Journal of Knowledge Information Technology and Systems

ISSN 1975-7700

http://www.kkits.or.kr

A Filtering Scheme to Improve the Performance of Last Level Cache

Young-Il Cho*

Department of Computer Science, University of Suwon

ABSTRACT

The last level cache(LLC) is commonly managed using LRU policy. However, LRU has a high overhead cost of moving cache lines into the most recently used position whenever a cache line is accessed. Also, LRU is prone to cache pollution when a sequence of single-use memory accesses that are larger than the cache size is fetched from memory. Cache performance and efficiency can be improved if some subset of these reuse lines can reside in the cache longer. Previous schemes approach this by bypassing never reused lines(0-reused lines). But, sometimes they deliver no benefit due to the lack of never reused lines. This paper proposes a new mechanism that filters out not only 0-reused lines but also 1-reused lines and accurately predicts 0/1-reused lines from incoming lines. Filtering of 0/1-reused lines provides more opportunities to fit the working set into cache size. Our proposed scheme is evaluated using a simulation environment where its effectiveness and performance-improvement capabilities are demonstrated. We present experimental results showing miss rate and IPC(Instruction Per Cycle) comparison of the proposed scheme and OBM(Optimal Bypass Monitor) against LRU for SPEC CPU2006 benchmarks. The result shows that the proposed scheme and OBM can improve IPC by an average of 20.1% and 14.4%, respectively. And the proposed scheme reduces the miss rate by 20.6% compared to LRU.

ⓒ 2017 KKITS All rights reserved

KEYWORDS: LLC, 0-reused line, Cache pollution, Bypassing, Miss rate, OBM

ARTICLE INFO: Received 8 August 2017, Revised 13 October 2017, Accepted 13 October 2017.

*Corresponding author is with the Department of Computer Science, University of Suwon, 17 Wauangil Bongdam-Eup Hwaseong-Si, Gyeonggi-Do, 18323, KOREA. *E-mail address*: yicho@suwon.ac.kr

www.kci.go.kr

1. 서 론

프로세스 기술의 발전으로 지난 수십 년 동안 CPU 성능은 크게 향상되었지만 메모리 기술은 같은 속도로 향상되지 않았다. 따라서 메모리로부터의 로드 요청을 수행할 때마다 상당한 오버헤드가 발생한다. 이런 메모리 액세스에 의한 성능 저하를 줄이기 위해 캐시 구조의 혁신과 개선을 가져왔으며 효율적인 LLC(Last Level Cache) 설계는 프로세서 성능에 중요하기 때문에 지속적인 연구의 중심에 있다. 그러나 고성능 프로세서에서 LLC는 효율적으로 사용되지 못하고 있으며 여러 연구에서 기존 캐시 설계의 효율성을 지적하였다[1-5].

SPEC CPU2006 벤치마크 중에서 약 50%가 LLC로 가져온 블록의 90% 이상이 퇴출되기 전에 다시 액세스 되지 않는다는 것을 보여주었다[3-9]. 재 액세스되지 않는 블록들은 LLC에 있는 동안 캐시 적중에 기여하지 못한다. 만약 재 액세스되지 않는 블록들이 캐시로 삽입되면서 유용한 캐시 블록들을 대신 퇴출시킨다면 불필요하게 캐시 공간을 차지하기 때문에 캐시 오염과 스래싱(thrashing)의 원인이 된다. LLC에서 재 액세스되지 않는 주된 이유는 상위 레벨 캐시(L1/L2)에 의해 지역성이 여과되기 때문이다. 즉, 상위 레벨에서 적중되기 때문에 LLC에서는 재 액세스가 발생하지 않는다.

이론적으로 가장 작은 미스 율을 얻기 위해서 LLC 캐시는 교체 대상자로 미래에 가장 멀리 액세스될 캐시라인을 선택하는 OPT를 사용해야한다. OPT 정책[10]은 미래에 대한 지식에 의존하기 때문에 비실제적이므로 대신에 LRU 정책이 간단성 때문에 실제로 사용된다.

