Celeritas: Status and Plans

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

HSF Detector Simulation Working Group Meeting

12 April, 2021

EXASCALE COMPUTING PROJECT

National Nuclear Security Administration

U.S. DEPARTMENT OF ENERGY | Office of Science

Outline

- Project Overview
- Technical Update
- 5-Year Plan

Celeritas Project

Celeritas project objective

Become the next-generation particle transport backend for LHC detector simulations

1. Why is this capability needed?
   - Current high-fidelity, time-dependent, detector energy deposition simulations will not scale to proposed 10× LHC 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

HEP/ASCR Celeritas core team

- ANL—Particle Transport Group
  - Amanda Lund
- Fermilab—Physics and Detector Simulation Group
  - Philippe Canal*, Soon Yung Jun*, Guilherme Lima*
- LBL—ATLAS Group
  - Vincent Pascuzzi*†
- ORNL—HPC Methods and Nuclear Applications Group
  - Tom Evans* (**PI**), Seth Johnson (**Code Lead**), Stefano Tognini†

- ASCR ExaSMR Team
- HEP

* Leveraged through other project funding
† Postdoc

Technical Update

Technical update: overview of first-year goals

- ✓Established initial code base with rigorous documentation, review, and testing requirements
- ✓Built technical foundation for GPU- and CPU-friendly physics development
- ✓Developed key algorithms for GPU HEP transport
- ✓Completed initial EM physics components
- ✓Characterized initial GPU/CPU performance

Source code

Language	Files	Comment	Code
C/C++ Header	300	8900	11781
C++	88	2513	5986
CMake	19	429	2048
CUDA	14	401	922
Python	5	253	387
SWIG	1	30	66
Total	**427**	**12526**	**21190**

Test code

Language	Files	Comment	Code
C++	79	1892	7401
C/C++ Header	32	637	1593
CUDA	12	237	772
Total	**123**	**2766**	**9766**

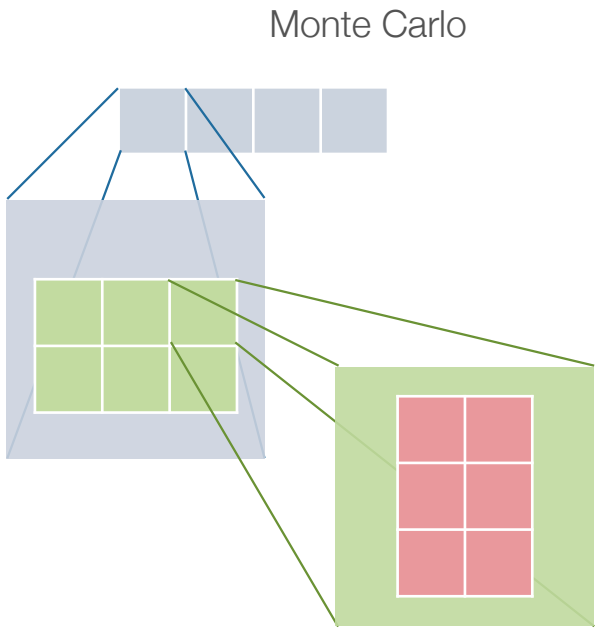
First commit: June 2020

Data model for rapid development of complex GPU physics

- Monte Carlo particle transport methods have **deep hierarchical heterogeneous data structures**
  - Material properties (Material/element/nuclide)
  - Macroscopic cross sections (Particle/material/process/energy)
  - Differential cross sections (Element/incident/exiting)
- Typical GPU algorithms designed for **wide homogeneous data**
- And can we reuse both data and execution code on both CPU and GPU?

Celeritas solution: new paradigm for GPU-friendly physics

Data model for rapid development of complex GPU physics



- Monte Carlo particle transport methods have **deep hierarchical heterogeneous data structures**
  - Material properties (Material/element/nuclide)
  - Macroscopic cross sections (Particle/material/process/energy)
  - Differential cross sections (Element/incident/exiting)
- Typical GPU algorithms designed for **wide homogeneous data**
- And can we reuse both data and execution code on both CPU and GPU?

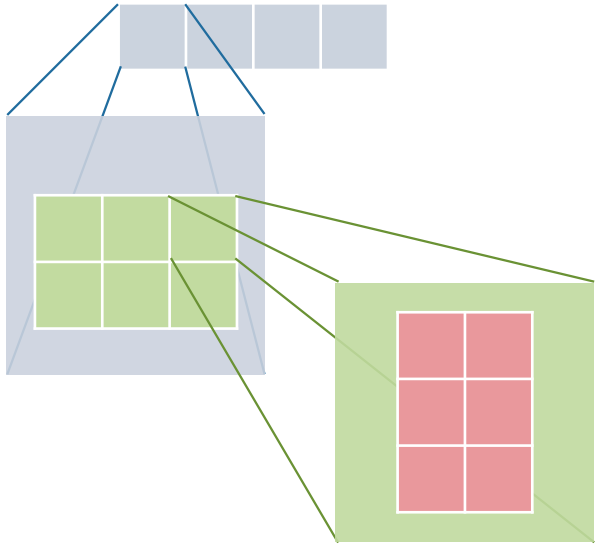
Celeritas solution: new paradigm for GPU-friendly physics

Data model for rapid development of complex GPU physics

- Monte Carlo particle transport methods have **deep hierarchical heterogeneous data structures**
  - Material properties (Material/element/nuclide)
  - Macroscopic cross sections (Particle/material/process/energy)
  - Differential cross sections (Element/incident/exiting)
- Typical GPU algorithms designed for **wide homogeneous data**
- And can we reuse both data and execution code on both CPU and GPU?

