칩 멀티 프로세서의 공유 버스를 이용한 유휴 캐시 활용 기법

강석빈*, 김주환*, 곽종욱**, 장성태***, 전주식*
*서울대학교 전기컴퓨터공학부
**영남대학교 컴퓨터공학과, ***수원대학교 컴퓨터학과
e-mail : ppp9494@snu.ac.kr

Idle Cache Exploiting Techniques for Shared Bus-based Chip Multi-processors

Seok-bin Kang*, Ju-hwan Kim*, Jong Wook Kwak**, Seong Tae Jhang***, Chu-shik Jhon*
*Dept. of Electrical Engineering and Computer Science, Seoul National University
**Dept. of Computer Engineering, Yeungnam University
***Dept. of Computer Science, The University of Suwon

요 약

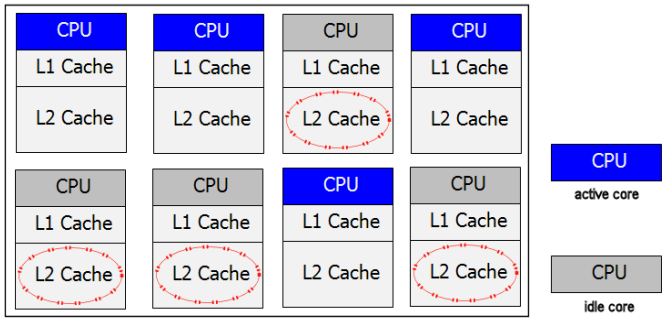
반도체 집적도의 향상과 제한된 프로세서 설계 능력으로 인한 칩 멀티 프로세서의 도입은 최근 수 년 동안 급속히 이루어졌으나, 다수의 프로세싱 코어를 효율적으로 사용하기 위한 기법은 부족한 실정이다. 칩 멀티 프로세서 상에서 실제 작업을 수행하지 않는 유휴 코어의 발생은 불가피하며, 이 때 코어가 소유한 자원들은 낭비될 수 밖에 없다. 기존의 연구들은 이렇게 낭비되는 자원 중에서 캐시의 효율적 관리를 위해 공유 캐시 형태로 캐시를 구성하였으나, 전체 캐시 관리에 따른 많은 오버헤드를 수반하였다. 본 논문에서는 이러한 유휴 캐시의 발생이 불가피함을 인지하고 그것을 칩 내 메모리 공간으로써 활용하여 칩 멀티 프로세서 전체의 성능을 향상시키는 기법을 제안한다. 이를 위해 ARM 코어 기반의 칩 멀티프로세서 시뮬레이터 환경을 구성하여 제안된 기법을 검증한다. 실험 결과 본 논문에서 소개된 기법은 4-코어 및 16 코어 기반 칩 멀티 프로세서 환경에서 각각 17%와 8%의 IPC 향상을 가져왔다.

1. 서론

전통적으로 프로세서에 관한 연구는 단일 프로세서의 성능을 극대화 시키는 방향으로 이루어 졌다. 하지만 꾸준한 칩 트랜지스터의 집적도 향상과 비교적 정체되어 있는 프로세서 설계 능력으로 인해 동일한 설계를 복제시키는 형태의 디자인 기법의 도입이 불가피해졌다. 그 결과, 최근의 마이크로 프로세서 개발은 단일 칩 내에 다수의 동일한 프로세싱 코어를 탑재하는 형태로 이루고 있다[1].

하지만 이와 같은 프로세싱 코어수의 증가가 직접적으로 그 성능 향상으로 이어지는 것은 아니다. 이 코어들을 충분히 활용할 수 있는 멀티 프로그래밍 기술 및 하드웨어 멀티 쓰레딩 기법의 부족 그리고 단일 코어에서 활용되는 스케줄링 정책의 사용으로 인해 프로세서 칩 내의 모든 코어를 사용하는 것은 사실상 불가능하다. 이로 인해 (그림 1)에서 보이는 바와 같이 작업을 수행하지 않는 유휴 코어와 그 코어에 속한 낭비되는 자원들이 발생하게 되며, 이는 자원 관리측면에서 큰 손실이다.

본 연구에서는 칩 멀티 프로세서 내부의 연결이 대부분 공유 버스로 이루어졌다는 사실에 기반하여, 이 버스를 이용하여 잠재적으로 발생 가능한 칩 내 유휴 캐시를 활용하는 기법을 제안한다. 한편, 본 논문에서 제안된 기법의 활용 목적은 전력 소모의 관찰을 제외한 프로세서 전체의 성능 향상으로 제한한다.



