ISSN No:-2456-2165

Dynamic Non-Decaying ABRIP for Shared Level 3 Cache Memory Systems

Nirmol Munvar*, Shoba Gopalakrishnan‡, Arati Phadke†
Department of Electronics Engineering,
K. J. Somaiya College of Engineering

Abstract:- When all blocks are occupied inside a cache of a processor, the system uses a concept known as 'Replacement policy' to evict an existing block inside the cache memory to bring the required block from the main memory. There are many replacement policies applied to caches which itself have many levels. The policy 'Application Behavior Aware Re-Reference Interval Prediction' is an extension of existing policy of 'Re-Reference Interval Prediction.' The ABRIP policy has two levels of RRPV, implemented in two levels as we will see further. However, the ABRIP policy's algorithm waits until all the cache blocks have max RRPV value hence 'decaying' the cache blocks. Proposed method intends to correct the decaying phenomenon by implementing Dynamic Non-decaying ABRIP or DND-ABRIP. Gem5 simulator was used in system emulation mode with SPEC2006 Benchmarks. We see a 0.1% improvement in IPC, 1.4% improvement in Read hits and 0.3% improvement in Write Hits for DND-ABRIP over ABRIP.

***Keywords*:-** *Replacement Policy, Re-Reference Interval Prediction, Application Behavioral Re-Reference Interval Prediction, decaying, Dynamic Non-decaying ABRIP.*

I. INTRODUCTION

With simplescalar or single core architecture being obsolete in todays modern processors, we have processors which have superscalar architecture, with both homogeneous and heterogeneous architectures. However, when you have a multicore environment, in the cache architecture the last level cache (LLC) is shared amongst all the cores, and then each core has two different levels of caches of their own, namely L2 and L1. Hence with the increase in number of cores, the burden on LLC increases and as a result efficient replacement policy need to be implemented to make the LLC less burdened. As multicore architectures have different processes i.e. workloads being executed, the data that is written/read from the cache blocks is of multiple nature. These cache blocks containing a variety of data can cause conflict between cores and can increase in the number of conflict misses. Current cache replacement techniques see the data reusability and make decisions to replace the cache blocks. However, this is only efficient in L1 and L2, at LLC where we have a variety of nature in cache blocks in terms of their reusability, conventional replacement techniques like LRU are not good enough to efficiently use the LLC of cache friendly and streaming type applications from SPEC2006 benchmark suite.

RRIP [1] or Re-Reference Interval prediction was one of the policies that was proposed, which was more efficient in taking account of the block reusability over LRU technique. However, the prediction accuracy depends upon the access rate of the application in case of RRIP policy. As a result, applications with frequent-access workloads interfere with applications with less frequent access workloads. Today almost every CMP both homogenous and heterogenous architectures have these kinds of mixed application with different access rates which utilize the LLC in different rates. Application Behavioral Re-Reference Interval Prediction of ABRIP [2] for LLC was another method that was proposed to address the issue of diversity in data stored in cache blocks of LLC. It uses two levels of RRPV, one over cache block and other over the core with a scaled weight over the core level RRPV and decide whether the cache block is to be replaced or not. As a result, cores with higher access rates can be differentiated with cores with lower access rates and hence LLC can be used effectively. But the ABRIP policy has a small 'decaying' issue which will be seen in detail in section III.

In this paper we propose a slight modification to the ABRIP policy, which resolves the decaying problem in it. The technique is called as Dynamic Non-Decaying ABRIP. This not only reduces the interference between the different types of workloads, but also solves the decaying issue. To do that we monitor both of the RRPV [1],[2] values, at core and at the cache block level and get the final ABr value, by combining the both RRPV values with the weight on the core level RRPV, we decide the block to be replaced, however if none of the blocks are at the maximum threshold but all the cache blocks are filled, then we evict the cache block with the maximum amount of combined RRPV value from all the cache blocks. We also see the quadcore implementation of ABRIP and BRRIP, which enables us to observe these policies for a more flexible sample space. In this project we see that ABRIP and BRRIP perform in a similar range for already tested dual-core environment. The DND-ABRIP policy shows an improvement of 0.1% over ABRIP in average IPC, while running on multiple quad-mixes.

