수퍼스칼라 프로세서에서 윈도우의 크기와 이슈폭에 대한 관계

이종복°
한성대학교 정보통신공학과
jblee@hansung.ac.kr°

The Relationship Between the Window Size and the Issue Width

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

요 약

수퍼스칼라 프로세서 구조의 성능을 분석할 때, 실행 구동형 모의실험 및 트레이스 구동형 모의실험이 광범위하게 수행되고 있으나, 시간과 공간을 많이 차지하고 또한 성능에 대한 근본 원리를 규명하기 어려운 단점이 있다. 본 논문은 수퍼스칼라 프로세서의 성능에 대하여 통찰력을 갖고, 이것을 기반으로 수퍼스칼라 프로세서의 모델을 마련하기 위하여 수퍼스칼라 프로세서의 윈도우의 크기와 이슈폭에 대한 관계를 규명하였다. 이것을 위하여 SPEC 2000 정수형 벤치마크 프로그램을 입력으로 하는 트레이스 구동 모의실험을 통하여 윈도우의 크기와 매 싸이클당 이슈되는 명령어의 개수에 대한 관계식을 도출하였으며,그 정확도는 평균 9.5 %를 기록하였다.

1. 서 론

컴퓨터 구조의 개발 단계에서 성능을 평가하기 위하여 주로 이용되는 simplescalar와 같은 실행 구동 모의실험(execution-driven simulation)[1] 또는 트레이스 구동 모의실험 (trace-driven simulation)은 비교적 정확하다는 장점이 있으나, 공간과 시간이 적잖게 소요되며, 프로세서 내부에서 발생하는 일에 대한 통찰력을 제공해주지 못하는 단점이 있다. 최근에는 통계적 모의실험이 명령어 트레이스를 새로 합성하는 방법으로 인하여 시간을 단축할 수 있어서 각광을 받고 있으나, 프로세서 내부의 원리를 설명하는데는 역시 부족하다 [2,3,4,5].

이러한 모의실험 위주에 대한 대안으로 프로세서의 분석적 모델을 들 수 있다 [6,7,8]. 분석적 모델은 모의실험 방식보다 시간이 적게 소요되고 프로세서의 동작에 대하여 통찰력을 얻을 수 있게 해준다. 그러나 오늘날 수퍼스칼라 프로세서 하드웨어의 높은 복잡도로 인하여 분석 모델을 통하여 높은 정확도를 얻기에는 어려움이 많다. 본 논문에서는 수퍼스칼라 프로세서의 분석 모델을 마련하기 위하여 수퍼스칼라 프로세서의 윈도우의 크기와 이슈폭에 대한 관계를 규명하였다. 이것을 위하여, SPEC 2000 벤치마크의 정수형 프로그램 10 개를 대상으로 다양한 윈도우 크기를 갖는 수퍼스칼라 마이크로 프로세서의 이슈폭을 측정하고 그 관계식을 도출하였으며, 실측값과 관계식을 통한 값을 비교하여 그 정확도를 평가하였다.

본 논문은 다음과 같이 구성된다. 2 장에서는 수퍼스칼라 프로세서의 윈도우의 크기와 이슈폭에 대하여 논하고, 3 장에서는 모의실험 환경을 다룬다. 4 장에서 모의실험결과를 보이고, 5 장에서 결론을 맺는다.

2. 윈도우의 크기와 이슈폭의 관계

수퍼스칼라 프로세서의 윈도우의 크기와 실제로 이슈되는 명령어의 평균 개수에 대한 관계는 수퍼스칼라 프로세서의 성능을 도출해내는데 매우 중요하다. Riseman과 Foster의 초기 연구에 의하면 [9], 매 싸이클당 이슈할 수 있는 명령어의 개수는 윈도우 크기의 제곱근에 해당한다고 발표하였으며, 최근에 이르러 Michaud와 Tejas는 윈도우의 크기와 이슈되는 명령어의 개수는 거듭제곱 성질을 따른다고 하였다 [10][11][12].

