와이드 이슈 프로세서를 위한 스트라이드 값 예측기의 모험적 갱신

(Speculative Updates of a Stride Value Predictor in Wide-Issue Processors)

전 병 찬 † 이 상 정 ††

(Byung-Chan Jeon) (Sang-Jeong Lee)

요 약 슈퍼스칼라 프로세서에서 값 예측(value prediction)은 한 명령의 결과를 미리 예측하여 명령들 간의 데이타 종속관계를 극복하고 실행함으로써 명령어 수준 병렬성(Instruction Level Parallesim, ILP)을 이용하는 기법이다. 값 예측기(value predictor)는 명령어 페치 시에 예측 테이블을 참조(lookup)하여 값을 예측하고, 명령의 실행 후 판명된 예측 결과에 따라 테이블을 갱신(update)하여 이 후의 참조를 대비한다. 그러나, 최근의 값 예측기는 프로세서의 명령 페치 및 이슈율이 커짐에 따라 예측 테이블이 갱신되기 전에 다시 같은 명령이 페치되어 갱신되지 못한 낡은 값(stale value)으로 예측되는 경우가 빈번히 발생하여 예측기의 성능이 저하되는 경향이 있다.

본 논문에서는 이러한 성능저하를 줄이기 위해 명령의 결과가 나올 때까지 기다리지 않고 테이블 값을 모험적으로 갱신(speculative update)하는 스트라이드 값 예측기(stride value predictor)를 제안한다. 제안된 방식의 타당성을 검증하기 위해 SimpleScalar 시뮬레이터 상에 제안된 예측기를 구현하여 SPECint95 벤치마크를 시뮬레이션하고 제안된 모험적 갱신의 스트라이드 예측기가 기존의 스트라이드 예측기 보다 성능이 향상됨을 보인다.

Abstract In superscalar processors, value prediction is a technique that breaks true data dependences by predicting the outcome of an instruction in order to exploit instruction level parallelism(ILP). A value predictor looks up the prediction table for the prediction value of an instruction in the instruction fetch stage, and updates with the prediction result and the resolved value after the execution of the instruction for the next prediction. However, as the instruction fetch and issue rates are increased, the same instruction is likely to fetch again before it has been updated in the predictor. Hence, the predictor looks up the stale value in the table and this mostly will cause incorrect value predictions.

In this paper, a stride value predictor with the capability of speculative updates, which can update the prediction table speculatively without waiting until the instruction has been completed, is proposed. Also, the performance of the scheme is examined using Simplescalar simulator for SPECint95 benchmarks in which our value predictor is added.

1. 서 론

· 본 연구는 한국과학재단 목적기초연구(2000-1-30300-008-3) 지원으로 수행되었음.

† 학생회원 : 순천향대학교 컴퓨터학부
leebc 66@hanmail.net
†† 정 회 원 : 순천향대학교 컴퓨터학부 교수
sjlee@asan.sch.ac.kr

논문접수 2001년 4월 4일
심사완료 2001년 8월 27일

와이드 이슈 슈퍼스칼라 프로세서(wide-issue superscalar processors)에서 높은 성능을 도달하기 위해서는 명령어 수준 병렬성(Instruction Level Parallelism, ILP)을 이용하여 다수의 명령을 동시에 이슈하고 처리해야 한다. ILP를 이용하는 주요장애는 명령어간의 종속 관계인 제어종속(control dependences)과 데이타종속(data dependences) 관계이다. 제어종속 관계는 프로그램의 제어흐름이 변경되는 분기 명령에 의해 발생한

다. 즉, 분기될 타겟주소와 분기조건 결과 생성의 지연으로 발생하는 문제이다. 제어종속 관계를 극복하는 방식으로는 조건 분기명령을 제거하고 제어종속 관계가 있는 명령을 조건실행(predicated execution)하여 제거하는 방식[1]과 조건분기의 결과를 미리 예측하여 제어종속관계가 있는 명령들을 미리 페치하여 모험적으로 실행(speculative execution)하는 방식[2,3,4,5]이 있다. 데이타 종속관계는 현재 명령이 이전명령의 수행결과를 참조할 때 발생하고, 이전 명령의 결과가 생성될 때까지 현재의 명령은 실행할 수가 없다. 데이타종속 관계는 와이드 이슈 프로세서에서 명령 간에 빈번히 발생하기 때문에 명령어 수준 병렬처리의 주요 장애가 되어 고성능 슈퍼스칼라 프로세서의 성능 저하의 주된 요인이 되고 있다. 따라서, 최근에는 실행되는 명령의 결과 값을 미리 예측하고, 이 후 데이타종속 관계가 있는 명령들에게 값을 조기에 공급하고 이들 명령들을 모험적으로 실행하여 성능향상을 꾀하는 값 예측(value prediction)방식에 관하여 활발히 연구가 진행되고 있다.[6-9,10-14,15-18,19]

값 예측기(value predictor)는 명령어 페치 시에 예측 테이블을 참조하여 값을 예측하고, 명령의 실행 후 수행결과와 판명된 예측결과에 따라 테이블을 갱신하여 이 후의 참조를 대비한다. 그러나, 최근의 프로세서들이 보다 높은 성능을 위하여 명령어의 페치 및 이슈율을 증대시킴에 따라 기존의 값 예측기를 이용하여 효율적으로 예측하는데 문제가 대두되고 있다. 즉, 페치와 이슈율이 커짐에 따라 명령의 수행 후에 예측 테이블이 갱신되기 전에 다시 같은 명령이 페치되는 비율이 높아지고, 갱신되지 못한 낡은 값(stale value)으로 예측하는 경우가 빈번히 발생한다. 예를 들어 명령 페치에서 결과 생성까지 4 클럭 소요되는 경우 4-이슈 프로세서에서는 이 기간 동안 최대 16개의 명령이 페치와 이슈되고 반면 8-이슈 프로세서에서는 최대 32개의 명령이 페치되고 이슈되어 짧은 루프를 실행하는 경우에 한 명령의 결과 생성 전에 같은 명령이 다시 페치될 확률이 더욱 높아지게 된다. 이러한 테이블 갱신의 지연은 명령의 결과가 생성될 때마다 매번 새로운 값으로 변경되어 예측 테이블에 저장하는 스트라이드 예측기의 경우에 더 큰 영향을 미친다[11].

본 논문에서는 위와 같은 테이블 갱신의 지연으로 인한 성능저하를 줄이기 위해 명령의 결과가 나올 때까지 기다리지 않고 예측 테이블 값을 모험적으로 갱신(speculative update)하는 스트라이드 값 예측기를 제안한다. 제안된 스트라이드 예측기는 지연 갱신되는 예측 테이블 엔트리를 추적하고 테이블의 값을 모험적으로 갱신하기 위해 예측 테이블에 에이지 카운터(age counter)를 도입한다. 에이지 카운터는 테이블 참조 시에는 증가하고 테이블 갱신 시에는 감소하여 갱신의 지연을 감지한다. 제안된 방식의 타당성을 검증하기 위해 사이클 수준에 시뮬레이션이 가능한 SimpleScalar 시뮬레이터[20] 상에 제안된 예측기를 구현하여 SPECint95 벤치마크를 시뮬레이션하고 제안된 스트라이드 모험적 갱신(stride speculative update)이 기존의 스트라이드 예측기 보다 성능이 향상됨을 보인다.

본 논문의 구성은 다음과 같다. 제2장에서는 기존의 값 예측기의 이론적 배경을 소개하고, 3장에서는 테이블 갱신 지연의 문제점을 고찰하며, 4장에서는 본 논문에서 제안한 스트라이드 값 예측기의 모험적 갱신을 기술한다. 그리고 5장에서는 실험을 통해 측정된 성능을 분석하고, 마지막으로 6장에서 결론을 기술한다.

