한정된 연산유닛에서 명령어 종속성을 이용하는 수퍼스칼라 프로세서의 이론적 성능 모델

논 문
59-2-33

A Theoretical Superscalar Microprocessor Performance Model with Limited Functional Units Using Instruction Dependencies

이 종 복*
(Jong-Bok Lee)

Abstract - In the initial design phase of superscalar microprocessors, a performance model is necessary. A theoretic performance model is very useful since performance for various architecture parameters can be obtained by simply computing equations, without repeating simulations. Previous studies established theoretic performance models using the relation between the instruction window size and the issue width, with the penalties due to branch mispredictions and cache misses. However, the study was intended for unlimited number of functional units, which is insufficient for the real case application. This paper proposes a superscalar microprocessor theoretical performance model which also works for the limited functional units. To enhance the accuracy of our limited functional unit model, instruction dependency rates are employed. By using trace-driven data of SPEC 2000 integer programs as input, this paper shows that the theoretically computed performance of superscalar microprocessor with limited number of functional units is quite similar to the measured performance.

Key Words : Performance model, Limited functional unit, Instruction dependencies

1. 서 론

마이크로 프로세서 구조의 개발 단계에서 성능을 평가하기 위하여 Simplescalar와 같은 실행 구동 모의실험 (execution-driven simulation) [1] 또는 트레이스 구동 모의실험 (trace-driven simulation)이 주로 이용된다. 이 실험들은 비교적 정확하지만 시뮬레이션을 거친 후 단순히 성능을 얻기 때문에 프로세서 내부에서 발생하는 일에 대한 통찰력을 제공해주지 못하고 공간과 시간이 적지 않게 소요되는 단점이 있다. 이러한 모의실험 위주에 대한 대안으로 프로세서의 이론적 모델을 들 수 있다 [2,3]. 이론적 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.

최근에 개발된 이론적 모델 중에서, 윈도우의 크기와 명령어 이슈폭의 관계를 이용하여 분기 미스 및 캐쉬 효과가 존재하지 않는 이상적인 상태에서의 프로세서의 성능을 도출하고, 여기에 트레이스 구동 모의실험에 의하여 측정한 분기 미스 페널티 및 명령어와 데이터 캐쉬 미스 페널티 데이터를 포함시켜 성능 모델을 구하는 연구가 비교적 높은 정확도를 제공하고 있다 [4]. 그러나, 이 연구에서는 무한한 연산유닛을 대상으로 하였기 때문에, 유한한 연산유닛으로 구동되는 실제 마이크로 프로세서 환경에 적용하기에는 부족하다. 본 논문에서는 이러한 단점을 극복하기 위하여, 한정된 연산유닛을 갖는 수퍼스칼라 프로세서에도 적용할 수 있는 이론적 성능 모델을 제안하였다. 이것을 위하여, 기존의 연구가 명령어를 구분하지 않은 것과 달리 본 논문에서는 명령어를 성분별로 분류하고 각 성분별 명령어의 이슈폭과 윈도우 크기 관계를 이용하여 한정된 연산유닛에 적용할 수 있는 성능 이론 모델을 개발하였다. 또한, 한정된 연산유닛 성능 모델의 정확도를 더욱 높이기 위하여 명령어 간의 종속성에 대한 정보를 추가 적용하였으며, 본 이론적 성능 모델을 이용하여 계산해낸 프로세서의 성능과 모의실험기를 통하여 측정한 성능을 비교하였다.

본 논문의 구성은 다음과 같다. 2 장에서는 한정된 연산유닛 및 종속성을 적용한 수퍼스칼라 마이크로 프로세서의 이론적 성능 모델에 대하여 제안하였으며, 3 장에서는 모의실험 환경, 4 장에서는 모의실험 결과를 나타냈다. 마지막으로 5 장에서는 결론을 도출하였다.

