분기 히스토리의 모험적 갱신을 허용하는 전역 히스토리 기반 분기예측기에서 분기예측실패를 위한 간단한 복구 메커니즘

(Simple Recovery Mechanism for Branch Misprediction in Global-History-Based Branch Predictors Allowing the Speculative Update of Branch History)

고 광 현 † 조 영 일 ††
(Kwang Hyun Ko) (Young Il Cho)

요 약 조건 분기예측은 프로세서 성능 개선을 위한 중요한 기술이다. 그러나, 분기예측실패는 많은 사이클을 낭비시키며, 비순서적 실행을 방해하고, 잘못 예측된 명령어들을 수행하게 되므로 전력을 낭비한다. 따라서 높은 정확도를 갖는 분기예측기는 좋은 성능을 갖는 프로세서를 위해 중요하다.

gshare와 GAg같은 전역 히스토리를 기반으로 하는 예측기에서는 히스토리의 명령어 완료시간 갱신(commit update)에 의해 많은 분기예측실패가 발생한다. 그런 문제를 해결하기 위해 히스토리를 모험적으로 갱신하고, 분기예측실패 시 히스토리를 복구시키는 메커니즘에 관한 연구의 필요성이 제시되었고, 연구되었다.

본 논문에서는 분기예측실패 발생 후 분기 히스토리를 복구하는 간단한 복구 메커니즘을 제안한다. 제안한 복구 메커니즘은 기존 분기예측기에 age_counter를 추가하고 분기 히스토리 레지스터 크기를 2배로 확장시킨다. age_counter는 미해결 분기명령어 수를 저장하며, 분기예측실패 후 분기 히스토리 레지스터를 복구하는데 사용한다.

SimpleScalar 3.0/PISA 툴셋과 SPECINT95 벤치마크 프로그램에서 시뮬레이션 한 결과, 제안된 복구 메커니즘을 gshare와 GAg 예측기에 적용하였을 때 예측 정확도와 프로세서 성능을 개선시킬 수 있었음을 보여준다. GAg와 gshare 예측기에서 예측정확도는 각각 9.21%와 2.14%가 개선되었고, IPC는 18.08%와 8.75% 개선되었다.

키워드 : 분기예측, 예측실패, 모험적 갱신, 분기 히스토리, 복구 메커니즘, 미해결 분기명령어, 예측정확도

Abstract Conditional branch prediction is an important technique for improving processor performance. Branch mispredictions, however, waste a large number of cycles, inhibit out-of-order execution, and waste electric power on mis-speculated instructions. Hence, the branch predictor with higher accuracy is necessary for good processor performance. In global-history-based predictors like gshare and GAg, many mispredictions come from commit update of the history. Some works on this subject have discussed the need for speculative update of the history and recovery mechanisms for branch mispredictions.

In this paper, we present a simple mechanism for recovering the branch history after a misprediction. The proposed mechanism adds an age_counter to the original predictor and doubles the size of the branch history register. The age_counter counts the number of outstanding branches and uses it to recover the branch history register.

Simulation results on the SimpleScalar 3.0/PISA tool set and the SPECINT95 benchmarks show that gshare and GAg with the proposed recovery mechanism improved the average prediction accuracy

† 종신회원 : 국립한국농업전문학교 교수
ko@kn.ac.kr
†† 비 회 원 : 수원대학교 컴퓨터학과 교수
yicho@suwon.ac.kr
논문접수 : 2004년 4월 20일
심사완료 : 2004년 12월 1일

by 2.14% and 9.21%, respectively and the average IPC by 8.75% and 18.08%, respectively over the original predictor.

Key words : branch prediction, misprediction, speculative update, branch history, recovery mechanism, outstanding branches, prediction accuracy

1. 서 론

사이클 당 다수의 명령어를 수행시키는 슈퍼스칼라 프로세서는 유용한 명령어를 계속 공급하기위해서 정확한 분기예측기를 필요로 한다. 그러나 파이프라인 깊이(depth)와 이슈 폭(width)이 증가할수록 분기예측실패(branch misprediction)는 올바른 경로의 명령어를 수행하기 전 잘못된 경로상의 명령어들을 수행하게 되므로 많은 사이클을 낭비하고 프로세서 자원을 낭비하게 된다. 이로 인해 예측정확도가 높은 동적으로 분기결과를 예측하는 기술에 대한 연구가 계속되고 있다. 동적 분기예측 방법은 실행시간동안 얻어지는 정보를 이용하여 예측기를 학습시켜 예측정확도를 향상시키는 방법이다. 동적 분기예측 방법 중 2-단계 적응 분기예측기(two-level adaptive branch predictor)[1]는 특히 효과적임이 증명되었다. 이는 선행 분기명령들의 분기결과를 예측하려는 분기명령과 상호관련(correlation)시켜 예측하기 때문이다. 현재, 2-단계 분기예측기와 그것을 변형시킨 방법은 여러 상용 프로세서에서 사용하고 있다. AMD의 K6와 K7은 GAs 예측기를 사용하고 있고[2], HP-PA 8700은 GAs에 Agree 메커니즘을 결합한 분기예측기를 사용하고 있고, Alpha 21264는 두 개의 2-단계 예측기를 혼합한 분기예측기를 사용하고 있다[3].

