Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior

Yoongu Kim
Michael Papamichael
Onur Mutlu
Mor Harchol-Balter
Motivation

• Memory is a shared resource

- Threads’ requests contend for memory
  - Degradation in single thread performance
  - Can even lead to starvation

• How to schedule memory requests to increase both system throughput and fairness?
Previous Scheduling Algorithms are Biased

No previous memory scheduling algorithm provides both the best fairness and system throughput.
Why do Previous Algorithms Fail?

**Throughput biased approach**
Prioritize less memory-intensive threads

**Fairness biased approach**
Take turns accessing memory

- **Good for throughput**: Less memory intensive
- **Does not starve**: Higher priority
- **Starvation ➔ Unfairness**
- **Not prioritized ➔ Reduced throughput**

Single policy for all threads is insufficient
Insight: Achieving Best of Both Worlds

For Throughput
- Prioritize memory-non-intensive threads

For Fairness
- Unfairness caused by memory-intensive being prioritized over each other
  - Shuffle threads
- Memory-intensive threads have different vulnerability to interference
  - Shuffle asymmetrically
Outline

- Motivation & Insights
- Overview
- Algorithm
- Bringing it All Together
- Evaluation
- Conclusion
Overview: Thread Cluster Memory Scheduling

1. Group threads into two clusters
2. Prioritize non-intensive cluster
3. Different policies for each cluster

Threads in the system

Memory-non-intensive

Memory-intensive

Non-intensive cluster

Prioritized

Intensive cluster

Throughput

Fairness
Outline

- Motivation & Insights
- Overview
- **Algorithm**
- Bringing it All Together
- Evaluation
- Conclusion
TCM Outline

1. Clustering
Clustering Threads

**Step 1** Sort threads by **MPKI** (misses per kilo-instruction)

- Sort threads by MPKI.
- Threads are clustered based on MPKI.
- Intensive cluster: threads with higher MPKI.
- Non-intensive cluster: threads with lower MPKI.

**Step 2** Memory bandwidth usage $\alpha T$ divides clusters.

- $T = \text{Total memory bandwidth usage}$
- $\alpha < 10\%$ Cluster Threshold

$\alpha T$ divides memories into intensive and non-intensive clusters based on MPKI.
1. Clustering

2. Between Clusters
Prioritize non-intensive cluster

- Increases system throughput
  - Non-intensive threads have greater potential for making progress

- Does not degrade fairness
  - Non-intensive threads are “light”
  - Rarely interfere with intensive threads
TCM Outline

1. Clustering

2. Between Clusters

3. Non-Intensive Cluster
Prioritize threads according to MPKI

- Increases system throughput
  - Least intensive thread has the greatest potential for making progress in the processor
TCM Outline

1. Clustering
2. Between Clusters
3. Non-Intensive Cluster
4. Intensive Cluster

Throughput
Fairness
Periodically shuffle the priority of threads

- Is treating all threads equally good enough?
- **BUT**: Equal turns ≠ Same slowdown
**Case Study: A Tale of Two Threads**

Case Study: Two intensive threads contending

1. *random-access*
2. *streaming*

Which is slowed down more easily?

Prioritize *random-access*

Prioritize *streaming*

*random-access* thread is more easily slowed down
Why are Threads Different?

Bank 1  Bank 2  Bank 3  Bank 4

Memory

rows
Why are Threads Different?

random-access

- All requests parallel
- High bank-level parallelism
Why are Threads Different?

*random-access*  
*streaming*

- All requests parallel
- High **bank-level parallelism**

- All requests ➔ Same row
- High **row-buffer locality**
Why are Threads Different?

**random-access**

- All requests parallel
- High **bank-level parallelism**

**streaming**

- All requests ➔ Same row
- High **row-buffer locality**

**stuck** ➔ req

Vulnerable to interference
1. Clustering
2. Between Clusters
3. Non-Intensive Cluster
4. Intensive Cluster

TCM Outline
Niceness

How to quantify difference between threads?

