미참조 선인출 블럭 캐슁에 의한 효율적인 명령어 선인출 기법

An Efficient Instruction Prefetch Scheme using Non - Referenced Prefetched Block Caching

저자 (Authors)
박기호, 한탁돈, 김신덕
Park Gi-Ho, Han Tack-Don, Kim Shin-Dug

출처 (Source)
(구)정보과학회논문지 22(8), 1995.8, 1263-1274 (12 pages)
JOURNAL OF THE KOREA INFORMATION SCIENCE SOCIETY 22(8), 1995.8, 1263-1274 (12 pages)

발행처 (Publisher)
한국정보과학회
KOREA INFORMATION SCIENCE SOCIETY

URL
http://www.dbpia.co.kr/Article/NODE00606412

APA Style
박기호, 한탁돈, 김신덕 (1995). 미참조 선인출 블럭 캐슁에 의한 효율적인 명령어 선인출 기법. (구)정보과학회논문지, 22(8), 1263-1274.

미참조 선인출 블럭 캐싱에 의한 효율적인 명령어 선인출 기법

(An Efficient Instruction Prefetch Scheme using Non-Referenced Prefetched Block Caching)

박 기 호 † 한 탁 돈 †† 김 신 덕 †††
(Park Gi-Ho) (Han Tack-Don) (Kim Shin-Dug)

요 약 메모리의 성능이 전체 시스템 성능에 미치는 영향이 증가됨에 따라 마이크로 프로세서를 사용하는 시스템에서 효율적인 캐쉬 메모리의 구성이 더욱 중요하게 되었다. 본 논문에서는 메모리 대역에 제한을 가지고 있는 마이크로 프로세서를 기반으로 하는 시스템에서 순차 블럭과 목표(target)블럭을 모두 선인출하는 명령어 선인출 기법을 제안한다. 메모리 대역의 제한을 완화하기 위해서 미참조 선인출(Non-Referenced Prefetch, NRP)캐쉬라는 새로운 개념의 캐쉬를 제안하였다. 미참조 선인출 캐쉬는 캐쉬 접근실패를 감소시키고 순차블럭과 목표블럭을 모두 선인출하는데 요구되는 부가적인 메모리 트래픽을 감소시키는 역할을 한다. 새로운 개념의 캐쉬인 미참조 선인출 캐쉬는 기존의 캐쉬가 CPU에 의해 참조된 블럭을 저장하는데 비해서, 선인출되었으나 CPU에 의해서 참조되지 않은 블럭을 저장하는데 사용된다.미참조 선인출 캐쉬를 사용함으로써 실제로는 하나의 블럭을 선인출하나 두개의 블럭을 모두 선인출한 것과 같은 효과를 얻을 수 있다. 특히 제안한 선인출 기법은 큰 메모리 지연을 갖는 시스템에서 이를 효과적으로 극복하기 위해 제안된 전진 선인출 기법에 적용하면 더욱 큰 성능의 향상을 얻을 수 있다. 제안한 선인출 기법의 성능을 측정하기 위해서 트레이스 구동 시뮬레이션(trace-driven simulation)을 수행하였다.성능평가에 사용한 응용 프로그램에 대해, 미참조 선인출 캐쉬는 두 블럭을 모두 선인출하는데 필요한 부가적인 메모리 트래픽의 50% ~ 112%를 감소시킴을 알 수 있었다 또한 제안한 명령어 선인출 기법을 사용하는 캐쉬 시스템은 이전의 다른 선인출 기법을 사용하는 캐쉬 시스템에 비해 향상된 메모리 접근 시간을 가짐을 확인하였다

Abstract As the impact of the memory system on system performance increases, the design of an efficient cache memory system becomes more important. A new instruction prefetch mechanism is proposed, which prefetches both the sequential and target blocks when a processor has a limited memory bandwidth, such as in the case of microprocessors. In this paper, a new concept of the cache, the NRP(Non-Referenced Prefetch) cache, is proposed to alleviate the memory bandwidth restriction. The proposed NRP cache is used in storing prefetched blocks, which have not been referenced by the CPU and would be otherwise discarded in previous prefetch mechanisms. However, because it is very likely that these prefetched blocks will be subsequently referenced by the CPU, an effective way is to store these blocks in the separate NRP cache By using the NRP cache, the performance improvement of prefetching both paths can be achieved while prefetching only one block of either path. Performance evaluation of the proposed prefetch mechanism is carried out through trace-driven simulation, and the results show that this approach provides an improvement in memory access time over other prefetch methods In particular, the NRP cache is more effective in lookahead prefetch schemes those aim at hiding longer memory latency Also, it is shown that the NRP cache eliminates 50% ~ 112% of the additional memory traffic required to prefetch both paths.

1. 서 론

CPU(Central Processing Unit)와 메모리간의 성능격차가 심화됨에 따라 효율적인 메모리 구성이 전체시스템의 성능에 중요한 영향을 미치게 되었다. 지속적으로 증가하는 CPU와 메인 메모리의 성능격차를 효과적으로

† 준 회 원 연세대학교 컴퓨터과학과
†† 종신회원 · 연세대학교 컴퓨터과학과 교수
††† 정 회 원 · 연세대학교 컴퓨터과학과 교수
논문접수 1994년 9월 12일
심사완료 1995년 6월 3일

극복하기 위해서 대부분의 컴퓨터 시스템에서는 캐쉬 메모리를 사용하고 있으며 최근의 고성능 마이크로 프로세서들은 수퍼스칼라(superscalar), 수퍼파이프라이닝(superpipelining)등 명령어 발생 빈도(instruction issue rate)를 증가시키는 기법을 채택하고 있어서 명령어 캐쉬의 성능이 매우 중요하게 되었다.

VLSI(Very Large Scale Intergration) 기술 발달에 따라 집적도가 향상되어 마이크로 프로세서가 칩내에 대용량의 캐쉬를 가지게되어 캐쉬접근실패를 구성하는 초기접근실패(compulsory miss), 충돌접근실패(conflict miss), 용량접근실패(capacity miss)중 초기접근실패가 차지하는 비율이 증가하게 되어 이를 효과적으로 감소시키는 선인출 기법에 대한 요구가 증가하고 있다[1].

본 연구에서는 미참조 선인출(Non-Referenced Prefetch,NRP) 캐쉬와 목표 주소 예측(target address prediction) 기법을 이용해서 순차블럭과 목표블럭의 두 블럭을 모두 선인출하는 효과적인 명령어 선인출 기법을 제안한다. 미참조 선인출 캐쉬는 선인출되었으나 CPU에 의해서 참조되지 않은 블럭을 저장하는 캐쉬이다. 기존의 기법에서는 이러한 블럭을 저장하지 않았으나, 이 블럭은 후에 CPU에 의해서 참조될 가능성이 많은 블럭이므로 분리된 미참조 선인출 캐쉬에 저장함으로써 이후의 참조에 의한 성능향상을 기할 수 있다. 미참조 선인출 캐쉬는 캐쉬접근실패를 줄이는 역할을 할뿐 아니라 저장하고 있는 블럭을 다시 인출하는데 필요한 메모리 트래픽을 줄이는 작용을 한다. 이러한 미참조 선인출 캐쉬에 의해서 실제로는 하나의 블럭당 하나의 블럭만을 선인출하나, 메모리 대역의 제한이 비교적 적은 수퍼컴퓨터 등에서만 수행이 가능한 순차, 비순차(non-sequential)경로를 모두 선인출하는 기법에 가까운 성능 향상을 얻을 수 있다. 또한 미참조 선인출 캐쉬는 메모리 지연(latency)이 큰 시스템에서 이를 극복하기 위해 제안된 전진 선인출 기법에 적용하면, 이 기법의 단점인 낮은 선인출 정확도를 보완하여 전진 선인출에 의한 성능향상을 증대시킨다. 전진 선인출 기법은 현재 참조되고 있는 블럭 바로 다음에 참조될 블럭이 아닌 전진의 정도만큼 후에 참조될 블럭을 선인출 하는 기법인데, 전진의 정도가 커짐에 따라 선인출 정확도가 감소하게 된다[8]. 미참조 선인출 캐쉬를 사용하는 제안한 기법은 순차와 비순차 경로를 모두 선인출하므로 낮은 선인출 정확도에 의한 성능저하를 효과적으로 완화할 수 있다.