Memory Requests	LRU				Hit/Miss	RRIP				Hit/Miss
a1	0	0	0	0	Miss	0	0	0	0	Miss
a2	a1	0	0	0	Miss	a1	0	0	0	Miss
a2	a2	a1	0	0	Hit	a2	a1	0	0	Hit
a1	a2	a1	0	0	Hit	a2	a1	0	0	Hit
b1	a2	a1	0	0	Miss	a2	a1	0	0	Miss
b2	b1	a2	a1	0	Miss	b1	a2	a1	0	Miss
b3	b2	b1	a2	a1	Miss	b2	b1	a2	a1	Miss
b4	b3	b2	b1	a2	Miss	b3	b2	a2	a1	Miss
a1	b4	b3	b2	b1	Miss	b4	b3	a2	a1	Hit
a2	a1	b4	b3	b2	Miss	b4	b3	a2	a1	Hit
a2	a2	a1	b4	b3	Hit	b4	b3	a2	a1	Hit
a1	a2	a1	b4	b3	Hit	b4	b3	a2	a1	Hit
	a2	a1	b4	b3	Hit	b4	b3	a2	a1	Hit

RRPV = 0	
RRPV = 1	
RRPV = 2	
RRPV = 3	

Fig 1:- Observation of Importance of RRIP Policy Over Conventional LRU Policy with Access Pattern a1a2a2a1b1b2b3b4a1a2a2a1 [1]

The rest of the paper is divided into the following sections. Section II shows the motivation behind this project. The Quadcore Implementation of BRRIP and ABRIP is shown in section III and DND-ABRIP is explained in section IV. The experimental methodology is explained in section V and section VI shows the results and observations. We conclude the paper in section VII.

II. MOTIVATION

Today there is a dire need of faster computing systems and hence there is a need to improve the speed of the system by any means. Some conventional means are improving the device physics of the transistors, scaling them down to accommodate more transistors inside the chip area and improving the processor architecture. When it comes to improvement in processor architecture, most improvements are done inside the main processor to improve speed and efficiency, and the memory architecture is overlooked. Even though cache memories are on the chip of the processor, they occupy very less space and hence, the priority to improve the performance of a processor is given to the main processor core and not the cache memory systems [8],[9],[10]. We need to also take into consideration of the fact that because the main processor core is given more priority, one cannot increase the number of cache memory cells as it would take up unnecessary area which can be better used for processor development.

Whenever data needs to fetched from the main memory to the LLC, the amount of cycles the operation consumes is very high. Hence there is a need to improve the utilization of LLC in CMP's for both homogenous and heterogenous cores without actually increasing the size or physics of the device, but by actually tampering the algorithms by which the cache systems work to utilize them more efficiently. Even with a single core environment we can see the effectiveness of RRIP policy as compared to the conventional LRU policy that is currently being used. DND-ABRIP is based on ABRIP which is based on RRIP. As we can see from Fig 1, RRIP policy has more hits for the same data blocks than LRU.

The data are in the pattern of a1a2a2a1b1b2b3b4a1a2a2a1, which means that some of the data blocks are being re-referred than other blocks, but after a certain amount of clock cycles. As a result, if those data blocks are kept inside the cache just long enough so that they are 'not' evicted from the cache when the core demands it. This is done by taking a new parameter for the cache blocks which is called as the Re-Reference Prediction Value or RRPV[1]. Unlike LRU where you would know that the data is either recently used or not, RRPV gives you a better insight on how recently it was referred. If the RRPV value was given by 'm' number of bits, then total possible RRPV values would be 2^m-1. So, if we consider that m = 2, then it means that total RRPV values would be from 0 to 3, which is 4 (same as in figure 1), hence it can stay inside the cache until the RRPV value has reached maximum which is 3. As a result, a block which is re-referred before its evicted, gets its RRPV value back to 0, and hence kept for a longer time inside the cache incase of another re-reference.

