International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 05 Issue: 09 | Sep 2018 www.irjet.net p-ISSN: 2395-0072

NRU Cache Replacement Policy for Multicore Processors using Multi2Sim

Vikram Bharadwaj[1], Vineet Kachapur[2]

[1,2]*Student, Dept. of Computer Science and Engineering, Sir M Visvesvaraya Institute of Technology, Bangalore, India*

Abstract:- *Cache replacement policy is a major design parameter of any memory hierarchy. The efficiency of the replacement policy affects both the hit rate and the access latency of a cache system. The higher the associativity of the cache, the more vital the replacement policy becomes. The memory system does not consist of a single cache, but a hierarchy of caches. If each cache in the hierarchy decides which block to replace in an independent manner, the performance of the whole memory hierarchy will suffer, especially if inclusion is to be maintained. For example, if the level 2 cache decides to evict an LRU(Least Recently Used) block, one or more blocks at level 1, possibly not LRU, can be evicted, therefore affecting the hit rate of level 1 and the overall performance. In this Project we study the comparative Analysis of cache behavior by varying the cache policy for L1 and L2 cache. We see a detailed analysis about the Hit Rate and Miss Rate together as the Hit Ratio by varying the Associativity of the Cache, the number of Sets in the Cache and also the number of Cores and Threads in the simulated CPU.*

Key Words: **Cache**, **Replacement Policy**, **LRU** (**Least Recently Used**), **Memory**, **Hierarchy**, **Hit Rate**, **Miss Rate**, **Associativity**.

1. INTRODUCTION

A multicore processor is a single computing component with more than one central processing units (called cores), which read and execute program instructions. In the single core processor, a series of requests for memory locations is send to the cache, where each request appears after the last one has been served. The time by the cache to serve the request depends on whether the memory location is present in the cache (a cache hit) or not (a miss occurs and the location is fetched from the main memory. If the cache is full, the memory location already in the cache gets evicted to make space for the new memory location. Which memory location (or block) is to be evicted is decided using the cache replacement policy, mainly LRU. We have compiled a set of results by running 2 benchmark suites: PARSEC 3.0 and Splash 2. We are using Multi2Sim, a CPU simulator, to emulate our specific requirements without actually physically implementing them. Multi2sim gives us the detailed simulation report on running a benchmark which includes the simulation time, real time, Instruction per Cycle (IPC), Cycles, Hits and Misses.

1.1 Existing System

Local replacement policies keep track of blocks behaviour within each cache set in order to be able to choose a victim block when a replacement is needed. The replacement policy gets its own information from accesses to its cache. However, a replacement policy P_i at cache L_i can indirectly affect the behaviour of the replacement policy P_j at another cache L_j through a cache miss. This is because p_i affects the number of cache misses in L_i and whether the victim block is dirty or not. The problem is that both of these factors also affect the replacement policy behaviour of the cache below L_i in the hierarchy. Therefore, the communication between the replacement policies of two caches in the hierarchy is established only after a cache miss. This limited connection between caches in terms of replacement policies may lead to much inefficiency.

The current system will give an idea of how the normal scheduling algorithms work and their efficiencies with respect to the system in which it is been put into.

1.2 Proposed System

The proposed system is a Not Recently Used (NRU) algorithm. The NRU replacement policy favours keeping pages in the memory that have been recently used. This algorithm works on the following basic principle: When a page is referenced, a referenced bit is set. Similarly, when a page is modified (written to), a modified bit is set. NRU Cache Replacement Policy is been written and been integrated with the simulator and Benchmarks are executed and analysed against the results from the default LRU/FIFO cache replacement policies.

