Improving last level cache by Read Reference Prediction

Palash Parmar
palparmar@tamu.edu
Texas A&M University

Abstract: *In modern computer architecture, where processors has pushed the boundaries by providing very good performance, the memory management is failed to catch that performance. Memory is the bottleneck in providing great computation power. Memory hierarchy tries to improve the performance but cache misses are still hurt the performance of overall system. Different cache level have different penalties of misses, out of which last level cache has very high penalties. Good cache management mechanisms tries to provide good hit rates thus improving the performance. While most cache replacement policies tries to evict the lines from the cache which are not potentially used in future, none of the policy focus on the difference of lines which are reused by reads versus that which are reused by writes only. Lines which are reused by read are more critical than the former as the process requires it immediately or it stalls the processor. Utilizing this idea, this paper proposes Read Reference Predictor which bypasses the cache line which are unlikely to receive any read request in future. RRP mechanism improves the cache performance by 4.8% over traditional LRU policy.*

I. INTRODUCTION

In memory hierarchy, while L1 cache tries to provide very less latency to processor, LLC on the other hand ensures the hit of cache line when it is miss in high level cache. Hence LLC must have a good replacement policy to match the disparity between high level cache and DRAM. A good amount of research has exploited the performance from LLC by insertion/replacement algorithms to maintain cache line that are likely to be reused. Adaptive Insertion Policy [1] pointed out that while some workload are LRU-friendly, there are some workloads which gives poor performance with LRU policy due to thrashing of cache and proposed a dynamic insertion policy which select between LRU and BIP policy in run time to suit most of the workload.

In Promotion/Insertion Pseudo-Partitioning of Multi-Core Shared Caches, Yuejian et al. proposed new cache management approach that combines dynamic insertion and promotion policies which provides the benefit like cache partitioning, adaptive insertion in cache. Daniel A. Jiménez [2] tries to identify whether an incoming block is dead upon arrival to bypass it from the cache by sampling over few sets of cache.

While most of the above work tries to exploit potential cache line which are either more like to be used or less likely to be used, all assumes the cost of load or store on performance is equal. In the processor pipelines, both load and store are treated differently, with greater attention given to load because of its immediate use in computation, while stores are buffered for some time before finally committing to the cache or memory. While processor handle load and store differently, memory management on the other hand prioritize equally.

This paper proposes policy which favors more critical read miss over less critical write miss. While most of the prior work tries to distinguish lines which are likely to reuse versus which are not, this paper proposes to distinguish between lines that are reused by reads versus those which are recused only by write.

The organisation of this paper is follows.

II. READ LINES VS. WRITE-ONLY LINES

For making a policy mentioned above, the cache has to be divided into two category. The line which serves as read request and other which serves as to buffer write requests that will potentially not be read in future. Such classification will require future knowledge of the work load. One approach can be to predict the tendency of workload using past tendencies. Once the tendency of workload is predicted, a simple algorithm will prioritize the incoming access into respective category. Performance with such an organisation is degraded if a dirty line from a write category is reused again because it might be evicted early. Such an issue can be overcome by specifically distinguishing write reuse and write only line. To be more clear, dirty line are all the lines which are likely to be used again versus like which are just write and never referenced again. Such an organization's performance is largely depends on the type of workload. On studying SPEC CPU2006 benchmarks, almost 46% of

With LRU policy		With write on-allocate		With read based LRU policy	
Load A	C A E F	Load A	A E F D	Load A	A E F D
Load D Stall	A C E F	Load D	A E F D	Load D	A E F D
Store B	D A C E	Store B	D A E F	Store B	D A E F
Load B	B D A C	Load B Stall	D A E F	Load B	B D A E
Load D	B D A C	Load D	B D A E	Load D	B D A E
Store C	D B A C	Store C	D B A E	Store C	D B A E
Load A	C D B A	Load A	D B A E	Load A	D B A E
Load E Stall	A C D B	Load E	A D B E	Load E	A D B E
Load F Stall	E A C D	Load F Stall	E A D B	Load F Stall	E A D B
Load E	F E A C	Load E	F E A D	Load E	F E A D
Store C	E F A C	Store C	E F A D	Store C	E F A D
Load A	C E F A	Load A	E F A D	Load A	E F A D

Figure 1. Example: Drawback of not taking read write difference into account. A,D,E,F are read only access, B is read and written request and C is write only request.

the cache lines in LLC is dirty line and remaining 54% are clean lines. But our interest is more in dividing dirty cache lines into two parts i.e. there are almost 37% of the dirty cache lines which are not referenced again and 9% lines which are referenced after written to cache.

