논문 2007-44CI-1-5

# 적응 가능한 분기 히스토리 길이를 사용하는 분기 예측 메커니즘

## ( A Branch Prediction Mechanism Using Adaptive Branch History Length )

조 영 일*

( Young-Il Cho )

요 약

최근, 프로세서의 파이프라인 깊이와 이슈 폭이 점차로 증가함에 따라 분기예측 실패에 의한 페널티가 더욱 증가하고 있다. 분기예측 실패는 프로세서 성능을 개선하는데 가장 심각한 성능 장애 요소이다. 따라서 좀 더 정확한 분기 예측기는 최신 프로세서들에게 필수적이다.

많은 분기예측기들은 분기 명령의 주소와 고정 분기히스토리 길이로 예측을 수행한다. 최적의 분기히스토리 길이는 프로그램과 프로그램에 있는 분기 명령에 따라 달라지므로 고정 분기히스토리를 사용하는 예측기들은 잠재적 성능을 얻을 수 없다.

본 논문에서는 5개 뱅크로부터의 예측 중 가장 높은 예측정확도를 갖는 뱅크로 예측하는 가변 길이 분기 히스토리를 사용하는 분기예측 메커니즘을 제안한다. 뱅크 0는 분기 명령의 주소만을 사용하여 인덱스 하는 bimodal 예측기이고, 나머지 뱅크는 다른 히스토리 길이와 분기 명령 PC로 인덱스 하는 예측기이다.

실험결과 제안한 메커니즘은 12, 13의 고정 히스토리 길이를 사용하는 gshare보다 최대 6.34% 예측 정확도를 개선시켰고, 각 벤치마크에 대한 최적의 히스토리 길이를 사용하는 gshare와 비교해도 최대 2.3% 개선시켰다.

Abstract

Processor pipelines have been growing deeper and issue widths wider over the years. If this trend continues, the branch misprediction penalty will become very high. Branch misprediction is the single most significant performance limiter for improving processor performance using deeper pipelining. Therefore, more accurate branch predictor becomes an essential part of modern processors.

Several branch predictors combine a part of the branch address with a fixed amount of global branch history to make a prediction. These predictors cannot perform uniformly well across all programs because the best amount of branch history to be used depends on the program and branches in the program. Therefore, predictors that use a fixed history length are unable to perform up to their potential performance.

In this paper, we propose a branch prediction mechanism, using variable length history, which predicts using a bank having higher prediction accuracy among predictions from five banks. Bank 0 is a bimodal predictor which is indexed with the 12 least significant bits of the branch address. Banks 1, 2, 3 and 4 are predictors which are indexed with different global history bits and the branch PC.

In simulation results, the proposed mechanism outperforms gshare predictors using fixed history length of 12 and 13 , up to 6.34% in prediction accuracy. Furthermore, the proposed mechanism outperforms gshare predictors using best history lengths for benchmarks, up to 2.3% in prediction accuracy.

**Keywords :** misprediction penalty, branch prediction, prediction accuracy, variable length history

## I. 서 론

현재, 마이크로프로세서는 파이프라인 스테이지 수가

* 평생회원, 수원대학교 컴퓨터학과
(Dept. of Computer Science, University of Suwon)
접수일자: 2006년9월28일, 수정완료일: 2007년1월11일

증가하고 이슈 폭이 증가하는 추세이므로 정확한 분기 예측은 프로세서의 성능 향상을 위해 중요하다[1]. 따라서 예측정확도가 높은 동적으로 분기결과를 예측하는 방법에 대한 연구가 계속되고 있다. 동적 분기예측 방법은 실행시간동안 얻어지는 정보를 이용하여 예측기를 학습시켜 예측정확도를 향상시키는 방법이다. 동적 분기예측방법 중 2-단계 적응 분기예측기(two-level adaptive branch predictor)[2]가 특히 효과적임이 증명되었다. 이는 선행 분기명령들의 분기결과를 현재 예측하려는 분기명령과 상호연관(correlation)시켜 예측하기 때문이다.

2단계 예측기는 일반적으로 고정된 길이의 상호연관(즉 고정된 분기 히스토리 길이)을 사용하는데 분기명령마다 상호연관의 정도가 다르다. 일반적으로는 긴 분기히스토리를 사용할 때 예측정확도가 향상되나 항상 그렇지는 않다. 긴 분기 히스토리를 사용하면 학습시간도 오래 걸리고, 한 분기명령이 PHT(Pattern History Table)의 여러 엔트리로 맵핑될 수 있으므로 다른 분기명령과 더 많은 충돌이 발생하므로 간섭에 의한 분기예측 실패가 증가할 수도 있다. Chang 등은 한 방향으로 편향되는 분기명령은 분기 히스토리 길이를 증가시킬 때 오히려 분기예측 정확도가 감소한다고 하였다[3].

