ASSIGNMENT-2: CACHE REPLACEMENT POLICIES

Shashank Nag EE19B118[1], **Sai Gautham Ravipati EE19B053**[1]
[1]Department of Electrical Engineering, Indian Institute of Technology, Madras

1 Introduction

The last level caches perform an important role in the case of multi-core systems. Further, since it is shared amongst the cores, it is important to have an efficient replacement policy for evicting the entries in order to store new entries. The existing cache replacement policies can be classified into the following categories - random replacement, queue-based, recency based, frequency based, re-reference interval prediction and those approximating Belady's MIN algorithm. In this assignment a few algorithms from the literature like Hawkeye, MockingJay and SHiP have been profiled and analysed. Further, other policies were experimented and profiled in comparison to the existing policies.

2 Existing Policies

The most frequently used replacement policy is LRU where the least recently used cache lines are discarded. The same has been profiled in comparison to that of other policies like SHiP, MockingJay and Hawkeye. The SHiP algorithm works on the principle of correlating the re-reference behaviour of a cache line to that of PC, memory and instruction-sequence based signature. While this is the case the other two are based on Belady's MIN algorithm of predicting the future re-use patterns based on the past access patterns. Hawkeye learns on the past accesses by the PC to predict whether a given access is cache friendly (used again later) or cache averse (not used again). This information is provided to a Re-reference interval predictor which performs the actual eviction decisions. MockingJay on the other hand performs a non-binary prediction which enables to perform decisions at a finer granularity. Further, it waits till sufficient information is available in-order to decide on the eviction of a cache line. The profiling results obtained for these policies are presented in Figure-1. From the plot, it can be inferred that Hawkeye performs better for most of the cases when compared to other policies. As per the literature MockingJay should be performing better in comparision to the other policies, but clearly, at-least for the first few policies it doesn't perform better while it does so for 'lbm'. This could be mainly because of the traces being used, as well as the number of instructions being used for profiling the replacement policy.

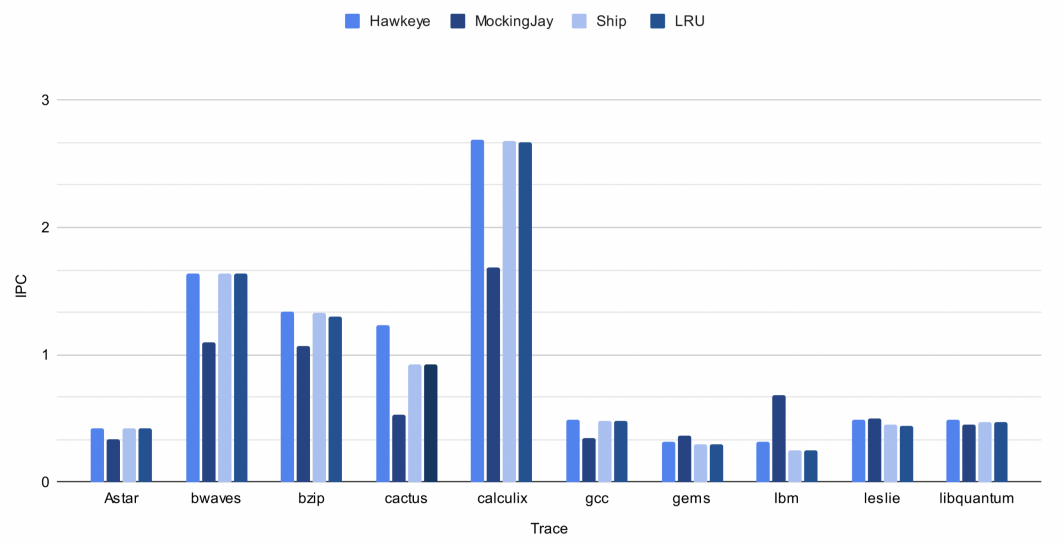

Figure 1: IPCs obtained by profiling different policies on SPEC benchmarks

3 Custom Policies

In the following few sections, several custom policies that have been experimented are presented. On observation, the performance of some of them was close to Hawkeye in some cases. They algorithm adopted has been described.

