Journal of Knowledge Information Technology and Systems(JKITS), Vol. 10, No. 2, pp. 145~152, April 2015

Journal of Knowledge Information Technology and Systems

ISSN 1975-7700

http://www.kkits.or.kr

A Cache Replacement Policy for Improving the Performance of Last Level Cache in Processors

Sang-Jeong Lee[1], Young-Il Cho[2*]

[1]*Department of Computer Engineering, Soonchunhyang University*
[2]*Department of Computer Science, University of Suwon*

ABSTRACT

Reduction in cache miss rates continues to be an important issue in processor design, especially at the last level cache(LLC). The Least Recently Used(LRU) replacement policy has been widely adapted in processors for the past decades. The LRU replacement policy represents the cache blocks in a set as LRU stack. So, it only use the recency information. And it is expensive to implement in hardware. Also in some workload, it raises the thrashing, i.e., blocks with high reuse evicting each other from the cache.

We propose an new replacement policy to solve problems of LRU replacement policy. In the proposed policy, a incoming block is inserted at the bottom of the set. To exploit the frequency information, a hit block exchanges its position with its adjacent block above in the set. A victim block can be chosen from bottom. Selecting the bottom block as the victim makes the proposed policy to protect the cache from polluting by less frequently used blocks. It also proposes to divide the blocks in a cache set into groups to resolve the thrashing. The victim block is selected alternately from each group. The proposed policy reduces the average MPKI(Miss Per Kilo Instructions) of the baseline 1MB 16-way LLC cache using LRU replacement policy by 12.9%. It reduces the storage requirement by 47% compared to LRU replacement policy.

KEYWORDS: Replacement policies, LRU, Recency, Frequency, Thrashing, Last level cache, Victims

ARTICLE INFO: Received 16 February 2015, Revised 16 March 2015, Accepted 10 April 2015.

*Corresponding author is with the Department of Computer Science, University of Suwon, 17 Wauangil Bongdam-Eup Hwaseong-Si, Gyeonggi-Do, 445-743, KOREA. *E-mail address*: yicho@suwon.ac.kr

1. 서 론

CPU와 주기억장치 사이의 속도 차이를 완화시키기 위해 캐시 메모리를 사용하며 캐시의 효율성을 증가시키기 위해 다중 레벨 캐시가 일반화되어 있다. 여러 매개변수가 효율적인 다중 레벨 캐시를 설계하는데 영향 주는데 그 중에서 중요한 매개변수가 교체정책이다. 교체정책은 전체 캐시 성능에 영향 줄 뿐 아니라 대역폭 이용과 응답시간에도 영향 준다. 효율적이지 못한 교체정책은 캐시미스를 증가시키고 많은 트래픽을 유발시킬 수 있기 때문에 미스 페널티를 증가시킨다.

LRU(Least Recently Used) 교체정책은 수십 년간 온-칩 프로세서에서 가장 일반적으로 사용되고 있는 캐시 교체정책이다. LRU 정책은 유효한 액세스 히스토리를 유지할 수 있기 때문에 높은 지역성을 갖는 워크로드에 대해서 특히 좋은 성능을 얻을 수 있다. LRU 교체정책은 캐시 액세스의 최신 정보만 고려하며 적중된 블록을 MRU(Most Recently Used) 위치에 놓기 때문에 액세스 빈도는 무시된다. 또한 LRU 교체정책은 워킹 셋이 가용 캐시 크기보다 클 때 순환 액세스 패턴에서 스래싱(thrashing)이 발생될 수 있고 구현 하드웨어 단가가 비싸다.

캐시미스 때 LRU 교체정책같이 들어오는 블록을 MRU 위치에 삽입하는게 아니라 LRU 위치로 삽입하는 LIP(LRU Insertion Policy)가 제안되었다[1]. LIP는 스래싱 문제를 해결하나 재사용 빈도는 높지만 즉시 재사용되지 않는 블록의 경우 바로 다음에 참조하는 블록에 의해 퇴출되기 때문에 재참조 시 캐시미스를 발생시키는 문제점을 갖는다. 그 외에도 LRU의 문제점을 해결하기 위해 수정된 LRU[2,3,4], hybrid 방법[5,6,7]이 연구되었다. 또한 LLC 미스율을 개선시키기 위해 재사용되지 않는 블록에 대해 LLC를 바이패스 시키는 방법이 연구되었다[8].

