분기 예측과 이중 경로 전략을 결합한 파이프라인 구조에 관한 연구

주 영 상† 조 경 산††

요 약

본 논문에서는 파이프라인 구조에서 분기 명령어로 인해 발생하는 제어적 문제를 개선하기 위한 분기 예측 및 다중 경로 전략을 분석하여, 기존 예측 전략에서 간과하였던 지연 사이클의 수를 줄일 수 있도록 예측 전략과 이중 경로 전략이 결합된 새로운 구조 및 전략을 제시한다. 또한, 기존의 예측 전략에서 예측 성공 확률의 측도로 사용하던 BEP 대신에 시스템 성능을 평가할 수 있는 CPI를 사용하여 기존의 제안들에 대해 본 논문에서 제시된 구조 및 전략의 성능 개선을 제시한다.

A Study on Pipelined Architecture with Branch Prediction and Two Paths Strategy

Young Sang Joo† Kyungsan Cho††

ABSTRACT

Pipelined architecture improves processor performance by overlapping the execution of several different instructions. The effect of control hazard stalls the pipeline and reduces processor performance. In order to reduce the effect of control hazard caused by branch, we proposes a new approach combining both branch prediction and two paths strategy. In addition, we verify the performance improvement in a proposed approach by utilizing system performance metric CPI rather than BEP.

1. 서 론

프로세서의 명령어 수행 성능을 평가하기 위한 측도로 프로그램을 실행하는 시간인 CPU 처리 시간은 다음과 같이 정의된다[1].

CPU 처리 시간=명령어 수×CPI(사이클 수/명령어)×사이클 시간 (1−1)

신속한 명령어 수행을 위해 CPU 처리 시간을 줄일 수 있는 방법들에 대한 다양한 연구가 진행되어 왔다. CPU 처리 시간을 줄이는 방법 중에서 특히 CPI를 줄이는 방법으로 프로세서(또는 프로세서의 각 장치)들을 여러개 사용하는 다중 프로세서 구조와 명령어 수행 과정을 여러 세부 동작 단계로 나누고 각 단계들이 동시에 수행되도록 하여 성능을 향상시키는 파이프라인 구조가 있다.

K개의 독립된 단계로 구성된 파이프라인 프로세서의 speedup은 N개의 명령어를 실행한 경우에 K*N/(K+N−1)이 된다. 이때, N의 값이 무한히 크면 speedup=K가 되므로 파이프라인 프로세서의 성능은 독립된 단계의 수만큼 향상된다고 할 수 있다. 그러나 실제 구현에서는 다음과 같은 세 가지 문제 때문에 파이프라인 내의 명령어 처리 과정이 일시 정지될 수 있다[2].

(1) 구조적 문제(Structural hazard) : 프로세

* 이 논문은 1995년도 교내 연구비 지원에 의하여 연구되었음
† 준 회 원 : 단국대학교 전산통계학과 박사과정
†† 종신회원 : 단국대학교 전산통계학과 부교수
논문접수 : 1995년9월25일, 심사완료 : 1995년11월30일

서가 충분한 자원 (cache, buffer, register 등)을 갖고 있지 않을 때.

(2) 자료적 문제(Data hazard) : 앞선 명령어의 수행으로 생성된 자료의 결과가 다음 명령어에 영향을 미칠 때.

(3) 제어적 문제(Control hazard) : 함수 호출/반환, 분기 명령 등이 실행될 때.

세 가지 요인들 중에서 제어적 문제(특히 조건 분기 명령어)가 파이프라인의 가장 큰 성능 저해 요소가 된다[3]. 제어적 문제를 발생시키는 조건 분기 명령어는 분기의 수행 결과인 목적 주소가 실행 단계에서 결정될 때까지, 다음 실행될 명령어가 올바르게 채취(fetch)되었는지 알 수 없다. 즉, 분기 명령어가 실제로 분기된다면 이미 파이프라인 내에 채취되어진 다른 명령어들은 쓸모 없게 되는데, 이를 "Pipeline Bubble"이라 한다[4]. 따라서 새로운 목적 명령어를 채취하기 위해 파이프라인이 정지된 만큼 사이클(이를 분기로 인한 페널티라 한다)이 낭비된다. 이 분기로 인한 페널티를 줄이기 위해 파이프라인 동결(Pipeline freeze)[5], 정적 및 동적 예측(Static and Dynamic prediction)[5, 6], 지연 분기(Delayed branch)[5, 7], 다중 경로(Fetch multiple path)[5]등의 다양한 전략들이 제시되어 왔다.

예측 전략은 분기의 결과를 미리 예측하여 목적 명령어를 채취하는 전략으로 예측의 정확도를 높이는 것이 가장 중요한 과제로 연구되어 왔다. 그러나 예측 전략은 예측이 잘못되면 무조건 분기 지연이 발생하므로, 본 연구에서는 최근에 제안된 새로운 Branch Prediction Table(BPT)를 사용한 예측 전략과 다중 경로 전략을 결합하여 잘못된 예측으로 인한 분기 지연을 줄일 수 있는 새로운 파이프라인 구조를 제안하고, 이 구조로 인한 성능 개선 정도를 분석하고자 한다. 특히 성능 분석의 측도로는 분기 지연 사이클만 측정하는 기존의 BEP(Branch Execution Penalty) 대신 CPU 처리 시간을 결정하는 CPI를 사용하겠다.

