韓國컴퓨터情報學會 論文誌
第14卷 第10號, 2009. 10.

2009-14-10-1-1

분기 선예측과 개선된 BTB 구조를 사용한 분기 예측 지연시간 은폐 기법

김 주 환*, 곽 종 욱*, 전 주 식**

Branch Prediction Latency Hiding Scheme using Branch Pre-Prediction and Modified BTB

Ju-Hwan Kim*, Jong Wook Kwak*, Chu Shik Jhon**

요 약

현대의 프로세서 아키텍처에서 정확한 분기 예측은 시스템의 성능에 지대한 영향을 끼친다. 최근의 연구들은 예측 정확도뿐만 아니라, 예측 지연시간 또한 성능에 막대한 영향을 끼친다는 것을 보여준다. 하지만, 예측 지연시간은 간과되는 경향이 있다. 본 논문에서는 분기 예측 지연시간을 극복하기 위한 분기 선예측 기법을 제안한다. 이 기법은 분기 장치를 인출 단계에서 분리함으로써, 분기 예측기가 명령어 인출 장치로부터의 아무런 정보도 없이 스스로 분기 예측을 진행 가능하게 한다. 또한, 제안된 기법을 지원하기 위해, BTB의 구조를 새롭게 개선하였다. 실험 결과는 제안된 기법이 동일한 수준의 분기 예측 정확도를 유지하면서, 대부분의 예측 지연시간을 은폐한다는 것을 보여준다. 더욱이 제안된 기법은 항상 1 싸이클의 예측 지연시간을 가지는 이상적인 분기 예측기를 사용한 경우보다도 더 나은 성능을 보여준다. 본 논문의 실험 결과에 따르면, 기존의 방식과 비교했을 때, 최대 11.92% 평균 5.15%의 IPC 향상을 가져온다.

Abstract

Precise branch predictor has a profound impact on system performance in modern processor architectures. Recent works show that prediction latency as well as prediction accuracy has a critical impact on overall system performance as well. However, prediction latency tends to be overlooked. In this paper, we propose Branch Pre-Prediction policy to tolerate branch prediction latency. The proposed solution allows that branch predictor can proceed its prediction without any information from the fetch engine, separating the prediction engine from fetch stage. In addition, we propose newly modified BTB structure to support our solution. The simulation result shows that proposed solution can hide most prediction latency with still providing the same level of prediction accuracy. Furthermore, the proposed solution shows even better performance than the ideal case, that is the predictor which always takes a single cycle prediction latency. In our experiments, IPC improvement is up to 11.92% and 5.15% in average, compared to conventional predictor system.

• 제1저자 : 김주환 교신저자 : 곽종욱
• 투고일 : 2009. 07. 20, 심사일 : 2009. 08. 30, 게재확정일 : 2009. 09. 29.
* 영남대학교 전자정보공학부 조교수 ** 서울대학교 전기컴퓨터공학부 교수

▸ Keyword : 지연시간(latency), 분기 예측(branch predict), 기본 블록(basic clock), 선예측(pre-prediction)

Ⅰ. 서 론

분기 목적지(branch target)가 계산되기 전에, 유용한 명령어들을 인출(fetch)하기 위해, 투기적 실행(speculative execution)을 지원하는 프로세서는 분기 예측(branch predict)의 결과에 의존한다. 그러므로 정확한 분기 예측기는 투기적 실행을 지원하고, 깊은 파이프라인(pipeline)을 가진 마이크로프로세서에서 고성능을 유지하기 위한 필수적인 부분이다[1]. 분기 명령어의 예측을 실패하게 되면, 인출 장치는 프로세서의 실행 장치가 필요로 하는 명령어들을 충분히 공급하지 못하게 된다. 그러므로 분기 예측의 정확도는 통상 예측기의 중요한 요소로 취급된다.

현재까지 대부분의 분기 예측에 관한 연구는 예측 정확도의 향상에 초점을 맞추어 왔다. 현대의 마이크로프로세서에는 높은 분기 예측 정확도를 위해, 큰 규모의 복합적(hybrid)이고 복잡한 분기 예측기가 사용되었다. 분기 예측기가 더 큰 하드웨어 공간을 사용하게 되면, 더 긴 분기 방향 정보(branch history)를 이용할 수 있고, PHT(Pattern History Table)에서의 가명현상(aliasing)을 줄일 수 있기 때문에, 분기 예측의 정확도가 높아지게 된다. 하지만 최근의 연구에 의하면, 분기 예측기의 정확도 향상이 실행 시간에 악영향을 가져오는 경우도 있다[2][3]. 더욱이 현대의 슈퍼스칼라 프로세서들은 클록 속도 또한 점점 더 빨라지고 있다. 이러한 현상은 필연적으로 분기 예측에 필요한 시간이 1 싸이클을 넘어서게 만든다. 1 싸이클을 넘는 분기 예측 지연시간은 인출 장치가 바로 다음 싸이클에 명령어를 인출하지 못하게 만들기 때문에, 시스템의 성능에 치명적인 영향을 미친다[4][5]. 본 논문에서는 분기 예측 지연시간을 극복하기 위해, 분기 선예측 기법을 제안한다. 제안된 기법은 파이프라인에 새롭게 분기 선예측 단계를 만들어서, 분기 예측 장치를 명령어 인출 단계에서 분리시켜 분기 선예측 단계에 집어넣는다. 제안된 기법은 분기 예측 정확도를 유지하면서 효율적으로 분기 예측 지연시간을 은폐시킨다. 뿐만 아니라, 선예측 대기열의 작용으로 1 싸이클에 다수개의 분기 예측을 수행하는 것과 같은 효과를 가짐으로써, 명령어 인출 대역폭을 증가시켜 전체 시스템의 성능 향상에 기여한다. 하지만, 이를 위해 몇몇 구조들을 추가했기 때문에 더 많은 공간이 필요하게 되며, 선예측 가능 여부가 BTB(Branch Target Buffer)의 적중률에 영향을 받는다는 단점이 있다.

