분할 기법을 이용한 저전력 명령어 캐쉬 설계
(Energy-aware Instruction Cache Design using Partitioning)

김 종 면 † 정 재 욱 †† 김 철 홍 †††
(Jong Myon Kim) (Jae Wook Jung) (Cheol Hong Kim)

요 약 최근의 내장형 프로세서를 설계하는데 있어서는 성능 못지 않게 에너지 효율성이 중요하게 고려되어야 한다. 내장형 프로세서에서 소모되는 에너지의 상당 부분은 캐쉬 메모리에서 소모되는 것으로 알려지고 있다. 특히 1차 명령어 캐쉬는 거의 매 사이클마다 접근이 이루어지므로 상당히 많은 양의 동적 에너지를 소모하게 된다. 그러므로, 내장형 프로세서를 설계하는데 있어서 1차 명령어 캐쉬의 에너지 효율성을 높이는 기법은 프로세서의 총 에너지 소모를 줄여주는 결과로 이어질 것으로 기대된다. 본 논문에서는 내장형 프로세서에 적합한 저전력 1차 명령어 캐쉬를 설계하는 기법을 제안하고자 한다. 제안하는 기법은 명령어 캐쉬를 여러 개의 작은 서브 캐쉬들로 분할하는 기법을 통해 명령어 접근 시 활성화되는 캐쉬의 크기를 줄임으로써 1차 명령어 캐쉬에서 소모되는 동적 에너지를 감소시켜 준다. 또한, 하나의 서브 캐쉬 크기를 페이지 크기와 동일하게 함으로써 캐쉬 내에서 태그가 차지하는 칩 공간을 없애고, 태그 비교에 소모되는 에너지도 없애는 효과를 얻는다. 제안하는 1차 명령어 캐쉬는 물리적인 접근 시간 감소를 통해 캐쉬 분할로 인한 성능 저하를 최대한 줄이고, 에너지 감소 효과는 최대로 얻고자 한다. 모의 실험 결과, 제안하는 구조는 기존의 1차 명령어 캐쉬 구조와 비교하여 명령어 접근에 소모되는 동적 에너지를 평균 37% ~ 60% 감소시키는 결과를 보인다.

키워드 : 명령어 캐쉬, 분할 캐쉬, 동적 에너지, 내장형 프로세서, 저전력 프로세서

Abstract Energy consumption in the instruction cacheaccounts for a significant portion of the total processor energy consumption. Therefore, reducing energy consumption in the instruction cache is important in designing embedded processors. This paper proposes a method for reducing dynamic energy consumption in the instruction cache by partitioning it to smaller (less energy-consuming) sub-caches. When a request comes into the proposed cache, only one sub-cache is accessed by utilizing the locality of applications. By contrast, the other sub-caches are not accessed, leading todynamic energy reduction. In addition, the proposed cache reduces dynamic energy consumption by eliminating the energy consumed in tag matching. We evaluated the energy efficiency by running cycle accurate simulator, SimpleScalar, with power parameters obtained from CACTI. Simulation results show that the proposed cache reduces dynamic energy consumption by 37%~60% compared to the traditional direct-mapped instruction cache.

Key words : Instruction cache, Partitioned cache, Dynamic energy, Embedded processor, Low power design

1. 서 론

내장형 프로세서(Embedded Processor)를 설계하는데 있어서는 칩 공간(Chip Area)을 줄이는 기법과 성능을 향상시키는 기법이 설계의 주된 초점이 되어 왔다. 하지만, 내장형 프로세서의 성능이 점차 강력해짐에 따라 프로세서 내에서 소모되는 에너지가 크게 증가하는 문제가 생기게 되었다. 내장형 프로세서의 경우, 소모되는 에너지가 많아지게 되면 배터리 사용 시간이 감소하고, 쿨링 비용(Cooling Costs)이 커지는 심각한 문제가 발생한다. 그러므로, 최근의 내장형 프로세서를 설계하는데 있어서는 성능 못지 않게 에너지 효율성이 중요하게 고려되고 있다.

· 이 논문은 BK21사업의 연구비 지원에 의하여 연구되었고, 툴은 IDEC 사업에서 지원받았음
† 정 회 원 : 울산대학교 컴퓨터정보통신공학부 교수
jongmyon.kim@gmail.com
†† 학생회원 : 전남대학교 전자컴퓨터공학부
jjw3240@hotmail.com
††† 정 회 원 : 전남대학교 전자컴퓨터공학부 교수
cheolhong@gmail.com
(Corresponding author임)
논문접수 : 2007년 9월 6일
심사완료 : 2007년 10월 12일

캐쉬(Cache) 메모리는 내장형 프로세서 내에서 소모되는 에너지의 상당 부분을 차지하는 것으로 알려진다[1]. 이에 따라, 캐쉬의 에너지 효율성을 높이는 기법을 통해 프로세서 내의 에너지 소모를 줄이기 위한 연구들이 지속적으로 이루어지고 있다. 필터 캐쉬(Filter Cache)라는 작은 크기의 캐쉬 모듈을 이용하여 메인 캐쉬(Main Cache)로의 접근 횟수를 감소시킴으로써 캐쉬 내의 에너지 소모를 줄이는 방법이 제안되었다[2]. 1차 명령어 캐쉬와 프로세서 코어 사이에 작은 크기의 캐쉬를 삽입한 후, 컴파일러와의 연동을 통해 캐쉬 내의 에너지 소모를 줄이기 위한 연구도 제안되었다[3]. 캐쉬에 대한 접근이 많지 않은 시간에는 연관 캐쉬(Set-associative Cache) 내의 일부 웨이들(way)을 동적으로 비활성화시키고, 캐쉬 접근이 많은 시간에만 모든 웨이들을 활성화시키는 기법을 통해서 캐쉬의 에너지 소모를 줄이는 기법도 제안되었다[4]. 캐쉬로의 접근 요청이 있을 때, 요청되는 데이타가 있을 것으로 예측(Prediction)되는 특정 웨이만을 먼저 접근하고, 이 예측이 틀린 경우에만 다른 웨이들을 접근함으로써 약간의 성능 저하를 통해 큰 에너지 감소 효과를 얻는 기법도 제안되었다[5].

계층적 메모리 구조에서 1차 명령어 캐쉬는 거의 모든 사이클마다 접근이 이루어지므로 캐쉬 메모리들 중에서도 상당히 많은 에너지를 소모하는 모듈이다[6]. 그러므로, 프로세서의 에너지 효율성은 1차 명령어 캐쉬의 에너지 효율성과 큰 상관 관계를 가지게 된다. 본 논문에서는 내장형 프로세서의 에너지 효율성을 높이기 위한 기법으로, 1차 명령어 캐쉬를 여러 개의 작은 크기의 서브 캐쉬(Sub-Cache)들로 분할하여 구성함으로써 1차 명령어 캐쉬에서 소모되는 에너지를 줄이는 기법을 제안하고자 한다. 이 기법에서는, 응용 프로그램의 지역성에 기반을 둔 예측을 통해 1차 명령어 캐쉬 접근 시 하나의 서브 캐쉬만 접근하고 다른 서브 캐쉬들은 활성화시키지 않음으로써 1 차 캐쉬 내에서 소모되는 동적 에너지를 감소시킨다. 또한, 서브 캐쉬 내의 태그 영역을 없앰으로써 칩 공간을 절약하고, 태그 비교에 소모되는 에너지도 없애 준다.

