선인출 전용 캐시를 이용한 적극적 선인출 필터링 기법

전 영 숙[†] · 김 석 일[††] · 전 중 남[††]

요 약

캐시 미스에 의한 메모리 참조 명령어는 응용 프로그램의 고속 수행을 방해하는 주 원인이다. 캐시 선인출 기법은 캐시 미스에 따른 지연시간을 줄이는 효과적인 방법이다. 그러나 너무 적극적으로 선인출을 할 경우에는 캐시 오염을 유발시켜 오히려 선인출에 의한 장점을 상쇄시킨다. 본 연구에서는 선인출로 인한 캐시의 오염을 줄이기 위해 필터 테이블을 참조하여 선인출 명령을 수행할 지의 여부를 동적으로 판단하는 적극적 선인출 필터링 기법을 제시한다. 정교한 필터링을 위하여 저장되어 있는 불필요한 선인출 데이터의 주소를 직접 사용하는 축출 주소 참조 방식을 제안하였다. 또한 동적 필터링의 정확성을 높이기 위하여 선인출 데이터의 캐시로부터의 출입을 증가 시키도록 작은 크기의 선인출 전용 캐시를 사용하였다. 선인출 전용 캐시의 사용으로 인해 유용한 요구 데이터들이 선인출 데이터들로 인하여 밀려나가지 않게 되었고, 또한 직접 주소 참조 방식을 통하여 필터링 정확성이 증가됨으로써 선인출 전용 캐시 내에도 유용한 선인출 데이터들만이 존재하게 되어 캐시 미스 수가 크게 감소되었다. 일반적으로 많이 사용되는 일반 벤치마크 프로그램과 멀티미디어 벤치마크 프로그램들에 대하여 실험한 결과, 제안된 방식의 캐시 미스율은 13.3% 감소하였고, 기존 방식에 비해 우수한 필터링 정확도를 가짐을 보였다.

An Active Prefetch Filtering Schemes using Exclusive Prefetch Cache

Young-Suk Chon[†] · Suk-il Kim[††] · Joong-nam Jeon[††]

ABSTRACT

Memory reference instruction caused by cache miss is the critical factor that limits the processing power of processor. Cache prefetching technique is an effective way to reduce the latency due to memory access. However, excessively aggressive prefetch leads to cache pollution and finally to cancel out the advantage of prefetch. In this study, an active prefetch filtering scheme is introduced which dynamically decides whether to commence prefetching after referring a filtering table to reduce the cache pollution due to unnecessary prefetches. For the precision filtering, an evicted address referencing scheme has been proposed where the filter directly compares the current prefetch address with previous unnecessary prefetch addresses stored in filtering table. Moreover, a small sized exclusive prefetch cache has been introduced to increase the amount of eviction of unnecessarily prefetched addresses to enhance the accuracy of dynamic filtering. The exclusive prefetch cache also prevents useful demand data from being pushed out by prefetched data, while the evicted address direct referencing scheme enables the prefetch cache to keep most of useful prefetch data within its small size. Experimental results from commonly used general and multimedia benchmarks show that the average cache miss ratio has been decreased by 13.3% by virtue of enhanced filtering accuracy compared with conventional schemes.

키워드 : 캐시 메모리(Cache Memory), 선인출 알고리즘(Prefetch Algorithm), 필터링(Filtering), 선인출 캐시 구조(Prefetch Cache Architecture)

1. 서 론

메모리 참조 지연시간을 줄이기 위하여 캐시 메모리가 도입되었는데, 이 경우 캐시 미스율을 감소시키는 것이 시스템 성능의 향상을 위하여 무엇보다도 중요하다. 캐시 구조의 선인출 기법은 프로세서가 사용할 것으로 예측되는 데이터를 실제로 요구하기 전에 미리 주기억장치에서 캐시로 인출하여 캐시 미스율을 감소시키는 방법이다. 이러한 캐시 선인출 방법으로는 OBL(one block-look-ahead) 방식[1], OBL 방식을 확장한 스트림 버퍼 선인출 방식[2], 마코프 예측기(Markov predictor) 방식[3-6], 상관관계(correlation) 방식[7], 참조예측표(RPT : Reference Prediction Table) 방식[7] 등이 있다. 이러한 하드웨어 캐시 선인출 방식들은 메모리 참조 지연시간을 줄일 수 있는 효과적인 방법이지만, 너무 과도한 선인출의 사용은 비효율적인 선인출에 의해 캐시 안의 유용한 데이터를 교체하여 캐시 오염 현상을 유발하고 버스 트래픽을 증가 시켜, 오히려 전반적인 성능을 감소시킬 수 있다.

※ 본 연구는 한국과학재단 목적기초연구(R05-2002-000-01470-0)와 2004년도 충북대학교 학술연구지원사업의 연구비 지원에 의하여 수행되었음.
† 준 회 원 : 충북대학교 대학원 컴퓨터과학과
†† 종신회원 : 충북대학교 전기전자컴퓨터공학부 교수
논문접수 : 2004년 10월 11일, 심사완료 : 2005년 1월 3일

따라서 개시된 선인출들 중에서 유용한 것들만을 선택적으로 판별하여 불필요한 선인출로 인한 성능의 저하를 막기 위해서는 선인출을 적절하게 필터링할 필요가 있다. 선인출 필터링을 위한 기존 방법으로는 정적 필터 구조와 동적 필터 구조가 있다. Srinivasan등에 의해 제안된 정적 필터는 선인출에 의해 생성된 트래픽과 미스들에 근거하여 선인출들을 분류하는 광범위한 분류법을 사용하였다[10]. 비효율적인 선인출을 줄이기 위해 Zhuang등은 동적 필터링 방법을 제안하였으며[11], 이 방법은 선인출기가 생성한 선인출 주소의 일부를 가지고 필터 테이블을 참조하여 상태를 네 가지로 관리하여 필터링을 수행한다. 이 구조는 불필요한 선인출의 수를 매우 감소시켰으나, 필터링 효과를 강조하기 위하여 컴파일러에 의해 삽입된 선인출 명령어와 하드웨어 기반의 동적 선인출 명령어를 모두 포함하여 최초 선인출의 수를 증가 시켰으며, 하드웨어 기반의 선인출 방법만을 적용한 경우의 성능평가는 보이지 않았다. 또한 이 방법은 전체 선인출 수에만 중점을 두었고, 전체 성능 면에서는 고려하지 않았다.

하나의 캐시 내에 요구 데이터(demand data)와 선인출 데이터를 모두 저장하는 통상적인 경우에 있어서 캐시 오염으로 인하여 발생되는 캐시 미스는 다음과 같은 세가지 성분으로 구분될 수 있다. 첫째로는 과도한 선인출로 인하여 유용한 요구 데이터들이 밀려나갈 수 있다. 선인출 데이터는 통상적으로 요구 데이터에 비해 재사용성이 낮으므로, 매우 이상적인 선인출 필터링 방법을 사용하여 유용한 선인출만을 하게 되는 경우라 하더라도 이 성분은 없어지지 않는다. 둘째로는 불필요한 선인출 데이터로 인하여 유용한 선인출 데이터들이 밀려나가게 될 수 있다. 이 성분은 선인출 필터링의 정확성이 매우 높은 경우에는 제거될 수도 있다. 마지막으로, 요구 데이터들에 의해서 유용한 선인출 데이터들이 밀려나갈 수도 있게 된다. 이것은 요구 데이터의 재사용성이 상대적으로 낮은 일반 벤치마크의 경우 특히 심해지는데, 매우 정확한 하드웨어 선인출의 경우 그 성능을 크게 감소시키는 요인이 된다. 이러한 성분들은 캐시 크기가 상대적으로 작고 또한 associativity가 낮은 경우 더욱 심각해진다.

본 논문에서는 이러한 문제점들을 해결하기 위해, 요구 데이터와 선인출 데이터의 상호 충돌 또는 경쟁을 방지함으로써 유용한 요구 데이터와 유용한 선인출 데이터 모두를 최대한 보존하기 위하여 선인출 전용 캐시를 도입한다. 또한 추가된 선인출 전용 캐시의 크기를 최소화함으로써 오버헤드를 줄임과 동시에 불필요한 선인출 정보의 양을 증가 시켜, 축출 주소 직접 참조 방식의 동적 필터링의 정확성을 극대화할 수 있는 방식을 제안한다. 제안된 적극적 필터링 방식은 어떠한 종류의 하드웨어 선인출 방식에 대해서도 동적 적응할 수 있는 범용 선인출 필터링 방식이며, 실험을 통하여 그 성능을 검증하였다.

