ORNL/TM-2023/3022

Celeritas R&D Report: Accelerating Geant4

Seth R. Johnson[1]
Elliott Biondo[1]
Julien Esseiva[4]
Soon Yung Jun[2]
Guilherme Lima[2]
Amanda Lund[3]
Ben Morgan[5]
Stefano C. Tognini[1]
Philippe Canal[2]
Marcel Demarteau[1]
Thomas Evans[1]
Paul Romano[3]

January 5, 2024

[1]Oak Ridge National Laboratory, Oak Ridge, TN, USA
[2]Fermi National Accelerator Laboratory, Batavia, IL, USA
[3]Argonne National Laboratory, Lemont, IL, USA
[4]Lawrence Berkeley National Laboratory, Berkeley, CA, USA
[5]University of Warwick, Coventry, United Kingdom

OAK RIDGE National Laboratory

ORNL IS MANAGED BY UT–BATTELLE LLC FOR THE US DEPARTMENT OF ENERGY

ORNL/TM-2023/3022

Computational Sciences & Engineering Division

CELERITAS R&D REPORT: ACCELERATING GEANT4

Seth R. Johnson[1]
Elliott Biondo[1]
Julien Esseiva[4]
Soon Yung Jun[2]
Guilherme Lima[2]
Amanda Lund[3]
Ben Morgan[5]
Stefano C. Tognini[1]
Philippe Canal[2]
Marcel Demarteau[1]
Thomas Evans[1]
Paul Romano[3]

January 5, 2024

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, TN 37831
managed by
UT–BATTELLE LLC
for the
US DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

ABSTRACT

Celeritas is a new Monte Carlo (MC) detector simulation code designed for computationally intensive applications on high-performance heterogeneous architectures. In the past two years Celeritas has advanced from prototyping a Graphics Processing Unit (GPU)-based single physics model in infinite medium to implementing a full set of electromagnetic (EM) physics processes in complex geometries. The current release of Celeritas, version 0.4, has incorporated full device-based navigation, an event loop in the presence of magnetic fields, and detector hit scoring. New functionality incorporates a scheduler to offload electromagnetic physics to the GPU within a Geant4-driven simulation, enabling straightforward integration of Celeritas into the high energy physics (HEP) experimental frameworks CMSSW and ATLAS FullSimLight. On the Perlmutter supercomputer, Celeritas performs EM physics between 3× and 18× faster using the machine's Nvidia GPUs compared to using only CPUs, corresponding to an electrical power efficiency up to a factor of 5. When running a multithreaded Geant4 ATLAS test beam application with full hadronic physics, using Celeritas to accelerate the EM physics results in an overall simulation speedup of 1.7–2.2× on GPU and 1.2× on CPU. In a CMS test application using $t\bar{t}$ events and the prototype Run 4 configuration, compared to Geant4 CPU, Celeritas with a Nvidia A100 improves overall throughput up to a factor of 2.7× but cannot be efficiently shared with more than 8 cores.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ASCR	Advanced Scientific Computing Research
BIH	bounding interval hierarchy
CMSSW	CMS software
CPU	Central Processing Unit
CSG	constructive solid geometry
DOE	U.S. Department of Energy
ECP	Exascale Computing Project
EM	electromagnetic
FOM	Figure of Merit
FTE	full time equivalent
GPU	Graphics Processing Unit
HEP	high energy physics
HF	hadronic forward
HL-LHC	High Luminosity Large Hadron Collider
I/O	input/output
LBNL	Lawrence Berkeley National Laboratory
LCF	Leadership Computing Facility
LHC	Large Hadron Collider
MC	Monte Carlo
MSC	multiple scattering
MT	multi-thread
ODE	ordinary differential equation
ORANGE	Oak Ridge Adaptable Nested Geometry Engine
ORNL	Oak Ridge National Laboratory
SD	sensitive detector
SIMD	single instruction, multiple data
SIMT	single instruction, multiple thread
TBB	Thread Building Block
TDP	Thermal Design Power
WLCG	Worldwide LHC Computing Grid

1. INTRODUCTION

The Celeritas particle transport code, started in early 2020 as an interdisciplinary multi-laboratory project, is an extensible library for HEP detector simulations on GPUs. Its initial goal is to implement the EM physics used in production runs of the main Large Hadron Collider (LHC) experiments, since those are the most computationally intensive. Key to the success of the project are proving the effectiveness of detector simulation on GPU, providing high-fidelity interoperability with Geant4, and establishing a solid collaboration with the LHC experiments.

Less than six months after its inception, Celeritas demonstrated effective GPU performance gains for a proof-of-concept physics demonstration (Johnson et al. 2021). A year after that, basic EM physics was implemented along with a tracking loop to perform simple comparisons against the Geant4 detector simulation code (Agostinelli et al. 2003; John Allison et al. 2006; J. Allison et al. 2016) for simple problems (Tognini et al. 2022). Now, Celeritas has implemented sufficient physics to replace Geant4's `G4EmStandardPhysics` by "offloading" $e^{\pm}$ and γ tracks to the GPU.

Celeritas provides high-quality facilities for seamlessly integrating into existing Geant4 applications, sending tracks to GPU and returning detector hits to the user application. The Celeritas team has been actively collaborating with the CMS and ATLAS projects to integrate into the experiment software workflows. The software has been integrated (but not yet merged upstream) as an external for CMSSW and Athena, and demonstration-level integration into CMSSW and ATLAS' FullSimLight are complete. Additional collaborations with other HEP projects (LZ and LEGEND) are under way.

The following sections will describe in detail the main aspects of Celeritas. The first few sections will present the historical context that led to its inception, the core team, the funding sources that made it possible, and its expanding engagement and impact with experiments and the scientific community. This is followed by a set of technical sections dedicated to its novel capabilities. These entail code infrastructure, geometry, field propagation, physics, input/output (I/O) capabilities, and external use—the latter being the interface developed in Celeritas to integrate it with existing Geant4 applications. Finally, the last two sections will focus on its usage and performance: this includes use-cases of Celeritas running within existing Geant4 applications, the verification that the implemented physics is correct, and present the performance results obtained by running Celeritas on a set of test-problems with increasing complexity.

1.1 BACKGROUND

The first investigation of using GPUs to accelerate Geant4 computing was a tangential part of the GeantV project (Amadio et al. 2018), which had the primary goal of using CPU single instruction, multiple data (SIMD) hardware to accelerate detector simulation. Toward the end of that experiment, a follow-up GeantX group (Canal 2019) brought Fermilab and Oak Ridge National Laboratory (ORNL) computational physicists together with computing experts from NERSC and Argonne to brainstorm pathways to exascale for detector simulation. This essentially informal collaboration was funded by Exascale Computing Project (ECP), inspired by the success of the ExaSMR code that successfully developed new algorithms for MC neutronics in nuclear reactors. The broad scope of "implementing Geant4 on GPUs" lead to many useful discussions but ultimately proved intractable as a starting point.

Celeritas was founded from those discussions as an entirely new project with the goal of *incrementally* developing GPU-targeted transport algorithms specifically for computationally intensive LHC simulations. The target of LHC production use is motivated by the high luminosity High Luminosity Large Hadron Collider (HL-LHC) upgrade, which will drive simulation requirements well beyond the projected computing

capacity that relies on traditional multicore CPU hardware (The ATLAS Collaboration 2020; CMS Offline Software and Computing 2021).

At the same time as LHC demands more compute capacity, the HPC landscape has changed so that GPUs are responsible for larger amounts of processing power due to their energy efficiency (Khan et al. 2021). Similarly, as machine learning tools become more widespread across all scientific disciplines, GPU uptake will continue to grow. In this scenario, the primary goal of Celeritas is to enable HEP simulation to take advantage of this increasing supply of GPU hardware.

At the same time, Celeritas strives for a higher simulation throughput per unit power using GPUs compared to a CPU-only machine. Because detector simulation is only a fraction of the experiment toolchain, and the initial capabilities of Celeritas will accelerate only a fraction of that, it is unreasonable to assume that the hypothetical power efficiency of Celeritas will drive any architectural purchasing decisions for new hardware for Worldwide LHC Computing Grid (WLCG). However, as more components of experiment toolchains use GPUs for acceleration for numerical simulations, reconstruction, and machine learning models, the economic considerations will likely change to favor an increasing fraction of machines with heterogeneous architectures.

1.2 INTERNAL ORGANIZATION

The Celeritas project is a multi-institute collaboration with participants from a variety of backgrounds (see Table 1) and in career stages from postdoctoral appointments to distinguished scientists. The project staff have cumulative decades of experience in developing and managing large scientific software projects in computational physics.

	Bachelor	Master	Doctorate
Physics	7	5	5
Engineering	2	3	3
Computer science	1	3	
Mathematics	1		

Table 1. Degrees earned by the eleven Celeritas core team and core advisors.

The team roster has changed from its original proposal. One founding member left the U.S. Department of Energy (DOE) lab system to work for industry, and three regular contributors have been incorporated into the core team.

1.2.1 Funding sources

Although initial exploratory work was funded by ECP, the main funding source for Celeritas is a SciDAC research grant as a cooperation between the HEP and Advanced Scientific Computing Research (ASCR) agencies within DOE. Even though the original proposal to SciDAC was to eventually implement all physics needed to simulate LHC detector interactions on GPU, the scope was narrowed to implement only EM physics, with the budget restricted to about 4 full time equivalents (FTEs) of work. Recently, DOE HEP has allocated an additional FTE of funding to develop GPU-based optical physics in Celeritas for developing new Cherenkov-based subdetector designs for LHC. Finally, a pending extension to the successful HEP-CCE program is expected to provide an additional two FTEs of funding for geometry development and further optical physics.

Contributions from the University of Warwick are underwritten by the SWIFT-HEP project, funded by the UK Research and Innovation's Science and Technology Facilities Council (Costanzo 2021). Starting in early 2024, about half an FTE per year will be dedicated to Celeritas for at least a year and a half.

Figure 1 presents a rough timeline summarizing the historical GPU-related projects from § 1.1, the Celeritas funding sources, and the LHC runs that motivate Celeritas.

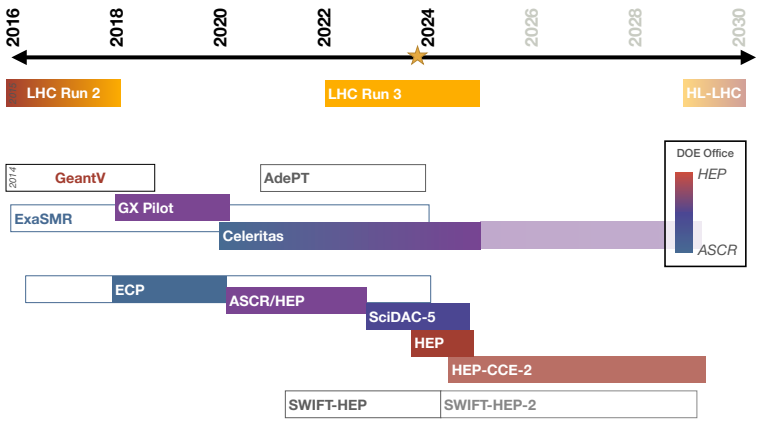
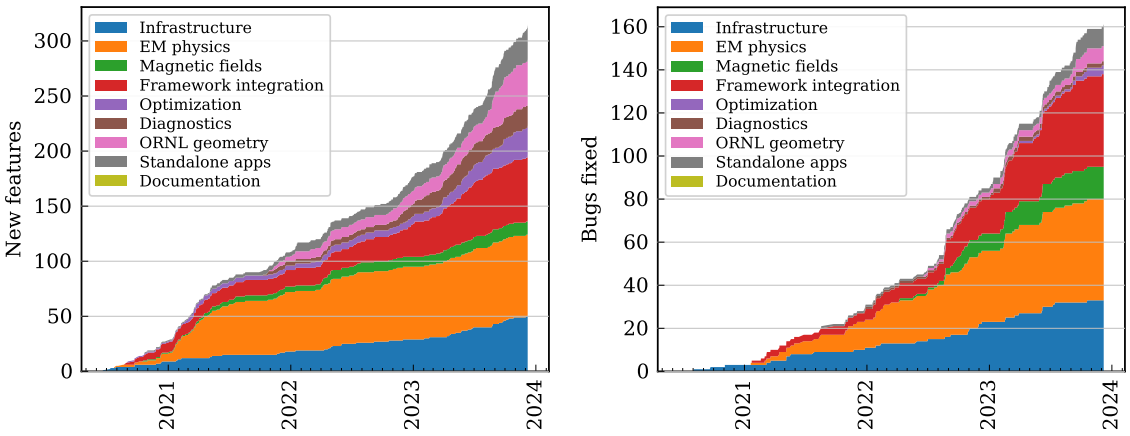


Figure 1. Timeline of LHC activity, relevant GPU projects, and Celeritas funding sources.

1.2.2 Development history

The development history of Celeritas illustrates that physics and core algorithms are only a part of the capabilities needed to succeed. Figure 2 shows the improvements and fixes to the Celeritas code base as a function of time as measured by GitHub pull request labels. The last year of Celeritas development has largely



(a) Enhancement PRs **(b) Bugfix PRs**

Figure 2. Number of pull requests for new features (a) and fixes (b) in Celeritas. Note the difference in scale between the two plots.

focused on software integration with Geant4 and GPU optimization. Much of the core physics and transport algorithms were implemented in the first year of the project. Additionally in the last year, the ExaSMR nuclear reactor simulation code has invested substantial development in the Celeritas implementation of Oak Ridge Adaptable Nested Geometry Engine (ORANGE), a GPU-based geometry model.

1.3 EXTERNAL COLLABORATION

The authors have engaged the scientific community in several ways. We have presented over fifty times at conferences, collaboration meetings, and seminars. We have formed persistent collaborations with physicists

from LHC experiments, professors at universities, and researchers at international institutions.

An inclusive, collaborative environment is necessary for the success of the project. We recognize that Celeritas will only be useful if we have a good relationship with the experiments and broader HEP community.

1.3.1 Community engagement

The Celeritas project is actively seeking community engagement for development. We have a documented and defined set of policies for community standards, code contributions, code reviews, and code releases. The popularity of Celeritas within the scientific software community has been increasing steadily as measured by the number of GitHub "stars" (Fig. 3).

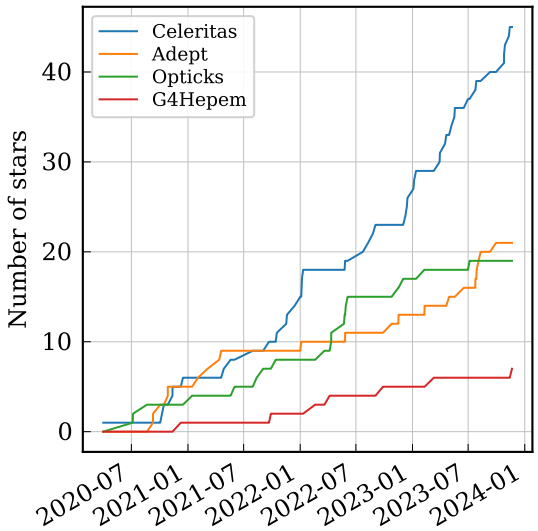


Figure 3. Github popularity of Celeritas and adjacent HEP GPU projects.

In the three years since the project's inception, three students have worked with our team, contributing code for new physics models and GPU optimizations. Our first student implemented a muon Bremsstrahlung model, another student explored possible performance optimizations for the magnetic field driver, and a third student wrote the Coulomb single scattering kernel and is working on optical photon physics. The core Celeritas team works closely with the students to ensure code quality as measured by unit tests, documentation, and validation problems.

1.3.2 Collaborations

Because of the close ties between Celeritas and Fermilab, the CMS experiment is the first integration target. We have semi-regular meetings with the CMSSW core computing and simulation teams.

