# Celeritas: accelerating Geant4 with GPUs*

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

[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

**Abstract.** Celeritas [1] is a new Monte Carlo (MC) detector simulation code designed for computationally intensive applications (specifically, High Luminosity Large Hadron Collider (HL-LHC) simulation) on high-performance heterogeneous architectures. In the past two years Celeritas has advanced from prototyping a 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 0.3, 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 integration of Celeritas into high energy physics (HEP) experimental frameworks such as CMSSW. On the Summit supercomputer, Celeritas performs EM physics between 6× and 32× faster using the machine's Nvidia GPUs compared to using only CPUs. 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 1.8–2.3× on GPU and 1.2× on CPU.

## 1 Introduction

The Celeritas particle transport code, started in early 2020, has been developed as an extensible library for detector simulations on GPUs. Its initial goal is to implement the EM physics used in production runs of the main LHC experiments, since those are the most computationally intensive. Less than six months after its inception, it demonstrated effective GPU performance gains for a proof-of-concept physics demonstration [2]. A year after that, basic EM physics was implemented along with a tracking loop to perform simple comparisons against the Geant4 detector simulation code [3] for simple problems [4]. Now, Celeritas has implemented sufficient physics to compare performance on a range of problems and to replace Geant4's `G4EmStandardPhysics` by "offloading" $e^{\pm}$ and $\gamma$ tracks to the GPU. The latest version includes facilities for seamlessly and almost trivially integrating Celeritas into existing Geant4 applications, sending tracks to Celeritas and sending detector hits back to the user application.

*This manuscript has been authored in part by UT–Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The publisher acknowledges the US government license to provide public access under the DOE Public Access Plan.

**e-mail: johnsonsr@ornl.gov

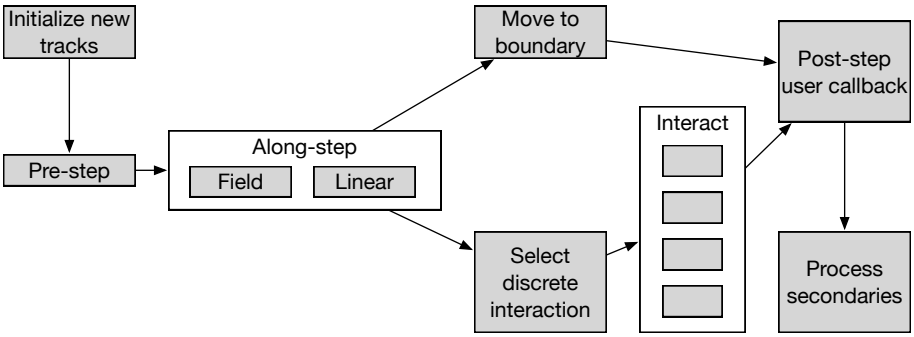



Figure 1: Action dependency graph in Celeritas. Each gray box is an action (which launches one or more kernels), and the white boxes are groups of actions that are selected on a track-by-track basis by a previous kernel.

## 2 Implementation

Celeritas was 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.

Every computational kernel is based on a function object parameterized on a thread ID, which in the current 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 [5], 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.

### 2.1 Tracking loop

The dependencies between kernels are constructed at run time by categorizing the kernels. Figure 1 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.

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 [2]). 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 [6] due to recursive virtual function calls.

As indicated in Fig. 1, two separate along-step kernels are launched when running a problem 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.

### 2.2 Geant4 integration

To integrate effectively into experimental frameworks and user code, Celeritas provides straightforward APIs 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. Figure 2 depicts the integration between a Geant4 user application, the Geant4 library code, and Celeritas.

