Journal of Korea Multimedia Society Vol. 7, No. 8, August 2004(pp. 1086-1096)

이차원 블록 구조에 근거한 선인출 기법

김 석 주†

요 약

스트리밍 데이터를 다루는 멀티미디어 응용의 경우 캐시 관리 측면에서 데이터의 시간적 지역성이 약하여 캐시의 효율이 감소하게 된다. 이는 캐시에 적재된 데이터가 대부분 다시 사용되지 않고 바뀌게 됨을 의미한다. 반면에 이러한 데이터들은 참조 명령에 따른 규칙적인 접근 패턴을 갖고 있는 경우가 많다. 이 논문에서는 약한 시간적 지역성을 나타내는 멀티미디어 응용 데이터에서도 통상적으로 내재된 메모리 참조의 규칙성을 적극적으로 활용하기 위해 동적 등차 참조 선인출 기법의 기능과 함께 이차원 배열 형식(블록)을 찾을 수 있는 방법을 제안한다. 제안된 방법은 블록 구조를 인식하고 이에 따라 선인출 주소를 계산 하므로 블록 참조 예측 기법 (BRPT: block-reference-prediction-technique)이라고 명한다. BRPT는 새로운 규칙으로 인해 선인출 기구가 더 복잡하지만 블록 패턴이 많은 응용의 경우 메모리 참조 시간을 크게 줄이는 것을 확인하였다.

A Multimedia Data Prefetching Based on 2 Dimensional Block Structure

Kim, Seok-Ju†

ABSTRACT

In case of a multimedia application which deals with streaming data, in terms of cache management, cache loses its efficiency due to weak temporal locality of the data. This means that when data have been brought into cache, much of the data are supposed to be replaced without being accessed again during its service. However, there is a good chance that such multimedia data has a commanding locality in it. In this paper, to take advantage of the memory reference regularity which typically innates even in the multimedia data showing up its weak temporal locality, a method is suggested. The suggested method with the feature of dynamic regular-stride reference prefetching can identify for 2-dimensional array format(block pattern). The suggested method is named as block-reference-prediction-technique (BRPT) since it identifies a block pattern and place an address to be prefetched by the regulation of the block format. BRPT proved to be reassuring to reduce memory reference time significantly for applications having abundant block patterns although new rule has complicated the prefetching system even further.

Key words: Dynamic Prefetching Unit(동적 선인출 기구), Streaming Pattern(스트리밍 형식), Cache System(캐시 시스템), Regulated Memory Reference(규칙적 메모리 참조)

※ 교신저자(Corresponding Author) : 김석주, 주소 : 대전광역시 서구 복수동 333번지(302-715), 전화 : 042)580-6212, FAX : 042)580-6159, E-mail : kimse@hcc.ac.kr
접수일 : 2003년 9월 1일, 완료일 : 2004년 1월 26일
† 정회원, 혜천대학 멀티미디어 전공 조교수
※ 본 논문은 2002년도 혜천대학 교내 학술연구비 지원에 의하여 수행된 것임.

1. 서 론

Moore의 법칙에 따르면 프로세서의 속도는 매 18개월마다 2배가 되나 메모리의 속도는 매년 1/10의 비율로 향상되어 왔다. 이러한 차이로 인해 메모리 참조 시간을 빠르게 하는 것이 프로세서의 성능을

지속적으로 향상시키는데 있어 매우 중요한 변수가 되고 있다. 최근에는 웹 등의 영향으로 정지영상, 동영상, 음성, 3차원 그래픽 및 애니메이션 등을 처리하는 멀티미디어 응용 프로그램이 컴퓨터 시스템이 처리하는 작업에서 매우 큰 비중을 차지하고 있다.

Fritts의 연구[1] 결과에 따르면 멀티미디어 응용 프로그램의 전체 수행 시간 중 25~50%가 메모리 참조에 소요되는 것으로 나타났다. 특히 멀티미디어 응용에 관련된 데이터는 대단위 배열의 형태 또는 스트리밍 패턴을 갖는 경우가 많다[2,3]. 이것은 멀티미디어 응용에서 적절한 성능을 얻기 위한 캐시 시스템의 비용이 상당히 높을 것이라는 것을 의미한다. 대용량 데이터를 다루는 경우 일정한 시점에 요구되는 데이터의 양이 급격히 증가하여 대규모의 데이터에 대한 참조가 한꺼번에 이루어지므로 이때 캐시에서 필요한 데이터를 찾지 못하는 경우가 많다. 또한 스트리밍 패턴의 경우 캐시 관리 관점에서 필요한 블록을 캐시에 교체해도 새로 적재된 블록을 제거하기 전에 다시 참조할 확률이 적다. 따라서 캐시의 시간적 지역성(temporal locality)이 떨어지며 이는 캐시의 성능을 저하시킨다[4,5].

본 연구에서는 이러한 문제에 대처하기 위해 메모리 참조 시 보다 정교한 패턴을 활용하는 방안을 찾고자 한다. 일반적으로 대부분의 멀티미디어 응용 프로그램은 대규모의 멀티미디어 데이터에 대하여 단순한 연산을 적용하는 형태를 가지므로 명령에 따라 규칙적으로 접근될 가능성, 즉 명령 지역성(commanding locality)이 매우 높다[4,5]. 스트리밍 데이터에서는 메모리 참조 시 매우 강한 규칙성을 보여준다[3,6]. 그 이유는 스트리밍 패턴을 갖는 데이터의 구조가 보통 배열 형태를 나타내며 배열 구조를 다루는데 있어 반복적인 블록 구조가 일반적인 수단이기 때문일 것이다.

대량의 데이터에 대한 단순처리를 주로 하는 멀티미디어 처리에서 메모리 참조 속도는 CPU의 처리속도 못지않게 큰 비중을 차지하므로 멀티미디어 응용 프로그램의 처리 속도를 높이기 위해서는 메모리 참조 속도를 향상시키기 위한 연구가 필수적이다. 최근에는 동적 선인출 기구가 기존의 캐시 제어기와 함께 사용되는 경우가 많다[7,8].

