http://dx.doi.org/10.7236/JIWIT.2012.12.5.123
JIWIT 2012-5-16

멀티코어 순차 수퍼스칼라 프로세서의 성능 연구

Performance Study of Multi-core In-Order Superscalar Processor Architecture

이종복*

Jongbok Lee

요 약 최근에 이르러 디지털 시스템의 성능을 극대화하기 위하여, 멀티코어 프로세서가 상용화 되어 널리 이용되고 있다. 이러한 멀티코어 프로세서를 구성하는 단위 코어의 성능을 높이면, 적은 개수의 코어를 가지고 시스템의 성능을 크게 향상시킬 수가 있다. 본 논문에서는 순차실행 방식의 수퍼스칼라를 단위 코어로 하는 멀티코어 프로세서 아키텍처를 제안하였다. 그리고, 윈도우 크기가 4에서 16이고 2-코어에서 16-코어로 구성되는 멀티코어 수퍼스칼라 프로세서에 대하여, SPEC 2000 벤치마크를 입력으로 하는 광범위한 모의실험을 수행하였다. 모의실험 결과, 윈도우의 크기가 16일 때 16-코어 수퍼스칼라 프로세서는 1-코어 수퍼스칼라 프로세서보다 8.4배의 성능 향상을 가져왔다. 또한, 같은 코어 개수를 가진 멀티 코어 수퍼스칼라 프로세서의 성능이 멀티코어 RISC 프로세서의 성능의 2 배를 기록하였다.

Abstract In order to overcome the hardware complexity and performance limit problems, recently the multi-core architecture has been prevalent. For hardware simplicity, usually RISC processor is adopted as the unit core processor. However, if the performance of unit core processor is enhanced, the overall performance of the multi-core processor architecture can be further enhanced. In this paper, in-order superscalar processor is utilized as the core for the multi-core processor architecture. Using SPEC 2000 benchmarks as input, the trace-driven simulation has been performed for the number of superscalar cores between 2 and 16 and the window size of 4 to 16 extensively. As a result, the 16-core superscalar processor for the window size of 16 results in 8.4 times speed up over the single core superscalar processor. When compared with the same number of cores, the multi-core superscalar processor performance doubles that of the multi-core RISC processor.

Key Words : multi-core processor, superscalar processor, in-order execution

Ⅰ. 서 론

최근 30년간 마이크로 프로세서 연구는 싱글코어 프로세서를 더욱 빠르게 실행하도록 설계하는 것을 목표로 하였으며, 수퍼스칼라 프로세서가 그 대표적인 예이다 [1]. 그러나, 이러한 수퍼스칼라 프로세서는 하드웨어의 복잡도에 비하여 성능의 향상이 미미하여 한계에 봉착하였다.

*정회원, 한성대학교 정보통신공학과
접수일자 : 2012년 7월 5일, 수정완료 : 2012년 9월 22일
게재확정일자 : 2012년 10월 12일

Received: 5 July 2012 / Revised: 22 September 2012 / Accepted: 12 October 2012
**Corresponding Author: jblee@hansung.ac..kr
Dept. of Information & Communications Engineering, Hansung University, Korea

이것을 해결하기 위하여 멀티코어 프로세서가 대두되었다 [2-7]. 멀티코어 프로세서는 응용 프로그램을 병렬화할 수 있다는 것을 전제로 할 때, 프로세서의 성능을 높이는 돌파구이다. 이러한 멀티코어 아키텍처의 단위 코어 프로세서로는 한 싸이클에 최대 한 개의 명령어만을 처리할 수 있는 RISC형태의 간단한 프로세서가 주로 쓰이고 있다. 그러나, 코어의 개수가 32 개 미만인 멀티코어 프로세서 시스템에서 그 성능의 극대화를 위하여 RISC보다 훨씬 성능이 뛰어난 수퍼스칼라 프로세서를 단위 코어로 활용한다면, 전체 시스템의 성능이 더욱 향상될 수 있다. 또한, 순차실행 (in-order) 방식의 수퍼스칼라 프로세서는 비순차실행 (out-of-order) 방식보다 하드웨어 복잡도가 낮아서 멀티코어 프로세서로 구성하기에 용이하다.

본 논문에서는 윈도우의 크기가 4에서 16의 범위인 순차실행 방식의 수퍼스칼라 멀티코어 프로세서의 구조를 제안하였으며, SPEC 2000 벤치마크에 대하여 2-코어에서 16-코어까지의 성능을 측정하고 분석하였다. 본 논문은 다음과 같이 구성된다. 2 장에서 멀티코어 수퍼스칼라 프로세서의 구조 및 모의실험기에 대하여 살펴보고, 3 장에서 모의실험 환경에 대하여 고찰한다. 4 장에서 모의실험 결과를 보이며, 5 장에서 결론을 맺는다.

