Cache Memory: An Analysis on Replacement Algorithms and Optimization Techniques

Qaisar Javaid, Ayesha Zafar, Muhammad Awais, Munam Ali Shah

Cache Memory: An Analysis on Replacement Algorithms and Optimization Techniques

QAISAR JAVAID*, AYESHA ZAFAR**, MUHAMMAD AWAIS**, AND MUNAM ALI SHAH**

RECEIVED ON 18.07.2016 ACCEPTED ON 16.08.2016

ABSTRACT

Caching strategies can improve the overall performance of a system by allowing the fast processor and slow memory to at a same pace. One important factor in caching is the replacement policy. Advancement in technology results in evolution of a huge number of techniques and algorithms implemented to improve cache performance. In this paper, analysis is done on different cache optimization techniques as well as replacement algorithms. Furthermore this paper presents a comprehensive statistical comparison of cache optimization techniques.To the best of our knowledge there is no numerical measure which can tell us the rating of specific cache optimization technique. We tried to come up with such a numerical figure. By statistical comparison we find out which technique is more consistent among all others. For said purpose we calculated mean and CV (Coefficient of Variation). CV tells us about which technique is more consistent. Comparative analysis of different techniques shows that victim cache has more consistent technique among all.

Key Words: Recency, Cache Hti, Cache Miss, Miss Latency, Threshold, Optimization.

1. INTRODUCTION

Cache is a high speed memory which is not as costly as registers but it is faster when compared to main memory. Cache memory is basically used to store data and information which are currently being used. To access data from main memory takes more time, therefore to reduce this time a special memory inside the CPU (Central Processing Unit) is reserved to keep small amount of data for some time. CPU having cache memory needs less time to wait for an instruction to be fetched from the memory for processing. Absence of cache memory decreases execution rate which affect the performance of CPU.

The main purpose of cache memory is to reduce the speed gap between slow memory and fast processor at a reduced cost [1]. It mostly consists of most recently accessed piece of main memory.All information is stored in some storage media like main memory.Whenever CPU/processor use some data or piece of information it is copied into some faster storage media like cache. when processor try to approach a particular piece of information again, the system checks it in cache first, if it is in cache processor use it from there if not found in cache it must be brought from main memory and copy it into cache assuming we will need it again.

Corresponding Author: (qaisar@iiu.edu.pk)
* Department of Computer Science & Software Engineering, International Islamic University, Islamabad.
** Department of Computer Science, COMSATS Institute of Information Technology, Islamabad.

Generally the cache memory is categorized in three L (Levels) i.e. L1, L2 and L3 cache [2]. L1 cache is actually the fastest or smallest than L2 and L3. Usually L1 cache resides in memory or it is also called on chip memory. L2 cache is faster than L3. Last level that is L3 is largest and slowest among all three levels. Inmulti-core processor severy processor has its own L1 and L2 cache. L3 cache is shared among all processors [3]. This hierarchy of cache levels is shown in Fig. 1.

Whenever desired chunk of information whether it is data or instruction is present in cache this situation is called cache hitand time taken to find out whether it is present in cache or not is called hit latency [4].If required data is not found in cache then it would be brought into the cache from main memory this situation is called cache miss [5]. Mainly, three type of cache misses exist: (i) compulsory misses which take place when a memory location is accessed for the first time, (ii) conflict misses which occur due to insufficient space when two blocks are mapped on the same location (iii) capacity misses takes place due to small space [6].

Due to small size of Cache Memory it is needed that its content has to be replaced according to the usage and

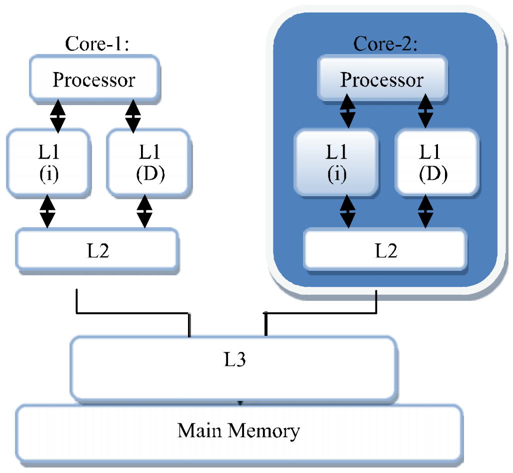

FIG. 1. SHOWING CACHE HIERARCHY

specific time period. An important component of cache is its replacement policy which is a decision about which page or data is replaced from cache to make space in cache for new data. Here the problem is that what piece of data is going to be replaced. Different history based prediction algorithm were developed and implemented these algorithms map pages according to their suitability for eviction. Although they have some drawbacks but in order to achieve better performance they are used.

