Journal of The Korea Society of Computer and Information
Vol. 18, No. 11, November 2013
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2013.18.11.001

재구성 가능한 라스트 레벨 캐쉬 구조를 위한 코어 인지 캐쉬 교체 기법

손 동 오*, 최 홍 준*, 김 종 면**, 김 철 홍*

Core-aware Cache Replacement Policy for Reconfigurable Last Level Cache

Dong-Oh Son*, Hong-Jun Choi*, Jong-Myon Kim**, Cheol-Hong Kim*

요 약

멀티코어 프로세서에서 라스트 레벨 캐쉬는 코어와 메모리의 속도 차이를 줄여주는 역할을 하는 중요한 하드웨어 자원이다. 때문에 라스트 레벨 캐쉬의 효율적인 관리는 프로세서의 성능에 큰 영향을 미친다. 라스트 레벨 캐쉬를 구성하는 공유/비공유 캐쉬는 코어들이 공유하는 데이터와 각 코어의 독립된 데이터를 각각 적재한다. 최근 많은 연구를 통해 라스트 레벨 캐쉬 관리기법이 연구되었지만 주로 공유 캐쉬에 대한 연구만 이뤄지고 있으며 라스트 레벨 캐쉬의 비공유 캐쉬에 대한 연구는 아직 미약하다. 라스트 레벨 캐쉬의 비공유 캐쉬는 각 코어에 동일한 영역이 할당되기 때문에 코어별 작업량이 다를 경우 캐쉬 관리가 효과적이지 않다. 본 논문에서는 라스트 레벨 캐쉬 중 비공유 캐쉬의 효율적인 관리를 위해 코어 인지 캐쉬 교체 기법을 제안한다. 제안된 코어 인지 캐쉬 교체 기법은 비공유 캐쉬를 동적으로 재구성함으로써, 라스트 레벨 캐쉬의 적중률을 향상시킨다. 또한, 우리는 캐쉬 교체 기법의 성능 향상을 위해 2비트 포화 카운터를 적용하였다. 실험 결과 기존의 교체 기법과 비교하여 9.23%의 적중률 향상과 12.85%의 라스트 레벨 캐쉬 접근 시간 감소의 효과가 있었다.

▸ Keywords : 멀티코어 프로세서, 캐쉬 분할기법, 라스트 레벨 캐쉬(LLC), 캐쉬 교체방식

Abstract

In multi-core processors, Last Level Cache(LLC) can reduce the speed gap between the memory and the core. For this reason, LLC has big impact on the performance of processors. LLC is composed of shared cache and private cache. In computer architecture community, most

•제1저자 : 손동오 •교신저자 : 김철홍
•투고일 : 2013. 6. 4, 심사일 : 2013. 8. 19, 게재확정일 : 2013. 9. 12.
* 전남대학교 전자컴퓨터공학부(School of Electronics and Computer Engineering, Chonnam National University)
** 울산대학교 컴퓨터정보통신공학부(School of Computer Engineering and Information Technology, University of Ulsan)
※ 이 논문은 2012년도 전남대학교 학술연구비 지원에 의하여 연구되었음
www.kci.go.kr

researchers have mainly focused on the management techniques for shared cache, while management techniques for private cache have not been widely researched. In conventional private LLC, memory is statically assigned to each core, resulting in serious performance degradation when the workloads are not fairly distributed. To overcome this problem, this paper proposes the replacement policy for managing private cache of LLC efficiently. As proposed core-aware cache replacement policy can reconfigure LLC dynamically, hit rate of LLC is increases drastically. Moreover, proposed policy uses 2-bit saturating counters to improve the performance. According to our simulation results, the proposed method can improve hit rates by 9.23% and reduce the access time by 12.85% compared to the conventional method.

▸Keywords: Multi-core processor, Cache Partitioning, Last Level Cache(LLC), Cache replacement

I. 서 론

공정기술의 발달에 힘입어 마이크로프로세서의 동작 주파수는 지속적으로 향상되고 있다. 동작 주파수의 증가는 마이크로프로세서의 성능 향상에 큰 역할을 하였지만 이에 따른 전력 소모량 증가와 높은 발열이 발생하는 문제의 원인이 되었다. 따라서 동작 주파수의 증가는 전력 소모량과 높은 발열에 의해서 제한되었다[1]. 이에 따라 주파수 증가의 한계를 해결하기 위해, 한 개의 코어를 사용하는 싱글코어 프로세서에서 다수의 코어를 단일 칩에 집적시키는 멀티코어 프로세서가 제안되었다[2-3]. 멀티코어 프로세서는 다수의 코어를 사용하기 때문에 싱글코어 프로세서와 비교하여 상대적으로 낮은 동작 주파수에서 동작한다. 따라서 기존의 싱글코어 프로세서의 전력 소모량과 발열 문제를 해결할 수 있는 장점이 있다. 최신의 마이크로프로세서는 여러 개의 코어들을 활용하는 멀티코어 프로세서로 구성되어 있다. 다수의 코어들을 하나의 칩에 적재하는 멀티코어 프로세서는 듀얼 코어, 트리플 코어, 쿼드 코어, 헥사 코어 프로세서 등 코어의 개수에 따라 구분되기도 한다.

멀티코어 프로세서는 메모리, 인터커넥션, 라스트 레벨 캐쉬 등의 자원을 여러 코어들이 공유한다. 이러한 공유 자원 중 라스트 레벨 캐쉬는 멀티코어 프로세서의 성능에 큰 영향을 미치는 중요한 공유 자원 중 하나이다. 라스트 레벨 캐쉬는 코어와 메인 메모리의 속도 차이를 줄여주는 역할을 하기 때문에 라스트 레벨 캐쉬의 관리는 시스템 성능 결정에 있어서 매우 중요하다. 또한, 마이크로프로세서의 속도가 증가하고 이에 따라 메모리 지연시간에 대한 중요성이 높아지면서 캐쉬의 설계와 효율적인 관리는 점차 중요해지고 있다[4]. 본 논문에서는 고성능 멀티코어 프로세서에 적합한 라스트 레벨 캐쉬 관리 기법을 제안하고자 한다. 라스트 레벨 캐쉬는 공유/비공유 캐쉬로 구성되어 있다. 본 논문에서는 라스트 레벨 캐쉬의 비공유 캐쉬에 연구의 초점을 맞췄다. 비공유 캐쉬는 각 코어에 동일한영역이 할당되어 있으나 각 코어별 작업량에 따라 비공유 캐쉬의 활용률은 효율적이지 않다. 코어별 작업량이 다르기 때문에 특정 코어에서 대량의 미스가 발생하여도 다른 코어에서는 사용되지 않는 캐쉬 블록이 있을 수 있다. 따라서 본 논문에서는 라스트 레벨 캐쉬 중 비공유 캐쉬의 효율적인 관리를 위해서 코어 인지 캐쉬 교체 기법을 제안한다. 제안된 기법은 캐쉬 활용률을 높이기 위해서 비공유 캐쉬를 하나의 영역으로 통합하고, 캐쉬 미스가 발생할 때 교체 기법의 범위를 코어 단위로 구분지어 캐쉬 블록을 교체한다. 제안된 기법의 평가를 위해 기존의 라스트 레벨 캐쉬 교체 기법과 캐쉬 분할 기법들과 함께 다양한 크기의 작업량과 시스템 환경을 활용하였다.