High Niceness Low

Bank-level parallelism
Vulnerability to interference

Row-buffer locality
Causes interference

Niceness

+ 

-
Shuffling: Round-Robin vs. Niceness-Aware

1. Round-Robin shuffling

2. Niceness-Aware shuffling

What can go wrong?
Shuffling: Round-Robin vs. Niceness-Aware

1. **Round-Robin** shuffling

   - What can go wrong?

2. **Niceness-Aware** shuffling

   - GOOD: Each thread prioritized once

---

**Priority**

- D
- C
- B
- A

**Time**

- D
- A
- B
- C
- D
- A
- B
- C
- D
- A

**ShuffleInterval**

- **Nice thread**
- **Least nice thread**

**Most prioritized**

- D
- A
- B
- C
- D

GOOD: Each thread prioritized once.
Shuffling: Round-Robin vs. Niceness-Aware

1. **Round-Robin shuffling**

What can go wrong?

2. **Niceness-Aware shuffling**

**GOOD:** Each thread prioritized once

**BAD:** Nice threads receive lots of interference

*ShuffleInterval*
Shuffling: Round-Robin vs. Niceness-Aware

1. **Round-Robin** shuffling

2. **Niceness-Aware** shuffling
Shuffling: Round-Robin vs. Niceness-Aware

1. **Round-Robin** shuffling

2. **Niceness-Aware** shuffling

**GOOD:** Each thread prioritized once
Shuffling: Round-Robin vs. Niceness-Aware

1. **Round-Robin** shuffling

2. **Niceness-Aware** shuffling

**GOOD:** Each thread prioritized once

**GOOD:** Least nice thread stays mostly deprioritized
TCM Outline

1. Clustering
2. Between Clusters
3. Non-Intensive Cluster
4. Intensive Cluster

Throughput
Fairness
Outline

- Motivation & Insights
- Overview
- Algorithm

- Bringing it All Together
- Evaluation
- Conclusion
Quantum-Based Operation

**Previous quantum** (~1M cycles)

**During quantum:**
- Monitor thread behavior
  1. Memory intensity
  2. Bank-level parallelism
  3. Row-buffer locality

**Current quantum** (~1M cycles)

**Shuffle interval** (~1K cycles)

**Beginning of quantum:**
- Perform clustering
- Compute niceness of intensive threads

**Time**
TCM Scheduling Algorithm

1. **Highest-rank**: Requests from higher ranked threads prioritized
   - Non-Intensive cluster > Intensive cluster
   - Non-Intensive cluster: lower intensity  ➞ higher rank
   - Intensive cluster: rank shuffling

2. **Row-hit**: Row-buffer hit requests are prioritized

3. **Oldest**: Older requests are prioritized
Implementation Costs

*Required storage at memory controller* (24 cores)

<table>
<thead>
<tr>
<th>Thread memory behavior</th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPKI</td>
<td>~0.2kb</td>
</tr>
<tr>
<td>Bank-level parallelism</td>
<td>~0.6kb</td>
</tr>
<tr>
<td>Row-buffer locality</td>
<td>~2.9kb</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>&lt; 4kbits</strong></td>
</tr>
</tbody>
</table>

- No computation is on the critical path
Outline

- Motivation & Insights
- Overview
- Algorithm
- Bringing it All Together
- Evaluation
- Conclusion
Metrics & Methodology

• Metrics

**System throughput**

\[
\text{Weighted Speedup} = \sum_i \frac{\text{IPC}_{i,\text{shared}}}{\text{IPC}_{i,\text{alone}}}
\]

**Unfairness**

\[
\text{Maximum Slowdown} = \max_i \frac{\text{IPC}_{i,\text{alone}}}{\text{IPC}_{i,\text{shared}}}
\]