이하 본 논문의 구성은 다음과 같다. 2장에서는 본 논문의 작성 동기를 기술하고 관련 연구들을 살펴본다. 3장에서는 기존의 1차 명령어 캐쉬 구조를 설명하고, 4장에서는 제안하는 분할 명령어 캐쉬 구조를 기술한다. 5장에서는 제안하는 구조의 효율성을 알아보기 위한 모의 실험 방법과 실험 결과를 기술한다. 끝으로 6장에서 결론을 맺는다.

2. 연구 배경 및 관련 연구

2.1 연구 배경

응용 프로그램의 가장 중요한 특성 중 하나는 지역성(Locality)이다. 시간 지역성(Temporal Locality)은 최근에 사용된 데이타는 가까운 시간 내에 다시 사용될 가능성이 높다는 것을 의미하고, 공간 지역성(Spatial Locality)은 주소가 가까운 데이타들은 짧은 시간 내에 함께 사용될 가능성이 높다는 것을 의미한다. 1차 캐쉬로 요청되는 데이타들의 지역성을 자세히 살펴보기 위해 모의 실험을 수행하여 다음과 같은 결과를 얻었다. 그림 1과 그림 2는 각각 정수 응용 프로그램과 실수 응용 프로그램에 대해서 1차 캐쉬로의 데이타 요청이 바로 직전에 요청된 데이타가 포함된 페이지를 재접근하는 확률을 보여준다. 모의 실험은 SimpleScalar[7] 시뮬레이터를 이용하여 SPEC CPU2000 벤치마크 프로그램들에 대해 수행하였다[8]. 그래프에서 보는 바와 같이 1차 명령어 캐쉬에 대한 접근은 거의 대부분 직전에 접근하였던 페이지를 재접근한다는 사실을 알 수 있다. 이러한 지역성 특성은, 정수 응용 프로그램보다 실수 응용 프로그램에서 보다 높게 나타난다(그림 2).

이와 같은 프로그램의 높은 지역성을 활용하여 1차 명령어 캐쉬에서 소모되는 에너지를 감소시키기 위해 본 논문에서는 분할을 통한 새로운 구조의 1차 명령어 캐쉬를 제안하고자 한다. 위의 그래프에서 보는 바와 같이 1차 데이타 캐쉬의 경우는 지역성이 명령어 캐쉬에

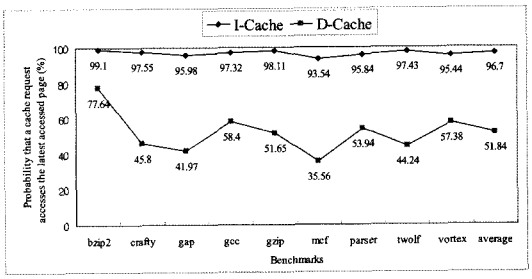


그림 1 1차 캐쉬 접근이 직전에 접근된 페이지를 재접근하는 확률 (정수 응용 프로그램)

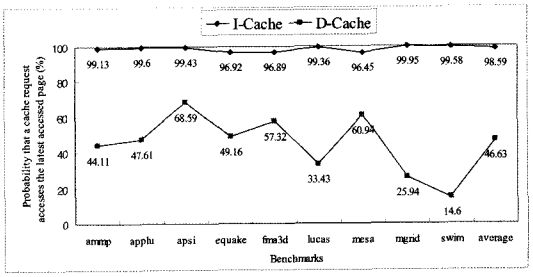


그림 2 1차 캐쉬 접근이 직전에 접근된 페이지를 재접근하는 확률 (실수 응용 프로그램)

비해 현저하게 떨어지므로 데이타 캐쉬에는 제안하는 구조가 적당하지 않을 것으로 판단된다.

2.2 관련 연구

캐쉬 분할에 관련된 연구들은 최근에 주로 이루어졌다. 본 논문에서 제안하는 구조와 유사한 구조로, 1차 명령어 캐쉬를 여러 개의 서브 캐쉬들로 분할함으로써 캐쉬 접근으로 인해 소모되는 에너지를 감소시키기 위한 기법이 제안되었다[9]. 이 연구에서는 TLB(Translation Lookaside Buffer)의 각 엔트리(Entry)에 서브 캐쉬 아이디를 새롭게 추가하여 특정 페이지 내의 데이타는 해당 TLB 엔트리의 서브 캐쉬 아이디가 가리키는 서브 캐쉬에만 할당하는 정책을 제안하였다. 본 논문에서는 하나의 서브 캐쉬에 여러 개의 페이지들을 동시에 할당하는 이 기법과 달리, 하나의 서브 캐쉬에 하나의 페이지만 할당되도록 하는 기법을 제안한다. 이를 통해, 서브 캐쉬 내에 태그 영역을 없앰으로써 캐쉬 접근 시 소모되는 에너지를 더욱 감소시키고, 하드웨어 구성을 훨씬 간단하게 함으로써 1차 명령어 캐쉬가 차지하는 칩 공간을 줄여주어 내장형 프로세서에 더욱 적합한 구조를 제안한다.

캐쉬를 분할함으로써 태그 영역을 감소시키는 연구도 이루어졌다[10]. 이 연구에서는 캐쉬를 여러 개로 분할하고, 분할된 캐쉬에는 적은 개수의 페이지들을 할당함으로써 캐쉬 내의 태그 영역을 줄이는 효과를 얻는다. 하지만, 이 연구에서는 캐쉬 접근 시 모든 서브 캐쉬들이 동시에 활성화되기 때문에 에너지 소모 감소에 대한 고려는 전혀 이루어지지 않았다고 볼 수 있다. 이와 달리, 본 논문에서는 내장형 프로세서에 적합한 저전력 1차 명령어 캐쉬 구조를 제안한다.

3. 기존의 명령어 캐쉬 구조

그림 3은 기존의 1차 명령어 캐쉬 구조를 보여주고 있다. 본 논문에서 목표로 하는 명령어 캐쉬는 Virtually-Indexed, Physically-Tagged(VI-PT) 캐쉬이다. 최근

