DOI: 10.3745/KIPSTA.2008.15-A.4.217

분기 정보의 추측적 사용과 효율적 복구 기법

곽 종 욱†

요 약

분기 명령어에 대한 예측 정확도는 시스템의 전체 성능 향상에 중대한 영향을 미친다. 분기 정보의 추측적 사용은 미완료 분기에 대한 히스토리 정보를 추측적으로 사용하여 분기 예측을 수행한다. 이러한 방식은 분기 명령어의 가장 최근 기록을 일관되게 사용할 수 있도록 도와주기 때문에 분기 예측의 정확도 향상에 크게 기여한다. 하지만 미완료 분기 히스토리는 올바르지 못한 정보일 수 있으며, 이런 경우 적절한 복구 기법이 필요하다. 이를 위해 본 논문에서는 분기 정보의 추측적 사용에 대한 성능 향상의 정도를 살피고, 분기 정보의 추측적 사용에 대한 필요성을 제시한다. 아울러, 분기 정보의 추측적 사용으로 인해 요구되는 적절한 복구 기법을 제안한다. 제안된 기법은 전역 분기 히스토리를 사용하는 분기 예측기와 지역 분기 히스토리를 사용하는 분기 예측기에 각각 적용 될 수 있는 방식들이다. 모의실험을 통해 본 논문에서 제안된 방식의 성능을 분석한 결과, 본 논문에서 제안된 기법이 최대 5.64%의 성능향상을 제공하였다. 아울러 프로그램 수행의 정확성을 해치지 않으면서 기존의 연구와 비교하여 90% 이상의 하드웨어 요구량의 감소를 가져왔다.

키워드 : 분기 예측, 전역 및 지역 분기 정보, 분기 히스토리, 추측적 사용, 복구 기법

Branch Prediction with Speculative History and Its Effective Recovery Method

Kwak, Jong Wook†

ABSTRACT

Branch prediction accuracy is critical for system performance in modern microprocessor architectures. The use of speculative update branch history provides substantial accuracy improvement in branch prediction. However, speculative update branch history is the information about uncommitted branch instruction and thus it may hurts program correctness, in case of miss-speculative execution. Therefore, speculative update branch history requires suitable recovery mechanisms to provide program correctness as well as performance improvement. In this paper, we propose recovery logics for speculative update branch history. The proposed solutions are recovery logics for both global history and local history. In simulation results, our solution provides performance improvement up to 5.64%. In addition, it guarantees the program correctness and almost 90% of additional hardware overhead is reduced, compared to previous works.

Keyword : Branch Prediction, Global/Local Branch History, Speculative History, Recovery Logic

1. 서 론

오늘날의 프로세서 환경은 프로그램의 실행 도중 분기 명령어가 나타날 때 이에 대한 최종적인 결과가 도출될 때까지 프로그램의 수행을 중단(stall)시키지 않는다. 그 대신, 예상되는 분기 경로의 수행을 추측적으로 실행(Speculative Execution)하여 그에 해당하는 다음 명령어를 인출(fetch)해 오는 방식을 취한다[1]. 이러한 수행 환경은 분기 예측 실패로 인한 예측 실패 비용 (Misprediction Penalty)을 증가 시키는 원인으로 작용하기 때문에 보다 더 정확한 분기 예측 기법이 요구된다[2][3]. 이와 관련하여, Sprangle 등은 분기 예측 실패율이 전체 시스템 상에서 단일 구성 요소로서는 가장 큰 시스템 성능 제약 요소 가운데 하나라는 사실을 보이고 있다[4].

일반적으로 분기 히스토리(Branch History)를 사용하여 분기 명령어들의 실행 결과를 예측하고자 할 경우, 현재 그 결과를 예측하고자 하는 분기 명령어와 실행 순서 상 가까운 분기 명령어들 일수록 예측 결과에 보다 더 많은 영향을 미치게 된다. 이러한 관점에서 접근하자면 분기 히스토리의 추측적 사용이 긍정적일 수 있다. 하지만, 추측적으로 사용된 분기 히스토리(Speculative Branch History)는 정확성이 결여된 정보로서, 현재 수행중인 미완료 분기에 대한 실행 결과이다. 다시 말해, 이는 최종적으로 수행 종료된 분기 명

† 정 회 원 : 영남대학교 전자정보공학부 전임강사
논문접수 : 2008년 4월 24일
수 정 일 : 2008년 6월 9일
심사완료 : 2008년 6월 9일

령어의 정확한 분기 히스토리를 사용하는 것이 아니라, 새로운 분기 명령어의 예측을 위해 이전 분기 명령어들의 “예측 결과”를 이용한다는 것이다.

따라서 이와 같은 예측 정보들이 경우에 따라서 올바르지 못한 것일 수 있다. 또한, 실제 프로그램 수행 과정에서 추측적으로 사용된 히스토리 정보가 올바르지 못하다고 밝혀질 경우, 이에 대한 적절한 복구 기법이 필요하다.

본 논문에서는 이와 같은 분기 정보의 추측적 사용에 대한 성능 향상의 정도를 살피고, 분기 정보의 추측적 사용에 대한 필요성을 제시한다. 아울러, 분기 정보의 추측적 사용으로 인해 요구되는 적절한 복구 기법을 제안한다. 제안된 기법은 전역 분기 히스토리(Global Branch History)를 사용하는 분기 예측기와 지역 분기 히스토리(Local Branch History)를 사용하는 분기 예측기에 각각 적용 될 수 있는 방식들이다

이하 본 논문의 구성은 다음과 같다. 2절에서는 관련연구 및 배경지식에 대해 설명 한다. 3절에서는 전역 및 지역 분기 정보의 추측적 활용 모듈 및 그에 대한 복구 방식을 소개한다. 그리고 4절에서는 모의실험을 통해 본 논문에서 제안된 방식과 기존 방식과의 성능상의 차이를 비교 분석하며, 5절에서 결론을 맺는다.

2. 관련 연구 및 배경 지식

2.1 관련 연구

Yeh와 Patt에 의해 제안된 이단계 적응형 분기 예측 기법에 대한 소개 이후로, 분기 명령어의 히스토리는 오늘날의 분기 예측기에 있어서 중요한 입력 요소 중 하나로 자리 잡았다[5][6]. 이와 같은 분기 히스토리는 이를 전체 분기 명령어들의 동적인 수행 경로 상에서 파악하느냐 혹은 각각의 분기 명령어별로 개별적으로 파악하느냐에 따라 전역 분기 히스토리와 지역 분기 히스토리로 구분되었으며, 오늘날까지 각각을 사용한 다양한 형태의 분기 예측기들이 제시되어 왔다.

또한 여기서 한걸음 더 나아가, 이와 같은 분기 정보를 어떻게 활용하는가에 대한 연구도 진행되었다. 그 중 대표적인 것이 분기 정보의 추측적 사용이다. 이와 관련된 연구들은 다음과 같다.

Hao 등은 이단계 적응형 분기 예측기에서 히스토리의 추측적 사용에 대한 필요성을 언급했다. 아울러, 분기 히스토리를 추측적으로 사용하지 못할 경우, 히스토리 저장 레지스터에 가장 최근 분기 명령어에 대한 이전 기록이 반영되지 않을 수 있다는 사실을 관찰하였으며 이에 대한 분기 예측 정확도의 차이를 분석하였다[7].

Talcott 등은 RS/6000 프로세서 상에서 미완료 분기 히스토리의 사용에 대한 성능상의 이득을 소개하였다. 저자들은 그들의 연구에서, skipped 기법과 substituted 기법이라 명명된 미완료 분기 히스토리를 처리하는 두가지 개념을 제시하였다[8].

