random Ray Method (TRRM) for neutral particle transport. The new variant, dubbed the "Immortal Ray Method," is a fundamental physics-based reworking of fundamental numerical aspects of the random ray method. The key development is a new way of handling initial conditions for rays that allows for use of a higher number of short-length rays rather than a lower number of long-length rays.

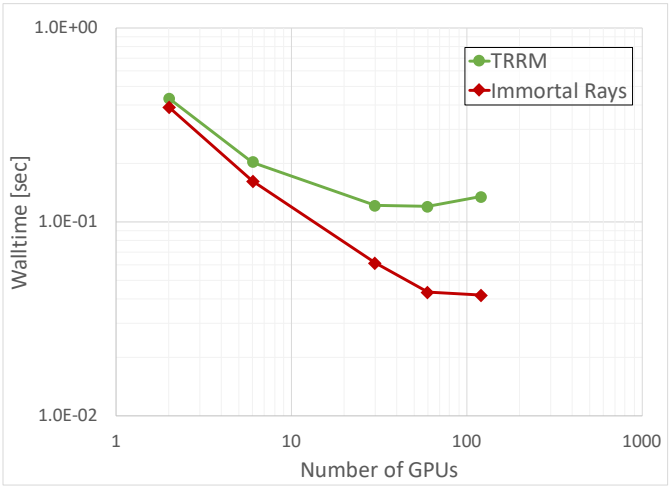
Significance and Impact

TRRM is a recently developed method for numerically solving the Boltzmann PDE in the context of neutral particle transport. It is notable due to its ability to perform full core simulations in a faster and much more memory efficient manner than was previously possible. Previous work identified parallel scaling losses on GPU-based systems when using random ray. The new Immortal Ray variant massively boosts available parallelism while simultaneously cutting communication costs – greatly improving scalability on GPU-based supercomputers.

Research Details

- Immortal ray improves node-level GPU performance on small problems by up to 4x.
- Resulted in strong scaling performance improvement of 3x on the Summit supercomputer.
- Identified several numerical problems introduced by the new method and developed and tested solutions that largely mitigate these issues.

Strong scaling study on the Summit supercomputer (lower is better)

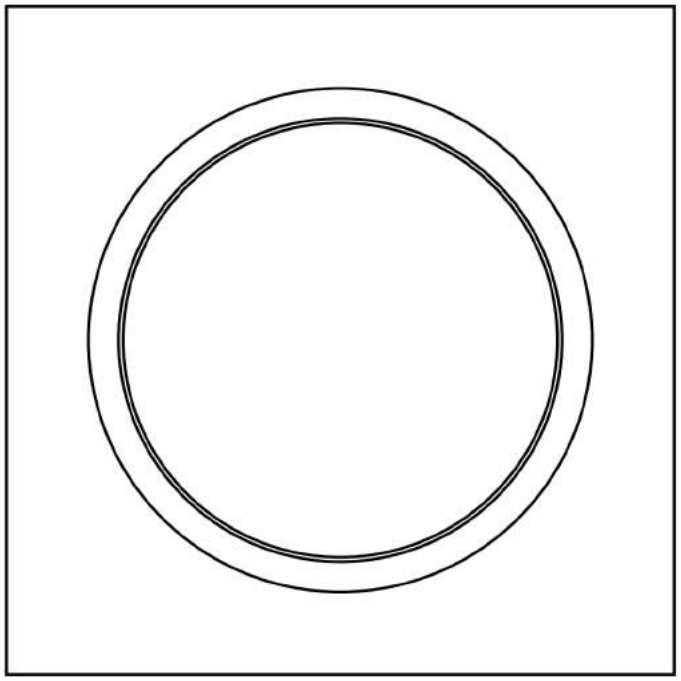


Citation: Tramm, J., Siegel, A. (2021*). Immortal Rays: Rethinking Random Ray Neutron Transport on GPU Architectures. (Submitted to) Journal of Parallel Computing – ECP Special Issue. (Under Review)*

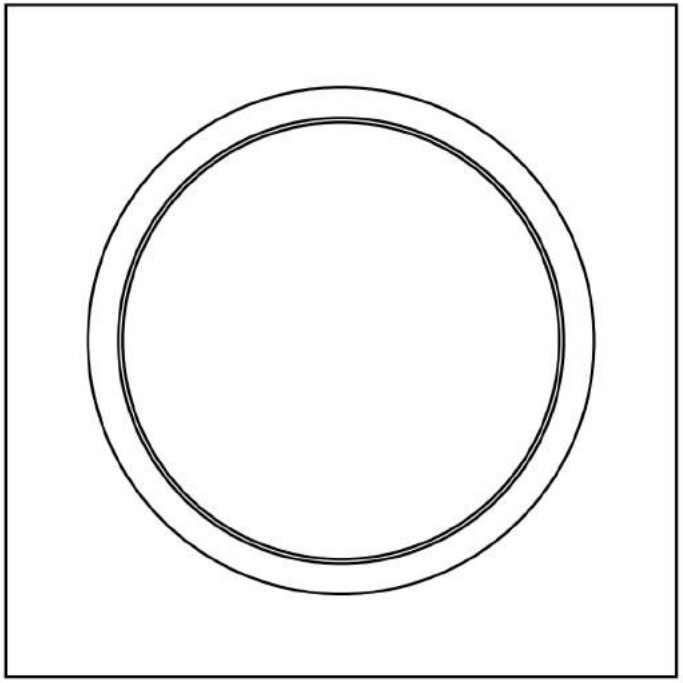
ANL: John Tramm, Andrew Siegel

Deterministic Method of Characteristics (MOC) vs. Random Ray

Traditional Deterministic



Random Ray



Things Required to Make it Work

- Starting angular flux approximation (dead zone)
- Two region stochastic convergence, similar to Monte Carlo (MC)
  - Shannon entropy monitoring

Advantages compared to traditional deterministic MOC:

- No storage of angular fluxes at boundaries
  - ➢ 50-1000x less memory needed
- Unbiased sparse iteration
  - ➢ 50-500x less work per iteration

John R. Tramm, Kord S. Smith, Benoit Forget, and Andrew R. Siegel. "The Random Ray Method for neutral particle transport." *Journal of Computational Physics*, 342:229 – 252, 2017.

