스트리밍 데이터의 선인출에 사용되는 참조예측표의 스칼라 우선 교체 전략

임 철 후[†] · 전 영 숙[†] · 김 석 일[††] · 전 중 남[†††]

요 약

멀티미디어 응용프로그램의 데이터는 주소 간격이 일정한 스트리밍 패턴으로 참조되는 특성이 있다. 이러한 특성을 선인출방법에 적용하여 멀티미디어 응용프로그램의 수행속도를 향상 시킬 수 있다. 참조예측표에 의한 선인출방법은 메모리 참조명령어의 과거 기록을 이용하여 규칙적으로 참조되는 메모리주소를 예측한다. 이 논문은 참조예측표를 사용하는 하드웨어 기반의 규칙 선인출방법에서 효율적인 참조예측표 운영방법을 제안한다. 참조예측표에 입력되는 메모리 참조명령어는 스칼라데이터 참조명령어와 스트리밍데이터 참조명령어로 구성된다. 스칼라데이터 참조명령어는 선인출에 사용되지 않으므로 스칼라데이터 참조명령어를 우선적으로 교체함으로서, 참조예측표를 효과적으로 사용할 수 있다. 이 방법은 기존 FIFO 방법과 비교할 때, 선인출에 사용되는 스트리밍데이터 참조명령어를 참조예측표에 더 오래 유지함으로써, 선인출 성능이 향상된다.

Scalar First Replacement Strategy for Reference Prediction Table Used in Prefetching Streaming Data

Chulhoo Lim[†] · YoungSuk Chon[†] · Sukil Kim[††] · Joongnam Jeon[†††]

ABSTRACT

Multimedia applications tend to access their data as a streaming pattern with regular intervals. This characteristic can be utilized in prefetching the multimedia data into cache memory so as to reduce their execution speeds. The reference-prediction prefetch algorithm predicts the memory address that seems to be used in the next time based on the previous history of memory references stored in the prediction reference table. This paper proposes a strategy to manipulate the reference prediction table which contains all of the data reference instructions to scalar and streaming data. We have recognized that the scalar reference instructions do not contribute to the data prefetching algorithm. Therefore, when replacing an element in the reference prediction table, the proposed algorithm preferentially selects the scalar reference instruction before the stream reference instruction. It makes the stream reference instruction to stay for a long time compared to the FIFO replacement policy, and eventually improves the performance of data prefetching.

키워드 : 캐시(Cache), 데이터 선인출(Data Prefetch), 참조예측표(Reference Prediction Table), 메모리 참조명령어(Memory Reference Instruction)

1. 서 론

메모리 참조는 컴퓨터의 실행 속도를 느리게 하는 주요 인이다. 캐시 메모리는 메모리 참조의 지역성(locality of memory reference)을 활용하여 메모리 참조 속도를 줄이는 컴퓨터의 구성요소이다. 최근에는 메모리 참조 시간을 감추기 위해 미래에 사용될 것으로 예측되는 데이터를 미리 캐시 메모리에 옮겨 놓는 캐시 선인출(cache prefetching) 방법이 연구되고 있다. 메모리 참조명령어는 단일 변수(스칼라데이터)를 참조하는 것과 반복문 안에서 배열의 다른 원소(스트리밍데이터)를 참조하는 것으로 구분될 수 있다. 이것들 중에서 선인출의 대상은 스트리밍데이터이다.

멀티미디어 응용프로그램은 많은 양의 데이터를 참조하며, 데이터 재사용성(reusability)이 적은 특징이 있다. Fritts [1]의 연구결과에 따르면, 멀티미디어 응용프로그램은 전체 수행시간 중 25~50%가 메모리 참조에 소요된다. 멀티미디어 데이터는 배열의 형태로 표현되며, 반복문 안에서 일정한 간격으로 참조되는 경향이 많다[3, 4]. 이러한 규칙성을

※ 본 연구는 한국과학재단 목적기초연구(R05-2002-000-01470-0) 지원으로 수행되었음.
† 준 회 원 : 충북대학교 대학원 컴퓨터과학과
†† 종신회원 : 충북대학교 컴퓨터과학과 교수
††† 종신회원 : 충북대학교 전기전자컴퓨터공학부 교수
논문접수 : 2004년 1월 31일, 심사완료 : 2004년 5월 27일

활용하면 선인출 효과를 높일 수 있다[2].

여러 가지 선인출방법 중 하나인 규칙 선인출방법은 스칼라데이터와 스트리밍데이터를 참조하는 모든 메모리 참조명령어들에 대한 과거 기록을 참조예측표(RPT : Reference Prediction Table)에 저장한다. 반복문 안에서 동일한 명령어가 스트리밍데이터를 참조하는 경우에 참조예측표에 저장되어 있는 정보를 이용하여 미래에 사용될 메모리 주소를 예측하여 선인출 명령을 발생시킨다. 이 참조예측표는 FIFO(First-In First-Out) 방법에 의하여 운영된다. 즉, 참조예측표가 모두 채워진 경우에 새로운 메모리 참조명령어가 발생하면 가장 오래된 메모리 참조명령어와 교체된다. 이 과정에서, 선인출의 대상이 아닌 스칼라데이터 참조명령어가 참조예측표 안에 계속 존재하고 스트리밍데이터 참조명령어가 교체되어 선인출 효과를 떨어뜨릴 가능성이 발생한다.

본 논문에서는 참조예측표 안에 스트리밍데이터 참조명령어를 최대한 오래 유지시켜 예측 효율을 높이고 참조예측표의 크기를 줄이는 방법을 제안한다. 참조예측표에 입력된 메모리 참조명령어는 프로그램이 수행되면서 스트리밍데이터 참조명령어와 스칼라데이터 참조명령어로 구분된다. 이때 스칼라데이터 참조명령어를 참조예측표의 한쪽 끝으로 이동시킨다. 참조예측표가 모두 채워져서 새로운 메모리 참조명령어가 입력되면 FIFO 방법에 의해 예측에 쓰이지 않는 스칼라데이터 참조명령어가 우선적으로 제거된다. 이 방법은 기존의 FIFO 방법보다 참조예측표의 사용 효율을 높일 수 있을 것으로 기대된다.

