다양한 수퍼스칼라 마이크로 프로세서 사양에 대한 통계적 모의실험

한성대학교 정보통신공학과
이종복
jblee@hansung.ac.kr

Statistical Simulation for Various Superscalar Microprocessor Configurations

Jongbok Lee
Dept. of Information and Communications Engineering, Hansung University

요 약

마이크로 프로세서 구조의 성능을 분석할 때, 트레이스 구동형 모의실험이 광범위하게 수행되고 있으나, 시간과 공간을 많이 차지하기 때문에 비실용적이다. 본 논문에서는 통계적 프로화일링 기법을 이용하여 다양한 하드웨어 사양을 갖는 수퍼스칼라 마이크로 프로세서의 성능을 통계적 모의실험에 의하여 측정하는 기법에 대하여 연구하였다. 이것을 위하여 SPEC 2000 벤치마크 프로그램의 특성을 통계적 프로화일링 기법으로 모델링하고 여기서 얻은 통계적 프로화일을 바탕으로 벤치마크 트레이스를 합성하여 모의실험을 수행하였다. 그 결과, 다양한 하드웨어 구성에 대하여 비교적 높은 정확도를 얻을 수 있었다.

1. 개요

컴퓨터 구조의 개발 단계에서 성능을 평가하기 위하여 주로 이용되는 트레이스 구동 모의실험 (trace-driven simulation)은 비교적 정확하다는 장점이 있으나, 공간과 시간이 매우 많이 소요되며, 하드웨어 사양이 바뀔 때마다 모의실험을 다시 반복해야하는 단점이 있다.

최근에 들어서 주목을 끌고 있는 방법은 프로세서와 프로그램의 통계적 특성을 수집하고 그것을 바탕으로 새로운 입력 트레이스를 합성하여, 이것을 확률적으로 모의실험하는 통계적 프로화일링 (statistical profiling) 방법이다 [1, 2]. 이 방법을 이용하여, 기존의 일반적인 트레이스 구동 모의실험 보다 짧은 시간에 마이크로 프로세서의 성능을 비교적 정확히 측정할 수 있다.

본 논문에서는 SPEC 2000 벤치마크의 정수형 프로그램 10개와 실수형 프로그램 8 개를 대상으로 다양한 하드웨어 사양을 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. 2 장에서는 통계적 프로화일링 기법에 대하여 논하고, 3 장에서는 모의실험 환경을 다룬다. 4 장에서 모의실험결과를 보이고, 5 장에서 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

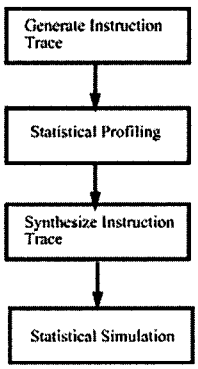


그림 1: 통계적 모의실험의 흐름도

위한 분석 및 통계 데이타의 수집 단계는 다음과 같다. 우선, 첫 단계에서 얻은 프로그램의 명령어 트레이스를 분석하여 프로그램의 고유한 특성과 지역적 특성에 대한 통계값들을 추출해낸다. 프로그램의 고유한 특성은 프로그램을 구성하는 명령어의 유형별 구성비와 레지스터 피연산자의 개수 및 명령어 간의 데이터 종속에 대한 분포로 구성된다. 이러한 특성은 주어진 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. 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 함수 (cumulative distribution function) 에 대응시키는 것이다. 각 통계 데이타의 누적 분포에서의 어느 특정한 지점을 선택함으로써, 명령어의 유형, 피연산자의 개수, 피연산자의 종속거리를 결정하여 명령어 코드를 만들어낼 수 있기 때문에 이와같은 작업을 반복하면 통계적 트레이스의 합성이 가능하다.