최근에는 분기명령의 지역 분기 히스토리(local branch history)와 전역 분기 히스토리(global branch history)를 선택적으로 사용하여 예측하는 하이브리드 분기예측기[4]와 지역 히스토리와 전역 히스토리를 결합하여 예측하는 결합형 예측기[5]가 연구되고 있다.

파이프라인 프로세서에서 분기명령어는 반입시간(fetch stage)에 분기결과를 예측하고, 해당 분기명령의 실제결과는 실행시간(execute stage)에 결정되며, 분기예측이 실패한 경우 명령어 완료시간에 잘못된 경로에서 수행된 명령어들을 무효화시키고 올바른 경로의 명령어들을 반입하여 실행시킨다.

분기를 예측하고, 올바른 결과가 결정되는 시간사이에 여러 사이클이 경과된다. 파이프라인 깊이와 이슈 폭이 큰 프로세서에서는 이 시간동안 상당수의 분기명령들이 반입되어 예측된다. 분기예측기가 명령어 완료시간에 갱신된다면 수행중인(in-flight) 이들 분기명령들의 결과는 부정확한 분기 히스토리(stale branch history)로 예측된 것이다. 이것은 특히 2-단계 적응 예측기들에서 문제가 된다[6,7]. 왜냐하면, 이들 방법은 이전 분기들의 동작에 대한 정확한 분기 히스토리에 의존하기 때문이다[8].

예측기가 실제결과(resolved outcome)대신 예측결과(predicted outcome)로 모험적 갱신(speculative update)을 할 수 있다면 문제를 해결할 수 있다. 예측이 올바르다면 모험적 갱신은 어떤 해도 발생하지 않는다. 그러나 분기예측실패(branch misprediction)의 경우 모험적 갱신은 분기 히스토리에 잘못된 정보를 삽입하였기 때문에 오염된 분기 히스토리를 분기명령을 예측하기 직전의 상태로 복구시켜야 한다. 이전에 제안된 복구 메커니즘(recovery mechanism)들은 분기명령 예측 시 큐(queue)나 리오더 버퍼(reorder buffer)에 분기 히스토리를 저장했다가 분기예측실패 시 저장하였던 값으로 복구하는 복잡한 메커니즘을 사용하였다[9-11].

본 논문에서는 분기예측실패 후에 분기 히스토리를 복구하는 간단한 메커니즘을 제안한다. 제안한 메커니즘에는 기존의 분기예측기에 미해결 분기명령어 수를 저장하는 age_counter을 추가하고, 분기 히스토리 레지스터 크기를 2배로 확장시킨다. age_counter를 이용하여 분기예측실패 후 분기 히스토리 레지스터를 복구시킨다.

SimpleScalar 3.0/PISA 툴셋으로 시뮬레이션 한 결과, 제안된 복구 메커니즘을 기존의 전역 히스토리 기반 분기예측기에 적용하였을 때 예측 정확도와 프로세서 성능을 개선시킬 수 있었음을 확인하였다.

2. 연구배경

2.1 관련 연구

이전의 연구에서 분기 히스토리 레지스터를 모험적 갱신할 필요성을 보여주었다. Yeh와 Patt[1]는 모험적 갱신이 예측 정확도를 개선시킬 수 있다고 주장하였으나, 분기예측실패 시 복구 메커니즘은 다루지 않았다. Talcott[6]은 오래된 분기 히스토리(older branch histories)는 최근의 분기 히스토리(newer branch histories)만큼 잘 동작하므로 모험적 갱신이 불필요하다고 주장했다. 그들은 고정된 수의 최근 분기들의 결과를 배제한 분기 히스토리를 사용하여 실험하였다. Hao[7]은 분기를 예측할 때마다 미해결 분기의 수가 다르다는 것을 확인함으로써 Talcott의 결과를 반박하였다. 그들의 결과는 전역 히스토리를 유지하나 각 분기가 자신의

2-비트 카운터 테이블을 갖는 GAp 예측기에서 고려하였다. Jourdan[10]은 gshare 예측기에 대해 이들 결과를 확인하였고, 전역 히스토리에 초점을 맞추어, 분기예측실패에 의해 오염되는 히스토리를 복구하는 메커니즘을 제안하였다. 그들은 복구할 분기 히스토리를 리오더 버퍼에 저장하는 것으로 계획하였다. Jourdan은 다른 분기예측기 구조에서 모험적 갱신의 유용함을 연구하였고, 특히, PHT(Pattern History Table)의 갱신시간은 예측 정확도에 민감하지 않다는 것을 보여주었다. 한편, Jourdan은 RAS(Return Address Stack)에서의 모험적 갱신과 복구도 성능에 많은 도움을 준다고 하였다. Skadron등[11]은 이 논문을 확대하여 간단한 RAS 복구 메커니즘을 소개하였다.

2.2 모험적 갱신과 복구 메커니즘 의 필요성

명령어의 반입 폭과 이슈 폭이 증가함에 따라 GAg [1], gshare[8]와 같이 전역 히스토리를 사용하는 분기예측기에서는 분기명령어가 예측된 후 실제 분기결과 값으로 예측기의 전역 히스토리 레지스터(GHR:Global History Register)를 수정하기 전에 다른 분기명령어를 예측하는 경우가 대부분이다. 이런 경우, 전역 히스토리 레지스터는 최근의 분기명령들에 대한 히스토리를 포함시키지 못하므로 최근의 분기명령들과 강한 상호관계(strong correlation)를 갖는 분기명령을 예측하려할 때 예측정확도가 떨어져 프로세서의 성능을 저하시키게 된다[7].

