Journal of Computer Science 10 (3): 492-498, 2014
ISSN: 1549-3636
© 2014 Science Publications
doi:10.3844/jcssp.2014.492.498 Published Online 10 (3) 2014 (http://www.thescipub.com/jcs.toc)

HIT RATE MAXIMIZATION BY LOGICAL CACHE PARTITIONING IN A MULTI-CORE ENVIRONMENT

[1]Muthukumar, S. and [2]P.K. Jawahar

[1]Department of Computer Science and Engineering, SVCE, Chennai, India
[2]Department of Electronics and Communication Engineering, BSA University, Chennai, India

Received 2013-10-14; Revised 2013-11-17; Accepted 2013-11-29

ABSTRACT

It is imperative for any level of cache memory in a multi-core architecture to have a well defined, dynamic replacement algorithm in place to ensure consistent superlative performance. The most prevalently used LRU replacement policy does not acquaint itself dynamically to the changes in the workload. As a result, it can lead to sub-optimal performance for certain applications whose workloads exhibit frequently fluctuating patterns. To overcome the limitation of this conventional LRU approach, our paper proposes a novel counter-based replacement technique which logically partitions the cache elements into four zones based on their 'likeliness' to be referenced by the processor in the near future. Categorizing the elements into different zones is achieved with the help of a 3-bit counter that is associated with every cache line. On a cache hit, the corresponding element is promoted from one zone to another zone. Replacement candidates are chosen from the zones in the ascending order of their 'likeliness factor' (i.e.,) the first search space for the victim would be the never likely to be referenced zone, followed by the subsequent zones till the most likely to be referenced zone is reached. Periodic zone demotion of elements also occurs to make sure that stale data does not pollute the cache. Experimental results obtained by using the PARSEC benchmarks have shown almost 7% improvement in the overall number of hits and 3% improvement in the average cache occupancy percentage when compared to LRU algorithm.

Keywords: Replacement, Counter, Cache, Hit Rate, Multi-Core, Zone

1. INTRODUCTION

Replacement policies play a vital role in determining the performance of a system. In a multi-core environment, where processing speed and throughput are of essence, selecting a suitable replacement technique for the underlying cache memories becomes crucial. Cache memory hierarchy in multi-core processors usually consists of two levels. The private level 1 (or L1) cache associated with every core and a relatively larger level 2(or L2) cache which may be shared by all the cores. If this turns out to be the final cache level, it can also be referred to as the Shared Last Level Cache (LLC).

In present day environment, LRU approach is widely deployed across shared LLCs to choose the replacement candidates. This method seems to work well with many applications found today. However when it comes to certain workloads which possess an unpredictable data reference pattern that fluctuates frequently with time, LRU can result in poor performance. A closer observation on LRU can help reveal the cause for this problem. It assumes that all the applications follow the same reference pattern (i.e.,) they abide by the spatial and temporal locality theories. Any data item that is not accessed by the processor over a period of time is tagged as 'least recently used' and evicted from the cache. When such a data item is required by the processor in future, it is not found in the cache and has to be fetched from the main memory.

Two other replacement strategies that are commonly being employed apart from LRU include the Not Recently Used (NRU) and Most Recently Used (MRU) policies. Functionally MRU is pretty much similar to LRU with a

Corresponding Author: Muthukumar, S., Department of Computer Science and Engineering, SVCE, Chennai, India slight difference that the most recently referred data item is chosen as victim. So there will not be much difference in terms of performance compared to LRU. NRU associates a single bit counter with every cache line to aid in making replacement decisions. It might be effective compared to LRU in certain cases, but a 1-bit counter may not be powerful enough to create the required dynamicity.

To address the shortcomings of the previously discussed algorithms and to improve the overall hit rate, the study proposes a novel counter based replacement technique that logically partitions the cache into four different zones namely-Most Likely to be Referenced (MLR), Likely to be Referenced (LR), Less Likely to be Referenced (LLR), Never Likely to be Referenced (NLR) in the decreasing order of their likeliness factor Replacement, insertion and promotion of data elements take place within these zones in such a manner that the overall hit rate is maximized.

