Performance Counter

Non-Uniform Memory Access Seminar
Karsten Tausche
2014-12-10
Performance Counter

Hardware Unit for event measurements

“Performance Monitoring Unit” (PMU)

Originally for CPU-Debugging

used by manufacturers

Model Specific Register (MSR)

Program and access PMUs
Categories

Total instruction count
Branch instruction (total/conditional)
Load and Store instructions
Arithmetic instructions
Cache events
Uncore events
Motivation

Analyze CPU behavior on instruction level
   High Performance Computing
   CPU simulators
   Embedded

Unmodified Code
   Does not effect cache contents
   Performance Counter vs. Instrumentation

No performance impact with active PMUs
Outline

- Accessing PMUs
- Performance Counters on Intel Xeon
  - Definitions
  - Core/Uncore events
- Accuracy
  - Analysis
  - Dependencies
  - Reducing Inaccuracies
- Tools
  - Intel Performance Counter Monitor
  - Perf
  - PAPI
  - Intel Perf
Counter Access

Select counted events
- Either fixed function PMU
- Or multiple countable events per PMU

Configure counters
- Start/stop
- Count in kernel/user-mode
- Read/write values

Programming via MSR bitfields
- CPU model specific field definitions
- MSRs accessible only in kernel mode
Counter Access

PMU event counting per core/hyper thread

Kernel mode driver and user space library/tool

Counter per software thread
Generalized PMU programming with event IDs
Access in user mode, without root privileges
[Abstraction from platform/operating system]
Performance Counter on Intel Xeon

“instructions_retired”

Executed by speculative out-of-order pipeline

Handled by Retirement Unit

Results visible to user
"Uncore" per socket resources

"Box" modular uncore unit
Core Performance Counter on Intel Xeon

Instructions counts total, branches, arithmetic, ...
Core Performance Counter on Intel Xeon

Intel Turbo Boost frequencies
Core Performance Counter on Intel Xeon

L1, L2 Cache hits and misses
Uncore Performance Counter on Intel Xeon

Last Level Cache (shared L3 cache) hits and misses
Uncore Performance Counter on Intel Xeon

Home Agent memory controller and cache coherency memory read/write, local/remote, conflicts, directory/snooping

Pbox Physical connection between cores or sockets
Integrated Memory Controller

**DRAM access**
read/write queues, ECC correctable errors, refreshes, thermal throttling
Uncore Performance Counter on Intel Xeon

QPI
Ring ↔ Link Layer (socket interconnect)

Filter event counts:
physical address,
Home Node ID,
instruction
Uncore Performance Counter on Intel Xeon

QPI
Ring ↔ Link Layer
(socket interconnect)
link speed, transfers, total link utilization
Uncore Performance Counter on Intel Xeon

Power Controller Unit
Socket energy usage
DRAM energy usage
Uncore Performance Counter on Intel Xeon

Power Controller Unit
per core
temperature
time spent in power states
Performance Counters per Box

- **QPI:** 4 per port + 3
- **PCU:** 4
- **Ubox (system config):** 2
- **PCIe:** 4
- **CBox:** 4
- **Home Agent:** 4
- **iMC:** 4 per channel
Performance Counters on Intel Xeon

ubuntu-numa0101.fsoc: Linux 3.13, 2x Intel Xeon E5-2620

• 57 (+x) Performance Monitoring Units per socket
• 634 countable events
• Allowing comprehensive runtime analysis
• Mostly focused on a few context specific events
Accuracy

Not defined/guaranteed by manufacturer
At least not for Intel/AMD

Speculative architecture
Out-of-order pipeline, serving multiple computing units
Branch prediction

Hardware parallelization
CPU behavior depending on timings, cache-contents, etc.
Accuracy Analysis

[Weaver2013]

Non-determinism

Identical run, different result

Overcount

Same (wrong) result for identical runs
Accuracy Analysis

[Weaver2013]
One deterministic event without overcount on Sandy Bridge EP: 
\texttt{BR\_INST\_RETIRED\_CONDITIONAL} 
(executed conditional branch instructions)

PMU hardly usable for deterministic implementations
  
Deterministic replay
Deterministic threading libraries
CPU simulators

Don’t use micro-operation-counter
  
System specific, undocumented low-level instructions in CISC processors
Inaccuracy Sources

[Weaver2013]

Hardware Interrupts: increment most events

- Nondeterministic overcount

Wrong/unintuitive counter behavior

- Floating point instructions with “wait for exception” – count twice?
- Count µOPs instead of retired instructions (e.g., load/store events)

Accessing counters

- Requires system call (interrupt, context switch)
Reduce Inaccuracies

Carefully controlled test environment
- Kernel version, tool versions
- Running processes
- BIOS/Power saving settings

Compiler/Runtime configuration
- E.g., Address space layout randomization
Reduce Inaccuracies

Prefer low-level APIs
  Prefer dynamic library over command line tool

Compare tools/libraries

[Read papers]

[Check errors with well known Assembly]
Error Rates

Counted instructions in the micro benchmark use by [Weaver2013]

Integer divides
- Nehalem: 11.2%
- Nehalem-EX: 1.1%
- Sandy Bridge EP: n/a
- Ivy Bridge: 2.8%