ATLAS integration is equally important. At an extended hackathon at Lawrence Berkeley National Laboratory (LBNL), an ATLAS test beam problem (Lachnit, Pezzotti, and Konstantinov 2022) was the target for the very first fully featured Geant4/Celeritas integration. Since then, our team has integrated Celeritas into the ATLAS FullSimLight mini-app (Bandieramonte, Bianchi, and Boudreau 2020).

With the new funding for optical photons, we are also beginning a collaboration with the LEGEND and LZ projects. We plan to integrate Celeritas into BACCARAT for GPU-accelerated photon simulation.

We have also been collaborating with other scientific software projects. Our ties to ORNL's computing resources have enabled us to work closely with the Nvidia and AMD teams. Our team also has several Geant4

collaborators and contributors which strengthens our ties to the upstream physics code and community. We additionally contribute to the VecGeom geometry package.

Finally, of course, we engage regularly with the CERN SFT group for Geant4 research and development. In 2022 we joined the AdePT team for the HSF Detector Simulation on GPU Community Meeting. We have had monthly meetings to discuss benchmark development, capability areas, and geometry algorithms.

2. IMPLEMENTATION

The core assumption in the initial Celeritas development is that the algorithms and code structure could change drastically based on software requirements and performance testing. Flexibility in the implementation of Celeritas is key to the project's success to date and its potential extension in the future.

The fundamental design in Celeritas is *modularity* using composition-based rather than inheritance-based design. Modularity is critical to code reuse and refactoring, which is continuously needed to support new code features and enhance code performance.

2.1 INFRASTRUCTURE

Celeritas is designed from the ground up to execute almost entirely the same code on both CPU and GPU. It does this with several careful abstractions, in-memory management, and execution launching. The CPU implementation can be parallelized over tracks using OpenMP. This design allows users to integrate, test, and validate Celeritas as part of a framework even without the presence of CUDA or GPUs.

The software infrastructre is also designed to be compatible with a variety of use cases in downstream applications. When configured, it automatically detects available system components such as Geant4 and VecGeom and will build with very limited capabilities for testing. It will install with a CMake configuration for easy use by downstream CMake applications. Finally, it can be built (and even installed) inline as part of an existing CMake project as a subdirectory.

2.1.1 Data model

Problem attributes and settings (together, *parameters*) are carefully separated from track- and event-specific data (*state*) to ensure maximum compatibility with track- and event-level parallelism. Celeritas uses data-oriented design principles (Sharvit 2022) to group related data and code, as in its separation of physics and geometry into completely independent structures.

Every computational kernel is based on a function object parameterized on a thread ID, which in the default implementation maps directly to an index in a large array of per-track state data such as position, direction, particle type, and kinetic energy. As with Shift (Hamilton and Evans 2019a), the Monte Carlo GPU neutron transport code that preceded Celeritas, the track state is effectively a struct of arrays. All tracks share a single state array, as the data and kernels are independent of the particle type.

Celeritas introduces event-level parallelism using "streams" that are analogous to CUDA streams: they can be associated with tasks, CPU threads, or neither. Each stream is associated with a separate "state collection" which may comprise numerous tracks executing in parallel. Celeritas has no `threadlocal` data, giving it much more flexibility to work with client threading or tasking models. The limited global data in Celeritas (mostly for diagnostics and global attributes such as environment variables) are protected by `std::mutex` where necessary for thread safety.

2.1.2 Stepping loop

The dependencies between kernels are constructed at run time by categorizing the kernels. Figure 4 depicts the kernels, which are then converted to a topologically sorted list of calls to execute on the track states. Each kernel in Celeritas is executed on all tracks using several mask arrays: an alive/inactive flag, an along-step action flag, and a post-step action flag. The loop is terminated when no more tracks are left alive. An execution of the series of kernels is a *step iteration*.

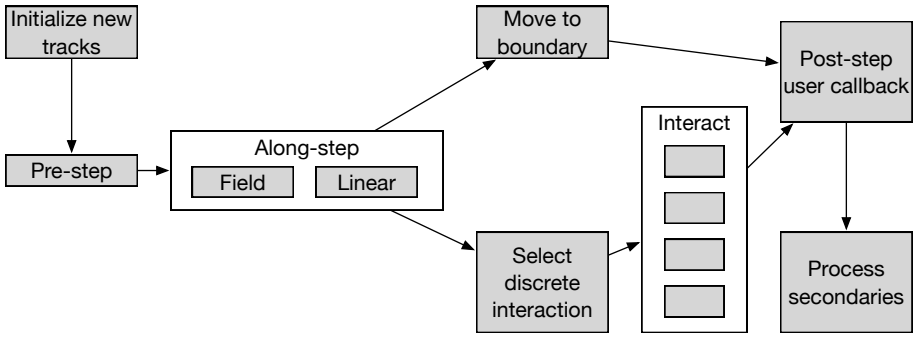


Figure 4. Action dependency graph in Celeritas. Each gray box is an action (which launches one or more kernels), and the white boxes are groups of actions that are selected on a track-by-track basis by a previous kernel.

Secondaries produced by the discrete physics interactions are allocated dynamically using an atomic counter and logic that allows a kernel to be rerun if the buffer runs out of space (see reference (Johnson et al. 2021)). When too many primaries are provided or secondaries are produced, they are added to a stack of "track initializers," a lightweight data type used to fill empty track slots once available. The primary memory limitation in Celeritas currently stems from a dynamic thread-local stack requirement in VecGeom (G. Amadio et al. 2020) due to recursive virtual function calls.

As indicated in Fig. 4, two separate along-step kernels are launched when running an application with a magnetic field: one with a field propagator, energy loss, and multiple scattering (MSC); the other with only linear propagation. Runtime options allow users to switch between different energy loss fluctuation and MSC models as well as different field representations. Because the along-step kernels and physics models are configurable and can be added from downstream applications, this effectively gives users full control over their physics implementations without rebuilding Celeritas.

2.1.3 Optimization

The first set of optimizations explored is to sort tracks based on specific properties such as the propagation kernel applied to them. An array `track_slots` maps the thread ID to its corresponding track ID. If sorting is disabled, then thread i will operate on track i. If sorting is enabled, then thread i will operate on track `track_slots[i]`. Sorting only reorder the `track_slots` array, the SoAs actually storing parameters and state are not sorted. This means that memory access patterns will not improve when sorting tracks as access it not coalesced, regardless of sorting. However, this has several other advantages. First, it allows to launch smaller kernels, and potentially reduce thread divergence. Second, sorting is much faster as data is not moved.

The second batch of optimizations explored is related to memory operations with respect to the management of the GPU. Streams allow asynchronous memory operations, which eliminate the need to synchronize the entire device every time data is moved from one memory space to the other. Implementing streams in Celeritas required updating the abstractions over memory operations and collections and defining an allocator for page-locked memory, which is necessary for asynchronous memory transfer. An abstraction for stream-ordered memory allocations was also added for managing asynchronous temporary buffers such as when sorting tracks, or when creating track initializers for secondaries. Implementation of these allocations uses a memory pool managed by the CUDA/HIP API and makes most allocation/deallocation extremely fast. Finally, Celeritas calls to the Thrust GPU algorithm library (Bell and Hoberock 2012) have been changed to not synchronize with the stream they execute on and use asynchronous API for allocation and memory transfer.

2.1.4 Random number generation and reproducibility

Celeritas by default uses the XORWOW (Marsaglia 2003) random number generator on both host and device. This bit shuffling engine has acceptable randomness properties and is used by other computational transport codes (Hamilton and Evans 2019a; Romano et al. 2015). A templated interface and configure-time switch means that any random number generator can be trivially integrated instead.

Reproducibility in Celeritas is guaranteed if a stream (e.g., a thread in Geant4 multi-thread (MT) mode) is given the same ordered primaries as input, and the number of track slots remains consistent across runs. However, because Geant4 threads can process different events on arbitrary threads, we are as of this writing implementing a feature to reset the random number state based on the event ID at the start of each event.

2.2 GEOMETRY

Celeritas defines a "geometry track view" interface to each geometry library it supports and provides a compile-time switch to choose an underlying geometry implementation for the main Celeritas stepping loop. Currently VecGeom (Apostolakis et al. 2015; G. Amadio et al. 2020) is the only production-level geometry for HEP applications. An in-house geometry, ORANGE (Johnson, Lefebvre, and Bekar 2023), is used for development of surface-based navigation and is used by the Shift (Pandya et al. 2016; Hamilton and Evans 2019b) nuclear engineering MC code for GPU neutron transport. Additional wrappers around the Geant4 navigator allow testing for comparison purposes in CPU mode.

2.2.1 VecGeom

Transporting tracks on the GPU requires the full description of the detector geometry to be available on the device. During job initialization, Celeritas performs a direct conversion from the in-memory Geant4 geometry into its corresponding VecGeom model on the CPU. Subsequently, the VecGeom model is transferred to the GPU device.

The geometry conversion process leverages the hierarchical structure of the Geant4 geometry(Agostinelli et al. 2003), as expressed in the Geant4 geometry classes, including:

- Solids, representing geometrical shapes and dimensions.
- Logical volumes, representing the containment relationships between each mother volume and its $(0 \dots N)$ daughters.
- Physical volumes, representing the translation plus rotation placement of each daughter inside its mother.

The conversion procedure for each of these classes is straightforward, while the conversion of a logical volume is implemented recursively, effectively expressing the hierarchical containment of all volumes. Starting from the top-most volume, its solid (shape), physical volume (placement) and logical volume are converted, followed by a loop to recursively convert each one of its daughters, resulting in the complete geometry being fully converted in CPU memory. Care must be taken when converting solids from Geant4 to Celeritas because of the different selection of units.

Navigation refers to the procedures responsible for following the track's position along the detector geometry, stopping the tracking across volume interfaces and accounting for different material properties across those volume boundaries.

In general, the navigation procedure tends to be computationally expensive, because it depends on geometrical calculations inside quite complex detector geometries, including potentially large numbers of daughters.

Hence it is desirable to minimize calls to navigation and geometry functions as much as possible. Celeritas uses some of the tools offered by VecGeom with this purpose in mind, as described below.

Another important tracking optimization comes from the use of a *safety distance*, which is the isotropic distance to any volume boundary. Staying within this bounding sphere guarantees a track will stay within the current volume and material. The geometry propagation kernel can reduce the number of distance-to-boundary calls by calculating and caching the safety distance. Celeritas does *not* currently take this approach because of the current high cost of the safety calculations on GPU.

2.2.2 ORANGE

Celeritas can also track particles using its native surface-based constructive solid geometry (CSG) representation, known as ORANGE. The principal advantage of the surface-based approach is the ability pre-process user-specified geometry into a form that allows for simplified tracking. The ORANGE construction process is shown in Figure 5. ORANGE has multiple front ends, each of which creates geometries in a common construction format, consisting of *shapes*, *transforms*, and *volumes*, as shown in pink in Figure 5. Shapes are a convex intersection of first- or second- order surface inequalities (e.g., planes, spheres, general quadratics) in combination with a surface *sense*. Shapes have associated transforms, which denote their 3D translations and rotations. Volumes currently consist of the intersection of any number of shapes, but the underlying tracking supports any arbitrary CSG tree with surfaces as leaves.

Shapes, transforms, and volumes are pre-processed to form the runtime geometry representation, shown in green in Figure 5. Surfaces are transformed, potentially into more general forms; then these surfaces are

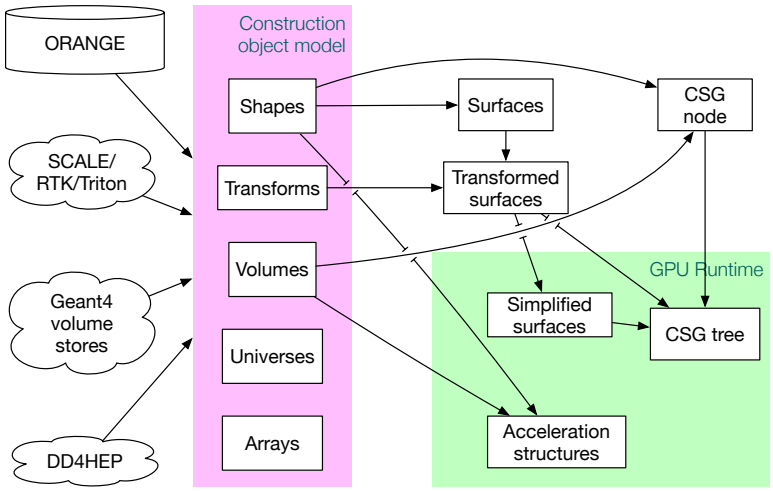


Figure 5. ORANGE construction object model.

progressively simplified. The first step in surface simplification is reducing the description of the surface to its most basic form. For example, if a general quadratic is specified with only first order terms, it is converted to a plane. Next, duplicated surfaces within a problem are eliminated, with volume definitions that rely on elided surfaces updated accordingly. Finally, a bounding interval hierarchy (BIH) (Wächter and Keller 2006) is created in order to accelerate tracking. The BIH is a tree-based structure that recursively subdivides volumes based on a partitioning plane. Point-in-volume operations can be accelerated by traversing this tree, which reduces time complexity from $O(N)$ to $O(\log(N))$.

Figure 6 illustrates the surface crossing process within an ORANGE geometry. Due to surface deduplication, a particle can cross from any cell into any neighbor cell by crossing exactly one surface. In this example, a

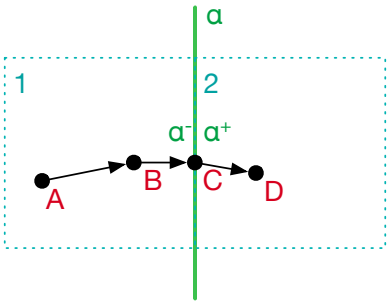


Figure 6. Surface tracking in ORANGE.

particle at location *B* first moves to point *C*, where its state is modified to indicate that is on surface *a*, with a negative surface sense. The particle then logically crosses the surface into cell 2: it remains at point *C*, but its surface sense is flipped from "inside" (negative dot product with respect to the plane's outward facing normal) to "outside" (positive). With this method, the particle is always within exactly one cell, which eliminates potential infinite loops, and complex logic caused by volume definitions with "skins" on the boundary.

2.3 EM PHYSICS

Celeritas' physics implementation follows a similar code design as used in Geant4, with virtual base classes for processes and models. Processes define the physical phenomenon itself, such as Compton scattering or photoelectric effect, while models are the mathematical description of the observed phenomenon. Therefore, a single process may have more than one model attached to it, each covering a different energy range. Since polymorphic inheritance is not performant on GPU, the concrete implementations of physics process classes and their associated model classes are *host-only*. A separate *interactor* class, analogous to the "post-step do it" method from Geant4, corresponds to each model and has the sole purpose to sample and return the final state of the interaction. The separation between model classes, which define the applicability of an interaction, from the physics interaction algorithm itself, is necessary to make the interactor a kernel to be launched on device. To allow for platform portability, thin *executor* wrappers are responsible for the kernel launches. This extra layer also means that *interactors* are pure C++, making Celeritas capable of running on both CPU and GPU.

The initial goal of Celeritas is to be capable of simulating EM showers, as they are the most intensive part of the detector simulation of LHC experiments. As such, the project's first years were dedicated to implementing standard EM physics for positrons, electrons, and photons, with the current list of models shown in Table 2.

New implementations of the physics models in Table 2 are necessary to be compatible with GPU architectures, but they strive to exactly replicate the Geant4 physics. This is achieved by: *(i)* reviewing the Physics Reference Manual (Agostinelli et al. 2023) and referenced articles, *(ii)* reviewing the Geant4 source code for undocumented modifications to the algorithms, and *(iii)* importing raw physics data directly from Geant4 when possible.

The attention in rewriting the physics algorithms and the strict use of assertions in Celeritas have helped detect bugs in Geant4.

- A discontinuity in the Urban MSC positron cross section at 10 MeV makes it possible for the final transport mean free path to be *larger* than the initial mean free path just above this threshold. As a result, the calculated effective mean free path by distance, α, can be negative, leading to an incorrect geometrical to true path length conversion for positrons with energies just above 10 MeV.