(그림 1) 유휴 코어 및 캐시의 발생

이하 본 논문의 구성은 다음과 같다. 2 장에서는 기존의 캐시 관리 기법을 다룬 논문을 소개하며, 3 장에서는 Snoop 버스를 이용한 유휴 캐시 활용 기법을 설명한다. 4 장에서는 3 장의 기법을 적용한 칩 멀티프로세서의 성능을 관찰하며, 마지막으로 5 장에서는 본 연구의 결론을 내린다.

2. 관련 연구

최근의 캐시 연구는 개별 캐시의 빠른 접근시간과 공유 캐시의 높은 hit rate 의 장점을 동시에 제공할 수 있는 복합적인 형태의 캐시에 관해 주로 이루어졌다.

D-NUCA[2]는 칩 내 캐시 블록의 공간적 위치가 그 접근 시간을 결정한다는 점에 착안하여, 캐시 뱅크 간의 dynamic cache block migration 을 통하여 특정 캐시 블록과 그 접근 프로세서 간의 공간적 위치를 동적으로 감소시킴으로써 캐시 접근의 평균 지연시간을 단축시키는 방법을 제안하였다.

Beckmann[3]은 L2 데이터 캐시를 다수의 뱅크로 이루어진 공유 캐시로 구성한 뒤, On-chip pre-fetch, cache-to-cache transfer, cache block migration 등의 기법을 사용하였다. 이를 통해 인접한 뱅크로부터 개별 캐시의 이점을 전체 뱅크로부터 공유 캐시의 이점을 취할 수 있는 방법을 제안 하였다.

Z. Chishti[4]는 뱅크 타입의 공유 캐시 환경에서 캐시 블록의 태그와 데이터 영역을 분리함으로써 불필요한 데이터의 중복과 전송을 최소화하였다.

Cooperative Cache[5]는 L2 데이터 캐시를 개별 캐시 형태로 구성한 뒤 clean block 의 cache-to-cache transfer, replication aware data replacement, replacement of inactive data 와 같은 기법을 활용하여 각각의 개별 캐시들이 유기적으로 협력함으로써 공유 캐시와 같은 효과를 낼 수 있는 방법을 제안하였다.

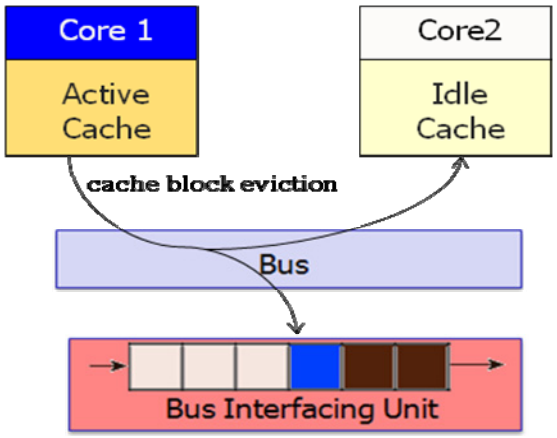
이상에서 소개된 기법들은 칩 내부에 존재하는 모든 캐시 자원들을 사용하는 것을 목표로 하고 있다. 때문에 유휴 캐시의 발생을 방지할 수 있지만, 이는 칩 전체 자원을 관리하기 위한 부가적인 하드웨어 장치와 성능 오버헤드를 요구하며 그 구현 또한 매우 복잡하다.

본 논문에서는 최하위 캐시가 개별 캐시로 구성된 환경에서 유휴 캐시의 발생을 허락함과 동시에 프로세서 사이의 공유를 가능케 함으로써, 최소한의 오버헤드로 공유 캐시와 유사한 효과를 얻을 수 있는 캐시 관리 기법을 제안한다.

3. 공유 버스를 이용한 유휴 캐시 활용기법

3.1. Victim Cache 로의 유휴 캐시 활용

Victim Cache[6]는 최하위 캐시와 메인 메모리 사이에 위치하여 최하위 캐시로부터 방출되는 캐시 블록을 저장한 뒤, 추후 요청이 있을 시에 해당 데이터를 공급한다. Victim Cache 는 그 관리의 간편성으로 인해 실제로 많이 도입되었으며, 비교적 적은 오버헤드로 높은 효과를 얻을 수 있기에 본 연구에서는 (그림 2)의 형태로 유휴 캐시를 Victim Cache 로서 사용함을 제안한다.



