공학석사 학위논문

# PCM controller에서 빠른 주소 변환을 위한 효율적인 prefetching 알고리즘

An efficient prefetching algorithm for fast address translation in a PCM controller

2021 년 8 월

서울대학교 대학원

전기 정보 공학부

김 형 석

# PCM controller에서 빠른 주소 변환을 위한 효율적인 prefetching 알고리즘

지도 교수 이 혁 재

이 논문을 공학석사 학위논문으로 제출함
2021 년 8 월

서울대학교 대학원
전기 정보 공학부
김 형 석

김형석의 공학석사 학위논문을 인준함
2021 년 8 월

위 원 장 _____ 김 태 환

부위원장 _____ 이 혁 재

위 원 _____ 김 장 우

# 초 록

DRAM의 스케일링 기술은 한계에 도달했고 이에 따라 DRAM을 대체할 여러 차세대 메모리 기술이 제안되었다. 그 중 상 변화 메모리(phase change memory, PCM)는 물질의 상 변화를 통한 저항 변화로 데이터를 저장하며 이에 따라 효과적인 스케일링이 가능하다. PCM에 접근하기 위해 PCM controller에서 논리적 주소를 물리적 주소로 변환하는 과정을 거친다. 주소 변환 요청 발생 시 DRAM에 접근하여 주소 변환 데이터를 받아온다. 주소 변환을 위한 지속적인 메모리 접근은 전체적인 성능 저하를 야기한다. Prefetcher는 이러한 문제를 효과적으로 해결할 수 있다. PCM workload의 경우 순 방향 순차 패턴 및 역 방향 순차 패턴이 주를 이룬다. 또한 일부 비 순차 패턴의 경우 PCM workload 내에 존재하는 순차 패턴과 연관성을 가지고 있다. 이러한 비 순차 패턴의 경우 뚜렷한 패턴이 존재하지 않아 기존의 prefetch 알고리즘으로 성공적인 prefetch를 진행할 수 없다. 본 연구에서 제안하는 prefetcher는 stream prefetcher와 유사한 방식을 택함으로 순차 패턴을 효율적으로 탐지할 수 있다. 뿐만 아니라 endurance와 depth parameter를 통해 뚜렷한 패턴이 존재하지 않지만 workload 내 순차 패턴과 연관성을 가지는 패턴에 대해 prefetch hit을 발생시킬 수 있다. 제안한 prefetcher의 경우 기존 prefetch 알고리즘 대비 7% 이상의 전체 시스템 성능 향상을 보였다. 또한 prefetch를 진행하지 않았을 경우에 비해 주소 변환 시간을 17% 수준으로 단축시켰으며 전체 시스템 지연시간을 40% 수준으로 단축시켰다.

**주요어:** 상 변화 메모리, 주소 변환, prefetch, 순차 패턴, 비 순차 패턴

**학 번:** 2019-21676

# 목 차

# 표 목차

# 그림 목차

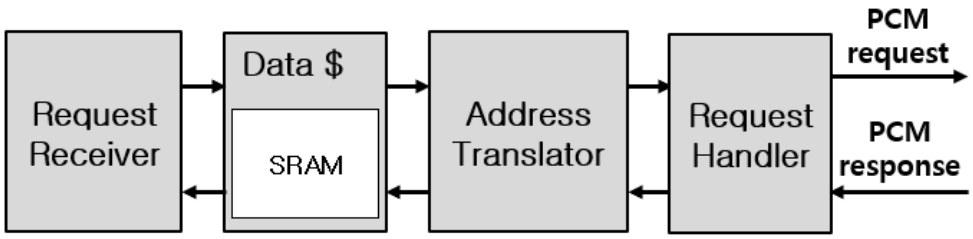
# 제 1 장 서 론

DRAM의 스케일링 기술은 한계에 도달했고 이에 따라 DRAM을 대체할 여러 차세대 메모리 기술이 제안되었다. 그 중 상 변화 메모리(phase change memory, PCM)는 물질의 상 변화를 통한 저항 변화로 데이터를 저장한다. 상 변화 물질은 열을 가할 시 비 결정 상태가 되어 높은 저항을 유지하며 냉각 시 결정 상태가 되어 낮은 저항을 유지한다. 이러한 저항 변화를 통한 데이터 저장 방식은 PCM이 효과적인 스케일링을 할 수 있게 한다. PCM은 Flash 메모리와 마찬가지로 비 휘발성이며 Flash 메모리에 비해 빠른 속도를 가진다. 이에 따라 DRAM과 storage 사이의 간극을 메울 수 있다. PCM은 서버용 메모리, AI, IoT와 같이 다양한 분야에 사용될 전망이다.

## 제 1 절 연구의 배경

PCM으로부터 데이터를 읽고 쓰기 위하여 PCM controller에서 [그림 1-1]과 같이 논리적 주소를 물리적 주소로 변환하는 과정을 거친다. 논리적 주소를 물리적 주소로 변환하기 위한 주소 변환 데이터는 DRAM에 저장되어 있으며 주소 변환 요청 발생 시 DRAM으로부터 데이터를 읽어온다. 주소 변환을 위한 DRAM 접근은 앞선 주소 변환 요청이 마무리되기 전까지 지연된다. 이에 따라 PCM controller의 전체적인 시스템 성능이 저하된다.

PCM controller의 성능 향상을 위해 빠른 주소 변환을 위한 메모리




[그림 1-1] PCM controller 구조

접근 지연 시간 감소는 필수적이다. 메모리 접근 지연을 최소화하기 위해 많은 연구들이 진행되었는데, 대표적으로 cache는 사용한 데이터를 저장함으로 추후 동일한 데이터를 시스템이 요구할 시 메모리에 접근하지 않고도 해당하는 데이터를 얻을 수 있게 한다. 하지만 데이터의 초기 접근 시 메모리 접근은 필수적이며 이에 따라 전체 시스템이 지연된다.

Prefetcher는 이러한 문제를 효과적으로 해결할 수 있다. Prefetcher는 사용 가능성이 높은 데이터를 미리 메모리로부터 읽어와 cache 혹은 prefetch buffer에 저장한다. 이러한 방식은 해당하는 데이터의 초기 접근에도 메모리 접근에 의한 지연을 최소화할 수 있다. 본 연구에서는 주소 변환 데이터를 미리 prefetch함으로 주소 변환 시간을 최소화하고 이에 따라 PCM controller의 성능을 향상시키는 방법을 제안한다.

## 제 2 절 연구의 내용

Prefetcher는 이전에 발생한 주소의 패턴을 통해 추후 발생할 패턴을 예측한다. 즉 성공적인 prefetch를 위해서는 목표로 하는 workload가 어떠한 특징을 가지는지에 대한 분석이 필수적이다. PCM

workload의 경우 순 방향 순차 패턴 및 역 방향 순차 패턴이 주를 이룬다. 또한 일부 비 순차 패턴의 경우 기존의 순차 패턴 및 추후 발생할 순차 패턴과 연관성이 존재한다. 이러한 특징을 가지는 PCM workload를 위해 기존의 prefetch 알고리즘을 적용하기엔 어려움이 따른다. Signature path prefetcher와 variable length prefetcher의 경우 tag에 따른 delta 변화를 관측함으로 순 방향 순차 패턴과 역 방향 순차 패턴에 대해 성공적으로 prefetch를 진행할 수 있다. 하지만 뚜렷한 패턴이 존재하지 않는 비 순차 패턴에 대해서는 성공적으로 prefetch를 진행할 수 없다. Stream prefetcher의 경우 순 방향 순차 패턴에 대해 성공적인 prefetch를 진행할 수 있지만, 역 방향 순차 패턴에 대해 성공적인 prefetch를 진행할 수 없다. 이에 따라 PCM controller는 독자적인 prefetch 알고리즘이 필요하다.

본 연구에서 제안하는 prefetcher의 경우 순 방향 순차 패턴 및 역 방향 순차 패턴을 효과적으로 탐지하여 prefetch를 진행한다. 또한 endurance와 delta parameter를 통해, 비 순차 패턴 중 기존에 발생한 순차 패턴 및 추후 발생할 순차 패턴과 연관성이 있는 패턴에 대해 성공적인 prefetch를 진행할 수 있다. 추가적인 prefetch hit 발생을 통해 평균 주소 변환 시간을 단축시킬 수 있으며 이에 따라 PCM controller의 전체적인 성능 향상 또한 기대할 수 있다.

## 제 3 절 논문의 구성

본 논문은 다음과 같이 구성되어 있다. 2장에서는 기존에 진행된 prefetch 알고리즘인 stream prefetcher, signature path prefetcher,

variable length delta prefetcher에 대해 소개한다. 3장은 제안 방법 및 구현에 대한 내용이며 1절은 PCM workload가 어떠한 패턴을 가지는지에 대한 내용을, 2절은 PCM workload를 위한 독자적인 알고리즘의 필요성과 제안할 알고리즘의 전반적인 내용을, 3절은 제안한 알고리즘에 대한 구체적인 구현 방식을 제시한다. 4장은 실험에 대한 내용이며 1절은 어떠한 환경에서 실험을 진행하였는지를 제시하고 2절은 실험 결과와 이에 따른 분석을 진행한다. 마지막 5장에서 본 연구의 효용성을 제시하고 제안한 prefetcher를 통한 성능 분석으로 마무리한다.

# 제 2 장 관련 연구

## 제 1 절 Stream Prefetcher

Stream prefetcher는 순 방향 순차 패턴을 효율적으로 탐지하여 prefetch를 진행하는 방법이다. 일반적인 prefetcher들은 추후 메모리 접근 가능성이 높은 주소에 해당하는 데이터를 상위 레벨의 cache에 저장을 하지만 stream prefetcher의 경우 stream buffer라는 별도의 저장공간에 데이터를 저장한다. Stream buffer의 경우 stream prefetcher의 parameter인 depth만큼의 저장공간을 가진다.

상위 레벨의 cache에서 miss가 발생할 경우 stream prefetcher에 접근한다. Stream prefetcher는 해당 주소의 다음 주소부터 depth개의 메모리 접근 요청을 발행한다. 메모리 접근 요청이 발행된 주소들은 stream buffer에 할당된다. 이 때 가장 작은 주소 값, 즉 miss가 발생한 주소의 다음 값을 stream buffer의 head로 설정한다. 이후 상위 레벨의 cache에서 miss가 발생할 경우 stream buffer의 head를 확인한다. 만약 miss가 발생한 주소 값이 stream buffer의 head와 동일할 경우 이는 주소 접근이 순차적임을 의미한다. 이에 따라 head의 데이터는 상위 레벨의 cache로 옮겨지고, head의 다음 주소 값을 stream buffer의 head로 재설정한다. Prefetch buffer는 하나의 여유공간이 생기며 prefetch 요청을 추가적으로 발행함에 따라 순 방향 순차패턴에 대한 지속적인 prefetch를 진행할 수 있다. Stream buffer의 head와 상위 레벨의 cache에서 miss가 발생한 주소 값이 일치하지 않을 경우

이는 메모리 접근 패턴이 순차적이지 않음을 의미한다. 이에 따라 prefetch buffer에 저장된 값을 모두 지우고 miss가 발생한 주소의 다음 값부터 depth개의 주소를 prefetch buffer에 할당한다.

Stream buffer가 하나만 존재할 경우 일부 비 순차 패턴에 의해 prefetch buffer에서 유용한 패턴이 방출되어 불필요한 메모리 접근이 다수 발생한다. 다수의 stream buffer를 사용하는 것은 이를 효과적으로 해결할 수 있다. 상위 레벨의 cache에서 miss가 발생한 주소 값은 stream buffer들의 head 값을 확인하여 기존 패턴의 연속인지를 확인한다. 기존 패턴의 연속이 아닐 경우 하나의 stream buffer를 할당하여 prefetch를 진행할 수 있다. 이는 유용한 패턴의 방출을 방지할 뿐 아니라 다수의 순차 패턴이 발생하는 경우, 각각의 순차 패턴에 대해 성공적인 prefetch를 진행할 수 있게 한다.

## 제 2 절 Signature Path Prefetcher

복잡한 패턴은 단순한 알고리즘을 가진 prefetcher로는 성공적인 prefetch를 진행할 수 없다. 복잡한 패턴에 대해 성공적인 prefetch를 진행하기 위해 학습이 충분히 이루어져야 하는데, signature path prefetcher는 이러한 복잡한 패턴을 효율적으로 저장하며, 정확한 예측을 진행한다.

상위 레벨의 cache에서 miss가 발생시 해당 주소를 통해 signature path prefetcher를 학습시킨다. 우선적으로 signature table에 접근하여 해당 page에 대한 초기 접근 여부를 확인한다. 초기 접근일 경우 page 값으로 signature table의 tag를 update해주며 offset 값으로 signature table의 last offset을 update 해준다. 초기 접근일

경우 별도의 prefetch를 진행하지 않는다. 두 번째 접근일 경우 기존의 last offset을 통해 얻은 delta 값으로 signature table의 signature 값을 update해주며 새로운 offset 값으로 last offset 값을 update 해준다. 이후에 발생하는 해당 page 접근의 경우 두 번째 접근과 마찬가지로 last offset 값을 update 해주며 기존의 last offset 값을 통해 얻은 delta 값으로 signature 값 또한 update 해준다. Delta 패턴의 효율적인 저장을 위해 기존의 signature 값을 left shift 시킨 뒤 delta 값과의 XOR 연산을 통해 새로운 signature 값을 얻을 수 있다. 즉 signature 값은 해당 page 주소들의 delta 패턴에 대한 정보이다.

Pattern table은 signature 값에 따라 발생 가능성 높은 delta 값들을 저장하는 역할을 한다. Signature table에 접근 시 해당 page에 signature 값이 존재할 경우 기존의 signature 값과 새롭게 얻은 delta 값을 pattern table에 전달한다. 우선적으로 signature 값과 delta 값을 통해 pattern table을 학습시킨다. 다음으로 signature 값과 delta값을 통해 signature table에서 update 된 signature 값을 얻는다. 새롭게 얻은 signature 값을 통해 pattern table을 탐색하여 일치하는 signature 값이 있는지 확인한다. 일치하는 signature 값이 있을 경우 이는 delta 패턴이 이전에도 발생함을 의미한다. 해당 signature의 delta confidence를 확인하여 가장 높은 confidence를 가지고 있는 영역의 delta값과 delta confidence값을 얻는다. Delta confidence를 signature의 signature confidence로 나눈 값은 signature 이후 발생한 delta 값들 중 해당 delta 값이 발생할 확률을 의미한다. 만약 이 값이 signature path prefetcher의 parameter 값인 prefetch threshold보다 크다면 delta 값을 통해 다음 주소를 예측할 수 있다. 이후 pattern table을 통해 예측한 delta 값과 pattern table을 탐색한 signature 값을

통해 새로운 signature 값을 얻고 이를 통해 pattern table을 다시 탐색한다. 이러한 lookahead 방식을 통해 지속적인 prefetch를 진행할 수 있다.