이하 본 논문의 구성은 다음과 같다. 2장에서는 연구배경 및 관련연구에 대해서 기술하고, 3장에서는 기존의 캐쉬 교체 기법과 제안하는 캐쉬 교체 기법에 대해서 자세히 설명한다. 4장에서는 실험환경과 실험결과를 상세히 설명하며, 마지막으로 5장에서 결론을 맺는다.

www.kci.go.kr

II. 연구배경 및 관련연구

1. 목표 대상 프로세서 구조

그림 1은 본 논문에서 대상으로 삼는 멀티코어 프로세서 구조에 대한 그림이다. 대상 멀티코어 프로세서 구조는 2개의 코어를 적재하고 있으며 비공유 캐쉬인 Level1, Level2 캐쉬로 구성되어 있다. 또한, 라스트 레벨 캐쉬는 상위 레벨 캐쉬와는 다르게 공유/비공유 캐쉬로 구성되어 있다. 본 논문에서 사용하는 라스트 레벨 캐쉬의 구조는 기존의 연구에서 사용하는 공유/비공유 캐쉬 구조처럼 각 코어와 데이터를 공유하는 공유 캐쉬와 특정 코어에만 데이터를 제공하는 비공유 캐쉬는 각 코어에 특정 영역으로 할당되어 있다[5-6]. 공유 캐쉬는 각 코어 간 데이터를 공유하여 데이터를 효과적으로 활용할 수 있는 장점이 있다. 하지만 공유 캐쉬는 각 코어에서 데이터 읽기/쓰기 작업으로 인해 캐쉬 일관성을 유지해야 한다. 반면, 비공유 캐쉬는 각 코어에 할당되어 지정된 코어의 데이터만을 적재하기 때문에 캐쉬 일관성을 고려하지 않아도 되지만, 각 코어별 사용률이 다르기 때문에 캐쉬 공간의 효율성이 떨어질 수 있다. 공유 캐쉬를 위한 캐쉬 일관성 연구는 많은 연구가 이뤄지고 있고 많은 논문이 발표 되었다[7-9]. 반면, 라스트 레벨 캐쉬의 비공유 캐쉬를 위한 연구는 아직까지 많은 연구가 이뤄지지 않고 있다. 따라서 본 논문에서는 라스트 레벨 캐쉬의 비공유 캐쉬에 초점을 맞춰 비공유 캐쉬의 성능향상을 위한 연구를 하고자 한다.

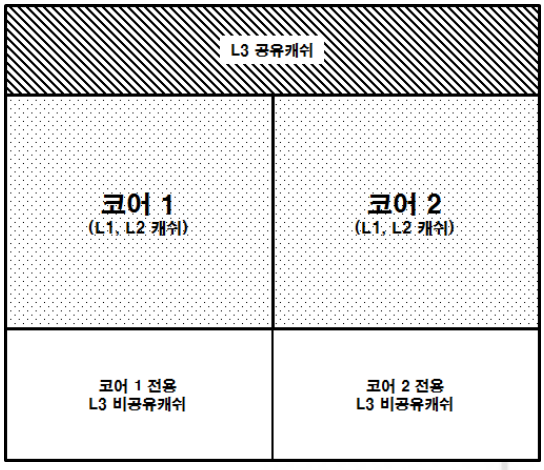

그림 1. 대상 멀티코어 프로세서 구조
Fig. 1. Multi-core processor architecture

2. 라스트 레벨 캐쉬 관리 기법

멀티코어 프로세서 내부 라스트 레벨 캐쉬 관리를 위해서는 다양한 기법들이 제안되었다[10-18]. 현재까지 제안된 기법은 크게 동적 캐쉬 분할 기법과 promotion 기반 캐쉬 관리 기법의 두 가지 그룹으로 분류된다[19].

동적 캐쉬 분할 기법은 처리량, 공정성, 대역폭 감소 등과 같은 정보를 활용하여 애플리케이션 단위로 캐쉬 영역을 할당하는 기법이다. 각 애플리케이션은 특정 영역을 할당받음으로써 각 애플리케이션 간의 간섭이 없어지기 때문에 높은 캐쉬 적중률을 보인다. 기존 연구에서 제안된 유틸리티 기반 캐쉬 분할 기법은 캐쉬의 히트 수를 극대화하는 기법으로 각 캐쉬의 웨이의 히트 수를 카운터를 이용하여 측정한다. 측정된 히트 카운터는 알고리즘에 의해 반복 수행되어 최적의 캐쉬 히트 수를 판단하고 이를 통해 캐쉬 영역을 할당하게 된다[14]. 또 다른 연구에서는 스트리밍 애플리케이션과 같이 재사용 되지 않는 캐쉬 블록을 모니터링하여 특별한 캐쉬 영역에 할당하여 캐쉬 적중률을 향상시킨다[17].

Promotion 기반 캐쉬 관리 기법은 캐쉬 영역 분할을 하지 않는다. Promotion 기반 캐쉬 관리 기법은 MRU 위치에 삽입되는 캐쉬 블록을 LRU 또는 LRU 주변에 삽입하여 캐쉬 블록의 교체를 빠르게 한다. 교체 전 캐쉬 히트가 발생하면 히트된 블록은 MRU 위치로 이동하게 된다. 이 같은 기법은 재사용 되지 않는 캐쉬 블록을 빠른 시간에 교체시키고 재사용되는 블록은 MRU 위치로 이동하여 오랜 시간 동안 캐쉬 내에 유지시킨다. 이를 통하여 스트리밍 애플리케이션에서 요구되는 블록들과 같이 재사용 되지 않는 캐쉬 블록들은 빠른 시간 내에 교체하고 재사용 되는 캐쉬 블록들은 캐쉬 내에 오랜 시간 유지되어 프로세서의 성능을 향상시킬 수 있다[12][16].

III. 라스트 레벨 캐쉬를 위한 캐쉬 교체방식

본 논문에서는 멀티코어 프로세서의 라스트 레벨 캐쉬의 성능을 향상시키기 위해 다양한 캐쉬 교체 기법에 대해서 실험을 수행한다. 다음은 다양한 캐쉬 교체 기법에 대한 설명이다.

