Implementation of Re-Reference Interval Prediction (RRIP) Cache Replacement

Sridhar Mareguddi
Department of Electrical and Computer Engineering
Texas A&M University
sridharm@neo.tamu.edu

ABSTRACT

Effective cache replacement is becoming an increasingly important issue in cache hierarchy design as large set-associative caches are widely used in high-performance systems. The numerous cache replacement policies available today attempt to minimize the number of block replacements by predicting the re-reference interval. As little or no information is available about the subsequent reference blocks, all these replacement policies fail to emulate the optimal replacement policy. The commonly used LRU replacement policy always predicts a near re-reference interval on a cache hits and misses. Such replacement policy works well when the working set is smaller than the available cache size but fails when there is a distant re-reference interval for a working set larger than the available cache size. To improve the performance of such working loads, we implement a strategy called "Re-Reference Interval Prediction (RRIP)" proposed in [1] which is both scan-resistant (Static RRIP) and thrash-resistant (Dynamic RRIP). We evaluate the performance of our implemented model using four of the integer benchmarks and four floating point benchmarks in SimpleScalar and the results clearly indicate that RRIP outperforms the commonly used LRU policy.

Categories and Subject Descriptors

B.3.2 [Design Styles]: Cache memories, C.1.4 [Parallel architectures]

General Terms

Design, Performance.

Keywords

Replacement, Scan Resistance, Thrashing, Adaptive.

1. INTRODUCTION

As per the optimal replacement policy, the optimal candidate for replacement is the one that is accessed farthest in the future. Unfortunately, as OPT requires the knowledge of future accesses, it is not practically implementable. Practical replacement policies like LRU, LFU uses the recency and the frequency respectively to predict future accesses. LRU chain can be thought as a Re-Reference Interval Prediction (RRIP) chain with the block at the head of the chain having a near immediate re-reference and the block at the tail of the chain as having a distant re-reference. Typically on a cache miss, the block at the tail of the RRIP chain is re-replaced since it is predicted to have a distant re-reference and the new incoming block is always placed at the head of the RRIP chain anticipating its near re-reference. Although LRU performs well with workloads having high locality, its performance degrades for applications whose re-references occur in a distant future. Such applications usually have a working set which is larger than the cache size or have bursts of reference to non-temporal data which discards the active working set from cache. The Dynamic Insertion Policy (DIP) proposed in [2] overcomes this limitation by dynamically choosing between LRU-Insertion Policy (LIP) which preserves part of the working set in the cache by replacing most recently filled cache blocks and the LRU for other workloads. Unfortunately DIP makes the same prediction as either LRU or LIP for all the references. The LRU component of DIP always predicts a near re-reference while LIP component always predicts a distant re-reference and hence fails to make a correct prediction when there is mixed access pattern containing both near reference and distant rereference. When there is a burst of references to data with distant rereference interval (scans) both DIP and LRU limits the cache performance.

In this paper, we have implemented a scan-resistant policy that uses Re-Reference Interval Prediction (RRIP) requiring minimal hardware overhead. The RRIP uses M-bit per cache block to store its Re-Reference Interval Prediction (RRPV). We have implemented both scan-resistant Static RRIP and thrash-resistant Dynamic RRIP which improves the cache performance over LRU.

2. PREVIOUS WORK

Many cache replacement policies that exist today focuses on improving the last level cache by targeting the dead blocks. Dead blocks is commonly addressed by using access frequency to predict the re-reference pattern. The Least Frequently Used (LFU) [7] replacement policy predicts the blocks access frequency and removes the block which are infrequently accessed. LFU significantly degrades the performance for workloads where recency is the preferred rather than the frequency. Several other policies combine both recency and frequency to address this issue but they have to be tuned on per-application basis which requires hardware overhead.

Belady in [2] proposed the optimal replacement policy (OPT) which provides the highest possible hit rate that can be ever achieved. OPT decides the optimal candidate for replacement by selecting the block that is accessed farthest in the future. This policy cannot be practically implemented as it requires the knowledge of future access blocks.

The anti-thrashing policies like LRU Insertion Policy (LIP), Bimodal Insertion Policy (BIP) and Dynamic Insertion Policy (DIP) performs well for memory intensive workloads. The replacement process is divided into two parts namely victim selection and insertion. The victim selection part chooses the block that should be evicted from the cache, whereas the insertion part selects where in the recency stack the new block should be placed. The recency stack is used to keep track of the recency of each line.

