Cache Replacement Strategies for Exclusive Cache Hierarchies

Tapojyoti Mandal and Manav Gurumoorthy

Department of Electrical and Computer Engineering, Texas A&M University
{tapojyoti.mandal,mana}@tamu.edu

Abstract—While Computer Processors have been steadily getting faster, the memory subsystem of the computer has not been able to keep pace. Owing to the complexities involved in making processors faster, a lot of the attention to achieve improvements in performance have moved over to making the memory sub-system faster. One such optimization is that of an efficient cache replacement policy. Our study involves the use of some different approaches to see how cache replacement can be optimized for an exclusive cache hierarchy. The advantage of an exclusive cache hierarchy is that the size of the cache is the sum of the different levels in the hierarchies, while in an inclusive cache it is typically the size of the last level cache. The techniques we looked at were designed for an inclusive cache but we have implemented them for our exclusive cache hierarchy to see how they fare and understand the challenges of using an exclusive cache hierarchy.

Index Terms—Computer Architecture, Multilevel Cache, Exclusive Cache, Cache Replacement, LRU

1 Introduction

The objective of this work is to find an efficient cache replacement strategy for an exclusive cache hierarchy. The policy implemented was tested on a test-bench that implements an L1, L2 and LLC. In the common cache replacement strategy Least Recently Used (LRU), a structure maintains recency information of accesses to a blocks in a set. The strategy is simple and the victim block is the one that is at the lowest in the recency structure. While the LRU strategy performs well it is known to thrash the memory in some workloads. It is this shortcoming of the LRU policy that most of the research in this area looks to improve upon. Our approach is also looking to find a design that will give a performance improvement over LRU. For our implementation we tried and experimented on a number of different approaches. Section 2 is a brief description of the various background work that has been done in this area of research. Section 3 describes the various methodologies that we have implemented and how that led to the SHIP implementation that performed best for us.

An exclusive cache hierarchy by definition means that if a block is present in an upper-level of the cache it is strictly not present in any lower levels of the cache. The advantage of an exclusive cache hierarchy is that the entire space of all the levels of the cache can be used completely, in contrast to an inclusive cache hierarchy. The disadvantage is that there is high-memory access latency. The exclusivity of the cache leads to complications in the cache replacement problem.

We implemented the set dueling approach that was inspired by Quershi et al [4]. While their implementation was for an inclusive cache hierarchy, we tried to see what performance improvements we'd see in our exclusive cache hierarchy. We also describe a method that we came up with that tracks a blocks movement through the various levels of the hierarchy and if this information could be used to make predictions to make more informed block evictions. We called this method the Global Tracking Table. The results from the first two approaches provided us with no improvement over LRU, in-fact we saw a decrease in performance which led us to try the Re-Reference Interval Prediction algorithm (RRIP) by Jaleel et al [3]. and Signature-based Hit Predictor (SHiP) [6] respectively. The RRIP implementation gave us a minute improvement over LRU. The paper describes three variations on RRIP. The first is called Static RRIP (SRRIP) that was designed to perform well in workloads with a working set greater than the cache or with workloads that have a poor locality of reference. The second is called Dynamic RRIP that like SRRIP is resistant to both memory thrashing workloads and loads with poor locality of reference. The third variation of RRIP is called Bimodal RRIP that like BIP uses to set dueling monitors dynamically between thrash resistant BRRIP and scan-resistant like SRRIP. We found a decent improvement over LRU in our implementation of RRIP, we played around with the Re-reference Prediction Value (RRPV) to see how the results changed. This is described in detail in Section 3. Lastly, we implemented SHiP that ties the re-reference behavior as seen in RRIP with a unique signature. The algorithm learns the re-reference of each line tied to the cache lines.

2 Background

In this section we describe some of the literature that we have referred in our implementations.

