Journal of The Korea Society of Computer and Information
Vol. 22 No. 3, pp. 9-16, March 2017

www.ksci.re.kr
https://doi.org/10.9708/jksci.2017.22.03.009

Reuse Information based Thrashing Resistant Cache Management Scheme

Gyu Yeon Sim*, Cheol Hong Kim**

Abstract

In recent computing systems, LRU replacement policy has been widely used because it can be simply implemented and applicable to most programs. However, if the working set size of the program is bigger than the actual cache size, LRU replacement policy may occur thrashing problem. Thrashing problem means that cache blocks are consistently replaced without re-referencing in the cache. This paper proposes a new cache management scheme to solve the thrashing problem in the second-level cache. The proposed scheme measures per set reuse frequency using EAF structure to find thrashing sets. When the cache miss occurs, it tests whether the address of the missed block is stored or not. If the address of the missed block is stored, it means that the recently evicted block is re-requested, so the reuse frequency is predicted high. In this case, the corresponding counter of the set is increased. When the counter value is bigger than the threshold value, we assume that the corresponding set shows high reuse frequency. The proposed scheme assigns the set with high reuse frequency to the additional small size cache to keep the blocks in the cache for a long time. Our experimental results show that the proposed scheme improves the IPC by 3.81% on average.

▸ Keyword : Thrashing, L2-Cache, Bloom Filter, Cache Replacement Policy

I. Introduction

최신 컴퓨팅 환경에서 캐쉬 메모리는 시스템 성능에 매우 큰 영향을 미친다. 캐쉬 메모리의 구조를 결정하는 설계 요소는 크기, 연관 방식, 교체 정책 등이 있는데, 그 중에서도 교체 정책은 캐쉬에 어떤 블록들을 더 오래 보관할지를 결정하는 방법으로써, 캐쉬를 효율적으로 사용하기 위해서는 효과적인 교체 정책이 필수적으로 요구된다. 교체 정책은 크게 두 가지 방식으로 구성된다. 첫 번째는 블록을 캐쉬에 삽입할 때 어떤 위치로 저장할 것인지를 결정하는 삽입 방식과 두번째는 새로운 블록이 삽일될 때 어떠한 블록을 교체시킬 것인가를 결정하는 교체 방식이다.

임베디드 프로세서에서는 구현이 간단하고 대부분의 프로그램에 적용 가능한 LRU(Least Recently Used) 교체정책이 널리 사용되었다. LRU 교체정책은 세트에 있는 여러 블록들 중에서 가장 오랫동안 접근되지 않은 블록을 교체시키는 정책이다. LRU는 시간적 지역성이 높아서 최근에 사용되었던 블록들이 다시 사용되는 빈도가 높은 경우에 매우 효과적이다. 하지만 프로그램이 데이터를 다시 사용하기까지 시간이 오래 걸리거나 프로그램에서 데이터를 순환적으로 사용하는데 그 범위가 캐쉬의 크기보다 큰 경우에는 LRU가 성능에 좋지 않은 영향을 미치게 된다. LRU 교체기법이 캐쉬의 성능을 저하시키는 경우들 중에서 가장 문제가 되

• First Author: Gyu Yeon Sim, Corresponding Author: Cheol Hong Kim
*Gyu Yeon Sim(sim.gyuyeon@gmail.com), School of Electronics and Computer Engineering, Chonnam National University
**Cheol Hong Kim(chkim22@chonnam.ac.kr), School of Electronics and Computer Engineering, Chonnam National University
• Received: 2016. 11. 09, Revised: 2016. 12. 06, Accepted: 2017. 03. 12.
• This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2015R1D1A3A01019454) and also supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the ITRC(Information Technology Research Center) support program (IITP-2016-R2718-16-0011) supervised by the IITP(Institute for information & communications Technology Promotion).

는 경우는 프로그램의 데이터 접근 범위인 작업 그룹의 크기가 캐쉬의 크기보다 크게 되어 재사용되는 데이터들이 MRU 위치에서 LRU 위치까지 이동한 후 지속적으로 교체되기만 할 뿐 적중이 발생하지 않아 캐쉬의 활용률이 크게 떨어지게 되는 경우인데, 이러한 경우를 캐쉬 쓰래싱(Thrashing)이라고 한다.

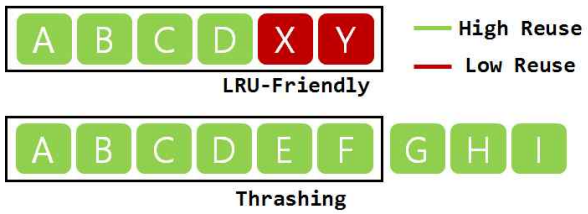


Fig. 1. Cache Thrashing

그림 1은 LRU 교체 정책을 사용했을 때 효과적인 경우와 쓰래싱이 발생하여 LRU가 캐쉬의 성능을 저하시키는 경우를 나타낸다. 일반적으로 캐쉬에서 LRU가 효율적으로 사용되는 경우는 재사용 빈도가 높은 블록의 수가 캐쉬의 크기보다 적어서 A, B, C, D에 대한 요청이 반복적으로 발생하는 경우 재사용 빈도가 높은 블록들이 캐쉬에 지속적으로 보관되어 대부분의 요청에 적중이 발생하는 경우이다. 이에 반해, 쓰래싱이 발생하는 경우에는 A부터 I까지의 블록들이 반복적으로 요청되면서 모든 블록들의 재사용 빈도가 높지만 MRU 위치에서 LRU 위치까지 이동한 후 차례대로 교체되기만 할 뿐 캐쉬에서 적중은 전혀 발생시키지 못하게 됨으로써 캐쉬 활용률을 심각하게 저하시킨다.

