International Journal of Networking and Computing – www.ijnc.org, ISSN 2185-2847
Volume 9, Number 2, pages 161–170, July 2019

A Cache Replacement Policy with Considering Global Fluctuations of Priority Values

Jubee Tada
Graduate School of Science and Engineering, Yamagata University
Yonezawa, Yamagata, 992-8510, Japan

Received: February 13, 2019
Revised: May 4, 2019
Accepted: May 29, 2019
Communicated by Minoru Uehara

Abstract

In the high-associativity caches, the hardware overheads of the cache replacement policy become a problem. To avoid this problem, the Adaptive Demotion Policy (ADP) is proposed. The ADP focuses on the priority value demotion at a cache miss, and it can achieve a higher performance compared with conventional cache replacement policies. The ADP can be implemented with small hardware resources, and the priority value update logic can be implemented with a small hardware cost. The ADP can suit for various applications by the appropriate selection of its insertion, promotion and selection policies. If the dynamic selection of the suitable policies for the running application is possible, the performance of the cache replacement policy will be increased. In order to achieve the dynamic selection of the suitable policies, this paper focuses on the global fluctuations of the priority values. At first, the cache is partitioned into several partitions. At every cache access, the total of priority values in each partition is calculated. At every set interval, the fluctuations of total priority values in all the partitions are checked, and the information is used to detect the behavior of the application. This paper adapts this mechanism to the ADP, and the adapted cache replacement policy is called the ADP-G. The performance evaluation shows that the ADP-G achieves the MPKI reductions and the IPC improvements, compared to the LRU policy, the RRIP policy and the ADP.

Keywords: Memory Hierarchy, Cache Memory, Cache Replacement Policy

1 Introduction

In recent computer systems, the amount of handling data is increased as the improvement of big data analysis or artificial intelligence. According to this increase, a large part of energy consumption is occupied by data transfer in the memory system. Therefore, to achieve the sustainable computer system, computer architects should decrease data transfer.

In order to decrease data transfer in the architecture side, it is needed to improve the performance of the cache. A replacement policy is important because that directly affects the performance of the cache. However, even if the high performance replacement policy is realized, the energy consumption of the computer system will be increased when the policy requires too much hardware cost. Therefore, to decrease the energy consumption of the memory system, a high performance replacement policy which can be realized by the small hardware overheads is needed.

Although the LRU policy is known as a high-performance replacement policy, in high associativity caches, the LRU policy causes an increase of LRU state bits for recording order of references,

and complicates the LRU state update logic [1]. In addition, the LRU policy does not have a tolerance for scan access pattern. In order to solve these problems, the Adaptive Demotion Policy (ADP) is proposed [2]. This policy can be realized by the small hardware overheads for high-associativity caches, and the priority update logic can be implemented with simple hardware. In addition, The ADP can suit scan/thrashing access patterns by changing its insertion, promotion and selection policies. The ADP can select these parameters suitable for the running application from two alternatives by the set-dueling [3]. However, the set-dueling uses several sets of the cache for each alternative. Therefore, extreme parameters are hard to adapt. Even if the cache uses the extra hardware to analyze the application behavior, the power consumption of the processor will be increased. Therefore, the simple scheme which can detect the behavior of the running application is needed.

This paper proposes a mechanism which dynamically detects the behavior of the running application and selects the suitable parameters for that. In order to achieve the dynamic selection of the suitable parameters, this paper focuses on the global fluctuations of the priority values. At first, the cache is partitioned into several partitions. At every cache access, the total of priority values in each partition is calculated. At every set interval, the fluctuations of the total of priority values in all partitions are checked, and the information is used to detect the behavior of the application. For example, if the totals of priority values in all partitions are decreased, scan/thrashing access will be occurred. Therefore, the parameters are set to that of the suitable for the scan/thrashing access. This paper adapts this mechanism to the ADP, and the adapted replacement policy is called ADP-G. In this paper, the parameters of the ADP-G such as the number of partitions are examined, and the performance of the ADP-G is evaluated by the processor simulator.

This paper is organized as follows. Section 2 outlines the basis of cache replacement policies and related work. Section 3 describes the behavior of the ADP-G. Section 4 evaluates the performance of ADP-G. Section 5 concludes this paper.