본 논문의 구성은 다음과 같다. 2장과 3장에서는 분기 예측 전략과 다중 경로 전략에 대한 기존 연구를 분석하고, 4장에서는 예측 전략과 결합된 새로운 이중 경로 전략과 이를 구현하기 위한 새로운 파이프라인 구조를 제안하고, 5장에서는 제안된 구조/전략을 기존의 전략들과 비교/평가한 후 6장의 결론으로 마무리한다.

2. 기존의 분기 예측 전략

분기의 방향을 미리 예측할 수 있다면(즉, 분기될지 아닐지를 미리 알 수 있다면) 분기로 인한 페널티를 줄일 수 있다. 이렇게 분기 방향을 예측하는 예측 전략은 분기 명령어를 번역 단계에서 분석하여 항상 일정하게 예측하는 정적 예측 전략과 과거 분기 명령어의 실제 수행 결과(이를 history라 한다)를 이용하여 현재 분기 명령어의 결과를 예측하는 동적 예측 전략으로 나눌 수 있다[5].

2.1 BTB를 사용하는 동적 예측 전략.

파이프라인에서 동적 예측 전략을 사용하기 위해서는 분기 명령어의 주소, 분기 예측 정보 및 목적 명령어에 대한 정보를 저장하는 장소가 필요하다. 일반적으로 이런 정보들을 저장하는 장치로 BTB(Branch Target Buffer)를 사용한다. BTB를 사용하는 전략에서는 다음의 두 가지 이유로 분기 지연이 발생한다.

(1) 분기 명령 채취 실패: 만일 채취된 분기 명령어가 BTB 내에 없으면 목적 명령어를 BTB에서 가져오지 못하고 캐쉬에서 가져오게 된다. 이런 경우에 발생하는 지연을 misfetch penalty라 한다.

(2) 분기 명령 예측 실패: 분기 방향을(taken될지 not taken될지) 잘못 예측하면, 이미 파이프라인에 들어와 있던 명령어들이 쓸모 없게 되어 새로운 명령어들을 채취해야 한다. 이런 경우에 발생하는 지연을 misprediction penalty라 한다.

일반적인 BTB 구조에서는 BTB에 분기 명령어가 없는 경우에는 예측 전략을 사용할 수 없고, 또 목적 주소를 저장하는 BTB에서는 예측이 올바른 경우에도 1 사이클의 분기 지연이 발생한다. 이런 지연은 목적 주소 대신 목적 명령어를 저장하면 해결되지만 BTB 크기가 커지는 단

점이 있다. 따라서, BTB를 사용하지 않고 예측 전략을 효과적으로 사용할 수 있으면 상당한 비용 절감 효과를 거둘 수 있다.

2.2 BTB를 사용하지 않는 예측 전략

BTB를 사용하는 목적을 Brad와 Dirk은 다음과 같이 3가지로 나누었는데[3], 다음 소절들에서 이들에 대한 해결책을 제시하여 BTB 없이 예측 전략을 사용하겠다.

(1) 분기의 결과를 예측할 수 있는 정보를 포함한다.
(2) 분기 명령어의 주소를 저장하여 파이프라인에 채취되는 명령어가 분기 명령어라는 것을 알 수 있게 한다.
(3) 분기 명령어의 분기 결과를 예측한 후 목적 명령어를 캐쉬에서 가져오게 되어 발생하는 분기 지연을 없애기 위하여 목적 명령어를 미리 가져온다.

2.2.1 분기 예측 테이블 구조

분기의 결과를 예측할 수 있는 정보를 BTB에 포함하지 않고 분기 예측 테이블(BPT)을 사용하면 BTB를 사용하는 목적 중에서 (1)은 해결된다. BPT는 분기 명령어들의 과거 수행 결과를 저장하고 이를 참조하여 현재 분기 명령어의 분기 방향을 동적으로 예측하는 장치이다. 최근에는 다음과 같은 BPT가 제안되고 있다.

① Yeh와 Patt은 예측의 정확도를 높이기 위하여 분기 명령의 History 뿐만 아니라 Pattern History를 저장한 2단계(two-level) BPT를 제안한다[8].

단계 1(Branch History Register) : 마지막으로 실행한 k개의 분기를 저장한다.

단계 2(Pattern History Table) : k개 분기의 마지막 j번 발생한 특정 분기 패턴을 저장한다.

이런 two-level BPT에는 다음과 같은 9가지 변형이 있다(GAg, GAs, GAp, PAg, PAs, PAp, SAg, SAs, SAp)[9].

② 조건 분기 명령어 중에서 명령어 채취 단계에서 바로 분기의 결과를 알 수 있는 명령어들(loop 명령어등)과 무조건 분기 명령어를 resolved 분기라 한다. 반면에 미리 분기의 결과를 알 수 없는 조건 분기 명령어, 간접 함수 호출/반환 등의 명령어를 unresolved 분기라 한다. 기존의 BPT는 모든 분기 명령어의 history를 갖고 있다. 그러나 resolved 분기는 분기의 결과를 명령어 채취 단계에서 바로 알 수 있으므로 BPT에서 예측할 필요가 없다. 따라서 BPT에서 모든 분기 명령어를 예측하지 않고 unresolved 분기만 예측하여 예측의 정확도를 높인다[10].

