

Cache Replacement Championship

Abeer Agrawal | Joe Berman

Fall 2011 18-740 Computer Architecture

What is the CRC?

The Cache Replacement Championship is an annual competition whereby entrants compete to develop the best cache replacement algorithms within a set of size constraints. The competition is led by individuals from Intel, IBM, Microsoft, NC State and Georgia Tech. There are two tracks to the competition, a single core and a multi core track. We focused on developing multicore algorithms, as they are a more important and expanding field of research.

Overview

Cache replacement policies have been a hot topic of research for many years, as memory latencies have grown longer because of faster processors and deeper cache hierarchies. Current last level caches have miss penalty times on the order of hundreds, if not thousands, of cycles, which means that last level cache misses can be particularly damaging to performance. We looked into several leading algorithms in order to ascertain useful techniques for developing the most effective last level cache replacement policy.

Methodology

We used a simulator, CMP \$im, that was provided to us by the CRC. We used the following configuration, as specified by the CRC:

- Our cache algorithm was used in the Level 2 (last level) cache
- The last level cache is 4MB in size and consists of 16-ways
- The last level cache is shared by all 4 cores
- All misses are treated the same and are assumed to have the same latency

We created traces from SPEC2006 benchmarks. All traces generated had 100 million instructions, and were generated after warming up the cache for 40 billion instructions.

Algorithms

We evaluated the following algorithms as our baselines for improvement. They use several different techniques to improve the performance of caches.

- Victim Tag Store: Stores the last n evicted tags in a FIFO queue. If upon insertion of a block, it's tag is found in the queue, insert the block into the most recently used (MRU) position, otherwise insert it into the least recently used (LRU) slot.
- **DRRIP**: Uses set dueling to determine how far in the future a block will be used by inserting either far in the future, to the LRU slot, with random insertions to the LRU-1 slot or just to the LRU-1 slot. Increments LRU stack counters to find the next block to be evicted from the cache.
- **TADIP**: Dynamically chooses using set dueling between inserting blocks at LRU (Linear Insertion policy) or mostly at LRU with random insertions at MRU (Bimodal Insertion Policy)

We implemented these algorithms in order to effectively gain an understanding of how they worked. After a thorough analysis, we were able to modify and improve upon the performance of them for certain workloads. We took these traces and generated mixes of traces based upon the sensitivity of the trace to cache size. The sensitivities were determined through the analysis of data by other research papers, as well as of our own testing.

Results are based upon speedup in CPI seen over the baseline of the Least Recently Used replacement policy.

- **Reference Set**: Breaks all of the ways into two sets, referenced and not-referenced. Additionally uses dynamic bypassing, random promotion and aging of blocks to change which reference set they are in. Only evicts from the not-referenced set.
- LRU: Used as a baseline against which to compare results. Insert new blocks at the MRU slot and evict blocks from the LRU slot.

Analysis & Results



We believed that inserting blocks into positions in the middle of the stack would be beneficial to cache performance. As you can see, we identified a maximum speedup across all of our mixes when we used set dueling to insert either at the MRU position or at the 11th position in the LRU stack. Inserting at the MRU position is the same as the traditional LRU insertion policy, which is most effective for workloads that have high temporal locality. Inserting at the 11th position is good for thrashing workloads, which have a working set larger than the size of the cache. In this case, if a block is unused within the next 5 evictions to that set, that original cache block itself is evicted. This provides more time for the block to be reused than in either DRRIP or TADIP, which insert blocks very close to or at the end of the LRU stack. It is a tradeoff between maintaining enough space in the cache for blocks to be re-referenced.

Initially, we tested the above algorithms for various benchmarks. The best performing algorithms were DRRIP and Refset. However, DRRIP and TADIP performed extremely poorly for the sensitive trace Astar.

We then tested a Thread Aware implementation of the RefSet algorithm, since we had noticed that thread awareness was extremely beneficial in converting single-core algorithms to multi-core ones. However, the results of this thread-aware RefSet were not very good.

Following this, we wanted to investigate the potential benefits of inserting into positions other than MRU and LRU. To do this, we tested our set dueling algorithm, which picked between inserting at MRU or in the middle of the LRU stack, depending on how the leader sets were doing. We noticed an improvement in the performance of various traces, especially Astar.

We believe that our best algorithm performs better on Astar than either DRRIP or TADIP because it allows for a longer amount of time for blocks to be re-referenced.

CPI for Different Algorithms for Astar



Special Thanks To:

Professor Onur Mutlu, TAs Justin Meza and Yoongu Kim and Vivek Seshadri

