Practical experience with Performance Monitors on Xeon and Itanium

Monthly Technical Review Meeting

24th October

Ryszard Jurga

Agenda

- Introduction to the CERN computing
- Introduction to Performance Monitoring
- Performance Monitors
  - events
  - countes
  - interfaces
  - tools
  - some results
- Profiling and some results
- Conclusions

Reminder


Performance Tools and PAPI
Philip J. Mucci
Center for Parallel Computers (PDC),
Royal Institute of Technology (KTH), Stockholm, Sweden
Innovative Computing Laboratory, UT Knoxville
mucci at cs.utk.edu
http://www.cs.utk.edu/~mucci
October 3rd, 2005

Computing@CERN

- High Throughput Computing (HTC)
  - lxbatch
  - x86 (Xeon), 32bit →64bit
- High Performance Computing (HPC)
  - opencluster
    - Computational Fluid Dynamics (CFD)
  - Itanium, 64bit
- multi processor/core boxes
- many simultaneously running jobs
  - interaction, interference
  - CPU/memory resources sharing

The need of monitoring

- The Large Hadron Collider computing requirements
  - 1000s per 1 full event (on CPU with 1000 SpecINT2000)
  - up to ~70k-100k CPUs
- Optimization
  - performance measurements
  - bottleneck identification
  - bottleneck analysis
  - reimplementation and redesign

"Bottlenecks occur in surprising places, so don't try to second guess and put in a speed hack until you have proven that's where the bottleneck is." Rob Pike