이 때, 로그 스케일에서의 거듭제곱 직선의 기울기는 0.5이므로 이것은 역시 제곱근 관계를 나타내었다. 이렇게 해서 얻은 윈도우의 크기 W와 명령어 이슈폭 I에 대하여 거듭제곱 관계식인 $I=\alpha w^{\beta}$의 그래프의 양변에 로그를 취하면 $\log_2 I=\log_2\alpha+\beta\log_2 W$가 되어 $y=a+bx$의 방정식의 형태가 된다. 주어진 데이타 세트가 (x_1,y_1), $(x_2,y_2),\ldots(x_n,y_n)$일 때 a와 b는 각각 수식 1,2와 같이 최소자승법(Least Square Method)을 이용하여 구할 수 있다.

$$a = \frac{(\sum_{i=1}^{n} x_i^2)(\sum_{i=1}^{n} y_i) - (\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} x_i y_i)}{n(\sum_{i=1}^{n} x_i^2) - (\sum_{i=1}^{n} x_i)^2}$$

수식 1

$$b = \frac{(\sum_{i=1}^{n} x_i y_i) - n(\sum_{i=1}^{n} x_i y_i)}{(\sum_{i=1}^{n} x_i)^2 - n(\sum_{i=1}^{n} x_i^2)}$$

수식 2

따라서, α와 β를 구함으로써, $I = \alpha W^{\beta}$ 관계식을 이용하여 주어진 벤치마크 별로 특정한 윈도우 크기에 대하여 명령어 이슈폭을 계산으로 구할 수 있다. 이러한 윈도우-이슈폭 관계 그래프는 특정한 하드웨어의 구현과는 상관없이 각 벤치마크의 기본적인 레지스터 기반의 데이타 종속에만 의존하는 고유한 특성이다.

3. 모의실험 환경

3.1 수퍼스칼라 프로세서

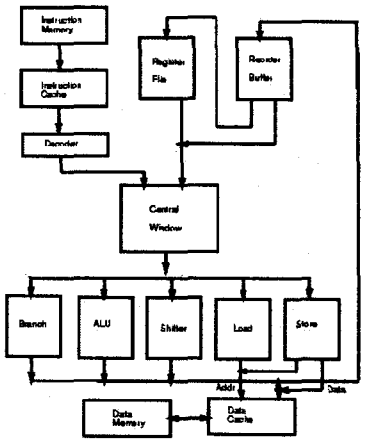

그림 1 수퍼스칼라 프로세서의 기본형.

본 논문에서는 그림 1과 같이 명령어 윈도우에서 동적 스케줄링을 관리하는 수퍼스칼라의 기본형을 이용하였다. 이 때, 순수한 윈도우의 크기와 명령어 이슈폭의 관계를 규명하기 위하여 모든 명령어가 단위 싸이클만에 실행되며, 무한한 연산유닛을 갖고 무한한 이슈폭을 가지며 미스 이벤트가 발생하지 않는 이상적인 수퍼스칼라 프로세서를 대상으로 하였다. 이 때, 크기가 제한되는 유일한 성분은 윈도우의 크기이며, 크기 4 부터 64에 이르는 5 가지로 설정하여 실험하였다. 명령어 코드 발생을 위하여 Supersparc 명령어 세트를 이용하였으며, 수퍼스칼라 모의실험기는 simplescalar와 유사한 형태로 개발되었다. 수퍼스칼라 프로세서는 무한히 공급되는 명령어를 인출부를 통하여 받아들인다. 명령어들은 파이프라인의 단계를 거쳐서 결국 명령어 윈도우에 삽입된다. 명령어 윈도우에 남은 공간이 있는한, 최대 인출율에 맞추어 명령어를 삽입할 수 있지만, 단일 분기 예측의 경우 분기 명령어를 만나면 그 싸이클에서는 더 이상의 명령어의 윈도우 삽입을 중단한다. 윈도우 내의 명령어는 Tomasulo 알고리즘에 의하여 비순차 실행 (out-of-order execution) 될 수 있으며 [13], 실제 종속 (true dependency)만이 명령어를 이슈하는데 있어서의 장애요인이다. 이 때, 윈도우 내의 명령어들은 윈도우 크기-명령어 이슈폭의 특성에 의하여 이슈된다.