동적 선인출 기구는 일련의 메모리 참조 기록을 바탕으로 지정된 메모리의 일부를 캐시로 미리 적재하여 캐시 미스를 줄이는 역할을 한다[9-11]. 동적 선인출 기구는 불필요한 데이터들이 캐시를 점유할 확률이 많은 멀티미디어 데이터 처리에서 매우 효과적인 방법으로 알려져 있다[12].

본 논문에서는 동적 등차 참조 선인출 기법의 기능을 제공하면서 데이터의 블록 패턴을 추적하여 필요한 블록을 캐시에 적재하는 동적 선인출 방법을 제안한다. 제안된 방법은 데이터 참조 과정에서 멀티미디어 데이터에 자주 나타나는 블록 형태를 찾을 수 있으며 선인출의 효율을 높일 수 있다. 이를 블록 참조 예측 기법(BRPT: Block Reference Prediction Technique)이라 명하였다 이 방법의 특징은 메모리 참조과정에서 블록 패턴을 발견하여 이를 참조 규칙의 기본으로 삼는 것이다.

2장에서는 본 연구에서 제안된 방법의 기본 방법이 되는 선인출 기법으로 Chen에 의해 제안된 참조 예측 기법(RPT: Reference Prediction Technique)[9]에 대해 설명한다. 3장에서는 블록 참조 예측 기법(BRPT: Block Reference Prediction Technique)을 제안하고 이의 원리를 예를 들어 설명한다. BRPT의 특징은 블록 형태에 따라 바뀌는 참조 간격(stride)을 검출하는 것이다. BRPT는 참조 거리 변화에 대해 기존의 방법에 몇 가지 규칙을 추가함으로써 대부분의 경우에 적절한 선인출 주소를 생성할 수 있다. 4장에서는 제안된 BRPT의 주소 간격 계산 과정을 설명한다. 5장에서는 BRPT의 효과를 조사하는 실험에 대해 설명한다.

6장은 결론으로 서브 블록에 대한 메모리 참조가 멀티미디어 응용에서 매우 흔한 것을 고려한다면 BRPT가 메모리의 참조시간을 줄이는 효과적인 방법이 될 수 있음을 주장하였다.

2. 선행 연구

동적 선인출 기법은 캐시 제어기가 프로그램을 실행하는 과정에서 메모리 참조 기록을 관찰하여 미래에 사용될 메모리 블록을 사용되는 시점보다 이른 시간에 캐시로 읽어 들여 실제로 데이터가 사용될 시점에 발생하는 캐시 미스를 방지하려는 방법이다. 동적 선인출을 위해 하드웨어는 메모리 참조 명령어의 수행을 관찰하고 어떤 규칙성에 의거하여 선인출 명령을 발생시킨다.

참조 예측 기법(RPT: Reference Prediction Technique)[9]은 메모리 참조에 대한 과거 기록으로부터

참조되는 메모리 주소 간격을 찾아 미래에 참조될 메모리 주소를 계산하여 선인출 함으로써 등차 참조형 데이터의 캐시적중률을 높일 수 있는 기법이다. 이 방법은 반복되는 메모리 참조들 간의 거리를 인식할 수 있으며 인접한 블록 뿐 아니라 진행 중인 참조들 사이의 거리에 따라 일정 거리(stride)로 연관된 블록들을 적재하도록 예측할 수 있다.

RPT의 기본적인 가정은 통상적으로 반복문의 영향을 받는 명령어와 연관된 메모리 참조는 또 다시 발생할 수 있으며 인접한 참조들 사이의 거리를 얻을 수 있다는 것이다. 이러한 관점에서 건너뛴 거리(stride)가 선택되며 참조예측표(Reference Prediction Table)에 명령어의 주소와 함께 기록된다. 연속하여 참조가 같은 방식으로 이루어지는 한 이 거리는 적절한 것으로 간주되며 다음 번의 참조 주소는 현재 주소에 거리를 더함으로써 쉽게 얻을 수 있다. 그러한 주소의 선택이 맞지 않은 것으로 판명되면 사용중인 거리를 달리하는 기회가 주어진다. 그러한 검사의 일환으로 상태 필드가 적절한 거리를 찾기 위해 어떤 적합한 단계를 취할 지에 대한 정보를 갖고 있다.

본 논문에서는 RPT를 기본적인 참조 예측 방법으로 사용한다.

3. 블록 참조 예측 기법(BRPT)

3.1 메모리 참조 규칙성

기존의 선인출 기법 중 가장 좋은 성능을 지니는 RPT 기법은 응용 프로그램이 참조하는 메모리 주소 간의 규칙성을 활용하여 선인출 할 메모리 주소를 예측한다. 많은 응용들이 데이터 원소들을 규칙성 있게 참조하기 때문에 RPT 기법은 우수한 성능을 발휘할 수 있다. 그러나 이러한 RPT 기법에도 반드시 수반되는 약점이 있다. 그것은 RPT 기법이 참조되는 메모리 주소의 규칙성을 1차원으로 해석하는 것이다. 이미지나 영상 처리에 관련된 많은 멀티미디어 응용은 2차원이나 그 이상의 차원으로 표현되는 데이터를 다루기 때문에 RPT 기법은 한계를 지닌다.

그림 1은 RPT 기법의 약점을 설명하기 위한 예이다. 영상처리나 이미지 압축 기법 등에서는 크기가 큰 이미지를 일반적으로 8*8 크기의 서브 블록으로 나누어 서브 블록 단위로 처리를 한다. 그림 1(a)은

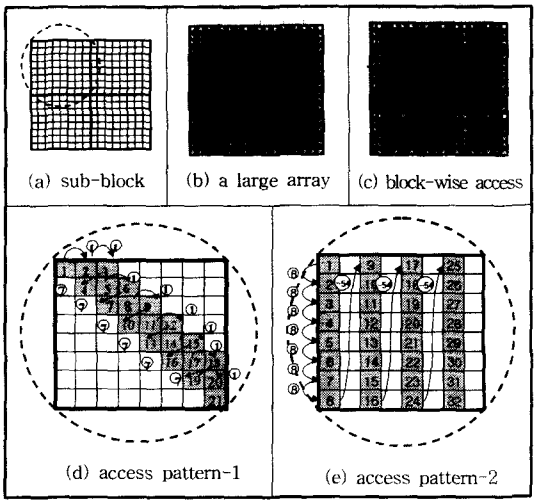


그림 1. 메모리 참조 규칙성

16*16 크기의 2차원 배열로 표현되는 데이터를 8*8의 서브 블록으로 분할하여 처리하는 것을 나타내며, 그림 1(b)-1(e)은 각 서브 블록 안에서 데이터 원소들이 규칙적으로 참조되는 패턴의 예를 나타낸다. 각 그림에서 배열 원소 안의 숫자는 원소들이 참조되는 순서를 나타내며 각 데이터의 크기를 1로 가정하고 메모리에 행 우선순위로 적재되어 있다고 가정한다.