본 논문의 구성은 다음과 같다. 2장에서는 캐시 선인출방법에 대해서 알아보고 3장에서는 규칙 선인출방법에서 사용되는 참조예측표의 기존 운영방법을 설명하고 문제점을 제시한다. 4장에서는 참조예측표의 기존 운영방법의 문제를 해결하기 위한 운영방법을 제안한다. 5장에서는 참조예측표의 기존 운영방법과 본 논문에서 제안된 방법을 벤치마크에 적용하여 실험한 결과를 제시하고, 마지막으로 6장에서는 본 논문의 결론과 앞으로의 연구 방향을 논한다.

2. 캐시 선인출방법

소프트웨어에 의한 선인출방법[9, 15]은 정적선인출이라고도 하며 프로그래머나 컴파일러가 프로그램 구조를 분석하여 실행 시간 이전에 선인출 명령어(prefetch instruction)를 프로그램에 삽입하여 실행코드를 만드는 방법이다. 이 방법은 컴파일 정보를 활용하여 데이터의 참조 패턴과 참조시점을 정확하게 예측할 수 있지만, 실행 유니트가 선인출 명령어를 수행하는 만큼 실행 사이클이 증가하는 문제점이 있다. 최근의 프로세서 구조들은 대부분 데이터 선인출을 구현하기 위한 방안으로 명령어 세트에 메모리 선인출 명령어를 포함하는 추세이다[13, 14]. 이와 비교하여 하드웨어 선인출방법[5, 11]은 동적 선인출이라고도 하며 선인출 하드웨어에 의해 메모리 참조명령어의 수행을 관찰하고, 이를 바탕으로 선인출할 메모리 주소를 계산한 후 실행 시간에 선인출 명령(prefetch command)을 발생시킨다. 이 과정은 실행 유니트와 독립적으로 수행되며, 복잡한 패턴의 데이터 참조는 예측할 수 없으나 단순한 참조 패턴에 대한 선인출에서는 우수한 성능을 발휘한다. 또한 실행 사이클의 증가를 유발하지 않아 스트리밍 데이터와 같이 단순한 규칙성을 갖는 데이터를 처리하는 프로그램에 적용하기 적합하다.

하드웨어 기반의 선인출방법에는 다중블록 선인출방법(multi-block prefetching)[12], 연속 선인출방법(stream buffer)[10, 11], 규칙 선인출방법(reference prediction prefetching)[5], 상관 선인출방법(Correlation Prefetching)[16, 17] 등이 있다.

다중블록 선인출방법은 어떤 메모리 블록의 참조에 대하여 캐시 미스가 발생했을 때, 요청된 메모리 블록을 캐시로 인출한다. 그리고 캐시 미스된 블록과 메모리 상에 연속적으로 배치되어 있는 블록을 정해진 수만큼 캐시로 선인출하는 방법이다. 이 방법은 메모리 블록의 실제 참조시점보다 너무 이르게 선인출 함으로서 선인출 시점과 참조시점 사이에 캐시에서 재사용될 수 있는 다른 메모리 블록들을 미리 교체하여 불필요한 캐시 미스가 발생하는 현상이 일어난다.

연속 선인출방법은 선인출 블록들을 캐시로 적재하기 전에 메모리와 캐시 사이에 위치한 FIFO 큐 형태의 스트림 버퍼에 저장하여 다중블록 선인출방법의 문제를 해결한다. 그러나 연속 선인출방법도 연속적인 메모리 블록을 차례로 참조하는 경우를 제외하면 메모리 블록을 스트림 버퍼로 적재하고 이를 제거하는 불필요한 작업을 반복해야 하는 문제점을 갖고 있다.

규칙 선인출방법은 메모리 참조에 대한 과거 기록으로부터 메모리 참조의 규칙성을 찾아 이를 바탕으로 미래에 참조될 메모리 주소를 계산하고, 이에 대한 선인출 명령을 발생하는 방법으로 이것을 구현하기 위해 참조예측표를 사용한다.

상관 선인출방법은 메모리 참조명령어가 참조하는 메모리의 패턴을 찾는다. 규칙 선인출방법은 메모리 참조명령어가 메모리 주소를 일정한 간격으로 참조할 때만 예측이 가능해진다. 그러나 상관 선인출방법은 메모리 참조명령어가 참조하는 메모리의 주소간격이 불규칙하더라도 참조되는 메모리가 일정한 패턴을 갖게 되면 예측이 가능해진다. 상

관 선인출방법 중 하나로 Markov 방법[16]이 있다. 이 방법은 Markov 확률 모델을 이용하여 주어진 메모리 참조명령어가 참조하는 메모리의 패턴을 결정한다.

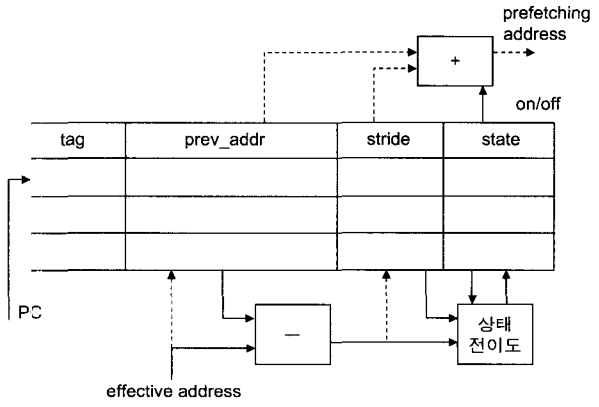
3. 규칙 선인출방법

규칙 선인출방법은 메모리를 참조하는 명령어의 이전 참조 주소에 기반하여 다음에 참조할 메모리 주소를 예측한다. 이러한 예측은 반복문에서 일어나는데 i번째 반복이 수행될 때, 다음 반복, 즉 $(i+1)$번째 반복에서 참조할 메모리 주소를 예측하여 선인출을 수행한다. 프로그램 카운터가 지시하는 메모리 참조명령어를 디코드 할 때, 참조예측표에 이 명령어와 일치하는 행이 있는지 검사하여, 없으면 이 명령어를 참조예측표에 추가하고, 만약 일치하는 행이 있고 다음 반복에 사용될 것이 예측된다면 선인출 명령을 발생시킨다.

