The evolution of CMS software performance studies

Matti Kortelainen
Helsinki Institute of Physics

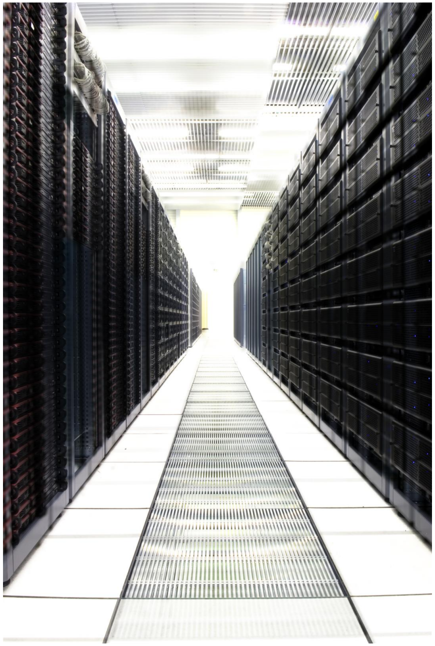
with P. Elmer, G. Eulisse, V. Innocente, C. Jones and L. Tuura
on behalf of the CMS collaboration

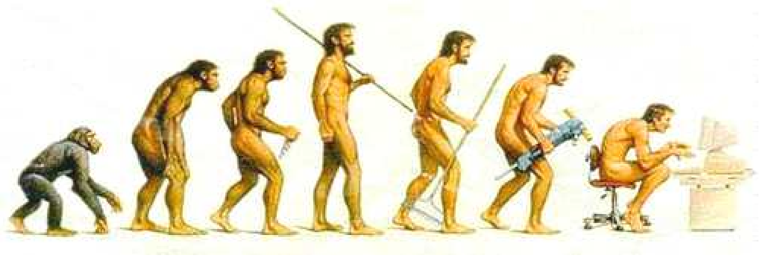
International Conference on Computing in High Energy and Nuclear Physics 2010, Taipei, Taiwan

October 19, 2010

Introduction

- Improving software performance and efficiency gives clear benefits
  - Less resources required
  - Less time needed
- The program performance must be measured
- What can be measured?
  - CPU time (per processed event)
  - Memory allocations and footprint
  - CPU/wall clock time ratios
  - I/O rates and patterns
  - Event data sizes, transactions to databases, application startup times, software compilation times, etc.



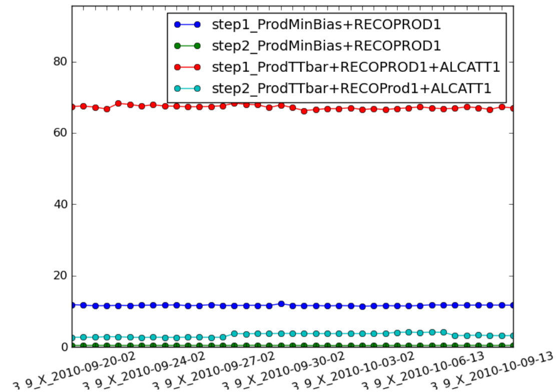


History

What has been done
Lessons learnt

Evolution of the optimization model

- From “Performance Task Force” ...
  - Started with cleanup of basic C++ errors
    - ⋆ Reducing dynamic memory churn
    - ⋆ Unnecessary copies and temporaries
- ... to work done routinely and systematically as a part of release integration, testing and planning
  - Various metrics (CPU time, memory footprint etc) produced automatically during integration builds
  - Results are available on the web



CPU time/event for series of integration builds

Observations: dynamic memory management & code size

- In the past, 20 % of CPU time was spent in memory (de)allocation
- Some common causes for the dynamic memory churn
  - Confusion how `std::vector` works, copying of large structures
  - Dynamic memory allocation in tight loops, numerous tiny objects
  - Multiple in-memory copies, strings used in inappropriate places
- CMS applications have from about 500 to over 1000 shared libraries
  - Now testing single big binaries, with shared libraries only for externals