분기예측기는 프로그램이 어떤 상태에 있는지를 식별하기 위해 전역 히스토리 레지스터를 사용하고, 예측은 해당 상태를 기초로 한다. 그러나 프로그램 수행 시 예측은 되었으나 결과가 아직 전역 히스토리 레지스터에 반영되지 않은 미해결 분기명령어 수의 다양한 동적 변화는 전역 히스토리 레지스터가 더 이상 특정 프로그램 상태의 모든 동적발생(every dynamic occurrences)에 대해 동일한 히스토리를 갖는 것을 보장할 수 없다. 따라서 전역 히스토리 레지스터는 프로그램에서 상태를 식별하는 예측기의 능력이 감소되게 된다. 다시 말해, 동일 분기명령의 다른 동적 발생들을 예측할 때 캐시미스, 다른 분기명령의 예측실패, 자원부족 등과 같이 동적으로 변하는 상태에 따라 미해결 분기명령어 수는 달라질 수 있다. 두 개의 동적 발생에 대한 실제 분기 히스토리가 같다면, 동일한 예측을 해야 하지만 예측할 때 미해결 분기명령어 수가 다르기 때문에 사용되는 전역 히스토리 레지스터의 내용이 달라진다. 이것은 예측기가 각 예측을 위해 다른 PHT 엔트리를 사용하게 하여 잠재적으로 다른 예측을 만들 수 있으며, 결과적으로 예측 정확도를 떨어지게 할 수 있다[7].

이런 문제들은 분기명령을 예측한 후 그 예측 값으로 전역 히스토리 레지스터를 모험적으로 갱신하면 해결될 수 있다[7,9]. 모험적으로 갱신한 예측 값이 실제 결과 값과 동일하면 어떤 해도 미치지 않는다. 그러나 예측 값이 실제 결과와 다른 경우 전역 히스토리 레지스터는 잘못된 경로상의 분기명령들의 모험적 갱신으로 오염되어 있으므로 이것을 복구시키지 않으면 예측실패 후 프로세서 상태가 복구된 뒤 올바른 경로상의 분기명령을 예측하게 될 때 오염된 전역 히스토리 레지스터를 사용하게 되므로 예측정확도가 떨어지는 결과를 낳게 된다. 따라서 분기예측실패 후 프로세서의 상태복구와 더불어 전역 히스토리 레지스터를 분기예측실패 직전의 히스토리로 복구시키는 메커니즘이 필요하다.

2.3 전역 히스토리 레지스터 복구 메커니즘

프로세서가 어떤 분기명령어의 분기예측실패 후 모든 명령어를 무효화하기 때문에 분기 히스토리의 모험적 갱신을 허용하는 분기예측기는 예측실패가 발생하기 직전의 분기 히스토리로 복구시키고, 올바른 분기결과로 분기 히스토리를 갱신한다. 분기실패와 그의 검출사이의 시간동안에 잘못된 경로상의 분기명령들은 분기 히스토리를 오염시키지만 이러한 잘못된 경로상의 명령어들은 무효화되기 때문에 분기 히스토리에 해를 주지 않는다. 분기실패가 검출되고, 분기 히스토리가 복구된 후, 올바른 경로상의 명령어가 반입되며, 이후의 분기명령은 복구된 분기 히스토리를 사용하게 된다. PHT 갱신은 명령어 완료시간에 수행하기 때문에 복구가 필요 없다.

다음은 분기예측실패 후 전역 분기 히스토리를 복구시키는 기존의 메커니즘을 설명한다.

2.3.1 History-Based 복구 메커니즘

첫 번째 방법은 모험적 갱신이 전역 히스토리 레지스터를 수정하는 방법이다[9]. 분기 히스토리 레지스터의 내용이 수정되기 전, OBQ(Outstanding Branch Queue)의 꼬리(tail)에 예측 직전의 전역 히스토리 레지스터의 내용을 저장한다. 따라서 OBQ는 분기명령들이 반입된 순서로 모든 미해결 분기명령어들에 대한 예측 직전의 전역 히스토리 레지스터의 내용을 저장한다. 예측은 항상 전역 히스토리 레지스터를 사용하고, 분기명령어가 완료되면 OBQ의 머리(head)를 제거한다. 분기예측실패 후의 복구는 OBQ에서 예측이 실패한 분기명령의 엔트리를 식별하기 위해 OBQ를 검색하고, 해당 엔트리 이후의 엔트리를 제거하고, 전역 히스토리 레지스터의 내용을 예측실패 직전의 히스토리로 복구시키고, 예측실패한 분기명령의 올바른 결과를 전역 히스토리 레지스터에 추가한다.

2.3.2 Future-Based 복구 메커니즘

두 번째 방법은 전역 히스토리 레지스터를 저장하는 대신 모험적으로 갱신한 히스토리를 OBQ에 저장한다.