만일 명령어를 이슈하였을 당시에 분기 예측 미스가 발생하면, 유용한 명령어의 인출이 중단되며, 윈도우 내의 명령어가 모두 소진되어야 명령어 인출이 다시 재개된다. 분기 예측을 위하여 비교적 높은 정확도를 나타내는 2 단계 적응형 분기 예측법을 이용하였다 [14,15]. 이슈된 명령어는 즉시 윈도우에서 삭제되지는 않으며, 선행하는 명령어가 모두 삭제된 이후에야 비로소 삭제된다. 이것은 simplescalar에서 RUU(Register Update Unit)를 FIFO 큐 방식으로 동작시켜서, 큐의 바닥에 도착한 명령어 순서대로 삭제하는 원리와 같다. 따라서 명령어들은 인터럽트나 트랩에 대비하여 프로세서의 상태를 보존하기 위해서 순차 종료 (in-order completion) 된다.

3.2 벤치마크 프로그램

입력 벤치마크 프로그램으로는 표 1에 보인 바와 같이 10 개의 SPEC 2000 정수형 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 [16]. Sparc V9 명령어는 Sparc의 기본 명령어 집합에 그래픽 전용 명령어가 추가된 것이다. 트레이스 구동 모의실험에서 각 프로그램마다 1 천만 개의 명령어를 발생시켜 모의실험에 이용하였다.

벤치마크	설명
bzip2	압축
crafty	체스 경기 놀이
gap	그룹 이론 해석기
gcc	C 프로그래밍 언어 컴파일러
mcf	조합 최적화
parser	워드 프로세서
perlbmk	PERL 프로그래밍 언어
twolf	배선 및 배치 모의실험기
vortex	객체지향형 데이터베이스
vpr	FPGA 회로 배치 및 배선

표 1 SPEC 2000 정수형 벤치마크 프로그램

3.3 모의실험기

본 논문에서 윈도우의 크기와 명령어 이슈폭을 측정하기 위하여 트레이스 구동 모의실험기가 필요하므로 리눅스 환경에서 C를 이용하여 자체적으로 개발하였다. 그림 2는 트레이스 구동 모의실험기에 대한 전 과정을 나타낸다.

```
main()
{
 config();
 init();
 fetch_instr();
 decode();
 issue();
 execute();
 commit();
}
```

그림 2 트레이스 구동 모의실험기

Config 함수에서는 모의실험기가 수행해야하는 명령어의 개수, 윈도우의 크기, 해독율, 이슈율, 퇴거율에 대한 입력을 받아들여서 설정한다. Init 함수에서는 레지스터 화일을 비롯한 각종 하드웨어를 초기화한다. Fetch_insr 함수에서는 한 싸이클에 최대 인출율에 해당하는 명령어를 명령어 캐쉬로 부터 인출하여 디코더로 가져온다.

Decode 함수에서 명령어들을 해독하고 각 명령어의 타임스탬프 (timestamp)를 부여한다 [17]. 타임스탬프는, 명령어간에 실제 종속 (true dependency)을 제외한 종속 관계를 제거하는 재명명(renaming)과 더불어, 윈도우 내의 각 명령어가 이슈되는 시점을 설정하는 효과가 있다. 타임스탬프를 이용하여 명령어의 종속관계를 구현하기 위하여 디코드된 명령어마다 타임스탬프를 가져야 하고, 또한 각 레지스터의 타임스탬프를 기록하는 레지스터-타임스탬프(RT) 테이블을 유지한다.

초기에 명령어의 타임스탬프는 현재 싸이클로 설정되고, 각 레지스터의 타임스탬프는 0으로 설정된다. 임의의 명령어는 자기가 가진 소오스 레지스터의 타임스탬프를 RT 테이블을 조회함으로써 알 수 있다. 어떤 명령어의 소오스 레지스터 1 개 또는 2 개의 타임스탬프 값을 조회한 결과, 모두 명령어 자신의 타임스탬프보다 그 값이 작다면, 이 명령어는 독립이고 이슈 가능하다. 모든 이슈 가능한 명령어는 현재 싸이클에 그 명령어의 결과 지연 싸이클(result latency)을 더한 값으로 그 명령어의 타임스탬프를 설정한다. 그리고 그 명령어의 결과 레지스터의 타임스탬프도 이 값으로 갱신하여 RT 테이블에 수록한다.

