2013년 4월 전자공학회 논문지 제 50 권 제 4 호
Journal of The Institute of Electronics Engineers of Korea Vol. 50, NO. 4, April 2013 http://dx.doi.org/10.5573/ieek.2013.50.4.052

논문 2012-50-4-7

고성능 프로세서를 위한 카운터 기반의 캐시 교체 알고리즘

(Cache Replacement Policy Based on Dynamic Counter for High Performance Processor)

정 도 영*, 이 용 석**

(Do Young Jung and Yong Surk Lee)

요 약

캐시 메모리의 성능에 큰 영향을 미치는 요소 중 하나인 캐시 교체 알고리즘 중에서 가장 최적의 성능을 가지는 알고리즘은 LRU알고리즘이다. LRU알고리즘은 데이터의 temporal locality특성이 강한 프로그램에서 좋은 성능을 보여주지만, 그렇지 않은 프로그램에서는 많은 캐시 미스를 발생시킨다. 본 논문에서는 LRU알고리즘의 이러한 단점을 개선하기 위한 새로운 카운터 기반 교체 알고리즘인 DCR(Dynamic Counter based Replacement) 알고리즘을 제안한다. 본 논문에서는 캐시에 저장된 이후로 교체되기 전까지 다시 사용되지 않는 데이터인 zero reuse line의 발생 추이를 관찰함으로써 프로그램의 temporal locality 특성이 시간에 따라 동적으로 변화함을 보였다. 그리고 이에 착안하여 제안하는 DCR 알고리즘은 주기적으로 zero reuse line의 수를 카운트하여 프로그램의 temporal locality 변화에 대응할 수 있도록 하였다. DCR 알고리즘은 기존의 LRU알고리즘과 비교하여 최대 2.7%, 평균 0.47%의 미스율 감소를 보였다.

Abstract

Replacement policy is one of the key factors determining the effectiveness of a cache. The LRU replacement policy has remained the standard for caches for many years. However, the traditional LRU has ineffective performance in zero-reuse line intensive workloads, although it performs well in high temporal locality workloads. To address this problem, We propose a new replacement policy; DCR(Dynamic Counter based Replacement) policy. A temporal locality of workload dynamically changes across time and DCR policy is based on the detection of these changing. DCR policy improves cache miss rate over a traditional LRU policy, by as much as 2.7% at maximum and 0.47% at average.

Keywords : Cache memory, Cache replacement policy, LRU, Zero reuse line, Counter based policy

Ⅰ. 서 론

메모리 참조 지연시간(Access latency)은 프로세서의 성능을 저하시키는 심각한 문제 중 하나이다. 따라서 메모리의 일부 데이터를 저장하는 캐시메모리(Cache memory; 이하 캐시)가 반드시 사용된다. 그런데 만약 필요한 데이터가 캐시에 없는 경우, 이를 캐시 미스(miss)가 발생하였다고 한다. 캐시 미스로 인한 프로세서의 성능저하는 수백 사이클(cycle)에 이르기 때문에, [1~2] 이를 줄이는 것은 매우 중요하다.

캐시 미스가 발생하는 유형 중 하나인 충돌(conflict) 미스는 하나 이상의 데이터가 동일한 세트(set)로 할당

* 학생회원, ** 평생회원, 연세대학교 전기전자공학부
(Department of Electrical Electronic Eng., Yonsei University)
※ 이 논문은 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. 20120005728).
접수일자: 2012년10월26일, 수정완료일: 2013년3월22일

되어 발생하는 충돌에 의한 미스를 말한다. 충돌 미스는 하나의 세트에 N개의 데이터를 저장할 수 있는 N-way set associative 캐시를 사용함으로써 줄일 수 있으며, way의 수가 많을수록 충돌 미스는 더욱 감소한다.[3] 하지만 set associative캐시에서 미스가 발생할 경우, 해당 세트의 데이터 중 어느 것을 제거하고 새로운 데이터와 교체할 것인지 선택하는 캐시 교체 알고리즘(replacement algorithm)이 반드시 필요하다. 교체 알고리즘은 캐시의 미스율(Miss ratio)에 큰 영향을 주는 요소이며, 특히 L2캐시와 같이 다수의 way를 가지는 캐시일수록 교체 알고리즘에 따른 미스율 차이가 크게 나타난다.