2 Related work

In the cache replacement policies which are based on the priority, each block in a set has a priority value, and the block which has the smallest priority value in the set is evicted at a cache miss. Under such a situation, four priority policies decide the behavior of the cache replacement policy: Insertion, Promotion, Demotion and Selection policies. In this paper, these policies are called IPDS policies. This section describes characteristics of IPDS policies and related work.

2.1 Characteristics of IPDS policies

2.1.1 Insertion Policy

The insertion policy decides the initial priority of the incoming block. If the initial priority of the incoming block is lower than the priorities of the blocks in the set, the incoming block will be evicted before re-reference when the incoming block does not hit before the next cache miss. On the other hand, if the block which is not re-referenced has a high priority, the block will spoil the performance of the cache. Therefore, a large initial priority value suits the temporal locality. In contrast, a small initial priority value is appropriate for a scan access pattern.

2.1.2 Promotion Policy

The promotion policy decides how to promote the priority of the hit block. Mainly the policy is selected from two alternatives: Hit Priority (HP) and Frequently Priority (FP). The HP policy promotes the priority to the maximum as like the LRU policy. It is suitable for the temporal locality. The FP policy increases the priority by one as like the Least Frequently Used (LFU) policy. Because the block which is not re-referenced tends to have a lower priority, the FP policy is suitable for a scan access pattern.

2.1.3 Demotion Policy

The demotion policy decides how to decrease the priority values of the blocks in the accessed set at a cache access. When the LRU policy is implemented with priority base, the priority values of the blocks which have a larger priority value than the priority value of the evicted block is decreased by one. This demotion is also executed at a cache hit, the complexity of the update logic will affect the processor cycle time. As the LFU policy does not demote the priority, it will cause never re-referenced blocks. In the RRIP policy [4], the subtraction value is set to the priority value of the evicted block. When the hit rate of the cache is high, a cache miss minimizes the priorities of all the existing blocks in the set. This causes a harmful effect to the performance by evicting useful data. The ADP is proposed to solve this problem. In ADP, the subtraction value is dynamically calculated with based on the average of the priority values of the accessed set.

2.1.4 Selection Policy

In priority-based replacement policies, the block which has the smallest priority value is selected as a victim. As like the LRU policy, if only one block has the smallest priority, the block is selected as a victim. However, several replacement policies cause several blocks which has the smallest priority value. Under such a case, one block is selected from these blocks by a selection policy. Two selection policies are considered: The Left-side Selection (LS) policy and the Random Selection (RS) policy. The LS policy selects the block which has the lowest way number as a victim. The LS policy is adopted in several replacement policies like as the RRIP policy. The RS policy selects the block as a victim at random. The Random replacement policy is considered that the all the blocks in the accessed set have same priority values, and the RS policy is adopted.

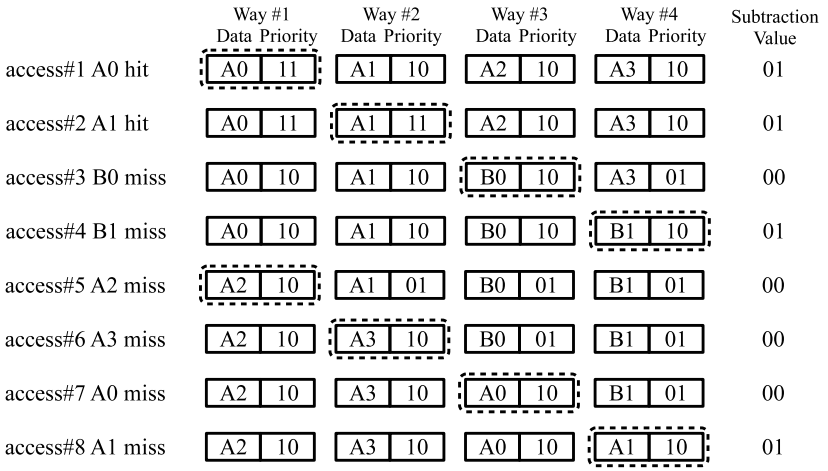
2.2 Adaptive Demotion Policy

ADP prepares priority values for each cache block. Regardless of the associativity, the 2-bit value is enough for storing the priority [4]. Figure 1 shows the behavior of ADP. Since caches should quickly send the data to computational resources when cache hits occur, the control logic at cache hits for updating priority value should be simple. The same as the RRIP policy, the priority value of a block is promoted by the promotion policy when cache hits occur in that block. The behavior at a cache miss is as follows. First, the block that has the smallest priority in the accessed set is selected as the victim. If there are two or more blocks which have the smallest value in the accessed set, one block is selected by the selection policy. Next, the priority values of all the blocks in the accessed set are decreased by the subtract value, which is generated by the demotion policy. Finally, a new block coming from the lower-level memory hierarchy is stored, and the priority value of the block is set to the initial priority value.