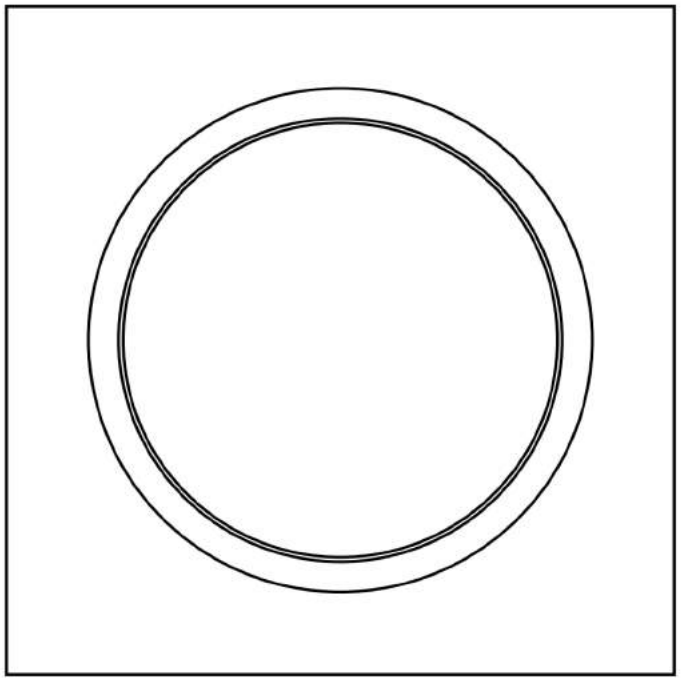
Example: Random Ray Particle Transport

Way too much if I introduce it all and try to explain everything

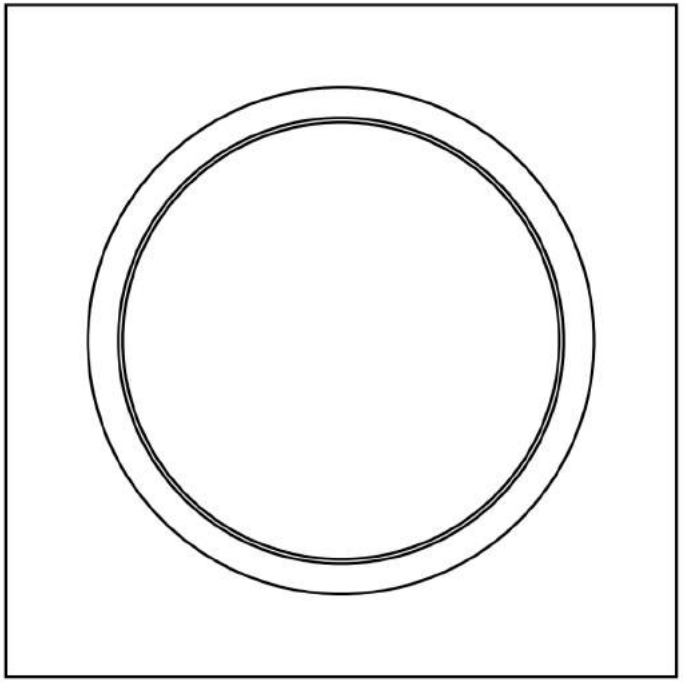
- Note the time to shill random ray.
- No one wants that.
- It's obnoxious.
- Boo
- Better idea:
  - What did you have to do to get up and running on GPU

Deterministic Method of Characteristics (MOC) vs. Random Ray

Traditional Deterministic



Random Ray



Things Required to Make it Work

- Starting angular flux approximation (dead zone)
- Two region stochastic convergence, similar to Monte Carlo (MC)
  - Shannon entropy monitoring

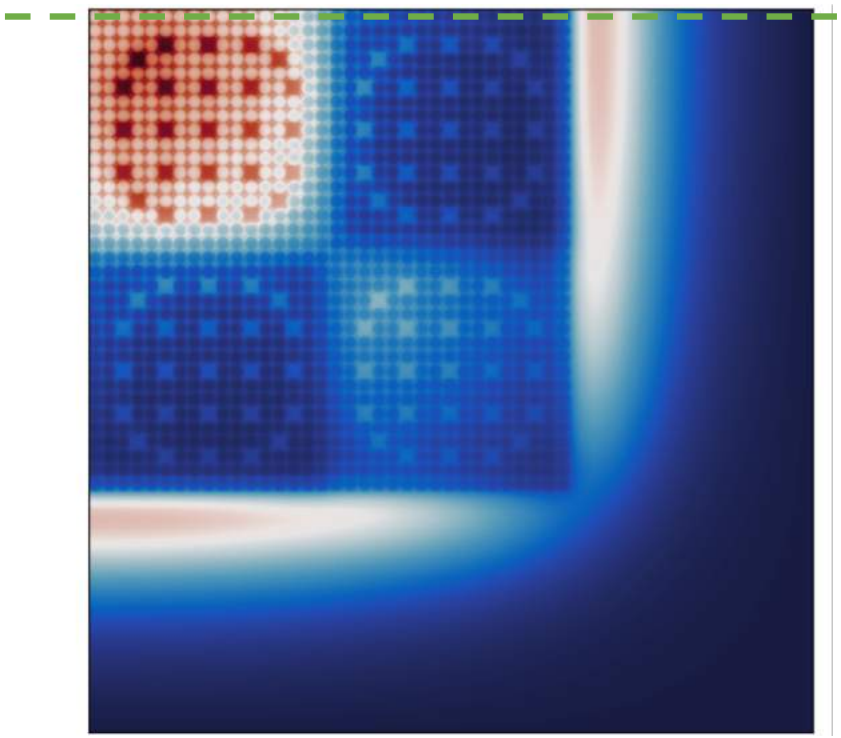
Advantages compared to traditional deterministic MOC:

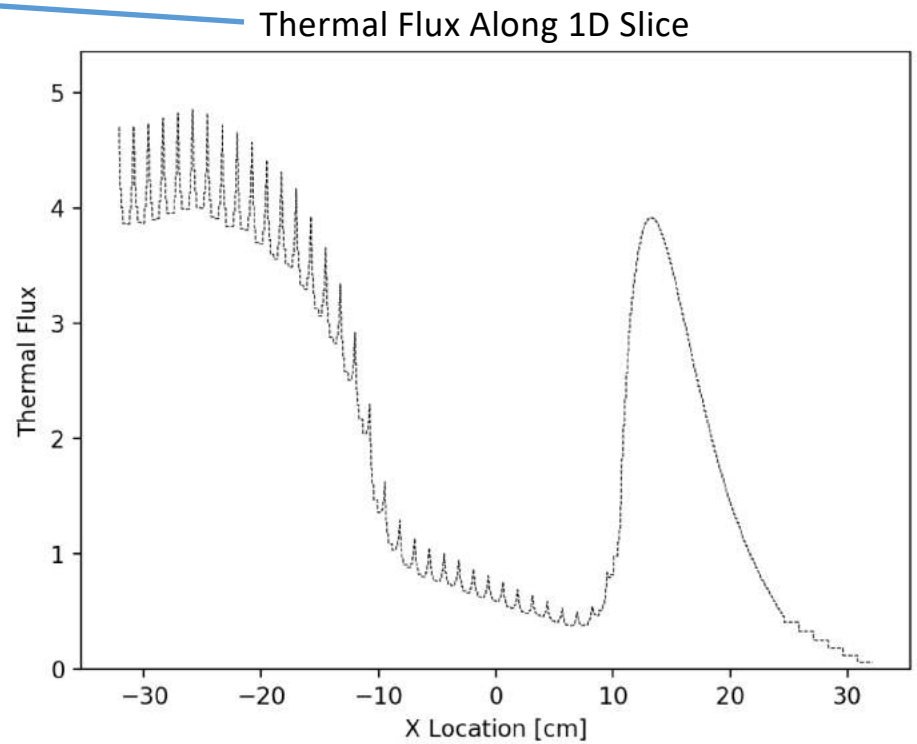
- No storage of angular fluxes at boundaries
  - ➢ 50-1000x less memory needed
- Unbiased sparse iteration
  - ➢ 50-500x less work per iteration

