International Journal of Computer Applications (0975 – 8887)
Volume 105 – No. 8, November 2014

Cache Replacement Policies for Improving LLC Performance in Multi-Core Processors

Muthukumar S
Professor, Department of CSE,
Sri Venkateshwara College of Engineering,
Tamil Nadu, India.

Jawahar P.K
Professor, Department of ECE,
BS Abdur Rahman University,
Tamil Nadu, India

ABSTRACT

Poor cache memory management can have adverse impact on the overall system performance. In a Chip Multi-Core (CMP) scenario, this effect can be enhanced as every core has a private cache apart from a larger shared cache. Replacement policy plays a key role in managing cache data. So it needs to be extremely efficient in order to extract the maximum potential of the cache memory. Over the years versatile set of replacement policies have been proposed and implemented and few of them (LRU, MRU etc) have proven to work well compared to others. However recent works have shown that few counter based replacement strategies have marginally outperformed LRU for certain workloads as LRU does not dynamically adapt to changing workload patterns. This work explores three counter based replacement techniques namely Context-Based Data Pattern Exploitation (CB-DPET), Logical Cache Partitioning Technique (LCP) and Sharing and Hit-Based Prioritizing Technique (SHP). Evaluation is carried out on 4 core and 8 core platforms (apart from 2 core platform which was already done as part of previous works) using PARSEC benchmarks and various performance metrics like throughput speedup, hit rate etc are captured and compared with that of LRU. All the three methods have produced better results on the performance metrics when compared to LRU.

General Terms

Cache Memory, Last Level Cache, Multi-Core Architecture

Keywords

Cache, Counter, Throughput, Hit Rate, Replacement.

1. INTRODUCTION

Cache memory helps in expediting the speed of data retrieval time in processors. As the number of cores increase, the importance of cache memory also increases. Every core has an inbuilt on-chip cache which is often referred to as the L1 cache or the level 1 cache. This cache lies closest to the processor and has the minimum data retrieval and storage time. All the available cores share a relatively larger L2 or level 2 cache or more traditionally called as the Last Level Cache (LLC). The general search hierarchy begins from the innermost L1 cache. If an element is not found there, the processor looks in the L2 cache and then finally search ends in secondary memory which is often time consuming.

This explains why it is extremely important to manage the available cache memory efficiently. Managing LLCs becomes even more important and challenging as data present in them are accessed by multiple cores which might result in coherency problems. Apart from that there are many factors that impact cache management. One of the most crucial factors is the replacement algorithm running on the cache. A good replacement algorithm implies faster data retrieval time and judicious cache space utilization. Both in turn can improve the overall system performance. One of the widely used replacement algorithms in modern time is the Least Recently Used (LRU) approach. It makes decisions based on the spatial and temporal localities.

While it works well with most applications, there are a few issues that need to be addressed. It is static in nature and does not change the way it makes replacement decisions when there is a change in the workload pattern. Certain workloads which do not obey the spatial and temporal locality principles, may not work well with LRU. This stresses the need for novel techniques that can provide the benefit of LRU and be more dynamic as well. This work proposes three novel counter based cache replacement strategies which have proven to have outperformed LRU in terms of the hit rate, IPC throughput etc. These techniques have been evaluated by applying over the LLC.

Rest of the paper is organized as follows: section 2 looks into the related work that was done in this field, section 3 provides an overview on all the three methods, sections 4 and 5 describes the experimental setup and analyses the obtained results respectively, Section 6 describes the hardware overhead involved in all the methods and finally section 7 summarizes the paper followed by the list of references.

2. RELATED WORK

Many works [6,7,812,14,15,17,19] have been proposed to improve the performance at the shared Last Level Caches (LLC). Multiple cores tend to compete for the data present in this cache level and hence care should be taken to ensure that the cache is managed effectively.

