캐쉬 미스와 분기예측 실패를 고려한 명령어 페치 모델의 성능분석

(Performance Analyses of Instruction Fetch Models Considering Cache Miss and Branch Misprediction)

김 선 모 † 정 진 하 †† 최 상 방 †††
(Sun Mo Kim) (Jin Ha Jung) (Sang Bang Choi)

요 약 캐쉬 메모리는 명령어와 데이타의 참조시간을 줄이기 위하여 프로세서에 의해 참조되어질 가능성이 높은 주 메모리의 내용을 일시적으로 저장하는 용량이 작고 빠른 메모리이다. 본 논문에서는 슈퍼스칼라 프로세서에 적용될 수 있는 네 가지 명령어 캐쉬 구조에 대하여 캐쉬 미스와 분기예측 실패를 고려한 해석적 모델을 제안하고 성능을 분석하였다. 슈퍼스칼라 구조의 다양한 파라미터들을 정의하여 명령어 페치를 모델링하였으며, 해석적 모델의 타당성을 검증하기 위하여 시뮬레이션을 수행하여 얻은 결과와 비교하였다. 해석적 모델과 시뮬레이션에 의하여 얻은 명령어 페치율은 대부분의 경우 10% 오차 내에서 일치하였다. 명령어 페치율에 있어서는 분기예측 실패로 인한 영향보다는 캐쉬 미스로 인한 성능저하가 더욱 큰 것으로 나타났다. 본 연구를 통하여 얻은 해석적 모델을 사용하면 시뮬레이션에서는 드러나지 않는 성능제약의 원인에 대한 명확한 규명이 가능하며, 캐쉬 성능에 있어서 캐쉬 미스와 분기예측 실패간의 관계에 대한 정확한 분석이 가능하다.

Abstract Cache memories are small fast memories used to temporarily hold the contents of main memory that are likely to be referenced by processors so as to reduce instruction and data access time. In this paper, we represent analytical models of instruction fetch process for four types of instruction cache structures that can be used for superscalar processors. In the models, we define various kinds of architectural parameters and take cache miss and branch misprediction into consideration. To prove the correctness of the proposed models, we performed extensive simulations and compared the results with the analytical models. Simulation results showed that the proposed model can estimate the instruction fetch rate accurately within 10% error in most cases. Both analytical model and simulation show that the increase of cache misses reduces the instruction fetch rate more severely than that of branch misprediction does. However, the analytical model can explain the causes of performance degradation which cannot be uncovered by the simulation method only. The model is also able to provide exact relationship between cache miss and branch misprediction for instruction fetch analysis.

1. 서 론

슈퍼스칼라 프로세서의 목적은 사이클 당 여러 개의 명령어를 동시에 수행시키는 것이며 이를 달성하기 위해서 프로그램 내의 명령어 레벨 병렬성(instruction level parallelism)을 이용한다[1, 2]. 그러나 메모리에서 프로세서 내의 명령어 디코드 단으로 명령어들이 빠른 속도로 전달되지 않는다면 이 병렬성은 충분히 이용될 수 없다. 또한 프로세서와 메모리간의 성능 차이가 점차 커지면서 캐쉬의 성능은 전체 시스템의 성능을 결정하는 가장 중요한 요소 중의 하나가 되었으며, 이에 따라 캐쉬를 모델링하고 성능을 향상시키기 위한 연구가 활발히 진행되고 있다. 그러나 실행하고자 하는 프로그램에 조그만 수정이 가해져도 캐쉬의 동작 및 성능의 변화는 매우 커지는 불규칙한 캐쉬의 특성 때문에 캐쉬의 동작은 해석하기가 매

† 비 회 원 : (주)LG 전자 연구원
sunmokim@lge.com
†† 비 회 원 : 인하대학교 전자공학과
g2001131@inhavision.inha.ac.kr
††† 종신회원 : 인하대학교 전자전기컴퓨터공학부 교수
sangbang@inha.ac.kr
논문접수 : 2001년 3월 2일
심사완료 : 2001년 9월 5일

우 어렵다[3, 4]. 따라서 지금까지 대부분의 연구는 시뮬레이션을 기초로 한 방법으로 이루어져 왔다. 그러나 시뮬레이션을 통한 방법의 경우 얻어진 결과에 대한 정확한 해석이 어려울 뿐만 아니라, 새로운 프로세서의 설계시에 현재와 파라미터를 사용한 무수히 많은 시뮬레이션을 다시 수행해야 하는 문제가 있다[5].

이러한 단점을 극복하기 위한 접근 방법으로 슈퍼스칼라 파이프라인 내의 명령어 흐름에 대한 해석적인 모델뿐만 아니라[6], 캐쉬 동작 및 성능의 해석적인 모델에 대한 많은 연구가 이루어져 왔다[4, 7, 8, 9, 10, 11, 12, 13, 14], Wolf 등은 소스 코드 변환을 위한 컴파일러의 최적화에 사용될 수 있는 경험적 방법이 가미된 해석적 모델을 제안하였으나, 일반적인 성능 측정 모델에는 적합하지 않았다[9, 13, 14]. Lam 등은 블록화된 알고리듬(blocked algorithm)의 분석과 같이 특정 알고리듬의 캐쉬 성능을 측정하기 위한 모델을 제안하기도 하였다[8, 10, 12]. 최근에는 보다 일반적인 성능분석에 적용할 수 있는 해석적 모델이 제안되고 있으며, Ghost와 Temam의 방법은 직접매핑 캐쉬(direct mapped cache)에, Haper의 모델은 집합연관 캐쉬(set associative cache)에 각각 중점을 두었다[4, 11, 15]. Wallace와 Bagherzadeh는 캐쉬의 명령어 페치를 해석적으로 분석하고 Yeh 등이 제안한 이중 분기예측 버퍼(dual branch target buffer)를 이용하는 새로운 페치 메커니즘을 제안했다[16, 17]. 그러나, 캐쉬의 페치 과정에 가장 큰 영향을 미치는 캐쉬 미스와 분기예측 실패(branch misprediction) 등은 고려하지 않았다.

슈퍼스칼라 프로세서에서 캐쉬 메모리로부터 파이프라인의 명령어 디코드 단으로 명령어가 전달되는 일련의 과정을 명령어 페치(fetch) 또는 인출이라고 하며, 사이클 당 페치되는 명령어의 수를 명령어 페치율(fetch rate)이라고 한다. 명령어 페치 단계에서 캐쉬 미스가 발생하면 캐쉬 메모리는 명령어를 더 이상 제공할 수 없고 메모리로부터 필요한 블록이 전달 될 때까지 페치 단은 기다려야만 한다. 또한 분기명령어에 의한 페널티를 줄이기 위해 프로세서들은 분기예측 버퍼(branch prediction buffer), 분기목표 버퍼(branch target buffer), 지연분기(delayed branch), 루프 언롤링(loop unrolling) 등의 다양한 기술을 사용하나 이들의 분기예측이 항상 정확하지는 않다. 분기 예측이 실패한 경우 이후의 명령어들은 다른 경로로부터 페치되어 진다. 그러므로 명령어 페치 과정은 캐쉬 미스와 분기예측 실패의 두 가지 중요한 파라미터에 의해 제한된다.

본 논문에서는 슈퍼스칼라 프로세서에 적용될 수 있는 네 가지 명령어 캐쉬 구조에 대하여 캐쉬 미스와 분기예측 실패를 고려한 해석적 모델을 제안하고 성능을 분석하였다. 명령어 캐쉬 구조는 Wallace의 분류[17]를 참고하였다. Basic 캐쉬는 명령어를 페치하는 방법 중에서 가장 기본적인 구조로서 페치 블록의 폭과 캐쉬 라인의 크기가 같은 경우이며, 캐쉬 라인의 크기를 페치 블록의 폭의 정수 배만큼 크게 하고 세이브 버퍼(save buffer)를 두어 basic 캐쉬의 단점을 보완한 것을 extended 캐쉬라 한다. Prefetch 캐쉬는 명령어 페치 단에 선인출 버퍼(prefetch buffer)를 두어 다음 사이클의 페치 시작점을 미리 예측할 수 있는 캐쉬구조이며, 캐쉬에 메모리 뱅크의 개념을 도입하여 현재의 명령어로 다음 두 번째 목적지 주소까지 예측하는 것을 interleaved 캐쉬라 한다. 각 캐쉬 구조에 대하여 우선 캐쉬 미스가 발생하지 않고 분기예측도 완벽하다는 가정 하에 캐쉬로부터의 명령어 페치율을 계산한다[17]. 이 때 명령어 페치율을 결정하는 파라미터는 프로그램 내의 분기명령어 비율이다. 본 논문에서는 여기에 새로운 파라미터인 캐쉬 미스율, 분기예측 실패율, 캐쉬 미스 페널티, 그리고 분기예측 실패 페널티를 정의하여 이러한 성능제약 요인으로 인해 캐쉬가 인출 서비스를 하지 못함으로써 추가적으로 소비하는 사이클을 계산한 후, 원래의 명령어 페치율에 적용하여 보다 정확한 명령어 페치율을 계산한다.