Celeritas solution: new paradigm for GPU-friendly physics

Monte Carlo

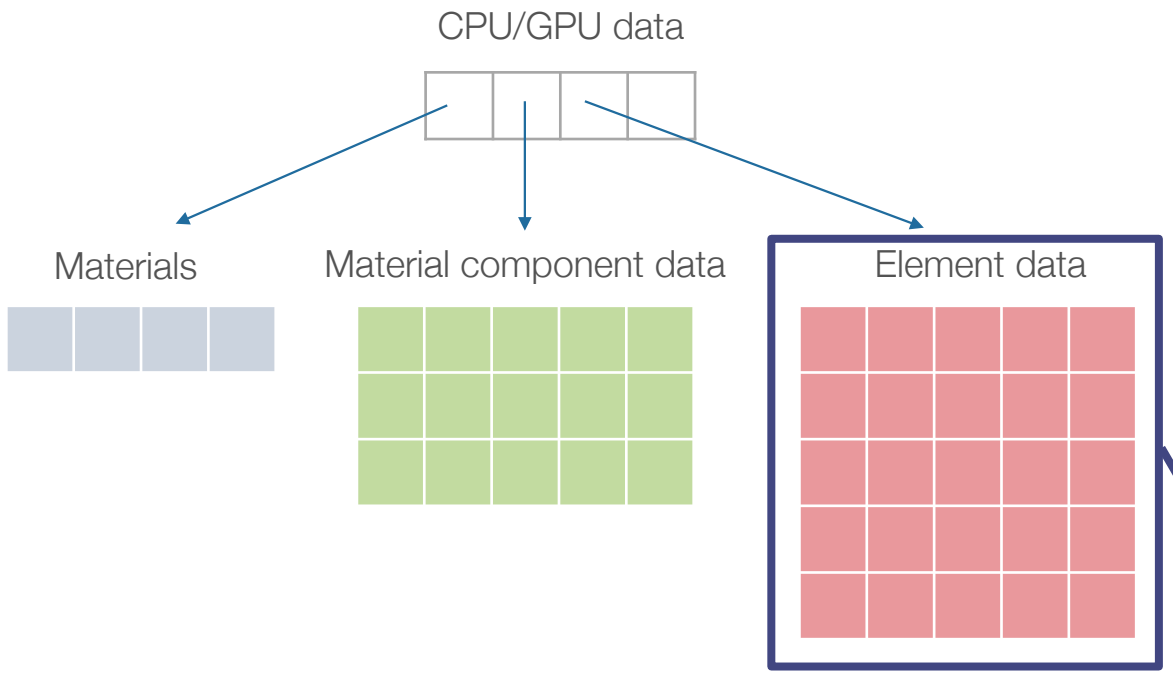


Typical GPU Application

Data model for rapid development of complex GPU physics

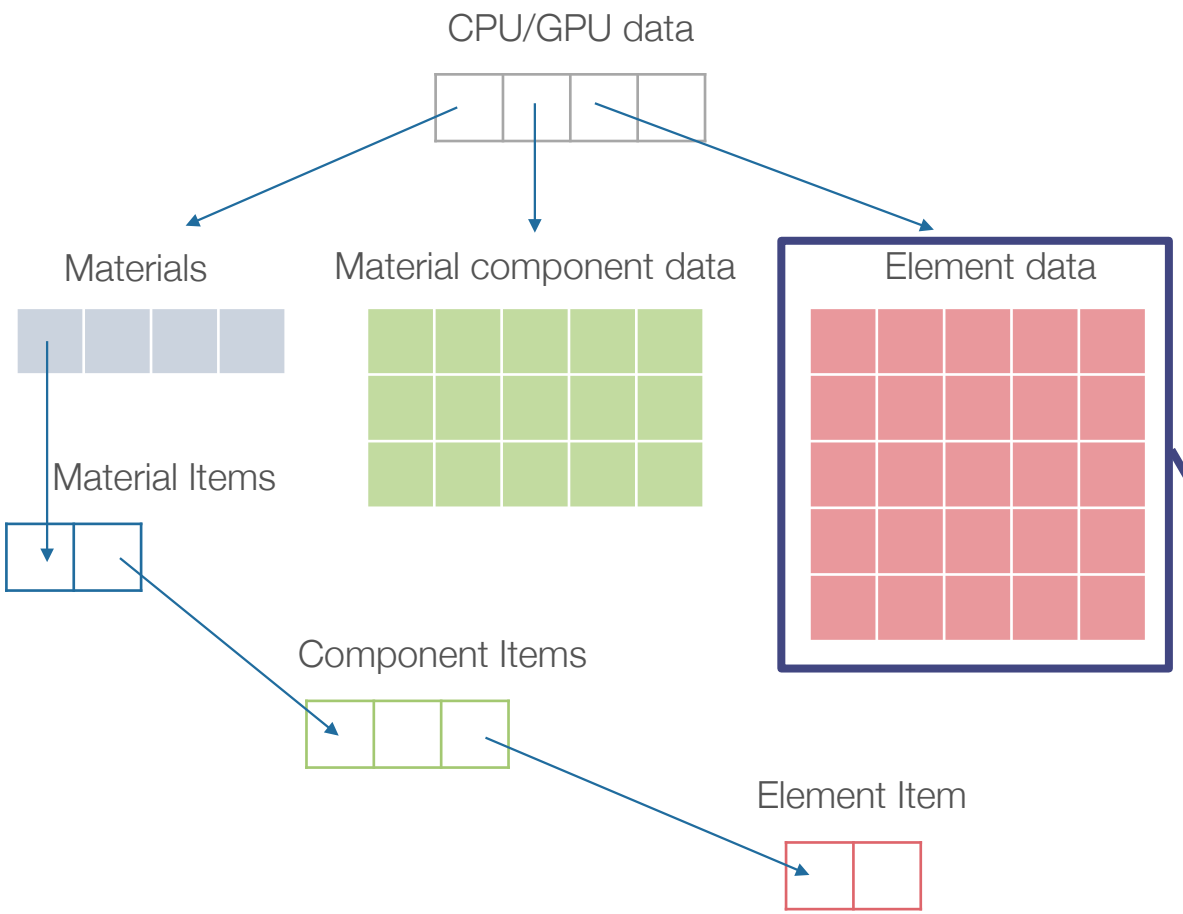
```
struct Element
{
    int            atomic_number;
    units::AmuMass atomic_mass;
};

struct MatElementComponent
{
    ItemId<Element> element;
    double          fraction;
};

struct Material
{
    double      number_density;
    double      temperature;
    MatterState matter_state;
    ItemRange<MatElementComponent> components;
};

template<Ownership W, MemSpace M>
struct MaterialParamsData
{
    template<class T>
    using Items = celeritas::Collection<T, W, M>;

    Items<Element>             elements;
    Items<MatElementComponent> elcomponents;
    Items<Material>            materials;
    unsigned int               max_elcomponents;

    template<Ownership W2, MemSpace M2>
    MaterialParamsData& operator=(
        const MaterialParamsData<W2, M2>& other);
}
```

