연산처리기 개수에 따른 슈퍼스칼라 프로세서의 성능 분석†

김지선*, 전중남**, 김석일**
*한국정보통신대학교 IT영재교육연구센터
**충북대학교 전기전자컴퓨터공학부
e-mail:jisuny@icu.ac.kr

A Performance Analysis of Superscalar Processor According to the Number of Functional Units

Ji-Sun Kim*, Joong-Nam Jeon**, Suk-Il Kim**
*Education Research Center for the Gifted in IT, ICU
**School of Electrical and Computer Engineering, Chungbuk National University

요 약

슈퍼스칼라 프로세서는 이슈대역폭에 비해 실제로 처리되는 명령어의 개수는 상대적으로 적다. 본 논문에서는 슈퍼스칼라 프로세서의 성능을 높이기 위해, 연산처리기 개수에 따른 슈퍼스칼라 프로세서의 성능을 측정하고, 연산처리기의 활용도를 측정하였다. 실험을 위해 연산처리기 개수는 각각 1개, 2개, 4개로 하였고, 목적 프로세서는 4개의 명령어를 동시에 이슈하고 실행할 수 있는 슈퍼스칼라 프로세서를 대상으로 실험하였다. 또한 연산처리기의 활용도를 분석하기 위해 시뮬레이터를 구현하여 명령어가 실행될 때, 실제 처리된 명령어의 개수를 측정하여 연산처리기의 활용도를 측정하였다. 이를 통해 슈퍼스칼라 프로세서에서 명령어를 실행할 때 필요한 연산처리기의 개수를 결정할 수 있었다. 실험 결과 4-way 슈퍼스칼라 프로세서에서 명령어 실행에 필요한 연산처리기의 개수는 2개가 적당함을 확인할 수 있었다.

1. 서 론

슈퍼스칼라 프로세서는 하나의 프로세서 내에 여러 개의 연산처리기를 두어 서로 독립적으로 실행할 수 있는 명령어들을 동시에 실행하여 작업의 처리 능력을 증대시키는 프로세서이다. 또한 동시에 실행할 수 있는 명령어들을 실시간에 하드웨어에 의해 찾아내기 때문에 모든 프로그램을 대상으로 실행이 가능한 구조이다[1]. 그러나 명령어 스케줄링에 따르는 데이터 및 제어 의존성 검사를 하드웨어에 의해 수행해야 하기 때문에 하드웨어 복잡도가 높고, 명령어 수준 병렬성을 추출할 수 있는 범위가 제한되어 있다. 그리고 한 사이클에서 연산처리기의 이슈 슬롯에 어떤 명령어도 이슈되지 않는 경우가 발생할 수 있다.

이에 따라 슈퍼스칼라 프로세서의 연산처리기는 어느 순간에는 수 사이클동안 아무런 동작도 하지 않는 상태에 빠질 수 있으며, 이로 인해 슈퍼스칼라 프로세서의 성능이 크게 저하 된다[2].

최근에는 이러한 슈퍼스칼라 프로세서의 쉬고 있는 연산처리기를 활용하기 위해 슈퍼스칼라 프로세서 구조에 멀티쓰레딩 기법을 추가하여 프로세서의 성능을 향상시키고 있다. 멀티쓰레딩 기법은 한 쓰레드에서 여러 가지 이유로 인해 더 이상 명령어를 실행할 수 없는 경우, 다른 쓰레드로 스위칭 하여 연산처리기의 활용도를 높일 수 있는 기법이다[3].

본 논문에서는 슈퍼스칼라 프로세서의 성능을 높이고 쉬는 연산처리기를 최소화하기 위해, 연산처리기의 개수를 변경하면서 슈퍼스칼라 프로세서의 명령어 실행에 대한 성능을 측정하고 분석하였다. 이 실험을 통해 슈퍼스칼라 프로세서에서 명령어를 실행할 때 요구되는 연산처리기의 개수를 예측할 수 있었다.

본 논문의 구성은 다음과 같다. 제 2절에서는 슈퍼스칼라의 성능을 저하시키는 요인들을 설명하고, 제 3절에서는 본 논문에서 목적으로 하는 슈퍼스칼라 프로세서 구조에 대해 설명하였다. 제 4절에서는 시뮬레이터를 이용하여 연산처리기 개수에 따른 슈퍼스칼라 프로세서의 성능을 측정하고 분석하였다. 마지막으로 제 5절에서는 본 논문의 결론을 맺는다.

