An Adaptive Insertion and Promotion Policy for Partitioned Shared Caches

To cite this article: Norfadila Mahrom et al 2018 IOP Conf. Ser.: Mater. Sci. Eng. 318 012048

View the article online for updates and enhancements.

Related content
- Evaluation of SuperLU on multicore architectures
  X S Li
- A Distributed Cache Update Deployment Strategy in CDN
  Xinhua E and Binjie Zhu
- Optical interconnection networks for high-performance computing systems
  Aleksandr Biberman and Keren Bergman
An Adaptive Insertion and Promotion Policy for Partitioned Shared Caches

Norfadila Mahrom¹,*, Michael Liebelt², Rafikha Aliana A. Raof³, Shuhaizar Daud¹ and Nur Hafizah Ghazali¹
¹School of Computer and Communication Engineering, Universiti Malaysia Perlis, Arau, 02600 Perlis, Malaysia
²School of Electrical and Electronic Engineering, Faculty of Engineering, Computer and Mathematical Sciences, The University of Adelaide, South Australia, 5005 Australia

E-mail: norfadila@unimap.edu.my

Abstract. Cache replacement policies in chip multiprocessors (CMP) have been investigated extensively and proven able to enhance shared cache management. However, competition among multiple processors executing different threads that require simultaneous access to a shared memory may cause cache contention and memory coherence problems on the chip. These issues also exist due to some drawbacks of the commonly used Least Recently Used (LRU) policy employed in multiprocessor systems, which are because of the cache lines residing in the cache longer than required. In image processing analysis of for example extra pulmonary tuberculosis (TB), an accurate diagnosis for tissue specimen is required. Therefore, a fast and reliable shared memory management system to execute algorithms for processing vast amount of specimen image is needed. In this paper, the effects of the cache replacement policy in a partitioned shared cache are investigated. The goal is to quantify whether better performance can be achieved by using less complex replacement strategies. This paper proposes a Middle Insertion 2 Positions Promotion (MI2PP) policy to eliminate cache misses that could adversely affect the access patterns and the throughput of the processors in the system. The policy employs a static predefined insertion point, near distance promotion, and the concept of ownership in the eviction policy to effectively improve cache thrashing and to avoid resource stealing among the processors.

1. Introduction
With the increasing number of applications simultaneously sharing the last level cache, cache contention and memory coherence issues have become one of the major focuses of processor architects. Competition among multiple applications for the last level cache (LLC) without efficient cache management may lead to an under-utilized system due to various forms of interference between the applications. However, cache replacement policy has been identified as one of the contributors to the overall improvement of system performance and has received much attention from researchers and computer architects. As the conventional LRU replacement policy is widely accepted as the standard replacement policy, many attempts have been made to improve the policy in respect of the three component policies of insertion decision, promotion and victim selection.
The commonly used LRU replacement policy employed in a multiprocessor system is inherited from uniprocessor systems. For that reason, some drawbacks of the LRU replacement policy are identified, which are generally due to lines or data that reside too long in the cache. The insertion of new lines at the highest priority position (i.e., the Most Recently Used (MRU) position) upon a miss has in turn caused zero-reuse lines and lines without high locality to spend a large amount of time occupying the cache [1][2]. On the other hand, the promotion strategy of the LRU replacement policy, which automatically moves hit lines to the MRU position, has also resulted in cache waste because it maximizes the time taken for dead lines to traverse from the MRU position to the lowest priority position (i.e., the LRU position) and eventually be evicted. A dead line refers to a cache line that is not reused from the time of its last access until it is evicted from the cache [1][8]. Even though the LRU policy is good at handling lines with high utility and temporal locality, other cores in the system have to wait longer before they can make use of the cache capacity while dead lines remain in the cache [8]. Nevertheless, the insertion policy and the associated promotion policy of LRU give each line a good chance of obtaining a hit while it traverses from the MRU to the LRU position, but may be inefficient for workloads in which the working set size is greater than the available cache space. This is because there are lines that are re-referenced immediately after they are replaced and this may lead to some problems such as cache thrashing, which can be improved by retaining some fraction of the working set in the cache long enough to generate more cache hits [3][4][11]. Therefore, many researchers have proposed a variety of techniques intended to improve the replacement policy so that better performance of the shared cache can be obtained.

