ISSN No. 0976-5697

Volume 8, No. 3, March – April 2017

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Enhanced Approach for Cache Behaviour and Performance Evaluation

Manish Kumar Verma
Dept. of Computer Science and Engg
MMM University of Technology
Gorakhpur, Uttar Pradesh, India

Dr.P.K.Singh
Dept. of Computer Science and Engg
MMM University of Technology
Gorakhpur, Uttar Pradesh, India

Abstract: In previous research on cache replacement policies, have seen that a block usually becomes dead after some uses. By classifying the dead block and discarding them early, there are various policies have been used but still there are number of problems such as cache pollution, high cache miss rate and cache overhead existed which degrades the system performance. In this paper, a new approach for cache replacement policy has introduced which reduces the cache miss rate and cache overhead by using shared L2 level cache. In terms of less miss rate and good hit rate there is need to deploy effective replacement policies. Cache replacement techniques is one of the crucial design parameter that affects the overall processor performance and also become more important with current growing technological moves in the direction of high associative cache .

Keywords: Cache pollution; Cache overhead; Cache miss rate; Cache replacement policy (LRU and LFU); Memory overhead;

I. INTRODUCTION

The aim of this paper work is to enhance the performance of the system through introducing a new cache replacement policy which reduces the miss rate and also memory overhead. Now after analysis the various cases of memory references with new cache replacement policy we found that performance of the processor is very much dependent on number of hits. Cache memory have been introduced to improve the performance and cost of the system in a computer architecture.to improve the performance of a Cache memory in terms of hit ratio and better response time, system needs to employ efficient Cache replacement policy and all replacement of blocks has been occurred in Cache memory. Cache is a high speed memory contains most recently accessed pieces of main memory. The structure of the paper is as follows: In Section1 we introduce the theory of cache memory. In Section2 we present our proposed work with the steps which is followed by us to achieve the objective. In section 3 we present the case study and analyse the performance of calculated method with respect to the conventional method. In section4 we conclude the paper. And at last we go for various references which I gone through. Cache memory bridges the gap between CPU and main memory. Increasing Cache Size gives the better performance but it is very costly. It is necessary because, the time it takes to bring an instruction into the processor is very long when compared with the time to execute the instruction. Cache memory helps to decrease the time it takes to move information to and from processor. Cache memory improves the system performance by following a concept of locality of reference. Localities of reference is defined as with reference to the hierarchy design, CPU performs the read and write operations only on a reusable data space (Cache memory).locality of reference categorized into two parts-

a) Temporal locality
b) Spatial locality

Temporal locality-The meaning of temporal locality is defined as the same word in the same block is referenced by the CPU in a near future.

Spatial locality- The meaning of spatial locality is defined as the adjacent words in the same block is referenced by the CPU in a sequence.

Whenever a new block from main memory is brought into the cache memory, and that time cache is full then it is necessary to replace the existing blocks of cache memory. For this approach we require cache replacement policies. To provide instruction and data which is reside in main memory to the processor at the high speed it can process them is one of the important challenging aspect. For achieving high speed processing, a better cache replacement policy must be implemented. A number of policies have been introduced. For maximum hit rate a good cache replacement algorithm must have characteristics such as,

- Less memory overhead.
- Faster access to data.
- Less response time.

Cache memory performance is evaluated on the basis of hit rate, miss rate, miss penalty, and average memory access time. Miss Rate is defined as the part of memory accesses that are not present in the cache while hit rate is the part of memory access that is present in the cache. Miss Penalty is defined as the total number of cycles CPU is stalled for a memory access determined by the sum of Cycles (time) to replace a block in the cache, upper level and Cycles (time) to deliver the block to the processor.

The memory design in the uniprocessor system was based on easier component, made up of some levels of cache memory, which provide data and instruction to single processor. But the processors with multicore system [4], caches are just a component of memory design; here the other parts involve consistency model, cache coherence support and interconnection within same chip. In multicore processors every core of a processor consists of every components of an individual processor such that registers, ALU, pipeline hardware, control unit and L1 data caches and instruction. In addition to several numbers of cores, multicore chips also involve L2 and sometimes L3 cache also [2]. There are basically three parameters in multicore organization are the number core on the chip, levels of cache memory and amount of the shared cache memory. Fast development of semiconductor technology makes it possible that the embedded system can gain the features of high-performance, miniaturization, and diversification. To enhance the inter - processor communication speed and the system performance, Chip Multi-processor architecture is generally used nowadays.

