Speeding up CMS simulations, reconstruction and HLT code using advanced compiler options

N. Forzano (CERN), V. Innocente (CERN), V. Ivantchenko (CERN/Princeton), S. Muzaffar (CERN), D. Piparo (CERN)

ACAT 2022

CMS Central Data Processing Workflows: Small Recap

- Generation: creation of events of a type from a Monte Carlo Generator (e.g. Pythia, MadGraph, Sherpa …)
- Simulation: transform gen particles into simulated hits in the CMS detector. The tool used is Geant 4, hybridated with some detector specific fast simulation techniques (e.g. Russian Roulette for neutrons, parametrized forward showers)
- Digitization: transform sim hits into the response the detector would have had in presence of such energy depositions. In simulation, runs virtually always coupled to Pileup mixing.
- Pileup mixing: overlay a “pileup only” event to the hard scatter, reading it from a “Pileup library” which is a CMS dataset (typically placed at FNAL or CERN, accessing it through XRootD remote reads)
- HLT: Runs on data at Point 5 and can be emulated offline. A differently configured, extremely fast reconstruction to decide what events are interesting and why (i.e. according to which “trigger path)”
- Reconstruction: common to simulation and data. Transform digi output in collection of high level objects, e.g. particles (photons, electrons, muons, jets etc.)
- MiniAOD/NanoAOD: common to simulation and data. Creation of analysis formats from reconstructed events

Notes:

- CMS can go from gen to reco in one single step, called “Fast Monte Carlo Chain”, >much faster than the high fidelity Geant 4 based simulation + reconstruction. Simplifications are used all over the places.
- All gen-sim-digi/mix-reco-mini/nanoAOD creation steps in production happen through subsequent jobs landing on the same node, leaving intermediate datasets on the local disk (not all intermediate tiers are stored centrally, nor moved anywhere else. This approach is called in jargon “Step-chain Monte Carlo”

CMS O&C Strategic Goals

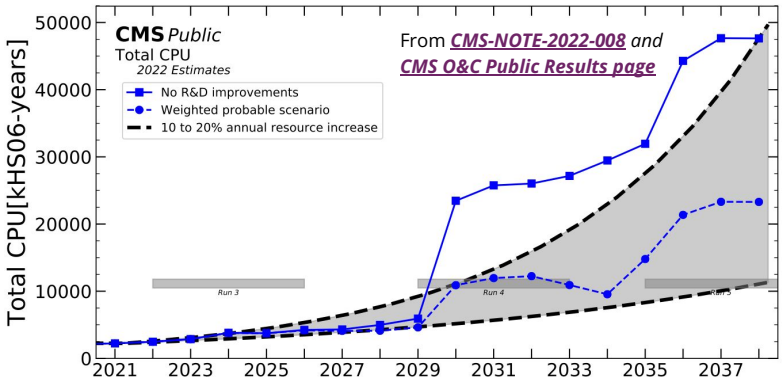
Efficiently use all of the resources available to us, while

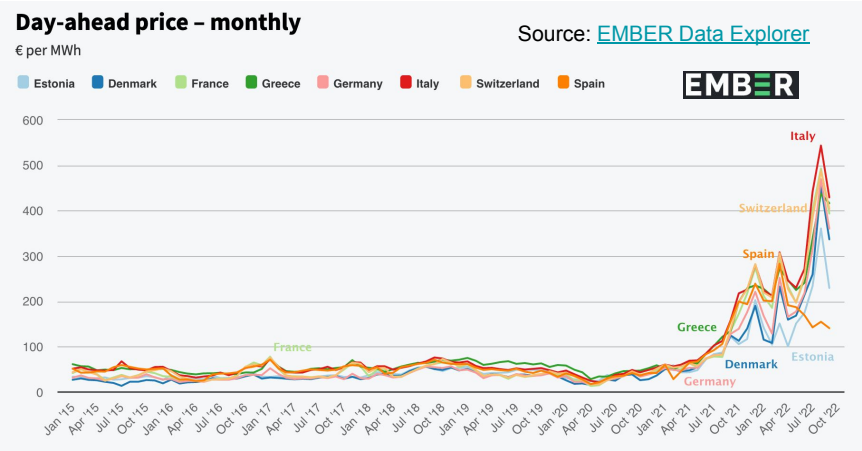
- Minimizing computing resource needs
- Maximizing throughput
- Minimizing job failures
- Minimizing manual operations (effort)
- Datasets requests are completed in short, predictable amounts of time (not completely under the control of O&C)

Why More Throughput?

- We must make efficient use of computing resources: this includes compute capacity
- Faster code means more flexibility: to schedule jobs, to plan sample production, to incorporate a last minute necessary feature boosting discovery potential if needed
- A powerful measure against scarcity
- One way of obtaining throughput increase: “technical improvements”, e.g. smarter usage of compilers

Projected CPU capacity needed by CMS in Run 4 and Run 5.