제안한 선인출기법의 성능평가는 트레이스 구동 시뮬레이션(trace-driven simulation)을 통해서 수행하였다. 선인출 기법의 성능을 보다 정확히 반영하기 위하여 메모리 접근 지연시간(memory access delay time)을 성능 척도로써 사용하였으며, 제안한 선인출 기법이 기존의 선인출 기법에 비해 향상된 메모리 접근 시간을 가짐을 확인하였다. 또한 성능평가를 통하여 두 경로를 모두 선인출하는데 부가적으로 요구되는 메모리 트래픽의 대부분이 NRP 캐쉬에 의해서 제거됨을 알 수 있었다.

본 논문의 구성은 다음과 같다. 2장에서는 기존의 선인출 기법에 관한 연구에 대해 알아본다. 캐쉬의 구성과 제안한 선인출 기법의 작동방식은 3장에서 설명한다. 4장에는 성능평가에 사용된 시뮬레이션 방법, 트레이스 생성과 결과를 기술하였다. 5장에서는 결론 및 향후 연구방향에 대해서 언급하였다.

2. 관련 연구

효과적인 메모리 시스템의 설계가 전체 시스템 성능에 미치는 영향이 중대함에 따라서 캐쉬 메모리에 관한 많은 연구가 수행되었다. 이와 함께 명령어 선인출에 대한 연구역시 많이 진행되었다[2,3,4,6,7,8,12,9]. 이들 기법들 중 대부분은 순차 선인출(sequential prefetch) 기법에 관한 연구였다[2,3,4,6,7]. 최근에 와서 이전의 수행결과를 이용해서 선인출할 블럭을 결정하는 선인출 기법이 제안되었다[8,12,9]. 본 절에서는 이전에 연구된 선인출 기법을 소개하고, 이들의 장단점을 제시한다.

2.1 참조의 순차성에 기반한 기법

메모리 접근의 순차성을 이용해서 캐쉬의 성능을 향상시키고자 하는 시도로써 블럭 크기의 증가와 순차 선인출 기법을 들 수 있다. 이들 기법은 메모리 참조의 순차성을 반영하여 어느정도의 성능의 향상을 얻을 수 있다. 그러나 블럭 크기의 증가는 캐쉬 접근 실패시의 인출 시간이 증가하며, 메모리 트래픽의 증가, 캐쉬 오염(cache pollution) 등을 일으킬 수 있다. 또한 블럭 크기는 메모리 지연(latency)과 전송률(transfer rate)등 시스템의 구조적인 특성(architectural feature)에 영향을 받으므로 단순히 선인출 기능의 강화를 위해서 증가시킬 수는 없다[13].

순차 선인출은 위에서 지적한 단순한 블럭크기의 증가에 의한 단점은 어느정도 해결할 수 있으나, 분기 명령 등에 의해서 비 순차적인 블럭을 참조하는 경우에는 좋은 성능을 얻을 수 없다.

2.2 분기 명령을 고려한 선인출 기법

분기 명령에 의한 선인출의 성능 저하를 극복하기 위해서 분기 명령을 고려한 선인출 기법이 제안되었다. 목표 선인출 기법은 무조건 분기뿐만 아니라 조건 분기명령도 이전의 수행경로를 따르는 경향이 많음에 기반하고

있다[10,11]. 목표 선인출 기법은 이전의 수행경로를 예측 테이블에 저장해서, 이후에 선인출 요구(prefetch request)를 발생시킬 때 이 정보를 이용해서 선인출 블럭을 결정한다. 즉, 어느 블럭 A가 이전의 수행에서 어떤 블럭 B를 참조한 경우에는 이후에 블럭 A가 참조될 때 블럭 B를 선인출 하는 기법이다. 따라서 이전의 수행이 순차블럭을 참조했으면 순차블럭을, 비순차블럭을 참조했으면 비순차블럭을 선인출한다. 분기명령의 성질을 이용하고 있기 때문에 순차선인출에 비해서 높은 선인출 정확도와 우수한 성능을 보인다[8,9].

목표 선인출 기법은 분기명령의 성질을 이용하고 있기 때문에 순차 선인출에 비해서 높은 선인출 정확도를 보이나 분기명령이 항상 이전의 경로를 따르는 것은 아니고, 또한 분기명령이 순차, 비순차경로를 번갈아 수행하는 경우에는 목표 선인출 기법으로는 큰 성능향상을 기대할 수 없다. 이러한 단점을 극복하기 위해서 Smith 등에 의해서 순차 선인출 기법과 목표 선인출 기법을 혼합한 형태의 복합 선인출 기법이 제안되었다[12]. 이러한 복합 선인출 방식에 의해서 순차 선인출과 목표 선인출에 의해서 얻을 수 있는 성능의 향상을 누적해서 얻을 수 있음을 보이고 있다[12]. 그러나 복합 선인출 기법은 메모리 대역폭의 제한이 비교적 적은 수퍼컴퓨터 등에서만 수행이 가능한 선인출 기법이다. 메모리 대역의 제한이 많은 마이크로 프로세서를 사용하는 시스템에서는 순차블럭과 목표블럭을 모두 선인출하는 기법은 수행이 불가능하다고 생각되었다[2,9].

본 논문에서는 제한된 메모리 대역을 갖는 마이크로 프로세서를 기반으로 하는 시스템에서 순차블럭과 목표블럭을 선인출 할 수 있도록 하기 위해 미참조 선인출(Non-Referenced Prefetch) 캐쉬를 제안하였다. 미참조 선인출 캐쉬는 선인출 되었으나 CPU에 의해 참조되지 않은 블럭을 저장해서 이후의 캐쉬 접근 실패를 줄이고 선인출시에 캐쉬에 존재하지 않는 블럭만을 선인출함으로써, 실제로는 하나의 블럭만을 선인출 하나 두 경로를 모두 선인출한 효과를 얻도록 한다. 제안된 선인출 기법을 수행하기 위한 캐쉬의 구성과 선인출 기법의 작동 방식은 3장에서 자세히 설명하겠다.

3. 제안한 캐쉬 시스템 구성 및 작동 방식

제안한 선인출 기법을 수행하기 위해서 NRP 캐쉬를 비롯해서 선인출 버퍼, 예측 테이블(prediction table) 등이 수정되었다. 이 장에서는 선인출 기법을 수행하기 위한 캐쉬의 구성과 그들의 작동 방식에 대해서 자세히 설명한다.

3.1 제안한 캐쉬 시스템의 구성

제안한 선인출 기법을 수행하기 위한 캐쉬 시스템의 구성은 <그림 1>에 나타나 있다.

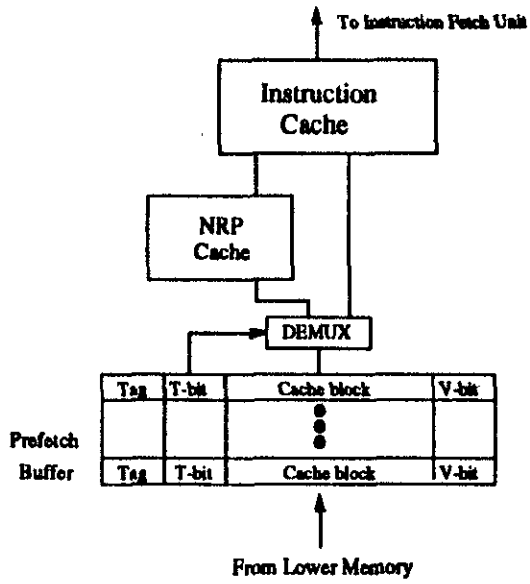


그림 1 NRP 캐쉬를 갖는 캐쉬 시스템 구조

캐쉬 시스템은 명령어 캐쉬(I-cache), 미참조 선인출(NRP) 캐쉬와 선인출 버퍼로 구성된다. 또한 이후에 선인출할 블럭을 결정하기 위한 예측 테이블이 존재한다. 이들 각 구성요소들에 대한 자세한 내용은 다음의 소절에서 설명하겠다.

