DOI: 10.3745/KIPSTA.2009.16-A.2.69

내장형 시스템을 위한 선택적 뱅크 알고리즘을 이용한 데이터 캐쉬 시스템

정 보 성[†] · 이 정 훈[††]

요 약

캐쉬의 성능을 향상시키는 가장 효과적인 방법은 프로그램 수행 특성에 내재되어 있는 시간적 (temporal locality) 지역성과 공간적 지역성 (spatial locality)을 활용하는 것이다. 본 논문은 프로그램 수행 특성에 적합한 시간적/공간적 지역성을 이용하기 위한 뱅크 선택 메커니즘을 가진 고성능 저전력 캐쉬 구조를 제안하였다. 제안하는 캐쉬 시스템은 다른 블록 크기와 다른 연관도를 가지는 두개의 캐쉬로 구성되어 진다. 즉 작은 블록 크기를 지원하는 직접사상 구조의 주 캐쉬(main direct-mapped cache)와 큰 블록을 지원하는 완전연관 버퍼 (fully associative buffer)로 구성되어 진다. 특히 주 캐쉬는 저전력을 위해 2-뱅크로 구성되며, 완전연관 버퍼에서 선택되어진 작은 블록은 제안된 뱅크 선택 알고리즘에 의해 주 캐쉬의 뱅크에 저장된다. 제안된 뱅크 선택 알고리즘과 3비트 상태 비트를 이용하여 시간적 지역성이 높은 데이터들을 주 캐쉬에 선택적으로 저장함으로써 고성능의 효과를 얻을 수 있었다. 제안된 알고리즘은 또한 충돌 미스 (conflict miss)와 캐쉬 오염 (cache pollution)을 효과적으로 줄여준다. 시뮬레이션 결과에 따르면, 평균 접근 실패율의 경우 Mibench 응용군에 대해 Victim 캐쉬에 비해 23%, STAS 캐쉬에 비해 32%의 감소효과를 보여준다. 평균 메모리 접근 시간의 경우 Victim 캐쉬에 비해 14%, STAS 캐쉬에 비해 18%의 감소효과를 얻을 수 있었다. 에너지 소비의 관점에서도 제안된 캐쉬 시스템은 Victim 캐쉬와 STAS 캐쉬에 비해 약 10% 감소 효과를 얻을 수 있었다.

키워드 : 시간적-공간적 지역성, 데이터 캐쉬, 이중 캐쉬, 저전력, 고성능

Data Cache System based on the Selective Bank Algorithm for Embedded System

Bo-Sung Jung[†] · Jung-Hoon Lee[††]

ABSTRACT

One of the most effective way to improve cache performance is to exploit both temporal and spatial locality given by any program executive characteristics. In this paper we present a high performance and low power cache structure with a bank selection mechanism that enhances exploitation of spatial and temporal locality. The proposed cache system consists of two parts, i.e., a main direct-mapped cache with a small block size and a fully associative buffer with a large block size as a multiple of the small block size. Especially, the main direct-mapped cache is constructed as two banks for low power consumption and stores a small block which is selected from fully associative buffer by the proposed bank selection algorithm. By using the bank selection algorithm and three state bits, We selectively extend the lifetime of those small blocks with high temporal locality by storing them in the main direct-mapped caches. This approach effectively reduces conflict misses and cache pollution at the same time. According to the simulation results, the average miss ratio, compared with the Victim and STAS caches with the same size, is improved by about 23% and 32% for Mibench applications respectively. The average memory access time is reduced by about 14% and 18% compared with the he victim and STAS caches respectively. It is also shown that energy consumption of the proposed cache is around 10% lower than other cache systems that we examine.

Keywords : Temporal-spatial Locality, Data Cache, Dual Cache, Low Power, High Performance

1. 서 론

오늘날 컴퓨팅 환경에서 캐쉬 메모리는 내장형 시스템에서 범용 프로세서와 동일하게 고속의 프로세서와 저속의 주 메모리 사이에서 메모리 접근 지연시간과 에너지 소비를 줄이고 전체 성능 향상을 위한 중요한 역할을 담당하고 있다. 일반적으로 범용 프로세서의 캐쉬 성능은 캐쉬 용량을 증대시켜 캐쉬 적중률을 높임으로써 가능하다. 그러나 내장형 시스템에서 캐쉬 용량의 증대는 전체적으로 시스템에서의 용량, 전력 소모, 또한 비용의 증가를 수반한다. [1] 특히 메

※ 이 논문은 2008년도 경상국립대학교 국외파견 지원에 의하여 연구되었음.
† 준 회 원 : 경상국립대학교 제어계측공학과 박사과정
†† 정 회 원 : 경상국립대학교 전기전자공학부 공학연구원 조교수(교신저자)
논문접수 : 2008년 9월 3일
수 정 일 : 1차 2009년 2월 9일
심사완료 : 2009년 2월 9일

모리 시스템에서의 전력 소모는 StrongARM 110 [2] 에서 캐쉬가 전체 전력 소모의 약 42%를 차지한다는 연구 결과를 고려하였을 때 저전력/고성능 캐쉬 설계의 중요성은 더욱 증대 된다. 그러나 작은 용량의 캐쉬 메모리는 높은 적중 실패율을 야기하여 성능 저하를 가져 올 뿐 아니라 전력 소모의 증대를 가져오는 문제점을 가져 올 수 있다.

현재 캐쉬 메모리의 성능 향상을 위한 효과적인 방법 중 하나가 프로그램 수행 특성에 대한 두 가지 지역성을 이용하는 것이다. 시간적 지역성은 최근에 참조가 일어난 블록이 가까운 시간 내에 다시 참조될 확률이 높다는 것을 의미하며, 공간적 지역성은 참조가 일어난 블록의 이웃 블록이 참조될 확률이 높다는 것을 의미한다. 그러나 이 두 지역성은 고정된 캐쉬 크기에서 서로 상반된 특성을 보인다. [3] 즉 캐쉬 블록 크기가 커질수록 공간적 지역성은 효과적으로 반영할 수 있지만 캐쉬 엔트리 수의 감소로 시간적 지역성에 대한 효과는 감소된다. 따라서 단일 캐쉬 구조는 이러한 프로그램 수행 특성에 적합한 지역성을 이용하기가 부적합하다.

본 논문에서는 내장형 시스템의 프로그램 수행 특성에 적합한 지역성을 효과적으로 이용하기 위해 두 개의 분리된 캐쉬 구조로 설계하였다. 작은 블록을 가지는 주 캐쉬는 시간적 지역성에 효과적으로 반영 할 수 있으며, 큰 블록을 가지는 버퍼는 공간적 지역성을 효과적으로 반영 할 수 있다. 그리고 주 캐쉬는 낮은 에너지 소비를 위하여 뱅크로 구성된다. 하지만 뱅크 구성은 낮은 소비 에너지와 빠른 메모리 접근 시간을 보이지만 일반 직접사상 캐쉬에 비해 높은 메모리 참조 실패율을 보인다. 제안된 캐쉬 시스템은 주 캐쉬에서 이러한 높은 참조 실패율을 피하기 위해 제안된 알고리즘에 의해 어떠한 조건에서 주 캐쉬의 모든 뱅크에 한 뱅크에 대한 데이터 값을 저장하도록 하였다. 즉 주 캐쉬의 한 뱅크에 저장될 데이터가 다른 뱅크에 저장 가능하다고 판단되면 그 데이터를 다른 뱅크에 저장하였다. 이러한 방법으로 제안된 캐쉬 시스템은 블록 충돌에 의한 메모리 참조 실패와 캐쉬 오염을 줄여 전체 성능 향상을 얻을 수 있었다. 시뮬레이션 결과에 따르면 제안된 캐쉬 구조는 내장형 시스템에서 기존의 직접사상 캐쉬, Victim 캐쉬, 그리고 STAS 캐쉬에 비해 각각 17%, 23% 및 32% 메모리 접근 실패율을 보이며, 에너지 소비 역시 평균 약 10%의 감소를 보이고 있다.

