캐시 파티션을 이용한 공유 2차 캐시 누설 에너지 관리 기법

(Leakage Energy Management Techniques via Shared L2 Cache Partitioning)

강 희 준 † 김 현 희 †† 김 지 홍 †††
(Hee-Joon Kang) (Hyunhee Kim) (Jihong Kim)

요 약 기존의 타임아웃 기반 캐시 누설 에너지 관리 기법들은 한동안 사용되지 않은 비활성화 상태의 캐시 라인의 전력 공급을 끊음으로써 누설 에너지 소모를 줄인다. 그러나, 이들 기법들은 단일 프로세서 환경에 적합하게 고안되었기 때문에, 태스크들 간의 간섭이 빈번히 발생하는 공유 2차 캐시를 사용하는 멀티프로세서 환경에서는 에너지 감소를 방해한다. 본 논문에서는 캐시 라인 비활성화 시간을 고려한 캐시 파티션 전략을 통해 캐시 간섭을 줄임으로써 멀티프로세서 환경의 공유 2차 캐시에서의 누설 에너지 감소 효과를 증가시키기 위한 기법을 제안한다. 또한, 각 태스크들의 특성을 고려하여 타임아웃을 설정하는 적응형 타임아웃 관리 기법을 통해 캐시 누설 에너지 소비를 감소시키는 기법을 제안한다. 시뮬레이션을 통한 실험 결과에서 기존의 기법과 비교하여 2-way CMP에서는 평균 73%, 4-way CMP에서는 평균 56% 정도의 누설 에너지 소비가 줄어드는 것을 확인하였다.

키워드 : 캐시 누설 에너지 관리, 멀티프로세서, 공유 2차 캐시, 캐시 파티션

Abstract The existing timeout based cache leakage management techniques reduce the leakage energy consumption of the cache significantly by switching off the power supply to the inactive cache line. Since these techniques were mainly proposed for single-processor systems, their efficiency is reduced significantly in multiprocessor systems with a shared L2 cache because of the cache interferences among simultaneously executing tasks. In this paper, we propose a novel cache partition strategy which partitions the shared L2 cache considering the inactive cycles of the cache line. Furthermore, we propose the adaptive task-aware timeout management technique which considers the characteristics of each task and adapts the timeout dynamically. Experimental results from the simulation show that the proposed technique reduces the leakage energy consumption of the shared L2 cache by 73% for the 2-way CMP and 56% for the 4-way CMP on average compared to the existing representative leakage management technique, respectively.

Key words : cache leakage energy management, multiprocessors, shared L2 cache, cache partitioning

· 본 논문은 BK21사업에 의하여 지원 되었으며, 정부(교육과학기술부)의 재원으로 한국과학재단의 지원을 받아(No. R0A-2007- 000-20116-0, R33-2008-000-10095-0) 수행되었습니다. 본 연구를 위해 연구 장비를 지원하고 공간을 제공한 서울대학교 컴퓨터 연구소에 감사드립니다.

† 정 회 원 : 서울대학교 컴퓨터공학과
kanghj@davinci.snu.ac.kr
†† 학생회원 : 서울대학교 컴퓨터공학과
hh0726@davinci.snu.ac.kr
††† 종신회원 : 서울대학교 컴퓨터공학과 교수
jihong@davinci.snu.ac.kr

논문접수 : 2009년 11월 12일
심사완료 : 2010년 1월 11일

Copyright©2010 한국정보과학회 : 개인 목적이나 교육 목적인 경우, 이 저작물의 전체 또는 일부에 대한 복사본 혹은 디지털 사본의 제작을 허가합니다. 이 때, 사본은 상업적 수단으로 사용할 수 없으며 첫 페이지에 본 문구와 출처를 반드시 명시해야 합니다. 이 외의 목적으로 복제, 배포, 출판, 전송 등 모든 유형의 사용행위를 하는 경우에 대하여는 사전에 허가를 얻고 비용을 지불해야 합니다.
정보과학회논문지: 시스템 및 이론 제37권 제1호(2010.10)

1. 서 론

프로세서 에너지 소비 문제는 갈수록 프로세서 설계 시에 그 중요성이 강조되어 가고 있다. 더욱이, 반도체 공정이 발전됨에 따라서 동적 에너지 소비에 비해서 누설 에너지 소비가 전체 프로세서 에너지 소비에서 차지하는 비중이 커지면서, 누설 에너지 소비 감소가 대단히 중요한 문제로 대두되고 있다. 특히, 최신의 마이크로프로세서(Microprocessor)는 2차 캐시에 많은 트랜지스터(Transistor)를 할애하기 때문에, 2차 캐시 누설 소비를 더욱 효율적으로 관리해야 할 필요가 있다.

기존의 누설 에너지 감소 연구들은 단일 프로세서 환경에서 진행되었으나, 앞으로는 단일 프로세서(Single-

processor)에 비해서 발열 제어 및 저전력 고성능 컴퓨팅에 보다 더 유리한 CMP(Chip Multi-Processor)와 같은 다중 프로세서(Multiprocessor)가 주류가 될 것으로 예상되기 때문에 새로운 환경에 맞춘 누설 에너지 관리 기법이 필요하다.

Cache decay[1]와 같은 캐시 라인 턴오프(Cache Line Turn-Off) 기법은 단일 프로세서 환경에서 고안된 대표적인 아키텍처 수준의 캐시 누설 에너지 관리 기법이다. Cache decay는 일정 시간 이상의 시간 간격 동안 접근이 되지 않은 캐시 라인을 슬립 모드로 전환시켜서 불필요한 누설 전력 발생을 차단한다. 그러나 cache decay는 단일 프로세서 환경에서 제안된 기법이기 때문에 다중 프로세서 환경에서 발생하는 특수한 문제들에 대한 고려는 되어 있지 않다. 이는 cache decay 기법을 CMP 시스템에 적용했을 때 그 효율을 떨어뜨리는 원인이 된다.

그림 1은 본 연구에서 대상으로 하는 CMP 구조를 나타낸다. 각 프로세서 코어는 개별 명령/데이터 1차 캐시를 갖고 있으며, 이들 개별 캐시들은 일관성 유지 프로토콜을 통해서 일관성을 항상 유지하게 된다. 2차 캐시는 공유 통합형 캐시 구조를 가지며, 1차 캐시들과 공유 버스를 통해서 연결된다. 본 연구에서의 모든 기법 적용 및 실험은 2차 공유 캐시에서 이루어진다.

위와 같은 구조의 CMP에서는 동시에 수행되는 여러 태스크가 하나의 2차 캐시를 공유해서 사용하기 때문에, 각 태스크는 자신이 필요로 하는 데이터를 캐시에 저장하기 위해서 다른 태스크가 점유하고 있는 캐시 라인을 빼앗으려는 경향을 보인다. 태스크가 다른 태스크의 캐시 데이터를 캐시 밖으로 쫓아내면서 자신의 데이터를 캐시에 저장하려고 하는 이러한 현상을 캐시 간섭(Cache Interference), 혹은 캐시 경쟁(Cache Contention)이라고 한다[2]. 기존에는 캐시 간섭으로 인해서 발생하는 캐시 미스 증가에 초점을 맞추고 연구가 진행되어 왔으나 캐시 간섭은 캐시 미스로 인한 성능 저하 뿐 아니라 저전력 컴퓨팅 측면에서도 악영향을 가져온다. Cache decay 기법을 공유 캐시에서 적용하면 캐시 간섭 현상으로 인해서, 이미 꺼져있던 캐시 라인들이 다른 태스크들의 캐시 접근으로 인해서 상대적으로 일찍 깨어나거나, 혹은 캐시 라인이 쫓겨나기 직전의 아무런 접근이 없는 시간(Dead Time)을 decay timeout 보다 짧게 만들어서 아예 꺼지지 못하도록 만든다. 이러한 현상들은 cache decay를 통해서 얻을 수 있는 캐시 누설 전력 감소를 줄어들게 한다.

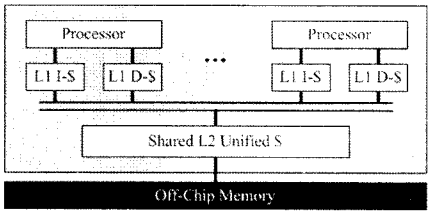


그림 1 공유 2차 캐시를 갖는 대상 CMP 시스템 구조

캐시 간섭을 줄이기 위해 이전부터 캐시 파티션(Cache Partition) 기법들이 연구되어 왔다. 캐시 파티션 기법은 동시에 수행되는 각 태스크에게 자신만의 배타적인 캐시 영역을 할당함으로써 다른 태스크의 캐시 접근에 의한 간섭을 받지 않도록 하는 기법이다. 그러나 기존의 캐시 파티션 기법은 cache decay와 함께 적용하기 어려운 캐시 동작 모니터링 장치를 사용하고 있을 뿐 아니라, 이 모니터링 장치를 구현하는 데 상당히 큰 에너지 부하가 요구되기 때문에 저전력 프로세서 설계 시에는 적합하지 않다.

