내장형 시스템을 위한 에너지-성능 측면에서 효율적인 2-레벨 데이터 캐쉬 구조의 설계

(Energy-Performance Efficient 2-Level Data Cache Architecture for Embedded System)

이 종 민 † 김 순 태 ††
(Jongmin Lee) (Soontae Kim)

요 약 온칩(on-chip) 캐쉬는 외부 메모리로의 접근을 감소시키며 빈번하게 접근되기 때문에 내장형 시스템의 성능과 에너지 소비 측면에서 중요한 역할을 한다. 본 논문에서는 내장형 시스템에 맞추어 설계된 2-레벨 데이터 캐쉬 메모리 구조를 제안하고자 한다. 레벨1(L1) 캐쉬의 구성으로 작은 크기, 직접사상(direct-mapped) 그리고 바로쓰기(write-through)를 채용한다. 대조적으로 레벨2(L2) 캐쉬는 보통의 캐쉬 크기와 집합연관(set-associativity) 그리고 나중쓰기(write-back) 정책을 채용한다. 결과적으로 L1 캐쉬는 빠른 접근 시간을 가지며 (한 사이클 이내) L2 캐쉬는 전체 캐쉬의 미스율(global miss rate)을 낮추는데 효과적이다. 작은 크기의 L1 데이터 캐쉬로 인한 증가된 캐쉬 미스율(miss rate)을 줄이기 위해 ECP(Early Cache hit Predictor)기법을 제안하였다. 제안된 ECP기법은 L1 캐쉬 히트 예측을 통해서 요청된 데이터가 L1 캐쉬에 있는지 예측할 수 있으며 추가적으로, ALU를 필요로 하지 않고 빠르게 유효주소(effective address)계산을 할 수 있다. 또한, 두 캐쉬 계층간 바로쓰기(write-through) 정책에서 오는 빈번한 L2 캐쉬 접근으로 인한 에너지 소비를 줄이기 위해 지정웨이 쓰기(one-way write) 기법을 제안하였다. 제안된 지정웨이 쓰기 기법을 이용하면 바로쓰기 정책으로 인한 L1 캐쉬에서 L2 캐쉬로의 쓰기 접근시 태그(tag) 비교 과정을 거치지 않고 하나의 지정된 웨이를 바로 접근할 수 있다. 사이클 단위 정확도의 시뮬레이터와 내장형 벤치마크를 이용한 실험 결과 본 논문에서 제안한 2-레벨 데이터 캐쉬 메모리 구조는 평균적으로 3.6%의 성능향상과 50%의 데이터 캐쉬 에너지 소비를 감소 시켰다.

키워드 : 2-레벨 데이터 캐쉬, 캐쉬 히트 예측, 지정웨이 쓰기

Abstract On-chip cache memories play an important role in both performance and energy consumption points of view in resource-constrained embedded systems by filtering many off-chip memory accesses. We propose a 2-level data cache architecture with a low energy-delay product tailored for the embedded systems. The L1 data cache is small and direct-mapped, and employs a write-through policy. In contrast, the L2 data cache is set-associative and adopts a write-back policy. Consequently, the L1 data cache is accessed in one cycle and is able to provide high cache bandwidth while the L2 data cache is effective in reducing global miss rate. To reduce the penalty of high miss rate caused by the small L1 cache and power consumption of address generation, we propose an ECP (Early Cache hit Predictor) scheme. The ECP predicts if the L1 cache has the requested data using both fast address generation and L1 cache hit prediction. To reduce high energy cost of accessing the L2 data cache due to heavy write-through traffic from the write buffer laid between the two cache

· 이 논문은 2008년도 정부재원(교육인적자원부 학술 연구조성사업비)으로 한국학술진흥재단의 지원을 받아 연구되었음(KRF-2008-331-D00457)

† 학생회원 : 한국과학기술원 전산학과
square55@kaist.ac.kr
†† 정 회 원 : 한국과학기술원 전산학과 교수
kims@kaist.ac.kr

논문접수 : 2010년 1월 20일
심사완료 : 2010년 7월 25일

정보과학회논문지: 시스템 및 이론 제37권 제5호(2010.10)

levels, we propose a one-way write scheme. From our simulation-based experiments using a cycle-accurate simulator and embedded benchmarks, the proposed 2-level data cache architecture shows average 3.6% and 50% improvements in overall system performance and the data cache energy consumption.

Key words : 2-level data cache, Early cache hit predictor, One-way write

1. 서 론

중앙처리장치(CPU)와 메모리(main memory)간의 성능 격차는 더욱 벌어지고 있다. 캐쉬 메모리가 중앙처리장치와 메모리간 성능차이를 완화하기 위해 사용되고 있지만 매년 빠르게 발전하는 중앙처리장치와 더디게 발전하는 메모리와의 속도 차이는 더욱 커지고 있다. 게다가 비디오 스트리밍(video-streaming), 이미지 처리(image-processing), 고성능 무선통신(high speed wireless communication) 등과 같은 응용 프로그램의 등장은 내장형 시스템에 대한 성능향상과 저전력 소비에 대한 요구를 더욱 높이게 되었다. 그러나 일반적으로 내장형 시스템은 제한적인 전력 소비요구와 성능을 가지고 있기 때문에 성능향상과 저전력 소비의 요구를 동시에 충족하는 것은 쉽지 않다. 따라서, 이러한 상반되는 요구를 동시에 만족하기 위해 내장형 시스템을 위한 새로운 메모리 구조를 탐험 하는 것은 중요하다고 할 수 있다. 왜냐하면, 메모리 시스템은 시스템 전력의 많은 부분을 소비하고 성능에 직접적으로 영향을 끼치는 부분이기 때문이다. 메모리 시스템의 구성원들 중 특별히 캐쉬 메모리는 매우 중요한 부분이다. 캐쉬 메모리는 내장형 프로세서 내에서 최대전력을 소모하고 시장 진입시간, 개발비용, 그리고 시스템의 성능 확장성(performance scalability)에 큰 걸림돌로 작용하고 있기 때문이다. 예를 들면, ARM920T 내장형 프로세서에서 캐쉬 메모리는 프로세서 전력소비의 44%를 차지하고 TLB등을 포함하면 온칩(on-chip) 메모리는 53%의 전력을 소비한다[1]. 또한 캐쉬 메모리는 내장형 프로세서에서 대부분의 칩 면적(area)을 차지한다. ARM920T 프로세서에서 프로세서 코어는 11만개의 트랜지스터로 구현이 되어 있는데 비해 프로세서 전체는 250만개의 트랜지스터로 구성되어 있다[1]. 이 두 숫자의 차이는 주로 온칩 캐쉬 메모리에 기인한다. 이러한 거대 규모로 인해서 캐쉬 메모리는 내장형 프로세서의 구성요소 중 최대 전력을 소비하고, 파이프라인에서 최대 접근 시간(access time)을 필요로 하기 때문에 프로세서의 클럭 사이클(clock cycle time)을 결정하는 중요한 요소이다[2]. 따라서 저전력을 소비하고 접근 시간이 짧은 특성을 가지는 캐쉬 메모리 구조는 내장형 시스템의 전체 전력 소비를 크게 감소시킬 수 있으며 메모리 접근 속도를 감소시킴으로써 프로세서의 성능을 향상시킬 수 있다.

본 논문에서는 고성능 프로세서에서 통용되는 2-레벨 캐쉬 구조와 유사하지만 자원 제한적인 내장형 시스템에 적합한 2-레벨 데이터 캐쉬 구조를 제안하였다. 제안된 캐쉬 구조에서 L1 데이터 캐쉬는 작은(1KB-2KB) 크기이며 직접사상(direct-mapped)을 사용하고 바로쓰기 정책(write-through)을 채용하였다. 반면에, L2 캐쉬는 일반적인 1-레벨 캐쉬와 비슷한 크기(8KB-16KB)를 갖고 집합연관 사상(set-associative)을 사용하며 나중쓰기 정책(write-back)을 채용하였다. 작은 크기와 직접사상을 채용한 L1 데이터 캐쉬는 저전력 소비와 빠른 접근 시간을 보장할 수 있으며 바로쓰기 정책을 통해 캐쉬의 처리량(bandwidth)을 최대화할 수 있다[3]. 나중쓰기(write-back)와 집합연관 사상을 채용한 L2 캐쉬는 L1 캐쉬에서의 캐쉬 미스를 보완하여 메모리로의 접근을 감소시킨다. L1 데이터 캐쉬에서 L2 캐쉬로의 쓰기 동작으로 인한 지연시간(delay)을 감소시키고 쓰기 합병(write merging)를 유도하기 위해 두 캐쉬 레벨 사이에 쓰기버퍼(write buffer)가 존재한다.

