2022 IEEE 7th International conference for Convergence in Technology (I2CT)
Pune, India. Apr 07-09, 2022

Improved Cache Replacement Policy based on Recency Time Re-Reference Interval Prediction

Chetan R Athni
Department of Electronics and Communication Engineering
PES University
Bangalore, India
chetanathni@pesu.pes.edu

Vaishnavi Vinod Chippalkatti
Department of Electronics and Communication Engineering
PES University
Bangalore, India
vaishnavivc@pesu.pes.edu

Ankita Nandakumar
Department of Electronics and Communication Engineering
PES University
Bangalore, India
ankitanandakumar@pesu.pes.edu

Nandana A V
Department of Electronics and Communication Engineering
PES University
Bangalore, India
nandanaav@pesu.pes.edu

Pavitra Y J
Department of Electronics and Communication Engineering
PES University
Bangalore, India
pavitra.yj@pes.edu

***Abstract*— The cache memory, having one of the fastest response times in the memory hierarchy (superseded only by processor registers), while also residing closest to the processor, has a major role in improving memory access times. While many different heuristics are involved in improving cache performance, the replacement policy plays a pivotal role. The work presents a novel replacement policy 'Recency Time Based Re-Reference Interval Prediction' (RT-RRIP), based on the concept of a recency time prefilter augmenting the performance of the re-reference interval prediction (RRIP) model, resulting in decrease in the number of cache misses and miss latency. An extensive analysis has been carried out using standardized SPEC CPU2006 benchmarks to compare the new policy with existing alternatives. Results reveal that the proposed cache replacement policy RT-RRIP has up to 12% performance improvement over RRIP and 4% performance improvement over LRU on average.**

***Keywords*— cache memory, access time, replacement policy, cache miss, miss latency, recency time prefilter, re-reference interval prediction**

I. Introduction

Computer systems run multiple application programs simultaneously and this requires fast access to data. As the performance of the system depends on the access time required to retrieve data, cache memories play a major role in the memory hierarchy. The cache block is present between the processor and main memory of the system to store data that is used frequently by the main memory. It is observed that most of the time spent in program execution is to execute instructions that repeat themselves such as simple or nested loops. It can be inferred that most of these instructions are present in localized areas of the program and are executed repeatedly whereas the rest of the instructions are accessed infrequently. The working of a cache memory block is based on the property of a computer system referred to as 'locality of reference'. Locality of reference consist of two principles namely spatial locality, and temporal locality. The possibility that when the processor requests for a data block or instruction, it is most likely to request for another data block or instruction present in the immediate next location is referred to as spatial locality, while temporal locality is when the same data block or instruction is requested multiple times as in the case of loop instructions.

Cache replacement algorithms decide if the incoming data block should replace the existing block present in cache memory. The algorithms are most efficient when the data blocks evicted are not requested for the longest time. These algorithms are analysed by two important performance parameters: cache hit ratio and cache miss ratio. Cache hits occur when a requested data block is present in the cache memory. If not, it is then referred to as a cache miss.

Some of the commonly used replacement policies are Least Recently Used (LRU), Least Frequently Used (LFU), Most Recently Used (MRU), Random Replacement (RR) and First In First Out (FIFO) [1]. LRU is a commonly used policy that discards data blocks which are least recently used in the cache memory. MRU is used when the probability of older blocks being accessed is high and therefore the data block that is accessed most recently is eliminated. In LFU policy, the less frequently used block is removed from the cache memory. Random blocks are replaced in the event of a cache miss, and space needs to be made available for the incoming block in the RR policy. In FIFO policy, the first data block to enter the cache will be evicted when there is no space available in cache memory, regardless of its accesses.

II. Previous Work

Various techniques, both software-based and hardware-based, have proposed an extensive range of heuristics used as yardsticks to optimize the performance of the cache replacement policy and obtain an optimal result closest to the optimal replacement policy Belady's MIN [2].

The usage of the deterministic naps scheme [3] reduces both static and dynamic power consumption. It uses a mechanism based on hashing that leads to a minimal number of cache lines being read during the execution of operations. Performance improvement is observed when no cache lines in a set are matched to the currently referenced address, thus skipping a few sets from being brought in, as they are guaranteed to be misses.