LRU(Least Recently Used)알고리즘은[4] 지금까지 제안된 다양한 캐시 교체 알고리즘 중에서 거의 최적의 성능을 보여준다. 이 알고리즘은 데이터를 최근에 참조된 순서대로 비교할 수 있도록 LRU스택(Stack)을 각각의 세트에 할당한뒤, 캐시 미스가 발생한 경우 이를 참고하여 가장 오랫동안 참조되지 않은 데이터(LRU)를 교체하는 알고리즘이다(그림 1(a)). LRU스택은 가장 최근에 참조된 데이터를 뜻하는 MRU(Most Recently Used)에서부터 LRU까지의 순서로 이루어져 있다. 새롭게 캐시에 저장되거나 최근에 참조된 데이터는 MRU가 되며(그림 1(b)), 이 데이터는 LRU가 되어 교체되기 전까지 캐시에 저장된다. LRU알고리즘은 데이터의 temporal locality(이하 locality)[1]특성이 강한 프로그램에서 매우 좋은 성능을 보여준다.

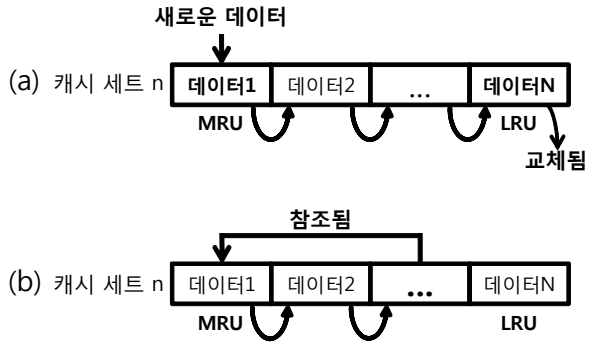

그림 1. LRU 알고리즘의 동작 (a) 캐시 미스 발생시 데이터 교체, (b) 새롭게 참조된 데이터의 MRU로의 이동

Fig. 1. An illustration of LRU policy (a) evicting the LRU when cache miss occurs, (b) putting the just referenced cache line on the MRU.

하지만 LRU알고리즘은 locality 특성이 좋지 않은 프로그램에서는 높은 미스율을 가진다. 이러한 프로그램에는 Zero reuse line(이하 0_reuse) 이 많기 때문인데, 0_reuse란 캐시에 저장된 이후로 교체 될 때까지 단 한 번도 참조되지 않는 데이터를 말한다. 따라서 캐시에 저장할 필요가 없는 데이터이지만, Qureshi의 연구에 따르면 L2 캐시에 저장된 전체 데이터 중 0_reuse가 차지하는 비중은 평균 60%에 이른다.[6] LRU알고리즘은 새롭게 저장되는 데이터를 반드시 MRU로 놓기 때문에, 앞으로 빈번하게 참조될 중요한 데이터들이 0_reuse에 의해 교체되어 추가적인 캐시 미스를 일으키고 프로세서의 성능저하가 발생한다. 이에 본 논문에서는 0_reuse를 효과적으로 제거하여 LRU알고리즘의 미스율을 줄일 수 있는 새로운 캐시 교체 알고리즘을 제안한다.

Ⅱ. 관련 연구

0_reuse로 인해 성능이 저하되는 LRU알고리즘의 문제를 개선하기 위하여 지금까지 다양한 교체 알고리즘들이 제안되어 왔다. LFU(Least Frequently Used)알고리즘은 각각의 데이터가 사용된 횟수를 카운트하고 카운터 값이 가장 작은 데이터를 교체 하는 알고리즘으로, 자주 사용되지 않는 데이터들이 빠르게 교체된다. 그러나 만약 특정한 데이터가 여러 번 사용되어 매우 높은 카운터 값을 가지게 될 경우, 그 후에는 전혀 사용되지 않더라도 쉽게 교체되지 않는 것이 단점이다. 이를 개선한 FBR(Frequency Based Replacement)알고리즘[5]은 캐시메모리 세트 내부에 카운터값을 증가시키지 않는 구역을 설정하여 특정 데이터가 지나치게 높은 카운터값을 가지는 문제를 방지하였다.