LRU Insertion Policy (LIP) tries to retain some of the working load in the cache by inserting the new incoming block at the tail of RRIP chain instead of inserting it at the head of the RRIP chain as in LRU. However LIP cannot perform well under changing workloads as new cache blocks are inserted in the LRU position retaining the stale blocks for a longer time without eviction.

Bimodal Insertion Policy works in similar way to LIP but it infrequently inserts some of the blocks at MRU position. This can be achieved by using a counter that decrements whenever there is a cache miss and inserts blocks into the MRU position when the counter value reaches zero. However BIP also degrades the cache performance when the workloads are recency friendly.

To overcome the disadvantages of LIP and BIP, an adaptive policy called DIP is proposed that dynamically chooses either LRU or BIP depending on whichever scheme is providing the maximum hits. This can be achieved by using a Policy Selector (PSEL) that increments the counter whenever LRU policy provides a cache miss and decrements the counter whenever BIP policy provides a cache miss. The most significant bit of the PSEL is used to select the winning policy that will be applied for the rest of the blocks. However this requires a mechanism to know whether each policy will yield a hit or a miss for each cache access.

Another area of research predicts when the re-reference interval of a cache block becomes distant, i.e., a cache block becomes dead [11]. A recent study applied dead block prediction at the LLC [16]. The proposed policy attaches a prediction with each cache block to determine whether or not the block is dead. The proposed policy uses the block's re-reference history to predict death after the block moves out of the head of the RRIP chain. The victim selection policy selects dead blocks closer to the tail of the RRIP chain. While dead block prediction improves cache performance, it requires additional hardware overhead for the dead block predictor. A recent study [9] shows that dead blocks occur when the application working set is larger than the available cache. In such scenarios, the proposed Dynamic Insertion Policy (DIP) [9] dynamically changes the insertion policy from always inserting blocks at the head of the RRIP chain to inserting the majority of the blocks at the tail of the RRIP chain. By doing so, DIP preserves some of the working set in the cache. Since DIP makes a single insertion policy decision for all references of a workload, DIP only targets workloads whose working set is larger than the available cache. Consequently, in the presence of scans, the LRU component policy of DIP is unable to preserve the active working set in the cache. Another recent study proposes pseudo-LIFO [17] based replacement policies. The policy proposes cache replacement using a fill stack as opposed to the recency stack. The proposed policy learns the re-reference probabilities of a cache block beyond each fill stack position and finds that evicting blocks from the upper portion of the fill stack improves cache utilization by evicting dead blocks quickly.

It would be highly desirable that a single cache replacement policy provide scan resistance and perform well for recency friendly workloads. The proposed policy however requires additional hardware to keep track of a block's fill stack position and also requires a dynamic mechanism to learn the best eviction position on the fill stack. Various other solutions [18, 15, 19, 20] exist but they either require significant additional hardware or they drastically change the organization of the existing cache. Reuse distance prediction [15] most closely resembles the work presented in this paper. Reuse distance prediction explicitly calculates the reuse distance of a given cache block by using a PC indexed predictor. RRIP does not explicitly calculate reuse distance. Instead, RRIP always predicts that all missing cache block will have the same re-reference interval and updates the prediction when more information is available, for example, on a re-reference. RRIP also differs from prior work in that it proposes a high performing practical scan-resistant cache replacement policy that does not require significant hardware overhead or changes to the existing cache structure.