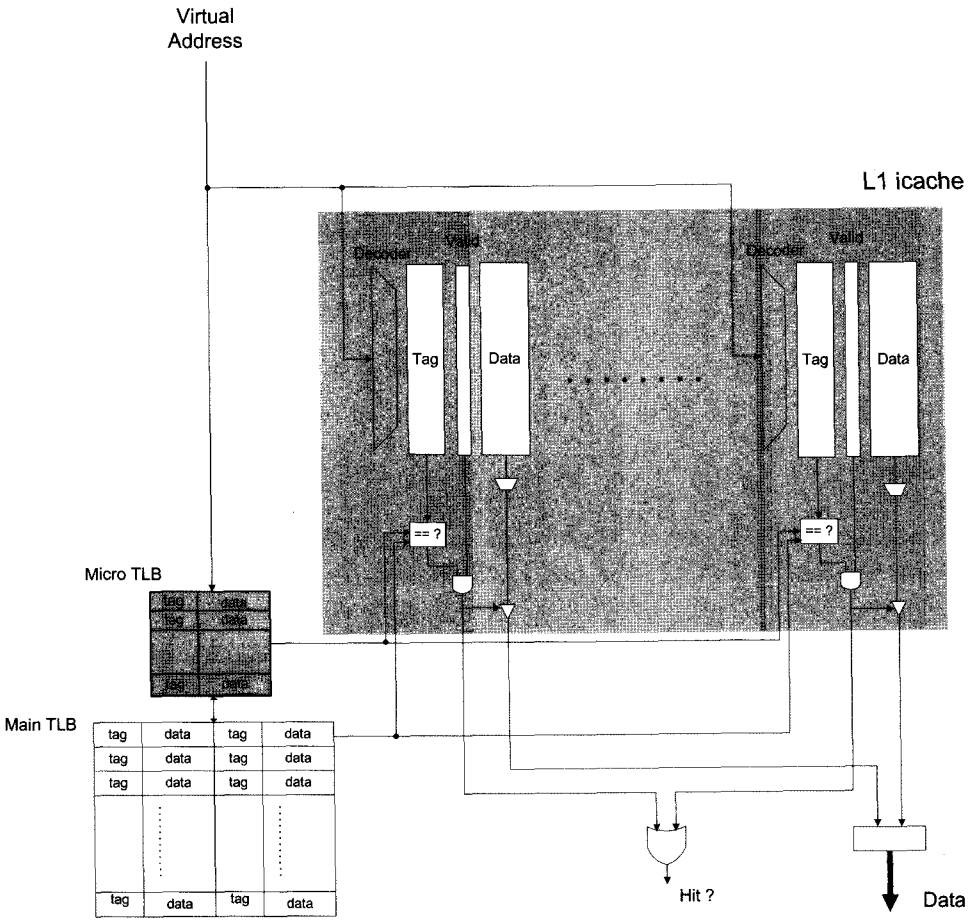


그림 3 기존의 1차 명령어 캐쉬 구조

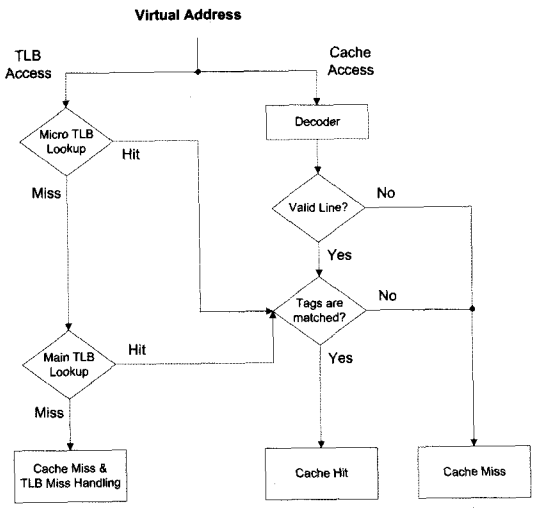


그림 4 기존 캐쉬에서의 명령어 접근 흐름도

설계되는 대부분의 프로세서에 적용되고 있는 VI-PT 캐쉬는 TLB 접근과 캐쉬 접근을 동시에 함으로써 데이타 접근 시간을 줄이는 장점을 가진다. 즉, 캐쉬 접근은 가상 주소(Virtual Address)를 이용하여 인덱싱하고, 캐쉬 접근과 동시에 TLB 접근을 한 후, TLB 접근을 통해 얻은 물리 주소(Physical Address)를 이용하여 가상 주소를 통해 가져온 데이타가 실제로 요구된 데이타인지를 검증하는 방법을 사용한다. 그림 3에서 보는 바와 같이 본 논문에서 목표로 하는 TLB 구조는 ARM11 [11]과 같은 최근의 내장형 프로세서에서 많이 사용되고 있는 2단계 TLB(2-Level TLB)이다. 2단계 TLB는 작은 크기의 마이크로 TLB(Micro TLB)를 메인 TLB (Main TLB) 위에 위치시킴으로써 메인 TLB로의 접근 횟수를 감소시켜 TLB내에서의 에너지 소모를 감소시킨다. 물론, 마이크로 TLB에서 미스(Miss)가 발생하는 경우에는 메인 TLB를 다시 접근해야 하므로 TLB의 평균 접근 시간이 길어지는 단점은 있다.

기존 캐쉬에서의 명령어 접근 과정은 그림 4에서 보이는 바와 같다. 프로세서로부터 명령어 요구가 캐쉬에 들어오면 가상 주소를 이용하여 캐쉬는 접근되고, 이와 동시에 물리 주소를 얻기 위해 TLB가 접근된다. 가상 주소를 통해 접근한 캐쉬 블록들이 모두 무효하다면 (Invalid) 캐쉬 접근은 미스가 된다. 가상 주소를 통해 접근한 블록들 중에 유효(Valid)한 블록들이 있으면, 해당 블록들의 태그를 TLB 접근을 통해 가져온 물리주소와 비교한다. 블록들 중에 태그가 일치하는 블록이 있다면 캐쉬 접근은 히트(Hit)이고, 일치하는 블록이 하나도 없으면 캐쉬 접근은 미스가 된다.

4. 제안하는 분할 명령어 캐쉬 구조

캐쉬에서 소모되는 에너지는 주로 캐쉬 크기(Size)나 캐쉬 연관성(Associativity)과 같은 캐쉬의 구성에 크게 좌우된다. 일반적으로, 한 번 접근 시 소모되는 동적 에너지는 캐쉬의 크기가 커질수록 증가하게 된다. 하지만, 캐쉬의 크기가 작게 되면 미스율(Miss Rates)의 증가로 인해 성능이 저하되므로 많은 에너지 소모에도 불구하고 크기가 큰 캐쉬를 사용하게 된다. 본 논문에서는 작은 크기의 캐쉬가 가지는 에너지 효율성의 장점과 큰 크기의 캐쉬가 가지는 성능의 장점을 모두 얻기 위해 여러 개의 작은 크기의 서브 캐쉬들로 구성되는 분할 명령어 캐쉬를 제안한다. 제안하는 분할 명령어 캐쉬로 프로세서의 요구가 들어오면, 분할 캐쉬는 예측을 통해 선택한 하나의 서브 캐쉬만 접근하고 다른 서브 캐쉬들은 활성화하지 않음으로써 한 번 접근 시 소모되는 에너지를 기존의 캐쉬와 비교하여 감소시키게 된다. 제안하는 분할 캐쉬 내의 각 서브 캐쉬들은 마이크로 TLB에 할당되어 있는 페이지들에 하나씩 대응된다. 즉, 분할 캐쉬 내의 서브 캐쉬 개수를 마이크로 TLB의 엔트리 수와 동일하게 함으로써 각각에 대해 1대 1 대응을 시킨다.

크기 못지 않게 캐쉬의 에너지 효율성에 큰 영향을 미치는 캐쉬의 구성 요소는 연관성이다. 연관성이 커질수록 캐쉬 접근에 사용되는 출력 드라이버(Output Driver)의 수, 비교기(Comparator)의 수, 감지 증폭기(Sense Amplifier)의 수가 증가하게 되므로 소모되는 에너지도 당연히 증가하게 된다. 예를 들어, 캐쉬 접근 시 직접 사상(Direct-Mapped) 캐쉬는 하나의 데이타만을 읽어오는데 반해, 4-way 연관 캐쉬는 4개의 웨이들에서 각각 데이타를 읽은 후 태그를 비교하여 하나만 선택하게 되므로 나머지 3개의 데이타를 읽는 과정에서 소모된 에너지의 낭비가 생기는 것이다. 에너지 효율성을 높이기 위해 제안하는 분할 캐쉬 내의 각 서브 캐쉬는 직접 사상 캐쉬로 구성한다. 또한, 앞에서 기술한 바와 같이 서브 캐쉬의 크기는 대응되는 페이지의 크기와 동일하므로 제안하는 분할 캐쉬에는 태그 영역이 필요 없게 된다. 페이지 내의 데이타는 특정 서브 캐쉬의 특정 엔트리에만 할당되기 때문에 태그 비교를 하지 않더라도 원하는 데이타를 찾을 수가 있게 된다.