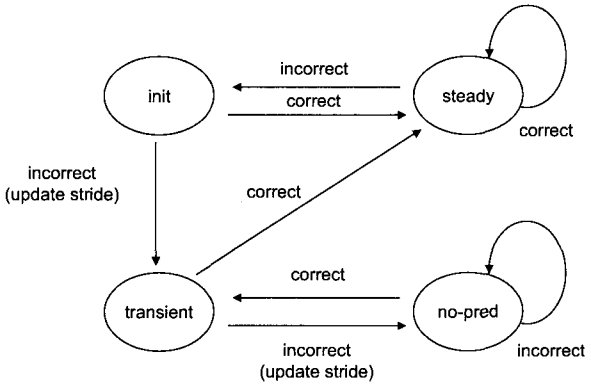
3.1 참조예측표 구조

참조예측표는 메모리 참조명령어들이 참조하는 메모리 주소의 규칙성을 찾기 위해 메모리 참조명령어들에 대한 주소간격의 상호관계와 이전에 참조된 메모리 주소의 정보를 유지하여야 한다. 이를 위해 참조예측표의 각 행은 명령어 주소(tag)에 의해 인덱스 되며, 참조한 메모리의 유효주소(prev_addr)를 저장하고, 저장된 유효주소와 현재 참조한 주소에 대한 주소간격(stride), 그리고 현재 계산된 주소간격과 이전에 있던 주소간격을 비교하여 상태(state)를 변화시켜야 한다.

(그림 1)은 참조예측표 구조를 나타낸 것이다. tag 필드는 메모리 참조명령어의 주소를 나타내기 위해 프로그램 카운터의 값을 이용하고, prev_addr 필드는 tag에 있는 메모리 참조명령어가 가장 마지막에 참조한 메모리 주소를 나타낸다. stride 필드는 마지막에 참조된 두 주소의 간격을



(그림 1) 참조예측표의 구조

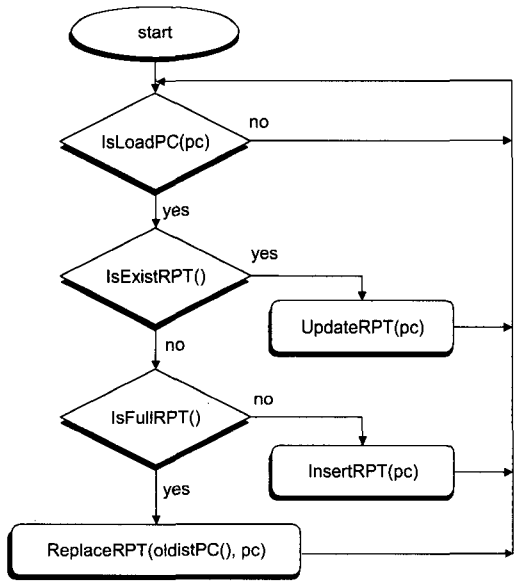


(그림 2) 참조예측표의 상태 전이도

나타낸다. state 필드는 지난 메모리 참조 패턴에 따라 (그림 2)의 상태 전이도에서 보여지는 4개의 상태를 나타내며 이 상태에 의해 선인출의 유무를 결정한다.

3.2 참조예측표 운영

기존의 참조예측표 운영방법은 새로운 메모리 참조가 발생하면 FIFO 방법으로 참조예측표 안에서 입력된지 가장 오래된 행을 제거하고, 새로운 메모리 참조명령어에 대한 정보를 참조예측표에 넣는다. (그림 3)에는 참조예측표에 대한 기존 운영 알고리즘이다.



(그림 3) 기존 참조예측표의 운영 알고리즘

참조예측표에 들어가는 메모리 참조명령어는 스칼라데이터 참조명령어와 스트리밍데이터 참조명령어로 구분된다. 스트리밍데이터 참조명령어가 프로그램에서 반복적으로 사용될 때, 이 명령어의 정보는 참조예측표에 저장되어 앞으로 참조할 메모리 주소를 예측하여 데이터를 선인출한다.

반면에 스칼라데이터 참조명령어는 선인출에 사용되지 않는다.

(그림 4)는 두개의 어떤 루프에서 사용되는 메모리 참조명령어들을 나열하였다. M_i는 스트리밍데이터 참조명령어이고 S_i는 스칼라데이터 참조명령어이다. 스트리밍데이터는 ((그림 4)(a))에서처럼 주소간격이 모두 2이고 Sequence 1과 Sequence 2는 프로그램 상에서 ((그림 4)(b))와 같이 사용된다고 가정한다. 모든 메모리 참조명령어가 참조하는 데이터는 캐시에 없고 참조예측표의 크기가 9개라고 가정할 때, 기존 참조예측표 방법으로 ((그림 4)(b))를 실행해 보자.

```
Sequence
  • Sequence 1 : M0, M1, S0, M2, S1, S2
  • Sequence 2 : M3, S3, M4, S4, S5, M5
Reference Address Sequence
  • M0 : 1000, 1002, 1004, 1008, 1010 … (pc : 100)
  • M1 : 2000, 2002, 2004, 2008, 2010 … (pc : 104)
  • M2 : 3000, 3002, 3004, 3008, 3010 … (pc : 112)
  • M3 : 4000, 4002, 4004, 4008, 4010 … (pc : 124)
  • M4 : 5000, 5002, 5004, 5008, 5010 … (pc : 132)
  • M5 : 6000, 6002, 6004, 6008, 6010 … (pc : 144)
```

(a) Loop Sequence

```
for i = 1 to 100 ..... 반복문 A
   for j = 1 to 10 ..... 반복문 B
      Sequence 1 ;
   end for
   for k = 1 to 10 ..... 반복문 C
      Sequence 2 ;
   end for
end for
```

(b) Program Sequence

(그림 4) 예제 코드

참조예측표의 기존 운영방법으로 ((그림 4)(b))를 수행했을 때 참조예측표의 변화는 (그림 5)와 같다. 참조예측표의 기존 운영방법에 따르면 반복문 B의 Sequence 1이 처음 실행되면 메모리 참조명령어 M_0, M_1, S_0, M_2, S_1, S_2가 순서대로 참조예측표에 입력된다((그림 5)(a)). M_0, M_1, M_2는 스트리밍데이터 참조명령어이기 때문에 참조예측표의 상태전이도에 따라 세 번째 반복 후부터 예측이 가능해지므로 ((그림 5)(b)), 4번째 반복부터는 캐시 미스를 유발하지 않는다. 반복문 C의 Sequence 2가 실행될 때 M_3, S_3, M_4, S_4, S_5, M_5가 순서대로 참조예측표에 입력된다. 참조예측표의 크기가 9개로 가정하였기 때문에, 이미 참조예측표에 들어있던 Sequence 1의 메모리 참조명령어 중 앞의 세 개 즉, M_0, M_1, S_0가 참조예측표에서 제거된다((그림 5)(c)). M_3, M_4, M_5는 스트리밍데이터 참조명령어이기 때문에 상태전이도에 따라 세 번째 반복에서 예측이 가능해진다. 반복문 A에 의해 다시 반복문 B가 실행될 때 참조예측표에는 M_0, M_1, S_0가 없기 때문에 M_0, M_1, S_0을 차례로 입력받으면서 참조예측표에 가장먼저 들어와 있던 M_2, S_1, S_2를 제거한다. Sequence 1에 의해 M_2, S_1, S_2가 필요하게 되었을 때 참조예측표는 입력된지 가장 오래된 M_3, S_3, M_4를 제거하고 M_2, S_1, S_2을 입력받는다. 결과적으로 Sequence 1은 참조예측표에서 모두 제거되었다가 다시 입력된다((그림 5)(d)). 따라서, 처음 반복에서와 같이 세 번째 반복 이후부터 예측이 가능해진다.