한편, 어떤 명령어의 소오스 레지스터의 타임스탬프가 명령어 자신의 타임스탬프보다 그 값이 크다면, 이 명령어는 선행하는 명령어에 종속인 셈이다. 이 때는, 해당 소오스 레지스터의 타임스탬프에 자신의 결과 지연 싸이클 수를 더하여 명령어 자신의 타임스탬프 및 결과 레지스터의 타임스탬프를 갱신해야 한다. 만일 임의의 명령어의 1 개 또는 2 개의 소오스 레지스터의 타임스탬프가 모두 명령어 자신의 타임스탬프보다 크다면, 둘 중 큰 값을 명령어 및 결과 레지스터의 타임스탬프로 갱신해야한다. 이와같이 각 명령어들은 RT 테이블을 통하여 서로 레지스터의 타임스탬프 값을 주고 받음으로써 명령어 간의 종속관계에 기반한 실행이 가능하다.

4. 모의실험 결과

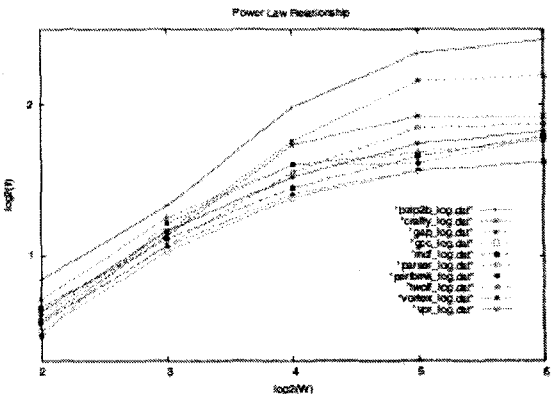

그림 3 윈도우 크기와 명령어 이슈폭의 거듭제곱 관계

그림 3은 10 개의 SPEC 2000 정수형 벤치마크에 대하여 윈도우의 크기와 명령어 이슈폭의 거듭제곱 관계를 로그 스케일로 그래프에 나타낸 것이다. 윈도우의 크기가 32일 때까지는 비교적 선형성을 나타내지만, 64에 이르러 다소 증가율이 둔화되는 현상을 관찰할 수 있다. 이것은 단일 분기 예측에 의하여 명령어 수준 병렬성이 기본 블럭 내에 머물러 있기 때문에 그 이상의 윈도우 크기 증가율이 이슈폭의 증가에 기여하지 못하기 때문에 발생한 것이다. 충분한 크기의 윈도우에 대하여도 명령어 이슈폭에 대한 선형성을 얻기 위하여 다중 분기 예측의 도입이 요구된다.

표 2는 이 그래프에 대하여 수식 1,2의 최소자승법을 이용하여 $I=\alpha W^{\beta}$에 곡선을 맞추어 α와 β값을 구한 것이다. α는 gap이 최저 0.97을, *twolf*가 최고 1.10을 나타내었으며 평균값은 1.03을 기록하였다. β는 *gcc*와 *vortex*가 최저 0.29를, *bzip2*가 최고 0.43을 기록하였으며 그 평균값은 0.34를 나타내었다.

그림 5는 실제로 측정한 이슈폭과 거듭제곱 관계를 이용하여 모델에 의하여 계산한 이슈폭의 평균오차를 각 벤

벤치마크	α	β
bzip2	1.02	0.43
crafty	1.01	0.36
gap	0.97	0.40
gcc	1.03	0.29
mcf	1.05	0.31
parser	1.00	0.34
perlbmk	1.03	0.32
twolf	1.10	0.31
vortex	1.05	0.29
vpr	1.04	0.32

표 2 각 벤치마크 프로그램의 거듭제곱 파라미터

치마크 프로그램 별로 4,8,16,32,64의 5 가지 윈도우 크기에 대하여 나타낸 것이다. *Mcf*에서 최저 6.31%를, *perlbmk*에서 최고 12.76 %의 평균 오차를 가져왔으며 평균 9.51 %를 기록하였다.

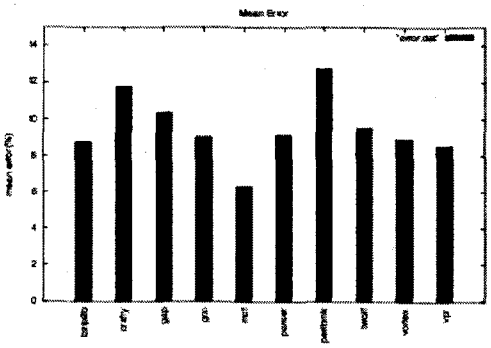

그림 4 실측 이슈폭과 거듭제곱근에 의한 이슈폭의 평균오차.

