Non-Determinism and Overcount on Modern Hardware Performance Counter Implementations

Vince Weaver

University of Maine

vincent.weaver@maine.edu

Dan Terpstra

University of Tennessee

terpstra@icl.utk.edu

Shirley Moore

University of Texas at El Paso

svmoore@utep.edu

ISPASS 2013 – 23 April 2013

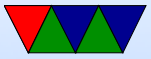
Hardware Performance Counters

- Low-level CPU counters measuring architectural events
- Not always documented well
- Never guaranteed by hardware engineers to be accurate
  Tend to be a bit of an afterthought
- Can they be deterministic?

Deterministic Program Example

Execute for exactly 10 million instructions on x86_64:

```
    #   total is 2 + 1 + 4999997*2 + 3
    .globl _start
_start:
    xor %rcx,%rcx        # pad total to 10M
    xor %rax,%rax        # pad total to 10M
    mov $4999997,%rcx    # load counter
loop:
    dec %rcx
    jnz loop             # repeat 4999997 times
exit:
    xor %rdi,%rdi        # return value of 0
    mov $60,%rax         # put exit syscall number (60) in rax
    syscall
```

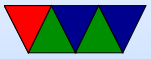



Results

```
perf stat -e instructions:u,r5301cb:u ./ten_million
 Performance counter stats for './ten_million':

        10,000,006 instructions:u              #    0.00  insns per cycle
                 2 r5301cb:u
...
        10,000,004 instructions:u              #    0.00  insns per cycle
                 1 r5301cb:u
...
        10,000,008 instructions:u              #    0.00  insns per cycle
                 3 r5301cb:u
```

Results on IvyBridge too high by $2 + (2 * \texttt{r5301cb:u})$? Why?

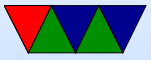


What Makes a Useful, Deterministic, Event?

- The result does not change run-to-run
  (it is not speculative)
- The expected value can be determined by code inspection
- The event occurs often in generic code

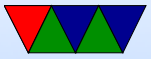
Is This Really a Problem?

- Have observed up to 2% error on real benchmarks, but often it is much less.
- Who needs Deterministic Events?



Uses of Deterministic Events

- Simulator Validation – compare against hardware
- Validating Basic Block Vectors
- Feedback Directed Optimization – want precise sample rate
- Hardware Checkpointing / Rollback, Intrusion Analysis – need to replay asynchronous events at exact time
- Parallel Deterministic Execution – want execution (and especially locks) to be deterministic



External Sources of Non-Determinism

- Operating System Interaction
- Program Layout
- Measurement Overhead
- Multi-thread interactions

Custom Assembly Benchmark

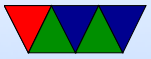
- Hand-coded microbenchmark with over 200 million dynamic instructions
- Exercises most integer, x87 floating point, MMX, and SSE instructions (up to SSE3)
- Various types of memory accesses, operand sizes (8-bit through 128-bit SSE) and addressing modes
- Code is looped many times to make anomalies stand out
- Compare against value from code inspection, also validate with DBI Utils (Pin, Valgrind, Qemu)

x86_64 machines investigated

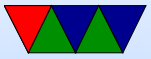
Processor	Linux Kernel	
Intel Atom 230	3.2	perf events
Intel Core2 X5355	2.6.36.2	perf events
Intel Nehalem X5570	2.6.38.6	perf events
Intel Nehalem-EX X7550	2.6.32-RHEL6	perf events
Intel Westmere-EX 8870	3.2	perf events
Intel SandyBridge-EP	2.6.32-RHEL6	perf events
Intel IvyBridge i5-3427U	3.2	perf events
Intel Pentium D	2.6.28	perfmon2
AMD Phenom 9500	2.6.29	perfmon2
AMD Istanbul 8439	2.6.35	perf events
AMD Bobcat E-350	3.2	perf events

Event Types Investigated

- total retired instructions
- retired branches (total and conditional)
- retired loads and stores
- retired floating point and SSE
- *not* speculative events (retired μops) or uncommon events (move instructions, cpuid, serializing, barriers, etc.)



