Improving Cache Performance using Victim Tag Stores

Vivek Seshadri, Onur Mutlu, Todd C Mowry, Michael A Kozuch
Caches

• Exploit temporal and spatial locality
• Reduce latency of memory accesses

• Concerns
  – Increasing off-chip memory latencies
  – Multi-core system
    • Shared last-level caches
    • Increased demand on main memory
    • Cache utilization more important
Problem

• Most of prior work
  – Primarily focused on replacement policies
  – Followed a single insertion policy for all blocks

• Comparatively little work on cache insertion policies

Goal: Improve the cache insertion policy and consequently, system performance
Cache Insertion Policy

Question: With what priority should a missed block be inserted?

Highest priority: Block stays in the cache for a long time
Lowest priority: Block gets evicted immediately

Neither policy is good for all the blocks
Our Approach

• Predict and insert
  – Predict the temporal locality of a missed block
  – Good-locality blocks: Insert with high priority
  – Bad-locality blocks: Insert with lowest priority

• Prediction mechanism: Key Insight
  – Good-locality block: If prematurely evicted, will be accessed again soon after eviction
  – Bad-locality block: Will not be accessed for a long time after getting evicted
VTS-Cache: Mechanism

Evicted block Address

Victim Tag Store (VTS)
(List of recently evicted block addresses)

Cache

Yes
- Prematurely evicted
- Good locality block
- Insert with high priority

Present in VTS?

No
- Bad locality block
- or first access
- Insert with lowest priority

Cache miss

Missed block address
Improving Performance

Applications with large working sets

- Large reuse distance
- Almost all blocks will be predicted bad-locality
- Inserting with lowest priority => thrashing
- Use bimodal insertion policy instead

Bimodal Insertion Policy: High priority with a low probability and lowest priority with a high probability

Insert predicted bad-locality blocks with the bimodal insertion policy
Improved VTS-Cache

Victim Tag Store (VTS)
(List of recently evicted block addresses)

Evicted block Address

Yes
- Prematurely evicted
- Good locality block
- Insert with high priority

No
- Bad locality block
- or first access
- Insert with lowest priority
- Insert with BIP

Present in VTS?

Cache miss  Missed block address
Improving Robustness

• LRU-friendly application
  – LRU is the best performing policy
  – Blocks that are just accessed have high locality
  – VTS predicts bad-locality on first access
  – Increases miss rate

• LRU-friendly workloads
  – All applications in the workload are LRU-friendly
  – Detect this situation using set-dueling
  – Disable VTS for LRU-friendly workloads
Improved Robust VTS-Cache

Evicted block Address

Victim Tag Store (VTS)
(List of recently evicted block addresses)

Present in VTS?

Yes

- Prematurely evicted
- Good locality block
- Insert with high priority

No

- Bad locality block
- or first access
- Insert with lowest priority
- Insert with BIP

Cache miss

Missed block address

Always high priority

-VTS priority
Practical Implementation

• Naïve implementation
  – Set-associative similar to main tag store
  – High storage overhead
  – High lookup latency and power consumption

• Implementing VTS using a Bloom filter
  – Bloom filter: Compact representation of a set
  – Inserts always succeed (unlike hash tables)
  – Test can have false positives (no correctness issues)
  – Easy to implement in hardware
  – Storage overhead: Less than 1.5% (of entire cache)
VTS-Cache: Final Look

1. Evicted block address → Insert into Bloom Filter, increment Counter
2. Missed block address → Test the filter, return prediction
3. When counter reaches max: Clear filter and reset Counter

Evicted block address

Missed block address

Always high priority

VTS priority

Cache

Bloom Filter

Counter
VTS-Cache: Benefits

1. Only adds a Bloom filter and a counter
   Easy to implement.

2. Does not modify cache structure
   Easy to design and verify

3. Operates only on a cache miss
   Works well with other mechanisms
Methodology

• System
  – Three level cache hierarchy
  – 32 KB L1 cache (private), 256 KB L2 cache (private)
  – Shared L3 cache (size depends on experiment)

• Benchmarks

• Multi-programmed workloads
  – Varying intensity and cache space sensitivity
  – 208 2-core workloads
  – 135 4-core workloads
# Prior Approaches

<table>
<thead>
<tr>
<th>Policies that follow per-application insertion policy</th>
<th></th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Policies that follow per-block insertion policy</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-Usage Block Prediction: <strong>SUB-P</strong> (Seznec – ACSAC 2007)</td>
<td></td>
</tr>
<tr>
<td>Run-time Cache Bypassing: <strong>RTB</strong> (Johnson – ISCA 1997)</td>
<td></td>
</tr>
<tr>
<td>Adaptive Replacement Cache: <strong>ARC</strong> (Megiddo – FAST 2003)</td>
<td></td>
</tr>
</tbody>
</table>
Main Results

• 2-core (1 MB last-level cache)
  – 15% better performance compared to LRU
  – 6% compared to TA-DRRIP (best previous mechanism)

• 4-core (2 MB last-level cache)
  – 21% better performance compared to LRU
  – 8% compared to TA-DRRIP

• Single core (1 MB last-level cache)
  – 5% better than LRU
  – On par with TA-DRRIP
2-core Performance

![Graph showing normalized weighted speedup vs workload number for different algorithms: TA-DIP, TA-DRRIP, VTS, and D-VTS. The graph illustrates the performance comparison across a range of workload numbers, with D-VTS showing the highest speedup.]
4-core Performance

- TA-DIP
- TA-DRRIP
- VTS
- D-VTS
Block-level Insertion Policy Approaches
Conclusion

• VTS-Cache
  – Predict temporal locality for missed block
    • how recently the block was evicted?
  – Determine insertion policy based on the prediction
    • Good-locality: High priority
    • Bad-locality: Bimodal insertion policy

• Provides significant performance improvements
  – Mechanisms with per-application insertion policies
  – Other mechanisms with block-level insertion policies
VTS with Bloom Filter + Counter

- Initialize
  - Clear Bloom filter and counter
- On an eviction
  - Inserted evicted block address into the filter
  - Increment counter
- On a miss
  - Test the Bloom filter for the missed address
  - Return prediction
- When counter reaches max (preferred VTS size)
  - Clear Bloom filter and counter
Single Core Results
Fairness Results: 2-core

The chart shows the normalized maximum slowdown for different workload groups and sensitivity levels. The workload groups include low, medium, and high intensity, as well as server and all. The chart compares four different techniques: TA-DIP, TA-DRRIP, VTS, and D-VTS.