제안된 해석적 모델의 타당성을 검증하기 위하여 여러 벤치마크 프로그램을 사용하여 시뮬레이션을 수행하였다. 본 논문에서 제안한 해석적 모델과 시뮬레이션을 통하여 얻은 명령어 페치율을 비교하면 대부분의 경우 10%의 오차 내에서 일치하였다. 제안된 해석적 모델과 시뮬레이션 결과로부터 basic 캐쉬와 extended 캐쉬에 비해 prefetch 캐쉬와 interleaved 캐쉬가 좋은 성능을 보여줌을 알 수 있다. 또한 슈퍼스칼라 프로세서에서 분기예측 실패가 명령어 페치율의 감소에 미치는 영향보다는 캐쉬 미스율이나 캐쉬 미스 페널티의 증가로 인한 감소가 더욱 크게 나타났다. 이러한 해석적 모델을 사용하여 얻을 수 있는 장점은 캐쉬 미스나 분기예측의 정확도 등이 각 캐쉬 구조의 성능에 미치는 영향을 정확히 분석할 수 있고, 시뮬레이션에서는 드러나지 않는 성능제약의 원인을 명확히 규명해서 프로세서의 성능향상을 위한 설계지식을 얻을 수 있다. 또한 슈퍼스칼라의 성능에 영향을 미치는 요인들을 몇 가지 파라미터들로 단순화시켜 분석함으로써 새로운 프로세서 설계시 수행해야 하는 수많은 시뮬레이션에 의한 부담을 감소시킬 수 있을 것이다.

본 논문은 다음과 같이 구성되어 있다. 제 2장에서는 슈퍼스칼라 프로세서에서의 캐쉬 동작 및 명령어 페치에 대하여 알아본다. 제 3장에서는 슈퍼스칼라 프로세서에 적용될 수 있는 네 가지 캐쉬 모델에 대하여 설명하고, 캐쉬 미스가 없고 분기예측도 완전하다는 가정 하에 명령어 페치율을 해석적으로 계산한다. 제 4장에서는 캐쉬 미스와 분기예측 실패를 고려하여 보다 정확한 해석적 모델을 고안한다. 제 5장에서는 제안한 해석적 모델의 타당성을 검증하기 위해서 수행한 시뮬레이션 방법 및 분석 결과를 설명하고, 제 6장에서는 본 논문의 결론을 맺는다.

2. 슈퍼스칼라 프로세서의 명령어 페치

2.1 명령어 인출 메커니즘

슈퍼스칼라 프로세서의 목적은 사이클 당 여러 개의 명령어를 동시에 수행시키는 것이며, 메모리로부터 프로세서의 명령어 디코드 단으로 충분한 속도로 명령어를 공급하기 위해서 고성능 프로세서는 예외 없이 캐쉬 메모리를 사용한다. 캐쉬 메모리는 프로세서가 고속으로 접근할 수 있는 메모리로서, 자주 참조되는 메모리 블록을 저장하여 이에 대한 액세스 시간을 줄이는 역할을 한다. 따라서 캐쉬는 프로세서와 메모리간의 속도 차이에서 오는 성능 저하를 막는데 매우 효과적이다.

캐쉬의 성능을 결정하는 파라미터로는 캐쉬 미스율, 캐쉬 미스 페널티, 캐쉬 히트 시간(cache hit time) 등이 있으며 각각의 파라미터를 향상시킴으로써 전체적인 캐쉬의 성능을 높이기 위한 많은 연구가 진행되고 있다. 캐쉬 미스율을 줄이기 위한 방법으로는 높은 집합연관도(higher set associative)의 사용, 빅팀 캐쉬(victim cache)의 사용, 하드웨어적인 선인출, 그리고 컴파일러에 최적화에 의한 선인출 등의 방법이 쓰인다. 또한 캐쉬 미스 페널티의 영향을 줄이기 위한 방법으로는 논블로킹 캐쉬(nonblocking cache)의 사용, 계층적 캐쉬 구조의 사용 등의 방법이 있다. 캐쉬 히트 시간은 프로세서의 클락율에 직접적으로 영향을 주므로 히트 시간을 향상시키기 위해 직접맵핑 또는 가상 캐쉬(virtual cache)를 사용한다[1].

그림 1은 슈퍼스칼라 프로세서의 명령어 페치 과정을 블록 다이어그램으로 나타낸 것이다. 캐쉬 라인(cache line) 혹은 열(row)은 캐쉬 안에 존재할 수 있는 정보의 최대 크기를 의미하며, 이는 명령어 인출기가 한 사이클에 동시에 읽을 수 있는 최대 명령어의 수를 나타낸다. 페치 블록(fetch blocks)의 폭(width) n은 캐쉬로부터 읽혀질 수 있는 순서적인 명령어들의 집합으로 정의하며, 명령어 페치 단으로부터 디코드 단으로 전달될 수 있는 명령어 수는 디코드 블록(decode blocks)의 폭 q로 정의한다. 페치 블록의 폭 n은 디코드 블록의 폭 q보다 항상 크거나 같다. 본 논문에서는 한 블록은 단지 한 개의 명령어만 포함한다고 가정한다.

명령어 캐쉬는 매 사이클마다 명령어 인출기로부터 받은 프로그램 카운터(PC) 값을 사용하여 페치 블록의 폭에 해당하는 최대 n개의 블록을 전송한다. 캐쉬 미스가 발생하는 경우 서브메모리(L2 캐쉬나 메인 메모리)로부터 원하는 라인을 읽어 다른 캐쉬 라인과 대치한다. 명령어 인출기는 다음 사이클에 인출할 새로운 명령어의 주소를 예측하고 캐쉬로부터 읽어 온 명령어를 명령어 디코더로 전달하는 기능을 담당한다. 명령어 인출기는 명령어 페치를 보다 효율적으로 하기 위해 분기예측기와 여러 기능의 버퍼로 구성될 수 있다. 명령어 디코더는 명령어 인출기로부터 디코드 블록의 폭에 해당하는 최대 q개의 블록을 받아 해석하고 분기예측의 성공 여부를 결정한다. 분기예측이 실패한 경우는 이 명령어 이후의 잘못된 패스를 통해서 얻어진 명령어들을 파이프라인으로부터 제거하고, 그 정보를 명령어 인출기로 보내어 캐쉬로부터 새로운 블록을 읽어 들이도록 한다.

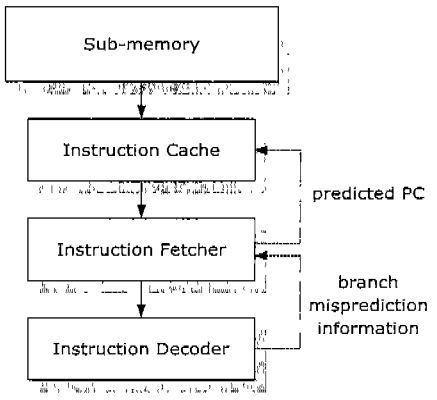

그림 1 슈퍼스칼라 프로세서에서의 명령어 인출 과정

2.2 명령어 페치의 성능 제한

명령어 페치의 동작은 캐쉬 미스와 분기예측 실패의 두 가지 중요한 요인으로 인하여 크게 제한을 받는다. 명령어 인출기로부터 요구된 명령어가 캐쉬 내부에 존재하지 않아서 서브메모리로부터 읽어들이는 동안 서비스가 불가능한 상태를 캐쉬 미스라고 하며, 이로 인해 명령어 페치가 지연되는 시간을 캐쉬 미스 페널티라고 한다. 슈퍼스칼라 프로세서에서 대부분의 명령어들은 캐쉬로부터

높은 히트율로 읽혀진다. 이것은 분기나 점프 명령을 제외한 대부분의 명령어들은 한 주소로부터 연속적으로 읽혀지기 때문이다. 그러나 현재 많이 사용되고 있는 멀티태스킹(multitasking)과 같은 운영체제 기술이 적용될 경우 서로 다른 프로그램들이 동시에 수행되므로, 프로그램들간의 문맥전환(context switch)이 자주 발생하여 명령어 캐쉬 미스가 증가될 수 있다. 또한 캐쉬 미스 페널티는 프로세서와 메모리간의 속도 차이에 비례하여 커지므로, 미스가 발생한 명령어를 먼저 읽어오는 방법(critical word first), 논블로킹 캐쉬, 다중레벨 캐쉬 등을 사용하여 페널티를 어느 정도 줄일 수는 있으나, 미스 페널티는 슈퍼스칼라 프로세서의 성능에 심각한 영향을 미치게 된다. 따라서 명령어 페치율의 계산에서 캐쉬 미스로 인한 영향은 가장 중요한 요소 중의 하나다.