tag	prev_addr	stride	state
100	1000	0	init
104	2000	0	init
108	S	0	init
112	3000	0	init
116	S	0	init
120	S	0	init

(a)

tag	prev_addr	stride	state
100	1004	2	steady
104	2004	2	steady
108	S	0	steady
112	3004	2	steady
116	S	0	steady
120	S	0	steady

(b)

tag	prev_addr	stride	state
112	3018	2	steady
116	S	0	steady
120	S	0	steady
124	4000	0	init
128	S	0	init
132	5000	0	init
136	S	0	init
140	S	0	init
144	6000	0	init

(c)

tag	prev_addr	stride	state
136	S	0	steady
140	S	0	steady
144	6018	2	steady
100	1020	0	init
104	2020	0	init
108	S	0	init
112	3020	0	init
116	S	0	init
120	S	0	init

(d)

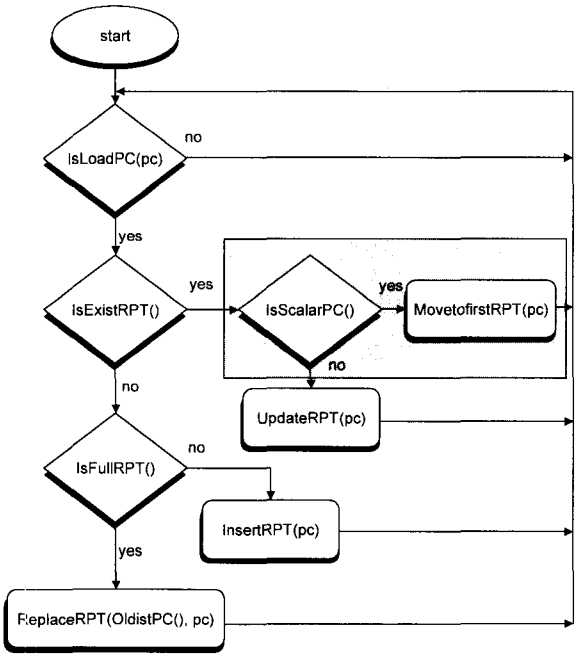
(그림 5) 참조예측표의 크기가 9인 규칙 선인출방법에서 참조예측표의 변화 (a) 반복문 B의 Sequence 1이 처음 실행된 후 (b) 반복문 B가 세 번째 실행된 후 (c) 반복문 C의 Sequence 2가 처음 실행된 후 (d) 반복문 A에 의해 반복문 B가 다시 실행된 후

이 예에서와 같이, FIFO 방법의 참조예측표 운영은 예측에 사용하지 않는 스칼라데이터와 예측에 사용되는 스트리밍데이터를 구분하지 않기 때문에, 매 반복마다 스트리밍데이터를 참조예측표에 다시 적재함으로써 선인출 효율이 떨어지는 현상이 발생한다. 이러한 문제점을 해결하는 가장 간단한 방법은 참조예측표의 크기를 이 프로그램에서 사용하는 메모리 참조명령어의 수만큼 늘리는 것이다. 그러나 참조예측표는 캐시 메모리와 같은 하드웨어로 구성되므로 가격이 비싸다. 따라서 참조예측표의 크기를 늘리지 않으면서

참조예측표의 예측 가능성을 최대화할 수 있는 새로운 운영방법이 제안되어야 한다.

4. 스칼라 우선 참조예측표 교체 알고리즘

참조예측표의 크기는 이상적으로 반복문 안에 포함되어 있는 메모리 참조명령어의 수와 같거나 커야 한다. 그러나 이 수는 응용프로그램의 종류마다 다르기 때문에 제한된 참조예측표를 효과적으로 운영하여 스트리밍데이터 참조에 대한 예측 가능성을 높여야 한다. 참조예측표의 기존 운영방법의 문제점은 FIFO 방법으로 운영하는 것이다. 즉, 메모리 참조명령어를 수행한 순서대로 참조예측표에 입력하고 다른 메모리 참조명령어가 입력될 때 참조예측표가 가득 찼으면, 입력된지 가장 오래된 순서대로 교체하여 예측가능성이 있는 스트리밍데이터 참조명령어가 예측가능성이 없는 스칼라데이터 참조명령어에 앞서 제거될 가능성이 높다는 것이다.



(그림 6) 제안하는 참조예측표의 운영 알고리즘

이 문제점을 해결하기 위해, 본 논문에서 새로운 참조예측표의 운영 알고리즘(그림 6)을 제안한다. (그림 6)에서 IsExistRPT() 함수에 의해 입력된 메모리 참조명령어가 참조예측표에 있을 경우, 입력된 메모리 참조명령어는 참조예측표에 있는 이전 정보와 현재 입력된 정보를 이용하여 참조예측표의 자신의 명령 행에 대한 정보를 갱신한다. 이때 IsScalarPC() 함수에 의해 갱신된 정보를 갖는 명령어가 스칼라데이터를 참조하면 이 메모리 참조명령어를 참조예측표의 가장 앞으로 이동시킨다. 이렇게 간단한 기능을 참조예측표의 기존 운영방법에 추가함으로서 기존 운영방법보다 예측가능성을 증가시킬 수 있다.

tag	prev_	stride	state
100	1000	0	init
104	2000	0	init
108	S	0	init
112	3000	0	init
116	S	0	init
120	S	0	init

(a)

tag	prev_	stride	state
120	S	0	steady
116	S	0	steady
108	S	0	steady
100	1002	2	trans
104	2002	2	trans
112	2002	2	trans

(b)

tag	prev_	stride	state
120	S	0	steady
116	S	0	steady
108	S	0	steady
100	1002	2	steady
104	2002	2	steady
112	2002	2	steady