2.2.2 명령어의 유형을 미리 예측

명령어의 유형을 미리 알 수 있으면 BTB에 분기 명령어의 주소를 저장할 필요가 없다. 명령어의 유형을 미리 알기 위한 방법으로 Brad와 Dirk은 다음과 같은 2가지 방법을 제시하였다. 첫 번째는 명령어 집합의 구조를 다음과 같은 2개의 비트를 포함하도록 구성하는 것이다[3].

Opcode	B1	B2	Encode Disp.

B1	B2	명령어 유형
0	0	일반 명령어
0	1	반환 명령어
1	0	조건 분기 명령어
1	1	무조건 분기 명령어

(그림 1) 명령어 set의 구조 및 C, R의 내용
(Fig. 1) Instruction Set and Explicit Displacement Encoding

두 번째는 BTB와 비슷한 구조의 NLS(next cache line and set) 표를 사용하는 것이다[12]. NLS는 분기 명령어의 유형을 저장하는 Type Field와 목적 주소를 저장하는 Line Field로 구성되는데, BTB에서는 분기 명령어의 주소와 목적 주소를 함께 저장하는 반면에 NLS는 목적 주소의 하위 주소만을 저장하므로 비용 면에서 효과적이다.

2.2.3 캐쉬에서 목적 명령어를 직접 채취

온칩 캐쉬는 접근이 1 사이클 이내에 가능하

므로 BPT와 캐쉬에 동시에 접근하여 목적 명령어를 직접 캐쉬에서 가져오게 되면 BTB에 목적 주소를 저장할 필요가 없다[3].

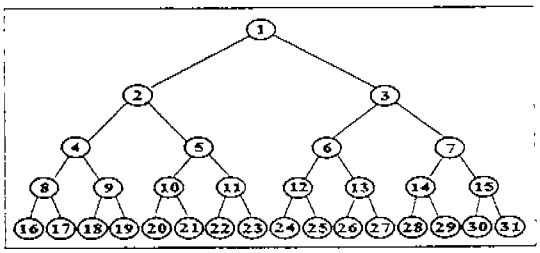
3. 다중 경로 전략의 구조와 문제점

앞 장에서 분석한 예측 전략으로는 misprediction시 발생하는 지연 사이클의 수는 줄일 수 없다. 이 사이클을 줄이는 한 방법으로 보조 파이프라인을 사용하는 다중 경로 전략이 사용된다.

3.1 이중 경로 전략

기존의 이중 경로 전략의 구조는 주 파이프라인과 보조 파이프라인으로 구성된다. 주 파이프라인을 IF, Decode, OF, Execution의 4 단계로 가정하면, 보조 파이프라인은 명령어가 분기 명령어라고 해석된 이후부터 Execution 이전까지만 분기 명령어의 taken(분기될 때의) 경로를 유지하므로 IF와 Decode의 2 단계로 충분하다. 주 파이프라인에서는 분기 명령어의 not taken (분기되지 않을 때의) 경로가 처리되고 보조 파이프라인에서는 주 파이프라인에서 분기 명령어가 해석된 다음 사이클에 taken 경로가 채취되어 처리된다.

그 결과 분기 명령어가 taken되는 경우에는 항상 1 사이클의 지연이 발생한다. 일반적으로 분기 명령어의 약 60% 정도가 taken되는 것[13]을 고려하면 상당한 문제가 된다. 또 다른 문제점은 분기 명령어가 연속적으로 채취될 때는 보조 파이프라인을 사용할 수 없다는 것이다. 단적으로 만일 모든 명령어가 분기 명령어이고 전부 taken된다고 하면 다음과 같은 분기 지연이 발생한다.



(그림 2) 모든 명령어가 분기 명령어인 경우의 이진 트리
(Fig. 2) All instructions are branches

(그림 3)에서 평균적인 지연 사이클은 2 사이클이다. 이런 문제를 해결하기 위하여 다중 경로 전략을 사용한다.

주 파이프라인	1	1	3	7	5	11	23	4	9	19	39
	2		1	3	7	5	11		4	9	19
	3			1	3	2	5			4	9
	4				1		2				4
보조 파이프라인	1			2	5		22			8	17
	2			2							8

(그림 3) 모든 명령어가 분기 명령어일 경우의 분기 지연
(Fig. 3) Branch Delay in case All instructions are branches

3.2 다중 경로 전략

다중 경로 전략은 이중 경로 전략에서 분기 명령어가 연속적으로 들어올 경우에 발생하는 문제를 해결하기 위하여 여러 개의 보조 파이프라인을 유지하는 전략이다. 만일 모든 명령어가 분기 명령어라면 주 및 보조 파이프라인에 들어와 있는 모든 명령어들에 대해 taken과 not taken 경로가 모두 필요하다. 파이프라인의 실행 단계를 n이라 하고 채취된 명령어가 해석되는 단계를 n'라 하면, 주 파이프라인에서 생성될 수 있는 분기 명령어들의 수는 최대 n-n'가 된다. 따라서, 보조 파이프라인의 단계 수는 n-n'가된다. 결국, 필요한 최대 보조 파이프라인의 수는 $2^{n-n'}$가된다.

다중 경로 전략을 사용한 경우에도 taken 분기 명령어가 채취되면 무조건 1 사이클의 지연이 발생한다. 만일 n-n'=6이라면 보조 파이프라인이 최대 32개가 필요하게되어, 이의 구현은 비용 및 구조/공간상 비 현실적이다.