Principal Google Engineer

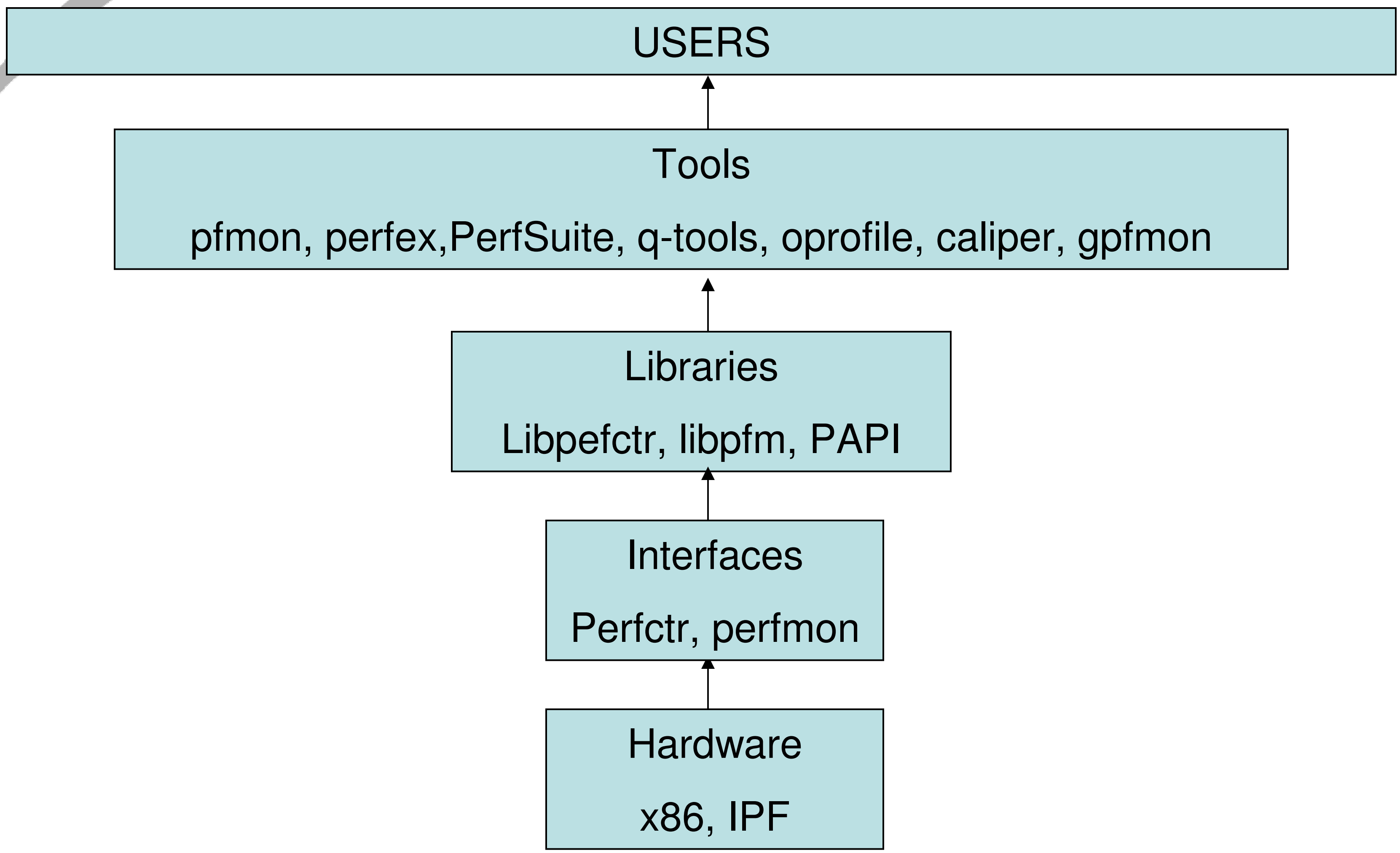
Performance Monitoring

- Goal
  - analysis of the behavior of a program while running
    - e.g. execution time, time spent per function, call graph
- Software Instrumentation
  - code snippets to collect required data
    - source
      - manual,
      - compiler assisted (gcc –pg, gprof)
    - binary
      - offline - binary translation (ATOM)
      - online - adds the code dynamically while running (PIN, DynInst)
      - high overhead
  - portable on the same platform family
  - half answer: show problems, where cycles are spent
  - time domain

Performance Monitoring HW

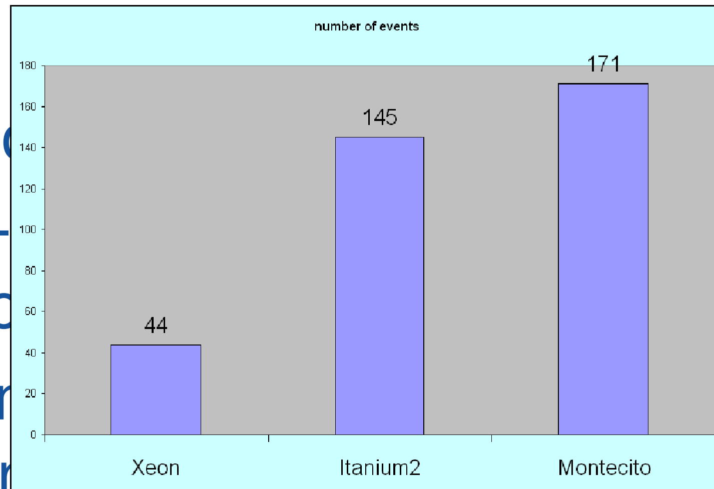
- Hardware approach
  - ✓ more detailed answer: show problems and their sources
  - ✓ special on-chip hardware of modern CPU: Performance Monitors
  - ✓ different domains: cpu cycles=time, instructions, cache misses etc.
  - ✓ less overhead
  - ✗ less portability
- Hybrid solution
  - both: instrumentation and hardware solutions
    - e.g. The Tuning and Analysis Utilities (TAU)

Performance Monitoring at CERN



Performance Monitor - events

- Ideas
  - count all … performance of applicati…
  - events – … of executed instructio… , cache misses
  - ✘ flat infor… al number of cycles a… ssor is halted
  - ✓ break down on Itanium – e.g. you can go from stalls into their sources



