(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2016년11월30일
(11) 등록번호 10-1681423
(24) 등록일자 2016년11월24일

(51) 국제특허분류(Int. Cl.)
G06F 9/38 (2006.01) *G06F 12/08* (2016.01)
(52) CPC특허분류
G06F 9/3814 (2013.01)
G06F 12/0862 (2013.01)
(21) 출원번호 10-2015-0113161
(22) 출원일자 2015년08월11일
심사청구일자 2015년08월11일
(56) 선행기술조사문헌
KR1020090026333 A
Nesbit, Kyle J., and James E. Smith. "Data cache prefetching using a global history buffer." Software, IEE Proceedings-. IEEE, 2004.

(73) 특허권자
인하대학교 산학협력단
인천광역시 남구 인하로 100, 인하대학교 (용현동)
(72) 발명자
최상방
서울특별시 양천구 목동동로 100, 1312동 404호 (신정동, 목동13단지아파트)
김진혁
인천광역시 부평구 마장로272번길 89, 302동 2001호 (산곡동, 경남아파트)
정용수
서울특별시 구로구 개봉로20길 6, 131동 1504호 (개봉동, 현대아파트1단지)
(74) 대리인
양성보

전체 청구항 수 : 총 5 항 심사관 : 서광훈

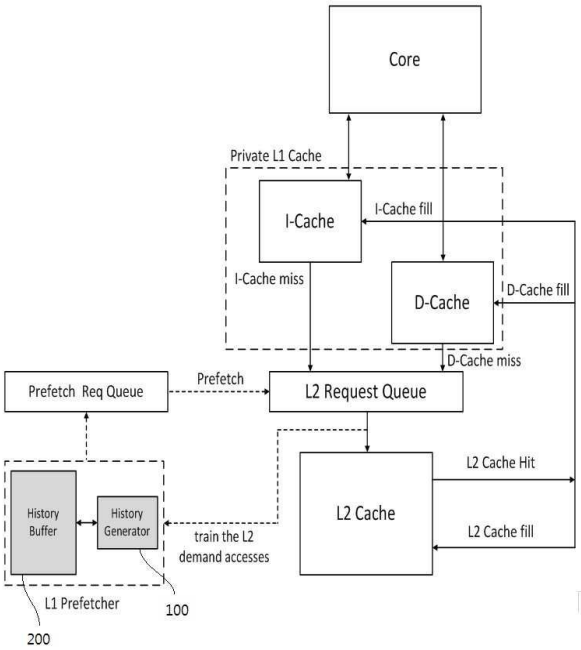
(54) 발명의 명칭 **변위 히스토리 버퍼를 이용한 명령어 및 데이터 프리페치 방법 및 시스템**

(57) 요 약

변위 히스토리 버퍼를 이용한 명령어 및 데이터 프리페치 방법 및 시스템이 제시된다. 히스토리 버퍼를 이용한 명령어 및 데이터 프리페치 방법에 있어서, 캐시 블록에서 캐시 미스가 발생하는 경우, 히스토리 생성기의 트리거 주소 필드에 저장된 트리거 주소에서 상기 캐시 블록의 주소를 검색하는 단계; 상기 히스토리 생성기의 트리

(뒷면에 계속)

대 표 도 - 도4



거 주소 필드에 저장된 트리거 주소가 없는 경우, 상기 캐시 블록의 주소를 저장하여 새로운 히스토리 레코드의 기준이 되는 트리거 주소로 설정하는 단계; 다른 캐시 블록에서 캐시 미스가 발생하는 경우, 상기 히스토리 생성기의 트리거 주소 필드에 저장되어 있는 상기 트리거 주소와의 변위 값을 히스토리 생성기의 변위 필드에 저장하는 단계; 상기 히스토리 생성기의 변위 필드의 엔트리에 변위 값들이 저장되어 저장 공간이 없거나 저장할 수 없는 범위의 캐시 블록이 발생하는 경우, 상기 히스토리 레코드의 생성을 중단하고 상기 엔트리를 히스토리 버퍼에 저장하는 단계; 및 상기 히스토리 생성기의 변위 필드의 범위가 아닌 캐시 블록에서 캐시 미스가 발생하는 경우, 기 저장된 트리거 주소의 블록과 변위 값들을 상기 히스토리 버퍼에 저장하고, 상기 히스토리 생성기의 트리거 주소 필드와 상기 히스토리 생성기의 변위 필드에 저장된 값들을 초기화하는 단계를 포함할 수 있다.

(52) CPC특허분류
G06F 2212/6022 (2013.01)
G06F 2212/6024 (2013.01)

명 세 서

청구범위

청구항 1

변위 히스토리 버퍼를 이용한 명령어 및 데이터 프리페치 방법에 있어서,

캐시 블록에서 캐시 미스가 발생하는 경우, 히스토리 생성기의 트리거 주소 필드에 저장된 트리거 주소에서 상기 캐시 블록의 주소를 검색하는 단계;

상기 히스토리 생성기의 트리거 주소 필드에 저장된 트리거 주소가 없는 경우, 상기 캐시 블록의 주소를 저장하여 새로운 히스토리 레코드의 기준이 되는 트리거 주소로 설정하는 단계;

다른 캐시 블록에서 캐시 미스가 발생하는 경우, 상기 히스토리 생성기의 트리거 주소 필드에 저장되어 있는 상기 트리거 주소와의 변위 값을 히스토리 생성기의 변위 필드에 저장하는 단계;

상기 히스토리 생성기의 변위 필드의 엔트리에 변위 값들이 저장되어 저장 공간이 없거나 저장할 수 없는 범위의 캐시 블록이 발생하는 경우, 상기 히스토리 레코드의 생성을 중단하고 상기 엔트리를 히스토리 버퍼에 저장하는 단계; 및

상기 히스토리 생성기의 변위 필드의 범위가 아닌 캐시 블록에서 캐시 미스가 발생하는 경우, 기 저장된 트리거 주소의 블록과 변위 값들을 상기 히스토리 버퍼에 저장하고, 상기 히스토리 생성기의 트리거 주소 필드와 상기 히스토리 생성기의 변위 필드에 저장된 값들을 초기화하는 단계

를 포함하고,

상기 히스토리 생성기와 상기 히스토리 버퍼 중 적어도 하나 이상은 캐시 미스가 발생한 주소들을 기록하여 히스토리 레코드를 만들어, 프로세서로부터 다시 상기 주소의 요청 시 상기 캐시 블록을 프리페치하는 것

을 특징으로 하는 변위 히스토리 버퍼를 이용한 명령어 및 데이터 프리페치 방법.

청구항 2

제1항에 있어서,

상기 히스토리 버퍼는,

히스토리 버퍼의 트리거 주소 필드와 히스토리 버퍼의 변위 필드로 구성되며 각각의 엔트리에는 상기 히스토리 레코드가 저장되고, 트리거 액세스 필드는 히스토리 버퍼의 인덱스 역할을 하며,

상기 히스토리 레코드와 트리거 액세스 필드의 정보가 일치하고, 트리거 액세스 필드에 일치하는 엔트리가 상기 히스토리 버퍼에 존재하는 경우, 상기 캐시 블록의 주소를 프리페치한 후 상기 히스토리 버퍼의 변위 필드에 저장되어 있는 변위 값을 저장된 순서에 따라 상기 캐시 블록의 주소 값에 더해 프리페치하는 것

을 특징으로 하는 변위 히스토리 버퍼를 이용한 명령어 및 데이터 프리페치 방법.

청구항 3

제2항에 있어서,

상기 히스토리 레코드의 첫 캐시 블록 주소인 상기 히스토리 버퍼의 트리거 주소 블록에 우선순위를 부여하여, 캐시 교체가 요청되는 경우 랜덤(Random) 교체 정책에 의해 상기 우선순위를 부여 받지 못한 상기 히스토리 버퍼의 변위 필드 블록부터 교체하는 것

을 특징으로 하는 변위 히스토리 버퍼를 이용한 명령어 및 데이터 프리페치 방법.

청구항 4

변위 히스토리 버퍼를 이용한 명령어 및 데이터 프리페치 시스템에 있어서,

프로세서가 접근한 프로그램의 순서 정보를 히스토리 레코드로 저장하는 히스토리 생성기(history generator); 및

상기 히스토리 생성기에서 저장된 상기 히스토리 레코드를 저장하는 히스토리 버퍼(history buffer)를 포함하고,

히스토리 생성기의 변위 필드의 엔트리에 변위 값들이 저장되어 저장 공간이 없거나 저장할 수 없는 범위의 캐시 블록이 발생하는 경우, 상기 히스토리 레코드의 생성을 중단하고 상기 엔트리를 상기 히스토리 버퍼에 저장하며,

상기 히스토리 버퍼는,

상기 히스토리 레코드가 저장되는 히스토리 버퍼의 트리거 주소 필드와 히스토리 버퍼의 변위 필드; 및

상기 히스토리 버퍼의 인덱스 역할을 하는 트리거 액세스 필드를 포함하고,

상기 히스토리 레코드와 트리거 액세스 필드의 정보가 일치하고 트리거 액세스 필드에 일치하는 엔트리가 상기 히스토리 버퍼에 존재하는 경우, 상기 캐시 블록의 주소를 프리페치한 후 상기 히스토리 버퍼의 변위 필드에 저장되어 있는 변위 값을 저장된 순서에 따라 상기 캐시 블록의 주소 값에 더해 프리페치하며, 상기 히스토리 레코드의 첫 캐시 블록 주소인 상기 히스토리 버퍼의 트리거 주소 필드의 블록에 우선순위를 부여하여, 캐시 블록의 교체가 요청되는 경우 랜덤(Random) 교체 정책에 의해 상기 우선순위를 부여 받지 못한 상기 히스토리 버퍼의 변위 필드의 캐시 블록부터 교체하는 것

을 특징으로 하는 변위 히스토리 버퍼를 이용한 명령어 및 데이터 프리페치 시스템.

청구항 5

제4항에 있어서,

상기 히스토리 생성기는,

캐시 미스가 발생한 캐시 블록의 주소를 저장하는 히스토리 생성기의 트리거 주소(trigger address) 필드; 및

상기 히스토리 생성기의 트리거 주소 필드와 다음 캐시 미스가 발생한 캐시 블록의 주소의 변위 값을 저장하며, 비트 수에 따라 저장할 수 있는 상기 변위 값의 범위가 설정되는 상기 히스토리 생성기의 변위(displacement) 필드

를 포함하는 변위 히스토리 버퍼를 이용한 명령어 및 데이터 프리페치 시스템.

청구항 6

삭제

청구항 7

삭제

발명의 설명

기 술 분 야

[0001] 아래의 실시예들은 변위 히스토리 버퍼를 이용한 명령어 및 데이터 프리페치 방법 및 시스템에 관한 것이다. 더욱 상세하게는, 사용될 데이터를 정확하게 예측하여 프리페치하는 변위 히스토리 버퍼를 이용한 명령어 및 데이터 프리페치 방법 및 시스템에 관한 것이다.

배 경 기 술

[0002] 현 세대 프로세서는 동작을 실행하는데 수 사이클이 소모되는데 비해 메모리에 접근하기 위해서는 수백 사이클을 기다려야 하기 때문에 처리속도가 저하된다. 약 100배 정도의 속도 차이 때문에 메모리에서 병목현상이 발생된다. 이에 프로세서와 메모리 사이의 속도 차이를 줄이기 위해 메모리를 계층화하여 프로세서와 메모리 사이에 캐시 메모리를 추가할 수 있다. 또한 메모리 레이턴시(Latency)를 최소화하기 위해 프리페치(Prefetch)

기법을 사용할 수 있다.

[0003] 프리페치 기법은 프로세서가 필요한 데이터를 예측해 미리 가져오는 방법이다. 처음에는 캐시에 어떤 블록도 저장되어있지 않기 때문에 가장 먼저 접근하는 블록에서 항상 캐시 미스가 발생된다. 이를 콜드 미스라 하며, 캐시 블록의 크기를 크게 하여 한 블록에 많은 양의 데이터를 저장하거나 명령어가 실행되기 전 프리페치를 통해 미리 블록을 가져와 캐시 미스를 줄일 수 있다. 프리페치는 학습 과정을 통해 미래에 사용될 데이터를 읽어오기 때문에 예측이 틀릴 수도 있다. 예측이 틀리는 경우에는 소중한 메모리 대역폭을 낭비하는 결과를 초래한다. 따라서 프리페치를 요청할 때 앞으로 사용될 데이터를 정확하게 예측하여 적절한 시간에 불러오는 것이 중요하다.