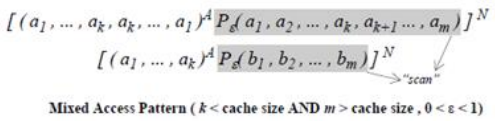
3. IMPLEMENTATION

3.1 SRRIP

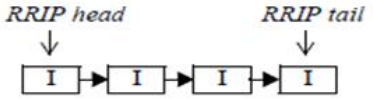
The newly proposed solution in this paper for cache replacement is based on Re-reference Interval Prediction (RRIP). RRIP uses M-bit per cache block to store one of the 2^M possible Re-reference Prediction Values (RRPV). Based on the cache access pattern, RRIP dynamically learns about the re-reference information for each block. A RRPV value of 0 indicates that the cache block is predicted to be re-reference in near future and a RRPV value of 2^M – 1 indicates that the cache block is predicted to be re-reference in distance future. A RRPV value closer to 0 means that the block will be re-referenced sooner than the blocks with a RRPV value more distant from 0. Based on these above ideas the RRIP algorithm is designed to predict the block to be replaced. Instead of always replacing a block with RRPV of 2^M – 1 (distant re-reference), we insert the new incoming block with RRPV value if 2^M-2. We call this interval as long re-reference. A long re-reference interval is defined as an intermediate re-reference interval that is skewed towards a distant re-reference interval. By having long-reference, RRIP gets more time to learn and improve the prediction about the re-reference interval of a block. On a cache miss, the RRIP chooses to replace the block with RRPV value of 2^M -1. The selection of block starts from left to right order until it finds a block with RRPV value 2^M-1. If no such blocks are found then RRIP updates by incrementing RRPV values for all the blocks. The block's RRPV value is changed when a cache hit occurs for that block. The algorithm used for this update of the RRPV register for each block in case of cache hit is called hit promotion policy. There are two variants of this hit promotion policy .The first one is called as Hit Priority(HP) which updates the RRPV value of a block receiving hit to 0 predicting that block to have near immediate re-reference. This policy degrades the performance of cache when a cache block is re-referenced only once after a cache insertion. The second policy called as Frequency Priority (FP) overcomes the problem of the Hit priority policy by decrementing the RRPV value of the block receiving a cache hit. This way it prioritizes

infrequently re-referenced cache blocks over frequently re-referenced cache blocks. Since this is determined statically on cache hits and misses, we call this policy as Static Re-Reference Interval Prediction (SRRIP) and the two variants of hit promotion as SRRIP-HP and SRRIP-FP.

The detailed algorithm to implement the above feature is give below.



Mixed Access Pattern ($k <$ cache size AND $m >$ cache size , $0 < \varepsilon < 1$)



Static RRIP Algorithm:

- Cache Hit:
  1. Set RRPV of block to '0'
- Cache Miss:
  1. Search for first '3' from left
  2. If '3' found go to step (5)
  3. Increment all RRPVs
  4. Go to step (1)
  5. Replace block and set RRPV to '2'

3.2 DRRIP

When the re-reference interval of all blocks is larger than the cache size, Static RRIP (SRRIP) causes cache thrashing. In order to avoid thrashing, Bimodal RRIP (BRRIP) is implemented which inserts majority of the cache blocks with RRPV value 2^M -1(distant re-reference) and few cache blocks with RRPV value 2^M-2 (long re-reference). This access pattern degrades the performance significantly when the access pattern doesn't generate the thrashing. In order to achieve consistent performance across all the cache access patterns, we used Dynamic Re-Reference Interval Prediction (DRRIP) to dynamically determine whether the application is best suited to scan-resistance SRRIP or thrash-resistance BRRIP. DRRIP uses Set Dueling approach to identify which replacement policy is best suitable for the application. Set Dueling uses Set Dueling Monitors (SDM) dedicates few blocks to SRRIP and few blocks to BRRIP and uses Single Policy Selection (PSEL) counter to determine winning policy. DRRIP choses the winning policy for the remaining sets of the cache.

The detailed algorithm to implement DRRIP is given below:

Dynamic RIP

- Bimodal RRIP (BRRIP)
  - Similar to Bimodal Insertion Policy of DIP
    - Insert majority of cache blocks with distant re-ref
    - Insert infrequently with a long re-ref interval
- Set Dueling
  - Choose between scan-resistant SRRIP and thrash-resistant BRRIP by using two Set Dueling Monitors
  - Use a single policy selection counter

.

4. PERFORMANCE ANALYSIS

4.1 Simulator

SimpleScalar is used for our performance studies. We evaluated only single core configurations. L1 instruction and data cache are 32KB 4-way associative with 64 bytes block size while L2 is unified 8-way 256 KB cache. LRU replacement policy is used for L1 data and instruction caches while we evaluate the implemented policy in LLC L2 cache. The 10-bit PSEL counter value for DRRIP.