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

2. 슈퍼스칼라 프로세서와 성능 저하

슈퍼스칼라 프로세서에서 두 개 이상의 명령어들이 같은 자원을 동시에 사용하고자 할 때, 자원의 부족으로 인해 경쟁하는 경우가 종종 발생한다. 이러한 현상을 자원충돌이라고 하며, 여기서 자원은 기억장치, 버스, 캐시, 레지스터 파일 포트나 연산처리기 등을 의미한다. 자원 충돌은 메모리 지연이나, 실행 시간의 지연 등의 요인으로 발생하여 전체 프로세서의 성능을 저하시키게 된다. 이러한 자원 충돌과 같은 문제는 경쟁이 발생하는 자원을 여러 개 사용하면 간단하게 해결될 수 있다. 예를 들어, 무한대의 자원을 추가할 수 있다고 가정하면 어떠한 자원 충돌도 발생할 가능성은 없다. 그러나 현실적으로 자원을 무한대로 추가하는 것은 불가능하며, 비용적인 측면을 고려해야하기 때문에 연산처리기의 경우 이슈 대역폭에 비례하는 수의 자원을 추가하게 된다. 그러나 이슈 대역폭 크기만큼의 명령어들을 이슈하는 것을 기대하기는 매우 어렵다. 예를 들어 캐시 미스등과 같은 이유로 인해 이슈 슬롯에 아무런 명령어들이 이슈되지 못하는 경우, 수직적 웨이스트나 수평적 웨이스트가 발생하여 연산처리기가 아무런 동작을 하지 않고 사이클을 점유하기 때문이다[4].

슈퍼스칼라 프로세서의 성능이 저하되는 요인으로 파이프라인 구조에서 발생하는 경우가 있다. 파이프라인의 단계의 수가 많다고 해서 항상 그 단계만큼의 명령어를 처리 할 수 있는 것은 아니다[5]. 그 이유는 한 사이클 당 이슈하는 명령어의 수가 증가하면, 일반적으로 제어 의존성이 발생할 빈도도 이슈율에 비례하여 증가하기 때문에 명령어들 사이의 자료 및 제어 의존성이 해결된 명령어들을 찾아내는 작업이 힘들어지기 때문이다. 또한 여러 개의 명령어들이 같은 연산처리기를 사용하려고 하는 자원 충돌 발생과 명령어의 종류에 따른 연산처리기의 점유 시간 차이와 같은 요인으로 인해, 순차적으로 명령어들이 이슈되지 않아 사이클 지연이 발생하기 때문이다. 또 다른 성능저하 요인으로는 데이터 및 제어 의존성, 로드 명령어의 지연, 분기 명령어의 지연과 같은 소프트웨어 요인이 있으며, 해결책으로 분기예측과 레지스터 재명명과 같은 기법을 사용한다. 그러나 RAW(Read After Write)의존성인 경우는 의존성이 있는 명령어의 실행이 완료되고, 결과가 해당 레지스터에 저장될 때까지 기다려야 하기 때문에 순차적인 명령어의 실행이 어려워지고, 이슈율이 낮아지게 된다.

이러한 이유로 인해 슈퍼스칼라 프로세서의 성능 향상에는 한계가 있다. 그러므로 이슈율이 높다 하더라도 실제적으로 사이클 당 처리되는 명령어의 수는 적으며, 이것은 슈퍼스칼라 프로세서에서 명령어 처리에 사용되고 있는 연산처리기들이 충분히 활용되고 있지 않음을 보여주고 있는 것이다. 따라서 본 논문에서는 연산처리기의 활용도를 높이고 쉬고 있는 연산처리기를 최소화하기 위해, 사이클 당 최대로 처리할 수 있는 연산처리기의 능력에 대한 실제로 처리한 명령어의 수를 연산처리기 개수에 따라 측정하고 분석하였다.

