선택적 희생 캐쉬를 이용한 저전력 고성능 시스템 설계 방안

(An Energy-Delay Efficient System with Adaptive Victim Caches)

김 철 홍 † 심 성 훈 † 전 주 식 †† 장 성 태 †††
(Cheol Hong Kim) (Sunghoon Shim) (Chu Shik Jhon) (Seong Tae Jhang)

요 약 계층적 메모리 구조를 사용하는 시스템에서 상위 캐쉬의 적중률은 전체 시스템의 성능을 결정하는 중요한 요소 중 하나이다. 시스템 설계 시 전력 효율성이 중요한 고려사항이 되고 있는 최근에는 전력 소모량이 많은 하위 캐쉬로의 접근을 줄이기 위해 상위 캐쉬의 적중률을 높이는 방안이 더욱 부각되고 있다. 본 논문에서는 선택적 희생 캐쉬를 이용하여 상위 캐쉬의 적중률을 높임으로써 저전력 고성능 시스템을 설계하는 방안을 제안하고자 한다. 희생 캐쉬는 직접 사상 1차 캐쉬에서의 충돌 미스로 인한 메모리 시스템의 성능 저하를 줄이기 위해 추가되는 모듈이다. 제안하는 구조는 희생 캐쉬로의 데이타 할당 정책을 변형하여 재참조 가능성이 높은 데이타를 보다 오랜 시간동안 상위 캐쉬 내에 유지시킴으로써 상위 캐쉬의 적중률을 높이고, 이를 통해 접근 시간이 길고 전력 소모량이 많은 하위 캐쉬로의 접근 횟수를 줄이고자 한다. 참조 기반 희생 캐쉬는 1차 캐쉬에서 교체되는 데이타 중에서 프로세서에 의해 많이 참조되었던 데이타만을 골라서 할당한다. 교체 기반 희생 캐쉬는 1차 캐쉬에서 교체되는 데이타 중에서 충돌 미스가 자주 발생하는 위치에 할당되었던 데이타만을 골라서 할당한다. Wattch를 사용한 실험 결과 제안하는 구조는 기존의 희생 캐쉬 시스템보다 좋은 성능을 보일뿐 아니라, 전력 효율성도 높음을 알 수 있다.

키워드 : 컴퓨터 구조, 계층적 메모리 시스템, 희생 캐쉬, 저전력 시스템

Abstract We propose a system aimed at achieving high energy-delay efficiency by using adaptive victim caches. Particularly, we investigate methods to improve the hit rates in the first level of memory hierarchy, which reduces the number of accesses to more power consuming memory structures such as L2 cache. Victim cache is a memory element for reducing conflict misses in a direct-mapped L1 cache. We present two techniques to fill the victim cache with the blocks that have higher probability to be re-reqeusted by processor. Hit-based victim cache is filled with the blocks which were referenced frequently by processor. Replacement-based victim cache is filled with the blocks which were evicted from the sets where block replacements had happened frequently. According to our simulations, replacement-based victim cache scheme outperforms the conventional victim cache scheme about 2 % on average and reduces the power consumption by up to 8 %.

Key words : computer architecture, hierarchical memory system, victim cache, low power system

1. 서 론

고성능 시스템을 설계하는데 있어서 평균 데이타 접근 시간을 감소시키는 것은 가장 중요한 요소 중 하나이다. 최근의 고성능 시스템에서는 메모리 접근의 효율성을 높이기 위해서 계층적 메모리 시스템(Hierarchical Memory System)을 사용한다. 계층적 메모리 시스템은 접근 시간이 빠른 작은 크기의 메모리를 상위에 위치시키고 접근 시간이 느린 큰 크기의 메모리를 하위에 위치시키는 구조로 이루어진다. 이와 같은 계층 구조에서 1차 캐쉬(L1 Cache)와 2차 캐쉬(L2 Cache)는 프로세서 칩 내에 위치함으로써(On-Chip Cache) 빠른 접근 시간을 제공하고 있다. 하지만, 프로세서 클럭 속도와 메모리 접근 시간의 차이가 점차 커지면서 2차 캐쉬와 같은 하위 캐쉬로의 접근 시간은 급격하게 증가하였다[1].

† 학생회원 : 서울대학교 전기컴퓨터공학부
kimch@panda.snu.ac.kr
shshim@panda.snu.ac.kr
†† 종신회원 : 서울대학교 전기컴퓨터공학부 교수
csjhon@panda.snu.ac.kr
††† 종신회원 : 수원대학교 컴퓨터학과 교수
stjhang@suwon.ac.kr
논문접수 : 2005년 4월 29일
심사완료 : 2005년 9월 13일

그 결과, 상대적으로 빠른 접근 시간을 제공하는 1차 캐쉬의 적중률(Hit Rates)을 향상시키기 위한 연구가 활발하게 이루어져왔다. N. P. Jouppi는 작은 수의 엔트리를 가지는 희생 캐쉬(Victim Cache)를 사용하여 직접 사상(Direct-Mapped) 1차 캐쉬에서 발생하는 충돌 미스(Conflict Miss)를 상당 부분 감소시킬 수 있음을 보였다[2]. 희생 캐쉬는 1차 캐쉬에 존재하던 데이타가 프로세서의 요청에 의해서 교체되었지만 가까운 시간 내에 재참조되는 경우 하위 캐쉬의 참조로 인해 데이타 접근 시간이 증가되는 것을 막기 위해 추가되는 메모리 모듈로써 빈번한 충돌 미스로 인해 직접 사상 캐쉬의 적중률이 저하되는 현상을 막기 위해 제안되었다. 희생 캐쉬에는 1차 캐쉬에서 교체된 데이타 블록들이 순차적으로 할당되며 1차 캐쉬와 동일한 위치에서 접근이 가능하도록 설계된다. Column-Associative 캐쉬는 서로 충돌하는 주소(Address)를 가지는 블록들을 서로 다른 셋(Set)에 위치하게 하는 해싱(hashing) 함수를 동적으로 적용하여 직접 사상 캐쉬의 충돌 미스를 감소시킨다[3]. Bounce-back 캐쉬는 소프트웨어를 이용하여 높은 시간 지역성(Temporal Locality)을 가지는 블록들이 캐쉬에서 교체되었을 경우에는 캐쉬로 다시 보내주는 기법을 통해 캐쉬의 적중률을 향상시킨다[4]. NTS(Non-Temporal Streaming) 모델은 캐쉬의 재사용 패턴을 분석하여 캐쉬를 물리적으로 자주 참조되는 영역과 그렇지 않은 영역으로 구분하여 할당 자체를 분리하여 관리하는 기법을 통해 기존 캐쉬보다 높은 성능을 제공한다[5]. MAT(Memory Address Table) 모델은 데이타의 참조 횟수에 따라 캐쉬 블록을 자주 참조되는 영역과 그렇지 않은 영역으로 동적으로 구분하는 기법을 통해 캐쉬의 적중률을 향상시킨다[6].

위에서 열거한 여러 기법들의 적용과 응용 프로그램에서 사용되는 데이타의 시간 지역성과 공간 지역성(Spatial Locality)의 특성으로 인해 최근의 1차 캐쉬는 프로세서가 요구하는 대부분의 메모리 요구를 만족시키고 있다. 하지만, 캐쉬의 성능이 지속적으로 증가한다고 하더라도 문제는 여전히 남아있다. 시스템 설계 시 시스템의 전력 소모량(Energy Consumption)이 중요한 고려사항으로 대두되면서 상당히 많은 양의 전력을 소모하는 고성능 캐쉬 메모리 구조에 변화가 필요하게 되었다. Selective cache ways는 캐쉬 접근이 활발하지 않은 시간 동안 캐쉬의 일부 way를 비활성화시키는 기법을 통해 캐쉬의 성능에는 크게 영향을 주지 않는 범위 내에서 전력 소모를 줄이는 방법을 제공한다[7]. 1차 캐쉬보다 작은 크기의 필터 캐쉬(Filter Cache)를 프로세서와 1차 캐쉬 사이에 삽입시켜 주 캐쉬로의 접근 빈도를 줄이는 기법도 캐쉬의 성능에는 큰 손실을 주지 않으면서 전력 소모량을 줄이기 위해 제안된 기술이다[8].

