Fully Dynamic Insertion and Promotion Policy

Minsik Oh, Byunghoon Lee, Kwangsu Kim, Eui-Young Chung
Department of Electrical and Electronic Engineering
Yonsei University, Seoul, Korea
stomlions@dtl.yonsei.com, {stomlions, bh2, kskim}@dtl.yonsei.ac.kr, eychung@yonsei.ac.kr

Abstract

As the number of cores integrated in chip multi-processors (CMPs) increases, the importance of last-level cache becomes prominent due to increased resource sharing. In order to efficiently manage shared resources, researches related to cache replacement policy have been steadily performed. In this work, we suggest a fully dynamic cache line insertion and promotion policy to dynamically adjust cache line allocation. The experiment results show that the proposed replacement policy improves the Miss per Kilo-Instructions(MPKI) by about 5.2% and 7.1% and the Instruction per Cycles(IPC) by about 2.2% and 2.5% compared to the baseline LRU replacement policy in dual-core and quad-core system, respectively.

Keywords: Insertion and promotion policy, replacement policy, cache management.

1. Introduction

In chip multiprocessor(CMP) architectures, shared last-level cache is adopted in order to maximize cache utilization by sharing resources. As the number of cores increases, so does the importance of shared last-level cache(LLC) management in order to satisfy performance requirement. In order to improve cache utilization, cache replacement policy has been progressively studied in academia [1-6, 8]. In general, cache replacement schemes consist of three parts: insertion, promotion and eviction.

An insertion policy which selects cache lines' position in LRU is suggested to filter low reusability workload out [1]. However, this replacement policy does not take into account the situation of cache contention issue which is harsh to system performance. To mitigate this problem, Xie et al. [2] suggested a utility-based pseudo-partitioning technique to dynamically orchestrate the insertion point per core. However this scheme results in a degradation of cache utilization, because the sum of insertion point should be equal to the associativity of the cache in the scheme. In case of promotion policy, the most of works used general LRU or single-step

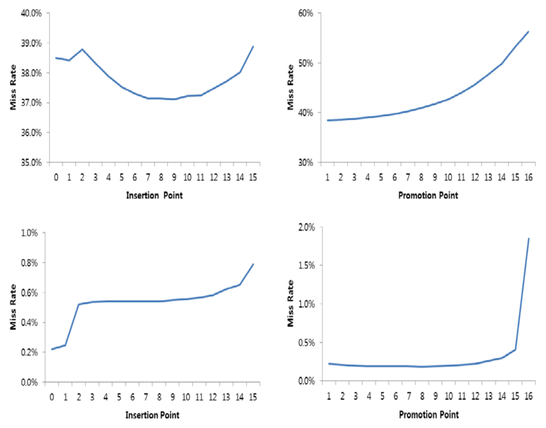

Figure 1: Miss rate curves for the position of insertion and promotion points. (a) bzips (upper graphs) (b) soplex (lower graphs)

promotion which is struggle to go upstream through allocated other cache lines gradually. The author in [6] proposed promotion policy using re-reference interval prediction scheme, but it only control promoted degree based on access frequency. There has also been a study [8] to differentiate long term cache lines and short term cache lines to dynamically adjust the promotion point. However, this work only focuses on preserving long term cache lines, and cannot reflect the situations of system performance degradation.

To our knowledge, there is no previous work to fully adjust both insertion and promotion position in a dynamic manner. Most studies only focus on insertion policy with empirically deduced promotion policies or promotion policy without considering workload characteristics. In this work, we suggest a fully dynamic insertion and promotion policy to manage LLC to adjust cache line allocation dynamically to improve the system performance.

2. Motivation

The objective of a cache replacement policy is to efficiently conserve useful cache line to improve cache utilization. To satisfy the goal, it is important to allocate each cache line to an appropriate position to prevent useful cache lines from evicted by the ones with low reusability within their lifetime.