그러나, 제안된 2-레벨 캐쉬 구조의 작은 L1 데이터 캐쉬 크기로 인해 L1 데이터 캐쉬 미스율이 증가하고 바로쓰기 정책으로 인한 L2 캐쉬로의 빈번한 접근 때문에 L2 캐쉬의 전력소비가 증가한다. 이러한 두 가지 문제점을 보완하기 위해서 ECP(Early Cache hit Predictor) 기법과 지정웨이 쓰기(one-way write)라 명명된 기법을 제안하였다. ECP는 파이프 라인의 실행(execution) 단계에서 기본 주소(base address)와 오프셋 주소(offset address)를 이용하여 L1 데이터 캐쉬로의 접근이 히트를 할 것인지 미스를 할 것인지 예측한다. 만약 예측의 결과가 L1 데이터 캐쉬에서 미스라고 판단 될 경우, L1 캐쉬로의 접근은 수행되지 않고 바로 L2 캐쉬로 접근하게 된다. 따라서 불필요한 L1 데이터 캐쉬로의 접근과 L1 데이터 캐쉬 미스율을 효과적으로 줄일 수 있다. 지정웨이 쓰기는 바로쓰기 정책(쓰기 히트시)에서 빈번한 L2 캐쉬로의 접근시 소비되는 전력을 줄이기 위해 제안되었다. L2 캐쉬에서 L1 데이터 캐쉬로 데이터가 로드될 때 해당 L2 캐쉬의 웨이 번호가 L1 캐쉬의 블록내의 추가된 비트에 기록된다. 기록된 웨이 번호는 바로 쓰기정책으로 인한 L2 캐쉬로의 접근시 사용되며 해당 웨이 번호에서 반드시 히트할 것이기 때문에 태그비트 비교를 생략할 수 있으며 모든 웨이를 활성화 하지 않게 한다.

본 논문의 기대효과로는 다음과 같은 것들이 있다.

- 제안된 2-레벨 데이터 캐쉬 구조는 특정한 어플리케이션이나 목적을 대상으로 하지 않았기 때문에 일반적인 임베디드 시스템에 적용이 가능하다.
- 기존의 캐쉬 메모리는 내장형 프로세서의 클럭 주파수를 증가시킬 경우 쉽게 성능이 확장되지 않는 단점이 있으나 본 논문에서 제안한 2-레벨 데이터 캐쉬는 성능 확장이 용이하다.
- 제안된 2-레벨 데이터 캐쉬 구조는 성능향상과 에너지 소비의 감소라는 상호 교환적인 두 가지 목표를 동시에 충족시키기 위해 고안되었으며 시뮬레이션 실험결과를 통해 제안된 기법들이 상당히 효과적임을 입증하였다.
- 본 논문에서 제안한 2-레벨 데이터 캐쉬를 위한 기법은 고성능 시스템에도 적용이 가능하다. 고성능 시스템에서도 L1과 L2 데이터 캐쉬 사이에 바로쓰기 정책을 채용하는 경우 L2 데이터 캐쉬의 증가된 에너지 소비를 줄이기 위해서 적용 가능하다.

아래의 내용은 다음과 같이 구성되어 있다. 2장에서 관련된 연구에 대해서 논의하고 3장에서 제안된 2-레벨 캐쉬 구조에 관해 자세하게 설명한다. 4장에서는 제안된 캐쉬 구조를 평가하기 위한 실험 준비와 실험결과에 대한 분석을 하고 마지막으로 5장에서 결론을 맺는다.

2. 관련 연구

지금까지 매우 다양한 종류의 캐쉬 구조들이 저전력 소비와 성능향상을 위해 제안되었다. 그 중 제안된 캐쉬 구조와 관련 있는 선행 연구들은 다음과 같다.

중앙처리장치와 명령어 캐쉬 사이(instruction cache)에 위치한 작은 크기의 필터 캐쉬(filter cache)가 있다[4-6]. 프로세서는 명령어 캐쉬 메모리에서 소비되는 전력을 줄이기 위해 명령어 L1 캐쉬를 접근하기 전에 보다 작은 크기를 가지는 레벨-0 형태의 필터 캐쉬를 접근한다. 필터 캐쉬가 작은 크기이기 때문에 명령어가 필터 캐쉬에서 히트한다면 명령어를 인출(fetch)하는 시간과 전력소비를 줄일 수 있다. 그러나 작은 크기의 필터 캐쉬는 일반적으로 히트율(hit rate)이 낮기 때문에 오히려 캐쉬 접근시간 및 전력 소비를 증가시킬 수도 있다. 이러한 문제를 해결하기 위해 예측에 기반을 둔 방법들이 제안되었다[5,6]. 필요한 명령어가 필터 캐쉬에 없고 L1 캐쉬에 있다고 예측될 경우 필터 캐쉬를 경유 하지 않고 바로 L1 캐쉬를 접근함으로써 캐쉬 접근 시간을 최소화 하고자 한다. 이외에도 loop 캐쉬나 tiny 캐쉬와 같은 유사한 연구가 있어왔다[7,8]. 이들 연구 모두 명령어 캐쉬만을 대상으로 하였다. 이에 반해 본 논문은 이를 데이터 캐쉬로 확장하려고 한다. 명령어는 대부분의 경우 순차적으로 접근되어 쉽게 예측이 가능하지만 데이터의 경우 다양하고 불규칙한 패턴을 보여서 예측이 쉽지 않기 때문에 명령어와는 다른 접근법이 요구된다. 데이터 캐쉬와 관련된 유사연구로는 인텔 Strong ARM/XSCALE에서 구현된 미니 데이터 캐쉬(mini data cache)이다. 주 데이터 캐쉬와 별도로 1KB내외의 미니 캐쉬를 두어서 프로파일링에 의해 데이터 스트림을 주 데이터 캐쉬와 미니 캐쉬로 정적으로 나누고 이 정보를 페이지 테이블에 저장한다. 이들 두 캐쉬는 메모리가 접근 될 때 동시에(parallel) 접근된다. 따라서 에너지 및 접근 속도 측면에서는 이득이 없고 데이터 캐쉬에서의 충돌(conflicts) 혹은 오염(pollution)을 감소시켜 데이터 캐쉬 미스율을 감소시키고 성능 향상을 이루는데 주로 활용이 된다. 이에 반해 본 논문에서 제안하는 2-레벨 캐쉬는 데이터 캐쉬의 에너지 및 접근 속도를 줄이기 위해 L1 데이터 캐쉬와 L2 캐쉬가 순차적으로 접근된다. 또한, 미니 캐쉬의 경우 데이터를 미니 캐쉬나 데이터 캐쉬 어느 곳에 위치시킬지 결정하기 위해 어플리케이션을 프로파일링 해야 하며 이 결과를 적용하기 위해 컴파일러의 도움이 필요하지만 본 논문에서 제안된 캐쉬 구조의 경우 어플리케이션 레벨의 도움을 필요로 하지 않는다.

집합연관사상(set-associative)을 사용한 캐쉬의 에너지 소비를 줄이기 위해 다양한 웨이 예측(way prediction) 기법들이 제안되었다. Reactive associative 기법[9]에서는 매번 캐쉬로 접근할 때 직접사상으로 접근할 것인지 집합연관사상으로 접근할 것인지를 결정하는 피드백(feedback) 메커니즘이 사용되었다. 오직 충돌을 일으키는 블록들만 집합연관사상을 필요로 하며 대부분의 블록들은 직접사상으로 접근이 가능하기 때문이다. 비균일(non-uniform) 캐쉬는[10] 각 셋(set)마다 다른 정도의 집합연관사상을 적용하였다. 각각의 셋을 위해 가장 최적화된 숫자의 웨이 개수를 지정하여 에너지 소비 감소를 꾀하였다.

캐쉬의 에너지를 줄이기 위한 또 다른 방법은 오직 하나의 웨이만 접근 하는 방법이다. 명령어를 위한 웨이 배치(way-placement) 기법에서[11] 컴파일러는 주소 비트를 이용하여 가장 빈번하게 접근 되는 명령어를 특정 웨이에 위치시킨다. 따라서 해당 명령어를 접근할 경우 모든 웨이를 비교할 필요 없이 지정된 웨이를 바로 접근 할 수 있다. 웨이 기록(way memoization) 기법은[12] 최근에 사용된 주소와 웨이 번호를 메모리 주소 버퍼(memory address buffer) 구조에 저장한다. 접근 하려는 주소가 메모리 주소 버퍼에서 히트할 경우, 집합연관사상 캐쉬내의 오직 하나의 웨이만 접근 된다. 그러나, 이 기법은 캐쉬가 히트 하는 경우만 적용 가능하며

주소 계산을 위해 32-비트 덧셈기가 항상 사용된다. 반대로, 본 논문에서 제안한 기법은 L1 데이터 캐쉬의 미스를 정확하게 예측하는 것에 초점을 두었으며 유효 주소 계산을 위한 32-bit 덧셈기는 대부분의 경우 사용되지 않는다. 위치 캐쉬(location cache)[13] 기법은 L2 캐쉬를 위해 다음에 사용될 것이라고 예측된 웨이 정보를 저장한다. L1 캐쉬가 미스이고 위치 캐쉬가 히트할 경우 L2 캐쉬의 오직 하나의 웨이만 접근된다. 하지만 위치 캐쉬는 중복된 L2 캐쉬 태그와 위치 캐쉬 자체로 인해 증가된 에너지 소비와 면적 오버헤드(overhead)가 큰 약점을 가지고 있다. 위치 캐쉬의 경우 오직 L1 캐쉬 미스일 경우만 동작하지만 본 논문에서 제안된 지정 웨이 쓰기(one-way write)기법은 모든 쓰기 연산에 동작 가능하다.