만약 예측한 주소 값이 해당 page를 넘어가게 될 경우, prefetch를 진행하지 않고 global history register를 update한다. Global history register를 update함에 따라 page에 대한 초기 접근 발생시에도 prefetch를 진행할 수 있게 된다. 만약 예측한 주소 값이 해당 page를 넘어가지 않게 되면 prefetch filter에 접근한다. Prefetch filter는 최근에 발생한 prefetch 주소 값을 저장하는 역할을 한다. 예측한 주소 값이 prefetch filter에 존재함은 해당 주소에 대한 prefetch 요청이 이미 발생함을 의미하며, prefetch를 진행할 필요가 없다. 만약 prefetch filter에 존재하지 않을 경우 prefetch filter를 update해주며 prefetch를 진행한다.

Signature path prefetcher의 경우 signature 값을 통한 delta 패턴 저장으로 효율적으로 패턴을 저장할 수 있으며 하나의 주소에 대한 지속적인 pattern table 탐색을 통해 복잡한 패턴에 대해서도 성공적인 prefetch를 진행할 수 있다. Prefetch filter를 통해 중복된 메모리 접근을 최소화할 수 있어 낮은 prefetch threshold값에 대해서도 높은 성능을 유지할 수 있다.

## 제 3 절 Variable Length Delta Prefetcher

단일 workload 내에서도 다양한 주소 패턴이 발생한다. 다양한 패턴들 중 단순한 패턴들은 쉽게 예측할 수 있고 훈련을 위해 적은

시간이 필요하지만 복잡한 패턴들은 예측이 어려우며 훈련을 위해 많은 시간이 필요하다. 주소 값이 prefetcher로 전달되었을 때 해당 주소 값이 단순한 패턴인지 복잡한 패턴인지 결정 짓기에 어려움이 따른다. Variable length delta prefetcher는 다양한 길이의 패턴을 효율적으로 탐지하여 성공적인 prefetch를 진행한다.

상위 레벨의 cache에서 miss가 발생 시 주소 값은 variable length delta prefetcher로 전달된다. 우선 delta history buffer에 접근을 하게 되며 해당하는 주소 값의 page가 존재하는지 확인한다. 존재할 경우 해당 page에 마지막으로 접근했던 주소 값을 통해 delta 값을 얻는다. 이를 통해 delta history buffer의 last 4 deltas를 update 해주고 last address 또한 update 한다. 이후 last 4 deltas를 delta prediction table로 전달한다. 만약 해당하는 page가 존재하지 않을 경우 이는 해당 page에 대한 초기 접근임을 의미한다. 이에 따라 nMRU에 따라 delta history buffer를 업데이트 해주며 주소를 offset prediction table로 전달한다.

Delta prediction table은 다양한 길이의 delta 패턴에 대해 성공적인 prefetch를 진행하기 위해 여러 개의 table을 가진다. 3개의 table을 가지고 있을 경우 첫 번째 table은 하나의 delta 값을 통해 다음 주소를 예측하며, 두 번째 table은 두개의 delta 값, 세 번째 table은 세개의 delta 값을 통해 다음 주소를 예측한다. 앞선 delta history buffer에서 최대 4개의 delta 값을 얻게 되고 이를 통해 delta prediction table을 update 할 수 있다. 만약 4개의 delta 값을 얻었을 경우 가장 최근에 발생한 delta를 제외하고 3개의 delta로 구성된 패턴이 생기며 세 번째 table에 해당하는 패턴이 존재하는지 확인한다. 패턴이 존재할 경우 해당 패턴에 해당하는 prediction 값과 delta

history buffer에서 받은 4개의 delta 값 중 가장 최근에 발생한 delta 값을 비교한다. 만약 일치할 경우 accuracy 값을 하나 증가시키며 일치하지 않을 경우 accuracy 값을 하나 감소시킨다. Accuracy 값이 감소하여 0이 될 경우 prediction 값을 교체해 준다. 얻은 delta의 개수가 3개일 경우 두 번째 table을 2개일 경우 첫 번째 table을 update 할 수 있다.

Prediction table을 update 해준 뒤 delta history buffer에서 받은 delta 값을 사용해 3개의 prediction table을 모두 탐색한다. 가장 많은 수의 delta를 사용하여 다음 delta를 예측하는 table에게 우선권이 주어진다. 즉 세 번째 table에서 hit이 발생하며 accuracy가 0이 아닐 경우 해당 table의 prediction 값을 사용한다. 세 번째 table에서 miss가 발생하며, 두 번째 table에서 hit이 발생하며 accuracy가 0이 아닐 경우 두 번째 table의 prediction 값을 사용한다. 얻은 prediction 값은 variable length delta prefetcher에 접근한 주소에 더해져 prefetch 요청이 발행된다.

Offset prediction table의 경우 해당 offset 값에 따른 delta 값을 예측한다. Offset prediction table은 delta history buffer에서 하나의 delta 값을 받았을 경우 update 될 수 있다. Delta 값이 하나일 경우 variable length delta prefetcher에 접근한 주소의 offset 값과 delta 값을 통해 offset prediction table을 update한다. Delta prediction table에서 사용한 accuracy 설정 방식으로 prediction 값을 변경할 수 있다. Delta history buffer에서 해당 page에 대한 초기 접근일 경우 주소 값은 offset prediction table로 전달된다. 해당 주소의 offset 값을 통해 offset prediction table을 탐색하며 일치하는 offset을 탐색한다. 일치하는 offset 값이 존재하며 accuracy 값이 0이 아닐 경우

prediction 값은 variable length delta prefetcher에 접근한 주소 값에 더해져 prefetch 요청이 발행된다.

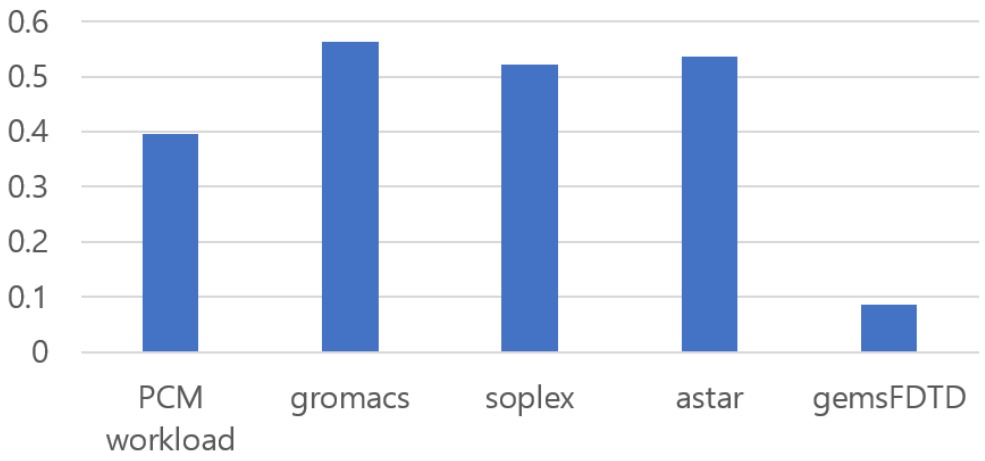
Variable length delta prefetcher의 경우 다수의 delta prediction table을 사용함에 따라 다양한 길이의 패턴에 대해서도 성공적으로 prefetch를 진행할 수 있으며, page에 대한 초기 접근에도 offset prediction table을 통해 prefetch hit을 발생시킬 수 있다.

# 제 3 장 제안 방법 및 구현

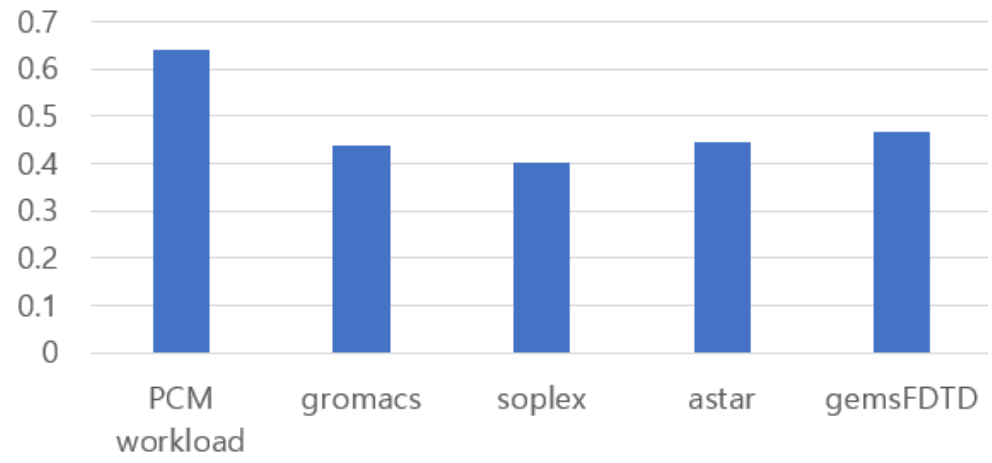
## 제 1 절 패턴 분석

PCM에 접근하기 위해 PCM controller에서 논리적 주소를 물리적 주소로 변환하는 과정을 거친다. 논리적 주소를 물리적 주소로 변환하기 위해 주소 변환 데이터가 필요하다. 이 데이터는 별도의 메모리에 저장되어 있기 때문에 주소 변환을 위한 메모리 접근은 필수적이다. 주소 변환을 위한 메모리 접근은 앞선 주소 변환 요청이 완료되기 전까지 지연된다. 이에 따라 PCM controller의 전체적인 성능이 저하된다. PCM controller에서 주소 변환 데이터를 요구하기 이전에 필요한 데이터를 예측하여 prefetch 하는 것은 주소 변환을 위한 지연 시간을 줄일 수 있으며 이에 따라 전체 시스템의 성능 향상 또한 기대할 수 있다. PCM controller에서 요청할 데이터를 예측하는 과정은 이전에 발생한 주소 변환 요청들의 패턴 분석을 통해 이루어진다. 따라서 prefetch를 진행하기 위해 주소 변환 요청들이 어떠한 패턴을 가지는지에 대한 분석이 선행되어야 한다. PCM workload의 주소 변환 패턴에 대한 정량적 분석을 위해 PCM workload로 UMASS Financial 1_08 workload를 사용하였다.

[그림 3-1]은 PCM workload와 이전 연구들이 목표로 하는 workload인 SPEC CPU 2006 workload의 비 순차 패턴 비율을 나타낸다. SPEC CPU 2006의 경우 gemsFDTD를 제외하고 높은 비 순차 패턴 비율을 보이며 PCM workload의 경우 40% 수준의



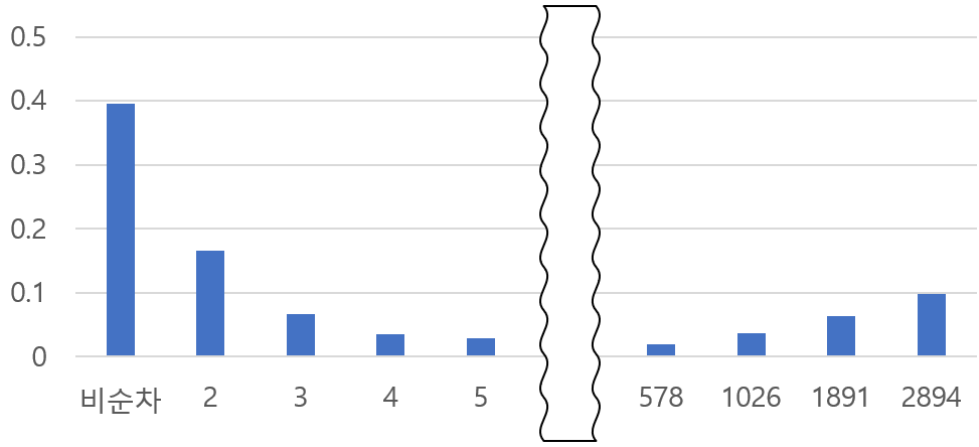

[그림 3-1] PCM workload와 SPEC CPU 2006 workload의 비 순차 패턴 비율




[그림 3-2] PCM workload와 SPEC CPU 2006 workload의 랜덤 delta 패턴 비율

상대적으로 낮은 비 순차 패턴 비율을 보인다.

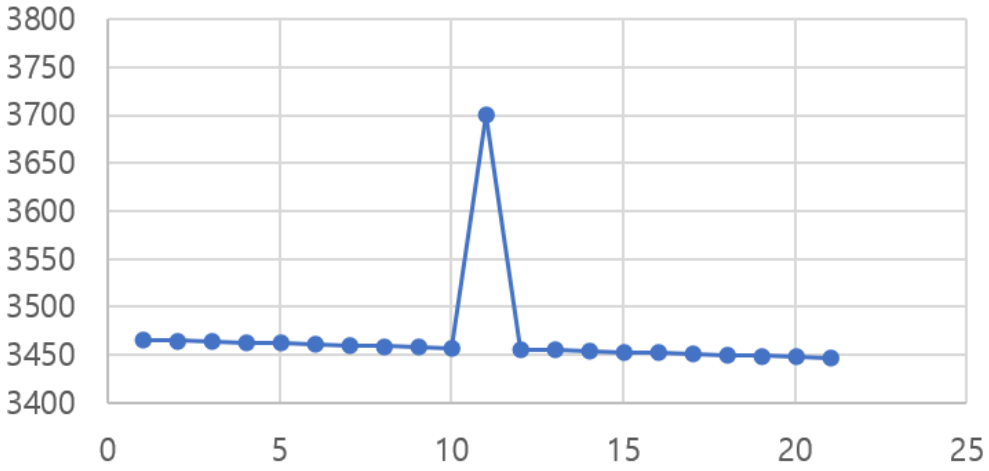
비 순차 패턴임은 복잡한 패턴 혹은 랜덤 패턴임을 의미한다. 비 순차 패턴 중 랜덤 패턴의 비율을 확인하기 위해 SPP와 VLDP에서 택하는 방식인, 동일 tag를 가지는 주소들의 delta 패턴을 관찰하였다. 특정 delta 패턴이 전 workload에 있어서 한 번만 발생할 경우 이는




[그림 3-3] PCM workload의 패턴 길이 분포

복잡한 패턴보단 랜덤 패턴에 가까움을 의미한다. Workload들의 tag 당 누적 delta 개수가 4개 이상일 경우를 추출하였고, 이 중 단 한 번만 쓰인 delta 패턴의 비율을 측정하였다. [그림 3-2]은 이전 연구의 workload와 PCM workload의 랜덤 delta 패턴의 비율이다. PCM workload의 경우 한 번만 쓰인 delta 패턴이 상대적으로 많이 존재하였으며 SPEC CPU 2006 workload들의 경우 상대적으로 적게 존재하였다. PCM workload의 경우 낮은 비 순차 패턴 비율을 가지고 있음에도 랜덤 패턴 비율이 높다. 이는 순차 패턴을 제외한 비 순차 패턴들이 복잡한 패턴보다 랜덤 패턴임을 의미한다.