John R. Tramm, Kord S. Smith, Benoit Forget, and Andrew R. Siegel. "The Random Ray Method for neutral particle transport." *Journal of Computational Physics*, 342:229 – 252, 2017.

Why does random do so much less work than deterministic MOC?

Example: Convergence of thermal flux in a slice of 2D C5G7 benchmark problem



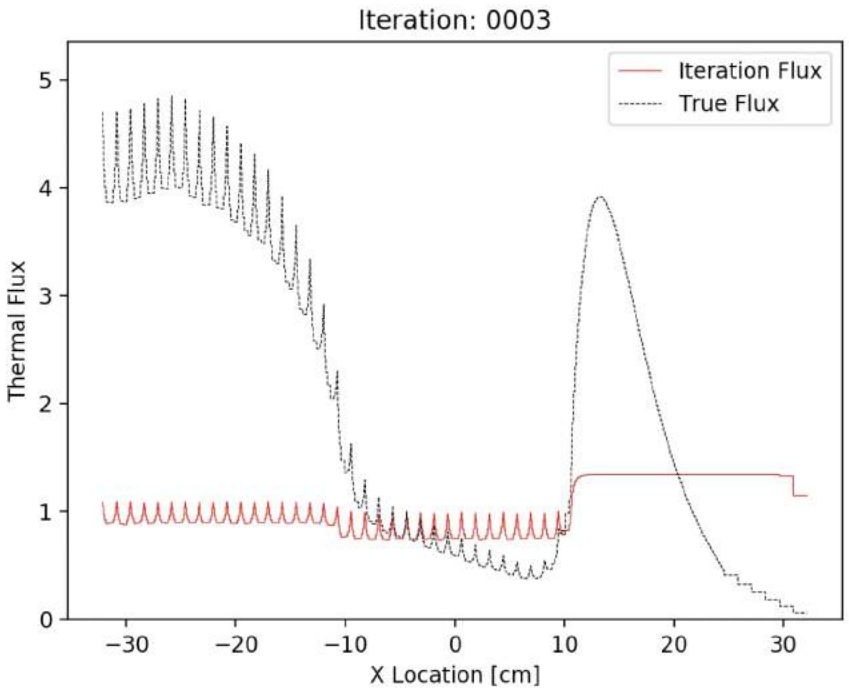


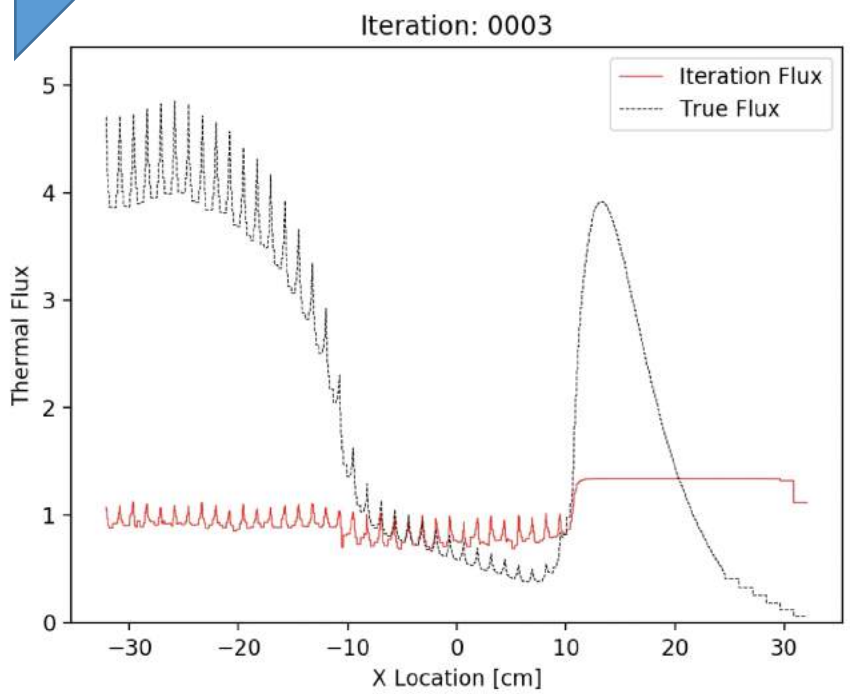
Deterministic MOC:
Decreasing Ray Density Increases Error