분기나 점프 명령어 사이의 다른 명령어들은 순차적으로 수행되며, 이들 명령어의 수를 수행길이(run length)라고 한다. 그러나 수행길이는 일반적으로 매우 짧으므로, 대부분의 슈퍼스칼라 프로세서들은 분기예측 버퍼(branch prediction buffer), 분기목표버퍼(branch target buffer), 지연분기(delayed branch)등의 여러 방법을 사용하여 분기명령에 의해 발생되는 성능저하를 줄이고 있다. 그러나 어떤 분기예측기법도 항상 정확하지 않으며, 분기예측이 실패하면 잘못된 분기명령어 이후의 모든 명령어는 무효화시키고 정확한 목적지 주소로부터 다시 명령어를 읽어들여야 한다. 이러한 과정에서 프로세서는 명령어의 흐름을 더 이상 진행시킬 수 없으므로 명령어 페치의 모델링에서 간과해서는 안되는 또 다른 파라미터가 분기예측 실패율(branch misprediction rate)과 분기예측 실패 페널티(branch misprediction penalty)이다[1, 18].

그림 1과 같은 명령어 페치 과정에서 캐쉬 라인의 크기는 n이고, 임의 명령어가 분기명령어일 확률은 B라고 한다. 한 캐쉬 라인에서 연속적으로 수행 가능한 명령어의 평균길이를 구하기 위해서, 캐쉬 라인의 n개의 명령어 중 적어도 한 개 이상의 분기명령어가 포함되는 경우와 그렇지 않은 경우로 나눈다. 라인 내에 분기명령어가 하나도 없다면 전체 n개의 명령어는 모두 연속적으로 수행되고 그 확률은 $(1-B)^n$이다. 라인 내에 분기명령어가 있는 경우, i개의 명령어가 순차적으로 수행될 확률은 $(1-B)^{i-1} \cdot B$이다. 따라서 명령어의 평균 수행길이 $L(n, B)$는 다음과 같이 얻어진다.

$$L(n, B) = n(1-B)^n + \sum_{i=1}^{n} i(1-B)^{i-1}B = \frac{1-(1-B)^n}{B} \quad (1)$$

캐쉬 라인의 크기 n을 늘려 무한대로 한다면 $L(n, B)$의 극한값은 다음과 같이 계산된다. 이것은 프로그램의 모든 명령어들이 한개의 캐쉬 라인에 포함되는 경우 얻을 수 있는 값이다.

$$\lim_{n \to \infty} L(n, B) = \frac{1}{B} \quad (2)$$

식 (2)는 소프트웨어적인 스케줄링이나 하드웨어적인 예측 기법과 상관없이 명령어의 평균 수행길이는 분기명령어의 확률에 의해서 제한됨을 보여준다. 그러므로 $1/B$는 명령어 인출기가 한 사이클에 페치하여 수행할 수 있는 최대 평균 명령어의 길이를 나타낸다.

3. 명령어 페치의 해석적 모델

3장과 4장에서는 슈퍼스칼라 프로세서에 적용될 수 있는 네 가지 타입의 캐쉬 구조에 대하여 설명하고 각 타입에 대한 명령어 페치의 해석적 모델을 제안한다. 여기서는 Wallace[17]의 해석방법을 참조하여 캐쉬 미스가 발생하지 않고 분기예측도 항상 정확하다는 가정 하에 각 캐쉬 구조의 성능을 분석하며, 다음 장에서는 이를 확장하여 캐쉬 미스와 분기예측 실패를 고려한 명령어 페치의 정확한 해석적 모델을 제시한다.

3.1 Basic 캐쉬

Basic 캐쉬의 명령어 인출기는 그림 2와 같이 디코더로 전달하기 위한 명령어들을 저장하는 버퍼와 분기예측기로 구성된다. 명령어 캐쉬로부터 명령어를 페치하는 방법 중에서 가장 기본적인 구조로서 페치 블록의 폭과 캐쉬 라인의 크기가 같은 경우이다. 명령어 인출기는 캐쉬 라인 전체를 한 사이클에 검색하여 명령어 디코드 단으로 전달한다. 그러나 캐쉬 라인 내에 분기명령어가 존재한다면, 명령어 인출기는 분기명령이 이후의 명령어들을 무효화시키고 분기명령어까지만 명령어 디코드 단으로 전달한다.

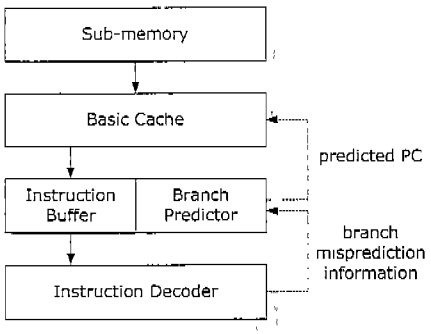

그림 2 Basic 캐쉬의 명령어 인출 과정

그림 3은 캐쉬 라인의 크기와 페치 블록의 폭을 4로 가정한 basic 캐쉬의 명령어 인출 예를 도식화한 것이다. 첫 번째 사이클에서 명령어 인출기는 라인 10의 두 번째 명령어를 시작점으로 명령어를 인출한다. 라인 10의 세 번째 명령어는 분기명령어이므로 인출기는 마지막 네 번째 명령어를 무효화시킨 후, 두 개의 명령어만을 명령어 디코드 단으로 보낸다. 두 번째 사이클에서는 분기예측기에 의하여 얻어진 분기명령어의 목적지 주소를 시작점(라인 20의 두 번째 명령어 add)으로 하여 라인 20의 명령어들을 읽어 검색한다. 라인 20에는 분기명령어가 없으므로 두 번째 이후의 세 명령어들은 모두 페치된다.

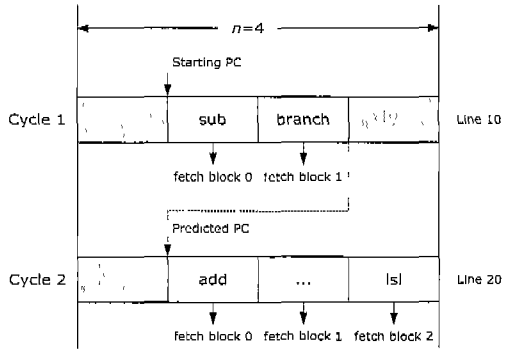

그림 3 basic 캐쉬의 명령어 인출 예

캐쉬 라인의 크기가 n이고 분기명령어의 확률이 B일 때 임의의 캐쉬 라인에서 분기가 일어날 확률은 $C(n, B)$로 나타내고, 한 캐쉬 라인 안에서 페치 시작점의 위치가 j번째 블록일 확률은 $S_j(n, B)$로 표시한다. 한 사이클에서 분기가 발생하면 다음 사이클에서 수행되는 명령어는 확률적으로 목적지 라인의 n개 블록 중 어디에서나 시작할 수 있다. 그러나 분기가 일어나지 않는다면 다음 라인의 첫 번째 블록에서 명령어 수행이 계속된다. 따라서 첫 번째 블록을 제외한 나머지 블록에서 명령어 페치가 시작될 확률은 $C(n, B)/n$이 되며, 첫 번째 블록은 이 확률에 분기가 발생하지 않을 확률인 $1-C(n, B)$을 더하여야 한다.

$$S_1(n, B) = \{1-C(n, B)\} + \frac{C(n, B)}{n} = 1 - \frac{n-1}{n} C(n, B)$$
$$S_j(n, B) = \frac{C(n, B)}{n}, \quad 2 \le j \le n \tag{3}$$

캐쉬 라인의 i번째 블록에서 분기가 발생한다면, 해당 라인에서 명령어의 수행은 첫 번째에서 i번째 사이의 한 블록에서 시작되었을 것이다. 따라서 i번째 블록에서 분기가 발생할 확률을 $c_i(n, B)$로 나타내면, 캐쉬 라인 전체에서 분기가 발생할 확률 $C(n, B)$는 각 위치 i에서 분기가 발생할 확률의 합과 같으므로 다음과 같이 구할 수 있다.

$$c_i(n, B) = \sum_{j=1}^{i} B(1-B)^{i-j} \cdot S_j(n, B) \tag{4}$$

$$C(n, B) = \sum_{i=1}^{n} c_i(n, B) = \frac{n}{1/B + n - 1} \tag{5}$$

Basic 캐쉬에서의 명령어 페치율은 페치의 시작점과 그 이후에 연속적으로 수행되는 명령어의 길이에 의하여 결정되므로 다음같이 얻어진다.

$$F_{basic}(n, B) = \sum_{i=1}^{n} S_i(n, B) \cdot L(n-i+1, B) = \frac{n}{1+B(n-1)} \tag{6}$$

3.2 Extended 캐쉬

Extended 캐쉬는 캐쉬 라인의 명령어 수행길이가 페치 블록의 폭만큼 크게 될 기회를 증가시키도록 설계된다. 명령어 수행길이를 증가시키기 위한 방법은 캐쉬 라인의 크기(m)을 페치 블록의 폭(n)보다 크게($m>n$) 하는 것이다. 그림 4는 extended 캐쉬의 명령어 페치 과정을 보인 것이다. 인출기는 m 블록의 캐쉬 라인을 읽어 최대 n개의 연속적인 명령어를 인출하며, 읽어온 캐쉬블록의 손실을 막기 위해 최대 $n-1$개의 명령어를 세이브 버퍼에 저장할 수 있다. 다음 사이클에서는 버퍼에 저장되어 있던 명령어와 캐쉬로부터 새로 읽어온 명령어를 적절히 결합하여 명령어 디코더로 전달한다.