본 논문의 구성은 다음과 같다. 관련 연구는 2장에서 소개되어지며, 3장은 제안된 캐쉬 시스템의 구조와 동작에 대해 설명한다. 4장에서는 성능 평가 지표와 소비 전력에 대한 실험 결과를 분석한다. 마지막으로 5장에서는 결론을 맺는다.

2. 관련 연구

기본적으로 캐쉬 메모리의 성능 향상을 위해 많은 연구가 이루어졌다. 연관 캐쉬 [4]는 n개의 직접사상 캐쉬를 이용하여 LRU 정책에 의해 많은 충돌 미스를 줄임으로써 성능 향상을 높였다. 하지만 느린 메모리 접근 시간과 많은 에너지를 요구하는 단점을 보이고 있다. 직접사상 캐쉬는 가장 큰 단점은 충돌 미스가 대단히 높다. 이에 충돌 미스를 줄이기 위한 효과적인 캐쉬 중 하나가 Victim 캐쉬 [5]이다. Victim 캐쉬는 주 캐쉬인 직접사상 캐쉬과 완전연관 버퍼인 희생(Victim) 버퍼로 구성된다. 직접사상 캐쉬에서 버려진 데이터는 희생 버퍼에 저장되어지며, 희생 버퍼는 대치 정책으로 FIFO (First In First Out)로 구동되어진다. 만약 희생 버퍼에서 참조 성공이 일어나면 직접사상 캐쉬와 스왑(Swap) 동작이 일어나며, 이때 추가적인 한 사이클과 에너지가 소비되어지는 단점을 보이고 있다. 또한 Zhang이 제안한 Victim 버퍼 [6]는 1비트 레지스터의 값으로 Victim 버퍼의 접근을 ON/OFF 하므로 기존의 Victim 캐쉬의 파워 소비를 줄이는 연구를 수행 하였다. 명령과 데이터가 통합된 L1캐쉬에 대해 각각의 Victim 버퍼를 사용하므로 저 전력뿐 아니라 성능향상을 추구하였다. 이렇듯 충돌 미스에 대해 좋은 성능개선을 보이는 Victim 캐쉬를 응용한 많은 연구가 진행되고 있다.

Langen [7]는 2개의 직접 사상 캐쉬와 충돌 예측 테이블(CDT)를 이용하여 충돌 미스를 줄임으로써 성능향상을 이루었다. CDT에 의해 현재 데이터가 충돌 미스라면 충돌 미스를 위한 직접사상 캐쉬(SCC :Sub-block in Case of Conflict cache)에 데이터를 올리며, 만약 미스중 충돌 미스가 아닌 참조 실패인 경우 우회 직접사상 캐쉬(BCC :Bypass in Case of Conflict cache)에 저장하므로 충돌 미스율을 크게 줄였다. 충돌 예측 테이블은 기본적으로 명령어와 데이터 태그를 저장하므로 충돌 미스를 결정한다.

이러한 충돌 미스를 구조적인 방법으로 피하여 성능을 향상 시키는 방법 외 성능 향상을 이루는 효과적인 방법 중 하나가 바로 프로그램 수행 특성에 적합한 시간적-공간적 지역성을 이용하는 것이다.

Selective 캐쉬 [8]는 큰 블록 크기를 가지는 공간적 캐쉬(spatial cache)와 작은 블록 크기를 가지는 시간적 캐쉬(temporal cache) 그리고 지역성 예측 테이블 (locality prediction table)로 구성되어있다. 만약 요구된 데이터가 시간적 캐쉬와 공간적 캐쉬에 없다면 참조 실패가 발생되며, 이때 하위 레벨에서 가져오는 데이터는 지역성 예측 테이블에 의해 시간적-공간적 캐쉬에 저장되어진다. 그러나 어떤 지역성도 가지지 않는다고 판단될 경우 이러한 데이터는 캐쉬에 저장되지 못하며 지역성 예측 테이블에 그 정보만을 저장시킨다.

UDDC (Unified Dual Data Cache) [9]는 Selective 캐쉬 [8]에서 시간적-공간적 지역성 캐쉬들을 하나의 캐쉬에 통합하여 프로그램 수행 특성에 적합한 지역성을 가진다. UDDC는 하나의 블록이 n개의 작은 블록들로 나누어지며, 참조시 해당하는 블록의 태그가 일치하면 작은 블록에서 데이터를 가지고 온다. 참조 실패시 예측 테이블에 시간적-공간적 지역성 판별 후 데이터 저장을 위한 동작을 하게 된다. 지역성 예측 테이블은 Selective 캐쉬와 동일하게 동작되며, 단 예측 테이블에 데이터가 없다면 이 데이터는 시간적 지역성으로 판단하여 동작된다.

The Non-Temporal Streaming (NTS) 캐쉬 [10]는 기존의 직접사상 캐쉬와 완전연관 버퍼를 병렬로 추가시켜 제안한 캐쉬 구조이다. NTS 캐쉬의 알고리즘은 시간적, 비시간적 블록 선택 데이터를 참조 흐름에 따라 나누는 것을 제안하였다. 요청되어진 데이터는 시간적 지역성을 보일 때까지 비 시간적으로 취급한다.

STAS 캐쉬 [11]는 간단한 하드웨어 구조적 특성으로 프로그램 수행 특성에 적합한 지역성을 효과적으로 이용한 캐쉬이다. 시간적 지역성을 위해 작은 블록을 가지는 직접사상 캐쉬와 공간적 지역성을 위해 큰 블록 크기를 가지는 완전연관 버퍼로 구성된다. 버퍼의 데이터 부분은 직접사상 캐쉬와 동일한 크기를 가지는 n개의 뱅크로 이루어지며, 참조가 발생한 작은 블록은 대치되어질 때 직접사상 캐쉬에 저장하게 된다.

Naz [12]는 시간적, 공간적 지역성을 사용하기 위해 Victim 캐쉬, Stream 버퍼 그리고 Split 캐쉬를 사용하였다. Victim 캐쉬는 충돌 미스에 좋은 성능 개선을 보이므로 시간적 지역성을 위해 사용되고, Stream 버퍼는 순차적인 데이터를 페칭하여 저장 하므로 공간적 지역성에 효과적으로 사용할 수 있다. 즉 Split 캐쉬 구조를 Victim 캐쉬와 Stream 버퍼로 응용하므로 시간적, 공간적 지역성을 이용한 캐쉬이다.

상위 연구된 캐쉬 시스템과 제안된 캐쉬 시스템의 차이점은 다음과 같다. 블록 충돌 참조 실패를 줄여 좋은 성능을 보이는 연관 캐쉬는 단일 구조 캐쉬로 프로그램 수행 특성에 적합한 시간적-공간적 지역성에 적합하지 않다. 하지만 UDDC는 단일 캐쉬 구조지만 데이터 부분을 작은 블록으로 나누어 시간적-공간적 지역성을 반영하였다. 하지만 시간적-공간적 지역성을 위한 예측 테이블이 사용되어진다.

제안된 캐쉬와 같이 이중 구조를 가진 캐쉬 시스템에서 Victim 캐쉬는 직접사상 캐쉬와 완전연관 버퍼로 다른 연관 캐쉬를 사용하지만, 모두 동일한 블록 크기를 가지므로 블록 크기에 따라 특정 지역성에만 효과적이다. NTS 캐쉬 역시 다른 연관 캐쉬 구조를 가지지만 동일한 블록 크기를 가진다. 하지만 NTS 캐쉬는 단지 시간적 지역성에 대한 효과 개선을 보이고 있다.