Fine Tracks

50x Lower Ray Density

Coarse Tracks





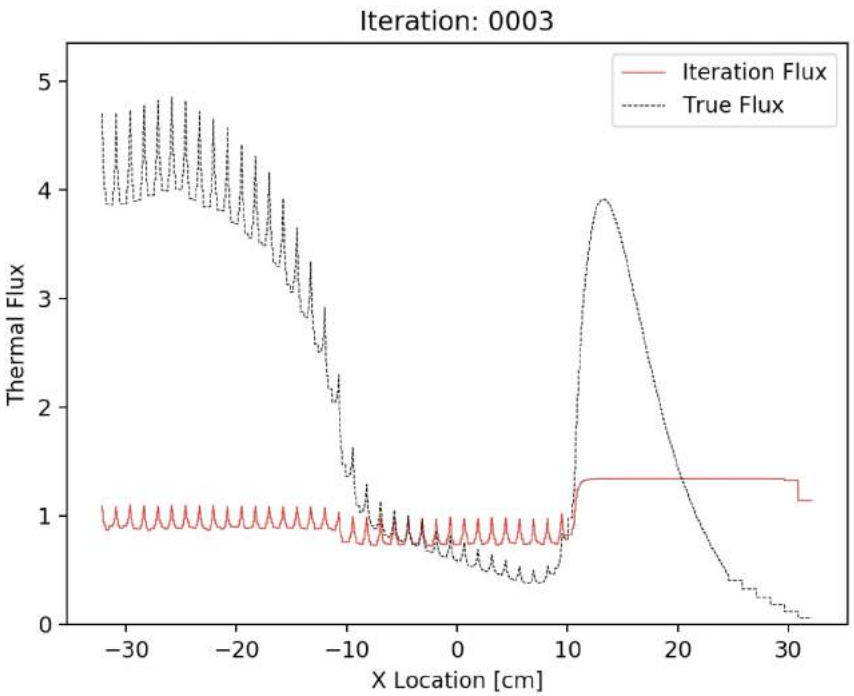
NATIONAL LABORATORY

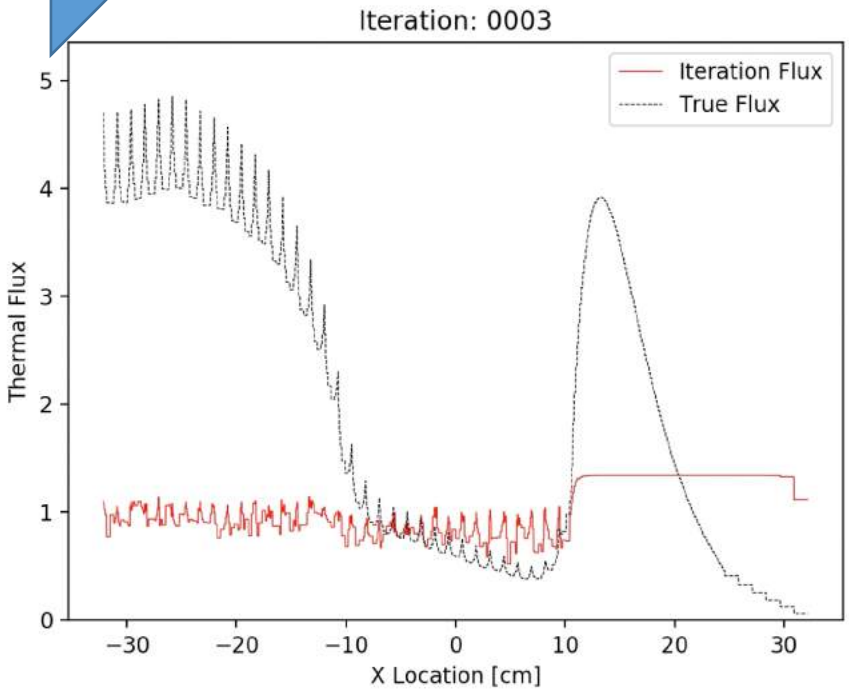
Random Ray:
Decreasing Ray Density Does Not Affect Error

Fine Rays

50x Lower Ray Density

Coarse Rays





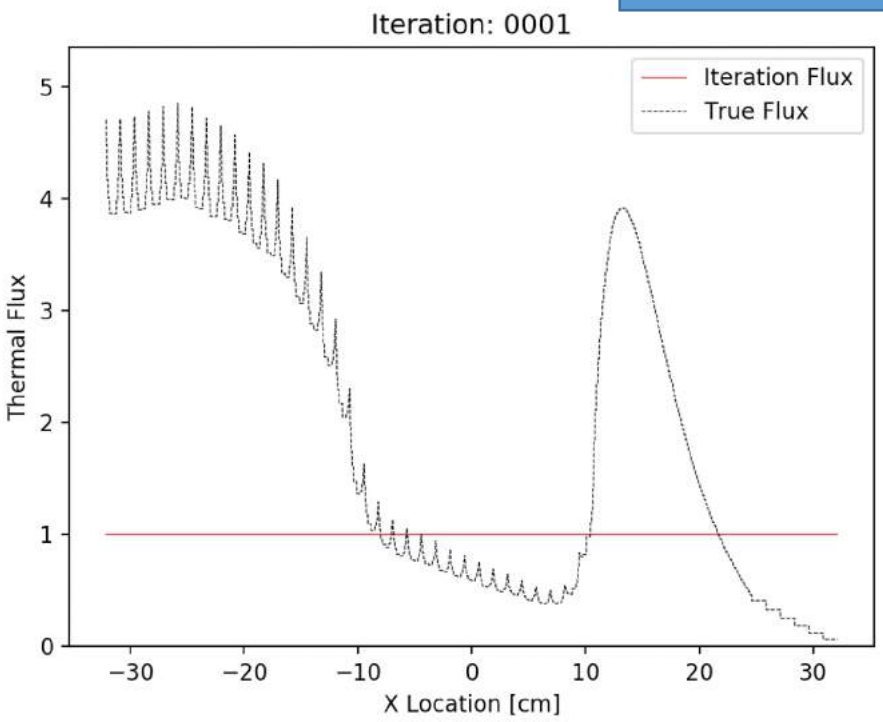
NATIONAL LABORATORY

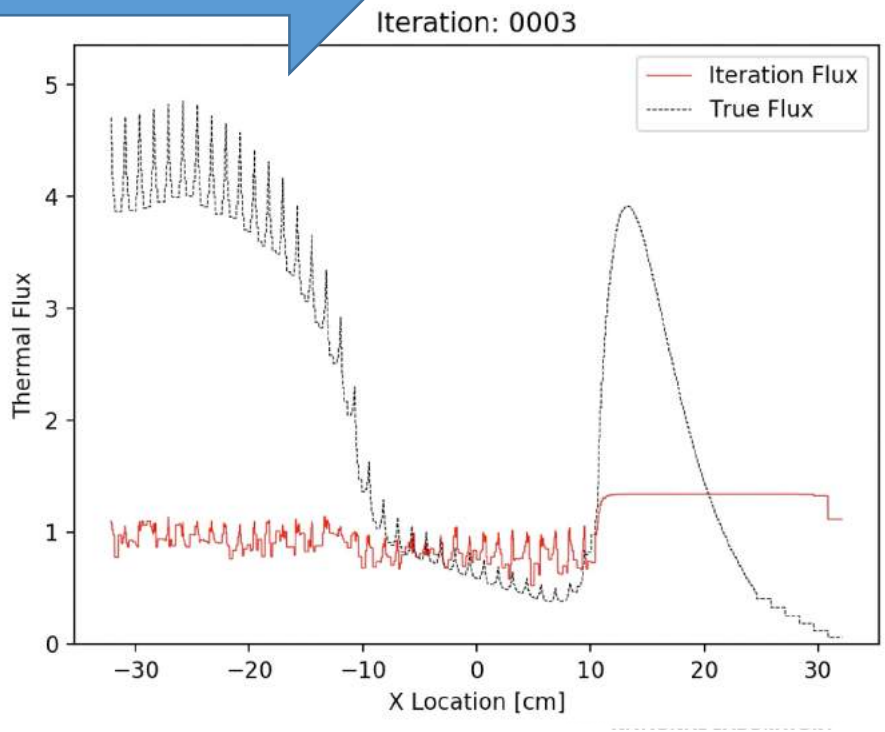
Deterministic vs. Random Ray: Runtime Comparison