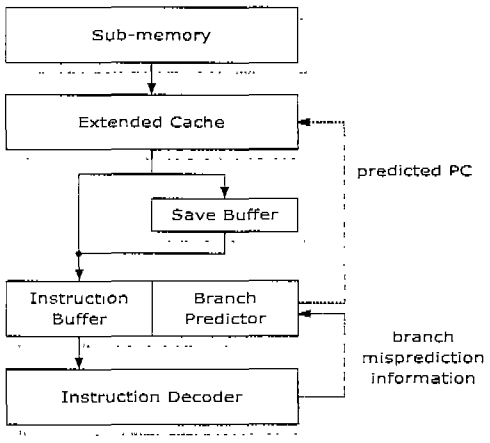

그림 4 Extended 캐쉬의 명령어 인출 과정

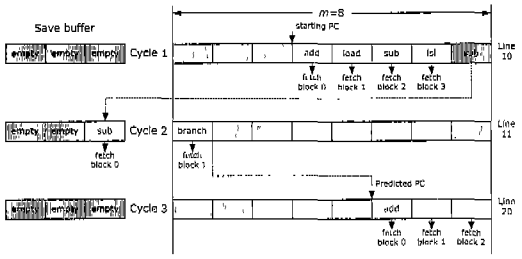

그림 5 Extended 캐쉬의 명령어 인출 예

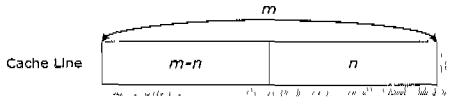

그림 6 Extended 캐쉬에서 라인의 분할

그림 5는 n=4이고, m=8인 extended 캐쉬의 명령어 인출 예를 보이고 있다. 첫 번째 사이클에서 라인 10의 네 번째 명령어를 시작점으로 인출을 시작한다. n은 4이므로 7번째 명령어인 lsl까지의 4개의 명령어가 디코드 단으로 전달되며 마지막 명령어(sub)는 세이브 버퍼에 저장된다. 다음 사이클에서는 버퍼에 저장되어 있는 명령어와 다음 라인(라인 11)의 첫 번째 명령어부터 결합하여 디코드 단으로 전달한다. 그러나 라인 11의 첫 번째 명령어가 분기명령어이므로 이후의 명령어는 무효화되고, 모두 두 개의 명령어만이 전달된다. 만약 라인 11의 처음 세 블록이 분기명령어가 아니라면 다음 사이클에서는 라인 11을 다시 읽게되므로 세이브 버퍼에는 나머지 명령어를 저장할 필요가 없다.

Extended 캐쉬에서 명령어의 페치율을 분석하기 위해서 캐쉬 라인을 그림 6과 같이 크기 m-n과 n의 두 부분으로 분할할 수 있다. 크기가 m-n인 첫 번째 블록의 경우 명령어 인출기는 페치 시작점의 위치와는 상관없이 n개의 명령어를 검색할 수 있으므로 명령어 페치율은 식 (1)을 그대로 적용할 수 있다. 크기가 n인 두 번째 블록의 경우는 basic 캐쉬와 같은 방법으로 동작하므로 식 (6)의 페치율을 따르면 전체 명령어 페치율은 다음과 같이 구할 수 있다.

$$F_{extended}(n, B, m) = \frac{m-n}{m} L(n, B) + \frac{n}{m} F_{basic}(n, B)$$
$$= \frac{(m-n)}{m} \cdot \frac{(1-(1-B)^n)}{B} + \frac{n}{m} \frac{n}{(1+B(n-1))} \quad (7)$$

3.3 Prefetch 캐쉬

선인출(prefetch) 기법은 최근 대부분의 프로세서에서 명령어 페치율을 높이기 위해 사용되며, 본 논문에서는 명령어 인출기 내에 선인출 버퍼(prefetch buffer)를 두어 분기예측기가 이를 참조하여 다음 사이클의 페치 시작점을 결정하는 기술을 모델링한다. Extended 캐쉬의 세이브 버퍼와 같이 한 사이클에서 사용하고 남은 명령어는 선인출 버퍼에 저장한다. 그림 7은 prefetch 캐쉬의 명령어 인출과정을 나타내며, 분기예측기는 다음 사이클의 페치 시작점을 결정할 때 선인출 버퍼 내에 명령어가 있으면 이를 참조하여 다음 목적지 주소를 예측한다.

그림 8은 prefetch 캐쉬의 명령어 인출 예를 보인다. 첫 번째 사이클에서 명령어 인출기는 라인 10의 첫 번째 명령어부터 네 번째 명령어까지를 명령어 디코드 단으로 전달하며 나머지 명령어를 선인출 버퍼에 저장한다. 두 번째 사이클에서 분기예측기는 선인출 버퍼내의 분기명령어를 검색하고 그 목적지 주소를 예측하여, 라인 20의 첫 번째 명령어까지를 결합한 후 명령어 디코드 단으로 전달한다. 그림 9는 그림 8과 같은 상황에서 extended 캐쉬의 동작을 설명한 것이다. 첫 번째 사이클에서 라인 10의 네 번째 명령어까지를 디코드 단으로 전달하고 세이브 버퍼 내에는 명령어를 저장하지 않는다. 두 번째 사이클에서는 라인 10을 다시 읽어 분기명령어를 포함한 세 명령어를 디코더로 전달한다. 그러나 마지막 명령어는 무효화시킨다. 세 번째 사이클에서 분기예측기는 세이브 버퍼 내에 저장된 명령어가 없으므로 두 번째 사이클의 branch 명령어를 이용하여 목적지 주소를 예측한다.

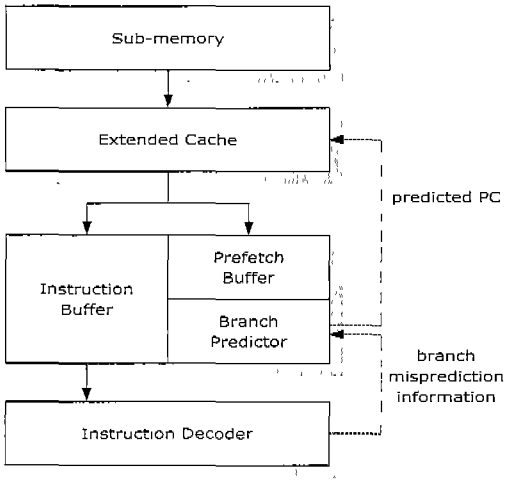

그림 7 Prefetch 캐쉬의 명령어 인출 과정

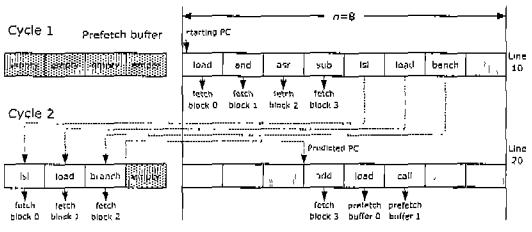

그림 8 Prefetch 캐쉬의 명령어 인출 예

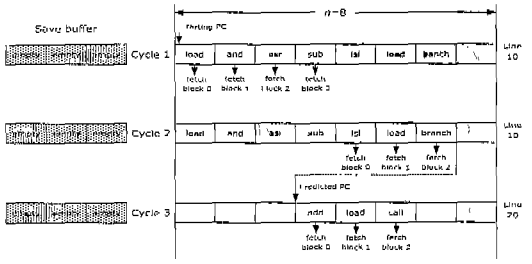

그림 9 그림 8과 같은 예를 Extended 캐쉬에 적용한 경우

Extended 캐쉬와는 달리 prefetch 캐쉬는 그림 8에서 보듯이 선인출 버퍼 내의 분기명령어를 참조하여 분기 예측을 할 수 있으므로 명령어의 페치율은 다음과 같이 구할 수 있다.

$$F_{prefetch}(n, B, m) = \frac{m-n}{m} L(n, B) + \frac{n}{m} L(n, B) = L(n, B) = \frac{1-(1-B)^n}{B} \quad (8)$$

3.4 Interleaved 캐쉬

그림 10과 같이 캐쉬 구조에 메모리 뱅크의 개념을 도입한 것이 interleaved 캐쉬이다[19]. 명령어 인출기

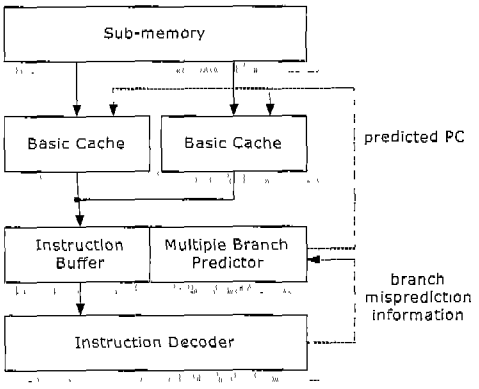

그림 10 Interleaved 캐쉬의 명령어 인출 과정

는 다중 분기예측기법을 사용하여 현재의 PC로부터 다음 두개의 목적지 주소를 예측하며, 독립적인 두개의 basic 캐쉬에 접근하여 명령어를 인출한다. Wallace는 이중 분기목적버퍼(dual branch target buffer)를 이용하여 interleaved 캐쉬를 연구하였으며, 명령어 페치율이 최대 2배까지 증가됨을 보였다.