2. 이론적 배경

그림 1은 임의 코드에 대한 값 예측의 예이다. 그림 1에서 j의 소스 R3는 i의 R3 결과가 생성되어야만 사용이 가능하며, j가 실행된 후 R5의 결과가 생성되면 명령 k, l이 실행될 수가 있다. 이러한 데이타 종속 관계가 있는 명령에 값 예측기를 사용하여 i, j의 결과 값 R3, R5를 미리 예측하면 i, j와 데이타 종속 관계를 갖는 명령 k, l들에 대한 종속관계가 제거되어 이들 명령들을 모험적으로 실행하여 예측이 올바른 경우 성능이 향상될 수 있다.

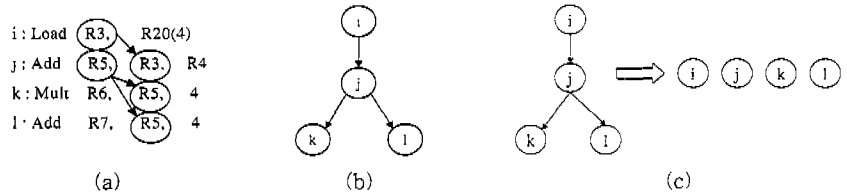


그림 1 값 예측 예 (a) 코드 예 (b) 데이타 종속 그래프 (c) 값 예측 후 종속관계 제거

값 예측기는 프로세서에 의해 반복 수행되어지는 명령들의 반복되는 과거의 결과 값의 유형에 근거하여 값을 예측한다. 값 예측기의 동작은 대개 비슷한데 명령의 값 예측은 다음에 수행될 명령의 PC를 사용하여 명령을 페치할 때 동시에 예측기의 테이블을 참조하여 값을 예측한다. 그리고 이 명령의 수행 후에 결과 값이 생성되면 이 값과 예측의 성공여부에 근거하여 예측 테이블을 갱신하고 이 후 예측을 준비한다. 일반적으로 반복 수행되는 명령은 표 1과 같은 상수(constant), 스트라이드, 그리고 비-스트라이드의 값 시퀀스(value sequence) 패턴을 갖는다[17].

상수 시퀀스는 명령의 수행 결과가 변하지 않고 계속 같은 값을 생성하는 유형이다. 이 유형의 값 예측은 최근의 수행된 결과 값을 예측 테이블에 저장하고 다음에 해당 명령의 페치 시에 이 결과 값을 참조하여 예측하면 된다. Lipasti 등은 이러한 패턴이 자주 발생함을 주목하여 최근 값 예측기(last value predictor)를 개발하였다[13].

스트라이드 시퀀스는 명령이 수행될 때마다 일정한 값만큼의 간격 차이(이를 *스트라이드*라 한다.)로 증가 감소되는 패턴으로 프로그램에서 배열이나 루프 인덱스들을 참조할 때 빈번히 발생하는 패턴들이다. 이러한 유형의 명령들의 예측을 위해서 스트라이드 값 예측기가 개발되었다[7,15,19]. 즉, 이 예측기는 3개의 명령을 수행결과를 추적하여 각 명령들 간의 결과 값의 차이인 두 개의 스트라이드를 구한다. 이 두 개의 스트라이드 값이 같으면 스트라이드 시퀀스로 간주하여 최근에 수행된 결과 값과 스트라이드를 예측 테이블에 저장한다. 다음에 해당 명령을 다시 페치하여 수행하는 경우 예측 테이블에 있는 최근의 결과 값과 스트라이드를 더하여 구해진 값을 예측 값으로 한다. 상수 시퀀스의 경우는 스트라이드가 0인 경우로 해석할 수 있기 때문에 스트라이드 값 예측기는 최근 값 예측기를 포함한다.

비-스트라이드 시퀀스는 위의 상수 시퀀스 또는 스트라이드 시퀀스에 속하지 않는 유형으로 표 1에서와 같이 두 가지로 분류될 수 있다. 반복패턴은 여러 값들의 시퀀스가 반복되는 경우이다. 구문형 값 예측기(context-based value predictor)는 이러한 반복되는 값들을 모두 테이블에 저장하고 최근의 값들의 패턴으로 근거하여 다음 값을 예측한다[17,21]. 예를 들어 표 1의 예에서 한 명령의 최근 값이 11이라면 이를 근거로 하여 다음 반복 패턴 22를 예측한다. 이 예측기는 비-스트라이드 시퀀스의 반복되는 패턴도 예측할 수 있어서 스트라이드 보다는 성능이 우수하지만 예측 테이블에 반복되는 값들을 모두 저장해야 하기 때문에 부가되는 하드웨어의 경비가 크다는 단점이 있다. 비 반복 패턴은 값 예측을 위해 어떤 패턴도 추출할 수 없는 경우로 예측이 불가능하다.

최근에는 최근 값 예측기, 스트라이드 값 예측기와 구문형 예측기를 혼합한 혼합형 예측기(hybrid value predictor)도 제안되었다[15,16,18,19]. 혼합형 예측기는 여러 유형의 반복되는 명령에 대해서 예측이 가능하여 성능이 우수하지만 예측 테이블의 크기 증가로 높은 하드웨어 경비가 요구된다.

표 1 값 시퀀스 유형

값의 유형	데이타 시퀀스 예
상수	3 3 3 3 . . .
	같은 값 반복
스트라이드	1 3 5 7 . . .
	스트라이드가 2
비-스트라이드	6 11 22 -5 6 11 22 -5 6 . . .
	반복 패턴
	17 9 -13 5 15 . . .
	비 반복 패턴

3. 예측 테이블 갱신 지연

그림 2는 기존의 스트라이드 값 예측기에서 예측 테이블의 갱신 지연으로 인하여 발생되는 문제점을 보여주는 그림이다. 그림 2의 (a)와 같이 명령 I를 포함하여 8개의 명령으로 구성된 루프를 가정한다. 또 8 이슈 프로세서가 8개의 루프 내의 명령을 한 클럭에 전부 페치하는 것으로 가정한다. 그림 2의 (b)는 임의의 사이클 i에서 부터 명령 I가 페치되어 파이프라인을 통해 수행되는 과정을 보여주는 그림이다. 사이클 i에서 명령 I가 예측 테이블을 참조하여 값을 예측하는 것으로 가정하면 프로세서는 사이클 i to i+3 동안에 명령 I의 4개의 인스턴스(I_i, I_{i-1}, I_{i-2}, I_{i-3})를 페치하고 예측 테이블을 참조하여 값을 예측한다. 그러나 명령 I_i의 예측 결과는 실행 후 writeback 스테이지에서 검증되기 때문에 i-3 사이클에서 예측 테이블이 갱신된다. 따라서 명령 I의 인스턴스 I_{i-1}, I_{i-2}, I_{i-3}을 사이클 i+1, i+2, i+3에서 각각 페치하고 값을 예측할 때 예측 테이블은 갱신되지 않기 때문에 I_i와 같은 값으로 예측되어 모두 틀린 값으로 예측하게 된다. 즉 예측 테이블이 갱신되기 전에 다시 동일 테이블의 엔트리가 페치되고 예측되기 때문에 모두 틀린 값으로 예측된다.

```
loop:
    addiu   St8, St8, 1; // 명령 I
    bne loop;          // 루프 종료 분기 Br

                (a) 루프 코드
사이클 :          ..... I      i+1    i+2    i+3 ... .....

fetch                    Ii     Ii+1   Ii+2   Ii+3   ---예측 테이블 참조
dispatch                        Ii     Ii+1   Ii+2
issue and execute                      Ii     Ii+1
writeback                                     Ii     ---예측 테이블 갱신

                (b) 파이프라인 수행
```

그림 2 스트라이드 값 예측기에서 예측 테이블의 갱신지연 문제점 예

그림 3은 모든 SPECint95 벤치마크 프로그램에 대해 한 명령이 페치되고 예측 테이블을 참조한 후 다시 이 명령이 페치되어 동일 예측 테이블 엔트리가 다시 참조되는 클럭의 간격을 보여주는 그림이다