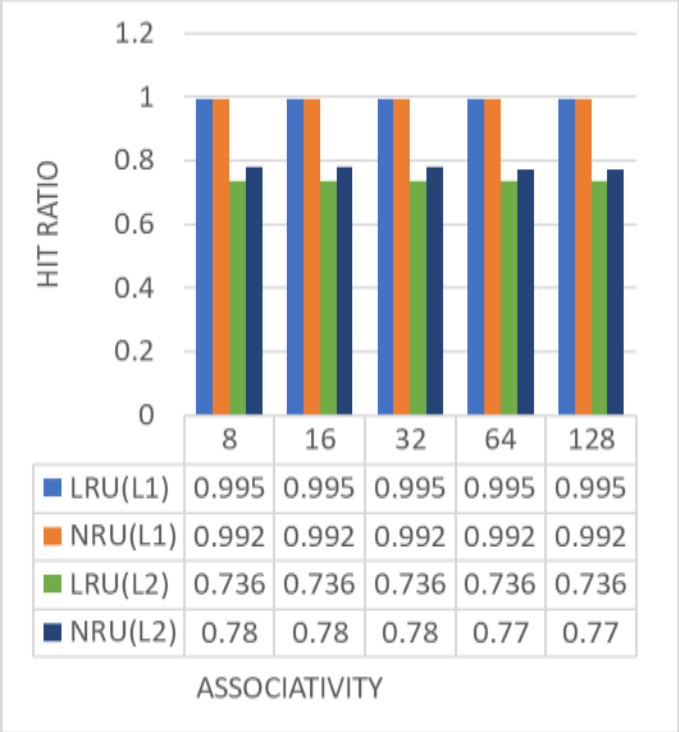
2. IMPLEMENTATION

Implementing a new Replacement Policy in Multi2Sim is pretty straightforward, but at the same time is a bit tedious too. Since we have to make changes to the "Cache.cc" file, we will be adding "NRU" as the new element in the enum. The algorithm is added and necessary changes to the code are done, and Multi2Sim is compiled again, once successfully compiled, we run the "make" on Multi2Sim.

The application will start without any errors and we are now ready to run the benchmarks and test the results against the already existing Replacement Policies.

For the first setup, we will Change the replacement policy in L1private cache and L2 shared cache to default LRU, and

compare it against the implemented NRU policy. Here, the parameter under study is he Hit Ratio of shared L2 cache vs varying associativity of the L2 cache. The number of Sets is kept a constant for the L2 Cache at 512, for varying Associativity, and the benchmarks used are PARSEC 3.0and Splpash2. It is observed that having NRU in both L1 private cache and L2 shared cache produces consistent better Hit Rates when compared to the default LRU, as depicted in the graph below (**Chart -1**).



	8	16	32	64	128
LRU(L1)	0.995	0.995	0.995	0.995	0.995
NRU(L1)	0.992	0.992	0.992	0.992	0.992
LRU(L2)	0.736	0.736	0.736	0.736	0.736
NRU(L2)	0.78	0.78	0.78	0.77	0.77

Chart -1: Hit Ratio vs Associativity for L1 and L2 Cache (LRU vs. NRU).

For the next observation, we compare all the default replacement policies, namely LRU, FIFO and RANDOM with NRU, for Instructions per Cycle (IPC) and the Hit Rate. In computer architecture, Instructions per Cycle (IPC) is one aspect of a processor's performance: the average number of instructions executed for each clock cycle. The graphs presented below (**Fig -1**) are plotted by executing the benchmarks Cholesky and Barnes For the most part, in both benchmark programs that are tested, IPC for FIFO and Random always remain consistently less than LRU and NRU. Thus, the two main policies that are to be considered are LRU and NRU. As we can see from the graphs obtained above, NRU policy outperforms LRU in both the benchmarks with a higher IPC for all tested Associativity values (8-128).

Thus, we can say, NRU performs better for these set of benchmarks than LRU, FIFO and Random, from the above obtained graphs.

The performance of cache memory is frequently measured in terms of a quantity called hit ratio. The CPU refers to memory & finds the word in cache, it is said to produce a hit. If the word is not found in cache, it is in main memory and it counts as miss. Thus, the Hit Ratio of a Cache is given by:

Hit Ratio = (Number of Hits) / (Total Number of Accesses to that Cache)

The 2 graphs presented below (**Fig -1**) are plotted by executing the benchmarks Swaptions and Black Scholes. The replacement policies LRU, FIFO and Random show similar results for both the benchmarks tested. All 3 of them have respective constant values for the most part while running the second benchmark (Blackscholes), while NRU outperforms all the 3. Although the values have varied during the execution of Swaptions, it is consistently lesser than the Hit Ratios offered by NRU.

Thus, we can say, NRU performs better for these set of benchmarks than LRU, FIFO and Random, from the above obtained graphs.