Observations: dynamic memory management & code size

- In the past, 20 % of CPU time was spent in memory (de)allocation
- Some common causes for the dynamic memory churn
  - Confusion how `std::vector` works, copying of large structures
  - Dynamic memory allocation in tight loops, numerous tiny objects
  - Multiple in-memory copies, strings used in inappropriate places
- CMS applications have from about 500 to over 1000 shared libraries
  - Now testing single big binaries, with shared libraries only for externals

```
std::vector<float> rescaledNoises() {
  std::vector<float> noises = getNoises();
  std::vector<float> gains = getGains();
  std::transform(noises.begin(), noises.end(), gains.begin(),
                 noises.begin(), std::divides<float>());
  return noises;
}
```

Example of `std::vector` copying

Observations: external issues

- Transition to 64 bit
  - More and larger registers, reduced function call overhead, etc.
  - Transition from x87 to SSE floating point simultaneously
  - 5–20 % improvements in CPU time seen
  - Memory footprint increases by 25–30 % (in RSS)
- GCC compiler version updates
  - 4.3.4 used in production, 4.5 testing is starting
- Vectorization
  - As supported by compiler (`tree-vectorize` in GCC)
  - Directly implemented in some CMSSW utilities algorithms such as geometrical vectors and rotations
- Starting performance measurements

CPU time/event by release

Reconstruction of TTbar MC events (64 bit)						
Release	Date	Time/event	# of allocs		Alloc rate	
3_5_X	2010-02-06	4.1 s	334 k	53.9 GB	821 kHz	126 MB/s
3_6_X	2010-04-16	3.4 s	314 k	53.5 GB	934 kHz	152 MB/s
3_7_X	2010-05-27	3.2 s	293 k	47.3 GB	914 kHz	140 MB/s
3_8_X	2010-07-21	3.1 s	284 k	42.6 GB	920 kHz	141 MB/s

- CPU time/event has decreased despite continued development in reconstruction algorithms
- Number of memory allocations has decreased (rate has not)
- About 8 % of CPU time still wasted in malloc/free/etc
- The comparison is not entirely fair, also other optimizations have been going on simultaneously
- Measured on 2.33 GHz Intel Xeon E5410 (Harpertown), 16 GB memory

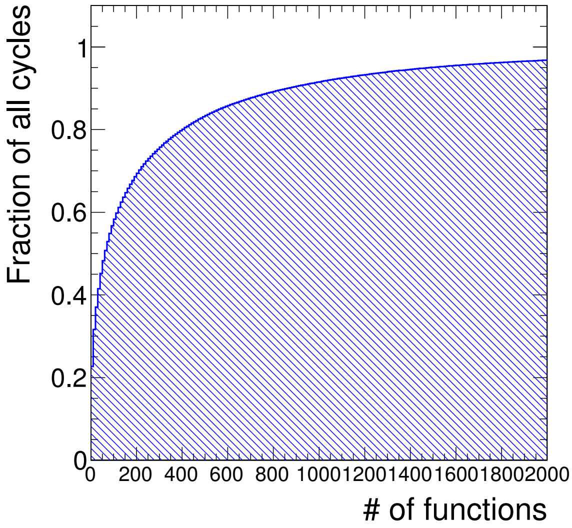
Present state

Where we are now

Status

We know what to do for the dynamic memory allocations, and we know how to use the tools like Valgrind and IgProf to detect and measure them.

Most of the hot spots have been fixed in the code. However, this makes it challenging to get a noticeable total impact with only small, localized optimizations. A consequence is that more work is required to get significant improvements.



IgProf

- A simple tool for measuring
  - Sampling profiles
  - Memory allocations
  - Memory leaks
- Works in Linux (both 32 and 64 bit), no recompilation needed
- Freely available at Source-Forge
- 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).

Few profilers are capable of correctly profiling CMS' C++ software. IgProf is a fast, light weight and correctly handles dynamically loaded shared libraries, threads and sub-processes started by the application. It requires no special privileges to run. The performance reports can be customised by applying filters and may include results from any number of profiling runs. This means you can both dig into the details and see the big picture from combined workloads.