The temporal based multi-level correlating inclusive cache replacement [6] proposes a method to improve the performance of inclusive caches. This method chooses the victim for replacement in LLC based on the information from all levels of cache hierarchy which in turn reduces the overhead of invalidation of highly referenced blocks in higher level caches. SHiP [7] proposed by Carole-Jean Wu et al, discusses about a technique to improve the performance of cache for multi-programmed workloads in CMPs. It correlates the re-reference behavior of every cache block with a unique signature based on memory region, program counter and instruction sequence history and makes intelligent replacement decisions. The adaptive LLC-memory traffic management (ARI) [8] focuses on the traffic from LLC to main memory due to writebacks, energy consumption etc. It suggests an adaptive technique to improve the performance by reducing writebacks and optimizing miss rate relative to LRU replacement policy.

While [6] involves active message passing between various cache levels, [7] requires an unique signature to be generated for every cache block, both capable of increasing the hardware overhead over a period of time.

Data shared by multiple threads needs to be treated with more importance when compared to other data. This is because any miss on them can stall many threads at the same time thereby affecting the performance. A dynamic cache management scheme [18] proposed by Fazal Hameed et al is another work that was targeted towards LLCs. But it does not attach any importance to shared data.

Mainak Chaudhuri et al have come up with a work [17] that discusses on making replacement decisions at the LLC based on the activity history that occurs in the inner levels of the cache. Depending on the hits/misses encountered here for a data item, replacement decisions are made at the LLC. But communicating such access history information across various levels of cache frequently can cause significant overhead. Counter based replacement techniques are gaining popularity in recent times [9, 13]. RRIP technique [9] suggested by Aameer Jaleel et al is a counter based replacement policy that predicts the re-reference interval of a cache block for efficient cache utilization with lesser hardware overhead. But in a multi-threaded scenario the sharing degree of cache blocks are not considered in this method while making replacement decisions. Mazen Kharbutli and Yan Solihin have proposed the counter based dynamic replacement approach [13] to enhance cache performance. Thrashing condition has not been taken into account here though.

Moinuddin K. Qureshi et al have come up with a set dueling algorithm [10] which modifies the insertion policy of a replacement algorithm for improving cache performance of memory intensive workloads. Cache partitioning helps in reducing the search space for a replacement candidate when a replacement needs to be made. Adaptive Bloom Filter Cache Partitioning Scheme for Multi-Core Architectures [16] dynamically partitions the available cache space using bloom filters and counters. Since every core is allocated an array of bloom filters and counters, the hardware overhead tend to increase as the number of cores increases. Phase Change Memory (PCM) was suggested as an alternative to the traditional DRAM and techniques have been proposed to improve the performance of the LLC in PCM [19] but the drawback of PCM is that the writes are much slower compared to DRAM.

Almost all the techniques discussed above either do not attach importance to data that is shared by multiple threads or possess significant hardware overhead. To address the shortcomings of these methods, this work focuses on three novel counter based replacement algorithms for shared LLC in a CMP environment namely – CB-DPET, LCP and SHP. Overview of each of the method is provided in the next section.

3. OVERVIEW OF THE COUNTER BASED REPLACEMENT TECHNIQUES

3.1 CB-DPET

Context-based data pattern exploitation technique [1] tries to maximize the overall hit percentage by making replacement decisions in accordance with the changing workload pattern. This is achieved with the help of a counter called PET counter that is associated with every cache block in the cache. It is a 3-bit long counter which can take values starting from 0 till 4. Based on the number of hits received by a particular cache block, this value is modified such that while performing replacement, it yields better results compared to the LRU approach.

Initially every block's counter value is set to the maximum value (4). When a new data item arrives CB-DPET searches for the cache block which holds the maximum possible value which is 4. This block is chosen as the replacement victim. Now when an element receives a hit, its counter value is brought down to 0 irrespective of whatever value it had previously. For every miss, counter value of every other block in the cache is decremented by one. This will ensure that the counter does not hold stale data at any point in time. Cache thrashing condition has been taken care with a slight modification in the insertion policy. With additional register support CB-DPET can be extended to suit multi-threaded applications as well. In a multi-threaded scenario, CB-DPET gives more importance to data that is shared by multiple threads, allowing it to stay in the cache for longer period of time compared to other data items.