RPT 기법에서는 반복문에서 메모리 참조 명령이 반복될 때마다 참조된 메모리 주소를 기록한다. 그리고 반복간의 메모리 주소 차이(stride)를 계산하여 이 값이 일정하게 유지되는 동안 다음에 참조될 메모리 주소를 예측하고 선인출 한다.

그림 1(b),(c)에서 행렬 곱셈의 두 번째 행렬의 참조와 같은 열기준으로 데이터 참조가 이루어진다고 가정한다. 그림 1(c) 에서 서브 블록이 증가하는 경우 RPT 기법은 메모리 참조의 규칙성이 저하된다. 그림에서 하얀 색으로 표시된 부분이 선인출 에러가 발생하는 곳을 나타낸다. 그러나 참조 패턴을 2차원으로 분석하면 이러한 에러를 막을 수 있다.

그림 1(e)에서는 stride의 순서가 8,8, ... ,(-54), 8, 8, ...,(-54)의 순서가 된다. 이 경우, stride 8이 유지되는 동안에는 다음 번 메모리 주소를 예측할 수 있지만 stride가 -54가 되면 규칙성이 상실되었으므로 다음 번 메모리 주소를 예측하지 못한다. 그림 1(d)의 경우는 패턴은 다르지만 그림 1(e)와 같이 stride가 반복되는 경우이다.

그림 1(d), (e)와 같이 서브 블록 안에서 하나 이상

의 stride가 번갈아 나타나는 경우에도 규칙성을 파악하고 메모리 주소를 예측하기 위한 기법을 다음 장에서 제안한다.

3.2 메모리 주소의 기록

선인출 기법에서 선인출 할 메모리 주소는 메모리 참조 명령어 m_i가 참조한 메모리 주소와 주소간격을 더하여 계산한다. m_i가 수행되는 동안 주소간격은 변할 수 있으며, 매 반복마다 주소간격을 계산하기 위해서는 명령어 m_i가 이전 반복문에서 참조한 메모리 주소를 저장해야 한다. 이를 위하여 참조예측표를 활용한다.

그림 2에서 태그(tag) 필드는 각 load/store 명령어의 프로그램 카운터의 값을 저장하여 명령어들을 구분하기 위한 용도로 사용된다. previous_address 필드는 load/store 명령어가 참조한 데이터의 메모리 주소를 저장하는 필드이다. stride와 state 필드는 각각 참조된 메모리 주소 사이의 간격과, 메모리 주소 간격이 규칙성을 유지하는가를 나타내는 필드이다.

기존의 RPT 기법에서는 stride와 state 필드를 각각 하나씩만 가지고 있지만 이 논문에서 제안하는 BRPT를 구현하기 위해 stride와 state 필드를 2개씩으로 확장하였다. L1의 Stride 필드는 메모리 참조 명령이 반복될 때마다 갱신된다. 만일 이 값이 일정하게 유지되면 이를 나타내기 위하여 L1의 state 필드를 steady로 기록한다. 일정하게 유지되던 L1 Stride 필드의 값이 변하면 L1 state 필드는 규칙성이 상실되었음을 나타내는 init 상태로 전이되고 이 때 바뀐 stride는 블록 형태를 나타낼 수 있으므로 L2 stride로 복사한다.

tag	previous_address	stride	state

(1) Reference Prediction Table

tag	previous _address	L_1		L_2	
		stride	state	stride	state

(2) Block Reference Prediction Table for Two Dimensional Array Structure

그림 2. 참조 예측표의 비교

이렇게 할 때, 그림 1(e)에서 본 8,8,...,8과 같은 규칙성은 L1 필드에 의해서 파악되고, -54의 규칙성은 L2 필드에 의해서 파악되므로 전체적인 메모리 참조가 규칙성이 있는 것으로 판단할 수 있다. 이러한 stride의 값을 저장하게 되는 필드를 L2_stride 라하고 이 값을 얻기 위해 단계적인 검사의 상태를 저장하는 필드를 L2_state라 한다. L2_state는 initial, first-change, transient, steady의 상태 값을 갖는다. 그림 2는 RPT 기법과 이 논문에서 제안하는 BRPT 기법의 참조예측표를 비교하고 있다.

4. 선인출 주소의 계산

4.1 RPT의 선인출 주소 계산

메모리 참조 명령어 m_i가 처음 수행될 때, 참조예측표는 명령어 m_i를 위하여 하나의 행을 할당하고, 명령어 m_i의 프로그램 카운터와 m_i가 현재 참조하는 데이터 메모리 주소로 태그 필드와 직전주소 필드를 각각 초기화한다. 주소간격 필드와 상태 필드를 각각 0과 초기 상태인 initial 상태로 초기화하고 직전주소에 주소간격을 더하여 선인출 주소를 계산한다.

반복문에 의하여 명령어 m_i가 다시 수행되고 m_i에 해당하는 행이 여전히 참조예측표에 남아 있다면, 명령어 m_i에 대하여 참조예측표에 기록된 내용과 m_i가 현재 참조한 메모리 주소를 이용하여 참조예측표의 기록을 갱신한다. 기록 갱신 과정은 첫째, 명령어 m_i가 현재 참조한 메모리 주소와 직전주소 필드의 차이 값을 계산하여 이 값으로 주소간격 필드를 갱신한다. 둘째, m_i가 현재 참조한 메모리 주소로 직전주소 필드를 갱신한다. 셋째 명령어 m_i가 이전 반복문을 수행할 때 계산한 선인출 주소와 m_i가 현재 참조한 메모리 주소를 비교하여 선인출 예측 성공 여부를 결정한다.