The filter data cache [4] is a low-sized cache designed to decrease the power consumption of the L1 I-cache. It is typically 1-2 KB in size. When a hit occurs in the filter cache, it can be sent directly to the processor with decreased latency and reduced energy consumption, due to its near distance to the processor. This makes it highly energy-efficient. When a miss is predicted to occur in the filter cache, it can be completely skipped and a direct access to the L1 cache can be made. The filter cache implementation can effectively identify 92.2% of the misses on average [5]. In these cases, the filter cache can be skipped and the L1-D cache is directly accessed. However, the usage of the filter cache increases the cache miss rate because of its low size.

In multi-core processor-based computing platforms, the last level cache (LLC) is shared between all the different cores to augment the performance. However, this is likely to result in cache pollution, as low temporal locality data can lead to the eviction of other data with high temporal locality. Page Reusability-Based Cache Partitioning (PRCP) [6] migrates all pages into either a low-reusability or a high-reusability area, thus providing for more effective partitioning of the cache. The "low-reusability group" is given the smallest share of the LLC via page colouring, leading to PRCP achieving performance improvements comparable to that of an optimal process-based static cache partitioning scheme. The major benefit stems from the fact that the PRCP approach requires no additional support in terms of hardware, changes in applications or a prior understanding of the characteristics of workloads for its functioning, thus reducing the overheads to a great extent.

Partitioning of the cache memory results in applications receiving dynamic capacities based on their requirements, thus affecting the underlying replacement policy. A "replacement policy adaptable miss curve estimation" (RME) [7] has been proposed to estimate dynamic workload patterns for a given application, based on any chosen replacement policy. The last level cache (LLC) is shared among all applications by the cache partitioning technique. The applications are expected to completely occupy the allotted area, leading to a reduction in cross-application interference and imbalance in capacity among applications. For an effective implementation of the partitioning technique, it is required to obtain an accurate prediction of the performance of the various applications in the shared last level cache with respect to the partition size they have been allotted with. This is obtained with the help of a miss curve, which is a discrete graph comparing the cache miss count against cache size. This partitioning scheme can be effectively configured with existing powerful replacement policies, thus leading to enhancement in performance.

In case of VLIW (Very Long Instruction Word) processors, the cache blocks must be shared when switching between different performance modes. In many architectures, unprecedented task switching between the cores lead to reduction in cache performance. This is due to the constant triggering of cache resize operations, which require the data and instructions to move from their present cores to their new destinations, thus resulting in direct misses. Cache downsizing can result in degradation of performance, with the worst case being when all live data present is evicted, causing a cold start. To overcome this, an adaptive cache methodology [8] is suggested that tracks the change of modes in the processor and mitigates the impact of the sharp increase in cache miss rate during a downsizing operation. The processor mode change is treated as a trigger to reconfigure the data cache. A transition period is implemented before the resizing operation takes place, where the active data is migrated and thus a smooth transition between cache sizes is achieved.

The implementation of the Least Recently Used cache replacement policy on a field programmable gate array (FPGA) is carried out in [9], with VHDL being used for the programming configuration. While most implementations use a static design for the cache model, a reconfigurable approach has been explored using a unique hardware design, thus allowing the parameters to be set dynamically by the processor, including cache size and mapping, based on application specifications. Two different implementations of the reconfigurable LRU cache have been illustrated, namely 'Tree-based pseudo LRU' and Conventional LRU. 'Tree-based pseudo LRU' requires a victim selection algorithm to select which block needs to be replaced on the occurrence of a cache miss. A conventional LRU implementation has been carried out using eight hex digits for each line in the cache memory, which determines the maximum associativity value.

The Reputation Based Cache Management Policy [10] is a multi-caching policy that takes replacement decisions based on multi-level cache management techniques. It uses multi-step history hint information to make its decisions for promotions and demotions. The presence of demote hints helps carry the parameters associated with blocks from higher levels, while promote hints aid in carrying the parameters of the blocks from lower levels. Each block that enters the cache is initialized with its own "K-step hint value" (KHV) [11]. The promotion algorithm is based upon the numeric KHV value assigned initially to the cache blocks. Promotion is done based on whether the existing block has a higher KHV value than the incoming one. Similarly, the demotion algorithm bases the demotion of the block to a lower level based on a similar approach.

