단속적(斷續的) 불규칙 주소간격을 갖는 멀티미디어 데이타를 위한 하드웨어 캐시 선인출 방법

(A Hardware Cache Prefetching Scheme for Multimedia Data with Intermittently Irregular Strides)

전 영 숙 † 문 현 주 †† 전 중 남 ††† 김 석 일 ††††
(Young-Suk Chon) (Hyun-Ju Moon) (Joongnam Jeon) (Sukil Kim)

요 약 멀티미디어 응용 프로그램은 방대한 양의 데이타를 실시간으로 고속 처리해야 한다. 적재/저장과 같은 메모리 참조 명령어는 프로세서의 고속 수행을 방해하는 주요인이다. 메모리 참조 속도를 향상시키기 위하여, 다음에 참조될 것으로 예상되는 데이타를 미리 캐시로 인출함으로써, 캐시 미스율과 전체 수행시간을 감소시키는 캐시 선인출 방법이 활용되고 있다. 본 연구에서는 기존의 참조예측표(RPT: Reference Prediction Table)를 사용하는 방법을 개선한 데이타 캐시 선인출 방법을 제시한다. 동일한 명령어가 참조하는 데이타의 주소간격을 계산할 때 캐시의 라인크기 단위의 주소간격을 사용하고, 규칙적인 주소간격에 불규칙한 간격이 하나 포함하더라도 선인출 효과를 유지할 수 있도록 선인출 알고리즘을 개선하였다.

일반적으로 많이 사용되는 멀티미디어 프로그램에 대하여 실험한 결과, 기존의 RPT 방식에 비하여 버스 사용량은 약 0.03% 증가한 반면에 캐시 미스율은 평균적으로 29% 정도 향상되었다.

키워드 : 멀티미디어 응용프로그램, 데이타 캐시 선인출, 참조예측표, 스트리밍 액세스 패턴

Abstract Multimedia applications are required to process the huge amount of data at high speed in real time. The memory reference instructions such as loads and stores are the main factor which limits the high speed execution of processor. To enhance the memory reference speed, cache prefetch schemes are used so as to reduce the cache miss ratio and the total execution time by previously fetching data into cache that is expected to be referenced in the future.

In this study, we present an advanced data cache prefetching scheme that improves the conventional RPT (reference prediction table) based scheme. We considers the cache line size in calculation of the address stride referenced by the same instruction, and enhances the prefetching algorithm so that the effect of prefetching could be maintained even if an irregular address stride is inserted into the series of uniform strides.

According to experiment results on multimedia benchmark programs, the cache miss ratio has been improved 29% in average compared to the conventional RPT scheme while the bus usage has increased relatively small amount (0.03%).

Key words : multimedia application, data cache prefetch, reference prediction table, streaming access pattern

· 본 연구는 한국과학재단 목적기초연구(R05-2002-000-01470-0)지원으로 수행되었음
† 학생회원 : 충북대학교 컴퓨터과학과
yschon@hanmail.net
†† 비 회 원 : 나사렛대학교 정보과학부 교수
hjmoon@kornu.ac.kr
††† 종신회원 : 충북대학교 전기전자컴퓨터공학부 교수
joongnam@cbu.ac.kr
†††† 종신회원 : 충북대학교 컴퓨터과학과 교수
ksi@cbucc.chungbuk.ac.kr
논문접수 : 2003년 10월 21일
심사완료 : 2004년 7월 29일

1. 서 론

멀티미디어 데이타를 처리하는 응용프로그램은 다량의 재사용성이 적은 메모리를 참조하는 특성이 있다. 메모리 참조 지연시간을 감소시킴으로써 전체 수행시간을 감소시킬 수 있으며, 가장 많이 사용되는 방법이 캐시 기억장치이다. 캐시 기억장치는 주기억장치보다 처리속도가 빠르고, 주기억장치 내용의 일부분을 복사하여 저장하고 있다가 프로세서가 적재/저장 등의 메모리 참조

명령어를 수행할 때에 먼저 캐시에 있는지를 검사하여 있으면(히트) 해당 단어(word)가 프로세서로 전달된다. 프로그램에 존재하는 참조의 지역성(locality of reference)과 재사용성(reusability)으로 인하여, 캐시 기억장치는 전반적인 프로그램 수행 속도를 단축시킨다.

캐시 구조의 선인출 기법은 프로세서가 사용할 것으로 예측되는 데이타를 실제로 필요하기 전에 주기억장치에서 캐시로 인출하여 캐시 미스율을 감소시키는 방법이다. 이 때에, 프로세서의 명령어 처리와 주기억장치의 데이타 액세스는 중첩 실행된다. 캐시 선인출 방법들 중에서 하드웨어에 의한 선인출 기법은 데이타 주소 기반 선인출 방법과 PC 기반 선인출 방법으로 구분되는데, 데이타 주소 기반 선인출 방법으로는 Smith에 의해 OBL(one block-look-ahead) 방식[1]이 제안되었고, Jouppi는 OBL 방식을 확장하여 스트림 버퍼 선인출 방식[2]을 제안하였다. 또한 Joshep과 Grunwald에 의해 Markov predictor 방식[3-6]이, Cucchiara등에 의해 neighbor 방식이 제안되었다[7-9]. PC(Program Counter) 기반 선인출 방법으로는 Chen과 Baer 등이 참조예측표(RPT: Reference Prediction Table) 방식[10]을 제안하였다.

RPT 방식은 멀티미디어 응용 프로그램에서 데이타의 주소간격이 일정하게 참조된다는 것을 이용하여 참조 패턴이 안정 상태일 때 연속한 다음 위치의 데이타를 선인출을 함으로써 캐시 미스율을 감소시킨다. 이러한 방법을 규칙 선인출 기법이라고 한다. 그러나 RPT 방식은 선인출 대상인 데이타의 주소간격(stride)을 계산할 때 주소에 할당되는 비트들을 모두 사용하기 때문에, 캐시 라인크기보다 적은 주소간격에 해당하는 데이타도 선인출 명령을 발생시키는 문제점이 있다. 또한 주소간격이 두 번 연속 일치하는 경우에 안정상태로 되어 선인출 명령을 발생시키지만, 불규칙한 주소간격이 하나라도 포함되는 경우에는 예측의 효과가 감소하는 단점이 있다.

본 논문에서는 RPT 방식을 두 가지 측면에서 개선한 M-RPT(Modified-RPT) 선인출 방법을 제안한다. 첫째, 선인출 하려는 데이타의 주소간격을 계산할 때 캐시 라인크기를 고려하여 블록주소간격을 사용한다. 선인출 주소간격이 라인크기보다 적은 경우는 이미 캐시 안에 존재하는 것이므로 캐시 선인출 명령을 발생하지 않음으로써, 불필요한 선인출 명령의 발생을 제거하는 효과가 있다. 둘째, 주소간격이 0이 아니면 선인출 명령을 발생시키고, 이전의 주소간격과 비교하여 주소간격이 두 번 연속 다른 경우에 선인출 명령을 발생시키는 주소를 변경하도록 RPT 구조 및 선인출 알고리즘을 개선한다. 그 결과 메모리 참조 주소 시퀀스에 불규칙한 주소간격이 한 개 포함되는 경우에도 올바른 데이타를 선인출할 수 있는 능력을 가질 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구로써, 기존 선인출 기법에 관한 연구를 요약하고 본 논문의 대상인 RPT 방식에 대하여 설명한다. 3장에서는 기존 RPT 방식의 문제점을 살펴보고, 4장에서는 제안하는 M-RPT 선인출 방법을 위한 하드웨어 구조와 선인출 알고리즘을 자세히 설명한다. 5장에서는 멀티미디어 데이타의 참조 패턴에 대한 통계적 분석을 수행하였다. 6장에서 실험을 통한 성능 평가 결과를 제시하고, 마지막으로 7장에서는 본 논문에서 얻은 결과를 정리한다.

2. 관련 연구

캐시 선인출은 프로세서가 명령어들을 수행하는 동안 미래에 사용될 것으로 예상되는 데이타를 미리 캐시에 인출함으로써 캐시 미스로 인한 지연시간을 줄일 수 있는 효과적인 방법이다. 캐시 선인출 방법은 소프트웨어 기반 기법과 하드웨어 기반 기법으로 구분된다. 소프트웨어 기반 기법[11]은 컴파일러가 실행시간 이전에 선인출 명령어를 프로그램에 삽입하는 기법으로 최근에는 명령어 세트에 메모리 선인출 명령어를 포함시켜 선인출 명령어를 발생시킨다. 이와 비교하여 하드웨어 기반 기법은 실행시간에 메모리 참조 명령어를 관찰하고, 실행 처리기와 독립적으로 선인출할 메모리 주소를 계산한 뒤에 선인출 명령어를 발생시킨다. 앞에서 기술한 대로 하드웨어 기반 기법은 데이타 주소 기반 구조와 PC (Program Counter) 기반 구조로 구분되며, 데이타 주소 기반 구조에는 neighbor 방식[7-9]이 있고, PC 기반 구조에는 RPT 방식[10]이 있다.

