Addressing Shared Resource Contention in Multicore Processors via Scheduling

ASPLOS’ 10
Sergey Zhuravlev, Sergey Blagodurov, Alexandra Fedorova
Simon Fraser University

Presented by Jingweijia Tan
Introduction

• Multicore processors become prevalent.

• Shared resource contention remains an unsolved problem in existing OS scheduling.
  – Load balancing

• Previous solutions focuses primarily on cache contention.
  – Not the dominant cause of performance degradation
Goal

• Investigate contention-aware scheduling techniques to mitigate performance degradation due to shared resource contention.
  – classification scheme
  – Scheduling policy
Contributions

• Analyze the effectiveness of various classification schemes

• Discover a classification scheme that addresses resource contention
  – Including cache space, memory controller, memory bus, and prefetching hardware.

• Design a new scheduling algorithm
Classification Schemes

• A “perfect scheduling policy” [Jiang, PACT’08]
  – Uses the co-run degradations to construct a graph theoretic representation of the problem, where threads are represented as nodes connected by edges, and the weights of the edges are given by the sum of the mutual co-run degradations between the two threads.
  – The optimal scheduling assignment can be found by solving a min-weight perfect matching problem.
Classification Schemes

- A "perfect scheduling policy" [Jiang, PACT’08]

<table>
<thead>
<tr>
<th></th>
<th>mcf</th>
<th>milc</th>
<th>games</th>
<th>namd</th>
</tr>
</thead>
<tbody>
<tr>
<td>mcf</td>
<td>48.01%</td>
<td>65.63%</td>
<td>2.0%</td>
<td>2.11%</td>
</tr>
<tr>
<td>milc</td>
<td>24.75%</td>
<td>45.39%</td>
<td>1.23%</td>
<td>1.11%</td>
</tr>
<tr>
<td>games</td>
<td>2.67%</td>
<td>4.48%</td>
<td>-1.01%</td>
<td>-1.21%</td>
</tr>
<tr>
<td>namd</td>
<td>1.48%</td>
<td>3.45%</td>
<td>-1.19%</td>
<td>-0.93%</td>
</tr>
</tbody>
</table>

Table 1. Co-run degradations of four obtained on a real system. Small negative degradations for some benchmarks occur as a result of sharing of certain libraries. The value on the intersection of row $i$ and column $j$ indicates the performance degradation that application $i$ experiences when co-scheduled with application $j$.

Figure 2. An overview of using Jiang’s method for determining the optimal and the worst thread schedule. Edges connecting nodes are labeled with mutual co-run degradations, i.e., the sum of individual degradations for a given pair. The average degradation for a schedule is computed by summing up all mutual degradations and dividing by the total number of applications (four in our case).
Classification Schemes

• SDC [Chandra, HPCA’05]
  – Model how two applications compete for the LRU position and estimate the extra misses.
  – The sum of the extra misses from the co-runners is the proxy for the performance degradation of this con-schedule
  – Construct a new stack distance profile that merges individual stack distance profiles of threads that run together.
Classification Schemes

• Animal Classes [Xie, ISCA’08]
  – Classify applications’ influence on each other when co-scheduled.
  – 4 different classes: turtle, sheep, rabbit, and devil.

• Miss Rate [Knauerhase, IEEE Micro’08]
Classification Schemes

• Pain
  – Cache sensitivity, cache intensity
  – Sensitivity: how much an application will suffer when cache space is taken away due to contention
  – Intensity: how much an application will hurt others by taking away their apace in a shared cache
Classification Schemes Evaluation

Figure 3. Degradation relative to optimal experienced by each classification scheme on systems with different numbers of cores.
Factors Causing Performance Degradation

- **FSB**: front-side bus

*Figure 4.* Percent contribution that each of the factors have on the total degradation.
Scheduling Algorithms

• A combination of a classification scheme and a scheduling policy
• Classification scheme: Miss Rate
  – Easy to obtain online
• Scheduling policy: Centralized Sort
  – Sort applications’ miss rates, and distributes them across cores, such that the total miss rate of all threads sharing a cache is equalized across all caches
Scheduling Algorithms

• Distributed Intensity (DI)
  – all threads are assigned a value which is their solo miss rate as determined from the stack distance profile.
  – The goal is then to distribute the threads across caches such that the miss rates are distributed as evenly as possible.

• Distributed Intensity Online (DIO)
  – obtains the miss rates of applications dynamically online via performance counters
Evaluation Platform

• Dell-Poweredge-2950 (Intel Xeon X5365)
  – eight cores placed on four chips
  – Each chip has a 4MB 16-way L2 cache shared by its two cores

• Dell-Poweredge-R805 (AMD Opteron 2350 Barcelona)
  – eight cores placed on two chips
  – Each chip has a 2MB 32-way L3 cache shared by its four cores
Workloads

- 14 benchmarks from SPEC CPU 2006 suite

<table>
<thead>
<tr>
<th>Workloads</th>
<th>2 memory-bound</th>
<th>2 CPU-bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Soplex</td>
<td>Sphinx</td>
<td>Gameess</td>
</tr>
<tr>
<td>2 Soplex</td>
<td>mcf</td>
<td>Gameess</td>
</tr>
<tr>
<td>3 MCF</td>
<td>Libquantum</td>
<td>Povray</td>
</tr>
<tr>
<td>4 MCF</td>
<td>Omnetpp</td>
<td>H264</td>
</tr>
<tr>
<td>5 Milc</td>
<td>Libquantum</td>
<td>Povray</td>
</tr>
<tr>
<td>6 Sphinx</td>
<td>gcc</td>
<td>NAMD</td>
</tr>
<tr>
<td>7 Lbm</td>
<td>Milc</td>
<td>SphinX</td>
</tr>
<tr>
<td>8 Lbm</td>
<td>Milc</td>
<td>MCF</td>
</tr>
</tbody>
</table>

Table 2. The workloads used for experiments.
Results

• Intel Xeon 4 cores

- DI and DIO perform better than RANDOM and are within 2% of OPTIMAL

**Figure 7.** Aggregate performance degradation of each workload with DI, DIO, RANDOM and WORST relative to OPTIMAL (low bars are good) for the Intel machine and 4 threads.
Results

- Intel Xeon 8 cores

**Figure 8.** Aggregate performance improvement of each workload with DI and DIO relative to DEFAULT (high bars are good) for Intel 8 threads.
Results

- AMD Opteron 8 cores

Figure 9. Aggregate performance improvement of each workload with DI and DIO relative to DEFAULT (high bars are good) for AMD 8 threads.
Discussion

- The classification scheme based on miss rates effectively reduces contention for shared resources using a scheduling approach.
- An algorithm based on this classification scheme can be effectively implemented online (DIO).
- Using contention-aware scheduling can help improve overall system efficiency.
Related Work

- Utility Cache Partitioning [Qureshi, MICRO’06]
  - Hardware based cache partition
  - estimates each application’s number of hits and misses for all possible number of ways allocated to the application
  - partition so as to minimize the number of cache misses for the co-running applications

- Cache Page Coloring [Tam, ASPLOS’09]
  - Software based cache partition
  - Each application is reserved a portion of the cache, and the physical memory is allocated such that the application’s cache lines map only into that reserved portion.
  - The size of the allocated cache portion is determined based on the marginal utility of allocating additional cache lines for that application.
Conclusions

- Identified factors other than cache space contention which cause performance degradation
- Predicted that in order to alleviate these factors it was necessary to minimize the total number of misses issued from each cache.
- Developed scheduling algorithms DI and DIO that distribute threads such that the miss rate is evenly distributed among the caches