[0004] 한국공개특허 10-2007-0108209호는 이러한 명령 프리페치 메카니즘에 관한 것으로, 조건 분기 명령에 대한 복수의 가중된 분기 예측값 중 하나를 생성하는 분기 예측 메카니즘을 갖는 프로세서에서의 명령 프리페치 방법에 관한 기술을 기재하고 있다.

발명의 내용

해결하려는 과제

[0005] 실시예들은 변위 히스토리 버퍼를 이용한 명령어 및 데이터 프리페치 방법 및 시스템에 관하여 기술하며, 보다 구체적으로 프리페치의 요청 시 사용될 데이터를 정확하게 예측하여 적절한 시간에 불러오는 명령어 및 데이터 프리페치 기술을 제공한다.

[0006] 실시예들은 캐시를 이용하는 프로세서 환경에서 명령어 데이터 처리 성공율을 높임으로써, 프로세서 처리 성능을 증가시키면서 캐시 메모리 구조를 단순화하여 전력, 비용 등을 절감할 수 있는 변위 히스토리 버퍼를 이용한 명령어 및 데이터 프리페치 방법 및 시스템을 제공하는데 있다.

과제의 해결 수단

[0007] 일 실시예에 따른 변위 히스토리 버퍼를 이용한 명령어 및 데이터 프리페치 방법에 있어서, 캐시 블록에서 캐시 미스가 발생하는 경우, 히스토리 생성기의 트리거 주소 필드에 저장된 트리거 주소에서 상기 캐시 블록의 주소를 검색하는 단계; 상기 히스토리 생성기의 트리거 주소 필드에 저장된 트리거 주소가 없는 경우, 상기 캐시 블록의 주소를 저장하여 새로운 히스토리 레코드의 기준이 되는 트리거 주소로 설정하는 단계; 다른 캐시 블록에서 캐시 미스가 발생하는 경우, 상기 히스토리 생성기의 트리거 주소 필드에 저장되어 있는 상기 트리거 주소와의 변위 값을 히스토리 생성기의 변위 필드에 저장하는 단계; 상기 히스토리 생성기의 변위 필드의 엔트리에 변위 값들이 저장되어 저장 공간이 없거나 저장할 수 없는 범위의 캐시 블록이 발생하는 경우, 상기 히스토리 레코드의 생성을 중단하고 상기 엔트리를 히스토리 버퍼에 저장하는 단계; 및 상기 히스토리 생성기의 변위 필드의 범위가 아닌 캐시 블록에서 캐시 미스가 발생하는 경우, 기 저장된 트리거 주소의 블록과 변위 값들을 상기 히스토리 버퍼에 저장하고, 상기 히스토리 생성기의 트리거 주소 필드와 상기 히스토리 생성기의 변위 필드에 저장된 값들을 초기화하는 단계를 포함하고, 상기 히스토리 생성기와 상기 히스토리 버퍼 중 적어도 하나 이상은 캐시 미스가 발생한 주소들을 기록하여 히스토리 레코드를 만들어, 프로세서로부터 다시 상기 주소의 요청 시 상기 캐시 블록을 프리페치한다.

[0008] 여기서, 상기 히스토리 버퍼는 히스토리 버퍼의 트리거 주소 필드와 히스토리 버퍼의 변위 필드로 구성되며 각각의 엔트리에는 상기 히스토리 레코드가 저장되고, 트리거 액세스 필드는 히스토리 버퍼의 인덱스 역할을 하며, 상기 히스토리 레코드와 트리거 액세스 필드의 정보가 일치하고, 트리거 액세스 필드에 일치하는 엔트리가 상기 히스토리 버퍼에 존재하는 경우, 상기 캐시 블록의 주소를 프리페치한 후 상기 히스토리 버퍼의 변위 필드에 저장되어 있는 변위 값을 저장된 순서에 따라 상기 캐시 블록의 주소 값에 더해 프리페치할 수 있다.

[0009] 상기 히스토리 레코드의 첫 캐시 블록 주소인 상기 히스토리 버퍼의 트리거 주소 블록에 우선순위를 부여하여, 캐시 교체가 요청되는 경우 랜덤(Random) 교체 정책에 의해 상기 우선순위를 부여 받지 못한 상기 히스토리 버퍼의 변위 필드 블록부터 교체할 수 있다.

[0010] 다른 실시예에 따른 변위 히스토리 버퍼를 이용한 명령어 및 데이터 프리페치 시스템에 있어서, 프로세서가 접근한 프로그램의 순서 정보를 히스토리 레코드로 저장하는 히스토리 생성기(history generator); 및 상기 히스토리 생성기에서 저장된 상기 히스토리 레코드를 저장하는 히스토리 버퍼(history buffer)를 포함하고, 히스토리 생성기의 변위 필드의 엔트리에 변위 값들이 저장되어 저장 공간이 없거나 저장할 수 없는 범위의 캐시 블록

이 발생하는 경우, 상기 히스토리 레코드의 생성을 중단하고 상기 엔트리를 상기 히스토리 버퍼에 저장할 수 있다.

[0011] 여기서, 상기 히스토리 생성기는, 캐시 미스가 발생한 캐시 블록의 주소를 저장하는 히스토리 생성기의 트리거 주소(trigger address) 필드; 및 상기 히스토리 생성기의 트리거 주소 필드와 다음 캐시 미스가 발생한 캐시 블록의 주소의 변위 값을 저장하며, 비트 수에 따라 저장할 수 있는 상기 변위 값의 범위가 설정되는 상기 히스토리 생성기의 변위(displacement) 필드를 포함할 수 있다.

[0012] 상기 히스토리 버퍼는, 상기 히스토리 레코드가 저장되는 히스토리 버퍼의 트리거 주소 필드와 히스토리 버퍼의 변위 필드; 및 상기 히스토리 버퍼의 인덱스 역할을 하는 트리거 액세스 필드를 포함하고, 상기 히스토리 레코드와 트리거 액세스 필드의 정보가 일치하고, 트리거 액세스 필드에 일치하는 엔트리가 상기 히스토리 버퍼에 존재하는 경우, 상기 캐시 블록의 주소를 프리페치한 후 상기 히스토리 버퍼의 변위 필드에 저장되어 있는 변위 값을 저장된 순서에 따라 상기 캐시 블록의 주소 값에 더해 프리페치할 수 있다.

[0013] 상기 히스토리 레코드의 첫 캐시 블록 주소인 상기 히스토리 버퍼의 트리거 주소 필드의 블록에 우선순위를 부여하여, 캐시 블록의 교체가 요청되는 경우 랜덤(Random) 교체 정책에 의해 상기 우선순위를 부여 받지 못한 상기 히스토리 버퍼의 변위 필드의 캐시 블록부터 교체할 수 있다.

발명의 효과

[0014] 실시예들에 따르면 프리페치의 요청 시 사용될 데이터를 정확하게 예측하여 적절한 시간에 불러오는 명령어 및 데이터 프리페치 기술을 제공할 수 있다.

[0015] 실시예들에 따르면 캐시를 이용하는 프로세서 환경에서 명령어 데이터 처리 성공율을 높임으로써, 프로세서 처리 성능을 증가시키면서 캐시 메모리 구조를 단순화하여 전력, 비용 등을 절감할 수 있는 변위 히스토리 버퍼를 이용한 명령어 및 데이터 프리페치 방법 및 시스템을 제공할 수 있다.

도면의 간단한 설명

[0016] 도 1은 일 실시예에 따른 변위를 이용한 히스토리 레코드 생성 과정을 설명하기 위한 도면이다.

도 2는 일 실시예에 따른 히스토리 버퍼와 예측 과정을 설명하기 위한 도면이다.

도 3은 일 실시예에 따른 트리거 블록에 우선순위를 부여하는 랜덤 교체 정책의 동작을 설명하기 위한 도면이다.

도 4는 일 실시예에 따른 프리페처 전체 시스템을 설명하기 위한 도면이다.

도 5는 일 실시예에 따른 L1 데이터 캐시의 미스율을 나타낸 그래프이다.

도 6은 일 실시예에 따른 L1 명령어 캐시의 미스율을 나타낸 그래프이다.

도 7은 일 실시예에 따른 8 비트 변위 필드의 L1 데이터 캐시의 미스율 변화를 나타낸 그래프이다.

도 8은 일 실시예에 따른 4 비트 변위 필드의 L1 데이터 캐시의 미스율 변화를 나타낸 그래프이다.

도 9는 일 실시예에 따른 8 비트 변위 필드의 L1 명령어 캐시의 미스율 변화를 나타낸 그래프이다.

도 10은 일 실시예에 따른 4 비트 변위 필드의 L1 명령어 캐시의 미스율 변화를 나타낸 그래프이다.

도 11 및 도 13는 일 실시예에 따른 캐시 교체 정책에 따른 L1 데이터 캐시의 미스율을 나타낸 그래프이다.

도 14는 일 실시예에 따른 IPC 스피드업 성능 비교를 나타낸 그래프이다.

발명을 실시하기 위한 구체적인 내용

[0017] 이하, 첨부된 도면을 참조하여 실시예들을 설명한다. 그러나, 기술되는 실시예들은 여러 가지 다른 형태로 변형될 수 있으며, 본 발명의 범위가 이하 설명되는 실시예들에 의하여 한정되는 것은 아니다. 또한, 여러 실시예들은 당해 기술분야에서 평균적인 지식을 가진 자에게 본 발명을 더욱 완전하게 설명하기 위해서 제공되는 것이다. 도면에서 요소들의 형상 및 크기 등은 보다 명확한 설명을 위해 과장될 수 있다.

[0018] 프리페치(Prefetch) 기법의 일 예로, 공간 기억 스트리밍(Spatial Memory Streaming; SMS)은 공간 지역성을 활용하기 위해 데이터 스트림의 히스토리를 비트로 표현하여 벡터로 저장하는 하드웨어 프리페치 기법이다. 학습 시간(learning time) 과정을 통해 공간 지역성을 보이는 다수의 캐시 블록들이 비트로 저장되며, 이 비트들의 집합인 공간 영역(spatial region)을 만든다. 공간 영역 내 속하는 임의의 캐시 블록이 데이터 캐시에서 무효화되거나 제거되었을 때 학습 시간 과정은 끝나게 된다. SMS는 필터 테이블(filter table), 누적 테이블(accumulation table), 패턴 히스토리 테이블(pattern history table) 등 3가지 하드웨어 테이블을 사용한다. 필터 테이블은 동일한 공간 영역을 생성하지 않기 위해 중복되는 공간 영역을 검사하며, 누적 테이블은 공간 영역을 생성하기 위한 테이블로서 태그, PC/offset, 패턴 필드로 구성된다. 패턴 히스토리 테이블은 누적 테이블에서 생성한 공간 영역 정보를 저장하는 버퍼 역할을 한다.

[0019] SMS는 공간적 지역성의 이점을 사용하기 위해 공간 영역의 시작 주소와 인접한 데이터 주소를 비트로 표현하여 공간 영역 만든다. 데이터 주소의 상위 비트가 공간 영역을 생성하기 위한 태그 값으로 태그 필드에 저장되고, 상위 비트를 제외한 나머지 하위 비트는 오프셋 필드에 저장된다. 그리고 Load/Store의 프로그램 카운터 값이 PC 필드에 저장된다. 필터 테이블에 동일한 태그 값이 있으면 동일한 공간 영역이 있다는 것을 의미하기 때문에 누적 테이블에서 해당 공간 영역을 찾아 새로운 정보를 업데이트한다. 동일한 공간 영역이 없으면 새로운 공간 영역의 태그 값, PC 값, 오프셋 값이 필터 테이블에 저장된다. 공간 영역의 범위는 패턴 필드에 저장된 비트벡터의 길이에 따라 결정된다. 공간 영역을 생성하는 동안 공간 영역의 범위에 있는 블록이 L1 데이터 캐시에서 방출되거나 무효화되면, 해당하는 공간 영역의 PC/offset, 비트 벡터 정보는 패턴 히스토리 테이블에 저장된다.