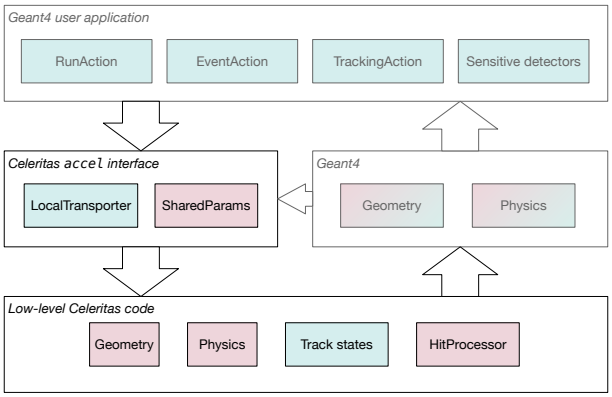


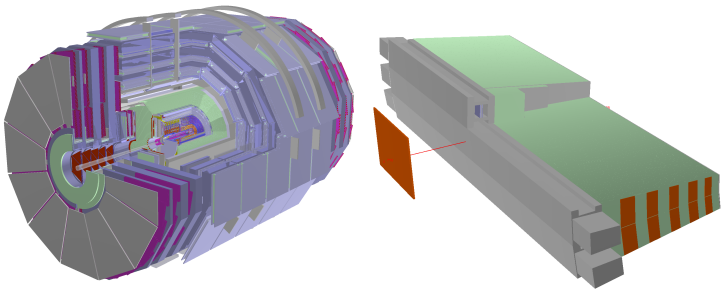

Figure 2: Depiction of Celeritas/Geant4 integration. Blue boxes are thread-local, and red boxes are shared. The blue-red boxes represent Geant4 "split" classes that have both shared and thread-local attributes.

During problem setup, Celeritas queries the Geant4 physics data (processes, models, materials, and particles) and sets up its problem internals accordingly. It then queries the global detector geometry setup and translates it to a VecGeom geometry. Logical volumes with sensitive detectors are also mapped to VecGeom volume IDs. These problem parameters are copied to GPU once and shared among all tracks.

Multithreading and multitasking are critical to achieving performance in Geant4 on modern multicore CPUs, and Celeritas has been designed to work seamlessly with any threading model. Because Celeritas carefully differentiates between shared problem parameters and local states, it is not necessary for any part of Celeritas to be declared `thread_local`. Clients simply need to provide a mapping between a Celeritas stream ID and their task data model: for regular Geant4 multithreaded applications, this is simply the Geant4 thread ID. All Celeritas streams in an application instance share a single GPU, relying on the underlying device management library to pipeline and execute the kernels using its stream implementation.

During execution, the Geant4 user `PreUserTrackingAction` queries the track's particle type. If it is an EM particle, it "kills" the Geant4 track after copying the key track attributes to a buffer for later "resurrection" in the Celeritas tracking loop. At the end of the event or when the buffer is full, the stored tracks are run through the main Celeritas tracking loop. Future work will provide an alternate interface for integrating as a "fast model" that can be used to selectively accelerate a single region of a detector or to send particles back to the Geant4 tracking loop when lepto-nuclear production is incorporated.

At each step iteration, a Celeritas kernel queries whether the tracks are in any sensitive detector (SD) volumes. If so, user-selected data (position, time, energy deposition, etc.) for those tracks is copied to the CPU. A loop over the tracks reconstructs Geant4 track, step, and hit objects, which are passed directly into the user SD "hit" method. For applications



(a) CMS-2018 (b) ATLAS tile calorimeter

Figure 3: Detector geometries for two of the Celeritas test problems.

that use detailed physical volume information (such as local position and replica number as needed by the Compact Muon Solenoid (CMS) High Granularity Calorimeter (HGCAL) [7]), a `G4Navigator` object finds the correct volume in the detector based on its ID.

For now, the developers expect most users to use the default SD integration because of its simplicity and reasonable efficiency. However, an extension point in Celeritas allows users to replace the hit reconstruction code with arbitrary user-based SDs for improved performance.

## 3 Results

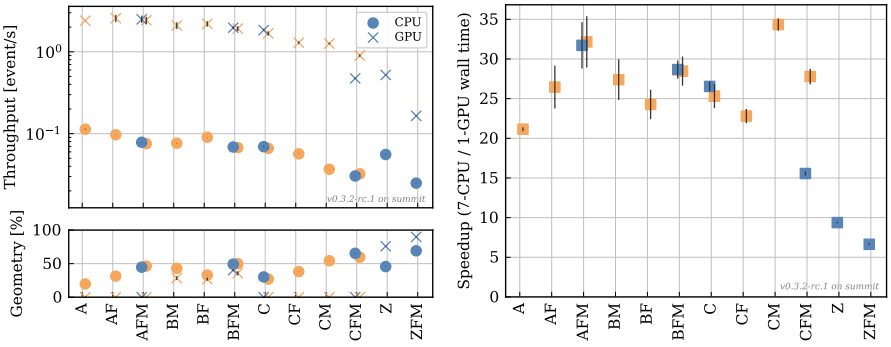
The performance of Celeritas as part of a Geant4 workflow, relative to standalone Geant4, will depend on the standalone performance of Celeritas EM particle tracking rate on the GPU and the overhead of the Geant4 conversion layer. To evaluate the performance of Celeritas on its own, a series of benchmark problems of increasing complexity, up to and including a full simulation of EM tracks in one of the Large Hadron Collider (LHC) Run 2 configurations of CMS (Fig. 3a), are regularly run on the Summit [8] supercomputer. To demonstrate performance as a part of a realistic Geant4 workflow, a second test problem uses Celeritas to accelerate the EM tracks in a test problem from ATLAS (Fig. 3b) [9]. Follow-on work will examine the performance difference between implementing the SDs on device versus reconstructing CPU hits.