그림 11은 interleaved 캐쉬의 명령어 인출 예를 보인다. 다중 분기예측기로부터 다음 두 개의 목적지 주소가 얻어지고, 동일 사이클에 이는 각각 캐쉬 뱅크 0과 1에 전달된다. 각 독립적인 캐쉬 뱅크는 해당되는 목적지 주소로부터 명령어를 인출한다. 뱅크 0의 라인 10으로부터 add, branch, 뱅크 1의 라인 20으로부터 lsl, jump의 네 명령어를 결합하여 디코드 단으로 전달한다. 다음 사이클에서 다중 분기예측기는 마지막 jump명령어를 이용하여 다음 두 개의 새로운 목적지 주소를 다시 예측한다. 캐쉬 뱅크를 독립적인 basic 캐쉬로 본다면 명령어 페치율은 basic 캐쉬 페치율의 두 배와 같으며 최대 페치 블록(n)의 폭만큼 인출될 수 있다.

$$F_{interleaved}(n, B) = 2F_{basic}(n, B) \quad (\leq n) \quad (9)$$

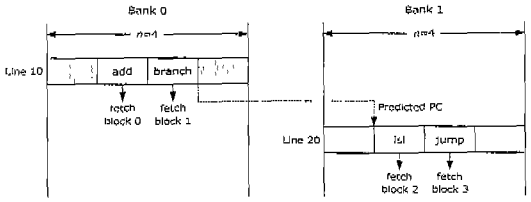

그림 11 Interleaved 캐쉬에서 명령어 인출 예

4. 캐쉬 미스와 분기예측 실패를 고려한 해석적 모델

명령어 캐쉬에서 캐쉬 미스가 발생하거나 분기예측이 실패한다면 명령어들은 미스페널티에 해당하는 사이클 동안 더 이상의 명령어를 페치를 할 수 없다. 본 장에서는 캐쉬 미스와 분기예측 실패로 의해 추가적으로 소비되는 클락 사이클을 고려한 해석적 모델을 고안한다.

4.1 Basic 캐쉬와 Extended 캐쉬

캐쉬 미스와 분기예측 실패를 고려한 명령어 페치의 해석적 모델을 위하여 우선 다음과 같은 파라미터를 정의한다. 다음 파라미터들 중에서 분기명령어의 빈도, 캐쉬 미스율, 분기예측 실패율 등은 모든 명령어에 동일하게 적용된다고 가정한다.

B : 프로그램 내에서 분기명령어의 빈도

R_C : 캐쉬 미스율

R_B : 분기예측 실패율

P_C : 캐쉬 미스 페널티

P_B : 분기예측 실패 페널티

C_i : case(i)에 대하여 추가로 소비되는 사이클

표 1은 명령어의 페치 과정을 따라 추가적으로 소요되는 사이클을 계산하기 위해 고려해야 할 사항들을 캐쉬 액세스 방식과 캐쉬 미스 발생여부, 그리고 분기예측 성공여부에 따라 분류한 것이다. 각 경우 case(i)에 대하여 추가적으로 소비되는 사이클을 구하면 다음과 같다. 우선 case(1)은 연속적인 캐쉬 액세스가 이루어지고 목적 명령어가 캐쉬에 존재하는 경우이고, case(3)은 분기명령어에 의한 비순차적으로 캐쉬를 액세스하나 캐쉬 히트가 발생하고 분기예측이 정확한 경우이다. 따라서 case(1)과 case(3)에 해당하는 경우에는 추가로 소비되는 사이클이 없으므로 앞장에서 구한 해석적 모델을 수정 없이 그대로 적용할 수 있다. 따라서 $C_1 = C_2 = 0$ 이다.

표 1 캐쉬 미스와 분기예측 실패를 고려한 명령어 인출의 여러 경우

Sequential Access		Nonsequential Access					
		CH			CM		
CH	CM	CBP	WBP		CBP	WBP	
			CH	CM		CH	CM
-	Consider P_C	-	Consider P_B	Consider P_C, P_B	Consider P_C	Consider P_C, P_B	Consider P_C, P_B
case (1)	case(2)	case(3)	case(4)	case(5)	case(6)	case(7)	case(8)

CH: cache hit, CM: cache miss
CBP: correct branch prediction,
WBP: wrong branch prediction

case(2)의 경우, 명령어 인출기는 연속적으로 다음 캐쉬 라인을 액세스하나 해당 라인이 캐쉬에 존재하지 않아 미스가 발생한 경우로, 서브메모리로부터 요구되는 라인이 읽혀지는 동안 명령어 페치가 이루어지지 못한다. 따라서, 해당 명령어를 액세스하기 위하여 소비되는 캐쉬 미스 페널티를 고려해야 한다.

$$C_2 = (1-B) \cdot R_C \cdot P_C \qquad (10)$$

case(4)의 경우, 분기명령어에 의해 비순차적으로 액세스되고 요구되는 목적 명령어는 캐쉬에 존재한다. 그러나 인출기는 분기예측이 실패하여 다시 캐쉬로부터 읽어들이는 경우이다. 이 경우 분기예측 실패로 인해서 추가로 소비되는 사이클은 다음과 같다.

$$C_4 = B \cdot (1-R_C)^2 \cdot R_B \cdot P_B \qquad (11)$$

case(5)에 해당하는 경우는 비순차적 명령어 액세스와 캐쉬 히트가 발생하지만 분기예측이 실패한다. 따라서 새로운 목적지 주소를 사용하여 다시 캐쉬를 액세스하지만 미스가 발생하여 서브 메모리로부터 해당 라인을 읽어야 하는 경우이다. 따라서, 분기예측의 실패로 인한 페널티와 캐쉬 미스로 인한 페널티를 동시에 고려해 주어야 한다. 이 경우에 해당하는 명령어에 의해서 추가로 소비되는 사이클은 다음과 같다.

$$C_5 = B \cdot (1-R_C) \cdot R_B \cdot R_C \cdot (P_C + P_B) \qquad (12)$$

case(6)의 경우는 분기명령어를 포함하고 예측된 명령어가 캐쉬에 존재하지 않아 해당 라인을 서브메모리로부터 읽어올 때까지 명령어 페치가 계속되지 못한다. 그러나 명령어의 분기예측은 성공하므로 단지 캐쉬 미스 페널티만 고려하면 된다.

$$C_6 = B \cdot R_C \cdot (1-R_B) \cdot P_C \qquad (13)$$

case(7)에서는 분기명령에 의해 비순차적으로 액세스하며 해당 라인이 캐쉬에 존재하지 않아 서브메모리로부터 읽어온다. 그러나 분기명령어를 사용한 분기예측이 실패하게 되며 새로운 목적지 주소의 명령어는 캐쉬에서 바로 서비스가 된다. 이 경우는 캐쉬 미스 페널티와 분기예측 실패로 인한 페널티를 모두 고려하여야 한다. 따라서 추가적으로 소비되는 사이클은 다음과 같이 구할 수 있다.

$$C_7 = B \cdot R_C \cdot R_B \cdot (1-R_C) \cdot (P_B + P_C) \qquad (14)$$

case(8)은 캐쉬가 서비스하지 못하는 시간이 가장 긴 경우이다. 비순차적인 명령어 액세스는 잘못된 분기예측에 의하여 이루어지며, 또한 새로운 목적지 주소는 캐쉬 미스를 유발한다. 분기예측에 의해서 얻어진 목적지 주소의 명령어가 캐쉬에 있지 않아 해당 라인이 서브메모리로부터 전달될 때까지 기다린다. 그러나 분기예측이 잘못되어 새로 결정된 주소로 캐쉬를 액세스하나 요구되는 라인이 존재하지 않아, 서브메모리로부터의 전달될 때까지 또 기다리게 된다. 이 경우 두번에 걸친 캐쉬 미스 페널티와 한번의 분기예측 실패 페널티가 고려되어야 한다. 이 경우에 추가로 소비되는 사이클은 다음과 같다.

$$C_8 = B \cdot R_C^2 \cdot R_B \cdot (2P_C + P_B) \qquad (15)$$

명령어 인출기가 요구하는 명령어가 캐쉬에 존재하고 분기예측도 정확하다면 case(1)과 case(3)의 경우와 같이 명령어 페치 과정에서 추가적으로 소비되는 사이클은 없다. 그러나 나머지의 경우에는 식 (10)에서 (15)까지와 같이 추가로 소비되는 사이클이 있으므로, 정확한 명령어 페치율을 계산하기 위해서는 이를 고려해야한다.