Cucchiara[7]등은 멀티미디어 데이타, 특히 이미지 데이타에 적합한 neighbor 선인출 방식을 제안하였다. 이 방식은 이미지 압축, 이미지 분석 등 대부분의 알고리즘이 8×8 또는 16×16 픽셀 블록에 대하여 작용하므로 2차원적인 공간적 국소성을 가진다는 사실에 착안한다. 즉, 캐시에서 현재 참조 중인 주소의 상하좌우 및 사선 방향의 이웃한 8연결 블록(8-connected blocks)의 데이타를 선인출 함으로써 멀티미디어 응용 프로그램, 특히 MPEG-2 및 MPEG-4 디코딩에서 캐시 미스율을 현저히 줄일 수 있다고 한다. 현재 참조된 주소가 A0이고 하나의 이미지 행이 NBrow개의 바이트로 구성된다고 할 때, 먼저 8개의 인접 데이타의 주소들이 캐시 내에 있는지를 검사하고, 없는 경우에는 선인출 명령을 발생시킨다. 이 방식은 두 가지의 단점이 있는데, 첫째, 8개의 주소들이 캐시 내에 있는지 여부를 알아내기 위한 룩업 횟수가 증가한다는 점과, 둘째, 선인출 횟수도 증가하여 메모리 버스의 bottleneck으로 작용할 수 있다는

점이다. 이러한 점을 극복하기 위하여 Cucchiara등은 캐시에서 8연결 블록에 해당되는 데이타를 검사하여 캐시 미스인 첫번째 데이타만 선인출하는 방식을 제안하였다[8,9]. 만일 8연결 블록의 데이타가 모두 캐시에 들어있지 않다면, 8번에 걸쳐 선인출 명령이 발생되며, 선인출에 따른 긴 지연시간을 줄이는 효과가 있다. 전자의 방식의 이름을 opt. neighbor라 하고, 후자는 8-step neighbor라 한다.

한편, Chen과 Baer[10]에 의해 제안된 RPT 방식은 PC 기반 선인출 방법이다. 이 방법은 적재/저장과 같은 메모리 참조 명령어의 동작 정보를 RPT에 저장하여 이전 정보를 근거로 선인출에 대한 주소를 계산하고, 조건이 만족되면 선인출 명령을 발생시킨다. RPT 구조는 그림 1과 같이 태그 필드, 이전주소 필드, 주소간격 필드, 상태 필드로 구성되어 있으며, 각 필드의 용도는 다음과 같다.

- 태그(tag) 필드: 적재/저장 명령어의 주소, 즉 명령어에 대한 PC의 값
- 이전주소(prev_addr) 필드: 이전의 반복문 안에서 동일한 명령어가 참조한 피연산자(operand)의 주소
- 주소간격(stride) 필드: 이전 피연산자의 주소와 현재 참조한 피연산자의 주소간격
- 상태 필드(state): 네 개의 상태를 가지며, 선인출 명령을 발생할지 여부를 결정

그림 1에서 선인출할 메모리 주소(prefetching address)는 이전주소와 주소간격을 더하여 계산된다. 적재/저장 명령어의 상태가 안정 상태이고 주소간격이 0이 아니면 선인출 명령이 발생한다. 만약 이전 반복문에서 계산된 선인출 주소와 현재 참조한 메모리 주소가 같다면, 예측이 성공한 것이고, 그렇지 않은 경우는 예측이 실패한 것이다. 그림 2는 상태 필드와 주소간격 필드의 갱신에 관한 상태도이다. 각 상태별 동작은 다음과 같다.

- 초기(init) 상태: 적재/저장 명령어가 처음 실행될 때의 초기 상태
- 전이(transient) 상태: 주소간격이 이전주소와 다른 상태
- 안정(steady) 상태: 두 번 연속적으로 주소간격이 일치하여 선인출 명령을 발생하는 상태
- 예측 불가(no-pred) 상태: 주소간격이 일정하지 않은 상태

RPT 방식은 주소간격이 일정하게 연속하여 데이타를 참조할 경우에, 선인출 효과가 크다. 그러나, 동일한 명령어가 참조하는 데이타 시퀀스에 불규칙한 주소간격이 포함되는 경우에는 예측의 효과가 감소하는 단점이 있다. 예를 들어 영상처리 프로그램의 경우, 크기가 작은

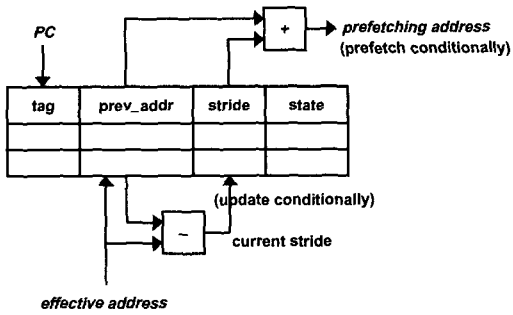

그림 1 RPT 구조

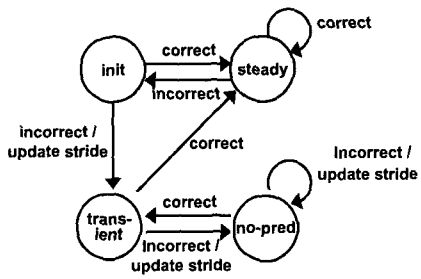

그림 2 RPT 방식의 상태도

서브블록 단위로 데이타를 처리하는 경우가 많이 있다. 이와 같이 데이타를 2차원적으로 참조하는 경우에는 RPT 방식의 선인출 효과가 감소한다. 이러한 문제를 해결하기 위해 멀티미디어 데이타가 2차원 이상의 서브블록 형태로 액세스 되는 경우에도 예측할 수 있도록 확장한 구조[12]가 제안되었다.

3. 기존 RPT 방식의 문제점

위에서 설명한 기존 RPT 방식의 문제점은 아래의 두 가지로 요약될 수 있다.

- 불필요한 선인출로 인한 룩업 오버헤드 증가
- 단속적 불규칙 데이타 참조 패턴인 경우 캐시 미스율 증가

이 장에서는 각각에 대한 자세한 분석을 하고자 한다.

3.1 불필요한 선인출로 인한 룩업 오버헤드 증가

RPT 방식은 주소간격을 계산할 때 식 (1)과 같이 주소에 할당되는 비트 전체를 사용하여 주소간격이 두 번 연속하여 일치하는 경우에 선인출 명령을 발생한다.

$$stride = prev\_addr - effective\ address \quad (1)$$

선인출 명령이 발생한 경우, 선인출 주소에 해당하는 데이타가 이미 캐시에 들어 있는지 검사하고, 만일 캐시 안에 포함되어 있으면(PH: prefetch hit) 선인출 동작을 수행하지 않고, 만일 캐시 안에 포함되어 있지 않으면 (PM: prefetch miss) 실제로 선인출 동작을 수행한다.

RPT 방식은 데이타의 주소간격(stride)을 계산할 때 주소에 할당되는 비트들을 모두 사용하기 때문에, 캐시 라인크기보다 적은 주소간격에 해당하는 데이타의 경우에(이 데이타는 이미 캐시 안에 들어와 있음에도 불구하고) 선인출 명령을 발생시키는 문제점이 있으며, 그 결과 선인출 명령에 포함된 주소가 캐시에 들어 있는지 검사하는 빈도(PH의 수)가 증가한다.

예를 들어, 크기가 256×256인 영상을 16×16의 서브블록 단위로 처리하는 경우를 생각해 보자. 한 픽셀의 크기가 한 바이트이고 영상 데이타가 행 우선으로 메모리에 1차원 배열에 할당된다고 가정하면, 영상 데이타를 저장하는 데 필요한 공간은 64K 바이트이다. 그 중에서 첫번째 서브블록의 메모리 인덱스는 그림 3과 같이 256×Y+X(X, Y: 0~15, X: 행 번호, Y: 열 번호)로 계산할 수 있다.

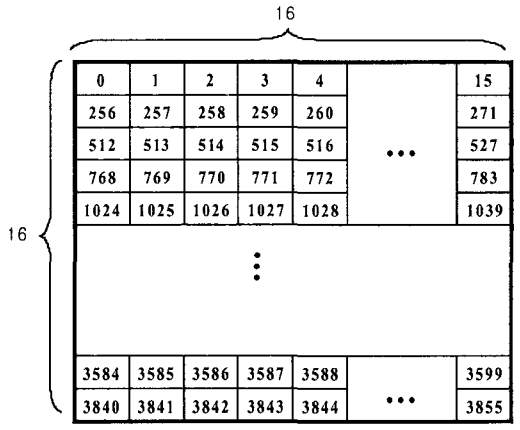

그림 3 256××256 영상 데이타의 첫번째 16×16 서브블록의 메모리 인덱스

캐시의 라인크기를 4 바이트, 그리고 동일한 명령어가 참조하는 데이타의 주소간격이 1이라고 가정한다. 이 경우에 데이타 배열 인덱스가 0~15, 256~271, …과 같이 한 개의 행을 처리할 때는 주소간격이 일정하지만, 각 행이 변경되는 시점에서는 주소간격이 불규칙하다. 이와 같은 가정 하에, RPT 방식의 선인출 동작 과정은 그림 4와 같다. PH는 선인출 명령을 발생시켰으나, 선인출 대상인 데이타가 이미 캐시 안에 있어서 실제로 메모리에서 선인출 하지 않음을 나타내며, PM은 캐시에서 미스가 발생되어 실제로 메모리에서 선인출 동작이 수행되는 것을 의미한다. 이와 같이 16×16 서브블록을 RPT 방식에 의하여 처리할 경우 선인출 명령은 모두 (16-3) + 15((16-2) = 223번 발생하지만, 그 중에서 48번(PM의 수)만 올바른 선인출 명령이고 나머지 175 번(PH의 수)은 이미 캐시 라인 안에 들어 있는 데이타에 대한 불필요한 선인출 명령이다.