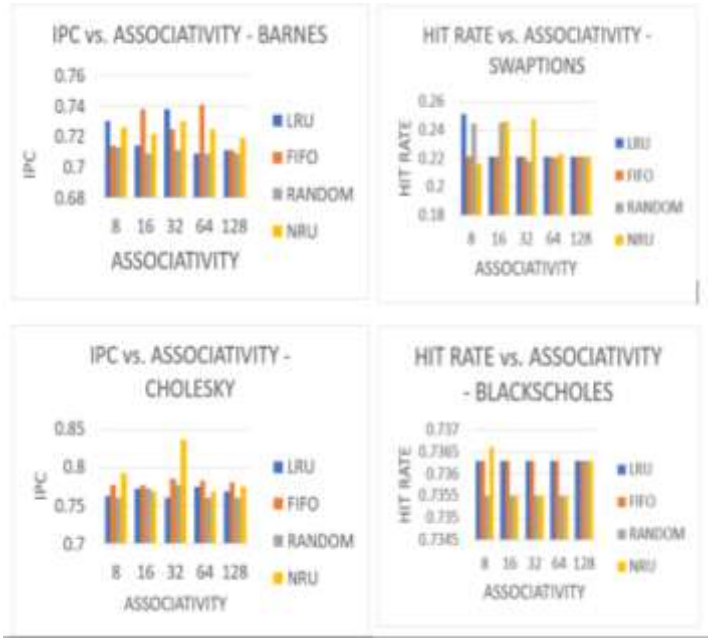


Fig -1: IPC/Hit Rate vs. Associativity for different benchmarks, for LRU, FIFO, NRU and RANDOM replacement policies.

From the following observations, it is clear that the proposed algorithm/policy, Not Recently Used (NRU), outperforms the default replacement policies when we compare the IPCs and the Hit Rates, the two critical parameters required to measure the performance of a replacement policy.

3. CONCLUSION

We have proposed an algorithm, NRU, which can outperform LRU replacement policy for most of the benchmarks programs which is simulated using Multi2sim simulator. The results we got were better than that of LRU policy. According to our results, Random performs worst in multicore processor and IPC remains constant for all values of associativity if number of sets is kept constant. FIFO on the other hand does not fall in a specific category. In fact, it is almost same as that of LRU at times or even is almost same as Random in some of the benchmarks. NRU has proved to be better than LRU, which is the most reliable and the best shared cache replacement policy, and also overcomes some

of the drawbacks of LRU such as, NRU checks whether the block is a "dirty" block or not. We hope, in future the project does reach its intended audience and bring a fresher perspective in this domain.

ACKNOWLEDGEMENT

We make a humble attempt to thank by expressing in words our profound sense of gratitude to those who have helped us. We express out immense gratitude to **Prof K R Kini**, Principal, Sir M Visvesvaraya Institute of Technology.

We wish to thank **Prof Dilip K Sen**, HOD of Department of Computer Science and Engineering.

We owe deep sense of gratitude to **Mrs. Sowjanya Lakshmi A**, Assistant Professor, Department of Computer Science and Engineering.

We dutifully express our thanks to **Mrs.Sheela Katavathe**, Associate Professor, Department of Computer Science and Engineering.

REFERENCES

[1] Suraj Sharma, Babu Lal "An Effective Cache Replacement Policy for Multicore Processors". NIT Rourkela for CSE Department, May 2013.
ethesis.nitrkl.ac.in/5526/1/212CS1087-14.pdf

[2] InformIT – The Not Recently Used Replacement Policy.

[3] W. Wong and J.-L. Baer, "Modified LRU policies for improving second level cache behavior, "in Sixth International Symposium on High-Performance Computer Architecture (HPCA - 6), 2000.

[4] T. Puzak, A. Hartstein, P. Emma, and V.Srinivasan. "Measuring the cost of a cache miss", in Workshop on Modelling, Benchmarking and Simulation (MOBS), 2006.

[5] **www.multi2sim.org**

AUTHORS

Vikram Bharadwaj,
Computer Science and Engineering from Sir M Visvesvaraya Institute of Technology, Bangalore.

Vineet Kachapur,
Computer Science and Engineering from Sir M Visvesvaraya Institute of Technology, Bangalore.