본 논문의 구성은 다음과 같다. 2장에서는 선인출 구조에 관한 기존 연구를 요약하고, 본 논문의 대상인 선인출 전용 캐시를 사용한 동적 필터 구조를 제안한다. 3장에서는 기존 필터 구조 및 제안하는 필터 구조에 대해 설명하고 선인출 전용 캐시를 도입함으로 인한 효과를 설명한다. 4장에서는 선인출 데이터의 정확도를 분석하고, 실험을 통한 성능 평가 결과를 제시한다. 마지막으로 5장에서는 본 논문에서 얻은 결과를 정리한다.

2. 선인출 전용 캐시를 갖는 선인출 필터링 구조

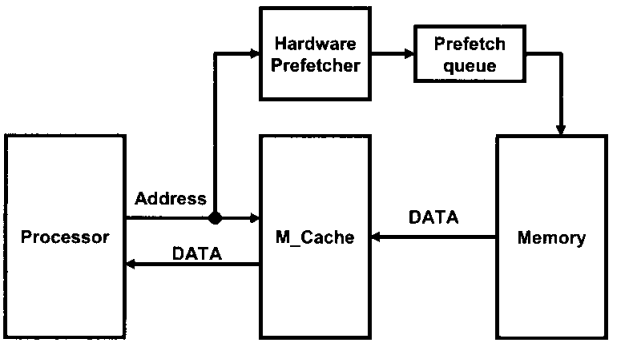
2.1 선인출 필터링

캐시 선인출은 프로세서가 명령어들을 수행하는 동안 미래에 사용될 것으로 예측되는 데이터를 미리 캐시에 인출함으로써 캐시 미스로 인한 지연시간을 줄이는 방법이다. 이러한 선인출 방식으로는 하드웨어를 이용하여 실시간으로 선인출을 개시하는 하드웨어 선인출 방법과 컴파일 시에 적절한 선인출 명령어를 삽입하는 소프트웨어 선인출 방법이 있다. (그림 1)은 하드웨어를 이용한 동적 선인출 구조를 보여준다. 프로세서가 캐시로 메모리 참조 명령을 전송할 때 그 메모리 주소는 동시에 선인출기(Hardware Prefetcher)에도 전달된다. 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. 이 구조는 하나의 데이터 캐시를 사용하여 요구 데이터와 선인출 데이터 모두를 저장하게 된다.

이와 같은 하드웨어 선인출 구조는 실제로 미래에 사용되어질 필요한 데이터 외에도 불필요한 데이터도 또한 선인출 함으로써 캐시 오염을 일으킬 수 있는 문제점을 갖고 있다. 이와 같이 불필요하게 선인출된 데이터는 유용한 요구 데이터 및 유용한 선인출 데이터를 캐시로부터 축출 시켜(evict) 캐시 미스율을 또로 증가 시키게 하는 요인이 된다. 따라서, 이러한 과도한 선인출을 줄이기 위해서는 원천적으로 정교한 하드웨어 선인출기를 사용하여 그 정확도를 증가 시켜야 하는데 이러한 정확한 선인출 방법을 위해서는 많은 연구가 이루어진 바 있으나[1-8], 벤치마크의 특성에 따라 그 성능이 매우 가변적이다.

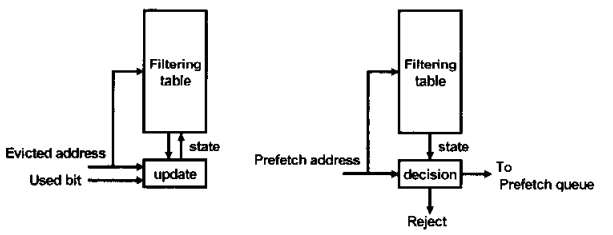
과도한 선인출을 줄이기 위한 다른 방법으로는 공격적인 선인출 방식을 사용하고 그 대신 개시된 선인출 중에서 실제로 미래에 사용될 선인출을 골라내어 최종적으로 유용한 선인출만을 통과시키는 선인출 필터를 사용하는 방법이 있을 수 있다. 이러한 접근 방식은 필터링 정확도를 높일 수만 있으면 요구(demand) 인출에 대한 미스율을 가장 많이 줄일 수 있고, 다양한 선인출 방식에 대하여 범용으로 적용할 수 있다는 장점이 있다. 이러한 필터는 미리 정해진 정적 필터일 수도 있고[10], 실행하면서 적절하게 그 특성이 변화해가는 동적 필터일 수도 있다. Srinivasan등에

의해 제안된 정적 필터는 선인출에 의해 생성된 트래픽과 미스들에 근거하여 선인출들을 분류하는 광범위한 분류법을 사용하였다. 정적 필터는 먼저 프로파일을 통해 오프라인으로 오염이 되는 선인출의 정보를 수집하여 이 정보를 통해 선인출을 수행한다. 이 프로파일 정보는 주어진 입력 데이터들에 대해 정확한 정보를 제공하지만, 워킹 세트가 변할 때 수행시간 동안에 동적 적응성이 결여되는 문제점이 있다.



(그림 1) 동적 데이터캐시 선인출 구조

본 논문에서는 어플리케이션의 종류와 관계없이 필터 특성이 적응하는 동적 필터만을 다루기로 한다. 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 않았었는지에 대한 정보는 캐시로부터 축출되는 선인출 데이터들이 캐시에 있는 동안 프로그램에 의해 참조되었었는지 아닌지의 여부로부터 얻어낼 수 있다. 즉, 캐시 내부에는 각 데이터가 선인출로 인한 데이터인지 여부와 참조된 내역이 각 데이터와 함께 저장되어 있어야 한다.



(a) 선인출 필터링 동작 (b) 선인출 필터의 갱신

(그림 2) 하드웨어 선인출 필터의 구조

(그림 2)는 동적 하드웨어 필터 구조를 보여준다. (그림 2) (a)에서와 같이 동적 필터 내부에는 과거의 선인출에 대한 결과(history)를 저장하는 필터링 테이블이 존재하여 선인출 전용 캐시에서 데이터가 축출될 때마다 필터 테이블을 적절하게 갱신한다. (그림 2) (b)에서와 같이 하드웨어 선인출기가 생성한 선인출 주소를 테이블로부터 참조(lookup)하여 필터링 조건에 만족하지 않는 경우에만 통과시켜 선인출 큐에 저장한다. 이와 같은 하드웨어 필터는 (그림 1)의 하드웨어 선인출기와 선인출 큐 사이에 들어가게 된다.

2.2 선인출 전용 캐시

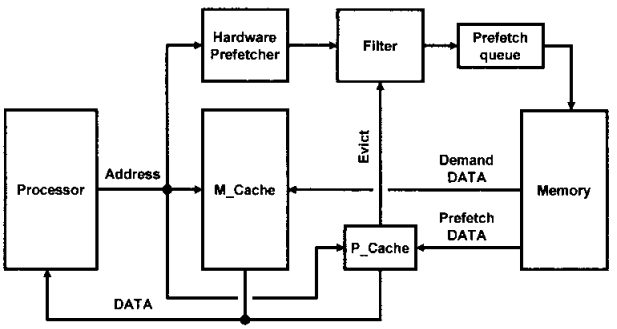
이와 같은 동적 선인출 필터는 과거의 선인출 결과를 바탕으로 하여 그 필터링 특성을 적응해 나가기 때문에 정확한 필터링 성능을 위해서는 가능한 한 많은 과거의 정보를 필요로 한다. 즉, 캐시로부터 축출되어 나오는 미 사용된 선인출 데이터의 수가 충분히 많을수록 통계적으로 더욱 정확한 필터로서의 특성을 얻을 수 있을 것이다. 또 한편으로는, 축출되는 선인출 데이터의 사용 여부가 해당 데이터의 선인출 정확도를 정확하게 반영하고 있어야 한다. 즉, 실제로는 유용한 선인출 데이터임에도 불구하고 여러 가지 요인에 의하여 미처 사용되지 않은 채로 축출된다면 필터는 이 데이터가 유용하지 않은 데이터라고 간주하여 이후의 선인출 시에는 필터링하여 버릴 것이다.

서론에서 설명한 바와 같이, 유용한 선인출 데이터가 부당하게 축출되는 요인은 두 가지가 있을 수 있다. 첫째로는, 불필요한 선인출 데이터가 너무 과도하게 많은 경우 유용한 선인출 데이터가 미리 축출될 수 있으며, 둘째로는 요구인출(demand fetch) 데이터가 상대적으로 많은 경우에도 또한 유용한 선인출 데이터가 미리 축출될 수 있다. 전자의 경우, 필터의 특성이 점차 최적값으로 다가가면서 점점 그 현상이 줄어들 수 있다. 하지만 이렇게 되기 위해서는 적은 양의 정보를 가지고도 필터의 특성이 조금이나마 개선될 수 있을 정도의 정확한 필터링 방식이어야 한다. 한편, 후자의 경우에는 시간이 지나도 그 현상이 없어지지 않는다. 즉, 유용한 선인출 데이터를 계속 필터링하게 될 것이다. 따라서 이러한 성분을 완전히 제거하여 필터의 특성을 최적으로 만들기 위해서는 근본적으로 요구 데이터와 선인출 데이터를 완전히 격리시켜야만 한다.