한편 LRU알고리즘이 새로운 데이터를 반드시 MRU에 저장하는 것과 반대로, LIP(LRU Insertion Policy)알고리즘은 새로운 데이터를 LRU에 저장하기 때문에 0_reuse를 빠르게 교체할 수 있다. 하지만 데이터가 캐시 메모리에 저장된 직후 곧바로 참조되지 않을 경우 지나치게 빨리 교체되기 때문에 많은 캐시 미스를 일으킨다. Qureshi가 제안한 DIP(Dynamic Insertion Policy)[6] 알고리즘은 새로운 데이터가 저장되는 위치를 MRU와 LRU중에서 선택하도록 하여 LIP의 단점을 개선하였다.

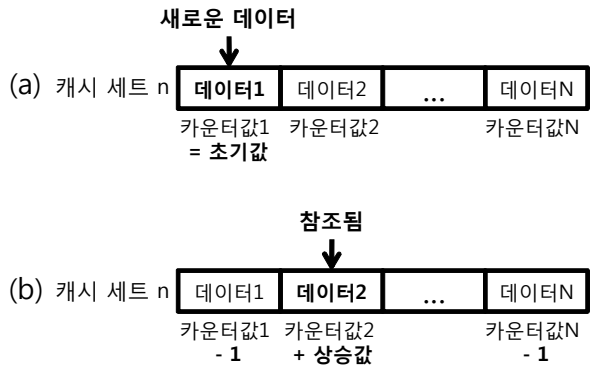

그림 2. WLRU 알고리즘의 카운터값 변경 방법 (a) 초기값 할당, (b) 참조시 카운터값 변경

Fig. 2. Method of changing counter value in WLRU algorithm (a) initialize counter value, (b) change counter value

Wang이 제안한 WLRU(Weighted LRU) 알고리즘은 카운터 기반의 알고리즘으로써 각각의 데이터 블록마다 카운터를 할당하고, 캐시 미스가 발생한 경우 해당 세트 내에서 가장 작은 카운터 값을 가지는 데이터 블록을 교체하는 알고리즘이다.[7] 캐시에 저장되는 데이터에는 카운터의 초기값이 할당되며(그림 2(a)), 이후 데이터가 참조되는 회수에 따라 카운터 값이 최대값과 0의 범위 안에서 변동한다. 어떤 데이터가 참조되면 그 데이터의 카운터 값은 상승하고, 해당 세트 내의 나머지 데이터의 카운터값은 1만큼 감소하도록 하였다(그림 2(b)).

WLRU 알고리즘은 새로 저정되는 데이터가 가지는 카운터의 초기값에 따라서 교체 알고리즘의 특성이 달라질 수 있다. 예를 들어 카운터의 초기값과 상승값이 최대값과 같은 경우, WLRU 알고리즘은 LRU알고리즘과 동일한 동작을 수행한다. 반대로 상승값이 최대값과 같고 초기값이 0이라면 이것은 LIP알고리즘으로 동작한다. 설정한 초기값에 따라서 전혀 다른 교체 알고리즘의 동작을 수행 할 수 있는 것이 WLRU 알고리즘의 장점이다.

하지만 WLRU 알고리즘은 한번 설정된 초기값을 프로그램 수행 중간에 조정할 수 없는 것이 단점이다. 특정한 프로그램의 locality 특성에 맞춰 설정한 카운터의 초기값은 다른 프로그램의 locality 특성에는 적합하지 않을 수 있으며, 그러한 프로그램에 대해서는 성능이 저하될 수 있다.

이러한 WLRU 알고리즘의 단점을 해결하기 위해 본 논문에서는 수행하고 있는 프로그램의 locality 특성에 맞춰 카운터의 초기값을 조정할 수 있도록 하는 캐시 교체 알고리즘인 동적 카운터 기반(Dynamic Counter based Replacement; 이하 DCR) 알고리즘을 제안한다. 이 알고리즘은 기존에 제안된 WLRU 알고리즘을 기반으로 하며, 현재 수행중인 프로그램의 locality 특성을 주기적으로 파악하여 이 특성에 적합하도록 카운터의 초기값을 조정할 수 있게 하였다. DCR알고리즘은 수행하고 있는 프로그램의 locality 특성을 파악하는 것이 매우 중요한데, 이를 파악하는 방법과 함께 제안하는 DCR 알고리즘에 대한 내용을 Ⅳ장에서 다룬다.