In this paper, all three components of the replacement policy for a partitioned shared cache are investigated. The aim is to provide an adaptive LRU replacement policy that is targeted for a system where the shared cache has been carefully partitioned and distributed among multiple cores by a well-accepted cache partitioning scheme. It is expected that the new adaptive dynamic cache replacement policy for a partitioned shared cache could achieve better performance to support image processing analysis, with smaller hardware overhead and power consumption. It is also expected that the result could be used to assist pathologists to diagnose TB slides in a shorter time.

2. Motivation and Background

Earlier studies have proposed to insert a new line either at the MRU position or the LRU position. That is, when the new line is inserted at the MRU position, frequent accesses of the line while it traverses towards LRU position can derive additional hits, but it will also waste cache capacity in the case of dead-on-arrival lines. The dead-on-arrival lines is defined as lines that are inserted and used only once in the cache and will never be reused at all after they get evicted. This is as opposed to the zero-reuse lines, which will not be reused between insertion and eviction, but may be reused again at a greater distance (after the lines have been evicted from the cache). Therefore, by comparison, insertion at the LRU position has shown substantial benefits on the zero-reuse lines as the lines can be evicted immediately after they are used [3][7]. However, on every cache hit, both strategies automatically move all the hit lines to the MRU position. This may be good for cache lines with multiple-reuse, but not for single-reuse lines or lines with many references at greater distance (particularly after these lines become dead). Therefore, it would be desirable if the cache spaces that these lines are occupying are used by other lines to generate additional hits.

The pseudo insertion/promotion policy (PIPP) [1] is one of the cache replacement techniques that have been augmented from the traditional LRU policy, particularly in the context of insertion and promotion policies. The main differences between PIPP and the LRU policy are: (1) PIPP does not always insert incoming cache lines at the highest priority position, and (2) upon a cache hit, PIPP will not automatically promote the hit line to the highest priority position. Note that the utility-based cache partitioning (UCP) scheme [6] is used by PIPP to determine the allocation of cache ways for competing applications in the system. However, PIPP does not explicitly partition the shared cache as UCP does. Instead it uses the cache allocation information to implicitly partition the cache. That is, the number of ways allocated to an application is used to specify the priority position for the application to
install all new incoming lines. The sequence order of the insertion position employed by PIPP starts from the lowest priority position to the highest priority position. Upon cache hits, PIPP promotes hit lines by a single priority position towards the highest priority position with a certain probability, ensuring that the priority increment remains unchanged. The promotion probability is used to prevent streaming applications from hurting the performance of non-streaming applications that are running in the system, as the streaming accesses into the cache are very unlikely to be reused. Therefore, the value of the promotion probability of a streaming application is set to be much smaller (e.g. $P_{\text{stream}}=1/64$) than the promotion probability of a non-streaming application (e.g. $P_{\text{prom}}=3/4$), so that a better chance can be given to the lines of the non-streaming applications to reside longer in the cache. The value of the promotion probability can be any reasonable values as [1] has proven that a wide range changes to the probability values has resulted in small overall performance improvement to the system.

Figure 1 illustrates the operation of the insertion policy of PIPP, in which the UCP has allocated 5 ways to Core1 and 3 ways to Core2. Thus, all new lines of Core1 are inserted at priority position 5 while all new lines of Core2 are installed at position 3. In PIPP, evictions always choose the line at the lowest priority position. Therefore, on every cache miss, the miss-causing core can evict cache lines that belong to another core in the system. This allows cache capacity stealing among the cores, which may result in an inefficient strategy in maintaining the cache ways that have been allocated to the cores.

It is observed that the insertion policy of PIPP seems to treat zero-reuse lines more effectively than the LRU policy because the lifetime of the zero-reuse lines can be reduced due to a shorter distance to traverse towards the lowest priority position for eviction. Additionally, the single priority position promotion of PIPP enables the single-reuse lines and the lines with a burst of references to occupy the cache long enough before they become dead, but will quickly become a potential victim once they are ready to be replaced. However, PIPP’s pseudo insertion/promotion policy does not explicitly enforce the cache partitioning and may lead to a large number of lines being taken away.
from any core(s) in the system to be used by another core(s). This is because PIPP only stimulates the system to create results similar to the hard/strict partition (i.e. the number of ways allocated to each core for every execution intervals will remain unchanged), so that the cache capacity stealing allowed by PIPP among all the applications may produce benefits in the utilization of the available cache resources. Hence, it is concluded that uncontrolled line stealing among cores by PIPP may severely degrade the system performance.