II. 멀티코어 수퍼스칼라 프로세서의 구조 및 모의실험기

1. 멀티코어 수퍼스칼라 프로세서의 구조

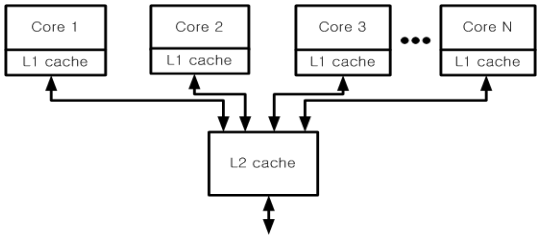

그림. 1. 멀티코어 수퍼스칼라 프로세서의 구조
Fig 1. The multi-core superscalar processor architecture

그림 1은 N 개의 코어로 구성되는 멀티코어 수퍼스칼라 프로세서 구조를 나타낸 것이다. 각 코어는 자체적으로 1 차 명령어 및 1 차 데이터 캐쉬를 가지며, 메인 메모리와 연결되는 공통의 2 차 캐쉬를 공유한다. 각 코어에 설치된 1 차 데이터 캐쉬의 일관성(cache-coherency)을 위하여 MESI 프로토콜을 이용한다.

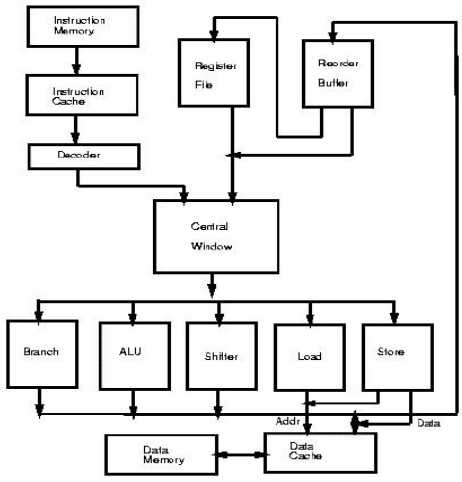

그림 2. 수퍼스칼라 코어 프로세서의 구조
Fig 2. The superscalar core processor.

본 논문에서는 그림 2와 같이 명령어 윈도우에서 동적 스케줄링에 의하여 명령어가 이슈되는 수퍼스칼라의 기본형을 이용하였다. 수퍼스칼라 프로세서는 매 싸이클마다 2 개 이상의 명령어를 인출부를 통하여 받아들인다. 명령어들은 파이프라인의 단계를 거쳐서 결국 명령어 윈도우에 삽입된다. 명령어 윈도우의 크기에 따라 최대 인출율에 맞추어 명령어를 삽입할 수 있지만, 분기명령어를 만나거나 캐쉬 미스가 발생하면 그 싸이클에서는 더 이상의 명령어의 윈도우 삽입을 중단하고, 윈도우 내의 명령어가 모두 소진된 후에야 명령어 인출이 다시 재개된다. 윈도우 내의 명령어는 재명명 (rename)되어 실제 종속 (true dependency)이 없을 경우 순차실행된다. 분기 예측을 위하여 비교적 높은 정확도를 나타내는 2 단계 적응형 분기 예측법을 이용하였다 [8].

2. 멀티코어 수퍼스칼라 프로세서 모의실험기

멀티코어 수퍼스칼라 프로세서는 제 1 단계 명령어 자취의 발생, 제 2 단계 명령어 자취에 대한 멀티코어 프로세서의 실행으로 나누어진다. 제 1 단계에서 명령어 자취는 쓰레드 단위의 병렬성을 코어에 대응시킬 수 있도록 발생되었다. 제 2 단계 멀티코어 수퍼스칼라 프로세서의

실행 과정을 명령어 인출, 명령어 재명명, 명령어 이슈 및 멀티코어 시뮬레이션으로 나누어 자세하게 기술하면 다음과 같다.