1. 고정 영역으로 코어에 할당된 비공유 캐쉬

그림 2는 고정 영역으로 코어에 할당된 비공유 캐쉬에 대한 그림이다. 본 논문에서는 공유 캐쉬에 대한 캐쉬 영역은

www.kci.go.kr

제외하고 오직 비공유 캐쉬 영역만을 실험 대상으로 삼는다. 그림을 보면 라스트 레벨 캐쉬가 2개의 코어의 영역으로 분할되었음을 알 수 있다. 16-way 세트연관사상을 기준으로 코어 1 8-way, 코어 2 8-way로 분할되어 있다. 라스트 레벨 캐쉬 영역을 각 코어의 비율에 맞게 분배하는 구조는 비공유 캐쉬를 포함하는 멀티코어 프로세서 구조에서 일반적으로 볼 수 있는 구조이다. 따라서 본 논문에서는 고정 영역으로 코어에 할당된 비공유 캐쉬를 성능평가의 대상으로 선택하였다. 고정 영역으로 코어에 할당된 비공유 캐쉬는 일반적인 비공유 캐쉬와 마찬가지로 오직 연결된 코어의 데이터만을 캐쉬 블록에 적재한다. 따라서 비공유 캐쉬인 L1, L2 캐쉬와 같은 원리로 동작하게 된다. 할당된 영역에 데이터를 적재하고 미스가 발생하면 시스템의 캐쉬 교체정책(FIFO, LRU, Random, etc.)에 따라 캐쉬 블록을 교체하게 된다. 이처럼 캐쉬 영역이 고정되면 캐쉬 크기에 따라 성능의 영향을 많이 받게 된다. 또한, 작업량이 한쪽 코어에만 집중되면 한쪽의 캐쉬에서 많은 블록 교체가 발생하고 다른 캐쉬의 공간을 활용할 수 없기 때문에 비효율적이다.

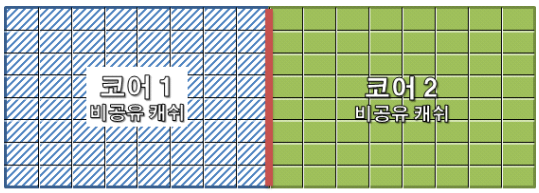

그림 2.고정 영역으로 코어에 할당된 비공유 캐쉬
Fig. 2. Conventional private cache

2. 동적 캐쉬 분할

그림 3은 동적 캐쉬 분할에 대한 그림이다. 동적 캐쉬 분할은 코어 1과 코어 2의 각 작업량에 따라서 비공유 캐쉬의 영역이 동적으로 변경되는 기법이다. 예를 들어 코어 2의 작업량이 증가함에 따라 코어 2의 비공유 캐쉬의 미스가 증가하게 되면 코어 2에 할당된 비공유 캐쉬의 영역을 증가시키고 코어 1에 할당된 비공유 캐쉬의 영역을 감소시킨다. 이러한 방법은 정적 캐쉬 분할과 비교하여 작업량 변화에 따라 캐쉬 영역이 효과적으로 변경되기 때문에 시스템 성능이 향상될 수 있다. 하지만 비공유 캐쉬 영역이 변경될 때마다 캐쉬 영역의 경계에 있는 부분은 캐쉬 교체 순서에 상관없이 바로 교체 되는 단점이 있다.

그림 3. 동적 캐쉬 분할 기법
Fig. 3. Dynamic cache partitioning replacement

3. 라인별 동적 캐쉬 분할

라인별 동적 캐쉬 분할은 동적 캐쉬 분할과 비슷한 특성이 있다. 그림 4는 라인별 동적 캐쉬 분할 기법에 대한 그림이다. 이 기법은 동적 캐쉬 분할 기법에서 발전된 기법으로 비공유 캐쉬 영역의 변경 범위를 캐쉬의 세트 수준이 아닌 각 라인 단위로 캐쉬 영역 변경 범위를 축소 시켰다. 기존의 동적 캐쉬 분할 기법은 코어 1과 코어 2의 미스 숫자에 따라 캐쉬 영역이 변경되는데 변경되는 캐쉬 영역은 캐쉬 라인의 숫자와 같다. 변경되는 캐쉬 영역은 시스템의 교체 알고리즘에 상관없이 교체되기 때문에 캐쉬 영역을 변경하면 많은 손해가 발생하게 된다. 따라서 이러한 문제점을 해결하기 위해 교체 영역의 범위를 각 라인 단위로 축소시킨 라인별 동적 캐쉬 분할 기법을 적용하였다. 라인별 동적 캐쉬 분할 기법을 적용하면 특정 코어에서 다수의 미스가 발생하면 각 라인별로 비공유 캐쉬 영역이 변경되기 때문에 불필요한 교체 명령 없이 캐쉬 영역을 활용할 수 있다.

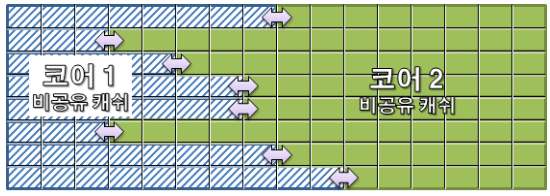

그림 4. 라인별 동적 캐쉬 분할 기법
Fig. 4. Dynamic line-based cache partitioning replacement

그림 5는 라인별 동적 캐쉬 분할의 예이다. 라인별 동적 캐쉬 분할 기법은 각 코어의 작업량에 따라 라인별로 비공유 캐쉬 영역이 할당되어 높은 성능을 보이지만 이 기법 역시 코어의 고정 영역을 변경할 때 변경되는 부분은 캐쉬 교체정책(LRU)의 순서에 상관없이 교체된다. 아래 그림처럼 8:8의 캐쉬 영역을 가진 비공유 캐쉬 라인에서 코어 2의 미스가 많이 발생하여 코어 2의 비공유 캐쉬에서 더 많은 영역을 요청하면 라인별 동적 캐쉬 분할 기법은 코어 2의 비공유 캐쉬 영

www.kci.go.kr

역을 확장하게 된다. 코어 2의 비공유 캐쉬 영역 확장에 따라 코어 1의 캐쉬 블록이 코어 2의 비공유 캐쉬로 교체되게 되는데, 교체되는 캐쉬 블록은 코어 2와 경계에 있는 코어 1의 캐쉬 블록 5번이 교체 된다. 코어 1의 캐쉬 블록 5번은 코어 1의 캐쉬에서 가장 오래된 블록 8번보다 먼저 교체됨으로써 시스템의 교체정책(LRU)를 통한 성능 향상을 기대할 수 없다. 이처럼 시스템의 교체정책(LRU)를 통한 교체가 아닌 무작위적인 캐쉬 교체가 발생한다면 성능 저하가 발생할 수 있다. 이와 같은 문제를 해결하기 위해 본 논문에서는 비공유 캐쉬의 캐쉬 블록을 교체할 때 가장 오래된 블록을 우선적으로 교체(LRU와 동일)하는 코어 인지 캐쉬 교체 기법을 제안한다.

