캐시 메모리의 유용성을 높이는 동적 선인출 필터링 기법

전 영 숙[†] · 이 병 권[††] · 이 춘 희[†††] · 김 석 일[††††] · 전 중 남[††††]

요 약

캐시 선인출 기법은 메모리 참조에 따른 지연시간을 줄이는 효과적인 방법이다. 그러나 너무 적극적으로 선인출할 경우에 캐시 오염을 유발시켜 선인출에 의한 장점을 상쇄시킬 뿐만 아니라 버스 트래픽을 증가시켜 전체 성능의 저하를 가져 올 수 있다. 본 연구에서는 선인출로 인한 캐시의 오염을 줄이기 위해 필터 테이블을 참조하여 선인출 명령을 수행할 지의 여부를 동적으로 판단하는 선인출 필터링 기법을 제시한다.

본 논문에서는 먼저 기존 연구에서의 문제점을 분석하기 위해 선인출 해싱 테이블 1bitSC 기법을 보였는데, 이 기법은 기존 연구와 같이 N:1 매핑을 사용하는 반면, 각 엔트리의 값을 1비트로 하여 두 가지 상태값을 갖도록 하였다. 비교 연구를 위해 완전 블록주소 테이블 기법을 제시하여 비교 기준으로 사용하였다. 마지막으로 본 논문의 주 아이디어인 정교한 필터링을 위한 선인출 블록주소 참조 테이블 기법을 제안하였다. 이 구조는 선인출 해싱 테이블 1bitSC기법과 같은 테이블 길이를 가지며, 각 엔트리의 내용은 완전 블록주소 테이블 기법과 같은 항목을 가지도록 하여 최근에 이 사용된 데이터의 블록주소가 필터 테이블의 하나의 엔트리와 대응되도록 1:1 매핑을 하였다.

일반적으로 많이 사용되는 선인출 기법과, 일반 벤치마크 프로그램과 멀티미디어 벤치마크 프로그램들에 대하여 캐시의 매개변수들을 변화시켜가면서 실험을 하였다.

PBALT기법은 필터링 하지 않은 경우에 비해 최대 22% 향상된 결과를 보이고, 기존 PHT2bSC 기법과 비교하여 캐시 미스율이 7.9% 감소하였다. 메모리 참조 지연 시간(MADT)은 제안하는 PBALT 기법이 기존 연구에 비해 6.1% 감소하여 전체 수행 시간에 있어서 성능이 향상되었다.

키워드 : 캐시 메모리, 선인출 알고리즘, 필터링, 선인출 캐시 구조

A Dynamic Prefetch Filtering Schemes to Enhance Usefulness Of Cache Memory

Young-Suk Chon[†] · Byung Kwon Lee[††] · Lee Chun Hee[†††] · Suk-il Kim[††††] · Joong-nam Jeon[††††]

ABSTRACT

The prefetching technique is an effective way to reduce the latency caused memory access. However, excessively aggressive prefetch not only leads to cache pollution so as to cancel out the benefits of prefetch but also increase bus traffic leading to overall performance degradation. In this thesis, a prefetch filtering scheme is proposed which dynamically decides whether to commence prefetching by referring a filtering table to reduce the cache pollution due to unnecessary prefetches

In this thesis, First, prefetch hashing table 1bitSC filtering scheme(PHT1bSC) has been shown to analyze the lock problem of the conventional scheme, this scheme such as conventional scheme used to be N:1 mapping, but it has the two state to 1bit value of each entries. A complete block address table filtering scheme(CBAT) has been introduced to be used as a reference for the comparative study. A prefetch block address lookup table scheme(PBALT) has been proposed as the main idea of this paper which exhibits the most exact filtering performance. This scheme has a length of the table the same as the PHT1bSC scheme, the contents of each entry have the fields the same as CBAT scheme recently, never referenced data block address has been 1:1 mapping a entry of the filter table.

On commonly used prefetch schemes and general benchmarks and multimedia programs simulates change cache parameters.

The PBALT scheme compared with no filtering has shown enhanced the greatest 22%, the cache miss ratio has been decreased by 7.9% by virtue of enhanced filtering accuracy compared with conventional PHT2bSC. The MADT of the proposed PBALT scheme has been decreased by 6.1% compared with conventional schemes to reduce the total execution time.

Key Words : Cache Memory, Prefetch Algorithm, Filtering, Prefetch Cache Architecture

1. 서 론

메모리 참조 지연 시간을 줄이기 위한 방법으로서 캐시 메모리[1, 2, 16]를 사용하며, 캐시 선인출(cache prefetch) 기법[1, 2, 7-9, 16]은 프로세서가 미래에 사용할 것으로 예측되는 데이터를 실제로 요구하기 전에 미리 주기억 장치로부터 인출하여 캐시로 적재하는 기법이다. 캐시 선인출 기법은 요구 미스(demand miss)를 줄이는데 도움이 되지만 너무 적극적인 선인출은 전체적인 메모리 버스 트래픽을 증가시켜 오히

† 준 회 원 : 충북대학교 대학원 컴퓨터과학과
†† 준 회 원 : 충북대학교 전자계산대학원 박사과정
††† 정 회 원 : 충북대학교 일반대학원 전자계산전공 박사과정 수료
†††† 종신회원 : 충북대학교 전기전자컴퓨터공학부 교수
논문접수 : 2005년 10월 20일, 심사완료 : 2006년 2월 27일

려 캐시 액세스를 지연시킬 수 있을 뿐만 아니라, 불필요한 데이터를 선인출 함으로써 캐시 오염을 발생시켜 선인출로 인한 이득을 상쇄시킬 수 있다는 문제점을 갖고 있다.

이러한 캐시 오염 문제를 해결하기 위한 방법으로서 선인출 데이터의 사용 유무에 대한 이력 정보를 이용하여, 사용되지 않을 것으로 판단되는 데이터는 선인출을 하지 않도록 제거하는 선인출 필터링(prefetch filtering) 기법[10-12]이 있다.

필터의 종류로는 필터링 특성이 고정된 정적 필터[10, 12]와 실행하면서 동적 프로파일에 따라 필터의 특성이 변하는 동적 필터[11]가 있다. Srinivasan 등에 의해 제안된 정적 필터링 기법은 선인출에 의해 생성된 트래픽과 미스 수에 근거한 프로파일에 따라 선인출들을 몇 개의 유형으로 분류하고 이러한 프로파일을 통해 오프라인으로 오염이 되는 정보를 수집하게 되며, 실제 수행 시에는 이 정보를 통해 선인출을 수행한다.

Zhuang 등이 제안한 동적 필터링 기법[11]은 선인출기가 생성한 선인출 주소의 일부를 가지고 필터 테이블을 참조하여 상태를 네 가지로 관리하여 각 주소의 상태에 따라 필터링을 수행한다.

이 기법의 문제점은 두 가지로 요약될 수 있다. 첫째, 정확성이 결여되는 이유는 필터 테이블 참조 시 해싱을 이용한 N:1 매핑을 하기 때문이다. 즉, 서로 다른 블록 주소에 대해서 같은 엔트리를 공유하기 때문에 필터링을 해야 하는 경우와 하지 말아야 하는 경우의 정보가 혼합되어 정확한 판단을 할 수 없도록 되어 있는 기법이라고 할 수 있다.

둘째, 위의 N:1 매핑과 더불어 각 엔트리의 상태를 2bit 포화 카운터를 사용함으로써 불필요한 선인출 뿐만 아니라 결과적으로 유용한 선인출 모두를 필터링하게 되어 결과적으로 모든 선인출이 대부분 금지되어 캐시 미스율이 증가하는 현상이 발생한다. 이 논문에서는 위와 같은 기존 연구의 문제점들을 해결하기 위해서, 정확한 필터링 즉, 유용한 선인출은 최대한 보존하면서 불필요한 선인출만을 선택적으로 감소시켜 전체 성능을 향상시킬 수 있는 필터링 방법을 제안하고자 한다. 이 논문의 주 아이디어인 선인출 블록 주소 참조 테이블 기법(PBALT)은 최근에 미사용된 데이터의 블록 주소가 필터 테이블의 하나의 엔트리에 대응되도록 하는 1:1 매핑 구조를 사용한 기법이다. 이 기법은 1:1 매핑 구조를 사용함으로써 기존 연구와 비교하여 선인출 수행의 여부를 정확하게 판단할 수 있으며 이로 인해 캐시 히트율을 높여 캐시 미스로 인한 메모리 액세스 지연 시간을 줄일 수 있다.

실험은 기존 연구의 문제점을 분석하기 위해 필터링 정확도와 시간에 따른 선인출 수를 분석하고, 제안한 기법과의 성능을 비교 평가하기 위해 캐시 미스율과 메모리 참조 지연 시간(MADT)을 지표로 삼아 성능을 검증하였다.

이 논문의 구성은 다음과 같다. 2장에서는 캐시 선인출 기법과 기존 연구로 정적 필터링 기법과 동적 필터링 기법에 대해서 설명한다. 3장에서는 효과적인 동적 선인출 필터링 기법을 제안하고, 4장에서는 실험을 통하여 이 논문에서 제안하는 필터링 기법의 성능을 분석한다. 마지막으로 5장에서는 이 논문에서 얻은 결론과 향후 연구 방향을 기술한다.

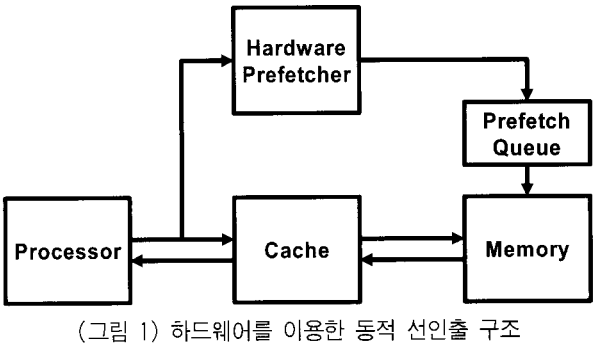
2. 캐시 선인출과 필터링 기법

2.1 캐시 선인출

캐시 선인출 기법[1, 2, 4, 7, 9]은 프로세서가 사용할 것으로 예측되는 데이터를 실제로 필요하기 전에 주기억 장치에서 캐시로 인출하여 캐시 미스로 인한 지연 시간을 줄일 수 있는 효과적인 방법이다.

