Impact of Cache Replacement Policies on Split Level One Cache Memory in Multicore System

[1]Dhammpal Ramtake, [2]Sanjay Kumar
[1]Research Scholar, [2]Professor,
[1] SoS in Computer Science & IT,
Pt. Ravishankar Shukla University, Raipur, Chhattisgarh, India.

Abstract: Nowadays, multicore system plays very significant role to process many computer applications. In multicore systems number of cores is embedded on one single chip. Each core has capability to execute millions of instruction in one second. In multicore environment as the number of levels of cache increases, the cache access time tends to consume a major percentage of memory latency and memory access latency decreases the performance of multicore systems. To reduce the access latency cache replacement policy is used. In this paper we implement the cache replacement policies such as LRU, FIFO, Random, PRLU and RRIP in gem5 simulation tool with 4 core (Quad core) X86 CPU architecture. In simulation 10×10, 100×100 and 500×500 floating point matrix multiplication object code is used as load. Results are obtained in hit ratio, miss ratio and access time parameters.

Index Terms **– cache replacement policy, Split cache, Multicore System.**

I. INTRODUCTION

The use of multiple processors on the same chip, also referred to as multiple cores, or multicore, provides the potential to increase performance without increasing the clock rate. Studies show that, within a processor, the increase in performance is roughly proportional to the square root of the increase in complexity. But if the software can support the effective use of multiple processors, then doubling the number of processors almost doubles performance. Thus, the strategy is to use two simpler processors on the chip rather than one more complex processor. In addition, with two processors, larger caches are justified [1]. In multicore architecture, two or more independent cores are combined into a single processing chip. Cache memory resides between CPU and main memory [2]. The cache contains a copy of data of portions of main memory. In most of the cases, each processor has its own private level-1 cache memory (L1). Normally, the L1 is split into instruction (I1) and data (D1) caches. Also, multicore processors may have one shared level-2 (L2) cache or multiple distributed and dedicated L2s. Caches are classified according to their function or the nature of the information stored in the cache [3]. An instruction cache stores only instruction. Data cache stores only the data. Unified cache stores both instruction and data.

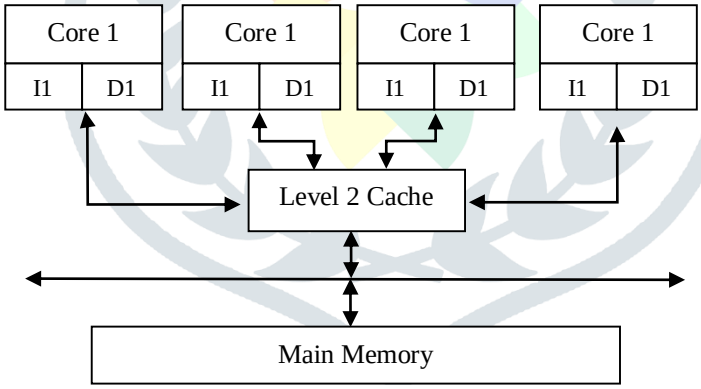

Fig.1.1. Quad core CPU with 2 Levels of cache memory

Fig.1.1 shows the basic model of quad core processor with non shared L1 spilt Cache memory and shared L2 cache connected via system bus. This paper discusses various cache structure and replacement algorithm which would exploit on this phenomenon in the context of a specific multicore system.

II. CACHE MEMORY

Cache memory is top level of memory in memory hierarchy of modern computer system. The main purpose of caches is to reduce the latency of references to memory. Ideally, the total latency should one clock cycle of instruction reads and data read or writes. Cache memory is divided into two different parts; one is cache data memory and another is cache tag memory. Cache data memory contains various collections of memory words called cache block or line or page. Each cache block has a block address or tag. Collection of all block addresses or tags is called cache tag memory. The basic characteristic of cache memory is its fast access time. Therefore, very little or no time must be wasted when searching for words in cache. The transformation of data from main memory to cache memory is referred to as mapping process [4]. Cache Mapping technique is classified into three different classes i.e. direct mapping, associative mapping and set associative mapping. Another factor which can also affect the performance of cache memory is locality of reference. In this principle references of instruction and data access pattern through CPU is recognised. The locality concept is classified into three part i.e. temporal locality, spatial locality and sequential locality. Temporal locality is often known as look backward. That is an instruction sequence such as a loop will reuse instructions. Spatial locality is often known as look forward. In this locality, portions of address space near the current location of reference are likely to be referenced in near feature [14]. In sequential locality next referenced will be the immediate successor of the present reference.