STAS 캐쉬와 Selective 캐쉬는 모두 시간적 지역성과 공간적 지역성을 위해 서로 다른 블록 크기의 이중 캐쉬로 구성되어진다. 하지만 Selective 캐쉬는 STAS 캐쉬, NTS 캐쉬, 그리고 Victim 캐쉬와 달리 동일한 연관 캐쉬로 구성되어진다. 그리고 공간적 지역성과 시간적 지역성을 위한 예측 테이블을 이용한다. 하지만 STAS 캐쉬는 서로 다른 연관 캐쉬와 블록을 사용하며, 시간적-공간적 지역성은 시간 간격에서 하드웨어 제어 메커니즘에 의해 결정하게 된다. 즉 공간적 지역성을 위한 버퍼에서 참조가 된 작은 블록이 시간적 지역성을 판단하여 주 캐쉬에 저장하므로 시간적-공간적 지역성을 효과적으로 반영하였다. 그러나 STAS 캐쉬는 버퍼에서의 대치 알고리즘 (FIFO)에 의해 무조건 주 캐쉬에 선택되어진 작은 블록을 저장하게 된다. 이러한 방법은 주 캐쉬의 데이터가 완전연관 버퍼의 작은 블록보다 최근 데이터 일 경우 캐쉬 오염의 원인이 된다.

제안된 캐쉬 시스템은 기본적으로 STAS 캐쉬와 동일한 구조를 가진다. 즉 서로 다른 연관 캐쉬와 다른 블록 크기로 추가 적인 예측 테이블 없이 시간 간격에 의해 두 지역성을 선택하여 사용한다. 특히 주 캐쉬를 뱅크로 구성하여 저전력뿐 아니라 제안된 뱅크 선택 알고리즘으로 특정 조건에서 주 캐쉬의 어떤 뱅크 데이터를 모든 뱅크에 저장하므로 STAS 캐쉬에서 발생할 수 있는 캐쉬 오염을 줄임으로써 성능 향상을 이루었다.

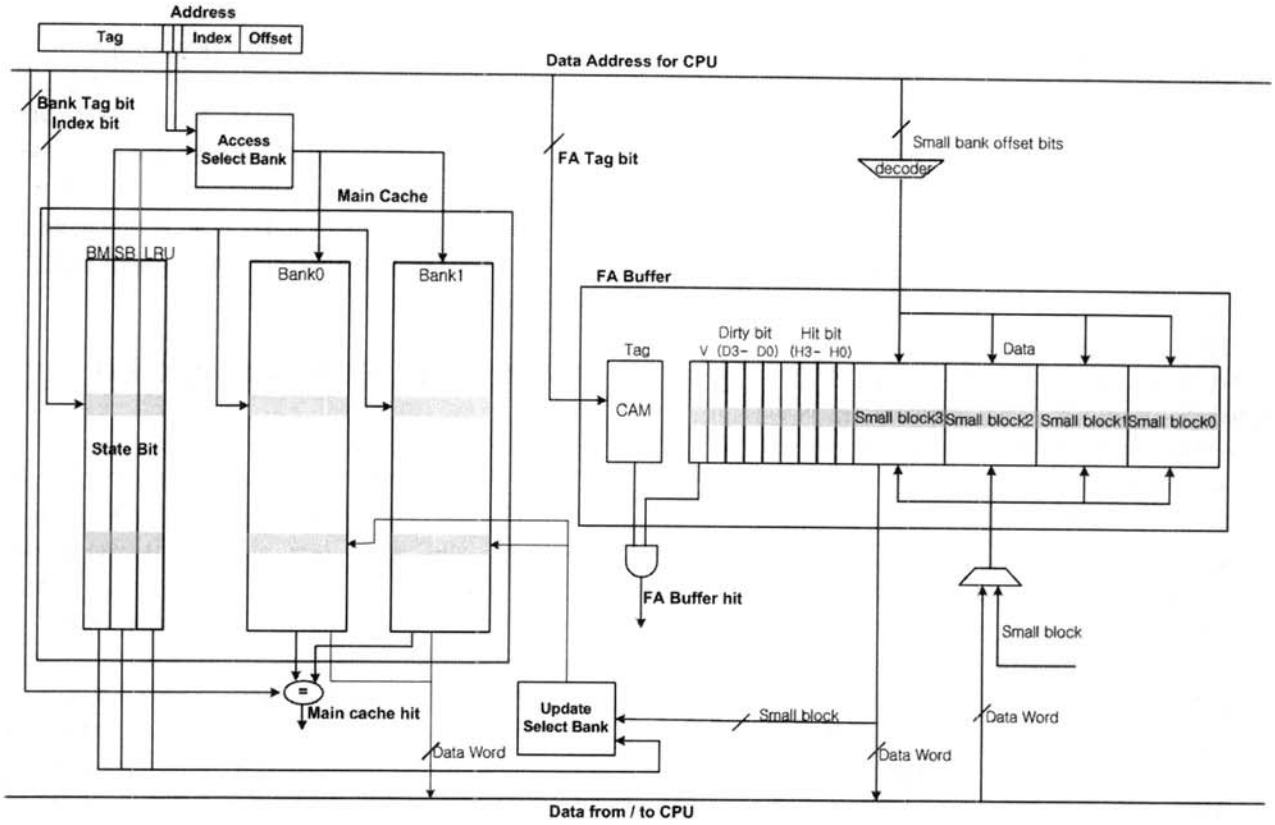
3. 제안된 캐쉬의 구조적 특징과 동작 원리

3.1 캐쉬 메모리 구조 및 동작

제안된 캐쉬 시스템의 주요 목적은 프로그램 수행 특성에 적합한 시간적-공간적 지역성을 이용하여 낮은 참조 실패율과 에너지 소비에 중점을 두고 있다. 따라서 제안된 캐쉬 시스템은 기본적으로 상반된 특성을 가진 두 종류의 지역성을 효과적으로 이용하기 위하여 두 개의 분리된 캐쉬 구조를 가진다.

본 논문에서 제안된 캐쉬 시스템의 구조는 (그림 1)과 같다. 제안된 캐쉬 시스템의 구조는 크게 시간적 지역성을 위한 작은 블록 크기를 가지는 주 뱅크 캐쉬와 공간적 지역성을 위한 큰 블록 크기를 가지는 버퍼 그리고 주 캐쉬의 상태를 나타내는 3비트의 상태 비트로 구성된다.

작은 블록을 가지는 주 캐쉬는 낮은 에너지 소비를 위해 2_뱅크 구조로 설계하였다. 버퍼는 작은 용량으로 접근 실패율이 낮은 완전연관 버퍼로 구성되어지며, 주 캐쉬와 동일한 계층에 위치하게 된다. 주 캐쉬의 각 뱅크들은 일반적인 직접사상 캐쉬와 동일한 구조로써 n-byte를 가지는 m개의 블록으로 구성된다. 완전연관 버퍼의 데이터 저장 공간은 여러 개의 작은 블록들로 구성되며, 각 블록의 크기는 주 캐쉬의 데이터 블록 크기와 동일하다. 즉 완전연관 버퍼의 데이터 뱅크 상수 c로부터 c*n byte, k개의 큰 블록으로 구성된다. 따라서 완전연관 버퍼의 총 바이트는 k*c*n byte이다. 그리고 버퍼의 태그 부분은 모든 엔트리를 동시에 비교 할 수 있는 CAM (Content Addressable Memory)으로 구현되어진다. 또한 완전연관 버퍼의 데이터 뱅크에 저장되어지는 블록들은 1비트를 가지고 있다. 이러한 1비트는 완전연관 버퍼의 작은 블록들에 대해 참조가 일어난 작은 블록을 나타내기 위한 적중 비트 (Hit bit)로 사용되어진다. 즉 완전연관 버퍼에서 메모리 참조 성공이 일어나면 해당하는 작은 블록에 대해 적중 비트를 갱신하게 된다. 주 캐쉬의 상태 비트는 각 뱅크들의 LRU (Last Recently Used)비트와 뱅크 선택을 위한 2비트, 총 3비트의 상태 비트로 구성된다. 2비트의 상태 비트들은 뱅크의 현재 접근 가능한 상태를 나타내는 BM비트 (Bank Mode bit)와 BM의 상태 변경시 접근 가능한 뱅크 비트를 나타내는 SB비트 (Second Bank bit)로 나누어진다. LRU비트는 주 캐쉬의 각 뱅크중 참조 성공이 발생했을 경우와 완전연관 버퍼에서 주 캐쉬로 데이



(그림 1) 제안된 캐쉬 시스템 구조

터 이동이 이루어 질 때 갱신이 이루어진다. 그리고 주 캐쉬의 뱅크 접근과 상태를 나타내는 2비트의 상태 비트(BM, SB)는 오직 완전연관 버퍼에서 주 캐쉬로 데이터 이동시 갱신이 이루어진다.

