특집

임베디드 병렬 구조에서 예측 기술

조영일(수원대학교 컴퓨터학과)

I. 서 론

최근 프로세서 속도경쟁이 PC용 프로세서와 노트북용 모바일 프로세서에서 임베디드 프로세서로 옮겨가고 있으며, 임베디드 프로세서도 가까운 장래에 1Ghz 시대에 진입할 전망이다.

임베디드 시스템은 경량화 및 저 전력의 실현을 위해서 시스템 칩(SoC) 형태로 설계되고 있다. 임베디드 시스템의 응용 분야가 다양화됨에 따라 디지털신호처리기, 마이크로컨트롤러나 네트워크프로세서등과 같은 임베디드 프로세서들의 하드웨어 구조가 응용분야에 적용시키기 위해 계속적으로 변화하고 있고, 성능도 급속히 향상되고 있다.

현재 고성능 프로세서에서 발생하는 전력 손실의 최대 원인은 누설 전류이다. 누설 전류는 트랜지스터가 최대 속도를 낼 수 있도록 설계될 때 증가되므로 칩의 동작 속도를 약간 느리게 설계한다면 상당량의 전력 소모를 감소시킬 수 있다.

마이크로 아키텍처에서 비순서적(out-of-order) 수행과 모험적(speculative) 수행 같은 특징들은 성능 이득을 얻을 수 있는 반면에 전력 사용을 크게 증가시킨다. 비순서적 수행과 모험적 수행을 허용하는 프로세서는 실행시간에 어떤 명령을 언제 실행해야 하는지 결정하기 위해 많은 하드웨어를 사용하기 때문이다. 이러한 하드웨어는 명령을 실행하기 위해 전력을 소모하는데, 성능을 극대화시키려면 필요한 명령어들만 실행시켜야한다. 만약 필요하지 않은 명령어들까지 실행한다면 이는 분명히 전력 낭비로 이어질 수밖에 없을 것이다.

프로세서의 성능을 향상시키고 전력 소모를 줄이기 위한 메커니즘으로 분기명령어가 반입될 때, 분기명령어의 결과를 미리 예측하고 예측된 값을 적용하여 바로 다음 사이클에 현재의 분기명령어 이후에 수행될 명령어를 미리 반입해서 수행하여 분기결과가 결정될 때까지 파이프라인의 지연을 제거시키는 방법을 분기예측 기법이라 한다.

선행 명령어의 결과 값을 후행 명령어가 입력으로 사용하는 실 데이터 종속관계(true data dependency)의 명령어는 선행 명령어의 결과가 구해질 때까지 지연되어야 하며, 이는 병렬 수행을 제약하여 프로세서의 성능을 감소시킨다. 이런 문제를 해결하여 프로세서의 성능을 향상시킬 수 있는 또 다른 메커니즘으로 데이터 종속적

인 명령어가 선행 명령어의 결과 값이 나올 때까지 기다리는 것이 아니라, 선행 명령어의 결과를 미리 예측하여 모험적으로 실행하므로 서 데이터 종속 관계를 제거하는 하드웨어 메커니즘을 결과값 예측기법이라 한다.

본고에서는 임베디드 병렬 구조를 갖는 프로세서에서 전력소모를 줄이고, 성능을 향상시킬 수 있는 다양한 분기예측기법과 결과값 예측기법을 소개하고 그들을 장단점을 고찰해 본다.

II. 분기 예측기 기술

프로그램이 수행되는 과정에서 발생되는 분기명령어의 비율은 전체 명령어 중 약 20%~30% 정도이다. 따라서 명령어 4~5개당 1개씩 분기명령어가 발생하게 될 수밖에 없는 프로그램의 특성과 파이프라인의 고성능화로 명령어의 이슈 폭이 크게 증가하게 된 최근의 프로세서들은 매 사이클마다 하나 이상의 분기명령어를 처리하게 되며. 또, 매 사이클마다 분기명령어 결과에 대한 제어종속성으로 인한 사이클 지연이 발생함에 따라 프로세서의 성능은 크게 감소된다. 프로세서의 성능감소를 줄이기 위해 분기명령어가 반입될 때, 분기명령어의 결과를 미리 예측하고 예측된 값을 적용하여 바로 다음 사이클에 현재의 분기명령어 이후에 수행될 명령어를 미리 반입해서 수행하고, 또다시 예측하고 또 미리 반입하는 동작을 반복하게 되는데, 이를 분기예측기법이라 한다. 이는 분기명령어의 결과인 분기방향을 미리 예측하여 "taken" 또는 "not taken"으로 분기방향을 결정한 후, 뒤이어지는 명령어를 반입하고, BTB(Branch Target Buffer)를 이용하여 반입된 명령어의 목적지 주소(destination address)를 예측하는 방법이다.

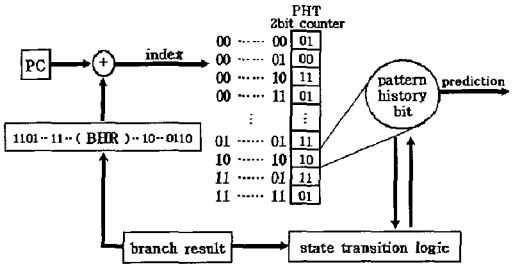
분기방향을 예측하는 방법은 크게 두 가지로 구분되는데, 컴파일 시간에 프로그램을 profiling한 통계결과를 이용하여 분기명령어에 분기결과를 미리 인코드(encode)한 뒤 실제 프로그램 수행 시 이용하는 정적 분기예측 기법과 프로그램 수행 중 현재 시점 이전에 발생한 결과를 누적시켜 이를 바탕으로 분기예측을 수행하는 동적 분기예측 기법으로 구분된다.

