슈퍼스칼라 프로세서의 자원 활용도 분석

김지선[0], 전중남, 김석일
충북대학교 컴퓨터과학과
kjs[0]@john.chungbuk.ac.kr
{joongnam, ksi}@cbucc.chungbuk.ac.kr

Resource Usage Analysis of Superscalar Processor

Jisun Kim[0], Joongnam Jeon and Sukil Kim
Department of Computer Science, Chungbuk National University

요 약

슈퍼스칼라 프로세서 구조에서 명령어 실행을 수행하는 데 사용되는 자원은 그 양에 비해 실제로 활용된 자원의 양은 적다. 본 논문에서는 낮은 자원활용도를 보이는 자원을 활용하는 방안으로 슈퍼스칼라 프로세서를 멀티쓰레드 프로세서로 확장하는데 필요한 기본 데이터를 얻기 위해서 실제로 활용되는 자원의 양을 측정하여 어느 정도의 자원을 활용할 수 있는 지와 자원이 충분히 활용되지 못하는 원인을 분석하였다. 실험을 위해 RA(Resource Analyzer)를 구현하여 SimpleScalar 시뮬레이터에서 제공되는 명령어 파이프라인 트레이스 파일을 분석하여 각 파이프라인 단계에서 처리되는 자원의 활용도를 실험하였다. 자원 활용도가 낮은 원인을 분석하기 위해 프로그램 내에 존재하는 데이터 의존성과 여러 가지 미스 요인들의 비율을 실험을 통해 알아본 결과 IPC(Instruction Per Cycle)는 평균 0.6으로 나타났으며, EX단계의 평균 활용 빈도는 22.9%로 낮아 멀티쓰레드 처리의 필요성이 있음을 확인할 수 있었다.

1. 서 론

마이크로프로세서의 성능을 향상시키기 위하여 프로그램의 병렬성(degree of parallelism)을 향상시키는 방법과 프로세서 연산자원의 활용을 증대시키기 위한 다양한 방법이 연구되고 있다. 그러나 프로그램 내의 명령어들 간의 의존성이나 연산시의 자원 충돌, 명령어 연산 시간의 차이 등과 같은 요인들로 인해 병렬성의 향상에는 한계가 존재할 수밖에 없다. 최근의 실험에 의하면 최대 8개의 명령어를 처리할 수 있는 슈퍼스칼라 프로세서의 경우에 실제 벤치마크에 대한 실험 결과 매 사이클 당 1.5 개의 명령어 이상을 실행하기 힘들다는 사실이 확인된 바 있다[1].

이에 따라 최근에는 한 개의 프로세서에서 하나의 쓰레드를 처리하는 대신 여러 개의 쓰레드를 동시에 처리하도록 하여 하나의 쓰레드를 실행하는 경우에 쉬는 연산 자원이 존재할 경우에 다른 쓰레드를 실행하도록 하여 연산 자원의 활용도를 향상시키는 멀티쓰레드 프로세서에 대한 연구가 활발히 진행중이다[1,2]. 즉 멀티쓰레드 프로세서 구조는 하나의 쓰레드에서 연산 자원을 점유하지 못하는 경우가 발생하였을 때 즉시 다른 쓰레드를 이슈하여 연산 자원이 쉬는 것을 최소한으로 줄여 연산처리기에서 한 사이클에 이슈할 명령어가 아무 것도 없을 때 발생하는 수직적 웨이스트(vertical waste)를 줄이는 구조이다. 따라서 효과적인 멀티쓰레드 프로세서 구조를 연구하기 위해서는 동시에 처리가 가능한 쓰레드의 존재 여부와 여러 개의 쓰레드 중에서 이슈할 쓰레드를 찾아 파이프라인으로 공급하는 방안이 마련되어야 한다. 본 논문에서는 멀티쓰레드 프로세서 구조의 연구에 앞서 여러 가지 벤치마크 프로그램 별로 연산자원이 얼마나 잘 활용되는 가를 확인하고 활용되지 않는 연산 자원을 활용할 수 있는 방안을 연구하였다.