Cache decay의 타임아웃을 수행 중에 조절하는 기존의 적응형 타임아웃 기법들도 역시 단일 프로세서 환경에서 제안됐기 때문에 다중 프로세서 환경에서 몇 가지 문제점이 발생한다. 공유 캐시가 보는 캐시 접근 패턴은 캐시를 공유하는 여러 태스크들의 평균적인 패턴이기 때문에 관측한 평균 패턴에 근거해서 하나의 타임아웃을 전체 태스크들에게 동일하게 적용해서는 실제 각 태스크의 특성을 제대로 반영하기 어렵다.

본 논문에서는 cache decay와 함께 적용될 수 있는 에너지 부하가 적은 태스크 별 캐시 히트/미스 예측 기법과 태스크 별 캐시 라인 슬립 시간 예측 기법을 제안한다. 그리고 예측 기법들을 통해서 얻은 정보를 이용해서 캐시 누설 에너지 소비는 크게 줄이면서, 캐시 미스는 많이 늘리지 않는 캐시 라인 슬립 시간을 고려한 캐시 파티션 전략을 제안한다. 또한, 각 태스크 별 캐시 접근 인터벌을 분석해서 태스크에 따라 다른 타임아웃을 적용하는 태스크 특성을 고려한 적응형 타임아웃 기법을 제안한다.

실험 결과 캐시 라인 슬립 시간을 고려한 캐시 파티션 기법을 통해서 캐시 간섭을 줄여, cache decay를 통해 얻을 수 있는 누설 에너지 소비 감소 효과를 증가시켰다. 뿐만 아니라 각 태스크 특성을 고려한 적응형 타임아웃 조절 기법을 통해서 캐시 누설 에너지 소비를 감소시켰다. 공유 2차 캐시에서 다른 기법은 사용하지 않고 cache decay만을 사용했을 때의 누설 에너지 소비에 비해, 제안한 기법들을 통해서 2-way CMP에서 평균 73%, 4-way CMP에서 평균 56%의 누설 에너지 소비를 감소시켰다.

2. 관련 연구

Gated-V_{dd}는 슬립 트랜지스터(Sleep Transistor)를

사용해서 SRAM 회로에 전력 공급을 끊음으로써 누설 전류 발생을 차단하는 기법이다. 이 기법은 누설 전력 소비를 완전히 차단하는 반면에, 전력 공급이 끊어진 SRAM 회로는 저장되어 있던 상태(State)를 잃어버리는 단점이 있다. Cache decay 기법은 Gated-V_{dd} [3]를 사용한 대표적인 캐시 누설 관리 기법이다. Cache decay는 일반적으로 캐시 라인이 짧은 시간 사용되고 나면 그 이후에는 대단히 긴 시간 동안 접근되지 않을 것이라는 관측에 그 기반을 두고 있다. Cache decay에서는 decay counter를 사용해 일정 시간 동안 외부에서의 접근이 없었던 캐시 라인을 찾아낸다. 이러한 캐시 라인은 앞으로 더 이상 쓰이지 않을 것으로 생각해 Gated-V_{dd} 기법을 이용해 전력 공급을 끊음으로써 캐시 라인에서 불필요하게 발생하는 누설 전력 소비를 차단한다.

앞에서 언급했듯이, cache decay는 공유 캐시에서 발생하는 캐시 간섭에 의해서 그 동작을 방해 받게 되는데, 이러한 문제는 캐시 간섭을 제거함으로써 해결할 수 있다. 캐시 간섭으로 인한 성능 저하의 문제는 기존부터 지적되어 왔으며, 캐시 간섭을 제거함으로써 성능을 높이고자 하는 연구가 진행 되어 왔다. 캐시 파티션(Cache Partition) 기법은 캐시 간섭을 제거하기 위해서 제안된 기법들 중의 하나이다[2,4-8].

기존의 캐시 파티션 기법은 대부분 marginal gain counter[4](혹은 stack distance profiling[6])를 이용한 캐시 모니터링 하드웨어를 기반으로 수행 중인 태스크들의 동작을 분석하고, 이를 기반으로 파티션 공간 할당에 따른 캐시 히트/미스를 예측한다. 하지만, marginal gain counter는 캐시 라인이 캐시 라인 턴오프 기법에 의해서 꺼졌을 때의 동작에 대해서 명확히 규정되어 있지 않기 때문에 cache decay와 함께 적용했을 때 원하는 결과를 보장하기 어렵다. 그 뿐 아니라 marginal gain counter를 구현하는 데 상당히 큰 에너지 부하가 요구되기 때문에 저전력 프로세서 설계 시에는 적합하지 않다.

반면, 공유 L2 캐시 기반에서 큰 사이즈로 인한 긴 접근 시간과 에너지 소모 증가의 단점을 극복하기 위해 [9], [10]의 기법들은 사유 L2 캐시에서의 누설 에너지를 줄이는 기법을 제안하였다. 그러나, 이들 논문에서 제안하는 누설 에너지 관리 기법들은 사유 L2 캐시의 특성에 특화되었기 때문에, 공유 L2 캐시에 적용하기는 어렵다. 또한, 캐시 내의 블록들 간의 일관성을 효율적으로 유지하기 위해 [11]과 같은 방법이 제안되고 있으나, 본 논문에서는 간단한 구조인 공유 버스에 기반한 스눕핑 캐시 일관성(snooping cache coherence) 기법을 사용한다.

3. 캐시 누설 관리 시스템

본 연구에서 제안하는 캐시 누설 관리 시스템은 크게 슬립 시간을 고려한 캐시 파티션(Sleep-Aware Cache Partitioning, SACP)과 태스크의 특성을 고려한 적응형 타임아웃 관리(Task-Aware Adaptive Timeout Management, TATM)의 2 가지 기법으로 나뉘며, 이 기법들은 S/W와 H/W로 일부분씩 구현되어 통합된다. 본 장에서는 제안한 캐시 누설 관리 시스템이 전체적으로 어떻게 동작하는지, 그리고 SACP와 TATM 각각이 어떻게 동작 하는지 설명한다.

3.1 제안 시스템의 전체 구조

그림 2는 본 논문에서 제안하는 전체 캐시 관리 시스템의 구조와 동작을 개략적으로 보여준다. 제안하는 기법은 H/W 컴포넌트와 S/W 컴포넌트로 나뉘어서 구현된다. H/W 컴포넌트와 S/W 컴포넌트는 전체 작업을 나누어서 담당하며 이들 사이의 통신은 레지스터 테이블로 구성된 전역 자료 구조를 통해서 이루어진다. 각종 캐시 이벤트 모니터링 장치, 캐시 파티션 메커니즘, 캐시 라인 턴오프 메커니즘이 H/W 컴포넌트로 구성되고, 캐시 동작 분석 모듈, 캐시 공간 할당 관리자, 타임아웃 조절 관리자가 S/W 모듈로 구성된다.

수행 중에 발생하는 각 태스크 별 전체 캐시 히트 수와 MRU 블록에서 발생하는 캐시 히트 수를 카운터를 이용해서 측정하고 이를 캐시 히트 예측 모듈로 전달하면, 캐시 히트 예측 모듈은 이를 통해서 각 태스크 별 공간 할당에 따른 캐시 히트 수를 모델 기반으로 예측한다. 그리고 연결된 캐시 라인의 슬립 시간을 측정할 수 있도록 수정된 decay counter를 이용해서 슬립 시간을 측정하고, 각 캐시 라인이 꺼져 있었던 시간을 전부 누적해서 이 값을 캐시 라인 슬립 시간 예측 모델로 전달하면, 캐시 라인 슬립 시간 예측 모듈은 마찬가지로 각 태스크 별 공간 할당에 따른 캐시 라인 슬립 시간을 모델 기반으로 예측한다. 캐시 공간 할당 관리자는 캐시 히트 및 캐시 라인 슬립 시간 예측에 기반을 두고, 캐시 히트 수와 캐시 라인 슬립 시간을 모두 고려한 캐시 공간을 할당한다.