1.1. Related Work

Lot of researches has been carried out to discover dynamic replacement algorithms that can be applied over cache memories. As mentioned earlier, the more adaptive a replacement technique is, the better is the performance boost obtained (Hameed *et al*., 2013; Janapsatya *et al*., 2010). The replacement algorithm suggested by Odule and Osingua (2013) makes replacement decisions at runtime based on the changing workload patterns. Though it adapts to the spatial and temporal features of the workload, it operates at a page-level granularity rather than at a finer block-level granularity which can be crucial when the workload pattern changes very frequently.

Shared LLC is accessed by multiple cores. So it is imperative to have a good replacement algorithm running over it. When a miss is encountered here, the resulting overhead can be higher compared to other cache levels. Many works (Jaleel *et al*., 2008; Wu and Martonosi, 2011; Srikantaiah *et al*., 2008) have emerged in recent times which strive to improve the performance at LLC. Reineke and Grund (2010) have even explored the possibility of execution history such as the initial hardware state of the cache affecting the sensitivity of replacement algorithms applied over it.

The lines which will never be referenced by the processor in the future (dead-lines) need to be eliminated in order to increase cache space utilization factor (Soares *et al*., 2008). Counter based cache replacement and cache bypassing algorithm proposed by Kharbutli and Solihin (2008), works on this issue and tries to predict the dead-lines well in advance and choose them as victims. But with a constantly fluctuating workload, the process of prediction might become intricate and the accuracy can be lost.

Hence in this study we focus on designing a novel counter based replacement algorithm for shared LLC in a CMP environment which makes decisions at a fine-grained block level. To deal with fluctuating data access pattern, it logically partitions the cache into different zones using a 3-bit counter and consistently transports the elements from one zone to another zone in accordance with the input data set pattern and thus maximizes the hit rate compared to LRU algorithm.

The rest of the study is organized as follows.

Section 2 explains our replacement strategy in detail, section 3 describes the experimental setup, sections 4 and 5 analyze the obtained results and compare the performance with LRU and finally Section 6 summarizes the study.

2. COUNTER BASED LOGICAL CACHE PARTITIONING APPROACH

As discussed earlier, schemes like LRU, MRU, NRU might lead to sub-optimal performance for certain applications as they do not dynamically accustom to the changing workload patterns. Hence in this work we propose a cache replacement technique which is targeted towards the shared LLC. It uses a 3-bit counter to dynamically shuffle the cache elements and logically partition them into four different zones based on their likeliness to be referenced by the processor in the near future. This counter from here on will be referred to as Logical Cache Partitioning (LCP) counter. The lower and upper bounds for the counter are set to ‘0’ and ‘7’ respectively. The prediction about the likeliness of reference of the data items is made with the help of the hits encountered in the cache. As the number of hits for a particular element increases, it is moved up the zone list till it reaches the MLR region. Only if it stays unreferenced for quite an amount of time, it is evicted from the cache. Let us assume that the mapping policy used is the set associative mapping (Henessey and Patterson, 2011). **Table 1** shows the counter value range for each zone.

As their names imply, the zones are arranged in the decreasing order of their likeliness factor. Every cache line is associated with a LCP counter which is initialized with a value of ‘-1’. Any replacement policy consists of three phases-Replacement, Insertion and Promotion. Each of which is explained in the subsequent sections.

Table 1. Zone categorization

Counter value range	Zone
6-7	MLR
3-5	LR
1-2	LLR
0	NLR

2.1. Replacement

When a miss is encountered in the cache, the data item is fetched from the secondary memory and brought into the cache thereby replacing one of the elements which was already present in the cache. This element is often referred to as the victim. The process of selecting a victim needs to be efficient in order to improve the hit rate. In our method, the victim is selected from the zones in the increasing order of their likeliness factor. Any cache line which comes under the NLR zone is considered first for replacement. If no such line is found search is performed again to check if any element falls in the LLR region and so on till MLR is reached. If two or more lines possess the same counter value that is considered for replacement during that iteration, the line that is encountered first is chosen as the victim. Once the replacement is made, LCP counters of all the other data elements are decremented by ‘1’. This is done to carry out gradual zone demotion as discussed earlier to flush out unused data items from the cache.