시간적 지역성을 가지는 주 캐쉬의 데이터는 공간적 지역성에 효과적인 큰 블록 크기를 가지는 완전 연관 버퍼에서 참조가 발생한 작은 블록의 데이터를 저장하게 된다. 이때 저장될 완전연관 버퍼의 작은 블록은 뱅크 비트들과 제안된 상태 비트의 조합으로 주 캐쉬의 뱅크를 결정하게 된다. 즉 완전연관 버퍼에서 선택되어진 작은 블록은 주 캐쉬의 어떤 뱅크에 저장되기 위해 뱅크 비트가 결정된다. 이때 선택되어진 주 캐쉬의 뱅크가 다른 뱅크 보다 다시 참조되어질 확률이 높다고 판단되고, 데이터가 다른 뱅크에 저장 할 수 있다면, 완전연관 버퍼에서 선택되어진 작은 블록은 주 캐쉬의 다른 뱅크에 저장이 된다. 이때 상태 비트는 갱신이 이루어지며, 주 캐쉬의 뱅크들의 데이터는 두 번째 뱅크 비트에 의해 뱅크 접근이 결정된다. 이러한 메커니즘은 주 캐쉬에서 참조가 발생할 확률이 높은 데이터를 선택하므로 충돌에 의한 참조 미스와 스레싱 효과 (thrashing effect)를 줄일 수 있다. 또한 시간적 지역성을 가지는 데이터의 수명 시간을 연장 시킬 수 있는 장점을 가진다.

3.2 상태 비트 및 메모리 접근 동작

제안된 캐쉬 시스템은 완전연관 버퍼에서 한번 참조가 발생한 작은 블록을 주 캐쉬의 각 뱅크에 저장되어진다. 이때 저장되어질 주 캐쉬의 뱅크의 결정은 3비트의 상태 비트와 뱅크 선택비트들에 의해 결정된다. 제안된 캐쉬 시스템에서 주 캐쉬의 뱅크 선택비트로 최하위 첫 태그 비트 (Tag bit)와 두 번째 태그 비트를 사용하였다. 최하위 첫 태그 비트는 기본적인 주 캐쉬의 뱅크 선택을 위한 비트로 사용된다. 그리고 두 번째 태그 비트는 주 캐쉬의 뱅크들이 모두 어떤 한 뱅크에 대한 데이터를 가질 때 주 캐쉬의 뱅크들을 선택하는 비트로 사용된다. 즉 주 캐쉬의 데이터 저장 방법에 따라 뱅크에 선택 비트가 결정된다.

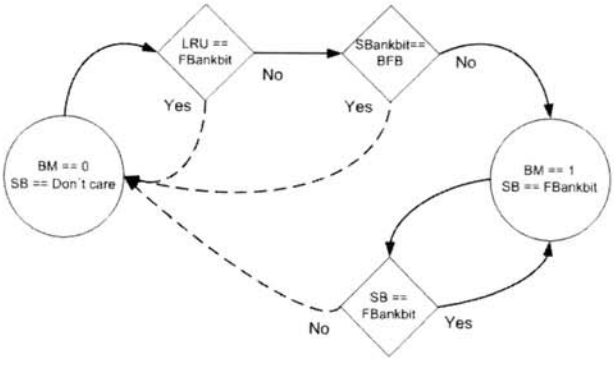
위에서 언급처럼 제안된 캐쉬 시스템에 주 캐쉬에 대한 3비트의 상태 비트를 추가하였다. 주 캐쉬의 상태 비트에서 LRU비트는 가장 나중에 사용한 뱅크를 정의하므로 주 캐쉬의 뱅크들이 어떤 한 뱅크의 데이터를 저장 할 수 있는지를 우선적으로 나타낸다. BM비트는 현재 주 캐쉬에 대한 접근이 첫 뱅크 비트 혹은 두 번째 뱅크 비트로 접근하는지를 나타내며, SB비트는 BM비트가 갱신되어 주 캐쉬의 접근이 두 번째 비트의 접근일 때, 첫 뱅크 비트를 나타내게 된다. 즉 BM비트가 '0'일 경우 주 캐쉬의 접근은 첫 뱅크 비트에 의해 각 뱅크가 결정된다. 반면 BM비트가 '1' 경우 각 뱅크들은 선택은 두 번째 뱅크 비트에 결정된다. 이때 SB비트는 첫 뱅크 비트 값을 저장하므로 주 캐쉬의 모든 블록이 어떤 뱅크의 값을 저장하는지를 나타낸다. 또한 이러한 방법은 주 캐쉬의 메모리 접근시 SB비트 값과 접근 뱅크 비

트의 값이 다르다면 주 캐쉬의 접근이 일어나지 않는 장점을 가진다. 앞서 설명처럼 BM, SB는 완전연관 버퍼에서 참조된 작은 블록이 주 캐쉬의 뱅크들에 저장될 상태에 따라 갱신이 된다. 또한 주 캐쉬에 저장될 상태는 BM, SB, LRU 그리고 뱅크선택 비트에 의해 결정된다. (그림 2)는 본 논문에서 제안된 상태 비트의 제어 흐름을 나타낸 그림이다.

여기서 FBank bit, SBank bit는 완전연관 버퍼에서 직접 사상 캐쉬로 대치어질 작은 블록의 최하위 2 비트를 나타내며, BFB (bank first bit)는 주 캐쉬에서 선택된 뱅크의 최하위 첫 비트를 나타낸다. 즉 BFB는 완전연관 버퍼에서 대치되어질 작은 블록의 두 번째 최하위 비트가 된다. 이는 제안된 캐쉬 구조에서 주 캐쉬의 기본 뱅크 선택은 태그 최하위 1비트를 사용하므로 태그 부분에서 1비트를 저장하지 않아도 태그를 계산 할 수 있다. 따라서 뱅크의 태그 부분은 최하위 1비트가 저장되지 않는다.

예로 만약 완전연관 버퍼에서 선택되어진 작은 블록의 계산 결과 주 캐쉬에 저장될 태그 부분이 00010이라면 첫 뱅크 비트(FBank bit) '0', 두 번째 뱅크 비트(SBank bit)는 '1'이 된다. 상태 비트 중 BM이 '0'일 경우 LRU비트가 '0'면 오직 이 데이터는 주 캐쉬의 뱅크0에 저장되어야한다. 하지만 LRU 비트가 '1'일 경우 주 캐쉬에 뱅크0에 저장된 최하위 태그 비트와 두 번째 비트인 '1'에 의해 데이터 저장 뱅크가 결정된다. 만약 주 캐쉬의 뱅크0의 최하위 첫 비트가 '1'이면 이 데이터는 뱅크0에 저장되며, 반면 최하위 첫 비트가 '0'이면 주 캐쉬의 뱅크1에 저장된다. 이때 주 캐쉬의 뱅크 1에 데이터가 저장되면 BM은 '1', SB는 '0'으로 갱신이 이루어진다.

반대로 BM비트가 '1'일 경우 주 캐쉬에 대해 2가지 접근이 발생한다. 먼저 SB가 '0'이면 주 캐쉬의 접근 되는 주 캐쉬의 뱅크들의 블록이 뱅크0 (첫 뱅크 비트가 0)의 데이터를 저장하고 있다. 따라서 완전연관 버퍼에서 대치되기 위해 선택되어진 작은 블록은 두 번째 뱅크에 의해 뱅크1에 저장된다. 반면 SB가 '1'일 경우 접근 되는 완전연관 버퍼와 다른 뱅크의 값이므로 주 캐쉬는 LRU에 의해 가장 최근에 사용되어진 데이터를 뱅크1에 저장하고 뱅크1에 완전연관 버퍼에서 선택되어진 작은 블록을 뱅크0에 저장하게 된다. 이때 상태 비트는 BM은'0', LRU는 1로 값이 갱신이 된다. BM이 '0'일 경우 SB의 값은 무효화로 처리된다.

