DOI: 10.3745/KIPSTA.2009.16A.5.347

분기 명령어의 조기 예측을 통한 예측지연시간 문제 해결

곽 종 욱[†] · 김 주 환[††]

요 약

정교한 분기 예측기의 설계는 오늘날의 프로세서 성능 향상에 중요한 역할을 하게 되었다. 분기 예측의 정확도가 더욱 더 중요해 지면서 정확도의 향상을 위한 다수의 기법들이 제안되었지만, 기존의 연구들은 예측 지연 시간을 간과하는 경향이 있었다. 본 논문에서는 예측 지연 시간 문제를 해결하고자 조기 예측 기법 (ESP, Early Start Prediction)을 제안한다. 조기 예측 기법은 분기 예측에 있어서 활용되는 분기 명령어의 주소 대신 그것과 일대일 대응이 되는 기본 블록의 시작 주소 (BB_SA, Basic Block Start Address)를 이용한다. 즉, 분기 명령어의 주소가 사용되는 기존의 환경에서, BB_SA를 활용하여 조기 예측을 시작함으로써, 예측 지연 시간을 숨긴다. 또한 제안된 기법은 짧은 간격 숨김 기법 (short interval hiding technique)을 통해 보다 더 나은 성능 향상을 기대할 수 있다. 실험 결과 본 논문에서 제안된 기법은 예측 지연 시간을 줄임으로써, 예측 지연 시간이 1 사이클인 이상적인 분기 예측기의 성능에 0.25% 이내로 근접한 IPC 결과를 얻었다. 또한 기본 블록의 시작주소와 분기 명령어 사이에 짧은 간격을 가질 경우에 대한 개선 방법을 추가적으로 적용시킬 경우, 기존의 방식과 비교하여 평균 4.2%, 최대 10.1%의 IPC 향상을 가져왔다.

키워드 : 분기 예측, 예측 지연 시간, Gshare 예측기, 명령어 인출, 기본 블록

Early Start Branch Prediction to Resolve Prediction Delay

Kwak, Jong Wook[†] · Kim, Ju-Hwan[††]

ABSTRACT

Precise branch prediction is a critical factor in the IPC Improvement of modern microprocessor architectures. In addition to the branch prediction accuracy, branch prediction delay have a profound impact on overall system performance as well. However, it tends to be overlooked when the architects design the branch predictor. To tolerate branch prediction delay, this paper proposes Early Start Prediction (ESP) technique. The proposed solution dynamically identifies the start instruction of basic block, called as Basic Block Start Address (BB_SA), and the solution uses BB_SA when predicting the branch direction, instead of branch instruction address itself. The performance of the proposed scheme can be further improved by combining short interval hiding technique between BB_SA and branch instruction. The simulation result shows that the proposed solution hides prediction latency, with providing same level of prediction accuracy compared to the conventional predictors. Furthermore, the combination with short interval hiding technique provides a substantial IPC improvement of up to 10.1%, and the IPC is actually same with ideal branch predictor, regardless of branch predictor configurations, such as clock frequency, delay model, and PHT size.

Keywords : Branch Prediction, Prediction Delay, Gshare Predictor, Instruction Fetch, Basic Block

1. 서 론

파이프라인(pipeline)의 깊이가 깊어지고 단위 시간당 제공(issue)되는 명령어의 수가 늘어나는 현 마이크로프로세서 환경에서, 분기 예측 정확도(branch prediction accuracy)는 시스템 전체 성능 향상에 중요한 영향을 미치게 되었다. 이는 파이프라인의 깊이 증가에 따라, 분기 예측 실패로 인한 예측 실패 비용(miss-penalty)이 더불어 증가하기 때문이며, 또한 인출(fetch) 단계와 실행(execution) 단계의 명령어 처리 속도의 차이로 인한 예측 실패 비용도 비례하여 커지기 때문이다. 이를 위해, 현재까지의 분기 예측과 관련된 연구에서는 주로 예측 정확도의 향상에 초점을 맞추어 진행 되어 왔다. 기존의 이와 같은 연구에 따르면, 분기 예측과 관련된 하드웨어의 용량이 커지면 분기 예측 테이블의 크기 증가로 인해 더 긴 분기 히스토리(branch history)를 이용할 수 있게 되고, 따라서 가명현상(aliasing)이 줄어들게 되므로

† 정 회 원 : 영남대학교 컴퓨터공학과 조교수
†† 준 회 원 : 서울대학교 전기컴퓨터공학부 박사과정
논문접수 : 2009년 5월 25일
수 정 일 : 1차 2009년 7월 22일, 2차 2009년 8월 18일
심사완료 : 2009년 8월 18일

분기 예측 정확도는 증가하게 된다. 이를 위해 대규모 분기 예측 테이블의 사용, 혹은 다수개의 분기 예측기를 활용하는 다중 예측 기법(hybrid branch prediction) 등을 통한 예측 정확도 향상의 노력이 많이 진행 되어 왔다[1, 2].

그러나 기존의 많은 연구에서는 분기 예측의 예측 지연 시간 문제(prediction delay)를 간과하는 경향이 있었다. 칩 설계에 대한 집적도가 높아짐에 따라 하드웨어의 용량이 커지고, 또한 클럭의 속도가 빨라지는 현 추세에서는, 분기 예측 지연 시간이 더 이상 간과 될 수 없는 문제가 되었다. 이는 대용량의 분기 예측 테이블의 사용으로 인하여 하드웨어 규모가 커지면서 분기 예측기로의 접근 지연 시간이 길어지고, 또한 클럭 주파수(clock frequency)가 빨라지면서 미세 공정의 사용으로 인한 회로의 지연 시간(wire delay)이 증가되기 때문이다. 따라서 분기 예측의 정확도를 향상시키더라도 그 만큼 예측에 드는 실행 비용이 증가하게 되어, 실질적인 성능 (IPC, Instruction Per Clock) 향상은 기대할 수 없게 되는 경우가 많다. Jimenez et al.은 그들의 연구에서, 정확도의 향상만을 위해 분기 예측기의 지연 시간을 증가시키는 것은 아무런 소용이 없다고 밝히고 있다. 또한, 평균 2 사이클의 접근 지연시간을 가지는 100% 정확한 이상적인 분기 예측기(ideal branch predictor)의 사용보다, 평균 1 사이클의 예측 지연 시간을 가지는 95% 전후의 정확도를 가지는 분기 예측기의 사용이 오히려 보다 더 높은 IPC를 제공할 수 있다는 사실을 보였다[3]. 그 외에 여러 연구들 또한 이와 유사한 사실을 보고하고 있다[4, 5].

본 논문에서는 분기 예측의 지연 시간을 단축시키기 위한 조기 분기 예측(Early Start Prediction, ESP) 기법을 제안한다. 제안될 방식은 분기 예측의 정확도를 유지하면서 분기 예측 지연 문제를 해결하여 시스템 전체적인 IPC 향상에 기여한다. 조기 예측 기법은 기존에 분기 예측에 있어서 사용되었던 분기 명령어의 주소(branch address) 대신, 그것과 일대일 대응이 되는 기본 블록의 시작 주소(BB_SA, Basic Block Start Address)를 이용한다. 이를 통해, 분기 주소가 사용될 때 이에 대응하는 기본 블록의 시작 주소를 활용하여 조기 예측을 시작함으로써 예측 지연 시간을 숨긴다. 본 논문에서 사용되는 기본 블록(basic block)이란 연속적인 코드들의 나열 집합으로서, 해당블록으로의 진입 이후부터 해당 블록의 진출사이에 있어서, 분기 명령어 혹은 예외처리(exception handling)과 같은 요소들로 인해 실행 경로가 바뀌지 않는 코드들의 집합을 의미한다[6].