2.1 명령어 인출, 재명명 및 이슈

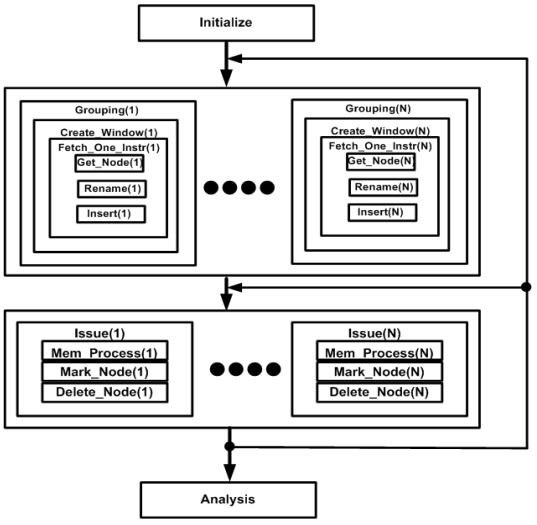

그림 3. 멀티코어 프로세서 모의실험기
Fig 3. The multi-core processor architecture simulator.

본 모의실험기는 그림 3에 나타낸 것과 같이 Initialize 블럭에서 초기화 작업을 거친 후에, Grouping 블럭이 Create_Window 블럭을 부르고 이것은 다시 Fetch_One_Instr 블럭을 불러서 각 코어는 매 싸이클마다 새로운 명령어를 인출받는다. 한 싸이클에 2 개 이상의 명령어를 인출받을 때, 분기명령어를 만나거나 캐쉬 미스가 발생하면 해당 싸이클에서는 그 이상의 인출을 하지 않고 인출을 멈춘다. Get_Node 블럭에서 인출한 명령어는 Rename 블럭에서 재명명 작업을 거치면서 명령어 종속에 의한 타임스탬프(timestamp) 값을 설정받는다.

타임스탬프 방식은 명령어 자취를 이용하는 모의실험에서 데이터 종속성을 신속하고 효율적으로 부여할 수 있는 핵심적인 방법이다. 레지스터 화일의 타임스탬프 값에 의하여, 코어 내부 및 코어와 코어 간 명령어 간의 종속성을 이용하여 명령어 병렬성을 구하는데 반영된다. 재명명을 거친 명령어는 insert 블럭에서 각 코어의 명령어 윈도우에 삽입된다.

Issue 블럭으로 진행하면, 싸이클이 증가함에 따라서 윈도우 내의 명령어는 해당하는 연산유닛을 활용할 수 있고 자체의 타임스탬프 값이 현재 싸이클 보다 작거나 같은 두 가지 조건이 만족될 때 한하여 삭제될 수 있다.

2.2 멀티코어 시뮬레이션

모든 N개의 멀티코어에 대하여 해당 코어의 윈도우 공간에 Grouping 블럭을 이용하여 명령어를 인출해서 채우고, 역시 N개의 멀티코어에 대하여 Issue 블럭으로 각 코어에 대하여 명령어를 실행하면서 종속성에 의하여 부여된 명령어의 타임스탬프가 충족되면 삭제한다. 위의 Issue 동작은 명령어를 인출한 후에 모든 코어에서 명령어가 삭제되어 전체 코어의 윈도우가 빈 상태가 될 때까지 반복적으로 실행된다. 전체 코어가 빈 상태가 되면, 다시 Grouping 블럭을 통하여 각 코어를 명령어로 채운다.

위 과정이 한번 실행될 때 마다 싸이클이 증가하므로, 매 싸이클 마다 명령어 실행 및 삭제가 가장 오래 걸리는 코어가 해당 싸이클 수를 결정한다. 이 과정은 입력으로 주어진 벤치마크 프로그램의 모든 명령어가 소진될 때까지 반복된다. 이 때, 모의실험에 입력으로 쓰인 명령어의 총 개수를 처리하기 위하여 소요된 총 싸이클 수로 나누면, 멀티코어 프로세서 시스템의 IPC(Instruction Per Cycle)을 계산할 수 있다.

Ⅲ. 모의실험 환경

표 1은 모의실험에 이용된 SPEC 2000 정수형 벤치마크 프로그램이다. SimpleScalar를 통하여 MIPS IV 10억 개의 명령어 자취를 임의의 발생시켜서 쓰레드 단위 병렬성을 코어에 대응시켜 모의실험기에 입력하였다 [9].