III. CACHE REPLACEMENT POLICIES

Cache replacement policies determine which data blocks should be removed from the cache when a new data block is added. For direct mapping, there is only one line for any particular block. For associative mapping and set associative mapping, a replacement algorithm is needed [3] [5]. In these caches, the general goal of replacement policies is to minimize future cache misses by removing a line that will not be referenced often in the future. There are various cache replacement policies such as FIFO (First In First Out), LRU (Least Recently Used), RANDOM, RRIP, P-LRU etc [5]. FIFO selects for replacement of the block least recently loaded into cache. LRU policy selects for replacement of the block that was least recently accessed by the processor. A RANDOM replacement policy would select a block to replace in a totally random order, with no regard to memory references or previous selections. Pseudo-LRU 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 [6]. A Re-Reference Interval Policy (RRIP) utilizes the cache blocks or lines history of average re-accesses time to predict its future access pattern. This prediction is achieved by a bit counter known as Re- Reference Value (RRV). Re- reference value counter is associated with each line on cache memory. When data is found or hit occurs then RRV is incremented by one. When miss is occurred or line is replaced then RRV is set to zero [7].

IV. LITERATURE REVIEW

In this literature review present study of development cache replacement policies is done where number of researcher presented their contribution. Damien Hardy et al [10] proposed a method to produce safe worst-case execution time estimations of set-associative instruction cache hierarchies, for different cache hierarchy management policies and different replacement policies between cache levels. Aamer Jaleel et al [11] proposed new Static RRIP (SRRIP) that is scan-resistant and Dynamic RRIP (DRRIP) that is both scan-resistant and thrash-resistant by using RRIP. These two policies were easily integrated into existing LRU approximations found in modern processors. Armin Vakil-Ghahani[12]et.al presented the strong correlation between the expected number of hits of a cache block and the reciprocal of its reuse distance. They were found a cost effective hit count based victim-selection replacement policy. According to et al. A. Jain [13] data prefetchers, cache replacement policies are faced with a large unexplored design space. In particular, they was observe that while Belady's MIN algorithm minimizes the total number of cache misses including those for prefetched lines it does not minimize the number of demand misses. These literatures are present scenario of development of cache replacement policies. In this paper, we study the feature trend of instruction and data cache behavior and make experiment which provides simulation and analysis of split non-sharable cache memory.

V. EXPERIMENTAL SETUP

In this work we have simulated various cache replacement policies such as LRU, FIFO, RANDOM, PLRU and RRIP with the help of GEM5 simulation. It is an event-driven open source simulation framework that has different abstraction levels, balancing simulation speed and accuracy. GEM5 simulator provides a flexible, modular simulation system that makes it possible exploring multicore, multiprocessor architecture features. It has a diverse set of CPU models, system execution modes and memory system models such as caches etc. [6]. This tool runs on Linux operating system, bind with GCC or Python compiler and make platform for binary file. To evaluate the performance of L1 cache on different cache replacement policies, we have implemented our experiment on quad core (4 core) processor with independent L1 instruction cache and L1 data cache, L2 shared cache with 4-way associative. After environment setup we give the input of binary code of matrix multiplication.

Table 1 Simulation Environment

Parameter	Values
CPU	X86 Quad core
L1 Instruction Cache Size (KB)	4, 8, 16, 32, 64, 128, 256, 512
L1 Data Cache Size (KB)	4, 8, 16, 32, 64, 128, 256, 512
L2 Unified cache	1 MB fixed
Associativity	4 Way Set Associativity
Replacement Policies	LRU, FIFO, Random, PLRU, RRIP
Main Memory Size	1 GB
Matric Size (Floating Pont)	10 × 10, 100 × 100, 500 × 500

5.1 Performance Parameter

For measuring the performance of cache memory we use hit ratio, miss ratio, instructions per cycle and cycle per instructions. Hit Ratio denoted by H is defined as the ratio of the total number of hits and total no. of hits and misses.

$$Hit\ Ratio = \frac{Total\ number\ of\ Hits}{Total\ number\ of\ hit + Total\ number\ of\ miss}$$

Effective Access Time (EAT)

Effective access time of a hierarchical memory is measured by its effective access time or the average time per access. EAT is a weighted average that uses the hit ratio and relative access time of successive levels of hierarchy. The actual access time for each level depends on the technology and method used for access.

$$EAT = Hit\ Rate \times L1\ cache\ access\ cycle + (1 - Hit\ Rate\ ) \times L2\ cache\ access\ cycle$$

In this paper we consider the access time in CPU cycles. In Gem5 simulation tool it is defined that, CPU takes one cycle for fetch the instruction or data form level one cache. Form level two CPU takes 6 cycles for fetch the instruction or data [8]. With these considerations results are evaluated.

IV. RESULTS AND DISCUSSION

After the implementation of above configuration in Gem5 simulation results are obtained.