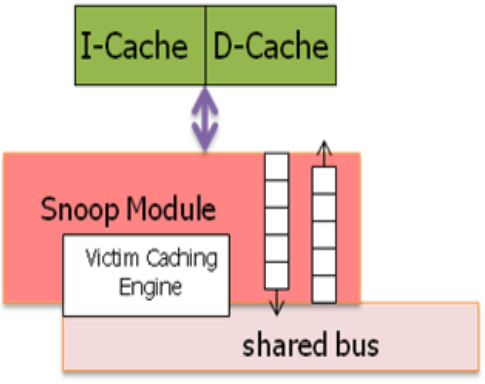
(그림 2) Victim Cache 로의 유휴캐시 사용

위 그림에서 Core 1 은 현재 작업이 할당되어 동작 중이며, Core 2 는 아무 작업도 하지 않는 유휴 코어이다. 두 코어는 외부 메모리 접근을 위한 On-chip Bus 를 공유하며 이 버스에 연결된 Bus Interfacing Unit 을 통해 외부 메모리로 접근한다. 이 때 Core 2 의 유휴 캐시(Idle Cache)는 공유 버스를 통해 Core 1 의 최하위 동작 캐시(Active Cache)의 외부 메모리 접근을 엿들을(snoop) 수 있다.

이러한 snoop 과정을 통하여 Core 2 의 유휴 캐시는 인접한 Core 1 최하위 동작 캐시를 보조하는 Victim Cache 역할을 수행한다. 본 논문에서는 Core 1 을 Core 2 의 연관 코어라 지칭한다. 한편, Core 1 의 최하위 동작 캐시에서 발생하는 메모리 요청에 대한 Core 2 유휴 캐시의 동작은 3.2 절에서 다룬다.

3.2. 유휴 캐시 활용을 위한 Snoop 프로토콜

코어는 (그림 3)과 같이 snoop module 를 통해 칩 내부 공유 버스에 연결되어 있으며, 이 snoop module 은 로컬 코어의 동작 유무에 따라 다음과 같은 작업을 수행한다.



(그림 3) Snoop Module 구성

가. 로컬 코어가 동작 중 일 때

1) 로컬 명령어 캐시와 데이터 캐시의 요청을 outgoing buffer 에 저장한 뒤, 공유 버스가 사용가능 할 때에 요청을 버스에 인가한다.

2) 공유 버스를 snooping 하면서, 1)에서 인가한 요청에 대한 응답이 오면 해당 데이터를 incoming buffer 에 저장한 뒤 캐시로 전달한다.

나. 로컬 코어가 동작 중이지 않을 때

1) 로컬 데이터 캐시의 요청을 outgoing buffer 에 저장한 뒤, 공유 버스가 사용 가능 할 때에 요청을 버스에 인가한다.

2) 공유 버스를 snooping 하면서, 버스 상에 연관 코어의 식별자(associated core id)를 가진 요청이 나타나면 해당 요청을 incoming buffer 로 가져온 뒤 Victim Caching Engine 에게 처리하도록 한다.

이처럼 로컬 코어가 동작 중이지 않을 때 snoop module 은 연관 코어의 요청을 Victim Caching Engine 으로 넘겨 줌으로써 유휴 상태인 자신의 로컬 캐시가 Victim Cache 로 사용되도록 한다.

3.3. Victim Caching Engine

Victim Caching Engine 은 연관 코어에서 발생하는 캐시 블록 방출과 캐시 읽기 실패를 확인한 뒤 각각에 맞는 동작을 수행함으로써 자신의 로컬 캐시가 Victim Cache 의 역할을 수행하도록 한다.

가. 연관 코어에서 캐시 블록(A)방출 시 유휴 코어의 Victim Caching Engine 동작

1) 연관 코어의 캐시 블록 방출을 확인하고, A 의 주소와 데이터를 snoop module 로부터 가져온다.

2) 1)에서 얻은 주소로 로컬 캐시를 탐색하여 hit 이면 3)로 진행하고 miss 하면 4)로 진행한다.

3) 로컬 캐시의 해당 주소 블록을 A 의 데이터로 업데이트 한 뒤, 5)로 진행한다.