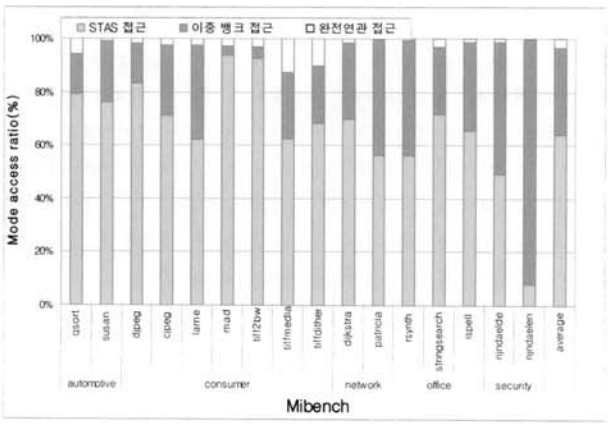


(그림 2) 제안된 상태 비트의 흐름도

기본적으로 제안된 캐쉬 시스템은 주 캐쉬와 완전연관 버퍼가 동일한 계층에 위치하여 동시 접근이 이루어진다. 그리고 위에서 언급처럼 주 캐쉬의 뱅크들은 어떤 뱅크의 데이터에 대해 두 뱅크모두 저장을 할 수 있다. 따라서 제안된 캐쉬에서 우리는 접근 방법을 크게 3가지로 나누어 실험하였다.

BM이 '0'의 값을 가질 때 제안된 캐쉬 시스템에서 주 캐쉬는 첫 뱅크 비트로 한 뱅크만이 완전연관 버퍼와 동일하게 접근이 발생한다. 우리는 이 접근을 STAS 접근이라고 정의하였다. 그리고 BM이 '1'인 경우 주 캐쉬의 해당 블록들은 모두 한 뱅크에 대한 데이터가 저장된다. 여기서 다시 두 가지의 접근이 일어난다. 우선 SB가 첫 뱅크 비트와 동일한 값일 경우 완전연관 버퍼와 두 번째 뱅크 비트에 의한 주 캐쉬의 뱅크 접근이 발생한다. 우리는 이 접근을 한 뱅크의 값이 모든 뱅크에 사용되므로 이중 뱅크 접근이라고 정의하였다. 그리고 SB의 값이 첫 뱅크 비트와 다르다면 제안된 캐쉬 시스템에서 주 캐쉬의 접근이 일어날 필요가 없다. 따라서 완전연관 버퍼의 접근만이 발생하게 된다. 이때 접근을 우리는 완전연관 접근이라고 정의 하였다.

(그림 3)은 위에서 언급된 제안된 캐쉬 시스템의 접근들을 나타낸 비율이다. 그림에서 보듯이 security 부분에서 약 40%의 이중 뱅크 접근이 발생하였다. 실제 시뮬레이션 결과 다른 비교 캐쉬에 비해 security에서 상당한 참조 실패율을 줄였다. 전체적으로 제안된 캐쉬 시스템에서 평균 35%의 이중 뱅크 접근이 발생하였다.



(그림 3) Mibench : 주 캐쉬 접근 모드

3.3 캐쉬 제어 알고리즘

여기서 제안된 캐쉬 시스템의 동작 원리를 상세히 설명하고자 한다. CPU로부터 임의의 메모리 참조가 발생하게 되면, 상태 비트 정보에 의해 주 캐쉬와 완전연관 버퍼가 동시에 접근이 일어나는 STAS 접근, 이중 뱅크 접근 그리고 완전연관 버퍼만 접근되는 완전연관 접근으로 3가지 접근이 발생한다. 두 캐쉬의 메모리 참조 실패시 완전연관 버퍼의 작은 블록이 주 캐쉬의 어떤 뱅크에 대한 저장은 제안된 상태 비트와 뱅크 비트들의 조합으로 이루어진다. 이때 완전

연관 버퍼에서 선택되어진 작은 블록의 저장 위치에 따라 상태 비트가 결정된다. 구체적 가능한 동작은 다음과 같다.

㉠ 주 캐쉬에서 적중 :

주 캐쉬에서 어떤 뱅크에서 적중이 일어나면 기존의 직접 사상 캐쉬와 동일한 방법으로 CPU 요청을 수행하게 된다. 읽기 명령어인 경우 요청한 데이터를 CPU로 보내고 캐쉬의 동작은 끝나게 되며, 쓰기 명령어인 경우 해당 블록에 쓰기 동작을 수행함과 동시에 갱신 비트 (dirty bit)를 1로 설정하게 된다. 그리고 상태 비트에서 LRU비트만이 갱신이 일어나게 된다.

㉡ 완전연관 버퍼에서 적중 :

메모리 주소가 CPU로부터 발생되어질 때 주소의 일부 비트를 이용하여 큰 블록 내의 여러 개의 뱅크들 중에서 부합되는 하나의 작은 블록만이 인출되어짐과 동시에 참조가 일어난 블록임을 나타내기 위하여 그 작은 블록의 적중 비트를 1로 설정되어 진다. 위에서 언급했듯이 완전연관 버퍼의 데이터 부분은 뱅크로 나누어져 있기 때문에 디코더를 이용하여 항상 완전연관 버퍼 참조 시 하나의 뱅크만 선택적으로 접근이 가능하다. 이것은 완전연관 버퍼에서 소비전력을 줄이기 위한 방법이며 뱅크로 구현된 주 캐쉬와 같이 작은 블록크기로 동작 되어질 수 있다. 예로 작은 블록 크기가 8-byte이고 큰 블록의 크기가 32-byte인 경우 완전연관 버퍼의 데이터 저장 뱅크의 수는 4개로써 구성되어진다. 따라서 한 번에 하나의 뱅크만 선택되어지므로 전력 소비적인 측면에서 유리한 장점을 가지게 된다. 또한 주 캐쉬의 접근 없이 완전연관 접근이 이루어지므로 더욱 에너지 소비 측면에서 유리한 장점을 가지게 된다. 만약 완전연관 버퍼내의 하나의 뱅크에서 적중이 발생되면 읽기 명령인 경우 적중이 발생한 뱅크의 작은 블록으로부터 요청한 데이터를 CPU로 보내고 동시에 그 블록의 적중 비트를 1로 설정한다. 쓰기 명령인 경우에는 참조가 일어난 블록에 쓰기 동작을 수행하고 갱신 비트를 동시에 1로 설정한다.

㉢ 두 캐쉬에서 접근 실패 :

만약 주 캐쉬와 완전연관 버퍼에서 모두 접근 실패가 발생하면 캐쉬 제어기는 접근 실패 (miss handing)을 실행하게 된다. 기본적인 동작은 주 캐쉬에서 접근 실패가 발생한 작은 블록을 포함한 큰 블록이 다음 메모리 계층으로부터 인출되어 완전연관 버퍼에 저장되어진다. 예를 들어 작은 블록이 8byte이고 큰 블록이 32byte인 경우 32byte 블록의 경계에 속하는 4개의 순차적인 작은 블록이 인출되어 완전연관 버퍼에 저장되어지며, 이때 완전연관 버퍼의 유효 비트 (valid bit)가 모두 1인 경우와 그렇지 않은 경우로 나누어진다.

- 완전연관 버퍼의 유효 비트가 모두 1이 아닌 경우: 접근 실패가 발생하였을 때 완전연관 버퍼 내에 적어도 하나의 엔트리가 무효화 상태이면 다음 메모리 계층으로부터 큰 블록을 인출하여 비어있는 엔트리에 저장시킴과 동시에 참조가 발생한 작은 블록의 적중 비트와 유효 비트를 각각 1로 설정한다. 만약 읽기 접근 실패이면 요청한 데이터를 CPU로 보내고, 쓰기 접근 실패이면 쓰기 정정 (write-back) 모드 방식으로 동작하게 되면 요청한 큰 블록을 완전연관 버퍼에 저장하게 된다.