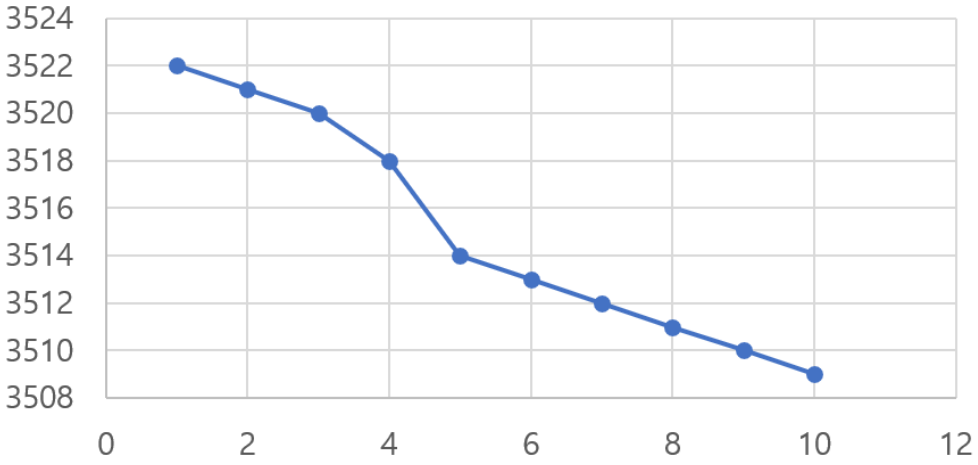
PCM workload에서 60%를 차지하는 순차 패턴의 경우 크게 두가지 특징을 가진다. 첫 번째로 순 방향 순차 패턴과 역 방향 순차 이 혼합되어 있다. 순 방향 순차 패턴의 경우 전체 workload의 36%를 차지하며 역 방향 순차 패턴의 경우 24%를 차지한다. 어떠한 패턴을 가지는지는 주소 값과 무관하다. 즉 이전에 순 방향으로 접근한 주소가 추후 역 방향으로 접근할 수도 있으며 비 순차적으로 접근할 수도 있다. 두 번째로 높은 패턴 지속성을 가진다. [그림 3-3]은 PCM workload의




[그림 3-4] 비 순차 패턴 예시 1

패턴 길이 분포다. 패턴의 길이는 2부터 2911까지 다양한 분포를 가지며 길이가 긴 패턴이 주를 이룬다. 주소 변환 요청 발생 시 해당 주소의 패턴 길이를 예측함에 어려움이 따른다. 이에 따라 특정 패턴의 마지막 주소 변환 요청 발생 시 불필요한 prefetch 요청 발행은 불가피하다. 길이가 긴 패턴이 다수 존재함은 불필요한 prefetch 요청 발행을 최소화할 수 있음을 의미한다.

PCM workload의 비 순차 패턴 비율은 약 40%이다. 즉 순차 패턴 및 역 방향 순차 패턴만 선별하여 prefetch를 하게 될 경우 40%의 주소 변환 요청은 필수적으로 메모리에 접근해야 함을 의미한다. 추가적인 성능 향상을 위해 prefetch 가능한 비 순차 패턴을 탐지하는 것이 필요하다. PCM workload의 비 순차 패턴은 크게 두 종류로 분류된다. 첫 번째로 기존의 패턴과 어떠한 관련성도 없는 패턴이다. [그림 3-4]의 경우 첫 번째 비 순차 패턴 종류의 대표적인 예시이다. [그림 3-4]의 열한 번째 발생한 주소 값의 경우 앞서 발생한 역 방향 순차 패턴의 마지막 값, 그리고 추후 발생할 역 방향 순차 패턴의 처음

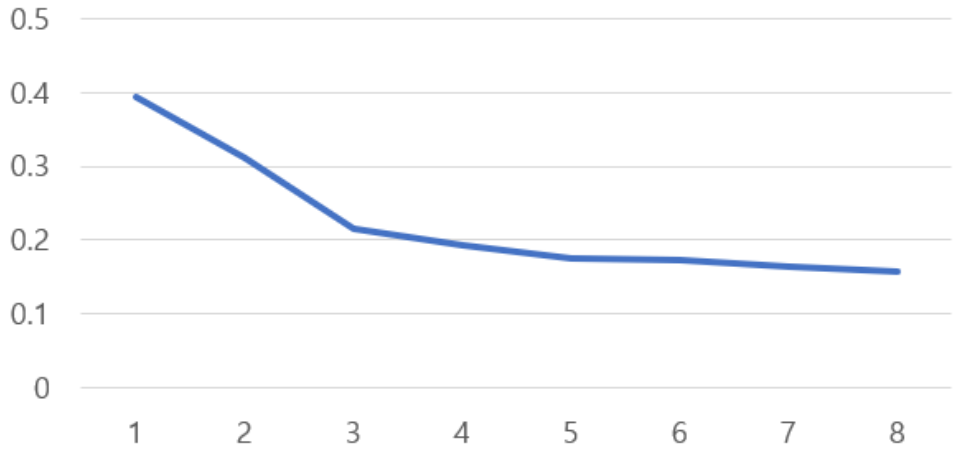



[그림 3-5] 비 순차 패턴의 예시 2

값과 큰 차이를 가진다. 두 번째는 기존의 순차 패턴과 관련성이 존재하는 패턴이다. [그림 3-5]의 경우 두 번째 비 순차 패턴 종류의 대표적인 예시이다. 네 번째 발생한 주소 값이 앞서 발생한 역 방향 순차 패턴의 마지막 값과 추후 발생할 역 방향 순차 패턴의 처음 값 사이에 위치하며 적은 차이를 가진다.

순 방향 순차 패턴 및 역 방향 순차 패턴이 지속되기 위해서 접근 주소는 기존에 존재하는 패턴의 마지막 값, 혹은 추후 발생할 패턴의 처음 값과의 차이가 1이여야 하며 차이의 방향성이 패턴의 방향성과 같아야 한다. 비 순차 패턴의 경우 PCM workload 내에 존재하는 어떠한 패턴과도 이러한 조건을 만족하지 못하기에 발생한다. [그림 3-5]의 경우 네 번째 발생한 주소 값은 앞서 발생한 패턴의 마지막 값과 2의 차이를 가지며 추후 발생할 패턴의 처음 값과 4의 차이를 가지기 때문에 비 순차 패턴으로 분류된다. 이러한 비 순차 패턴의 발생은 패턴의 지속성을 방해하는 요인이며 지속적인 prefetch를 방해한다.

패턴 지속 조건 변경 시 두 번째 종류의 비 순차 패턴을 순 방향 순차 패턴 혹은 역 방향 순차 패턴의 일부로 포함시킬 수 있다. 패턴




[그림 3-6] Endurance에 따른 비 순차 패턴 비율

지속성의 조건을 기존에 존재하는 패턴의 마지막 값, 혹은 추후 발생할 패턴의 처음 값과의 차이가 n이여야 하며 차이의 방향성이 패턴의 방향성과 같아야 한다 로 재정의 하고 n의 값을 endurance로 명명한다. Endurance 값을 4로 설정 시, [그림 3-5]의 네 번째 주소 값은 앞선 패턴의 마지막 값과의 차이가 2로 설정한 endurance보다 작기 때문에 앞선 패턴에 포함시킬 수 있다. 뿐만 아니라 추후 발생하는 패턴과 네 번째 주소 값의 차이는 설정한 endurance와 같은 값을 가지므로 추후 발생할 패턴에 포함시킬 수 있다. 이에 따라 [그림 3-5]의 모든 주소들은 gap이 존재하는 하나의 역 방향 순차 패턴으로 여길 수 있다. [그림 3-6]은 endurance에 따른 비 순차 패턴 비율 그래프이다. Endurance가 높아짐에 따라 기존 순차 패턴과 관련성이 존재하는 비 순차 패턴이 기존에 존재하는 순차 패턴에 포함된다. 이에 따라 PCM workload의 비 순차 패턴 비율은 40%에서 16%까지 낮아진다.

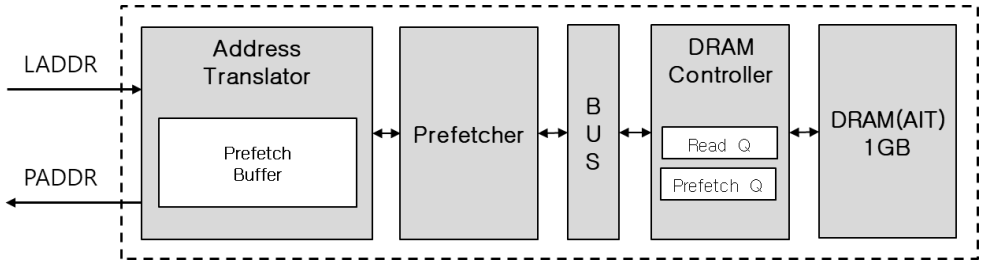
PCM workload들의 경우 높은 지역성을 가지며 순차 패턴이 주를 이룬다. 이에 따라 순차 패턴을 효율적으로 탐지하여 prefetch하는 것이

중요하다. 비 순차 패턴들의 경우 뚜렷한 패턴이 존재하지 않는 랜덤 패턴이 주를 이룬다. 두 종류의 비 순차 패턴 모두 정확히 다음 접근 주소를 예측하는 것은 어렵다. 하지만 두 번째 비 순차 패턴과 같이 gap이 존재하는 하나의 순차 패턴으로 여길 수 있는 패턴의 경우 적절한 prefetch depth 및 prefetch endurance 설정을 통해 추가적인 prefetch hit을 기대할 수 있다.

## 제 2 절 제안 방법

Stream prefetcher의 경우 순 방향 순차 패턴에 대해 효율적인 prefetch를 진행하기 위해 연구되었다. SPP와 VLDP의 경우 gemsFDTD와 같이 순차 패턴 비율이 높은 workload뿐만 아니라 gromacs, soplex, astar와 같이 복잡한 패턴 비율이 높은 workload에서도 성공적인 prefetch를 하기 위해 연구되었다. PCM workload들의 경우 이전 연구들이 목표로 하는 workload들과 다른 양상을 보인다. [그림 3-1]에서 확인할 수 있듯 PCM workload는 낮은 비 순차 패턴 비율을 보이며 [그림 3-2]에서 확인할 수 있듯 높은 랜덤 패턴 비율을 보인다. 복잡한 패턴에 대한 효율적인 prefetch를 진행할 수 있는 SPP와 VLDP의 경우 랜덤 패턴에 대해 prefetch hit을 기대할 수 없으며 stream prefetcher의 경우 랜덤 패턴을 효율적으로 탐지할 수 없어 불필요한 메모리 접근을 다수 발생시킨다. 이는 PCM controller의 성능 향상을 위해 새로운 알고리즘이 필요함을 의미한다.

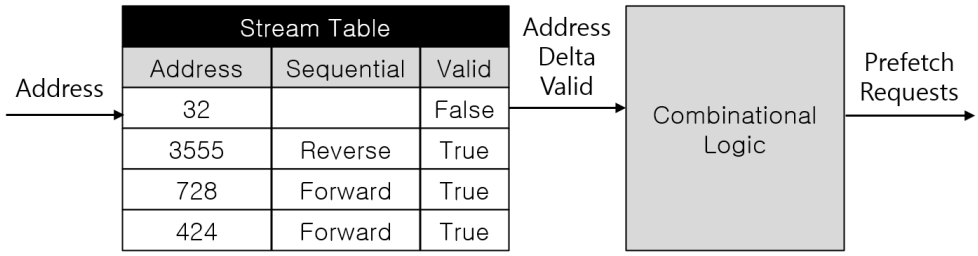
PCM controller에서는 주소 변환 데이터가 저장되어 있는 DRAM을 제외하고 별도의 저장 공간이 존재하지 않아 prefetch된




[그림 3-7] PCM controller의 주소 변환 과정 1

데이터를 저장할 prefetch buffer가 필요하다. 본 연구에서는 prefetch buffer로 8KB 8-way cache를 사용하였다. [그림 3-7]은 PCM controller의 주소 변환 시스템이다. 주소 변환 요청 발생 시 address translator에서 해당하는 주소를 prefetcher로 보낸다. Prefetcher에서 받은 주소를 통해 이후 발생할 가능성이 높은 주소 값을 예측하여 DRAM으로 보내며, DRAM으로부터 얻은 주소 변환 데이터는 address translator의 prefetch buffer에 저장된다. DRAM으로부터 주소 변환 데이터를 받아오는 과정은 약 30 cycle이 소요된다. 예측한 주소 값이 추후 발생하게 된다면 prefetch buffer를 통해 해당하는 데이터를 얻을 수 있으며 이는 1 cycle이 소요된다. PCM controller에서 prefetch hit이 발생할 경우 miss가 발생한 경우에 비해 큰 이득을 얻을 수 있다. 또한 PCM workload의 경우 순차 패턴의 비율이 높으며, 일부 비 순차 패턴의 경우 기존에 존재하는 순차 패턴과 연관성이 있음을 확인하였다. 이러한 비 순차 패턴에 대한 prefetch hit을 발생시키기 위해 본 연구에서는 stream prefetcher와 유사한, 새로운 prefetch 방식을 제안한다.

제안하는 prefetcher의 경우 두가지 특징을 가진다. 우선 순 방향 순차 패턴과 역 방향 순차 패턴에 대해 모두 prefetch를 진행할 수 있게 한다. Stream prefetcher의 경우 접근하는 모든 주소에 대해 순

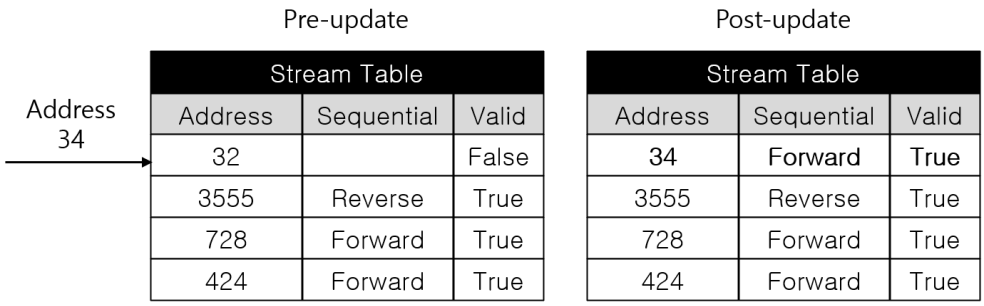



[그림 3-8] 제안하는 prefetcher의 구조

방향 순차 패턴으로 예측하기 때문에 패턴의 시작 주소에 대해서도 prefetch를 진행하지만 제안하는 prefetcher의 경우 prefetch 진행을 위하여 방향성을 결정해야 한다. 이에 따라 우선적으로 해당 주소의 접근이 순 방향 순차 패턴인지 역 방향 순차 패턴인지 추후 접근하는 주소를 통해 확인한다. 이후 주소 접근 양상에 따라 순 방향 혹은 역 방향으로 prefetch를 진행한다. 다음으로 endurance parameter 사용을 통해 두 번째 비 순차 패턴, 즉 gap이 존재하는 순차 패턴에 대해 지속적인 prefetch를 진행한다. 제안하는 prefetcher의 경우 [그림 3-5]의 3 번째 주소 발생 시 역 방향으로 depth만큼 prefetch를 진행한다. 이에 따라 4 번째 주소에 대한 prefetch hit을 기대할 수 있다. 또한 적절한 endurance 값 설정 시 4 번째 주소를 기존의 패턴의 일부로 판단하여 depth만큼 prefetch를 진행한다. 이에 따라 추후 발생하는 주소들에 대해 prefetch hit을 발생시킬 수 있으며 지속적인 prefetch를 진행할 수 있다.

