How to Write Fast Numerical Code

Spring 2014

Lecture: Performance Counters and applying the Roofline Model

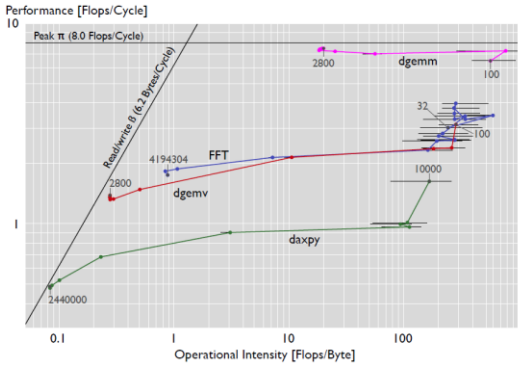
Instructor: Markus Püschel

TA: Daniele Spampinato & Alen Stojanov

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Roofline Measurements

Core i7 Sandy Bridge, 6 cores
*Code: Intel MKL, **sequential***
Warm cache



- **What quantities are needed?**
  - Work (W)
  - Runtime (T)
  - Memory Traffic (Q)

How can we measure them?

Performance Counters

```
ReadCounter(start);

/* Sum two arrays */
for(i = 0; i < num_runs; i++)
  z[i] = x[i] + y[i];

ReadCounter(end);
```

#counted Events = end - start

- **All modern processors include performance counters**
  - Intel Pentium Pro – Intel i3/5/7
  - AMD K7 and AMD AMD64
  - IBM PPC970, PPC970MP, POWER4+, IBM Cell processors (incl. Sony PS3)
  - MIPS: 5K, 20K, 25KF, 34K, 5KC, 74K, …..
  - ARM Cortex
  - ….

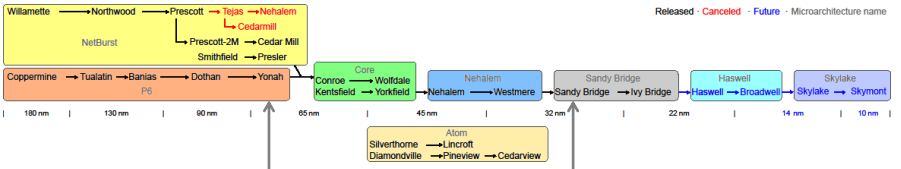
Performance Monitor Unit (PMU)

- **CPU unit capable of collecting microarchitectural events:**
  - Cycles, issued/retired instructions, cache misses, …
- **CPU-specific implementation, e.g.:**
  - Intel reference: *Intel 64 and IA-32 Architectures Software Developer Manuals*, Vol. 3B, Ch. 18-19
  - ARMv7 reference: *ARM Architecture Reference Manual - ARMv7-A and ARMv7-R edition*, Ch. C12
- **Normally composed of set of registers for counting and control**
- **Encoded events can be architectural or non-architectural**

Table 19-2. Non-Architectural Performance Events In the Processor Core of 4th Generation Intel® Core™ Processors

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
03H	02H	LD_BLOCKS.STORE_FORWARD	loads blocked by overlapping with store buffer that cannot be forwarded .	
03H	08H	LD_BLOCKS.NO_SR	The number of times that split load operations are	

Intel: PMUs evolution



Performance Monitoring version 1

- *2 programmable Counters per Core*
- *3 fixed Counters per Core*
- *40 bit width*
- *System Wide Counting*

Performance Monitoring version 3

- *8 programmable Counters per Core*
- *3 fixed Counters per Core*
- *2 programmable Counters for LLC Communication per Core*
- *2 programmable Counters Uncore*
- *1 fixed Counter Uncore*
- *48 bit width*
- *per HW Thread Counting*
- *Precise Event Based Sampling*

Intel: Accessing the Counters

- **Performance Monitoring v1-v3 defines how to program the counters**
- **Counters differ between microarchitectures**
- **To access directly**
  - Acquire root somehow
  - Disable counter in control Machine Specific Register (MSR)
  - Program events and behaviour you like in configuartion MSR
  - Enable counters in control and configuartion MSR
  - Check overflow MSR / read value from counter MSR

Intel: Accessing the Counters

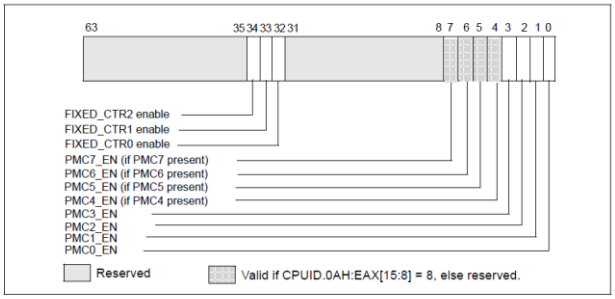


Figure 18-26. IA32_PERF_GLOBAL_CTRL MSR in Intel® Microarchitecture Code Name Sandy Bridge

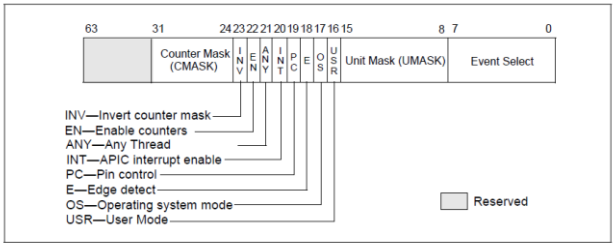
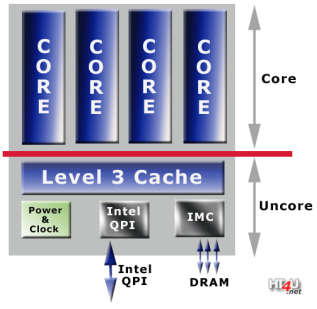