캐쉬 쓰래싱 문제를 해결하기 위해서 교체정책에 관한 많은 연구가 수행되었다. 쓰래싱이 발생하게 되면 캐쉬에 대한 접근이 대부분 적중 실패하게 되어 캐쉬의 효율성이 크게 떨어지게 되는데, 이런 경우에는 LRU 교체 정책처럼 캐쉬에 최근 요청된 데이터를 가장 오래 보관하는 것이 아니라, 데이터 중 일부는 오래 보관하고 나머지들은 보관기간을 짧게 해주면 오래 보관되는 데이터들 중에서는 적중이 발생할 수 있어서 캐쉬의 효율성을 향상시킬 수 있다. 이를 위해서 기존 연구들에서는 주소값을 바탕으로 블록들의 재사용 특성을 예측하거나 각 블록 별로 재사용 특성을 예측하여 재사용 빈도가 높게 예측되는 블록들은 캐쉬에 오랫동안 보관될 수 있도록 하고 나머지 블록들은 캐쉬에서 보관하지 않도록 하거나 잠시 동안만 보관되도록 함으로써 캐쉬의 효율성을 향상시켰다[1, 2].

이처럼 지금까지의 연구들은 대부분 캐쉬에 블록을 삽입하는 방식을 바꾸어 캐쉬에 일부의 블록만을 남김으로써 쓰래싱 문제를 해결하려고 하였다. 하지만 이들은 LRU 교체 정책에 적합한 프로그램들에 대해서는 성능저하를 발생시킬 가능성이 있어 문제가 된다.

또한 이전의 연구들에서 캐쉬에 대한 접근이 전체 세트에 대해 균등하게 발생하지 않는다는 사실을 확인할 수 있다[3, 4-7]. 이는 대부분의 프로그램에서 빈번히 발생하는데 캐쉬에 대한 접근이 각 세트에 대해 균등하게 분포되지 않아 세트별로 접근되는 수가 매우 상이하게 나타나는 현상이다. 이 경우 전체 작업 그룹의 크기는 캐쉬의 크기보다 작더라도 특정 세트에만 접근하게 되어 캐쉬에 공간적 여유가 있더라도 해당 세트에서는 지속적으로 쓰래싱이 발생하게 된다. 그림 2는 이러한 현상이 매우 두드러지게 나타나는 ammp 벤치마크의 세트별 충돌 적중 실패의 (conflict miss) 수를 나타낸다. 가로축이 세트의 번호를 나타내고 세로축은 적중 실패 수를 나타내는데, 세트에 따라 그 수가 매우 상이한 것을 확인할 수 있다.

본 논문에서는 이에 착안하여 2차 레벨 캐쉬의 쓰래싱을 완화시키기 위해 작은 크기의 캐쉬를 2차 레벨에 추가하고 쓰래싱이 많이 발생하는 세트들을 탐지하여 이 세트들을 추가시킨 캐쉬에 할당함으로써 쓰래싱이 많이 발생하고 있는 세트들에서 재사용 가능성이 있는 블록들이 2차 레벨 캐쉬에 오래 머무르게 하는 기법을 제안하고자 한다.

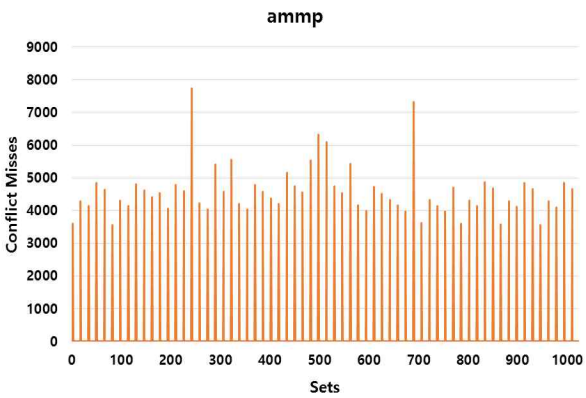


Fig. 2. Number of conflict misses (Per-Set)

본 논문의 구성은 다음과 같다. 2장에서는 쓰레싱 문제를 해결하기 위해 제안된 기존 연구들을 소개한다. 3장에서는 본 논문에서 제안하는 세트별 재사용 빈도 정보 기반 쓰래싱 저항 캐쉬 관리 기법을 기술하고, 4장에서는 모의실험결과를 분석하고 성능을 평가한다. 마지막으로 5장에서는 본 논문의 결론과 향후 연구계획에 대해서 기술한다.

II. Related Work

캐쉬 쓰래싱 문제를 해결하기 위한 교체 정책들에 대해 많은 연구가 이루어졌다. 기존 연구들은 주로 캐쉬 블록들의 특성을 파악해 분류한 뒤 분류된 그룹별로 재사용 특성을 예측하여 그 특성에 맞게 블록을 삽입하는 방식을 바꾸는 방법으로 제안되었다.

DIP(Dynamic Insertion Policy)[8]는 쓰래싱 문제를 해결

하기 위해 작업 그룹 중에 일부만 캐쉬에 보관될 수 있도록 블록 삽입 방식을 바꾸어 보관된 부분에서라도 적중을 발생시킬 수 있는 기법을 제안하였다.

기존의 LRU는 작업 그룹이 캐쉬 크기보다 크게 되면 블록이 MRU 위치에서 LRU 위치까지 이동하면서 한 번도 적중되지 못하고 교체된 후 다시 요청되기를 반복하여 캐쉬의 활용률이 크게 떨어지는데 이것을 방지하기 위하여 블록이 삽입될 때의 위치를 LRU 위치로 바꾸어 대부분의 블록들은 LRU위치에서 곧바로 캐쉬에서 교체되도록 하고 블록이 재 참조되는 경우에만 MRU 위치로 이동시켜 일부의 블록들만 캐쉬에 보관시킬 수 있는 LIP(LRU Insertion Policy)[8] 또한 제안되었다. 이 기법에서 캐쉬 블록들은 LRU 위치로 삽입되고 재 참조될 때 MRU 위치로 이동하게 된다. 따라서 새로 삽입된 블록이 재참조 되어 MRU 위치로 이동하지 않는 한 한번 MRU 위치로 이동한 블록들은 그대로 계속 유지되게 되므로 미래에 참조될 가능성이 없는 블록들도 그대로 MRU 위치에서 계속 유지되는 문제가 발생한다.

이를 막기 위해 BIP(Bimodal Insertion Policy)[8]를 제안하여 모든 블록들을 LRU 위치에 삽입하지 않고 일정 비율의 블록은 MRU 위치에 삽입하도록 하여 MRU 위치의 블록들이 계속 유지되지 않도록 할 수 있는 방법도 제안되었다.