Ⅲ. 실 험 환 경

본 논문에서 제안하는 교체 알고리즘과 기존의 LRU 알고리즘의 성능을 비교, 분석하기 위하여 Simple Scalar[11] 시뮬레이터를 사용하여 교체 알고리즘을 구현하였다. 표 1은 시뮬레이터에서 사용한 L1캐시와 L2캐시메모리의 설정을 나타낸 것이다. 모든 실험에서 L1캐시의 설정은 그대로 유지하였다. 또한 제안된 교체 알고리즘의 성능을 검증하기 위한 벤치마크 프로그램으로는 SPEC CPU2000[12] 벤치마크 중 ALPHA 바이너리를 사용하였다.

표 1. 실험환경

Table 1. Experimental Environment

L1 I-캐시	16kB; 64B line; 2-way; LRU
L1 D-캐시	16kB; 64B line; 2-way; LRU
L2 캐시	1MB; 64B line; 16-way

Ⅳ. 제안하는 캐시 교체 알고리즘

1. 프로그램에 따른 0_reuse의 특성

0_reuse는 캐시의 공간을 낭비하는 불필요한 데이터이지만, 프로그램 내부에서 0_reuse의 발생이 어떠한 형태를 가지는지에 대해서는 연구가 이루어지지 않았다. 이에 본 절에서는 캐시를 참조하는 일정 구간마다 발생하는 0_reuse의 수를 카운트하여 프로그램의 locality 특성을 관찰하였다. 그림 3은 각각의 벤치마크 프로그램에서 1M의 크기를 가진 L2캐시를 1,000,000(1M)번 참조할 때마다 발생하는 0_reuse의 수를 나타낸 것이다. 각 그래프의 x-축은 캐시를 1M번

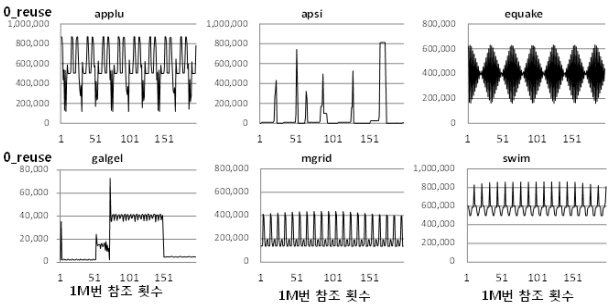

그림 3. LRU 알고리즘을 사용하는 L2 캐시를 1M번 참조할 때마다 발생하는 0_reuse의 수

Fig. 3. The occurrence of 0_reuse per 1M access of L2 cache using LRU algorithm.

참조한 횟수를 나타내며, y-축은 캐시를 1M번 참조하였을 때 발생하는 0_reuse의 수를 카운트한 결과를 나타낸 것이다.

applu와 equake, mgrid, 그리고 swim에서는 발생하는 0_reuse의 수가 일정한 규칙을 가지고 급격하게 변화함을 알 수 있다. 또한 apsi에서는 0_reuse가 국부적으로 발생하며, galgel에서는 급격한 변화가 자주 일어나지 않는 균일한 특성을 보여준다. 그림 3을 통해 각각의 프로그램마다 locality 특성이 다양하다는 사실 뿐만 아니라, 프로그램의 locality특성은 일정하지 않고, 동적으로 변화한다는 것을 알 수 있다.

2. 0_reuse특성에 따른 카운터 기반 교체 알고리즘

DCR 알고리즘은 프로그램의 locality특성이 동적으로 변화한다는 사실에 기반을 두어 카운터의 초기값을 변경하도록 하였다. 캐시를 참조하는 일정 구간을 초기값 변경 구간이라 하며, 이 구간동안 발생하는 0_reuse의 수를 카운트하여 그 수를 기준으로 카운터의 초기값을 변경함으로써, 동적으로 변화하는 프로그램의 locality특성에 대응할 수 있도록 하였다.