Table 2. Current status of Celeritas' EM physics. Particle symbols are defined in (Workman et al. 2022).

Particle	Process	Model(s)
γ	photon conversion	Bethe–Heitler
	Compton scattering	Klein–Nishina
	photoelectric effect	Livermore
	Rayleigh scattering	Livermore
$e^{\pm}$	ionization	Møller–Bhabha
	bremsstrahlung	Seltzer–Berger, relativistic
	pair annihilation	EPlusGG
	Coulomb scattering	eCoulombScattering
	multiple scattering	Urban, WentzelVI

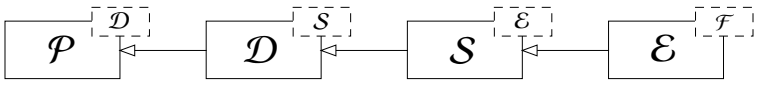
- The scaled bremsstrahlung differential cross section data used in the Seltzer-Berger model has incorrect incident electron and reduced photon energy grids for $Z > 92$. The grid sizes (14×31) and values are inconsistent with the tables (which have dimension 32×57), resulting in inaccurate simulation of electron bremsstrahlung with any of the transuranic elements.

Mathematical and physical constants are implemented from the SI specifications and CODATA values. Additional physics data is imported from Geant4 into Celeritas: *(i)* fundamental properties of particles, elements, and isotopes; *(ii)* geometry-dependent information, such as material definitions and the volumes that use them; *(iii)* simulation-specific data, which includes EM and transportation parameters; and *iv)* cross-section data for all the models used in the simulation. All these quantities are imported into a single `ImportData` struct. This can be done at startup time by initializing Geant4 with the single purpose of copying the data into memory to be used by Celeritas, or can be stored into a ROOT (Brun et al. 2020) file for faster initialization when the simulation is re-run multiple times or comparing the same version of Celeritas against different versions of Geant4 physics.

Figure 7 visualizes the interaction between the imported Geant4 data and the Celeritas physics runtime.

2.4 FIELDS

Celeritas supports the integration of the equation of motion of a charged particle in a field ($\mathcal{F}$). A set of templated classes that are linked in a hierarchical dependency provides a flexible configuration of the field propagator ($\mathcal{P}$) with different types of field drivers ($\mathcal{D}$), integration steppers ($\mathcal{S}$), and the equation of motion ($\mathcal{E}$);



$\mathcal{F}$ returns a vector of field values at a given point including a constant vector (i.e., a uniform field). The equation of motion, $\mathcal{E}$, evaluates the right hand side of the ordinary differential equations (ODEs) (the decomposition of the Lorenz equation along the particle trajectory, $s = vt$ where v is the velocity of the particle) for the component of the field ODE state, $\psi(\boldsymbol{x}, \boldsymbol{p}, t)$ where $\boldsymbol{x}$ and $\boldsymbol{p}$ are the position and momentum of the particle at a given time (t), respectively. The field state used by $\mathcal{E}$ can be extended to include the polarization ($\boldsymbol{\sigma}$ or spin) of the particle or different combinations of an electric field ($\boldsymbol{E}$) and a magnetic field ($\boldsymbol{B}$). For example, the canonical form of the equation of motion takes the form $\mathrm{d}\boldsymbol{q}_i/\mathrm{d}s = f(\boldsymbol{E}(\boldsymbol{x}, t), \boldsymbol{B}(\mathbf{x}, t))$ where $\boldsymbol{q}_i$ is each component of the field state.

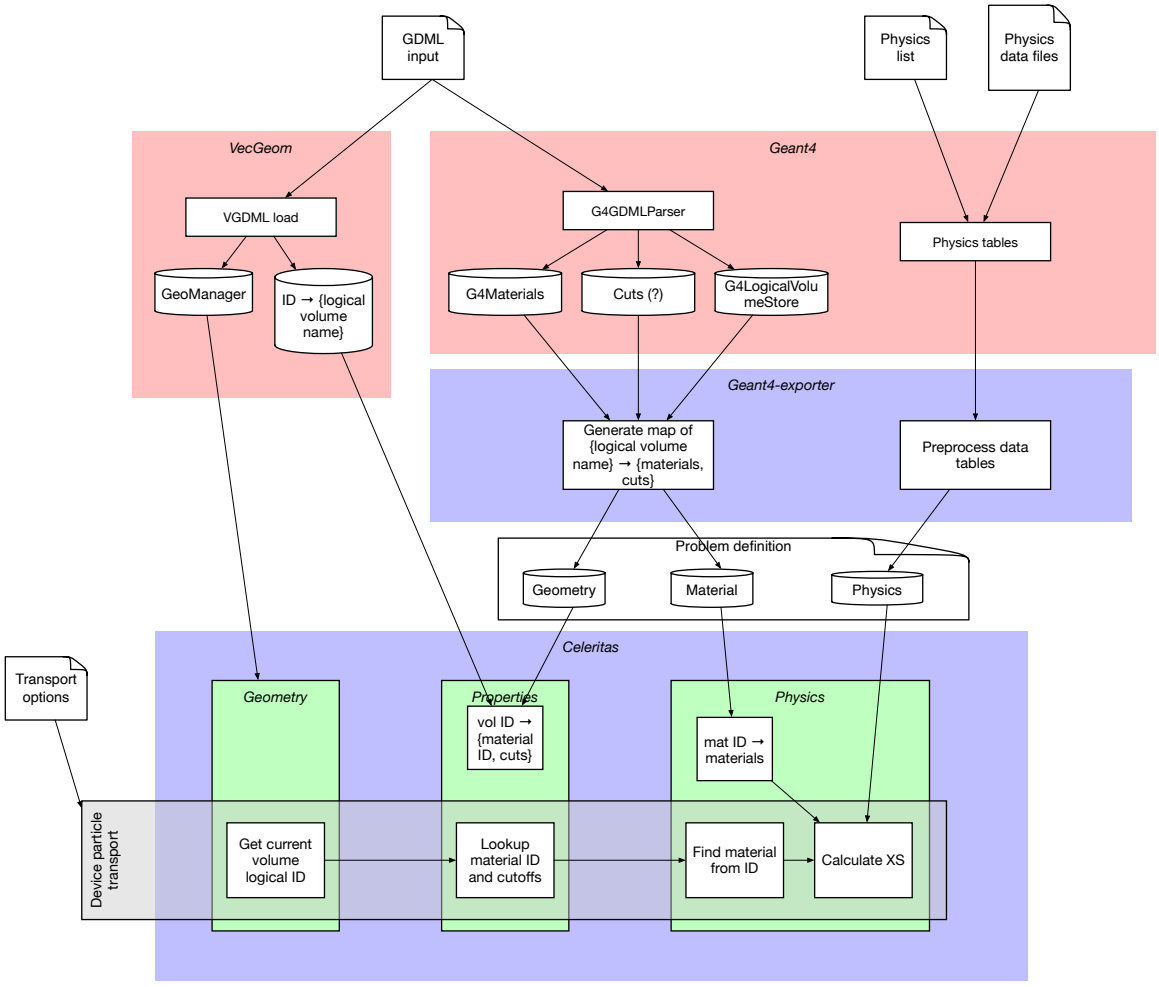


Figure 7. Geant data export workflow.

An integration stepper, $\mathcal{S}$, numerically integrates the right hand side of the the equation of motion in a non-uniform field and finds a proposed post step state, currently $\psi(\boldsymbol{x}_f, \boldsymbol{p}_f)$, and its associated truncation error for a given step size (h),

$$\psi(\boldsymbol{x}_f, \boldsymbol{p}_f) \xleftarrow{S} \psi(\boldsymbol{x}, \boldsymbol{p}; h) \, .$$

The truncation error of an integration stepper is defined as $\max(|\Delta \boldsymbol{x}|/(h \cdot \epsilon), |\Delta \boldsymbol{p}|/(p \cdot \epsilon))$ where ϵ (the maximum relative error scale) is a settable parameter, and $|\Delta \boldsymbol{x}|$ and $|\Delta \boldsymbol{p}|$ are self-estimated heuristically. Celeritas currently provides the classical 4^{th} order Runge–Kutta (Winkler 1993) and Dormand–Prince RK5(4)7M (Dormand and Prince 1980) integrators, introduced for an initial implementation since the Runge–Kutta stepper has essentially no coefficients while the Dormand–Prince stepper evaluates the less number (7 stages) of the right hand side of the equation compared to the Runge–Kutta. An analytic helix stepper for uniform z-aligned magnetic fields is also included for testing purposes. Figure 8 shows the performance of different steppers with a test setup where e^- with a velocity $\boldsymbol{v} = (0, 0.96c, 0.28c)$, where c is the speed of light, travels in an uniform magnetic field $\boldsymbol{B} = (0, 0, 1)$ T. The analytical solution for this case is a helix motion in which the radius and the pitch along z are about 3.8 cm and 6.7 cm, respectively. The Dormand–Prince

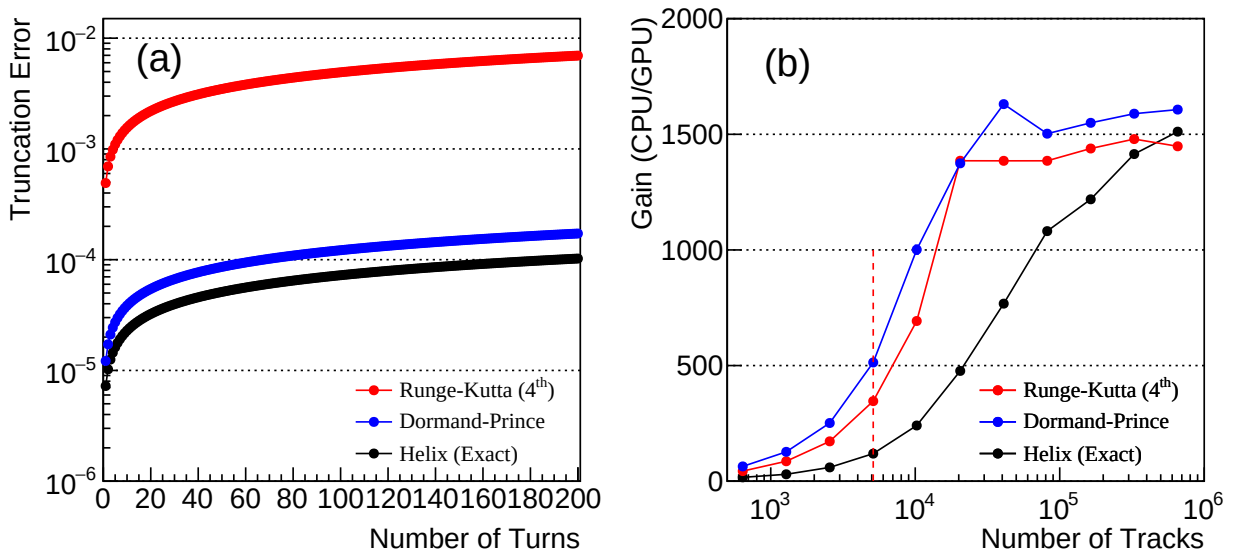


Figure 8. Performance comparison of integration steppers. Measured on the Intel® Xeon® Gold 6248 CPU (40 cores @ 2.50GHz) and the Nvidia Tesla V100 GPU (5120 CUDA cores @1.38 GHz), with e^- of $\vec{v}(0, 0.96c, 0.28c)$ in an uniform magnetic field $B(0, 0, 1T)$: (a) the accumulated truncation error of the stepper as the number of helix turns, (b) the GPU-CPU equivalence, where the dotted line is the total number of CUDA cores.

stepper performs relatively better in both the resultant accuracy and the relative gain by GPU compared to the Runge–Kutta stepper.

The field driver, $\mathcal{D}$, is responsible for finding the curved trajectory for a given step length (h) using an integration stepper within a required tolerance (i.e., a tracking accuracy). The core of the driver algorithm is an adaptive step control which advances ψ_i within a reference truncation error (ϵ_0) and estimates a good step size (h_i) for the next integration:

$$\psi(h) \xleftarrow{D} \sum_{\mathcal{S}_i} \psi_i(h_i \,|\, \epsilon_i < \epsilon_0)$$

where ϵ_i is the truncation error of each substep (h_i) which is subject to $h = \sum h_i$. For an efficient adaptive step control with a complex detector geometry, the proposed chord of which the miss-distance (the closest distance from the curved trajectory to the chord) is smaller than a reference distance is first tested and accepted if its stepping error is within a reference accuracy (δ_{chord}). Otherwise, the more accurate step integration is performed with the relative scale of the integration error. Field parameters for the adaptive step controls can be configurable during the run time by changing field driver options and can be optimized for a balance between performance and accuracy. The implementation of the current field driver is based on `G4ChordFinder` and `G4MagIntegratorDriver` with significant modifications.

The top level field propagator, $\mathcal{P}$, updates the final position and the momentum direction of the final field state (ψ_f) after integrating for the input physics step length (h) along the curved trajectory in the field or locates the intersection point on the the nearest boundary within a required accuracy (δ_{int}), if the charged particle crosses a volume surface or surfaces (i.e., the geometry limited step state, ψ_{geom}):

$$\psi_f \xleftarrow{P} \min(\psi_{\text{stepper}}, \psi_{\text{geom}}).$$

Details of the propagator algorithm are described in the next subsection.

Besides the uniform magnetic field, a couple of user-defined non-uniform magnetic fields are implemented and tested: *(i)* a volume-based (r–z map) magnetic field and *(ii)* a parameterized magnetic field extracted from CMSSW. The r–z map field consists of the azimuthal symmetric 2-dimensional r–z grids of magnetic field values for the entire CMS detector region, which can be converted to the vector of (B_x, B_y, B_z) at any given point using a linear interpolation. The implementation of the parameterized magnetic field is based on the TOSCA computation version 1103l of CMSSW (Klyukhin et al. 2008) which evaluates the r and z component of the CMS magnetic field (B_r, B_z) with polynomial functions inside the CMS tracker region ($r < 1.15$ m and $|z| < 2.80$ m). The timing performances of the propagation in different user-defined fields, which find the final field state $\psi(\boldsymbol{x}, \boldsymbol{p})$ at the end of each step or at an intersection point on a boundary if a track crosses a volume consisting of a set of layers (x–z planes), are shown in Figure 9. The relative gain by GPU over a single core CPU for all tested cases is $200 - 300$ times for processing the number of tracks around 10^4–10^6.

2.4.1 Field propagator

The field propagation algorithm in Celeritas differs from the Geant4 algorithm. Because load balancing and divergence are critically important for GPU kernels, the propagator algorithm is designed so that each geometry function (distance finding, direction changing, movement) is called in exactly one place in the algorithm. The algorithm itself (Fig. 10) guarantees convergence by always reducing the substep trial length or breaking out of the loop.

Although the surface-based ORANGE geometry implementation will always cross a boundary successfully, the current volume-based VecGeom implementation can get "stuck" on a boundary, reporting a zero distance-to-boundary regardless of the direction or requested step length. When this happens, the particle is "bumped" a small distance along its original direction of travel.

One notable difference from Geant4 is that if the requested propagation length (i.e., the physical step length) is slightly greater than the calculated boundary distance, the track is moved *without* setting its boundary flag to true. This avoids an artificial bias to undergoing an interaction directly on a boundary. The requirement is also necessary for ORANGE, which (currently) cannot safely initialize new tracks on a boundary. Figure 11 clarifies this behavior near the boundaries.