There exists a fair distance between the processor speed and the memory access latency, so a great effort has been put in this regard to reduce this gap. There is lot of work done to overcome this gap that is hardware based, compiler based and operating system based.

This paper discusses several basic and advance cache optimization techniques and replacement policies proposed till date. Rest of the paper is categorized is as follows: Section 2 is based on cache replacement policies, Section 3 discusses cache optimization techniques, Section 4 is about performance evaluation ,Section 5 is discussion and Paper is concluded in section 6.

2. REPLACEMENT ALGORITHMS

Replacement algorithms/policies are used in order to attain optimized usage of cache. When cache is full, then replacement policies decide which piece of data is replaced in order to make space for new data that is currently being used. An efficient algorithm is that which can take less time and number of cache misses are low and also balancing cost. Following are some of the algorithms.

LRU (Least Recently Used) Algorithm): This algorithm discards the least recently used item from the cache in order to make space for the new data item. In order to achieve this, history of all data items that is which data item is used when, is kept. A variable known as Aging Bit is used to store this information, Although this algorithm provides better performance but cost of implementation is much more [6]. Variants of LRU are the most popular among all other algorithms. Key advantage of this policy is its simple implementation, time and space overhead is constant. “Recency” is a main factor in this algorithm while LRU takes into account the characteristics of”recency” of the workload; it ignores and exploits the capabilities of “frequency” of a workload [7].

HLFU (History Least Recently Used) Algorithm): Cache replacement in typical LFU algorithm is performed by replacing the least frequently requested objects. But in HLFU which is an extension of LFU considers the History function in cache replacement process. Respective LFU threshold which is a linear function on the amount of currently used cache. The HLFU algorithm will replace the cached objects based on Hist value as compared to the defined threshold in LFU.

LFU (Least Frequently Used) Algorithm: This algorithm counts how often data items have been used. The data items which are used less are deleted from the cache first. If all objects have same frequency then this algorithm randomly discards any data item [8].

GDS (Greedy Dual Size): In this algorithm index is calculated according to the size of a file. Larger the file smaller is the index. File with the smallest index is replaced in this algorithm. Inflation value is used to keep track of frequently accessed files in the cache [9].

CAR (Clock with Adaptive Replacement) Algorithm: CAR is simple to implement and it has very low overhead on cache hits. It shows high performance and it also provides service of self-tuning. It is scan resistant which result in low space overheads that are less than 1% [10]. CAR does not care for certain workloads [11].

ARC (Adaptive Replacement Cache): This algorithm is easy to implement running time is not dependent on cache size. ARC has a low space overhead of approximately 0.75% of the size of the cache. ARC is a scan resistant also leads to self-tuning. This algorithm continuously balanced recency and frequency features by responding to changing access pattern [12]. In this algorithm cache is divide into two queues, each is handle by using CLOCK or LRUthat contains pages accessed only once, while the other contains the page which are accessed more than one time [13]. Like other algorithms ARC also has a constant complexity per request. “Ghost cache” is a special term used in this algorithm to handle the data element which will be used in near future [14-15].

RR (Random Replacement) Algorithm: This algorithm randomly selects any of the data item from the cache and replace it with the desire one [4]. This algorithm does not need to keep track of the history of the data contents and it does not need any data structure. Due to which it consumes less resources, therefore its cost is less as compare to other algorithms [16].

