International Journal of New Computer Architectures and their Applications (IJNCAA) 4(4): 177-183
The Society of Digital Information and Wireless Communications, 2014 (ISSN: 2220-9085)

Adaptive Cache Replacement Policy (ACRP): A Dynamic Replacement Policy for Cache Management

Olakanmi O.O (Ph.D)
Electrical and Electronic Engineering,
Technology Drive, Offices 6, New Faculty of Engineering Building.
University of Ibadan, Ibadan, Nigeria
Email: olakanmi.oladayo@ui.edu.ng

ABSTRACT
The discovery of memory access time as one of the major design issues in the processor has increased effort in the development of different cache replacement policies to manage what to be the content of caches in both uniprocessor and multiprocessor. This is to improve the overall performance of the processor. Most time the potential benefits of these policies are not instantaneous due to the varying nature of the workloads. Therefore, it is difficult to identify which particular policy to apply during cache's content replacement.
In this paper, an adaptive replacement technique using the minimum threshold value of the improvement factor of cache, which was analytically obtained, is proposed in order to explore the combine merits of fundamental replacement policies and strikes a minimum balance with their demerits. ACRP approach uses the threshold values of improvement factor obtained from the analytic modeling to determine when the current replacement policy's performance becomes or getting worse, the ACRP switches to another replacement policy from the pool of replacement policy. Specifically, the ACRP approach solves the problem associated with wrong usage of replacement policy and reduces processor cache miss rate caused by wrong evictions.

Keywords: Cache, replacement policy, cache hierarchy, processor, adaptive caching

1. INTRODUCTION

Cache replacement policy is a management scheme used to improve the performance of cached systems. The major performance limitation of non-cache systems is due to upper limit set by the main memory access time and cache size, therefore reducing the overall performance of the cached system.

Cache memory is not a new concept; it has been one of the performance enhancement metrics in uniprocessor systems since the era of the earliest uniprocessor systems. Cache is a low access time and a low capacity memory which is used by the processor to temporarily hold those portions of the main memory which are currently in use or may be used in nearest future. It is majorly employed to increase the execution time of the machine due to its lower access time compared to main memory. However, cache has a low capacity, therefore, must only be used to store highly essential data during execution. In order to make cache to function properly, certain factors are needed to be considered at the design stage of the cache which includes: the fetch algorithm, the placement and replacement algorithms, line size, cache coherence, the behaviour of separate data and instruction caches and cache size. When information is requested by the CPU from main memory and the cache is full, some information in the cache must be selected for replacement. There are many replacement algorithms, such as FIFO (First in, First Out), LRU (least recently used), and random etc.

In the single core, the processor presents the cache with a series of requests for memory locations, where each request appears after the last one has been served. The time to serve each request depends on whether the memory location is in the cache (a cache hit occurs) or not (and then a miss occurs and the location is fetched from the main memory). If the cache is full, the new memory location enters the cache, and another memory location gets evicted.

Many factors affect the optimal performance of some of these replacement policies. Among these is the characteristic of workload which varies among workloads. The performance of replacement policy reduces when the cache is shared among processing elements, executing different workloads due to differences in the characteristics of access to metadata and data. All these can be sufficed to the application specificity of replacement policy which can be solved using adaptive replacement scheme. However, selection of adaptive criteria and what determines the criteria is main focus on development of adaptive replace-

ment schemes. Manual tuning is one of the ways to do this, however apart from being tedious it is very prone to errors. Another means is the application of machine learning algorithm as proposed in (Ismail et al. n.d.). In this paper analytical method is used to obtain the adaptive criteria. An analytical model of one and two level cache was done. A simple replacement status flag is used to call for change of the replacement policy

The remaining part of this paper is arranged as follows: section 2 contains reviews of some related works on cache replacement policies. The methodology of the proposed adaptive cache replacement policy and results of the analytical model are presented in section 3. Section 4 is the conclusion and recommendation.

2. RELATED WORKS

Currently, research efforts have been towards improving some of the conventional cache replacement policies. For example, (Moinuddin et al. 2006) highlighted some of the demerits of the traditional cache replacement policies and proposed a modified replacement policies called a Memory Level Parallelism (MLP) aware cache replacement policy. In MLP aware replacement policy, the algorithm computes the MLP-based costing for each cache miss and uses the MLP cost and recency to determine the cache block to be replaced. It further creates a low-hardware overhead mechanism called Sampling Based Adaptive Replacement (SBAR) which is capable of choosing between the proposed MLP-aware and traditional replacement policy. MLP- aware cache replacement policy improves performance by as much as 23% (Moinuddin et al. 2006).