4. 예측 전략과 결합된 이중 경로 전략

예측 전략의 문제점은 예측의 정확도는 높일 수 있으나 예측이 실패했을 때의 분기 지연은 줄일 수 없고 명령어 채취 단계에서 예측하기 때문에 misfetch penalty가 발생할 수 있다는 것이

다. 이중 경로 전략의 문제점은 taken 되는 분기 명령어가 채취되는 구조인 1 사이클의 지연이 발생되며, 분기 명령어가 연속적으로 들어오고 모두 taken될 경우 큰 분기 지연이 발생한다. 따라서 이런 문제점들을 해결할 수 있도록 예측 전략과 결합된 새로운 이중 경로 전략을 제시한다.

4.1 새로운 이중 경로 전략

기존의 이중 경로 전략은 주 파이프라인은 무조건 분기 명령어의 not taken 경로를 처리하고, 보조 파이프라인은 taken 경로를 처리한다. 그러나 본 연구에서 제안하는 이중 경로 전략은 주 파이프라인과 보조 파이프라인에서 모두 예측 전략을 사용한다. 이때 예측 전략은 unresolved 분기만 예측하는 2 단계 BPT(PAs)를 사용하고 [10, 11], 명령어 셋에 명령의 유형을 알 수 있는 2개의 비트를 포함하도록 하여 misfetch가 발생하지 않도록 한다.

주 및 보조파이프라인의 기능은 다음과 같다.

주 파이프라인 : 동적 예측 전략을 사용하여 명령어 채취 단계에서 분기 명령어의 결과를 예측하고 분기로 예측되면 taken 경로를 처리하고 그렇지 않으면 not taken 경로를 처리한다.

보조 파이프라인 : 주 파이프라인과는 반대 경로를 채취하여 처리하고, 동적 예측 전략을 사용한다.

주 파이프라인	1	1	3	7	15	18	37
	2		1	3	7	9	18
	3			1	3	4	9
	4				1	2	4
보조 파이프라인	1		2	4	9	19	36
	2			2	4		
	3				2		

(그림 4) 새로운 이중 경로 전략
(Fig. 4) New Multiple Path Strategy

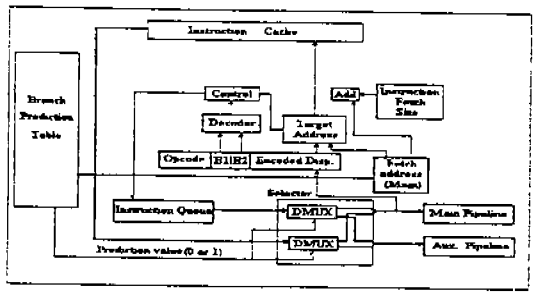
본 전략에서는 기존의 이중 경로 전략과는 달리 명령어 채취 단계(단계 1)에서 unresolved 분기 명령어라는 것이 알려지고, 분기 명령어가 채취되면 예측 전략을 사용하여 예측되는 경로를 각각 채취한다. 그 결과 taken 분기 명령어를 만나도 분기 지연이 발생하지 않고, taken 분기 명령어가 연속으로 채취되어도 예측이 정확하다면 다음 그림에서처럼 분기 지연이 발생하지 않는다. 또한, 단일 경로 예측 전략에서는 예측이 잘못되면 3 사이클의 분기 지연이 발생하였으나 이중 경로 전략에서는 예측이 잘못되어도 분기 지연이 발생하지 않는다.

4.2 프로세서 구조

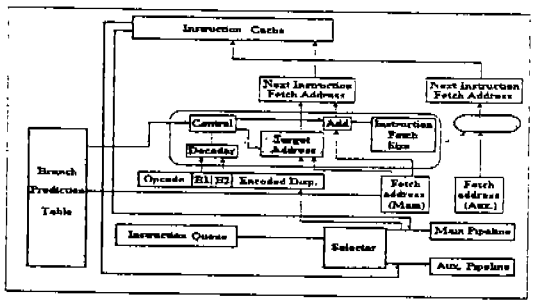
본 연구에서는 2.2절에서처럼 명령어 내에 명령어 유형을 나타내는 2 개의 비트를 갖는 프로세서 구조(그림 5)를 제안한다.

무조건 분기 명령어와 조건 분기 명령어들 중에서 루프 명령어는 분기의 결과를 실행 단계 이전에 알 수 있으므로, 이들을 제외한 unresolved 조건 분기 명령어에 대해서만 보조 파이프라인을 사용하고 예측 전략을 적용한다. 순차적인 다음 명령어는 명령어 큐에서, 목적 명령어는 캐쉬에서 파이프라인으로 보내진다.