수식 1은 카운터의 초기값을 결정하는 방법을 나타낸다. 카운터의 초기값(init)은 0_reuse의 수(N_{0_reuse})에 반비례하며, 0에서부터 최대값(max)까지의 범위를 가질 수 있다. 한 구간(interval)에서 발생한 0_reuse가 많은 경우 초기값은 작은 값을 가지며, DCR 알고리즘은 LIP 알고리즘과 유사하게 동작한다. 반대로 발생한 0_reuse의 수가 적은 경우 초기값은 높은 값을 가져 LRU 알고리즘과 유사한 동작을 수행하게 된다.

$$init = \left(1 - \frac{N_{0\_reuse}}{interval}\right) \times max \tag{1}$$

3. 샘플링을 이용한 초기값의 계산

DCR 알고리즘이 프로그램의 locality특성에 맞는 최적의 초기값을 결정하기 위해서는 0_reuse의 수를 정확하게 카운트하는 것이 매우 중요하다. 하지만 정확한 0_reuse의 수를 카운트하기 위해서는 LRU알고리즘을 사용해야만 하며, DCR 알고리즘으로는 정확한 카운트가 불가능하다. 왜냐하면 낮은 초기값을 할당받아 빠르게 교체된 0_reuse가 실제로 locality가 없는 데이터인지, 아니면 조금 뒤에 사용될 유용한 데이터인지 알 수 없기 때문이다. 그림 4는 DCR 알고리즘을 사용하여 0_reuse의 수를 카운트 한 결과이다. 그림 3의 결과에 비해 훨씬 많은 수의 데이터를 0_reuse로 카운트했음을 알 수 있다.

이러한 문제를 해결하기 위해 DCR 알고리즘은 샘플링(sampling)에 기반을 두어 0_reuse의 수를 카운트 하도록 하였다. L2캐시의 1024개의 세트 중 매 32개의 세트마다 하나의 세트를 LRU알고리즘으로 동작하게 하

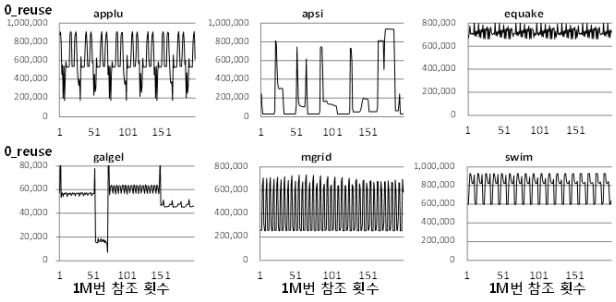

그림 4. DCR 알고리즘만을 사용하는 경우에 발생하는 0_reuse의 수

Fig. 4. The number of 0_reuse using DCR algorithm.

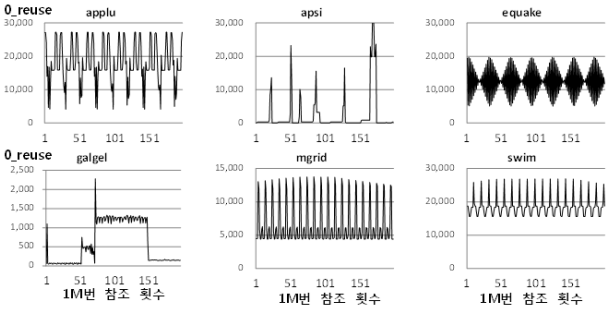

그림 5. 32개의 LRU 세트에서 발생하는 0_reuse의 수

Fig. 5. The number of 0_reuse in 32 LRU sets.

여 0_reuse의 수를 카운트 한다. 그리고 나머지 세트는 DCR 알고리즘으로 동작한다. 캐시메모리의 데이터는 상당히 균일한 특성을 가지고 참조되기 때문에, 아주 일부의 세트만으로도 캐시의 참조 특성을 매우 정확하게 추산(approximation)하는 것이 가능하다.[9~10] 그림 5는 L2캐시의 1024개의 세트 중 LRU 알고리즘으로 동작하는 32개의 세트에서 발생하는 0_reuse의 수를 나타낸 것이다. 그림 3의 결과와 비교해보았을 때, 일부의 세트만으로도 0_reuse의 수를 상당히 정확한 비율로 카운트하였음을 알 수 있다.