When we take ABRIP into consideration with the standard RRIP policy, ABRIP policy implements the RRPV values at two levels, one at the core, and one at the cache block itself (same as RRIP). Hence there is an additional RRPV parameter at the core. This is called as two-level RRPV. As discussed in section I, that single core systems have become obsolete, the multicore environments have a better way to separate the data blocks in terms of their re-reference by the core RRPV values. So, if we consider that in a dual core system, one of the cores is working with a heavy working set, with new data needed to fetched frequently and another core with a light working set and the same data in the cache can be re-used, then there can be interference issues inside the LLC because of the difference in nature of the data that is fetched in.

ISSN No:-2456-2165

Memory Requests	RRIP				Hit/Miss	ABRIP with Cr Weight α = 4						Hit/Miss
						Weighted Core RRPV						
a1	0	0	0	0	Miss	a-8	b-X	0	0	0	0	Miss
b1	a1	0	0	0	Miss	a-8	b-8	a1	0	0	0	Miss
b2	b1	a1	0	0	Miss	a-8	b-12	b1	a1	0	0	Miss
b3	b2	b1	a1	0	Miss	a-8	b-12	b2	b1	a1	0	Miss
a2	b3	b2	b1	a1	Miss	a-8	b-12	b3	b2	b1	a1	Miss
b4	a2	b3	b2	b1	Miss	a-12	b-12	b3	b2	a2	a1	Miss
b5	a2	b4	b3	b2	Miss	a-12	b-12	b4	b3	a2	a1	Miss
b6	b5	b4	b3	b2	Miss	a-12	b-12	b5	b4	a2	a1	Miss
a2	b6	b5	b4	b3	Miss	a-12	b-12	b6	b5	a2	a1	Hit
b7	a2	b6	b5	b4	Miss	a-0	b-12	b6	b5	a2	a1	Miss
b8	a2	b7	b6	b5	Miss	a-0	b-12	b7	b6	a2	a1	Miss
b9	b8	b7	b6	b5	Miss	a-0	b-12	b8	b7	a2	a1	Miss
a1	b9	b8	b7	b6	Miss	a-0	b-12	b9	b8	a2	a1	Miss
	a1	b9	b8	b7	Miss			b9	b8	a2	a1	Hit

RRPV = 3	
RRPV = 2	
RRPV = 1	
RRPV = 0	

Fig 2:- Observation of ABRIP in a Multicore Environment with Betterment over RRIP on the Workload Patter a1b1b2b3a2b4b5b6a2b7b8b9a1 [2]

As we can see in figure 2, the ABRIP policy in a multicore environment (dual core) is showing a more efficient performance than RRIP. The access pattern of data is a1b1b2b3a2b4b5b6a2b7b8b9a1…., with workload 'a' given to the first core and 'b' to the other. We can see from the type of access pattern the difference between the nature of the data blocks that are fetched into the LLC. When we look at RRIP, it has no efficient mechanism to differentiate between the cores, and hence workload 'a' is evicted before it is re-referred. However, if we see at ABRIP, because of RRPV at core level, we can now correctly differentiate between the cores and the access rates of the cores. If a core demands a data block and it is present in the cache (hit), then both the core RRPV and block RRPV are made 0, hence the cache friendlier data workloads remain the cache longer as compared to less friendlier workloads or streaming type workloads. The core RRPV value or the Cr is weighted with a weight 'α', as a result, the core RRPV value is of greater significance as compared to the block RRPV value in determining which block to be evicted.