The LRU policy is designed for workloads having cyclic access patterns [12]. A variation of LRU based on LIP (LRU Insertion Policy), when combined with re-reference counts [13] has been found to reduce the miss rate associated with the L2 cache. The linked list of blocks, referred to as the RRIP chain, is ordered based on the probability of the blocks being referenced in the future, with the blocks at the head being re-refenced almost immediately. This implies that on a cache miss, removal of the block at the end of this list will prove beneficial. On a hit, the block's re-reference count is compared to a fixed threshold value. If it is found to be higher, the cache block is promoted to the most recently used position, else it is promoted to the least recently used position. Since the references between LLC and main memory are greatly avoided, this policy gives a great performance improvement..

The policy based on Expected Hit Count is established on the "strong correlation between the expected number of hits of a block and the reciprocal of its reuse distance" [14]. Blocks that are estimated to have lesser hits in the future are prioritized to be evicted. Re-Reference Interval Prediction (RRIP) is used to measure the number of hits a block experiences. A Re-Reference Prediction Value (RRPV) is associated with every cache block and the difference between the number of further hits a block is expected to receive and RRPV is used for the victim selection.

The Average Eviction Time model [15] utilizes reuse time and eviction time as parameters in the cache miss estimation. The time interval between the current access and the most recent access of a data block is referred to as reuse time. The time interval between the final access to a block and its eventual eviction is referred to as eviction time. It determines the amount of time the block is expected to be a resident of the cache after its final access takes place. The AET model is based on the cache miss probability hypothesis which states that when the reuse time of the block currently being accessed is higher than the average, it is likely to result in a cache miss. This is due to the fact that the probability of references to blocks with a reuse time higher than the average is the probability of the block being no longer available in the cache.

III. Methodology

The methodology consists of a recency time prefilter and a re-reference Interval Prediction (RRIP) predictor [16], as shown in Fig. 1. The aim is to augment the accuracy of the RRIP predictor by first passing the data blocks through the recency time prefilter. Cache data blocks are fed to the prefilter resulting in blocks ranked based on their recency time. A subset of blocks is obtained based on the threshold parameter (ideally the average recency time) and this subset is further passed to the RRIP predictor which will evict a data block based on the Re-reference Prediction Values (RRPV). The newly designed policy is thus aptly named RT-RRIP, based on the two stages of operation.

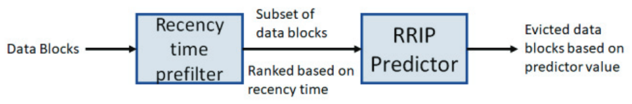

Data Blocks
Recency time prefilter
Subset of data blocks
Ranked based on recency time
RRIP Predictor
Evicted data blocks based on predictor value

Fig. 1. Block diagram of the methodology

Recency Time Prefilter

On the occurrence of a cache miss, a victim cache block needs to be evicted to make place for a newly inserted block. The process of eviction is done by first filtering the data blocks based on recency time. Recency time is defined as the time at which the block was most recently accessed. Thus, the recency time prefilter filters a certain number of blocks and feeds the subset of blocks to the predictor for further computation of the block to be evicted.

Each block entering the cache is assigned a recency time. In the event of a cache hit or on the insertion of a new block, the recency time of that block in the cache is updated. If there is a cache miss, a threshold value is set based on the average recency time of all the blocks. The blocks having recency time lesser than the set threshold value are fed to the predictor.

B. Re-Reference Interval Predictor (RRIP)

The selected blocks for eviction from the prefilter are given as an input to the predictor. The RRIP predictor associates each entering block with a Re-reference Prediction Value (RRPV). It assigns all blocks with one of 2^k possible RRPVs, with k being initially defined. Newly entering blocks are assumed to have a “long” re-reference interval (i.e., an intermediate re-reference interval with higher possibility of the interval being distant) and are thus assigned an RRPV of 2^k -2. On the occurrence of a cache miss, the predictor evicts the leftmost (first) block that has an RRPV of 2^k -1 (a block whose re-reference interval has been predicted to be distant). If none of the selected blocks have an RRPV of 2^k -1, then the predictor increments the RRPV of all the selected blocks until a block with such value is found. In case of a cache hit, the RRPV of the accessed block is decremented.

