Can Hardware Performance Counters Produce Expected, Deterministic Results?

Vince Weaver

Innovative Computing Lab, University of Tennessee

3rd Workshop on Functionality of Hardware Performance Monitoring

4 December 2010

Uses of Deterministic Events

- Simulator Validation
- Basic Block Vector (BBV) Analysis
- Feedback-Driven Optimization
- Deterministic Multi-threading
- Intrusion Analysis (via Replay)
- Performance Analysis

Ideal Deterministic Events

- Results are same run-to-run
- Event is frequent enough to be useful
- The expected count can easily be determined by code inspection
- Available on many processors

Some Sources of Non-Determinism

- Operating System Interaction
- Program Layout / Address Space Randomization
- Measurement Overhead
- Multi-Core Interaction
- Hardware Implementation Details

Common Potentially Deterministic Events

- Retired Instructions
- Retired Branches
- Retired Loads and Stores
- Retired Multiplies and Divides
- Retired μops
- Retired Floating Point and SSE
- Other (`fxch`, `cpuid`, move operations, serializing instructions, memory barriers, not-taken branches)

How Do We Find Sources of Non-Determinism?

- Run existing large benchmarks and wade through large instruction traces when things don't match up?
- Use Dynamic Binary Instrumentation for comparison?
- Create many small assembly benchmarks to find inherent overhead?
- Create a large assembly benchmark that attempts to exercise all corners of the architecture?

Our Assembly Benchmark

- Exercises most Integer, X87, MMX and SSE instructions.
- Tests a wide variety of address modes and bit-widths.
- Executes over 200 million dynamic instructions.
- Available from our website

  http://www.cs.utk.edu/~vweaver1/projects/deterministic/

x86_64 Machines Investigated

Processor	Kernel
Intel Atom 230	2.6.32 perf events
Intel Core2 T9900	2.6.32 perf events
Intel Nehalem X5570	2.6.31 perf events
Intel Nehalem-EX X7560	2.6.34 perf events
Intel Pentium D	2.6.28 perfmon2
AMD Phenom 9500	2.6.29 perfmon2
AMD Istanbul 8439	2.6.32 perf events

Contributors to Retired Instruction Count on x86_64

- +1 extra for every Hardware Interrupt
- +1 extra for each page-fault
- +1 for first floating point instruction
- +1 extra for each floating point exception
- +1 on AMD on FP state save if exception bit set
- Instruction double counts on Pentium D with `INSTRUCTIONS_RETIRED:NBOGUSNTAG`

Retired Instruction Results

Difference from the expected 226,990,030

Machine	Before Adjustment	Adjusted
Core2	10,879±319	13±1
Atom	11,601±495	-41±12
Nehalem	11,409±3	8±2
Nehalem-EX	11,915±9	8±2
Pentium D *(inst retired)*	2,610,571±8	561±3
Pentium D *(inst completed)*	10,794±28	-50±5
Phenom	310,601±11	12±0
Istanbul	311,830±78	11±1

Contributors to Retired Branch Count on x86_64

- +1 extra for every Hardware Interrupt
- +1 extra for each page-fault
- Core2 counts `cpuid` as a branch

Retired Branch Results

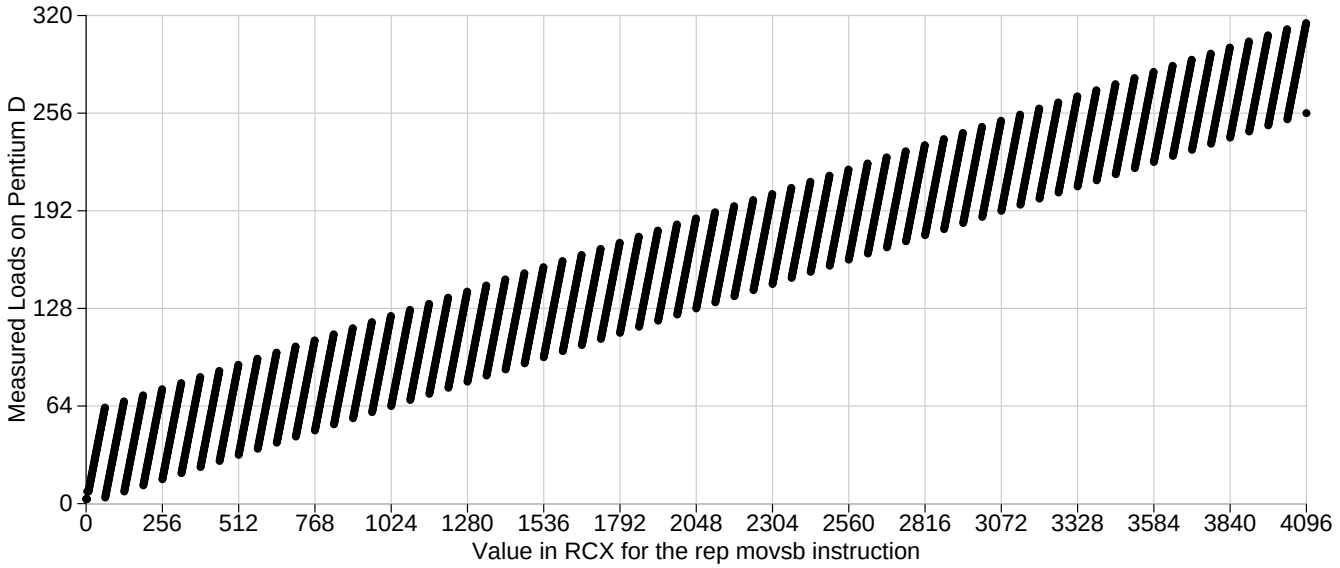
Difference from the expected 9,240,001

Machine	Before Adjustment	Adjusted
Core2	111,002±332	13±1
Atom	11,542±11	-43±4
Nehalem	11,409±4	8±1
Nehalem-EX	11,914±7	8±1
Pentium D	10,773±2	-56±5
Phenom	10,598±5	9±0
Istanbul	11,819±10	8±2

