MOESI-prime: Preventing Coherence-Induced Hammering in Commodity Workloads

Kevin Loughlin
Stefan Saroiu, Alec Wolman, Yatin A. Manerkar, Baris Kasikci
What is Rowhammer (RH)?

- Frequent ACTs of same DRAM row(s) can corrupt data in nearby rows
- ACT rate above **RH threshold** (ex: 20,000 ACTs/64 ms) can flip bits
Commodity Workloads: Dangerous ACT Rates

- **Motivation** Decreasing RH thresholds (fewer ACTs needed to flip bits)
  - Carefully-crafted, *malicious* code known to pose increasing danger

- **Key Contribution #1** Coherence-induced hammering
  - Common, *non-malicious* code can also yield dangerous ACT rates

- **Key Contribution #2** MOESI-prime coherence protocol
  - Mitigates coherence-induced hammering
Outline

• Background: Rowhammer, ccNUMA

• Problem: Coherence-Induced Hammering

• Mitigation: MOESI-prime

• Evaluation and Takeaways
Malicious Hammering

• Ex: repeatedly flush cache line in aggressor row to force DRAM accesses

<table>
<thead>
<tr>
<th></th>
<th>while(true)</th>
<th>flush(row1_addr)</th>
<th>read(row1_addr)</th>
</tr>
</thead>
</table>

Aggressor Row

Victim Row

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>C1</td>
<td>C2</td>
<td>C3</td>
<td></td>
</tr>
<tr>
<td>R0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>R1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>R2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>R3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Worst-case: every DRAM access requires row ACT
  • Additional techniques/conditions increase likelihood of row ACT
ccNUMA Can Change DRAM Access Frequency

- ccNUMA: cache coherency across multiple nodes (ex: sockets)
  - Each cache line has a single “local” node

- **Remote** LLC miss: go to local node
- **Local** LLC miss: check *memory directory*...
### State-of-the-Art ccNUMA: Multiple Directories

- Directories track cache line ownership across cores
  - Ex: is a core’s copy of a cache line **Modified**, **Invalid**, etc.?
- Separate directories track local/remote ownership

#### LLC-level directory: **local** node ownership

<table>
<thead>
<tr>
<th>LOCAL</th>
<th>REMOTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core A</td>
<td>Core A</td>
</tr>
<tr>
<td>L1+L2</td>
<td>L1+L2</td>
</tr>
</tbody>
</table>

#### Memory directory: **remote** node ownership

<table>
<thead>
<tr>
<th>LOCAL</th>
<th>REMOTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core B</td>
<td>Core B</td>
</tr>
<tr>
<td>L1+L2</td>
<td>L1+L2</td>
</tr>
</tbody>
</table>

- **Insight:** Memory directory state is only meaningful if the local node isn’t owner
Memory Directory Implementation

• Each cache line’s remote state co-located with line in DRAM

For today, two important memory directory states...
• A: snoop-All: line *might* be owned (dirty) on a remote node
• I: remote-Invalid: line not valid on any remote node
Outline

• Background: Rowhammer, ccNUMA

• **Problem: Coherence-Induced Hammering**

• Mitigation: MOESI-prime

• Evaluation and Takeaways
Identifying Coherence-Induced Hammering

- Platform: Intel dual-socket Skylake server (ccNUMA)
  - Used DDR4 bus analyzer to record memory traces

- Ran commodity workloads on single node and two nodes
  -Measured highest ACT rate observed for single row within 64 ms (DDR4)
  - Compared to RH threshold of 20,000 ACTs

- Ran additional micro-benchmarks to isolate hammering sources
  - See paper
ccNUMA Increases Highest Row ACT Rates

Commodity Benchmarks

Takeaway
Commodity workloads can produce dangerous ACT rates!
Common Across Benchmarks: Dirty Sharing

- Dirty sharing: cache line sharing with at least 1 writer
- Consider migratory sharing of lock-protected data

Migratory sharing occurs in commodity code!

<table>
<thead>
<tr>
<th>Thread A</th>
<th>Thread B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 while(true)</td>
<td>1 while(true)</td>
</tr>
<tr>
<td>2 acquire_lock()</td>
<td>2 acquire_lock()</td>
</tr>
<tr>
<td>3 write(shared_var)</td>
<td>3 write(shared_var)</td>
</tr>
<tr>
<td>4 release_lock()</td>
<td>4 release_lock()</td>
</tr>
</tbody>
</table>
Sources of Coherence-Induced Hammering

• Problem #1: Redundant Memory Directory Writes TODAY!