Link Time and Profile Guided Optimization (LTO & PGO)

- Two strategies to reduce application runtime, both relying on the compiler
  - No algorithmic or implementation changes, a “technical improvement”
- LTO: instrument compilation units with metadata, consulted to optimize when building shared objects
  - Expands the scope of inter-procedural optimizations to encompass everything that is visible at link time
- PGO: implies two compilation passes and one execution
  - Build instrumented binaries, produce a profile of the application, re-build from sources+profile
  - Inlining, block ordering, register allocation, conditional branch optimization, virtual call speculation, etc.
- Executive summary:
  - LTO has immediate benefits for all CMS applications: 2-3% speedup
  - PGO is effective for all flavours of sim, HLT and reco just by profiling a few processes
  - LTO and PGO improvements are almost uncorrelated, combined are worth ~10% speedup
  - Nothing CMS specific in the procedure followed: is there potential for other experiments?

Testbed and Results Description

- AMD EPYC 7302 16-Core Processor, exclusive usage
- GCC 10.3.0
- CMSSW_12_X cycle (2022 data taking release), Geant 4 10.7.2
- Improvements measured on event loop (thousands of events)
- Run 2 PU conditions
- CMS specific code re-built entirely. For sim tests, Geant 4 and VecGeom as well.
- PGO profiles obtained in sequential mode. If MT active, internal locks in instrumented binaries make runtime substantially identical.
- Improvements measured to be identical in sequential and MT mode using all cores
- Flags:
  - LTO: *-flto -fipa-icf -flto-odr-type-merging -fno-fat-lto-objects –Wodr*
  - PGO: *-Wno-error=maybe-uninitialized* and 1st pass *-fprofile-generate=profile-dir*, 2nd pass *-fprofile-use=profile-dir –fprofile-correction*

Simulation Based on Geant 4

One process to optimize them all

- LTO: choose TTBar simulation. **Runtime reduction: 3.2 %**
- **PGO: Need to verify that the profile generated with process_1 can also optimize process_X**
- Representative set of *standard candles* was chosen, a matrix was built
  - Build binaries with profiles obtained with a process_1 and run process_2

Events	**For Running →**	**150**	**384**	**384**	**384**	**150**
For Profiling ↓	Processes	TTBar	MinBias	$Z \rightarrow \mu\mu$	Single e	Phase-2 TTBar
25	TTBar	**10.7**	10.2	10.4	16.0	9.2
64	MinBias	8.9	9.5	10.8	**11.9**	9
64	$Z \rightarrow \mu\mu$	9.5	11.0	9.5	**12.0**	8.2
64	Single e	6.8	7.7	6.9	**12.6**	6.6
25	Phase-2 TTBar	7.6	8.4	7.0	8.8	**12.1**

Partial bottomline:

- The profile generated with TTBar & Run 3 detector is enough to optimize for all processes
  - Even using Run 3 detector profiles for the Phase-2 detector workflow!
- **PGO (TTBar & Run 2 detector) + LTO are worth 10% of the simulation runtime**

Offline and Online Reconstruction

High Level Trigger and Offline Reconstruction

- HLT and Offline Reconstruction: same code base
  - Different parameters, local VS global, some algorithms used exclusively in one of the two flavours (e.g. production of pixel-only tracks)
- Very different set of modules for Run 3 and Phase-2 processing
  - Reflect the upgrade of the detector (E.g. HGCal, MTD subdetectors)
  - Not the case for the simulation of the passage of particles through matter
- Runtime performance of reconstruction is a priority for CMS
  - Key aspect for a successful Run 3 and Phase-2 upgrade
  - Aspirational -10%/y, achieved (and then some) in the last 3 years

Building the "Reco and HLT Matrix"

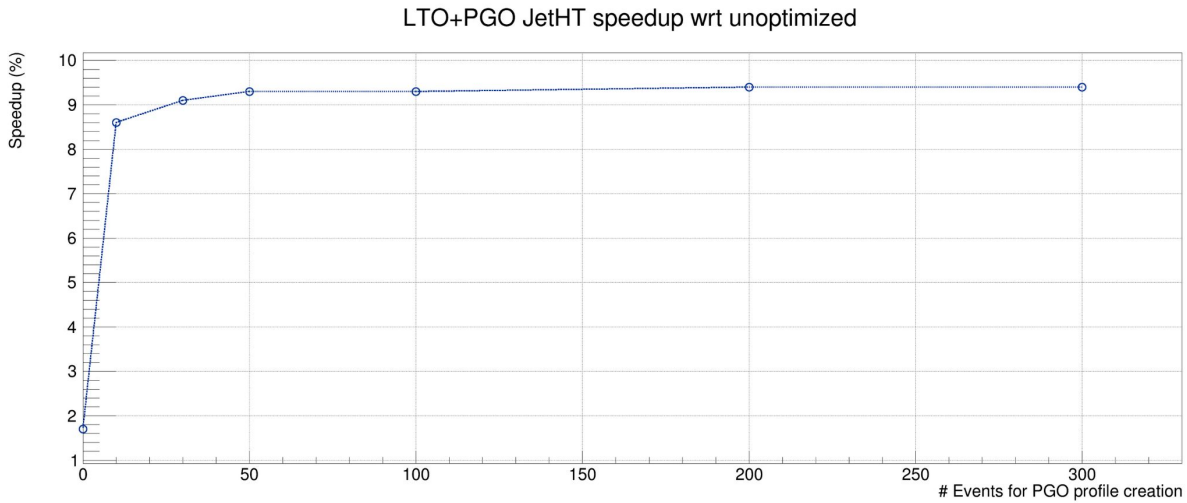
- Approach inspired from simulation: select primary datasets (events selected by the same set of triggers) instead of generator processes
- Runtime reduction
- LTO: 1.7% Reco and 2.7% HLT - PGO+LTO: 9.4% Reco and 10.5% HLT
  - PGO alone: ~7% in both cases
- As for simulation, one flavour of events seems adequate to produce profile for all of them
  - Somewhat smaller speedup when crossing online/offline and Run 3/Phase-2 boundaries (expected)

