Towards a Flexible *and Realistic* Hardware Performance Monitor Infrastructure

Philip J. Mucci

Innovative Computing Laboratory,
University of Tennessee, Knoxville, TN

Parallel Center for Computers,
Royal Institute of Technology, Stockholm, Sweden

`http://icl.cs.utk.edu/~mucci`

Maintain A User-Centric Focus

- This is not the Grid.
  - This work does not exist to maintain a funding pipeline.
- Our goal is to **empower the users to improve the efficiency** of these systems.
- The audience for this work is already small, so we must maintain focus.
  - Smaller factions, like power consumption/monitoring will have even a harder time.

Next Generation Design

- While highly advanced functionality may be desired, we must strive to produce *realistic recommendations* that stand a chance of being *heard*.
- This means that the recommendations should:
  - Be implementable without Alien intervention.
  - Require reasonably low architectural complexity.
  - Tolerant routing requirements.
  - etc...

Delivering Functionality

- Performance monitor functionality was never asked for.
  - These counters were discovered “under a rock”, and then exercised and exposed.
  - Slowly led to usage by the people that really needed them who never knew they even existed (and couldn't ask for them.)
- Features that exist in the hardware but not in software, do not exist.
- You can lead or you can follow.

Brinkley's Killer App

- Stop trying to 'figure out' what to do. Measure it!
  - Numerical kernels. (Atlas)
  - Aggressive source transformations. (Rose)
  - Compilers. (PGO)
  - Schedulers. (HT-aware)
  - Page placement/migration. (SunFire)
  - Network collectives. (consider binomial vs. binary broadcast on IB/Myrinet)
- As software engineers we must push for the availability of this functionality. (prefetching)

What # of Counters?

- Our scope of the understanding of usage is too narrow.
  - System Monitoring
  - System/Kernel Dynamic Adaptation
  - Application Monitoring
  - Application Performance Analysis

System Monitoring

- Evaluate the performance of a system as a whole.
- Snapshot, high-level views.
- Continuous collection, aggregation.
- No support for HT/CMT/SMT needed.

System Monitoring Applications

- PerfMiner
- Ganglia
- NWPerf
- SuperMon
- CluMon
- Nagios
- PCP

System Optimization

- Adaptive Kernel Subsystems
  - Dynamic page migration
  - TLB coalescing
  - Advanced HT/SMT scheduling.
- System throughput optimization
  - Profile samples that cross user/kernel domain.

System Optimization Mechanisms

- Oprofile
- Perfmon
- DCPI/ProfileMe
- KernInst
- DTrace

Application Monitoring

- Measure actual application performance via batch system. (or BSD like collection mechanisms.)
  - Workload characterization
- Per thread/per application metrics.
- Isolate deficits in throughput, efficiency and productivity.
- Dedicated CMT/SMT/HT counters.

Application Monitoring Systems

- PerfMiner (+ Easy)
- NWPerf
- Work at NCSA (+ OpenPBS)

Application Compilation, Analysis, Modeling and Optimization

- Focused on items code that the user has direct control over.
- Non-SUID/non-root/exclusive thread scope access and virtualization
- This is the focus of most user tools.
- Dedicated CMT/SMT/HT counters.

Compilers and Tools

- HPCToolkit
- PerfSuite
- SvPablo
- TAU
- Vampir
- Lots of vendor tools, compilers and modeling systems.

What Number of Counters?

- At least 2, possibly 3 of these systems must exist simultaneously.
- 1 needs replicated hardware for CMT/SMT/HT.
- Hardware measurements are **never** singletons.
- When measuring performance, the set of usable registers should be able to measure:
  - At least 2 ratios. (TLB miss-rate, BP corr. predicted)
  - Total cycles.
- Consider 2-3 blocks of 6 counters.

Consider 2 Blocks of 6 Counters

- Supports system monitoring/profiling and application tuning and analysis.
- Each block is it's own domain and must be protected and be able to be used independently!
- Symmetric design is ideal but not required.

2 Blocks of 6 Counters

- 1 control register per group, with individual event select/mask fields.
  - Keeps counter set up cost/code very low.
- High speed counter 'kill bit'.
  - Allows user code to quickly pause/enable the counters without syscalls. (IA64)
  - Counter control operations are always privileged.

2 Blocks of 6 Counters

- Guess what? >= 32 bits is enough.
  - Current software assumes counter will not overflow during a time-slice.
  - Software always has to handle overflow regardless of size for statistical profiling.
  - Counters be part of the process/thread struct for the for the application domain.
    - Saved/restored on context switch.
    - Lazy evaluation like FP registers.