2.1 LIP and BIP

In this section, we talk about the work by Qureshi et al [4]. This paper talks about an improvement that can be made over LRU. As it is known that LRU is susceptible to thrashing in memory-intensive workloads. For these workloads, they found that a block of memory traverses the recency list from the Most recently used position to the Least Recently used position without ever being used. Their work showed that by making a simple change to the position in which a new block is inserted, significantly reduce the cache misses in a memory intensive workload. They proposed the LRU Insertion Policy (LIP) which places an incoming block directly at the LRU position rather than at MRU. That way when a block that is accessed only once is immediately kicked out of the cache, rather than thrashing the LRU stack. The LIP was shown to improve the performance of the cache in memory intensive workloads. The paper also proposes an improvement over LIP, called the Bimodal insertion policy, which is similar to LIP but it places a block in the MRU position once in a while, this is done to ensure that the algorithm can adapt to changes in the working set and still have the thrashing protection offered by LIP. The main idea that they highlight in this paper is that of Set Dueling. They called the policy the Dynamic Insertion Policy (DIP). This policy was inspired from the fact that LRU performs well in some work loads and BIP performs well in certain workloads. DIP basically makes these two policies compete with each other, this way the policies will be able to dynamically adapt to differet workloads and the policy that is favored by a given workload will win the duel. Hence the DIP policy performed better than just BIP or LRU alone.

2.2 RRIP

Here we discuss the work done by Jaleel et al [3]. The paper talks of the re-reference interval. LRU is known to perform poorly in workloads where accesses occur outside the temporal locality. These occur in workloads where the working set is larger than the size of the cache. In these workloads, memory blocks outside the cache's purview are accessed in bursts, this leads to LRU performing poorly. Such workloads are said to have a distant re-reference interval. LRU only keeps track of blocks that were recently accesssed, so the work in this paper looks to improve the performance in predicting blocks for replacement having a distant re-reference interval and is called the Re-Reference Interval Prediction (RRIP). In this work, the LRU position stack, i.e. the structure that holds the block in the order of their last use - from most recent to least recent- is though of as an RRIP chain. Like in LRU the block at the top is the most recently used block, in RRIP the block at the begining of the chain is thought of as having a 'near-immediate' re-reference interval, which means that the block is highly likely to be accessed again. While a block at the end of the chain has a 'distant' re-reference interval, and is most likely not to be accessed again. Similar to LRU, the block at the end of the chain is chosen as the victim at the time of block replacement. RRIP does this by requiring an M-bit register per cache block that stores a number called the Re-reference Prediction Value (RRPV). The value of this counter gives the reference interval of the block. For a given set the victim is the set with the lowest RRPV value. RRIP with just 1 bit ($M = 1$) for the RRPV is identical to the Not Recently Used policy. If the bit is 0, it will have a near-immediate reference and if the bit is 1, the predicted re-reference is in the distant future making it the victim for eviction. As many blocks in the cache can have a the same nru-bit value, NRU needed a way to break ties, in the paper the authors say that NRU begins search for the victim from the same location, hence solving this issue. With only 1-bit of information, the NRU can only predict a near-immediate re-reference or a distant re-reference interval. By increasing the number of bits used in the prediction ($M > 1$) the granularity of the prediction was increased and this is the basis for RRIP. RRIP inserts a block into the cache with an intermediate RRPV value, this means that it is midway between near re-reference and distance re-reference interval. The authors used an RRPV value of $2^M - 2$ to represent this intermediate re-reference interval and call it a 'long re-reference interval'. The authors say that by always predicting a new block to have a long re-reference interval, allowed RRIP more time to learn reference patterns, improving the predictions additionally it also prevented blocks with a distant re-reference interval from polluting the cache. For victim selection RRIP selects the victim block as one that is predicted to have a distant re-reference interval. Adding to the idea of RRIP the authors proposed RRIP with hit promotion (RRIP-HP), where a blocks RRPV is updated whenever a hit occurs and updates the value to 0, signifying that a block that has been accessed will be accessed again in the near future. The flip side of this is that a block that is accessed only once after insertion can have a detrimental effect on cache performance. To tackle this issue of RRIP-HP, RRIP frequency Priority is talked about, that only updates the RRPV by one on a cache hit. RRIP is seen to have poor performance in the case that the re-reference interval of all the blocks is greater than the size of the cache. In these cases, RRIP is seen to thrash the memory. To overcome this the authors talk about the Bimodal RRIP, that is analogous to the Bimodal Insertion Policy (BIP) [4]. BRRIP like BIP, inserts blocks with a distant re-reference interval (RRPV is $2^M - 1$) and with a small probability inserts blocks with a long re-reference interval(RRPV of $2^M - 2$) .The authors lastly introduce Dynamic RRIP (DRRIP) that is analogous to Dynamic Insertion Policy, DRRIP uses the idea of set dueling between SRRIP and BRRIP. The algorithm employed by Jaleel et, got a speedup over LRU of 4% to 10% for SRRIP and DRRIP.