표 1. SPEC 2000 벤치마크 프로그램
Table 1. SPEC 2000 benchmark programs

벤치마크	설 명
bzip2	압축
crafty	체스 경기 놀이
gap	그룹 이론 해석기
gcc	C 프로그래밍 언어 컴파일러
gzip	압축
mcf	조합 최적화
parser	워드 프로세서
twolf	배선 및 배치 모의실험기

표 2는 모의실험에 이용된 멀티코어 수퍼스칼라 프로세서 아키텍처의 사양을 나타낸 것이다. 멀티코어의 개수는 1 개, 2 개, 4 개, 8 개, 16 개를 대상으로, 윈도우의 크기는 4, 8, 16으로 정하였다. 각 코어는 순차 수퍼스칼라 방식으로 운영되며, 각 윈도우의 크기에 대하여 매 싸이클 마다 2 개, 4 개 및 8 개의 명령어를 인출한다. 각 코어의 연산유닛은 정수형 유닛, 로드 및 스토어 유닛, 그리고 분기명령어 유닛으로 구성되며, 각각 2 개에서 4 개로 구성된다.

표 2. 모의실험에 이용된 멀티코어 수퍼스칼라 프로세서 아키텍쳐 하드웨어의 사양

Table 2. The architecture specification of each superscalar processor core

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

1차 명령어 캐쉬와 1 차 데이터 캐쉬의 용량이 작으면 멀티코어 프로세서의 성능을 충분히 끌어올릴 수 없으므로, 명령어 캐쉬와 데이터 캐쉬는 64 KB의 용량을 갖도록 설정하였다. 단일 코어 프로세서의 경우, 명령어 캐쉬에 연관매핑(associative- mapping)을 이용하고 데이터 캐쉬에 직접매핑(direct- mapping)을 이용하더라도 높은 캐쉬 히트율을 얻을 수 있다. 그러나, MESI 프로토콜을 이용하는 멀티코어 프로세서에서는 데이터 캐쉬 역시 연관 캐쉬를 활용해야 충분한 데이터 캐쉬 히트율을 확보할 수 있다. 따라서, 본 실험에서 1 차 명령어 캐쉬 및 1 차 데이터 캐쉬는 모두 2 차 연관도(set associativity)를 갖는다. 단, 메인 메모리는 별도로 모델링하지 않았으므로, 2 차 통합 캐쉬는 100 % 히트가 난다고 가정하였다. 분기 명령어는 2 단계 적응형 분기 예측 방식을 적용하였으며, 14 비트 전역 히스토리 방식을 채택하였고 분기 어드레스 캐쉬는 2048 개의 엔트리를 갖는다 [9] .

Ⅳ. 모의실험 및 결과

그림 4부터 그림 6에 윈도우의 크기가 4, 8, 16일 때 1-코어부터 16-코어로 구성되는 멀티코어 수퍼스칼라 프로세서에 대한 모의실험 결과를 보였다.

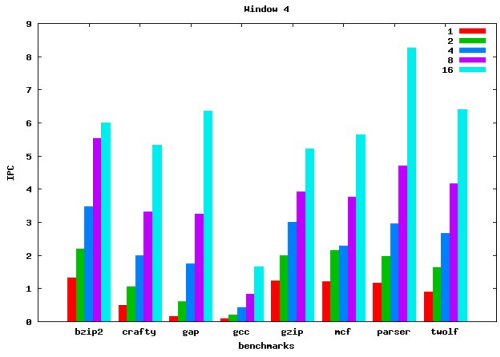

그림 4. 윈도우의 크기가 4일 때 1~16 개 멀티코어 수퍼스칼라 프로세서의 성능 결과

Fig 4. Performance of multi-core superscalar processor architecture for window size of 4.

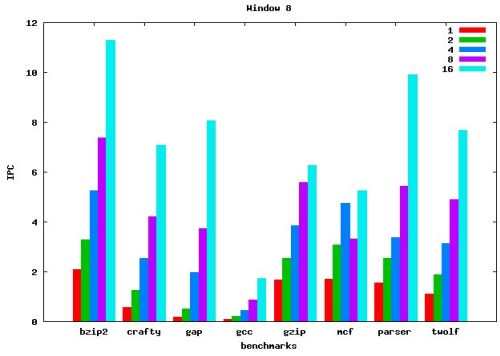

그림 5. 윈도우의 크기가 8일 때 1~16 개 멀티코어 수퍼스칼라 프로세서의 성능 결과

Fig 5. Performance of multi-core superscalar processor architecture for window size of 8.

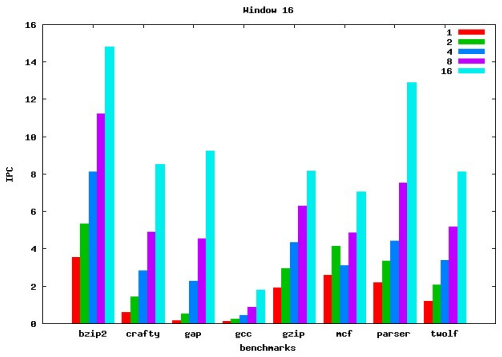

그림 6. 윈도우의 크기가 16일 때 1~16 개 멀티코어 수퍼스칼라 프로세서의 성능 결과

Fig 6. Performance of multi-core superscalar processor architecture for window size of 16.