명령어 큐에서 명령어가 주 파이프라인의 명령어 채취 단계로 보내질 때 동시에 명령어에 있는



(그림 5) 프로세서 구조(보조 파이프라인이 빈 경우)
(Fig. 5) A processor structure



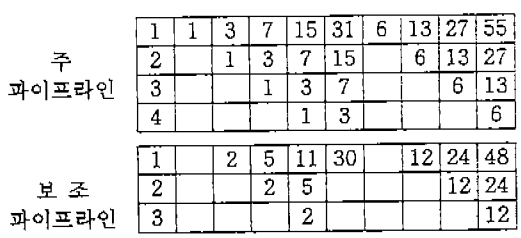
(그림 6) 예측 전략만 사용될 때의 운영 방법
(Fig. 6) Fulled Auxiliary Pipeline

명령어의 유형에 대한 정보를 포함하고 있는 두 개의 비트가 해석(decode)된다. Control은 해석된 결과를 참고하여 일반 명령이라면 명령어 큐로 신호를 보내고, 루프 명령어라면 명령어 큐와 연결되어 있는 루프 계수 레지스터(loop count register)를 사용하여 루프 명령어가 주 파이프라인으로 보내질 때마다 1씩 감소시키는 방법을 사용하여 0이 될 때까지 목적 주소에 신호를 전달해 taken될 때의 다음 명령의 주소를 캐쉬에 보낸다. 만일 unresolved 분기 명령어가 채취되면 Control은 목적 주소와 명령어 큐에 동시에 신호를 보낸다. 그 결과 taken 목적 명령어는 캐쉬에서, 순차적인 다음 명령어는 명령어 큐에서 선택기(Selector)로 전달된다. 선택기는 BPT로부터의 예측 값을 선택 선으로 하여 taken 목적 명령어와 순차적인 다음 명령어를 주 및 보조 파이프라인으로 보내고 보조 파이프라인이 채워져 있음을 Control과 BPT에 알려준다.

주 파이프라인과 보조 파이프라인에 동시에 채취된 명령어들이 unresolved 분기 명령어이면, 이들을 각각 BPT에서 예측하여 Control에서 Add와 목적 주소중 하나에 신호를 전달해 다음 명령의 주소를 캐쉬로 보내면, 캐쉬는 주 및 보조 파이프라인에 각각의 다음 명령어를 보낸다 (그림 6).

4.3 새로운 이중 경로 전략에서의 분기 지연

unresolved 분기 명령어가 채취되고, 2 사이클 이내에 다시 unresolved 분기 명령어가 채취되면 이 명령어에 대해서는 보조 파이프라인을 사용할 수 없으므로 예측의 정확도에 의지해야 한다. 다음 그림은 이런 특별한 경우를 보인 것이다.



(그림 7) 새로운 이중 경로 전략에서의 분기 지연
(Fig. 7) Branch Delay of New Two Paths

위의 그림에서 첫 번째 분기 명령어 1은 not taken으로 예측되므로 주 파이프라인은 명령어 큐에서 명령어 3을 채취하고, 보조 파이프라인은 명령어 1에 대한 taken 경로인 명령어 2를 캐쉬로부터 채취하였다. 명령어 1이 수행된 후 다음 사이클에서 주 파이프라인은 분기 명령어 15를 not taken으로 예측하여 주 파이프라인에서는 31을 보조 파이프라인에서는 30을 채취한다. 그러나 분기 명령어 3을 실행한 결과 예측이 잘못되었으므로 모든 명령어들이 버려지고 3에 대한 taken 목적 명령어인 6이 채취된다. 하지만, 명령어 3에 대한 taken 경로를 보조 파이프라인에 유지할 수 없었으므로, 예측 전략을 사용하지 않은 기존의 이중 경로 전략에서와 마찬가지로 3 사이클의 분기 지연이 발생한다.

결론적으로, unresolved 분기 명령어가 채취되고 분기의 결과가 알려지는 사이클 이내에 채취되는 unresolved 분기 명령어에 대한 예측이 실패할 경우에만 분기 지연이 발생하고 예로 든 파이프라인의 경우의 확률은 아래와 같다.

$$\text{분기 지연 발생 확률} = (\text{2 사이클 이내에 unresolved 분기 명령어가 존재할 확률}) \times (\text{예측이 실패할 확률}) \quad (4-1)$$

이때의 분기 지연 사이클은 unresolved 분기 명령어 두개(위의 그림에서 1, 3)를 처리할 때 3 사이클이 발생하므로 3/2=1.5 사이클이 된다.

5. 새로운 이중 경로 전략의 성능 평가

파이프라인에서 분기 처리 전략의 성능을 평가하기 위하여 여러 성능 측도가 사용되고 있다. 본 장에서는 먼저 기존의 측도들을 설명하고, 새로운 이중 경로 전략의 성능을 평가할 수 있도록 변형된 측도를 제안한다. 또한 이를 사용하여 기존의 여러 전략들과 성능을 비교 평가하여 본 연구에서 제안한 전략으로 파이프라인의 성능이 얼마나 개선될 수 있는가를 보이고, 캐쉬 히트 율이 성능에 미치는 영향도 분석한다.

