Journal of Knowledge Information Technology and Systems(JKITS), Vol. 15, No. 2, pp. 265-273, April 2020
https://doi.org/10.34163/jkits.2020.15.2.012

Journal of Knowledge Information Technology and Systems

ISSN 1975-7700

http://www.kkits.or.kr

An Effective Last-Level Cache Replacement and Bypass Policy Using Hit Count

Young-Il Cho*

Division of Computer Science, University of Suwon

ABSTRACT

Caching techniques are fundamental for bridging the performance gap between components in a computer system, such as the performance of processor and memory. To get more performance out of the cache hierarchy, processors will rely on effective cache replacement policies. Variability in generational behavior of cache blocks is a key challenge for cache replacement policies that aim to identify less reused blocks as early and accurately as possible to maximize the cache efficiency. Existing processors use variants of LRU policy to determine replacement targets. But, there is a big gap between what LRU policy provides and what the optimal replacement policy provides by Belady's MIN. In this paper, we observed that the number of hits of a block is similar to the average number of hits when previously residing in the cache. This observation is used to improve the efficiency of the LLC through a low cost but effective replacement policy. With little overhead, the proposed policy is an effective replacement policy that associates a predicted hit count to a block and evicts the block to have fewer remaining hits in the future. Also, blocks that are predicted as non-reusable blocks are bypassed to improve the cache efficiency. The proposal provides a 5.8% miss reduction and a 4.8% performance improvement over baseline LRU.

KEYWORDS: Cache replacement policy, LRU, Belady's Min, Hit count, LLC

ARTICLE INFO: Received 3 March 2020, Revised 13 March 2020, Accepted 10 April 2020.

*Corresponding author is with Division of Computer Science, University of Suwon, 17 Wauangil Bongdam-Eup Hwaseong-Si, Gyeonggi-Do, 18323, KOREA.

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

1. 서 론

프로세서 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) 교체 정책은 프로세서 성능에 중요하기 때문에 지속적인 연구의 중심에 있다.

일반적으로 캐시 용량은 응용 프로그램의 작업 셋(working set)보다 훨씬 작으므로 캐시 블록 교체를 피할 수 없다. 가장 효율적인 교체 정책은 장래에 가장 오랫동안 필요하지 않은 데이터 블록을 캐시로부터 퇴출하고 가까운 장래에 사용될 블록들로 캐시를 유지하는 것이다.

기존 프로세서들은 LRU(Least Recently Used) 교체 정책의 변형을 사용하여 희생자를 결정하였다[1-10]. LRU 교체 정책은 메모리 집중적이지 않은 응용 프로그램에서는 잘 수행되지만 작업 셋이 캐시 용량보다 큰 메모리 집중적인 응용 프로그램에서는 빈약하게 수행된다. 또한 상위 레벨 캐시(L1, L2)가 대부분의 지역성을 걸러 내기 때문에 상위 레벨 캐시에 사용하는 전통적인 교체 정책들은 LLC에서 빈약하게 수행된다[5-8]. 예를 들어, 응용 프로그램의 작업 셋이 캐시 용량보다 크다면 LRU 정책은 재사용되지 않는 블록들이 캐시에 삽입되고 유용한 캐시 블록들을 퇴출시킨다면 불필요한 블록들로 캐시 공간을 차지하기 때문에 스래싱(thrashing) 및 성능 절벽을 유발한다.

이전 연구들은 LRU를 개선하거나 최적의 교체를 시도하는 여러 방법들을 제안했다. 이들 방법은 크게 세 가지 광범위한 기술을 사용한다는 것을 관찰했다. 첫째, 최근성(recency)은 오래 전에 사용된 라인보다 최근에 사용된 라인을 우선한다. LRU 정책에서 최근성만의 사용은 스래싱을 유발한다. 따라서 대부분의 고성능 정책은 최근성을 다른 기술과 결합한다[1-6]. 둘째, 보호(protection)는 작업 셋의 일부를 퇴출로부터 보호한다. 작업 셋의 크기가 캐시 용량을 초과하면 스래싱이 발생할 수 있다. 작업 셋의 일부를 퇴출로부터 보호하여 스래싱을 방지한다[7,8]. 셋째, 분류(classification)는 액세스들을 여러 부류로 나누고, 각 부류의 블록을 다르게 처리한다. 즉, 블록들을 재사용 또는 바이패싱으로 분류한다. 재사용되는 블록만 캐시에 저장하고 바이패싱 블록은 캐시에 있는 동안 재사용되지 않으므로 캐시를 바이패스 시킨다. 분류는 블록이 현저하게 다른 액세스 패턴을 가질 때 잘 작동한다[9,10].

