A New Method to Enhance LRU Page Replacement Algorithm Performance

Ali A. Titinchi, Nasser Halasa

Abstract : In modern operating systems, the essential goal of the main memory management is to allow multiprogramming. To achieve this goal, virtual memory and paging scheme are used. The virtual memory needs a powerful page replacement algorithm to decide which page to swap out from the memory when a page fault occurs and the main memory is full. Many algorithms are proposed over the years. These algorithms aim to incur minimum software and hardware overhead. One of the well-known replacement algorithms is the Least Recently Used algorithm (LRU). The LRU has been considered as a common and successful policy. However, it imposes significant time overhead when executed. This paper proposes a novel approximation to the LRU. The new approach will use the counter technique to implement the LRU, but this counter will be evaluated based on extended Not Recently Used (NRU) m-bits orientation. The proposed algorithm has been evaluated and compared to the LRU for different workloads. The evaluation shows promising and good results. On average, the paper's algorithm reduces the page fault rate by 13% and the execution time by 9.5%.

Index Terms: demand paging, LRU, memory management, NRU, page replacement algorithm, page fault, Virtual memory,

1 INTRODUCTION:

Memory management is the main part of operating system in which all the activities related to the memory are performed [1]. Nowadays, multiprograms are loaded and executed in the main memory at the same time. This leads to the concept of virtual memory. Virtual memory makes the system appears to have more memory than it actually has. That is done by sharing the main memory between the competing processes [2]. To deal with virtual memory, the operating system may apply the paging technique to partition the processes. In this technique, main memory is divided into frames where each frame is similar to the page's size [3]. Pages are brought into main memory only when the running process demands them, this is known as demand paging. However, demanding a page requires an efficient page replacement algorithm to decide which page to evict in case of a page fault and the memory is full. In fact, these replacement policies severely affect the performance of virtual memory [4]. So, the researchers always try to design a powerful page replacement algorithm by minimizing the page fault rate with the least overhead to the system. The Least Recently Used (LRU) is the most famous one. LRU performs nearly as well as an optimal policy, but the problem is that it imposes significant overhead and time response when executed in real time [5]. The huge overhead of the LRU is coming from the way it makes the decision. It keeps track when a page is used and replaces the page that has the longest period of time in the memory being unused [6]. Two methods have been suggested to implement the LRU.

- *Ali A. Titinchi, Electrical and Computer Eng. Dept. University of Nizwa/ Nizwa, Oman Email: ali.abdulhafidh@unizwa.edu.om*
- *Nasser Halasa, Computer Engineering Dept. Philadelphia University/ Amman, Jordan Email: nhalasa@philadelphia.edu.jo*

These are linked list or counter method. In both the above methods, the updating and the choice of a victim gage will cause a high cost of execution and slow response time [7]. Many researchers try to solve the hardware and software time cost of LRU algorithm. Sometime via proposing an approximated algorithm to LRU, and sometime via using dedicated hardware to accelerate the execution of the algorithm [2]. The remainder of the paper is organized as follows. Section 2 reviews some of the related works. In section 3 we describe the proposed algorithm. The system evaluation and results have been reported in section 4. Section 5 concludes and summarizes the paper.

2 LITERATURE REVIEW

Here, we will illustrate and focus on different approaches to achieve these goals. [8] suggests an algorithm called LRUL to solve the problem of bad performance of LRU with loop accesses pattern. It adds a parameter to monitor the last use distance (LUD), which is the distance between the current occurrence of a page to its last occurrence looking backward in the reference string from the current reference. The algorithm as mentioned in the paper, reduces the number of page faults but no attention given to the overhead of the algorithm when executed. In [9], a new cache replacement policy is proposed, namely Adaptive Replacement Cache (ARC). This algorithm makes an adaptation on LRU by maintaining two LRU lists of pages. These two lists will combine the LRU and LFU solutions and dynamically adjusts between them according to the workload type. The paper shows that even ARC outperforms LRU with respect to hit ratio, but still has overhead comparable to that of LRU. [10] proposes a novel replacement algorithm called SF-LRU (Second Chance-Frequency-Least Recently Used). The philosophy of the design is to modify the LRU by adding a parameter called RFCU (Recency-Frequency Control value). This algorithm provides a second chance to the block that may be deleted according to the LRU's rules. The proposal shows little improvement in miss ratio over LRU but it is more complicated. An algorithm named Clock algorithm is implemented in [11]. It arranges elements in a circle and captures the recency of each page in the memory with a bit called recently used bit. The value of the bit will be