LIP와 DIP는 LRU 교체정책이 적합한 프로그램에서는 성능을 크게 저하시킬 수 있다. 이를 방지하기 위하여 세트 샘플링 기법을 사용하여 선택된 몇 개의 세트를 샘플링 하여 현재 프로그램이 LRU 교체정책에 잘 맞는지 BIP 교체정책에 잘 맞는지를 측정하고, 이를 바탕으로 캐쉬의 교체정책을 결정한다. 하지만 이 경우 세트별로 적중 실패의 비율이 다르게 되면 전체 시스템의 성능이 감소할 가능성이 있다.

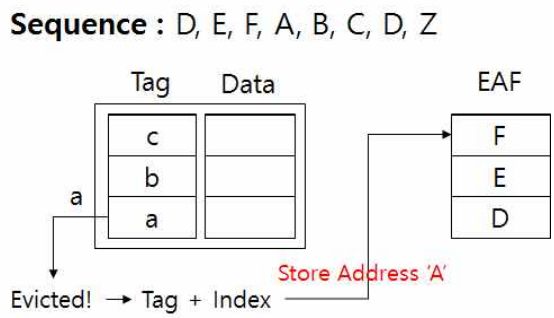


Fig. 3. Evicted Address Filter

EAF(Evicted-Address Filter)[9]는 캐쉬 블록 단위로 재사용 빈도를 예측하여 블록의 삽입 위치를 결정한다. 기존의 연구들은 PC(Program Counter)값을 바탕으로 블록들의 그룹을 나누어 각기 다른 삽입 방식을 선택하였다. 이 경우에는 각각의 블록들이 속해 있는 그룹의 특성과 다른 특성을 가지게 되는 경우 캐쉬 성능이 크게 저하될 수 있다. 이를 해결하기 위해서는 모든 블록들의 재사용 빈도를 예측하여야 하는데 이를 위해 시스템의 모든 블록에 대해 재사용 빈도 관련 정보를 저장하는 것은 너무 큰 메모리 공간을 요구하게 되어 사실상 불가능하다.

EAF에서는 모든 블록들을 저장하지 않으면서 블록 단위의 재사용 빈도 예측을 수행한다. 블록의 재사용 빈도가 높다는 것은 블록이 교체되고 얼마 지나지 않아 다시 사용될 확률이 높다는 것을 나타내고 재사용 빈도가 낮다는 것은 블록이 교체된 후 다시 사용되지 않는다는 것을 나타낸다. 이를 바탕으로 EAF에서는 최근 캐쉬에서 교체된 블록들의 주소들을 저장하여 블록들의 재사용 빈도를 예측하는 구조를 제안하였다. 그림 3은 EAF의 구조를 나타낸다. 프로세서에서 블록의 요청이 D, E, F, A, B, C, D, Z의 순서로 발생하였다고 가정한다. D블록이 캐쉬에 재요청되었을 때 해당 블록은 이미 캐쉬에서 교체된 상태이기 때문에 적중 실패가 발생하게 되고 재사용 빈도 예측을 위해 EAF를 검사한다. D블록의 경우 현재 EAF에 저장되어 있으므로 재사용 빈도가 높게 예측된다. 이와 반대로, 캐쉬에서 적중 실패한 주소가 EAF에서 존재하지 않는다면, 그 블록은 오랫동안 요청되지 않다가 요청된 것이므로 재사용 빈도가 낮은 것으로 예측된다. 이렇게 블록단위로 재사용 빈도를 예측한 후, 재사용 빈도가 높게 예측되는 블록들은 MRU 위치로 삽입하고 그렇지 않은 블록들은 BIP를 사용하여 캐쉬에 삽입한다.

하지만 이와 같은 방법도 결국 교체된 블록의 주소를 따로 저장해야 하므로 큰 크기의 작업 그룹을 측정하기 위해서는 캐쉬의 크기와 유사한 크기의 저장 공간을 필요로 하는데, 이는 시스템 전체 블록의 정보를 저장하는 것보다는 작지만 역시 상당한 크기의 메모리 공간을 요구한다. EAF에서는 측정 가능한 작업 그룹의 크기는 줄이지 않으면서 필요한 공간의 크기를 감소시키기 위하여 블룸 필터[10]를 사용한다. 블룸 필터는 특정 값(EAF에서는 주소 값) 자체를 저장하는 것이 아니라 그 값의 현재 저장상태만을 확인할 수 있도록 하는 확률적인 구조이다. EAF에서는 블록의 주소 값 자체가 필요한 것이 아니라 적중 실패가 발생한 블록이 최근에 교체되었는지를 확인하기 위해 EAF에 존재하는지 확인만 하면 되므로 블룸 필터를 적용시켜 저장 공간을 크게 감소시킬 수 있다.

III. Cache Management Scheme Using Per-Set Reuse Information

본 장에서는 세트별 재사용 정보 기반 쓰래싱 저항 캐쉬 관리 기법을 기술한다. 본 논문에서는 세트단위로 쓰래싱을 예측하고 쓰래싱이 높게 발생되는 것으로 예측되는 세트들에 2차 레벨에 추가시킨 작은 크기의 캐쉬를 할당해 준다. 이를 위해서는 쓰래싱이 발생한 세트 예측이 필요한데, 본 논문에서는 EAF[9]에서 전체 캐쉬의 블록 단위 재사용 빈도 예측을 위해 사용되었던 구조를 수정하여 사용한다.

EAF에서는 2차 레벨 캐쉬에 교체된 주소를 저장시키기 위해 블룸 필터를 활용하였다. 즉, 캐쉬 전체 단위로 교체된 블록들의 주소를 모두 저장하여 블록들의 재사용 빈도를 예측하였다. 본 논문에서는 이와 같은 구조를 세트 단위로 적용한다. 블록이 교체된 주소를 블룸 필터를 통해서 세트 단위로 할당되어 있는 비트 배열에 각각 저장한 후 각 세트에서 적중 실패가 발생한 경우 해당 블록이 세트별 비트 배열에 저장되어 있는지를 검사한다. 해당 세트와 연관된 비트 배열에 블록이 존재 한다면, 그 블록은 최근에 교체되었지만 곧바로 요청되어 적중 실패가 발생한 것으로서 재사용 빈도가 높다고 예측할 수 있다. 이와 같은 방법으로 세트 단위의 블록 재사용 빈도를 예측하여 재사용 빈도가 높게 예측되는 블록들이 위치하는 세트들을 추가시킨 작은 크기의 캐쉬에 할당한다.