In the (Keqiu Liy et al. 2010), the problem of cache replacement for multimedia object caching by exploring the minimal access cost of caching any number of versions of a multimedia object was proposed. An optimal solution for calculating the minimal access cost of caching any number of versions of the same multimedia object suggested and its extensive analysis was done. The performance metric was to minimize the total access cost by considering both transmission and transcoding costs. Based on this optimal solution, an efficient cache replacement algorithm for multimedia object caching was developed. The simulation results showed that the proposed algorithm outperforms comparison algorithms in terms of all the performance metrics considered.

CMP Cooperative Caching was proposed in (Jichuan & Gurindar 2006), a unified framework to manage a CMP's aggregate on-chip cache resources. Cooperative caching combines the strengths of private and shared cache organizations by forming an aggregate shared cache through cooperation among private caches (Jichuan & Gurindar 2006). Locally active data are attracted to the private caches by their accessing processors to reduce remote on-chip references, while globally active data are cooperatively identified and kept in the aggregate cache to reduce off-chip accesses. Examples of cooperation include a cache-to-cache transfers of clean data, replication-aware data replacement, and global replacement of inactive data. Cooperative caching was implemented by either modifying an existing cache replacement policy and cache coherence protocol, or by the new implementation of a directory-based protocol. The experimental results showed that cooperative caching performed robustly over a range of system/cache sizes and memory latencies. For an 8-core CMP with 1MB L2 cache per core, the best cooperative caching scheme improves the performance of multithreaded commercial workloads by 5-11% with a shared cache and 4-38% with private caches. For a 4-core CMP running multiprogrammed SPEC2000 workloads, cooperative caching is on average of 11% and 6% faster for the shared and private cache organizations respectively (Jichuan & Gurindar 2006).

Least Recently Used (LRU) is the replacement policy which evicts the page for which his last use was the earliest in time. LRU uses a cache which is a constant factor bigger than the online algorithm, its competitive ratio is constant (Bergh & Summerfield 1976) Not only does LRU have nice theoretical features, it also works well in practice. Although it is a little complicated to exactly keep track of the exact time in which every memory location in the cache was used, heuristics which approximate LRU's behaviour are used in most of the standard caches. LRU replacement policy is a commonly used in the cache management; however, its inability to cope with access patterns with locality mars its performance consistence. Some improvement on LRU, such as LRU-K and 2Q, attempts to enhance LRU capacity by making use of the additional historical information about previous block references other than only the recency information used in LRU had been suggested in some of the previous research works. However, the added history information greatly increases complexity which does not consistently favours performance improvement. Although, many recently proposed policies, such as UBM and SEQ, improve replacement performance by exploiting access regularities in references, but this only addresses LRU problems in certain specific and well-defined cases such as access patterns like sequences and loops. Reference (Song & Xiaodong 2002) also proposed another form of a replacement algorithm called Low Inter reference Recency Set (LIRS). LIRS effectively addresses the limits of LRU by using re-

cency to evaluate Inter-Reference Recency (IRR) for making a replacement decision. At the same time, LIRS almost retains the same simple assumption of LRU to predict future access behavior of blocks. The simulation results showed that LIRS significantly outperforms LRU, and outperforms other existing replacement algorithms in most cases with lower cost.

Another replacement was proposed in (Cristian et al. 2013). This is called RAM-frugal cache replacement policy that approximates the least-recently-used (LRU) policy. It uses two in-memory Bloom sub-filters (TBF) for maintaining the recency information and leverages an on-flash key–value store to cache objects. TBF requires only one byte of RAM per cached object, making it suitable for implementing very large flash-based caches. The evaluation results showed that TBF achieves a better cache hit rate and operations per second comparable to those of LRU in spite of its much smaller memory requirements (Cristian et al. 2013).

In (Hassidim 2009) (Smith 1982), it was shown that the optimal algorithm to choose which location to evict is Furthest in the Future (FITF) algorithm, which evicts the memory location which will be used in the latest possible time. The more interesting question is the online version of the problem, in which the algorithm sees the next request only after serving the last one. In this case, the common measure of the performance of an algorithm is its competitive ratio, or the number of misses it makes divided by the number an optimal online algorithm would make, on the worst case sequence. For this caching problem, attaining a good ratio is impossible, as all deterministic caching strategies have a competitive ratio which is at least k, the size of the cache.