3.1.1 명령어 캐쉬(I-Cache)

명령어 캐쉬의 구성과 관리는 기존의 명령어 캐쉬와 같다. 본 연구에서는 빠른 접근 시간과 작은 하드웨어 비용을 갖는 직접 사상 명령어 캐쉬와, 명령어 캐쉬와 데이터 캐쉬가 분리되어 있는 캐쉬 시스템을 가정하였다.

3.1.2 미참조 선인출(Non-Referenced Prefetch ,NRP) 캐쉬

본 논문에서는 메모리 대역의 제한을 완화하고, 캐쉬 접근 실패를 감소시키기 위해서 미참조 선인출(NRP) 캐쉬를 제안하였다. 미참조 선인출 캐쉬는 앞에서 설명한 바와 같이 CPU에 의해서 참조되지 않은 선인출 블럭을 저장하는데 사용한다. 미참조 선인출 캐쉬의 크기와 연관도는 명령어 캐쉬와는 무관하게 구성될 수 있다. 본 논문에서는 명령어 캐쉬의 1/8 크기와 같은 연관도

를 갖는 미참조 선인출 캐쉬를 선택하였다. 미참조 선인출 캐쉬의 크기를 변화시키면서 성능을 평가한 결과 1/8크기를 갖는 미참조 선인출 캐쉬가 부가적인 하드웨어 부담에 비해 큰 성능향상을 보이기 때문에 이러한 선택을 하였다.

3 1.3 선인출 버퍼(Prefetch Buffer)

선인출 버퍼는 8개의 엔트리를 가지며, 이들은 완전 연관도를 갖도록 관리된다. 선인출 버퍼는 태그 필드(tag field), T-비트(transfer-bit)필드, 캐쉬 블럭, V-비트(valid-bit)필드로 구성된다. T-비트는 해당하는 선인출 버퍼의 엔트리가 NRP캐쉬로 이동될 것인가를 나타내기 위한 필드이다. T-비트는 해당 엔트리가 선인출될 때 1로 세트(set)된다. CPU에 의해서 해당 엔트리가 참조된 경우에는 T-비트는 0으로 리셋(reset)된다. 선인출 버퍼 엔트리가 교체될 때 T-비트가 1인 경우에는 해당 엔트리의 캐쉬 블럭을 미참조 선인출 캐쉬로 이동한다. T-비트가 0인 경우에는 미참조 선인출 캐쉬로 이동하지 않는데 이는 이 블럭은 이미 CPU에 의해서 참조되었기 때문이다. 이러한 방식에 의해서 미참조 선인출 캐쉬와 명령어 캐쉬는 공통된 블럭을 가지지 않는다.

3.1.4 예측 테이블(Prediction Table)

이전의 수행경로를 저장하기 위한 예측 테이블의 구조는 <그림 2>에 나타나 있다.

이것은 이전의 연구에서 제안한 예측 테이블과 유사한 구조를 갖는다[12,9]. 예측 테이블의 각 엔트리는 다음과 같은 세 개의 필드로 구성되어 있다.

● 현재 블럭 주소(Current Block Address) : 현재 CPU에 의해서 참조되는 블럭의 주소를 저장하는 필드이며, 이 필드가 예측 테이블에서 각 엔트리를 검색하는 키로써 이용된다.

● 목표 블럭 주소(Target Block Address) : 이전의 수행에서 현재 블럭이후에 참조된 다음 블럭의 주소가 저장되는 필드이다. 이 필드에는 순차블럭이 아닌 블럭의 주소만이 저장되는데 순차블럭의 주소는 현재 블럭의 주소를 이용해서 쉽게 계산할 수 있기 때문이다

● 이전 수행 정보(History Information) : 이전의 수행의 결과가 순차적인 수행이었는가 그렇지 않은가를 나타내는 비트 필드이다.

예측 테이블은 기존의 BTB(Branch Target Buffer)에서 분기 주소(branch address), 분기 목표 주소(branch target address)와 1 비트의 분기 예측 비트(prediction bit)를 갖는 구조와 유사하다[14]. 단지 논문의 예측 테이블은 분기 명령의 주소와 분기 목표 주소 대신 분기 명령과 분기 명령의 목표 주소를 포함하는 캐쉬 블럭의 주소를 저장하는 것이 일반적인 BTB와 다르다.

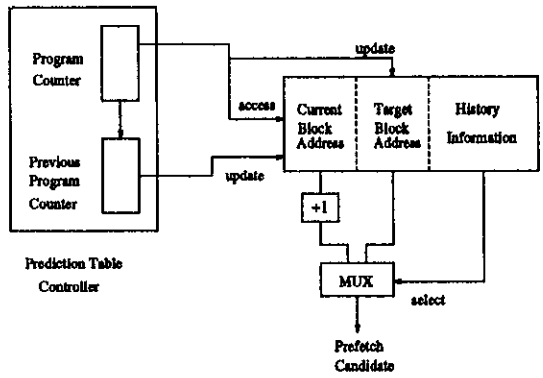


그림 2 이전 수행 정보를 저장하는 예측 테이블의 구조

예측 테이블은 어떠한 블럭이후에 참조된 블럭의 주소를 저장한다. 이를 위해서 CPU의 참조가 새로운 캐쉬 블럭으로 이동한 경우에 현재의 PC(Program Counter)와 이전 PC의 블럭 주소를 이용해서 수정된다. 새로운 블럭으로 CPU의 참조가 이동되었을 때, 이전 PC의 블럭 주소는 예측 테이블의 현재 블럭 주소 필드에 저장되며 현재 PC의 블럭 주소는 예측 테이블의 목표 블럭 주소 필드에 저장된다. 예측 테이블의 목표 블럭 주소 필드는 CPU 참조가 비순차적인 블럭으로 이동되었을 경우에만 갱신된다. 이는 앞에서 언급한 바와 같이 순차블럭의 주소는 현재블럭 주소에 1을 더함으로써 쉽게 계산될 수 있으며, 목표 블럭 주소가 두 경로를 모두 선인출 하는데 요구되기 때문이다 이전 수행정보 필드는 이전의 수행이 순차블럭을 참조하였는가 그렇지 않은가를 나타내며 예측 테이블을 이용해서 선인출 블럭을 결정할 때 이 필드를 이용해서 순차블럭과 목표블럭 중에 어떤 블럭을 우선적으로 선인출 할 것인가를 결정한다. 본 논문에서 사용한 예측 테이블은 직접 사상 방법을 사용하고 명령어 캐쉬의 블럭 수와 동일한 수의 예측 테이블 엔트리를 갖도록 구성하였다.

3.2 작동 방식

본 연구에서 제안한 선인출 기법은 CPU 참조가 어떤 블럭 A에서 새로운 블럭 B로 이동되었을 때 예측 테이블을 갱신하고 선인출을 수행한다. 예측 테이블의

갱신과 선인출은 다음과 같이 수행된다.

● 예측 테이블을 CPU 참조의 이동에 따라 갱신한다.

1. CPU 참조가 블럭 A에서 비순차블럭 B로 이동했을 경우

— 예측 테이블에서 블럭 A에 해당하는 엔트리를 검색한다.

-- 만약 해당 엔트리가 예측 테이블에 존재하지 않으면, A에 해당하는 엔트리를 만든다.

— 블럭 B의 주소를 블럭 A에 해당하는 예측 테이블 엔트리의 목표 주소 필드에 저장한다

— 이전 수행 정보 필드를 비순차경로로 세트한다.

2. CPU 참조가 블럭 A에서 순차블럭 B로 이동했을 경우

— 예측 테이블에 블럭 A에 해당하는 엔트리가 존재하면 이전 수행 정보 필드를 순차경로로 세트한다.