IV. Algorithm

The replacement policy is triggered whenever there is an occurrence of a cache miss, i.e., the referenced cache block is not available in the current state of the cache memory, as shown in Fig. 2. Therefore, the role of the replacement policy is to choose a block which is least likely to be referenced in the near future. The RT-RRIP replacement policy achieves this by associating each cache block with an additional parameter aptly named ‘recency time’. On the occurrence of a cache miss, all the blocks of the cache are visited and only those blocks whose recency time is lesser than a set threshold (average of all recency times) are passed to the re-reference interval predictor. The candidate to be evicted is thus chosen out of this reduced subset of data blocks.

ALGORITHM: Newly Designed Replacement Policy: RT-RRIP

```
Input: Data block to be accessed
Output: Data block to be evicted
// All blocks are assigned one of 2^k possible RRPVs, with k being
initially defined
 1: function ReplacementPolicy
 2:     if (cache miss)
          // Recency time prefilter
 3:       for (all the data blocks in the cache)
 4:           if (recency time <= set threshold)
 5:              Pass the data blocks to the predictor for eviction
 6:       end for
          // RRIP predictor
          // Input: Selected data blocks from the prefilter
 7:       while (RRPV (of any the selected data block)! = 2^k-1) do
 8:             for (all the selected data blocks)
 9:                   RRPV (data block) ++
10:             end for
11:       end while
12:         evict (leftmost data block with RRPV of 2^k-1)
13:         Add the accessed data block to the cache
14:         Associate accessed data block with recency time
15:         RRPV (accessed data block) <- 2^k-2
16:     else   // In case of a hit
17:         Update the recency time of the accessed data block
18:         RRPV (accessed data block) --
19: end function
```

Fig. 2. Algorithm of the RT-RRIP policy

The RRIP predictor requires a block with re-reference prediction value of 2^k-1 to perform the eviction. If no such candidate is available, the RRPV values associated with all blocks are continuously incremented until at least one block satisfies the criterion. The leftmost candidate with re-reference prediction value of 2^k -1 is evicted and the newly arriving data block is placed into the cache. The recency time of the new block is then calculated, while the recency time of the existing data blocks are updated. The re-reference prediction value of the accessed data block is assigned as 2^k -2 as per the operation of a regular re-reference interval prediction model. On a cache hit, the recency time associated with each data block is updated, and the re-reference prediction value of the accessed cache block is decremented, as per the operation of a regular re-reference interval prediction model.

V. Analysis And Results

To analyze the expected improvement in cache hit rate using the proposed methodology, the GEM5 simulator is used [17]. GEM5 is an open-source modular system-level simulator used for computer architecture research. The modular approach helps model specific components within a computer system. It is especially preferred for its flexible implementation of memory systems, consisting of caches and primary memory modules. It supports multiple instruction set architectures (ISAs), including ARM, MIPS, RISC-V, ALPHA and x86 architectures to name a few. The ALPHA architecture has been chosen for the simulations.

The 'Standard Performance Evaluation Corporation' (SPEC) provides globally-accepted and standardized benchmarks for the analytical measurement of many computer performance parameters. The SPEC CPU2006 benchmark suite has long been accepted as the industry-standard benchmark for CPU-intensive analytics, with tests performed on both the processor as well as its memory subsystem. The specific benchmarks from this compilation that have been used during the evaluation phase include LBM, BZIP2, NAMD and LESLIE3D [18].

A. *Performance of RT-RRIP over various cache size and associativity combinations*

Simulations were run on the ALPHA architecture for analysing the change in miss rate and miss latency over a range of cache sizes and associativity values for the L1-I cache, with the other cache levels having fixed parameters of 32KB with 4-way associativity for the L1-D cache and 256KB with 8-way associativity for the L2 cache, as outlined in Fig. 3 and Fig. 4.

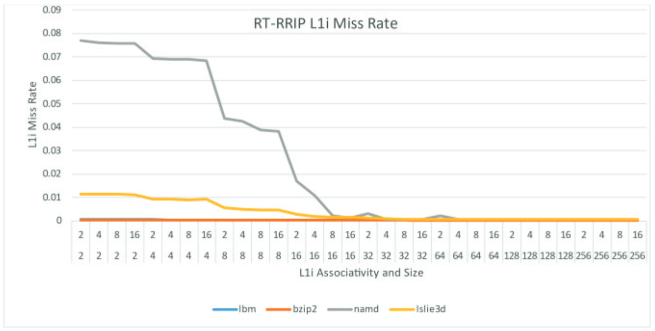


Fig. 3. Variation of L1-I cache miss rate under various workloads for the RT-RRIP Replacement Policy