2.3 Global Tracking Table

This method was something that we came up with. The idea was to create a table that held the tag and index value of a single memory access and a saturating counter that tracks the number of times it was accessed in the different levels of the cache. The idea was that this table tracks a memory block as it moves across the L1, L2 and LLC. We don't really care about the offset bits, but the tag and index bits togehter were used to track a block across the hierarchies. Then based on the number of times the block was accessed and evicted we updated the saturating counter to predict whether it is a block that was heavily accessed or weakly accessed. A higher value in the counter meant that the block was in high demand in the table, while a low value of the saturating counter meant that the block was not highly valued and had gone down the access priority. In the LLC we had LRU as the default replacement policy. Based on the LRU policy, the LRU block is always selected as the victim block. Every time a block is selected for eviction by LRU, we used the tag and index bits of that block and checked if this block existed in our table and if it did and had a saturating counter value that is high then we override LRU's decision and choose a different block. The block chosen was the next block in the cache position that has a low saturating counter value in the global tracking table or one that is not present in the table. To tune this algorithm we played with the depth of the table. The depth was chosen based on the number of memory accesses in a basic block. Since we were not sure of the optimal depth of the table so we used an experiment to figure this out. The table is itself like a SET containing blocks that have been accessed in the last couple of memory accesses and during each new access the LRU block gets evicted and a newer block gets placed in. We were expecting that backwards loops which will cause the memory accesses to be repeated and hence the global tracking would help avoid unnecessary evictions from the LLC. Though at the time this idea seemed like it would be a good way to avoid unnecessary evictions, our experiments showed very poor results. We belive this was because as the depth of the table went on increasing it was as though we needed to solve a meta-cache replacement problem for our table, hence we had to move away from this idea and try other implementations.

3 Signature-based Hit Predictor

3.1 Basic Framework

SHiP(Signature-based Hit Predictor) [6] is a cache replacement policy which is based on the re-reference idea of SRRIP[reference SRRIP paper]. SRRIP has a static insertion policy where the insertion is with a RRVP value of RRVP_MAX_SIZE-2. BRRIP, which is the Bimodal alternative of SRRIP also does a static insertion between the RRVP values of RRVP_MAX_SIZE-2 and RRVP_MAX_SIZE-1. But SRRIP doesn't dynamically learn from the access patterns in the cache. The unique idea behind SHiP is that each cache insertion is associated with a distinct signature. The signature of each of the cache accesses is then used to dynamically learn about the access patterns in order to decide the best possible insertion. The intuition is that if a cache line insertion by any given signature doesn't receive subsequent hits before getting evicted then a future insertion of a cache block with same signature will most likely have a distant re-reference interval and hence can be inserted with a RRVP value of RRVP_MAX_SIZE-1.

