Celeritas: toward GPU-based particle transport for detector simulations in HEP experiments

Thomas Evans, ORNL
Seth R Johnson, ORNL
Philippe Canal, Fermilab

HSF-WLCG Workshop
Nov. 23, 2020

ORNL: Stefano Tognini

Fermilab: Soon Yung Jun, Guilherme Lima

Argonne: Amanda Lund

LBL: Vincent Pascuzzi

NNSA National Nuclear Security Administration

U.S. DEPARTMENT OF ENERGY | Office of Science

Celeritas Project

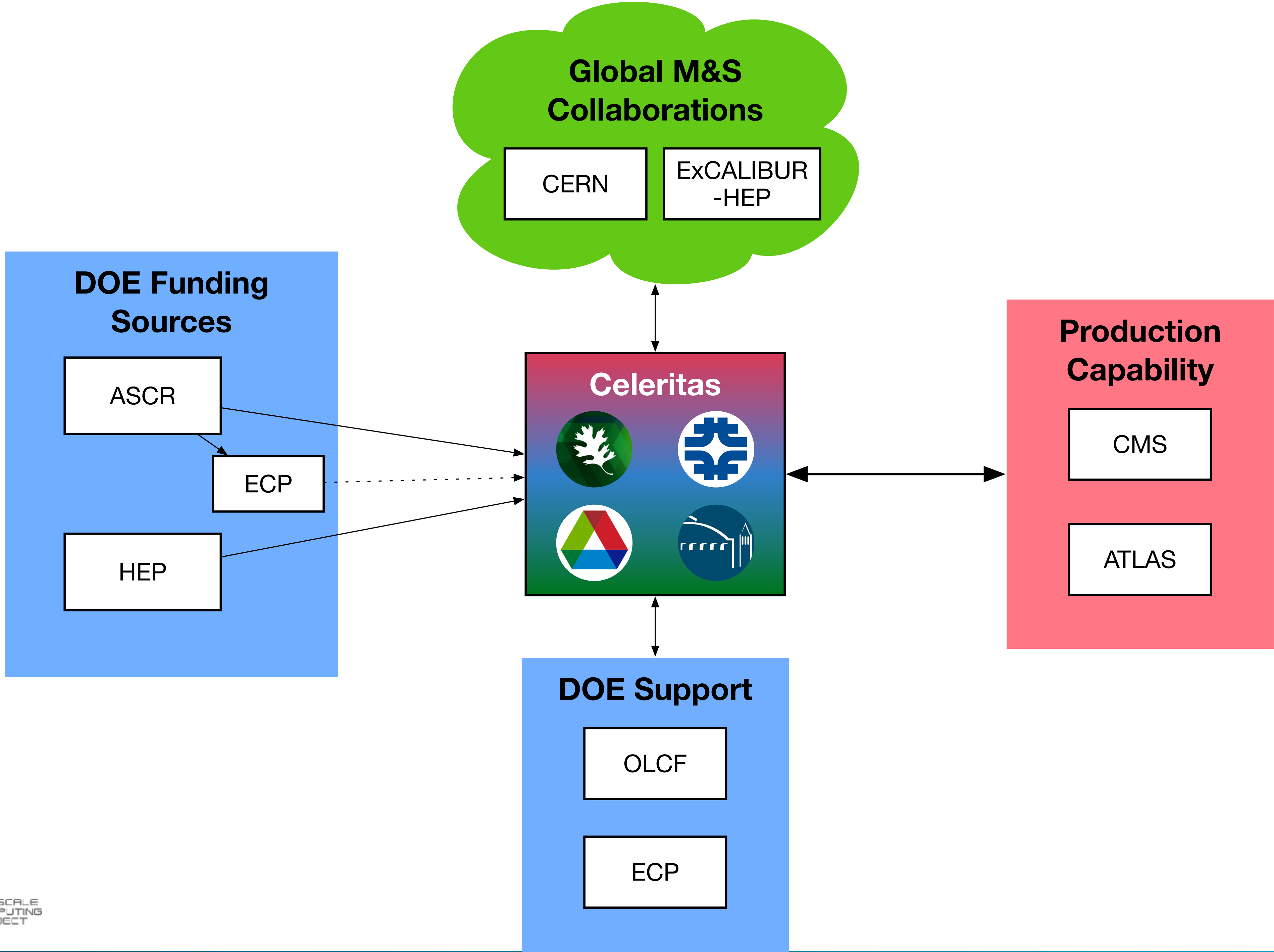
Project objective

Deliver a GPU-accelerated particle transport application for HEP detector simulations

