Triggering Performance Counters for Energy Efficiency Measurements

Norbert Schmitt
Jóakim v. Kistowski
Samuel Kounev

Chair of Software Engineering
University of Würzburg
http://se.informatik.uni-wuerzburg.de/

Kiel, 08/11/2016
- Increasing server energy consumption
- 61 billion kWh in 2006
- An estimated 140 billion kWh in 2020 [6,7]
Simulation of large networks and/or high data traffic to put externally driven workloads under load.

Example: Load Balancer

Which is the most efficient machine for the workload?

Huppler [8] defines criteria for a good benchmark:

- Repeatable
- Economical
- …
Efficiency is measured under different load levels

Complicated to calibrate and maintain load levels with external load generators due to latency
Motivation

- Approximate externally driven workloads on the SUT without the need for extra hardware
- Use Performance Counters for approximation
- Develop a modularized Performance Event Trigger Framework (PET) to approximate workloads
Approach – Event Trigger

- Performance Counter [1,2]
  - Occurrence Events
    How often has an event been observed
  - Duration Events
    Accumulated clock cycles for which an event has been observed

- Event Trigger
  - Stand-Alone implementation to cause \( i \) counted events

Counter State: \( n \)  

![Event Trigger Diagram]

\[\text{Counter State: } n+i\]
Approach – Side Effects

- Some Performance Counters cannot be modified without affecting other Counter Values
- They can be imposed either by hardware constraints or the implementation of an event trigger

**Events counted:**
- ✗ L1d miss event
- ✗ L2 miss event
- ✗ L3 miss event
- ✔ Read byte event

Example: Trigger event „Read byte from memory controller“ (Accessing main memory)
Different implementations to incorporate side effects

1. Naive: Neglect side effects
2. Accumulation: Sum over all side effects $s_i$ caused by triggering a number of events $v_i$
   \[ s_{total} = \sum_{i=1}^{n} s_i \times v_i \]
3. Simulated Annealing: A numerical solution between the imposed side effects and event triggers
Approach – Evaluate Event Triggers

- Run the event trigger as a single process and in parallel as workloads can and do use multithreading
  - Single process
    → If it does not work in single process, the implementation might be erroneous
  - 4/8 processes
    → Number of physical/logical CPUs to determine if the implementation does scale in a multithreaded environment

- Each event trigger is set a reference value, a target, it has to reach
  - The lower the deviation from the target value the better is the implementation of the event trigger
Use the different caching modes supported by the CPU to prohibit caching → Automatically miss L3

- **Strong Uncachable (UC)**
  - Set by Memory Type Range Register (MTRR)
  - Linux Kernel documentation discourages the use of MTRR [4]

- **Uncachable Minus (UC-)**
  - Set by Page Attribute Table (PAT)
  - Function in Linux Kernel available
Uncachable memory can be mmap-ed to user space

Target value of $8 \times 10^6$
L3 misses for 8 processes not reached

Even worse for 1 and 4 processes

No L3 misses are actually triggered
Traverse a large array of at least twice the cache size

**Problem:** Hardware prefetching loading data we do not want in cache [3]

**Solution:** Increase stride to 6 times the cache line size with a deviation of $-2.4\%$ when 8 processes are running.
- Constraints for triggering L2 misses hitting L3:
  - Instead of traversing a large array, it must be small enough to fit inside L3 not to produce L3 misses on accident
  - The array must also be large enough for long strides to not access data already prefetched and generating a L2 hit

- Try to “confuse” the prefetcher by adding a random factor $r$ after a stride $s$
  $$p_n = p_{n-1} + s + r$$
L2 miss / L3 hits scale well to four processes

<table>
<thead>
<tr>
<th>Processes</th>
<th>Instruction Set</th>
<th>Result (w/o random)</th>
<th>Result (w/ random)</th>
<th>Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SIMD</td>
<td>165,031</td>
<td>923,975</td>
<td>−7.60%</td>
</tr>
<tr>
<td></td>
<td>Assembler</td>
<td>204,282</td>
<td>966,181</td>
<td>−3.38%</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>213,862</td>
<td>971,336</td>
<td>−2.87%</td>
</tr>
<tr>
<td>4</td>
<td>SIMD</td>
<td>443,037</td>
<td>3,391,786</td>
<td>−15.21%</td>
</tr>
<tr>
<td></td>
<td>Assembler</td>
<td>1,225,584</td>
<td>3,408,932</td>
<td>−14.78%</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>743,568</td>
<td>3,537,446</td>
<td>−11.56%</td>
</tr>
<tr>
<td>8</td>
<td>SIMD</td>
<td>2,284,193</td>
<td>5,253,885</td>
<td>−34.33%</td>
</tr>
<tr>
<td></td>
<td>Assembler</td>
<td>1,747,484</td>
<td>4,047,238</td>
<td>−50.59%</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>1,516,549</td>
<td>4,199,528</td>
<td>−47.51%</td>
</tr>
</tbody>
</table>

