Victim BTB를 활용한 히트율 개선과 효율적인 통합 분기 예측

주 영 상† · 조 경 산††

요 약

본 논문에서는 파이프라인 프로세서의 분기 명령어 처리 성능 향상을 목적으로, BTB의 미스율을 줄이고 분기 예측의 정확도를 개선하기 위해 victim cache를 활용한 2-단계 BTB 구조를 제안한다. 2-단계 BTB는 기존의 BTB에 작은 크기의 victim BTB를 추가한 구조로, 적은 비용으로 BTB 미스율을 개선하고, 동적 예측(dynamic prediction)과 정적 예측(static prediction)이 함께 사용되는 기존의 통합 분기 예측(Hybrid Branch Prediction) 구조의 예측 정확도를 높이도록 운영된다. 본 논문에서 제안된 2-단계 BTB에 의한 성능 개선을 4개 벤치마크 프로그램에 대한 trace-driven 시뮬레이션을 통해 검증한 결과, 기존의 BTB에 비해 2.5~8%의 비용 증가로 BTB 미스율이 26.5% 개선되고, 기존의 gshare에 비해 64%의 비용 증가로 예측 정확도는 26.75% 개선되었다.

Improving Hit Ratio and Hybrid Branch Prediction Performance with Victim BTB

Young-Sang Joo† · Kyung-San Cho††

ABSTRACT

In order to improve the branch prediction accuracy and to reduce the BTB miss rate, this paper proposes a two-level BTB structure that adds small-sized victim BTB to the conventional BTB. With small cost, two-level BTB can reduce the BTB miss rate as well as improve the prediction accuracy of the hybrid branch prediction strategy which combines dynamic prediction and static prediction.

Through the trace-driven simulation of four benchmark programs, the performance improvement by the proposed two-level BTB structure is analysed and validated. Our proposed BTB structure can improve the BTB miss rate by 26.5% and the misprediction rate by 26.75%.

1. 서 론

파이프라인 프로세서의 성능은 실행 흐름을 변경하는 실행 제어 명령어들에 의해 큰 영향을 받는다. 실행 제어 명령어에는 조건 분기 명령어, 무조건 분기 명령어, 간접 분기 명령어, 함수 호출 등이 있는데, 조건 코드 값에 의해 분기 방향이 결정되는 조건 분기 명령어가 파이프라인의 성능 저하에 가장 큰 원인이 된다[1].

파이프라인 프로세서에서 명령어가 명령어 페치(IF), 명령어 해석(DE), 오퍼랜드 페치(OF), 실행(EX)

※ 이 논문은 단국대학교 대학 연구비 지원에 의하여 연구되었음.
† 준 회 원 : 단국대학교 대학원 전산통계학과
†† 종신회원 : 단국대학교 전산통계학과 교수
논문접수 : 1998년 4월 21일, 심사완료 : 1998년 8월 4일

의 순서로 실행된다고 가정하면, 조건 분기 명령어는 실행 단계까지 분기될지 여부를 정할 수 없으므로 분기가 발생하는 경우에는 분기 지연(delay)이 발생한다. 그러나 분기 방향을 미리 예측하여 분기 목적 명령어를 바로 페치 할 수 있다면, 이 같은 분기 지연을 줄일 수 있다.

또한, 분기 명령어가 분기로 예측되는 경우에 분기될 목적 명령어의 주소를 알지 못하면, 목적 주소가 알려질 때까지 파이프라인이 정지되는 지연이 발생한다. 이와 같은 분기 지연을 예방하기 위해, 분기 명령어의 주소와 분기될 목적 주소(또는 목적 명령어)를 저장하는 버퍼인 BTB(Branch Target Buffer)를 사용하여 분기 목적 명령어를 미리 페치할 수 있다[2].

분기 여부를 실행 단계 이전에 결정하기 위해서는 분기 방향을 미리 예측하는 기법을 사용하는데, 이를 분기 예측 전략(Branch Prediction Strategy)이라 한다. 분기 예측 전략은 정적 예측(static prediction)과 동적 예측(dynamic prediction)의 두 가지 방법으로 구현된다. 정적 예측은 모든 분기 명령어에 대해 항상 분기됨(taken) 또는 분기되지 않음(not taken)으로 예측하며, 동적 예측은 분기 명령어의 과거 분기 실행 기록(history)을 사용하여 과거 분기 패턴에 따라 명령어 수행 시에 예측한다.

각 예측 방법의 특성을 이해하기 위해 조건 분기 명령어들의 특성을 분석할 필요가 있다. 조건 분기 명령어의 분기 특성은 분기 명령어가 분기되는 확률(pr(br))로 분석하거나, 특정한 길이의 분기 패턴이 발생하는 빈도로 분석하는 방법이 있다[3][4].

〈표 1〉 분기 명령어의 분기 확률에 따른 분류
〈Table 1〉 Percentage of each branch class

pr(br) / 벤치마크	0% ~10%	10% ~ 90%	90% ~100%
SPECint92	22%	34%	44%
SPECint95	28%	34%	38%

<표 1>은 SPEC 벤치마크 프로그램의 실행 시에 각 분기 명령어가 분기되는 확률에 따른 분포를 보여주고, <표 2>는 과거에 분기가 수행된 패턴의 길이에 따라 분기 명령어를 분류한 것이다 [4]. 패턴 길이가 2라는 의미는 분기됨을 1로 분기되지 않음을 0으로 표시할 때 과거 분기 기록이 101010과 같이 분기됨, 분기되지 않음이 반복되는 형태로 나타나는 경우를 의미한다.