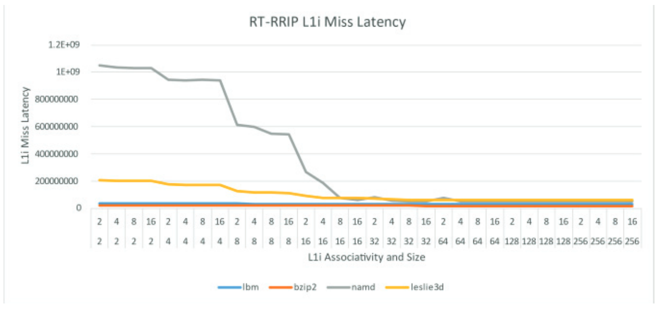


Fig. 4. Variation of L1-I cache miss latency under various workloads for the RT-RRIP Replacement Policy

The change in miss rate and miss latency over a range of cache sizes and associativity values for the L1-D cache were analysed, with a 32KB 4-way associative L1-I cache and 256KB 8-way associative L2 cache, as outlined in Fig. 5 and Fig. 6.

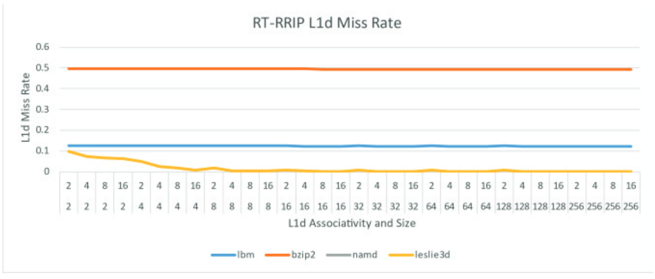


Fig. 5. Variation of L1-D cache miss rate under various workloads for the RT-RRIP Replacement Policy

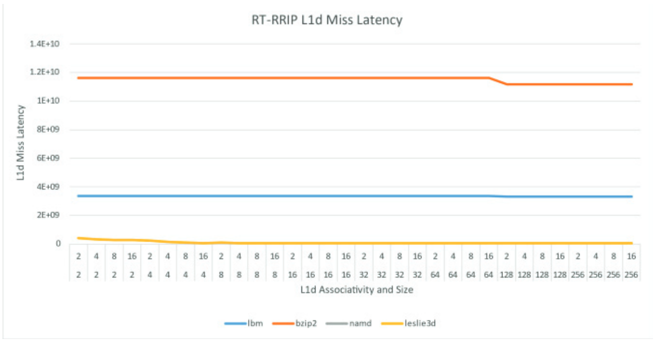


Fig. 6. Variation of L1-D cache miss latency under various workloads for the RT-RRIP Replacement Policy

The change in miss rate and miss latency over a range of cache sizes and associativity values for the L2 cache were analysed, with 32KB 4-way associative L1-I cache and L1-D cache, as outlined in Fig. 7 and Fig. 8.

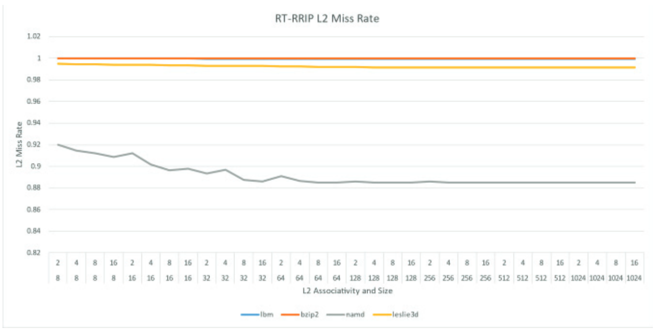


Fig. 7. Variation of L2 cache miss rate under various workloads for the RT-RRIP Replacement Policy

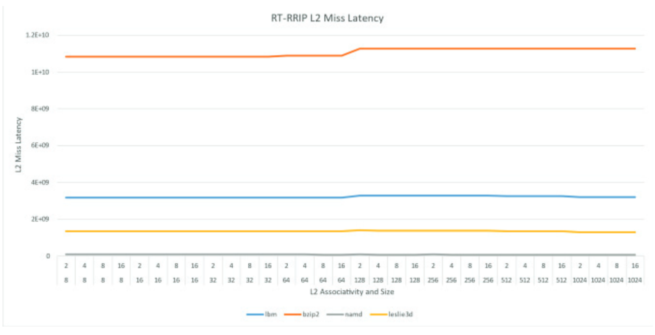


Fig. 8. Variation of L2 cache miss latency under various workloads for the RT-RRIP Replacement Policy

In all the cases, both the cache miss rate and miss latency tend to show downward trends as the variation in size and associativity is increased for certain workloads, while it remains approximately at a constant level with negligible difference for the other workloads.