## 제 3 절 구현 방법

[그림 3-8]은 본 연구에서 제안하는 prefetcher의 구조이다. 해당

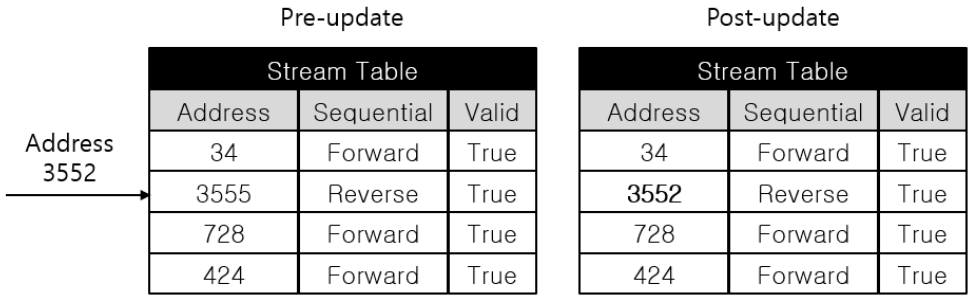



[그림 3-9] Stream Table update 예시 1

prefetcher는 stream table과 combinational logic으로 구성되어 있다. Stream table은 접근 주소 값이 기존 패턴의 일부인지 판별한다. 만약 기존 패턴의 일부일 경우 stream table 요소의 정보들을 combinational logic으로 전달한다. Combinational logic은 stream table에서 얻은 결과 값을 통해 prefetch 요청을 발행한다. 제인하는 알고리즘은 두개의 parameter를 사용한다. 첫 번째로 depth는 prefetch를 얼마나 많이 진행할 것을 의미하며 둘째로 endurance는 어느 정도의 거리까지 기존 패턴의 일부로 여기는지를 의미한다.

Stream table은 address, sequential, valid 3개의 정보를 저장한다. Address 영역은 패턴이 진행될 가능성이 존재하는 주소 값 혹은 진행되고 있는 패턴의 주소 값을 의미한다. 두 요소를 구분하기 위해 valid flag를 사용한다. Valid 값이 false일 경우 저장된 주소는 패턴이 진행될 가능성이 존재하는 주소 값을 의미하며, true일 경우 진행되고 있는 패턴의 주소 값을 의미한다. 마지막으로 sequential 영역은 valid 값이 true일 경우, 진행되고 있는 패턴의 방향성을 나타낸다. 이러한 저장방식을 통해 진행될 가능성이 존재하는 주소 값 및 진행되고 있는 패턴의 주소 값을 추가적인 메모리 사용 없이 동시에 저장할 수 있다.

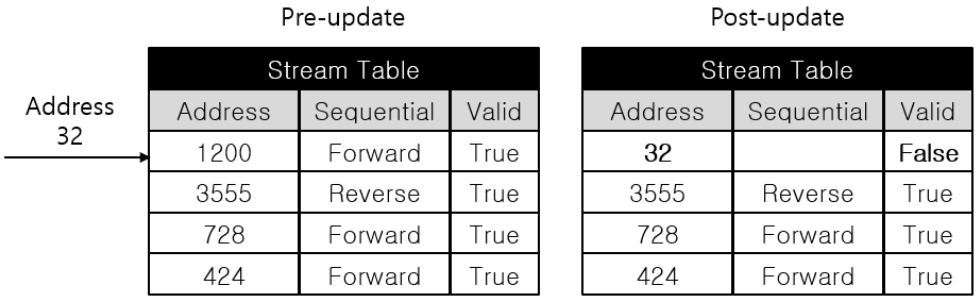
Stream table에 접근 시 해당하는 주소가 기존 패턴의 일부인지




[그림 3-10] Stream Table update 예시 2

파악하기 위해 접근 주소 값이 stream table의 주소 값과의 차이가 endurance 이하인지 파악한다. Endurance 이하일 경우 valid 값을 확인한다. Valid 값이 false임은 stream table의 해당 요소가 패턴이 진행될 가능성이 존재하는 주소 값임을 의미하며, 아직 어떠한 방향으로 prefetch를 진행할지 알 수 없음을 의미한다. 새롭게 접근한 주소 값을 통해 이 요소의 방향성을 결정할 수 있게 된다. [그림 3-9]은 이를 나타낸다. Endurance를 4로 가정하였을 때 주소 34는 stream table 요소 32와 endurance보다 적은 차이가 나며, 요소 32의 valid flag는 false이다. 이에 따라 주소 34는 stream table 요소 32 패턴의 일부이며 추후 순 방향 진행할 것으로 예측한다. 이후 [그림 3-9]와 같이 stream table을 update한다. 즉 해당하는 요소의 valid flag가 false일 경우 endurance 조건만 만족하게 되면 prefetch를 진행할 수 있게 된다.

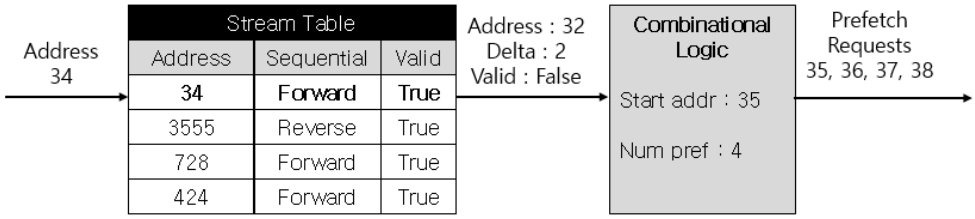
반면 valid flag가 true일 경우 진행된 패턴이 순 방향인지 혹은 역 방향인지 또한 고려해야한다. [그림 3-10]은 이를 나타낸다. 주소 3552의 경우 stream table 요소 3555와의 차이가 endurance보다 적으며 요소 3555의 valid flag는 true이다. Stream table의 해당 요소의 sequential flag는 reverse이며, 이는 해당 패턴이 역 방향으로




[그림 3-11] Stream Table update 예시 3

진행되었음을 의미한다. Sequential flag와, stream table 요소 3555 이후 주소 3552의 접근은 역 방향 진행으로 같기 때문에 기존 패턴의 일부로 판단할 수 있다. 이후 [그림 3-10]과 같이 stream table을 update하고 combinational logic으로 stream table의 데이터를 전달한다. 만약 접근 주소가 3557이었을 경우 stream table의 요소 3555와 endurance보다 적은 차이가 남에도 불구하고 접근 주소가 기존 패턴의 진행 방향과 다르기 때문에 기존 패턴의 일부가 아님으로 판별한다.

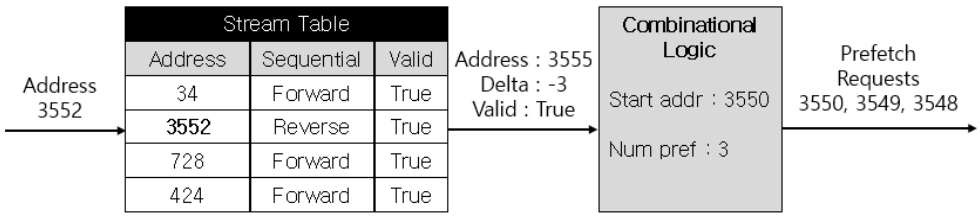
[그림 3-11]은 접근 주소가 기존에 발생한 패턴의 일부가 아닌 경우, 즉 stream table miss인 경우를 나타낸다. 주소 32의 경우 stream table의 요소와의 차이가 모두 endurance 이상이다. 이는 즉 주소 32의 경우 이전에 발생한 어떠한 패턴의 일부도 아님을 의미한다. 하지만 이 주소는 다른 패턴의 시작 주소가 될 가능성이 존재한다. 이에 따라 stream table update는 필수적이다. 만약 stream table이 모두 차 있다면 가장 마지막에 쓰인 table 요소가 stream이 앞으로 진행될 가능성이 가장 낮기 때문에 LRU 방식에 따라 table의 요소를 교체한다. [그림 3-11]의 경우 stream table의 첫 번째 요소에 주소 32가 update 되는 것을 확인할 수 있다. Stream table에서의 방출은 해당




[그림 3-12] Combinational logic의 주소 선택 예시 1

패턴이 다시 진행되기 위해서 prefetcher로 재 접근 해야함을 의미한다. PCM workload의 특성 상 주소 값에 따른 패턴이 고정적이지 않아 순방향 순차 접근을 하였던 주소가 추후 역 방향 순차 접근을 하거나 비순차적으로 접근할 가능성이 존재한다. 이전에 발생한 특정 주소가 시간이 지남에 따라 stream table에서 방출되고, 이에 따라 추후 발생한 동일한 주소는 이전 주소의 접근 패턴과 독립적으로 prefetch를 진행할 수 있다.

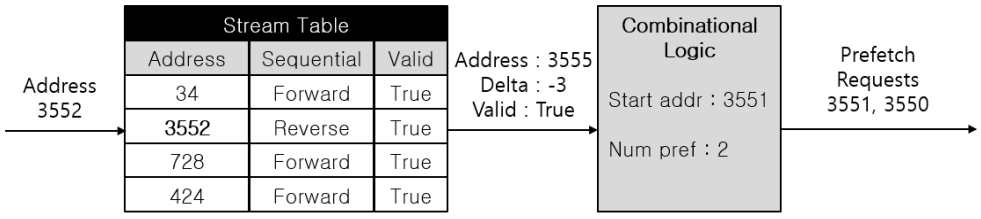
Stream table 접근 주소가 endurance 조건을 충족시키지 못하거나 충족시킴에도 방향성이 다를 경우 table miss로 판별한다. 이 경우 다른 패턴의 시작 주소가 될 가능성이 존재하지만 추후 패턴이 진행될 방향성을 예측하는 것에는 어려움이 따른다. 이에 따라 prefetch를 진행하지 않기 위해 combinational logic으로 어떠한 데이터도 보내지 않는다. Stream table hit이 발생했을 경우 combinational logic은 hit이 발생한 stream table 요소 값과 delta 값 그리고 valid flag를 전달받는다. Valid flag가 false일 경우 이는 접근 주소가 해당 패턴의 두 번째 접근임을 의미하며 해당 패턴에 있어서 진행된 prefetch가 없음을 의미한다. 반면 valid flag가 true일 경우 이는 해당 패턴에 있어서 진행된 prefetch가 있음을 의미한다. Combinational logic은 valid flag가 true일 경우 이미 발생한 prefetch 주소 값들을 배제하여




[그림 3-13] Combinational logic의 주소 선택 예시 2

prefetch를 진행한다. Combinational logic에서 이미 발생한 prefetch를 배제하는 것은 prefetch를 시작할 주소 값 설정과 진행할 prefetch 개수 설정을 통해 할 수 있다.

[그림 3-12]와 [그림 3-13]은 prefetcher 접근 주소에 따른 prefetch 주소 선택 과정이다. Endurance와 depth는 4로 가정한다. [그림 3-12]은 prefetcher 접근 주소가 34이며 stream table update 이전 valid flag가 false인 경우이다. Stream table을 통해 받은 delta 값은 2로 해당 패턴이 순 방향 패턴임을 의미한다. Valid flag는 false로 이전에 발생한 prefetch가 없으므로 고려할 필요가 없다. 이에 따라 prefetch 시작 주소는 prefetcher 접근 주소의 다음 값인 35가 되며, prefetch를 진행할 개수는 depth, 즉 4가 된다. [그림 3-13]은 prefetcher 접근 주소가 3552이며 valid flag가 true인 경우이다. Stream table을 통해 받은 stream table 요소 값은 3555이며 delta 값은 -3이다. 이는 해당 패턴이 역 방향 순차 패턴임을 의미한다. Valid flag가 true임은 해당 패턴의 이전에 발생한 주소 3555가 prefetcher에 접근하였을 시 depth만큼 prefetch를 진행하였음을 의미한다. 즉 주소 3554, 3553, 3552, 3351은 prefetch할 필요성이 없다. 이에 따라 prefetch 시작 주소를 3550으로 설정한다. 기존에 진행된 prefetch 주소가 없을 경우 prefetch 시작 주소는 3551이며 prefetch를 진행할

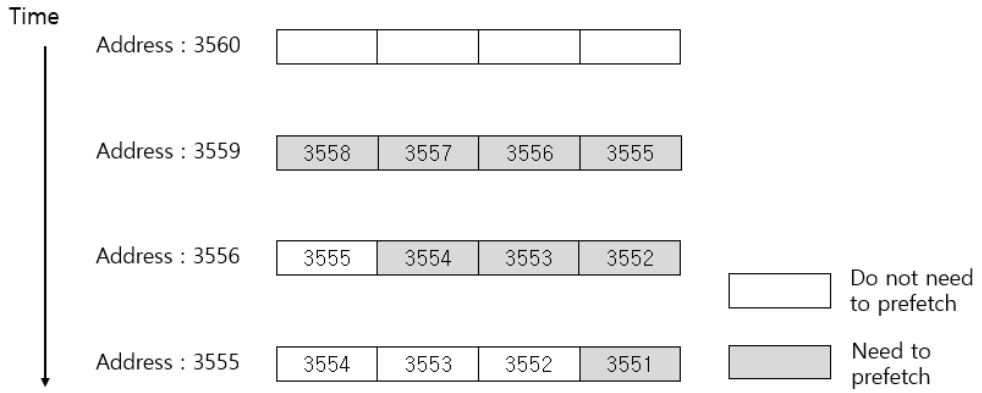



[그림 3-14] Combinational logic의 주소 선택 예시 3

개수는 depth, 즉 4이다. 위 경우 기존에 발생한 prefetch가 존재함에 따라 시작 주소가 3551에서 3550으로 변화하고, 이에 prefetch를 진행할 개수는 4에서 3이 된다. 일반적으로 prefetch를 진행할 개수는 delta의 절대값과 동일하다.

Endurance 값이 depth보다 클 경우 부수적인 logic이 필요하다. [그림 3-14]은 depth가 2이며 endurance가 4인 경우 prefetcher에 주소 값 3552가 접근한 경우이다. Stream table에서 받은 주소 값을 통해 해당 패턴의 마지막 주소 값이 3555인 것을 확인할 수 있다. 주소 값 3554와 3553은 prefetch가 진행되었음을 유추할 수 있다. 기존에 진행된 prefetch 주소가 없을 경우 prefetch 시작 주소는 3551이며 prefetch를 진행할 개수는 depth, 즉 2이다. 기존에 발생한 prefetch는 주소 3552가 접근하여 발생할 prefetch 주소의 어떠한 것들과도 겹치지 않게 된다. 이에 따라 prefetch 시작 주소는 주소 3552의 다음 값인 3551이 되고, prefetch를 진행할 개수는 depth가 된다. 즉 endurance가 depth보다 클 경우, delta의 절대값이 depth보다 큰 경우가 발생할 수 있고, 이러한 경우 valid flag가 false일 때와 마찬가지로 이전에 발생한 prefetch를 고려하지 않고 prefetch를 진행한다.