본 연구에서는 decay counter를 캐시 라인이 꺼지면 슬립 시간을 측정하는 데 사용하지만, 캐시 라인이 켜져 있을 경우에는 캐시 히트 인터벌의 길이를 측정하는 데도 사용한다. 변형된 decay counter를 사용해서, 각 태스크 별로 캐시 히트 인터벌을 측정하고 이를 캐시 히트 인터벌 분석기로 전달하면, 캐시 히트 인터벌 분석기는 이를 모아서 각 태스크 별로 캐시 히트 인터벌 길이에 따른 분포를 조사한다. 이를 근거로 타임아웃 조절 관리자는 해당 태스크에 어울리는 타임아웃 길이를 설

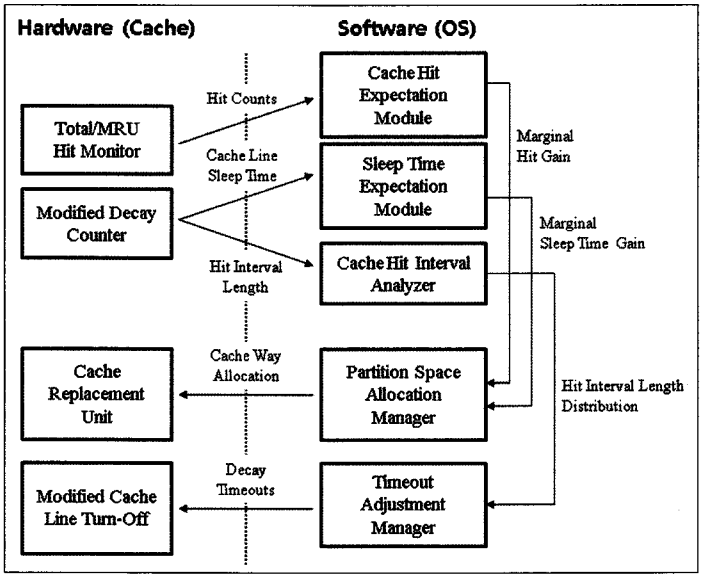


그림 2 제안한 캐시 누설 관리 시스템

정한다. 이러한 일련의 작업은 OS에 의해서 주기적으로 수행되며, 이 수행 주기를 본 논문에서는 500만 사이클로 설정하고 실험을 수행했다.

3.2 슬립 시간을 고려한 캐시 파티션 기법

본 논문에서는 한 태스크에게 하나의 캐시 웨이를 더 할당할 때 증가하는 이득을 한계 효용(Marginal Gain)으로 정의한다. 본 장에서는 캐시 히트 한계 효용과 캐시 라인 슬립 시간 한계 효용, 2가지 척도의 한계 효용을 이용한 누설 전력 소비 감소에 적합한 캐시 파티션 기법을 소개한다.

3.2.1 캐시 히트 예측 기법

일반적으로 한 태스크에게 캐시 공간을 더 많이 할당하면 할수록, 캐시 공간 할당에 의해서 얻을 수 있는 한계 효용은 줄어든다. 캐시 웨이 수만큼의 marginal gain counter를 사용하는 것은 구현에 따른 부하도 클 뿐 아니라, marginal gain counter로는 턴오프 된 캐시 라인에 의해서 발생하는 캐시 미스(Extra Cache Miss)를 턴오프와 관계없이 발생하는 캐시 미스(Ideal Cache Miss)를 구분하지 못하기 때문에 본 논문에서는 일반적인 태스크의 한계 효용 경향을 분석해서 모델 기반의 캐시 히트 예측 기법을 고안했다.

표 1은 제안하는 캐시 히트 예측 기법과 캐시 파티션 기법을 구현하기 위해서 각 프로세서 코어마다 추가해야 하는 자료 구조를 나타낸다. 각 프로세서 별로 해당 프로세서에서 수행되고 있는 태스크의 전체 캐시 히트와 MRU 블록에서 발생한 캐시 히트를 측정하는 카운터가 추가된다. 그리고 현재의 파티션 공간 할당량을 나타내는 레지스터 테이블을 추가해서 실제 파티션을 할 때 cache replacement unit이 참조하도록 하고, 이 값을 캐시 예측 모델에서도 사용한다. 또한, 그림 3과 같이, 각 캐시 라인 마다 Core ID를 추가해야 한다. 이는 각 캐시 라인의 소유자를 알 수 있게 함으로써, 캐시 히트 미스가 발생했을 때, 어느 프로세서의 카운터를 조절해야 하는지를 알 수 있게 한다.

표 1 캐시 히트 예측 기법을 위해서 프로세서 코어마다 추가하는 장치

Description	P1	P2	P3	P4
Total cache hit counter	25	10	17	9
MRU cache hit counter	19	6	2	5
Space allocation	7	3	4	2

Index	Tag	LRU data	Data	*Core ID*
…	…	…	…	…

그림 3 캐시 라인 마다 Core ID 추가

우리는 일반적인 프로그램들의 한계 효용 변화를 확인하기 위해서 SPEC CPU2000 벤치마크[12]를 사용해서, 각 프로그램을 수행시키는 중에 각 LRU 스택 위치에 발생하는 캐시 히트가 전체 캐시 히트에서 차지하는 비율을 분석 해보았다. 그림 4는 그 실험 결과를 보여준다. [7]은 이미 i번째 스택 위치에서 발생하는 캐시 히트는 한 태스크에게 할당된 캐시 웨이의 수를 (i-1)웨이에서 i웨이로 증가시킬 때의 한계 효용과 같음을 보인바 있다. 위의 그림을 통해서 일반적으로 태스크의 한계 효

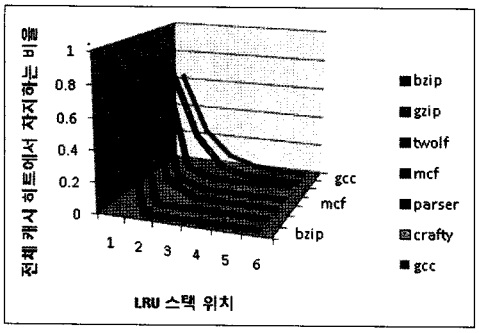


그림 4 LRU 위치에 따른 캐시 히트 비율

용은 해당 태스크에게 더 많은 캐시 공간을 할당 할수록 감소하며, 그 감소 추이는 지수함수와 유사함을 알 수 있다.

본 논문에서 제안하는 캐시 히트 예측 기법은 전체 캐시 히트 수와 MRU 블록에서 발생하는 캐시 미스 수를 수행 중에 추가한 하드웨어 카운터를 이용해서 측정하고, 이를 이용해서 캐시 히트 한계 효용을 태스크가 할당 받은 캐시 웨이에 대한 지수 함수의 형태로 모델링한다. i-1 만큼의 캐시 웨이를 할당 받은 태스크 t에게 캐시 웨이 하나를 더 할당할 때의 캐시 히트 한계 효용 $HIT_t(i)$를 다음과 같이 정의한다.

$$HIT_t(i) = c_0 c_1^{i-1}$$

태스크 t가 캐시 웨이 ω만큼을 할당받았을 때, $HIT_t(1)$은 태스크 t의 MRU 블록에서의 캐시 히트 수와 같으며, $HIT_t(\omega)$는 전체 캐시 히트 수와 같다. 태스크 t의 전체 캐시 히트 수를 $C_{TOTAL,t}$, MRU 블록에서의 캐시 히트 수를 $C_{MRU,t}$라고 할 때, 캐시 히트 한계 효용 $HIT_t(i)$는 최종적으로 다음과 같이 계산된다.

$$HIT_t(i) = C_{MRU,t} \times \left(1 - \frac{C_{MRU,t}}{C_{TOTAL,t}}\right)^{i-1}$$

Marginal gain counter 기법을 사용할 경우 각 캐시 셋마다 캐시 웨이 수에 해당하는 카운터 셋이 필요하지만, 우리의 기법은 각 캐시 셋마다 1개의 카운터만 추가하면 된다. 또한, marginal gain counter는 모든 캐시 셋에 전부 모니터 장치를 설치하는데 비해서 우리 기법은 에너지 부하를 줄이기 위해서 전체의 일부 셋(전체 캐시 셋의 1/32)에만 카운터를 설치한다. [4]에서는 부하를 줄이기 위해서 캐시 전체를 모니터 하지 않고, 일부만을 모니터 하더라도 전체를 모니터 하는 것과 유사한 결과를 얻을 수 있다는 것을 이미 보였다.

3.2.2 캐시 라인 슬립 시간 예측 기법

캐시 파티션에 활용할 정보로 가장 좋은 것은 실제의 누설 에너지 소비량을 사용하는 것이겠지만, 실시간으로 캐시에서 발생하는 누설 에너지 소비량을 측정하는 것은 어려운 일이다. 실질적으로 파티션을 통해서 누설을 얼마만큼 줄일 수 있느냐 하는 정보만 있으면 충분하기 때문에, 캐시 라인이 꺼져서 슬립 모드에 머문 시간이 누설 감소량과 정비례한다는 가정 하에서 지난주기 동안의 캐시 라인 슬립 시간을 파티션 공간 할당에 반영한다.