However, as you can see in the same figure (Fig 2), between memory requests a2 and b4, the RRPV values of the cache blocks for both instances are '2', which isn't maximum, and the eviction only takes place whenever the RRPV values reaches maximum. This is a issue as it will have to wait an additional clock cycle to increment the RRPV values to maximum. This is a slight glimpse of the decaying issue which will be explained in section IV in further detail. On seeing the case shown in figure 2, we see that the system is of dual core, but if we increase the number of cores to 4 or 8, there can be more interference and more cores with different diverse data types in the same LLC, which can lead to further clock cycles being delayed because of the decaying. As a result, DND-ABRIP is proposed, which addresses the decaying issue.

III. QUADCORE IMPLEMENTATION OF RRIP AND ABRIP

As explained in section II and I, that most systems today are on quadcore or more. Furthermore, the increase in number of cores increase the sample space for number of benchmarks running at the same time, which can give us newer insights in the working of both RRIP and ABRIP polices. In [2], we see the ABRIP policy for a dual core environment, showing an average of 16% improvement over RRIP. This paper shows the working of both RRIP and ABRIP over quad cores.

ISSN No:-2456-2165

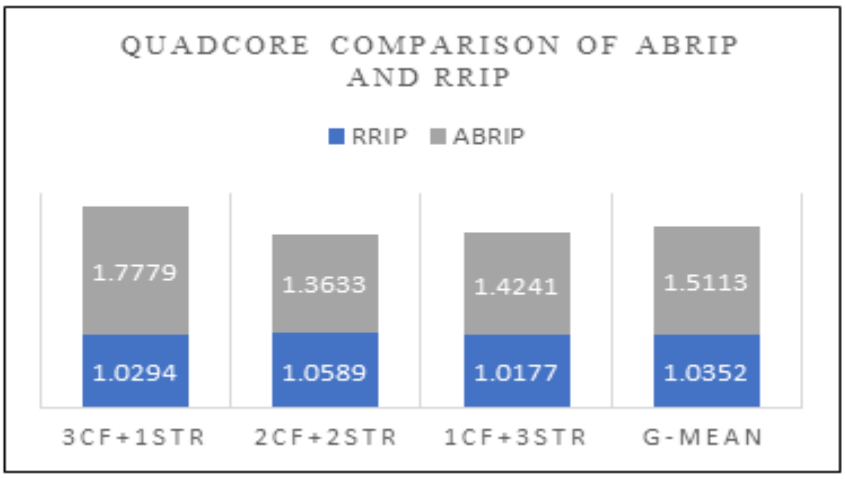

Fig 3:- Quadcore IPC Comparison of ABRIP and RRIP

a2	b3	b2	b1	a1	Miss	a-8	b-12	b3	b2	b1	a1	Miss
One Cycle Skipped to Increment RRPV values												
a2	b3	b2	b1	a1	Miss	a-8	b-12	b3	b2	b1	a1	Miss
b4	a2	b3	b2	b1	Miss	a-12	b-12	b3	b2	a2	a1	Miss

Fig 4 Observation of Decaying in a Typical Mixed Workload Operation

Algorithm 1 Victim selection policy for ABRip

```
1: iterate:
2: if any ABr==max_ABr then
3:     Victim found return victim index
4: else if any Cr==max_Cr then
5:     all Br++ goto iterate
6: else
7:     all Cr++
8:     all Br++ goto iterate
```

As we can see from figure 3, ABRIP still holds to be the better replacement policy as compared to RRIP even in the quadcore environment. The importance of having a quadcore simulation is that the LLC size doesn't change much, however when you have a greater number of cores working at different levels of workloads at the same time, the data inside the LLC becomes more and more diverse. We can also see the interactive effects of having three programs that are cache friendly and one streaming (3CF+1STR) and on the complete other way around one cache friendly and 3 streaming type programs (1CF+3STR).

From figure 2 we can see that ABRIP policy favors the cache friendly type of behavior where one cache friendly and one streaming type workloads is executed. From the similar result that we see from figure 3 we can say that ABRIP keep the cache friendlier workloads as compared to streaming ones despite having more than one cache friendly workloads. However, if all the workloads are cache friendly, the management of ABRIP and the effectiveness of separating the LLC in favor of cache friendly data workloads fail as all the workloads are of cache friendly types.