최근의 마이크로프로세서들은 충돌 미스(Conflict Miss)의 감소를 통해 적중률을 높이기 위해서 대부분 연관 사상(Set-Associativity)으로 구성된다. 하지만, 연관 사상 캐쉬는 직접 사상(Direct Mapped) 캐쉬에 비해 접근 시간이 길고 참조 시 보다 많은 양의 전력을 소모하는 단점을 가진다. 예를 들어, 4-way 캐쉬의 경우 데이타 참조 시 4개의 way로부터 데이타를 읽은 후, 태그(Tag)를 비교하여 일치되는 하나의 데이타만을 읽어 오기 때문에, 3개의 way로부터 데이타를 읽는 과정에서 발생하는 전력의 낭비가 생기게 된다. 본 논문에서는 직접 사상 1차 캐쉬를 사용하는 계층적 메모리 시스템에 희생 캐쉬를 추가하고, 1차 캐쉬의 접근 패턴을 통해 희생 캐쉬에 할당되는 데이타들에 대한 선택 작업을 수행하여 1차 캐쉬와 희생 캐쉬로 구성되는 일차 레벨 메모리(First Level Memory)의 적중률을 높임으로써 긴 접근 시간과 많은 전력 소모를 필요로 하는 2차 캐쉬나 주메모리로의 접근 횟수를 줄여서 시스템의 성능을 향상시킴과 동시에 전력 소모량도 줄일 수 있는 기법을 제안하고자 한다.

이하 본 논문의 구성은 다음과 같다. 2장에서는 논문의 연구 대상이 되는 희생 캐쉬를 추가한 계층적 메모리 시스템의 구조와 본 논문의 작성 동기를 설명하고, 3장에서는 새로운 데이타 할당 방법을 적용한 선택적 희생 캐쉬 구조를 제안하고, 각 구조에 대해 정성적 분석을 한다. 4장에서는 성능 평가를 수행하는 실험 방법과 제안하는 구조에 대한 실험 결과를 보여주고, 끝으로 5장에서 결론을 맺는다.

2. 희생 캐쉬를 사용하는 계층적 메모리 시스템

희생 캐쉬를 사용하는 일반적인 계층적 메모리 시스템 구조는 그림 1과 같다. 시스템의 동작은 다음과 같은 방식으로 이루어진다. 프로세서로부터 데이타 접근 요청이 있는 경우, 데이타 주소의 인덱스(Index) 값에 따라 1차 캐쉬에서 해당 엔트리를 찾게 된다. 그 후, 엔트리에 위치하는 데이타의 유효 비트를 확인하고 태그 값과 요청하는 데이타의 태그 값을 비교하여 접근하는 데이타가 유효한지 그리고 올바른 데이타인지를 판단한다. 이와 병행하여 희생 캐쉬 내에 유효한 데이타가 있는지 검색한다. 1차 캐쉬에 요청하는 데이타가 있는 경우 데이타를 프로세서로 전송한다. 1차 캐쉬에는 유효한 데이타가 없고 희생 캐쉬에 요청하는 데이타가 존재하는 경우에는 데이타를 프로세서로 전송하고, 1차 캐쉬의 해당 엔트리에 있는 데이타와 참조된 희생 캐쉬 내의 데이타를 서로 교환한다. 1차 캐쉬와 희생 캐쉬 모두 요청하는 데이타가 없는 경우에는 하위 캐쉬로의 접근이 이

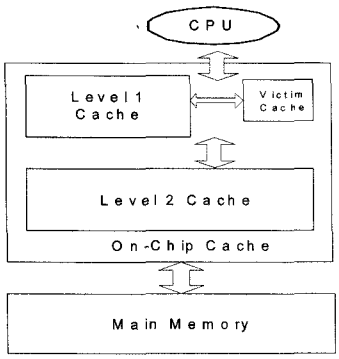


그림 1 계층적 메모리 시스템

루어지고 1차 캐쉬에 하위 캐쉬로부터 읽어온 데이타를 위치시킨다. 이때 충돌로 인해 1차 캐쉬에서 교체되는 데이타는 희생 캐쉬에 할당된다. 희생 캐쉬는 완전 연관(Fully Associative) 구조로 이루어져 있고, FIFO 또는 LRU와 같은 블럭 교체 방식에 따라 1차 캐쉬에서 교체된 데이타를 새로 할당한다.

희생 캐쉬를 사용하는 계층적 메모리 시스템의 성능을 향상시키기 위한 연구와 전력 소모량을 줄이기 위한 연구는 기존의 논문들을 통해 제안된 바 있다. Stiliadis와 Varma는 직접 사상 캐쉬의 성능을 향상시키기 위한 기법으로 Selective Victim Caching(SVC)을 제안하였다[9]. 이 기법은 하위 캐쉬로부터 일차 레벨 메모리로 올라오는 데이타에 대해 기존의 캐쉬 접근 정보를 기반으로 예측을 수행하여 1차 캐쉬로 할당할지 희생 캐쉬로 할당할지를 결정한다. 이를 통해 일차 레벨 메모리의 적중률을 향상시키는 결과를 보였다. SVC 기법에서는 예측을 통해 일차 레벨 메모리로 올라오는 데이타를 선택적으로 1차 캐쉬 또는 희생 캐쉬에 할당하는 데 반해, 본 논문에서는 하위 캐쉬에서 올라오는 데이타는 무조건 1차 캐쉬로 할당하고, 1차 캐쉬에 할당이 된 데이타의 접근 정보를 바탕으로 희생 캐쉬에 들어가는 데이타를 선택하는 기법을 제안하고자 한다. 1차 캐쉬에서 교체된 후 짧은 시간안에 프로세서로부터 재참조될 가능성이 높은 데이타만을 선택하여 희생 캐쉬에 할당하고자 하는 것이다. 희생 캐쉬를 사용하는 일차 레벨 메모리의 전력 소모량을 줄이기 위한 기법으로는, 희생 캐쉬로 접근하는 명령 중에서 참조 미스가 예상되는 명령들에 대해서는 미리 접근을 막음으로써 접근 횟수의 감소를 통해 희생 캐쉬의 전력 소모를 줄일 수 있는 방안이 소개되었다[10]. 본 논문에서는 희생 캐쉬의 접근을 막는 기법보다는 참조 확률이 높은 데이타를 선택하여 할당함으로써 일차 레벨 메모리의 적중률을 높여 전력 소모량을 줄이는 기법을 제안한다.

기존의 희생 캐쉬 구조는 1차 캐쉬에서 교체되어 희생 캐쉬에 할당된 데이타가 단기간에 프로세서로부터 재요청되는 경우에는 일차 레벨 메모리의 적중률을 향상시킬 수 있지만, 추후 요청되지 않는 데이타들도 희생 캐쉬에 할당되는 문제점을 가지고 있다. 즉, 희생 캐쉬에 할당되는 데이타 중에는 희생 캐쉬에 유지되는 기간 동안 프로세서로부터 한번도 참조되지 않는 데이타가 상당 부분을 차지하고 있는 것이다. 그림 2와 그림 3은 각각 정수 응용프로그램과 실수 응용프로그램에서의 희생 캐쉬에 할당되는 데이타 중에서 교체될 때까지 한번도 참조되지 않는 데이타의 비율을 나타내고 있다. 모의 실험은 SimpleScalar를 사용하여 16KB의 크기를 가지는 1차 캐쉬와 8개의 엔트리를 가지는 희생 캐쉬, 32B의 데이타 블럭 크기를 가정하고 수행하였다[11]. 실험 결과, 아래의 그림에서 보는 바와 같이 100개의 데이타가 희생 캐쉬에 할당되는 경우, 15개 정도의 데이타는 참조되어 1차 캐쉬로 이동하는데 반해, 85개 정도의 데이타는 참조되지 않고 2차 캐쉬로 내려가게 된다. 위의 결과를 통해, 성능 향상에는 기여하지 않고 희생 캐쉬에 불필요한 블록 교체를 유발하는 데이타가 상당 부분 있다는 사실을 확인할 수 있다. 1차 캐쉬로부터 얻어지는 정보를 바탕으로 희생 캐쉬로의 데이타 할당을 선택적으로 수행한다면 보다 효율적인 계층적 메모리 시스템을 설계할 수 있을 것으로 기대된다.

