Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

Reclaim Least Recently Used Replacement Policy for Level 2 Cache in Multicore Systems

Dr. Dhammpal Ramtake
Govt. Navin College Hasoud, Dist. Sakti, Chhattisgarh, India

Dr. Sanjay Kumar
SoS in Computer Science & IT, Pt. Ravishankar Shkula University, Raipur, Chhattisgarh, India

Abstract: - Multi-core processors reflect an evolutionary change in modern computing and set the new high-performance computing (HPC) standard. The efficiency of the cache memory is an important factor for evaluating overall processor performance. In a multicore processor, concurrent processes resides in main memory uses a shared cache. The shared cache memory reduces the access time, bus overhead, delay and improves processor utilization. Cache replacement policies for L2 shared cache can be improve the performance of multicore systems efficiently. LRU cache replacement policy gives better reflect the locality of reference and is widely used. LRU is not optimal for reducing the shared cache miss ratio and it is also failed to predict frequently used data. In this paper we implement a newly proposed a reclaim oldest block cache replacement policy (RLLRU) and compared with existing cache replacement policies such as LRU, FIFO, PLRU, LFU and Random. Results are obtained on various parameters such as hit ratio, average access time, execution time etc. The experiment is performed on Gem5 Simulation tool with SPLASH 3.0 FFT and Radix benchmarks.

Keywords: - ***Cache replacement policies, Execution time, Hit ratio, LRU, RLLRU.***

1 INTRODUCTION

Computer pioneer correctly predicate that computer professionals and researchers would want unlimited amount of fast memory. An economical solution to that desire is a memory hierarchy, which takes advantage of locality and cost performance of memory technologies. The development of processor had changed from increasing the working frequency to putting more cores into a single chip. As a result, the memory requests of processor are growing rapidly. By disparity, the growth of main memory is growing slower than the development of processor and it could not satisfy the demand of processor [1]. Multicore processors have become a mainstream design choice for modern high performance microprocessors with excellent performance [22]. Therefore the cache memories are introduced, which play an important role in improving performance of processors and reducing memory request.

In multicore processors, two or more independent cores are combined into a single processing chip. In most of the cases, each processor has its own private level-1 cache memory (L1). Generally, the L1 cache memory is split into instruction cache and data cache. Also, multicore processors may have one shared level-2 (L2) cache or multiple distributed and dedicated L2s cache [2][19]. The clock of the processor is several hundred times faster than the access latency of main memory. Cache provides the service to reduce this gap and make the performance of system better [5][20].

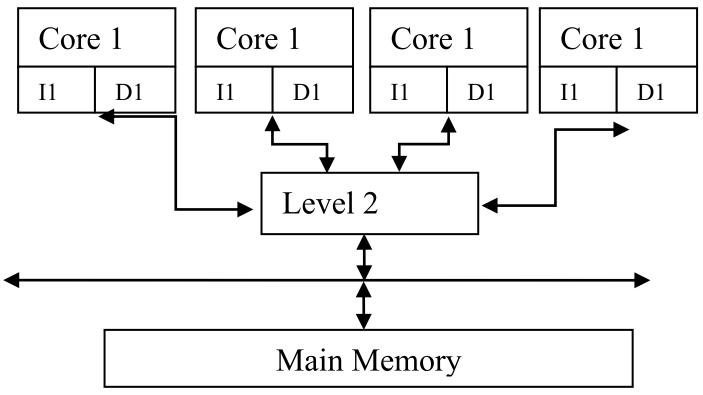

Fig.: 1.1 Multicore CPU with 2 Levels

The major dimensions of cache design are organization or structure of caches, set of rules that is followed when the cache is accessed and cache address space. Apart from these techniques compiler optimization for reordering the address stream and principle of locality of reference are also considered for designing of cache. In this paper we study and analysis the impact of level 2 cache replacement policies in multicore processors. Level 2 cache memory is shared among all the cores multicore processors. Therefore, it is very changing issue. In Least recently Used (LRU) replacement policy every cache line or block are searched by processor to find oldest one. The LRU is efficient, still it has some disadvantages such as LRU replacement policy wastes valuable high speed cache memory. Each time when a cache hit occurs, the cache controller must put a time counter value in memory location associated with the cache memory line. First In First Out (FIFO) replacement policy firstly added line or block is searched.

