Hardware Counters Based Methods for the Analysis of Shared Memory Parallel Codes

Oscar García Lorenzo
José Carlos Cabaleiro Domínguez
Tomás Fernández Pena

CITIUS
24/05/2013
Index

1. Introduction
2. Manycore Issues
3. PEBS
4. Memory Analysis
5. Performance Analysis
6. Automatic Optimisation
7. Recap and Future Work
Introduction

Manycore Issues

Memory Issue

Heterogeneity Issue

Memory Issue Solutions (Not touching the code)

Heterogeneity Issue Solutions (Not touching the code)
Introduction

Precise Event Based Sampling

- Hardware Counters in Intel Processors.
- We use them to identify the issues.
- Samples the state of the core (Registers and Counters).
- Saves samples in a buffer.
- Very precise and low overhead.
Introduction

Manycore Issues
PEBS
Memory Analysis
Performance Analysis
Automatic Optimisation
Recap and Future Work

Tools

Memory Analysis
Visual Memory Analysis Tool
Memory Issue

Performance Analysis
Visual Performance Analysis Tool
Memory Issue
Heterogeneity Issue

Automatic Optimisation
Runtime Migration Tool
Memory Issue Solutions
Heterogeneity Issue Solutions
Memory Issue

▷ There is affinity among Cores and Memory:
  - Memory cells are at different distances.
  - There are complex interconnexions.
  - Data can be placed anywhere.

▷ Data has locality.
  - Caches store data.
  - Some levels are shared among cores.
  - Cores read from other core’s caches, many hops.

Intel Xeon Phi ring.
Memory Issue Solutions (Not touching the code)

▷ For the affinity among Cores and Memory:
  - Place threads in cores near the Memory cells that store their data.
  - Place memory pages in cells near the cores accessing them.

▷ For locality.
  - Place threads that access same data nearby.

▷ To do this we need to know:
  - Which data access each thread.
  - Where threads are at (core).
  - Where data is (relative to core).
Heterogeneity Issue

- Cores are not equal.
  - May be by design.
  - Power scaling.
  - Noise.
  - Something else.

- Threads doing the same work take different times.

- Linux just balances threads among processors, if at all.

Intel Xeon E5-4600. 4 processor SMP.
Heterogeneity Issue

Heterogeneity Issue Solutions (Not touching the code)

- To help deal with heterogeneity:
  - Place threads in cores adapted to their work.
  - Balance the load among cores.
  - Move threads in runtime.

- To do this we need to know:
  - Where are all threads at (all cores).
  - What is each thread doing.
  - What is the performance of each thread in each core.
# Index

1. Introduction
2. Manycore Issues
3. PEBS
4. Memory Analysis
5. Performance Analysis
6. Automatic Optimisation
7. Recap and Future Work
PEBS
Precise Event Bases Sampling

▷ Hardware Counters in Intel Processors.
▷ PEBS:
  - Gets one sample each X events (most hardware events supported).
  - Each sample gets the state of the whole core (Registers and Counters).
  - Saves samples in a buffer.
  - Low overhead.
  - Use supported by Linux kernel, nothing else needed.
▷ In addition, very precise information about memory usage:
  - When sampling load or store instructions.
  - Returns the virtual address of the data involved.
  - Returns the latency (in cycles) of the memory access.
PEBS supports only a subset of precise, non-architected events.

### PEBS Buffer Management

<table>
<thead>
<tr>
<th>Offset</th>
<th>64bit/8 bytes Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0H</td>
<td>BTS Buffer base</td>
</tr>
<tr>
<td>8H</td>
<td>BTS index</td>
</tr>
<tr>
<td>10H</td>
<td>BTS absolute max</td>
</tr>
<tr>
<td>18H</td>
<td>BTS intr threshold</td>
</tr>
<tr>
<td>20H</td>
<td>PEBS Buffer base</td>
</tr>
<tr>
<td>28H</td>
<td>PEBS index</td>
</tr>
<tr>
<td>30H</td>
<td>PEBS absolute max</td>
</tr>
<tr>
<td>38H</td>
<td>PEBS intr threshold</td>
</tr>
<tr>
<td>40H</td>
<td>PEBS PMC0 reset</td>
</tr>
<tr>
<td>48H</td>
<td>PEBS PMC1 reset</td>
</tr>
<tr>
<td>50H</td>
<td>PEBS PMC2 reset</td>
</tr>
<tr>
<td>58H</td>
<td>PEBS PMC3 reset</td>
</tr>
<tr>
<td>60H</td>
<td>reserved</td>
</tr>
</tbody>
</table>