논문의 나머지 부분은 다음과 같이 구성된다. 2장에서 관련 연구를 다루고, 3장에서 분기 선예측 기법을 설명하며, 4장에서 실험결과를 보여주고, 5장에서 결론을 내린다.

Ⅱ. 관련 연구

분기 예측 지연시간을 은폐하기 위해 이미 다수의 연구가 제안되었다.

그 중에 하나는 어헤드 파이프라인(ahead pipeline) 예측기이다[2][6][7]. 어헤드 파이프라인 예측기는 분기 예측에 N 싸이클이 소요된다고 하면, N 싸이클 이전에 그 당시 사용 가능한 정보들만을 이용하여 예측을 시작하는 기법이다. N 싸이클 이전에는 예측해야 하는 분기의 주소나 완벽한 분기 방향 정보를 얻을 수 없기 때문에, 이러한 정보의 부족으로 정확한 엔트리를 가져올 수 없다. 대신에 가능한 모든 후보자들을 가져와서, N 싸이클 이후, 즉 예측 결과가 사용될 시점에 N 싸이클 동안 추가된 정보를 사용하여 그 후보자들 중에 최종 엔트리를 선택한다. 이러한 방식은 현재의 분기 명령어 주소를 분기 예측에 활용할 수 없기 때문에, 필연적으로 분기 예측 정확도가 떨어진다. 또한, 분기 예측 실패(mis-prediction) 직후에도 분기 예측 지연시간을 은폐하기 위해 많은 양의 체크포인트 파일(checkpoint file)과 복구 회로(recovery logic)가 필요하다. 어쨌든, 이 기법은 분기 예측 지연시간을 은폐하기 위해 분기 예측 정확도를 희생하고 있다.

분기 예측 지연시간을 은폐하기 위한 또 하나의 기법은 예측 오버라이드(prediction override) 기법이다[5]. 이 기법은 항상 두 개의 예측을 동시에 시작한다. 첫 번째 예측은 빠르고 단순한 예측기를 사용하고, 두 번째 예측은 느리지만 더 정확한 주예측기를 사용한다. 첫 번째 예측 결과를 사용하여 명령어 인출을 먼저 진행하다가, 두 번째의 주예측기 결과가 나오면 두 예측 결과를 서로 비교하여 다를 경우, 그 동안의 인출을 무효화하고 주예측기의 결과를 사용하여 다시 인출을 진행한다. 오버라이드 기법의 경우, 빠른 첫 번째 예측기의 결과가 틀렸을 경우, 주예측기의 예측 지연시간을 은폐할 수 없다.

최근에 분기 예측 지연시간을 은폐하기 위한 기법으로 ESP 예측기가 제안되었다[8]. 이 기법은 예측기가 동적으로 기본 블록(basic block)의 시작 주소를 찾아내어, 그 기본 블록의 시작 주소를 분기 명령어 주소 대신에 분기 예측에 사용하는 기법이다. 이 기법은 동일한 수준의 분기 예측 정확도를

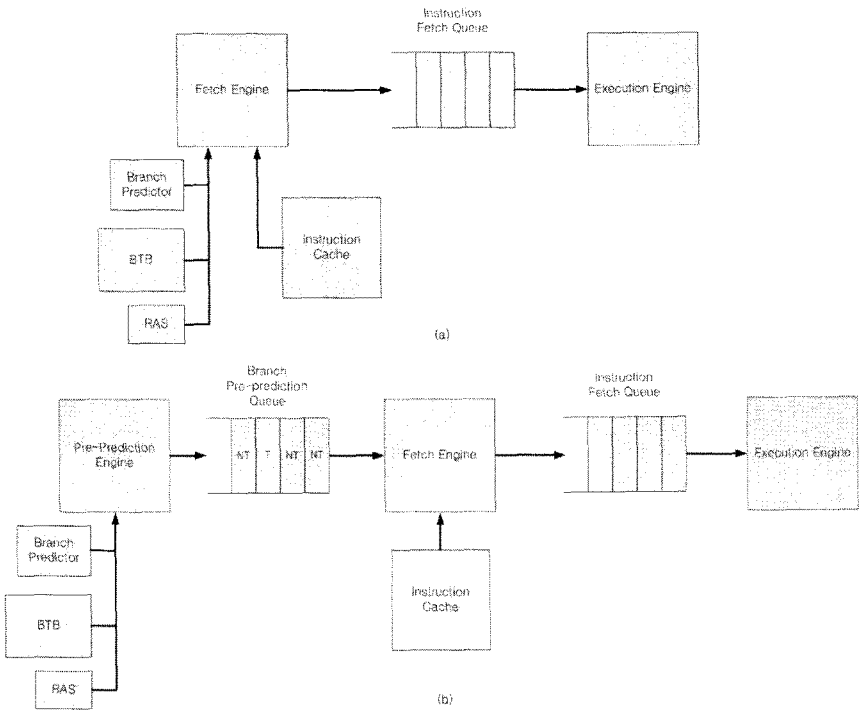

그림 1. 분기 선예측 시스템
Fig. 1. Branch Pre-Prediction

유지하면서 효과적으로 분기 예측 지연시간을 은폐한다. 하지만, 기본 블록의 시작 명령어가 분기 명령어로부터 충분히 떨어져있지 않는 경우, 분기 예측 지연시간을 완전히 은폐하지 못하기 때문에, 추가적인 성능 향상의 가능성이 남아있다.

