Current and Future Performance of the CMS Simulation

Kevin Pedro[1,*] (on behalf of the CMS Collaboration)

[1]Fermi National Accelerator Laboratory, Batavia, IL, USA

Abstract. The CMS full simulation using Geant4 has delivered billions of simulated events for analysis during Runs 1 and 2 of the LHC. However, the HL-LHC dataset will be an order of magnitude larger, with a similar increase in occupancy per event. In addition, the upgraded CMS detector will be considerably more complex, with an extended silicon tracker and a high granularity calorimeter in the endcap region. Increases in conventional computing resources are subject to both technological and budgetary limitations, so novel approaches are needed to improve software efficiency and to take advantage of new architectures and heterogeneous resources. Several projects are in development to address these needs, including the vectorized geometry library VecGeom and the GeantV transport engine, which uses track-level parallelization. The current computing performance of the CMS simulation will be presented as a baseline, along with an overview of the various optimizations already available for Geant4. Finally, the progress and outlook for integrating VecGeom and GeantV in the CMS software framework will be discussed.

1 Introduction

The CMS experiment uses a dedicated software package called CMSSW [1] to produce Monte Carlo simulation events [2–4]. Primary physics processes are generated by programs such as PYTHIA [5, 6] and MADGRAPH [7, 8], and the output particle information is converted into the standard HEPMC format [9, 10]. The detector simulation employs models of various types of particle interactions with materials, called "physics lists", to propagate particles through the detector. This simulation uses the GEANT4 software [11–13]. Subsequently, the response from detector electronics is simulated in a dedicated digitization step to produce output signals, and then various reconstruction algorithms are applied to those signals. The detector simulation step is the most expensive in terms of CPU usage, consuming 40% of the total computing budget of CMS [14]. The other steps in producing Monte Carlo samples—event generation, digitization, reconstruction, and analysis—together consume 45%, with reconstruction as the largest contributor. The remainder of the CPU time is used to process observed data. Within the detector simulation step, evaluating the geometry and magnetic field propagation uses 60% of the time; electromagnetic (EM) physics models use 15%; hadronic physics use 10%; and other components, including CMS-specific operations, use the remaining 15%.

The high-luminosity upgrade to the LHC will have significant implications for the CMS computing budget. This motivates continued efforts to improve the speed and efficiency of

[*]e-mail: pedrok@fnal.gov

the detector simulation. More information about the CMS detector and planned upgrades is given in Section 2. CMS has activated numerous technical options and implemented several approximations to improve the CPU usage of the detector simulation. The impacts of these improvements will be discussed in Section 3. Potential improvements using vectorized libraries for geometry and simulation are being evaluated. Some progress has already been made towards integrating these libraries in CMSSW; the status and outlook will be described in Section 4.

2 The CMS detector and upgrades

The central feature of the CMS apparatus is a superconducting solenoid of 6 m internal diameter, providing a magnetic field of 3.8 T. Within the solenoid volume are a silicon pixel and strip tracker, a lead tungstate crystal electromagnetic calorimeter (ECAL), and a brass and scintillator hadron calorimeter (HCAL), each composed of a barrel and two endcap sections. Forward calorimeters extend the pseudorapidity coverage provided by the barrel and endcap detectors. Muons are detected in gas-ionization chambers embedded in the steel flux-return yoke outside the solenoid. A more detailed description of the CMS detector, together with a definition of the coordinate system used and the relevant kinematic variables, can be found in Ref. [15].

In the original design and installation of the CMS detector (“Phase 0”), the pixel system included 66 million channels, the strip tracker system 9.6 million channels, the ECAL 76 thousand channels, the HCAL 7 thousand channels, and the forward calorimeter 2 thousand channels. Leading up to Run 3 of the LHC, several detector upgrades have been or will be installed, called the “Phase 1” upgrades [16, 17]. These upgrades include a new pixel detector, which has 127 million channels. The readout electronics for the HCAL are also upgraded, increasing its channel count to 15 thousand.