Therefore, this paper proposes algorithms that are intended to strictly partition the shared cache and explicitly enforce the original way-partitioning decisions. One may say that some of the cache capacity allocated to the cores may be under-utilized, but such a situation is not likely to happen if the cache is properly partitioned by the cache partitioning scheme. Moreover, if all components of the cache replacement policy are improved accordingly, the occupancy of the partitioned shared cache can be controlled thereby improving the overall performance of the system.

Since UCP is widely accepted, it is used in this paper with the implementation of an adaptive LRU replacement policy that has been adopted from the concept of PIPP. The adaptive insertion and promotion strategies of the new policy are augmented from the concept introduced by PIPP in order to maintain the benefits received by zero-reuse lines, single-reuse lines and multiple-reuse lines at a short distance. Additionally, the eviction policy of the traditional LRU policy will be improved so that it better satisfies the requirements of a multiprocessor system, such that the cache capacity stealing can be controlled among the cores.

3. Adaptive Insertion and Promotion Policy

In this section, the concept of adaptive insertion and promotion policy of the new replacement policy is presented. Then, an example to detail how the new policy works in a partitioned shared cache is provided.

3.1 Middle Insertion 2 Positions Promotion Policy (MI2PP)

The MI2PP policy is proposed with the objective of providing better thrashing resistance to a partitioned shared cache by maintaining the number of cache ways received by all cores equal to the number of ways that have been allocated originally. The policy is targeted to improve the insertion and promotion strategies of a cache replacement policy so that the cache occupancy of zero-reuse and single-reuse lines is not wasted, while multiple-reuse lines could be retained in the cache long enough to incur additional cache hits before eviction. In other words, the MI2PP policy tries to retain as many of the beneficial cache lines as possible, so that they could manifest more cache hits in the allocated cache space and minimize the residency of zero-reuse and single-reuse lines in the allocated space without yielding significant hardware overhead. Therefore, the MI2PP policy chooses to insert the new lines at the middle position of the priority stack based on the associativity of the shared cache and the number of cores in the system.

Thus:

$$\text{InsertionPosition} = \frac{\text{SetAssociativity}}{\text{NumberOfCores}}$$  \hspace{1cm} (1)

The MI2PP insertion may not be as good as LRU insertion in minimizing the residence time of the zero-reuse lines in the cache. However, it could reduce the time spent by the zero-reuse lines by approximately half compared to those installed at the MRU position. MI2PP could also reduce the number of misses incurred due to dead-on-arrival lines. In addition, for the lines with single-reuse and multiple-reuse, they can be protected and have more chances for further hits while they traverse from the middle position to the LRU position. This is in contrast to the immediate disadvantage if they are inserted at the LRU position. However, if the next references are not in the near-future, LRU insertion will remain a better policy than MI2PP.
On a cache hit, MI2PP implements a different promotion policy from that of LRU. The hit line will automatically be moved by two priority positions towards the highest priority position. The reason for this change is that if the lines were automatically shifted to the MRU position, as in the LRU policy, single-reuse and multiple-reuse lines at a greater distance may consume cache capacity without providing any benefit. Hence, MI2PP will shorten the distance for the dead lines to traverse towards the LRU position. This means that the lines will have potential to be evicted sooner than in the conventional MRU promotion policy.

The MI2PP promotion strategy will also slow down the movement of the lines with multiple-reuse towards the MRU position if the next accesses of other cache lines happen at priority positions lower than the positions of the multiple-reuse lines, providing the multiple-reuse lines with more opportunity to derive additional hits along the way. Given that all new lines are installed at the middle priority position and get promoted by only two positions upon cache hits, this approach does not cause any disadvantage to the many frequent access lines. This is because these lines are preserved at priority positions which are high enough for them to be kept in the cache long enough to generate more hits before eviction.

Since the target of the MI2PP policy is to ensure that the number of cache ways received by all cores is equal to the number of ways they have been allocated, victim selection for the cache line replacement is modified by taking into account the total number of ways belonging to the miss-causing application/core. If the miss-causing core gets more ways than it is allocated at the time of the cache miss, the LRU line of the core will be selected as an evictee. Otherwise, the LRU line in the set that does not belong to the miss-causing core will be replaced. This technique is used to enforce the cache partition and helps to preserve the benefits of partitioning the cache by sustaining adequate cache resources among all the cores.