1. Why is this capability needed?
   - Current high-fidelity, time-dependent, detector energy deposition simulations will not scale to proposed 10× luminosity increase in 2025-2026
2. What are the technical capabilities and opportunities needed for breakthroughs in the detector mod-sim area?
   - Efficiently use leadership class hardware (GPUs) to increase particle tracking throughput with concurrent improvements in I/O and post-processing analysis
3. How is the current project different from previous efforts and how will it enable those breakthroughs?
   - Accelerator technology (GPUs) has reached maturity
   - Monte Carlo transport applications have demonstrated performance on these architectures in ECP (exascaleproject.org) and CASL (casl.gov) for science campaign level simulations
   - Through ECP, we have access to a complete ecosystem of high performance libraries and tools
4. What is the long-term strategy for integration with Geant4?
   - Use Celeritas to offload EM physics in a standard Geant4-constructed application
   - Use Celeritas as part of a broader LHC workflow for complete detector simulation
   - Combinations of both approaches should be possible

Requirements

- Utilize modern heterogeneous (CPU + GPU) architectures at all scales (laptop to exa)
- Read events from HEP community event generators (Pythia/HEPMC3, etc)
- Use community Geant4 geometry models (VecGeom/GDML)
- Include (ultimately) complete physics models for HEP detector simulation in Geant4
  - Preliminary focus is on high-energy EM physics
- Target most compute-intensive component of HEP detector simulation workflow: time-dependent, detector energy deposition (hit generation)
  - Complements and is part of standard Geant-driven LHC simulation workflow



Global M&S Collaborations
CERN
ExCALIBUR -HEP
DOE Funding Sources
ASCR
ECP
HEP
Celeritas
Production Capability
CMS
ATLAS
DOE Support
OLCF
ECP
EXASCALE COMPUTING PROJECT

Celeritas Code

Development approach

- Bottom-up development approach
- Parallelize effort over individual components
- Integration guided by weekly group meetings
- Modularity will enable restructuring for performance
- Demo mini-app progression for integrating components

Source code

Language	Files	Comment	Code
C/C++ Header	182	4537	5974
C++	45	1180	2494
CMake	17	344	1609
Bourne Shell	14	96	381
CUDA	7	202	358
Python	4	473	320
Total	**269**	**6832**	**11136**

Test code

Language	Files	Comment	Code
C++	49	1037	3172
C/C++ Header	22	394	1021
CUDA	8	159	510
Total	**79**	**1590**	**4703**