All caches configurations specified with following format:
<name>:<nsets>:<bsize>:<assoc>:<rrpv_m_value>:
<psel_reg_width> : <repl>
where

<name>	- name of the cache being defined
<nsets>	- number of sets in the cache
<bsize>	- block size of the cache
<assoc>	- associativity of the cache
<rrpv_m_value>	- width of the RRPV
<psel_reg_width>	- width of the PSEL
<repl>	- block replacement strategy

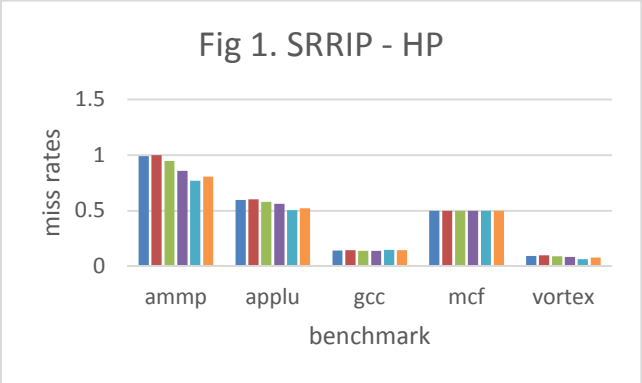
We have extended the replacement policies supported by SimpleScalar to include RRIP. Below table lists all the options including the existing ones.

Replacement Policy	Mnemonic	New
LRU	<l>	
FIFO	<f>	
Random	<r>	
SRRIP_HP	<h>	✓
SRRIP_FP	<s>	✓
BRRIP	<b>	✓
DRRIP	<d>	✓

4.2 Benchmarks

For our analysis we have used four of the integer benchmarks from SPEC2000 namely *bzip2*, *gcc*, *mcf*, *vortex* and four of the floating point benchmarks namely *ammp*, apsi, *equake* and lucas. These benchmarks are selected because of their sensitivity to memory latency and there is an opportunity to improve their performance through enhanced replacement decisions. The SPEC2000 workloads were all collected using Pinpoints for the reference input set while the real world workloads were all collected on a hardware tracing platform. The real world workloads include both operating system and user-level activity

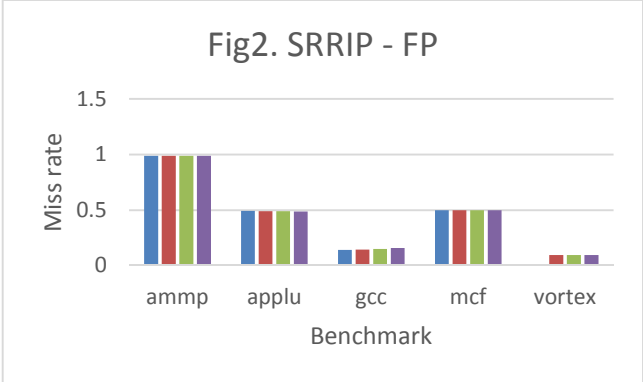
while the SPEC2000 workloads only include user-level activity. The workloads were all run for 200M instructions. Simulations were run until *all* benchmarks ran 200 million instructions. Statistics for each core were collected only for the first 200 million instructions. If the end of the trace is reached, the model rewinds the trace and restarts from the beginning. The simulation methodology is similar to recent work on shared caches [12, 13, 14, 15].



5. RESULTS AND ANALYSIS

5.1 SRRIP Sensitivity to RRPV on Insertion

Figure 1 compares SRRIP-FP performance for M=1, 2, 3, 4 and 5 against the LRU policy whereas figure 2 compares the performance of SRRIP-FP for the same RRPV values against LRU replacement policy. A RRPV value of 1 represents the NRU replacement policy. Both SRRIP policies always predict a long re-reference interval on cache insertion. The x-axis shows different workloads while the y-axis shows cache misses. The x-axis labels *ammp*, *applu*, *gcc*, *mcf* and *vortex* represent the spec2000 workloads. Figure shows that SRRIP-HP reduces MPKI by a maximum of 28% for *ammp* workloads whereas *applu* and *vortex* showed a maximum performance improvement of 18.6% and 45.5 % respectively. The reductions in MPKI allow SRRIP-HP to outperform LRU by an average of 6% across all workloads. The reductions in MPKI allow SRRIP-HP to outperform LRU by 5% across all workloads. *mcf* benchmark which favors the recency over distant re-reference gains no benefit by varying M. On average, SRRIP is insensitive when the width of the RRPV register is increased beyond 5. Some workloads like *gcc* experience performance degradation when the width of the RRPV register increases. This is because wider RRPV registers retain stale blocks in the cache for long periods of time after their last hit and reduce the effective cache capacity. Generally, a 2-bit or 3-bit RRPV is sufficient to be scan-resistant. As the figure indicates, both SRRIP-HP and SRRIP-FP outperform LRU. NRU (M=1) almost always performs worse than LRU. Additionally, SRRIP-HP performs better than SRRIP-FP. This implies that the first order benefit of a scan-resistant replacement algorithm is not from precisely detecting frequently referenced data in the cache but from preserving data that receives cache hits, i.e., the active working set.