Every processor uses dedicated L1 cache for its own cache and L2 cache for shared cache memory [3] in on-chip multiprocessor architecture as given in Fig. 2. Sharing Cache

L2 not only provides prevention of the copy of hardware resource but also provide high utilization. If processor unable to access to its private cache L1 and shared cache L2 is used for accessing the external memory. As there is increase in access time which results in degradation of system performance. In chip multiprocessor architecture, cache pollution has negative influence on cache performance. It also caused by bad prediction of any processor which increment the cache miss rate due to cache pollution problem ,several cache replacement policies such as random, FIFO,LFU and LRU has been used [3].Random replacement policy replaces the cache items randomly and FIFO replaces which was first loaded. They remove despite the item used frequently LRU replaces the shortcoming.

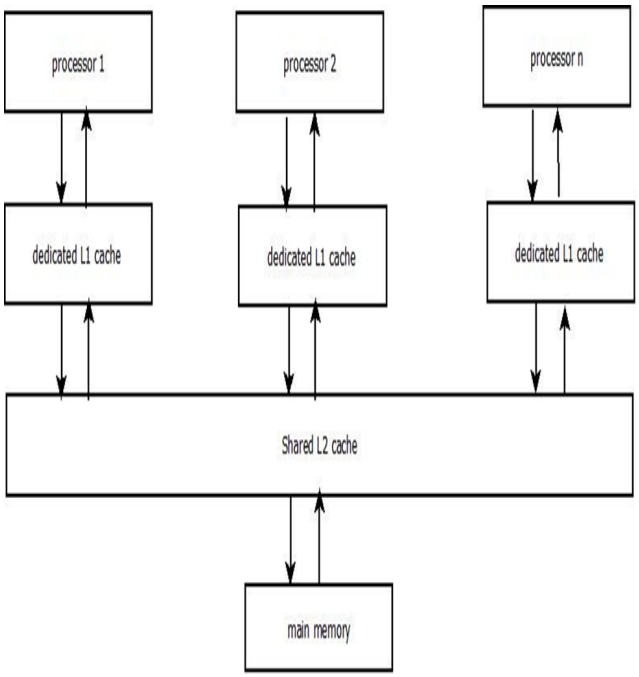

Fig1. Structure of on chip multiprocessor with L1, L2 cache and external memory [1]

II. PROPOSED WORK

In this work our contribution is to increase the hit rate, decreases the cache pollution and overhead of the memory with making some changes in the existing model which is LRU (least recently used). We have proposed a cache replacement policy to minimize the cache miss rate in shared L2 cache is described. In order to reduce the speed-up gap between CPU (central processing unit) speed and main memory performance, in current generation computers, memory hierarchy implemented by keeping registers inside, cache is on the chip and off the chip and virtual memory on the hard disk. With reference to the hierarchy design CPU performs the read and writes operation only in the cache memory. In this process CPU checks the availability of the data block in the cache. The performance is balanced in computer architecture via cache memory. In terms of less miss rate and good hit rate there is need to deploy effective replacement policies. . Cache replacement techniques is one of the crucial design parameter that affects the overall processor performance and also become more important with current growing technological moves in the direction of high associative cache .There are various cache replacement policies such as first in first out(FIFO),least recently used(LRU) [5],least frequently used(LFU),optimal page replacement in all these policies LRU is best in terms of more hit rate and less miss rate. Others are degrading the performance of the system. But still there are more challenges present in LRU policy that's why we are going to implement a new policy which is modified version of existing policy such as LRU. For this new approach we have follow some important steps for which it will clear the how the procedure going on. These steps such as:

A. Steps of Proposed Replacement Policy

- Firstly we take frame table which is limited in size.
- For counting the number of times a particular block of table is reference and store the counting value a counter is associated with each block.
- As we know initially cache misses occur and if cache is empty then we place new block at the top of the table, simultaneously we increment the counter value by one which is associated with that block.
- If cache is not empty then we calculate the threshold value by using the below formula threshold value (Thv) = $\lceil sum\ of\ counter\ value / stack\ size \rceil$
- In the next step we continue the above step 3 and step 4 till the table size.
    i. If counter value is greater or equal to the calculated threshold value (Thv) then we do nothing.
    ii. Otherwise we take a temporary table and transfer the label of blocks which has counter value less than threshold value (Thv) and apply LRU only on the temporary table and replace the existing reference with new coming references.
- After that remove the labels from temporary table and finally update cache memory with counter value.