In Fig. 11a, the intersection point along the chord is past the boundary and within the intersection tolerance. Therefore the substep is accepted as a boundary crossing and the substep loop is terminated. Because the

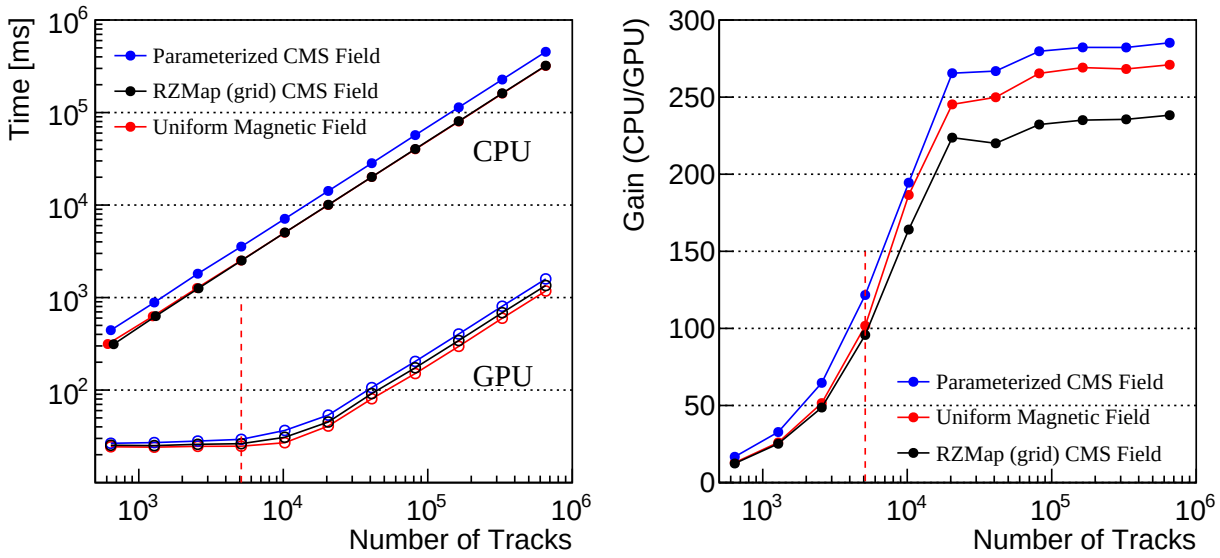


Figure 9. CMSSW magnetic field performance comparison. The performance of the propagation in different magnetic fields with a setup where e^- crosses 8 x–z planes per revolution around the origin and travels the total distance equivalent to 10 revolutions by taking 100 steps per revolution in 1) the parameterized CMS magnetic field in the CMS tracker region, 2) the volume-based 2-dimensional r–z map extracted from CMSSW, and 3) an uniform field $B(0, 0, 3.8T)$: (left) the elapsed time on the Intel® Xeon® Gold 6248 CPU (40 cores @2.50 GHz) and the Nvidia Tesla V100 GPU (5120 CUDA cores @1.38 GHz) and (right) the relative performance in time (CPU/GPU) as the number of tracks.

substep is *beyond* the boundary, the substep length is reduced to reflect that the entire requested path is not traveled. This prevents the geometry and physics steps from a false coincidence.

The second case, Fig. 11b, shows a substep that ends near a boundary but still inside the current volume. The boundary is only detected by searching slightly beyond the chord length, using a "delta search" length. In this case, the substep distance is accepted, the loop is terminated, and no boundary is crossed.

In the absence of a field, or when used for neutral particles, the implementation of the algorithm, boundary search, and boundary crossing reduces exactly to the linear propagation algorithm. This useful property allows the charged and neutral particles to share a single propagation kernel. The first implementation of magnetic fields in Celeritas did this for convenience, but as an optimization two separate kernels are now used.

2.5 OUTPUT

Simulating detector output is another critical component in the stepping loop. Managing output in Celeritas and any other GPU-based simulation tool is complicated by the separate execution spaces and the track-level parallelism: multiple tracks are interacting with multiple detectors at the same time, and these hits must be copied back to the CPU.

Celeritas defines a "step collector" helper class that can selectively extract different track and step properties at the beginning and end of a step. It filters the tracks based on requirements and track attributes and copies the attributes back to the host in a C++ container amenable as a user extension point.

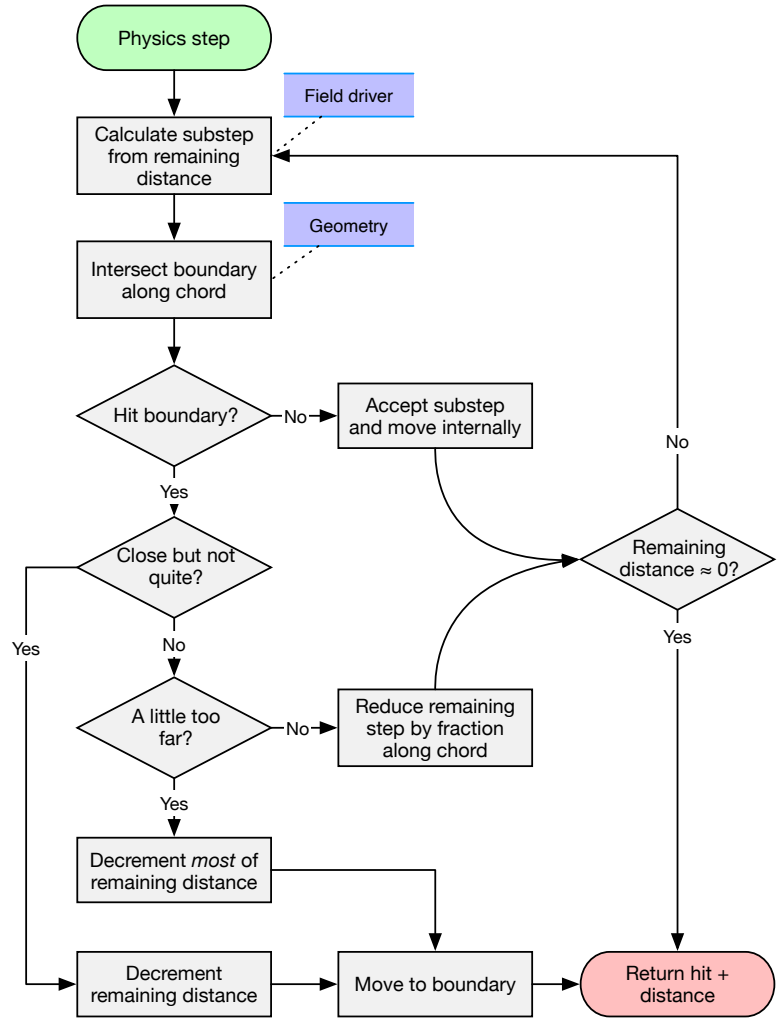
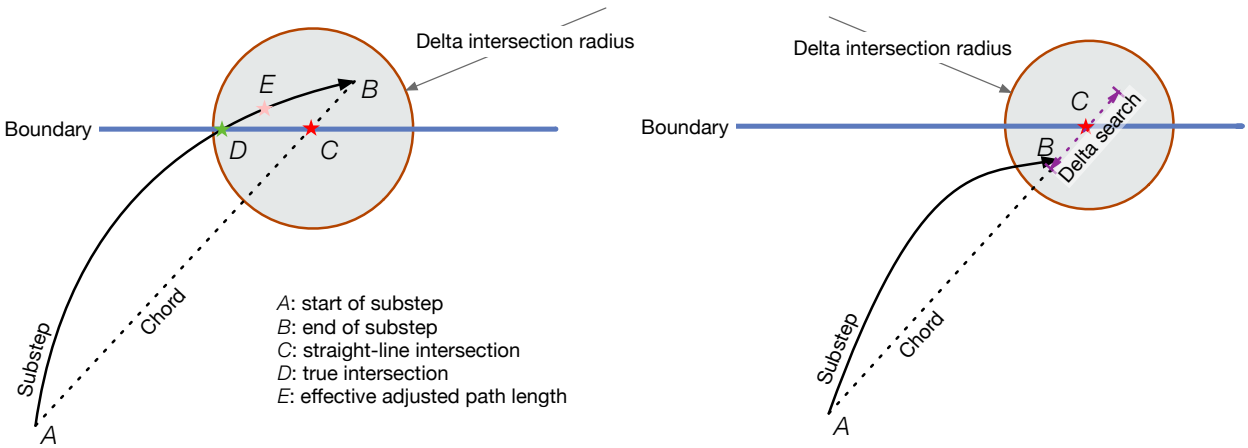


Figure 10. Field propagation algorithm.



(a) Converged and crossed boundary

(b) Converged without boundary crossing

Figure 11. Field propagation algorithm near boundary crossings.

2.5.1 MC truth

Simulation output data is currently stored at the step level. Since Celeritas is most efficient when stepping millions of distinct tracks, possibly from multiple events, in parallel, there is currently no mechanism to detect the end of a track's lifespan or the end of a given event as Geant4 does with its `G4VUserEventAction` and `G4VUserTrackingAction` virtual base classes. Therefore, the only interface that allows the extraction of all of the history of the simulation is via a single `StepInterface` virtual class, which is a callback class that gathers every step of every thread (i.e., track) and allows the user to process the collected steps on either host or device.

The current interface for a MC truth output is the `RootStepWriter` concrete implementation of the `StepInterface`. This implementation is host-only and writes a stream of steps to a ROOT output file. Since the I/O happens once every step, and for every step it flushes the latest step of every thread, the produced file is an out-of-order stream of simulation steps. Thus, a common event-based data output requires the writing of a post-processing tool by the end-user.

2.5.2 Diagnostics

The same abstractions used for the stepping loop kernels and the step collection in Celeritas can be leveraged to write custom actions for collecting arbitrary data at each step through callbacks. These "diagnostic" classes are useful for processing step data directly on the GPU and for quick, general verification with minimal overhead and no post-processing. Celeritas provides a few example user diagnostics, which accumulate certain quantities at each step and output the result as plain text using the Celeritas output interface.

`SimpleCalo` is an event-integrated scoring system which inherits from the `StepInterface` and accumulates energy deposition in sensitive regions on the GPU by directly obtaining and processing the gathered step data used for sensitive detector (SD) hits and MC truth. The `StepDiagnostic` and `ActionDiagnostic` are concrete implementations of the `ExplicitActionInterface` which accumulate data for each particle type by interacting directly with the track state data: the former collects the distribution of steps per track, and the latter tallies the post-step actions. Corresponding diagnostics in the `accel` library collect the equivalent data from a Geant4 simulation, facilitating a fast, direct comparison between Geant4 and Celeritas for a few integrated quantities.

2.5.3 Geant4 hits

The most complex integration in Celeritas is interacting with user detectors. Ideally, all detector code would be on the GPU to reduce bandwidth and improve utilization. In practice, it is impossible to write a complex GPU detector for an application whose Geant4 implementation is constantly changing. Therefore, Celeritas has a high-fidelity detector interface that at every step iteration reconstructs Geant4 detector hits and sends them directly back to user detectors.

When the Celeritas parameters are initialized, the `HitManager` class loops over all Geant4 volumes to find which have SDs attached and maps them to corresponding VecGeom volumes. Because the Geant4 `G4VSensitiveDetector` themselves are thread-local, and initialization happens on the "main" thread, further initialization is delayed to the first time a local stream is executed. This complication is necessary not only because of the thread-local Geant4 detector objects but also because Geant4 object allocators require that construction and destruction happen on the same thread. Even using a class can allocate internal components, so great care is taken to ensure that a Celeritas stream hews closely to the Geant4 objects it accesses and creates.

Every step that takes place in a sensitive detector volume (and optionally only those with nonzero energy deposition) is sent to the CPU during the step iteration. This list of hits, all for independent threads, is then

looped over again. The quantities requested by the user are used to reconstruct a `G4Track` object. The most complex item in this reconstruction is the `G4TouchableHistory`, which encapsulates the full hierarchical logical geometry state. To do this, Celeritas uses the spatial position to initialize a local navigator, moving slightly to a nearby boundary forward or backward along the direction of flight until the VecGeom logical volume matches the Geant4 logical volume. This reconstruction process has the interesting side effect of illuminating bugs in the geometry navigation code (from a point in space corresponding to a mismatched volume) or errors in the user detector description (from an ill-defined point corresponding to multiple volumes).

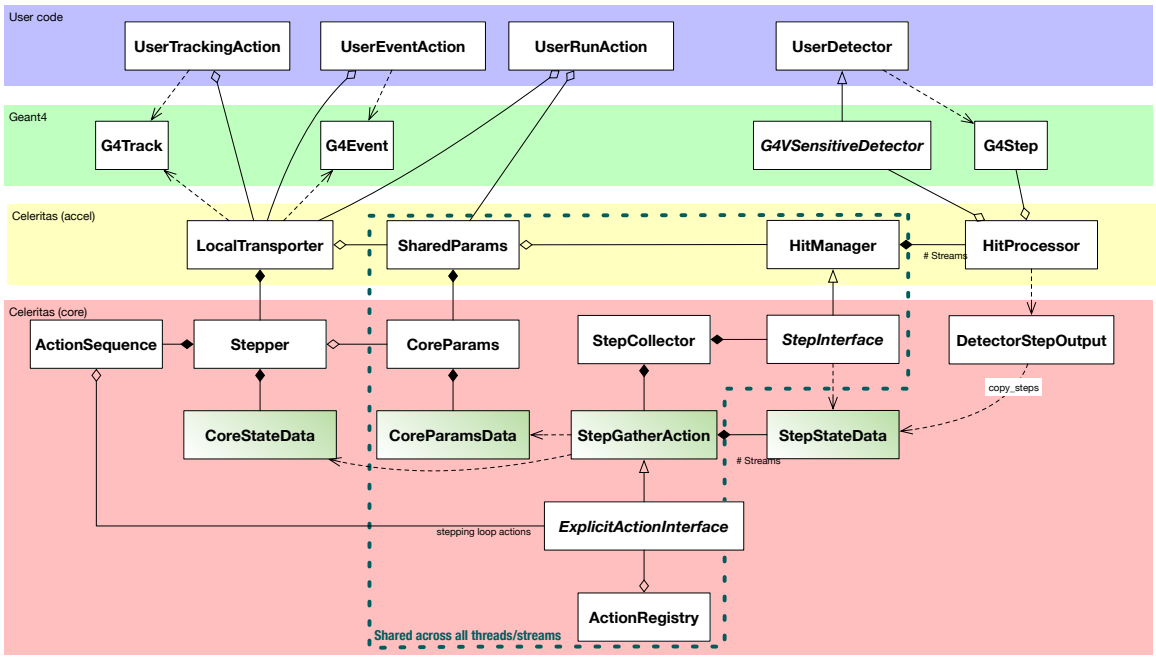


Figure 12. Geant4/Celeritas integration UML diagram.

2.6 EXTERNAL USE

To integrate effectively into experimental frameworks and user code, Celeritas provides straightforward APIs utilizing Geant4's main user hooks to minimize the amount of development and configuration work downstream. The developers have invested substantial effort to automate and streamline the Geant4–Celeritas integration by building several key interfaces between the two codes in the *accel* library, as shown in Figure 12.

The primary classes of *accel* that are responsible for communication between Geant4 and Celeritas are `SetupOptions`, `SharedParams`, and `LocalTransporter`. `SetupOptions` defines control options for Celeritas, such as number of tracks to run in parallel, device heap/stack memory, and what step data to collect for forwarding to any SD that are hit. During problem setup, `SharedParams` queries the Geant4 geometry and physics processes, models, materials, and particles data using `SetupOptions` and Celeritas' API to set up the problem internals accordingly. Geant4 logical volumes with SDs are also mapped to VecGeom volume IDs. These problem parameters are copied to GPU once and shared among all CPU threads and GPU tracks. `LocalTransporter` provides the interface to offload `G4Track`s from Geant4 to Celeritas, constructing a `Stepper` from the problem's `SetupOptions` and `SharedParams`. To integrate correctly with Geant4's multithreading system, each worker thread requires a thread-local instance of `LocalTransporter`.

During execution, `G4Track`s passed to `LocalTransporter` have their key attributes appended to a buffer for later "resurrection" in the Celeritas tracking loop. At the end of the event or when the buffer is full, these

buffered tracks are run through the main Celeritas tracking loop. At each step iteration, a Celeritas kernel queries whether the tracks are in any SD volumes. If so, data marked by `SetupOptions` (position, time, energy deposition, touchable, etc.) for those tracks is copied to the CPU. A loop over the tracks reconstructs Geant4 `G4Step` objects, which are passed directly into the `G4VSensitiveDetector::Hit` method of the associated SD.