In summary, the goal of MI2PP policy was to provide an effective yet not so complex cache replacement policy that works well in a partitioned shared cache, since an optimal cache partitioning scheme used in the system may cost significant hardware overhead. Therefore, MI2PP policy uses information already available from the baseline LRU replacement policy and exploits the information to make better use of the partitioned shared cache resources to gain better overall system performance.

### 3.2 Implementation Example

Since MI2PP policy was adapted from PIPP, a simple example used in [1] is presented in Figure 2 to deliver a detailed explanation of MI2PP. Figure 2(a) illustrates a simple operation of PIPP, while Figure 2(b) demonstrates the operation of MI2PP policy on a dual-core system.

The two cores, namely core0 and core1 are allocated with five and three cache ways, respectively. To differentiate the lines occupied by the cores, the cache lines of core0 are represented by numerals in squares, while letters in black circles represent cache lines of core1. For both of the policies, the insertion position of each core is shown in the figure. For PIPP, when core1 makes a first request for line D which is not in the cache, the line is installed at position 3. Similarly with core0, when references to line ζ and η during the second and third requests incurred cache misses, both lines are inserted at position 5. Conversely, MI2PP always inserts all the new lines requested by any core at the same location, which is at position 4. When the next reference of line D (i.e., the fourth request) by core1 creates a hit, PIPP will automatically move the line by a single position. On the other hand, the line will be shifted slightly higher by MI2PP, which is by two priority positions.

One of the significant differences between the two policies that can be observed is in the context of cache capacity management. In some execution intervals, it can be seen that PIPP demonstrates cache allocation deviation. This is due to the highlighted fact that the policy does not explicitly enforce the cache partitions, but only pseudo-partitions the cache. On the contrary, MI2PP strictly enforces the cache partitions and always maintains the target allocation of 5 lines to core0 and 3 lines to core1 in every execution interval.
To better understand the similarities and differences between PIPP and MI2PP policy, this discussion is narrow down to the operation of both policies towards zero-reuse, single-reuse and multiple-reuse lines. Assume that line ζ that belongs to core0 does not exhibit any reuse. PIPP inserts the line at position 5, while MI2PP installs at position 4. After several hits and misses, MI2PP eliminates line ζ from the cache more quickly than PIPP. This shows the insertion at priority position 4 (at the middle of the priority stack) can reduce the lifetime of dead lines in the cache. However, if MI2PP is compared with the LRU insertion such as implemented in thread-aware dynamic insertion policy (TADIP) [3], MI2PP is not efficient in minimizing the age of the dead lines although it has been shown in Figure 2 to perform better than PIPP. This is because the LRU insertion in TADIP will immediately evict zero-reuse lines from the cache upon next cache access as all new incoming lines will be inserted at the LRU position.

Next, consider line η as a single-reuse line which is reused while it is still in the cache (i.e., the tenth request). MI2PP will promote the line by two priority positions, which is slightly higher than PIPP. However, the lower insertion at position 4 by MI2PP helps to push the line downwards to the end of the priority order sooner than PIPP. This shows the insertion at priority position 4 (at the middle of the priority stack) can reduce the lifetime of dead lines in the cache. However, if MI2PP is compared with the LRU insertion such as implemented in thread-aware dynamic insertion policy (TADIP) [3], MI2PP is not efficient in minimizing the age of the dead lines although it has been shown in Figure 2 to perform better than PIPP. This is because the LRU insertion in TADIP will immediately evict zero-reuse lines from the cache upon next cache access as all new incoming lines will be inserted at the LRU position.

Figure 2. Example operation of (a) PIPP from Xie and Loh [2009], and (b) MI2PP policy for a variety of insertions and promotions activities. Assume evictions for both policies always choose the lowest-priority cache line in the set.
insertion at the middle position is an advantage to shorten the lifetime of the single-reuse line after the line has been reused.

Finally, after line D is brought into the cache, MI2PP keeps the multiple-reuse line long enough to be exposed to cache hits on every next reference. On the other hand, PIPP eliminates the line more quickly than MI2PP after the line has been first reused, which costs a cache miss upon second reference to the line. The treatment of line D illustrates how the promotion by two priority positions of MI2PP can be better than PIPP in maximising the amount of time for multiple-reuse lines residing in the cache. Overall, the example demonstrates that MI2PP can save more cache misses compared to PIPP.