본 논문의 구성은 다음과 같다. 제2절에서는 본 논문에서 다룰 목적으로 하는 슈퍼스칼라 프로세서 구조에 대해 설명하고, 제 3절에서는 SimpleScalar 시뮬레이터를 이용하여 여러 가지 벤치마크에 대한 연산 자원의 활용도를 측정하였다. 마지막으로 제 4절에서는 본 논문의 결론을 맺는다.

2. 슈퍼스칼라 프로세서와 파이프라인 구조

슈퍼스칼라 프로세서는 빠른 연산을 위하여 동시에 여러 개의 명령어를 동시에 실행할 수 있는 명령어 파이프라인으로 구성된다. 또한 명령어 파이프라인은 여러 단계의 스테이지로 구성된다. 예를 들면 MIPS R2000 프로세서의 경우에는 ALU 명령어의 경우에 IA(instruction address generation), IF (instruction fetch), IF, D(instruction decode), EX(execute), …, EX, PA(put away)의 단계를 거쳐 연산이 종료된다[3]. 여기서 EX 단계는 명령어에 따라 몇 개의 사이클을 점유한다.MIPS R4000의 파이프라인은 IF, IF, D/RF(decode/operand fetch), EX, …, EX, C(error check), PA의 단계를 거친다[4]. IBM RS6000 프로세서의 경우에는 IF, DI(dispatch), D/RF, EX, PA의 5단계로 파이프라인을 구성하고 있다[5]. 이렇듯 슈퍼스칼라 프로세서의 파이프라인 구조는 프로세서별로 서로 상이하며 그 스테이지 길이도 서로 다르다. 또한 파이프라인 스테이지의 길이가 길수록 보다 복잡한 명령어를 하나의 파이프라인에서 수행할 수 있으나 파이프라인의 길이가 짧으면 복잡한 명령어는 두 개 또는 세 개의 간단한 명령어를 실행함으로써 수행하도록 한다. 또한 파이프라인 스테이지가 길면 이를 보상하기 위하여 파이프라인 블록을 이중 블록을 사용하여 하나의 CPU 사이클에 두 개의 파이프라인 단계를 진행하도록 하기도 한다[3,4].

본 연구는 한국과학재단 목적기초연구(R05-2002-000-01470-0)지원으로 수행되었음.

본 논문에서 목표로 삼는 슈퍼스칼라 프로세서는 IF, DA, EX, …, EX, WB, CT의 파이프라인 단계를 거치는 것으로 간주하였다. 여기서 DA 단계는 명령어의 디코드를 하고 연산 자원이 가용할 경우에 명령어를 실행 단계로 진입시키는 단계이다. 만일 이 단계에서 오퍼런드(operand)가 준비되지 않았거나 연산 자원이 가용하지 않은 경우에는 명령어를 EX 단계로 진입시키지 않고 다음 사이클에서도 DA 단계를 계속 점유한다. WB 단계는 연산의 결과를 임시 레지스터 버퍼에 기록하는 단계이며 CT 단계는 임시 레지스터 버퍼에 저장된 실행 결과를 레지스터 파일로 복사할 것인지 무시할 것인지를 확인하여 처리하는 단계이다. 예를 들어, 연산 결과가 잘못된 분기 예측에 의하여 무시해야하는 명령어일 경우에는 CT 단계에서 임시 레지스터 버퍼의 정보를 레지스터 파일로 복사하지 않고 제거해버리며, 분기 예측이 올바른 경우에는 임시 레지스터 버퍼의 정보를 레지스터 파일로 복사하면 해당 명령어의 실행이 완전히 끝나게 된다. 따라서 WB과 CT 단계는 MIPS R4000과 IBM RS6000의 PA 단계와 동일한 작업을 수행하나 두 개의 파이프라인 사이클을 필요로 하는 것으로 간주하면 된다. 본 논문에서는 임시 레지스터 버퍼를 16개로 설정하였다.

EX 단계의 파이프라인 사이클 점유는 정수 명령어인 경우에는 1사이클이 필요한 것으로 간주하며, 곱셈과 나눗셈의 경우에는 각각 2사이클과 12 사이클이 필요하다고 간주하였다. 또한 실수연산의 경우에는 2 사이클이 필요하나 곱셈과 나눗셈 연산의 경우에는 각각 4사이클과 12사이클이 필요한 것으로 간주하였다[6].