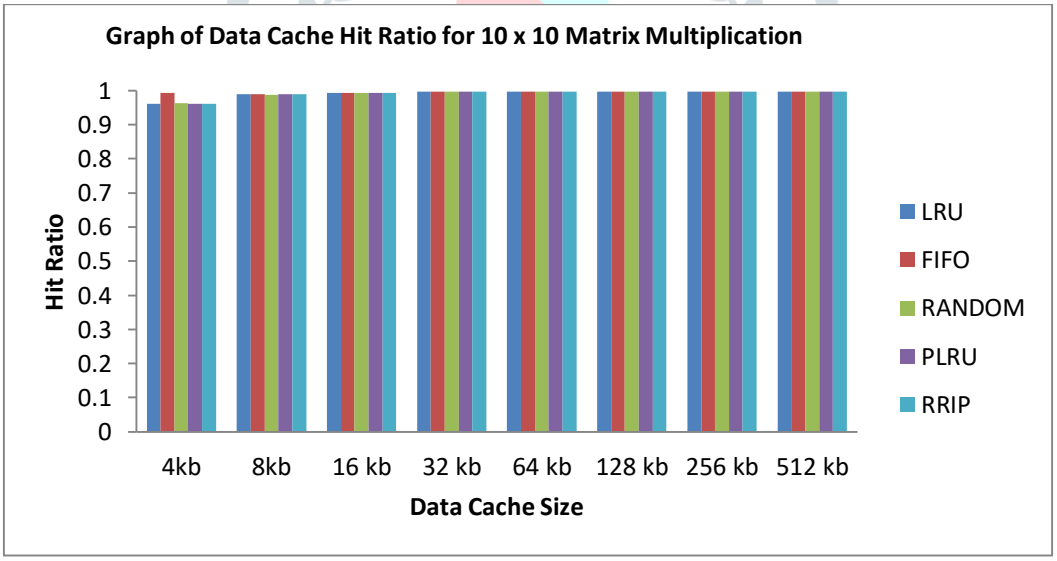

Fig.:1 Graph of Data Cache Hit Ratio for 10 x 10 Matrix Multiplication

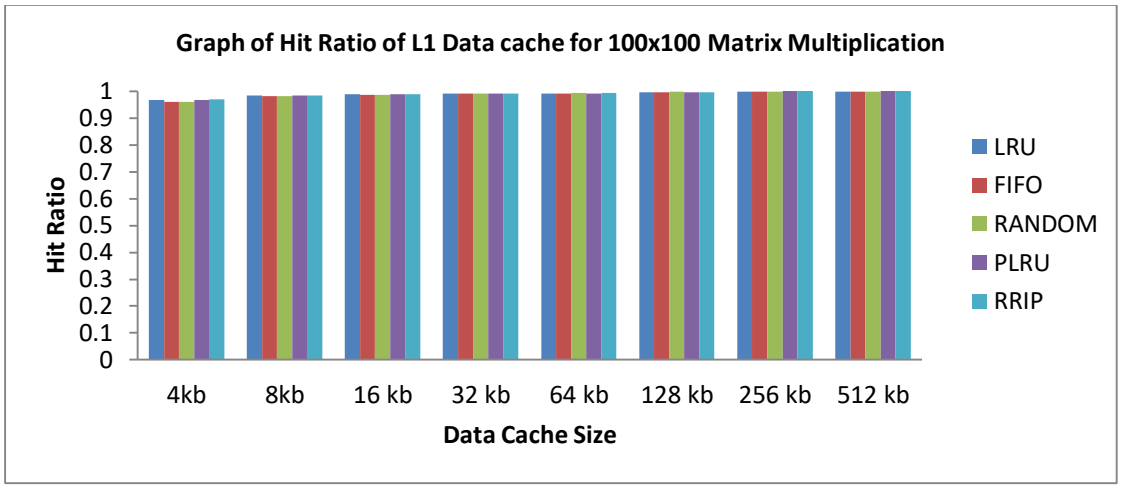

Fig.:2 Graph of Data Cache Hit Ratio for 100 x 100 Matrix Multiplication

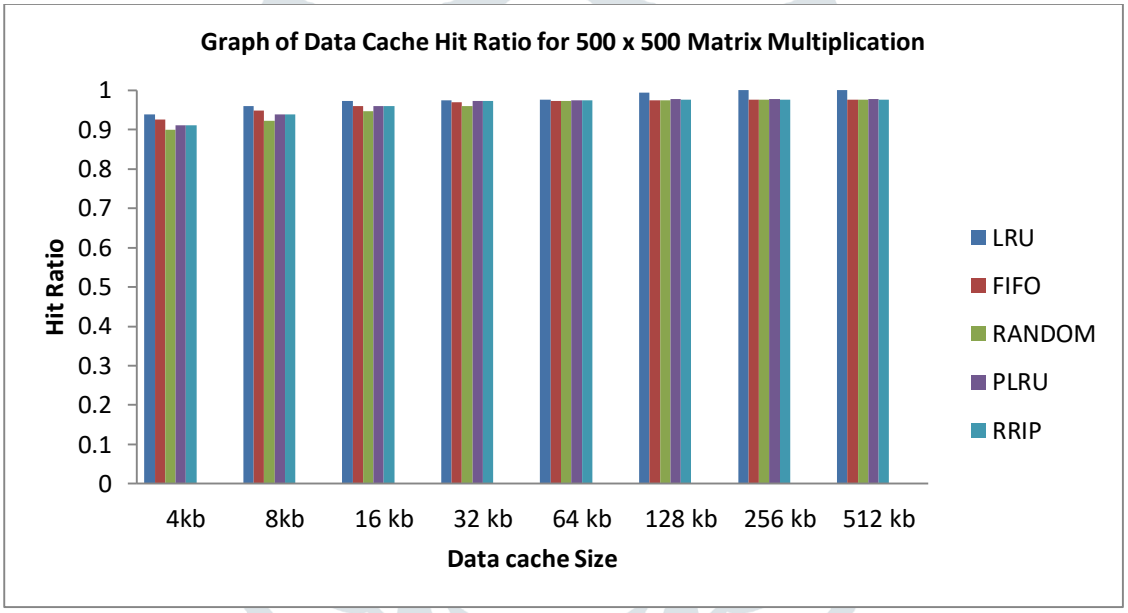

Fig.:3 Graph of Data Cache Hit Ratio for 500 x 500 Matrix Multiplication

Fig. 1, 2 and 3 shows the hit ratio of data cache. From these graphs we observed that when data cache size increases, for the 10 x 10 problem LRU and FIFO replacement policy gives better hit ratio. For the 100 x 100 and 500 x 500 problem LRU and RRIP replacement policies gives better hit rate.

