

Adaptive Subset Based Replacement Policy for High Performance Caching

Liqiang He, Yan Sun, Chaozhong Zhang

► **To cite this version:**

Liqiang He, Yan Sun, Chaozhong Zhang. Adaptive Subset Based Replacement Policy for High Performance Caching. Joel Emer. JWAC 2010 - 1st JILP Workshop on Computer Architecture Competitions: cache replacement Championship, Jun 2010, Saint Malo, France. 2010. <inria-00492980>

HAL Id: inria-00492980

<https://hal.inria.fr/inria-00492980>

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.

Adaptive Subset Based Replacement Policy for High Performance Caching

Liqiang He Yan Sun Chaozhong Zhang

College of Computer Science, Inner Mongolia University

Hohhot, Inner Mongolia 010021 P.R.China

liqiang@imu.edu.cn

brandon.sunyan@gmail.com

zcz917@yahoo.com.cn

Abstract

An Adaptive Subset Based Replacement Policy (ASRP) is proposed in this paper. In ASRP policy, each set in Last-Level Cache (LLC) is divided into multiple subsets, and one subset is active and others are inactive at a given time. The victim block for a miss is only chosen from the active subset using LRU policy. A counter in each set records the cache misses in the set during a period of time. When the value of the counter is greater than a threshold, the active subset is changed to the neighbor one. To adapt the program behavior changing, set dueling is used to choose a threshold from different thresholds according to which one causes the least number of misses in the sampling sets. Using the given framework for this competition our ASRP policy gets a geometric average of improvement over LRU by 4.5% for 28 SPEC CPU 2006 programs and some programs gain improvements up to 50%.

1. Introduction

Cache replacement policy plays an important role in a cache design. The Least Recently Used (LRU) policy has been widely adapted in a microprocessor for the past decades. Recently progresses in this research topic include [1-10].

In this paper, we first propose a Subset Based Replacement Policy (SRP) for high performance caching. The idea is to divide one cache set into multiple subsets and victims are always taken from one active subset at a miss occurring. The reason of dividing is trying to maintain a longer access history through keeping part of the history in a group of blocks in the subset and make the cache more accurately captures the most frequently access blocks. And each subset maintains its own local LRU stack. On a cache miss, the incoming block is put into the local LRU position. And on a cache hit, the block is moved to the Most Recently Used (MRU) position like in LRU policy. The active subset periodically changes based on a count of misses to the set. To adapt the program behavior's changing, we further propose an Adaptive Subset Based Replacement Policy (ASRP) which uses a set dueling [3] technique to dynamically select a threshold from a pool of thresholds according to which one causes fewest misses in the sampling sets. Compare with LRU policy, ASRP obtains a 4.5% of geometric average miss reduction for 28 SPEC CPU 2006 programs, and some programs gain miss reductions up to 50%.

2. Experimental Methodology

We use the framework provided by this competition to do our experiments. All the simulations are done for a single-core processor. The details of the cache configurations are shown in Table 1.

Table 1. Baseline system configuration.

L1 I/D	32KB, 64B lines, 4/8-way, LRU
L2	256KB, 64B lines, 8-way, LRU, 10 cyc. hits
L3	1MB, 64B lines, 16-way, 30 cyc. hits

We select 28 SPEC CPU 2006 benchmarks in our

experiment. The cache access traces are obtained using Pin [11]. All the programs are simulated for 100M instructions after fast forwarding 40 billion instructions.

3. Subset Based Replacement Policy

3.1 Motivation

Cache replacement policy tries to keep the *best* blocks in a set in order to contribute maximum number of hits for the future accesses. Traditional LRU brings an incoming block directly to the MRU position and assumes that the block has the highest probability to be used again in the future. Such policy is good for workloads whose working sets are smaller than the available cache size or for workloads that have high temporal locality.

When the working set is greater than the cache capacity, the frequently used blocks in LRU policy can be easily evicted from LRU stack by less frequently used blocks or even by other frequently used blocks when it suffers a cache-thrashing problem. At this case, [3] points out that cache performance can be improved by retaining some fraction of the working set long enough that at least that fraction of the working set contributes to cache hits. Our Subset Based Replacement Policy (SRP) is a new method to adopt this idea.

3.2 Subset based replacement policy

Pierre [12] shows that thread migration can boost the performance of single program through exploiting the total cache capacity in multicore processor. The analysis result inspires us that grouping a cache (each group likes a private cache in a core) and keeping part of accessed blocks in a group (like the cache in a core where the program does not run) may also help to reduce the misses of a program. To do this, partition is a normally used scheme. The SRP policy we propose in this section is an adoption of this scheme.

In SRP, we evenly divide one set into multiple subsets. At a given time, only one subset is active which can accept the incoming block. A local LRU stack same as in LRU policy is maintained among the blocks in the subset. On a cache miss, victim is the least recently used block in the active subset, and the incoming block is not moved to local MRU position. And a cache hit will move the block into the local MRU position in the corresponding subset no matter it is active or not.

Fig. 1 explains the rationale. If a block B brought by the miss m_{i-1} is not accessed again before the new miss m_i occurs, then the new incoming block B' will replace B . But if it is accessed, such as the sequence of m_k, h_k, m_{k+1} , miss m_{k+1} will replace the block by m_{k-1} instead of the block K by m_k because K has been moved the local MRU position in h_k . And for a same reason, if an access h_{i+1} hits a block H which was brought by a missed access m_{i+1} in a non-active subset, H will not be replaced when the subset changes to active.

As shown in Fig.1, the active subset changing in SRP policy is based on a count of misses to the set.