정적 분기예측 기법은 임의의 입력 데이터로 프로파일(profiling)한 정보의 통계결과를 이용하는 방법이므로 실제 데이터에 의한 명령어 수행 시 입력 데이터의 내용이 임의의 데이터와 서로 다르므로 잘못된 예측결과가 도출될 수 있다는 단점이 있고, 동적 분기예측 기법은 이전에 처리되었던 분기결과들에 대한 누적결과 값들을 히스토리로 사용하여 예측하므로 정적 예측방식보다 높은 정확도를 갖는 반면, 분기명령어의 처리결과를 예측하기 위해서는 일정량 이상의 분기명령어들의 수행기록을 축적시켜야 하므로 분기예측을 위해 일정량 이상의 수행기록이 생성되기 전까지는 분기명령어에 대한 예측이 이루어질 수 없다는 문제점이 있다.

이 장에서는 대표적인 동적 분기예측 기법을 중심으로 그들의 동작 메커니즘과 특징을 알아본다.

1. 2-단계 적응훈련 분기예측기법

2-단계 적응훈련 분기예측기법(Two-Level Adaptive Training Branch Prediction)은 프로그램 수행시작 시점부터 현 상태 바로직전 까지 수행되었던 선행 분기명령어들의 분기결과를 기록하여 보관하는 역할을 담당하는 쉬프트 레지스터(shift register)구조를 갖는 BHR과 분기명령어



〈그림 1〉 2-단계 적응훈련 분기예측기

별로 분기패턴을 누적시키는 역할을 담당하는 2비트 포화카운터(Saturating counter)에 의한 PHT(Pattern History Table)에 의해 <그림 1>과 같이 구성된다.

분기명령어들 간에는 상호 연관성이 존재하는데 BHR은 이 연관성을 이용하는 것으로 레지스터의 크기가 n비트인 경우 최근 n개의 분기명령어들의 분기결과를 저장하게 된다. 분기명령어의 수행이 완료되면 BHR은 분기명령어가 수행되어 결과를 얻을 때 마다 1비트씩 왼쪽으로 쉬프트(shift)시키고 n번째 비트에 가장최근에 수행된 분기명령어의 결과가 "taken"이면 "1"을 "not taken"이면 "0"을 저장한다.

BHR의 비트 수에 따라 크기가 결정되는 PHT는 BHR이 n비트 레지스터이면 2^n개의 엔트리(entry)를 갖게 되는데 PHT의 각각의 엔트리는 2비트 포화카운터이며 분기명령어의 주소(program counter)와 BHR의 내용을 XOR(exclusive OR)시켜 인덱스 한다. 이와 같은 방식을 사용하는 분기예측기를 gshare 예측기라 하며 BHR의 값만으로 인덱스 하는 예측기를 GAg 예측기라 한다. 분기명령어의 주소(program counter)는 분기명령어의 지역성을 위해 사용되고 BHR은 분기명령어들 간의 연관성을 도출하여 대표 값을 생성하기위해 사용된다. PHT 엔트리의 카운터는 분기결과가 "taken"이면 1을 증가시키고 "not taken"이면 1을 감소시킨다.

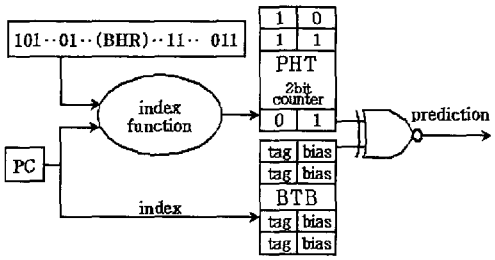
분기명령어의 예측은 인덱스 된 PHT 엔트리의 카운터 값이 2이상이면 "taken"으로 1이하이면 "not taken"으로 예측하여 fall through 필드의 주소와 일치되는 해당명령어를 반입하여 수행하고, "taken"으로 예측 되면 BTB(Branch Target Buffer)에서 "taken" 방향에 있는 해당목적주소로 분기하게 된다.

2. Agree 예측기

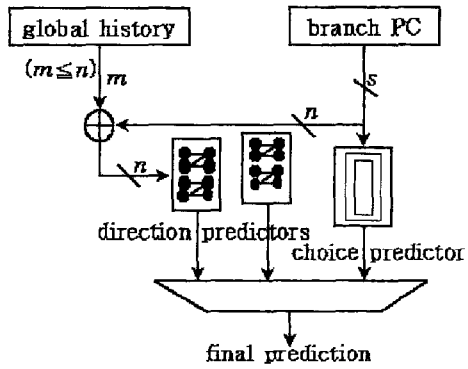
2-단계 적응훈련 분기예측기법의 경우 PHT의 엔트리수가 제한적일 수밖에 없으므로 어떤 프로그램의 수행을 위한 명령어스트림 안에 2개 이상의 분기명령들(unrelated)이 PHT의 동일 엔트리로 매핑(mapping) 될 수 있으며 이로 인한 간섭(interference)이 발생하게 되는데 이것이 2-단계 적응훈련 분기예측기법의 주된 단점이다.