— 블럭 A에 해당하는 예측 테이블 엔트리가 존재하지 않으면, 예측 테이블을 갱신하지 않는다. 이는 예측 테이블에 해당 엔트리가 존재하지 않는 경우에는 기본적으로(default) 순차 선인출을 수행하기 때문이다

● 선인출은 선인출될 후보 블럭을 결정한 후에 수행된다

1. CPU에 의해 참조되는 블럭에 해당하는 엔트리가 예측 테이블에 존재하지 않으면 순차블럭을 선인출 한다. 이 경우에 선인출 참조(prefetch lookup)를 통해서 선인출할 블럭이 이미 캐쉬(명령어, NRP)나 선인출 버퍼에 존재하면 블럭을 선인출하지 않는다.

표 1 선인출 후보(prefetch candidate) 블럭 결정

이전수행결과		순차참조시	비순차참조시
선인출 후보블럭	1	Cur_blk_addr+1	Tar_blk_addr
	2	Tar_blk_addr	Cur_blk_addr + 1

2 현재 CPU에 의해서 참조되는 블럭에 해당하는 엔트리가 예측 테이블에 존재하면 이전 수행 정보를 이용해서 우선적으로 선인출할 블럭을 결정한다 이전 수행 정보에 따라 선인출할 후보 블럭을 〈표 1〉과 같이 설정한다. 이는 가장 최근의 수행 결과를 이용하여 우선적으로 선인출 할 블럭을 결정하는 방식으로 바로 이전 수행 결과를 이용하여 예측을 하는(prediction on last outcome) 기법이다[15]

(a) 선인출 후보 블럭 1이 캐쉬나 선인출 버퍼에 존재하지 않으면, 선인출 후보 블럭을 하위 메모리로부터 선인출 한다. 선인출하는 블럭에 의해서 교체되는 선인출 버퍼내의 블럭은 T-비트의 값에 따라 NRP캐쉬로 이동된다.

(b) 선인출 후보 블럭이 캐쉬나 선인출 버퍼에 이미 존재하면, 선인출 후보 블럭 2에 대한 선인출을 시도한다.

3. 두 경로의 블럭이 모두 이미 캐쉬나 선인출 블럭에 존재하면, 선인출을 수행하지 않는다.

명령어 캐쉬접근실패와 새로운 선인출 요청은 현재 진행되고 있는 선인출보다 우선적으로 처리된다. 이것은 선인출에 의해서 정상적인 메모리 참조가 지연되지 않도록 하기 위함이다. 새로운 선인출 요청이 현재 진행중인 선인출보다 우선권을 갖는 이유는 새로운 선인출 요청이 발생했다는 것은 현재 진행중인 선인출이 쓸모 없는 선인출일 가능성이 많다는 것으로 생각할 수 있기 때문이다.

CPU가 명령어 캐쉬에 있는 블럭을 참조할 경우에는 아무런 지연이 발생하지 않는다. 그러나 CPU가 NRP 캐쉬에 있는 블럭이나 선인출 버퍼에 있는 블럭을 참조하기 위해서는 1 CPU 싸이클의 지연을 갖는다. 이는 [3]에서 언급한 바와 같이 복잡한 선인출 기법을 사용하는 경우에 선인출 버퍼에서 CPU로 바로 블럭을 리턴(return)하는 경우에 CPU 싸이클 시간(cycle time)을 증가시킬 수 있고, CPU 싸이클 시간 증가에 의한 성능의 저하가 선인출에 의한 캐쉬접근실패의 감소보다 전체 시스템의 성능에 보다 큰 영향을 미치기 때문이다[5]. NRP캐쉬의 경우는 명령어 캐쉬와 같은 직접 사상방식을 사용하고, 용량이 작기 때문에 이러한 지연이 요구되지는 않으나, NRP 캐쉬로부터 직접 CPU로 블럭을 리턴하기 위해서는 명령어 인출 경로에 멀티플렉서(Multiplexor)를 두어야 하는데, 이는 역시 CPU 싸이클 시간을 증가시킬 수 있으므로 역시 1 CPU 싸이클의 지연을 갖도록 하였다.

4. 성능평가

제안한 선인출 기법은 트레이스 구동 시뮬레이션을 통해서 성능평가를 수행하였다. 이 장에서는 시뮬레이션에 사용된 트레이스들과 성능평가 결과를 제시한다. 4.1절에서는 시뮬레이션에 사용된 트레이스들과 시스템 변수(system parameter)에 대해서 설명하며, 시뮬레이션 결과는 4.2절에서 제시하였다

4.1 시뮬레이션

트레이스를 생성하는데 사용된 프로그램들은 〈표 2〉에 제시되어 있다.

트레이스는 Spa 툴[16]을 이용하여서 SUN Sparcstation상에서 생성하였으며, 각 프로그램에 대해서 10,000,000개의 명령어 참조를 생성하여 시뮬레이션

에 이용하였다. 고성능의 RISC 프로세서를 기반으로 하는 컴퓨터 시스템을 가정하여 시스템 변수를 선택하였다[9]. 〈표 3〉에는 시뮬레이션에 사용한 시스템 변수들을 제시하였다.

표 2 트레이스 생성에 사용된 프로그램들

사용된 프로그램	프로그램의 성격
gcc	GNU C 컴파일러
espresso	부울함수(Boolean function) 최적화
spice	아날로그회로 시뮬레이터
f77	FORTRAN 컴파일러
latex	타잎 셋터(type setter)

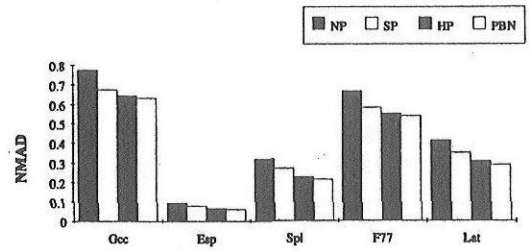
표 3 시뮬레이션에 사용된 시스템 변수들

시스템변수	사용된 값
명령어 캐쉬 크기	8/16/32 킬로바이트
명령어 캐쉬 블럭 크기	16/32 바이트
명령어 캐쉬 연관도	직접사상(direct-mapped)
명령어 참조 성공시 CPI	1
NRP 캐쉬 크기	명령어 캐쉬 크기의 1/8
NRP 캐쉬 연관도	직접사상(direct-mapped)
메모리 지연(latency)	8/16 싸이클
전송률(transfer rate)	1워드(4바이트)/싸이클
예측 테이블의 엔트리 수	명령어 캐쉬의 블럭 수
예측 테이블의 연관도	직접사상(direct-mapped)
예측 테이블을 갱신하는데 요구되는 지연시간	1 싸이클
선인출 버퍼나 NRP 캐쉬를 참조할때 요구되는 지연시간	1 싸이클

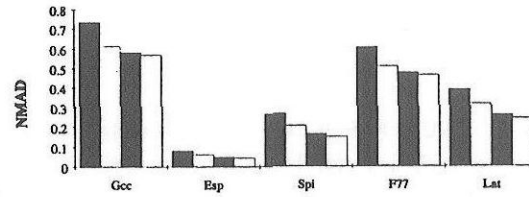
시뮬레이션이 매우 많은 시간을 필요로 하기 때문에 몇몇 캐쉬 변수들을 고정시키고 성능평가를 수행하였다. 특히 본 논문에서 수행한 성능평가에는 직접 사상 캐쉬(direct-mapped cache)만을 고려하였는데, 이는 시뮬레이션 공간의 감소와 함께 마이크로 프로세서에서의 직접 사상 캐쉬의 장점 때문이기도 하다. 직접 사상 캐쉬는 가장 빠른 접근 시간을 가지며, 제어회로등이 간단해서 하드웨어 비용 역시 적다. 또한 캐쉬의 접근 시간에 의해서 CPU 싸이클 시간이 결정되는 RISC 마이크로 프로세서의 경우에는 직접 사상 캐쉬가 많이 채택된다[17].