What method of access?

- Always READABLE by regular user mode programs.
  - Syscall is almost 1000 cycles on IA64.
- Shame on you guys!
  - Less than a dozen cycles would be awfully nice.
    - Opteron 1.4: 14 cycles
    - Athlon64: 20 cycles
    - Pentium IV (model 3/2): 226/146 cycles
    - PentiumPro: 33 cycles
    - PPC750: 2 cycles (Whew...)

What method of access?

- Precise interrupt information.
  - Hardware should identify which counter.
  - Hardware should assist instruction and data address attribution. Either through:
    - Deterministic wait
    - Precise interrupt mode. (Like FP exceptions...)
    - Deposit of instruction virtual address either to a buffer or just a mailbox.
    - Provide virtual address of last data access prior to event.

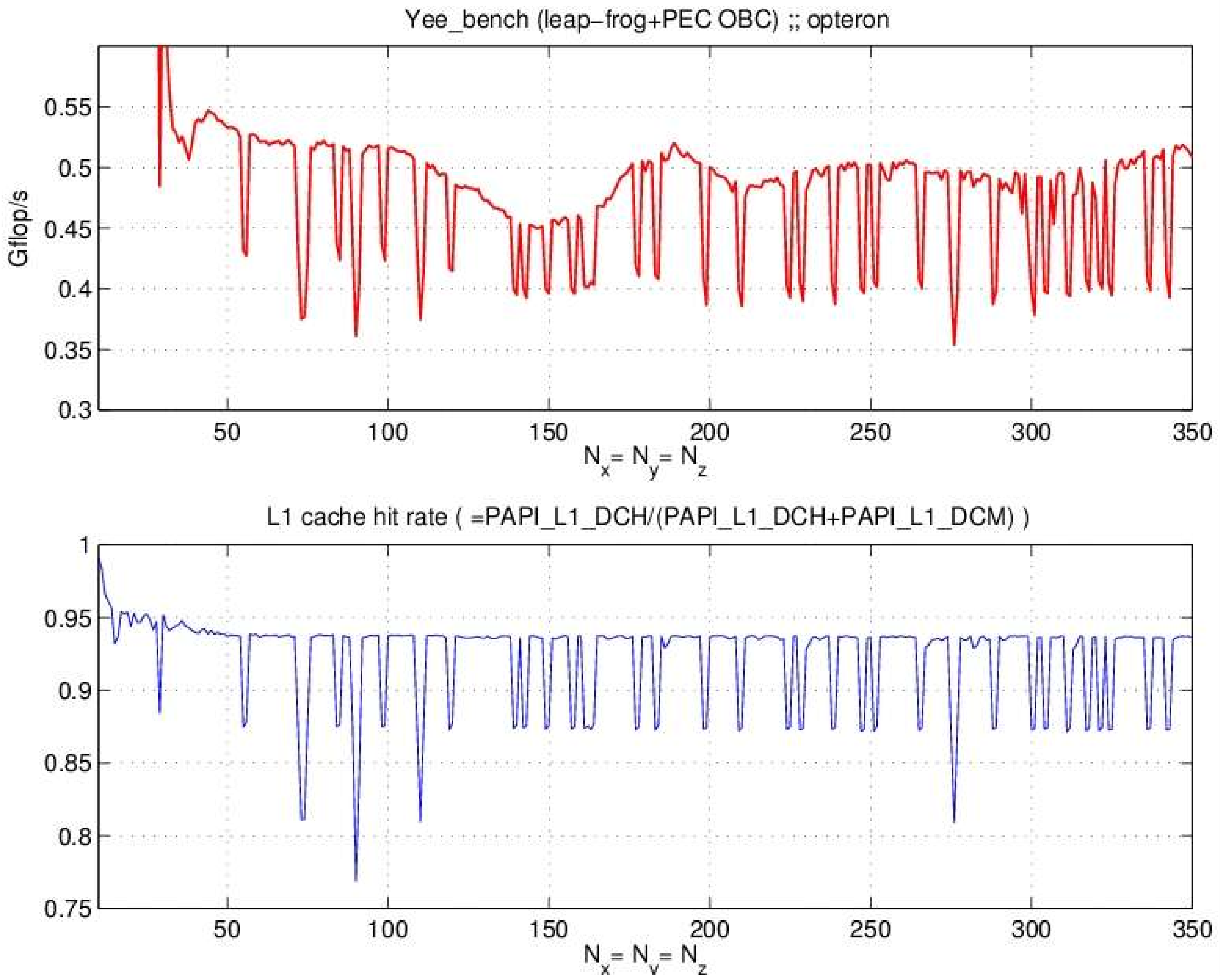
Which events?