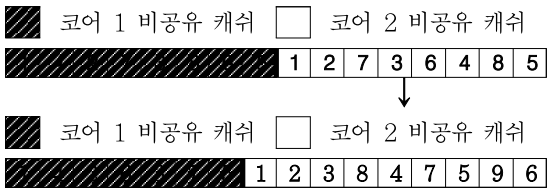

그림 5. 라인별 동적 캐쉬 분할의 예
Fig. 5. An example of dynamic line-based cache partitioning replacement

4. 코어 인지 캐쉬 교체

다음은 코어 인지 캐쉬 교체 기법에 대한 설명이다. 본 논문에서는 라스트 레벨 캐쉬에서 비공유 캐쉬의 효과적인 교체 기법 적용을 위해 코어 인지 캐쉬 교체 기법을 제안하였다.

4.1 코어 인지 캐쉬 교체 기법

그림 6은 코어 인지 캐쉬 교체 기법에 대한 그림이다. 코어 인지 캐쉬 교체는 본 논문에서 제안하는 기법으로 원리는 다음과 같다. 제안된 기법은 각 코어의 비공유 캐쉬의 영역이 분할되어 있지 않다. 이 기법에 사용되는 라스트 레벨 캐쉬의 비공유 캐쉬는 코어에 상관없이 비공유 캐쉬 영역을 같이 사용한다. 따라서 고정 영역으로 할당된 비공유 캐쉬에 비해서 2배의 캐쉬 공간을 활용하는 장점이 있다. 또한, 특정 코어에서 작업량 증가로 인해 많은 미스가 발생하면 캐쉬 교체정책(LRU)의 범위를 하나의 라인으로 하지 않고 미스가 발생하지 않는 코어의 캐쉬 블록만을 교체 대상으로 삼는다. 예를 들어 코어 2의 작업량 증가에 따라 코어 2 캐쉬 블록의 미스가 많이 발생하게 되면 아래 그림처럼 코어 1의 캐쉬 블록 영역(상향 대각선)만을 교체 대상으로 선정하여 선정된 코어의 영역을 교체 대상으로 삼는다. 이와 같은 기법은 특정 코어에서 미스가 증가하게 될 때 해당 코어의 캐쉬 블록은 유지한 채 다른 코어의 캐쉬 블록만을 교체하여 캐쉬 영역을 확장시켜 주는 기법이다.

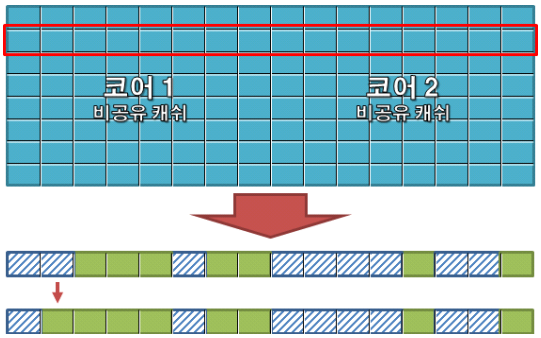

그림 6. 코어 인지 캐쉬 분할 기법
Fig. 6. Core-aware cache replacement

그림 7은 코어 인지 캐쉬 교체 기법의 예이다. 코어 1의 캐쉬 블록과 코어 2의 캐쉬 블록이 코어의 비공유 캐쉬 영역의 구분 없이 각 블록에 적재되어 있다. 각 블록의 숫자는 시스템의 교체정책(LRU)에 의한 삽입 순서를 나타낸다(1번이 MRU, 16번이 LRU). 이때 코어 2의 작업량이 증가하여 캐쉬의 미스가 증가하게 되면 코어 2를 위한 캐쉬 블록이 추가로 필요하게 된다. 제안된 기법은 아래 그림처럼 캐쉬 블록의 교체 대상을 16개의 캐쉬 블록이 아닌 오직 코어 1의 캐쉬 블록으로만 한정 짓는다. 따라서 미스가 발생할 때 시스템의 교체정책(LRU)에 따라 교체되어야 할 코어 2의 캐쉬 블록인 16번은 교체되지 않는다. 교체대상을 상대적으로 캐쉬 미스가 적은 코어 1의 캐쉬로 한정 짓기 때문에 코어 1의 캐쉬 블록 중 가장 오래된 캐쉬 블록인 14번 블록이 교체된다. 이처럼 코어 인지 캐쉬 교체 기법을 사용함으로써 라인별 동적 캐쉬 분할 기법의 문제가 되던 캐쉬 교체 순서 문제를 해결할 수 있다.

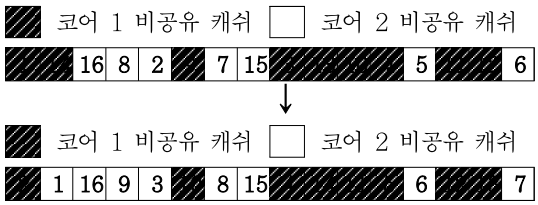

그림 7. 코어 인지 캐쉬 교체 기법의 예
Fig. 7. An example of core-aware cache replacement

4.2 코어 인지 캐쉬 교체 기법의 교체 주기 설정

코어 인지 캐쉬 교체 기법을 효과적으로 적용하기 위해서

www.kci.go.kr

는 기법의 교체 주기의 설정이 중요하다. 기법의 교체 주기의 설정에 따라 캐쉬 교체 빈도가 결정되기 때문이다. 기법의 교체 주기가 너무 길면 캐쉬 교체 기법의 효과가 미미할 수 있으며, 캐쉬 교체 주기가 너무 짧으면 불필요한 캐쉬 블록 교체가 일어날 수 있기 때문이다. 따라서 본 논문에서는 코어 인지 캐쉬 교체 기법의 효과적인 캐쉬 교체를 위해 코어별 미스의 숫자에 따라 기법의 교체 주기를 설정하여 실험해 보았다. 이 실험을 위해서 지역성이 낮고 작은 크기의 작업량을 시스템에 할당하였다(작업량에 대한 설명은 4장에서 자세히 설명한다). 낮은 지역성과 작은 크기의 작업량은 캐쉬 교체 기법에 많은 영향을 받기 때문에 기법의 교체 주기 변경에 따른 라스트 레벨 캐쉬의 적중률 차이를 비교하기 위해서 할당하였다.

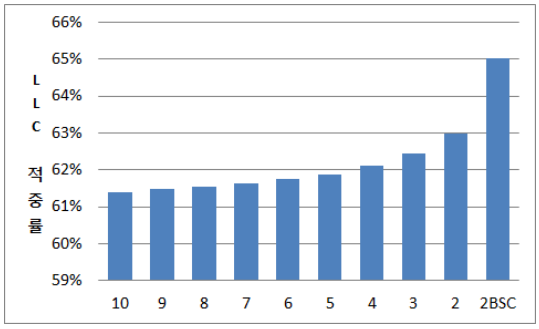

그림 8. 교체 주기(동일 코어의 미스 횟수)에 따른 라스트 레벨 캐쉬 적중률
Fig. 8. LLC hit rates according to replacement period

그림 8은 기법의 교체 주기를 동일 코어에서 발생하는 추가적인 미스 횟수로 설정하였을 때의 라스트 레벨 캐쉬의 적중률이다. 예를 들어 미스 횟수가 10으로 설정될 때 코어 1에서 코어 2보다 미스가 10번 더 발생하게 되면 코어 인지 캐쉬 교체 기법이 코어 2의 캐쉬 블록에 적용되어 코어 2의 캐쉬 블록이 교체된다. 각 코어별로 미스가 발생하면 카운터에 의해 미스 횟수가 증가하거나 감소한다. 코어 인지 캐쉬 교체 기법이 적용되면 미스 횟수 카운트는 초기화되고 다시 코어별 미스 횟수를 카운트하게 된다. 실험결과 미스 횟수에 의한 교체 주기가 짧아질수록 라스트 레벨 캐쉬의 적중률이 향상되는 것을 볼 수 있다.