간섭은 어떤 명령어의 분기결과가 다른 분기명령어의 분기를 틀리게 예측하게 하는 원인을 제공하여 부정적인 영향을 주는 부정적 간섭(negative interference), 바르게 예측하게 하는 원인으로 작용하여 긍정적 영향을 주는 긍정적 간섭(positive interference), 아무런 영향을 주지 않는 중립적 간섭(neutral interference)으로 구분될 수 있으며, 부정적 간섭은 분기예측 정확도를 감소시키게 되므로 제거되어야한다.

간섭을 제거하기위해 <그림 2>와같이 설계된 Agree예측기법은 부정적 간섭을 긍정 또는 중립적 간섭으로 변환시켜 예측정확도를 개선시키려는 방법이다. BTB의 각 엔트리에 bias비트를 추가하여 PHT를 갱신할 때 분기결과가 bias비트와



〈그림 2〉 Agree 예측기법의 예

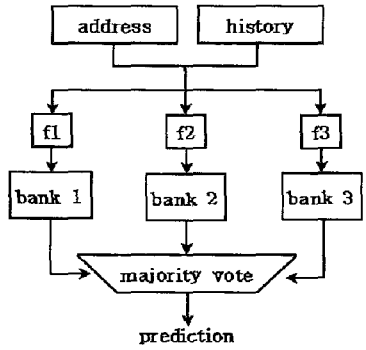


〈그림 3〉의 Bi-mode Predictor

일치하면 2비트 포화카운터를 증가시키고, 일치하지 않으면 감소시키며, 선택된 PHT의 엔트리 카운터의 값이 2이상이면 bias비트로 예측하고 1 이하이면 bias비트의 반대 값으로 예측하므로 서 부정적 간섭을 긍정적 간섭으로 변환시킨다.

3. Bi-mode 예측기

Bi-mode 예측기(그림 3)는 부정적 간섭을 제거하기위해 PHT를 "choice PHT"와 "direction PHT"로 분할 구분(taken direction PHT, not taken direction PHT)하여 "taken"으로 bias되는



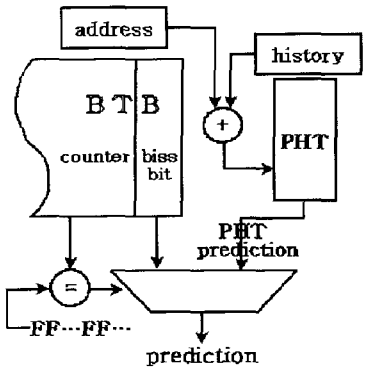
〈그림 4〉 Skewed Branch Predictor

분기들은 "taken direction PHT"에서 "not taken"으로 bias되는 분기들은 "not taken direction PHT"에서 예측하게 하므로 서 부정적 간섭을 제거한다.

4. Skewed 예측기

발생되는 간섭의 대부분은 PHT의 크기가 작아서 발생되는 것이 아니고 PHT의 associativity 부족으로 발생된다는 관찰 결과를 근거로 하였을 때 간섭을 제거하는 가장 좋은 방법은 PHT를 "set associative"하게 하는 방법이나 이 방법은 태그가 요구되므로 하드웨어 비용 측면에서 효과적이지 못하다.

Skewed 분기예측기는 <그림 4>과 같이 PHT를 서로 다른 메커니즘을 갖고 있는 3개의 "bank"를 갖는다. 각 "bank"는 주소와 전역 히스토리에 의한 독자적 해싱함수(f1, f2, f3)를 사용하여 엔트리를 선택한다. 3개 "bank"의 다수결과 값에 따라 분기결과를 예측하는 기법으로 예측결과에 따라 예측이 틀렸으면 모든 "bank"를 갱신하고 예측이 맞았으면 올바르게 예측한 "bank"만 갱신하는 부분갱신(partial update) 방식을 사용하므



〈그림 5〉 Filter 예측기

로 서 부정적 간섭을 최소화 시킨다.

5. Filter 예측기

PHT에 중복되어 저장되는 정보의 양을 감소시켜 PHT의 기능을 극대화시키려는데 주된 목적을 두고 있는 "Filter 메커니즘"은 "highly biased branch"는 1비트를 사용하여 높은 예측 정확도를 유지할 수 있다는 것이 기본 구상이다. <그림 5>와 같이 PHT로부터 "highly biased branch"의 "filtering"은 BTB 엔트리에 첨가된 bias비트와 포화 카운터에 의해 수행되며 분기가 BTB 엔트리에 할당될 때 bias비트는 분기방향 값으로 설정하고 카운터를 초기화 하며, 분기결과가 결정된 후 분기방향이 bias비트와 같으면 카운터를 증가시키게 되며 분기방향이 bias비트와 다르면 카운터를 감소시켜 카운터가 0이면 bias비트의 값을 반대로(toggle)한다. 또, 카운터가 포화상태가 아니면 PHT로 예측하고 카운터가 포화상태이면 "highly bias"되는 분기이므로 BTB의 bias비트에 의해 지시된 방향으로 예측한다.