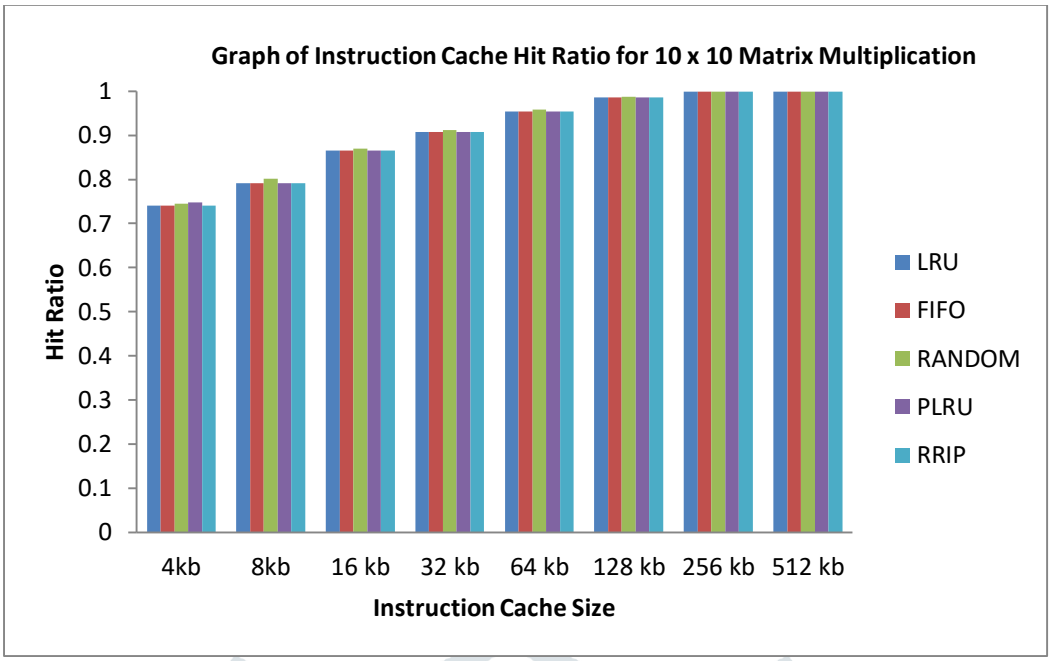

Fig.:4 Graph of Instruction Cache Hit Ratio for 10 x 10 Matrix Multiplication

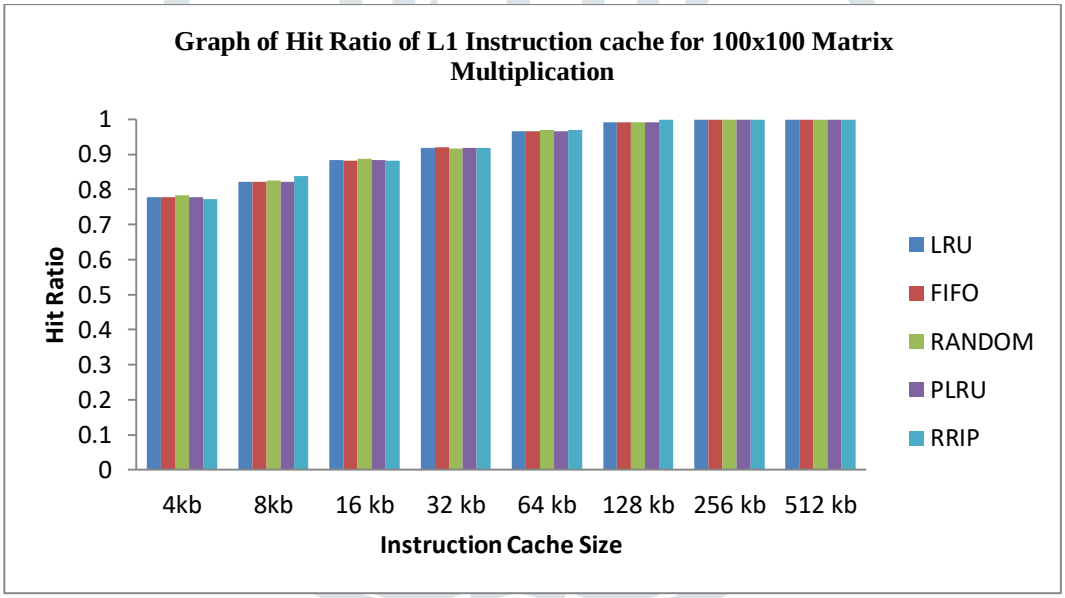

Fig.:5 Graph of Instruction Cache Hit Ratio for 100 x 100 Matrix Multiplication

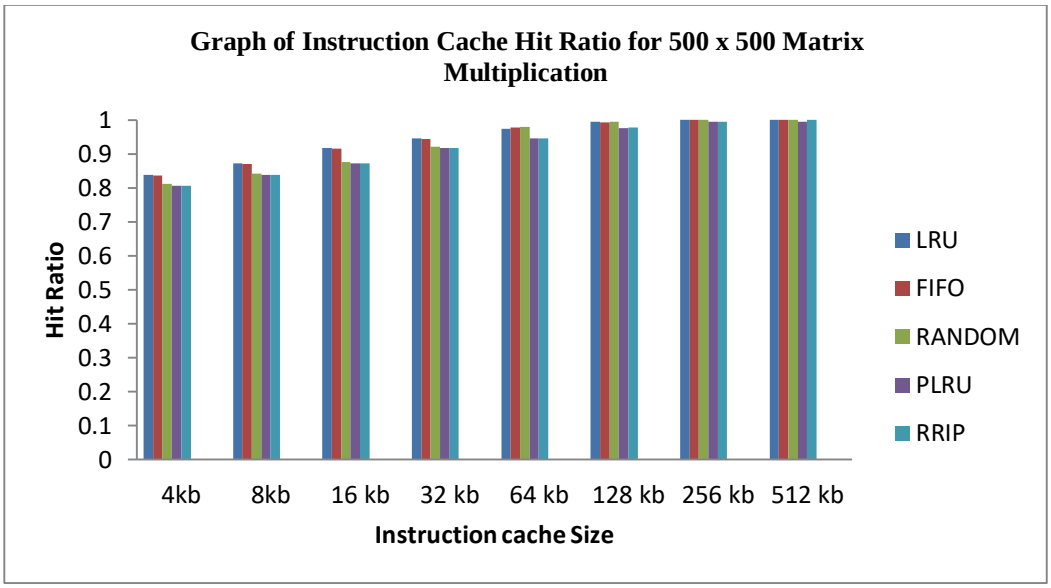

Fig.:6 Graph of Instruction Cache Hit Ratio for 100 x 100 Matrix Multiplication