(그림 1)은 하드웨어를 이용한 동적 선인출 구조를 보여준다. 프로세서가 캐시로 메모리 참조 명령어를 전송할 때 그 메모리 주소는 선인출기(Hardware Prefetcher)에도 동시에 전송된다. 선인출기는 그 메모리 주소를 기반으로 적절한 선인출 알고리즘에 의해 선인출 여부를 결정하고 선인출할 주소를 계산한다. 계산된 주소는 선인출 큐(Prefetch Queue)에 저장되고, 이후 해당 주소의 데이터가 메모리에서 인출되어 캐시에 저장된다.

캐시 선인출 방법은 정적 선인출 기법[21]과 동적 선인출 기법[1,2,7,8,9,16]으로 구분되며, 이 논문에서는 제안하는 필터링 기법의 실험을 위하여 OBL[1-2], RPT[9], Correlation [7] 세가지 동적 선인출 기법들을 사용한다.



(그림 1) 하드웨어를 이용한 동적 선인출 구조

2.2 캐시 선인출 필터 기법

하드웨어 선인출 구조는 실제로 미래에 사용될 필요한 데이터 외에도 불필요한 데이터도 또한 선인출함으로써 캐시 오염을 일으킬 수 있다는 문제점을 갖고 있다. 이와 같이 불필요하게 선인출된 데이터는 유용한 요구 데이터 및 유용한 선인출 데이터를 캐시로부터 축출시켜 캐시 미스율을 다시 증가시키게 하는 요인이 된다. 과도한 선인출을 줄이기 위한 다른 방법으로는 공격적인 선인출 방식을 사용하고 그 대신 개시된 선인출 중에서 실제로 미래에 사용될 선인출을 선택하여 최종적으로 유용한 선인출만을 통과시키는 선인출 필터를 사용하는 방법이 있을 수 있다.

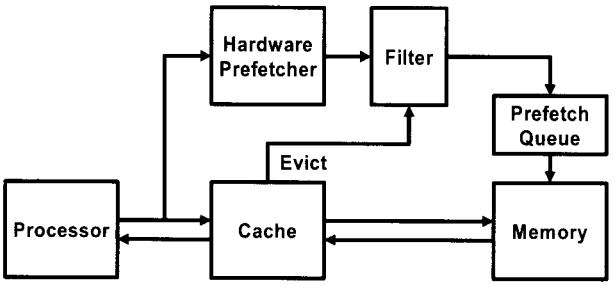
(그림 2)는 필터링을 적용한 하드웨어 선인출 구조를 보여주고 있다. 그 동작을 살펴보면 다음과 같다.

- 프로세서에 의해 요청된 데이터 주소가 하드웨어 선인출기에도 동시에 전달된다.
- 하드웨어 선인출기에서는 적절한 선인출 기법을 이용하여 선인출을 수행할지 여부를 결정한다.
- 계산된 선인출 주소는 먼저 필터 테이블을 참조하여 필

터링 조건에 만족하지 않는 경우에만 선인출 큐에 저장된다.

- 필터 테이블은 캐시에서 선인출 된 데이터가 축출될 때마다 그 정보를 피드백하여 동적으로 내용을 갱신한다.

즉, 하드웨어 선인출기로부터 생성된 모든 선인출 명령을 선인출 큐에 전달하는 것이 아니라, 이들 중 앞으로 사용될 가능성이 높은 선인출만을 여과하여 선인출을 한다.

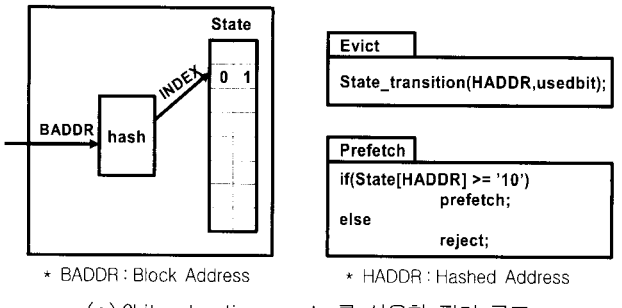


(그림 2) 필터링을 적용하는 하드웨어 선인출 구조

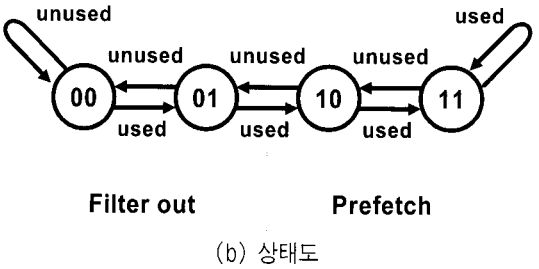
이러한 동적 선인출 필터는 과거에 개시하였던 선인출의 결과를 바탕으로 자기 자신을 동적으로 갱신하여 선인출 알고리즘과 무관하게 유용한 선인출과 불필요한 선인출을 성공적으로 판단할 수 있어야 한다. 선인출의 결과가 유용했었는지 그렇지 않았는지에 대한 정보는 캐시로부터 축출되는 선인출 데이터들이 캐시에 있는 동안 프로그램에 의해 참조되었는지 아닌지의 여부로부터 얻어낼 수 있다.

2.3 선인출 해싱 테이블 2bitSC 기법

(그림 3)은 기존 연구로 Zhuang등에 의해 제안된 선인출 해싱 테이블 2bitSC 필터(PHT2bSC: Prefetch Hashing Table 2bitSC) 구조를 보여준다. (그림 3)(a)은 서로 다른 데이터 주소에 대해서 같은 인덱스 주소를 공유하는 N:1 매핑 구조를



(a) 2bit saturation counter를 사용한 필터 구조



(b) 상태도

(그림 3) 선인출 해싱 테이블 2bitSC 구조(PHT2bSC)

보이고 있다. 해당 선인출 주소(BADDR : Block ADDRess)는 해시 함수를 통과하여 필터 테이블에 인덱스를 한다. 인덱스된 위치의 각 엔트리 값은 2bit로 구성되어 있으며, 캐시에서 데이터가 축출될 때마다 축출된 데이터의 주소를 이용하여 (그림 3)(b)에서와 같이 상태가 변경된다. 선인출기가 생성한 선인출 주소의 인덱스 주소로 테이블을 참조하여 해당 엔트리의 상태값이 2 이상인 경우에는 선인출 명령을 수행하고, 그렇지 않은 경우에는 선인출을 하지 않는다.

이 기법을 제안한 Zhuang의 논문에서는 선인출 명령이 많이 발생되도록 하기 위해 소프트웨어 기반과 하드웨어 기반의 선인출 기법 두 가지(OBL, Correlation)를 이용하여 과다하게 선인출을 수행하게 한 후 발생된 선인출 명령을 모두 사용해서 필터링 효과를 검증했다. 이는 하드웨어 기반 선인출만을 사용하는 통상적인 경우에 비하여 필터링 효과가 과장된 측면이 있다. 왜냐하면 하드웨어 기반의 선인출 기법은 소프트웨어 기반의 선인출 기법에 비해서 선인출의 수가 매우 많은 반면 선인출 정확도는 매우 낮기 때문에 선인출 수가 매우 적고 선인출 정확도가 상대적으로 높은 소프트웨어 기반 선인출 명령어와의 구분이 용이해지게 되어, 하드웨어 선인출 명령어들만을 대상으로 하는 경우에 비해 필터링 정확도 및 필터링 효과가 훨씬 높게 측정될 수 있기 때문이다. 극단적으로 하드웨어 선인출에 의한 선인출 명령들을 거의 모두 제거하게 되는 경우에도 소프트웨어에 의한 매우 적은 수의 정확한 선인출들로 인해서 선인출 효율이 매우 높게 측정될 수 있게 된다.

또한 이 기법은 불필요한 선인출의 수는 매우 감소시켰으나, 감소되는 전체 선인출 수에만 중점을 두었고, 전체 성능면에서는 평가되지 않았다. 즉, 성능을 저하시키지 않고 선인출 수를 감소시키기 위해서는 유용한 선인출은 보존되면서 불필요한 선인출만 감소시켜야 되는데, 결과적으로 이 구조는 유용한 선인출과 불필요한 선인출 모두를 감소시켰다.

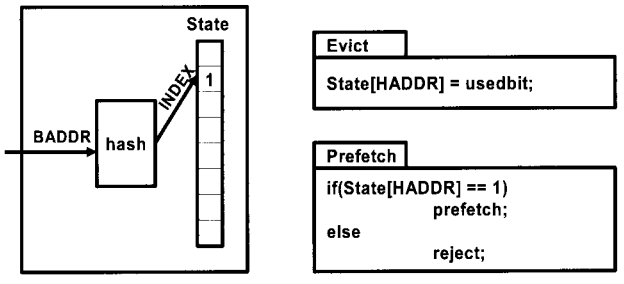
기존 동적 필터링 기법의 문제점은 두 가지로 요약될 수 있다.

첫째로는 해싱을 이용한 N:1 매핑을 하기 때문에 정확성이 결여된다는 점이다. 즉, 서로 다른 데이터 주소에 대해서 같은 엔트리를 공유하기 때문에 필터링을 해야 하는 경우와 하지 말아야 하는 경우의 정보가 혼합되어(aliasing) 정교한 판단을 할 수 없도록 되어 있는 구조라고 할 수 있다.

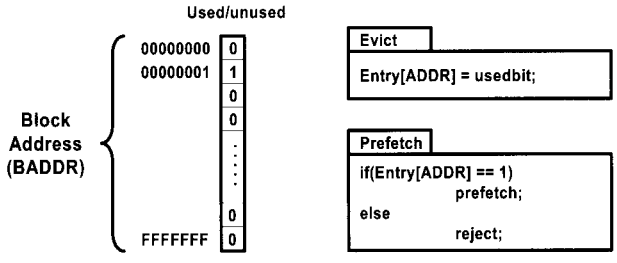
둘째로는 위와 같은 aliasing 효과와 더불어 2bit 포화 카운터를 사용함으로써 결과적으로 모든 선인출이 대부분 억제되어 버리는 결과를 낳는다. 따라서 불필요한 선인출 뿐만 아니라 유용한 선인출 모두를 억제시켜 결과적으로 캐시 미스율을 오히려 증가시키게 된다. 이러한 현상은 이 논문에서는 잠김(stuck, lock) 현상이라고 정의하기로 한다.