Now on the basis of the above steps, we can draw an algorithm which we follow for our proposed policy that is shown below:

B. Algorithm of Proposed Policy

Algorithm (ELRU)
If (cache Hit)
{
Increment the counter value corresponding reference set and sort the cache block on the basis of counter value.
}
Else if (cache miss)
{
If (cache is empty)
{
Place the new block at the top of stack and increment the counter value by one.
}
Else
{
Calculate the threshold value (Thv) = $\lceil sum\ of\ counter\ value / stack\ size \rceil$
For (i=0 to stack size)
{
If (counter value >=Thv)

Do nothing
Else
{
Take a T.T and transfer the label of block which has less counter value than Thv and apply LRU policy only on T.T and replace existing reference with new incoming reference.
}
At last remove the labels of T.T and finally update cache memory with counter value.
}
}

To implement our proposed cache replacement policy we have to use java programming language which is also known as high level language and its platform environment initially released by Sun Microsystems. Further to implementing our proposed method platform to be used such as NetBeans which is well known software development platform. Further to implementing our proposed method firstly we have calculated the performance of our approach by generating a random reference string using Rand Function, between definite ranges.

Case 1:

fx = Random between (1,20)

17,9,14,16,12,4,13,1,4,11,4,5,5,11,8,9,5,1,13,6,11,14,6,7,12,16,11,8,2,3,13,2,12,10,18,16,14,19,7,11,3,2,7,8,14,2,16,6,11,12,16,17,1,18,17,4,15,11,1,5,14,16,18,10,10,16,13,3,8,17,7,7,4,2,9,17,12,10,16,10,16,9,14,9,11,7,17,17,12,15,11,19,13,13,4,15,17,13,15,18,9,13,14,15,19,11,11,6,14,6,19,15,7,19,15,19,16,13,1,14,7,19,6,7,9,14,19,8,11,17,18,4,16,9,7,9,11,10,3,15,17,19,19,16,6,5,8,9,15,19,12,10,16,1,4,4,14,8,8,9,17,9,5,4,19,18,3,13,13,6,13,16,19,5,2,5,11,18,19,5,7,1,7,8,12,6,17,11,4,14,1,13,5,18,14,6,13,15,9,11

Using the above method we generate the various random numbers of sizes 200 in terms on number of sets which is shown in the table. The table shows the evaluation of the number of hits in the existing policy which is compared with our proposed methodology corresponding to the particular sets.

Table. I *Comparison between LRU replacement policy and proposed replacement policy with set size 200*

Set no.	***Number of hits***	
	LRU cache replacement policy	***Proposed cache replacement policy***
Set 1	37	39
Set 2	41	43
Set 3	47	54
Set 4	43	53
Set 5	48	50
Set 6	38	38
Set 7	43	43
Set 8	45	50
Set 9	39	42
Set 10	34	42

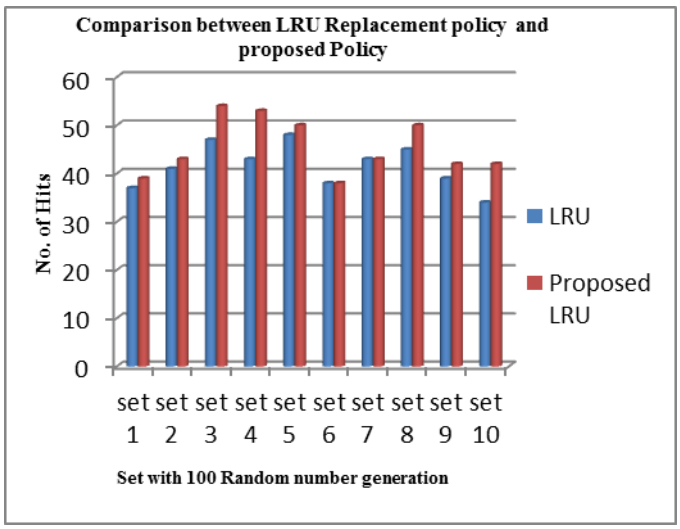

Fig 2.Graphical representation of the above given data sets in the table

Case 2:

Now we show that how the number of hit increases when we are going to increase the size of random number and reduces the range of random number as 1 to 20. Memory references are given as:

fx = Random between (1,20)

