Cache Replacement Policy
Lab2-Advance Memory Systems

Ramyad Hadidi

January 28, 2015

Abstract

This is the report for Lab2 assignemt for Advance Memory Systems (ECE-7103A) taught by Moinuddin Qureshi on Spring 2015. This assignment is about cache replacement policies. In this assignment three new policies is implemented and their metrics are compared with each other. Also, their results is compared with traditional cache replacement policies like LRU and Random. The policies that are implemented are DRRIP[2] and SHiP-PC[4] plus another replacement policy based on student's preference. These policies are tested with a set of Spec2006 benchmarks. The system for the experiments is a 4-core out-of-order CMP (Chip Multiprocessor) with three level of cache. Our policy is implemented on a 4 MB set-associative Last Level Cache (LLC). The metrics we used are number of misses and Cycles Per Instruction (CPI). The rest of details could be find in the report.

1 Simulation

Simulation is done with CMP$im [1] simulator. Only the source file for cache replacement is modified (`replacement_state.cpp` and `replacement_state.h`). Simulation is done for a 4-core system which each processor has an 8-stage, 4-wide pipeline. Each processor has its own data and instruction L1 cache. Dcache is 32 KB 8-way set associative with LRU replacement. The L2 data cache is 256 KB, 8-way set-associative with true LRU replacement. The Last Level Cache (LLC) is a 16-way cache with our policy. All simulations in this report are done with 4-threads and 200 M instruction for each. If each thread done its 200 M instruction it will reload its trace again, until all threads are done with their 200 M instructions. In this simulation we **do not** warm-up the cache. However for the contestant policy in the final section I used 10 M instruction as warm-up.

Table 1 shows our benchmark mixes.

Set	Name
mcf-mcf-mcf-mcf	mcf
bwaves-bwaves-bwaves-bwaves	bwaves
bzip2-bzip2-bzip2-bzip2	bzip2
zeusmp-zeusmp-zeusmp-zeusmp	zeusmp
cactusADM-cactusADM-cactusADM-cactusADM	cactus
hmmer-hmmer-hmmer-hmmer	hmmer
gemsFDTD-gemsFDTD-gemsFDTD-gemsFDTD	gems
sphinx-sphinx-sphinx-sphinx	sphinx
mcf-bwaves-hmmer-sphinx	mix1
gemsFDTD-bzip2-zeusmp-cactusADM	mix2
mcf-gemsFDTD-bzip2-cactusADM	mix3

Table 1: Our benchmark in this report & their names

2 Implementation of SHiP-PC & DRRIP

We have implemented two policies based on previous papers. The first one is Dynamic Re-Reference Interval Prediction (DRRIP) which is proposed in [2]. DRRIP uses set dueling to dynamically choose between Static Re-Reference Interval Prediction (SRRIP) and Bimodal Re-Reference Interval Prediction (BRRIP). In our Implementation we use the Hit Priority(HP) version. In RRIP policy we have M-bit of Re-Reference Prediction Value (RRVP) for each cache line. This value act as a metric for victim selection. A lower value represents a cache line that might be used in near-future. So, in every eviction we choose the line maximum RRVP value (RRIP-MAX) as a victim. In insertion, we insert new line with RRVP value of $RRIP - MAX - 1$. In each hit, we improve that line by giving its RRVP a value of zero. Since this report purpose is not describing every detail of the policies we will skip the details.

The second policy is Signature Based Hit Predictor (SHiP)[4]. In this policy we simply keep a signature for each cache line to represent its behavior. This signature could be PC, memory address or instruction sequence. In order to use less storage, the paper uses hash function to minimize storage overheads. SHiP is the same as SRRIP, however on misses if it finds the line in its table (Signature History Counter Table, SHCT) it will insert with higher priority. In this assignment we have implemented the SHiP-PC which uses PC as lines' signatures.

Table 2 shows our parameters setups for my SHiP-PC and DRRIP simulations. In section 4 we will compare the metrics for all the policies.