L2 misses / L3 hits generated; Targets: $1 \times 10^6$, $4 \times 10^6$ and $8 \times 10^6$

Generating L2 misses with virtual CPUs provides no benefits. Triggering L2 misses only scales to the number of physical CPUs
- Read bytes implemented in the same way as L3 misses

- Overcounting by a large margin
  Target for 1 process: \(0.64 \times 10^8\)
  Target for 8 processes: \(0.512 \times 10^9\)
Use the kernel module to circumvent the caches

<table>
<thead>
<tr>
<th>Processes</th>
<th>Instruction Set</th>
<th>Result</th>
<th>Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SIMD</td>
<td>64.003 * 10^6</td>
<td>0.004%</td>
</tr>
<tr>
<td></td>
<td>Assembler</td>
<td>64.038 * 10^6</td>
<td>0.060%</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>64.025 * 10^6</td>
<td>0.039%</td>
</tr>
<tr>
<td>4</td>
<td>SIMD</td>
<td>242.41 * 10^6</td>
<td>−5.31%</td>
</tr>
<tr>
<td></td>
<td>Assembler</td>
<td>255.98 * 10^6</td>
<td>−0.01%</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>191.16 * 10^6</td>
<td>−25.33%</td>
</tr>
<tr>
<td>8</td>
<td>SIMD</td>
<td>326.43 * 10^6</td>
<td>−36.24%</td>
</tr>
<tr>
<td></td>
<td>Assembler</td>
<td>343.67 * 10^6</td>
<td>−32.88%</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>347.82 * 10^6</td>
<td>−32.07%</td>
</tr>
</tbody>
</table>

Bytes read with uncachable memory
Targets: 64 * 10^6, 256 * 10^6 and 512 * 10^6

Read bytes from memory controller scales well to the number of physical CPUs for the SIMD and Assembler instruction sets
Uncachable memory worked well for bytes read so intuitively it should work when writing bytes.

Underestimating bytes written for all instruction sets ranging from $-47.58\%$ to $-73.51\%$

Do not use uncachable memory when triggering bytes written.

SIMD and ASM reach deviations of $-0.03\%$ and $-0.01\%$.

Bytes written for a single process and a stride of 6

Target: $6.4 \times 10^7$
The event trigger is struggling to reach its target value if multiple processes are used.
Create C++11 threads that can be joined instantly

- Intuitively, each thread should cause two switches → Half the amount of event triggers

Introducing a linear factor of 0.5

- Unexpected major deviations
- Removing the factor results in large overcounting

But a linear factor can still improve the accuracy

<table>
<thead>
<tr>
<th>Processes</th>
<th>Factor 0.5</th>
<th>Factor 0.8</th>
<th>Factor 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Result</td>
<td>Deviation</td>
<td>Result</td>
</tr>
<tr>
<td>1</td>
<td>70,350</td>
<td>−29.7%</td>
<td>100,588</td>
</tr>
<tr>
<td>4</td>
<td>271,683</td>
<td>−32.1%</td>
<td>400,689</td>
</tr>
<tr>
<td>8</td>
<td>470,265</td>
<td>−41.2%</td>
<td>757,056</td>
</tr>
</tbody>
</table>

Context switches triggered
Targets: $1 \times 10^5$, $4 \times 10^5$ and $8 \times 10^5$
Approximating Workloads

<table>
<thead>
<tr>
<th>Workload</th>
<th>Measurement</th>
<th>Mean</th>
<th>Max</th>
<th>CV</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSJ</td>
<td>Naive</td>
<td>12.35%</td>
<td>26.44%</td>
<td>19.37%</td>
</tr>
<tr>
<td></td>
<td>Accumulation</td>
<td>13.28%</td>
<td>27.61%</td>
<td>7.03%</td>
</tr>
<tr>
<td></td>
<td>Simulated Annealing</td>
<td>-5.25%</td>
<td>-9.35%</td>
<td>3.66%</td>
</tr>
</tbody>
</table>

- Naive measurement with side effects has a low throughput due to long runtimes
- Accumulation still overestimates power consumption despite removing side effects
Externally driven workloads can be approximated.

Underestimation expected due to the NIC not stressed in the approximation.
Know your hardware to avoid unwanted effects on the events to trigger

Simultaneous Multithreading (SMT) is in most cases not beneficial when triggering performance events on purpose

Intuition can be misleading and counterproductive

Externally driven workloads can be approximated with reasonable accuracy

Complex testbed setups can be simplified for faster and easier deployment → The PET framework reaches an average accuracy from below 10% down to 1%
Thank You!

norbert.schmitt@uni-wuerzburg.de
References