Counters

- 40bit (Xeon), 48bit(Itanium2, Montecito)
- 18 (Xeon), 4 (Itanium2), 12 (Montecito)
  - ✗ not enough
  - 4 groups of counters (Xeon)
    - up to 6 counters per one group
    - ✗ Only up 2 counters/group can run independently
    - ✗ counters are assigned to specific events
    - ✗ at-retirement events require up to 2 counters (Xeon and tagging)
  - ✓ counters are freely available (IPF)
  - ✗ certain events can not be measured together

- other features:
  - enable the cascading of paired counters (Xeon)

Performance Monitors – other features

- capture event information and instruction pointer – useful in profiling
  - Precise Event-Based Sampling (PEBS) on Xeon
  - Event Address Registers (EARs) on IPF
- tracing branches – determine the path taken to reach a particular code location – useful in a call graph approach
  - Branch Trace Store (BTS) Xeon
  - Branch Trace Buffer (BTB) IPF
- Instruction Address Range Matching (IPF)
  - counting within the IP range

Our requirements

- we look for universal and portable interface and tools
- support for
  - x86 - lxbatch
  - IPF - opencluster
- kernel 2.4 & 2.6
- user/kernel domain
- per thread/system-wide context
- multiplexing
- counting, sampling, profiling
- working with sources/binaries

Interfaces - perfctr

- perfctr
  - Intel x86, AMD K7/K8, Cyrix , VIA C3, WinChip, PowerPC,
  - ✗ no IPF
  - Lesser GPL
  - libperfctr.so library
  - ✓ kernel 2.4 & 2.6
  - ✗ no multiplexing
  - ✗ no documentation (apart from comments in source files)

Interfaces - perfmon

- perfmon
  - ✓ IPF, Pentium M/P6,Pentium 4/Xeon (32&64bit), Opteron 64bit, MIPS 5k/20k, Power5
  - GPL, MIT License
  - libpfm.so library
  - kernel 2.6
  - ✘ no kernel 2.4 support
  - ✓ multiplexing
  - ✘ library supports mainly for IPF
    - ✓ recently updated library support – e.g. Xeon
  - ✓ good documentation

Cross Platform Interface

- Performance Application Programming Interface (PAPI)
  - x86, IPF
    - perfctr & perfmon
  - Linux/Windows
  - all counter operations
  - multiplexing
  - user/kernel domain
  - counters are aggregated for the current process
    - ✘ not for any others in the system

Tools - pfmon

- http://perfmon2.sourceforge.net
  - ✓ basic counting
  - ✓ sampling
  - ✓ per thread/system-wide mode
  - ✓ user/kernel domain
  - ✗ 2.6 kernel
    - ✓ SLC4 is coming
  - ✗ only for IPF (perfmon)
    - ✓ more processors supported in a new version
  - ✗ no multiplexing
    - ✓ multiplexing support already available
      - ✗ not in a sampling mode
  - ✗ no profiling
    - ✓ profiling support in a new version
      - ✓ --smpl-module=inst-hist, --smpl-show-function
      - ✗ --resolve-addresses (shared libraries)
  - ✓ number of function calls (IPF only)
  - ✓ wrapping script ***i2prof.pl*** – lots of metrics

Tools - PerfSuite

- http://perfsuite.ncsa.uiuc.edu (psrun, psprocess ...)
  - ✓ basic counting
  - ✓ sampling
  - ✗ no system-wide mode
  - ✓ user/kernel domain
  - ✓ 2.4 & 2.6 kernel
  - ✓ X86 & IPF (perfctr, perfmon, PAPI)
  - ✓ more processors supported in a new version
  - ✓ profiling support
  - ✗ flat profile – neither number of function calls nor call graph

Tools - Caliper

- http://hp.com/go/caliper
  - ✓ counting/profiling
  - ✓ shared libraries
  - ✓ per process/system wide mode
  - ✓ overall value for all CPU
    - ✗ no break down into multi core/processors
  - ✗ no multiplexing
    - ✓ updated in new release
  - ✓ flat profile/number of function calls/call graph
  - ✓ gives guidance about improving the performance
  - ✗ only IPF