LRU 교체정책은 빈번히 사용되는 블록들을 덜 빈번히 사용되는 블록들로 교체시킬 수 있다. 따라서 교체할 때 빈도수 정보를 사용할 수 있다면 빈번히 사용되는 블록들을 계속 유지할 수 있으므로 캐시미스를 감소시킬 수 있다. 일반적으로 순환 액세스 패턴은 비순환 액세스 패턴과 섞여있다. 교체정책이 액세스 빈도 정보의 장점을 이용한다면 순환 액세스되는 블록들을 캐시에 유지하고 비순환 액세스되는 블록들을 캐시에서 빨리 퇴출시키므로 스래싱 문제를 해결할 수 있다.

본 논문에서는 기존의 LRU 교체정책보다 성능을 개선시키고, LRU 교체정책에 비해 스토리지 요구와 회로 복잡도를 상당히 감소시키는 새 캐시 교체 정책을 제안한다. 제안한 교체정책은 캐시미스 시 들어오는 블록을 큐의 바닥으로 삽입하므로서 연속적으로 사용되지 않는 블록들을 빨리 퇴출시킨다. 또한 빈번이 사용되는 블록이 큐 길이 보다 작은 사용 빈도를 가질 경우, 큐의 꼭대기에 있는 블록들은 액세스되지 않아도 큐에 계속 머무는 문제를 해결하기 위해 캐시 셋에 있는 블록들을 분할(16-way 캐시에서 16 블록을 4 그룹으로 분할)한다. 제안한 교체정책에서 적중된 블록은 큐의 위쪽으로 이동시킨다. 따라서 빈번히 액세스되는 블록들은 덜 빈번히 액세스되는 블록들 위쪽에 위치하므로 빈도 정보를 고려하게 된다.

LRU 교체정책은 들어오는 블록을 MRU 위치로 삽입하기 때문에 16-way 셋 연관일 경우 모두 16 블록의 상태를 갱신해야하지만 제안한 교체정책은 16-way 셋을 4개 그룹으로 분할하고, 들어오는 블록을 교체 그룹의 바닥으로 삽입하므로 교체 그룹의 바닥 블록만 갱신하고 나머지 그룹은 갱신할 필요 없기 때문에 갱신 시간을 단축시킬 수 있다.

제안한 교체정책은 베이스라인 1M 16-way L3 캐시의 평균 MPKI를 12.9% 감소시키고, 스토리지

요구를 47% 감소시키며, 기존의 LRU 교체정책에 비해 회로 복잡도를 간단화시킨다. 또한 시스템 성능을 평균 6.8% 개선시킨다.

2. 제안한 교체정책

캐시 미스가 발생했을 때 빈도 정보를 사용하는 교체정책들이 제안되었으나 이들 교체정책은 추가 카운터나 테이블 등 많은 구현 하드웨어를 요구한다[9,10]. 본 논문에서는 카운터, 테이블, 추가 구조 등이 필요 없이 최신 및 빈도 정보를 고려하는 효율적인 교체정책을 제안한다. 제안한 교체 방법은 삽입 정책, 프로모션 정책, 희생자 선택 정책, 셋 분할 정책을 포함한다.

첫째, 삽입 정책은 캐시 미스 때 들어오는 블록을 큐의 바닥에 삽입한다. 들어오는 블록은 단지 한 번 액세스되었고 큐에 있는 다른 블록들은 들어오는 블록보다 더 많이 액세스되었기 때문에 들어오는 블록을 바닥에 삽입하는 것은 빈도 정보를 교체정책에 반영시킨다.