A final class, `SimpleOffload`, is provided to orchestrate the above classes with Geant4's Initialization, Run and Event User Actions. Intended to be constructed as a thread-local instance, it provides helper functions to be called in user's concrete actions so that the Celeritas problem data and transporter are correctly synchronized with Geant4.

Within the requirement of each worker thread having its own instance of `LocalTransporter`, any Geant4 hook that allows modification of `G4Track`s may call `LocalTransporter::Push` to offload tracks to Celeritas. It is the caller's responsibility to ensure the `G4Track` is suitable for offload and to kill it on the Geant4 side. Two *accel* interfaces are currently provided that can be used to offload via Geant4's `G4UserTrackingAction::PreUserTrackingAction` or `G4VFastSimulationModel` hooks. In the first case, `SimpleOffload` provides a `PreUserTrackingAction` method that the user may call in that method of their concrete `G4UserTrackingAction` to immediately offload all EM particles to Celeritas. In the second case, `FastSimulationOffload` implements a concrete `G4VFastSimulationModel` that can be used to offload EM particles entering a specific detector region. Basic use of both methods and overall *accel* integration are shown in the `simple-offload.cc` and `fastsim-offload.cc` examples under `example/accel`.

Future work on Celeritas–Geant4 integration will address potential performance and integration limitations of the SD and offload hooks. For now, the developers expect most users to use the default "step onload" SD integration because of its simplicity, reasonable efficiency, and ability to directly compare Celeritas/Geant4 results via scoring outputs. However, an extension point in Celeritas allows users to replace the step/track reconstruction code with arbitrary user-based SDs for improved performance. For offload of tracks, an additional interface using the per-particle `G4VTrackingManager` introduced in Geant4 11.0 will be provided. Full support for regions will also be added in Celeritas so that tracks exiting the region attached to `FastSimulationOffload` may be sent back to the Geant4 tracking loop. This "onload" would also allow non-EM particles created by, e.g. lepto-nuclear processes, to be handled.

3. TEST PROBLEMS

A variety of test problems have been used by the Celeritas team to benchmark the code's performance and evaluate its accuracy.

3.1 SIMPLIFIED DETECTORS

The performance of Celeritas is regularly measured using a set of benchmarks (Tognini et al. 2022) of increasing complexity that test different aspects of the code. The current preliminary set of benchmarks shows the performance cost of adding different features, such as multiple materials, multiple geometric regions, magnetic fields, and additional physics capabilities. All benchmarks start with 1300 primaries per event, all 10 GeV electrons. This number is chosen to simulate the amount of energy per LHC collision deposited into the material. With the exception of TestEM3, which emits monodirectionally into the left side of a series of detector slabs, all "test beams" emit isotropically at the origin.

Because the primary target for Celeritas is data center–class hardware for production runs, rather than consumer hardware for development, the two primary machines used to evaluate performance are the DOE supercomputers Summit (Facility 2018) and Perlmutter (Center 2022). These machines both use Nvidia GPUs, which are necessary for comparing realistic problems due to the CUDA-specific implementation of VecGeom. Some runs using ORANGE show performance results using the Frontier supercomputer (Atchley et al. 2023), which uses AMD GPUs that VecGeom does not support. These supercomputers and key characteristics are shown in 3.

Machine/Arch	Card	Thermal Design Power (TDP) (W)	Cores or SMs	Cards per node
Summit/CPU	IBM Power9	190	22†	2
Summit/GPU	Nvidia V100	250	80	6
Perlmutter/CPU	AMD EPYC 7763	280	64	1
Perlmutter/GPU	Nvidia A100	250	108	4
Frontier/CPU	AMD EPYC 7453	225	64‡	1
Frontier/GPU	AMD MI250x	500	220	4*

Table 3. Hardware characteristics of the three supercomputers measured. Physical core count (without multithreading) is shown for the CPU case and "streaming multiprocessors" for the GPU case.

On each supercomputer, each benchmark problem is run on a single compute node with one process per discrete GPU. These independent processes run simultaneously with different starting seeds for the pseudorandom number generator in order to give a better estimate of the variance over a range of potential events. One set of benchmarks is run with each instance using one CPU and one discrete GPU, and a second set executes an OpenMP-multithreaded run of C/G CPUs per instance, where C is the number of CPUs per node and G is the number of discrete GPUs. Comparing the two gives an effective measure of the GPU-to-CPU core equivalence for each benchmark.

3.2 ATLAS TILE CALORIMETER

The Celeritas team integrated EM offloading into a standalone ATLAS tile calorimeter (Henriques 2015) test beam application developed for previous Geant4 validation (Lachnit, Pezzotti, and Konstantinov 2022),

†. By default, 2 processes are reserved for the system, so each CPU problem uses 7 cores per task.
‡. By default, 8 processes are reserved for the system, so each CPU problem uses 7 cores per task.
∗. The AMD MI250x card contains two discrete GPUs, so benchmark problems use 8 processes per Frontier job.

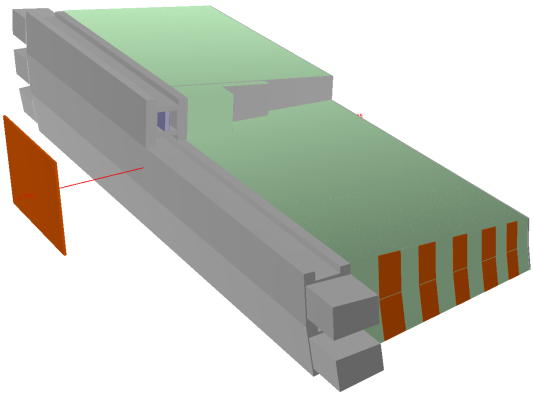


Figure 13. ATLAS tile calorimeter modules.

which models two of the 64 hadronic calorimeter modules installed around ATLAS. This integration has now been adapted into the plugin-based architecture of the FullSimLight (Bandieramonte, Bianchi, and Boudreau 2020) ATLAS framework development tool. It provides a practical and straightforward test case for Celeritas accelerating Geant4 using a nontrivial geometry.

The test problem uses a monodirectional beam of 18 GeV π^+ particles with a total of 2048 primaries per execution. One set of runs uses a single primary per event as per the original test problem, and a second set uses 64 primaries per event to model the workload of a full-scale ATLAS simulation. All hadronic processes are modeled in Geant4 using the `FTFP_BERT` physics list; the standard EM models and processes are executed in Celeritas using the offloading procedure described in § 2.6. Each simulation is run using a quarter of a node on Perlmutter: a single Nvidia A100 GPU and 32 hardware threads. The threads are pinned to one NUMA node (16 physical cores) and use the closest GPU.

3.3 CMS

As one of the two flagship CMS experiments, the CMS detector (Fig. 14) is important to model and accelerate using Celeritas. Production simulation of CMS is performed with the CMS software framework, CMSSW, which has a complex pipeline of event generation, simulation, digitization, and reconstruction. The detector geometry in CMSSW is built in memory using configurable options so that any particular version of the framework can build and simulate the detector at any stage of its evolution, some of which may have different sub-detectors. As an example, the Run 4 (high luminosity upgrade) will have a new high granularity calorimeter HGCAL (CMS Collaboration 2021), which is projected to be expensive to simulate.

In addition to different geometry configurations, CMSSW has multiple simulation use cases that can have different physics options. The physics configurations are tuned to have the highest performance without negatively impacting the simulation quality. The granularity of these options (e.g., region-specific magnetic field options and tracking cuts) are outside the scope of the initial Celeritas implementation, making a direct comparison between CMSSW and Celeritas impossible. As an alternative for exploratory purposes, the core physics configuration and geometry are exported from CMSSW and loaded into a standalone Celeritas-based Geant4 application, `celer-g4`. This application allows a realistic simulation of CMS with full physics to be compared with and without Celeritas acceleration.

Production simulation of $t\bar{t}$ events is expensive and representative of the work that Celeritas needs to accelerate. A set of events from 14 TeV pp collisions resulting in $t\bar{t}$ were generated using Pythia and used as inputs to the Celeritas Geant4 application, where Geant4 simulates the hadrons and offloads EM tracks to Celeritas. A second alternative to generating input is to run CMSSW with a special Celeritas-based option to dump all offloaded EM tracks. These tracks can then be run in a special EM-only test application.

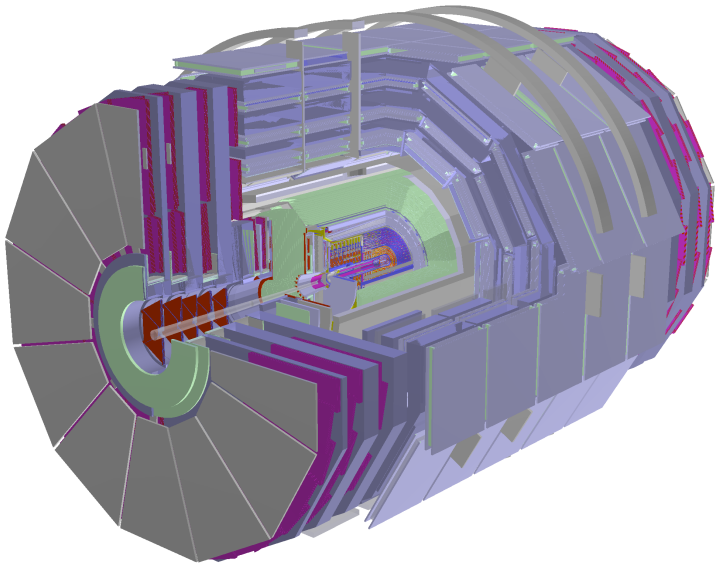


Figure 14. CMS detector in its Run 3 (2018) configuration.

4. RESULTS

The overarching goal of this project is to reduce the computational burden and energy cost of MC simulations in HEP experiment workflows. Two Figures of Merit (FOMs) are defined as key indicators of the value of GPU acceleration.

The first figure of merit is the ratio of the simulation throughput when Celeritas runs on GPUs versus the available Central Processing Units (CPUs) of a given machine. Since GPUs will provide the bulk of the compute capability on high-end HPC hardware in the future, this FOM represents the ability of Celeritas to effectively use next-generation hardware. It must be sufficiently high to justify running on DOE Leadership Computing Facility (LCF) resources, for example.

A second performance metric is critical to the viability of Celeritas in the broader HEP community: the work done for the same amount of cost in power consumption and hardware, using Geant4 on CPU as a baseline. This FOM is the motivation for HEP workflows to adapt to using Celeritas for MC transport. The Geant4 team supposes that a factor of two speedup resulting from "adiabatic improvements" to the code is not outside the realm of possibility Marc Verderi and Alberto Ribon 2021, so a factor of 2× improvement of Celeritas per over Geant4 runtime is our second target. Assuming that electricity consumption and waste heat disposal are the primary constraints for independent HEP computing centers, this factor should be evaluated by comparing the performance of Geant4 on CPUs with a Celeritas on GPUs.

4.1 PHYSICS VERIFICATION

Celeritas physics must be equivalent to Geant4 physics for performance comparisons to be justifiable. Additional verification and validation is necessary to give users confidence in the Celeritas EM implementations.

4.1.1 Per-model verification

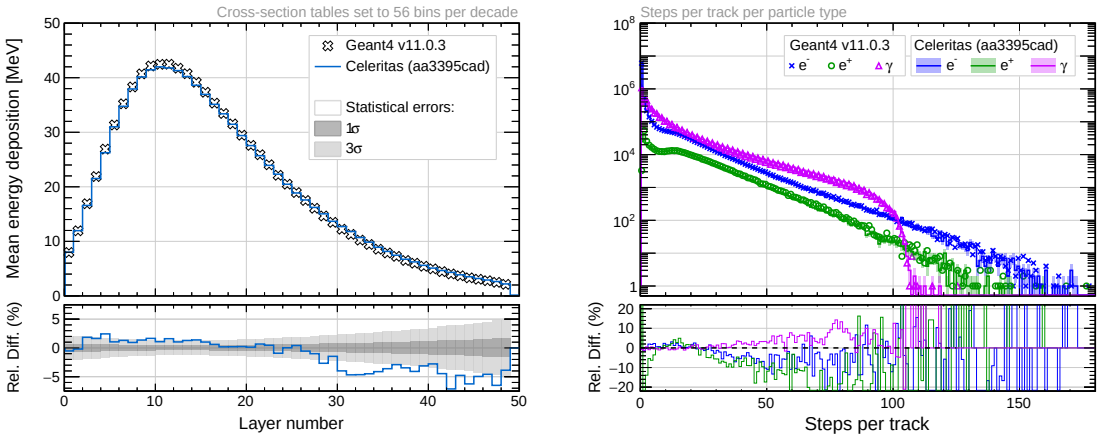
The per-model verification currently uses two separate standalone applications: one for Celeritas and one for Geant4. Each application allows for run-time selection of geometry, primaries, and physics list. The standalone Celeritas application uses the `RootStepWriter` I/O interface described in §2.5, while the Geant4 application has its own ROOT I/O. A small post-processing tool gathers Celeritas' steps to create tracks and events, so that both Celeritas and Geant4 have identical event-based output files.

A final ROOT macro generates comparison plots between both MC codes. The plots validate step information (such as step length, step time, steps per track, steps per process, and so on), track information (total track length, vertex energy, direction and position), and energy deposition in sensitive detectors. The comparison uses mainly two geometries: one with no spatial complexity whatsoever, a Pb cube with side of 500 meters, and the TestEM3 problem described in 3. The verification process found several errors in the Celeritas physics implementation.

4.1.2 Full physics verification

The same plots used in the per-model verification are used with all physics processes enabled, showing that the stepping loop is performing as expected and is in statistical agreement with Geant4. This is confirmed by Fig. 15, which presents the total energy deposited and the number of steps per track for the TestEM3 geometry using a test beam of 10^4 1 GeV e^- emitted from $(-22, 0, 0)$ cm along the $+x$ axis.

As the Geant4 offloading library becomes more consolidated and with more comprehensive I/O, the goal is to move away from two independent applications, such that a single executable will provide both Geant4-only and Celeritas-only outputs by leveraging Celeritas' Geant4-offloading capabilities.

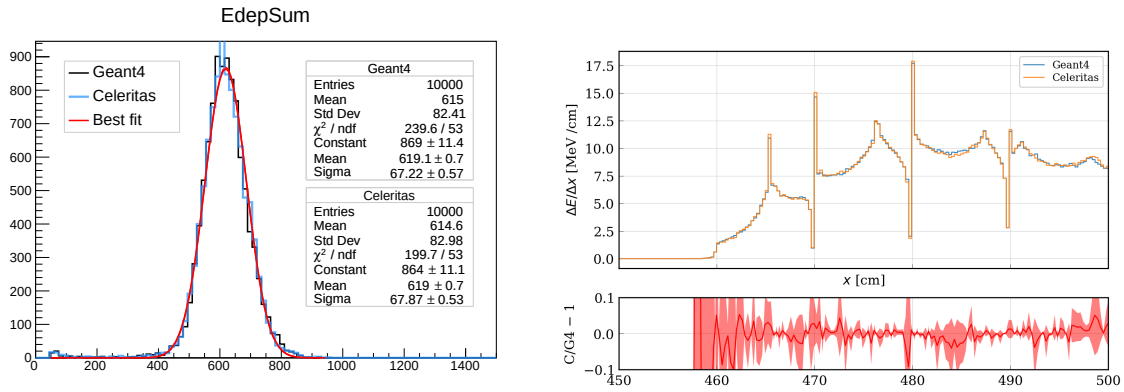


(a) Energy deposition **(b) Step distribution**

Figure 15. Comparisons between Celeritas and Geant4 for the TestEM3 geometry. Energy deposition is shown per layer (gap + absorber pair), and the step distribution is given per track per particle type.