To learn the pattern, SHiP requires two additional fields to be stored with cache lines: the *signature* itself and a single bit to track the *outcome*. The *outcome* bit is initially set to 0 during insertion of a cache block. It is set to 1 if the cache block is re-referenced while in the cache. Another structure which is required is a *Signature History Counter Table (SHCT)* which is a table of saturating counters. Whenever a cache line is re-referenced, that is receives a hit, its signature is used to index into the SHCT and the corresponding saturating counter is incremented. If the cache line is evicted from cache before receiving any hits then its corresponding counter in SHCT is decremented. Hence, a counter value of 0 in SHCT indicates that any cache line which is going to be inserted into the cache with this particular signature is most likely going to have a distant re-reference interval. While a cache line with positive value of counter is likely to receive some hits in the cache. Althought the counters in SHCT don't give any indication of the exact re-reference intervals but they provide a good indication of whether a cache line will receive any hits or not after entering the cache and before getting evicted.

3.2 Signature

There are many ways to choose a signature but we have used three signatures in our experimentation.

1) **Program Counter (PC):** Cache accesses can be grouped based on the Program Counter(PC) values of their instructions. Certain bits of the PC can be hashed together to form a signature which can be then used to index into the SHCT.
2) **Memory Address:** Cache accesses can be grouped based on the Memory address of the cache lines. In this case also a certain number of bits of the address can be selected for hashing and indexing the SHCT.
3) **Program Counter History**: The PC History signature is to track the history of a certain number of PCs leading upto this cache line access and then used this history to index into the SHCT. Since, PCs are ususally 32-bit or 64-bit and History Length can be from 16 to 30 last PCs, XORing them together isn't really the best method to obtain a signature. Rather we used a truncated sum of PCs as suggested in the paper [1] to obtain a signature.

3.3 Modifications for Exclusive Cache

The original SHiP implementation was for exclusive cache hierarchy. For our case we had to make certain modifications in order to retain information of the signatures associated with cache lines as they are shifted around the three levels of cache in our environment. The exclusive cache hierarchy with certain modifications is depcited in Figure 1. Whenever a cache line

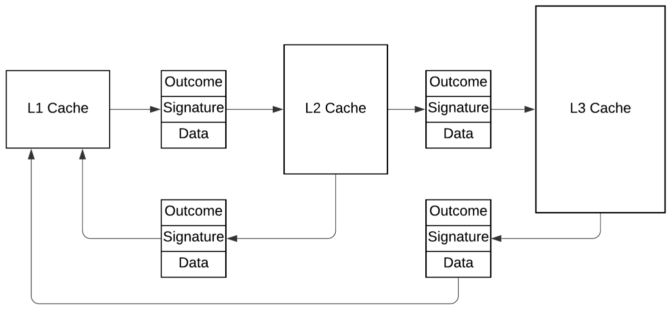

Fig. 1: Exclusive Cache SHiP

is evicted for L1 to L2 or from L2 to L3 then we need to retain the *outcome* and *signature* of the cache line with which it was inserted into L1. In order to do that we created structures between each level of cache, so that whenever a cache line is evicted its corresponding signature and outcome is retained in the next level of cache. Also we see that structures have been created to pass information from L3 to L1 and L2 to L1. This is for the case that if a cache line in L2 or L3 receives a hit then it will be invalidated and placed in L1 and in such case we want to retain its signature and outcome in L1. Hence, these structures help us retaining the outcome and signature of a cache line throughout its time in the cache hierarchy untill its evicted from L3. With this we have a consistent information related to every cache line based on which we update the SHCT.

3.4 SHCT Organization

The Signature History Counter Table is simple to implement. It is a table of saturating counters. There are only two variable parameters which are discussed as follows.

1) **SHCT Size:** As discussed in the SHiP paper, we have implemented a SHCT with 64K entries. We did experiment with even larger entries but the improvement in IPC were insignificant. With a 64K entry SHCT, we will be using 16-bits of our signature for indexing into SHCT. The fact that a larger entry SHCT didn't provide any significant improvement suggests that a 64K entry is sufficient to avoid any destructive aliasing between the signatures.
2) **Saturating Counter Size:** We varied the Saturating counter Max Size between 4 to 32 to check for the best results. A greater Saturating Counter Max value makes the prediction more accurate with a strongly-biased signature while a lower Max value makes it more quicker in terms of learning the access pattern.