표 2 캐시 라인 슬립 시간 예측 기법을 위해서 추가하는 누적 캐시 라인 슬립 시간 카운터

Description	P1	P2	P3	P4
Accumulated sleep time	43	12	8	57

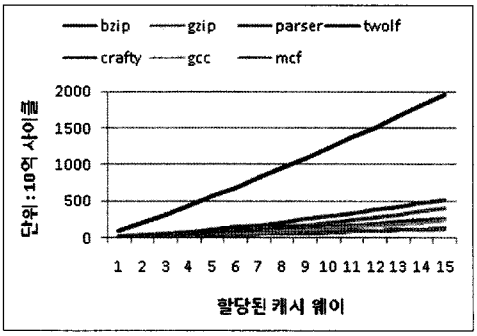


그림 5 할당된 캐시 웨이에 따른 누적 캐시 라인 슬립 시간

표 2는 우리의 캐시 슬립 시간 예측 기법을 구현하기 위해서 필요로 하는 추가 자료 구조를 나타낸다. 각 프로세서 코어에서 수행되는 태스크의 누적 캐시 라인 슬립 시간을 측정하기 위해서 각 코어마다 1개씩의 카운터를 추가한다. 그림 5는 SPEC 벤치마크들을 대상으로 고정 타임아웃 cache decay를 실행했을 때, 각 프로그램에 할당된 캐시 공간(캐시 웨이 수)에 따른 누적 캐시 라인 슬립 시간을 나타낸다. 그림 5에서 볼 수 있듯이, cache decay에 의해서 얻을 수 있는 캐시 라인 슬립 시간은 프로그램의 특성에 따라서 크게 차이가 나며, 한 프로그램에서는 할당된 캐시 웨이와 누적 캐시 라인 슬립 시간은 정비례에 가까운 모습을 보인다.

할당된 캐시 웨이와 누적 캐시 라인 슬립 시간이 정비례한다고 가정하면, 캐시 웨이 하나를 더 할당했을 때의 한계 효용은 직선 함수의 기울기가 된다. 따라서 캐시 웨이 ω를 할당 받은 태스크 t의 지난 주기 동안의 누적 캐시 라인 슬립 시간이 $C_{SleepTime,t}$일 때, 누적 캐시 슬립 시간 한계효용 $SLEEP_t$는 다음과 같이 쉽게 얻을 수 있다.

$$SLEEP_t = C_{SleepTime,t} * \omega$$

그림 6은 캐시 라인 슬립 시간을 측정할 수 있도록 변경된 decay counter의 모습을 나타낸다. Decay counter는 n-bit FSM으로 구현된다. 기존의 cache decay에서

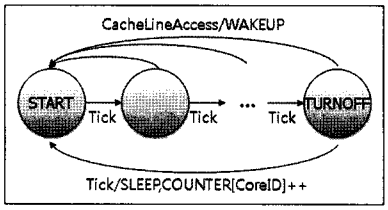


그림 6 캐시 라인 슬립 시간 측정을 위해서 변경된 decay counter

사용됐던 decay counter는 일단 turn-off 상태가 되면, 깨어날 때까지 상태가 변하지 않지만, 우리의 decay counter는 슬립 모드로 들어간 캐시 라인에 연결된 decay counter가 turn-off 상태로 전이될 때마다, 해당 캐시 라인의 소유자 태스크의 슬립 시간 카운터를 증가시킨다. 이를 통해서 수행 중에 각 태스크의 캐시 슬립 시간을 측정할 수 있다. 캐시 라인 슬립 시간을 측정하는 데 소모되는 동적 에너지 소비를 줄이기 위해서 캐시 히트 예측 기법에서와 마찬가지로, 변경된 decay counter는 전체 캐시 중에서 1/32에 해당하는 캐시 셋에만 설치되고, 나머지 decay counter는 기존의 decay counter를 그대로 사용한다.

3.2.3 슬립 시간을 고려한 캐시 파티션 전략

본 연구에서 사용하는 하드웨어 캐시 파티션 메커니즘은 column caching[7]이다. Column caching은 각 태스크마다 하나씩 주어진 bit vector를 보고, 각 태스크에 할당된 웨이에 해당 태스크의 캐시 라인을 저장하는 형태로 동적 캐시 파티션을 수행한다. Column caching은 공간 활용 측면에서는 다른 파티션 메커니즘에 비해서는 떨어지지만, victim을 선택하는데 하드웨어적으로 루프를 구현할 필요가 없기 때문에, data path도 짧게 유지할 수 있으며, 에너지 부하 및 추가적인 공간 부하도 적은 것이 장점이다.

그림 7은 column caching에서 추가로 사용하는 bit vector 테이블을 나타낸다. 그림 7의 값들은 표 2에서 각 태스크에게 할당된 양에 따라서 실제로 16-way 캐시를 분할한 모습을 표현하고 있다. 한 bit vector의 각 비트는 각각 캐시 웨이 하나에 대응되며 태스크가 해당 캐시 웨이에 캐시 데이터를 저장할 수 있는지를 표시한다. 캐시의 교체 관리자는 캐시 데이터를 지정된 캐시 웨이에만 저장하도록 수정된다. 자신의 데이터가 자신에게 해당된 캐시 웨이에 저장되어 있지 않더라도, 읽기는 제한 없이 수행 가능하다.

Description	Core 1	Core 2	Core 3	Core 4
Way allocation	0xFE00	0x01C0	0x003C	0x0003

그림 7 Column caching을 위해 필요한 자료 구조

OS는 매 파티션 주기마다 파티션 공간 재 할당 루틴을 호출해서 테이블의 값들을 재설정한다. 단, 이 시점에서는 테이블에서 각 태스크가 가져야 할 공간의 크기만 명시해두는 것일 뿐이며, 실제 물리적인 파티션 작업은 공간 재 할당 이후에 캐시 미스가 실제 발생했을 때, 캐시의 블록 교체 관리자에 의해서 서서히 수행된다.

그림 8은 본 연구에서 사용한 전체적인 파티션 공간 할당 과정을 보여준다. 공간 재 할당 루틴이 호출되면 각 태스크가 적어도 하나 이상의 캐시 웨이를 사용할 수 있도록, 각 태스크는 최소 하나씩의 캐시 웨이를 제공받는다. 모든 태스크에게 캐시 웨이를 하나씩 할당하고 난 이후에는 greedy 알고리즘 형태로 남은 캐시 공간을 할당한다. 현재 상태에서 가장 큰 한계 효용을 갖는 태스크에게 하나의 캐시 웨이를 더 할당하며, 이 과정을 더 이상 남은 캐시 공간이 없을 때까지 반복해서 수행하게 된다.

[7]에서는 한계 효용을 캐시 공간을 한 덩어리(chunk) 더 할당했을 때 증가하는 캐시 히트 1가지로 정의했지만, 우리는 각 전략에 따라서 한계 효용을 다르게 정의한다. 본 논문에서 제안하는 2가지 동적 캐시 파티션 전략의 차이는 결국 한계 효용이 가장 큰 태스크를 어떻게 정의하는가의 차이라고 할 수 있다.

그림 8은 캐시 히트 수 만을 고려해서 공간을 할당하는 알고리즘을 보여준다. 이 전략에서 태스크 t의 한계 효용은 태스크 t에 캐시 웨이를 하나 더 할당할 때 증가하는 캐시 히트 수, $P_t(w)$로 정의된다. 이 전략은 현재의 캐시 할당 상태에서 캐시 웨이를 하나 더 할당할 때 증가하는 캐시 히트가 가장 많은 태스크를 찾아서 해당 태스크에 캐시 웨이를 하나 더 할당한다. 이 과정을 더 이상 할당되지 않고 남은 캐시 웨이가 없을 때까지 반복해서 수행한다.

