Software Performance Monitoring

Daniele Francesco Kruse

July 2010

Summary

1. Performance Monitoring
2. Performance Counters
3. *Core* and *Nehalem* PMUs – Overview
4. *Nehalem* : Overview of the architecture
5. μops flow in *Nehalem* pipeline
6. Perfmon2
7. Cycle Accounting Analysis
8. The 4-way Performance Monitoring
9. *PfmCodeAnalyser* : a new tool for fast monitoring
10. *David Levinthal* : the expert
11. Conclusions

Performance Monitoring

DEF : The action of collecting information related to how an application or system performs

HOW : Obtain micro-architectural level information from hardware performance counters

WHY : To identify bottlenecks, and possibly remove them in order to improve application performance

Performance Counters

- All recent processor architectures include a processor–specific **PMU**
- The **P**erformance **M**onitoring **U**nit contains several performance counters
- Performance counters are able to count micro-architectural events from many hardware sources (cpu pipeline, caches, bus, etc…)

Core and *Nehalem* PMUs - Overview

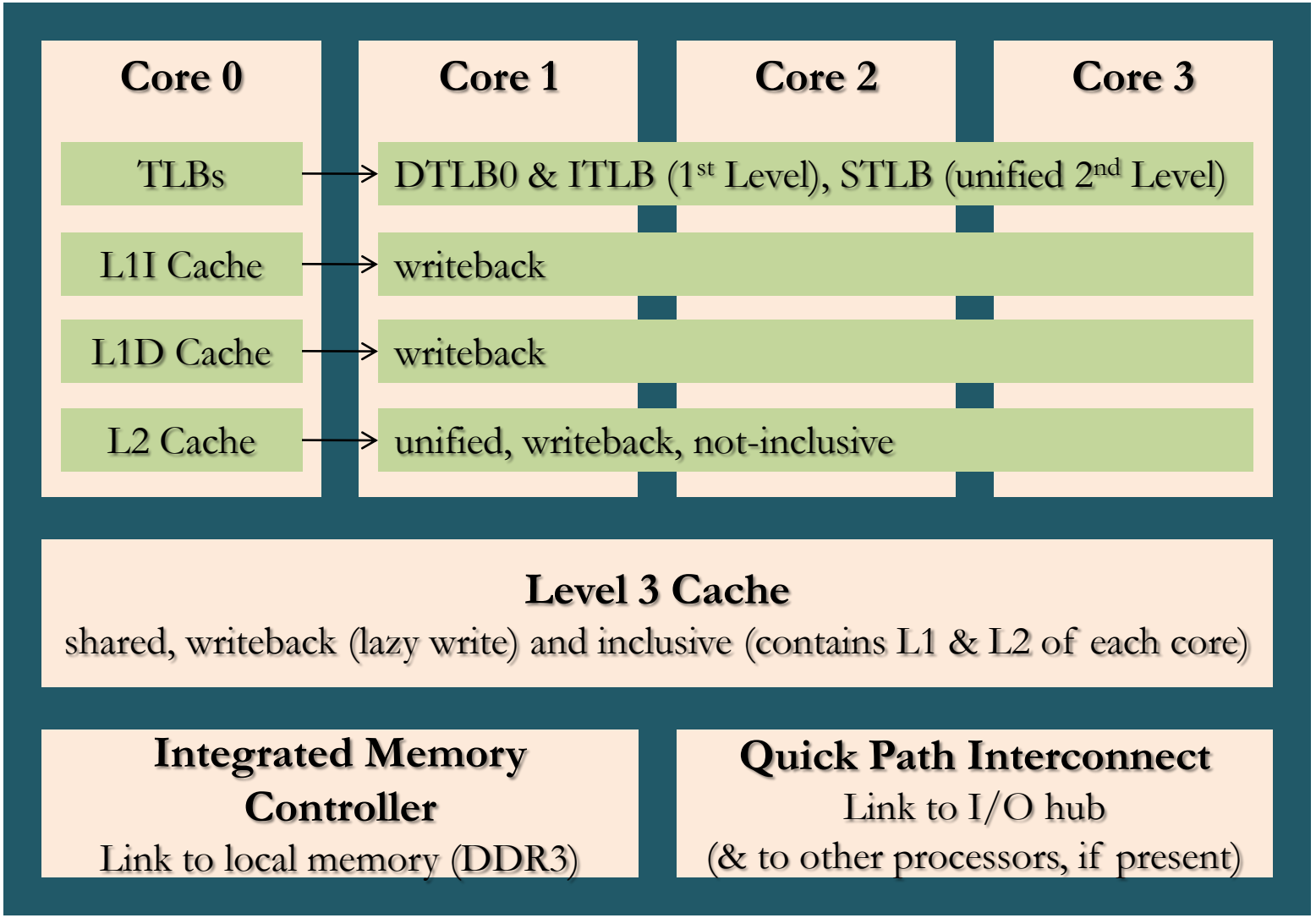
Intel *Core* Microarchitecture PMU

- 3 fixed counters
  (INSTRUCTIONS_RETIRED, UNHALTED_CORE_CYCLES, UNHALTED_REFERENCE_CYCLES)
- 2 programmable counters