둘째, 프로모션 정책은 적중된 블록을 큐에서 위에 있는 블록과 교환한다. 다만 큐의 꼭대기에 있는 블록이 적중이면 위치 교환이 필요 없다. 프로모션 정책은 빈번히 사용되는 블록들을 큐의 꼭대기 쪽으로 이동시키고 덜 빈번히 사용되는 블록들을 큐의 바닥 쪽으로 이동시킨다. 따라서 가장 최근에 자주 액세스된 블록들은 꼭대기 쪽으로 이동되어 바닥으로부터 멀어지므로 큐가 빈도 정보와 최근 정보를 갖게 한다.

셋째, 캐시미스 시 희생 블록은 큐의 바닥에서 선택된다. 바닥 블록은 큐에서 빈도수가 가장 작은 블록이기 때문에 바닥 블록을 희생자로 선택하는 것은 덜 빈번히 사용되는 블록들을 캐시로부터 제거시킨다.

캐시미스 시 들어오는 블록을 큐의 바닥에 삽입하면 빈도수는 크나 연속해서 사용되지 않는 블록을 퇴출시키는 문제점을 유발할 수 있다. 이런 문제를 해결하기 위해 셋을 여러 그룹(큐)으로 분할하는 정책을 사용한다. 셋 분할 정책은 들어오는 블록들에게 여러 엔트리를 제공하기 때문에 연속적인 캐시 미스 시에도 들어오는 블록들을 다른 그룹에 삽입할 수 있으므로 프로모션될 기회를 갖기 전에 서로 퇴출시키는 문제를 해결할 수 있다. 큐의 꼭대기에 도달하기 위해서는 임의 블록의 적중 수가 큐 길이보다 크거나 같아야한다. 가장 빈번이 사용되는 블록이 큐 길이 보다 작은 사용 빈도를 갖는다면 어떤 캐시 블록도 큐의 꼭대기에 도달할 수 없기 때문에 큐의 꼭대기에 있는 블록들은 다시 사용되지 않아도 큐에 계속 남게 되는 문제점을 갖게 되는데, 이 문제는 큐의 길이를 작게 하면 해결된다. 셋 분할 정책은 셋을 여러 그룹으로 분할하므로 서 길이를 작게 만든다. 예를 들어, 16 엔트리 셋일 때 임의 블록이 꼭대기에 도달하려면 16번 이상 액세스되야 한다. 그러나 한 셋을 4개 그룹으로 분할한다면 각 그룹은 4 엔트리를 갖기 때문에 4번 이상 액세스되는 블록은 큐의 꼭대기에 도달할 수 있다. 셋 분할 정책은 16-way 셋일 때 4개 그룹(G_0, G_1, G_2, G_3)으로 분할한다. 희생자는 4개 그룹의 희생자 대상 블록들로부터 교대로 선택되며 각 그룹은 교체정책을 독립적으로 사용한다. 즉, 들어오는 블록은 선택된 그룹(selected group)의 희생자 블록 위치로 삽입되며 선택된 그룹의 상태만 갱신되고 나머지 그룹들의 상태는 갱신되지 않으며, 다음 미스 때 들어오는 블록은 다음 선택된 그룹에 삽입된다. 다음 캐시 미스 때 희생자 블록을 갖는 선택된 그룹을 지시하기 위해 각 셋에 2비트의 RRC(Round Robin Counter)를 둔다. RRC는 초기에 0으로 설정되고 캐시 미스가 발생할 때마다 '1' 씩 증가되고 RRC가 '3' 일 때 미스가 발생하면 '0' 으로 설정한다.

캐시 미스 때 들어오는 블록들을 여러 그룹으로 교대로 삽입하므로 서 들어오는 블록들이 서로 퇴출시키는 문제를 해결한다. 즉, 새로 들어오는 블록은 직전에 삽입된 블록을 퇴출시키지 않으므로 바닥 블록들이 프로모션될 기회를 갖게 된다.

순환적으로 액세스되는 블록들이 다수의 그룹에 할당되고 희생자는 이들 그룹으로부터 교대로 선택되기 때문에 순환적 액세스 패턴에서 발생하는 스래싱 문제도 해결될 수 있다.

