Understanding Performance

• To understand low-level performance, we need visibility into what the machine is doing
  – We mostly only (mostly) control data and code
• Sometimes it is enough to know time
  – Algorithm-level fix, data-structure-level fix
• But to make an existing algorithm/data-structure work better, we need to know how it interacts with HW
  – Must answer the why

Major Types of HW Support

● Performance Counters
  – Region granularity
● Event based sampling
  – Imprecise usually
  – Limited information per run
  – Instruction granularity

● Instruction based sampling
  – Instruction granularity
  – Precise
  – Lots of Information per run
  • Can capture address traces

Hardware Generates Events

● What depends on the implementation
  – L1 cache misses
  – TLB prefetch triggers
  – FP ops
  – FMA issues
  – Stalled ROB cycles
● Where does this information go?
  – Usually nowhere

Performance Counters

● We need to count the events
● HW provides special programmable counters
● Program a counter to count a specific Event
● Software (ring-0) can read and write the counters

Performance Counters - Use

● Very fine-grained detail
  – e.g. Number of times processor stalled because register file couldn’t allocate enough registers for all issuable instructions
  – e.g. number of Prefetches generated by the prefetch engine (v.s. Number of prefetches issued (not the same thing!))
● Course-grained regions
  – We can count over a region of code
  – Fairly expensive to read counter, so finer granularity causes increased perturbation
Using Performance Counters

Much like gettimeofday() and printf
Instrument region to read the counter before and after
Print the value

What's wrong with performance counters?

Sampling

• What we want
  – fine granularity information – information about individual instructions
  – cheap collection
• Consider political polls

Event Based Sampling

• Let's solve the course-granularity problem
• Extend performance counters with a programmable limit value
• Cause an interrupt when the counter overflows (exceeds limit value)
• That's it (from the hardware side)

EBS - Software

• Interrupt handler in the OS
  – Records PC (and process id) from interrupt
  – Records event type
  – Resets counter
• Produces a log of <PC,event> pairs
• Map PCs back to code
• Now we have a sampled profile of any event we want
  – Which branches have bad miss rates?
  – Where are the high latency memory accesses?

EBS - Inaccuracies

• Interrupt is generated after event
• Pipeline drains instructions
• Skew: latency from PC triggering event to interrupt
• Masking of events by other events
• Events missed during sampling interrupt
• Events miss counted during SMI
• Other platform-specific problems
**EBS – In Practice (HW)**

- HW has many types of events and sub-events it can monitor
  - Nehaleum ~100 documented
    - Only 7 are defined as stable across future processors
- HW has a few programmable counters
  - Various restrictions on what can be programmed
    - e.g. Nehaleum: 3 fixed, 4 programmable, only 2 can count any event
- Usually fairly flexible overflow programming

**EBS – In Practice (SW)**

- Usually a flat profile (why?)
- Sampling can be used to collect more events than HW supports simultaneously
  - Rotate which events are currently programmed
- Extremely useful for monitoring OS overhead
- Low overhead
  - Unless long runs
  - Unless short sampling period
- Software usually corrects for skew as best it can

**Problems with Event Based Sampling**

- Limited data collection
  - Per run
  - Types of data
- In-exact
  - Sample point and event cause don’t match

**Instruction Based Sampling**

- Let’s collect all the information, some of the time

**IBS – Basic Idea**

- Eliminate skew by recording information for specific instructions
- Eliminate the need for multiple runs by collecting every useful bit of information
  - e.g. Virtual and physical address trace
    - Allows data-centric profiling
  - e.g. branch history trace
    - Correlated branches can be optimized
- Do this without pipeline overhead
  - Don’t slow down the processor to collect data

**IBS – how much data?**

- Recording the entire state of the processor is expensive
  - At least 2kb of interesting state per cycle
  - 4 GHz -> 1TB of data per second
- Sample instructions
  - About 64 Bytes of interesting data
  - At every 10,000 instructions, 4GHz, IPC 2 -> 24.4 MB per second
**IBS – What Information - AMD**

- The instruction address for the op
- The tag-to-retire time (cycles from tagged to retired)
- The completion-to-retire time (cycles from completed to retired)
- Whether branch semantics (a “branch op”)  
  - If the branch op was mispredicted  
  - If the branch was taken  
  - If the branch was a return  
  - If the return was mispredicted
- Whether the op was a resync that caused a pipeline flush
- Whether the op performed a load and/or store operation  
  - If the operation missed in the data cache  
  - If the operation missed in the level 1 or level 2 DTLB  
  - The page size of the level 1 or level 2 address translation  
  - If the operation caused a misaligned access  
  - The DC miss latency (in cycles) if the load operation missed in the data cache  
  - The virtual and physical address of the requested memory location  
  - If the access was made to local or remote memory

**IBS – AMD Fetch**

- The fetch address
- Whether the fetch completed or aborted
- Whether the fetch missed in the instruction cache
- Whether the fetch missed in the level 1 or level 2 instruction translation lookaside buffer (ITLB)
- The page size of the address translation
- The fetch latency, i.e., cycles from when the fetch was initiated to when the fetch either completed or aborted

**IBS - Mechanism**

- HW selects instruction to monitor and tags it
- Record all events and stats as tagged instructions execute  
  - Completion time, execution time, branch stats and address, ld/st stats and addresses, cache stats, latencies, etc
- At retire, record info to a log
- Log is stored in memory as a circular buffer  
  - Asynchronous and infrequent passing of data to userspace (rather than every sample)  
  - Data doesn't have to be completely written at retire