Contributors to Retired Load Count

- +1 extra for every Hardware Interrupt
- +1 extra for each page-fault
- +1 extra for FP/SSE exceptions
- Conditional moves *always* a load
- `fbstp` 80-bit FP BCD store also counts as load
- `rep cmps` counts as one (not two) loads
- Core2 counts various instructions twice
- Nehalem does *not* count `padd`

Pentium D Retired Loads



- Exposes microcoded loads (various counted double)
- Prefetches *not* included
- Page faults 5 loads, `fldenv` 7, `frstor` 23, `fxrstor` 26.

Retired Load Results

Difference from the expected 79,590,000

Machine	Before Adjustment	Adjusted
Core2	1,710,807±376	14±1
Atom	—	—
Nehalem	-288,590±3	9±1
Nehalem-EX	-288,086±7	8±3
Pentium D	2,402,843,955±12	3096±17
Phenom	—	—
Istanbul	—	—

Contributors to Retired Store Count on x86_64

- +1 extra for every Hardware Interrupt
- +1 extra for each page-fault
- Nehalem counts `cpuid`, `sfence` `mfence` and `clflush`
- Pentium D exposes microcode behavior, as with loads

Retired Store Results

Difference from the expected 24,060,000

Machine	Before Adjustment	Adjusted
Core2	0±0	0±0
Atom	—	—
Nehalem	411,408±4	9±1
Nehalem-EX	411,914±6	9±1
Pentium D	163,402,604±185	11,776±175
Phenom	—	—
Istanbul	—	—

Retired Mul/Div

Unadjusted results

Machine	Multiplies	Divides
Core2	15,800,049± 68	5,800,016± 33
Atom	13,700,000± 0	7,000,000± 0
Nehalem	19,975,243± 1202	3,125,067± 48
Nehalem-EX	8,514,161±758,870	3,246,165±9,162
Pentium D	n/a	n/a
Phenom	69,242,930± 62,492	n/a
Istanbul	69,796,975±219,398	n/a

Retired Floating Point

Unadjusted results for various available FP events.

Machine	FP1	FP2
Core2	72,600,239±187	39,099,997±0
Atom	38,800,000± 0	n/a
Nehalem	50,150,590±131	17,199,998±2
Nehalem-EX	50,155,704±562	17,199,998±2
Pentium D	100,400,310±413	n/a
Phenom	26,600,001± 0	112,700,001±0
Istanbul	26,600,001± 0	112,700,001±0

Retired SSE

Unadjusted results for SSE

Machine	SSE
Core2	23,200,000± 0
Atom	88,299,597± 792
Nehalem	24,200,849± 154
Nehalem-EX	24,007,005± 197,401
Pentium D	54,639,963±4,943,158
Phenom	15,800,000± 0
Istanbul	15,800,000± 0

Retired μops

Machine	μops
Core2	14,234,856,824± 8,926
Atom	12,651,163,475± 63,870
Nehalem	11,746,070,128±258,282
Nehalem-EX	11,746,732,506± 47,996
Pentium D	12,551,781,963± 4,601
Phenom	10,550,974,722± 36,819
Istanbul	10,551,189,637±139,283

Other Architectures

- ARM – Cannot select only userspace events
- ia64 – Loads, Stores, Instructions appear deterministic
- POWER – On Power6 Instructions appear deterministic, Branches do not
- SPARC – on Niagara1, Instructions appear deterministic

Compensating

- **Aggregate Counts** – can be adjusted after the fact if all relevant data can be captured in parallel.
  Double-counted instructions are difficult.

- **Overflow Mode** – more difficult. One solution is to stop early and slowly single-step.

DBI Tools

- Pin, Valgrind, Qemu, etc.
- All count `rep` prefixed string instructions individually, requiring intervention
- Pin gives expected result on our assembly benchmark
- Inject own environment variables, disrupting glibc-using programs

SPEC CPU 2000 Benchmarks

Retired Instruction Results on Core2

Bench	Pin Results	Adjusted Counter Results	Difference
wupwise	360,553,377,202± 0	360,553,378,908± 175	1,706
gzip.graph	65,982,806,258± 0	65,985,332,330± 9	2,526,072
gcc.expr	7,253,042,753± 71	7,257,808,289± 43	4,765,536
eon.cook	59,410,255,668±144	59,432,884,285± 211	22,628,617
art.110	37,455,717,089± 0	37,684,112,743± 46	228,395,654

- Some run-to-run variation even in the DBI tool.
- Difference seems to be due to non-determinism in applications, not the counters.

Conclusions / Future Work

- It may be possible to obtain deterministic counts on x86_64, though it is more difficult than on other architectures.
- More work needs to be done on understanding the behavior of larger applications.
- We hope to use this work as a basis to understand the behavior of *Non*-Deterministic events

Questions?

vweaver1@eecs.utk.edu

http://www.cs.utk.edu/~vweaver1/projects/deterministic