Study of State-of-the-art Cache Replacement Policies

Siddharth Jayashankar
170699

Abir Mukherjee
19111005

ABSTRACT

We present an empirical evaluation and analysis of a few state-of-the-art cache replacement policies namely - Dynamic Insertion Policy (DIP), Hawk-Eye and Signature based Hit Predictor (SHiP). The analysis compares the the performance of the replacement policies on a few SPEC CPU benchmark suites and discusses salient features of each policy.

KEYWORDS

Cache, Replacement Policy, LRU, DIP, SHiP, Hawk-Eye

1 INTRODUCTION

The time needed for main memory accesses is large when compared to the time period of the processor clock. This gap has been steadily widening. Thus reducing memory access time is a crucial to improve performance. The motivation for caching is the availability of fast access memories that can store only a small subset of main memory. The selection of this subset is crucial to performance of the processor.

Belady proved the optimal cache replacement policy [1]. However, Belady's OPT is impractical because it requires knowledge of the future. Simple replacement policies such as Least Recently Used (LRU) are based locality in memory access patterns. Several cache replacement policies have been proposed that attempt to improve cache hit rates.

In this study, we present an analysis of three state-of-the-art cache replacement policies:

(1) Dynamic Insetion Policy [2]
(2) Signature based Hit Predictor [3]
(3) Hawk-Eye [4]

All data was collected by running trace files on using the ChampSim [5] simulator. The ChampSim simulator configuration used in this study are :

Cores	1
LLC Cache	2MB ; 16 Way
Prefetcher	Off

2 DYNAMIC INSERTION POLICY

The LRU Replacement Policy is susceptible to thrashing for workloads that have a working set larger than the actual cache size. In case of a thrashing workload, the LIP (LRU Insertion policy) provides better performance. The Bi-modal insertion policy is an enhancement of LRU and frequently inserts an incoming block in the MRU position rather than the LRU position. Dynamic Insertion Policy (DIP) is implemented by selecting between the two policies using Set Duelling.

Implementation

The Dynamic Insertion Policy is implemented by partitioning the cache into groups of sets called constituencies. Two sets in each constituency are chosen as sampler sets - one for LRU and one for BIP. A saturating counter is used determine the replacement policy at a constituency level. Misses in the BIP sampler set increment the counter and misses in the LRU sampler decrement the counter. The MSB of the counter is used to determine the policy. The victim is the block in the LRU position in either policy.

One advantage of DIP is that is simple to implement and does not require much hardware overhead when compared to the default LRU. Only a sampler cache that stores tags and saturating counters are required.

For our implementation of DIP, we used a 10 bit saturating counter and divided the cache into 64 constituencies of 32 sets each. These parameter choices were motivated by the recommendations in the paper. [2].

Observations

We simulated the following traces on the ChampSim simulator for 1 Billion Instructions. The following are the results for the LLC Miss Rates.

Table 1: Miss Rates Per 1K Accesses

Trace	LRU	DIP
xalancbmk	927.87	**831.08**
astar	158.86	158.82
omnetpp	363.07	**281.13**
perlbench	68.95	73.96
gcc	**408.40**	424.38
gromacs	363.95	364.83

Traces xalancbmk and omnetpp show a big improvement in performance using DIP when compared to LRU. Whereas traces gcc shows a drop in performance using DIP. astar and gromacs retain more or less the same performance on both DIP and LRU.

Analysing the constituency wise sampler data provides some explanations.

Analysis

In traces xalancbmk and omnetpp , more than 80-90% of the constituencies have a strong preference (used for > 90% of all access to the constituency) for one of the two policies. Further, about 25% of the constituencies have a strong bias towards BIP. The presence of a large number of constituencies that are LRU averse causes the miss rate to decrease when BIP is used inplace of LRU in these constituencies. The DIP is able to identify these constituencies and apply BIP to them. This is responsible for the decline in miss rates in these two traces.