Tools – oprofile,q-tools

- oprofile.sourceforge.net
  - ✓ x86 & IPF, ...
  - system-wide profiler
    - ✓ shared libraries
  - kernel 2.4 & 2.6
  - requires root access
  - up to all available counters
    - ✗ no multiplexing
  - output
    - ✓ per library, per function, call graph
- http://www.hpl.hp.com/research/linux/q-tools
  - system-wide profiling (q-syscollect)
  - ✓ one/multi thread mode (qprof)
  - ✓ shared libraries
  - one hardware event
  - ✓ flat profile or call graph
  - ✗ only IPF

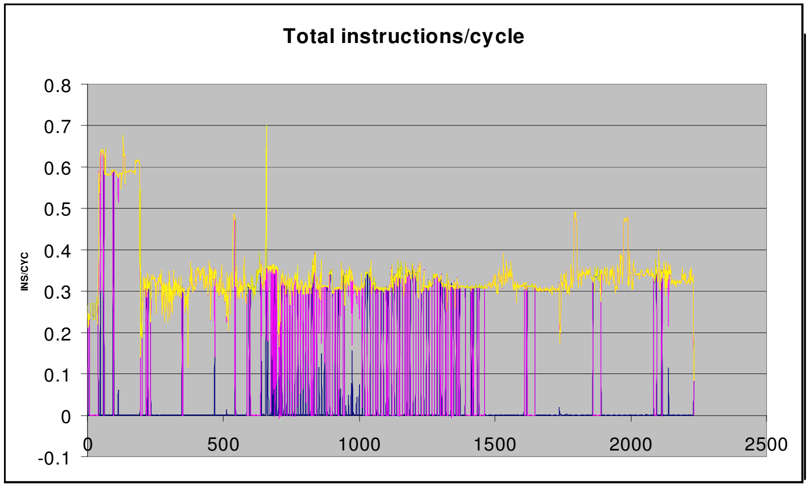
Experience with Xeon

- ✘ there is no tool which meets our requirements in the domain of system-wide monitoring (counting, sampling)
- we decided to develop our own tool
  - gpfmon
    - uses perfctr interface and library
    - user/kernel domain
    - per single or total CPU
    - ✓ enables multiplexing
      - 4 even sets
        - cpu cycles, instructions completed, branches taken predicted and mispredicted, L2 load and store missed, FP, scalar, load and stores instructions
      - we miss average 2% samples, apart from L2 store missed – 92%

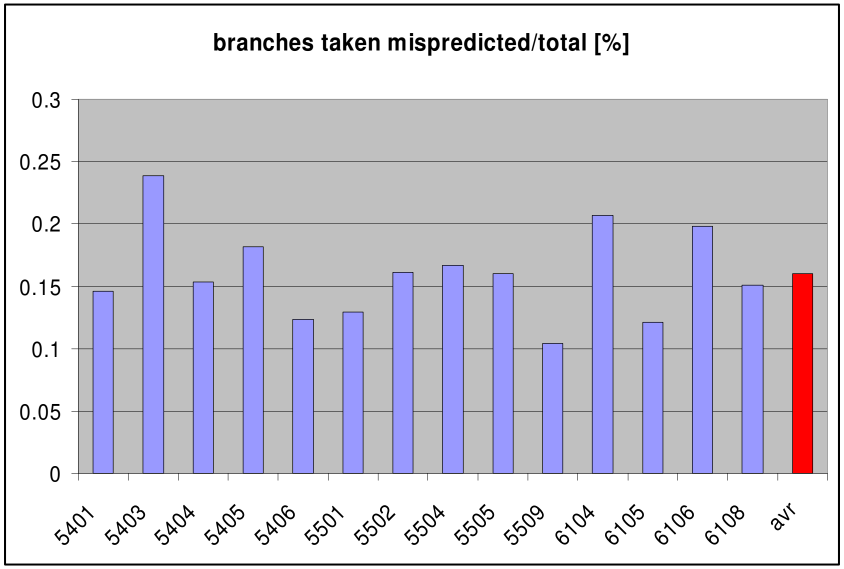
CYC	TOT	BR_TP	BR_TM	L2LM	L2SM
CYC	TOT	FP	LD	L2LM	L2SM
CYC	TOT	SDS	ST	L2LM	L2SM
CYC	TOT	LDST	BR	L2LM	L2SM

gpfmon sample results

- Geant4 Atlas simulation
  - IPC - 0.34
  - FP - 18%
  - LD+ST - 63% (7% LD caused L2 cache miss)
  - Branches - 10%, taken predicted/mispredicted=36