2. CACHE REPLACEMENT POLICIES

Once the cache becomes fail or miss occurs or cache is full then a new block or line is brought into the cache. Whenever, a new block or line is brought into the cache, then the existing block or line must be replaced. For direct mapping, there is only one line for any particular block. For associative mapping and set associative mapping, a replacement algorithm is needed. In these caches, the general goal of replacement policies is to minimize future cache misses by evicting a line that will not be referenced often in the future [6]. Cache replacement policies determine which data blocks should be removed from the cache when a new data block is arrived. There are various cache replacement policies such as FIFO (First In First Out), LRU (Least Recently Used), Random etc. FIFO selects for replacement of the block least recently loaded into cache. In FIFO there is a circular counter. When miss occurs then circular counter is incremented by one [7][20]. FIFO is very simple to implement. LRU policy selects for replacement of the block that was least recently accessed by the processor and replaces that block. This policy is based on the assumption that the least recently used block is the one, least likely to be referenced in the future. The LRU is efficient, still it has some disadvantages such as LRU replacement policy wastes valuable high speed cache memory. Each time when a cache hit occurs, the cache controller must put a time counter value in memory location associated with the cache memory line. Another disadvantage with the LRU replacement policy is that it requires complex logic for implementation. When a replacement occurs, the cache controller compares all the line time counter values of all line of cache memory [8][17]. To reduce the cost and complexity of the LRU policy Random policy can be used, but potentially at the expense of performance. A random replacement policy would select a block to replace in a totally random order, with no regard to memory references or previous selections. Least Frequently Policy (LFU) counts how often cache blocks or lines have been used. The blocks or lines which are used less are removed from the cache first. If all elements have same frequency then this policy removes any blocks or lines from cache [1][18][21].

Pseudo-LRU (PLRU) is a tree-based approximation of the LRU policy. In the tree-based replacement policy (number of ways -1) bits are used to track the accesses to the cache blocks or lines, where number of ways represents the number of cache blocks or lines in a set [7].

3. RECLAIM LEAST RECENTLY USED CACHE REPLACEMENT POLICY

In Least Recently Used policy access frequency of data is not predicted. LRU replacement policy considers only the access information of the most recently used data block or lines. In Reclaim LRU policy cache blocks or lines are assigned bits that hold reclaiming value. This reclaim value define that the lines or blocks are reused in future. This policy search only those line or block which does not have any reclaim value. Therefore, proposed policy take optimum time to find line block where new data come from upper level of memory.

RLLRU resolves searching deficiency and reusability of the LRU by introducing the concept of non reclaimed blocks. RLLRU has the following advantages:

(1) RLRU can easily find location for eviction where new data block.

(2) RLLRU also used to reference line so that without delay data will be accessed.

(3) RLLRU takes less time to search the line or block.

3.1 RLLRU Model

In a cache replacement policy selecting most appropriate line or block of data to evict is complicated task. For this we propose RLLRU reclaimed valued based cache replacement policy. RLLRU depends on pre prediction bits value. The Reclaim sets are defined as follows:

0	***0***	***Oldest Line (OL)***
0	***1***	***Less Least Line (LLL)***
1	***0***	***Newly Arrived Line (NAL)***
1	***1***	***Referenced Line (RL)***

In RLRLU policy first OL bits are used to identify the oldest line of cache memory and LLL is used to recognize that line is less frequently accessed through processor. When new data is inserted in cache line than bit value of NAL is assigned and RL bits are used to identify the line which will be accessed in future. Here 8 lines of cache are presented. Where A, B, C, D, E, F, G, H data elements are stored and corresponding reclaim bits are shown.

A	B	C	D	E	F	G	H
00	00	00	01	01	01	01	10

Fig.: 1.2 cache line with reclaim

Suppose X new data element is fetched from upper level of memory. Than first cache logic will search the oldest lines of cache then insert new data element into that any of oldest line. After that next less least line become oldest line. In fig 1.3, line where D is stored reclaim bits value is changed from 01 to 00.

X	B	C	D	E	F	G	H
10	00	00	**00**	01	01	01	10

Fig.: 1.3 cache lines with OL

Now, next element G is accessed. Therefore, G line will be referenced line and reclaim value of this line become 01 to 11 is shown in fig.:1.4.

X	B	C	D	E	F	G	H
10	00	00	**00**	01	01	**11**	10

Fig.: 1.4 cache lines with RL

Now again element H is accessed. Therefore, H line will be referenced line and reclaim value of this line become 10 to 11.

X	B	C	D	E	F	G	H
10	00	00	**00**	01	01	**11**	**11**

Fig.: 1.5 cache lines with RL

Now again Y new data block is fetched from upper level of memory than it is placed on oldest line (which is firstly inserted). Therefore, next less oldest line become oldest and bits value is changed from 01 to 00.