4 Methodology and Observation

We have evaluated our implementation with the framework provided in the exclusiu environment. We have a 4-way set associative L1 Cache of size 64KB, a 8-way set associative L2 Cache of size 256KB and a 16-way set associative L3 Cache of size 4MB. We are using a 16K entry SHCT. We have implemented SRRIP at all the three levels of cache. We have divided our observation into sections focussed on the parameters which can be changed to observe changes in IPC. We have divided our observation into three sections each for the impact of Saturating Counter Max Size, RRVP Max Size and Signature type. In each of this section we detail the impact of varying these parameters over IPC.

4.1 Saturating Counter Max Size

In our SHCT we initialize all the saturating counters at a value of *MAX_SIZE/2*. For example, if we have a 3-bit saturating counter we initialize it to a value of 4. This way we don't make it biased towards distant or near re-reference in the initial stages. Figure 2 represensts the Speedup over LRU for 3 different max values. We can observe that there is not much difference in speedup for the three different max values. Hence, we chose a 8-bit saturating counter value for our final configuration.

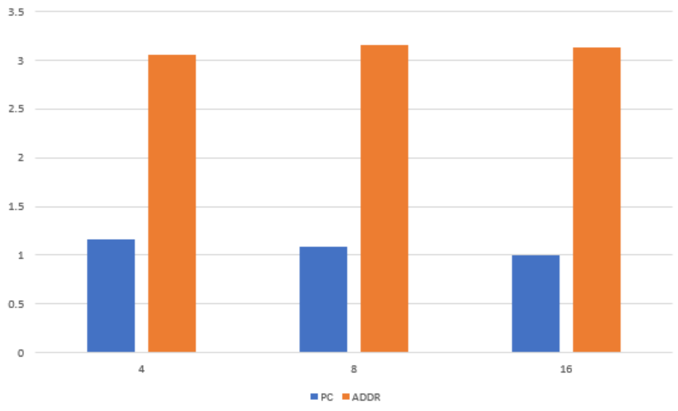

Fig. 2: Exclusive Cache SHiP

4.2 RRVP Max Size

The RRVP value was also experimented with. In the SRRIP and SHiP paper a suggested value of 0 to 3 is suggested. We observed something different in our experiments. We observed that as we increased the RRVP value the performance improved. RRVP values don't make much of a difference in L2 and L3 because we are either inserting cache lines with RRVP value of RRVP Max Size-2 or RRVP Max Size-1 based on the indication by SHCT saturating counters. At L2 and L3, the RRIP policy almost works as LRU. This is because. let's say a cache line is inserted with a RRVP value of 2, then it will be evicted only when all other cache lines with RRVP values of 2 and 3 before it get evicted. But at L1 they help avoid the scanning problem that has been extensively discussed in SRRIP section. In Figure 3 we can see that a RRVP value of 32 provides the best results for us. Hence we selected this as our final configuration.