그림 3의 결과는 8-이슈를 갖는 프로세서 상에서 측정된 결과이고(머신 구성은 5.1절 표 2 참조), 명령의 비율은 값 예측이 가능한 명령, 즉 레지스터에 결과를 저장하는 명령 중에서 측정된 비율이다(SPECint95 벤치마크는 전체 수행된 명령 중 평균 69.4%가 레지스터에 결과를 저장하는 명령으로 측정되었다). x 축은 다시 참조되는 간격의 클럭 사이클을 보여주며, y 축은 전체 예측되는 명령 중에서 차지하는 비율을 나타낸다. 그림 3에서 평균적으로 1 클럭 후에 명령이 다시 참조되는 비율이 1.8%, 2 클럭 후에는 1.9%, 3 클럭 후에는 3.9%, 4 클럭 후에는 4.4%, 그리고 5 클럭 후에는 5.3% 임을 알 수 있다. 그림 4는 그림 3의 결과에 보여준 각 벤치마크 프로그램의 명령비율의 평균값을 재참조 클럭 간격이 증가됨에 따라 누적하여 도시한 그림이다(그림 3의 8-이슈 프로세서(Stride8f) 뿐만 아니라 4-이슈 프로세서에 대한 측정 값(Stride4f)도 삽입하였다). 예를들어 8-이슈의 재참조 클럭 간격 3의 명령 비율이 6.07% 인데 이는 재참조 클럭 간격 1,2,3에서 참조되는 명령의 평균값을 누적해서 더한 값으로 6.07%의 명령이 3클럭 이내에서 다시 참조됨을 의미한다. 그림 4에 나타난 바와 같이 8-이슈인 경우에 전체 예측 명령 중에 13.8%의 명령들이 5 클럭 사이클 이내에 다시 페치되어 예측 테이블을 참조하고, 4 이슈인 경우에는 5.9%의 명령들이 5 클럭 이내에 다시 참조된다. 명령의 페치 및 이슈율이 4-이슈에서 8-이슈로 증가함에 따라 다시 참조되는 비율이 크게 증가하는 것을 알 수 있다

최근의 고성능 프로세서는 높은 ILP를 위해 파이프라인 길이를 늘리는 추세에 있고 대개 5개 이상의 파이프라인 스테이지를 갖는다. 이는 명령의 결과 값이 생성되고 생성된 데이터 값을 이용하여 예측 테이블을 갱신하는데 최소한 3에서 5 클럭 이상이 필요함을 의미한다. (실제 코드에서는 명령 수행 시 하드웨어 자원 부족으로 인한 기능장치 부족, 캐시 미스 등으로 더 많은 클럭이 요구된다). 따라서 이들 클럭 이내에서 다시 페치되고 예측되는 명령들은 새로운 결과 값으로 예측 테이블을 갱신하기 전에 다시 테이블을 참조한다. 8 이슈인

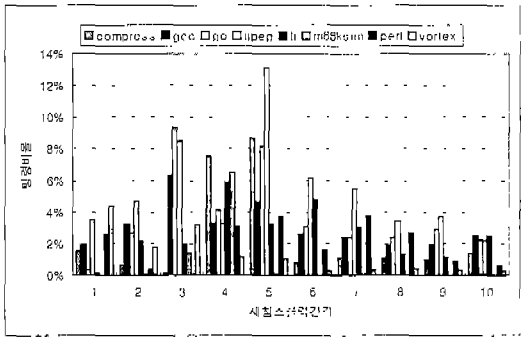


그림 3 예측테이블을 다시 참조하는 명령들의 클럭 간격별 분포

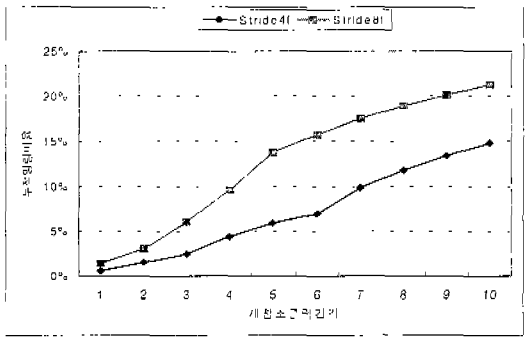


그림 4 예측테이블을 다시 참조하는 명령들의 클럭 간격별 누적 분포

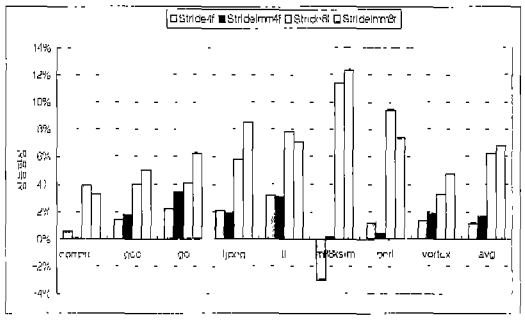


그림 5 예측 테이블 갱신 지연의 성능영향 효과

경우에는 전체 예측 명령 중 13.8%, 4-이슈인 경우는 5.9%의 명령들의 값 예측이 잘못된 값으로 예측될 확률이 높음을 의미한다.

그림 5는 4-이슈, 8-이슈 프로세서에서 스트라이드 예측기에 대해 성능을 측정 비교한 그림으로, 값 예측을 시도하지 않은 머신을 기준으로 하여 각 예측기를 사용했을 경우에 성능 향상(speedup)을 측정하였다. 그림5에서 Stride는 기존의 정상적인 스트라이드 예측기를 표시하고, StrideImm는 예측 테이블의 참조와 동시에 결과값으로 테이블을 갱신하는 것을 가정한 즉각적인 갱신(Immediate Update)을 하는 예측기를 나타낸다. 그림 5에 나타난 바와 같이 4-이슈인 경우에 최대 1.2 %(평균 0.5 %), 8-이슈인 경우에 최대 2.7 %(평균 0.6 %)로 Stride와 StrideImm 사이에 성능에서 차이가 있음을 알 수 있다. 4-이슈의 m88ksim의 경우 값 예측으로 오히려 성능이 더 저하되는데 이는 잘못된 값 예측으로 부가된 클럭 손실로 인한 것이다. 또한 compress, perl 등은 StrideImm가 Stride 보다도 성능이 저하되는데, 이는 예측된 모든 값이 이 후의 명령과 데이타 종속관계가 있는 것이 아니고, 잘못된 분기예측으로 잘못 페치된 명령들의 더 적극적인 값 예측으로 인한 부가적인 손실에 기인한다.

그림 6은 SimpleScalar 시뮬레이터를 사용하여 SPECint95 벤치마크 중 하나인 Compress 벤치마크의 40D68번지의 한 명령에 명령 페치 및 예측과 테이블의 갱신을 추적하여 실행한 예이다. 그림 6의 결과는 8개 명령을 페치하고 이슈하는 프로세서에서 수행된 결과이다(머신 구성은 5.1절 표2 참조).