used to evict a page from the memory or not. The clock algorithm is an example of an approximation to LRU. It shows a good fault rate and less CPU fault service time comparing to LRU. Authors in [12] develops a replacement algorithm using Re-reference Interval Prediction (RRIP). This algorithm is designed based on the NRU policy. However, NRU is an approximation to LRU using one bit (nru-bit) to sign each page in the memory. The primitive NRU relatively has a high page fault rate. So, RRIP modify the NRU by using M bits instead of one bit to sign every memory page. The paper shows that the more status can improve the performance of NRU drastically with less complexity and overhead to the system compared to LRU. In [13] a hybrid LRU algorithm is submitted to decrease overall page fault rate. The algorithm uses an extra feature which is the total number of references for each page. This number will be counted on each referred page. Also, a bit is used to indicate when the page is modified. The evaluation of H-LRU reports better hit ratio than LRU but with high overhead to the system. [14] proposes a replacement policy, called Low Inter-reference Recency Set (LIRS). LIRS effectively addresses the LRU limits with weak locality workloads, by using recency to evaluate ITER-Reference Recency (IRR) for making a replacement decision. IRR of a page refers to the number of other pages accessed between two consecutive references to that page. Now, in case of a miss, the page with high IRR and with highest recency is replaced. The authors show that LIRS outperforms LRU in most cases, but with additional space and execution time overhead. Hence, they plan to design an LIRS approximation with less overhead cost. [15] presents an improved CLOCK replacement policy, called CLOCK-Pro. This algorithm attacks weaknesses of LRU by changing the criteria of selecting pages for eviction. The idea here, is to approximate LIRS [14] using CLOCK algorithm. The CLOCK-Pro uses reuse distance which is defined as the number of distinct page accesses between current access and previous access of a page. The authors categorize the pages into cold and hot types. Hot pages are accessed frequently, while cold pages are not. Also, the algorithm uses a test period to track how old the page is in the memory. Three clock hands are used as indications for hot, cold and test. Measuring results show significant page fault reduction of CLOCK-Pro over CLOCK. Also, the execution times of some commonly programs can be reduced concretely. [16] proposes a new adaptive cache replacement policy, called Dueling CLOCK. The CLOCK algorithm is an approximation of LRU with low overhead cost and good performance when used for page replacement. But, it's major drawback is that it is not scan resistant. Hence, the authors suggest a scan resistant CLOCK algorithm, and a set dueling approach that dynamically chooses between CLOCK algorithm and the scan resistant version of the CLOCK algorithm. Researcher in [17] goes different way to improve the performance of LRU. This is done by using FPGA technique to implement the traditional LRU algorithm in hardware. The design shows an excellent improvement in speeding up the response of the LRU algorithm for the Hit/Miss cache requests. [18] presents an LRU algorithm based on fuzzy logic with reduced inference rules. In this proposal, the page's recency of reference is used as input crisp variable to calculate the fuzzy value. Then a knowledge base that composed of five inference rules are used to calculate the page replacement weights for the pages in the memory. These pages' weights are used to select the victim page for replacement when there is a page fault and no empty page frame available in the memory. Another work of interest is [19], in which a replacement algorithm named LRU-Time is derived to improve the performance of LRU in both the page fault rate and speed of replacement decision. The approach of the paper, is that while LRU evict the page that hasn't been used for the longest time, the LRU-Time will evict the page based on three factors. These factors are least frequently used, least recently used, and time limit to narrow the number of pages the algorithm searches through to find the victim page. The authors say that reducing the scope of searching will lead to increase up the speed rate of page eviction process. All the papers in the literatures review above focus on one of the following approaches:

1- Proposal to modify the LRU algorithm to improve the page fault rate.
2- Proposal of a new algorithm to work as an approximation of LRU algorithm to reduce the overhead cost.
3- Proposal of hardware implementation for LRU algorithm to speed up its execution in the real time environment.