〈표 2〉 분기 패턴의 발생 빈도
〈Table 2〉 Frequency of branch patterns

분기 패턴 길이	발생 빈도
1	50%
2	4%
3	2%
4	4%
그 외(5 이상)	40%

앞의 표들을 분석해 보면, 대부분 분기되거나(90~100%) 분기되지 않는(0%~10%) 강편향 분기(strongly biased branch)가 전체 분기 명령어의 60% 이상이며, 패턴 길이 1(과거 분기 기록이 1111 또는 0000과 같이, 항상 분기되거나 또는 분기되지 않는 경우)인 분기 명령어의 비율은 50% 임을 알 수 있다. 이러한 강편향 분기 명령어들은 정적 예측 전략을 적용하는 것이 예측 정확도가 높게된다.

그러나 약편향 분기(weakly biased branch)인 경우는 정적 예측 전략보다는 동적 분기 예측 전략이 보다 높은 정확도를 얻을 수 있다. 따라서, 한가지 분기 예측 전략으로 강편향 분기 명령어와 약 편향 분기 명령어들을 모두 정확히 예측하는 것은 불가능하다.

본 연구에서는 BTB의 히트 율을 높이기 위해 2-단계 BTB를 제안하고, 각 분기 명령어의 특성에 따라 가장 적합한 예측 방법을 선택하여 예측 정확도를 높이는 기존의 통합 분기 예측 전략[4][5][6]을 2-단계 BTB와 함께 사용하여 보다 높은 예측 정확도를 얻을 수 있음을 시뮬레이션을 통해 분석한다.

본 논문의 구성은 다음과 같다. 2 장에서는 기존의 BTB 구조와 동적 예측 전략 및 통합 분기 예측 방법들을 설명하고, 3 장에서는 BTB 미스율을 개선하기 위한 2-단계 BTB 구조를 제안하고, 4 장에서는 본 연구에서 제안하는 기법의 검증 및 분석에 사용된 벤치마크 프로그램들의 특성을 소개하고, 제안 구조의 성능을 분석하며, 5 장에서는 2-단계 BTB가 통합 분기

예측 전략의 예측 정확도에 미치는 영향을 분석하고, 6 장의 결론으로 마무리한다.

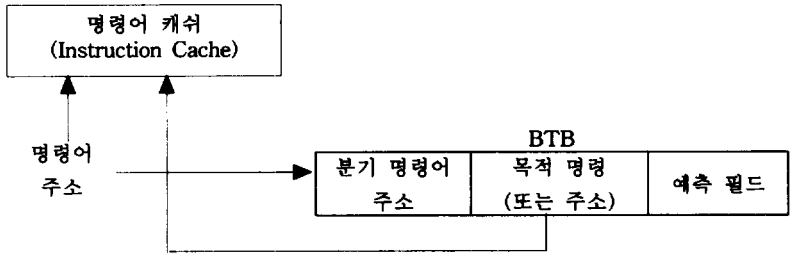
2. 관련 연구(BTB 및 통합 분기 예측의 기존 연구)

BTB는 그림 1과 같이 명령어의 주소로 참조되며, 분기 명령어의 주소와 분기될 목적 주소(또는 목적 명령) 그리고 예측 필드로 구성된다[11]. 예측 필드에는 각 분기 명령어의 분기 여부를 예측하는데 필요한 정보가 저장되며, 그 내용은 예측 방법에 따라 다르게 구현될 수 있다. 최근에는 BTB에서 예측 필드를 분리하여 별도로 저장하는 구조가 사용되며 이를 BPT(Branch Prediction Table)이라 한다. BPT는 하나의 표로 구성되거나, 또는 기록표(History table)와 패턴표(Pattern table)로 구성된 2-단계 표(two-level table)로 구현된다[7].

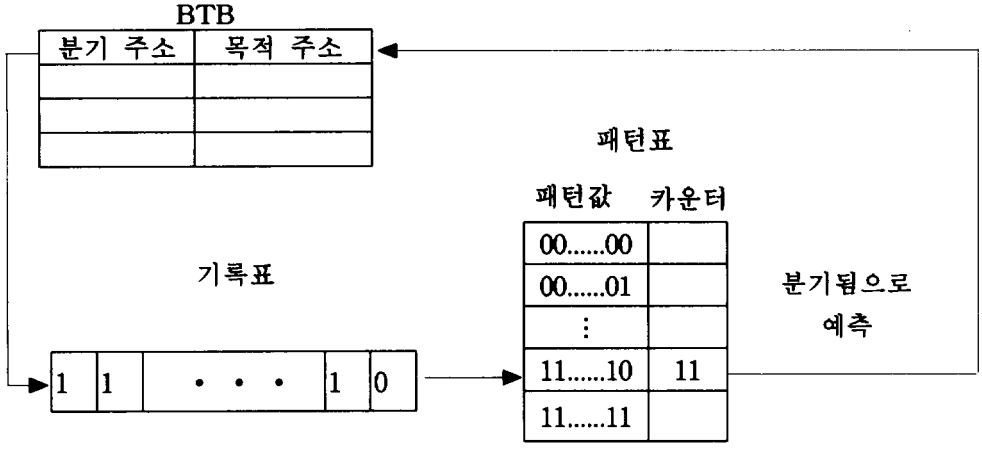
분기 명령어가 페치 또는 해석 단계에서 분기됨으로 예측되더라도 BTB에서 미스가 발생할 경우에는 목적 주소를 알 수 없기 때문에, 분기될 목적 주소를 구하는 지연이 발생한다. 따라서 BTB 미스율을 줄이기 위해 모든 분기 명령어를 BTB에 저장하는 대신에, 조건 분기 명령어만 저장하는 방법이 사용되어 왔다[1].