첫 번째 열의 값은 수행된 클럭 사이클을 표시하고, 명령 "addiu r2,r2,3"의 행은 명령의 페치를 나타내고, "UPDATE" 행은 결과 값 생성 후 예측 테이블의 갱신을 의미한다. 또한 "갱신지연" 열의 X는 이전에 페치된

```
클럭                              갱신지연  실제값 예측값
        ..........
31243: 406D68 addiu   r2,r2,3        5     5
31250: 406D68 UPDATE
31421: 406D68 addiu   r2,r2,3        8     8
31424: 406D68 addiu   r2,r2,3    X   11     8
31426: 406D68 addiu   r2,r2,3    X   14     8
31428: 406D68 UPDATE
31429: 406D68 addiu   r2,r2,3    X   17    11
31430: 406D68 UPDATE
31431: 406D68 addiu   r2,r2,3    X   20    14
31432: 406D68 UPDATE
31433 406D68 UPDATE
31434: 406D68 addiu   r2,r2,3    X   23    20
31436: 406D68 UPDATE
31439: 406D68 UPDATE
31440: 406D68 addiu   r2,r2,3        26    26
31446: 406D68 addiu   r2,r2,3    X   29    26
31447: 406D68 UPDATE
31449  406D68 addiu   r2,r2,3    X   32    29
31451: 406D68 addiu   r2,r2,3    X   35    29
31453: 406D68 UPDATE
31454: 406D68 addiu   r2,r2,3    X   38    32
31455: 406D68 UPDATE
31456: 406D68 addiu   r2,r2,3    X   41    35
31457: 406D68 UPDATE
31459: 406D68 UPDATE
31459: 406D68 addiu   r2,r2,3    X   44    41
31461: 406D68 UPDATE
        ..........
        ..........
```

그림 6 compress 벤치마크 코드 수행 예

명령이 갱신되기 전에 또 다시 예측되는 경우를 표시한다. 또한 "실제값"은 실제 이 명령 수행 후의 결과 값을 나타내며, "예측값"은 스트라이드 예측기를 사용하여 이 명령 페치 시에 예측되는 값이다. 예를들어 클럭 사이클 31421의 명령은 7 클럭 사이클 후의 31428 클럭에서 갱신되는데 이 사이에 31424, 31426 클럭 사이클의 2개 명령이 페치되어 갱신되지 않는 값으로 예측된다. 만약 클럭 31250에서 최종 값 5, 스트라이드 3으로 예측 테이블을 갱신하였다고 가정하자. 클럭 31421에서 r2의 값은 예측 테이블의 최종 값 5와 스트라이드 3을 더하여 8로 예측할 것이다. 그리고 예측 테이블이 갱신되기 전에 명령이 페치되어서 클럭 31424, 31426에서도 또한 8로 예측 할 것이고 이는 실제 값 11, 14 과는 다르게 되어 예측 실패를 가져올 것이다. 위의 예에서 갱신 지연된 모든 명령의 값 예측이 틀리게 된다.

본 논문에서는 이상과 같은 테이블 갱신의 지연으로 인하여 잘못 예측되는 경우를 줄이기 위해 명령의 결과가 나올 때까지 기다리지 않고 예측 테이블 값을 모험적으로 갱신하는 스트라이드 값 예측기를 제안한다. 제

안된 스트라이드 예측기는 지연 갱신되는 예측 테이블 엔트리를 추적하고 테이블의 값을 모험적으로 갱신하기 위해 예측기 테이블에 에이지 카운터(age counter)를 도입한다.

4. 모험적 갱신 스트라이드 예측기

4.1 스트라이드 예측기 구성도

그림 7은 모험적 갱신 기능이 추가된 제안한 스트라이드 예측기의 구성도이다. 그림 7의 (a)는 예측 테이블 엔트리의 필드 구성을 나타낸다. 태그(tag) 필드는 PC로 인덱스된 엔트리의 매치 여부를 검증하기 위한 필드로 테이블의 크기에 따라 가변인데 SimpleScalar 머신 상에서 8K 엔트리를 갖는 테이블의 경우 16 비트가 된다. 상태(state) 필드는 3가지의 예측 상태를 표시하는 필드로 초기상태, 전이상태, 안정 상태의 값을 표시하기 위해 2비트로 표현된다. 값(value) 필드는 가장 최근의 명령 수행 결과 값을 저장하기 위해 32비트가 필요하고, 스트라이드 필드는 최근 두 명령의 결과 값의 차를 저장하기 위해 32비트로 표현된다. 또한 에이지 카운터는 모험적 갱신을 위해 도입된 카운터며, 5비트를 갖는다.

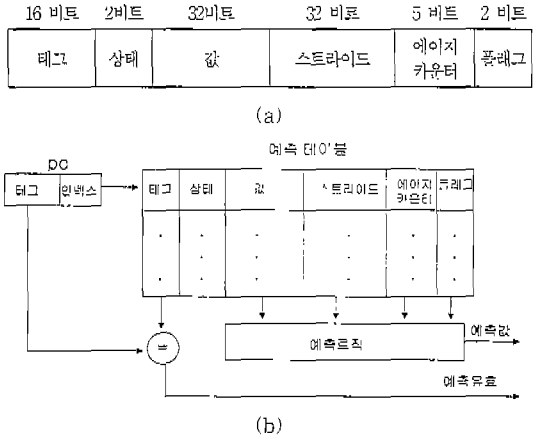


그림 7 스트라이드 예측의 (a) 예측 테이블 엔트리의 필드 구성 (b) 구성도

플래그 필드 모험적 갱신을 위해 테이블의 상태를 표시하는 2비트의 크기를 갖는다. (4.2 절 참조). 그림 7의 (b)는 전체 예측기 구성도를 보여주며, 명령이 페치될 때 PC를 이용하여 테이블을 인덱스한다. 이때 태그를 비교하여 매치되면 예측유효 비트를 1로 세팅하고, 테이블 미스인 경우에는 0으로 세팅하여 예측 값이 유효하지 않음을 표시한다. 모험적 갱신이 아니고 상태의 값이 안정상태인 정상적인 경우에는 테이블의 값 필드와 스트라이드 필드의 값을 더하여 인덱스된 명령의 값을 예측한다.

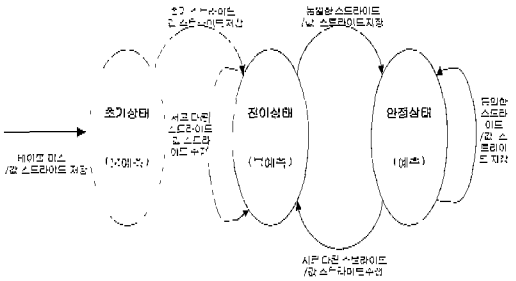


그림 8 스트라이드 값 예측 상태 천이도

그림 8은 스트라이드 예측기의 값 예측을 위한 3 상태 천이도를 보여주는 그림이다. 상태의 값은 예측 테이블의 상태 필드에 저장된다. 먼저 예측될 명령이 처음 테이블에 등록될 때 상태는 초기상태가 되고 명령의 값을 예측하지 않는다. 이 명령의 실행 후 예측 테이블 갱신 시에 실행 결과 값(value1)을 테이블의 값 필드에 저장한다. 이 후 같은 명령이 다시 페치되고 상태가 초기상태이면 상태를 전이상태로 변환하고 값은 예측되지 않는다. 또한 현재 실행된 결과 값(value2)에서 테이블의 값 필드에 저장된 값(value1)의 차를 구하여 스트라이드(stride1)를 스트라이드 필드에 저장하고 최근의 실행 값(value2)도 값 필드에 저장한다. 또 한번 동일한 명령이 페치되어(상태가 전이상태이므로 값은 예측되지는 않는다. 테이블이 갱신될 때 현재 실행 결과 값(value3)과 테이블의 값 필드에 저장된 값(value2)의 차로 새로운 스트라이드(stride2)를 구하고, 새로운 스트라이드(stride2)와 이 전의 스트라이드(stride1)의 값이 같으면 상태는 전이상태에서 안정상태로 변환된다. 만약 같지 않으면 상태는 그대로 전이상태로 남아있게 된다. 그리고 예측 테이블의 값과 스트라이드 필드에 각각 새로운 값으로 값(value2)와 스트라이드(stride2)를 저장한다. 명령의 페치 시 예측 테이블의 상태의 값이 안정상태일 경우에만 값 필드와 스트라이드를 더한 값으로 예측한다.