3.2 LCP

As the name implies, logical cache partitioning approach [2] divides the cache data into logical zones based on the likeliness to be referenced by the processor in the future. This prediction information is collected by associating every cache block with a 3-bit counter called LCP counter. It can take all possible values from 0 to 7. This information is then used to make judicious replacement decisions. The zone information is shown in table.

Table.1 LCP Counter Value Range and Corresponding Zones

Counter Value Range	Zone
6-7	MLR
3-5	LR
1-2	LLR
0	NLR

MLR, LR, LLR and NLR refer to Most Likely to be referenced, Likely to be Referenced, Less Likely to be referenced and Not Likely to be Referenced respectively. For every hit received by an element, its LCP counter value is set to the next higher zone's starting value. For every miss, the counter values of all the other blocks are decremented by one. Care needs to be taken to ensure that the counter value does not overshoot its specified range while decrementing or incrementing.

While making replacement decisions the candidate is chosen from the bottom most zone (see table 1). If no element is found to exist in that range, the next higher zone is chosen as the search space till the MLR zone is reached. During insertion, the new element is always inserted in the LLR zone.

3.3 SHP

Studies have shown that in multi-threaded applications sharing nature of data items plays a vital role in deciding the performance of the system. CB-DPET attaches importance to shared data but it does not capture the degree of sharing of every cache block (i.e.) the extent to which every block is shared. This information can help determine the importance of the data item which in turn will aid in making replacement decisions. Sharing and hit based prioritizing algorithm [3] collects this information with the help of a counter called SHP counter, which is 2-bits in length, and few other registers.SHP counter iterates through its entire range of 0 to 3. Table 2 shows the sharing degree values and their descriptions. Nature of sharing is determined from the number of threads that try to access the data item. To keep track of all the threads that try to access a cache block, a dynamic Thread Tracker (TT) filter is associated with every cache block.

Sometimes sharing degree alone may not be enough while making replacement decisions. An additional hit counter is augmented with every cache block to keep track of the number of hits received. This information is combined with the sharing degree of the block to arrive at a priority for that block. This priority is later used by the algorithm to make efficient replacement decisions.

Table.2 Sharing degree values and their descriptions

Number of Accessing Threads	Sharing Degree Counter Value	Nature of Sharing
1	0	Private/Not Shared
2-3	1	Lightly Shared
4-7	2	Heavily Shared
8-10	3	Very Heavily Shared

4. EXPERIMENTAL SETUP

For experimental purposes, a full-fledged, open-source, modular, object-oriented computer system architecture simulator platform called Gem5 [20] has been chosen. One of its key features includes the support for multiple Instruction Set Architectures (ISAs) and multiple processor cores. Alpha ISA has been chosen to evaluate our method. Simulation has been carried out on 4-core and 8-core processor architectures. The basic cache hierarchy in both the cases goes as follows:

Every core has private L1 cache which is further sub-divided into instruction and data caches. At the next level there is a relatively larger L2 cache which is shared by all the available cores.

Table.3 Cache Configuration Parameters

Attributes	L1 Cache	L2 Cache
Total Size	64kB	2MB
Line size	64B	64B
Associativity	2	8
Replacement Algorithm	LRU	CB-DPET/LCP/SHP

From table 3 it can be seen that the L2 cache is selected to run the proposed replacement techniques one at a time. As far as the workloads are concerned, a benchmark suite called PARSEC (Princeton Application Repository for Shared-mEmory Computers) [21, 22] has been chosen, which comprises of versatile, large-scale commercial multi-threaded workloads. Eight of them have been selected from the list to evaluate all the three techniques.

5. RESULTS AND DISCUSSIONS

Results show the average Instructions Per Cycle (IPC) speedup and percentage increase in overall number of hits obtained when compared to LRU for different number cores (2,4 and 8 core architectures).