또한, 유용하지 않은 선인출에 대한 충분한 정보를 얻기 위해서는 캐시의 크기가 작을수록 유리하다. 그러나, 캐시의 크기를 작게 하면 한번 요구(demand)된 데이터를 모두 저장할 수 없기 때문에 이로 인하여 캐시 미스율이 매우 증가하게 된다. 이러한 이유로도 요구 데이터를 저장하는 공간과 선인출 데이터를 저장하는 공간을 서로 분리시켜야 할 필요성이 있다. 선인출되는 데이터는 요구 데이터에 비해 재사용성이 상대적으로 낮다고 가정하면, 우수한 필터링 방식을 갖춘다면 선인출 데이터를 저장하기 위한 공간은 매우 작아도 될 것으로 예측할 수 있다. 따라서 본 논문에서는 기존의 캐시(M_Cache : Main Cache)와는 별도로 매우 작은 크기의 선인출 전용 캐시(P_Cache : Prefetch Cache)를 구비하는 구조를 제안하였다.

(그림 3)은 또한 선인출 전용 캐시를 갖는 필터링을 적용한 하드웨어 선인출 구조를 보여주고 있다. 즉, 하드웨어 선인출기로부터 생성된 모든 선인출 명령을 선인출 큐

에 전달하는 것이 아니라, 이들 중 적절한 선인출만을 여과하여 실제 선인출을 일으키도록 선인출 필터가 추가된다. 필터 테이블은 선인출 전용 캐시에서 데이터가 축출될 때마다 적절하게 갱신한다. 요구 인출에 의한 데이터는 모두 메인 캐시에 저장되며, 선인출에 의한 데이터는 선인출 전용 캐시에만 저장된다. 프로세서가 캐시에 데이터를 요구할 때에, 메인 캐시와 선인출 전용 캐시가 동시에 검색되고 두 캐시 중 하나에 해당 데이터가 있을 경우 히트가 되며 프로세서로 전송된다. 두 캐시 모두에서 데이터가 발견되지 않는 경우 미스가 되며 해당 데이터가 주 기억장치로부터 메인 캐시로 채워진다. 따라서 어떤 데이터가 두 캐시 내에 모두 존재하는 일은 없게 되며, 두 캐시간에 데이터가 이동되지 않는다. 또한, 필터의 갱신을 위해서는 선인출 전용 캐시로부터 축출된 데이터만 사용되므로, 메인 캐시 내에는 추가적인 정보를 저장하지 않아도 된다.



(그림 3) 필터링을 적용하는 하드웨어 선인출 구조

이와 같은 선인출 전용 캐시 구조는 3장에서 설명할 제안된 필터링 방식과 결합되어 높은 필터링 성능을 가지므로, 선인출 전용 캐시의 크기가 매우 작더라도 문제가 없다.

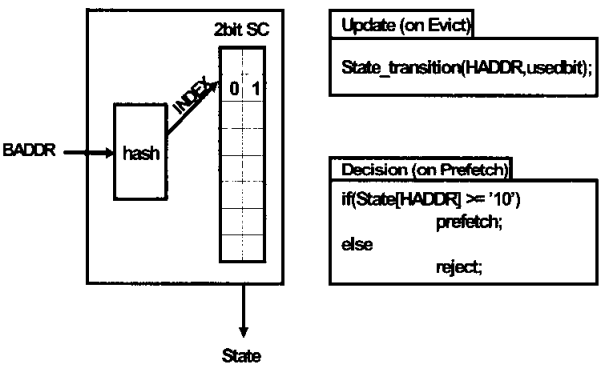
3. 선인출 필터링 방식

이 장에서는 필터링 효과를 높이기 위해 축출 주소 참조 구조를 제안하며, 분리된 캐시를 사용했을 때의 효과를 기존의 방식과 비교하기로 한다.

3.1 해시 필터 구조(Hash)

(그림 4)는 Zhuang등에 의해 제안된 동적 필터 구조이다. 이 구조는 서로 다른 데이터 주소에 대해서 같은 인덱스 주소를 공유하는 N : 1 매핑 구조를 보이고 있다. 해당 선인출 주소(BADDR : block address)는 해시 함수를 통과하여 필터 테이블에 인덱스를 한다. 인덱스된 위치의 엔트리 값은 2bit로 구성이 되어 있으며, 선인출 전용 캐시에서 축출될 때마다 그 상태를 변경시킨다. 선인출기가 생성한 선인출 주소의 인덱스 주소로 테이블을 참조하여 해당 엔트리의 상태값이 2이상인 경우에 선인출 명령을 수행하고, 그렇지 않은 경우에는 선인출을 하지 않는다.

이러한 기존 동적 필터 구조의 문제점은 다음의 두 가지로 요약될 수 있다. 첫째로는 해싱을 이용한 N대1 매핑에 따른 정확성이 결여된다는 점이다. 즉, 서로 다른 데이터 주소에 대해서 같은 엔트리를 공유하기 때문에 필터링을 해야 하는 경우와 하지 말아야 하는 것에 대한 정확한 판단을 하지 못한다(aliasing). 둘째로는 2bit 포화 카운터를 사용하기 때문에 결과적으로 불필요한 선인출과 유용한 선인출 모두를 억제 시켜 캐시 미스율을 증가시키게 된다. 2bit 포화 카운터를 쓰게 된 동기는 이 방식이 N대1 매핑을 사용하기 때문에 같은 엔트리를 공유하는 서로 다른 주소들의 선인출 유용도의 평균값을 구하기 위한 것이다. 그러나 같은 해시 주소를 갖는 주소들 중에서 최초 선인출 된 어떤 주소가 미 사용된 채로 축출된 이후에는 유용할 수도 있는 다른 주소들에 대한 선인출이 원천적으로 이루어지지 않기 때문에 카운터의 값은 의도와는 달리 평균값이 아닌 점점 0에 가까이 가게 된다. 즉, 엔트리를 공유하는 주소들 중에서 사용되지 않는 선인출 주소가 일부라도 포함된 경우 그 해당 주소들에 대한 선인출이 모두 제외된다. 이러한 현상은 카운터의 깊이가 중심값으로부터 2이상이기 때문에 더욱 심한데, 왜냐하면 한번 '00'이 되면 다시 선인출 할 수 있게 되기 까지는 이전에 미리 캐시로 선인출 하였던 데이터가 두 번 이상 사용이 되고 축출되어 나가야만 된다. 그러나 그러한 경우는 극히 적으므로 한번 금지된 인덱스 주소는 그 상태에서 빠져 나오기가 어렵게 되는, 즉, 잠김(lock) 현상이 일어나게 된다.



(그림 4) 해시 필터 구조(Hash)

해시 함수를 사용한 구조의 이러한 잠김 현상은 앞에서 설명한 것과 같이 불필요한 선인출 데이터가 너무 과도하게 많아서 유용한 선인출 데이터를 미리 축출 시키게 되는 경우 더욱 심화된다. 즉, N : 1 매핑으로 인하여 정확성이 결여되기 때문에 시간이 지나더라도 필터의 특성이 조금도 나아지지 않는다. 따라서 시간이 지나도 유용한 선인출 데이터를 계속 필터링하게 되고, 이로 인하여 사용이 되고 축출되는 데이터의 수는 급속히 줄어들게 되어 결국은 선인출 자체가 개시되지 않게 되는 것이다.

3.2 축출 주소 참조 구조(Evicted Address List : EAL)

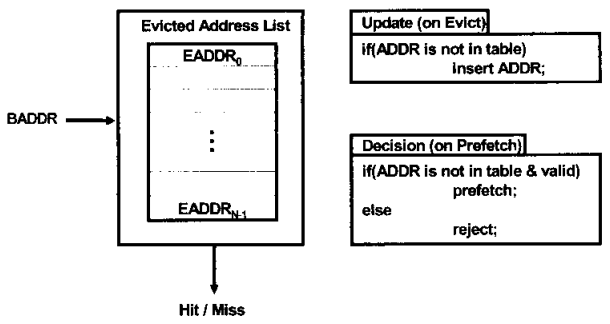
Zhuang에 의한 해시 필터 구조는 불필요한 선인출뿐 아니라 유용한 선인출도 무분별하게 감소시킨다는 단점이 있다. 즉, 선인출 필터가 가져야 할 바람직한 특성은 다음과 같은 것이다.