ACME (Ismail et al. 2010) is the closest scheme to the one proposed in this paper. It is an Adaptive Caching Using Multiple Experts which uses a machine learning algorithm to dynamically determine the performance of replacement policy by assigning different weights to all the replacement policy in the expert pool in accordance to their real time performances. That is, replacement policy with highest weight at that moment and for the current workload has the best performance. And, it will be the replacement policy until another replacement policy with higher weight exists. In as this scheme improve the performance of the caching system, however, it introduces overhead such as delay in weight assignment, and overdependence on the machine learning algorithm. As a result of this, adaptive scheme using a threshold value which is compared with improvement factor of the cached system obtained on a real time basis is another alternative.

3. DESIGN APPROACH OF ADAPTIVE CACHE REPLACEMENT POLICY (ACRP)

Motivated by the limits of machine learning algorithm approach and manual tuning method, an approach called Adaptive Cache Replacement Policy is proposed. The ACRP takes advantages of timing by creating a performance measuring metric which dynamically determines when the replacement policy used is no longer contributing to effective management of the cache. The pre-stage of ACRP proffers an analytical modeling solution of two-level cache in a processor which produces an average improvement factor based on cache miss and hit rate for each level of cache. The obtained value, threshold improvement factor, serves as a metering pointer to change the replacement policy of the multiprocessor system from recency to time based or vice versa depending on the differential value between current cache improvement factor and analytical threshold improvement factor.

The two common cache hierarchies, one level cache and two-level cache, are modeled as shown in the next section. Figure 1 illustrates the working of 2-level cache. A request is sent by processing element (PE) to the L1-cache if the request is found in L1 then a hit otherwise miss will be recorded. As a result of miss in the upper level cache the request passed to the L2-cache. In a case of miss being recorded in L2, the request passed to main memory and the L1 and L2 are updated. In case L1 and (or) L2 cache are full or getting full the replacement policy comes in by making a vital decision on which block(s) to be removed. A wrong eviction will certainly cause miss rate in no later time. Different replacement policy considers different factors for eviction; however, these factors are associated with different data pattern exhibited by workload. This means a replacement policy may bias towards a workload with certain data pattern.

Figure 2 shows the proposed adaptive cache replacement policy block diagram. It has a pool of replacement policy containing LRU, LFU, and LIRS etc. ACRP is divided into two sections which are Decision unit and Calculated Improvement Factor (CIF) generation unit. The CIF generation unit uses the record of misses and hits to generate the CIF which will be compared with the Threshold Improvement Factor (TIF). In a case the current CIF is lower than the TIF a new replacement policy will replace the current replacement policy. As a result of this, the new replacement policy becomes current policy which will be used to make eviction decision. It is assumed that the adaptive caching design is using demotion in a 2-level cache A demotion strategy in more than a level cache involves downward movement of evicted objects to the next level

instead of outright discard of the evicted objects. (WONG, GANGER & WILKES 2000) as referenced in (Ismail et al. 2010) enumerated the benefits of demotion over exclusive caching.

