ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 2, February 2018

Performance Analysis of First Level Cache Memory Replacement Policies in Multicore Systems

[1] Dhammpal Ramtake, [2] Sanjay Kumar
[1] School of Study in Computer science & IT
Pt. Ravishankar Shukla University, Raipur (Chhattisgarh) 492010 India

Abstract **- Nowadays, processing speed is one of the most important performance criteria of modern multicore processors. For achieving higher processing speed of processor various components are used, in which cache is one of them. As modern processors include multiple levels of caches and as cache associativity increases, it is important to revisit the effectiveness of common cache replacement policies. In this paper, we have analyzed the impact of different replacement policies such as LRU (Least Recently Used), FIFO (First In First Out), RANDOM, DIP (Dynamic Insertion Policy), PLRU-t (Pseudo Least Recently Used tree-based). We have used Simple Scalar as a simulation tool. We have taken the problem of matrix multiplication of different size 10 x 10, 100 x 100, 500 x 500.**

Keywords: Cache memory; Multicore system; replacement policies.

I. INTRODUCTION

Dictionary meaning of cache is "A collection of item of the same type stored in a hidden or inaccessible place". Caches are generally the top level of the memory hierarchy and are made of SRAM (Static Random access Memory). The main structural difference between a cache and other level in the memory hierarchy is that caches use hardware to locate memory addresses whereas other memories use software or a combination of software and hardware. Cache memories are small fast memories used to temporarily hold the contents of portions of main memory that are likely to be used. Today caches have become an integral part of all processors. Performance improvement of microprocessors historically comes from both increasing the speed or frequency at which the processors run and by increasing the amount of task performed in each cycle. The increasing number of transistors on a chip has led to different ways of increasing parallelism [1].

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.

A. Cache Memory

Cache memory was first seen in the IBM system/360 Model 85 in 1960. In 1980s, the Intel 486DX microprocessor introduced an on chip 8 KB L1 cache for the first time. In early 1990s an off chip L2 cache appeared with the 86DX4 and Pentium microprocessor. Generally, microprocessors usually have 128 KB or more of L1 and 512 KB or more of L2 and optional 2 MB or more Level 3 (L3) cache [2]. Cache memory resides between CPU and main memory.

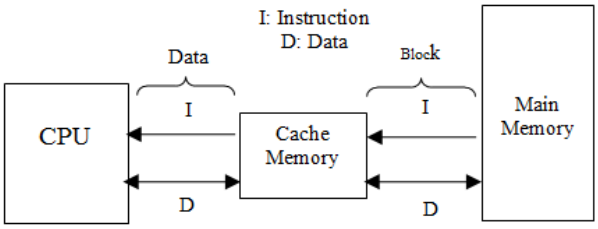


Fig:-1. Block Diagram of Cache

The cache contains a copy of data of portions of main memory. When the processor attempts to read a word of memory a check is made to determine if the word is in the cache. If so, the word is delivered to the processor. If not complete block consisting of that memory word is brought into the cache and then that word is delivered to the processor.

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. When the CPU refers to memory and finds the word in cache, it

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 2, February 2018

is said to produce a cache hit. If the word is not found in cache, it is in main memory and it counts as a miss. M. M. D.Hill et. al [3] classify cache misses into three categories: compulsory miss, conflict miss and capacity miss. A compulsory miss is the first access to a cache line. A capacity miss occurs when the cache size is too small to hold all the cache lines referenced by a program. A conflict miss occurs when multiple cache lines are mapped to the same set in the cache and the program subsequently references an evicted line. The transformation of data from main memory to cache memory is referred as a mapping process. Basically, there are three methods for mapping addresses to cache locations - direct mapping, associative mapping and set associative mapping. Direct mapping is the simplest technique which maps each block of main memory into only one possible cache line. In associative mapping each block of main memory maps into any line of the cache. In set associative mapping cache is divided into sets, each of which consists of cache lines or blocks and each block of main memory maps into any of lines of set [4].

II. REPLACEMENT POLICIES

Cache replacement policies determine which data blocks should be removed from the cache when a new data block is added. Well known policies are as follows: FIFO (First In First Out), LRU (Least Recently Used), RANDOM. FIFO selects for replacement of the block least recently loaded into cache. FIFO has advantage that it is very easy to implement by using a circular counter which points to the next cache block to be replaced; the counter is updated on every cache miss [5]. LRU policy selects for replacement of the block that was least recently (oldest block) accessed by the processor. 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 cache memory line time counter values [6]. 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 random order, with no consider to memory references or previous selections [7]. Apart from this basic policies many researcher enhanced the replacement policies for next generation computing era. For our analysis we study basic policies as well as some enhanced policies.