그 외에도, 현재, 훌륭한 정확도를 가지는 많은 최신 분기 예측기들이 제안되었다[9][10][11][12]. 하지만, 문제는 이들 중 대부분이 긴 분기 예측 지연시간을 가진다는 것이다. 우리의 지식 범위 안에서 가장 정확한 분기 예측기는 TAGE 예측기 이다[9]. TAGE 예측기는 전역 분기 방향/경로 정보(global branch/path history)와 분기 주소를 입력으로 하는 독립적인 함수들을 인덱스로 사용하는 다수의 예측 테이블(prediction table)을 가지고 있다. 예측 테이블에 태그(tag)를 사용했으며, 분기 정보의 길이를 기하학적으로 확장하는 방법을 사용한다. 본 논문에서 우리의 실험에 TAGE 예측기가 사용되었다.

III. 분기 선예측 기법

이 장에서는 분기 예측 지연시간을 은폐하기 위한 분기 선예측 기법을 제안한다.

3.1 선예측 장치

우선, 그림 1(a)는 일반적인 마이크로프로세서 시스템의 전단부(front-end)를 보여준다. 이러한 구조에서 명령어 인출 장치(fetch engine)는 분기 예측기, BTB, RAS(Return Address Stack)를 사용하여 다음에 인출될 블록의 시작 주소를 계산한다. 그 이후에 인출 장치는 명령어들을 명령어 캐시(instruction cache)에서 가져와서 명령어 인출 대기열(instruction fetch queue)에 집어넣는다. 이러한 일반적인 구조에서는 분기 명령어 혹은 기본 블록의 시작 명령어가 명령어 인출 장치에 나타났을 때에서야, 분기 예측을 시작할 수 있으며, 1 싸이클에 단 하나의 분기 예측만 실행 가능하다.

그림 1(b)는 제안된 구조인 분기 선예측 기법의 전단부를 보여준다. 인출 단계 이전에 선예측 장치라고 불리는 새로운 파이프라인 단계를 추가하였으며, 이러한 방법으로 다음 기본 블록의 시작주소를 예측하는 부분을 인출 단계에서 분리하였다. 선예측 장치는 분기 예측기, BTB, RAS를 사용하여 다음 기본 블록의 시작 주소를 계산하고, 분기 예측 결과와 다음 기본 블록의 시작 주소를 선예측 대기열(pre-prediction queue)에 집어넣는다. 분기 명령어가 인출 장치에 나타나게 되면, 인출 장치는 선예측 대기열에서 분기 예측 결과와 다음 기본 블록의 시작 주소를 가져온다. 이 정보를 이용하여 인출

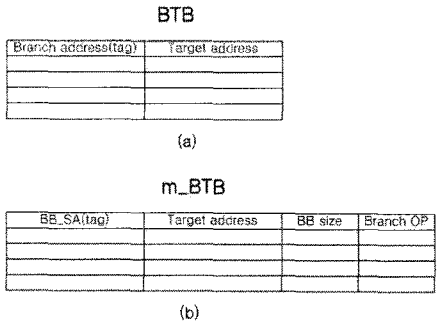

그림 2. BTB와 개선된 BTB
Fig. 2. BTB and Modified BTB

장치는 정확한 명령어들을 명령어 캐시에서 가져와서 명령어 인출 대기열에 집어넣는다.

3.2 선예측 대기열

그림 1(b)에서 보여지듯이, 선예측 대기열은 매 엔트리 마다 2가지 종류의 내용물을 가진다. 그중 하나는 Taken/NotTaken을 가르키는 분기 예측 결과이고, 나머지 하나는 다음 기본 블록의 시작 주소이다. 분기 예측 결과가 Taken일 경우는 다음 기본 블록의 시작 주소는 목적 주소(target address)가 되고, 분기 예측의 결과가 NotTaken일 경우는 다음 기본 볼록의 시작 주소는 해당 블록 바로 다음 명령어의 주소가 된다. 해당 블록 바로 다음 주소는 현재 기본 블록의 시작 주소에 현재 기본 블록의 크기를 더해서 계산할 수 있다.

3.3 개선된 BTB 와 분기 예측기

분기 선예측 기법을 지원하기 위해, BTB의 구조를 개선하였다. 그림 2(a)는 일반적인 BTB의 구조를 나타낸다. 일반적인 BTB는 분기 명령어의 주소로 인덱스(index) 하며, 오직 목적 주소 값만을 가지고 있다. 그림 2(b)는 개선된 BTB 구조를 나타낸다. 개선된 BTB는 기본 블록의 시작 주소(BB_SA)로 인덱스 하며, 목적 주소 값 외에 추가적으로 기본 블록의 크기 값과 분기 명령어의 명령 코드(OP code) 값을 가진다. 개선된 BTB의 동작 방식은 다음과 같다. 분기 결과가 Taken이든 NotTaken이든 관계없이, 현제의 BB_SA 값과 기본 블록 크기 값, 목적 주소 값을 사용하여 다음 BB_SA 값을 계산 가능하다. 그럼으로써, 선예측 장치는 다음 기본 블록이 인출되기 전에, BTB와 분기 예측기의 인덱스 값을 미리 알 수 있다. 분기 명령 코드 값은 인출 장치로부터의 아무런 정보 없이 다른 구조들의 값을 올바르게 유지하기 위해 필요하다. 예를 들면, RAS 값을 유지하기 위해서는 해당 분기가 함수 호출(function call)인지 아닌지와 함수 복귀(function return)인지 아닌지를 알아야 하며, 분기 방향 정보 값을 바르게 유지하기 위해서는 해당 분기가 조건 분기인지 아닌지를 알아야 한다.