이런 future-based 복구 메커니즘[9]은 완료시간 히스토리(commit history)를 전역 히스토리 레지스터에서 유지한다. 동작은 history-based 메커니즘보다 약간 복잡한데 그 이유는 예측 시 두 장소 중 한 곳에 있는 가장 최근의 히스토리를 찾아야하기 때문이다. 예측은 OBQ가 모험적 갱신된 히스토리를 포함하고 있는지 조사하는 것을 요구한다. 포함하고 있으면 예측은 가장 최근의 모험적 히스토리를 갖고 있는 OBQ의 꼬리를 사용한다. 포함하고 있지 않다면 예측은 전역 히스토리 레지스터를 사용한다. 분기명령이 완료될 때 OBQ의 머리를 새롭게 완료시간 히스토리로서 전역 히스토리 레지스터로 보낸다.

복구 메커니즘은 예측이 실패한 분기명령에 대응하는 OBQ 엔트리를 찾고, 해당 엔트리 이후의 엔트리를 OBQ에서 제거한다. 전역 히스토리 레지스터는 항상 완료시간 히스토리를 갖고 있으므로 복구시킬 필요가 없다

3. 제안한 전역 히스토리 레지스터 복구 메커니즘

전역 히스토리 레지스터의 모험적 갱신을 허용하는 분기예측기에서 분기예측실패 시 전역 히스토리 레지스터를 예측이 실패한 분기명령의 분기예측 직전의 상태로 복구시키는 기존에 제안된 메커니즘은 2장에 소개하였다. 기존의 메커니즘은 대부분이 큐를 사용하므로 복잡한 하드웨어를 요구하고 있다. 본 논문에서는 기존의 메커니즘과 같이 전역 히스토리 레지스터를 복구하면서 간단한 하드웨어로 구현시키는 메커니즘을 제안한다. 기존의 복구 메커니즘인 history-based 메커니즘과 future-based 메커니즘에서 OBQ의 크기는 PHT의 엔트리 수가 8K, 명령어 윈도우(instruction window)의 엔트리 수가 128, 최대 미해결 분기수를 20개라 가정했을 때 OBQ의 한 엔트리 크기는 21비트(전역 히스토리 레지스터의 내용을 저장하기 위한 13비트, 한 분기 명령이 OBQ에 여러 엔트리를 가질 수 있으므로 이를 식별하기 위해 명령어 윈도우 엔트리 식별자를 OBQ 엔트리에 태그하기 위한 8비트)이므로 총 420비트와 큐를 구동시키는 로직이 필요하다. 이에 비해 제안된 메커니즘은 SHR를 위해 13비트가 추가되고, age_counter를 위해 5비트가 추가되므로 총 18비트가 필요하다.

3.1 분기예측과 모험적 갱신방법

제안한 분기예측기에는 모험적 갱신과 분기예측실패 시 분기 히스토리를 복구하기 위해 age_counter를 추가하였고, 전역 히스토리 레지스터 대신 SHR(Speculative History Register)를 사용한다. age_counter는 현재 반입되어 예측되었으나 아직 실제결과로 분기 히스토리를 갱신하지 못한 미해결 분기명령어의 수를 나타낸다. 새로운 분기명령이 반입되면 미해결 분기명령어가 한 개 증가하므로 age_counter를 '1' 증가시키고, 분기명령어의 완료시간에 분기명령어가 올바로 예측이 되었으면 미해결 분기명령어가 한 개 줄어들게 되므로 age_counter를 '1' 감소시킨다.

기존의 예측기에서 전역 히스토리 레지스터는 반입된 순서대로 완료시간에 실제 분기결과로 갱신된 분기 히스토리를 가지나, 본 논문의 분기예측기에 있는 SHR은 완료시간에 갱신된 히스토리와 현재 수행중인 분기명령들의 예측 값으로 모험적 갱신한 히스토리를 포함한다.

그림 1은 age_counter가 c일 경우 즉, 미해결 분기명령이 c 개일 때의 SHR의 상태를 나타낸다. B_0에서 B_{C-1}은 수행중인 분기명령들의 모험적 갱신 히스토리를 나타내고, B_C에서 B_{n+m-1}은 완료시간 갱신 히스토리를 나타낸다. SHR은 전역 히스토리 레지스터와는 다르게 모험적 갱신을 위해 m 비트를 추가하였다. 시뮬레이션 시 m의 값은 n와 동일한 값을 갖도록 하였다.

전역 히스토리 레지스터의 모험적 갱신을 허용하는 gshare 분기예측기에서 조건 분기명령에 대한 예측은 그림 2와 같이 SHR에서 최근의 마지막 n 개(PHT 엔트리 수가 2^n 일 때)의 분기명령어 결과를 분기명령어의 프로그램 카운터와 exclusive-or 하여 PHT를 인덱스하

