계층적 메모리 구조의 효과를 극대화하는 캐시 제어기

(A Cache Controller to Maximize Effectiveness of Hierarchical Memory Architecture)

어 봉 용 † 주 영 관 †† 전 중 남 ††† 김 석 일 †††
(Bong Yong Uh) (Young Kwan Ju) (Joong Nam Cheon) (Suk Il Kim)

요 약 이 논문에서는 계층적 캐시 구조에서 기존의 레벨 2 캐시 미스 시에만 선인출 하도록 되어있는 구조를 레벨 1 캐시 미스 시에도 선인출 하도록 하는 캐시구조를 제안하였다. 즉, 레벨 1 캐시 미스가 발생하면 레벨 2 캐시로부터 요구블록과 선인출 블록을 선택하여 레벨 1 캐시와 선인출 캐시에 각각 적재한다. 11개의 벤치마크 프로그램에 대한 실험결과, 레벨 1 캐시 선인출기와 레벨 2 캐시 선인출기로 구성한 계층적 캐시구조가 레벨 2 캐시 선인출기만 채용한 기존의 캐시구조에 비하여 최대 19%의 성능향상을 얻을 수 있었다.

키워드 : 선인출 캐시, 메모리 계층 구조, 시뮬레이션

Abstract A cache architecture is proposed here which evokes prefetch at level 1 cache miss. Existing structures only prefetch at level 2 cache miss. In the proposed cache architecture, level 1 cache miss would select demand fetch block and prefetch block from the level 2 cache and store to level 1 cache and prefetch cache, respectively. According to an experimental analysis using 11 benchmark programs, the hierarchical cache architecture that employs both a level 1 cache prefetcher and a level 2 cache prefetcher obtained a maximum 19% increased performance when compared to the cache architecture that employs only a level 2 cache prefetcher.

Key words : Prefetch Cache, Memory Hierarchy Architecture, Simulation

1. 서 론

컴퓨터 구조에서 중앙처리장치 처리속도와 주 메모리의 입출력속도간의 차이가 갈수록 벌어지고 있다. 또한 갈수록 그 크기가 증대되는 응용프로그램에서의 메모리 참조비율은 전체 명령어 대비 43%에 이르고 있다[1]. 이는 잦은 메모리 참조로 인하여 메모리 참조에 걸리는 시간 동안 중앙처리장치가 대기해야 하는 시간이 늘어나 응용프로그램의 전체 실행시간이 그만큼 늦어짐을 의미한다.

이 문제를 해결하기 위해서는 메모리 참조에 따른 지연시간을 감소시켜야 하며, 그 일환으로 저렴한 비용으로 평균 메모리 참조지연시간을 줄이는 계층적 캐시구조[2-4], 메모리 참조명령어를 가능한 한 먼저 수행하도록 하고 동시에 다른 명령어를 처리하도록 하는 명령어 재배치 기법[5], 필요한 데이타를 필요한 시점 이전에 인출하여 캐시나 레지스터에 적재하는 캐시 선인출 기법[3]등이 제안되었다. 그 중에서도 캐시 선인출 기법은 기존의 계층적 캐시구조에서 간단한 캐시 선인출 알고리즘을 캐시제어기에 추가하는 것으로도 캐시 미스율을 크게 낮출 수 있어서 근래에 많은 연구가 수행된 바 있다[2,3,6,7].

캐시 선인출과 관련한 대부분의 연구에서는 레벨 2(L_2)캐시 미스가 발생되었을 경우에 캐시 미스를 일으킨 메모리 블록 주소를 중심으로 연속한 이웃 메모리 블록을 선인출하는 단일블록 예측(One Block Lookahead: OBL)기법[8], 여러 개의 이웃한 메모리 블록을

· 이 논문은 2004년도 충북대학교 학술연구지원사업의 연구비 지원에 의하여 연구되었음
† 정 회 원 : 유비쿼터스 바이오정보기술센터 연구원
uby@chungbuk.ac.kr
†† 학생회원 : 충북대학교 컴퓨터과학과
juyg@john.chungbuk.ac.kr
††† 종신회원 : 충북대학교 전기전자컴퓨터공학부 교수
joongnam@cbu.ac.kr
ksi@chungbuk.ac.kr
(Corresponding author임)
논문접수 : 2005년 3월 2일
심사완료 : 2005년 7월 24일

선인출하도록 하는 다블록 예측(Multi Block Look-ahead: MBL)기법[9]이 연구된 바 있다.

이들 기법은 선인출 블록의 주소결정이 간단한 반면 선인출하는 메모리 블록의 주소가 선인출한 몇 개의 블록을 벗어나는 경우에 캐시에 적재된 선인출된 메모리 블록의 효용성이 없거나 떨어지며, 효용성이 낮은 메모리 블록들이 캐시에 적재됨으로 인하여 유용한 캐시의 메모리 블록들이 자주 교체되는 캐시오염 현상이 발생하게 된다.

캐시오염을 줄이기 위하여 보다 정교한 선인출 기법도 연구되었다. 예를 들면, 메모리 블록의 참조주소를 토대로 다음 번에 참조될 메모리블록의 주소를 예측하는 것을 들 수 있다[7,10-12]. 이 기법은 OBL이나 MBL에 비하여 정확한 메모리 블록의 예측이 가능하므로 캐시오염의 가능성이 그만큼 줄어드는 장점이 있다.

캐시 선인출과 관련하여 캐시 선인출 기법에 대한 연구는 활발히 진행되었으나 캐시구조에 대한 연구는 부분적인 연구에 국한되었다. 예를 들면 캐시오염을 줄이기 위하여 캐시와 메모리장치간에 버퍼를 두고 선인출된 블록을 버퍼에 일단 적재하고 이들 중 실제로 참조된 블록만을 캐시로 이동시키는 구조[9]가 제시된 바 있으며, 기존의 캐시와 별도로 선인출 캐시를 추가하여 선인출 캐시에 선인출 블록을 적재하고 요구 블록은 기존의 캐시에 적재하는 이중캐시구조[10,13]가 제안된 바 있다. 그러나 이들 캐시 선인출 기법이나 선인출 캐시 구조는 캐시구조가 계층구조로 구성되어 있음에도 불구하고 레벨 2 캐시 미스에 의해 선인출 요구가 발생하고 이로 의하여 선인출되는 메모리 블록이 레벨 1 캐시와 메모리 사이의 버퍼나 선인출 캐시에 적재되는 형태를 취하고 있다.