4) A 를 로컬 캐시에 삽입하며, 동시에 방출되는 캐시 블록의 쓰기 작업을 snoop module 에 요청한다.

5) 외부 버스 접근 장치(bus interfacing unit)에 A 의 쓰기 작업을 취소하는 요청을 보낸다.

나. 연관 코어에서 캐시 블록(B) 읽기 실패 시 유휴 코어의 Victim Caching Engine 동작

1) 연관 코어의 캐시 블록 읽기 실패를 확인하고, B 의 주소를 snoop module 로부터 가져온다.

2) 1)에서 얻은 주소로 로컬 캐시를 탐색하여 hit 하면 3)으로 진행하고, miss 하면 종료한다.

3) 2)의 탐색에서 얻은 캐시 블록의 데이터를 snoop module 을 통해 연관 코어에게 전달하도록 요청한다.

4) 외부 버스 접근 장치(bus interfacing unit)에 B 의 읽기 작업을 취소하는 요청을 보낸다.

위의 과정을 통해 유휴 캐시는 연관 코어의 Victim Cache 로 동작하여 외부 메모리로의 접근을 줄이며 동시에 연관 코어의 성능 향상에 기여한다.

4. 실험 환경 및 성능 평가

4.1. 실험 환경

본 연구에서 제안한 기법의 성능 평가를 위해 ARMv5 에 기반한 Simit-ARM[7]을 프로세싱 코어 시뮬레이터로 사용하고, SystemC[8]를 기반으로 한 칩 멀티 프로세서 시뮬레이터를 제작하였다. 그 상세 설정은 <표 1>과 같다.

<표 1> 4-코어 및 16-코어 실험 환경 설정

	4 Core	16Core
Block Size	32B	64B
L1 I-cache	16KB, 32-way, 2Cycles	16KB, 32-way, 2Cycles
L1 D-cache	16KB, 32-way, 2Cycle	16KB, 32-way, 2Cycles
L2 D-cache	None	512KB, 32-way, 8Cycles
External Memory	80Cycles	250Cycles
Idle Cache Access	8Cycles	8Cycles

▪ 4-코어 칩 멀티 프로세서 환경

Desktop 환경에 비해 비교적 작은 메모리 구조를 가지고 있는 embedded 프로세서에서의 캐시 성능은 전체 성능에 큰 영향을 미친다. 따라서 본 연구에서는 최근 발표된 ARM11 MPCore 를 모델로 삼아 다음과 같이 실험 환경을 구성하였다.[9]

1) 칩 멀티 프로세서는 총 4 개의 동일한 코어로 구성되어 있으며, 각각의 코어는 L1 명령어 캐시와 L1 데이터 캐시를 가진다.

2) 각 코어의 L1 데이터 캐시는 해당 코어가 유휴 상태 일 때에 유휴 캐시로 전환 및 사용이 가능하다.

3) 유휴 캐시 활용 시에 필요한 코어 간의 연관(association)은 인접한 두 개의 코어가 쌍을 이뤄 고정한다.

▪ 16-코어 칩 멀티 프로세서 환경

높은 집적도를 가진 16-코어 칩 멀티 프로세서 환경은 다음과 같이 구성된다.

1) 칩 멀티 프로세서는 총 4 개의 동일한 코어로 구성되어 있으며, 각각의 코어는 L1 명령어 캐시와

L1 데이터 캐시 및 L2 데이터 캐시를 가진다.

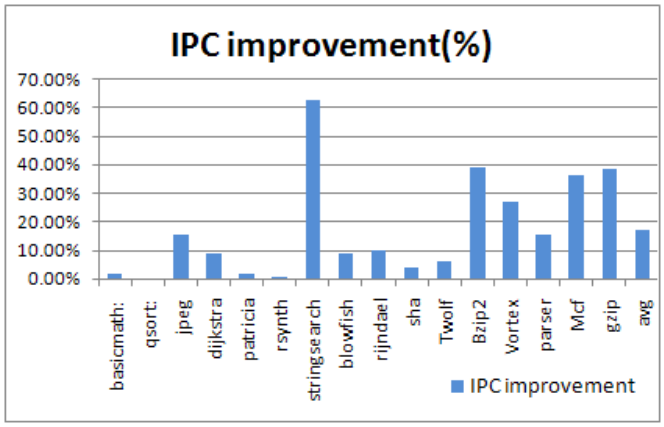
2) 각 코어의 L2 데이터 캐시는 해당 코어가 유휴 상태 일 때에 유휴 캐시로 전환 및 사용 가능하다.