Traditional Deterministic (Fine Tracks)

Lower Ray Density
=
Faster per Iteration

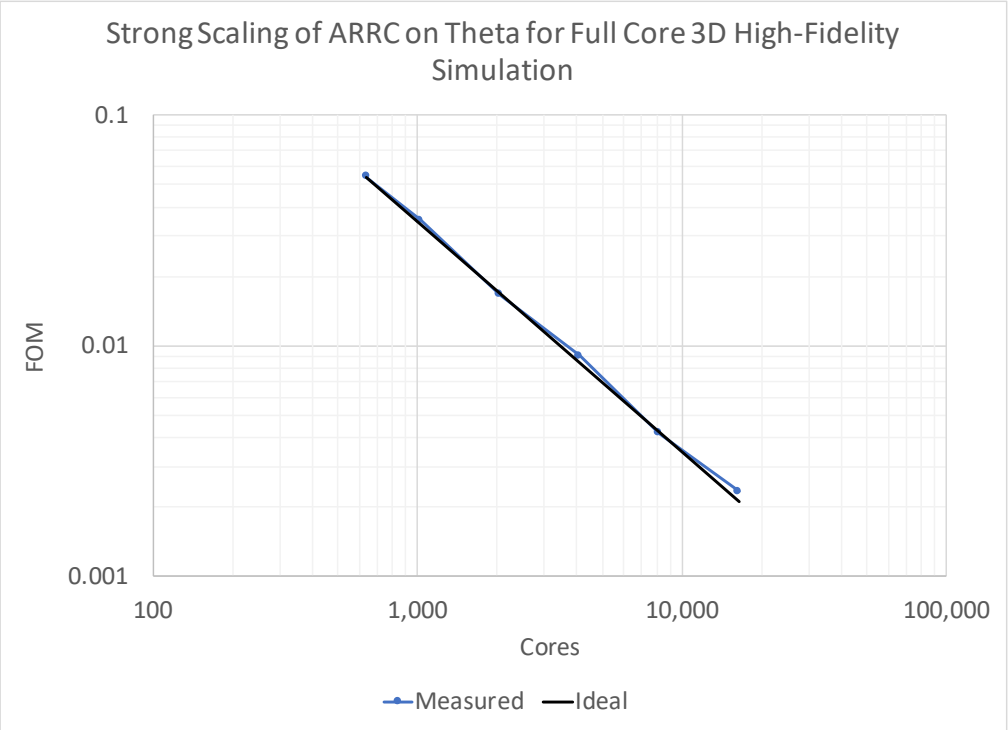
Random Ray (Coarse Rays)





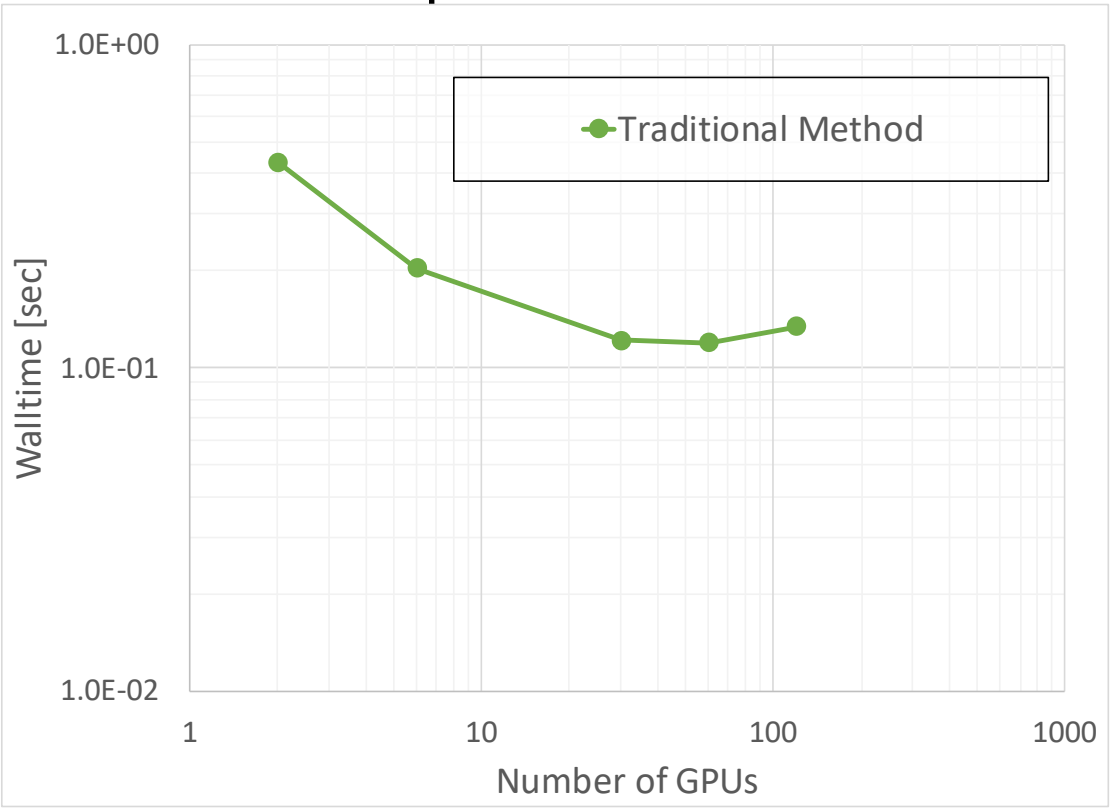
This is Great, Right?

- For a full core 3D reactor simulation:
  - OpenMOC: 4.81E+14 integrations per iteration
  - ARRC: 1.24E+12 integrations per iteration
  - Random ray requires 388x less work per iteration
- Works great on CPU-Based Supercomputers!