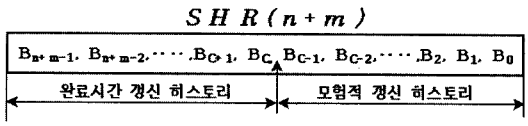


그림 1 age_counter가 c일 경우 SHR의 상태

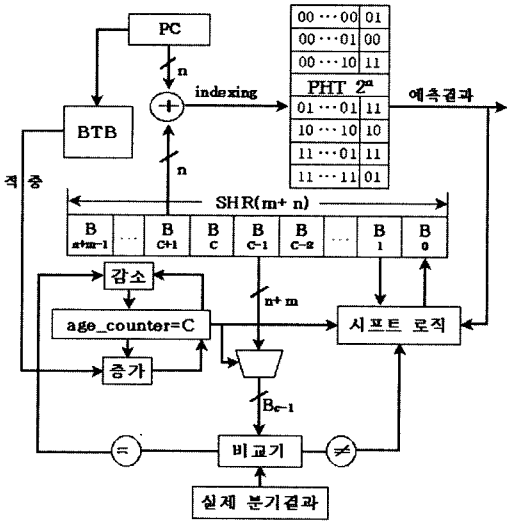


그림 2 gshare에 제안한 복구 메커니즘을 적용한 분기예측기 구조

고(GAg의 경우 최근 마지막 n개의 분기명령어 결과를 사용하여 PHT 인덱스) 선택된 PHT 엔트리(포화 카운터)의 값이 '2' 이상이면 taken으로 '1' 이하면 not-taken으로 분기를 예측한다. 조건 분기명령을 예측할 때 age_counter의 내용을 '1' 증가시켜 현재 미해결 분기명령어의 수가 '1' 증가되었음을 나타낸다. 모험적 갱신은 분기명령에 대한 분기예측결과를 사용하여 SHR을 갱신하게 되는데, SHR을 좌로 '1' 비트 이동시키고, B_0에 예측한 결과를 저장한다.

3.2 분기예측실패 시 복구 메커니즘

SHR을 모험적으로 갱신한 예측 값이 실제결과 값과 동일하다면 프로세서는 정상적으로 계속 동작한다. 그러나 예측 값이 실제결과와 다를 경우 SHR은 잘못 예측된 경로상의 분기명령들에 대한 모험적 갱신으로 오염되어 있으므로 SHR을 복구시키지 않으면 예측실패 후 완료시간에 프로세서 상태가 복구된 뒤 올바른 경로상의 분기명령을 예측하려할 때 오염된 SHR을 사용할 수 밖에 없어 예측정확도가 떨어지는 결과를 낳게 된다. 따라서 오염된 SHR에 대해 예측이 실패된 분기명령의 예측 직전 히스토리로 SHR을 복구시켜야 한다.

본 논문에서 제시한 복구 메커니즘은 age_counter를 사용하여 완료시간에 SHR을 예측 직전의 히스토리로 복구시킨다.

그림 1에서 BC는 현재 마지막으로 완료된 분기명령의 분기결과를 나타낸다. 만약 어떤 분기명령의 실제결과가 구해졌다면 완료시간에 그 분기명령이 모험적으로 갱신된 결과인 BC-1과 실제결과를 비교한다. 만약 동일하다면 올바로 예측한 경우이므로 프로세서는 정상적으로 진행된다. 이때 age_counter를 '1' 감소시켜 분기 히스토리에서 미해결 분기명령의 수가 하나 줄었음을 나타낸다. 그러나 두 결과가 다르다면 분기예측실패가 검출된 경우이고, SHR에서 BC-2로부터 B_0까지는 잘못 예측된 경로상의 분기명령들이 모험적으로 갱신됨에 따라 분기 히스토리를 오염시키게 된 것이다. 따라서 SHR을 c-1 비트만큼 우로 이동시켜 예측이 실패된 분기명령의 예측 바로 직전의 상태로 SHR을 복구시키고, 예측이 실패한 분기명령 이후에 반입되어 모험적으로 갱신되었던 히스토리를 무효화 시킨 뒤, 예측실패한 분기명령의 실제결과를 B_0에 저장한다. 이때 age_counter의 값은 '0' 으로 초기화시켜, 현재 미해결 분기명령어가 없음을 나타낸다.

4. 성능측정 및 분석

4.1 실험환경

성능 측정을 위해 슈퍼스칼라 프로세서의 사이클 수준 시뮬레이터인 SimpleScalar 3.0/PISA 툴셋[12]에서 모험적 갱신을 허용하는 gshare와 GAg 분기예측기에 본 논문에서 제안한 분기예측실패 복구 메커니즘을 적용하도록 bpred와 sim-outorder를 수정하였다.

표 1은 시뮬레이션된 프로세서의 구성에 대한 머신

표 1 시뮬레이션 프로세서 구성에 사용된 머신 파라미터

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	inorder
	Functional units	8 i-ALU(1), 8 f-ALU(2) 2 i-MULT/DIV(3/12) 2 f-MULT/DIV(4/12)	( ) is latency
Memory	Memory ports	2 ports, 8byte bus	( ) is latency
	Memory access latency	first_chunk(18), inter_chunk(2)	
Branch predictor	gshare	(1K/2K/4K/8K) entries	( ) is PHT entry size
	GAg	(1K/2K/4K/8K) entries	
	BTB	512 sets, 4 way	
Cache	L1 data cache	128K, 32B block, 4 way, LRU, 1 cycle latency	
	L1 instruction cache	512K, 32B block, d-map, LRU, 1 cycle latency	
	L2 unified cache	1M, 64B block, 4 way, LRU, 6 cycle latency	

표 2 벤치마크 프로그램과 입력 데이타

벤치마크	입력 데이타	수행된 명령어수 (million)	비 고
gcc	cccp.i	200	The GNU C compiler version 2.5.3.
li	train.lsp	183	Xlisp interpreter.
vortex	persons.250	200	An object oriented database.
go	50 9 2stone9.in	200	An internationally ranked go playing program.
m88ksim	dcrand.lit	200	A chip simulator for the Motorola 88100 microprocessor.
compress	100000 e 2231	200	An in-memory version of the common UNIX utility.
perl	primes.pl	200	An interpreter for the Perl language.
ijpeg	penguin.ppm	200	Image compression/decompression on in memory images.