BTB 미스로 발생하는 지연은 구조에 따라 다르나, 보통 1-2 사이클로 알려져 있으며, 슈퍼 파이프라인 구조에서는 이 지연이 더 커지므로 성능에 큰 영향을 미치게된다. 따라서 BTB 미스율이 분기 처리 성능의 중요한 요소이며, 본 연구에서는 victim cache를 활용한 2-단계 BTB 구조로 BTB 미스율을 개선하는 방법을 제안한다.

BTB와 함께 분기 예측에 주로 사용되는 동적 예측 전략은 예측에 사용되는 정보를 저장하고 이를 처리하는 방법에 따라, 2-비트 크기의 포화 카운터(satura-



(그림 1) BTB 구조
(Fig. 1) Branch Target Buffer



(그림 2) 2-단계 예측기 구조
(Fig. 2) Two-level Predictor

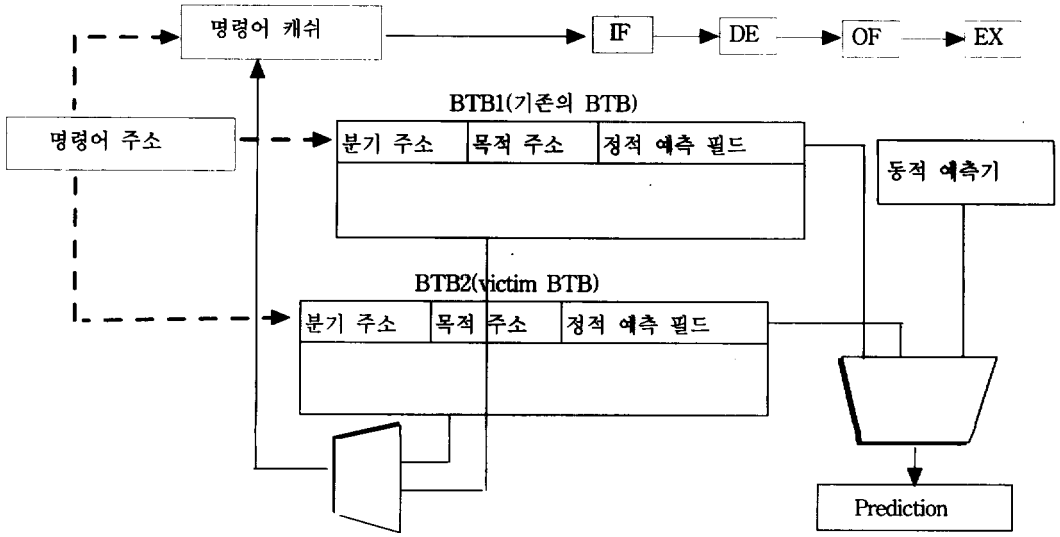
ting counter)를 사용하는 전략과 기록표와 패턴표로 구성된 2-단계 예측기(predictor)를 사용하는 전략으로 나눌 수 있다. 카운터를 사용하는 경우에는 분기 명령어가 실제로 분기되면 카운터 값을 증가시키고 분기되지 않으면 카운터 값을 감소시키며, 카운터의 값이 지정된 값 이상이면 분기로 그렇지 않으면 분기 안됨으로 예측한다. 2-단계 예측기에서 기록표는 분기 명령어들의 분기 기록을 저장하며, 패턴표는 특정한 분기 기록을 갖는 분기 명령어의 과거 분기 패턴을 이용하여 분기를 예측한다.

그림 2는 2-단계 예측기의 구조를 보인 것이다. 기록표는 분기 기록 쉬프트 레지스터로 구현되며, 패턴표 내의 각 항목은 패턴값과 카운터로 구성된다. 기록표의 과거 분기 패턴에 의해 인덱스 되는 패턴표 항목의 카운터 값이 지정된 값 이상이면 분기됨으로 예측한다. 2-단계 예측기의 구현 시에 기록표와 패턴표가 각 분기 명령어에 대해 공유되는 경우를 전역 예측(Global Prediction)이라 하고, 각기 따로 유지하는 경우를 지역 예측(Local 또는 Private Prediction)이라 한다[7]. 그림 2는 전역 기록표와 전역 패턴표를 갖는 경우이다.