(c)

tag	prev_	stride	state
100	1018	2	steady
104	2018	2	steady
112	3018	2	steady
124	4000	0	init
128	S	0	init
132	5000	0	init
136	S	0	init
140	S	0	init
144	6000	0	init

(d)

tag	prev_	stride	state
128	S	0	steady
136	S	0	steady
140	S	0	steady
100	1018	2	steady
104	2018	2	steady
112	3018	2	steady
124	4002	2	trans
132	5002	2	trans
144	6002	2	trans

(e)

tag	prev_	stride	state
100	1020	2	steady
104	2020	2	steady
112	3020	2	steady
124	4018	2	steady
132	5018	2	steady
144	6018	2	steady
108	S	0	init
116	S	0	init
120	S	0	init

(f)

(그림 7) 참조예측표의 크기가 9인 규칙 선인출방법에서 참조예측표의 변화 (a) 반복문 B의 Sequence 1이 처음 실행된 후 (b) 반복문 B가 두 번째 실행된 후 (c) 반복문 B가 세 번째 실행된 후 (d) 반복문 C의 Sequence 2가 처음 실행된 후 (e) 반복문 C가 두 번째 실행된 후 (f) 반복문 A에 의해 반복문 B가 다시 실행된 후

((그림 4)(b))프로그램을 크기가 9인 참조예측표를 이용하여 제안된 운영방법으로 수행하였을 경우, 예측에 사용되는 스트리밍데이터 참조명령어가 ((그림 7)(f))와 같이 참조예측표에 지속적으로 존재한다. 그러므로 반복문 A에 의해 반복문 B, C가 두 번째 반복될 때부터 스트리밍데이터 참조

명령어에 대하여 계속 데이터 선인출 명령을 발생할 수 있게 된다.

5. 실험 결과 및 분석

멀티미디어 응용프로그램에 대한 미디어 데이터의 검출에 따른 선인출방법의 성능을 분석하기 위해, Digital Alpha DEC 시스템에서 트레이스 구동 시뮬레이션을 통하여, 각 벤치마크에 따라 캐시의 수행을 모의 실험한 결과로 캐시 미스 수를 제시하고 이를 분석하였다. 응용프로그램의 명령어 트레이스를 생성하기 위하여, Alpha CPU용 목적 코드를 이용하여 벤치마크 프로그램을 분석할 수 있도록 개발한 ATOM[6] 시뮬레이터를 사용하였다. 이 실험에서는 데이터 적재/저장 명령어만을 트레이스하며, 이 명령어의 프로그램 카운터와 유효주소 값을 기록하였다. 캐시의 수행과정을 모의하고 이에 관한 분석결과를 생성하기 위하여 트레이스 구동방법의 시뮬레이터인 dinero Ⅲ[7]를 바탕으로 캐시 시뮬레이터를 구현하였다. 이 캐시 시뮬레이터는 ATOM형 실행코드로부터 생성된 메모리 참조명령어 트레이스를 입력으로 사용하여 적재/저장 명령어의 메모리 참조회수와 캐시 미스율 등의 결과를 산출하도록 개발한 시뮬레이터이다.

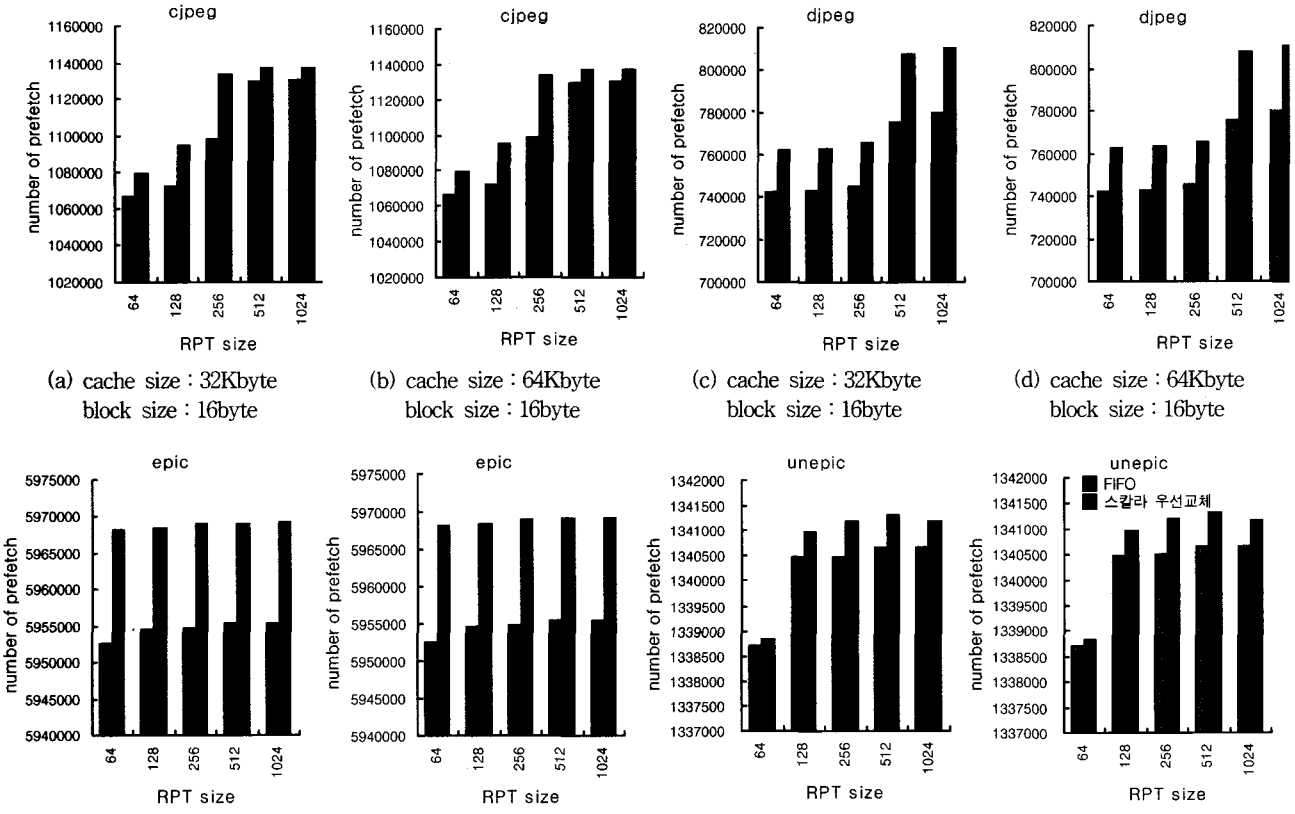
〈표 1〉 미디어 벤치마크의 특징

program module	description	data source	input size	image/ frame size
cjpeg	흑백 및 칼라 이미지를 위한 표준 압축 방법	image files of .ppm format	100K bytes	172×189
djpeg	cjpeg에 의해 압축된 이미지의 해제	image file of .jpg format		
epic	실험용 이미지 압축 도구	image files of .raw format	60K bytes	256×256
unepic	epic에 의해 압축된 이미지의 해제	image files of .E format		