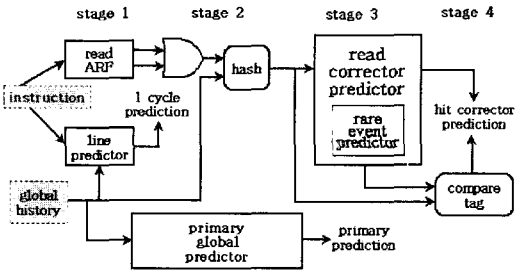
BTB의 해당 엔트리의 카운터가 포화상태일 때 분기 결과로 PHT를 갱신시키지 않으므로 해당 분기는 PHT로부터 "filtering"된다. 카운터의 크기가 PHT 크기에 의해 조정되는 방식으로 PHT 크기가 크면 "filtering"의 필요성이 작아지므로 카운터의 크기를 크게 한다. 따라서 "filter 메커니즘"은 PHT에 저장되는 정보의 양을 감소시켜, 중립적 및 부정적 간섭을 감소시키는 방법이다.

6. 다단계 예측기

파이프라인의 깊이와 이슈 폭이 커질수록 더 정확한 분기예측기가 요구되고, 이로 인해 분기예측기의 크기는 더 커지고 처리방법 또한 더 복잡해지게 되지만 크고 복잡해지는 만큼 예측을 하는데 걸리는 시간이 길어지므로 분기 명령어 반입시간에 예측을 하지 못하게 된다. 또한 예측이 실패했을 경우 많은 사이클이 낭비되며, 그동안 잘못된 경로의 명령어를 불필요하게 수행했으므로 전력을 낭비하게 된다.

이와 같은 문제를 해결하고자 여러 개의 예측기를 다단계로 구성하는 기법들이 최근의 추세이다. 즉, 명령어 반입시간에 간단하고 속도가 빠르나 예측정확도가 떨어지는 간단한 예측기를 사용하여 예측하고, 몇 사이클 후에 복잡하여 속도가 느린 반면 예측결과가 비교적 정확한 예측기로 예측하여 간단한 예측기의 예측 결과와 일치하면 계속 진행시키고, 일치하지 않으면 간단한 예측기에 의해 수행된 경로를 취소시키고, 정확한 예측기에 의해 예측된 경로를 반입하여 수행한다.

다단계 예측기는 <그림 6>과 같이 간단한 예측기(line predictor)에 의해 명령어 반입시간(stage 1)에 분기예측을 가능하게 하며, 간단한



〈그림 6〉 다단계 예측기

예측기가 예측을 실패했을 경우 분기결과가 구해질 때까지 기다리지 않고, 정확한 예측기(rare event predictor)의 예측결과로 간단한 예측기의 예측 실패를 복구하므로 서 예측실패에 대한 페널티를 감소시켜 성능 향상을 이룩한다.

III. 결과값 예측기 기술

임베디드 병렬구조 프로세서에서 명령어의 반입 폭과 이슈 폭이 계속 증가하고 있으므로 병렬 수행에 있어서 프로세서의 속도를 높이기 위한 여러 가지 기술들이 계속 연구 개발되고 있다. 또한 집적회로의 제조기술이 발전함에 따라 클럭 주파수가 계속하여 높아지고 있으며, 컴파일러의 최적화와 명령어 세트(instruction set)의 개선에 힘입어 파이프라인 구조(pipeline architecture)의 효율이 점차 향상되고 있다. 컴퓨터에서 가장 큰 성능향상을 얻을 수 있는 기술은 프로세서의 구조를 개선하여 한 사이클에 다수의 명령어를 병렬로 수행시키는 것이다.

선행 명령어의 결과값을 후행 명령어가 입력으로 사용하는 실 데이터 종속관계(true data dependency)의 명령어는 선행 명령어의 결과가 구해질 때까지 지연되어야 하며, 이는 병렬 수행을 제약한다. 이러한 문제를 해결하기 위해서 결과값 예측기법(data value prediction mechanism)을 사용하게 된다. 결과값 예측기법은 데이터 종속적인 명령어가 선행 명령어의 결과값이 나올 때까지 기다리는 것(instruction stall)이 아니라, 예측된 선행 명령어의 결과를 미리 예측(prediction)하여 모험적으로 실행(speculative execution)함으로써 데이터 종속 관계를 제거하는 하드웨어 메커니즘이다.

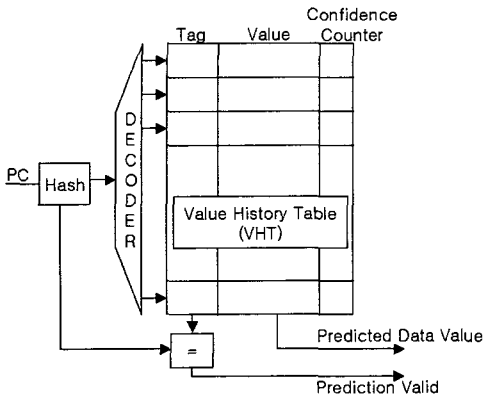
이 장에서는 기존의 대표적인 결과값 예측기인 최근 결과값 예측기, 스트라이드 결과값 예측기, 2-단계 결과값 예측기, 혼합형 예측기의 구조를 살펴보고 각 예측기의 장단점을 고찰한다.