그림 9는 캐시 라인 슬립 시간을 캐시 히트 수와 함께 고려한 파티션 공간 할당 알고리즘이다. 우리는 캐시 누설 감소량이 캐시 라인 슬립 시간과 비례한다고 가정

CACHE HIT BASED SPACE ALLOCATION

```
alloc[NUM_CORE] := {1, 1, … , 1}
remain_space := NUM_WAY-NUM_CORE

while remain_space > 0
  max_marginal_gain := 0
  for i = 1 to NUM_CORE
    t := alloc[i]
    if Pt(t) > max_marginal_gain
      max_marginal_gain := Pt(t)
      proc_id := i
    end if
  end for
  alloc[proc_id] := alloc[proc_id]+1
  remain_space := remain_space-1
end while
return alloc
```

그림 8 캐시 히트 기반 공간 할당 알고리즘

```
SLEEP TIME AWARE SPACE ALLOCATION
alloc[NUM_CORE] := {1, 1, … , 1}
remain_space := NUM_WAY-NUM_CORE

while remain_space > 0
  max_marginal_gain := 0
  for i = 1 to NUM_CORE
    t := alloc[i]
    ω := alloc[i]/(alloc[1]+..+alloc[NUM_CORE])
    hit_gain := (1-ω)*P_i(t)
    sleep_gain := ω*(S_i(t+1)-S_i(t))
    gain := hit_gain+sleep_gain
    if gain > max_marginal_gain
      max_marginal_gain := gain
      proc_id := i
    end if
  end for
  alloc[proc_id] := alloc[proc_id]+1
  remain_space := remain_space-1
end while
return alloc
```

그림 9 캐시 라인 슬립 시간 고려 공간 할당 알고리즘

하였기 때문에, 이는 캐시 누설 감소량과 캐시 히트 수를 함께 고려한 공간 할당 전략으로 생각할 수 있다. 이 전략에서 태스크 t의 한계 효용은 태스크 t에 캐시 웨이를 하나 더 할당했을 때 늘어나는 캐시 히트 수와 캐시 라인 슬립 시간의 가중 합(weighted sum)으로 정의된다. 캐시 웨이를 하나 더 할당했을 때 증가하는 캐시 히트 수는 역시 $P_t(w)$를 사용하며, 증가하는 캐시 라인 슬립시간을 2절에서 정의한 $S_t(w)$를 이용한다. $S_t(w)$를 선형함수로 모델했기 때문에, 캐시 슬립 시간 증가량은 함수의 기울기와 동일하다.

위의 알고리즘에서는 ω로 가중치를 나타낸다. 가중치 ω가 클수록 캐시 블록 슬립 시간 이득이 공간 할당에 미치는 영향이 커지게 된다. 우리는 태스크가 캐시 공간을 적게 가졌을 때는 캐시 히트 수를 우대하고 캐시 공간을 많이 가졌을 때는 캐시 슬립 시간을 우대하도록 가중치 ω를 설정했다.

3.3 태스크 특성을 고려한 적응형 타임아웃 관리 기법

기존의 cache decay 기반의 타임아웃 관리 기법들은 한 CPU에서는 동시에 한 개의 태스크만이 수행되는 환경을 대상으로 했다. 따라서 하나의 캐시에는 한 태스크의 데이터만이 저장되기 때문에, decay 타임아웃을 현재 수행 중인 태스크 1개에 최적화하는 것이 가능했다. 하지만 CMP처럼 서로 다른 특성을 갖는 여러 개의 태스크가 동시에 수행되는 환경에서는 공유 캐시의 입장에선 모든 태스크들의 평균적인 행동 패턴을 인식할 수 밖에 없기 때문에 기존의 기법들은 이러한 평균적인 패턴에 최적화해서 타임아웃을 조절하게 되며, 이는 실제 태스크 개개의 특성을 반영하기 어렵다.

그림 10은 2-way CMP에서 bzip과 parser를 함께 수행시켰을 때 얻은 실험결과이다. 아래와 가운데 막대는 각각 bzip과 parser의 활성 인터벌(live interval) 분포를 나타내고, 위 막대는 전체 활성 인터벌 분포를 나타낸다. 아래 막대는 bzip의 경우 99% 이상의 활성 인터벌이 100k 사이클 이하의 길이를 갖는 비교적 짧은 활성 인터벌이라는 것을 보여준다. 그러나 가운데 막대를 보면 parser의 경우에는 100k 사이클 이하의 길이를 갖는 활성 인터벌의 분포가 전체의 50%를 넘지 않는다는 것을 확인할 수 있다. 전체 활성화 인터벌의 90% 정도를 캐시 히트로 보호하면서 cache decay를 사용하려면, bzip의 경우에는 100k 사이클 정도의 타임아웃으로도 충분하지만 parser의 경우에는 1000k 사이클 이상의 긴 타임아웃이 필요하다.

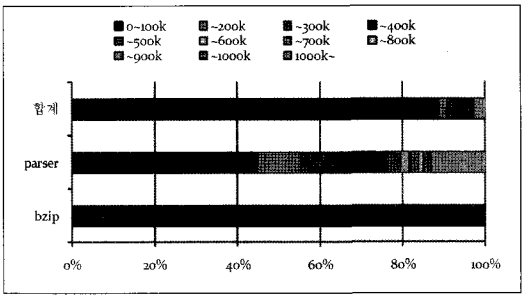


그림 10 태스크에 따른 전체 활성 인터벌 분포

위 막대의 분포를 알고 있다면, 전체 활성 인터벌의 90% 이상을 캐시 히트로 보호하기 위해서는 100k~200k 사이클의 타임아웃으로도 충분하다고 생각할 수 있다. 이 경우 parser의 수많은 활성 인터벌은 decay 기법에 의해서 추가적인 캐시 미스가 되어버린다. 즉, 공유 캐시의 입장에서는 동시에 수행되는 태스크들의 평균적인 캐시 접근 패턴을 보게 되기 때문에 실제 태스크의 특성을 충분히 반영한 최적화된 타임아웃 길이를 결정하기 어렵게 된다.

만약 각 태스크 별 특성을 이미 알고 있을 때에도, 모든 태스크들이 하나의 같은 타임아웃을 공유해야 한다면, 또 다른 문제가 발생한다. 하나의 타임아웃을 공유해야 한다면, 보다 긴 타임아웃을 요구하는 태스크 parser에 맞추어서 타임아웃을 설정해야 한다. 이럴 경우 bzip의 캐시 라인들은 불필요하게 많은 시간을 누설 전류를 발생시키면서 타임아웃을 기다리게 되며, 이는 에너지 측면에서 낭비가 될 수 있다. 따라서, 본 논문에서는 동시에 수행되는 각 태스크마다 자신만의 타임아웃을 하나씩 제공하기 위한 기법을 제안한다. 이 타임아웃은 해당 태스크의 모든 캐시 라인에게 동일하게 적용되며, OS에서 각 태스크에 맞는 최적의 타임아웃을 설정한다.

기존 cache decay 기법에서는 decay counter로 2비트 FSM을 사용했지만, 우리는 3비트 FSM을 사용한다. 2비트 FSM이 3비트 FSM에 비해서 그 자체가 갖는 추

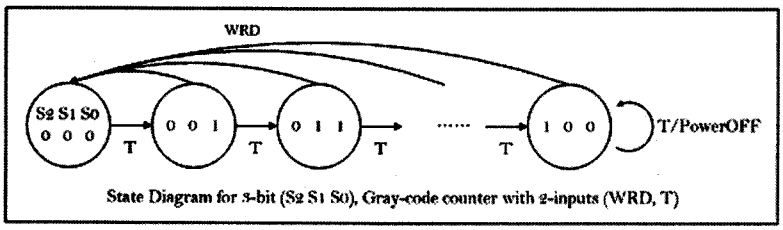
Description	Core 1	Core 2	Core 3	Core 4
Timeout (# of ticks)	$110_{(2)}$	$011_{(2)}$	$001_{(2)}$	$111_{(2)}$

그림 11 타임아웃 차등 적용을 위해 추가된 자료 구조

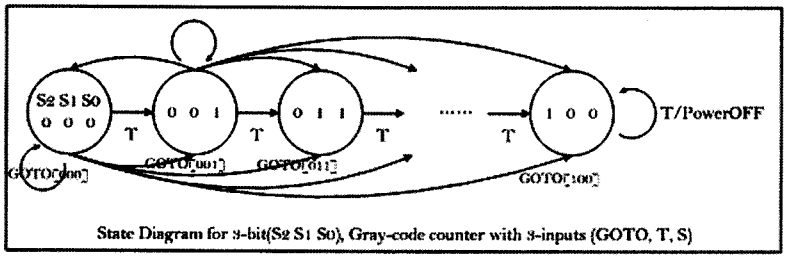
가적인 에너지 부하는 적지만, 3비트 FSM을 사용하면 2비트 FSM을 사용하는 것에 비해서 더욱 세세하게 타임아웃을 조절할 수 있다. 그림 11은 각 태스크의 타임아웃을 저장하는 타임아웃 테이블을 나타낸다. 각 태스크가 캐시 라인에 접근할 때마다 캐시 라인에 연결된 FSM은 상태를 초기화하게 되는데, 이 때 상태를 테이블에 저장된 값의 상태로 FSM을 전이시키게 된다. FSM 상태를 그레이 코드(gray code)로 인코딩(encoding)하기 때문에, 타임아웃 테이블에도 마찬가지로 그레이 코드로 인코딩을 해서 저장한다. OS에 의해서 타임아웃 재설정 루틴이 호출될 때마다 위의 테이블은 현재 수행 중인 태스크의 동작 형태에 맞추어서 재조정된다. 테이블에 저장된 타임아웃은 태스크가 캐시 라인에 접근할 때 마다 FSM에 입력으로 전달된다.