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. Dynamic Insertion Policy (DIP) is a combination of two different replacement policies one is LRU and other is Bimodal Insertion Policy (BIP). BIP frequently places the incoming line in the Most Recently Used (MRU) position. Dynamic insertion policy selects the traditional LRU policy and BIP depending upon which policy has less number of miss. DIP requires runtime estimation of misses occurred by both the competing policies. For selection of policy, DIP uses the Policy Selector (PSEL), a saturated counter which keeps the hit or miss information [8][11].

Another factor which can also affect the performance of cache memory is locality of reference. The principle of locality of reference is a phenomenon describing the same value or related storage location being frequently accessed. Locality of reference assists the cache replacement polices there are two type of localities first is spatial and second is temporal. In spatial locality nearby memory locations are accessed frequently. In temporal locality same memory location is referenced frequently. In this work we also compare hardware complexity of the policies shown below table 1.1.

Hardware Complexity of Cache Replacement Policies

Replacement policy	Storage Requirement (bits)	Action on Hit	Action on Miss
Random	Log_2(Way)	None	Update the Linear Feedback Shift Register
LRU	No. of set × Way × Log_2 (Way)	Update the LRU Stack	Update the LRU Stack

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 2, February 2018

FIFO	No. of set × Log_2(Way)	None	Update the FIFO Counter
DIP	No. of Way	Update PSEL counter	Update PSEL counter
PLRU	No. of Set × (way – 1)	Increment the TREE counter	Decrement the TREE counter

Table 1.1 Hardware complexity of Replacement Policy

III. REVIEW OF LITRATURE

Our Review is based on various cache replacement policies and performance issues in multicore processors. Here, we present a brief review of the related work.

Hussein Al-Zoubi et al [7] explored the performance of cache on the basis of replacement policies such as LRU, FIFO and Random. They found that the LRU policy in the data cache has better performance than FIFO and Random.

According to Gheith A bandah et al [11] LRU Policy has been the standard replacement policy used for caches (L1, L2 caches). All basic policies such as FIFO, Random etc. are compared with the LRU and sometimes with other proposed replacement policies.

James E. Smith et al [13] presented the instruction cache replacement policies. They proposed a new loop model. In this loop model, they found that random replacement has performed better than LRU and FIFO. However, each simulation has different cache sizes, different cache associativity and different benchmarks. Therefore, the performance comparisons of policies are less accurate. A unified simulation should be implemented for all policies to compare their performance.

IV. PROBLEM OF MATRIX MULTIPILCATION

In this paper we have taken the matrix multiplication. Matrix multiplication used to solve various problems in computer science like pattern recognizes, image processing and many scientific computations. It takes order of n3 time to compute n * n matrix. Here we present the block of multiplication code.
Ans[N][M] = A[N][M]*B[N][M]

Where N is Number of rows and M is Number of Columns.

```
        for (i=0;i<N; i++)
            {
              for (j=0; j<M; j++ )
                  {    Ans[0][0]=0;
                    for (k=0; k < N ; k++)
    {
       Ans[i][j] = Ans[i][j] + A[i][k] * B[k][j];
}}}
```

This code have temporal locality as index variable i, j, k and spatial locality as next element will be fetched i.e. an array.

V. PERFORMANCE METRIC

For measuring the performance of cache memory we use hit ratio and miss ratio.

Hit Ratio and Miss Ratio

Hit Ratio denoted by H is defined as the ratio of the total number of hits and total no. of hits and misses.

$$H = \frac{Total\ No.of\ Hits}{Total\ No.of\ Hits + Total\ No.of\ Misses}$$

The cache hit ratio H should be almost one. s Miss Ratio is denoted by M is defined as

$$M = 1 - Hit\ Ratio$$

For measuring the performance of multi core processor we use Instruction Per Cycle (IPC) and Cycle Per Instruction (CPI).

IPC is number of instructions are executed in one cycle.

IPC= 1/(Number of Instruction)

CPI is number of cycles are needed to execute one instruction.

EXPRIMENTAL EVALUATION

In this paper we have simulated various cache replacement policies such as LRU, FIFO, RANDOM, PLRU and DIP with the help of Simple Scalar trace driven simulator. The Simple Scalar is an open source trace driven simulation tool set for computer architecture. The Simple Scalar tool Set performs fast, flexible, and accurate simulations of a modern microprocessors [12]. This tool runs on Linux operating system, bind with GCC or FORTAN compiler and make a cross platform for binary file.