1. Bloom Filter

본 논문에서는 쓰래싱을 완화시키기 위하여 2차 레벨에 작은 크기의 캐쉬를 추가한다. 추가된 작은 크기의 캐쉬를 효율적으로 사용하기 위해서는 쓰래싱이 가장 심각하게 발생하는 세트들을 할당하여야 한다. 이를 위해서 EAF에서 전체 캐쉬 단위의 교체된 블록들을 효율적으로 저장하기 위해 사용되던 블룸 필터의 구조를 수정한다.

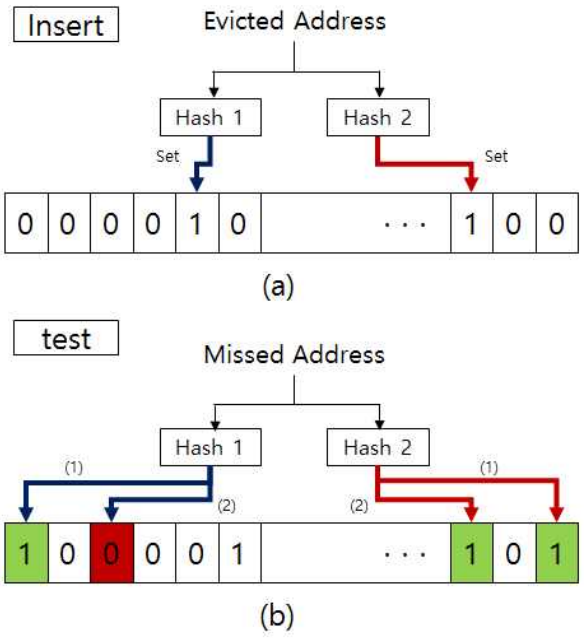


Fig. 4. Bloom filter operation

이에 앞서 블룸 필터의 구조를 간략하게 설명하도록 한다. 블룸 필터는 동작은 3가지로 이루어진다. 그림 4는 삽입과 검사의 2가지 블룸 필터 동작을 나타낸다. 4-(a)는 삽입 동작을 나타내는데, 교체된 블록의 주소가 결정되면 이 주소를 2개의 해시 함수를 사용하여 비트 배열의 색인으로 변환시킨다. 이 2개의 색인을 사용해서 각 비트 배열의 값들을 '1'로 세트시켜주면 동작이 완료된다. 4-(b)는 검사 동작을 나타낸다. 검사는 적중 실패가 발생한 주소의 재사용 빈도를 검사하기 위하여 실시되는데 검사와 똑같은 방식으로 2개의 해시 함수를 사용하여 적중 실패가 발생한 주소를 색인으로 변환시켜 비트 배열을 접근한다. 여기서 중요한 점은 2개의 해시 함수를 사용해 접근한 비트 배열의 값 2개가 전부 '1'이여야 해당 주소가 블룸 필터에 존재한다는 것이다. 따라서 (1)의 접근은 블룸 필터에 해당 주소가 존재하는 것이고, (2)의 접근은 해당 주소가 존재하지 않는 경우이다. 마지막은 초기화 동작이다. 블룸 필터는 비트 배열을 사용하여 데이터를 저장하는데 이러한 비트 배열을 초기화 해주지 않으면 어떤 값을 검사해도 해당 값이 존재한다는 결과가 나오기 때문이다. 본 논문에서는 2가지의 해시 함수를 사용하므로 하나의 데이터를 저장하는데 필요한 비트 배열의 값은 2개이다. 따라서 비트 배열이 저장할 수 있는 주소의 수는 비트배열의 수를 2로 나눈 수이며, 이 수 만큼의 주소가 저장된 후에는 비트 배열의 값을 다시 초기화 해주어야 한다. 본 논문에서는 이를 위해 블룸 필터에 저장된 주소의 숫자를 세기 위한 카운터를 사용한다.

데이터를 저장하는 비트 배열의 길이는 세트별로 저장할 수 있는 교체된 블록의 수를 나타내는 것으로 볼 수 있으며 비트 배열이 길수록 더 많은 블록들을 저장할 수 있다. 본 논문에서는 실험에 8웨이의 2차 레벨 캐쉬를 사용하였는데 비트 배열은 웨이 수와 동일하게 8개의 블록을 저장할 수 있도록 한다. 따라서 저장된 주소의 숫자를 세기 위한 카운터는 3비트를 사용한다.

2. Implementation

그림 5는 EAF와 블룸 필터를 포함하는 제안 기법의 전체적인 동작방식을 나타낸다. 세트별로 불균형하게 발생하는 쓰래싱을 해결하기 위해 쓰래싱이 발생하는 세트들을 추가시킨 쓰래싱 완화 캐쉬에 할당한다. EAF 구조를 사용하여 측정된 세트별 쓰래싱 발생빈도를 쓰래싱 카운터에 저장시키고 그 값을 기준으로 쓰래싱 완화 캐쉬에 할당 시킬 세트들을 결정한다. 2차 레벨 캐쉬에서 블록이 교체된 경우 교체되는 블록의 주소를 EAF 구조에 저장시킨다. 교체된 블록의 주소로 2개의 해시 함수를 사용하여 인덱스를 생성시켜 비트배열의 해당하는 위치에 접근하여 해당 비트의 값을 '1'로 바꿔주고 블룸 필터 초기화를 위해 사용될 블룸 카운터 값을 1증가 시킨다. 그 후 2차 레벨 캐쉬에 대한 요청이 적중 실패가 되면 해당 주소로 2개의 해시 함수를 사용하여 EAF 구조에 접근 시킬 인덱스를 생성시켜 해당 위치에 접근한다. 비트배열의 두 값이 모두 '1'인 경우, 해당 블록이 최근에 교체되어 블룸 필터에 저장되어 있는 것으로 재사용 빈도가 높은 블록이 교체된 것으로 볼 수 있다. 따라서 쓰래싱 카운터에 접근하여 값을 1증가 시켜준다. 만약 쓰래싱 카운터 값이 실험적으로 도출된 최적의 임계값인 5를 넘는 경우에는 해당 세트를 추가시킨 쓰래싱 완화 캐쉬에 할당하여 쓰래싱 완화 캐쉬로도 블록이 적재될 수 있도록 하여 쓰래싱을 완