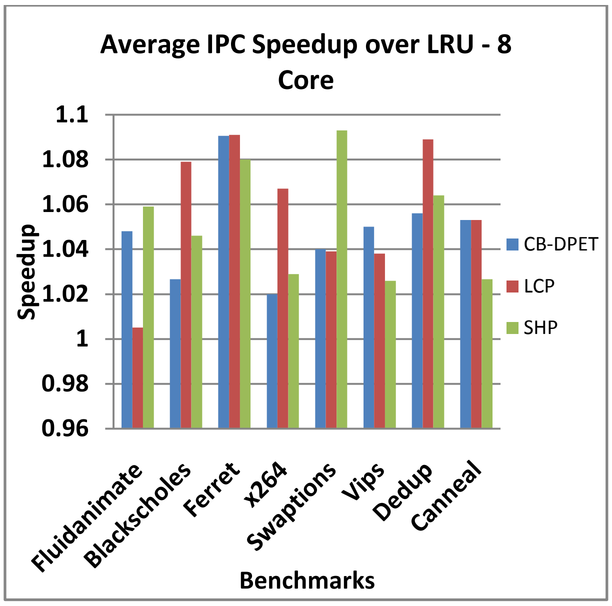

Figure 1: Average IPC Speedup Over LRU – 8 Core

Throughput can be measured from the total number of instructions that get executed in a single clock cycle (Instruction Per Cycle or IPC). In multi-threaded applications, throughput is the sum of individual thread IPCs as shown below.

$$\text{Throughput} = \sum_{i=1}^{n} IPC_i$$

Throughput Speedup of Algorithm k over LRU is IPC_k IPC_{LRU} where 'k' can refer to either one of CB-DPET, LCP or SHP.