IV. DYNAMIC NON-DECAYING ABRIP

A. The Decaying Issue in ABRIP

As it is evident from figure 4, the processor waits until one of the cache blocks reaches the maximum RRPV value, hence if the entire cache is filled but none of them have RRPV value as maximum, then those cache blocks will not be evicted and the RRPV values will be incremented in subsequent clock cycles until one of them reaches maximum RRPV value. The phenomenon of the cache blocks waiting for getting their RRPV value to maximum is called the *decaying issue* and it is observed both in ABRIP and RRIP (figure 2). This is also evident from the algorithm proposed for ABRIP [2] where we can see that if the cache blocks do not reach the *maxAbr* value, they will not be evicted.

We can also have a situation where all cache blocks with very low RRPV values, but the entire cache block is filled, hence it has to wait for more clock cycles as

compared to figure 4 unless one of them reaches a maximum ABr or RRPV value. Hence the decaying of cache blocks is for a long time, further making the processor wait and decreasing the overall IPC of the operations of the multicore architecture. We also see the effect of multiple cores working on the same size of LLC (usually in the range of up to 20 Mbytes), which results in a higher variety in the nature of the workloads of the data present or fetched inside the cache blocks. As a result, it is necessary to monitor the decaying issue as higher number of cores for the same size of LLC creates larger bottlenecks and larger decaying periods.

As we can see from the above proposed algorithm a cache block will only be evicted if there is a block with maximum ABr value (max_ABr), else there would be no eviction. As a result, if the cache is full, and none of the cache blocks are of maximum ABr or RRPV values, then the processor will wait for a few cycles while incrementing the respective values and then evict the bock with the maximum ABr value.

B. Dynamic Non-Decaying ABRIP

In order to remove the decaying issue, we make a slight change in the victim selection algorithm of the ABRIP policy which is given in *Algorithm 2*

Algorithm 2 Victim selection policy for DND ABRip

Iterate

1. Perform same ABr calculation for each block as that of ABRIP
2. **If** block ABr=max_ABr **then**
Block = candidate
Return victim
Increment Cr and Br values
Go to Iterate
3. **Else if** block ABr > Candidate ABr **then**
Block = candidate
Return victim
Increment Cr and Br values
Go to Iterate