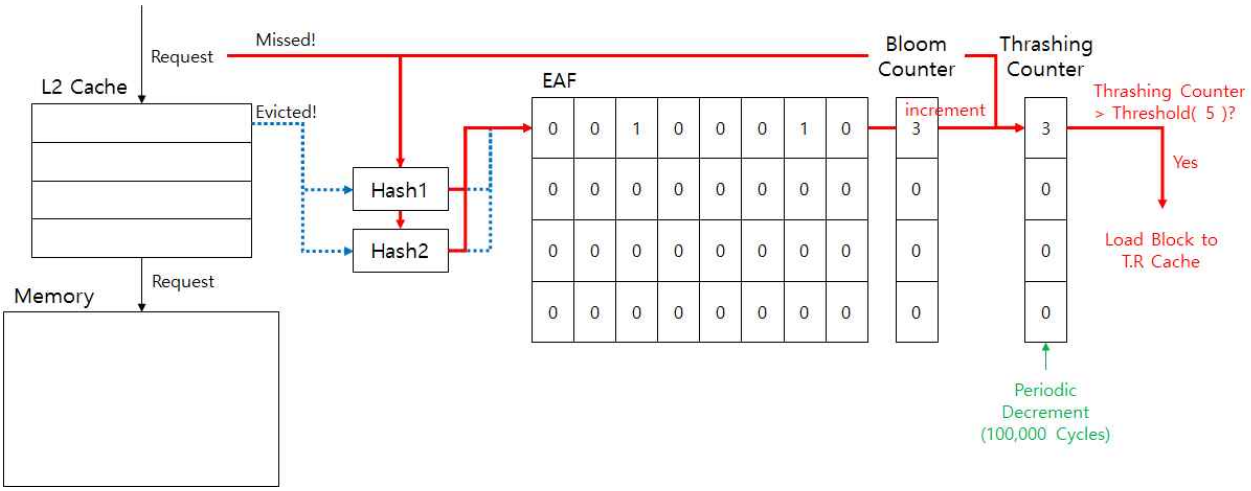


Fig. 5. Thrashing resistant cache management scheme

화 시킬 수 있도록 한다. 쓰래싱 카운터 값은 오래된 정보를 가지고 있지 않도록 10만 사이클 마다 그 값을 1씩 감소시켜 준다.

그림 6은 메모리에서 2차 레벨로의 블록 전송 방식을 보여주고 있다. 2차 레벨 캐쉬 세트의 상태는 쓰래싱이 발생한 세트와 그렇지 않은 세트로 나뉜다. 세트별 상태를 구분하기 위해 본 논문에서는 블룸 필터를 사용하여 측정한 블록별 재사용 빈도 예측 값과 쓰래싱 카운터를 사용한다. 쓰래싱 완화 캐쉬는 작은 크기의 캐쉬이므로 쓰래싱이 발생한 세트의 블록들을 전부 쓰래싱 완화 캐쉬로만 적재시킨다면 블록들을 오래 보관하지 못하고 계속해서 교체되게 될 것이다. 따라서 블록을 2차 레벨 캐쉬와 쓰래싱 완화 캐쉬를 번갈아 적재시킨다. 2차 레벨 캐쉬에서 적중 실패가 발생할 경우 블록을 하나 교체시키게 되는데 이 블록은 블룸 필터에 삽입되게 되므로 블룸 필터의 카운터 값은 증가하게 된다. 이 카운터 값은 블록을 적재 시킬 때마다 변하게 되므로 이 값의 최하위 비트를 사용하여 0인 경우에는 2차 레벨 캐쉬로 적재하고 1인 경우에는 쓰래싱 완화 캐쉬로 적재하도록 한다.

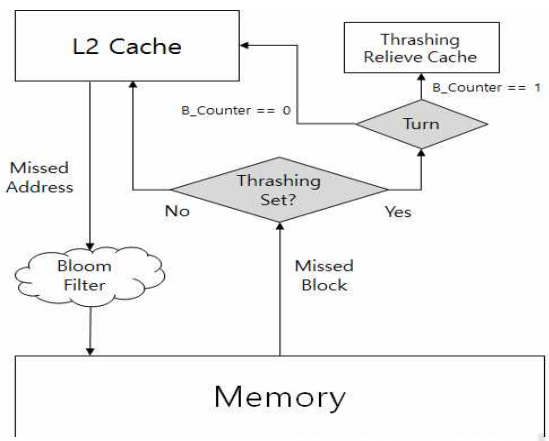


Fig. 6. Block transfer method

IV. Experiments

1. Experiment Environments

본 장에서는 제안된 세트별 재사용 정보 기반 쓰래싱 저항 캐쉬 관리 기법의 효과를 분석한다. 변경된 메모리 구조의 성능을 평가하기 위해서 SimpleScalar[11] 시뮬레이터를 사용한다. 입력 벤치마크 프로그램으로는 SPEC CPU2000[12]에서 10개의 벤치마크를 선정하여 사용한다. 실험을 위해 사용한 시스템의 구성 변수들은 표 1과 같다.