본 논문에서는 기법의 교체 주기의 성능 향상을 위해서 특정 코어에서 발생하는 추가적인 2번의 미스 횟수보다 더 짧은 교체 주기를 사용하는 2비트 포화 카운터를 적용하였다. 2비트 포화 카운터는 연속적으로 2번의 다른 값이 발생하기 전까지는 현재 값을 유지한다. 예를 들어 코어 1에서 연속적으로 2번의 미스가 발생하면 코어 2에서 연속적으로 2번 이상의 미스가 발생되기 전까지 캐쉬 교체 대상은 코어 2의 캐쉬 블록만을 교체 대상으로 설정한다. 따라서 연속적인 캐쉬 미스가 없으면 항상 교체 기법이 적용되기 때문에 가장 짧은 교체 주기가 설정된다. 교체 주기의 값이 2일 때는 62.98%의 적중률을 보이지만 2비트 포화 카운터를 적용하면 65.01%로 2.03% 적중률 향상을 보임으로써 교체 주기가 짧을수록 캐쉬 적중률 향상에 도움이 됨을 알 수 있다.

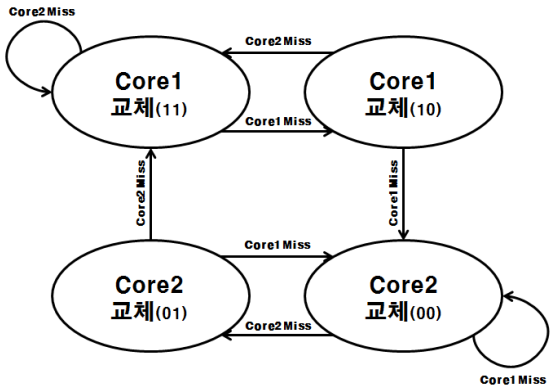

그림 9. 코어 인지 캐쉬 교체 기법을 위한 2비트 포화 카운터
Fig. 9. 2-bit saturating counter for core-aware cache replacement

그림 9는 코어 인지 캐쉬 교체 기법에 적용된 2비트 포화 카운터에 대한 그림이다. 작동원리는 각 코어의 미스가 연속적으로 2번 이상으로 발생할 때 캐쉬 교체 대상을 비공유 캐쉬 전체에서 다른 코어의 비공유 캐쉬로 한정한다. 2비트 포화 카운터의 위치가 (00)비트일 때, 코어 2에서 캐쉬 미스가 연속으로 2번 발생하게 되면 (01)비트를 거쳐 (11)비트로 변경된다. 이 경우 캐쉬 교체 대상은 코어 1의 비공유 캐쉬만을 대상으로 교체 정책이 적용된다. 또한, 코어 1의 미스가 발생하고 코어 2의 미스가 발생하게 되면 카운터의 위치는 (10)비트를 거쳐 (11)비트로 되돌아온다. 이처럼 코어 인지 캐쉬 교체 기법을 위한 2비트 포화 카운터는 연속적으로 특정 코어의 미스가 2번 발생 될 때까지 비공유 캐쉬의 교체 대상은 카운터가 위치한 코어의 비공유 캐쉬만을 교체 대상으로 삼는다.

4.3 오버헤드

코어 인지 캐쉬 교체 기법은 캐쉬 블록에 적재되는 코어를 구별하기 위해서 각 캐쉬 블록에 추가적인 비트가 필요하며 제안된 기법의 교체 주기 설정을 위해 2비트 포화 카운터가

www.kci.go.kr

필요하다. 코어 인지 캐쉬 교체 기법에 필요한 추가 구성은 아래와 같다.

1) 캐쉬 블록당 1비트의 정보비트 : 캐쉬 블록에 적재되는 데이터의 식별을 위해 캐쉬의 각 블록에 추가적으로 1비트의 정보비트가 필요하다. 코어 1의 데이터 블록은 0, 코어 2의 데이터 블록은 1로 표시하여 각 코어의 데이터를 구별한다.
2) 라인당 2비트의 포화 카운터 : 캐쉬의 각 라인에서 교체 주기 설정을 위해서 2비트의 포화 카운터가 필요하다.

$$\frac{16bit(1)+2bit(2)}{64Byte(\text{캐쉬 블록 크기})*16\,Way}=0.0017\%$$

라스트 레벨 캐쉬는 하나의 블록당 64Byte의 큰 블록을 사용함으로 제안된 기법에 사용되는 추가적인 비트는 거의 무시할 수준이다. 추가적인 비트로 인해 발생하는 하드웨어 오버헤드는 8Mbyte의 라스트 레벨 캐쉬를 기준으로 0.0017%의 아주 작은 오버헤드가 발생한다. 이것은 실제 시스템에 영향을 미치지 못할 정도로 미미한 수준이다.

IV. 실험

본 논문의 실험에서 사용된 시뮬레이터와 작업량의 정보는 다음과 같다.

1. 실험환경

표 1. 시뮬레이터 구성 변수
Table 1. Simulation parameters

실험인자	값
Set-Associative Mapping	16-way
Set	8192 Sets
Cache Line	64Byte
Size	8MB

멀티코어 프로세서에서 비공유 캐쉬의 다양한 교체 기법의 성능을 분석하기 위해 라스트 레벨 캐쉬 시뮬레이터를 구현하였다. 시뮬레이터 구현은 C언어로 구현하였고 구현된 시뮬레이터의 구성 변수는 표 1과 같다. 공유 캐쉬는 캐쉬 일관성을 고려해야 하기 때문에 본 논문에서는 캐쉬 일관성이 적용되지 않는 각 코어의 비공유 캐쉬만을 대상으로 한정하여 실험하였다. 캐쉬 초기화를 위해 100만번의 캐쉬 접근을 수행하였고 실험은 1,000만번의 캐쉬 접근이 이뤄진다.

시뮬레이터의 동작은 다음과 같다. (1) 캐쉬 초기화 작업을 통해 무작위 데이터를 캐쉬에 삽입한다. (2) 시뮬레이터 동작은 실험에서 설정된 지역성의 비율에 따라 지역성과 비지역성을 먼저 판단한다. (3) 지역성이 발생하는 데이터는 최근에 히트한 데이터 블록(시간적 지역성) 또는 최근에 히트한 데이터 블록의 주변 데이터(공간적 지역성)를 사용하여 캐쉬 히트/미스 여부를 판별한다. (4) 비지역성 데이터는 무작위로 데이터를 선택한 후 캐쉬 히트/미스 여부를 판단한다. (5) 캐쉬 미스가 발생하면 각 교체 기법에 따라 캐쉬 교체가 적용된다.