(1) 제거성 (rejection) : 불필요한 선인출 수는 최대한 감소시킨다.
(2) 통과성 (conserve) : 유용한 선인출 수는 거의 동일하게 유지한다.

제거성 외에도 보존성이 또한 중요한 이유는 유용한 선인출이 감소될 경우 선인출을 하지 않는 것과 마찬가지로 캐시 미스율이 증가하기 때문이다. 즉, 필터링의 원래 목적인 불필요한 선인출을 감소시키는 것은 이로 인한 캐시 오염을 막기 위한 것인데, 이를 위하여 유용한 선인출마저 감소시키게 된다면 캐시 미스율을 감소시키기 위한 선인출 본연의 목적과 상반되는 결과를 초래하는 것이다. 따라서 선인출 필터링의 성능을 가늠하기 위한 척도로서 다음과 같이 필터링 정확도를 정의할 수 있다.

$$filtering\ \ accuracy = \sqrt{rejectivit\ y \times conservity} = \sqrt{\frac{bad\ \ prefetch_{\ no\_filter} - bad\ \ prefetch}{bad\ \ prefetch_{\ no\_filter}} \times \frac{good\ \ prefetch}{good\ \ prefetch_{\ no\_filter}}} \quad (1)$$

(그림 5)는 이와 같은 필터링 정확도를 극대화하기 위하여 제안된 필터링 테이블 (Evicted Address List)구조이다. 선인출 전용 캐시로부터 축출된 사용되지 않은 주소값들을 유한한 크기의 테이블에 저장한다. 선인출기가 생성한 선인출 주소값이 이 필터 테이블 내에 존재하는 경우(Hit) 선인출 명령을 제거한다. 이 구조는 블록주소 전부를 테이블의 각 엔트리에 저장하므로 각 엔트리의 크기가 커지는 문제점이 있는 반면, 각 엔트리에는 하나의 주소만이 할당되며 이러한 1대1 매핑에 따른 정확한 필터링을 할 수 있는 장점이 있다. 즉, N대1 매핑에서와 같은 aliasing 문제가 생기지 않고, 제거하고자 하는 특정 블록 주소들만을 정교하게 필터링할 수 있다. 매우 이상적인 경우, 이러한 테이블의 크기가 무한대가 되면 미 사용된 모든 블록주소를 저장할 수 있게 될 것이다. 실제적인 경우 미 사용된 주소의 개수가 테이블 크기보다 더 커지게 되면 모든 미사용 주소를 포함할 수 없게 되어서 테이블의 각 엔트리간에 LRU나 FIFO 교체정책을 써야 한다. 본 논문에서는 LRU 정책을 사용하였다.



(그림 5) 축출 주소 참조를 갖는 필터 구조(EAL)

미 사용된 주소의 개수가 테이블 크기보다 더 커지게 되면 테이블에 저장된 지 오래된 엔트리들은 시간이 지나면서 교체되어 없어질 수 있는데, 이러한 성질은 간헐적으로 잘못 판단된 필터링 엔트리를 보정할 수 있는 기회가 될 수 있다. 즉, 유용한 선인출임에도 불구하고 여러 가지 원인에 의하여 사용되기 전에 미리 축출된 경우 필터는 이를 불필요한 선인출로 간주하게 되는데, 그 결과가 끝까지 남아있게 되면 선인출 효과를 감소시킬 수 있으므로 오래된 엔트리들은 시간이 지나면서 저절로 없어지게 하는 것이 좋다. 또한, 적절하게 선택되어 필터링 효과가 큰 엔트리의 경우에는 LRU 정책에 의해 테이블 내에서 계속 생존할 수 있게 되므로 테이블의 전체 크기는 주요한 불필요한 선인출들만을 포함시키기에 충분할 정도만 되면 필터링 성능에 큰 영향을 주지 않는다.

3.3 선인출 전용 캐시의 효과

이와 같은 축출 주소 참조 필터링 구조는 앞서서 언급된 선인출 전용 캐시 구조와 결합될 때에 그 성능이 최대로 발휘된다.

첫째, 선인출 전용 캐시의 크기만을 따로 줄일 수 있으므로 불필요한 선인출 데이터가 축출되는 양이 많아져서 필터의 특성을 적응시키기 위한 정보가 많아진다. 따라서, 필터링 정확도를 증가 시킬 수 있게 된다. 또한, 축출 주소 참조 구조의 경우 그 정교함으로 인하여 해시 필터 구조에 비해서 유용한 선인출의 보존도는 높은 반면 불필요한 선인출의 제거도는 낮아진다. 이러한 경우에는 축출되는 정보가 많을수록 보다 더 적극적인 필터링이 가능하다.

둘째, 요구 데이터와 선인출 데이터간에 완전한 분리가 이루어져 과도한 선인출로 인한 메인 캐시의 오염되지 않으며, 또한 상대적으로 요구 데이터 수가 많을 경우에도 이로 인하여 유용한 선인출 데이터가 축출되지 않는다. 유용한 선인출들 중 대부분은 재사용성이 낮아서 한두번만 사용되면 더 이상 사용되지 않는 경우가 많다. 따라서 선인출 데이터와 요구 데이터는

그 성질이 근본적으로 큰 차이가 있는 것이다. 즉, 요구 데이터는 한 번 요구가 되어 캐시로 들어온 다음에는 언제든지 다시 사용될 가능성이 있으므로 가능한 한 모든 데이터를 계속 저장하는 것이 캐시 미스율 측면에서 유리하나, 선인출 데이터는 한두 번 요구가 된 뒤에는 재사용될 가능성이 매우 낮은 반면 벤치마크 전체적으로 유용한 선인출 데이터의 전체 개수는 요구 데이터에 비해서 훨씬 많다. 물론 계속해서 재사용되는 선인출 데이터도 있으나 이러한 것들은 그 수가 많지 않으므로 선인출 전용 캐시는 그와 같이 반복되어 사용되는 선인출 데이터를 모두 포함할 수 있는 공간 및 일회성인 선인출 데이터의 임시 저장 공간 정도만을 갖추면 된다. 통상적인 경우와 같이 요구 데이터와 선인출 데이터를 모두 같은 캐시에 저장할 경우, 엄청난 양의 선인출 데이터에도 불구하고 요구 데이터가 축출되지 않고 유지될 수 있을 만큼의 매우 큰 공간이 필요하게 된다. 따라서, 두 캐시를 분리하는 경우 필요한 메인 캐시와 선인출 전용 캐시의 크기를 합한 것보다도 훨씬 더 큰 크기의 캐시가 필요하였던 것이다. 선인출 전용 캐시를 사용하여 캐시를 분리하는 경우, 메인 캐시의 크기도 줄어들게 되며, 만일 정확한 선인출 필터링이 수반된다면 선인출 전용 캐시는 더욱 더 감소될 수 있다. 예를 들어 다음 장에서 설명될 시뮬레이션 결과에 따르면, 축출 주소 참조 필터링 구조와 선인출 전용 캐시 구조를 결합하는 경우 기존 방법과 비교하여 1/4의 용량만을 가지고도 거의 동일한 캐시 미스율을 보인다.

4. 시뮬레이션 및 성능 분석

4.1 시뮬레이션 환경

필터링을 적용한 선인출 데이터의 효과를 분석하기 위하여 DEC 사의 ATOM 시뮬레이터[12]를 이용하여 트레이스 구동 시뮬레이션을 수행하였다. 트레이스는 메모리 참조 명령어에 대하여 적재(load) 혹은 저장(store)인지 여부, 필요한 피연산자의 유효주소, 그리고 PC로 구성되어 있다. 캐시 시뮬레이터는 이 트레이스를 입력으로 받아 캐시의 성능 및 선인출의 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 실험대상으로 하였고, 또한 제안된 방식의 경우 데이터 캐시도 메인 캐시와 선인출 전용 캐시를 분리하여 데이터 캐시의 성능을 측정하였다. 캐시 시뮬레이터는 캐시 크기, 블록크기, 사상함수 그리고 교체 알고리즘 등의 매개변수를 입력으로 캐시 분석 결과를 생성한다. <표 1>에 분석 모델을 위한 본 연구와 관련된 캐시 시뮬레이터의 매개변수들과 값을 제시하였다. 대표적인 멀티미디어 벤치마크 프로그램인 MPEG(mpeg2enc, mpeg2dec)과 SpecInt2000의 일반 벤치마크 프로그램인 parser, crafty, vortex, gap을 대상으로 1천만번의 메모리 참조 명령어에 대하여 실험하였으며, <표 2>는 벤치마크 프로그램들의 특징을 나타낸다.