[그림 3-15]은 endurance가 4이며 depth가 4일 경우 접근




[그림 3-15] 접근 주소에 따른 prefetch 진행

주소에 따른 prefetch 진행 상황을 나타낸다. 패턴의 시작 주소인 3560이 접근 시 해당 패턴이 순 방향 패턴일지 역 방향 패턴인지 알 수 없기 때문에 prefetch를 진행하지 않는다. 두 번째 주소인 3559 접근 시 해당 패턴은 역 방향 순차 패턴임을 확인하고 해당 방향으로 depth 만큼 prefetch를 진행한다. 다음으로 주소 3556과 3555가 prefetcher에 접근 시 combinational logic을 통해 이미 발생한 prefetch 주소를 제외하여 prefetch를 진행한다.

# 제 4 장 실 험

## 제 1 절 실험 환경 및 구성

제안한 prefetcher는 Linux Kernel 4.4.0 version에서 C++ 언어로 구현하였다. PCM simulator의 경우 cycle accurate하게 구현하였으며 주소 변환 데이터를 가지고 있는 DRAM의 경우 cycle accurate한 실험 진행을 위하여 DRAMsim3를 사용하였다. 주소 변환 데이터의 크기는 64B이며 prefetch buffer의 경우 32KB/8-way cache를 기본 값으로 설정하였다.

제안한 prefetcher의 성능 평가를 위하여 stream prefetcher, signature path prefetcher, variable length delta prefetcher 세개의 알고리즘을 사용하였다. Cycle accurate한 PCM simulator와의 호환성을 위해 세개의 알고리즘 모두 C++언어로 재 구현하였다. Stream prefetcher의 경우 성능 평가에 있어 다른 prefetch 알고리즘과의 통일성을 위해 prefetch한 데이터를 자체 stream buffer가 아닌 prefetch buffer에 저장한다. Stream prefetcher에서는 제안한 알고리즘에서 사용한 stream table과 유사한 방법으로 prefetch를 진행하며 기존 stream prefetcher 알고리즘과 동일하게 작동하도록 구현하였다. 즉 본 연구에서의 stream prefetcher는 prefetch 요청을 발행하는 역할만 한다. 상위 레벨 cache가 존재하지 않을 경우 stream prefetcher 사용 시 [300, 306, 300, 306]과 같이 진동하는 주소 접근에 대해 중복된 request를 다수 발생시키며 이에

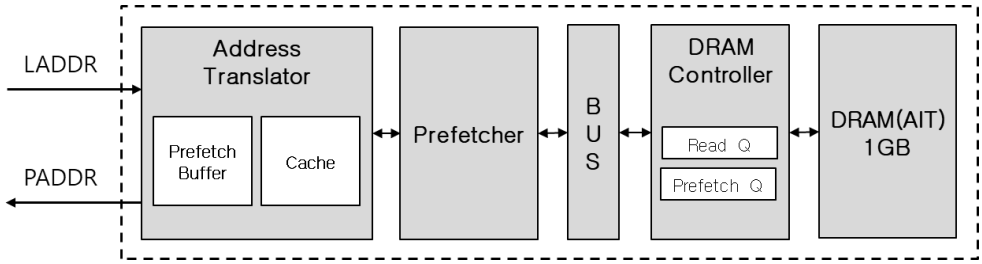
| Prefetcher Type | Parameters |
|---|---|
| Proposed Prefetcher | 32 entry Stream Table |
| SPP | 64 entry ST, PT, GHR, PF |
| VLDP | 64 entry DHB, OPT, 64 entry DPT x3 |
| Stream Prefetcher | 32 entry Stream Table |

[표 4-1] Prefetcher parameters

따라 전체적인 성능 저하를 야기할 수 있다. 이러한 문제를 해결하기 위해 stream prefetcher의 경우 다른 prefetcher들과 마찬가지로 불필요한 request를 제거하는 prefetch filter를 추가적으로 구현하였다.

각각의 prefetcher의 parameter의 경우 PCM workload에서의 효율적인 동작을 위해 재 조정되었다. 제안한 prefetcher의 경우 depth와 endurance 모두 3으로 설정하였다. Signature path prefetcher의 prefetch threshold의 경우 제안한 prefetcher와 비슷한 수준의 depth로 prefetch를 진행하기 위해 0.25로 설정하였다. Stream prefetcher의 경우 제안한 prefetcher와 동일하게 depth를 3으로 설정하였다. 기타 prefetcher parameter는 [표 4-1]을 통해 확인할 수 있다.

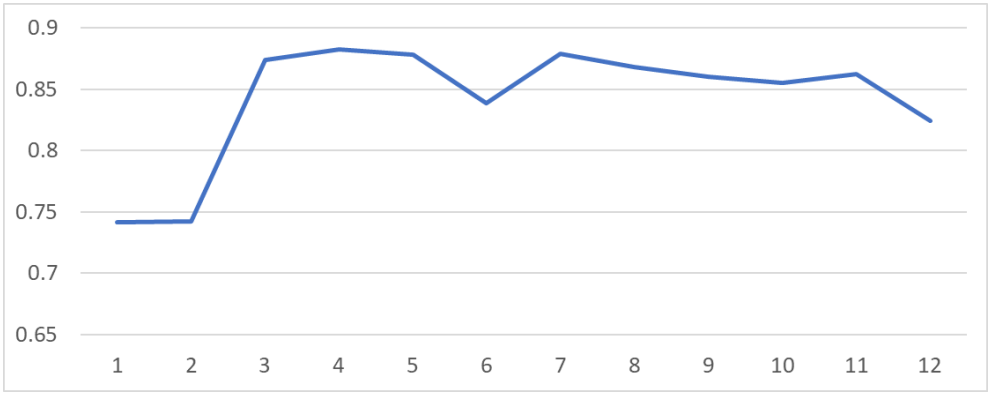
성능 평가는 크게 세 가지로 구분된다. 우선 UMASS Financial 1_08 workload에 대해 prefetch parameter에 따른 성능 변화 실험을 진행한다. Prefetch depth를 3으로 고정하고 endurance를 1부터 12까지 변화시키며 실험을 진행한다. 다음으로 prefetch endurance를 3으로 고정한 뒤 depth를 1부터 12까지 변화시키며 실험을 진행한다. Parameter 값에 따른 성능 평가는 prefetch coverage, prefetch accuracy, prefetch 요청 수 및 전체 시스템 지연 시간을 통해 진행한다.




[그림 4-1] PCM controller의 주소 변환 과정 2

다음으로 UMASS Financial 1 workload들에 대해 이전 연구의 prefetcher를 사용한 경우와 제안한 prefetcher를 사용한 경우를 비교한다. 이에 대한 평가는 prefetch coverage, 정규화 된 발행 prefetch 요청 수, prefetch accuracy, 최대 주소 변환 시간, 평균 주소 변환 시간 및 정규화 된 전체 시스템 지연 시간을 통해 진행한다.

마지막으로 UMASS Financial 1 workload들에 대해 prefetcher와 cache 모두 사용하지 않은 경우와 제안한 prefetcher만 사용한 경우, cache만 사용한 경우, 그리고 제안한 prefetcher와 cache 모두 사용한 경우를 비교한다. Cache의 경우 [그림 4-1]과 같이 address translator 내부에 구현하였으며 64KB/8-way cache를 사용하였다. Prefetcher와 cache를 모두 사용한 경우, address translator에 주소 변환 요청 접근 시 우선적으로 prefetch buffer를 탐색한다. 만약 prefetch buffer에서 hit이 발생할 경우 cache에 접근하지 않는다. Prefetch buffer에서 miss가 발생할 경우 해당 주소는 cache와 prefetcher에 동시 접근한다. Cache에서 hit이 발생할 경우 5 cycle 이내에 주소 변환 데이터를 얻을 수 있다. Cache의 hit, miss와 무관하게 prefetch를 진행함으로 prefetcher가 패턴에 대해 충분히 학습할 수 있으며, 이에 따라 prefetch buffer에서 hit이 발생할 확률을

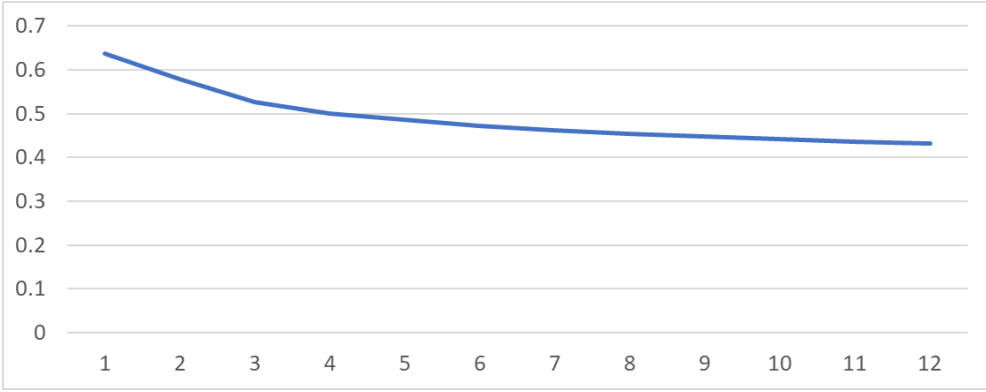



[그림 4-2] Depth가 3일 때 endurance 값에 따른 prefetch coverage 변화

높인다. Prefetcher 유무 및 cache 유무에 따른 평가의 경우 최대 주소 변환 시간, 평균 주소 변환 시간 및 정규화 된 전체 시스템 지연 시간을 통해 진행한다.

## 제 2 절 실험 결과

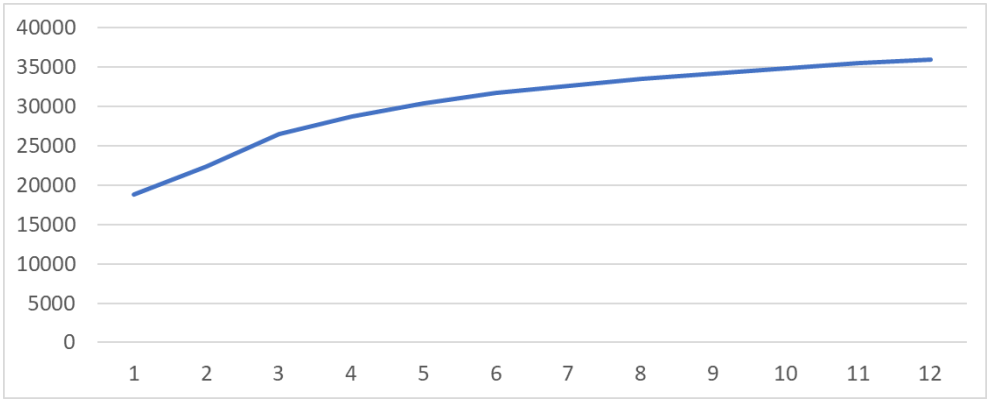
[그림 4-2]은 depth가 3일 때 endurance 값에 따른 prefetch coverage 변화이다. Endurance 값이 낮을 경우 prefetch coverage는 낮은 값을 가지며 endurance가 3에서 5 사이일 경우 비슷한 수준의 prefetch coverage를 가진다. 이후 endurance가 커짐에 따라 prefetch coverage가 낮아지는 양상을 보인다. Endurance 값이 작을 경우 기존 패턴의 일부임에도 prefetch를 진행할 수 없다. 즉 gap이 존재하는 순차 패턴에 대해 prefetch hit을 발생시킬 수 없으며 이에 따라 낮은 prefetch coverage를 가진다. Endurance 값을 매우 높게 설정할 경우 기존 패턴의 일부가 아님에도 prefetch를 진행하게 된다. 제안한 prefetcher의 경우 접근한 주소가 기존 패턴의 일부로 판단될 경우



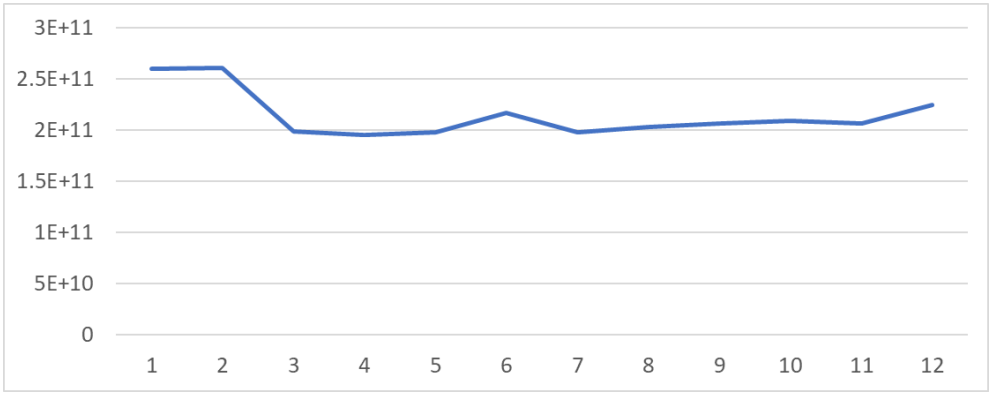

[그림 4-3] Depth가 3일 때 endurance 값에 따른 prefetch accuracy 변화

stream table을 해당 주소로 업데이트 하게 되는데, 기존 패턴의 일부가 아님에도 stream table을 하게 될 경우 기존 패턴이 stream table에서 방출된다. 이에 따라 패턴 pollution이 발생하며 기존 패턴에 해당하는 주소가 들어오게 될 경우에도 알맞은 prefetch를 진행할 수 없게 된다. 뿐만 아니라 패턴에 대한 잘못된 예측은 prefetch buffer에서 유용한 주소 변환 데이터를 방출시킨다. 이에 따라 endurance 값이 커질수록 prefetch coverage 값은 낮아지게 된다. Endurance 값의 증가는 gap이 존재하는 순차 패턴에 대한 추가적인 prefetch hit을 발생시키지만 stream table의 패턴 pollution과 prefetch buffer pollution을 발생시킨다. 상충되는 효과로 endurance 값이 3에서 7 사이일 경우 비슷한 수준의 prefetch coverage를 가진다.