그림 12의 (a)는 기존 cache decay에서 decay counter로 사용했던 FSM의 다이어그램이다. 외부에 있는 전역 카운터에서 일정 시간이 지날 때 마다 틱(tick) T를 각 캐시 라인으로 전달한다. 각 캐시 라인에 연결된 FSM은 T가 입력될 때 마다 현재의 상태에서 다음 상태로 전이를 하게 된다. 마지막 turn-off 상태에서 T를 받으면, FSM은 캐시 라인을 끄도록 하는 신호를 발생시킨다. 캐시 라인에 접근이 발생할 때마다 FSM으로 WRD 신호가 입력되며 그 때 마다 FSM은 초기 상태로 전이되어 counter를 리셋한다. 그림 12의 (b)는 타임아웃 차등 적용을 위해서 변형된 FSM을 나타낸다. (a)와 마찬가지로 3비트 FSM으로 구현되어 있지만, 외부로부터 3종류의 입력을 받도록 구현되어 있다. 전역 카운터에서 발생된 틱 T가 입력되었을 때의 동작은 (a)와 동일하다. (a)와 (b)의 차이점은 캐시 라인에 접근이 발생했을 때의 동작이다. (a)에서는 캐시 라인에 접근이 발생했을 때 WRD 신호가 입력되며 항상 초기 상태로 전이된다. 하지만 (b)는 캐시 라인에 접근이 발생하면 접근을 발생시킨 태스크의 타임아웃 테이블에서 전이 할 상태를 읽어서 이를 GOTO 신호와 함께 FSM의 입력으로 넣는다. FSM은 이 입력에 의해서 해당 상태로 바로 전이를 하게 되고, 이를 통해서 태스크 마다 다른 타임아웃을 적용할 수 있다. 이 작업은 실제 캐시 라인에서 데이터를 읽거나 쓰는 작업과 병렬적으로 수행할 수 있기 때문에 data path의 길이를 늘이지 않고 수행할 수 있다.

각 태스크의 캐시 라인 턴오프 때문에 부수적으로 발생하는 캐시 미스로 인해서 활성 인터벌의 분포가 잘못 측정되지 않도록 하고자 했다. 그래서 캐시 라인 턴오프 여부에 관계없이 캐시 히트/미스 여부를 판단할 수 있도록, 캐시 라인이 슬립 상태로 전환되더라도 태그 필드에는 계속 전력을 공급하도록 구성했다. 다만, 태그 필드에 전력을 계속 공급할 경우에 누설 전류가 발생하기 때문에, 모니터링에 따르는 에너지 부하를 줄이기 위해서 활성 인터벌을 모니터하는 영역을 전체 캐시 중 일부 캐시



(a) Cache decay에서 decay counter로 사용된 FSM



(b) 타임아웃 차등 적용을 위해서 반영된 FSM

그림 12 기존의 decay counter와 변형된 decay counter 비교

셋(전체의 1/32)로 한정하고, 모니터하는 영역의 캐시 셋에만 태그 필드에 전력을 항상 공급하게 했다.

본 기법을 위해 인터벌 분포 분석을 위해 각 프로세서 코어 마다 2FSM 비트수 개의 활성 인터벌 카운터를 갖도록 하였다. 각 활성 인터벌 카운터는 해당 길이만큼의 활성 인터벌이 지난 파티션 주기 동안 몇 번 발생했는가를 나타낸다. 활성 인터벌 카운터는 파티션 주기 동안 정보를 수집하며, 새로운 파티션 주기가 시작될 때 리셋된다. 활성 인터벌 카운터는 캐시 히트가 발생할 때 마다 캐시 라인의 FSM이 지난번 접근으로부터 몇 번의 틱(tick)이 지났는지에 따라서 업데이트 된다. 캐시 라인이 꺼져 있어서 실제로 다시 메모리에서 읽어 와야 하더라도 캐시 태그에서 매치가 있으면 히트로 간주하고 카운터를 증가시킨다. 만약 이번에 접근한 캐시 라인의 이전 소유자가 태스크 1이었고, 지난 번 접근했을 때로부터 3 번의 상태 전이가 발생했다면, 태스크 1의 3번 활성 인터벌 카운터의 값을 1 증가시킨다. 태스크 t의 i번 활성 인터벌 카운터의 값을 Live(p,i)라고 할 때, AccLive(p,i)를 우리는 다음과 같이 정의한다. 즉, AccLive(p,i)는 0번 활성 인터벌 카운터부터 i번 활성 인터벌 카운터까지의 누적 값을 의미한다.

$$AccLive(p,i) \equiv \sum_{k=0}^{i} Live(p,k)$$

그림 13은 각 태스크 별로 어떻게 타임아웃을 설정하는지를 나타내는 알고리즘이다. 위의 알고리즘에서 τ는 우리가 전체 캐시 히트 중에 캐시 라인 턴 오프를 하더라도 미스로 바꾸지 않고 보존하고 싶은 캐시 히트의 비율을 나타내는 역치(threshold)값을 나타낸다. 우리는 최대한 보수적(conservative)으로 접근하기 위해서 τ를 0.9로 설정했다. 즉, 위의 알고리즘은 각 태스크 마다 (전체 캐시 히트 수 * τ) 보다 많은 캐시 히트를 보호하는 가장 작은 타임아웃을 찾아서 이를 태스크의 타임아웃으로 설정하는 알고리즘이다. 타임아웃 재설정 루틴은 매 주기가 끝날 때 OS에 의해서 새로 호출되고, OS는 최종적으로 이렇게 설정된 타임아웃을 그레이 코드로 변형해서 타임아웃 테이블에 저장한다. 그리고 활성 인터벌 카운터를 초기화한다.

PROCESS-AWARE TIMEOUT MANAGEMENT

```
timeout[NUM_CORE] := {0, 0, ···, 0}
max_fsm := 2^NBIT_FSM-1
for i = 1 to NUM_CORE
  for j = 0 to max_fsm
    if (AccLive(i,j)/AccLive(i,max_fsm)) > τ
      timeout[i] := j
      exit inner loop
    end if
  end for
end for
return timeout
```

그림 13 태스크 특성 고려 타임아웃 설정 알고리즘

4. 실험 결과

본 논문에서 제안한 파티션 기법들과 타임아웃 관리 기법에 대한 성능 평가를 위해서 CATS[13] 멀티 프로세서 시뮬레이터를 사용하였다. CATS를 우리의 실험 환경에 맞도록 수정해서 기법을 적용하였다. 표 3은 실험 파라미터를 보여준다. 2-way CMP 환경과 4-way CMP 환경에서 각 프로세서 코어에 각각 다른 어플리케이션이 수행되는 멀티 프로그램(multi-programmed) 형태로 벤치마크를 수행해서 실험했다.

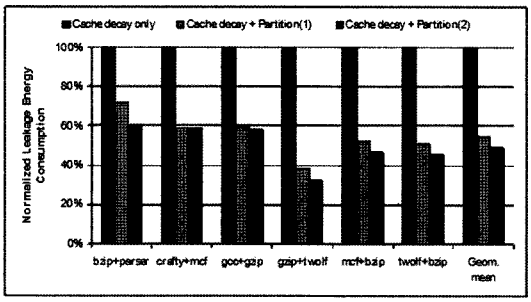
표 3 실험 파라미터

Parameter	Value	
Processor	2-way, 4-way CMP	
Cache	L1 Inst/Data	8KB,4-way(LRU), 64B block, 2 cycle latency
	Shared L2	1MB, 16-way(LRU), 64B block, 20 cycle latency
Memory	260 cycle latency	

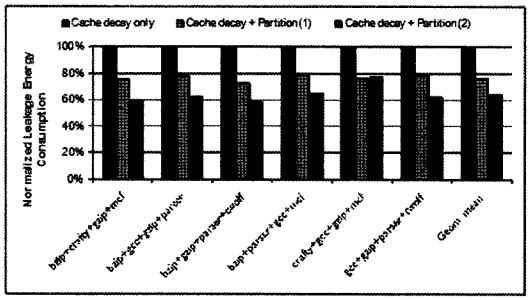
각종 캐시 누설 관리 기법은 L2 공유 캐시에서 구현되고 수행했으며, L1 개별 캐시와 L2 공유 캐시는 공유 버스(shared bus)를 이용한 연결을 가정한다. 프로세서 코어, L1 캐시, L2 캐시는 온칩(on-chip)에 함께 있으며 오프칩(off-chip)에 공유 메모리를 가지고 있는 형태의 구조를 가정한다. 캐시 및 추가로 사용된 자료 구조의 각종 파라미터는 CACTI-4.2[14]를 통해서 얻었다.