<그림 1>은 교체정책과 프로모션 정책을 설명한다.

```
Algorithm Cache_Management
begin
    if accessed block A is hit
        if A is top of the group
            return;
        // 프로모션
        else exchange A with a block above A;
    else // 캐시 미스
        begin
            evict a bottom block of selected group ;
            insert a incoming block
                into bottom of selected group;
            //change selected group into next group;
            ++RRC;
            if RRC == 4
                RRC = 0;
            selected group = GRRC;
        end
end.
```

그림1. 제안된 캐시 관리 알고리즘

Figure 1. The proposed cache management algorithm.

3. 실험방법

3.1 캐시구조

캐시미스와 프로세서 성능을 측정하기 위해 SimpleScalar[11]를 사용한다.

<표 1>은 베이스라인 캐시 구성을 보여준다. 베이스라인 L1 명령어 캐시는 32KB 2-way 셋 연관(set associative)이고 데이터 캐시는 4-way 셋 연관이다. 베이스라인 L2 캐시는 256MB 8-way 셋 연관이다. L3 캐시는 1MB 16-way 셋 연관이고 베이스라인의 모든 레벨의 캐시는 64B 라인 크기를 사용한다. 표에서 LRU는 기존의 LRU 교체정책을 의미하고, LRU 교체정책을 베이스라인 캐시에서 사용한다.

표1. 베이스라인 캐시 구성

Table 1. Configuration of baseline cache

L1 Inst/Data	32KB, 64B 라인, 2-way/4-way, LRU, 1 cycle latency
L2	256KB, 64B 라인, 8-way, LRU, 8 cycle latency
L3	1MB, 64B 라인, 16-way, LRU, 20 cycle latency

3.2 벤치마크

실험에서 SPEC CPU2000 벤치마크를 사용한다. 콜드 스타트(cold start) 미스의 영향을 제거하기 위해 준비시간(warm-up)을 갖고 reference 입력을 사용하여 SimPoint[12]로 부터 각 벤치마크에 대해 250M 명령어 트레이스를 얻었다. 어떤 교체정책도 강제 미스(compulsory miss : 블록을 처음 액세스할 때 캐시에서 발생하는 미스)에는 영향 주지 않으므로 강제 미스가 캐시미스의 큰 백분율을 차지할 경우 교체정책은 캐시 성능에 영향주지 못한다. 따라서 26개 벤치마크로부터 강제 미스가 전체 캐시 미스의 50% 미만인 16개 워크로드들을 실험에서 선택했다.

4. 실험 결과 및 평가

이 장에서는 셋에서 몇 번의 미스가 발생했을 때 희생자 그룹을 변경할지를 알아보고, 캐시 크기

가 변할 때 캐시미스에 주는 영향을 알아본다. 또한 셋을 몇 개 그룹으로 분할했을 때 가장 미스를 감소시키는지를 알아보고, 프로세서의 마지막 레벨 캐시의 교체정책으로 제안한 정책을 사용했을 때 성능 향상을 알아본다.

4.1 희생자 그룹 선택

캐시미스가 발생하면 희생자를 선택해야한다. 희생자는 4개 그룹 중 한 그룹의 희생자를 선택할 수 있다. 임의의 그룹이 희생자 그룹일 때 해당 셋에서 몇 번의 미스가 발생했을 때 희생자 그룹을 변경할지를 결정하기 위해 여러 임계치에 대해 실험하여 보았다.

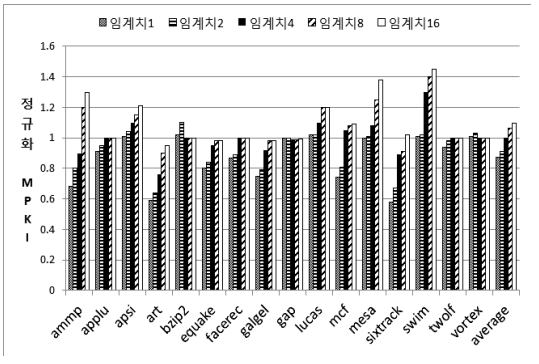

그림2. 여러 임계치에 대한 정규화된 MPKI
Figure 2. Normalized MPKI for different thresholds.