LRU는 메모리 비집중적인 워크로드에서는 잘 수행되지만 워킹 셋이 캐시보다 큰 메모리 집중적인 워크로드에서는 빈약하게 수행되므로 OPT와의 격차가 크다. 여러 연구가 그 격차를 메우기 위해 제안되고 있다. 주 제약은 워크로드보다는 워킹 셋이 캐시 크기에 적합하지 않다는 것이다. 사실, 그 격차는 충분한 캐시 용량이 제공된다면 극적으로 좁혀질 수 있다. 해법은 OPT와 같이 워킹 셋의 일부를 캐시에 충분히 오랫동안 유지시키는 것이다. 그러면 최소한 유지시킨 부분이 캐시 적중에 기여하게 된다.

재사용되지 않는 라인(0-재사용 라인)들을 바이패스 시키는 것도 워킹 셋을 감소시키는 좋은 방법이지만 재사용되지 않는 라인 수의 제약 때문에 성능을 제한한다.

본 논문에서 제안한 방법은 워킹 셋을 감소시키는 것을 목표로 0-재사용 라인뿐 아니라 1-재사용 라인들도 LLC에 들어가는 것을 제한하여 워킹 셋이 캐시 크기에 적합하도록 기회를 증가시키므로서 바이패싱 메커니즘의 한계를 극복한다. 이를 위해 이전의 재사용 정보에 따라 미스 라인의 재사용 빈도를 예측하는데 사용되는 재사용 빈도 예측기(Reuse Frequency predictor : RFP)를 제안한다. 재사용 빈도 예측에 기초하여 0/1-재사용 라인들을 식별하여 BB(Bypass Buffer)에 저장한다. 0/1-재사용 라인들을 여과시키므로 서 LLC에는 워킹 셋 중 자주 재사용되는 라인들만 저장하기 때문에 캐시 적중률을 증가시키고 스래싱도 방지할 수 있다.

0/1-재사용 라인들이 짧은 생존 시간(life span)을 갖는다는 관찰을 통해 작은 BB를 사용하여도 그들을 완벽하게 이용하면서 빨리 퇴거시키는 것을 가능하게 한다.

제안 방법은 LLC 캐시 교체정책의 수정 없이 메모리 적극적인 워크로드에 대해 LLC의 성능을 개선시킴을 보여준다.

실행 구동 시뮬레이션에 기초한 실험 결과는 2MB 16-way LRU LLC 캐시에 512-엔트리 BB와 RFP를 추가할 경우, SPEC2006 벤치마크에 대해 4MB 16-way LRU를 갖는 베이스라인 보다 평균 미

www.kci.go.kr

스 율을 20.1% 개선시키고, IPC를 20.6% 개선시킨다.

2. 연구 동기

LRU 교체정책을 사용하는 LLC에서 시간적 지역성이 여과되기 때문에 재 액세스가 발생하지 않고, 워킹 셋의 크기가 캐시 크기를 초과하는 경우 캐시 오염과 스래싱(thrashing)의 원인이 된다.

라인들의 재사용 빈도를 관찰하기 위해 LLC의 성능이 실행시간에 크게 영향을 주는 벤치마크에 대해 측정하였다. 즉, 23개 SPEC CPU2006 벤치마크 중에서 LLC 크기를 1MB에서 4MB로 증가할 때 최소 10% 이상 성능이 개선되는, LLC 성능에 종속적인 10개 벤치마크에 대해 평가한다.

<그림 1>은 LRU 교체정책을 사용하는 2MB LLC에서 라인이 LLC로 삽입 후 LLC에 있는 동안 재 액세스되지 않는 라인의 양을 보여준다.

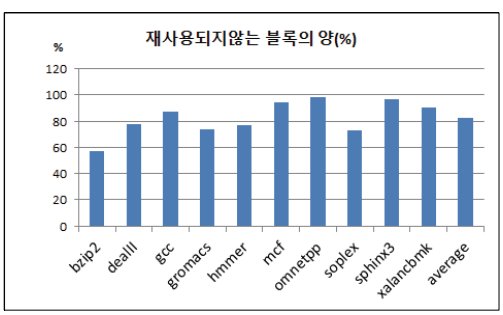

그림 1. 캐시에 삽입 후 재 액세스되지 않는 LLC 라인의 양
Figure 1. Fraction of LLC blocks that are never re-accessed after being brought into the cache