SLRU (Segmented LRU) Algorithm: This algorithm partitions the cache into two portions, one is unprotected and the other is protected. Protected portion is reserved for mostly used objects. When first request for an object has been done then this object is inserted into the unprotected portion. On a cache hit the object is moved into the protected portion [17]. Both portions are managed by LRU technique. But content from unprotected part has been removed and content from protected part has been moved back to the unprotected part a recently used content. This method requires a variable that calculates what percentage of the cache space is reserved for protected part [18].

LR+5LF Algorithm: LR+5FU replacement policy is a combination of two popular replacement policies i.e. LRU and LFU. The problems arrived in LRU and LFU policies are solved by new policy called LR+5FU [6]. The weighing problem of LRU AND LFU is solved by this algorithm. LR+5LF policy reduces cache miss with greater amount than LRU, FIFO, and LFU at L1 and L2 cache [19].

FIFO Algorithm: The first in first out algorithm removes the page that has not been used for a long time. It treats the pages as a circular buffer, and pages are removed in a round robin fashion. It cause early page fault [20].

LLF (Lowest Latency First): This algorithm keeps the average latency to a minimum by first expelling the object with the lowest download latency [7].This algorithm gives the best result in cases where the data is retrieved by executing a query against a relational database.

Algorithms that are discussed above are classified in several classes. These classes are made in term of different parameters discussed in [9-2]. In all parametersrecency and frequency are the most important factors. This calcification is also used by other authors.Classification of classes is:

(i) Recency Based Algorithms

(ii) frequency Based Algorithms

(iii) Recency/Frequency Based Algorithms

(iv) Function based algorithms

(v) Randomized algorithms

This classification described in **Fig. 2** and comparisons between different algorithms are shown in **Table 1**.

3. OPTIMIZATION TECHNIQUES

Cache optimization techniques are classified on the basis of their application and comparison is made in their own domain to ensure homogeneity prevails.Cache optimization is achieved by reducing hit time, reducing miss penalty, increasing cache bandwidth and by reducing miss rate. Above said task is achieved by different techniques.

One way to minimize the gap between memory latency and CPU cycle is the use of a multilevel cache. Miss rate

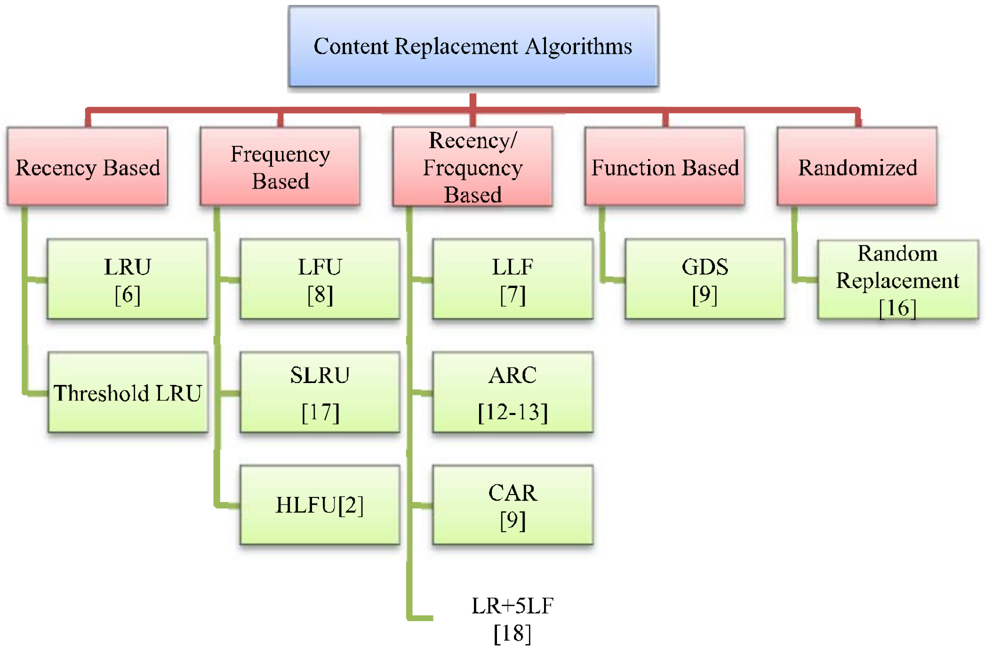

FIG. 2. SHOWING CLASSIFICATION OF REPLACEMENT ALGORITHMS

of L1 (First Level) cache can be reduced by introducing L2 cache. Anothertechnique called ULCC (User Level Cache Control) [21] has been implemented through which user can control space allocation in cache. This technique is hard to implement but it produces less hit rate and also reduce cache pollution [22].