Belady's MIN[11]은 미래에 대한 지식을 가지고 최적의 교체를 제공한다. 그러나 미래 참조에 대한 지식을 필요로 하기 때문에 실제 구현하는 것은 현실적이지 않다.

Hawkeye[12]는 과거 액세스에 대해 Belady' MIN을 재구성하고 향후 액세스의 캐싱 동작을 예측하기 위해 최적 솔루션을 학습한다. 과거 액세스에 대한 최적의 솔루션을 계산하기 위해 메모리 액세스 명령이 캐시 친화적 블록을 액세스하는지 혹은 캐시 반-친화적 블록을 액세스하는 경향이 있는지를 학습하는 프로그램카운터 기반 예측기를 사용한다. 캐시 친화적인 것으로 예상된 블록은 캐시에 높은 우선순위로 삽입시키고, 캐시 반-친화적으로 예상되는 블록은 낮은 우선순위로 삽입시킨다.

SHiP(Signature-based Hit-Predictor)[4]는 블록의 시그네처(미스의 원인인 프로그램 카운터)를 기반으로 블록의 재사용 패턴을 학습하여 재사용의 가능성을 예측한다. SHiP++[5]는 SHiP를 다음과 같이 몇 가지 개선시켰다. 캐시 삽입 및 캐시 적중 시에

재사용 예측을 개선시키고, 캐시 적중 시 SHCT(Signature History Counter Table)에 대한 학습을 개선시키고, 마지막으로 모든 캐시 액세스 유형에 대해 단일 재사용 예측을 사용하는 대신 캐시 액세스 유형을 분석하여 재사용 예측을 수행하도록 개선시켰다.

본 논문에서는 캐시 블록의 적중 수는 이전에 캐시에 거주했을 때의 평균 적중 수와 비슷하다는 관찰을 이용하여 캐시 블록의 적중 수를 예측하고 이를 실제 수행되는 동안 적중 수와 비교하여 그 차이가 가장 적은 블록을 교체 대상으로 선택한다. 즉 제안한 교체 정책은 예상 적중 수를 블록과 연관시키고 남은 적중 수가 더 적은 블록을 퇴출시키는 효율적인 교체 정책이다. 또한 재사용되지 않는 것으로 예측되는 블록은 바이패스 시켜 캐시 효율성을 향상시킨다. 제안 방법은 베이스라인 LRU 교체 정책에 비해 5.8%의 미스 감소와 4.8%의 성능 향상을 제공한다.

본 논문의 구성은 다음과 같다. 2장에서는 연구 동기, 3장에서는 제안방법, 4장에서는 실험환경, 5장에서는 성능평가, 6장에서는 결론을 맺는다.

2. 연구 동기

데드 블록 예측기(Dead Block Predictor)는 시기적절하고 정확한 방식으로 데드 블록을 식별하고 퇴출함으로써 유효 수명이 끝나지 않은 블록들을 캐시에 유지하여 더 많은 적중을 유도한다[13-18]. 응용 프로그램들에서 캐시 블록들은 재사용 패턴의 다양성을 보임에 따라 데드 블록 예측기의 임무는 복잡해졌다. 다양성의 원인은 모험적 수행(speculation)과 같은 마이크로 아키텍처 노이즈, 제어 흐름 다양성, 다른 스레드들의 캐시 압력 등 수없이 많다. 다양성은 한 캐시 수명에서 다음 캐시 수명까지 개별 캐시 블록들의 일관성 없는 동작으로 나타난다. 이러한 비일관성은 데드 블록 예측기가 블록의 유효 수명의 끝을 안정적으로 식별하는데 있어서 문제가 되고, 결과로 예측 정확도나 커버리지 또는 둘 모두를 낮추게 한다. 교체 결정을 내리기 위해 데드 블록 예측기를 사용하는 이전 방법들은 기본적으로 라이브 블록을 실수로 데드로 식별하면 캐시 미스가 불가피하다는 문제점과 셋(set)의 모든 블록이 라이브 블록으로 예상될 때마다 교체 정책은 가장 적합한 희생자를 효과적으로 선택할 수 없다는 문제점을 갖는다.