이하, 본 논문의 구성은 다음과 같다. 2장에서는 분기 예측 지연 문제를 해결하기 위해 제안되었던 기존의 연구들을 소개한다. 3장에서는 기본 블록의 시작 주소를 이용한 조기 예측 기법을 제시한다. 4장에서는 모의실험을 통해 본 논문에서 제안된 기법의 성능을 검증한다. 아울러, 기본 블록의 시작 주소와 분기 명령어 주소 사이의 좁은 간격에 대한 문제 해결을 고려한 향상된 기법을 추가적으로 소개하고 이에 대한 성능도 관찰한다. 끝으로 5장에서는 결론을 맺는다.

2. 관련 연구

분기 예측의 지연시간을 감소시키기 위해서, 분기 예측 테이블로의 접근을 파이프라인화 하여 수행하는 Pipelined 분기 예측기가 제안되었다. 이 방식은, 분기 예측기가 예측 테이블을 접근하는 과정을 파이프라인화시켜 매 사이클마다 분기 예측을 가능하게 한다. 이전 사이클에 분기 히스토리를 사용하여 분기 예측 테이블을 접근한 뒤, 이를 작은 크기의 예측 테이블 버퍼에 저장해 둔다. 그리고 해당 분기 명령어가 나타나면 그 동안 추가된 분기 히스토리 정보를 이용하여 예측 테이블 버퍼에서 최종 엔트리를 선택한다. 하지만 주어진 방식은 분기 히스토리 정보는 추후에 보정하지만, 나머지 정보들은 분기 예측 테이블에 접근하는 시점에서 유효한 정보만 사용할 수 있기 때문에, 분기 히스토리 이외에 다른 정보를 추가적으로 사용하는 예측 알고리즘에 적용할 경우, 예측 정확도가 떨어지게 된다[7].

또한, 분기 예측의 지연시간을 감소시키기 위해서 Lookahead 분기 예측기가 제안되었다. 주어진 방식은 분기 예측기가 분기 명령어의 주소를 활용하여 예측을 수행하지만, 현재 수행 중인 해당 분기 명령어를 예측하지 못하고 그 다음에 나올 분기 명령어를 예측한다. 즉, 분기 명령어를 하나씩 뛰어넘어 예측을 수행하기 때문에 분기 예측 지연 시간은 숨길 수 있지만 예측 정확도가 감소하게 된다[8]. 또한 100% 정확한 예측기의 설계는 불가능하다고 판단할 때, 수행되지 않을 분기 명령어의 예측을 진행하는 경우도 발생하여 프로세서 자원의 낭비를 초래할 수 있다.

Lookahead 예측기의 개선된 형태라 할 수 있는 Cascading 분기 예측기도 제안되었다. 이 방법은 분기 예측을 위해서 분기 명령어의 목표 주소(target address)와 분기 히스토리(branch history)를 사용한다. 주어진 방식 역시 Lookahead 예측기의 경우에서처럼, 분기 명령어들의 간격만큼 예측 지연 시간을 감출 수 있다. 해당 방식은 분기 목적 버퍼 (BTB, Branch Target Buffer)의 주소를 사용하기 때문에, BTB의 적중률에 따라 예측 정확도의 성능이 좌우되게 된다[9].

예측 결과에 대한 Overriding 기법 역시 잘 알려진 예측 지연 시간 감소 기법이다. 이 방식은 정확하지만 느린 예측 시간을 가진 주 예측기(main predictor)와 부정확하지만 빠른 예측을 수행하는 보조 예측기(sub predictor)를 사용한다. 분기 명령어가 나타나면, 빠른 보조 예측기를 사용하여 예측을 수행 한 뒤, 이후 주 예측기의 결과가 나오면 이전의 예측 결과와 상호 비교한다. 예측 결과가 만약 일치 하지 않는 경우는 이슈된 명령어들을 취소하고 주 예측기의 결과에 맞게 재 수행 하게 된다. 이 방식은 주 예측기가 올바른 예측을 수행하더라도 보조 예측기가 잘못된 예측을 수행할 경우 파이프라인의 빈 단계(bubble stage)가 발생하게 된다[9].

특정 프로세서 환경 하에서 분기 예측 지연시간을 감소시키고자 했던 연구도 소개 되었다. Falcon et al.은 SMT (Simultaneous Multi-Threading) 프로세서 환경에서의 분기 예측 지연 시간의 감소에 대한 연구를 소개하였다. 제안된

기법은 명령어 인출(fetch) 유닛을 분리되게 구현하고, 파이프라인화된 쓰레드 상호간의 분기 예측기법을 사용한다. 해당 방식은 쓰레드(thread) 실행 환경 하에서 적용 가능한 방식으로, 특정 환경에서의 특화된 구현이라 할 수 있다[10].

다중 블록 선 예측(Multi-Block Ahead Prediction)은 몇몇개의 블록 이전에 미리 예측을 수행하는 방법이다. 이는 예측 지연 시간을 효율적으로 감소시킬 수 있지만, 예측 시점에서의 정보가 부족하기 때문에 예측 정확도는 줄어들게 된다. 이와 같은 현상은 고성능의 분기 예측기일수록 더욱 심해진다. 실제로 이러한 정확도의 하락으로 성능 또한 좋지 못하다. 관점에 따라 해당 연구는 기 언급된 파이프라인화된 예측기가 다중 블록 선 예측기의 더 발전된 형태로도 볼 수 있으나, 예측 지연 시간의 감소를 위해 예측 정확도를 희생시키는 방법은 효율적이라 할 수 없다[8].

Trace 재구성(reconstruction)의 관점에서 기존의 trace 캐시[11]는 명령어 인출의 대역폭을 확보하자는 점에서는 본 논문에서 제안하는 방식과 공통의 목표를 가지고 있다. 하지만, 이는 접근 방식이 다르다. Trace 캐시는 인출 대역폭(fetch bandwidth)을 위해 이전에 인출 되었던 명령어의 특정 단위를 trace로 저장하여 재활용하는 기법이며, trace 예측에 있어서 각각의 분기 명령어에 대하여 예측을 수행하는 것이 아니라, 분기 명령어들이 포함된 명령어 단위인 trace에 대해서 예측을 수행하게 된다. 따라서 각각의 분기 명령어에 대한 예측 정확도는 감소할 수밖에 없다. 또한 이 방식은 전역 분기 정보(global branch history)와 같은 정보도 효율적으로 이용하지 못한다. 물론 인출 대역폭의 관점에서만 본다면, trace 캐시의 경우 trace 예측이 정확할 경우 trace를 이용하여 분기 명령어가 포함된 명령어들을 한번에 인출 할 수 있기 때문에 더 효율적일 수도 있지만, 본 논문의 주안점과는 궤를 달리 한다[11].

이상에서 소개된 바와 같이 분기 예측의 지연 시간 감소를 위해 기존에 제시된 여러 기법들은 각각 나름대로의 한계를 가진다. 대용량의 분기 예측 테이블의 필요 혹은 추가적인 하드웨어 복잡도의 요구는 실제 구현에 있어서 많은 문제점을 노출한다. 특히 내장형 환경에서는 보다 더 심각한 문제로 제시될 수 있다. 또한, 예측 지연 시간 감소를 위해 예측 정확도를 희생 한다거나 BTB와 같은 다른 모듈에 상호 의존된 형태로 구현되어서는 안 될 것이다. 본 논문에서 제시될 ESP 기법은 이상의 문제점을 해결함과 아울러 그 단순한 구현으로 인해 일반 프로세서 환경 뿐만 아니라, 내장형 환경에서도 다양하게 적용될 수 있는 장점을 가지고 있다.