즉, 캐시구조가 레벨 1 캐시 및 레벨 2 캐시로 구성되어 있으나 기존의 선인출 캐시구조는 단일 레벨에 국한되어 있어서 계층구조 캐시의 효과를 충분히 살리지 못하고 있다. 본 논문에서는 계층으로 구성된 캐시구조를 보다 효과적으로 활용하는 선인출 제어기법을 제안하였다. 본 논문의 구성은 다음과 같다. 제2절에서는 기존의 계층적 캐시구조를 분석하고 연구의 동기를 기술하였다. 제3절에서는 본 논문에서 제안하는 선인출 기법이 적용된 계층적 캐시구조를 제시하였다. 제4절에서는 본 논문에서 제안한 구조와 기존의 방법을 비교하기 위한 모의실험을 수행하고 그 결과를 비교하였다. 마지막으로 제5절에서는 본 논문의 결론을 도출하였다.

2. 관련 연구

계층적 메모리 구조는 주 메모리 장치와 중앙처리장치 사이에 다단계의 캐시 장치를 두어 비용을 낮추는 대신 평균 메모리 참조시간을 줄이기 위한 방법이다. 근래에는 2단계의 캐시장치로 구성한 계층적 메모리구조가 개인용 컴퓨터에서 슈퍼컴퓨터에 이르기까지 광범위하게 채용되고 있다. 그림 1은 레벨 1 캐시와 레벨 2 캐시로 구성된 계층적 메모리 구조를 도식화한 것이다. 여기서 레벨 1 (L_1)캐시는 그 크기가 작은 대신 참조속도가 매우 빨라 CPU사이클당 하나의 워드(Word)를 참조 할 수 있다.

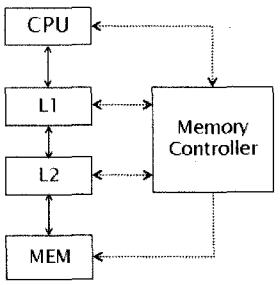


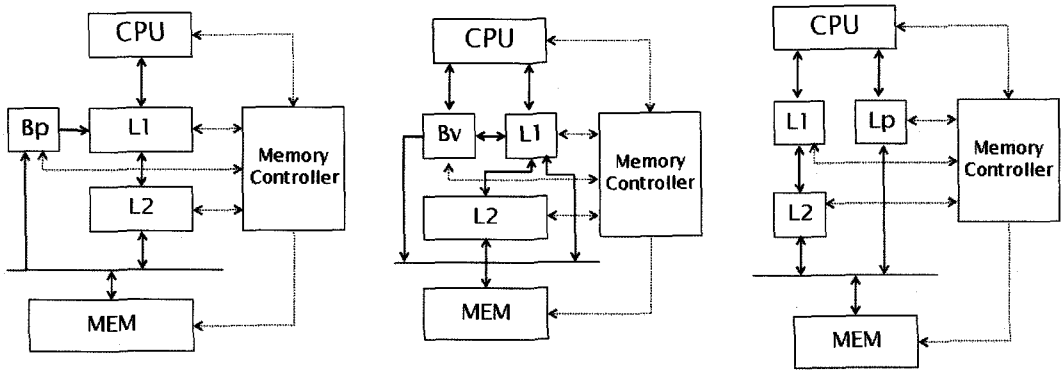
그림 1 이단 계층의 메모리 구조

예를 들어, 인텔사의 Pentium[14]은 칩 내에16KB의 L_1캐시를 내장하고 있으며 Pentium III[15]는 16KB, Pentium IV[15]는 8KB의 L_1캐시를 내장하고 있다. 모토롤라사의 PowerPC 60X[16]는 칩 내에 32KB의 L_1캐시를 내장하고 있다. 또한 이들 프로세서에서 L_1캐시의 참조지연시간은 2~3사이클을 갖는다.

레벨 2 (L_2) 캐시의 경우에는 L_1에 비하여 더 큰 용량을 갖고 있지만 참조지연시간은 길다. 예를 들어 Pentium과 Pentium III, IV의 경우에는 메모리 제어기가 허용하는 L_2캐시의 크기가 각각 512KB, 1024KB, 2048KB이며 PowerPC 60X의 경우에는 1024KB이다. 또한 이들 L_2캐시의 참조지연시간은 12~45사이클이다[7,14-17]. L_2캐시와 메모리간의 참조지연시간 비율이 10~20배 되는 반면 L_2캐시와 L_1캐시의 참조지연시간은 12~30배의 비율을 갖는다. 메모리 참조 지연시간은 150~320사이클을 갖는다. 기존의 연구들은 참조지연시간이 상대적으로 큰 메모리와 L_1캐시간의 미스 감소에 중점을 두고 있다. 그 동안 연구된 선인출 구조는 그림 2와 같다.

2.1 선인출 버퍼구조

선인출 버퍼구조[9]는 그림 2(a)와 같이 메모리와 L_1 간에 빠른 버퍼(B_p)를 두고 선인출하는 메모리 블록을 B_p에 일단 저장하도록 하는 구조이다. 즉, CPU가 L_1을 참조할 때 B_p를 함께 참조하도록 하고 B_p에 적재된 메모리블록이 참조되면 이 블록을 B_p에서 L_1으로 옮겨 이 블록에 포함된 데이타의 반복적인 사용에 대비한다. 따



(a) 선인출 버퍼구조 (b) 희생 버퍼구조 (c) 선인출 캐시구조

그림 2 데이타 선인출을 위한 캐시 구조

라서 이 구조에서는 사용되지 않을 블록이 L_1에 적재됨으로 인하여 발생하는 캐시오염을 방지할 수 있다.

또한 B_p는 읽기전용(read only)버퍼로 사용된다. 그 이유는 B_p에 있는 블록을 처음 참조하면 이 블록은 L_1으로 이동되므로 B_p에 존재하는 블록은 한번도 참조되지 않은 블록들이다. 따라서 가장 오래된 블록은 그 동안 사용되지 않았으며 앞으로도 사용될 가능성이 없으므로 B_p의 교체전략으로는 FIFO가 쓰인다.