분기명령마다 최적의 상호연관 분기 히스토리 길이를 구하는 기존의 방법들이 제안되었으나 프로파일링(profiling)을 통해 정적으로 구하는 방법들은 프로파일링에 사용하는 데이터와 실행시간에 사용하는 입력 데이터의 차이로 인한 문제가 발생하므로 실행시간에 동적으로 최적의 분기 히스토리 길이를 구하는 방법들이 연구되고 있다. 그러나 기존의 동적 방법들은 복잡한 구조로 인해 예측시간이 지연되어 명령어 반입 단계(fetch stage) 내에 예측을 완료할 수 없다는 문제점을 갖고 있다.

본 논문에서는 분기명령을 다른 길이의 분기 히스토리로 예측하는 여러 뱅크 중 가장 예측정확도가 높은 뱅크로 부터의 예측결과로 최종 예측하는 분기 예측기를 제안한다. 즉 실행시간에 동적으로 각 분기명령을 예측하기 위해 사용할 최적의 분기 히스토리 길이를 구하고 이를 사용하여 해당 분기를 예측하므로 서 예측정확도를 개선시키면서 명령어 반입 단계 내에 지연 없이 예측한다.

Ⅱ장에서는 이론적 배경에 대해 기술하고, Ⅲ장에서는 분기 히스토리 길이에 따라 예측정확도의 변화를 분석한 결과를 기술하고, Ⅳ장에서는 본 논문에서 제안한 가변 길이 히스토리 예측기에 대해 기술한다. Ⅴ장에서는 실험 및 성능 평가를 기술하고, 마지막으로 결론을 기술한다.

## Ⅱ. 이론적 배경

프로그램에서 어떤 분기명령은 지역 분기 히스토리(local branch history)를 사용할 때 예측정확도가 높아지고 어떤 분기명령은 전역 분기 히스토리(global branch history)를 사용할 때 예측정확도가 높아진다는 분석 결과를 이용하여 지역 분기 히스토리와 전역 분기 히스토리를 선택적으로 사용하여 예측하는 하이브리드 분기예측기[4]와 지역 히스토리와 전역 히스토리를 결합하여 예측하는 결합형 예측기[5]가 연구되었다.

한편, Nair는 동적 수행경로를 기반으로 하는 예측기를 제안하였다[6]. 이 예측기는 분기 히스토리 레지스터에 선행 분기 명령들의 분기 결과를 저장하는 대신에 분기 주소의 일부를 저장하는 데 각 분기 명령에 대한 정보를 위해 여러 비트의 저장을 요구하지만 분기결과를 저장하는 방법보다 좀 더 강한 상호연관을 가지며 짧은 학습시간을 요구하는 장점을 갖는다.

현재 프로세서에서 사용되는 대부분의 분기 예측기는 2단계 분기예측기를 기초로 하고 있다. McFarling이 제안한 gshare 예측기[7]는 분기명령의 주소와 분기 히스토리 레지스터(Branch History Register : BHR)를 XOR하여 PHT를 인덱스 하므로 서 간섭을 감소시켜 예측 정확도를 향상시키는 방법이다. gshare 예측기는 높은 예측 정확도와 빠른 예측을 수행하므로 현재 많은 프로세서에서 사용되고 있다.

Tarlescu 등은 분기명령마다 다른 분기 히스토리 길이를 할당하는 방법을 제안하였다[8]. 이 방법은 프로그램의 수행 전에 프로파일링을 통해 각 분기 명령어의 최적의 히스토리 길이를 찾는 정적 방법이다. 프로파일링에 사용된 입력 데이터와 실행시간에 사용되는 입력 데이터가 다를 수 있기 때문에 프로파일링을 통해 구한 최적의 분기 히스토리 길이가 실제 실행 시간에는 올바르지 않을 수 있다.

히스토리 길이를 동적으로 조절하는 방법으로 Juan 등에 의해 제안된 DHLF(Dynamic History Length Fitting) 방법이 있다[9]. 이 방법은 프로그램 수행을 다수의 인터벌로 구분하여, 각 인터벌마다 분기 예측의 정확도 변화를 조사하여 그 결과에 따라 다음 인터벌까지 사용될 분기 히스토리 길이를 결정한다. 히스토리

길이를 변경하면 새로운 변경에 적응하기 위한 학습시간이 필요하고 이 시간동안 많은 분기예측 실패를 유발할 수 있다. 이런 문제를 해결하기 위해 학습시간에 의한 예측 실패를 최소화하도록 히스토리 길이 변경을 최소화하는 방법을 제안하고 이를 위해 프로파일링 정보를 이용하는 방법을 제안하였다[10].

