Toward GPU Accelerated Full Simulation of Optical Calorimetery with Celeritas

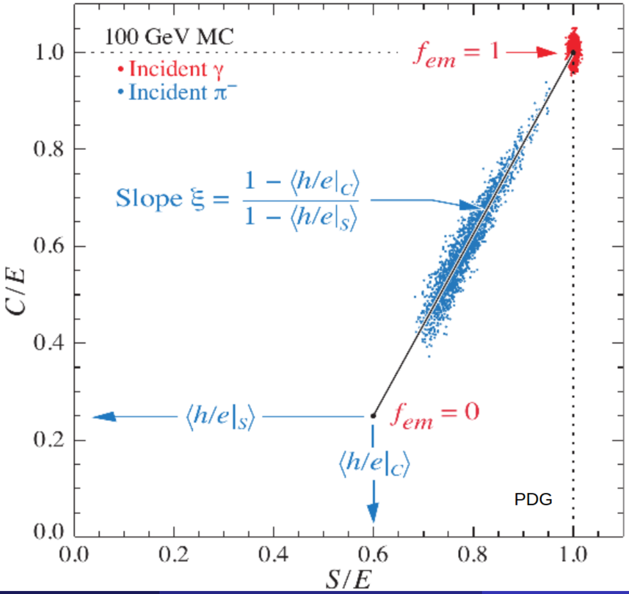
Hayden Hollenbeck

University of Virginia

CalVision Workshop August 6th, 2024

Outline

CalVision: Goals

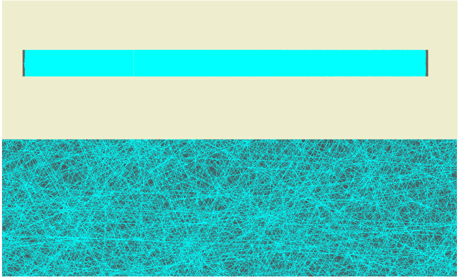


Single Bar Simulation

Simulating single crystal response

- PWO_4 crystal with 1 SiPM on the front face and 2 SiPMs on the rear face
- Single incident charged particle
- Options to set filters in front of each SiPM

Goal: Measure scintillation and Cerenkov signals in each SiPM



Above: Single crystal with SiPMs (grey) on front and rear faces. Incident 1 GeV muon.

Below: Zoomed image showing optical photon tracks (blue), with $\sim 20,000$ Cerenkov photons and $\sim 100,000$ scintillation photons.

Fast Parameterization: Scintillation

Scintillation properties:

- Isotropic emission
- Single emission spectrum
- Number of photons depends only on energy deposit (Birk's Law)

Fast Parameterization

Setup:

- Create position dependent bins in the crystal
- Simulate random photons in each bin
- If photon reaches SiPM sensitive area, record wavelength and time

During Simulation:

- Record position, time, and energy of every energy deposit in the crystal
- Manually kill all scintillation photons at track start

Post-Processing:

- Use Birk's law to generate photons for each deposit
- Convolute with pre-generated histograms

Fast Parameterization: Cerenkov

Cerenkov properties:

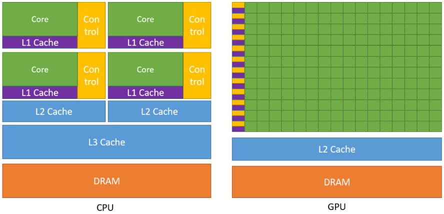
- Well-defined cone angle from particle velocity
- Highly directed
- Emission spectrum is velocity dependent

Both time and energy are sensitive to particle velocity!

- Spectrum and angle depend on speed
- Path depends on particle direction and angle
- Need to bin in all 3 velocity components!
- May also need finer bins in position

No easy way to do fast sim for Cerenkov …

Celeritas: Use GPUs to fully simulate optical physics quickly



Goal: high performance for parallelizable, floating-point problems

Different hardware architectures:

- Optimal use cases for each?
- Why and how do they get their performance boosts?
- What's their limitations?

Resources:

- CUDA programming guide
- How CUDA Programming Works - GTC 2022

CPU Parallelism

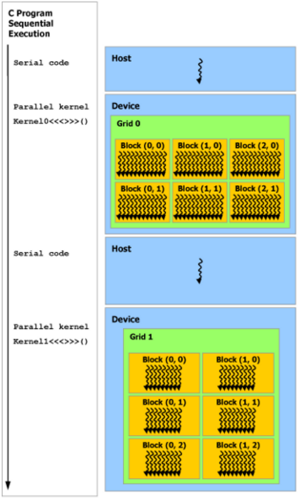
Example CPU: *AMD EPYC 9454*
Atomic execution unit: 1 thread on 1 core

- **Multicore**: multiple cores on each CPU (*48 cores*)
- **SIMD**: Single Instruction Multiple Data
  - Vectorized instructions on larger registers (128-, 256-, 512-bits)
  - Arrays of data need to be aligned and sequential
- **Multi-Threading**: simulate multiple threads on a single core through interleaving
  - Context switching - very expensive!
  - (Hardware) Simultaneous multithreading - simultaneous threads on a single core (*2 per core*)
- **Branching**: local thread flow control is completely independent of other threads
  - Can spoil pipelining and prefetching
  - Mitigation through branch predicition
- **Memory Access**: Many caches levels & few registers - save and reuse results
  - CPU Stalls: threads must wait for cache lookup
  - *L3 cache: 256 MB*

GPU Execution Hierarchy

Execution Hierarchy

- Thread - Single process run on a single core (same as CPU thread)
- Warp - 32 threads that get run simultaneously
  - Smallest *GPU execution* unit
  - All threads run the same command on the same clock tick
  - Branching handling by disabling certain threads
- Thread Block - Set of identical threads to run
  - Smallest *user execution* unit
  - User may specify thread code and block dimensions
  - Each thread has a unique ID - specifies what data it should run on
  - Blocks guaranteed to run with same shared data cache



GPU Architecture

Hardware Hierarchy

- Streaming Multiprocessor (SM) - set of processors with a common shared memory
  - Warp Schedulers - Each schedular runs 1 warp / clock cycle
    - Streaming Processor (SP) - Cores that run a single thread
    - Register File - Register memory for threads
  - L1 Cache / Shared Memory - Common local memory available to threads

Thread registers persistent - no context switch penalty!