2. 실험결과

실험결과는 고정 영역으로 코어에 할당된 비공유 캐쉬의 교체 기법을 비교 대상으로 적중률 차이를 비교하였다.

2.1 작업량의 구성

표 2. 지역성과 크기에 따른 작업량 분류
Table 2. Classified applications according to locality and workload size

지역성	High	High	High	Low	Low	Low
크기	Big	Middle	Small	Big	Middle	Small

제안된 기법의 평가를 위해 다양한 종류의 작업량을 선정하여 실험하였다. 실험에 사용된 작업량의 종류는 총 6가지를 사용하였다. 먼저 높은 지역성(시간적 지역성 : 40%, 공간적 지역성 40%)과 낮은 지역성(시간적 지역성 : 10%, 공간적 지역성 10%)으로 작업량을 구분하였고 세부적으로 큰 크기의 작업량(512MB)과 중간 크기의 작업량(256MB), 그리고 작은 크기의 작업량(128MB)으로 분류하였다. 작업량의 구성은 표 2와 같다.

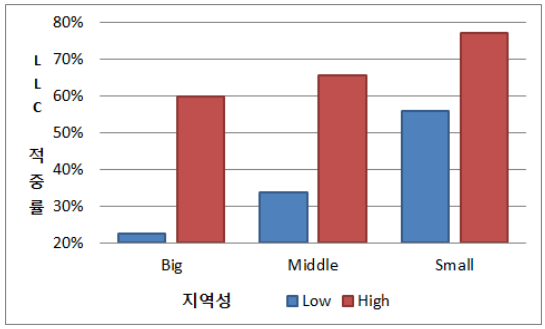

그림 10. 작업량 분류에 따른 라스트 레벨 캐쉬 적중률
Fig. 10. LLC hit rates according to workload characteristics

www.kci.go.kr

각 작업량은 시뮬레이터에서 1,000만번 캐쉬 접근이 발생한다. 작업량의 분류에 따른 라스트 레벨 캐쉬 적중률은 그림 10과 같다. 다양한 데이터를 얻기 위해 총 6개의 작업량을 구분하여 실험하였으며 각 작업량별로 라스트 레벨 캐쉬의 적중률은 다양한 값을 보인다. 실제 라스트 레벨 캐쉬는 벤치마크에 따라 0.3%에서 100%까지 다양한 적중률을 보인다[20-22]. 따라서 본 논문에서는 다양한 실험을 위해 위와 같이 22%에서 77%까지의 적중률을 보이는 6개의 작업량을 통해 실험하였다.

2.2 라스트 레벨 캐쉬 적중률

동적 캐쉬 분할

그림 11의 왼쪽 그래프는 동적 캐쉬 분할에 따른 라스트 레벨 캐쉬의 적중률 차이이다. 적중률 차이는 고정 영역으로 코어에 할당된 비공유 캐쉬를 비교 대상으로 하였다. 코어의 작업량에 따라 비공유 캐쉬 영역이 변화하는 동적 캐쉬 분할은 고정 영역으로 코어에 할당된 비공유 캐쉬의 교체 기법과 비교하여 적중률 향상이 높다. 이러한 이유는 기존의 교체 기법은 코어의 작업량에 상관없이 비공유 캐쉬 영역이 고정적이기 때문에 코어의 작업량에 따른 캐쉬 영역의 확보가 되지 않아 한쪽의 코어에서 많은 작업량이 발생하면 캐쉬 미스가 많이 발생하게 되기 때문이다. 반면 동적 캐쉬 분할은 코어의 작업량에 따라 캐쉬 영역을 동적으로 할당해 줌으로써 한쪽의 코어에서 많은 작업량이 발생해도 작업량에 따라 비공유 캐쉬의 영역을 확장시켜 주기 때문에 캐쉬 적중률이 향상된다. 하지만 캐쉬 영역을 변경할 때 변경되는 코어의 캐쉬 영역은 캐쉬 교체순서와 상관없이 교체되기 때문에 약간의 손실이 발생한다.

또한, 기존의 교체 기법과 비교하여 지역성이 낮은 작업량이 지역성이 높은 작업량보다 캐쉬 적중률 향상의 폭이 크다. 지역성이 낮은 작업량은 낮은 지역성으로 인해 새로운 캐쉬 블록이 자주 삽입되어 캐쉬 미스가 많이 발생한다. 캐쉬 미스가 많이 발생할수록 동적 캐쉬 분할은 캐쉬 영역을 효과적으로 관리할 수 있기 때문에 기존의 교체 기법보다 높은 캐쉬 적중률을 보인다. 마찬가지로 크기가 작은 작업량이 크기가 큰 작업량보다 캐쉬 적중률 향상 폭이 크다. 작업량의 크기가 작을수록 캐쉬 교체 기법의 영향을 많이 받기 때문에 적중률이 향상된다. 실험결과 비공유 캐쉬의 적중률은 기존의 교체 기법과 비교하여 1.12%에서 7.18%까지의 적중률 향상이 있고 지역성이 낮으면 평균 4.26%, 지역성이 높으면 평균 2.26%의 적중률 향상이 있다.

라인별 동적 캐쉬 분할

그림 11의 가운데 그래프는 라인별 동적 캐쉬 분할에 따른 라스트 레벨 캐쉬의 적중률 차이이다. 라인별 동적 캐쉬 분할은 동적 캐쉬 분할에서 캐쉬 영역 변경할 때 문제가 되던 불필요한 캐쉬 블록 교체를 해결한 기법으로 동적 캐쉬 분할과 비교하여 약간의 적중률 향상이 있다. 이것은 캐쉬 영역을 변경할 때 한꺼번에 교체되는 캐쉬 영역을 각 라인 단계로 변경함으로 불필요한 블록 교체를 방지하기 때문에 적중률이 향상된다. 나머지 실험결과는 동적 캐쉬 분할과 비슷한 패턴을 보인다. 지역성이 낮은 작업량과 크기가 작은 작업량이 기존 캐쉬 교체 기법과 비교하여 높은 적중률 향상이 있다. 실험결과 비공유 캐쉬의 적중률은 기존의 교체 기법과 비교하여 1.37%에서 7.20%까지의 적중률 향상이 있고 지역성이 낮