파라메타를 나타낸다. 8 이슈 프로세서를 기반으로 하였고, 1K~8K 엔트리 PHT를 갖는 gshare와 GAg 분기예측기로 실험하였다. 모든 분기예측기의 BTB(Branch Target Buffer)는 512 sets 4-way로 고정하였다. 128 KB L1 데이타 캐시, 512KB L1 명령어 캐시, 1MB L2 통합 캐시를 사용하였다.

표 2는 시뮬레이션에서 사용된 SPECINT95 벤치마크 프로그램[13]과 입력 데이타를 설명한다. 입력 데이타는 ref input을 사용하였고, 시뮬레이션 시간을 줄이기 위해 벤치마크 프로그램의 수행된 명령어수를 200M (million)까지로 제한하였다.

4.2 성능 분석

1K~8K 엔트리의 PHT를 갖는 모험적 갱신을 허용하는 gshare와 GAg 분기예측기에 제안한 복구 메커니즘을 적용하지 않았을 경우(GAg, gshare)와 적용했을 경우(GAg+recovery, gshare+recovery)의 예측정확도와 성능 향상(speedup)을 측정하였다. 또, gshare와 GAg 분기예측기에 기존의 복구메커니즘인 History-Based 복구 메커니즘과 Future-Based 복구 메커니즘을 적용하여 실험한 후 제안한 복구메커니즘의 실험결과와 비교한 결과 예측정확도와 성능향상의 정도가 모두 오차 허용범위(예측정확도는 최고 0.05%이내, IPC는 0.01이내)내에 있었으므로 기존의 복구메커니즘과 제안된 메커니즘의 결과 비교는 생략하였다. 그러나 제안한 복구 메커니즘은 간단한 하드웨어 추가로 기존의 복잡한 하드웨어를 요구하는 복구 메커니즘과 같은 예측정확도와 성능을 개선시킴을 확인할 수 있었다.

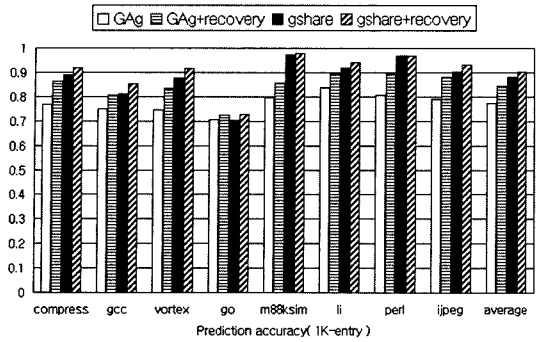
그림 3은 1K(그림 3(a)), 2K(그림 3(b)), 4K(그림 3(c)), 8K(그림 3(d)) 엔트리의 PHT를 갖는 GAg와 gshare에 제안한 분기예측실패 복구메커니즘을 적용하기전과 적용후의 예측정확도를 나타낸다. 그림의 범례에서 GAg는 분기예측실패 복구 메커니즘을 적용하지 않았을 경우의 예측정확도를 나타내고, GAg+recovery는 GAg 예측기에 분기예측실패 복구 메커니즘을 적용했을 경우의 예측정확도를 나타낸다. 또, gshare는 분기예측실패 복구 메커니즘을 적용하지 않았을 경우의 예측정확도를 나타내고, GAg+recovery는 gshare에 분기예측실패 복구 메커니즘을 적용했을 경우의 예측정확도를 나타낸다.

GAg에서는 분기예측실패 복구 메커니즘을 적용시키는 경우 최소 8.89%(1K엔트리 PHT), 최대 9.63%(8K 엔트리 PHT), 평균 9.21%의 예측정확도가 개선되었다. GAg 예측기에서는 PHT 엔트리수가 증가할수록 분기예측실패 복구 메커니즘이 예측 정확도를 좀더 개선시키는 것으로 나타났다.

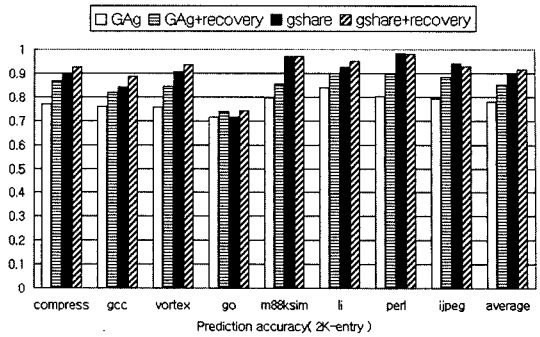
gshare에서는 분기예측실패 복구 메커니즘을 적용시키는 경우 최소 1.9%(8K 엔트리 PHT), 최대 2.65% (1K 엔트리 PHT), 평균 2.14%의 예측정확도가 개선되었다. gshare에서는 엔트리수가 적을 때 분기예측실패 복구 메커니즘에 의한 개선이 약간 좋은 것으로 나타났다.