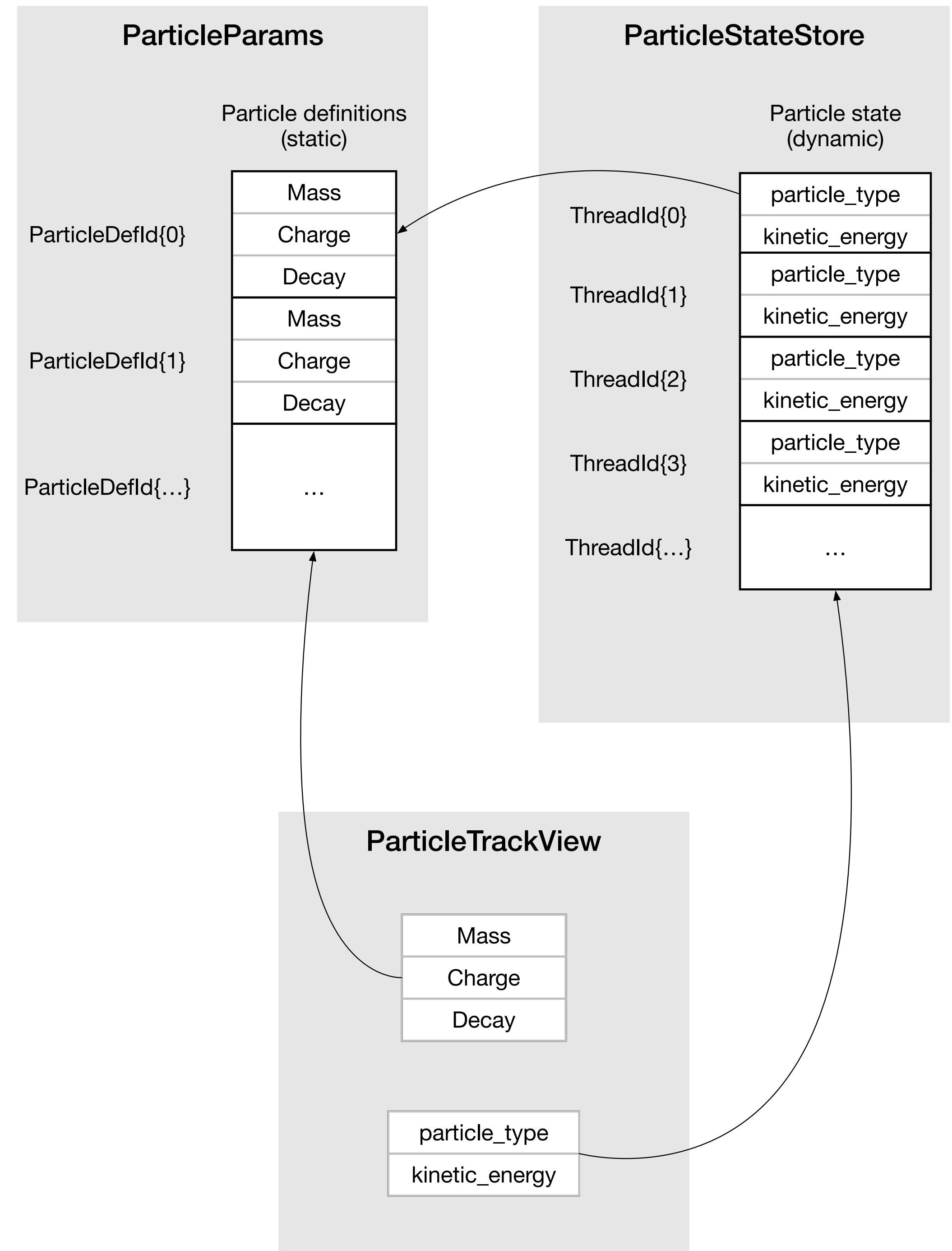
Celeritas code base statistics

Current development focus areas

- **Infrastructure**
  - Cohesive framework (GitHub wiki)
  - Import GPU algorithms from Shift GPU code
- **Physics**
  - Leverage Geant4 physics implementations and documentation
  - Export preprocessed Geant4 data where possible
- **Geometry**
  - Interface with VecGeom CUDA library

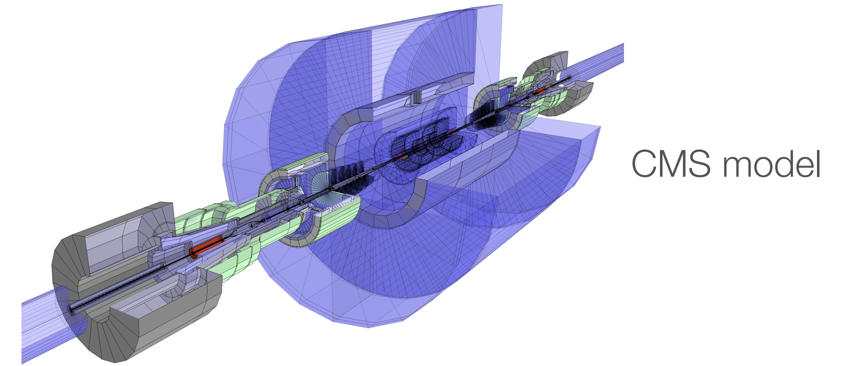
Software abstractions for portability

- Kernel code is separate from memory management and memory layout
    - Enables development, testing, and debugging on CPU
    - Allows isolated experimentation with data layouts to optimize GPU performance
    - Could enable (for example) Kokkos data management
- Kernels are plain C++ with annotations
    - Macros to enable CUDA device code, extensible to HIP and other performance abstraction layers
    - Runtime initialization code is host-only