그림 5는 제안하는 분할 명령어 캐쉬의 구조를 보여준다. 기존의 명령어 캐쉬와 비교하여 크게 3가지 부분이 변경되었다. 첫째, 마이크로 TLB의 각 엔트리에 아이디(id) 필드를 추가하였다. 아이디 필드는 해당 엔트리에 위치하는 페이지 내의 데이타가 할당되는 서브 캐쉬의 아이디를 나타낸다. 제안하는 구조에서는 마이크로

TLB의 엔트리 개수와 서브 캐쉬의 개수가 동일하기 때문에, 마이크로 TLB에 위치하는 페이지들은 페이지 내의 데이타를 할당하는 하나의 서브 캐쉬를 가지게 되므로 대응된 서브 캐쉬를 가리키기 위해 아이디 필드를 마이크로 TLB에 추가하였다. 둘째, 직전에 접근된 서브 캐쉬의 아이디를 저장하기 위해 PSC(Predicted Sub-Cache id) 레지스터가 추가되었다. 분할 캐쉬로의 접근을 PSC 레지스터에 저장된 아이디 값을 사용하여 이루어지게 함으로써, 직전에 접근된 서브 캐쉬만 접근이 이루어지도록 한다. 셋째, 제안하는 분할 캐쉬에는 기존의 캐쉬와 달리 태그 영역이 존재하지 않는다.

제안하는 분할 캐쉬에서의 명령어 접근 과정은 그림 6에서 보이는 바와 같다. 프로세서로부터 명령어 요구가 들어오면, PSC 레지스터에 담긴 내용을 참고하여 바로 직전에 접근되었던 서브 캐쉬만을 접근한다. 이와 동시에 TLB에 대한 접근도 이루어진다. 이때, 마이크로 TLB에 대한 접근이 히트인 경우에는, 요구된 데이타가 1차 캐쉬에 할당되어 있는 페이지들 중에 들어있다는 것을 의미한다. 마이크로 TLB에 위치하는 페이지들은 모두 하나씩의 서브 캐쉬에 대응되어 있기 때문이다. 이 경우에는, 올바른 서브 캐쉬가 접근되었는지 확인하기 위해 마이크로 TLB에서 히트가 발생한 엔트리의 아이디와 PSC 레지스터에 담겨있는 아이디를 서로 비교한다. 두 값이 일치하는 경우에는 올바른 서브 캐쉬로의 접근이 이루어졌다는 것을 의미한다. 이 경우에는, 서브 캐쉬에서 히트가 발생하면 명령어를 프로세서로 보내주고, 미스가 발생하는 경우에는 하위 레벨 메모리로부터 요구하는 데이타를 가져온다.

마이크로 TLB에서 히트가 발생한 엔트리의 아이디와 PSC 레지스터에 담긴 아이디가 일치하지 않는 경우에는 잘못된 서브 캐쉬를 접근했다는 것을 의미한다. 이 경우에는, 마이크로 TLB에서 히트가 발생한 엔트리의 아이디가 가리키는 새로운 서브 캐쉬를 다시 접근하여야 한다. 이와 같이 서브 캐쉬 예측이 틀린 경우에는 새로운 서브 캐쉬 접근으로 인해 캐쉬 접근 시간이 증가하게 된다.

마이크로 TLB에서 미스가 발생하는 경우에는, 요구된 데이타가 1차 캐쉬에 할당되어 있는 페이지들이 아닌 다른 페이지 내의 데이타임을 의미한다. 이 경우에는 마이크로 TLB에서 엔트리의 교체가 발생하게 된다(교