마지막으로, 이렇게 합성된 트레이스를 분기 미스율 및 캐쉬 미스율에 대한 정보와 함께 수퍼스칼라 프로세서 모의실험기로 입력한다. 이 때 사용되는 프로세서 모델은, 실제로 값을 계산하고 결과를 저장하거나 메모리 계층을 모델링할 필요가 없으므로 매우 간단하다. 이와 같이 난수를 이용하여 발생한 합성 트레이스를 통계 모의실험기로 처리하면서 수천 싸이클 동안 결과의 표준편차가 적정한 수준 미만일 때 그 성능이 수렴된 것으로 간주한다. 이것은 보통 모의실험을 시작한지 수 만 싸이클 만에 달성된다. 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. 모의실험 환경

3.1 수퍼스칼라 프로세서

본 논문에서는 명령어 윈도우에서 동적 스케쥴링을 관리하는 수퍼스칼라의 기본형을 이용하였으며 명령어 코드 발생을 위하여 Supersparc 명령어 세트를 이용하였으며 [3], 수퍼스칼라 모의실험기는 SimpleScalar와 유사한 형태로 개발되었다 [4].

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

표 1: 아키텍쳐 사양.

유형	값
윈도우 크기	16/32/64
인출율	4/8/16
이슈율	4/8/16
퇴거율	4/8/16
연산 유닛	4/8/16 정수형 산술논리연산 유닛, 2/4/8 로드/스토어 유닛, 1/2/4 실수형 덧셈기, 1/2/4 실수형 곱셈기
L1 명령어 캐쉬	64 KB, 2 차 세트 연관,16 B 블럭, 미스 지연 10 싸이클
L1 데이타 캐쉬	64 KB, 직접 매핑,32 B 블럭 미스 지연 10 싸이클
분기 주소 캐쉬	2K 항목
분기 예측기	16 비트 전역 히스토리 방식 예측 미스 지연 6 싸이클

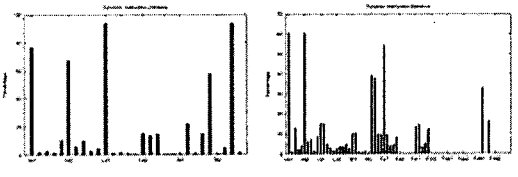
3.2 벤치마크 프로그램

입력 벤치마크 프로그램으로는 10 개의 SPEC 2000 정수형 프로그램과 8 개의 SPEC 2000 실수형 프로그램이 사용되었다. 이 프로그램을 SunOS 5.6 운영체제하의 Sun Sparc Ultra-2 머신에서 C 컴파일러를 이용하여 실행 가능 화일을 얻었다. 이 실행 가능 화일과 Shade를 이용하여 Sparc V9 명령어 트레이스를 생성하였다 [5]. Sparc V9 명령어는 Sparc의 기본 명령어 집합에 그래픽 전용 명령어가 추가된 것이다. 초기의 일반적인 트레이스 구동 모의실험에서 각 프로그램마다 1 천만 개의 명령어를 발생시켜 모의실험에 이용하였다.

4. 모의실험 결과

4.1 명령어 유형별 종속거리의 분포율

그림 2에 정수형 프로그램 gap과 실수형 프로그램 swim에 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 소수점 미만의 종속거리 분포율도 중요하다.

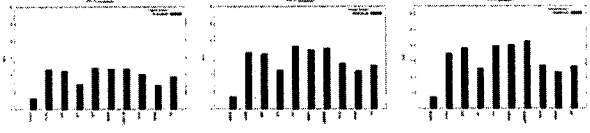


(a) gap (b) swim

그림 2: 유형별 명령어의 종속거리 분포율.

4.2 다양한 윈도우, 인출율 및 연산 유닛의 크기에 따르는 비교

그림 3에서 다양한 윈도우, 인출율 및 연산 유닛 크기를 가지며 단일 분기 예측일 때 트레이스 구동 모의실험으로 측정한 성능과 통계적 프로화일링 모델에 의하여 측정한 성능을 IPC (Instruction Per Cycle)로 나타내었다. 이 때의 사양은 표 1를 따르는데, 예를 들어 윈도우의 크기가 16일 때는 인출율, 이슈율, 퇴거율이 모두 4이고 정수형 산술 논리 연산 유닛은 4 개, 로드 스토어 유닛은 2 개이며, 실수형 덧셈기 및 곱셈기는 각각 1 개이다. 그림 4.2(a)는 윈도우 크기가 16이고 인출율이 4인 경우이다. 성능의 조화평균은 1.7을 나타내었으며, 이 때 상대오차의 평균은 3.8 %를 기록하였다. Bzip2는 데이타 캐쉬 히트율이 71.6 %에 불과하여 잠재적으로 명령어 수준 병렬성에서 도달 가능한 성능의 92 %가 각종 캐쉬에 의하여 감소되어 윈도우 크기 및 연산 유닛의 증가에 가장 둔감하다. 그림 4.2(b)는