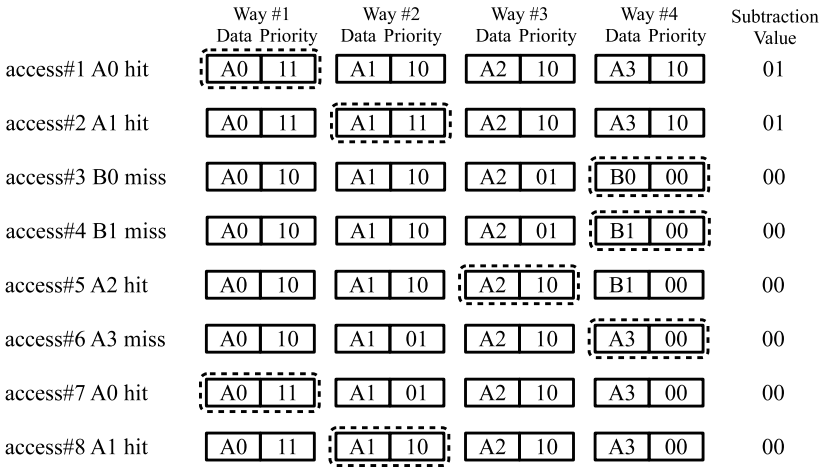
The ADP can suit for any characteristics of applications by selecting appropriate policies. Figure 1(a) shows the behavior of ADP when the IPDS policies are I2(10), HP, HOA and LS. I2(10) means the initial priority value is 2 in decimal and 10 in binary. HOA means the half of the average, that takes the half of the average of the priority values in the accessed set, then sets it as the subtraction value at a cache miss. On the other hand, Figure 1(b) shows the case of the IPDS policies are I0(00), FP, HOA and RS. In order to dynamically change these policies, the D-ADP is proposed [2]. The D-ADP uses the set-dueling method [3] to select the appropriate policies from two alternatives. In the D-ADP, aggressive policies are hardly to adopt as these two alternatives because the extreme policies sometime cause significantly increase of the cache misses.

3 A Replacement Policy with Considering Global Fluctuations of Priority Values

In order to improve the performance of the replacement policy, it is considered effective to change the behavior according to the running application. However, preparing the hardware that dynamically detects the characteristics of the running application will increase the energy consumption of the



(a) Behavior of the ADP (Insertion:I2(10), Promotion: HP, Demotion:HOA, Selection:LS)



(b) Behavior of the ADP (Insertion:I0(00), Promotion: FP, Demotion:HOA, Selection:RS)

Figure 1: Behavior of the ADP.