One of the other ways of cache optimization is compiler based optimization in which loops are optimizing through compiler. To set the accessed data in cache loops must be reduced to smaller size. In this way all the tasks will be executed consecutively which will be using same data from cache [23].

Different methods are used to make cache's performance better. One of them is jigsaw [24]. It is used to solve scalability and interference issues. It helps to define how data would be mapped to shares. Every share has a unique id. Jigsaw produces better performance than NUCA design [25]. In NUCA (Non Uniform Cache Access) cache

TABLE 1. COMPARISON OF REPLACEMENT ALGORITHMS

Performance Parameters	Algorithms										
	LRU [7]	HLFU [2]	LFU [8]	LLF [7]	CAR [10]	ARC [15]	FIFO [20]	GDS [9]	RR [16]	SLRU [17]	LR+5LF [19]
Cache Miss	Low cache miss rate		Minimum miss latency		Miss rate decreases	Low miss latency		high	Higher miss rate	293k	Reduces cache misses
Cache Hit	4.24% on 16mb cache and 9.19% on 4GB cache [11]	Low hit ratio as compare to LFU	1077k [10]		42.42%	61.87% on 512 byte size pages	33.8%	Minimum hit rate	Low cache hit rate.	1553k	high
Resource Utilization	Need to track down recent list update overhead	Extra storage is require to keep history	Need to track down frequently accessed data.	Minimum resources utilize	High resource utilization	Maximum resource utilization		Minimum resource utilization because block with	Very small amount of resources required	Cost of segmentation needs high resource utilization	High because it track down both recent list and frequency.
Queuing	Complexity fluctuate b/w constant and logarithmic values	Single queue is used	Complexity fluctuate b/w constant and logarithmic values	Single queue is used	Over head of two circular buffers	Partition the cache into two queues [11]	Single queue is used	Single queue used	No queue use. randomly selected	Both single and circular queues are implemented	Same as LFU and LRU.
Size of Physical Memory	Complexity increase when size increase		Logarithmic implementation on in cache size	With the increase in size latency also increase	Constant complex as size increase		Ignore usage pattern and paging overhead due to large size.	Size of cache doesn't matter	Doesn't matter because content has to be replaced randomly.	Cause complexity because different segments has to be managed accordingly	Large memory size
High	Almost low		Low		Very low	Low	Low		High	Very low	Low
Recency/ Frequency	It doesn't capture frequency	It capture frequency	Pay no attention to recency		Capture both recency and frequency	Balanced recency and frequency work.	No need to have a recency or frequency parameter		No need recency/ frequency parameter	Keep track of frequency of cache elements	Uses both recency and frequency
Extra Parameter	Needs to pre-tune parameter [13]	Needs a parameter to controlling references	Needs a tune able parameter to controlling references					Cost parameter needed	Relative cost calculation parameters needed	Maintenance of different segments needed extra parameter	Needs both tune able and pre-tune parameter

is managed in such a way that most of the data is served by fastest bank. To more data to faster banks in steps, a switched network is used. The core feature of NUCA design is the low latency access.

A set of compiler algorithms have been written for the prediction about the data to be reused in near future. These predictions are used to make the hit rates better. The algorithm used for the purpose is evict-me which uses cache line tag of one bit. So when ever evict-me tag is set for any cache line, the cache line will be replaced [26] but this technique is complex and consume high energy.

Different methods are used to implement two way set associative cache. One of them is predictive sequential associative cache. Using this method to implement set associative cache access time becomes approximately equal to direct mapped cache [27]. This method uses different prediction which helps to reduces access time and hit rate.

One way to improve the performance of the cache is to produce the next data to be used by the cache, data perfecting is used to produce data in advance which is next to be used [28].