2-단계 예측 방법에서 기록표 또는 패턴표가 여러 분기 명령어들에 의해 공유되는 경우에, 2 개 이상의 분기 명령어가 하나의 패턴표로 매핑될 수 있으며, 이를 간섭(interference)이라 한다. 간섭은 서로 다른 주 분기 방향(majority direction)을 갖는 두 개의 분기 명령어가 동일한 패턴표의 항목으로 매핑 되는 경우에 예측이 실패하는 주요 요인이 된다[3].

각 분기 명령어의 특성에 따라 가장 적합한 예측 방법을 선택하여 예측 정확도를 높이려는 목적으로 개발된 통합 분기 예측 방법은 정적 예측 전략과 2 개 이상의 동적 예측 전략을 결합하여 간섭이 발생하는 것을 회피할 수 있다. 예를 들면, 정적 예측 방법(filtering)과 모든 분기 명령어들이 하나의 기록표를 공유하는 전역 예측 방법(gshare) 및 각 분기 명령어가 별도의 기록표를 갖는 지역 예측 방법(pshare)을 결합하여, 패턴 길이 1인 분기 명령어들은 정적으로 예측하고, 그 외의 분기 명령어는 선택기(selector)를 통해 전역 예측 또는 지역 예측 방법 중 하나로 예측한다[4].

따라서, 통합 분기 예측을 사용하는 대부분의 기존 연구는 선택기를 사용하여 각 분기 명령어에 대해 정적 예측, 전역 예측 및 지역 예측 방법 중에서 가장 적절한 예측 기법을 선택하는 방법이 주로 사용된다. 그러나 선택기 구현에 대한 추가 비용이 필요하고, 지역 예측 방법은 각 분기 명령어에 대한 별도의 정보를 저장하기 때문에 상대적으로 높은 비용이 요구된다.



(그림 3) 2-단계 BTB 구조
(Fig. 3) Two-level BTB

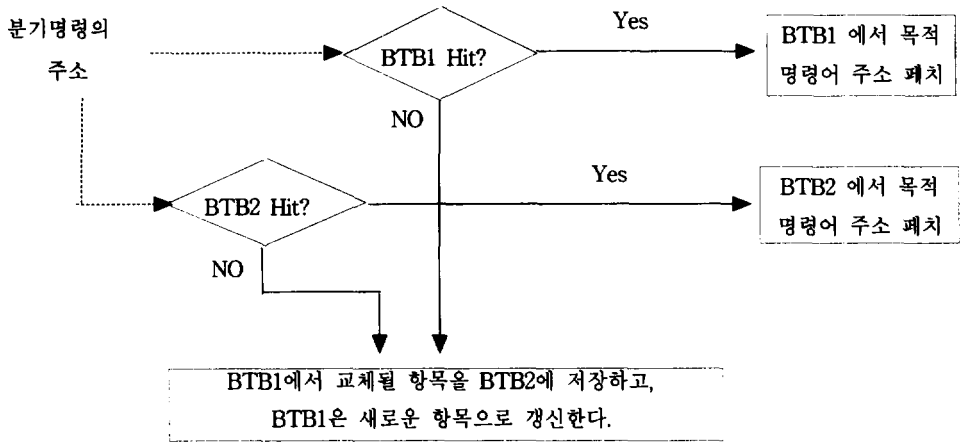
지역 예측 방법은 패턴 길이 2~4인 분기 명령어들을 예측하는데 유용하지만[4], <표 1>에서 이들은 단지 10%에 불과함을 알 수 있다.

3. Victim BTB 구조 및 운영 방법

분기 예측을 사용하는 파이프라인 프로세서는 BTB에서 미스가 발생할 경우 지연이 발생한다. 따라서 파이프라인의 성능을 향상시키기 위해서는 BTB의 미스율을 줄이는 것이 예측의 정확도를 높이는 만큼 중요하다. 실제 프로세서의 구현에서는 작은 크기의 BTB가 사용되기 때문에 큰 미스율이 발생하여, 파이프라인의 성능 저하를 초래할 수 있다. 따라서 작은 크기의 BTB에서 미스율을 줄일 수 있도록, 캐쉬 메모리에 대해 Jouppi가 제안한 victim cache와 유사한 victim BTB를 사용하는 2-단계 BTB를 제안한다[10]. Jouppi는 직접 사상(direct-mapped) 캐쉬와 1~5개의 entry로 구성된 완전 연관 사상(fully-associative mapped) victim cache를 함께 사용하는 구조를 제안하였다. 이 구조는 캐쉬 미스가 발생하면, victim cache를 참조하여 히트 시에는 그 항목을 캐쉬로 swapping 한 후 캐쉬에서 프로서로 명령어를 페치한다. 따라서, 1 사이클의 미스 페널티가 발생한다