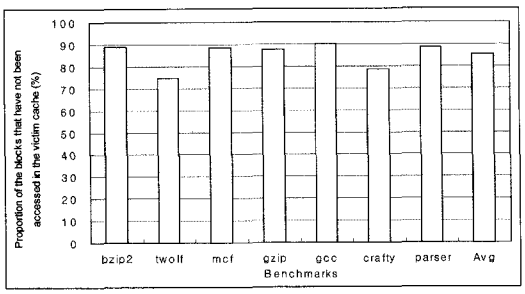


그림 2 정수 응용프로그램에서의 희생 캐쉬 내의 사용되지 않는 데이타의 비율

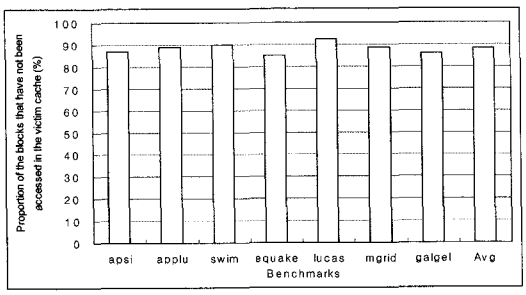


그림 3 실수 응용프로그램에서의 희생 캐쉬 내의 사용되지 않는 데이타의 비율

3. 선택적 희생 캐쉬를 사용하는 계층적 메모리 시스템

본 논문에서는 1차 캐쉬의 접근 패턴을 기반으로 하여 희생 캐쉬에서 발생하는 불필요한 데이타 교체를 줄이는 두 가지 방안을 제시한다. 참조 기반 희생 캐쉬(Hit-based Victim Cache)는 1차 캐쉬에서 교체되는 데이타 중에서 1차 캐쉬에 있는 동안 프로세서로부터 많은 접근이 이루어진 데이타를 선택하여 할당한다. 교체 기반 희생 캐쉬(Replacement-based Victim Cache)는 1차 캐쉬에서 충돌 미스가 자주 발생하는 셋에 할당되었던 데이타가 1차 캐쉬에서 교체되어질 때 희생 캐쉬에 할당한다. 이러한 기법들을 적용하여 선택적으로 희생 캐쉬에 할당되는 데이타가 프로세서에 의해 보다 높은 확률로 참조되는지 알아보고자 한다. 제안하는 구조를 통해 희생 캐쉬에서 발생되는 블럭 교체 횟수를 줄이고, 일차 레벨 메모리의 적중률을 높인다면 시스템의 성능 향상은 물론 전력 소모량의 감소 효과 또한 얻을 수 있을 것으로 기대된다.

3.1 참조 기반 희생 캐쉬

그림 4와 그림 5는 각각 기존의 1차 캐쉬 구조와 참조 기반 희생 캐쉬 시스템에서의 1차 캐쉬 구조를 보여주고 있다. 참조 기반 희생 캐쉬를 사용하는 시스템에서는 1차 캐쉬에 모드 비트(Mode Bit)와 전역 카운터(Global Counter)를 추가하고, 1차 캐쉬의 각 라인에는 참조 카운터(Hit Counter) 필드를 추가하였다.

2-비트로 구성되는 참조 카운터의 동작 방식은 그림 6과 같다. 동적 전력 소모량(Dynamic Energy Consumption)을 최소화하기 위해 상태 전이 시에 하나의 비트만 변화되도록 하기 위해 그레이 코딩 방식을 사용하였다. 1차 캐쉬에 데이타 블록이 할당될 때, 그 블록의 참조 카운터는 S0 상태로 초기화된다. 그 후, 해당 블록에 대한 프로세서의 접근 요청이 히트(Hit)로 판명되면 참조 카운터의 상태는 S1, S2, S3의 순서로 차례차례 변경된다. 1차 캐쉬에서 데이타 블록이 교체될 때, 해당 블록의 참조 카운터 상태를 봄으로써 프로세서로부터 자주 참조된 블록인지 여부를 판별할 수 있게 된다.

전역 카운터는 1차 캐쉬 내에 참조 카운터의 상태가 S3인 블록의 수를 계산하는 데 사용되고, 모드 비트는 전역 카운터의 값에 따라 설정되는 것으로 교체되는 블록의 교체 방식을 결정하는데 사용된다. 전역 카운터의 값이 임계값(Threshold) T 보다 작은 경우에 모드 비트의 상태는 0을 유지하고, 전역 카운터의 값이 임계값 T 이상이 되면 모드 비트의 상태는 1이 된다. 즉, 1차 캐쉬 내에 프로세서로부터 자주 참조되는 블록들의 수가 증가되면 모드 비트는 1이 된다.

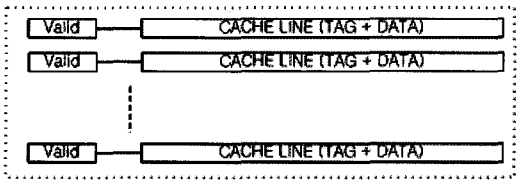


그림 4 일반적인 1차 캐쉬 구조

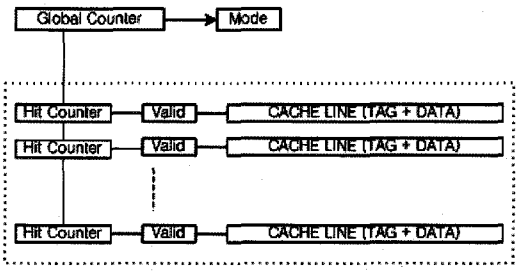


그림 5 참조 기반 희생 캐쉬 구조에서의 1차 캐쉬 구조

모드 비트가 0인 경우에는 기존의 희생 캐쉬를 사용하는 시스템과 같은 방식으로 1차 캐쉬에서 교체되는 모든 블록들은 참조 카운터의 상태에 관계없이 희생 캐쉬로 할당된다(그림 7(a)). 반면에, 모드 비트가 1인 상태에서는 1차 캐쉬에서 교체되는 블록들의 참조 카운터 상태를 보고 S3 상태에 있던 블록들은 희생 캐쉬로 할당하고, S0, S1, S2 상태에 있던 블록들은 희생 캐쉬로 할당하지 않고 바로 2차 캐쉬로 보낸다(그림 7(b)).