〈표 1〉 분석 모델을 위한 매개 변수의 값

Parameter	Description
M-Dcache size	4K~1Mbyte
P-Cache size	2K~16Kbyte
Block size	32byte
Word size	4byte
M-Dcache Assoc.	Directmapped
P-Cache Assoc.	a1, a2, a4, a8, a16, fully assoc
Bus width	Block size
Replacement	LRU
Filtering table size	Cache set size

〈표 2〉 벤치마크 프로그램의 특징

Bench program	Input file	Description
vortex	bendian.raw	Object-oriented Database
gap	Algfld.g	interpreter
parser	2.1.dict	Word processing
crafty	reference input	Game Playing: Chess
mpeg2encoder	pirates.par	digital video와 audio 압축에 관한 표준 도구
mpeg2decoder	mei16v2.m2v	

(그림 6)은 요구인출과 선인출 명령을 발생하기 위해 필터 구조를 포함한 경우의 필터링과 갱신과정에 대한 흐름도이다. 각 번호에 대한 내용은 다음과 같다.

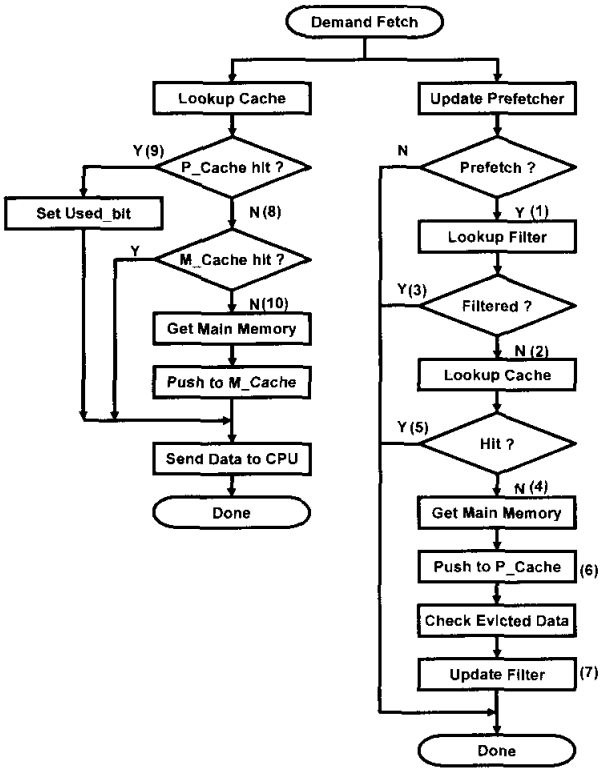
(1) Prefetcher : 각 선인출 알고리즘에 의해 계산된 선인출할 데이터의 주소
(2) Miss_filter : 필터 테이블을 참조하여 필터링 조건에 만족하지 않은 경우
(3) Hit_filter : 필터 테이블을 참조하여 필터링 조건에 만족하는 경우
(4) C_M : 두 캐시를 참조하여 미스가 발생한 수로 실제로 선인출 명령을 발생
(5) C_H : 두 캐시를 참조하여 히트가 난 경우로 실제로 선인출 명령을 발생시키지 않음
(6) Push-P_Cache : 두 캐시에서 미스가 발생한 경우 메모리를 접근한 후 해당 데이터를 선인출 전용 캐시에 저장

(7) Update-FT : 축출된 데이터의 사용여부로 필터 테이블을 갱신

(8) Miss-P_Cache : 요구 인출에 의해 선인출 전용 캐시에서 미스가 발생한 경우

(9) Hit P_Cache : 요구 인출에 의해 선인출 전용 캐시에서 히트가 발생한 경우, Used_bit를 1로 한다.

(10) D_M : 프로그램이 실행하면서 데이터 액세스를 필요로 하였으나, 두 캐시에서 미스가 발생한 경우로 메모리를 접근한 후 해당 데이터를 메인 캐시에 저장. D_M을 프로그램의 전체 메모리 액세스 수로 나누면 캐시 미스율이다.



(그림 6) 선인출 명령 발생을 위한 필터 구조를 포함한 흐름도

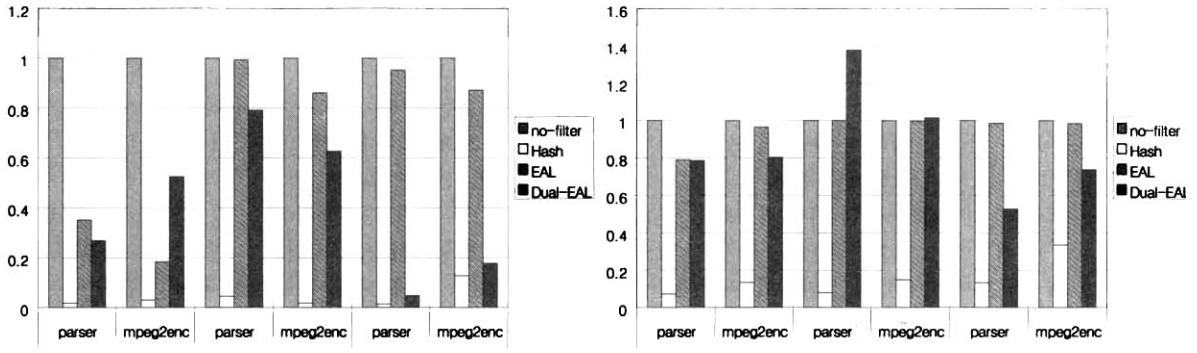
4.2 필터링 정확도 분석

필터링을 적용한 선인출 알고리즘의 정확도를 각 벤치마크에 대하여 비교하였다. 필터링 정확도라고 하는 것은 불필요한 선인출은 최대한 억제하는 대신 유용한 선인출에는 영향을 주지 않는 것이라고 할 수 있다. 따라서 각각의 필터링 방식을 적용한 결과를 필터링을 적용하지 않은 경우와 비교함으로써 필터링 정확도를 평가해 볼 수 있다. 즉, 필터링 하지 않았을 때의 유용한 선인출 및 불필요한 선인출과 필터링을 적용하였을 때의 유용한 선인출 및 불필요한 선인출의 수를 이용하여 3장의 식 (1)과 같이 계산할 수 있는데, 결과적으로 불필요한 선인출의 억제도와 유용한 선인출의 보존도의 기하평균이 되어 두 값이 모두 1에 가까울수록 정확도가 높아지게 된다. 여기에서 불필요한 선인출을 다음의 2가지로 서로 다르게 정의할 수 있다. 첫째, 선인출을 한 뒤 캐시에서 사용되지 않은 채 축출된 데이터이다. 이 경우, 유용한 선인출은 캐시에서 한 번 이상 사용된 뒤 축출된 데이터가 된다. 둘째로, 전체 트레이스를 검색하여 선인출을 발생시킨 시점 이후에 해당 주소에 대한 요구 인출이 트레이스 내에서 발견되지 않은 경우 그 선인출을 불필요한 선인출이라고 할 수 있다. 두 정의간에 차이점이 있는데, 첫 번째 정의에서는, 만일 계속 캐시에 남아 있었더라면 사용되었을 데이터를 너무 일찍 선인출 함으로써 사용되기도 전에 다른 데이터에 의해 축출되어 버리는 경우도 불필요한 선인출에 포함되는데, 두 번째 정의에서는 이 경우는 유용한 선인출에 해당된다. 이 경우 필터링 정확도가 캐시의 크기 등 여러 요인에 의해 영향을 받을 수 있으므로 본 논문에서는 첫 번째 정의를 사용하였다. 그러나 두 가지 정의에 의한 상대적인 결과는 큰 차이가 없음이 실험을 통해 알 수 있었다.

(그림 7)은 대표적인 두 가지 벤치마크(일반 벤치마크인 parser와 멀티미디어 벤치마크인 mpeg2enc)에 대해서 앞에서 설명한 각각의 선인출 방식에서의 필터링 결과를 보여준다. 각각의 그래프는 불필요한 선인출(bad prefetch)과 유용한 선인출(good prefetch)의 개수를 필터링 하지 않은 결과에 대해서 정규화한 값을 보여준다. (그림 7)에서 캐시 크기는 64Kbyte이고 라인크기는 32byte, 필터링 테이블의 엔트리 수는 2048개로 하였다. 4개의 막대그래프로 이루어진 각각의 그룹은 위에서 언급한 두 가지 벤치마크에 대해서 세 가지의 하드웨어 선인출 방식에 대한 필터링 결과에 해당한다. 4개의 막대그래프는 차례대로 (1) 필터링 하지 않은 경우, (2) 기존의 해시 필터 구조, (3) 제안된 축출 주소 참조 방식을 기존 캐시 구조에 적용한 경우, 그리고 (4) 제안된 축출 주소 참조 방식을 제안된 선인출 전용 캐시 구조와 결합한 경우의 결과에 각각 해당한다.