$$C_{total} = \sum_{i=1}^{8} C_i = C_2 + C_4 + C_5 + C_6 + C_7 + C_8 \quad (16)$$

여기서 C_{total}은 페치 블록을 액세스하기 위한 추가적으로 소비되는 사이클을 의미한다. 식 (6)과 식 (7)에서 얻은 명령어 페치율은 매 사이클당 평균적으로 인출될 수 있는 명령어의 수를 나타내므로, 캐쉬 미스와 분기예측 실패로 인하여 추가적으로 소요되는 사이클을 고려하면 basic 캐쉬와 extended 캐쉬의 평균 명령어 페치율은 다음과 같이 수정할 수 있다.

$$F^{m}_{original} = \frac{F_{original}}{1 + C_{total}} \quad (17)$$

4.2 Prefetch 캐쉬

Prefetch 캐쉬에서는 표 1의 각 경우에 따라 다른 페치율을 적용해야한다. case(1)과 case(3)에서는 페널티가 없으므로 이 두 경우에 해당하는 확률과 식 (8)을 이용하여 구할 수 있다.

$$\begin{aligned} F_{case(1)+case(3)} &= [(1-B)(1-R_C) \\ &\quad + B(1-R_C)(1-R_B)] \cdot F_{prefetch} \\ &= W \cdot F_{prefetch} \end{aligned} \quad (18)$$

식 (18)에서 첫 번째 항은 case(1)의 확률, 두 번째 항은 case(2)의 확률을 나타내며, 이 두 확률의 합은 W로 나타낸다.

그러나 분기예측이 실패하거나 캐쉬 미스가 발생한 경우는 extended 캐쉬의 세이브 버퍼와 같이 선인출 버퍼 내에 있는 명령어만이 디코더로 전달된다. 따라서, case(1)과 case(3)을 제외한 경우에는 extended 캐쉬의 페치율을 적용하여야 하며, 캐쉬 미스나 분기예측 실패로 인해 추가로 소비되는 사이클도 고려해 주어야한다.

$$F_{case(2)+\sum_{i=4}^{8} case(i)} = (1-W) \cdot \frac{F_{extended}}{1 + C_{total}} \quad (19)$$

따라서, 모든 경우를 고려한 prefetch 캐쉬의 평균 페치율은 다음과 같다.

$$F^{m}_{extended} = W \cdot F_{prefetch} + (1-W) \cdot \frac{F_{extended}}{1 + C_{total}} \quad (20)$$

4.3 Interleaved 캐쉬

표 2와 3은 interleaved 캐쉬에서 순차적인 액세스와 비순차적인 액세스 각각에 대하여 캐쉬 미스, 분기예측 정확도에 따라 명령어 페치 과정을 여러 경우로 나누어 나타낸 것이다. Interleaved 캐쉬에서는 두 번째 뱅크에서 캐쉬 미스가 발생하는 경우 basic 캐쉬의 페치율을 적용하여야 한다. 표 2와 3의 모든 경우는 다음과 같이 네 가지 그룹으로 크게 구분할 수 있다.

첫 번째 그룹(W_I)은 basic 캐쉬의 페치율을 적용하는 경우이며, case(2), (3), (7), (9)를 포함한다.

예를 들어 case (7)의 경우를 보면, 두 뱅크 모두에서 캐쉬 히트가 발생하고 뱅크 0에서는 분기예측이 성공하지만 뱅크 1에서는 분기예측이 실패한다. 따라서, 뱅크 0에서의 명령어만이 인출되어 마치 basic 캐쉬와 같이 동작한다. 이러한 그룹에 해당하는 확률은 다음과 같이 계산된다.

$$\begin{aligned} W_I &= W_{2,3,7,9} \\ &= (1-B)(1-R_C)^2 R_B + (1-B)(1-R_C)R_C \\ &\quad + B(1-R_C)^2(1-R_B)R_B + B(1-R_C)R_C(1-R_B) \end{aligned} \quad (21)$$

두 번째 그룹(W_{II})은 basic 캐쉬의 페치율을 사용하나 추가로 소비되는 사이클을 고려하여야 하는 경우이며, case(5), (12)가 여기에 속한다. Case(5)의 경우는 순차적인 명령어 액세스이지만 뱅크 0에서 캐쉬 미스가 발생하여 페널티동안 캐쉬 라인을 읽을 수 없다. 새로운 라인을 서브메모리로부터 전달받은 후에 이를 인출하지만, 뱅크 1의 분기예측이 실패하여 뱅크 0에서 인출된 명령어만이 유효하게 된다. 결과적으로 basic 캐쉬와 같이 동작하나, basic 캐쉬의 페치율과 함께 추가 소비되는 사이클을 고려해주어야 한다. 다음 식에서 C_5, C_{12}는 case(5)와 case(12)에서 추가로 소비되는 사이클을 각각 나타낸다.

$$\begin{aligned} W_{II} &= W_{5,12} = (1-B) \cdot R_C \cdot R_B + B \cdot R_C \cdot (1-R_B) \cdot R_B \\ C_5 &= (1-B) \cdot R_C \cdot R_B \cdot P_C \\ C_{12} &= B \cdot R_C \cdot (1-R_B) \cdot R_B \cdot P_C \end{aligned} \quad (22)$$

세 번째 그룹(W_{III})은 interleaved 캐쉬의 페치율을 그대로 적용하는 경우이다. Case(1)과 case(6)은 두 뱅크 모두에서 캐쉬 미스가 발생하지 않고 분기예측도 정확하므로 다음 확률을 사용하여 interleaved 캐쉬의 개념을 그대로 적용할 수 있다.

표 2 Interleaved 캐쉬의 순차적인 캐쉬 액세스에서 명령어 인출의 여러 경우

Bank 0, 1 CH		Bank 1 CM	Bank 0 CM	
Bank 1 CBP	Bank 1 WBP		Bank 1 CBP	Bank 1 WBP
-	-	-	Consider P_C	Consider P_C
case(1)	case(2)	case(3)	case(4)	case(5)

표 3 Interleaved 캐쉬의 비순차적인 캐쉬 액세스에서 명령어 인출의 여러 경우

Bank 0, 1 CH			Bank 1 CM		Bank 0 CM		
Bank 0, 1 CBP	Bank 1 WBP	Bank 0 WBP	Bank 0 CBP	Bank 0 WBP	Bank 0,1 CBP	Bank 1 WBP	Bank 0 WBP
-	-	Consider P_B	-	Consider P_B	Consider P_C	Consider P_C	Consider P_C, P_B
case(6)	case(7)	case(8)	case(9)	case(10)	case(11)	case(12)	case(13)

$$W_{III} = W_{1,6} = (1-B)(1-R_C)^2(1-R_B) + B(1-R_C)^2(1-R_B)^2 \quad (23)$$

네 번째 그룹(W_{IV})은 interleaved 캐쉬의 페치율을 사용하나 추가로 소비되는 사이클을 고려해 주어야 하는 경우이며, case(4), (8), (10), (11), (13)을 포함한다. case(11)의 경우는 비순차적으로 캐쉬를 액세스하며 뱅크 0에서 미스가 발생한다. 캐쉬 미스 페널티동안 명령어 인출 서비스는 이루어지지 못하여 추가로 소비되는 사이클을 고려해주어야 한다. 메모리로부터 새로운 라인이 전달된 후에는 두 뱅크 모두에서 분기예측이 성공하여 interleaved 캐쉬의 페치율을 적용할 수 있다. Case (8), (10), (13)은 비순차적으로 캐쉬를 액세스하고 뱅크 0의 분기예측이 실패하는 경우로 더욱 세분하여 나눌 수 있지만, 더 이상의 구분은 의미가 없다고 가정한다.

$$\begin{aligned} W_{IV} &= W_{4,8,10,11,13} = 1-(W_I + W_{II} + W_{III}) \\ C_4 &= (1-B)\cdot R_C \cdot (1-R_B)\cdot P_C \\ C_8 &= B\cdot (1-R_C)^2 \cdot R_B \cdot P_B \\ C_{10} &= B\cdot (1-R_C)\cdot R_C \cdot R_B \cdot P_B \\ C_{11} &= B\cdot R_C \cdot (1-R_B)^2 \cdot P_C \\ C_{13} &= B\cdot R_C \cdot R_B \cdot (P_C + P_B) \end{aligned} \quad (24)$$

위의 모든 경우를 고려하여 interleaved 캐쉬의 페치율을 수정하면 다음과 같다.

$$\begin{aligned} F^m_{interleaved} &= W_I \cdot F_{basic} \\ &+ W_{II} \cdot \frac{F_{basic}}{1+C_5+C_{12}} + W_{III} \cdot F_{interleaved} \\ &+ W_{IV} \cdot \frac{F_{interleaved}}{1+C_4+C_8+C_{10}+C_{11}+C_{13}} \end{aligned} \quad (25)$$