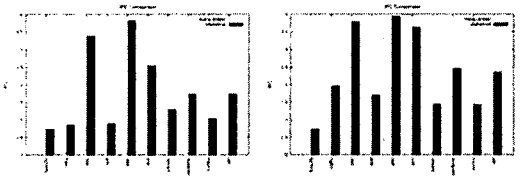
(a) w = 16 (b) w = 32 (c) w = 64

그림 3: 다양한 윈도우 크기 및 연산 유닛 사양에 대한 성능 비교.

윈도우 크기가 32이고 인출율이 8인 경우이다. 이와같이 윈도우 크기가 2 배가 되고 인출율 및 연산유닛의 크기가 증가하였을 때 crafty의 성능이 최고 52 %만큼 증가하였으며, 전체적으

로 평균 37 % 향상하였다. 이 때 성능의 조화평균은 실측에 의한 값이 1.7, 통계에 의한 값이 1.8을 나타내었으며, 상대오차의 평균은 4.9 %를 기록하였다. 마지막으로 그림 4.2(c)에서 윈도우의 크기가 32에서 64로 증가하고 인출율이 16으로 개선되었을 때의 결과를 보였다. 윈도우의 크기가 64인 경우에 연산 유닛의 개수가 충분하여 bzip2와 vortex를 제외하고는 연산 유닛의 부족으로 인한 성능의 제약은 나타나지 않았으며, 주로 캐쉬에 의하여 성능이 저하되었다. 윈도우의 크기 32인 경우에 비하여 gap이 24.3 %의 최고 성능 향상을 나타냈으며 전체적으로는 평균 12.5 % 개선되었다. 이 때, 성능의 조화평균은 실측과 통계 모두 1.9를, 상대오차의 평균은 2.7 %를 기록하였다.

4.3 다양한 캐쉬 크기에 의한 성능의 비교

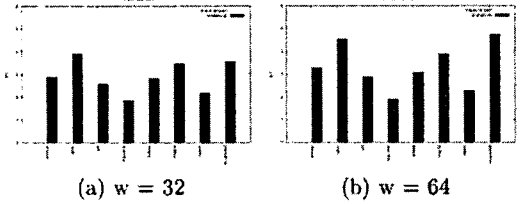


(a) 16k (b) 32k

그림 4: 다양한 캐쉬 크기에 의한 성능의 비교.

그림 4(a)는 명령어 캐쉬와 데이타 캐쉬가 각각 16 KB이고, 분기 주소 캐쉬의 엔트리가 512 개일 때의 정수형 프로그램에 대한 성능 비교 결과이다. 충분히 크지 못한 명령어 캐쉬의 용량에 의하여, Gcc의 명령어 캐쉬 히트율이 53.4 %, perlbmk와 vortex가 각각 63.1 %와 65.2 %를 기록하였다. 한편, 분기 주소 캐쉬의 용량도 부족하여, gcc, perlbmk, vortex에서 분기 주소 캐쉬의 히트율이 약 40 % 미만으로 극히 저조하였다. 이와같은 캐쉬의 낮은 히트율이 성능에 큰 제약을 가하여, 실측한 성능의 조화평균은 1.3에 불과하였다. 통계 모델에 의한 성능의 조화평균은 1.4를 기록하였으며, 상대오차의 평균은 6.6 %를 나타내었다. 한편, 그림 4(b)는 명령어 캐쉬와 데이타 캐쉬가 그 2 배인 32 KB이고, 분기 주소 캐쉬의 엔트리가 1024개일 때의 성능 비교 결과이다. 이 때 gcc의 명령어 캐쉬와 분기 주소 캐쉬의 히트율이 각각 83.5 %와 70.9 %로 증가하였으며, vortex도 각 히트율이 78.8 %와 79.2 % 로 증가하였다. 성능의 조화평균은 실측에 의한 값과 통계에 의한 값이 모두 1.8을, 상대오차의 평균은 4.9 %를 기록하였다. 캐쉬 크기를 2 배로 증가시킴에 따라 전체 성능이 1.4 배 향상된 것을 알 수 있다.