3. 목적프로세서 구조

본 논문에서 연산처리기의 개수에 따른 슈퍼스칼라 프로세서의 성능을 측정하기 위해 설계한 목적 프로세서는 그림 1과 같다. 목적 프로세서는 하드웨어에 의한 동적인 명령어 스케쥴링과 비순차 이슈를 하는 슈퍼스칼라 프로세서이다. 목적 프로세서의 캐시는 L1(Level 1)과 L2(Level 2) 캐시가 있고, L1 캐시를 명령어 캐시와 데이터 캐시로 구분하였다. L2 캐시에는 명령어와 데이터를 모두 저장할 수 있도록 하여, L1 캐시에서 캐시 미스가 발생하면 L2 캐시를 액세스할 수 있도록 하였다. 캐시 구성은 L1 캐시는 명령어 캐시와 데이터 캐시를 각각 블록 크기 32Byte로 설정하였으며, 캐시의 크기를 각각 256개로 구성하였다. 즉, 명령어 캐시와 데이터 캐시는 각각 8KB라고 간주하였다. 또한 L2 캐시는 블록 크기 64Byte이며 1024개로 구성하여 총 256Kbyte이다. 또한 캐시 정책의 경우에 LRU 방식, 직접 매핑(direct mapping)방식을 사용하는 것으로 간주하였다.

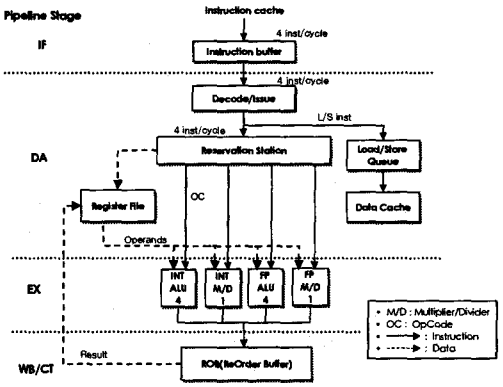

그림 1. 목적 프로세서 구조

목적 프로세서의 파이프라인 구조는 5개의 명령어 파이프라인 단계를 가지고 있으며, 각 단계는 IF(명령어 인출), DA(명령어 해독), EX(명령어 실행), ... , WB(결과저장), CT(실행종료)로 이루어져 있다. 연산처리기는 정수형 가산기/곱셈·나눗셈기, 실수형 가산기/곱셈·나눗셈기로 구성되어 있으며, 실험을 위해 연산처리기의 개수를 변경할 수 있도록 하였다. 각 명령어의 사이클 점유 시간은 정수형 명령어의 경우 덧셈과 뺄셈 명령어는 각각 1사이클이 소요되며, 곱셈 명령어와 나눗셈 명령어는 각각 3사이클과 12사이클을 점유한다. 실수형 명령어에 대해서는 덧셈과 뺄셈 명령어는 각각 2사이클, 곱셈과 나눗셈 명령어는 각각 4사이클과 12사이클이 소요된다. 그림 1에서 실선은 명령어의 흐름을, 점선은 데이터의 흐름을 나타낸다.

목적 프로세서의 실행 과정은 다음과 같다. 매 사이클마다 명령어 캐시에서 명령어를 이슈하고, 디코드한 다음 Reservation Station 에서 실행 중인 명령어와 실행이 준비된 명령어들간의 의존성 검사를 하드웨어에 의해 동적으로 수행된다. 의존성 검사 후 실행 가능한 명령어는 명령어의

종류에 맞게 각 연산처리기에서 명령어의 순서에 관계없이 이슈되어 실행된다. 실행이 끝나면, 비순차적으로 이슈된 명령어의 순서를 재조정하기 위해 Reorder Buffer에 저장되고, 정확한 프로그램의 순서에 맞게 결과를 레지스터에 저장함으로써 명령어의 실행을 종료한다.

4. 실험 및 분석

4.1 실험 환경

목적 프로세서의 성능을 측정하기 위해, simplescalar tool set/Alpha 2.0[6]에서 명령어의 실행 사이클을 분석할 수 있는 sim-outorder를 이용하였다. sim-outorder 시뮬레이터는 명령어의 비순차 이슈가 가능한 IF, DA, EX, WB, CT 명령어 처리 파이프라인 구조로 구성된 슈퍼스칼라 프로세서 성능 측정 머신이다. 연산처리기의 개수에 따른 자원구성은 표 1과 같고, 연산처리기를 각각 1개, 2개, 4개로 변경해가며 실험하였다. 또한 명령어의 의존성과 실행 순서를 담당하는 Reorder buffer와 Reservation Station의 엔트리 개수는 16개로 하였으며, Load/Store 명령어의 실행 순서를 담당하는 Load/Store Queue의 엔트리 개수는 8 개로 하였다.