2. 수퍼스칼라 프로세서의 이론적 성능 모델

2.1. 윈도우의 크기와 이슈폭의 관계

수퍼스칼라 프로세서의 윈도우의 크기와 실제로 이슈되는 명령어의 평균 개수에 대한 관계는 수퍼스칼라 프로세서의 이론적 성능모델을 도출해내는데 매우 중요하다. 수퍼스칼라 마이크로 프로세서에서 윈도우의 크기 W와 매 싸이클당 실행되는 명령어 이슈폭 I 사이에는 수식 1과 같은 관계가

* 정 회 원 : 한성대 공대 정보통신공학과 부교수
E-mail : jblee@hansung.ac.kr
접수일자 : 2009년 11월 30일
최종완료 : 2010년 1월 18일

성립한다 [5,6].

$$I = \alpha W^{\beta} \tag{1}$$

이 때, $I = \alpha W^{\beta}$의 그래프의 양변에 로그를 취하고 주어진 데이타 세트 (x_1,y_1), (x_2,y_2),...(x_n,y_n)를 이용하여 α와 β를 각각 최소자승법(Least Square Method)을 이용하여 구할 수 있다. 따라서, 위 관계식을 이용하여 주어진 벤치마크별로 특정한 윈도우 크기에 대하여 명령어 이슈폭을 계산으로 구할 수 있다. 이러한 윈도우-이슈폭 관계는 특정한 하드웨어의 구현과는 상관없이 각 벤치마크 프로그램의 기본적인 레지스터 기반의 데이터 종속에만 의존하는 고유한 특성이다.

2.2 연산유닛을 기반으로 하는 이론적 성능 모델

연산유닛의 개수가 무한하고 캐쉬 효과가 있을 때, 프로세서의 전체 성능 CPI_U는 다음 수식 2와 같이 CPI_{SSU}, CPI_{BPMISS}, CPI_{ICMISS}, CPI_{DCMISS}의 합으로 표현된다.

$$CPI_U = CPI_{SSU} + CPI_{BPMISS} + CPI_{ICMISS} + CPI_{DCMISS} \tag{2}$$

이 식에서 CPI_{SSU}는 프로세서에서 어떤 종류의 미스도 발생하지 않은 이상적인 조건에서의 클럭 사이클 수를 의미하며, IPC_{SSU}의 역수를 취함으로써 구하는데, IPC_{SSU}는 윈도우의 크기 W와 α, β가 주어지면 수식 1로부터 구할 수 있다. 또한 CPI_{BPMISS}, CPI_{ICMISS}, CPI_{DCMISS}는 각각 분기 미스, 명령어 캐쉬, 데이터 캐쉬 미스에 해당하는 페날티로 인하여 추가로 소요된 클럭 사이클을 의미하며, 각각 수식 3으로부터 구할 수 있다.

$$CPI_{BPMISS} = \frac{N_{br} \times P_{br}}{N_{instr}}$$
$$CPI_{DCMISS} = \frac{N_{ic} \times P_{ic}}{N_{instr}} \tag{3}$$
$$CPI_{BPMISS} = \frac{N_{dc} \times P_{dc}}{N_{instr}}$$

수식 4에서 N_{br}, N_{ic}, N_{dc}는 각각 분기 미스 수, 명령어 캐쉬 미스 수, 데이터 캐쉬 미스 수를 나타내며, P_{br}, P_{ic}, P_{dc}는 각각 분기 미스, 명령어 캐쉬 미스, 데이터 캐쉬 미스 페날티 사이클 수를 의미한다. N_{instr}는 각 벤치마크 프로그램별로 수행된 총 명령어의 수를 뜻한다 [4].

본 논문에서는 연산유닛 개수의 제약을 반영하기 위하여 명령어를 ALU 명령어, 로드 명령어, 스토어 명령어, 분기 명령어의 네 가지로 분류하였다. 이 때, 정수형 프로그램만을 대상으로 하기 때문에, 실수형 명령어는 그 대상에서 제외하였다. 연산유닛에 대한 제약이 반영된 성능을 IPC_{SSL}이라 하고, 각 연산 유닛의 성분별 IPC를 IPC_{alu}, IPC_{ld}, IPC_{st}, IPC_{br}이라 할 때 수식 4와 같이 표현 가능하다.