Figure 1 is the examples of LLC miss rate with different position of insertion and promotion points in a single-core systems. This graph is obtained using bzip2 and soplex in SPEC2006 benchmark suite. Insertion and promotion points indicate the position in LRU stack, e.g. point ‘0’ indicates MRU position and ‘15’ indicates LRU position. In Figure 1 (a), LLC miss rate for insertion point shows that cache utilization is better between LRU and MRU point, unlike promotion point. Intuitively, it should be beneficial performance-wise to adjust the insertion point to 6~8 and promotion point to 0. While, in Figure 1 (b), miss rate is not sensitive to the insertion point, but to the promotion point at nearby MRU position. For workloads with similar characteristic, it will be beneficial to offer MRU position to promotion point rather than insertion point.

Through this observation, it is apparent that previous work only focuses on insertion policy or promotion policy which cannot consider the case of benefiting from adjusting along with the insertion and promotion point. In this work, we propose a fully dynamic insertion and promotion replacement policy based on this observation.

3. Dynamic Insertion/Promotion Policy

In this section, we introduce the proposed insertion and promotion policy.

The proposed policies are based on the following characteristics of computational workloads. Kaseridis et al. in [3] categorized workloads into the following three categories: cache-friendly, cache-fitting, cache-thrashing and streaming workloads. First of all, in a cache-friendly workload, also referred as a capacity-sensitive workload, the performance of the workload is improved over the increase in cache capacity. In cache-fitting workload, on the other hand, the performance is saturated when a certain cache capacity reaches a sufficient. Finally, in case of cache thrashing or streaming workload, due to the large working set size or low reusability of the data, the change in the cache capacity incurs negligible to no performance improvement. Due to the aforementioned characteristics, cache-friendly and cache-fitting workloads have a certain capacity range in which capacity-sensitivity is maximized, so it is important that the data from these workloads is stored near MRU position and low-reuse data from cache-thrashing and streaming workloads is stored near LRU position to prevent relevant data in LLC from being evicted, taking account of cache contentions between processes.

Our proposed fully dynamic insertion and promotion policy, in a certain interval, moves the data towards MRU position or LRU position with respect to the hit count change incurred by the adjustment of insertion and promotion point from the previous interval. That is, if the performance gain/loss of a process is highly affected by the change in the hit count, the insertion and promotion point would move towards MRU position, or move towards LRU position otherwise.

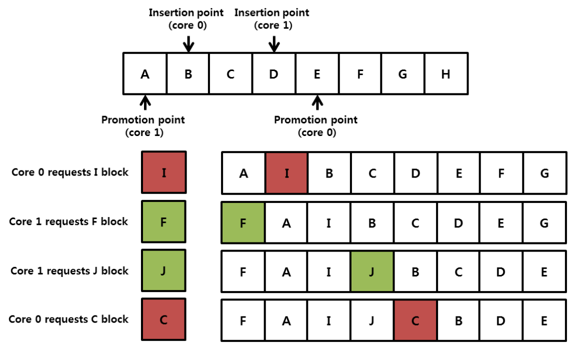

Figure 2: Example of the operation of insertion and promotion policy

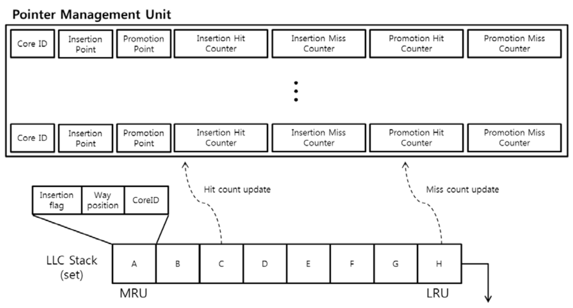

Figure 3: Insertion and promotion pointer management Unit

Figure 2 depicts an example showing the basic operation of the proposed dynamic insertion and promotion policy. Each core has its dedicated insertion point and promotion point, and adjusts them independently using the point positioning algorithm discussed later in this paper. When a cache miss occurs after a tag access, a new block(I, J) is allocated to the cache line pointed by the requesting core’s insertion point. When a cache hit occurs, the target block(C, F) references the promotion point and is migrated to the pointed position.