Skadron 등은 분기 정보의 추측적 사용에 대한 성능상의 이득을 정량적(quantitative) 방법을 통해 다양하게 제시하였다. 그들은 지역 분기 히스토리에 있어서 추측적 분기 정보의 사용이 제공하는 성능상의 이득을 실험적 결과를 통해 제시하였으며, 이때 필요한 다양한 하드웨어 모델을 소개하였다[9].

한편, Alpha EV8 분기 예측기에 있어서는, 최신의 분기 예측 기술들이 많이 접목되어 구현되었다. 해당 프로세서의 분기 예측기 역시 현재 수행 중인 분기 명령어(In-flight Branch)에 대한 미완료 분기 히스토리를 추측적으로 활용한다[10].

2.2 배경 지식

추측적 분기 히스토리의 사용에 대한 필요성을 분석적으로 설명하기 위해, (그림 1)과 (그림 2)에 제시된 관련 예제 코드를 살펴보자.

(그림 1)에 나타난 예제 코드의 경우, 마지막 분기 명령어인 Br3의 분기 방향은 Br1 혹은 Br2의 결과에 강하게 연관되어 있다는 것을 알 수 있다. 즉 Br1과 Br2의 분기 결과를 알고 있으면, Br3의 분기 방향은 쉽게 예측이 가능하다. 이처럼 분기 명령어들은 경우에 따라서 다른 분기 명령어들 사이에 강한 상호 연관성(correlation)을 가질 수 있으며, 위의 샘플 코드와 같은 경우를 분기 방향 연관성(Direction Correlation)이라고 한다[11].

반면, (그림 2)에 제시된 예제 코드에서는 BrC의 분기 방향이 BrD와 직접적인 연관은 없다고 할 수 있다. 즉 분기 방향 연관성은 존재하지 않는다. 하지만, 프로그램의 수행이 BrC에 도달하였을 경우, 마지막 분기 명령어인 BrD의 조건이 만족하리라는 것을 알 수 있다. 이처럼 프로그램 수행 경로 상에서 존재하는 또 다른 형태의 상호 연관성을 수행 경로 연관성(In-Path Correlation)이라고 한다[11].

```
if (condA)                    Br1
...
if (condB)                    Br2
...
if (condA AND condB)          Br3
...
```

(그림 1) 분기 방향 연관성 예제

```
if (NOT(cond1))               BrA
...
else if (NOT(cond2))          BrB
...
else if (cond3)               BrC
...

if (cond1 AND cond2)          BrD
...
```

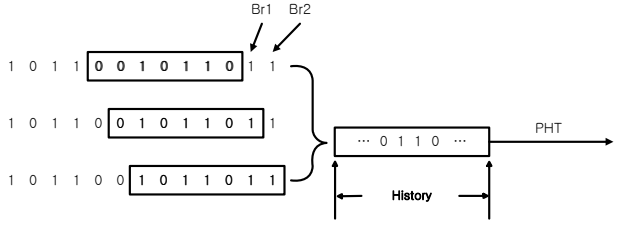
(그림 2) 수행 경로 연관성 예제

이처럼, 위의 두 예제 코드에서 보여주듯이, 현재 예측하고자 하는 분기 명령어와 실행 순서상 가까운 최근의 분기 명령어들일수록 보다 더 강한 상호 연관성을 가지게 되며, 이는 곧 현재 그 결과를 예측하고자 하는 분기 명령어에 대해 결정적인 영향을 미칠 수 있다는 의미이다.

한편, 오늘날의 마이크로프로세서는 파이프라인의 단계수가 늘어나고 단위 시간당 제공되는 명령어들의 개수가 증가해 나가는 추세를 보인다. 이와 같은 상황에서는 각각의 파이프라인 단계 내부에 현재 수행 중인 미완료 분기 명령어(In-Flight Branch)들이 더욱 더 많이 존재하게 된다. 이럴 경우 미완료 분기 명령어들에 대한 분기 정보는 현재 분기 방향을 예측하고자 하는 분기 명령어의 수행 단계에 있어서 그 결과가 아직 반영되지 못할 수 있다는 것을 의미한다. 이는 앞선 두 예제 코드에서 보여 지듯이, Br3과 강한 연관을 가진 Br1 혹은 Br2, 그리고 BrD와 강한 연관을 가진 BrC와 같은 분기 명령어들의 분기 정보가 정작 Br3과 BrD의 예측 단계에서는 미처 그 결과가 반영되지 못함을 의미한다. 그리고 이는 분기 예측의 정확도를 감소시키는 결정적 요인 중 하나로 작용하며, 여기에 추측적 분기 정보의 사용에 대한 필요성이 있다고 할 수 있다.

그 외에도, 분기 히스토리를 추측적으로 사용하지 못할 경우는 Hao 등에 의해 설명된 시간 문제(Timing Problem)가 발생한다. (그림 3)에 이와 관련된 문제 상황이 묘사되어 있다. 이와 같은 시간 문제는 동일한 분기 명령어에 대해 반복적인 수행이 일어난다 하더라도, 매번 똑같은 상황 하에서 그 수행이 재현되지 않을 수 있음을 의미한다. 즉, 동일 분기 명령어에 대한 분기 예측 시에 매번 서로 다른 입력 요소의 조합으로 분기 예측이 이루어 질 수 있다는 것이다. 그리고 이와 같은 동일 상황을 저해하는 요인으로, 캐쉬 메모리의 접근 실패(Cache Miss), 다른 이전 분기 명령어의 분기 예측 성공 여부와 그에 종속된 상호 연관성, 그리고 명령 윈도우(Instruction Window)등과 같은 시스템 자원의 뜻하지 않은 부족 등을 그 예로 들 수 있다[7].

분기 기록들의 시간 문제를 (그림 1)에 제시된 예제 코드를 통해 부연 설명하자면 다음과 같다. Br3의 예측 결과는 Br1과 Br2의 분기 방향에 결정적이다. 하지만 앞서 설명한 여러 원인들에 의해 (그림 3)에서 나타나듯이, 매번 서로 다른 분기 히스토리들이 Br3의 예측을 위해 사용될 수 있다. 이는 분기 예측 테이블(Pattern History Table)의 입력 요소에 대한 비일관적 사용을 야기하게 되며, 분기 예측 테이블의 가명현상(aliasing)을 초래하는 원인 중 하나로 작용한다.



(그림 3) 분기 기록들의 시간 문제
(Timing Problem)

[12][13]. 이러한 가명현상은 분기 예측의 정확도를 감소시키는 결정적인 요인으로 알려져 있으며, 특별히 가명현상의 해결에 초점을 맞춘 연구와 노력들이 다양하게 제시되고 있다[14][15].

이처럼 분기 히스토리의 추측적 사용은 가장 연관성이 높은 분기 명령어들로 구성되는 최근 분기 히스토리를 일관성 있게 사용할 수 있도록 도와줄 뿐만 아니라, 분기 예측 테이블의 가명현상을 감소시키는 역할도 수행한다. 이 같은 원인과 특징들에 의해 분기 정보의 추측적 사용은 분기 예측의 정확도 향상에 크기 기여한다.