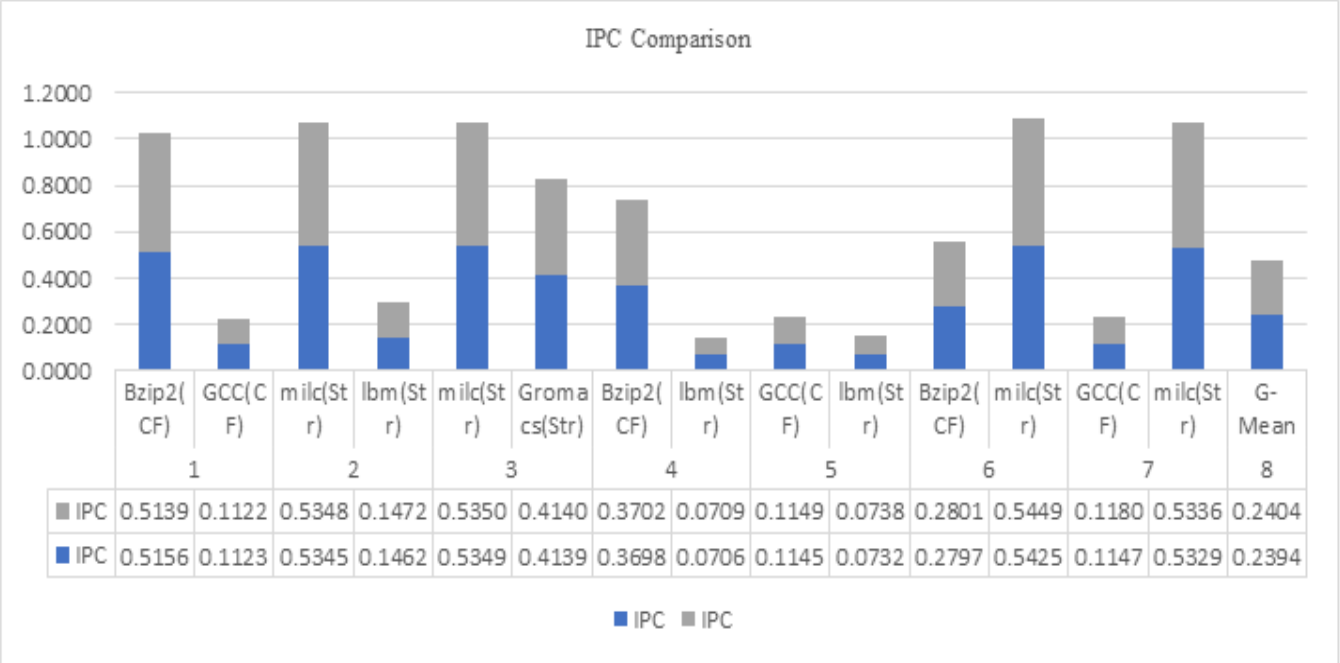

Fig 5:- IPC Comparison between ABRIP and DND-ABRIP

From the above algorithm we can see that while victim selection the processor will record the candidates ABr value and compare it with other blocks' ABr values, if any other block has ABr value = maximum it will immediately be selected for eviction, however if none of the blocks have maximum ABr value, then the cache block which has the maximum ABr value among all blocks in the cache will be selected for eviction. Hence removing the need of waiting to increment the RRPV or ABr values and solving the decaying issue.

V. EXPERITMENTAL METHODOLOGY

A. Simulating Environment

To demonstrate the working of algorithm 2, we have used Gem5 simulator. It is a multicore simulating environment with 4 cores working at a time, however, the environment supports invoking of additional cores if needed. The architecture used is X86 architecture with Gem5 using allotted main memory (RAM) of 512 Mbytes. It is observed under DerivO3 (out-of-order) execution type structure.

The L1 I and L1D private caches are of 64 and 32 Kbytes respectively with both working with 2-way associavity. The L2 private cache is of 2 Mbytes and has 8-way associavity and the LLC is of 16 Mbytes with 16-way associavity. We compare the ABRIP and DND-ABRIP policies over the LLC.

B. Benchmarks

We use 5 benchmark workloads in a combination of 7 types of mixes as shown in table 1. We run each benchmark for about 100 million instructions after fast forwarding 1 billion instructions in order to prevent the initial compulsory misses. We see the observation of each of the benchmark in a dual core environment which is same as that of [2]. The spec2006 benchmarks are linked with the gem5 simulator by adding a python script which enables us to observe and run these benchmarks via Gem5.

Mix No	Benchmarks	Mix Type
1	Bzip2+GCC	Cf+Cf
2	Milc+lbm	Str+Str
3	Milc+Gromacs	Str+Str
4	Bzip2+lbm	Cf+Str
5	GCC+lbm	Cf+Str
6	Bzip2+milc	Cf+Str
7	GCC+milc	Cf+Str

Table 1:- Benchmark Mixes used

VI. RESULTS

The IPC Comparison or improvement in instruction per cycles is given in figure 5. We see that DND-ABRIP is overall on an average 0.1% better than ABRIP in terms of IPC betterment. The number of read and write hits are improved by 1.4 and 0.3% respectively. Such a small improvement is observed as DND-ABRIP only removes decaying of a few clock cycles in the entire operation which gives only a slight improvement in the performance.

VII. CONCLUSIONS

There are two conclusions that can be drawn from this project.

As we can see from figure 3, the ABRIP is better than the existing policies. This was seen with workloads of different natures combined and executed simultaneously to give the result. The Quadcore assessment shows that not only ABRIP is scalable to a greater number of cores but it also has same improvement over RRIP i.e. 16%.

