통계적 모의실험을 이용하는 프로세서의 성능 모델

(The Processor Performance Model Using Statistical Simulation)

이 종 복 †

(Jongbok Lee)

요 약 마이크로 프로세서 구조의 성능을 분석할 때, 트레이스 구동형 모의실험이 광범위하게 수행되고 있으나, 시간과 공간을 많이 차지하기 때문에 최근에 이르러 통계적 모의실험이 그 대안으로 떠오르고 있다. 기존의 통계적 모의실험이 단일 분기 예측법에 대하여 연구가 수행된 것과 달리, 본 논문에서는 다중 분기 예측법을 이용하는 고성능 수퍼스칼라 프로세서에 대한 통계적 프로화일링 모델을 제안하였다. 이때, 다중 분기 예측법은 최근 들어 유망한 기법으로 대두되고 있는 퍼셉트론 분기 예측법을 기반으로 하였다. 이것을 위하여 SPEC 2000 벤치마크 프로그램의 특성을 통계적 프로화일링 기법으로 모델링하고, 여기서 얻은 통계적 프로화일을 바탕으로 벤치마크 트레이스를 합성하여 모의실험을 수행하였다. 그 결과, 제안하는 방식으로 다중 분기 예측을 이용하는 수퍼스칼라 프로세서에서도 비교적 높은 정확도를 얻을 수 있었다.

키워드 : 통계적 모의실험, 수퍼스칼라 프로세서, 다중 분기 예측법

Abstract Trace-driven simulation is widely used for measuring the performance of a microprocessor in its initial design phase. However, since it requires much time and disk space, the statistical simulation has been studied as an alternative method. In this paper, statistical simulations are performed for a high performance superscalar microprocessor with a perceptron-based multiple branch predictor. For the verification, various hardware configurations are simulated using SPEC2000 benchmarks programs as input. As a result, we show that the statistical simulation is quite accurate and time saving for the evaluation of microprocessor architectures with multiple branch prediction.

Key words : statistical simulation, superscalar processor, multiple branch prediction

1. 서 론

컴퓨터 구조의 개발 단계에서 성능을 평가하기 위하여 모의실험이 광범위하게 행하여지며, 이 때 트레이스 구동 모의실험(trace-driven simulation)이 주로 이용된다. 트레이스 구동 모의실험은 비교적 정확하다는 장점이 있으나, 공간과 시간이 매우 많이 소요되며, 하드웨어 사양이 바뀔 때마다 모의실험을 다시 반복해야하는 단점이 있다. 이러한 난관을 타개하기 위한 여러가지 방법이 연구되어왔다.

최근에 들어서 주목을 끌고 있는 방법은 벤치마크에서 추출한 명령어 트레이스를 입력으로 하여 모의실험하는 기존의 방법과는 달리, 프로세서와 프로그램의 통계적 특성을 수집하고 그것을 바탕으로 새로운 입력 트레이스를 합성하여, 이것을 확률적으로 모의실험하는 통계적 프로화일링(statistical profiling) 방법이다[1,2]. 통계적 프로화일링이란 프로그램의 중요한 특성에 대한 분포를 기록한 집합이며, 프로그램의 특성과 더불어 프로세서에 대한 특성을 통계적 방법에 의하여 얻는다. 통계적 프로화일을 얻은 후에, 이 자료를 바탕으로 통계적 명령어 트레이스를 새로 합성한다. 합성된 명령어 트레이스는 통계적 프로화일링 기법으로 임의로 발생하였기 때문에 각 벤치마크 프로그램의 특성을 함축적으로 잘 대표한다. 따라서 합성된 새로운 명령어 트레이스는 간단한 트레이스 구동형 모의 실험기에서 실행 가능하며, 기존의 일반적인 트레이스 구동 모의실험 보다 짧은 시

· 이 논문은 한국과학재단의 해외 Post-Doc 연수지원에 의하여 연구되었음

† 정 회 원 : 한성대학교 정보통신공학과 교수
jblee@hansung.ac.kr

논문접수 : 2005년 8월 23일
심사완료 : 2006년 1월 19일

간에 성능에 대한 예측을 비교적 정확히 할 수 있으므로, 마이크로 프로세서의 초기 설계 과정에서 성능 평가를 시행할 때 유용하게 이용할 수 있다.

한편, 고성능 수퍼스칼라 마이크로 프로세서의 명령어 대역폭을 증가시키는 방편으로 다중 분기 예측법[3,4] 또는 트레이스 캐쉬[5]와 같은 방법이 널리 연구되고 있다. 그러나 통계적 프로화일링에 의한 모의실험은 단일 분기 예측에 대하여만 연구가 되었으며, 다중 분기 예측에 대한 결과는 아직 보고되지 않았다. 본 논문에서는 퍼셉트론[6,7]을 기반으로 하여 다중 분기 예측법을 적용하는 통계적 프로화일링 기법을 제안하였다. 이것을 위하여, SPEC 2000 벤치마크의 정수형 프로그램 10 개와 실수형 프로그램 8 개를 대상으로 다중 분기 예측법을 이용하는 수퍼스칼라 마이크로 프로세서의 성능을 통계적 프로화일링 기법을 이용하여 측정하였다. 그리고 이것을 기존의 일반적인 트레이스 구동 모의실험으로 측정한 결과와 비교하여 그 정확도를 평가하였다.

본 논문은 다음과 같이 구성된다. 2장에서는 통계적 프로화일링 기법에 대하여 논하고, 3장에서는 퍼셉트론을 이용한 다중 분기 예측법에 대하여 고찰한다. 4장에서는 모의실험 환경을 다룬다. 5장에서 모의실험 결과를 보이고, 6장에서 결론을 맺는다.

