Performance Counters

Georg Ofenbeck
Spring 2013
Read 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 a “Machine Specific Register” (MSR)

Easily access able counter (dedicated instruction, user mode)
Performance Counters

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
- ....

```c
ReadCounter(start);

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

ReadCounter(end);

#counted Events = end - start```

```c
ReadCounter(start);

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

ReadCounter(end);

#counted Events = end - start```
Performance Counters

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**
- ....

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

#counted Events = end - start
```
Types of Counters (Intel)

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*
Types of Counters (Intel)

Fixed function counters

- **Predefined events that are commonly used**
- **TSC, instructions retried, core clock cycles, ...**

General purpose performance counters

- **can be programmed to follow a specific event**

[Image: http://images.ht4u.net/reviews/2009/intel_lynnfield_core_i5_core_i7/coreUncoreNehalem.png]
Types of Counters (Intel)

Fixed function counters
- Predefined events that are commonly used
- TSC, instructions retried, core clock cycles, ...

General purpose performance counters
- can be programmed to follow a specific event

Precise-event based sampling
- Trigger interrupt coupled to counter
- Allows to e.g. trace memory access
Evolution of Performance Counters

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

- 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”
Accessing the Counters

Perfmon(1-3) defines how to program the counters

Counters differ between microarchitectures (and in-between)

To access directly

- Acquire root somehow (MSR access)
- Disable counter in control MSR
- Program events and behaviour you like in config MSR
- Enable counters in control and config MSR
- Check overflow MSR / read value from counter MSR
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
Tool for Counters

Intel VTune
- *Sampling based*

Perf, papi, libpfm4
- *Linux only, uncore poorly supported*

Intel PCM
- *Intel only, Cross OS, direct access to MSRs*
Caveats

Generally many, many, many things that can go wrong

- Example flop counter with perf

Sum Reduction

REPS=30 - .005 Threshold on 1st derivative
Caveats

General
- *Dead code elimination*, “smart” compiler, Initialization
- Asynchronous calls
- Alignment
- HW prefetcher

Timing
- *Frequency scaling*
- *per thread counters don’t capture total runtime*

Flops
- *Distinguishing single / double precision not necessary possible*

Memory
- *On desktop Intel machines not straightforward*
- *WB cache, prefetcher, ...*
Perfplot

Tool to ease the effort of creating performance / roofline plots

Modified Intel PCM to allow start / stop measurements

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

for(r = 0; r < nr_repeats; r++){
    measurement_start();
    /* Sum two arrays */
    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

In collaboration with

- 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