- A big question. We must stay focused on what the counters are used for.
- The goal of our work should not initially be to service the needs of a small research community.
- Only the 'simplest' events are of meaning to the average application engineers.
  - Rudimentary knowledge of processor microarchitecture.
  - Can easily be abstracted from detailed processor metrics by the right software. (Shameless PAPI plug.)

Which events? (cont.)

- Remember the average user?
  - LD/ST/Prefetch
  - I/D Cache Miss/Accesses at every level
  - Conditional Branches (TK,NTK,CRP,MPR)
  - Work (Integer, FP, Vector)
  - SMP protocol events.
  - FP exceptions/traps.

Example


Yee_bench (leap–frog+PEC OBC) ;; opteron
Gflop/s
0.55
0.5
0.45
0.4
0.35
0.3
50
100
150
200
250
300
350
$N_x = N_y = N_z$
L1 cache hit rate (=PAPI_L1_DCH/(PAPI_L1_DCH+PAPI_L1_DCM))
1
0.95
0.9
0.85
0.8
0.75
50
100
150
200
250
300
350
$N_x = N_y = N_z$

Which events? (cont.)

- Stall metrics that relate to:
  - Processor stalls. (Implies the latter)
  - Functional unit/queue stalls. (Does not imply the former.)
- Functional unit/queue activity.
  - Power monitoring.
- Event thresholding.
- Edge detect for actual costs.
- Does not care about issued counts. (Can I do anything to really change it?)

Which events? (cont.)

- Non aggregate functionality
  - Precise interrupt functionality.
  - Hardware support for randomization.
  - Hardware support for event tracing/sampling.
    - Locality, Latency (DA and PC)
    - Branch behavior (From, to PC)
    - SMP/Numa traffic (From, to, VA)
  - A good/fast virtual to physical mapping mechanism.

General Suggestions...

- Stop ADDING events. Delete them!
- Remember the R10K?
  - 32 *reasonably* well documented and *almost* verified registers.
  - Pentium IV space is about 30,000 (legal and non-legal) configurations.
- Counter groups make usage and programming hard.

General Suggestions...

- Don't think of an operation in a particular functional unit as always executing work.
  - Don't include register moves in floating point counts. (you know who you are...)
  - I want to count FP events. Don't make me pick between single, double, packed, unpacked vector or standard floating point operations.
- Giving me ½ of an important ratio gives me nothing.

A Standardized Linux Interface

- IMHO, not quite bad as Stephane makes it out to be. There are only 2 interfaces. (Oprofile doesn't have one.)
- Numerous tools have been developed or ported support hardware performance counters with an interface that hides the complexity.
  - PerfSuite, HPCToolkit, SvPablo, Tau, lots of others...
  - Portland Group Prof, Allinea's Opt Tool, Vampir, Paraver
- Many tools run cross platform out of the box today with native event support.

The PerfCtr Linux Interface

- PerfCtr is x86/x86_64/PPC/PPC64 and I have personally ported to MIPS and PPC440 in < 2 weeks.
- Perfmon got the grandfather treatment. No fair!
- PerfCtr integration is in the Andrew Morton kernels.
- Consider that one can perform 100 measurements of the Opteron in the time one can do 1 on the IA64 from the lowest level API.

Mandatory Software Functionality (Kernel)

- Virtualized, memory mapped access to counters.
  - User level instruction to read the counter.
  - Accumulation of hardware counter with 64 bit quantity mmap'ed from the kernel's thread struct.
- Virtualized TSC. (provides a simple and high resolution virtual timer. getrusage() runs at HZ.)
- Interrupt dispatch to user level.
  - At a minimum, this is a signal delivered to the process or thread who's counter overflowed. (AIX!)
  - Multiple counter overflow.

Additional Kernel Functionality

- Kernel level counter multiplexing
- Better handling of PMC interrupts:
  - Buffered interrupts. Save a bunch and their contexts in a memory mapped buffer.
  - Double buffer for lossless operation.
- Trace/profile buffers for address/branch/event sampling/tracing.
- Lightweight event dispatch mechanism. (This is a Unix problem solvable by kernel mechanism.)
- Randomization? Show me the money.

PerfCtr + PerfMon

- This merge could provide everything we need and almost what we want.
- Perfmon exceptions:
  - Virtual TSC
  - High speed mmap()'d counter access through user library.
  - Multiplex implementation can be improved like that in PAPI.
- Working with Stephane and Mikael to make this happen. Redhat/Suse waiting...