Enlarging the size of cache reduce the chance to occur capacity misses. Simple cache and way prediction methods are used to reduce cache hit time.Comparison of different techniques is shown in Table 3.

4. PERFORMANCE EVALUATION

To date researcher came up with many cache optimization techniques. People judged them by actually using them. To the best of our knowledge there is no numerical measure which can tell us about rating of specific cache optimization techniques. We tried to come up with such a numerical figure.

Table 2 shows collected performance parameters against cache optimization techniques. Their value are given as H (High), M (Medium), L (Low) and N (Not Applicable). In Table 3 numerical values are assigned against, H, M and L .Table 4 is numerical replacement of Table 2 which is explained by Table 3.

In Table 4 if the particular parameter decreases the performance then values are assigned as 1, 2, and 3 for H, M and L, respectively, if performance increases values are assigned as 3, 2, and 1 for H, M and L, respectively. The last two column of Table 4 shows Mean and CV which tells us about more consistent technique for cache optimization.

CV provides information about data i.e. how much data is scattered from its mean. As low as the value of coefficient is, technique is more consistent. It is calculated in last column of Table 4.These tables give us a better view to judge the subject techniques. Fig. 3 shows the graphical representation of Table 4.

5. DISCUSSIONS

In section 1 different cache replacement policies which have been implemented in the past have been studied and compared with each other. Policies are compared on the basis of following parameters, Cache miss, cache hit, and Resource utilization, Queuing, Size of physical memory, Page movement overhead, Recency/frequency and extra parameter.

We also studied different optimization techniques and compared them in Table 2. Table 2 is little bit modified form of table presented in [6]. We use this table to perform statistical analysis on different techniques. By using statistical comparison we are able to find out that which technique is more consistent and reliable.

By comparing different techniques we analyze that cache miss rate is reduced by using larger block size, larger cache, and way prediction method. Larger block size reduces hit time, increases miss penalty and consume high power. Higher associativity produce fast access time but they have low cycle time. ULCC have fast access time and also used to eliminate cache pollution. Pipelined cache reduced miss penalty. Multilevel cache has very less miss penalty but yields high cycle time and have high power consumption.

TABLE 3. WEIGHTS ARE GIVEN TO PARAMETERS IN TABLE 4 AS FOLLOWING CRITERIA

	If Decrease Performance	If Increase Performance
High	1	3
Medium	2	2
Low	3	1
Not Applicable	0	0

TABLE 2. COMPARISON OF OPTIMIZATION TECHNIQUES ON THE BASIS OF MR, MP,HT, HR, PC, AT, COST AND COMPLEXITY, CT [6]

Techniques	Cache Comparison Performance Parameters								
	MR	MP	HT	HR	PC	AT	Cost	Complexity	CT
Larger Block Size [29-30]	Reduce compulsory misses and increase conflict misses	H	L	H	H	L	N	0	H
Higher ssociation [30]	Reduce conflict and capacity misses	L	H	L	H	H	H	1	H
Way rediction [28]	Conflict misses reduce	H	H	H	L	L	N	2	H
Column ssociative cache [31]	conflict misses reduce	N	H	L	H	H	H	2	H
Reactive associative cache [32]	reduce misses	L	H	H	N	L	H	1	L
Victim cache [33]	Cache misses reduce	L	M	H	L	M	H	1	H
Pipelined cache	N	L	H	L	N	H	H	2	L
Jigsaw [24]	Interference misses reduce	L	H	L	L	H	H	3	N
Larger cache [30]	Miss rate reduces	N	H	H	H	L	H	2	L
Cache miss lookaside [34]	Conflict misses reduce	L	H	L	H	H	H	2	H
Multilevel cache	Cache misses (M)	M	H	H	H	L	H	2	H
Predictive sequential associative cache [27]	As a 2 way associative cache	L	L	L	H	H	H	1	H
Small cache [1]	MRIncrease (H)	N	L	N	L	H	L	0	N
ULCC [21]	Cache pollution eliminate	N	H	L	N	H	H	2	N
Resizing and remapping [35]	N	N	N	N	L	N	N	2	N
MR is Miss Rate, MP is Miss Penality, HT is Hit Time, HR is Hit Rate, PC is Power Consumption, AT is Access Time, and CT is Cycle Time									