5. 시뮬레이션과 분석

5.1 시뮬레이터

본 논문에서는 시뮬레이션을 사용하여 제 4장에서 제시한 명령어 페치의 해석적 모델의 타당성을 분석하였다. 본 논문에서 사용하는 시뮬레이터는 그림 1과 같은 페치 과정을 기본 모델로 하였으며 분류된 캐쉬 종류에 따라 기본 모델을 변형시켜 명령어 페치율을 측정하였다. 시뮬레이션은 프로세서의 동작을 정확히 모델링할 수 있어야 하므로 C++언어를 사용해서 작성하였으며 클락 사이클 단위로 동작한다. 시뮬레이션을 수행하기 위해서는 선택한 벤치마크 프로그램을 실제로 수행시켜 얻은 명령어 트레이스가 필요하다. 시뮬레이션에는 SPEC 95의 GCC, Compress, Li, Tomcatv 벤치마크 프로그램을 선 스팍(Sun SPARC) 워크스테이션에서 GNU C/C++ 컴파일러를 이용하여 컴파일 하였다. 각 벤치마크 프로그램에 대한 명령어 트레이스를 얻기 위하여 Spa 패키지 [20]에 포함되어 있는 명령어 트레이스 생성기인 Spy 프로그램을 사용하였다.

시뮬레이터는 다음과 같이 동작한다. 명령어 인출 과정에서 캐쉬 라인에 분기명령어가 포함되어 있는 경우에는 분기명령어 이후의 명령어는 무효화시키고 분기명령어까지 만을 명령어 디코드 단으로 전달한다. 명령어의 흐름에서 캐쉬 미스가 발생하면 새로운 라인이 서브 메모리로부터 전달될 때까지 캐쉬는 더 이상 인출서비스를 하지 않으며, 분기예측이 실패하면 정확한 목적지 주소가 이용 가능할 때까지 캐쉬를 액세스할 수 없다. 해석적 모델과 시뮬레이션을 동일한 조건으로 비교하기 위하여 해석적 모델에서 사용되는 파라미터의 하나인 분기명령어의 비율 B는 시뮬레이션을 통해서 얻었다. 이 값은 벤치마크 프로그램에 따라 다름으로 각 프로그램의 B 값을 시뮬레이션을 사용하여 구한 후 해석적 모델에서 적용하였다. 표 4는 시뮬레이션을 사용하여 얻어진 각 벤치마크 프로그램들의 분기명령어의 비율을 나타낸다. 분기명령어의 비율 B는 분기명령어의 수를 전체 수행된 명령어의 수로 나눈 값이다.

표 4 명령어 트레이스에 사용된 벤치마크 프로그램과 분기 명령어 비율

Program	GCC	Compress	Li	Tomcatv
Frequency of Branch Instruction (%)	13.333	10.070	14.493	4.55

5.2 해석적 모델과 시뮬레이션의 비교

시뮬레이션에서 사용된 캐쉬는 256개의 엔트리(entry)를 가지며, 캐쉬 라인의 크기(n)는 4와 8인 모델을 각각 사용한다. 또한, 캐쉬 미스율 $R_C = 10\%$, 분기예측 실패율 $R_B = 5\%$, 캐쉬 미스 페널티 $P_C = 10$ 사이클, 그리고 분기예측 실패 페널티 $P_B = 3$ 사이클로 각각 가정한다. 캐쉬의 집합연관도(set associativity)는 미스율에 직접적으로 영향을 주며, 멀티 레벨 캐쉬는 미스 페널티를 줄이기 위해 사용된다. 다시 말하면 멀티 웨이(multi-way), 멀티 레벨(multi-level) 캐쉬는 미스율 R_C와 페널티 P_C 를 조절하여 그 영향을 시뮬레이션에 반영할 수 있다. 해석적 모델과 시뮬레이션으로부터 캐쉬 구조와 액세스 패턴은 명령어 페치율에 가장 큰 영향을 미치는 것을 알 수 있다.

그림 12-15는 각 벤치마크 프로그램에 대하여 캐쉬 타입별 명령어 페치율을 그래프로 나타낸 것이다. 각 그

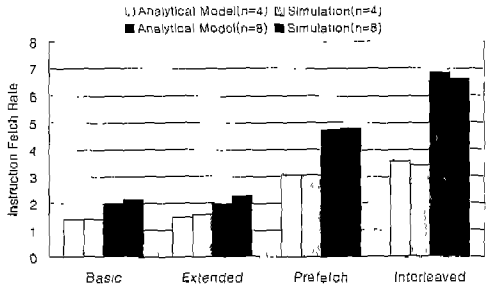

그림 12 GCC 벤치마크 프로그램에서 명령어 페치율의 비교

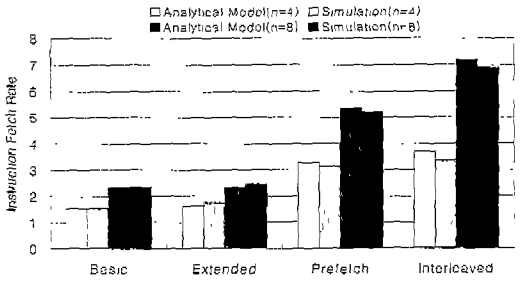

그림 13 Compress 벤치마크 프로그램에서 명령어 페치율의 비교

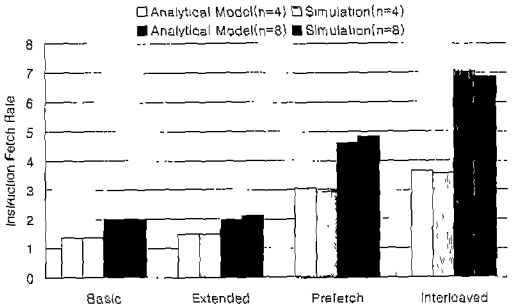

그림 14 Li 벤치마크 프로그램에서 명령어 페치율의 비교

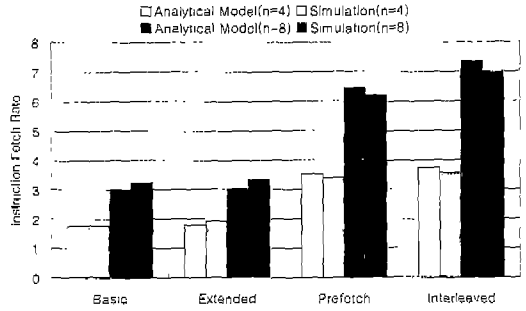

그림 15 Tomcatv 벤치마크 프로그램에서 명령어 페치율의 비교

래프는 $n=4$와 $n=8$일 때의 페치율을 함께 보여준다. 그림 12에서 14는 정수 연산 벤치마크 프로그램인 GCC, Compress, Li에 대한 명령어 페치율의 그래프이다. 분기명령어의 비율이 높은 GCC나 Li 벤치마크 프로그램에서는 명령어 페치율이 낮게 나타난다. 그림 15는 분기명령어의 비율이 낮은 실수 연산 프로그램인 Tomcatv의 경우로 명령어 페치율이 상당히 높게 나타난다. 또한 basic 캐쉬나 extended 캐쉬보다 prefetch 캐쉬나 interleaved 캐쉬를 사용하면 명령어 페치율이 급격히 증가한다. 제안된 해석적 모델은 프로세서의 평균 명령어 페치율을 매우 근사하게 예측함을 알 수 있다. 그러나 interleaved 캐쉬의 경우 그 동작의 복잡성 때문에 해석적 모델과 시뮬레이션 결과의 차이가 다른 캐쉬 타입에 비해서 약간 크게 나타난다.

5.3 캐쉬 미스와 분기예측 실패의 영향

그림 16은 Li 벤치마크 프로그램에서 분기명령어의 빈도를 증가시켜 얻은 명령어 페치율의 변화를 그래프로 나타낸 것이다. Interleaved 캐쉬의 경우 다중 분기예측을 사용하므로 다른 캐쉬 구조에 비하여 분기명령어 빈도의 변화에 가장 적은 영향을 받는다. 그림 17과 18은 $n=8$인 interleaved 캐쉬 구조에서 캐쉬 미스와 분기예측 실패가 GCC 벤치마크 프로그램의 명령어 페치율에 얼마나 영향을 미치는지 보여준다. 그림 17은 캐쉬 미스 페널티(10 사이클)와 분기예측 실패 페널티(3 사이클)를 고정시키고 캐쉬 미스율과 분기예측 실패율은 변화시켜 얻은 명령어 페치율을 나타낸 것이다. 그림 18에서는 캐쉬 미스율은 10%, 분기예측 실패율은 5%로

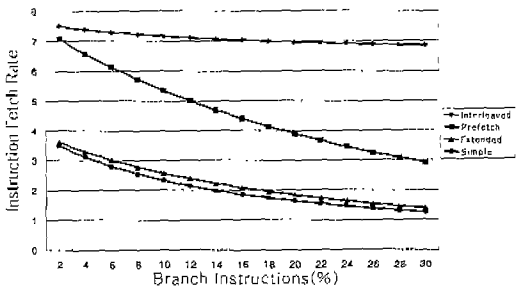

그림 16 분기명령어의 영향

고정시키고, 캐쉬 미스 페널티와 분기예측 실패 페널티를 증가시키는 경우의 명령어 페치율의 감소를 보여준다. 위 그림으로부터 분기예측 실패에 의한 영향보다는 캐쉬 미스율이나 캐쉬 미스 페널티의 증가에 따른 명령어 페치율의 감소가 더 큰 것을 알 수 있다. 즉, 캐시 미스는 분기예측 실패보다 명령어 페치에 보다 민감하게 작용한다.