2,7,15,15,9,13,19,13,13,6,11,17,16,8,3,14,7,10,13,17,15,9,10,4,18,14,7,10,13,17,15,9,10,4,18,14,8,2,2,6,5,5,1,15,12,2,13,9,16,11,14,13,15,5,13,7,3,4,16,6,16,3,2,12,9,16,3,1,6,5,7,7,6,19,6,6,7,15,3,9,16,7,12,3,12,17,10,12,2,2,6,12,15,6,10,17,14,5,10,6,4,4,6,1,1,17,17,6,15,12,18,16,4,10,2,3,11,19,19,4,3,3,8,19,7,1,16,11,9,13,1,12,2,19,18,14,17,16,8,3,3,19,6,3,9,14,8,13,16,1,5,9,14,17,6,18,2,14,4,9,12,4,4,4,10,18,11,6,19,16,3,13,14,9,1,15,1,17,6,1,2,16,6,13,11,2,1,12,2,3,18,19,13,6,16,12,16,1,13,6,2,10,11,1,16,13,11,10,13,10,4,7,7,3,11,19,12,9,7,17,10,10,10,13,7,10,11,18,4,12,19,18,13,11,7,17,6,9,11,2,15,14,11,7,4,14,14,16,3,19,11,10,15,3,12,10,14,1,17,9,9,19,2,3,11,6,14,10,16,3,12,1,14,12,10,14,11,1,15,11,15,8,4,15,19,11,1,16,7,8,1,5,8,11,6,7,13,7,17,18,7,11,10,2,15,9,4,7,17,11,15,19,11,15,3,8,1,9,2,16,4,6,4,19,5,18,3,12,5,11,1,5,18,18,1,18,7,13,7,1,18,1,18,18,3,18,7,10,4,13,4,19,8,2,4,6,13,5,14,7,4,11,13,3,2,19,7,1,5,6,5,8,5,13,1,16,13,18,7,7,9,10,9,1,10,4,9,16,2,6,6,8,13,10,13,19,19,13,2

Using the above method we generate the various random numbers of sizes 400 in terms on number of sets which is shown in the table. The table shows the evaluation of the number of hits in the existing policy which is compared with our proposed methodology corresponding to the particular sets.

Table. II *Comparison between LRU replacement policy and proposed replacement policy with set size 400*

Set no.	*Number of hits*	
	LRU cache replacement policy	*Proposed cache replacement policy*
Set 1	89	89
Set 2	80	94
Set 3	87	88
Set 4	82	89
Set 5	74	82
Set 6	82	84
Set 7	80	81
Set 8	81	81
Set 9	86	95
Set 10	81	84

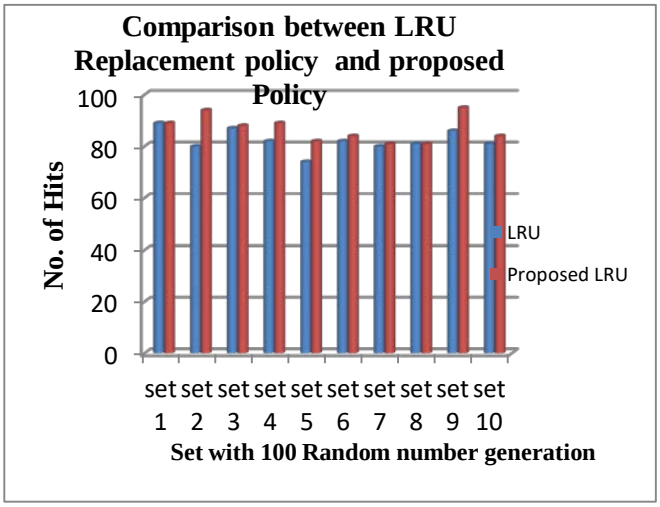

Fig 3.Graphical representation of the above given data sets in the table

III. CONCLUSION

As many concept were given in this topic which will helpful for us in the field of cache optimization. From the above discussion and based on the experimental results we can conclude that our proposed method (proposed LRU) has better performance than LRU. And we can say that proposed LRU the will be used for the future reference and give contribution in the field of cache performance.

IV. ACKNOWLEDGMENT

The satisfaction that accompanies that the successful completion of any task would be incomplete without the mention of people whose ceaseless cooperation made it possible, whose constant guidance and encouragement crown all efforts with success. I am grateful to my guide Dr. P.K.SINGH for the guidance, inspiration and constructive suggestions that helped me in the preparation of this progress report. I wish to thank my parents who have been always been a source of inspiration for their never-ending support and love throughout completion of the report and I am also thankful my friends who have helped in successful completion of the partial report.

V. REFERENCES

[1]. W. Stallings, "Computer Organization and Architecture," Eighth Edition, 2011.

[2]. S. Kumar, P. K. Singh, “An Overview of Hardware Based Cache Optimization Techniques,” International Journal of Advance Research in Science and Engg, vol. no. 4, Special Issue (01), September 2015.

[3]. Fu, John WC, and Janak H. Patel. "Data prefetching in multiprocessor vector cache memories." ACM SIGARCH Computer Architecture News, Vol. 19. No. 3. ACM, 1991.

[4]. H Khatoon, S. H. Mirza, T. Altaf, “Aware Cache Optimization Techniques for Multi Core Processors”, 2011 Frontiers of Information Technology.

[5]. Cantin J. F, Hill M. D., Cache Performance of theSPECCPU2000Benchmarks,http://www.cs.wisc.edu/ multifacet/misc/spec2000 cachedata.