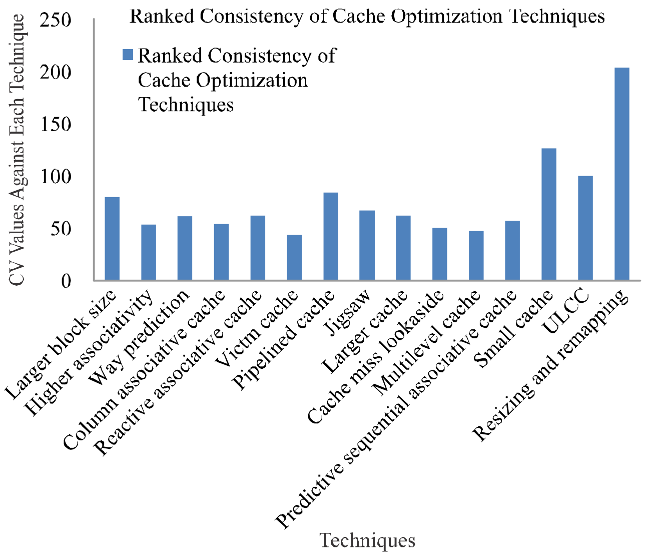

FIG. 3. STATISTICAL COMPARISON OF CACHE OPTIMIZATION TECHNIQUES GIVEN IN TABLE 4 USING COEFFICIENT OF VARIATION

TABLE 4. STATISTICAL COMPARISON OF CACHE OPTIMIZATION METHODS GIVEN IN TABLE 3

Techniques	Cache Performance Parameters Numerical Representation										
	MR	MP	HT	HR	PC	AT	Cost	Complexity	CT	Mean	Coefficient of Variation
Larger block size	2	1	3	3	1	3	0	0	1	1.55	79.45
Higher associativity	2	3	1	1	1	1	1	1	1	1.33	53.03
Way prediction	2	1	1	3	3	3	0	2	1	1.77	61.47
Column associative cache	2	0	1	1	1	1	1	2	1	1.11	54.08
Reactive associative cache	2	3	1	3	0	3	1	1	3	1.88	61.76
Victim cache	2	3	2	3	3	2	1	1	1	2	43.30
Pipelined cache	0	3	1	1	0	1	1	2	3	1.33	83.85
Jigsaw	2	3	1	1	3	1	1	3	0	1.66	67.08
Larger cache	2	0	1	3	1	3	1	2	3	1.77	61.74
Cache miss lookaside	2	3	1	1	1	1	1	2	1	1.44	50.29
Multilevel cache	2	2	1	1	1	3	1	2	1	1.55	46.87
Predictive sequential associative cache	2	3	3	1	1	1	1	1	1	1.55	56.69
Small cache	1	0	1	0	3	1	1	0	0	0.77	126.22
ULCC	3	0	1	1	0	1	1	2	0	1	00
Resizing and remapping	0	0	0	0	3	0	0	2	0	0.55	203.46

6. CONCLUSIONS

The cache is a critical part of performance there are many cache replacement policies and optimization techniques exist. We tried to provide comprehensive review on both proposed till date.

At the end we presented a comprehensive tabular representation of cache optimization techniques and their ranked numerical representation is made, which enable us to see which technique is more consistent. We compared total of 15 techniques. By computing values graph shows that victim cache has lowest coefficient of variation which shows that it is more consistent among all other techniques. Resizing and Remapping has highest coefficient of variation which means that it is less consistent technique among all compared techniques.

In future we would like to perform experiments on techniques like LRU+5LF, CAR, ARC and compare their results by homogeneously passing them through a large set of instructions.

ACKNOWLEDGMENTS

Authors would like to acknowledge with thanks the anonymous referees for their useful suggestions that led us to enhance the quality of the paper. Authors are also thankful to the International Islamic University, and COMSATS Institute of Information Technology, Islamabad, Pakistan, for providing platform to carry out this research.