$$IPC_{SSL} = IPC_{alu} + IPC_{ld} + IPC_{st} + IPC_{br} \tag{4}$$

만일 연산유닛의 개수가 충분하다면 각 벤치마크 프로그램마다 각 명령어의 성분별 IPC는 윈도우의 크기와 성분별 명령어 각각의 α, β 값을 이용하여 계산에 의하여 구할 수 있다. 그러나 성분별 연산유닛의 개수가 부족할 때, 각 사이클마다 실제로 실행 가능한 유형별 명령어의 개수는 해당 연산유닛의 개수를 초과하여 실행될 수 없으므로 주어진 연산유닛의 개수로 한정된다. 따라서, 이 성질을 이용하여, 각 성분별 IPC는 두 가지 상황에 따라 다음 수식 5와 같이 모델링할 수 있다. 여기서 $\alpha_{alu}, \beta_{alu}, \alpha_{ld}, \beta_{ld}, \alpha_{st}, \beta_{st}, \alpha_{br}, \beta_{br}$는 명령어 유형별로 최소자승법으로 측정한 명령어의 성분별 파라미터이며, $N_{alu}, N_{ld}, N_{st}, N_{br}$은 각각 실제로 주어진 ALU, 로드, 스토어, 분기 연산유닛의 개수에 해당한다. 성분별 IPC를 계산하여 수식 4로부터 그 합인 IPC_{SSL}을 구한 후에, 그 역수를 취하여 CPI_{SSL}을 계산할 수 있다. 한정된 개수의 연산유닛을 갖는 수퍼스칼라 마이크로 프로세서의 성능인 CPI_L은 수식 6과 같이 나타낼 수 있다.

$$\begin{aligned}
&If\ (\alpha_{alu} W^{\beta_{alu}} \leqq N_{alu})\ IPC_{alu} = \alpha_{alu} W^{\beta_{alu}} \\
&\qquad else\ \ IPC_{alu} = N_{alu} \\
&If\ (\alpha_{ld} W^{\beta_{ld}} \leqq N_{ld})\ IPC_{ld} = \alpha_{ld} W^{\beta_{ld}} \\
&\qquad else\ \ IPC_{ld} = N_{ld} \\
&If\ (\alpha_{st} W^{\beta_{st}} \leqq N_{st})\ IPC_{st} = \alpha_{st} W^{\beta_{st}} \\
&\qquad else\ \ IPC_{st} = N_{st} \\
&If\ (\alpha_{br} W^{\beta_{br}} \leqq N_{br})\ IPC_{br} = \alpha_{br} W^{\beta_{br}} \\
&\qquad else\ \ IPC_{br} = N_{br}
\end{aligned} \tag{5}$$

$$CPI_L = CPI_{SSL} + CPI_{BPMISS} + CPI_{ICMISS} + CPI_{DCMISS} \tag{6}$$

2.3 명령어 간 종속성을 반영하는 이론적 성능 모델

특정한 연산유닛의 제약으로 인하여 실행되지 못한 명령어 A와, 이 명령어에 종속인 명령어 B가 존재한다고 가정한다. 이 때, 명령어 B는 연산유닛이 충분히 있어도 명령어 A 때문에 실행될 수 없다. 실행될 수 없는 명령어 B의 개수는 종속관계에 있는 명령어 A가 실행되는 연산유닛의 부족분에 비례하며, 이러한 명령어 B에 의하여 프로세서의 클럭 사이클이 추가로 늘어나게 된다. 수식 6만으로는 이러한 명령어 종속성과 관련된 성능을 모델링하는데 한계가 있으므로, 본 논문에서는 여러 가지 유형의 명령어 쌍 사이의 종속율을 측정하고, 이것을 반영하여 더욱 정확한 성능 모델을 만들었다. 수식 7은 ALU 명령어가 또 다른 ALU, 로드, 스토어, 분기 명령어에 종속을 부여할 때를 나타낸 것이며, 수

식 8은 로드 명령어가 ALU, 또 다른 로드, 스토어 명령어에 종속을 부여할 때의 상황을 고려한 것이다.