We must configure the architecture as to control each insertion point and promotion point. For that, to decision and query to insertion and promotion point, we set the pointer management unit which consists of insertion/promotion point information and hit/miss counters for each cores. A 1-bit insert flag is also added to each cache line to differentiate insertion area occurred by initially allocated cache lines and a promotion area where the cache lines are reused. As for the insertion point, insert flag is checked upon cache hit form LLC access. We check whether the block is in the insertion area or the promotion area, then check which core the block was allocated by to increase the corresponded hit count in either the insertion stat or promotion stat. This is in order to estimate how many cache hits will occur when a new or promoted cache block when placed in a certain LRU stack position. For an essentially same reason, when an evicted block is selected, we check the block's core ID and estimate how many misses will occur when data is stored to the defined insertion/promotion point. Based on the above scheme, we get a performance indication to each insertion and promotion point.

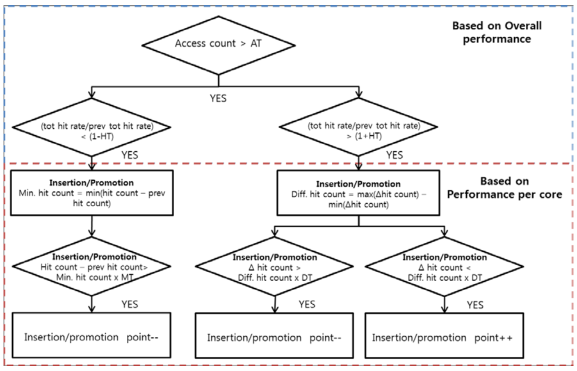
Based on Overall performance
Access count > AT
YES
(tot hit rate/prev tot hit rate) < (1-HT)
YES
(tot hit rate/prev tot hit rate) > (1+HT)
YES
Based on Performance per core
Insertion/Promotion Min. hit count = min(hit count – prev hit count)
Insertion/Promotion Diff. hit count = max(Δhit count) – min(Δhit count)
Insertion/Promotion Hit count – prev hit count> Min. hit count x MT
YES
Insertion/Promotion Δ hit count > Diff. hit count x DT
YES
Insertion/Promotion Δ hit count < Diff. hit count x DT
YES
Insertion/promotion point--
Insertion/promotion point--
Insertion/promotion point++

Figure 4: Insertion and Promotion point decision method

The proposed method configures the insertion and promotion point by core, based on the statistics from the pointer management unit through the flow depicted in Figure 4. There are 2-stages to decide insertion and promotion point using overall hit rate in cache memory and difference of hit count per core.

The proposed dynamic insertion and promotion policy operates in an interval-based manner and the interval is defined by the number of LLC accesses. At first stage, when the number of LLC accesses exceeds the pre-defined access threshold(AT), total hit rate is calculated using the hit/miss count monitored per-core. If total hit rate has decreased by a certain hit ratio threshold(HT) from the total hit rate calculated from the previous interval, hit count of each insertion and promotion area is calculated to compensate for the performance degradation caused by reduced capacity from the point adjustment at second stage. Let Min. hit count refer to the largest hit count loss from every area. A point which causes a hit count loss within a certain portion of Min. hit count(MT) will be reduced and moved to MRU position in order to increase the alive time of the data. On the other hand, if total hit rate has increased by certain rate(HT) from the total hit rate calculated in the previous interval, cache-friendly and cache-fitting workload which incurs a large difference in hit count is identified from cache-thrashing and streaming workload, and point is adjusted accordingly. First, hit count difference is calculated in each insertion and promotion area and the difference between the largest value and the smallest value is computed (diff. hit count). This is to give the processors running cache-friendly or cache-fitting workload priority in obtaining cache capacity. Near maximum hit count difference region, or near diff. hit count region, we reduce the point to move the cache line towards MRU position, and near minimum hit count difference region, we increase the point to move the cache line towards LRU position. The region is defined using a differential threshold value (DT).