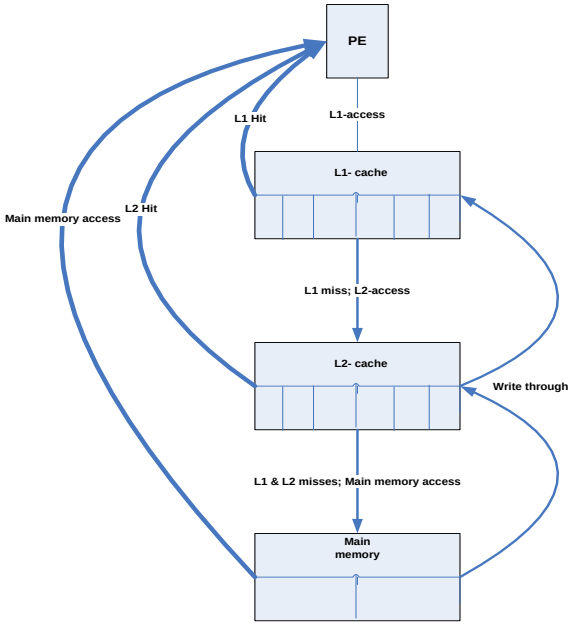

Fig 1: 2- Level Cache schematic showing a different level of access

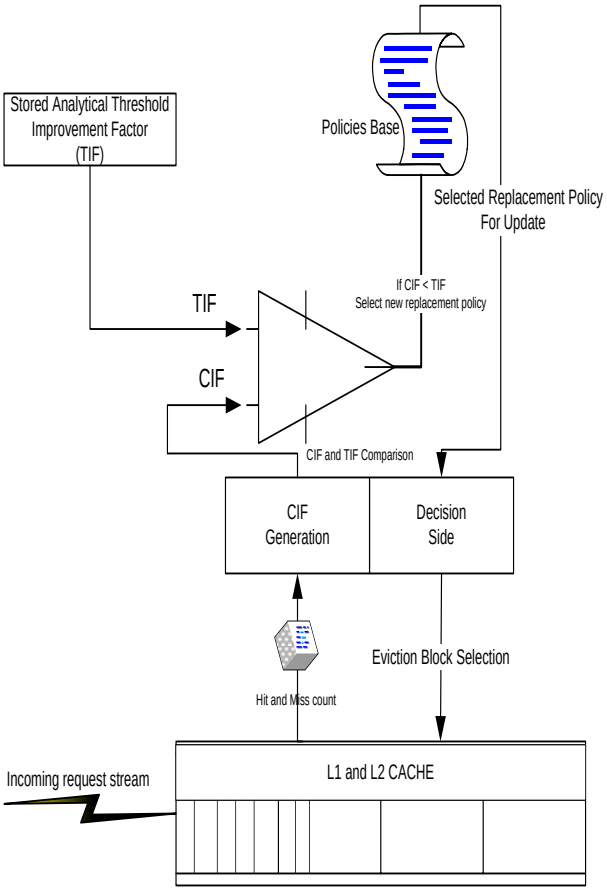

Figure 2: Adaptive Cache Replacement Policy (ACRP)

3.1. Analytical Modelling of 1 and 2-Level Cache to obtain Thresholds for the ACRP

As shown in Figure 1, a n-level cache was used where $1 \leq n \leq 2$. Each level of the cache offers different access time. Level 1 offers fastest access speed, say s' than level 2 and level 3. Level 2 offers faster access speed s'', than level 3 which is s'''. The memory access hierarchy is then modeled thus:

L1 cache: This is a fast SRAM of access time 0.5ns.

L2 cache: L2 cache is a larger but slower SRAM of access time $7ns$

Main memory is the largest but slowest of access time $100ns$

Assuming that

$p = Probability\ that\ cache\ is\ accessed$

$s' = cache\ \ access\ time$

$s''' = main\ memory\ access\ time$ Of cached system

$S = main\ memory\ access\ time$ Of non-cached system

3.1.1 One-Level Cache

For 'R' main memory accesses, any access that finally resolved at main memory must have beenchecked the cache. Making main memory access time to become $(s' + s''')$ access time. Therefore, R times main memory accesses gives:

Maximum access time $= \ pRs' + (1-p)(s' + s''')R$ ……. (1)

Assuming the multiprocessor is not cached; for an 'R' main memory access the maximum access time is Rs''' where s''' is the access time of the non-cache main memory. Assuming that IF is the improvement factor of cached memory over non-cache memory. Then estimated improvement factor IF is:

$$RS = IF\big(pRs' + (1-p)(s' + s''')\big)R \ldots\ldots\ldots\ldots\ldots\ldots (2)$$

$$IF = S/(s' + s''' - ps''') \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (3)$$

From eqn. 3, if there is cache hit $i.e\ s''' = 0$ equation.3 becomes:

$$IF = S/s' > 1 \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (4)$$

Equation 4 shows the improvement factor of the memory hierarchy whenever there is cache hit for one level cache memory. Equation 3 can be used to determine the pointer value which marks the point at which cache is no longer contributing to the speedup of the system.