주소선이 32비트, 라인크기는 4바이트로 가정할 때, 캐시의 라인크기를 고려한 경우의 블록주소간격(bstride)은 식 (2)로 계산할 수 있다.

$$bstride = (prev\_addr \text{ AND } FFFFFFFC_{(16)}) - (effective\ address \text{ AND } FFFFFFFC_{(16)}) \quad (2)$$

동일한 명령어가 참조하는 데이타의 주소간격이 1이라 하더라도, 캐시의 라인크기를 고려하면, 그림 5와 같이, 같은 캐시라인 안에서는 블록주소간격이 0이고, 캐시라인의 경계를 넘을 경우만 0이 아닌 블록주소간격을 얻을 수 있다. 따라서, 블록주소간격을 사용함으로써 그림 4의 PH에 해당하는 불필요한 선인출 명령이 발생하는 현상을 제거할 수 있다.

3.2 단속적 불규칙 데이타 참조 패턴에 대한 캐시 미스율 증가

그림 4에서와 같이 멀티미디어 데이타에서는 일정한

I : 초기(init) 상태, T : 전이(transient) 상태, S : 안정(steady) 상태

배열인덱스	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
주소간격	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
상태	I	T	S	S	S	S	S	S	S	S	S	S	S	S	S	S
결과				PH	PM	PH	PH	PH	PM	PH	PH	PH	PM	PH	PH	PH
배열인덱스	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271
주소간격	241	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
상태	I	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S
결과			PH	PH	PM	PH	PH	PH	PM	PH	PH	PH	PM	PH	PH	PH
배열인덱스	512	513	514	515	516	517	518	519	520	521	522	523	524	525	526	527
주소간격	241	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
상태	I	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S
결과			PH	PH	PM	PH	PH	PH	PM	PH	PH	PH	PM	PH	PH	PH
	...				...				...				...			

그림 4 주소간격이 1일 때 RPT 방식의 16×16 서브블록 처리 과정

address		0	4	8	12	16	20	28	32	36	40	44	48	56	60	64	68
stride		0	4	4	4	4	4	8	4	4	4	4	4	8	4	4	4
RPT	state	I	T	S	S	S	S	I	S	S	S	S	S	I	S	S	S
	H/M	M	M	M	H	H	H	M	M	H	H	H	H	M	M	H	H

그림 5 단속적 불규칙 참조 패턴에 대한 RPT 방식의 히트/미스(M:Miss, H:Hit)

주소간격을 유지하다가 차원이 바뀌는 경우에는 주소값이 불연속적으로 바뀌게 된다. 이러한 현상은 일반 데이타의 다중 루프로 구성된 배열 계산시에도 나타날 수 있다. 즉, 주소간격의 관점에서 보면 일정한 주소간격들 사이에 불규칙적인 주소간격이 단속적으로 나타나게 된다. 이러한 경우 그림 5에서와 같이 RPT 방식은 단속적 주소간격에서만 미스가 발생되어야 하는데, 상태가 초기상태로 바뀌기 때문에 연이은 규칙성을 갖는 데이타에 대해서도 미스가 발생하는 문제점이 있다(address : 32, 60인 경우).

이와 같은 패턴이 실제 멀티미디어 데이타 처리에서 발생되는 빈도를 알아보기 위하여 주소간격 패턴에 대한 통계적인 분석을 수행하였는데 자세한 설명은 5장에서 다룰 것이다.

4. M-RPT 선인출 알고리즘

주소간격이 캐시의 라인크기보다 적은 경우, 앞 절에서 제시한 바와 같이 블록주소간격에 0이 많이 포함된다. 따라서, 기존의 RPT 방식에 의한 캐시 선인출 구조를 수정할 필요가 있다. 제안하는 선인출 기법(M-RPT: Modified Reference Prediction Table)의 구조는 그림 6이며, 각 필드는 다음과 같은 용도로 사용된다.

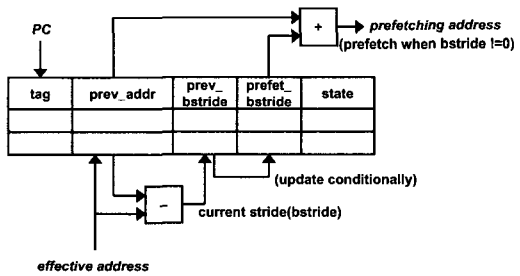

그림 6 M-RPT(Modified Reference Prediction Table)

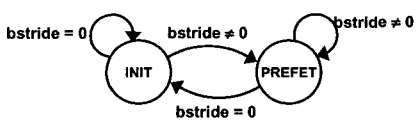

그림 7 M-RPT 방식의 상태도

- 태그(tag) 필드 : 적재/저장 명령어의 주소, 즉 명령어에 대한 PC의 값
- 이전블록주소(prev_addr) 필드: 이전의 반복문 안에서 동일한 명령어가 참조한 피연산자의 주소
- 이전블록주소간격(prev_bstride) 필드: 이전 피연산자의 주소와 현재 참조한 피연산자의 블록주소간격(current bstride)이 저장된다.
- 선인출블록주소간격(prefet_bstride) 필드: 선인출 주소로 사용하는 블록주소간격으로서, 상태에 따라 조건적으로 이전주소간격의 값으로 갱신된다. 초기값은 0이고, 일단 0이 아닌 블록주소간격으로 갱신되면 항상 0이 아닌 값을 갖는다.
- 상태 필드(state): 두 개의 상태를 가지며, 선인출 명령을 발생할지 여부를 결정한다.

명령어가 처음 실행될 때의 초기값으로 tag 필드는 명령어에 대한 PC의 값, prev_addr 필드는 현재 참조한 데이타의 블록주소, prev_bstride, prefet_bstride 두 개의 필드는 0(INIT_VALUE), state 필드는 INIT 상태이다.

그림 7은 M-RPT 방식의 상태도이다. 각 상태별 동작은 다음과 같다.

- 초기(INIT) 상태: 적재/저장 명령어가 처음 실행될 때의 초기 상태이며, 블록주소간격이 0이면 다시 초기상태로 전이한다. 이 상태에서는 계산된 블록주소간격(current bstride)을 prev_bstride에 저장한다. 만약 블록주소간격이 0이면 초기 상태로 유지하고, 블록주소간격이 0이 아니면 선인출 상태로 전이한다.
- 선인출(PREFET) 상태: prefet_bstride가 0이 아닌 경우에(prev_addr + prefet_bstride)의 주소에 대하여 선인출 명령을 발생한다. 만약 블록주소간격이 0인 경우에는 prefet_bstride을 변경하지 않고 초기상태로 전이하고, 블록주소간격이 0이 아닌 경우에는 상태는 변하지 않으며, 다음 조건을 만족하는 경우에 선인출 주소간격(prefet_bstride)을 갱신한다.

```
if ( (prev_bstride = current_bstride)
OR (prefet_bstride = INIT_VALUE))
    update prefet_bstride;
```

첫번째 조건은 0이 아닌 블록주소간격이 두 번 연속하여 일치하는 경우이고, 두번째 조건은 prefet_bstride가 초기값을 갖는 경우이다. prefet_bstride는 일단 0이

I : 초기(INIT)상태, P : 선인출(PREFET)상태

배열인덱스	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
블록주소간격	0	0	0	0	4	0	0	0	4	0	0	0	4	0	0	0
상태	I	I	I	I	P	I	I	I	P	I	I	I	P	I	I	I
결과					PM				PM				PM			
배열인덱스	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271
블록주소간격	244	0	0	0	4	0	0	0	4	0	0	0	4	0	0	0
상태	P	I	I	I	P	I	I	I	P	I	I	I	P	I	I	I
결과	PM				PM				PM				PM			
배열인덱스	512	513	514	515	516	517	518	519	520	521	522	523	524	525	526	527
블록주소간격	244	0	0	0	4	0	0	0	4	0	0	0	4	0	0	0
상태	P	I	I	I	P	I	I	I	P	I	I	I	P	I	I	I
결과	PM				PM				PM				PM			
	...				...				...				...			

그림 8 주소간격이 1일 때 M-RPT 방식의 16×16 서브블록 처리 과정

아닌 값으로 대치되면, 항상 0이 아닌 값을 갖는다.