[0020] 프리페치 기법의 다른 예로, 프로액티브 명령어 페치(Proactive Instruction Fetch; PIF)는 순차적인 접근 패턴을 보이는 명령어의 특성을 이용해 명령어 스트림의 순서를 비트로 저장하는 하드웨어 프리페치 기법이다. PIF는 압축기(compactor), 히스토리 버퍼(history buffer), 스트림 어드레스 버퍼(stream address buffer), 인덱스 테이블(index table)필터 등 4가지 하드웨어 테이블로 구성된다.

[0021] 압축기는 완료된 명령어(retired instruction)의 주소를 개별적으로 저장하지 않고 동일한 명령어 블록에 속하는 연속적인 PC 값을 하나의 주소로 축약해 공간 영역을 만든다. 압축기는 시간적 압축(temporal compaction) 단계와 공간적 압축(spatial compaction) 단계로 구성된다. 명령어 블록 간 공간적 지역성의 이점을 얻기 위해 공간적 압축기는 인접한 명령어 블록의 집합인 공간 영역 내에 명령어 블록 주소를 모은다. 공간 영역에서 트리거라고 부르는 첫 번째 액세스 명령어 블록에 따라 공간 압축기는 공간 영역의 범위를 정의한다. 새로운 공간 영역은 트리거를 기준으로 이전의 N 블록과 이후의 M 블록으로 총 N + M + 1의 명령어 블록으로 구성된다. 압축기는 새로운 공간 영역의 경계를 정의하는 트리거 PC를 기록하고, 각 비트 자리가 공간 영역 내의 명령어 블록을 나타내는 비트 벡터를 초기화한다. 현재 공간 영역 내의 명령어로서 해당 비트는 비트 벡터에서 설정된다. 현재 공간 영역의 범위 밖에 있는 명령어가 완료된다면 현재 트리거 PC 값과 비트벡터를 기록하고 시간적 압축기로 보낸다. 그리고 공간적 압축기는 새로운 공간 영역 내의 블록들을 감시하기 시작한다.

[0022] 특정한 코드를 무수히 많이 반복하는 코드에서는 일반적으로 몇 개의 영역에 걸쳐 공간적 루프 명령 풋 프린트, 실행된 명령의 많은 부분을 구성한다. 이상적으로는 명령어 프리페치 기법을 사용할 경우, 루프의 모든 명령어 블록은 첫 루프를 반복하기 전에 L1 명령어 캐시로 프리페치된다. 명령어 블록이 이미 캐시에 저장되어있기 때문에 후속으로 반복하는 명령어를 예측해도 이점은 없다. 이러한 첫 루프를 실행 후에 반복하는 공간 영역을 기록하는 중복을 피하기 위해 시간적 압축기는 가장 최근에 관찰된 공간 영역 레코드의 개수를 추적한다.

[0023] 히스토리 버퍼는 FIFO(First In First Out) 순으로 완료된 명령들의 순서를 저장하는 순환 버퍼이다. 각각의 저장위치에 트리거 명령어의 블록 주소와 인접한 블록들을 저장한다. 인덱스 테이블에 저장된 트리거 PC 값은 맵핑된 내용을 이용해 가장 최근에 히스토리 버퍼에 기록된 내용을 가리키고 있기 때문에 히스토리 버퍼에 저장된 내용을 빠르게 검색할 수 있다. 스트림 어드레스 버퍼는 히스토리 테이블에 있는 시퀀스를 가리키는 포인터를 유지한다. 이 포인터는 초기에 인덱스 테이블에서 설정된다. 각각의 레코드에 대해 스트림 어드레스 버퍼는 비트벡터로 인코딩된 명령어 블록들 주소를 계산하고 이 주소들에 대해 프리페치를 한다.

[0024] 아래에서는 본 발명에 따른 하드웨어 프리페처(Prefetcher)에 대해 설명한다. 하드웨어 프리페처는 프로세서가 접근한 프로그램의 순서 정보를 기록하는 히스토리 생성기(history generator)와, 히스토리 생성 과정에서 기록한 정보들을 저장하는 히스토리 버퍼(history buffer)를 포함하여 이루어질 수 있다.

[0025] 본 발명에 따른 하드웨어 프리페처는 프로세서가 접근한 프로그램의 순서를 히스토리 레코드로 저장할 수 있다. 히스토리를 생성하는 과정에서 히스토리 레코드의 첫 블록 주소를 트리거 블록이라 하며, 트리거 블록과 그 다음으로 캐시 미스가 발생한 블록 주소의 차이 값(변위 값)을 변위 필드에 저장할 수 있다. 공간 영역을 나타내기 위해 변위 필드의 비트 수에 따라 저장할 수 있는 범위는 조절 가능하다. 또한, 히스토리 레코드의 첫 블록 주소인 트리거 블록에 우선권을 부여하여 캐시 교체가 요청되었을 경우 우선권이 없는 블록을 먼저 교체할 수 있다.

[0026] 프로세서는 데이터를 로드할 때 해당 데이터의 인접한 데이터도 참조할 가능성이 크다. 캐시는 공간적 지역성을 이용해 메모리에서 데이터를 로드할 때 주변 데이터들까지 같이 로드하는데 캐시와 메모리 사이에는 블록 단위로 데이터를 교환할 수 있다. 히스토리 생성기의 변위 필드에는 히스토리 레코드의 시작 블록인 트리거 블록과 트리거 블록 이후에 캐시 미스가 발생한 블록들의 주소 차이(변위 값)가 저장될 수 있다.

[0027] 표 1은 히스토리 생성기의 변위 필드가 비트 수에 따라 저장할 수 있는 블록 주소의 범위를 나타낼 수 있다.

표 1

Bits	Coverage	
	negative	positive
2	-2	+1
3	-3	+4
4	-8	+7
5	-16	+15
6	-32	+31
7	-64	+63
8	-128	+127

[0028]

[0029] 표 1을 참조하면, 히스토리 생성기의 변위 필드에서 비트 수가 커질수록 두 블록의 간격 차이가 큰 블록을 저장할 수 있다. 하지만 많은 수의 변위를 저장하기 위해서는 변위 필드의 크기가 커지기 때문에 스토리지 오버헤드가 발생할 수 있다. 반면, 변위 필드의 비트 수가 작을수록 트리거 주소를 기준으로 매우 인접한 블록만 변위 필드에 저장할 수 있다.

[0030] 일례로, 변위 필드를 4 비트와 8 비트로 설정한 하드웨어 프리페처를 나타낼 수 있다. 그리고 히스토리 생성 단계에서 만들어지는 변위 필드의 전체 크기를 32 비트로 설정할 수 있다. 이 때 변위 필드를 4 비트로 설정하면 총 8개의 변위 필드로 구성된 히스토리 레코드가 만들어지고, 변위 필드를 8 비트로 설정하면 총 4개의 변위 필드로 구성된 히스토리 레코드가 만들어질 수 있다.

[0031] 이하에서 변위를 이용한 히스토리 레코드 생성 과정을 구체적으로 설명하기로 한다.

[0032] 히스토리 생성기는 캐시 미스가 발생한 주소들을 기록하여 히스토리를 만든 뒤, 프로세서가 다시 이 주소를 필요로 할 때 캐시 블록을 프리페치할 수 있다. 하나의 캐시 블록에서 캐시 미스가 발생할 때마다 프리페치 요청을 하지 않고, 캐시 미스가 발생한 캐시 블록들을 모아 공간적 지역성을 활용한 레코드를 만들어 프로그램의 히스토리를 만들 수 있다.

[0033] 히스토리 생성기의 테이블은 히스토리 레코드의 첫 시작 블록 주소를 저장하는 트리거 주소(trigger address) 필드, 블록들 간의 주소 차이를 저장하는 변위(displacement) 필드 등 2개의 필드로 구성될 수 있다.

[0034] 본 발명의 일 실시예에 따른 변위 히스토리 버퍼를 이용한 명령어 및 데이터 프리페치 방법을 아래와 같이 나타

낼 수 있다.

[0035] 변위 히스토리 버퍼를 이용한 명령어 및 데이터 프리페치 방법에 있어서, 캐시 블록에서 캐시 미스가 발생하는 경우, 히스토리 생성기의 트리거 주소 필드에 저장된 트리거 주소에서 캐시 블록의 주소를 검색하는 단계, 히스토리 생성기의 트리거 주소 필드에 저장된 트리거 주소가 없는 경우, 캐시 블록의 주소를 저장하여 새로운 히스토리 레코드의 기준이 되는 트리거 주소로 설정하는 단계, 다른 캐시 블록에서 캐시 미스가 발생하는 경우, 히스토리 생성기의 트리거 주소 필드에 저장되어 있는 트리거 주소와의 변위 값을 히스토리 생성기의 변위 필드에 저장하는 단계, 히스토리 생성기의 변위 필드의 엔트리에 변위 값들이 저장되어 저장 공간이 없거나 저장할 수 없는 범위의 캐시 블록이 발생하는 경우, 히스토리 레코드의 생성을 중단하고 엔트리를 히스토리 버퍼에 저장하는 단계; 및 히스토리 생성기의 변위 필드의 범위가 아닌 캐시 블록에서 캐시 미스가 발생하는 경우, 기 저장된 트리거 주소의 블록과 변위 값들을 히스토리 버퍼에 저장하고, 히스토리 생성기의 트리거 주소 필드와 히스토리 생성기의 변위 필드에 저장된 값들을 초기화하는 단계를 포함하고, 히스토리 생성기와 히스토리 버퍼 중 적어도 하나 이상은 캐시 미스가 발생한 주소들을 기록하여 히스토리 레코드를 만들어, 프로세서로부터 다시 주소의 요청 시 캐시 블록을 프리페치할 수 있다.

[0036] 여기서, 히스토리 버퍼는 히스토리 버퍼의 트리거 주소 필드와 히스토리 버퍼의 변위 필드로 구성되며 각각의 엔트리에는 히스토리 레코드가 저장되고, 트리거 액세스 필드는 히스토리 버퍼의 인덱스 역할을 하며, 히스토리 레코드와 트리거 액세스 필드의 정보가 일치하고, 트리거 액세스 필드에 일치하는 엔트리가 히스토리 버퍼에 존재하는 경우, 캐시 블록의 주소를 프리페치한 후 히스토리 버퍼의 변위 필드에 저장되어 있는 변위 값을 저장된 순서에 따라 캐시 블록의 주소 값에 더해 프리페치할 수 있다.

[0037] 그리고 히스토리 레코드의 첫 캐시 블록 주소인 히스토리 버퍼의 트리거 주소 블록에 우선순위를 부여하여, 캐시 교체가 요청되는 경우 랜덤(Random) 교체 정책에 의해 우선순위를 부여 받지 못한 히스토리 버퍼의 변위 필드 블록부터 교체할 수 있다.

[0038] 본 발명의 실시예들에 따르면 프리페치의 요청 시 사용될 데이터를 정확하게 예측하여 적절한 시간에 불러오는 명령어 및 데이터 프리페치 기술을 제공할 수 있다. 또한, 캐시를 이용하는 프로세서 환경에서 명령어 데이터 처리 성공율을 높임으로써, 프로세서 처리 성능을 증가시키면서 캐시 메모리 구조를 단순화하여 전력, 비용 등을 절감할 수 있다.

[0039] 계속해서 변위 히스토리 버퍼를 이용한 명령어 및 데이터 프리페치 방법의 각 단계에 대해 더 구체적으로 설명하기로 한다.

[0040] 도 1은 일 실시예에 따른 변위를 이용한 히스토리 레코드 생성 과정을 설명하기 위한 도면이다.

[0041] 도 1을 참조하여 히스토리 생성기(100)의 동작을 하나의 실시예를 이용하여 구체적으로 설명할 수 있다. 예를 들어 블록 A에서 캐시 미스가 발생하는 경우, 히스토리 생성기의 트리거 주소 필드(110)를 검색할 수 있다. 트리거 주소 필드(110)에 저장된 값이 없는 경우에는 블록 A의 주소를 저장할 수 있다. 블록 A의 주소는 새로운 레코드의 기준이 되는 트리거 주소가 될 수 있다.

[0042] 이후, 블록 A+3에서 캐시 미스가 발생하면 트리거 주소 필드에 저장되어 있는 트리거 블록 주소와의 차이 +3을 변위 필드(120)에 저장할 수 있다. 또한, 이후에 미스가 발생한 블록들이 일례로 A-4, A+1, A-2인 경우, 트리거 주소와의 차이를 변위 필드(120)에 순서대로 -4, +1, -2를 각각의 변위 필드에 저장할 수 있다.