1. 최근 결과값 예측기

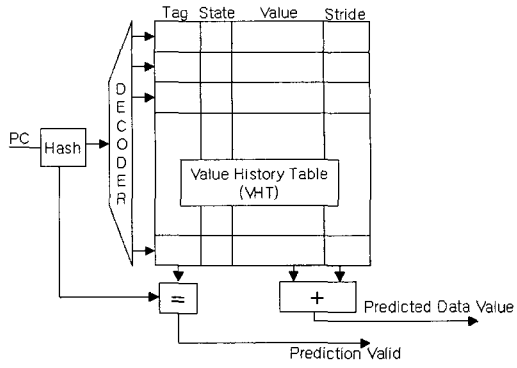
Lipasti는 프로그램의 실행동안 대부분 명령어들의 결과값 변화가 적음을 주목하고 레지스터 값을 변경하는 명령어들에 대해 재사용되는 결과값의 지역성(locality)을 실험하였다. 실험결과 1개의 값만을 재사용하는 경우가 평균 49%, 4개의 값 범위 내에서 재사용되는 경우가 평균 61% 정도임을 측정하고, Lipasti는 이를 근거로 최근 결과값 예측기(last value predictor)를 제안하였다.

최근 결과값 예측기는 명령어가 가장 최근에 생성한 결과값을 예측 테이블에 저장하여, 다음에 동일 명령어를 반입할 때, 바로 이전의 수행 시 저장된 결과값을 예측값으로 사용하는 방법이다.

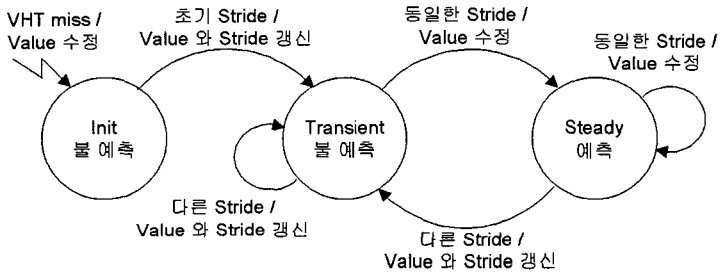
<그림 7>은 최근 결과값 예측기의 구조를 보여주고 있다. VHT(Value History Table)의 각 엔트리는 태그(Tag) 필드와 결과값(last Value) 필드로 구성된다. 태그 필드는 해당 엔트리를 식별하기 위해 현재 매핑된 명령어의 주소를 저장하고,



〈그림 7〉 Last 결과값 예측기의 구조



〈그림 8〉 스트라이드 결과값 예측기의 구조



〈그림 9〉 스트라이드 결과값 예측기의 상태 전이도

결과값 필드는 수행 명령어의 가장 최근의 결과값을 저장한다.

예측되는 명령어의 주소(PC)로 VHT 테이블을 인덱스하고, 태그 필드가 적중하면 해당 엔트리에 저장된 결과값을 예측값으로 사용하게 된다. 최근 결과값 예측기는 VHT의 각 엔트리에 마지막으로 생성된 결과값 하나만을 저장하기 때문에 적은 하드웨어 비용을 요구하지만 변하지 않는 값을 생성하는 명령어들만 예측할 수 있어서 예측 정확도가 40%~50%로 낮다는 단점을 갖는다.

2. 스트라이드 결과값 예측기

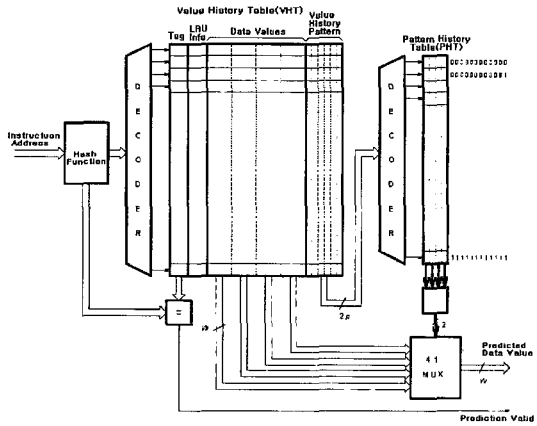
Gabbay와 Wang등이 제안한 스트라이드 결과값 예측기(stride value predictor)는 한 명령어의 결과값들이 변화되는 스트라이드를 검사하여 이들 결과값들이 일정한 간격으로 변한다면 그 명령어의 이후 결과값의 예측이 용이함을 착안하여 설계되었다. 스트라이드 결과값 예측기는 마지막으로 수행된 명령어의 결과값과 마지막 두 번의 수행된 결과값의 차이값(stride)을 VHT에 저장하여 이후 동일 명령어의 반입 시 마지막

수행 결과값과 차이값(stride)의 합으로 결과값을 예측한다.

<그림 8>는 스트라이드 결과값 예측기의 구조를 보여주고 있다. 스트라이드 결과값 예측기의 VHT 엔트리는 태그(Tag) 필드, 상태(State) 필드, 결과값(Value) 필드, 차이값(Stride) 필드로 구성된다. 상태 필드는 2비트로 구성되며 3개의 상태(Init, Transient, Steady)를 가진다.

다음 스트라이드 시퀀스를 검출하기 위해 상태 필드는 <그림 9>과 같은 상태 전이를 한다. 처음 만나는 명령어는 결과값(V1)을 생성할 때 VHT에 하나의 엔트리를 할당하여 결과값을 해당 엔트리의 결과값 필드에 저장하고, 상태 필드의 값을 초기 상태(initial state:"Init")로 설정한다. 초기("Init") 상태에서 동일한 명령어와 그 결과값(V2)을 생성하게 되면 이전 결과값(V1)과의 차(S1=V2-V1)와 그 결과값(V2)를 해당 엔트리의 결과값 필드와 스트라이드 필드에 저장한다. 이때 상태 필드는 과도기 상태(transient state:"Transient")로 설정된다.