이들 재 액세스되지 않는 라인들은 LLC에 있는 동안 어떤 적중에도 기여하지 못한다. 만약 캐시로 삽입될 때 더 유용한 캐시 라인들을 교체한다면 그 라인들이 불필요하게 캐시 공간을 차지하여 캐시 오염과 스래싱의 원인이 된다. <그림 1>에서 SPEC CPU2006 벤치마크에서 절반 정도가 LLC로 가져온 라인의 90% 이상이 퇴출되기 전에 재 액세스되지 않음을 보여준다. 평균, LLC 블록의 80.7%가 퇴출되기 전에 재 액세스되지 않는다. 이는 다른 연구[4-7]에서도 비슷한 관찰을 하였다. 재 액세스되지 않는 주 이유는 L1, L2 캐시의 여과 효과 때문에 LLC에 있는 캐시 라인들은 높은 시간 지역성을 갖지 못한다. 즉, 어떤 미스 라인이 LLC에 삽입된 후 그 라인의 참조는 L1, L2에서 적중되기 때문에 LLC에서는 액세스되지 않는다. 이것은 LRU 정책의 부정적 효과를 증폭시킨다[12].

LLC에 삽입 후 퇴출될 때까지 재사용되지 않는 캐시라인을 LLC에 저장하지 않는 zero-bypassing 방법이 제안되었다[6]. 그러나 이 방법은 LLC에 삽입과 퇴출 사이에 재 액세스되지 않는 라인(0-재사용 라인)이 부족할 경우, 특히 워킹 셋이 캐시 용량을 초과하는 경우 성능 향상에 실패할 수 있다.

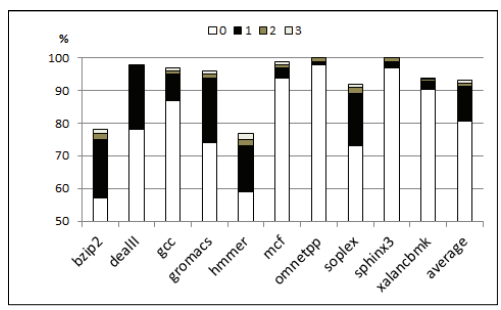

그림 2. LRU 교체정책을 갖는 LLC에서 SPEC 2006 벤치마크에 대한 0, 1, 2, 3 재사용 라인의 양
Figure 2. Fraction of 0, 1, 2 and 3 reused lines for SPEC 2006 benchmarks in LLC with LRU replacement policy.

<그림 2>는 LLC 성능에 종속적인 10개 벤치마크에 대해 퇴출되는 LLC 라인 중에서 3이하의 재사용 빈도를 갖는 라인의 점유율을 보여준다. 대부분의 퇴출된 라인은 결코 재사용되지 않거나 약간 재사용된다는 것을 볼 수 있다. <그림 2>에서 0-재사용 라인이 80.7%, 1-reuse 라인이 10.5%이다. 즉,

www.kci.go.kr

평균으로 0/1-재사용 라인이 전체의 90% 이상임을 보여준다. 만약 0/1-재사용 라인들을 여과시키고 LLC에는 워킹 셋 중 재사용 빈도가 높은 라인들만 저장한다면 캐시 오염 및 스래싱을 방지할 수 있다는 것을 암시한다.

3. 제안 방법

본 논문에서 제안한 재사용 빈도가 낮은 라인을 여과시키는 구조는 <그림 3>과 같다. 상위 레벨 캐시로부터의 요청을 서비스하기 위해 LLC와 BB를 동시에 탐색한다. LLC와 BB 모두에서 미스이면 요청은 메모리 제어기로 보내진다. 메모리로부터 미스라인을 반입하는 동안 RFP(Reuse Frequency Predictor)는 미스 라인에 대한 재사용 빈도를 예측하여 0/1-재사용 라인은 여과시키고 나머지 라인들은 LLC에 삽입한다.

3.1 아키텍처

제안 방법은 캐시라인의 재사용 빈도를 예측하는 재사용 빈도 예측기와 0/1-재사용 라인을 여과하는 바이패스 버퍼를 LLC에 추가하였다.