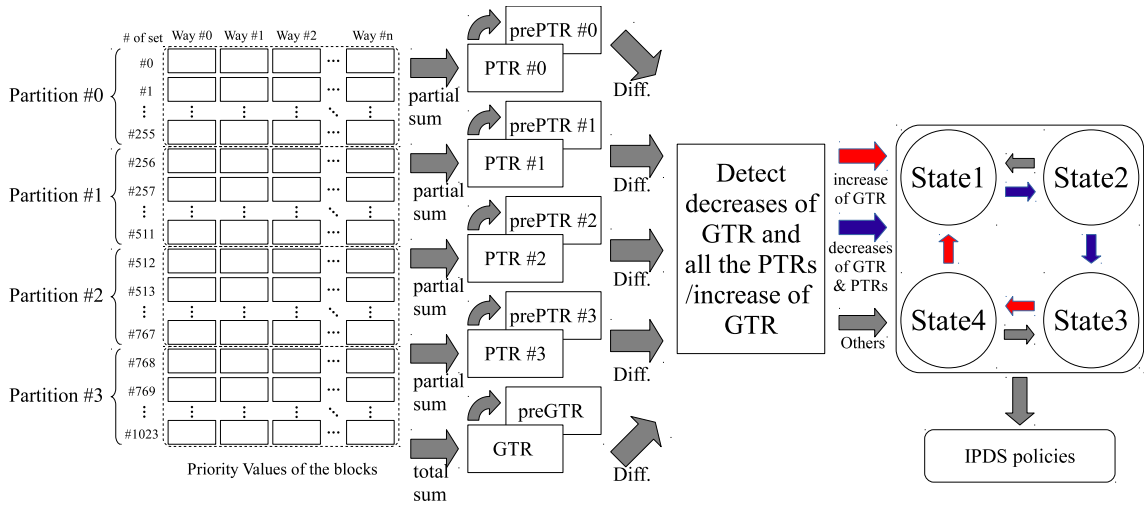


Figure 2: Outline of the ADP-G

processor. This paper proposes a method to dynamically decide the IPDS policies based on the global fluctuations of the priority values in the cache. Figure 2 shows the outline of the proposed system. The global total register (GTR) is prepared to store the total of the priority values of all blocks in the cache. Also several partial total registers (PTRs) (in Fig.2, the number of PTRs is four) are prepared to store the total of the priority values of partitions which consists of contiguous sets in the cache. The values of all the registers are updated at every cache access. In addition, the previous value registers (preGTR and prePTRs) are prepared, and the values of GTR and PTRs are saved to preGTR and prePTRs at regular intervals, respectively. At regular intervals, the behavior of the running application is detected by comparing the values of recent GTR and PTRs to preGTR and prePTRs.

The detail of the method is as follows. There are four states to select the IPDS policies. State 1 and 2 output the policies suitable for the temporal locality, and State 3 and 4 output the policies suitable for scan/thrashing access. The behavior at State 1 is as follows. At regular intervals, GTR and preGTR are compared. If the decrease of GTR is larger than the threshold value, PTRs and prePTRs are compared. If all the PTRs are decreased, the state is change to State 2. The behavior of State 2 is same as that of State 1 except the next state is State 3. The behavior of State 3 is as follows. At regular intervals, GTR and preGTR are compared. If the increase is larger than the threshold value, the state is change to State 4. At State 4, the next state is State 1 and the other behavior is same as that of State 3.

Here, the hardware resources to implement the ADP-G and the power consumption of that are considered. At first, the numbers of bits of PTR and GTR are discussed. These numbers are decided by the base 2 logarithm of the value which is the number of blocks times the maximum priority value of a block. For example, when the cache size is 2 Mbytes and the block size is 64 bytes, the number of blocks in the cache is 32768. As the maximum priority value of a block is 3, the maximum number of GTR is 98304. If the number of partitions is four, one partition consists of 8192 blocks and the maximum number of PTR is 24576. Therefore, the number of bits of GTR is 17 bits, and these of PTRs are 15x4 bits. Also, prePTR and preGTR need as same the number of bits of PTR and GTR, totally 154 bits is required to implement the ADP-G. This number of bits is 0.23% of the number of bits to store the priority values. Therefore, these registers do not make a large influence on the power consumption of the whole cache system.

Next, the complexity of the hardware to update PTR and GTR is discussed. In order to update PTR and GTR, two saturation adders are required. The numbers of bits of these adders are the same as the numbers of bits of PTR and GTR. As the value to update the PTR and GTR can be generated by the controller of ADP, other extra hardware does not be required. Therefore, it is

Table 1: Cache Parameters

cache	size	assoc.	latency
L1I	32KB	8	1
L1D	32KB	8	1
L2	256KB	8	8
L3	2MB/4MB	16	20
Main Memory	4GB	-	150

considered that the ADP-G can be implemented with small hardware resources.

4 Evaluation of the ADP-G

The simulation experiments are performed to show the performance of the ADP-G and the effects of the parameters of the ADP-G and the IPDS policies. For these experiments, the simulator including the ADP-G is developed based on the gem5 simulator system with Alpha 21264 instruction [5]. The benchmarks included in the SPEC CPU2006 [6] are examined as workloads for the experiments. For each benchmark, a representative phase of one billion instructions is extracted from the full execution by SimPoints [7], and the extracted phase is used for the evaluations. In each simulation, the first 50M instructions are used for warming up, and the following 950M instructions are for obtaining the simulation results. Table 1 shows cache parameters used in the simulation. In all the experiments, the L3 cache adopts the experimental replacement policy. The other caches adopt the LRU policy. In all the caches, the block size is 64 bytes.