Fig.: 4, 5, and 6 presents the hit ratio of Instruction cache. From these graphs we observed that when instruction cache size increases, for the 10 x 10, 100 x 100 and 500x 500 problems LRU replacement policies gives better hit ratio.

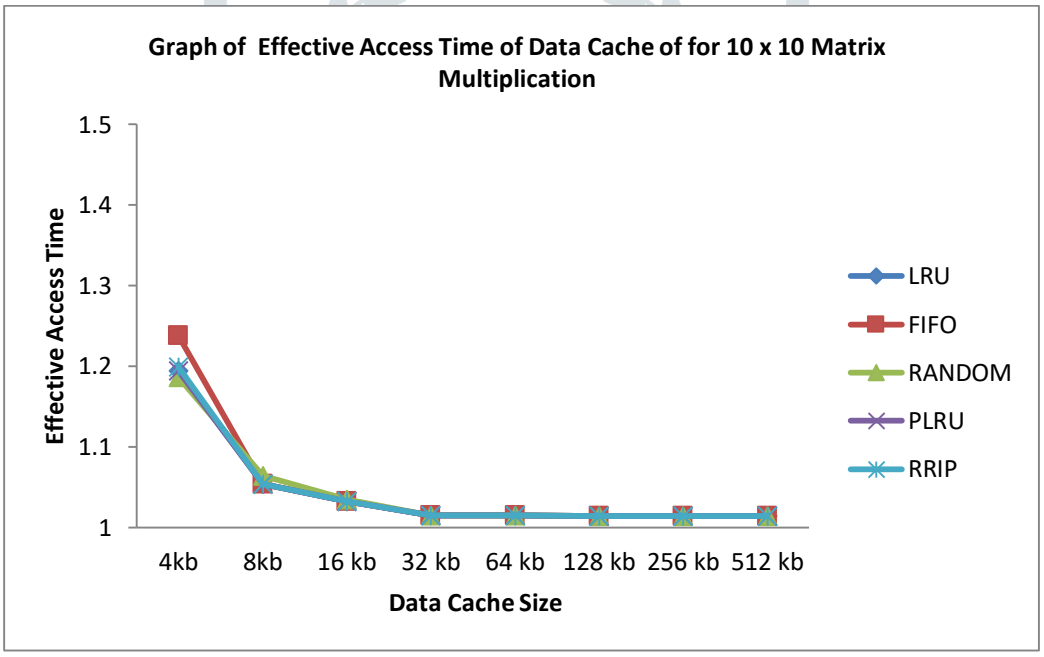

Fig.: 7 Graph of Effective Access Time of Data Cache of for 10 x 10 Matrix Multiplication