4.1.3 ATLAS tile calorimeter

The results from the sensitive detector output (Fig. 16, generated from a different run of the same problem setup using 10 000 primaries) demonstrate excellent agreement between Celeritas and Geant4 physics and verify that the hit reconstruction is working correctly.



(a) Energy deposition per event **(b) Binned energy deposition**

Figure 16. Comparison of Celeritas and Geant4 energy deposition in the ATLAS tile calorimeter problem. The plots show (a) a distribution over events and (b) binned as a function of z.

4.2 INTEGRATION

Integration fidelity and ease of use are two important critera for Celeritas. A "high fidelity" integration will be able to adapt all options from Geant4 to perfectly reproduce the same physics. Ease of use can be measured in part by the lines of code needed to integrate Celeritas.

4.2.1 FullSimLight and Athena

Given the complexity of the full Athena (ATLAS Collaboration 2021) framework and the use of custom solids incompatible with VecGeom/GPU in the EMEC geometry model, integration testing has begun with

the simpler FullSimLight standalone application. An initial test problem in the form of the ATLAS tile calorimeter test beam setup has been built as a FullSimLight plugin as described in Section 3. With a new EMEC geometry model using Geant4 native solids nearing completion, extension to cover the full ATLAS geometry is expected soon. A preliminary integration of Celeritas into the ATLASExternals toolchain with the ATLAS-specific Geant4 and VecGeom dependencies is in progress as the first step in testing Celeritas within the full Athena framework.

4.2.2 CMSSW

Integration with CMSSW is challenging. The toolchain complexity of the full CMSSW workflow is greater than a standalone application due to the number of dependencies and the custom build system. Additionally, CMSSW's use of Geant4 is nonstandard: instead of a run manager, it has a custom manager used to execute the multithreaded Geant4 inside of a task-based TBB environment. This caused some unusual conditions that are prohibited in a standard Geant4 run, such as inconsistent sensitive detectors created between the master and worker threads. Finally, the use of `G4Track::GetUserInformation` by CMSSW interferes with the track reconstruction process used by Celeritas.

CMSSW processes events collected from LHC collisions or simulated with the CMS detector response using Geant4. CMSSW supports MT workflows and uses the Intel Thread Building Block (TBB) library for their threading model in which the Oscar MT producer interfaces with an external Geant4 multithreaded application for the CMS detector simulation. To demonstrate the full capability of offloading EM particle transport on GPUs, Celeritas provides a library of common interfaces to experimental frameworks along with the Celeritas core libraries. As the Celeritas transport engine is to be independent from external thread pool models or work schedulers, it is straightforward to integrate Celeritas into CMSSW through the Oscar MT Producer and the Geant4 user actions.

The first step of the integration is configuring Celeritas as one of selected tools of CMSSW, which includes building VecGeom with enabling the CUDA capability (backend) and Geant4 with VecGeom/CUDA and DD4HEP under the CMSSW environment (cmsenv). A new configuration file, celeritas.xml, is added to setup Celeritas as an external package. The second part is to integrate the Celeritas Geant4 interfaces into the CMS Geant4 application (SimG4Core/Application) for offloading specialized tasks to device. The current offloading workflow of Celeritas integrated in CMSSW follows the default Geant4 event-basis multithreaded approach in which the Celeritas transporter is thread-local, but all core parameters or external data used by the Celeritas transporter are created on the master thread and shared among workers. The hybrid approach of transporting EM particles on GPU while simulating the rest of particles on CPU cores is pipelined with the Geant4 user tracking action (`G4UserTrackingAction`) and the user event action (`G4UserEventAction`) after the transporter is activated in each worker thread (i.e., inside `RunManagerMTWorker`).

Shared data needed for Celeritas (interaction cross sections, material tables, geometry defintion, etc.) are dynamically created during the initialization stage of the CMS Geant4 run manager (`RunManagerMT`). The thread local transporter is initialized on each worker thread (`RunManagerMTWorker`) with shared data as well as thread-specific data—all data are commonly available for both host and device so that Celeritas supports CPU-only workflows as well. Currently, all transporters share one GPU device even though different combinations of CPU threads and the number of GPU devices are possible for the more sophisticated hybrid workflow. Alternative workflows, including a Geant4 fast simulation interface or a specialized tracking manager, will be explored as future work.

A temporary workflow is demonstrated with a test branch of CMSSW available in the Celeritas fork of the CMSSW repository *Celeritas (CMSSW fork).* For an initial integration and demonstration, offloading all EM particles from processing 100 $t\bar{t}$ events with the CMS Run 3 as well as the HL-LHC detector configuration with different magnetic fields (zero, uniform and RZ-map field) are tested. Each kernel of the Celeritas

transporter is launched when there are enough number of tracks to be processed concurrently from a temporary buffer owned by the transporter. As the kernel is executed with a pre-defined chuck of collected tracks, the remaining tracks are flushed at the end of each event action. To support user-specific sensitive detectors, output hits from the transporter (i.e., from GPUs) are converted to G4Step data and passed to the hit process method of Geant4 so that the user hit I/O is decoupled from the device code. As mentioned earlier, Celeritas prepares shared geometry data on the master thread of the Oscar MT producer where SD volumes are not readily available yet in the tested version of CMSSW 13.0.6, a list of SD volumes were pre-built and are attached manually to logical volumes for a given detector configuration during the geometry conversion for GPU.

4.2.3 Standalone Celeritas Geant4 app

Figure 17 shows the relative speedup gain (CPU/GPU) as the number of CPU threads measured on one of consumer GPU cards (Nvidia RTX3090) for processing 32 $t\bar{t}$ events with the CMS Run 3 detector configuration and different types of magnetic field setups, (a) without I/O, (b) with I/O.

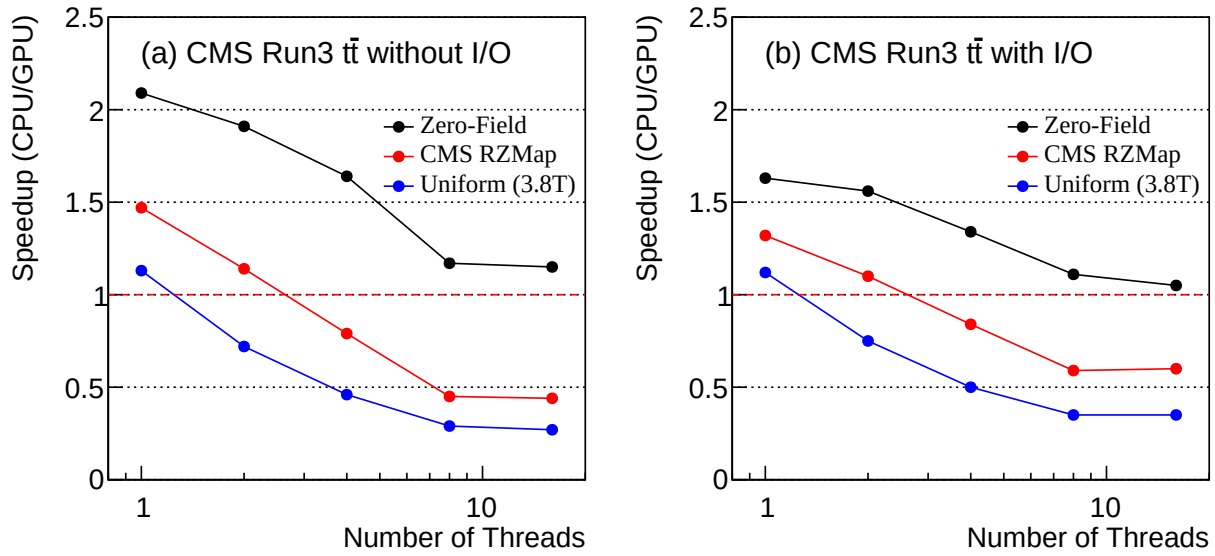


Figure 17. Relative speedup (CPU/GPU) as the number of CPU threads for processing 32 $t\bar{t}$ events with the CMS Run 3 detector configuration and different types of magnetic field setups. Measured on the Intel® Core i9-10900K CPU (20 Cores @ 3.70GHz, Total Mem: 65.7 GB) and the Nvidia GeForce RTX3090 (10496 CUDA Cores @1.70 GHz, Total global Mem: 24.3 GB): (a) without I/O, (b) with I/O.

The relative speedup of offloading EM track to Celeritas on an Nvidia A100 GPU with the CMS HL-LHC detector configuration and the three different field setups is shown in Figure 18. The overhead of reconstructing the SD hits is relatively small, accounting for $\sim$ 15% of the runtime in the zero field case and $\sim$ 5 – 10% with a magnetic field. The large variance in the time per event and small total number of events simulated results in load imbalance and poor scaling for both Geant4 and Celeritas, though the parallel efficiency is worse with offloading.

Verification with CMS is still very preliminary at this stage. Figure 19 bins the number of hits as a function of radial and axial position, hit energy, and hit time in the Run 3 configuration of CMS from CMSSW 13.0.6 using the standalone Celeritas Geant4 test application.

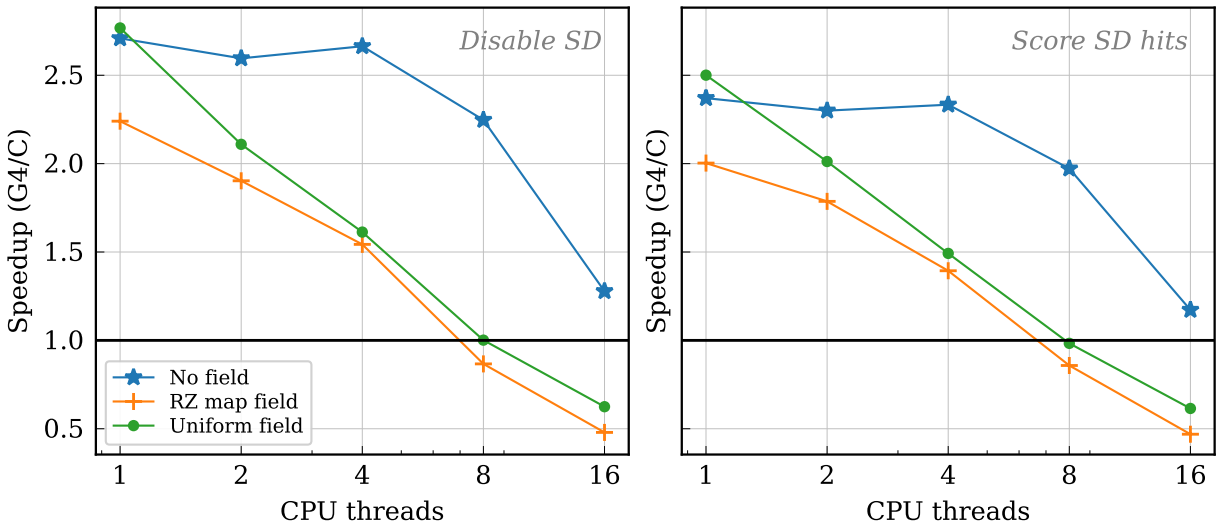


Figure 18. Relative speedup of offloading EM tracks to the GPU through Celeritas as a function of CPU threads. 32 $t\bar{t}$ events were simulated using the CMS HLLHC geometry with different magnetic field and SD hit collection configurations. Performance was measured on an AMD EPYC 7532 32-core processor @ 2.4 Ghz and a 40 GB Nvidia A100.

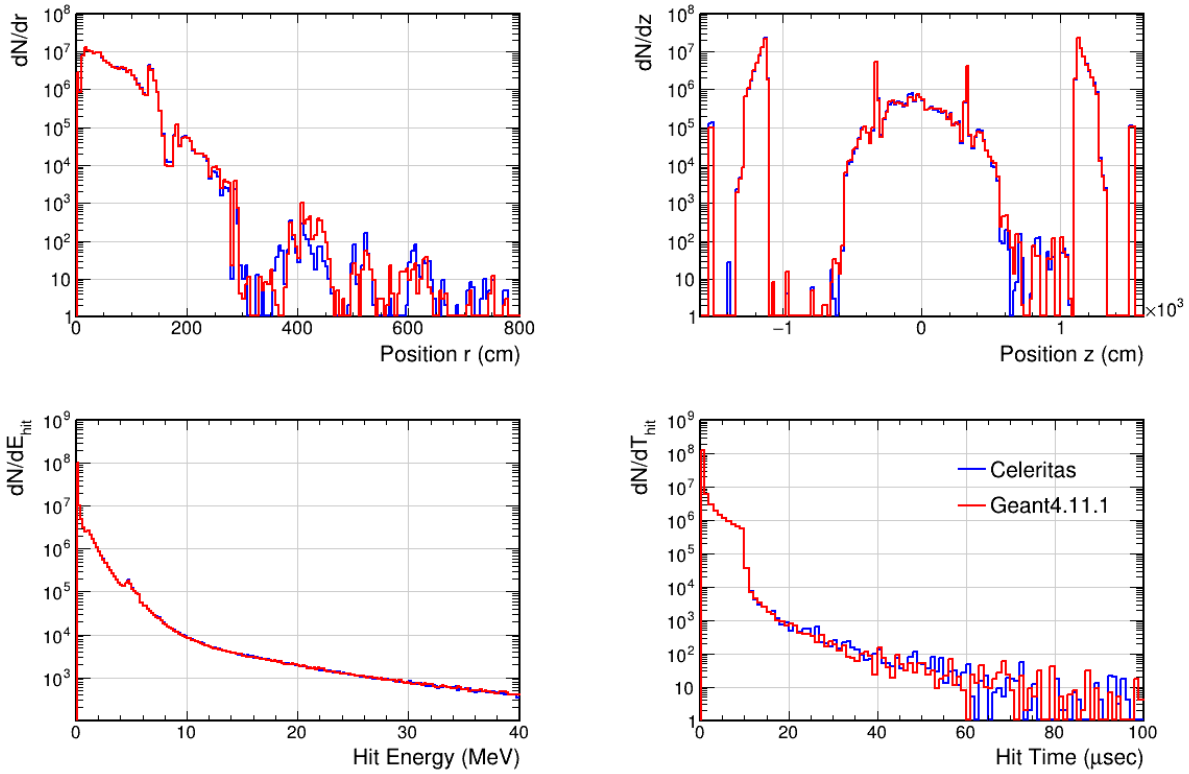


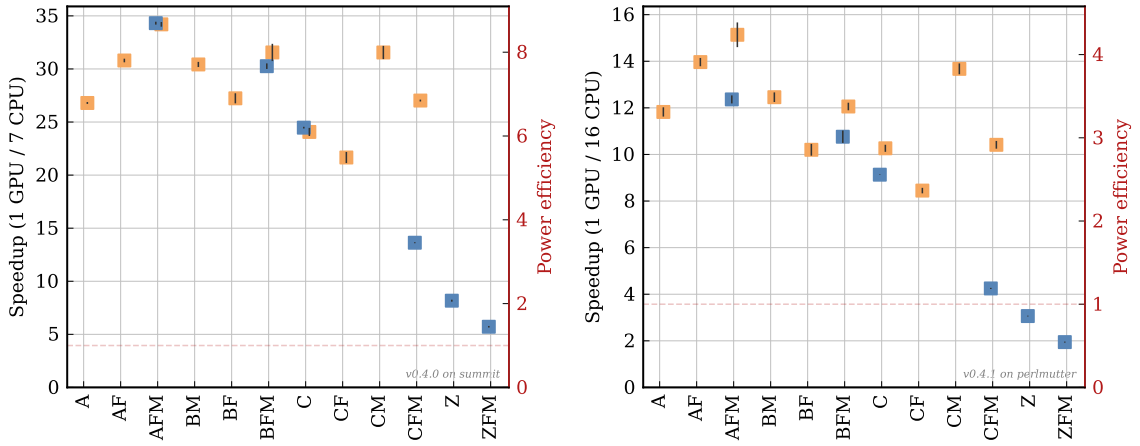
Figure 19. Preliminary comparison of hit count rates in sensitive detectors with Geant4 with and without Celeritas acceleration.