각 연산유닛의 개수가 계산된 성분별 IPC보다 충분한 경우는 이 공식을 사용하지 않으며 CPI는 변화가 없다. 그러나 각 연산유닛의 개수가 계산된 성분별 IPC보다 부족할 경우, 부족분만큼 종속관계에 있는 명령어는 실행할 때 기다려야 하므로 이것을 위하여 추가의 싸이클이 소요된다. 이 때, ALU - ALU 명령어 종속율을 ALU_{ALU}, ALU - 로드 명령어 종속율을 ALU_{LD}, ..., 로드-스토어 명령어 종속율을 LD_{ST} 등으로 나타냈다. 최종적으로, 연산유닛에 의한 제약과, 명령어 간 종속에 의한 효과를 모두 반영한 성능을 CPI_{L-Dep}라고 할 때 수식 9와 같이 나타낼 수 있다.

$$CPI_{ALU-Dep} = CPI_L \times \frac{IPC_{ALU} - N_{ALU}}{IPC_{ALU}} \times (ALU_{ALU} + ALU_{LD} + ALU_{ST} + ALU_{BR}) \tag{7}$$

$$CPI_{LD-Dep} = CPI_L \times \frac{IPC_{LD} - N_{LD}}{IPC_{LD}} \times (LD_{ALU} + LD_{LD} + LD_{ST}) \tag{8}$$

$$CPI_{L-Dep} = CPI_L + CPI_{ALU-Dep} + CPI_{LD-Dep} \tag{9}$$

3. 모의 실험 환경

3.1 수퍼스칼라 마이크로 프로세서

본 논문에서는 명령어 윈도우에서 동적 스케줄링에 의하여 명령어가 이슈되는 수퍼스칼라의 기본형을 이용하였다. 이 때, 윈도우의 크기와 성분별 명령어 이슈폭의 관계를 규명할 때는 무한한 연산유닛을 갖는 이상적인 수퍼스칼라 프로세서를 대상으로 하였고, 그 이외에 성능을 비교하기 위한 용도로는 한정된 연산유닛을 적용하였다. 수퍼스칼라 프로세서는 명령어를 인출부를 통하여 받아들인다. 명령어들은 파이프라인의 단계를 거쳐서 결국 명령어 윈도우에 삽입된다. 명령어 윈도우에 남은 공간이 있는 한, 최대 인출율에 맞추어 명령어를 삽입할 수 있지만, 분기명령어를 만나면 그 싸이클에서는 더 이상의 명령어의 윈도우 삽입을 중단한다. 윈도우 내의 명령어는 비순차 실행 (out-of-order execution) 될 수 있으며, 실제 종속 (true dependency)만이 명령어를 이슈하는데 있어서의 장애요인이다.

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. 이슈된 명령어는 즉시 윈도우에서 삭제되지는 않으며, 선행하는 명령어가 모두 삭제된 이후에야 비로소 삭제된다. 본 논문에서 수퍼스칼라 마이크로 프로세서에 대한 트레이스 구동 모의실험기가 필요하므로, 리눅스 환경에서 C를 이용하여 개발하였다.

3.2 모의실험 하드웨어 사양

표 1은 본 모의실험에 대한 수퍼스칼라 마이크로 프로세서의 하드웨어 사양을 나타낸 것이다. 다양한 명령어 윈도우에 대한 모의실험을 수행하기 위하여 그 크기를 4, 8, 16, 32, 64의 다섯 가지로 정하였다. 또한. 다양한 연산유닛에 대한 결과를 비교하기 위하여 1, 2, 3, 4번의 네 가지 서로 다른 연산유닛 사양을 적용하였다.

표 1 모의실험에 이용된 아키텍쳐 하드웨어 사양
Table 1 Architecture configurations

항목		값
명령어 윈도우의 크기		4,8,16,32, 64
인출율, 이슈율, 퇴거율		4,4,8,8,8
연산유닛 사양	1	ALU(1), load(1), store(1), branch(1)
	2	ALU(2), load(1), store(1), branch(1)
	3	ALU(3), load(2), store(1), branch(1)
	4	ALU(4), load(2), store(1), branch(1)
1 차 명령어 캐쉬		32 KB, 2 차 연관, 16 B 미스 페날티 10 싸이클
1 차 데이터 캐쉬		32 KB, 직접, 32 B 미스 페날티 10 싸이클
분기 어드레스 캐쉬		2 K 엔트리
분기 예측기		14 비트 전역 히스토리 방식 미스 페날티 6 싸이클
결과 지연 싸이클		ALU(1), 분기(1), 로드(1), 스토어(1),

명령어 캐쉬와 데이터 캐쉬는 1 차에 한하여 적용하였으며, 2 차 캐쉬는 적용하지 않았다. 정수형 명령어의 지연 싸이클은 모두 1로 설정하였다. 1 차 명령어 캐쉬의 미스 페날티 싸이클, 1 차 데이터 캐쉬의 미스 페날티 싸이클, 분기 예측 오류 미스 페날티 싸이클은 각각 10, 10, 6이며, 이것은 수식 3에서 각각 P_{ic}, P_{dc}, P_{br}에 해당한다.

