The CMS Software Performance at the Start of Data Taking

Gabriele Benelli
CERN

Technisches Seminar, DESY Zeuthen, October 21st 2008

Outline

• The CMS Experiment
• CMS Offline software
• Performance Profiling in CMS
• CMSSW Performance Suite
• Using CMSSW as a benchmarking tool
• Conclusions/Outlook
The CMS experiment

Tracker
- At detectors 40 TB/sec (40 MHz x 1 MB/evt)
- Level 1 + High Level Trigger reduce the event rate to 100 Hz (i.e. 100 MB/s)

4T Superconducting Solenoid
- Total: a few PetaBytes/year
- Offline reconstruction (simulation and analysis) done using the Grid, thousands of computers distributed worldwide, organized in Tier centers

3600 Collaborators
183 Institutes
58 Countries
18500 Tons
31m x 15m x 15m
4T Magnetic Field

Tiered Computing Model
CMS Offline Software

- CMS software framework (CMSSW) consists of approximately:
  - 1100 individual (CVS) packages, organized in 100 Subsystems
  - 2 Millions lines of code (SLOC)
  - 100 external packages (mainly Open Source)
  - 1.5 GB of “data” packages (mainly for Fast Simulation)

250 active developers
(a lot of work in the last 3 years!)

Development metrics

Source Lines Of Code (SLOC) in C++, Fortran, Python
[F77 contribution due to externally developed generators]

Configuration Language contribution
CMS Offline Development Model

- New tags get queued by developers and need to be blessed (i.e. tested) by the relevant level 2 manager for them to be collected by the Tag Collector

- Integration Builds:
  - Two per day for each release cycle and platform
  - Development and Production Releases
    - “open” and “close” phases
  - Full set of packages has to build always
  - Partial releases done later (FWLite, Online)

Performance Profiling

- The CMS Software Framework (CMSSW) rapid evolution requires strict quality assurance and the measurement and monitoring of performance is integral part of the Release Validation (RelVal) effort

- The 3 key metrics of software performance:
  - CPU Time
  - File Size
  - Memory
    - Number of CPUs and frequency
    - Disk and Tape Storage
    - RAM and Cache Memory Configuration

- The monitoring of these offers guidance to software development and optimization effort
Profilers in CMSSW

- CMSSW contains internal tools to profile itself in terms of CPU time, file size and memory use
- Also external tools (IgProf and Valgrind) are used to gain more information, with significant penalty in terms of execution time
- A Performance Suite of tools has been developed to make use of all these profiling tools, integrating them in the Release Validation, providing a regular measurement of CMSSW performance and allowing regression between software releases

The CMSSW Performance Suite

- Several physics processes (candles) have been selected among the ones used in RelVal, in order to probe all aspects of code
- All candles are profiled using the internal profilers, while for the more time consuming external tools specific candles and fewer events are used

<table>
<thead>
<tr>
<th>Internal Profilers</th>
<th>IgProf Profilers</th>
<th>Valgrind Profilers</th>
</tr>
</thead>
<tbody>
<tr>
<td>TimeReport, Timing Service, EdmEventSize, SimpleMemoryCheck</td>
<td>IgProfPerf, IgProfMem (Total), IgProfMem (Live), IgProfMem (Analyse)</td>
<td>ValgrindFCE (callgrind), Valgrind (memcheck)</td>
</tr>
<tr>
<td>100 events/candle</td>
<td>5 events/candle</td>
<td>1 event/candle</td>
</tr>
</tbody>
</table>

Candles:
- HiggsZZ4LM200
- MinBias
- SingleElectronE1000
- SingleMuMinusPt10
- SinglePiMinusE1000
- TTbar
- QCD_80_120
Profiling CMSSW Steps

- Performance profiling is based on simulated data and it is done for each step individually (output of one step used as input for the next).
- CMSSW pre-release development cycle is 1 week, so based on the different steps execution time, two 24hrs workflows have been foreseen, running in parallel on separate machines:

  ![Simulation Diagram]

  - GEN+SIM → DIGI
  - DIGI PILE-UP

  GEN+SIM+DIGI (Un-profiled)

  ![Trigger + Reconstruction Diagram]