GPU Hardware Specs

Ex: NVIDIA A100 Tensor Core GPU

- 108 Streaming Multiprocessors per GPU
- 64 32-bit floating point cores per SM
- 32 64-bit floating point cores per SM
- 192 KB of combined shared memory and L1 cache
- 4 warp schedulers per SM
- 64 max warps per SM = 2048 max threads per SM (not simultaneous, just managed!)

Roughly $108 \times 4 \times 32 = 13,824$ simultaneous threads!

GPU Performance Considerations

- Memory Access
  - Access aligned to cache lines most efficient - random access has major speed penalties
  - Delays in memory access hidden by warp scheduling - need high occupancy!
- Branching
  - Minimize time spent executing 2 different branches
  - Don't avoid entirely! Many optimizations, tricks, and ways to minimize penalties!
- Host-Device transfer
  - Initialize constant and global data once
  - Transfer only when necessary
  - Minimize transfer during hot loops
- Occupancy
  - Maximize number of warps available to be run
  - Maximize number of threads in a warp - may need to reorganize data!

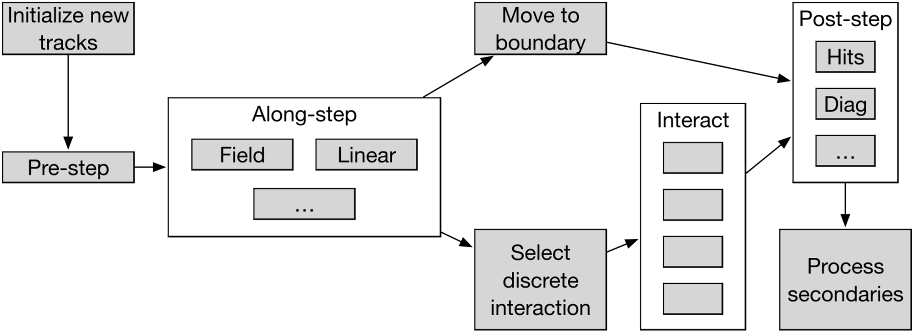
Celeritas: Overview

- Standalone GPU accelerated simulation code
- Runs both with and without GPUs
- Can drop into existing Geant4 code to offload tracks to the GPU
- Currently implements high energy EM physics
- Continual unit testing and physics validation against Geant4
- Developers at ORNL, FNAL, ANL, BNL, and more!

`https://github.com/celeritas-project/celeritas`

Celeritas R&D Report: Accelerating Geant4. `https://doi.org/10.2172/2281972`

Celeritas: Core Physics Loop



Celeritas: Actions

Q: When does work actually get done on the GPU?
A: Actions - per track GPU kernels!

1. Eg: tracks need to undergo discrete actions every step
2. One action determines which discrete interaction for every track
3. Collect actions of same discrete interaction
4. Run appropriate kernel on track collection

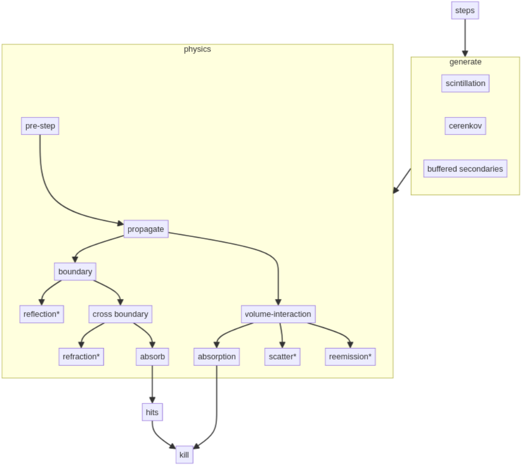
Design Logic:

1. Need to handle variable number of tracks
2. Need to handle variable number of actions
3. Need to handle randomly chosen kernels each step

Celeritas: Optical Physics

Optical photons: simulated as distinct particles from high energy photons

- Entirely separate optical physics loop after core physics runs
- Simpler physics - optimized algorithm



Celeritas: Generators

Need to move photons from core physics to optical physics loop

- Same idea as fast parameterization: record energy deposits, locations, times
- Store as small data structure, wait for the core loop to end
- Use generators to initialize optical photons from each record

Generators

- Scintillation
- Cerenkov

Both currently implemented in Celeritas

Celeritas: Processes

Discrete Optical Processes: currently being implemented

- Absorption
- Rayleigh Scattering
- Wavelength Shifting
- Mie Scattering

Boundary Optical Processes: yet to be implemented

- Refraction
- Reflection

Conclusion

- Geant4 struggles to simulate high count optical physics events
- Fast parameterizations good for scintillation, not for precise Cerenkov
- GPUs allow simulating many many tracks concurrently
- GPU accelerated code Celeritas can be readily integrated into Geant4
  - Major focus is currently on optical physics!

Actively looking for help with development and integration with experiments - get in contact with us!