Table 1. System Parameters

System Parameter	Value
Instruction L1 Cache	16K Bytes, 256 Sets, 32B Line, 2Way
Instruction L1 Cache Latency	1 Cycle
Data L1 Cache	16K Bytes, 256 Sets, 32B Line, 2Way
Data L1 Cache Latency	1 Cycle
Unified L2 Cache	256K Bytes, 1024 Sets, 32B Line, 8Way
Unified L2 Cache Latency	16 Cycles

앞서 기술한 바와 같이 제안하는 기법은 2차 레벨에 작은 크기의 캐쉬를 추가하여 2차 레벨에서의 적중 실패를 감소시킴으로써 전체적인 평균 메모리 접근시간을 줄임으로써 성능을 향상시키는 것을 목적으로 한다. 따라서 시스템의 성능이 캐쉬에 영향을 거의 받지 않는 벤치마크들은 제안하는 기법을 적용하여도 성능을 향상시키기가 어렵다. 그러므로 제안하는 기법이 효과적으로 성능을 향상시키는지 확인하기 위해서 우선 성능이 캐쉬의 크기에 민감한 벤치마크와 그렇지 않은 벤치마크로 구

분한다.

본 논문에서는 캐쉬의 크기에 민감한 벤치마크와 그렇지 않은 벤치마크들을 구분짓기 위하여 하나의 통합된 캐쉬만을 사용한 경우에 천개의 명령어 당 적중 실패(MPKI, Misses Per Kilo Instructions)가 얼마나 발생하는 지를 측정한다. 10개의 벤치마크에 대해 측정한 결과 값을 MPKI의 변화율을 기준으로 하여 캐쉬에 민감한 벤치마크(CS, Cache Sensitive)와 캐쉬에 민감하지 않은 벤치마크(CI, Cache Insensitive) 두 가지로 분류한다. 그림 7과 그림 8은 두 종류 벤치마크의 MPKI를 나타내고 있다.

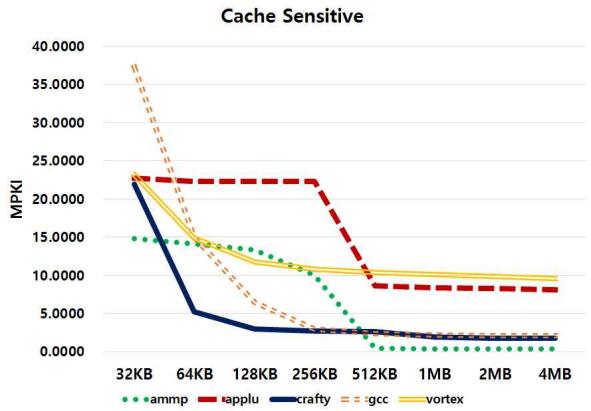


Fig. 7. MPKI of cache sensitive benchmarks

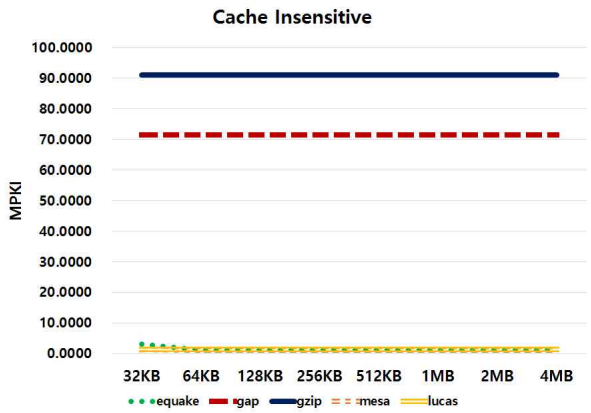


Fig. 8. MPKI of cache insensitive benchmarks

그림 7은 캐쉬의 크기에 따라 성능이 크게 차이나는 캐쉬 크기에 민감한 벤치마크들이다. 그래프를 살펴보면 캐쉬 크기가 32KB에서 1MB까지 증가함에 따라 전체적으로 MPKI가 감소하는 추세를 보이는 것을 볼 수 있다. 이 중에서도 특히 ammp, applu 벤치마크를 살펴보면 특정 부분에서 그래프가 급격하게 꺾이는 것을 볼 수 있다. 두 벤치마크의 경우 256KB 이전까지는 MPKI가 거의 일정하다가 512KB 이상의 캐쉬를 사용한 경우에 MPKI가 급격하게 떨어진다. 이는 512KB보다 작은 크기의 캐쉬를 사용한 경우, 작업 그룹의 크기가 캐쉬의 크기보다 크게 되어 지속적으로 적중실패가 발생하다가 512KB 이상의 캐쉬를 사용하게 되면 작업 그룹이 캐쉬의 크기보다 작게 되어 적중 실패가 급격하게 감소한 것으로 볼 수 있다. 나머지 벤치마크들은 32KB에서 캐쉬의 크기를 증가시킴에 따라 지속적으로 MPKI가 감소되는 것을 볼 수 있다. crafty, vortex 벤치마크의 경우 64KB, gcc 벤치마크의 경우 128KB 이전까지 급격하게 감소하다가 이후에는 거의 일정한 비율로 감소하는 것을 볼 수 있다. 이들은 LRU 교체 정책에 맞는 벤치마크들로 32KB부터 적중이 발생하였으며 캐쉬가 특정 크기 이상이 되면 전체 작업그룹을 수용할 수 있어서 MPKI가 일정하게 유지되는 것임을 알 수 있다.

그림 8의 벤치마크들은 캐쉬 크기가 증가하는 것과는 관계없이 MPKI값이 일정한 것을 볼 수 있는데 이러한 벤치마크들은 캐쉬에 상관없이 성능이 일정할 것임을 예측할 수 있다. 해당 벤치마크들에는 캐쉬 관리 기법을 바꾸어도 성능에 미치는 영향이 적을 것이라는 것을 알 수 있다.

2. Experiment Results

실험 결과를 분석하기에 앞서 기존 시스템에서 2차 레벨 캐쉬의 크기에 따른 성능 향상의 정도를 측정하여 본 실험의 결과가 기존의 시스템에 비해 캐쉬를 얼마나 효율적으로 사용할 수 있는지를 비교할 수 있도록 한다.