윈도우의 크기가 4일 때, 1-코어, 2-코어, 4-코어, 8-코어, 16-코어 수퍼스칼라 프로세서 성능의 기하평균은 각각 0.6, 1.2, 2.0, 3.3, 5.2 IPC를 기록하였다. 윈도우의 크기가 8로 증가하면, 각 성능은 0.8, 1.4, 2.6, 3.9, 6.4 IPC로 향상되었다. 마지막으로 윈도우의 크기가 16일 때, 성능은 0.9, 1.7, 2.9, 4.8, 7.7 IPC로 크게 증가하였다. 각 벤치마크 프로그램 별로 수퍼스칼라 코어의 개수가 2 배가 되면 성능이 1.7 배 향상되었고, 윈도우의 크기가 2 배가 되면 성능이 1.2 배 증가하였다. 윈도우의 크기가 16인 16-코어 수퍼스칼라 프로세서는 1-코어 수퍼스칼라 프로세서보다 8.4 배의 성능이 향상되었다.

윈도우의 크기가 16일 때, 멀티코어 수퍼스칼라 프로세서의 성능을 같은 개수의 멀티코어 RISC 프로세서의 성능과 비교하면, 멀티코어 수퍼스칼라 프로세서는 멀티코어 RISC 프로세서보다 평균 2 배의 성능을 가져왔다. 이것은 수퍼스칼라 프로세서를 코어로 채택했을 경우, 단지 절반의 코어 개수를 가지고도 RISC 프로세서를 코어로 채택했을 때와 동일한 성능을 나타낼 수 있다는 것을 의미한다.

V. 결 론

본 논문에서는 현재 널리 이용되고 있는 멀티코어 프로세서 아키텍처의 단위 코어로서, 윈도우 크기가 4에서 16인 순차 수퍼스칼라 프로세서의 채택을 제안하였다. 이것을 위하여 8 개의 SPEC 2000 벤치마크에 대하여 코어 수 2 개부터 16 개까지의 멀티코어 프로세서에 대한 성능을 광범위한 모의실험을 통하여 측정하였다. 그 결과 코어의 개수가 같을 때, 멀티코어 수퍼스칼라 프로세서는 멀티코어 RISC 프로세서에 비하여 평균 2 배의 성능 향상을 기록하였다.

추후로, 각 코어에 대하여 순차 실행이 아닌 비순차 실행을 하는 수퍼스칼라를 채택하거나, 단일 분기 예측이 아닌, 다중 분기 예측(multiple branch prediction)을 도입하여 개별 코어의 성능을 극대화했을 때 멀티코어 프로세서의 성능에 나타나는 효과를 연구할 수 있다.

참 고 문 헌

[1] 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.

[2] D. E. Culler and J. P. Singh, "Parallel Computer Architecture", Morgan Kauffmann Publishers, Inc. Aug. 1998.

[3] S. W. Keckler, K. Olukotun, and H. Peter Hofsee, "Multicore Processors and Systems", Springer. 2009.

[4] T. Ungerer, B. Robic, and J. Silk, "Multithreaded Processors", The Computer Journal, Vol. 45, No. 3, 2002

[5] D. Pham et. al, "The Design and Implementation of a First-Generation CELL processor", ISSCC 2005.

[6] D. Genbrugge and L. Eeckhout, “Chip Multiprocessor Design Space Exploration through Statistical Simulation,” IEEE Transactions on Computers 58(12), pp.1668-1681, Dec. 2009.

[7] A. Rico, A. Duran. F. Cabarcas, Y. Etsion, A. Ramirex, and M. Valero, “Trace-driven Simulation of Multithreaded Applications,” ISPASS, 2011.

[8] 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, pp.124-134,

May. 1992.

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

저자 소개

이 종 복(정회원)

- 1964년 8월 : 20일생.
- 1988년 : 서울대 컴퓨터공학과 졸업.
- 1998년 : 동 대학 전기공학부 졸업 (공박).
- 1998년 ~ 2000년 : LG반도체 선임 연구원.
- 2000년 ~ 현재 : 한성대 정보통신 공학과 교수

- Tel : 02-760-4497, Fax : 02-760-4435
- E-mail : jblee@hansung.ac.kr

※ 본 연구는 2011년도 한성대학교 연구년 지원과제 임