In gcc, only 50% of the constituencies have a strong preference for the LRU policy. None of the constituencies have a strong preference for BIP. Thus the potential for improvement in gcc using DIP is less. Further the significant number of constituencies that keep switching between policies increase the miss rate as adaptability is slow and a sub-optimal replacement policy operates in this period. Thus a mixed access pattern does not augur well with the DIP and can lead to an increase in miss rates.

Traces astar and gromacs have a very similar performance on both DIP and LRU because the DIP effectively implements LRU in all constituencies.

A disadvantage of the DIP is its slow adaptability. If the PSEL counter is deep into a single policy and the nature of the work load changes, the constituency will operate with an incorrect policy until the counter changes. Since the counter increments linearly, the constituency will have to incur many misses until the policy changes. This can significantly impact performance, particularly when the workload changes to an LRU friendly workload from an LRU averse workload.

The original paper's implementation using different sets in each constituency as the LRU and BIP samplers can result in an incorrect policy selection. An uneven distribution of cold misses over the two sampler sets can lead to an incorrect penalisation of the concerned policy and can significantly degrade performance, particularly when LRU is incorrectly penalised. Constituencies that have a significantly larger number of cold misses in their LRU sampler when compared to the BIP sampler have a large miss rate for the DIP when compared to LRU only. gcc has some constituencies where cold misses in the LRU sampler significantly outnumber the cold misses in the BIP sampler. Using the same sampler set for both LRU and BIP elimnates this factor. gcc shows a decrease in miss rate by 14.79 Misses Per 1K access when only the same set is used to sample LRU and BIP . xalancbmk and omnetpp , on the other hand, have a much more even distribution of cold misses and do not face this issue.

DIP is suitable for protecting LRU from Thrashing Workloads and provides an improvement over LRU in such cases. Since DIP reverts to LRU for non-thrashing workloads, its performance is bottle-necked by the performance of LRU for non-thrashing workloads.

3 SHIP: SIGNATURE-BASED HIT PREDICTOR FOR HIGH PERFORMANCE CACHING

SHiP is a signature based method for predicting future hits of a cache line. SHiP predicts the re-reference interval for a given signature. Accesses with the same signature are assigned the same re-reference interval. SHiP is evaluated using the Static RRIP replacement policy. LRU assumes a near-immediate re-reference interval for incoming blocks. This assumption results in performance loss for thrashing and mixed access patterns. SHiP deals with mixed access patterns, by diminishing the loss of frequently referenced working set cache lines, occurring due to scans.

Implementation

SHiP can use three types of signatures - SHiP-Mem, SHiP-PC and SHiP-ISeq. SHiP-Mem is a memory region based signature. SHiP-PC uses the Program Counter for prediction, and SHiP-ISeq uses the Instruction sequence as its signature. SHiP assumes that instructions with same signature will have same re-reference interval. This essentially creates buckets of instructions for every re-reference interval based on the similarity between them.

Every cache line has a field to store the signature and an outcome bit that is set when a cache line is referenced after insertion. For every signature, a saturating counter Signature History Counter Table(SHCT) is used to predict the reference interval for that signature. For a M-bit counter, a value of 0 indicates distant re-reference(DR) and a value of $2_M - 1$ indicates near immediate re-reference, while a value in between indicates intermediate re-reference(IR). On a cache hit, the outcome bit of the line is set and the SHCT value for the corresponding signature is incremented. On a miss, if the outcome bit was set low of the evicted line, then the SHCT value is decremented for the corresponding signature. The SHCT value of the signature of the incoming line is used to predict its re-reference interval. For practical hardware implementation, two optimizations are made. SHiP-S is a variant in which the sets are sampled instead of using the entire cache to reduce the amount of hardware required. Another optimised variant is SHiP-R, where the width of the saturating counters are reduced.

Observations

The above traces were run for SHiP. We observe that in traces xalancmbk & omnetpp, SHiP gives very high performance compared to LRU. For traces, gromacs and perlbench, the performance is similar to LRU while it degrades for other traces.