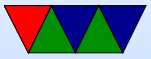
	Intel Core2	Intel Nehalem / Westmere
Retired Instructions	INSTRUCTIONS_RETIRED *(instructions:u)*	INSTRUCTIONS_RETIRED *(instructions:u)*
Retired Branches	BRANCH_INSTRUCTIONS_RETIRED *(branches:u)*	BRANCH_INSTRUCTIONS_RETIRED *(branches:u)*
Retired Cond Branches	BR_CND_EXEC *(r53008b:u)*	BR_INST_RETIRED:CONDITIONAL *(r5301c4:u)*
Retired Loads	INST_RETIRED:LOADS *(r5001c0:u)*	MEM_INST_RETIRED:LOADS *(r50010b:u)*
Retired Stores	INST_RETIRED:STORES *(r5002c0:u)*	MEM_INST_RETIRED:STORES *(r50020b:u)*
Multiplies	MUL *(r510012:u)*	ARITH:MUL *(r500214:u)*
Divides	DIV *(r510013:u)*	ARITH:DIV *(r1d40114:u)*
FP	FP_COMP_OPS_EXE *(r500010:u)*	FP_COMP_OPS_EXE:X87 *(r500110:u)*
SSE	SIMD_INSTR_RETIRED *(r5000ce:u)*	FP_COMP_OPS_EXE:SSE_FP *(r500410:u)*
Retired Uops	UOPS_RETIRED *(r500fc2:u)*	UOPS_RETIRED:ANY *(r5001c2:u)*
Hardware Interrupts	HW_INT_RCV *(r5000c8:u)*	HW_INT:RCV *(r50011d:u)*



Results

	Atom	Core2	Nehalem Nehalem-EX	Westmere-EX	SandyBridge-EP IvyBridge	Pentium D	Phenom Istanbul Bobcat
Total Instructions	hpEF	hpEF	hpEF	hpEF	hpEF	hpEFD	hpEFD
Total Branches	hp	hpD	hp	hp	hp	hp	hp
Conditional Branches	–	p	D	**DETRM**	**DETRM**	!	–
Loads	–	hpD	hpM	hp	U	hpU	–
Stores	–	**DETRM**	hpD	hpD	U	hpU	–

Sources of nondeterminism:	**h**	Hardware Interrupts
	p	Page Faults
Sources of overcount:	**E**	x87/SSE exceptions
	F	OS Lazy FP handling
	D	Instructions Overcounted
	M	Instructions Undercounted
	U	Counts micro-ops
Missing Results:	–	Event not available
	!	Test not run



Sources of Non-Determinism

- Hardware interrupts – most events increment an extra time for every hardware interrupt (most common is periodic timer)
- Page faults – first time memory page accessed, extra instruction

Overcount

- In addition to non-determinism, many events suffer from over (or under) count where an instruction triggers multiple times
- Overcount *is* deterministic, but cannot be predicted in advance unless you know exact dynamic instruction mix

Sources of Overcount Found on Most Events

- x87 top-of-stack pointer overflows
- Floating point unit used first time
- `rep`-prefixed string instructions count as single instruction (DBI tools count each, Pin behavior change)

Overcount in Total Retired Instructions

- AMD – `fninit`, `fnsave`, `fnclex` overcount when x87 exception flags set

- Pentium D – two different events
  `INSTRUCTIONS_COMPLETED:NBOGUS`
  `INSTRUCTIONS_RETIRED:NBOGUSNTAG`
  Latter is deterministic (except when interrupt rep string) but has overcount, specifically `fldcw` which can cause 2% error on some SPEC2k benchmarks.