REFERENCES

[1] Gagged, G., Paresh, R., and Madarkar, J., "Survey on Hardware Based Advanced Technique for Cache Optimization for RISC Based System Architecture", International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, No. 9, pp. 156-160, 2013.

[2] Patidar, K., "A Taxonomy of Cache Replacement Algorithms",International Journal of New Technologies in Science & Engineering, Volume 2, No. 3, pp. 98-108, 2015.

[3] Khatoon, H., Mirza, S.H., and Altaf, T., "Operating System-Aware Cache Optimization Techniques for Multi Core Processors", Proceedings of International Conference on Frontiers of Information Technology (FIT), pp. 99-105, 2011..

[4] Psounis, K., Prabhakar, B., and Science, C., "A Randomized Web-Cache Replacement Scheme", Proceedings of IEEE 20th Annual Joint Conference on Computer and Communications Societies, Volume 3, pp. 1407-1415, 2001.

[5] Kowarschik, M., and Wei, B.C., "An Overview of Cache Optimization Techniques and Cache-Aware Numerical Algorithms", Algorithms for Memory Hierarchies, Springer Berlin Heidelberg, pp.213-232, 2003.

[6] Ahmed, M.W., andShah, M.A., "Cache Memory: An Analysis on Optimization Techniques", International Journal of Computer and IT, Volume 4, No. 2, pp. 414-418, 2015.

[7] Butt, A.R., Gniady, C., and Hu, Y.C.,"The Performance Impact of Kernel Prefetching on Buffer Cache Replacement Algorithm", IEEE Transactions on Computers, Volume 56, No. 7, pp. 889–908, 2007.

[8] Swain, D., Paikaray, B., and Swain, D., "AWRP:Adaptive Weight Ranking Policy", arXiv Preprint arXiv, Volume 3, No. 2, pp. 1107,4851, 2011.

[9] Podlipnig, S., and Böszörmenyi, L., "A Survey of Web Cache Replacement Strategies",ACM Computing Surveys, Volume 35, No. 4, pp. 374-398, 2003.

[10] Chavan, A.S., Nayak, K.R., Vora, K.D., Purohit, M.D., and Chawan, P.M., "A Comparison of Page Replacement Algorithms", International Journal of Engineering and Technology, Volume3, No. 2, pp. 171-174, 2011.

[11] Bansal,S.,and Modha,D.S.,"CAR: Clock with Adaptive Replacement", Proceedings of 3rd USENIX Conference on File and Storage Technologies, Volume 4, pp. 187-200, 2004.

[12] Megiddo, N.,and Modha, D.S., "Outperforming LRU with an Adaptive Replacement Cache Algorithm", Computer, Volume 37, No. 4, pp. 58-65, 2004.

[13] Megiddo,N., Modha, D.S., and Jose, S., "A Simple Adaptive Cache Algorithm Outperforms LRU", IBM Research Report, Computer Science, 2003.

[14] Butt, A.R.,Gniady, C., and Hu, Y.C., "The Performance Impact of Kernel Prefetching on Buffer Cache Replacement Algorithms", ACM SIGMETRICS Performance Evaluation Review, Volume 37, No. 1, pp. 157-168, 2005.

[15] Janapsatya, A., Ignjatoviæ, A., Peddersen, J., and Parameswaran, S., “Dueling CLOCK: Adaptive Cache Replacement Policy Based on The CLOCK Algorithm”, Proceedings of Conference on Design, Automation and Test in Europe, pp. 920-925, 2010.

[16] Bhattacharjee, A., and Debnath, B.K., “A New Web Cache Replacement Algorithm”, Proceedings of IEEE Conference on Pacific Rim Conference on Communications, Computers and Signal Processing, pp. 420-423, 2005.

[17] Gao, H., and Wilkerson, C., “A Dueling Segmented LRU Replacement Algorithm with Adaptive Bypassing”, 1st JILP Worshop on Computer Architecture Competitions: Cache Replacement Championship, 2010.

[18] Morales, K.,and Lee, B.K., “Fixed Segmented LRU Cache Replacement Scheme with Selective Caching”, IEEE Proceedings of 31st International Conference on Performance Computing and Communications, pp. 199-200, 2012.