2.2. Insertion

This phase is encountered as soon as the replacement candidate is found. The new incoming data item is inserted into the corresponding cache set and its LCP counter value is set to ‘2’ (LLR zone).

2.3. Promotion

A hit on any data item in the cache calls for the promotion phase. The LCP value associated with the cache line is incremented to the final value of its immediate upper zone. For example, if it was earlier in the LLR zone (LCP value ‘1’ or ‘2’), its LCP value is set to ‘5’ (i.e.,) the element now falls within the LR zone.

2.4. Boundary Condition Check

It is essential that the LCP counter value does not overshoot its specified range. Thus whenever it is modified, a boundary condition check is carried out to ensure that the value does not go below ‘0’ or beyond ‘7’.

2.5. Comparison with LRU

Consider the following data set:

. . .2 9 1 7 6 1 7 5 9 1 0 7 5 4 8 3 6 1 7 . . .

It can be seen that data items 1 and 7 occur frequently. **Table 2** shows the working of LRU and LCP on this data pattern. Assume that the cache can hold 4 blocks at a time. Incoming data items at that point of time are shown in the leftmost column. In the right most column, ‘m’ indicates a miss and ‘h’ indicates hit. Initially the cache contains invalid blocks. Counter values associated with all the blocks are set to -1. After applying both the techniques, LCP has resulted in 3 hits more than LRU. It is to be noted that frequently occurring data items 1 and 7 have resulted in hits towards the end unlike LRU. This is primarily because LRU follows the same approach irrespective of the workload pattern and tags 1 and 7 as ‘least recently used’ whereas LCP dynamically adjusts itself according to the access pattern change.

3. EXPERIMENTAL SETUP

For simulation purpose we have chosen an open-source, full system simulator called Gem5 (Binkert *et al.*, 2011) which is capable of simulating a variety of Instruction Set Architectures (ISAs). We make use of the Alpha ISA with 2 cores at 2 GHz clock frequency. Supported cache levels include a private L1 cache which is further sub-divided into instruction and data cache and an L2 cache which is shared between the available cores. The size of L1 and L2 cache are set to 64 kB and 2 MB respectively. Line size for both the caches is 64B. L1 cache is 2-way associative and L2 cache is 8-way associative.

3.1. Benchmark

Eight workloads have been selected from the princeton application repository for shared-memory computers (Bienia *et al.*, 2008; Gebhart *et al.*, 2009), a benchmark suite that comprises numerous large scale commercial multi-threaded workloads targeted towards CMP, to evaluate our method. Every workload is unique and their working set size varies considerably. **Table 3** highlights the key characteristics of all the PARSEC benchmarks used.

4. RESULTS

From here on, the term LCP will be used to represent our method in the graphs. Percentage increase in the overall number of hits at L2 is shown in **Fig. 1**. The x-axis represents each benchmark and the y-axis shows the percentage change in hits compared to LRU. **Figure 2** shows the total number of replacements made at L2. **Figure 3** shows the miss rate recorded by the individual cores and also the overall miss rate at L2. **Figure 4** shows the average L2 cache occupancy percentage for all the benchmarks.

Table 2. Working of LRU and LCP for the data set shown above

Input	Cache contents (LRU)				Hit/miss	Input	Cache contents (LCP)				Hit/miss
2	2				m	2	2_2	$_{-1}$	$_{-1}$	$_{-1}$	m
9	9	2			m	9	2_2	9_2	$_{-1}$	$_{-1}$	m
1	1	9	2		m	1	2_2	9_2	1_2	$_{-1}$	m
7	7	1	9	2	m	7	2_2	9_2	1_2	7_2	m
6	6	7	1	9	m	6	6_2	9_1	1_1	7_1	m
1	1	6	7	9	h	1	6_2	9_1	1_5	7_1	h
7	7	1	6	9	h	7	6_2	9_1	1_5	7_5	h
5	5	7	1	6	m	5	6_1	5_2	1_4	7_4	m
9	9	5	7	1	m	9	9_2	5_1	1_3	7_3	m
1	1	9	5	7	h	1	9_2	5_1	1_7	7_3	h
0	0	1	9	5	m	0	9_1	0_2	1_6	7_2	m
7	7	0	1	9	m	7	0_2	9_1	1_6	7_5	h
5	5	7	0	1	m	5	0_1	5_2	1_5	7_4	m
4	4	5	7	0	m	4	4_2	5_1	1_4	7_3	m
8	8	4	5	7	m	8	4_1	8_2	1_3	7_2	m
3	3	8	4	5	m	3	3_2	8_1	1_2	7_1	m
6	6	3	8	4	m	6	3_1	6_2	1_1	7_0	m
1	1	6	3	8	m	1	3_1	6_2	1_5	7_0	h
7	7	1	6	3	m	7	3_1	6_2	1_5	7_2	h