본 연구에서 제안하는 BTB 구조는 앞에서 설명한 캐쉬 구조와 유사하게 그림 3과 같이 구성된다. 첫 단계 BTB1은 기존의 BTB(2-way 또는 4-way 셋 연관 매핑)로, 두 번째 단계 BTB2는 작은 항목 수(수십 개)의 완전 연관 victim BTB로 구성된다. victim cache에서 캐쉬로 swapping 한 후 프로세서로 명령어가 페치되어 발생하는 페널티를 없애기 위해, 제안 구조에서는 그림 4와 같이 BTB1과 BTB2 사이의 swapping은 하지 않는다. 분기 명령어의 페치 시에 명령어 캐쉬와 동시에 BTB1과 BTB2에 접근하여, BTB1에서 히트 시에는 BTB1의 해당 목적 주소를 참조하고, BTB1이 미스하고 BTB2가 히트 시에는 BTB2에서 목적 주소를 얻는다. BTB1과 BTB2에서 모두 미스인 경우는 기존의 BTB에서와 같이 한 항목이 replacement 정책에 의해 교체되고 새로 페치된 항목을 BTB1에 저장한다. 그러나 기존의 BTB에서는 교체되는 항목이 버려지는 반면에 제안한 구조에서는 이 항목이 BTB2에 저장되고 BTB2의 한 항목이 LRU replacement 정책에 의해 버려진다.

프로세서에서 구현되는 BTB는 크기가 적고 direct 또는 2-way set associative 방식으로 구현되므로, 서로 다른 명령어가 같은 항목으로 매핑되는 conflict miss가 발생할 확률이 높다. 따라서 BTB miss를 줄이기 위하여 BTB 크기를 증가시키거나 set associativity 정도를 증가시키는 방법을 사용할 수 있다. 그러나 이런 방법들은 고 비용을 요구하므로, 본 연구에서 제안한 victim BTB는 실제 프로세서 구현에 활용할 수 있도록 적은 비용으로 conflict miss로 인한 BTB 미스를



(그림 4) 2-단계 BTB의 알고리즘
(Fig. 4) Algorithm of Two-level BTB

줄일 수 있다는 장점이 있다. 제안 구조의 미스율 개선 정도와 기존 BTB와의 비용대비 개선 정도는 4장에서 분석되며, 정적 예측 전략을 사용하는 통합 분기 예측 전략의 성능에 대한 영향은 5장에서 분석된다.

4. 작업부하 및 제안구조의 성능분석

본 연구에서는 제안된 운영 구조에 의한 BTB 미스율 및 예측 정확도를 분석하기 위해 trace-driven 시뮬레이션을 수행하였다.

시뮬레이션에 사용된 입력 자료는 SUN사에서 제공하는 shade를 사용하여 trace 하였다. shade는 SPARC v8과 SPARC v9에서 수행되는 분기 명령어에 관한 정보를 trace하고 시뮬레이트 할 수 있는 C library를 제공한다[8][9]. 본 연구에서는 shade의 여러 함수들을 사용하여 분기 명령어 주소, 분기 목적 주소, 분기 방향 등을 추출하도록 trace 프로그램을 작성하였다. 작업부하는 shade를 활용한 trace 프로그램과 benchmark 프로그램들을 다음의 실행 환경을 통해 수집하였다.

커널 구조: Sun4m
응용 구조: Sparc
커널 버전: SunOS 5.5.1

작업부하의 생성을 위해 사용된 벤치마크 프로그램은 Colorado 전산학과에서 C 언어로 작성한 benchmark 프로그램 중 espresso, gs, make, p2c를 사용하였다. 각 벤치마크의 프로그램 특성과 수행 특성은 <표 3>과 같다.

시뮬레이션은 각 응용 프로그램에 대해 수행 조건 분기 명령어 중에서 1000 만개씩을 사용하여, 여러 구조의 BTB에 대한 미스율과 다양한 예측 전략들의 예측 정확도를 측정하도록 구현하였다.

<표 4>는 기존의 BTB와 제안된 2-단계 BTB의 미스율을 비교한 결과이다. 기존의 BTB 구조에서 BTB 크기를 2 배로 증가시키면 espresso는 48.3%, gs는 43.0%, make는 45.6%, p2c는 29.3%의 비율로 미스율이 개선된다. 즉, BTB 크기를 2 배로 증가시키는 경우

〈표 3〉 사용된 벤치마크 프로그램
〈Table 3〉 Benchmark programs

벤치마크프로그램	프로그램 특성	작업정도	명령어 수	조건 분기 명령어 비율
espresso	PLA 논리 최적화	mlp4	85,539,946	15.3%
gs	포스트 스크립트 페이지-묘사 언어에 대한 해석 수행	large	231,865,158	14.0%
make	GNU make 프로그램	perl	79,996,383	16.4%
p2c	파스칼을 C로 변환	mf	406,976,984	20.0%

〈표 4〉 기존의 BTB와 2-단계 BTB의 BTB 미스율
〈Table 4〉 Misprediction rate of BTB and two-level BTB

		2-way set associative			4-way set associative		
	BTB1의 크기	256	512	1 K	256	512	1K
espresso	1-단계 BTB	9.8%	5.4%	2.6%	7.6%	4.5%	2.0%
	2-단계 BTB	6.0%	3.6%	1.6%	5.8%	3.3%	1.3%
gs	1-단계 BTB	38.5%	25.1%	13.4%	35.5%	25.6%	9.6%
	2-단계 BTB	31.2%	21.5%	9.4%	31.1%	21.9%	5.2%
make	1-단계 BTB	11.9%	10.3%	1.8%	4.6%	3.1%	1.4%
	2-단계 BTB	3.6%	2.6%	1.3%	3.4%	2.5%	0.8%
p2c	1-단계 BTB	25.1%	17.7%	12.2%	19.6%	15.1%	10.1%
	2-단계 BTB	22.9%	16.8%	10.6%	19.4%	14.5%	9.0%

평균 41.2% 미스율이 개선됨을 알 수 있다.