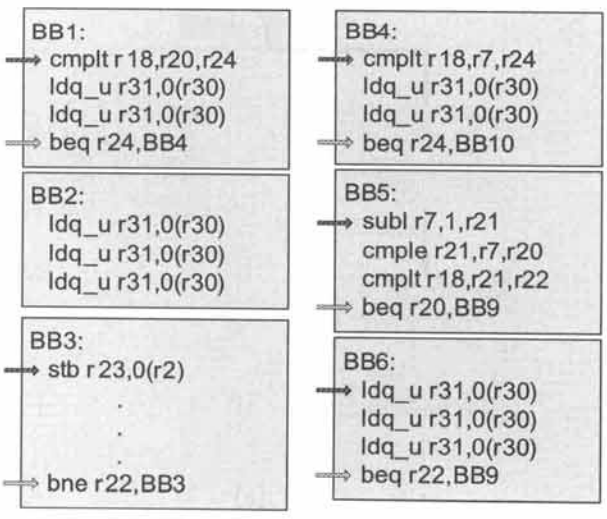
3. 조기 분기 예측 기법

이절에서는 분기 예측의 지연 시간을 감소시키기 위한 조기 분기 예측 기법(ESP, Early Start Prediction) 을 제안한다. 대부분의 분기 예측기들은 그들의 분기 예측에 있어서, 분기 명령어의 주소값(branch address)과 분기 히스토리(branch history)를 입력값으로 사용한다. 조기 분기 예측 기법에서는 기본 블록의 시작 주소(BB_SA, Basic Block Start Address)를 분기 명령어 주소값의 대체 입력으로 활용한다.

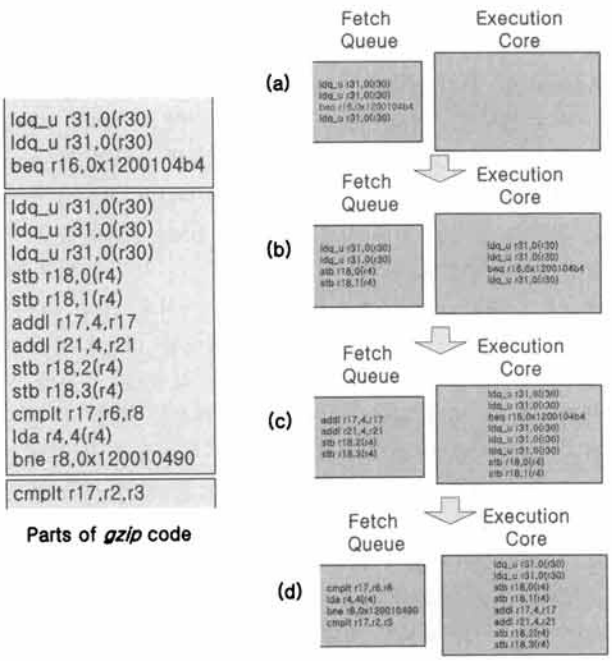
본 논문에서 제안된 조기 분기 예측 기법의 설명을 위해, (그림 1)에 SPEC 벤치마크 프로그램 가운데 하나인 *164.gzip*에 해당하는 코드의 일부가 소개되어 있다[12]. 제시된 코드는 관찰의 편의를 위해 기본 블록(BB, Basic Block)들로 구분하여 나타내었다. (그림 1)에서 보이듯이, 각각의 기본 블록들은 0 혹은 1개의 분기 명령어들을 가지고 있다. 즉, 각 분기 명령어들은 그 분기 명령어에 일대일 대응을 이루는 고유한 기본 블록을 가지고 있다. 이는 각 기본 블록의 시작주소가 각 분기 명령어들과 일대일 대응을 이룰 수 있다는 것을 의미한다. 본 논문에서는 이를 활용하여 분기 명령어의 주소 대신, 기본 블록의 시작 주소를 분기 예측에 입력 요소로 활용한다. 일반적으로 기본 블록의 시작 주소는 분기 명령어보다 앞서 나온다. 따라서, 분기 예측에 있어서 이 같은 기본 블록의 시작 주소를 활용할 경우, 분기 명령어에 비해 기본 블록의 시작 주소가 앞서 나오는 사이클 수만큼 예측 지연 시간을 숨기는 효과를 가지게 된다.

비록 각각의 분기 명령어들은 그에 해당하는 각각의 기본 블록들을 가지고 있지만, 컴파일러에 의해 변환된 실행 파일에는 이와 같은 정보가 유지 되지 않는다. 따라서 실행시간에 동적으로 기본 블록의 시작 주소를 파악하기 위한 방법이 필요하다. 이를 위해 본 논문에서는 동적 발견 방법(dynamic heuristic)을 활용하여, 기본 블록의 시작 주소를 각 기본 블록의 첫 번째 명령어의 주소값으로 정의한다. 이는 간단하게 분기 명령어의 다음 명령어에 해당하는 주소가 해당 명령어를 포함하는 기본 블록의 시작 주소가 된다. 이와 같은 방법으로 사용할 경우, 가령 (그림 1)에 제시된 기본 블록 BB1에 대한 기본 블록 시작 주소(BB_SA)는 명령어 "cmpltr18,r20,r24"의 주소값이 된다.

한편, (그림 2)는 *164.gzip*의 또 다른 코드의 일부이다. (그림 2)에 제시된 코드 및 실행 순서는 4-way 슈퍼스칼라



(그림 1) *164.gzip* 코드 일부



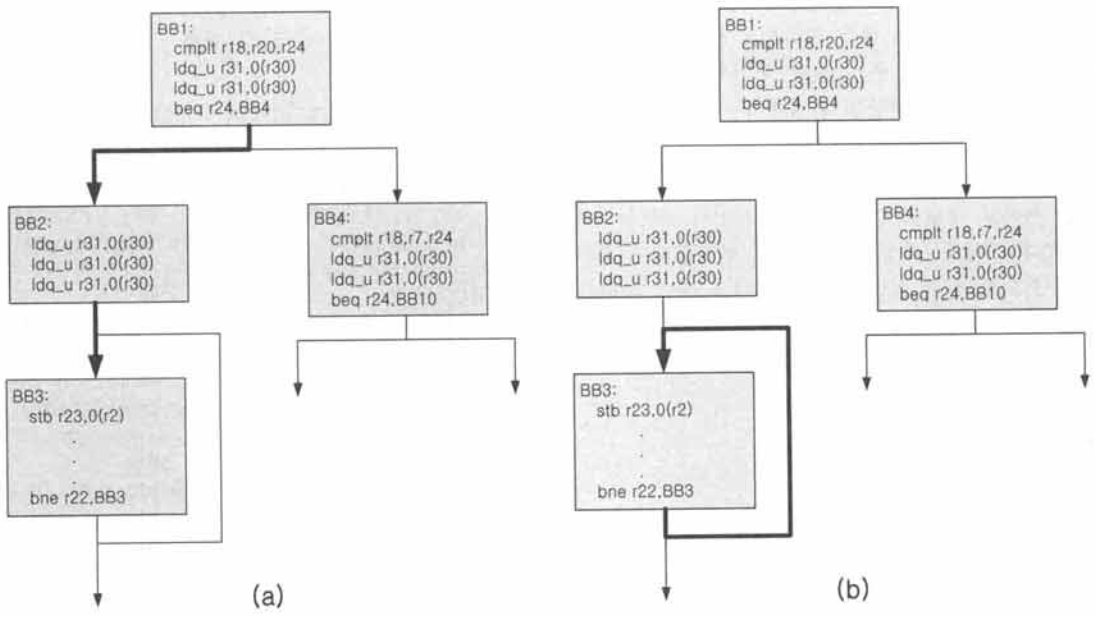
(그림 2) 명령어 인출 및 실행 순서

(superscalar) 프로세서를 가정하여, 이를 인출 모듈(fetch module)과 실행 모듈(execution module)의 관점에서 나타내었다. 설명의 편의를 위해, 위의 코드에서 "bne r8, 0x120010490"에 해당하는 명령어를 현재 예측하고자 하는 분기 명령어라고 가정한다. 주어진 경우를 분석해 보자. 이는 (그림 2)에서 보여 지듯이, 해당 분기 명령어가 (d) 단계에서 인출 모듈에 입력되게 된다. 동시에 "bne r8, 0x120010490" 명령어에 대한 분기 예측이 시작된다. 만약 1 사이클 이내에 주어진 분기 명령어에 대한 예측 결과가 도출 될 경우, (d) 단계의 바로 다음 단계에서 예측의 결과로 수행될 다음 명령어를 정상적으로 인출 해 올 수 있게 된다. 하지만, 고속의 클럭 스피드, 미세 공정으로 인한 지연 모델(delay model), 대규모 분기 예측 테이블의 사용으로 인한 예측 지연 증가 등의 이유로, 바로 다음 사이클에 분기 예측의 결과가 정상적으로 도출 되지 않을 수 있다. 이 경우, 파이프라인의 멈춤(stall) 현상이 불가피하다.