RPT에서 참조간격을 구하려면 인접한 메모리 참조 주소 사이에서 최소한 두 번 같은 간격이 연속하여 발견되어야 한다. 같은 간격이 2번 연속 반복되는 것이 주소간격의 값을 갱신하기 위한 필요조건이다. 어떤 주소 간격의 값은 다른 주소 간격을 찾기 전까지는 바뀌지 않고 그대로 유지된다. 이러한 방침은 2번 연속하여 주소 예측이 실패하기 전까지는 그 간격이 의미 있다고 간주하는 것이다.

4.2 BRPT의 선인출 주소 계산

그림 3은 BRPT를 적용 할 때 선인출 주소가 얻어지는 과정을 보여주는 예이다. 그림 3의 예에서는 메모리 참조가 2차원 배열 형태에서 행우선 순위로 진행된다고 가정하였다.

어떤 행의 끝에 해당하는 데이터를 참조한 후 다음 행의 처음 원소까지의 거리에 해당하는 stride를 발생시키는 것이 제안된 BRTP가 수행 하고자하는 역할이다.

이를 위해 L2_state의 초기 값은 initial 이다. 그림 3의 예에서 첫 번째 행의 참조가 끝나고 2번째 행의 첫 번째 열의 값을 참조할 때 L1_state 가 steady에서 initial로 바뀌면 이때 L2_state를 first-change로 바꾸고 참조예측표에 저장된 이전주소를 이용하여 첫 번째 행의 마지막 행의 주소를 address_keep 변수(레지스터)에 저장한다. 그림 2에서 보인 BRPT 참조예측표 외에 2차원 규칙성을 찾기 위하여 특정 시점의 주소나 예측 적중회수를 기록하기 위한 변수를 저장하는 레지스터가 필요하며 address_keep 도 그러한 레지스터들의 하나이다.

L2_state가 first-change라는 것은 2번째 행에 포함된 원소를 참조할 개연성이 있다는 의미이다. 따라서 L1_state가 initial에서 steady로 바뀌면(이 상태변화는 다음 번의 참조가 L1_stride를 유지하면 발생) 그 행에 포함된 원소의 개수(즉 열의 개수)를 찾아내는 단서를 갖는다. 새로운 참조가 발생하면서 L1_stride가 변화가 없다면 (L1_state의 지속) 변수 hit_count를 1 증가시킨다. 따라서 매번 L1_stride 가 적중하여 steady의 상태가 될 때마다 hit_count 변수에 횟수가 기록된다.

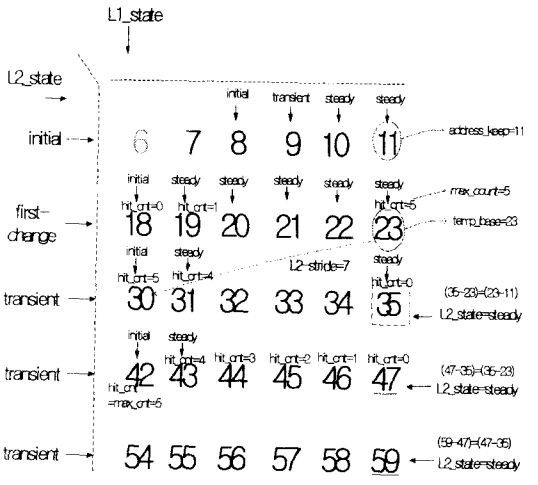


그림 3. BRPT의 적용 예

만일 L1_state가 initial로 바뀌면 이것은 3번째 행의 첫 번째 값을 참조한 것으로 볼 수 있다. 따라서 L1_state가 steady에서 initial로 바뀌면 이때 L2_state를 first-change에서 transient로 바꾸고 두 번째 행의 마지막 행의 주소를 temp_base 변수에 저장한다. 또한 hit_count 변수를 max_count에 저장한다.

max_count는 최초로 L1_state의 변화가 있고 L1_state의 값이 steady로 지속된 값을 가지므로 한 행에 포함된 열의 개수와 연관이 있다. 또한 현재 참조된 주소(3번째 행의 첫 번째 원소)와 참조예측표에 저장된 previous_address필드의 값(2번째 행의 마지막 원소의 주소로 간주됨) 차이를 L2_stride에 저장한다. 그리고 참조 예측표의 previous_address에 대한 갱신이 이루어지고 다음 참조를 기다린다.

L2_state가 transient인 경우 L1_state를 관찰하여 steady가 될 때마다 hit_count를 하나씩 줄인다. 따라서 hit_count가 0이 되면 이것은 현재 참조된 데이터가 3번째 행의 마지막 원소임을 나타낸다. 이때 2차원 배열 형태를 인식하는 검사를 하는데 이것은 현재 참조된 주소(3번째 행의 마지막 원소의 주소로 간주됨)와 저장된 temp_base의 차이가 temp_base와 address_keep의 차이와 일치하는지를 비교함으로써 이루어진다.

만일 두 값이 일치하면 참조거리의 변화가 일어난 시점(즉 L1_state가 steady에서 initial로 바뀐 경우)에 같은 참조거리가 2번 나타났고 현재의 행에서 참조한 원소의 개수가 이전 행의 원소의 개수와 최소한 같다는 것을 나타낸다. 따라서 L1_stride가 계속 유지되는 데이터가 앞으로 더 있는지의 여부는 알 수 없지만 이 시점에서 취할 수 있는 가장 최선의 선택은 2차원 배열 형태를 발견한 것으로 판단하여 현재 참조된 주소를 3번째 행의 마지막 원소로 간주하는 것이다.

이때 L2_state는 transient에서 steady로 바뀌며 미리 저장된 L2_stride를 이용하여 선인출 주소를 계산하여 선인출 명령을 발생시킨다. 그러나 이렇게 계산된(L1_stride 대신에 L2_stride를 적용) 선인출 주소가 적절한 것인지를 확정할 수는 없다. 따라서 L1_state는 steady 상태로 유지된다. 이러한 상황에

서 다음 원소에 대한 메모리 참조가 발생한 후 실제 읽은 이 주소가 선인출 주소와 같은 것으로 판명되면 이것은 현재 참조된 원소가 4번째 행의 첫 번째 원소로 간주할 수 있다는 사실을 나타는 것이다.

