HAL
open science

Cache Replacement Policy Using Map-based Adaptive Insertion

Yasuo Ishii, Mary Inaba, Kei Hiraki

To cite this version:

Yasuo Ishii, Mary Inaba, Kei Hiraki. Cache Replacement Policy Using Map-based Adaptive Insertion. JWAC 2010 - 1st JILP Worshop on Computer Architecture Competitions: cache replacement Championship, Jun 2010, Saint Malo, France. inria-00492985

HAL Id: inria-00492985

https://inria.hal.science/inria-00492985v1

Submitted on 17 Jun 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Cache Replacement Policy Using Map-based Adaptive Insertion

Yasuo Ishii
The University of Tokyo, NEC
yishii@is.s.u-tokyo.ac.jp

Mary Inaba
The University of Tokyo
mary@is.s.u-tokyo.ac.jp

Kei Hiraki
The University of Tokyo
hiraki@is.s.u-tokyo.ac.jp

ABSTRACT

In this paper, we propose a map-based adaptive insertion policy (MAIP) for a novel cache replacement. The MAIP estimates the data reuse possibility on the basis of data reuse history. To track data reuse history, the MAIP employs a bitmap data structure, which we call memory access map. The memory access map holds all memory accessed locations in a fixed sized memory area to detect the data reuse. It can cover a large memory area that is compared to the size of a large L3 cache memory. The MAIP can use a large amount of data reuse history from the memory access map. On the basis of reuse history from both the original cache tag and the memory access map, the MAIP estimates reuse possibility of the incoming line in terms of two metrics: (1) spatial locality and (2) temporal locality. The combination of these metrics improves the accuracy of the reuse possibility estimation because each locality supports different memory access patterns. When the reuse possibility is insufficient, the incoming cache line is not inserted into the MRU position so as to evict it before other cache lines.

We evaluate the MAIP by performing a simulation study. The simulation result shows that the MAIP reduces the cache miss count by 8.3% compared to the LRU policy while the Dynamic Insertion Policy (DIP) reduces the cache miss count by 0.1%. The MAIP improves performance by 2.1% in a single-core configuration and by 9.1% in a multi-core configuration compared with the traditional LRU policy.

1. INTRODUCTION

The cache replacement policy plays an important role in improving system performance by reducing cache miss counts. The optimal replacement policy (OPT) [1] is known as the best replacement policy, but it is not practical because OPT uses future information for deciding the victim set. The least recently used (LRU) policy has been used as a de-facto standard for decades. The traditional LRU policy shows good performance when the working set size is not larger than the available cache memory size. However, the LRU policy suffers from a cache thrashing problem in the case of a large working set because a useless cache line whose reuse count is zero is inserted into the MRU position. Once an incoming cache line is inserted into the MRU position, it is not evicted until it moves to the LRU position. It leads to undesirable cache replacements involving the eviction of useful cache lines and degrades system performance.

Many replacement policies have been proposed to improve the traditional LRU policy. The Dynamic Insertion Policy (DIP) [2] [3] is an insertion policy that mitigates the impact of the cache thrashing problem. The DIP inserts most incoming cache lines into the LRU position for the application that suffers from cache thrashing problem. The position where the incoming line is inserted is decided by a random mechanism. It improves performance with a small amount of additional hardware cost because it does not need memory access history in detail. However, its insertion policy is too simple for a processor that employs both a L2 cache and a L3 cache because the random insertion algorithm is not effective without sufficient memory accesses to the L3 cache. Generally, the L3 cache is not frequently accessed because most memory accesses result in a cache hit to the L2 cache. The dead block prediction (DBP) [4] and less reused filter (LRF) [5] estimate the reuse possibility of cache lines. When a cache line is estimated to be useless, the cache line is evicted before other cache lines. Such estimation techniques can work well even when an amount of the memory accesses to the L3 cache is insufficient because they controls the replacement policy on the basis of the data reuse history. However, these replacement algorithms require a large storage budget to track the cache lines evicted early because the wrong evictions cannot be detected without the history of the evicted cache lines. Moreover, the total budget size grows with the cache size. With a multi-megabyte cache memory, these policies are not practical due to its hardware cost. To overcome the drawbacks of the insertion algorithm of DIP and the hardware cost of DBP and LRF, the reuse possibility estimation mechanism that is based on a cost-effective memory tracking data structure is required.

In this paper, we propose a map-based adaptive insertion policy (MAIP). The MAIP is a new insertion policy for cache replacement. In this policy, the insertion mechanism in the traditional LRU replacement is modified by a reuse possibility estimation to evict useless cache lines earlier than LRU policy. The MAIP estimates the usefulness of the incoming line when a cache miss occurs. When the incoming line is estimated to be useful, it is inserted into the MRU position.