초기(Init) 상태와 과도기(Transient) 상태는 예측을 위해 준비중인 단계로 예측을 수행하지 않는다. 과도기(Transient) 상태에서 동일한 명령어가 결과값(V3)을 생성하게 되면 새로운 스트라이드 값(S2=V3-V2)을 계산하여 해당 엔트리의 결과값 필드와 스트라이드 필드를 변경한다. 이때 새로운 스트라이드 값(S2)이 이전 스트라이드 값(S1)과 동일하다면 상태 필드는 안정 상태(Steady state:"Steady")로 변경되고, 동일하지 않은 경우에는 과도기(Transient) 상태를 유지한다. 안정(Steady) 상태가 유지되는 동안에만 결과값 필드의 값과 스트라이드 필드의 합으로 예측값을 제공한다. 안정(Steady) 상태에서 새로운 스트라이드 값이 이전 스트라이드 값과 다르다면 과



〈그림 10〉 2-단계 결과값 예측기의 구조

도기(Transient) 상태로 전이하여 같은 과정을 반복한다.

스트라이드 결과값 예측기는 반복문 안에서 사용되는 변수(loop induction variables)와 배열을 통해 순차적으로 증감하는 패턴을 잘 예측한다. 스트라이드 예측 방법은 일정한 값을 갖는 패턴(zero stride)과 일정하게 증감하는 패턴 모두를 예측할 수 있어서 최근 결과값 예측기보다 예측 정확도가 높지만, 반복형 비스트라이드 시퀀스 패턴과 같이 규칙적인 패턴을 결과값으로 생성하는 명령어들에 대해서는 예측하지 못한다는 단점이 있다.

3. 2-단계 결과값 예측기

Wang등은 기존의 분기 예측(branch prediction)에서 사용된 2-단계 적응 분기 예측(two-level adaptive branch prediction) 방식을 결과값 예측기에 적용한 2-단계 결과값 예측기(two-level data value predictor)를 개발하였다. 2-단계 결과값 예측기는 <그림 10>와 같이 VHT

와 PHT(Pattern History Table)의 두 개의 테이블로 구성된다.

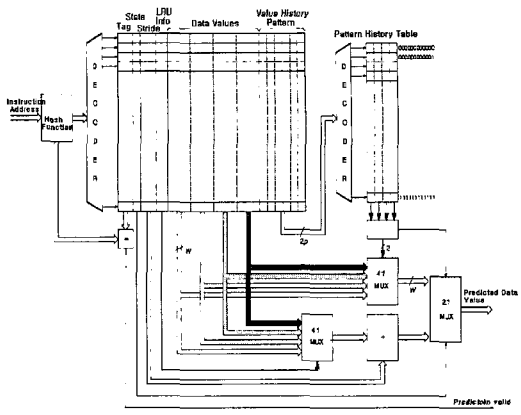
예측한 명령어의 주소로 인덱스되는 VHT 테이블의 각 엔트리는 태그(tag) 필드, LRU (LRU_info) 필드, 네 개의 결과값(Data Values) 필드, VHP(Value History Pattern) 필드로 구성된다. 결과값 필드는 최근에 사용된 결과값들 중에서 중복되지 않은 서로 다른 네 개의 결과값을 저장하며, LRU 필드는 결과값 필드에 저장된 값들의 사용된 순서에 대한 정보를 가지고 있다. VHP 필드는 결과값의 중복과 관계없이 단순히 결과값 필드에서 6 개의 결과값이 나타난 순서대로의 위치 정보를 저장한다.

VHP의 내용으로 인덱스되는 PHT는 VHT의 결과값 필드에서 어떤 값으로 예측을 수행할지 결정하기 위해 네 개의 카운터 필드를 사용한다. 이 카운터 필드 중에서 최대값으로 설정되어 있는 위치에 대응되는 VHT의 결과값을 예측값으로 사용하게 된다. 명령어 수행 후 생성된 결과값이 VHT의 결과값 필드에 없는 경우에는 LRU 필드의 정보를 참조하여 최근에 가장 적게 사용된 결과값을 새로운 결과값으로 대체한다.

2-단계 결과값 예측기는 높은 예측 정확도를 갖지만, 여러 개의 결과값을 저장하기 때문에 많은 하드웨어 비용이 요구된다는 단점을 갖는다.

4. Wang의 혼합형 결과값 예측기

명령어에 의해 생성되는 결과값의 시퀀스는 다양한 형태로 나타난다. 각각의 예측기는 각 예측기에 적합한 특정 결과값의 시퀀스에서는 좋은 결과를 보이지만 다른 형태의 결과값 시퀀스에 대해서는 예측율이 떨어지는 문제점이 있다. 따라서 이들 예측기의 장점을 결합한 혼합형 결과값 예측기들이 개발되었다.