Data model for rapid development of complex GPU physics



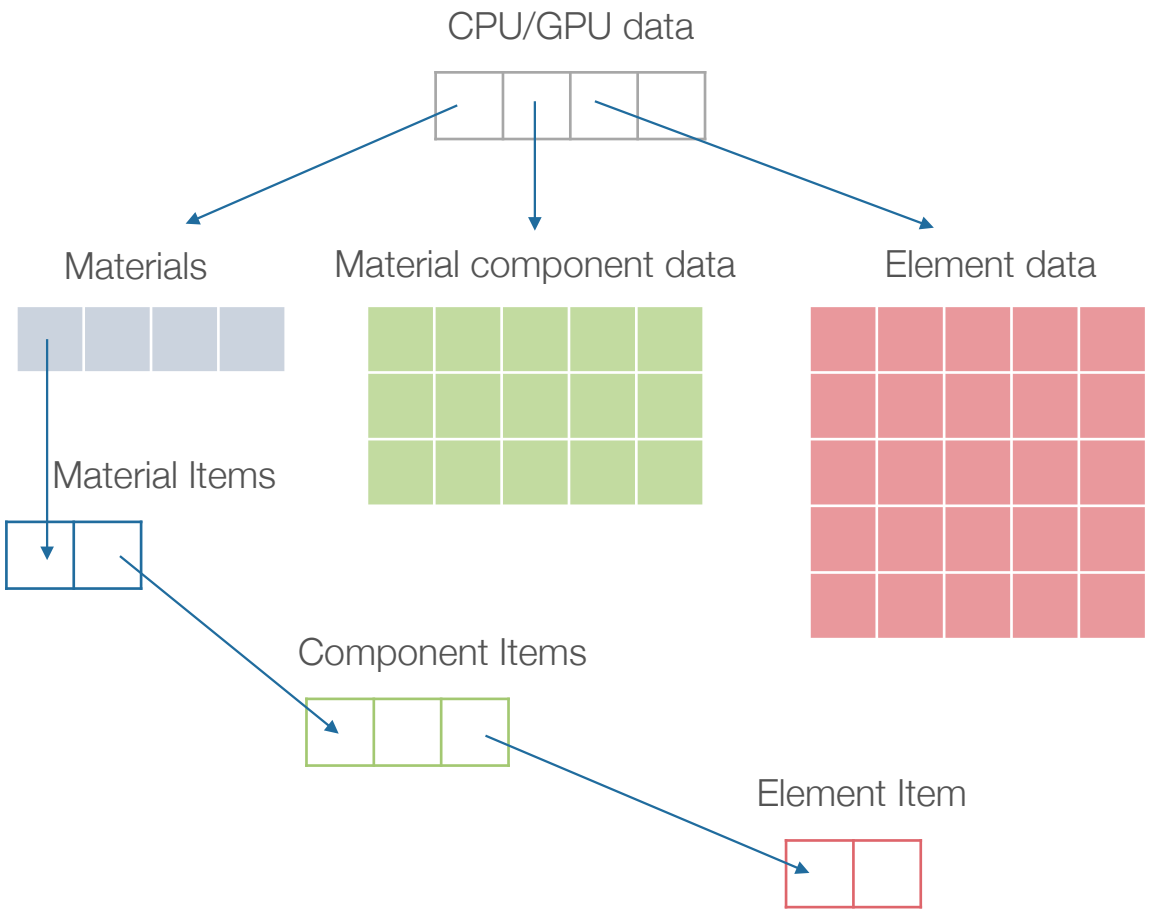

```
struct Element
{
    int            atomic_number;
    units::AmuMass atomic_mass;
};

struct MatElementComponent
{
    ItemId<Element> element;
    double          fraction;
};

struct Material
{
    double      number_density;
    double      temperature;
    MatterState matter_state;
    ItemRange<MatElementComponent> components;
};

template<Ownership W, MemSpace M>
struct MaterialParamsData
{
    template<class T>
    using Items = celeritas::Collection<T, W, M>;

    Items<Element>             elements;
    Items<MatElementComponent> elcomponents;
    Items<Material>            materials;
    unsigned int               max_elcomponents;

    template<Ownership W2, MemSpace M2>
    MaterialParamsData& operator=(
        const MaterialParamsData<W2, M2>& other);
}
```

Data model for rapid development of complex GPU physics




```
struct Element
{
    int            atomic_number;
    units::AmuMass atomic_mass;
};

struct MatElementComponent
{
    ItemId<Element> element;
    double          fraction;
};

struct Material
{
    double      number_density;
    double      temperature;
    MatterState matter_state;
    ItemRange<MatElementComponent> components;
};

template<Ownership W, MemSpace M>
struct MaterialParamsData
{
    template<class T>
    using Items = celeritas::Collection<T, W, M>;

    Items<Element>             elements;
    Items<MatElementComponent> elcomponents;
    Items<Material>            materials;
    unsigned int               max_elcomponents;

    template<Ownership W2, MemSpace M2>
    MaterialParamsData& operator=(
        const MaterialParamsData<W2, M2>& other);
}
```