In order to examine the effects of the parameters of the ADP-G, a preliminary experiment is conducted. The examined parameters are the two threshold values (*TH* and *TL*), and the number of partitions. *TH* is the threshold value for State 1 and State 2. *TL* is that for State 3 and State 4. These threshold values are decided by the maximum value of GTR. The examined *THs* are the quotient of the maximum value of GTR divided by 64 and 32 (shown as 1/64 and 1/32 in the figures), and the examined *TLs* are zero and the quotient of the maximum value of GTR divided by 64 (shown as 1/64 in the figures). The examined numbers of partitions are four and eight. In this experiment, the IPDS policies which are outputted from the State 1 and State 2 are I2, HP, HOA, and LS. State 3 and State 4 output I0, FP, HOA, and RS. The interval is set to the half of the number of sets. The cache size of the L3 is set to 2MB.

Figures 3 and 4 show the Misses Per Kilo-Instructions (MPKI) rate of the ADP-G with various parameters compared to the LRU policy. Figure 3 shows the results when the number of partitions is four, and Fig. 4 shows the results when the number of partitions is eight. Regardless of the number of partitions, the performance of ADP-G shows the same tendency. To reduce the number of PTRs, four is a reasonable value as the number of partitions. In terms of the *TL*, the performance of the zero is lower than that of the 1/64 with several benchmarks. In terms of the *TH*, the 1/64 achieves a high-performance in several benchmarks, but a few benchmarks show a significantly drop of the performance. As the low threshold values causes a high sensitivity of ADP-G, these decrease the performance with several benchmarks.

Based on the results, the parameters of the ADP-G which used as follow experimentation is decided. The number of partitions is set to four. *TH* is set to the quotient of the maximum value of GTR divided by 32. *TL* is set to the quotient of the maximum value of GTR divided by 64.

In the evaluations, the performances of the LRU policy and the Static-RRIP (SRRIP) policy, and the ADP are also evaluated. The SRRIP policy adopts the HP policy as the promotion policy, and the initial priority is set to I1. In the experiments, the configurations of ADP are illustrated like as I2/HP/HOA/LS that mean the initial priority value of I2, the HP, the HOA, and the LS policies are adopted. The D-ADP selects the IPDS policies from two alternatives which are same as the ADP-G by the set-dueling.

Figures 5 and 6 show the MPKI reduction rate of all the benchmarks compared to the LRU policy

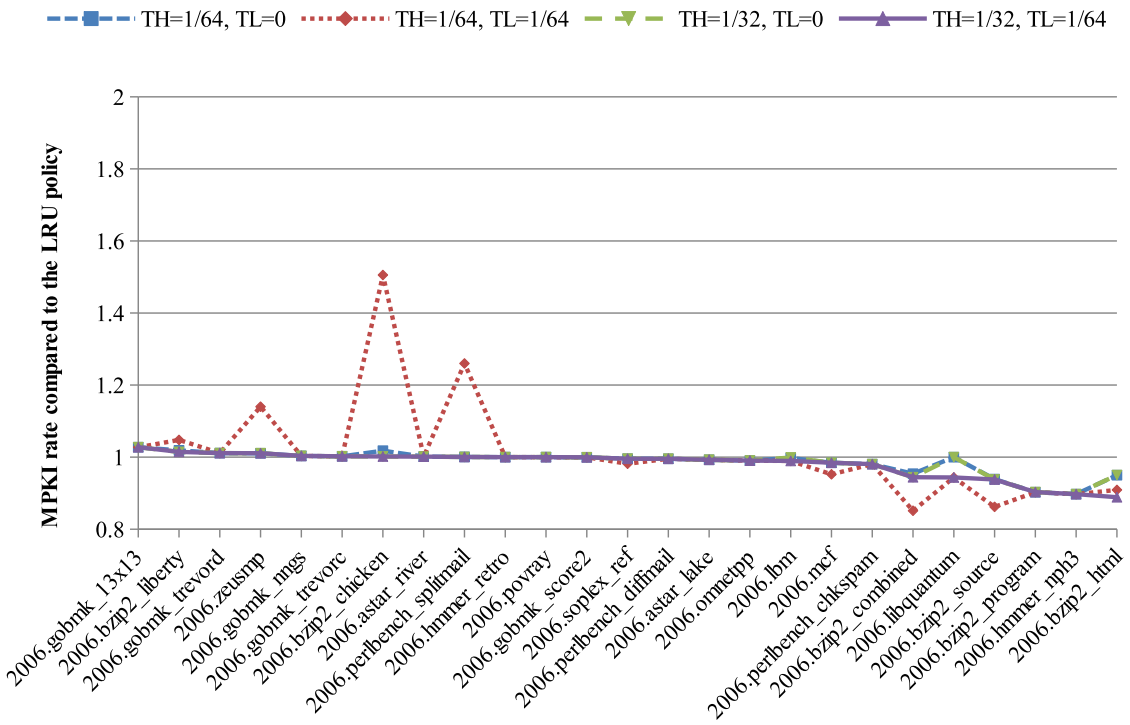


Figure 3: MPKI rate of the ADP-G compared to the LRU policy (four-partitioned).