In Run 2, the LHC is currently operating at a luminosity close to $2 \times 10^{34}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$, almost double its original design luminosity. After Run 3, it will undergo a high luminosity (HL) upgrade, becoming the HL-LHC. The HL-LHC will reach a luminosity of at least $5 \times 10^{34}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$ and potentially up to $7.5 \times 10^{34}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$. This will lead to a high rate of secondary proton-proton collisions, up to 200 interactions per bunch crossing (pileup) compared to an average of 30 in Run 2, and associated large radiation doses. To cope with these extreme conditions, the CMS detector will undergo a more substantial upgrade, called the “Phase 2” upgrade [18]. There will be a new pixel detector with 1947 million channels [19]. In addition, the endcap calorimeter system, both ECAL and HCAL, will be replaced with an integrated high granularity calorimeter (HGCal) [20]. The HGCal will have 6 million channels, a substantial increase from the current calorimeters. This may require more accurate physics lists in Geant4 in order to achieve desired levels of agreement between simulation and observation.

The upgrades to the CMS detector are reflected in the simulated geometry used in the CMS software. The original CMS geometry has 2.1 million elements, and the Phase 1 CMS geometry is similar. Because of the greater complexity of the Phase 2 pixel detector and HGCal, the Phase 2 CMS geometry has 21.9 million elements. These increases in channel counts and geometry elements translate into an increase in CPU usage for the simulation, depicted in Fig. 1.

The jump in the amount of data and the detector complexity for the HL-LHC upgrade will place numerous demands on CMS computing. We will need to simulate 10 times more events to keep up with the recorded data. The reconstruction algorithms will take longer to run because of higher occupancies from pileup and the aforementioned increases in detector channels. To compensate for the larger fraction of CPU needed for reconstruction, the detector simulation will need to use a smaller fraction. In total, the yearly CPU requirements are expected to grow by more than an order of magnitude compared to the current run [21]. For Phase 2, the CMS simulation must provide more events and more accuracy, with a more complicated geometry, while using a smaller fraction of the CPU budget. To achieve these goals, we must find significant improvements in efficiency, even beyond those already implemented in CMSSW.

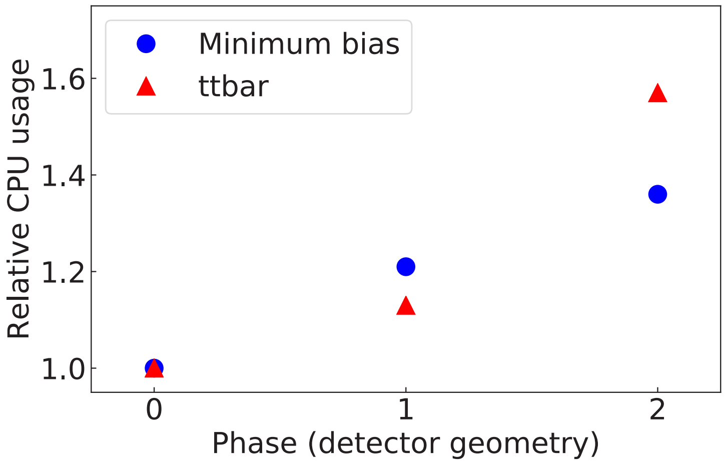

Figure 1. CPU usage for different CMS geometries, based on Table 1 from Ref. [14]. The values for each process type are normalized to the Phase 0 values.

3 Existing improvements

The CMS simulation includes a number of improvements over the baseline CPU performance from GEANT4. Some of these improvements are purely technical, while others are approximations that preserve the essential physical accuracy of the simulation while reducing the computational time, in some cases substantially. In Ref. [14], each of the improvements in the following list was profiled to determine its impact on CPU usage for simulations of several physics processes: minimum bias ("minbias") and top pair production ("ttbar" or $t\bar{t}$). The improvements considered are:

- **Static library**: all simulation code in CMSSW is compiled into one static library file, to avoid calls to the procedure linkage table that would occur if libraries were loaded dynamically.
- **Production cuts**: the minimum requirement on the expected flight distance of any secondary particles that would be created is customized for each detector region. For the pixel system, the cut is 0.01 mm; for the strip tracker, 0.1 mm; for the ECAL and HCAL, 1 mm; for the muon systems, 0.002 mm; and for the support structure, 1 cm.
- **Tracking cut**: within the vacuum chamber, between the interaction point and the start of the pixel system, charged particles with energy less than 2 MeV are rejected. This avoids the possibility of looping electrons or positrons.
- **Time cut**: the maximum propagation time considered is 500 ns.

Table 1. The impact of each improvement in the CMS simulation, for minbias and $t\bar{t}$ processes with the Phase 0 geometry and GEANT4 version 10.2. Each entry is normalized to the "No optimizations" row. The last entry is measured with all optimizations enabled together. This table is based on Tables 2 and 3 from Ref. [14].