2. 통계적 프로화일링 기법

통계적 모의실험의 전 과정을 자세히 살펴보면 그림 1과 같이 크게 4 단계로 나누어지는데, 첫째, 일반적인 프로그램 트레이스의 발생, 둘째 통계적 프로화일링 기법에 의한 분석 및 통계 데이타의 수집, 셋째, 통계적 트레이스의 합성, 넷째 합성된 트레이스에 대한 통계적 트레이스 구동 모의실험이다. 첫번째의 프로그램 트레이스 발생은 기존의 방법과 동일하다. 특정한 벤치마크 프로그램을 구체적인 명령어 트레이스 발생기를 통하여 명령어 트레이스를 생성한다.

두번째의 통계적 프로화일링을 위한 분석 및 통계 데이타의 수집 단계는 다음과 같다. 우선, 첫 단계에서 얻은 프로그램의 명령어 트레이스를 분석하여 프로그램의 고유한 특성과 지역적 특성에 대한 통계값들을 추출해 낸다. 프로그램의 고유한 특성은 프로그램을 구성하는 명령어의 유형별 구성비와 레지스터 피연산자의 개수 및 명령어 간의 데이타 종속에 대한 분포로 구성된다. 이 때, 명령어의 유형별 분포를 구하기 위하여 원래의 아키텍쳐가 갖는 명령어 집합에서 총 7 개의 간단한 명령어 집합으로 축소시킨다. 한편, 명령어 간의 상호 레지스터 종속은, 유형별 명령어의 소오스 레지스터와 그 레지스터를 목적 레지스터로 하여 종속을 부여하는 선행하는 명령어 간의 거리로 정한다. 이러한 특성은 주어

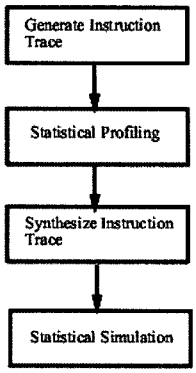

그림 1 통계적 모의실험의 흐름도

진 마이크로 프로세서의 구조와는 무관하고 단지 컴파일러와 마이크로 프로세서의 명령어 집합에만 영향을 받는 고유한 성질이다. 지역적 특성은 분기 미스율이나 캐쉬 미스율 등을 의미하며, 이것은 마이크로 프로세서 하드웨어 구조에 의하여 영향을 받는다. 이 단계는 각 벤치마크 프로그램에 대하여 단 한 번만 시행한다.

세번째, 위에서 얻은 통계적 특성을 기반으로 하여 난수를 발생시켜서 통계적 특성을 갖는 새로운 명령어 트레이스를 합성한다. 통계적 트레이스를 발생시키는 방법은 0부터 1 사이의 난수를 발생시키고 이 값을 통계적 프로화일링에 의하여 얻은 누적 분포 함수에 대응시키는 것이다. 위와같이 명령어의 유형 뿐만이 아니라, 피연산자의 개수, 피연산자의 종속거리를 통계적 프로화일링을 기반으로 결정하여 명령어 코드를 만들어낼 수 있기 때문에 이와같은 작업을 반복하면 프로그램의 특성을 함축적으로 내포하는 통계적 트레이스의 합성이 가능하다.

마지막으로, 이렇게 합성된 트레이스를 분기 히트율 및 캐쉬 히트율에 대한 정보와 함께 수퍼스칼라 프로세서 모의실험기로 입력하여 그 성능을 측정한다. 한편, 통계적 프로화일링에 의하여 자료를 확보한 후에는, 하드웨어 조건을 바꿔서 전체 설계 공간에 대하여 다양한 시도를 할 수가 있다. 즉, 윈도우의 크기, 명령어의 인출율, 명령어의 실행 지연 싸이클, 파이프라인 단계의 수를 변화시켜가면서 새로운 결과를 간편하게 얻을 수가 있으므로 매우 유용하다.

3. 퍼셉트론을 이용한 다중 분기 예측법

3.1 퍼셉트론 분기 예측법

퍼셉트론 분기 예측법은 인공지능 분야의 신경망 회로에서 이용하는 학습 방식을 분기 예측 기법에 응용한 것이다[6,7]. 이것을 위하여 프로그램의 수행 결과 분기한 것을 1로, 분기하지 않는 것을 -1로 구분하여 분기 히스토리 레지스터에 기록하고, 현재 분기 결과와 분기

히스토리 레지스터의 결과들을 곱하고 더하여 가중치 벡터를 생성한다. 따라서, 분기를 많이 할수록 가중치는 값이 커지고, 분기를 적게 할수록 가중치는 값이 작아지는 방향으로 퍼셉트론을 학습시킨다. 다음의 분기명령어에 대한 예측을 수행할 때, 분기 히스토리 레지스터에 수록된 정보와 가중치 벡터를 곱하고 합하여 그 값이 양수이면 분기하는 것으로, 음수면 분기하지 않는 것으로 결정한다.

3.2 다중 분기 예측법

기존의 2 단계 적응형 분기 예측법은 분기 히스토리 레지스터와 패턴 히스토리 테이블을 이용한다[8]. 다중 분기 예측법은 이러한 2 단계 적응형 분기 예측법을 응용 및 확장한 것이다. 이 때, M 개의 다중 분기 예측을 수행하기 위하여, 첫번째, 두번째, ..., M 번째 분기 명령어에 대하여 각각 히스토리 레지스터의 k 비트, k-1 비트, ..., k-M+1 비트로 패턴 히스토리 테이블을 인덱스하여 각 분기 명령어를 예측한다.

퍼셉트론 분기 예측 역시 위에서 살펴본 바와 같이 일정한 길이의 분기 결과를 히스토리 레지스터에 기록하고 임의의 분기 어드레스를 가지고 가중치 벡터로 구성된 테이블을 접근하여 2 단계로 예측을 수행하므로, 전역 히스토리 방식에 의한 2 단계 다중 분기 예측법을 동일하게 적용할 수 있다.