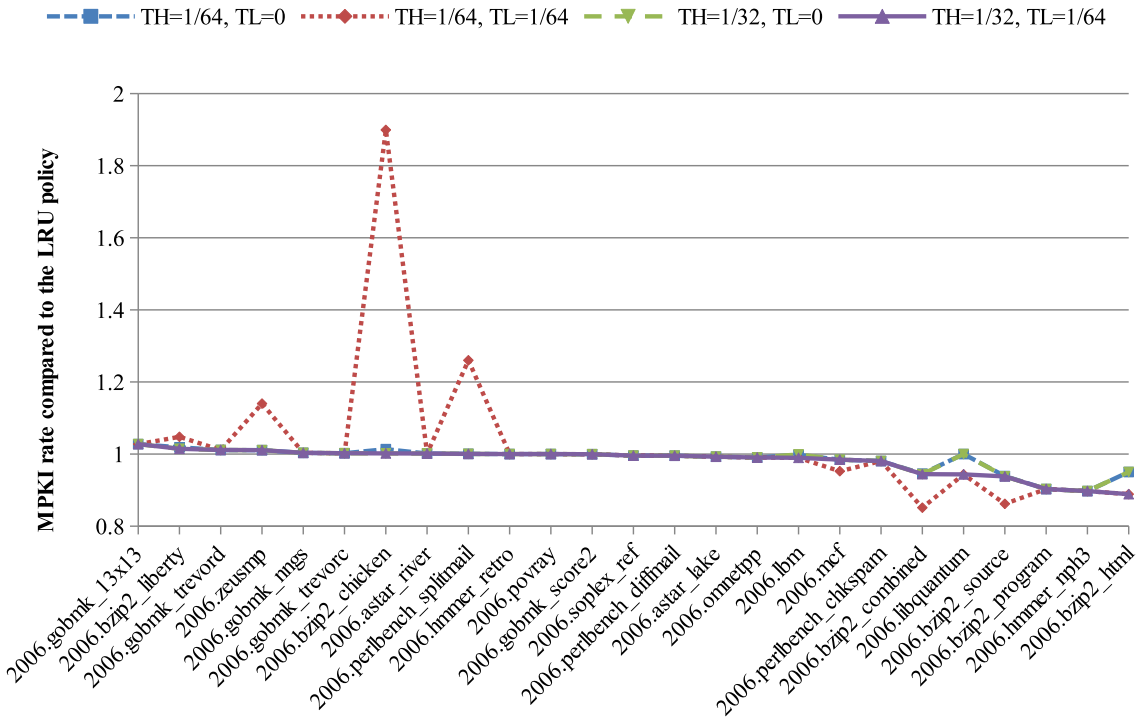


Figure 4: MPKI rate of the ADP-G compared to the LRU policy (eight-partitioned).