5.1 성능 평가 측도

예측의 정확도(PA)는 예측 전략의 성능을 평가하는데 사용되는 측도이다. BEP는 PA와는 달리 분기로 인해 실제로 발생하는 지연 사이클의 평균이므로 예측 전략과 다른 분기 처리 전략을 비교하는데 사용될 수 있다.

$$BEP = P_{MfB} * \text{misfetch penalty} + P_{MpS} * \text{misprediction penalty} \quad (5-1)$$

P_{MfB} : ratio of misfetched branch

P_{MpB} : ratio of mispredicted branch

본 연구에서 제안하는 구조의 경우는 명령어의 유형을 완전히 알 수 있는 정보를 명령어 집합에 포함하므로 misfetch 페널티가 발생하지 않고, 4.3 절에서 설명된 특별한 경우에만 분기 지연이 발생한다. 그러므로 Brad와 Dirk이 제시한 BEP를 다음과 같이 개선하여 사용한다.

$$BEP = P_{CMpB} * \text{misprediction penalty} \quad (5-2)$$

P_{CMpB}=unresolved 분기 명령어가 채취되고 2 사이클 이내에 잘못 예측된 unresolved 분기 명령어를 만날 확률

파이프라인에서 여러 가지 분기 전략을 사용하는 이유는 명령어 하나의 평균 처리 시간을 1 사이클에 가깝게 유지하는 것이다. 그러므로 명령어에 대한 평균적인 처리 시간(CPI)를 구하여 다른 여러 분기 처리 전략들과 본 논문에서 제안하는 전략을 비교하는 것이 BEP를 비교하는 것보다 더 큰 의미를 가진다. 최적 상태에서 명령어를 처리하는데 기본적으로 1 사이클이 필요하므로 CPI는 각 전략에서 다음과 표시된다.

① 예측 전략에서의 CPI인 CPI(pd)는 다음과 같다.

$$CPI(pd) = 1 + P_b(P_{MfB} * \text{misfetch penalty} + P_{MpB} * \text{misprediction penalty}) \quad (5-3)$$

P_b=명령어가 분기 명령어일 확률

② 기존의 이중 경로 전략에서의 CPI인 CPI(tp)는 다음과 같다.

$$CPI(tp) = 1 + P_b * P_{bt}(1 + P_{CB}) \quad (5-4)$$

P_{bt}=분기 명령어가 taken될 확률

P_{CB}=분기 명령어가 연속적으로 채취될 확률

③ 새로운 이중 경로 전략에서의 CPI인 CPI(pd/tp)는 다음과 같다.

$$CPI(pd/tp) = 1 + P_b * P_{CMpB} * 1.5 \quad (5-5)$$

$P_{CMpB} = P_{2CB} * P_{MpB} * P_{UB}$

P_{2CB}=분기 명령어가 채취된 뒤 2 사이클 이내에 다시 분기 명령어가 채취될 확률.

P_{UB}=분기 명령어가 unresolved 일 확률.

5.2 기존의 분기 제안들과 새로운 제안의 비교/평가

다음 표는 SUN workstation에서 sed, cpp, gawk, espresso 프로그램들을 분석하여 분기 명령어들 간의 거리를 분석한 것이다[13].

〈표 1〉 분기 명령어들 간의 거리
〈Table 1〉 Branch Distance

	cpp	espresso	gawk	sed
Distance	%	%	%	%
0	0.36	0.01	0.23	3.06
1	11.34	22.46	16.51	22.96

〈표 1〉에서 거리가 0이라는 의미는 분기 명령어가 바로 연속하여 나타나는 것을 말한다. 거리 0, 1의 합은 2 사이클 이내에 분기 명령어가 나타날 확률이 된다.

2 사이클 이내에 새로운 분기 명령어가 나타날 확률은 약 19.37% 가 된다. 〈표 2〉는 SPEC89 벤치마크를 시뮬레이션한 것으로 전체 명령어중 모든 분기 명령어의 분율과 unresolved 분기만 예측한 것의 예측의 정확도를 나타낸 것이다[10].

〈표 2〉 SPEC89 벤치마크에서 예측의 정확도
〈Table 2〉 Prediction accuracy(SPEC89)

	% of Branches(P_b)	예측의 정확도
doduc	10.15%	95.58%
eqntott	31.37%	94.63%
espresso	28.10%	88.69%
fppp	2.40%	97.38%
spice	21.55%	98.72%
xlisp	21.28%	89.71%

〈표 1〉과 〈표 2〉에서 공통적인 프로그램인 espresso를 사용하여 새로운 이중 경로 전략과

기존의 예측 및 이중 경로 전략의 성능을 비교 평가하면

CPI(pd)=1.095이고, P_{bt}= 0.6인 경우이라 하면 〈표 1〉에서 P_{CB}=0.01이므로

CPI(tp)=1.170되고, P_{CMpB}=0.0247이므로

CPI(pd/tp) = 1.010이 된다.

위의 계산 결과를 정리하면 새로운 이중 경로 전략은 예측 전략에 대해서는 약 8.4%, 이중 경로 전략에 대해서는 약 15.8% 정도로 성능이 향상됨을 알 수 있다.

〈표 3〉은 좀더 객관 적인 평가를 위해 P_{2CB}를 0.1937, P_{CB}를 0.0092, P_{bt}를 0.6, P_{MiB}는 0.00895로 일률적으로 가정하고, 평가에 필요한 다른 요소는 SPEC89 벤치마크를 시뮬레이션한 결과[18]를 이용하여 새로운 이중 경로 전략과 unresolved 분기 명령어만 예측한 예측 전략 및 이중 경로 전략의 CPI를 비교한 표이다.

