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:

```assembly
# 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 $2 + (2\times r5301cb:u)$? 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

<table>
<thead>
<tr>
<th>Processor</th>
<th>Linux Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Atom 230</td>
<td>3.2 perf events</td>
</tr>
<tr>
<td>Intel Core2 X5355</td>
<td>2.6.36.2 perf events</td>
</tr>
<tr>
<td>Intel Nehalem X5570</td>
<td>2.6.38.6 perf events</td>
</tr>
<tr>
<td>Intel Nehalem-EX X7550</td>
<td>2.6.32-RHEL6 perf events</td>
</tr>
<tr>
<td>Intel Westmere-EX 8870</td>
<td>3.2 perf events</td>
</tr>
<tr>
<td>Intel SandyBridge-EP</td>
<td>2.6.32-RHEL6 perf events</td>
</tr>
<tr>
<td>Intel IvyBridge i5-3427U</td>
<td>3.2 perf events</td>
</tr>
<tr>
<td>Intel Pentium D</td>
<td>2.6.28 perfmon2</td>
</tr>
<tr>
<td>AMD Phenom 9500</td>
<td>2.6.29 perfmon2</td>
</tr>
<tr>
<td>AMD Istanbul 8439</td>
<td>2.6.35 perf events</td>
</tr>
<tr>
<td>AMD Bobcat E-350</td>
<td>3.2 perf events</td>
</tr>
</tbody>
</table>
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.)
<table>
<thead>
<tr>
<th></th>
<th>Intel Core2</th>
<th>Intel Nehalem / Westmere</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retired Instructions</td>
<td>INSTRUCTIONS_RETIRED (instructions:u)</td>
<td>INSTRUCTIONS_RETIRED (instructions:u)</td>
</tr>
<tr>
<td>Retired Branches</td>
<td>BRANCH_INSTRUCTIONS_RETIRED (branches:u)</td>
<td>BRANCH_INSTRUCTIONS_RETIRED (branches:u)</td>
</tr>
<tr>
<td>Retired Cond Branches</td>
<td>BR_CND_EXEC (r53008b:u)</td>
<td>BR_INST_RETIRED:CONDITIONAL (r5301c4:u)</td>
</tr>
<tr>
<td>Retired Loads</td>
<td>INST_RETIRED:LOADS (r5001c0:u)</td>
<td>MEM_INST_RETIRED:LOADS (r50010b:u)</td>
</tr>
<tr>
<td>Retired Stores</td>
<td>INST_RETIRED:STORES (r5002c0:u)</td>
<td>MEM_INST_RETIRED:STORES (r50020b:u)</td>
</tr>
<tr>
<td>Multiplies</td>
<td>MUL (r510012:u)</td>
<td>ARITH:MUL (r500214:u)</td>
</tr>
<tr>
<td>Divides</td>
<td>DIV (r510013:u)</td>
<td>ARITH:DIV (r1d40114:u)</td>
</tr>
<tr>
<td>FP</td>
<td>FP_COMP_OPS_EXE (r500010:u)</td>
<td>FP_COMP_OPS_EXE:X87 (r500110:u)</td>
</tr>
<tr>
<td>SSE</td>
<td>SIMD_INSTR_RETIRED (r5000ce:u)</td>
<td>FP_COMP_OPS_EXE:SSE_FP (r500410:u)</td>
</tr>
<tr>
<td>Retired Uops</td>
<td>UOPS_RETIRED (r500fc2:u)</td>
<td>UOPS_RETIRED:ANY (r5001c2:u)</td>
</tr>
<tr>
<td>Hardware Interrupts</td>
<td>HW_INT_RCV (r5000c8:u)</td>
<td>HW_INT:RCV (r50011d:u)</td>
</tr>
</tbody>
</table>
## Results

<table>
<thead>
<tr>
<th></th>
<th>Atom</th>
<th>Core2</th>
<th>Nehalem Nehalem-EX</th>
<th>Westmere-EX</th>
<th>SandyBridge EP IvyBridge</th>
<th>Pentium D</th>
<th>Phenom Istanbul Bobcat</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total Instructions</strong></td>
<td>hpEF</td>
<td>hpEF</td>
<td>hpEF</td>
<td>hpEF</td>
<td>hpEF</td>
<td>hpEFD</td>
<td>hpEFD</td>
</tr>
<tr>
<td><strong>Total Branches</strong></td>
<td>hp</td>
<td>hpD</td>
<td>hp</td>
<td>hp</td>
<td>hp</td>
<td>hp</td>
<td>hp</td>
</tr>
<tr>
<td><strong>Conditional Branches</strong></td>
<td>–</td>
<td>p</td>
<td>D</td>
<td>DETRM</td>
<td>DETRM</td>
<td>!</td>
<td>–</td>
</tr>
<tr>
<td><strong>Loads</strong></td>
<td>–</td>
<td>hpD</td>
<td>hpM</td>
<td>hp</td>
<td>U</td>
<td>hpU</td>
<td>–</td>
</tr>
<tr>
<td><strong>Stores</strong></td>
<td>–</td>
<td>DETRM</td>
<td>hpD</td>
<td>hpD</td>
<td>U</td>
<td>hpU</td>
<td>–</td>
</tr>
</tbody>
</table>

**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

Value in RCX for the rep movsb instruction

Measured Loads on Pentium D
Overcount in Retired Stores

- Nehalem, Westmere – cpuid, sfence, mfence, clflush all count as stores

- SandyBridge, IvyBridge – measure $\mu$ops
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)

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