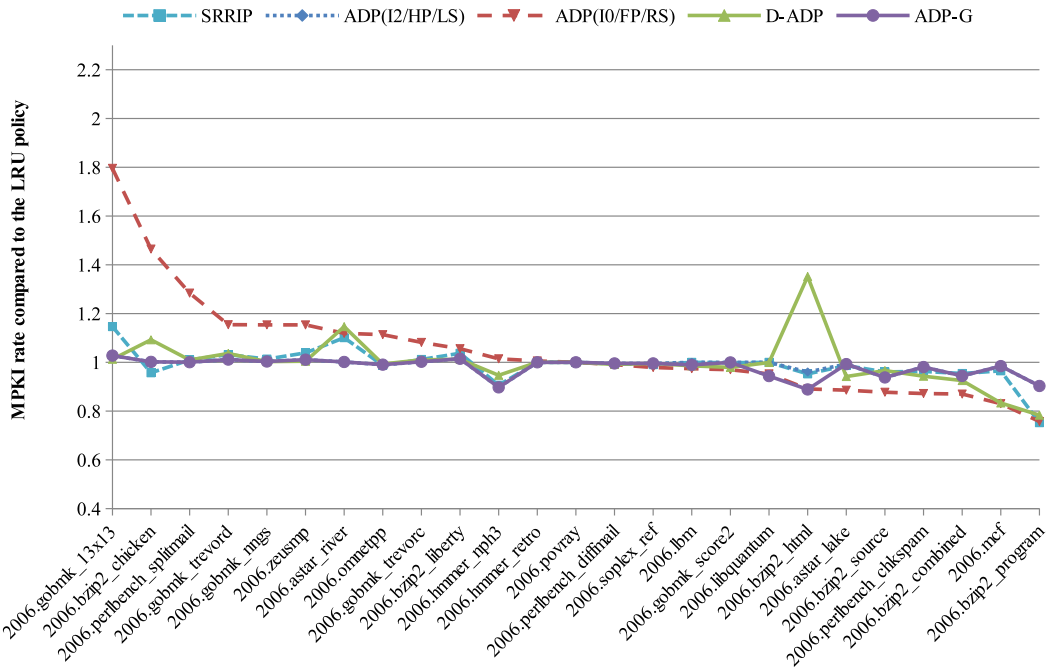


Figure 5: MPKI rate compared to the LRU policy (2MB).

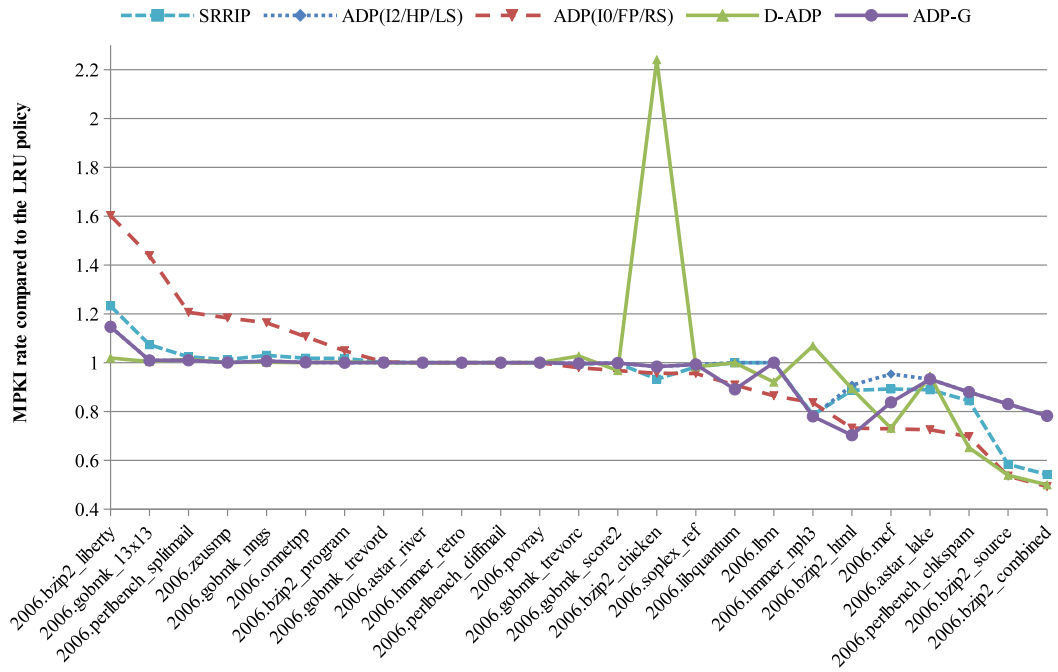


Figure 6: MPKI rate compared to the LRU policy (4MB).