3.3 다중 분기 예측법의 통계적 모의실험에의 적용

단일 분기 예측법을 이용하는 기존의 통계적 모의실험에서는, 단일 분기 예측 정확도의 통계적 분포를 이용하여 각 분기 명령어가 올바르게 예측되는가의 여부를 결정한다. 그러나 본 논문에서 제안하는 다중 분기 예측법의 통계적 모의실험에의 적용에서는, 통계적 프로화일링 단계에서 연속적인 블럭에 대한 분기 예측 정확도를 구한다. 매 싸이클 당 2 개의 분기 명령어를 예측할 때는, 연속적인 블럭 2 개의 예측 정확도를 얻는다. 이 때 분기 예측 정확도는 연속적으로 올바르게 예측되는 블록에 따라 각각 2-블럭과 1-블럭이라 명명한다. 또한, 매 싸이클당 3 개의 분기 명령어를 예측할 때는 각각 3-블럭, 2-블럭과 1-블럭으로 정의하였다.

위와같이 다중 블럭에 대하여 분기 예측에 대한 통계를 얻은 후에 그 결과를 통계적 모의실험에 이용한다. 매 싸이클마다, 통계적 분포에 따라 다중 블럭의 종류를 결정함으로써, 프로그램의 흐름에서 만나는 연속적인 분기 명령어에 대한 결과를 결정한다. 예를 들어, 3 차의 다중 분기 예측법에서 난수 발생에 의하여 3-블럭이 결정되었다면, 그 싸이클에서 첫번째, 두번째 및 세번째 블럭에 대한 예측이 모두 옳게 예측되었다고 가정한다. 그러나, 2-블럭이나 1-블럭이 결정되었다면, 각각 첫번째와 두번째 블럭 및 첫번째 블럭에 대한 예측만이 옳다고 가정하는 방법이다.

4. 모의실험 환경

4.1 수퍼스칼라 프로세서

본 논문에서는 명령어 윈도우에서 동적 스케줄링을 관리하는 수퍼스칼라의 기본형에 다중 분기 예측기를 결합한 구조를 이용하였으며, 명령어 코드 발생을 위하여 Supersparc 명령어 세트를 이용하였다[9]. 표 1에 본 연구에서 사용된 각 명령어의 이슈 지연(issue latency) 및 결과 지연(result latency) 싸이클 수를 나타내었다.

표 1 명령어의 지연 특성

연산유닛	단일/2배 정밀도	이슈 지연	결과 지연
산술연산기	단일	1	1
분기	단일	1	1
로드	단일/2배	1	1
스토어	단일/2배	1	1
실수형 덧셈	단일/2배	1	1
실수형 곱셈	단일	1	3
	2배	1	3
실수형 나눗셈	단일	4	6
	2배	7	9

다중 분기 예측의 경우 퍼셉트론 방식의 2 단계 적응형 분기 예측법을 이용하여 동시에 2 개 또는 3 개의 기본 블록을 인출할 수 있다. 퍼셉트론을 이용한 분기 예측 알고리즘에서 분기 히스토리 레지스터의 길이는 8로 하였고, 가중치 벡터로 구성되는 테이블의 항목 수는 4096개로 하였다.

일반적인 정수형 프로그램의 기본블럭의 크기가 5이기 때문에, 최대 3 개의 기본블럭을 감안하여 한 싸이클에 인출할 수 있는 명령어의 최대수를 16으로 정하였다. 기준 마이크로 프로세서의 윈도우의 크기는 64이고, 리오더 버퍼를 운용하여 레지스터 재명명(register renaming)과 인터럽트 처리에 대비하였다.

1 차 명령어 캐쉬(L1-instruction cache) 및 1 차 데이타 캐쉬(L1-data cache) 및 분기 예측을 위한 분기 주소 캐쉬(branch address cache)도 구체적인 사양에 의하여 동작한다. 본 모의실험에서 각종 2 차 캐쉬(L2-cache)는 1 차 캐쉬(L1-cache)와 달리 100% 히트가 난다고 가정하였다. 본 논문에서 사용하는 연산유닛, 캐쉬의 기본 사양과 미스 페날티 싸이클 수에 대하여 표 2에 일목요연하게 나타내었다. 미스 페날티 싸이클은 실제 트레이스 구동 모의실험 및 통계적 모의실험을 할 때도 동일하게 적용된다.

표 2 아키텍쳐 사양

유형	값
윈도우 크기	16/32/64
인출율	4/8/16
이슈율	4/8/16
퇴거율	4/8/16
연산유닛	4/8/16 정수형 산술논리연산 유닛 2/4/8 로드/스토어 유닛 1/2/4 실수형 덧셈기 1/2/4 실수형 곱셈기
1차 명령어 캐쉬	128 KB, 2 차 세트 연관, 16 B 블록, 미스 지연 10 싸이클
1차 데이타 캐쉬	128 KB, 직접 매핑, 32 B 블록, 미스 지연 10 싸이클
분기 주소 캐쉬	2 K 항목
분기 예측기	8 비트 히스토리, 4096 퍼셉트론 테이블, 예측 미스 지연 6 싸이클

4.2 벤치마크 프로그램

입력 벤치마크 프로그램으로는 10 개의 SPEC 2000 정수형 프로그램과 8 개의 SPEC 2000 실수형 프로그램이 사용되었다. 이 프로그램을 SunOS 5.6 운영체제 하의 Sun Sparc Ultra-2 머신에서 C 컴파일러를 이용하여 실행 가능 화일을 얻었다. 이 실행 가능 화일과 Shade를 이용하여 Sparc V9 명령어 트레이스를 생성하였다[10]. 초기의 일반적인 트레이스 구동 모의실험에서 각 프로그램마다 1 천만 개의 명령어를 발생시켜 모의실험에 이용하였다.