그러나 BTB1의 크기 변화 없이 단자 20 항목 크기의 victim BTB2를 사용하면, espresso는 33.0%, gs는 22.7%, make는 43.3%, p2c는 6.9% 미스율이 개선되므로, 평균 26.5% 미스율이 개선됨을 알 수 있다. 즉, BTB 크기를 2 배로 증가시켜 얻어지는 미스율 개선의 64%를 BTB에 작은 크기의 victim BTB2(2.5~8%의 용량)를 추가하여 얻을 수 있다.

gs, p2c는 espresso, make와 비교해 상대적으로 성능개선 정도가 낮은데, 그 이유는 gs, p2c의 참조 페이지 수가 상대적으로 많으므로 작은 크기의 BTB(256이나 512 항목)에서는 히트율이 매우 낮기 때문이다. 따라서, gs 또는 p2c와 같은 프로그램은 1 K 항목 이상을 갖는 4-way 방식의 BTB를 사용하는 것이 바람직하며, 20개의 victim BTB를 추가할 경우 gs는 46%, p2c는 10% 정도의 개선 효과를 기대할 수 있음을 알 수 있다.

앞의 분석으로부터, 본 장에서 제안한 victim BTB를 활용한 2-단계 BTB는 약간의 비용으로 BTB의 미스율을 개선할 수 있는 효율적인 구조임을 확인할 수 있다.

5. victim BTB를 사용한 효율적인 통합 분기 예측

패턴 길이가 1인 분기 명령어는 정적 예측(또는 지역 예측) 방법으로, 패턴 길이 1-4인 분기 명령어는 지역 예측 방법으로, 그 외는 전역 예측 방법으로 예측하는 것이 정확하다. 지역 예측에서는 분기 명령어가 패턴표에 처음 기록되는 경우에 카운터가 올바른 예측값으로 설정되는 기간동안 정확한 예측을 할 수 없다. 따라서 패턴 길이 1인 분기 명령어(전체 분기 명령어 중에서 50%가 이에 해당)는 지역 예측 보다 정적 예측에서 더 정확히 예측할 수 있다.

전체 분기 명령어 중에서 10% 정도인 패턴길이 2-4인 분기 명령어는 지역 예측 방법이 정확하다. 기존의 연구에서는 통합 분기 예측을 위해 전역 예측과 지역 예측을 결합하거나 또는 정적 예측과 전역 및 지역 예측을 결합하여 사용한다. 그런데, 지역 예측 방법을 사용한 통합 분기 예측은 각 분기 명령어마다 과거 분기 기록을 별도로 유지해야하고 각 분기 명령어마다 가장 적합한 예측 방법을 결정하기 위해 선택기를 사용해야 하므로 고비용이 요구된다. 따라서, 이 구조는 비용면에서 비효율적인 결합이다.

Chang은 filtering이라는 정적 예측 전략을 사용하고, 동적 예측을 위해 gshare의 전역 예측 및 pshare의 지역 예측을 사용하는 통합 분기 예측 구조를 제안하였다[4]. 이 구조에서는 정적 예측 필드를 4 비트 크기의 포화 카운터와 1 비트 크기의 방향 비트(Direction bit)로 구성하였으며, 카운터가 1111이 되는 경우에 방향 비트 값이 1이면 분기됨으로, 0이면 분기되지 않음으로 예측한다. 그 외의 분기 명령어는 BPST (Branch Predictor Selection Table)라는 선택기(selector)를 통해 gshare 또는 pshare 중 하나로 예측한다. BPST에는 각 분기 명령어가 선택할 예측 방법에 대한 정보가 기록된다. 예를 들면, BPST의 각 항목에 2 비트 크기의 카운터를 사용하여 그 값이 2 이상이면 gshare를, 2 미만이면 pshare를 선택하도록 한다. 그러나 이 구조에서는 BTB에서 미스 시 정적 예측을 수행하는 필드(카운터와 방향 비트)가 분기 명령어 주소, 분기 목적 주소 등과 함께 교체된다. 따라서 BTB 미스율이 예측 정확도에 직접 영향을 미침에도 불구하고, BTB 미스율을 줄이려는 연구는 병행되지 못하였다. 또한, 고비용의 지역 예측 방법과 선택기의 구현은 비용 면에서 비효율 적이다.

따라서 본 연구에서는 gshare와 [4]에서 제안된 정적 예측을 결합한 통합 분기 예측 구조를 본 연구의 3장에서 제안한 2-단계 BTB와 결합하여 적은 비용으로 예측 정확도를 높이는 통합 분기 예측 방법을 제안하고, 이로 인한 성능 개선 정도를 분석한다.