또한, 본 논문에서 고려하는 목표 프로세서는 L1캐시를 명령어 캐시와 데이터 캐시로 구분하고 있으며 L2캐시에는 명령어와 데이터를 모두 저장할 수 있도록 하였다. 본 논문에서는 L1 캐시의 명령어 캐시와 데이터 캐시를 각각 블록 크기 32Byte로 설정하였으며, 캐시의 크기를 각각 256개로 구성하였다. 즉, 명령어 캐시와 데이터 캐시는 각각 8KB라고 간주하였다. 또한 L2캐시는 블록 크기 64Byte이며 1024개로 구성하여 총 256Kbyte이다. 또한 캐시 정책의 경우에 LRU 방식, 직접 매핑(direct mapping)방식을 사용하는 것으로 간주하였다.

3. 실험 및 분석

3.1 실험 환경

본 논문에서 필요한 슈퍼스칼라 프로세서의 자원 활용도를 측정하기 위해, SimpleScalar/Alpha 2.0 툴셋[6]에서 명령어의 실행 사이클을 분석할 수 있는 sim-outorder를 이용하였다. Sim-outorder 시뮬레이터는 명령어의 비순차 이슈가 가능한 IF, DA, EX, WB, CT의 명령어 처리 파이프라인 구조로 구성된 슈퍼스칼라 프로세서 성능 측정 머신이다.

특별히 본 논문에서는 목표로 하는 슈퍼스칼라 프로세서 구조에 맞추어 표 1과 같이 명령어 이슈 크기를 4개로 설정하였다. 즉, 동시에 인출할 수 있는 명령어의 수, 디코딩 및 스케줄링 명령어의 수를 최대 4로 구성하였다. 또한 연산 처리기는 동시에 4개의 실수 명령어 또는 정수 명령어를 선택적으로 실행할 수 있도록 하였다.

표 1. 프로세서 자원 구성 파라미터

연산처리기 구성		개수
Resources	Fetch unit	4
	Decode unit	4
	Commit unit	4
Functional Units	INT ALU unit	4
	FP ALU unit	4

실험에 사용된 벤치마크 프로그램은 SPEC95 CINT/CFP 벤치마크 프로그램 중에서 표 2와 같이 정수 프로그램 4개와 실수 프로그램 5개를 선택하였다. 표 2에서 입력데이터는 해당 벤치마크를 실행하기 위하여 필요한 입력 프로그램이다.

표 2. 벤치마크 프로그램

정수형 프로그램	입력 데이터	실수형 프로그램	입력 데이터
go	2stone9.in	tomcatv	tomcatv.model
ijpeg	penguin.ppm	su2cor	su2cor.model
perl	scrabbl.in	hydro2d	hydro2d.model
vortex	persons.250	applu	applu.in
		wave5	wave5.in

주어진 벤치마크 프로그램을 sim-outorder 시뮬레이터에서 실행하면 각각의 벤치마크 프로그램의 파이프라인 트레이스를 얻을 수 있다. 실험을 통하여 생성된 파이프라인 트레이스는 벤치마크 프로그램의 첫 번째 명령어의 시작부터 마지막 명령어의 실행에 이르는 과정의 매 사이클마다 파이프라인의 사용 현황을 보여준다. 따라서 본 논문에서 확인하고자 하는 파이프라인 단계의 자원 활용도를 분석하기 위해서 파이프라인 트레이스에서 이를 추출하기 위한 자원 활용 분석기(RA: Resource Analyzer)를 구현하였다. RA는 매 사이클마다 5개의 파이프라인 단계가 사용되는지 여부를 확인하여 총 사이클 중에서 해당 파이프라인 단계가 얼마나 사용되는지를 얻을 수 있었다. 그림 1은 실험의 처리 과정을 보여준다.