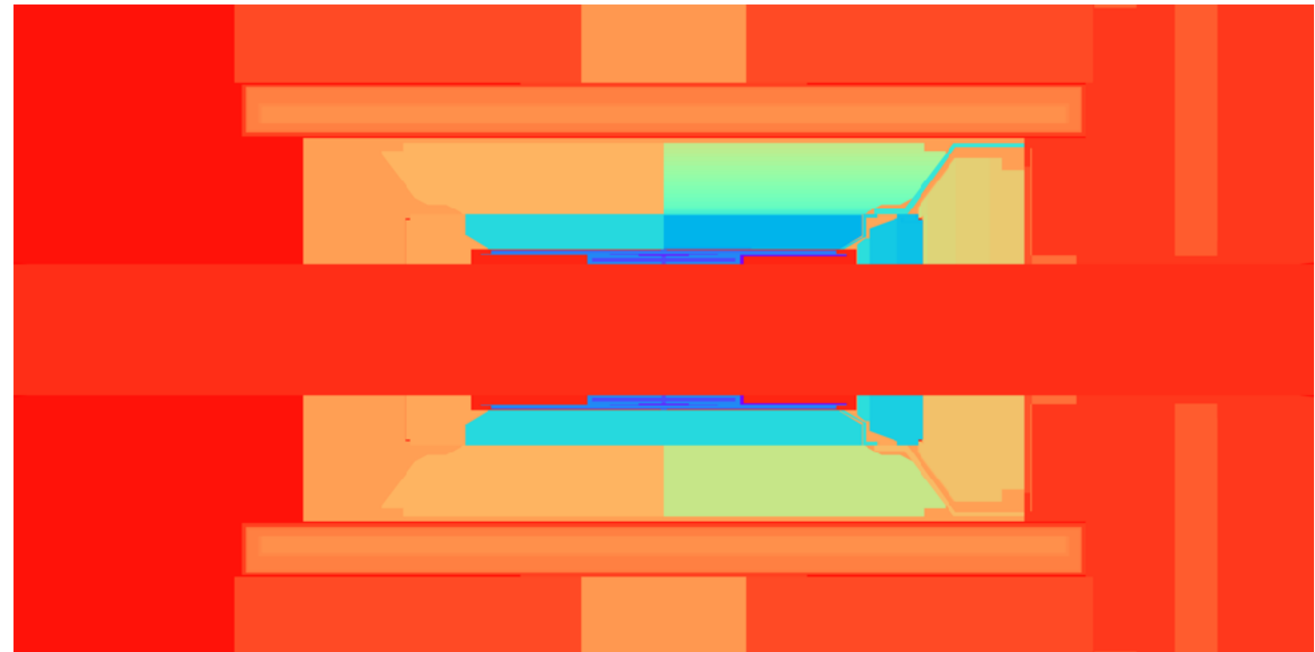


EXASCALE COMPUTING PROJECT

Geometry mini-app: CMS ray trace


CMS model

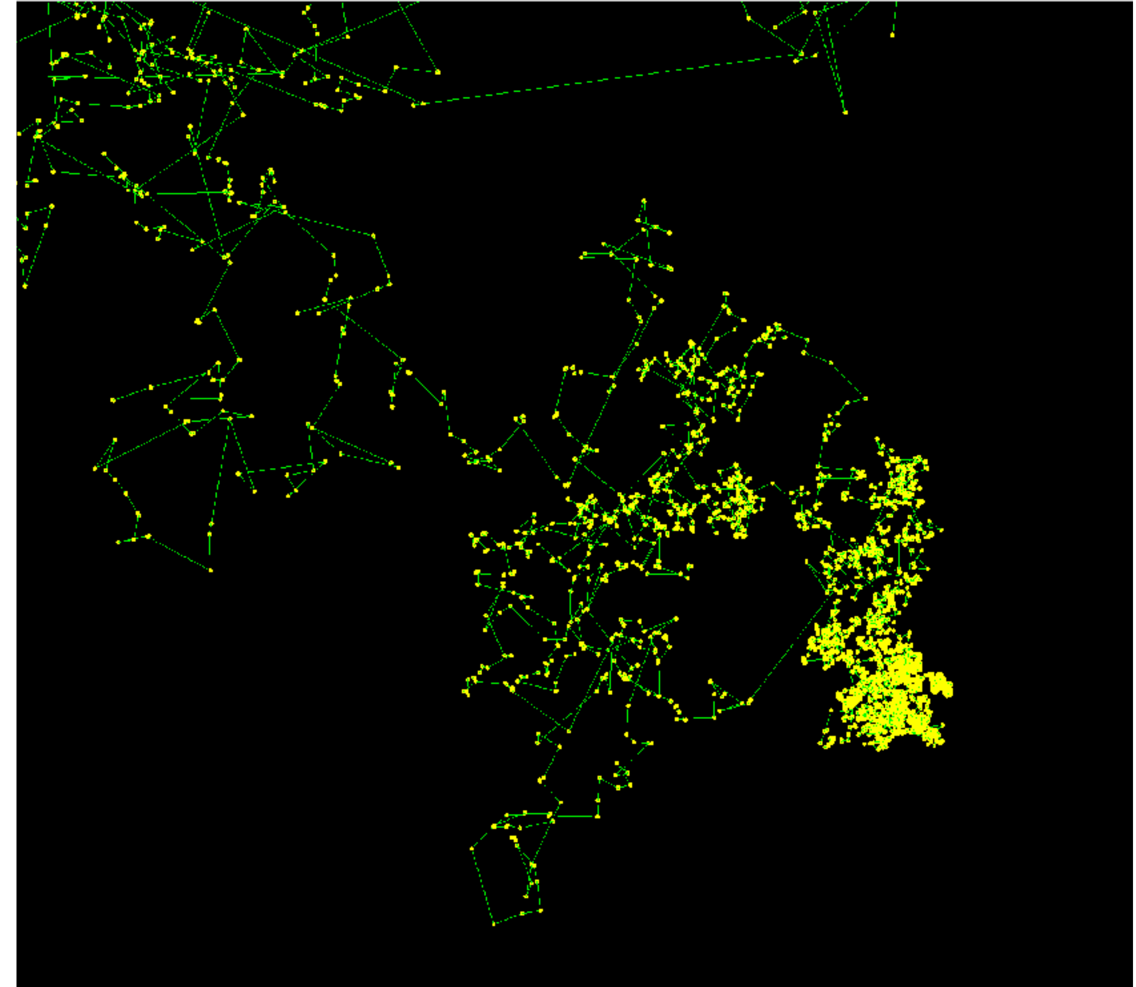
- VecGeom linked into Celeritas externally
  - CUDA device object library/separable compilation for GPU support
  - Invaluable support from VecGeom liasons
- Raytrace mini-app
  - Integration test of VecGeom interface, LinearPropagator, MaterialTrackView
  - Each horizontal line is a CUDA thread