3. 분기 정보의 추측적 사용과 복구 기법

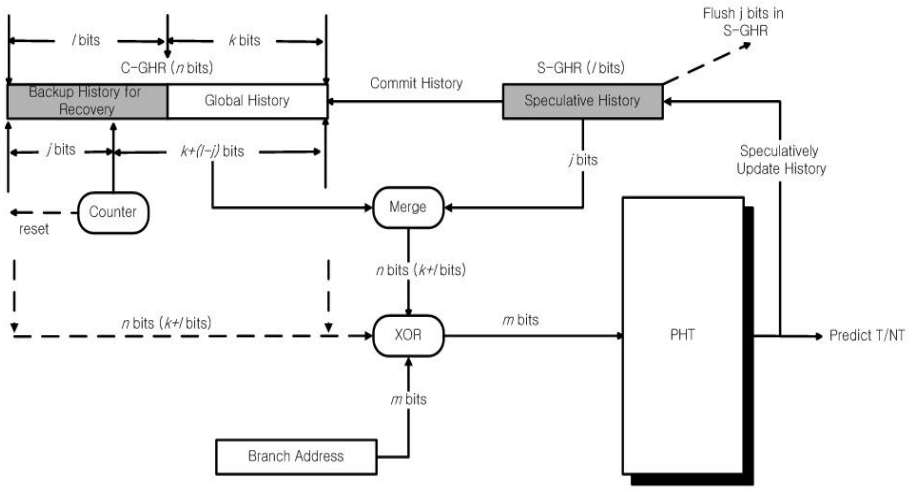
분기 정보의 추측적 사용은 전술 한 바와 같이 분기 예측의 정확도 향상에 기여한다. 하지만 이와 같은 추측적 정보는 수행이 종료된 분기 명령어들의 최종적인 정보를 의미하는 것이 아니다. 다시 말해, 올바른 수행 결과를 이용하여 다음의 예측을 수행하는 것이 아니라, 오류의 가능성이 있는 예측 결과를 이용하여 다음의 예측을 수행하는 것이다. 따라서 잘못된 분기 정보를 사용했을 경우, 이에 대한 적절한 복구 기법이 필요하다. 이 절에서는 전역 분기 히스토리와 지역 분기 히스토리에 대해, 각각의 경우에 대한 추측적 정보의 사용 방법과 그에 대한 적절한 복구 기법을 제시한다.

3.1 전역 히스토리의 추측적 사용과 복구

전역 분기 히스토리를 사용하는 분기 예측기로 GAg[5][6]와 gshare[16] 등을 그 예로 들 수 있다. 이들처럼 전역 분기 히스토리를 사용하는 분기 예측기들은 분기 명령어들의 전역 히스토리를 보관하기 위해서 쉬프트 레지스터(Shift Register) 형태로 구현된 전역 히스토리 레지스터(GHR)를 사용하여 분기 정보를 저장 관리한다. 본 논문에서는 많은 다른 연구들에서 보이듯이, 성능 향상의 비교 대상으로 주로 사용되는 gshare 예측기를 이용한 전역 분기 히스토리의 추측적 사용 및 복구 모델을 제시한다.

(그림 4)는 전역 분기 히스토리를 사용하는 분기 예측 모듈에서, 추측적 분기 정보의 사용과 그로 인해 요구되는 복구 모듈이다. 기존의 방식은 분기 히스토리를 저장하는 레지스터로 GHR을 하나만 사용하였지만, 본 논문에서는 분기 정보의 추측적 사용을 위해 추측적 정보만을 별도로 저장하는 레지스터를 추가로 구성한다. 이를 구분하기 위해 기존의 수행 완료된 분기 히스토리를 가지도록 설계된 GHR을 C-GHR(Committed GHR)로 명명하며, 추측적 정보만을 별도로 저장하기 위해 본 논문에서 추가적으로 사용하는 GHR을 S-GHR (Speculative GHR)로 명명하여 이를 구분한다.

기존의 gshare 예측기는 전역 분기 히스토리와 분기 명령어의 주소값(Program Counter)을 입력으로 사용한다. 그리고 이들은 인덱스 함수인 xor 함수의 출력 결과로 PHT에 접근하게 된다. 하지만 본 논문에서는 분기 히스토리의 추측적 사용을 위해 (그림 4)에서와 같이 C-GHR과 S-GHR의



(그림 4) 전역 히스토리의 추측적 사용과 복구 기법

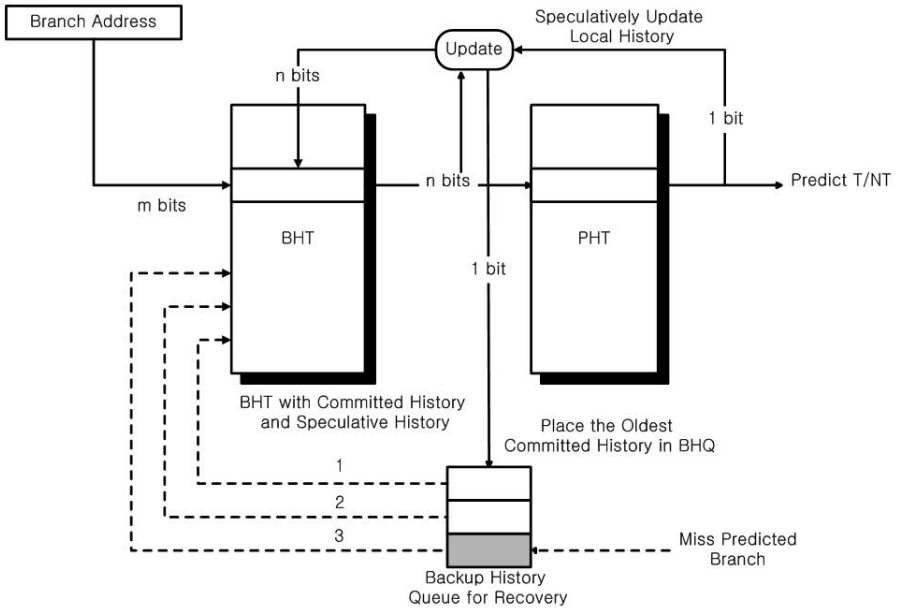
값을 서로 병합(merge)하여 사용한다. 그리고 그 결과를 분기 명령어의 주소값과 함께 인덱스 함수에 대입한다. 다시 말해, 현재의 분기 예측에 필요로 하는 S-GHR의 비트 수만큼 기존의 오래된 순서대로의 히스토리들을 제외한 나머지와 추가적으로 병합하여 사용하게 된다. 이때, S-GHR의 추가적인 사용으로 인해 병합과정에서 제외된 S-GHR 비트수만큼의 오래된 순서대로의 C-GHR 정보는 추후 발생할 수 있는 복구 작업을 백업(Backup History for Recovery)된다. 이를 위해 C-GHR은 다시 GH(Global History) 부분과 BHR(Backup History for Recovery) 부분으로 재차 구분하여 명명한다.

초기값으로 n 비트의 분기 히스토리와 m 비트의 분기 주소값($m \geq n$)을 사용한다면, C-GHR은 n 비트로 구성되게 된다. 이 때, n 비트의 크기는 시스템이 사용하는 2^n 크기의 PHT와 연관되어 결정된다. 그리고 추측적 정보의 추가적 사용을 위해 S-GHR은 l 비트로 구성한다. 이는 시스템에서 사용할 수 있는 추측적 분기 정보가 최대 l 개임을 의미한다. 추측적 분기 정보의 사용을 위한 적절한 l 비트의 크기는 4장의 실험 결과 분석을 통해 추가적으로 제시한다.