Data model for rapid development of complex GPU physics



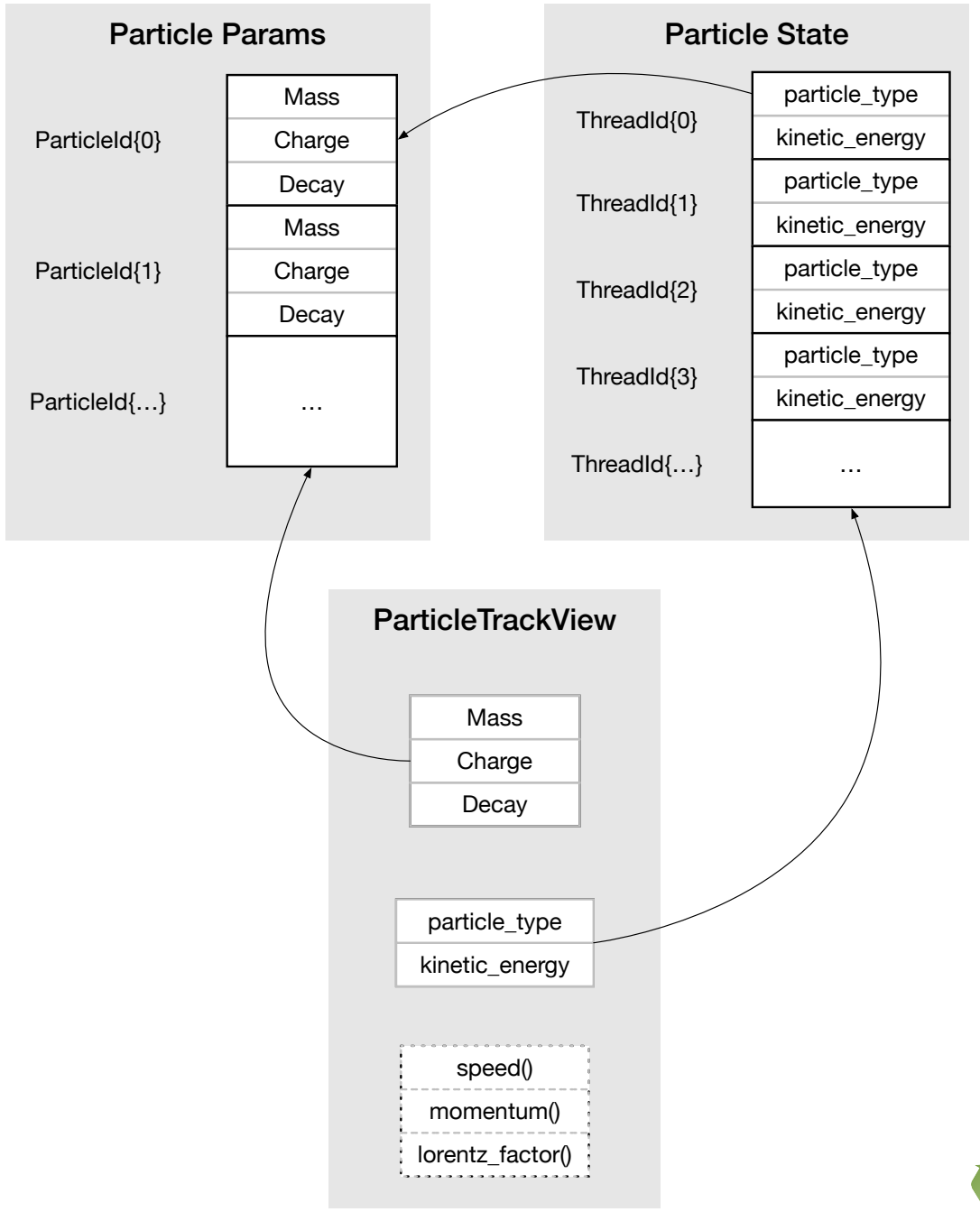
- Easily assemble data on CPU
- Define data structures *once*
- Data and execution on both CPU and GPU
- **Payoff:** safe and effective framework for physicists to implement and test GPU-compatible physics

Current hierarchical physics data in Celeritas

- Material: materials→element components→elements
- Physics: particles→processes→{tables→materials→grids, models}
- Livermore PE: elements→{grids, subshells→{grids, params}}
- Seltzer–Berger tables: elements→2D grids
- Rayleigh parameters: elements→params