2.2 희생버퍼구조

희생버퍼구조[20]는 L_1캐시가 선인출 블록을 저장할 때 L_1캐시로부터 교체되어야 하는 블록을 임시로 저장하는 버퍼를 운영하는 구조이다. 즉, L_2캐시 미스가 발생하면, 요구된 메모리 블록 주소를 토대로 선인출할 메모리 블록이 결정되고 각각의 메모리 블록은 L_2캐시를 거쳐 L_1캐시에 적재되고, 이어서 선인출블록이 L_1캐시에 적재된다. L_1캐시에 선인출블록이 적재되는 경우에 L_1에 적재되었던 어떤 블록이 제거되어야 한다. 이때 이 메모리 블록은 다시 참조될 가능성이 있음에도 불구하고 사용되지 않을 수도 있는 선인출 블록으로 인하여 L_1캐시로부터 강제로 제거되는 문제점이 있다.

이러한 문제점을 개선하기 위하여 그림 2(b)와 같이 희생버퍼(B_v)를 L_1캐시와 L_2캐시 사이에 두고 어떤 선인출 메모리블록이 L_1캐시에 적재될 때 L_1캐시로부터 제거되어야 하는 블록을 B_v에 저장하도록 한다. 즉, CPU가 L_1캐시에 접근할 때 B_v도 동시에 접근하도록 한다.

만일 참조하는 데이타의 주소가 B_v에 적재된 블록을 가리키면 CPU는 이 데이타를 참조하여 연산을 하며, 동시에 메모리 제어기는 참조된 B_v의 블록을 다시 L_1캐시로 이동시킨다. 이때 L_1캐시에서 제거되는 블록은 L_2캐시를 거쳐 주 메모리로 이동한다. 희생버퍼 B_v는 FIFO방식으로 운영된다. 이 구조는 작은 희생버퍼로 선인출에 의한 캐시오염을 완화시킬 수 있는 특징이 있다.

L_2캐시를 희생버퍼로 사용할 수도 있다. 이는 L_1캐시에서 참조된 적이 있는 블록들이 L_2캐시로 교체된 후에도 이들 블록들의 65%이상이 다시 참조될 수 있다는 연구결과[17]에 따라 선인출에 의하여 L_1캐시로부터 제거되는 블록을 L_2캐시에 임시로 보관하는 것이다. 이때 이 블록으로 인하여 L_2캐시에서 희생되어야 하는 문제를 최소화하기 위하여 이 블록의 우선순위를 L_2캐시에서 가장 낮게 책정한다. 이 방법은 별도의 희생버퍼를 추가하지 않고도 충분한 선인출 효과를 얻을 수 있는 장점이 있다[17].

2.3 선인출 캐시 구조

선인출 캐시 구조[6]는 L_1과 별도로 선인출 블록을 저장할 레벨 1 수준의 캐시를 운영하는 구조이다. 즉, L_2캐시의 미스에 의하여 요구된 메모리 블록의 주소를 토대로 선인출할 메모리 블록이 결정되면, 메모리 제어기는 이들 메모리 블록을 주 메모리로 요구하게 된다.

일정한 메모리 지연시간이 경과한 후에 이들 메모리 블록이 준비되면 요구된 메모리 블록은 L_1캐시에 적재되고 선인출한 메모리 블록은 선인출 캐시에 적재된다. 따라서 이 구조에서는 선인출 블록은 L_1캐시에 영향을 주지 않으므로 L_1캐시의 오염을 근본적으로 방지할 수 있는 장점이 있다. 그림 2(c)는 선인출 캐시 구조를 도식화 한 것으로 L_p가 선인출 캐시이다. 이 구조에서 CPU는 메모리를 참조할 때 L_1과 L_p를 동시에 접근한다. 만일 L_p에 있는 메모리 블록이 참조되더라도 L_p가 캐시역할을 담당하므로 선인출 버퍼나 희생버퍼의 경우와는 달리 이 블록을 L_1으로 이동할 필요가 없는 장점이 있다.

이상과 같은 세가지 구조는 나름대로 CPU가 필요로 하는 데이타를 사전에 선인출할 수 있도록 하므로 선인출 기능이 없는 구조와 비교하면 매우 우수한 효과를

보여준다[18,19]. 그러나 세가지 구조모두 L_2캐시의 미스에 의하여 선인출이 시작되며 주 메모리와 L_1캐시간의 속도차이를 극복하는데 초점을 모으고 있다. 그러나 계층적으로 구성된 메모리구조에서 L_1캐시와 L_2캐시간의 참조속도 비율이 L_2캐시와 주 메모리간의 참조속도비율에 비하여 매우 크다. 표 1은 주요 프로세서 별로 계층별 캐시 참조지연속도의 비율을 비교한 것이다.

표 1 캐시와 메모리 레벨간의 참조지연시간 비율

프로세서	비 율	
	L_2 / L_1	L_2 캐시 크기
Alpha 21164	16 / 2	96KB
PowerPC 604	29 / 2	512KB
MPC7400	30 / 2	1024KB
UltraSPARC	38 / 2	512KB
Pentium	12 / 2	256KB
Pentium II	35 / 3	256KB
Pentium III	45 / 3	512KB
Pentium IV	40 / 3	512KB

또한 L_1캐시 미스 수는 L_2캐시 미스수에 비하여 10~15배에 달한다[17]. 이는 L_2캐시 미스시에 주 메모리로부터 필요한 메모리블록을 선인출할 필요가 있을 뿐 아니라 L_1캐시의 미스가 발생하였을 때에도 L_2에 적재된 메모리블록 중에서 곧 사용될 블록을 L_1으로 선인출할 필요가 있음을 알려준다. 본 논문에서는 이러한 관찰을 토대로 L_2캐시 미스뿐 아니라 L_1캐시 미스 시에도 각각 주 메모리장치 및 L_2캐시로부터 메모리 블록을 선인출하는 구조의 캐시를 제안하였다.