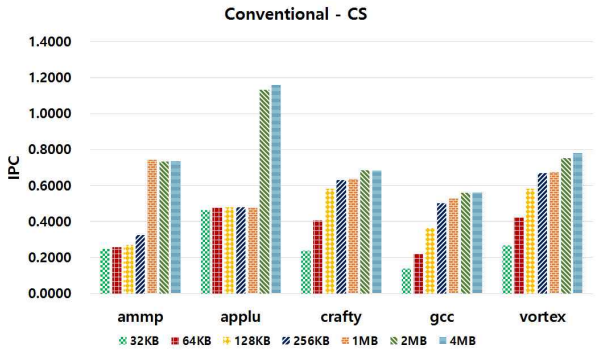


Fig. 9. Performance of cache sensitive benchmarks according to cache size

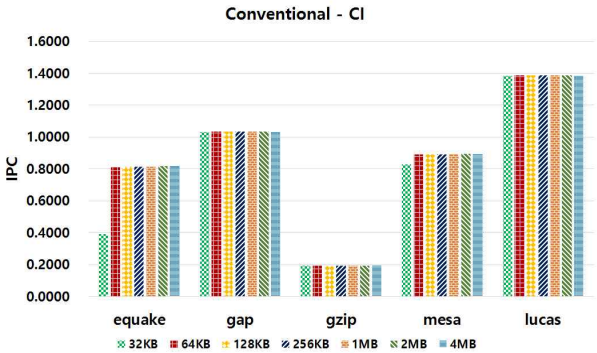


Fig. 10. Performance of cache insensitive benchmarks according to cache size

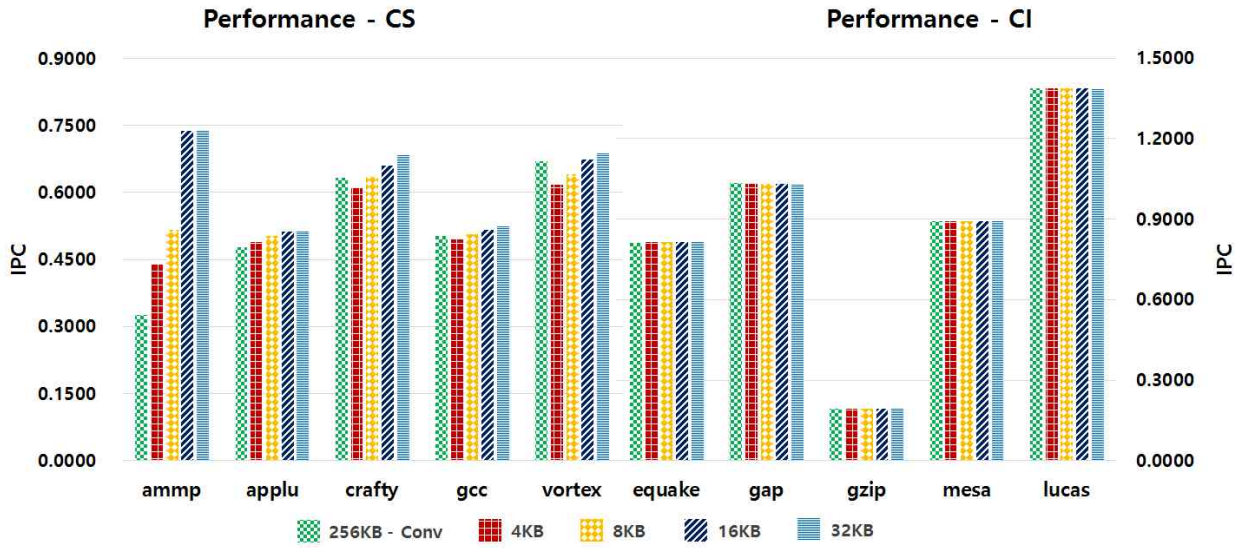


Fig. 11. Performance of thrashing resistant cache management scheme according to thrashing relieve cache size

그림 9와 그림 10은 기존 시스템에서 2차 레벨 캐쉬의 크기에 따른 IPC(Instruction Per Cycles)를 나타낸다. 그림 9는 캐쉬 크기에 민감한 벤치마크들의 캐쉬 크기에 따른 성능을 보여주고 있는데 이러한 벤치마크들은 캐쉬의 크기가 증가함에 따라서 성능이 비례하여 증가하고 있는 것을 볼 수 있다. 그림 10은 캐쉬 크기에 민감하지 않은 벤치마크들을 보여주는데 이 벤치마크들은 캐쉬의 크기가 증가하여도 성능에는 거의 영향을 받지 못하는 것을 볼 수 있다. 성능이 캐쉬의 크기에 영향을 받지 않는 벤치마크 들은 캐쉬를 효율적으로 사용하는 기법을 적용한다 하더라도 성능 향상이 크지 않을 것이라는 것을 예측할 수 있다.

본 논문에서 수행하는 실험에서 모든 벤치마크들에 대해 2차 레벨 캐쉬의 크기는 256KB로 고정하고 쓰래싱 완화 캐쉬의 크기를 4KB, 8KB, 16KB, 32KB로 사용하는 경우에 대해 성능을 측정한다.

그림 11은 제안된 구조를 사용하여 성능을 측정한 결과를 나타낸다. 좌측 5개의 벤치마크는 2차 레벨 캐쉬의 크기에 민감한 벤치마크들이고 우측 5개의 벤치마크는 2차 레벨 캐쉬의 크기에 민감하지 않은 벤치마크들이다. 각 벤치마크별 그래프에서 가장 좌측 값은 쓰래싱 완화 캐쉬를 추가하지 않고 기존의 256KB의 캐쉬만 사용하였을 때의 IPC를 나타낸다.