한편, 복잡한 형태의 대규모 다중 분기 예측기(Hybrid Branch Predictor)가 제안되었다. 다중 분기 예측기의 다양한 구성 요소(Sub-Predictor)들에서 사용되는 입력 히스토리 길이를 각 예측기 별로 서로 다르게 구성하여 그 결과를 선택적으로 사용하는 방법[11]이 제시되었으나 많은 하드웨어 자원의 활용이 가능한 다중 분기 예측기에서 가능한 방식이다.

Michaud는 다른 히스토리 길이를 갖는 뱅크들로 구성된 예측기를 제안하였다[12]. 예측하려는 분기명령이 할당된 뱅크 중에서 히스토리 길이가 가장 긴 뱅크의 결과로 예측을 한다. 이 예측기는 긴 분기히스토리를 사용할 때 예측정확도가 높다는 일반적인 가정을 도입하였고 여러 뱅크를 동시에 액세스할 수는 있으나 히스토리 길이가 긴 순서로 순차적으로 분기명령에 대해 엔트리가 할당되었는지를 조사하므로 예측시간이 지연되기 때문에 빠른 예측기로는 사용할 수 없다.

이상에서와 같이, 기존의 방법들은 대부분 히스토리 길이를 조정하기 위한 사전 프로파일링 단계를 필요로 하거나, 예측 시간이 길거나, 수행 중간에 히스토리 길이 변환을 위한 인터벌을 필요로 한다. 이러한 방식은 실제 프로그램의 수행 시에 동적으로 히스토리 길이를 조절하지 못하기 때문에, 현실적인 구현 가능성과는 거리가 멀다.

perceptron 예측기[13]는 분기 히스토리에 있는 이전 분기명령들의 분기결과와 현재 예측하려는 분기명령사이의 상호연관을 가중치로 표시하고, 가중치가 큰 분기명령이 예측하려는 분기명령의 예측에 좀 더 기여하도록 하는 예측 방법이다. 그러나 perceptron 예측기는 가중치를 계산하는 오버헤드 때문에 예측을 지연시키는 문제점을 갖는다.

예측정확도를 높이기 위해 큰 예측기를 사용하면 한 사이클 내에 예측을 할 수 없기 때문에 다음 사이클에 반입될 명령어의 주소를 알 수가 없다. 이를 위해 Jimenez 등이 제안한 overriding 예측기는 두 번의 예측을 제공한다[14]. 첫 번째 예측은 한 사이클 내에 예측하나 예측정확도가 낮은 예측기가 제공하고, 두 번째 예측은 여러 사이클이 걸리나 예측정확도가 높은 예측기가 제공한다. 첫 번째 예측을 사용하여 진행되다 두 번째 예측이 구해지면 첫 번째 예측과 비교하여 일치하면 계속 진행하고 일치하지 않으면 첫 번째 예측에 의해 수행된 경로의 명령어들을 무효화시키는 recovery를 수행한다.

## III. 분기 히스토리 길이에 따른 예측정확도 분석

그림 1은 0~16 사이의 고정 길이 분기 히스토리를 사용하는 gshare 예측기에서 SPECint95 벤치마크에 대한 예측 정확도를 나타낸다.

벤치마크 compress, li, perl, m88ksim, ijpeg는 히스토리 길이를 증가시키면 예상 정확도가 증가하는데 이는 이들 벤치마크 프로그램들은 비교적 적은 정적 분기명령(static branch)을 가지므로 히스토리 길이를 증가시켜도 PHT에서 충돌(conflict)과 분기예측 테이블 미스가 적게 발생하기 때문이라 추측된다. 한편, 벤치마크 gcc, go, vortex는 히스토리 길이를 증가시키면 오히려 예상정확도가 감소하는 것으로 나타났다. 이들 벤치마크들은 비교적 많은 정적 분기명령을 가지므로 히스토리 길이를 증가시키면 PHT에서 더 많은 충돌과 더 많은 분기예측 테이블 미스가 발생하기 때문이라 추측된다.

그림 1에서 보는바와 같이 벤치마크 프로그램마다 다른 히스토리 길이에서 가장 높은 예상정확도를 가지며 이는 각 벤치마크 내에 있는 분기명령들도 다른 히스토리 길이에서 최적의 예측 정확도를 갖는 것으로도 추측할 수 있다. 따라서 수행 중 동적으로 각 분기명령마다 가장 예측정확도가 높은 분기 히스토리 길이로 적

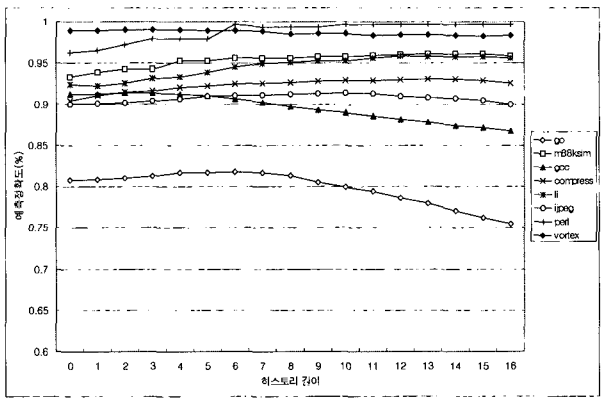



그림 1. 고정길이 분기 히스토리를 갖는 gshare 예측기의 예측정확도

Fig. 1. Prediction accuracy of fixed history length gshare.