〈표 3〉 기존의 전략들과 본 전략의 CPI (SPEC89 벤치마크)
(Table 3) CPI of new proposal and other proposal

	P_b	P_{MpB}	P_{CMpB}	P_{UB}	CPI (pd)	CPI (tp)	CPI (pd/tp)
doduc	0.10	0.0472	0.0045	0.4891	1.015	1.061	1.0007
eqntott	0.31	0.0479	0.0060	0.6435	1.047	1.182	1.0028
espresso	0.28	0.0315	0.0060	0.9711	1.029	1.170	1.0025
fppp	0.02	0.0311	0.0022	0.3646	1.002	1.012	1.00007
spice	0.22	0.0170	0.0021	0.6342	1.013	1.133	1.0007
xlisp	0.21	0.1095	0.0202	0.9543	1.071	1.127	1.0064

〈표 3〉에서 보면 새로운 이중 경로 전략은 예측 전략에 대해서 최소 0.2%에서 최대 6.4%까지, 평균적으로 약 2.7%정도 성능이 향상된다. 이중 경로 전략에 대해서는 1.2%에서 17.9%까지, 평균적으로 약 11.2%정도 성능이 향상된다.

지금까지 분기로 인한 파이프라인의 성능 저하를 줄일 수 있는 새로운 이중 경로 전략을 제안하고 이로 인한 성능 향상에 초점을 맞추어 분석해 보았다. 전체 조건으로 분기 명령어를 포함한 모든 명령어가 항상 캐쉬에 있는 것으로 가정하였다. 그러나 실제 시스템에서는 캐쉬의 히트 율이 1이 될 수는 없으므로 캐쉬에서 히트될 때와 미스될 때를 함께 고려해야 한다. 미스될 때의 지연 사이클을 구하기 위해 먼저 메모리 시스템 전체의 평균 접근 시간인 T_{eff}(유효 접근 시간)을 구해 보면 다음 식으로 나타낼 수 있다.[1]

$$T_{eff}=\sum f_i \cdot t_i = h_1t_1+(1-h_1)h_2t_2+\cdots+(1-h_1)(1-h_2)\cdots(1-h_{n-1})t_n \quad (5-6)$$

이때 h_i는 i 번째 메모리(Mi)의 히트 율이고, f_i는 접근 빈도, t_i는 접근 시간을 의미한다.

계층적 메모리 구조를 갖는 파이프라인 프로세서에서(M1을 캐쉬, M2를 주 메모리라 하면)는 만일 캐쉬에서 미스가 발생하면 파이프라인은 주 메모리에서 이 명령어를 캐쉬로 가져올 때까지 일시 정지된 후 다시 실행된다. 따라서 CPI를 계산할 때 캐쉬 미스로 인한 평균 지연 사이클을 더해야 한다.

캐쉬 미스로 인한 평균 지연 시간은 메모리 시스템 전체의 평균 접근 시간인 T_{eff}와 h_1t_1의 차이고, 각 단계의 메모리 접근 시간을 사이클 시간(T)으로 나누어주면 각 단계에서 접근에 필요한 사이클 수를 구할 수 있다.

$$CPI(pd/tp)=1+P_b(P_{CMpB}*1.5)+(1-h_1)(\text{캐쉬 미스로 인한 지연 사이클}) \quad (5-7)$$

위의 식과 〈표 3〉을 사용하여 캐쉬와 주 메모리의 2 단계로 구성되어 있는 실제 시스템의 CPI를 구하기 위하여 먼저 다음과 같은 가정을 한다.

① t_1=사이클 시간(T)[1] ② t_2 는 t_1의 2~4 배 더 크다.[12]

$$CPI(pd/tp)=1+P_b(P_{CMpB}*1.5)+(1-h_1)t_2/t_1 \quad (5-8)$$

〈표 4〉는 캐쉬 히트 율을 고려하여 각 전략들의 CPI를 다시 구한 것이다. 이 CPI를 사용하여 본 전략으로 인한 성능 개선 정도를 각 전략들과 비교하여 분석해 보면 캐쉬 히트 율이 작을 수록, 캐쉬와 메인 메모리의 속도의 차이가 클 수록 상대적인 성능 개선 정도가 떨어짐을 알 수 있다. 이것은 CPI 전체 값에서 각 전략의 성능에 관계없이 공통적으로 모든 전략에 더해지는 메모리 시스템 의한 지연 사이클의 수가 증가하기 때문이다.

〈표 4〉 캐쉬 히트 율을 고려한 각 전략의 CPI
〈Table 4〉 CPI of each proposal with considered cache hit ratio

h1	0.96		0.94		0.90	
t2/t1	2	4	2	4	2	4
doduc(pd/tp)	1.08070	1.16070	1.12070	1.24070	1.20070	1.40070
(pd)	1.09500	1.17500	1.13500	1.25500	1.21500	1.41500
(tp)	1.14100	1.22100	1.18100	1.30100	1.26100	1.46100
eqntott(pd/tp)	1.08280	1.16280	1.12280	1.24280	1.20280	1.40280
(pd)	1.12700	1.20700	1.16700	1.28700	1.24700	1.44700
(tp)	1.26200	1.34200	1.30200	1.42200	1.38200	1.58200
espresso(pd/tp)	1.08250	1.16250	1.12250	1.24250	1.20250	1.40250
(pd)	1.10900	1.18900	1.14900	1.26900	1.22900	1.42900
(tp)	1.25000	1.33000	1.29000	1.41000	1.37000	1.57000
fppp(pd/tp)	1.08007	1.16007	1.12007	1.24007	1.20007	1.40007
(pd)	1.08200	1.16200	1.12200	1.24200	1.20200	1.40200
(tp)	1.09200	1.17200	1.13200	1.25200	1.21200	1.41200

결론적으로, 새로운 이중 경로 전략을 실제 사용되는 시스템에 적용했을 때와 근사한 성능 개선 정도를 분석한 결과 $t_2/t_1=4$일 때, 예측 전략에 대해서 평균적으로 약 1.81%정도 성능이 향상되고, 이중 경로 전략에 대해서는 약 10.45% 정도 성능이 향상됨을 알 수 있다.