3. 계층적 선인출 메모리 구조

3.1 목표 구조

이 논문에서 제안하는 계층적 선인출 메모리 구조는 그림 3과 같다. 목표구조는 레벨 1 캐시로 L_1과 L_p를 두고 각각 요구된 메모리 블록과 선인출 블록을 구분하여 적재하도록 한다. 따라서 기본적인 구조는 제2절의 선인출 캐시구조와 동일하다. 그러나 목표구조에서는 L_1 캐시 미스가 발생할 때 참조할 블록과 함께 하나의 블록을 추가로 L_2캐시로부터 선인출하여 L_p캐시로 이동하도록 한다. 즉 기존의 구조에서 L_1캐시 미스는 단순히 L_2캐시에 적재된 하나의 메모리 블록을 L_1캐시로 이동하거나 L_2캐시 미스를 유발시키도록 하였으나 목표구조에서는 L_1캐시의 캐시 미스가 L_2캐시에 적재된 두 개의 메모리 블록을 차례로 이동하도록 구성하였다. 그림 4는 프로세서가 메모리 참조명령어를 수행하는 과정에서 메모리 제어기가 수행하는 캐시 운용 절차를 흐름도로 나타낸 것이다.

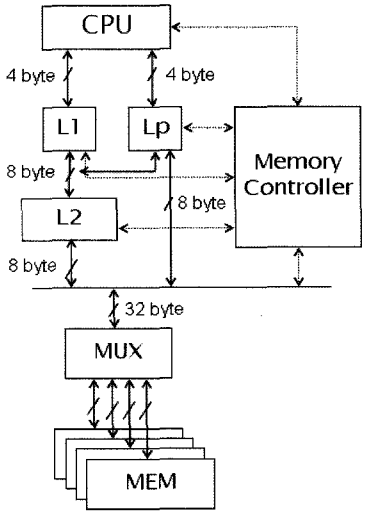


그림 3 목표 구조

그림 4에서 회색으로 표시한 부분이 기존의 선인출 캐시구조에 추가된 L_1캐시의 선인출 기능부분이다. 즉, L_1캐시 미스가 발생되면 L_2에 필요로 하는 메모리블록을 요구하고 이어서 선인출할 블록의 주소를 결정한 후에 선인출 블록이 준비되면 이 블록을 L_p로 적재한다. 이는 선인출기의 부정확한 예측으로 인한 기존 L_1캐시 블록을 강제로 제거하는 것을 방지함으로써 L_1캐시 오염현상을 피하였다.

만일 선인출 블록이 L_2에 존재하지 않을 경우에는 해당 블록을 메모리로 요청하지 않는다. 즉 L_1캐시 미스로 인한 선인출 블록이 L_2캐시에 존재하지 않으면 L_p캐시로 선인출 하지 않는다. L_1캐시와 L_2캐시의 미스로 인하여 발생하는 선인출 블록의 결정방법은 3.2절에서 다루기로 한다.

CPU와 L_1캐시 및 L_p캐시간의 대역폭은 모두 1워드(4 byte)이다. 또한 L_1캐시 및 L_p캐시와 L_2캐시간의 대역폭은 2워드이며 L_p캐시와 메모리간에도 대역폭은 2워드이다. 그러나 주 메모리는 4-way 인터리빙 방식으로 구성하여 4개의 독립된 메모리 모듈로부터 4개의 워드를 참조하여 L_2캐시에 전송하도록 구성하였다. 따라서 L_p캐시는 메모리 모듈로부터 빠른 선인출을 위하여 4개의 멤버 모듈 중 하나의 모듈로부터 1워드만을 인출하여 적재하도록 하였다. L_2캐시로부터 L_2캐시의 블록사이즈인 4워드를 모두 선인출하는 경우에 이를 위한 참조지연시간이 길어지며 엔트리수가 적은 선인출 캐시의 블록이 자주 교체됨에 따라 선인출 캐시의 힛트율이 적어지게 된다.

이 논문에서 제안한 프로세서의 구조에서 모든 비 메모리 참조 명령어는 한 사이클에 동작하는 것으로 가정

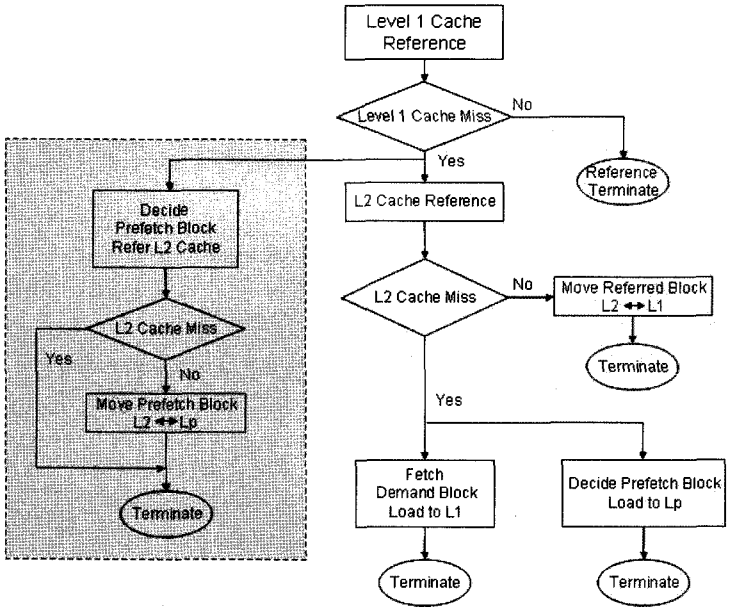


그림 4 목표구조의 메모리 참조 흐름도

하였다. L_1캐시는 write-through 교체정책을 채택하였다. L_2캐시는 4-way 인터리빙 방식을 위한 멀티뱅크로 동작하며, 128 바이트 라인으로 된 512Kb크기의 직접 사상구조이다. 이 캐시는 write-back교체정책을 채택하였다. 주소와 데이타버스는 L_1에 대해 각각 4사이클의 지연을 갖고 있다. 주 메모리는 요구인출과 선인출의 메모리 접근을 위해 4개의 메모리 뱅크로 동작한다. 여기서 주소와 데이타버스는 외부메모리와 4사이클의 지연을 가지며 선인출 큐의 길이는 64엔트리로 구성되어 있다.

레벨 1 캐시의 미스 발생시 다음 캐시 블록주소를 계산하고 인출명령어를 발생시키기 위해서는 OBL알고리즘을 구성하는 ADDER와 L_2캐시에서 선인출한 블록을 선인출 버퍼로 복사하는 로직이 필요하다. 이렇듯 목표구조에 추가되어야 하는 회로는 L_1캐시와 L_2캐시간의 버스 대역폭을 2배로 늘리는 것보다 적은 수의 게이트로 구성될 수 있으므로 전체적으로 하드웨어의 복잡도는 증가하지 않는다.