2.1 Dirty Read Lines

Organisation of distinguishing write reuse versus write only lines can be achieved by write no-allocate policy in hardware. Many workload in SPEC CPU2006 benchmarks have large amount if write lines which are reread again (eg. 401.bzip, 435.gromacs, 450.soplex etc.). Such a methodology can be best explained by figure 1. In figure 1, the example mention 6 cache access, out of which D, E, F are read only access, B is read and written to and C is only written to access. A sample iteration of loop is run on baseline LRU policy, only read based policy, and last with distinguish write reuse and write only access. This example best explain the advantage of proposed idea with the critical part begin distinguishing write reuse versus write only access. For this purpose, a novel Read Reference predictor is proposed.

III. READ REFERENCE PREDICTOR

Past work of Piquet et al. [3], building upon work of Tyson [4], observed that some cache lines are not reused versus other there that are reused. Piquet's policy mark the PC of memory access with a saturation counter which is incremented every time it reread cache line versus that which are not read at all. Whenever processor request new memory access, it first has to go to history table index by PC, if counter is 0, it's a no reuse cache line and bypassed by the cache. Read Reference Predictor is motivated by same methodology. Instead of targeting cache line for reused, Read Reference Predictor target only write reused line. With such an implementation, the proposed idea of differentiating write reuse versus write only cache lines is achieved. Such methodology can bypass the write only cache line from past experience. Another advantage of Read Reference Predictor is once the write only cache lines are eliminated, it can be backed by any good replacement/insertion policy to further boost the performance. In this paper's implementation, we used Adaptive Insertion Policy [1] for replacement/insertion of remaining cache lines. While Adaptive Insertion Policy emphasis on allocating small number of cache line to most recent position, performance can be further improved if instead of randomly allocating some cache line to most recent positions, Adaptive Insertion Policy can utilize the prediction table from RRP to identify candidate which are very likely to be used in future (PC of memory access whose saturation counter is max).

RRP uses additional bit for each cache line for storing a critical bit, which becomes 1 when there is read reuse. There is another table index with PC to track a saturation counter which provided a little hysteresis. Every time there is a cache hit with processor writing into the cache, the critical bit of cache line becomes 1. Whenever there is an eviction of cache line from the

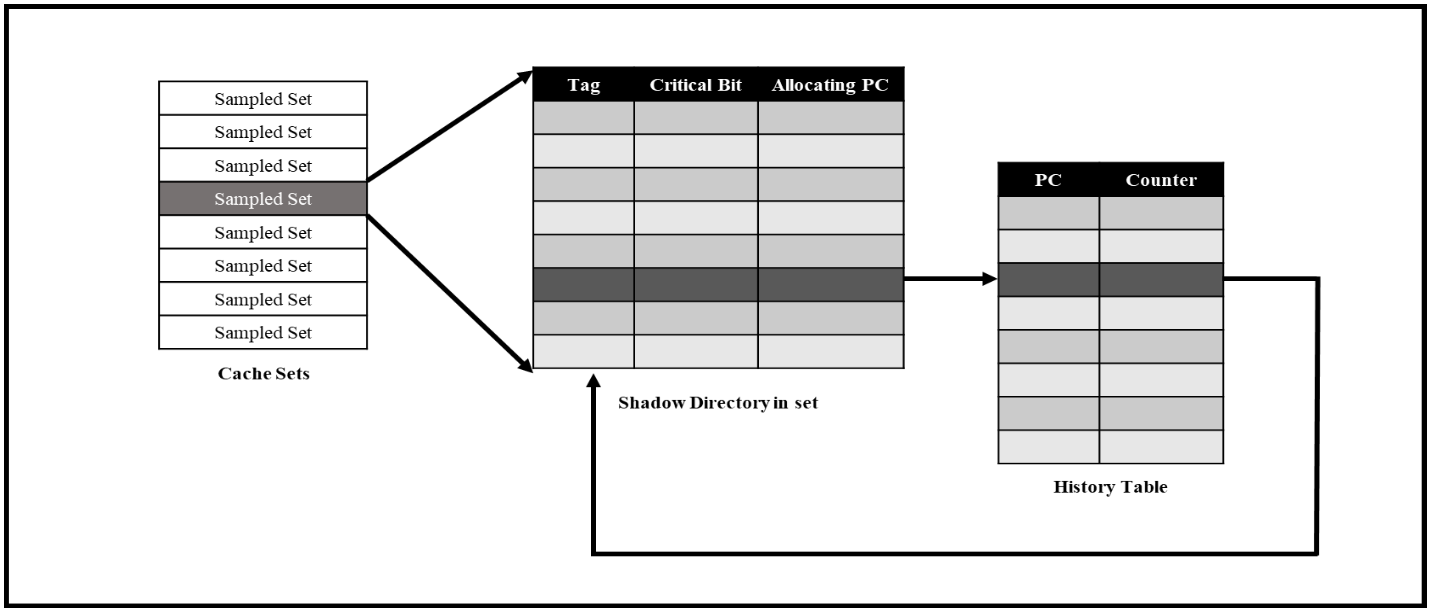

Figure 2. Overall workflow of Read Reference Predictor (RRP)