3 Contestant Policy

I chose Evicted-Address Filter (EAF)[3] as my third policy. I tired to mix this idea with RRIP, but the results was not better than ordinary EAF (This is

Parameter	Value
Number of Leader Sets	64
RRVP bits	2 bits
Dueling Counter bits	4 bits
Bimodal Insertion Probability	3.1%
Number of SHCT entries	16×1024
Number of Signature bits	14 bits
SHCT counter bits	3 bits

Table 2: Parameters for SHiP-PC & DRRIP

RRIP-EAF in results). In EAF we save the physical memory address of each evicted cache line in a Bloom Filter. Bloom filter acts as a table, but with more less storage overhead. A bloom filter can tell if it has not a value, but it cannot be sure that it has a value in it or not (False Positive). In every miss, if we had that address in our EAF we will insert it as MRU. If not, we will insert it as Bimodal (Always at LRU, on a rare probability at MRU). Figure 3 shows this policy. RRIP-EAF, mix SRRIP policy with EAF. Instead of insert to MRU and LRU positions we insert at RRIP-MAX and RRIP-Max-1. Also, I have implemented the Biomodal policy when we get a miss from EAF, therefore cache will have the chance to adopt to the new workload.

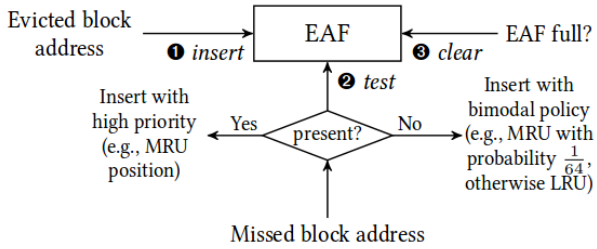

Figure 1: EAF Mechanism

4 Results

In this section we present our results from different policies. Figure 4 shows the improvement of number of misses. As it can be seen SHiP-PC gets the best result. The second best policy is could be both EAF version. However, in numbers D-EAF do better. Second graph shows the CPI for different polices for each benchmark. In the end, I have included the exact results in the file attached to this report.

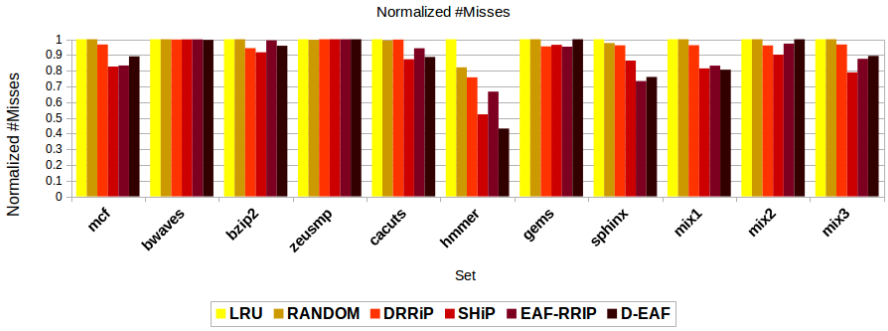

Figure 2: Normalized Number of Misses

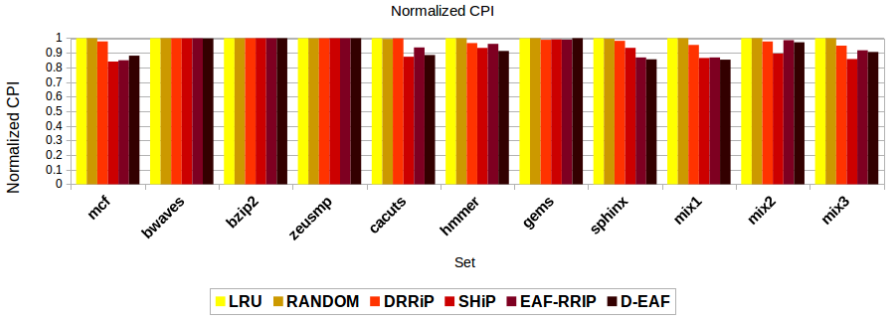

Figure 3: Normalized CPI

References

[1] A. Jaleel, R. S. Cohn, C.-K. Luk, and B. Jacob. Cmp$im: Apin-based on-the-fly multi-core cache simulator. In *In Proc. of the 4th Workshop on Modeling, Benchmarking and Simulation*.

[2] Aamer Jaleel, Kevin B. Theobald, Jr. Simon C. Steely, and Joel Emer. High performance cache replacement using re-reference interval prediction (rrip). In *ISCA '10 Proceedings of the 37th annual international symposium on Computer architecture*, 2010.

[3] Vivek Seshadri, Onur Mutlu, Michael A. Kozuch, and Todd C. Mowry. The evicted-address filter: a unified mechanism to address both cache pollution and thrashing. In *PACT '12 Proceedings of the 21st international conference on Parallel architectures and compilation techniques*.

[4] Carole-Jean Wu, Aamer Jaleel, Will Hasenplaugh, Margaret Martonosi, Jr. Simon C. Steely, and Joel Emer. Ship: signature-based hit predictor for high performance caching. In *MICRO-44 Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, 2011.