위의 방식을 통해 참조 기반 희생 캐쉬를 사용하는 시스템에서는, 1차 캐쉬 내에 프로세서로부터 자주 참조된 블록들이 많은 경우에는 희생 캐쉬를 그 블록들에 한정해서 사용한다. 그러므로, 프로세서로부터 자주 참조된 블록들이 자주 참조되지 않은 블록들에 비해 짧은 시간 내에 프로세서로부터 재참조되는 확률이 높은 응용 프로그램에서는 희생 캐쉬를 기존 시스템보다 효율적으로 사용할 수 있게 된다. 희생 캐쉬의 적중률이 높

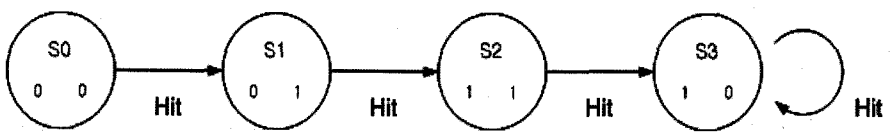
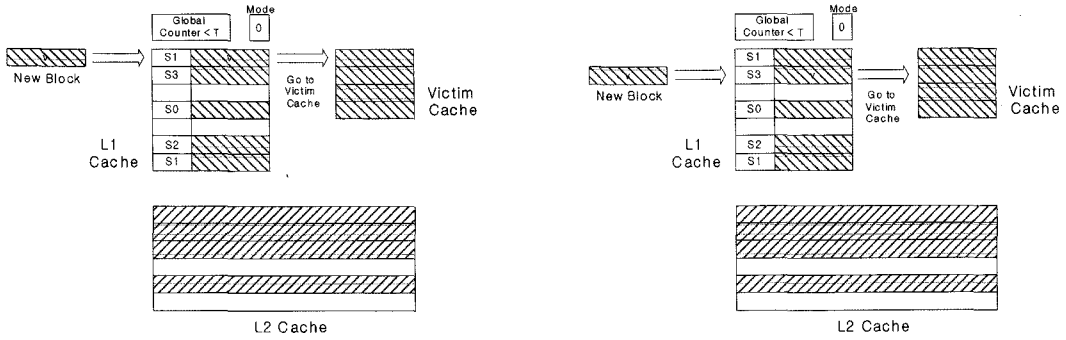
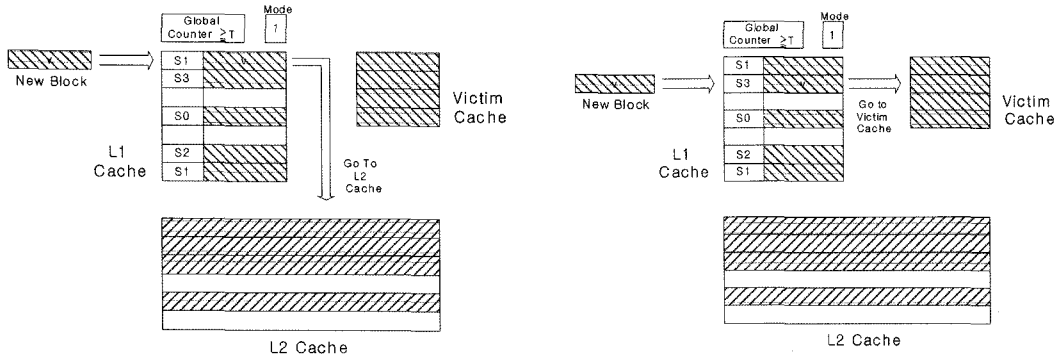


그림 6 참조 카운터의 State Diagram



(a) 모드 비트가 0일 경우 1차 캐쉬 블록의 교체



(b) 모드 비트가 1일 경우 1차 캐쉬 블록의 교체

그림 7 선택적 희생 캐쉬 구조에서의 블록 교체 방식

아지면, 일차 레벨 메모리의 적중률이 증가되면서 접근 시간이 길고 전력 소모가 많은 2차 캐쉬로의 접근 횟수를 줄일 수 있게 된다. 또한, 희생 캐쉬에서 발생되는 데이타 교체 횟수를 줄임으로써 전력 소모를 줄일 수도 있게 된다.

3.2 교체 기반 희생 캐쉬

교체 기반 희생 캐쉬를 사용하는 시스템에는 참조 기반 희생 캐쉬를 사용하는 시스템과 마찬가지로 1차 캐쉬에 모드 비트와 전역 카운터를 추가하고, 각각의 1차 캐쉬 셋에는 2-비트의 교체 카운터(Replacement Counter) 필드를 추가하였다. 변경된 1차 캐쉬 구조는 그림 8과 같다. 교체 기반 희생 캐쉬 시스템에서는 1차 캐쉬에서 상대적으로 교체가 자주 발생하는 셋에 할당되는 데이타 블록들을 위주로 희생 캐쉬를 사용한다. 직접 사상 1차 캐쉬의 특정 셋에서 다른 셋에 비해 상대적으로 많은 충돌 미스가 발생하고, 이 셋에 위치하는 블록들이 프로세서로부터 더 높은 확률로 재요청된다면, 이 구조는 일차 레벨 메모리의 적중률을 향상시킬 수 있을 것으로 기대된다.

교체 카운터의 상태 전이는 그림 9와 같이 이루어진다. 각 셋의 교체 카운터는 상태 S0로 초기화된다. 그

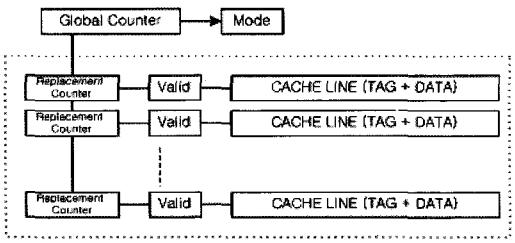


그림 8 교체 기반 희생 캐쉬 구조에서의 1차 캐쉬 구조

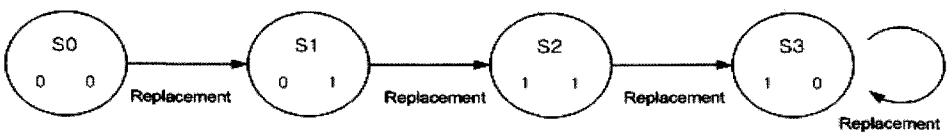


그림 9 교체 카운터의 State Diagram

후, 해당 셋에서 충돌 미스로 인해 데이타 블록의 교체가 발생하면 교체 카운터의 상태는 S1, S2, S3의 순서대로 변경된다. 교체 카운터를 각 셋에 추가함으로써 교체가 자주 발생하는 셋을 판별할 수 있게 된다.

전역 카운터는 참조 기반 희생 캐쉬 시스템에서와 마찬가지로 교체 카운터의 상태가 S3인 셋의 수를 계산하는데 사용된다. 모드 비트 또한 참조 기반 희생 캐쉬 시스템에서와 마찬가지로 1차 캐쉬의 데이타 교체 시 블럭의 처리 방식을 결정하는 데 사용된다. 모드 비트는 전역 카운터의 값이 임계값 T보다 작은 경우에는 0의 값을 유지하고, 전역 카운터의 값이 임계값 T 이상이 되면 1의 상태가 된다. 이 때, 교체 기반 희생 캐쉬 시스템에서 1차 캐쉬의 대부분의 셋의 교체 카운터 상태가 S3가 되어 기존의 희생 캐쉬 시스템과 유사한 동작이 이루어지는 것을 막기 위해, 전역 카운터의 값이 초기화값(Initialization Value)에 도달하면 모드 비트, 전역 카운터, 각 셋의 교체 카운터 상태를 모두 초기화시킨다. 초기화값은 다음 장에서 설명하는 모의 실험을 통하여 결정하였다.