Floating point instructions
- Nehalem: 0.0%
- Nehalem EX 0.0%
- Sandy Bridge EP: 0.003%
- Ivy Bridge: 0.08%
Using Performance Counters

Use Profilers first

Using automated tests
Partly implemented with Performance Counter
Optimize as much as possible

Low level analysis with Performance Counters

Optimize problematic code sections
Platform specific optimization
Benefit from minimal overhead
Tools and Libraries

- Intel Performance Counter Monitor
- Perfmon/libpfm
- Linux Perf
- PAPI
Intel Performance Counter Monitor

Tools and C++ programming interface
Full support for Intel core/uncore events
Supports newer Intel Xeon, Core i, Atom

Uncore mainly available on server platforms
Intel Performance Counter Monitor

Provided by Intel as source code

Driver; GUI/command line tools; C++ library

Linux

build upon MSR kernel module
KDE: ksysguard plug-in

Windows

compile/modify sample driver
Perfmon plug-in
Intel Performance Counter Monitor
Ksysguard (KDE System Monitor)
Intel Performance Counter Monitor

Windows Performance Monitor

[Image of a graph showing performance metrics]

2014-12-10  NUMA Seminar - Karsten Tausche  34
perf – “Performance Counters for Linux”

Part of Linux since v2.6.31 (2009)
before: patch and compile your kernel

Command-line tool “perf” (userspace)
Debian/Ubuntu-Package “linux-tools”

**perf list**
Detects supported events
But no support for finding relevant events

**perf stat -e [eventName] command**
Run command and count eventName
Linux: perfmon2 / libpfm

Originated from own kernel subsystem

Kernel driver: perfmon2, userspace library: libpfm

Superseded by Linux’ perf_events interface

Part of kernel since v2.6.31, 2009

Libpfm3 – IA64-subsystem in Linux by HP

Libpfm4 – complete rewrite by Google
Linux: libpfm4

Using Linux perf_events interface

libpfm4
  Retrieve supported events per source
  Translating event IDs and names
  Program events

Architecture support
  Intel x86: since Pentium P6, Core Duo/Solo, Atom, Nehalem
  AMD64 x86: K7, K8 and newer; uncore since Bulldozer
  Some ARM, SPARC, IBM Power, MIPS models
Libpfm4 on ubuntu-numa0101.fsoc

Source code: examples and tools

showevtinfo (example tool)

Lists supported and detected PMUs
Xeon E5-2620 (Sandy Bridge EP):
Lists 4196 available events, 634 supported
Per event: PMU, index, parameters, description
Libpfm4 on ubuntu-numa0101.fsoc
showeventinfo

IDX : 37748741
PMU name : ix86arch (Intel X86 architectural PMU)
Name : BRANCH_INSTRUCTIONS RETIRED
Equiv : None
Flags : None
Desc : count branch instructions at retirement. Specifically, this event counts the retirement of the last micro-op of a branch instruction
Code : 0xc4
Modif-00 : 0x00 : PMU : [k] : monitor at priv level 0 (boolean)
Modif-01 : 0x01 : PMU : [u] : monitor at priv level 1, 2, 3 (boolean)
Modif-02 : 0x02 : PMU : [e] : edge level (may require counter-mask >= 1) (boolean)
Modif-03 : 0x03 : PMU : [i] : invert (boolean)
Modif-04 : 0x04 : PMU : [c] : counter-mask in range [0-255] (integer)
Modif-05 : 0x05 : PMU : [t] : measure any thread (boolean)
PAPI

Performance Application Programming Interface

Platform independent PMU interface

- Provide standard definitions for performance metrics
- “easy to use, well documented, freely available”

Windows support discontinued after XP

Preset Events

- Supported across [nearly] all platforms

High Level API

- Simplified access to Preset Events

Low Level API

- Adds access to native events
PAPI
Performance Application Programming Interface

High Level API: core events only
Uncore still requiring Low Level API

papi_avail
Check available events

papi_event_chooser
List events that can be combined with a given list

papi_native_avail
Detailed information about native events
Demo

**perf**

- **perf stat -e instructions:u -e cache-misses:u ./cache-miss**
  
  count total instructions and cache-misses, both in user mode

- **perf stat -r 10 ...**
  
  run command 10 times, print average and stddev

**libpfm4 examples in libpfm/libpfm-4.5.0/examples**

- **./showevtinfo | less**
  
  lists first PMUs supported by libpfm and detected on your system
  also lists all supported events with description and parameters
Demo sources

/NUMASem/demo_PerformanceCounter/

- cache-miss / cache-miss2
  use perf stat to check cache misses with different array iteration patterns

- libpfm_cache-miss
  Uses libpfm4 library calls to count cache events
  List of measured events is defined by “eventNames” string vector
  This is a simplified version of libpfm examples:
  /NUMASem/libpfm-4.5.0/perf_examples/self.c

- libpfm_qpi_remote
  Uses libpfm4 to count QPI events and libnuma to pin the task and allocated memory to different sockets.
  See line 75 to switch between allocating memory on local or remote socket.
Sources

• Weaver, V.M.; Terpstra, D.; Moore, S. “Non-determinism and overcount on modern hardware performance counter implementations”, 2013
• http://perfmon2.sourceforge.net/
• http://icl.cs.utk.edu/projects/papi/wiki/Main_Page
• Linux man pages