하지만, 주어진 분기 명령어에 대한 분기 예측을 만약 (c) 단계에서 조기에 수행 할 수 있었다면, 1 사이클의 추가적인 예측 지연 시간을 확보 할 수 있다. 즉, 2 사이클의 예측 지연 시간을 숨길 수 있다. 이는 마찬가지로 만약 (b) 단계에서 분기 예측을 조기에 수행 하였다면, 2 사이클의 추가적인 예측 지연 시간 확보와 3 사이클의 예측 지연 시간 숨김이 가능하다. (a) 단계의 경우는 3 사이클의 추가적인 예측 지연 시간 확보와 4 사이클의 예측 지연 시간 숨김이 가능하게 된다.

앞서 설명한 바와 같이, 기본 블록의 시작 주소로서 각 기본 블록의 첫 번째 명령어의 주소값을 분기 명령어 대신 활용할 경우, (그림 2)의 예제에서는 최대 3 사이클의 추가적인 예측 지연 시간의 확보 및 4 사이클의 예측 지연 시간 숨김이 가능하다. 즉, "bne r8, 0x120010490"의 예측을 위해, 이와 일대일 대응을 이루는 "ldq_u r31, 0(r30)" 명령어의 주소값을 사용할 경우, 추가적인 예측 지연 시간 없이 다음 명령어를 인출하는 것이 가능하게 된다.

하지만, 모든 기본 블록들이 그에 해당하는 분기 명령어를 가지는 것은 아니다. 예를 들어 (그림 1)의 기본 블록 2(BB2)는 그에 대응하는 분기 명령어를 가지지 않는다. 이와 같은 상황은 분기 명령어와 기본 블록의 시작 주소 사이에 가명현상을 유발 시킬 수 있다. (그림 3)에 가명 현상에 대한 실질적인 예가 나타나 있다. (그림 3)은 (그림 1)의 코드를 실행 관점에서 재구성한 것이다. (그림 3)에서 기본 블



(그림 3) 가명 현상 예제도

록 3(BB3)의 "bne r22, BB3" 명령어를 현재 예측하고자 하는 분기 명령어라고 가정한다. 이 경우, 기본 블록 3에 도달하는 두가지 서로 다른 경로가 존재하게 된다. 즉, (그림 3)의 (a)에 해당하는 경로와 (b)에 해당하는 경로가 그것이다. 그림 (a)에 해당하는 경로일 경우, 기본 블록 2(BB2)의 "ldq_u r31, 0(r30)"에 해당하는 주소값이 "bne r22, BB3"의 기본 블록 시작주소가 된다. 한편, (b)에 해당하는 경로로 실행이 진행 되었을 경우는 "stb r23, 0(r2)"에 해당하는 주소값이 "bne r22, BB3"의 기본 블록 시작 주소가 된다.

이와 같이 모든 기본 블록들이 분기 명령어를 가지는 것은 아니라는 사실에 의해, 분기 명령어와 기본 블록의 시작 주소 사이에 가명현상이 발생할 수 있다. 따라서 조기 분기 예측에 있어서 이 같은 가명현상 문제가 실제 분기 예측에 미치는 성능상의 영향 정도에 대한 분석이 필요하다. 이에 대해서는 본 논문의 4절에서 추가적으로 논의한다. 하지만 실험 결과 확인된 바로, 본 논문에서 제안된 기본 블록 시작 주소의 활용은 가명 현상에 영향을 받는 정도가 미미한 것으로 파악되었다. 다음으로, 본 논문에서 제안되는 조기 분기 예측에 대한 실질적인 구현 사례를 소개한다.

본 논문에서 제안된 조기 분기 예측 기법은 분기 명령어의 주소를 활용하는 대부분의 분기 예측기에 적용이 가능하다. 여러 분기 예측기들 가운데 본 논문에서는 *gshare* 분기 예측기를 이용하여 분기 예측기를 구현 사례로 제시하고, 또한 4절에서는 이에 대한 성능 향상의 정도를 비교 분석한다[14]. *gshare* 예측기는 하드웨어의 단순함에도 불구하고 비교적 높은 예측 정확도를 제공하기 때문에, 분기 예측과 관련된 연구에서 성능 향상의 비교 대상으로 자주 사용되는 분기 예측기 가운데 하나이다[15, 16].

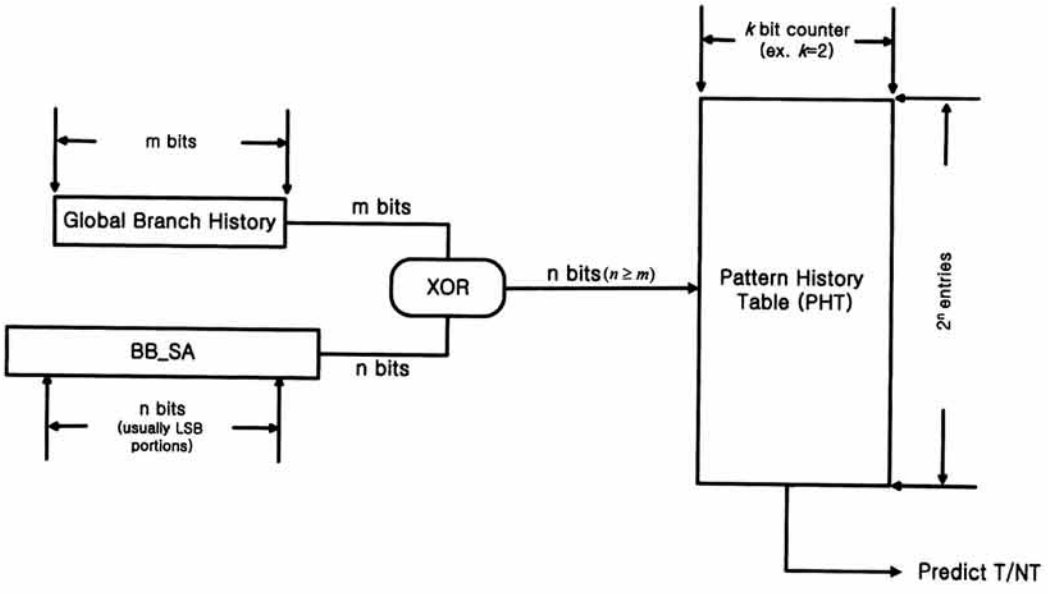
(그림 4)는 ESP *gshare* 예측기의 기본 구조를 보여 준다. (그림 4)에서 보여 지듯이, 기존의 *gshare* 기법은 전역 분기 히스토리(global branch history)와 분기 명령어의 주소값(program counter)을 사용하여, 이를 인덱스 함수인 *xor* 함수에 대입한다. 본 논문에서는 분기 명령어의 주소값을 기본 블록의 시작 주소로 변경한다. 한편, 인덱스 함수인 *xor* 함수의 실행 결과를 이용해 분기 예측 테이블(PHT, Pattern History Table)에 접근하며, 해당 카운터의 상위 비트 값에 따라 분기 예측을 수행한다. *xor* 함수는 동작의 단순성에 의해 빠른 함수 출력을 보장하고 그로 인해 예측 지연 시간 감소에 효과적이다. 뿐만 아니라, 함수의 특성 상 입력값에 대한 출력을 고르게 분포시키는 특성을 가지고 있어 가명현상을 효과적으로 줄일 수 있다. 분기 예측 테이블의 각 엔트리들은 k 비트 카운터(k-bit counter)로 구성되어 있다.