The MAIP can estimate the reuse possibility with reasonable hardware cost. It uses a memory access map to track a large amount of memory accesses. A memory access map holds memory accesses in a memory area with fixed size (we call it "zone"), like macroblocks [6]. A bitmap data structure holds previously accessed locations in the zone. The bitmap data structure helps the MAIP track a large amount of memory accesses because only a few bits are required for tracking one accessed location. In this study, the memory access map could cover a memory area of up to 3.0 MB with a 7.7 KB budget size. To detect a previously accessed location, the MAIP uses the memory access map as an additional cache tag and estimates the reuse possibility on the basis of both the original cache tag and memory access map.

When a cache miss occurs, the MAIP estimates the reuse possibility of the incoming line in two terms of different metrics: (1) spatial locality and (2) temporal locality. The spatial locality is derived from the reuse possibility of the neighbors of the incoming cache line. The temporal locality is derived from the reuse possibility associated with the memory access instruction that initiates the cache miss. The MAIP can estimate the reuse possibility of an incoming

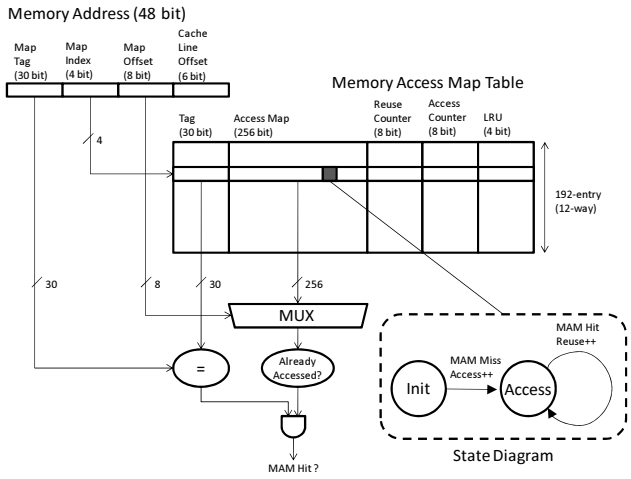

Figure 1: Memory Access Map

cache line more accurately than existing replacement policies by estimating the possibility from both localities.

2. DESIGN AND IMPLEMENTATION

The MAIP estimates the reuse possibility from both spatial locality and temporal locality. The reuse possibility estimation based on the spatial locality is useful for stream applications because the reuse probability of the incoming line is almost the same as that of the neighboring lines. The temporal locality is also useful for reuse possibility estimation because each memory access instruction has different properties. To exploit the spatial locality, the MAIP uses a memory access map. The MAIP counts the number of times the data has been reused in a zone for the reuse possibility estimation. When data in the zone is rarely reused, the incoming cache line is estimated to be useless. On the other hand, the MAIP employs a bypass filter table to exploit the temporal locality. This table tracks data reuse for each memory access instruction. When instructions rarely reuse data, the incoming lines that are initiated by these instructions are considered to be useless. The MAIP combines the two reuse probabilities estimated from the two different metrics.

The MAIP consists of three parts: (1) a memory access map, (2) bypass filter, and (3) reuse possibility estimation logic. We also use the traditional LRU replacement policy for applications with small working set. To switch between the MAIP and the LRU policy, the enhanced set dueling mechanism is adopted. In the rest of this section, we explain each component in detail.

2.1 Data Structure

2.1.1 Memory Access Map

The memory access map is a bitmap data structure for tracking previous memory accesses. Originally, this data structure was proposed for hardware data prefetching [7]. The memory access map is mapped to the fixed sized consecutive memory address space. We use a memory access map table for keeping track of the memory accesses. It has a set-associative-cache-like data structure. Figure 1 shows an overview of the memory access map.

Each memory access map employs (a) an address tag, (b) access states, (c) a reuse counter, (d) an access counter, and (e) an LRU stack position. The address tag contains the corresponding zone address. The access states hold

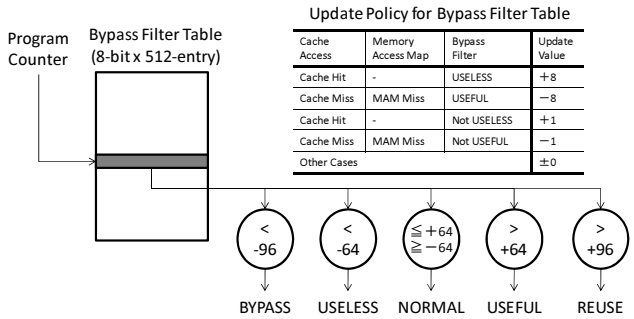

Figure 2: Bypass Filter Table

information on previously accessed locations of the zone. Each state is attached to the cache line of the zone and can represent two states (INIT and ACCESS). The INIT state indicates that the corresponding location has not yet been accessed. The ACCESS state indicates that the corresponding location has already been accessed. The reuse counter and access counter are saturating counters that track the reuse frequency of the corresponding zone.