예측된 주소가 일치하면 앞으로 지속될 데이터 흐름에서 2차원 배열 형태가 지속되는 지를 계속 검사해야 한다. 그러한 준비를 위해 temp_base에 저장된 값을 address_keep으로 옮기고 previous_address 필드에 저장된 값(세 번째 행의 마지막 원소의 주소로 간주됨)을 temp_base에 저장한 후 L2_state를 transient로 바꾼다. 또한 저장된 max_count 값을 hit_count에 넣는다. L2_state가 transient로 바뀐 상태에서 위에서 설명한 데로 새로운 행에 해당하는 데이터에서 마지막 원소를 찾아 L2_stride를 적용하는 과정을 반복한다.

만일 L2_state 가 transient 이고 hit_count가 0 이되지 않거나(행에 포함된 원소들의 개수가 달라진 경우) hit_count 가 0일 때 해당 주소와 temp_base의 차이가 temp_base와 address_keep의 차이와 같지 않은 경우(2차원 배열의 형태가 아닌 경우)에는 L2_state를 initial 상태로 전환한다. 따라서 연속된 데이터에서 2차원 배열 형태를 갖는 데이터가 있는지를 계속 관찰하게 된다.

그림 4는 위에서 설명한 BRPT의 동작과정을 C 프로그램으로 작성한 것을 정리하여 보인 것이다.

그림 5와 그림 6은 L1_state와 L2_state에 대한 변환 규칙을 보여준다.

4.3 BRPT의 하드웨어 요구 및 동작 특성

정확한 BRPT의 시간 복잡도와 공간 복잡도는 설계 단계에서 설계도구의 출력 값이나 시뮬레이션 등

```
Set_prefetch_address() {
    if (flag for applying L2_stride is set)
        prefetch_address=address+L2_stride;
                otherwise prefetch_address=address+L1_stride;
}
adjust_temporary_value() {
    if(L1_state is INIT)and(L2_state is FIRST-CHANGE)
        address_keep= previous_address;
    if(L1_state is INIT)and (L2_state is TRANSIENT)
        temp_base = previous_address;
        max_hit_count=hit_count;
    if(L1_state is STEADY)and (L2_state is FIRST_CHANGE))
        hit_count++;
    if(L1_state is STEADY) and (L2_state is TRANSIENT)
        if((--hit_count) is 0) and
          (address is (2* temp_base — address_keep))
              set flag for applying L2_stride;
              L2_state=STEADY;
              address_keep=temp_base_address ;
               temp_base=address;
}
update_state() {
    if(L1_state is STEADY) and (L2_state is STEADY)
       L1_state= INIT; L2_state =TRANSIENT;
    update L1_state according to the rule shown in transition
              diagram.
 As this update proceeds, if there is a change STEADY to INIT
    if((L1_state is INIT) L1_state= FIRST-CHANGE;
    if((L1_state is FIRST-CHANGE) L1_state= TRANSIENT;
 }
```

```
Update_stride() {
      if(L1_state is INIT) and (L2_state is TRANSIENT)
          L2_stride=address-previous_address;
      In case prefetching was correct,
      if((L1_state is TRANSIT))
          L1_stride= address-previous_address;
      In case prefetching was incorrect,
      if((L1_state is TRANSIENT)or (L1_state is NO_PRED))
          L1_stride= address-previous_address;
}
Block Reference Prediction {
      For a coming memory reference instruction,
          conduct its operation.
      Using PC address of the instruction as tag,
          Seek out relevant slot in RPT.
      If there is no corresponding slot,
          Reserve one slot (either replace one or
                          assign a vacant slot)
      Comparing prefetching address to current address of
          referenced element,
          check out whether prefetching is correct or not.
      Update L1_state and L2_state( call update_state()).
      Update L1_stride and L2_stride( call update_stride()).
      Set temporary variables necessary to searching for 2
          dimensional array structure
           (call adjust_temporary_value()).
      Set prefetching address( call set_prefetch_address()).
      Issue prefetching instruction if necessary.
      Replace previous_address field with current referenced
          address.
}
```

그림 4. Block Reference Prediction Technique

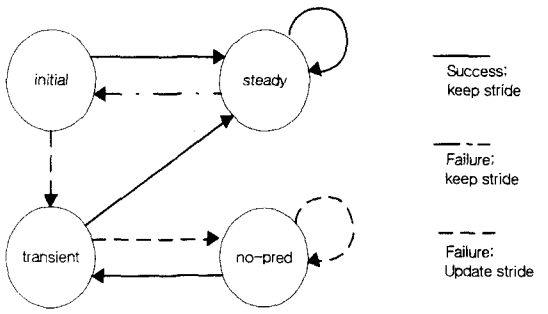


그림 5. L1_state 변환표

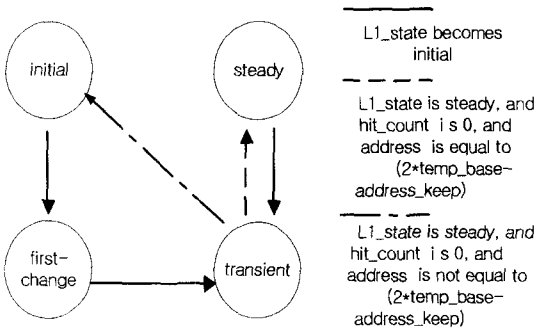


그림 6. L2_state 변환표

의 방법을 통해 확인할 수 있다. 본 논문에서는 그러한 연구 결과 없이 시간 복잡도와 공간 복잡도에 관련하여 중요한 특성만을 기술한다.

시간 복잡도의 측면에서, 그림 2에서 보인 대로 참조 예측표에는 L2_state와 L2_stride의 2개의 필드가 추가되며 참조 예측표의 각각의 slot에 대해 임시 변수를 저장하기 위한 4개의 레지스터가 필요하다. 이러한 요구를 반영하여 그림 7에 BRPT를 구현하기 위한 하드웨어를 나타내었다. 참조예측표의 크기는 1차원에 비해 2배가 필요하며 기타 상태 제어에 따른 회로가 필요하다. 2차원 규칙을 찾기 위한 추가적인 연산 시간은 동적 선인출 기구가 CPU와 다른 별도의 프로세서나 회로로 구성되고 비교적 간단한 연산임을 감안하면 부정적인 영향을 주지 않을 것으로 판단된다.

