Current and Future Performance of the CMS Simulation

Kevin Pedro (FNAL)
on behalf of the CMS Collaboration
July 9, 2018
Overview

• CMS full simulation uses Geant4

• Sim is 40% of total CPU time used by CMS → most expensive “step” in MC production (vs. generation, digitization, reconstruction)

• Largest contributors to CPU usage in Geant4: geometry, magnetic field, EM physics

• CMS has implemented numerous technical options and approximations to improve CPU usage in the simulation

• Continue to explore new options and improvements

➢ Including GeantV, vectorized transport engine (currently in development)
The CMS Detector

“Phase 0”

**BRIL**
- Pixels
- Tracker

**ECAL**

**HCAL**

**Solenoid**

**Steel Yoke**

**Muons**

**STEEL RETURN YOKE**
- ~13000 tonnes

**SILICON TRACKER**
- Pixels (100 x 150 μm²)
- ~1m² ~66M channels
- Microstrips (80-180μm)
- ~200m² ~9.6M channels

**BRIL**
- Luminosity Telescope: ~200k Si pixels (100 x 150 μm²)
- Beam Monitors: 80 diamond sensors, 40 quartz counters

**CRYSTAL ELECTROMAGNETIC CALORIMETER (ECAL)**
- ~76k scintillating PbWO₄ crystals

**PRESHOWER**
- Silicon strips (6cm x 2mm)
- ~16m² ~137k channels

**SUPERCONDUCTING SOLENOID**
- Niobium-titanium coil carrying ~18000 A

**HADRON CALORIMETER (HCAL)**
- Brass + plastic scintillator
- ~7k channels

**FORWARD CALORIMETER**
- Steel + quartz fibres
- ~2k channels

**MUON CHAMBERS**
- Barrel: 250 Drift Tube & 480 Resistive Plate Chambers
- Endcaps: 473 Cathode Strip & 432 Resistive Plate Chambers

<table>
<thead>
<tr>
<th>Total weight</th>
<th>14000 tonnes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall diameter</td>
<td>15.0 m</td>
</tr>
<tr>
<td>Overall length</td>
<td>28.7 m</td>
</tr>
<tr>
<td>Magnetic field</td>
<td>3.8 T</td>
</tr>
</tbody>
</table>
The CMS Detector

Phase 1 Upgrade

127M

SILICON TRACKER

- Pixels (100 x 150 μm)
  - ~1m²  ~66M channels
- Microstrips (80-180μm)
  - ~200m²  ~9.6M channels

BRIL

- Pixels
- Tracker
- ECAL
- HCAL
- Solenoid
- Steel Yoke
- Muons

Total weight: 14000 tonnes
Overall diameter: 15.0 m
Overall length: 28.7 m
Magnetic field: 3.8 T

CRystal Electromagnetic Calorimeter (ECAL)

- ~76k scintillating PbWO₄ crystals

Preshower

- Silicon strips (6cm x 2mm)
  - ~16m²  ~137k channels

Hadron Calorimeter (HCAL)

- Brass + plastic scintillator
  - ~7k channels

Muhon Chambers

- Barrel: 250 Drift Tube & 480 Resistive Plate Chambers
- Endcaps: 473 Cathode Strip & 432 Resistive Plate Chambers
The CMS Detector

**Phase 1 Upgrade**

**Phase 2 Upgrade**

**SILICON TRACKER**
- Pixels (100 x 150 μm$^2$)
  - ~1m$^2$ ~66M channels
- Microstrips (80-180μm)
  - ~200m$^2$ ~9.6M channels

**BRIL**
- Luminosity Telescope: ~200k Si pixels (100 x 150 μm$^2$)
- Beam Monitors: 80 diamond sensors, 40 quartz counters

**ECAL**

**HCAL**

**Solenoid**

**Steel Yoke**

**Muons**

**STEEL RETURN YOKE**
- ~13000 tonnes

**SUPERCONDUCTING SOLENOID**
- Nickel-titanium coil carrying ~16000 A

**HADRON CALORIMETER (HCAL)**
- Brass + plastic scintillator
  - ~3k channels

**CRYSTAL ELECTROMAGNETIC CALORIMETER (ECAL)**
- ~76k scintillating PbWO$_4$ crystals

**PRESHOWER**
- Silicon strips (6cm x 2mm)
  - ~16m$^2$ ~13k channels

**FORWARD CALORIMETER**
- Steel + quartz fibres
  - ~2k channels

**MUON CHAMBERS**
- Barrel: 250 Drift Tube & 480 Resistive Plate Chambers
- Endcaps: 473 Cathode Strip & 432 Resistive Plate Chambers