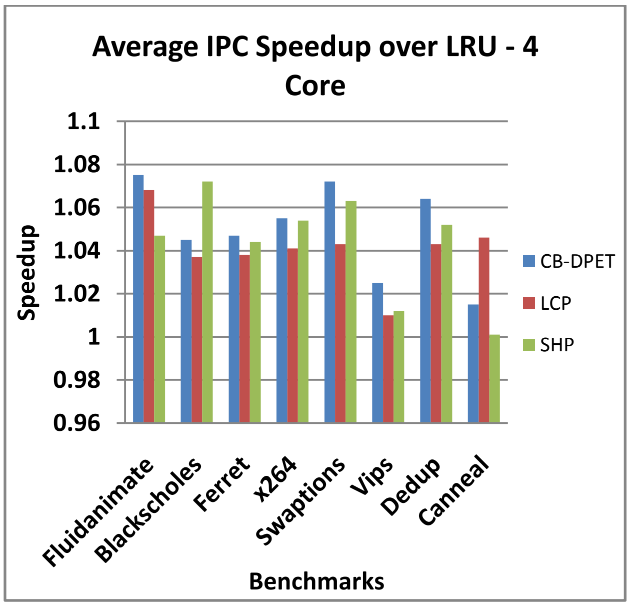

Figure 2: Average IPC Speedup over LRU – 4 Cores

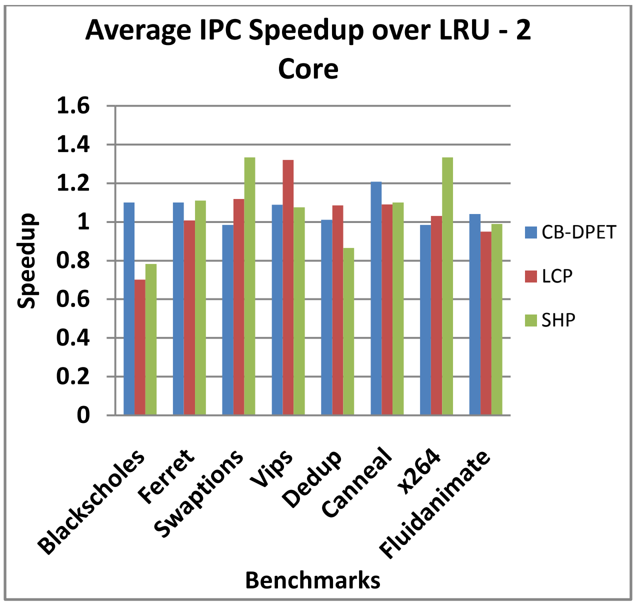

Figure 3: Average IPC Speedup Over LRU – 2 Core

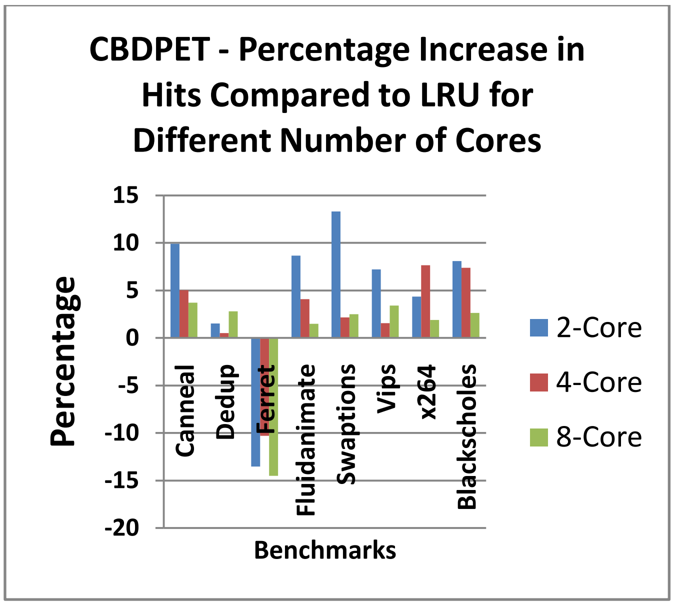

Figure 4: CB-DPET - Percentage Increase in Hits Compared to LRU for Different Number of Cores

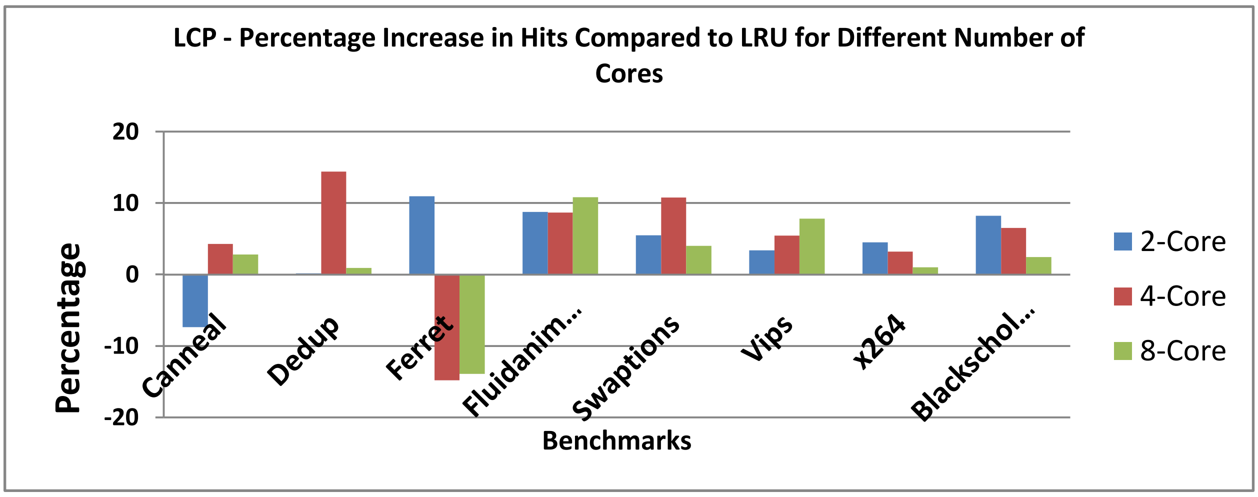

Figure 5: LCP - Percentage Increase in Hits Compared to LRU for Different Number of Core