분기 예측기도 입력 벡터로 분기 명령어 주소 대신 BB_SA를 사용하도록 수정하였다.

표 1. 실험 환경
Table 1. Simulation Environment

Processor Core	
Issue width	8 instructions
ROB, issue queue	64
Load-store queue	32
Commit width	8 instructions
Fetch Unit	
Fetch width	8 instructions
Fetch Queue	16
Pre-Prediction queue	8
Branch Target Buffer	1024 sets, 4 ways
Return Address Stack	32 entries
Mis-prediction penalty	9 cycles
Fast branch predictor	Bimodal, 1 K bits
Main branch predictor	TAGE, less than 32 KB
Main predictor latency	3 cycles

3.4 선예측 장치의 동작 방식

본 논문에서는 선예측 장치의 작동 방식에 관련 연구에서 언급된, 예측 오버라이드 기법의 개념을 적용하였다. 그림 3은 선예측 장치에서 분기 예측기와 BTB의 동작을 보여주는 시간 도표이다.

시점 (a)에서는, BB1의 BB_SA 값을 이용하여, BTB의 접근을 시작한다. 또한, 빠른 예측기와 느린 주예측기의 예측도 동시에 시작한다.

시점 (b)에서는, BB1의 BTB 접근과 빠른 예측기의 예측이 종료된다. 선예측 장치는 빠른 예측기의 결과 값과 BB1의 바로 다음 명령어의 주소 값과 목적 주소 값을 알고 있기 때문에, 다음 BB_SA(BB2) 값을 계산 가능하다. 선예측 장치는 빠른 예측기의 결과 값과 다음 BB_SA 값을 선예측 대기열에 집어넣고, 시점 (a)에서처럼, BB2의 BTB접근과 분기 예측을 시작한다.

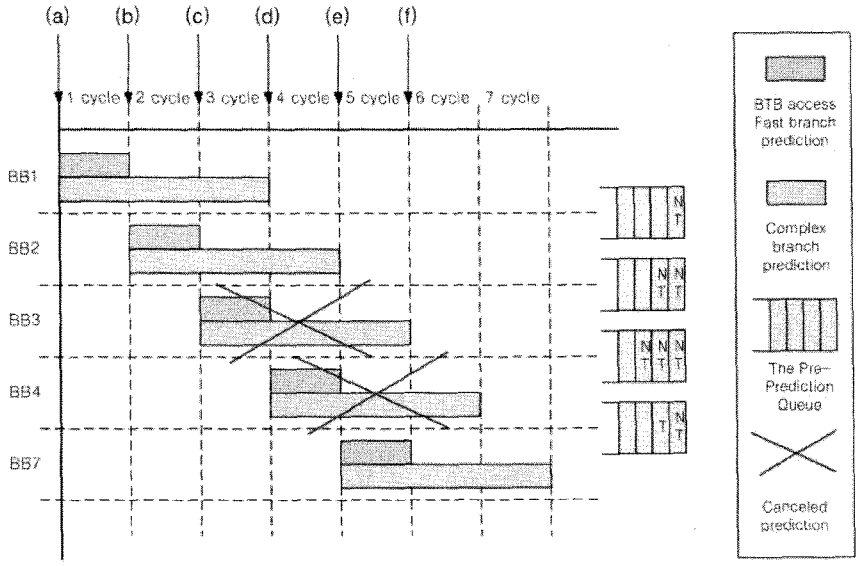

그림 3. 선예측 장치의 시간 도표
Fig. 3. Timing Diagram in Pre-Prediction Engine

시점 (c)에서는, 시점 (b)에서처럼 다음 BB_SA 값을 계산하여, 대기열에 집어넣고, 다음 예측을 시작한다.

시점 (d)에서는, 시점 (a)에서 시작한 BB1의 주예측기의 예측이 종료된다. 주예측기의 결과 값과 BB1의 빠른 예측기의 결과 값을 비교하는데, 이 시나리오에서는 두 결과 값은 일치하게 된다. 그러면, 더 이상 아무것도 하지 않고, 다음 블록(BB4)의 BTB 접근과 예측을 시작한다.

시점 (e)에서는, BB2의 주예측기의 예측이 종료된다. 시점 (d)에서와 달리, BB2의 주예측기의 결과 값과 빠른 예측기의 결과 값은 서로 일치 하지 않는다. 그러면 선예측 장치는 선예측 대기열에서 BB3과 BB4의 결과 값을 제거하고, 예측 실패에 의해서 집어넣어진 값을 올바르게 수정한다. 선예측 장치는 올바른 주예측기의 결과 값과 BB2의 BB_SA 값, BB2의 크기 값, BB2의 목적 주소 값을 사용하여 다음 BB_SA(BB7) 값을 다시 계산하여, 예측을 계속 수행한다.

IV. 성능 평가

이 장에서는 제안된 기법의 성능을 평가한다.

