CMS software performance studies

Matti Kortelainen

Helsinki Institute of Physics

with P. Elmer, G. Eulisse, V. Innocente, C. Jones and L. Tuura (CMS Collaboration)

First Thematic CERN School of Computing
Split, Croatia

June 7, 2013

Adapted from slides presented in CHEP 2010

Introduction

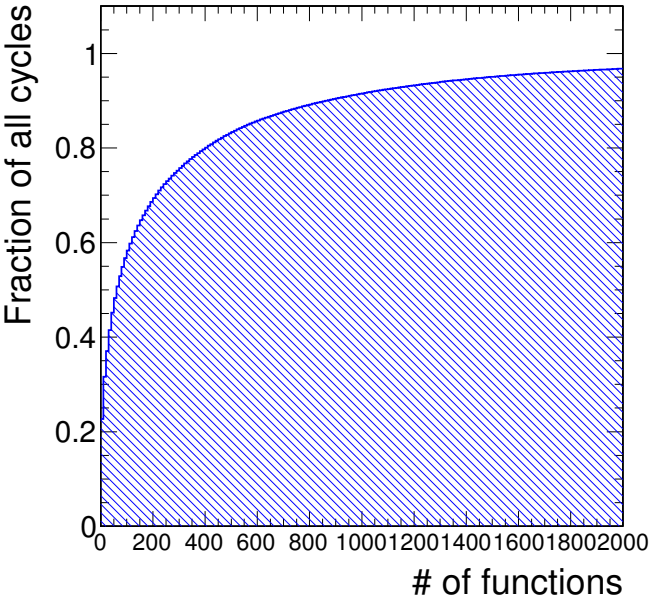
- CMS software (CMSSW)
  - 2.5M lines of C++, 600k lines of python
  - General-purpose event processing framework
    - ⋆ Same codebase for simulation, high-level trigger, reconstruction, analysis
  - ∼ 500–1000 shared libraries (depending on workflow)
  - Algorithms and data formats separated, “event” used as a data store
- Framework in production today is single-threaded
  - Support for forking and copy-on-write added years ago
  - Full multithreading being added (TBB), release expected on autumn
- Performance is monitored both continously, and as a special effort

Tools

- Main workhorse is IgProf (http://igprof.sourceforge.net)
  - Simple CPU/memory profiler developed in CMS
- Simple timers
  - Both `/usr/bin/time` and instrumented
- `perfmon` / `perf`
- Intel Performance Tuning Utility (PTU)
- GOoDA (http://code.google.com/p/gooda)
  - Uses `perf`, predefined set of PMU events, reports viewable in web browser
  - Can show event counts per source line / asm instruction / basic block (but not very precisely)
- Not just a matter of measurements, but also how to share the results to your collaborators
  - We have paid a lot of attention to web-based reports