Our proposal is to introduce a new page replacement algorithm that has better page fault rate and lower overhead cost compared to LRU. This improvement is done based on the use of a novel extended NRU policy to approximate the LRU as will be shown next section.

3 THE PROPOSED ALGORITHM

The strategy of LRU is that when a page fault occurs and no free frame is available, then replace the page that has been unused for the longest time. To fully implement LRU, there are two traditional methods. The first method is to maintain a linked list of all pages in the memory with the most recently used page at the front and the least recently used page at the rear. The second method is to maintain a large counter (64-bit) that is automatically incremented after each memory reference. Furthermore, each page in the memory must also have a field with the same size as the above counter. The field of the referenced page is always updated with the recent counter's value. When a page fault occurs, then searching is achieved to find the evicted page. This page is the one with the lowest counting number. It is known that the above two methods are very time consuming when they are programmed. Especially, if we know that the page replacement algorithm is a real time activity. Our proposal goes to overcome this problem by suggesting a novel approach based on the counter method explained above. In this section, the proposed algorithm will be explained in detail. The design of the algorithm starts by customizing a 64-bit counter to trace the number of memory references. The large size of the counter is necessary because of the huge (millions) memory references that may occur within the cycle life of program execution. Also, a field of 4 bits is assigned to each page in the memory. The novelty of the design comes from the way of looking at these fields. Each field will categorize its related page into a class of recency use. This classification will be used when

a page replacement is requested, this will be clear later. In fact, the algorithm will work on the classification of a page in two levels according to the counting value at the moment of the memory reference. Fig 1 shows the first level of classification. Here, four flags are allocated to the four partitions of the counter, where each partition is of 16 bits size.

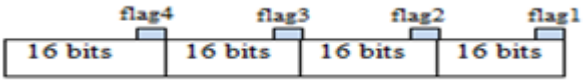

***Fig.1**. Counter partitions for first level of page classification*

Fig 2 below, illustrates the relation between these flags and the counting value that found in the counter.

counting value	flag 4	flag 3	flag 2	flag 1	class no.
$1<=count<2^{16}$	0	0	0	1	1
$2^{16}<=count<2^{32}$	0	0	1	1	2
$2^{32}<=count<2^{48}$	0	1	1	1	3
$2^{48}<=count<2^{64}$	1	1	1	1	4

***Fig. 2**. Relation between counter value, flags, and class no. in level 1*

In this level and as mentioned above, the algorithm will rank the pages in the memory in four classes from the oldest group of pages to the newest group of pages. This is done according to the flags set of each page. The proposed policy will look to the classes to compare between these groups of pages in the memory and select a victim page randomly from between the pages in the group of the lowest non-empty class. It is clear that this approach uses the NRU way of thinking to approximate the LRU method to replace a page. So, instead of depending on the exact time of page use, our extended NRU algorithm will make the decision of replacement and evict a page from a group of pages that are not used for long period of time. However, to improve the accuracy of the approximation of the algorithm, the pages will be farther classified in more specific classes in level 2. The idea behind the two level decision is that when a page is needed to be evicted on page fault, then at the first level the algorithm will choose the group of pages in the lowest numbered non-empty class. This chosen will lean on the flags values as explained before. If this group consists of one page, then this page is the victim one. But, if more than one page, are recognized in the lowest numbered class then the algorithm will take this group of pages to the second level of decision. For example, if class 1 is the lowest non-empty class then the mentioned part of the counter as shown in Fig 3 for all pages in class 1 will be more treated in the second level. The mentioned 16 bits for all pages in class 1 are divide into four parts each with 4 bits size. Again, on the second level, all pages that are nominated from the first level will be categorized into four classes. As in level 1, these classes are arranged according to the flags values as shown in Fig.4.