• Methodology

  – Core model
    • 4 GHz processor, 128-entry instruction window
    • 512 KB/core L2 cache
  – Memory model: DDR2
  – 96 multiprogrammed SPEC CPU2006 workloads
Previous Work

**FRFCFS** [Rixner et al., ISCA00]: Prioritizes row-buffer hits
- Thread-oblivious \(\rightarrow\) Low throughput & Low fairness

**STFM** [Mutlu et al., MICRO07]: Equalizes thread slowdowns
- Non-intensive threads not prioritized \(\rightarrow\) Low throughput

**PAR-BS** [Mutlu et al., ISCA08]: Prioritizes oldest batch of requests while preserving bank-level parallelism
- Non-intensive threads not always prioritized \(\rightarrow\) Low throughput

**ATLAS** [Kim et al., HPCA10]: Prioritizes threads with less memory service
- Most intensive thread starves \(\rightarrow\) Low fairness
Results: Fairness vs. Throughput

Averaged over 96 workloads

Better fairness

Maximum Slowdown

Weighted Speedup

Better system throughput

TCM provides best fairness and system throughput

TCM provides best fairness and system throughput
Results: Fairness-Throughput Tradeoff

When configuration parameter is varied...

Better fairness

Better system throughput

TCM allows robust fairness-throughput tradeoff
Operating System Support

• *ClusterThreshold* is a tunable knob
  – OS can trade off between fairness and throughput

• Enforcing thread weights
  – OS assigns weights to threads
  – TCM enforces thread weights within each cluster
Outline

- Motivation & Insights
- Overview
- Algorithm

- Bringing it All Together
- Evaluation
- Conclusion

Fairness

Throughput
Conclusion

• No previous memory scheduling algorithm provides both high system throughput and fairness
  – Problem: They use a single policy for all threads

• TCM groups threads into two clusters
  1. Prioritize non-intensive cluster → throughput
  2. Shuffle priorities in intensive cluster → fairness
  3. Shuffling should favor nice threads → fairness

• TCM provides the best system throughput and fairness
THANK YOU
Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior

Yoongu Kim
Michael Papamichael
Onur Mutlu
Mor Harchol-Balter

Carnegie Mellon
Thread Weight Support

• Even if heaviest weighted thread happens to be the most intensive thread...
  – Not prioritized over the least intensive thread
Harmonic Speedup

Better fairness

Better system throughput

Weighted speedup

Harmonic speedup

FRFCFS
STFM
PAR_BS
ATLAS
TCM
Shuffling Algorithm Comparison

• Niceness-Aware shuffling
  – Average of maximum slowdown is lower
  – Variance of maximum slowdown is lower

<table>
<thead>
<tr>
<th>Shuffling Algorithm</th>
<th>Round-Robin</th>
<th>Niceness-Aware</th>
</tr>
</thead>
<tbody>
<tr>
<td>E(Maximum Slowdown)</td>
<td>5.58</td>
<td>4.84</td>
</tr>
<tr>
<td>VAR(Maximum Slowdown)</td>
<td>1.61</td>
<td>0.85</td>
</tr>
</tbody>
</table>
## Sensitivity Results

<table>
<thead>
<tr>
<th>ShuffleInterval (cycles)</th>
<th>500</th>
<th>600</th>
<th>700</th>
<th>800</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Throughput</td>
<td>14.2</td>
<td>14.3</td>
<td>14.2</td>
<td>14.7</td>
</tr>
<tr>
<td>Maximum Slowdown</td>
<td>6.0</td>
<td>5.4</td>
<td>5.9</td>
<td>5.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of Cores</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>24</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Throughput (compared to ATLAS)</td>
<td>0%</td>
<td>3%</td>
<td>2%</td>
<td>1%</td>
<td>1%</td>
</tr>
<tr>
<td>Maximum Slowdown (compared to ATLAS)</td>
<td>-4%</td>
<td>-30%</td>
<td>-29%</td>
<td>-30%</td>
<td>-41%</td>
</tr>
</tbody>
</table>