3.2 선인출 알고리즘

제안된 목표구조에서는 각 캐시 레벨마다 선인출기가 필요하다. 우선 L_1캐시 선인출기는 L_1캐시의 캐시 미스에 의하여 요구인출 블록을 L_2캐시로부터 L_1캐시로 복사 이동시키고, 곧이어 L_1캐시가 필요로 하는 선인출 블록을 선택하여 L_p캐시로 적재하도록 한다. L_2캐시 선인출기는 기존의 구조와 동일하다. 즉 L_2캐시에서 캐시가 발생하면 선인출기는 메모리로부터 요구인출 블록을 포함한 캐시라인을 L_2캐시에 적재하고 해당블록을 L_1캐시에 적재한다. 이어서 선인출기는 주 메모리로부터 선인출 블록을 선택하여 인출한다. 레벨 1 캐시 선인출기는 레벨 2 선인출기와는 달리 선인출을 위한 절차가 두 사이클 내에 처리될 수 있어야 한다. 따라서 레벨 1 캐시 선인출기에 사용될 수 있는 선인출 알고리즘은 처리속도가 빠를수록 좋다. 따라서 본 논문에서는 선인출 블록의 주소를 가장 빠르게 결정할 수 있는 OBL(One Block Lookahead)기법을 사용한다.

레벨 2 캐시 선인출기법은 이미 많은 연구자에 의하여 연구된 바 있다. 예를 들어, OBL이나 MBL는 연속한 이웃 메모리 블록을 선인출하는 기법으로 알고리즘이 간단한 장점이 있으나 선인출 메모리 블록의 주소가 선인출한 몇 개의 블록을 벗어나는 경우에 효용성이 낮고 이로 인한 캐시오염을 유발시키는 단점이 있다. 상관예측표 선인출 기법[21]은 메모리 블록의 참조주소를 토대로 다음 번에 참조될 메모리블록의 주소를 예측하는 기법이다. 따라서 이 알고리즘은 참조하는 메모리 블록의 주소변화가 일정할 때 매우 유용한 방법이다. 이는 OBL이나 MBL에 비하여 정확한 선인출 블록의 예측이 가능하나 참조주소를 기억시킬 공간이 많이 소요되는 단점이 있다. 참조예측표기법[11]은 상관예측표 선인출 기법에 비하여 그 구조가 간단하다. 즉 일정한 메모리 블록 주소 차이를 가지며 참조되는 메모리 블록이 예측한 선인출 블록과 일치하는 동안 자동적으로 선인출 블록을 결정한다. 이 구조는 상관 예측표 선인출 기법에

대하여 훨씬 간단하다. 보다 복잡하고 진일보한 기법으로 필터 기법이 있다. 이는 선인출하고자 하는 블록들 중에서 실제로 참조되지 않는 블록들인지 여부에 대한 정보를 유지하고, 선인출이 필요한 시점에서 이들 블록이 사용되었던 적이 있는 경우에만 선인출을 하도록 허용하는 기법이다. 본 논문에서는 이들 몇 가지 후보 중에서 상관 예측표 선인출 기법과 필터 기법을 L_2캐시 선인출기의 알고리즘으로 선정하였다.

L_1캐시 선인출기에 OBL기법을 적용하면 결국 L_2캐시로부터 2개의 연속된 블록을 파이프라인 방식으로 L_1캐시로 이동하는 것과 동일한 효과를 지닌다. 즉 L_2캐시와 L_1캐시간의 대역폭이 2워드 (4바이트 × 2)인 셈이 되므로 L_1캐시와 L_2캐시간의 버스를 두 배로 확장한 효과를 얻을 수 있다. 실제로 L_1캐시 선인출 기는 요구블록과 선인출 블록을 시차를 두고 파이프라인 형식으로 한번에 한 워드씩 데이타가 이동하므로 버스를 확장할 필요는 없다.

4. 실험 및 분석

4.1 실험 환경

이 논문에서 목표로 하는 계층적으로 구성한 캐시 구조의 성능 분석을 위해 알파 기종에서 제공되는 Atom [22]을 사용하여 응용 프로그램의 트레이스를 얻었다. 또한 실험에서는 범용벤치마크인 SPEC과 미디어 벤치마크인 MediaBench로 이루어진 11개의 프로그램으로부터 생성된 4천만 개 이상의 명령어로 구성된 트레이스를 바탕으로 각 계층별 캐시 미스율과 성공률, 선인출의 효과 및 벤치마크의 총 실행 사이클을 측정하였다. 이를 위하여 본 논문에서는 멀티쓰레드 방식으로 동작하는 메모리 제어기용 시뮬레이터를 구현하였다.

이 실험에서 사용된 벤치마크 프로그램의 속성은 표 2와 같다. 표 2에서 7개의 프로그램은 MediaBench에서 선정하였으며 4개는 모두 SPEC2000 INT에서 선정하였다. 표 2에서 f_1은 아무런 선인출기가 없는 기존의 계층적으로 구성한 캐시구조에서 L_1캐시 미스율을, f_2는 L_2캐시 미스율을 측정한 것이다.

4.2 실험 결과 및 분석

선인출기를 채용함에 따른 계층적 캐시구조의 성능향상을 비교하기 위하여 선인출기가 하나도 없는 기존의 캐시구조, 레벨 2 캐시 선인출기만 채용하고 있는 캐시구조 및 본 논문에서 제안하는 레벨 1 캐시 및 레벨 2 캐시용 선인출기가 모두 포함된 캐시구조에서 표 2의 벤치마크를 대상으로 사이클당 실행 명령어 수, L_1캐시와 L_1캐시의 캐시 미스율, 커버리지, 선인출된 블록의 참조 성공률을 측정하였다.

레벨 2 캐시 선인출기에는 상관예측표 선인출 기법[6, 21]과 필터 기법 [7]을 모두 적용하는 것으로 가정하였으며, 레벨 1 캐시 선인출기에는 OBL[8]을 적용하였다. 따라서 실험에서는 표 3과 같이 5가지 경우를 실험하였다.