3.3 모의실험 입력용 벤치마크

입력 벤치마크 프로그램으로는 *bzip2, crafty, gap, gzip, mcf, parser* 이상 6 개의 SPEC 2000 정수형 프로그램이 사용되었다. 이 프로그램을 Linux 2.6 운영체제하의 Pentium 4에서 Simplescalar의 크로스 컴파일러를 이용하여 MIPS-IV 명령어를 기반으로 하는 실행 가능 화일을 얻었으며, 이 화일을 실행하면서 각 벤치마크마다 MIPS-IV 명령어 트레이스 1억 개를 발생시켰다. 이 때, 1 억 개의 명령어는 그 프로그램의 특성을 가장 잘 나타낼 수 있는 부분을 SimPoint 도구를 이용하여 발췌함으로써, 각 벤치마크 프로그램이 종료할 때까지 모의실험을 하지 않고도 최종 성능과 근접하게 나오도록 하였다 [7].

4. 모의실험 결과

4.1 윈도우 크기와 유형별 명령어의 이슈폭 관계

표 2 각 벤치마크 프로그램의 명령어 유형별 거듭제곱 파라미터

Table 2 Parameters obtained per each benchmark

벤치마크	α				β			
	int	ld	st	br	int	ld	st	br
bzip2	0.23	0.21	0.08	0.06	0.65	0.65	0.65	0.65
crafty	0.43	0.18	0.07	0.10	0.49	0.49	0.49	0.49
gap	0.50	0.22	0.09	0.35	0.28	0.28	0.28	0.28
gzip	0.42	0.18	0.08	0.10	0.51	0.51	0.51	0.51
mcf	0.34	0.37	0.14	0.20	0.41	0.41	0.41	0.41
parser	0.27	0.30	0.11	0.12	0.48	0.48	0.48	0.48

표 2에 각 벤치마크에 대하여 측정한 성분별 명령어의 α와 β값을 나타냈다. α값은 0.06에서 0.5 사이의 값을 가지며, 0.28에서 0.65 사이의 값을 갖는 β 보다 값의 변화가 크다. β 값은 결국 0.5 근처의 값을 나타내므로, 명령어 이슈폭과 윈도우 크기 간에 제곱근 관계에 있음을 본 모의실험에 의한 측정 결과로 재확인할 수 있다. 여기에서 얻은 결과를 2장에서 제안된 수식 5에 대입하여 명령어형 성분별 IPC를 계산하는데 쓰인다.

4.2 명령어 유형별 종속율 및 윈도우의 크기와 성분별 IPC의 관계

그림 1은 각 벤치마크 프로그램에 대하여 명령어 유형 간 종속율을 측정한 결과를 보인다. 본 논문에서는 다른 명령어 쌍 보다 큰 비중을 차지하는 ALU-ALU, ALU-로드, 로드-ALU 명령어 쌍의 결과만 이용하였으며 기타 명령어 쌍은 비중이 낮아서 무시하였다. 이 때 ALU-ALU 명령어 종속율이 가장 크며, 평균 51.5 %를 나타냈다. 그 다음으로 로드-ALU 명령어가 평균 30.2 %를 기록하였고, ALU-로드 명령어는 비교적 낮은 24.4 %를 나타냈다. 여기에서 얻은 결과를 2 장에서 제안한 수식 7과 수식 8에 대입하여 성능 모델에 종속성을 반영시켰다.