X	Y	C	D	E	F	G	H
10	10	00	**00**	**00**	01	**11**	**11**

Fig.: 1.6 cache lines with RL

3.2 RLLRU Algorithm

```
Step1: Initially set reclaim bits 00 to all the cache line.
Step2: For every insertion line should be changed 00 to 01.
Step3: If cache memory become full or element is not
       found in cache then call the search procedure.
  Loop until all oldest lined which marked as 00.
       first of oldest line = new data element;
       Reclaim bit value = 10
  End of loop
  Next of this the bit value of less oldest line =00
    End of If
Step4: If data is found in cache line.
       If reclaim bit value =10(Newly inserted)
    Then reclaim bit value = 11 (Reference. bit)
       Else if reclaim bit value = 00 (oldest bit)
          Then reclaim bit value = 11
       Else if reclaim bit value = 01
    Then reclaim bit value = 11 (Reference. bit)
       End of If
Step5: In the case of lines are not accessed 2 or 3 iterations
       Less oldest line = 00 (oldest line)
       Referenced line = 01(less oldest line)
```

4. EXPERIMENTAL SETUP

In this paper, we used the Gem5 simulation tool which is a full system simulator. The gem5 simulator is an open source computer architecture simulator. Gem5 simulator provides a flexible, modular simulation system that makes it possible exploring multicore, multiprocessor architecture features. It has a various set of CPU models, system execution modes and memory system models such as caches etc [6][22]. In this tool we create environment of quad and octa core processors. For both the processors different cache replacement policies such as LRU, FIFO, Random, PLRU and our newly proposed policy are implemented.

Table 1 Simulation Parameters

Parameter	Values
CPU	X86 Quad core, Octa core
L1 Instruction Cache Size (KB)	32 KB fixed
L1 Data Cache Size (KB)	32 KB fixed
L2 Unified cache	32KB, 64KB, 128KB, 256KB, 512KB, 1024KB, 2048KB
Associativity	4 Way, 8 way
Replacement Policies	LRU, FIFO, Random, PLRU, LFU, RLLRU
Main Memory Size	1 GB
Benchmark	SPLASH 2 FFT and Redix

4.1 Performance Parameters

Hit Ratio: If data is found in cache memory then it is called as cache HIT. If data is not found in cache then it is called as cache MISS [4][5][18].

Hit Ratio = No. of hits / (No. of Hits + No. of Miss)

The cache hit ratio should be almost one. Miss Ratio is defined as:

Miss Ratio = 1- Hit Ratio

Execution Time: Time to complete the execution of workload [19].

5. RESULT AND ANALYSIS

Simulation results are the average of the result obtains after running same parameter 10 times for each. Here we are compared the result of existing replacement policies with our proposed cache replacement policy. Simulation results are obtained in various parameters such as Hit Ratio, Execution time etc. Here we present the graphs of these parameters and analysis the performance of multicore systems.

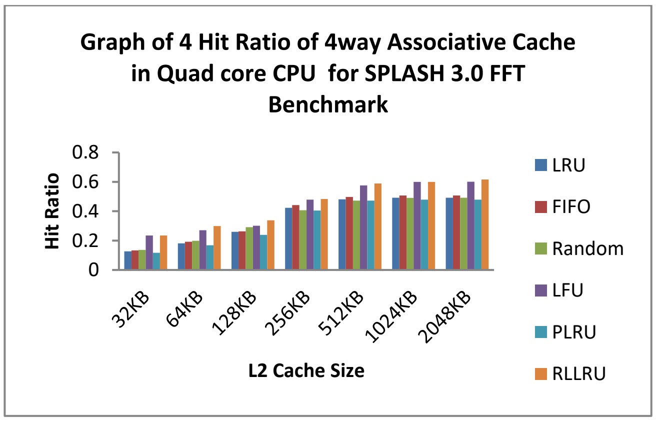

Fig.: 5.1 hit ratio of 4 way associative L2 cache for SPLASH FFT benchmark in quad core CPU.

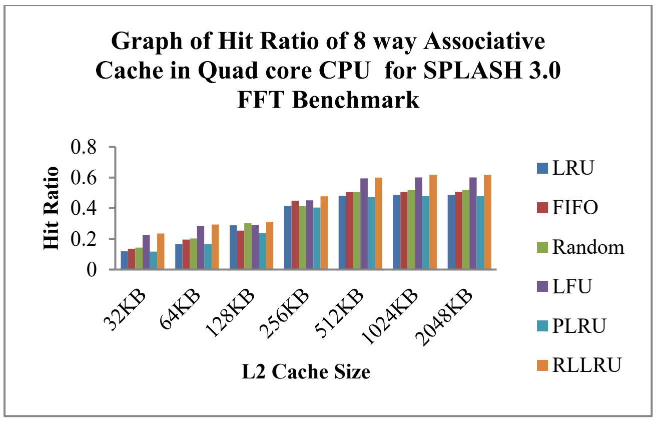

Fig.: 5.2 hit ratio of 8 way associative L2 cache for SPLASH FFT benchmark in quad core CPU.