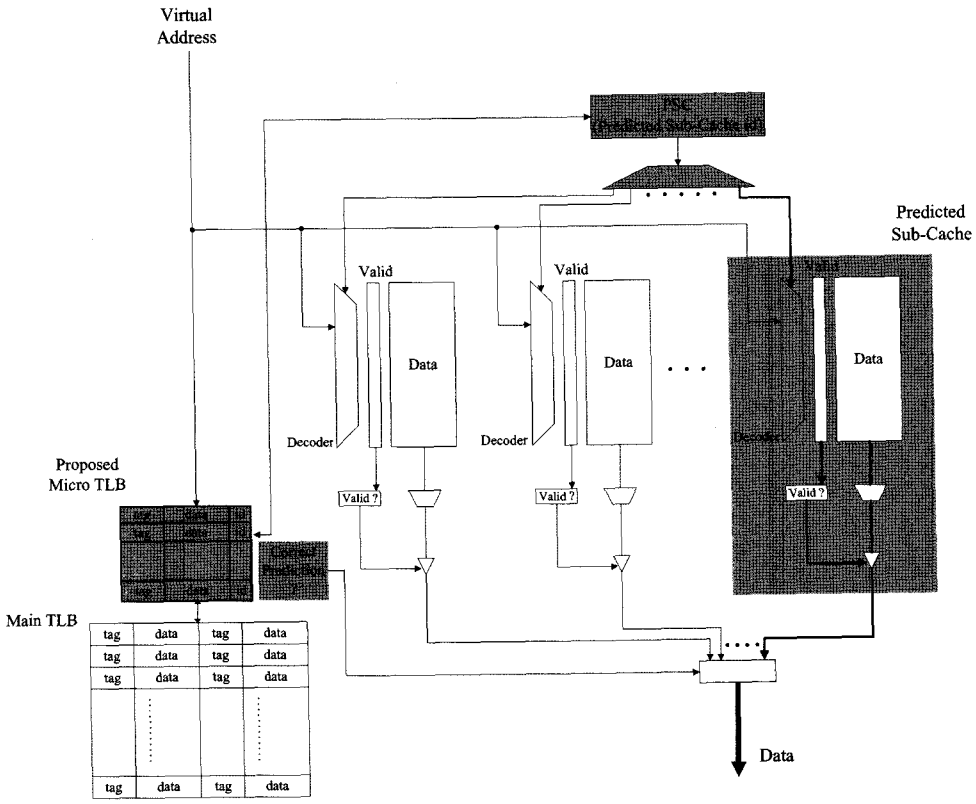


그림 5 제안하는 분할 명령어 캐쉬 구조

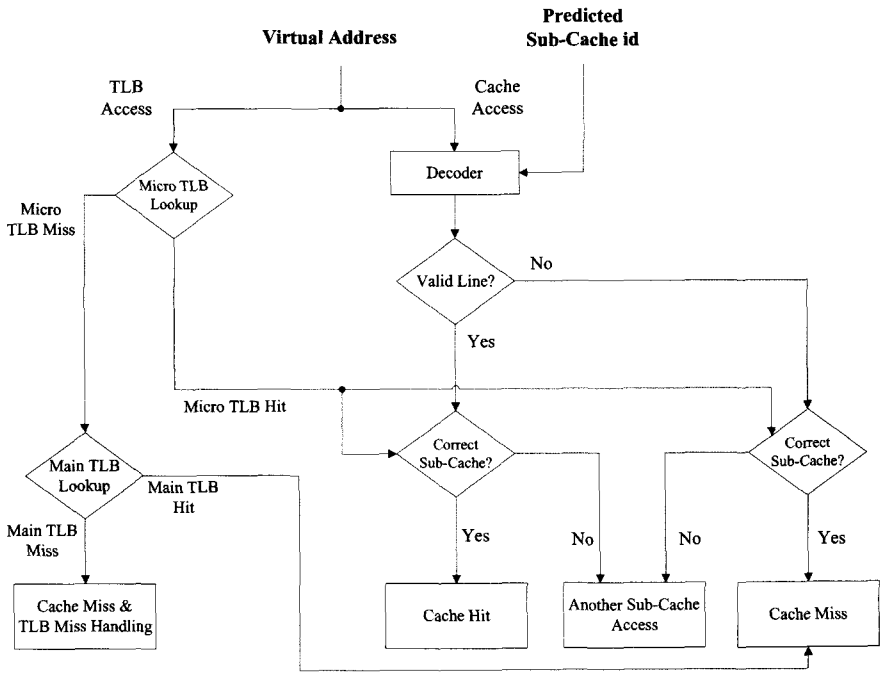


그림 6 제안하는 분할 캐쉬에서의 명령어 접근 흐름도

체는 LRU 기법을 통해 이루어진다). 이 경우, 교체되는 마이크로 TLB 엔트리에 대응하고 있는 서브 캐쉬에는 새로운 페이지의 데이타가 할당될 것이므로 해당 서브 캐쉬를 새롭게 초기화하고 요청된 데이타를 하위 레벨 메모리로부터 가져온다. 명령어 캐쉬는 읽기 접근만이 허용되기 때문에 Write-back을 해줄 필요가 없으므로 서브 캐쉬 모든 엔트리의 유효화 비트(Valid Bit)를 초기화하는 것만으로 쉽게 캐쉬를 초기화할 수 있다. 초기화를 한 후, 새롭게 마이크로 TLB 엔트리에 위치한 페이지 내의 데이타가 요청되면, 요구된 데이타는 초기화된 서브 캐쉬로 들어가게 된다.

제안하는 분할 캐쉬에서는 하나의 페이지가 같은 크기의 서브 캐쉬에 할당되므로 기존의 캐쉬와 달리 충돌 미스(Conflict Miss)가 발생하지 않는다. 하지만, 분할 캐쉬 내에서는 마이크로 TLB의 엔트리 교체가 발생할 때마다 서브 캐쉬가 초기화되기 때문에 그로 인해 많은 강제 미스(Compulsory Miss)가 발생할 것으로 예상된다. 만약, 마이크로 TLB에 페이지가 새롭게 할당될 때 페이지 내의 데이타를 모두 서브 캐쉬로 가져온다면 강제 미스를 없앨 수도 있을 것이다. 하지만, 이러한 동작은 시스템 버스에 심각한 정체 현상을 가져오게 될 것이고, 많은 데이타의 전송으로 인해 에너지 소모도 상당히 커질 것이다. 그러므로, 제안하는 분할 캐쉬는 해당 데이타가 프로세서로부터 요청될 경우에만 하위 레벨 메모리의 접근을 통해 데이타를 가져오도록 한다.

기존의 명령어 캐쉬와 비교하여 제안하는 분할 명령어 캐쉬는 하드웨어 오버헤드를 거의 가지지 않는다. 첫째, 마이크로 TLB의 각 엔트리에 아이디 필드를 추가하게 되지만, 마이크로 TLB의 엔트리 개수가 적고, 아이디 필드는 2~4 비트의 공간만 차지한다. PSC 레지스터로 사용되는 레지스터가 하나 추가되고, 서브 캐쉬 예측의 정확성을 확인하기 위해 비교기(Comparator)가 하나 필요하지만, 이로 인한 오버헤드는 무시할만 하다고 볼 수 있다. 또한, 기존의 명령어 캐쉬와 달리 태그 영역이 사라지게 되므로 공간(Area) 측면에서는 기존의 캐쉬보다 훨씬 좋다고 볼 수 있다.

제안하는 분할 캐쉬는 접근 시 활성화되는 캐쉬의 크기를 줄이고 태그 비교를 없앰으로써 1차 명령어 캐쉬가 한 번 접근될 때 소모되는 동적 에너지를 크게 줄일 수 있을 것으로 기대된다. 그러므로, 캐쉬의 미스율이 증가하여 성능이 크게 저하되지만 않는다면 저전력을 요구하는 내장형 프로세서의 1차 캐쉬로 사용될 수 있을 것으로 기대된다. 2장에서 살펴본 바와 같이 1차 명령어 캐쉬로의 대부분의 요청은 바로 직전에 접근한 페이지 내의 데이타일 확률이 상당히 높으므로, 제안하는 분할 캐쉬의 미스율은 기존의 캐쉬와 비교하여 크게 증가하지는 않을 것으로 생각된다.

5. 모의 실험 및 실험 결과

이 장에서는 제안하는 분할 캐쉬의 성능 및 에너지 효율성을 기존의 캐쉬와 비교하기 위한 모의 실험 방법과 실험 결과를 기술한다. 모의 실험은 SimpleScalar 시뮬레이터를 수정하여 수행하였다[7]. 실험에 관련된 에너지 소모 관련 변수들은 0.18 um 공정을 가정한 CACTI를 사용하여 얻은 값들을 사용하였다[12]. 시뮬레이터의 입력으로는 SPEC CPU2000 벤치마크 프로그램들을 사용하였다[8]. 모의 실험에 사용한 프로세서 모델은 표 1에서 보이는 바와 같이, ARM의 향후 개발 예정인 내장형 프로세서를 목표로 하여 2차 레벨 캐쉬를 가지는 2-way superscalar 프로세서를 사용하였다[13].

5.1 명령어 접근 시간 및 성능

표 2는 명령어 접근 시간을 계산하기 위한 분석 모델에 사용되는 기호와 그 의미를 표시한다. 표에 나타낸 기호를 사용하면, 기존의 캐쉬를 통한 명령어 접근에 소모되는 평균 시간(Instruction Fetching Delay)은 "$D(conv) = D_{L1\text{-}lookup} + D_{L1\text{-}miss}$"로 정의할 수 있고, 제안하는 분할 캐쉬를 통한 명령어 접근에 소모되는 평균 시간은 "$D(pic) = D_{L1\text{-}lookup} + D_{L1\text{-}miss} + D_{Overhead\text{-}in\text{-}pic}$"로 정의할 수 있다. 이러한 D(conv)과 D(pic)를 측정하여 비교 분석하기 위해 모의 실험을 수행하였다.