표 3 실험에 사용된 캐시 구조

Legend	L_1 Prefetcher	L_2 Prefetcher
R	None	None
NC	None	Correlation Table Algorithm
NF	None	Filtering Table Algorithm
OC	OBL	Correlation Table Algorithm
OF	OBL	Filtering Table Algorithm

4.2.1 사이클당 명령어 수

동일한 벤치마크에 대하여 표 3에 보인 5가지 캐시구조에서 실행에 소요되는 총 사이클을 측정하면 캐시구조간의 성능비교가 가능하다. 그러나 총 사이클은 벤치마크별로 큰 차이가 있으므로 본 논문에서는 단위사이클당 실행된 명령어수(Instruction Per Cycle: IPC)를 기준으로 비교하였다. 그림 5는 IPC를 측정한 결과이다.

그림 5에서 알 수 있듯이 레벨 1 캐시 선인출기와 L_2캐시 선인출기를 모두 도입한 OC와 OF구조가 L_2캐시

표 2 벤치마크 응용프로그램

Benchmark	Description	f1	f2
Cjpeg	JPEG image Compression	.0202	.0121
Djpeg	JPEG image Decompression	.0173	.0084
Mpegenc	MPEG2 Compression	.0031	.0014
Mpegdec	MPEG2 Decompression	.0009	.0006
Epic	EPIC image Compression	.0201	.0148
Unepic	EPIC image Decompression	.0345	.0122
Ghostscript	An interpreter for the PostScript language	.0407	.0232
Gcc	C Programming Language Compiler	.0160	.0066
Gzip	GNU Compression	.0367	.0319
Perl	PERL Programming Language	.0420	.0257
TimberWolfSC	Place and Route Simulator	.0886	.0555

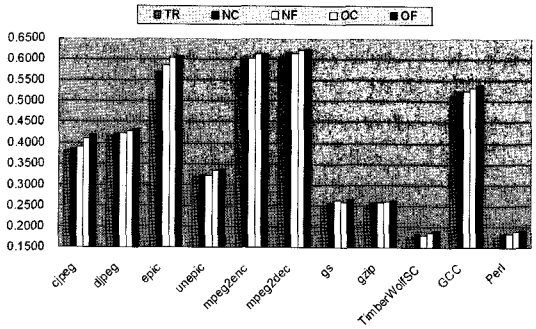


그림 5 단위 사이클당 실행된 명령어 수

선인출기만 도입한 NC와 NF구조보다 IPC가 더 큼을 알 수 있다. 이는 간단한 알고리즘으로 구성한 레벨 1 캐시 선인출기를 채용함으로써 기존의 레벨 2 캐시 선인출기만 채용한 구조에 비하여 그 성능이 크게 향상되는 것을 보여주는 것이다. 또한 레벨 2 캐시 선인출기에 대하여서는 상관예측표 기법에 비하여 필터 기법을 적용한 구조가 성능이 더 좋았음을 확인할 수 있었다.

4.2.2 캐시 미스율 비교

L_1캐시와 L_2캐시의 미스율이 높을수록 평균 메모리 참조시간이 늘어남을 의미한다. 또한 L_1캐시의 경우에도 미스율이 증가할수록 L_2캐시에서의 선인출 횟수가 증가한다. 마찬가지로 L_2캐시의 미스율이 증가할수록 주 메모리를 대상으로 하는 선인출의 횟수가 늘어난다. 따라서 5가지 조합의 캐시구조의 성능을 비교하기 위하여 L_1캐시와 L_2캐시 미스율을 각각 측정하였다. 그림 6은 L_1캐시의 미스율을, 그림 7은 L_2캐시의 미스율을 측정한 결과이다.

그림 6은 다섯 가지 계층 구조별 L_1캐시의 캐시 미스율을 나타낸 것이다. 그림 6에서 알 수 있듯이 본 논문에서 제안한 레벨 1 캐시 선인출기를 추가한 계층구조인 OC와 OF구조에서 L_1캐시 미스율이 레벨 2 캐시 선인출기만 채용한 구조에 비하여 더 낮음을 알 수 있다. 그림 6에서 unepic과 TimberWolfSC 벤치마크의 경우에는 NC가 NF보다 우수한 것으로 판명되었다. 그 이유는 테이블이 필터 기법에 사용된 16KB크기가 상관예측

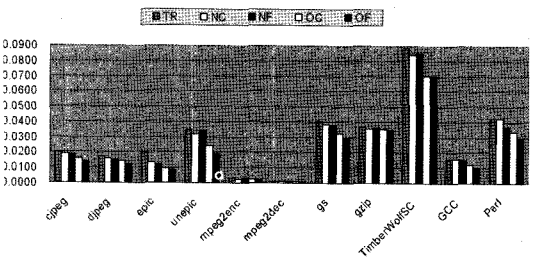


그림 6 L_1 캐시 미스율

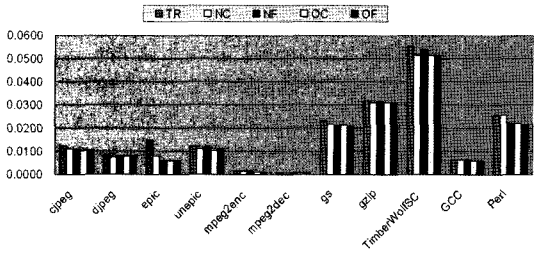


그림 7 L_2 캐시 미스율

표 기법에 사용된 2MB크기의 테이블에 비하여 훨씬 작기 때문에 워킹셋의 크기가 필터용 테이블의 범위를 벗어나므로 성능저하가 발생한 것으로 판단된다.

그림 7은 5개의 구조별로 L_2캐시의 캐시 미스율을 측정한 것이다. L_1캐시 미스율을 측정한 결과와 마찬가지로 OC와 OF구조에서의 L_2캐시 미스율이 각각 NC와 NF구조의 L_2캐시 미스율에 비하여 전반적으로 낮은 것을 알 수 있었다. 그러나 OC와 OF구조간의 비교 또는 NC와 NF구조간의 L_2캐시 미스수에 대한 비교에서는 상호간에 우열을 가리기 어려웠다. 그 이유는 레벨 2 캐시 선인출기에 사용된 상관예측표 기법과 필터 기법의 성능이 벤치마크에 따라 변화하기 때문인 것으로 판단된다.