[0043] 히스토리 레코드를 생성하는 과정에서 변위 필드의 엔트리에 변위 값들이 모두 저장이 되거나 변위 필드(120)에 저장할 수 없는 범위의 블록이 발생하는 경우, 히스토리 레코드의 생성을 중단하고 해당 엔트리를 히스토리 버퍼(200)에 저장할 수 있다.

[0044] 트리거 주소 블록 A에 대한 레코드를 생성하다가 변위 필드 범위 밖의 블록 B에서 캐시 미스가 발생하는 경우에는, 이전까지 기록해놓은 트리거 주소 블록 A와 변위 값들을 히스토리 버퍼(200)에 저장하고, 히스토리 생성기의 트리거 주소 필드(110)와 변위 필드(120)에 저장된 값들을 모두 초기화할 수 있다.

[0045] 이후, 블록 B에 대한 레코드를 생성하기 위해 트리거 주소 필드(110)에 블록 B의 주소를 저장할 수 있다. 또한, 이후에 블록 B+2에서 캐시 미스가 발생하면 트리거 주소 필드(110)에 저장되어 있는 블록 B의 주소와의 차이(변위 값)를 변위 필드(120)에 저장할 수 있다. 다음으로, 예를 들어 블록 B-5에서 캐시 미스가 발생하면

블록 B의 주소와의 차이를 변위 필드(120)에 저장할 수 있다.

[0046] 도 2는 일 실시예에 따른 히스토리 버퍼와 히스토리 레코드를 이용한 예측 과정을 설명하기 위한 도면이다.

[0047] 히스토리 버퍼(200)는 트리거 주소 필드(210)와 변위 필드(220)로 구성되며, 각각의 엔트리에는 히스토리 생성 단계에서 만든 히스토리 레코드가 저장될 수 있다. 트리거 액세스 필드(230)는 히스토리 버퍼(200)의 인덱스 역할을 할 수 있다.

[0048] 도 2를 참조하면, 예를 들어 저장된 히스토리 레코드와 트리거 액세스 정보가 일치하였을 때 동작을 나타낼 수 있다. 트리거 액세스에 일치하는 엔트리가 히스토리 버퍼(200)에 존재하는 경우, 블록 A의 주소를 가장 먼저 프리페치한 후 변위 필드(220)에 저장되어 있는 변위 값을 저장된 순서에 따라 블록 A의 주소 값에 더해 블록 A+3, A-4, A+1, A-2를 프리페치할 수 있다.

[0049] 더욱이, 아래와 같이 캐시 교체 정책의 최적화를 수행할 수 있다.

[0050] 캐시에서 캐시 적중률을 높이는데 캐시 교체 정책이 중요한 역할을 할 수 있다. Direct Mapped 캐시에서는 캐시 교체 정책을 고민할 필요가 없지만 Set-Associative 캐시에서는 최대 N개 중 하나의 캐시 블록을 제거해야 하기 때문에 어떤 교체 정책을 사용할지 판단할 필요가 있다. 가장 단순한 방법에는 임의로 캐시 블록을 교체하는 랜덤(Random) 교체 정책과 가장 오래된 캐시 블록을 교체하는 LRU(Least Recently Used) 교체 정책이 있다.

[0051] LRU 교체 정책은 최근에 참조된 캐시 블록을 순서대로 비교할 수 있도록 LRU 스택을 개별 캐시 세트에 할당할 수 있다. 캐시 미스가 발생하면 LRU 스택을 살펴보고 가장 오래 사용되지 않은 블록을 교체하는 방법이다. 캐시 블록마다 타임스탬프를 기록하기 위해 N-way Set-Associative 캐시일 때, $N\log_2 N$-bit 디멀티플렉서 1개, $2\log_2 N$-bit 멀티플렉서 1개, N-bit 디멀티플렉서 1개, $N\log_2 N$-bit 멀티플렉서 1개, $N\log_2 N$-bit 비교기, $\log_2 N$-bit 레지스터 N개, N-bit 우선순위 인코더 1개가 필요하다. 따라서 캐시의 Set-Associativity가 높아질수록 하드웨어 복잡도가 높아진다.

[0052] 본 발명에 따른 트리거 블록을 기준으로 변위를 저장하는 방법에서 트리거 블록이 먼저 캐시에 저장되고, 이어서 변위 값에 해당하는 블록들이 순서대로 저장될 수 있다. 따라서 LRU 교체 정책을 사용하는 경우 캐시 공간이 가득 찼을 때 트리거 블록이 교체될 확률이 높다. 트리거 블록이 교체된 후, 프로세서가 교체된 트리거 블록을 필요로 했을 때 히스토리 버퍼에서 해당하는 히스토리 레코드를 찾아 트리거 블록과 변위 블록들을 프리페치할 수 있다. 따라서 캐시에는 동일한 변위 블록들이 중복되어 저장될 수 있다.

[0053] 이와 같은 LRU 교체정책을 사용했을 때의 단점을 보완하기 위해 교체할 블록을 임의로 선택하는 랜덤(Random) 교체 정책에 트리거 블록에 우선순위를 부여하여 히스토리 레코드의 변위 블록부터 교체하는 방법을 이용할 수 있다.

[0054] 도 3은 일 실시예에 따른 트리거 블록에 우선순위를 부여하는 랜덤 교체 정책의 동작을 설명하기 위한 도면이다.

[0055] 도 3을 참조하면, 트리거 블록에 우선순위를 부여하고 히스토리 레코드의 변위 블록부터 교체하는 랜덤(Random) 교체 정책을 설명할 수 있다. 랜덤(Random) 교체 정책은 LRU 교체 정책과 달리 각각의 블록마다 타임스탬프나 접근 시간을 기록하지 않고, 히스토리 레코드의 트리거 블록에만 우선순위를 기록할 수 있다.

[0056] 예를 들어 액세스 스트림 순서대로 캐시에 저장되어 캐시 공간이 가득 찼을 때, 가장 오래 사용되지 않은 블록은 블록 A가 될 수 있다. 여기서 블록 A는 우선순위 비트 값이 1인 트리거 블록이기 때문에 블록 A를 제외한 우선순위 비트가 0인 나머지 블록들 중 임의의 블록이 새로운 블록으로 교체될 수 있다. 캐시 공간에 우선순위 비트가 0인 블록이 다 교체되고 1인 블록만 남아있는 경우, 트리거 블록 중 임의의 블록을 제거하고 새로운 블록으로 교체할 수 있다.

[0057] 트리거 블록에 우선순위를 부여하는 랜덤(Random) 교체 정책은 N-way set-associative 캐시에서 임의의 블록을 선택하기 위한 N개의 선형 피드백 시프트 레지스터와 트리거 블록을 구별하는 N개의 비교기가 사용될 수 있다. 이러한 랜덤(Random) 교체 정책은 LRU 교체 정책에 비해 하드웨어 구조가 간단하며 구현하기 용이하다.

[0058] 도 4는 일 실시예에 따른 프리페처 전체 시스템을 설명하기 위한 도면이다.

[0059] 도 1, 2 및 4를 참조하면, 본 발명의 일 실시예에 따른 변위 히스토리 버퍼를 이용한 명령어 및 데이터 프리페치 시스템은 히스토리 생성기(history generator, 100) 및 히스토리 버퍼(history buffer, 200)를 포함하여 이루어질 수 있다.

[0060] 히스토리 생성기(100)는 프로세서가 접근한 프로그램의 순서 정보를 히스토리 레코드로 저장할 수 있다.

[0061] 이러한 히스토리 생성기(100)는 히스토리 생성기의 트리거 주소(trigger address) 필드(110) 및 히스토리 생성기의 변위(displacement) 필드(120)를 포함할 수 있다.

[0062] 히스토리 생성기의 트리거 주소 필드(110)는 캐시 미스가 발생한 캐시 블록의 주소를 저장할 수 있다.

[0063] 히스토리 생성기의 변위 필드(120)는 히스토리 생성기의 트리거 주소 필드(110)와 다음 캐시 미스가 발생한 캐시 블록의 주소의 변위 값을 저장할 수 있다. 히스토리 생성기의 변위 필드(120)는 비트 수에 따라 저장할 수 있는 변위 값의 범위가 설정될 수 있다.

[0064] 히스토리 버퍼(200)는 히스토리 생성기(100)에서 저장된 히스토리 레코드를 저장할 수 있다.

[0065] 히스토리 버퍼(200)는 히스토리 버퍼의 트리거 주소 필드(210)와 히스토리 버퍼의 변위 필드(220)를 포함할 수 있다. 히스토리 버퍼의 트리거 주소 필드(210)와 히스토리 버퍼의 변위 필드(220)는 히스토리 레코드가 저장될 수 있다.

[0066] 히스토리 버퍼(200)는 인덱스 역할을 하는 트리거 액세스 필드(230)를 더 포함하여 이루어질 수 있다. 히스토리 레코드와 트리거 액세스 필드(230)의 정보가 일치하고, 트리거 액세스 필드(230)에 일치하는 엔트리가 히스토리 버퍼(200)에 존재하는 경우, 캐시 블록의 주소를 프리페치한 후 히스토리 버퍼의 변위 필드(220)에 저장되어 있는 변위 값을 저장된 순서에 따라 캐시 블록의 주소 값에 더해 프리페치할 수 있다.

[0067] 히스토리 생성기의 변위 필드(120)의 엔트리에 변위 값들이 저장되어 저장 공간이 없거나 저장할 수 없는 범위의 캐시 블록이 발생하는 경우, 히스토리 레코드의 생성을 중단하고 엔트리를 히스토리 버퍼(200)에 저장할 수 있다.

[0068] 히스토리 레코드의 첫 캐시 블록 주소인 히스토리 버퍼의 트리거 주소 필드(210)의 블록에 우선순위를 부여하여, 캐시 블록의 교체가 요청되는 경우 랜덤(Random) 교체 정책에 의해 우선순위를 부여 받지 못한 히스토리 버퍼의 변위 필드(220)의 캐시 블록부터 교체할 수 있다.

[0069] 본 발명에 따른 변위 히스토리 버퍼를 이용한 명령어 및 데이터 프리페치 기법은 캐시를 이용하는 프로세서 환경에서 명령어 데이터 처리 성공율을 높임으로써, 프로세서 처리 성능을 증가시키면서 캐시 메모리 구조를 단순화하여 전력, 비용 등을 절감할 수 있다.

[0070] 본 발명에 따른 프리페처에 대해 성능을 분석하고 비교할 수 있다. 프리페처를 검증하기 위해서 gem5 simulator를 이용해 구현하고, 벤치마크 프로그램으로는 PARSEC(Princeton Application Repository for Shared-Memory Computers)를 사용하여 성능을 분석할 수 있다. 기존의 프리페처와 본 발명에 따른 프리페처의 L1 명령어 및 데이터 캐시와 IPC(Instruction per Cycle)을 비교할 수 있다. Set-Associative 변화에 따른 L1 명령어 및 데이터 캐시 미스율의 변화도 확인할 수 있다.

[0071] 표 2는 본 발명에 따른 프리페처 성능을 비교하기 위한 시스템 구성 환경을 나타낼 수 있다.

표 2

Core	ALPHA ISA Quad core, 3.4GHz, OoO 6-Wide Dispatch 8-Wide Commit 40-entry ROB 16-entry RAS 32-entry Issue Queue
I-Cache	32kB, 64B 1, 2, 4, 8-way
D-Cache	32kB, 2-way, 64B 1, 2, 4, 8-way
L2-Cache	2MB, 8-way, 64B 12ns hit latency
Main Memory	2GB, 54ns access latency

[0072]

[0073] 표 2는 프리페처의 성능을 평가하기 위해 실험에 사용한 시스템 구성 환경을 나타낼 수 있다. ALPHA ISA(Instruction Set Architecture)를 사용하고, 시뮬레이션으로 약 10억 개의 명령어를 실행할 수 있다. L1 캐시는 private L1 캐시로 L1-D캐시와 L1-I캐시의 크기를 각각 32kB를 사용하고, L2 캐시는 shared L2 캐시로 크기를 2MB로 고정할 수 있다. L1과 L2 캐시 블록의 크기는 64바이트가 사용될 수 있다. 분기 예측기는 토너먼트 분기 예측기를 사용할 수 있다. 다양한 환경에서 L1 캐시의 성능을 평가하기 위해 크기, Set-Associative의 크기와 L1 캐시의 크기를 16kB, 8kB로 바꿔가며 실험을 할 수 있다.