Quick start

- Introduction
- IgProf in CMSSW
- Running igprof
- Analysing results

Details and more

- Full details on analysis output
- Profile output file format
- The IgHook tapping library
- Papers and documents
- Authors
- History

http://igprof.sourceforge.net/

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&, ed
	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:

http://cms-service-sdtweb.web.cern.ch/cms-service-sdtweb/igperf/vocms81/slc5_amd64_gcc434/380p4/navigator/recottbar02_perf/16

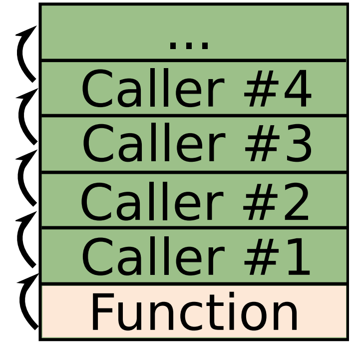
IgProf: memory allocations

Rank	% total	Counts (Bytes): to / from this	Counts (Bytes): Total	Calls (# allocs): to / from this	Calls (# allocs): 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::ED
[16]	**87.50**	**0**	**40,063,717,931**	**0**	**265,883,752**	**4**	**4**	**edm::EDProducer::do**
	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	ConversionTrackCand
	7.03	3,216,447,860	3,216,447,860	20,888,236	20,888,236	2	2	GsfTrackProducer::p
	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::p
	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::
	1.35	616,394,680	616,394,680	2,792,145	2,792,145	2	2	PFElecTkProducer::p
	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::p
	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:

http://cms-service-sdtweb.web.cern.ch/cms-service-sdtweb/igperf/vocms81/slc5_amd64_gcc434/380p4/navigator/recottbar03_total/16

IgProf: status update

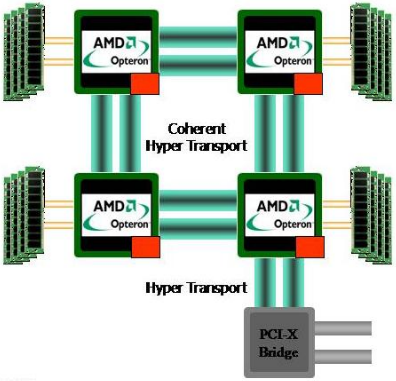
- IgProf now supports 64-bit Linux systems
- Originally ∼10 % of profile hits lost due to stack unwinding inaccuracies
- GCC issues
  - Only versions 4.5.0 and later generate sufficiently accurate unwind info
  - Need to rebuild as much as possible, including `libm`, with 4.5.0+
  - Issues tracing through global constructors (`_init`, `crt`) may still remain
- `libunwind` issues
  - Latest git version needed, includes several of our critical fixes, especially for accuracy and reliability
  - Factor 5–6 performance improvement, not in git yet but criticial for memory profiling



Determine the function call chain by unwinding the call stack

Profiling with CPU performance events

CPUs and memory architectures have become more and more complex. Although sampling profilers can tell where the problems are, they don't tell too much about what the actual problem is.



Modern CPUs have a Performance Monitoring Unit (PMU) which has counters for various performance events, like for

- Retired instructions
- Unhalted cycles ("all" cycles), stalled cycles
- Cache hits and misses, local/remote memory access (NUMA)

Tools: Perfmon2 (see talk by D. Kruse in this session) and Intel Performance Tuning Utility (PTU)

Intel Performance Tuning Utility

- Commercial product from Intel
- Profiling by the performance events
- Plugin to Eclipse, kernel module for the PMU interaction
- Can show events per
  - process, module, source file, function
  - source line, basic block and even assembly line (limited precision)
- Has also more sophisticated analysis tools
  - Difference of two profiles
  - Cache line distributions
- Results can be studied in Eclipse, and exported to a spreadsheet