Bloom Filter[14] 기법은 명령어의 스케줄링 부담(overhead)을 줄이기 위해서 캐쉬 미스를 미리 결정한다. 멀티 레벨 캐쉬를 위한 다양한 캐쉬 미스 결정 기법들이 제안 되었다. 그러나 제안된 기법들은 L2/L3 캐쉬 미스를 예측하기 위해 파이프라인 단계 중 L1 캐쉬 접근 단계에서 이미 계산된 유효 주소를 사용하기 때문에 예측 동작으로 인해 추가되는 지연을 피할 수 없다. 본 연구에서 제안한 ECP기법은 L1 캐쉬의 미스를 완전한 유효주소의 계산 없이 실행 단계에서 예측가능하다.

빠른 주소 변환[15,16] 기법은 읽기 응답시간을 줄이기 위해 제안되었다. 파이프라인의 decode 단계에서, 셋(set) 인덱스 값이 간단한 OR 연산을 통해 계산되며 데이터 캐쉬는 OR 연산을 통해 계산된 인덱스를 통해 접근 된다. 이와 동시에 블록 오프셋 값과 태그 값이 덧셈기를 이용하여 계산된다. 그러나 이 기법은 정확도가 매우 낮기 때문에(65%) 데이터 캐쉬로의 재접근을 유발하여 결과적으로 에너지 소비를 증가시킨다.

3. 제안된 2-레벨 데이터 캐쉬 구조

3.1 내장형 시스템을 위한 캐쉬 설계

캐쉬의 접근 시간(access time)과 전력소비는 캐쉬의 크기, 연관사상 정도(associativity), 포트(port)수에 비례한다. 따라서 빠른 접근 시간을 가지며 저전력을 소비하는 캐쉬 메모리는 크기가 작고 연관 사상 정도가 낮으며 적은 포트 수를 가져야 한다. 그러나 작고 연관사상 정도가 낮은 캐쉬 메모리는 캐쉬 미스율(miss rate)을 증가시켜서 시스템 전체측면에서 성능을 저하시키고 전력 소비를 증가시킬 수 있다. 따라서 전력소비를 증가시키지 않으면서 고성능을 낼 수 있는 적당한 크기와 연관사상 정도를 가지는 캐쉬 메모리가 주로 사용이 되고 있다. 예를 들면 8KB에서 64KB의 크기를 가지며 1-way에서 16-way의 연관사상이 주로 사용되고 있다[2,17-19]. 프로세서 설계자들은 작은 구현 기술(small technology)을 사용함으로써 회로의 지연시간(delay)과 전력소비를 줄여왔다[20]. 그러나 캐쉬 메모리를 구현하는데 사용되는 SRAM은 이러한 technology scaling에 의해서 접근시간이 잘 확장되지 않는 특성을 지닌다. 예를 들면, 펜티엄 프로세서에서 명령어 캐쉬(instruction cache)를 접근하는데 1사이클이 소요됐으나 펜티엄프로와 펜티엄3에서는 2사이클이, 펜티엄 4에서는 4사이클이 소요되고 있다[2]. 따라서 고성능을 위해 주 메모리(main memory) 접근회수를 줄이기 위해서는 거대한 캐쉬 메모리가 필요하지만 이는 복수의 접근 사이클이 필요해 결국은 성능 향상이 이루어지지 않을 수 있다. 따라서 technology scaling에 의해서 내장형 시스템의 성능 향상을 이루고자 할 경우 혹은 클럭 주파수를 증가시키기 위해 파이프라인 수를 증가시키고자 할 경우 캐쉬 메모리가 가장 큰 걸림돌이 된다.

본 논문에서는 해결책을 고성능 시스템에서 사용된 방법에서 힌트를 얻고자 한다. 고성능 데스크톱이나 서버 시스템에서는 2-레벨의 캐쉬 메모리 혹은 그 이상의 레벨 구조가 보편적으로 사용되고 있다[18,19]. 현재의 고성능 컴퓨터 시스템에서 프로세서의 클럭 사이클은 일반적으로 1ns 이하이다. 그러나 주 메모리를 접근하기 위해서는 100ns에 가까운 시간이 걸리고 있다. 따라서 한번 캐쉬 메모리에서 미스가 나게 되면 적어도 100사이클 이상의 페널티(penalty)가 주어지게 된다. 이처럼 2-레벨 캐쉬 메모리는 점증하는 프로세서와 주 메모리 시스템 사이의 성능차이를 완화시키기 위해 도입되었다. 그러나 내장형 프로세서들은 일반적으로 50MHz에서 800MHz의 낮은 클럭 주파수에서 동작하기 때문에 주 메모리 접근 시간이 상대적으로 낮아서 아직은 대체로 2-레벨의 캐쉬 메모리는 채택이 되지 않고 있다[17]. 고성능 시스템에서 사용되는 2-레벨 캐쉬 메모리 구조에서는 L1 캐쉬와 L2 캐쉬의 목적이나 특징이 상이하다. L1 캐쉬는 대부분의 메모리 접근을 만족 시켜줄 수 있는 만큼의 크기를 가지면서 동시에 빠른 접근속도를 필요로 한다. 일반적으로 L1 캐쉬는 8KB에서 64KB의 크기를 갖는다[2]. 반대로 L2 캐쉬는 주 메모리 접근 횟수를 최소화시키기 위해서 큰 구조를 갖는다. 예를 들면 펜티엄 4의 경우 256KB이상 크기의 L2 캐쉬를 가지고 있다[19]. 그러나 고성능 컴퓨터 시스템에서 사용되고 있는 2-레벨 캐쉬 메모리는 내장형 시스템에서는 채택하기가 어렵다. 일반적으로 내장형 시스템에서는 주 메모리 접근 페널티가 위에서 언급했듯이 상대적으로 작고 저비용으로 구현이 되기 때문에 이런 고비용의 넓은 칩면적을 필요로 하고 고전력을 소비하는 2-레벨 캐쉬 메모리 구조는 적합하지 않다[21]. 최근에 개발된 3세대

인텔 XSCALE 프로세서에서는 고주파수에서 높은 주 메모리 접근 페널티를 줄이기 위해 256KB혹은 512KB의 대형 L2 캐쉬를 지원하고 있지만 본 논문에서 다루고자 하는 2-레벨 캐쉬 구조와는 상이한 특징을 가진다.

3.2 2-레벨 데이터 캐쉬

그림 1(왼쪽)은 일반적으로 내장형 시스템에 많이 채용되고 있는 하바드(Harvard) 캐쉬 구조를 보여준다. 즉, 명령어와 데이터를 동시에 서로 다른 분리된 메모리 시스템에서 읽고 쓸 수 있다. 따라서 L1에는 명령어 캐쉬(instruction cache)와 데이터 캐쉬(data cache)가 독립적으로 존재한다. 이런 기존의 일반적인 집합연관사상 1-레벨 캐쉬는 세 가지의 단점을 가지고 있다.

첫째로, 집합연관 캐쉬는 긴 읽기 시간을 유발한다. 이것은 주로 모든 캐쉬 웨이를 읽은 후에 태그(tag)를 비교하고 마지막으로 요청된 한 웨이를 선택하기 때문이다. 예를 들어 XScale은 캐쉬 읽기 연산을 위해 2사이클을 요구한다. 두번째로, 집합연관의 쓰기 처리량(bandwidth) 이 직접사상(바로쓰기 정책과 결합)의 쓰기 처리량보다 낮다. 집합연관사상 캐쉬에서는 실제 쓰기 연산이 하나의 웨이에 수행되기 전에 요청된 어드레스를 가지고 있는 특정웨이가 반드시 선택되어야 한다. 따라서 쓰기 연산은 읽기 연산 보다 최소한 한 사이클을 더 필요로 한다. 이러한 집합연관 캐쉬는 쓰기 연산이 완료되기 전에 다른 캐쉬로의 요청을 수행할 수 없기 때문에 쓰기 처리량을 감소시키고 읽기 연산을 지연시킨다. 세번째로, 집합연관 캐쉬는 읽기 연산을 수행할 때 히트 시간을 줄이기 위해 다수의 캐쉬 웨이가 동시에 접근되어야 하기 때문에 필연적으로 많은 전력을 소비한다.

본 논문에서는 이러한 단점을 극복하기 위해 그림 1(오른쪽)과 같은 2-레벨 데이터 캐쉬 구조를 제안하였다. 제안된 캐쉬 구조에서 L1 캐쉬의 목적은 접근시간과 전력소비를 최소화하는 것이다. 따라서 작은 크기의 (1KB) 직접사상을(direct-mapped) 사용하는 캐쉬를 L1 캐쉬의 구조로 채용하였다. 작은 크기의 캐쉬는 앞에서도 언급했듯이 빠른 접근속도를 가지고 저전력을 소비한다. 또한 작은 크기의 L1 캐쉬는 높은 CPU 클럭 주파수에서도 한 사이클의 접근 시간을 가정할 수 있다. 쓰기 정책(write policy)에 따라 캐쉬는 바로쓰기(write-through) 캐쉬 및 나중쓰기(write-back) 캐쉬로 나뉜다[2].