명령 파이프라인 상에서 값 예측기의 동작은 크게 예측 테이블 참조(lookup) 및 갱신(update), 두 가지 동작으로 구분할 수 있다. 한 명령(이 후 예측명령) 페치 시에 PC 주소로 테이블을 인덱스하고 태그가 매치되면

앞에서 언급한 상태를 참조하여 안정상태이면 예측을 시도한다. 명령 디코드 후에 예측된 값들은 명령 윈도우에서 소스 오퍼랜드의 값이 생성되기를 기다리는 데이타 종속 명령(이 후 종속명령)들에게 보내지게 된다. 명령 윈도우에서 대기하고 있는 종속명령의 모든 소스 오퍼랜도가 가용하고(소스 오퍼랜도의 값이 실제 값이든지 예측 값이든지 관계없이), 기능장치(functional unit)들을 사용할 수 있으면 이 종속명령들은 실행을 위해 이슈된다. 예측명령의 실행 후에 올바르게 예측되어 실제 값이 예측된 값과 같으면 파이프라인은 중지되지 않고 계속 진행된다. 만약 예측이 틀린 경우에는 이 예측 값으로 실행된 종속명령들을 취소하고 올바른 소스 오퍼랜드 값으로 다시 이슈하여 실행한다. 예측된 값을 사용한 종속명령의 실행이 아직 끝나지 않은 경우에는 단순히 명령의 실행 값을 무시하면 된다. 그러나 비순차 실행(out-of-order execution)으로 인하여 종속명령이 이미 실행된 경우에는 이 명령들이 리오더 버퍼(reorder buffer)에 있으므로 리오더 버퍼에서 엔트리를 제거하고 다시 이슈한다. 예측 실패 시의 부가되는 페널티는 머신의 파이프라인 구조에 따라 다른데 대개 예측 값의 비교 및 재 이슈를 위해 부가되는 1 클럭 사이클과 잘못 수행된 명령이 하드웨어 자원(기능장치, 리오더 버퍼 등)을 점유함으로써 다른 명령들의 이슈 및 종료가 지연되는 페널티가 있다[16,17]. 명령 실행 후에는 예측의 성공여부, 예측명령의 실행 결과 값을 가지고 예측 테이블을 갱신한다. 테이블의 갱신은 그림 8과 같은 상태 전이를 갖고 수행된다.

4.2 모험적 갱신 동작

제안된 스트라이드 예측기는 지연 갱신되는 예측 테이블 엔트리를 추적하고 테이블의 값을 모험적으로 갱신하기 위해 예측 테이블에 에이지 카운터를 도입한다. 에이지 카운터는 5비트의 크기를 가지며, 테이블 참조 시에는 1씩 증가하고, 테이블 갱신 시에는 1씩 감소하여 갱신의 지연을 감지한다. 한 명령이 페치 된 후 예측 테이블을 참조하여 카운터가 증가된 후 갱신되지 않은 채 이 명령이 또 다시 페치되어 참조되면 카운터의 값이 0이 아닌 값이 되면 예측 테이블의 값이 아직 갱신되지 않은 값을 가짐을 의미한다. 0이 아닌 에이지 카운터의 값은 지연되어 갱신되지 못한 명령의 수를 반영하므로 이 값과 스트라이드로부터 올바른 예측 값을 테이블 참조 시에 계산할 수 있다. 이때 테이블의 값 필드는 이 새로운 값으로 대체되고 이후의 테이블 참조 시마다 명령의 수행결과가 나올 때까지 기다리지 않고 모험적으로 갱신된다.

그림 9는 제안된 스트라이드 예측기에서 명령을 페치할 때 예측 테이블 참조와 명령 실행 후 예측 테이블 갱신 시에 에이지 카운터 동작을 보여주는 그림으로 (a)는 테이블 참조 시의 동작이고 (b) 테이블 갱신 시의 동작이다. 그림 9의 (a)에서 명령 페치 시 예측 테이블을 참조 할 때 테이블에 엔트리가 없으면 테이블을 할당하고 before_update 플래그를 사용하여 갱신 전에 테이블이 할당되었음을 표시한다. before_update가 참이면 한번도 갱신이 발생하지 않음을 의미하고 에이지 카운터 만을 증가시킨다. 현재 모험적 갱신 모드가 아니지만 안정상태이고, 에이지 카운터가 0이 아니면 모험적 갱신 모드의 시작이므로 에이지 카운터 값을 참조하여 새로운 value의 값을 계산하고, speculative_update를

```
if (prediction table miss) {  /* 예측 테이블 미스이면 */
  table=allocation new entry of the table.
  table.age=0;
  table.before_update=TRUE;
  return.
}
else if (table.before_update) {   /* 초기 테이블 갱신 전 참조이면 */
  ++table.age.
  return;
}
if (!table.speculative_update) { /* 현재 모험적 갱신모드가 아니고 */
  if (table.state==Steady) {      /* 안정상태이면 */
            if (table.age > 0) {       /* 모험적 갱신 모드 시작 */
          table.value = table.value + table.stride *(table.age+1).
          table.speculative_update = TRUE.
                ++table.age;
          return table.value;
      }
      else {      /* 정상적인 모드 예측 */
                ++table.age;
          return( table.value + table.stride).
      }
  }
  else      /* 안정상태가 아니면 */
            ++table.age;
}
else {     /* 계속되는 모험적 갱신 모드이면  */
  table.value=table.value + table.stride.
  ++table.age;
  return table.value.
}
```

(a)

```
if (table.age > 0) {  /* 모험적 갱신 되었으면서 */
   if(prediction is incorrect) { /* 예측이 틀렸으면 */
            table.speculative_update = FALSE;
            table.state = Init;
            table.value = resolved data;
   }
}
--table.age;
if (!table.before_update)
   update prediction table;  /* 그림 7의 상태전이에 따라 갱신 */
else { /* 테이블 초기 엔트리 등록 */
   table.before_update = FALSE;
   table.value = resolved data;
   table.state = Init;
}
```

(b)

그림 9 모험적 갱신 알고리듬 (a) 예측 테이블 참조 (b) 예측 테이블 갱신

참으로 세트시킨다. 만약 안정상태이고 에이지 카운터가 0이면 어떤 갱신도 지연되지 않음을 표시하고 정상적인 예측 동작을 수행하고, 안정상태가 아닌 경우에는 단순히 에이지 카운터만 증가시킨다. 모험적 갱신 모드이면(speculative_update가 참) 값 필드의 값을 모험적으로 갱신한 후 이 값으로 예측한다.

그림 9의 (b)는 예측 테이블 생신 동작이다. 먼저 모험적 갱신으로 예측된 값이 틀린 경우인가를 조사하고, 이에 해당하면 계속되는 더 이상의 잘못된 예측을 방지하기 위해서 speculative_update 플래그를 거짓으로 상태를 초기상태로 세팅하고 값 필드에 실제 결과 값을 저장하고, 에이지 카운터를 감소시킨다. 또한 갱신 전에 테이블이 할당되지 않았으면(before_update가 거짓) 정상적으로 테이블을 갱신한다(그림 8의 상태전이도 참조). before_update가 참이면 테이블에 할당 후 처음 갱신이므로 초기상태로 세트하고 명령의 결과 값을 저장한다.

에이지 카운터를 이용하여 예측 시 분기 예측 실패로 인하여 잘못 수행된 명령으로 인한 테이블 오염을 고려해야 한다. 즉, 분기 명령의 예측 실패로 인하여 잘못 수행된 명령으로 인해 예측 테이블이 오염되고, 후에 이 테이블 참조 시 오염된 값을 이용한 값을 예측하는 것을 방지해야 한다. 와이드 이슈 프로세서에서는 한 클럭에 많은 명령을 페치하고 이슈하기 때문에 잘못 수행된 명령의 비율이 아주 크다. 그림 13의 SPECint95 벤치마크에서 전체 수행된 명령 중 평균 50.7%, 76.3 %의 명령이 분기 예측 실패로 잘못 수행되어 취소된다. 에이지 카운터의 값이 한 번 잘못되면 이 후 모험적 갱신으로 예측되는 모든 값은 틀리게 된다. 이를 위해서 분기 명령의 예측이 잘못되어 분기 명령 이 후에 잘못 페치되어 수행된 명령을 취소하는 복구동작(recovery operation)을 수행하여 취소되는 명령은 PC로 예측 테이블을 참조하여 에이지 카운터를 감소시킨다.