4.2.3 커버리지 분석

커버리지란 아무런 선인출기를 채용하지 않은 구조(TR)의 L_1캐시 미스를 기준으로 선인출기를 채용한 구조에서의 L_1캐시 미스율의 변화 정도를 나타낸다. 따라서 어떤 캐시구조의 커버리지가 크다는 것은 TR구조에 비하여 선인출이 그만큼 효과가 있다는 뜻이 된다. 그림 8은 선인출기를 채용한 4가지 캐시구조의 커버리지를 측정한 결과이다.

그림 8에서 알 수 있듯이 계층구조의 캐시 선인출기를 채용한 구조가 레벨 2 캐시 선인출기만 채용한 경우에 비하여 성능이 우수함을 알 수 있었다. OC와 OF구조간에는 약간의 차이를 보이는데 이는 벤치마크별로 상관예측기법과 필터 기법의 선호도가 다르기 때문인

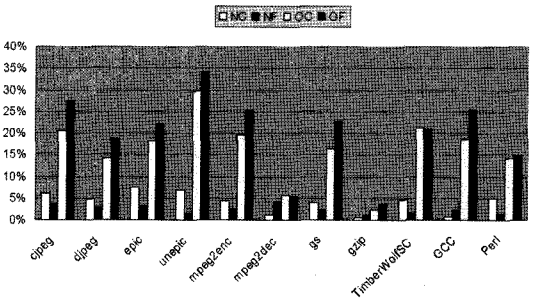


그림 8 커버리지

것으로 분석된다.

4.2.4 선인출 정확도 분석

선인출 캐시 L_p캐시에 적재된 선인출 블록이 교체되기 전에 참조되어야만 선인출의 효과가 있다. L_p캐시에 적재된 블록의 참조성공률을 측정함으로써 여러 가지 캐시구조의 성능을 비교하였다. 그림 9는 선인출 정확도를 측정한 것이다.

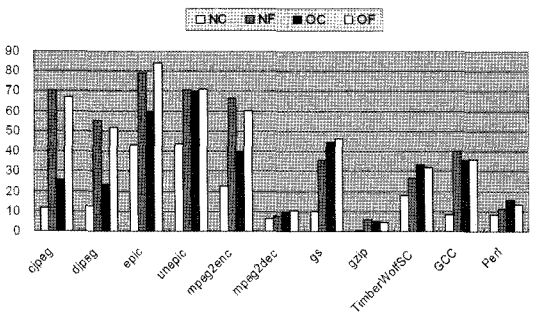


그림 9 선인출 정확도

그림 9는 각 캐시 구조에서의 선인출 정확도를 보여준다. 미디어 응용 프로그램에서 필터 기법이 상관 예측표의 구조보다 더 높은 정확도를 갖는 것을 보여준다. 이는 OBL알고리즘이 스트리밍 참조특성을 갖는 멀티미디어 프로그램 수행에 적합함을 보여주는 것이다. 한편 SPEC 응용 프로그램에서는 응용 프로그램 특성으로 인하여 상관 예측표 기법을 이용한 NC구조와 OC구조가 다른 구조들에 비해 높은 정확도를 보여준다.

그림 9에서 OC, OF의 경우 정확도가 오히려 떨어지는 경우를 볼 수 있다. 그러나 선인출 캐시의 정확도가 높은 것이 성능저하를 초래한다고 볼 수 없다. 그 이유는 L_1캐시와 별개로 선인출 캐시를 채용하는 것은 L_1캐시의 캐시오염을 전혀 유발하지 않기 때문이다. 또한 미디어 데이타를 다루는 벤치마크의 경우에는 L_p캐시의 선인출 정확도가 60%를 상회함을 알 수 있다. 그러나 mpegdec 및 SPEC의 경우에는 L_p캐시의 정확도가 50%미만이었다.

NC와 OC구조에서 레벨 2 캐시 선인출기로 채택한 상관예측표 기법은 미디어 프로그램에서 그 효율성이 필터 기법보다 떨어지기 때문에 정확도가 낮게 나옴을 볼 수 있다. 따라서 이 구조에서 레벨 1 캐시 선인출기를 채용하였을 때 모든 경우에서 정확도가 증가하고 커버리지가 증가함을 볼 수 있다. 그러나 레벨 2 캐시 선인출기로 필터 기법을 채택한 NF와 OF구조는 레벨 1 캐시 선인출기 채택으로 인한 과도한 선인출로 인하여 커버리지는 증가하지만 정확도가 상대적으로 떨어짐을 알 수 있다.

5. 결 론

이 논문에서는 기존의 L_2캐시 선인출기만 있던 계층적 캐시구조에 L_1캐시 선인출기를 도입한 캐시구조를 제안하였다. 제안한 구조를 대상으로 여러 가지 벤치마크를 실행시켜 본 결과 L_1캐시 선인출기와 L_2캐시 선인출기를 모두 적용한 구조, 즉 본 논문에서 제안하는 구조를 채용한 CPU가 모든 벤치마크에 대하여 가장 빠른 계산이 가능함을 보여주었다. 이는 L_1캐시 선인출기에 OBL과 같은 간단한 기법을 적용하면 L_2캐시와 L_1캐시 간의 대역폭을 늘린 효과가 있음을 입증하는 것이다. 이에 대하여 미디어 벤치마크와 SPEC벤치마크의 경우에 동일한 결과를 보여주었다. 이는 계층적 캐시구조에서 레벨 1 캐시 선인출기의 효과는 미디어응용프로그램이나 일반 계산용 응용프로그램을 모두 처리하는 범용 프로세서에 적용될 수 있는 기법임을 알려준다.

11개의 벤치마크 프로그램에 대한 실험을 통하여 레벨 1 캐시 선인출기와 레벨 2 캐시 선인출기로 된 계층적 캐시구조가 레벨 2 캐시 선인출기만 채용한 구조에 비하여 성능이 우수하였음을 알 수 있었다. 따라서 간단한 레벨 1 캐시 선인출기로 성능향상을 최대 19% 얻을 수 있었다. 따라서 계층 구조로 구성된 캐시장치에 각 계층별로 선인출기를 채용하면 메모리 참조지연을 더욱 줄일 수 있는 프로세서의 개발이 가능할 것이다.

참 고 문 헌

[1] A. Grama, A. Gupta, G. Karapis and V. Kumar, *Introduction to Parallel Computing, 2nd Edition*, Addison Wesley, 2003.