4. Conclusion

The implementation of MI2PP policy will be done using Qsys System Integration Tool. Then, the multiprocessor system will be evaluated by using the Altera DE1-SoC development board that is equipped with the Cyclone V FPGA with ARM Cortex-A9. For all of the MI2PP evaluations, the performance metrics that will be used to quantify the performance of the system are the \( IPC \) throughput, weighted speedup which indicates reduction in execution time, and the harmonic mean of weighted speedup which accounts for and balances both fairness and performance of the system \([5][12]\).

Let \( IPC_i \) denote the IPC of the \( i \)th application, \( SingleIPC_i \) the stand alone IPC of the same application if it is executed in isolation, and \( N \) the number of processes the system executes concurrently. The formulae for the aforementioned metrics are:

\[
\text{IPC throughput} = \sum_{i=0}^{N} IPC_i
\]

(2)

\[
\text{Weighted speedup} = \sum_{i=0}^{N} \frac{IPC_i}{SingleIPC_i}
\]

(3)

\[
\text{Harmonic mean} = \frac{N}{\sum_{i=0}^{N} \left( \frac{SingleIPC_i}{IPC_i} \right)}
\]

(4)

It is expected that the new adaptive dynamic cache replacement policy for a partitioned shared cache in system will increase the hit rate of the shared cache and the instructions per cycle executed by the multiple processors in the system. Hence, the replacement policy could support the image processing analysis that required massive memory utilization, as well as reducing hardware overhead and saving power consumed by the shared cache. In future, more review will be done to further strengthen this research. Quantitative analysis of different techniques and policies used to achieve better performance of the multiprocessors also will be done.

Acknowledgements

The authors wish to thanks Universiti Malaysia Perlis for funding this research project under the Short Term Grant Scheme (Grant No. 9001-00568).

References

1. Y. Xie and G.H. Loh. (2009). PIPP: promotion/insertion pseudo-partitioning of multi-core shared caches. ACM SIGARCH Computer Architecture News, 37(3), pp. 174-183.
2. C. Li, D. Wang, Y. Xue, H. Wang and X. Zhang, “Enhanced adaptive insertion policy for shared caches,” in *Advanced Parallel Processing Technologies*, 2011, pp. 16-30.

3. A. Jaleel, W. Hasenplaugh, M. Qureshi, J. Sebot, S. Steely Jr and J. Emer, “Adaptive insertion policies for managing shared caches,” in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, 2008, pp. 208-219.

4. M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely and J. Emer. (2007). Adaptive insertion policies for high performance caching. *ACM SIGARCH Computer Architecture News, 35*(2), pp. 381-391.

5. K. Luo, J. Gummaraju and M. Franklin, “Balancing throughput and fairness in SMT processors,” in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2001, pp. 164-171.

6. M. K. Qureshi and Y. N. Patt, “Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches,” in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, 2006, pp. 423-432.

7. V. Mekkat, A. Holey, P. C. Yew and A. Zhai, "Managing shared last-level cache in a heterogeneous multicore processor," in *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques*, Edinburgh, 2013, pp. 225-234.

8. Cong Thuan Do, Hong-Jun Choi, Jong Myon Kim, Cheol Hong Kim. A new cache replacement algorithm for last-level caches by exploiting tag-distance correlation of cache lines. *Microprocessors and Microsystems, Volume 39, Issue 4*, 2015, pp. 286-295.

9. Terasic Technologies Inc. (2016, August 24). DE1-SoC User Manual. Terasic Technologies Inc.

10. K. Veropoulos, C. Campbell, Learmonth, G., B. Knight, and J. Simpson (1998), “The Automated Identification of Tubercle Bacilli Using Image Processing and Neural Computing Techniques,” in *Proceeding of the 8th International Conference on Artificial Neural Networks*, Skövde, Sweden, pp. 797-802.

11. J. Wu, X. Sui, Y. Tang, X. Zhu, J. Wang and G. Chen, “Cache management with partitioning-aware eviction and thread-aware insertion/promotion policy.” in *International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, 2010, pp. 374-381.

12. A. Jaleel, H. H. Najaf Abadi, S. Subramaniam, S. C. Steely and J. Emer. (2012, March). "CRUISE: cache replacement and utility aware scheduling," in *ACM SIGARCH Computer Architecture News* (Vol. 40, No. 1, pp. 249-260).

13. G. Jia, G. Han, J. Jiang and L. Liu, "Dynamic Adaptive Replacement Policy in Shared Last-Level Cache of DRAM/PCM Hybrid Memory for Big Data Storage," in *IEEE Transactions on Industrial Informatics*, vol. 13, no. 4, pp. 1951-1960, Aug. 2017.