  - DIGI PILE-UP ➔ L1 ➔ DIGI2RAW ➔ HLT ➔ RAW2DIGI+RECO

CMSSW Performance Suite

- In order to ensure meaningfulness to regression analysis, the Performance Suite is run on 2 dedicated multicore machines.
- The power-saving BIOS settings and daemons have been disabled and while we run on one core, we run a small cache-contained benchmark on all the other cores, to ensure the same load conditions.
- Once the profiling is done, the suite writes static html reports for each profile and archives a tarball of the working directory on tape.
- Finally, all the logs and static html with tables and graphs are published on a web server.
CPU Time Performance

• Sample regression CMSSW_3_1_X (G4 9.2 BETA) vs. CMSSW_2_1_9

CPU Time regression summary

GEN-SIM, DIGI

Change in processing time for each event between runs:

<table>
<thead>
<tr>
<th>Event Number</th>
<th>Processing Time Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.125</td>
</tr>
<tr>
<td>10</td>
<td>0.136</td>
</tr>
<tr>
<td>100</td>
<td>0.147</td>
</tr>
</tbody>
</table>

Intel 3GHz
4 cores
8GB RAM

Gabriele Benelli, CERN
DESY Zeuthen, October 21st 2008
## CPU Time Performance

### Sample regression

<table>
<thead>
<tr>
<th>Condition</th>
<th>GEN-SIM CPU</th>
<th>DIGI CPU</th>
<th>GEN-SIM ROOT</th>
<th>DIGI ROOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>QCD_80_120</td>
<td>104.67 ms</td>
<td>90.32 ms</td>
<td>1.01</td>
<td>0.95</td>
</tr>
<tr>
<td>TTbar</td>
<td>123.36 ms</td>
<td>110.79 ms</td>
<td>1.02</td>
<td>0.99</td>
</tr>
</tbody>
</table>

**CPU Time regression summary**

- **GEN-SIM**
- **DIGI**

### GEN+SIM

- **TTbar**

---

## File Size on disk

### Sample regression

<table>
<thead>
<tr>
<th>Condition</th>
<th>GEN-SIM</th>
<th>DIGI</th>
</tr>
</thead>
<tbody>
<tr>
<td>QCD_80_120</td>
<td>104.67 MB</td>
<td>90.32 MB</td>
</tr>
<tr>
<td>TTbar</td>
<td>123.36 MB</td>
<td>110.79 MB</td>
</tr>
</tbody>
</table>

**CPU Time regression summary**

- **GEN-SIM**
- **TTbar**

---

**Release CPU times**

**Total**

- **GEN-SIM**
- **DIGI**

**CPU and RAM**

- **Intel 3GHz 4 cores 8GB RAM**
Memory Performance

- Virtual memory size (VSIZE) is the parameter constrained in the CMS Computing TDR (1 GB/core).
- The Performance Suite can be used for regression to validate bug fixes with an arbitrary number of events.

Memory Allocation

- IgProf can profile memory allocation, preserving all the callstack information for each function, producing very insightful reports.
- Besides the memory information, the number of times the functions are called is relevant for CPU time profiling.
Memory Leaks

• Memory errors and leaks are hunted down with Valgrind MemCheck:

Valgrind MemCheck output (63 leaks)

Leak 1: 1,093,816 bytes indirectly lost (record 403 of 811) [brief]

Leak 2: 1,880,268 (104,992 direct, 1,775,276 indirect) bytes definitely lost (record 771 of 811) [brief]

Leak observed in SiPixelDigitizer, module observed already in CPU Time

CMSSW as a machine benchmarking tool
CMSSW benchmarking

- In the context of the HEPIX CPU performance working group, CMSSW has been used as a tool to do CPU benchmarking.
- The working group charge is to:
  - Test the validity of the industry-standard benchmarks (SPEC CPU) when compared with HEP experiments code.
  - Provide some recommendation to guide institutional purchases.
  - CMSSW applications have been used to benchmark a number of machines with different architecture.

All tests run using the Performance Suite, in CMSSW_2_0_0_pre5 release (making use of the CPU time profiling capability).