[0074] PARSEC은 인텔과 프린스턴 대학에서 만들었으며, 최신 프로그램의 경향을 반영하고 다양한 workload를 지원하여 칩 멀티 프로세서를 평가할 수 있는 벤치마크이다. 칩 멀티 프로세서의 성능을 평가하기 위해 다양한 크기의 Input Set을 제공할 수 있다. 실험에서는 Simlarge Input Set을 사용할 수 있다.

[0075] 표 3은 시뮬레이션에서 사용한 PARSEC 프로그램의 특성을 나타낼 수 있다.

표 3

Program		Application Domain	Problem Size	Working Set
INT	dedup	Enterprise Storage	184 MB data	unbounded
	freqmine	Data Mining	990,000 transactions	unbounded
	x264	Media Processing	128 frames, 640 × 360 pixels	medium
FP	blackscholes	Financial Analysis	65,535 options	small
	bodytrack	Computer Vision	4 frames, 4,000 particles	medium
	canneal	Engineering	400,000 elements	unbounded
	ferret	Similarity Search	256 queries, 34,973 images	unbounded
	fluidanimate	Animation	5frames, 300,000 transactions	large
	streamcluster	Data Mining	16,384 points per block, 1 block	medium
	swaptions	Financial Analysis	64 swaptions, 20,000 simulations	medium

[0076]

[0077] blackscholes와 swaptions는 재무분석, bodytrack은 컴퓨터 비전, canneal은 엔지니어링, dedup은 엔터프라이즈 스토리지, ferret은 이미지 검색, fluidanimate는 애니메이션, freqmine과 streamcluster는 데이터 마이닝, x264는 이미지 처리 관련 프로그램이다.

[0078] 시뮬레이션 결과 및 분석에 따라 PARSEC 벤치마크 프로그램 별로 L1 데이터 캐시와 명령어 캐시의 미스율을 나타낼 수 있다. 세로축은 미스율을 나타내며, 가로축은 각각의 벤치마크 프로그램의 명칭을 나타낼 수 있다. dedup, freqmine, x264 등 3개의 어플리케이션은 정수형 프로그램이며, blackscholes, bodytrack, canneal, ferret, fluidanimate, streamcluster, swaption 등 7개의 어플리케이션은 실수형 프로그램이 될 수 있다.

[0079] 모든 하드웨어 프리페처는 2K의 엔트리를 가지며, 각 프로그램 별로 세 개의 막대로 표시된 미스율이 포함될 수 있다. 첫 번째 막대의 크기는 기존에 제안되었던 프리페처의 미스율이며, 두 번째와 세 번째 막대의 크기는 제안한 프리페치의 미스율을 나타낼 수 있다. 두 번째 막대는 변위 필드의 한 영역의 크기가 8비트이며, 세 번째 막대는 변위 필드의 한 영역의 크기가 4 비트인 프리페처를 나타낼 수 있다.

[0080] 도 5는 일 실시예에 따른 L1 데이터 캐시의 미스율을 나타낸 그래프이다.

[0081] 도 5를 참조하면, BitVector(32bits)는 기존에 제안된 SMS를 나타내며, Displacement(8bitsx4)와 Displacement(4bitsx8)는 제안된 프리페처를 나타낼 수 있다. SMS의 경우 평균 약 27.1%의 미스율을 나타낼 수 있다. 변위 필드가 8 비트인 제안된 프리페처의 경우 평균 약 23%, 변위 필드를 4비트로 설정한 제안된 프리페처의 경우 평균 약 18.7%의 나타낼 수 있다. SMS와 비교했을 때 제안된 프리페처의 L1 데이터 미스율은 변위 필드의 크기를 8비트로 고정했을 경우 약 15% 감소하였고, 4 비트로 고정했을 경우 약 44.5% 감소할 수 있다. 4 비트 변위 필드를 사용한 프리페처의 방법을 사용할 경우 L1 데이터 캐시의 미스율이 가장 낮은 결과를 나타

낼 수 있다.

[0082] 도 6은 일 실시예에 따른 L1 명령어 캐시의 미스율을 나타낸 그래프이다.

[0083] 도 6을 참조하면, BitVector(32bits)는 기존에 제안된 PIF를 나타내며 평균 약 12.8%의 미스율을 나타낼 수 있다. 변위 필드를 8비트로 설정한 제안된 프리페처의 경우 평균 약 11%, 변위 필드를 4 비트로 설정한 제안된 프리페처의 경우 평균 약 9.5%의 미스율을 나타낼 수 있다. PIF와 비교했을 때 제안된 프리페처의 명령어 미스율은 변위 필드의 크기를 8비트로 고정했을 경우 약 14.3% 감소하였고, 4 비트로 고정했을 경우 약 26.1% 감소할 수 있다. L1 명령어 캐시도 4 비트 변위 필드를 사용한 프리페치의 미스율이 가장 낮은 결과를 나타낼 수 있다.

[0084] 명령어 액세스는 순차적이거나 반복적인 패턴을 보이는 반면에 데이터 액세스는 순회적이거나 다양한 패턴을 나타낼 수 있다. 제안하는 프리페처를 사용했을 때, 명령어 캐시의 미스율은 평균 31% 감소하였고 데이터 캐시의 미스율은 평균 44.5% 감소하여 데이터 캐시에서 제안된 프리페처가 더 효율적임을 확인할 수 있다. .

[0085] 아래에서는 Set-Associativity에 따른 L1 캐시의 미스율 변화를 구체적으로 나타낼 수 있다.

[0086] 도 7 내지 도 10은 본 발명에 따른 프리페처의 Set-Associativity의 크기 변화에 따른 L1 데이터 캐시와 명령어 캐시의 미스율 변화를 비교한 것이다. Set-Associativity의 크기는 1-way부터 8-way까지 바꿔가며 실험할 수 있다.

[0087] 도 7은 일 실시예에 따른 8 비트 변위 필드의 L1 데이터 캐시의 미스율 변화를 나타낸 그래프이다.

[0088] 도 7을 참조하면, 변위 필드의 크기를 8 비트로 설정하였으며 4개의 변위 필드가 하나의 히스토리 레코드를 형성할 수 있다. Set-Associativity를 1-way로 설정했을 때 L1 데이터 캐시의 미스율은 평균 약 25.3%, 2-way로 설정했을 때 평균 약 23%, 4-way로 설정했을 때 평균 약 21.8%, 8-way로 설정했을 때 평균 약 18.3% 미스율을 나타낼 수 있다.

[0089] 도 8은 일 실시예에 따른 4 비트 변위 필드의 L1 데이터 캐시의 미스율 변화를 나타낸 그래프이다.

[0090] 도 8을 참조하면, 변위 필드의 크기를 4비트로 설정하며 8개의 변위 필드가 하나의 히스토리 레코드를 형성할 수 있다. Set-Associativity를 1-way로 설정했을 때 L1 데이터 캐시의 미스율은 평균 약 19.8%, 2-way로 설정했을 때 평균 약 18%, 4-way로 설정했을 때 평균 약 17.6%, 8-way로 설정했을 때 평균 약 17.1% 미스율을 나타낼 수 있다.

[0091] 도 9는 일 실시예에 따른 8 비트 변위 필드의 L1 명령어 캐시의 미스율 변화를 나타낸 그래프이다.

[0092] 도 9를 참조하면, 변위 필드의 크기를 8 비트로 설정하였으며 4개의 변위 필드가 하나의 히스토리 레코드를 형성할 수 있다. Set-Associativity를 1-way로 설정했을 경우와, L1 명령어 캐시의 미스율은 평균 약 9.8%, 2-way로 설정했을 경우와, 평균 약 8.6%, 4-way로 설정했을 경우와, 평균 약 7.4%, 8-way로 설정했을 경우, 그리고 평균 약 6.5% 미스율을 나타낼 수 있다.

[0093] 도 10은 일 실시예에 따른 4 비트 변위 필드의 L1 명령어 캐시의 미스율 변화를 나타낸 그래프이다.

[0094] 도 10을 참조하면, 변위 필드의 크기를 4 비트로 설정하며 8개의 변위 필드가 하나의 히스토리 레코드를 형성할 수 있다. Set-Associativity를 1-way로 설정했을 경우 L1 명령어 캐시의 미스율은 평균 약 9.4%이고, 2-way로 설정했을 경우 평균 약 7.7%이며, 4-way로 설정했을 경우 평균 약 6.5%이고, 8-way로 설정했을 경우 평균 약 5.6% 미스율을 나타낼 수 있다. 평균적으로 8-way일 때, L1 데이터 및 명령어 캐시의 미스율이 가장 낮음을 확인할 수 있다.

[0095] 계속해서, 캐시 교체 정책에 따른 L1 캐시의 미스율을 구체적으로 나타낼 수 있다.

[0096] 아래에서는 캐시 교체 정책에 따른 L1 캐시 미스율의 변화에 대해 나타낼 수 있다. LRU replacement는 하드웨어 프리페처를 적용하지 않고 LRU교체 정책을 사용한 경우이며, Random with priority는 4비트 변위 필드를 사용하는 제안된 프리페처를 사용하고 트리거 블록에 우선순위를 가지게 하는 랜덤(Random) 교체 정책을 사용한 경우가 될 수 있다.

[0097] 도 11 및 도 13는 일 실시예에 따른 캐시 교체 정책에 따른 L1 데이터 캐시의 미스율을 나타낸 그래프이다. 즉, 도 11부터 도 13까지 L1 데이터와 명령어 캐시의 크기를 각각 8kB, 16kB, 32kB로 설정했을 때 L1 데이터와 명령어 캐시의 미스율을 나타낸 그래프이다.

[0098] 도 11을 참조하면, 데이터 캐시의 크기를 8kB로 설정하여 벤치마크를 수행한 결과이다. 프리페처를 적용하지 않고 LRU 교체 정책을 사용한 경우 평균 약 74% 미스율을 보이며, 10개 프로그램 모두 높은 미스율을 나타낼 수 있다. 제안된 프리페처에 우선순위 랜덤(Random) 교체 정책을 사용한 경우 21% 미스율을 보이며, LRU 교체 정책과 비교해 제안한 우선순위 랜덤(Random) 교체 정책을 사용할 경우 미스율은 약 70.6% 감소할 수 있다.

[0099] 도 12에 도시된 바와 같이, 데이터 캐시의 크기를 16kB로 설정하여 벤치마크를 수행한 결과이다. 프리페처를 적용하지 않고 LRU 교체 정책을 사용한 경우 평균 약 74% 미스율을 나타낼 수 있다. 제안된 프리페처에 우선순위 랜덤(Random) 교체 정책을 사용한 경우 17.5% 미스율을 보이며, LRU 교체 정책과 비교해 제안된 우선순위 랜덤(Random) 교체 정책을 사용할 경우 미스율은 약 70.8% 감소할 수 있다.

[0100] 도 13에 도시된 바와 같이, 데이터 캐시의 크기를 32kB로 설정하여 벤치마크를 수행한 결과이다. 프리페처를 적용하지 않고 LRU 교체 정책을 사용한 경우 평균 약 45.5% 미스율을 보이며, 10개 프로그램 모두 높은 미스율을 나타낼 수 있다. 제안된 프리페처에 우선순위 랜덤(Random) 교체 정책을 사용한 경우 15.5% 미스율을 보이며, LRU 교체 정책과 비교해 미스율은 약 65.8% 감소될 수 있다. L1 데이터 캐시의 크기를 변경하며 미스율을 확인해본 결과, 캐시 크기를 크게 할수록 L1 데이터 캐시의 미스율은 낮아졌고, 우선순위 랜덤(Random) 교체 정책을 사용한 제안된 프리페처에서는 LRU를 적용한 프리페처를 사용하지 않는 방법보다 L1 데이터 미스율이 더 낮은 결과를 나타낼 수 있다.

[0101] 도 14는 일 실시예에 따른 IPC 스피드업 성능 비교를 나타낸 그래프이다.