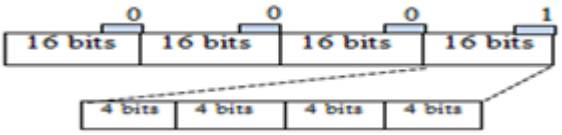

***Fig. 3**. Partitions of counter in the second level for more page classification*

***Fig. 4**. Relation between counter value, flags, and class no. in level 2*

In level 2, the algorithm will take the final decision by randomly choosing one victim page from the pages in the lowest non-empty class. As a summary, the pages will be classified into four classes at the first level. Then the group of pages that are transferred to the second level, again will be categorized into four classes. At the end, a page from the group found in the lowest non-empty class, will be selected randomly for replacement. However, if we merge the classification of the first level with the classification of the second level, then we can say that the pages are classified in [4x4=16] different classes. These 16 classes are distributed across the whole counting range from 1 to 2^{64} as shown in Fig 5. The goals of the proposed algorithm which is based on the extended multi-bits NRU method, is to get better page fault rate and reduce the execution time compared to the LRU. These improvements will be proved by experiments in the next section when the algorithm is evaluated. Before going to evaluate the algorithm, it is important to mention a practical feature that is considered in the design. This feature is proposed to support the goal of reducing the execution time to find the victim page. The idea is that if all pages in the memory become in class 3 of the first level (0111) at any moment, then the algorithm will reset the general counter and all the pages' fields that indicate their classes categories, then the counting of the recency of use for the pages starts again. Practically, this idea is very necessary and acceptable for the following two reasons:

1- When all pages in the main memory become in the same class of level 1 then they will be at the same scale of importance. Hence, all these pages should be moved to level 2 of the decision. This will increase the overhead of the time to find the victim page.

2- The above problem will be highly effective and more serious when all pages become in class 3 of level 1. This is because, the random choice of a victim page from between the pages in class 3 may result in a large margin of inaccuracy as shown for example in Fig (6). Here, if there are two pages and both are in the same class (class 3), then these pages should be moved to level 2 of decision. In level 2, any of them could randomly be selected as a victim page. In this case, if page 2 is selected then this will be a bad decision. That is because it is much better to select page 1 as it is very much older in use than page 2. This is clear

from their counters values as recorded from the general counter of the system at the moment of the last use for each page.

Overall Class No.	Binary representation Level1	level2	Counter value
0	00	00	$1 <= value < 2^4$
1	00	01	$2^4 <= value < 2^8$
2	00	10	$2^8 <= value < 2^{12}$
3	00	11	$2^{12} <= value < 2^{16}$
4	01	00	$2^{16} <= value < 2^{20}$
5	01	01	$2^{20} <= value < 2^{24}$
6	01	10	$2^{24} <= value < 2^{28}$
7	01	11	$2^{28} <= value < 2^{32}$
8	10	00	$2^{32} <= value < 2^{36}$
9	10	01	$2^{36} <= value < 2^{40}$
10	10	10	$2^{40} <= value < 2^{44}$
11	10	11	$2^{44} <= value < 2^{48}$
12	11	00	$2^{48} <= value < 2^{52}$
13	11	01	$2^{52} <= value < 2^{56}$
14	11	10	$2^{56} <= value < 2^{60}$
15	11	11	$2^{60} <= value < 2^{64}$