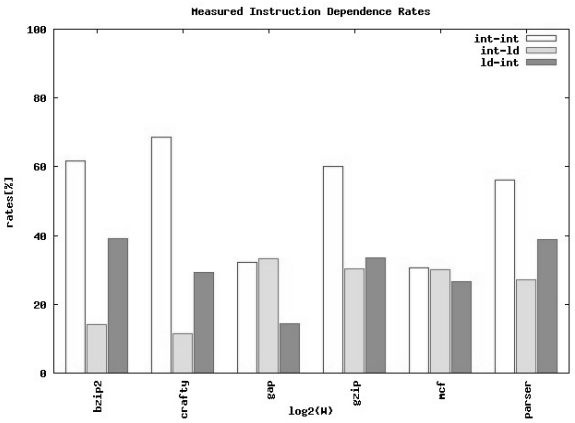

그림 1 명령어 유형간 종속율 (%)

Fig. 1 Instruction dependencies for instruction classes

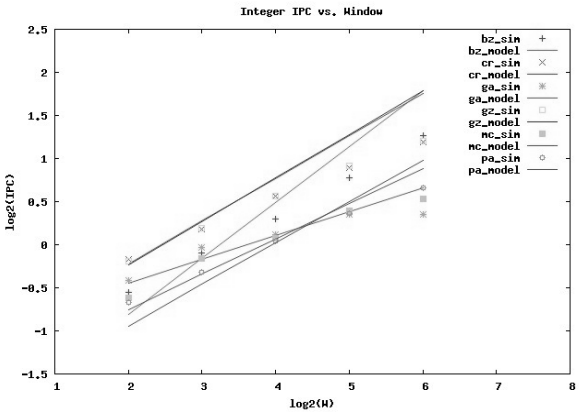

그림 2 윈도우 크기에 대한 산술논리 IPC

Fig. 2 ALU IPCs vs. window sizes

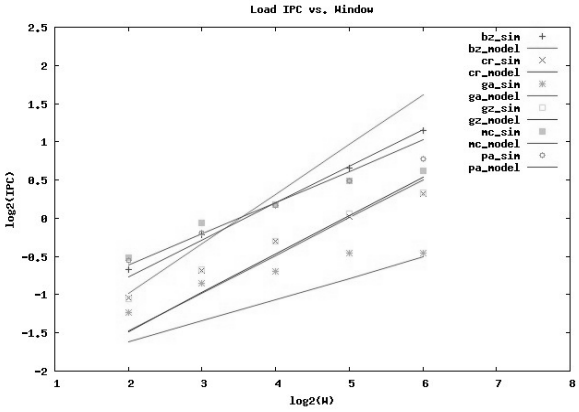

그림 3 윈도우 크기에 대한 로드형 IPC

Fig. 3 Load IPCs vs. window sizes

그림 2와 그림 3은 윈도우 크기 4, 8, 16, 32, 64에 대하여 ALU 명령어, 로드 명령어 각각의 평균 이슈폭을, 측정값과 최소자승법에 의하여 선형 근사시킨 결과를 함께 로그 스케일로 나타낸 것이다. 그래프에서 측정한 성분별 *IPC*는 점으로 나타내고, 근사시킨 결과는 직선으로 나타냈다. ALU 명령어는 parser에서 최저값을, gap에서 최고값을 나타냈으며, 그 값의 범위가 -0.16에서 1.26로 다른 유형의 명령어보다 가장 큰 증가율을 나타냈다. 그 이유는 각 벤치마크 프로그램에서 ALU 명령어의 비중이 가장 크기 때문이다. 로드 명령어의 이슈폭이 그 다음으로 높은 값을 기록하며, 스토어와 분기 명령어는 낮은 값을 기록하였다. 종합적으로 볼 때, 윈도우의 크기가 증가함에 따라, 명령어의 성분별 이슈폭이 로그 스케일로 선형 증가함을 알 수 있다. 윈도우의 크기와

성분별 IPC 쌍의 값들을 최소자승법으로 선형 근사시킨 결과, 5 가지 윈도우 크기에 대하여 평균 4.0 %의 상대오차를 기록하였다.

4.3 수퍼스칼라 마이크로 프로세서의 모의실험과 이론적 성능모델에 의한 계산값 성능 비교

그림 4부터 그림 7은 기존의 트레이스 구동형 모의실험을 실행하여 수퍼스칼라 마이크로 프로세서의 성능을 측정한 결과와, 본 논문에서 제안하는 이론 모델에 의하여 수식으로 계산한 성능을 *IPC*로 나타내어 비교한 것이다. 각 벤치마크 프로그램에 대하여, 각 프로세서의 서로 다른 하드웨어 사양에 대한 성능을 마디로 표현하고 이 마디들을 선분으로 연결하여 그래프로 도시하였다. 이 때, 연산유닛의 사양은 표 1에 기술한 것과 같다.