4.2 성능평가 결과

성능평가는 〈표 3〉에 제시된 시스템 변수들을 가지고 생성된 트레이스를 이용해서 수행하였다. 성능척도로는 메모리 접근 지연 시간을 사용하였다. 대부분의 캐쉬 메모리의 성능척도는 접근 실패율(miss ratio)을 사용하나 선인출의 경우에는 시간(timing)정보가 고려된 메모리 접근 지연시간을 사용하는 것이 보다 정확한 성능평가를 할 수 있다. 선인출 요구가 발생된 후 그 블럭이 선인출 버퍼에 도달하기 전에 CPU에 의해서 참조되게 되면 접근 실패는 아니지만 CPU는 참조할 블럭의 전송이 끝날 때까지 기다려야 한다. 접근 실패율은 이러한 지연을 반영하지 못하여 선인출에 의한 성능향상을 실제보다 과장하게 된다.



(a) 블럭 크기 = 4 워드(16 바이트), 명령어 캐쉬 = 8 킬로 바이트



(b) 블럭 크기 = 8 워드(32 바이트), 명령어 캐쉬 = 8 킬로 바이트

그림 3 정규화된 메모리 접근 지연(NMAD) (메모리 지연 = 8, CPI = 1)

메모리 접근 지연은 메모리 참조로 지연된 싸이클 시간(delayed cycle time)으로 정의된다. 메모리 참조에 의한 전체 시스템의 성능 저하정도를 알아보기 위해서 항상 캐쉬접근성공을 하는 시스템의 수행 시간으로 메모리 접근 지연시간을 나눈 정규화된 메모리 접근 지연(normalized memory access delay, NMAD)을 사용하여 각 선인출 기법을 비교하였다. NMAD는 다음의 식에 의해서 계산된다.

$$NMAD = \frac{\text{특정 선인출 기법의 메모리 접근 지연시간}}{\text{캐쉬 참조가 100\% 성공인 시스템의 수행시간}} \quad (1)$$

〈그림3〉에서는 8 킬로 바이트 명령어 캐쉬를 갖는 시스템의 NMAD를 보이고 있다. 그림의 (a)에서는 블럭의 크기가 4워드(16바이트)인 캐쉬를, (b)에서는 8워드(32바이트)인 캐쉬 시스템의 성능을 보이고 있다. 그림

에서 각 선인출 기법을 나타내는 다음의 약어들을 사용하였다.

- NP : 선인출을 수행하지 않는 시스템(No Prefetch)
- SP : 순차 선인출을 수행하는 캐쉬 시스템. 새로운 블럭이 참조될 때 그 다음블럭을 항상 선인출 한다.(Sequential Prefetch)
- HP : 이전 수행정보를 이용해서 선인출을 수행하는 시스템(Prefetch using History information)
- PBN : NRP 캐쉬를 이용해서 두 경로를 모두 선인출 하는 시스템(Prefetch Both instruction paths with NRP cache)

제안한 NRP 캐쉬를 이용한 선인출 기법의 HP 선인출 기법에 대한 상대적인 성능향상을 측정하기 위해서 상대적 성능 증가(Relative Performance Improvment, RPI)를 정의하여 사용하였다. RPI를 계산하기 의해 다음의 수식들을 정의하였다.

NMAD(HP_m) : m 킬로바이트 명령어 캐쉬를 가지는 HP 기법의 NMAD

NMAD(PBN_m) : m 킬로바이트 명령어 캐쉬를 가지는 PBN 기법의 NMAD

△NMAD(HP_m) : 2m 킬로바이트 명령어 캐쉬와 m 킬로바이트 명령어 캐쉬를 갖는 HP선인출 기법간의 NMAD 차

△NMAD(PBN_m) : m 킬로 바이트의 명령어 캐쉬를 갖는 PBN 기법과 HP 기법의 NMAD 차

상대적 성능 증가(RPI)는 다음의 식에 의해서 계산된다.

$$\triangle NMAD(HP_m) = NMAD(HP_{2m}) - NMAD(HP_m) \quad (2)$$

$$\triangle NMAD(PBN_m) = NMAD(PBN_m) - NMAD(HP_m) \quad (3)$$

$$RPI(PBN_m) = \frac{\Delta NMAD(PBN_m)}{\Delta NMAD(HP_m)} = \frac{NMAD(PBN_m) - NMAD(HP_m)}{NMAD(HP_{2m}) - NMAD(HP_m)} \quad (4)$$

〈표 4〉에 여러 캐쉬 구성에서의 RPI의 값을 제시하고 있다.

표 4 RPI(PBN_m)(상대적 성능 증가) (%) (메모리 지연 = 8, CPI = 1)

	블럭크기 (워드)	gcc	espresso	spice	f77	latex
PBN_8	4	4.4	28.8	9.3	5.4	9.4
	8	7.1	39.2	16.2	6.4	10.0
PBN_{16}	4	4.7	51.4	8.9	11.0	16.9
	8	6.7	67.1	11.7	11.5	20.4

PBN 선인출 기법을 위해서 HP 기법에 비해 추가적으로 요구되는 하드웨어 비용은 약 1/8 정도라 할 수 있다. PBN 기법의 추가적인 하드웨어 비용은 명령어 캐쉬 크기의 1/8 크기인 NRP 캐쉬와 선인출 블럭을 NRP 캐쉬로 이동시키기 위해서 수정된 선인출 버퍼들을 들 수 있다. 그러나 HP 선인출 기법과 같은 개수의 예측 테이블을 사용하는 등의 요소에 의해서 추가적인 하드웨어 비용은 HP 선인출 기법의 약 1/8 정도로 볼 수 있다. 그러므로 PBN 선인출 기법이 하드웨어 비용에 비해 우수한 성능을 보임을 주장하기 위해서는 RPI가 1/8, 즉 12.5%보다 큰 값을 가져야 한다. 그러나 〈그림 3〉과 〈표 4〉에 나타나 있는 RPI를 보면, 기대했던 것에 못 미침을 알 수 있다. 몇몇의 결과에서는 RPI가 12.5%보다 크나, 또 일부의 구성과 트레이스에서는 12.5%에 미치지 못함을 알 수 있다. 이러한 작은 성능 향상은 HP 선인출 기법의 선인출 정확도(prefetch accuracy)가 높기 때문인 것으로 생각할 수 있다. HP 선인출 기법이 선인출 정확도가 높은 경우에는 다른 경로를 선인출 하여 얻을 수 있는 성능 향상이 작기 때문이다.

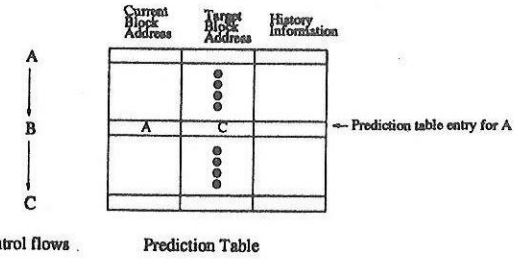


그림 4 전진 선인출을 위한 예측 테이블의 갱신(전진 선인출 정도 = 2)

4.2.1 전진 선인출에의 적용

〈그림 3〉과 〈표 4〉에서 보는 바와 같이 HP 선인출 기법과 PBN 선인출 기법의 다른 기법에 대한 성능의 증가가 작다. 이는 정확한 선인출에 의해서 얻을 수 있는 성능의 증가가 매우 작기 때문이다. 즉 블럭의 처음에 올바른 선인출 요구를 발생시키는 경우에도, 선인출에 의해서 얻을 수 있는 이득은 4 CPU 싸이클(4 워드 블럭의 경우, 8 워드 블럭은 8)에 불과하다. 이에 비해서 하나의 블럭을 인출하는데 필요한 시간은 12 싸이클(4 워드 블럭의 경우, 8 워드 블럭의 경우는 16)이기 때문에 정확한 선인출에 의해서 얻는 성능의 향상이 그리 크지 못하다.