[0102] 도 14를 참조하면, IPC(Instruction per Cycle) Speedup을 비교한 결과로, PIF의 경우는 평균 약 1.19이며 변위 필드를 8 비트로 설정한 제안된 프리페처는 평균 약 1.38, 변위 필드를 4비트로 설정한 제안된 프리페처는 평균 약 1.48의 IPC를 나타낼 수 있다. 공통적으로 canneal 프로그램에서 3가지 방법 모두 가장 낮은 IPC SpeedUp의 결과를 나타낼 수 있다. 이는, canneal은 칩 디자인을 할 때 라우팅 비용을 최소로 하기 위해 수도 랜덤(pseudo-random)하게 임의의 입력 데이터를 서로 교환하기 때문이다.

[0103] 또한, 3가지 방법 모두 x264 프로그램에서 가장 높은 IPC SpeedUp의 결과를 나타낼 수 있다. PIF의 방법 대비 변위 필드의 크기를 8비트로 설정한 프리페처의 IPC는 평균 약 15%정도 IPC가 향상되었고, 변위 필드의 크기를 4비트로 설정한 프리페처의 IPC는 평균 약 24%정도 IPC가 향상되었다.

[0104] 이상에서 설명된 장치는 하드웨어 구성요소, 소프트웨어 구성요소, 및/또는 하드웨어 구성요소 및 소프트웨어 구성요소의 조합으로 구현될 수 있다. 예를 들어, 실시예들에서 설명된 장치 및 구성요소는, 예를 들어, 프로세서, 컨트롤러, ALU(arithmetic logic unit), 디지털 신호 프로세서(digital signal processor), 마이크로컴퓨터, FPA(field programmable array), PLU(programmable logic unit), 마이크로프로세서, 또는 명령(instruction)을 실행하고 응답할 수 있는 다른 어떠한 장치와 같이, 하나 이상의 범용 컴퓨터 또는 특수 목적 컴퓨터를 이용하여 구현될 수 있다. 처리 장치는 운영 체제(OS) 및 상기 운영 체제 상에서 수행되는 하나 이상의 소프트웨어 애플리케이션을 수행할 수 있다. 또한, 처리 장치는 소프트웨어의 실행에 응답하여, 데이터를 접근, 저장, 조작, 처리 및 생성할 수도 있다. 이해의 편의를 위하여, 처리 장치는 하나가 사용되는 것으로 설명된 경우도 있지만, 해당 기술분야에서 통상의 지식을 가진 자는, 처리 장치가 복수 개의 처리 요소(processing element) 및/또는 복수 유형의 처리 요소를 포함할 수 있음을 알 수 있다. 예를 들어, 처리 장치는 복수 개의 프로세서 또는 하나의 프로세서 및 하나의 컨트롤러를 포함할 수 있다. 또한, 병렬 프로세서(parallel processor)와 같은, 다른 처리 구성(processing configuration)도 가능하다.

[0105] 소프트웨어는 컴퓨터 프로그램(computer program), 코드(code), 명령(instruction), 또는 이들 중 하나 이상의 조합을 포함할 수 있으며, 원하는 대로 동작하도록 처리 장치를 구성하거나 독립적으로 또는 결합적으로(collectively) 처리 장치를 명령할 수 있다. 소프트웨어 및/또는 데이터는, 처리 장치에 의하여 해석되거나 처리 장치에 명령 또는 데이터를 제공하기 위하여, 어떤 유형의 기계, 구성요소(component), 물리적 장치, 가상 장치(virtual equipment), 컴퓨터 저장 매체 또는 장치, 또는 전송되는 신호 파(signal wave)에 영구적으로,

또는 일시적으로 구체화(embody)될 수 있다. 소프트웨어는 네트워크로 연결된 컴퓨터 시스템 상에 분산되어서, 분산된 방법으로 저장되거나 실행될 수도 있다. 소프트웨어 및 데이터는 하나 이상의 컴퓨터 판독 가능 기록 매체에 저장될 수 있다.

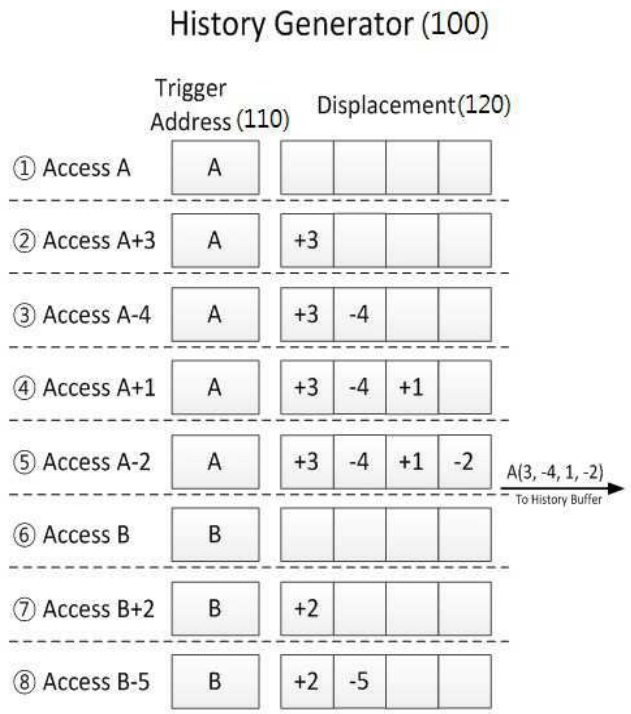
[0106] 실시예에 따른 방법은 다양한 컴퓨터 수단을 통하여 수행될 수 있는 프로그램 명령 형태로 구현되어 컴퓨터 판독 가능 매체에 기록될 수 있다. 상기 컴퓨터 판독 가능 매체는 프로그램 명령, 데이터 파일, 데이터 구조 등을 단독으로 또는 조합하여 포함할 수 있다. 상기 매체에 기록되는 프로그램 명령은 실시예를 위하여 특별히 설계되고 구성된 것들이거나 컴퓨터 소프트웨어 당업자에게 공지되어 사용 가능한 것일 수도 있다. 컴퓨터 판독 가능 기록 매체의 예에는 하드 디스크, 플로피 디스크 및 자기 테이프와 같은 자기 매체(magnetic media), CD-ROM, DVD와 같은 광기록 매체(optical media), 플롭티컬 디스크(floptical disk)와 같은 자기-광 매체(magneto-optical media), 및 롬(ROM), 램(RAM), 플래시 메모리 등과 같은 프로그램 명령을 저장하고 수행하도록 특별히 구성된 하드웨어 장치가 포함된다. 프로그램 명령의 예에는 컴파일러에 의해 만들어지는 것과 같은 기계어 코드뿐만 아니라 인터프리터 등을 사용해서 컴퓨터에 의해서 실행될 수 있는 고급 언어 코드를 포함한다. 상기된 하드웨어 장치는 실시예의 동작을 수행하기 위해 하나 이상의 소프트웨어 모듈로서 작동하도록 구성될 수 있으며, 그 역도 마찬가지이다.

[0107] 이상과 같이 실시예들이 비록 한정된 실시예와 도면에 의해 설명되었으나, 해당 기술분야에서 통상의 지식을 가진 자라면 상기의 기재로부터 다양한 수정 및 변형이 가능하다. 예를 들어, 설명된 기술들이 설명된 방법과 다른 순서로 수행되거나, 및/또는 설명된 시스템, 구조, 장치, 회로 등의 구성요소들이 설명된 방법과 다른 형태로 결합 또는 조합되거나, 다른 구성요소 또는 균등물에 의하여 대치되거나 치환되더라도 적절한 결과가 달성될 수 있다.

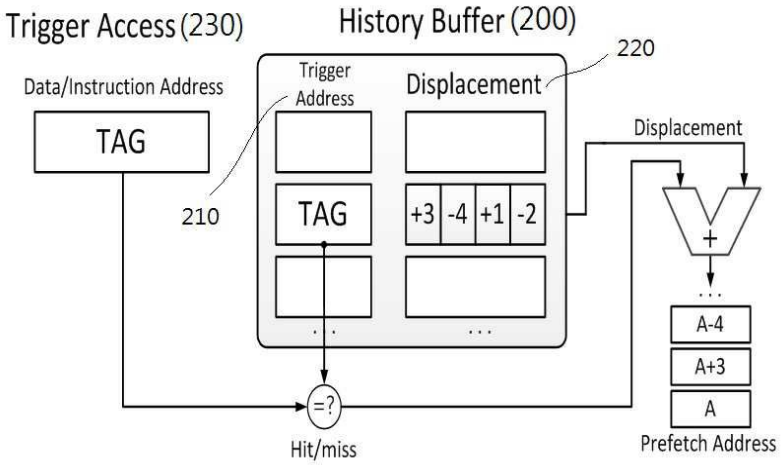
[0108] 그러므로, 다른 구현들, 다른 실시예들 및 특허청구범위와 균등한 것들도 후술하는 특허청구범위의 범위에 속한다.

