A Hardware Evaluation of Cache Partitioning to Improve Utilization and Energy-Efficiency while Preserving Responsiveness

Henry Cook, Miquel Moreto, Sarah Bird, Kanh Dao, David Patterson, Krste Asanovic

University of California, Berkeley
Universitat Politecnica de Catalunya
Responsiveness (In The Cloud)

• Introducing server-side search result delays of \(< 0.5\) seconds impacts critical business metrics
  – Time to click, satisfaction, daily searches per user
• The cost of added delay increases over time and persists afterwards
• Results were so negative that some A/B experiments were halted ahead of schedule

Responsiveness (On The Client)

• “Save user data and app state information. ...This step is necessary because your app might be quietly killed while in the background for any number of reasons.”

• “Using these calls causes your app to be killed immediately.”

• “When your app is suspended, if it is found to be using a shared resource, the app is killed.”
Underutilization

“The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines”
Luiz André Barroso and Urs Hölzle, 2009.


A New Opportunity

• Sandy Bridge client device prototype HW
  – Way-based LLC partitioning
  – Energy counters

• Full size parallel benchmarks, full system stack

• Goal: Evaluate the energy-saving potential of consolidation with HW for cache partitioning
Machine Resources

4 cores, 8 HyperThreads

6MB, 12 way LLC

DCU prefetchers

Interconnect BW

MLC prefetchers

DRAM BW

Partitionable

Unpartitionable
Way-Based Partitioning

HT1

Way 0

Way N

HT2
Methodology

• Multiple benchmark suites
  – Spec2006, PARSEC, DaCapo, other parallel kernels
  – Full/large/native input sets
• Unmodified Linux 2.6.36
• Libpfm library built on perf_events
• Running Average Power Limit (RAPL) interfaces
  – 16us granularity
• ACme external power meter
  – 1 sec granularity
  – http://acme.cs.berkeley.edu
Hierarchical K-means Clustering
Race to Halt

• Scattered points are the 8x12 possible allocations
• Energy $\alpha$ performance
• Applies across all benchmarks and allocations
LLC and HT Utilization

Energy

LLC size

# HTs
Multiprogram Contention
Static Partitioning: Unpartitioned

- Baseline for measuring foreground app degradation is to just let apps share each way of the LLC
- Replacement policy evicts based on usage patterns

<table>
<thead>
<tr>
<th>Static partitioning</th>
<th>Average slowdown</th>
<th>Worst-case slowdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unpartitioned</td>
<td>5.9%</td>
<td>34.0%</td>
</tr>
</tbody>
</table>
Static Partitioning: Fair Partitioning

- Fair partitioning gives each app half of the cache, regardless of need
- Most naïve use of partitioning

<table>
<thead>
<tr>
<th>Static partitioning</th>
<th>Average slowdown</th>
<th>Worst-case slowdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unpartitioned</td>
<td>5.9%</td>
<td>34.0%</td>
</tr>
<tr>
<td>Fair</td>
<td>6.0%</td>
<td>16.3%</td>
</tr>
</tbody>
</table>
Static Partitioning: Ideal Partitioning

• Ideal partitioning uses the “best” allocation
  – Heuristic is: smallest FG alloc whose perf was within 1% of giving FG the whole machine, yet allows BG to run in remainder

• Oracular static partitioning

<table>
<thead>
<tr>
<th>Static partitioning</th>
<th>Average slowdown</th>
<th>Worst-case slowdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unpartitioned</td>
<td>5.9%</td>
<td>34.0%</td>
</tr>
<tr>
<td>Fair</td>
<td>6.0%</td>
<td>16.3%</td>
</tr>
<tr>
<td>Ideal</td>
<td>2.3%</td>
<td>7.4%</td>
</tr>
</tbody>
</table>
Static Partitioning: Takeaways

• Partitioning mitigates worst-case degradation
• For metrics like energy or weighted speedup, consolidation is effective but differences between sharing strategies are small on average
• High variance across application pairs
• Pairing strategy >> sharing strategy
Applications Have Phases

• Can we dynamically determine the LLC requirements and further consolidate?
Dynamic Algorithm

• Use performance counters to detect changes in required LLC alloc, via miss rate
• When a phase change is detected, explore allocations to determine new required size
• Give FG maximum alloc, then shrink alloc until miss rate is negatively impacted
• Hold allocation fixed until another change in miss rate is detected
Dynamic Algorithm

FG Miss Rate

FG  FG  FG  FG  FG  FG  BG  BG  BG  BG  BG  BG
Dynamic Algorithm

FG Miss Rate

FG

BG
Dynamic Algorithm

FG Miss Rate

FG

BG

23
Dynamic Algorithm

FG Miss Rate

FG

BG

24
Dynamic Algorithm

FG Miss Rate

FG

BG
Dynamic Algorithm
Dynamic Algorithm

FG Miss Rate

FG

BG
Dynamic algorithm

FG Miss Rate

FG  BG

28
Dynamic Algorithm Results

• In some cases we see significant throughput increases (up to 2.5x), resulting in a 19% throughput improvement on average
  – FG performance never worsens more than 2%
• Using a shared LLC results in a 53% throughput improvement on average
  – However, this scenario can often result in significant perf loss (up to 35%) for FG app
• Throughput correlated with energy/task
Future Work

• Explain discrepancies between real machine utilities and others’ simulated results
• More big data workloads
• App-pair-specific dynamic mechanism tuning
• Mechanisms for BW partitioning
• Mechanisms to preserve prefetcher efficacy
Conclusions

• The race-to-halt paradigm still allows for consolidation opportunities
• LLC partitioning alone is not enough to prevent degradation, but mitigates worst case
• Consolidation is very effective for saving energy, but pairing strategy >> static sharing strategy
• Dynamic LLC partitioning can be effective at reducing energy per background task while preserving FG performance
LLC sensitivity
Thread scalability
Utilization Diversity
BW Hog
Prefetcher Sensitivity
Wall vs socket