cache, the counter of respective PC is incremented or decremented according to the critical bit. Every time there is cache miss, the cache management policy will identify the counter of history table using PC, and if it is zero, the line is simply bypassed the cache. If it is not zero, the victim is evicted from cache using Adaptive Insertion Policy. The same saturation counter is used to help BIP place the incoming line at MRU or LRU. If the counter is max, it is likely to be used very often and placed at MRU, rest are placed at LRU.

IV. RESULTS

This section summarize the results of RRP when compared with baseline LRU policy. Results are shown for single core, without prefetching and uses standard SPEC CPU2006 Benchmark. Figure shows the bar graph of IPC with respect to LRU for all the traces. RRP achieved an overall geometric mean increment of 4.86% over LRU policy. Our worst case performance is shown by GemsFDFT and zeusmp with decrement of 6.3% over LRU. Our best case performance is shown by sphinx3 workload with increment of 86.7% over LRU policy. The significant reduction on some workloads can be because of the sampling errors in the predictor. Performance from sphinx3 benchmark is due to the fact that most of the lines in this workload is write only lines and RRP bypassed such line while processor request it.

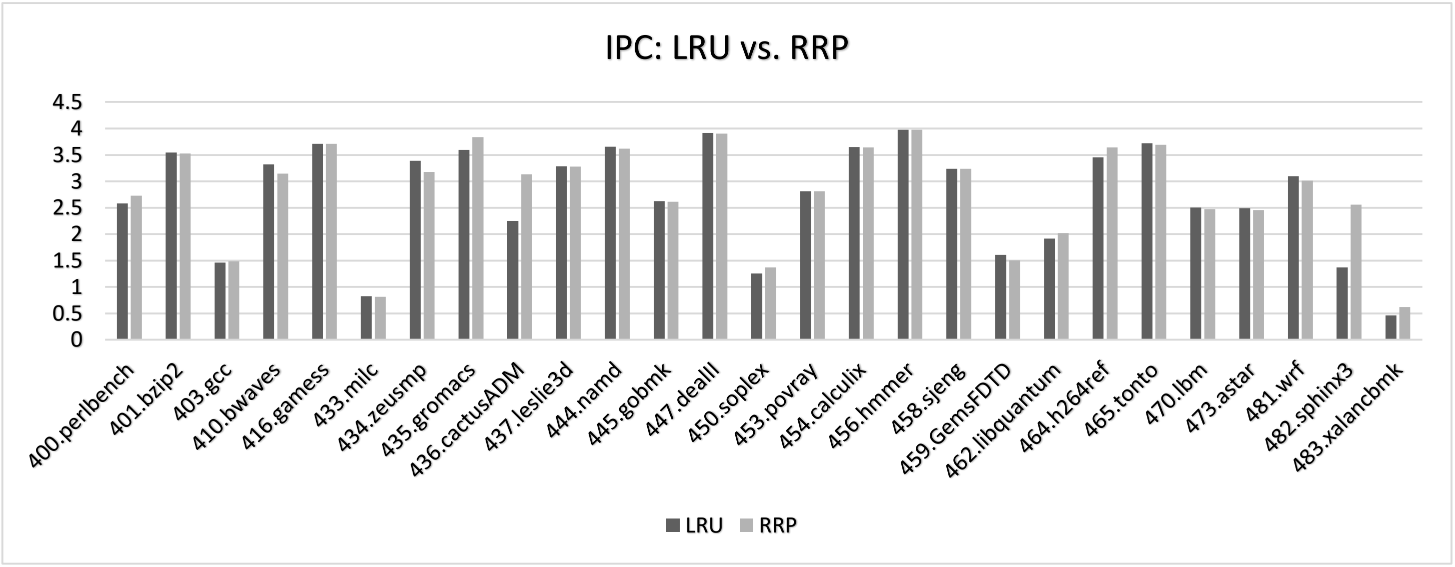

Figure 3 IPC comparison of LRU vs. RRP on SPEC CPU2006 benchmark

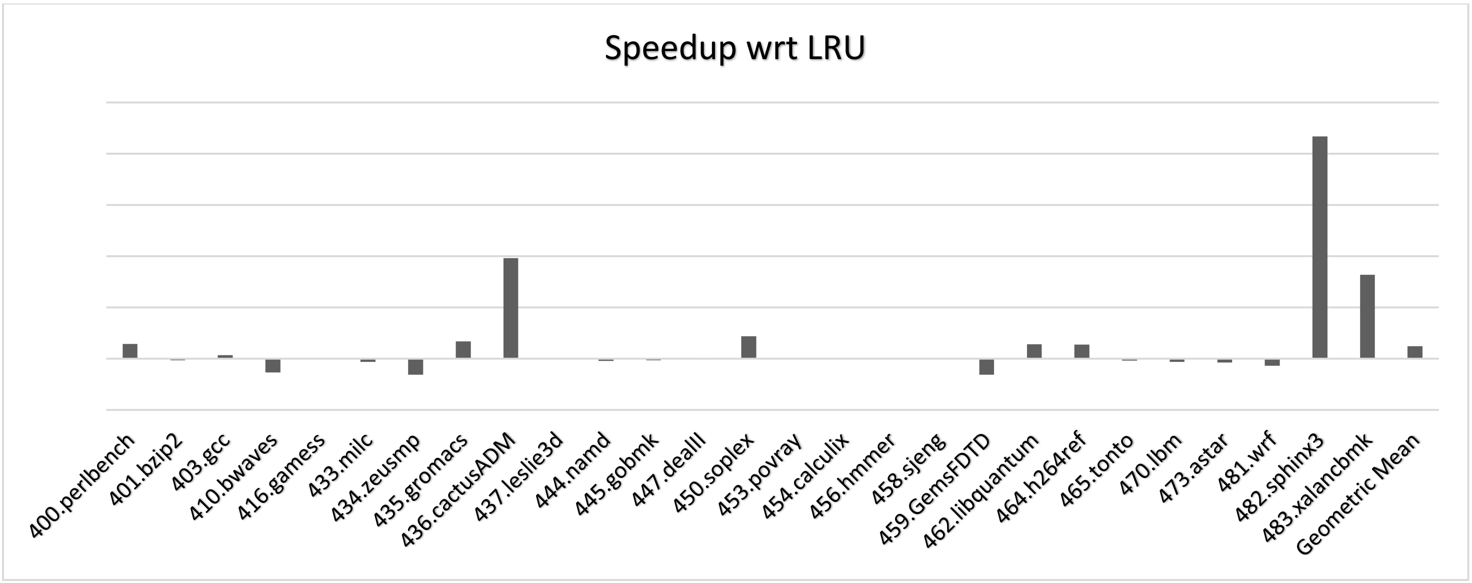

Figure 4 Speedup of LRU vs. RRP on SPEC CPU2006 benchmark

V. CONCLUSION