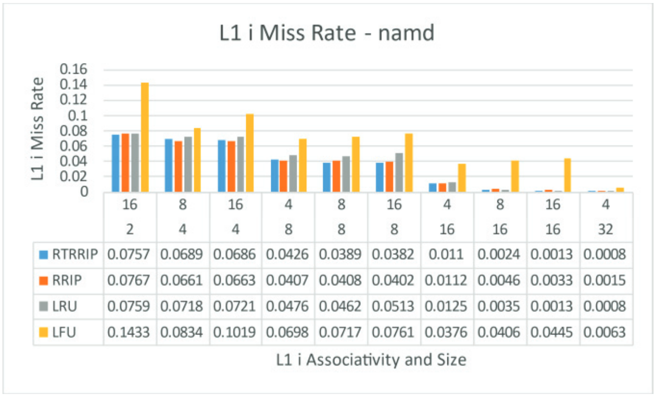
Comparing RT-RRIP to policies reported in literature

The newly developed RT-RRIP replacement policy has been compared with existing replacement policies and the performance has been evaluated across different workloads for average performance improvement over LRU and RRIP as represented in Table 1.

TABLE I. COMPARISON OF RT-RRIP WITH RRIP AND LRU

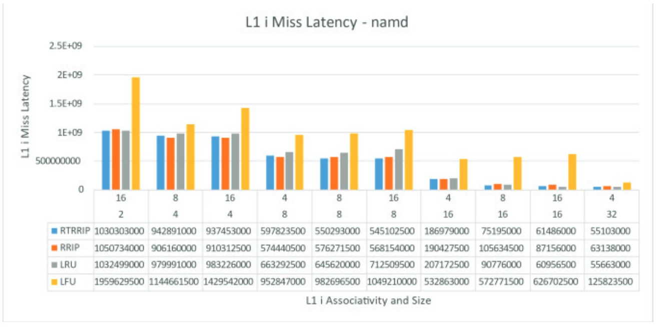
Parameter	***% Improvement over RRIP***	***% Improvement over LRU***
L1-I Miss Rate	12.49	4.15
L1-I Miss Latency	3.96	3.16
L1-D Miss Rate	5.92	2.24
L1-D Miss Latency	2.13	1.32

Fig. 9 and Fig. 10 show the comparison of RT-RRIP with RRIP, LRU and LFU for varying L1-I cache sizes and associativity values, with a 32KB 4-way associative L1-D cache and 256KB 8-way associative L2 cache.



	16	8	16	4	8	16	4	8	16	4
	2	4	4	8	8	8	16	16	16	32
RTRRIP	0.0757	0.0689	0.0686	0.0426	0.0389	0.0382	0.011	0.0024	0.0013	0.0008
RRIP	0.0767	0.0661	0.0663	0.0407	0.0408	0.0402	0.0112	0.0046	0.0033	0.0015
LRU	0.0759	0.0718	0.0721	0.0476	0.0462	0.0513	0.0125	0.0035	0.0013	0.0008
LFU	0.1433	0.0834	0.1019	0.0698	0.0717	0.0761	0.0376	0.0406	0.0445	0.0063

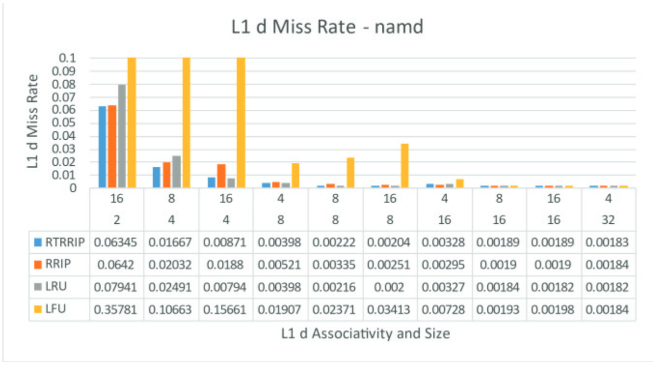
Fig. 9. Variation of L1-I cache miss rate under namd workload for various replacement policies