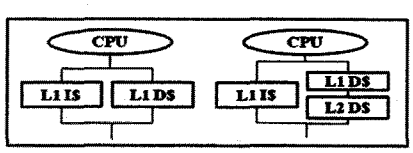

그림 1 일반적인 1-레벨 데이터 캐쉬 구조(왼쪽)와 제안된 2-레벨 데이터 캐쉬 구조(오른쪽)

L1 캐쉬는 이 두 정책 중 하나를 채택할 수 있다. 바로쓰기 캐쉬는 쓰기 명령어의 빠른 실행 속도를 위해 일반적으로 쓰기 버퍼와 함께 사용되며 캐쉬내의 블록이 모두 클린(clean)한 상태이기 때문에 블록 교체시에도 빠른 속도를 보장한다. 하지만, 각 쓰기 접근에 대해 하위 메모리 구조로의 쓰기를 유발하여 에너지소비를 증가시킨다. 반대로 나중쓰기 캐쉬는 하위 메모리로의 접근을 줄여 에너지소비를 감소시키지만 캐쉬 구조를 복잡하게 만들며 캐쉬 미스시에 더티(dirty) 블록을 교체하고 새로운 블록을 읽어야 하기 때문에 접근 시간을 증가시킨다[3]. 본 논문에서는 직접사상 캐쉬의 빠른 접근시간을 최대화하기 위해 L1 캐쉬는 바로 쓰기 정책을 기반으로 하였다. 직접사상과 바로쓰기 정책을 채용한 캐쉬는 웨이의 수가 하나뿐이고 캐쉬 라인은 항상 클린(clean)한 상태이기 때문에 쓰기 미스시에 웨이를 선택하기 위한 과정 없이 안전하게 덮어 쓰일 수 있기 때문에 쓰기연산을 빠르게(한 사이클 이내) 마칠 수 있다. 또한, 표 1과 같이 작은 크기이며 직접사상을 채용한 캐쉬는 에너지 소비 측면에서 효율적이다. L2 캐쉬의 크기는 일반적인 1-레벨 캐쉬에서의 L1 캐쉬 크기로 맞추었다. L2 캐쉬는 메모리로의 접근을 최소화하기 위해서 나중쓰기 정책이 채용되었다. L1 캐쉬와 L2 캐쉬 사이의 쓰기 트래픽을 줄이고 쓰기 합병(write merging)을 유도하기 위해서 두 캐쉬 사이에 쓰기 버퍼가 사용되었다.

3.3 Early Cache hit Predictor (ECP) 기법

제안된 2-레벨 데이터 캐쉬 구조는 L1 데이터 캐쉬에서 작은 크기와 직접사상을 채용하여 빠른 접근 시간과 낮은 에너지 소비를 보여주지만, 같은 이유로 인해 캐쉬의 용량 부족과 메모리 주소가 한 인덱스에 충돌할 수 있는 확률이 높아졌기 때문에 높은 미스율(miss rate)을 보일 수 있다. 위와 같은 이유로 증가한 캐쉬 미스들은 요청된 데이터를 찾기 위해 L2캐쉬를 접근해야 하기 때문에 추가 사이클을 유발하여 프로세서 전체 성능에 악영향을 미칠 수 있다. 본 논문에서는 이러한 문제를 해결하기 위해 ECP(Early Cache hit Predictor)라고 명명한 기법을 제안하였다. 제안된 ECP 기법을 사용하면 L1 데이터 캐쉬로의 접근이 히트할 것인지 미스할 것인지에 대한 예측이 가능하다. 만약 ECP의 결과가 미스가 날것이라고 예측되면 L1 데이터 캐쉬로의 접근은 수행되지 않고 바로 L2 데이터 캐쉬로의 접근이 수행된다. 따라서 L1캐쉬에서 발생하는 미스로 인한 추가 사이클의 발생을 사전에 방지할 수 있다.

L1 데이터 캐쉬를 정확하게 예측하기 위해서는 유효주소(effective memory address)가 필요하다. 이는 예측 과정이 유효주소 계산 이후에 배치되는 것으로 제한되어야 함을 의미한다. 일반적인 컴퓨팅 환경에서 유효

표 1 180nm 공정의 CACTI 시뮬레이터로부터 구한 캐쉬 에너지 소비

구성	1K/1W	2K/1W	8K/4W	16K/4W	32K/4W
에너지(pJ)	320	330	904	939	995
1W-에너지(pJ)			421	467	505

1W-에너지는 지정웨이 쓰기 기법이 적용되었을 때의 에너지 소비를 나타냄.

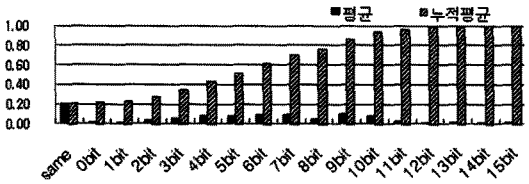

그림 2 비트 위치에 따른 기본 주소와 유효주소의 차이

주소 계산은 파이프라인 단계 중 Execution 단계에서 이루어지고 계산된 유효 주소는 데이터 캐쉬 접근 단계에서 사용된다. 즉, 기본 파이프 라인 구조를 변경하지 않고 예측동작이 수행 가능한 단계는 Execution 단계 이후에 가능하다. 그러나 ECP 기법이 데이터 캐쉬 접근 단계에 배치된다면 데이터 캐쉬를 접근하기 전에 예측 동작으로 인한 추가적인 지연을 피하기가 어렵다. 예측을 수행하는 동작으로 인한 지연시간을 줄이기 위해 ECP 기법은 전체 파이프라인 단계 중 Execution 단계에 배치되었다. ALU연산의 결과로 계산된 유효주소를 사용하여 예측동작을 수행하는 것 역시 Execution단계에서 추가적인 지연시간을 피하기 어렵기 때문에 유효주소계산을 빠르게 수행하기 위해 다음과 같은 관찰 결과가 사용되었다.

유효주소는 기본 어드레스(base address)에 오프셋(offset)을 더하거나 빼는 연산을 통해 계산된다. ECP기법은 대부분의 경우 메모리 명령어의 기본 어드레스(base address)에 더해지거나 빼지는 오프셋(offset)의 크기가 작다는 원리를 이용하였다. 그림 2는 실험된 벤치마크에서 계산전의 기본 어드레스와 오프셋 비트가 계산 된 이후의 실제 유효 주소(effective memory address)간의 비트위치에 대한 차이의 평균을 보여준다. 예를 들어, 그림 2의 '0 bit'는 기본 주소와 유효주소간의 차이가 오직 LSB비트에서만 나타날 때(즉, 1st부터 MSB까지의 비트들은 모두 같은 경우)의 비율이다. 오프셋의 크기가 작기 때문에 기본 주소와 유효주소는 12번째 이상의 비트위치에서 거의 같은 값(평균 98.5%)을 갖고 있다. 그러므로 전체 32비트의 더하기나 빼기 연산 수행 없이 부분적인 기본 주소와 오프셋 주소의 연산만으로 대부분의 유효주소를 계산하는 것이 가능하다.

이러한 관찰을 토대로 본 논문에서는 그림 3과 같이 ECP를 설계하였다. 제안된 ECP 기법은 거의 대부분의 경우에 정확한 유효주소를 계산할 수 있다. 오프셋 크기가 ECP의 계산 가능한 범위(그림 3은 13bit)를 초과하는 경우에 한해 32비트 전체 ALU 연산이 필요하다.

이러한 이유 때문에 메모리 명령어의 주소를 계산하기 위한 실행(execution) 단계에서 에너지 소비를 줄이기 위해서 ALU가 비활성화 된다. 제안된 ECP로 계산할 수 없는 유효주소를 감지하기 위해 큰 크기의 오프셋이나 더하기 연산의 결과로 캐리(carry)가 발생하는 경우를 감지할 수 있는 부분이 추가되었다(그림 3의 사선으로 채색된 부분). ECP 기법으로 계산할 수 없는 유효주소가 감지된 경우 파이프라인의 실행 단계는 stall되고 다음 사이클에서 유효주소를 계산하기 위해서 32비트 ALU가 활성화된다. 이런 경우에는 정확한 유효주소를 계산하기 위해서 추가적으로 한 사이클을 더 필요로 한다.