PTU: function list

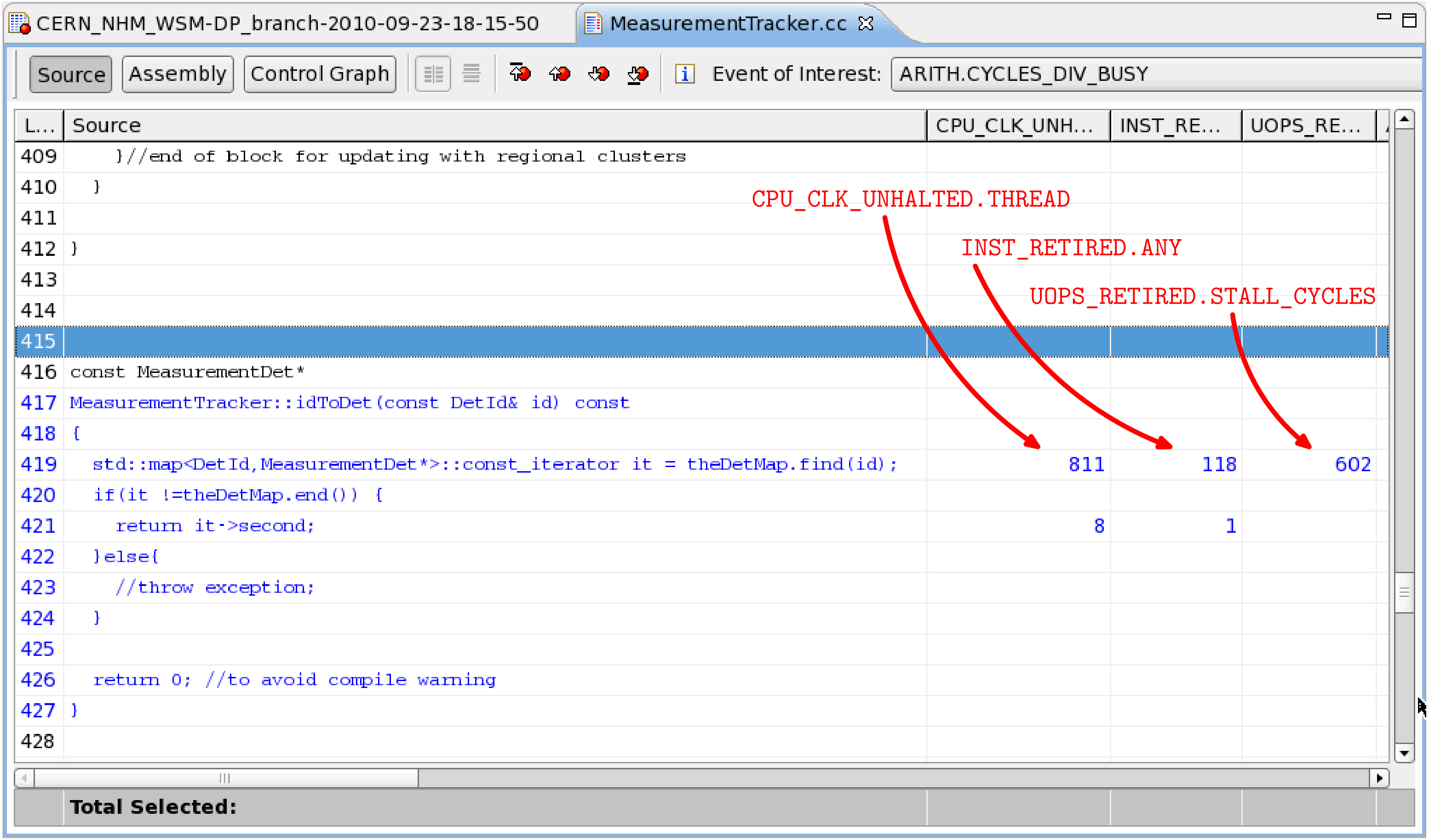
CERN_NHM_WSM-DP_branch-2010-09-23-18-15-50 | MeasurementTracker.cc

Function	CPU_CLK_UNHALTED.THREAD	INST_RETIRED.ANY	UOPS_RETIRED.STALL_CYCLES
KullbackLeiblerDistance<(unsigned int)...	1,194	2,986	210
JacobianLocalToCartesian::JacobianLoca...	1,183	1,342	714
Cint::G__CallFunc::Execute(void*)	1,164	1,657	518
LinearGridInterpolator3D::interpolate(d...	1,156	1,175	770
ROOT::Math::SMatrix<double, (unsigne...	1,134	2,063	252
sin	1,127	922	714
BasicSingleTrajectoryState::checkGloba...	1,098	879	602
SteppingHelixPropagator::refToMagVolu...	1,080	1,275	826
Similarity<double, 5u, 5u, ROOT::Math::...	1,062	1,883	322
G__get_ifunc_internal	1,050	1,489	266
__ieee754_atan2f	1,035	794	448
DetIdAssociator::fillSet(std::set<DetId, ...	972	789	784
cos	968	876	532
<unknown(s)>	885	1,282	280
ROOT::Math::SMatrix<double, (unsigne...	883	2,654	98
SiStripRecHitMatcher::project(GeomDet...	858	1,420	406
<unknown(s)>	846	864	364
ROOT::Math::SMatrix<double, (unsigne...	845	650	434
Cint::G__MethodInfo::IsValid(void)	828	1,023	84
std::string::find(char const*, unsigned l...	820	1,302	392
MeasurementTracker::idToDet(DetId con...	819	119	602
SteppingHelixPropagator::propagate(St...	818	908	588