〈표 2〉 실험에 사용된 메모리 구조

Memory Architecture	
data cache size	32k~64k
block size	16byte
associativity	4-way
Reference Prediction Table size	64, 128, 256, 512, 1024 entry
Cache replacement policy	LRU

벤치마크 프로그램으로는 멀티미디어용 벤치마크로 개발된 MediaBench[8] 중 네 개의 프로그램을 선정하여 실험하



(a) cache size : 32Kbyte block size : 16byte
(b) cache size : 64Kbyte block size : 16byte
(c) cache size : 32Kbyte block size : 16byte
(d) cache size : 64Kbyte block size : 16byte
(e) cache size : 32Kbyte block size : 16byte
(f) cache size : 64Kbyte block size : 16byte
(g) cache size : 32Kbyte block size : 16byte
(h) cache size : 64Kbyte block size : 16byte

(그림 8) 참조예측표에 의해 발생된 선인출 명령의 수(nP)

였다. <표 1>에 실험에 사용된 벤치마크 프로그램의 특징과 각 프로그램의 수행을 위하여 사용된 입력파일의 특징을 요약하였다. 본 실험은 캐시구조와 참조예측표의 크기를 다양하게 변화시키면서 FIFO 방법과 스칼라 우선교체 방법에 대해 선인출 명령 발생 수와 캐시 미스 수를 측정하여 그 결과를 제시한다. <표 2>는 실험에 사용할 메모리 구조를 나타낸다.

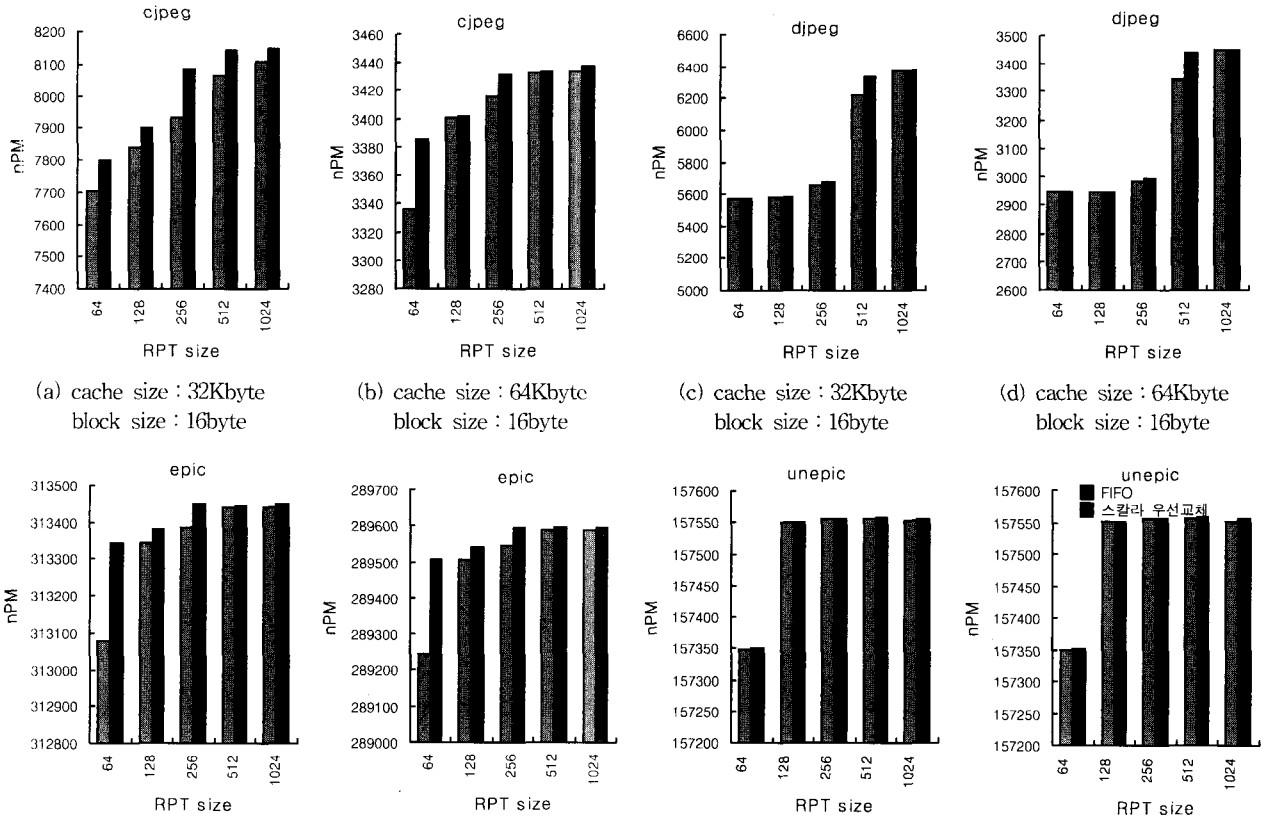
제안하는 스칼라 우선교체 방법과 기존의 FIFO 방법의 선인출 효과를 비교 평가하기 위하여 다음과 같은 평가 항목을 선정하였다.

- 선인출 명령 수(nP : number of Prefetch) : 선인출 알고리즘에 의하여 발생한 선인출 명령의 수
- 선인출 명령 히트 수(nPH : number of Prefetch Hit) : 선인출 알고리즘에 의하여 발생한 선인출 명령의 대상이 이미 캐시 안에 존재하여 선인출 동작이 필요 없는 선인출 명령의 수
- 선인출 명령 미스 수(nPM : number of Prefetch Miss) : 선인출 알고리즘에 의하여 발생한 선인출 명령의 대상이 캐시 안에 존재하지 않아 실제로 선인출 동작을 수행하는 선인출 명령의 수
- 캐시 미스 수(nDM : number of Demand Miss) : 프로그램이 실행하면서 필요로 하는 데이터가 캐시에 없어서 주기억장치로부터 데이터를 가져와야 하는 경우의 수.