그림 10은 그림 6의 compress 벤치마크 코드에 대하여 모험적 갱신을 적용하여 수행한 예를 보여주는 그림이다. 그림에서 "에이지"는 에이지 카운터의 값을 표시하는데 "addiu r2, r2, 3"에 표시된 에이지 카운터 값은 명령 페치 시 예측 테이블을 참조할 때의 값이다(즉 아직 증가되기 전의 값으로 참조 후 이 값에 1이 증가된다). "UPDATE"에 표시된 에이지 카운터 값은 예측 테이블 갱신 시에 카운터 값을 1 감소한 후의 값이다 (이 값은 다음 예측 시에 참조되는 에이지 카운터 값이다).

그림 10의 클럭 사이클 31424에서 에이지 카운터가 0이 아닌 1이므로 모험적 갱신 모드가 시작된다. 이 때 예측 값은 " *value*+*stride**(*age*+1)", 즉 "5+3*(1+1)"

클럭			실제값	에이지	예측값	
	.....					
31243:	406D68 addiu	r2,r2,3	5	0	5	
31250	406D68 UPDATE			0		
31421:	406D68 addiu	r2,r2,3	8	0	8	
31424:	406D68 addiu	r2,r2,3	11	1	11	<- 모험적갱신모드 시작
31426:	406D68 addiu	r2,r2,3	14	2	14	
31428:	406D68 UPDATE			2		
31429:	406D68 addiu	r2,r2,3	17	2	17	
31430:	406D68 UPDATE			2		
31431	406D68 addiu	r2,r2,3	20	2	20	
31432:	406D68 UPDATE			2		
31433:	406D68 UPDATE			1		
31434:	406D68 addiu	r2,r2,3	23	1	23	
31436:	406D68 UPDATE			1		
31439:	406D68 UPDATE			0		
31440:	406D68 addiu	r2,r2,3	26	0	26	
31446:	406D68 addiu	r2,r2,3	29	1	29	<- 모험적갱신모드 시작
31447:	406D68 UPDATE			1		
31449:	406D68 addiu	r2,r2,3	32	1	32	
31451:	406D68 addiu	r2,r2,3	35	2	35	
31453:	406D68 UPDATE			2		
31454:	406D68 addiu	r2,r2,3	38	2	38	
31455	406D68 UPDATE			2		
31456:	406D68 addiu	r2,r2,3	41	2	41	
31457:	406D68 UPDATE			2		
31459:	406D68 UPDATE			1		
31459:	406D68 addiu	r2,r2 3	44	1	44	
31461:	406D68 UPDATE			1		
	.....					
	.....					

그림 10 그림 6의 코드에 대한 모험적 갱신 수행 예

으로 계산되어 11이 되고 올바른 값으로 예측됨을 알 수 있다. 이 예측 값 11은 모험적으로 갱신되어 예측 테이블의 값 필드에 저장된다. 다음의 클럭 31426 부터는 모험적 갱신 모드로 동작하여 예측 테이블의 값 필드에 스트라이드를 더하여 예측한다.

그림 6에서 갱신지연으로 틀리게 값을 예측한 명령들이 모험적 갱신을 사용하면 그림 10에 나타난 바와 같이 올바른 예측을 할 수 있음을 알 수 있다.

5. 성능 측정 및 분석

5.1 실험방법

본 논문의 성능 측정을 위해 SimpleScalar/Alpha 3.0 툴셋[20]에서 사이클 수준 시뮬레이터인 sim-outorder에 제안된 예측기를 삽입하여 시뮬레이션하였다. Simple Scalar 머신은 RUU(Register Update Unit)과 5 스테이지 파이프라인을 갖는 비순차 이슈(out-of-order issue) 슈퍼스칼라 프로세서를 모델링한 머신이다. 표 2는 시뮬레이션 시 사용된 머신 파라미터를 보여주는 그림이다. 표2에서 RUU는 비순차 이슈를 위한 명령 윈도우와 리오더 버퍼를 결합한 것으로 큐 구조로 구성되어 있다.

LSQ(load/store queue)는 비순차 이슈 머신에서 메모리 참조 명령어들의 수행된 순서를 유지하기 위해 사용되는 큐이다. 기능장치(functional unit)로 8-이슈(4-이슈)인 경우 8개(4개)의 정수 ALU, 4개(2개)의 실수 가산기, 각각 2개의 정수 및 실수 곱셈기, 나눗셈기를 갖는다. 괄호 안의 값은 기능장치의 수행 레이턴시(latency)를 표시한다. 분기 예측기로는 각각 16K의 엔트리를 갖는 gshare 방식과 bimodal 방식의 혼합형 예측기를 사용하였다[19,20]. 분기 타겟주소가 저장된 BTB(branch target buffer)는 2K 엔트리의 2-웨이 세트 어소시에이티브(set-associative)로 매핑된다. L1 명령 캐시와 데이타 캐시는 각각 256KB의 크기를 가지고

표 2 시뮬레이션 머신 구성 파라미터

	파라미터	값	의미
프로세서 코아	RUU 크기 LSQ 크기 페치 크기 이슈 크기 기능장치	256 64 8(4) 명령/사이클 8(4) 명령/사이클 8(2)개정수 ALU[1] 4(2)개 실수 ADD[2] 2(1)개 정수 MUL[2] DIV[12] 2(1)개 실수 MUL[4] DIV[12]	명령윈도우 + 리오더버퍼 로드/스토아 큐 ( ): 4-이슈 머신 구성 4-이슈 머신 구성,[ ]:레이턴시
분기예측기	BTB 혼합형 분기예측기	2K,2-웨이 16K gshare + 16K bimodal	분기타겟버퍼 gshare + bimodal
캐시 메모리	L1 데이타 캐시 L1 명령 캐시 L2 통합캐시	256KB, 2-웨이, 32 바이트 블록 256KB, 직접매핑,32 바이트 블록 1MB, 4-웨이, 64 바이트 블록	
값 예측기	8K 엔트리 스트라이드 값 예측기		

표 3 벤치마크 프로그램

프로그램	입력	수행명령 수	기본 IPC		분기 예측률		예측 명령비율	
			4-이슈	8-이슈	4-이슈	8-이슈	4-이슈	8-이슈
compress	10000 e 2231	35,261,720	2.1292	2.7738	0.9401	0.9390	0.6849	0.6963
gcc	jump.i	38,772,653	1.4727	1.7692	0.9402	0.9378	0.6792	0.6907
go	5 9	81,530,040	1.3631	1.5204	0.8528	0.8505	0.8224	0.8341
ijpeg	tinyrosc.ppm	63,409,713	2.2733	3.2256	0.9446	0.9435	0.7465	0.7552
li	queen6.lsp	41,524,887	1.8413	2.5894	0.9679	0.9671	0.6224	0.6411
m88ksim	dhry.big.100	240,738,821	3.2317	5.5756	0.9996	0.9995	0.6853	0.6856
perl	scrabbl.in	40,353,113	1.8729	2.3622	0.9711	0.9686	0.6395	0.6501
vortex	persons.250	66,740,617	1.9817	2.7003	0.9874	0.9872	0.5986	0.6012
평 균		76,041,446	2.0207	2.8145	0.95046	0.94915	0.68485	0.69428

L2 통합 캐시는 1MB의 크기를 갖는다.