우리는 SPEC CPU2000 정수형 벤치마크에서 몇 개의 프로그램을 선정해서 무작위로 쌍을 맺어서 수행시키는 형태로 실험을 진행했으며, 동시에 수행되는 태스크는 공유 라이브러리를 제외한 공유 데이터는 가지지 않는다고 가정했다. 본 실험에서는 17비트 전역 카운터(global counter)와 3비트 decay counter를 사용해서 기본 타임아웃을 1024K 사이클로 설정했다. 전역 카운터는 128K 사이클마다 한 번씩 틱을 발생시키며, 그 때마다 각 캐시 라인의 decay counter도 하나씩 카운트를 수행한다. 기본적으로 decay counter가 8번 카운트를 하는 동안 캐시 라인에 접근이 없으면 해당 캐시 라인은 슬립 상태로 전환된다.

그림 14를 통해서 캐시 파티션 기법이 CMP 환경에서 cache decay 기법의 누설 에너지 절감 효과를 확대한다는 사실을 확인할 수 있다. 캐시 파티션을 수행했을 때 2-way CMP와 4-way CMP 양쪽에서 모두 큰 누설 에너지 감소가 있었다. 범례에서 Partition(1)은 캐시

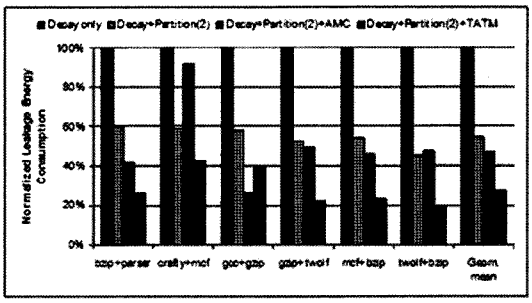


(a) 2-way CMP에서의 누설 에너지 소비 감소

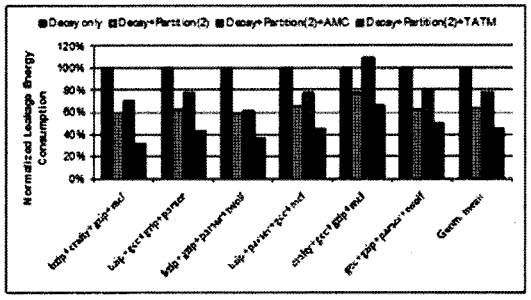


(b) 4-way CMP에서의 누설 에너지 소비 감소

그림 14 동적 캐시 파티션의 누설 에너지 절감 효과



(a) 2-way CMP에서의 누설 에너지 소비 감소



(b) 4-way CMP에서의 누설 에너지 소비 감소

그림 15 적응형 타임아웃 조절의 누설 에너지 절감 효과

히트 기반의 파티션 전략(전략1)을 사용했을 때의 실험 결과를 의미하고, Partition(2)는 본 연구에서 제안한 캐시 라인 슬립 시간을 함께 고려한 캐시 파티션 전략(전략2)을 사용했을 때의 실험 결과를 의미한다.

2-way CMP와 4-way CMP의 모두에서 Partition(2)가 Partition(1) 보다 더 큰 누설 에너지 감소 효과를 보였다. 2-way CMP에서의 누설 에너지 소비 감소가 4-way CMP에서의 누설 에너지 소비 감소 보다 큰 것은 캐시 공간이 상대적으로 여유 있고, 캐시 간섭이 적기 때문이다. 재미있는 사실은 2-way CMP가 누설 에너지 감소는 4-way CMP 보다 크지만, Partition(2)가 Partition(1)에 비해서 갖는 누설 에너지 감소의 효과는 반대로 4-way CMP에서 더욱 크게 나타난다. 이는 제안한 캐시 라인 슬립 시간을 고려한 파티션 전략이 캐시 간섭이 큰 환경에서의 누설 에너지 관리에 있어서 기존의 파티션 전략 보다 더 적합하다는 것을 보여준다.

그림 15는 적응형 타임아웃 조절 기법의 누설 전력 감소 효과를 보여준다. TATM(Task-Aware Timeout Management)은 본 논문에서 제안하는 태스크의 특성을 고려한 적응형 타임아웃 조절 기법이다. 제안한 태스크 특성 고려 타임아웃 관리 기법의 성능을 비교 평가하기 위해서 기존 단일 프로세스 환경에서 제안된 적응형 타임아웃 조절 기법인 AMC[15]와 비교했다.

AMC는 미리 다음 주기의 ideal miss와 sleep miss를 예측해서 이상적인 타임아웃으로 타임아웃을 설정하는 것이 아니라, 타임아웃을 한 번씩 늘이거나 줄이는 형태의 적응형 타임아웃 조절 기법이기 때문에, 캐시 미스 수에 부담을 줄 수 밖에 없고, 이상적인 타임아웃을 찾아가는 데 걸리는 시간이 우리가 제안한 기법에 비해서 오래 걸리는 단점이 있다. AMC는 각 태스크 별로 ideal miss와 sleep miss를 분석하는 것이 아니기 때문에, 함께 수행되는 태스크들의 평균적인 동작에 의해서 타임아웃을 조절할 수 밖에 없고, 이 때문에 불리한 타임아웃으로 괴로워하는 태스크가 생길 수 밖에 없다. 또한, 모든 태스크의 평균적인 동작을 보기 때문에 타임아웃을 조절할 수 있는 기회를 놓치는 경우도 상당히 많다. AMC는 ideal miss와 sleep miss를 구분하기 위해서 캐시의 태그 배열을 모두 켜둔 상태로 유지한다. 이 때문에 태그 배열에서 상당한 누설 전력이 발생하게 되는 문제점이 있다. 그리고, 전역 카운터가 틱을 발생시킬 때 마다 모든 캐시 라인이 전역 타임아웃 레지스터에 접근하는 큰 동적 에너지 부하가 있다. 또한 모든 캐시 라인이 한꺼번에 전역 타임아웃 레지스터에 접근하기 때문에 생기는 부가적인 문제점도 함께 존재한다.

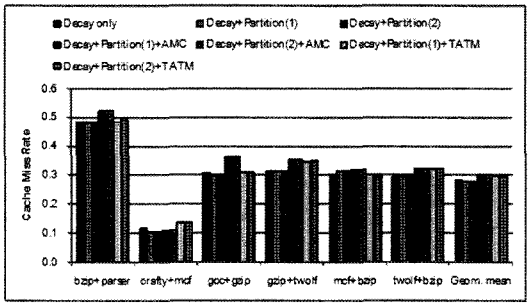
TATM을 Partition(2)와 함께 사용했을 때의 누설 에너지 소비량은 cache decay만을 사용했을 때에 비해서 평균적으로 2-way CMP에서 72.5%, 4-way CMP에서 55.8% 감소했으며, Partitition(2)를 cache decay

와 함께 사용했을 때에 비해서는 평균적으로 2-way CMP에서 49.7%, 4-way CMP에서 30.9% 감소했다. 즉, TATM을 사용함에 따라서 각 태스크가 적용 받는 타임아웃 길이가 짧아지기 때문에 무의미하게 대기하는 시간이 감소하고, 이로써 더 많은 누설 에너지 소비를 감소시킬 수 있다.

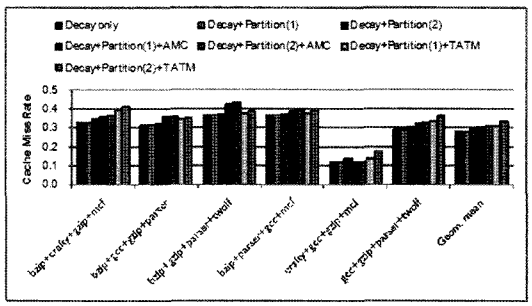
AMC는 TATM과는 다른 실험 결과를 보여준다. AMC는 2-way CMP에서는 평균적으로 누설 에너지 소비를 줄여주는 것으로 보인다. 하지만, 그 감소량이 TATM을 사용했을 때에 비해서 대단히 작은 것을 볼 수 있다. (crafty+mcf) 벤치마크에서는 오히려 에너지 소비가 증가하는 것을 볼 수 있는데, 그 이유는 AMC가 거의 타임아웃 시간을 줄이지 못하면서, AMC 구현에 따른 에너지 부하에 따른 에너지 소비가 늘어났기 때문이다. AMC가 타임아웃 시간을 줄이지 못한 것은 두 태스크의 캐시 접근이 섞이게 되면서 그 접근 패턴을 제대로 파악하지 못했기 때문에 발생한 것이다. 4-way CMP에서의 실험에서는 에너지 소비가 감소된 벤치마크가 하나도 존재하지 않는다. CMP의 프로세서 코어가 늘어날수록 캐시 접근이 섞이는 정도가 커지게 되고, 이에 따라서 접근 패턴을 파악하기가 점점 어려워져서 타임아웃을 줄일 수 있는 기회를 포착하는 것이 힘들어지기 때문이다. AMC와 TATM의 비교는 결국 멀티 프로세스 환경에서는 캐시 전체의 패턴을 분석하는 것 보다는 수행되는 각각의 태스크의 접근 패턴을 구분해서 분석하고 또 그 각각의 패턴에 따른 정책 결정이 필요함을 보여준다.