한편, S-GHR의 크기만큼 n 비트 C-GHR 중 l 비트를 복구를 위해서 추가적으로 사용되는 최대 BHR 영역으로 구분한다. 즉, 분기 히스토리의 최대 백업량은 추측적 정보의 최대 사용량과 동일하다. 결과적으로 C-GHR은 l 비트의 BHR과 이를 제외한 나머지 k 비트의 GH로 구성($n = k + l$)된다. 그리고 최대 l 비트 중에서 현재 사용되는 추측적 정보의 양을 j 비트라고 한다면($l \geq j$), 이는 현재 C-GHR에서 S-GHR과의 병합에서 제외될 BHR의 비트수를 의미하기도 한다. 이 같은 j 비트 정보는 카운터에 의해 표시된다. 즉 gshare 예측기를 추측적 정보와 함께 사용할 경우, $k+(l-j)$ 비트 만큼의 C-GHR 정보를 이용하여 j 비트 S-GHR과 병합하게 되며, 그 결과로 얻어진 n 비트($n = k + l$)가 분기 명령어의 주소값 m 비트($m \geq n$)와 함께 입력 함수에 대입된다. 끝으로, 이와 같은 방식으로 S-GHR에서 사용된 추측적 분기 정보는, 분기 명령어의 실행 완료 시점에서 올바른 예측이라고 결론이 났을 경우, C-GHR로 옮겨지게 된다. 이상의 내용이 (그림 4)에 묘사되어 있다.

만약 분기 예측에 사용된 추측적 정보가 올바르지 못한 경우는 적절한 방법을 사용하여 올바른 정보로 복구하여야 한다. (그림 4)의 점선이 복구 관련 동작을 묘사하고 있다. 비순서 실행(Out-of-Order Execution)이 지원되는 프로세서라 하더라도, 순차적으로 실행 종료(In-Order Commit)가 이루어진다. 이러한 경우, 가장 이전 예측이 올바르지 못하다고 결론이 날 것임으로, 잘못된 예측을 이용하여 예측한 그 뒤의 예측들도 모두 틀린 결과를 초래하게 된다. 따라서 S-GHR의 모든 비트를 제거(flush)하고 카운터를 리셋한다. 그 후, 기존의 GH와 복구를 위해 백업된 BHR을 이용하여 분기 예측을 다시 수행하게 된다. 즉, 기존의 gshare 예측기에서와 동일하게 추측적 분기 정보 없이 C-GHR만을 이용하여 분기 명령어의 주소값과 함께 인덱스 함수에 대입된다. 그리고 그 이후부터 추측적 정보의 사용이 다시 하나씩 증가하게 된다.

본 논문에서 제시된 이와 같은 기법은 큐(Queue)를 이용하여 복구 단계에서 필요한 정보를 백업시킨 이전 연구와 비교하여 볼 때, 평균 90% 이상의 하드웨어 복잡도의 절감 효과를 가진다[9][10]. 기존의 연구에서 필요로 하는 추가적인 하드웨어 복잡도는 (전역 분기 히스토리 비트의 수) × (허용 가능한 In-Flight 분기 명령어들을 저장할 큐 엔트리 수)이다. 하지만, 본 논문에서 제시된 방식의 하드웨어 요구량은 S-GHR의 저장을 위한 l 비트(In-Flight 분기 명령어들의 수)와 이를 위한 $\overline{\log_2 l}$ 비트 크기의 카운터이다. 예를 들어, 2 비트 카운터를 사용하는 4K개 엔트리의 PHT에서 최대 20개의 In-Flight 분기를 허용할 경우, 기존의 방식은 전체 8K 비트 (4K × 2비트) PHT에서 추가적인 240 비트 (12 비트 × 20)의 하드웨어가 요구된다. 이에 반해 본 논문에서 제시된 기법은 25 비트(20 비트 + $\overline{\log_2 20}$)의 추가 하드웨어 요구량을 나타낸다. 대략 하드웨어 복잡도를 10% 수



(그림 5) 지역 히스토리의 추측적 사용과 복구 기법 (이전기록 기반)

준으로 감소시킨 결과이다.

3.2 지역 히스토리의 추측적 사용과 복구

전역 분기 히스토리와 더불어, 지역 분기 히스토리를 사용하는 분기 예측기도 존재한다. 지역 히스토리를 사용할 경우 전역 히스토리를 사용하는 경우와 비교하여 다음과 같은 장점을 취할 수 있다.

첫째, 비교적 자주 등장하지 않는 분기 명령어로, 제한된 전역 히스토리 길이 내에서 그 동작 패턴을 완전히 파악하기 힘들 정도로 드물게 수행되는 분기 명령어가 있을 수 있다. 가장 바깥쪽의 커다란 루프 형태의 분기 명령어가 이에 해당되는 예라 할 수 있다. 이러한 경우는 전역 히스토리에 비해 지역 히스토리가 그 행동 패턴을 명확히 파악 할 수 있다. 일반적으로 전역 히스토리를 사용할 경우 더 우수한 성능 향상을 보이지만, 실제로 특정 응용 프로그램에 있어서는 전역 히스토리에 비해 지역 히스토리를 사용하였을 경우에 더 우수한 결과를 보인다[17].

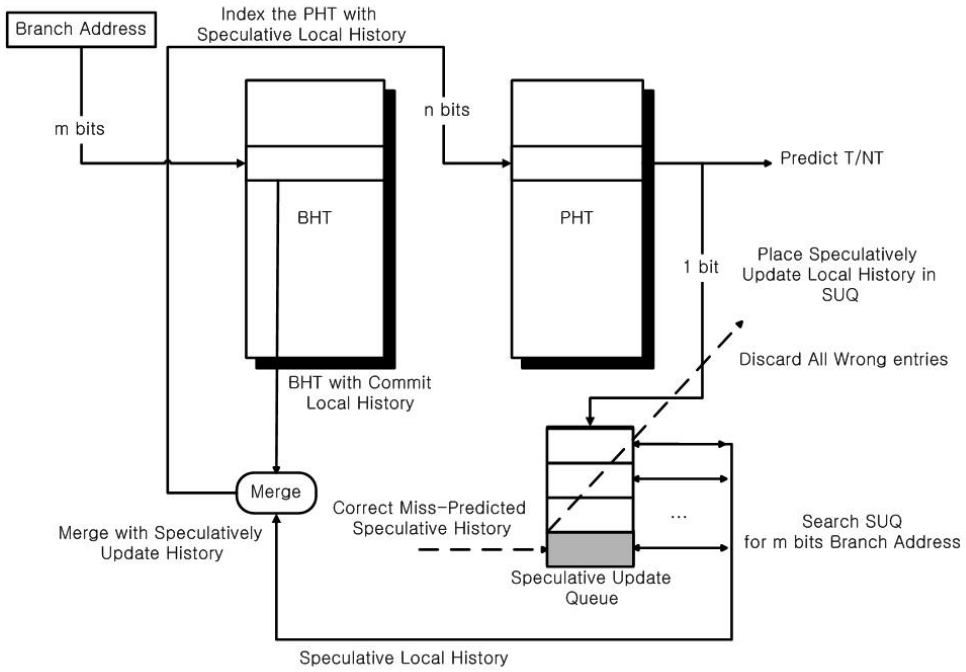
둘째, 전역 히스토리를 사용하는 기법에 있어서는 특정한 단일 분기 명령어의 잘못된 예측(partial flaw)이 전체 GHR 상에서 잘못된 값의 사용을 연쇄적으로 초래할 수 있게 된다. 즉, 하나의 잘못된 예측값이 GHR 상에 계속 존재하게 되기 때문에, 이후의 모든 분기 명령어들은 이 같은 잘못된 값을 계속적으로 오용하게 된다. 이에 비해 지역 히스토리를 사용하는 기법은, 단일 분기 명령어의 잘못된 예측이라 하더라도 그 영향 범위가 해당 분기 명령어에 국한되어 발생하기 때문에 문제의 범위를 국소화(localization) 시킬 수 있다는 장점이 있다.