〈표 5〉 기존의 분기 예측 방법들과 제안한 구조의 예측 정확도
〈Table 5〉 Prediction accuracy of branch prediction strategies

예측 방법 / 벤치마크	gshare	filtering +gshare	filtering +gshare +pshare	gshare +filtering(BTB1) +filtering(BTB2)
espresso	91.52	92.39	92.54	92.47
gs	91.45	92.91	92.97	93.41
make	92.98	97.98	97.99	98.03
p2c	93.29	93.44	93.57	93.54
평 균	92.30	94.18	94.26	94.36

〈표 6〉 각 분기 예측 방법들의 총 용량(비용)
〈Table 6〉 Total implementation Cost(Capacity) of branch prediction strategies

예측 방법	총 용 량
gshare	12+8K ≈ 8 K bit
filtering+gshare	(12+8K)+(4K+1K) ≈ 13 K bit
gshare +filtering+pshare	(12+8K)+(4K+1K)+(8K+2^{8}*2)+(3K) ≈ 24.5 K bit
gshare +filtering(BTB1)+filtering(BTB2)	(12+8K)+(4K+1K)+(20*5) ≈ 13.1 K bit

<표 5>는 벤치마크 프로그램들의 시뮬레이션을 통해, 기존의 예측 방법들의 예측 정확도와 본 연구에서 제안한 구조(최 우측)의 예측 정확도를 비교한 것이다. <표 6>은 예측에 필요한 각각의 필드들이 다음과 같이 구성되어 있을 때, 각 예측 방법들의 총 비용을 구한 것이다.

- BTB : 4-way 셋 연관 매핑(1 K 항목)
- victim BTB : 완전 연관 매핑(20 항목)
- filtering : 4 bit 카운터, 1 비트 크기의 방향 비트
- gshare : 12 비트 전역 기록표, 2 비트 카운터로 구성된 패턴표
- pshare : 1 K 개의 8 비트 지역 기록표, 2 비트 카운터로 구성된 패턴표
- BPST : 1 K 개의 3 비트 카운터(filtering +gshare +pshare 에서만 필요)

위의 표들에서는 gshare와 나머지 예측 방법들의 예측 정확도를 동일한 비용에서 비교하지 않았으며, 동일한 비용에서 통합 분기 예측 방법들의 예측 정확도가 gshare 보다 높다는 것은 이미 밝혀진바 있다[4]. 본 연구에서는 어떤 통합 분기 예측 방법이 비용 면에서 가장 효율적인지를 분석하였다. gshare만을 사용하여 예측하는 방법과 비교해 gshare+filtering은 총 용량이 63%(8K-->13K) 증가할 때 예측 정확도가 24.42%(92.30-->94.18) 개선되고, gshare+filtering+pshare는 총 용량이 206%(8K-->24.5K) 증가할 때 예측 정확도가 25.45%(92.30-->94.26) 개선된다. 이에 반해 본 연구에서 제안한 방법은 gshare+filtering과 거의 비슷한 비용 증가(64%)로, gshare+filtering 및 gshare+filtering+pshare 보다 높은 예측 정확도의 개선(26.75%)을 얻을 수 있다.

앞의 분석으로부터 지역 예측 방법을 제외하여 선택기가 사용되지 않는 구조가 비용 면에서 효율적이고, 정적 예측에 필요한 필드를 BTB에 포함하는 구조에서는 본 연구에서 제안한 2-단계 BTB를 사용하면 정적 예측 방법으로 예측되는 분기 명령어의 수가 증가되어 예측 정확도를 개선할 수 있으므로, 본 연구에서 제안한 2-단계 BTB에 정적 예측과 전역 예측을 결합하는 통합 분기 예측 구조가 비용 면에서 효과적임을 확인할 수 있다.

6. 결론 및 연구 과제

분기 예측 전략을 사용하는 기존의 연구들에서는 분기 명령어의 주소와 분기될 목적 주소를 저장하기 위해 BTB를 사용하였는데, 예측 정확도와 함께 BTB의 미스율은 파이프라인 프로세서의 성능에 큰 영향을 미친다.

예측 정확도를 개선하기 위해 분기가 발생하는 패턴에 따라 분기 명령어를 분류하면, 패턴 길이 1인 분기 명령어는 정적 예측 방법, 2~4는 지역 예측 방법, 5 이상은 전역 예측 방법에서 각기 정확히 예측 할 수 있다. 이런 이유로 이 예측 방법들을 결합하여 예측하는 통합 분기 예측 전략이 연구되고 있다. 그러나 지역 예측 방법과 전역 예측 방법을 함께 사용하기 위해서는 별도의 선택기가 필요하고, 단지 10%의 분기 명

령어의 예측을 위해 고비용으로 구현되는 지역 예측 방법은 비효율적이다. 따라서 지역 예측 방법과 전역 예측 방법의 결합보다는 정적 예측과 전역 예측 방법을 함께 사용하는 통합 분기 예측이 효율적이다.

정적 예측 방법과 동적 예측 방법을 결합한 통합 분기 예측 방법에서는 정적 예측 방법에 필요한 필드들이 BTB 포함되는 것이 일반적이다. 따라서, BTB에서 미스가 발생하면 정적 예측 방법에 필요한 필드들이 함께 교체되므로 예측 정확도에 영향을 미치게된다. 본 연구에서는 BTB의 미스율을 줄이고 예측 정확도를 개선하기 위해 victim cache를 활용한 2-단계 BTB 구조를 제안하였다.