그림 4의 예제 주소 데이타에 M-RPT 방식을 적용한 결과를 그림 8에 보였다. 참조하는 주소간격이 1인 경우, 블록주소간격에 0이 3 번씩 포함되어 있다. 행이 변경될 때 블록주소간격이 변하지만, 두 번 연속 블록주소간격이 변하지 않기 때문에 선인출 주소 계산에 사용되는 prefet_bstride는 갱신되지 않는다. 그림 10의 16×16 서브블록에 대하여, 블록주소간격이 0이 아닌 경우에는 선인출 명령을 발생하며 선인출 대상이 캐시 안에 존재하지 않아 실제로 메모리에서 캐시로 데이타를 가져오는 PM의 수가 첫번째 행에서 3 번, 나머지 행에서 15×4=60 번, 모두 63번 발생되며, RPT 방식에서 발생하는 불필요한 PH는 발생하지 않는다. 이와 같이 M-RPT 방식은 선인출 주소를 계산할 때 캐시 라인크기를 고려한 블록주소간격을 사용하기 때문에, 블록주소간격이 0인 경우 즉, 동일한 캐시라인에 포함되어 있는 데이타를 참조하는 경우는 선인출 명령을 발생시키지 않는다. 그리고, M-RPT 방식은 0이 아닌 블록주소간격이 두 번 연속하여 변경될 때 prefet_bstride를 갱신하기 때문에, 주소간격에 불규칙한 간격이 하나 포함되는 경우도 다음 라인부터 올바르게 선인출 주소를 예측할 수 있는 장점이 있다.

5. 참조 패턴 분석을 통한 RPT 방식과 M-RPT 방식의 비교

그림 9에서와 같이 특정한 PC(Program Counter)주소에 대하여 참조되는 데이타들의 주소간격 시퀀스를 윈도우 크기를 4로 하여 해당 윈도우 내에 나타날 수 있는 모든 조합을 각각의 패턴으로 정의하였으며, 윈도우를 1 데이타씩 시프트시켜 가면서 연속적으로 나타나

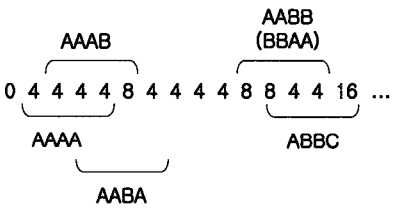

그림 9 주소간격 패턴들에 대한 예제

는 각 패턴에 대한 빈도를 모든 PC주소에 대하여 구하였다.

그림 10은 그림 9에서의 예제와 같은 주소간격 패턴들을 각 벤치마크 프로그램 별로 모든 PC 주소에 대해서 수행한 결과이다. 그림과 같이 벤치마크 프로그램 별로 빈도분포가 조금씩 다른데, 같은 벤치마크 프로그램에 대해서는 입력 데이타 파일에 따라 비슷한 분포를 가짐을 알 수 있다. 먼저 주소가 초기상태로부터 시작하는 주소간격 패턴을 initial이라 했으며, 이를 두 가지로 구분하였는데 initial-I의 패턴들은 RPT 방식과 M-RPT방식 모두가 미스가 발생되는 반면, initial-II의 패턴들은 RPT에서는 미스인데 M-RPT에서는 히트가 난다. 그리고 intermittent 비율이 uniform에 비해서는 낮지만 상당히 높은 비율을 갖고 있으며, 블록주소로 하는 경우 uniform의 비율이 증가하고 intermittent의 비율은 감소하는 경향이 있다. M-RPT의 경우에도 intermittent 4가지 패턴 중 "AAAB"에서만 미스가 한번 발생하기 때문에 블록주소로 하는 것이 미스율을 조금 더 줄일 수 있다.

어떤 PC에 대해서는 모두 같은 주소들을 갖는 경우도 있는데, 이때에는 주소간격 패턴이 "0000"이 되며 이는 "AAAA"로 카운트하였다(initial의 경우에는 I에서만 카운트하였다).

주소간격패턴	분류	mpeg2dec(mei)		mpeg2dec(waterski)		mpeg2enc		parser	
		address	block address	address	block address	address	block address	address	block address
I I A I A B I A B C I A A B I B A A	initial-I	14.1	14.0	15.0	14.6	6.25	6.26	35.2	35.8
I A A I A A A I A B A	initial-II	10.6	10.6	9.91	9.95	6.14	6.07	5.31	2.15
A A A A	uniform	73.7	74.8	73.6	74.5	84.4	86.2	34.6	39.5
A A A B A A B A A B A A B A A A	intermittent	1.42	0.37	1.36	0.46	3.16	0.85	7.34	3.61
A A B B A B B A A B A B	alternating	0.022	0.16	0.028	0.19	0.016	0.64	0.76	1.93
A B C C A B C B ... A B C D	random	0.15	0.044	0.20	0.050	0.006	0.026	16.8	16.9

그림 10 벤치마크 프로그램들에서의 주소간격 패턴의 빈도에 대한 통계적 분석(%)

표 1 블록주소 및 원래주소를 사용한 경우의 RPT/M-RPT 비교 (mpeg2dec)

방식	MRPT		RPT		
사용주소간격	블록주소	원래주소	원래주소	블록주소	블록주소
알고리즘 수정	원래 알고리즘 사용				0 주소간격 고려
Miss 수	765	835	877	6200	1134
Prefetch 수	3081956	4367090	3396307	2161007	2277745

블록주소일 경우에 M-RPT에 적용되어 있는 것처럼 0의 블록주소간격이 포함되어 있는 경우 즉, "8 0 8 0 8 0 8"은 "8 8 8 8"인 것으로 처리하였다(원래주소의 경우는 "4 4 4 4 4 4 4 … 4"에 해당함). 한편, RPT 방식의 경우에서는 알고리즘의 수정 없이 단순히 원래주소의 주소간격을 블록주소간격으로 바꾸게 되면 그림 13(b)에 보인 것과 같이 미스율이 급격하게 증가된다. 이는 블록주소를 사용할 경우 0인 주소간격이 많아지게 되는데 0 주소간격을 만나게 되면 RPT 방식의 알고리즘에 의해 안정상태가 유지되지 않아 선인출 명령을 발생시키지 않기 때문이다. 물론, RPT 방식의 이러한 문제점은 주소간격이 0인 경우를 고려할 수 있도록 알고리즘을 수정하여 해결할 수 있을 것이라고 생각될 수도 있다. 그러나 표 1에서 나타낸 바와 같이, RPT의 경우는 이와 같이 수정하더라도 블록주소를 사용하는 것이 원래주소를 사용하는 것보다 미스율이 여전히 더 크다. 이것은 RPT 방식의 알고리즘이 블록주소가 아닌 원래주소를 사용하는 경우에 대하여 가장 낮은 미스율을 갖도록 최적화된 알고리즘이라는 것을 보여준다. 결론적으로, 미스율 관점에서는 블록주소를 사용한 M-RPT < 원래주소를 사용한 M-RPT < 원래주소를 사용한 RPT < 블록주소를 사용한 RPT의 순서로 미스율이 커지며, 선인출 횟수 관점에서는 블록주소를 사용한 RPT < 블록주소를 사용한 M-RPT < 원래주소를 사용한 RPT < 원래주소를 사용한 M-RPT의 순서로 선인출 횟수가 증가한다.

M-RPT는 state가 매우 간단하기 때문에 실제로 초기에는 히트가 빨리 나기 시작하고 중간에도 AA와 같이 같은 주소간격이 두 번만 들어오게 되면 히트가 난다. intermittent한 주소간격이 하나 들어와도 RPT는 state가 바뀌는 반면 M-RPT에서는 선인출주소간격이 계속 유지 된다. 그림 11은 그림 10의 주소간격 패턴들 중 세가지 패턴들에 대하여 RPT방식과 M-RPT방식의 동작을 설명하고 있다. 주소간격 패턴이 IABA인 경우

RPT 방식에서는 전부 미스가 발생되고, M-RPT방식에서는 첫번째 A로 인해 선인출주소간격이 계속 유지가 되어 두 번째 A가 히트 됨을 볼 수 있다. AABA인 경우는 RPT 방식에서 intermittent한 B로 인해 상태가 초기상태로 되어 선인출 명령을 발생할 수 없으며, M-RPT 방식에서는 선인출주소간격을 유지하여 마지막 A 데이타가 히트가 된다.

