간단한 전역 분기 히스토리 복구 메커니즘

김주익*O 고광현** 조영일*
수원대 컴퓨터학과, 한국농업전문학교
mkby12@hanmail.net, ko@kn.ac.kr, yicho@suwon.ac.kr

Simple Recovery Mechanism for Global Branch History

Ju-ick Kim*O Kwang-hyun Ko** Young-il Cho*
*Dept. Computer Science Univ. of Suwon, **Korea National Agricultural College

요 약

조건 분기예측실패는 많은 사이클을 낭비시키며, 비순서적 실행을 방해하고, 잘못 예측된 명령어들을 수행하게 되므로 전력을 낭비한다. gshare와 GAg같은 전역 히스토리를 기반으로 하는 예측기에서는 히스토리의 명령어 완료시간 갱신(commit update)에 의해 많은 분기예측실패가 발생한다. 이를 위해 히스토리를 모험적으로 갱신하고, 분기예측실패 시 히스토리를 복구시키는 메커니즘에 관한 연구들이 제시되었다.

본 논문에서는 기존 분기예측기에 age_counter를 추가하여 미해결 분기명령어 수를 저장하며, 이를 분기예측실패 후 분기 히스토리 레지스터를 복구하는데 사용하는 간단한 복구 메커니즘을 제안한다. SimpleScalar 3.0/PISA 툴셋과 SPECINT95 벤치마크 프로그램에서 시뮬레이션 한 결과, 제안된 복구 메커니즘은 GAg와 gshare 예측기에서 예측정확도는 각각 9.21%와 2.14%가 개선되었고, IPC는 18.08%와 8.75% 개선되었다.

1. 서 론

슈퍼스칼라 프로세서에서 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. 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.

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 이들 분기명령들의 결과는 부정확한 분기히스토리로 예측된 것이다. 이것은 이전 분기들의 동작에 대한 정확한 분기 히스토리에 의존하는 2-단계 적응 예측기들에서 문제가 된다[2,3,4].

예측기가 실제결과대신 예측결과로 모험적 갱신(speculative update)을 할 수 있다면 문제를 해결할 수 있다. 예측이 올바르다면 모험적 갱신은 어떤 해도 발생하지 않는다. 그러나 분기예측실패의 경우 모험적 갱신은 분기 히스토리에 잘못된 정보를 삽입하였기 때문에 오염된 분기 히스토리를 분기명령을 예측하기 직전의 상태로 복구시켜야 한다.

History-Based 복구 메커니즘(recovery mechanism)은 모험적 갱신이 전역 히스토리 레지스터를 수정하는 방법이다[5]. 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.

Future-Based 복구 메커니즘[5]은 전역 히스토리 레지스터를 저장하는 대신 모험적으로 갱신한 히스토리를 OBQ에 저장한다. Future-Based 복구 메커니즘은 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.

위에 기술한 기존의 복구 메커니즘들은 분기명령 예측 시 큐나 리오더 버퍼에 분기 히스토리를 저장했다가 분기예측실패 시 저장하였던 값으로 복구하는 복잡한 메커니즘을 사용하였다.

본 논문에서는 기존 분기예측기에 age_counter를 추가하여 미해결 분기명령어 수를 저장하며, 분기예측실패 후 분기 히스토리 레지스터를 복구하는 간단한 복구 메커니즘을 제안한다.

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. 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 기존의 메커니즘은 대부분이 복잡한 하드웨어를 요구하고 있다.

본 논문에서는 기존의 메커니즘과 같이 분기예측실패 시 전역 히스토리 레지스터를 복구하면서 간단한 하드웨어로 구현시키는 메커니즘을 제안한다. 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.

전역 히스토리 레지스터의 모험적 갱신을 허용하는 gshare 분기예측기에서 조건 분기명령에 대한 예측은 그림1와 같이 SHR에서 최근 마지막 n 개의 분기명령어 결과와 프로그램 카운터를 사용하여 PHT를 인덱스하고 선택된 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.

SHR을 모험적으로 갱신한 예측 값이 실제결과 값과 동일하다면 프로세서는 정상적으로 계속 동작하나 예측 값이 실제결과와 다를 경우 SHR은 잘못 예측된 경로상의 분기명령들에 대한 모험적 갱신으로 오염되어 있으므로 SHR을 복구시키지 않으면 오염된 SHR을 사용하게 되어 예측정확도가 떨어지게 된다. 따라서 오염된 SHR에 대해 예측이 실패한 분기명령의 예측 직전 히스토리로 SHR을 복구시켜야 한다.

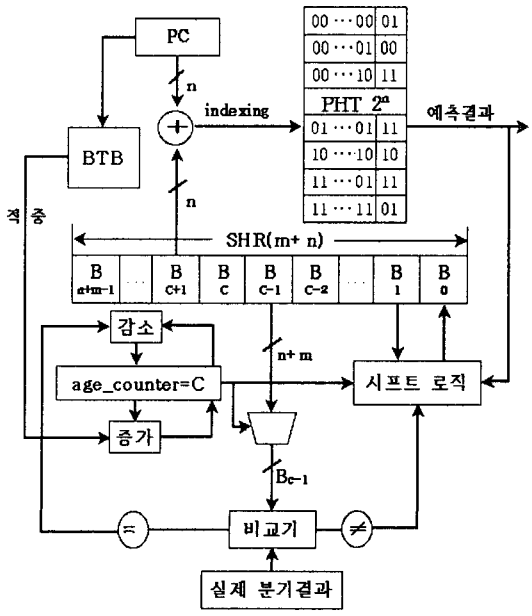

그림1. gshare에 제안한 복구 메커니즘을 적용한 예측기 구조

본 논문에서 제시한 복구 메커니즘은 age_counter를 사용하여 완료시간에 SHR을 예측 직전의 히스토리로 복구시킨다. 그림1에서 B_C는 현재 마지막으로 완료된 분기명령의 분기결과를 나타낸다. 만약 어떤 분기명령의 실제결과가 구해졌다면 완료시간에 그 분기명령이 모험적으로 갱신된 결과인 B_{C-1}과 실제결과를 비교한다. 만약 동일하다면 예측이 올바른 경우이므로 프로세서는 정상적으로 진행된다. 이때 age_counter를 '1' 감소시켜 분기 히스토리에서 미해결 분기명령의 수가 하나 줄었음을 나타낸다. 만약 두 결과가 다르다면 분기예측실패가 검출된 경우이고, SHR의 B_{C-2}로부터 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.