교체 기반 희생 캐쉬 시스템의 동작 방식은 다음과 같다. 모드 비트가 0일 경우에는 기존의 희생 캐쉬 시스템과 마찬가지로 1차 캐쉬에서 교체되는 모든 데이타 블록을 교체 기반 희생 캐쉬에 할당한다(그림 7(a)). 모드 비트가 1이 되면, 1차 캐쉬에서 교체되는 데이타 블록 중에서 1차 캐쉬에서 위치하던 셋의 상태가 S3인 블록들은 희생 캐쉬로 보내고, S0, S1, S2 상태인 셋에서 교체된 블록들은 희생 캐쉬를 거치지 않고 바로 2차 캐쉬로 보낸다(그림 7(b)). 이 구조에서는 1차 캐쉬에 충돌 미스가 자주 발생하는 특정 셋들이 존재하는 경우, 희생 캐쉬는 그 셋들에 할당되는 데이타 블록에 한해서 할당을 받게 된다. 이 구조는 직접 사상 1차 캐쉬에서 충돌 미스가 모든 셋에 고르게 발생하지 않고 특정 셋에서 상대적으로 많이 발생하는 현상을 해결하기 위해 제안한다. 1차 캐쉬에서 충돌 미스가 발생할 확률이 높은 블록들에 대해서 희생 캐쉬를 사용하여, 이 블록들이 짧은 시간 내에 프로세서로부터 재참조되는 확률이 그렇지 않은 블록들과 비교하여 더 높은지 여부를 알아보고, 만약 이러한 블록들이 더 높은 확률로 재참조된다면 교체 기반 희생 캐쉬 시스템은 희생 캐쉬에서 발생하는

```
if( Global Counter < 임계값 )
   Mode = 0;
else
   Mode = 1;

if(Mode == 0){
   All evicted blocks from the L1 cache are loaded
   into victim cache;
}
else{
   Evicted block from the line, whose Hit Counter is
   S3, is loaded into victim cache;
   Evicted block from the line, whose Hit Counter is
   S0 or S1 or S2, is loaded into L2 cache
}
```

그림 10 참조 기반 희생 캐쉬 시스템에서의 희생 캐쉬 사용 방식

```
if( Global Counter < 임계값 )
  Mode = 0;
else
  Mode = 1;

if(Global Counter == 초기화값)
  Mode, Global Counter, all Replacement Counters are
  reset;

if(Mode == 0){
  All evicted blocks from the L1 cache are loaded into
  victim cache;
}
else{
  Evicted block from the line, whose Replacement
  Counter is S3, is loaded into victim cache;
  Evicted block from the line, whose Replacement
  Counter is S0 or S1 or S2, is loaded into L2 cache
}
```

그림 11 교체 기반 희생 캐쉬 시스템에서의 희생 캐쉬 사용 방식

표 1 희생 캐쉬 시스템 비교

	기존의 희생 캐쉬	참조 기반 희생 캐쉬	교체 기반 희생 캐쉬
희생 캐쉬 사용 방식	1차 캐쉬에서 교체되는 모든 데이타 무조건 할당	그림 10	그림 11
장 점	하드웨어가 단순 (부가적인 카운터 등이 불필요)	참조가 자주 이루어진 데이타의 재참조 확률이 높은 경우 희생 캐쉬의 적중률 증가 (데이타의 시간 지역성 활용)	교체가 자주 발생하는 셋에서의 충돌로 인한 캐쉬의 적중률 감소를 최소화 (직접 사상 캐쉬의 단점을 가장 잘 극복할 수 있음)
단 점	희생 캐쉬에 할당된 데이타 중 사용되지 않는 데이타의 비율이 높음 (그림 2)	기존의 희생 캐쉬에 비해 하드웨어가 복잡	기존의 희생 캐쉬에 비해 하드웨어가 복잡

데이타의 교체 횟수를 감소시킬 수 있을 뿐만 아니라 일차 레벨 메모리의 적중률을 높임으로써 시스템에서 소모되는 전력을 줄일 수 있게 된다.

4. 시스템 성능 모의실험 및 분석

4.1 모의실험 환경 및 인자

표 1에서 보이는 기존의 희생 캐쉬 시스템과 본 논문에서 제안된 두 가지 희생 캐쉬 시스템의 성능 및 전력 효율성을 분석하기 위한 모의실험 도구로는 SimpleScalar 시뮬레이터와 SimpleScalar을 기반으로 전력 소모량을 계산해주는 시뮬레이터인 Wattch를 사용한다[11,12]. SimpleScalar는 최신 프로세서 기술을 바탕으로 빠른 수행 기반 실험을 수행하는 도구(execution-driven simulator)로서, out-of-order issue 프로세서, non-blocking caches, speculative execution, 최신의 다양한 branch prediction 기법 등을 지원하고 있다. Wattch는 SimpleScalar의 동작을 기반으로 시스템의 전력 소모량을 계산한다. 본 실험에서는 전력 소모량을 측정하기 위해 0.18 um 기술을 가정하였다. 실험에 사용된 주요 실험인자들은 표 2와 같다.

실험의 입력 데이타로는 표 3에 기술된 7개의 SPEC CPU2000 정수 응용프로그램과 7개의 SPEC CPU2000 실수 응용프로그램을 사용한다[13]. 시뮬레이터의 입력으로는 SimpleScalar GCC 컴파일러를 통해 PISA 타입으로 컴파일된 코드를 사용하였고, 각 벤치마크 프로그램은 끝가지 수행되었다.

4.2 모의 실험 결과 및 분석

모의 실험을 통해 선택적 희생 캐쉬 시스템에서 사용되는 임계값과 교체 기반 희생 캐쉬 시스템의 초기화값을 결정하고, 결정된 값을 기반으로 하여 기존의 희생 캐쉬 시스템, 참조 기반 희생 캐쉬 시스템, 교체 기반 희생 캐쉬 시스템에 대해 미스율, 희생 캐쉬 내에서의

표 2 모의 실험 인자

실험 인자	값
Functional Units	4 integer ALUs, 4 FP ALUs, 1 integer multiplier/divider, 1 FP multiplier/divider
LSQ size	32 entries
ROB size	64 entries
Fetch Width	4 instructions/cycle
Decode Width	4 instructions/cycle
Issue Width	4 instructions/cycle
Commit Width	4 instructions/cycle
Branch Predictor	Combination
BTB	1024 entries, 2-way
Victim cache	8 entries, fully-associative, 32 byte blocks, 1 cycle latency
L1 i-cache, L1 d-cache	8KB ~ 16 KB, 1-way, 32 byte blocks, 1 cycle latency, write-back
L2 cache	256 KB unified, 4-way, 64 byte blocks, 8 cycle latency, write-back
Memory	100 cycle latency

표 3 벤치마크 프로그램

분류	벤치마크	설명	명령어 수
CINT2000	bzip2	Compression	8822143334
	twolf	Place and Route Simulator	258758690
	mcf	Combinatorial Optimization	259643444
	gzip	Compression	1661128319
	gcc	C Programming Language Compiler	2016631331
	crafty	Game Playing: Chess	4264781571
	parser	Word Processing	4203522039
CFP2000	apsi	Meteorology	5282090527
	applu	Parabolic/Elliptic Partial Differential Equations	182721153
	swim	Shallow Water Modeling	431764092
	equake	Seismic Wave Propagation Simulation	1443352221
	lucas	Number Theory/Primality Testing	3712064958
	mgrid	Multi-grid Solver: 3D Potential Field	16766602879
	galgel	Computational Fluid Dynamics	5282090527

교체 빈도, 시스템 성능, 전력 소모량을 비교해 보았다.

실험 결과에서, *Origin*은 기존의 희생 캐쉬를 사용하는 시스템을 의미하고 *Hit:nK*와 *Rep:nK*는 각각 참조 기반 희생 캐쉬를 사용하는 시스템과 교체 기반 희생 캐쉬를 사용하는 시스템을 의미한다. *nK*는 1차 캐쉬의 크기를 나타내는 값이다.

4.2.1 임계값(Threshold)

전역카운터의 임계값을 희생 캐쉬 엔트리 수(8)의 배수인 8, 16, 24, 32, 40으로 변경해가며 참조 기반 희생 캐쉬 시스템과 교체 기반 희생 캐쉬 시스템의 성능을 정수 응용 프로그램과 실수 응용 프로그램에 대해 측정한 결과는 그림 12와 그림 13에 나타낸 바와 같다. 아래 그림에서 나타내는 성능 향상률은 동일한 크기의 1차 캐쉬를 가지는 기존의 희생 캐쉬 시스템과 비교한 결과이다.