4. 모의 실험 및 성능 분석

이 절에서는, 본 논문에서 제시된 기법의 성능 평가를 수행한다. 우선, 기본 블록의 시작 주소를 활용할 경우에 대한 가명현상의 영향도를 파악한다. 즉 분기 예측에 있어서 예측 정확도의 차이를 분석하며, 다음으로 본 논문에서 제안된 기법이 제공하는 IPC의 향상 정도를 소개한다. 끝으로, 기본 블록의 시작 주소와 분기 명령어 사이의 짧은 간격이 발생하는 경우에 대한 추가적인 논의를 수행하며, 이를 반영하기 위한 추가적 개선 형태의 IC_ESP *gshare* 예측기를 소개한다.

4.1 모의실험 환경 및 벤치마크 프로그램

본 논문에서의 모의실험은 구동 기반 시뮬레이터인 *SimpleScalar*로 진행되었다[13]. 이벤트 구동형 시뮬레이터(event-driven simulator)인 *SimpleScalar*는 빠른 모의 실험과 높은 정확도의 결과를 보장하는 강력한 모의실험 환경으로, 비순서 실행(out-of-order issue), 비중단 캐쉬(non-blocking cache),



(그림 4) ESP *gshare* 예측기

〈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 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

투기적 실행(speculative execution)등과 같은 최신 프로세서 기술을 지원함과 아울러, 다양한 분기 예측 기법의 사용을 가능하게 한다. <표 1>에 본 논문에서 사용된 실험 환경이 제시되어 있다.

한편, 모의실험에 사용된 벤치마크 프로그램은 SPEC에서 제공되는 CPU 성능 측정 프로그램인 SPEC CINT 프로그램들이며, 이 가운데 실험 환경에서 지원하는 10개의 벤치마크를 사용하였다. 일반적으로 SPEC에서 제공하는 CPU 성능 평가 프로그램은 정수형 프로그램인 CINT와 실수형 프로그램인 CFP로 구분되는데, CFP 프로그램의 경우는 과학 계산 형태의 응용 프로그램이 주를 이루며, 이들은 매우 정형화되어 있다. 이 같은 경우, 분기 예측의 정확도가 매우 높게 나타나기 때문에, 분기 예측의 개선으로 인한 성능 향상의 정도를 효율적으로 관찰하기에는 곤란하다. 따라서 CFP 프로그램들은 분기 예측과 관련된 연구에서 일반적으로 제외된다[12].

4.2 실험 결과 및 분석

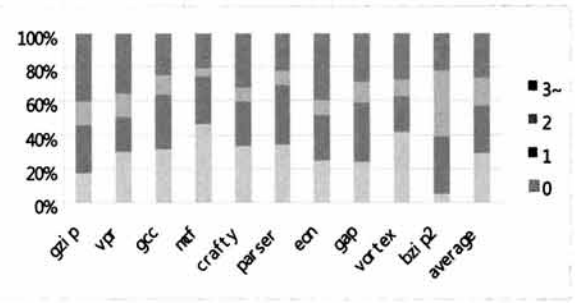
우선, 분기 명령어의 주소값을 기본 블록의 시작 주소로 대체함에서 오는 분기 예측 정확도의 차이를 분석할 필요가 있다. 이를 위해 본 논문에서는 제안된 조기 분기 예측 기법과 기존의 예측 방식 사이의 예측 정확도의 차이를 분석한다. <표 2>는 분기 명령어의 주소값을 사용했을 경우(A)와 기본 블록의 시작 주소를 활용한 경우(B)에 대한 분기 예측 정확도 및 두 가지 기법에서의 분기 예측 정확도의 차이((A)-(B))를 보여주고 있다. 전술한 바와 같이, 기본 블록의 시작 주소를 활용할 경우 분기 예측의 정확도를 감소시킬 가능성이 있다. 특히, (그림 3)을 통해 소개되었듯이, 분기 명령어와 기본 블록 시작 주소 사이의 가명현상이 발생할 경우는 더욱 문제가 될 수 있다.

하지만, <표 2>의 실험 결과에서 보이는 바와 같이, 두가지 방식에 있어서 예측 정확도의 차이는 극히 미미하다고 할 수 있다. 즉, 본 논문에서 활용하는 동적 발견 방법(dynamic heuristic)이 가명현상을 발생 시킬 수 있지만, 정확한 기본 블록의 시작주소를 찾지 못해서 발생하는 가명현상이 실제로 전체 시스템의 성능에 많은 영향을 주지 못한다는 사실을 확인 할 수 있다. 만약 가명현상이 전혀 없이 완벽하게 분기 명령어와 기본 블록의 시작 주소를 매칭 시켰다면, 예측 정확도 또한 일치하여야 한다. 이는 1:1 매칭이 가능한 다른 address로 치환 한 것과 같기 때문이다. 하지만, <표 2>의 실험 결과에서 보여주듯이 이러한 가명현상으로 인한 accuracy의 저하는 극히 미미 하므로, 단순한 형태의 동적 발견 방법(dynamic heuristic)을 활용하여도 거의 100%에 가까운 기본 블록의 시작 주소를 올바르게 파악한다고 판단된다. 실험 결과 분기 예측 정확도의 차이는 평균 0.047%이며, 16KB 분기 예측 테이블이 사용된 경우는 최소 0.012%에 해당하는 예측 정확도의 차이를 보인다. 이와 같은 수치는 또한 구동 기반 시뮬레이터의 일반적인 오차 범위 이내라고 판단해도 무방할 정도의 미미한 차이이다. 이와 같이, 본 논문에서 제안된 방식을 사용할 경우 분기 예측 정확도의 차이는 사실상 존재하지 않는다고 판단할 수 있으며, 이는 분기 명령어의 주소값 대신 기본 블록의 시작 주소가 분기 예측시에 새로운 입력 요소로 대체될 수 있음을 의미한다.

한편, (그림 5)는 기본 블록 시작 주소와 분기 명령어 사이의 거리(interval)를 측정한 실험 결과이다. 즉, 기본 블록 시작 주소에 해당하는 명령어가 인출되고 몇 사이클 후에 분기 명령어가 인출 되는지를 보여주는 결과이다. 실험 결과에서 보여 주듯이, 약 71.2%는 기본 블록 시작 주소와 분기 명령어 사이의 거리가 1 사이클 이상이다. 또한 그 가운