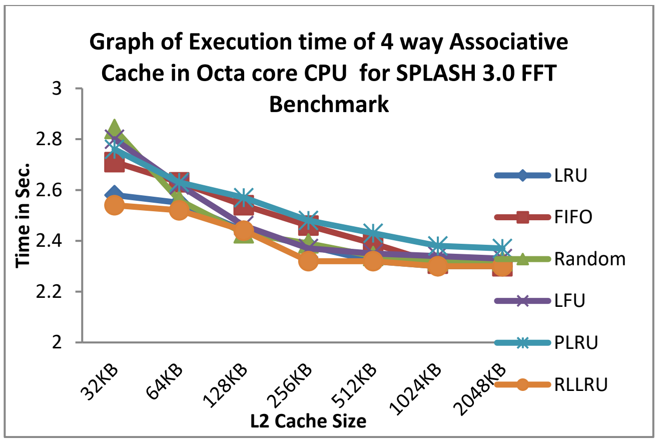

Fig.: 5.5 Execution time of 4 way associative L2 cache for SPLASH FFT benchmark in quad core CPU.

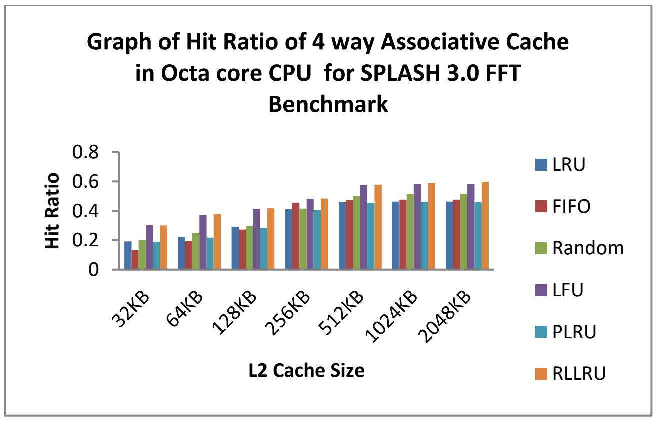

Fig.: 5.3 hit ratio of 4 way associative L2 cache for SPLASH FFT benchmark in octa core CPU.

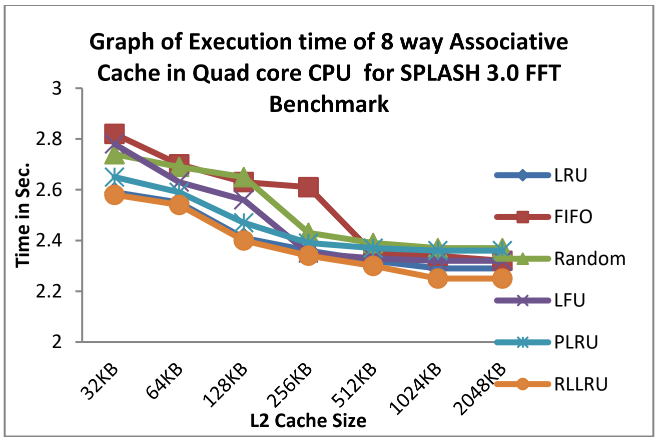

Fig.: 5.6 Execution time of 8 way associative L2 cache for SPLASH FFT benchmark in octa core CPU.

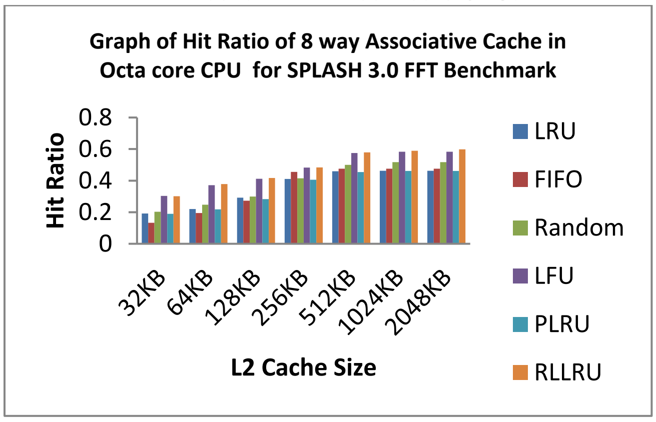

Fig.: 5.4 hit ratio of 8 way associative L2 cache for SPLASH FFT benchmark in octa core CPU.