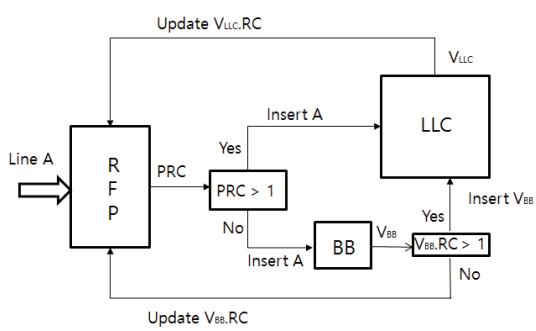

그림 3. 제안 방법의 구조

Figure 3. the architecture of proposed scheme

3.1.1 재사용 빈도 예측기

0/1-재사용 라인은 캐시에 들어가기 전에 여과시켜야한다. 이를 위해 어떤 미스 라인을 LCC에 삽입하기 전에 0/1-재사용 라인인지를 결정할 수 있는 구조가 필요하다. 결정은 캐시라인의 장래 재사용 빈도에 의존한다. 이는 프로그램의 반복적 메모리 동작은 규칙적이고 예측 가능한 다양한 캐시 참조 패턴을 나타내기 때문이다[11-14].

캐시라인에 대한 재사용 빈도의 예측은 수행동안 어떤 캐시라인이 미스일 때 그 라인의 재사용 빈도를 해당 라인의 마지막 재사용 빈도와 같다고 가정하여 예측한다.

<그림 4>는 SPEC2006 벤치마크에 대해 올바른 예측의 백분율을 보여준다. <그림 4>에서 MAddr는 직전 재사용 빈도를 예측할 때 RFP를 인덱스하기 위해 미스된 라인의 주소만을 사용한 경우이다. 어떤 벤치마크들은 90% 이상의 높은 예측력을 가지나 일부 벤치마크들은 낮은 예측력을 갖는다. 낮은 예측 정확도를 갖는 이유는 단지 라인 주소만을 인덱스로 사용하는 것은 여러 프로그램 페이스(phases)를 구별할 수 없기 때문이다. 즉 동일 메모리 주소가 여러 재사용 동작을 갖는 것을 구별 못하기 때문이다.

어떤 라인의 직전 재사용 빈도를 예측 및 저장하기 위해 미스를 발생시킨 명령어의 PC(Program Counter)와 미스된 라인 주소(MAddr)를 결합시키면 모든 벤치마크에서 예측 정확도가 개선된다. <그림 4>에서 PCMAddr는 미스를 발생시킨 명령어의 PC와 미스된 라인 주소를 결합하여 예측한 경우이다. 평균적으로 라인 주소와 PC를 결합하는 방법은 라인 주소만 사용한 방법보다 예측 정확도를 평균 67%에서 85%로 향상시킨다. 이 결과는 캐시라인의 재사용 빈도는 예측 가능하다는 것을 의미한다.

www.kci.go.kr

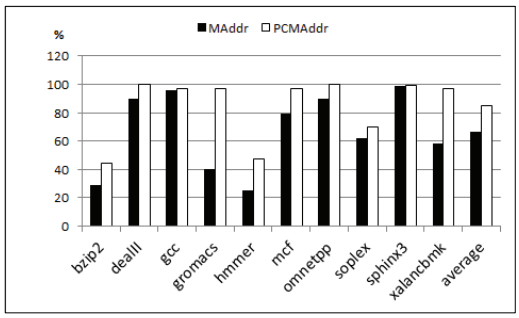

그림 4. 예측 정확도
Figure 4. The prediction accuracy

RFP는 캐시에 없는 라인들의 최근 재사용 빈도를 유지한다. 어떤 라인에 대해 미스가 발생하면 그것의 재사용 빈도는 LLC나 BB에서 직전 퇴출 때 예측기에 저장된 재사용 빈도에 기초하여 예측된다. LLC와 BB로부터 퇴출되는 라인에 대해 RFP에 엔트리를 할당하고 퇴출라인의 RC(reuse count)가 엔트리에 저장된다. 즉, 예측기의 각 엔트리는 재사용 빈도 값을 저장한다.

<그림 2>에서 보듯이 약 90% 캐시라인이 두 번 미만으로 재사용되므로 2-비트 엔트리로도 높은 커버리지를 가질 수 있고 스토리지 효율성을 위해 충분하다는 것을 보여준다.