본 제안은 남은 적중 수가 0인 데드 블록에 의존하는 대신, 이전에 캐시에 거주했을 때 적중 수에 대한 정보를 활용하여 캐시 블록의 예상 적중 수를 추정하고 희생자 결정 과정에서 그 정보를 이용한다. 즉, 일반적으로 캐시 블록의 적중 수는 그 블록이 이전에 캐시에 거주했을 때 평균 적중 수와 유사한 경향을 가지므로 현 거주에서 남은 적중 수가 가장 적은 블록을 희생자로 선택한다.

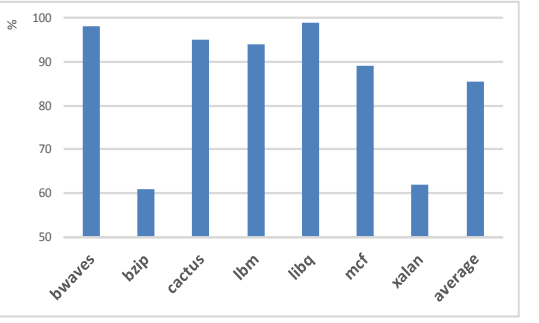

그림 1. 블록의 적중 수와 이전에 캐시 거주시의 평균 적중수의 차가 1 이하인 블록의 백분율

Figure 1. Percentage of blocks that the difference between the number of hits of a block and the average number of hits when previously residing in the cache is one or less

<그림 1>은 LLC에 Belady's MIN의 교체 알고리즘을 적용했을 때 실제 적중 수와 이전 캐시 거주할 때 평균 적중 수의 차가 1이하인 것이 평균

85%임을 보여준다. 차이가 0인 것이 약 68%이고 차가 1인 것이 약 17%이다. 이는 블록의 실제 적중 수는 이전에 캐시에 거주했을 때 평균 적중 수와 비슷하다는 것을 의미한다. 이와 같은 분석을 이용하여 본 논문에서는 캐시 블록의 적중 수를 예측하고 이를 실제 수행되는 동안의 적중 수와 비교하여 그 차이가 가장 적은 블록을 교체 대상으로 선택하고, 또한 예상 적중 수가 0인 블록은 캐시를 바이패스 시키는 LLC를 위한 효율적인 교체 및 바이패스 정책을 제안한다.

3. 제안 방법

본 논문에서는 캐시에 이전 거주할 때 블록의 평균 적중 수를 예측하기 위해 적중 카운트 예측기(Hit Count Predictor, HCP)를 제안한다. 제안 방법의 구조는 <그림 2>와 같다.

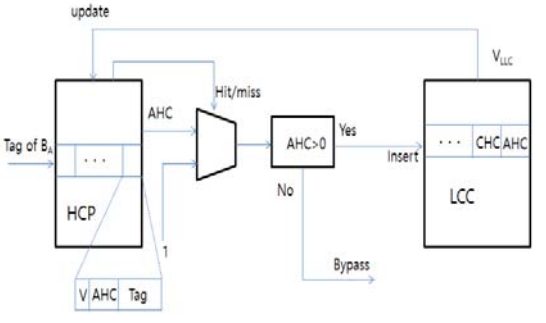

그림 2. 제안 방법의 구조
Figure 2. The architecture of proposed scheme

메모리 요청을 서비스하기 위해 LLC와 적중 카운트 예측기를 동시에 탐색한다. LLC에서 미스이면 요청은 메모리 제어기로 보내진다. 메모리로부터 미스 블록을 반입하는 동안 적중 카운트 예측기는 미스 블록에 대한 평균 적중 수를 예측하여 LLC로 보낸다. 예상 적중 수는 이전 LLC에 거주하는 동안 평균 적중 수를 나타내며 캐시 미스가 발생할 때 희생자를 선택할 때와 캐시에 있는 동안 재사용되지 않는 블록을 바이패스 하는데 이용한다.