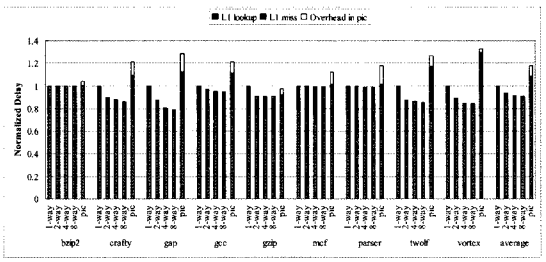
모의 실험을 통해 얻은 명령어 접근 평균 시간은 그림 7에서 보이는 바와 같다. 그래프에서 "pic"는 제안하는 분할 명령어 캐쉬를 나타낸다. "L1 lookup"은 1차 캐쉬를 접근하는데 소모된 시간을 나타내고, "L1 miss"는 1차 캐쉬 미스로 인해 하위 레벨 메모리를 접근하는데 소모된 시간을 나타낸다. "Overhead in pic"는 제안하는 분할 명령어 캐쉬에서 서브 캐쉬 예측 실패로 인해 잘못된 서브 캐쉬를 접근하는데 소모된 시간을 나타낸다. 그래프에서 보이는 바와 같이, 기존 캐쉬를 통한 명령어 접근 시간에는 "Overhead in pic"가 나타나지 않는다. 그래프의 세로축은 기존의 직접 사상 1차 명령어 캐쉬의 명령어 접근 평균 시간에 정규화된 각 캐쉬의 명령어 접근 평균 시간을 표시한다. 그래프를 보면, 기존 캐쉬의 경우 연관성이 커질수록 충돌 미스가 줄어들게 되므로 1차 캐쉬의 미스율이 감소하여 명령어 접근 시간이 단축되고 있다는 것을 알 수 있다. 16KB 크기의 분할 캐쉬는 기존의 직접 사상 캐쉬와 비교하여 정수 응용 프로그램에 대해서는 평균 18%, 실수 응용 프로그램에 대해서는 평균 13%의 명령어 접근 시간을 증가시킨다. 32KB 크기의 분할 캐쉬는 정수 응용 프로그램에서는 평균 10%, 실수 응용 프로그램에서는 평균 5%의 명령어 접근 시간 증가를 보인다. 명령어 접근 시

표 1 모의 실험 인자

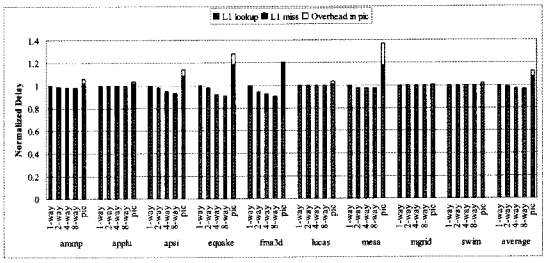
실험 인자	값
Functional Units	2 integer ALUs, 2 FP ALUs, 1 integer multiplier/divider, 1 FP multiplier/divider
Fetch, Decode, Issue, Commit	2 instructions/cycle
Branch Predictor	Bimodal
Micro TLB	fully-associative, 1 cycle latency
Main TLB	32 entry, fully-associative, 1 cycle latency, 30 cycle miss penalty
L1 i-cache	16KB and 32KB, 1-way to 8-way, 32 byte lines, 1 cycle latency
Sub-cache in the partitioned cache	4KB (Page size), 1-way, 32 byte lines, 1 cycle latency
L1 d-cache	32KB, 4-way, 32 byte lines, 1 cycle latency, write-back
L2 cache	256KB unified, 4-way, 64 byte lines, 8 cycle latency, write-back
Memory	64 cycle latency

표 2 분석 모델의 기호 및 의미

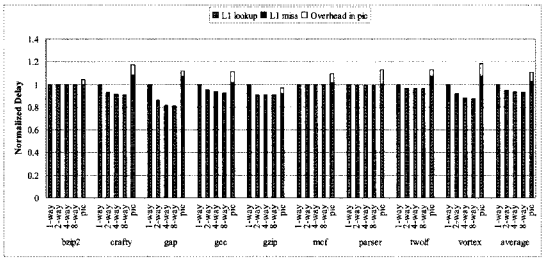
기호	의미
D(conv)	기존 캐쉬에서 명령어 인출 (Fetch)에 소모되는 평균 시간
D(pic)	제안하는 분할 캐쉬에서 명령어 인출에 소모되는 평균 시간
DL1-lookup	명령어 캐쉬 접근에 소요되는 사이클 (cycle) 수, 기존 캐쉬와 분할 캐쉬 모두 1 cycle로 가정함
DL1-miss	명령어 캐쉬 미스로 인해 소요되는 사이클 수, 아래 공식으로 계산됨 $D_{L1\text{-}miss}$ = iL1 miss rates iL1 miss penalty, iL1 miss penalty = L2 access latency (8 cycles) + L2 miss rates Memory access latency (64 cycles)
DOverhead-in-pic	분할 캐쉬에서 서브 캐쉬 예측 실패로 인해 소요되는 사이클 수, 아래 공식으로 계산됨 $D_{Overhead\text{-}in\text{-}pic}$ = Sub-cache misprediction rates iL1 access latency (1 cycle)



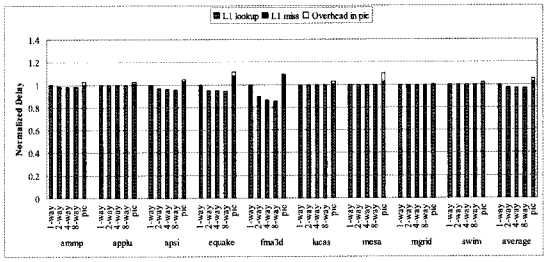
(a) 16KB L1 I-Cache (CINT)



(b) 16KB L1 I-Cache (CFP)



(c) 32KB L1 I-Cache (CINT)

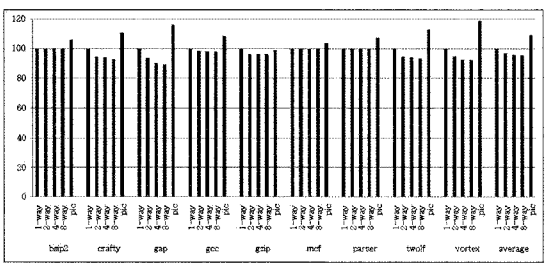


(d) 32KB L1 I-Cache (CFP)

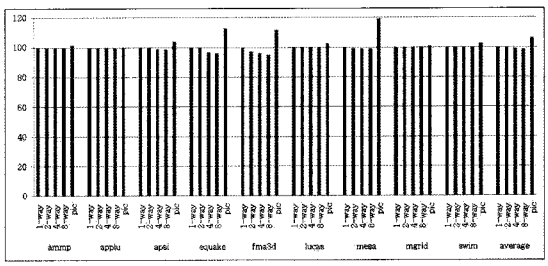
그림 7 명령어 접근 평균 시간 비교

간 증가의 이유로는 크게 두 가지를 들 수 있다. 첫째, 1차 캐쉬에 할당되는 데이타를 마이크로 TLB에 할당된 페이지 내의 데이타로 한정함으로써 캐쉬의 미스율이 증가하게 된다. 둘째, 그래프에서 "Overhead in pic"로 표시되고 있는 바와 같이 서브 캐쉬 예측 실패로 인해 또 다른 서브 캐쉬를 접근하는데 시간이 소모된다. 특이하게, 제안하는 분할 캐쉬는 gzip 응용 프로그램에 대해서는 충돌 미스를 줄임으로 인해 오히려 기존의 직접 사상 캐쉬보다 좋은 성능을 보이기도 한다는 것을 알 수 있다.