To evaluate the performance of L1 cache on different cache replacement policies, we have implemented our experiment on quad core intel i3 (4 core) processor with

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 2, February 2018

independent L1 instruction cache and L1 data cache, L2 shared cache with 2-way associative. After environment setup we give the input of binary code of matrix multiplication.

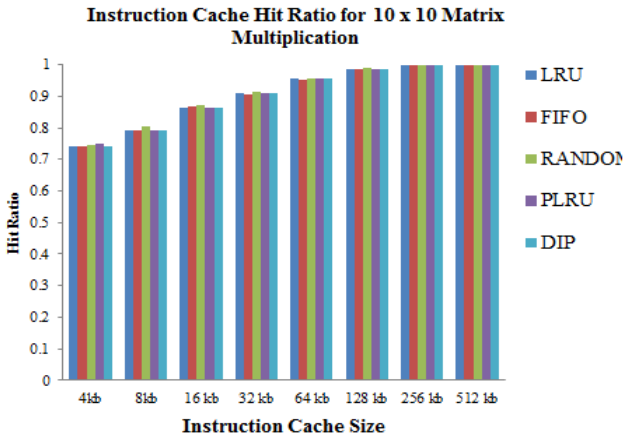
Table 1.2 Simulation Configurations

L1 Instruction Cache Size (KB)	4, 8, 16, 32, 64, 128, 256, 512
L1 I Data Cache Size (KB)	4, 8, 16 , 32, 64, 128, 256, 512
L2 Unified cache	1 MB fixed
Replacement Policies	LRU, FIFO, RANDOM, PLRU, DIP
Matrix size (float data type)	10 x 10, 100 x 100, 500 x 500

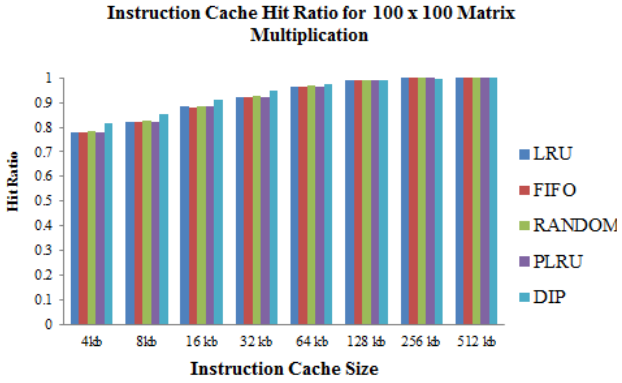
Experiment Results

After the implementation of above configuration we measures Hit ratio, Miss ratio, IPC, CPI and completion time. Result graphs are shown below:

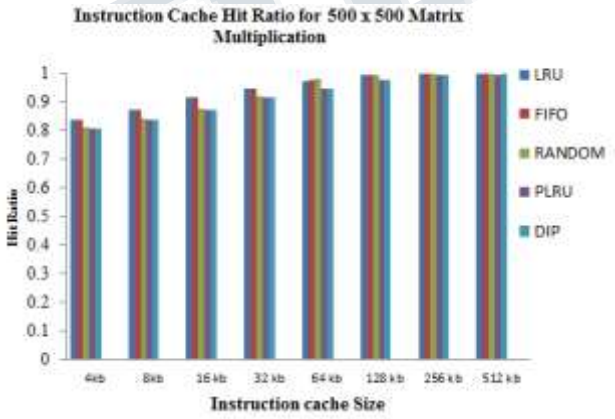
For problem of 10 x10, 100x 100, 500 x 500 matrix hit ratio graph is:



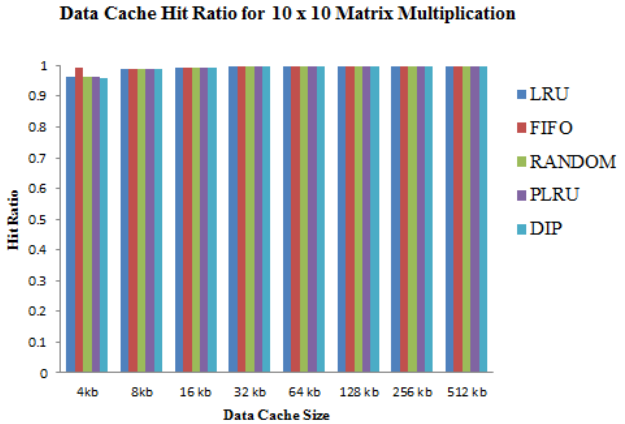
Graph 1.1 Instruction Cache Hit Ratio for 10 x 10 Matrix Multiplication