공간 복잡도의 측면에서, 임시 변수를 저장하는 레지스터들은 메모리 페이지의 정보와 함께 하드디스크에 저장되도록 하였으며 프로그램이 메모리에 적재될 때 관련 정보는 L1_cache로 적재된다. 이 때 어떤 load/store 명령과 다른 load/store 명령과의 CPU 사이클의 개수가 L1_cache의 latency보다 큰

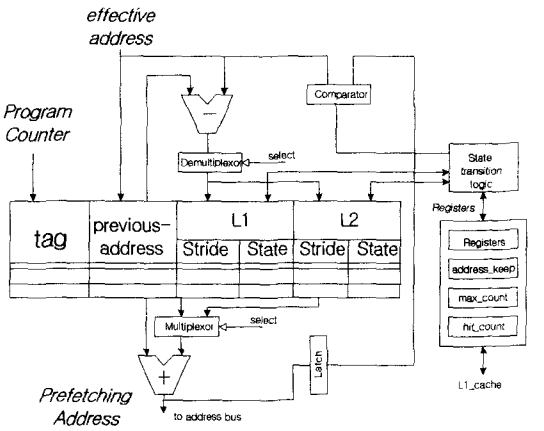


그림 7. BRPT를 구현하기 위한 하드웨어 기구

경우에는 그림 7에 보인대로 회로 내에 포함된 4 개의 레지스터만을 사용하더라도 처리가 가능할 것이다. 그러나 이 경우 임시 변수들을 저장하기 위한 레지스터의 개수는 줄어드나 메모리 bandwidth를 요구하는 단점이 있다.

BRPT는 1차원 규칙성이 발견된 경우 이러한 규칙성을 바탕으로 2차원 규칙성을 찾게 된다. 1차원 규칙성만 있는 경우에는 RPT와 같은 성능을 보이게 된다.

캐시나 BRPT를 구현할 때 요구되는 하드웨어 비용은 집적기술과 매핑 방법 등 여러 가지 설계 조건에 따라 달라질 수 있으므로 BRPT의 하드웨어 비용을 데이터 캐시 크기에 따른 비용과 상대적으로 비교한다.

Chen[9]에 의하면 RPT의 경우 추가되는 논리회로와 테이블의 오버헤드를 고려할 때 512 entries를 갖는 하드웨어 선인출 기구가 4K byte 데이터 캐시와 같은 크기이다. address space의 변화에 따라 캐시의 크기가 바뀔 수 있으나 512 entries를 갖는 RPT 선인출 기구의 테이블의 하나의 entry의 크기는 같은 512 entries를 갖는 데이터 캐시의 8 byte(tag과 data 포함) entry를 구현하는 크기와 같다고 가정할 수 있다. BRPT를 구현하는 선인출 기구에서 테이블의 크기가 추가되는 필드(L2-state)와 주변 회로의 overhead를 그림 7에 보인대로 최소한으로 가정하면 BRPT 엔트리당 오버헤드는 데이터 캐시에서 8 bytes의 두 배인 16 bytes를 구현하는 오버헤드에 상당한다고 추정할 수 있다. 따라서 512 entries를 갖

는 하드웨어 선인출 기구가 8K byte를 갖는 데이터 캐시와 비슷한 크기를 갖는다고 가정할 수 있다.

5. 실 험

BRPT의 성능을 분석하기 위하여 SimpleScalar 2.0 tool set의 sim-cache.c와 cache.c, main을 수정하여 sim-brpt라는 시뮬레이터를 새로 만들어 사용하였다. SimpleScalar(심플스칼라)는 execution-driven simulation 방식으로 시뮬레이터가 프로그램을 수행하면서 트레이스를 생성시킨다. 기존의 sim-cache 시뮬레이터는 선인출을 사용하지 않는 경우를 보여주므로 성능을 비교할 수 있었다.

벤치마크로는 MediaBench[13]에 포함된 mpeg2enc와 mpeg2dec의 소스를 SimpleScalar의 컴파일러로 컴파일 하여 실행파일을 만들어 시뮬레이터의 입력으로 사용하였다. 또 다른 벤치마크로 정수 원소로 구성된 10×10 행렬을 곱하기 위한 프로그램을 컴파일 하여 사용하였다. 표 1은 실험에서 사용한 MediaBench에 포함된 벤치마크 프로그램의 옵션을 나타낸다.

실험 지표로서 선인출을 적용하여 필요한 블록을 L1-데이터 캐시에 가져온다고 가정하고 L1-데이터 캐시의 캐시 미스율을 조사하였다. L1-데이터 캐시는 각 블록의 크기는 32Byte, 매핑 방식은 Direct Mapping, Replacement 방침은 LRU(Least Recently Used)을 가정하고 블록 엔트리의 개수를 64,128,256, 512,1024로 바꿔가며 조사 하였다.

이와는 별도로 블록 엔트리의 개수가 256이며 L1-데이터 캐시와 같은 사양을 갖는 L1-명령 캐시가 존재하며, 하위레벨 캐시인 L2-캐시는 1024 블록 엔트리, 블록의 크기가 64 byte이고 4-way set associative 방식으로 동작하며 데이터와 명령어가 공유하는 것으로 가정하였다. L2-캐시의 형태를 바꿔서 실험하였으나 실험 결과에 거의 영향을 주지 않으므로 그 내용은 생략한다. 메모리 크기는 SimpleScalar 머신의 기본 메모리인 2^{31} bytes(=2048Mbytes)로 정하였으며 메모리 블록의 크기는 64K bytes이다.