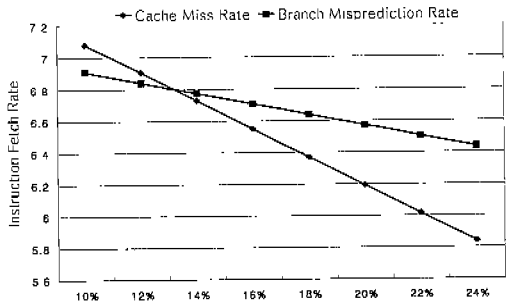

그림 17 캐쉬 미스율과 분기예측 실패율의 영향

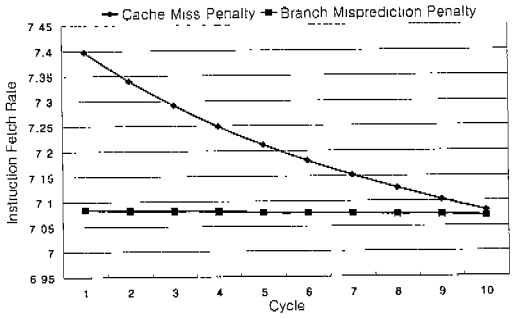

그림 18 캐쉬 미스 페널티와 분기예측 실패 페널티의 영향

6. 결 론

본 논문에서는 슈퍼스칼라 프로세서에 적용될 수 있는 네 가지 명령어 캐쉬 구조에 대하여 분기명령어의 비율, 캐쉬 미스율, 캐쉬 미스 페널티, 분기예측 실패율, 그리고 분기예측 실패 페널티를 파라미터로 사용하여 캐쉬로부터의 명령어 페치율을 해석적으로 계산하였으며, 각 파라미터가 명령어 페치율에 미치는 영향을 분석하였다. 제안한 해석적인 모델은 분기명령어의 비율을 포함하여 캐쉬의 명령어 인출 과정에 큰 영향을 미치는 캐쉬 미스와 분기예측 실패를 고려함으로써 이러한 원인들이 각 캐쉬 타입의 명령어 페치율에 미치는 변화를 정확히 분석할 수 있다.

본 논문에서 제안한 해석적 모델의 타당성을 검증하기 위하여 많은 시뮬레이션을 수행하였다. 해석적 모델과 시뮬레이션의 결과를 비교하면 대부분 경우 10% 이내의 오차범위 안에서 일치하는 것을 알 수 있다. 이렇게 약간의 오차가 발생하는 이유는 명령어 인출 동작의 동적인 특성 때문이며, 대부분의 해석적 모델에서는 피할 수 없다고 여겨진다. 해석적 모델과 시뮬레이션 결과로부터 basic 캐쉬나 extended 캐쉬에 비해 prefetch 캐쉬와 interleaved 캐쉬가 매우 높은 성능을 보여줌을 알 수 있다. 또한 분기예측 실패로 인한 페치율의 감소보다는 캐쉬 미스나 캐쉬 미스 페널티의 증가로 인한 페치율의 감소가 훨씬 큰 것을 알 수 있다. 이러한 해석적인 모델을 사용해서 얻을 수 있는 장점은 프로세서 설계시 시뮬레이션에서는 드러나지 않는 성능제약의 원인을 정확히 분석할 수 있으며, 슈퍼스칼라 프로세서의 성능을 향상시킬 수 있는 캐쉬 구조에 관한 중요한 설계 지식을 얻을 수 있다.

참 고 문 헌

[1] J.L. Hennessy and D.A. Patterson, "Computer architecture: A quantitative approach," *Morgan Kaufmann Publishers*, 2nd Ed. 1996.

[2] M. Johnson, *Superscalar microprocessor design*, Englewood Cliffs, N. J.: Prentice Hall, 1991.

[3] F. Bodin and A. Seznec, "Skewed associativity improves program performance and enhances predictability," *IEEE Trans. Computers*, vol. 46, no. 5, pp. 530-544, May 1997.

[4] O. Temam, C. Fricker, and W. Jalby, "Cache interference phenomena," *Proc. ACM SIGMETRICS*, pp. 261-271, 1994.

[5] R.A. Uhlig and T.N. Mudge, "Trace-driven memory simulation: A survey," *ACM Computing Surveys*, vol. 29, no. 2, pp. 129-170, June 1997.

[6] H.J. Kim, S.M. Kim, and S.B. Choi, "System performance analyses of out-of-order superscalar processors using analytical method," *IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences*, vol. E82-A, no. 6, pp. 927-938. June 1999.

[7] A. Agarwal, M. Horowitz, and J. Hennessy, "An analytical cache model," *ACM Trans. Computer Systems*, vol. 7, no. 2, pp. 184-215, May 1989.

[8] S. Coleman and K.S. McKinley, "Tile size selection using cache organization and data

layout," *Proc. SIGPLAN '95 Conf. Programming Language Design and Implementation*, vol. 30, pp. 279-289, June 1995.
[9] T. Fahringer, "Automatic cache performance prediction in a parallelizing computer," *Proc. AICA '93-International Section*, Sept. 1993.
[10] C. Fricker, O. Temam, and W. Jalby, "Influence of cross interferences on blocked loops: A case study with matrix-vector multiply," *ACM Trans. Programming Languages and Systems*, vol. 17, no. 4, pp. 561-575, July 1995.
[11] S. Ghost, M. Martonosi, and S. Malik, "Cache miss equations: An analytical representation of cache misses," *Proc. 11th ACM Int'l Conf. Supercomputing*, Vienna, Austria, July 1997.
[12] M.S. Lam, E.E. Rothberg, and M.E. Wolf, "The cache performance and optimizations of blocked algorithms," *Proc. Fourth Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, pp. 63-74. Santa Clara, Calif., 1991.
[13] K.S. McKinley and O. Temam, "A quantitative analysis of loop nest locality," *Proc. Seventh Conf. Architectural Support for Programming Languages and Operating Systems*, vol. 7, Oct. 1996.
[14] M.E. Wolf and M.S. Lam, "A data locality optimizing algorithm," *Proc. SIGPLAN '91 Conf. Programming Language Design and Implementation*, vol. 26, pp. 30-44, June 1991.
[15] J.S. Harper, D.J. Kerbyson, and G.R. Nudd, "Analytical modeling of set-associative cache behavior," *IEEE Trans. on Computers*, vol. 48, no. 10, pp. 1009-1023, Oct. 1999.
[16] T.Y. Yeh, D.T. Marr, and Y.N. Patt, "Increasing the instruction fetch rate via multiple branch prediction and a branch address cache," *Proc. Seventh ACM Int'l Conf. Supercomputing*, pp. 67-76, Tokyo, July 1993.
[17] S. Wallace and N. Bagherzadeh, "Modeled and measured instruction fetching performance for superscalar microprocessors," *IEEE Trans. Parallel and Distributed Systems*, vol. 9, no. 6, pp. 570-578, June 1998.
[18] M.D. Smith, M. Johnson, and M.A. Horowitz, "Limits on multiple instruction issue," *Proc. Third Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, pp. 290-302, Apr. 1989.
[19] T.M. Conte, K.N. Menezes, P.M. Mills, and B.A. Patel, "Optimization of instruction fetch mechanisms for high issue rates," *Proc. 22nd Ann. Int'l Symp. Computer Architecture*, pp. 333-344, June 1995.
[20] G. Irlam, "Spa" Personal Communication http://www.base.com/gordoni/spa/cat1/spy.1, 1995.
[21] Standard Performance Evaluation Corporation, "SPEC CPU95 benchmark," http://www.specbench.org/osg/cpu95, Mar. 1998.

김 선 모
1998년 인하대학교 전자공학과(학사). 2000년 인하대학교 전자공학과(석사). 2000년 ~ (주) LG 전자. 관심분야는 Computer Architecture, Parallel and Distributed system.

정 진 하
1992년 인하대학교 전자공학과(학사). 1994년 인하대학교 전자공학과(석사). 1994년 ~ 1999년 (주)한미 기술연구소. 2000년 인하대학교 전자공학과(박사과정). 관심분야는 컴퓨터구조, Fault-tolerant computing, Parallel processing.

최 상 방
정보과학회논문지 : 시스템 및이론
제 28 권 제 1 호 참조