4. Experiment Evaluation

To evaluate the proposed replacement policy, we use GEM5[7] full system simulator. Our processor configuration is 2-core and 4-core both operating at 2GHz. Each core has 32kB, 8way instruction cache and data cache. They also have 1MB, 16-way associative L2 cache per core. To simulate multi-programmed workloads, we use combinations of workloads from SPEC CPU2006 benchmark suite, which are selected according to their characteristics: cache friendly, cache fitting, cache thrashing and streaming. To verify the performance of the proposed replacement policy with other policies, we implement Thread-aware Adaptive Insertion Policy (TADIP) [1], Promotion/Insertion Pseudo-Partitioning (PIPP) [2], and Lifetime-aware LRU Promotion Policy [8] and compared the performance, along with the baseline LRU policy. The experiment is conducted during 200-billion ticks.

Figure 5 compares the normalized MPKI and IPC throughput among the selected replacement policies for 2-cores system. X-axis indicates workload characteristics, 'F-cache friendly', 'I-cache fitting', 'T-cache thrashing', 'S-streaming', and combined into mixed multi-programs. In case of PIPP, A loss of capacity results in degradation of cache utilization in all cases. Even satisfying capacity, insertion policy biased to LRU or MRU position, i.e. TADIP, cannot be optimized performance at workloads included cache friendly and fitting workload. While lifetime-aware policy only focus on promotion effect without consideration of performance loss from shorten inserted cache line, proposed replacement policy considered not only insertion position but promotion position improves cache utilization considerably in almost all cases except the workloads included cache thrashing which yields constant reusability regardless of capacity. Through this experiment, we observed that the proposed replacement policy yields MPKI improvement of about 5.2% and 2.8%, and IPC improvement of about 2.2% and 0.4% compared to the baseline LRU policy and TADIP, which outperforms PIPP and lifetime-aware policy on average, respectively.

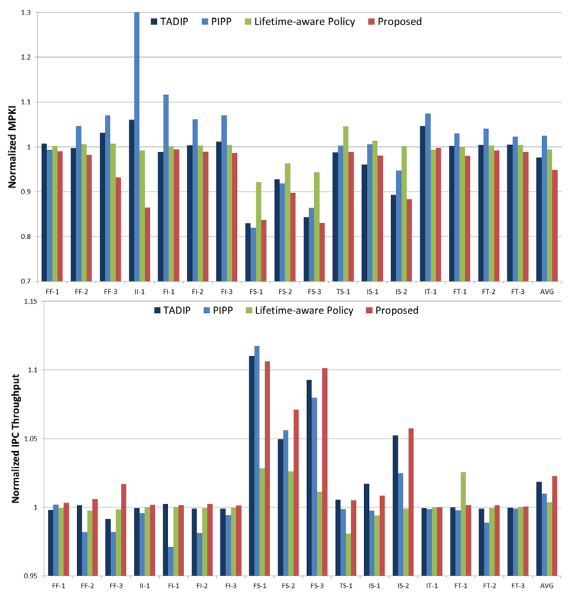

Figure 5: 2-core experiment result. (a) normalized MPKI (upper graphs) (b) normalized IPC (lower graphs)