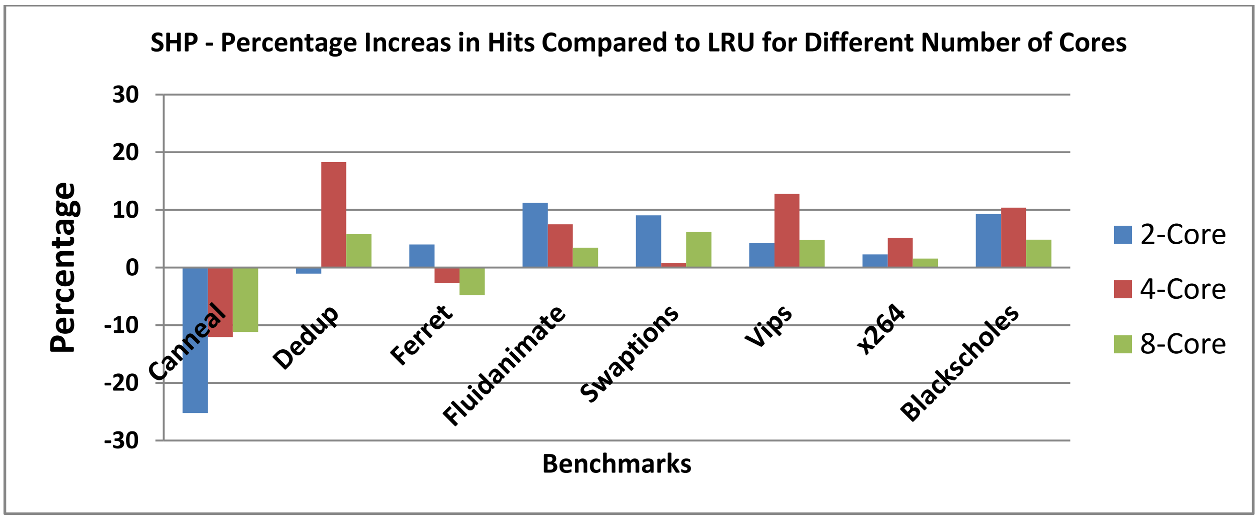

Figure 6: SHP - Percentage Increase in Hits Compared to LRU for Different Number of Cores

Figures 1, 2 and 3 shows the throughput speedup obtained for all the three methods in 8-core, 4-core and 2-core scenarios respectively. It can be seen that majority of the benchmarks have reported an average speedup of more than 1.

Figure 4 shows the percentage increase in overall number of hits obtained by applying CB-DPET in a 2-core, 4-core and 8-core scenario. Except for *ferret* all the other benchmarks have reported increase in hits compared to LRU. Similarly Figures 5 and 6 show the hit percentage increase observed across LCP and SHP respectively in all the three scenarios. Majority of benchmarks have outperformed LRU when it comes to cache hits. Table 4 highlights the approximate percentage increase in overall hits for all the three methods when compared to LRU.

Table.4 Average Percentage Increase in Overall Hits Compared to LRU

Method	2-Core	4-Core	8-Core
CB-DPET	8%	4%	3%
LCP	6%	9%	5%
SHP	7%	9%	5%

6. HARDWARE OVERHEAD

6.1 CB-DPET

PET counter requires frequent lookup so it can be kept it as a part of the cache hardware to expedite the access time. Considering a 8-way associative, 2MB cache which has 4096 sets and the cache block size is 64 Bytes. Additional Storage Required for PET counter = 4096 [number of sets] * 8 [number of cache blocks per set] * 3/8 [3-bits required per PET counter converted to bytes]