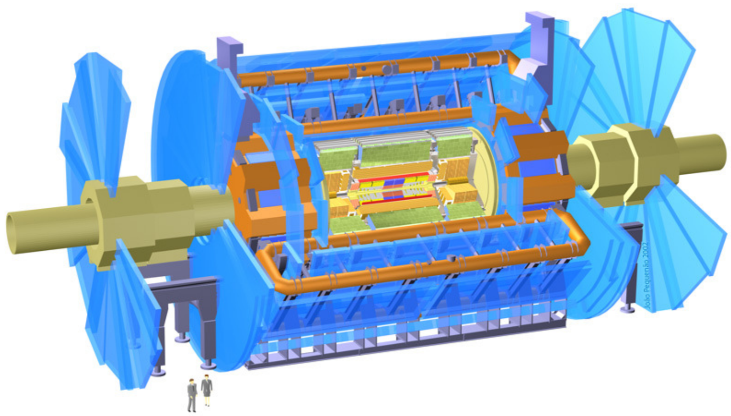


- Lxbatch (Averages)
  - IPC – 0.5
  - FP -14%
  - LD+ST – 52%
  - Branches – 10%, taken predicted/mispredicted=56



Profiling

- no access to sources of profiled applications
- from a single executable up to huge applications with more than 400 shared libs and profiling time up to 12 hours
  - Geant4 libraries and benchmarks (Xeon, Itanium)
  - Atlas and LHCb simulations
  - Atlas reconstruction



Profiling

- we use PerfSuite
  - Does not work with python scripts running from command line
  - unpredictable behavior on AFS (Andrew File System)
  - a problem with resoving function names
    - unknown functions with static libraries
    - a huge problem with shared libraries
      - in order to monitor, PerfSuite has to know all of them in advance
        - LD_PRELOAD variable – a big challange - how to select interesting libraries from 400+ without causing dependence error?
        - use oprofile to have another look