전반적으로 기존 해시 필터 구조(Hash)가 good 및 bad 선인출 모두 가장 낮은 값을 가지므로, 하드웨어 선인출기에서 개시된 선인출들이 가장 많이 필터링 되었음을 알 수 있다. 그러나, 유용한 선인출(good prefetch)의 수가 줄어들었다는 것은 필터링 정확도, 특히 보존도가 크게 떨어진다는 것을 의미하며 선인출에 의해 감소된 캐시 미스율이 상당 부분 상쇄되었을 것이라는 사실을 예측할 수 있다. 각각의 필터링 방식에 대한 캐시 미스율 성능은 다음절에서 상세히 논의될 것이다.

축출 주소 참조 방식을 통합 캐시 구조에 적용한 경우(EAL)에는, 유용한 선인출은 거의 그대로 보존되었지만, 불필요한 선인출도 OBL 방식을 제외하면 크게 줄어들지 않았다. 즉, 보존도는 매우 우수하나, 억제도는 상대적으로 낮다. OBL 선인출 방식에서는 불필요한 선인출이 대폭 줄어든 것을 볼 수 있다. 이것은 OBL 방식이 다른 두 선인출 방식(RPT 및 correlation)에 비하여 선인출기로부터 최초로 개시되는 선인출 명령의 수가 상대적으로 훨씬 많고, 또한 이로 인하여 축출되는 때 사용된 선인출의 수가 많아져서 축출 주소 리스트의 엔트리들이 많이 채워지게 되어 필터링 억제도가 높아지기 때문이다.



(그림 7) good과 bad 선인출 수 비교(좌측 : bad prefetch, 우측 : good prefetch)

축출 주소 참조 방식과 선인출 전용 캐시 구조를 모두 사용한 경우(Dual EAL)에는 불필요한 선인출 수가 correlation에 대해서는 대폭 줄어들었고 RPT에 대해서도 많이 줄어들었다. 주목할 만한 것은 유용한 선인출 수가 RPT의 경우에 필터링 하지 않은 경우에 비해서조차 더 증가한 것을 볼 수 있는데, 이것은 선인출 전용 캐시의 크기가 작아져서 상대적으로 축출되는 데이터의 수가 증가하였기 때문이다. 그럼에도 불구하고 bad 선인출 수, 즉 미 사용되고 축출된 데이터의 수가 줄어들었다는 것은 애초에 선인출 전용 캐시로 유입되는 불필요한 선인출의 수는 그림에 나타난 것보다 도 훨씬 더 많이 줄어들었다는 것을 의미한다. 또한, 유용한 선인출 수, 즉 사용되고 축출된 데이터의 수가 증가하였다는 것은 만일 이러한 데이터가 다음번에 또다시 요구되는 일이 많다고 가정하면 캐시 미스 수가 증가할 수도 있을 것이다. 그러나 다음 장에서도 언급될 것이지만 제안된 방식에서 캐시 미스 수가 오히려 대폭 줄어든 것을 보면, 선인출 데이터의 대부분은 앞서 말한 바와 같이 재사용성이 매우 낮거나, 또는 다음번에 다시 사용되기 전에 또 다시 선인출 명령이 발생되기 때문에 문제가 되지 않는 것임을 알 수 있다. 또한, 몇 번이고 재사용되는 일부 선인출 데이터는 LRU 정책에 의하여 거의 축출되지 않을 것이다. 한편, correlation에서의 bad 선인출 수가 OBL이나 RPT에 비해 대폭 줄어든 것으로 나타난 것은 correlation 방식의 특성 상 최초로 선인출을 개시하는 수가 다른 방식에 비해 매우 낮아서, 미 사용되고 축출되는 데이터의 수 (bad 선인출 수)가 실제 불필요한 선인출의 수를 잘 대변해 주고 있기 때문이다.

〈표 3〉 선인출 필터링에 대한 정확도 분석

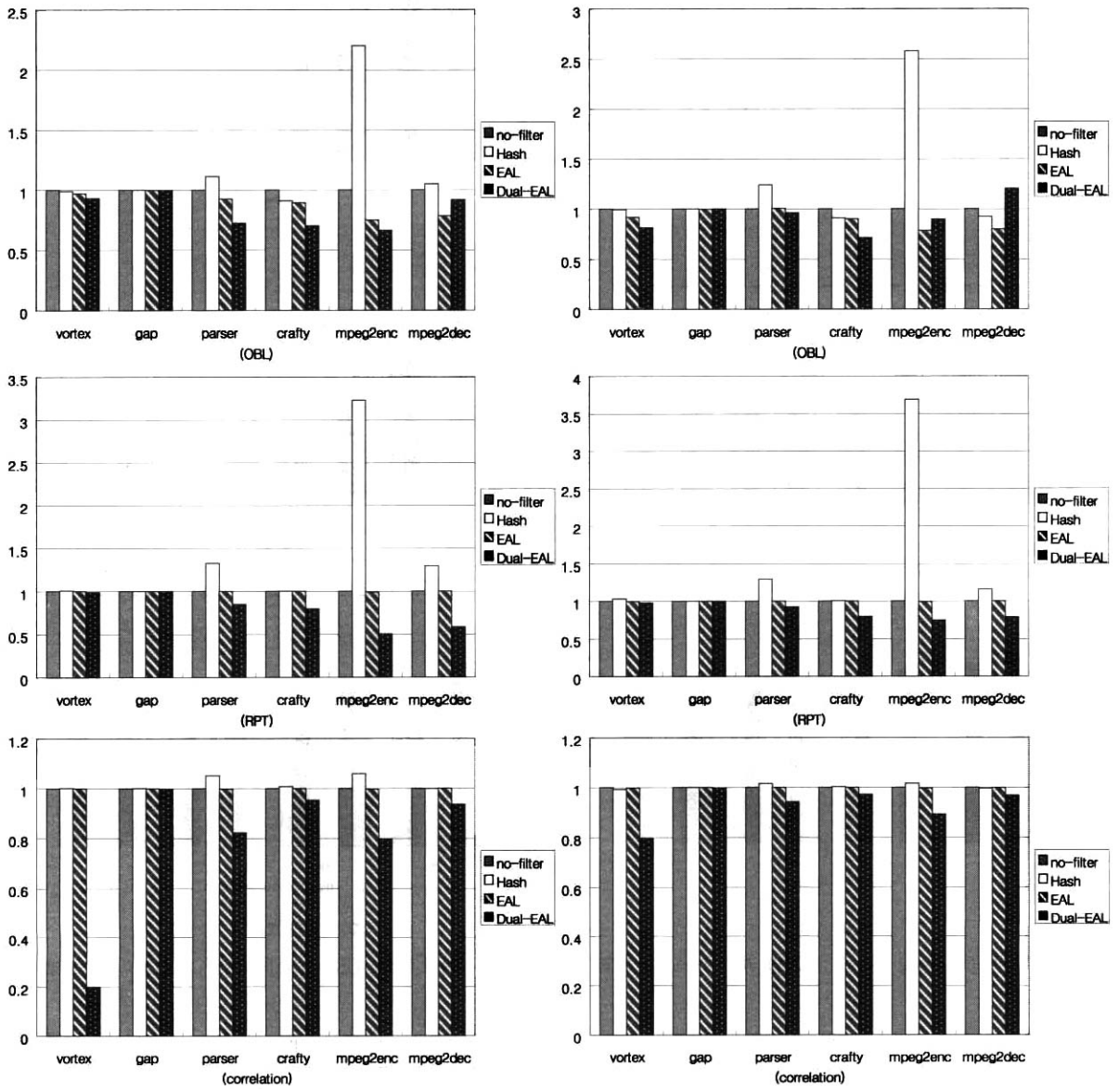
		OBL		RPT		CORR		Average
		parser	mpeg2enc	parser	mpeg2enc	parser	mpeg2enc	
No-filter	good	33,701	25,204	31,947	25,604	19,493	6,771	
	bad	42,125	12,743	950	1,537	15,964	5,487	
	accuracy	0%	0%	0%	0%	0%	0%	0%
Hash	good	2,438	3,368	2,513	3,772	2,578	2,271	
	bad	736	379	43	26	214	701	
	accuracy	26.7%	36.0%	27.4%	38.1%	36.1%	54.1%	36.4%
EAL	good	26,611	24,324	31,945	25,509	19,208	6,661	
	bad	14,803	2,343	943	1,324	15,207	4,784	
	accuracy	71.6%	88.7%	8.6%	37.2%	21.6%	35.5%	43.8%
Dual-EAL	good	26,481	26,481	43,982	25,961	10,250	4,997	
	bad	11,345	6,690	751	963	773	979	
	accuracy	75.8%	61.7%	53.7%	61.5%	70.7%	70.7%	66.9%

<표 3>은 선인출 필터링 방식들에 대해 선인출 알고리즘별로 정확도를 나타낸 것이다. 정확도(accuracy)는 3장의 식 (1)을 사용하여 계산한 결과를 백분율로 환산한 것이다. 평균적으로 분리된 캐시를 사용한 제안된 Dual EAL방식이 가장 우수한 필터링 정확도를 가짐을 알 수 있다.