During memory access, the memory access map table is accessed in parallel with the cache. If no entry is found in the table, a new entry is allocated and initialized. If an entry is found, the corresponding map is read and updated. When a new entry is allocated, the LRU entry in the memory access map table is replaced. During the initialization phase, all access states are initialized to the INIT state and the counters are initialized to zero. When there is a hit entry, the corresponding memory access map is read and updated according to the following steps. First, the access state that is referred to by the actual memory request is read from the table. When the state is the ACCESS state, it indicates that the data of the corresponding location is being reused. A memory access to the ACCESS state is called the "MAM hit." In the case of "MAM hit," the reuse counter is incremented. Otherwise, the access counter is incremented and the access state moves to the ACCESS state. Finally, the corresponding memory access map is updated to the MRU position.

The memory access map has advantage to track memory accesses. As shown in figure 1, each entry with a 306 (= 30 + 256 + 8 + 8 + 4) bit budget can cover 256 cache lines (= 16 KB). Therefore, the 192 entries (= 7.7 KB) cover up to 3.0 MB of memory address space. The cost-efficiency of the memory access map is much higher than conventional shadow tags because each entry of a shadow tag requires tag information.

2.1.2 Bypass Filter

A bypass filter is a table for tracking the reuse possibility of each memory access instruction. An overview of the bypass filter table is shown in figure 2. The bypass filter table is indexed by lower bits of the program counter of the instruction that initiates the corresponding memory accesses. Each entry of the bypass filter table is a saturating counter. The counter value represents the reuse possibility of the corresponding instruction. When the counter value is greater than a threshold, the MAIP considers the incoming cache line to be a useful cache line. Otherwise, the MAIP regards the memory access as a useless cache line.

In this study, the reuse possibility of the bypass filter table is classified into five groups according to the counter value. These five groups are BYPASS, USELESS, NORMAL,

USEFUL, and REUSE. BYPASS is the group with the lowest reuse possibility, and REUSE has the highest reuse possibility.

The MAIP updates the bypass filter table according to the lookup results of the cache memory and the memory access map. When the lookups result in neither a cache hit nor "MAM hit," the corresponding entry of the bypass filter is decremented. In the case where the corresponding data is USEFUL or REUSE, the counter is decreased by 8. Otherwise, the counter is decreased by 1. When the cache memory lookup results in a cache hit, the corresponding entry of the bypass filter is incremented. In the case where the corresponding data is USELESS or BYPASS, the counter is incremented by 8. Otherwise, the counter is incremented by 1.

2.2 Reuse Possibility Estimation

During cache memory access, lookup of the memory access map table and lookup of the bypass filter table are performed in parallel with the lookup of the cache memory. These lookups provide information on previous memory accesses. When a lookup of the cache memory results in a cache miss, the MAIP estimates the reuse possibility of the incoming line. When the MAIP estimates the reuse possibility to be insufficient, the incoming line is inserted into the LRU position or is bypassed to insert into the cache memory.

The MAIP estimates the reuse possibility on the basis of both spatial locality and temporal locality. The reuse possibility based on the spatial locality is estimated from information in the memory access map. The memory access map holds the reuse count and access count of the corresponding zone. When the reuse count is larger than the access count, the zone is judged to have high reuse possibility because many neighbors of the incoming line are accessed repeatedly. On the other hand, when the reuse count is much smaller than the access count, the zone is judged to have insufficient reuse possibility. In such a case, the incoming line bypasses the cache so that the useful data already stored in the cache memory is not replaced.

The MAIP also estimates the reuse possibility on the basis of the temporal locality, from the information in the bypass filter table. When the incoming line corresponds to a BYPASS instruction, the incoming line bypasses the cache. On the other hand, the line corresponding to a REUSE instruction is inserted into the MRU side of the cache memory.

In this study, the MAIP controls the insertion position of an incoming line according to the tracking memory access information. The MAIP takes into account the cache hit/miss (H_c), MAM hit/miss (H_m), reuse count (C_r), access count (C_a), and bypass filter state (S_b) to decide the insertion position.