CMS raytrace, tracks initialized on CPU and traced on GPU

https://github.com/celeritas-project/celeritas/pull/40

Physics mini-app: Klein–Nishina photon transport

- Simplest **physical** simulation we can run
  - Photon-only transport (electrons created but immediately killed and locally deposited)
  - Single process (inelastic scattering neglecting binding energy)
  - Single infinite material (aluminum)
  - 100 MeV monodirectional point source
- Celeritas (GPU and CPU-only versions), Geant4
- Verification: axial energy deposition



https://github.com/celeritas-project/celeritas/pull/51

Physics mini-app: features

- Stepping kernel is parallel over tracks (one launch per step)
  - Interaction length sampling with (uniform-in-energy) cross section calculation
  - Movement and direction updates
  - Full Klein–Nishina Compton scattering (neglecting binding energy)
  - Allocation and construction of secondary particle
  - Detector hit allocation and assignment (energy, position, direction, time, event)
- Tallying kernel to process “hit list” into bins
- Reduction kernel for termination criteria and diagnostics

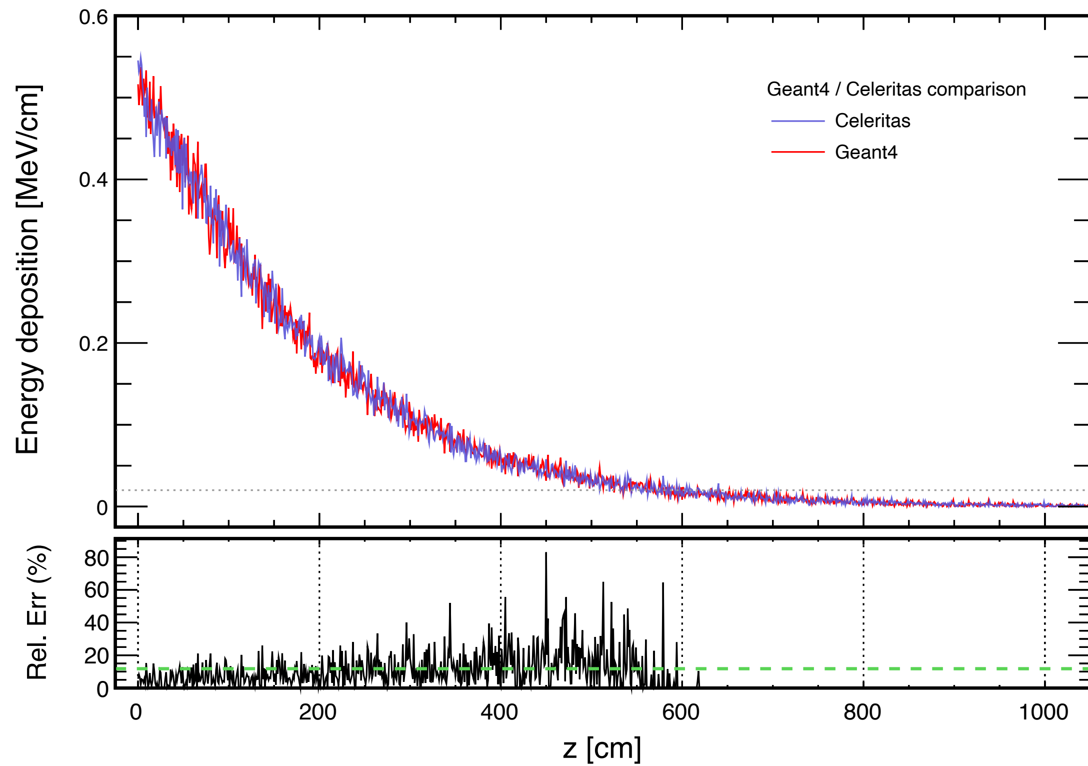
```
//---------------------------------------------------------------------------/
/*!
 * Perform a single interaction per particle track.
 *
 * The interaction:
 * - Clears the energy deposition
 * - Samples the KN interaction
 * - Allocates and emits a secondary
 * - Kills the secondary, depositing its local energy
 * - Applies the interaction (updating track direction and energy)
 */
__global__ void iterate_kn(ParamPointers const             params,
                           StatePointers const             states,
                           SecondaryAllocatorPointers const secondaries,
                           DetectorPointers const          detector)
{
    SecondaryAllocatorView allocate_secondaries(secondaries);
    DetectorView           detector_hit(detector);
    PhysicsArrayCalculator calc_xs(params.xs);

    for (int tid = blockIdx.x * blockDim.x + threadIdx.x;
         tid < static_cast<int>(states.size());
         tid += blockDim.x * gridDim.x)
    {
        // Skip loop if already dead
        if (!states.alive[tid])
        {
            continue;
        }

        // Construct particle accessor from immutable and thread-local data
        ParticleTrackView particle(
            params.particle, states.particle, ThreadId(tid));
        RngEngine rng(states.rng, ThreadId(tid));

        // Move to collision
        {
            // Calculate cross section at the particle's energy
            real_type                          sigma = calc_xs(particle);
            ExponentialDistribution<real_type> sample_distance(sigma);
            // Sample distance-to-collision
            real_type distance = sample_distance(rng);
            // Move particle
            axpy(distance, states.direction[tid], &states.position[tid]);
            // Update time
            states.time[tid] += distance * unit_cast(particle.speed());
        }

        Hit h;
        h.pos    = states.position[tid];
        h.thread = ThreadId(tid);
        h.time   = states.time[tid];

        if (particle.energy() < KleinNishinaInteractor::min_incident_energy())
        {
            // Particle is below interaction energy
            h.dir              = states.direction[tid];
            h.energy_deposited = particle.energy();

            // Deposit energy and kill
            detector_hit(h);
            states.alive[tid] = false;
            continue;
        }

        // Construct RNG and interaction interfaces
        KleinNishinaInteractor interact(params.kn_interactor,
                                        particle,
                                        states.direction[tid],
                                        allocate_secondaries);

        // Perform interaction: should emit a single particle (an electron)
        Interaction interaction = interact(rng);
        CHECK(interaction);
        CHECK(interaction.secondaries.size() == 1);

        // Deposit energy from the secondary (effectively, an infinite energy
        // cutoff)
        {
            const auto& secondary = interaction.secondaries.front();
            h.dir                 = secondary.direction;
            h.energy_deposited    = secondary.energy;
            detector_hit(h);
        }

        // Update post-interaction state (apply interaction)
        states.direction[tid] = interaction.direction;
        particle.energy(interaction.energy);
    }
}
```