참조예측표를 FIFO 방법으로 운영하였을 때보다 스칼라 우선교체 방법으로 운영하였을 때 cjpeg과 djpeg에서의 선인출 명령 발생 빈도는 0.08%~4.14% 증가하였다. 스칼라 우선교체 방법은 참조예측표 내의 스트리밍데이터 참조명령어가 FIFO 방법보다 더 오랫동안 존재하도록 하여 선인출 명령 발생빈도를 증가시켰다는 사실이 입증되었다.

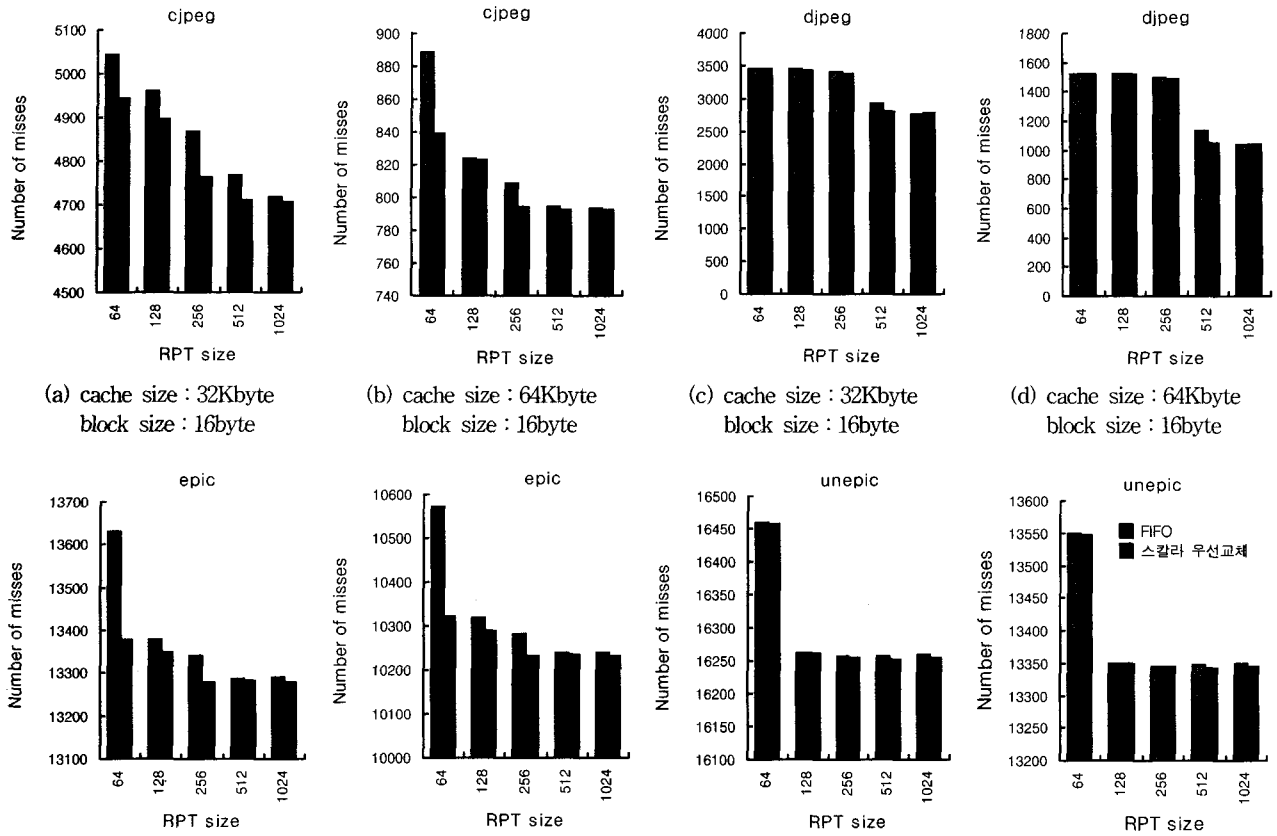
선인출 명령이 발생되면 선인출 대상이 캐시에 있는지 검사하여, 있으면 선인출 동작을 하지 않고, 없으면 선인출 동작을 수행하여 주기억장치에 있는 선인출 대상을 캐시로 가져온다. (그림 9)는 (그림 8)에서 발생된 선인출 명령의 수중에서 실제로 선인출 동작을 수행한 선인출 명령의 수이다.

스칼라 우선교체 방법으로 발생된 선인출 명령 중 실제로 선인출 동작을 수행한 선인출 명령의 수는 FIFO 방법보다 0.03%~2.87% 증가했다. 참조예측표를 FIFO 방법으



(a) cache size : 32Kbyte block size : 16byte
(b) cache size : 64Kbyte block size : 16byte
(c) cache size : 32Kbyte block size : 16byte
(d) cache size : 64Kbyte block size : 16byte
(e) cache size : 32Kbyte block size : 16byte
(f) cache size : 64Kbyte block size : 16byte
(g) cache size : 32Kbyte block size : 16byte
(h) cache size : 64Kbyte block size : 16byte

(그림 9) 참조예측표에 의해 실제 선인출 된 수(nPM)



(a) cache size : 32Kbyte block size : 16byte
(b) cache size : 64Kbyte block size : 16byte
(c) cache size : 32Kbyte block size : 16byte
(d) cache size : 64Kbyte block size : 16byte
(e) cache size : 32Kbyte block size : 16byte
(f) cache size : 64Kbyte block size : 16byte
(g) cache size : 32Kbyte block size : 16byte
(h) cache size : 64Kbyte block size : 16byte

(그림 10) 참조예측표 운영방법의 캐시 미스 수(nDM)

로 운영했을 때보다 스트리밍데이터 참조명령어에 대한 선인출 명령을 더 많이 발생시킴으로서 예측 가능성을 높인 것이다. 그 결과 전체 캐시 미스 수는 0.03%~5.62% 감소되었다(그림 10).

(그림 10)은 참조예측표를 FIFO 방법과 스칼라 우선교체 방법을 이용하여 운영했을 때 발생된 캐시 미스수이다. djpeg과 unepic에서 유사한 캐시 미스수를 나타내지만 cjpeg과 epic에서는 FIFO 방법보다 스칼라 우선교체 방법이 더 적은 미스 수를 나타내고 있다. 제안된 운영방법은 참조예측표의 크기가 작을수록 캐시 미스 수를 더 많이 감소시킨다. FIFO 방법의 경우 참조예측표의 크기가 1024 엔트리를 가질 때 미스수가 최적화 되지만 스칼라 우선교체 방법은 참조예측표의 크기가 512 엔트리를 가질 때 캐시 미스 수는 더 이상 증가되지 않고 최적화 된다.