본 연구에서는 캐시 라인을 끔으로써 누설 에너지 소비를 없애기 때문에 캐시 라인을 끔으로 인한 추가적인 캐시 미스가 발생하게 된다. 캐시 미스로 인한 성능 저하 및 추가적인 메모리 접근은 전체 에너지 소비를 증가시키는 원인이 되기 때문에 누설 전력 소비를 줄일 때 캐시 미스 증가를 가능한 억제해야 한다. 그림 16은 각 기법에 따른 캐시 미스율을 보여준다. 캐시 파티션만을 cache decay와 함께 적용했을 경우에는 캐시 미스율의 변화가 그리 크지 않다. 약간 감소하거나, 약간 증가하는 수준에 불과하다. 즉, 제안한 파티션 전략은 캐시 미스를 별로 늘리지 않으면서, cache decay에 의해서 절약되는 누설 전력의 양을 크게 증가시킴을 위의 실험결과에서 확인할 수 있다.

제안한 캐시 파티션 전략이 캐시 미스에 큰 영향을 끼치지 않는 반면에 실질적으로 decay 타임아웃을 조절하게 되는 적응형 타임아웃 기법은 캐시 미스에 큰 영향을 준다. 이는 타임아웃을 짧게 유지하게 되면 캐시 라인을 잘못 꺼버리기 쉽고 이는 곧 캐시 미스의 증가를 야기하기 때문이다. 기존 단일 프로세서 환경에서 제안되었던 AMC 및 본 논문에서 제안한 TATM 모두 캐시 미스를 증가시킨다. 그렇지만 AMC나 TATM 모두 10% 이내의 캐시 미스율 증가를 보인다. 이는 적응형 타임아웃 기법을 사용해서 얻을 수 있는 누설 전력 소비 감소에 비하면 대단히 작은 양이기 때문에 의미가 있다고 할 수 있다.



(a) 2-way CMP에서의 캐시 미스율 변화



(b) 4-way CMP에서의 캐시 미스율 변화

그림 16 기법 적용에 따른 캐시 미스율 변화

5. 결론 및 향후 작업

기존의 cache decay 기반의 캐시 누설 관리 기법은 단일 프로세서 환경에서 제안되었기 때문에 CMP와 같은 멀티 프로세서 환경에서는 기존에 없었던 문제점을 갖게 된다. 우리는 기존의 cache decay 기반 기법이 멀티 프로세서 환경에서 어떤 문제점을 갖는지를 분석하고 이를 해결하는 기법을 제안하였다.

본 논문에서는 동적 캐시 파티션을 이용해서 캐시 간섭을 줄임으로써 cache decay 기반 캐시 누설 관리 기법의 효과를 극대화 시키는 기법을 제안하였다. 파티션 전략으로는 캐시 미스 기반의 전략과 캐시 미스뿐 아니라 캐시 라인 슬립 시간을 함께 고려하는 전략을 제안해서 수행 시간과 누설 전력 절감을 함께 추구했다. 또한 기존의 cache decay 기반 타임아웃 관리 기법과 달리 동시에 수행 중인 서로 다른 태스크의 특성을 분석해서 각 태스크의 캐시 라인에 다른 타임아웃을 적용하는 기법을 제안했다.

실험을 통해서 제안한 기법이 공유 L2 캐시에서 cache decay만을 사용했을 때에 비해, 2-way CMP에서 평균 73%, 4-way CMP에서 평균 56%의 누설 에너지 소비 감소를 얻을 수 있음을 보였다.

본 연구에서는 캐시 파티션 메커니즘으로 column caching[7]을 사용했지만, column caching을 사용할 때는 파티션이 수정될 때 파티션 경계가 움직이는 일이 잦아서 기존 경계와 중첩되는 부분이 많이 생겨서 이 부분에서 발생하는 간섭문제가 있을 수 있다. 파티션이 변경될 때 비교적 경계 이동의 부담이 적은 파티션 메커니즘의 연구도 진행할 예정이다.

본 연구는 2-way, 4-way와 같은 비교적 적은 수의 프로세서 코어가 있는 CMP 환경을 대상으로 연구했지만, 프로세서 코어가 많아지면 제안한 파티션 알고리즘이 늘어난 캐시 간섭을 제대로 통제하지 못해서 원하는 결과를 얻지 못할 수도 있다. 따라서 많은 수의 프로세서 코어가 있는 환경에서의 캐시 누설 관리 기법을 연구할 계획이다.

참 고 문 헌

[1] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache decay: exploiting generational behavior to reduce cache leakage power," *Proc. of ISCA*, pp.240-251, 2001.

[2] Y. Solihin, F. Guo, and S. Kim, "Predicting cache space contention in utility computing servers," *Proc. of IPDPS*, pp.8-15, 2005.

[3] M. D. Powell, S. Yang, B. Falsafi, K. Roy, and T. B. Vijaykumar, "Reducing leakage in a high-performance deep-submicron instruction cache," *IEEE Trans. on VLSI*, vol.9, no.1, pp.77-89, 2001.

[4] M. Qureshi and Y. Patt, "Utility-based cache partitioning: a low-overhead, high-performance, run-time mechanism to partition shared caches," *Proc. of MICRO*, pp.423-432, 2006.

[5] N. Rafique, W-T. Lim, and M. Thottethodi, "Architectural support for operating system-driven CMP cache management," *Proc. of PACT*, pp.2-12, 2006.

[6] D. Chandra, F. Guo, S. Kim, and Y. Solihin, "Predicting inter-thread cache contention on a chip multi-processor architecture," *Proc. of HPCA*, pp.340-351, 2005.

[7] G. E. Suh, L. Rudolph, and S. Devadas, "Dynamic partitioning of shared cache memory," *Journal of Supercomputing*, vol.28, no.1, pp.7-26, 2004.

[8] T. Y. Yeh and G. Reinman, "Fast and fair: data-stream quality of service," *Proc. of CASES*, pp.237-248, 2005.

[9] H. Kim, S. Youn, and J. Kim, "A Leakage-Aware Cache Sharing Technique for Low-Power Chip Multi-processors (CMPs) with Private L2 Caches," *Proc. of MEDEA*, pp.30-37, 2008.

[10] H. Kim and J. Kim, "A Leakage-Aware L2 Cache Management Technique for Producer-Consumer Sharing in Low-Power Chip Multiprocessors," *Proc. of COOL Chips XII*, pp.437-450, 2009.

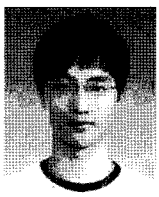
[11] J. Park and L. Choi, "A Preliminary Study on a Cache Coherence Protocol for Multi-Core Processors with Ring Interconnects," Journal of KIISE : Computing Practices and Letters, vol.14, no.8, pp. 768-772, 2008.

[12] SPEC CPU2000 benchmark, http://www.spec.org/cpu2000/.

[13] D. Kim, S. Ha, and R. Gupta, "CATS: cycle accurate transaction-driven simulation with multiple processor simulators," *Proc. of DATE*, pp.749-754, 2007.

[14] D. Tarian, S. Thoziyoor, and N. Jouppi. "CACTI: an integrated cache access time, cycle time, area, leakage, and dynamic power model," http://www.hpl.hp.com/personal/Norman_Jouppi/cacti4.html

[15] H. Zhou, M. C. Toburen, E. Rotenberg, and T. M. Conte. "Adaptive mode control: a static-power-efficient cache design, In ACM Trans. on Embedded Computing Systems," vol.2, no.3, pp.347-372, 2003.



강 희 준

2006년 서울대학교 컴퓨터공학부 학사 2008년 서울대학교 전기컴퓨터공학부 석사. 2008년~현재 LG전자 MC연구소 재직. 관심분야는 임베디드 소프트웨어 최적화, 모바일 소프트웨어 플랫폼, 소프트웨어 아키텍처

김 현 희

2004년 중앙대학교 컴퓨터공학부 학사 2006년 서울대학교 전기컴퓨터 공학부 석사. 2006년~현재 서울대학교 전기,컴퓨터 공학부 박사과정. 관심분야는 칩 멀티 프로세서 아키텍처, 저전력 시스템, 임베디드 소프트웨어

김 지 홍

1986년 서울대학교 계산통계학과 학사 1988년 University of Washington 컴퓨터과학과 석사. 1995년 University of Washington 컴퓨터과학 및 공학과 박사 1995년~1997년 미국 Texas Instruments 선임연구원. 1997년~현재 서울대학교 전기·컴퓨터공학부 교수. 관심분야는 임베디드 소프트웨어, 저전력 시스템, 멀티미디어 시스템, 컴퓨터 구조