2.4 선인출 해싱 테이블 1bitSC 기법

기존 PHT2bSC 기법에서는 해싱과 2bit 포화 카운터를 사용함으로써 전체 선인출의 총합을 많이 감소시키기는 하지만 유용한 선인출 수 또한 줄이게 되어 전체 캐시 미스율을 늘리



(그림 4) 선인출 해싱 테이블 1bitSC 구조(PHT1bSC)



(그림 5) 완전 블록 주소 테이블 구조(CBAT)

는 단점을 갖는다. 이는 위에서 설명한 잠김 현상 때문인데, 이 문제점을 비교 분석하기 위해 각 엔트리의 값이 2가지 상태만을 갖는 선인출 해싱 테이블 1bitSC 기법(PHT1bSC: Prefetch Hashing Table 1bitSC)을 보였다.

(그림 4)에서와 같이 기존 구조의 해싱을 이용한 필터 테이블의 인덱싱을 그대로 사용하고, 포화 카운터의 깊이(depth)만을 2로 줄여서 사용이 되지 않고 축출된 경우에는 0상태를 갖고, 사용이 되고 축출된 경우는 1상태를 갖게 한다. 1상태에서 사용이 되고 축출된 데이터는 1상태를 유지하고 사용이 되지 않고 축출된 경우에는 0상태로 전이 된다. (그림 4)는 두 가지 상태를 갖는 경우의 상태 전이를 나타낸 것이다.

PHT1bSC 기법은 하나의 상태 비트만을 두어서 사용 여부에 따라 즉각적으로 선인출 여부가 바뀌도록 의도한 것이다. 이 경우에도 N:1 매핑에 의한 정확도 문제는 해결되지 않지만, 한번 상태가 0이 되더라도 사용된 선인출 데이터가 하나라도 캐시에서 축출될 경우 다시 상태가 1이 되므로 위의 기존 카운터를 사용한 경우에서와 같은 필터링 상태에서의 잠김 현상이 일어날 확률이 감소하게 된다.

결론적으로 잠김 현상을 줄이기 위해 PHT2bSC 기법에서 used 방향은 캐시에서 데이터가 축출될 때만 천이하는 것이 아니라 조금 더 복잡하고 지능적인 알고리즘으로 처리해 주어야 할 필요가 있다.

2.5 완전 블록 주소 테이블 기법

과거 이력 정보를 이용하는 방법에서 가장 이상적인 정보가 되기 위해서는 모든 블록 주소에 대해서 선인출 후 사용이 되었는지의 여부를 모두 필터링 테이블에 저장해야 한다.

(그림 5)는 완전 블록 주소 테이블(CBAT : Complete Block address Table)구조를 나타내고 있는데, 이 구조는 과거 이력 정보를 이용하는 필터 구조들을 비교하고 가장 바람직한 방식을 유도하기 위한 기준을 마련하기 위해서 제시한 것이다.

이 기법은 모든 블록 주소에 대하여 선인출된 이후부터 캐시에서 축출될 때까지 해당 데이터가 사용되었었는지 여부를 모두 필터링 테이블에 저장하며, 이후에 또 다시 동일 주소의 데이터가 선인출되려고 할 때 해당 테이블을 참조하여 사용되지 않은 경우에는 선인출 명령을 개시하지 않는 기법이다.

이 기법은 데이터의 블록 주소 전부를 저장해야 하므로 테이블의 크기가 너무 커져서 실제로는 구현하기가 어렵다는 문제점이 있다. 여기서 블록 주소는 논리 주소에서 블록 안에 저장되는 각각의 워드들을 표현하기 위한 오프셋을 제거한 것을 말한다. (그림 5)에서와 같이 블록 주소의 비트수가 N이라 할 때 필터 테이블의 엔트리 수는 총 2^N개가 필요하게 된다. 예를 들면, 논리 주소가 32bit이고 블록크기가 16byte일 때 블록 주소는 32bit - 4bit = 28bit이며, 필터 테이블의 크기는 2^{28} x 1bit가 된다.

한편, (그림 5)의 테이블을 보면 각 엔트리의 값이 1인 경우가 적고 0인 경우가 많은 희소 행렬 형태를 취하고 있음을 알 수 있다. 따라서 이 기법을 시뮬레이션으로 구현하기 위해서 실제로 선인출된 데이터의 블록 주소와 엔트리 값이 0인 엔트리들 만을 구현한다. 즉, 사용이 되지 않은 블록 주소들만을 필터 테이블에 저장하되, 테이블이 오버플로우되어 원래 의도했던 것과 결과가 달라지는 것을 방지하기 위하여 테이블의 크기를 충분히 크게 하여 구현한다.

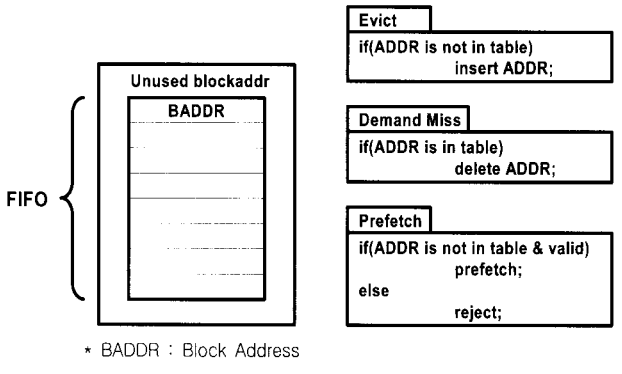
이러한 완전 블록 주소 테이블 기법도 가장 이상적인 기법이라고는 할 수 없는데, 왜냐하면 동일한 데이터 주소라 할지라도 서로 다른 명령어에 대하여 발생될 수 있으며, 명령어에 따라서 불필요한 선인출이 될 수도 있고, 유용한 선인출이 될 수도 있기 때문이다. 실제로, 유용한 선인출인가 불필요한 선인출인가 하는 것은 사실은 데이터 주소가 아닌 명령어에 따라서 정해진다고 할 수 있다.

따라서, 진정으로 이상적인 선인출 필터는 각각의 데이터 주소에 대한 선인출 사용 유무를 가지고 있는 것이 아니고, 각각의 PC (program counter)에 대한 선인출 사용 여부를 가지고 있어야 한다. 그렇게 하기 위해서는 캐시로부터 축출되는 모든 데이터들이 애초에 어떤 명령어에 대해서 선인출이 발생되었는지를 알 수 있어야 한다. 즉, 필터 테이블의 모든 데이터에 대해서 데이터와 함께 PC 정보도 같이 저장되어야 한다. 이것은 필터 테이블의 크기가 커지기 때문에, 이 논문에서는 PC 정보를 이용하지 않고 데이터 주소만을 이용하여 필터링을 수행하도록 한다.

3. 선인출 블록 주소 참조 테이블 필터링 기법

3.1 선인출 블록 주소 참조 테이블 기법

완전 블록 주소 테이블 기법(CBAT)은 과거의 모든 이력 정보를 가지고 있는 반면, 이 논문에서 제안한 선인출 블록 주소 참조 테이블 기법(PBALT: Prefetch Block address Lookup



(그림 6) 선인출 블록 주소 참조 테이블 구조(PBALT)

Table)은 최근에 미사용된 선인출 데이터의 블록 주소만을 저장하도록 테이블의 크기를 제한한 기법이다.

(그림 6)은 선인출 블록 주소 참조 테이블 구조로 위에서 설명한 완전 블록 주소 테이블 기법이 그 크기로 인하여 실제로 구현이 거의 불가능하므로 대신에 사용이 되지 않은 블록 주소만을 FIFO로 구현한 필터 테이블에 저장하며 테이블 크기를 제한한 구조이다.

선인출 데이터가 캐시에서 축출될 때 사용이 되지 않은 경우 필터 테이블에 해당 블록 주소를 저장하고 이후 선인출기가 생성한 선인출 주소가 필터 테이블에 존재하는 경우에 선인출 명령을 제거한다.

이 기법은 블록 주소 전부를 테이블에 저장하므로 각 엔트리의 크기는 커지는 문제점이 있는 반면, 모든 블록 주소를 저장하는 CBAT 구조와 비교하여 엔트리의 수가 크게 줄어드는 장점이 있다. 또한 기존 Zhuang등에 의해 제안된 PHT2bSC 기법과 비교하면 각 엔트리에는 하나의 주소만이 할당되며 이러한 1:1 매핑에 따른 정확한 필터링을 할 수 있다는 장점이 있다. 한편, 미사용된 주소의 개수가 테이블 크기보다 더 커지게 되면 모든 미사용 주소를 포함할 수 없게 되어서 LRU나 FIFO 교체 정책을 써야 한다.

3.2 선인출 필터링 기법 비교

<표 1>은 PHT2bSC, PHT1bSC, PBALT, CBAT 필터링 기법들에 대해 각각의 특징과 단점 및 하드웨어 비용에 대해서 비교한 것이다.

CBAT 기법은 가장 바람직한 방식을 유도하기 위해 제시한 것으로 과거 이력 정보를 이용하는 방법들 중에서 가장 이상적인 구조이긴 하나 모든 블록 주소에 대해서 선인출 후 사용되었는지 여부를 저장해야 하므로 사실상 구현이 불가능한 문제점이 있다.

기존 PHT2bSC 기법은 해싱에 따른 aliasing 문제로 정확성 결여와 2bit 포화 카운터를 함께 사용하기 때문에 잠김 현상이 일어나 선인출 수행을 대부분 금지시키는 문제가 있다.

이에 비해 PHT1bSC 기법은 기존 연구의 해싱은 그대로 사용하여 aliasing 문제는 해결되지 않았지만, 각 엔트리의 값을 1비트 포화 카운터를 사용함으로써 PHT2bSC 기법과 같은 잠김 현상을 줄일 수 있다.

이 논문에서 제안하는 PBALT 기법은 최근에 미사용된 선인출 데이터만을 한정하여 해당 블록 주소를 필터 테이블에 저장을 하며, 1:1 매핑으로 인한 정확한 필터링을 하는 특징이 있는 반면, 각 엔트리에 블록 주소 전부를 저장해야 하기 때문에 다른 기법들과 비교하여 그 크기가 커지는 문제점이 있을 수 있다.