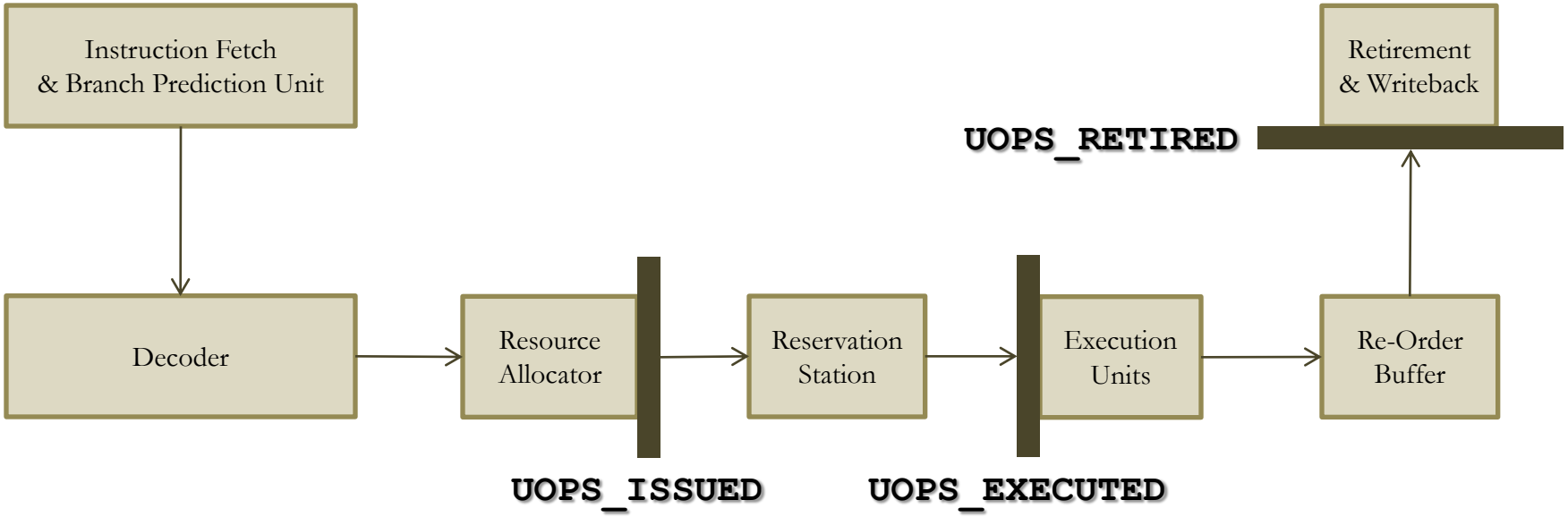
Intel *Nehalem* Microarchitecture PMU

- 3 fixed *core*-counters
  (INSTRUCTIONS_RETIRED, UNHALTED_CORE_CYCLES, UNHALTED_REFERENCE_CYCLES)
- 4 programmable *core*-counters
- 1 fixed *uncore*-counter (UNCORE_CLOCK_CYCLES)
- 8 programmable *uncore*-counters

