Improving Cache Performance by Exploiting Read-Write Disparity

Samira Khan, Alaa R. Alameldeen, Chris Wilkerson, Onur Mutlu, and Daniel A. Jiménez
Summary

• Read misses are more critical than write misses
  • Read misses can stall processor, writes are not on the critical path

• **Problem**: Cache management does not exploit read-write disparity

• **Goal**: Design a cache that favors reads over writes to improve performance
  • Lines that are *only written to* are *less critical*
  • *Prioritize* lines that service *read requests*

• **Key observation**: Applications differ in their read reuse behavior in clean and dirty lines

• **Idea**: Read-Write Partitioning
  • Dynamically partition the cache between clean and dirty lines
  • Protect the partition that has more read hits

• Improves performance over three recent mechanisms
Outline

• Motivation
• Reuse Behavior of Dirty Lines
• Read-Write Partitioning
• Results
• Conclusion
Motivation

- Read and write misses are not equally critical
- Read misses are more critical than write misses
  - Read misses can stall the processor
  - Writes are not on the critical path

Cache management does not exploit the disparity between read-write requests
Key Idea

- Favor reads over writes in cache
- Differentiate between read vs. only written to lines
- Cache should protect lines that serve read requests
  - Lines that are only written to are less critical
- Improve performance by maximizing read hits
- An Example

\[
\text{Rd A} \rightarrow \text{Wr B} \rightarrow \text{Rd B} \rightarrow \text{Wr C} \rightarrow \text{Rd D}
\]

A D Read-Only  B Read and Written  C Write-Only
An Example

LRU Replacement Policy

2 stalls per iteration

Read-Biased Replacement Policy

1 stall per iteration

Evicting lines that are written differently depending on performance requests
Outline

• Motivation
• Reuse Behavior of Dirty Lines
• Read-Write Partitioning
• Results
• Conclusion
Reuse Behavior of Dirty Lines

• Not all dirty lines are the same
• Write-only Lines
  • Do not receive read requests, can be evicted
• Read-Write Lines
  • Receive read requests, should be kept in the cache

Evicting write-only lines provides more space for read lines and can improve performance
Reuse Behavior of Dirty Lines

Applications have different read and write behavior. 9.4% lines are dirty and written.
Outline

• Motivation
• Reuse Behavior of Dirty Lines
• Read-Write Partitioning
• Results
• Conclusion
Read-Write Partitioning

• **Goal:** Exploit different read reuse behavior in dirty lines to maximize number of read hits

• **Observation:**
  – Some applications have more reads to clean lines
  – Other applications have more reads to dirty lines

• **Read-Write Partitioning:**
  – Dynamically partitions the cache in clean and dirty lines
  – Evict lines from the partition that has less read reuse

**Improves performance by protecting lines with more read reuse**
Read-Write Partitioning

Applications have significantly different read reuse behavior in clean and dirty lines.
Read-Write Partitioning

- Utilize disparity in read reuse in clean and dirty lines
- Partition the cache into clean and dirty lines
- Predict the partition size that maximizes read hits
- Maintain the partition through replacement
  - DIP [Qureshi et al. 2007] selects victim within the partition
Predicting Partition Size

- Predicts partition size using sampled shadow tags
  - Based on utility-based partitioning [Qureshi et al. 2006]
- Counts the number of read hits in clean and dirty lines
- Picks the partition \((x, \text{associativity} - x)\) that maximizes number of read hits
Outline

• Motivation
• Reuse Behavior of Dirty Lines
• Read-Write Partitioning
  • Results
• Conclusion
Methodology

• CMP$im x86 cycle-accurate simulator [Jaleel et al. 2008]
• 4MB 16-way set-associative LLC
• 32KB I+D L1, 256KB L2
• 200-cycle DRAM access time
• 550m representative instructions
• Benchmarks:
  – 10 memory-intensive SPEC benchmarks
  – 35 multi-programmed applications
Comparison Points

• DIP, RRIP: Insertion Policy [Qureshi et al. 2007, Jaleel et al. 2010]
  – Avoid thrashing and cache pollution
    • Dynamically insert lines at different stack positions
  – Low overhead
  – Do not differentiate between read-write accesses

• SUP+: Single-Use Reference Predictor [Piquet et al. 2007]
  – Avoids cache pollution
    • Bypasses lines that do not receive re-references
  – High accuracy
  – Does not differentiate between read-write accesses
    • Does not bypass write-only lines
  – High storage overhead, needs PC in LLC
Comparison Points: Read Reference Predictor (RRP)

• A new predictor inspired by prior works [Tyson et al. 1995, Piquet et al. 2007]

• Identifies read and write-only lines by allocating PC
  – Bypasses write-only lines

• Writebacks are not associated with any PC

<table>
<thead>
<tr>
<th>PC P: Rd A</th>
<th>PC Q: Wb A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wb A</td>
<td>Wb A</td>
</tr>
<tr>
<td>Wb A</td>
<td>Wb A</td>
</tr>
</tbody>
</table>

Allocating PC from L1

Mask set at PC along whole page identifies PC read, again
Single Core Performance

Differing read vs. write-only lines improves performance over recent mechanisms.

- RWP performs within 3.4% of RRP, but requires 18X less storage overhead.

Speedup vs. Baseline

- DIP
- RRIP
- SUP+
- RRP
- RWP

48.4KB vs. 2.6KB

Performance:
- +8% for RRP
- +4.6% for RWP

Differences in read/write balance on RRP improve performance by 18X less storage overhead.
4 Core Performance

Differences in whether vs. over migrating lines improves performance for memory intensive mechanisms.
Average Memory Traffic

Increases writeback traffic by 2.5%, but reduces overall memory traffic by 16%
Partition size varies significantly for some benchmarks
Partition size varies significantly during the runtime for some benchmarks
Outline

• Motivation
• Reuse Behavior of Dirty Lines
• Read-Write Partitioning
• Results
• Conclusion
Conclusion

- **Problem**: Cache management does not exploit read-write disparity
- **Goal**: Design a cache that favors read requests over write requests to improve performance
  - Lines that are *only written to* are *less critical*
  - **Protect** lines that serve *read requests*
- **Key observation**: Applications differ in their read reuse behavior in clean and dirty lines
- **Idea**: Read-Write Partitioning
  - Dynamically partition the cache in clean and dirty lines
  - Protect the partition that has more read hits
- **Results**: Improves performance over three recent mechanisms
Thank you
Improving Cache Performance by Exploiting Read-Write Disparity

Samira Khan, Alaa R. Alameldeen, Chris Wilkerson, Onur Mutlu, and Daniel A. Jiménez