**Total weight**: 14000 tonnes
**Overall diameter**: 15.0 m
**Overall length**: 28.7 m
**Magnetic field**: 3.8 T

Kevin Pedro (FNAL)
The CMS Detector

**BRIL**
- Pixels Tracker
- ECAL
- HCAL
- Solenoid
- Steel Yoke
- Muons

**Silicon Tracker**
- Pixels (100 x 150 μm$^2$)
- ~1m$^2$ ~66M channels
- Microstrips (80-180μm)
- ~200m$^2$ ~9.6M channels

**Luminosity Telescope**
- ~200k Si pixels (100 x 150 μm$^2$)
- Beam Monitors: 80 diamond sensors, 40 quartz counters

**Crystal Electromagnetic Calorimeter (ECAL)**
- ~76k scintillating PbWO$_4$ crystals

**Forward Calorimeter**
- Steel + quartz fibres
- ~2k channels

**Muon Chambers**
- Barrel: 250 Drift Tube & 480 Resistive Plate Chambers
- Endcaps: 473 Cathode Strip & 432 Resistive Plate Chambers

**Hadron Calorimeter (HCAL)**
- Brass + plastic scintillator
- ~7k channels

**Phase 1 Upgrade**

**Total weight**: 14000 tonnes
**Overall diameter**: 15.0 m
**Overall length**: 28.7 m
**Magnetic field**: 3.8 T

Kevin Pedro (FNAL)
The CMS Detector

Phase 1 Upgrade

Crystal Electromagnetic Calorimeter (ECAL)
~76k scintillating PbWO₄ crystals

Phase 2 Upgrade

High Granularity Calorimeter (HGCAL)
Silicon, scintillator
~6M channels
Challenges of Phase 2

- CMS Phase 0 and Phase 1 simulation geometries have 2.1 million elements
- Phase 2 geometry has 21.9 million elements: $10 \times \text{(Phase 1)}$
- This translates to an increase in CPU time for simulation

- Simulate more events to keep up with HL-LHC data volumes: $10 \times \text{(Phase 1)}$
- May also need to improve accuracy of physics lists to simulate HGCal
- Reconstruction will take longer due to high pileup and granular detectors
  - Need more events, more accuracy, in more complicated geometry… w/ relatively smaller fraction of total CPU usage
Existing Improvements

- **Static library**: avoid calls to procedure linkage table (PLT) for dynamic loading of libraries
- **Production cuts**: 0.01mm (pixel), 0.1mm (strip tracker), 1 mm (ECAL/HCAL), 0.002 mm (muon systems), 1 cm (support structure)
- **Tracking cut**: 2 MeV (within beampipe) → avoid looping electrons
- **Time cut**: 500 ns
- **Shower library**: use pre-generated showers in forward region (HF, ZDC, Castor)
- **Russian roulette**: discard N-1 neutrons < 10 MeV or gammas < 5 MeV (in calorimeters), retain N\(^{th}\) particle and assign it a weight of N
- **FTFP_BERT_EMM**: modified physics list, simplified multiple scattering model for most regions (default used for HCAL, HGCal)

> When all optimizations applied together, CMS achieves ~3–5× speedup!
Results of Existing Improvements

- From HEP Software Foundation Community White Paper
  - CMS Phase 0 detector, Geant4 10.2
- HF shower library, Russian Roulette have largest impacts
- Cumulative effects: with all improvements, simulation is $4.7\times$ ($3.4\times$) faster for MinBias (ttbar)
- CMS simulation takes 4.3 sec$^\dagger$/event (24.6 sec$^\dagger$/event) for MinBias (ttbar)