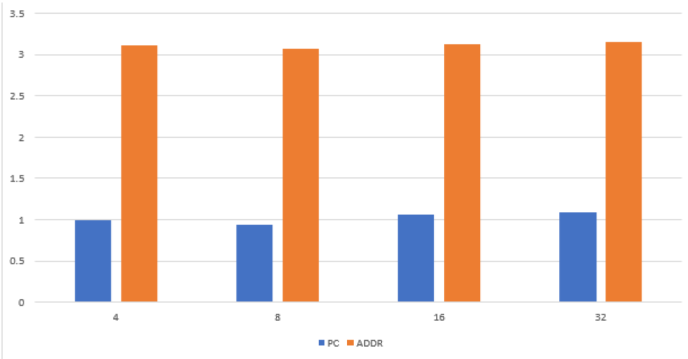

Fig. 3: Exclusive Cache SHiP

4.3 Signature Configuration

As discussed earliier we focussed on two signatures. PC of the instruction and Address of the cache line. These are the two signatures which we primarily focussed on. One thing which is perplexing is that the Address based signature significantly outperforms the PC based signature as can be seen in Figure 2 and Figure 3. We ran multiple iterations and wrote our code in different ways but in all the cases we got similar results. This is perplexing because in majority of research papers PC has been the primary focus as learning signature. We couldn't confirm this with other competitors and hence we can't conclude if this behavior is specific to us because of our code implementation or other competing teams are observing a similar behavior. Figure 4 shows the percentage speedup over LRU for PC and ADDR based signature. Also one other perplexing observation was the type of hashing that performed well for ADDR signature. We used the lower 16-bits of tag and index as signature, i.e. tag_index_bits[15:0]. And then we used an XOR of the upper tag+index bits i.e. tag_index_bits[47:32] xor tag_index_bits[31:16]. The second case provided much better results as compared the the first signature. Intuitively the first hashing, a simple one, seems a good indication but that was not the case in our experiments. We are not really sure as to what could be the reason. One of the probable answers can be that higher bits represent the present of more larger data structures and larger jumps in the control flow graph of the program.

We also experimented with PC History as signature, but that also didn't provide any improvements over the Address based signature. We noticed that the L1 Cache is a Unified Cache, hence the accesses contain Instruction Access as well as Data Access. Hence, we built separate PC History Table for instruction and data accesses each, but that segregation also didn't provide promising results. The results were similar to a simple PC based signature and hence the idea was dropped. One possible reason could be an incorrect way of encoding the PC History. We used truncated sum of the past PCs but it's difficult to ascertain if the implementation is exactly the same as discussed in [1] because they haven't detailed how it was implemented.

5 Results

We finally selected our combination of RRVP size=32, saturating counter size=8, SHCT size=16K entries and signature=ADDR as described in the section X. Table X represents the LRU and SHiP policy IPC. We can see that the final speedup obtained is around 3.2% over LRU.

Figure 4 and Figure 5 represent the percentage Speedup over LRU for each of the benchmark with our policy. We can see