위의 그림에서 보는 바와 같이 전역 카운터의 임계값을 16으로 설정한 시스템이 다른 시스템들에 비해 보다 좋은 성능을 보이고 있다. 이는 전역 카운터의 임계값이 너무 작아지게 되면 희생 캐쉬를 사용할 수 있는 블록의 수가 많지 않은 상태에서 선택적으로 희생 캐쉬를 사용함으로써 희생 캐쉬의 효율성이 떨어지게 되고, 임계값이 너무 커지게 되면 카운터의 상태가 S3인 블록들의 수가 임계값에 도달하게 될 때까지 기다린 후에 선택적인 할당을 수행함으로써 상당 기간 동안 제안하는 구조가 기존의 희생 캐쉬 시스템과 같은 할당 방법을 수행하게 됨으로써 성능이 저하되는 것으로 분석된다. 따라서, 위의 결과를 바탕으로 선택적 희생 캐쉬 시스템에서의 임계값 T는 16으로 설정하였다.

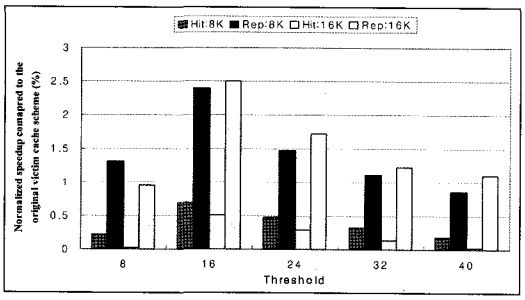


그림 12 정수 응용프로그램에서의 임계값에 따른 선택적 희생 캐쉬 시스템의 성능

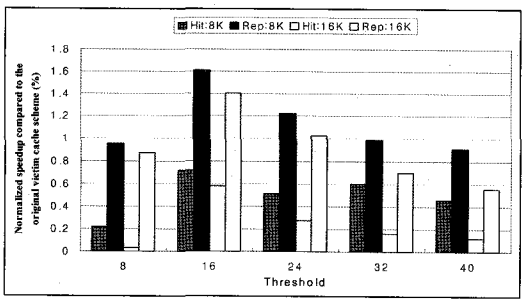


그림 13 실수 응용프로그램에서의 임계값에 따른 선택적 희생 캐쉬 시스템의 성능

4.2.2 교체 기반 희생 캐쉬 시스템의 초기화 값(Initialization Value)

교체 기반 희생 캐쉬 시스템에서 모드 비트, 전역 카운터, 각 셋의 교체 카운터 상태를 초기화시키는 값을 임계값 T의 배수에 해당하는 2T, 3T, 4T, 5T, 6T로 변경해 가며 각 시스템의 성능 향상을 비교한 결과는 그림 14와 같다. 그림에서 보이는 성능 향상률은 동일한 크기의 1차 캐쉬를 가지는 기존의 희생 캐쉬 시스템에 대한 상대적인 성능 향상률을 나타낸다. 그림 14에서 나타내는 바와 같이 모의 실험 결과 1차 캐쉬의 크기에 관계없이 모든 환경에서 초기화 값을 4T로 하는 시스템이 가장 좋은 성능을 보이고 있다. 이러한 결과는 초기화 값이 4T보다 작은 경우에는 너무 빈번한 초기화 작업을 유발하여 교체 기반 희생 캐쉬 시스템의 성능 저하가 발생되고, 초기화 값이 4T보다 큰 경우에는 너무 많은 셋의 교체 카운터 상태가 S3 상태로 유지되어 선택적 할당 기법의 효율성이 떨어지는 것으로 분석된다. 이러한 결과를 통해 교체 기반 희생 캐쉬 시스템에서 모드 비트, 전역 카운터, 교체 카운터들의 초기화는 전역 카운터의 값이 임계값의 4배가 되는 시점마다 이루어지게 하였다.

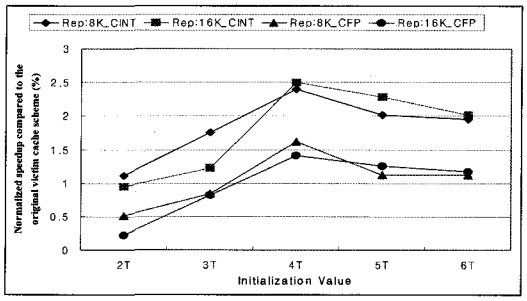


그림 14 교체 기반 희생 캐쉬 시스템의 초기화 값에 따른 성능 비교

4.2.3 미스율(Miss Rates)

1차 캐쉬의 크기를 8KB와 16KB로 변경해 가며 실험을 수행한 결과 일차 레벨 메모리의 미스율은 표 4와 같은 결과를 보인다. 표에서 나타내는 일차 레벨 메모리의 미스율은 프로세서가 요구하는 캐쉬 접근 중에서 1차 캐쉬와 희생 캐쉬에서 모두 미스가 발생하여 2차 캐쉬로 가는 요구의 비율을 나타낸다. 표에서 보는 바와 같이,

교체 기반 희생 캐쉬 시스템이 일차 레벨 메모리의 미스율 측면에서 상대적으로 좋은 성능을 보이고 있다. 참조 기반 희생 캐쉬 시스템 또한 기존의 희생 캐쉬를 사용하는 시스템에 비해서는 좋은 성능을 보이고 있다.

위의 결과를 통해, 교체가 자주 발생하는 셋에 할당되는 데이타 블록들이 짧은 시간 내에 프로세서로부터 다시 요청될 확률이 상대적으로 높다는 것을 알 수 있다. 프로세서로부터 많은 참조를 받고 교체된 블록 또한 그렇지 않은 블록들에 비해 상대적으로 높은 확률로 재참조된다는 사실도 실험 결과를 통해 알 수 있다.

교체 기반 희생 캐쉬 시스템은 기존의 희생 캐쉬 시스템과 비교하여 일차 레벨 메모리의 미스율을 평균적으로 1차 캐쉬의 크기가 8KB인 경우에는 0.9%(CINT), 1.1%(CFP) 감소시켰고, 1차 캐쉬의 크기가 16KB인 경우에는 0.8%(CINT, CFP) 감소시켰다. 참조 기반 희생 캐쉬 시스템은 1차 캐쉬의 크기가 8KB인 경우에는 평균 0.4%(CINT), 0.5%(CFP) 감소시켰고, 1차 캐쉬의 크기가 16KB인 경우에는 평균 0.2%(CINT), 0.3%(CFP)의 미스율을 감소시켰다. 직접 사상 1차 캐쉬에서 발생하는 충돌 미스의 수는 캐쉬의 크기가 작을수록 많아지므로, 제안하는 구조들은 1차 캐쉬의 크기가 작은 경우에 보다 많은 미스율의 감소 효과를 가져옴을 알 수 있다. 일차 레벨 메모리의 미스율 감소는 2차 캐쉬로의 접근 횟수를 줄이고, 이는 메모리 시스템의 성능을 향상시키고 전력 소모량을 줄이는 결과를 가져올 것으로 기대된다.