〈표 2〉 분기 예측 정확도의 차이

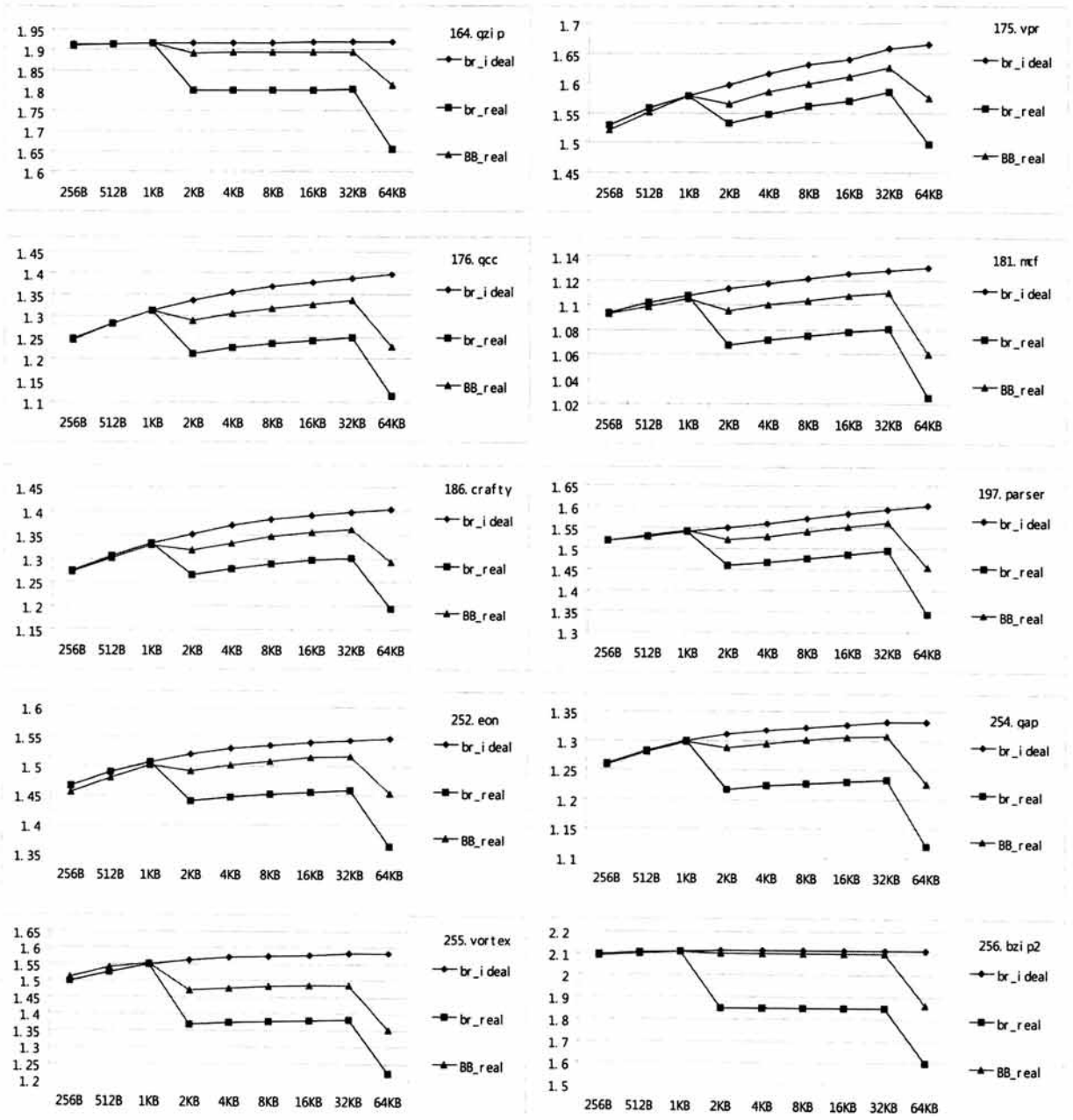
Size	Branch Address (A)	BB_SA (B)	(A) - (B)
512B	0.92148	0.91993	0.00155
1KB	0.93238	0.93182	0.00056
2KB	0.94085	0.94016	0.00069
4KB	0.94724	0.94674	0.0005
8KB	0.95175	0.95161	0.00014
16KB	0.95538	0.9555	-0.00012
32KB	0.95874	0.95855	0.00019
64KB	0.96104	0.96081	0.00023



(그림 5) 기본 블록 시작 주소와 분기 명령어 사이의 거리 비율

데 42.7%는 2 사이클, 29.3%는 3 사이클 이상이다. 즉, 주어진 실험 결과를 통해 전체 분기 명령어 가운데 28.5%는 1 사이클, 13.4%는 2 사이클, 29.3%는 3 사이클의 예측 시간 여유가 생긴다는 것을 알 수 있다. 이는 <표 2>에 제시된 분기 예측 정확도 하에서, 해당 비율만큼의 IPC 향상을 가져 올 수 있다는 의미를 내포한다.

(그림 6)의 실험 결과는 본 논문에서 제안된 기법의 IPC 향상 정도를 보여준다. 그래프에서 X축은 PHT 크기이며, Y축은 IPC를 의미한다. 주어진 실험에서의 비교 대상은 다음과 같다. 실험에서 *br_ideal*은 분기 예측기의 구성에 관계



(그림 6) 조기 분기 예측 기법의 IPC (X축: PHT 크기, Y축: IPC)

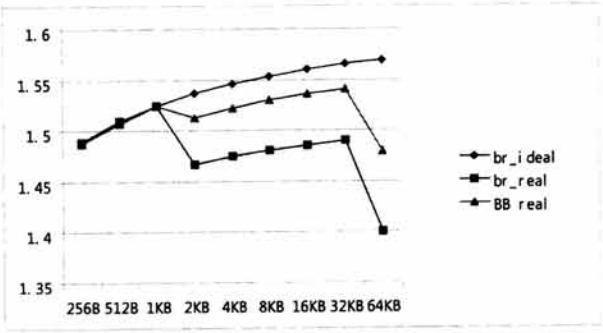
없이 항상 1 사이클의 예측 지연을 가지는 이상적인 분기 예측기(ideal branch predictor)를 의미한다. *br_real*은, 분기 예측기의 크기, 시스템 구성 환경 그리고 구현 공정 등에 종속되어, 일정한 예측 지연 시간의 증가를 야기하는 현실적인 예측기(realistic branch predictor)를 의미한다. 본 논문에서는 3GHz의 프로세서 클럭 속도와 180nm 의 설계 공정을 가정하였다[9]. *br_real*의 시스템 구성은 <표 1>을 기본으로 사용하며, [9] 문헌에서 활용하는 예측 지연 시간을 소비하는 기본적인 분기 예측기이다. 한편, *BB_real*은 본 논문에서 새롭게 제시된 방식을 의미한다. 즉, 기본 블록(BB)의 시작 주소를 활용한 현실적인 모델을 의미한다.

실험 결과에서 보여 지듯이, *br_real*은 분기 예측기의 크기가 증가함에 따라 예측 지연 시간이 2 사이클 혹은 3 사이클로 증가하게 되며, 이에 대응하는 IPC 급락 지점(drop point)이 존재한다. 본 논문에서의 실험 결과로는 2KB와 32KB가 각각에 해당되는 지점이다. 실험 결과에서도 알 수 있듯이, 주어진 경우는 추가적인 하드웨어의 투입에도 불구하고 오히려 IPC 성능이 감소함을 관찰할 수 있다. 분기 예측 지연 시간의 중요성을 다시 한번 확인 시켜 주는 실험 결과라 하겠다.

한편, 항상 1 사이클의 예측 지연시간만을 요구하는 *br_ideal*은 분기 예측기의 크기 증가함에 따라 이에 비례하여 분기 예측 정확도가 꾸준히 향상됨을 알 수 있다. 이 같은 결과는 *br_ideal*의 경우, 어떠한 예측 지연 시간도 발생하지 않는 이상적인 예측기이므로 IPC 또한 연속적으로 증가하게 된다. 하지만 이는 실제 구현 불가능한 모델로 새롭게 제안되는 모델의 성능 비교 대상으로 활용될 뿐이다.

끝으로, 본 논문에서 제시된 *BB_real* 기법은 *br_real* 기법에 비하여 상당한 성능상의 향상을 가져왔으며, *br_ideal*에 보다 더 근접한 성능을 나타냄을 알 수 있다. 이를 보다 더 효과적으로 관찰하기 위해, (그림 7)에 (그림 6)에 제시된 실험 결과들의 평균치를 제시하였다. 그래프에서 X축은 PHT 크기이며, Y축은 IPC를 의미한다. 실험을 통해 본 논문에서 제시된 기법은 기존의 기법 대비 최대 5.3%의 IPC 향상을 가져 왔으며, 또한 *br_ideal* 기법과 비교하여 IPC에 있어서 평균 1.5% 이내의 근소한 차이를 나타내고 있다.