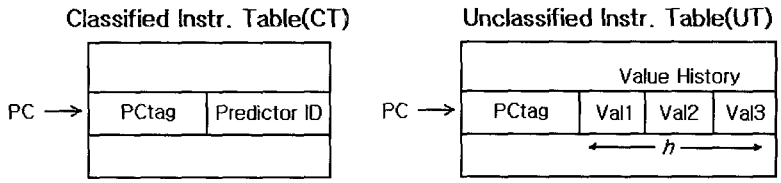


〈그림 11〉 혼합형 결과값 예측기의 구조

Wang 등은 벤치마크 프로그램 특성에 따라서는 스트라이드 결과값 예측기가 2-단계 결과값 예측기보다 예측 정확도가 더 높은 사실에 주목하고 스트라이드 결과값 예측기와 2-단계 결과값 예측기를 결합한 혼합한 결과값 예측기(hybrid value predictor)를 제안하였다.

<그림 10>은 Wang의 혼합형 결과값 예측기의 구조를 보여준다. 혼합형 결과값 예측기는 일정한 값으로 증감하는 명령을 예측하는 스트라이드 결과값 예측기와 2-단계 결과값 예측기를 결합한 것으로 신뢰성 카운터(confidence counter)의 값에 의하여 두 예측기 중 한 예측기로부터 결과값을 예측하는 방법이다. 명령어가 두 개의 예측기에서 모두 엔트리를 갖고 있다면, 카운터의 값이 높은 예측기의 예측 값을 선택한다.

Wang의 혼합형 결과값 예측기는 스트라이드의 특성을 갖는 명령어와 다양한 패턴을 갖는 명령어를 모두 예측하게 됨으로써 높은 예측 정확도를 얻을 수 있지만, 하나의 명령어가 두 개의 예측기 테이블 엔트리에 중복 저장되어 높은 하드웨



〈그림 12〉 Rychlik의 호납형 예측기의 분류 테이블

어의 비용을 필요로 한다는 단점을 갖고 있다.

5. Rychlik의 혼합형 결과값 예측기

Rychlik등이 제안한 혼합형 결과값 예측기는 최근 결과값 예측기, 스트라이드 결과값 예측기, fcm 결과값 예측기를 혼합한 예측기이다. Rychlik 등은 분류 테이블을 이용하여 실행 시간 동안 명령어들의 패턴을 분류함으로써 가장 잘 예측할 수 있는 예측기에 할당하는 동적 분류(dynamic classification) 방법을 사용하였다.

Rychlik등의 동적 분류는 새로운 명령을 만나면 바로 예측기를 지정하는 것이 아니라 <그림 11>과 같은 분류 테이블에서 일정한 학습 기간(learning period)을 갖는다.

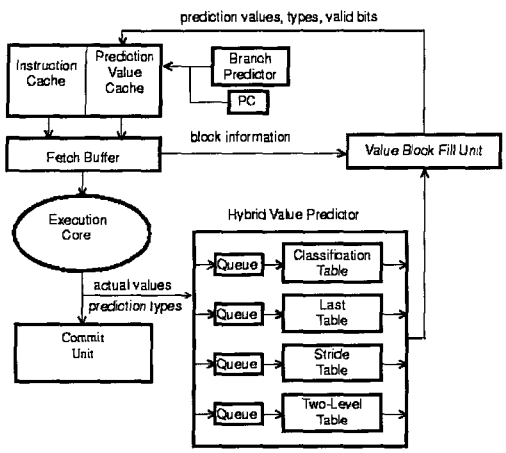
CT(Classified instruction Table)는 명령 주소에 의해 인덱스 되고 명령에 대해 분류된 예측 정보가 PID(Prediction IDentification)에 저장된다. PID는 "Don't Predict", "Use Popular Last Value", "Use Stride+", "Use FCM" 중 하나의 값을 갖는다.

UT(Unclassified instruction Table)는 각 명령어에 대하여 학습 기간 동안 3개의 최근 결과값을 저장하며, 각 명령에 대한 예측기를 선택하기 위하여 사용된다. UT에 있는 결과값들 사이의 모든 차(difference)가 0이면 Popular Last Value, 모든 차가 같으면 Stride+, 이외의 경우에는 FCM으로 지정된다. 선택된 예측기로 예측할 때 신뢰성 카운터 값이 0이 되면 해당 예측기가 잘못 선택된 것으로 간주하여 해당 예측기로부터 퇴거(evict)하고 분류 과정을 다시 거치게 된다. 만약 PID가 FCM이라면 "Don't Predict"로 설정되며, 예측을 시도하지 않는다.

6. Lee의 혼합형 결과값 예측기

프로세서의 인출(fetch)과 이슈 폭(issue width)이 증가함에 따라 많은 명령어들이 실제 결과값으로 예측 테이블을 변경하기 전에 테이블에 다시 접근하는 경우가 많이 발생하게 된다. 이런 경우 부적절한 데이터 값을 사용하게 되어 부정확한 값을 예측하게 된다. 이런 잘못된 예측은 잘못예측에 의한 페널티를 증가시키며, 프로세서의 성능을 저해하는 요인이 된다. Lee 등은 이런 문제점을 해결하기 위해 명령 캐시에 PVC(Prediction Value Cache)를 결합한 동적 분류를 갖는 혼합형 예측기를 제안하였다.