응시킬 수 있는 예측기의 필요성이 요구된다.

그림 1에서 벤치마크들은 히스토리 길이가 6~12사이에서 높은 예상정확도를 갖는다는 분석을 바탕으로 본 논문에서는 각 분기명령마다 6~12 사이의 히스토리 길이 중 가장 높은 예상정확도를 갖는 최적의 길이로 적응하는 분기 예측기를 제안하고자 한다.

## Ⅳ. 제안한 가변 길이 히스토리 예측기

일반적으로는 분기명령은 긴 히스토리 길이를 사용할 때 예상정확도가 향상되지만 항상 그렇지만은 않다. 어떤 분기는 작은 히스토리 길이를 사용할 때 예측 정확도가 증가할 수 있다. 따라서 각 분기 명령마다 최적의 히스토리 길이를 사용할 때 가장 예측 정확도가 높아진다.

본 논문에서는 각 분기 명령에 대한 최적의 히스토리 길이를 동적으로 구하여 예측하는 분기예측방법을 제안한다. 제안한 동적으로 최적의 분기 히스토리 길이를 구하여 예측하는 분기예측 메커니즘은 그림 2와 같다.

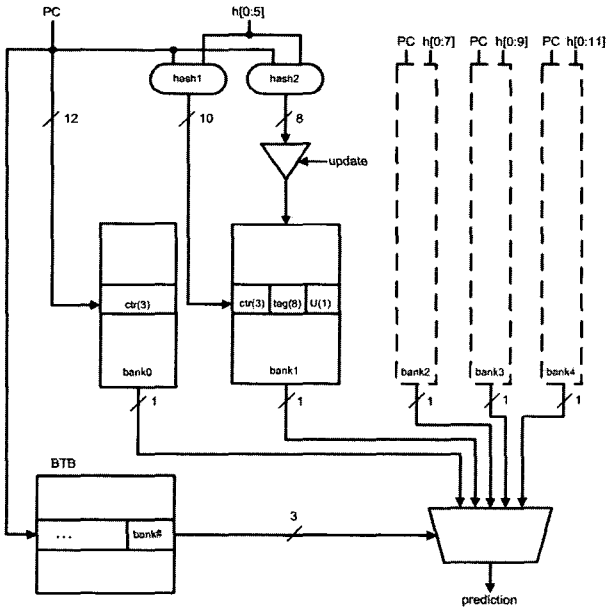



그림 2. 최적의 분기 히스토리 길이로 예측하는 제안한 분기예측기 구조

Fig. 2. The proposed branch predictor which predicts using optimal branch history length.

### 1. 예측기 테이블 구조

예측기 테이블은 5개의 뱅크로 구성된다. 뱅크 0은 bimodal 예측기로 분기 히스토리를 사용하지 않고 분기명령의 주소(program counter : PC)로만 인덱스 한다. 뱅크 0은 뱅크 1~4에 예측하려는 분기 명령에 대한 엔트리가 없을 경우에도 사용된다. 뱅크 0의 엔트리는 3비트 포화카운터로만 구성된다. 뱅크 1~4는 PC와 분기 히스토리 길이(뱅크 1 : 6, 뱅크 2 : 8, 뱅크 3 : 10, 뱅크 4 : 12)를 사용하여 인덱스 한다. 뱅크 1~4의 엔트리는 1비트의 U필드, 8비트의 태그필드, 3비트의 포화 카운터필드를 갖는다. U필드는 해당 엔트리가 임의의 분기명령에 대해 유효한 엔트리인지를 나타낸다. 태그필드는 PC와 분기 히스토리를 해시시킨 결과를 저장하여 예측하려는 분기명령이 해당 뱅크에 엔트리가 할당이 되었는지를 검사하는데 사용된다. 포화카운터는 분기 방향을 예측하는데 사용한다. 임계치 이상이면 taken으로 그렇지 아니면 not-taken으로 예측한다.