= approximately 12124.16 Bytes (OR) **12.12KB**

DM Register is allocated on a per thread basis and it depends on the application during its run time so it is kept more like software based register rather than a hardware one. The decision to have block status register either as a part of hardware or software depends during implementation time because the value it can hold is not fixed right now. Assuming that it can take only a '0' (say shared) and '1' (not shared) and assuming that the thread id range do not start from '0': If implemented in hardware. Additional Storage Required for blockstatus register = approx **4KB**. The total additional storage required is approximately: **16KB (i.e.) 0.78% overall increase in area of a 2MB L2 Cache.**

6.2 LCP

Similar to PET, LCP counter also requires frequent lookup so keeping it as a part of the cache hardware to expedite the access time. Storage overhead is similar to that of PET counter which is approximately **12.12KB** since LCP is also a 3-bit counter. As no other registers/counters are required LCP results in **0.59% overall increase in area of a 2MB L2 Cache.**

6.3 SHP

SHP requires a 2-bit Sharing Degree counter which requires frequent look up and hence keeping it as a part of the hardware. Additional Storage Required for Sharing Degree counter: 4096 [number of sets] * 8 [number of cache blocks per set] * 2/8 [2-bits required per counter converted to bytes] = 8192 bytes (or) **8.19KB**. Thread Tracker filter which is used to keep track of the threads that accesses the blocks can be implemented as a part of software which gives the flexibility of dynamically allocating and freeing the memory as and when required. Hence SHP results in **0.39% overall increase in area of a 2MB L2 Cache.**

7. CONCLUSION

Cache management is imperative to achieve high performance levels in any system architecture. In a multi-processor environment LLC play vital role. So it is essential to have an efficient and dynamic replacement algorithm in place. This work discusses about three novel counter based replacement techniques

- CB-DPET associates a 3-bit counter with every cache block which aids in making replacement decision in accordance with the changing workload patterns. Additional registers are added to support multi-threaded applications.
- LCP associates a 3-bit counter with every cache block to classify it into any one of the logical zones based on the number of hits it received. This information is then used in making judicious replacement decisions.
- SHP is another counter based approach where the sharing degree of every cache block is computed with the help of a 2-bit counter. This information is combined with the number of hits received by the element to arrive at a priority for every block which is then used in making efficient replacement decisions.

Experimental results have shown improvement in throughput speedup of up to 1.4 and percentage increase in overall number of hits of up to 9% compared to LRU.

8. REFERENCES

[1] Muthukumar S and P K Jawahar, "Cache Replacement for Multi-Threaded Applications Using Context Based Data Pattern Exploitation Technique", Malaysian Journal of Computer Science, Vol 26(4), 2013, p.277-293.

[2] Muthukumar S and P K Jawahar, "Hit Rate Maximization by Logical Cache Partitioning in a Multi-Core Environment", Journal of Computer Science, 10(3), 2014, p.492-498.

[3] Muthukumar S and P K Jawahar, "Sharing and Hit Based Prioritizing Replacement Algorithm for Multi-Threaded Applications", International Journal of Computer Applications (IJCA), Vol 90, Issue 12, 2014, p.34-38.

[4] Muthukumar S and P K Jawahar, "Redundant Cache Data Eviction in a Multi-Core Environment", International Journal of Advances in Engineering and Technology (IJAET), Vol 5, Issue 2, 2013, p.168-175.

[5] Felipe L. Madruga, Henrique C. Freitas, Philippe O. A. Navaux and P K Jawahar, "Parallel Shared-Memory Workloads Performance on Asymmetric Multi-Core Architectures", 18th Euromicro Conference on Parallel, Distributed and Network-Based Processing, 2010, p.163-169.

[6] Yingying Tiyan, Samira M. Khan, Daniel A. Jimenez, "Temporal-Based Multilevel Correlating Inclusive Cache Replacement", ACM Transactions on Architecture and Code-Optimization, Vol 10, No 4, Article 33, 2013.