4. 하드웨어

DCR 알고리즘을 구현하는데 필요한 하드웨어로 그림 6과 같이 0_reuse의 수를 파악하기 위해 각각의 LRU스택에 추가된 1비트 크기의 0_reuse 비트가 있다. 최초에 데이터가 캐시에 저장되는 경우 0_reuse 비트는 0으로 초기화되며, 그 후 데이터가 재참조 되었을 때 1로 변경된다. 이를 통해 교체되는 데이터의 0_reuse 여부를 알 수 있다. 그림 7은 앞 절에서 설명한 DCR 알고리즘의 구조를 나타낸 것이다. 제안된 알고리즘의 구현에는 초기값 변경 구간마다 초기값을 변경해주기 위해 캐시 참조 회수를 카운트 해주는 참조 카운터와 0_reuse의 수를 카운트하기 0_reuse 카운터, 그리고 새로운 초기값을 계산하기 위한 연산기가 필요하다.

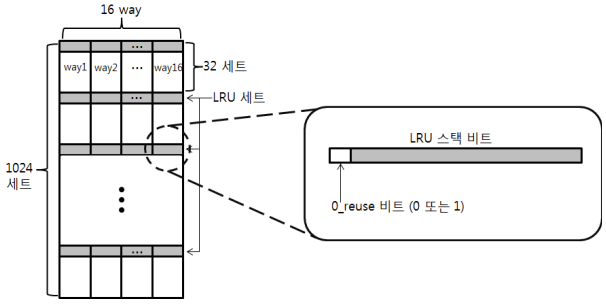

그림 6. 0_reuse를 판별하기 위해 0_reuse 비트가 추가된 LRU 세트의 구조
Fig. 6. Structure of LRU set to determine 0_reuse

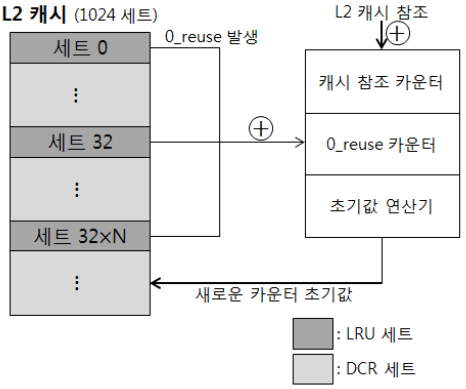

그림 7. DCR 알고리즘의 구조
Fig. 7. The structure of the DCR algorithm.

V. 성 능 평 가

이 장에서는 Ⅳ장에서 제안한 DCR 알고리즘의 성능을 기존의 교체 알고리즘과 비교, 분석한다. 표 2는 시뮬레이션을 수행한 DCR 알고리즘의 변수들을 나타낸 것이다.

카운터의 최대값은 511을 가지도록 설정하였다. 일반적으로 카운터의 최대값은 클수록 좋은 성능을 보여주지만,[8] 많은 하드웨어를 필요로 한다. 그림 8은 parser 벤치마크에서 카운터의 최대값이 캐시 미스율에 미치는 영향을 보여준다. 캐시 미스율은 카운터의 최대값이 511일 때부터 감소하는 폭이 작아짐을 알 수 있다.

캐시 데이터가 참조되었을 때 카운터의 증가값은 최대값보다 약간 작은 값을 가진다. 이는 매우 적은 횟수만 참조된 뒤, 더 이상 참조되지 않는 데이터들이 높은 카운터값을 가져 쉽게 교체되지 않는 문제를 방지하기

표 2. DCR 알고리즘의 변수
Table 2. The variables of the proposed algorithm

변수	수치
초기값 변경 구간	100K, 500K, 1M
최대값	511 (최소값 0)
증가값	392

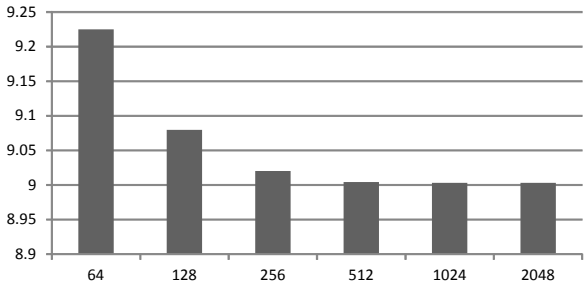

그림 8. parser 벤치마크에서 카운터의 최대값이 미스율에 미치는 영향
Fig. 8. Impact of maximum weight on miss rates for parser benchmark.