[그림 4-3]은 depth가 3일 때 endurance 값에 따른 prefetch accuracy 변화이다. Endurance 값이 커짐에 따라 prefetch accuracy는 낮아지며 이후 0.4 근처로 포화되는 것을 확인할 수 있다. Endurance 값이 1일 경우 gap이 존재하지 않는 순차 패턴에 대해서만 prefetch를 진행하며 endurance 값이 커질수록 더 큰 gap에 대해서도 prefetch를

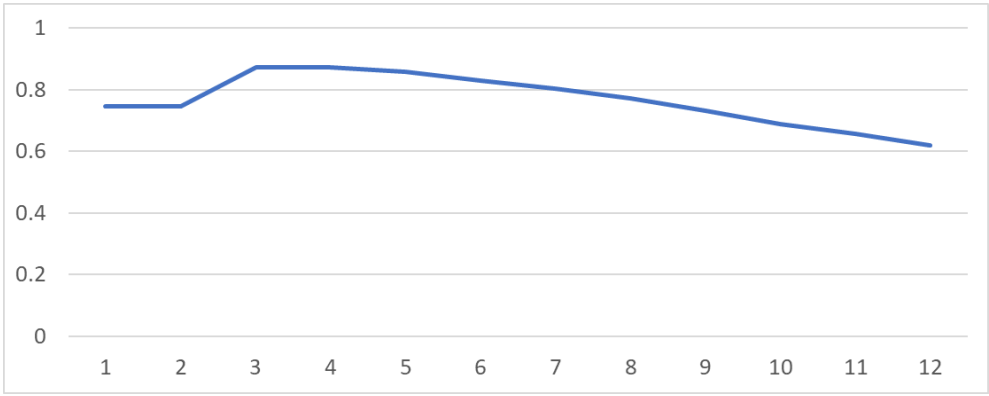



[그림 4-4] Depth가 3일 때 endurance 값에 따른 prefetch 요청 수 변화




[그림 4-5] Depth가 3일 때 endurance 값에 따른 전체 시스템 지연 시간 변화

진행하게 된다. 제안하는 prefetcher의 경우 gap이 존재하지 않는 순차 패턴에 대해서는 하나의 prefetch 요청만 발행하지만 gap이 존재하는 순차 패턴에 대해서는 gap을 메우기 위해 필수적으로 2개 이상의 prefetch 요청을 발행한다. [그림 4-4]에서 확인할 수 있듯 endurance 값이 커짐에 따라 불필요한 prefetch 요청이 증가하여 prefetch accuracy는 낮아진다. 지역성이 큰 PCM workload 특성 상 gap이 큰 순차 패턴 비율이 상대적으로 적다. Endurance 값이 충분히 클 경우

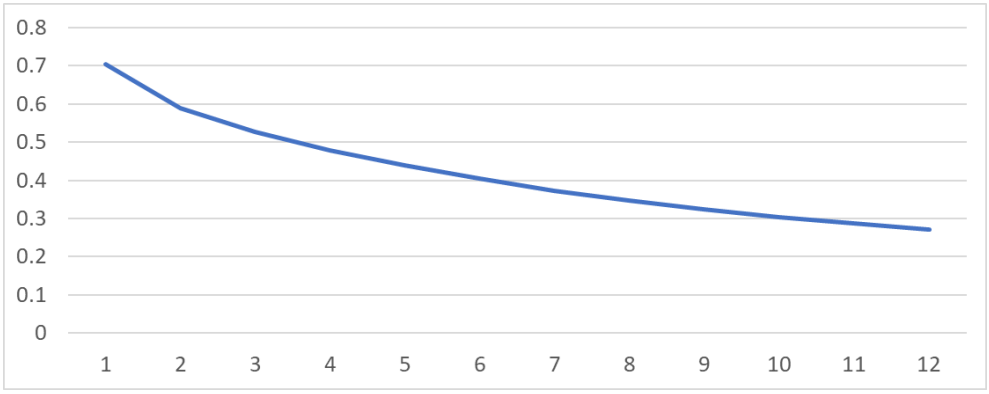



[그림 4-6] Endurance가 3일 때 depth 값에 따른 prefetch coverage 변화

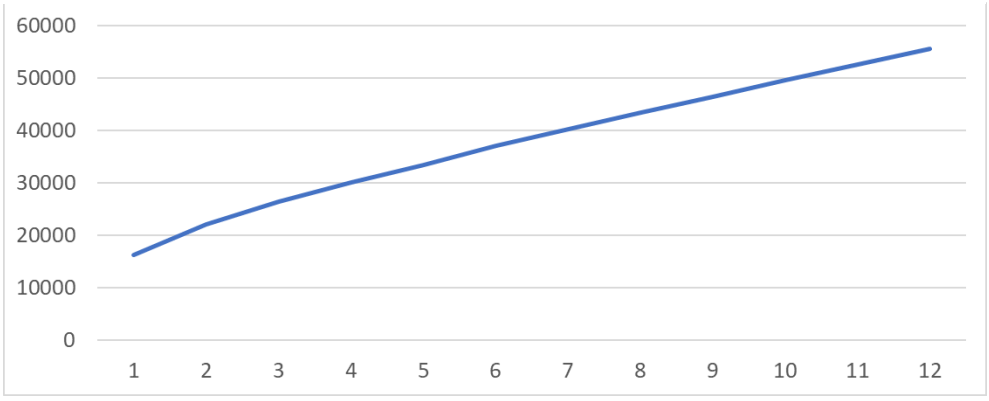
[그림 4-4]에서 확인할 수 있듯 endurance 값 변화에 따른 prefetch 요청 수 변화는 감소하며 이에 따라 prefetch accuracy는 0.4 근처로 포화된다.

[그림 4-5]은 depth 값이 3일 때 endurance 값에 따른 전체 시스템 지연 시간이다. Endurance 값이 작을 경우 낮은 prefetch coverage로 인해 메모리 접근 수가 많아져 높은 전체 시스템 지연 시간을 가지는 것을 확인할 수 있다. Endurance 값이 3에서 5 사이일 경우 비슷한 수준의 prefetch coverage를 가져 비슷한 수준의 전체 시스템 지연 시간을 가진다. 이후 endurance 값이 커짐에 따라 prefetch coverage가 다시 낮아져 전체 시스템 지연 시간이 높아지는 것을 확인할 수 있다.

[그림 4-6]은 endurance 값이 3일 때 depth 값에 따른 prefetch coverage 변화이다. Depth 값이 작을 경우 낮은 prefetch coverage 값을 가지며 depth 값이 3일 경우 가장 높은 prefetch coverage를 가진다. 이후 depth가 커질수록 prefetch coverage는 낮아진다. Depth 값이 작을 경우 PCM workload에서 발생하는, gap이 존재하는 순차

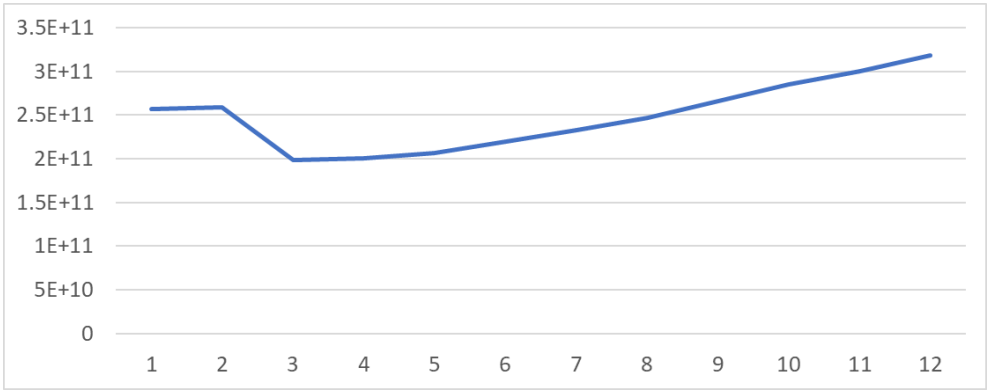



[그림 4-7] Endurance가 3일 때 depth 값에 따른 prefetch accuracy 변화




[그림 4-8] Endurance가 3일 때 depth 값에 따른 prefetch 요청 수 변화

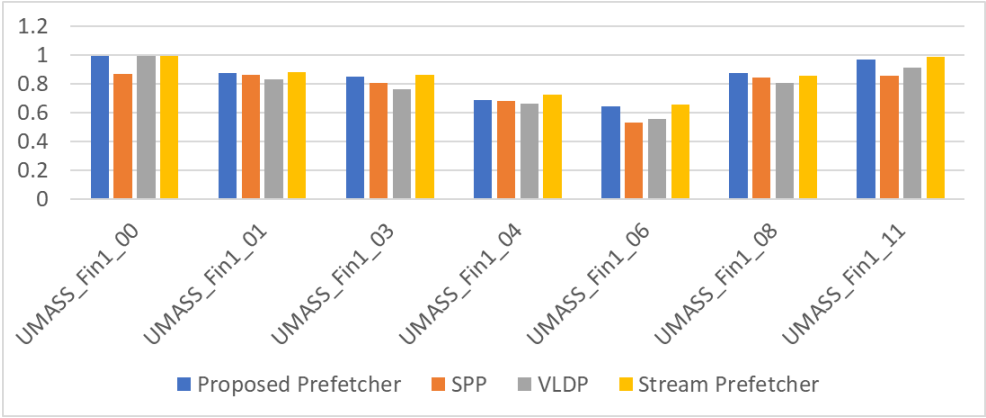
패턴에 대해 성공적인 prefetch를 진행할 수 있는 가능성이 감소한다. 즉 PCM controller에서 주소 변환 데이터를 요구하기 전에 해당 데이터를 prefetch하지 못할 가능성이 높아진다. 이에 따라 높은 prefetch coverage를 기대하기 어렵다. Depth가 지나치게 커질 경우 평균적으로 더 많은 수의 불필요한 prefetch 요청이 발행되어 유용한 데이터가 prefetch buffer에서 방출된다. 이에 따라 더 많은 prefetch를 진행했음에도 낮은 prefetch coverage를 가지게 된다.




[그림 4-9] Endurance가 3일 때 depth 값에 따른 전체 시스템 지연 시간 변화

[그림 4-7]은 endurance 값이 3일 때 depth 값에 따른 prefetch accuracy 변화이다. Depth 값이 커짐에 따라 prefetch accuracy가 낮아지는 것을 확인할 수 있다. Prefetcher에 접근하는 주소 값을 통해 해당 패턴의 길이를 예측하는 것에는 어려움이 따른다. 이에 따라 제안한 prefetcher와 같이 depth를 통해 prefetch를 진행하는 방식의 경우 패턴의 마지막 주소 값에 대해서 불필요한 요청을 발행한다. [그림 4-8]에서 확인할 수 있듯 depth가 커질수록 prefetch 요청 수가 증가한다. Endurance 값의 변화에 따른 prefetch 요청 수의 경우 endurance 값이 증가함에 따라 prefetch 요청 수 변화가 감소하는 양상을 보였지만 depth 값의 변화에 따른 prefetch 요청 수의 경우 depth 값과 무관하게 지속적으로 prefetch 요청 수가 증가하는 양상을 보였다. 이에 따라 depth에 따른 prefetch accuracy의 경우 특정 값으로 포화되지 않고 지속적으로 감소한다.

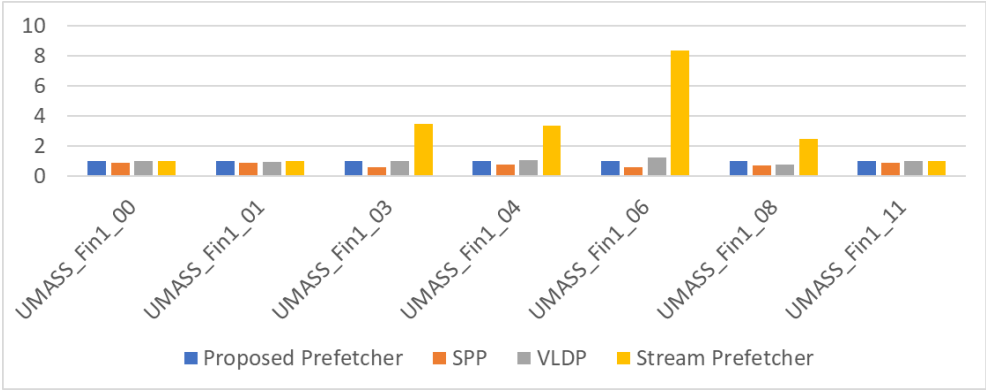
[그림 4-9]은 endurance가 3일 때 depth 값에 따른 전체 시스템 지연 시간이다. Depth가 작을 경우 낮은 prefetch coverage로 인해




[그림 4-10] Prefetcher 종류에 따른 prefetch coverage

메모리 접근 수가 많아져 높은 전체 시스템 지연 시간을 가지는 것을 확인할 수 있다. Depth가 3일 때 가장 높은 prefetch coverage를 가져 가장 낮은 수준의 전체 시스템 지연 시간을 가진다. 이후 depth 값이 커짐에 따라 prefetch coverage가 다시 낮아져 전체 시스템 지연 시간이 높아지는 것을 확인할 수 있다. [그림 4-5]와 [그림 4-9]를 통해 알 수 있듯 전체 시스템 지연 시간 변화의 경우 prefetch coverage와 높은 상관관계를 가지며 prefetch accuracy와는 상대적으로 낮은 상관관계를 가진다.