In this paper, I tried to prioritize the read request more than the write request because miss in read request degrade the IPC more than write request. This work increases probability of critical read miss at the cost of less critical write miss, increases the overall IPC of system. Key contribution in this paper is the distinguishing between read reuse versus write only cache lines and there effect on the performance of the cache. Read Reference Predictor extract the differentiation using more complex yet efficient approach. RRP uses PC to identify the past trend of particular memory access and predict the access accordingly. RRP is backed by Adaptive Insertion Policy for placing new incoming line into the cache. Adaptive Insertion Policy uses prediction table of RRP to place more likely cache line at MRU instead of LRU. Such practice further improve the performance as this prediction is better than randomly choosing lines to be placed at MRU.

VI. ACKNOWLEDGEMENT

I would like to thank Professor **Daniel A. Jiménez** who gave the opportunity to make this project through which I gained valuable knowledge on topic Cache Replacement Policy. His constant guidance and motivation helped in completing the project successfully.

X. REFERENCES

1. M. Qureshi et al. Adaptive insertion policies for high performance caching. ISCA, 2007
2. S. M. Khan, Y. Tian and D. A. Jimenez, "Sampling Dead Block Prediction for Last-Level Caches," 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture, Atlanta, GA, 2010, pp. 175-186.
3. T. Piquet et al. Exploiting single-usage for effective memory management. ACSAC, 2007.
4. G. Tyson et al. A modified approach to data cache management. MICRO, 1995.
5. Samira Khan, Alaa R. Alameldeen, Chris Wilkerson, OnurMutlu and Daniel A. Jimenez. Improving cache performance using read-write partitioning. HPCA, 2014.