이와 같은 장점에서 보여주듯이, 지역 히스토리를 사용하는 분기 예측기의 필요성은 분명하다. 실제로 지역 히스토리를 사용하는 분기 예측기들은 지역 분기 히스토리를 단독으로 사용하는 경우는 드물며, 토너먼트 예측기(Tournament Predictor)와 같은 예에서 보여주듯이, 주로 다른 여러 예측기들과 함께 결합된 형태(Hybrid Predictor)로 사용된다[18][19]. 이 절에서는, 지역 히스토리를 사용하는 분기 예측기에서 분기 정보의 추측적 사용법 대해 설명하며, 이때 추가적으로 필요한 복구 모듈과 복구 방법에 대해서 설명한다.

(그림 5)는 지역 분기 히스토리를 사용하는 분기 예측기에서 추측적 분기 정보의 사용과 이때 필요한 복구 모듈이다. 일반적으로 지역 히스토리를 사용하는 경우는, 이전기록 기반과 이후기록 기반의 두 가지 형태의 추측적 정보 사용 방식이 있을 수 있다[20]. (그림 5)는 이전기록 기반에 근거한 방식으로, 본 논문에서 제안하는 지역 히스토리의 추측적 사용과 이에 대한 복구 모델이다.

(그림 5)에서 보여 지듯이, 제안된 기법은 분기 예측이 이루어질 때, 새롭게 생성된 추측적 분기 정보를 BHT (Branch History Table)에 바로 반영하여 그 값을 갱신한다. 그리고 이때 가장 먼저 생성된 가장 이전의 1 비트 지역 히스토리는 BHQ (Backup History Queue for Recovery)에 저장되게 된다. 이 값은, 추측적 정보가 잘못 사용되었다고 판단될 경우 복구 과정에서 사용된다. 결과적으로 BHT는 완료된 분기 정보와 함께 추측적 분기 정보를 함께 보관하고 있는 형태가 된다. BHT는 이 같은 n 비트의 값을 이용하여 분기 예측 단계에서 PHT로 접근하게 된다. 이와 같은 방식으로 사용된 추측적 분기 정보는, 분기 명령어의 실행 완료 시점에서 올바른 예측이라고 결론이 났을 경우, 최종적으로 BHQ의 해당 엔트리를 삭제하게 된다.

복구 기법은 다음과 같다. 만약, 추측적 분기 정보가 잘못 사용 되었다고 결론이 날 경우 (그림 5)의 점선에서 나타나는 바와 같이 복구 과정이 수행된다. 우선, 잘못 예측된 분기 명령어로 인해 제거된 지역 히스토리를 BHQ에서 찾은



(그림 6) 지역 히스토리의 추측적 사용과 복구 기법 (이후기록 기반)

뒤, 큐의 해당 엔트리에서부터 역순으로 BHT의 값을 복원한다. 일단 복구가 되게 되면, BHT의 해당 엔트리들은 추측적 분기 정보 없이 완료된 올바른 분기 정보만을 포함하게 되며, 그 이후부터 추측적 정보의 사용이 다시 하나씩 증가하게 된다.

한편, (그림 6)은 이후기록 기반에 근거하여 제안된 방식이다. (그림 6)의 방식은, (그림 5)와는 반대로, 새롭게 생성된 추측적 분기 정보가 BHT에 반영되지 않고, 별도의 SUQ(Speculative Update Queue)에 저장되게 된다. 따라서 이후기록 기반 방식에서의 BHT는 완료된 분기 명령어들의 히스토리만을 보관하고 있는 형태가 된다.

분기 예측 방법은 다음과 같다. 새로운 분기 명령어의 예측 단계에서, (그림 6)에 제시된 방식은 완료된 이전기록을 보관하고 있는 BHT와 추측적 분기 정보를 포함하고 있는 SUQ의 해당 엔트리를 서로 병합하게 된다. 이처럼 분기 예측 단계마다 추측적 정보를 생성한 후, PHT에 접근이 이루어진다. 끝으로 이와 같은 방식으로 사용된 추측적 분기 정보는, 분기 명령어의 실행 완료 시점에서 올바른 예측이라고 결론이 났을 경우, SUQ의 해당 엔트리를 삭제하고 BHT는 기존의 방식과 동일하게 새로운 값으로 갱신된다.

복구 기법은 다음과 같다. 만약, 추측적 분기 정보가 잘못 사용 되었다고 판단될 경우 (그림 6)의 점선에서 나타나는 바와 같이 복구 과정이 진행된다. (그림 5)와 비교하여, 이후기록 기반의 BHT는 어떠한 추측적 정보도 보관하고 있지 않다. 따라서, 단순히 추측적 정보를 포함하고 있는 SUQ 엔트리를 찾아, 그 이후의 엔트리들을 삭제(flush)하는 것만으로 복구가 완료된다.

(그림 5)와 (그림 6)에서 제시된 두 가지 기법의 특징을 비교하면 다음과 같다. (그림 6)에 나타난 기법은, 상대적으로 (그림 5)의 기법과 비교하여, 추측적 정보 생성을 위한 별도의 병합 단계가 필요하다. 이는 분기 예측에 있어서 일반 경우(common case)라 할 수 있는 분기 예측 시간(Prediction Time)을 다소간 증가시키는 경향이 있다. 하지만 추측적 정보의 잘못된 사용으로 인한 복구가 상대적으로 간단하다. 즉, 큐의 삭제(queue flush)로 간단히 복구가 이루어진다. 이에 비해 (그림 5)에 나타난 기법은, 분기 예측 시간이 상대적으로 빠르다는 장점이 있으나 (그림 6)과 비교하여 복구 시에 단점을 가진다. 즉, 복구 과정에서 큐의 각 엔트리에 대한 단계별 복구 절차가 필요하게 된다. 하지만, 시스템의 분기 처리 비용이 클 경우 이 같은 단계별 복구 절차는 분기 예측 실패 비용(Miss-Prediction Penalty)과 중첩되어 상쇄될 수 있다.

따라서, 이전기록 기반 방식은 높은 분기 예측 정확도를 보이는 응용 프로그램들에서와 큰 분기 예측 실패 비용을 요구하는 시스템의 경우에 적용이 유리하며, 이후기록 기반 방식은 상대적으로 낮은 분기 예측 정확도를 보이는 응용 프로그램들과 작은 분기 예측 실패 비용을 가지는 시스템의 경우 보다 더 유리하게 적용될 수 있을 것이다.

4. 성능 평가

이 절에서는, 본 논문에서 제시된 기법의 성능 평가를 수행한다. 적절한 복구 기법이 활용되는 분기 정보의 추측적 사용이 제공하는 성능상의 향상 정도를 살피고, 추측적 분기 정보의 사용에 대한 필요성을 설명한다. 또한 복구 예비 비트의 사용 완화 기법과 평균 In-Flight 분기 명령어들의 개수에 대한 분석도 함께 수행한다.

4.1 실험 환경 및 벤치마크 프로그램

본 논문에서의 모의실험은 구동 기반 시뮬레이터인 SimpleScalar로 진행되었다[21]. 이벤트 구동형 시뮬레이터(Event-Driven Simulator)인 SimpleScalar는 빠른 모의 실험과 높은 정확도의 결과를 보장하는 강력한 모의실험 환경으로, 비순서 실행(Out-of-Order Issue), 비중단 캐쉬(Non-Blocking Cache), 추측적 실행(Speculative Execution)과 같은 최신 프로세서 기술을 지원하며, 아울러 다양한 분기 예측 기법의 사용을 가능하게 한다. <표 1>에 본 논문에서 사용된 실험 환경이 제시되어 있다.