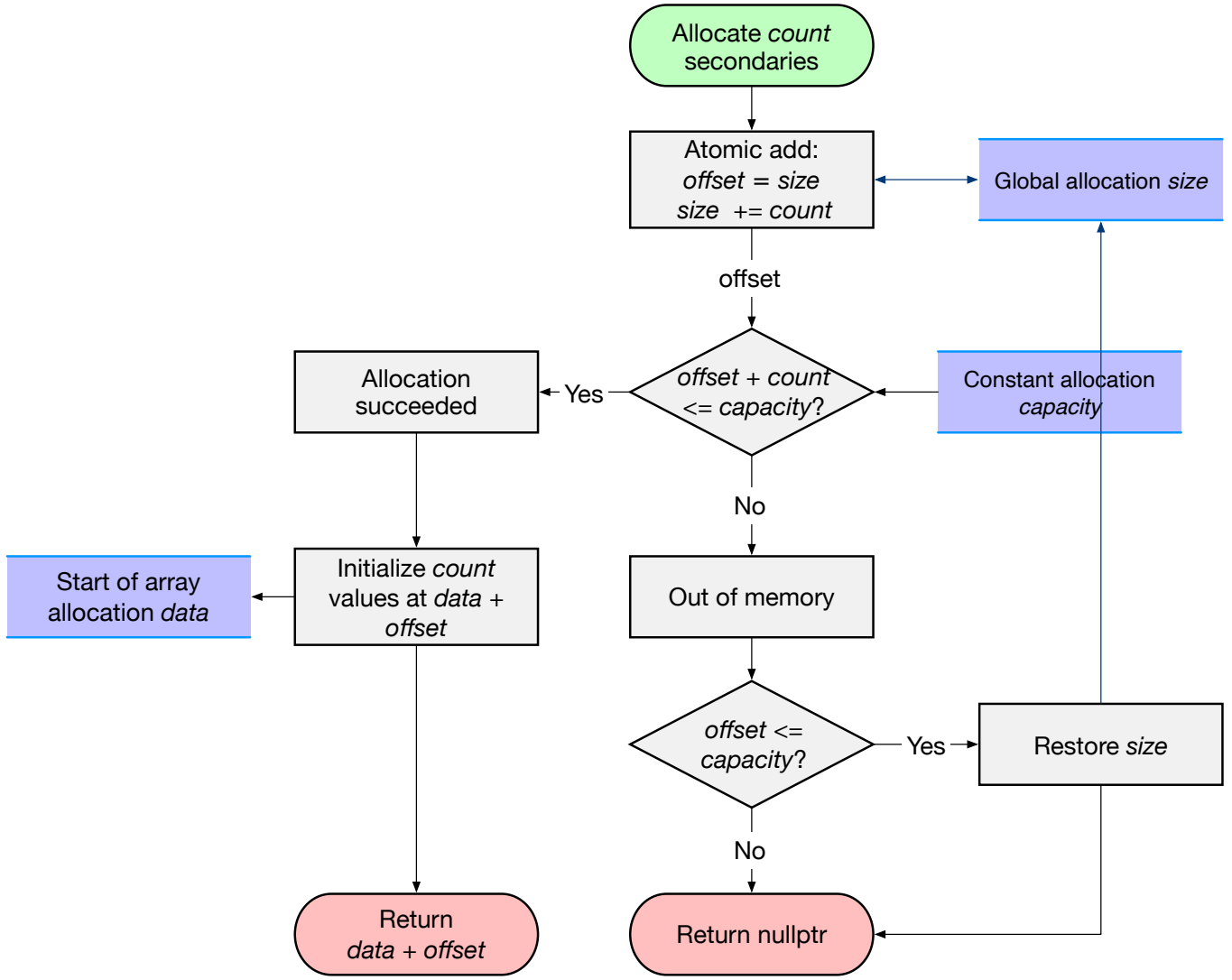
Software abstractions for portability

- 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



Stack allocator for fast on-device secondary/hit construction

- Novel algorithm for satisfying **performance** and **reproducibility** requirements
- Unknown number of tracks (running in parallel, one per thread) creating unknown number of secondaries per track
- Finite amount of GPU memory preallocated
- Failure to allocate secondaries for a thread *must* result in graceful failure to avoid advancing RNG (random number generator) state



Physics capabilities (GPU)

✓Discrete process interactions including dynamic allocation of secondaries

✓Continuous slowing-down energy loss

✓Multiple competing processes

✓Multiple materials

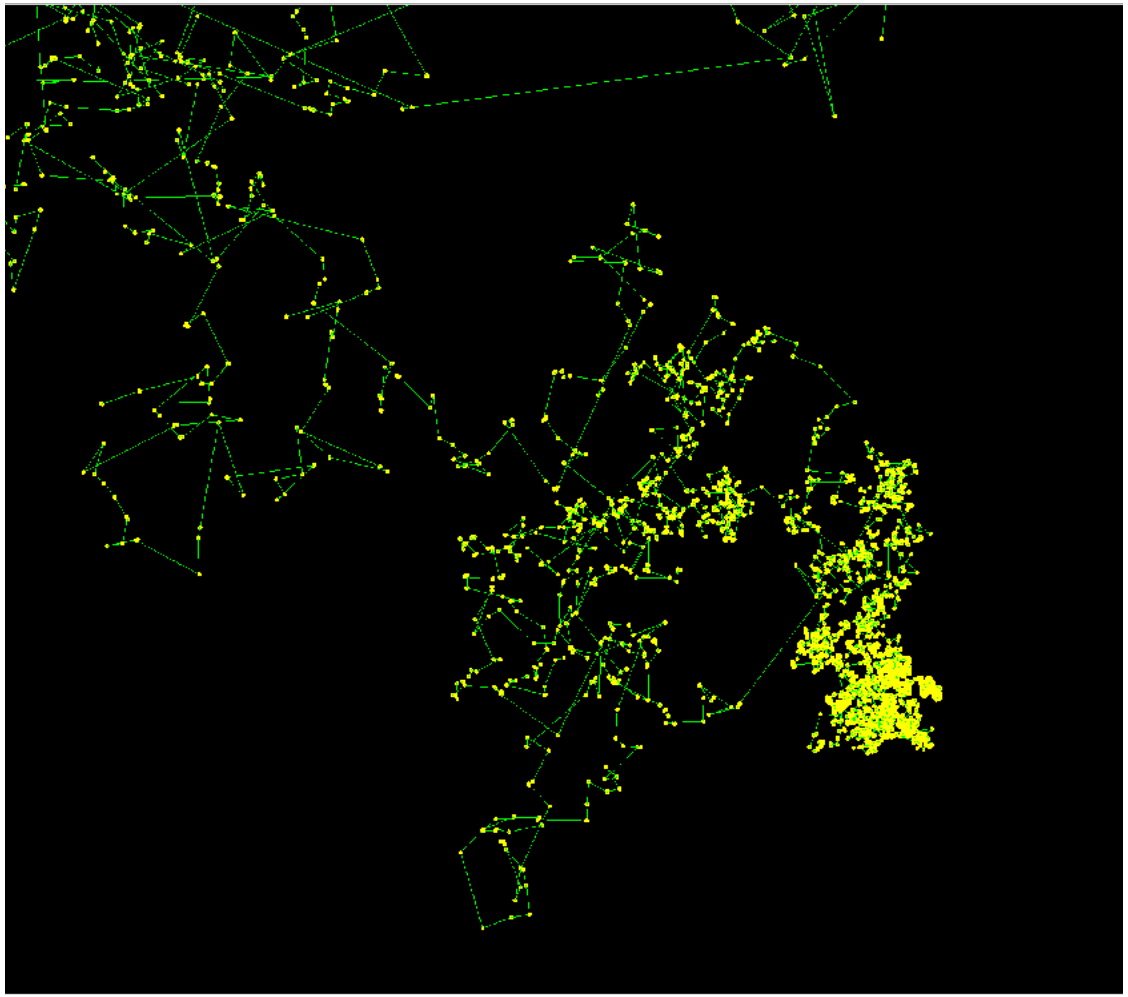
→Uniform magnetic field

→Integrated stepping loop

	Process (Model)	Status
γ	Conversion (Bethe-Heitler)	Implemented
	Compton scattering (Klein-Nishina)	Verified
	Photoelectric (Livermore)	Implemented
	Rayleigh scattering (Livermore)	In progress
e	Ionization (Møller-Bhabha)	Implemented
	Bremsstrahlung (Seltzer-Berger)	In progress
	Multiple scattering	Deferred
μ		Deferred

Physics mini-app: Klein–Nishina photon transport (Nov. 2020)

