Thread Clustering: Sharing-Aware Scheduling on SMP-CMP-SMT Multiprocessors

David Tam, Reza Azimi, Michael Stumm

Department of Electrical and Computer Engineering University of Toronto, Canada
Outline

• Introduction: OpenPower 720 Architecture
• Motivation
• Performance Management Unit
• Design of thread Clustering Scheme
• Evaluation
• Contribute
• Summary
OpenPower 720 Server

- Design: *Performance, Scalability, Reliability etc*
- Power7 processors (SMP-CMP-SMT Multiprocessor)
  - Designed Multi-core architecture (called **CMP**) for leading the throughput
  - Shared memory multiprocessors (**SMP**)
  - Simultaneous Multithreading (**SMT**)

<table>
<thead>
<tr>
<th>Item</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Chips</td>
<td>2</td>
</tr>
<tr>
<td># of Cores</td>
<td>2 per chip</td>
</tr>
<tr>
<td>CPU Cores</td>
<td>IBM Power5, 1.5GHz, 2-way SMT</td>
</tr>
<tr>
<td>L1 DCache</td>
<td>64KB, 4-way associative, per core</td>
</tr>
<tr>
<td>L1 ICache</td>
<td>64KB, 4-way associative, per core</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>2MB, 10-way associative, per chip</td>
</tr>
<tr>
<td>L3 Cache</td>
<td>36MB, 12-way associative, per chip, off-chip</td>
</tr>
<tr>
<td>RAM</td>
<td>8GB (2 banks x 4GB)</td>
</tr>
</tbody>
</table>

*Table 1: IBM OpenPower 720 specification.*

*Figure 1. Power4 (a) and Power5 (b) system structures.*
Simutaneous multithreading (SMT)

• Multiple independent threads to execute SIMULTANEOUSLY on the SAME core
  • Increase Core Efficiency

• Example
  • Single thread: The process pipeline get stalled when waiting for data to arrive from memory
  • If one thread is waiting for a floating point operation, another thread can use the integer units

• Power5
  • By SMT, 2 virtual processor per real processor
Motivation

• The poor performance is ... what...

Solution? Power-5 (8-logical processors)
• Increase Cache Size ... Money!
• Add more processor ... Power!
• Hire more Chip Architecture engineers.. ??

Figure 1: The IBM OpenPower 720 architecture. The numbers on the arrows indicate the access latency from a thread to different levels of the memory hierarchy. Any cross-chip sharing takes at least 120 CPU cycles.

Figure 3: The stall breakdown for VolanoMark. The stalls due to data cache misses are further broken down according to the source from where the cache miss is eventually satisfied.

Existing OS doesn’t handle the complexity of multicore processors
Overview of Thread Clustering Scheme

• Design of Thread Clustering Scheme

1. Monitoring Stall Breakdown
2. Detecting Sharing Patterns
3. Thread Clustering
4. Thread Migration

Figure 2: Default versus clustered scheduling. The solid lines represent high-latency cross-chip communications, the dashed lines are low-latency intra-chip communications (when sharing occurs within the on-chip L1 and L2 caches).
Step 1: Monitoring Stall Breakdown

- **PMU**
  - Detect various event that can count in Processor

- **Introduction PMU on Cortex-R**
  - Select Only three event register
  - Overflow handling
  - Difficult to extract high-level insight