5.2 SRRIP Sensitivity to Cache Configuration

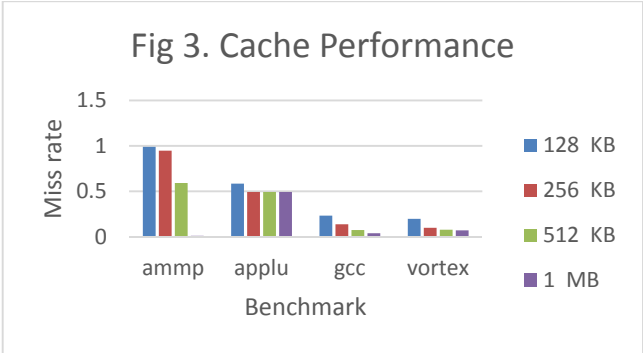


Figure 2 presents SRRIP performance for the different workload categories on different LLC sizes: 256KB, 512KB and 1MB. All LLCs are 16-way set associative with RRPV value set to 2. The y-axis shows the miss rates when the cache size is varied. The figure shows that increasing the cache size will result in fewer misses and hence better performance. Increasing the associativity from 2-way to 32-way provided results comparable to the above figure. These results show that SRRIP is scalable to different cache configurations.

5.3 DRRIP Performance

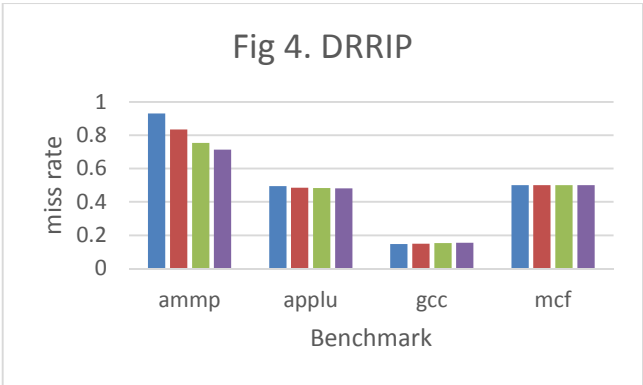


Figure 4 presents the performance of DRRIP with variation in M (2, 3, 5). The figure shows that DRRIP significantly improves cache performance for *ammp* workload whereas for *applu* and

gcc, the improvement is negligible. *mcf* showed no improvements as RRPV value is increased. On average, DRRIP improves performance by 5% above SRRIP. Since both 2-bit and 3-bit DRRIP perform similarly, we can say that 2-bit DRRIP is sufficient for scan-resistance and thrash-resistance. The performance analysis in [1] used *sphinx3*, *hmmer*, and *mcf* benchmarks for DRRIP. These workloads have a *knee* in the working set that is slightly larger than a 2MB. Across most workloads, DRRIP has similar or better performance than SRRIP. Since DRRIP optimizes for the cache miss metric and not the throughput metric, DRRIP can degrade performance when the cost of a miss varies in an application.

6. SUMMARY

The commonly used LRU replacement policy performs badly when the workloads have a distant re-references (scans). It evicts the block at the tail of the chain before it is likely to be reused. Retaining such blocks for a longer period of time provide more hits and hence improves cache performance significantly. The Dynamic Insertion Policy overcomes this limitation by dynamically choosing between LRU-Insertion Policy which preserves part of the working set in the cache by replacing most recently filled cache blocks and the LRU replacement policy for other workloads. Unfortunately DIP makes the same prediction as either LRU or LIP for all the references. DIP degrades the cache performance for mixed access patters.