GAg+recovery가 gshare+recovery보다 예측정확도가 많이 개선된 것은 GAg는 PHT를 인덱스할 때 전역 히스토리 레지스터만 사용하지만 gshare는 전역 히스토리 레지스터와 분기명령의 주소를 exclusive-or하여 인덱스 하므로 전역 히스토리 레지스터에 대한 영향이 GAg보다 작기 때문이라 생각된다.

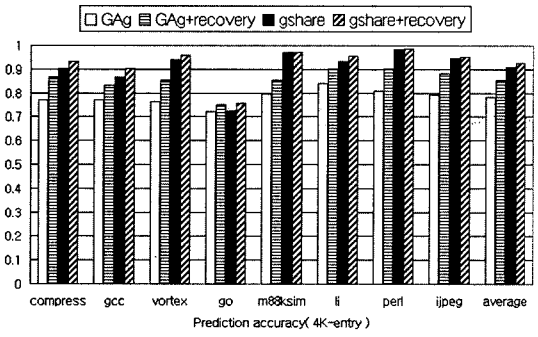
그림 4는 GAg와 gshare에 제안한 분기예측실패 복구 메커니즘을 적용하기 전에 비해 적용후의 성능 향상(speedup)을 IPC(Instructions Per Cycle)로 나타내었다. GAg에서는 분기예측실패 복구 메커니즘을 적용시 최소 17%(2K 엔트리PHT), 최대 20.20%(8K 엔트리 PHT), 평균 18.08%의 IPC가 개선되었다. IPC도 예측정확도와 같이 GAg에서는 PHT 엔트리수가 증가하면 IPC가 좀더 개선되는 것으로 나타났다. gshare에서는 분기예측실패 복구 메커니즘을 적용시 최소 6.36%(8K 엔트리 PHT), 최대 13.09%(1K 엔트리 PHT), 평균 8.75%의 IPC가 개선되었다. 이것은 GAg+recovery가 gshare+recovery보다 예측정확도를 더 개선시키기 때문에 프로세서의 성능을 더 많이 개선시킨 결과라 생각된다.