그림 12는 각 주소간격 패턴들에 대해서 RPT 방식과 M-RPT방식의 캐시 미스율을 통계적 데이타를 이용하여 계산한 결과이다. 같은 분류에 속한 패턴들에 대해서

	RPT				M-RPT			
주소간격패턴	I	A	B	A	I	A	B	A
선인출주소간격	0	A	B	A	0	A	A	A
state	I	T	N	N	I	P	P	P
H/M	M	M	M	M	M	M	M	H
주소간격패턴	I	A	A	A	I	A	A	A
선인출주소간격	0	A	A	A	0	A	A	A
state	I	T	S	S	I	P	P	P
H/M	M	M	M	H	M	M	H	H
주소간격패턴	A	A	B	A	A	A	B	A
선인출주소간격	x	x	x	x	x	A	A	A
state	x	x	x	x	x	P	P	P
H/M	?	?	M	M	?	?	M	H

그림 11 세가지 주소간격 패턴에 대한 RPT 방식과 M-RPT 방식의 비교

분류	주소간격패턴	RPT	M-RPT	RPT miss ratio	M-RPT miss ratio
initial-I	I I A I A B I A B C I A A B I B A A	M M M M M M	M M M M M M	14.16	14.04
initial-II	I A A I A A A I A B A	M H M	H H H	5.30	0
uniform	A A A A	H	H	0	0
intermittent	A A A B A A B A A B A A B A A A	M M M M/H	M H M/H H	1.24	0.32
alternating	A A B B A B B A A B A B	M M M	M M M/H	0.02	0.15
random	A B C C A B C B ⋮ A B C D	M	M	0.10	0.03
Total				20.82	14.54

그림 12 주소간격 패턴의 분류 별 RPT 방식과 M-RPT 방식의 캐시 미스율 비교(%)

address		0	4	8	12	16	20	28	32	36	40	44	48	56	60	64	68
stride		0	4	4	4	4	4	8	4	4	4	4	4	8	4	4	4
RPT	state	I	T	S	S	S	S	I	S	S	S	S	S	I	S	S	S
	H/M	M	M	M	H	H	H	M	M	H	H	H	H	M	M	H	H
M-RPT	state	I	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
	H/M	M	M	H	H	H	H	M	H	H	H	H	H	M	H	H	H

(a) 원래주소를 사용한 경우

address		0	4	8	12	16	20	28	32	36	40	44	48	56	60	64	68
block stride		0	0	8	0	8	0	8	8	0	8	0	8	8	0	8	0
RPT	state	I	S	I	S	I	T	N	T	N	N	N	N	T	N	N	N
	H/M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
M-RPT	state	I	I	P	I	P	I	P	P	I	P	I	P	P	I	P	I
	H/M	M	M	M	H	H	H	H	H	H	H	H	H	H	H	H	H

(b) 블록주소를 사용한 경우

그림 13 단속적 불규칙 참조 패턴에 대한 RPT 방식과 M-RPT 방식의 비교

도 각각에 대한 빈도율을 적용하였으며, 'H'는 0, 'M'는 1, 'M/H'는 0.5를 각각 곱한 후 합계를 취하였다. 각 분류에 대하여 전반적으로 M-RPT 방식이 RPT 방식보다 미스율이 적게 나왔으며, 특히 intermittent한 경우에는 RPT의 25.8%의 낮은 미스율을 보이고 있다.

그림 13은 RPT 방식과 M-RPT 방식에서 단속적 불규칙 참조 패턴에 대한 히트와 미스를 비교한 것이다. RPT 방식에서는 주어진 address에서 7개의 미스가, M-RPT 방식에서는 4개의 미스가 발생되었다.

6. 시뮬레이션 및 성능 분석

6.1 시뮬레이션 환경

멀티미디어 데이타의 높은 지역성과 규칙적인 참조성을 바탕으로 한 데이타 선인출의 효과를 분석하기 위하여 DEC 사의 ATOM 시뮬레이터[13]를 이용하여 트레이스 구동 시뮬레이션을 수행하였다. 트레이스는 메모리 참조 명령어에 대하여 PC의 값, 적재 혹은 저장인지 여부, 그리고 필요한 피연산자의 유효주소로 구성되어 있다. 캐시 시뮬레이터는 이 트레이스를 입력으로 받아 캐시의 성능 및 선인출의 효과를 분석한다. 캐시 시뮬레이터는 위스콘신 대학에서 개발한 Dinero Ⅲ[14]에 RPT 방법과 M-RPT 방법을 추가하여 사용하였다. 명령어 캐시와 데이타 캐시가 분리되어 있는 캐시 구조를 실험 대상으로 하였고 데이타 캐시의 성능만 측정하였다. 캐시 시뮬레이터는 캐시크기, 블록크기, 사상함수 그리고 교체 알고리즘 등의 매개 변수를 입력으로 캐시 분석 결과를 생성한다. 표 2에 분석 모델을 위한 본 연구와 관련된 캐시 시뮬레이터의 매개 변수들과 값을 제시하였다. 대표적인 멀티미디어 벤치마크 프로그램인 MPEG (mpeg2enc, mpeg2dec)과 SpecInt2000의 일반 벤치마

표 2 분석 모델을 위한 매개 변수의 값

parameter	description
D-cache size	64k~128k
block size	4byte~32byte
word size	4byte
associativity	directed-mapped cache
write-policy	copy back
bus_width	block size

표 3 벤치마크 프로그램 특징

특성	benchmark program	description
멀티미디어	mpeg2encoder	digital video와 audio 압축에 관한 표준 도구
	mpeg2decoder	
SpecInt2000(일반)	Parser	Word processing
	Gap	Group Theory, interpreter

크 프로그램인 parser와 gap을 대상으로 1천만번의 메모리 참조 명령어에 대하여 실험하였으며, 표 3은 벤치마크 프로그램들의 특징을 나타낸다.

제안하는 M-RPT 방식과 기존의 RPT 방식, 8-step neighbor, opt. neighbor 방식의 선인출 효과를 비교 평가하기 위하여, 다음과 같은 평가 항목을 선정하였다.

- 선인출 명령 히트 수(PH): 선인출 알고리즘이 발생한 선인출 명령의 대상이 이미 캐시 안에 존재하여 실제로 선인출 동작이 필요 없는 선인출 명령의 수
- 선인출 명령 미스 수(PM): 선인출 알고리즘이 발생한 선인출 명령의 대상이 캐시 안에 존재하지 않아 실제로 주기억장치에서 선인출하는 선인출 명령의 수
- 캐시 미스 수(DM: Demand Miss): 프로그램이 실행하면서 데이타 액세스를 필요로 하였으나, 캐시에서 미스가 발생한 수. DM을 프로그램의 전체 메모리 액세스 수로 나누면 캐시 미스율이다.
- 버스 사용량: 캐시 메모리와 주기억장치사이의 버스에 데이타를 실어 나르는 회수를 나타낸다.(버스 사용량은 캐시 미스 수와 선인출 수 및 쓰기 정책(write policy)으로 채택한 copy back을 모두 반영한 결과이다.)

6.2 성능 분석

그림 14는 캐시크기가 64K바이트, 128K바이트이고 캐시 라인크기가 16바이트, RPT와 M-RTP 방식의 엔트리 크기는 1024인 경우에 mpeg2enc와 parser 벤치마크 프로그램의 전체 선인출 명령 발생 수에 대한 선인출 히트 수(PH)와 선인출 미스 수(PM)를 그래프로 나타낸 것이다. 멀티미디어 벤치마크인 경우(mpeg2enc) M-RPT 방식은 기존의 RPT 방식에 비하여, PH가 평균 68.8% 감소하였다. 이것은 M-RPT 방식이 불필요한 캐시 선인출 명령의 발생을 68.8% 이상 줄일 수 있음을 의미하고, 선인출 대상이 캐시 안에 존재하는가를 검사하는 선인출을 담당하는 하드웨어 로직의 부담이 감소

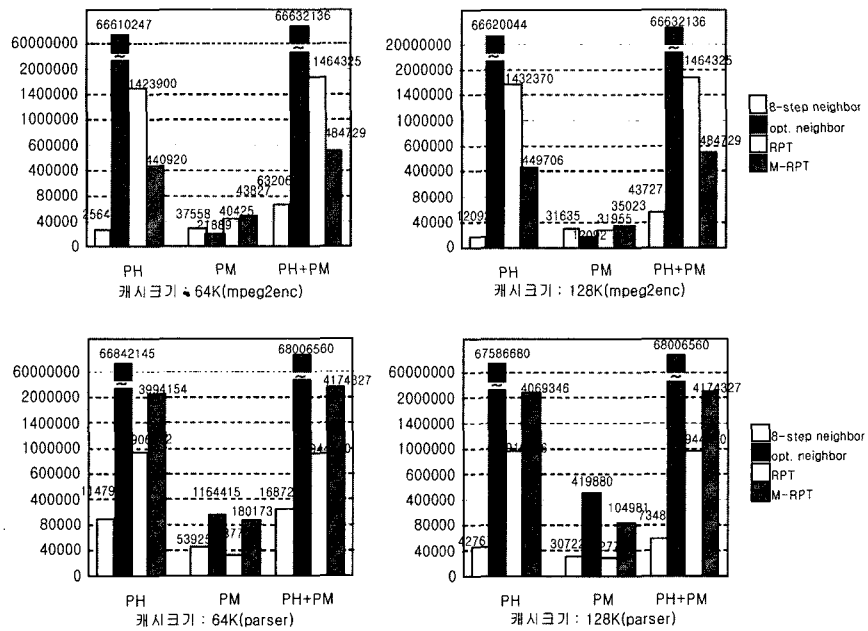

그림 14 선인출 명령 발생 수의 비교 (캐시크기=64K바이트, 128K바이트, 라인크기=16바이트)

함을 의미한다. PM은 M-RPT 방식이 더 많이 발생하였다. 따라서, M-RPT 방식이 실질적으로 주기억장치에서 캐시로 선인출을 더 많이 수행한다는 것을 알 수 있다. 일반 벤치마크인 경우(parser) M-RPT 방식이 RPT 방식에 비하여 PH가 평균 4.422배로 증가하였는데, 이는 M-RPT 방법이 그림 7의 상태도에 의하여 RPT 방법보다 선인출 명령을 더 많이 발생하며 또한 연속적인 데이타를 참조하지 않는 일반 벤치마크의 특징 때문인 것을 알 수 있다.