예측기의 엔트리를 인덱스하기 위해 주소 일부와 PC 일부를 결합한 방법을 사용한다. 라인 주소의 하위 12비트는 시뮬레이션 환경에서 대부분의 캐시라인을 구별하는데 충분하다는 것을 알게 되었다. PC 일부는 프로그램 페이스(phases)를 구별한다. 동일 LLC 라인을 액세스하는 명령어 수는 일반적으로 적기 때문에 전체 주소만큼 많은 비트가 필요하지 않다. 본 연구에서는 4-비트 PC 부분을 구성하기 위해 미스된 load/store 명령의 PC[5:2]와 PC[9:6]을 XOR 시켰다

3.1.2 Bypass Buffer

0/1-재사용 라인들을 여과하고 빨리 축출하는 것에 추가하여 예측 메커니즘에서 잘못된 예측을 보상하기 위해 Bypassing Buffer(BB)를 사용할 수 있다. 예를 들어, 0-재사용 라인으로 예측되었으나 실제로 한번 재사용된 라인에 대해 그 라인을 바이패스하는 대신에 BB에 놓으면 캐시 적중을 제공할 기회를 제공한다. 대부분의 0/1-재사용 라인의 짧은 생명 주기(life span) 때문에 512 엔트리의 작은 BB는 여과된 라인들을 이용하는데 충분하다. 수많은 엔트리를 갖는 기존의 LLC와 비교하면 BB의 공간 오버헤드는 미세하다.

제안한 구조에서는 BB를 셋-연관으로 만든다. BB는 태그에 2개의 필드(PC_M, RC)가 추가된다. PC_M는 해당 라인에서 미스된 명령어의 4비트 XOR된 PC이고, RC는 라인이 삽입된 후 현재까지 재사용 빈도를 지시하는 2비트 카운터이다.

BB는 LRU 교체정책을 사용하며 임의 엔트리가 BB로부터 퇴출될 때 RC가 2이상이 되면 0/1-재사용 라인이 아닌 것으로 간주하여 LLC로 이동시킨다.

3.1.3 LLC 엔트리 수정

라인들이 LLC로 들어간 후 그들의 재사용 정보를 추적하기 위해 LLC 태그에도 두 필드(PC_M, RC)를 추가한다. 두 필드는 BB에서와 같은 기능을 갖는다.

3.2 동작

라인 A에 대한 요청을 서비스하기 위해 LLC와 BB가 동시에 탐색된다(그림 5). LLC와 BB에 있는 라인들은 상호 배타적이므로 둘 중에 하나에서 적중하던지 둘 모두에서 미스된다.

www.kci.go.kr

```
Algorithm Cache_Management
begin
  if line A is hit on LLC // LLC에서 적중
    increase RC_A in LLC;
  else if line A is hit on BB // BB에서 적중
    increase RC_A in BB;
  else // 미스
    index RFP using PC+A.addr
      and get PRC of line A;
    if PRC > 1 // non 0/1-reused line
      insert A to LLC
        and set reuse_inform;
      update RC_of_V_LLC in RFP;
    else // 0/1-reused line
      insert A to BB
        and set reuse_inform;
      if RC_of_V_BB > 1
        insert V_BB to LLC
          and set reuse_inform;
        update RC_of_V_LLC in RFP;
      else update RC_of_V_BB in RFP;
end.
```

그림 5. 캐시 동작 알고리즘

Figure 5. Algorithm for cache operation

3.2.1. 적중일 경우

요청한 라인 A가 LLC에서 적중이면 해당 액세스는 라인 A와 관련된 재사용 카운터(RC_A)를 갱신하는 것을 제외하고 기존의 캐시에서와 동일하게 처리된다. BB에서 적중이면 마찬가지로 재사용 카운터(RC_A)를 증가시킨다.

3.2.2. 미스일 경우

요청한 라인 A가 LLC와 BB에서 모두 미스이면 메모리 제어기로 요청을 보낸다. 데이터의 반입동안 캐시미스를 처리하는 동작이 진행된다.