Table 3. Key characteristics of PARSEC benchmarks used

Program	Application domain	Working set
Blackscholes	Financial analysis	Small
Canneal	Computer vision	Medium
Dedup	Enterprise storage	Unbounded
Ferret	Similarity search	Unbounded
Fluidanimate	Animation	Large
Swaptions	Financial analysis	Medium
Vips	Media processing	Medium
X264	Media processing	Medium

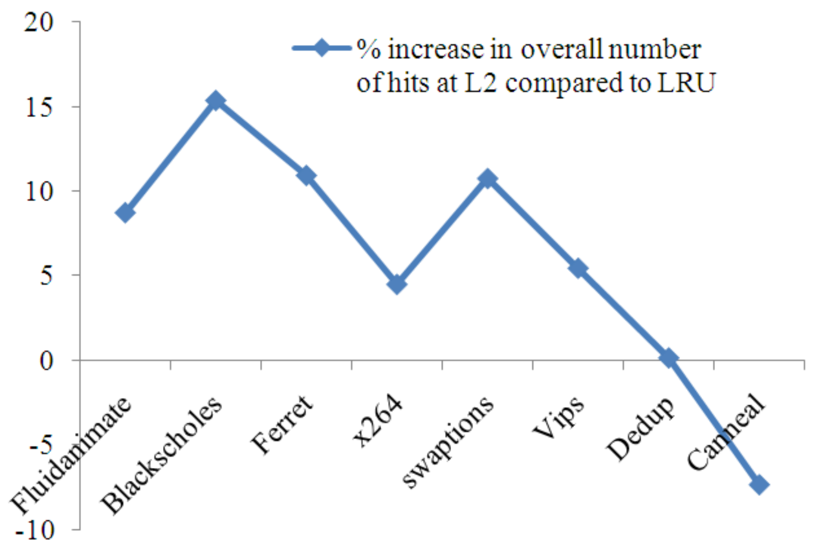

Fig. 1. Percentage increase in overall number of hits at L2 compared to LRU

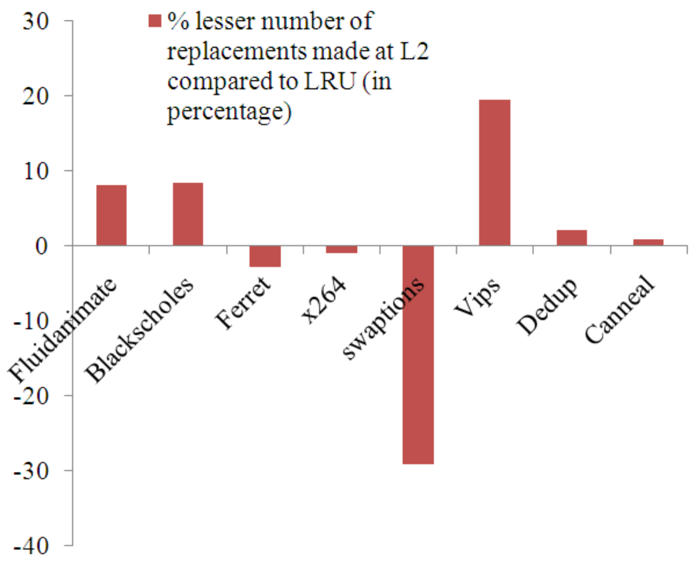

Fig. 2. Change in number of replacements made at L2

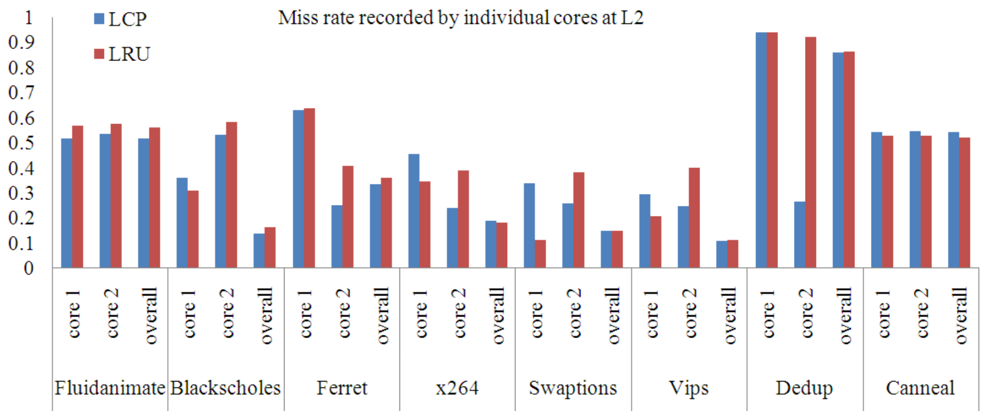

Fig. 3. Miss rate recorded by individual cores

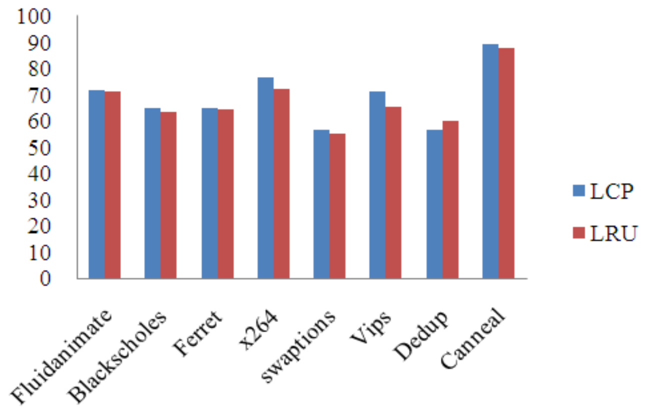

Fig. 4. Average cache occupancy percentage

5. DISCUSSION

Miss rate can be defined as the number of misses expressed as a fraction of total number of accesses. As it can be seen from **Fig. 3** there is reduction in the core-wise miss rate and overall miss rate across majority of the benchmarks when compared to LRU.

Apart from two workloads, all the others have shown significant improvement in hits. Percentage increase in hits varies from a minimum of 0.1% (Dedup) to maximum of up to 15.1% (Blackscholes) as shown in **Fig. 1**. Percentage increase in hits is computed from the difference between the number of hits in LCP and LRU.

Number of replacements reflects the efficiency of any replacement algorithm. A maximum of almost 20% decrease in the number of replacements can be observed across the given workloads compared to LRU from **Fig. 2**. Cache occupancy refers to the amount of cache that is being effectively utilized to improve the performance for any workload. **Figure 4** indicates cache utilization is marginally higher for majority of the benchmarks when LCP is applied compared to LRU.

6. CONCLUSION

In this study we have come up with a dynamic and a structured replacement strategy to be adopted across the LLC. Key points pertaining to our replacement policy goes as follows:

- Elements of the cache are logically partitioned into four zones based on their likeliness to be referenced by the processor with the help of a 3-bit LCP counter associated with every cache line. The minimum value that the counter can hold is ‘0’ and the maximum value is ‘7’
- Replacement candidates are chosen from the zones in the increasing order of their likeliness factor starting from the NLR zone. Initially all the counter values are set to ‘-1’. Conceptually all the blocks contain invalid data
- For every hit, the corresponding element is moved up by one zone by adjusting the LCP counter value. If it has reached the top most zone (MLR) the counter value is left untouched
- For every miss the counter value is decremented by ‘1’ to prevent stale data items from polluting the cache over a period of time. When a new data arrives, its LCP value is set to ‘2’. Boundary condition check needs to be applied whenever the LCP counter value is modified to make sure that it does not overshoot its designated range

Experimental results obtained by applying our method on PARSEC benchmarks have shown a marginal improvement of 7% in the overall number of hits and 3% improvement in the average cache occupancy percentage when compared to the conventional LRU approach.