4.3 전체 성능 분석

위에서 분석한 필터링 정확도 특성에 의하여 캐시의 전

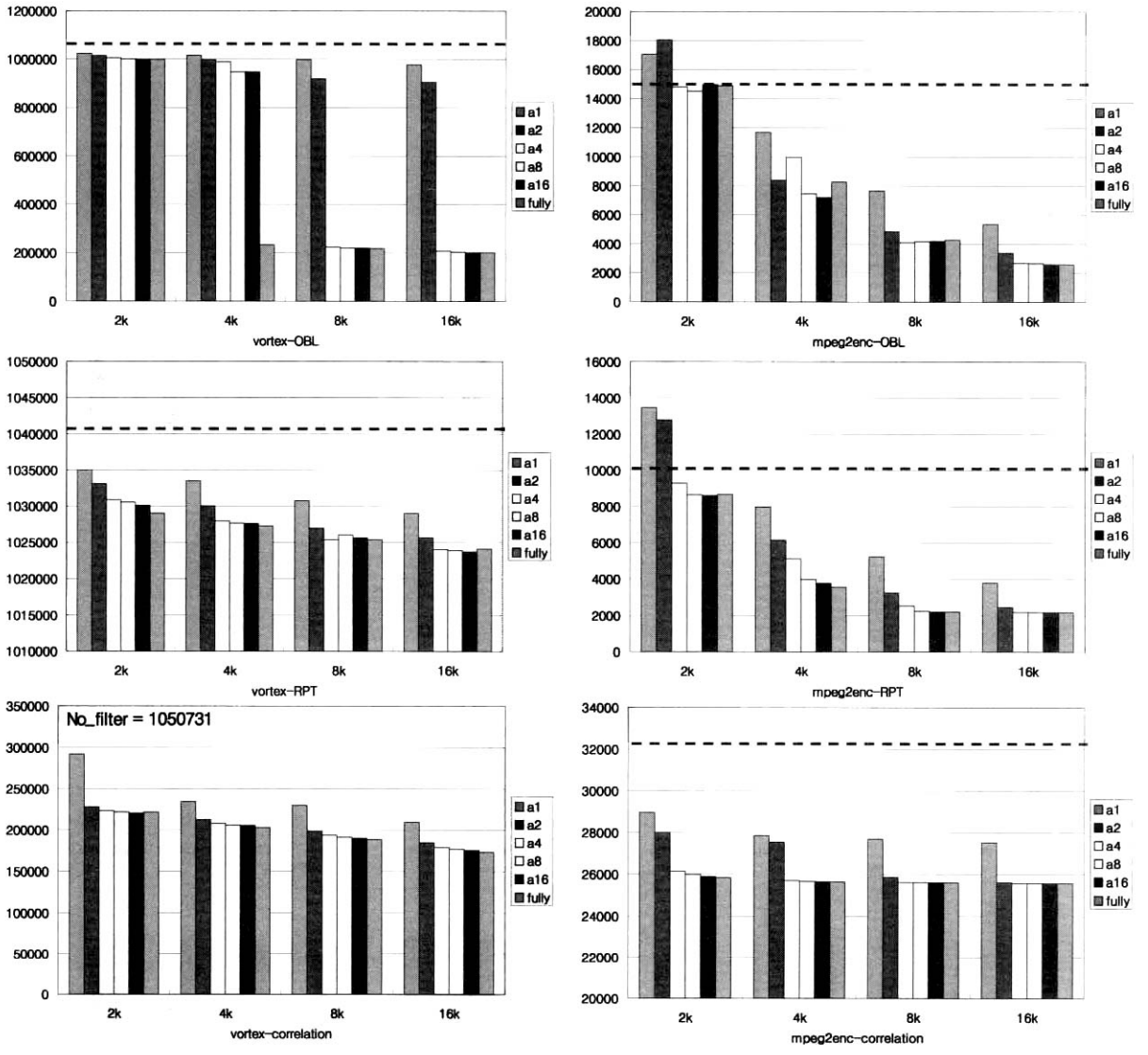
체적인 성능이 어떻게 영향을 받는지를 분석해 보았다. (그림 8)은 세가지 선인출 알고리즘과 6가지 벤치마크(vortex, gap, parser, crafty, mpeg2enc, mpeg2dec)에 대하여 메인 캐시의 크기를 64Kbyte와 128Kbyte로 하였을 때의 각 필터링 방식에서의 캐시 미스율을 필터링 하지 않은 경우에 대하여 정규화한 결과이다. 블록크기는 앞에서와 같이 32byte이고, 캐시(또는 메인 캐시)는 direct mapped 방식(1-way)을 사용하였고, 선인출 전용 캐시의 크기는 두 가지 캐시 크기에 대해 모두 4Kbyte, 4way방식을 사용하였다. 평균적으로 필터링을 하지 않는 방식을 기준으로 해시 필터 방식(Hash)은 미스율이 오히려 1% 증가하였고, 하나의 캐시를 사용하는 축출 주소 참조 방식(EAL)은 0.8% 감소하였으며, 분리된 캐시를 사용한 축출 주소 방식(Dual-EAL)은 미스율이 13% 감소하였다.



(그림 8) 벤치마크별 캐시 미스 수 (top : OBL, middle : RPT, bottom : correlation)

특히 선인출로 인한 효과가 큰 멀티미디어 벤치마크의 경우에 기존의 해시 필터 방식은 캐시 미스율이 최대 3.5배 이상 증가하기도 하는데, 이것은 필터링 정확도가 낮아서 유용한 선인출이 대폭 감소되었기 때문이다. 제안된 방식(Dual EAL)에서는 멀티미디어 벤치마크의 경우 캐시 미스율이 최대 절반가량 줄어들며 vortex에 대해서는 correlation방식에서 최대 80% 이상 캐시 미스율이 줄어들었다. 이것은 앞서 설명한 바와 같이, 선인출 전용 캐시의

사용으로 선인출 데이터와 요구 데이터가 각각의 분리된 캐시에 저장되기 때문에 상호간의 충돌/경쟁이 없어졌기 때문이며, 정확한 필터링 방식으로 인하여 전체 캐시 크기의 6%(128K의 경우는 3%) 정도의 작은 선인출 전용 캐시만을 가지고도 필요한 선인출 데이터를 유지할 수 있기 때문이다.



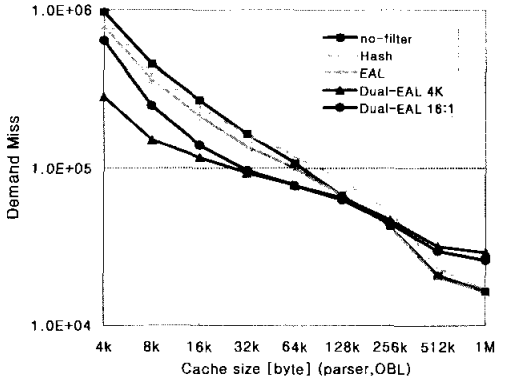
(a) vortex (b) mpeg2enc

(그림 9) 선인출 전용 캐시의 크기별 캐시 미스 수 비교(메인 캐시 = 64K) (top : OBL, middle : RPT, bottom : correlation)

(그림 9)는 선인출 전용 캐시의 크기를 2Kbyte에서 16Kbyte까지 변화 시키고, associativity를 1-way, 2-way, 4-way, 8-way, 16-way 및 fully-associative로 구분하여 캐시 미스 수를 비교한 것이다. 또한 비교를 위하여 필터링 하지 않았을 때의 통합 캐시 구조에 대한 캐시 미스 수를 점선으로 표시하였다. 전반적으로 선인출 캐시의 크기가 클수록 또한 associativity를 크게 할수록 미스수가 감소한다. 선인출 캐시의 크기가 2Kbyte인 경우에는 4-way 이상인 때에, 4Kbyte 이상인 경우에는 모든 associativity의 경우에 대해서 통합 캐시에 비해서 캐시 미스 수가 더 작아진다. 4Kbyte fully associative 또는 8Kbyte, 4-way 이상이 되면 캐시 미스 수는 통합 캐시의 절반 이하로 감소된다. 통합 캐시의 경우 캐시 크기가 2배가 될 때에 캐시 미스 수는 60~70% 정도가 되고, 캐시 크기가 4배가 되어야 캐시 미스 수가 비로소 절반 이하로 내려가게 되는 것을 고려할 때, 제안된 방식의 선인출 전용 캐시를 위하여 증가되는 수 퍼센트 정도의 오버헤드는 그것을 지불하는 대가로 얻게 되는 캐시 크기의 이득에 비하여 무시할 만 한 것이라 할 수 있다.