먼저, 미스가 발생한 명령어의 PC와 라인 A의 주소로부터 구성된 인덱스로 RFP를 검색하여 PRC(Predicted Reuse Count)를 얻는다. 그런 다음 라인 A의 PRC에 기초하여, 라인 A가 0/1-재사용 라인인지를 결정한다. 즉, PRC가 1보다 크면 라인 A를 0/1-재사용 라인이 아닌 것으로 간주하여 LLC에 삽입한다. 그렇지 않으면 라인 A를 0/1-재사용 라인으로 간주하고 BB에 삽입한다. 이때 LLC 엔트리(혹은 BB 엔트리)의 RC는 0으로 초기화한다.

라인 A가 LLC 혹은 BB에 삽입될 경우, LLC(혹은 BB)로부터 퇴출된 라인 V_{LLC}(혹은 V_{BB})을 처리한다. 즉, V_{LLC}(혹은 V_{BB})의 재사용 정보를 사용하여 RFP를 갱신한다. 다만 V_{BB}를 퇴출할 경우, V_{BB}의 재사용 빈도(RC)가 1보다 크면 V_{BB}는 자주 재사용되는 라인으로 간주하여 LLC로 삽입되며, 이때 LLC로부터 퇴출되는 라인은 앞에서와 같은 방법으로 처리하면 된다.

위 동작들은 메모리로부터 데이터 반입과 동시에 처리되므로 위 동작의 경과시간(latency)은 숨겨진다.

4. 실험환경

제안한 방법은 [15]에 기초한 실행-구동 시뮬레이터를 사용하여 평가한다. 128-엔트리 ROB와 8-스테이지를 갖는 4-way 비순차적 슈퍼스칼라 프로세서를 모델링하였다.

표 1. 시스템 구성

Table 1. System Configuration

CPU	out-of-order, 8 wide fetch/decode/commit
L1 Inst/Data	32KB, 64B 라인, 4-way/8-way, LRU, 1 cycle latency
L2	256KB, 64B 라인, 8-way, LRU, 8 cycle latency
L3	2MB, 64B 라인, 16-way, LRU, 20 cycle latency
BB	512-엔트리, 8-way, 64B, LRU, 4-cycle hit
RFP	64K-엔트리 2비트 카운터, direct-mapped, 1-cycle hit

www.kci.go.kr

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

제안한 방법을 평가하기 위해 SPEC CPU2006 벤치마크를 사용한다. 본 방법은 효율성과 성능을 개선시키는 구조이기 때문에 LLC의 성능이 실행시간에 크게 영향을 주는 벤치마크에서 영향이 뚜렷하게 나타난다. 따라서 23개 SPEC CPU2006 벤치마크 중에서 LLC 크기를 1MB에서 4MB로 증가할 때 최소 10% 이상의 성능이 개선되는, LLC 성능에 종속적인 10개 벤치마크에 대해 평가한다. 이들 벤치마크는 콜드 스타트(cold start) 미스의 영향을 제거하기 위해 준비시간(warm-up)을 갖고 reference 입력을 사용하여 SimPoint[16]로 부터 각 벤치마크에 대해 250M명령어 트레이스를 얻었다.

5. 실험결과

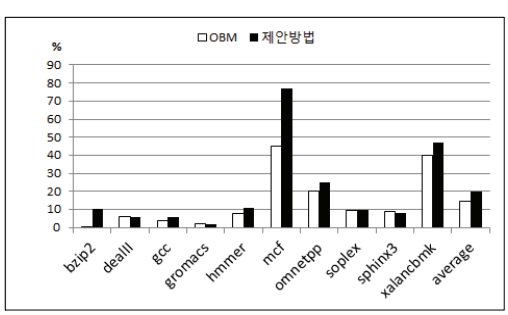

그림 6. 벤치마크에서 성능 향상
Figure 6. Performance improvement in benchmarks

<그림 6>은 제안한 방법과 OBM(Optimal Bypass Monitor)[8]에 대해 IPC 성능 개선을 비교하였다. 성능 개선은 LRU 교체정책을 사용하는 4MB LLC를 갖는 베이스라인 캐시 구성에 대한 상대적 값으로 계산되었다. 제안 방법은 LLC에 성능 종속적인 벤치마크에 대해 평균 20.1%, 최대 77%(mcf) 성능 향상을 얻었다. 제안 방법은 대부분의 벤치마크에서 성능 향상을 보였다. 평균 성능 향상은 제안 방법이 20.1%, OBM이 14.4%으로 제안 방법이 OBM보다 우수한 것으로 나타났다.