Cache bypass occurs when (a) $C_r < (C_a/16)$ or (b) $S_b = BYPASS$. In such cases, the incoming cache line is not inserted into the cache memory because the MAIP cannot detect sufficient reuse possibility. Otherwise, the incoming line is inserted into the cache memory. The MAIP decides the insertion position, as shown in figure 3. Basically, a cache line with high reuse possibility tends to be inserted into the MRU side.

2.3 Adaptive Dedicated Set Reduction (ADSR)

Although the MAIP shows good performance in many cases, the traditional LRU replacement policy has an advantage when the workload size is small for the available cache size. We use set dueling [2] to choose the better of the two policies. Set dueling employs saturating counters to

```
function get_insert_position()
    int ins_pos = 15
    if(H_m)              {ins_pos = ins_pos/2}
    if(C_r > C_a)        {ins_pos = ins_pos/2}
    if(S_b = REUSE   ) {ins_pos = 0          }
    if(S_b = USEFUL ) {ins_pos = ins_pos/2}
    if(S_b = USELESS){ins_pos = 15         }
    return ins_pos
endfunction
```

Figure 3: Function for Determining Insert Position (0 is MRU, 15 is LRU)

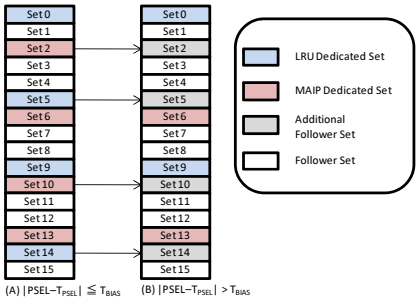

Figure 4: Adaptive Dedicated Set Reduction

keep track of which of the two policies incurs fewer cache misses. The existing set dueling monitor employs a few policy selection (PSEL) counters.

A drawback of set dueling is that the dedicated set must use one policy even if the policy selection counter is strongly biased toward the other policy. To overcome the drawback, we propose adaptive dedicated set reduction (ADSR). ADSR reduces the number of the dedicated set when the PSEL counter shows a strong bias toward one policy (figure 4). It contributes to a reduction in cache misses due to the dedicated policy.

3. EVALUATION

We evaluate the MAIP within the CRC framework. We use SPEC CPU2006 as the benchmark for our evaluation. Each benchmark is compiled by using GCC 4.1.2 with the "-O3 -fomit-frame-pointer -funroll-all-loops" option. The simulation skips the first 40G instructions and evaluates the following 100M instructions. We use ref inputs for the simulation. For the multi-core configuration, 15 workloads adopted from a previous study [3] are used. As competitive replacement policies, we also evaluate OPT [1] and DIP [2] in a single-core environment and TADIP-F [3] in a multi-core environment.

3.1 Hardware Cost

MAIP employs a 12-way set-associative memory access map and a 512-entry bypass filter table. These components are allocated to each core. Table 1 shows the budget counts

Table 1: Budget count of per-core resource

Resource Name		Budget
Memory Access Map	192-entry, 12-way	58752-bit
Bypass Filter	512-entry	4096-bit
PSEL Counter	1 counter	10-bit
Total Cost		62858-bit

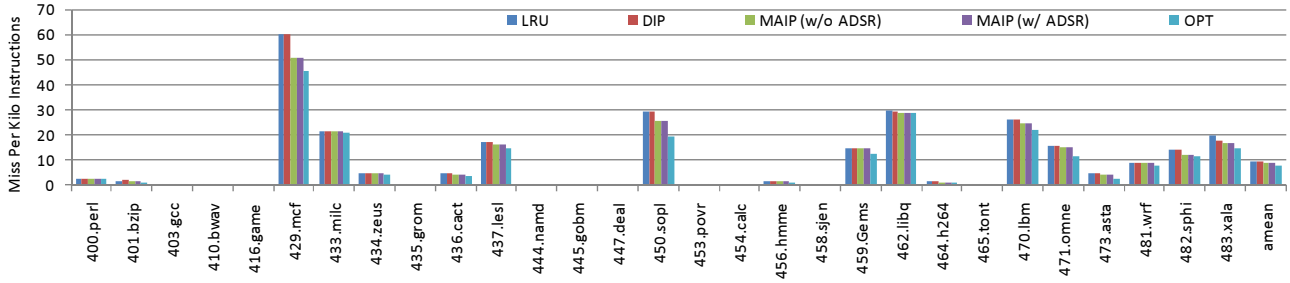

Figure 5: Cache Miss Count per 1000 Instructions

Table 2: Total budget count

	1 core	4 cores
LRU	16K-line × 4-bit	64K-line × 4-bit
Per Core	1 core × 62858-bit	4 cores × 62858-bit
Total Cost	128394-bit	513576-bit