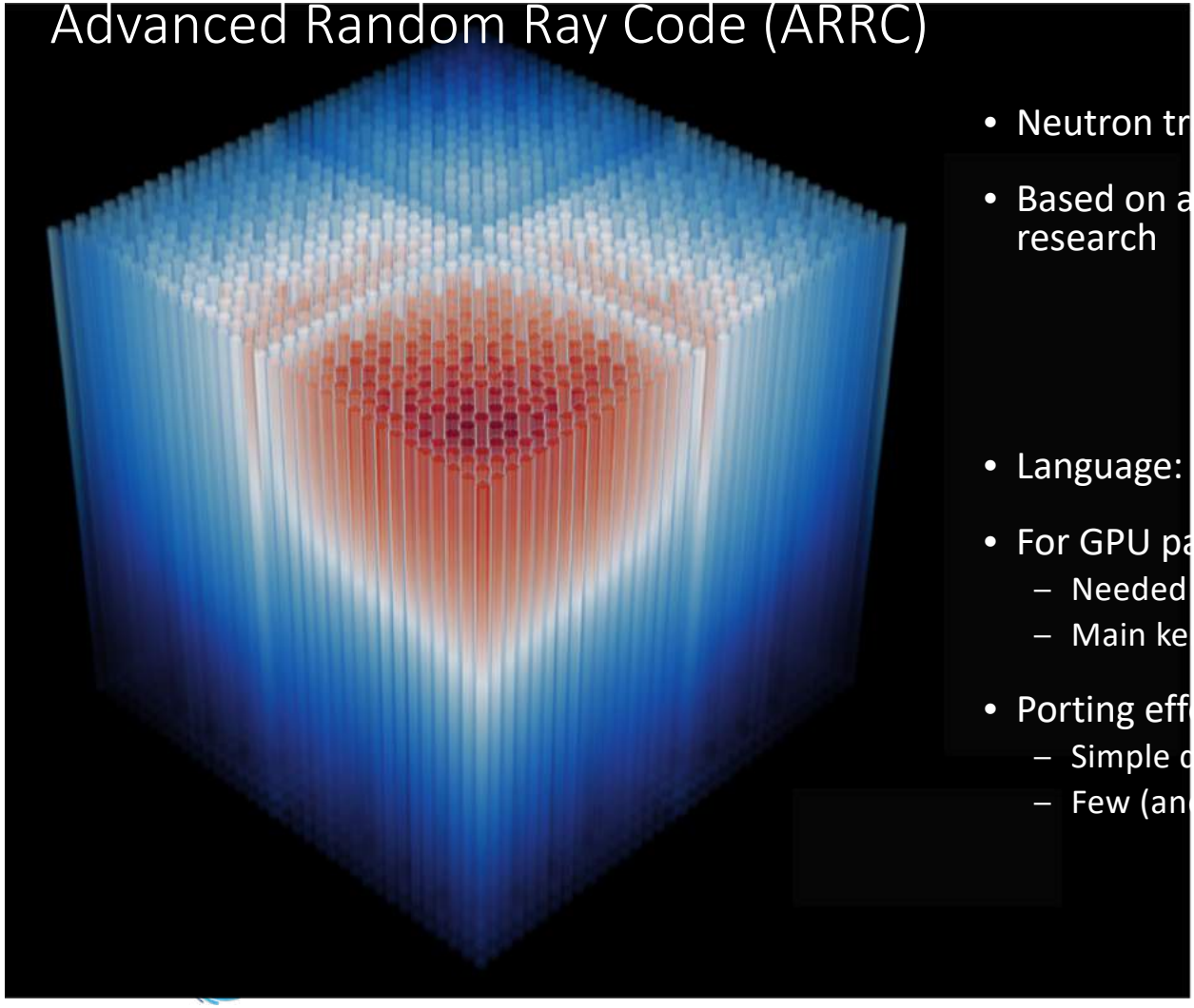


Problem: Less work means less parallelism

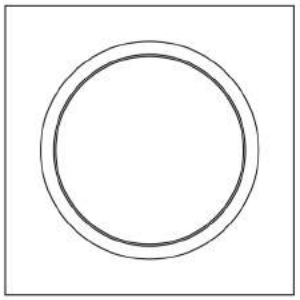
- Not



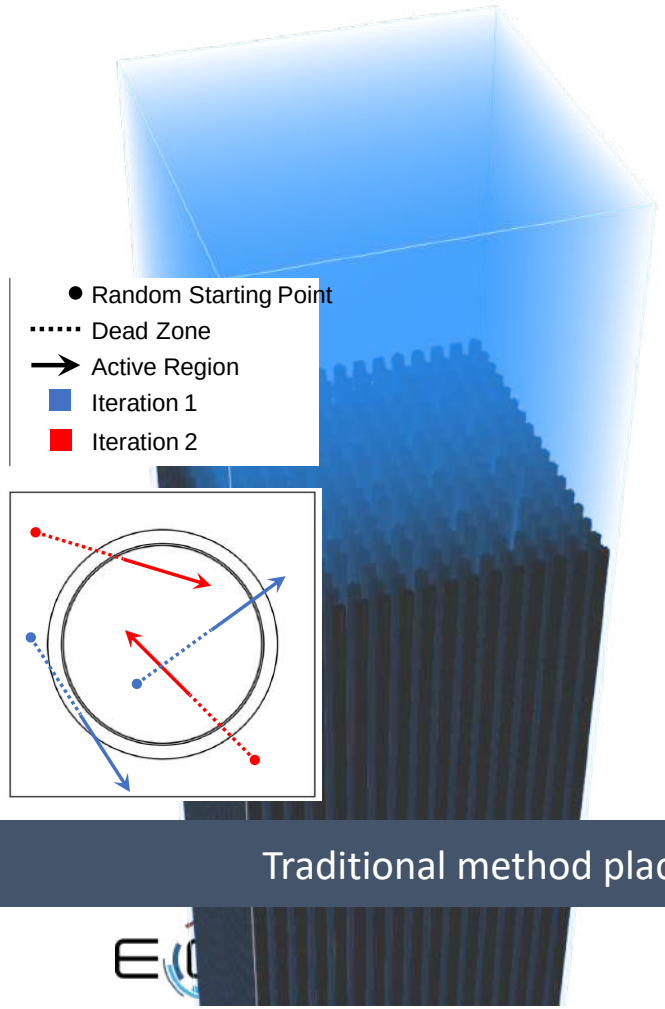
Advanced Random Ray Code (ARRC)