Figure 18-6. Layout of IA32_PERFEVTSELx MSRs Supporting Architectural Performance Monitoring Version 3

Intel: Types of Counters

- **Fixed function counters**
  - Predefined events that are commonly used
  - TSC, instructions retired, core clock cycles, …
- **General purpose performance counters**
  - can be programmed to follow a specific event



Intel: Types of Counters

- **Fixed function counters**
  - Predefined events that are commonly used
  - TSC, instructions retired, core clock cycles, …
- **General purpose performance counters**
  - can be programmed to follow a specific event
- **Precise Event-Based Sampling (PEBS)**
  - Can keep track of architectural state right after instruction causes event
  - Can trigger interrupt (PMI) coupled to counter

ARMv7: PMUs

- **ARM defines two performance monitors extensions: PMUv1 and PMUv2**
- **Basic form must provide a coprocessor interface (CP15) with:**
  - A 32 bit cycle counter (CCNT)
  - Up to 31 programmable counters
  - Control registers
- **ARM doesn't define a degree of inaccuracy for the counters**
  - Mainly to keep implementation and validation cost low
- **Example reading from CCNT**

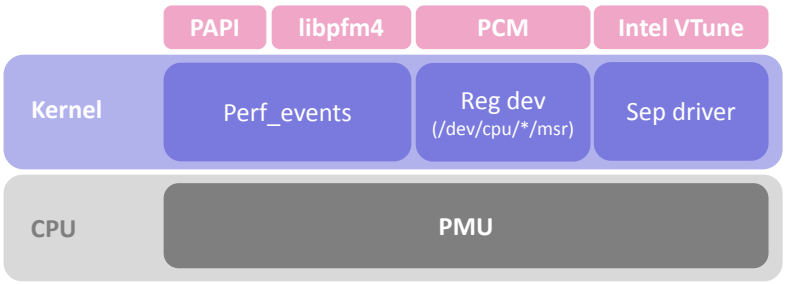
```
inline INT32 read_ccnt() {
        INT32 cycles;
        ASM VOLATILE ("MRC p15, 0, %0, c9, c13, 0\n\t" : "=r"
(cycles));
        return cycles;
}
```

Software Interfaces and Tools

- **Kernel interfaces, e.g., on Linux**
  - Perfmon2 (<= 2.6.30)
  - Perf_events (>= 2.6.32)
- **Machine independent APIs**
  - PAPI (relys on kernel interfaces of targeted Oss)
- **Vendor specific, e.g., Intel**
  - Vtune (Sampling based, uses its own drivers)
  - Intel PCM (Cross OS, direct access to MSRs)

Software Interfaces and Tools

Example of software stack on Intel platform



Perfplot

- **Tool to ease the effort of creating performance / roofline plots**
- **Modified Intel PCM to allow start / stop measurements**

```
measurement_init(counters); //Array with Mask/Eventnr

for(r = 0; r < nr_repeats; r++){
        measurement_start();
        /* Code to be measured */
        for(i = 0; i < n; i++)
          z[i] = x[i] + y[i];
        measurement_stop();
}

measurement_end(); //Dump results to files
```

- **Instrument your code as depicted and link with the modified PCM**

Perfplot

- **Collaboration between**
  - Georg Ofenbeck
  - Ruedi Steinman
  - Victoria Caparros Cabezas
  - Daniele Spampinato
- **Available at https://github.com/GeorgOfenbeck/perfplot**
- **Scala scripts to automate**
  - Compilation and execution in temporary directories
  - Retrieving the results and collecting them for plots
- **Python plot scripts for**
  - Performance plots
  - Roofline plots

Timing on Intel: Time Step Counter

```
CPUID();
RDTSC(start);

/* Sum two arrays */
for(i = 0; i < num_runs; i++)
  z[i] = x[i] + y[i];

RDTSC(end);
CPUID();
```

#cycles = end - start

- **"Read Time Step Counter" instruction to read Invariant TSC**
- **Monotonically increasing counter, wrap around > 10y**
- **Stored in 64-bit IA32_TIME_STAMP_COUNTER MSR**
- **Easily accessable counter (dedicated instruction, user mode)**
- **Starting from Nehalem RDTSCP**

Timing on ARM: Cycle Counter

- **Stored in 32-bit CCNT register on CP15**
- **Can count every cycle or every 64th cycle**
- **Could be used thru interfaces (e.g., perf_events)**
- **On some implementation high-overhead of interfaces requires direct access**
- **Direct access normally requires:**

```
reset_ccnt();
reset_overflow_flags();
enable_ccnt();

/* Computation */

disable_ccnt();
cycles = read_ccnt();
/* check for overflow */
if (counting_every_64) cycles = cycles << 6;
```

Caveats

- **General**
  - Compiler optimizations
  - Asynchronous calls
- **Runtime**
  - Frequency scaling
  - TSC (or similar counters) is system-wide, everything is measured
  - Parallel scenario, reference cycles difficult to relate to wallclock time
- **Work**
  - Distinguishing single / double precision not necessary possible
- **Memory Traffic**
  - WB cache, prefetcher, …