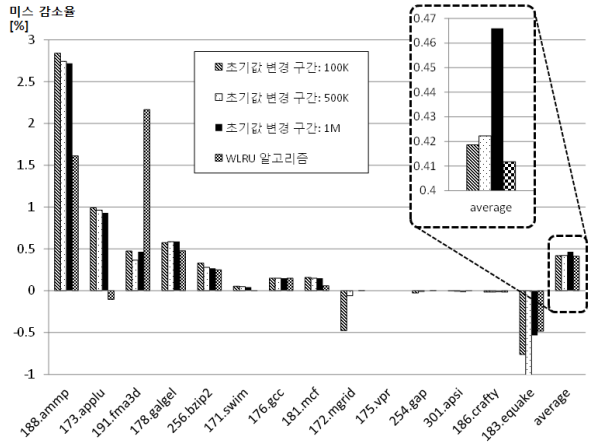

그림 9. LRU 알고리즘 대비 캐시 미스율의 감소
Fig. 9. Miss ratio reduction over LRU algorithm.

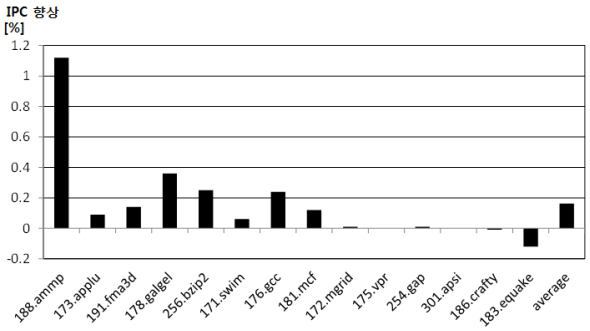

그림 10. LRU 알고리즘 대비 IPC의 향상
Fig. 10. IPC improvement over LRU algorithm.

위해서이다.

그림 9는 기존의 WLRU 알고리즘과 DCR 알고리즘의 LRU 알고리즘 대비 미스율 감소를 측정한 결과이다. WLRU 알고리즘은 511의 초기값과 최대값을 가지며, 참조된 데이터의 카운터 증가값은 392이다. DCR 알고리즘은 초기값 변경 구간을 100K와 500K, 그리고 1M인 경우에 대하여 결과를 확인하였다.

DCR 알고리즘이 가장 우수한 결과를 보여주는 벤치마크 프로그램은 ammp로, LRU알고리즘에 비해 2.72%에서 2.85%의 미스율이 감소하였다. 또한 applu에서는 0.93%에서 1.0%, galgel에서는 0.57%에서 0.6%의 미스율이 감소하였다. 하지만 equake에서는 반대로 미스율이 0.53%에서 1.0%증가하였다. 초기값 변경 구간이 1M인 DCR 알고리즘은 LRU알고리즘과 비교하여 평균 0.47%의 미스율 감소를 보여준다. WLRU 알고리즘은 LRU알고리즘과 비교하여 평균 0.41%의 미스율 감소를 보여주며, 따라서 제안된 DCR 알고리즘이 기존의 WLRU 알고리즘에 비해 더욱 좋은 성능을 보여줌을 알 수 있다.

그림 9의 결과를 통해 초기값 변경 구간에 따른 DCR 알고리즘의 성능차이를 알 수 있다. 초기값 변경 구간이 100K인 경우, ammp에서 LRU대비 2.85%의 미스율 감소로 가장 우수한 결과를 보여주었다. 반면 mgrid에서는 오히려 0.48%의 미스율이 증가하였고, equake에서는 0.76% 증가하였다. 초기값 변경 구간이 1M인 경우, ammp에서 미스율 감소는 2.72%로 100K의 경우에 비해 다소 낮은 성능을 보여주었다. 하지만, equake에서는 미스율이 0.53%증가하는데 그쳤고, mgrid에서는 0.001%로, 그 수치는 미미하지만 100K인 경우와 반대로 미스율이 감소하였다. 이 결과를 통해 최적의 초기값 변경 구간은 프로그램에 따라 다름을 알 수 있으며, 따라서 향후 초기값 변경 구간을 동적으로 변화시킬 수 있는 방법에 대해 추후 연구가 이루어져야 할 것으로 판단된다.