3.1.2 Two-Level Cache

For 'R' HMPM main memory accesses, any memory access that finally resolved at level 3 (that is main memory) must have been checked in level 1 and level 2. Making level 3 cache access to have $(s' + s'' + s''')$ access time. Also, any memory access at level 2 must have been checked in level 1. This makes level 2 cache access to have $(s' + s'')$ access time. Therefore, R times HMPM main memory accesses gives:

$$\text{Maximum access time} = pRs' + p'R(s' + s'') + (1 - p - p')(s' + s'' + s''')R \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (5)$$

Assuming the multiprocessor is not cached; for 'R' main memory accesses the maximum access time is Rs, where s is the access time of the non-cache main memory. Assuming that *IF* is the improvement factor of cached memory compare to that without cache, we can estimate the improvement factor *IF* as:

$$RS = IF\big(pRs' + p'R(s' + s'') + (1 - p - p')(s' + s'' + s''')\big)R \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (6)$$

$$IF = S/(s' + s'' + s''' - ps'' - ps''' - p's''')$$

$$IF = S/(s' + s'' + s''' - p(s'' + s''') - p's''') \ldots\ldots\ldots (7)$$

From equation 7, assuming there is cache level 1 hit, the improvement factor becomes:

$$IF = S/s' > 1 \ \ i.e\ s'' = s''' = 0 \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (8)$$

In addition, if there is cache level 1 miss, but cache level 2 hit the memory hierarchy will still reduce access time and the improvement factor is:

$$IF = S\ /(s' + s'') - p(ps'') > 1 \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (9)$$

Equation 7 shows that whenever the Improvement Factor (IF) is less than zero, it implies that there is cache L1 & L2 misses and the hierarchy gives worst case access time. Meanwhile, high cache miss rate depends on the size of the cache and the efficiency of the replacement policy engaged in the cache management. Size has economic implication on the overall cost of the system, that is, there is an upper limit to the size of cache otherwise the cost will mar the speedup of the memory hierarchy. Therefore, an efficient replacement policy is the only metric that can reduce the cache miss rate..

ACRP Algorithm:

1. Set the replacement policy for the cache.
2. Pre load the cache with cacheable memory
3. Reset the replace replacement policy change pin.
4. Set the Threshold Improvement factor (TIF) to 25%
5. Perform cache access for n number of times
6. Generate the Current Improvement Factor (CIF) using equation 3 or 7
7. If CIF < TIF change the current replacement policy to new policy by asserting the replacement policy change pin otherwise maintain the initial state of the pin.
8. Jump to 5

3.1.3. Improvement Factor's Benchmark for ACRP

The strength of ACRP lies in obtaining an accurate threshold value for the improvement factor. This can be achieved numerically or analytically. However, numerical method involves a real time monitoring of cache and non-cached system, intermittently recording and calculating their hit and miss ratios from where the threshold value could now be obtained. This method is subjected to experimental errors and not cost friendly. Analytical method used equations 3 and 7 to obtain the benchmark value using various probability values to obtain the improvement factor for one and two level cache. This is then used as a pointer to when to dynamically change the replacement policy of the cache system. Once the improvement factor drops below the threshold value the ACRP will change the replacement policy. Fig 3-5 show the output of the analytical equation 3 and 7 for the one and two-level cache respectively, when used to determine TIF for one level cache

The main strength of ACRP is that it reduces the conflict misses which could be avoided if the cache replacement policy had not wrongly evicted an entry earlier. In this case ACRP will solely reduce the replacement misses which are due to the particular wrong choice of the replacement policy by monitoring when the current replacement policy starts to wrongly evicting block. ACRP first preloads the cache with part of the cacheable memory. This will reduce misses which are associated to compulsory or cold misses. The ACRP algorithm uses equation 9 to determine the

threshold improvement factor which is dynamically compared with Current Improvement Factor (CIF). In case the TIF is greater than the CIF the ACRP changes the current replacement policy to another replacement policy from the replacement policy base as shown in Figure.2 by asserting the change replacement policy's pin and then starts monitoring the performance of the new replacement policy through the CIF. However, if the TIF is less than CIF then the status quo will be maintained for the current replacement policy.