3.3 하드웨어 비용 비교

각 필터링 기법의 하드웨어 비용을 비교하면 CBAT 기법은 블록 주소 비트수가 n이라 할 때 2^n개의 블록 주소 수 x 1bit의 비용이 요구된다. 예를 들어 n이 32인 경우 4Gb가 필요하게 되므로 실제로는 구현이 어렵다는 문제점을 가지고 있다. 따라서 실험에서는 사용이 되지 않은 선인출 데이터의 블록 주소만을 테이블에 저장하며 시뮬레이션이 수행되는 동안에 출현하는 모든 미사용 선인출 주소가 저장될 수 있을 정도로 테이블의 엔트리 수를 충분히 크게 하여 구현하였다.

기존 PHT2bSC 기법은 E_N(엔트리 수) x 2bit로 테이블의 구조가 간단하다는 장점을 가지고, PHT1bSC 기법은 E_N(엔트리 수) x 1bit의 비용이 요구된다. 이 논문에서 제안하는 PBALT 기법은 E_N(엔트리 수) x n(블록 주소 비트 수) 만큼의 비용이 요구되는데, 기존 PHT2bSC 기법과 비교하면 오버헤드 부분이 된다.

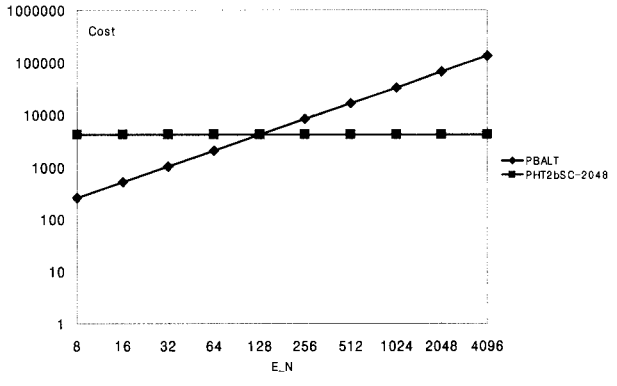
기존 연구와 제안하는 필터링 기법의 하드웨어 비용 및 성능을 비교하기 위해 캐시 크기는 64k, 블록 크기는 32byte, 엔트리 수(E_N)는 64k/32 = 2048, 블록 주소 비트 수는 32bit라 가정한다.

(그림 7)는 기존 PHT2bSC 기법과 제안하는 PBALT 기법의 하드웨어 비용을 비교한 것이다. 엔트리 수가 2048일 때 PHT2bSC 기법에서 요구되는 비용은 2048 x 2 = 4096이고,

〈표 1〉 선인출 필터 기법들의 특징 비교

선인출 필터 기법	특징	단점	H/W 비용
PHT2bSC 기법	Hasing, 2bit Saturation counter 사용 필터 구조가 간단	N:1 mapping에 따른 정확성 결여, 잠김현상 발생	E_N * 2bit
PHT1bSC 기법	Hasing, 1bit Saturation counter 사용 필터 구조가 간단, 잠김현상 완화	N:1 mapping에 따른 정확성 결여	E_N * 1bit
PBALT 기법	1:1 mapping에 따른 정확한 필터링	블록주소 전부를 저장하므로 엔트리의 크기가 커짐	E_N * n
CBAT 기법	과거 이력 정보를 이용하는 경우에서 가장 이상적인 방법	블록주소 전부를 저장(구현 불가)	2^n * 1bit

• n : Block address 비트 수 , E_N : entry의 수



(그림 7) h/w 비용 비교

PBALT 기법에서 요구되는 비용은 2048 x 32 = 65536 이 된다. y축의 PHT2bSC의 비용(cost)을 고정시키고 x축의 PBALT 엔트리 수를 증가시킬 때 두 선이 만나는 지점이 테이블 용량이 같아지게 된다. 즉, 그림에서 PBALT의 엔트리 수가 128개인 경우 동등한 조건이 된다. 만약 PBALT 기법의 엔트리 수가 128개인 경우 기존 PHT2bSC 기법과 캐시미스 수를 비교하여 더 적게 나온다면 동등한 기준에서 유용성이 있다고 할 수 있고, 또한 동등한 기준에서 뿐만 아니라 더 적은 엔트리 수를 가지고도 기존 연구보다 캐시 미스 수가 더 적게 나온다면 성능이 더 좋다고 말할 수 있다.

4. 시뮬레이션 및 성능 분석

4.1 시뮬레이션 환경

필터링을 적용한 선인출 데이터의 효과를 분석하기 위하여 DEC사의 ATOM 시뮬레이터[12]를 이용하여 적재/저장 명령어에 대한 트레이스 구동 시뮬레이션을 수행하였다. 트레이스에는 메모리 참조 명령어에 대하여 load 혹은 store인지 여부, 그리고 필요한 피연산자의 유효 주소, PC값으로 구성되어 있다. ATOM 시뮬레이터의 출력 결과인 명령어 트레이스를 캐시 시뮬레이터의 입력으로 받아 캐시의 성능 및 선인출 필터링의 효과를 분석한다.

캐시 시뮬레이터는 위스콘신 대학에서 개발한 Dinero III [13]을 기반으로 선인출 알고리즘(OBL, RPT, Correlation)과 필터링 방식들을 추가하여 사용하였다. 명령어 캐시와 데이터 캐시가 분리되어 있는 캐시 구조를 실험 대상으로 하였고, 데이터 캐시의 성능만 측정하였다. 캐시 시뮬레이터는 캐시 크기, 블록 크기, 사상 함수 등의 매개변수를 입력으로 캐시 분석 결과를 생성한다. 필터 테이블의 엔트리 수는 캐시 라인크기로 하였다. 즉, CBAT를 제외한 PHT2bSC, PHT1BSC, PBALT 기법들에서 엔트리 수를 동등한 크기로 하여 실험하였다.

<표 2>에 분석 모델을 위한 이 연구와 관련된 캐시 시뮬레이터의 매개 변수들과 값을 제시하였다. 대표적인 멀티미디어 벤치마크 프로그램인 MPEG(mpeg2enc, mpeg2dec)[14]과 SpecInt2000[15]의 일반 벤치마크 프로그램인 parser, crafty, vortex, gzip을 대상으로 1천만번의 메모리 참조 명령어에 대하여 실험하였으며, <표 3>는 벤치마크 프로그램들의 특징을 나타낸다.

〈표 2〉 분석 모델을 위한 매개 변수의 값

Parameter	Description
D-cache size	4K~1M
Block size	32byte
Word size	4byte
Associativity	Direct mapped
Bus width	Block size
History table entry size	Cache set size

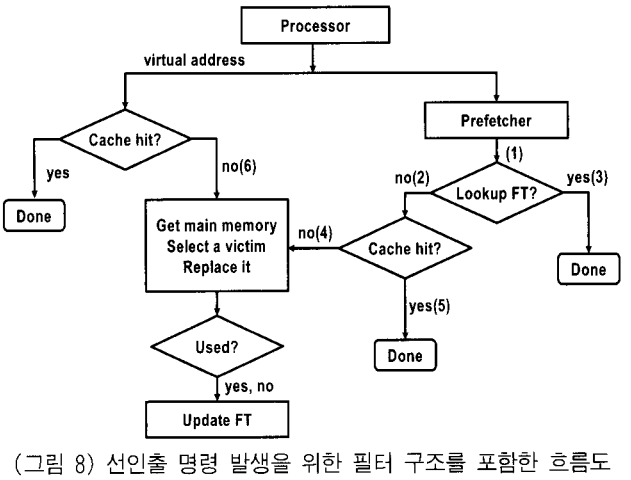
〈표 3〉 벤치마크 프로그램 특징

Bench program	Input file	Description
vortex	bendian.raw	Object-oriented Database
gzip	tar file	Compression
parser	2.1.dict	Word processing
ctafty	reference input	Game Playing : Chess
mpeg2encorder	pirates.par	digital video와 audio 압축에 관한 표준 도구
mpeg2decoder	meil6v2.m2v	

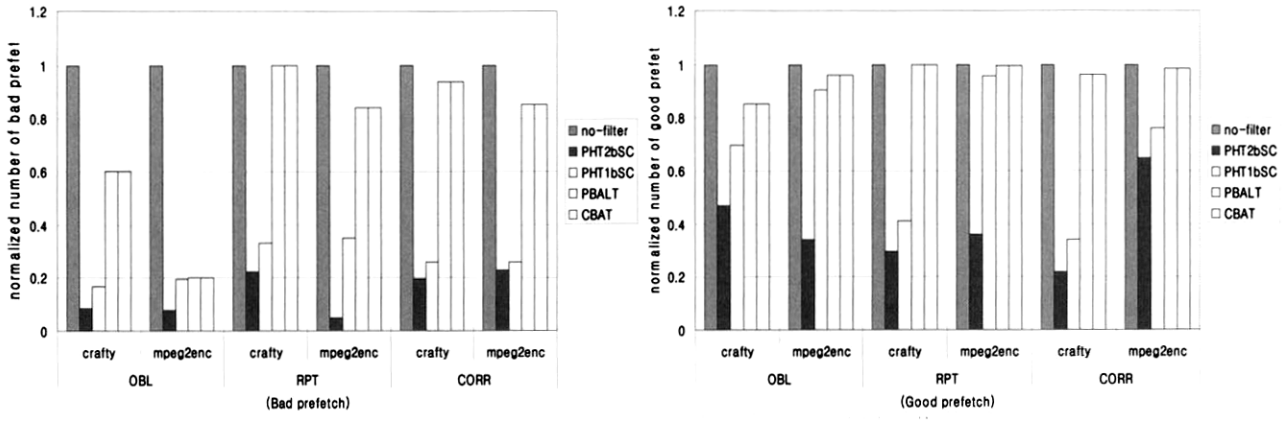
4.2 시뮬레이션 흐름도

(그림 8)은 요구 인출과 선인출 명령을 발생하기 위해 필터 구조를 포함한 경우의 필터링과 갱신 과정에 대한 흐름도이다. 각 번호에 대한 내용은 다음과 같다.