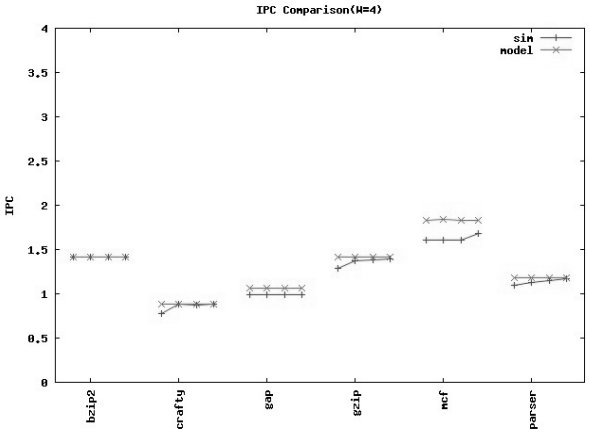

그림 4 윈도우 크기 = 4, 연산유닛이 4 가지인 경우의 성능 비교

Fig. 4 IPC comparison for window size of 4 and 4 different functional units

그림 4는 윈도우의 크기를 4로 고정하고, 연산유닛을 1, 2, 3, 4번의 4 가지 사양으로 변화하면서 모의실험을 시행한 결과이다. 각 연산유닛 사양에 대하여 평균 상대 오차는 각각 8 %, 6 %, 4%, 3 %를 기록하였다.

그림 5에서 윈도우의 크기를 8로 증가하고 동일한 조건일 때, 평균 상대오차는 각각 13 %, 8 %, 3 %, 2 %로 다소 증가하였다. 고정된 윈도우의 크기에서 상대오차의 정확도는 연산유닛이 부족할수록 커지고, 충분할수록 작아지는 것을 관찰할 수 있다. 그 이유는 연산유닛이 부족할수록 연산유닛 한정 모델과 명령어 종속 모델의 적용되는 비중이 커지기 때문이다.

그림 6에서는 연산유닛 사양을 3 번으로 고정하고 윈도우를 4, 8, 16, 32, 64 다섯 가지 크기로 바꿔가면서 모의실행한 결과를 나타냈다. 이 때 평균 상대오차는 4 %, 3, %, 5 %, 10 %, 18 %를 각각 나타냈다.