선인출 절차로 L1-표 3은 캐시의 크기가 변할 때 BRPT를 사용한 경우와 선인출을 사용하지 않은 경우의 캐시 미스율을 조사한 것이다. 표 3의 내용을 그림 8에 표시 하였다. 데이터 캐시에 선인출 주소를 갖는 블록이 있으면 아무런 조치도 취하지 않고 블록 없는 경우에만 미리 적재 하는 요구를 하도록 하였다. 이때 적재 요구가 하위 캐시를 거쳐 메모리까지 전달될 수 있으므로 선인출 주소가 유효한 프로그램 데이터 세그먼트에 포함되는지의 여부 등을 조사하여 범위를 벗어난 값은 배재하도록 하였다.

표 2는 L1-데이터 캐시의 크기가 256 엔트리 일 때 sim-brpt로 얻어진 결과를 나타낸다. 결과 값의 첫 번째 행은 해당 심플스칼라 실행파일에 포함된 명령어의 개수를 나타내며 두 번째 행은 메모리 참조 명령어의 개수를 나타낸다. 심플스칼라 시뮬레이터에서는 메모리 참조 명령어 중 double word단위로 참조하는 명령어는 캐시를 2번 접근하는 것으로 간주되므로 실제 캐시 접근 회수는 이 값들보다 약간 많은 것으로 나타난다. 세 번째 행의 값은 메모리 참조의 규칙성을 발견하여 선인출 명령을 발생시킨 개수이다. 즉 L1_state가 steady이거나 L2_state가 steady일 때를 의미한다. 이때 캐시에 선인출 주소로서 계산된 값을 포함한 블록이 있는 경우를

표 1. 벤치마크 프로그램

program	description	program module	data source	command / parameter
MPEG	transforming tool for digital video stream	mpeg2enc	image frames of Y.U.V format	mpeg2enc test.par out.m2v
		mpeg2dec	mpeg video streams	mpeg2dec -f -b test.m2v -o0 new%d

표 2. 실험 결과 예(BRPT 적용, L1-data cache size: 256 entries)

	matrix multiplication	mpeg2enc	mpeg2dec
number of instructions	24495	13989562	8017819
number of memory reference	8229 load:3730 store:4499	38868548 load: 36189524 store: 2679024	2339099 load: 1422920 store: 916179
number of prediction	5585 probe_hit:5191 probe_miss: 394	35087365 probe_hit: 26102259 probe_miss: 8985106	1950142 probe_hit: 1940052 probe_miss: 10090
L2-stride hit	194	29598	204
number of entries of RPT	460	4237	2029
cache miss rate	0.0111 (w/o pref: 0.0570)	0.0279 (w/o pref: 0.0801)	0.0254 (w/o pref: 0.0275)

표 3. 캐시 미스율

cache size: (entries)	matrix multiplication		mpeg2enc		mpeg2dec	
	BRPT	Basic	BRPT	Baic	BRPT	Basic
64	0.0174	0.0605	0.0954	0.1041	0.0570	0.0588
128	0.0139	0.0597	0.0729	0.0918	0.0318	0.0379
256	0.0111	0.0570	0.0279	0.0801	0.0188	0.0275
512	0.0095	0.0554	0.0239	0.0783	0.0147	0.0210
1024	0.0075	0.0534	0.0165	0.0313	0.0024	0.0108

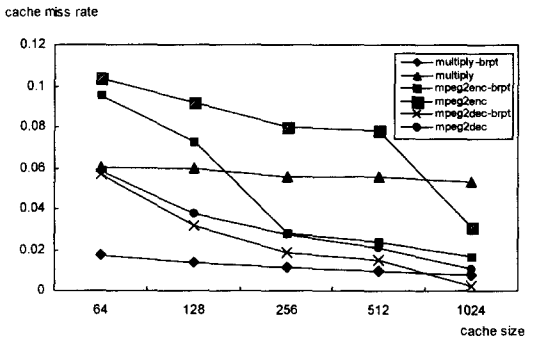


그림 8. 캐시 미스율의비교

probe_hit로 그렇지 않은 경우를 probe_miss로 나타내었다.

네 번째 행의 L2-stride hit는 블록 형태의 규칙성을 찾아 L2_stride의 값을 적용하여 선인출 한 경우 실제 그 값을 참조한 경우를 나타낸다. 다섯 번째 행의 RPT(참조예측표)의 크기는 참조예측표의 포함되는 엔트리의 개수에 제약을 주지 않을 때 저장되는 엔트리의 개수를 의미한다. 즉 이 실험에서는 참조예측표의 크기를 무한대로 가정하였다. 여섯 번째 행은 캐시 미스율을 나타내며 선인출을 사용하지 않은 경우의 캐시 미스율도 함께 보였다.

표 3은 캐시의 크기가 변할 때 BRPT를 사용한 경우와 선인출을 사용하지 않은 경우의 캐시 미스율을 조사한 것이다. 표 3의 내용을 그림 8에 표시하였다.

그림 8을 관찰하면 working set이 작은 비교적 작은 행렬 곱하기(multiply)의 경우 캐시의 크기가 작은 경우에도 일정하게 좋은 선인출 효과를 얻을 수 있음을 알 수 있다. 비디오 스트림 엔코더(mpeg2enc)의 경우 캐시가 충분한 working set을 획득하기 전에 선인출을 적용하면 작은 캐시로도 충분히 좋은 효과를 얻을 수 있음을 알 수 있다. BRPT를 적용하는데 필요한 Reference Prediction Table도 캐시이므로 구현하는데 일정량의 크기를 차지하겠지만 블록 참조 규칙성이 뚜렷한 multiply의 경우 캐시를 무작정 늘리는 것보다 선인출 기구를 구현 하는 것이 효과적임을 알 수 있다. 디코더(mpeg2dec)의 경우 다른 벤치마크 프로그램만큼 BRPT의 효과가 뚜렷하게 나타나지는 않았으나 일정한 효과가 있음을 알 수 있다.