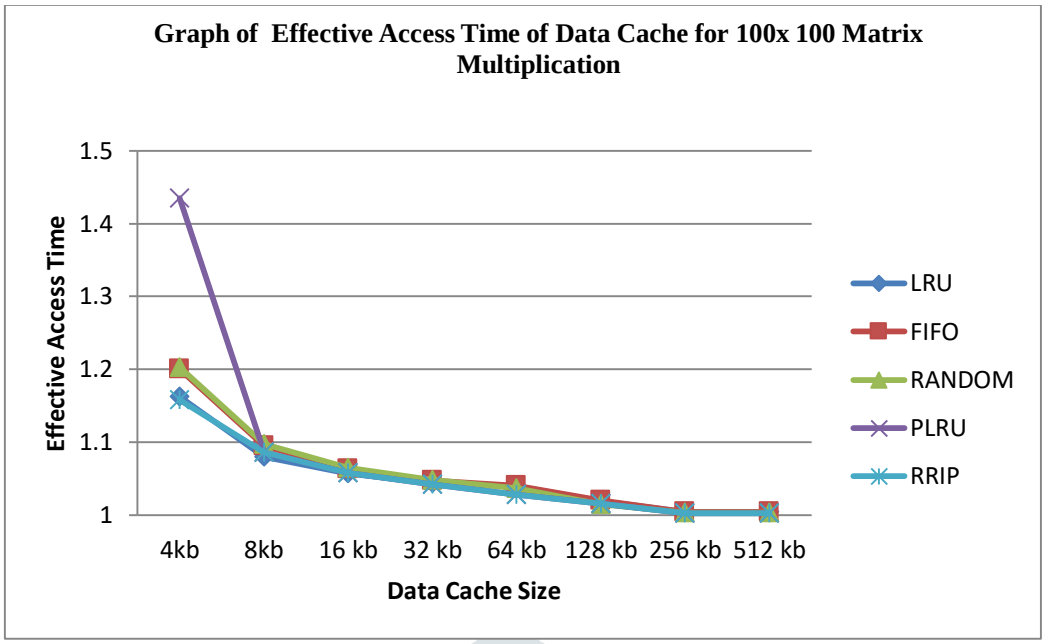

Fig.: 8 Graph of Effective Access Time of Data Cache of for 100 x 100 Matrix Multiplication

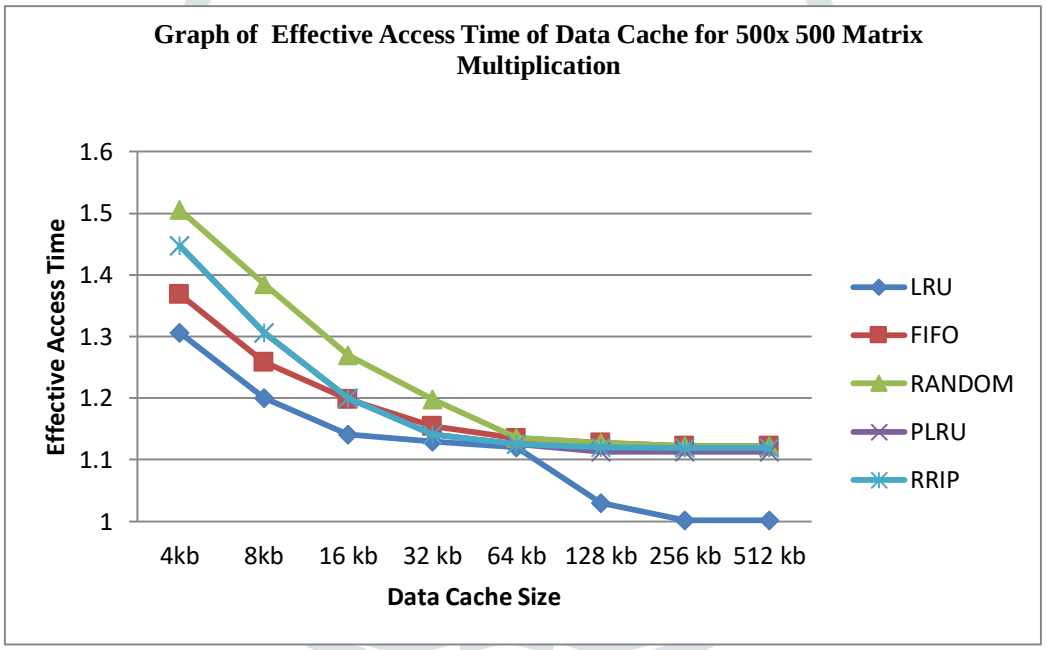

Fig.: 9 Graph of Effective Access Time of Data Cache of for 100 x 100 Matrix Multiplication

Fig.: 7, 8 and 9 presents the effective access time of level one data cache for 10 x 10, 100 x 100, and 500 x 500 matrix multiplication respectively. From these graphs, it is observed that for increasing size of data cache with LRU and RRIP policy memory access time (CPU cycles) decreases.