참고로, 본 논문에서는 기본 블록의 시작주소와 실제 예



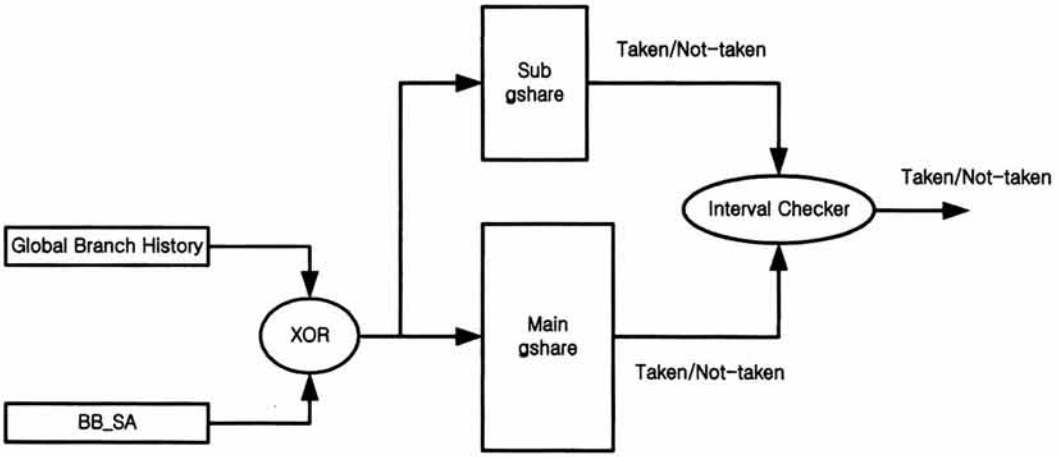
(그림 7) 조기 분기 예측 기법의 평균 IPC (X축: PHT 크기, Y축: IPC)

측의 대상이 되는 분기 명령어와의 거리가 중요한 요소가 된다. 가령, 4 instruction fetch/issue가 가능한 superscalar processor를 가정한다면, 1~4거리에 있는 명령어의 경우 0 또는 1 cycle의 차이만을 가질 수 있으므로, 기본 블록의 시작주소를 사용하는 것과 실제 분기 명령어에 대한 예측을 수행하는 것과 큰 차이를 가질 수 없다고 할 수 있다. 하지만, (그림 7)에서 주어진 성능 향상의 원인은 instruction fetch의 특수성에서 기인한다. 실제 execution은 data dependency, cache miss 혹은 RoB full 등 여러 가지 특수 상황으로 인해 4-issue superscalaring을 가정해도 항상 4 instruction씩 issue될 수 있는 것은 아니다. 또한 (그림 5)의 결과를 통해 알 수 있듯이, 대략 43% 이상은 2 cycle 이상의 여유를 가진다는 사실이다. 매 싸이클마다 4 instruction씩 정확하게 fetch 할 수 있다면, 위의 비율은 더 줄어들 것이라고 판단할 수 있으나, 실제는 매 싸이클 4 instruction씩 fetch할 수 없기 때문에 제안하는 예측 기법의 성능 향상의 원인 가운데 하나를 이를 통해 확인 할 수 있다.

한편, 본 논문에서 제안된 조기 분기 예측 기법에서는 기본 블록의 시작 주소와 분기 명령어 사이에 충분한 거리(즉, 사이클)를 확보하지 못할 경우, 기존의 방식과 비교하여 얻을 수 있는 성능 향상의 폭이 줄어들게 된다. 다시 말해, (그림 5)에 제시된 실험 결과 가운데, 기본 블록의 시작 주소와 분기 명령어 사이의 거리가 0 인 경우는 본 논문에서 제시된 기법을 통해 성능 향상을 기대 할 수 없게 된다. 엄밀한 의미에서 이는 기존 방식에서도 동일하게 존재하는 한계로서, 본 논문에서 제안된 기법 고유 문제라 할 수는 없다. 본 논문의 실험 결과를 통해 확인 된 바로는, 전체 분기 명령어 가운데 약 28.8%가 기본 블록의 시작 주소와 분기 명령어 사이의 거리가 0 사이클인 경우이다. 즉, 기본 블록의 시작 주소에 해당하는 명령어와 분기 명령어가 동시에 인출되는 경우가 28.8%이다. 이 경우는 분기 예측에 있어서 기본 블록의 시작 주소로 분기 주소를 대체한다하더라도 분기 예측에 필요한 추가적인 예측 지연 시간을 확보할 수 없다.

이와 같은 문제를 해결하고자 본 논문에서는, (그림 4)의 구조를 기반으로, 보다 더 개선된 형태의 새로운 분기 예측기를 제안한다. 제안된 예측기는 (그림 8)에 나타나 있으며, 이를 본 논문에서는 Interval Check ESP (IC_ESP) *gshare* 예측기라 명명한다. IC_ESP 예측기는 기본적으로 기존의 ESP *gshare* 예측기를 활용한다. 다만 분기 명령어의 주소 필드를 기본 블록 시작 주소(BB_SA)로 대체하며 분기 히스토리(branch history)는 기존의 방식과 동일하게 사용된다. 그리고 이를 주 예측기(main predictor)라 한다. 또한 IC_ESP 예측기는 1 사이클의 예측 지연 시간을 가지는 작지만 빠른, 하지만 주 예측기에 비해서 정확성면에서는 떨어지는 보조 예측기(sub predictor)를 활용한다.

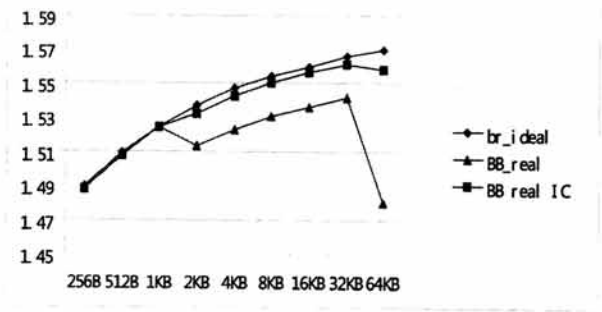
이와 더불어 기본 블록의 시작 주소와 분기 명령어 사이의 간격을 Interval Check 모듈이 검사한다. Interval Check 모듈은 두 예측기로부터 들어온 입력 값 중 하나를 전달



(그림 8) IC_ESP *gshare* 예측기

(bypass)하는 역할을 한다. 이때 두 입력이 동시에 들어온 경우는 주 예측기의 결과에 우선권을 두게 한다. 주 예측기의 결과가 아직 출력되지 않은 상황에서는 보조 예측기의 결과를 활용하며, 예측 지연 시간 이후에 출력된 주 예측기의 결과값이 보조 예측기의 결과값과 상이할 경우는 해당 예측 결과를 overriding 하게 된다. 본 논문에서는 기본적으로 *gshare* 예측기를 활용하였으므로, 이를 기반으로 한 IC_ESP 예측기를 제안하였다. 하지만 이는 분기 명령어의 주소값을 활용하는 어떠한 형태의 분기 예측기에서도 확장되어 적용할 수 있다.

본 논문에서 추가적으로 제안된 IC_ESP 예측기를 활용한 실험 결과가 (그림 9)에 제시되어 있다. (그림 9)에 제시된 결과는 (그림 6)과 (그림 7)에 제시된 결과와 동일한 실험 조건이며, 전체 벤치마크 프로그램의 평균치를 소개하였다. *br_ideal*은, 전술한 바와 같이, 분기 예측기의 구성에 관계없이 항상 1 사이클의 예측 지연을 가지는 이상적인 분기 예측기를 의미하며, *BB_real*은 기본 블록의 시작 주소를 활용하는 방식으로, (그림 9)에 제시된 결과는 (그림 7)에 제시된 평균 결과값이다. 한편, *BB_real_IC*는 *BB_real* 방식에서 Interval Check 기능을 추가적으로 활용하는 방식이다. 즉, 기본 블록 시작 주소와 분기 명령어 사이의 0 cycle 거리를 가지는 경우에 대한 지연 시간을 보정해 주는 방식에 대한 결과이다. 참고로 (그림 9)에서는 *br_real*의 실험 결과는 제외하였다. 그래프에서 X축은 PHT 크기이며, Y축은 IPC를 의미한다.