본 논문에서는 쓰래싱 완화 캐쉬를 2차 레벨에 추가하여 전체 작업 그룹 중 일부를 2차 레벨에 남겨두어 적중을 추가로 발생시켜 캐쉬의 효율을 증가시켜 성능을 향상시킨다. 캐쉬의 크기에 민감한 벤치마크들 중에서 ammp, applu 벤치마크의 경우 16KB의 쓰래싱 완화 캐쉬를 사용하였을 때, IPC가 각각 126.64%, 7.21% 증가함을 알 수 있다. 이는 그림 7에서 볼 수 있듯이, 작업 그룹의 크기가 커서 지속적으로 적중 실패만 발생하던 것을 작은 크기의 쓰래싱 완화 캐쉬를 사용하여 일부를 2차 레벨에 오랫동안 머무를 수 있게 함으로써 성능이 향상된 것으로 볼 수 있다. 특히, ammp 벤치마크는 그림 2에서와 같이 세트별 적중 실패의 수가 매우 상이하여 크게 성능이 향상된 것으로 보인다. 16KB의 쓰래싱 완화 캐쉬를 사용할 때, ammp, applu 벤치마크는 기존의 시스템에서 거의 1MB의 캐쉬를 사용한 것과 유사한 성능을 나타낸다. 나머지 crafty, gcc, vortex 벤치마크들은 16KB의 쓰래싱 완화 캐쉬를 사용할 때 IPC가 각각 4.56%, 2.73%, 0.73% 증가한다. 그림 7에서 볼 수 있듯이, 이 벤치마크들은 LRU 교체 정책에 적합한 벤치마크들이었는데 교체 정책을 바꾸지 않고 작은 크기의 쓰래싱 완화 캐쉬를 추가함으로써 성능을 감소시키지 않고 오히려 추가 캐쉬의 공간을 활용하여 성능을 증가시킬 수 있다.

나머지 오른쪽의 5개의 벤치마크들은 캐쉬의 크기에 민감하지 않은 벤치마크들로서 예상대로 모든 벤치마크들에서 쓰래싱 완화 캐쉬에 따른 성능의 변화가 없음을 확인할 수 있다.

V. Conclusions

본 논문에서는 2차 레벨 캐쉬에서 발생하는 쓰래싱 문제를 해결하기 위해 2차 레벨에 쓰래싱을 완화시키기 위한 작은 크기를 캐쉬를 추가하였으며 이를 포함한 캐쉬 관리 기법을 제안하였다. 제안하는 기법은 블룸 필터를 사용하여 세트별로 교체되는 블록이 재사용되는 수를 측정하여 그 값이 임계값을 넘기는 경우에만 해당 세트를 쓰래싱 완화 캐쉬로 할당하여 쓰래싱이 가장 문제가 되는 세트들에서 작업 그룹의 일부만이라도 남길 수 있도록 하여 쓰래싱을 완화한다. 실험을 통해 확인한 결과, 캐쉬 크기에 민감한 쓰래싱이 많이 발생하는 벤치마크들에서는 16KB의 쓰래싱 완화 캐쉬를 사용한 경우 평균 3.81%의

IPC를 향상시키는 것을 확인하였다. 제안하는 기법을 쓰래싱이 많이 발생하는 캐쉬 크기에 민감한 프로그램들에 적용하는 경우에는 성능을 크게 향상 시킬 수 있을 것으로 예상된다.

향후에는 캐쉬 크기에 민감하지 않은 벤치마크들까지 포함하여 모든 프로그램에 대해 적용하여 성능을 향상시킬 수 있는 캐쉬 관리 기법을 연구하고자 한다.

REFERENCES

[1] 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 Proceedings of the International Symposium on Microarchitecture (MICRO), pp. 430-441, 2011.

[2] G. S. Tyson, M. K. Farrens, J. Matthews, and A. R. Pleszkun, "A Modified approach to data cache management," In Proceedings of the International Symposium on Microarchitecture (MICRO), pp. 93-103, 1995.

[3] D. Rolán, B. B. Fraguela, and R. Doallo, "Adaptive Line Placement with the Set Balancing Cache," In Proceedings of the International Symposium on Microarchitecture (MICRO), pp. 529-540, 2009.

[4] J. Peir, Y. Lee, and W. W. Hsu, "Capturing Dynamic Memory Reference Behavior with Adaptive Cache Topology," In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp. 240-250, 1998.

[5] M. K. Qureshi, D. Thompson, and Y. N. Patt, "The V-Way Cache: Demand Based Associativity via Global Replacement," In Proceedings of International Symposium on Computer Architecture (ISCA), pp. 544-555, 2005.

[6] J. M. Kim and S. W. Chung, "Group-Based Replacement Algorithm to Reduce Cache Miss in Last Level Cache," Journal of The Korea Society of Computer and Information, Vol. 6, No. 5, pp. 44-50, Oct. 2010.

[7] D. O. Son, H. J. Choi, J. M. Kim., and C. H. Kim, "Core-aware Cache Replacement Policy for Reconfigurable Last Level Cache," Journal of The Korea Society of Computer and Information, Vol. 18, No. 11, pp. 1-12, Nov. 2013.

[8] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, Jr., and J. Emer, "Adaptive Insertion Policies for High Performance Caching," In Proceedings of International Symposium on Computer Architecture (ISCA), pp. 381-391, 2007.

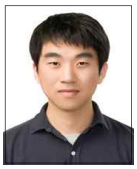
[9] V. Seshadri, O. Mutlu, M. A. Kozuch, and T. C. Mowry, "The Evicted-Address Filter: A Unified Mechanism to Address Both Cache Pollution and Thrashing," In Proceedings of International Conference on Parallel Architectures and Compilation Techniques (PACT), pp. 355-366, 2012.

[10] B. H. Bloom, "Space/time Trade-offs in Hash Coding with Allowable Errors," Communications of the ACM, pp. 422-426, 1970.

[11] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An Infrastructure for Computer System Modeling," Computer, Vol.35, No.2, pp. 59-67, 2002.

[12] SPEC CPU2000 Benchmarks, http://www.specbench.org

Authors



Gyu Yeon Sim received the B.S degree and M.S in electronics and computer engineering from Chonnam National University, Gwangju, Korea in 2015 and 2017 respectively.

His research interests include computer architecture, multiprocessors and embedded systems.

Cheol Hong Kim received the B.S. degree in Computer Engineering from Seoul National University, Seoul, Korea in 1998 and M.S. degree in 2000. He received the Ph.D. in Electrical and Computer Engineering from Seoul National University in 2006. He worked as a senior engineer for SoC Laboratory in Samsung Electronics, Korea from Dec. 2005 to Jan. 2007. Now is working as an Associate Professor at School of Electronics and Computer Engineering, Chonnam National University, Korea. His research interests include embedded systems, mobile systems, computer architecture, SoC design, low power systems, and multiprocessors.