	16	8	16	4	8	16	4	8	16	4
	2	4	4	8	8	8	16	16	16	32
RTRRIP	1030303000	942891000	937453000	597823500	550293000	545102500	186979000	75195000	61486000	55103000
RRIP	1050734000	906160000	910312500	574440500	576271500	568154000	190427500	105634500	87156000	63138000
LRU	1032499000	979991000	983226000	663292500	645620000	712509500	207172500	90776000	60956500	55663000
LFU	1959629500	1144661500	1429542000	952847000	982696500	1049210000	532863000	572771500	626702500	125823500

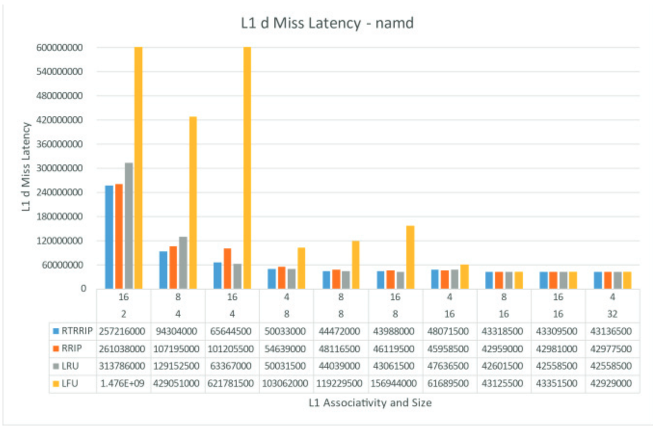
Fig. 10. Variation of L1-I cache miss latency under namd workload for various replacement policies

Fig. 11 and Fig. 12 show the comparison of RT-RRIP with RRIP, LRU and LFU for varying L1-D cache sizes and associativity values, with a 32KB 4-way associative L1-I cache and 256KB 8-way associative L2 cache.



	16	8	16	4	8	16	4	8	16	4
	2	4	4	8	8	8	16	16	16	32
RTRRIP	0.06345	0.01667	0.00871	0.00398	0.00222	0.00204	0.00328	0.00189	0.00189	0.00183
RRIP	0.0642	0.02032	0.0188	0.00521	0.00335	0.00251	0.00295	0.0019	0.0019	0.00184
LRU	0.07941	0.02491	0.00794	0.00398	0.00216	0.002	0.00327	0.00184	0.00182	0.00182
LFU	0.35781	0.10663	0.15661	0.01907	0.02371	0.03413	0.00728	0.00193	0.00198	0.00184

Fig. 11. Variation of L1-D cache miss rate under namd workload for various replacement policies



	16	8	16	4	8	16	4	8	16	4
	2	4	4	8	8	8	16	16	16	32
RTRRIP	257216000	94304000	65644500	50033000	44472000	43988000	48071500	43318500	43309500	43136500
RRIP	261038000	107195000	101205500	54639000	48116500	46119500	45958500	42959000	42981000	42977500
LRU	313786000	129152500	63367000	50031500	44039000	43061500	47636500	42601500	42558500	42558500
LFU	1.476E+09	429051000	621781500	103062000	119229500	156944000	61689500	43125500	43351500	42929000

Fig. 12. Variation of L1-D cache miss latency under namd workload for various replacement policies

VI. CONCLUSION

To build on the limitations of existing policies and augment the performance even further, a newly-designed and improved cache replacement policy built upon the popular Re-Reference Interval Prediction (RRIP) model was implemented, using the concept of recency time to decrease the occurrence of cache misses and reduce the miss latency. An extensive range of simulations were performed over all regularly used cache size and associativity values, along with the usage of standardized SPEC benchmarks.

While the L1-I cache showed more instances of an improvement for all the workloads simulated, the L1-D cache also showed many combinations where RT-RRIP was the clear winner in terms of both cache miss rate and miss latency. However, the optimization of the replacement policy for the L2 cache can be worked upon and improved, since most cases did not yield favourable results. An adaptive policy built on RT-RRIP that can cater to frequency-based workloads would be an added advantage, since it was observed that most of the workloads were frequency-favouring in terms of their L2 accesses.