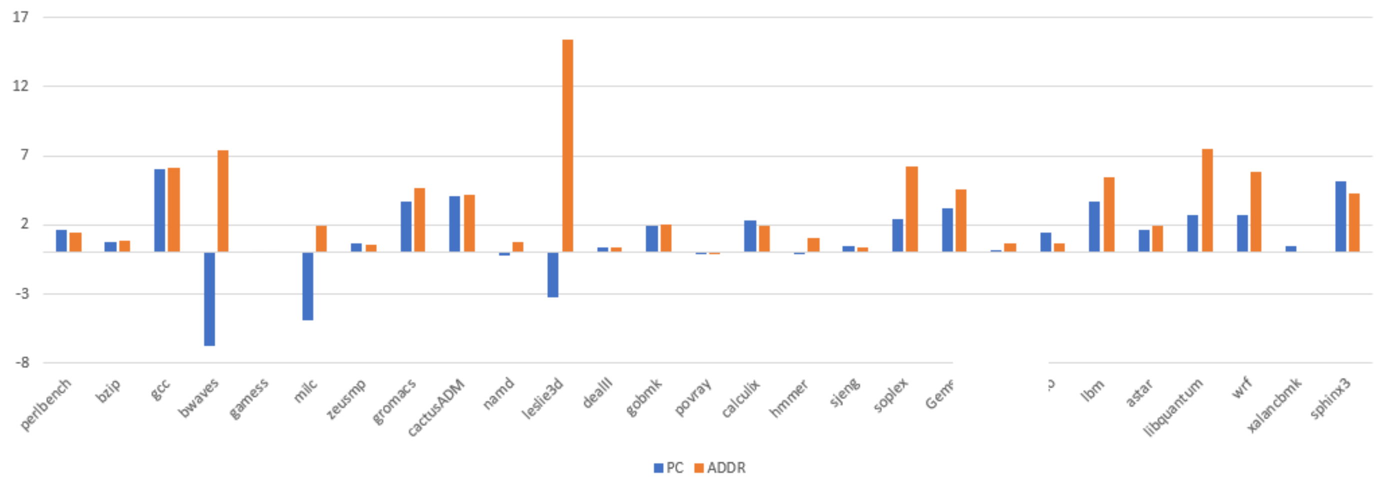

Fig. 4: PC and ADDR Signature Performance

	LRU	SHiP	Speedup
400.perlbench-50B	2.015	2.044	1.014
401.bzip2-226B	1.625	1.638	1.008
403.gcc-16B	0.520	0.552	1.062
410.bwaves-1963B	2.515	2.700	1.074
416.gamess-875B	3.639	3.642	1.001
433.milc-337B	0.550	0.561	1.019
434.zeusmp-10B	2.578	2.594	1.006
435.gromacs-111B	2.924	3.061	1.047
436.cactusADM-1804B	1.640	1.708	1.041
444.namd-426B	3.224	3.249	1.008
437.leslie3d-149B	2.047	2.363	1.154
447.dealII-3B	3.702	3.716	1.004
445.gobmk-30B	1.812	1.850	1.021
453.povray-576B	2.760	2.758	0.999
454.calculix-460B	1.765	1.800	1.020
456.hmmer-88B	2.137	2.161	1.011
458.sjeng-767B	3.029	3.040	1.004
450.soplex-247B	0.607	0.645	1.062
459.GemsFDTD-1491B	1.020	1.067	1.046
464.h264ref-97B	2.460	2.476	1.006
465.tonto-1769B	1.872	1.885	1.007
470.lbm-1274B	1.585	1.671	1.054
473.astar-42B	0.635	0.648	1.020
462.libquantum-714B	0.261	0.281	1.075
481.wrf-196B	1.968	2.083	1.058
483.xalancbmk-127B	0.433	0.434	1.001
482.sphinx3-417B	1.074	1.120	1.043
Geometric Mean	**1.533**	**1.581**	**1.032**

TABLE 1: Speedup Statistics

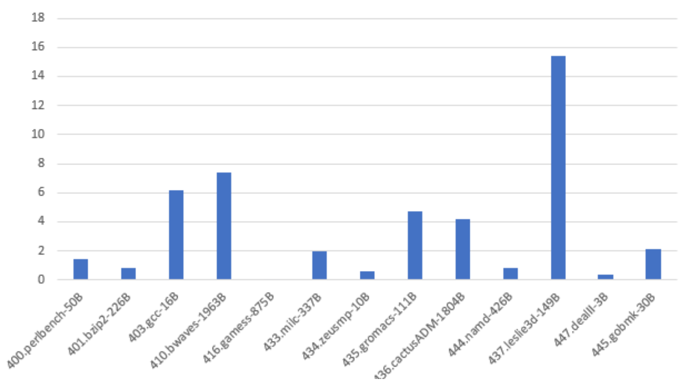

Fig. 5: Speedup over LRU

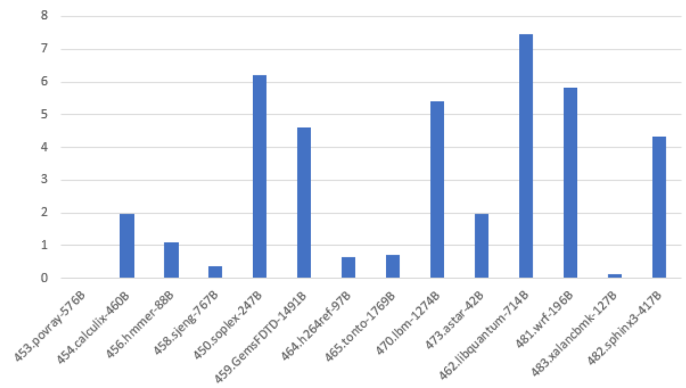

Fig. 6: Speedup over LRU

that for most of the benchmarks we obtained some amount of speedup over LRU. leslie3d performed the best where we got around 15% of improvement. Certain benchmarks such as gamess and povray got no improvement over LRU.