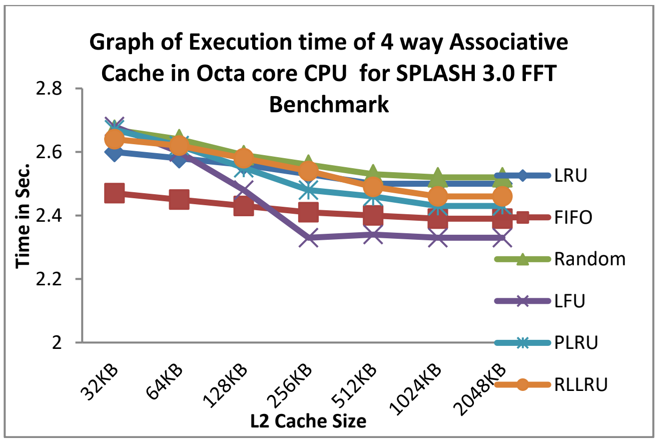

Fig.: 5.7 Execution time of 4 way associative L2 cache for SPLASH FFT benchmark in Octa core CPU.

Fig.: 5.1, 5.2, 5.3 and 5.4 presents the hit ratio of L2 cache memory. From these graphs we observed that when L2 cache size increases, for the SPLASH 3.0 FFT proposed RLLRU replacement policy gives high hit ratio.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

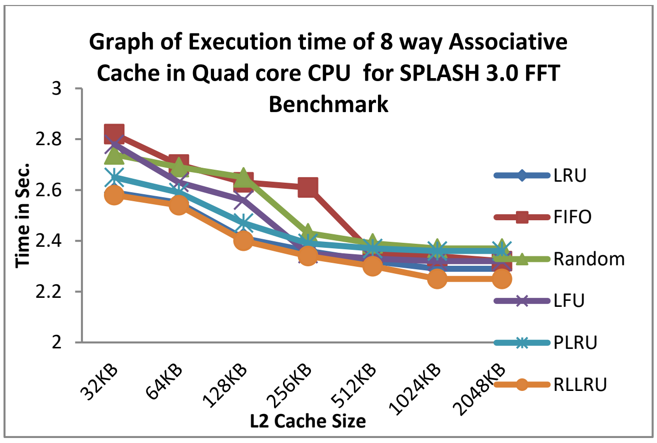

Fig.: 5.8 Execution time of 8 way associative L2 cache for SPLASH FFT benchmark in Octa core CPU.

Fig.: 5.5, 5.6, 5.7 and 5.8 presents the execution time of quad and octa processors. From these graphs we observed that when L2 cache size increases, for the SPLASH 3.0 FFT LRU and our proposed RLLRU replacement policy take less time to execute.

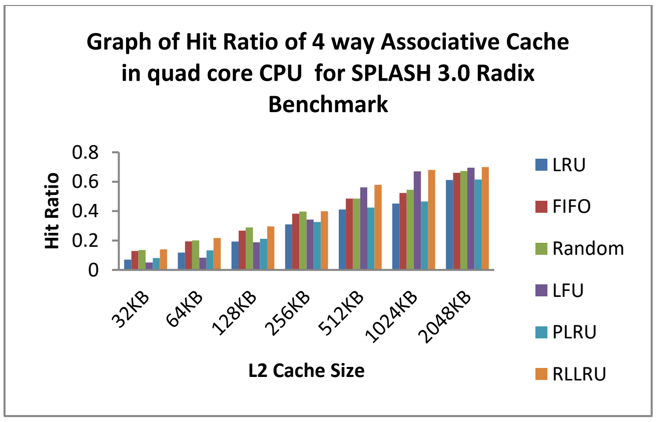

Fig.: 5.9 hit ratio of 4 way associative L2 cache for SPLASH Radix benchmark in quad core CPU.

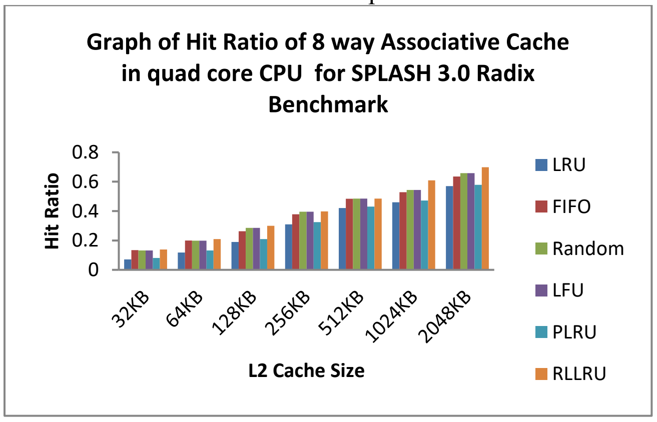

Fig.: 5.10 hit ratio of 8 way associative L2 cache for SPLASH Radix benchmark in quad core CPU.