<그림 2>는 임계치를 1, 2, 4, 8, 16으로 변경하면서 가장 캐시 미스를 적게 발생하는 임계치를 구하였다. 임계치가 16을 초과할 경우 캐시 미스가 상당히 증가하므로 16을 초과하는 것은 실험에서 제외하였다. 실험결과 벤치마크마다 차이는 있지만 임계치를 1로 했을 때 평균적으로 가장 적은 캐시 미스를 보였다. 임계치가 1일 경우는 해당 셋에서 매 캐시미스마다 희생자 그룹을 변경하는 것을 의미한다. 따라서 본 논문에서는 해당 셋에서 캐시 미스가 발생 할 때마다 희생자 그룹을 다음 그룹으로 선택하도록 하였다.

4.2 캐시 크기의 영향

<그림 3>은 캐시 연관을 16-way로 고정시키고, L3 캐시 크기가 1MB, 2MB, 4MB, 8MB 했을 때 베이스라인 캐시에 대한 정규화 MPKI를 보여준다. 제안한 교체정책은 캐시 크기가 증가하면 모든 벤치마크에서 MPKI가 감소됨을 보여준다.

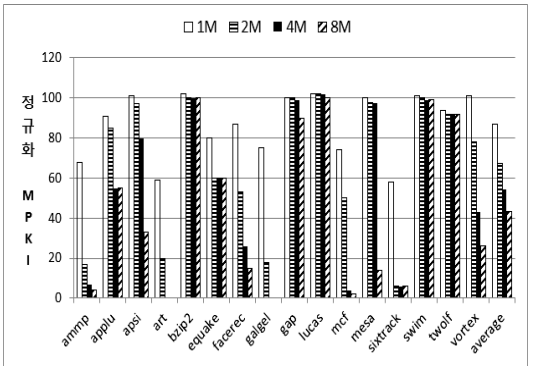

그림3. 여러 캐시 크기에 대한 정규화된 MPKI
Figure 3. Normalized MPKI for different cache sizes.

4.3 분할 그룹 수의 영향

<그림 4>는 16-way 셋을 기준으로 그룹을 1, 2, 4, 8개(1-GR, 2-GR, 4-GR, 8-GR)로 분할했을 경우 베이스라인으로 사용한 LRU와 비교한 MPKI 감소를 보여준다. 1-GR는 셋이 한 개의 그룹을 가지므로 분할하지 않은 경우를 의미한다.

실험 결과는 벤치마크마다 최대로 MPKI를 감소시킨 그룹 수가 다르지만 4개 그룹으로 분할했을 경우가 최대 46.8%(art), 평균 12.9%로 가장 좋은 결과를 얻었다.

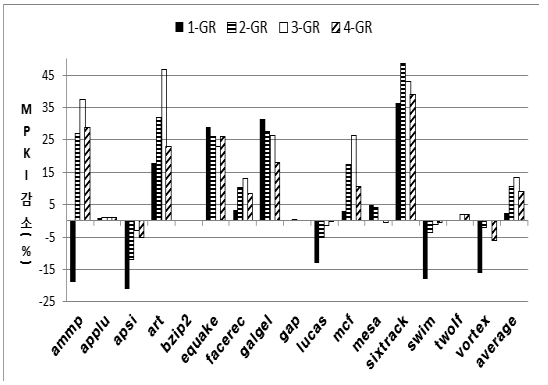

그림4. 분할 그룹 수에 대해 LRU보다 제안 방법의 MPKI 감소율

Figure 4. MPKI reduction rate over LRU of proposed scheme for the number of divided groups

4.4 시스템 성능 평가

제안한 교체정책이 프로세서에 주는 영향을 평가하기 위해 SimpleScalar를 사용하였다. 시스템 성능을 측정하기 위한 프로세서는 사이클 당 4 개 명령어가 이슈되고, 4개 기능 유닛(functional units)을 가지며, 명령어 윈도우 크기는 32로 가정하였고, Hybrid 분기예측기(bimodal + 2-level)를 사용하며 분기예측 미스 페널티는 10 사이클로 하였다.