[2] J. Fritts, "Multi-Level Memory Prefetching for Media and Streaming Processing," *Proceedings International Conference on Multimedia and Expo*, 2002.

[3] J. L. Bear and W. H. Wang, "Architectural Choices for Multi-level Cache Hierarchies," *Proceedings 16th international Conference on Parallel Processing*, pp.258-256, 1987.

[4] H. J. Moon, J. N. Jeon and S. I. Kim, "Design of A Media Processor Equipped with Dual Cache," *Journal KISS*, Vol.29, No.9, pp.573-581, October, 2002.

[5] N. B. Gaddis, J. R. Butler, A. Kumar and W. J. Queen, "A 56-entry instruction reorder buffer, Solid-State Circuits Conference," *Digest of Technical Papers. 43rd ISSCC.*, pp.212-213, 447, February, 1996.

[6] D. Joseph and D. Grunwald, "Prefetching Using Markov Predictors," *Proceedings 24th Inl. Symp. Computer Architecture*, pp.252-263, June, 1997.

[7] X. Zhang and H. S. Lee, "A hardware-based

cache pollution filtering mechanism for aggressive prefetches," *Proceedings 2003 International Conference on Parallel Processing*, pp.286 - 293, 6-9 October, 2003.

[8] A. Smith, "Sequential Program Prefetching in Memory Hierarchies," *IEEE Computer*, 11(2):7-21, 1997.

[9] N. P. Jouppi, "Improving Direct-mapped Cache Performance by the Addition of a Small Fully associative Cache and Prefetch Buffers," *Proceedings 17th Annual International Symposium on Computer Architecture*, pp.364-373, May, 1990.

[10] T. Horel and G. Lauterbach, "UltraSPARC-III: Designing Third-generation 64-bit Performance," *IEEE Micro*, Vol.19, No.3, pp.73-85, May, 1999.

[11] T. F. Chen and J. L. Baer, "Effective Hardware-Based Data Prefetching for High Performance Processors," *IEEE Transactions on Computers*, 44(5):609-623, May, 1995.

[12] Y. S. Jeon, H. J. Moon, J. N. Jeon and S. I. Kim, "A Hardware Cache Prefetching Scheme for Multimedia Data with Intermittently Irregular Strides," *KIPS Architecture*, Vol.31, No.11, pp.0658-0672, 2004.

[13] K. K. Chan, C. C. Hay, J. R. Keller, G. P. Kurpanek, F. X. Schumacher and J. Zheng, "Design of the HP PA 7200 CPU," *Hewlett-Packard Journal*, Vol.47, No.1, pp.25-33, February, 1996.

[14] ___, *Pentium Processor User's Manual*, Vol.1 Pentium Processor Databook, Intel, 1993.

[15] ___, *IA-32 Intel® Architecture Software Developer's Manual*, Vol. 1 Basic Architecture, Intel, 2004.

[16] M. Denamn, "PowerPC 604," *Hot Chips VI*, pp.193-200, 1994.

[17] O. Mutlu, H. S. Kim, D. N. Armstrong and Y. N. Patt, "Cache Filtering Techniques to Reduce the Negative Impact of Useless Speculative Memory References on Processor Performance," *Computer Architecture and High Performance Computing*, SBAC-PAD 2004. 16th Symposium on, 27-29, pp.2-9, October, 2004.

[18] S. J. Kim, *The Cache Structure of A General Purpose Processor with Media Processing Capabilities*, Ph. D. Thesis, Dept. of Computer Science, Chungbuk National University, February, 2005.

[19] Y. K. Ju, J. N. Jeon and S. I. Kim, "Performance Improvement of A Processor with Independent Media Cache," *KIPS Architecture*, Vol.10, No.02, pp.0143-0146, 2003.

[20] J. H. Lee, et. al., "An Intelligent Cache System with Hardware Prefetching for High Performance," *IEEE Transactions on Computers*, 5(5), pp.607-617, 2003.

[21] Y. Solihin, J. Lee and J. Torrellas, "Correlation prefetching with a user-level memory thread," *IEEE Transactions on Parallel and Distributed Systems*, Vol.14, pp.563-580, June, 2003.

[22] A. Srivastava and A. Eustace, ATOM: A System for Building Customized Program Analysis Tools, *Proceedings ACM SIGPLAN 94*, 196-205, 1994.

[23] J. Hennessy, D. Citron, D. Patterson and G. Sohi, "The use and abuse of SPEC: An ISCA panel," *IEEE Micro*, Vol.23, pp.73-77, July-August, 2003.

[24] H. S. Stone, *High-Performance Computer Architecture*, Addison Wesley, 1993.

[25] J. R. Goodman, *Cache Consistency and Sequential Consistency*, Technical Report TR-1006, University of Wisconsin-Madison, February, 1991.

어 봉 용

1998년 2월 충북대학교 컴퓨터과학과(이학사). 2005년 8월 충북대학교 전자계산학과(이학석사). 관심분야는 데이타 선인출, 병렬 컴퓨터구조, 임베디드 소프트웨어, 이기종 분산처리

주 영 관

1999년 2월 청주대학교 컴퓨터정보공학과(공학사). 2004년 2월 충북대학교 전자계산학과(이학석사). 2004년 3월~현재 충북대학교 전자계산학과 박사과정. 관심분야는 데이타 선인출, 병렬 컴퓨터구조, 모바일 컴퓨팅, 이기종 분산처리

전 중 남

연세대학교 전자공학과. 연세대학교 전자공학과 석사. 1990년 연세대학교 전자공학과 박사 졸업. 1990~현재 충북대학교 전기전자컴퓨터공학부 교수. 1996년~1998 Texas A&M 방문연구원. 관심분야는 컴퓨터구조, 임베디드 시스템

김 석 일(교신저자)

1975년 서울대학교 전기공학과(공학사) 1975년~1990년 국방과학연구소 선임연구원. 1989년 미국 North Carolina주립대학(공학박사). 1990년~2004년 충북대학교 전기전자컴퓨터공학부 학부장. 2004년 9월~현재 유비쿼터스바이오정보기술연구센터 연구소장. 관심분야는 병렬 컴퓨터구조, 슈퍼컴퓨팅, 병렬처리 언어, 이기종 분산처리, 시각장애인 사용자 인터페이스