본 논문의 실험에서는 사건 구동형 모의실험기(event-driven simulator)인 SimpleScalar를 사용하였다. SimpleScalar는 구체적인 명령어 레벨 모의실험과 싸이클 단위 모의실험을 가능하게 해주는 강력한 모의실험 환경을 제공해준다. 우리는 SimpleScalar를 제안된 구조에 맞게 수정하였으며, 구체적인 실험 환경은 표 1에 나타나있다. 성능 평가 기준 프로그램(benchmark program)은 SPEC CPU 2000의 정수형 프로그램(CINT)에서 임의로 선택하여 사용하였다. 그리고, 우리의 지식 범위 내에서 가장 정확한 분기 예측기인 TAGE 예측기를 실험의 주예측기로 사용하였다. 실제로 TAGE 예측기에 반복문 예측기(loop predictor)를 추가한 L-TAGE 예측기는 CBP-2(the 2nd Championship Branch Prediction Competition)에서 우승한 바 있다. 그리고, TAGE 예측기는 분기 예측에 3 싸이클이 소요된다.

4.1 예측 정확도

우선, 제안된 기법의 예측 정확도를 살펴보자. 기존의 구조에서 분기 명령어의 주소 값을 BB_SA 값으로 바꾸었고, 파이프라인 단계에 선예측 장치를 새롭게 추가하였다. 이러한 변화가 분기 예측 정확도에 미치는 영향은 표 2에 나타나있다.

표 2에서, "br 1 cycle"은 항상 1 싸이클의 예측 시간이 걸리고, 분기 주소 값을 사용하는 이상적인 분기 예측기를 뜻한다. "BB 1 cycle"은 역시 항상 1 싸이클의 예측 시간이 걸리는 이상적인 분기 예측기이지만, 분기 주소 값 대신 BB_SA 값을 사용하는 분기 예측기를 뜻한다. "BB override"는 BB_SA 값을 사용하고 현실적인 예측 지연시간을 반영하였으

표 2. 분기 예측 정확도
Table 2. Branch Prediction Accuracy

	164 gzip	175 vpr	176 gcc	181 mcf	186 crafty	197 parser	252 eon	254 gap	255 vortex	256 bzip2	average
br 1 cycle	96.2	90.96	97.87	96.21	97.28	97.6	98.31	98.98	99.95	97.9	97.126
BB 1 cycle	96.2	90.94	97.84	96.22	97.26	97.58	98.31	98.98	99.95	97.9	97.118
BB override	96.2	90.93	97.84	96.21	97.26	97.58	98.31	98.97	99.95	97.9	97.115
Pre-Prediction	96.2	90.93	97.82	96.21	97.26	97.58	98.31	98.96	99.95	97.9	97.112

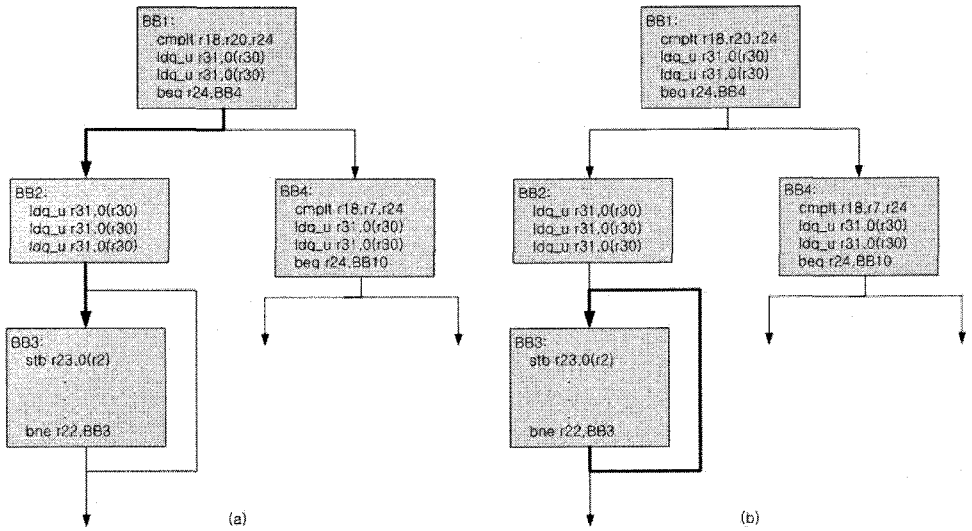

그림 4. 가명현상 예제
Fig. 4. Aliasing Example

며, 마지막으로, "Pre-Prediction"은 BB_SA값을 사용하고, 본 논문에서 제안한, 오버라이드 기술을 적용한 선예측 장치를 사용하는 분기 예측기를 뜻한다.

표 2에 나타나는 실험 결과를 살펴보면, 분기 예측 정확도는 거의 동일하다. "br 1 cycle"과 "BB 1 cycle"의 평균 차이 값은 0.008%이다. 이 차이 값은 BB_SA를 검출하는 동적 알고리즘에 의해 가명 현상(aliasing)이 발생하기 때문이다. 가명 현상의 예는 그림 4에 나타나있다.

그림 4를 살펴보면, BB3의 분기 명령어에 도달하는 경로가 (a)와 (b)로 2가지로 나뉜다. (a)의 경우, BB3의 분기 명령어를 예측하기 위해 BB2의 시작 주소가 사용되고, (b)의 경우, BB3의 분기 명령어를 예측하기 위해 BB3의 시작 주소가 사용된다. 그럼으로 BB3의 분기 명령어를 예측하기 위해 더 많은 엔트리가 할당되게 된다. 하지만, 이러한 현상은 부정적인 효과만 있는 것이 아니라, 긍정적인 효과도 있다. (a)의 경우와 (b)의 경우에 BB3의 분기 명령어의 행동 방식이 서로 다를 가능성이 있다. 두 경우에 BB3의 분기 명령어의 행동 방식이 서로 다를 경우에는, 두 경우에 서로 다른 엔트리가 할당된 것이 오히려 예측 정확도에 도움을 주게 된다. 실제로 181.mcf의 경우 약간 이지만, BB_SA 값을 사용하였을 경우 예측 정확도의 향상이 있다.