한편, 모의실험에 사용된 벤치마크 프로그램은 SPEC(Standard Performance Evaluation Corporation)에서 제공하는 CPU 성능 측정 프로그램인 SPEC CINT2000 프로그램들이다. 특히, 본 논문에서는 이 가운데 무작위로 선별된 정수형 프로그램들을 활용한다. 일반적으로 SPEC에서 제공하는 CPU 성능 평가 프로그램은 정수형 프로그램인 CINT와 실수형 프로그램인 CFP로 구분된다. 이 가운데 CFP 프로그램들은 과학 계산 형태의 응용 프로그램이 주를 이루며, 이들은 매우 정형화되어 있는 코드 형태를 가진다. 이 같은 경우, 분기 예측의 정확도가 매우 높게 나타나기 때문에 분기 예측의 개선으로 인한 성능 향상의 정도를 효율적으로 관찰하기에 곤란하다. 따라서 CFP 프로그램들은 분기 예측과 관련된 연구에서 일반적으로 제외된다[22].

〈표 1〉 모의 실험 인자

Parameter	Value
Fetch Queue	4 entries
Fetch, Decode Width	4 instructions
ROB entries	16entries
LSQ entries	8entries
Functional Units(integer)	4 ALUs, 1 Mult/Div
Functional Units(floating point)	4 ALUs, 1 Mult/Div
Instruction TLB	64(16 × 4-way)entries, 4K pages, 30 cycle miss
Data TLB	128(32 × 4-way)entries, 4K pages, 30 cycle miss
Predictor Style	gshare
BTB entries	2048(512 × 4-way) entries
RAS entries	8 entries
Extra Branch Miss-prediction Penalty	3 cycles
L1 I-Cache	16 KB, direct map, 32B line, 1 cycle
L1 D-Cache	16 KB, 4-way, 32B line, 1 cycle
L2 Cache(unified)	256 KB, 4-way, 64B line, 6 cycles
Memory Latency	first_chunk=18 cycles, inter_chunk=2 cycles

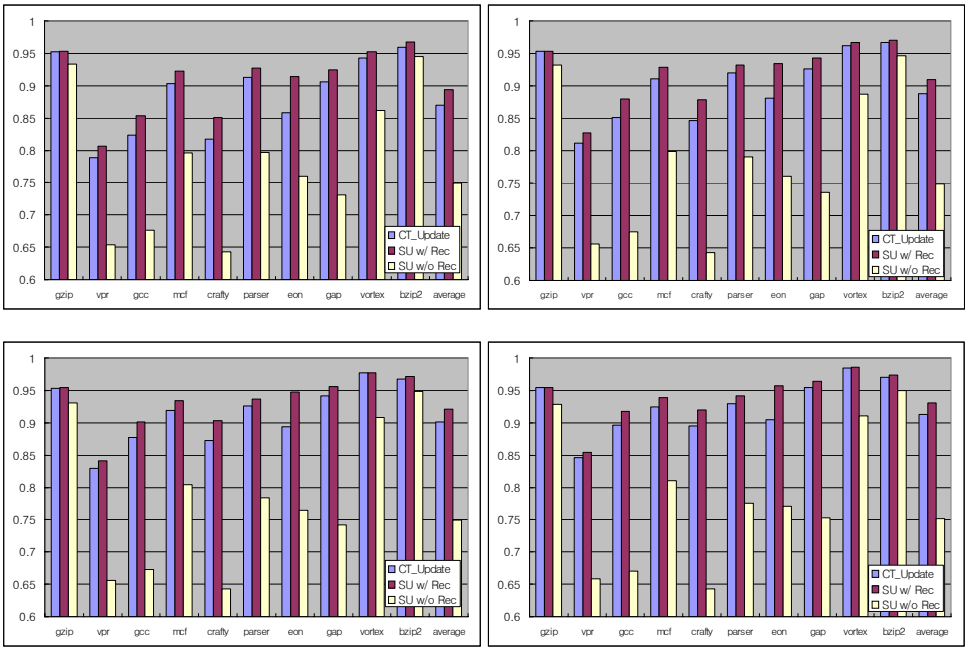
4.2 실험 결과

(그림 7)은 (그림 4)에 나타난 방식의 실험결과로, 추측적 분기 히스토리를 사용하며 이에 대한 복구 기법을 적용 시켰을 경우에 대한 분기 예측 정확도이다. 주어진 실험에서 PHT는 제시된 실험 결과의 순서대로 각각 1K, 2K, 4K 그리고 8K의 크기를 가진다. 실험의 비교 대상은 다음과 같다. CT_Update(Commit Time Update)는 기존의 방식으로, 추측적 분기 히스토리를 사용하지 않고 분기 명령어가 정상적으로 Commit 되었을 경우 그 결과를 PHT에 반영하는 방식이다. SU w/ Rec(Speculative Update with Recovery)는 본 논문에서 제시된 추측적 기법의 사용과 함께 이에 대한 적절한 복구 기법을 활용하는 방식이다. 끝으로 SU w/o Rec(Speculative Update without Recovery)는 추측적 분기 정보를 사용하지만, 본 논문에서 제안된 방식과 같은 적절한 복구 기법을 활용하지 않는 방식을 뜻한다.

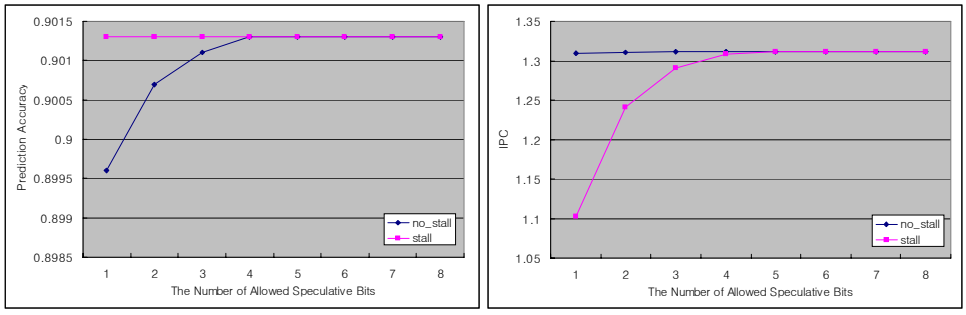
(그림 7)에서 보이듯이, 적절한 복구 기법이 제공된 추측적 분기 정보의 사용은 분기 예측의 정확도 측면에서 평균 3%의 성능 향상을 가져왔으며, 응용에 따라서는 *252.eon*의 경우 최대 5.64%의 정확도 향상을 보였다. 이 같은 수치는 분기 예측의 정확도의 향상과 관련하여, 최근 발표되는 논문에서 제시된 성능 향상의 정도와 비교해 볼 때 비교적 큰 수치이다. 즉, 새로운 분기 예측기의 제안도 분기 예측의 성능 향상을 위한 주된 연구 방법이 될 수 있겠지만, 주어진 예측기 내에서 추측적 분기 정보를 적절한 복구 기법과 함께 효율적으로 활용하는 방법 역시 시스템 성능 향상의 또 다른 방법이 될 수 있음을 보여 준다. 끝으로 SU w/o Rec는 실험 결과에서 알 수 있듯이 그 결과가 좋지 못하다. SU w/o Rec는 모든 크기의 PHT와 모든 응용 프로그램에 있어서 좋지 못한 성능을 보였다. 또한, SU w/o Rec는 CT_Update보다도 오히려 성능이 좋지 않다는 것을 확인 할 수 있다. 이 같은 결과는, 분기 정보의 추측적 사용이 가장 최근 분기 명령어들의 상호 연관성을 일관되게 반영할 수 있다는 측면에서 장점이 될 수 있지만, 적절한 복구 기법이 제공되지 않는 추측적 분기 정보의 사용은 무의미 하다는 것을 보여준다. 본 논문에서 제시된 복구 기법의 중요성을 강조해 주는 결과라 하겠다.