3.1 아키텍처

응용프로그램은 제어흐름 및 데이터 액세스의 반복성을 갖기 때문에 적중 카운트 예측기는 캐시에서 이전 거주 때 블록의 적중 수를 기반으로 캐시 블록의 적중 수를 예상한다. 각 캐시 블록에 대해 이전 거주동안 발생한 누적 평균 적중 수를 저장하여 이 평균 적중 수를 예상 적중 수로 사용한다. 그러나 모든 캐시 블록의 적중 기록을 저장하려면 상당한 스토리지 오버헤드가 발생한다. 그런데 캐시 블록의 적중 수를 각 블록의 과거 히스토리로 사용하여 예측하는 경우와 블록 주소의 MSB(most significant bits)를 공유하는 블록들의 과거 히스토리를 사용하여 예측한 경우가 거의 비슷한 결과를 갖는다[3,4,8]. 따라서 각 캐시 블록 당 적중 기록을 저장하는 대신 캐시의 태그로 표시되는 각 영역(region)마다 적중 기록을 수집한다. LLC의 고유 태그 수는 고유 블록 수보다 훨씬 적기 때문에 적중 카운트 예측기의 저장 오버헤드를 줄일 수 있다.

제안 방식은 블록의 이전 거주 동안의 평균 적중 수를 저장하기 위해 적중 카운트 예측기를 사용한다. 적중 카운트 예측기는 총 128셋을 가지며 각 셋은 16-way 연관 구조를 갖는다.

<그림 2>에서 예측기의 구조와 셋에 있는 각 항목의 필드들을 보여준다. V 필드는 항목이 유효한지를 나타내는 단일 비트이다. Tag 필드는 예측기에 대한 인덱스(7비트)로 사용되지 않는 LLC에 있는 캐시 블록의 태그 비트 일부(20 비트)를 저장한다. AHC(Average Hit Count) 필드는 3비트 포화 카

운터로 이전 거주 동안 대응하는 캐시 블록의 누적 평균 적중 수를 저장한다. 적중 카운트 예측기의 각 셋은 LRU 교체 정책으로 관리된다.

제안 방법은 LLC의 각 블록에 3비트 CHC(Current Hit Count)와 3비트 AHC의 추가를 요구한다. CHC는 블록이 LLC에서 현 거주 동안 그 블록에서 발생한 적중 수를 저장하는 포화 카운터이고, AHC는 블록이 LLC에 할당될 때 적중 카운트 예측기에서 예측된 해당 블록에 대한 평균 적중 수를 저장한다.

3.2 동작

```
Algorithm Cache_Management
begin
  if block A is a hit on LLC
    increase CHC_A ; //블록 A의 CHC를 증가
  else // miss on LLC
    if block A is hit on HCP // HCP에서 적중
      if HCP_A.AHC > 0
        allocate entry LLC_A for block A into LLC;
        //HCP에서 블록A의 AHC를 LLC에 할당된
        //블록A의 AHC에 저장
        set LLC_A.AHC to HCP_A.AHC;
      else // HCP_A.AHC == 0
        bypass on LLC;
    else // miss on HCP
      allocate a entry HCP_A for block A into HCP;
      // HCP_A.AHC를 default value('1')로 설정
      set HCP_A.AHC to 1;
      allocate a entry LLC_A for block A into LLC;
      // HCP_A.AHC도 default value('1')로 설정
      set LLC_A.AHC to 1 ;
end.
```

그림 3. 캐시 동작 알고리즘
Figure 3. Algorithm for cache operation

블록 A에 대한 요청을 서비스하기 위해 LLC와 적중 카운트 예측기가 동시에 검색되고 <그림3>의 알고리즘으로 수행한다.

3.2.1. LLC에서 적중일 경우

요청한 블록 A가 LLC에서 적중이면 해당 액세스는 블록 A와 관련된 적중 카운트(CHCA)를 증가시키는 것을 제외하고 기존 캐시에서와 동일하게 처리된다.