도면

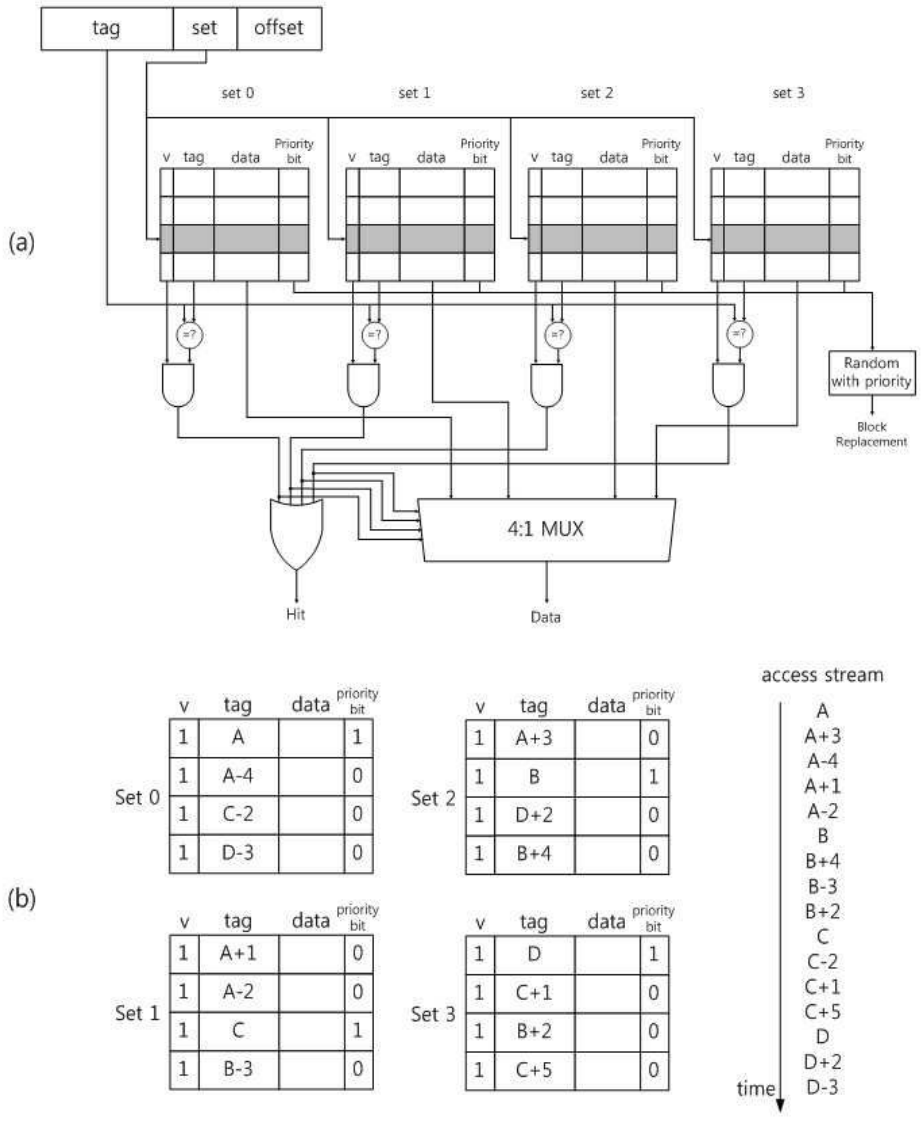
도면1



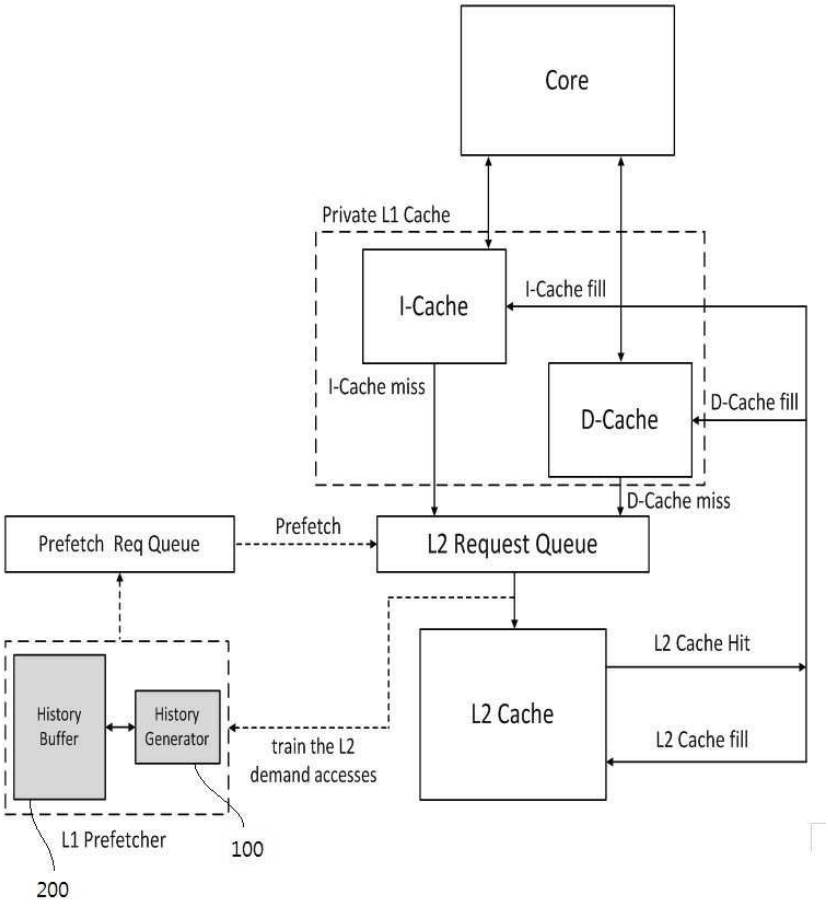
도면2


Trigger Access (230)
History Buffer (200)
220
Data/Instruction Address
Trigger Address
Displacement
TAG
210
TAG
+3 -4 +1 -2
Displacement
A-4
A+3
A
=?
Hit/miss
Prefetch Address

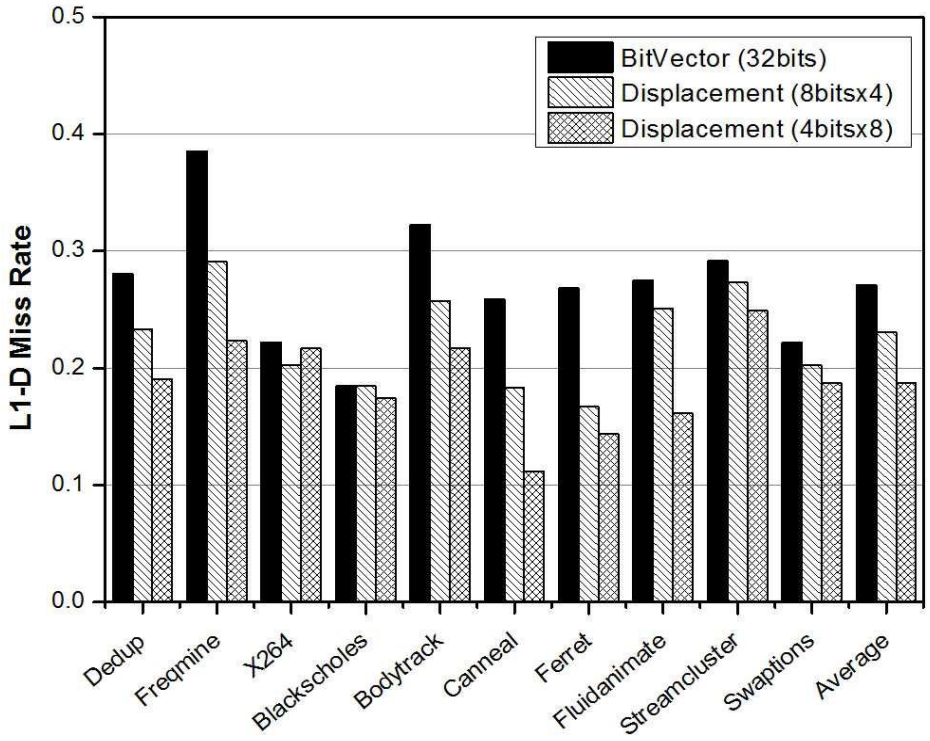
도면3


(a)
tag
set
offset
set 0
set 1
set 2
set 3
v tag data Priority bit
Random with priority
Block Replacement
4:1 MUX
Hit
Data
(b)
Set 0
Set 1
Set 2
Set 3
v tag data priority bit
1 A 1
1 A-4 0
1 C-2 0
1 D-3 0
1 A+3 0
1 B 1
1 D+2 0
1 B+4 0
1 A+1 0
1 A-2 0
1 C 1
1 B-3 0
1 D 1
1 C+1 0
1 B+2 0
1 C+5 0
access stream
A
A+3
A-4
A+1
A-2
B
B+4
B-3
B+2
C
C-2
C+1
C+5
D
D+2
D-3
time

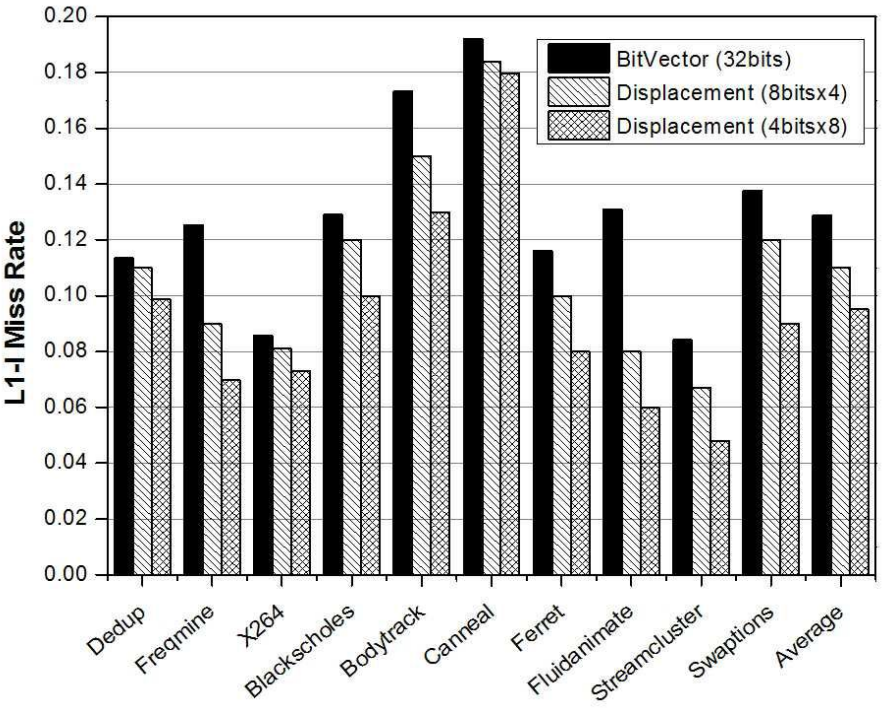
도면4


Core
Private L1 Cache
I-Cache
I-Cache fill
I-Cache miss
D-Cache
D-Cache fill
D-Cache miss
Prefetch Req Queue
Prefetch
L2 Request Queue
L2 Cache Hit
History Buffer
History Generator
train the L2 demand accesses
L2 Cache
L2 Cache fill
L1 Prefetcher
100
200

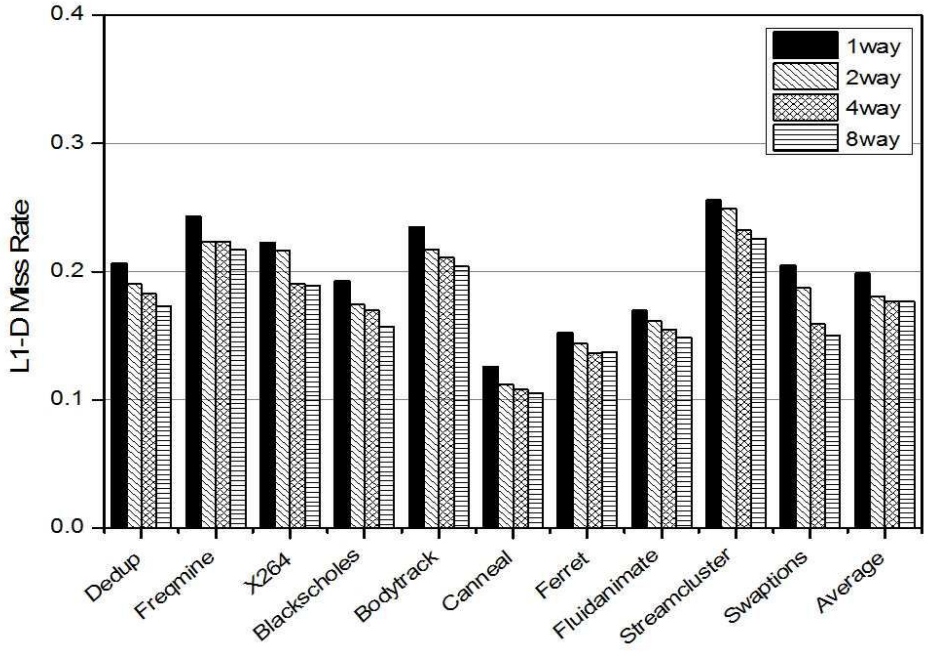
도면5


BitVector (32bits)
Displacement (8bitsx4)
Displacement (4bitsx8)
L1-D Miss Rate
0.0
0.1
0.2
0.3
0.4
0.5
Dedup
Freqmine
X264
Blackscholes
Bodytrack
Canneal
Ferret
Fluidanimate
Streamcluster
Swaptions
Average

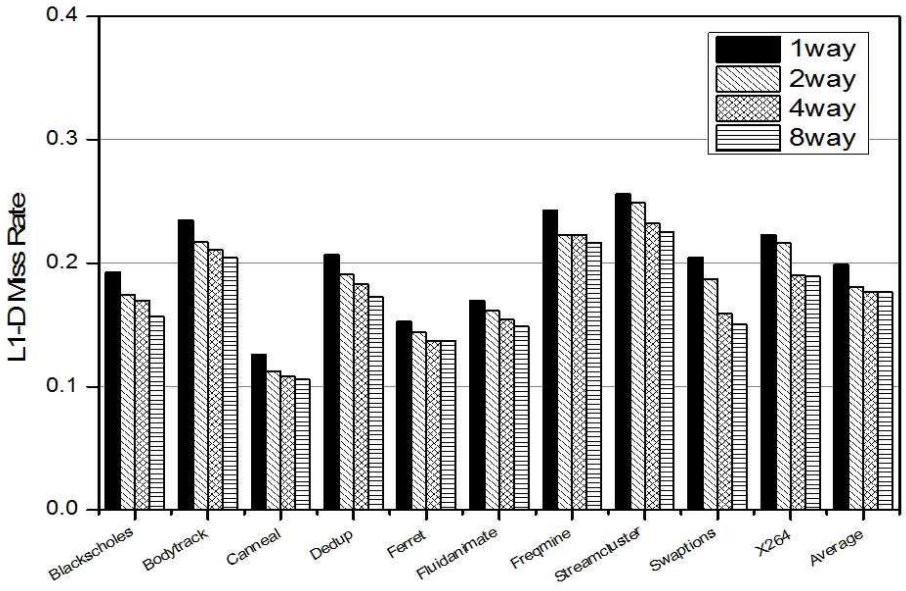
도면6


BitVector (32bits)
Displacement (8bitsx4)
Displacement (4bitsx8)
L1-I Miss Rate
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16
0.18
0.20
Dedup
Freqmine
X264
Blackscholes
Bodytrack
Canneal
Ferret
Fluidanimate
Streamcluster
Swaptions
Average