Overcount in Retired Branches

- AMD – Linux kernel / perf_event issue: wrong event definition until Linux 2.6.35
- Core2 – `cpuid` instruction counts as a branch

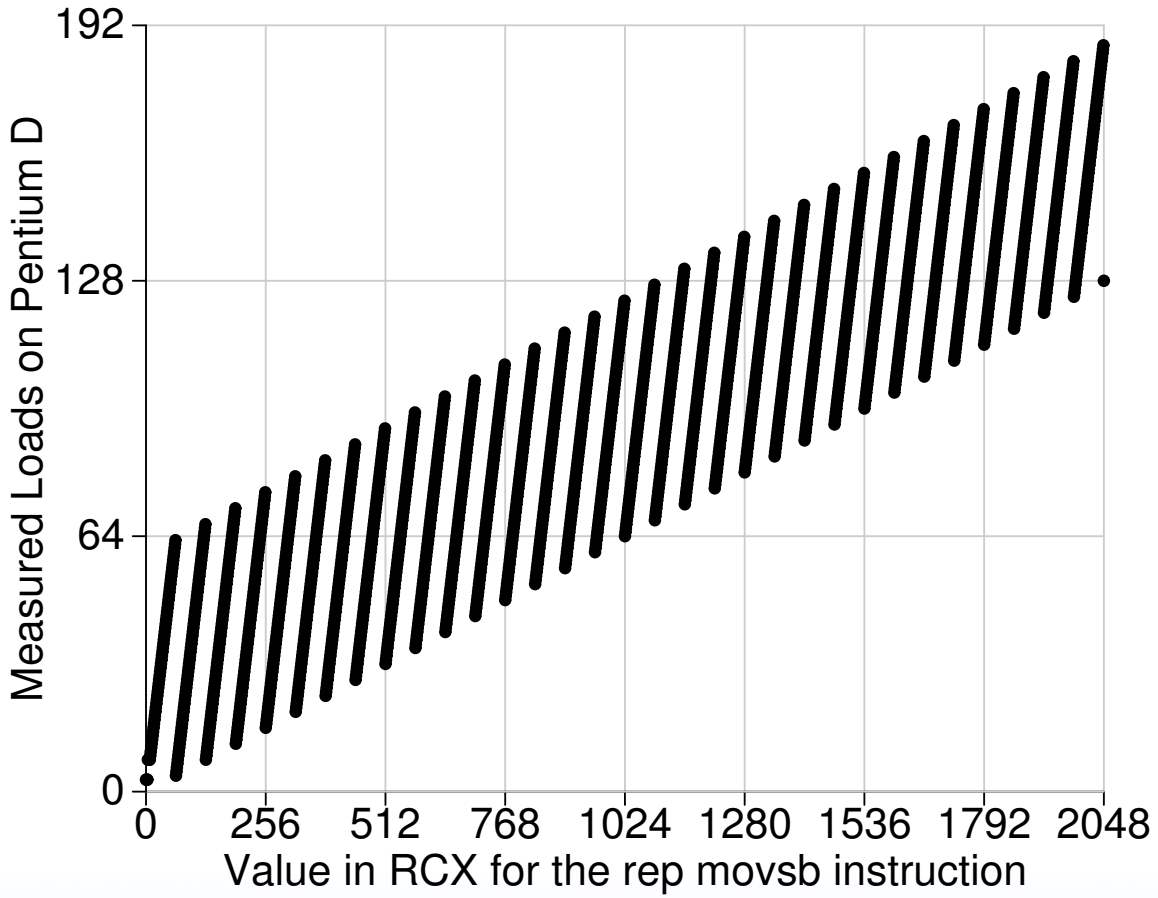
Overcount in Retired Conditional Branches

- Nehalem – overcounts for may instructions that start with opcode `0f` (cond branches but also some MMX and SSE)

Overcount in Retired Loads

- Core2 – `leave` counts twice.
  `fstenv`, `fxsave`, `fsave` count as loads.
  `maskmovq`, `maskmovdqu`, `movups`, `movupd`, `movdqu` count even when a store to memory.
- Nehalem – `paddb`, `paddw`, `paddd` *under* count
- Pentium D, SandyBridge, IvyBridge – measure μops