Nehalem : Overview of the architecture



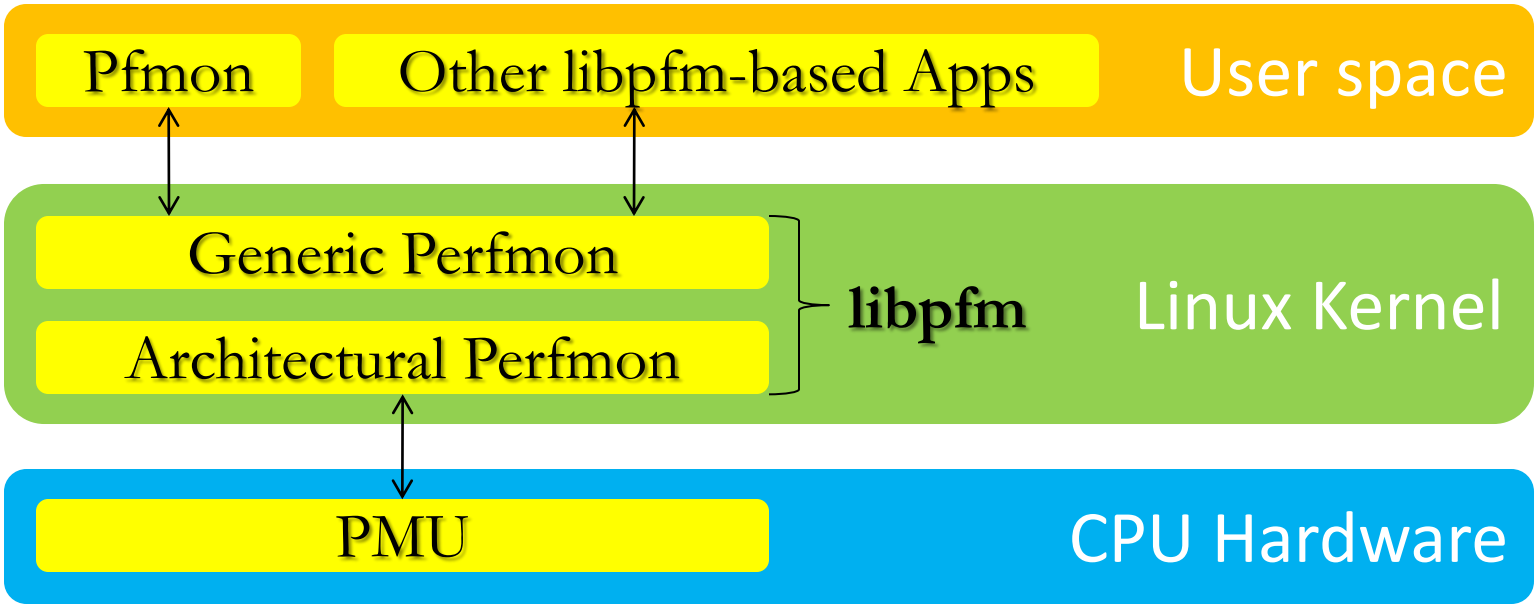
µops flow in *Nehalem* pipeline



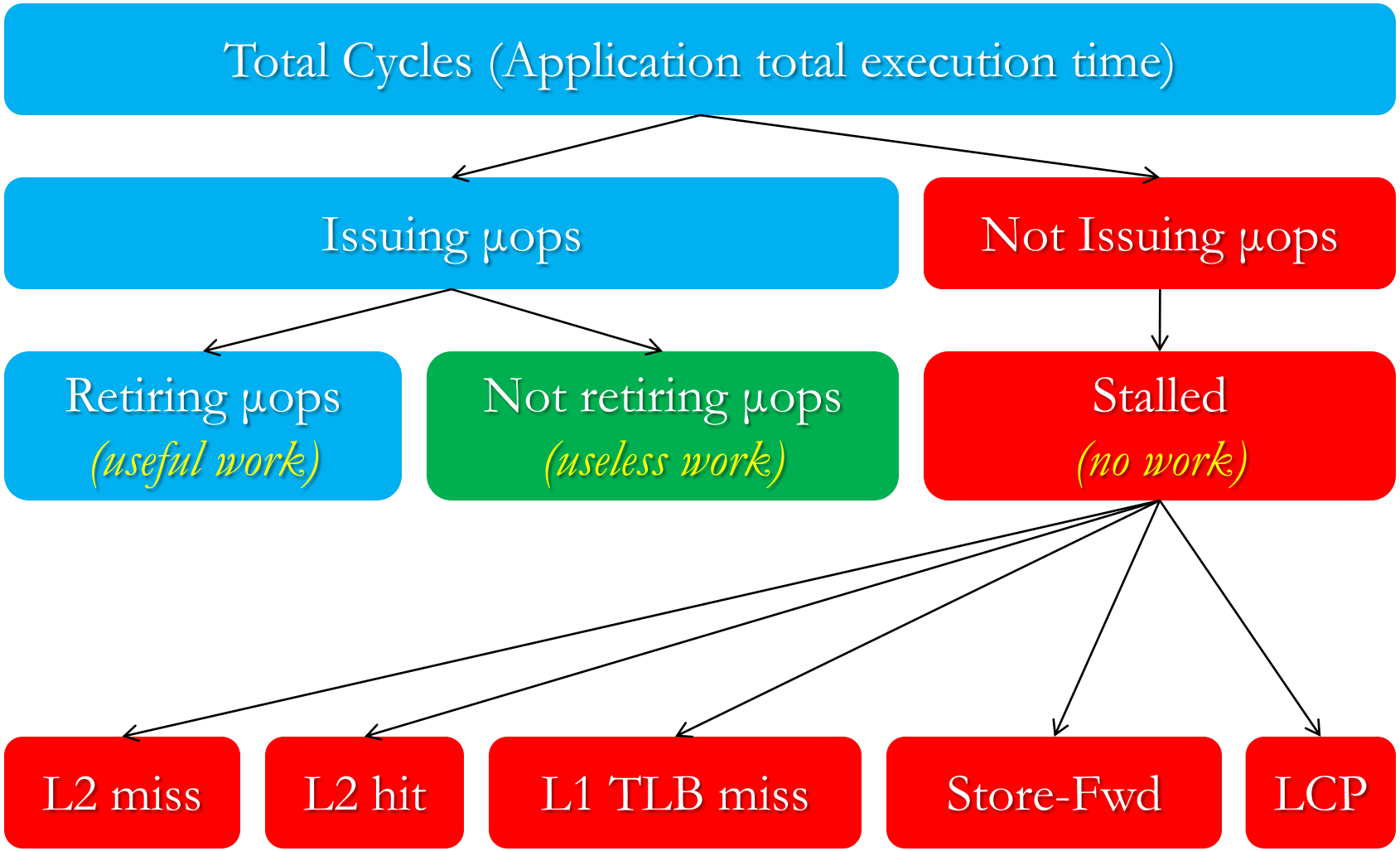
- We are mainly interested in `UOPS_EXECUTED` (dispatched) and `UOPS_RETIRED` (the useful ones).
- Mispredicted `UOPS_ISSUED` may be eliminated before being executed.