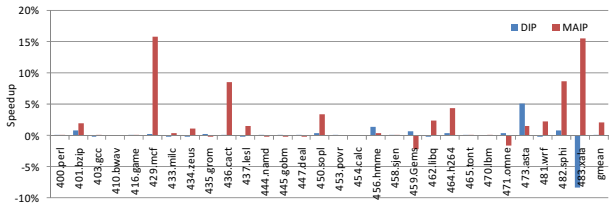

Figure 6: Speedup in Single-core

for the resource allocated to one core. Each memory access map entry requires 306 bits as shown in figure 1. Table 2 shows the total budget count for each configuration. A policy selection counter is attached to each core for set dueling.

3.2 Result

Figure 5 shows the cache miss count of the single-core configuration. The MAIP with ADSR reduces the cache miss by 8.3% while OPT reduces it by 18.2%. This result shows that the use of ADSR reduces the cache miss by 1.0%. Figure 6 shows the speedup over the LRU policy. The MAIP improves the performance by 2.1% compared to the LRU policy. Figure 7 shows the result for the multi-core configuration. It improves the performance by 9.1% compared to the LRU policy.

4. CONCLUSION

In this paper, we propose a map-based adaptive insertion policy (MAIP) that improves cache efficiency by reducing the amount of useless data in the cache memory. The MAIP estimates the reuse possibility of an incoming cache line. Unless the incoming line is estimated to be useful, the incoming line is not inserted into the MRU position. The MAIP uses a memory access map for tracking the memory access history so as to obtain accurate estimation. The memory access map can track a large amount of memory accesses. Consequently, the MAIP is suitable for a system with a large L3 cache because of its cost-effective data structure. On the basis of the tracking history, the MAIP estimates the reuse possibility from both spatial locality and temporal locality. Depending on the reuse possibility, the MAIP decides the insertion position of the incoming line.

We evaluate the MAIP within the CRC framework. According to the simulation results, the MAIP reduces cache miss count by 8.3% compared to the traditional LRU policy. The MAIP improves performance by 2.1% for the single-core configuration and by 9.1% for the multi-core configuration. The evaluation results show the advantage of the MAIP over existing replacement policies.

Although we show that the use of the MAIP results in performance improvement, there is still room for improving the performance of the memory subsystem like data prefetching. Fortunately, the memory access map is originally proposed as the data structure for data prefetching. Evaluating an interaction of a novel prefetch mechanism is a part of our future work.

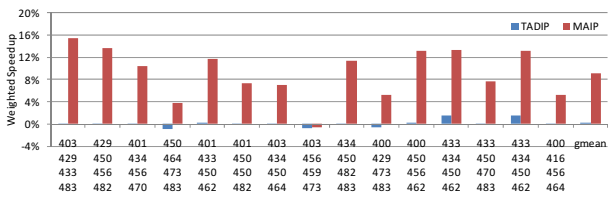

Figure 7: Weighted Speedup in Multi-core

5. REFERENCES

[1] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Syst. J.*, vol. 5, no. 2, pp. 78–101, 1966.

[2] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer, "Adaptive insertion policies for high performance caching," in *ISCA '07* , pp. 381–391, ACM, 2007.

[3] A. Jaleel, W. Hasenplaugh, M. Qureshi, J. Sebot, S. Steely, Jr., and J. Emer, "Adaptive insertion policies for managing shared caches," in *PACT '08*, pp. 208–219, ACM, 2008.

[4] H. Liu, M. Ferdman, J. Huh, and D. Burger, "Cache bursts: A new approach for eliminating dead blocks and increasing cache efficiency," in *MICRO '08*, pp. 222–233, IEEE Computer Society, 2008.

[5] L. Xiang, T. Chen, Q. Shi, and W. Hu, "Less reused filter: improving l2 cache performance via filtering less reused lines," in *ICS '09*, pp. 68–79, ACM, 2009.

[6] T. L. Johnson and W.-m. W. Hwu, "Run-time adaptive cache hierarchy management via reference analysis," *SIGARCH Comput. Archit. News*, vol. 25, no. 2, pp. 315–326, 1997.

[7] Y. Ishii, M. Inaba, and K. Hiraki, "Access map pattern matching prefetch: Optimization friendly method," *The first JILP Data Prefetching Championship (DPC-1)*, 2009.