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Relative CPU usage</th>
<th>MinBias</th>
<th>ttbar</th>
</tr>
</thead>
<tbody>
<tr>
<td>No optimizations</td>
<td></td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Static library</td>
<td></td>
<td>0.95</td>
<td>0.93</td>
</tr>
<tr>
<td>Production cuts</td>
<td></td>
<td>0.93</td>
<td>0.97</td>
</tr>
<tr>
<td>Tracking cut</td>
<td></td>
<td>0.69</td>
<td>0.88</td>
</tr>
<tr>
<td>Time cut</td>
<td></td>
<td>0.95</td>
<td>0.97</td>
</tr>
<tr>
<td>Shower library</td>
<td></td>
<td>0.60</td>
<td>0.74</td>
</tr>
<tr>
<td>Russian roulette</td>
<td></td>
<td>0.75</td>
<td>0.71</td>
</tr>
<tr>
<td>FTFP_BERT_EMM</td>
<td></td>
<td>0.87</td>
<td>0.83</td>
</tr>
<tr>
<td>All optimizations</td>
<td></td>
<td>0.21</td>
<td>0.29</td>
</tr>
</tbody>
</table>

$^\dagger$1 sec = 11 HS06 for test machine
• Geant4 includes event-level multithreading
• Nearly perfect scaling with physical cores, further 30% gain from hyperthreading
• Memory reduced by factor of 10 (vs. multiprocessing approach)

• CMSSW framework supports multithreading
• Similar gains in throughput observed, memory usage remains under 2GB
  ➢ More efficient use of grid resources (included in CMS production releases)
New Improvements: Geometry

**VecGeom**: new library for detector geometry

- Supports vectorization and new architectures
- Code rewritten to be more modern and efficient (vs. Geant4, ROOT, USolids)
- Can be used in scalar mode with Geant4
- CMS observes *7–13% speedup* with similar memory usage
  → Just from code improvements, no vectorization!

- Included in latest CMS production releases
  - First mainstream use of vectorized library by experiment

<table>
<thead>
<tr>
<th>Geometry library</th>
<th>MinBias</th>
<th>ttbar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>VecGeom</td>
<td>0.87</td>
<td>0.93</td>
</tr>
</tbody>
</table>
New Improvements: Magnetic Field

• Faster stepper (G4DormandPrince745) for tracking in magnetic field
  o Also a more robust algorithm

• Smart tracking: energy-dependent propagation through EM fields

• CMS observes 8–10% speedup with these optimizations (preliminary)
  o Enabled by migration to latest Geant4 version 10.4

<table>
<thead>
<tr>
<th>Stepper</th>
<th>MinBias</th>
<th>ttbar</th>
</tr>
</thead>
<tbody>
<tr>
<td>G4ClassicalRK4</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>G4DormandPrince745</td>
<td>0.93</td>
<td>0.98</td>
</tr>
<tr>
<td>G4DormandPrince745 + smart tracking</td>
<td>0.92</td>
<td>0.90</td>
</tr>
</tbody>
</table>

(tested w/ gcc 7.0 and 16 threads)
Potential Improvements: GeantV

- CMS has already achieved significant speedups in Geant4 and enabled event-level multithreading for more efficient use of resources
- However, even this will not suffice for the demands of Phase 2
- Enter GeantV: Vectorized Transport Engine
  - Track-level parallelism: process multiple events simultaneously
  - Exploit single instruction, multiple data (SIMD) vectorization
  - Group similar tracks into basket (based on particle type, geometry/material)
  - Send entire basket to algorithm: process particles in parallel
Early Testing of GeantV in CMSSW

- Started with integration into toy-mt-framework → included in alpha release
  - Used for CMS multithreading R&D (Intel Thread Building Blocks)
  - Now have a working example compatible w/ CMSSW development release
- Run GeantV in “external loop” mode using CMSSW ExternalWork feature:
  - Asynchronous task-based processing
    - Co-development approach: test consistency of threading models, interfaces
      - Provide feedback to prevent divergence between CMS and GeantV
Elements of GeantV Integration

- **Generate** events in CMSSW framework, convert HepMC to GeantV format
- **Build** CMSSW geometry natively and pass to GeantV engine
  - Using **constant magnetic field**, limited **EM-only physics list**
  - Sensitive detectors and scoring not yet adapted to new interfaces
  - Production cuts also not yet included
- **First integration** of GeantV into experimental software framework
  - Run with elements specified above
  - Integration with downstream steps (e.g. digitization): longer timescale, requires more development for thread-safe scoring
- CMS will test GeantV **beta release**, targeting demonstration of speedup
  - Community decision to support GeantV engine as part of Geant4 on timescale of HL-LHC
Conclusions