### 3.1 Performance regression suite

The performance of Celeritas is regularly measured using a set of benchmarks [4] 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 $1300 \times 10$ GeV electrons per event at an energy of 10 GeV each. 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.

Each benchmark problem is run on the Summit supercomputer at Oak Ridge Leadership Computing Facility (OLCF) with a single compute node, which has six Nvidia V100 GPUs where each GPU is assigned to seven user-accessible IBM Power9 CPUs [8]. Six separate instances of each problem configuration are 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 GPU, and a second set executes an OpenMP-multithreaded run of seven CPUs per instance. Comparing the two gives an effective measure of the GPU-per-CPU core equivalence for each benchmark. Figure 4 shows (a) the raw throughput measured as events per second of wall time for a single instance and (b) the relative speedup of using GPUs versus neglecting them. The GPU-to-CPU equivalence is the speedup multiplied by the 7 CPUs used in each run:



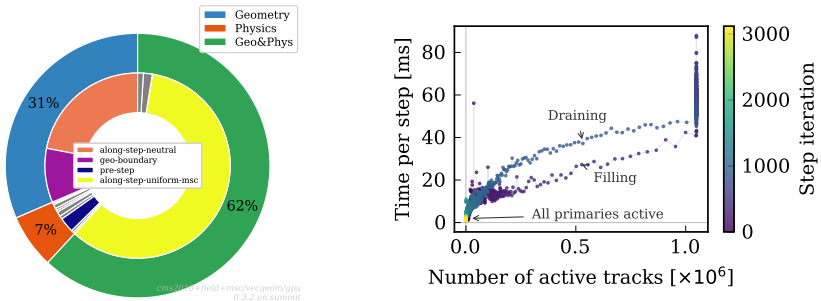

(a) Throughput (b) Relative performance

Figure 4: Absolute and relative performance of the regression tests with (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 Oak Ridge Adaptable Nested Geometry Engine (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 245 cores of an IBM Power9.

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. 4). In that problem, using Celeritas' in-house, surface-based ORANGE navigator results in almost 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. 5a, 91% 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 5b shows the cost per step iteration as a function of active tracks. 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.




(a) Action compute time (GPU)

(b) Time per step

Figure 5: Timing results for CMS-2018 (with field and multiple scattering) on GPU. Fig. 5a 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. 5b shows the distribution of wall time per step iteration (with one to ~ 1M particles per iteration).

### 3.2 ATLAS tile calorimeter

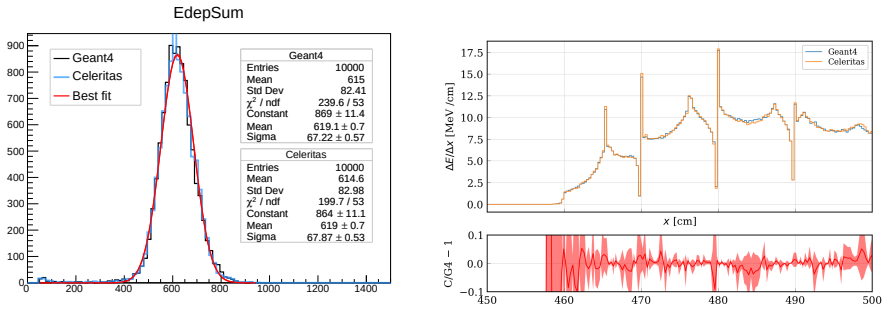
The Celeritas team integrated EM offloading into a standalone ATLAS tile calorimeter [10] test beam application developed for previous Geant4 validation [11], which models one of the 64 hadronic calorimeter modules installed around ATLAS. This integration was then adapted into the plugin-based architecture of the FullSimLight [12] 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 $\pi^+$ 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 § 2.1. Each simulation is run using a quarter of a node on the Perlmutter supercomputer [13]: a single Nvidia A100 GPU and 32 threads on an AMD EPYC 7763 CPU. The threads are pinned to one NUMA node (16 physical cores) and use the closest GPU.

The results from the sensitive detector output (Fig. 6, 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.