REFERENCES

[1] S. Kumar and P. K. Singh, "An overview of modern cache memory and performance analysis of replacement policies," 2016 IEEE International Conference on Engineering and Technology (ICETECH), 2016, pp. 210-214.

[2] A. Jain and C. Lin, “Back to the future: Leveraging Belady’s algorithm for improved cache replacement,” in Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit., 2016, pp. 78–89.

[3] O. D. Olorode and M. Nourani, "Improving Cache Power and Performance Using Deterministic Naps and Early Miss Detection," in IEEE Transactions on Multi-Scale Computing Systems, vol. 1, no. 3, 1 July-Sept. 2015, pp. 150-158.

[4] J. Kin, M. Gupta, and W. H. Mangione-Smith, “The filter cache: An energy efficient memory structure,” in Proc. 30th Annu. IEEE/ACM Int. Symp. Microarchit., Dec. 1997, pp. 184–193.

[5] J. Lee and S. Kim, " Filter Data Cache: An Energy-Efficient Small L0 Data Cache Architecture Driven by Miss Cost Reduction," in IEEE Transactions on Computers, vol.64, no. 7, 1 July 2015, pp. 1927-1939.

[6] J. Park, H. Yeom and Y. Son, "Page Reusability-Based Cache Partitioning for Multi-Core Systems," in IEEE Transactions on Computers, vol. 69, no. 6, 1 June 2020, pp. 812-818.

[7] B. Lee, K. Kim and E. Chung, "Replacement Policy Adaptable Miss Curve Estimation for Efficient Cache Partitioning," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 2, Feb. 2018, pp. 445-457.

[8] S. Hu and J. Haung, "Exploring Adaptive Cache for Reconfigurable VLIW Processor," in IEEE Access, vol. 7, 2019, pp. 72634-72646.

[9] S. S. Omran and I. A. Amory, "Implementation of LRU Replacement Policy for Reconfigurable Cache Memory Using FPGA," International Conference on Advanced Science and Engineering (ICOASE), Duhok, 2018, pp. 13-18.

[10] K. Kelwade, S. Sahu, G. Kawade, N. Korde, S. Upadhye and M. Motghare, "Reputation based cache management policy for performance improvement," 2017 International Conference on Intelligent Sustainable Systems (ICISS), Palladam, 2017, pp. 582-587.

[11] C. Wu, X. He, Q. Cao and C. Xie, "Hint-K: An Efficient Multi-level Cache Using K-Step Hints," 2010 39th International Conference on Parallel Processing, 2010, pp. 624-633.

[12] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely Jr. and J. Emer, “Adaptive insertion policies for high-performance caching”, in International Symposium on Computer Architecture (ISCA), 2007, pp 381-391.

[13] S. Sreedharan and S. Asokan, "A cache replacement policy based on re-reference count," 2017 International Conference on Inventive Communication and Computational Technologies (ICICCT), Coimbatore, 2017, pp. 129-134.

[14] A. Vakil-Ghahani, S. Mahdizadeh-Shahri, M. Lotfi-Namin, M. Bakhshalipour, P. Lotfi-Kamran and H. Sarbazi-Azad, "Cache Replacement Policy Based on Expected Hit Count," in IEEE Computer Architecture Letters, vol. 17, no. 1, 1 Jan.-June 2018, pp. 64-67.

[15] Xiameng Hu, Xiaolin Wang, Lan Zhou, Yingwei Luo, Zhenlin Wang, Chen Ding,and Chencheng Ye, “Fast Miss Ratio Curve Modeling for Storage Cache,” in ACM Transactions on Storage, vol. 14,no. 2, Article 12, April 2018, pp 1-34.

[16] A. Jaleel, K. B. Theobald, S. C. Steely Jr., and J. Emer, “High performance cache replacement using re-reference interval prediction (RRIP),” in ACM SIGARCH Comput. Archit. News, vol. 38, no. 3, 2010, pp. 60–71.

[17] N. Binkert et al., “The GEM5 simulator,” in ACM SIGARCH Comput. Archit. News, vol. 39, no. 2, 2011, pp. 1–7.

[18] John L. Henning, “SPEC CPU2006 benchmark descriptions,” in ACM SIGARCH Comput. Archit. News, vol. 34, no. 4, 2006, pp 1–17.