시뮬레이션에 사용한 SPECint95 벤치마크 프로그램은 표 3과 같다. m88ksim(240M)을 제외하고는 수행되는 명령어의 수가 100M가 되도록 입력을 인가하였다. IPC(Instructions Per Cycle)는 4-이슈, 8-이슈에 대해 값을 예측하지 않는 경우에 측정된 성능이며, 분기 예측률은 시뮬레이션에 사용된 혼합형 분기 예측기의 분기 방향의 예측 성공률이다. 예측 명령비율은 전체 수행된 명령 중 값 예측이 가능한 명령, 즉 레지스터에 값을 저장하는 명령들의 비율을 표시한다.

5.2 성능분석

제안된 모험적 갱신의 스트라이드 예측기의 성능을 분석하기 위해 예측 정확도(prediction accuracy), 성능 향상(speedup) 및 분기실패 후의 복구 효과를 분석한다.

5.2.1 예측 정확도

그림 11은 8-이슈 프로세서에서 기존의 스트라이드 예측기(Stride8f)와 제안된 모험적 갱신을 갖는 스트라이드 예측기(StrideSspRcv8f)의 예측 정확도를 보여주는 그림이다. 예측 정확도는 레지스터에 결과 값을 저장하는 값 예측이 가능한 명령 중에서 안전상태에 도달하여 실제로 값을 예측한 명령 중에서 올바르게 값을 예측한 비율이다. 스트라이드 예측기가 평균 88.8%의 예측 정확도를 보인 반면에 제안된 예측기는 92.7%의 예측 정확도로 약 4% 정도 더 정확하게 예측함을 알 수 있다. 가장 큰 차이로는 compress 벤치마크인 경우에는 약 10% 정도의 예측 정확도의 차이를 보였고, m88ksim의 경우 예측 정확도의 차이가 0.01%로 가장 작은 차이를 보였다.

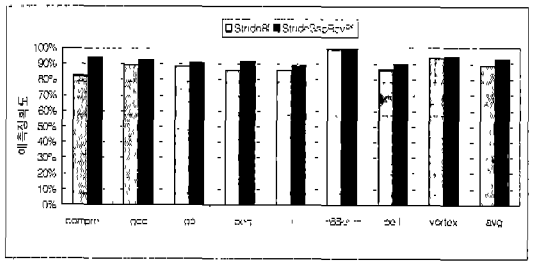
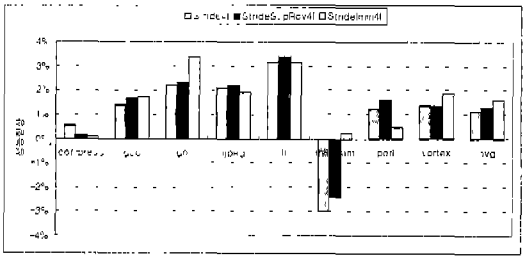


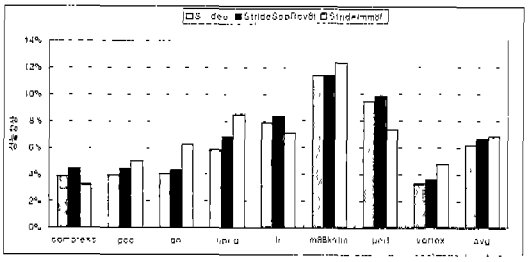
그림 11 예측 정확도 비교

5.2.2 성능 향상

그림 12는 (a)는 4-이슈, (b)는 8-이슈 프로세서에 대하여 값 예측을 시도하지 않은 경우를 기준으로 값 예측을 하였을 경우의 성능 향상(speedup)을 보여주는 그림이다. 그림에서 Stride는 기존의 스트라이드 예측기, StrideSspRcv는 제안된 모험적 갱신의 스트라이드 예측기를 표시하며, StrideImm은 그림 5에서 설명한 바와 같이 예측 테이블의 참조와 동시에 결과 값으로 테이블을 갱신하는 것을 가정한 스트라이드 예측기를 나타낸다. 성능 향상을 위해 먼저 각 예측기의 프로세서에 대해 IPC(Instruction Per Cycle)를 측정해서 이를 값 예측을 하지 않는 기본 프로세서의 IPC와 비교하여 성능 향상을 측정하였다.



(a) 4-이슈



(b) 8-이슈

그림 12 예측기 별 성능 향상 비교

그림 12의 (a)에서 4-이슈 프로세서인 경우 Stride가 1.11%, StrideSspRcv가 1.28%, StrideImm이 1.58%의 평균 성능 향상이 측정되었다. 그림 12의 (b)에 나타난 바와 같이 8-이슈 프로세서인 경우 Stride가 6.22%, StrideSspRcv가 6.65%, StrideImm이 6.81%의 평균 성능향상을 보였다. 제안된 StrideSspRcv가 Stride 보다 4-이슈인 경우 최대 0.3%(평균 0.17%), 8-이슈인 경우 최대 1.1%(평균 0.16%)의 성능이 향상되었다. 그러나 StrideSspRcv의 성능은 StrideImm의 성능과 비교하여 4-이슈인 경우 평균 0.3%, 8-이슈인 경우 평균 0.16% 정도 성능이 떨어졌다. 예측 정확도의 향상보다 실제 수행 성능의 향상이 작은 것은 모든 값 예측이 성능 향상에 기여하는 것이 아니라 이슈 시에 대기하고 있는 데이타 종속 관계가 있는 명령의 값 예측의 경우에만 성능이 향상되기 때문이다. 그림 12 (a)의 4-이슈 결과에서 m88ksim의 경우 Stride4f, StrideSspRcv4f의 경우

값 예측을 함으로써 오히려 성능이 저하되는 결과를 보였다. 이는 값 예측의 실패로 인한 손실이 값 예측으로 성공으로 인한 성능 이득 보다도 더 크기 때문에 발생한 결과이다. 그림 12의 (a)와 (b)에 나타난 바와 같이 값 예측은 명령의 이슈가 커질수록 더 큰 성능 향상을 보임을 알 수 있다(8-이슈인 경우 4-이슈 보다 평균 약 5% 정도 더 높은 성능 향상을 나타냈다.

5.2.3 분기실패 후 복구 효과

그림 13은 4-이슈, 8-이슈 프로세서에서 전체 수행된 명령 중에서 분기예측 실패로 취소되는 명령의 비율을 보여 주는 그림이다. 즉, 전체 수행 명령 중에서 4-이슈인 경우 평균 50.7%, 8-이슈인 경우 평균76.3 %의 명령이 분기명령의 예측 실패로 잘못 수행되어 취소된다. 이들 취소되는 명령의 페치 시에 모험적으로 갱신되면 에이지 카운터는 증가하지만 수행취소로 테이블이 갱신되지 못하는 경우가 발생하여 에이지 카운터가 감소되지 못하는 경우가 발생하고, 이 경우 모험적 예측기의 성능에 영향을 미친다. 따라서 본 논문에서 제안된 모험적 갱신 스트라이드 예측기는 4.2절에서 기술한 바와 같이 이들 명령의 취소 시에 에이지 카운터를 감소시킨다.

그림 14는 8-이슈 프로세서에서 취소되는 명령의 에이지 카운터 복구 효과를 보여주는 그림이다. StrideSsp는 명령 페치 시에 모험적 갱신을 하지만 분기예측 실패로 취소되는 명령의 에이지 카운터 감소를 하지 않은 경우를 나타내며, StrideSspRcv는 에이지 카운터를 감소시켜 복구하는 모험적 갱신의 스트라이드 예측기이다. 그림 14에 나타난 바와 같이 에이지 카운터의 복구로 복구하지 않는 경우보다 최대 2.97% (평균 1.4%) 정도의 성능이 향상되었다.