1. We implemented Re-reference Interval Prediction (RRIP) that statically predicts the re-reference interval of all missing cache blocks to be an intermediate rereference interval that is between a near-immediate rereference interval and a distant re-reference interval. RRIP updates the re-reference prediction to be shorter than the previous prediction upon a re-reference. We call this policy as Static RRIP (SRRIP). We show that SRRIP is scan-resistant and only requires 2-bits per cache block.
2. We implemented two SRRIP policies: *SRRIP-Hit Priority (SRRIPHP)* and *SRRIP-Frequency Priority (SRRIP-FP)*. SRRIP-HP predicts that any cache block that receives a hit will have a *near-immediate* re-reference and thus should be retained in the cache for an extended period of time. SRRIP-FP on the other hand predicts that frequently referenced cache blocks will have a *near-immediate* re-reference and thus they should be retained in the cache for an extended period of time.
3. We implemented *Dynamic RRIP (DRRIP)* as an enhancement to SRRIP-HP. DRRIP provides both scan-resistance and thrash resistance by using set dueling to dynamically select between inserting all missing cache blocks with an *intermediate* rereference interval or with a *distant* re-reference interval.

7. REFERENCES

[1] A. Jaleel, K. B. Theobald, S. C. Steely Jr., and J. Emer. High performance cache replacement using re-reference interval prediction (RRIP). In Proc. of the 38th International Symposium on Computer Architecture, 2010

[2] L. A. Belady. A study of replacement algorithms for a virtual-storage computer. In IBM Systems journal, pages 78–101, 1966.

[3] H. Al-Zoubi, A. Milenkovic, M. Milenkovic. “Performance evaluation of cache replacement policies for the SPEC CPU2000 benchmark suite.” In ACMSE, 2004.

[4] A. Jaleel, W. Hasenplaugh, M. K. Qureshi, S. C. Steely Jr., J. Emer. “Adaptive Insertion Policies for Managing Shared Caches”. In PACT, 2008.

[5] G. Keramidas, P. Petoumenos, S. Kaxiras. “Cache replacement based on reuse-distance prediction”. In ICCD, 2007

[6] R. Subramanian, Y. Smaragdakis, G. Loh. “Adaptive caches: Effective shaping of cache behavior to workloads.” In MICRO-39, 2006.

[7] D. Lee, J. Choi, J. Kim, S. H. Noh, S. Lyul Min, Y. Cho, C. Sang Kim. “LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies,” IEEE Trans.Computers, vol. 50, no. 12, pp. 1352–1360, 2001.

[8] Qureshi M., Jaleel A., Patt Y., Jr. S. & Emer J. Set-Dueling-Controlled Adaptive Insertion for High-Performance Caching. IEEE Micro, pp. 91-98, 2008

[9] Qureshi M., Jaleel A., Patt Y., Jr. S. & Emer J. Adaptive Insertion Policies for High Performance Caching. Proceedings of the 34th annual international symposium on Computer architecture (ISCA’07), pp. 381-391, 2007

[10] A. Lai, C. Fide, B. Falsafi. Dead-block prediction & dead-block correlating prefetchers. In ISCA-28, 2001

[11] W. Lin and S. K. Reinhardt. “Predicting last-touch references under optimal replacement.” Technical Report CSE-TR-447-02, U. of Michigan, 2002.

[12] A. Jaleel, W. Hasenplaugh, M. K. Qureshi, S. C. Steely Jr., J. Emer. “Adaptive Insertion Policies for Managing Shared Caches”. In PACT, 2008.

[13] M. Chaudhuri. “Pseudo-LIFO: The Foundation of a New Family of Replacement Policies for Last-level Caches”. In Micro, 2009.

[14] Y. Xie and G. Loh. “PIPP: Promotion/Insertion Pseudo-Partitioning of Multi-Core Shared Caches.” In ISCA-36, 2009

[15] G. Loh. “Extending the Effectiveness of 3D-Stacked DRAM Caches with an Adaptive Multi-Queue Policy”. In Micro, 2009.

[16] D. Lee, J. Choi, J. Kim, S. H. Noh, S. Lyul Min, Y. Cho, C. Sang Kim. “LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies,” IEEE Trans. Computers, vol. 50, no. 12, pp. 1352–1360, 2001.

[17] M. Chaudhuri. “Pseudo-LIFO: The Foundation of a New Family of Replacement Policies for Last-level Caches”. In Micro, 2009.

[18] T. Johnson and D. Shasha, “2Q: A low overhead high performance buffer management replacement algorithm,” In VLDB Conf., 1994.

[19] K. Rajan and G. Ramaswamy. “Emulating Optimal Replacement with a Shepherd Cache”. In Micro-40, 2007.

[20] Y. Zhou and J. F. Philbin, “The multi-queue replacement algorithm for second level buffer caches,” in USENIX Annual Tech. Conf, 2001.