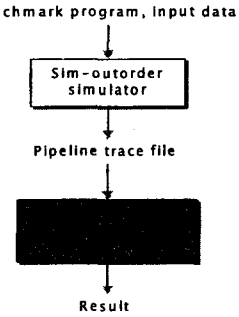

그림 1. 실험 환경

3.2 실험 결과 및 분석

표3은 각 벤치마크 프로그램 별로 프로그램을 실행하였을 때에 총사이클 중에서 실제로 의미 있는 명령어가 실행된 비율(IPC), 명령어 캐시와 데이터 캐시의 미스율과 L2 캐시의 미스율을 측정한 것이다. 평균 IPC는 4개의 명령어를 이슈하는 프로세서 구조에서 0.636으로 매 사이클마다 평균 0.636개의 명령어가 실행됨을 의미한다. 따라서 4개의 명령어가 실행될 수 있음에도 불구하고 실제로는 평균 0.636개가 실행되므로 연산처리기의 이용률이 15.9%밖에 되지 않는 것을 알 수 있다.

표 3에서 분기 미스율은 분기시 분기 예측이 일어날 것이라고 가정했으나 분기 예측이 일어나지 않는 경우를 측정한 것이다. 실험 결과 8KByte의 명령어 캐시에서 명령어 캐시의 미스율은 평균 0.058%이었으며, 데이터 캐시의 평균 미스율이 0.096%이었다. L2 캐시(256KByte)의 평균 미스율이 0.431%이었으며, 분기 예측의 평균 미스율이 7.946%이었다.

표 3. 벤치마크 프로그램의 IPC와 캐시 및 분기 미스율

	IPC	명령어 캐시 미스율(%)	데이터 캐시 미스율(%)	L2 캐시 미스율(%)	분기 미스율(%
go	0.200	0.121	0.203	0.632	0.865
ijpeg	1.058	0.028	0.094	0.440	5.156
perl	0.701	0.051	0.063	0.236	9.108
vortex	0.930	0.088	0.055	0.349	7.505
tomcatv	0.670	0.043	0.091	0.476	7.249
su2cor	0.492	0.072	0.088	0.430	10.362
hydro2d	0.542	0.066	0.092	0.413	10.453
applu	0.551	0.057	0.086	0.436	10.660
wave5	0.590	0.062	0.093	0.411	10.124
평균치	0.636	0.058	0.096	0.431	7.943

RA를 통하여 9개의 벤치마크 프로그램의 파이프라인 트레이스에서 5개의 파이프라인 단계별 자원 활용도를 측정한 결과는 그림 2와 3과 같다. 그림 2는 벤치마크별로 자원활용도를 백분율로 나타낸 것이며, 그림 3은 각 파이프라인 단계별로 평균 활용도를 계산한 결과이다. 그림에서 EX 단계와 CT 단계의 활용도가 다른 파이프라인 단계에 비하여 활용도가 낮은 것을 보여준다. 이는 명령어가 연산처리기를 점유하거나 자료 의존성으로 인하여 명령어가 연산처리기로 진입하지 못함에 따라 계속적으로 DA 단계를 반복해야하며 이로 인하여 IF 단계도 다음 명령어를 인출하지 못함을 보여주는 것이다. 마찬가지로 CT 단계의 활용도가 낮은 것은 실제로 EX 단계에서 명령어가 적기 때문이다. 그러나 WB 단계는 CT 단계에서 처리될 때까지 의미 없는 WB 작업을 계속 반복해야하기 때문이다.

슈퍼스칼라 프로세서를 멀티쓰레드 프로세서로 활용할 수 있는지를 알 수 있는 중요한 요소인 EX 단계와 CT 단계의 활용도를 측정한 결과, 각각 10%에서 32%까지와 6%에서 32%까지 활용되고 있음을 확인할 수 있었으며, 평균적으로 22.9%와 21.9%의 활용도를 보여주고 있음을 알 수 있다. 따라서 4개의 연산처리기를 하나의 쓰레드를 위하여 사용하는 것은 연산처리기의 자원을 낭비하는 요소가 됨을 알 수 있으며, 4개의 연산처리기로 구성된 프로세서는 최대 4개의 쓰레드를 처리하는 것이 바람직함을 보여준다.