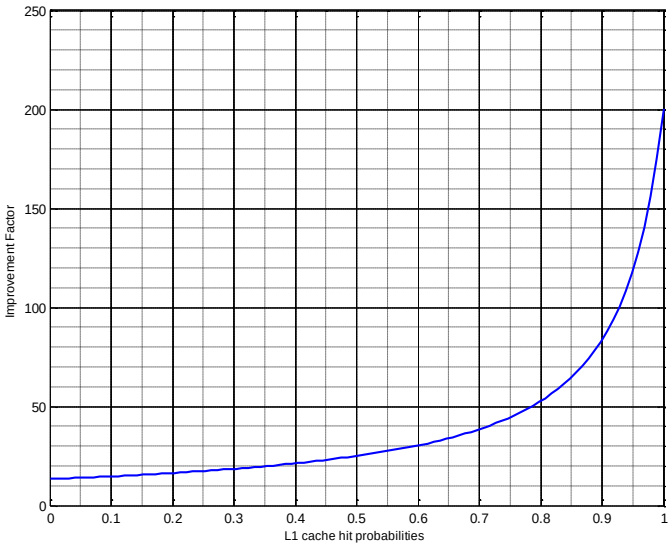

Figure 3: Estimated improvement factor for different probabilities of the L1 cache Hit

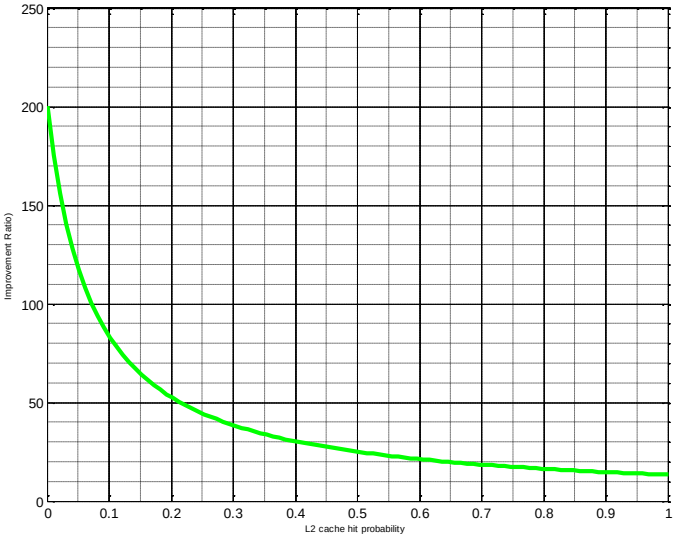

Figure 4: Estimated improvement factor for different probabilities of the L2 cache Hit

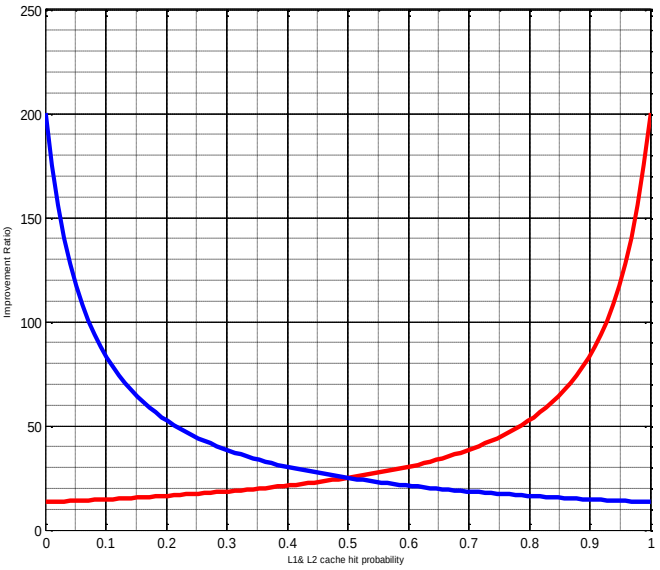

Figure 5: Estimated improvement Factors for different probabilities of HMPM L1 and L2 cache hit

4. RESULTS AND DISCUSSIONS

The mathematical models of L1 &L2 cache were implemented using different probabilities of both cache level n probabilities. This was done to obtain the threshold improvement factor and its corresponding cache level hit rate. Figure 3 and 4 illustrate the improvement factors of 1-level and 2-level cache respectively for different cache hit rates. Figure 5 shows that the minimum threshold improvement factor (25.2) for a 2-level cache is reached when the cache hit rate of both L1 and L2 slides to 25%. From this, the current replacement policy has started wrong eviction of blocks and any further wrong eviction will affect the speedup of the system. However, this must be after cold misses period has passed otherwise the recorded misses are due to cold miss.

Any occurrence of miss after threshold value implies that the current replacement policy has started wrong evictions and needs to be replaced with another policy .Once the threshold value is reached, the ACRP changes the replacement policy to another policy in order to prevent further eviction of wrong blocks. The threshold value approach in adaptive caching is easy to implement, however, it should be noted that its performance depends solely on the accuracy of the obtained threshold value. That is the analytical model must be closed to the true form of the caching system before one can say the threshold value is accurate.