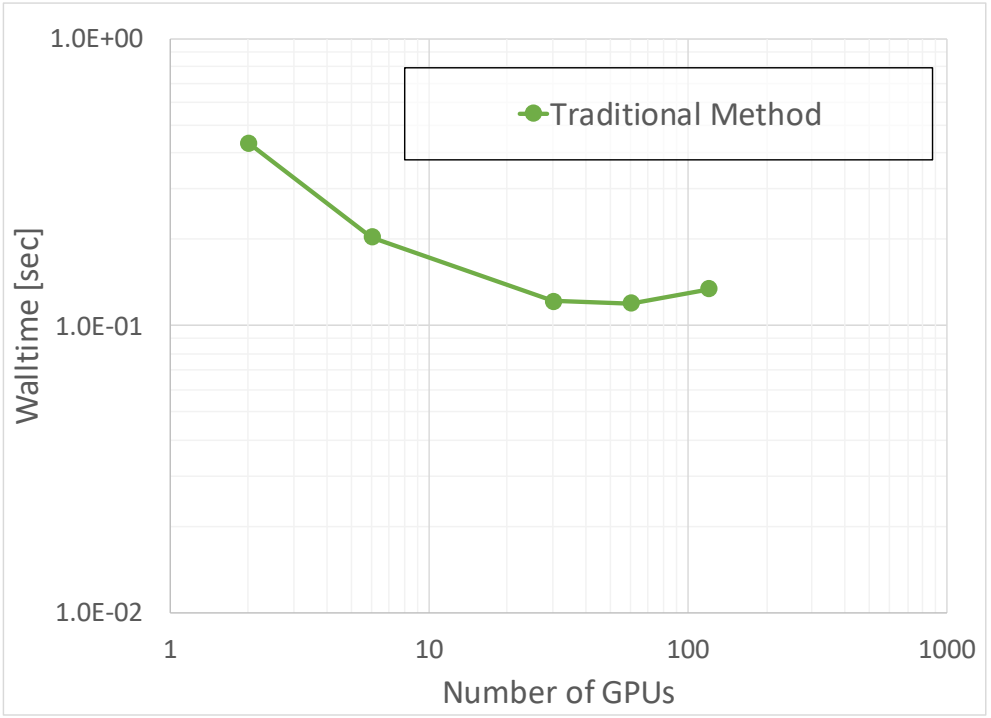


- Neutron tr
- Based on a
  research
- Language:
- For GPU pa
  - Needed
  - Main ke
- Porting eff
  - Simple c
  - Few (an



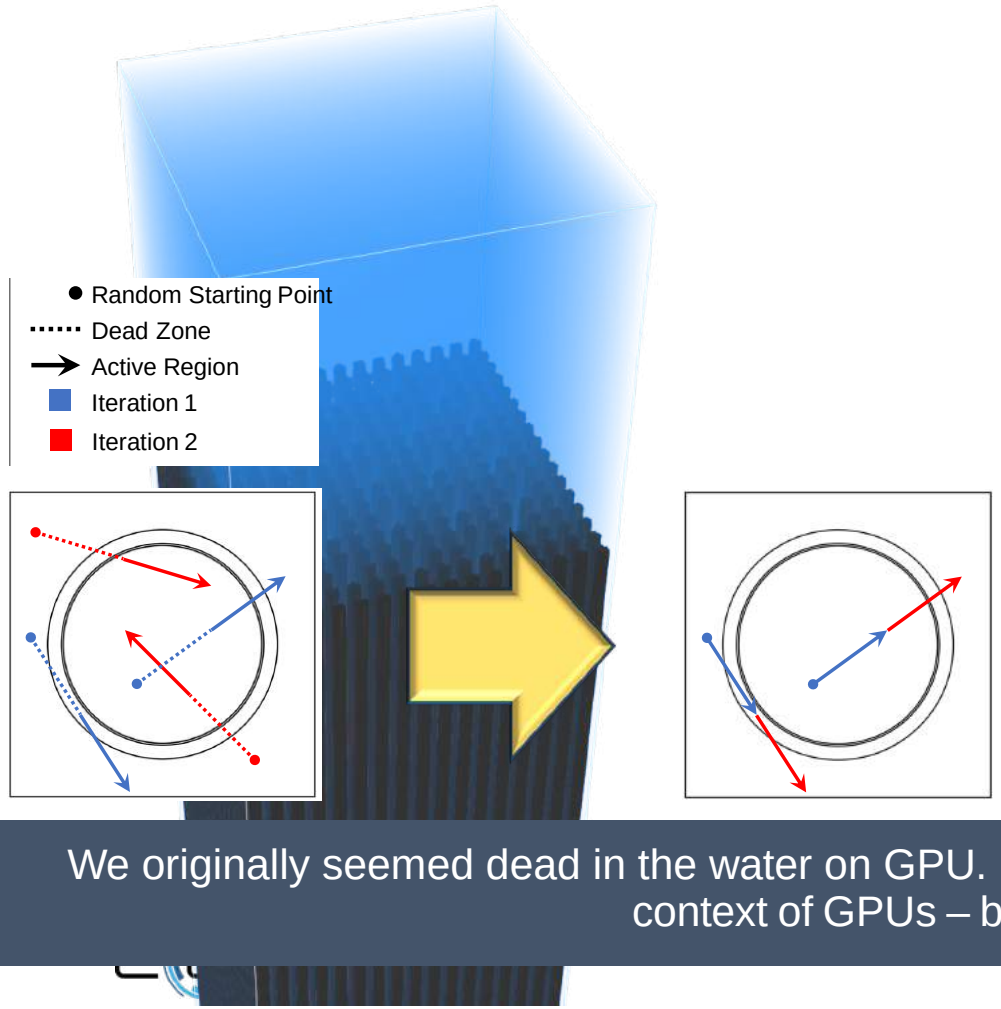
Argonne NATIONAL LABORATORY

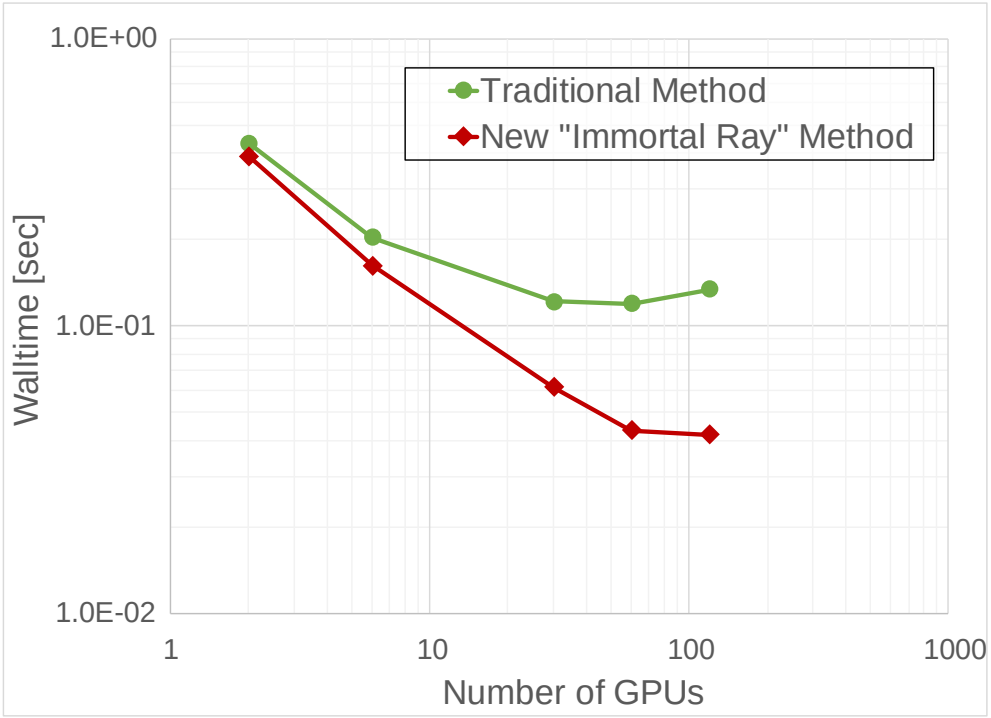




Traditional method places hard caps on available parallelism – inhibiting strong scaling.