3.2.2. LLC에서 미스일 경우

요청한 블록 A가 LLC에서 미스일 때 메모리로부터 들어오는 블록 A를 LLC에 할당하고, 적중 카운트 예측기에서 블록 A를 검색하여 적중이면 해당 항목의 AHC를 조사하여 양수이면 해당 항목의 AHC(그림3에서 HCP_A.AHC)를 블록 A를 위해 LLC에 할당된 항목의 AHC(그림3에서 LLC_A.AHC) 필드에 설정한다. 만약 해당 항목의 AHC가 0이면 이전 LLC에 거주할 때 재사용되지 않은 블록이므로 바이패스 시킨다. 적중 카운트 예측기에서 미스이면 LRU 정책에 따라 적중 카운트 예측기에 엔트리를 할당하고 AHC는 기본 값(default value)인 1로 설정하고 LLC에 할당된 항목의 AHC도 1로 설정한다. 기본 값을 1로 설정한 이유는 SPEC CPU2006 벤치마크의 절반 정도에서 LLC로 가져온 블록의 약 90%가 1번 이하의 재사용됨을 관찰하였기 때문이다[5-8].

3.3. 희생자 선택

요청한 블록 A가 LLC에서 미스일 때 메모리로부터 들어오는 블록 A를 LLC에 할당하기 위해 셋에서 희생자를 선택해야한다.

셋의 항목들에 대해, AHC 필드는 해당 블록에 대해 예측되는 누적 평균 적중 수를 나타낸다. 또한 CHC는 해당 블록이 LLC에 들어간 이후 그 블

록에서 발생한 적중 수를 나타낸다. 셋의 항목들에 대해 'AHC – CHC'는 해당 블록의 예상되는 남은 적중 수를 나타낸다. 셋의 항목들 중 'AHC – CHC'가 가장 작은 항목이 앞으로 기대되는 적중 수가 가장 작으므로 그 항목을 희생자 블록으로 선택한다. 'AHC – CHC'가 가장 적은 항목이 복수일 때는 임의로 한 항목을 희생자로 선택한다.

3.4. 적중 카운트 예측기의 갱신

임의 블록이 LLC에서 퇴출될 때 적중 카운트 예측기가 갱신되며 퇴출 블록(V_{LLC})의 태그로 적중 카운트 예측기를 검색한다. 검색 결과가 적중이면, 검색된 항목의 AHC를 퇴출 블록의 '(AHC + CHC) / 2' 값으로 갱신하여 AHC가 누적 평균 적중 수를 갖게 한다. 검색이 실패이면 적중 카운터 예측기의 해당 셋에서 LRU 교체 정책으로 선택된 퇴출 항목(즉, LRU 항목)을 새 항목에 할당하고, V 비트를 설정하고, 태그를 갱신하고, AHC를 LLC에서 퇴출 블록의 CHC로 설정한다.

4. 실험환경

제안한 방법의 성능을 평가하기 위해 SimFlex[18]를 사용하였다. 제안 방법은 128-엔트리 ROB와 8-스태이지를 갖는 4-way 비순차적 슈퍼스칼라 프로세서를 모델링하였다.

<표 1>은 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 블록 크기를 사용한다. 표에서 LRU는 기존의 LRU 교체 정책을 의미한다. 캐시에 이전 거주할 때 블록의 평균 적중 수를 예측하기 위한 적중 카운트 예측기(HCP)는 128 엔트리 16-way 셋 연관이다.

표 1. 시스템 구성
Table 1. System Configuration

CPU	out-of-order, 8 wide fetch/decode/commit
L1 Inst/Data	32KB, 64B 블록, 4-way/8-way, LRU, 1 cycle latency
L2	256KB, 64B 블록, 8-way, LRU, 8 cycle latency
L3	2MB, 64B 블록, 16-way, LRU, 20 cycle latency
HCP	128 sets 16-way set-associativity

제안한 방법을 평가하기 위해 SPEC CPU2006 벤치마크를 사용한다. 제안 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 7개 벤치마크에 대해 평가한다. 이들 벤치마크는 콜드 스타트(cold start) 미스의 영향을 제거하기 위해 준비시간(warm-up)을 갖고 reference 입력을 사용하여 SimPoint[19]로 부터 각 벤치마크에 대해 250M명령어 트레이스를 얻었다.

5. 실험결과

본 연구에서는 제안한 교체 정책의 효율성을 측정하기 위해 최신의 교체 정책인 SHiP++, Hawkeye와 제안 방법이 베이스라인 LRU 교체 정책에 비해 미스 감소율과 성능 향상율을 비교한다.