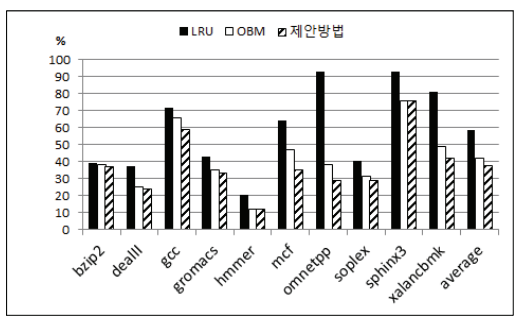

그림 7. LRU, OBM, 제안방법의 미스율
Figure 7. Miss rates of LRU, OBM and proposed scheme

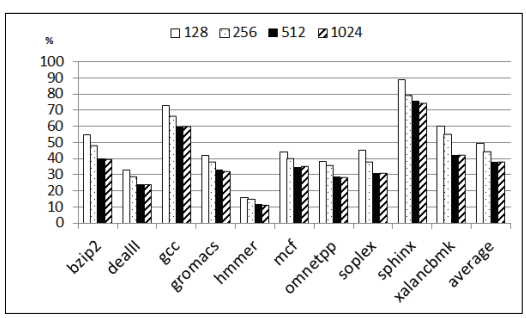

그림 8. BB 크기 변화가 미스 율에 미치는 영향
Figure 8. Effect of BB size change on miss rate

<그림 7>은 OBM과 제안 방법이 캐시에 유용한 라인들을 유지하고 유용하지 않은 캐시 라인들을 여과함에 의해 LRU에 비해 미스 율이 각각 16.5%, 20.6% 감소하였음을 보여준다. 미스 율도 제안 방법이 OBM보다 우수한 것으로 나타났다.

BB 크기에 대한 효과를 평가하기 위해 LLC 캐

www.kci.go.kr

시 크기를 2M로 유지하고, BB의 엔트리 수를 128, 256, 512, 1024로 변화시키면서 미스 율을 측정하였다. <그림 8>에서 엔트리 수가 128에서 256과 512로 증가 시에는 미스 율이 급격히 감소하나 512에서 1024로 증가 시킬 때 미스 율의 감소가 미약하였다. 이는 512 엔트리로도 충분함을 의미한다.

6. 결 론

LRU는 메모리 비집중적인 워크로드에서는 잘 수행되지만 워킹 셋이 캐시보다 큰 메모리 집중적인 워크로드에서는 빈약하게 수행한다.

본 논문에서 워킹 셋을 감소시키는 것을 목표로 0-재사용 라인뿐 아니라 1-재사용 라인들도 LLC에 들어가는 것을 제한하여 워킹 셋이 캐시 크기에 적합하도록 기회를 증가시키므로 서 캐시 미스 율을 감소시키고 스래싱도 방지할 수 있는 방법을 제안하였다.

실행 구동 시뮬레이션에 기초한 실험 결과는 2MB 16-way LLC 캐시에 512-엔트리 BB와 RFP를 추가했을 때 4MB 16-way LLC에 비해 SPEC2006 벤치마크에 대해 평균 미스 율을 20.1% 감소시키고, 평균 IPC를 20.6% 개선시켰다.

References

[1] X. Chen, S. Wu, L. Chang, W. Huang, and W. Hwo, *Adaptive cache bypass and insertion for many-core accelerators*, In MES-2014, pp. 1-12, 2014.

[2] N. Duong, D. Zhao, T. Kim, R. Cammarota, M. Valelo, and A. Veidenbaum, *Improving cache management policies using dynamic reuse distances*, In MICRO-2012, pp. 389-400, 2012.

[3] S. Lee, and Y. Cho, *A cache replacement policy for improving the performance of last level cache in processors*, JKITS, Vol. 10, No. 2, pp. 145-152, Apr. 2015.