한편, (그림 8)은 (그림 4)에서 제시된 방식의 변형으로, *186.gcc*의 실험 예를 보인 것이다. (그림 4)에서 제시된 기법은, 잘못 사용된 추측적 정보의 복원을 위해 BHR을 사용하고 이때 BHR의 최대 크기는 S-GHR의 최대 크기와 동일하다. 또한 (그림 4)에서는 S-GHR의 모든 엔트리가 다 사용되었을 경우, 더 이상 추측적 정보의 사용 증가를 허용하지 않는다. 다시 말해, 파이프라인 상에서 수행 중단(stall)이 일어난다. (그림 8)의 실험 결과는, 이러한 제약을 완화시켜 S-GHR의 모든 엔트리가 다 사용 되었을 경우라 하더라도, 추측적 정보의 추가적 사용을 중단시키지 않고 BHR의 가장 오래된 부분을 계속 버려 나가면서 중단 없이(no_stall) 수행을 계속 이어 나가게 하였을 경우에 대한 실험 결과이다. 가령, x 축에 있어서 5의 경우는 최대 허용 가능한 추측적 분기 정보의 개수가 5 임을 의미하며, 6번째부터 추가적으로 발생하는 추측적 분기 정보의 사용 시, 파이프라인 상에서 수행 중단을 발생시키는 경우가 stall이며, 그렇지 않는 경우가 no_stall이다.

(그림 8)에 주어진 실험 결과는, 분기 예측 정확도와 IPC 측면에 있어서 서로 상반된 결과를 보여 준다. 그 이유는 다음과 같이 분석할 수 있다. 우선, stall 방식의 경우는 복구를 위한 예비 정보를 안전하게 보관하고 있기 때문에 분



(그림 7) 분기 예측 정확도 (전역 기법)

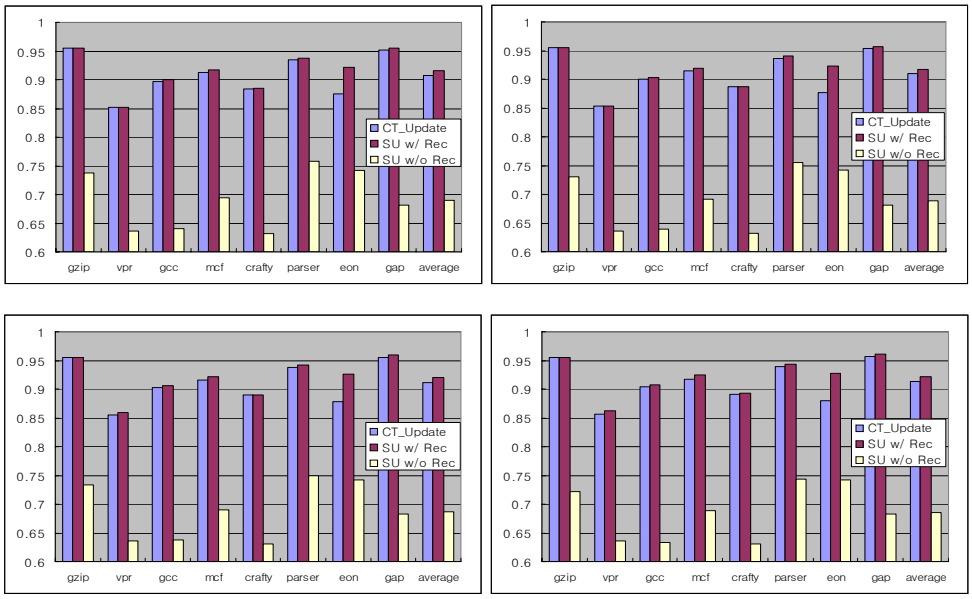


(그림 8) 복구 예비 비트의 사용 완화

기 예측의 정확도 측면에서 보다 더 우수하다. 하지만 이 같은 방식은 분기 정보의 추측적 사용에 대한 한계치를 초과했을 경우, 파이프라인 상에서의 수행을 중단시키기 때문에 전체 수행 시간에 있어서의 결과는 좋지 못하다. 반대로, no_stall 방식의 경우는 복구 예비 비트가 소실될 수 있기 때문에 분기 예측 정확도 측면에서는 불리하지만, 수행 시간 측면에서는 유리하다. 이는 파이프라인 상에서 수행 중단이 발생되지 않기 때문이다. 비록, no_stall의 경우 분기 예측 정확도 측면에서 다소간 좋지 못한 결과를 보이지만, y 축의 단위(scale)에서 보여 지듯이 예측 정확도의 차이는 사실상 미미하다고 할 수 있다. 또한 x 축의 값이 한계치에 다가 갈수록, 두가지 방식의 실험 결과는 매우 유사하다. 이에 비해, IPC의 차이는 상대적으로 조금 더 크다고 할 수 있다.

복구를 위한 백업 정보를 다소간 소실시키더라도 성능상의 향상을 가져 올 수 있다는 이와 같은 실험 결과는, 실제 복구 과정이 발생 할 때 모든 비트를 사용하는 경우는 드물며, 부분적인 복구가 일어나는 경우가 많음을 의미한다. 이 같은 결과는, 주어진 시스템에서 추측적 정보의 사용을 조금이라도 더 증가시키는 것이 성능면에서 오히려 더 유리할 수 있다는 것을 의미한다. 즉, 하드웨어의 구현 복잡도를 고려하여, S-GHR의 적절한 크기 선택과 no_stall 방식을 선택적으로 사용하게 함으로 해서, 예측 정확도 면에서는 불리하지만 전체 수행 시간 측면에서는 오히려 더 유리하게 만들 수 있다. 특히 이와 같은 방식은 In-Flight 분기 명령어의 개수가 많은 응용 프로그램의 수행 환경일수록 보다 더 유리하게 작용할 수 있다. 하지만, 이 경우 실제 정확한 복구 정보가 소실되기 때문에 모든 응용 프로그램에 대해 일반적으로 적용 될 수는 없다. 따라서 하드웨어 자원의 추가적 허용이 가능한 환경이라면 복구 예비 비트의 사용 완화보다는 S-GHR의 크기를 증가시키는 것이, 성능 향상과 정확한 수행을 위해서 보다 더 바람직한 방식이라 하겠다.

(그림 9)에 제시된 결과는, 지역 히스토리를 사용할 경우의 분기 예측 정확도를 나타낸다. 주어진 실험 환경과 비교

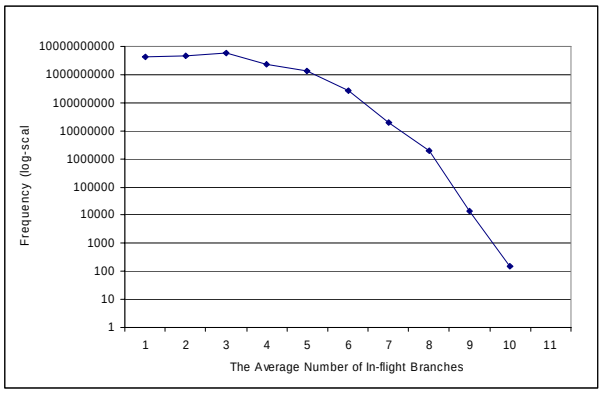


(그림 9) 분기 예측 정확도 (지역 기법)

대상은 (그림 7)에 제시된 것과 동일하다. 그리고 지역 히스토리를 사용하는 방식에서 추가적으로 필요한 BHT의 크기는 다음과 같이 결정하였다. 우선, 히스토리의 길이(width)는 PHT의 엔트리 개수와 연관하여 결정하였으며, BHT의 엔트리 수(depth)는 PHT의 엔트리 수와 동일하게 설정하였다. 전역 기법에서 보이는 결과와 유사하게 지역 기법에서도 전체적으로 SU w/ Rec가 CT_Update보다 성능면에서 우월하다는 것을 알 수 있다. 그리고 SU w/o Rec는, 전역 기법에서와 마찬가지로, 성능면에서 추측적 정보를 사용하지 않는 CT_Update 보다도 불리하다는 것을 알 수 있다.