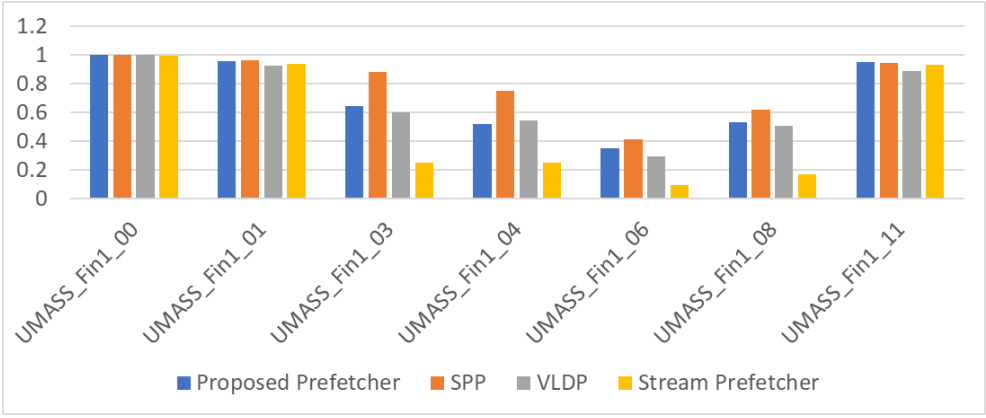
다음으로는 prefetcher의 종류에 따른 실험을 진행하였다. [그림 4-10]은 prefetcher 종류에 따른 prefetch coverage를 나타낸다. 제안한 prefetcher의 경우 평균 84%, SPP의 경우 평균 78%, VLDP의 경우 평균 79%, stream prefetcher의 경우 평균 85%의 prefetch coverage를 가진다. 제안한 prefetcher의 경우 stream table을 통해 순방향 및 역 방향 순차 패턴에 대해 성공적인 prefetch를 진행할 수 있다. Stream prefetcher와 다르게 순차 패턴의 진행 방향에 대한 학습이 필수적이므로 길이가 2인 패턴들에 대해 prefetch hit을




[그림 4-11] Prefetcher 종류에 따른

정규화 된 발행 prefetch 요청 수

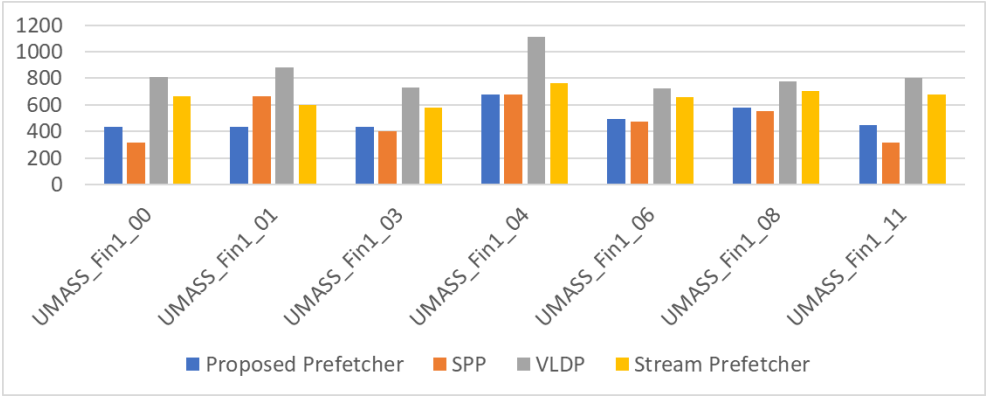
기대하기 어렵다. 하지만 SPP, VLDP와 다르게 두 번째 비 순차 패턴, 즉 gap이 존재하는 순차 패턴이 발생하는 경우 이전에 발생한 패턴과의 연관성을 고려한 prefetch를 진행하여 추가적인 prefetch 요청을 발행한다. [그림 4-11]을 통해 확인할 수 있듯 stream prefetcher 대비 적은 수, SPP, VLDP 대비 많은 수의 prefetch 요청을 발행한다. 이전에 발생한 패턴과의 연관성을 고려하여 prefetch 하므로 gap이 존재하는 순차 패턴에 대해 추가적인 prefetch hit을 기대할 수 있다. 이에 따라 SPP, VLDP 대비 높은 prefetch coverage, stream prefetcher 대비 낮은 prefetch coverage를 가진다. SPP의 경우 접근 주소의 tag에 따른 delta 패턴 분석을 통해 prefetch를 진행한다. 이에 따라 순 방향 및 역 방향 순차 패턴에 대해 성공적인 prefetch를 진행할 수 있다. 하지만 gap이 존재하는 순차 패턴이 발생하는 경우, delta 패턴은 높은 확률로 SPP의 pattern table에 학습되지 않은 패턴이다. 이에 따라 어떠한 prefetch도 진행하지 않는다. [그림 4-11]을 통해 확인할 수 있듯 SPP의 경우 다른 prefetch 알고리즘 대비 적은 수의 prefetch를 진행하는 것을 확인할 수 있다. 적은 수의




[그림 4-12] Prefetcher 종류에 따른 prefetch accuracy

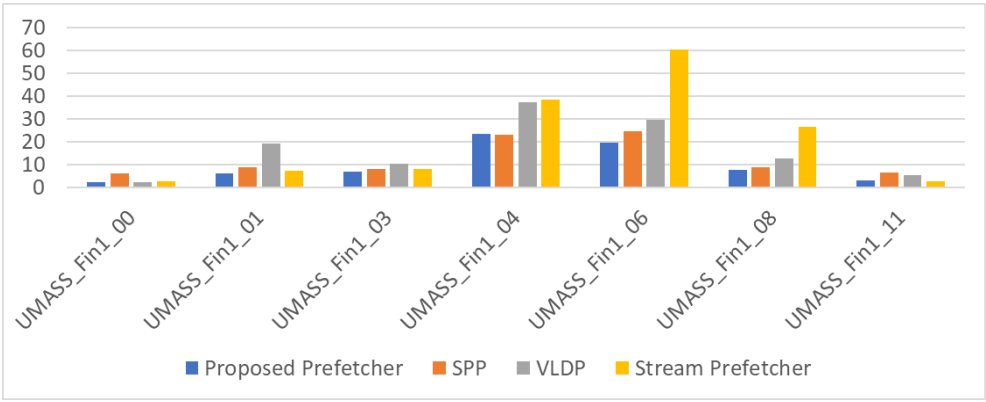
prefetch 요청을 발행함에 따라 높은 prefetch coverage를 기대하기 어렵다. VLDP의 경우 SPP와 마찬가지로 접근 주소의 tag에 따른 delta 패턴 분석을 통해 prefetch를 진행한다. 이에 따라 순 방향 및 역 방향 순차 패턴에 대해 성공적인 prefetch를 진행할 수 있다. VLDP의 경우 SPP와는 다르게 해당하는 delta 패턴이 학습되지 않은 경우에도 offset에 해당하는 delta 값이 학습되었을 경우 offset prediction table을 통해 prefetch를 진행할 수 있다. [그림 4-11]을 통해 확인할 수 있듯 SPP 대비 많은 수의 prefetch를 진행한다. 하지만 비 순차 패턴의 경우 offset prediction table을 통한 예측이 어려워 상대적으로 낮은 prefetch coverage를 가진다. Stream prefetcher의 경우 패턴에 대한 훈련이 불필요하다. 접근하는 모든 주소들에 대해 순 방향으로 depth만큼 prefetch를 진행하게 된다. [그림 4-11]을 통해 확인할 수 있듯 다른 prefetcher 대비 많은 수의 prefetch 요청을 발행한다. 길이가 2인 패턴들에 대해서도 prefetch hit을 기대할 수 있으며 이에 따라 가장 높은 prefetch coverage를 기대할 수 있다.

[그림 4-12]는 prefetcher 종류에 따른 prefetch accuracy를




[그림 4-13] Prefetcher 종류에 따른 최대 주소 변환 시간

나타낸다. 제안한 prefetcher의 경우 평균 70%, SPP의 경우 평균 79%, VLDP의 경우 평균 67%, stream prefetcher의 경우 평균 51%의 prefetch coverage를 가진다. 제안한 prefetcher의 경우 depth를 통한 prefetch 방식으로 불필요한 prefetch 요청이 많이 발생하여 SPP 대비 상대적으로 낮은 prefetch accuracy를 가진다. 하지만 stream prefetcher와는 다르게 해당하는 주소의 패턴 방향성이 학습되기 전에는 prefetch를 진행할 수 없기 때문에 stream prefetcher 대비 상대적으로 높은 prefetch accuracy를 가진다. SPP의 경우 delta 패턴에 대한 충분한 학습이 이루어지기 전까지 prefetch를 진행할 수 없다. 즉 랜덤 패턴이 발생할 경우 이에 대한 prefetch를 진행하지 않으며 이에 따라 가장 높은 prefetch accuracy를 가진다. VLDP의 경우 delta 패턴이 학습되지 않았을 경우 offset을 통한 예측을 진행한다. 하지만 비 순차 패턴들의 경우 offset을 통한 예측이 어려워 SPP 대비 상대적으로 낮은 prefetch accuracy를 가진다. Stream prefetcher의 경우 접근하는 모든 주소들에 대해 순 방향으로 depth만큼 prefetch를 진행한다. 길이가 짧은 패턴들에 대해 불필요한 prefetch 요청이 많이 발행되고 이에 따라 가장 낮은 prefetch



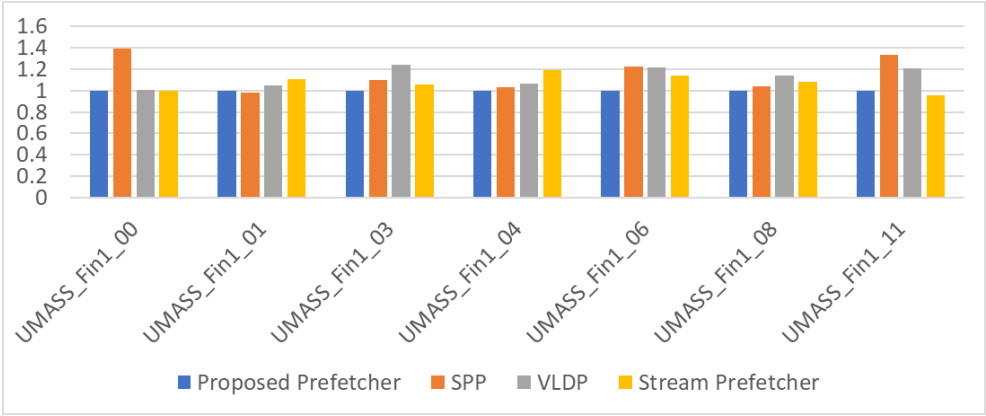

[그림 4-14] Prefetcher 종류에 따른 평균 주소 변환 시간

accuracy를 가진다.

[그림 4-13]은 prefetcher 종류에 따른 최대 주소 변환 시간이다. 제안한 prefetcher의 경우 상대적으로 낮은 최대 주소 변환 시간을 가지며 SPP의 경우 가장 낮은 최대 주소 변환 시간을 가진다. VLDP의 경우 가장 높은 최대 주소 변환 시간을 가지며 stream prefetcher의 경우 상대적으로 높은 최대 주소 변환 시간을 가진다. 최대 주소 변환 시간의 경우 prefetch coverage가 높을수록, prefetch accuracy가 높을수록, 발행 prefetch 요청 수가 적을수록 낮은 값을 가진다. 제안한 prefetcher의 경우 높은 prefetch coverage와 prefetch accuracy를 가지고 있을 뿐 아니라 발행 prefetch 요청 수 또한 많지 않아 낮은 최대 주소 변환 시간을 가진다. SPP의 경우 prefetch coverage가 낮지만 매우 높은 prefetch accuracy를 가지고 있으며 발행 prefetch 요청 수 또한 매우 작기 때문에 가장 낮은 최대 주소 변환 시간을 가진다. VLDP의 경우 낮은 prefetch coverage, prefetch accuracy를 가지며 발행 prefetch 요청 수 또한 높기 때문에 높은 최대 주소 변환 시간을 가진다. Stream prefetcher의 경우 매우 낮은 prefetch accuracy와 매우 높은 발행 prefetch 요청 수를 가지지만 매우 높은

prefetch coverage로 인해 VLDP 대비 작은 수준의 최대 주소 변환 시간을 가진다.

[그림 4-14]은 prefetcher 종류에 따른 평균 주소 변환 시간이다. 제안한 prefetcher의 경우 주소 변환을 위해 평균 9.8 cycle이 소요되었으며, SPP의 경우 평균 12.3 cycle, VLDP의 경우 평균 16.6 cycle, stream prefetcher의 경우 평균 20.8 cycle이 소요되었다. UMASS Financial 1_00 workload와 UMASS Financial 1_11 workload와 같이 순차 패턴이 주를 이루는 workload의 경우 4 종류의 prefetcher 모두 낮은 평균 주소 변환 시간을 가지는 반면 UMASS Financial 1_04 workload와 UMASS Financial 1_06 workload와 같이 랜덤 패턴이 다수 존재하는 workload들의 경우 4 종류의 prefetcher 모두 높은 평균 주소 변환 시간을 가진다. Prefetch buffer에서 hit이 발생할 경우 1cycle 내에 주소 변환 데이터를 받아올 수 있으며 이에 따라 2 cycle 내에 주소 변환 요청을 처리할 수 있다. 반면 miss가 발생할 경우 [그림 4-13]에서 확인할 수 있듯 1000 cycle 이상 소요되기도 한다. 평균 주소 변환 시간 경우 주소 변환 요청들의 주소 변환 시간의 산술평균을 통해 계산한다. 이에 따라 평균 주소 변환 시간은 prefetch coverage 및 최대 주소 변환 시간에 영향을 받는다. 제안한 prefetcher의 경우 높은 prefetch coverage를 가지고 있으며 낮은 최대 주소 변환 시간을 가진다. 이에 따라 가장 낮은 평균 주소 변환 시간을 가진다. SPP의 경우 낮은 prefetch coverage를 가지고 있지만 최대 주소 변환 시간이 가장 낮아 상대적으로 낮은 평균 주소 변환 시간을 가진다. VLDP의 경우 매우 낮은 prefetch coverage를 가지며 최대 주소 변환 시간 또한 매우 커 높은 평균 주소 변환 시간을 가진다. Stream prefetcher의 경우 높은 prefetch coverage를 가지지만

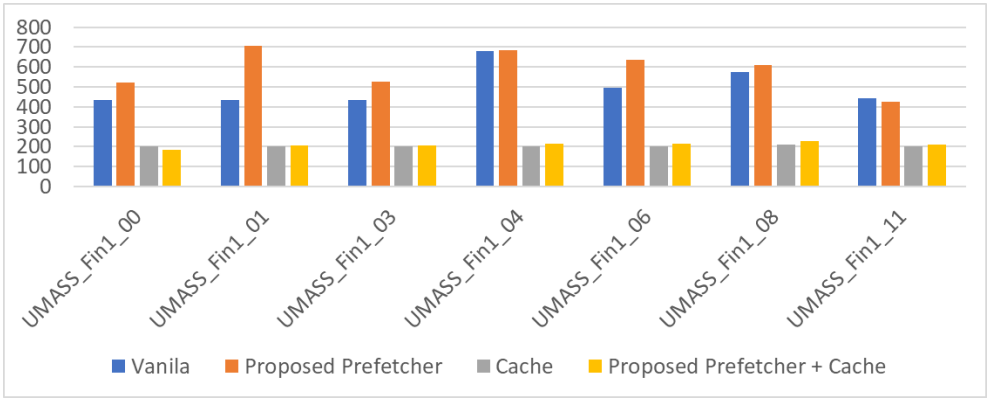



[그림 4-15] Prefetcher 종류에 따른

정규화 된 전체 시스템 지연 시간

높은 최대 주소 변환 시간을 가져 매우 높은 평균 주소 변환 시간을 가진다.