Some challenges

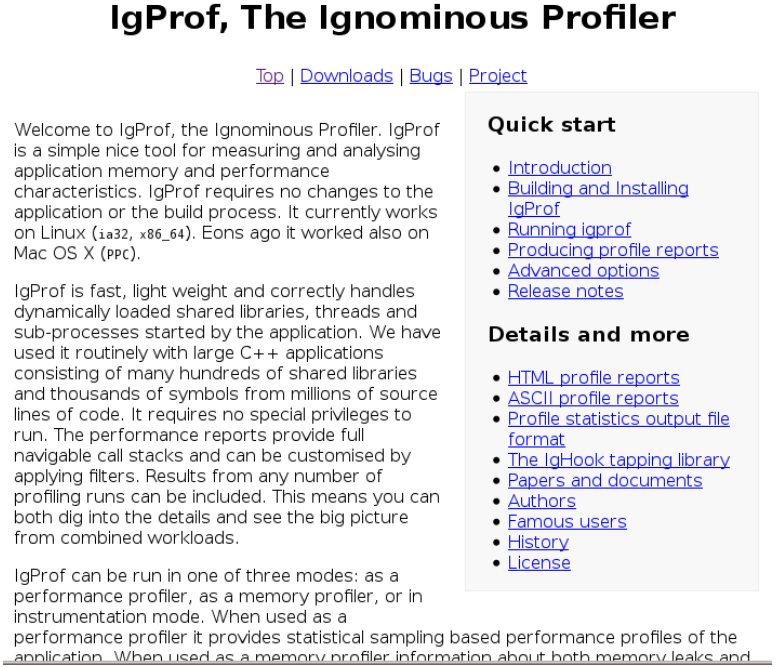
- Diminishing returns



  - Many, many functions
  - Significant improvements need re-engineering
- Need $\sim 2\times$ improvement after LS1 to keep same physics performance
  - Fortunately, this includes algorithmic improvements

IgProf

- A simple tool for measuring
  - ⋆ Sampling profiles
  - ⋆ Memory allocations
  - ⋆ Memory leaks
- Works in Linux (both 32 and 64 bit), no recompilation needed
- Handles shared libraries, threads, subprocesses
- Freely available at SourceForge
- Web-based navigator for easy browsing and sharing of the reports



IgProf, The Ignominous Profiler

Top | Downloads | Bugs | Project

Welcome to IgProf, the Ignominous Profiler. IgProf is a simple nice tool for measuring and analysing application memory and performance characteristics. IgProf requires no changes to the application or the build process. It currently works on Linux (`ia32`, `x86_64`). Eons ago it worked also on Mac OS X (`PPC`).

IgProf is fast, light weight and correctly handles dynamically loaded shared libraries, threads and sub-processes started by the application. We have used it routinely with large C++ applications consisting of many hundreds of shared libraries and thousands of symbols from millions of source lines of code. It requires no special privileges to run. The performance reports provide full navigable call stacks and can be customised by applying filters. Results from any number of profiling runs can be included. This means you can both dig into the details and see the big picture from combined workloads.

IgProf can be run in one of three modes: as a performance profiler, as a memory profiler, or in instrumentation mode. When used as a performance profiler it provides statistical sampling based performance profiles of the application. When used as a memory profiler information about both memory leaks and

Quick start

- Introduction
- Building and Installing IgProf
- Running igprof
- Producing profile reports
- Advanced options
- Release notes

Details and more

- HTML profile reports
- ASCII profile reports
- Profile statistics output file format
- The IgHook tapping library
- Papers and documents
- Authors
- Famous users
- History
- License

IgProf: performance profile

Seconds

Rank	% total	Counts: to / from this	Counts: Total	Paths: Including child / parent	Paths: Total	Symbol name
	74.07	213.90	213.91	2	2	edm::WorkerT<edm::EDProducer>::implDoBegin(edm::Ev
[16]	**74.07**	**0.01**	**213.89**	2	2	**edm::EDProducer::doEvent(edm::EventPrincipal&, edm**
	17.46	50.43	50.43	2	2	cms::CkfTrackCandidateMakerBase::produceBase(edm::
	11.99	34.63	34.63	2	2	ConversionTrackCandidateProducer::produce(edm::Eve
	4.96	14.33	14.33	2	2	MuonIdProducer::produce(edm::Event&, edm::EventSet
	4.76	13.74	13.74	2	2	GsfTrackProducer::produce(edm::Event&, edm::EventS
	4.66	13.47	13.47	2	2	TrackProducer::produce(edm::Event&, edm::EventSetu
	2.80	8.10	8.10	2	2	SeedGeneratorFromRegionHitsEDProducer::produce(edm
	1.65	4.76	4.76	2	2	SiStripRecHitConverter::produce(edm::Event&, edm::
	1.64	4.74	4.74	2	2	EcalUncalibRecHitProducer::produce(edm::Event&, ec
	1.64	4.74	4.74	2	2	PrimaryVertexProducer::produce(edm::Event&, edm::E
	1.38	3.98	3.98	2	2	GoodSeedProducer::produce(edm::Event&, edm::EventS
	1.37	3.95	3.95	2	2	CaloTowersCreator::produce(edm::Event&, edm::Event
	1.24	3.59	3.59	2	2	CosmicMuonProducer::produce(edm::Event&, edm::Ever
	0.92	2.65	2.65	2	2	PFElecTkProducer::produce(edm::Event&, edm::EventS
	0.91	2.62	2.62	2	2	PFBlockProducer::produce(edm::Event&, edm::EventSe
	0.80	2.30	2.30	2	2	SecondaryVertexProducer::produce(edm::Event&, edm:

IgProf: memory allocations

Bytes # allocs

Rank	% total	Counts: to / from this	Counts: Total	Calls: to / from this	Calls: Total	Paths: Including child / parent	Paths: Total	Symbol name
	87.50	40,063,717,931	40,063,717,931	265,883,752	265,883,752	4	4	edm::WorkerT<edm::EI
[16]	**87.50**	**0**	**40,063,717,931**	**0**	**265,883,752**	**4**	**4**	**edm::EDProducer::doI**
	18.24	8,352,884,389	8,352,884,389	61,183,471	61,183,471	2	2	cms::CkfTrackCandida
	16.54	7,573,767,686	7,573,767,686	37,888,574	37,888,574	2	2	ConversionTrackCandi
	7.03	3,216,447,860	3,216,447,860	20,888,236	20,888,236	2	2	GsfTrackProducer::pr
	6.33	2,899,892,529	2,899,892,529	2,910,740	2,910,740	2	2	SiStripRecHitConver
	6.22	2,848,508,108	2,848,508,108	11,970,216	11,970,216	2	2	SeedGeneratorFromRe
	4.05	1,852,820,570	1,852,820,570	7,827,912	7,827,912	2	2	TrackProducer::produ
	2.89	1,321,908,244	1,321,908,244	6,932,668	6,932,668	2	2	PrimaryVertexProduce
	2.69	1,231,376,825	1,231,376,825	8,419,371	8,419,371	2	2	GoodSeedProducer::pr
	1.64	752,137,339	752,137,339	10,328,177	10,328,177	2	2	MuonIdProducer::proc
	1.57	720,185,219	720,185,219	11,214,313	11,214,313	2	2	JetPlusTrackProducer
	1.36	621,227,344	621,227,344	7,594,459	7,594,459	2	2	CaloTowersCreator::p
	1.35	616,394,680	616,394,680	2,792,145	2,792,145	2	2	PFElecTkProducer::pr
	1.16	531,244,964	531,244,964	4,591,777	4,591,777	2	2	SecondaryVertexProdu
	1.11	507,228,642	507,228,642	6,084,587	6,084,587	2	2	PFRecHitProducer::pr
	1.00	457,500,032	457,500,032	3,623,841	3,623,841	2	2	VirtualJetProducer:
	0.97	444,230,330	444,230,330	4,036,259	4,036,259	2	2	reco::modules::Analy
	0.95	436,926,779	436,926,779	3,280,413	3,280,413	2	2	PFDisplacedVertexPro
	0.92	419,294,139	419,294,139	2,072,230	2,072,230	2	2	PixelTrackProducer:

Some observations

- In the past, 20 % of CPU time wasted in memory (de)allocation
- Some common causes for memory churn
  - Confusion how `std::vector` works (excessive copying)
  - Dynamic memory allocation in tight loops, numerous tiny objects
  - Multiple in-memory copies, strings used in inappropriate places
- Benefitted from compiler updates, from transition to 64 bit
- Vectorization
  - Autovectorization
  - Explicitly implemented, in some utilities and algorithms (geometrical vectors and rotations)
  - Also via abstractions
    - ⋆ E.g. VDT, developed in house for fast and approximate transcendental functions (https://svnweb.cern.ch/trac/vdt)

Some observations from PMU events

- Identified functions with bad CPI, or high number of div/sqrt
- About half of cycles wasted in front-end decoder stalls
  - I.e. CPU is starved from instructions
  - Known for some time, exact reasons still not known
  - Some known causes are
    - ⋆ Bad branch prediction performance
    - ⋆ High sensitivity to instruction cache misses
  - Possible sources include
    - ⋆ Code size and locality, pointer chasing (incl. virtual functions)
- None of these can be seen in sampling profiles!
  - PMU events allow to try to see what is really going on