ECE 571 – Advanced Microprocessor-Based Design Lecture 6

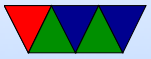
Vince Weaver

http://www.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

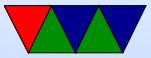
31 January 2013

Hardware Performance Counters: The Operating System Interface



Operating Systems

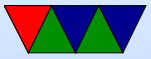
- UNIX – long history of support
- Windows – no native support (can get Intel Vtune)
- OSX – no native support (can get shark)
- Linux – On 95% of Top 500 computers, many embedded systems



Operating System Interface

A typical operating system performance counter interface will provide the following:

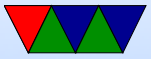
- A way to select which events are being monitored
- A way to start and stop counting
- A method of reading counter results when finished, and
- If the CPU supports notification on counter overflow, some mechanism for passing on overflow information



Operating System Interface

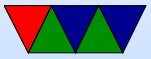
Some operating systems provide additional features:

- Event scheduling: often there are limitations on which events can go into which counters,
- Multiplexing: the OS can hide the fact that only a limited number of counters are available by swapping events in and out and extrapolating counts using time accounting,
- Per-thread counting: by loading and saving counter



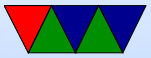
values at context switch time a count specific to a process can be achieved,

- Attaching to a process: counts can be taken from an already running process, and
- Per-cpu counting: as with per-thread counting, counts can be accumulated per-cpu.



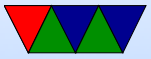
Older Linux Interfaces

- Historical – typically just exported msrs
- Oprofile – only does profiling
- Perfctr – good but required kernel patch
- Perfmon2 – was making headway until perf_event came from nowhere and became official

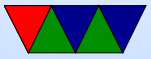


perf_event

- Developed from scratch in 2.6.31 by Molnar and Gleixner
- Everything in the kernel
- `perf_event_open()` syscall (manpage still under development)
- perf_event_attr structure with 40 complex interdependent parameters
- ioctl() system call to enable/disable

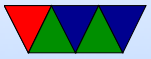


- read() system call to read values
- can gather sampled data in circular buffer
- can get signal on overflow or full buffer



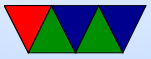
perf_event Generalized Events

- perf_event provides support for "common" generalized events
- makes things easier for user at expense of papering over the differences between events
- events need to be validated to make sure they are providing useful results



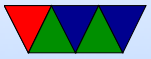
perf_event Generalized Events Issues

- Which event to choose (Nehalem)
- From 2.6.31 to 2.6.35 AMD "branches" was taken not total
- Nehalem L1 DCACHE reads.
  PAPI uses `L1D_CACHE_LD:MESI`;
  perf uses `MEM_INST_RETIRED:LOADS`



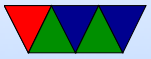
perf_event Event Scheduling

- Some events have hardware constraints. Can only be in one counter
- You can do this scheduling in userspace; lets the algorithm be changed more easily
- Scheduling can be expensive; do so at event start can slow things down.



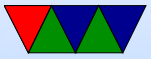
perf_event Multiplexing

- You may wish to measure more events simultaneously than hardware can support (NMI watchdog may steal one too)
- perf_event supports this in-kernel (you can also do this in userspace)
- there are various ways to try to ensure good statistical results. in kernel you have to trust the kernel programmers.



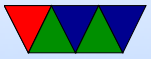
perf_event Event Names

- Event names are provided in the hardware manuals, but can be inconsistent
- Traditionally used libraries to provide names. libpfm4
- perf tool is starting to provide own list of events (they refuse to link libpfm4) that are based on a hybrid of libpfm4 and kernel names
- Also some event names are provided by the kernel under `/sys`



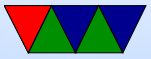
perf_event Software Events

- perf_event provides internal kernel events through same interface
- `page-fault`, `task-clock`, `cpu-clock`, etc.

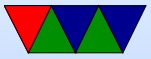


perf_event Perf Tool

- Included with kernel source code
- Tied to kernel, but backwards compatible
- Most kernel devs use this rather than outside tools

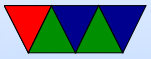


perf_event Hardware Features



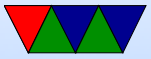
Offcore Response

- Allows measuring memory events that go “off” the core
- Requires access to two different MSRs.
- Shared resource, requires extra handling
- “raw” access to events delayed until “generic” support available



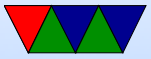
Uncore/Northbridge

- On a chip there are shared areas not the “core”
- Memory controller, L2 / L3 cache, etc.
- Additional counters and events to measure these.
- Shared resource. Could leak information. Need extra handling.