Graph 1.2 Instruction Cache Hit Ratio for 100 x 100 Matrix Multiplication



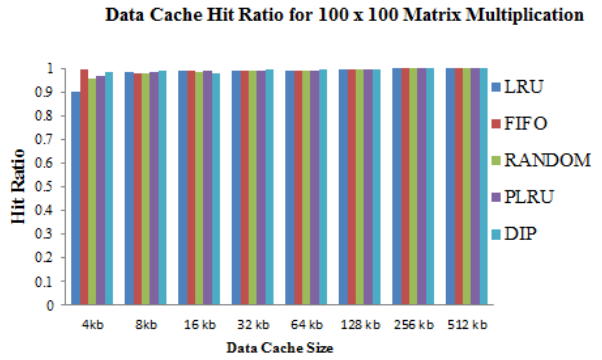
Graph 1.3 Instruction Cache Hit Ratio for 500 x 500 Matrix Multiplication



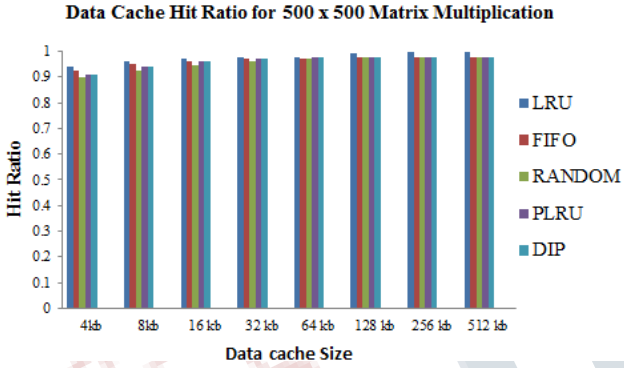
Graph 1.4 Data Cache Hit Ratio for 10 x 10 Matrix Multiplication

ISSN (Online) 2394-2320

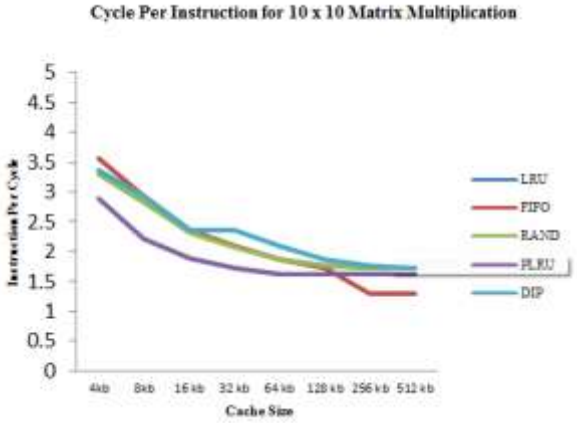
International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 2, February 2018



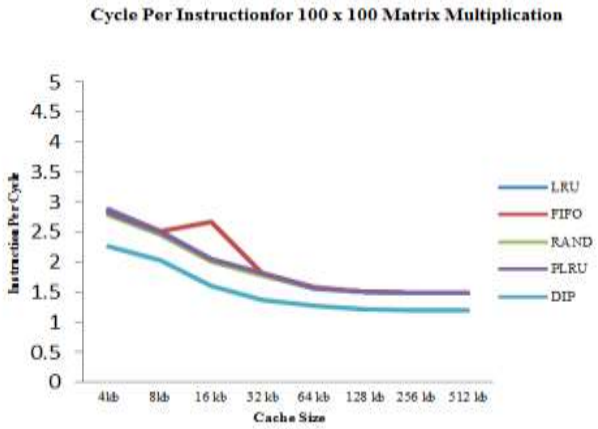
Graph 1.5 Data Cache Hit Ratio for 100 x 100 Matrix Multiplication



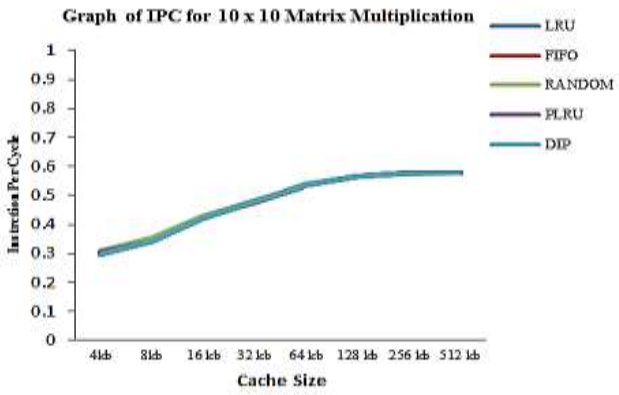
Graph 1.6 Data Cache Hit Ratio for 500 x 500 Matrix Multiplication