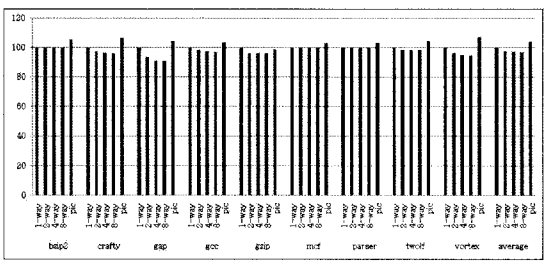
명령어 접근 시간 증가로 인한 성능 저하 결과는 그림 8에서 보이는 바와 같다. 그림 8은 직접 사상 1차



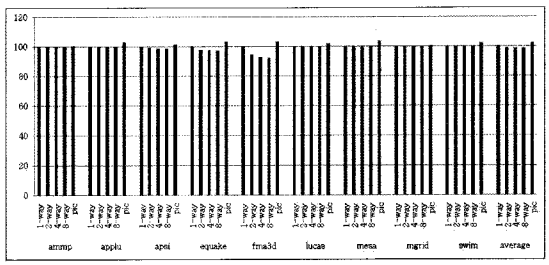
(a) 16KB L1 I-Cache (CINT)



(b) 16KB L1 I-Cache (CFP)



(c) 32KB L1 I-Cache (CINT)



(d) 32KB L1 I-Cache (CFP)

그림 8 프로그램 수행 시간 비교

명령어 캐쉬를 사용하는 시스템의 프로그램 수행 시간에 정규화된 각 시스템의 프로그램 수행 시간을 나타낸다. 그래프에서 보이는 바와 같이, 16KB 크기의 분할 캐쉬를 사용하는 시스템의 성능은 기존의 직접 사상 캐쉬를 사용하는 시스템과 비교하여 정수 응용 프로그램에 대해서는 평균 9%, 실수 응용 프로그램에 대해서는 평균 6% 저하됨을 알 수 있다. 32KB 크기의 분할 캐쉬를 사용하는 시스템은 정수 응용 프로그램에서는 평균 4%, 실수 응용 프로그램에서는 평균 2%의 성능 저하 결과를 보인다.

전체적으로 보았을 때, 제안하는 분할 캐쉬는 정수 응용 프로그램에 비해 실수 응용 프로그램에서 상대적으로 좋은 성능을 보이고 있다. 이는 2장에서 살펴본 바와 같이, 실수 응용 프로그램의 지역성이 정수 응용 프로그램에 비해 보다 높기 때문인 것으로 판단된다. 또한, 기존 캐쉬와 분할 캐쉬의 명령어 접근 시간 차이는 캐쉬의 크기가 증가함에 따라 감소하고 있다는 것을 그래프를 통해 알 수 있다. 캐쉬의 크기가 증가하게 되면 분할 캐쉬가 포함할 수 있는 페이지의 수(서브 캐쉬의 수)가 증가하게 되므로, 캐쉬의 미스율이 상대적으로 감소하기 때문인 것으로 생각된다. 그러므로, 제안하는 분할 캐쉬는 캐쉬 크기가 증가함에 따라 보다 효율적일 수 있을 것으로 예상된다.

그림 7과 그림 8에서 보이는 성능 비교 결과는 표 1에서 보이는 모의 실험 구성을 통해 얻은 결과이다. 모의 실험에서는, 캐쉬 구성에 관계없이 모든 1차 명령어 캐쉬의 접근 시간을 1 사이클로 가정하였다. 하지만, 실제 캐쉬의 물리적인 접근 시간은 캐쉬의 구성에 따라 다르게 나타난다. 표 3은 CACTI를 통해 얻은 캐쉬의 구성에 따른 물리적인 접근 시간을 보여준다. 표에서 보는 바와 같이, 캐쉬의 연관성이 커지게 되면 일반적으로 캐쉬의 물리적인 접근 시간은 증가하게 된다. 표에서 나타낸 분할 캐쉬의 접근 시간은 "1 AND gate delay + 서브 캐쉬 접근 시간"으로 계산하였다. "1 AND gate delay"는 PSC 레지스터가 가리키는 서브 캐쉬를 활성화하기 위해 필요한 시간으로 0.114 ns이고(삼성전자의 ASIC STD130 DATABOOK[14]), 서브 캐쉬 접근 시간은 0.873 ns (CACTI) 이다. 표 3에서 보이는 바와 같이, 제안하는 분할 캐쉬는 기존의 직접 사상 캐쉬에 비해서도 빠른 물리적 접근 시간을 제공한다. 분할 캐쉬에서는 접근되는 캐쉬의 크기가 작아지고, 캐쉬 내에서 태그 접근으로 인한 지연 시간이 사라지기 때문이다. 이와 같은 물리적인 접근 시간 감소는 캐쉬의 크기가 증가함에 따라 더 크게 나타난다. 기존 캐쉬의 경우에는 캐쉬 크기가 증가하면 접근 시간이 증가하는데 반해 제안하는 분할 캐쉬는 접근 시간이 캐쉬의 전체 크기가 아닌 서브 캐쉬의 크기에 좌우되기 때문이다. 그러므로, 프로세서의 속도가 점차 빨라지고 캐쉬 크기가 커지게 되면 제안하는 분할 캐쉬는 상대적으로 보다 좋은 성능을 보일 것으로 기대된다.

5.2 에너지 소모

표 4는 CACTI를 통해 얻은 각 캐쉬 구성에 따른 한 번 접근 시 소모되는 동적 에너지를 보이고 있다. 기존 캐쉬의 경우, 캐쉬의 연관성이 증가함에 따라 에너지 소모가 커진다는 것을 알 수 있다. 제안하는 분할 캐쉬는 한 번 접근 시 직접 사상 캐쉬에 비해서도 적은 양의 에너지를 소모한다는 것을 알 수 있다. 캐쉬 분할을 통해 활성화되는 캐쉬의 크기를 줄이고 태그 비교에 소모되는 에너지를 없앴기 때문이다. 또한, 기존 캐쉬의 경우에는 캐쉬의 크기가 증가하게 되면 자연스럽게 소모되는 에너지도 커지는 데 반해, 제안하는 분할 캐쉬는 전체 캐쉬의 크기에 관계없이 예측된 하나의 서브 캐쉬만을 접근하기 때문에 캐쉬의 크기가 커지더라도 한 번 접근 시 소모되는 에너지에는 차이가 생기지 않는다는 장점을 가진다.

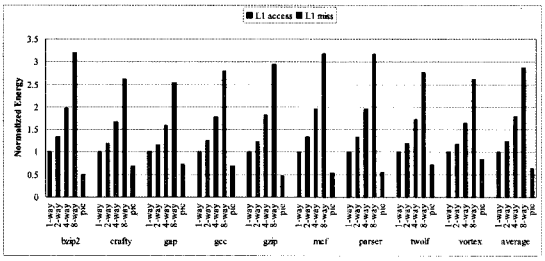
SimpleScalar를 통한 모의 실험 결과에 CACTI의 에너지 변수를 대입하여 얻은 에너지 소모 결과 비교는 그림 9와 같다. 이 그래프에서 보이는 총 에너지 소모는 명령어 접근에 소모되는 동적 에너지의 총합을 나타낸다. 각 그래프의 세로축은 기존의 직접 사상 1차 명령어 캐쉬의 에너지 소모에 정규화한 각 구조의 에너지 소모 결과를 나타낸다. 그래프에서, "L1 access"는 1차 캐쉬 접근에 소모된 동적 에너지를 나타내고, "L1 miss"는 1차 캐쉬 미스로 인해 하위 레벨 메모리를 접근하는데 소모된 에너지를 나타낸다. 그래프에서 보이는 바와 같이, 16KB 크기의 분할 캐쉬는 기존의 직접 사상 캐쉬와 비교하여 정수 응용 프로그램에서는 평균 37%, 실수 응용 프로그램에서는 평균 42%의 동적 에너지 소모를 감소시킨다. 32KB 크기의 분할 캐쉬는 정수 응용 프로그램에서는 평균 57%, 실수 응용 프로그램에서는 평균 60%의 동적 에너지 소모를 감소시킨다. 예상한대로 분할 캐쉬는 캐쉬의 크기가 커질수록 상대적으로 좋은 에