6.1. Limitations

The number of bits used to represent LCP counter is kept to three owing to hardware limitations. Since it is has to be associated to with every cache line, increasing the number of bits might increase the hardware complexity. Benchmark evaluation has been carried out with a fixed set of system parameters (i.e.,) L1 and L2 cache sizes have been fixed to 64 kB and 2 MB respectively and the Instruction Set Architecture is chosen as Alpha.

6.2. Future Research

LCP is proven to have yielded good results at L2 cache level. Studies can be conducted to measure the performance by applying our method across all the available cache levels. Combining the current cache optimization techniques such as data prefetching along with LCP can produce really high performance.

7. REFERENCES

Bienia, C., S. Kumar, J.P. Singh and K. Li, 2008. The PARSEC benchmark suite: Characterization and architectural implications. Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, Oct. 25-29, ACM Press, New York, USA., pp: 72-81. DOI: 10.1145/1454115.1454128

Binkert, N., B. Beckmann, G. Black, S.K. Reinhardt and A. Saidi, 2011. The gem5 simulator. SIGARCH Comput. Archit., 39: 1-7. DOI: 10.1145/2024716.2024718

Gebhart, M., J. Hestness, E. Fatehi, P. Gratz and S.W. Keckler, 2009. Running PARSEC 2.1 on M5. The University of Texas at Austin.

Hameed, F., L. Bauer and J. Henkel, 2013. Dynamic cache management in multi-core architectures through run-time adaptation. Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, Mar. 12-16, IEEE Xplore Press, Dresden, pp: 485-490. DOI: 10.1109/DATE.2012.6176518

Henessey, J.L. and D.A. Patterson, 2011. Computer Architecture: A Quantitative Approach. 5th Edn., Morgan Kaufmann, ISBN-10: 9780123838728, pp: 856.

Jaleel, A., W. Hasenplaugh, M. Qureshi, J. Sebot and S. Steely *et al*., 2008. Adaptive insertion policies for managing shared caches. Proceedings of the 17th International Conference on Parallel Architectures And Compilation Techniques, Oct. 25-29, ACM Press, New York, USA., pp: 208-219. DOI: 10.1145/1454115.1454145

Janapsatya, A., A. Ignjatovic, J. Peddersen and S. Parameswaran, 2010. Dueling CLOCK: Adaptive cache replacement policy based on the CLOCK algorithm. Proceedings of the Design, Automation and Test in Europe Conference Exhibition, Mar. 8-12, IEEE Xplore Press, Dresden, pp: 920-925. DOI: 10.1109/DATE.2010.5456920

Kharbutli, M. and Y. Solihin, 2008. Counter-Based cache replacement and bypassing algorithms. IEEE Trans. Comput., 57: 433-447. DOI: 10.1109/TC.2007.70816

Odule, T.J. and I.A. Osinuga, 2013. Dynamically self-adjusting cache replacement algorithm. Int. J. Future Generat. Commun. Netw., 6: 25-25.

Reineke, J. and D. Grund, 2010. Sensitivity of cache replacement policies. Trans. Embedded Comput. Syst., 9: 39-39. DOI: 10.1145/2435227.2435238

Soares, L., D. Tam and M. Stumm, 2008. Reducing the harmful effects of last-level cache polluters with an os-level, software-only pollute buffer. Proceedings of the 41st Annual IEEE/ACM International Symposium Microarchitecture, Nov. 8-12, IEEE Xplore Press, Lake Como, pp: 258-269. DOI: 10.1109/MICRO.2008.4771796

Srikantaiah, S., M. Kandemir and M.J. Irwin, 2008. Adaptive set pinning: Managing shared caches in chip multiprocessors. Proceedings of the 13th International Conference on Architectural Support for Programming Languages Operating Systems, Mar. 01-05, ACM Press, New York, USA., pp: 135-144. DOI: 10.1145/1346281.1346299

Wu, C.J. and M. Martonosi, 2011. Adaptive timekeeping replacement: Fine-grained capacity management for shared CMP caches. ACM Trans. Archit. Code Optim. DOI: 10.1145/1952998.1953001