---

**Table 6.1: Event bus interface bit functions**

<table>
<thead>
<tr>
<th>EVENTBUS bit position</th>
<th>Description</th>
<th>CPU Update</th>
<th>Event Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NRB</td>
<td>Software increment. The register is incremented only on writes to the Software Increment Register. See CR0.SI.</td>
<td>-</td>
<td>0x00</td>
</tr>
<tr>
<td>[0]</td>
<td>Instruction cache miss. Each instruction fetch from normal Cacheable memory that causes a refill from the level 2 memory system generates this event. Accesses that do not cause a new cache refill, but are satisfied from refilling data of a previous miss are not counted. Where Instruction fetches consist of multiple instructions, these accesses count as single events. CP15 cache maintenance operations do not count as events.</td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>[1]</td>
<td>Data cache miss. Each data read from or write to normal Cacheable memory that causes a refill from the level 2 memory system generates this event. Accesses that do not cause a new cache refill, but are satisfied from refilling data of a previous miss are not counted. Each access to a cache line to normal Cacheable memory that causes a new line fill is counted, including the multiple transactions of an LDM and STM. Write-through writes that hit in the cache do not cause a line fill and so are not counted. CP15 cache maintenance operations do not count as events.</td>
<td>-</td>
<td>0x63</td>
</tr>
<tr>
<td>[2]</td>
<td>Data cache access. Each access to a cache line is counted including the multiple transactions of an LDM, STM, or other operations. CP15 cache maintenance operations do not count as events.</td>
<td>-</td>
<td>0x64</td>
</tr>
<tr>
<td>[3]</td>
<td>Data Read architecturally executed. This event occurs for every instruction that explicitly reads data, including str.</td>
<td>-</td>
<td>0x65</td>
</tr>
<tr>
<td>[4]</td>
<td>Data Write architecturally executed. This event occurs for every instruction that explicitly writes data, including sot.</td>
<td>-</td>
<td>0x66</td>
</tr>
<tr>
<td>[5]</td>
<td>Instruction architecturally executed.</td>
<td>-</td>
<td>0x67</td>
</tr>
<tr>
<td>[6]</td>
<td>User-initiated pair of instructions architecturally executed.</td>
<td>-</td>
<td>0x68</td>
</tr>
<tr>
<td>[7]</td>
<td>Exception taken. This event occurs on each exception taken.</td>
<td>-</td>
<td>0x69</td>
</tr>
<tr>
<td>[8]</td>
<td>Exception return architecturally executed. This event occurs on every exception return, for example, VFS, MOV PSR, LDM, PCH.</td>
<td>-</td>
<td>0x6A</td>
</tr>
<tr>
<td>[9]</td>
<td>Change to Context ID executed.</td>
<td>-</td>
<td>0x6B</td>
</tr>
<tr>
<td>[10]</td>
<td>Software change of PC, except by an exception, architecturally executed.</td>
<td>-</td>
<td>0x6C</td>
</tr>
</tbody>
</table>

**PMU On**

- **Func A**
  - Data Cache Miss: 0
  - Branch Mis-Prediction: 0
  - Instruction count: 0
  - ClockCount: 0

- **Func B**
  - Data Cache Miss: A
  - Branch Mis-Prediction: B
  - Instruction count: C
  - ClockCount: D

**PMU Off**

- **Func C**
Step 2: Detect Sharing Patterns

- Construction of shMaps
  - Build shMap (*summary data structure*), count remote cache access (8-bit counter)
  - Set one region index on shMap Vector if cache miss is satisfied by remote cache access
  - Region size: 128 bytes (equal to cache size)
  - But how to enclose total virtual address space with only 128 region entry
    - Use simple hashing function (region = address % 128 ?)
Step2: Detect Sharing Patterns

• For Low Overhead & Reduce Noise (false report) on Step2
  • **Temporal Sampling**: not every time to record and process when remote cache access, Only one set in N occurrences of remote cache access
  • **Spatial Sampling**: To reduce hash collisions (*false report*) and can maintain small memory size of shmap vector
Step 3: Thread Clustering

- Define the similarity of two shMap vectors

\[ \text{similarity}(T_1, T_2) = \sum_{i=0}^{N} T_1[i] \times T_2[i] \]

where \( i \) is the \( i \)th entry of the vector \( T_x[ ] \)

*The similarity value will be high when two threads are sharing data (Thread A and B)*

<table>
<thead>
<tr>
<th>shMap Vector (Thread A)</th>
<th>256</th>
<th>200</th>
<th>130</th>
</tr>
</thead>
<tbody>
<tr>
<td>shMap Vector (Thread B)</td>
<td>256</td>
<td>150</td>
<td></td>
</tr>
</tbody>
</table>
Evaluation (1/2)

- Continuous vertical dark-line means clustered threads

a. Microbenchmark

b. SPECjbb2000

c. RUBiS

d. VolanoMark
Evaluation (2/2)

- Performance improvement of up-to 7%
- Reduce stall due to remote cache access

**Figure 6:** The impact of the scheduling schemes on reducing stalls caused by remote cache accesses. The baseline is Linux default scheduling.

**Figure 7:** The performance impact of scheduling schemes on application performance. The baseline is Linux default scheduling.
Related Work: is it being used nowdays?

• 추가 준비 중
Summary

• Propose New Thread scheduling
  • Using run-time information from hardware performance count
  • Detection sharing pattern different threads
  • Find Best location thread position not making remote cache access anymore
  • OS Job scheduler to re-assign threads that share data to the same chip domain (memory domain) with low overhead
Thank you!

Questions?