6 Failed Experiments

In this section we would like to discuss a few experiments that we tried but didn't give us any improvements over the final implementation.

1) In Young et al [7] the idea was suggested that cache lines with a signature haveing a high saturating counter value to be inserted with RRVP value of 0. We checked for a saturating counter value of 7 and inserted our cache lines with a RRVP value of Max_Size-3 instead of 0, since our Max_Size for RRVP is 32. This didn't provide any improvements. We observed minute change in IPCs.
2) As discussed earlier, tracking the PC history didn't provide any significant improvements fo us. There is a high probability that we are making a mistake in our encoding of the history of PCs because in many advanced cache replacement policies such as [2] and [5] PC History is used.
3) We implemented Set Duelling as well. Set Dueling as introduced in [3] is supposed to select the best of two policies. DRRIP uses set dueling between SRRIP and

BRRIP. But we don't need set dueling for SHiP because it is a dynamic learning policy. Unlike DRRIP where the selection is between two static policies, SHiP learns the access patterns of the signatures and then decides insertion. But our idea was to set duel between PC and Addr Signature. We created two SHCT one for PC and the other for ADDR signatures. And then used set duelling for selecting between the signatures. The motivation for this was the observation that for some of the benchmarks a PC based signature did provide better IPC than an ADDR based signature. But unforunately this idea also didn't give us better results. It did provide better results than a PC based signature but performed worse than a purely ADDR based signature.

7 Conclusion

In this work we looked at various techniques for cache replacement for an exclusive cache hierarchy. We looked at many different approaches designed for an inclusive cache hierarchy and tried tweaking the algorithms to work for the exclusive cache hierarchy. Our paper discusses how the different algorithms performed on an exclusive cache hierarchy and the speedup that was achieved over LRU. We managed to achieve a speedup of 3.2% over LRU using the SHiP algorithm, which was the best performing approach for us.

References

[1] An-Chow Lai, C. Fide, and B. Falsafi, "Dead-block prediction amp; dead-block correlating prefetchers," in *Proceedings 28th Annual International Symposium on Computer Architecture*, June 2001, pp. 144–154.

[2] A. Jain and C. Lin, "Hawkeye : Leveraging belady s algorithm for improved cache replacement," 2017.

[3] A. Jaleel, K. B. Theobald, S. C. Steely, Jr., and J. Emer, "High performance cache replacement using re-reference interval prediction (rrip)," *SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 60–71, Jun. 2010. [Online]. Available: http://doi.acm.org/10.1145/1816038.1815971

[4] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer, "Adaptive insertion policies for high performance caching," in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ser. ISCA '07. New York, NY, USA: ACM, 2007, pp. 381–391. [Online]. Available: http://doi.acm.org/10.1145/1250662.1250709

[5] E. Teran, Z. Wang, and D. A. Jiménez, "Perceptron learning for reuse prediction," in *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-49. Piscataway, NJ, USA: IEEE Press, 2016, pp. 2:1–2:12. [Online]. Available: http://dl.acm.org/citation.cfm?id=3195638.3195641

[6] C. Wu, A. Jaleel, W. Hasenplaugh, M. Martonosi, S. C. Steely, and J. Emer, "Ship: Signature-based hit predictor for high performance caching," in *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2011, pp. 430–441.

[7] V. Young, C.-C. Chou, A. Jaleel, and M. K. Qureshi, "Ship + + : Enhancing signature-based hit predictor for improved cache performance," 2017.