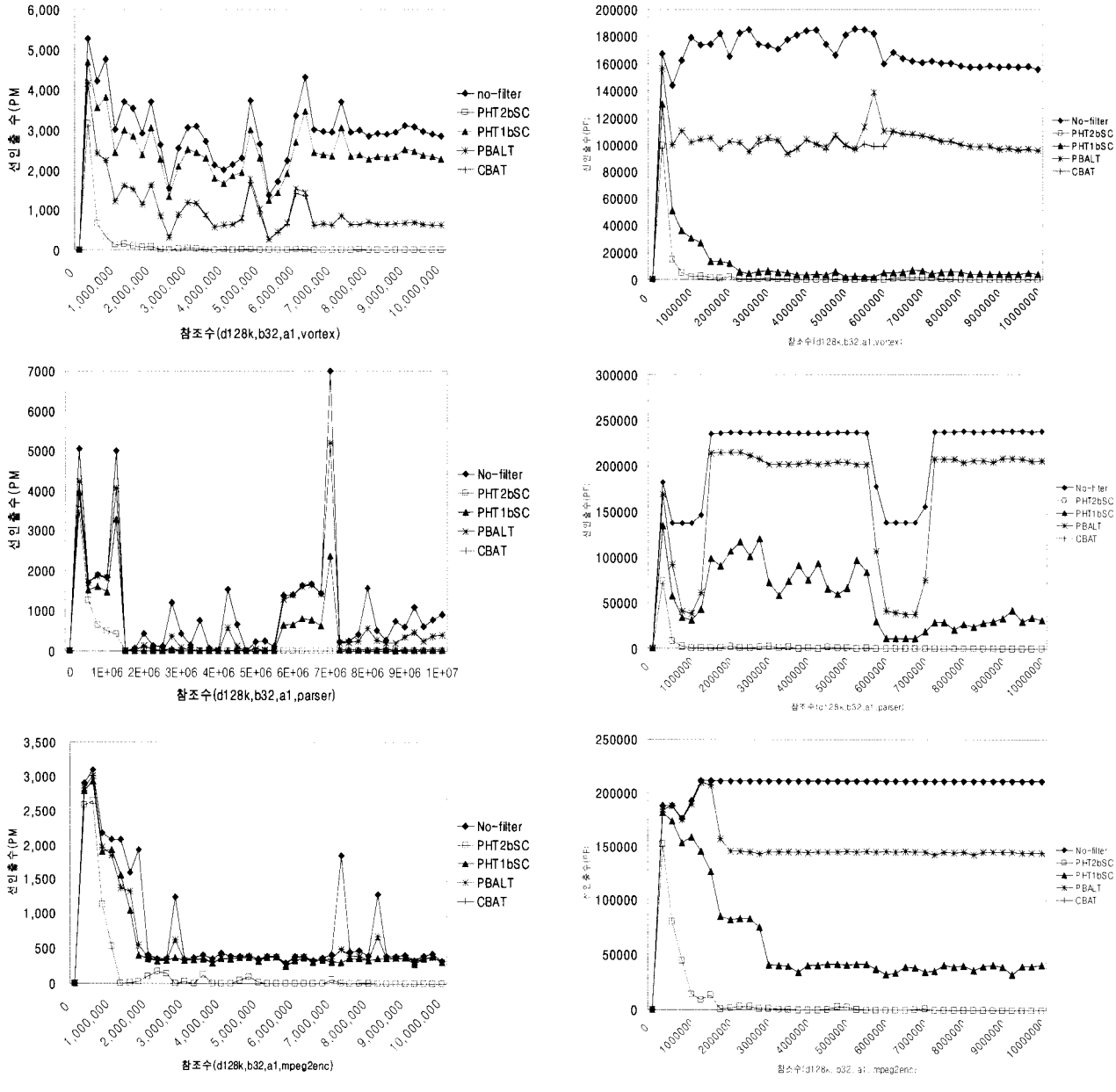
(1) Prefetcher : 각 선인출 알고리즘에 의해 계산된 전체 선인출할 데이터의 개수
(2) Miss_filter : 필터 테이블을 참조하여 필터링 조건에 만족하지 않은 경우의 수
(3) Hit_filter : 필터 테이블을 참조하여 필터링 조건에 만족하는 경우의 수
(4) C_M : 캐시를 참조하여 미스가 발생한 수로 실제로 선인출 명령을 발생



(그림 8) 선인출 명령 발생을 위한 필터 구조를 포함한 흐름도



(그림 9) good과 bad 선인출 수 비교(no-filter에 대해 정규화)



(a) PM(Prefetch Miss)수 비교

(b) PF(Prefetch Fetch)수 비교

(그림 10) 시간대별 선인출 수

(5) C_H : 캐시를 참조하여 히트가 난 경우로 실제로 선인출 명령을 발생시키지 않음

(6) D_M : 프로그램이 실행하면서 데이터 액세스를 필요로 하였으나, 캐시에서 미스가 발생한 수. D_M을 프로그램의 전체 메모리 액세스 수로 나누면 캐시 미스율이다.

4.3 시뮬레이션 결과 및 성능 분석

4.3.1 필터링 정확도 분석

필터링 정확도라고 하는 것은 불필요한 선인출은 최대한 억제하는 대신 유용한 선인출에는 영향을 주지 않는 것이라고 할 수 있다. 따라서 각각의 필터링 기법을 적용한 결과와 필터링을 적용하지 않은 경우와 비교함으로써 필터링 정확도를 평가해 볼 수 있다. 여기에서 불필요한 선인출은 선인출을 한 뒤 캐시에서 사용이 되지 않고 축출된 수를 말하며 유용한 선인출은 캐시로 선인출을 한 뒤 사용이 되고 축출된 수를 말한다.

(그림 9)는 대표적인 두 가지 벤치마크(일반 벤치마크인 crafty와 멀티미디어 벤치마크인 mpeg2enc)에 대해서 각 선인출 기법에서의 각각의 필터링 결과를 보여준다. 각각의 그래프는 불필요한 선인출(bad)과 유용한 선인출(good)의 수를 필터링 하지 않은(no-filter) 결과에 대해서 정규화한 값이다.

전반적으로 기존 선인출 해싱 테이블 2bitSC 기법(PHT2bSC)이 good 및 bad 선인출 모두 가장 낮은 값, 즉 가장 많이 필터링 되었음을 알 수 있다. 완전 블록 주소 테이블 기법(CBAT)의 경우, 원래의 선인출 수가 많지 않은 RPT와 Correlation에 대해서는 good과 bad 선인출 모두 크게 변하지 않았다. 그러나 선인출 개수가 많은 OBL에 대해서는 good 선인출은 대부분이(70%이상) 유지되는 반면, bad 선인출은 20~40%로 줄어들었음을 알 수 있다.

한편 선인출 블록 주소 참조 테이블 기법(PBALT)은 테이블 크기가 제한적임에도 불구하고 CBAT의 경우와 거의 동일한 결과를 보였다. 필터링 테이블의 크기는 CBAT 기법을 제외하면 모두 캐시의 라인 수와 동일하게 하여 실험하였다. 즉, (그림 9)의 경우 캐시 크기는 128Kbyte이고 블록 크기는 32byte이므로 필터링 테이블의 엔트리 수는 4096개로 하였다. PHT1bSC 기법의 경우 OBL과 RPT의 mpeg2enc의 경우 good에 비해 bad의 개수가 상대적으로 더 많이 감소되었다.

요약하면 기존 PHT2bSC 기법의 경우 전체 선인출 수는 가장 많이 감소하였으나 평균적으로 bad 선인출보다 good 선인출이 더 많이 감소하였으므로 필터링 정확도는 좋지 않다고 할 수 있다. PHT1bSC 기법의 경우는 기존 방식과 유사하나 선인출 수가 많은 OBL의 경우는 정확도가 약간 높았다. 또한 완전 블록 주소 테이블 구조(CBAT)와 선인출 블록 주소 참조 테이블 구조(PBALT)는 동일한 성능을 보이는 데, 특히 선인출 수가 많은 OBL의 경우는 필터링 정확도를 가짐을 알 수 있었다.

4.3.2. 시간에 따른 선인출 특성 분석

PHT2bSC 기법의 경우에서는 잠김 현상으로 인해 거의 모든 선인출이 일정 시간 이후 대부분 억제 될 것으로 예측하였다. 이와 같은 사실을 검증하기 위하여 시간에 따른 선인출 수를 (그림 10)과 같이 일반 벤치마크 vortex, parser와 멀티미디어 벤치마크 mpeg2enc에 대해서 분석하였다.

(그림 10)은 메모리 참조 수행에 따른 시간대 별 선인출 수를 측정한 것이다. (그림 10)(a)는 PM(Prefetch Miss) 수를 나타낸 것으로 이 수는 필터링 조건에 만족하지 않아 선인출 명령을 수행할 때, 해당 데이터가 캐시에서 미스가 발생하여 실제로 선인출 명령을 수행한 수를 나타낸다. 그림에서 볼 수 있듯이 기존 PHT2bSC 기법은 초기부터 급격히 감소하기 시작하여 120만회의 메모리 참조 이후부터는 거의 선인출을 하지 않게 된다. 반면, 제시된 필터링 기법들에서는 선인출 수가 필터링하지 않은 경우에 비해 전체적으로 감소되어 있을 뿐 시간에 따라서 균등한 값을 유지하고 있다. 따라서 시간이 지날수록 기존 연구에서는 선인출을 전혀 하지 않은 경우와 같이 캐시미스가 계속 늘어나서 선인출을 전혀 하지 않은 경우와 마찬가지가 될 것이다. (그림 10)(b)는 PF(Prefetch Fetch)의 수로 필터링 조건에 만족하지 않아 선인출 큐에 저장된 선인출 수를 나타낸다. 전반적으로 PM수에 비해 일정한 값을 가짐을 볼 수 있다.

본 실험에서는 일천만개의 트레이스를 사용하였으나, 그 수가 더 늘어날수록 기존 방식의 good 및 bad 선인출 수는 0에 가까워질 것이다.

기존 연구의 카운터 비트 수를 1로 줄인 선인출 해싱 테이블 1bitSC 기법(PHT1bSC)의 경우, 해싱을 사용함에도 불구하고 시간이 지나더라도 선인출의 수가 기존 연구에서와 같이 0이 되어 버리지 않고 일정하게 유지되는 것으로 보아, 기존 연구에서의 잠김 현상이 해싱 뿐 아니라 2비트 이상의 포화 카운터를 사용하였기 때문인 것을 알 수 있다. 따라서, 해싱을 사용하더라도 좀 더 복잡한 처리 방식을 접합할 경우 유용한 필터링 성능을 얻어낼 수도 있다는 것을 예측할 수 있다.