[7] Carole-Jean Wu, Aamer Jaleel, Will Hasenplaugh, Margaret Martonosi, Simon C. Steely Jr. Joel Emer, "SHiP: Signature-based Hit Predictor for High Performance Caching", Proceedings of the 44th Annual

IEEE/ACM International Symposium on Microarchitecture 2011, p.430-441.

[8] Viacheslav V. Fedorov, Sheng Qiu, A. L. Narasimha Reddy, "ARI: Adaptive LLC-Memory Traffic Management", ACM Transactions on Architecture and Code-Optimization, Vol 10, No 4, Article 46, 2013.

[9] Aamer Jaleel, Kevin B. Theobald, Simon C. Steely Jr. Joel Emer, "High Performance Cache Replacement Using Re-Reference Interval Prediction (RRIP)", Proceedings of the 37th annual International Symposium on Computer Architecture (ISCA), Vol 38, Issue 3, 2010, p.60-71.

[10] Moinuddin K. Qureshi, Aamer Jaleel, Yale N. Patt, Simon C. Steely Jr. Joel Emer, "Set-Dueling-Controlled Adaptive Insertion For High-Performance Caching", Proceedings of the 37th annual International Symposium on Computer Architecture (ISCA), Vol 28, Issue 1, 2009, p.91-98.

[11] Aamer Jaleel, William Hasenplaugh, Moinuddin Qureshi, Julien Sebot, Simon Steely, Joel Emer, "Adaptive Insertion Policies for Managing Shared Caches", ACM Parallel Architectures and Compilation Techniques (PACT), Oct. 2013, p.208-219.

[12] Mazen Kharbutli, Yan Solihin, "Counter Based Cache Replacement and Bypassing Algorithms", IEEE Transactions on Computers, Vol. 57, Issue. 4, April 2008, p.433-447.

[13] Carole-Jean Wu, Margaret Martonosi, "Adaptive Timekeeping Replacement: Fine-Grained Capacity Management for Shared CMP Caches", ACM Transactions on Architecture and Code Optimization, Vol. 8, No. 1, Article 3, April 2011.

[14] Shekhar Srikantaiah, Mahmut Kandemir, Mary Jane Irwin, "Adaptive Set Pinning: Managing Shared Caches in Chip Multiprocessors", ACM Architectural Support for Programming Languages and Operating Systems (ASPLOS), Vol. 36, Issue. 1, March 2008, p.135-144.

[15] Konstantinos Nikas. Matthew Horsnell. Jim Garside. 2008. An Adaptive Bloom Filter Cache Partitioning Scheme for Multi-Core Architectures. In Proceedings of the IEEE International Conference on Embedded Computer Systems Architectures Modeling and Simulation, p.25-32.

[16] Mainak Chaudhuri, Jayesh Gaur, Nithiyanandan Bashyam, Srinivas Subramoney, Joseph Nuzman, "Introducing Hierarchy-Awareness in Replacement and Bypass Algorithms for Last-Level Caches", ACM Parallel Architectures and Compilation Techniques (PACT), Sep. 2012, p.293-304.

[17] Fazal Hameed. Bauer L. and Henkel J. 2012. Dynamic Cache Management in Multi-Core Architectures through Runtime Adaptation. In Proceedings of Design Automation & Test in Europe Conference & Exhibition (DATE), p.485-490.

[18] Miao Zhou, Yu Du, Bruce Chilers, Rami Melham, Daniel Mosse, "Writeback-Aware Partitioning and Replacement for Last-Level Caches in Phase Change Main Memory Systems", ACM Transactions on Architecture and Code Optimization, Vol. 8, No. 4, Article 53, Jan. 2012.

[19] N. Binkert et al. "The gem5 simulator", SIGARCH Computer. Architecture New, Vol. 39, Issue. 2, May 2011, p.1-7.

[20] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, Kai Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications", Princeton University Technical Report, TR-811-08, Jan. 2008.

[21] M. Gebhart et al., "Running PARSEC 2.1 on M5", University of Texas at Austin, Department of Computer Science, Technical Report, TR-09-32, Oct. 2009.