Complex Pentium D Behavior



Overcount in Retired Stores

- Nehalem, Westmere – `cpuid`, `sfence`, `mfence`, `clflush` all count as stores
- SandyBridge, IvyBridge – measure μops

Summary

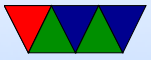
Only two known x86_64 events are deterministic with no overcount:

- `INST_RETIRED:STORES` on Core2
- `BR_INST_RETIRED:CONDITIONAL` on Westmere, SandyBridge and IvyBridge

Compensating for Non-Determinism

Is it possible to compensate for non-determinism?

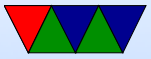
- For total aggregate counts, can subtract off interrupt counts (if a HW Interrupt event available)
- Sampling and Fast-forwarding a bit trickier.
  Can use ReVirt methodology: set counter to overflow early by a safe amount, compensate, then single-step to get exact



Compensating for Overcount

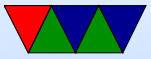
Is it possible to compensate for non-determinism?

- Difficult for aggregate counts; you need to know the exact instruction mix
- FastForward is easier, as if you are trying to get to the same place you will have traversed the same instruction mix and have the same overcounts



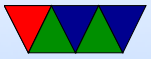
Non-x86_64 Architectures

- ARM – can't measure userspace only on Cortex A8/A9
- ia64 – `STORES_RETIRED`, `LOADS_RETIRED`, and `IA64_INST_RETIRED` appear deterministic
- POWER6 – `instructions:u` deterministic, `branches:u` has overcount
- SPARC Niagara T-1 – `INSTR_CNT` deterministic



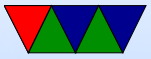
Full-sized Benchmarks (SPEC CPU 2000)

Benchmark	Pin Results	Counter Results	Difference
164.gzip.graphic	9,220,255,442+/-0	9,220,318,816+/-1	63,374
171.swim	18,657,590,092+/-0	18,657,604,499+/-0	14,407
175.vpr.place	10,506,996,023+/-0	10,507,367,334+/-1	371,311
175.vpr.route	8,498,211,242+/-0	8,498,625,210+/-1	413,968
176.gcc.200	10,809,876,957+/-0	10,810,247,099+/-14	370,142
177.mesa	35,256,814,647+/-0	35,256,814,675+/-0	28
178.galgel	25,736,467,292+/-0	25,736,468,525+/-0	1,233
179.art.110	3,467,916,650+/-0	3,467,916,650+/-0	0
179.art.470	3,792,351,365+/-0	3,792,351,365+/-0	0
186.crafty	14,715,329,050+/-0	14,715,329,550+/-0	500
187.facerec	17,108,726,507+/-0	17,175,891,130+/-6	67,164,623
188.ammp	31,435,756,072+/-0	31,435,756,072+/-0	0
197.parser	32,254,247,249+/-0	32,254,090,688+/-0	-156,561
200.sixtrack	24,831,293,048+/-0	24,831,447,915+/-1	154,867
252.eon.cook	9,168,538,965+/-10	9,168,538,925+/-21	-40
253.perlbmk.957	853,729,475+/-0	853,824,516+/-0	95,041
253.perlbmk.diffmail	5,192,919,547+/-2	5,192,873,218+/-0	-46,329
253.perlbmk.makerand	188,774,998+/-2	188,774,884+/-1	-114
253.perlbmk.perfect	3,498,063,997+/-2	3,498,435,094+/-0	371,097
254.gap	25,380,689,015+/-0	25,380,688,751+/-0	-264



Future Work

- Test more extensively on other architectures
- Auto-generate tests
- Work with chip vendors
- Look at more events and options (Fixed Counter 2)



Questions?

vincent.weaver@maine.edu

All code and data is available

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

git://github.com/deater/deterministic.git