4.2.4 희생 캐쉬 내에서의 교체 빈도(Replacement Frequency in Victim Cache)

그림 15와 그림 16은 기존의 희생 캐쉬와 비교한 참조 기반 희생 캐쉬와 교체 기반 희생 캐쉬에서 발생하는 교체 빈도의 감소율을 보여주고 있다. 각 그래프에서 세로축은 동일한 크기의 1차 캐쉬와 동작하는 기존의 희생 캐쉬와 비교한 상대적인 교체 빈도의 감소율을 나

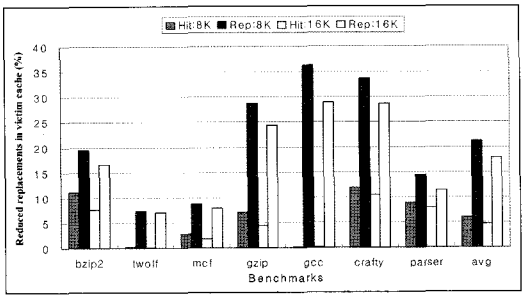


그림 15 정수 응용프로그램에서의 희생 캐쉬 내에서의 교체 빈도 감소율

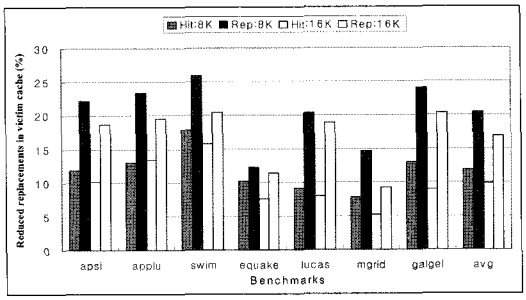


그림 16 실수 응용프로그램에서의 희생 캐쉬 내에서의 교체 빈도 감소율

표 4 일차 레벨 메모리의 미스율 비교

		미스율					
		Origin:8K	**Hit:8K**	**Rep:8K**	**Origin:16K**	**Hit:16K**	**Rep:16K**
CINT	bzip2	0.018	0.017	0.015	0.011	0.011	0.010
	twolf	0.041	0.032	0.029	0.028	0.020	0.016
	mcf	0.087	0.084	0.074	0.079	0.076	0.068
	gzip	0.033	0.032	0.026	0.027	0.026	0.019
	gcc	0.043	0.041	0.037	0.034	0.034	0.029
	crafty	0.067	0.059	0.050	0.045	0.039	0.033
	parser	0.031	0.031	0.025	0.029	0.029	0.022
	Avg	0.046	0.042	0.037	0.036	0.034	0.028
CFP	apsi	0.091	0.086	0.078	0.075	0.072	0.068
	applu	0.114	0.106	0.095	0.081	0.075	0.069
	swim	0.108	0.096	0.095	0.095	0.085	0.084
	equake	0.042	0.042	0.038	0.030	0.030	0.025
	lucas	0.039	0.036	0.029	0.036	0.032	0.028
	mgrid	0.072	0.067	0.065	0.054	0.052	0.049
	galgel	0.067	0.064	0.052	0.050	0.050	0.038
	Avg	0.076	0.071	0.065	0.060	0.057	0.052

타내고, 가로축은 실험에 사용된 벤치마크 프로그램들을 나타낸다. 1차 캐쉬에서 교체되어지는 블록들 중에서 선택적으로 희생 캐쉬에 할당하는 기법을 사용하므로 희생 캐쉬 내에서 발생되는 블록 교체 횟수는 상당 부분 감소한다.

1차 캐쉬의 크기에 관계없이 대부분의 응용 프로그램들에서 교체 기반 희생 캐쉬가 참조 기반 희생 캐쉬에 비해 보다 많은 수의 교체 횟수를 줄이는 것을 알 수 있다. 1차 캐쉬의 크기가 8KB인 경우 교체 기반 희생 캐쉬는 교체 빈도를 평균 21%(CINT), 20%(CFP) 감소시켰고, 참조 기반 희생 캐쉬는 교체 빈도를 평균 6%(CINT), 12%(CFP) 감소시켰다. 1차 캐쉬의 크기가 16KB인 경우에는 교체 기반 희생 캐쉬는 18%(CINT), 17%(CFP)의 교체 빈도를 감소시켰고, 참조 기반 희생 캐쉬는 5%(CINT), 10%(CFP)의 교체 빈도를 감소시켰다.

이 결과를 통해, 대부분의 응용 프로그램에서 1차 캐쉬의 충돌 미스로 인한 블록 교체가 특정 셋에 집중하여 상대적으로 많이 발생하고 있다는 사실을 알 수 있다. 희생 캐쉬를 이러한 셋에 할당되는 블록들에게 한정하여 사용함으로써 희생 캐쉬에서 발생하는 교체 빈도를 줄일 수 있게 된다. 프로세서로부터 많이 참조된 블록들로 한정하여 희생 캐쉬를 사용하게 하는 기법을 통해서도 희생 캐쉬 내에서의 교체 횟수를 감소시킬 수 있다. 희생 캐쉬 내에서 발생하는 교체 횟수를 줄이는 것은 메모리에서 소모되는 전력의 감소로 이어진다.

4.2.5 수행 시간(Execution Times)

그림 17과 그림 18은 제안하는 구조가 보이는 기존의 희생 캐쉬를 사용하는 시스템에 대한 상대적인 성능 향상을 나타내고 있다. 그림에서 보이는 결과를 통해 희생 캐쉬에서 발생하는 블록 교체 횟수를 줄이고, 보다 높은 확률로 프로세서로부터 재참조될 블록들을 위해 희생 캐쉬를 사용함으로써 시스템의 성능을 향상시킬 수 있음을 알 수 있다. 교체 기반 희생 캐쉬 시스템은 1차 캐쉬의 크기가 8KB인 경우에는 평균 2.39%(CINT), 1.62%(CFP)의 성능 향상률을 보이고, 1차 캐쉬의 크기가 16KB인 경우에는 평균 2.53%(CINT), 1.41%(CFP)의 성능 향상률을 보인다. 참조 기반 희생 캐쉬 시스템은 1차 캐쉬의 크기가 8KB인 경우에는 평균 0.69%(CINT), 0.72%(CFP)의 성능을 향상시키고, 1차 캐쉬의 크기가 16KB인 경우에는 평균 0.5%(CINT), 0.58%(CFP)의 성능을 향상시키고 있다. 1차 캐쉬의 크기에 관계없이 실험한 모든 응용 프로그램에서 교체 기반 희생 캐쉬를 사용하는 시스템이 가장 좋은 성능을 보이고 있다. 이러한 성능 향상은 2차 캐쉬로의 접근을 줄임으로 인한 평균 데이타 접근 시간의 감소를 통해 이루어진다.

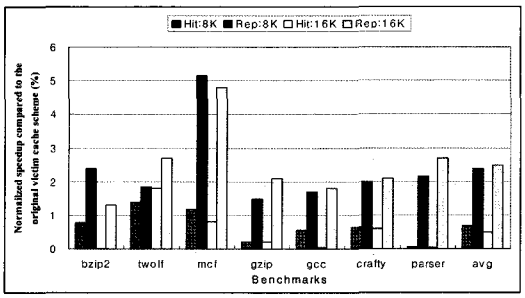


그림 17 정수 응용프로그램에서 기존의 시스템에 대한 상대적 성능 향상

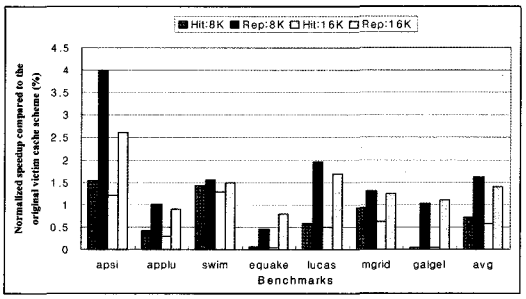


그림 18 실수 응용프로그램에서 기존의 시스템에 대한 상대적 성능 향상

4.2.6 전력 소모량(Energy Consumption)

이 절에서는 제안하는 시스템에서의 전력 소모량을 기존의 희생 캐쉬 시스템과 비교 분석한다. 본 논문에서 비교하는 총 전력 소모량은 1차 캐쉬, 희생 캐쉬, 2차 캐쉬의 전력 소모량의 합으로 계산한다. 선택적 희생 캐쉬를 사용하는 시스템에서 기존의 시스템에 비해 추가적으로 요구되는 참조 카운터, 교체 카운터를 유지하는데 필요한 전력 소모량 또한 총 전력 소모량에 포함되었다. 기존 시스템과 비교한 전력 소모량의 감소율은 그림 19와 그림 20에서 보여주고 있다.