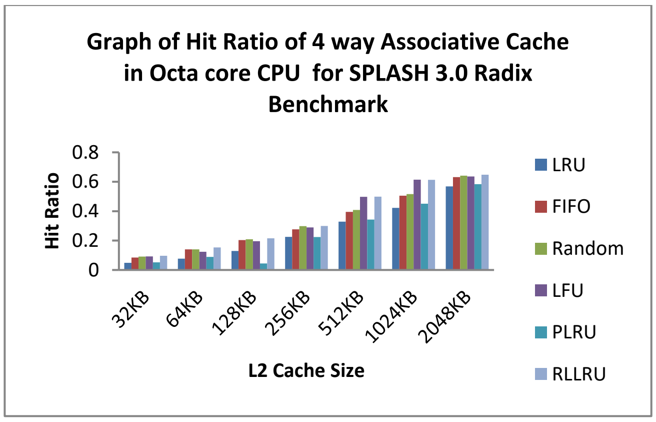

Fig.: 5.11 hit ratio of 4 way associative L2 cache for SPLASH Radix benchmark in octa core CPU.

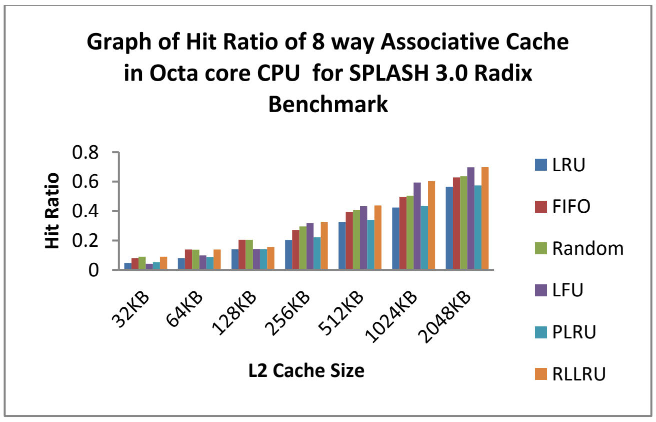

Fig.: 5.12 hit ratio of 8 way associative L2 cache for SPLASH Radix benchmark in octa core CPU.

Figure 5.9, 5.10, 5.11 and 5.12 presents the hit ratio of L2 cache memory. From these graphs we observed that when L2 cache size increases, for the SPLASH 3.0 Radix benchmark our proposed RLLRU replacement and LFU policy gives high hit ratio.

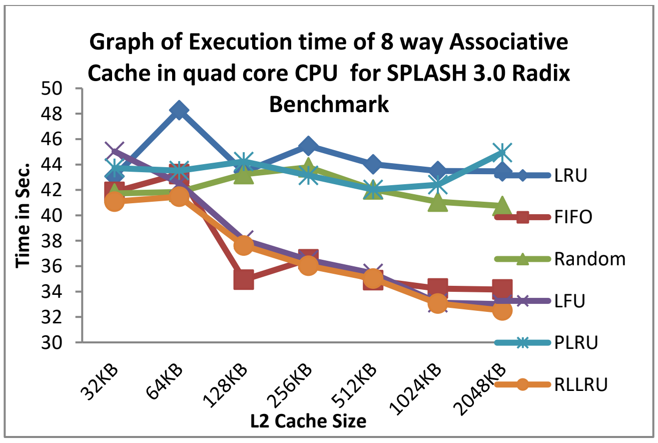

Fig.: 5.13 Execution time of 4 way associative L2 cache for SPLASH FFT benchmark in quad core CPU.

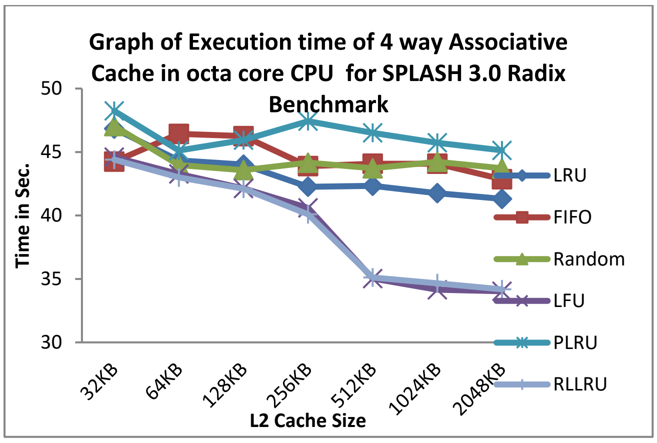

Fig.: 5.14 Execution time of 8 way associative L2 cache for SPLASH Radix benchmark in quad core CPU.

