Hardware Performance Monitoring

Computer Systems Performance Modeling and Evaluation

Department of Electrical and Computer Engineering
Shiraz University
Fall 1388
Reza Azimi

The Problem

- The performance of a CPU is greatly affected by
  - its internal implementation.
    - examples: pipeline width, cache size, cache replacement policy, etc.
  - how a program use its resources.
    - examples:
      - Memory access pattern affecting cache performance
      - Instruction control flow affecting cache performance and branch prediction
      - The amount of inherent parallelism in the instructions affecting the utilization of functional units.

- We want to look inside a CPU as it operates
  - There are so many events occurring inside a CPU with substantial performance impact.
Solutions

- Simulation
  - Build a detailed and accurate model of the processor implementation and simulate the CPU by executing the model.
  - Advantage: can produce accurate and reliable information.
  - Disadvantage: VERY slow.
    - Accurate simulation of a 1-minute program could take days!

- Adding Performance Monitoring to CPU
  - CPU monitors its own behavior and reports it to software.

Performance Monitoring Unit (PMU)

- A component inside CPU
- Tight integration with CPU core => each CPU core has its own PMU
- Originally designed by computer hardware engineers for debugging CPUs
- Also known as Hardware Performance Counters (HPCs)
Why Using PMUs?

- Low Overhead
  - Hardware implementation

- Low Perturbation
  - PMU does not use any of the computational/storage resources needed for normal operation of the CPU (examples: registers, ALUs, etc.)

- High Resolution
  - Capable of monitoring detailed micro-architectural events that cannot be monitored without hardware support.

- Widespread
  - All major industrial processors have included PMUs in their design

Basic PMU Features

- A set of performance counters
  - CPU Registers
  - Can be programmed by user to count CPU events

- Data Monitors
  - Registers or buffers
  - Record memory addresses being used by the processor.

- Software Notification Mechanism
Performance Events

- Program Characterization Events
  - Examples: Instruction types, executed code paths

- Memory Access Events
  - Examples: accesses to different elements in the memory hierarchy (L1, L2, L3, or DRAM)

- Pipeline Utilization/Contention
  - Examples: whether CPU's floating-point units are fully utilized or not, how long does it take for an instruction to go through one stage of the pipeline.

CPU Arch.: AMD Opteron Core
CPU Arch.: Intel’s Nehalem

Hardware Performance Counters

- CPU Registers
  - Normally there are only a few them.
  - Usually 32-bit registers.

- Once programmed, count performance events in the CPU automatically, with no software interference.

- Their value can be read at any point during the execution.
PMU Notification Mechanism

- **Counter Overflow**
  - A performance counter can be initialized such that it overflows after every N occurrence of an event.

- **Overflow Exception**
  - When a counter overflows, the PMU will generate a software exception for CPU.
  - State of the execution at the time of exception is recorded and handed to the exception handler.

- **Exception Handler**
  - An exception handler (normally inside the OS kernel) is triggered.
  - The instruction/data that has caused the event counter to overflow is examined.

Programming PMUs: Summary

- **Set the Event Types**
  - What performance events to be monitored.

- **Sampling Frequency**
  - How often an overflow exception to be triggerd.

- **Monitoring Domain**
  - Only user level
  - Only inside the OS kernel
  - Both user and kernel
Acquiring PMU-generated Info.

- **Reading HPCs Directly**
  - Requires code instrumentation
    
    ```
    Count1 = Read (CYCLE_COUNTER);
    for (i = 0; i < N; i++) {
        compute();
    }
    Count2 = Read (CYCLE_COUNTER);
    CyclesSpent = Count2 - Count1;
    ```

- **Using Overflow Interrupts**
  - Set counter’s countdown value to N
  - That means there is one notification for every N occurrence of the event
  - Enables statistical sampling of the execution state

Case Study: CPI Breakdown

- **Challenge:**
  - It is hard to translate values of HPCs to their performance impact.

- **Example**
  - **Question:** how significant is 1 million L1 D Cache misses?
    
    It depends on many factors including
    - Degree of available Instruction-Level Parallelism (ILP) in the program
    - L1’s mechanisms for prefetching
    - L2 access latency
    - L2 bandwidth
    - Contention on L2’s ports (e.g., in a CMP with shared L2)
    - L2’s Hit Rate
Cycle-per Instruction Breakdown

- Main Focus: cycles where no instruction completes (stalls).

- Rationale
  - Most cycles are stalls.
    - 75% on average
  - CPI without stalls is almost constant.

Real CPI vs. Ideal CPI

<table>
<thead>
<tr>
<th>Cycles Per Instruction (CPI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ideal CPI</td>
</tr>
<tr>
<td>CPI without Stalls</td>
</tr>
<tr>
<td>Real CPI</td>
</tr>
</tbody>
</table>

Performance Modeling and Evaluation, Shiraz University, Fall 1388, by Reza Azimi
Stall Types

- No instruction to execute
  - Icache misses
  - branch mispredictions

- Required data is not available.
  - memory hierarchy delay
  - functional units delay

Stall Breakdown

- ICache Miss
  - No Instrs.
  - Branch Misprediction
  - Other

- DCache Miss
  - Load/Store Unit
  - Integer Unit
  - Floating Point Unit
  - Other

- LSU Latency / Others

- Completion Cycles
  - No Data
  - Completion Cycles
Stall Breakdown for FFT

Sources of Bottlenecks
- TLB misses
- FPU latency

Challenges of Using PMUs
- PMUs are specific to the CPU implementation
  - Understanding them requires deep knowledge of the architecture
  - Code that uses PMU features is not portable
- Inflexible
  - A fixed set of events can be monitored. New events cannot be added to the set after manufacturing the CPU.
  - Small number events can be monitored at a time.
Further Reading