본 논문에서 기본 프로세서로 채용한 XSCALE에서 메모리 명령어의 오프셋 값은 항상 양수 값이다. 빼기 연산을 수행하기 위해서는 양수 값을 2의 보수로 변환하는 과정이 필요하다. 그림 3의 왼쪽 상단에 오프셋 레지스터에서부터 회색의 선택기(mux)로 가는 점선들은 빼기 연산을 위해 2의 보수로 변환된 오프셋을 나타낸다. ECP 기법에서는 유효주소 계산을 위해서 3개의 RCA(Ripple Carry Adder)가 사용되었다. RCA는 Carry lookahead adder와 같은 복잡한 형태의 덧셈기보다 에너지측면에서 효율적이다. 하지만, 바이트 오프셋과 인

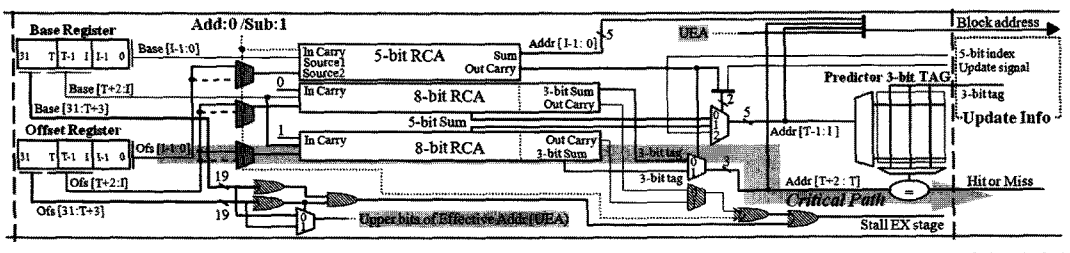

그림 3 제안된 3비트 태그 Early Cache hit Predictor. 회색으로 채색된 선택기(mux)들은 더하기/빼기 연산을 선택하기 위해 사용되었다. 사선으로 채색된 논리소자들은 Execution 단계의 stall을 결정하기 위해 사용되었다.

덱스 그리고 하위 태그 비트들의 연산을 하나의 RCA를 사용하여 순차적으로 계산할 경우 연산속도 측면에서 효율적이지 않기 때문에 조건부 가산기(Conditional Sum Adder)에서 사용된 원리를 응용하여 연산과정을 부분적으로 나누었다. 하나의 세트(8-비트 RCA)는 입력되는 바이트 오프셋의 캐리를 '0'으로 가정하고 다른 하나는 입력되는 캐리를 '1'로 가정하여 두 세트의 출력을 만들어 두고 5-비트 RCA에 의해 계산되는 바이트 오프셋의 캐리의 값이 결정될 때 캐리 전달에 의한 지연 없이 한 출력의 세트를 선택하는 구조이다. 즉, 바이트 오프셋 주소를 계산하기 위해 5비트 RCA가 사용되었으며 인덱스/태그 주소를 계산하기 위해 두 개의 8비트 RCA가 사용되었다. 따라서 두 개의 8비트 RCA는 5비트 RCA와 평행하게 연산을 수행하게 된다. 5비트 RCA의 계산결과는 선택된 8비트 RCA의 연산 결과 중 먼저 계산된 5비트의 연산 결과를 사용해 예측기(predictor)를 인덱스 하는 데 사용된다. 8비트 RCA의 나머지 3비트는 예측기내의 태그 비트들과 비교된다. ECP안의 예측기는 오직 태그비트로만 구성되어 있다. 예측기에 저장된 태그비트는 L1 데이터 캐쉬의 하위 태그 비트들이며 데이터가 L1 데이터 캐쉬에 로드될 때 업데이트된다. 따라서 예측기는 현재 L1 데이터 캐쉬의 하위 태그비트와 같은 값을 항상 유지한다. 인덱스된 예측기의 태그 값들을 읽는 연산은 8비트 RCA에서 남은 3비트를 계산하는 연산과 평행(parallel)하게 수행된다. 예측기내의 태그값 중 하나의 값이 일치하는 경우 예측기는 L1 캐쉬에서 히트가 발생할 것이라고 예측을 한다. 예측기가 현재 L1 데이터 캐쉬의 하위 태그비트를 저장하고 있기 때문에 히트라고 예측할 경우 실제로는 태그의 상위 값이 달라서 L1 데이터 캐쉬에서 미스가 발생할 수 있다. 하지만 만약 예측기내에 일치하는 태그 값이 존재하지 않는다면 예측기는 미스를 예측하고 이 경우에는 하위비트가 일치하지 않았기 때문에 확실하게 미스를 보장할 수 있다.

ECP 기법에서 사용된 예측기로 인해 유발될 수 있는 접근시간, 에너지 소비, 필요면적에 관한 오버헤드는 예측기에 저장되는 태그비트 수에 의해 결정된다. 저장되는 태그비트의 숫자가 증가할 경우 인덱스와 태그비트를 계산하기 위한 RCA의 크기와 예측기의 에너지 소비가 증가한다. 본 논문에서는 예측기에 저장되는 태그비트 수를 3, 4, 5로 나누어 실험을 수행하였다.

3.4 지정웨이 쓰기(one-way write)기법

L1 데이터 캐쉬에서 채용한 바로쓰기(write-through) 정책으로 인해 각 메모리 쓰기 명령어마다 L2 데이터 캐쉬에 접근하게 되어 L2 데이터 캐쉬의 에너지를 소비가 증가하게 된다. 이러한 증가된 에너지 소비를 경감시키기 위해서 지정웨이 쓰기라고 명명한 기법을 제안하였다. 그림 4는 제안된 지정웨이 쓰기 기법을 보여주고 있다.

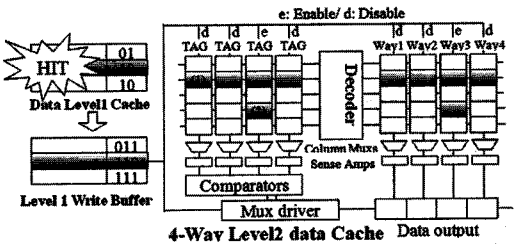

그림 4 지정웨이 쓰기 기법의 블록 다이어그램. (1)은 L2캐쉬가 집합연관으로 접근될 때 활성화되는 부분을 나타내고 (2)는 지정웨이 쓰기 기법이 적용되었을 때를 나타낸다.

L1 캐쉬와 쓰기 버퍼(write buffer)에 2-3비트가 추가되었으며 추가된 비트는 쓰기가 수행될 데이터가 저장되어 있는 L2 캐쉬의 웨이 번호를 나타내고 있다. L1 캐쉬의 각각의 블록에는 2비트가 추가되었고 쓰기 버퍼의 각각의 블록에는 3비트가 추가되었다. 추가된 비트는 다음과 같은 과정을 통해 사용된다. 데이터가 L2 캐쉬에서 L1 캐쉬로 로드될 때 해당 데이터가 저장된 L2 캐쉬의 웨이 번호가 L1 캐쉬의 추가된 2비트에 저장된다.

지정웨이 쓰기기법으로 인한 L2 캐쉬로의 쓰기과정에서 L1 데이터 캐쉬에는 데이터가 있지만 L2 캐쉬에는 해당 되는 캐쉬 블럭이 존재하지 않을 수 있는 상황을 배제하기 위해 본 논문에서는 L1 캐쉬와 L2 캐쉬 간에 포함관계(inclusive property)를 가정하였다. 즉, L2캐쉬에서 하나의 블록이 교체될 때, L1 캐쉬에 해당되는 데이터가 있다면 그 데이터는 무효(invalid)상태로 표시되게 된다. 따라서 모든 경우에 대해 L1 캐쉬에서 쓰기연산이 히트한 경우 히트한 데이터는 반드시 추가된 2비트가 가리키는 L2 캐쉬의 웨이 위치에 존재하게 된다. 이러한 경우에 L2 캐쉬에서 쓰기연산에 대한 태그 비교연산이 불필요하기 때문에 태그 비교연산이 생략되어 쓰기연산에 대한 에너지 소비를 줄일 수 있다. 만약 L1 캐쉬에서 쓰기 연산이 미스한 경우 쓰기연산이 수행될 데이터는 no write allocation을 가정했기 때문에 쓰기 버퍼에만 쓰이게 된다. 쓰기버퍼에서 L2 캐쉬로 쓰기 연산을 수행할 때 지정웨이 쓰기가 수행 되어야 할지 여부를 나타내기 위해서 L1 캐쉬 블록에 추가된 2비트 보다 1비트 많은 3비트가 쓰기 버퍼에 사용되었다. 만약 3비트중 가장 왼쪽비트가 0의 값을 가지고 있다면 L1 캐쉬에서 미스가 발생했고 L2 캐쉬는 보통의 집합연관사상 방법으로 접근되어야 하는 것을 나타낸다. 만약 1의 값이라면 L1 캐쉬에서 쓰기 연산이 히트가 발생

했기 때문에 나머지 2비트가 가리키는 L2 캐쉬내의 하나의 웨이만 접근되어야 하는 것을 나타낸다.

L1 캐쉬와 쓰기 버퍼에 추가된 비트로 인한 오버헤드는 아래의 식처럼 계산될 수 있다. 1KB의 L1 캐쉬는 32바이트의 블록 크기를 가정한 경우 32개의 셋을 가지고 있으며 쓰기버퍼는 8개의 엔트리를 가지고 있다. 본 논문에서 구성한 캐쉬의 경우 지정웨이 쓰기를 위해서 총 88비트가 추가 되었으며 이를 캐쉬 크기에 비교해보면 매우 적은 비트이다. L2 캐쉬의 연관(associative) 정도가 증가하더라도 L1 캐쉬에 필요한 추가비트는 LOG 배로 증가 하기 때문에 매우 적은 증가량을 보여준다. 예를 들어 L2 캐쉬의 연관 정도가 16이라면 L1캐쉬의 각 셋(set)에 필요한 추가비트는 4-비트가 될 것이다.