3.1 RePred - Towards Belady's MIN Cache Replacement Policy

The ideal case scenario cache replacement policy is governed by the Belady's MIN algorithm. With the knowledge of cache access patterns to the future, the Belady's MIN algorithm dictates victim identification for eviction from the cache. The Belady's

MIN states that the cache line that would be accessed farthest in the future, would be evicted. Due to the inherent non-causality in the algorithm, the Belady's MIN cannot be realised in the exact sense. Several prior cache replacement policies, including Hawkeye and MockingJay, have tried to closely model the behaviour of Belady's MIN using the consideration that at the time of eviction, we have seen the access pattern of the cache line - and hence we can now understand whether or not the line should have been cached in the past as per the Belady's MIN algorithm.

In this work, we propose a novel cache replacement policy, termed RePred, that seeks to predict the Reuse Distance of a line in the cache, and use the same to determine eviction candidates. Through this, we aim to closely mimic the Belady's MIN algorithm, and evict the line in the cache having the longest reuse time at a given instant. Although previous policies such as Hawkeye and MockingJay seek to realise the same, they use RRIP based scheme for victim selection. This limits the nature of information available while making the decision, with just the interval of reuse distance (and cache friendly/ averse line) being available for making decisions, rather than the actual reuse distance itself. This work seeks to address this gap and have a dynamic Reuse Distance Predictor, to take suitable decisions on selecting a victim.

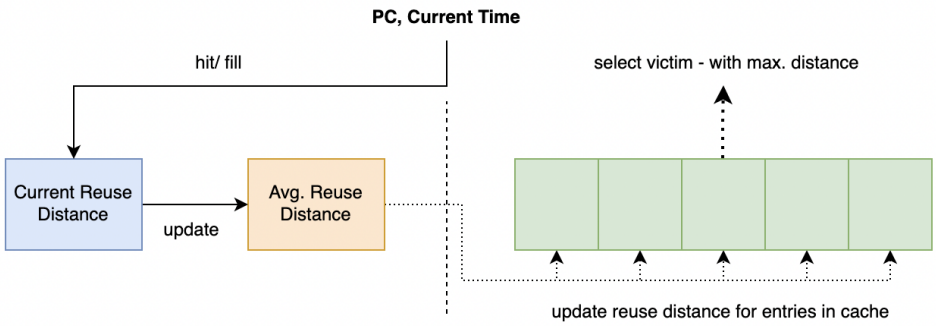

Figure 2: Updation of Reuse Distance Predictor state at every cache hit/ fill

The PC is mapped to a subset using a CRC and LSB based hash table. For each of these entries in the hash table, the last accessed time and the average reuse distance is stored. At every hit/miss, the states of these entries corresponding to the PCs are updated such that the last accessed time is replaced by the current time, and the average reuse distance is updated based on a weighted average. A separate $dist$ vector is used to store the expected reuse distance for each line in the cache. At every hit/ fill, the value of $dist$ is updated as per the computed average reuse distance. When a victim is to be selected, the $dist$ vector is scanned through to get the line with the maximum reuse distance.

3.2 LruEye - Dynamic Selection of Cache Replacement Policy

A single type of cache replacement policy may not be suitable for all access patters, while different policies might offer different advantages for certain access types. This part of the work takes this as the motivation and seeks to dynamically switch between cache replacement policies based on the current performance trends. For a given running policy, an associated counter is incremented for hits and decremented for misses. If the counter falls below a certain threshold, we switch the policy, and the counters are reset. However it is to be noted that though the victim is selected based on the "running" policy, the state vectors for both the policies need to be continuously updated to keep track of the lines in the cache and their behaviour. We have analysed the case of dynamic switching between Hawkeye and LRU cache replacement policies. Although this approach comes at the cost of additional hardware, the IPC is almost similar to that of Hawkeye and LRU indicating that, this approach could be more robust and could improve the IPC by dynamically changing the replacement algorithm. Further, this behaviour could be captured effectively by considering traces where Hawkeye performs worse than LRU. The performance of the previous two policies can be verified from the data presented in Figure-3.