일차 레벨 메모리의 적중률을 높임으로써 2차 캐쉬로의 블록 접근을 줄이는 결과를 통해 메모리 시스템의 전력 소모량을 감소시킬수 있었다. 또한, 희생 캐쉬에서 발생되는 블록 교체 횟수를 줄임으로 인해 메모리 시스템에서 소모되는 전력을 줄일 수 있다. 이는 일차 레벨 메모리의 적중률이 비슷한 parser와 같은 응용프로그램에서도 실험 결과, 참조 기반 희생 캐쉬 시스템이 기존의 시스템에 비해 전력 소모량을 줄이고 있는 결과를 통해 알 수 있다.

1차 캐쉬의 크기가 8KB인 경우, 참조 기반 희생 캐쉬 시스템은 1.02%(CINT), 2.81%(CFP)의 전력 소모를 감소시키고, 교체 기반 희생 캐쉬 시스템은 2.55%

(CINT), 4.71%(CFP)의 전력 소모를 감소시켰다. 1차 캐쉬의 크기가 16KB인 경우에는, 참조 기반 희생 캐쉬 시스템은 0.67%(CINT), 2.53%(CFP)의 전력 소모를 감소시켰고, 교체 기반 희생 캐쉬 시스템은 2.69%(CINT), 4.22%(CFP)의 전력 소모를 감소시켰다. 실험 결과, 전력 소모 측면에서도 교체 기반 희생 캐쉬 시스템이 가장 효율적임을 알 수 있다.

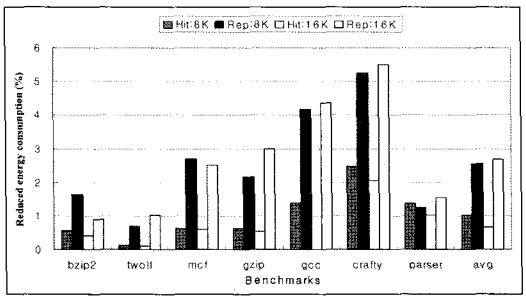


그림 19 정수 응용프로그램에서 기존의 시스템에 대한 상대적 전력 소모 감소율

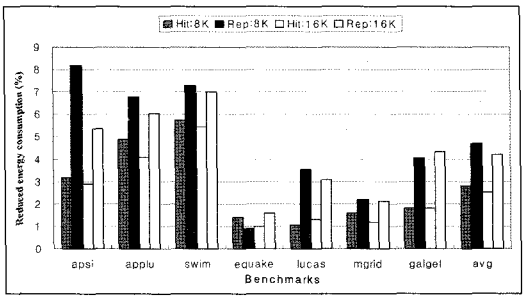


그림 20 실수 응용프로그램에서 기존의 시스템에 대한 상대적 전력 소모 감소율

5. 결 론

계층적 메모리 구조를 사용하는 시스템에서 일차 레벨 메모리의 적중률은 전체 시스템의 성능을 결정하는 중요한 요소로, 이를 높임으로써 시스템의 성능 향상은 물론 시스템에서 소모되는 전력량을 감소시킬 수 있다.

본 논문에서는 계층적 메모리 구조를 사용하는 시스템에서 일차 레벨 메모리의 적중률을 증가시키기 위해서 1차 캐쉬의 접근 패턴을 기반으로 희생 캐쉬로의 블록 할당을 선택적으로 수행하는 두 가지 기법을 제시하였다. 참조 기반 희생 캐쉬는 프로세서로부터 많은 참조를 받은 후에 교체되는 블록들로 희생 캐쉬를 채우는 방식을 사용하였고, 교체 기반 희생 캐쉬는 1차 캐쉬에서 블록 교체가 빈번하게 발생하는 셋에 할당되었던 블록들로 희생 캐쉬를 채우는 방식을 사용하였다. 실험 결과, 비교한 세 가지 메모리 시스템 중에서 교체 기반 희생 캐쉬를 사용하는 시스템이 가장 효율적인 시스템이라는 결론을 얻을 수 있었다. 일차 레벨 메모리의 적중률을 향상시킴으로 인해 긴 접근 시간과 많은 전력을 소모하는 2차 캐쉬와 같은 하위 메모리 구조로의 데이타 접근을 줄일 수 있었고, 희생 캐쉬에서 발생하는 데이타 블럭의 교체 횟수 또한 감소시켰다.

참 고 문 헌

[1] J. L. Hennessy and D. A. Patterson, Computer Architecture: A Quantitative Approach, Second Edition, Morgan Kaufmann, 1996.

[2] N. P. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," In Proceedings of the 17th Annual International Symposium on Computer Architecture, pp. 248-259, 1998.

[3] A. Agarwal and S.D. Pudar, "Column-Associative Caches: A Technique for Reducing the Miss Rate of Direct Mapped Caches," Proceedings of the 20th Int'l Symposium on Computer Architecture, pp. 179-190, May 1993.

[4] O. Temam and N. Drach, "Software Assistance for Data Caches," Proceedings of the 1st Int'l Symposium on High-Performance Computer Architecture, pp. 154, Jan. 1995.

[5] Jude A. Rivers and Edward S. Davidson, "Reducing Conflicts In Direct-mapped Caches with a Temporality-based Design," Proceedings of the 1996 International Conference on Parallel Processing, pp.. 151-162, Aug. 1996.

[6] T. Johnson and W.W. Hwu, "Run-Time Adaptive Cache Hierarchy management via Reference Analysis," Proc. ISCA-24, 1997.

[7] David H. Albonesi, "Selective cache ways: On-Demand Cache Resource Allocation," In Proceedings of Int'l Symposium of Microarchitecture, pp. 248-259, Nov. 1999.

[8] Kin, J., M. Gupta, and W. H. Mangione-Smith, "The Filter Cache: an energy efficient memory structure," In Proceedings of Int'l Symposium on Microarchitecture, pp. 184-193, Dec. 2001.

[9] Dimitrios Stiliadis, Anujan Varma, "Selective Victim Caching: A Method to Improve the Performance of Direct-Mapped Caches," IEEE Transactions on Computers, Vol 46, No 5, pp. 603-610, 1997.

[10] Gokhan Memik, Glenn Reinman, and William H. Mangione-Smith, "Reducing Energy and Delay Using Efficient Victim Caches," In Proceedings of Int'l Symposium on Low Power Electronics and Design, Aug. 2003.

[11] D. Burger, T. M. Austin, and S. Bennett,

"Evaluating future micro-processors: the Simple-Scalar tool set," Tech. Report TR-1308, Univ. of Wisconsin-Madison Computer Sciences Dept., 1997.

[12] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," In Proceedings of the 27th Annual International Symposium on Computer Architecture, pp. 83-94, June 2000.

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

김 철 홍

1998년 2월 서울대학교 컴퓨터공학과 공학사. 2000년 2월 서울대학교 대학원 컴퓨터공학부 석사. 2000년 3월~현재 서울대학교 대학원 전기컴퓨터공학부 박사과정. 관심분야는 컴퓨터 구조, 저전력 시스템, 내장형 시스템, 병렬 처리

심 성 훈

1997년 2월 동국대학교 전자계산학과 공학사. 1999년 2월 서울대학교 대학원 컴퓨터공학부 석사. 2000년 3월~현재 서울대학교 대학원 전기컴퓨터공학부 박사과정. 관심분야는 컴퓨터 구조, 저전력 시스템, 병렬처리 시스템

전 주 식

정보과학회논문지 : 시스템 및 이론
제 32 권 제 5 호 참조

장 성 태

1986년 2월 서울대학교 전자계산기공학과 공학사. 1988년 2월 서울대학교 대학원 컴퓨터공학과 석사. 1994년 2월 서울대학교 대학원 컴퓨터공학과 박사. 1994년 3월~현재 수원대학교 IT대학 컴퓨터학과 부교수. 관심분야는 다중 프로세서 시스템, 병렬 처리, 캐쉬 구조, 저전력 임베디드 프로세서 설계, 모바일 시스템