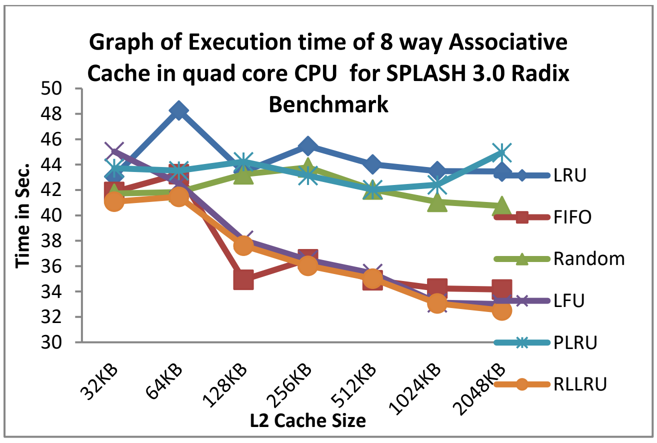

Fig.: 5.15 Execution time of 8 way associative L2 cache for SPLASH Radix benchmark in octa core CPU.

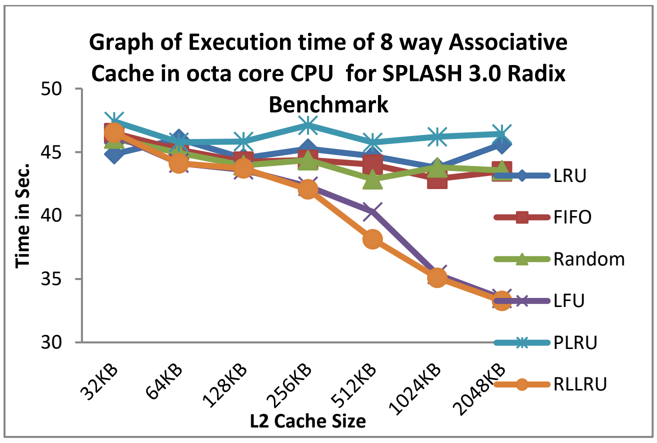

Fig.: 5.16 Execution time of 8 way associative L2 cache for SPLASH Radix benchmark in octa core CPU.

Fig.: 5.13, 5.14, 5.15 and 5.16 presents the execution time of quad and octa processors. From these graphs we observed that when L2 cache size increases, for the SPLASH 3.0 Radix benchmark LFU and our proposed RLLRU replacement policy take less time to execute.

CONCLUSION

In this paper we present a new RLLRU cache replacement policy which is enhanced from of the LRU policy. In our proposed policy search time of LRU is reduced by associating reclaim bits. This paper also describes the cache replacement policies in the form of their performance analysis on SPLASH 3.0 FFT and Radix benchmark. The analysis of experimental results presents that our proposed RLLRU is the most scalable cache replacement policy. Sometime LFU and Random replacement policies give better results. In our future work we will improves the performance of RLLRU policy for different benchmark and compare with other replacement policies in multicore systems