전체적인 선인출 명령 발생 수(PH+PM)도 멀티미디어 벤치마크인 경우 M-RPT 방식이 RPT 방식보다 33%~66.9% 정도로 감소하였고, 일반 벤치마크인 경우는 4.419배로 증가하였다.

opt. neighbor 방식은 다른 방식들과 비교하여 전체적인 선인출 명령 발생 수가 현저히 증가된 것을 볼 수 있다. 반면 8-step *neighbor* 방식에서는 *opt. neighbor* 방식의 선인출 명령 발생 수를 줄이기 위한 방안으로 8번의 참조에 걸쳐 분산하여 수행되고, 최대 한번의 선인출 명령이 발생되도록 하여 선인출 수를 줄일 수 있으나, 이 방식은 다른 방식들과 비교하여 캐시 미스율이 높다는 단점이 있다(그림 15 참조).

그림 15는 벤치마크 프로그램들에 대하여 캐시크기별로 캐시 라인크기를 변화시키면서 각각에 대한 캐시 미스율을 나타낸 것이다. mpeg2enc와 mpeg2dec은 캐시의 라인크기와 무관하게 RPT 방식과 비교하여 M-RPT 방식의 성능이 우수한 결과를 보였다. 따라서,

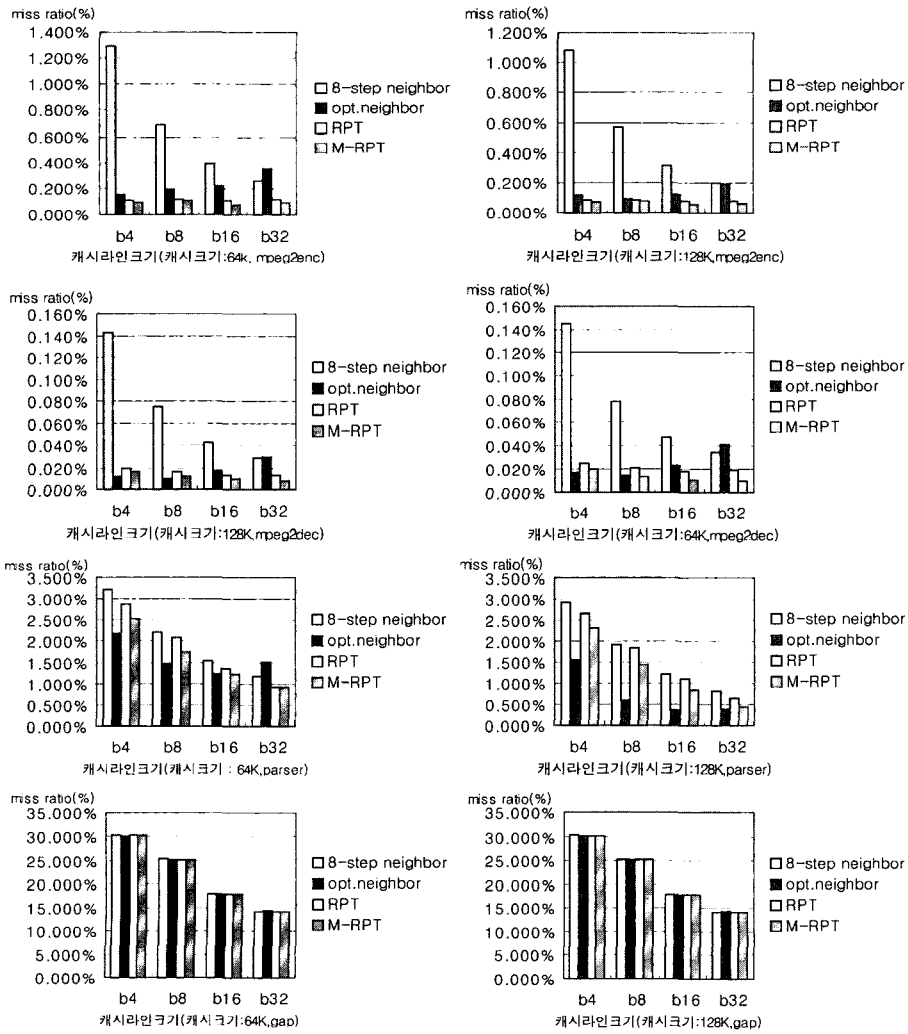

그림 15 벤치마크의 캐시 미스율

M-RPT의 경우 PM의 수가 증가하였으나, 올바로 데이타가 선인출 되어 프로그램 실행시의 캐시 미스수가 감소하였다고 판단할 수 있다. 캐시 미스율은 M-RPT방식이 RPT 방식과 비교하여 평균 29%정도 성능이 향상되었다. 8-step neighbor와 opt. neighbor 방식은 벤치마크 프로그램에 따라 다른 성능 차이를 나타내는데, 멀티미디어 벤치마크인 경우 캐시 미스율의 효과가 8-step neighbor보다 opt. neighbor가 캐시 라인크기가 작을수록 성능이 좋아지지만 캐시 라인크기가 클수록 차이가 줄어드는 것을 볼 수 있다. 일반 벤치마크인 parser와 gap에 대해서는 멀티미디어 벤치마크와 다르게 캐시 미스수가 전반적으로 증가하여 선인출의 효과가 감소함을 알 수 있다.

그림 16은 벤치마크 프로그램들의 버스 사용량을 나타낸 것이다. 버스 사용량은 캐시 미스 수와 선인출 수 및 copy back을 모두 반영한 결과이다. 캐시 미스율에 있어서 효과가 좋은 opt. neighbor와 M-RPT 선인출 방식은 선인출 횟수가 증가되어 버스 사용량이 증가되는 단점이 있다. 멀티미디어 벤치마크 프로그램의 경우, M-RPT 방식을 RPT 방식과 비교해 보면 버스 사용량이 평균 0.03% 증가 하였다.

M-RPT방식에 의한 전체 수행시간을 평가하기 위하여 동적 분석을 수행하였다. 캐시 성능 향상에 따른 프로그램 수행 시간은 캐시 미스율 자체만으로도 어느 정도 그 경향을 예측하는 것이 가능하지만, 선인출을 발생시킨 시점으로부터 선인출된 데이타가 실제로 캐시 내

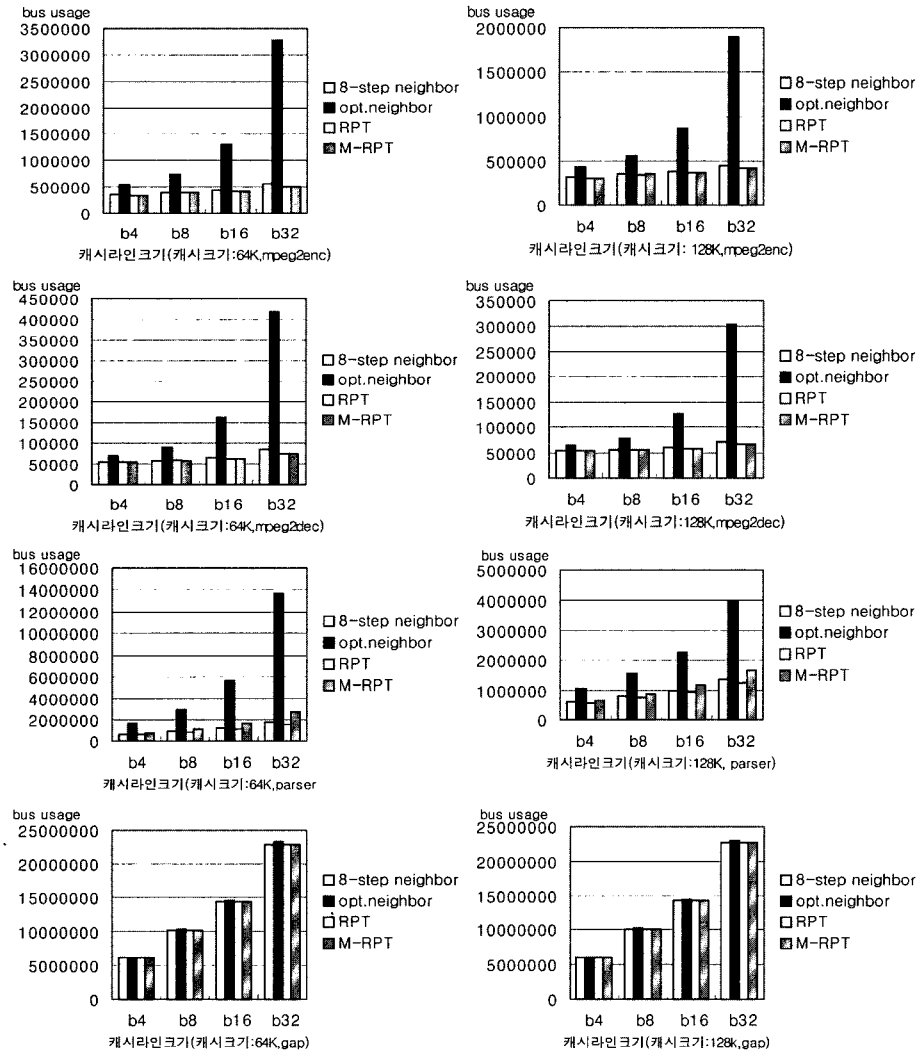

그림 16 벤치마크의 버스 사용량

부로 저장되어 다음 번 참조 시에 히트하기까지는 유한한 개수의 사이클이 지나야 하고, 선인출을 위한 메모리 참조가 프로그램 실행에 의한 메모리 참조의 타이밍에도 영향을 미칠 수 있기 때문에 미스율 만으로 계산한 수행 시간과 다소 차이가 날 수 있다. 본 논문에서는 이러한 영향을 고려할 수 있도록 Dinero III를 변형한 시뮬레이터를 이용하여 실험을 수행하였다.