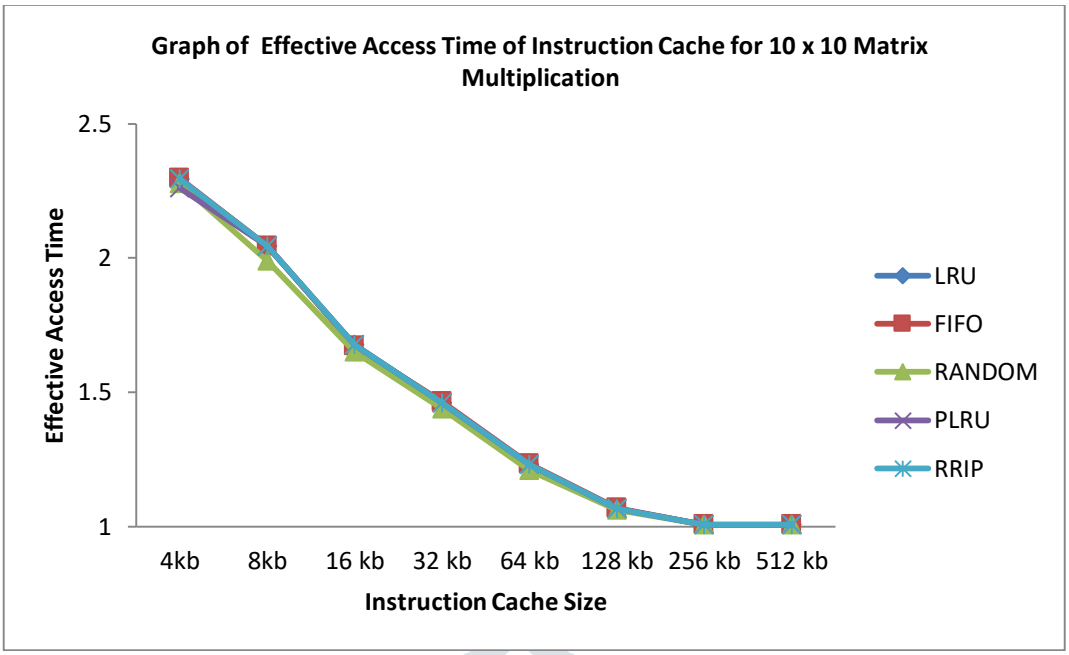

Fig.: 10 Graph of Effective Access Time of Instruction Cache of for 100 x 100 Matrix Multiplication

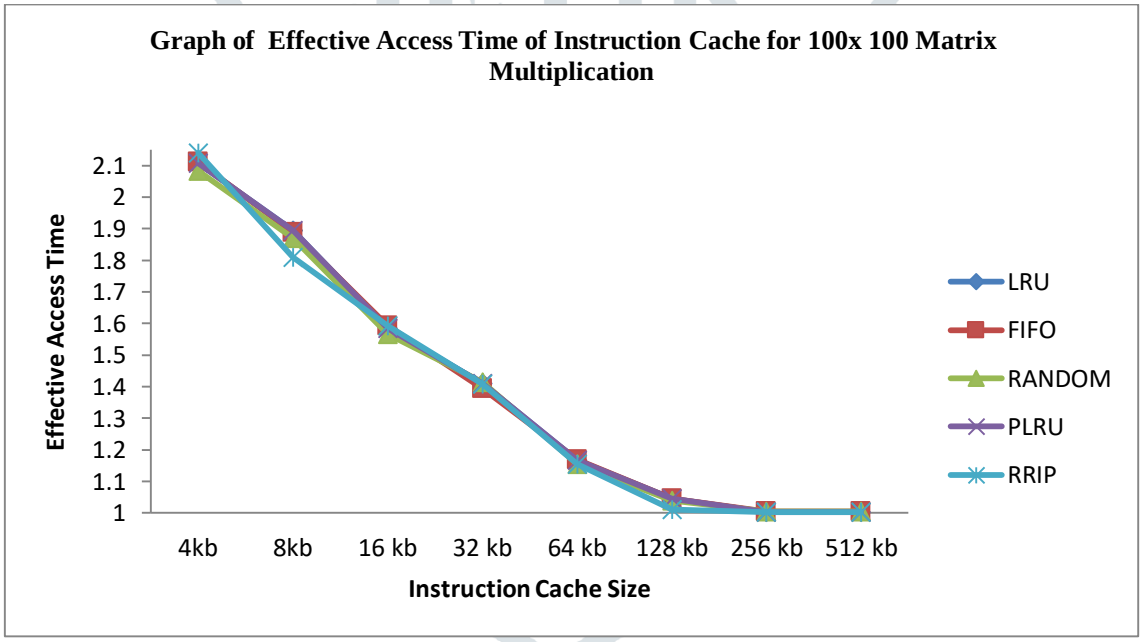

Fig.: 11 Graph of Effective Access Time of Instruction Cache of for 100 x 100 Matrix Multiplication