	Relative CPU usage	
Configuration	Minbias	$t\bar{t}$
No optimizations	1.00	1.00
Static library	0.95	0.93
Production cuts	0.93	0.97
Tracking cut	0.69	0.88
Time cut	0.95	0.97
Shower library	0.60	0.74
Russian roulette	0.75	0.71
FTFP_BERT_EMM	0.87	0.83
All optimizations	0.21	0.29

- **Shower library**: a library of pre-generated showers is used for the forward calorimeter. Particle multiplicities are highest in the forward region, so this relatively small volume can take a disproportionate amount of CPU to simulate completely.
- **Russian roulette**: an algorithm which discards, at random, $N - 1$ neutrons with energy less than 10 MeV or photons with energy less than 5 MeV in the calorimeter system. The discarded particles' interactions with the detector are not simulated. The N^{th} particle is retained and assigned a weight of N to account for the energies of the discarded particles.
- **FTFP_BERT_EMM**: a modified physics list with a simplified model of multiple scattering used in most regions, except for HCAL and HGCal.

The impact of each improvement, as well as the overall impact, can be seen in Table 1. The shower library and Russian roulette improvements have the largest effects. Overall, the CMS simulation is 4.7 (3.4) times faster for minbias ($t\bar{t}$) processes, compared to the baseline version with no optimizations. This is a significant achievement made possible by years of work and collaboration. In absolute terms, the CMS simulation takes 4.3 sec/event (24.6 sec/event) for minbias ($t\bar{t}$), where 1 sec = 11 HS06 for the benchmark machine.

Multithreading is another important development. GEANT4 supports event-level multithreading with nearly perfect scaling up to the limit of physical cores (57 physical cores in a Xeon Phi test machine), and a further 30% gain from hyperthreading. The memory usage is reduced by a factor of 10 with respect to a multiprocessing approach with no memory sharing. The CMSSW framework supports multithreading [22] and this facility is employed in the production of simulated events. Similar gains in throughput are observed in the CMSSW framework, compared to the standalone GEANT4 tests described previously. The memory usage remains under 2 GB when using up to 8 threads, which is ideal for a production environment. Though the absolute time per event does not change, this improvement allows CMS to make more efficient use of grid resources.

CMS continues to explore further improvements to the CPU efficiency of the simulation. One such improvement is the use of VECGEOM [23, 24], a new library for detector geometry. This library supports vectorization and new computing architectures. The code is written to be more modern and efficient, compared to existing detector geometry libraries. GEANT4 supports the use of VECGEOM in scalar mode, without vectorization. CMS has tested this combination and observes a 7–13% speed improvement, depending on the process, with similar memory usage. This improvement comes just from the code improvements in the new library, because vectorization is not supported in Geant4. This improvement is included in the latest production releases of CMSSW and constitutes the first mainstream usage of a new vectorized library by a major experiment.

An even more recent set of improvements concerns the magnetic field evaluation. A more efficient stepping algorithm for tracking in the magnetic field, called **G4DormandPrince745**, has been compared to the Geant4 default algorithm, **G4ClassicalRK4**. The new algorithm uses fewer magnetic field evaluations to achieve the same integration accuracy. In addition, a smart tracking algorithm is employed for energy-dependent propagation of particles through EM fields. CMS observes an 8–10% speed improvement with these new optimizations in preliminary tests, conducted with gcc 7.0 and 16 threads. These tests were enabled by migration to Geant4 version 10.4 in CMSSW.

4 Potential improvements

CMS has already achieved significant speed improvements in Geant4 and enabled event-level multithreading for more efficient use of resources. However, even those improvements will not suffice to meet the demands of the HL-LHC and the Phase 2 detector upgrades. One potential step forward is GeantV, the vectorized transport engine [25]. While Geant4 allows event-level parallelism, GeantV is based on track-level parallelism: processing multiple events simultaneously in a single engine. It exploits single instruction, multiple data (SIMD) vectorization. By grouping tracks from multiple events—based on similarity in terms of particle type, geometry, or material—into a basket, the entire basket can be processed together. GeantV vectorizes all components of the simulation, including geometry navigation, magnetic field propagation, and physics algorithms. To accomplish this, it builds on lower-level libraries such as VecCore [26] and VecMath [27]. These libraries serve as abstraction layers to support different architectures, processors, and instruction sets.