- All seven “candles” (physics processes) were used.
- Run 100 events per candle.
- Run GEN+SIM, DIGI, RECO steps separately.
- Run the 7 candles sequentially on each core.
- Four different tests:
  - Loading all cores simultaneously.
  - Loading 1, 3 (only for 4 cores machines) and 5 cores (only for 8 cores machines) with our application while running a CPU-intensive, cache-contained benchmarking tool (cmsScimark2) on the other cores.
Benchmarked machines

- Used 10 machines with different architectures, frequencies, memory:
  - 7 machines at CERN from the lxbench cluster
    - Cluster TWiki: https://twiki.cern.ch/twiki/bin/view/FIOgroup/TsiLxbench
  - 1 machine at DESY Zeuten (hpbl1)
  - 2 machines at INFN Padua (lxcmssrv7, lxcmssrv8)

<table>
<thead>
<tr>
<th></th>
<th>lxbench01</th>
<th>lxbench02</th>
<th>lxbench03</th>
<th>lxbench04</th>
<th>lxbench05</th>
<th>lxbench06</th>
<th>lxbench07</th>
<th>lxcmssrv07</th>
<th>lxcmssrv08</th>
<th>hpbl1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of cores</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Frequency (GHz)</td>
<td>2.8</td>
<td>2.8</td>
<td>2.2</td>
<td>2.66</td>
<td>3.0</td>
<td>2.6</td>
<td>2.33</td>
<td>2.33</td>
<td>2.1</td>
<td>2.83</td>
</tr>
<tr>
<td>Cache (could be L2/L3) (MB)</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>8</td>
<td>12</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>Memory (GB)</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Processor</td>
<td>Nocona</td>
<td>Irvingdale</td>
<td>Opteron 275</td>
<td>Woodcrest</td>
<td>Woodcrest</td>
<td>Opteron 2218 Rev.9</td>
<td>Clovertown</td>
<td>Xeon Harper/Town ES410</td>
<td>Xeon Harper/Town ES440</td>
<td>Opteron Barcelona 2352</td>
</tr>
<tr>
<td>Vendor</td>
<td>Intel</td>
<td>Intel</td>
<td>AMD</td>
<td>Intel</td>
<td>Intel</td>
<td>AMD</td>
<td>Intel</td>
<td>Intel</td>
<td>Intel</td>
<td>AMD</td>
</tr>
</tbody>
</table>

Benchmarking the cluster

- Basically the Performance Suite was submitted on the wanted cores with 100 events and all the internal profilers
- Once the Suite was done running, the logfiles were “harvested” and the timing information from the framework was used to calculate the average for each candle
- The data was collected in a Python dictionary and then tables for publication on the HEPiX Wiki were produced
- Using the same data structure, comparison/analysis plots for the various machines could be generated with Matplotlib
CMSSW Benchmarking

- The result of the benchmarking is seconds/event averaged on the 99 events (skipping the first one to avoid biases due to initialization).
- The results are reported in 3 formats:
  - seconds/event per core
  - events/second per core
  - events/second per machine
- Link: https://hepix.caspur.it/processors/dokuwiki/doku.php?id=benchmarks:cms

Comparing GEN+SIM

- All cores loaded
- AMD
GEN+SIM normalized by frequency

**Machine Maximum Performance Normalized by Frequency**

- **SinglePMinusE1000**
- **QCD_80_120**
- **TTbar**

![Graph showing performance normalized by frequency]

All cores loaded

AMD

GEN+SIM normalized by frequency and # cores

**Machine Maximum Performance Normalized by Frequency, Number Of Cores**

- **SinglePMinusE1000**
- **QCD_80_120**
- **TTbar**

![Graph showing performance normalized by frequency and number of cores]

All cores loaded
Comparing RECO

RECO normalized by frequency

All cores loaded
**RECO vs SPEC2000**

Machine Maximum Performance Normalized by SPEC2000

- SinglePMinusE1000
- QCD_80_120
- TTbar

**RECO normalized by frequency and # cores**

Machine Maximum Performance Normalized by Frequency, Number Of Cores

- SinglePMinusE1000
- QCD_80_120
- TTbar

All cores loaded
SIM vs SPEC2000

Machine Maximum Performance Normalized by SPEC2000

- SinglePiMinusE1000
- QCD_80_120
- TTbar

SIM vs SPECint2006

Machine Maximum Performance Normalized by SPECint2006

- SinglePiMinusE1000
- QCD_80_120
- TTbar
SIM vs SPECfp2006

Machine Maximum Performance Normalized by SPECfp2006

Scaling with multicores

Excellent scaling to the last core (cmsScimark2)
CMSSW Benchmarking Results

- Observed a different behavior in AMD vs. Intel machines for complex vs. simple events at the RECO step