Last Branch Record

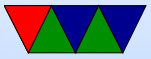
- Useful for backtraces and also debugging



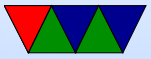
Sampled Interfaces

- AMD IBS – Instruction based sampling
  address, latency, cache miss, TLB miss obtained along with minimal "skid" (results provided match exactly with PC so can attribute the values to that which caused it)

- Intel PEBS – Precise Event-Based Scheduling
  additional information can be configured to be collected immediately after an event is triggered. Full register state as well as latency

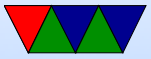


- current perf_event support limited to reduced skid, work underway for the rest



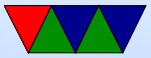
rdpmc instruction

- Allow users direct reads of performance counters w/o system call
- In theory should be faster as less overhead
- on perfctr was faster; on perf_event not so much for unknown reasons. part of the issue is perf_event can only do delta, requiring two calls



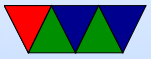
AMD Lightweight Profiling

- Attempts to give full support of profiling to user. No need for kernel. Mostly support need to enable the feature and save extra state on context switch
- perf_event refuse to merge support; insist kernel should control all

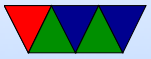


Virtualized Counters

- How to handle when running inside Virtual machine?
- Can measure at different levels; outside total performance, inside performance, hypervisor performance
- Recent Linux supports passing performance counter values inside
- Various limitations. Compatibility of interface? Save/restore when VM switched out?

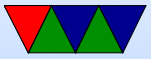


- Does help with performance analysis; before in absence of steal time data, time has “no meaning” inside of VM

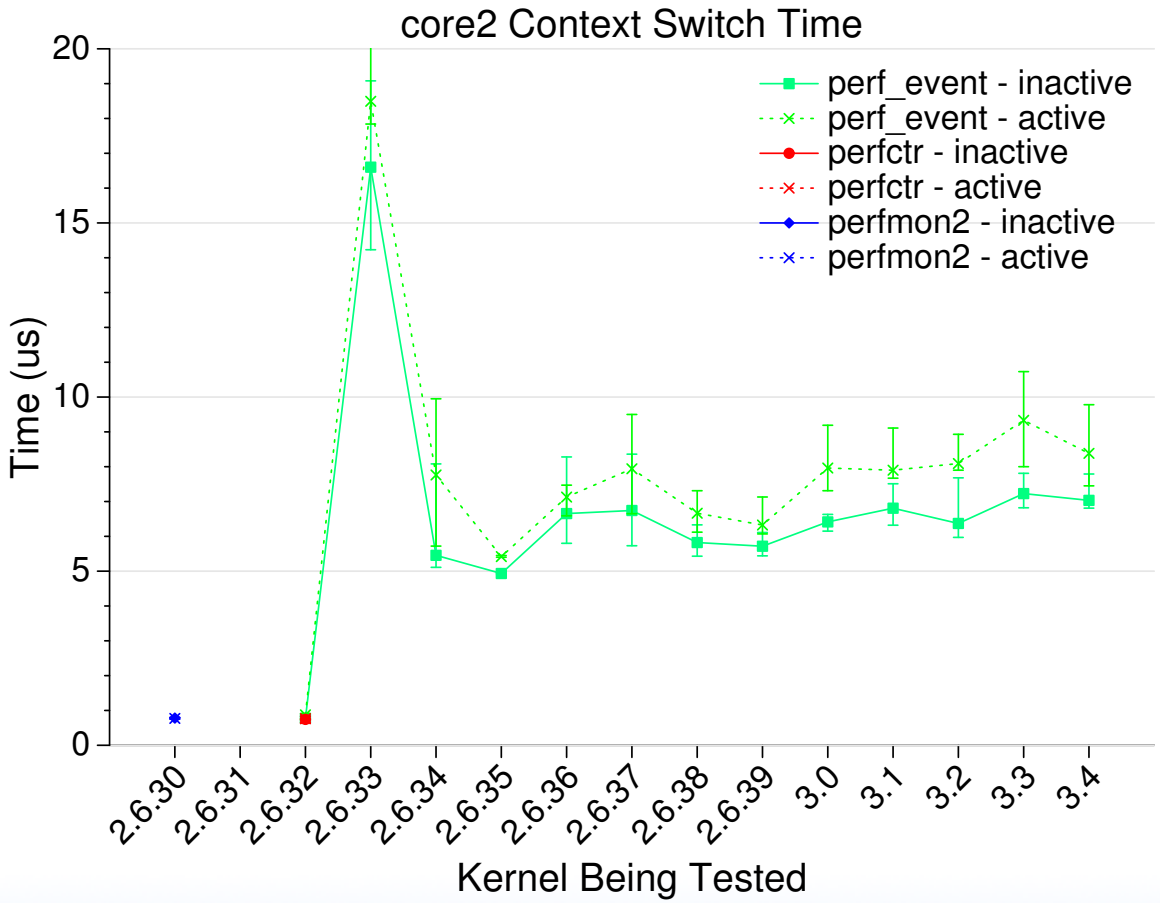


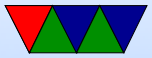
non-CPU counters

- things like network cards, GPUs, etc.



perf_event Context Switch Overhead





perf_event Start/Stop/Read Overhead