CPI ~6.8

Limit 95% | Gra...ity Functio | Process All | Thread All | Module All | Cpu Total

PTU: source view



Lessons from PTU

- The actually executed code can be seen from the retired instructions and the retired cycles events
- Found functions with
  - bad Cycles/Instruction (CPI) ratio
  - high number of divisions and square roots (ARITH.CYCLES_DIV_BUSY)
- Almost half of the cycles are spent in front end decoder stalls
  - In other words, CPU starved from instructions
  - Not analyzed, known causes include bad branch prediction performance and high sensitivity to icache misses (L1I, L2, ITLB)
  - Possible sources include code size and locality[1], and function pointer chasing (incl. virtual functions)
  - Single big binaries expected to yield much more insight

None of these can be seen in sampling profiles!

[1] L. Tuura, V. Innocente, G. Eulisse, Analysing CMS software performance using IgProf, OProfile and callgrind, CHEP07

ptuview, a web display for PTU reports

- Analyzing PTU reports requires either Eclipse+PTU itself, or the reports can be exported to a spreadsheet
  - Neither is very handy for sharing the information in a large collaboration
  - Started to write a tool for displaying the reports on the web
- `ptuview` is a CGI script written in Python
  - JavaScript used for UI improvements
- Uses the spreadsheet files exported from PTU as the data store
- Lists events per function, also source and assembly views
- Performance events organized as a tree to intuitively guide through the most important events
- Freely available at `http://mkortela.web.cern.ch/mkortela/ptuview/`
- Development effort continues

ptuview: tree diagram

Hotspot view - function

View all events

◀ All program cycles	4.8e+11			
	◀ Executing	53.18 %		
		▶ Port utilization		
		Microcode sequencer	0.56 %	
		Wasted work (of uops)	18.29 %	
		Mispredicted branches (of all branches)	2.79 %	
		Indirect branches (of all branches)	13.46 %	
	◀ Stalled	46.82 %		
		▶ Memory latency total		
		Cycles the divider is busy	7.76 %	
		▶ Load ordering stalls		
		▶ Bandwidth		
		◀ Front End total		
			Decoder stalls	47.57 %
			▶ Backend instruction starvation	N/A

http://mkortela.web.cern.ch/mkortela/cgi-bin/demo/ptuview/cern_wsm/hot/function

ptuview: function list

- All program cycles 4.8e+11
  - Executing 53.18 %
  - Stalled 46.82 %
    - Memory latency total
    - Cycles the divider is busy 7.76 %
    - Load ordering stalls
    - Bandwidth
    - Front End total

Tooltip for the formula

((ARITH.CYCLES_DIV_BUSY / CPU_CLK_UNHALTED.THREAD) * 100.0)

CPU_CLK_UNHALTED.THREAD	Executing	UOPS_RETIRED.STALL_CYCLES		ARITH.CYCLES_DIV_BUSY		Function
3.4e+10	63.30 %	1.3e+10	36.70 %	0	0.00 %	deflate_slow
2.6e+10	49.93 %	1.3e+10	50.07 %	2.8e+07	0.11 %	_int_malloc
2.2e+10	67.41 %	7.3e+09	32.59 %	0	0.00 %	CLHEP::operator*(CLHEP::HepMatrix const& ...
1.9e+10	61.94 %	7.4e+09	38.06 %	0	0.00 %	CLHEP::operator*(CLHEP::HepSymMatrix con ...
1.3e+10	59.20 %	5.3e+09	40.80 %	0	0.00 %	_int_free
1.2e+10	75.44 %	3e+09	24.56 %	0	0.00 %	CLHEP::operator*(CLHEP::HepSymMatrix con ...
1.1e+10	53.44 %	5.3e+09	46.56 %	6.2e+08	5.39 %	__ieee754_exp
1e+10	40.65 %	6e+09	59.35 %	1.2e+09	12.15 %	__ieee754_log
1e+10	43.43 %	5.7e+09	56.57 %	2e+09	19.88 %	__ieee754_atan2
9.2e+09	21.82 %	7.2e+09	78.18 %	8.4e+07	0.91 %	do_lookup_x
8.5e+09	77.88 %	1.9e+09	22.12 %	2.8e+07	0.33 %	CLHEP::HepSymMatrix::num_row(void) const
8.2e+09	60.40 %	3.2e+09	39.60 %	0	0.00 %	PyEval_EvalFrameEx
6.7e+09	28.76 %	4.8e+09	71.24 %	4.4e+09	65.79 %	magfieldparam::TkBfield::Bcyl(double, do ...
5.8e+09	53.03 %	2.7e+09	46.97 %	2.8e+07	0.48 %	malloc
5.6e+09	10.08 %	5.1e+09	89.92 %	2.9e+09	50.67 %	SteppingHelixPropagator::makeAtomStep(St ...
5.4e+09	35.82 %	3.4e+09	64.18 %	7.6e+08	14.09 %	atanf