[19] Abdel F.A., and Samra, A.A., “Least Recently Plus Five Least Frequently Replacement Policy (LR+5LF)”, International Arabic Journal of Information Technology, Volume 9, No. 1, pp. 16-21, 2012.

[20] Wang, Q., “WLRU CPU Cache Replacement Algorithm”, Doctoral Dissertation, The University of Western Ontario London, 2006.

[21] Ding, X., Wang, K., and Zhang, X, “ULCC: A User-Level Facility for Optimizing Shared Cache Performance on Multicores”, ACM Sigplan Notices, Volume 46, No. 8, pp. 103-112, 2011.

[22] Sandberg, A., Eklöv, D., and Hagersten, E., “Reducing Cache Pollution Through Detection and Elimination of Non-Temporal Memory Accesses”, Proceedings of IEEE Conference on High Performance Computing, Networking, Storage and Analysis, pp. 1-11, 2010.

[23] Ishizaka, K., Obata, M., and Kasahara, H, “Cache Optimization for Coarse Grain Task Parallel Processing Using Inter-Array Padding”, International Workshop on Languages and Compilers for Parallel Computing. Springer Berlin Heidelberg, pp. 64-76, 2003.

[24] Beckmann, N., and Sanchez,D, “Jigsaw: Scalable Software-Defined Caches”, Proceedings of IEEE 22nd International Conference on Parallel Architectures and Compilation Techniques, pp. 213–224, 2013.

[25] Kim, C., Burger, D., and Keckler, S.W., “An Adaptive, Non-Uniform Cache Structure for Wire-Delay Dominated On-Chip Caches”, ACM Sigplan Notices, Volume 37, No. 10, pp.211–222, 2002.

[26] Wang, Z., McKinley, K.S., Rosenberg, A.L., and Weems, C.C., “Using the Compiler to Improve Cache Replacement Decisions”, Proceedings of IEEE International Conference on Parallel Architectures and Compilation Techniques, pp. 199-208, 2002.

[27] Calder, B., Grunwald, D., and Emer, J.,”Predictive Sequential Associative Cache”, Proceedings of IEEE 2nd International Symposium on High-Performance Computer Architecture, pp. 244-253, 1996.

[28] VanderWiel, S., and Lilja, D.J., “A Survey of Data Prefetching Techniques”, Proceedings of the 23rd International Symposium on Computer Architecture, 1996.

[29] Huang, C.C., and Nagarajan, V., “Increasing Cache Capacity via Critical-Words-Only Cache”, Proceedings of IEEE 32nd International Conference on Computer Design (ICCD), pp. 125-132, 2014.

[30] Sawant, R., Ramaprasad, B.H., Govindwar, S., and Mothe, N., “Memory Hierarchies-Basic Design and Optimization Techniques Survey on Memory Hierarchies – Basic Design and Cache Optimization Techniques”, 2010.

[31] Agarwal, A., and Pudar, S.D., “Column-Associative Caches: Caches A Technique for Reducing the Miss Rate of Direct-Mapped”, ACM, Volume 21, No. 2, pp. 179-190, 1993.

[32] Batson, B., and Vijaykumar, T.N., “Reactive-Associative Caches”, Proceedings of IEEE International Conference on Parallel Architectures and Compilation Techniques, pp. 49-60, 2001.

[33] Stiliadis, D., and Varma, A., “Selective Victim Caching: A Method to Improve the Performance of Direct-Mapped Caches”, IEEE Transactions on Computers, Volume 46, No. 5, pp. 603–610, 1997.

[34] Bershad, B.N., Lee, D., Romer, T.H., and Chen, J.B., “Avoiding Conflict Misses Dynamically in Large Direct-Mapped Caches”, ACM SIGPLAN Notices, Volume 29, No. 11, pp. 158-170, 1994.

[35] Ramaswamy, S., and Yalamanchili, S., “Improving Cache Efficiency via Resizing + Remapping”, Proceedings of IEEE 25th International Conference on Computer Design (ICCD), pp. 47–54, 2007.