5. 결론

본 논문에서는, 수퍼스칼라 프로세서의 분석적 성능 모델의 기반을 마련하기 위하여 윈도우의 크기와 명령어 이슈폭에 대한 관계를 최소자승법을 이용하여 거듭제곱의 함수로 도출하였다. 이것을 위하여 SPEC 2000 정수형 벤치마크를 입력으로 이용하여 그 모델의 정확성을 모의실험으로 측정하였다. 그 결과, 거듭제곱 함수로 유도해낸 모델에 의한 계산 결과와 실제 측정한 값과의 오차는 평균 9.5 %를 기록하였다.

분석적 성능 모델은 기존의 실행 구동 모의실험 또는 트레이스 구동 모의실험보다 시간이 적게 소요되는 장점 이외에, 수퍼스칼라 프로세서 내부에서 발생하는 일에 대하여 하드웨어 성분별로 통찰력을 제공하므로 유익하다.

향 후 연구과제로, 윈도우와 명령어 이슈폭의 관계 이외에 분기 미스나 명령어 및 데이타 캐쉬의 효과를 고려하고, 아울러 제한된 이슈폭과 제한된 연산유닛 및 복수의 지연 싸이클을 갖는 명령어 조건을 추가하여 실제 성능에 근접하도록 성능 모델을 발전시켜 완성하는 것이다.

참고 문헌

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

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

[3] 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.

[4] L. Eeckout, K. D. Bosschere, and H. Neefs, "Performance Analysis through Synthetic Trace Generation," in International Symposium on Performance Analysis of Systems and Software, Apr. 2000.

[5] L. Eckout, 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.

[6] P. K. Dubey, G. B. Adams III, and M. J. Flynn, "Instruction Window Size Trade-Offs and Characterization of Program Parallelism," IEEE *Transactions on Computers, vol. 43, pp 431-442*, Apr. 1994.

[7] D. B. Noonburg and J. P. Shen, "Theoretical Modeling of Superscalar Processor Performance," in Micro-27, Aug. 1994, pp.52-62.

[8] D. B. Noonburg and J. P. Shen, "A Framework for Statistical Modeling of Superscalar Processor Performance," in Proceedings on Third International Symposium on High Performance Computer Architecture, 1997.

[9] E. Riseman and C. Foster, "The Inhibition of Potential Parallelism by Conditional Jumps," IEEE Transactions on Computers, vol. C-21, pp.1405-1411, 1972.

[10] P. Michaud, A. Seznec, and S. Jourdan, "Exploring Instruction Fetch Bandwidth Requirement in Wide Issue Superscalar Processors," in International Symposium on Parallel Architectures and Compilation Techniques, 1999.

[11] P. Michaud, A. Seznec, and S. Jourdan, "An Exploration of Instruction Fetch Requirement in Wide Issue Superscalar Processors," in International Journal of Parallel Programming, 2001, vol. 29.
[12] T. S. Karkhanis and J. E. Smith, "A First-Order Superscalar Processor Model," in Proceedings of the 31st Annual International Symposium on Computer Architecture, 2004.
[13] R. M. Tomasulo, "An Efficient Algorithm for Exploiting Multiple Arithmetic Units," IBM Journal, vol. 11, pp.25-33, Jan. 1967.
[14] T-Y. Yeh and Y. N. Patt, "Two-Level Adaptive Branch Prediction," in The 24th ACM/IEEE International Symposium and Workshop on Microarchitecture, Nov. 1991, pp.51-61.
[15] T-Y. Yeh and Y. N. Patt, "Alternative Implementations of Two-Level Adaptive Branch Prediction," in Proceedings of the 19th International Symposium on Computer Architecture, May. 1992, pp.124-134.
[16] Introduction to Shade, Sun Microsystems, Inc., Jun. 1997.
[17] G.H. Loh, "A Time-stamping Algorithm for Efficient Performance Estimation of Superscalar Processors," in SIGMETIRCS/Performance, 2001, pp. 72-81.