표 3 캐쉬 구성에 따른 물리적 접근 시간 비교

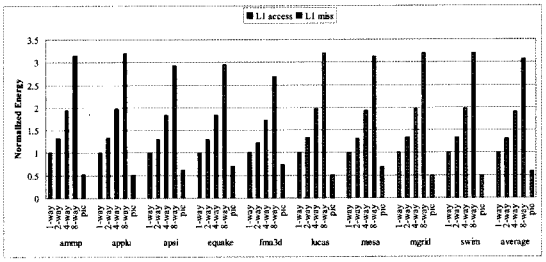
Access Time (ns)	1-way	2-way	4-way	8-way	pic
16KB	1.129	1.312	1.316	1.383	0.987
32KB	1.312	1.455	1.437	1.457	0.987

표 4 캐쉬 구성에 따른 한 번 접근 시 소모 에너지 비교

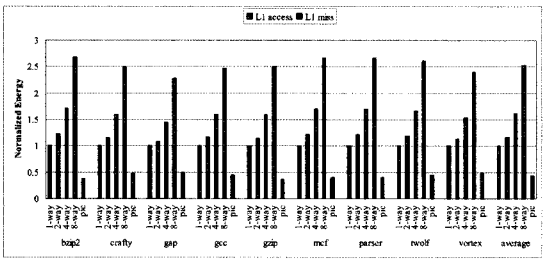
Energy (nJ)	1-way	2-way	4-way	8-way	pic
16KB	0.473	0.634	0.935	1.516	0.232
32KB	0.621	0.759	1.059	1.666	0.232



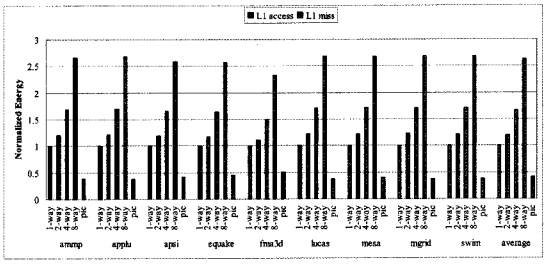
(a) 16KB L1 I-Cache (CINT)



(b) 16KB L1 I-Cache (CFP)



(c) 32KB L1 I-Cache (CINT)



(d) 32KB L1 I-Cache (CFP)

그림 9 명령어 접근에 소모되는 동적 에너지 비교

너지 효율성을 보인다. 또한, 지역성이 보다 좋은 실수 응용 프로그램에서 정수 응용 프로그램에 비해 보다 좋은 에너지 효율성을 보이고 있다.

6. 결 론

본 논문에서는, 내장형 프로세서에 적합한 저전력 1차 명령어 캐쉬 구조를 제안하였다. 1차 명령어 캐쉬를 작은 크기의 여러 개의 서브 캐쉬들로 분할하고 분할된 하나의 서브 캐쉬에는 마이크로 TLB에 할당된 하나의 페이지를 대응시켰다. 이를 통해, 프로세서로부터 명령어 접근 요구가 왔을 때 예측된 하나의 서브 캐쉬만을 접근함으로써 활성화되는 캐쉬의 크기를 줄이고, 캐쉬 내에서의 태그 비교를 없앰으로써 기존의 캐쉬 구조와 비교하여 훨씬 높은 에너지 효율성을 제공한다. 분할 캐쉬 내의 서브 캐쉬 예측은 프로그램의 지역성을 활용하여 바로 직전에 접근된 서브 캐쉬를 접근하도록 하였다. 제안하는 분할 캐쉬는 물리적인 접근 시간을 고려하면 기존의 캐쉬와 비교하여 성능 감소도 그리 크기 않을 것으로 기대된다. 그러므로, 제안하는 분할 캐쉬는 향후 내장형 프로세서에 사용될 1차 명령어 캐쉬의 구조로 적합할 것으로 기대된다.

참 고 문 헌

[1] Simon Segars, "Low power design techniques for microprocessors," In Proceedings of International Solid-State Circuits Conference, 2001.

[2] J. Kin, M. Gupta, and W. Mangione-Smith, "The Filter Cache: An Energy Efficient Memory Structure," In Proceedings of International Symposium on Microarchitecture, pp. 184-193, 1997.

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

[4] David H. Albonesi, "Selective Cache Ways: On-Demand cache resource allocation," In Proceedings of International Symposium on Microarchitecture, pp. 70-75, 1999.

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

[6] J. Montanaro et al., "A 160 Mhz, 32b, 0.5W CMOS RISC microprocessor," In Proceedings of International Solid-State Circuits Conference, pp. 214-229, 1996.

[7] D. Burger, T. M. Austin, and S. Bennett, "Evaluating future micro-processors: the SimpleScalar tool set," Tech. Report TR-1308, Univ. of Wisconsin-Madison Computer Sciences Dept., 1997.

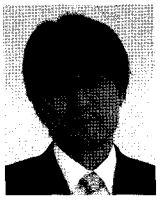
[8] SPEC CPU2000 Benchmarks, http://www.specbench.org.

[9] SoonTae Kim, N. Vijaykrishnan, Mahmut Kandemir, Anand Sivasubramaniam and Mary Jane Irwin, "Partitioned Instruction Cache Architecture for

Energy Efficiency," ACM Transactions on Embedded Computing Systems, Vol. 2, Issue 2, pp. 163-185, 2003.
[10] Yen-Jen Chang, Feipei Lai, and Shanq-Jang Ruan, "Cache Design for Eliminating the Address Translation Bottleneck and Reducing the Tag Area Cost," In Proceedings of International Conference on Computer Design, pp. 334, 2002.
[11] ARM Corp., ARM1136J(F)-S, http://www.arm.com/products/CPUs/ARM1136JF-S.html.
[12] P. Shivakumar and N. P. Jouppi, "CACTI 3.0: An Integrated Cache Timing, Power, and Area Model," TR-WRL-2001-2, 2001.
[13] Mike Muller, "At the Heart of Innovation," http://www.arm.com/miscPDFs/6871.pdf.
[14] ASIC STD130 DATABOOK, Samsung Electronics, 2001.

김 종 면
1995년 명지대학교 전기공학과 학사. 2000년 University of Florida Electrical & Computer Engineering 석사. 2005년 Georgia Institute of Technology Electrical & Computer Engineering 박사 2005년~2007년 삼성종합기술원 전임연구원. 2007년~현재 울산대학교 컴퓨터정보통신공학부 교수 관심분야는 임베디드 SoC 설계, 컴퓨터구조, Application-Specific 프로세서 설계, 병렬처리

정 재 욱
2006년 호남대학교 정보통신공학과 학사 2006년~현재 전남대학교 컴퓨터정보통신공학과 석사과정. 관심분야는 임베디드 시스템, SoC 설계, 컴퓨터구조



김 철 홍
1998년 서울대학교 컴퓨터공학과 학사 2000년 서울대학교 대학원 컴퓨터공학부 석사. 2006년 서울대학교 대학원 전기컴퓨터공학부 박사. 2005년~2007년 삼성전자 반도체총괄 SYS.LSI사업부 책임연구원. 2007년~현재 전남대학교 전자컴퓨터공학부 교수. 관심분야는 임베디드시스템, 컴퓨터구조, SoC 설계, 저전력 설계