Table 2: LLC Miss Rates Per 1K Accesses

Trace	LRU	SHiP
astar	154.37	176.51
xalancbmk	929.83	**653.97**
omnetpp	300.37	**185.03**
gromacs	536.05	536.05
perlbench	106.21	107.13
gcc	**721.66**	864.10

Analysis

Unlike LRU, SHiP doesn't rely on spatial and temporal locality. This makes it suitable for LLCs as most of the locality is filtered out by the upper level caches. It has low hardware overhead compared to other modern methods and can be used with any ordered replacement policy such as SRRIP. It dynamically assigns re-reference intervals on incoming blocks rather than doing it naively as in LRU. This helps in separating DR lines from IR lines. It has very low misprediction rate for DR lines where misprediction overhead is high.

SHiP is dependent on the underlying replacement policy and hence the performance is dependent on the heuristic used, which may not be suitable for a particular access pattern. Also, once the outcome bit is set, the age of the cache line doesn't affect the SHCT value of the corresponding signature. Using this age information of cache lines may provide better performance in ordered replacement policies.

As can be seen, SHiP gives very good performance for traces xalancbmk and omnetpp. This can be attributed to SHiP's ability to predict re-reference intervals very well in mixed access patterns. SHiP recognizes working sets that will be re-referenced soon in mixed access patterns and predicts re-reference values that help the underlying SRRIP in keeping the working set in the cache. In the trace gromacs, SHiP performs similar to LRU. This happens when the scan length exceeds the underlying SRRIP threshold or the active working set has not been re-referenced before the scan in mixed patterns. In this case, SHiP degenerates to LRU.

In the others, where the performance is lower for SHiP, it happens because the predictions do not have much confidence. This happens when the signatures are not consistent in capturing the underlying access patterns due to large amount of non-recurring working sets. Here, mispredictions in re-reference interval occurs which leads to SRRIP performing worse than LRU.
Some workloads perform better on SHiP-Iseq while some on SHiP-PC. A possible extension of the above method is to adaptively select the SHiP variety in order to extract more performance. Another extension is to include the aging parameter for the SHCT values in order to model cache demand. Cache demand is another factor which determines when a line is evicted.

4 HAWK-EYE REPLACEMENT ALGORITHM

Unlike other state-of-the-art replacement algorithms that are based on specific access patterns, Hawkeye replacement algorithm is based on the behaviour of Belady's optimal algorithm. It applies Belady's OPT algorithm over past cache access in order to predict the behaviour of those load instructions in the future.

Implementation

It is made up of two components - OPTgen, that simulates the OPT algorithm over the past accesses, and a Hawkeye predictor which predicts the behaviour of future loads based on their PCs past behaviour. To simulate OPT, liveness intervals along with Set Dueling is used. Liveness intervals give information about both the reuse distance as well as the demand on the cache. Set Dueling is used to reduce the requirement of monitoring every set of the cache. It samples a few set, and then applies the result of the algorithm on the set to the entire cache. A history of 8x the cache size is maintained for the OPTgen algorithm to simulate OPT.
Usage interval of a reference X is defined as the interval or number of accesses between reference X and it's next reference. Liveness interval is the duration of cache-resident line. Based on these two definitions, a reference can be said to miss if the number of overlapping liveness intervals in the usage interval of the reference matches the cache capacity. Otherwise, it is a hit. OPTgen uses an occupancy vector to model the number of overlapping liveness intervals at a time. Occupancy vector is updated at next reuse of a line based on the overlapping liveness intervals in the usage interval of the reference. Size of the occupancy vector is reduced by increasing the granularity to a time quantum of multiple accesses.
The Hawkeye predictor uses the OPTgen algorithm to learn whether a PC corresponding to a load instruction will produce further hits or misses. If OPTgen determines that a particular line will lead more hits, then the PC which accessed that line will be trained positively, predicting that it will produce hits whenever re-encountered. The predictor has 8K entries, uses a 3-bit counter - representing the line's replacement state - for training and is indexed by a 13-bit hashed PC.
The high-order bit of the 3-bit counter determines whether a line is cache-friendly(1) or cache-averse(0). Hawkeye first chooses cache-averse lines for eviction, and when no such