Sorting

Links to source views

http://mkortela.web.cern.ch/mkortela/cgi-bin/demo/ptuview/cern_wsm/hot/function

ptuview: source view

.THREAD	Executing	UOPS_RETIRED.STALL_CYCLES		Line	
0	N/A	0	N/A	412	}
0	N/A	0	N/A	413	
0	N/A	0	N/A	414	
0	N/A	0	N/A	415	
0	N/A	0	N/A	416	const MeasurementDet*
0	N/A	0	N/A	417	MeasurementTracker::idToDet(const DetId& id) const
0	N/A	0	N/A	418	{
1.6e+09	25.77 %	1.2e+09	74.23 %	419	std::map<DetId,MeasurementDet*>::const_iterator it = theDetMap.find(id);
0	N/A	0	N/A	420	if(it !=theDetMap.end()) {
1.6e+07	100.00 %	0	0.00 %	421	return it->second;
0	N/A	0	N/A	422	}else{
0	N/A	0	N/A	423	//throw exception;
0	N/A	0	N/A	424	}
0	N/A	0	N/A	425	
0	N/A	0	N/A	426	return 0; //to avoid compile warning
0	N/A	0	N/A	427	}
0	N/A	0	N/A	428	

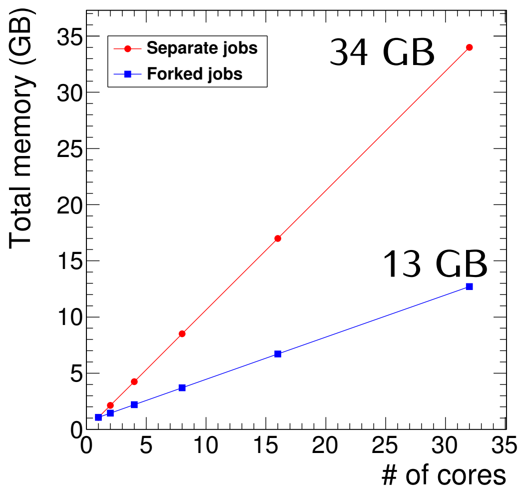
CPU_CLK_UNHALTED.THREAD

Links to assembly view

http://mkortela.web.cern.ch/mkortela/cgi-bin/demo/ptuview/cern_wsm/src/67#414

Multicore

- Although we could benefit from multicore machines by running blindly a process per core, we would also need more memory (O(1 GB)/process)
- Most of the memory is only read and could be shared between the processes (program code, conditions, geometry, etc)
- For example, by forking and relying on copy-on-write by the operating system we can save substantial amount of memory
- See talk by C. Jones in parallel session 18 (today morning) for more information



Measurement with 64 bit software on 4 CPU, 8 core/CPU 2 GHz AMD Opteron

Conclusions

- Long experience from a dedicated effort for performance optimization
- Moved from a “task force” to a routine work done as a part of the release integration and testing
- Continued to improve the CMSSW reconstruction performance
- Actively and systematically with better tools
  - IgProf has been updated to 64 bit
  - We are in transition to use tools with CPU performance events
    - Perfmon2
    - Intel Performance Tuning Utility (PTU)
  - A web display (`ptuview`) being developed to ease the sharing of the performance reports in a large collaboration
- Future plans
  - Continue the development of multicore aware CMSSW
  - Continue to learn the new tools and to interpret their output