Events	**Running →**	**200**	**200**	**200**	**200**	**100**	**4313**
Profiling ↓	PD's	JetHT	MET	Single μ	ZeroBias	Phase-2 TTbar	HLT
70	JetHT	**9.4**	**9.4**	8.9	8.8	6.4	4.2
70	MET	9.2	**9.3**	7.6	7.7	6.1	3.9
70	Single μ	**9.3**	8.3	8.1	7.5	5.2	4.1
70	ZeroBias	**9.3**	**9.3**	8.1	8.1	5.7	2.3
30	Phase-2 TTBar	2.1	2.0	2.1	1.9	**10.9**	-
300	HLT	-	-	-	-	-	**10.5**

Simplified processes description - JetHT: high Pt events, MET: high MET events, Phase-2: TTBar 200 PU, HLT: standard set of unbiased events used to profile online reconstruction

Profile creation and measurement after LTO+PGO

- The “saturation” of the PGO optimization level was studied
  - A few tens of events are enough to reach max speedup
- Complete analysis of performance counters performed with *perf*
- Less instructions loaded, less *misses



Perf Counter	Unoptimized	PGO+LTO	Variation %
iTLB-loads	17620921006	5568894549	-68.4
dTLB-load-misses	527917138	344237835	-34.8
node-stores	30976346	26027365	-16.0
iTLB-load-misses	381219478	322504541	-15.4
dTLB-store-misses	24175552	20500680	-15.2
cache-misses	1368672746	1223379963	-10.6
LLC-load-misses	472081806	423969272	-10.2
LLC-load-misses	469760289	424200758	-9.7
branch-loads	1524479528697	1408692929959	-7.6
dTLB-stores	1534101493842	1420429100439	-7.4
L1-dcache-stores	1534054425994	1421056802376	-7.4
dTLB-loads	3488599263973	3241507461473	-7.1
L1-dcache-loads	3487189366873	3240380640099	-7.1
L1-dcache-loads	3488546809127	3243968339735	-7.0
instructions	9884281659440	9248376466655	-6.4
L1-icache-load-misses	347871102911	329468597171	-5.3

Putting It All Together

Merging Profiles

- Profiling time:
  - Run 3 Sim, Reco, HLT 1h:30m
  - Phase-2 Sim, Reco: 3h:30m
- Straightforward: *gcov-tool* worked *out of the box* to merge 5 profiles directories into one
- Use that directory to instruct GCC to rebuild CMS specific code as well as Geant 4
  - **A solid 10% speedup gained across the board**

Proc. Step	Simulation					Reconstruction					
Process	TTBar	MinBias	$Z \rightarrow \mu\mu$	Single e	Phase-2 TTBar	JetHT	MET	Single μ	ZeroBias	Phase-2 TTBar	HLT
Speedup	10.5	10.3	10.6	15.9	11.9	9.1	8.9	9.0	8.6	11.3	11.3

Status of LTO and PGO in Production

- LTO: applied to all CMS libraries and Geant 4
  - Campaign ongoing to fix all warnings
  - Will be validated in terms of physics results (nothing should change, but need to check)
  - Target: CMSSW release cycle 13_X, intended for 2023 data taking
- PGO: a substantial change in the build process. Working on an optimal deployment strategy considering
  - Limitations of the building infrastructure
  - Frequency of profile production: 2x day, 1x week?
  - Prioritization: simulation or reconstruction? Phase-1 or Phase-2?
  - Developer experience: tooling to transparently trigger PGO for everybody? Only for central builds?
  - 8% speedup on top of LTO is worth some thinking!

Conclusions

Conclusions

- Technical improvements, such as advanced compiler flags and optimizations, are a powerful tool to speed up HEP code
- CMS investigated Link Time Optimization and Profile Guided Optimization
  - For simulation, reconstruction (Run 2 and Phase-2) and HLT (Run 2)
- LTO: 2-3% speedup for sim, reco and HLT, just adding flags
- PGO: additional ~8% speedup for all processing types studied
  - Five workflows seem to be enough to produce profiles to optimize all workflows
  - A few tens of events are enough to produce complete profiles
  - Merging profiles is straightforward
- Processing throughput increase reflected by low level quantities measured via performance counters
  - E.g. # cache misses, # instruction load/stores
- LTO and PGO are experiment independent optimizations (just requires a good compiler)