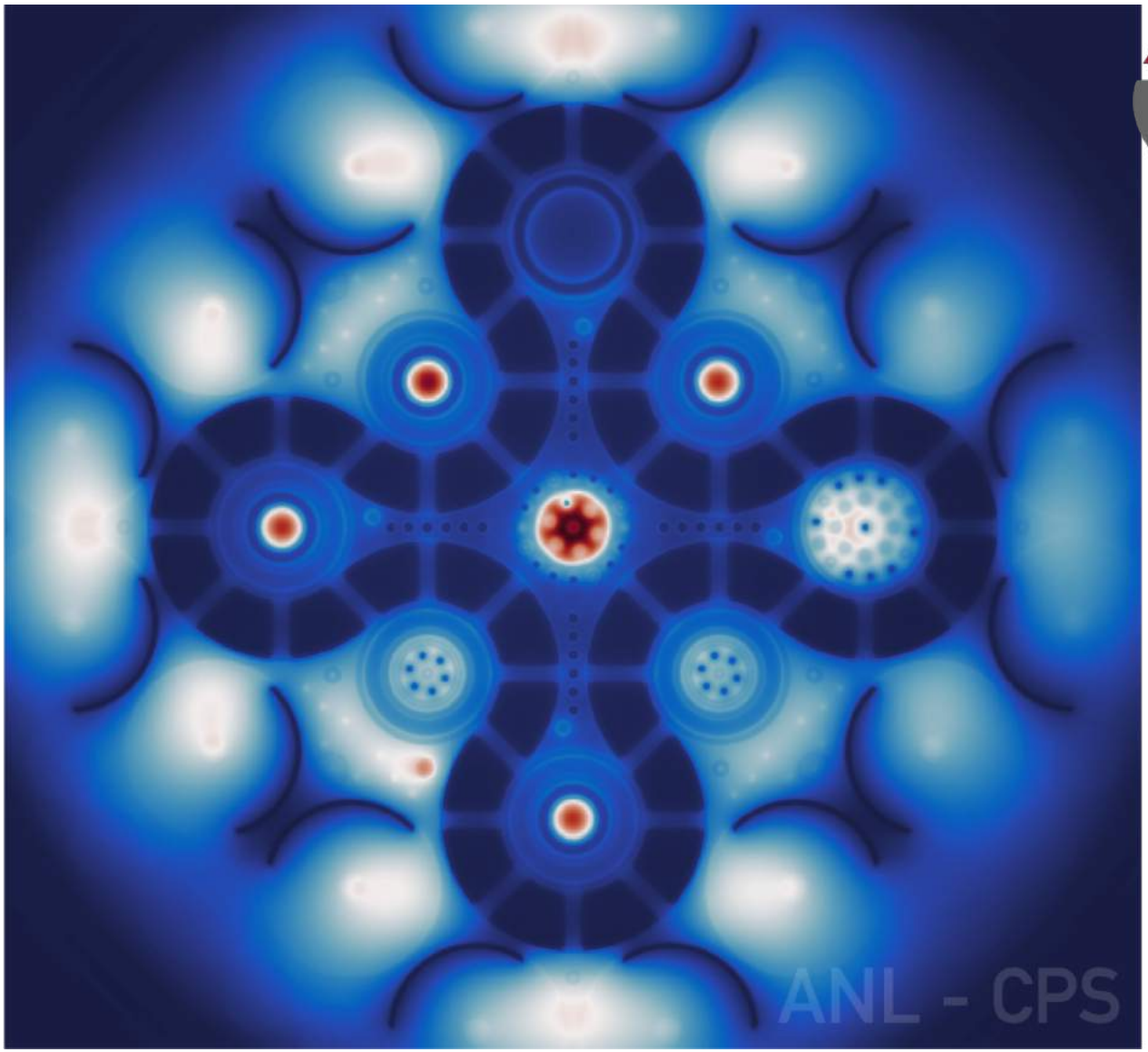
Argonne NATIONAL LABORATORY





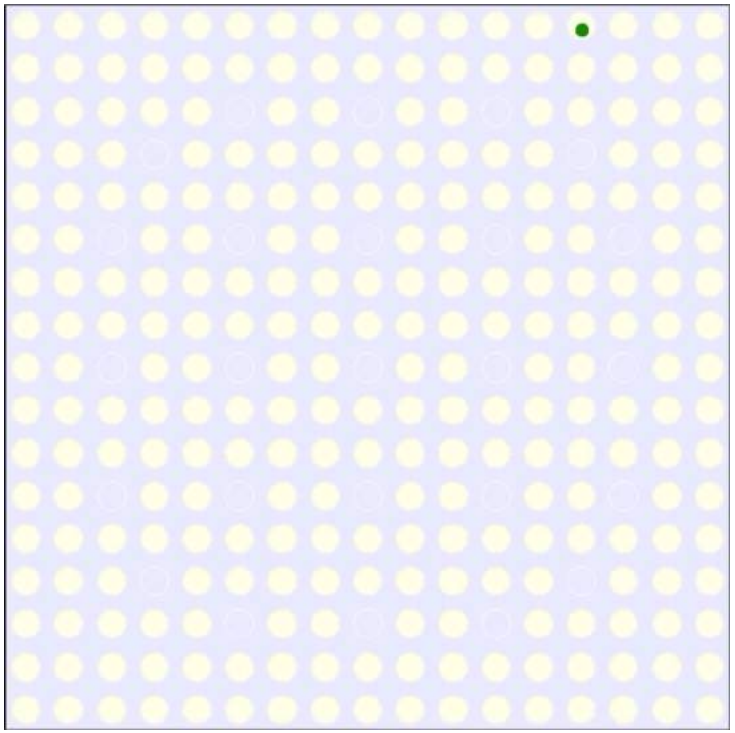
We originally seemed dead in the water on GPU. Innovation was required to rethink fundamental numeri
context of GPUs – boosted scalability by 3x.

NATIONAL LABORATORY



OpenMC

Argonne
NATIONAL LABORATORY



Traditional "History-Based" Method

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

New "Event-Based" Method

- Only execute one low level event type at a time (kernel splitting)
- Parallelism expressed over particles requiring that event
- Greatly reduces thread divergence

Another MC application, Shift, found the event-based method 3.5x faster than traditional CPU-optimal history-based method!

Overview

Lab Name