3. 실험 및 성능 측정

성능 측정을 위해 슈퍼스칼라 프로세서의 사이클 수준 시뮬레이터인 SimpleScalar 3.0/PISA 툴셋[6]에서 모험적 갱신을 허용하는 gshare와 GAg 분기예측기에 본 논문에서 제안한 분기예측실패 복구 메커니즘을 적용하였다.

시뮬레이션된 프로세서의 구성에 대한 머신 파라메타는 다음과 같다. 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 데이터 캐시, 512KB L1 명령어 캐시, 1MB L2 통합 캐시를 사용하였다.

시뮬레이션에서는 8개의 SPECINT95 벤치마크 프로그램과 입력 데이터는 ref input을 사용하였고, 시뮬레이션 시간을 줄이기 위해 벤치마크 프로그램의 수행된 명령어수를 200M(million)까지로 제한하였다.

1K~8K 엔트리의 PHT를 갖는 모험적 갱신을 허용하는 gshare와 GAg 분기예측기에 제안한 복구 메커니즘을 적용하지 않았을 경우와 적용했을 경우의 예측정확도와 성능 향상(speedup)을 측정하였다. 제안한 복구 메커니즘은 간단한 하드웨어 추가로 기존의 복잡한 하드웨어를 요구하는 복구 메커니즘과 같은 예측정확도와 성능을 개선시킴을 확인할 수 있었다.

그림3은 1K, 2K, 4K, 8K엔트리의 PHT를 갖는 GAg와 gshare에 제안한 분기예측실패 복구메커니즘을 적용하기전과 적용후의 8개 벤치마크 프로그램에 대한 평균 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, gshare+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 개

선된 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.

또, PHT의 엔트리수가 증가하면 예상정확도가 약간 증가하지만 큰 변화는 없는 것으로 나타났다. 이와 같은 결과에 따라 제안된 복구메커니즘은 PHT 엔트리 수에 큰 영향을 받지 않는다는 것을 확인할 수 있었다.

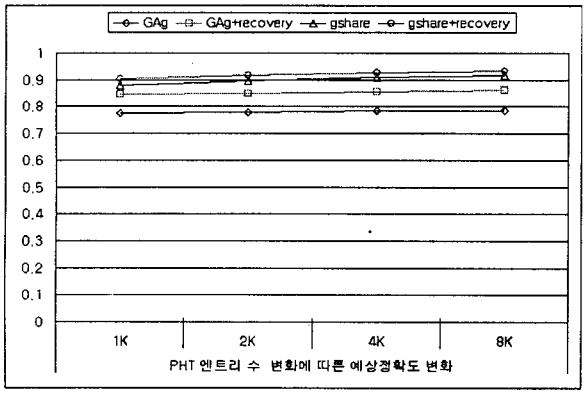

그림3 PHT 엔트리 수 변화에 따른 예상정확도 변화

그림4는 GAg와 gshare에 제안한 분기예측 실패 복구 메커니즘을 적용하기 전에 비해 적용 후의 8개 벤치마크 프로그램의 평균 성능 향상을 IPC(Instructions Per Cycle)로 나타낸 것으로 각각 1K, 2K, 4K, 8K 엔트리의 PHT를 갖는 GAg와 gshare에 제안한 분기예측 실패 복구방법을 적용하기 전과 적용 후의 IPC의 변화를 나타내고 있다.

GAg에서는 분기예측 실패 복구방법 적용 시 최소 17%(2K 엔트리 PHT), 최대 20.20%(8K 엔트리 PHT), 평균 18.08%의 IPC가 개선되었음을 보여주고 있으며, IPC 또한 예측정확도와 같이 GAg에서는 PHT 엔트리 수가 증가하면 IPC가 좀더 개선되는 것으로 나타났다. 또, gshare에서는 분기예측 실패 복구방법 적용 시 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.

PHT의 엔트리수가 증가하면 IPC도 약간 증가하지만 큰 변화는 없는 것으로 나타나서 예상정확도에서와 같이 IPC에서도 제안된 복구메커니즘이 PHT 엔트리 수에 큰 영향을 받지 않는다는 것을 확인할 수 있었다.

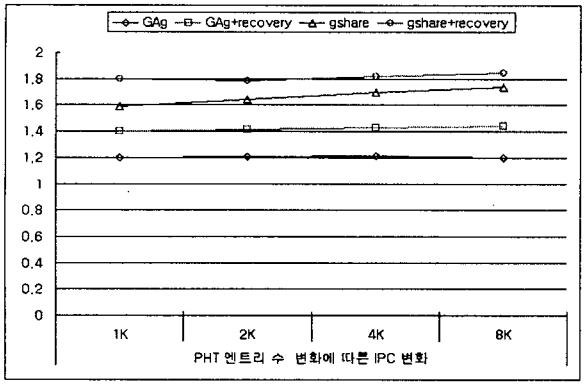

그림4 PHT 엔트리 수 변화에 따른 IPC 변화

4. 결 론

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

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.

참고문헌

[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] 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.

[3] 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.

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

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

[6] 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.