정확한 선인출에 의한 성능 향상을 증가시키기 위해서 사용되는 방식중의 하나가 전진 선인출 기법이다[8]. 전진 선인출 기법은 현재 CPU에 의해서 참조되고 있는 블럭의 d번째 후에 참조될 것으로 예상되는 블럭을 선인출 하는 선인출 기법이다. 여기에서 d는 전진 선인출의 정도(degree of lookahead prefetching)라고 한다. 전진 선인출을 수행하기 위해서는 예측 테이블을 갱신하는 방법 역시 바뀌게 된다. 〈그림 4〉에는 전진 선인출의 정도가 2인 전진 선인출 기법에서 예측 테이블을 갱신하는 방법을 보이고 있다.

CPU의 참조가 블럭 A에서 블럭 B로 그리고 블럭 C로 이동했을 경우에, 블럭 A에 대한 예측 테이블 엔트리의 목표 블럭 주소에는 블럭 C의 주소가 저장된다. 전진 선인출의 경우에도 이전의 목표 선인출 기법에서와 같이 순차 블럭이 참조(여기서는 블럭 i에 대한 i+2블럭)된 경우에는, 이미 그 블럭에 대한 예측 테이블 엔트리가 존재할 때에만 이전 수행 정보 필드를 순차 참조로 갱신하고, 존재하지 않는 경우에는 예측 테이블을 갱신하지 않는다.

표 5 전진 선인출시의 선인출 후보(prefetch candidate) 블럭 결정

이전수행결과		순차참조시	비순차참조시
선인출 후보블럭	1	Cur_blk_addr+1	Tar_blk_addr
	2	Cur_blk_addr+2	Tar_blk_addr-1
	1	Tar_blk_addr	Cur_blk_addr + 1
	2	Tar_blk_addr-1	Cur_blk_addr + 2

이후의 선인출 시에는 〈표 5〉에서와 같은 선인출 후보 블럭에 대해서 선인출을 시도한다.

순차 경로를 따르는 경우에 바로 다음 블럭에 대한 참조에서 접근 실패가 발생해서 그 다음블럭에 대한 선인출을 중단시키기 때문에, 순차 참조를 한 경우에는 Current Block Address+1인 블럭을 먼저 선인출 한다. 또한 전진 선인출의 경우 목표 블럭이 이미 존재하는 경우에는 Target Block Address - 1인 블럭에 대한 선인출 요구 역시 발생시킨다. 이는 목표 블럭을 참조하기 이전에 이 블럭을 참조할 가능성이 많기 때문이다.

논문에서 고려하는 전진 선인출의 정도가 2인 전진 선인출에서는 2 블럭 이후에 참조될 블럭을 선인출하게 되므로 발생한 선인출 요구를 저장하기 위해 non-blocking 캐쉬 시스템에서 사용하는 인출 요구 리스트와 유사한 선인출 요구 버퍼를 두었다[18]. 캐쉬 접근 실패 시에는 현재 수행되고 있는 선인출을 중단하고 캐쉬 접근 실패를 처리한다. 그러나 새로운 선인출 요구는 현재 진행되고 있는 선인출은 중단시키지 않으며, 선인출 요구큐(prefetch request queue)의 최선두에 저장되어서 현재 진행되고 있는 선인출이 종료된 후에 선인출을 시도한다. 즉 선인출 요구 버퍼는 후입 선출의 메커니즘을 갖게 된다. 본 논문에서는 8개의 엔트리를 갖는 선인출 요구 버퍼를 두었다. 또한 선인출 버퍼에 저장되어 있던 선인출 요구를 실제로 선인출이 시작되기 이전에 다시 선인출 참조(prefetch lookup)를 수행해서 캐쉬내의 메모리에 존재하지 않는 경우에만 선인출을 수행한다. 이는 선인출 버퍼에서 지연되는 시간동안에 그 블럭이 캐쉬 메모리에 적재(load)될 수 있기 때문이다.

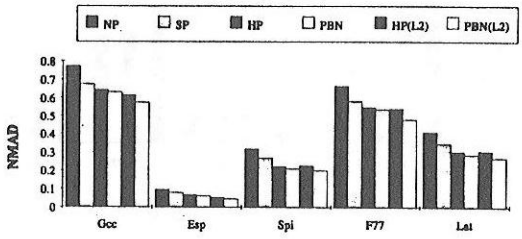
표 6 선인출 정확도(prefetch accuracy)(%) (메모리 지연 = 8, CPI = 1)

트레이스	블럭크기(워드)	HP_8	HP_8(L2)	HP_{16}	HP_{16}(L2)
Gcc	4	78.6	61.3	79.9	62.7
	8	69.8	55.8	71.2	57.6
Esp	4	84.6	73.8	83.3	71.6
	8	78.3	60.9	76.4	59.1
Spi	4	87.0	73.3	87.5	73.6
	8	78.7	71.1	81.3	68.7
F77	4	77.4	60.5	79.0	62.9
	8	68.3	53.4	70.5	55.7
Lat	4	85.6	70.8	85.2	70.7
	8	77.4	61.6	79.4	62.3

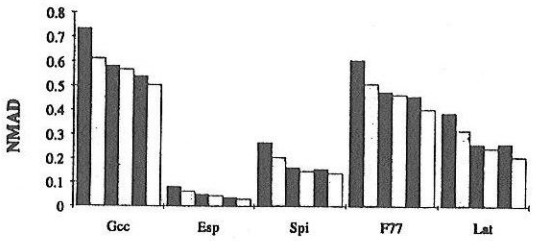
전진 선인출의 정도를 증가시킴에 따라서 정확한 선인출에 의해서 얻을 수 있는 성능의 향상은 증가하나, 전진 선인출 정도의 증가는 선인출 정확도를 감소시킨다. 〈표 6〉에서는 HP 선인출 기법과, 전진 선인출의 정도가 2인 HP 기법인 HP(L2) 기법의 선인출 정확도(prefetch accracy, PA)를 보이고 있다. 선인출 정확도는 선인출된 블럭 중에 CPU에 의해서 참조되는 블럭의 비율로 정의 하였다.

〈표 6〉에서 보여지는 바와 같이 전진의 정도가 2인 전진 선인출의 정확도가 전진 선인출을 수행하지 않는 HP 선인출 기법에 비해 많이 낮음을 알 수 있다. 본 논문에서 제안한 PBN 선인출 기법은 다른 경로를 함께 선인출하기 때문에 선인출 정확도의 저하를 완화할 수 있으므로 전진 선인출시에 보다 우수한 성능을 기대할 수 있을 것으로 생각된다. 〈그림 5〉에는 8 킬로 바이트의 명령어 캐쉬를 갖는 시스템에서 HP(L2)와 PBN(L2) 기법의 NMAD가 〈그림 3〉에서 보여진 다른

기법들의 NMAD와 함께 제시되어 있다.



(a) 블럭 크기 = 4 워드(16 바이트), 명령어 캐쉬 = 8 킬로 바이트



(b) 블럭 크기 = 8 워드(32 바이트), 명령어 캐쉬 = 8 킬로 바이트

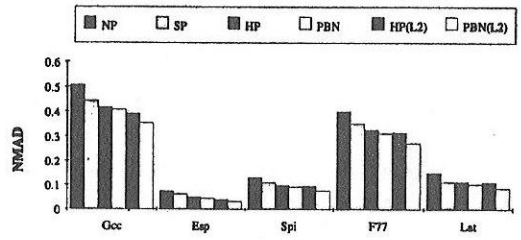
그림 5 정규화된 메모리 접근 지연(NMAD) (메모리 지연 = 8, CPI = 1).

HP(L2) 기법과 PBN(L2) 기법은 각각 전진 선인출의 정도가 2인 HP, PBN 선인출 기법을 의미한다. 예상했던 바와 같이 PBN(L2) 선인출 기법은 다른 선인출 기법에 대해서 큰 성능의 향상을 보이고 있다. HP(L2) 기법은 그리 좋은 성능을 보이지 못하고 있는데 이는 [8]에서 지적한 바와 같이 낮은 선인출 정확도와 캐쉬 접근 실패가 몰려서 발생하는 등의 이유에 의한 것으로 생각된다. 16킬로 바이트의 명령어 캐쉬를 갖는 여러 선인출 기법의 NMAD들을 〈그림 6〉에 제시하였다.