### 2. BTB 구조

제안한 분기예측기는 예측할 때 BTB(Branch Target Buffer)에 있는 정보를 사용한다. 이를 위해 BTB 엔트리에 bank# 필드를 추가하였다. bank# 필드는 예측 테이블 및 BTB를 갱신할 때 태그가 일치된 뱅크들의 엔트리들 중에 포화카운터가 0 혹은 7에 가장 근접한 뱅크 번호를 저장한다. 0에 근접하는 경우는 해당 분기명령이 not-taken일 확률이 높은 것을 의미하고, 7에 근접할 때는 taken일 확률이 높은 것을 의미한다. 여러 뱅크의 포화카운터가 동일 크기로 근접할 때는 히스토리 길이가 큰 뱅크 번호를 저장한다(일반적으로 긴 히스토리 길이를 사용할 경우 예측 정확도가 높다). 즉 bank# 필드에는 분기명령에 대해 다음 예측할 때 가장 예측 정확도가 높은 히스토리 길이를 갖는 뱅크 번호를 저장한다. 따라서 분기명령을 예측할 때 bank# 필드에 있는 값을 사용하여 5개 뱅크의 예측 결과 중 최종 예측을 선택한다. bank# 필드 값이 i이면 뱅크 i의 예측을 최종 예측으로 선택한다.

### 3. 예측 메커니즘

반입된 분기명령을 예측하기 위해 5개의 뱅크를 동시에 액세스 한다. 뱅크 0은 PC의 하위 12비트를 사용하여 인덱스하고, 뱅크 1은 PC와 최근 6비트 분기 히스토리를 사용하여 인덱스하고, 뱅크 2는 PC와 최근 8비트 분기 히스토리를 사용하여 인덱스하고, 뱅크 3은 PC와 최근 10비트 분기 히스토리를 사용하여 인덱스하고 뱅크 4는 PC와 최근 12비트 분기 히스토리를 사용하여 인덱스 한다. 각 뱅크에서는 인덱스된 엔트리의 태그필드와 현재 예측하려는 분기명령에 대해 계산된 태그

값이 일치하면 포화카운터 값에 따라 예상한다. 포화카운터 값이 4보다 크거나 같으면 taken으로 예측하고, 아니면 not taken으로 예측한다. 현재 예측하려는 분기명령의 최종 예측은 각 뱅크의 예측 결과 중 BTB의 bank# 필드에 저장된 뱅크 번호를 사용하여 선택한다.

다음은 예측 메커니즘의 절차이다.

(단계 1) PC와 6~12 비트의 분기히스토리로 모든 뱅크와 BTB를 동시에 액세스한다.

(a) 뱅크 0는 PC로만 액세스

(b) 뱅크 1~4는 PC와 6~12 비트의 분기히스토리로 액세스

(c) BTB는 PC로만 액세스

(단계 2) 각 뱅크에서 선택된 엔트리의 tag 필드와 hash2로 구한 tag가 같으면 포화 카운터인 ctr 필드로 예측한다.

(a) ctr 값이 4이상이면 taken으로 예측

(b) ctr 값이 3이하면 not-taken으로 예측

(단계 3) 단계 2에서 구한 각 뱅크의 예측 결과 중에서 단계 1에서 선택된 BTB 엔트리의 bank# 필드로 최종 예측한다.

### 4. 예측기 갱신

예측을 한 후 예측결과와 실제 결과를 비교하여 예측기 테이블과 BTB를 수정한다.

#### (1) 예측이 맞았을 경우

예측을 수행한 뱅크 번호가 i일 경우, 뱅크 i의 해당 엔트리의 포화 카운터만 수정한다. 실제 결과가 taken이면 포화카운터를 '1' 증가시키고, not taken이면 '1' 감소시킨다. 뱅크번호 i가 0이 아닐 경우에는 해당 엔트리의 U 필드를 세트시켜 그 엔트리가 현 분기명령을 예측하는데 유용하다는 것을 표시한다. 즉, 해당 엔트리는 현 분기명령을 예측하는데 적합한 히스토리 길이이므로 계속 사용하도록 한다. 이때 BTB의 대응 엔트리는 예측이 맞았으므로 bank# 필드를 수정할 필요가 없다. 또한 다른 뱅크들의 엔트리를 수정하지 않으므로 서 다른 분기명령을 예측할 때 올바른 정보를 계속 유지하도록 한다.

#### (2) 예측이 틀렸을 경우

예측을 수행한 뱅크 번호가 i라 가정하자. 먼저 뱅크 i의 엔트리와 태그 매치된 엔트리의 포화 카운터를 모두 수정한다. 실제 결과가 taken이면 포화카운터를 '1' 증가시키고, not taken이면 '1' 감소시킨다. 뱅크번호 i가 0이 아닐 경우에는 뱅크 i의 해당 엔트리의 U 필드를 리셋시켜 그 엔트리가 현 분기명령을 예측하는데 유용하지 않다는 것을 표시한다. 즉, 해당 엔트리는 현 분기명령을 예측하는데 적합한 히스토리 길이가 아니므로 다른 분기명령에 할당하도록 한다.