표 1. 자원구성도

Resources	Size	Description
Fetch	4	Fetch bandwidth
Decode	4	Decode bandwidth
Issue	4	Issue bandwidth
WB	4	WriteBack bandwidth
Int FUs	1 2 4	Number of Int FUs
FP FUs	1 2 4	Number of FP FUs
Reservation Station	16	Number of entry
Reorder Buffer	16	Number of entry
Load/Store Queue	8	Number of entry

표 1과 같은 자원 구성에 따라 본 논문에서 실험에 사용된 벤치마크 프로그램은 SPEC95 벤치마크 프로그램 중에서 표 2와 같이 프로그램 4개를 선택하였다. 입력데이터는 해당 벤치마크가 필요로 하는 데이터이다.

표 2. 벤치마크프로그램

벤치마크	입력데이터
ijpeg	penguin.ppm
perl	scrabbl.in
vortex	persons.250
su2cor	su2cor.in

그림 2는 본 논문의 실험 환경이다. 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. 그림 2의 실험 환경에서 파이프라인 트레이스를 살펴보면 '@'는 현재 사이클을 의미하며, 다음 라인은 현재 사이클에서 명령어 캐시로부터 인출된 명령어들을 나타낸다. 그 다음 라인은 다섯 단계로 이루어진 파이프라인에서 처리되고 있는 명령어이다.

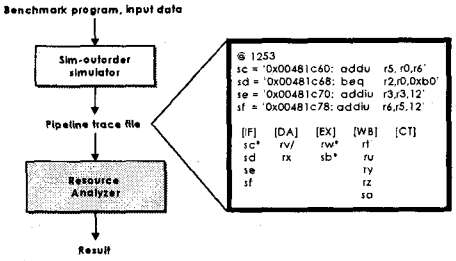

그림 2. 실험 환경

본 논문에서는 연산처리기가 해당 벤치마크 프로그램의 명령어 실행에 대한 결과를 분석하기 위해 시뮬레이터(Resource Analyzer)를 구현하였다. 이 시뮬레이터는 그림 2에서 보는 바와 같이 명령어 파이프라인 트레이스 파일을 입력으로 받아 사이클마다 연산처리기에서 실행한 명령어의 개수를 연산처리기의 개수에 따라 측정한 결과를 보여준다.

4.2 실험 결과 및 분석

본 논문에서는 프로세서의 성능을 측정하기 위해, 사이클당 실행한 명령어의 수인 IPC(Instruction Per Cycle)와 연산처리기의 명령어 실행에 대한 연산처리기 활용도를 측정하였다. 그림 3은 연산처리기 개수에 따른 IPC의 변화를 보여주고 있다. 연산처리기가 1개일 때는 0.4~0.7개를 사이클당 처리하고, 2개일 때는 0.5~0.9개, 4개일 때는 0.5~1.0개를 처리하고 있음을 알 수 있었다. 벤치마크 프로그램의 IPC 측정 결과 4-way 슈퍼스칼라 프로세서에서 사이클 당 실행된 명령어는 이슈율에 비례하지 않음을 알 수 있었다.

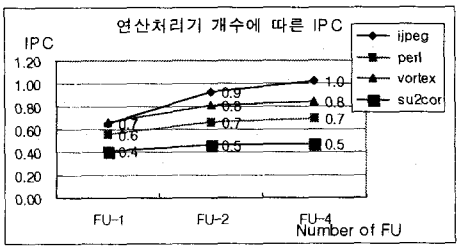

그림 3. 연산처리기 개수에 따른 IPC

연산처리기 개수가 1개일 때 IPC가 낮은 이유는 연산처리기의 부족으로 인한 명령어의 실행이 순차적으로 이루어

지지 않기 때문임을 연산처리기의 활용도 실험 결과 확인할 수 있었다. 연산처리기가 2개일 때는 실행할 명령어에 따른 연산처리기가 적당하여 IPC도 증가한 것이다.