line exists, it chooses the oldest cache-friendly line. As the working set changes, it is more likely that cache averse lines from the new working set are encountered before the cache-friendly lines of the old working set are evicted. In that case, cache-averse lines from the new set are evicted before old cache friendly lines are evicted, and thus making it easier to make correct predictions during phase transition. The predictor is retrained when cache-friendly lines are evicted to accommodate for phase change. A 3-bit counter based RRIP policy is used to determine the eviction priority of a line. Cache-averse lines have a value of 7 while the cache-friendly lines have a value of 0. At every cache access, this value is updated. On cache-replacement, the line with the highest counter value is chosen and the PC corresponding to the line is detrained in the predictor, if present in the sampler. This insertion policy differs from other RRIP policies. It gives more importance to the Hawkeye prediction than cache hits & misses, only cache-friendly lines are able to achieve 0 RRIP value, and cache-averse lines are always prioritised over cache-friendly ones.

Observations

Table 3: LLC Miss Rates Per 1K Accesses

Trace	LRU	Hawk-Eye
astar	154.37	**195.43**
xalancbmk	**929.83**	750.10
omnetpp	**300.37**	236.12
gromacs	536.05	536.05
perlbench	106.21	99.63
gcc	721.66	**828.98**

As can be observed, Hawk-eye performs very well for the traces xalancmbk and omnetpp. It's shows a dip in its performance for the other traces compared to LRU.

Analysis

Performance of Hawk-eye is dependent on it's reconstruction of Belady for the past accesses and using it to predict the correct re-reference interval.

traces like xalancmbk and omnetpp are LRU averse and Hawk-Eye is able to analyse the access pattern and make accurate predictions about the re-reference interval which causes the large drop in miss rates.

Like SHiP, this method predicts the re-reference intervals to be used by the underlying RRIP, and hence performs well on traces that SHiP performs well on. Its slight performance gain in perlbench can be attributed to the fact that, unlike SHiP, Hawk-eye is not based on any underlying assumption about the type of reuse. Hence, it can predict patterns for any reuse length.

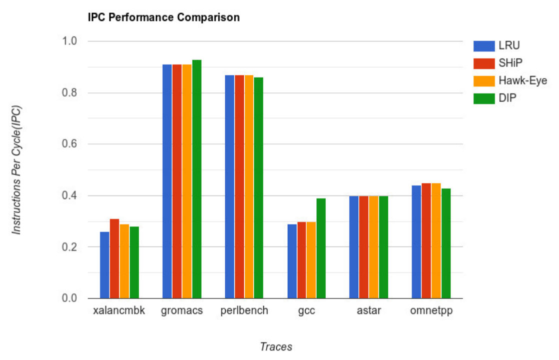

Figure 1: Comparative analysis of the LLC Cache Replacement Policies on the basis of IPC

Hawk-eye fails when its underlying assumption that past OPT decisions predict the future OPT decisions fails. This is seen gcc where access do not obey a lot of structure and pattern. In such a case, Hawk-predicts wrong re-reference intervals, and the underlying RRIP chooses the wrong victims. This leads to performance drop.

5 ACKNOWLEDGMENTS

We would like to acknowledge ChampSim for the simulator and code for the SHiP replacement policy. We would also like to acknowledge the 2^{nd} Cache Replacment Championship (CRC2) for the traces.

REFERENCES

[1] L. A. Belady. A study of replacement algorithms for a virtualstorage computer. In IBM Syst. J., volume 5, June 1966.
[2] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely Jr., and J. Emer. Adaptive insertion policies for high performance caching. In Proc. of the 35th International Symposium on Computer Architecture, 2007.
[3] Jean-Wu et al. SHiP: Signature-based Hit Predictor for High Performance Caching
[4] A. Jain and C. Lin. Back to the Future: Leveraging Belady's Algorithm for Improved Cache Replacement
[5] ChampSim Simulator. https://github.com/ChampSim/ChampSim
[6] Second Cache Replacement Chammpionship. https://crc2.ece.tamu.edu/