벤치마크 프로그램들에 대해 BTB 미스율을 분석하면, BTB의 크기를 2 배로 증가시키면 미스율이 평균 41.2% 개선되고, BTB1의 크기는 변화시키지 않고 victim BTB2(20 개)를 추가하면 미스율이 평균 26.5% 개선된다. 즉, 2-단계 BTB는 적은 추가 비용(2.5~8%)으로 BTB 미스율을 개선할 수 있음을 알 수 있다. 또한 본 연구에서 제안한 방법은 gshare+filtering과 거의 비슷한 비용 증가(64%)로, gshare+filtering+pshare 및 gshare+filtering 보다 예측 정확도가 높게 개선(26.75%)된다.

특히, 적은 크기의 BTB를 사용하는 실제 슈퍼 파이프라인 프로세서에서는 BTB 미스율이 높고 미스 시에 큰 분기 지연이 발생하므로, 본 연구에서 제안한 2-단계 BTB가 적은 추가 비용으로 프로세서의 성능을 개선할 수 실용적인 구조임을 알 수 있다.

<표 7>의 예측 방법들은 모든 분기 명령어가 하나의 기록표와 패턴표를 공유하고, 기록표에서 패턴표를 참조하는 인덱싱 방법만 서로 다른 전역 예측 방법들의 여러 변형들이다. 변형 GAg는 기존의 GAg의 인덱싱 방법에서 분기 주소와 기록 값을 결합하는 방식을 변경한 것으로, make를 제외한 다른 프로그램들에서는 변형된 GAg가 최소의 간섭이 발생함을 알 수 있다. 따라서, 간섭이 발생할 수 있는 분기 명령어들을 예측하고 분리하여 서로 다른 방법으로 인덱싱하는 전역 예측 방법 등을 사용하면, 간섭을 예방하는 것이 가능할 것이다. 간섭의 효율적 예방 방법의 개발은 본 연구와 결합되어 분기 예측의 정확도를 높일 수 있는 향후 연구 과제이다.

〈표 7〉 각 전역 예측 방법들의 총 간섭의 수
〈Table 7〉 Number of Interferences in each global prediction strategy

예측방법 / 프로그램	gshare	GAg	변형 GAg
espresso	968627	892269	751143
gs	1526165	2271693	1223155
make	594332	459699	774673
p2c	1555986	122442	1150656

참 고 문 헌

[1] Brad Calder and Dirk Grunward, "Fast & Accurate Instruction Fetch and Branch Prediction," Proc. 21th Ann. Symp. Computer Architecture, pp.2-11, 1994.

[2] Harvey G. Cragon, Memory Systems and Pipelined Processors, Jones and Bartlett Publishers, 1996.

[3] Cliff Young et. al, "A Comparative Analysis of Schemes for Correlated Branch Prediction," Proc. 22th Ann. Symp. Computer Architecture, pp.276-286, 1995.

[4] Po-Young Chang, "Classification-Directed Branch Predictor Design," The University of Michigan, available at http://www.cs.umich.edu., 1997.

[5] S. MCFarling, "Combining branch predictors," Technical Report TN-36, Digital Western Research Laboratory, June 1993.

[6] Eric Sprangle et. al, "The Agree Predictor: A Mechanism for Reducing Negative Branch History Interference," Proc. 24th Ann. Symp. Computer Architecture, pp.284-291, 1997.

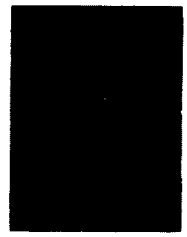
[7] Tse-Yu Yeh and Tale N Patt, "A Comparison of Dynamic Branch Predictors that use Two Levels of Branch History," Proc. 20th Ann. Symp. Computer Architecture, pp.257-266, 1993.

[8] Bob Cmelik and David Keppel, "Shade: A Fast Instruction-Set Simulator for Execution Profiling," Sigmetrics, ACM, pp.138-137, 1994.

[9] Sun microsystem, "shade user's manual," available at http://sw.sun.com/shade.

[10] Norman P. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a small Fully-Associative Cache and Prefetch Buffer," Proc. 17th Ann. Symp. Computer Architecture, pp.364-373, 1990.

[11] 주 영상, 조 경산, "분기 예측과 이중 경로 전략을 결합한 파이프라인 구조에 관한 연구," 정보 처리 논문지, 제3권 제1호, pp.181-190, 1996.



주 영 상

1990년 고려대학교 재료공학과 졸업(학사)

1995년 단국대학교 전산통계학과 졸업(이학석사)

1995년~현재 단국대학교 대학원 전산통계학과 박사과정

관심분야 : 구조론 및 성능평가

조 경 산

1979년 서울대학교 전자공학과 졸업(학사)

1981년 한국과학원 전기 및 전자공학과 졸업(공학석사)

1988년 텍사스대학교(오스틴소재) 전기전산공학과 졸업(Ph.D.)

1988년~1990년 삼성전자 컴퓨터 부문 책임 연구원

1990년~현재 단국대학교 전산통계학과 부교수

관심분야 : 구조론 및 성능평가, 컴퓨터 통신, 시뮬레이션