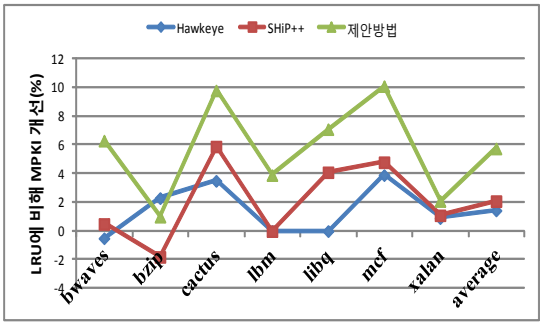

그림 4. 벤치마크에서 MPKI 개선
Figure 4. MPKI improvement in benchmarks

<그림 4>는 베이스라인 LRU 교체 정책에 비해 평가된 정책들의 MPKI(Miss Per Kilo Instructions) 감소율을 보여준다. 그림에서 보듯이 제안 방법은 다른 정책보다 우수한 것으로 나타났다. 제안 방법은 LRU에 비해 MPKI를 평균 5.8% 감소시켰다. 이에 비해 최신의 교체 정책인 Hawkeye는 LRU에 비해 MPKI를 평균 1.5%를 감소시켰고, SHiP++는 평균 2.1% 감소시켰다.

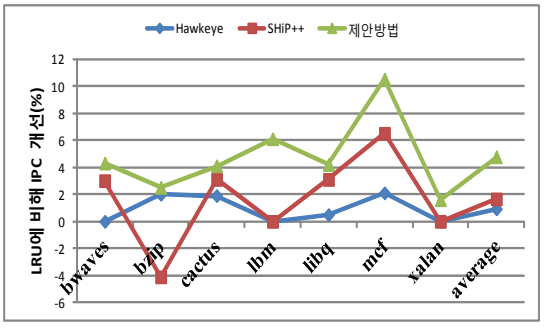

그림 5. 벤치마크에서 성능 개선
Figure 5. Performance improvement in benchmarks

<그림 5>는 베이스라인 LRU에 비해 평가된 정책들의 성능 향상을 비교하였다. 성능 측정 기준으로 IPC(Instruction Per Clock)를 사용한다. 본 제안의 성능은 mcf에서 IPC를 최대 10.5% 향상시켰고, 평균 4.8% 향상시켰다. 최신의 교체 정책인 Hawkeye, SHiP++는 IPC를 각각 평균 1%, 1.7% 향상시켰다.

6. 결 론

LRU 교체 정책은 메모리 비집중적인 워크로드에서는 잘 수행되지만 워킹 셋이 캐시 용량보다 큰 메모리 집중적인 워크로드에서는 빈약하게 수행한다.

본 논문에서는 캐시 블록의 적중 수는 그 블록이 이전에 캐시에 거주했을 때 평균 적중 수와 비슷하다는 관찰을 이용하여 현 거주에서 남은 적중 수가 가장 적은 블록을 희생자로 선택하고, 재사용되지 않는 것으로 예상되는 블록은 바이패스 시켜 LLC의 효율성을 향상시키는 방법을 제안하였다. 제안 방법은 블록의 이전 거주 동안의 평균 적중 수를 저장하기 위해 16-way 연관 구조를 갖는 128 셋의 적중 카운트 예측기를 사용하고, 블록이 LLC에 거주하는 동안 적중 수를 저장하기 위해 LLC의 각 블록에 3비트 CHC를 추가하였다.

실행 구동 시뮬레이션에 기초한 실험 결과, 제안한 교체 정책은 LRU 교체 정책에 비해 MPKI를 평균 5.8% 감소시켰고, 성능은 IPC를 평균 4.8% 향상시켰다.

References

[1] N. Beckmann, and D. Sanchez, *Modeling cache performance beyond LRU*, In HPCA-2010, pp. 225-236, 2016.

[2] V. G. Armin, M. S. Sara, L. N. Mohammad-Reza, B. Mohammad, L. K. Pejman, and S. A. Hamid, *Cache replacement policy based on expected hit count*, In ISCA-2017 CRC-2 Workshop,

2017.

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

[4] C-J. Wu, A. Jaleel, W. Hasenplaugh, M. Martonosi, S. C. Steely Jr, and J. Emer, *SHiP: Signature-based hit predictor for high performance caching*, In MICRO-2011, pp. 430-441, 2011.

[5] V. Young, C-C. Chou, A. Jaleel, and M. K. Qureshi, *SHiP + + : Enhancing signature-based hit predictor for improved cache performance*, In ISCA-2017 CRC-2 Workshop, 2017.