4.3 모의실험기

본 논문에서 제안하는 통계적 모의실험을 수행하기 위하여 통계적 모의실험기가 필요하며, 기존의 일반적인 트레이스 구동 모의실험과의 성능을 비교하기 위하여 일반적인 트레이스 구동 모의실험기 역시 필요하다. 일반적인 트레이스 구동 모의실험기는 Sparc 명령어를 트레이스로 받아들이면서 동시에 퍼셉트론을 이용한 단일 및 다중 분기 예측법을 수행해야 한다. 또한, 통계적 모의실험기는 축약된 명령어 집합으로 합성된 통계적 명령어 트레이스를 대상으로 확률적으로 캐쉬 및 분기 예측 정확도를 발생시켜서 트레이스 구동 방식으로 실행되어야 한다. 따라서 본 연구에서는 두 가지 모의실험기를 리눅스 환경에서 C를 이용하여 개발하였다.

5. 모의실험 결과

5.1 기본 특성

표 3은 10개의 정수형 및 8개의 실수형 SPEC 2000 벤치마크 프로그램에 대한 기본 특성을 나타낸 것이다. 이 때 명령어 캐쉬와 데이타 캐쉬는 64 KB의 용량을 가지며, 분기 주소 캐쉬는 2048 개의 항목을 가진 것을 기준으로 하였다. 충분한 용량의 명령어 캐쉬로 인하여, 명령어 캐쉬는 94.7% 이상의 히트율을 보이며, 데이타 캐쉬 히트율 역시, bzip2를 제외하고는 96.7% 이상을 기록하였다. 분기 예측 정확도는 단일 분기 예측을 시행하였을 때의 결과이며, 분기 주소 캐쉬 히트율은, 불규칙한 분기 형태로 인하여 88.9%를 나타내는 gcc를 제외하고 97.3% 이상을 나타내었다.

표 3 벤치마크 프로그램의 기본 특성

벤치마크	기본 블럭	분기 예측 정확도 (%)	명령어 캐쉬 히트율 (%)	데이타 캐쉬 히트율 (%)	분기주소 캐쉬 히트율 (%)
bzip2	11.77	97.91	99.96	71.62	99.90
crafty	5.30	89.43	99.91	99.85	99.54
gap	11.56	94.64	99.78	96.74	99.04
gcc	6.05	91.10	94.73	99.65	88.89
mcf	7.36	95.90	99.99	99.45	99.98
parser	7.21	93.57	99.84	99.76	99.19
perlbmk	8.04	95.36	99.36	99.25	97.75
twolf	4.37	93.61	98.56	99.60	99.26
vortex	6.00	90.63	96.22	98.58	92.37
vpr	4.49	94.25	97.46	99.76	97.27
ammp	5.64	94.06	98.95	99.69	97.21
apsi	8.22	95.54	99.86	99.15	99.36
art	6.72	84.83	99.90	99.07	99.52
equake	4.54	82.04	99.91	98.98	99.58
mesa	5.95	95.41	99.84	99.04	95.89
mgrid	6.16	96.04	99.85	98.25	94.42
swim	6.90	84.70	99.85	95.34	97.53
wupwise	13.76	90.60	99.83	99.69	99.29

5.2 명령어의 유형별 분포율

표 4에 정수형 및 실수형 벤치마크 프로그램에 대하여 통계적 프로화일링 기법에 의하여 측정한 7 가지 명령어 유형에 대한 분포 비율을 나타내었다. Bzip2는 정수형 프로그램이지만 예외적으로 24%의 실수형 덧셈 명령어로 구성된다. 실수형 벤치마크 프로그램은 정수형 프로그램과는 달리 실수형 덧셈 외에 실수형 곱셈과 실수형 나눗셈 명령어가 포함되었다. 단, 1% 미만의 점유율을 나타내는 경우는 <1과 같이 표현하였다.

5.3 명령어 유형별 피연산자 수의 분포율

표 5에서는, 정수형 벤치마크 gap과 실수형 벤치마크 wupwise에 대하여 유형별 명령어에 대한 피연산자 개수의 백분율을 나타내었다. 분기 명령어는 즉시 모드(immediate mode)로 이용되어 레지스터 피연산자가 없으므로 나타내지 않았으며, 실수형 명령어의 경우, 실수형 덧셈 명령어 외에 실수형 곱셈 및 실수형 나눗셈 명령어는 피연산자가 2 개인 형태 밖에 없으므로 표시하지 않았다. Gap에서 정수형 명령어의 경우 Sparc V9의

표 4 벤치마크 프로그램의 명령어 유형별 분포 비율(%)

벤치마크	정수형	로드	스토어	분기	실수형 덧셈	실수형 곱셈	실수형 나눗셈
bzip2	25	31	12	9	24	0	0
crafty	40	28	13	19	0	0	0
gap	63	19	10	9	<1	0	0
gcc	43	29	11	17	<1	0	0
mcf	35	34	17	14	0	0	0
parser	50	25	11	14	0	0	0
perlbmk	39	34	14	12	<1	<1	<1
twolf	48	17	11	23	<1	0	0
vortex	48	21	11	17	3	0	0
vpr	50	22	5	22	<1	<1	<1
ammp	59	18	5	18	<1	<1	<1
apsi	47	24	18	12	<1	<1	<1
art	56	21	8	15	<1	<1	<1
equake	57	16	5	22	<1	<1	<1
mesa	36	28	19	17	<1	<1	<1
mgrid	57	21	6	16	<1	<1	<1
swim	44	22	9	15	6	5	<1
wupwise	54	25	9	7	3	1	<1

표 5 유형별 명령어의 종속거리 분포율의 예(%)

명령어 유형	피연산자 개수	gap	wupwise
정수형	0	28.6	4.3
	1	58.7	67.2
	2	12.7	28.5
로드	1	54.6	78.7
	2	45.4	21.3
스토어	1	79.1	75.8
	2	20.9	24.2
실수형 덧셈	1	100.0	44.4
	2	0.0	55.6

특성에 따라 피연산자 개수가 0인 경우도 존재하며, 피연산자 개수가 1 개인 경우가 2 개인 경우보다 4.6 배 많다는 것을 알 수 있다. 10 개의 정수형 벤치마크 및 8 개의 실수형 벤치마크에 대해서 이와 같은 유형별 명령어의 피연산자 개수의 분포율을 모두 구하여 통계적 트레이스를 합성하는데 이용하였다.