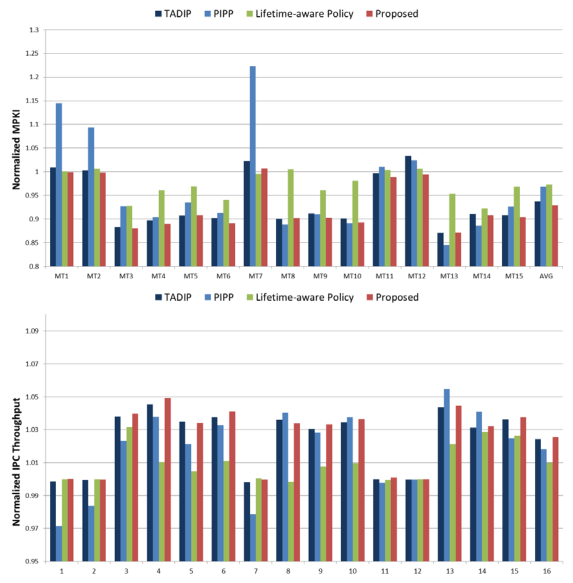

Figure 6: 4-core experiment result. (a) normalized MPKI (upper graphs) (b) normalized IPC (lower graphs)

In Figure 6 which shows the experiment results performed in a 4-core system, the workloads are randomly composed of a variety of workloads with distinct characteristics. We find that the proposed replacement policy outperforms LRU policy by about 7.1% and 2.5% on average and exhibits a slight improvement of 0.8% and 0.2% over TADIP on average in MPKI and IPC, respectively. TADIP can allocate cache lines which are not expected to be reused in near-future, to LRU position immediately, whereas our proposed replacement policy will move the affected cache lines to LRU position gradually. For this reason, experiments performed with cache thrashing or streaming workloads severely limit the overall performance improvement despite the improvement of many others.

5. Conclusion

In this paper, we proposed a fully dynamic insertion and promotion replacement policy targeting LLC in CMPs, to improve system performance. We demonstrated that adjustment of not only insertion point but also promotion point will improve cache utilization and provided insights related to it. Our experiment is conducted using various combinations of workloads through which we were able to explore large real-application area.

However, to prevent cache performance degradation, the proposed replacement policy adjusts insertion and promotion point step-by-step, gradually. We will further improve out replacement policy with new techniques including a method of program phase change detection and a more flexible adjustment of points as future work.

6. Acknowledgement

This work was supported by the ICT R&D program of MSIP/IITP, [2016 (R7177-16-0233), Development of Application Program Optimization Tools for High Performance Computing Systems] and LG Electronics

7. References

[1] Aamer Jaleel, William Hasenplaugh, Moinuddin Qureshi, Julien Sebot, Simon Steely Jr., Joel Emer, “Adaptive Insertion Policies for Managing Shared Caches”, *PACT*, October 2008.

[2] Yuejian Xie and Garbriel H. Loh, “PPIP: Promotion/Insertion Pseudo-Partitioning of Multi-Core Shared Caches”, *ISCA 2009*.

[3] Dimitris Kaseridis and Muhammad Faisal lqbal, “Cache Friendliness-Aware Management of Shared Last-Level Caches for High Performance Multi-Core Systems”, *TC*, Vol. 63, pp. 874-887, April 2014

[4] Aamer Jaleel, Kevin B. Theobald, Simon C. Steely Jr. and Joel Emer, “High Performance Cache Replacment Using Re-Reference Interval Prediction (RRIP)”, *ISCA, June 2010*

[5] Fang Juan and Li Chengyan, “An Improved Multi-core Shared Cache Replacement Algorithm”, *DCABES*, pp. 13-17, October 2012

[6] Gangyong jia, Xi Li, Chao Wang, Xuehai Zhou and Zongwei Zhu, “Cache Promotion Policy using Re-Reference Interval Prediction”, *CLUSTER*, pp. 534-537, September 2012

[7] N. Binkert, el al., “The gem5 Simulator”, *ACM S’IGARCH computer Architecture News*, Vol. 39, No. 2, pp. 1-7, May 2011

[8] Hong-Yi Wu, el al., “Lifetime-aware LRU Promotion Policy for Last-level Cache”, *VLSI-DAT*, pp. 1-4, April 2015