BRPT의 캐시 미스 감소 효과를 RPT와 비교하기 위해 각각의 벤치마크 프로그램에 대해 수식 (1)을 사용하여 캐시 미스 발생 횟수가 감소하는 비율을 비교하였다. 여기서 C는 캐시 미스 발생율을 의미한다. 선인출의 효과가 뚜렷하게 나타나는 cache 크기가 256 엔트리인 경우를 조사하면 matrix multiplication의 RPT의 캐시 미스율이 0.0167 이므로 34%의 감소율을 보여준다. mpeg2enc의 경우 RPT의 경우 캐시 미스율이 0.0672로 58%가 감소된다. 이러한 높은 비율의 감소는 mpeg2enc의 경우 선인출을 하지 않는 경우 캐시의 write-back 비율이 0.0036으로 전형적인 스트리밍 패턴의 형태를 나타내며 만일 잘못된 선인출을 하면 바로 캐시 미스로 연결될 수 있으므로 블록 패턴을 적용한 BRPT가 더욱 유효함을 알 수 있다. mpeg2dec의 경우 감소율은 0.0001(0.1%)으로 BRPT와 RPT의 성능이 동일함을 알 수 있었다.

$$Cache\ Miss\ Reduction\ Rate = \frac{C_{RPT} - C_{BRPT}}{C_{RPT}} \quad (1)$$

그림 9는 캐시의 크기가 256 인 경우 얻어진 실험 지표를 바탕으로 평균 메모리 접근 시간을 수식 (2)에 의해 계산한 결과를 보여준다. 캐시에 대한 읽기/쓰기는 1 사이클에 이루어지며 메모리(여기서는 dl2/il2 캐시)는 20 사이클이 소요된다고 가정하였다. 심플스칼라 시뮬레이터에서 캐시는 운영 방식이 write back, write allocate 방식으로 운영되므로 다음의 식을 적용하였다. 선인출을 사용하지 않은 경우 write_back_rate(dirty bit를 set하는 비율)는 multiply의 경우 0.0236, mpeg2enc는 0.0036, mpeg2dec는 0.014이며 선인출을 사용한 경우 이 값들 보다 약간 적은 값들을 가지나 그 차이는 미미하였다.

평균 메모리 접근 시간(Average clock per memory access)

= penalty of hit + penalty of miss

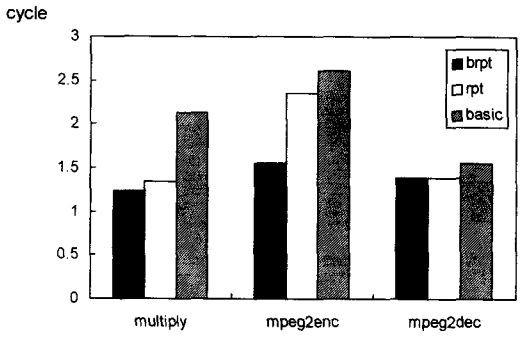


그림 9. 평균 메모리 접근 시간
(캐시 크기 : 256 엔트리)

= hit_rate×hit_time+miss_rate×((miss_time+hit_time)+write_back_rate×miss_time)
=(1-miss_rate)×1+miss_rate×(20+1+write_back_rate×20) (2)

그림 9에서 basic은 선인출을 사용하지 않는 경우를 나타낸다.

6. 결 론

높은 지역성을 가진 메모리 참조에 대해 보다 높은 규칙성을 찾고 이를 적용하여 필요한 블록을 캐시에 미리 적재하는 방법은 캐시의 오염을 줄이고 멀티미디어 데이터의 특성상 낮은 활용도를 보이는 캐시의 성능을 높일 수 있다. BRPT는 멀티미디어 응용에서 더욱 자주 나타나는 블록 패턴을 인식하여 이를 활용한다.

응용 프로그램의 크기가 증가하고 배열의 크기가 증가하는 추세에서 서브 블록별로 계산을 수행하는 응용이 많음을 고려할 때 BRPT가 메모리 접근 시간을 줄이는 효과 적이고 유익한 방법이 되리라 예상된다.

참 고 문 헌

[1] J. Fritts, "Multi-Level Memory Prefetching for Media and Streaming Processing," *Proceedings of International Conference on Multimedia and Expo*, pp. 101-104, 2002.

[2] S. Carr, K. S. Mckinley and C. W. Tseng, "Compiler Optimization for Improving Data Locality," *Proceedings of 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 252-262, 1994.

[3] M. E. Wolf and M. S. Lam, "A Data Locality Optimizing Algorithm," *Proceedings of SIGPLAN'91 Conference on Programming Language Design and Implementation*, pp. 30-44, 1991.

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

[5] F. Harmsze, A. Timmer and J. Meerbergen, "Memory Arbitration and Cache Management in Stream-Based Systems," Proceedings of the DATE 2000, pp. 257-262, 2000.

[6] C. K. Luk, Optimizing the Cache Performance of Non_Numeric Applications, Ph.D. Thesis, University of Toronto, 2000.

[7] S. P. VanderWiel and D.J. Lilja, "When Caches Aren't Enough: Data Prefetching Techniques," *IEEE Computer*, Vol. 30, No. 7, pp. 23-30, 1997.

[8] K. Diefendorff and P. K. Dubey, "How Multimedia Workloads Will Change Processor Design," *IEEE Computer*, Vol. 30, No. 9, pp.43-45, 1997.

[9] T. F. Chen and J. L. Baer, "Effective Hardware-Based Data Prefetching for High Performance Processors," *IEEE Transations on Computers*, Vol. 44, No.5, pp. 609-623, 1995.

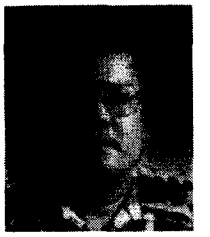
[10] T. Horel and G. Lauterbach, "UltraSPAC-III: Designing Third-Generation 64-bit Performance," *IEEE Micro*, Vol. 19, No. 3, pp. 73-85, 1999.

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

[12] D. F. Zucker, M. J. Flynn and R. B. Lee, "A Comparison of Hardware Prefetching Tech-

niques for Multimedia Benchmarks," *Proceedings of International Conference on Multimedia Computing and Systems*, pp. 236-244, 1996.

[13] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. "MediaBench: A Tool for Evaluating and Synthesizing Multimedia Communications Systems," *Proceedings of the 30th Annual international Symposium on Microarchitecture*, pp. 330-335, 1997.



김 석 주

1982년 서울대학교 전자계산기 공학과

1984년 한국 과학 기술원 전산학과(공학석사)

1997년~현재 혜천대학 컴퓨터·멀티미디어 계열 조교수

관심분야 : 멀티미디어 응용