5.4 명령어 유형별 종속거리의 분포율

그림 2에 정수형 프로그램 gap에 대하여 프로화일링 기법으로 측정한 명령어 간의 종속 거리(dynamic instruction distance)를 명령어 유형 및 피연산자 개수별로 거리가 1부터 5인 것까지 그 분포를 백분율로 나타내었다. 이 그림에서 알 수 있듯이 대부분의 명령어는 거리가 1이나 2인 명령어에 종속임을 알 수 있으며, 명령어 간의 종속 거리가 8을 넘으면 그 확률 분포가 급격하게 줄어든다. 그러나 정밀한 통계적 프로화일링을 위하여 윈도우 크기만큼 떨어진 거리에 놓여있는 소수

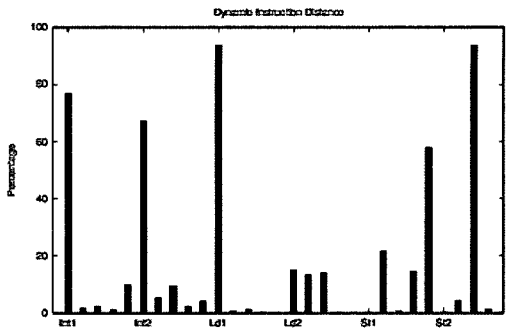

그림 2 유형별 명령어 간의 종속거리 분포율(%)

점 미만의 종속거리 분포율도 포함되었다.

5.5 다중 분기 예측 정확도

표 6에 정수형 및 실수형 벤치마크에 대하여 측정한 퍼셉트론 방식의 다중 분기 예측도를 나타내었다. 앞에서 설명한 대로, 2 개 및 3 개의 연속적인 블럭에 대한 예측도를 표시하였다. 2 차의 분기 예측에서, 대부분의 예측도는 2-블럭에 속하며, 1-블럭의 예측도는 crafty에서 최대 10.2%를 넘지 않았다. 3 차의 분기 예측에서도 마찬가지로 대부분이 3-블럭에 속하며, 2-블럭과 1-블럭은 perlbmk에서 10.6%를 초과하지 않았다. 실수형 벤치마크도 모든 블럭에 대한 예측이 옳은 경우가 대부분이었다. 그러나 2 차의 다중 분기 예측에서 art, equake, swim에서 1-블럭이 10.4% 이상을 기록하였으며, 3 차의 다중 분기 예측에서 swim의 2-블럭이 26.8%의 높은 값을 나타내었다.

표 6 다중 분기 예측 정확도(%)

벤치마크	2 차		3 차		
	2-블럭	1-블럭	3-블럭	2-블럭	1-블럭
bzip2	94.80	2.21	72.17	8.06	2.02
crafty	79.27	10.20	67.88	7.89	0.30
gap	82.87	7.52	65.59	9.57	2.51
gcc	82.77	7.30	73.34	7.09	1.25
mcf	92.67	3.58	88.70	4.76	0.07
perlbmk	87.33	5.61	60.81	10.63	2.21
twolf	88.08	5.79	83.82	4.45	0.91
vortex	80.87	8.24	70.81	8.45	1.88
vpr	89.16	5.05	82.70	6.24	0.52
ammp	88.52	3.85	82.41	5.47	0.64
apsi	92.33	3.38	64.19	10.37	2.66
art	68.54	11.00	48.20	12.93	0.37
equake	63.76	12.56	42.61	10.16	0.29
mesa	91.50	5.88	87.95	5.47	0.13
mgrid	92.95	3.15	83.56	4.57	1.15
swim	70.44	10.40	51.25	26.76	0.90
wupwise	82.75	7.39	70.20	8.41	2.24

5.6 소규모 프로세서 사양에 대한 성능 비교

그림 3은 윈도우 크기가 16일 때 정수형 벤치마크의 모의실험 결과를 보인 것이다. 이 때 구체적인 아키텍처 사양은 표 2에 나타낸 것과 같다. 그림의 각 벤치마크에 대하여, 3 개의 점은 각각 단일 분기, 2 개 분기, 3 개 분기 예측에 대한 결과를 나타내며, 비교가 용이하도록 일반적인 트레이스 구동 모의실험에 의한 결과와 통계적 모의실험에 의한 결과를 함께 도시하였다.

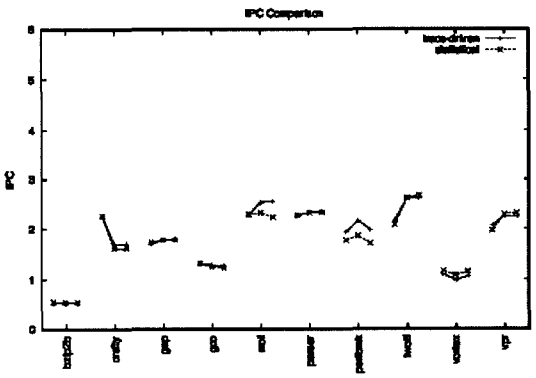

그림 3 정수형 벤치마크 성능의 비교(w = 16)