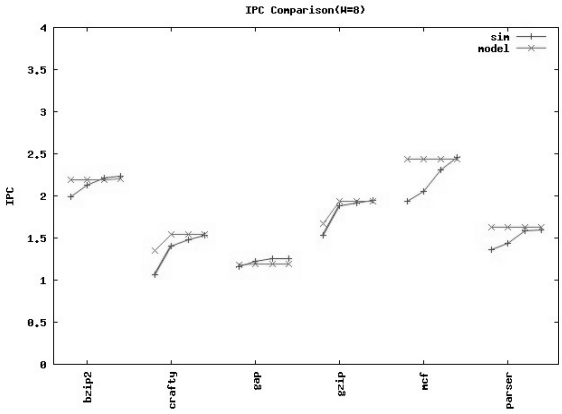

그림 5 윈도우 크기 = 8, 연산유닛이 4 가지인 경우의 성능 비교

Fig. 5 IPC comparison for window size of 8 and 4 different functional units

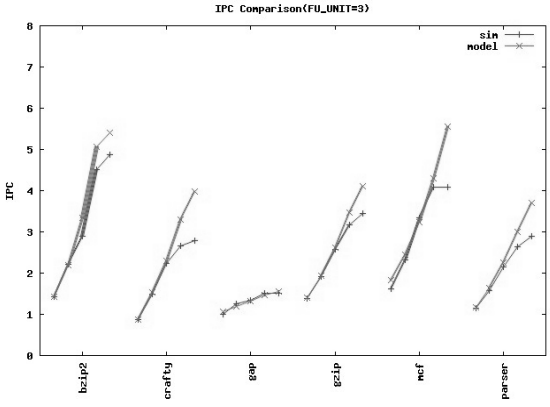

그림 6 연산유닛이 3, 윈도우 크기가 5 가지일 때의 성능 비교

Fig. 6 IPC comparison for functional unit of 3 and 5 different window sizes

마지막으로 그림 7은 연산유닛 사양을 4 번으로 고정하고 동일한 조건에서 모의실험을 시행한 결과를 보인다. 이 때 평균 상대 오차는 각각 3 %, 2 %, 3 %, 5 %, 13 %를 나타냈다. 연산유닛의 사양이 고정되었을 때는 윈도우의 크기가 작을수록 정확도가 증가하고, 커질수록 정확도가 감소하는 것을 확인할 수 있다. 이것은 윈도우의 크기가 커질수록, 윈도우의 크기와 성분별 명령어의 평균 이슈폭을 최소자승법으로 선형 근사시키는 과정에서 오차가 벌어진 결과이다.

종합하면, 본 논문에서 한정된 연산유닛에 적용 가능한 성능 이론 모델을 개발하여 수식으로 계산한 값을 모의실험을 수행한 결과와 비교하여, 마이크로 프로세서 구조 사양에 따라 최저 2 %에서 최고 18 %의 상대오차를 얻을 수 있었다. 또한 한정된 연산유닛 성능 이론 모델에 명령어간 종속성을 반영한 결과가, 명령어간 종속성을 반영하지 않은 성능 이론 모델 결과보다 최저 2 %에서 최고 38 %까지 그 정확도를 높이는데 기여하여, 명령어 종속성을 한정된 연산유닛 성능 이론 모델에 적용하는 것이 유용함을 입증하였다.

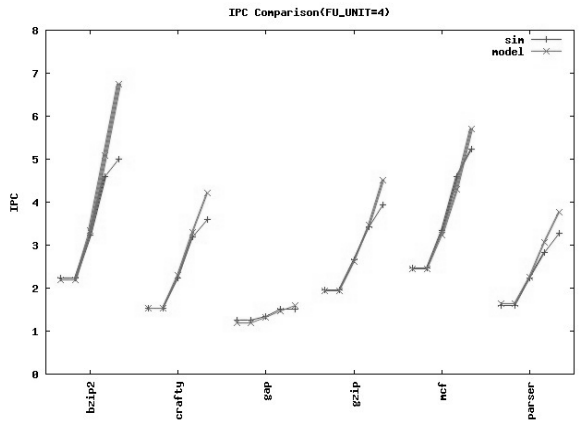

그림 7 연산유닛이 4, 윈도우 크기가 5 가지일 때의 성능 비교

Fig. 7 IPC comparison for functional unit of 4 and 5 different window sizes

5. 결 론

본 논문에서는 윈도우의 크기와 연산 유닛 및 명령어 간의 종속성을 반영한 수퍼스칼라 마이크로 프로세서의 이론적 성능 수식 모델을 제안하였으며, 이것을 이용하여 계산으로 성능을 구한 결과 사양에 따라서 2 %에서 18 %의 범위의 정확도를 나타냈다.

본 모델에 의하여 임의의 크기를 갖는 윈도우 및 연산유닛에 대하여 하드웨어 사양이 바뀔 때마다 모의실험을 반복하지 않고도 계산을 통하여 적은 오차를 갖는 성능을 즉시 구할 수 있으므로 매우 효과적이다. 향 후 연구 방향은 명령어 인출율에 의한 한계 및 복수의 지연 사이클을 갖는 명령어에 대하여 적용하여 그 활용도를 더욱 향상시키는 것이다.

감사의 글

본 연구는 2009년도 한성대학교 교내지원 연구과제임.

참 고 문 헌

[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] 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.

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

[4] 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.

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

[6] 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.

[7] G. Hamerly, E. Perelman, J. Lau, and B. Calder, "SimPoint 3.0 : Faster and More Flexible Program Analysis," in Workshop on Modeling, Benchmarking and Simulation, Jun. 2005.

저 자 소 개

이 종 복 (李 鍾 馥)

1964년 8월 20일생. 1988년 서울대 컴퓨터공학과 졸업. 1998년 동 대학 전기공학부 졸업(공박). 2000년~현재 한성대 정보통신공학과 부교수

Tel : 02-760-4497

Fax : 02-760-4435

E-mail : jblee@hansung.ac.kr