Stepping kernel

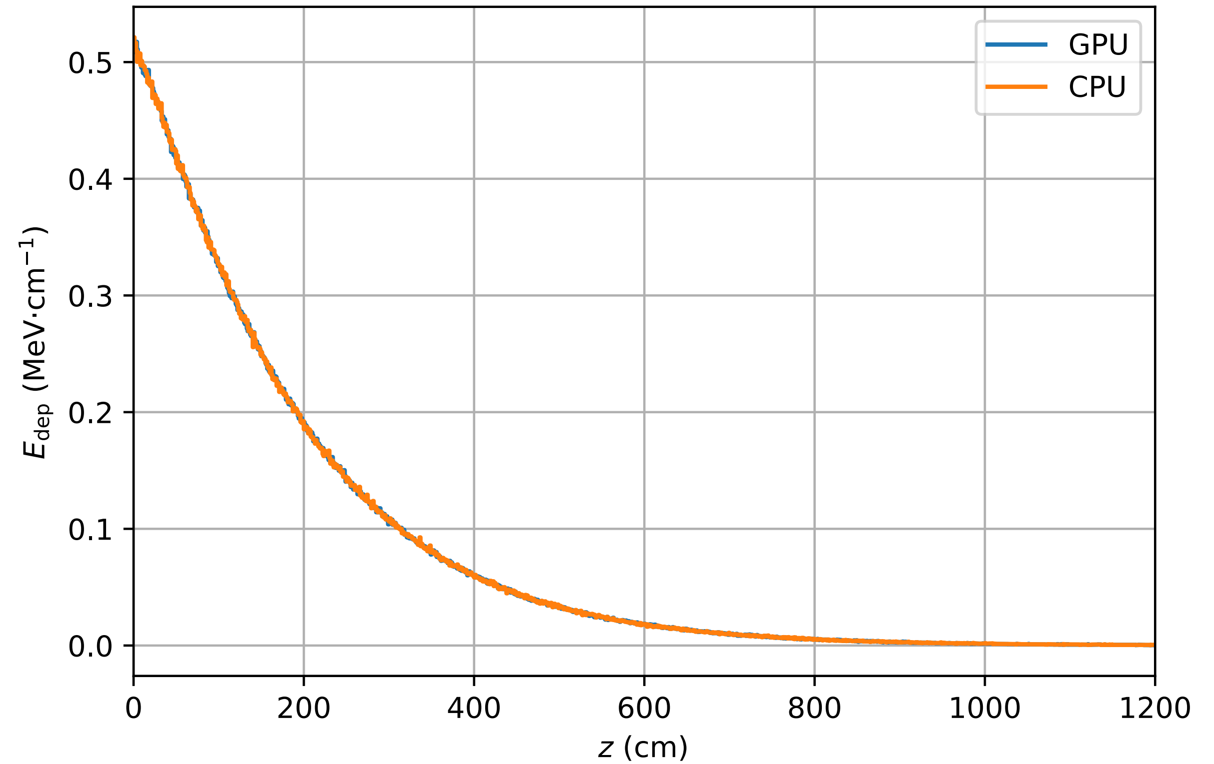
EXASCALE COMPUTING PROJECT

Celeritas reproduces Geant4 physics results (so far...)

Celeritas GPU vs Geant4 (64K tracks)



Celeritas GPU vs CPU (4M tracks)