### PEBS Record Format

<table>
<thead>
<tr>
<th>Offset</th>
<th>64bit/8 bytes Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>R/EFLAGS</td>
</tr>
<tr>
<td>0x8</td>
<td>E/REIP</td>
</tr>
<tr>
<td>0x10</td>
<td>R/EAX</td>
</tr>
<tr>
<td>0x18</td>
<td>E/EBX</td>
</tr>
<tr>
<td>0x20</td>
<td>R/ECX</td>
</tr>
<tr>
<td>0x28</td>
<td>R/EDX</td>
</tr>
<tr>
<td>0x30</td>
<td>R/ESI</td>
</tr>
<tr>
<td>0x38</td>
<td>R/EDI</td>
</tr>
<tr>
<td>0x40</td>
<td>W/EBP</td>
</tr>
<tr>
<td>0x48</td>
<td>R/ESP</td>
</tr>
<tr>
<td>0x50</td>
<td>R8</td>
</tr>
<tr>
<td>0x58</td>
<td>R9</td>
</tr>
<tr>
<td>0x60</td>
<td>R10</td>
</tr>
<tr>
<td>0x68</td>
<td>R11</td>
</tr>
<tr>
<td>0x70</td>
<td>R12</td>
</tr>
<tr>
<td>0x78</td>
<td>R13</td>
</tr>
<tr>
<td>0x80</td>
<td>R14</td>
</tr>
<tr>
<td>0x88</td>
<td>R15</td>
</tr>
<tr>
<td>0x90</td>
<td>IA32_PERF_GLOBAL_STATUS</td>
</tr>
<tr>
<td>0x98</td>
<td>Data linear Address</td>
</tr>
<tr>
<td>0xA0</td>
<td>Data Source encoding</td>
</tr>
<tr>
<td>0xA8</td>
<td>Latency (core cycles)</td>
</tr>
</tbody>
</table>

PEBS Buffer.
PEBS Information we read

Memory Issue Solutions

To do this we need to know:

- Which data access each thread.
  - Sample load and store instructions.
  - Virtual address of the data, ADDR (OK).

- Where threads are at (core).
  - Any instruction.
  - PID, TID, CPU (OK).

- Where data is (relative to core).
  - Sample load and store instructions.
  - Latency of the load instruction, LATENCY, plus ADDR (OK).
Heterogeneity Issue Solutions

To do this we need to know:

- Where are all threads at (all cores).
  - Sample system wide, all cores (OK).

- What is each thread doing.
  - Sample Flops/sec and number of cache lines loaded.
  - We know if the code is computation bound or memory bound (OK).

- What is the performance of each thread in each core.
  - Sample Flops/sec, number of cache lines loaded, instructions for all cores.
  - We know the entire system performance, we can compare (OK).
Memory Analysis

Description

▷ The data is captured from PEBS.
▷ It allows us to see the captured accesses to memory.
  - Diverse levels of detail available.
  - Can see data per thread or whole system.
  - Can see data per instruction.
  - Can see data per core.
▷ It shows the latency of each access.
▷ It can cross-check data:
  - It can find false sharing.
  - It can simulate caches.
Memory Analysis

General Occurrence histogram. In the Histogram L2 misses are shown in red, L3 in green, and Main Memory in orange.
Memory Analysis

Detailed Latency histogram.
Memory Analysis

Normal Use

Cache misses, each bar represents 400 consecutive addresses.

Latencies, each bar represents 16 addresses.

Event table.
Memory Analysis

Data split

bcsstk29

Thread 0

Thread 1

Thread 2

Thread 3
Memory Analysis

Memory Cells Latencies

sme3Da

Core 0

Core 1

Core 8

Core 9
Memory Analysis

Difference between cyclic and block distribution.

\[ b = 4, \text{Occurrences} \]

\[ b = 4, \text{Latencies} \]

Block distribution, Occurrences

Block distribution, Latencies
Performance Analysis

Performance Analysis Tool Description

- The data is captured from PEBS, visualised in R.
- It allows to see the GFlops/sec, Flops/Byte and mean MemoryLatency.
  - Can see data per thread or per core.
  - Can see the evolution of a code in time.
  - Can detect phases in execution.
- It captures the noise in the system.
- Can capture data from many concurrent processes.
  - It can find differences among cores.
  - It can find differences among threads.