6. 결 론

참조예측표를 사용하여 선인출하는 메모리 구조에서, 이상적인 참조예측표의 크기는 응용프로그램의 반복문에 있는 메모리 참조명령어를 포함할 수 있어야 한다. 참조예측표를 이용하여 선인출을 할 경우 메모리 참조명령어 중 필요한 것은 스트리밍데이터 참조명령어이다. 즉, 스칼라데이터 참조명령어는 참조예측표에 오랫동안 있을 필요가 없다. 또한 참조예측표는 크기가 클수록 반복문 내에 존재하는 메모리 참조명령어를 최대한 오래 유지시킬 수 있어 스트리밍데이터에 대하여 가능한 많이 예측할 수 있다. 참조예측표는조명령어가 예측에 사용되지 않는 메모리 참조명령어에 앞서 제거되어 예측가능성이 낮아질 수 있었다. 본 논문에서 제안한 알고리즘은 참조예측표의 기존 운영방법에 스칼라데이터 참조명령어를 참조예측표의 가장 뒤쪽으로 이동시키는 알고리즘을 추가한다. 이렇게 함으로서 새로운 메모리 참조명령어가 입력될 때 예측에 사용되지 않는 메모리 참조명령어를 우선적으로 제거하여 예측 가능성을 최대화하였다. 참조예측표를 이용한 선인출방법이 하드웨어로 이루어진 동적 선인출방법인 것을 고려해 볼 때, 제안된 운영 방법은 참조예측표의 크기가 작을수록 더 좋은 효과를 볼 수 있다. 또한 FIFO 방법은 캐시 미스 수를 1024 크기에서 최적화하지만 스칼라 우선교체 방법의 경우 그 절반

인 512엔트리에서 최적화 된다.

참조예측표와 같은 선인출 알고리즘에 의해 선인출 명령이 대량 발생할 경우 캐시 오염이라는 문제가 발생한다. 캐시 미스 오염은 단일 캐시 내에 스트리밍데이터와 같은 재사용성이 떨어지는 데이터가 존재하여 캐시의 활용률을 떨어트리는 것을 말한다. 이 문제를 해결하기 위해 적절한 시기를 찾아 데이터를 선인출하고, 단일 캐시 내에서 스칼라 데이터와 스트리밍데이터를 구분하여 캐시의 활용율을 높일 수 있는 새로운 캐시 운영방법에 대한 연구가 필요하다.

참 고 문 헌

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

[2] S. P. VanderWiel, D. J. Lilja, "When Caches Aren't Enough : Data Prefetching Techniques," *IEEE Computer*, pp.23-20, July, 1997.

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

[4] C. K. Luk, Optimizing the Cache Performance of Non_ Numeric Applications, Ph. D. Thesis, *University of Toronto*, 2000.

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

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

[7] M. D. Hill, Dinero III Cache Simulator, Technical Report, Computer Sciences Department, *University of Wisconsin, Madison*, 1990.

[8] C. Lee, M. Potkonjak and W. H. Mangione-Smith, "Media Bench : A Tool for Evaluating and Synthesizing Multimedia Communications Systems," *Proceedings of the 30th Annual international Symposium on Micro architecture*, December, 1997.

[9] C. H. Chi and K. K. Fang, "Compiler Driven Data Cache Prefetching for High Performance Computer," *Proceedings of the Regional 10th Annual International Conference*, Vol.2, No.Conf. 9, pp.274-278. August, 1994.

[10] S. Palacharla and R. Kessler, "Evaluating Stream Buffers as a Secondary Cache Replacement," *Proc. 21st Int'l Symp. Computer Architecture*, pp.24-33, Apr., 1994.

[11] N. P. Jouppi, "Improving Direct-mapped Cache Performance by the Addition of a Small Fully associative Cache and Prefetch Buffers," *Proceedings of 17th Annual International Symposium on Computer Architecture*, pp.364-373, May, 1990.

[12] A. Smith, "Sequential Program Prefetching in Memory Hierarchies," *IEEE Computer*, Vol.11, No.2, pp.7-21, 1978.

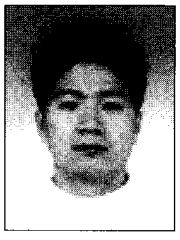
[13] C. Basoglu, W. Lee and J. S. O'Donnell, "The MAP1000A VLIW mediaprocessor," *IEEE Micro*, Vol.20, No.2, pp.48-59, March, 2000.

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

[15] P. Ranganathan, V. S. Pai, H. Abdel-Shafi, S. V. Adve, "The interaction of Software Prefetching with ILP Processors in Shared-Memory Systems," ACM SIGARCH Computer Architecture News, Vol.25, No.2, pp.144-156, May, 1997.

[16] D. Joseph, "Prefetching using Markov Predictors," Computer Architecture, 1997. Conference Proceedings, The 24th Annual International Symposium on, pp.252-263, June, 1997.

[17] Y. Solihin, J. Lee, J. Torrellas, "Correlation prefetching with a user-level memory thread," Parallel and Distributed Systems, IEEE Transactions on, Vol.14, Issue 6, pp.563-580, June, 2003.



임 철 후

e-mail : pedal7@hananet.net

2001년 한밭대학교 컴퓨터과학과(학사)

2001년~현재 충북대학교 컴퓨터과학과 석사과정

관심분야 : 컴퓨터 구조, 병렬처리, 임베디드 시스템 등

전 영 숙

e-mail : yschon@hanmail.net

1996년 한남대학교 전자계산학과(학사)

1998년 한남대학교 컴퓨터공학과(석사)

2002년 충북대학교 컴퓨터과학과 박사수료

관심분야 : 고성능 컴퓨터 구조, 병렬처리

김 석 일

e-mail : ksi@cbucc.chungbuk.ac.kr
1975년 서울대학교 학사학위 취득
1975년~1995년 국방과학연구소 선암 연구원으로 근무
1985년~1989년 미국 North Carolina State University에서 공학박사 취득
1990년~현재 충북대학교 컴퓨터과학과 교수로 재직중
관심분야 : 병렬처리 컴퓨터구조, 슈퍼컴퓨팅, 이기종 분산처리, 시각장애사용자 인터페이스 등

전 중 남

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