4.3.3 선인출 블록 주소 참조 테이블의 성능 분석

4.3.3.1 캐시 미스율 비교

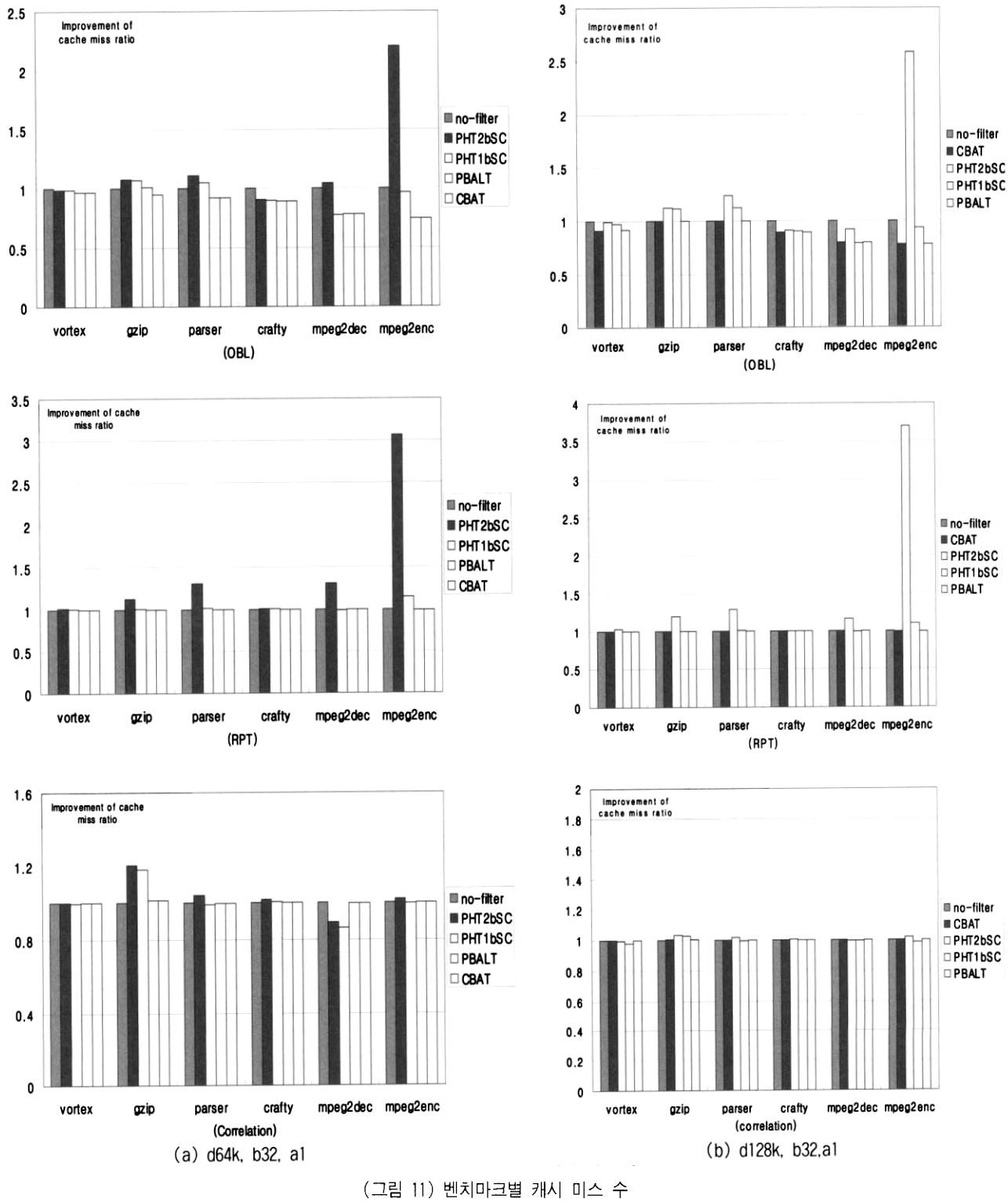
위에서 분석한 선인출 특성에 의하여 캐시에 전체적인 성능이 어떻게 영향을 받는지를 분석해 보았다.

(그림 11)는 필터링을 하지 않은 경우(no-filter)의 미스율을 기준으로 각 필터링 기법들의 미스율에 대한 성능 향상도를 다음의 식 (1)과 같이 정의하여 그래프로 나타낸 것이다.

$$E_{CM\_ratio} = \frac{M_{filtering}}{M_{no\_filter}} \qquad \text{식 (1)}$$

식(1)에 보인 성능 향상도 E_{CM_ratio}의 결과가 1보다 작으면 필터링을 하지 않은 경우보다 미스율이 낮아서 성능이 향상되었음을 의미하며, 1보다 큰 경우에는 필터링을 하지 않은 경우보다 미스율이 증가되어 필터링 효과가 떨어짐을 의미한다.

실험 결과 하드웨어 선인출기에 의해 선인출이 개시되는 개수가 가장 적은 Correlation 기법 선인출의 경우는 필터링

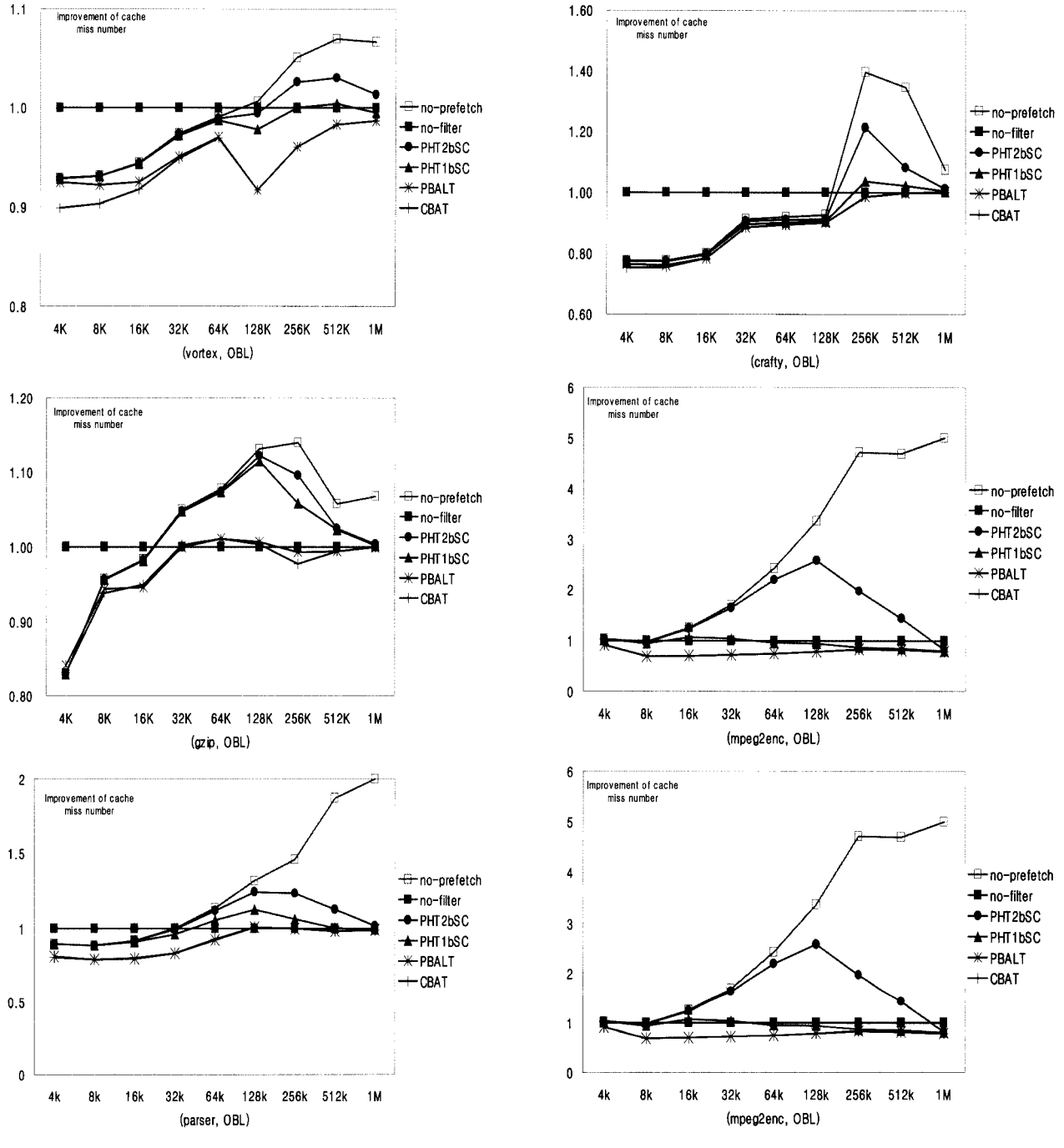


(그림 11) 벤치마크별 캐시 미스 수

방식에 따라서 큰 차이가 나지 않는다. 단지, PHT2bSC 기법이 다른 기법들보다 미스율이 미약하게 큰 것을 알 수 있다. 다음으로 RPT의 경우 PHT2bSC 기법은 가장 미스율이 높고, 그 다음으로 PHT1bSC 기법이 약간 높으며, 나머지 구조들은 비슷한 결과를 보인다. OBL의 경우에는 벤치마크에 따라서 crafty와 mpeg2dec의 경우에는 필터링한 결과가 하지 않은 결과에 비해 모두 더 낮은 미스율을 보인다.

한편 PBALT 기법과 CBAT 기법은 전반적으로 필터링을 하지 않은 경우에 비해 최대 22% 향상된 결과를 보인다. 평균적으로 PHT2bSC 기법을 기준으로 PBALT 기법이 7.9% 캐시 미스율이 감소하였다.

(그림 12)는 필터링을 하지 않은 경우(no-filter)의 미스 수



(그림 12) 캐시 크기에 따른 미스 수 비교

를 기준으로 각 필터링 기법들의 미스 수에 대한 성능 향상도를 식 (2)과 같이 정의하여 그래프로 나타낸 것이다.

$$E_{DM} = \frac{M_{filtering}}{M_{no\_filter}} \qquad \text{식 (2)}$$

실험 결과, 캐시 크기가 커짐에 따라서 필터링한 경우의 미스 수는 필터링하지 않은 경우에 비해 점점 더 증가한다. 이것은 필터링하지 않은 경우 불필요한 선인출로 인한 캐시의 오염이 캐시 크기가 커짐에 따라 점점 그 영향이 줄어드는 반면, 필터링한 경우에서는 필터링으로 인한 유용한 선인출의 감소로 인한 미스율 증가가 두드러지게 나타나기 때문이다.