태그 매치된 엔트리가 없을 경우 즉, 현재 반입되어 예측을 하고 있는 분기명령은 뱅크에 엔트리가 할당되지 않은 경우이므로 뱅크 1~4에 새로운 엔트리를 할당한다. 0보다 큰 뱅크 번호 중 i를 제외한 나머지 뱅크들의 엔트리들 중 U 필드가 리셋인 엔트리들에 현 분기명령을 할당한다. 태그 필드에 계산된 태그를 저장하고, U 필드는 리셋으로 초기화하고 포화카운터는 실제 결과가 taken이면 4로 not taken이면 3으로 초기화한다. 만약 리셋인 엔트리가 없을 경우에는 포화카운트가 3이나 4에 가장 근접한 값을 갖는 엔트리를 임의로 선택하여 할당한다.

예측한 분기명령에 할당된 BTB 엔트리의 bank# 필드를 수정한다. 실제 결과가 taken일 경우에는 태그 매치된 뱅크들과 뱅크 0 중에서 포화카운터가 7에 가장 가까운 뱅크 번호로 설정하고, not taken일 경우에는 포화카운터가 0에 가장 가까운 뱅크 번호로 설정한다. 만약 포화카운터가 동일한 여러 뱅크가 있을 경우 뱅크 번호가 큰 값으로 설정한다. 이는 일반적으로 히스토리 길이가 길 때 예측정확도가 높기 때문이다.

## V. 실험 및 성능 평가

본 논문에서 제안한 분기예측기의 예측정확도와 성능 향상을 평가하기 위해 gshare 예측기와 비교한다. gshare는 반입 단계 이내에 분기 명령의 분기 방향을 예측 완료시키고 예측정확도가 높은 대표적인 빠른 예측기이다. 예측정확도가 높더라도 많은 분기명령을 커버하지 못하면 성능이 감소할 수도 있다. 따라서 이 장에서는 예측기의 예측정확도와 사이클 당 수행되는 명령어 수(Instructions Per Cycle : IPC)로 나타내는 성능을 측정하여 제안한 분기예측기의 타당성을 확인한다.

성능 측정을 위해 슈퍼스칼라 프로세서의 사이클 수준 시뮬레이터인 SimpleScalar 3.0/PISA 툴셋[15]에서 본 논문에서 제안한 동적으로 히스토리 길이를 적응시키는 예측기를 구현하였다.

표 1은 시뮬레이션된 프로세서의 구성에 대한 머신 파라메타를 나타낸다. 8 이슈 프로세서를 기반으로 하

표 1. 시뮬레이션에 사용된 프로세서 파라미터
Table 1. processor parameters of simulation.

| 구 분 | 인 수 | 값 | 기 타 |
|---|---|---|---|
| Processor Core | RUU size | 128 entries | Instruction window |
| | LSQ size | 64 entries | load/store queue |
| | Fetch width | 8 instructions/cycle | in order |
| | Decode width | 8 instructions/cycle | in order |
| | Issue width | 8 instructions/cycle | out of order |
| | Commit width | 8 instructions/cycle | in order |
| | Functional units | 8 i-ALU(1),<br>8 f-ALU(2)<br>2 i-MULT/DIV(3/12)<br>2f-MULT/DIV(4/12) | ( ) is latency |
| Memory | Memory ports | 2 ports, 8byte bus | |
| | Memory access latency | first_chunk(18),<br>inter_chunk(2) | ( ) is latency |
| Branch predictor | Return Address Stack | 8 entries | |
| | BTB | 1024 entries, 4 way | |
| Cache | L1 data cache | 128K, 32B block,<br>2 way, LRU,<br>1 cycle latency | |
| | L1 instruction cache | 128K, 32B block,<br>2 way, LRU,<br>1 cycle latency | |
| | L2 unified cache | 1M, 64B block,<br>4 way, LRU,<br>4 cycle latency | |

표 2. 벤치마크 프로그램 요약
Table 2. Benchmarks summary.

| 벤치마크 | 입력 데이터 | 수행된 명령어 수(M) | 동적 조건 분기명령수(M) | 정적 조건 분기명령 수 | 비 고 |
|---|---|---|---|---|---|
| gcc | cccp.i | 200 | 17.2 | 4622 | The GNU C compiler version 2.5.3. |
| li | train.lsp | 183 | 32.6 | 1105 | Xlisp interpreter. |
| vortex | persons.250 | 200 | 27.4 | 14419 | An object oriented database. |
| go | 50 9 2stone9.in | 200 | 11.7 | 301 | An internationally ranked go playing program. |
| m88ksim | dcrand.lit | 200 | 32.4 | 774 | A chip simulator for the Motorola 88100 |
| compress | 100000 e 2231 | 200 | 13.2 | 1268 | An in-memory version of the common UNIX utility. |
| perl | primes.pl | 200 | 21.4 | 936 | An interpreter for the Perl language. |
| ijpeg | penguin.ppm | 200 | 25.8 | 6627 | Image compression/ decompression on in memory images. |

였고, 모든 분기예측기의 BTB는 1K 엔트리 4-way로 고정하였다. 128KB L1 데이터 캐시, 128KB L1 명령어 캐시, 1MB L2 통합 캐시를 사용하였다.