PerfSuite & LD_PRELOAD

```
Profile Information
==================================================
Class              : PAPI
Event              : PAPI_TOT_CYC (Total cycles)
Period             : 50000
Samples            : 719
Domain             : user
Run Time           : 17.52 (seconds)
Min Self %         : (all)
Module Summary
--------------------------------------------------------------------
Samples   Self %  Total %  Module
  376  52.29%  52.29%  /usr/bin/python
  178  24.76%  77.05%  /lib/ld-2.3.2.so
  159  22.11%  99.17%  /lib/tls/libc-2.3.2.so
    4   0.56%  99.72%  /lib/tls/libpthread-0.60.so
    1   0.14%  99.86%  /lib/libdl-2.3.2.so
    1   0.14% 100.00%  /lib/libutil-2.3.2.so
Function Summary
--------------------------------------------------------------------
Samples   Self %  Total %  Function
  376  52.29%  52.29%  ??
  110  15.30%  67.59%  do_lookup_versioned
   40   5.56%  73.16%  _int_malloc
   31   4.31%  77.47%  strcmp
   22   3.06%  80.53%  _dl_lookup_versioned_symbol
   19   2.64%  83.17%  memcpy
   16   2.23%  85.40%  __libc_malloc
   11   1.53%  86.93%  free
    7   0.97%  87.90%  _int_free
    7   0.97%  88.87%  strlen
    6   0.83%  89.71%  memset
    6   0.83%  90.54%  do_lookup
    5   0.70%  91.24%  malloc_consolidate
    5   0.70%  91.93%  __mempcpy
    4   0.56%  92.49%  __i686.get_pc_thunk.bx
    3   0.42%  92.91%  strerror_r
    3   0.42%  93.32%  mremap_chunk
    3   0.42%  93.74%  _int_realloc
    2   0.28%  94.02%  .L969
    2   0.28%  94.30%  realloc
    2   0.28%  94.58%  mallopt
```

```
Profile Information
==================================================================
Class              : PAPI
Event              : PAPI_TOT_CYC (Total cycles)
Period             : 50000
Samples            : 721514
Domain             : user
Run Time           : 17.60 (seconds)
Min Self %         : (all)
Module Summary
--------------------------------------------------------------------
Samples   Self %  Total %  Module
 465515  64.52%  64.52%  /afs/cern.ch/user/o/oplaatl3/testdll/libhello2.so.1
 255433  35.40%  99.92%  /afs/cern.ch/user/o/oplaatl3/testdll/libhello1.so.1
    391   0.05%  99.98%  /usr/bin/python
    145   0.02% 100.00%  /lib/tls/libc-2.3.2.so
     26   0.00% 100.00%  /lib/ld-2.3.2.so
      4   0.00% 100.00%  /lib/tls/libpthread-0.60.so
Function Summary
--------------------------------------------------------------------
Samples   Self %  Total %  Function
 255433  35.40%  35.40%  hello(int*)
 254920  35.33%  70.73%  sum(int*)
 210595  29.19%  99.92%  count(int*, int)
    392   0.05%  99.98%  ??
     36   0.00%  99.98%  _int_malloc
     22   0.00%  99.98%  memcpy
     13   0.00%  99.99%  __libc_malloc
     11   0.00%  99.99%  free
     10   0.00%  99.99%  do_lookup_versioned
      7   0.00%  99.99%  strcmp
      6   0.00%  99.99%  __open_nocancel
      5   0.00%  99.99%  _int_free
      4   0.00%  99.99%  memset
      4   0.00%  99.99%  malloc_consolidate
```