Trace	Hawkeye	MockingJay	Ship	LRU	RePred	LruEye
Astar	0.421049	0.336868	0.424034	0.424508	0.314788	0.423848
bwaves	1.63776	1.0987	1.63645	1.6365	1.63498	1.63637
bzip	1.34329	1.0653	1.32969	1.30547	1.12899	1.31236
cactus	1.23464	0.533476	0.923474	0.921829	0.890554	1.00699
calculix	2.68999	1.68457	2.68247	2.66895	1.98172	2.6582
gcc	0.491106	0.350014	0.478259	0.477217	0.350515	0.475698
gems	0.319384	0.364602	0.295863	0.29575	0.291446	0.30789
lbm	0.322453	0.681144	0.252362	0.252413	0.251384	0.246097
leslie	0.490258	0.499474	0.454345	0.444685	0.412839	0.45231
libquantum	0.493416	0.450697	0.471654	0.471336	0.46326	0.460652

Figure 3: IPC data for different traces

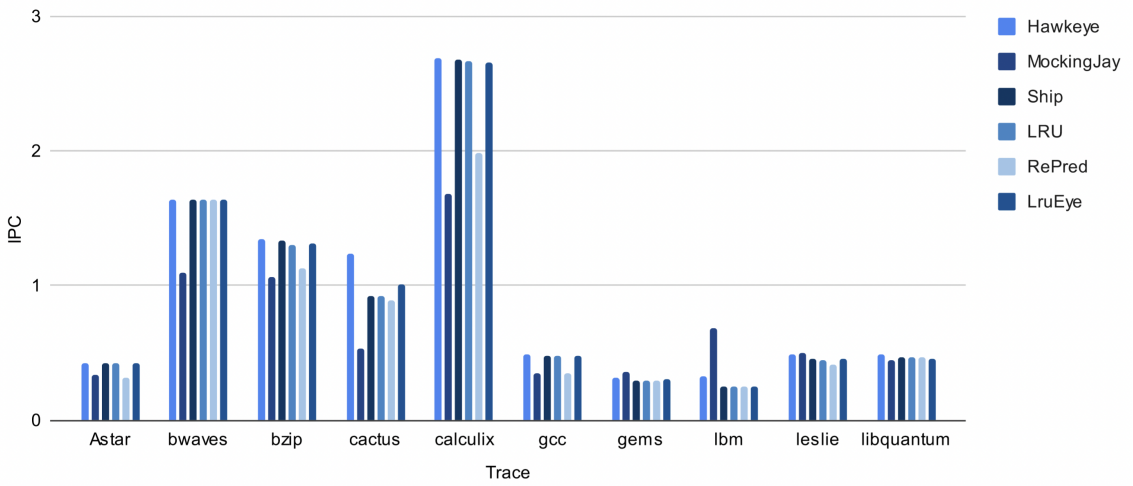

Figure 4: Plot of IPC vs Trace for different policies

3.3 Hawkway - Hawkeye + Leeway

Hawkeye pre-dominantly uses a RRIP counter to select victims for eviction. However, there exists other sophisticated policies to identify an optimal victim from the lines in the cache. For the same, we use a policy called Leeway for identifying the victim based on a reuse aware management policy. Leeway uses the stack distance to learn the temporal reuse characteristics of the cache blocks, to come up with a dead block predictor. At the same time, we allow Hawkeye to maintain its states and decide on cache fills that need not be cached, and could be bypassed instead. Due to portability issues, only a couple of traces were ran on the old version of CRC2. For the bzip trace available with the CRC2 version, Hawkway achieves an IPC of 0.809317, against a 0.805549 by Hawkeye.

4 Conclusion

In this assignment, multiple cache replacement policies have been profiled using the SPEC benchmarks. The following inferences were drawn:

- From the profiling, we could clearly see that the performance of the cache replacement policy is also heavily dependent on the program corresponding to the traces, and the underlying data access patterns.
- There is a no clear winner amongst the existing replacement policies. The state-of-the art Mockingjay algorithm performs better than Hawkeye only for a few cases, and performs pretty bad for the others. Similarly in some cases, Hawkeye gives a poorer IPC in comparison to the others. This is because all of these algorithms are just trying to mimic the golden standard of Belady's MIN, and a optimal result cannot be guaranteed.
- For some of the program traces, our custom replacement policies beats the current state-of-the-art Mockingjay replacement policy, and is comparable to Hawkeye replacement policy.

5 References

[1] Mockingjay : https://github.com/ishanashah/Mockingjay [2] Hawkeye : https://ieeexplore.ieee.org/document/9773195 [3] Leeway : https://github.com/faldupriyank/leeway [4] Cache Replacement Championship