유용한 선인출의 감소가 거의 없는 PBALT 기법과 CBAT 기법의 경우에는 캐시 크기가 증가함에 따라 상대적으로 미스율이 증가하는 경향을 보이기는 하지만 그 값이 필터링하지 않은 경우보다는 항상 작다. 기존의 PHT2bSC 기법의 경우 캐시 크기가 작을 때에는 미사용된 선인출이 더욱 증가하여 이로 인한 대부분의 선인출이 잠김 상태에 있기 때문에 선인출 하지 않은 경우와 미스율이 거의 같음을 알 수 있다.

PHT1BSC 기법의 경우 캐시 크기가 작을 때에는 PHT2bSC

〈표 4〉 벤치마크별 메모리 참조 지연 시간

(a) d64k, b32

OBL

d64k,b32,a1	vortex	gzip	parser	crafty	mpeg2dec	mpeg2enc
no-filter	15.9417	6.7238	1.6167	2.2249	0.0342	0.2250
PHT2bSC	15.7712	7.2188	1.7994	2.0181	0.0359	0.4958
PHT1bSC	15.7434	7.2091	1.7007	2.0020	0.0266	0.2166
PBALT	15.4715	6.7991	1.4948	1.9887	0.0268	0.1685
CBAT	15.4595	6.3678	1.4952	1.9887	0.0268	0.1685

RPT

d64k,b32,a1	vortex	gzip	parser	crafty	mpeg2dec	mpeg2enc
no-filter	15.6099	6.3678	1.3536	2.0180	0.0279	0.1516
PHT2bSC	15.7680	7.1761	1.7712	2.0249	0.0365	0.4639
PHT1bSC	15.6243	6.3744	1.3767	2.0193	0.0275	0.1738
PBALT	15.6049	6.3678	1.3535	2.0180	0.0279	0.1501
CBAT	15.6049	6.3678	1.3535	2.0180	0.0279	0.1501

CORR

d64k,b32,a1	vortex	gzip	parser	crafty	mpeg2dec	mpeg2enc
no-filter	15.7610	5.9903	1.7103	2.0066	0.0794	0.5187
PHT2bSC	15.7821	7.2093	1.7817	2.0432	0.0707	0.5288
PHT1bSC	15.7051	7.0780	1.6932	2.0187	0.0685	0.5171
PBALT	15.7598	6.0678	1.7067	2.0061	0.0790	0.5198
CBAT	15.7598	6.0679	1.7067	2.0062	0.0790	0.5198

(b)d128k, b32

OBL

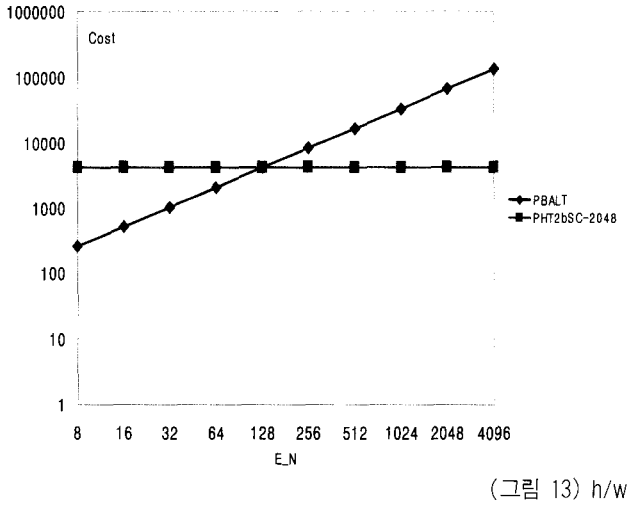
d128k, b32,a1	vortex	gzip	parser	crafty	mpeg2dec	mpeg2enc
no-filter	3.2196	3.4059	1.0154	2.1202	0.0236	0.1350
PHT2bSC	3.2024	3.8222	1.2617	1.9317	0.0217	0.3484
PHT1bSC	3.1482	3.7955	1.1413	1.9166	0.0185	0.1271
PBALT	2.9531	3.4284	1.0208	1.9096	0.0188	0.1061
CBAT	2.9523	3.4190	1.0208	1.9096	0.0188	0.1061

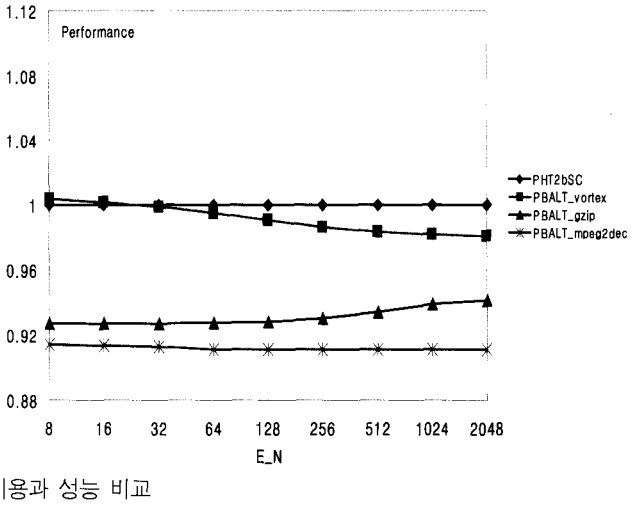
RPT

d128k, b32,a1	vortex	gzip	parser	crafty	mpeg2dec	mpeg2enc
no-filter	3.1088	3.1745	0.9744	1.9377	0.0196	0.0923
PHT2bSC	3.2049	3.7862	1.2615	1.9434	0.0227	0.3408
PHT1bSC	3.1099	3.1771	0.9825	1.9392	0.0194	0.1009
PBALT	3.1044	3.1744	0.9744	1.9377	0.0196	0.0916
CBAT	3.1044	3.1744	0.9744	1.9377	0.0196	0.0916

Correlation

d128k, b32,a1	vortex	gzip	parser	crafty	mpeg2dec	mpeg2enc
no-filter	3.2408	3.7028	1.2693	1.9548	0.0641	0.4052
PHT2bSC	3.2189	3.8232	1.2891	1.9607	0.0639	0.4119
PHT1bSC	3.1737	3.8008	1.2628	1.9585	0.0638	0.3983
PBALT	3.2366	3.7167	1.2692	1.9546	0.0641	0.4042
CBAT	3.2366	3.7167	1.2692	1.9546	0.0641	0.4042





(그림 13) h/w 비용과 성능 비교

기법과 거의 같은 값을 가지나 캐시 크기가 커짐에 따라 그 차이가 커짐을 알 수 있다. 즉, PHT1bSC의 경우에도 캐시 크기가 작은 경우 미사용된 선인출이 많으면 대부분의 선인출이 잠김 상태에 있는 것을 알 수 있다.

PHT2bSC의 경우 캐시 크기가 128Kbyte보다 커지게 되면 미스율이 다시 감소하는데, 이것은 축출되어 나가는 데이터 수가 급격히 감소하기 때문에 잠김 상태에 있는 엔트리가 거의 없어지기 때문이다.

일반적으로 일반 벤치마크의 경우에는 64Kbyte 이하의 캐시 크기에 대하여 PBALT 기법의 필터링 기법을 사용하는 것이 유리하며, 멀티미디어 벤치마크에서는 캐시 크기에 관계없이 PBALT 기법을 사용하는 것이 유리하나, 256Kbyte 이상에서는 PHT1bSC 기법도 거의 같은 성능을 낼 수 있다.

한편 기존 연구인 PHT2bSC 기법은 캐시 크기가 1Mbyte로 매우 큰 경우를 제외하고는 제안된 방식에 비해 성능이 낮으며 일반 벤치마크의 32Kbyte 이하의 캐시 크기를 제외하면 필터링 하지 않은 경우에 비해서도 항상 나쁨을 알 수 있다.

제안하는 기법과 CBAT 기법은 그래프의 위치가 가장 아래에 있으며, 기존 연구와 비교하여 캐시 미스 수가 낮음을 알 수 있다.

4.3.3.2 메모리 참조 지연 시간 비교

<표 4>은 각 벤치마크 프로그램을 수행할 때 평균 메모리 참조 지연 시간(MADT)을 다음의 식(4.3)과 같이 정의하여 계산한 것이다.

식 (3)은 모든 메모리 참조 명령어의 수행 시 캐시 미스에 의해 지연이 일어나는 사이클의 총합을 전체 수행된 명령어

의 수로 나눈 것이다. 캐시에서 히트가 발생했을 때 참조하는 시간은 1 cycle로, 캐시에서 미스가 발생했을 때 메모리를 참조하는데 걸리는 시간은 150 cycle로 가정하였다.

$$MADT = \frac{tatal\ delay\ in\ accessing\ memory}{number\ of\ memory\ reference\ instructions} \quad 식\ (3)$$

- 캐시 미스 시 : memory latency time = 150 cycle
- 캐시 히트 시 : 1 cycle

<표 4>에서 선인출 수가 많은 OBL 선인출 기법의 경우 필터링을 하지 않는 no-filter와 비교하여 기존 PHT2bSC 기법은 3.4%, PHT1bSC 기법은 1.0% MADT가 증가하였으며, PBALT 기법은 3.5% 감소하여 제안하는 기법이 전체 수행 시간에 있어서 성능이 향상되었음을 알 수 있다.

전체 선인출 알고리즘에서 평균적으로 No-filter을 기준으로 PHT2bSC 기법은 MADT가 5.3%, PHT1bSC 기법은 1.4% 각각 증가하고, 제안하는 PBALT 기법의 경우에는 MADT가 1.1% 감소하였다.

4.3.4 오버헤드 비교 분석

기존 PHT2bSC 기법과 제안하는 PBALT 기법의 오버헤드를 비교하기 위해 다음 실험을 하였다. 기존 PHT2bSC 기법은 테이블의 구조가 간단하다는 장점을 가지며, 하드웨어 비용을 계산하면 엔트리 수(E_N) x 2 bit가 된다. 이와 비교하여 제안하는 PBALT 기법은 엔트리 수(E_N) x 블록 주소 비트수(n)로 각 엔트리의 크기가 커지는 문제점이 있는데, (그림 13)에서 PBALT 구조의 엔트리 수를 줄이더라도 기존 PHT2bSC 기법보다 캐시 미스 수가 더 적게 출력되는 것을 볼 수 있다. 예를 들어 엔트리 개수를 2048개라 하고, 블록 주소의 비트 수를 32라 할 때에 PBALT의 경우 엔트리 수가 64(=2048/32)개 x 2 = 128개인 경우 테이블의 용량이 PHT2bSC의 경우와 같아지게 된다.