When the misses are more than a threshold X , the active subset is changed to the neighbor one. In another word, we force up to X consecutive misses only replacing the blocks in the active subset, and keep the blocks in other subsets who were brought into the set before the X misses staying the cache. For a set with N subsets, it must after $(N-1)*X$ misses that a subset can change to active again, and during that period of time the blocks can stay in the subset without being replaced. So a greater value of X , a longer time that the blocks in non-active subsets can be kept in the set thus a more possibility that an old block in the subsets contributes to a hit if it is accessed again.

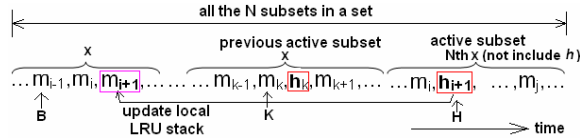


Figure 1. SRP policy.

As an example in Fig.2, a 4-ways cache set is divided into two subsets, and each subset contains two blocks. Subset 0 is set to active at initialization. The threshold of changing active subset is 4. From the figure, our SRP policy has 3 more hits than LRU policy. Keeping the replacement occurring in the active subset during the four misses potentially lengthen the window of access history thus some prior accessed blocks, block 1, 4 and 5 (accessed again after **six** or **five** consecutive misses) for example, can stay in the set and contribute to the hits for the future accesses. On the contrary, the LRU evicts a block if it is not accessed again during a **four** consecutive misses which is less than the history window stored in SRP policy and causes new miss in the future.

Trace	1	2	3	4	5	6	1	2	3	4	5	6
L	1		3	3	4	5	6	1	2	3	4	5
R		2	2	2	3	4	5	6	1	2	3	4
U	1	1	1	1	2	3	4	5	6	1	2	3
M/H	x	x	x	x	x	x	x	x	x	x	x	x
S	1	1	1	1	1	1	1	1	4	4	4	4
R	1	2	3	4	4	4	4	4	4	1	1	6
P					5	5	5	5	5	5	5	5
M/H	x	x	x	x	x	x	✓	✓	✓	✓	✓	x

Figure 2. Example of SRP policy.

3.3 Hardware implementation

The hardware structure of SRP policy is shown in Fig.3. For a 16-ways last-level cache (LLC) used in this championship, each set is evenly divided into four subsets. A dedicated 2bits *active_group* counter in the set points to the active subset and the other subsets are inactive. Another *set_miss* counter is used to count the number of misses during a period of time. When the value is greater than the value (all sets use the same value) in the global *threshold* register, the active subset is moved to the neighbor one. In our design, we experientially set the *set_miss* as a 7 bits counter.

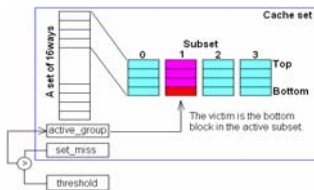


Figure 3. Hardware implementation of SRP policy. At initialization time, the *active_group* and

set_miss counters are set to zero, and the value in *threshold* register is set to an experiential value, 4 for example.

3.4 Case study of thrashing workload

Fig.4 shows the cache access pattern of *xalancbmk* in an arbitrary selected set, No. 513, represented by the unique block address. The Fig.4a is the whole picture of the accessing which has a very regular pattern, but the working set (approximate 35 blocks) is greater than the cache capacity, 16, thus causing a thrashing problem. And the Fig.4b is a zoom in snapshot of the first two hundreds accesses. It shows that the frequently accessed blocks express a cyclic pattern.

For such workload, at a cache miss happening SRP uses the LRU block in the active subset as a victim, and keep the other blocks staying in the cache. It assumes that the new incoming block will not be used again in the near future because of the big working set and limited cache size, and hopes the other blocks being retained in the cache can contribute hits for future accesses. When a block has a hit in a subset (no matter it is active or not), it is moved to MRU position of the subset thus the future replacement will select other LRU block if the subset is active at that moment. Through the LRU replacing on miss and MRU placing on hit, the cache can potentially hold maximum number of the frequently used blocks with the limited space.

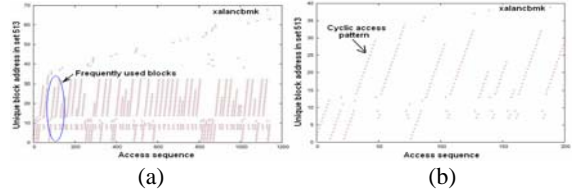


Figure 4. Unique block addresses (a: whole trace and b: a zoom in) accessed by *xalancbmk* in set 513.

SRP uses a threshold to control the frequency of active subset changing. A bigger threshold value, a less frequent changing of the active subset thus the more possible is the older blocks can stay in the cache. On the contrary, a smaller threshold value, a more frequent changing of the active subset thus having more chance to select victim from multiple LRU blocks in different subsets and letting each LRU block staying in the cache longer than at the case with a bigger threshold.

Fig. 5 shows the cache performance of *xalancbmk* varies with different threshold values. SRP successfully reduces the cache misses comparing with LRU. Fig.5 also shows that the performance of SRP policy is very sensitive to the threshold value. For different thresholds, SRP can hold different fractions of cache data, and at threshold 4 and 8 it obtains the best performance.

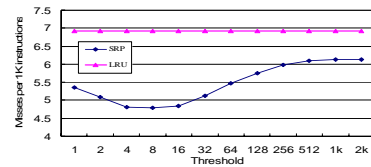


Figure 5. MPKI vs. threshold values for *xalancbmk*.

3.5 Case study of LRU-friendly workload

Fig. 6 shows the unique block addresses accessed by a LRU-friendly workload, *GemsFDTD*, in set 513.