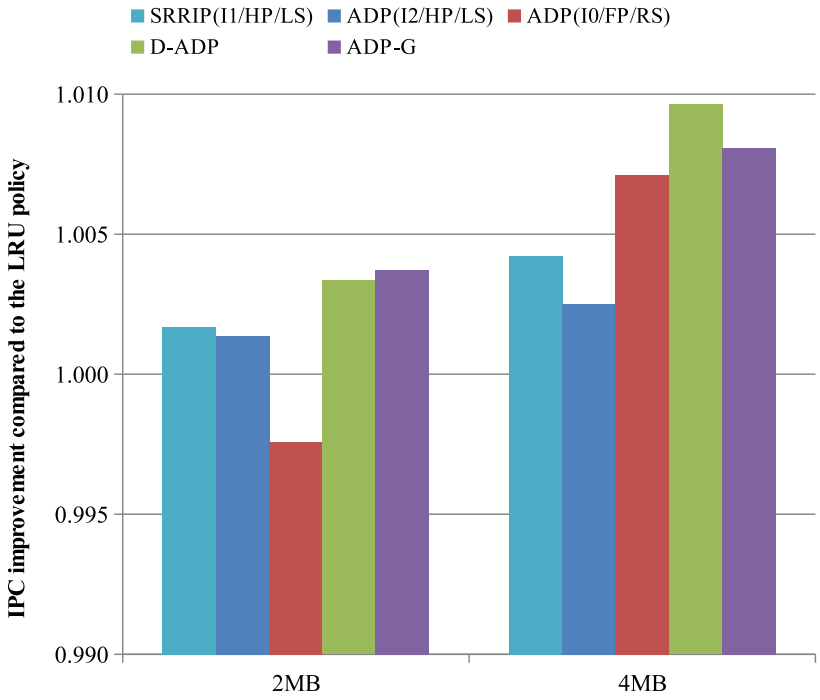


Figure 7: IPC improvement compared to the LRU policy.

when the cache sizes are 2MB and 4MB, respectively. As shown in Fig. 5 and 6, the appropriate policies of the ADP for each application are clearly separated. In addition, the D-ADP causes significantly increase of the cache misses in several applications with these two alternative policies. To improve the performance of the D-ADP, the policies without these significantly increase of the cache misses should be selected.

As compared to the D-ADP, the ADP-G does not cause the cache misses compared to the ADP with two alternatives. In geometric mean of all the benchmarks, the ADP-G achieves a 2.0% MPKI reduction compared to the LRU policy when the cache size is 2MB. When the cache size is 4MB, the MPKI reduction rate of the ADP-G is 5.4% in the geometric mean, and it is same as that of the D-ADP.

Figure 7 shows the geometric mean of the IPC improvement compared to the LRU policy. When the cache size is 2MB, the ADP-G achieves a 0.37% IPC improvement compared to the LRU policy. The improvement rate is higher than the SRRIP and the D-ADP. Although the ADP-G achieves a 0.81% IPC improvement compared to the LRU policy when the cache size is 4MB, the improvement rate is lower than the D-ADP. As shown in Fig. 7, the IPC improvement of the ADP(I0/FP/HOA/RS) is larger than that of the ADP(I2/HP/HOA/LS) when the cache size is 4MB. When the cache size is large, the scan/thrashing suitable policies can achieve a higher performance than the temporal locality suitable policies. In future works, the method which can select the appropriate IPDS policies with considering the size of the cache should be discussed.

5 Conclusions

This paper proposes a high-performance cache replacement policy called the ADP-G. The ADP-G focuses on the global fluctuations of the priority values to detect the behavior of the running application and select the suitable parameters for that. Evaluation results show the ADP-G achieves the MPKI reduction compared to the LRU policy, the RRIP policy and the ADP. In geometric mean of all the benchmarks, the ADP-G achieves a 2.0% MPKI reduction and a 0.37% IPC improvement

compared to the LRU policy when the cache size is 2MB.

References

[1] T. S. B. Sudarshan et al. Highly efficient lru implementations for high associativity cache memory. In *Proceedings of 12th IEEE International Conference on Advanced Computing and Communications*, 2004.

[2] J. Tada et al. An adaptive demotion policy for high-associativity caches. In *Proceedings of international symposium on Highly Efficient Accelerators and Reconfigurable Technologies (HEART2017)*, 2017.

[3] M. K. Qureshi et al. Adaptive insertion policies for high performance caching. In *Proceedings of International Symposium on Computer Architecture (ISCA 2007)*, 2007.

[4] A. Jaleel et al. High performance cache replacement using re-reference interval prediction (rrip). In *Proceedings of International Symposium on Computer Architecture (ISCA 2010)*, pages 60–71, 2010.

[5] N. Binkert et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[6] J. L. Henning. Spec cpu2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 34(4):1–17, 2006.

[7] G. Harmerly et al. Simpoint 3.0: Faster and more flexible program phase analysis. *Journal of Instruction Level Parallelism*, 7(4):1–28, 2005.