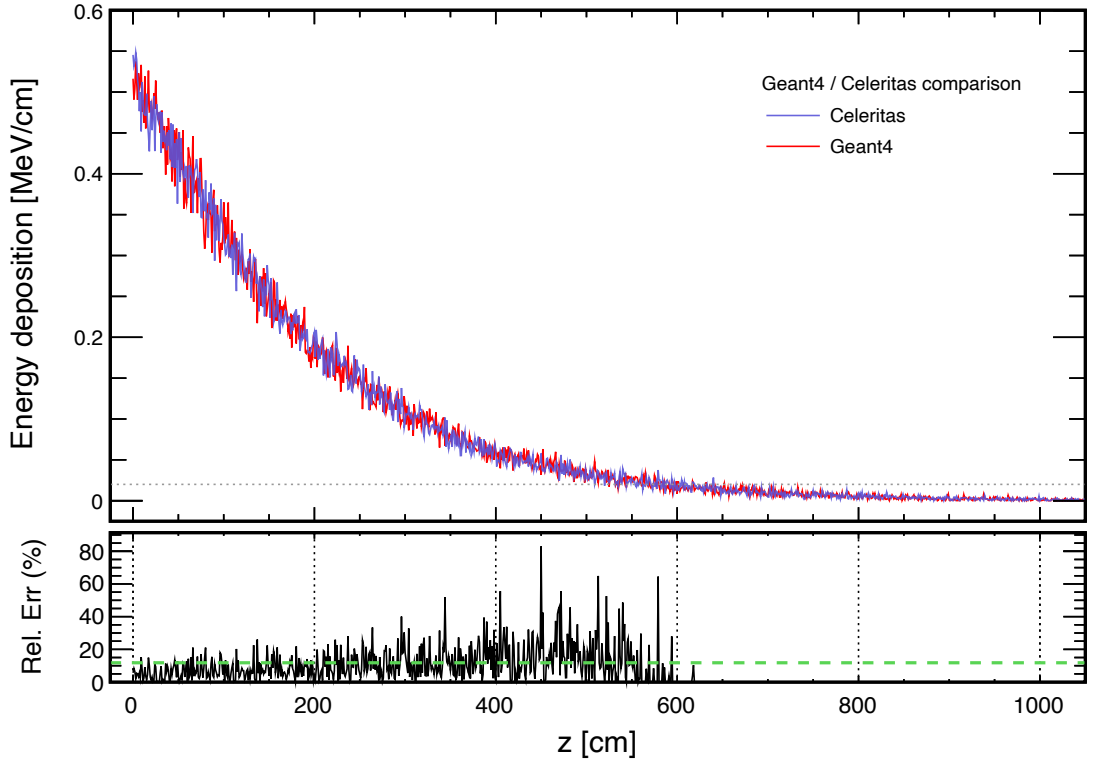
- Simplest **physical** simulation we can run
  - Photon-only transport (electrons created but immediately killed and locally deposited)
  - Single process (inelastic scattering neglecting binding energy)
  - Single infinite material (aluminum)
  - 100 MeV monodirectional point source
- Celeritas (GPU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



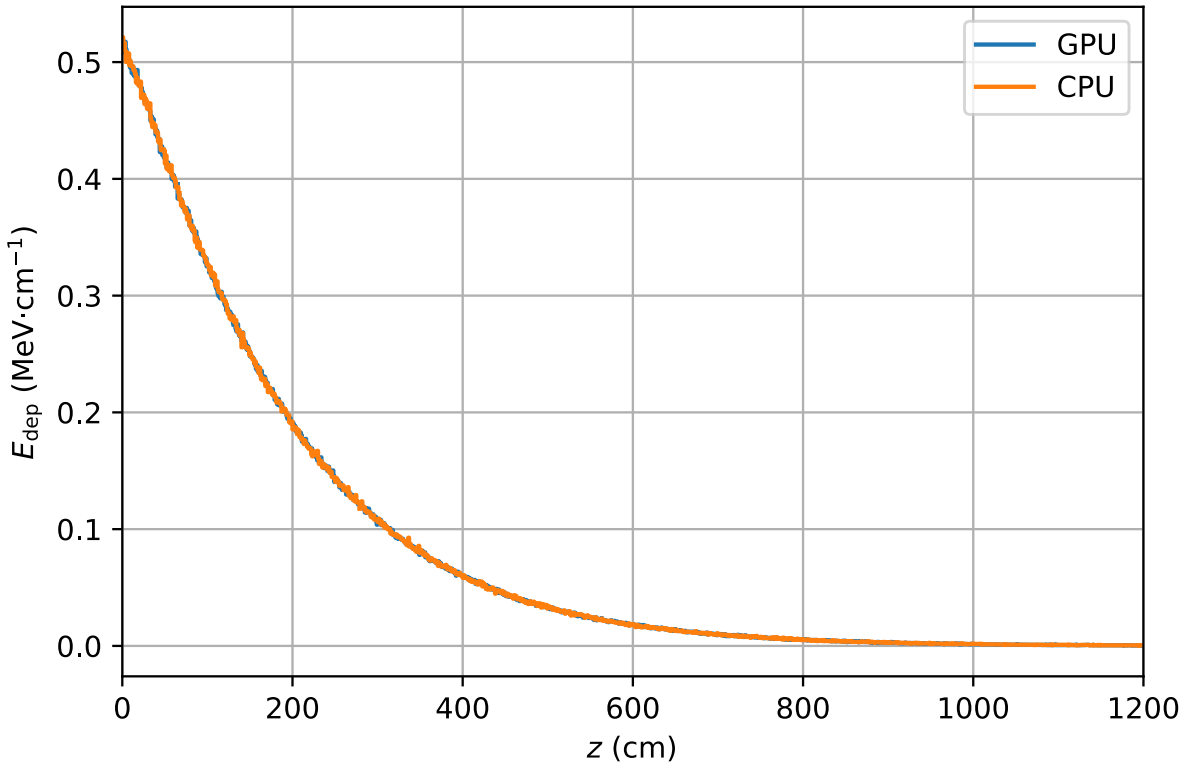
https://github.com/celeritas-project/celeritas/pull/51

Celeritas reproduces Geant4 physics results (Nov. 2020)

Celeritas GPU vs Geant4 (64K tracks)