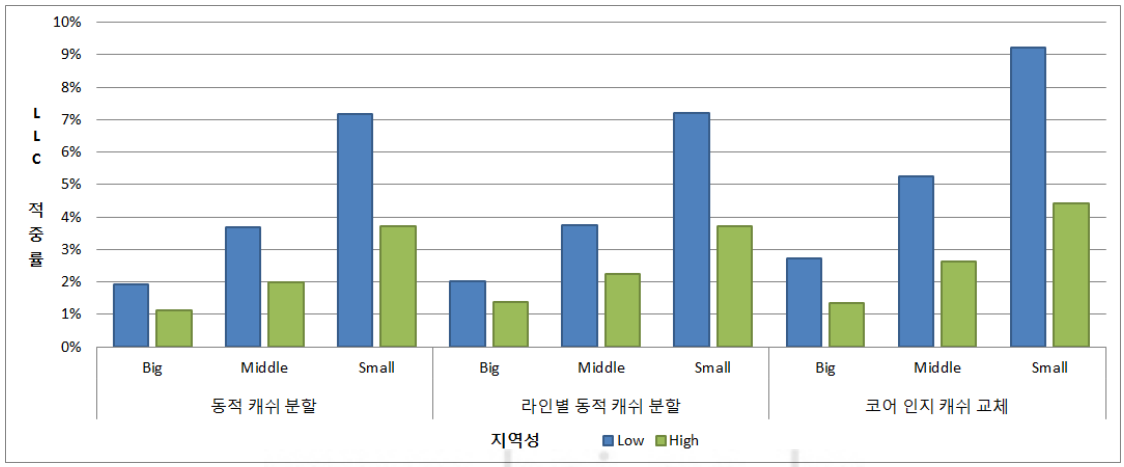

그림 11. 라스트 레벨 캐쉬 적중률
Fig. 11. LLC hit rates

www.kci.go.kr

으면 평균 4.32%, 지역성이 높으면 평균 2.44%의 적중률 향상이 있다.

코어 인지 캐쉬 교체

그림 11의 오른쪽 그래프는 본 논문에서 제안하는 코어 인지 캐쉬 교체 기법에 따른 라스트 레벨 캐쉬의 적중률 차이이다. 코어 인지 캐쉬 교체 기법은 미스가 발생하면 교체되는 캐쉬 블록을 다른 코어에서만 선택함으로써 미스가 발생한 코어의 비공유 캐쉬 영역의 확보와 함께 교체되는 캐쉬 블록에도 LRU 캐쉬 교체 정책이 적용되어 기존의 캐쉬 교체 기법과 비교하여 높은 적중률 차이를 보인다. 앞의 캐쉬 분할 기법들과 마찬가지로 지역성이 낮은 작업량과 크기가 작은 작업량이 기존 캐쉬 교체 기법과 비교하여 높은 적중률 향상이 있다. 실험결과 비공유 캐쉬의 적중률은 기존의 교체 기법과 비교하여 1.34%에서 9.23%까지의 적중률 향상이 있고 지역성이 낮으면 평균 5.73%, 지역성이 높으면 평균 2.79%의 적중률 향상이 있다.

모든 실험 결과에서 비공유 캐쉬의 영역을 고정적으로 사용하는 것보다는 동적으로 사용하는 것이 캐쉬 관리에 있어서 효과적임을 알 수 있다. 특히 제안된 코어 인지 캐쉬 교체 기법은 9.23%의 높은 적중률 향상을 볼 수 있다. 이처럼 본 논문에서 제안하는 코어 인지 캐쉬 교체 기법은 라스트 레벨 캐쉬의 적중률 향상은 메모리의 접근 횟수를 줄이는 효과가 있기 때문에 중앙처리장치와 메모리의 속도 문제를 해결하는 방법이 될 수 있다.

2.3 라스트 레벨 캐쉬 접근 시간

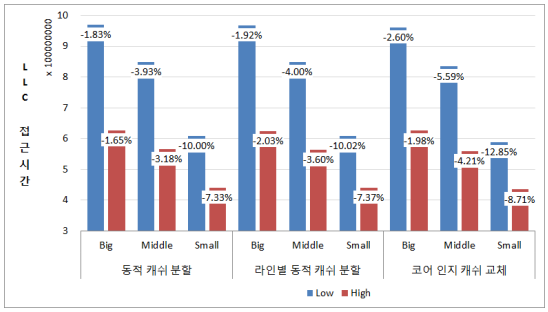

그림 12. 라스트 레벨 캐쉬 접근 시간
Fig. 12. LLC access time

그림 12는 라스트 레벨 캐쉬를 위한 교체 기법의 접근 시간이다. 고정적으로 코어에 할당되는 비공유 캐쉬를 기준으로 하여 다른 교체 기법들을 정규화하여 그래프로 나타내었다. 결과를 보면 모든 캐쉬 교체 기법에서 접근 시간 감소의 효과를 볼 수 있다. 동적 캐쉬 분할은 1.65%에서 10.00%, 라인별 동적 캐쉬 분할은 1.92%에서 10.02%의 접근 시간 감소 효과가 있다. 또한, 제안하는 코어 인지 캐쉬 교체 기법은 1.98%에서 12.85%의 아주 높은 접근 시간 감소의 효과가 있다.

V. 결론

고성능 멀티코어 프로세서에서 라스트 레벨 캐쉬는 아주 중요한 자원이다. 본 논문에서는 고성능 멀티코어 프로세서를 위한 라스트 레벨 캐쉬의 효율적인 관리를 위해서 비공유 캐쉬의 교체 기법에 대한 연구를 수행하였고 새로운 캐쉬 교체 기법을 제안하였다. 제안된 코어 인지 캐쉬 교체 기법은 캐쉬 교체 대상을 코어 단위로 구분하여 미스가 많이 발생하는 코어의 비공유 캐쉬 블록은 유지하고 다른 코어의 비공유 캐쉬 블록만을 교체 대상으로 삼는다. 이러한 교체를 위해서 2비트 포화 카운터를 사용하여 기법의 캐쉬 교체 주기를 판단하였다. 실험결과 고정 영역으로 코어에 할당된 비공유 캐쉬의 교체 기법과 비교하여 제안된 기법은 9.23%의 적중률 향상과 12.85%의 캐쉬 접근 시간 향상을 볼 수 있었다. 본 논문에서는 2개의 코어를 가진 멀티코어 프로세서 시스템 환경으로 실험하였으나 제안된 아이디어는 다수의 코어를 가진 멀티코어 프로세서에 적용할 수 있다. 향후 연구에서는 공유/비공유 캐쉬를 포함하는 라스트 레벨 캐쉬의 효과적인 캐쉬 관리 기법에 대한 연구를 진행하고자 한다.

참고문헌

[1] V. Agarwal, M. S. Hrishikesh, S. W. Keckler, and D. Burger, "Clock rate versus IPC: the end of the road for conventional microArchitectures," In Proceedings of 27th international symposium on computer architecture, pp. 248-259, Vancouver, Canada, June. 2000.

www.kci.go.kr

[2] Y. J. Kwon, C. D. Kim, S. R. Maeng, and J. H. Huh, "Virtualizing performance asymmetric multi-core systems," In Proceedings of 38th International Symposium on Computer Architecture, pp. 45-56, San Jose, USA, June. 2011.