4.3 PERFORMANCE

The values for power efficiency in this section are *estimates* based on vendor-reported average Thermal Design Power (TDP) shown in Table 3. Because the CPU-only and GPU-only executions were alternated during benchmark execution, it is unlikely but not impossible that either case was throttled due to overall node power consumption. Preliminary indications of self-reported device power indicate that while running Celeritas, the Nvidia devices consume far less than the power envelope, possibly because the application is unable to achieve full utilization of the device and its tensor cores. If that holds true, the power usage reported here is a *conservative* estimate and real efficiency numbers may be higher.

4.3.1 Simplified calorimeters

Figure 20 shows (a) the raw throughput measured as events per second of wall time for a single instance and (b) the relative speedup by using GPUs available in the machine versus neglecting them and using only CPUs. The GPU-to-CPU equivalence for Summit is the speedup multiplied by the number of CPUs used in each run.



(a) Summit efficiency **(b) Throughput**

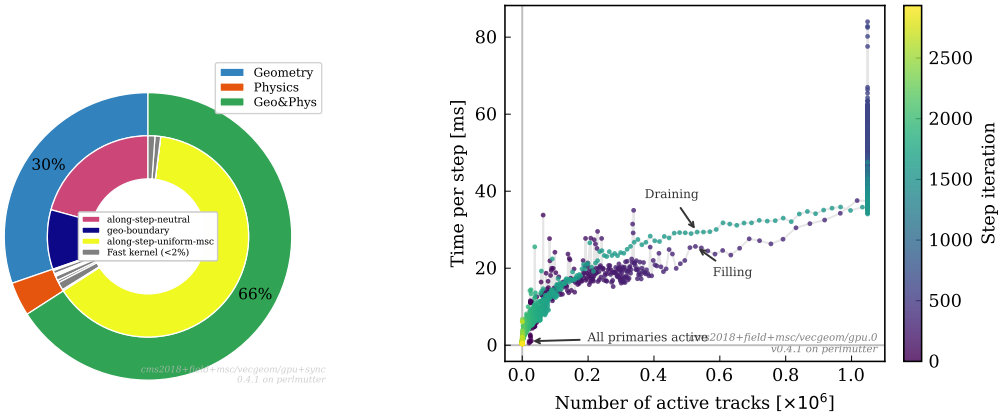
Figure 20. Absolute and relative performance of the regression tests. The problems are (A) infinite-medium, (B) simple-CMS, (C) TestEM3, and (Z) CMS-2018; the modifiers are (F) using a 1 tesla uniform magnetic field, and (M) using the Urban multiple scattering model. Instances with ORANGE navigation are orange; VecGeom are blue. Error bars are measured with 6 executions with different starting seeds.

On Summit's hardware, this means a single V100 can give approximately the same throughput as 40 to 240 cores of an IBM Power9. On Perlmutter, an A100 provides the same throughput as 32 to 240 cores of an AMD EPYC.

The more computationally difficult problems have slower throughput as expected, but the greater geometric complexity disproportionately affects the performance of the GPU simulation, especially using VecGeom, as shown by the TestEM3 case with multiple scattering and magnetic field (the CFM case in Fig. 20). In that problem, using Celeritas' in-house, surface-based ORANGE navigator results in more than a factor of two speedup on the GPU versus the VecGeom volume-based implementation, even though the two implementations have approximate performance parity on the CPU.

The impact of the geometry complexity in the most difficult problem, the Run 2 configuration of CMS with multiple scattering and field propagation (ZFM in the figure), is especially clear when examining performance timers. As show in Fig. 21a, 95% of the compute time is spent in the geometry-heavy along-step propagation

and boundary crossing kernels. This is fantastic news for future optimization since only a small part of the tracking loop needs to be targeted. Figure 21b shows the cost per step iteration as a function of active tracks.



(a) Action compute time (GPU) **(b) Time per step**

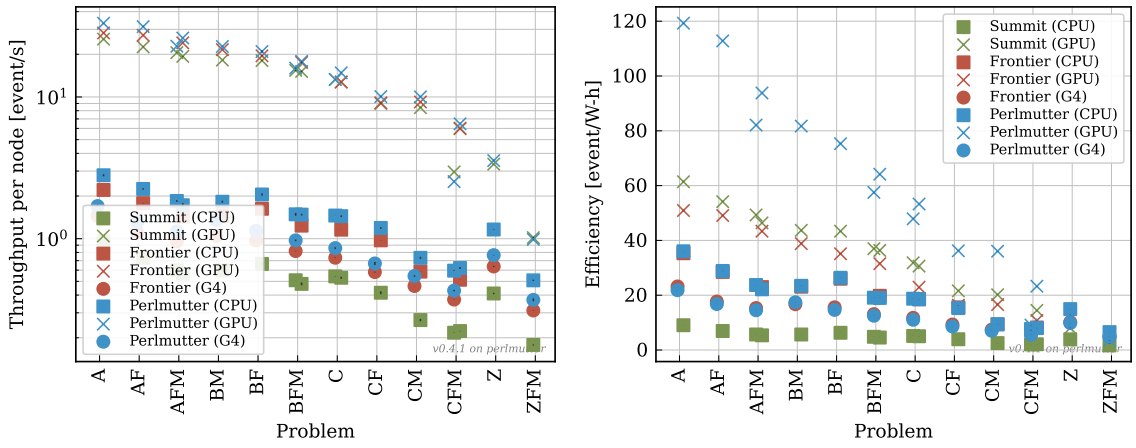
Figure 21. Timing results for CMS-2018 (with field and multiple scattering) on Perlmutter GPU. Fig. 21a shows on the inner wheel the integrated time spent in each "action," roughly corresponding to a GPU kernel, with the kernels partitioned into the outer wheel components based on whether they interact with the geometry navigation, the physics, or both. Fig. 21b shows the distribution of wall time per step iteration (with one to ~ 1M particles per iteration).

The outliers at the beginning are likely due to low-energy particles stuck looping in the beamline, a good target for future optimization. The performance per step also shows that the overhead of running the loop with very few active particles is small, which is important to overall performance since individual tracks will always have a broad distribution of steps.

The two key metrics motivating GPU development are the performance *capacity* (Fig. 22a), taking advantage of hardware present on existing system; and energy *efficiency* (estimated in Fig. 22b using vendor-reported thermal power limits), motivating the use of GPUs in purchasing decisions. The decreasing throughput with increasing problem complexity is because each even requires more work (steps and computations per step). Regardless of the problem, though, the trend of relative performance is qualitatively similar. The CPU performance per watt and capacity per node are consistently far lower than GPU. The current generation of AMD GPU hardware (MI250x on Frontier) has about the same efficiency as the last generation of Nvidia GPUs (V100 on Summit), in part because of the lower cache and higher latency of AMD. The Nvidia A100s powering Perlmutter are by far the leader in performance per watt, and the AMD CPUs nearly have efficiency parity with their GPUs for the more difficult simulations.

The energy efficiency plot above does *not* account for the additional performance improvement of Celeritas on CPU compared to pure Geant4. Figure 23 displays the relative performance improvement of Celeritas to pure Geant4. Celeritas on CPU through the standalone application runs anywhere from 50% faster to twice as fast compared to Geant4. More tests need to be run to evaluate the performance discrepancy between using the standalone application versus the Geant4 driver.

Another potential metric of performance for comparing GPUs to CPUs, suggested by the AdePT team, is the throughput per "core," where core is a CPU hardware core or a GPU SM (streaming multiprocessor) (which is in fact a grouping of many single instruction, multiple thread (SIMT) CUDA cores). Figure 24 plots this metric for the EM performance problems, but it shows no clear correlation across GPU devices nor between GPU and CPU and is therefore not a useful metric for comparison. This is hardly surprising since Monte



(a) Capacity **(b) Efficiency**

Figure 22. Events simulated per compute node (a) and per unit energy (b) using for a variety of DOE supercomputers on CPU and GPU architectures (see the extended document for descriptions) using the EM-only test problems with Celeritas and Geant4. Problem complexity increases to the right; see Fig. 20 for a description of the problems.

Carlo simulations are known to be limited by memory bandwidth rather than floating point throughput. The bandwidth of GPU cards changes between one generation and the next, and even between cards in the same generation. Even the number of CUDA cores per SM changes between cards.

4.3.2 ATLAS tile calorimeter

Figure 25 illustrates the strong scaling of Celeritas and Geant4 when only a single GPU is shared among the CPUs. Due to the small amount of work per event, Fig. 25a shows that the overhead of coupling Celeritas to Geant4 exceeds the potential GPU-based performance boost, especially as the hardware resources hinder the number of parallel kernel launches. Fascinatingly, using Celeritas on CPU consistently improves the *overall* application performance by about 20% (i.e., a 1.2× speedup), despite the overhead of translating data types between Celeritas and Geant4.

In contrast, Fig. 25b shows the utility of the GPU for more computationally intensive problems. Offloading EM tracks to GPU from a single CPU accelerates the *overall* time by a factor of 2.3×, including the startup overhead, the overhead of transferring EM "primaries" to Celeritas, transferring hits back to CPU, and reconstructing Geant4 hit objects. Even with 32 hyperthreads sharing a single GPU, Celeritas has a speedup of 1.7×. The maximum speedup possible, calculated by instantaneously killing all tracks that could be offloaded, is about 2.5× for a single-threaded Geant4 run and about 2.2× with 16 CPU threads. This implies that Celeritas on GPU is effectively almost instantaneous.

The results from the tile calorimeter scaling suggest Celeritas should run effectively in an environment with several CPU Geant4 threads sharing a single GPU used by Celeritas.

4.3.3 CMSSW

CMS uses many optimized parameters for Geant4 physics models and the magnetic field propagation in different sub-detector regions as well as a shower library for the hadronic forward (HF) calorimeter simulation. Currently, Celeritas tracks EM particles in all detector regions with a set of default parameters provided by Geant4 including HF. Therefore, the direct comparison between the default CMS detector simulation and

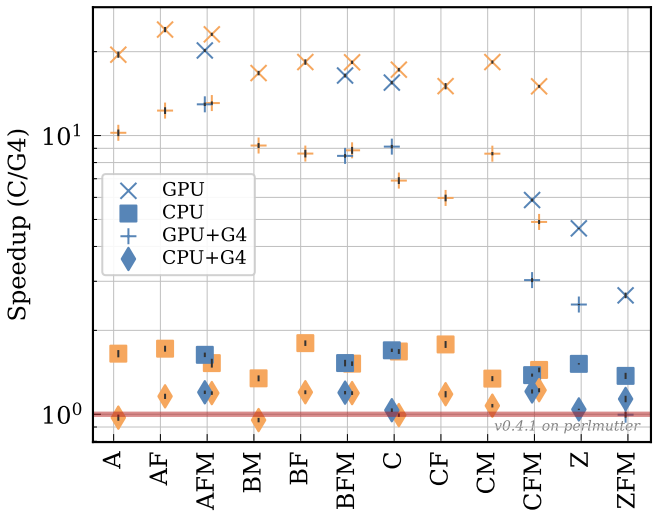


Figure 23. Throughput improvement of Celeritas compared to Geant4 for the EM-only test problems. Anything above the red line is faster using Celeritas. An orange marker represents a run with ORANGE navigation, and blue markers represent VecGeom navigation. The "+G4" cases are when running through the Geant4 application and offloading to Celeritas; the other cases are through a standalone Celeritas-managed application.

Celeritas is not possible at this point. Nevertheless, the relative speedups between CMSSW (version 13_0_6) and CMSSW with offloading EM particle transport by Celeritas are shown in Figure 26(a) for the CMS Run 3 and HL-LHC detector configurations. The projected speedup is taking account the relative number of hits between the default CMS simulation and CMSSW with EM offloading using Celeritas assuming that the simulation time is roughly proportional to the number of sensitive hits shown in Figure 26(b). The projected time, t_p, for CMSSW with the Celeritas offloading is estimated as follows,

$$t_p = t_o + w \cdot (t - t_o)$$

where t_o is the time taken by the simulation without any EM particle and w is the relative fraction of the number sensitive hits between CMSSW and Celeritas. The primary difference in sensitive hit distribution is seen in the HF detector region (around 11 m $< |z| <$ 14 m) where CMS uses a shower library. The weight factors, w, for Run 3 and HL-LHC are about 0.16 and 0.19, respectively. The extension for introducing more configurable parameters in Celeritas EM physics processes and the field propagation codes in different detector regions to compare performance directly to CMSSW is being underway.

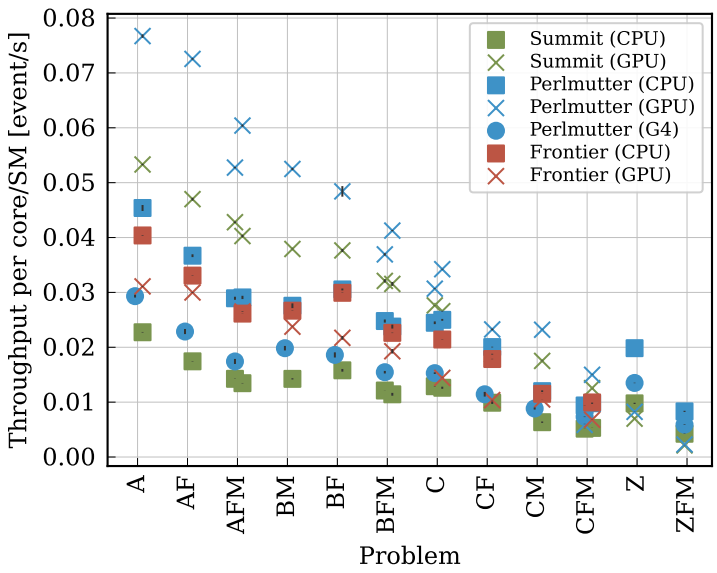
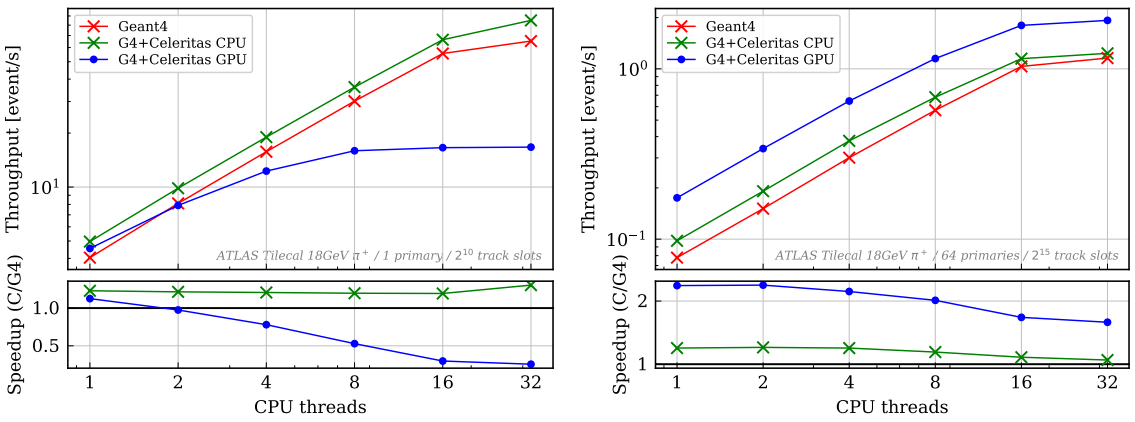


Figure 24. Throughput as a measure per of events per core per second for the EM ORANGE problems.



(a) 1 primary per event **(b) 64 primaries per event**

Figure 25. Comparison of Celeritas and Geant4 scaling with the number of CPU threads. The test execution uses only one primary per event in Fig. 25a, whereas Fig. 25b has 64 primaries per event.