(그림 10)은 캐시크기를 4Kbyte부터 1Mbyte까지 증가 시키면서 캐시 미스 수를 측정한 결과이다. 제안된 방식 중

"Dual EAL 4K"는 메인 캐시의 크기와 무관하게 선인출 전용 캐시의 크기를 4Kbyte로 고정시킨 경우이고, "Dual-EAL 16:1"은 선인출 전용 캐시의 크기를 메인 캐시의 1/16이 되도록 가변 시킨 것이다. 그림에서 16Kbyte의 메인 캐시와 4Kbyte의 선인출 캐시를 갖는 제안된 방식(Dual EAL 4k)의 경우 64Kbyte의 크기를 가진 기존 방식의 경우와 캐시 미스 수가 거의 같음을 알 수 있다. 또한, 32Kbyte의 메인 캐시와 2Kbyte의 선인출 캐시를 갖는 제안된 방식(Dual EAL 16:1)의 경우에는 64Kbyte의 기존 방식보다는 미스 수가 작고 128Kbyte의 기존 방식보다는 미스 수가 크다. 따라서 본 논문에서 제안된 선인출 전용 캐시 및 축출 주소 참조 필터링을 이용하면 약2배에서 4배 정도의 캐시 크기의 이득을 얻을 수 있음을 알 수 있다.



(그림 10) 캐시 크기에 따른 캐시 미스 수

캐시 크기가 256Kbyte 이상이 되면 분리된 캐시를 사용하는 경우가 통합 캐시를 사용하는 방식들에 비해 미스 수가 더 커지게 된다. 이것은 통합 캐시의 경우 캐시 크기가 매우 커서 잘못된 선인출이 있더라도 이것들로 인하여 요구 데이터 또는 유용한 선인출 데이터가 축출되는 일이 거의 없기 때문이다. 반면, 분리된 캐시 구조의 선인출 캐시의 경우 크기가 작기 때문에 일부 유용한 선인출 데이터가 축출되는 경우가 원래 존재하였는데, 전체(또는 메인) 캐시 크기가 128Kbyte 이하인 경우에는 이로 인한 미스 수 증가가 기존 방식의 과도한 선인출 또는 요구 데이터와 선인출 데이터 간의 경쟁으로 인한 미스 수 증가에 비해 미미하였었으나, 캐시 크기가 256Kbyte 보다 더 커지게 되면 기존 방식의 문제점은 과도한 캐시 크기로 인하여 저절로 없어지는 반면 분리된 선인출 전용 캐시는 그러한 과도한 캐시 크기의 혜택을 받지 못하게 되어 위에서 설명한 것과 같은 capacity 미스가 계속 남아있기 때문이다. 이와 같이 캐시 크기가 매우 큰 경우에는 선인출 전용 캐시의 크기가 메인 캐시 크기의 1/4 정도가 되어 주어야 통합 캐시 방식과 거의 같은 미스 수를 낼 수 있었다.

한편 해시 필터 구조에서는 캐시 크기가 커질수록 필터링하지 않은 경우에 비해 미스 수가 점점 증가하다가 캐시 크기가 512Kbyte 이상이 되면 다시 필터링 하지 않은 경우와 비슷하게 감소하는 경향을 보인다. 이것은 캐시 크기가 과도하게 큰 경우에는 미 사용된 선인출 데이터들이 거의 축출되지 않기 때문에 앞 장에서 설명한 것과 같은 해시 구조 특유의 잠김 현상이 일어나지 않게 되고, 필터링 자체가 거의 일어나지 않기 때문에 aliasing에 의한 문제도 없기 때문이다. 반면에, 제안된 구조에서는 필터링이 계속 일어나서 버스 사용량을 줄이게 되며 이 때에 유용한 선인출에 대한 보존도가 100% 보다는 낮기 때문에 위에서 언급된 바와 같이 과도하게 큰 캐시에서의 미스 수 증가의 한가지 요인으로 해석될 수도 있다.

5. 결 론

과도한 선인출로 인한 캐시의 오염을 막기 위해 기존의 해시 필터 구조가 갖는 문제점을 보완한 축출 주소 참조 필터링 구조와 선인출 전용 캐시 구조를 도입하였다. 선인출 전용 캐시는 선인출 데이터와 요구 데이터를 완벽하게 분리시켜서 상호간의 충돌 또는 경쟁을 제거하여 캐시 미스 수를 감소시키는 역할을 수행함과 동시에 그것의 작은 크기로 인하여 축출되는 불필요한 선인출 정보를 풍부하게 하여 제안된 필터링 구조의 억제성을 향상시키는 역할을 한다. 축출 주소 참조 필터링 구조는 해시 함수를 사용하지 않기 때문에 근본적으로 유용한 선인출에 대한 보존성이 매우 우수하며, 반면 상대적으로 취약한 억제성은 상기와 같이 선인출 전용 캐시와의 결합으로 향상될 수 있었다. 기존의 해시 필터 구조가 aliasing으로 인하여 필터로서의 기본적인 기능인 선택성이 거의 없어 불필요한 데이터 뿐 아니라 유용한 데이터들 마저 필터링 한다는 것과 이로 인한 잠김 현상으로 일정 시간이 지난 뒤에는 거의 모든 선인출을 차단시키게 되어 선인출로 인한 미스 수 감소 효과를 무효화 시킨다는 것을 보였다. 이러한 필터링 방식에서의 정확도를 수치화하기 위하여 필터링 정확도라는 척도를 제안하였으며, 이 척도를 이용하여 제안된 방식이 평균적으로 67%정도의 우수한 필터링 정확도를 가짐을 실험 결과를 통하여 보였다.

일반 벤치마크와 멀티미디어 벤치마크 프로그램을 가지고 실험을 한 결과, 캐시 미스율은 모든 선인출 알고리즘과 벤치마크들에 대하여 필터링을 하지 않은 방식과 비교하여 평균적으로 미스율이 EAL 방식은 0.8%, 선인출 전용 캐시를 갖는 Dual-EAL 방식은 13% 감소함을 알 수 있었으며, 벤치마크 및 선인출 방식에 따라 최대 80%이상의 캐시 미스 수를 줄일 수 있음을 보였다. 제안된 구조는 재사용성이 높은 요구 데이터와 재사용성이 낮고 유입이 잦은 선인출 데이터라는 서로 다른 성질을 갖는 두 데이터를 완전하게 분리하므로, 각각이 저장되는 분리된 두 캐시 모두의 크기를 줄일 수 있게 하여 준다. 전반적으로 동일한 캐시 미스율을 위해서 제안된 방식은 기존 방식에 비해 전체 캐시 크기를 2배에서 4배까지 줄일 수 있도록 함을 실험을 통하여 알 수 있었다. 따라서 제안된 방식을 사용하여 L1 또는 L2 데이터 캐시의 크기를 줄이거나 성능을 향상시킬 수 있을 것으로 보인다.

동일한 캐시 미스율에 대하여 더 작은 크기의 캐시를 사용할 수 있다는 것은 캐시로의 접근 지연 시간을 빠르게 할 수 있다는 것을 내포한다는 것을 또한 주목할 필요가 있다.

또한, 본 논문에서 제안된 방식들을 응용하여 웹 또는 데이터베이스 선인출 등 여러 다양한 분야에 확장하여 적용하는 연구가 추가적으로 진행되어야 할 것으로 생각된다.

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 per formance 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] J. H. Lee, S. W. Jeong, S. D. Kim and C. C. Weems, "An Intelligent 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 박사수료
관심분야 : 고성능 컴퓨터 구조, 병렬 처리, 메모리 시스템, 멀티미디어 시스템

김 석 일

e-mail : ksi@cbucc.chungbuk.ac.kr
1975년 서울대학교(학사)
1985년~1989년 미국 North Carolina State University(공학박사)
1975년~1995년 국방과학 연구소 선임 연구원
1990년~현재 충북대학교 전기전자컴퓨터 공학부 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 전기전자컴퓨터 공학부 교수
관심분야 : 컴퓨터 구조, 임베디드 시스템