그림 4는 연산처리기의 활용도를 측정한 것으로 연산처리기가 각각 1개, 2개, 4일 때를 측정하였다. 연산처리기가 1개일 때는 명령어가 실행할 수 있음에도 불구하고, 연산처리기의 개수가 1개로 한정되어 있어 연산처리기의 활용도는 높다. 그러나 명령어마다 사이클 점유 시간이 있어 IPC에서 확인된 바와 같이 실제로 처리한 명령어 수는 적음을 알 수 있었다. 연산처리기가 2개일 때는 4개의 벤치마크 프로그램에 대해 33.9%~57.3%의 활용 범위를 보이고 있었다. 연산처리기의 개수가 4일 때의 활용도는 17.6%~31.9%를 보이고 있었다. 이것은 연산처리기가 4개일 경우에는 명령어의 의존성 문제와 같은 이유로 인해 이슈율이 낮아 연산처리기의 활용도가 낮고, 쉬고 있는 연산처리기가 많기 때문이다.

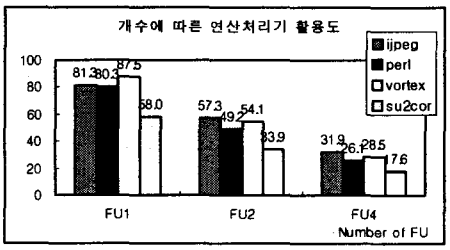

그림 4. 연산처리기의 활용도

IPC 측정 결과와 연산처리기의 활용도를 분석한 결과, IPC가 연산처리기 2개일 때와 4개일 때 커다란 차이가 없으므로, 4개의 명령어가 실행하는 슈퍼스칼라 프로세서에서는 2개의 연산처리기가 적당함을 실험 결과 확인할 수 있었다.

5. 결론

본 논문에서는 슈퍼스칼라 프로세서의 성능을 높이고, 쉬고 있는 연산처리기의 개수를 최소화하기 위해 연산처리기 개수에 따른 슈퍼스칼라 프로세서의 성능을 측정하였다. 4-way 이슈 슈퍼스칼라 프로세서인 경우, 사이클 당 처리하는 명령어는 1.2 미만으로 실행 가능한 명령의 수보다 현저하게 적음을 실험 결과 알 수 있었다. 이러한 원인은 프로그램이 복잡해짐에 따라, 명령어들 간의 의존성이 커지게 되어 매 사이클 당 실행 가능한 명령어들을 찾기가 어렵기 때문이다. 본 논문에서 측정한 연산처리기 활용도 실험 결과, 연산처리기의 자원활용도는 연산처리기가 1개일 때는 실행할 명령어에 비해 연산처리기의 부족으로 명령어들이 순차적으로 실행할 수 없어 상대적으로 IPC가 0.4~0.7로 낮았다. 연산처리기가 2개일 때와 4개일 때는 IPC는 0.5~1.0의 분포를 나타내었고, 연산처리기의 활용도에서 큰 차이를 보이고 있었다. 즉 연산처리기가 2개일 때는 33.9%~57.3%의 활용범위를 보이고 4개일 때는17.6%~31.9%의 활용 범위를 보이고 있었다. 이것은 4개의 명령어를 이슈할 수 있는 프로세서의 경우 연산처리기 2개를 사용하는 것이 슈퍼스칼라 프로세서의 연산처리기의 활용도와 프로세서의 성능을 높일 수 있음을 확인할 수 있었다. 또한 논문의 실험 결과를 통해, 명령어 이슈 대역폭에 따라 연산처리기 개수를 결정하는 것은 연산처리기의 활용도를 낮추고, 프로세서의 성능을 저하시키는 원인임을 알 수 있었다.

참고 문헌

[1] J. E. Smith, G. S. Sohi, "The Microarchitecture of Superscalar Processors," *IEEE Proceedings*, Vol. 83, No. 12, Dec. 1995, pp. 1609-1624.

[2] S. J. Eggers, J. S. Emer, H. M. Leby and et al, "Simultaneous Multithreading: A Platform for Next-Generation Processors," *IEEE Micro*, Vol. 17, No. 5, Sep.-Oct. 1997, pp. 12-19.

[3] G. T. Byrd and M. A. Holliday, "Multithreaded Processor Architectures," *IEEE Spectrum*, Vol. 32, No. 8, Aug. 1995, pp. 38-46.

[4] D. Sima, T. Fountain and P. Kacsuk, *Advanced Computer Architectures*, Addison-Wesley.

[5] A. Bashteen, I. Lui and J. Mullan, "A Superpipeline Approach to the MIPS Architecture," *Compcon Spring '91 Digest Papers*, 1991, pp. 8-12.

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