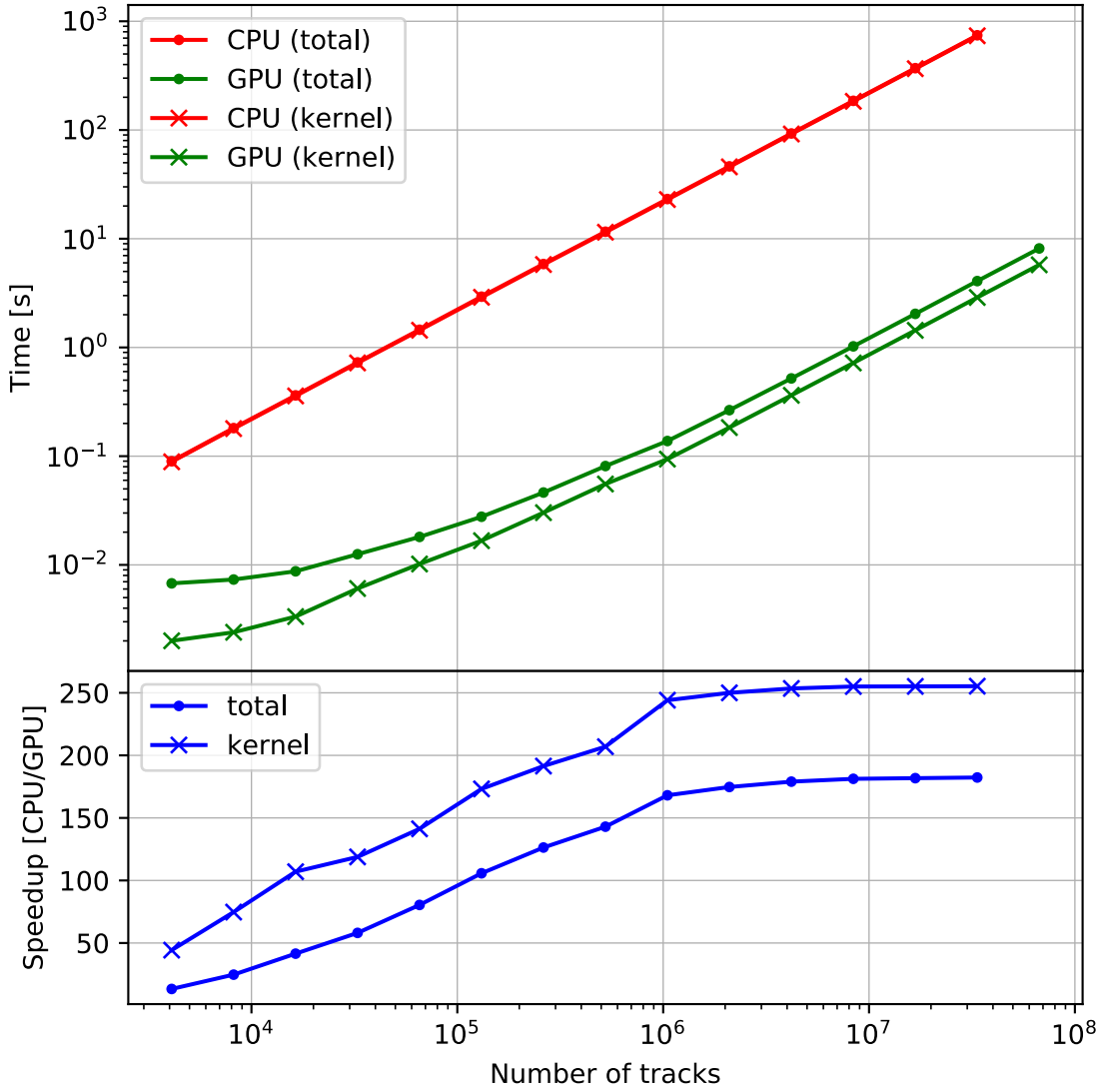
Celeritas GPU vs CPU (4M tracks)



First successful verification of Celeritas physics

Initial mini-app CPU/GPU results (Nov. 2020)

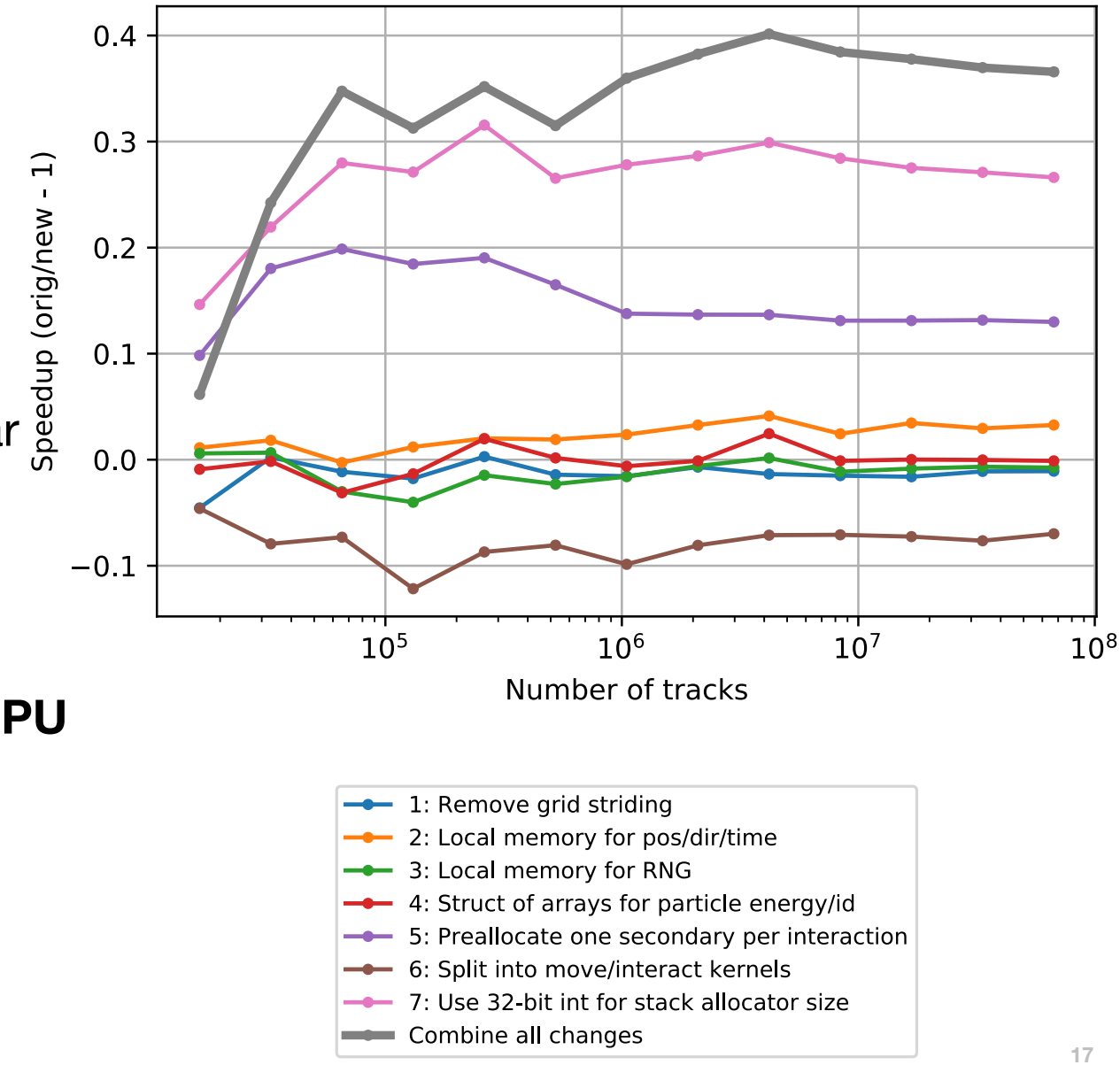
- Shared Celeritas physics code
- Sequential events on CPU and parallel events on GPU
- Primary GPU kernel is an entire “step”
- Single core of Intel “Cascade Lake” Xeon (2.3 GHz) vs single Nvidia Tesla V100 (1.53 GHz)
- **~150×** overall speedup for 1M tracks in parallel