단일 분기 예측의 경우, mcf가 분기 예측 오류나 데이타 캐쉬 미스에 의한 성능의 손상을 입지 않아서 가장 높은 IPC를 나타내었다. 이와는 대조적으로 bzip2는 데이타 캐쉬 히트율이 64.9%로 극히 저조하여 가장 낮은 IPC를 기록하였다. 단일 분기 예측의 경우 일반적인 트레이스 구동 모의실험과 통계적 모의실험에 대한 성능의 조화평균은 모두 1.6 IPC를 나타내었으며, 상대오차는 3.3%를 기록하였다. 매 싸이클마다 2 개의 분기어를 예측하였을 경우, crafty, gcc, vortex를 제외하고 대부분의 벤치마크들은 성능이 향상되었다. 위의 세 개의 프로그램들은 주로 명령어 캐쉬 미스로 인하여 성능이 증가하지 못하였다. 단일 분기 예측에 비하여 평균 성능의 향상률은 2.3%이고, 성능의 조화평균은 모두 1.7 IPC를 기록하였으며, 상대오차는 4.6%이다. 매 싸이클 당 3 개의 분기어를 예측하였을 때, 단일 분기 예측에 대하여 성능의 향상률은 2.1%에 불과하였으며 2 개의 분기어를 예측하였을 때의 평균 성능을 능가하지 못하였다. 그 이유는 소규모 하드웨어 사양으로 인하여 분기 오류에 의한 페날티, 명령어 및 데이타 캐쉬 미스가 성능을 크게 감소시켰기 때문이다. 이 때, 성능의 조화평균은 1.7 IPC, 상대오차의 평균은 5.4%를 기록하였다.

그림 4는 같은 조건에서 실수형 벤치마크에 대한 결과를 보인 것이다. 대부분의 실수형 프로그램에서, 성능에 영향을 미치는 요인은 연산 유닛의 부족이 아닌 분기

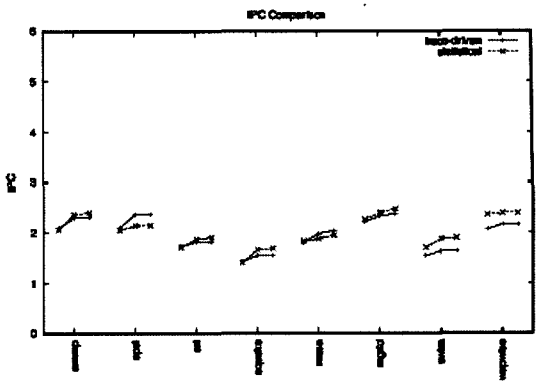

그림 4 실수형 벤치마크 성능의 비교(w = 16)

예측 오류와 데이타 캐쉬 미스로부터 기인하였다. 단일 분기 예측을 수행하였을 때, mgrid가 96%의 높은 분기 예측율과 높은 명령어 캐쉬 히트율로 인하여 최고 2.2 IPC를 기록하였다. 일반적인 트레이스 구동 모의실험과 통계적 모의실험에 의한 성능의 조화평균은 모두 1.9 IPC이며, 상대오차의 평균은 4.4%를 기록하였다. 매 싸이클 당 2 개의 분기어를 예측하였을 때, apsi가 최고의 성능 향상을 보이며 mgrid를 추월하였다. 단일 분기 예측에 대한 성능의 평균 향상률은 7.6%이며, 이것은 정수형 벤치마크의 결과를 능가하는 것이다. 일반 모의실험과 통계적 모의실험은 모두 2.0 IPC의 조화평균과 7.2%의 상대오차를 기록하였다. 마지막으로 매 싸이클 당 3 개의 분기어를 예측하였을 때, 실수형 벤치마크의 성능은 단일 분기 예측에 비하여 평균 8.6% 증가하였으며, 이것은 역시 정수형 벤치마크의 결과를 능가하는 것이다. 이 때 성능의 조화평균은 2.0 IPC, 상대오차의 평균은 7.7%를 나타내었다.

5.7 중간 규모 프로세서 사양에 대한 성능의 비교

윈도우 크기가 32일 때, 정수형 벤치마크에 대한 모의실험 결과를 그림 5에 도시하였으며, 이 때, 인출율, 이슈율, 퇴거율 및 이용 가능한 연산 유닛의 개수도 함께 개선되었다. 단일 분기 예측일 때, 윈도우 크기가 16인 경우보다 전체 벤치마크의 성능이 평균 33% 향상되었다. 일반 모의실험과 통계적 모의실험에 대한 성능의 조화평균은 1.9 IPC를 나타내었으며, 상대오차의 평균은 3.9%를 기록하였다.

2 차의 다중 분기 예측에서, 단일 분기 예측에 대한 성능의 평균 증가율은 8.6%를 나타내었다. 일반 모의실험과 통계적 모의실험은 각각 2.3 IPC와 2.4 IPC를 나타내었으며, 상대오차의 평균은 3.1%이다. 3 차의 다중 분기 예측을 시행할 때, crafty, gcc, vortex를 제외하고는 모든 벤치마크의 성능이 단일 분기 예측에 비하여 증가하였다. 단일 분기 예측에 대한 성능의 향상률

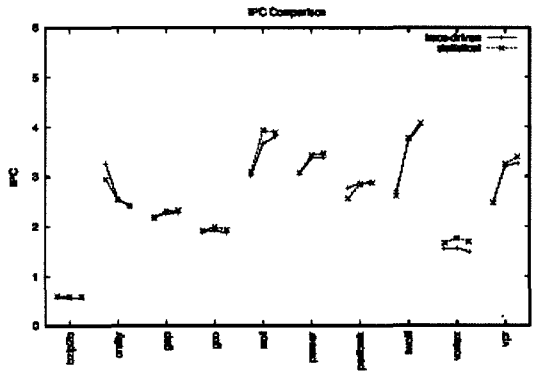

그림 5 정수형 벤치마크 성능의 비교(w = 32)

9.3%는 2 차의 다중 분기 예측의 결과를 능가한 것이다. 그러나 2 차에서 3 차의 다중 분기 예측으로 이행할 때 성능의 증가율은 0.1%에 불과하였다. 일반 모의실험과 본 모의실험의 성능의 조화평균은 각각 2.3 IPC와 2.4 IPC를 나타내었고 상대 오차의 평균은 2.9%이다.