The second conclusion that can be drawn, as we can see from figure 5 is DND-ABRIP improves 0.1% overall IPC over ABRIP, as it saves a few clock cycles i.e. it doesn't allow the cache blocks to decay. Interestingly the heterogenous combination of Cf+Cf type workload has a decreased performance over ABRIP, which is as the same nature of ABRIP as ABRIP also has a decreased performance over RRIP in this situation.

ACKOWLEDGEMENT

I would firstly like to thank and dedicate this project to my guide, Dr. Shoba Gopalakrishnan, who despite of her doing a PhD and taking numerous lectures, always took the time to help me with the project and her guidance has been the biggest factor in the progress of the project. I would also like to thank my Co-Guide, Prof. Arti Phadke, for all her positive criticisms towards my presentation skills for both Reports and PPTs and also for her continued guidance and support during the year. I would then like to thank the experts, both departmental expert Prof. Anil Thosar and external expert Prof J, Kundargi for their support and guidance towards the project over the years. I would especially want to thank all of the aforementioned faculty members and Dean Dr. Sudha Gupta and Dr. R. Karandikar and Head of the Department, Dr. J. H. Nirmal for their support for me, when I had contracted bronchitis in between of the project and co-operating with me in the delay of the First Stage Presentation because of the aforementioned disease. I would like to thank my classmates and my family for their support and motivation and encouragement I had during the low points of the project and helping me to try again and again.

REFERENCES

[1]. Aamer Jaleel, Kevin B. Theobald, Simon C. Steely Jr., Joel Emer, "High Performance Cache Replacement using Re-Reference Interval Prediction" *2010 IEEE International Symposium on Computer Architecture (ISCA)*, Saint-Malo, France, 2010

[2]. P. Lathigara, S. Balachandran and V. Singh, "Application behavior aware re-reference interval prediction for shared LLC," 2015 33rd IEEE International Conference on Computer Design (ICCD), New York, NY, 2015, pp. 172-179.doi: 10.1109/ICCD.2015.7357099

[3]. Newton, S. K. Mahto, S. Pai and V. Singh, "DAAIP: Deadblock Aware Adaptive Insertion Policy for High Performance Caching," *2017 IEEE International Conference on Computer Design (ICCD)*, Boston, MA, 2017, pp. 345-352. doi: 10.1109/ICCD.2017.60

[4]. X. Zhang, C. Li, H. Wang and D. Wang, "A Cache Replacement Policy Using Adaptive Insertion and Re-reference Prediction," *2010 22nd International Symposium on Computer Architecture and High-Performance Computing*, Petropolis, 2010, pp. 95-102.
doi: 10.1109/SBAC-PAD.2010.21

[5]. M. Bakhshalipour, P. Lotfi-Kamran and H. Sarbazi-Azad, "Domino Temporal Data Prefetcher," *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Vienna, 2018, pp. 131-142. doi: 10.1109/HPCA.2018.00021

[6]. T. Zheng, H. Zhu and M. Erez, "SIPT: Speculatively Indexed, Physically Tagged Caches," *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Vienna, 2018, pp. 118-130. doi: 10.1109/HPCA.2018.00020

[7]. N. El-Sayed, A. Mukkara, P. Tsai, H. Kasture, X. Ma and D. Sanchez, "KPart: A Hybrid Cache Partitioning-Sharing Technique for Commodity Multicores," *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Vienna, 2018, pp. 104-117. doi: 10.1109/HPCA.2018.00019

[8]. W. Stallings, "*Cache Memory*," in Computer Organization and Architecture, 8th ed., Upper Saddle River, NJ, USA: Pearson Prentice Hall, 2006, pp.117

[9]. Nicholas Carter, "*Cache Memory Organization*," in Computer Architecture and Organisation: Schaum's Outlines Series, 2nd ed., McGraw-Hill Publications

[10]. Carl Hamache, "*Cache Memory Organization*," in Computer Organization, 5th ed., McGraw-Hill Publications