마지막으로 감소된 미스율을 통해 프로세서의 속도 향상을 측정하였다. 이는 프로세서의 단위 클럭당 수행하는 명령어의 개수인 IPC(Instructions Per Cycle)을 통해 측정할 수 있다. 그림 10은 초기값 변경 구간이 1M인 DCR 알고리즘의 LRU 알고리즘 대비 IPC 향상을 측정한 것이다. LRU알고리즘과 비교하여 최대 1.1%, 평균 0.17%의 IPC 향상을 보였다.

VI. 결 론

LRU알고리즘은 데이터의 locality특성을 고려하지 않고 새롭게 저장되는 데이터를 반드시 MRU로 놓기 때문에, 0_reuse가 많은 프로그램에서 성능이 저하된다. 이러한 단점을 개선하기 위하여 본 논문에서는 WLRU 알고리즘을 기초로 하는 DCR 알고리즘을 제안하였다.

DCR 알고리즘은 프로그램의 locality 특성이 동적으로 변화한다는 점을 이용한 알고리즘이다. 일정한 초기값 변경 구간마다 발생하는 0_reuse의 수를 카운트하여 그 수에 따라 WLRU 알고리즘의 변수 중 하나인 초기값을 동적으로 변화시킬 수 있도록 하였다. 제안하는 DCR 알고리즘은 기존의 LRU알고리즘과 비교하여 최대 2.72%, 평균 0.47%의 미스율 감소를 보였다.

참 고 문 헌

[1] Hennessy J. L, Patterson D., “Computer Architecture: A Quantitative Approach”, Fourth Edition, Morgan Kaufmann Publishers, pp. 288-309, 2007.

[2] M. V. Wilkes, “The Memory Gap and the Future of High Performance Memories”, SIGARCH Comput. Archit. News, 2001.

[3] M. D. Hill, A. J. Smith, “Evaluation associativity in CPU Caches”, IEEE Trans. Comput., 38(12):1612-1630, 1989.

[4] H. Al-Zoubi, A. Milenkovic, M. Milenkovic, “Performance Evaluation of Cache Replacement Policies for the SPEC CPU2000 Benchmark Suite”, ACM-SE 42: Proceedings of the 42nd annual southeast regional conference P267-272, New York, USA, 2004.

[5] J. T. Robinson, M. V. Devarakonda, “Data Cache Management Using Frequency-based Replacement”, In Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, P.134-142, 1990.

[6] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, J. Emer, “Adaptive Insertion Policies for High Performance Caching”, ISCA 2007: Proceedings of the 34th annual international symposium on Computer Architecture, P.381-391, New York, NY, USA, 2007.

[7] Q. Wang, “WLRU CPU Cache Replacement Algorithm”, Ph.D. Thesis, The University of Western Ontario, 2006.

[8] N. Duong, R. Cammarota, D. Zhao, T. Kim, and A. Veidenbaum, “SCORE: A Score-based Memory Cache Replacement Policy”, JWAC 2010 : 1st JILP Worshop on Computer Architecture Compettions: Cache Replacement Championship, 2010.

[9] M. K. Qureshi, D. N. Lynch, O. Mutlu, and T. N. Patt, “A case for MLP-aware cache replacement”, ISCA 2006: Proceedings of the 33th annual international symposium on Computer Architecture, 2007.

[10] S. Khan, Y. Tian, and D. A. Jimenez, “Sampling Dead Block Prediction for Last-Level Caches”, Micro, 2010.

[11] www.simplescalar.com

[12] www.spec.org/cpu2000/

저 자 소 개

정 도 영(학생회원)

2011년 세종대학교 전자공학과 학사 졸업.

2013년 연세대학교 전기전자공학부 석사 졸업.

2013년 LG 디스플레이 연구원.

<주관심분야 : 마이크로 프로세서, 캐쉬메모리 구조, SoC>

이 용 석(평생회원)-교신저자

1973년 연세대학교 전기공학과 학사 졸업.

1977년 University of Michigan, Ann Arbor 석사 졸업.

1981년 University of Michigan, Ann Arbor 박사 졸업.

1993년~현재 연세대학교 전기전자공학과 정교수.

<주관심분야 : 마이크로 프로세서, 네트워크 프로세서, 고성능 연산기 설계, SoC>