한편, 윈도우 크기가 32일 때 실수형 벤치마크의 결과를 그림 6에 나타내었다. 일반 모의실험의 성능은 2.2 IPC를 보였고, 통계적 모의실험은 2.1 IPC를 나타내었으며, 상대오차의 평균은 3.7%이다. 2 차의 다중 분기 예측으로 인하여, 실수형 벤치마크는 단일 분기 예측에 비하여 13%의 성능 향상을 가져왔다. 단일 분기에서 2 차의 분기로 이행하였을 때 실수형 벤치마크가 정수형 벤치마크보다 더욱 높은 성능의 향상률을 기록하였다. 이 때, 상대오차의 평균은 9.8%로 다소 증가하였다. 마지막으로 3 차의 다중 분기 예측을 수행하였을 때, 단일 분기 예측에 비하여 15%의 성능 향상을 가져왔으며, 이것은 역시 정수형 벤치마크의 경우를 능가하는 것이다. 일반 모의실험과 통계적 모의실험의 IPC는 각각 2.7과 2.9를 기록하였으며, 상대오차의 평균은 10.8%이다.

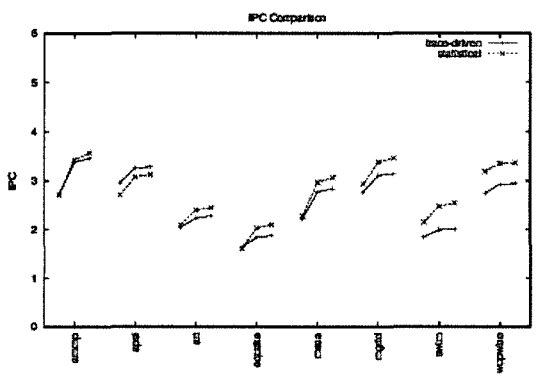

그림 6 실수형 벤치마크 성능의 비교(w = 32)

5.8 대규모 프로세서 사양에 대한 성능 비교

윈도우 크기가 64인 경우의 정수형 벤치마크의 모의실험 결과를 그림 7에 나타내었다. 단일 분기 예측을 시행할 때, 연산 유닛의 부족으로 인한 성능의 저하는 거의 없으며, 주로 분기 예측 미스와 데이타 캐쉬 미스가 성능에 영향을 끼치는 주 요소이다. 일반 모의실험과 통계적 모의실험의 성능의 조화평균은 각각 2.6 IPC와 2.5 IPC를 나타내었으며 상대오차의 평균은 2.7%이다.

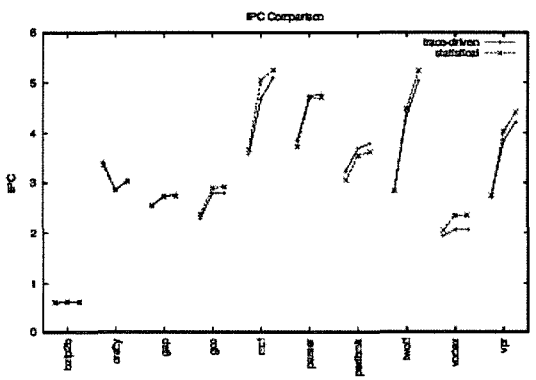

그림 7 정수형 벤치마크 성능의 비교(w = 64)

정수형 벤치마크에 대하여 매 싸이클 당 2 개의 분기어를 예측하였을 때, 단일 분기 예측의 결과에 대하여 18%의 성능 향상을 보였다. 일반 모의실험과 통계적 모의실험은 각각 2.9 IPC 및 3.0 IPC의 조화평균을 나타내었으며, 상대오차의 평균은 4.0%이다. 2 차에서 3 차의 다중 분기 예측으로 이행하였을 때, 대부분의 프로그램의 성능이 개선되었으나, 그 증가율은 다소 둔화되었다. 그러나 윈도우 크기가 32인 경우와 비교하면 64인 경우가 성능의 향상이 뚜렷하게 나타났다. 평균적으로, 3 차의 분기 예측 결과는 단일 분기 예측에 대하여 25%, 2 차의 분기 예측에 대하여 4.8%의 성능 향상률을 가져왔다. 일반 모의실험과 통계적 모의실험은 각각 3.0 IPC 및 3.1 IPC를 나타내었으며, 상대오차의 평균은 3.9%이다.

실수형 벤치마크의 경우 단일 분기 예측을 시행하였을 때, wupwise가 윈도우 크기 32인 경우에 비교하여 최고 35%의 성능 향상을 나타내면서 최고를 기록하였다. 전체 프로그램은 평균 17% 성능 향상률을 보였다. 일반 모의실험은 2.7 IPC를, 통계적 모의실험은 2.8 IPC를 나타내었으며, 상대오차의 평균은 6.0%이다. 2 개의 분기를 예측할 때, ammp가 최고로 성능이 향상되어 4.3 IPC를 기록하였으며, 단일 분기 예측에 대한전체 프로그램의 평균 성능 향상률은 19.2%이다. 일반 모의실

험과 통계적 모의실험은 각각 3.5 IPC와 3.6 IPC를 나타내었고 상대오차의 평균은 10.3%이다. 마지막으로 3 개의 분기를 예측할 때, 단일 분기 예측과 2 개의 분기 예측에 대한 성능의 평균 상승률은 26.0%와 5.5%이다. 일반 모의실험과 통계적 모의실험은 각각 3.4 IPC와 3.7 IPC를 기록하였으며 상대오차의 평균은 10.2%이다. 모의실험의 전 과정을 통하여 다중 분기 예측법은 정수형 벤치마크보다 실수형 벤치마크에 대하여 더욱 좋은 결과를 나타내었으며, 이것은 [3]에서 시행된 연구 결과와도 일치한다.