"BB override"는 "BB 1 cycle"과 비교하여 예측 정확도가 평균 0.003% 떨어진다. 이 경우, 예측 정확도가 떨어질 이유가 전혀 없다. 다만, 명령어 실행의 흐름이 바뀌어서 그 영향

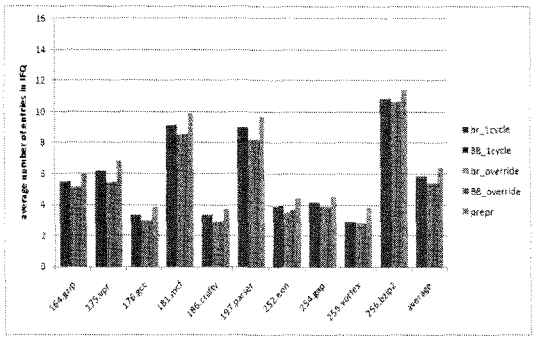

그림 5. 명령어 인출 대기열의 사용률
Fig. 5. Utilization of Instruction Fetch Queue

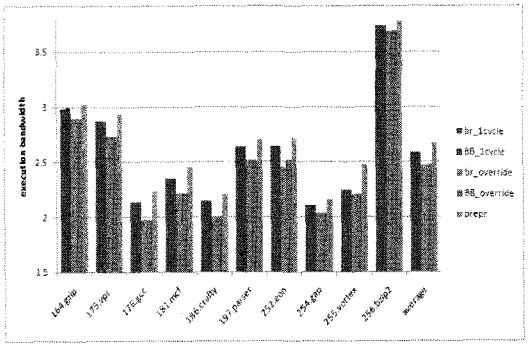

그림 6. 실행 대역폭
Fig. 6. Execution Bandwidth

을 받았을 가능성은 있다. 이것은 다시 말하면, 사건 구동형 모의 실험의 오차 범위라고 얘기 할 수 있다.

"BB override"와 "Pre-Prediction" 사이에도 평균 0.003%의 정확도 하락이 발생한다. 이것은 실행 중 분기(in flight branch)의 개수가 증가했기 때문이다. 실행 중 분기의 개수가 늘어나게 되면, 분기의 행동 방식이 바뀌었을 때, 새로운 행동 방식을 분기 예측기에 반영하는 데 필요한 시간이 늘어나게 된다.

종합하여, "br 1 cycle"과 "Pre-prediction"의 분기 예측 정확도의 차이는 0.014% 이다. 이 정도 값은 실질적으로 같다고 할 수 있다. 이 정도의 매우 작은 차이는 L2 크기 혹은 LSQ 크기 등과 같이 분기 예측과 관계없는 실험 인자를 바꾸었을 때에도 발생할 수도 있다.

4.2 명령어 인출 대기열의 사용률과 실행 대역폭

다음은 명령어 인출 대기열의 사용률과 실행 대역폭이다. 이 두 평가 척도는 인출 장치의 성능과 밀접한 영향이 있다.

그림 5와 그림 6은 분기 선예측 시스템에서의 명령어 인출 대기열의 사용률과 실행 대역폭을 보여준다. "1cycle"은 분기 예측에 항상 1 싸이클이 소요되는 이상적인 예측기를 뜻하며, "override"는 관련 연구에서 언급된 예측 오버라이드 기법을 사용했음을 의미한다. "br"은 분기 예측에 분기 명령어의 주소 값을 사용했음을 뜻하며, "BB"는 분기 명령어의 주소 값 대신에 BB_SA 값을 사용했음을 의미한다. 마지막으로 "prepr"은 제안된 기법을 사용했음을 의미한다.

그림 5를 살펴보면, "prepr"의 명령어 인출 대기열의 평균 엔트리 개수가 "br_1cycle"과 비교하여 18.95% 증가했다. 이는 분기 선예측 시스템이 평균적으로 사이클당 더 많은 명령어를 인출한 다는 것을 뜻한다. 이러한 명령어 인출 대기열의 사용률 증가는 명령어 실행 대역폭의 증가로 이어진다. 그림 6을 살펴보면, 평균 실행 대역폭이 8.00% 증가한 것을 볼 수 있다. 실행 대역폭은 분기 예측 실패 경로와 같은 투기적 실행 경로를 포함한 싸이클당 평균 명령어 실행 개수이다.

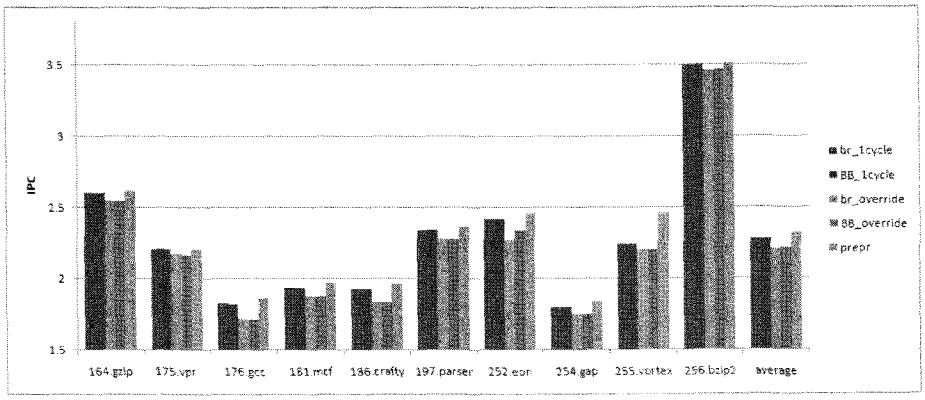

그림 7. 분기 선예측 시스템의 IPC
Fig. 7. IPC of Branch Pre-Prediction system

4.3 IPC 향상

다음은 IPC 향상을 보여준다. IPC는 전체 시스템의 성능을 나타내는 평가 척도 이다.