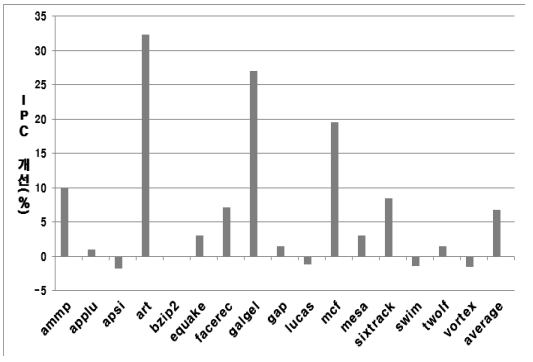

그림5. LRU보다 제안 알고리즘의 성능 개선율

Figure 5. IPC improvement rate over LRU of proposed scheme

<그림 5>는 제안한 교체정책을 사용했을 때 베이스라인보다 IPC(Instruction Per Cycle) 개선을 보여준다. IPC 개선은 최대 32.3%(art)이고 평균 6.8%이다. 제안한 교체정책은 스토리지와 회로 복잡도를 감소시키면서도 LLC(L3) 캐시의 액세스 사이클을 증가시키지 않는다.

4.5 하드웨어 및 회로 복잡도

제안한 교체정책은 그룹(4개 엔트리)당 상태 저장을 위해 8(=4*2비트)비트가 필요하다. 따라서 한 셋은 4개 그룹을 가지므로 한 셋에 32비트를 요구한다. 또한 희생자 그룹을 선택하기 위해 한 셋당 2비트의 RRC가 필요하다. 1MB L3 캐시의 경우 34 비트 * 1024 셋 = 34K비트(4.25KB)가 필요하다.

기존의 16-way LRU는 한 엔트리의 상태 저장을 위해 4비트가 필요하므로 한 셋을 위해 64비트가 필요하다. 1MB L3 캐시인 경우 64비트 * 1024 셋 = 64K 비트(8KB)가 필요하다. 따라서 제안한 교체정책을 사용하면 LRU보다 47% 하드웨어 스토리지 감소를 얻는다.

LRU에서는 들어오는 블록이 MRU 위치로 삽입될 때 모두 16 캐시 블록의 상태가 갱신되지만 제안한 방법은 들어오는 블록이 삽입될 때 바닥 블록만 갱신되기 때문에 회로 복잡도와 갱신 시간이 감소된다.

5. 결 론

본 논문에서는 수십 년 동안 프로세서에서 널리 사용되고 있는 LRU 교체정책의 문제점을 해결시킨 새로운 캐시 교체정책을 제안하였다. 제안한 교체정책은 LRU 교체정책보다 성능을 개선시키고, 스토리지 요구와 회로 복잡도를 상당히 감소시켰다. 제안 정책은 적중된 블록을 큐의 위쪽으로 이동시키므로 서 빈도 정보도 고려하였고, 캐시미스 시

들어오는 블록을 큐의 바닥으로 삽입하므로 서 사용 빈도가 작은 블록들을 빨리 퇴출시킨다. 또한 셋 분할을 통해 높은 재사용을 갖는 블록들이 캐시로부터 서로 퇴출시키는 스래싱 문제를 해결하였다.

실험 결과 제안한 교체정책은 LRU를 사용하는 베이스라인 1MB 16-way L3 캐시에서 평균 MPKI를 12.9% 감소시키고, 스토리지 요구를 47% 감소시키며, 시스템 성능을 평균 6.8% 개선시켰다.

References

[1] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer, *Adaptive insertion policies for high-performance caching,* In ISCA-07, Vol. 35, Issue 2, pp. 381-391, 2007.

[2] H. Dybdahl, p. Stenstom, and L. Natvig, *An LRU-based replacement algorithm augmented with frequency of access in shared chip-multiprocessor caches,* MEDEA-06, pp. 45-52, 2006.

[3] S. Jiang and X. Zhang, *LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance,* In Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, Vol. 30, Issue 1, pp. 31-42, 2002.

[4] W. Wong and J. L. Baer, *Modified LRU policies for improving second-level cache behavior*, In HPCA-2000, pp. 49-60, 2000.