표 2는 시뮬레이션에서 사용된 SPEC 벤치마크 프로그램[16], 입력 데이터, 동적으로 수행된 조건 분기 명령어 수와 정적 조건 분기 명령어 수를 나타낸다. 입력 데이터는 ref input을 사용하였고, 시뮬레이션 시간을 줄이기 위해 벤치마크 프로그램의 수행된 명령어수를 200M(million)까지로 제한하였다.

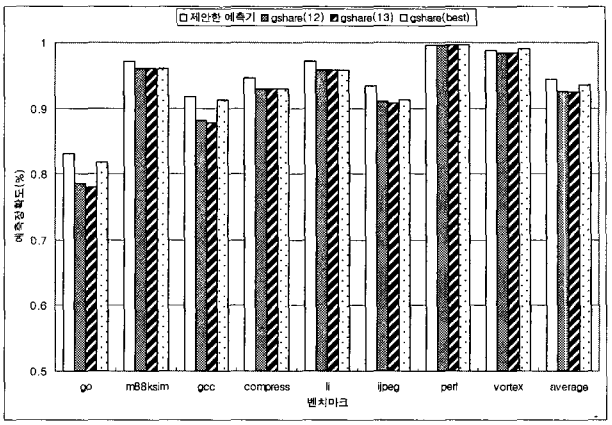



그림 3. 제안한 예측기와 고정 히스토리 길이 gshare의 예측정확도
Fig. 3. Prediction accuracy of proposed predictor and fixed history length gshare.

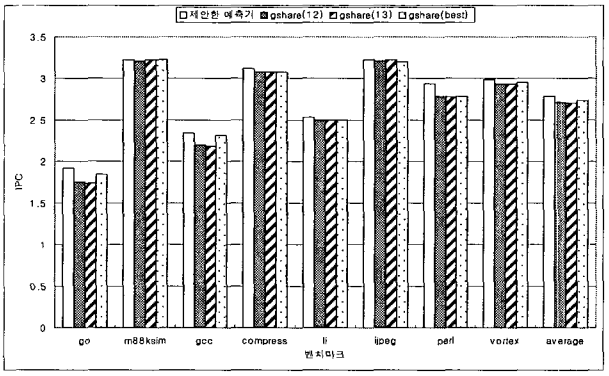



그림 4. 제안한 예측기와 고정 히스토리 길이 gshare의 IPC
Fig. 4. IPC of proposed predictor and fixed history length gshare.

그림 3은 본 논문에서 제안한 예측기, 고정 분기 히스토리 길이 12를 사용한 gshare 예측기(gshare(12)), 고정 분기 히스토리 길이 13을 사용한 gshare 예측기(gshare(13)), 고정 분기 히스토리 길이가 0에서 16사이에서 예측정확도가 가장 높은 경우(gshare(best))의 예측정확도를 비교한 결과이다. 제안한 예측기를 gshare(12) 예측기와 비교하면 최대 5.75%(go), 평균 1.94%의 예측정확도가 개선되었다. gshare(13) 예측기와 비교하면 최대 6.34%(go), 평균 2.05%의 예측정확도가 개선되었다. 또한 gshare(best)와 비교했을 때도 최대 2.3%(go), 평균 0.96%의 예측정확도가 개선되었다.

그림 4는 본 논문에서 제안한 예측기, gshare(12), gshare(13), gshare(best)의 IPC를 비교한 결과이다. 제안한 예측기를 gshare(12)와 비교하면 최대 9.4%(go), 평균 2.77%의 IPC가 개선되었다. gshare(13)과 비교하면 최대 10.66%(go), 평균 2.84%의 IPC가 개선되었다.

또한 gshare(best)와 비교했을 때도 최대 5.12%(perl), 평균 1.72%의 IPC가 개선되었다.

본 논문에서 제안한 예측기의 하드웨어 비용은 뱅크 0은 3비트 포화 카운터 4K 엔트리로 구성되며 뱅크 1~4는 12비트(3비트 포화카운터+1비트 U비트+8비트 태그) 1K 엔트리로 구성되므로 총 60K 비트이다. 고정 히스토리 길이를 사용하는 gshare와 비교하면 14 고정 히스토리 길이 이하보다는 많은 하드웨어 비용을 요구하지만 15 고정 히스토리 길이 이상보다는 적은 하드웨어 비용을 요구한다. 그런데 제안한 예측기는 5개의 뱅크가 독립적으로 동시에 액세스되기 때문에 gshare에 비해 작은 액세스 시간을 요구하므로 예측 지연을 발생시키지 않는다.

## VI. 결 론

분기명령의 주소와 선행 분기명령의 결과인 분기 히스토리를 결합하여 예측하는 방법들은 사용되는 분기 히스토리 길이가 예측 정확도에 영향을 준다. 따라서 예측하려는 분기 명령을 위한 최적의 히스토리 길이를 실행시간에 동적으로 구하는 방법은 매우 중요하다.