- 완전연관 버퍼의 유효 비트가 모두 1인 경우: 우리는 완전연관 버퍼의 대치 알고리즘으로 선입선출 (FIFO) 방식을 채택하였다. 접근 실패시 가장 오래된 큰 블록이 선택되어지며, 그 대치 블록에 속하는 작은 블록 중에 이미 참조가 발생한 블록을 선택하여 뱅크로 구성된 주 캐쉬에 저장하게 된다. 그러므로 완전연관 버퍼에서 직접 다음 계층 메모리로 쓰기 정정되는 경우는 발생하지 않는다. 하지만 만약 뱅크로 구성된 주 캐쉬의 대치될 블록에 유효 데이터가 존재할 때 그 대치 블록의 갱신 비트가 1인 경우 다음 계층 메모리로 쓰기 정정이 일어나며, 0인 경우 무시하게 된다.

대치 블록에 속하는 한번 참조가 발생한 작은 블록의 주 캐쉬의 저장은 3.2절에서 설명한 알고리즘에 의해 각 뱅크를 선택하게 되며, 각 뱅크에 대한 상태 비트들이 갱신이 이루어진다.

4. 성능 평가

이 장에서는 시뮬레이션 수행을 통하여 제안된 캐쉬 구조의 메모리 참조 실패율, 평균 메모리 접근 시간 그리고 소비 에너지를 다른 캐쉬와 비교 분석하여 성능평가를 하였다. 제안된 캐쉬의 성능 평가를 위해 내장형 시스템을 평가하는데 사용되는 벤치마크인 Mibench [13, 14]와 일반 시스템 평가하는 Spec2000(CINT, CFP) [15, 16]을 각각 arm 바이너리와 alpha 바이너리코드로 컴파일 하여 사용하였다.

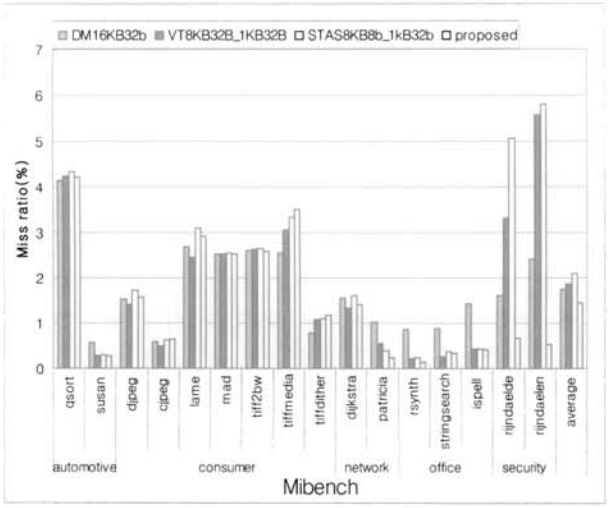
먼저 Simplescalar 프로세서 [17] 시뮬레이터를 사용하여 각각의 벤치마크를 1억개의 명령어를 수행하는 동안의 데이터 참조 주소를 모니터링 하였고, 데이터 캐쉬 참조 주소 트레이스를 캐쉬 시뮬레이터에 입력하여 제안하는 캐쉬 구조의 성능을 평가 하였다. 비교 대상 캐쉬로 일반적인 기존의 직접사상 캐쉬와 Victim cache 그리고 간단한 하드웨어 구조로 2 지역성의 효과적으로 사용한 STAS 캐쉬를 비교 캐쉬로 사용하였다.

4.1 접근 실패율과 평균 메모리 접근 시간

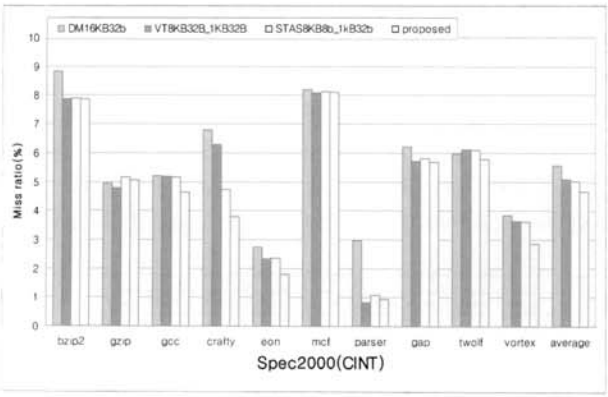
일반적으로 캐쉬의 성능을 나타내는 지표로서 메모리 참조 실패율 (miss ratio)과 평균 메모리 접근 시간 (AMAT: average memory access time)을 이용한다. 성능 평가를 위하여 대표적인 캐쉬 구조와 제안된 캐쉬의 접근 실패율과 평균 메모리 접근 시간을 비교 분석하였다. 기존의 직접사상 캐쉬는 DM으로 표기 하였으며, "16KB32b"는 32bytes의 블록 크기를 가지는 16Kbyte의 캐쉬 크기를 나타낸다. 또한 Victim 캐쉬 (VT)와 STAS 캐쉬는 각각 제안된 캐쉬와 동일하게 완전연관 버퍼는 32개의 엔트리를 가지는 1Kbyte로

구성하였다. 제안된 캐쉬는 8byte의 블록 크기를 가지는 8Kbyte의 2-뱅크로 구성된 주 캐쉬와 32byte의 블록 크기를 가지는 1Kbyte의 완전연관 버퍼로 구성되어 진다. 실험은 각각의 벤치마크에 대해 이상 언급된 블록 크기와 캐쉬 크기로 수행하였다.

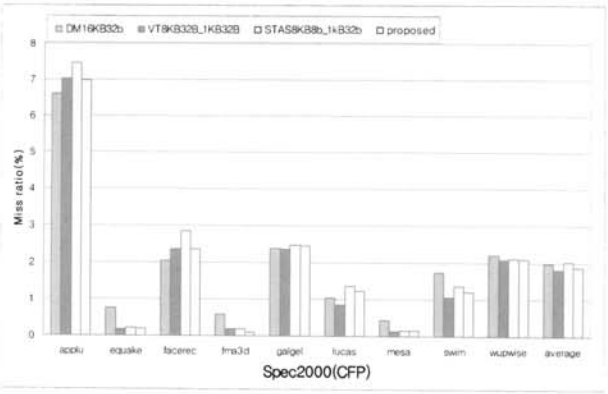
(그림 4)-(그림 6)은 제안된 캐쉬와 다른 비교 캐쉬들의 메모리 참조 실패율을 각 벤치마크 별로 표시 하고 있다. (그림 4)는 Mibench에서의 메모리 참조 실패율을 보이고 있



(그림 4) Mibench : 메모리 접근 실패율



(그림 5) Spec 2000(CINT) : 메모리 접근 실패율



(그림 6) Spec2000 (CFP) : 메모리 접근 실패율

다. 제안된 캐쉬가 동일한 용량의 Victim 캐쉬, STAS 캐쉬에 비해 각각 23%와 32%의 낮은 메모리 참조 실패율을 보이고 있으며, 16KB의 직접사상 캐쉬와 비교해서도 약 17%의 낮은 메모리 참조 실패율을 보이고 있다. 특히 Security 벤치마크에서 다른 캐쉬들에 비해 아주 좋은 성능을 보이고 있다. 이는 (그림 3)에서 보는 것처럼 제안된 캐쉬 구조에서 이중 뱅크 접근이 STAS 접근 보다 월등히 많은 것을 알 수 있다. 즉 Security 벤치마크에서 제안된 캐쉬 구조가 좋은 성능을 보이는 것은 이러한 이중 뱅크에 의해 충돌 미스와 스래싱 효과를 줄이므로써 메모리 참조 성공률을 높인 원인으로 분석된다.

(그림 5)와 (그림 6)은 일반 시스템 응용군인 Spec2000의 정수형 타입 (CINT)과 실수형 타입 (CFP)에서의 메모리 참조 실패율을 나타낸 것이다. 역시 제안된 캐쉬 구조는 정수형 타입에서 16KB 직접사상 캐쉬에 비해 약 18%의 낮은 메모리 참조 실패율을 보이며, Victim 캐쉬와 STAS 캐쉬에 비해 약 10%의 낮은 메모리 참조 실패율을 보이고 있다. 하지만 실수형 타입에서는 16KB 직접사상 캐쉬와 STAS 캐쉬에 대해 좋은 성능을 보이며, Victim 캐쉬와는 거의 동일한 성능을 보이고 있다. 따라서 Spec2000에서도 평균적으로 제안된 캐쉬 구조가 우수하다는 것을 알 수 있다.