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

Performance enhancements (Feb. 2021)

- Code structure facilitates performance exploration
- “Non-traditional” bottlenecks: atomics and irregular memory access
- Combining different “tweaks”: **38%** GPU speedup in mini-app
- **~220× overall mini-app speedup compared to CPU**

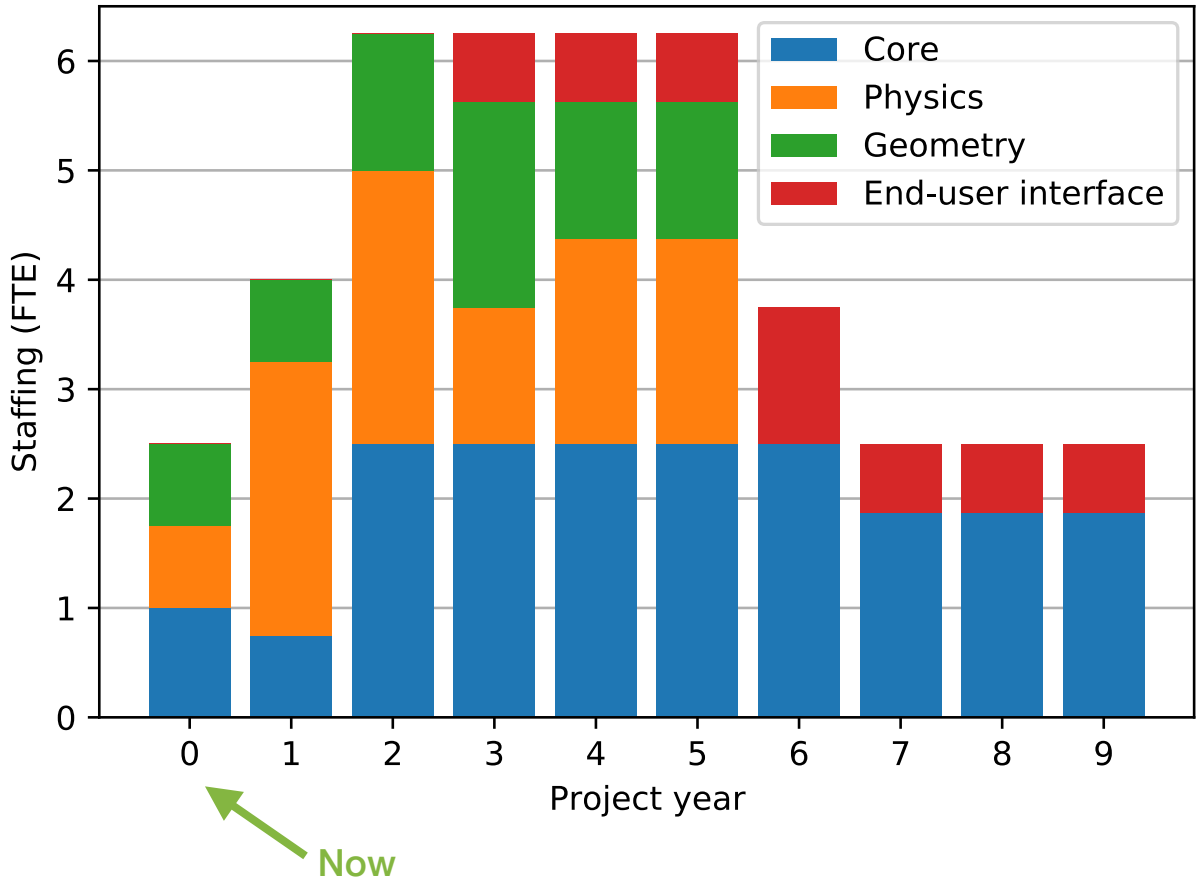


5-Year Plan

5-Year Celeritas Plan

- Period until July 1, 2021
  - Form core team and primary collaborations
  - Integrate into Geant/HEP community
  - Implement fundamental infrastructure
- 3-year plan (July 1, 2021–July 1, 2024)
  - Incorporate Celeritas EM physics into HEP experimental workflows
- 2-year continuation (through 2026)
  - Celeritas user app/framework with complete standard high-energy physics

Proposed project staffing and milestones



Year	Principal milestone
0	Start-up and application demonstration
1	Complete EM physics
2	EM physics integration into experimental workflows
3	Validation of EM physics
4	Hadronic physics for LHC
5	Validation of LHC physics
6+	Transition to operations; integration into federated networks

Principal milestones per year

July 1, 2021 — July 1, 2022

- Prototype performance portability
  - Gather requirements for performance portability
  - Kokkos/HIP/SYCL
  - Performance portable geometry
- Complete EM physics
- Integration with experimental detector output post-processing
  - Client-defined scoring regions

July 1, 2022 — July 1, 2023

- Baselining EM physics simulation runtime performance
  - 32-bit (fast) vs 64-bit
- Validation of EM physics
  - Benchmarks
  - CMS or other experiments
- Integrated Celeritas + Geant4 workflow for EM Physics

July 1, 2023 — July 1, 2024

- Integration of expanded physics to support experimental campaigns
  - e.g. hadronic physics
- Demonstration of full event simulation in Celeritas
- Integration as a training data generator
  - Support external AI/ML fast simulations

QUESTIONS?