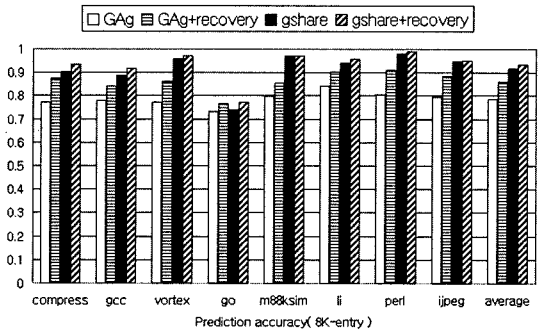
(a) 1K 엔트리 PHT



(b) 2K 엔트리 PHT

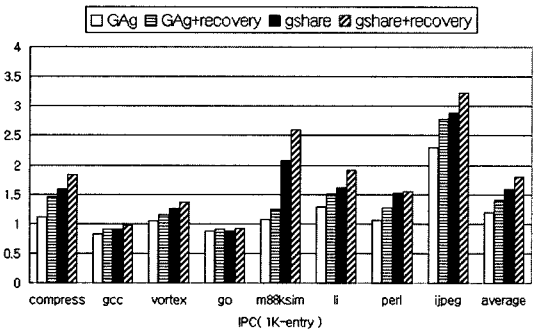


(c) 4K엔트리 PHT

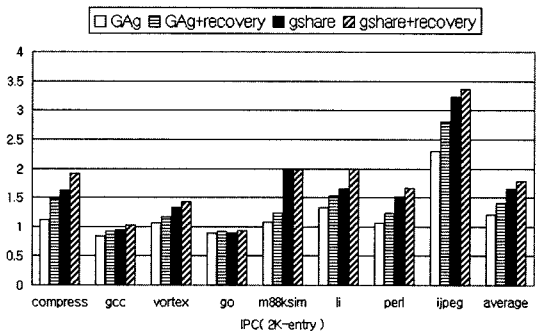


(d) 8K엔트리 PHT

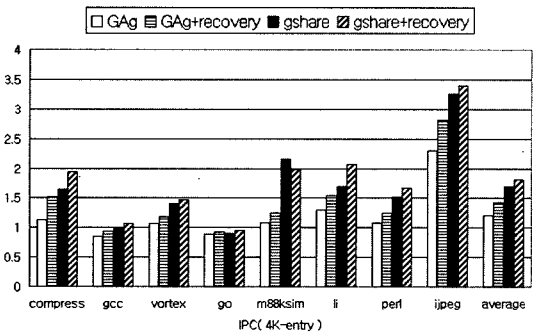
그림 3 GAg와 gshare에 복구 메커니즘을 적용시 예측정확도 비교



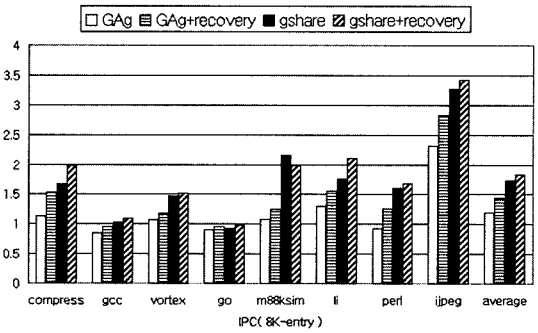
(a) 1K엔트리 PHT



(b) 2K엔트리 PHT



(c) 4K엔트리 PHT



(d) 8K엔트리 PHT

그림 4 GAg와 gshare에 복구 메커니즘을 적용시 IPC 비교

5. 결 론

본 논문에서는 전역 히스토리 레지스터를 모험적으로 갱신하는 전역 히스토리 기반 분기예측기에서 분기예측이 실패 후 전역 히스토리 레지스터를 분기예측 직전의 히스토리로 복구하는 간단한 메커니즘을 제안하였다. 기존의 복구 메커니즘이 큐와 같은 복잡한 하드웨어의 추가를 요구하는 반면에 제안한 메커니즘은 거의 하드웨어의 추가 없이 동일한 효과를 얻었다.

전역 히스토리를 기반으로 하는 예측기인 GAg와 gshare에 제안한 복구 메커니즘을 추가하여 SimpleScalar 3.0/PISA 툴셋에서 시뮬레이션 한 결과 예측정확도의 증가와 함께 프로세서 성능 향상의 효과를 확인하였다. 제안한 복구 메커니즘을 GAg 예측기에 적용한 결과 적용전보다 평균 9.21%의 예측정확도가 개선되었고, 평균 18.08%의 IPC가 개선되었다. 또, gshare 예측기에 적용한 결과 평균 2.14%의 예측정확도가 개선되었고, 평균 8.75%의 IPC가 개선되었다.

제안한 복구 메커니즘을 하이브리드 분기예측기와 오버라이딩 분기예측기에 적용시키면 예측정확도와 성능 개선에 도움이 될 것이라 기대된다.

참 고 문 헌

[1] T.-Y. Yeh and Y. N. Patt, *"Alternative implementations of two-level adaptive branch prediction,"* in *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pp. 124-34, May 1992.

[2] K. Diefendorff, *"K7 challenges Intel,"* *Microprocessor Report*, pp. 1, 6-11, Oct. 26 1998.

[3] R. E. Kessler, E. J. McLellan, and D. A. Webb, *"The Alpha 21264 microprocessor architecture,"* in *Proceedings of the 1998 International Conference on Computer Design*, pp. 90-95, Oct. 1998.

[4] G.H. Loh, D.S. Henry, *"Predicting Conditional Branches with Fusion-based Hybrid Predictors,"* *PACT2002*, pp 395-405, Sep. 2002.

[5] Z. Lu, J. Lach, M. Stan, and K. Skadron. *"Alloyed Branch History: Combining Global and Local Branch History for Robust Performance,"* *International Journal of Parallel Programming, Kluwer*, volume 31, number 2, Apr. 2003.

[6] A. R. Talcott, W. Yamamoto, M. J. Serrano, R. C. Wood, and M. Nemirovsky, *"The impact of unresolved branches on branch prediction scheme performance,"* in *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pp. 12-21, Apr. 1994.

[7] E. Hao, P.-Y. Chang, and Y. Patt, *"The effect of speculatively updating branch history on branch prediction accuracy, revisited,"* in *Proceedings of the 27th Annual International Symposium on Microarchitecture*, pp. 228-32, Nov. 1994.

[8] M. Evers, S. J. Patel, R. S. Chappell, and Y. N. Patt, *"An analysis of correlation and predictability: What makes two-level branch predictors work,"* in *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pp. 52-61, June 1998.

[9] K. Skadron, and M. Martonosi, *"Speculative Updates of Local and Global Branch History : A Quantitative Analysis,"* *JILP* Vol. 2, Jan. 2000.

[10] S. Jourdan, J. Stark, T.-H. Hsing, and Y. N. Patt, *"Recovery requirements of branch prediction storage structures in the presence of mispredicted-path execution,"* *International Journal of Parallel Programming*, vol. 25, pp. 363-83, Oct. 1997.

[11] K. Skadron, P. S. Ahuja, M. Martonosi, and D. W. Clark, *"Improving prediction for procedure returns with return-address-stack repair mechanisms,"* in *Proceedings of the 31st Annual ACM/IEEE International Symposium on Microarchitecture*, pp. 259-71, Dec. 1998.

[12] D. Burger, T. M. Austin, and S. Bennett, *"Evaluating future microprocessors: the SimpleScalar tool set,"* *Tech. Report TR-1308, University of Wisconsin-Madison Computer Sciences Department*, July 1996.

[13] The Standard Performance Evaluation Corporation, *"SPEC CPU95 Benchmarks,"* *WWW site: http://www.specbench.org/osg/cpu95*, Dec. 1999.

고 광 현

1985년 서울산업대학교 공학사. 1989년 한양대학교 공학석사. 2005년 수원대학교 이학박사. 1978년~1997년 농촌진흥청 전산실장. 1997년~현재 국립한국농업전문학교 교수. 관심분야는 ILP프로세서, 자연어처리, 농업정보기술 등

조 영 일

1980년 한양대학교 전자공학과 공학사
1982년 한양대학교 전자공학과 공학석사
1985년 한양대학교 전자공학과 공학박사
1986년 3월~현재 수원대학교 컴퓨터학과 교수. 관심분야는 ILP 프로세서, 이동통신, 최적화 컴파일러 설계