6. 결 론

본 논문에서는, 퍼셉트론 방식의 다중 분기 예측법을 이용하는 수퍼스칼라 마이크로 프로세서에 대하여 통계적 프로화일링 기법을 적용하였으며, 이 때 2 개 및 3 개의 연속적인 블럭에 대한 예측율을 이용하는 것을 제안하였다. 이것을 위하여 대규모 윈도우와 높은 인출 대역폭을 갖는 프로세서에 SPEC 2000 정수형 및 실수형 벤치마크를 입력으로 이용하여 그 모델의 정확성을 모의실험으로 측정하였다. 그 결과, 단일 분기 예측에 대하여 상대 오차의 평균은 8% 미만을 유지하였으며, 2 차 및 3 차의 다중 분기 예측에서도 그 값이 11%를 넘지 않았다. 단일 분기 예측에서 다중 분기 예측으로 이행함에 따라 상대오차는 다소 증가하였지만 통계적 모의실험에 의하여 얻은 성능은 일반적인 트레이스 구동 모의실험과 동일한 경향을 보였으며 매우 높은 정확도를 견지하였다.

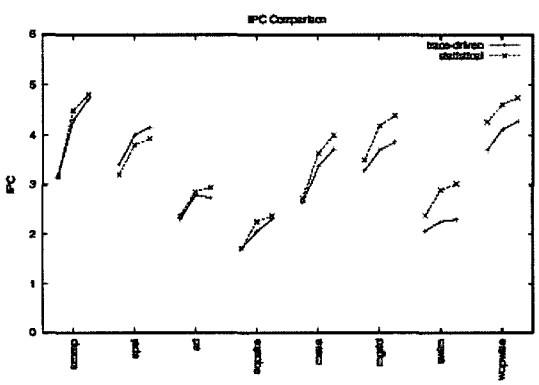

그림 8 실수형 벤치마크 성능의 비교(w = 64)

명령어 수준 병렬성에 국한하였을 때는 정수형 프로그램이 다중 분기 예측법에 의하여 성능이 향상될 더 큰 잠재력을 가지고 있으나, 모의실험 결과 각종 캐쉬의 영향으로 기대했던 만큼 실수형 벤치마크의 성능 향상을 능가하지 못하였다. 같은 이유로, 3 차의 다중 분기 예측 결과가 2 차의 다중 분기 예측 결과를 능가하지 못하는 경우도 일부 발생하였다.

통계적 모의실험에 대한 정확도를 더욱 향상시키기 위하여 향 후 다음과 같은 방법을 모색할 수 있다. 첫째, 통계적 프로화일링 단계에서 상호 종속인 명령어 쌍에 대한 정보를 추가하여 더욱 구체적인 분석을 시행하는 것이다. 두번째, 벤치마크 프로그램의 전체 단위로 통계적 프로화일링을 수행하는 대신에 각 기본블럭 단위로 더욱 세밀한 프로화일링을 수행하는 것을 들 수 있다. 마지막으로, 프로그램의 동적 흐름에 대한 정보를 프로화일링에 결합시키는 것이다.

참 고 문 헌

[1] S. Nussbaum and J. E. Smith, "Modeling Superscalar Processors via Statistical Simulation," in International Conference on Parallel Architectures and Compilation Techniques, Sep. 2001, pp. 15-24.

[2] L. Eeckout, R. H. Bell Jr., B. Stougie, K. D. Bosschere, and L. K. John, "Control Flow Modeling in Statistical Simulation for Accurate and Efficient Processor Design Studies," in International Symposium on Performance Analysis of Systems and Software, 2004.

[3] T-Y. Yeh, D. T. Marr, and Y. N. Patt, "Increasing the Instruction Fetch Rate via Multiple Branch Prediction and a Branch Address Cache," in The 7th International Conference on Supercomputing, Jul. 1993, pp. 67-76.

[4] R. Rakvic, B. Black, and J. P. Shen, "Completion Time Multiple Branch Prediction for Enhancing Trace Cache Performance," in Annual International Symposium on Computer Architecture, 2000, pp. 47-58.

[5] E. Rotenberg, S. Benett, and J. E. Smith, "Trace Cache : a Low Latency Approach to High Bandwidth Instruction Fetching," in Proceedings of the 29th Annual International Symposium on Microarchitecture, Dec. 1996, pp. 24-34.

[6] D. A. Jimenez and C. Lin, "Dynamic Branch Prediction with Perceptrons," in Proceedings of the Seventh International Symposium on High Performance Computer Architecture, 2001. pp. 197-206.

[7] D. A. Jimenez and C. Lin, "Neural Methods for Dynamic Branch Prediction," ACM Transactions on Computer Systems, vol. 20, pp. 369-397, 2002.

[8] T-Y. Yeh and Y. N. Patt, "Two-Level Adaptive Branch Prediction," in The 24th ACM/IEEE International Symposium and Workshop on Microarchitectures, Nov. 1991, pp. 51-61.

[9] The SPARC Architecture Manual, Prentice-Hall, Inc., 1992.

[10] Introduction to Shade, Sun Microsystems, Inc., Jun. 1997.

이 종 복

1988년 서울대학교 컴퓨터공학과 학사
1990년 서울대학교 컴퓨터공학과 석사
1998년 서울대학교 전기공학부 박사
1998년~2000년 LG 반도체 선임연구원
2000년~현재 한성대학교 정보통신공학과 교수. 관심분야는 마이크로 프로세서 구조, 분기 예측 알고리즘, 프로세서 성능 모델 등