***Fig. 5**. Merge of level 1 and level 2 classes in one overall classes numbers*

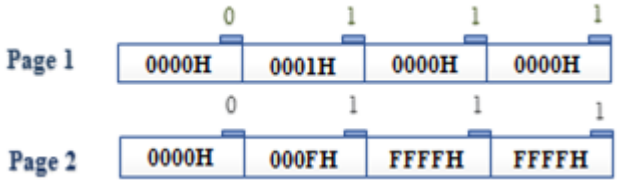

***Fig. 6.** Example to show why the algorithm resets the counter of use when all pages become in class 3 of level 1*

4 EVALUATION OF THE PROPOSED ALGORITHM

The objective of the paper is to design and implement a page replacement algorithm to overcome the LRU algorithm performance. Two factors are evaluated to test the performance of the proposed algorithm. These are the page fault rate and the overhead of time spent to decide the victim page when replacement needed. Three case studies with different page reference string type and different page frame numbers are taken for evaluation. The case studies are simulated by programs written using C++ language and executed on a machine using microprocessor running at 16 MHz. Case study (1): Sequential reference string [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20] of length 20 is imposed to LRU and the proposed algorithm, as page invoking sequence. Two experiments include 3-Frame and 4-Frame are considered and the results are reported in the bar diagram shown in Fig (7-a, b).

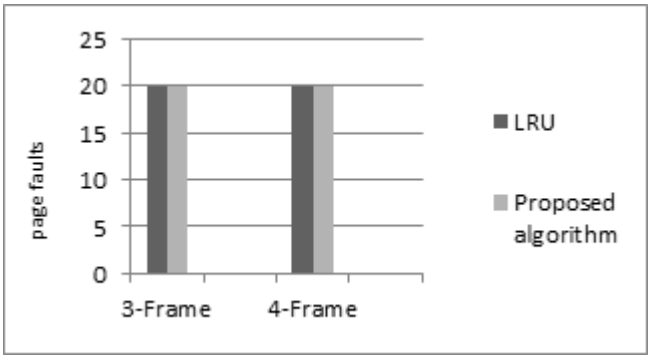

***Fig. 7-a**. Page fault rate evaluation for case 1*

Fig. 7-b. Execution time evaluation for case 1Case study (2): Random reference string [7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1] of length 20 is imposed to LRU and the proposed algorithm, as page invoking sequence. Two experiments include 3-Frame and 4-Frame are considered and the results are reported in the bar diagram shown in Fig (8-a, b).

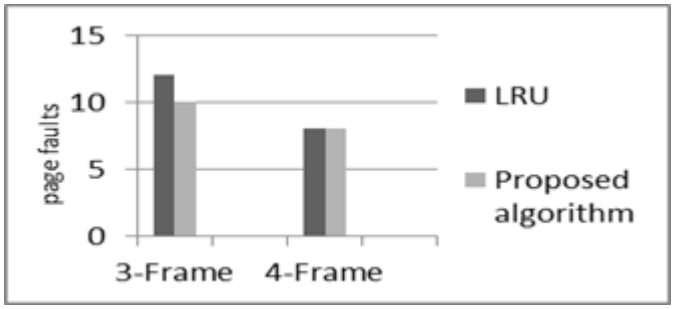

***Fig. 8-a**. Page fault rate evaluation for case 2*

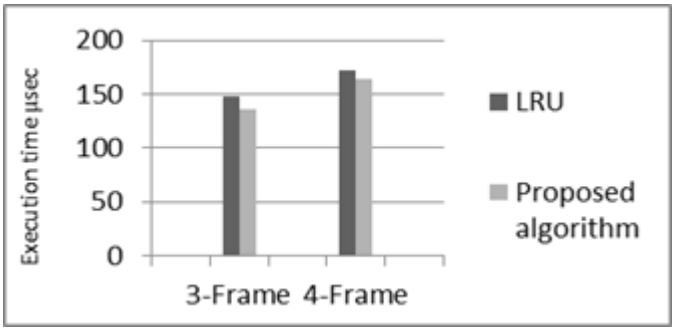

***Fig. 8-b**. Execution time evaluation for case 2*

Case study (3): Looping-like reference string [1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6, 1, 2,3, 4, 5, 6,1,2, 3, 4, 5, 6] of length 24 is imposed to LRU and the proposed algorithm, as page invoking sequence. Two experiments include 3-Frame and 4-Frame are considered and the results are reported in the bar diagram shown in Fig (9-a, b).