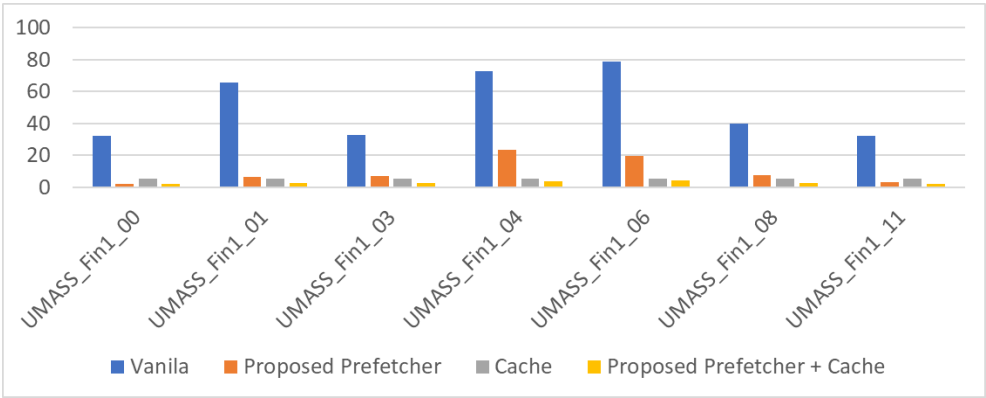
Stream prefetcher의 경우 VLDP 대비 높은 prefetch coverage를 가지며 낮은 최대 주소 변환 시간을 가지지만 더 높은 평균 주소 변환 시간을 가짐을 확인할 수 있다. 특히 UMASS Financial workload 1_06과 UMASS Financial workload 1_08에서 다른 prefetcher 대비 매우 높은 평균 주소 변환 시간을 가진다. UMASS Financial workload 1_06의 경우 랜덤 패턴이 다수 존재하며 Financial workload 1_08의 경우 역방향 순차 패턴이 다수 존재한다. Stream prefetcher의 경우 랜덤 패턴 혹은 역 방향 순차 패턴에 대해서도 순 방향으로 depth만큼 prefetch를 진행한다. 반면 VLDP의 경우 랜덤 패턴 및 역 방향 순차 패턴에 대해 offset prediction table을 통해 하나의 prefetch를 진행한다. 즉 stream prefetcher의 경우 prefetch buffer miss 발생 시 VLDP 대비 높은 단위 시간 당 메모리 접근 수를 가진다. 이에 따라 높은 prefetch coverage를 가지고 있음에도 VLDP 대비 높은 prefetch




[그림 4-16] Prefetcher 유무 및 cache 유무에 따른 최대 주소 변환 시간

miss penalty를 가져 높은 평균 주소 변환 시간을 가진다.

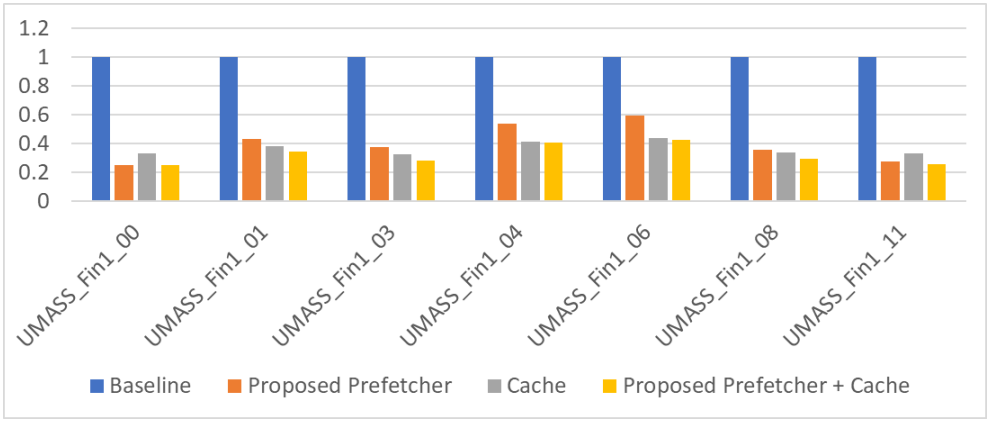
[그림 4-15]는 prefetcher 종류에 따른 정규화 된 전체 시스템 지연 시간이다. 제안한 prefetcher의 경우 SPP 대비 15%, VLDP 대비 13%, stream prefetcher 대비 7%의 성능 향상을 보인다. 제안한 prefetcher의 경우 depth와 endurance를 사용해 gap이 존재하는 순차 패턴에 대해 prefetch를 진행할 수 있어 SPP, VLDP 대비 높은 prefetch coverage를 가진다. 또한 stream prefetcher와 다르게 랜덤 패턴에 대해 prefetch를 진행하지 않으며 역 방향 순차 패턴에 대해서도 알맞은 prefetch를 진행할 수 있어 높은 prefetch accuracy를 가진다. 높은 prefetch coverage 및 prefetch accuracy로 낮은 평균 주소 변환 시간을 가지며 이에 따라 전체 시스템 지연 시간 또한 감소함을 확인할 수 있다. VLDP와 stream prefetcher의 경우 SPP 대비 높은 평균 주소 변환 시간을 가지지만 전체 시스템 지연 시간의 경우 SPP 대비 낮은 값을 가지는 것을 확인할 수 있다. VLDP와 stream prefetcher의 경우 [그림 4-10]을 통해 확인할 수 있듯 SPP대비 높은 prefetch coverage를 가진다. 높은 prefetch




[그림 4-17] Prefetcher 유무 및 cache 유무에 따른 평균 주소 변환 시간

coverage로 인해 prefetch buffer miss 발생 빈도가 낮아지며 이에 따라 memory latency hiding이 발생한다. 따라서 낮은 시스템 지연 시간을 가진다.

[그림 4-16]은 제안한 prefetcher 유무 및 cache 유무에 따른 최대 주소 변환 시간이다. Prefetcher만 사용하였을 경우 최대 주소 변환 시간은 19% 증가하였고 cache만 사용하였을 경우 59% 감소하였으며 prefetcher와 cache를 모두 사용한 경우 57% 감소하였다. Prefetcher를 사용할 경우 prefetch buffer에서 miss가 발생할 경우 depth만큼 prefetch를 진행한다. Prefetch buffer hit을 통해 메모리 접근 수를 줄일 수 있지만 지속적인 prefetch buffer miss 발생은 다수의 prefetch 요청을 발행하여 일부 주소 변환 요청을 지연시킨다. 이에 따라 baseline 대비 높은 최대 주소 변환 시간을 가진다. Cache만 사용할 경우 prefetcher만 사용하였을 경우와 다르게 추가적인 메모리 접근을 발생시키지 않는다. 이에 따라 baseline 대비 낮은 최대 주소 변환 시간을 가진다. Prefetcher와 cache를 모두 사용하였을 경우 prefetch를 통한 추가적인 메모리 접근이 발생할 수




[그림 4-18] Prefetcher 유무 및 cache 유무에 따른 정규화 된 전체 시스템 지연 시간

있지만 cache를 통한 일부 prefetch buffer miss에 대한 빠른 주소 변환을 통해 앞선 메모리 접근에 의한 지연을 최소화할 수 있다. 이에 따라 baseline 대비 낮은 수준의 최대 주소 변환 시간을 가진다.

[그림 4-17]은 제안한 prefetcher 유무 및 cache 유무에 따른 평균 주소 변환 시간이다. Prefetcher만 사용하였을 경우 평균 주소 변환 시간은 baseline 대비 17% 수준이며, cache만 사용하였을 경우 12% 수준, prefetcher와 cache를 모두 사용할 경우 6% 수준이다. Prefetcher를 사용할 경우 prefetch buffer hit을 통한 메모리 접근 수 감소로 baseline 대비 낮은 평균 주소 변환 시간을 가진다. PCM workload 특성 상 높은 지역성으로 cache를 사용할 경우 또한 baseline 대비 낮은 평균 주소 변환 시간을 가진다. Prefetcher만 사용할 경우 prefetch buffer miss 발생 시 메모리 접근은 필수적이지만 cache를 통해 이를 효과적으로 해결할 수 있다. 또한 cache만 사용할 경우 특정 주소에 대한 초기 접근 시 메모리 접근이 필수적이지만 prefetcher를 통해 이를 효과적으로 해결할 수 있다. 이에 따라

prefetcher와 cache를 모두 사용할 경우 가장 낮은 평균 주소 변환 시간을 가진다.

[그림 4-18]은 제안한 prefetcher 유무 및 cache 유무에 따른 정규화 된 전체 시스템 지연 시간이다. Prefetcher만 사용하였을 경우 baseline 대비 40% 수준, cache만 사용하였을 경우 baseline 대비 36% 수준, prefetcher와 cache를 모두 사용하였을 경우 baseline 대비 32% 수준을 가진다. 빠른 주소 변환으로 인해 전체 시스템 지연이 최소됨을 확인할 수 있다.

# 제 5 장 결 론

성공적인 prefetch를 위해서 접근 주소 패턴에 대한 분석이 선행되어야 한다. PCM workload의 경우 패턴의 길이가 긴 순 방향 순차 패턴과 역 방향 순차 패턴이 주를 이룬다. 일부 비 순차 패턴의 경우 이전에 발생한 패턴 혹은 추후 발생하는 패턴과 연관성을 가지고 있다. 이러한 비 순차 패턴은 tag에 따른 뚜렷한 delta 패턴이 존재하지 않는다. 이에 따라 기존의 prefetch 알고리즘 적용이 어렵고 독자적인 prefetch 알고리즘이 필요하다.

본 연구에서 제안한 prefetcher는 stream table의 사용으로 순차 패턴들에 대해 성공적인 prefetch를 진행할 수 있다. 또한 endurance와 depth parameter를 설정함으로 gap이 존재하는 순차 패턴에 대해서도 성공적인 prefetch를 진행할 수 있다. Prefetch endurance의 경우 낮게 설정할 경우 패턴 지속성이 감소하며 높게 설정할 경우 기존 패턴의 eviction을 유발한다. 또한 prefetch depth의 경우 낮게 설정할 경우, gap이 존재하는 순차 패턴에 대해 성공적인 prefetch를 진행하기 어렵고 높게 설정할 경우 prefetch buffer의 pollution을 유발한다. 이에 따라 적절한 수준의 endurance 값과 depth 값 설정은 필수적이다.

제안한 prefetcher의 경우 기존 prefetch 알고리즘에 비해 높은 prefetch coverage를 보였고 7% 이상의 전체 시스템 성능 향상을 보였다. 또한 prefetch를 진행하지 않았을 경우에 비해 주소 변환 시간을 17% 수준으로 단축시켰으며 전체 시스템 지연시간을 40% 수준으로 단축시켰다.

# 참고 문헌

[1] H. -S. P. Wong *et al.*, "Phase Change Memory," in *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2201-2227, Dec. 2010.

[2] B. C. Lee *et al.*, "Phase-Change Technology and the Future of Main Memory," in *IEEE Micro*, vol. 30, no. 1, pp. 143-143, Jan.-Feb. 2010.

[3] M. K. Qureshi, V. Srinivasan and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," *Proceedings of the 36th annual international symposium on Computer architecture,* 2009, pp. 24-33.

[4] S. P. Vanderweil and D. J. Lilja, "Data prefetch mechanisms," in *ACM Computing Surveys*, vol. 32, no. 2, pp. 174-199, Jun. 2000.

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

[6] J. Kim, S. H. Pugsley, P. V. Gratz, A. L. N. Reddy, C. Wilkerson and Z. Chishti, "Path confidence based lookahead prefetching," *2016*

*49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1-12.

[7] M. Shevgoor, S. Koladiya, R. Balasubramonian, C. Wilkerson, S. H. Pugsley and Z. Chishti, "Efficiently prefetching complex address patterns," *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2015, pp. 141-152.

[8] D. Joseph and D. Grunwald, "Prefetching using Markov predictors," in *IEEE Transactions on Computers*, vol. 48, no. 2, pp. 121-133, Feb. 1999.

[9] D. Wang, B. Ganesh, N. Tuaycharoen, K. Baynes, A. Jaleel, and B. Jacob, "DRAMsim: A Memory System Simulator" in *ACM SIGARCH Computer Architecture News,* vol. 33, no. 4, pp. 100-107, Nov. 2005.

[10] P. Rosenfeld, E. Cooper-Balis and B. Jacob, "DRAMSim2: A Cycle Accurate Memory System Simulator," in *IEEE Computer Architecture Letters*, vol. 10, no. 1, pp. 16-19, Jan.-June. 2011.

[11] S. Li, Z. Yang, D. Reddy, A. Srivastava and B. Jacob, "DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator," in *IEEE Computer Architecture Letters*, vol. 19, no. 2, pp. 106-109, July.-Dec. 2020.

[12] S. Srinath, O. Mutlu, H. Kim and Y. N. Patt, "Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers," *2007 IEEE 13th International Symposium on High Performance Computer Architecture*, 2007, pp. 63-74.

[13] D. Kroft, "Lockup-free instruction fetch/prefetch cache organization," *Proceedings of the 8th annual symposium on Computer Architecture,* 1981, pp. 81-87.

[14] E. Bhatia, G. Chacon, S. Pugsley, E. Teran, P. V. Gratz and D. A. Jimenez, "Perceptron-Based Prefetch Filtering," *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, 2019, pp. 1-13.

[15] T. Alexander and G. Kedem, "Distributed prefetch-buffer/cache design for high performance memory systems," *Proceedings. Second International Symposium on High-Performance Computer Architecture*, 1996, pp. 254-263.

[16] K. J. Nesbit and J. E. Smith, "Data cache prefetching using a global history buffer," in *IEEE Micro*, vol. 25, no. 1, pp. 90-97, Jan.-Feb. 2005.

[17] J. Baer and T. Chen, "An effective on-chip preloading scheme to reduce data access penalty," *Supercomputing '91:Proceedings of*

*the 1991 ACM/IEEE Conference on Supercomputing*, 1991, pp. 176-186.

[18] S. Palacharla and R. E. Kessler, "Evaluating stream buffers as a secondary cache replacement," *Proceedings of 21 International Symposium on Computer Architecture*, 1994, pp. 24-33.

[19] S. H. Pugsley *et al.*, "Sandbox Prefetching: Safe run-time evaluation of aggressive prefetchers," *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, 2014, pp. 626-637.

[20] S. Somogyi, T. F. Wenisch, A. Ailamaki, and B. Falsafi, "Spatio-temporal memory streaming," *in ACM SIGARCH Computer Architecture News,* vol. 37, no. 3, pp. 69-80, Jun. 2009.

[21] J. L. Hennessy and D. A. Patterson, *Computer architecture: A quantitative approach.* San Francisco: Morgan Kaufmann, 2011.

[22] D. A. Patterson and J. L. Hennessy, *Computer organization and design: The hardware/software Interface ARM edition.* Amsterdam: Elsevier, 2017.

## Abstract

# An efficient prefetching algorithm for fast address translation in a PCM Controller

Hyung-Suk Kim
School of Electrical and Computer Engineering
The Graduate School
Seoul National University

DRAM's scaling technology has reached its limit and several next-generation memory technologies have been proposed to replace DRAM. Among them, phase change memory (PCM) stores data as resistance changes through phase changes in matter, enabling effective scaling. To access PCM, PCM controller converts logical address to physical address. When an address conversion request occurs, DRAM is accessed to receive address conversion data. Continuous memory access for address translation causes overall performance degradation. Prefetcher can solve these problems effectively. For PCM workloads, forward sequential patterns and reverse sequential patterns are the main ones. Some non-sequential patterns have associations with sequential patterns that exist within PCM workloads. For these non-sequential patterns, no distinct pattern exists, and successful prefetching cannot be carried out with

existing prefetch algorithms. Prefetcher proposed in this work can efficiently detect sequential patterns by taking a similar approach to stream prefetcher. Furthermore, the endurance and depth parameters allow a prefetch hit to occur for patterns that no distinct pattern exists but are associated with sequential patterns in the workload. The proposed prefetcher showed more than 7% improvement in overall system performance compared to the existing prefetch algorithm. It also reduced address translation time to 17% and overall system latency to 40% compared to the case of not proceeding prefetch.

Keywords: phase change memory, address translation, prefetch, sequential pattern, non-sequential pattern
Student Number: 2019-21676