- Compared CMSSW applications with SPEC benchmarks: differences due to architecture/type of event are larger than the differences between different SPEC benchmarks

- The CMSSW application scales nicely with the current multicore architectures

- A number of open issues from this first experience:
  - Statistical treatment of the data (number of events used, reproducibility, significance of the measurements)
  - Interpretation of the results (“one score” benchmark, weighting of several scores, picking representative candle(s), scores)
  - Graphical/data analysis

- The results of this work, done earlier this year, has inspired the development of a CMSSW benchmark utility that would address these issues and provide the necessary functionality to be used by Tier centers in assessing the CPU performance of machines
CMSSW benchmarking tools

- The idea is to have included in the CMSSW release a “suite” of benchmarking tools.

- The basic functionality would be to run a special command of the Performance Suite, then harvest the log files for the CPU timing information.

- The command above would return a score (maybe with its composition, in terms of multiple candles, or multiple processing steps, with relative weights, so that a Tier1 vs Tier2 could decide which score is most relevant for their use scenario).

- Since this kind of benchmarking usually involves more than one machine (since one is interested in comparing them), we thought of implementing server-client communication via XML-RPC in Python.

- Finally the data from multiple machines would have to be harvested from the logfiles, analyzed and reported in plots and scores.

Client/Server Benchmarking

Using XML-RPC Transport

Server 1
Benchmark Server Application

Server 2
Submit Job Specification
Return Profiling Data

Server 3

Benchmark Client Application
CMSSW benchmarking tool

• Then all any user would have to do, is to follow the APT based installation procedure for CMSSW on each machine intended to be benchmarked.

• On each machine launch an XMLRPC server

• On one machine launch the XMLRPC client, that ships the commands for the Performance Suite to the servers

• Wait for the Suite to finish on each machine (could do multicore running, or multiple runs on the same core(s)) and to report all results in term of a pickled file that contains dictionary data structure

• On this data structure run by default a basic analysis that would produce one score, a table with all results and the relevant comparison plots.

• Up to now the client/server functionality is under testing, the data analysis and score composition requires still some work

Conclusions

• The Performance Suite is an integral part of the CMSSW Release Validation process, providing developers with in-depth performance results and regression information within a 24 hrs cycle

• The Performance Suite can handle a lot of profile data and summarize it making it available and usable by developers,

• Besides its default behavior, it can be tweaked for more statistics, changing tests, single candles, new versions of external tools, new profiling tools, it can also be used as the core of the CMSSW benchmarking utility

• The framework is ready to handle the first collision data. A few improvements are in the works and the performance suite is used in testing and guiding optimization

• A command-line and XML-RPC based client/server CMSSW architecture benchmarking suite is being implemented and will soon ship with the release
The CMS Experiment

38 Countries, 183 Institutes, 3000 scientists and engineers (including 400 students)

**Muon Chambers**
- Barrel: Austria, Bulgaria, CERN, China, Germany, Hungary, Italy, Spain
- Endcap: Belarus, Bulgaria, China, Colombia, Korea, Pakistan, Russia, USA

**HCAL**
- Barrel: Bulgaria, India, Spain, USA
- Endcap: Belarus, Bulgaria, Georgia, Russia, Ukraine, Uzbekistan

**FEET**
- Pakistan

**FEET**
- China

**Preshower**
- Armenia, CERN, Greece, India, Russia, Taiwan

**CRYSTAL ECAL**
- Belarus, CERN, China, Croatia, Cyprus, France, Italy, Japan*, Korea, Switzerland, USA

**Superconducting Magnet**
- All countries in CMS contribute to Magnet financing in particular: Finland, France, Italy, Japan*, Korea, Switzerland, USA