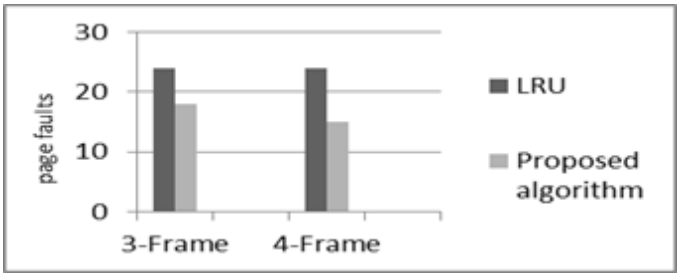

***Fig. 9-a**. Page fault rate evaluation for case 3*

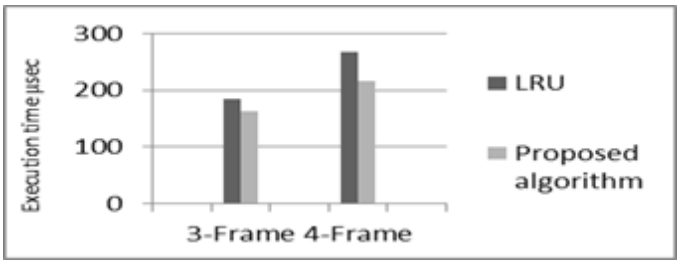

***Fig. 9-b**. Execution time evaluation for case 3*

Table1 shows a summary of improvement of the proposed algorithm comparing to LRU for the three case studies. The improvement is evaluated for both the page fault rate and time of execution. As shown, the calculated average of improvement for all cases, for page fault rate is about 13% and for time needed for execution is about 9.5%. The results show that the paper's algorithm achieves good average enhancement comparing to the LRU algorithm.

***Table 1**.*
Summary results of the evaluation tests

Case no.	No. of frames	Page fault rate improvement	Execution time improvement
1	3	0%	5%
	4	0%	7%
2	3	17%	9%
	4	0%	5%
3	3	25%	11%
	4	37%	20%

5 CONCLUSION

In this paper, a new page replacement algorithm is proposed and implemented. The core of the design is based on a novel idea to extend the NRU policy. The goal of the research's algorithm is to improve the performance of the LRU algorithm. For this purpose, two parameters are tested and evaluated, these are the page fault rate and the execution time needed to make the decisions of pages' replacements. Investigating the results shown in the comparing bar charts and the summary table, clarifies that in average, the proposed algorithm outperforms the LRU by 13% for the page fault rate and by 9.5% for the execution time. This analysis shows promising results that give the incentive to work further on the proposal to enhance the performance more. Moreover, and because of the real time nature of the page replacement algorithm, we plan in the future to reduce the execution time as much as possible. This will be done by suggesting a hardware model based on FPGA circuits to implement the algorithm.

6 REFERENCES

[1] Muhammad Waqar, Anas Bilal, Ambreen Malik, and Imran Anwar,"Comparative Analysis of Replacement Algorithms Techniques Regarding to Technical Aspects", European Journal of Engineering and Technology Vol. 4 No. 5, 2016.

[2] S.M. Shamsheer Daula, Dr. K.E Sreenivasa Murthy, G Amjad Khan," A Throughput Analysis on Page Replacement Algorithms in Cache Memory Management", International Journal of Engineering Research and Applications (IJERA), Vol. 2, Issue 2, pp.126-130, Mar-Apr 2012.