REFERENCES

[1] John L. Hennessy, David A. Pattersons,"Computer Architecture A Quantitative Approach", Third Edition Morgan Kaufmann Elsevier, ISBN 81-7867-266-9, 2003, pp 390-391.
[2] Alan Jay Smith, "Cache memory", Computing Surveys, ACM 0010-4892/82/0900-0473, Vol. 14, No. 3, September 1982, pp 473-531.
[3] Abu Asaduzzaman, Fadi N. Sibai, Manira Rani, "Improving cache locking performance of modern embedded systems via the addition of a miss table at the L2 cache level", Journal of Systems Architecture, Elsevier, Vol. 56, No. 6, 2010, pp 151–162.
[4] John P.Hayes, "Computer Architecture and organization", Third Edition Tata McGraw Hill, ISBN: 0-7-0027355-3, 1998, pp 451-452.
[5] William Stalling, "Computer Organisation and Architecture", Seventh Edition Pearson Education, ISBN: 978-81-7758993-1, 2005, pp 30-31.
[6] Hussein Al-Zoubi, AleksandarMlienkovic, Milena Mlienkovic, "Performance Evaluation of Cache Replacement Policies for the SPEC CPU2000 Benchmark Suit", Proceeding of the 42th annual southeast regional conference (ACM-SE'42), ACM, ISBN: 1-58113-870-9, April 2004, pp 267-272.
[7] A.Malamy, R.Patel and N.Hayes, "Methods and Apparatus for Implementing A Pseudo-LRU Cache Memory Replacement Scheme with A Locking Feature", United States Patent, patent no. 5353425, October 1994.
[8] Moinuddin K. Qureshi, Aamer Jaleel, Yale N. Patt, Simon C. Steely Jr. , Joel Emer, "Adaptive Insertion Policies for High Performance Caching", Proceedings of the 34th annual international symposium on Computer architecture (ISCA'07), ACM, ISBN: 978-1-59593-706-3, June 2007, pp 381-391.
[9] Gangyong Jia, Xi Li, Chao Wang, Xuehai Zhou, Zongwei Zhu, "Cache Promotion Policy using Re-Reference Interval Prediction",IEEE International Conference on Cluster Computing 12 , IEEE Computer Society, ISBN: 978-0-7695-4807-4/12, 2012, pp 534-537.
[10] James E. Smith, James R. Goodman, "Instruction cache replacement policies and organizations", IEEE Transactions on Computers, Vol. 34, No. 3, March 1985, pp 234-241.
[11] Jaeheon Jeong and Michel Dubois,"Cost-Sensitive Cache Replacement Algorithms", HPCA '03 Proceedings in the 9th International Conference, ACM, ISBN: 0-7695-1871-0, 2003, pp 327-338.
[12] M.S. Obaidat, "A Trace-Driven Simulation Study Cache Systems of Two-Level", Computers Electric Engineer, Elsevier, Vol. 21, No. 3, 1995, pp 201-210.

[13] Fei Guo and Yan Solihin, "An Analtical Model for Cache Replacement policy Performance", Proceedings of the joint international conference on Measurement and modeling of computer systems SIGMETRICS '06, ACM, ISBN: 59593-320-4/06/0006, June 2006, pp 228-239.

[14] Damien Hardy, Isabelle Puaut, "WCET analysis of instruction cache hierarchies", Journal of Systems Architecture, Elsevier, vol. 57, Issue 7 , 2011, pp 675–734.

[15] Nimrod Megiddo, Dharmendra S. Modha, "Outperforming LRU with an Adaptive Replacement Cache Algorithm", IEEE Computer Society, Vol. 37, No. 4, 2004, pp 58- 65.

[16] Fang Juan, Li Chengyan, "An Improved Multi-core Shared Cache Replacement Algorithm", proceeding of the 12th International Symposium on Distributed Computing and Applications to Business, Engineering & Science, IEEE Computer Society, ISBN: 978-1-4673-2630-8, 2012, pp 13-17.

[17] K. M. AnandKumar, S. Akash, D. Ganesh and M. S. Christy, "A hybrid cache replacement policy for heterogeneous multi-cores," 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Delhi, India, ISBN 978-1-4799-3080-7,2014, pp. 594-599.

[18] S. Ding and Y. Li, "LRU2-MRU collaborative cache replacement algorithm on multi-core system," 2012 IEEE International Conference on Computer Science and Automation Engineering (CSAE), Zhangjiajie, China, 978-1-4673-0089-6 ,2012, pp. 395-398.

[19] Dhammpal Ramtake, Sanjay Kumar, "Cache Replacement and Allocation Technology in Multicore System: A study of Security Issue". Research J. Engineering and Tech.,Vol. 11, Issue 2, 2020, ISSN (Online) 2321-581X, pp 103-108,

[20] Dhammpal Ramtake, Sanjay Kumar, "Impact of Cache Replacement Policies on Split Level One Cache Memory in Multicore System", International Journal of Emerging Technologies and Innovative Research, Vol.6, Issue 4, April 2019, ISSN:2349-5162, pp 790-798.

[21] Dhammpal Ramtake, Sanjay Kumar, "Performance Analysis of First Level Cache memory Replacement Policies in Multicore Systems", IJERCSE, vol. 5, 2018, ISSN (Online) 2394-2320, pp 505-511.

[22] Dhammpal Ramtake, N. Singh, Sanjay Kumar and V. K. Patle, "Cache Associativity Analysis of Multicore Systems," IEEE Xplore, 2020 International Conference on Computer Science, Engineering and Applications (ICCSEA), Gunupur, India, 2020, ISBN: 978-1-7281-5830-3, pp. 1-4.

[23] Sanjay Kumar, "Mathematical modeling and Simulation of a buffered Fault Tolerant Double Tree Network" in proc. of 15th International Conference on Advanced Computing and Communications ADCOM 2007 at IIT, Guwahati on 18-21 December, 2007, pp 422-431.