[6] G. Keramidas, P. Petoumenos, and S. Kaxiras, *Where replacement algorithms fail: A thorough analysis*, In Proc. of 7th ACM international conference on Computing frontiers, pp. 141-150, 2010.

[7] M. K. Qureshi, A. Jaleel, Y. Patt, S. Steely Jr., and J. Emer, *Adaptive insertion policies for high performance caching*, In ACM SIGARCH Computer Architecture News Vol. 35, No. 2, pp. 381-391, 2007.

[8] 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, 2015.

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

[10] S. M. Khan, Z. Wang, and D. A. Jiménez, *Decoupled dynamic cache segmentation*, In HPCA2012, pp. 25-29, 2012.

[11] L. A. Belady, *A study of replacement algorithms for a virtual-storage computer*, IBM Systems Journal, Vol. 5, No. 2, pp. 78−101, 1966.

[12] A. Jain, and C. Lin, *Back to the future: Leveraging Belady's algorithm for improved cache replacement*, In ISCA-2016, pp. 78-89, 2016.

[13] A. Jaleel, W. Hasenplaugh, M. Qureshi, J. Sebot, S. Steely, and J. Emer, *Adaptive insertion policies for managing shared caches*, In PACT-2008. pp. 208-219, 2008.

[14] S. M. Khan, D. A. Jimenez, D. Burger, and B. Falsafi, *Using dead blocks as a virtual victim cache*. In PACT-2010, pp. 489-500, 2010.

[15] A-C. Lai, and B. Falsafi, *Selective, accurate, and timely self-invalidation using last-touch prediction*, In ISCA-2000, pp. 139-148, 2000.

[16] A-C. Lai, C. Fide, and B. Falsafi, *Dead-block prediction & dead-block correlating prefetchers*, In ISCA-2001, pp. 144-154, 2001.

[17] H. Liu, M. Ferdman, J. Huh, and D. Burger, *Cache Bursts: A new approach for eliminating dead blocks and increasing cache efficiency*, In MICRO-2008, pp. 222-233, 2008.

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

[19] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe, *SimFlex: Statistical sampling of computer system simulation*, *IEEE Micro*, Vol. 26, No. 4, pp. 18-31, 2006.

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

Journal of Knowledge Information Technology and Systems(JKITS), Vol. 15, No. 2, pp. 265-273, April 2020

적중 수를 이용한 효율적인 마지막 수준 캐시 교체 및 바이패스 정책

조영일

수원대학교 컴퓨터학부 교수

요 약

캐싱 기술은 프로세서 및 메모리 성능과 같은 컴퓨터 시스템의 구성 요소 간 성능 격차를 해소하는데 필수적이다. 캐시 계층에서 더 많은 성능을 얻기 위해 프로세서는 효율적인 캐시 교체 정책에 의존한다. 캐시 블록의 세대별 동작 변화는 캐시 효율성을 최대화하기 위해 재사용 빈도가 낮은 블록을 가능한 신속하고 정확하게 식별하는 것을 목표로 하는 캐시 교체 정책의 핵심 과제이다. 기존 프로세서는 LRU 정책의 변형을 사용하여 교체 대상을 결정한다. 그러나 LRU 정책이 제공하는 것과 최적의 교체 정책인 Belady's MIN이 제공하는 것 사이에는 큰 차이가 있다. 본 논문에서는 캐시 블록의 적중 수는 이전에 캐시에 거주했을 때 평균 적중 수와 비슷하다는 것을 관찰했다. 이 관찰은 저렴하지만 효율적인 교체 정책을 통해 LLC의 효율성을 향상시키는 데 사용된다. 제안된 정책은 예상 적중 횟수를 블록에 연관시키고 장래에 더 적은 적중 횟수를 갖는 블록을 퇴출하는 적은 오버헤드를 갖는 효율적인 교체 정책이다. 또한 재사용되지 않는 블록으로 예측되는 블록은 바이패스 시켜 캐시 효율성을 향상시킨다. 제안 방법은 베이스라인 LRU에 비해 5.8%의 미스 감소와 4.8%의 성능 향상을 제공한다.

Young-Il Cho received the B.S., M.S. and Ph.D. in electronic engineering from the Hanyang University in 1980, 1982 and 1985 respectively. He has been a professor in Division 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