• Problem #2: Mis-Speculated DRAM Reads

• Problem #3: Downgrade Writebacks
Hammering Writes: Migratory Sharing

### Local write
- **Local Writer**: Modified
- **Remote Writer**: Invalid
- **Mem Dir State**: snoop-All (local owner, N/A)

### Remote write
- **Nodes**: LLC Dir State, Mem Dir State
- **Local Writer**: Invalid, Invalid
- **Remote Writer**: Invalid, Modified
- **Mem Dir State**: snoop-All

**WHY REDUNDANT?**
No processor state indicates memory directory is already in snoop-All.
Outline

• Background: Rowhammer, ccNUMA

• Problem: Coherence-Induced Hammering

• **Mitigation: MOESI-prime**

• Evaluation and Takeaways
“Prime” States to Avoid Redundant Writes

• Problem: processor can’t recognize memory directory is already snoop-All
  • snoop-All: cache line might be dirty on a remote node

• Fix: for “conventional” dirty processor coherence states, add “prime” states
  • Prime means memory directory in snoop-All, otherwise equivalent to conventional

<table>
<thead>
<tr>
<th>Modified</th>
<th>Modified-prime</th>
</tr>
</thead>
<tbody>
<tr>
<td>- dirty + read-write</td>
<td>- dirty + read-write</td>
</tr>
<tr>
<td>- Mem dir state unknown</td>
<td>- Mem dir state = snoop-All</td>
</tr>
</tbody>
</table>

• Similarity of conventional and prime states helps preserve correctness
MOESI-prime in Action

Remote write

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Local Writer</th>
<th>Remote Writer</th>
</tr>
</thead>
<tbody>
<tr>
<td>LLC Dir State</td>
<td>Invalid</td>
<td>Invalid</td>
</tr>
<tr>
<td>Mem Dir State</td>
<td>remote-Invalid</td>
<td></td>
</tr>
</tbody>
</table>

Local write

<table>
<thead>
<tr>
<th>Local Writer</th>
<th>Remote Writer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modified-prime</td>
<td>Invalid</td>
</tr>
<tr>
<td>snoop-All (local owner, N/A)</td>
<td></td>
</tr>
</tbody>
</table>

Remote write

<table>
<thead>
<tr>
<th>Local Writer</th>
<th>Remote Writer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid</td>
<td>Modified-prime</td>
</tr>
<tr>
<td>snoop-All</td>
<td></td>
</tr>
</tbody>
</table>

Modified-prime
- dirty + read-write
- Mem dir state = snoop-All
Outline

• Background: Rowhammer, ccNUMA

• Problem: Coherence-Induced Hammering

• Mitigation: MOESI-prime

• Evaluation and Takeaways
Evaluation

• Gem5 configurations modelled after major cloud provider’s settings
  • Compared MOESI-prime to MESI and MOESI baseline protocols
• Micro-benchmarks: MOESI-prime prevents coherence-induced hammering
• Commodity benchmarks: PARSEC-3.0 and SPLASH-2x
  • Many workloads exhibit >20,000 ACTs/64 ms to single row in baseline protocols

<table>
<thead>
<tr>
<th>(2-nodes) Average Metrics Normalized to MESI Baseline (Higher is Better)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metric</td>
</tr>
<tr>
<td>-------------------------</td>
</tr>
<tr>
<td>Decrease in Max ACTs</td>
</tr>
<tr>
<td>Exec Time</td>
</tr>
<tr>
<td>DRAM Power</td>
</tr>
</tbody>
</table>

**Takeaway**

MOESI-prime mitigates coherence-induced hammering, and can even slightly improve performance and power!
Recap

• Key Contribution #1: Coherence-induced hammering
  • Commodity workloads can yield dangerous ACT rates

• Key Contribution #2: MOESI-prime coherence protocol
  • Mitigates coherence-induced hammering

• Check out the paper for much more!
  • Ex: other sources of coherence-induced hammering, proof of correctness
Thanks to my awesome collaborators!

Stefan Saroiu

Alec Wolman

Yatin A. Manerkar

Baris Kasikci
Thanks for listening! Questions?

Scan for Full Paper