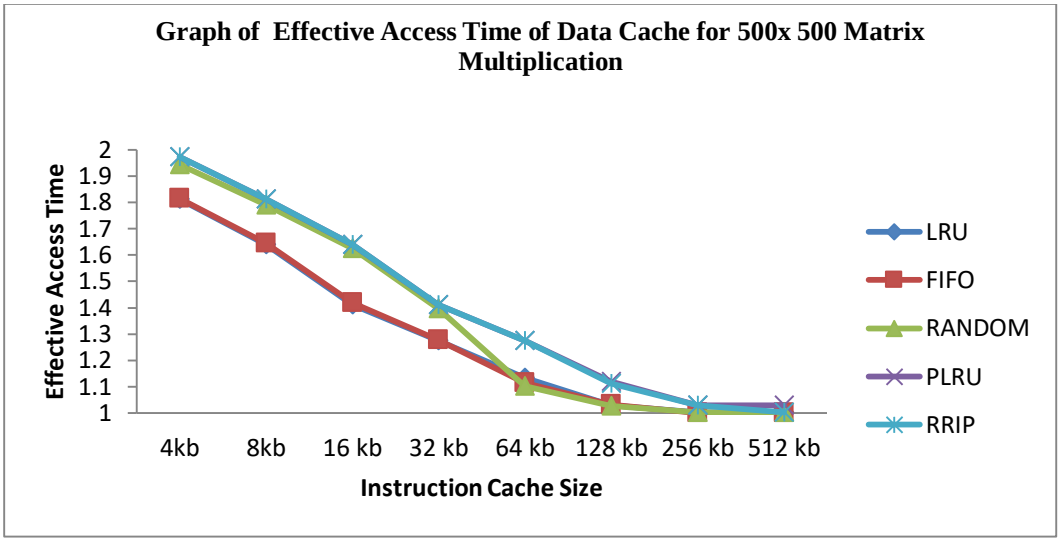

Fig.: 12 Graph of Effective Access Time of Instruction Cache of for 500 x 500 Matrix Multiplication

Fig. 10, 11 and 12 represents the effective access time of level one instruction cache for 10 x 10, 100 x 100 and 500 x 500 matrix multiplication respectively. From these graphs, it is observed that for increasing size of instruction cache with FIFO and random policies memory access time (CPU cycles) decreases.

CONCLUSION

In this paper simulation and performance evaluation is performed for level one split cache memory in multicore system. Cache replacement policies are implemented on Gem 5 tool for multicore system. It is observed from the result analysis that performance of cache memory with temporal locality work load, LRU policy and sometime Random policy performed better compared with other replacement policies. The LRU strategy has the unfortunate property that line can evicted just before it requested but workload type is reference instruction and sequential element of data. Therefore, main conclusion is that the performance is depended on occurrence instruction and data access. In our future work we may implement these policies for level 2 and level 3 cache memories in multicore systems with various CPU benchmarks.

REFERENCES

[1] William Stalling, "Computer Organisation and Architecture", Ninth Edition Pearson Education, ISBN 13: 978-0-13-293633-0, 2013, pp 43-44.
[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, Vol. 56, No. 6, Elsevier, 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] 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.
[6] 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.
[7] 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.
[8] N. Binkert "The gem5 simulator", ACM SIGARCH Computer Architecture News, vol.39 No. 2, 2011, pp 1-7.
[9] Anastasiia Butko, Rafael Garibotti, Luciano Ost and Gilles Sassatelli, "Accuracy Evaluation of GEM5 Simulator System", 2012
[10] Damien Hardy, Isabelle Puaut, "WCET analysis of instruction cache hierarchies", Journal of Systems Architecture, Elsevier, vol. 57, Issue 7 , 2011, pp 675–734.
[11] Aamer Jaleel, Kevin B. Theobald, Simon C. Steely Jr. Joel Emer, "High Performance Cache Replacement Using Re-Reference Interval Prediction (RRIP)", proceeding of the 37th International Symposium on Computer Architecture (ISCA'10,) ACM, ISBN: 978-1-4503-0053, June 2010, pp 60-72.
[12] A. Vakil-Ghahani, S. Mahdizadeh-Shahri, M. Lotfi-Namin, M. Bakhshalipour, P. Lotfi-Kamran and H. Sarbazi-Azad, "Cache Replacement Policy Based on Expected Hit Count," IEEE Computer Architecture Letters, vol. 17, no. 1, june 2018, pp. 64-67,
[13] A. Jain and C. Lin, "Rethinking Belady's Algorithm to Accommodate Prefetching," 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), Los Angeles, CA, 2018, pp. 110-123,
[14] Dhammpal Ramtake and Sanjay Kumar, "Performance Analysis of First Level Cache memory Replacement Policies in Multicore Systems", IJERCSE, vol. 5, pp 505-511, 2018.