그림 7은 분기 선예측 시스템의 IPC를 보여준다. "br_override"와 비교하여, 분기 선예측 시스템은 평균 5.15%, 최대 11.92%의 성능 향상을 보여준다. 제안된 기법은 항상 1 사이클의 예측 지연시간을 가지는 이상적인 분기 예측기를 사용한 경우보다 2.00% 더 뛰어난 성능을 보여준다. 더욱이 예측 지연 시간 은폐 기법을 전혀 사용하지 않은 시스템과 비교하면, 36.60%의 IPC 향상을 보여준다.

이러한 성능 향상의 주요 원인은 선예측 대기열의 작용 때문이다. 이상적인 예측기조차, 1 싸이클에 1개의 분기 명령어만을 예측 할 수 있다. 제안된 기법도 1 싸이클에 1개의 분기 명령어만을 예측할 수 있는 건 마찬가지지만, 선예측 대기열의 작용으로 1 싸이클에 2개 이상의 분기 명령어를 예측 하는 것과 같은 효과를 가져온다. 긴 데이터 종속성(data dependency)과 데이터 캐시 접근실패(data cache miss) 혹은 다른 원인으로 실행 장치(execution engine)가 명령어들을 효율적으로 소모하지 못하고 있으면, 선예측 장치의 분기 예측 결과 값이 계속 선예측 대기열에 쌓이게 된다. 이런식으로 쌓여있는 분기 예측 결과 값은 실행 장치가 다시 실행을 재계하여 빠르게 명령어들을 소모시켜갈 때 사용되게 된다. 이 경우 선예측 대기열에서 1 싸이클에 2개 이상의 예측 결과를 가져올 수 있다. 이는 마치 1 싸이클에 2개 이상의 분기 예측을 수행하는 것과 유사한 효과를 가져온다. 이러한 현상은 매우 큰 기본 블록 뒤에 작은 기본 블록들이 따라올 때도 나타난다. 매우 큰 기본 블록 때문에, 한동안 인출 장치에는 분기 명령어가 나타나지 않게 되고, 그 동안 선예측 대기열에는 분기 예측 결과 값들이 쌓이게 된다. 매우 큰 기본 블록의 인출이 끝나고, 뒤 따르는 작은 기본 블록들을 인출하기 시작하면, 선예측 대기열에 미리 쌓여있는 분기 예측 결과 값들을 사용하여, 1 싸이클에 2개 이상의 분기 명령어를 인출 가능하게 된다.

하지만, 선예측 대기열에서 1 싸이클에 여러개의 분기 예측 결과 값들을 가져 올 때 Taken 값이 섞여 있으면, 인출 주소 값의 흐름이 불연속적이게 되어서, 1 싸이클에 여러 번의 L1 캐시의 접근을 요구하게 된다. 본 논문의 실험에서는 공정성을 위해서 1 싸이클에 여러 번의 접근이 가능한 캐시를 설계하지 않았다. 그럼으로써, 선예측 대기열은 NotTaken 결과 값들에 한해서만 1 싸이클에 여러개의 분기 예측 결과 값을 가져 올 수 있는 제약을 가지게 되었다.

그림 7을 살펴보면, 175.vpr과 256.bzip2는 다른 프로그램에 비해 성능 향상의 폭이 적고, 255.vortex는 다른 프로그램에 비해 성능 향상의 폭이 크다. 175.vpr의 경우 다른 프로그램에 비해, 분기 예측 정확도가 낮다. 이는 잦은 명령어 실행 흐름의 수정을 가져오고, 결과적으로 선예측 기법의 장점을 충분히 이용하기 전에, 선예측 대기열을 자주 플러쉬(flush) 시키게 된다. 256.bzip2의 경우, 다른 프로그램에 비해 긴 데이터 종속성과 너무 많은 L2 캐시 접근 실패를 가진다. 그로인해 ROB 엔트리가 꽉 차는 경우가 다른 프로그램에 비해 2배 이상으로 자주 발생하고, 이에 따라, 실행 장치가 충분한 명령어들을 실행하지 못해서, 인출 장치가 인출 대역폭(fetch width)을 완전히 활용하지 못하는 경우가 자주 발생한다. 결과적으로 분기 선예측 시스템이 1 싸이클에 2개 이상의 분기를 인출 가능하게 해주지만, bzip2에서는 1싸이클에 2개 이상의 분기를 인출하는 경우가 자주 발생하지 않게 된다. 255.vortex의 경우, 분기 예측 정확도도 다른 프로그램에 비해 높고, L2 캐시의 접근 실패도 다른 프로그램보다 적다. 그래서, ROB 엔트리가 꽉 차는 경우가 다른 프로그램보다 반이하로 적고, 결과적으로 더 많은 성능 향상을 보여준다.