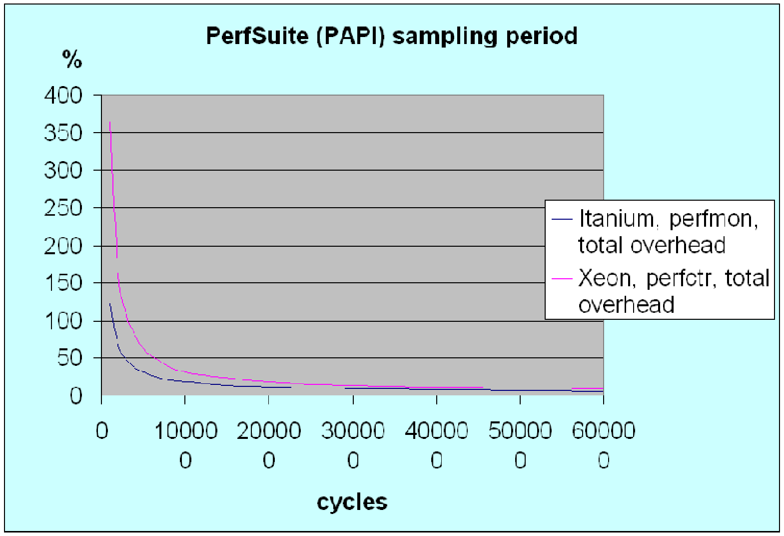
PerfSuite – Atlas simulation profiling

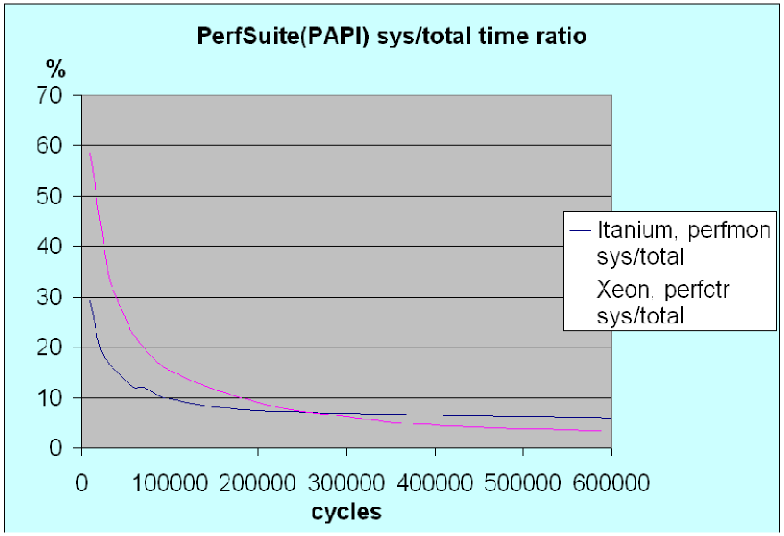
Function Summary

Samples	Self %	Total %	Function
30728046	**7.82%**	**7.82%**	**??**
7320624	1.86%	9.68%	G4Transportation::AlongStepGetPhysicalInteractionLength()
6689448	1.70%	11.38%	G4VoxelNavigation::ComputeStep()
5873290	1.49%	12.87%	G4Navigator::ComputeStep()
5490913	1.40%	14.27%	G4PolyconeSide::Intersect()
5278498	1.34%	15.61%	G4SteppingManager::Stepping()
5050076	1.28%	16.90%	G4PropagatorInField::ComputeStep()
4919562	1.25%	18.15%	G4Navigator::LocateGlobalPointAndSetup()
4503773	1.15%	19.30%	G4PolyconeSide::DistanceAway()
4366559	1.11%	20.41%	G4IntersectingCone::LineHitsCone1()
4295632	1.09%	21.50%	G4VoxelNavigation::LocateNextVoxel()
4199824	1.07%	22.57%	G4SteppingManager::DefinePhysicalStepLength()
4033883	1.03%	23.59%	G4MultipleScattering52::GetContinuousStepLimit()
3938509	1.00%	24.60%	G4SteppingManager::InvokePSDIP()
3912491	1.00%	25.59%	G4MultipleScattering52::PostStepDoIt()
3766114	0.96%	26.55%	G4ClassicalRK4::DumbStepper()
3722683	0.95%	27.50%	G4SteppingManager::InvokeAlongStepDoItProcs()
3620741	0.92%	28.42%	_int_malloc
3620692	0.92%	29.34%	G4Navigator::LocateGlobalPointAndUpdateTouchableHandle()
3604007	0.92%	30.25%	LArWheelCalculator::DistanceToTheNeutralFibre()
3598039	0.92%	31.17%	vfprintf
3581444	0.91%	32.08%	G4UniversalFluctuation::SampleFluctuations()
3259393	0.83%	32.91%	G4ElectroNuclearCrossSection::GetCrossSection()
3127408	0.80%	33.71%	G4PolyconeSide::PointOnCone()
2872853	0.73%	34.44%	G4NavigationLevelRep::G4NavigationLevelRep()
2833732	0.72%	35.16%	G4Transportation::PostStepDoIt()
2737712	0.70%	35.85%	G4ChordFinder::FindNextChord()
2733244	0.70%	36.55%	G4Tubs::DistanceToIn()
2721568	0.69%	37.24%	G4VEnergyLossProcess::AlongStepDoIt()
2654097	0.68%	37.92%	G4MagErrorStepper::Stepper()
2610649	0.66%	38.58%	G4ParticleChangeForTransport::UpdateStepForAlongStep()
2550219	0.65%	39.23%	G4SandiaTable::GetSandiaCofPerAtom()
2540628	0.65%	39.88%	G4PhotoNuclearCrossSection::GetCrossSection()
2413526	0.61%	40.49%	G4Transportation::AlongStepDoIt()
2407946	0.61%	41.10%	CLHEP::HepJamesRandom::flat()
2396557	0.61%	41.71%	G4PolyPhiFace::Intersect()
2390632	0.61%	42.32%	G4HadronCrossSections::CalcScatteringCrossSections()
2343439	0.60%	42.92%	G4MagInt_Driver::QuickAdvance()
2277101	0.58%	43.50%	CLHEP::HepRotation::rotateAxes()
2244256	0.57%	44.07%	G4PhysicsVector::GetValue()
2242211	0.57%	44.64%	G4SteppingManager::SetInitialStep()
2228671	0.57%	45.20%	G4Tubs::Inside()
2171964	0.55%	45.76%	G4NormalNavigation::ComputeStep()
2132567	0.54%	46.30%	G4VEmProcess::GetMeanFreePath()
2083184	0.53%	46.83%	G4VoxelNavigation::LevelLocate()
2067235	0.53%	47.35%	G4VoxelNavigation::ComputeVoxelSafety()
1978418	0.50%	47.86%	G4VoxelNavigation::VoxelLocate()
1944091	0.49%	48.35%	G4PropagatorInField::IntersectChord()
1892005	0.48%	48.83%	G4eBremsstrahlungModel::SampleSecondaries()
1889364	0.48%	49.31%	G4PolyconeSide::Inside()
1875195	0.48%	49.79%	G4PolyconeSide::Distance()
1853968	0.47%	50.26%	G4AffineTransform::G4AffineTransform()
1842997	0.47%	50.73%	G4EnclosingCylinder::MustBeOutside()
1841307	0.47%	51.20%	G4StepPoint::operator=()
1805522	0.46%	51.66%	G4ParticleChange::CheckIt()
1793612	0.46%	52.12%	G4VCSGfaceted::DistanceToOut()
1772012	0.45%	52.57%	G4TrackingManager::ProcessOneTrack()
1731569	0.44%	53.01%	G4HadronicProcess::GetMeanFreePath()
1631892	0.42%	53.42%	G4MagInt_Driver::AccurateAdvance()
1612075	0.41%	53.83%	G4ChordFinder::AdvanceChordLimited()
1546908	0.39%	54.23%	std::vector<G4VTrajectoryPoint*, std::allocator<G4VTrajectoryPoint*> >::_M_insert_aux()
1504737	0.38%	54.61%	G4VContinuousDiscreteProcess::PostStepGetPhysicalInteractionLength()
1471951	0.37%	54.98%	G4VEnergyLossProcess::GetMeanFreePath()
1456046	0.37%	55.35%	_int_free
1449124	0.37%	55.72%	G4VCSGfaceted::DistanceToIn()