캐쉬 시스템의 성능 평가를 보다 정확하게 하기 위해 성능 평가의 또 하나의 지표인 평균 메모리 접근 시간을 이용하였다. AMAT를 얻기 위한 식은 다음과 같다.

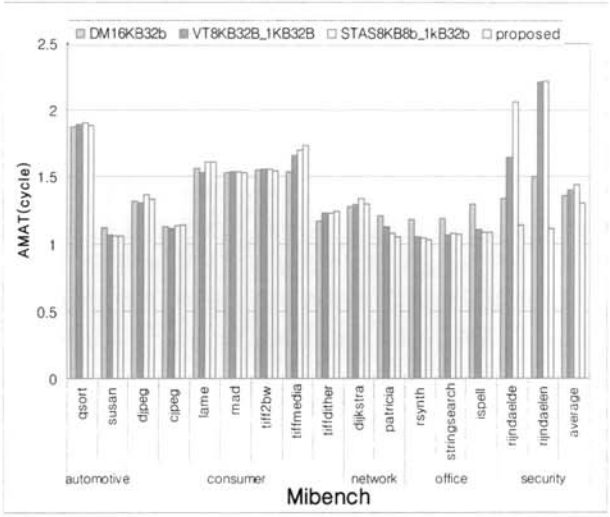
Average memory access time (AMAT)
= Hit time + Miss rate * Miss penalty.....(수식 1)

여기서 hit time은 캐쉬에서 적중을 처리하는데 걸리는 시간이며, miss penalty는 캐쉬 접근 실패 시 이를 처리하는데 추가되는 시간이다. 시뮬레이션을 수행하기 위한 구체적인 변수 값들은 <표 1>로 정의되어진다. 이러한 변수 값들은 Hittachi SH4 또는 ARM920T와 같은 일반적인 32bit 내장형 프로세서에서 사용되어지는 값들을 사용하였다. 이를 기준으로 8byte 블록이 64byte 데이터 버스를 통하여 매 사이클 전송가능하며, 32byte의 블록을 인출하는 사이클은 22 클럭 사이클이 소요된다.

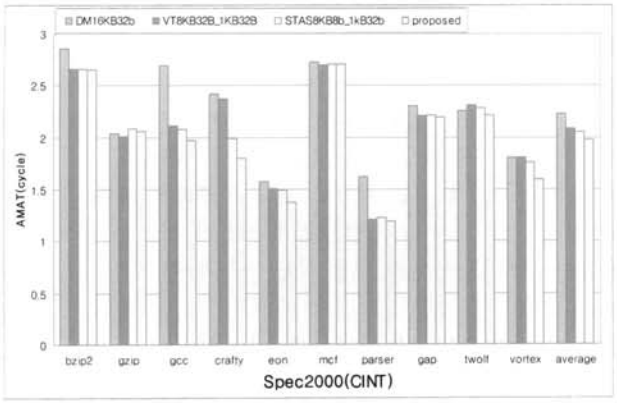
(그림 7)-(그림 9)는 Mibench와 Spec2000에서 제안된 캐쉬 시스템과 비교 캐쉬들의 평균 메모리 접근 시간을 나타낸 그래프이다. 전반적인 벤치마크에서 제안된 캐쉬 시스템

〈표 1〉 시뮬레이션 변수들

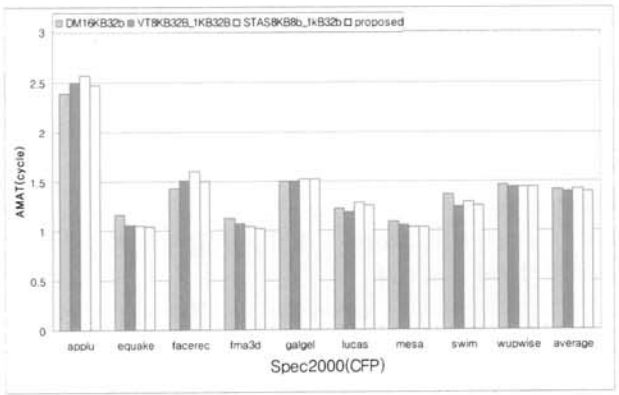
System parameters	Values
CPU clock	200MHz
L2 Cache	None
Memory latency	15 / cpucycle
Memory bandwidth	1.6Gbytes / sec
Banked cache hit time	1 / cpucycle
FAB hit time	1 / cpucycle



(그림 7) Mibench : 평균 메모리 접근 시간



(그림 8) Spec2000(CINT) : 평균 메모리 접근 시간



(그림 9) Spec2000(CFP) : 평균 메모리 접근 시간

의 평균 메모리 접근 시간이 비교 캐쉬들 비해 높은 성능 향상을 보이고 있다.

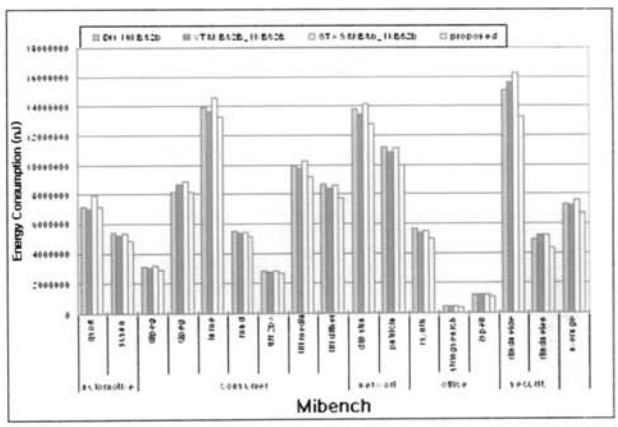
4.2 소비 에너지

제안된 캐쉬 구조의 메모리 참조 실패율과 평균 메모리 접근 시간을 기준으로 하는 성능 이외에 에너지 소비 감소 효과를 측정하기 위하여 캐쉬 구조에 대한 여러 가지 인수들을 계산하여 주는 툴 CACTI 3.2 [18]를 사용하여 캐쉬의 에너지 소비를 평가하기 위하여 사용되는 인수 값을 획득하였다. 본 논문에서 사용되는 CACTI의 공정 값은 $0.13\mu m$이며, 전압은 1.3V로 가정하였다. 비교 평가 캐쉬는 메모리 접근 시간과 참조 실패율을 위한 비교 캐쉬와 동일하며, 내장형 시스템을 위한 응용 프로그램인 Mibench을 사용하여 비교 분석하였다. 전체 에너지 소비를 얻기 위한 식은 다음과 같다.

Overall_energy_consumption
= number_of_hits * hit_energy_consumption + number_of_misses * miss_handling_energy_consumption
(수식 2)

여기서 number_of_hits, hit_energy_consumption 그리고 number_of_misses는 각각 다시 주 캐쉬와 완전연관 버퍼가 동시에 접근 할 때와 완전연관 버퍼만 접근할 경우 나누어진다.

(그림 10)은 Mibench에서 비교 캐쉬와 제안된 캐쉬의 소비 에너지를 나타낸 그래프다. 제안된 캐쉬의 에너지 소비는 비교 캐쉬들에 비해 더 낮은 소비 에너지를 보이고 있다. 결과적으로 직접사상 캐쉬, Victim 캐쉬, 그리고 STAS 캐쉬에 비해 전체적으로 약 10%의 낮은 소비 에너지를 보이고 있다. 메모리 참조 실패율에 비해 에너지 소비의 비율이 낮은 원인은 Victim 캐쉬의 경우 버퍼에서 참조 성공이 발생하면 직접사상 캐쉬와의 데이터 스왑으로 인하여 버퍼에서의 낮은 접근 에너지만을 소비하지만 제안된 캐쉬구조와 STAS 캐쉬 경우 버퍼에서 Victim 캐쉬에 비해 많은 접근 성공으로 인한 에너지 소비가 높기 때문이다. 그럼에도 불구하고 제안된 캐쉬 구조는 Victim 캐쉬에 비해 좋은 소비 에너지를 보이고 있다. 따라서 제안된 캐쉬 구조는 메모리 접근 시간과 실패율뿐 아니라 에너지 소비관점에서도 비교 캐쉬들과 비교하여 좋은 성능을 보이는 것을 알 수 있다.