4.4 실수형 벤치마크에 의한 성능의 비교



(a) w = 32 (b) w = 64

그림 5: 실수형 벤치마크에 의한 성능의 비교.

그림 5(a)는 윈도우의 크기가 32일 때 SPEC 2000 실수형 벤치마크에 대한 모의실험 결과를 나타내었다. 이 때 명령어 캐쉬 및 데이타 캐쉬는 64 KB의 크기로 구성되며 분기 주소 캐쉬는 2048 개의 항목을 갖는다. 실수형 벤치마크의 경우 연산 유닛의 부족으로 인한 성능 하락은 없고 캐쉬에 의한 제약으로 equake에서 최고 55.4 %의 성능 손실을 가져왔으며, 평균적으로 37.7 %의 성능이 하락하였다. 그림 5(b)에서와 같이 윈도우 크기를 32에서 64로 증가시키고 인출율 및 연산 유닛의 크기를 개선하였을 때 wupwise에서 최고 36 %, 전체적으로는 평균 18 %의 성능이 증가되었다. 윈도우의 크기가 2 배가 된 만큼 성능이 향상될 여지가 증가하여, 캐쉬에 의한 성능의 제약은 평균 42.5 %를 기록하였다. 윈도우의 크기가 32인 경우 성능의 기하 평균값은 실측에 의한 값이 2.5, 통계에 의한 값이 2.8을, 윈도우의 크기가 64인 경우 실측에 의한 값이 2.9, 통계에 의한 값이 3.0을 기록하였다. 한편, 상대 오차의 평균값은 각각 6.6 %와 9.9 %를 기록하였다.

5. 결론

본 논문에서는, 다양한 크기의 윈도우와 인출율 및 연산 유닛과 각종 캐쉬 효과를 포함하는 수퍼스칼라 마이크로 프로세서에 대하여 통계적 프로화일링 기법을 적용하였다. 이것을 위하여 대규모 윈도우와 높은 인출 대역폭을 갖는 프로세서에 SPEC 2000 정수형 및 실수형 벤치마크를 입력으로 이용하여 그 모델의 정확성을 모의실험으로 측정하였다.

그 결과, 기존의 트레이스 구동 모의실험으로 측정한 값에 비하여 새롭게 합성된 통계적 트레이스를 입력으로 하여 모의실험이 매우 빠르게 최종 결과에 수렴하였으며, 다양한 윈도우 크기 및 연산 유닛의 개수에 대한 마이크로 프로세서의 사양에 따라 정수형 벤치마크의 상대오차의 평균값이 최저 2.7 %에서 최대 4.9 %, 다양한 캐쉬 사양에 대하여 4.9 %에서 6.6 %, 실수형 벤치마크에서 에서 6.6 %에서 9.9 % 범위의 비교적 정확한 결과를 얻었다.

통계적 모의실험 방법은 기존의 트레이스 구동 모의실험보다 시간이 적게 소요되는 장점 이외에, 임의의 벤치마크 프로그램에 대하여 통계적 프로화일링 자료를 얻은 후에는 다양한 하드웨어 환경을 갖는 프로세서의 모의실험을 시행할 수 있으므로 성능 평가에 대한 유연성과 편리성을 제공한다. 이것으로 마이크로 프로세서의 설계 초기 단계에서 통계적 프로화일링 기법이 프로세서의 성능을 평가하는데 적합한 방법임을 알 수 있었다.

오차가 상대적으로 큰 일부 프로그램의 경우, 명령어의 유형 및 종속율 이루는 명령어의 쌍에 대한 정보를 추가하여 더욱 세분화하거나, 통계적 프로화일링을 전체 프로그램에 대하여 시행하지 않고 기본 블럭에 대하여 시행하는 방법 및 프로그램의 동적 흐름을 파악하여 그 통계적 성질을 보강한다면 그 오차를 더욱 줄일 수 있다.

참고 문헌

[1] R. Carl and J. E. Smith, "Modeling Superscalar Processors via Statistical Simulaton," in *Workshop on Performance Analysis and Its Impact on Design*, Jun. 1998.

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

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

[4] T. Austin, E. Larson, and D. Ernest, "SimpleScalar : An Infrastructure for Computer System Modeling," *Computer*, vol. 35, no. 2, pp. 59–67, Feb. 2002.

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