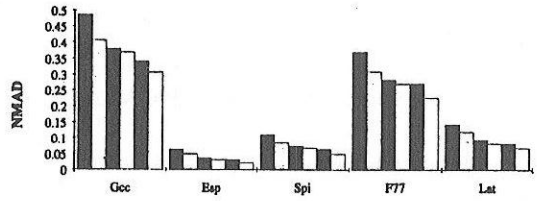
또한 여러 캐쉬 구성에서의 PBN선인출 기법의 RPI를 〈표 7〉에서 보이고 있다.

RPI(PBN_m(L2))는 RPI(PBN_m)과 유사하게 다음의 식에 의해서 계산된다.

$$RPI(PBN_m(L2)) = \frac{NMAD(PBN_m(L2)) - NMAD(HP_m(L2))}{NMAD(HP_{2m}(L2)) - NMAD(HP_m(L2))} \quad (5)$$



(a) 블럭 크기 = 4 워드(16 바이트), 명령어 캐쉬 = 16 킬로 바이트



(b) 블럭 크기 = 8 워드(32 바이트), 명령어 캐쉬 = 16 킬로 바이트

그림 6 정규화된 메모리 접근 지연(NMAD) (메모리 지연 = 8, CPI = 1)

표 7 RPI(PBN_m)(상대적 성능 증가) (%) (메모리 지연 = 8, CPI = 1)

트레이스	블럭크기 (워드)	PBN_8	PBN_8(L2)	PBN_{16}	PBN_{16}(L2)
Gcc	4	4.4	18.1	4.7	22.2
	8	7.1	17.1	6.7	22.7
Esp	4	28.8	57.5	51.4	104.7
	8	39.2	72.6	67.1	108.2
Spi	4	9.3	20.4	9.0	29.9
	8	16.2	19.3	11.7	38.2
F77	4	5.4	26.5	11.0	36.1
	8	6.4	28.3	11.5	41.2
Lat	4	9.4	18.8	16.9	37.7
	8	10.0	32.4	20.4	31.5

〈표 7〉에서 볼 수 있는 바와 같이 표에 제시된 모든 구성에서 RPI(PBN_m(L2))는 12.5%를 넘는다. 즉 PBN 선인출 기법이 수행하는데 필요한 하드웨어 비용에 비해서 얻을 수 있는 성능의 증가가 크다는 것을 말할 수 있다. 또한 모든 RPI(PBN_m(L2))는 대응하는

RPI(PBN_m)의 값보다 크다. 이는 제안한 미참조 선인출 캐쉬가 전진 선인출 기법에 사용되는 경우에 더욱 우수한 성능을 보인다는 것을 의미한다.

4.2.2 큰 메모리 지연을 갖는 시스템에서의 성능 향상

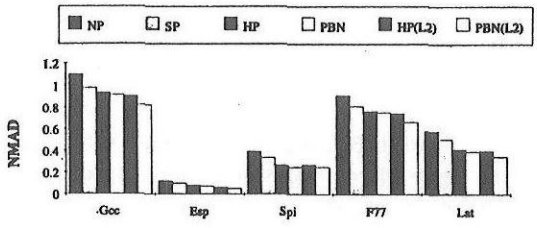
마이크로 프로세서의 동작 속도가 메인 메모리의 접근 속도보다 매우 빠르게 증가하고 있기 때문에 상대적인 속도의 차는 점차로 증가하고 있다. 이러한 추세에 따라서 보다 큰 메모리 지연을 갖는 시스템에서 PBN 선인출 기법의 성능을 살펴보는 것 역시 매우 중요하다고 할 수 있다. 〈그림 7〉에서는 메모리 지연이 16인 시스템에서의 각 선인출 기법의 성능을 보이고 있다. 메모리 지연이 증가함에 따라서 큰 캐쉬 블럭 크기를 갖는 시스템이 좋은 성능을 보이므로 〈그림 7〉에서는 캐쉬 블럭 크기가 8워드(32바이트)인 시스템의 NMAD만을 제시하였다.

〈표 8〉에서는 메모리 지연이 16 싸이클인 시스템의 RPI를 나타내고 있다.

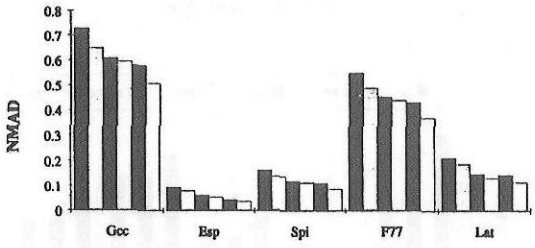
표 8 RPI(PBN_m)(상대적 성능 증가) (%) (메모리 지연 = 16, CPI = 1)

트레이스	PBN_8	PBN_8(L2)	PBN_{16}	PBN_{16}(L2)
Gcc	5.2	24.3	4.6	28.4
Esp	25.5	61.8	55.6	62.5
Spi	11.8	12.6	8.3	32.4
F77	3.4	25.4	7.1	35.2
Lat	8.0	20.6	18.4	35.3

〈그림 7〉과 〈표 8〉을 보면 PBN 선인출 기법의 HP 선인출 기법에 대한 성능 향상이 메모리 지연이 큰 시스템에서도 상당함을 알 수 있다. 이러한 결과는 CPU와 메모리 간의 성능격차가 증가하더라도 PBN 선인출 기법이 효율적으로 적용될 수 있을 것이라는 예상을 가능하게 한다.



(a) 명령어 캐쉬 = 8 킬로 바이트



(b) 명령어 캐쉬 = 16 킬로 바이트

그림 7 정규화된 메모리 접근 지연(NMAD) (메모리 지연 = 16, CPI = 1)

4.2.3 선인출에 요구되는 메모리 트래픽

선인출 기법의 문제점의 하나로써 선인출에 의한 메모리 트래픽의 증가가 지적되고 있다. 특히 제안한 선인출의 경우에는 순차와 비순차의 두 블럭을 모두 선인출하기 때문에 메모리 트래픽의 증가가 더욱 심각할 것으로 예상된다. 본 절에서는 이러한 메모리 트래픽의 증가를 감소시키는 역할을 하는 미참조 선인출 캐쉬가 부가적인 메모리 트래픽에 미치는 영향을 확인한다.

NRP 캐쉬가 메모리 트래픽에 미치는 영향을 살펴보기 위해서 NRP 캐쉬를 가지지 않고 두 경로를 선인출하는 HP 기법과 미참조 선인출 캐쉬를 사용하는 PBN 기법의 메모리 트래픽을 비교하였다. 다음의 용어들이 이러한 비교를 위해서 정의되었다.

MT(PBN) : PBN 선인출 기법을 사용할 때 선인출되는 블럭의 수와 캐쉬 접근 실패 수의 합.

MTb(HP) : 두 경로를 모두 선인출 하는 HP 기법을 사용할 때 선인출되는 블럭의 수와 캐쉬 접근 실패 수의 합.

MT_{nb}(HP) : 한 경로만을 선인출하는 HP 기법을 사용할 때 선인출되는 블럭의 수와 캐쉬 접근 실패 수의 합.

MT_{nb}(HP)는 선인출 후보 블럭이 이미 캐쉬나 선인출 버퍼에 존재하는 경우에 다른 후보 블럭의 선인출을 시도하지 않는 HP 선인출 기법, MT_b(HP)는 선인출 후보 블럭이 이미 캐쉬나 선인출 버퍼에 존재하면 다른 선인출 후보 블럭에 대한 선인출을 시도하는 HP 선인출 기법의 메모리 트래픽 양이다. 증가한 메모리 트래픽중 미참조 선인출 캐쉬에 의해서 감소된 트래픽의 비율 MT_r은 다음의 식에 의해서 계산된다.

$$MT_r = (1 - \frac{MT(PBN) - MT_{nb}(HP)}{MT_b(HP) - MT_{nb}(HP)}) \times 100(\%) \quad (6)$$

각 트레이스에 대한 MTr값은 〈표 9〉에 나타나 있다.