6. 결 론

파이프라인 구조에서 제어적 문제를 개선하기 위한 예측 전략에서는 예측이 실패할 경우 분기 지연이 무조건 발생하고, 명령어 채취 단계에서 예측할 때 misfetch가 발생한다. 기존의 이중 경로 전략에서는 taken 분기 명령어가 채취되면 항상 분기 지연이 발생하고, 이런 taken 분기 명령어가 연속적으로 채취되는 경우에는 더 큰 분기 지연이 발생한다. 본 연구에서 제안한 동적 예측 전략과 이중 경로 전략을 결합한 파이프라인 구조에서는 주 파이프라인과 보조 파이프라인에서 각각 예측 전략을 사용하여 unresolved 분기 명령어가 채취된 후 2 사이클 이내에 잘못 예측된 unresolved 분기 명령어를 만나는 특별한 경우에만 분기 지연이 발생하므로 성능 개선 효과를 얻을 수 있다.

새로운 이중 경로 전략을 사용하는 파이프라인 구조의 단점은 구조가 복잡하고 비용이 많이 든다는 점이다. 이런 단점을 해결하기 위하여 본 연구에서는 BTB를 사용하지 않고 unresolved 분기 명령어만 BPT에서 예측하고, 2 단계 BPT 중에서 비용 면에서 가장 효과적인 PAs를 채택하여 구조를 단순화하고 비용을 절감한다.

결론적으로 예측 전략에 대해서는 CPI가 평균적으로 약 2.7%, 이중 경로 전략에 대해서는 약 11.2%의 성능 개선 효과가 있음을 보였다. 또한, 새로운 이중 경로 전략을 실제 사용되는 시스템에 적용했을 때와 근사한 성능 개선 정도를 분석하기 위하여 캐쉬 미스 율을 고려한 각 전략들의 CPI를 다시 분석한 결과 본 논문에서 제안한 전략은 예측 전략에 비해 평균적으로 약 1.81 % 성능이 향상되고, 이중 경로 전략에 비해서는 약 10.45% 성능이 향상되었다.

참 고 문 헌

[1] Kai Hwang, 'Advanced Computer Architecture: Parallelism, Scalability Programmability', McGraw-Hill, pp. 14-17, 1993.

[2] Bob Ryan, "M1 Challenges Pentium," BYTE, pp. 83-87, Jan. 1994.

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

[4] Wen-mei W. Hwu et al, "Comparing Software and Hardware Schemes For Reducing the Cost of Branches," Proc. 16th Ann. Symp. Computer Architecture, pp. 224-232, 1989.

[5] Harvey G. Cragon, 'Branch Strategy Taxonomy and Performance Models', IEEE Computer Society Press, 1991.

[6] J.F.K. Lee and Smith, "Branch Prediction Strategies and Branch Target Buffer Design," IEEE Computer, pp. 6-22, Jan. 1984.

[7] David J. Lilja, "Reducing the Branch Penalty in Pipelined Processors," IEEE Trans. Computers, pp. 47-55, July

1988.

[8] Tse-Yu Yeh and Yale N. Patt, "Alternative Implementations of Two- Level Adaptive Branch Prediction," Proc. 19th Ann. Symp. Computer Architecture, pp. 124-134, 1992.

[9] Tse-Yu Yeh and Yale 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.

[10] Adam R. Talcott et al, "The Impact of unresolved Branch on Branch Prediction Scheme Performance," Proc. 21th Ann. Symp. Computer Architecture, pp. 12-21, 1994.

[11] Brad Calder and Dirk Grunward, "Next Cache Line and Set Prediction," Proc. 22th Ann. Symp. Computer Architecture, pp. 287-296, 1995.

[12] Mcfarling, S. and Hennessy, J., "Reducing the Cost of Branches," Proc. 13th Ann. Symp. Computer Architecture, pp. 396-403, 1986.

[13] Ishaq H. Unwala and Harvey G. Cragon, "A Study of MIPS Programs," Computer Architecture, Vol. 22, No. 5, pp. 30-40, Dec. 1994.

주 영 상

1989년 고려대학교 재료공학과 졸업(학사)
1995년 단국대학교 대학원 전산통계학과 대학원 졸업(이학석사)
1995년~현재 단국대학교 대학원 전산통계학과 박사과정
관심분야 : 컴퓨터 구조, 성능 분석, 병렬 구조

조 경 산

1979년 서울대학교 전자공학과 졸업(학사)
1981년 한국과학원 전기 및 전자공학과 졸업(공학석사)
1988년 텍사스대학교(오스틴 소재)전기 전산 공학과 졸업(Ph. D.)
1988년~90년 삼성전자 컴퓨터 부문 책임연구원
1990년~현재 단국대학교 전산통계학과 부교수
관심분야 : 컴퓨터 구조, 성능 분석, 병렬 구조, 시뮬레이션