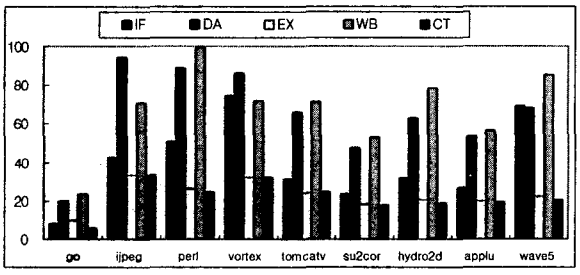

그림 2. 파이프라인 단계별 각 자원 활용도

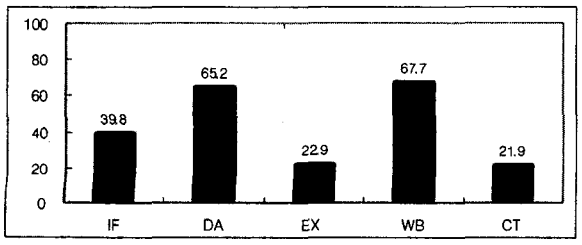

그림 3. 평균 자원 활용도

4.결론

본 논문에서는 슈퍼스칼라 프로세서 구조에서 9 개의 벤치마크 프로그램을 실행하는 경우 연산처리기의 이용률을 측정하였다. 실험 결과 4개의 연산처리기로 구성된 슈퍼스칼라 프로세서에서 벤치마크 프로그램이 실행되는 과정에서 명령어의 실제 실행율은 불과 15.9% 수준으로 매우 낮았다. 또한 연산처리기의 활용도를 측정한 결과는 평균 22.9%로 주어진 자원을 충분히 활용하지 못하고 있음을 알 수 있었다. 그 이유는 주원인은 데이터 의존성과 캐시 및 분기 미스 등의 요인으로 판단된다. 연산 처리기의 활용도를 높이기 위해서는 보다 적은 수의 연산처리기를 사용하는 것이 좋으나 이는 비록 프로그램의 병렬도가 때로는 매우 높으므로 연산 처리기의 수를 줄이는 경우에는 그만큼 총 실행 사이클이 늘어나게 된다. 따라서 연산처리기의 수를 줄이지 않고 활용도를 높이는 방안으로 슈퍼스칼라 프로세서 구조에 멀티쓰레드 처리 기능을 추가하여, 하나의 쓰레드를 처리하면서 여러 가지 이유로 더이상의 쓰레드 처리가 안 되는 경우에 다른 쓰레드를 처리하도록 한다면 연산처리기의 활용도를 높일 수 있을 것으로 보인다. 그러나 이 경우에 쓰레드의 전환에 따른 오버헤드가 필요하며 하드웨어의 지원이 뒤따라야 하는 문제가 대두된다.

앞으로는 쓰레드의 전환을 소프트웨어에 의하여 처리하는 경우와 하드웨어가 지원하는 경우에 대한 실험을 수행하여 슈퍼스칼라 프로세서를 멀티쓰레드 구조를 변환하는데 관한 연구를 계속할 예정이다.

참고문헌

[1].D. M. Tullsen, J. E. Susan and H. M. Levy," Simultaneous Multithreading: Maximizing On-Chip Computer Architecture," Proc., 22nd Annual International Symposium on Computer Architecture, pp. 392-403, 1995

[2].R. Goncalves and P. Navaux, "Improving SMT performance scheduling processes," Proc. 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing, pp. 327-334, 2002

[3].G. Kane, "MIPS R2000 RISC Architecture," Prentice Hall, Englewood Cliffs, NJ, 1987.

[4].S. Mirapuri, M. Woodacre, N. Vasseghi, "The Mips R4000 processor," IEEE Micro, Vol.12, No.2, pp. 10-22, April. 1992

[5].R. R. Oehler and R. D. Groves, "IBM RISC System/6000 Processor Architecture," IBM Journal of Research and Development, vol. 34, pp. 23-36, Jan. 1990.

[6].D. Burger and T. Austin, "The SimpleScalar Tool Set, Version 2.0," Technical Report CS-RT-97-1342, University of Wisconsin Madison, June. 1997