• CMS has substantially reduced CPU usage of Geant4 full simulation
  o ~3–5× speedup using various technical improvements and physics-preserving approximations
  o Continue to find ~10% improvements, e.g. from VecGeom and magnetic field stepper/tracking optimizations

• HL-LHC and Phase 2 upgrades bring significant challenges:
  ➢ Need more events, more accuracy, in more complicated geometry… w/ relatively smaller fraction of total CPU usage

• GeantV is one promising approach to speed up full simulation even further
  o Track-level parallelism (rather than event-level), vectorized components
  o Alpha release is available, beta release planned for 2019
  ➢ Successful early integration in CMS software framework!
  o Aim for 2–5× speedup with final version
Acknowledgements

• Results and R&D presented here are the products of years of work by many scientists, developers, etc. – a (multi-) team effort!

• Thanks to:
  o Geant4 Collaboration
  o GeantV R&D Team
  o CMS Simulation Group
  o CMS Core Software Group
  o HEP Software Foundation
  o Support from Intel, Fermilab, and CERN OpenLab
References


Repositories

- **CMSSW** ([GitHub/cms-sw](https://github/cms-sw))
  - CMS Offline Software, ~6 million LOC
- **VecCore** ([GitHub/root-project](https://github/root-project))
  - SIMD abstraction library
  - Supports backends: Vc, UME::SIMD, CUDA
- **VecMath** ([GitHub/root-project](https://github/root-project))
  - Vectorized math utilities
  - Built on top of VecCore
- **VecGeom** ([CERN/GitLab](https://cern/gitlab))
  - Vectorized geometry and navigation, multi-particle interface
- **GeantV** ([CERN/GitLab](https://cern/gitlab))
  - Alpha release now available!
  - [cmsToyGV](https://gitlab/cms/sw) example
- **toy-mt-framework** ([GitHub/Drl15Jones](https://github/drl15jones))
  - Original toy framework for CMS multithreading development
- **install-geant** ([GitHub/kpedro88](https://github/kpedro88), **SimGVCore** ([GitHub/kpedro88](https://github/kpedro88))
  - Test repositories to install and integrate GeantV in CMSSW
Backup
CMS & LHC Upgrade Schedule

- Phase 1 upgrades began during Run 2 and will be in operation through the end of Run 3 (installation finishes during Long Shutdown 2)
- Phase 2 upgrades will be in operation during Runs 4, 5 (installation during Long Shutdown 3)
Simulation Test Details

- Machine: olhswepl6.cern.ch (CERN OpenLab)
- Single-threaded runs
- Compiler: gcc 6.3.0
- Geant4: version 10.2
- FTFP_BERT physics list
- Pythia event generation: $\sqrt{s} = 13$ TeV, 300 events, $|\eta| < 5.5$ (minimum bias, ttbar)
- Particle gun event generation: 50 GeV electrons, muons, pions flat distribution in $\eta = [-0.8, 0.8]$, $\eta = [2.0, 2.7]$, $\phi = [0, 2\pi]$
- Geometry: 2016 detector version (default)
Multithreading

← Standalone test

CMSSW framework ↓

Memory for ttbar at 13 TeV

CPU for ttbar at 13 TeV

4 threads
8 threads
8 threads forced init
External Work in CMSSW (1)

Setup:

• TBB controls running modules
• Concurrent processing of multiple events
• Separate helper thread to control external
• Can wait until enough work is buffered before running external process
Acquire:

- Module `acquire()` method called
- Pulls data from event
- Copies data to buffer
- Buffer includes callback to start next phase of module running
External Work in CMSSW (3)

Work starts:

• External process runs
• Data pulled from buffer
• Next waiting modules can run (concurrently)
External Work in CMSSW (4)

Work finishes:

- Results copied to buffer
- Callback puts module back into queue
Produce:

- Module `produce()` method is called
- Pulls results from buffer
- Data used to create objects to put into event