[4] S. Lee, and Y. Cho, *Bypassing scheme for inclusive last level caches,* JKITS, Vol. 11, No. 2, pp. 155-162, Apr. 2016.

[5] A. Jaleel, K. Theobald, S. Steely, and J. Emer, *High performance cache replacement using re-reference interval prediction (RRIP)*, In ISCA-2010, pp. 60-71, 2010.

[6] M. Qureshi, A. Jaleel, Y. Patt, S. Steely, and J. Emer, *Adaptive insertion policies for high performance caching,* In ISCA-2007, pp. 381-391, 2007.

[7] S. Gupta, H. Gao, and H. Zhou, *Adaptive cache bypassing for inclusive last level caches*, In IPDPS-2013, pp. 1243-1253, 2013.

[8] L. Li, D. Tong, and X. Cheng, *Optimal bypass monitor for high performance last-level caches*, In PACT-2013, pp. 315-324, 2013.

[9] M. Chaudhuri, J. Nuzman, I*ntroducing hierarchy-awareness in replacement and bypass algorithms for last-level caches*, In PACT-2013, pp. 293-304, 2013.

[10] L. Belady, *A study of replacement algorithms for a virtual-storage computer*, IBM Systems Journal, pp. 78-101, 1966

[11] M. Kharbutli, and Y. Solihin, *Counter-based cache replacement and bypassing algorithms*, IEEE Trans. on Computers, Vol. 57, No. 4, pp. 433-447, 2008.

[12] N. Beckmann, and D. Sanchez, *Modeling cache performance beyond LRU*, HPCA-2016, pp. 581-593, 2016.

[13] H. Dybdahl, and P. Stenstrom, *Enhancing last-level cache performance by block bypassing and early miss determination*, In

www.kci.go.kr

ACSAC-2006, pp. 52-66, 2006.

[14] C. Li, S. Song, H. Dai, A. Sidelnik, S. Hari, and H. Zhou, *Locality-driven dynamic GPU cache bypassing*, In ICS-2015, pp. 67-77, 2015.

[15] V. Krishnan, and J. Torrellas, *A direct-execution framework for fast and accurate simulation of superscalar processors*, In PACT 1998, pp. 286-293, 1998.

[16] SimPoint home page : http://cseweb.ucsd.edu/calder/simpoint/, Oct. 2015.

마지막 수준 캐시의 성능을 향상시키는 여과 기법

조영일

수원대학교 컴퓨터학과

요 약

마지막 수준 캐시(LLC)는 일반적으로 LRU 정책을 사용하여 관리된다. 그러나, LRU는 캐시 라인이 액세스 될 때마다 캐시 라인을 가장 최근에 사용 된 위치로 이동시키는 높은 오버헤드를 갖는다. 또한 LRU는 캐시 크기보다 큰 단일 사용 메모리 액세스 시퀀스가 메모리에서 반입될 때 캐시 오염을 일으키기 쉽다. 이러한 재사용 라인의 일부가 캐시에 더 오래 상주 할 수 있으면 캐시 성능과 효율성이 향상 될 수 있다. 이전의 방식들은 결코 재사용되지 않는 라인(0-재사용 라인)을 바이패스하여 해결한다. 그러나 때때로 그들 방식은 재사용되지 않는 라인의 부족으로 이득을 제공하지 못한다. 본 논문에서는 0-재사용 라인뿐만 아니라 1-재사용 라인을 여과하며 그들을 정확하게 예측하는 새로운 메커니즘을 제안한다. 0/1-재사용 라인을 여과하는 것은 워킹 세트를 캐시 크기에 맞추는 더 많은 기회를 제공한다. 제안 기법은 효율성 및 성능 향상 기능이 입증된 시뮬레이션 환경을 사용하여 평가된다. LRU, OBM (Optimal Bypass Monitor), 제안 기법을 SPEC CPU2006 벤치마크에 대해 미스 율과 IPC(Instruction Per Cycle) 비교를 보여주는 실험 결과를 제시한다. 결과는 제안 기법과 OBM이 각각 평균 20.1%와 14.4%의 IPC를 향상시킬 수 있음을 보여준다. 그리고 제안 기법은 LRU에 비해 미스 율을 20.6% 감소시킨다.

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

www.kci.go.kr