[5] D. A. Jiménez, *Insertion and promotion for tree-based pseudoLRU last-level caches*, MICRO-13, pp. 284-296, 2013.

[6] M. K. Qureshi, D. N. Lynch, O. Mutlu, and Y. N. Patt, *A case for MLP-aware cache replacement,* In ISCA-06, Vol. 34, Issue 2, pp. 167-178, 2006.

[7] R. Subramanian, Y. Smaragdakis, and G. Loh, *Adaptive caches: Effective shaping of cache behavior to workloads,* In MICRO-06, pp. 385-396, 2006.

[8] S. Gupta, H. Gao, H. Zhou, *Adaptive Cache Bypassing for Inclusive Last Level Caches*, In IPDPS-2013, pp.1243-1254, 2013.

[9] J. T. Robinson, and M. V. Devarakonda, *Data cache management using frequency-based replacement*, SIGMETRICS '90, Vol. 18, Issue 1, pp. 134-142 May 1990.

[10] D. Lee, J. Choi, j. Kim, H. Noh, S. Min, Y. Cho, C. Kim, *LRFU : A Spectrum of Policies that Subsumes LRU and LFU Policies*, IEEE Trans. on Computer Vol.50, No.12, pp.1352-1361, Dec. 2001

[11] D. Burger, and T. M. Austin. *The simplescalar tool se*t, Univ. of Wisconsin-Madison. Technical Report #1342, 1997.

[12] SimPoint home page : http://cseweb.ucsd.edu/~calder/simpoint/

프로세서에서 마지막 레벨 캐시의 성능 개선 위한 캐시 교체정책

이상정[1], 조영일[2]

[1]*순천향대학교 컴퓨터공학과*

[2]*수원대학교 컴퓨터학과*

요 약

캐시 미스율의 감소는 프로세서 설계에서 중요하고 마지막 레벨 캐시에서 더욱 중요하다. LRU(Least Recently Used) 교체정책은 과거 수십 년간 프로세서

에서 널리 이용되고 있다. LRU 교체정책은 셋에 있는 블록들을 LRU 스택으로 유지하므로 최신 정보만 이용하며, 구현 하드웨어 단가가 높다. 또한 LRU 교체정책은 어떤 워크로드들에서 높은 재사용을 갖는 라인들이 캐시로부터 서로 퇴출시키는 스래싱 문제를 일으킨다. 본 논문에서는 LRU 교체정책의 문제점을 해결하는 새로운 교체정책을 제안한다. 제안된 정책에서 들어오는 블록은 셋의 바닥에 삽입되며, 빈도 정보를 이용하기 위해 적중된 블록은 셋에서 바로 위 블록과 교환한다. 사용 빈도수가 낮은 블록들에 의해 캐시가 오염되는 것을 방지하기 위해 희생자 블록을 바닥으로부터 선택한다. 제안한 정책은 스래싱을 해결하기 위해 캐시 셋의 블록들을 여러 그룹으로 분할하며, 희생자 블록은 각 그룹으로부터 교대로 선택한다. 제안한 교체정책은 LRU 정책을 사용하는 베이스라인 1M 16-way LLC(Last Level Cache) 캐시의 평균 MPKI(Miss Per Kilo Instructions)를 12.9% 감소시키고, LRU 교체정책에 비해 스토리지 요구를 47% 감소시킨다.

Sang-Jeong Lee received the B.S., M.S., Ph.D. in electronic engineering from the Hanyang University in 1983, 1985 and 1988, respectively He is currently a professor at the department of computer science and engineering in Soonchunhyang University, Korea. His current research areas are computer architecture, network application and embedded system.

E-mail address : sjlee@sch.ac.kr

Young-Il Cho received the B.S., M.S., Ph.D. in electronic engineering from the Hanyang University in 1980, 1982 and 1985 respectively. He has been a professor in the Department of Computer Science at University of Suwon since 1986. His current research interests include computer architecture, high performance microarchitecture and global sensor network.

E-mail address: yicho@suwon.ac.kr