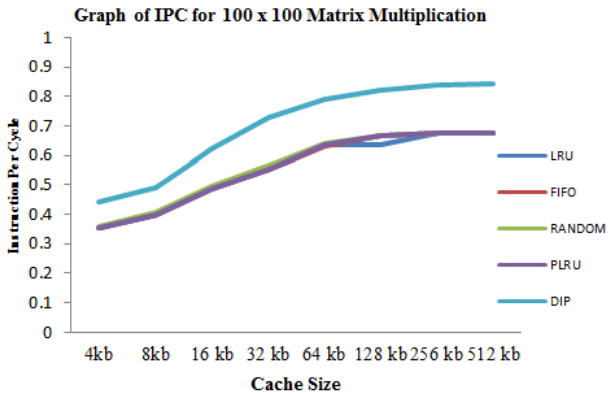
Graph 1.7 of Cycle Per Instruction for 10 x 10 Matrix Multiplication



Graph1.8 for Cycle Per Instruction of 100 x 100 Matrix Multiplication



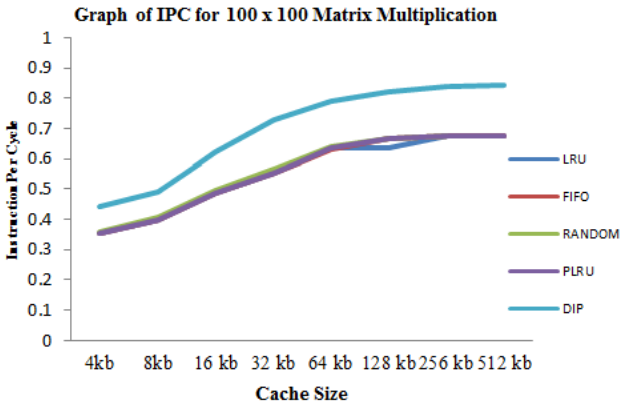
Graph 1.9 for Cycle Per Instruction of 500 x 500 Matrix Multiplication.



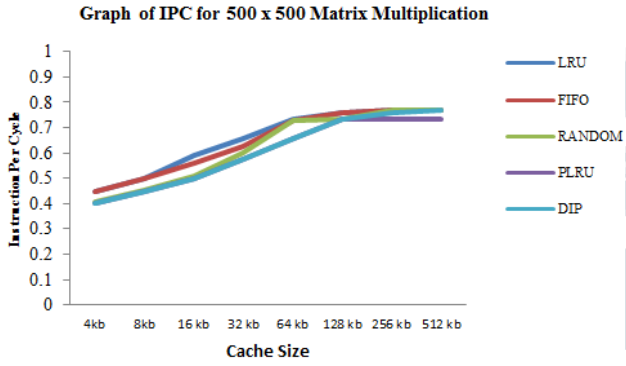
Graph 1.10 Instructions Per Cycle for 10 x 10 Matrix Multiplication

ISSN (Online) 2394-2320

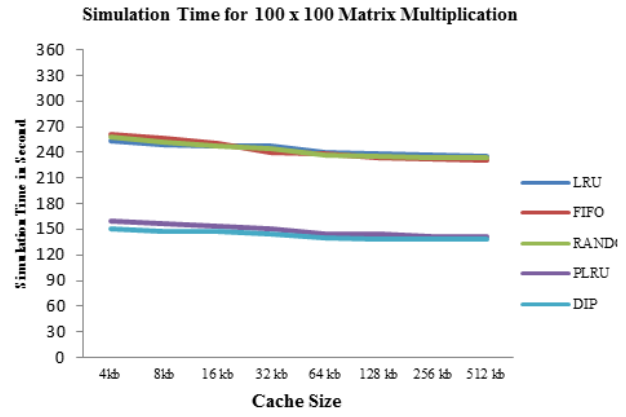
International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 2, February 2018