명령 캐시 블록에서의 각 명령은 PVC 블록에 대응되는 엔트리를 가지며, PVC의 각 엔트리는 명령 결과에 대해 예측 값, 예측 형태, 타당 비트(valid bit)로 구성된다. 각 명령에 대한 예측 값은 명령 캐시로부터 명령이 인출되는 것과 같은 방법으로 인출된다. 예측 테이블 조사 후에 VFU(Value block Fill Unit)를 통해 대응되는



〈그림 13〉 Lee의 혼합형 결과값 예측기

PVC로 보내는 예측 값을 생성하며, 명령의 결과값으로 WB 단계에 예측 테이블을 변경한다. Lee의 예측기는 부적절한 예측 값을 식별하기 위해 age 카운터를 사용한다. age 카운터는 예측 값이 변경되기 전에 예측 값에 몇 번이나 접근했는 지를 지시하는 카운터로, 스트라이 분류 형태에 대해 예측 값은 "age 카운터 * 스트라이드 값 + PVC 예측값"으로 예측한다.

Rychlik의 예측기와는 다르게 명령의 분류 예측 형태를 다른 테이블을 사용하지 않고 PVC에 저장하며 오직 하나의 값과 하나의 스트라이드 값만을 저장하나, 동적 분류를 위해 별도의 분류 테이블을 사용하고 있다. 또한, 정확한 예측을 위해 예측 테이블을 모험적으로 갱신하고자 하였으나, 일부 스트라이드 형태에 대해서만 모험적 갱신을 수행하고 있다.

Ⅳ. 결 론

임베디드 병렬구조에서 비순서적 수행과 모험적 수행은 성능을 향상시킬 수 있는 반면에 전력 사용을 크게 증가시킨다. 비순서적 수행과 모험적 수행을 허용하는 프로세서는 병렬로 수행할 수 있는 명령어를 동적으로 결정하는 이슈로직을 사용하는데 이러한 하드웨어는 명령을 실행하기 위해 많은 전력을 소모하는데, 프로세서의 성능을 극대화시키려면 불필요한 명령어들의 수행을 방지하여 전력 낭비를 감소시켜야한다.

프로세서의 성능을 향상시키고 전력 소모를 줄이기 위한 메커니즘으로 분기명령어가 반입될 때, 분기명령어의 결과를 미리 예측하여 분기결과가 결정될 때까지 파이프라인의 지연을 제거시키는 여러 분기예측 기법을 알아보았다.

또한 선행 명령어의 결과값을 후행 명령어가 입력으로 사용하는 실 데이터 종속관계를 제거하여 명령어의 병렬 수행을 최대화시키는 여러 결과값 예측기법을 알아보았다.

최근 임베디드 프로세서의 성능이 PC용 프로세서와 노트북용 프로세서의 성능에 근접해 가는 추세로 볼 때 하드웨어를 사용하여 동적으로 예측하는 분기 예측기법과 결과값 예측기도 임베디드 프로세서에 내장시키므로 서 프로세서 성능 향상과 전력소모 감소에 이점을 주리라 기대된다.

참고문헌

[1] E. S. Robert, C. M. Alsup and Y. N. Patt, "The Agree Predictor : A Mechanism for Reducing Negative Branch History Interference" In Proc. Computer Architecture News, Vol. 25, No.2, pp284-291, May 1997.

[2] C. C. Lee, I. C. K. Chen, T. N. Mudge, "The bi-mode branch predictor". In Proc. 25th Int'l Symposium on Microarchitecture, p4-13, 1997.

[3] Daniel A. Jimémez, "Reconsidering Complex Branch Predictors," In Proc. The 9th International Symposium on High Performance Computer Architecture, 2003.

[4] D. A. Jimenez, C. Lin. "Dynamic branch prediction with perceptrons," In Proc. 7th Int'l Symposium on High Performance Computer Architecture, p197-206, 2001.

[5] T.Y Yeh and Y. N. Patt, " Alternative Implementations of Two-Level Adaptive Branch Prediction", ISCA-19, pp.124-134, 1992.

[6] K. Wang and M. Franklin, "Highly Accurate Data Value Prediction using Hybrid Predictors", MICRO-30, pp.281-290, December 1997.

[7] T. Nakra, R. Gupta and M.L. Soffa, "Global Context-based Value Prediction", HPCA-5, January 1999.

[8] B. Rychlik, J.W. Faistl, B.P. Krug, A.Y. Kurland, J.J. Sung, M.N. Velev, and J.P. Shen, "Efficient and Accurate Value Prediction Using Dynamic Classification", Technical Report of Microarchitecture Research Team in Dept. of Electrical and Computer Engineering, Carnegie Mellon Univ., 1998.

[9] S.J. Lee, P.C. Yew, "On Some Implementation Issue for Value Prediction on Wide-Issue ILP Processors", Intl. Conference on Parallel Architecures and Computer Technique, 2000.

[10] R. Thomas, M. Franklin, C. Wilkerson, J. Stark "Improving Branch Prediction by Dynamic Dataflow-based Identification of Correlated Branches from a Large Global History", In Proc. 30th ISCA, May, 2003.

저자소개

조 영 일

1980년 한양대학교 전자공학과 졸업
1982년 한양대학교 전자공학과 대학원 졸업(공학석사)
1985년 한양대학교 전자공학과 대학원 졸업(공학박사)
1986년 - 현재 수원대학교 정보공학대학 조교수, 부교수, 교수
2002년 - 현재 ITS 학회 이사 및 편집위원
2004년 - 현재 대한전자공학회 학회지 편집위원
주관심분야 임베디드 병렬 구조 , 이동 통신, 최적화 컴파일러