*Log2(# of L2 cache ways) * # of L1 data cache sets + (Log2(# of L2 cache ways) + 1) * # of write-buffer entries*

4. 결 과

4.1 실험 환경

본 논문에서 사용한 기본 프로세서는 인텔사의 XSCALE이다. XSCALE은 내장형 시스템을 목적으로 고성능과 저전력 소비를 위해 설계된 프로세서이다. XSCALE은 7-8 단계의 싱글-이슈 수퍼파이프라인 코어를 채용했으며 32KB의 32웨이 집합연관사상의 인스트럭션 캐쉬와 데이터 캐쉬를 가지고 있다. 최근 출시된 3세대 XSCALE의 경우 같은 크기의 4웨이 명령어 캐쉬와 데이터 캐쉬를 채용하였으며 본 논문에서는 3세대 XSCALE을 기본 프로세서로 적용하였다.

제안된 2-레벨 캐쉬 구조에 대한 성능 평가 및 실험은 시뮬레이션을 기반으로 하여 이루어졌다. 시뮬레이션을 통한 방법은 실제 프로토타입이나 최종 제품이 만들어 지기 전에 제안되는 캐쉬 구조의 성능을 상당히 정확하게 평가할 수 있고 구현상의 복잡도 및 문제점 등을 미리 알 수 있는 장점이 있다. 실제로 학계 및 산업계에서 이러한 시뮬레이션을 통한 성능측정 방법이 널리 사용되고 있다. 본 논문에 가장 적합한 시뮬레이터로 최근에 개발된 XTREM이 있다[22]. 이 시뮬레이터는 Simplescalar-ARM에 기반을 두고 인텔 XSCALE 아키텍처를 모델링하고 있다. XTREM은 사이클 단위로 성능 측정이 가능한 Sim-XSCALE과 Wattch를[23] 기반으로 하는 파워 모델로 구성되어 있다. XTREM시뮬레이터의 평가 결과 성능 및 전력소비 측정에서 실제 시스템과 비교해 5% 정도의 오차를 내는 것으로 알려져 있다. XTREM 시뮬레이터의 출력으로 다양한 정보들이 제공된다. 예를 들면, CPI(Cycles Per Instruction), 수행된 총 명령어 수, 메모리 접근 명령어 수, L1 캐쉬 히트율/미스율, L2 캐쉬 히트율/미스율, TLB 히트율/미스율, 총 수행 사이클 등이다. 이들 정보를 이용해서 제안된 2-레벨 캐쉬 구조의 성능 및 전체 시스템에 미치는 영향 등을 측정할 수 있다. XTREM에서 사용한 캐쉬 파워 모델은 높은 연관 캐쉬를 위한 CAM-based 모델로서 제안된 캐쉬 구조와 맞지 않기 때문에 에너지 소비를 계산하기 위해 캐쉬 메모리를 모델링한 CACTI를[21] 이용하여 전력소비 데이터를 구하였다. 또한, 제안된 캐쉬 메모리 설계에서 사용된 논리소자를 모델링 할 수 있는 Synopsis design compiler가[24] 본 연구에서 사용되었다. 이들 툴을 이용해 다양한 구조의 크기를 가지는 캐쉬 메모리의 접근시간, 필요면적 및 전력소비를 서로 다른 공정기술(process technology)에 대해서 측정할 수 있다. 이 전력소비 데이터를 XTREM 시뮬레이터 상에 프로그램한 뒤 시뮬레이션을 통해 제안된 2-레벨 캐쉬 구조의 전력소비를 측정하였다. 제안된 2-레벨의 캐쉬 구조들의 성능 및 전력소비를 실험하기 위해 Mibench와[25] MPEG2 디코더를 벤치마크로 선택하였다. 이들 벤치마크는 현재 내장형 시스템의 성능 및 전력소비, 최적화 등을 위해서 널리 사용되고 있다. 본 논문에서 제안된 2-레벨의 캐쉬 메모리 구조들이 시뮬레이터 상에서 구현되고 성능 및 전력소비가 평가되었다. 표 2는 실험에서 사용된 기본 프로세서의 변수 값들을 보여준다.

표 2 기본 프로세서의 변수 값들

변수	값
프로세서 주파수	200Mhz
파이프라인 단계 수	7, 8 Stages
캐쉬 구성	8KB or 16KB 4-way 32-Byte Block size
명령어 TLB, 데이터 TLB	32-Entries Fully Associative
쓰기 버퍼	8 entries
주 메모리 버스	32-bit wide
주 메모리 지연시간	18 cycles or 90 ns
캐쉬 지연시간	Load (2cycles) / Store (3cycles)

4.2 실험 결과

Synopsys design compiler(TSMC 0.18um standard cell library)를[24] 사용하여 얻은 ECP의 각 요소별 지연시간, 에너지소비, 필요면적이 표 3에 나타나 있다. 제안된 ECP 기법에서 L1 데이터 캐쉬의 히트/미스 예측과 유효주소 계산이 execution 파이프라인 단계에서 수행되기 때문에 크리티컬 패스의 지연시간이 중요하다. 그림 3에서 가장 긴 지연시간을 보일 것으로 판단되는 경로를 크리티컬 패스로 지정하고 시뮬레이션 실험을 통해 3-비트 크기의 예측기(predictor)를 가지는 ECP기법의 크리티컬 패스의 지연시간을 계산하였다. 크리티컬

표 3 ECP에서 사용된 요소들의 접근시간, 면적, 에너지 소비

요소	지연시간(ns)	에너지(pJ)	면적(um^2)
5-비트 RCA	0.62	0.6566	1036
8-비트 RCA	0.97	1.0023	1844
1-비트 MUX	0.12	0.54	64
3-비트 MUX	0.12	0.202	192
5-비트 MUX	0.12	0.393	320
8-비트 MUX	0.12	0.53	512
19-비트 MUX	0.12	1.386	1216
예측기(predictor)	0.3	24	14400
3-비트 비교기	0.12	0.7	300
감지(detection) 부분	0.43(pure gate)	0.629	968
ECP을 위한 총합	1.33	31.9642	23528
32-비트 ALU	1.21	192.39	105517

Critical Path = Mux(0.12)+8-비트 RCA(0.97)+Mux(0.12)+비교기 (0.12)
에너지&면적 = 1-비트 Mux + 5-비트 Muxs*2 + 8-비트 Mux*2 + 5-비트 RCA + 8-비트 RCAs*2 + 3-비트 Mux + 19-비트 Mux + 예측기 + 비교기 + 감지부분

패스를 결정하기 위해서ECP기법 안에서 가장 큰 지연시간을 가질 수 있는 두 가지 경로를 비교하였다. 첫 번째 경로는 8-비트 RCA에서 5-비트 중간 결과가 5-비트 MUX를 거쳐서 예측기(predictor)를 인덱스하고 3-비트 비교기의 결과가 나오는 경우이다. 두 번째 경로는 8-비트 RCA의 연산이 끝난 후 3-비트 태그값이 3-비트 MUX를 통해 3-비트 비교기의 결과를 출력하는 경우이다. 실험을 통해 계산한 각각의 크리티컬 패스 지연시간은 첫 번째 경우가 1.28ns이고 두 번째 경우가 1.33ns이다. 32-비트 ALU ECP기법에서 사용된 RCA (Ripple Carry Adder)는 에너지 소비와 공간(area)적인 측면에서 CLA(Carry Lookahead Adder)에 비해 월등히 뛰어나지만 하나의 비트씩 순차적으로 계산이 되어야 하기 때문에 속도가 느리다는 단점을 가지고 있다. ECP기법에서는 RCA의 문제점을 극복하기 위해 그림 3과 같이 부분 유효주소 계산을 중첩하여 RCA로 인한 Delay 감소를 최소화하였다. 실험에서 사용된 Synopsis design compiler는 매우 다양한 라이브러리(library)를 가지고 있으며 각 라이브러리 마다 다른 결과를 보여줄 수 있다.

ECP의 지연시간을 ALU의 지연 시간과 비교하기 위해 32비트 CLA를 Synopsys design compiler를 이용해서 합성하였다. ALU의 지연시간은 표 3에도 나타난 것과 같이 1.2ns로 측정되었다. 따라서 ECP 기법으로 인해 CPU의 클럭 사이클 시간이 증가하는 것으로 예상할 수 있지만 ECP의 지연시간 중 가장 긴 부분을 차지하는 RCA의 경우 선행 논문에[26] 제안된 것과 같은 진보된 RCA를 사용한다면 매우 적은 양의 에너지 소비를 동반하여 RCA의 지연시간이 50%까지 줄어들게 된다. 이는 ECP의 크리티컬 패스의 지연시간이 ECP 내부 요소의 선택에 따라서 충분히 ALU의 지연지간 보다 작아질 수 있음을 의미한다. ECP의 에너지 소비와 면적에 대한 총 오버헤드는 32비트 ALU와 비교했을 때 16.6%와 22.3%이다.