It is important to have early tests of this new engine in experiments' software frameworks, in order to avoid incompatibilities between the threading models and interfaces that would limit adoption by the experiments. This effort began with integration into toy-mt-framework [28, 29], a standalone library used by CMS for multithreading research and development with Intel Thread Building Blocks (TBB) [30]. The integration was successful, and an example is included in the first public release of GeantV [25]. Subsequently, a working example has been developed within the CMSSW development release environment [1]. Repositories exist to install [31] and run [32] GeantV in CMSSW.

This example uses a new feature in CMSSW called ExternalWork, which enables asynchronous task-based processing. The ExternalWork feature has two steps: *acquire* and *produce*. The CMSSW module takes as input generated events in HepMC format, which are created by other CMSSW modules. In the *acquire* step, the HepMC input is converted to the native GeantV format and passed to the GeantV engine. A separate TBB task is then queued for the GeantV engine to process the event, along with any other events that may be processing in parallel. Along with the GeantV TBB task, the CMSSW framework can enqueue additional TBB tasks to do non-GeantV work, thereby sharing the available threads. Once GeantV finishes with an event, a callback function is executed, placing the second step, *produce*, into the task queue. In the *produce* step, the CMSSW module creates its output products and finishes with that event. The CMSSW geometry is also used directly by GeantV, in the ROOT geometry format [33, 34].

There are currently some limitations to the CMSSW integration of GeantV. A constant magnetic field and a limited EM-only physics list are used, rather than the full equivalents in the standard CMS simulation. Production cuts have not yet been included, for simplicity.

Sensitive detectors and scoring are not yet adapted to the new GeantV interfaces. With event-level parallelism, the scoring code that turns Geant4 hits into CMSSW data structures must only process one event at a time; separate instances of the scoring classes are created for each thread. In contrast, the track-level parallelism in GeantV implies that any scoring code must handle input from multiple events and multiple threads simultaneously. Work is ongoing to update the scoring code to the required level of thread safety. Because of this limitation, integration with downstream steps such as digitization has not yet been accomplished. However, once the scoring code is adapted, downstream steps will proceed without difficulty, as these rely on CMS event products rather than direct GeantV information.

The first public release of GeantV is not faster than Geant4, as the vectorization of various components is not yet complete. The second public release of GeantV, scheduled for 2019, will be faster, depending on the choice of computing architecture, processor instruction set, and other options. CMS plans to target the second public release of GeantV for complete integration and demonstration of a speed improvement in the CMSSW framework. Depending on these results, there will be a community decision to support the GeantV engine as part of Geant4 on the timescale of the HL-LHC upgrades.

5 Conclusions

The CPU usage of the CMS full simulation, which uses Geant4, has been substantially reduced. Various technical improvements and physics-preserving approximations make the simulation 3–5 times faster, compared to baseline performance. We continue to find $\approx$10% improvements in CPU usage from sources such as VecGeom and magnetic field algorithm optimizations. The simulation can run in multithreaded mode with event-level parallelism, enabling more efficient use of grid computing resources.

The high luminosity LHC upgrade and the corresponding CMS detector upgrades will bring significant challenges. There will be an order of magnitude increase in the amount of data, up to 200 proton-proton interactions per bunch crossing, and highly granular detector systems. The CMS simulation will be required to provide more events with more accuracy and a more complicated geometry, while using a smaller fraction of the total CPU budget.

GeantV is one promising approach to speed up the simulation even further. It uses track-level parallelism, rather than event-level, to exploit SIMD vectorization. The first public release of the software is available, and the second public release, including a speed improvement over Geant4, is planned for 2019. The first release has been successfully integrated in the CMSSW framework. The eventual goal is to improve CPU usage by 2–5 times in the final release.

References

[1] CMS Collaboration, *CMSSW*, [software] (2018), (accessed 2018-09-04), `https://github.com/cms-sw/cmssw`

[2] D.V. Elvira (CMS), NSS/MIC 2007 C07-10-28 **3**, 2081 (2007)

[3] D.J. Lange, M. Hildreth, V.N. Ivantchenko, I. Osborne (CMS), J. Phys. Conf. Ser. **608**, 012056 (2015)

[4] M. Hildreth, V.N. Ivanchenko, D.J. Lange (CMS), J. Phys. Conf. Ser. **898**, 042040 (2017)

[5] T. Sjöstrand et al., *Pythia8*, [software] (2017), version 8.230 (accessed 2018-09-04), `http://home.thep.lu.se/~torbjorn/pythia8/pythia8230.tgz`