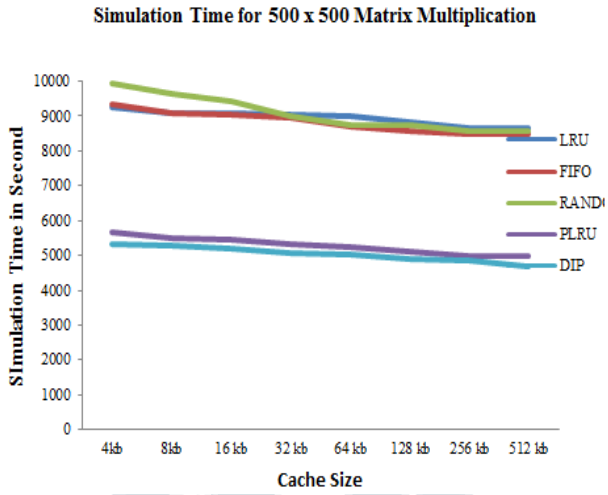
Graph 1.11 Instructions Per Cycle for 100 x 100 Matrix Multiplication



Graph 1.12 Instructions Per Cycle for 500 x 500 Matrix Multiplication Here we present simulation time graph for 100 x 100 and 500 x 500 with different cache size.



Graph 1.13 Simulation Time of 100 x 100 Matrix Multiplication.



Graph1.14 Simulation Time of 500 x 500 Matrix Multiplication.

B) Result Analysis

Graph 1.1, 1.2, 1.3 presents the hit ratio of instruction cache. From these graphs we observed that when Instruction cache size increases, for the 10 x 10 problem RANDOM replacement policy gives better hit ratio. For the 100 x 100 problem DIP replacement policy gives better hit rate. For the 500 x 500 problem LRU replacement policy gives better hit rate. Graph 1.4, 1.5, 1.6 presents the hit ratio of Data cache. From these graphs we observed that when Data cache size increases, for the 10 x 10, 100 x 100 problems FIFO and DIP replacement policies gives better hit ratio. For the 500 x 500 problem LRU replacement policy gives better hit rate.

Graph 1.7, 1.8, 1.9 presents the CPI from these graphs we observed that for the 10 x 10 and 100 x 100 problems PLRU and DIP replacement policies CPU takes less number of cycles. For the 500 x 500 problem LRU replacement policy CPU takes less number of cycles.
Graph 1.10, 1.11, 1.12 presents the IPC, from these graphs we observed that for the 10 x 10 and 100 x 100 problems with DIP replacement policies gives better result. For the 500 x 500 problem LRU replacement policy gives better result.
Graph 1.13, 1.14 presents the execution time, from these graphs we observed that for the 100 x 100 and 500 x 500 problem DIP replacement policies gives better result as compared with other replacement policies.

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 2, February 2018

CONCLUSION

In this paper, we focused on impact of replacement policies of first level split cache memory in multi core (quad core) processor. From our simulation we observed that the larger cache size improves cache performance by taking advantage of spatial locality. From the result analysis we conclude that when the matrix multiplication problem size is smaller, than all replacement policies perform better. In Some simulation LRU policy performs better but due to hardware complexity it takes large time to compute the problem. For the large matrix multiplication problem DIP replacement policy takes very long time to compute. In our future work we will test other replacement policies with CPU intensive problems and to enhanced cache replacement policy.

REFERENCES

[1] Alan Jay Smith, "Cache memory", Computing Surveys, ACM 0010-4892/82/0900-0473, Vol. 14, No. 3, September 1982, pp 473-531.
[2] 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.
[3] M. D. Hill and A. J. Smith, "Evaluating associativity in CPU caches", IEEE Transactions on Computers, Vol. 38, No. 12, December 1989, pp 1612-1630.
[4] Zhenlin Wang, "Cooperative Hardware/Software Caching for Next-Generation Memory Systems", Thesis (Ph.D), Department of Computer Science, University of Massachusetts at Amherst, February 2004, pp 31.
[5] John P.Hayes, "Computer Architecture and organization", Third Edition Tata McGraw Hill, ISBN: 0-7-0027355-3, 1998, pp 451-452.
[6] William Stalling, "Computer Organisation and Architecture", Seventh Edition Pearson Education, ISBN: 978-81-7758993-1, 2005, pp 30-31.
[7] 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.
[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] 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.
[10] 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.
[11] Gheith Abandah, "A Study on Cache Replacement Policies", http://www.abandah.com/gheith/Courses/CPE731_F09/Research_Projects/3_Report.pdf, University of Jordan, retrieved as on August 2017.
[12] Doug Burger, Todd M. Austin, "The SimpleScalar Tool Set, Version 2.0", http://www.simplescalar.com, retrieved as on April 2017.
[13] 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.