표 4는 종합적인 벤치마크 시뮬레이션 결과를 보여주고 있다. 표 4의 세로축은 수행된 각각의 벤치마크를 나타내고 있으며 가로축은 수행된 명령어의 수, 성능 향상 비율, 캐쉬의 미스율, ECP 기법의 미스율을 나타내고 있다. 실험 결과를 비교하기 위해서 아무 구조적 변화를 취하지 않은 1-레벨 데이터 캐쉬를 Base라 하였고 2-레벨 데이터 캐쉬의 구조적 변화만을 취한 경우를 *2-level* 이라 하였으며 마지막으로 2-레벨 데이터 캐쉬에 ECP 기법까지 취한 경우를 *ECP-tagX* 라 명칭하였다. 본 논문에서는 3비트 하위 태그 비트를 저장하는 ECP 구조를 기본으로 가정하였다. 2-레벨 데이터 캐쉬

표 4 벤치마크 실험 결과

벤치마크	#명령어 (Mil.)	#읽기 (Mil.)	#쓰기 (Mil.)	사이클 (Mil.) /성능 향상율(%)			미스율(%)		예측실패확률(%)		
				Base	2-level	ECP-tag3	Base	2-level(L1/L2)	Tag3	Tag4	Tag5
Qsort	706.6	89.6	64	1250.5	0.9	0.94	0.87	16.06/0.90	1.92	0.25	0.13
Susan	29.7	6.9	2.8	47.23	4.75	4.65	0.29	4.33/0.31	0.47	0.23	0.10
Jpeg decoder	23.3	7.9	3.2	49.5	4.83	7.85	2.91	33.04/3.46	1.57	1.03	0.30
Lame	1114.4	403	205.2	2589.8	2.5	4.86	3.35	17.12/3.43	1.19	0.27	0.16
Dijkstra	272.6	95.5	21.7	565.75	5.15	5.84	1.58	14.61/1.57	0.84	0.33	0.10
Patricia	640.4	153.1	104.1	2333.1	1.47	1.87	0.62	16.32/0.62	0.64	0.24	0.19
Ispell	817.7	259.1	184.3	2219.7	2.43	3.19	0.9	22.19/0.9	1.99	1.62	1.22
Blowfish	544	207.4	181	1432.3	1.6	1.97	0.07	0.56/0.07	0.18	0.04	0.04
Sha	138.2	23.8	10	182.9	2.22	2.27	0.24	1.55/0.24	0.18	0.01	0.00
Rijndael	319.9	139.9	43.5	706.37	1.25	3.02	1.9	5.56/1.84	3.95	1.11	1.01
Mpeg2 decoder	319.9	107.9	59.7	685.98	4.20	2.76	1.22	13.18/1.18	0.82	0.42	0.14
평균					2.85	3.57	1.27	13.14/1.32	1.25	0.50	0.31

구조에서 L1 데이터 캐쉬의 경우 대부분의 벤치마크에서 캐쉬 미스율이 증가했다. 이는 작은 L1 데이터 캐쉬 크기 때문이다. 그러나 susan이나 sha같은 몇몇의 벤치마크는 매우 낮은 미스율을 나타내었다. 이러한 벤치마크는 작은 데이터 워킹셋 크기를 갖고 있기 때문에 작은 L1 데이터 캐쉬만으로도 충분함을 나타낸다.

성능 향상 측면에서는 작은 크기의 L1 데이터 캐쉬로의 접근이 1사이클이 걸리기 때문에 제안된 2-레벨 캐쉬 구조는 ECP이 적용되지 않은 *2-level* 의 경우 평균적으로 2.85%의 성능 향상이 이루어졌다. 이는 작은 크기의 L1 데이터 캐쉬가 높은 미스율을 보이더라도 작은 크기로부터 얻는 이득이 더 큰 것을 알 수 있다. ECP를 적용했을 경우 L1 데이터 캐쉬의 미스를 더 낮출 수 있기 때문에 미스로 인한 추가 사이클 방지하여 3.6%의 성능향상을 나타내었다. 제안된 ECP 기법은 전체 성능에 두 가지 측면에서 영향을 미칠 있다. 첫째, 미스가 예측되었을 때 L1 데이터 캐쉬를 bypassing하는 때(긍정적)와 둘째, ECP 기법을 통해 계산할 수 없는 유효주소가 탐지되었을 때(부정적)이다. stall된 execution 파이프라인 단계가 얼마나 많은 사이클을 증가시키는지 알아보기 위해 우리는 ALU를 항상 활성화 하는 경우와 ECP 기법을 적용하여 비활성/활성화 하는 경우를 비교하였다. 증가된 평균적인 전체 사이클은 3비트 태그에서 0.25%이다. 대부분의 경우에 유효주소를 ECP 기법을 이용해 계산할 수 있기 때문에 사이클 증가가 매우 작음을 알 수 있다.

표 4에 보인 것과 같이 ECP 기법의 미스율은 대부분의 벤치마크에서 낮은 결과를 나타내었다. 또한 저장되는 태그 크기가 3에서 5로 증가할 때 미스율이 더욱 낮아지는 것으로 나타났다. 그림 5는 ECP 기법의 예측결과를 보다 자세하게 나타내고 있다. ECP 기법은 L1 캐쉬의 하위 태그비트를 저장하고 있기 때문에 미스라고 예측한 경우 L1 캐쉬에서 확실한 미스를 보장할 수 있다. 따라서 히트라고 예측 하였지만 실제로 L1 데이터 캐쉬에서 미스가 발생한 경우(h m')를 예측의 실패라 정의할 수 있다. 그림 5에 나타난 것과 같이 비록 3비트의 하위 태그를 저장하더라도 대부분의 L1 데이터 캐쉬 미스는 ECP 기법을 통해 정확하게 예측되었다. 따라서 본 논문에서는 3비트 ECP를 기본 구조로 채용하였다. ECP기법의 예측결과를 살펴보면 많은 경우에(약 85%)에 캐쉬 히트라고 예측을 하고 실제로 히트하는 경우가 많다. 따라서 히트라고 예측하는 ECP과정이 불필요하게 보일 수 있지만, ECP는 L1 캐쉬 히트 예측과 동시에 유효주소 계산을 대부분의 경우(캐쉬 예측결과와 상관없이) 32비트 ALU를 대신하여 계산하기 때문에 ALU를 disable시키면서 큰 이득을 얻을 수 있다. 32비트 ALU는 표 3에 보인 것과 같이 ECP에서 사용되는 모든 컴포넌트를 합한것의 약 6배의 에너지를 소비한다. 따라서, 벤치마크를 이용한 시뮬레이션 결과에서 ECP기법을 사용할 경우 ALU에서 상당한 에너지 소비의 감소를 예측할 수 있다.

그림 6은 데이터 캐쉬와 ECP 기법에서 사용된 각 요소들 그리고 메모리 주소 연산을 위해 ALU에서 소비된 에너지 소비를 보여준다. 지정웨이 쓰기 기법을 적용했을 때 2-level의 경우 데이터 캐쉬의 평균적인 에너지 소비는 Base에 비교하여 37.7% 감소하였다. 지정웨이 쓰기와 ECP 기법을 동시에 적용했을 때, 2-레벨 데이터 캐쉬의 평균적인 에너지소비는 Base에 비교하여 41% 감소하였으며 메 모리 연산을 위한 ALU의 경우 에너지 소비는 평균 97% 감소하였다. 종합적으로 메모리 연산을 위한 ALU 에너지 소비 감소와 3비트 태그 ECP, 지정웨이 쓰기 기법을 적용하였을 때 50%가 평균적으로 감소하였다. 따라서 본 논문에서 제안한 2-레벨 데이터 캐쉬 구조와 제안된 기법들을 적용할 경우 전체

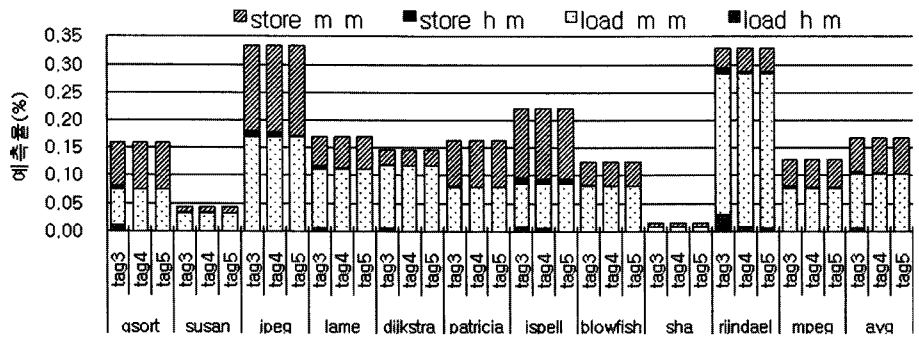

그림 5 읽기 연산과 쓰기 연산에 따른 ECP의 미스율. 'h m'은 예측기가 히트라고 예측했지만 L1 데이터 캐쉬에서 미스가 발생한 경우. 'm m'은 예측기가 미스라고 예측하고 실제로 L1 캐쉬에 데이터가 없는 경우를 나타낸다. 'h h'는 각 막대의 나머지 부분을 나타낸다.