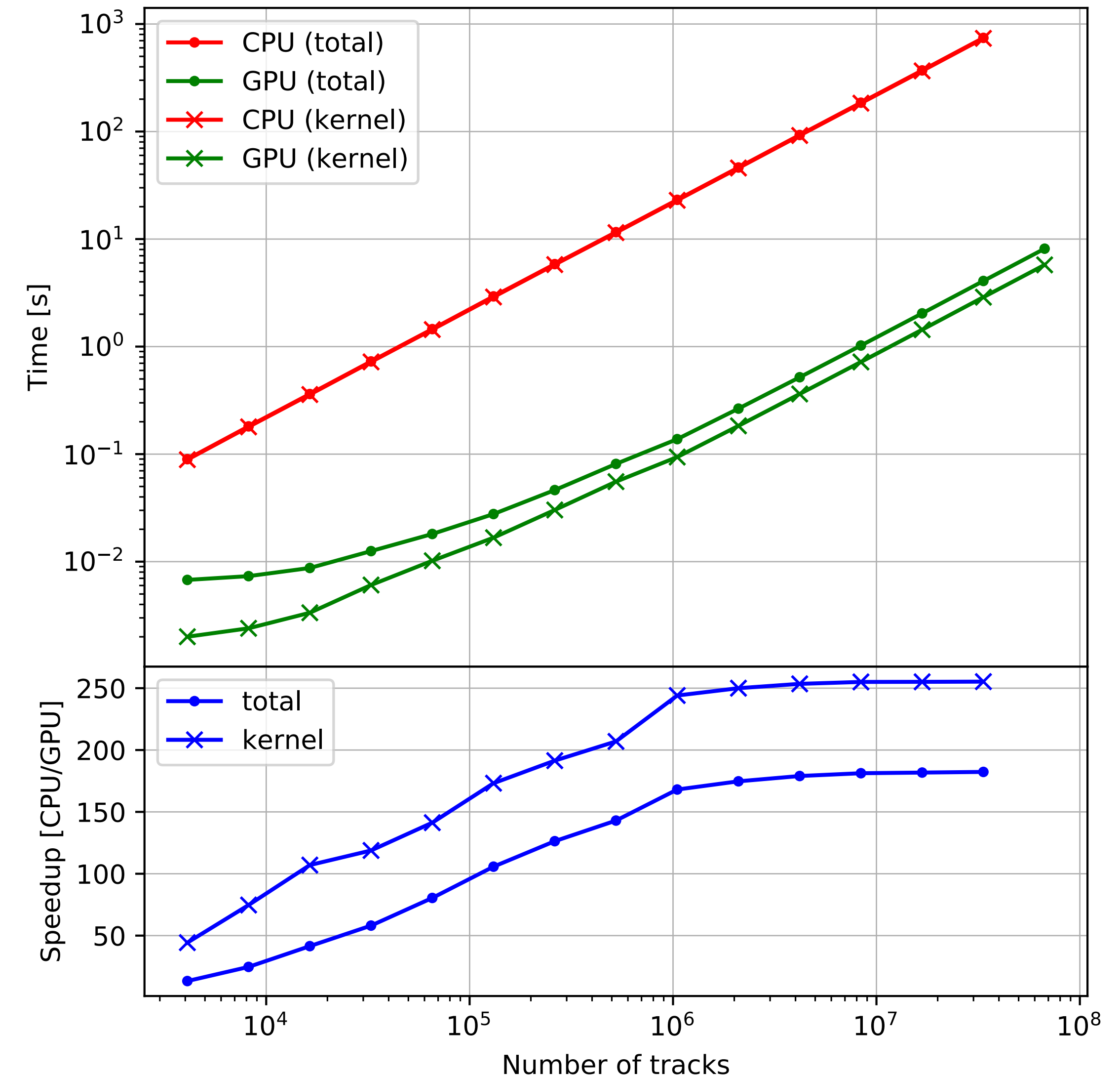


Successful verification of Celeritas physics

Celeritas CPU/GPU comparison

- Identical kernels, sequential events on CPU and parallel events on GPU
- Primary GPU kernel is an entire "step"
- Single core of Intel "Cascade Lake" Xeon (2.3 GHz)
  vs single Nvidia Tesla V100 (1.53 GHz)
- **~150×** overall speedup for 1M tracks in parallel

Likely a representative upper bound for raw performance



CUDA 10.1: `-O3 --use_fast_math`
GCC 8.3: `-O3 -march=skylake-avx512 -mtune=skylake-avx512`

Summary and Future Work

Summary of accomplishments *(first code commit: June 16, 2020)*

- Infrastructure
  - ✓ Established code base, procedures, and project tracking: https://github.com/celeritas-project
  - ✓ Extensible build system that supports unit testing and Continuous Integration (CI) using modern CMake
- Geometry
  - ✓ CUDA-enabled VecGeom integrated and distributed as a Spack package
  - ✓ Track initialization ported to device
  - ✓ Ray-tracing tests and application driver developed
- Physics
  - ✓ Integrated ability to source events from common HEP event-generators (HEPMC3, Pythia)
  - ✓ Built device memory pool allocators for efficient generation of secondary particles in device memory
  - ✓ Built, tested, and profiled simple event loop (Klein–Nishina interactions) on device
  - ✓ Built infrastructure and interactors for EM physics models

Work plan for next 3 months

- Work on full set of standard EM physics models is proceeding
  - Cross section tables
  - Physical materials
  - EM physics interaction models
- Integration of geometry tracker into transport app
- Implementation of multi-process transport loop
  - Scheduler for full process loop
  - Device-side optimization
- Background EM field integration

Target delivery: February 1

- Base:
  - Multi-process EM physics
  - Integrated geometry with material definitions
- Stretch:
  - Constant EM field integrator

July 1 milestone: EM Physics hit-generation application

- Features:
  - Functionally representative EM Physics list
  - Constant background EM field integration
  - Full VecGeom integration
  - Sensitive detector hit-scoring
- Objective:
  - Perform detailed performance analysis of transport loop on device
  - Establish baseline Figure-of-Merit in events simulated per wall-clock-time
  - Identify risks to performance and robustness