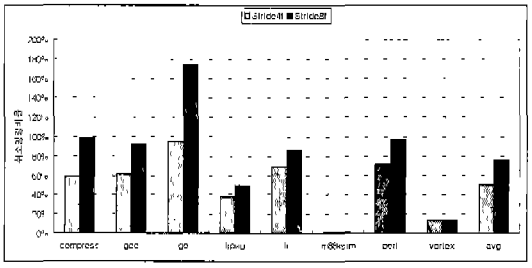


그림 13 분기예측 실패로 취소되는 명령비율

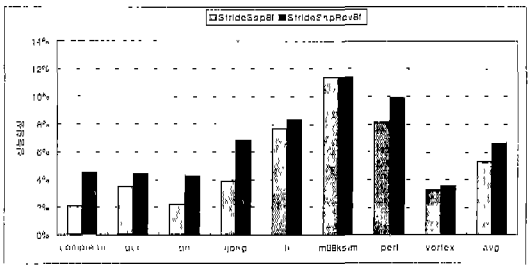


그림 14 분기예측 실패 시 에이지 카운터 복구 효과

6. 결 론

본 논문에서는 예측 테이블 갱신의 지연으로 인한 성능 저하를 줄이기 위해 명령의 결과가 나올 때까지 기다리지 않고 예측 테이블 값을 모험적으로 갱신하는 스트라이드 값 예측기를 제안하였다. 제안된 스트라이드 예측기는 지연 갱신되는 예측 테이블 엔트리를 추적하고 테이블의 값을 모험적으로 갱신하기 위해 예측 테이블에 에이지 카운터를 도입하였다. 에이지 카운터는 테이블 참조 시에는 증가하고 테이블 갱신 시에는 감소하여 갱신의 지연을 감지하였다. 분기예측 실패로 인하여 취소되는 명령들에 대해서는 에이지 카운터의 복구를 위해 이들 명령이 취소될 때 에이지 카운터의 값도 복구하였다.

제안된 방식의 타당성을 검증하기 위해 사이클 수준에 시뮬레이션이 가능한 SimpleScalar 시뮬레이터 상에 제안된 예측기를 구현하여 SPECint95 벤치마크를 시뮬레이션하였다. 실험 결과 제안된 모험적 갱신의 스트라이드 예측기가 기존의 스트라이드 예측기 보다 최고 10% 정도 높은 예측 정확도를 보였다. 또한 8-이슈 프로세서인 경우 기존의 스트라이드 예측기보다 최고 1.1% 정도 성능이 향상됨을 보여 기존의 값 예측방법보다 성능이 좋음을 실험결과로 확인 하였다. 향 후 제안된 모험적 갱신의 스트라이드 예측기를 적용한 혼합형 예측기를 구현하면 좀 더 높은 성능향상이 기대된다.

참 고 문 헌

[1] S.A.Mahlke, "Exploiting Instruction Level Parallelism in the Presence of Conditional Branches," Ph.D dissertation, Department of Electrical and Computer Engineering, University of Illinois. Urbana, IL, 1996.

[2] T.Heil, Z.Smith, and J.Smith, "Improving Branch Predictors by Correlating on Data Values," Proceedings of the 32nd International Symposium Microarchitecture (MICRO-32), Nov. 1999.

[3] R.Rakvic, B.Black, and P.Shen, "Completion Time Multiple Branch Prediction for Enhancing Trace Cache Performance," Proceedings of the 27th Annual International Symposium on Computer

Architecture (ISCA-27), June 2000.
[4] K.Skadron, M.Martonosi, and D.Clark. "Speculative Updates of Local and Global Branch History" A Quantitative Analysis," Journal of Instruction-Level Parallelism, vol.2, Jan. 2000.
[5] T. Yeh and Y.Patt, "Two-Level Adaptive Branch Prediction," Proceedings of the 24th International Symposium microarchitecture (MICRO-24), Nov. 1991.
[6] B.Calder, B.Reinman and D.Tullsen, "Selective Value Prediction," Proceedings of the 26th International Symposium on Computer Architecture (ISCA-26), May 1999
[7] F.Gabbay, and A.Mendelson, "Speculative Execution Based on Value Prediction," EE Department TR 1080, Technion-Israel Institute of Technology, Nov, 1996
[8] F.Gabbay, and A.Mendelson, "The Effect of Instruction Fetch Bandwidth on Value Prediction," Proceedings of the 25th International Symposium on Computer Architecture (ISCA-25), p.272-281, 1998.
[9] J.Gonzalez and A.Gonzalez, "The Potential of Data Value Speculation to Boost ILP," Proceedings of the International Conference on Supercomputing, 1998.
[10] Sang-Jeong Lee and P.Yew, "Decoupled Value Prediction on Trace Processors," Proceedings of the 6th International Symposium on High Performance Computer Architecture (HPCA-6), 2000.
[11] Sang-Jeong Lee and P.Yew, "On Some Implementation Issue for Value Prediction on Wide-Issue ILP Processors," International Conference on Parallel Architecture and Compilation Techniques (PACT 2000), Oct. 2000.
[12] Sang-Jeong Lee, and Pen-Chung Yew, "On Table Bandwidth and Its Update Delay for Value Prediction on Wide-Issue ILP Processors," IEEE Transaction on Computers, Vol.50 No.8, Aug. 2001
[13] M.Lipasti, and J.Shen, "Exceeding th Limit via Value Prediction," Proceedings of the 29th International Symposium on Microarchitecture(MICRO-29), Dec. 1996
[14] M.Lipasti, Value Locality and Speculative Execution, Ph.D. thesis, Carnegie Mellon University, April 1997.
[15] B. Rychlik, F.Faistl, B.Krug, A.Kurland, J.Jung, Miroslav, N.Velev, and Jshen, "Efficient and Accurate Value Prediction Using Dynamic Classification," Technical Report of Microachitecture Research Team in Dept of Electrical and Computer Engineering, Carnegie mellon Univ., 1992.
[16] B.Rychlik, J.Faistl, B.Krug, and J.Shen, "Efficacy and Performance Impact of Value Prediction," Parallel Architectures and Compilation Techniques, Paris, Oct. 1998.
[17] T.Sato, "Analyzing Overhead of Reissued Instructions on Data Speculative Processors," Workshop on Performance Analysis and its impact on Design Held in Conjunction with ISCA-25, 1998.
[18] Y.Sazeides, and J.Smith, "The Predicatability of Data Values," Proceedings of the 30th International Symposium on Microarchitecture (MICRO-30), Dec. 1997.
[19] K.Wang, M.Franklin, "Highly Accurate data value Predictions using Hybrid Predictor," Proceedings of the 30th International Symposium on Microarchitecture (MICRO-30), Dec. 1997.
[20] D.Burger and T.Austin, "The Simplescalar Tool Set, Version 2.0," Technical Report CS-RT-97-1342, University of Wisconsin, Madison, June 1997
[21] T.Sherwood and B.Calder, "Loop Termination Prediction," Technical Report CS99-639 of University of California, San Diego, Dept. of Computer Science and Engineering, Dec. 1999
[22] J.Huang and D.Lilja, "Exploiting Basic Block Value Locality with Basic Block Reuse," Proceedings of the 5th International Symposium on High Performance Computer Architecture (HPCA-5), 1999.
[23] M.Johnson, Superscalar Microprocessor Design, Prentice Hall, 1991

전 병 찬
1992년 2월 한밭대학교 전자계산학과(공학사). 1994년 2월 수원대학교 전자계산학과(이학석사). 1997년 3월 ~ 현재 순천향대학교 전자계산학과 박사과정 재학중. 관심분야는 고성능 프로세서 설계, 마이크로 프로세서 응용, 최적화 컴파일러

이 상 정
1983년 2월 한양대학교 전자공학과 졸업(공학사). 1985년 2월 한양대학교 대학원 전자공학과 졸업(공학석사). 1988년 8월 한양대학교 대학원 전자공학과 졸업(공학박사). 1988년 9월 ~ 현재 순천향대학교 공과대학 컴퓨터학부 교수. 관심분야는 고성능 프로세서 설계, 최적화 컴파일러, 마이크로프로세서 응용