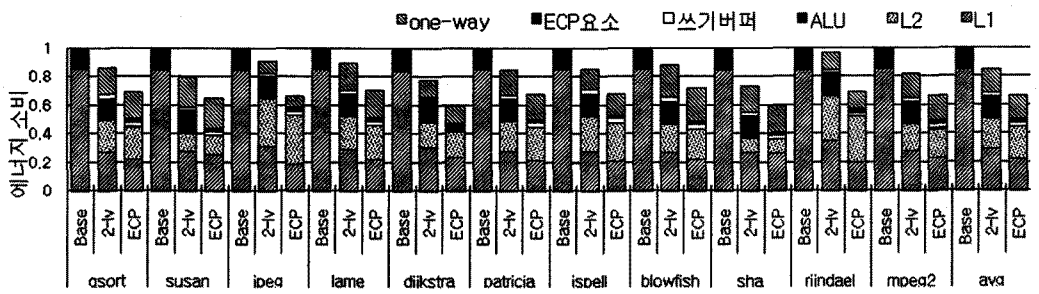

그림 6 각 캐쉬 구조들의 에너지 소비가 1-레벨 기본 캐쉬 구조로 정규화 되었다. "one-way"부분은 지정웨이 쓰기 기법을 적용하였을 때 L2 데이터 캐쉬에서 감소되는 에너지 소비를 나타낸다. ECP에서는 3비트 태그 크기를 사용하였다.

적인 성능향상과 함께 데이터 캐쉬의 에너지 소비와 메모리 주소 계산을 위한 ALU에너지 소비가 상당히 감소되는 것을 확인할 수 있다.

5. 결 론

본 논문에서는 에너지-성능 측면에서 효율적인 2-레벨 데이터 캐쉬 구조를 제안하였다. 제안된 2-레벨 데이터 캐쉬 구조는 작은 크기의 L1 캐쉬의 크기로 인하여 낮은 지연시간과 클럭 사이클을 높이는데 용이한 빠른 사이클 시간을 제공할 수 있다. 작은 크기로 인해 증가한 L1 데이터 캐쉬의 미스율을 경감시키기 위해서 ECP(Early Cache hit Predictor)기법을 제안하였다. 또한 바로쓰기(write-through)정책의 채용으로 인한 증가된 L2 데이터 캐쉬의 에너지 소비를 줄이기 위해 지정웨이 쓰기 정책을 제안하였다. 사이클 단위 정확도의 시뮬레이터와 내장형 벤치마크를 이용한 실험 결과 제안된 2-레벨 데이터 캐쉬 메모리 구조는 평균적으로 3.6%의 성능향상과 50%의 데이터 캐쉬 에너지와 유효주소 계산으로 인한 ALU의 에너지 소비를 감소시켰다. 제안된 2-레벨 데이터 캐쉬 구조는 성능향상과 에너지 소비의 감소라는 상호 교환적인 두 가지 목표를 동시에 충족시키는데 효과적임을 실험을 통해 입증하였다.

참 고 문 헌

[1] S. Segars, "Low Power Design Techniques for Microprocessors," *ISSCC*, Feb. 2001.

[2] J. L. Hennessy and D. A. Patterson, Computer Architecture A Quantitative Approach 3rd ed., Morgan Kaufmann Publisher, 2003.

[3] N. P. Jouppi, "Cache Write Policies and Performance," In *Proc. International symposium on Computer Architecture*, pp.191-201, 1993.

[4] J. Kin., M. Gupta., W. H. Mangione-Smith, "The Filter Cache: An Energy Efficient Memory Structure," In *Proc. International Symposium on Microarchitecture*, pp.184-193, 1997.

[5] W. Tang, R. Gupta, A. Nicolau, "Design a Predictive Filter Cache for Energy Savings in High Performance Processor Architecture," In *Proc. International Conference on Computer Design*, pp.68-75, 2001.

[6] N. Bellas, I. Hajj, and C. Polychronopouos, "Using dynamic cache management techniques to reduce energy in a high-performance processor," In *Proc. International Symposium on Low Power Electronics and Design*, pp.64-69, 1999.

[7] L. H. Lee, B. Meyer, and J. Arends, "Instruction Fetch Energy Reduction Using Loop Caches For Embedded Applications with Small Tight Loops," In *Proc. International Symposium on Low Power Electronics and Design*, pp.267-269, 1999.

[8] A. Gordon-Ross, S. Cotterell, and F. Vahid, "Tiny instruction caches for low power embedded systems," *ACM Transactions on Embedded Computing Systems*, vol.2, Issue 4, pp.449-481, 2003.

[9] M. D. Powell, A. Agarwal, T. N. Vijaykumar, B. Falsafi, and K. Roy, "Reducing Set-Associative Cache Energy via Way-Prediction and Selective Direct-Mapping," In *Proc. International symposium on Microarchitecture*, pp.54-65, 2001.

[10] T. Ishihara, F. Fallah, "A Non-Uniform Cache Architecture for Low Power System Design," In *Proc. International Symposium on Low power Electronics and Design*, pp.363-368, 2005.

[11] T. M. Jones, S. Bartolini, B. D. Bus, J. Cavazos and Michael F.P. O'Boyle, "Instruction Cache Energy Saving Through Compiler Way-Placement," In *Proc. Design, Automation and Test in Europe*, pp.1196-1201, 2008.

[12] A. Ma, M. Zhang and K. Asanovi, "Way memoization to reduce fetch energy in instruction caches," ISCA Workshop on Complexity Effective Design, 2001.

[13] R. Min, W. B. Jone and Y. Hu, "Location Cache: A Low-Power L2 Cache System," In *Proc. International Symposium on Low Power Electronics*

and Design, pp.120-125, 2004.

[14] J. K. Peir, S. C. Lai, S. L. Lu, J. Stark and K. Lai, "Bloom Filtering Cache Misses Accurate Data Speculation and Prefetching," In *Proc. International Conference on Supercomuting*, pp.189-198, 2002.

[15] T. M. Austin, D. N. Pnevmatikatos and G. S. Sohi, "Streamlining Data Cache Access with Fast Address Calculation," In *Proc. International Symposium on Computer Architecture*, pp.369-380, 1995.

[16] T. M. Austin and G. S. Sohi, "Zero-Cycle Loads: Microarchitecture Support for Reducing Load Latency," In *Proc. International Symposium on Microarchitecture*, pp.82-92, 1995.

[17] Advanced Digital Chips at http://www.adchips.co.kr.

[18] R. E. Kessler, "The Alpha 21264 microprocessor," In *Proc. IEEE MICRO*, pp.24-36, April 1996.

[19] G. Hinton, D.Sager, M, Upton, D. Boggs, D. Carmean, A. Kyker and P. Roussel, "The Microarchitecture of the Pentium 4 processor," *Intel Technology Journal*, 2001.

[20] A. Chandrakasan, W. J. Bowhill and F. Fox, "Design of High-Performance Microprocessor Circuits," IEEE Press, 2001.

[21] P. Shivakumar and N. P. jouppi, "CACTI 3.0: An Integrated Cache Timing, Power, and Area Model," WRL Research Report (Feb. 2001).

[22] G. Contreras, M. Martonosi, J. Peng, R. Ju and G. Y. Lueh, "XTREM: A Power Simulator for the XScale Core," In *Proc. ACM SIGPLAN/SIGBED Conference on Compilers, Architectures, and Synthesis*, pp.115-125, 2004.

[23] D. Brooks, V. Tiwari, M. Martonosi, "Wattch: a framework for architectural for architectural-level power analysis and optimizations," In *Proc. International Symposium on High-Performance Computer Architecture*, pp.83-94, 2000.

[24] Oklahoma State University System on Chip (SoC) Design Flows, http://avatar.ecen.okstate.edu/projects/scells/

[25] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," In *Proc. 4th Annual Workshop on Workload Characterization, 2001*. http://www.eecs.umich.edu/mibench/.

[26] C. J. Fang, C. H. Huang, J. S. Wang and C. W. Yeh, "Fast and Compact Dynamic Ripple Carry Adder Design," *ASIC*, 2002.

이 종 민

2008년 단국대학교 컴퓨터공학 학사. 2008년~현재 KAIST 전산학과 석박사통합과정. 관심분야는 내장형 시스템, 컴퓨터 구조, 저전력 컴퓨팅

김 순 태

1996년 중앙대학교 전자계산학과 학사 1998년 중앙대학교 컴퓨터공학과 석사 2003년 Penn State Univ. 컴퓨터공학 박사. 2004년~2007년 Univ. of South Florida Dept. of CSE 조교수. 2007년~2009년 한국정보통신대학교 공학부 조교수. 2009년~현재 KAIST 전산학과 조교수. 관심분야는 내장형 시스템, 컴퓨터 구조, 저전력 컴퓨팅, 고신뢰성 컴퓨팅, 사이버-피지컬 시스템