[3] Farhad Soleimanian Gharehchopogh, Awat Maroufi, Isa Maleki," AFRP: A New Approach in Page Replacement Algorithm with Hybird Aging and Mamdani Fuzzy Interface Algorithms", International Journal of Academic Research, Vol. 6, No. 1,. January 2014.

[4] Anvita Saxena," A Study of Page Replacement Algorithms", International Journal of Engineering Research and General Science Volume 2, Issue 4, June-July, 2014.

[5] Mohd Zeeshan Farooqui, Mohd Shoaib, Mohammad Zunnun Khan," A Comprehensive Survey of Page Replacement Algorithms", International Journal of Advanced Research in Computer Engineering & Technology (IJARCET),Volume 3 Issue 1, January 2014.

[6] Mahesh Kumar M R, Renuka Rajendra B," An Input Enhancement Technique to Maximize the Performance of Page Replacement Algorithms", International Journal of Research in Engineering and Technology, Volume: 04, Issue: 06, June 2015.

[7] Andrew S. Tanenbaum, Herbert Bos," Modern Operating Systems", Fourth edition,pp. 213-214, 2015.

[8] Ruchin Gupta, Narendra Teotia, "Least Recently Used Page Replacement using Last Use Distance (LRUL)", International Journal of Computer Applications, Volume 84 – No 2, December 2013.

[9] N. Megiddo and D. S. Modha, "ARC: A Self-Tunning, Low Overhead Replacement Cache,"Proc.Usenix Conf. File and Storage Technologies (FAST 2003),pp.115-130, 2nd Usenix, 2003.

[10] Jaafar Alghazo, Adil Akaaboune, Nazeih Botros, "SF-LRU Cache Replacement Algorithm", Records of the 2004 International Workshop on Memory Technology, Design and Testing (MTDT'04), IEEE, 2004.

[11] F. J. Corbato," A Paging Experiment with the Multics System", Project MAC (Massachusetts Institute of Technology), Defense Technical Information Center, 1968.

[12] Aamer Jaleel, Kevin, B. Theobald, Simon C.,

Steely Jr., Joel Emer, " High Performance Cache Replacement Using Re-Reference Interval Prediction (RRIP)", Proceedings of the 37th annual international symposium on Computer architecture (ISCA'10) , June 2010.

[13] Pooja Khulbe, Shruti Pant," Hybrid (LRU) Page-Replacement Algorithm", International Journal of Computer Applications, Volume 91 – No.16, April 2014.

[14] Song Jiang, Xiaodong Zhang," LIRS: An Efficient Low Interreference Recency Set", Proceedings of the 2002 ACM SIGMENTRICS, Internatinal conference on Measurement and Modeling of computersystems, pp. 31-42, June 2002.

[15] Song Jiang,Feng Chen, Xiaodong Zhang," CLOCK-Pro: An Effective Improvement of the CLOCK Replacement", USENIX Annual Technical Conference, 2005.

[16] Andhi Janapsatya, Aleksandar Ignjatovi´c, Jorgen Peddersen, Sri Parameswaran," Dueling CLOCK: Adaptive Cache Replacement Policy Based on The CLOCK Algorithm",Conference on Design Automationand Test in Europe, Europe Design and Automation Associatio ,March 2010.

[17] Nasser Halasa,, Ali A. Titinchi," An Efficient FPGA Circuit Design for speeding up Cache LRU Replacement Algorithm", 13th International Multi-Conference on Systems, Signals & Devices IEEE , pp. 527-530, 2016.

[18] Ali A. Titinchi, Nasser Halasa," FPGA implementation of simplified Fuzzy LRU replacement algorithm", 16th International Multi-Conference on Systems, Signals & Devices (SSD'19) IEEE,pp. 669-674, March 2019.

[19] Nabeel Zanoon , Evon Abu-Taieh,Hatem Salem Abu-Hamatta ,"A Novel Approach to Improve LRU Page Replacement Algorithm", Journal of Engineering and Applied Sciences, Volume: 13,| Issue: 2 , pp. 478-483, 2018.