Profiling challenges

- Profiling overhead on Xeon and Itanium2
  - bigger on Xeon with perfctr than on Itanium2 with perfmon
  - for small sampling periods more time is spent in a kernel than in a user space – „Heisenberg Effect”





Our Conclusions

- PMU and already available tools for IPF let you explore CPU resources in more details than on the x86 family

- both perfmon2 and pfmon include support for more and more processors and more useful features (event multiplexing, profiling) which makes them more interesting in our applications

Our Conclusions

- tools are a step behind hardware and do not take full advantage of performance units, e.g. BTS on Xeon

- one scalable and portable tool on different platforms would be an ideal solution
  - from 'hello world program' up to a huge framework
  - shared and dynamic loaded libraries
  - resolving function names

Our Conclusion

- simple profiling is not always a full answer, we need something more
  - number of function calls
  - call graph
  - without using hardware support we suffer from a big overhead (e.g. on Xeon PIN ~800%, ATOM 6300% with one of Geant4’s examples)

after the Gelato conference...

- We contribute to perfmon2 & pfmon by:
  - improving the resolution of function names from shared libraries
  - testing on x86 (Xeon) as well as on IPF
- Number of function calls
  - Dynamic instrumentation
    - PIN & ATOM
  - triggers (x86) and check-point options in pfmon
- preparing to move to the 64bit performance monitoring

Questions and answers

Q&A?