3) 유휴 캐시 활용 시에 필요한 코어 간의 연관(association)은 인접한 두 개의 코어가 쌍을 이뤄 고정한다.

4) 4 개의 코어는 하나의 sub-set 을 구성하며 sub-set 은 내부 공유 버스를 가진다. 4 개의 sub-set 은 상위 공유 버스를 통해 연결되며 이를 통해 외부 메모리로 접근한다.

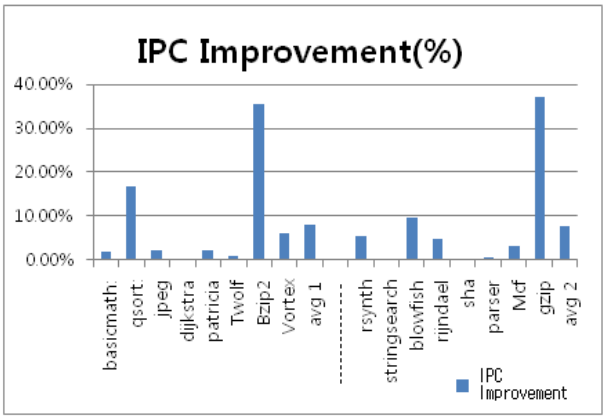
4.2. 실험 결과

본 연구의 검증을 위해 Mibench 의 벤치마크 10 가지와 SPEC INT2000 의 벤치마크 6 가지를 사용하여 실험하였다.



(그림 4) 4-코어 모델에서의 벤치마크 별 성능 향상

4-코어 모델의 실험에서는 4 개의 코어 중 2 개의 코어만이 동작중인 상황을 가정하였고, 나머지 2 개 코어의 캐시는 유휴 캐시로 사용되었다. 실험 결과 벤치마크 별로 평균 17%의 IPC 성능 향상을 확인할 수 있었다.



(그림 5) 16-코어 모델에서의 벤치마크 별 성능 향상

16-코어 모델의 실험에서는 16 개의 코어 중 8 개의 코어만이 동작중인 상황을 가정하였고, 나머지 8 개 코어의 캐시는 유휴 캐시로 사용되었다. 실험 결과 벤치마크 별로 평균 8%의 IPC 성능향상을 확인할 수 있었다.

5. 결론

본 논문에서는 공유 버스에 기반한 칩 멀티 프로세서 상에서 발생 가능한 유휴 코어의 자원인 유휴 캐시를 확인하고, 이를 Victim Cache 로서 활용하여 프로세서의 성능을 향상시킬 수 있는 기법을 제안하였다. 또한 기존의 연구들이 가지고 있던 하드웨어 및 성능 오버헤드를 최소화 하기 위해, 개별 캐시 기반의 snoop 버스 프로토콜을 고안하였다. 이를 검증하기 위해 각각 4 개, 16 개의 코어로 구성된 칩 멀티 프로세서 실험 환경을 작성하였고, 그 결과 각각의 환경에서 17%와 8%의 IPC 성능 향상을 확인하였다.

.

참고문헌

[1]Borkar,S.“Thousand Core Chips—A Technology Perspective” Proc. 44th Design Automation Conf. (DAC07).

[2]C.Kim, and D.Burger, “An adaptive, non-uniform cache st ructure for wire-delay dominated on-chip caches “ .ASPLOS-X

[3]B.M. Beckmann, D.A.Wood. “Managing wire delay in large chip-multiprocessor caches”. *MICRO* 37th

[4]Z., Chishti, M.,and Powell D. “Optimizing replication, communication and capacity allocation in CMPs”. Proceedings of the 32nd International symposium on Computer Architecture

[5]Jichuan, Chang and Gurindar, Sohi S. “Cooperative Caching for Chip Multiprocessors”. Proceedings of the 33rd International symposium on Computer Architecture

[6]N., Jouppi. “Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers”. Proceedings of 27th International Symposium on Computer Architecture.

[7]W.Qin and S.Malik. “Flexible and Formal Modeling of Microprocessors with Application to Retargetable Simulation”. Design, Automation, and Test in Europe (DATE 2003)

[8]T.Grötker, et al. “System Design with SystemC”. Springer.

[9]Kazuyuki Hirata, ARM Corporation. “ARM11 MPCore”