끝으로, (그림 10)은 프로그램의 수행 도중 나타나는 In-Flight 분기의 수를 나타낸다. (그림 10)에서 제시된 수치는 *log-scale*로 표시 되었다. (그림 10)에 제시된 결과를 통해, 본 논문에서 제안된 기법이 필요로 하는 추가적 하드웨어 비용을 유추 할 수 있다. (그림 4)에 제시된 전역 분기 히스토리에 대한 복구의 경우는 S-GHR이 주된 추가 하드웨어 비용이며, (그림 5)와 (그림 6)에 제시된 지역 분기 히스토리에 대한 복구의 경우는 각각 BHQ와 SUQ가 주된 추가 하드웨어 비용이다. 이들의 크기는 In-Flight 분기의 수와 직접적인 관계가 있다. 한편, 현재 수행이 진행 중인 In-Flight 분기의 수는 실제 프로세서의 파이프라인 단계 수와 단위 시간당 제공되는 명령어의 수와 밀접하게 연관되어 있다. 따라서 해당 시스템의 환경에 따라서 그 결과가 다를 수 있으며, 본 논문에서의 결과는 표 1에 제시된 시스템 설정값과 함께 Simplescalar의 기본값을 사용한 수치이다.

(그림 10)의 실험 결과에서 보여주듯이, 전체적으로 10개 이하의 In-Flight 분기가 있다는 것을 알 수 있으며, 대부분의 경우 4~5개 이하의 분기 명령어들이 In-Flight 분기로 존재한다는 것을 알 수 있다. 가격 대비 성능의 관점에서 본다면, 대략 8개 정도의 분기로 파악하는 것이 바람직하다. 따라서 본 실험 결과를 통해, 추가적으로 사용되는 레지스터의 비트수와 큐의 엔트리 수는 8이 적당하다고 할 수 있다.



(그림 10) 평균 In-Flight 분기 명령어의 수

5. 결 론

분기 명령어에 대한 분기 예측 정확도는 시스템 전체의 성능 향상에 큰 영향을 미친다. 일반적으로 분기 예측 실패율은 전체 시스템의 성능 향상에 있어서 단일 요소로는 가장 큰 성능 제약 요소 가운데 하나로 알려져 있다.

본 논문에서는 이와 같은 분기 예측의 정확도를 높이기 위해, 분기 정보의 추측적 사용 및 이를 위한 효율적 복구 모델을 제안하였다. 제안된 기법은 각각 전역 분기 정보와 지역 분기 정보의 사용 여부에 따라 서로 다르게 적용 될 수 있다. 이를 위해 본 논문에서는 전역 분기 정보의 추측적 사용을 가능케 하는 gshare 예측기를 구현 사례로 소개하였다. 아울러, 지역 분기 정보의 추측적 사용을 위해서는 이전기록 기반 방식과 이후기록 기반 방식의 두가지 형태를

제시하였으며, 각각의 경우에 대한 효과적인 응용 프로그램 특성 및 시스템 형태를 소개하였다.

한편, 모의실험을 통해 본 논문에서 소개된 방식의 성능을 분석하였다. 또한, 복구 예비 비트의 사용 완화와 그에 대한 성능상의 차이, In-Flight 분기 명령어의 수에 대한 실험 결과와 그에 대한 분석도 함께 제시하였다. 제안된 기법은 기존 방식에 있어서 요구되던 하드웨어 복잡도를 90% 이상 절감시켰으며, 실험 결과 프로그램 수행의 정확성을 해치지 않으면서 최대 5.64%의 성능 향상을 제공하였다. 평균 In-Flight 분기 명령어의 개수는 4~5개 정도로 확인되었으며, 가격 대비 성능 관점에서 8개의 분기 명령어에 대한 관리가 적절하다고 판단된다.

참 고 문 헌

[1] R. Tomasulo, "An efficient algorithm for exploiting multiple arithmetic units," IBM J. Research and Development 11:1(January), 25-33, 1967.

[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 ISCA, 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 onMicroarchitecture, pp.228-232, Nov., 1994.

[4] E. Sprangle and D. Carmean, "Increasing processor performance by implementing deeper pipelines," In Proc. 29th Int'l Symp. on Computer Architecture, pp.25-34, 2002.

[5] Yeh, T. Y. and Patt, Y. N., "Two-level adaptive branch prediction," In Proceedings of the 24th ACM/IEEE International Symposium on Microarchitecture, 51-61, 1991.

[6] 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.

[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 MICRO, pp.228-232, Nov., 1994.

[8] 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.

[9] K. Skadron, M. Martonosi, and D. Clark. "Speculative updates of local and global branch history: A quantitative analysis," JILP, Vol.2, Jan., 2000.

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

[11] M. Evers, S. J. Patel, R. S. Chapell, and Y. N. Patt, "An analysis of correlation and predictability: What makes two-level branch predictors work," In Proceedings of the 25th Annual Intl. Symposium on Computer Architecture, pages 52-61, June, 1998.

[12] S. Sechrest, S., C.-C.Lee, T. Mudge, "Correlation and Aliasing in Dynamic Branch Predictors," in Proceedings of the 23rd ISCA, 22-32, 1996.

[13] A. R. Talcott, M. Nemirovsky, and R. C. Wood, "The Influence of Branch Prediction Table Interference on Branch Prediction Scheme Performance," International Conference on Parallel Architectures and Compilation Techniques, 1995.

[14] Chih-Chieh Lee, I. K. Chen, and T. Mudge, "The Bi-Mode Branch Predictor," International Symposium on Microarchitecture, 1997.

[15] Eric Sprangle, R. Chappell, M. Alsup, and Y. Patt, "The Agree Predictor: A Mechanism for Reducing Negative Branch History Interference," Intl. Symposium on Computer Architecture, 1997.

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

[17] S. Kim and G. Tyson, "Analyzing the working set characteristics of branch execution," in Proceedings of the 31st Annual ACM/IEEE International Symposium on Microarchitecture, pp.49-58, Dec., 1998.

[18] R. E. Kessler, "The Alpha 21264 Microprocessor," IEEE Micro, Volume 19, Issue 2, pp.24-36, 1999.

[19] E. J. McLellan, D. A. Webb, "The Alpha 21264 Microprocessor Architecture," In Proceedings of the International Conference on Computer Design, pp.90-95, IEEE, 1998.

[20] 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.363-383, Oct., 1997.

[21] 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.

[22] SPEC CPU2000 Benchmarks, http://www.specbench.org

곽 종 욱

e-mail : kwak@ynu.ac.kr

1998년 경북대학교 컴퓨터공학과(학사)

2001년 서울대학교 대학원 컴퓨터공학과 (공학석사)

2006년 서울대학교 대학원 전기컴퓨터공학부 (공학박사)

2006년~2007년 삼성전자 SOC 연구소 책임연구원

2007년~현 재 영남대학교 전자정보공학부 전임강사

관심분야 : 컴퓨터 구조, 저전력 내장형 시스템, 고성능 컴퓨팅 등