표 9 MT_r : 감소된 메모리 트래픽(%)(메모리 지연 = 8, CPI = 1).

캐쉬 크기 (킬로바이트)	블럭크기 (워드)	gcc	esp	spi	f77	lat
8	4	59.8	97.4	84.5	64.8	71.7
	8	50.3	86.6	71.6	57.4	82.4
16	4	67.9	112.1	91.6	70.9	92.8
	8	60.2	106.6	90.1	63.9	89.8

MT_r은 50.3%에서 112.1%의 값을 갖는다. 이는 미참조 선인출 캐쉬가 두 경로를 선인출 하는데 필요한 추가적인 메모리 트래픽의 50% ~ 112%정도를 감소시킨다는 것이다. MT_r이 112%라는 것은 PBN(L2) 선인출 기법이 HP_{nb}기법에 비해 오히려 적은 메모리 트래픽을 갖는다는 것이다. PBN(L2) 선인출 기법은 HP_{nb} 기법에 비해 위의 캐쉬 구성에서 최대 55.8%의 메모리 트래픽의 증가만을 일으킨다. 그러므로 미참조 선인출 캐쉬가 추가적인 메모리 트래픽의 상당부분을 감소시킨다고 할 수 있다.

5. 결론 및 향후 연구방향

본 논문에서는 CPU에 의해서 참조되지 않은 선인출 블럭을 저장하는 미참조 선인출 캐쉬를 제안하였다. 미참조 선인출 캐쉬는 CPU에 의해서 참조되지 않은 블럭을 저장함으로써 캐쉬 접근 실패를 감소시키며, 순차와 비순차경로를 모두 선인출 하는데 추가적으로 요구되는 메모리 트래픽의 대부분을 제거할 수 있었다.

미참조 선인출 캐쉬에 의해서 제거되는 추가적인 메모리 트래픽은 약 50% ~ 112%에 이른다. 또한 미참조 선인출 캐쉬는 캐쉬접근실패를 줄여서 이를 이용하는 PBN 선인출 기법은 향상된 메모리 성능을 갖는다. 제안한 선인출 기법은 트레이스 구동 시뮬레이션을 통해서 성능평가를 수행하였으며, 제안한 선인출 기법은 다른 선인출 기법에 비해서 우수한 메모리 성능을 보인다. 특히 제안한 미참조 선인출 캐쉬는 큰 메모리 지연을 효과적으로 극복하기 위해서 전진 선인출을 수행하는 경우에 더욱 큰 성능의 향상을 얻을 수 있다. 미참조 선인출 캐쉬 등 제안한 선인출 기법을 수행하기 위해서 필요한 추가적인 하드웨어 비용이 단순히 명령어 캐쉬의 용량을 증가시키는 것에 비해서 최고 10배에 가까운 효율을 가짐을 확인하였다. 이것은 미참조 선인출 캐쉬가 단순히 명령어 캐쉬의 크기를 증가시키는 것의 좋은 대안이 될 수 있음을 보여주었다

본 연구에서는 또한 CPU와 메모리 간의 지속적인 성능격차가 증가할 것을 고려해서 큰 메모리 지연을 갖는 시스템에서 제안한 선인출 기법의 성능을 평가하였는데, 이러한 시스템 환경에서도 제안한 선인출 기법이 우수한 성능을 보임을 알 수 있었다. 이는 앞으로 CPU와 메모리간의 성능격차가 증가하여도 제안한 선인출 기법이 효과적인 성능향상을 보일 것이라는 예상을 할 수 있게 해준다.

향후 연구방향은 다음과 같다. 바로 이전의 수행 경로뿐만 아니라 몇 회의 수행 정보를 이용해서 복수개의 목표 주소를 예측 테이블에 저장하여 선인출을 수행하는 방법에 대한 연구를 고려하고 있다. 다른 연관도 등 여러 캐쉬 시스템 변수의 변화와 문맥 교환이나 운영체제에 의한 메모리 참조가 제안한 선인출의 성능에 어떠한 영향을 미치는가에 대한 연구도 중요할 것으로 생각한다.

참 고 문 헌

[1] John L. Hennessy and David A. Patterson. Computer Architecture: A Quantitative Approach. Morgan Kauffman, San Mateo, CA, 1990

[2] Mark D. Hill, "Aspects of Cache Memory and Instruction Buffer Performance," Ph. D. Thesis, Computer Science Division Technical Report UCB/CSD 87/381, University of California, Berkeley, Nov. 1987.

[3] Norman P. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," 17th ISCA, May 1990, pp 364-373.

[4] David Kroft, "Lockup-free Instruction Fetch/Prefetch Cache Organization," 8th ISCA, 1981, pp. 81-87.

[5] Steven Przybylski, "The Performance Impact of Block Sizes and Fetch Strategies," 17th ISCA, May 1990, pp. 160-169.

[6] Alan J Smith, "Sequential Program Prefetching in Memory Hierarchies," IEEE Computer, Vol. 11, No. 12, Dec. 1978, pp. 7-21.

[7] Alan J. Smith, "Cache Memories," ACM Computing Surveys, Vol 14, No. 3, Sep 1982, pp. 473-530.

[8] 김 성백, 박 선호외, "효율적인 전진 선인출 기법," 정보과학회 논문지, 제 20권, 제 6호, 1993년 6월, pp. 903-915.

[9] Honesty C. Young and Eugene J. Shekita, "An

Intelligent I-Cache Prefetch Mechanism,'' ICCD 1993, Oct. 1993, pp. 44-49.

[10] Johnny K. F. Lee and Alan J. Smith, ''Branch Prediction Strategies and Branch Target Buffer Design,'' IEEE Computer, Vol. 17, No. 1, Jan. 1984, pp. 6-22.

[11] James E. Smith, ''A Study of Branch Prediction Strategies,'' 8th ISCA, 1981, pp. 135-148.

[12] James E. Smith and W.-C. Hsu, ''Prefetching in Supercomputer Instruction Caches,'' Supercomputing '92, Nov. 1992, pp. 588-597.

[13] Alan Jay Smith, ''Line(Block) Size Choice for CPU Cache Memories,'' IEEE Transactions on Computers, Vol. C-36, No. 9, Sep. 1987, pp. 1063-1075.

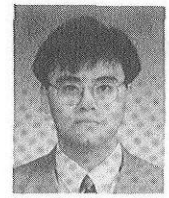
[14] *Chris H. Perleberg and Alan Jay Smith,* ''Branch Target Buffer Design and Optimization,'' IEEE Transactions on Computers, Vol. 42, No. 4, Apr. 1993, pp. 396-412.

[15] Harvey G. Cragon, Branch Strategy Taxonomy and Performance Models, IEEE Computer Society Press, Los Alamitos, CA, 1992

[16] Gorden Irlam, ''Spa tool,'' Unix-style manual page, The Spa tool and documentation are available from ftp.adelaide.edu.au

[17] Mark D. Hill, ''A Case for Direct-Mapped Caches,'' IEEE Computer, Vol. 21, No. 12, Dec. 1988, pp. 25-40.

[18] Tien-Fu Chen and Jean-Loap Baer, ''Reducing Memory Latency via Non-Blocking and Prefetching Caches,'' 5th ASPLOS, Oct. 1992, pp. 51-61.



박 기 호

1993년 연세대학교 이과대학 전산과학과(학사). 1995년 연세대학교 대학원 전산과학과(석사). 1995년 ~현재 연세대학교 대학원 전산과학과 박사과정. 관심분야 : 컴퓨터 구조, 병렬 처리 시스템, VLSI design임

한 탁 돈

제 22권 1호 참조

김 신 덕

1982년 연세대학교 공과대학 전자공학과(학사). 1987년 University of Oklahoma 전기공학(석사). 1991년 Purdue University 전기공학(박사). 1993.2~1994.2 광운대학교 컴퓨터공학과 조교수. 1994.3~현재 연세대학교 이과대학 컴퓨터과학과 조교수. 관심분야는 병렬 처리 시스템, 컴퓨터 구조, heterogeneous computing임