(그림 10) Mibench : Energy Consumption (nJ)

5. 결 론

본 논문은 내장형 시스템에서 데이터 캐쉬의 성능 향상을

위한 캐쉬 구조를 제안하였으며, 기본적으로 프로그램 수행 특성에 적합한 두 지역성을 이용하기 위하여 이중 캐쉬 구조로 설계하였다. 이중 캐쉬는 낮은 소비 에너지와 시간적 지역성을 효과적으로 이용하기 위해 작은 블록을 가지는 주 캐쉬를 2-뱅크 구조로 구현하였으며, 공간적 지역성을 위해 큰 블록을 가지는 완전연관 버퍼로 구성되며, 큰 블록은 몇 개의 주 캐쉬와 동일한 크기의 블록들로 구성된다. 특히 주 캐쉬의 2-뱅크 구조는 직접사상 캐쉬보다 낮은 메모리 참조 성공률을 보완하기 위하여 제안된 알고리즘에 의해 선택되어진 주 캐쉬의 뱅크가 다른 뱅크보다 다시 참조 되어질 확률이 높고, 데이터가 다른 뱅크에 저장 가능한 조건에서 주 캐쉬의 선택되어진 뱅크가 아닌 다른 뱅크에 데이터를 저장하였다. 이러한 방법은 주 캐쉬에서 충돌 미스에 의한 참조 실패를 줄일 뿐 아니라 다시 참조 확률이 높은 데이터를 오래 가짐으로써 캐쉬 오염을 줄일 수 있는 장점을 보인다. 더욱이 시간적 지역성을 고려한 주 캐쉬는 참조 확률이 높은 데이터의 수명 시간을 연장하는 결과를 가지고 온다.

요컨대 제안된 캐쉬 시스템은 프로그램 수행 특성에 적합한 지역성을 이용하기 위해 2중 캐쉬 구조로 설계하였다. 시간적 지역성을 위한 주 캐쉬는 저전력을 위한 뱅크 구조이며, 제안된 알고리즘에 의해 특정 조건에서 주 캐쉬의 두 뱅크를 어떤 한 뱅크에 대한 데이터 저장을 할 수 있도록 설계하였다. 공간적 지역성을 위한 완전연관 버퍼는 데이터 저장 부분을 주 캐쉬와 동일한 크기의 작은 블록들로 구성되며, 참조시 작은 한 블록만을 접근 하므로 낮은 에너지 소비에 용이하도록 설계하였다.

시뮬레이션 결과 내장형 응용 프로그램에서 전체 평균 약 35% 이중 뱅크 접근이 일어난 것을 확인하였다. 이에 메모리 접근 실패율은 16Kb 직접사상 캐쉬와 동일한 용량을 가지는 Victim 캐쉬와 STAS 캐쉬에 비해 각각 17%, 23% 및 32%의 낮은 메모리 접근 실패율을 보였으며, 평균 메모리 접근 시간 역시 16Kb 직접사상 캐쉬, Victim 캐쉬, 그리고 STAS 캐쉬에 비해 각각 10%, 14%, 그리고 18%의 성능 향상 효과를 보였다. 또한 에너지 소비관점에서 다른 캐쉬 구조에 비해 약 10%의 낮은 소비 에너지를 보이고 있다. 평균 메모리 접근 시간, 메모리 접근 실패율 그리고 소비 에너지등 전체적인 평가에서 제안된 캐쉬 구조는 기존의 대표적인 캐쉬 구조에서 비해 모든 부분에서 좋은 성능을 보이고 있다.

참 고 문 헌

[1] W. Shiue, S. Udayanarayanan, and C. Chakrabati, "Data memory design and exploration for low-power embedded systems," ACM Trams. Design Automation of Electronic Systems, Vol.6 No.4, pp.553-568, Oct., 2001.

[2] S. Santhanam, "Strong ARM SA110-A 160Mhz 32b 0.5W CMOS ARM Processor," Hot Chips 8: A Symposium on High-Performance Chips, Aug., 1996.

[3] A. j. Smith, "Cache Memories," ACM Computing Surveys, Vol.14, No.3, pp.473-530, Sep., 1982

[4] Kessler, R. E, Jooss, R., Lebeck, A.,and Hill, M. D, "Inexpensive Implementations of Set-Associativity," Proc. of the 16th International Symposium on Computer Architectures, pp.131-139, 1989.

[5] Norman P. Jouppi, "Improving Direct-Mapped Cache Perfor mance by the Addition of a Small Fully Associative Cache and Prefetch Buffer," 17th ISCA, pp.364-347, May, 1990.

[6] C. Zhang and F. Vahid, "Using a Victim Buffer in an Application-Specific Memory Hierarchy," Design Automation and Test in Europe Conference (DATE), pp.220-225, February, 2004.

[7] P. J. de Langen et al, "Reducing traffic generated by conflict misses in caches," The 1st ACM International Conference on Computing Frontiers, pp.235-239, Apr., 2004.

[8] A. Gonzalez, C. Aliagas and M. Mateo, Data Cache with Multiple Caching Strategies Tuned to Differnt Tyoes of Locality, Suppercomputing '95, pp.338-347, July, 1995,

[9] B. Juurlink, "Unified Dual Data Cache," Proceedings. Euromicro Symposium on Digital System Design, pp.33-40, Sept., 2003.

[10] 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, Vol.I, pp.151-162, Aug., 1996.

[11] J. H. Lee, J. S. Lee, and S. D. Kim, "A New Cache Architecture Based on Temporal and Spatial Loçality, " J. Systems Architecture, vol.46, pp.1451-1467, Sept., 2000.

[12] A. Naz, M. Rezaei, K. Kavi and P. Sweany, Improving data. cache performance with integrated use of split caches, victim. cache and stream buffers, in Proceedings of the Workshop. on Memory performance dealing with appli cations, systems and architecture, Conference (DATA) Sept., 2004.

[13] Guthaus, M.R.; Ringenberg, J.S. Ernst, D. Austin, T.M. Mudge, T.; Brown, R.B., "MiBench: A free, commercially representative embedded benchmark suite," IEEE 4th Annual Workshop on Workload Characterization, Austin, TX, December, 2001.

[14] Mibench Version 1.0 http://www.eecs.umich.edu/mibench/

[15] Henning, John L., "SPEC CPU2000: Measuring CPU Perfor mance in the New Millennium," IEEE Computer, Vol.33, No.7, pp.28-35, July, 2000.

[16] SPEC Benchmark Suite. Information available at http://www.spec.org

[17] D. Burger and T. M. Austin, "The SimpleScalar tool set, version 2.0, Technical Report TR-97-1342," University of Wisconsin-Madison, 1997.

[18] G. Reinman. and N. P. Jouppi, "CACTI 3.0: An integrated cache timing and power, and area model," Compaq WRL Report, Aug., 2001.

정 보 성

e-mail : blueking80@gnu.ac.kr
2006년 경상국립대학교 제어계측공학과(학사)
2008년 경상국립대학교 제어계측공학과(석사)
2008년~현 재 경상국립대학교 제어계측공학과 박사과정

관심분야 : 지능형 메모리 시스템, 내장형 시스템 및 SOC 시스템

이 정 훈

e-mail : leejh@gsnu.ac.kr
1999년 성균관대학교 제어계측공학과(학사)
2001년 연세대학교 컴퓨터과학과(석사)
2004년 연세대학교 컴퓨터과학과(박사)
2004년~현 재 경상국립대학교 전기전자공학부 공학연구원 조교수

관심분야 : 지능형 메모리 시스템, 저전력/고성능 마이크로프로세서, 내장형 시스템 및 SOC 시스템 등