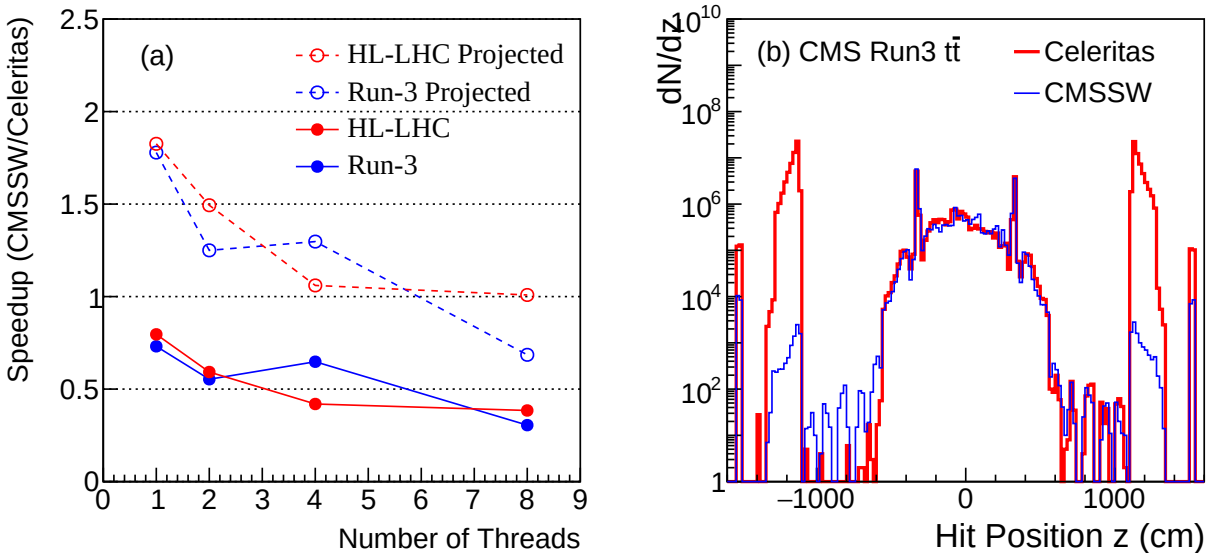


Figure 26. Comparison of CMSSW and Celeritas. Figure (a) shows the relative speedup measured on the Intel® Xeon® Gold 6248 CPU (40 cores @2.50GHz) and the Nvidia Tesla V100 GPU for processing 100 $t\bar{t}$events with the CMS Run 3 and HL-LHC detector configurations. Figure (b) shows the distribution of the z–position of sensitive hits from simulation with 32 $t\bar{t}$events and the CMS Run 3 detector configuration.

5. CONCLUSIONS

Designed from the ground up for performance on GPU, Celeritas has demonstrated the viability of simulating HEP detectors on next-generation of high performance computing architecture. It now has the ability to directly integrate into Geant4 applications and transport EM tracks on both GPU and CPU. It supports a variety of software configurations, Geant4 versions, threading modes, and user integration paradigms. Integrating into a Geant4 application takes about 100 lines of mostly boilerplate code, since Celeritas can automatically converts user geometry in memory and imports Geant4 physics data and options. It reconstructs SD hits to return to the calling application in lieu of requiring the code implementing the detectors' response to be rewritten with GPU-compatible data structures.

Celeritas has been integrated into EM-only test applications, into ATLAS FullSimLight, and into CMSSW. The EM test programs demonstrate on the Perlmutter supercomputer that a single Nvidia A100 GPU runs Celeritas at the same rate as rate of 42–290 AMD EPYC CPU cores depending on detector setup, corresponding to a power efficiency gain of up to a factor of 4. This performance gain does *not* include the $\sim 50\%$ relative performance gain compared to Geant4 of using the Celeritas physics and tracking loop.

Geometry is currently the most limiting factor in overall performance, but since the ORANGE navigation shows 2.5× performance improvement compared to VecGeom on the most challenging problem ORANGE can run, it is expected that GPU performance for realistic detector problems will grow substantially with the advent of surface-based navigation.

Integration into Geant4 applications has demonstrated substantial performance gains by utilizing available GPU hardware. A beam of 18 GeV π^+ on an ATLAS tile calorimeter shows that Celertias can effectively partner with Geant4 in physics, improving overall performance by 75% with 16 EPYC CPU threads sharing an A100 GPU. Because of the geometric complexity of CMS and the current performance limitations of VecGeom on GPU, full simulation of the Run 4 configuration shows modest performance gains: at 8 EPYC threads to one A100 GPU, Celeritas has 2.25× throughput when run without a magnetic field but only breaks even when run with a uniform 4T field. Performance options that have been implemented and tested, but not yet evaluated for integrated problems, are likely to further improve performance of Celeritas full-physics problems.

The fact that Celeritas shows a performance boost of 20% even without a GPU motivates further testing in a wide variety of applications and a greater investment toward integrating into Geant4 and the LHC experiment frameworks. The potential gains in power efficiency, up to a factor of 5 for EM performance, should motivate serious consideration for purchasing data center–grade Nvidia GPUs to supplement traditional CPUs in new WLCG hardware. Even though these positive results are preliminary, and some derive from simplified problems, they herald a bright future for GPU simulation in HEP.

ACKNOWLEDGMENTS

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research and Office of High Energy Physics, Scientific Discovery through Advanced Computing (SciDAC) program. Work for this paper was supported by Oak Ridge National Laboratory (ORNL), which is managed and operated by UT–Battelle, LLC, for the U.S. Department of Energy (DOE) under Contract No. DE-AC05-00OR22725 and by Fermi National Accelerator Laboratory, managed and operated by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy. This research was supported by the Exascale Computing Project (ECP), project number 17-SC-20-SC. The ECP is a collaborative effort of two DOE organizations, the Office of Science and the National Nuclear Security Administration, that are responsible for the planning and preparation of a capable exascale ecosystem—including software, applications, hardware, advanced system engineering, and early testbed platforms—to support the nation's exascale computing imperative. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231 using NERSC award HEP-ERCAP-0023868.

REFERENCES

Agostinelli, S., et al. 2023. *Geant4 Physics Reference Manual.* https://geant4-userdoc.web.cern.ch/UsersGuides/PhysicsReferenceManual/html/index.html.

Agostinelli, S., J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Arce, M. Asai, et al. 2003. "Geant4—a simulation toolkit" [in en]. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 506, no. 3 (July): 250–303. ISSN: 01689002, accessed March 23, 2018.

Allison, J., K. Amako, J. Apostolakis, P. Arce, M. Asai, T. Aso, E. Bagli, et al. 2016. "Recent developments in Geant4." *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 835:186–225. ISSN: 0168-9002. https://doi.org/https://doi.org/10.1016/j.nima.2016.06.125. https://www.sciencedirect.com/science/article/pii/S0168900216306957.

Allison, John, et al. 2006. "Geant4 developments and applications." *IEEE Trans. Nucl. Sci.* 53:270. https://doi.org/10.1109/TNS.2006.869826.

Amadio, G, Ananya, J Apostolakis, M Bandieramonte, S Behera, A Bhattacharyya, R Brun, et al. 2018. "GeantV alpha release" [in en]. *Journal of Physics: Conference Series* 1085 (September): 032037. ISSN: 1742-6588, 1742-6596, accessed July 11, 2023. https://doi.org/10.1088/1742-6596/1085/3/032037. https://iopscience.iop.org/article/10.1088/1742-6596/1085/3/032037.

Amadio, G., A. Ananya, J. Apostolakis, M. Bandieramonte, S. Banerjee, A. Bhattacharyya, C. Bianchini, et al. 2020. *GeantV: Results from the prototype of concurrent vector particle transport simulation in HEP* [in en]. ArXiv:2005.00949 [hep-ex, physics:physics], September. Accessed December 8, 2023. http://arxiv.org/abs/2005.00949.

Apostolakis, J, M Bandieramonte, G Bitzes, R Brun, P Canal, F Carminati, G Cosmo, et al. 2015. "Towards a high performance geometry library for particle-detector simulations" [in en]. *Journal of Physics: Conference Series* 608 (May): 012023. ISSN: 1742-6588, 1742-6596, accessed December 6, 2019. https://doi.org/10.1088/1742-6596/608/1/012023. http://stacks.iop.org/1742-6596/608/i=1/a=012023?key=crossref.bdd3f6ed96e42e9beb181cc658a48d4f.

Atchley, Scott, Christopher Zimmer, John Lange, David Bernholdt, Veronica Melesse Vergara, Thomas Beck, Michael Brim, et al. 2023. "Frontier: Exploring Exascale." In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis.* SC '23. Denver, CO, USA: Association for Computing Machinery. https://doi.org/10.1145/3581784.3607089. https://doi.org/10.1145/3581784.3607089.

ATLAS Collaboration. 2021. *Athena.* V. 21.0.127, May. https://doi.org/10.5281/zenodo.2641996. https://zenodo.org/records/4772550.

Bandieramonte, Marilena, Riccardo Maria Bianchi, and Joseph Boudreau. 2020. "FullSimLight: ATLAS standalone Geant4 simulation." *EPJ Web Conf.* 245:02029. https://doi.org/10.1051/epjconf/202024502029.

Bell, Nathan, and Jared Hoberock. 2012. "Thrust: A productivity-oriented library for CUDA." In *GPU computing gems Jade edition,* 359–371. Elsevier.

Brun, Rene, Fons Rademakers, Philippe Canal, Axel Naumann, Olivier Couet, Lorenzo Moneta, Vassil Vassilev, et al. 2020. *root-project/root: v6.18/02.* V. v6-18-02, June. https://doi.org/10.5281/zenodo.3895860. https://doi.org/10.5281/zenodo.3895860.

Canal, Philippe. 2019. *Geant Exascale Pilot Project* [in en].

Celeritas (CMSSW fork). https://github.com/celeritas-project/cmssw.

Center, National Energy Research Scientific Computing. 2022. *Perlmutter.* https://www.nersc.gov/systems/perlmutter.

CMS Collaboration. 2021. *The Phase-2 upgrade of the CMS Data Acquisition and High Level Trigger Technical Design Report* [in en]. Technical report CERN-LHCC-2021-007. CERN, June.

CMS Offline Software and Computing. 2021. *Evolution of the CMS Computing Model towards Phase-2.* CMS Note 2021/001. Geneva. Switzerland: CERN, January. https://cds.cern.ch/record/2751565/files/NOTE2021_001.pdf.

Costanzo, Davide. 2021. *SWIFT-HEP,* April. https://swift.hep.ac.uk/overview.

Dormand, J. R., and P. J. Prince. 1980. "A family of embedded Runge-Kutta formulae." *Journal Computational and Applied Mathematics* 6, no 1.

Facility, Oak Ridge Leadership Computing. 2018. *Summit: Oak Ridge National Laboratory's next High Performance Supercomputer,* April. https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit.

Hamilton, Steven P., and Thomas M. Evans. 2019a. "Continuous-energy Monte Carlo neutron transport on GPUs in the Shift code" [in en]. *Annals of Nuclear Energy* 128 (June): 236–247. ISSN: 03064549, accessed January 17, 2019. https://doi.org/10.1016/j.anucene.2019.01.012.

———. 2019b. "Continuous-energy Monte Carlo neutron transport on GPUs in the Shift code." *Annals of Nuclear Energy* 128:236–247. ISSN: 0306-4549.

Henriques, A. 2015. "The ATLAS tile calorimeter." In *2015 4th International Conference on Advancements in Nuclear Instrumentation Measurement Methods and their Applications (ANIMMA),* 1–7. https://doi.org/10.1109/ANIMMA.2015.7465554.

Johnson, Seth R., Rob Lefebvre, and Kursat Bekar. 2023. *ORANGE: Oak Ridge Advanced Nested Geometry Engine.* Technical report ORNL/TM-2023/3190. (pending publication). Oak Ridge National Laboratory.

Johnson, Seth R., Stefano C. Tognini, Philippe Canal, Thomas Evans, Soon Yung Jun, Guilherme Lima, Amanda Lund, and Vincent R. Pascuzzi. 2021. "Novel features and GPU performance analysis for EM particle transport in the Celeritas code" [in en], edited by C. Biscarat, S. Campana, B. Hegner, S. Roiser, C.I. Rovelli, and G.A. Stewart. *EPJ Web of Conferences* 251:03030. ISSN: 2100-014X, accessed March 31, 2023. https://doi.org/10.1051/epjconf/202125103030.

Khan, Awais, Hyogi Sim, Sudharshan S. Vazhkudai, Ali R. Butt, and Youngjae Kim. 2021. "An Analysis of System Balance and Architectural Trends Based on Top500 Supercomputers" [in en]. In *The International Conference on High Performance Computing in Asia-Pacific Region,* 11–22. Virtual Event Republic of Korea: ACM, January. ISBN: 978-1-4503-8842-9, accessed November 13, 2023. https://doi.org/10.1145/3432261.3432263. https://dl.acm.org/doi/10.1145/3432261.3432263.

Klyukhin, V.I., A. Ball, F. Bergsma, D. Campi, B. Cure, A. Gaddi, H. Gerwig, et al. 2008. "Measurement of the CMS Magnetic Field" [in en]. *IEEE Transactions on Applied Superconductivity* 18, no. 2 (June): 395–398. ISSN: 1051-8223, 1558-2515, accessed November 25, 2023. https://doi.org/10.1109/TASC.2008.921242. http://ieeexplore.ieee.org/document/4520242/.

Lachnit, Stephan, Lorenzo Pezzotti, and Dmitri Konstantinov. 2022. *Standalone Geant4 validation of the ATLAS Tile Calorimeter* [in en]. Technical report CERN-STUDENTS-Note-2022-069. CERN, August.

Marc Verderi and Alberto Ribon. 2021. *Geant4 Introduction,* April. Accessed January 21, 2022. https://indico.cern.ch/event/1028379/.

Marsaglia, George. 2003. "Xorshift RNGs." *Journal of Statistical software* 8:1–6.

Pandya, Tara M., Seth R. Johnson, Thomas M. Evans, Gregory G. Davidson, Steven P. Hamilton, and Andrew T. Godfrey. 2016. "Implementation, capabilities, and benchmarking of Shift, a massively parallel Monte Carlo radiation transport code." *Journal of Computational Physics* 308 (March): 239–272. ISSN: 00219991. https://doi.org/10.1016/j.jcp.2015.12.037.

Romano, Paul K, Nicholas E Horelik, Bryan R Herman, Adam G Nelson, Benoit Forget, and Kord Smith. 2015. "OpenMC: A state-of-the-art Monte Carlo code for research and development." *Annals of Nuclear Energy* 82:90–97.

Sharvit, Yehonathan. 2022. *Data-Oriented Programming: Reduce Software Complexity.* Simon / Schuster.

The ATLAS Collaboration. 2020. *ATLAS HL-LHC Computing Conceptual Design Report.* Technical report CERN-LHCC-2020-015/LHCC-G-178. CERN, November.

Tognini, S. C., P. Canal, T. M. Evans, G. Lima, A. L. Lund, S. R. Johnson, S. Y. Jun, V. R. Pascuzzi, and P. K. Romano. 2022. *Celeritas: GPU-accelerated particle transport for detector simulation in High Energy Physics experiments* [in en]. ArXiv:2203.09467 [hep-ex, physics:physics], March. Accessed March 31, 2023. http://arxiv.org/abs/2203.09467.

Wächter, Carsten, and Alexander Keller. 2006. "Instant Ray Tracing: The Bounding Interval Hierarchy." In *Symposium on Rendering,* edited by Tomas Akenine-Moeller and Wolfgang Heidrich. The Eurographics Association. ISBN: 3-905673-35-5. https://doi.org/10.2312/EGWR/EGSR06/139-149.

Winkler, Joab R. 1993. "Numerical recipes in C: The art of scientific computing, second edition" [in en]. *Endeavour* 17, no. 4 (January): 201. ISSN: 01609327, accessed August 7, 2019. https://doi.org/10.1016/0160-9327(93)90069-F. https://linkinghub.elsevier.com/retrieve/pii/016093279390069F.

Workman, R. L., et al. 2022. "Review of Particle Physics." *PTEP* 2022:083C01. https://doi.org/10.1093/ptep/ptac097.