(그림 9) IC_ESP 예측기의 IPC 향상 정도 (X축: PHT 크기, Y축: IPC)

(그림 9)에서 나타난 바와 같이, *BB_real_IC* 기법이 *BB_real* 방식에 비해 성능상 보다 더 우수함을 알 수 있다. 더 나아가 *BB_real_IC* 기법은 *br_real* 기법과 사실상 거의 유사한 결과를 보인다고 할 수 있다. 제안된 *BB_real_IC* 기법은 분기 예측기의 크기에 관계없이, *br_real*에 비하여 평균 0.25% 이내의 IPC 차이를 보이고 있다. 결론적으로 본 논문에서 최종 제안된 기법은 기존의 방식에 비해 평균 4.2%, 최대 10.1%의 IPC 성능 향상을 가져 왔다.

끝으로, 복구(recovery)에 대하여 언급하면 다음과 같다. 우선, BB_SA와 실제 분기 명령어 사이의 거리가 가변적이므로, 이에 따른 복구 (recovery) 구간도 변할 것이다. 하지만 이는 본 논문의 제안으로 인한 특수한 고려 사항은 아니다. 다시 말해, 복구 회로 (recovery logic)는 투기적 실행 (speculative execution)을 지원하는 프로세서에 이미 포함되어 있는 기능이다. 그리고 복구 회로에서는 복구해야 할 명령어의 숫자가 가변적이라는 사실에 대하여, 심지어 정확히 몇 싸이클 후에 결과가 나온다는 것을 알고 있는 경우에서조차도, 해당 싸이클 동안 실행되는 명령어의 숫자는 데이터 종속성 (data dependency) 등의 원인에 의해 가변적이게 된다. 따라서 복구 모듈은 다른 여타의 투기적 실행을 지원하는 프로세서의 기존 복구 회로를 활용할 수 있다.

5. 결 론

분기 예측의 정확도가 점점 더 중요해 지면서 정확도의 향상을 위한 다양한 기법들이 제안되었지만, 예측 지연 시간은 간과되는 경향이 있었다. 그러나 실제로는 예측 지연 시간이, 미미한 정도의 정확도 향상보다는, 오히려 성능 향상에 더 큰 영향을 미친다는 사실이 연구되고 있다.

본 논문에서는 이러한 예측 지연 시간 문제를 해결하고자 조기 예측 기법을 제안하였다. 조기 예측 기법은 분기 명령

어의 주소 대신 그것과 일대일 대응이 되는 기본 블록의 시작 주소를 이용하여, 분기 주소가 사용될 때 보다 일찍 예측을 시작함으로서, 예측 지연 시간을 숨길 수 있었다. 또한, 기본 블록 시작 주소와 분기 명령어간의 거리를 분석한 결과, 71.2%가 1 사이클 이상임을 확인하였다. 특히 이 가운데 13.4%는 2 사이클, 29.3%는 3 사이클 이상이었다. 따라서 조기 예측을 적용하면 71.2%의 분기 명령어에 대하여 1~3 사이클 혹은 그 이상의 예측 지연 시간을 숨길 수 있게 된다. 또한 기본 블록의 시작 주소와 분기 명령어 사이의 짧은 간격이 발생하는 경우에 대해서도, Interval Check 기법을 보조 예측기와 함께 사용하게 함으로 해서, 추가적인 예측 지연 시간이 필요치 않게 된다.

실험 결과 제안된 기법은, 분기 예측 테이블의 크기와 상관없이 항상 1 사이클의 접근 시간을 가지는 이상적인 분기 예측기의 경우에 매우 근접한 결과를 보여 주었다. 그리고 분기 명령어의 주소와 기본 블록의 주소가 동시에 인출되는 28.8%의 분기 명령어에 대해서도 성능상의 향상을 관찰할 수 있었다. 제안된 기법은, 1 사이클의 예측 지연 시간을 가지는 이상적인 분기 예측기 대비, 평균 0.25% 이내의 매우 근소한 IPC 차이를 가지며, 기존의 현실적인 방법과 비교하여, 평균 4.2% 최대 10.1%의 IPC 향상을 가져 왔다. 본 논문에서 제안된 기법은 향후 클럭 속도가 더욱 빨라지고 공정의 크기가 더욱 미세해 짐으로 인해 1 사이클의 접근 시간이 요구되는 PHT의 크기가 더욱 작아질 때, 보다 더 큰 의미를 가질 수 있게 될 것이다.

참 고 문 헌

[1] D. A. Patterson and J. L. Hennessy, "Computer architecture: a quantitative approach" 4th edition, Morgan Kaufman, 2007.

[2] 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.

[3] D. A. Jimenez, "Reconsidering Complex Branch Predictor",In Proceedings of the 9th International Symposium on High Performance Computer Architecture, pp.43-52, 2003.

[4] O. J. Santana, A. Ramirez, M. Valero, "Latency Tolerant Branch Predictors," In Proceedings of Innovative Architecture for Future Generation High-Performance Processors and Systems, pp.30-39, 2003.

[5] G. H. Loh, "Revisiting the Performance Impact of Branch Predictor Latencies," In Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software, pp.59-69, 2006.

[6] S. S. Muchnick, "Advanced Compiler Design Implementation," Morgan Kaufman, 1997.

[7] A. Seznec and A. Fraboulet, "Effective ahead pipelining of instruction block address generation," In Proceedings of the 30th ISCA, 2003.

[8] A. Seznec, S. Jourdan, P. Sainrat, P. Michaud, "Multiple-block ahead branch predictors," In Proceedings of the 7th ASPLOS, pp.116-127, 1996.

[9] D. A. Jimenez, S. W. Keckler, and C. Lin, "The impact of delay on the design of branch predictors," In Proc. 33rd Int'l Symp. on Microarchitecture, pp.67-76, 2000.

[10] A. Falcon, O. Santana, A. Ramirez and M. Valero, "Tolerating Branch Predictor Latency on SMT," ISHPC2003, LNCS 2858, pp.86-98, 2003.

[11] E. Rotenberg, S. Bennett, J. Smith, "Trace Cache: a low latency approach to high bandwidth instruction fetching," 29th IEEE MICRO, 1996.

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

[13] SimpleScalar LLC to serve and project, http://www.simplescalar.com/

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

[15] J. W. Kwak and C. S. Jhon., "High Performance Embedded Branch Predictor by Combining Branch Direction History and Global Branch History," IET Computer & Digital Techniques, Vol.2, Issue 2, pp.142-154, 2008.

[16] R. Thomas, M. Franklin, C. Wilkerson and J. Stark, "Improving Branch Prediction By Dynamic Dataflow-based Identification of Correlated Branches From a Large Global History," In Proc. of the International Symposium on Computer Architecture, pp.314-323, 2003.

곽 종 욱

e-mail : kwak@ynu.ac.kr

1998년 경북대학교 컴퓨터공학과(학사)

2001년 서울대학교 컴퓨터공학과(공학석사)

2006년 서울대학교 전기컴퓨터공학부(공학박사)

2006년~2007년 삼성전자 SOC 연구소 책임연구원

2007년~현 재 영남대학교 컴퓨터공학과 조교수

관심분야 : 컴퓨터 구조, 저전력 내장형 시스템, 고성능 컴퓨팅 등

김 주 환

e-mail : jhkim@mslab.snu.ac.kr

2001년 서울대학교 컴퓨터공학과(학사)

2004년 서울대학교 전기컴퓨터공학부(석·박사 연계과정이수)

2004년~현 재 서울대학교 전기컴퓨터공학부 박사과정

관심분야 : 고성능 컴퓨터 구조, 병렬 컴퓨팅, 차세대 모바일 시스템 등