<table>
<thead>
<tr>
<th>Total weight</th>
<th>12500 T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall diameter</td>
<td>15.0 m</td>
</tr>
<tr>
<td>Overall length</td>
<td>21.5 m</td>
</tr>
<tr>
<td>Magnetic field</td>
<td>4 Tesla</td>
</tr>
</tbody>
</table>

**Forward Calorimeter**
- Hungary, Iran, Russia, Turkey, USA

**Trigger, Data Acquisition & Offline Computing**
- Austria, Belgium, CERN, Finland, France, Germany, Italy, Japan*, Mexico, New Zealand, Spain, Switzerland, UK, USA

**Tracker**
- Barrel: Austria, Belgium, CERN, Finland, France, Germany, Italy, Japan*, Mexico, New Zealand, Spain, Switzerland, UK, USA

**RETURN YOKE**
- Barrel: Estonia, Germany, Greece, Russia
- Endcap: Japan*, USA

**Superconducting Magnet**
- All countries in CMS contribute to Magnet financing

**Forward Calorimeter**
- Hungary, Iran, Russia, Turkey, USA

**Cryogenic ECAL**
- * Only through industrial contracts

- Total weight: 12500 T
- Overall diameter: 15.0 m
- Overall length: 21.5 m
- Magnetic field: 4 Tesla
Points to make:

1. Automated procedure to measure and monitor performance
2. Part of Release Validation QA
3. Quick response time/ enough detailed information
4. Wide coverage of code, candles, processing steps
5. Robust (based on RelVal samples), options added
6. Used also for other tests
7. Re-used for benchmarking purposes
8. Can include robustness and reproducibility tests
9. All above... but we need to present

CPU:

1. Time vs.Evt + histo for MinBias SIM? (mention possibility to skip first event?) with and without regression
   In the same TimeReport breakdown by module (Tabella con tutte le candele)
   IgProfPerf con Regression
2. Size Tabella con tutte le sizes to see progressive increase,
   EdmSize with Regression
3. Memory
   SimpleMemcheck:
   (Round-up?), example of bumps and coming down, comparison between successive steps, or Pile-up
   IgProfMem con Regression

Key point in Results:

1. CPU Time: a table and a couple of plots
2. File Size: a regression plot of EdmEventSize or ls table
3. Memory: a plot of SimpleMemoryCheck/Total memory estimate from MemTotal (?)
Memory Performance

Simulation Test Beam Validation

Scintillator Saturation Effects
### Simulation Test Beam Validation

**Mean Energy Response**

ECAL+HCAL with pion beam

![Graph showing mean energy response](image1)

**Longitudinal Shower Profile**

HCAL

![Graph showing longitudinal shower profile](image2)

---

### GEANT4 Physics Lists

<table>
<thead>
<tr>
<th>GEANT4 Physics Lists</th>
<th>CPU Time (%)</th>
<th>Event Size (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MinBias</td>
<td>TTbar</td>
</tr>
<tr>
<td>QGSP_EMV</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>QGSP</td>
<td>116</td>
<td>120</td>
</tr>
<tr>
<td>QGSP_BERT_EMV</td>
<td>141</td>
<td>146</td>
</tr>
<tr>
<td>QGSP_BERT</td>
<td>158</td>
<td>169</td>
</tr>
</tbody>
</table>
A little bit of history

- Until the last release cycle (21X) CMSSW used a special configuration language to configure the (one and only) cmsRun executable

- The full transition to Python happened with 210 and came with a major improvement in maintainability: cmsDriver.py

- This script is a command-line utility that prepares a full python configuration file, based on a few command-line options, and launches cmsRun on it.

- This is highly configurable and covers most use cases

Details on Python tools

- Graphics to show the cmsDriver, cmsRelvalreport, cmsPerfSuite, cmsRelRegress, cmsPerfRegress, cmsRelvalreportInput ...

- Examples of uses of one and the other (customize fragment and its effect in the log, parsers and plot drawing, regression and publishing on a webserver)

- Issues about the machines used to do the measurements (for CPU only, but also for memory in case of ununderstood crashes)

- Uses of the suite for other aims:
  - tests on external packages (G4 reproducibility, its performance, robustness),
  - use as a machine benchmarking building block (moving to the second part of the talk)
Fast!