5. CONCLUSION

The paper presented the mathematical models of access time and improvement ratio of one and two level cache memory hierarchies. The results of the analytical models are used to propose a novel replacement policy which monitors the efficiency of the current replacement policy in either one or two level cache. However, a simulator need to be developed in order to compare the efficiency of ACRP with some other existing replacement policies mentioned in the related works sections.

REFERENCES

1. Belady, L. (1966). A study of replacement algorithms for virtual-storage computer. *IBM Systems Journal*, 78-101.
2. Bergh, H., & Summerfield, A. (1976). CPU Busy not all productive Utilisation. *Share Computer Measurement and Evaluation, 39*, 95-97.
3. Cristian, U., Biplob, D., Stephen, R., & Akshat, A. (2013). TBF: A Memory-Efficient Replacement Policy for Flash-based Caches. *29th IEEE International Conference on Data Engineering.* Brisbane, Australia.
4. Evan, S., Hazim, S., Lixin, Z., & Ram, R. (2005). Adaptive Mechanisms and Policies for Managing Cache Hierarchies in Chip Multiprocessors. *32nd International Symposium on Computer Architecture* (pp. 346-356). IEEE.
5. Hassidim, A. (2009). Cache Replacement Policies for Multicore Processors.
6. Ismail, A., Ahmed, A., Robert, G., Ethan, M., Scott, B., & Darrel, L. (2010). ACME: Adaptive Caching Using Multiple Experts.
7. Jiang, S., & Zhang, X. (n.d.). LIRS: An Efficient Low Inter-reference Recency Set Replacement Policy to Improve Buffer Cache Performanc .
8. Jichuan, C., & Gurindar, S. S. (2006). Cooperative Caching for Chip Multiprocessors. *Proceeding of the 33rd annual International symposium on Computer Architecture ISA '06. 34(2)*, pp. 264-276. ACM New York.
9. Keqiu Liy, z., Takashi, N., Hong, S., Francis, Y. L., & Weishi, Z. (n.d.). An Efficient Cache Replacement Algorithm for Multimedia Object Caching.
10. Miguel, A. V., Juan, M. S., & Francisco, A. Z. (n.d.). Simulation of Cache Memory Systems on Symmetric Multiprocessors with Educational Purposes.
11. Moinuddin, Q., Daniel, L., Onu, M., & Yale, P. (2006). A case for MLP-Aware Cache Replacement. *ISCA'06 Proceeding of the 33rd annual international symposium on Computer Architecture. 34*, pp. Pp 167-178. Austin: IEEE Computer Society.
12. N, R., Srinivas, V., & Ammasai, G. (2011). Performance of Cache Memory Subsytems for Multicore Architectures. *International Journal of Computer Science and Engineering Applications (IJCSEA), 1*(5), 59-71.
13. Smith, A. J. (1982). Cache Memories. *Computing Surveys, 14*(3).
14. Song, J., & Xiaodong, Z. (2002). LIRS: An Efficient Low Interreference Recency Set Replacement Policy to Improve Buffer Cache Performance. *ACM SIGMETRICS conference on Measurement and Modeling of Computer Systems.*
15. Sujit, D., Priya, R., & Sulabha, A. (2010). Cache Coherence in Centralized Shared Memory and Distributed Shared Memory Architectures. *International Journal on Computer Science and Engineering*, 39-44.
16. Wenyu, Q., Keqiu, L., Hong, S., & Yingwe, J. (2005). The cache Replacement Problem for Multimedia Object Caching. *First International Conference on Semantics, Knowledge and Grid SKG '05*, (p. 26).
17. WONG, T. M., GANGER, G. R., & WILKES, J. (2000). My cache or yours? Making storage more exclusive. *CMU-CS-00-157*. Carnegie Mellon University.

AUTHOR PROFILE

O.O Olakanmi received the B.Tech in Computer Engineering from Ladoke Akintola University of Technology, Ogbomosho in 2000. Also, he received M.sc. in Computer Science and Ph.D. in Electrical and Electronic Engineering from University of Ibadan. He is major in Parallel Computing, High Performance Computing & Information Security, and currently a lecturer in the Department of Electrical & Electronic Engineering.