(그림 13)은 캐시의 크기가 64Kbyte인 경우의 하드웨어 비용과 성능을 비교한 것이다. 그림에서 y축의 PHT2bSC의 cost값을 고정시키고 x축의 PBALT 엔트리 수를 증가시킬 때 두 선이 만나는 지점이 테이블 용량이 같아지게 된다. 즉, 그림에서 PBALT의 엔트리 수가 128개인 경우 동일한 하드웨어 비용이 된다. 각 벤치마크들에서 기존 PHT2bSC 기법의 캐시 미스 수를 기준으로 제안하는 PBALT 기법의 미스 수를 비교한 것이다. x축의 엔트리 수가 128개인 경우 기존 연구보다 제안하는 PBALT 기법의 모든 벤치마크에서 미스 수가 낮은 것을 볼 수 있다. 그리고 vortex인 경우에는 엔트리 수가 32개인 경우까지도 미스 수가 낮으며 나머지 벤치마크들에 대해서는 모든 엔트리 수에 대해서 기존 연구보다 미스 수가 낮음을 볼 수 있다.

5. 결 론

캐시 기억 장치에 사용되는 데이터 선인출 기법은 프로세서가 연산을 수행하는 동안 필요한 오퍼랜드를 미리 캐시로 적재해서 메모리 액세스 시간을 줄이는 효과적인 방법이다. 그러나 너무 적극적인 선인출은 캐시의 오염을 일으켜 캐시가 갖는 본래의 이점을 상쇄시킬 수 있다. 이러한 캐시 오염 문제를 해결하기 위해 선인출 필터링 방법을 사용한다.

가능한 필터링 방법들이 여러 가지가 있는데, 그 중에서 기존 연구나 이 논문에서는 과거 이력 정보를 이용하였으며, 블록 주소 정보만을 가지고 필터링을 수행하도록 하였다.

기존 연구인 PHT2bSC 기법은 블록 주소를 해싱을 하여 필터 테이블로 인덱스를 하므로 선인출한 데이터가 사용이 되지 않고 캐시에서 축출될 경우 동일한 인덱스를 갖는 블록 주소 전체의 선인출이 금지되며, 한번 00으로 금지가 된 인덱스 주소는 2회 이상 사용이 되고 캐시에서 축출이 되어야 다시 살아나게 되는데, 그러한 경우는 극히 적으므로 빠져 나오기가 어렵게 되는 즉, 잠김 상태가 됨을 시간에 대한 선인출 동작 분석을 통하여 알 수 있었다. 이러한 PHT2bSC 기법의 문제점을 완화하기 위해 한번이라도 참조가 될 경우 선인출을 할 수 있도록 한 방법이 PHT1bSC 기법이다. CBAT 기법은 필터 구조들을 비교하고 가장 바람직한 방식을 유도하기 위한 기준을 마련하기 위해서 제시하였으며 해싱을 하지 않고 모든 블록 주소들의 과거 이력 정보를 갖는다. 그러나 이 기법은 테이블 크기가 너무 커져서 구현이 불가능한 문제점을 가진다.

이에 본 논문의 주 아이디어로서 선인출 블록 주소 참조 테이블 기법을 제안하였다. 이 방법은 최근에 사용이 되지 않은 블록 주소만을 필터 테이블에 저장하도록 테이블 크기를 제한하였고, PHT2bSC 구조가 N:1 매핑인데 비해 1:1 매핑이므로 더 정확하게 필터링을 하는 장점을 가지게 되며, CBAT 기법과 거의 동일한 성능을 가짐을 실험을 통하여 보였다.

기존 연구와 성능을 비교하기 위한 지표로는 캐시 미스율과 메모리 참조 지연 시간을 사용하였으며, 일반 벤치마크와 멀티미디어 벤치마크 프로그램을 가지고 실험을 한 결과, 캐시 미스율은 모든 선인출 알고리즘과 벤치마크들에 대하여 기존 PHT2bSC 기법과 비교하여 제안한 PBALT 기법이 7.9% 감소함을 알 수 있었으며, MADT는 6.1% 감소하였다. 이 논문에서 사용한 벤치 마크들 뿐 만 아니라 다른 벤치마크들에서도 비슷한 결과를 볼 수 있었다. 이러한 결과들을 토대로 good 선인출의 수를 보존하면서 bad 선인출은 선택적으로 감소시키는 것이 전체 캐시 성능을 향상시킨다는 것을 보였다.

제안한 PBALT 기법의 오버헤드로 각 엔트리의 크기가 커지는 문제점이 있는데, 동등한 비교 기준을 가지도록 하기 위해 제안한 기법의 엔트리 수를 1/16배 이하로 줄이더라도 기존 연구보다 더 나은 성능을 보임을 실험을 통해 확인할 수 있었다. 해싱을 사용하는 경우에도 기존의 방식과는 다른 좀 더 지능적인 엔트리 관리 방법을 사용하면 바람직한 필터링 성능을 낼 수도 있음을 보였으며, 이를 포함하여 PC 정보를 효율적으로 이용할 수 있는 방법들이 추가적으로 이루어져야 할 것으로 보인다.

선인출의 정확도를 높이고 캐시의 오염을 방지하기 위해 이 논문에서 제안한 선인출 필터링 기법을 응용하여 웹이나 데이터베이스 분야의 선인출에 적용하고 모바일 컴퓨팅, 센서 네트워크에 적용을 위한 에너지 소모를 고려한 캐시 및 메모리 운영 방법에 대해 연구되어야 할 것으로 생각한다.

참 고 문 헌

[1] A. J. Smith, "Cache Memories," ACM Computing Surveys, 14:473-530, Sep., 1982.

[2] N. P. Jouppi, "Improving directed-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," Proc. of the 17th Annual International Symposium on Computer Architecture, pp. 364-373, May, 1990.

[3] D. Joseph and D. Grunwald, "Prefetching Using Markov Predictors," IEEE Trans. on computers, Vol.48, No.2, Feb., 1999.

[4] D. Joshep and D. Grunwald, "Prefetching Using Markov Predictors," in proc. Of the 24th Annual Intl. Symp. On Computer Architecture, pp.252-263, June, 1997.

[5] J. Kim, K. V. Palem and W-F. Wong, " A Framework for Data Prefetching using Off-line Training of Markovian Predictors," in Proc. IEEE Intl. Conf. on Computer Design (ICCD), pp.340-347, Sep., 2002.

[6] G. Hariprakash, R. Achutharaman, A. R. Omondi, "DSTRIDE : Data-cache miss-address-based stride prefetching scheme for multimedia processors," 6th Australasian Computer Systems Architecture Conference (AustCSAC'01), pp.62-70, Jan., 29-30, 2001.

[7] Y. Solihin, J. Lee and J. Torrellas, "Correlation Prefetching with a User-Level Memory Thread," IEEE Trans. Computers, Vol.14, No.6, June, 2003.

[8] J. L. Baer and T-Fu Chen, "An effective on-chip preloading scheme to reduce data access penalty," In Proceedings of Supercomputing '91, pp.176-186, Nov., 1991.

[9] T-Fu Chen and J-L Baer, "Effective Hardware-Based data prefetching for High-Performance Processors," IEEE Trans. Computers, Vol.44, No.5, pp.609-623, May, 1995.

[10] V. Srinivasan, E. S. Davidson and G. S. Tyson, " A Prefetch Taxonomy," IEEE Trans. Computers, Vol.53, No.2, pp.126-140, Feb., 2004.

[11] X. Zhuang and H-H S. Lee, "Hardware-based Cache Pollution Filtering Mechanism for Aggressive Prefetches," in Proc. IEEE Int. conf. on Parallel Processing, pp.286-293, Oct., 2003.

[12] A. Srivastava and A. Eustace, "ATOM : A System for Building Customized Program Analysis Tools," Proceedings of the ACM SIGPLAN 94, pp.196-205, 1994.

[13] M. D. Hill, "Dinero Ⅲ Cache Simulator," Technical Report, Department Computer Science, University of Wisconsin, Madison. 1990.

[14] Mediabenchmark program : http://cares.icsl.ucla.edu/appications.html

[15] SpecInt2000 benchmark program: http://www.spec.org/osg/cpu2000/CINT2000

[16] J. H. Lee, S. W. Jeong, S. D. Kim and C. C. Weems, "An Intelli- gent Cache System with Hardware Prefetching for High Performance," IEEE Trans. on computers, Vol.52, No.5, May, 2003.

전 영 숙

e-mail : yschon@hanmail.net

1996년 한남대학교 전자계산학과(학사)

1998년 한남대학교 컴퓨터공학과(석사)

2002년 충북대학교 컴퓨터과학과 박사 수료

관심분야 : 고성능 컴퓨터 구조, 병렬 처리, 메모리 시스템, 멀티미디어 시스템

이 병 권

e-mail : sonic474@msn.com

1999년 한밭대학교 전자계산(학사)

2002년 한남대학교 컴퓨터공학(석사)

2003년 충북대학교 전자계산대학원 박사과정

이 춘 희

e-mail : autostory@hanmail.net

1994년 충북대학교 컴퓨터과학과(학사)

2000년 충북대학교 교육대학원 전자계산 교육(석사)

2005년 충북대학교 일반대학원 전자계산 전공 박사과정 수료

관심분야 : 센서 네트워크, 생체인식, 병렬처리

김 석 일

e-mail : ksi@cbucc.chungbuk.ac.kr
1975년 서울대학교(학사)
1985년~1989년 미국 North Carolina State University(공학박사)
1975년~1995년 국방과학연구소 선임연구원
1990년~현재 충북대학교 전기전자컴퓨터공학부 교수
관심분야 : 병렬처리 컴퓨터 구조, 슈퍼 컴퓨팅, 이기종 분산처리, 시각장애 사용자 인터페이스

전 중 남

e-mail : joongnam@cbu.ac.kr
1990년 연세대학교 전자공학과(박사)
1996년~1998년 미국 텍사스 A&M University 연구원
1990년~현재 충북대학교 전기전자컴퓨터공학부 교수
관심분야 : 컴퓨터 구조, 임베디드 시스템