캐시의 동적 동작은 하위 레벨의 메모리 모델을 어떻게 설정하느냐에 따라 많은 영향을 받는데, 본 논문에서 사용한 메모리 모델은 다음과 같다[11].

- 캐시 히트 = 1사이클
- 캐시 미스(demand 및 prefetch) = 19cycle
- 메모리 지연 시간 (memory latency time) = 15cycle
- 전송 시간 (burst bus transfer time) = 4cycle
- 한번에 버스를 사용할 수 있는 참조 수 = 1 (no split bus)
- 메모리 요청 인터리빙(memory request interleaving) = 사용

선인출 참조는 가장 빠른 요청부터 순차적으로 처리되며, 선인출이 진행되는 동안에도 다음 번 명령어의 인출이 가능하도록 하였다. 또한, 프로그램 요구(Demand)에 의한 캐시미스인 경우는 해당 메모리 참조가 완료될 때까지 명령어 인출을 대기하도록 하였다. 선인출 큐의 크기는 4로 설정하였고, 선인출 큐가 모두 채워진 상태에서 또 다른 선인출 요청이 발생하면 가장 먼저 접수된 선인출 요청의 처리가 완료될 때까지 기다리도록 하였다. 프로그램 요구에 의한 캐시미스인 경우는 가장 우선적으로 참조를 진행시키도록 하되, 먼저 접수된 선인출 참조에 의한 전송이 버스상에서 진행 중인 경우에는 이것을 먼저 완료한 후 미스에 의한 참조를 처리하도록 하였다.

각 선인출 방식에 대한 전체 수행시간을 비교하기 위하여 메모리 참조 지연 시간(MADT : memory access delay time)을 가지고 성능을 비교하였으며, 식(3)과 같이 계산된다.

$$MADT = \frac{Total\ Cycles}{Total\ Instructions} - 1 \qquad (3)$$

그림 17은 멀티미디어 벤치마크 프로그램인 mpeg2enc와 mpeg2dec을 캐시 크기는 64K, 128K로 하고 캐시 라인크기를 변화시키면서 비교하였다. RPT 방식과 M-RPT 방식을 비교해 보면 M-RPT 방식이 평균 35.8% MADT가 적게 나오며, 이는 전체 수행시간에 있어서 M-RPT 방식이 RPT 방식보다 성능이 향상되었음을 나타내준다. 한편, opt. neighbor의 경우 나머지 3가지 방식과는 달리 캐시 라인크기가 커질수록 MADT가 나빠지는데, 이것은 과다한 선인출로 인하여 선인출 대기 시간이 길어져서 사이클 수가 증가하기 때문이다. 실제로 MADT의 발생 원인을 1) 프로그램 요구에 의한 캐시 미스 처리를 위해서 대기한 사이클 수와, 2) 선인출 큐가 모두 찬 상태에서 또 다른 선인출 요구가 들어올 경우 이를 위해 대기한 사이클 수의 두 가지로 구분하여 측정한 결과, opt. neighbor를 제외한 나머지 3가지 방식에서는 2)에 의한 대기 사이클이 거의 발생하지 않은 반면, opt. neighbor에서는 매우 높은 퍼센트로 2)에 의한 메모리 참조 지연이 발생하고 있음을 볼 수 있었다. 멀티미디어 데이타의 경우 M-RPT의 MADT가 가장 나은 결과를 보인다. 동적 시뮬레이션을 고려하지 않은 캐시 미스율에서는 8-step neighbor방식이 극명하게 나쁜 결과를 보였던 것에 반해, MADT에서는 다른 방식과 큰 차이가 없는 결과를 보이고 있는 것은 8-step neighbor는 MPEG 데이타에 최적화된 NBrow 값을 적용하여 simulation 하였으므로, 프로그램 요구가 들어오기 전에 해당 주소의 데이타를 선인출할 수 있는 반면, 다른 방식에서는 프로그램 요구에 의한 참조 요구가 들어온 이후에 선인출 데이타가 캐시에 적재하는 일이 생길 수 있어 선인출로 인한 캐시 미스를 방지하지 못하는 경우가 생기기 때문이다. 그림 12의 결과는 미리 정해진 포맷을 갖는 데이타에 대해서는 8-step neighbor 방식이 매우 우수한 선인출 방식임을 보여 주는 것이기도 하지만, 다른 한편으로 M RPT 방식이 neighbor등의 방식에서와 같이 특정한 영상 크기(NBrow) 또는 미리 형식을 위한 최적화 없이도 그보다 더 우수한 성능을 내고 있다는 것을 보여주는 결과라고 할 수 있다. 한편, 멀티미디어 데이타가 아닌 일반 데이타에 대해서는 opt. neighbor를 제외한 3가지 방식들이 모두 유사한 성능을 나타내고 있다. 즉, 불규칙한 주소간격이 대부분을 차지하는 경우에 대해서는 여러 가지 선인출 방식간에 큰 차이가 없다는 것을 보여준다.

6.3 오버헤드 비교

본 논문에서 제안된 방식은 하드웨어 선인출 방식으로서 소프트웨어 선인출 방식에 비해 부가적으로 온칩에 추가되는 회로들이 있다. 두 방식의 오버헤드를 비교해 보면 먼저, 기존의 RPT 방식에서는 1024개의 엔트리를 가지는 RPT 테이블은 8Kbyte 데이타 캐시와 대등한 비용 부담을 갖는다고 할 수 있다[11]. 왜냐하면, 이전주소(prev_addr)와 주소간격(stride)이 각 엔트리 당 각각 4byte의 크기를 차지하며, 태그는 데이타 캐시의 태그 필드를 2-bit 상태 전이 필드는 데이타 캐시의 상태 비트(status bit)를 각각 그대로 사용할 수 있기 때문이다. 또한 태그 비교기는 원래 데이타 캐시의 비교기를 사용할 수 있다. 이 외에 주소간격 계산을 위한 뺄셈기 및 선인출 주소 계산을 위한 가산기, 그리고 주소간격 갱신 여

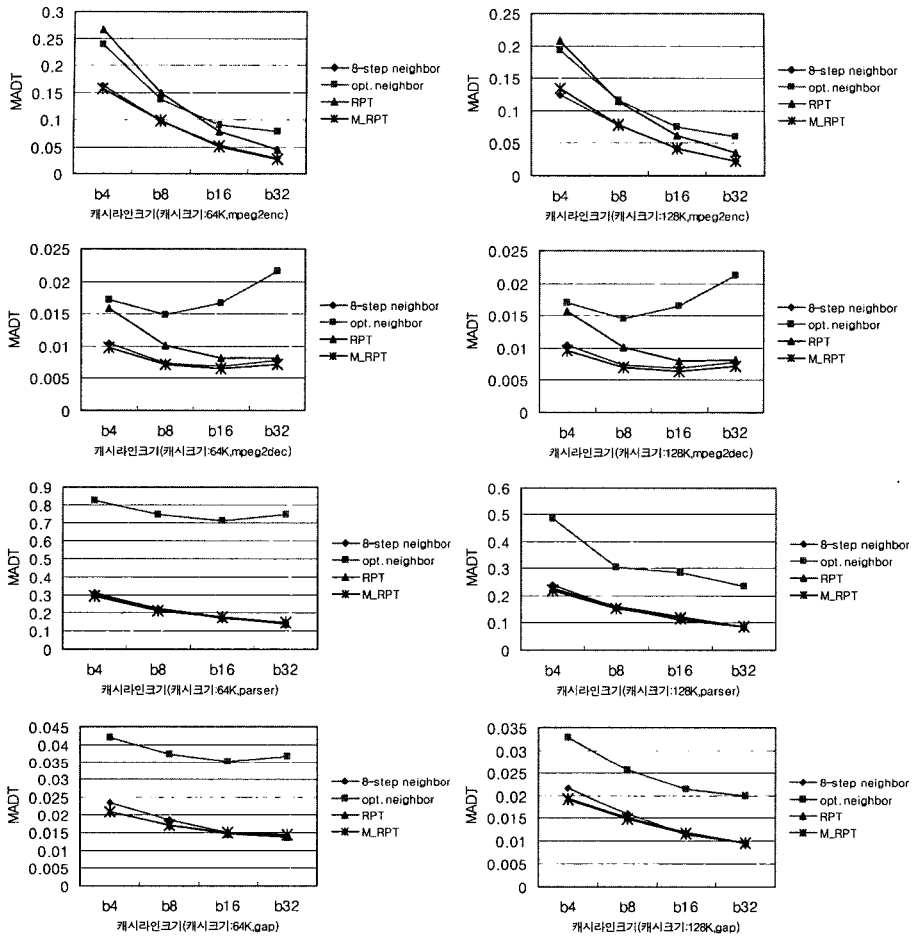

그림 17 멀티미디어 벤치마크의 MADT

부를 판단하기 위한 별도의 비교기가 필요한데, 레지스터 비트 동치(rbe : register bit equivalence)로 환산하여 이들 모두를 약 130rbe로 가정하였다[15,16].