[3] M. DeVuyst, A. Venkat, and D. M. Tullsen, "Execution migration in a heterogeneous-ISA chip multiprocessor," In Proceedings of 17th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 261-272, London, UK, Mar. 2012.

[4] M. K. Qureshi, D. Thompson, and Y. N. Patt, "The V-Way Cache : Demand-Based Associativity via Global Replacement," In Proceedings of The 32nd International Symposium on Computer Architecture, pp. 544-555, Madison, USA, June. 2005.

[5] H. Dybdahl, and P. Stenstrom, "An Adaptive Shared/Private NUCA Cache Partitioning Scheme for Chip Multiprocessors," In Proceedings of 13th International Symposium on High Performance Computer Architecture, pp. 2-12, Phoenix, USA, Feb. 2007.

[6] A. Jaleel, M. Mattina, and B. Jacob, "Last Level Cache (LLC) Performance of Data Mining Workloads On a CMP —A Case Study of Parallel Bioinformatics Workloads," In Proceedings of 12th International Symposium on High Performance Computer Architecture, pp. 88-98, Austin, USA, Feb. 2006.

[7] J. H. Kelm, D. R. Johnson, W. Tuohy, S. S. Lumetta, and S. J. Patel, "Cohesion: An Adaptive Hybrid Memory Model for Accelerators," IEEE MICRO, Vol. 31, Issue 1, pp. 42-55, Jan.-Feb. 2011.

[8] A. Meixner, and D. J. Sorin, "Error Detection via Online Checking of Cache Coherence with Token Coherence Signatures," In Proceedings of 13th International Symposium on High Performance Computer Architecture, pp. 145-156, Phoenix, USA, Feb. 2007.

[9] L. Cheng, J. B. Carter, and D. Dai, "An Adaptive Cache Coherence Protocol Optimized for Producer-Consumer Sharing," In Proceedings of 13th International Symposium on High Performance Computer Architecture, pp. 328-339, Phoenix, USA, Feb. 2007.

[10] M. Chaudhuri, "Pseudo-LIFO: the foundation of a new family of replacement policies for last-level caches," In Proceedings of 42nd Microarchitecture, pp. 401-412, New York, USA. Dec. 2009.

[11] A. Jaleel, W. Hasenplaugh, M. Qureshi, J. Sebot, S. Steely. Jr, and J. Emer, "Adaptive insertion policies for managingshared caches," In Proceedings of the 17th international conference on Parallel architectures and compilation techniques, pp. 208-219, Toronto, Canada, Oct. 2008.

[12] A. Jaleel, K. B. Theobald, S. C. Steely. Jr, and J. Emer, "Highperformance cache replacement using re-reference intervalprediction (RRIP)," In Proceedings of 32nd International Symposium on Computer Architecture, pp. 60-71, Madison, USA, June. 2010.

[13] S. Kim, D. Chandra, and D. Solihin, "Fair cache sharing and partitioning in a chip multiprocessor architecture," In Proceedings of the 13th international conference on Parallel architectures and compilation techniques, pp. 111-122, Antibes Juan-les-Pins, France, Sep. 2004.

[14] M. K. Qureshi, and Y. N. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," In Proceedings of 39th Microarchitecture, pp. 423-432, Orlando, USA, Dec. 2006.

[15] S. Srikantaiah, M. Kandemir, and Q. Wang, "Sharp control:Controlled shared cache management in chip multiprocessors," In Proceedings of 42nd Microarchitecture, pp. 517

www.kci.go.kr

-528, New York, USA, dec. 2009.

[16] Y. Xie and G. H. Loh, "PIPP: promotion/insertion pseudopartitioning of multi-core shared caches," In Proceedings of The 36th International Symposium on Computer Architecture, pp. 174-183, Austin, USA, June. 2009.

[17] Y. Xie and G. H. Loh, "Scalable shared-cache management by containing thrashing workloads," In Proceedings of High Performance Embedded Architectures and Compilers, pp. 262-276. Pisa, Italy, Jan, 2010.

[18] J. M. Kim and S. W. Chung, "Group-Based Replacement Algorithm to Reduce Cache Miss in Last Level Cache," Journal of The Korea Society of Computer and Information, Vol. 6, No. 5, pp.44-50, Oct. 2010.

[19] J. Lee, and H. Kim, "TAP: A TLP-Aware Cache Management Policy for a CPU-GPU Heterogeneous Architecture," In Proceedings of 18th International Symposium on High Performance Computer Architecture, pp. 91-102, New Orleans, USA, Feb. 2012.

[20] E. Perelman, M. Polito, J. B, J. Sampson, B. Calder, and C. Dulong, "Detecting Phases in Parallel Applications on Shared Memory Architectures," In Proceedings of International Parallel and Distributed Processing Symposium, pp. 88-88, Rhodes Island, Greece, April. 2006.

[21] Z. Zhang, Z. Zhu, and X. Zhang, "Design and Optimization of Large Size and Low Overhead Off-Chip Caches," IEEE Transactions on Computer, Vol. 53, Issue 7, pp. 843-855, July. 2004.

[22] Y. Yang, P. Xiang, M. Mantor, and H. Zhou, "CPU-Assisted GPGPU on Fused CPU-GPU Architectures," In Proceedings of 18th International Symposium on High Performance Computer Architecture, pp. 1-12, New Orleans, USA, Feb. 2012.

저 자 소 개

손 동 오

2010: 전남대학교 전자컴퓨터공학부 공학사
2012: 전남대학교 전자컴퓨터공학과 석사
현 재: 전남대학교 전자컴퓨터공학과 박사과정
관심분야: 컴퓨터구조, 임베디드시스템
Email : sdo1127@gmail.com

최 홍 준

2009: 전남대학교 전자컴퓨터공학부 공학사
2011: 전남대학교 전자컴퓨터공학과 석사
현 재: 전남대학교 전자컴퓨터공학과 박사과정
관심분야: 저전력 설계, 컴퓨터 구조
Email :chj6083@gmail.com

김 종 면

1995: 명지대학교 전기공학사
2000: University of Florida ECE 석사
2005: Georgia Institute of Technology ECE 박사
2005-2007: 삼성종합기술원 전임연구원
현 재: 울산대학교 컴퓨터정보통신공학부 교수
관심분야: 임베디드 SoC, 컴퓨터구조, 프로세서 설계, 병렬처리
Email : jmkim07@ulsan.ac.kr

www.kci.go.kr

김 철 홍

1998: 서울대학교 컴퓨터공학사
2000: 서울대학교 컴퓨터공학부 석사
2006: 서울대학교
전기컴퓨터공학부 박사
2005-2007: 삼성전자 반도체총괄
SYS.LSI사업부
책임 연구원
현 재: 전남대학교
전자컴퓨터공학부 교수
관심분야: 임베디드시스템, 컴퓨터구조, SoC 설계, 저전력 설계
Email : chkim22@chonnam.ac.kr

www.kci.go.kr