Performance Analysis

Visual Performance Analysis Tool

Memory Issue

Heterogeneity Issue
Performance Analysis

The R Environment GUI

The R Environment GUI.
Performance Analysis  System with 8 Cores

Dynamic Roofline for system with 8 cores.
Performance Analysis

Dynamic Roofline

Roofline ua.A No Optimisations

Roofline ua.A

Density. ua.A No Optimisations

Density. ua.A
Performance Analysis  Dynamic Roofline

Performance Analysis  
Roofline 3D

Roofline 3D, GFlops/FlopB  
Roofline 3D, GFlops/Latency(cycles)

Roofline 3D. GFlops/FlopsB/Latency. Processor 0 is shown in red, Processor 1 in black
Performance Analysis  

Roofline 3D

Roofline 3D, GFlops/FlopB  
Roofline 3D, FlopB/Latency(cycles)

Roofline 3D of the ep.A and ft.A benchmarks. GFlops/FlopsB/Latency. Processor 0 is shown in red, Processor 1 in black
Automatic Optimisation

Runtime Migration Tool

Automatic Performance Optimisation

Runtime Migration Tool

Memory Issue Solutions

Heterogeneity Issue Solutions
Automatic Optimisation
What needs to be done

Memory Issue Solutions

▷ For the affinity among Cores and Memory:
  - Place threads in cores near the Memory cells that store their data.
  - Place memory pages in cells near the cores accessing them.

▷ For locality.
  - Place threads that access same data nearby.

Heterogeneity Issue Solutions

▷ To help deal with heterogeneity:
  - Place threads in cores adapted to their work.
  - Balance the load among cores.
  - Move threads in runtime.
Automatic Optimisation

Memory Issue Solutions

▷ For the affinity among Cores and Memory:
  - Place threads in cores near the Memory cells that store their data.
  - Place memory pages in cells near the cores accessing them.
    > Page Migration, Juan A. Lorenzo PhD dissertation.

▷ For locality.
  - Place threads that access same data Nearby.

Heterogeneity Issue Solutions

▷ To help deal with heterogeneity:
  - Place threads in cores adapted to their work.
  - Balance the load among cores.
  - Move threads in runtime.
Automatic Optimisation

Some Results

▷ A code with many memory accesses.
▷ All threads do the same work.
▷ Two processors (0 and 1), each connected to a 8GB memory cell.
▷ All data is stored near the processor 0.
▷ Threads in processor 1 take longer.
▷ We measure memory access latency for each thread.
▷ Simple Strategy: The worst thread is moved.
   - To an empty core if possible.
   - Interchange with the best.
▷ Times:
   - Without migration: 4:31 min
   - With thread migration: 4:16 min
▷ Works somewhat.
We can move threads between cores.
We can move memory pages between cells.
We can capture data about memory usage, per core and thread.
We can capture data about performance, per core and thread.
The problem now is to combine everything.
  - We need strategies and algorithms.

Then we will have to test it.
  - We need access to more machines and systems.
  - Software support is not ready.
Recap

Memory Analysis
Visual Memory Analysis Tool

Memory Issue

Performance Analysis
Visual Performance Analysis Tool

Memory Issue
Heterogeneity Issue

Automatic Optimisation
Runtime Migration Tool

Memory Issue Solutions
Heterogeneity Issue Solutions
Future Work

- We need algorithms.
- We need to test them.
- We need to validate them in more systems.
Publications


- Oscar G. Lorenzo, Juan A. Lorenzo, José C. Cabaleiro, Dora B. Heras, Marcos Suarez, Juan C. Pichel "A Study of Memory Access Patterns in Irregular Parallel Codes Using Hardware Counter-Based Tools", Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA), Las Vegas (USA), 2011, pages 920–923.

- Oscar G. Lorenzo; Tomás F. Pena; José C. Cabaleiro; Juan C. Pichel; Juan A. Lorenzo; Francisco F. Rivera, "Hardware Counter Based Analysis of Memory Accesses in SMPs", ISPA2012 Madrid, July 2012.
Oscar G. Lorenzo; Tomás F. Pena; José C. Cabaleiro; Juan C. Pichel; Francisco F. Rivera, “DyRM: A Dynamic Roofline Model Based on Runtime Information”, CMMSE 2013 (Accepted)

Oscar G. Lorenzo; Tomás F. Pena; José C. Cabaleiro; Juan C. Pichel; Juan A. Lorenzo; Francisco F. Rivera, “A Hardware Counters Based Toolkit for the Analysis of Memory Accesses in SMPs”, Concurrency and Computation: Practice and Experience (Under Review)
Thank you!!