뒷장에서 설명하게 될 M-RPT에서는 RPT 테이블이 조금 더 커지게 되는데, 기존 RPT의 주소간격 필드가 이전블록주소간격(prev_bstride)와 선인출블록주소간격(prefet_bstride)의 2개로 분리되기 때문이다. 태그 필드를 4byte로 가정하면, 한 RPT 라인의 데이타 비트의 크기는

RPT : 32(tag) + 32(prev_addr) + 32(stride) + 2(state) = 98 bits

M-RPT : 32(tag) + 32(prev_addr) + 64(stride) + 1(state) = 129 bits

가 되는데, [15]에서와 같이

$$RAM = 0.6\times(\#entries + L_{sense\_amp})\times(\#data\_bits + W_{driver}) \quad (4)$$

로 계산되어질 수 있고, 이 식 (4)를 적용하면,

RPT = $0.6\times(1024 + 6) \times (98 + 6) = 64{,}272rbe$

M-RPT = $0.6\times(1024 + 6) \times (129 + 6) = 80{,}430rbe$

로 평가된다. 또한, 64Kbyte directed mapped D-cache도 이와 유사하게 계산된다. (캐시 라인크기 = 8byte)

64Kbyte D-cache = $0.6\times(8192 + 6) \times (98 + 6)$
$= 511{,}555rbe$

본 논문에서 비교한 opt. neighbor와 8-step neighbor 방식의 경우는 인접한 주소를 계산하기 위해 가산기와 카운터 등만 추가되므로 이를 마찬가지로 $130rbe$ 정도로만 가정하였다. 이를 바탕으로 4가지 선인출 방식을 표 4에 비교하였다.

7. 결 론

캐시 기억장치에 데이타 선인출 기법은 프로세서가 연산을 수행하는 동안 필요한 오퍼랜드를 미리 캐시로

표 4 각 선인출 방식의 오버헤드 비교(64KB-8byte, mpeg2enc)

비교항목	8 step neighbor	Opt.neighbor	RPT	M_RPT
필요한 하드웨어	알고리즘, 수행을 위한 하드웨어		RPT 테이블, 상태도 처리 로직	
영상크기와의 의존성	영상크기에 선인출 대상이 의존함			
Area	511,685	511,685	575,957	592,115
미스율	0.489%	0.244%	0.767%	0.486%
MADT	0.0980	0.1373	0.1500	0.0973

적재해서 메모리 액세스 시간을 줄이는 효과적인 방법이다. 이러한 선인출 기법은 멀티미디어 데이타가 갖는 지역성과 규칙성을 바탕으로 하여 동작한다.

본 논문에서는 기존의 RPT 방식을 변형한 M-RPT 방식을 제안하였다. M-RPT 선인출 방법은 데이타의 선인출 주소를 계산할 때 캐시 라인크기를 고려하여 블록주소간격을 사용하였으며, 블록주소간격이 0이 아닌 경우에 선인출 명령을 발생시키고, 블록주소간격이 0인 경우는 이미 캐시 안에 존재하는 데이타를 참조하는 경우이므로 선인출 명령을 발생시키지 않는다. 또한 메모리를 참조하는 주소간격이 두 번 연속 틀려질 때 선인출 주소를 변경하도록 선인출 알고리즘을 수정하였으며, 그 결과 메모리 참조 주소 시퀀스에 불규칙한 주소간격이 한 개 포함되는 경우에도 올바른 데이타를 선인출할 수 있는 능력을 가질 수 있게 되었다. 멀티미디어 벤치마크 프로그램의 경우, M-RPT 방식은 RPT 방식에 비하여 두 가지 장점을 가지는데, 하나는 전체 선인출 명령 발생 수가 33~66.9% 감소하였으며, 또 하나는 캐시 미스율이 평균 29% 성능이 향상되었다. 반면에 버스 사용량은 평균 0.03% 증가하였다. 또한, 동적 분석의 결과는 M-RPT가 멀티미디어 데이타에 대해서 가장 적은 메모리 참조 지연 시간을 갖는다. 일반 벤치마크 프로그램의 경우, 어느 방식을 사용하더라도 선인출 효과가 적음을 알 수 있었다.

하드웨어에 기반을 둔 현재까지의 캐시 선인출 알고리즘들은 프로그램의 수행 과정 중에 선인출에 관한 정보를 분석하여 선인출 대상인 주소를 발생한다. 그렇지만, 캐시로 선인출된 데이타가 실제로 사용이 되었는지 아니면 캐시만 오염시키고 사용되지 않았는지에 대한 캐시 선인출 결과를 사용하지 않는다. 앞으로, 이러한 캐시 선인출 결과를 활용하여 잘못된 데이타에 대한 선인출을 방지함으로써, 버스 트래픽을 감소시키는 연구가 수행되어야 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 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] H. G, A. R, and 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] R. Cucchiara, M. Piccardi and A. Prati, "Hardware Prefetching Technique for Cache Memories in Multimedia Applications," *in proc. Of IEEE Intl. Workshop on Computer Architectures for Machine Perception (CAMP)*, 2000.

[8] R. Cucchiara, M. Piccardi and A. Prati, "Temporal Analysis of cache Prefetching Strategics for Multimedia Applications," *in Proc. Of IEEE Intl. Performance, Computing and Communications Conf. (IPCCC)*, pp. 311-318, Apr. 2001.

[9] R. Cucchiara, A. Prati, M. Piccardi, "Data-type dependent cache prefetching for MPEG applications," *in Proc. Of IEEE Intl. Performance, Computing and communications Conf. (IPCCC)*, pp. 115-122, Apr. 2002.

[10] 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.

[11] 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.

[12] H. J. Moon, "A Cache Managing Strategy for Fast Media Data Access," *Ph.D. thesis, Computer Science Department Chungbuk, National University*, Feb. 2003.

[13] A. Srivastava and A. Eustace, "ATOM : A System for Building Customized Program Analysis Tools," *Proceedings of the ACM SIGPLAN 94*, pp. 196-205, 1994.
[14] M. D. Hill, "Dinero Ⅲ Cache Simulator," *Technical Report, Department Computer Science, University of Wisconsin, Madison,* 1990.
[15] 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.
[16] J. M. Mulder, N. T. Quach, and M. J. Flynn, "An Area Model for On-Chip Memories and its Applications," *IEEE Journal of Solid State Circuits*, Vol. 26, No 2, pp. 98-106, Feb. 1991.
[17] J. L. Baer and T-Fu Chen, "An Effective on-Chip Preloading Scheme to Reduce data Access Penalty," *ACM*, pp. 176-186, 1991.
[18] K.I. Farkas and N.P. Jouppi, "Complexity/ Performance Tradeoffs Architecture," *Proc. of the Int. Symp. on computer architecture*, pp. 211-222, Apr. 1994.

전 영 숙
1996년 한남대학교 전자계산학과(학사) 1998년 한남대학교 컴퓨터 공학과(석사) 2002년 충북대학교 컴퓨터과학과 박사수료. 관심분야는 고성능 컴퓨터 구조, 병렬 처리, 메모리 시스템

문 현 주
1995년 충북대학교 자연과학대학 컴퓨터과학과(학사). 1997년 충북대학교 대학원 전자계산학과(석사). 2003년 충북대학교 대학원 전자계산학과(박사). 2003년~현재 나사렛대학교 정보과학부 전임 강사. 관심분야는 데이타 선인출, 메모리 시스템, 멀티미디어 시스템, 전력 분석 모델, 그리드 컴퓨팅

김 석 일
1975년 서울대학교 학사학위 취득. 1975년~1995년 국방과학연구소 선임 연구원으로 근무. 1985년~1989년 : 미국 North Carolina State University에서 공학박사 학위 취득. 1990년~현재 충북대학교 컴퓨터 과학과 교수로 재직중. 관심분야는 병렬처리 컴퓨터 구조, 슈퍼컴퓨팅, 이기종 분산처리, 시각장애 사용자 인터페이스 등

전 중 남
1990년 연세대학교 전자공학과(박사). 1996년~1998년 미국 텍사스 A&M University 연구원. 1990년~현재 충북대학교 전기전자컴퓨터 공학부 교수. 관심분야는 컴퓨터 구조, 임베디드 시스템