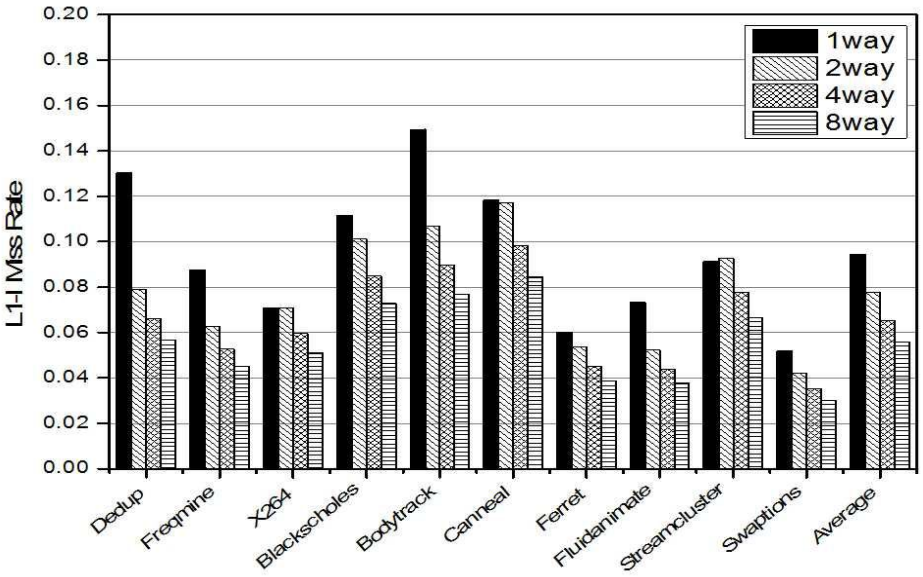
도면7


0.4
0.3
0.2
0.1
0.0
L1-D Miss Rate
1way
2way
4way
8way
Dedup
Freqmine
X264
Blackscholes
Bodytrack
Canneal
Ferret
Fluidanimate
Streamcluster
Swaptions
Average

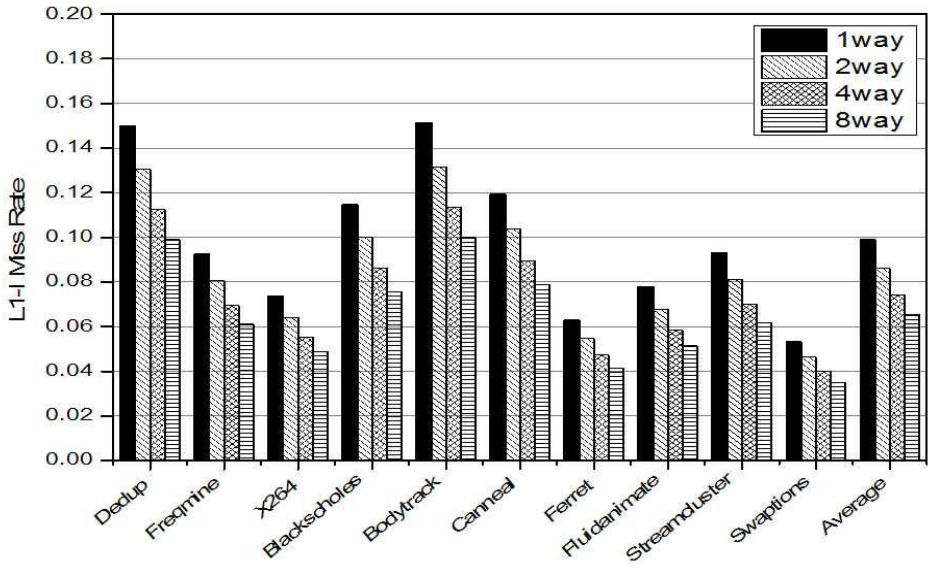
도면8


0.4
0.3
0.2
0.1
0.0
L1-D Miss Rate
1way
2way
4way
8way
Blackscholes
Bodytrack
Canneal
Dedup
Ferret
Fluidanimate
Freqmine
Streamcluster
Swaptions
X264
Average

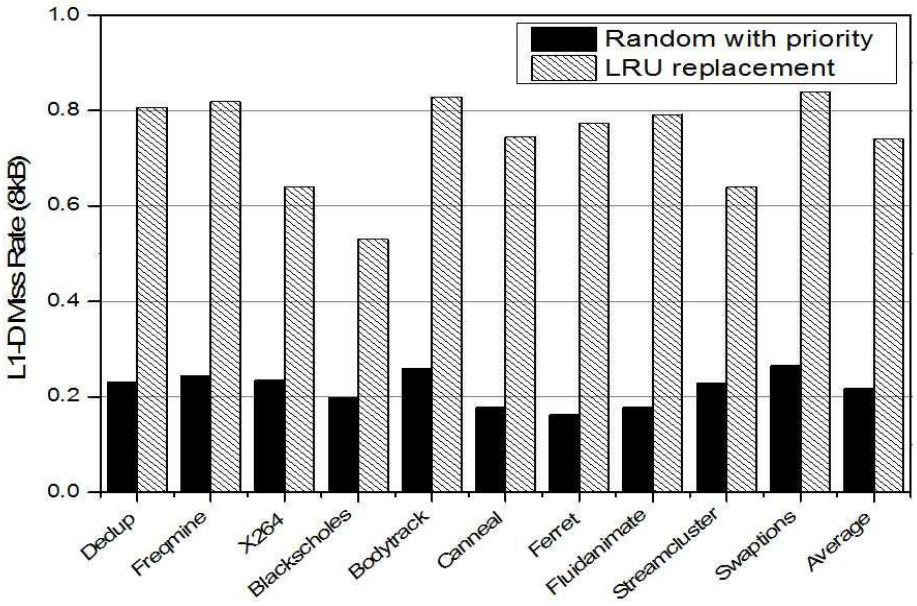
도면9


L1-I Mss Rate
0.20
0.18
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
1way
2way
4way
8way
Dedup
Freqmine
X264
Blackscholes
Bodytrack
Canneal
Ferret
Fluidanimate
Streamcluster
Swaptions
Average

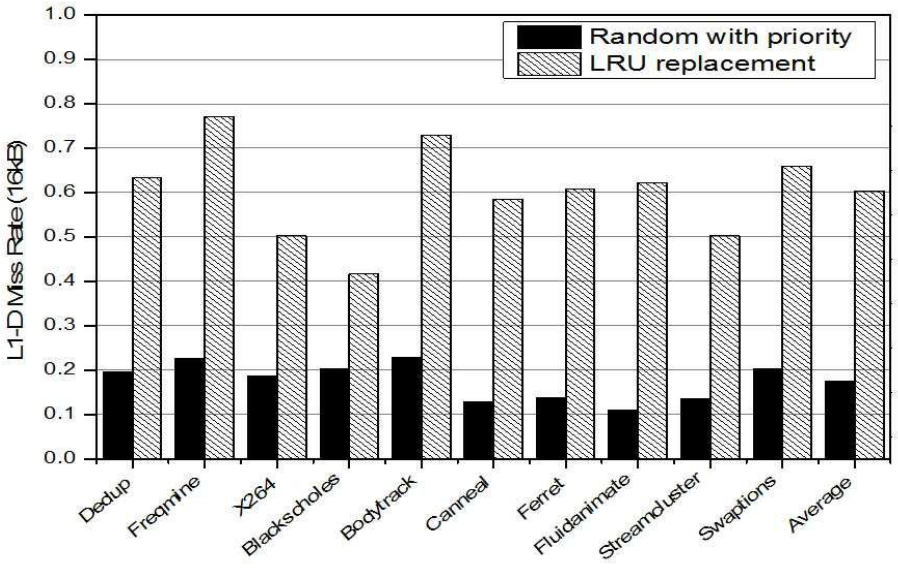
도면10


L1-I Mss Rate
0.20
0.18
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
1way
2way
4way
8way
Dedup
Freqmine
X264
Blackscholes
Bodytrack
Canneal
Ferret
Fluidanimate
Streamcluster
Swaptions
Average

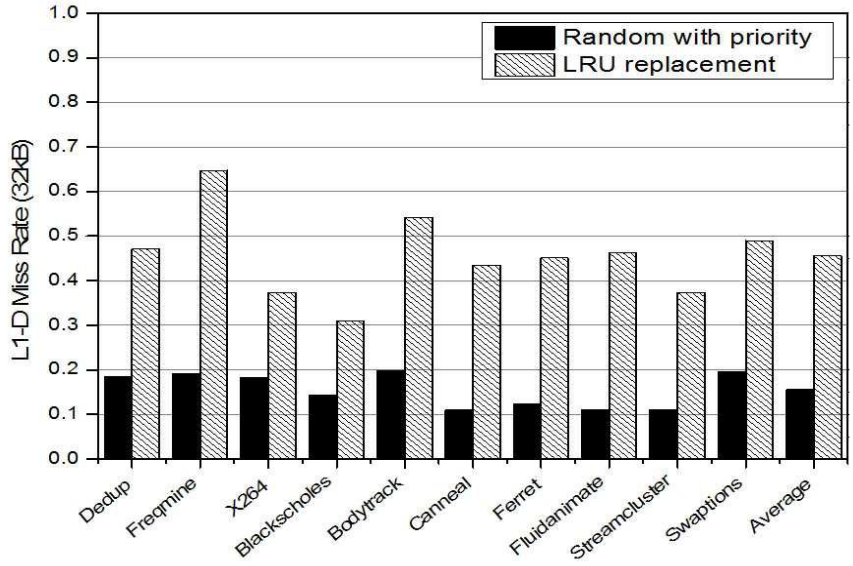
도면11


Random with priority
LRU replacement
L1-DMiss Rate (8kB)
1.0
0.8
0.6
0.4
0.2
0.0
Dedup
Freqmine
X264
Blackscholes
Bodytrack
Canneal
Ferret
Fluidanimate
Streamcluster
Swaptions
Average

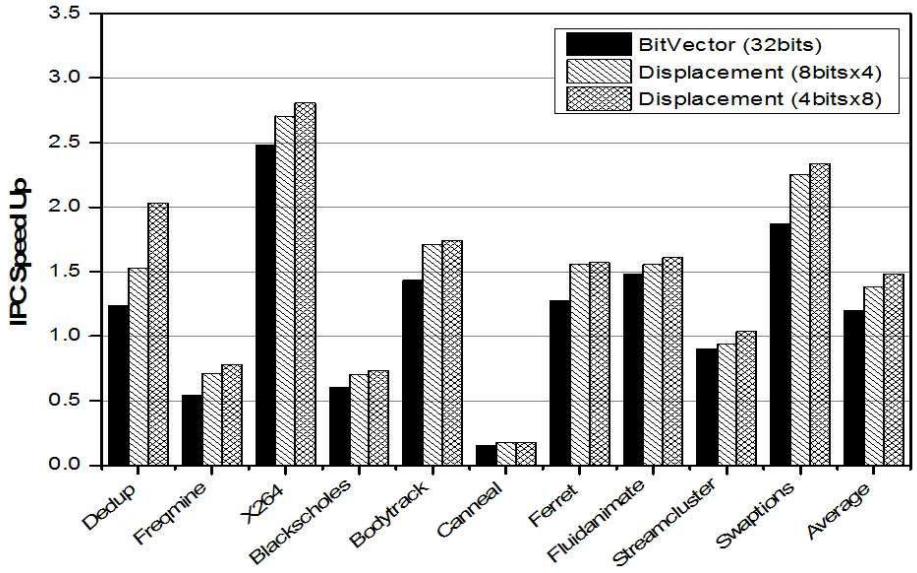
도면12


Random with priority
LRU replacement
L1-DMiss Rate (16kB)
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Dedup
Freqmine
X264
Blackscholes
Bodytrack
Canneal
Ferret
Fluidanimate
Streamcluster
Swaptions
Average

도면13


Random with priority
LRU replacement
L1-D Miss Rate (32kB)
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Dedup
Freqmine
X264
Blackscholes
Bodytrack
Canneal
Ferret
Fluidanimate
Streamcluster
Swaptions
Average

도면14


BitVector (32bits)
Displacement (8bitsx4)
Displacement (4bitsx8)
IPC Speed Up
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Dedup
Freqmine
X264
Blackscholes
Bodytrack
Canneal
Ferret
Fluidanimate
Streamcluster
Swaptions
Average