본 논문에서는 다른 길이의 분기 히스토리로 예측하는 여러 뱅크 중 가장 예측정확도가 높은 뱅크로 부터의 예측결과로 최종 예측하는 분기 예측기를 제안하였다. 실행시간에 동적으로 각 분기명령을 예측하는데 사용할 최적의 분기 히스토리 길이를 구하고 이를 사용하여 해당 분기를 예측하므로 서 예측 정확도를 개선시키면서 명령어 반입 단계 내에 지연 없이 예측할 수 있다.

제안한 예측기는 고정 히스토리 길이 12, 13의 gshare에 비해 각각 평균 1.94%, 2.05%의 예측정확도를 개선시켰고, 히스토리 길이 0부터 16에서 예측정확도가 가장 높은 gshare에 비해서도 평균 0.96%를 개선시켰다. 또한 IPC는 고정 히스토리 길이 12, 13의 gshare에 비해 각각 평균 2.77%, 2.84%를 개선시켰고, 최적의 고정 히스토리 길이를 갖는 gshare에 비해서도 평균 1.72%를 개선시켰다. 또한 고정 히스토리 길이 0에서 16사이에서 예측정확도가 가장 높은 gshare에 비해서도 예측정확도와 IPC를 개선시켰다.

## 참 고 문 헌


[1] E. Sprangle and D. Carmean. "Increasing processor performance by implementing deeper pipelines" In Proc. of the 29th ISCA, pp. 25-34, May 2002.

[2] T. Y. Yeh and Y. N. Patt, "Alternative implementations of two-level adaptive branch prediction," In Proc. of the 19th ISCA, pp. 124-134, May 1992.

[3] P. Y. Chang, E. Hao, T. Y. Yeh, and Y. Patt.,"Branch classification: a new mechanism for improving branch predictor performance." In Proc. of the 27th MICRO, pp. 22-31, Nov. 1994.

[4] P. Y. Chang, E. Hao, and Y. N. Patt."Alternative implementations of hybrid branch predictors." In Proc. of 28th MICRO, pp. 252-257, Dec. 1995.

[5] K. Skadron, M. Martonosi and W. Clark, "A Taxonomy of Branch Mispredictions, and Alloyed Prediction as a Robust Solution to Wrong-History Mispredictions", In Proc. of PACT-2000, pp. 196-206, Oct. 2000.

[6] Ravi Nair, "Dynamic path-based branch predictor", In Proc. of 28th MICRO, pp. 15-23, Dec. 1995.

[7] S. McFarling,"Combining branch predictors.", Tech. Rep. TN-36, Digital Western Research Lab., June 1993.

[8] M. D. Tarlescu, K. B. Theobald, and G. R. Gao, "Elastic history buffer: A low-cost method to improve branch prediction accuracy", In Proc. Int'l Conf. on Computer Design, pp. 82-87, 1997.

[9] T. Juan, S. Sanjeevan, and J. J. Navarro, "Dynamic history length fitting: A third level of adaptivity for branch prediction", In Proc. of the 25th ISCA, pp. 155-166, May 1998.

[10] A. Falcon et al.,"Studying New Ways for Improving Adaptive History Length Branch predictors", ISHPC 2002, pp. 271-280, 2002.

[11] A. Seznec, S. Felix, V. Krishnan, and Y. Sazeides. "Design tradeoffs for the ev8 branch predictor", In Proc. of the 29th ISCA, pp. 295-306, May 2002.

[12] Pierre Michaud,"A PPM-like, Tag-based Predictor", JILP Vol.7, pp. 1-10, April 2005.

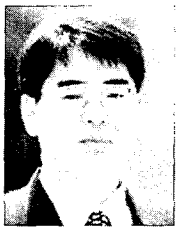
[13] D. A. Jimenez and C. Lin, "Dynamic branch prediction with perceptrons.", In Proc. of the 7th HPCA, pp. 197-206, Feb. 2001.

[14] D. A. Jimenez, "Reconsidering Complex Branch Predictors', In Proc. of the 9th HPCA, pp. 43-52, Feb. 2003.

[15] D. Burger, T. M. Austin, and S. Bennett, "Evaluating future micro-processors: the SimpleScalar tool set", Tech. Report TR-1308, Univ. of Wisconsin-Madison Computer Sciences Dept., 1997.

[16] SPEC Benchmarks, http://www.specbench.org

저 자 소 개



조 영 일(평생회원)
1980년 한양대학교 전자공학과 학사
1982년 한양대학교 전자공학과 석사
1985년 한양대학교 전자공학과 박사
1986년 3월~현재 수원대학교 컴퓨터학과 교수
<주관심분야 : 병렬 구조, 센서 네트워크, 최적화 컴파일러>