V. 결론

현대의 프로세서 아키텍처에서 정확한 분기 예측은 시스템의 성능에 지대한 영향을 끼친다. 예측 정확도뿐 만 아니라, 예측 지연시간 또한 성능에 막대한 영향을 끼치지만, 예측 지연시간은 간과되는 경향이 있다. 본 논문에서는 분기 예측 지연시간을 극복하기 위한 분기 선예측 기법을 제안한다. 이 기법은 분기 장치를 인출 단계에서 분리함으로써, 분기 예측기가 명령어 인출 장치로부터의 아무런 정보도 없이 스스로 분기 예측을 진행 가능하게 한다. 또한, 제안된 기법을 지원하기 위해, BTB의 구조를 새롭게 개선하였다. 본 논문의 실험 결과에서 제안된 기법은 동일한 수준의 분기 예측 정확도를 유지하면서, 대부분의 예측 지연시간을 은폐한다는 것을 보여준다. 더욱이 제안된 기법은 항상 1 싸이클의 예측 지연시간을 가지는 이상적인 경우보다도 더 나은 성능을 보여준다. 본 논문의 실험 결과에 따르면, 기존의 방식과 비교했을 때, 최대 11.92% 평균 5.15%의 IPC 향상을 가져온다.

제안된 기법은 BTB에 새로운 내용 값을 더 추가하고, 분기 예측기를 수정함으로써, 1 싸이클에 1개 이상의 분기 예측이 가능하게끔 확장이 가능하다. 하지만, 이러한 수정은 디자인 복잡도(design complexity)를 폭발적으로 증가시킨다. 제안된 기법은 인출될 명령어의 주소를 명령어 캐시에 접근하기 전에 미리 알 수 있기 때문에, 캐시 프리패치(cache prefetch) 기법과 조합할 경우 더 큰 성능 향상을 가져 올 것으로 기대된다.

참고문헌

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

[2] D. A. Jimenez, *"Reconsidering complex branch predictors,"* In Proceedings of the 9th HPCA, pp. 43-52, 2003.

[3] Santana, O.J. et al. "Latency Tolerant Branch Predictors," In Proceedings of Innovative Architecture for Future Generation High-Performance Processors and Systems, pp. 30-39, 2003.

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

[5] 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.

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

[7] A. Seznec et al., "Effective ahead pipelining of instruction block address generation," In Proceedings of the 30th ISCA, pp. 241-252, 2003.

[8] J. W. Kwak, J. H. Kim, S. T. Jhang and C. S. Jhon, *"Early Start Prediction to Tolerate Branch Prediction Latency,"* Information an International Interdisciplinary journal, Vol. 11, No. 5, 2008.

[9] A. Seznec, "The L-TAGE Branch Predictor," Journal of Instruction-Level Parallelism, Vol. 9, May, 2007.

[10] D. Jimenez, *"Piecewise liner branch prediction,"* In Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture, Dec, 2003.

[11] D. JiMenez and C. Lin, *"Neural methods for dynamic branch prediction,"* ACM Transactions on Computer Systems, Nov. 2002.

[12] P. Michaud, *"A ppm-like, tag-based predictor,"* Journal of Instruction Level Parallelism, Vol. 7, Apr. 2005.

[13] Hongliang Gao and Huiyang Zhou, *"PMPM: Prediction by Combining Multiple Partial Matches,"* Journal of Instruction-Level Parallelism, Vol. 9, May, 2007.

[14] Yasuo Ishii, *"Fused Two-Level Branch Prediction with Ahead Calculation,"* Journal of Instruction-Level Parallelism, Vol. 9, May, 2007.

[15] Yasuyuki Ninomiya and Kôki Abe, "A3PBP: A Path Traced Perceptron Branch Predictor Using Local History for Weight Selection," Journal of Instruction-Level Parallelism, Vol. 9, May, 2007.

[16] Hans Vandierendonck and Andre Seznec, "Speculative Return Address Stack Management Revisted," ACM Transaction on Architecture and Code Optimization, Vol. 5, No. 3, Article 15, 2008.

[17] Falcon Ayose, Santana Oliverio J., Ramirez Alex, Valero Mateo, "A latency-conscious SMT branch prediction architecture," International journal of high performance computing and networking, Vol. 2, No. 1, pp. 11-21, 2004.

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

[19] J. W. Kwak, J.-H. Kim, and C. S. Jhon, "The Impact of Branch Direction History combined with Global Branch History in Branch Prediction," IEICE Transactions on Information and System, Vol. E88-D, No. 7, pp. 1754-1758, July 2005.

[20] Kaveh Aasaraai, Amirali Baniasadi and Ehsan Atoofian, "Computational and storage power optimizations for the O-GEHL branch predictor," Proceedings of the 4th international conference on Computing frontiers, pp. 105-112, May, 2007.

[21] 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.

[22] McFarling, S., "Combining branch predictors.

Tech. Rep. TN-36m," Digital Western Research Lab., June, 1993.
[23] SimpleScalar LLC, *http://www.simplescalar.com/*
[24] SPEC CPU Benchmarks, *http://www.specbench.org/*

저 자 소 개

김 주 환

2001 : 서울대학교 공학사
2001 - 현재 :
서울대학교 석박사 통합과정
관심분야 : 컴퓨터 구조, 고성능 컴퓨팅

곽 종 욱

1998 : 경북대학교 공학사
2001 : 서울대학교 공학석사
2006 : 서울대학교 공학박사
2007 : 삼성전자 SOC 연구소 책임 연구원
2007 - 현재 :
영남대학교 전자정보공학부 조교수
관심분야 : 컴퓨터 구조, 임베디드 시스템

전 주 식

1974 : 서울대학교 이학사
1976 : 한국과학기술원 이학석사
1982 : University of Utah 이학박사
1994 - 현재 : 서울대학교 전기.컴퓨터 공학부 교수
관심분야 : 시스템 설계, 모바일 시스템, 저전력 컴퓨팅