[6] T. Sjöstrand, S. Ask, J.R. Christiansen, R. Corke, N. Desai, P. Ilten, S. Mrenna, S. Prestel, C.O. Rasmussen, P.Z. Skands, Comput. Phys. Commun. **191**, 159 (2015), `1410.3012`

[7] J. Alwall et al., *Madgraph5*, [software] (2018), version 2.3.3 (accessed 2018-09-04), `https://launchpad.net/mg5amcnlo`

[8] J. Alwall, R. Frederix, S. Frixione, V. Hirschi, F. Maltoni, O. Mattelaer, H.S. Shao, T. Stelzer, P. Torrielli, M. Zaro, JHEP **07**, 079 (2014), `1405.0301`

[9] HepMC project, *HepMC2*, [software] (2012), version 2.06.07 (accessed 2018-09-04), `http://lcgapp.cern.ch/project/simu/HepMC/download/HepMC-2.06.07.tar.gz`

[10] M. Dobbs, J.B. Hansen, Comput. Phys. Commun. **134**, 41 (2001)

[11] Geant4 project, *Geant4*, [software] (2016), version 10.02.p02 (accessed 2018-09-04), `https://github.com/Geant4/geant4/releases/tag/v10.2.2`

[12] S. Agostinelli et al., Nucl. Instrum. Meth. A **506**, 250 (2003)

[13] J. Allison et al., Nucl. Instrum. Meth. A **835**, 186 (2016)

[14] J. Apostolakis et al. (HEP Software Foundation) (2018), `1803.04165`

[15] CMS Collaboration, JINST **3**, S08004 (2008)

[16] CMS Collaboration, CMS Technical Design Report CERN-LHCC-2012-016, CMS-TDR-011, CERN (2012), `https://cds.cern.ch/record/1481838`

[17] CMS Collaboration, CMS Technical Design Report CERN-LHCC-2012-015, CMS-TDR-010, CERN (2012), `https://cds.cern.ch/record/1481837`

[18] CMS Collaboration, CMS Technical Proposal CERN-LHCC-2015-010, CMS-TDR-15-02, CERN (2015), `https://cds.cern.ch/record/2020886`

[19] CMS Collaboration, CMS Technical Design Report CERN-LHCC-2017-009, CMS-TDR-014, CERN (2017), `https://cds.cern.ch/record/2272264`

[20] CMS Collaboration, CMS Technical Design Report CERN-LHCC-2017-023, CMS-TDR-019, CERN (2017), `https://cds.cern.ch/record/2293646`

[21] A.A. Alves, Jr et al. (HEP Software Foundation) (2017), `1712.06982`

[22] C.D. Jones (CMS), J. Phys. Conf. Ser. **898**, 042008 (2017)

[23] VecGeom project, *VecGeom*, [software] (2017), version 00.05.00 (accessed 2018-09-04), `https://gitlab.cern.ch/VecGeom/VecGeom/tags/v00.05.00`

[24] J. Apostolakis et al., J. Phys. Conf. Ser. **608**, 012023 (2015)

[25] GeantV project, *GeantV*, [software] (2018), version 0.3.0 (alpha) (accessed 2018-09-04), `https://gitlab.cern.ch/GeantV/geant/tags/alpha`

[26] VecCore project, *VecCore*, [software] (2017), version 0.4.2 (accessed 2018-09-04), `https://github.com/root-project/veccore/releases/tag/v0.4.2`

[27] VecMath project, *VecMath*, [software] (2017), (accessed 2018-09-04), `https://github.com/root-project/vecmath`

[28] C.D. Jones, *toy-mt-framework*, [software] (2018), (accessed 2018-09-04), `https://github.com/Dr15Jones/toy-mt-framework`

[29] E. Sexton-Kennedy, P. Gartung, C.D. Jones, J. Phys. Conf. Ser. **898**, 042045 (2017)

[30] Intel, *Thread Building Blocks*, [software] (2018), (accessed 2018-09-04), `https://www.threadingbuildingblocks.org`

[31] K. Pedro, *install-geant*, [software] (2018), (accessed 2018-09-04), `https://github.com/kpedro88/install-geant`

[32] K. Pedro, *SimGVCore*, [software] (2018), (accessed 2018-09-04), `https://github.com/kpedro88/SimGVCore`

[33] ROOT project, *ROOT*, [software] (2018), version 6.12.07 (accessed 2018-09-04), `https://github.com/root-project/root/releases/tag/v6-12-06`

[34] R. Brun, F. Rademakers, Nucl. Instrum. Meth. A **389**, 81 (1997)