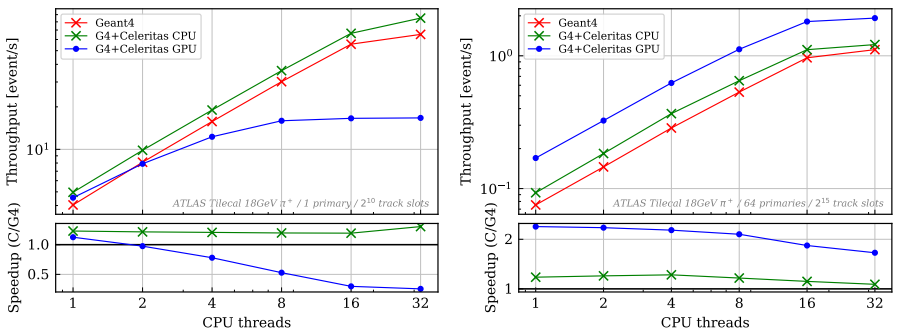
Figure 7 illustrates the strong scaling of Celeritas and Geant4 when only a single GPU is shared among the CPUs. Due to the small amount of work per event, Fig. 7a shows that the overhead of coupling Celeritas to Geant4 exceeds the potential GPU-based performance boost, especially as the hardware resources 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. 7b 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




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

Figure 6: Comparison of Celeritas and Geant4 energy deposition in the ATLAS tile calorimeter problem as a distribution over events (a) and binned as a function of $z$ (b).




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

Figure 7: Comparison of Celeritas and Geant4 scaling with the number of CPU threads. The test execution uses only one primary per event in Fig. 7a, whereas Fig. 7b has 64 primaries per event.

Celeritas, transferring hits back to CPU, and reconstructing Geant4 hit objects. Even with 32 hyperthreads sharing a single GPU, Celeritas has a speedup of 1.8×.

## 4 Conclusion

Celeritas now has the ability to directly integrate into Geant4 applications and transport EM tracks on GPU or CPU. Its whole-application speedup of 80% by using GPU resources in high performance computing (HPC) hardware for realistic LHC detector simulations makes a strong case for its integration as an EM accelerator. The fact that Celeritas shows a performance boost of 20% even without a GPU motivates further testing in a wider variety of applications. The performance gain by enabling GPUs in standalone Celeritas demonstrates the potential for additional improvements if more physics is added to Celeritas, and the performance analysis of the CMS execution suggests even greater performance may be obtained from the GPU by optimizing the geometry engine.

With the new Geant4 integration capabilities and demonstrated performance gains, Celeritas is ready for integration and validation in experiment frameworks. Integration tasks are under way, and future work will compare performance and accuracy in production CMS and ATLAS runs.

## Acknowledgements

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

[1] S.R. Johnson, A. Lund, S.Y. Jun, S. Tognini, G. Lima, P. Romano, P. Canal, B. Morgan, T. Evans, *Celeritas* (2022), published: [Computer Software] https://doi.org/10.11578/dc.20221011.1, https://doi.org/10.11578/dc.20221011.1

[2] S.R. Johnson, S.C. Tognini, P. Canal, T. Evans, S.Y. Jun, G. Lima, A. Lund, V.R. Pascuzzi, EPJ Web of Conferences **251**, 03030 (2021)

[3] S. Agostinelli, J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Arce, M. Asai, D. Axen, S. Banerjee, G. Barrand et al., Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **506**, 250 (2003)

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

[5] S.P. Hamilton, T.M. Evans, Annals of Nuclear Energy **128**, 236 (2019)

[6] S. Wenzel, J. Apostolakis, G. Cosmo, EPJ Web Conf. **245**, 02024 (2020)

[7] CMS Collaboration, Tech. Rep. CERN-LHCC-2021-007 (2021)

[8] Oak Ridge Leadership Computing Facility, *Summit: Oak Ridge National Laboratory's next high performance supercomputer* (2018), https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit

[9] The ATLAS Collaboration, Tech. Rep. CERN-LHCC-2020-015/LHCC-G-178, CERN (2020)

[10] A. Henriques, *The ATLAS tile calorimeter*, in *2015 4th International Conference on Advancements in Nuclear Instrumentation Measurement Methods and their Applications (ANIMMA)* (2015), pp. 1–7

[11] S. Lachnit, L. Pezzotti, D. Konstantinov, Tech. Rep. CERN-STUDENTS-Note-2022-069, CERN (2022)
[12] M. Bandieramonte, R.M. Bianchi, J. Boudreau, EPJ Web Conf. **245**, 02029 (2020)
[13] National Energy Research Scientific Computing Center, *Perlmutter* (2022), https://www.nersc.gov/systems/perlmutter