Perfmon2

- A generic API to access the PMU (`libpfm`)
- Developed by Stéphane Eranian
- Portable across all new processor micro-architectures
- Supports system-wide and per-thread monitoring
- Supports counting and sampling



Cycle Accounting Analysis



The 4-way Performance Monitoring

	Overall (pfmon)	*Modular*
Counting	1. Overall Analysis	3. Module Level Analysis
Sampling	2. Symbol Level Analysis	4. Modular Symbol Level Analysis

PfmCodeAnalyser : a new tool for fast monitoring

- Unreasonable (and useless) to run a complete analysis for every change in code
- Often interested in only small part of code and in one single event
- Solution: a fast, precise and light "singleton" class called *PfmCodeAnalyser*
- How to use it:

```
#include<PfmCodeAnalyser.h>

PfmCodeAnalyser::Instance("INSTRUCTIONS_RETIRED").start();

   //code to monitor

PfmCodeAnalyser::Instance().stop();
```

PfmCodeAnalyser : a new tool for fast monitoring

```
PfmCodeAnalyser::Instance("INSTRUCTIONS_RETIRED",  0, 0,
                          "UNHALTED_CORE_CYCLES",  0, 0,
                          "ARITH:CYCLES_DIV_BUSY", 0, 0,
                          "UOPS_RETIRED:ANY",      0, 0).start();
```

```
Event: INSTRUCTIONS_RETIRED
     Total count:105000018525
Number of counts:10
   Average count:10500001852.5

Event: UNHALTED_CORE_CYCLES
     Total count:56009070544
Number of counts:10
   Average count:5600907054.4

Event: ARITH:CYCLES_DIV_BUSY
     Total count:28000202972
Number of counts:10
   Average count:2800020297.2

Event: UOPS_RETIRED:ANY
     Total count:138003585913
Number of counts:10
   Average count:13800358591.3
```

David Levinthal : the expert

- Intel senior engineer specialized in performance monitoring of applications
- He is going to be here from the 13th to the 30th of July
- He will give a detailed lecture about performance monitoring on the 21st or 22nd of July in 513-1-024
- Anyone who's interested in partecipating may contact us for details

Conclusions

1. We gave a very brief introduction to Nehalem multicore architecture and to Performance Monitoring using hardware performance counters

2. A consistent set of events has been monitored across the 4 different analysis approaches in CMSSW, Gaudi and Geant4 (*Cycle Accounting Analysis*)

3. A monitoring tool has been developed for quick performance monitoring: *PfmCodeAnalyser*

4. The report and background of the work done for CMSSW is available at http://cern.ch/dkruse/pfmon.pdf

Questions ?