http://dx.doi.org/10.7236/JIWIT.2012.12.3.9
JIWIT 2012-3-2

멀티코어 프로세서의 명령어 자취형 모의실험에 대한 연구

A Study of Trace-driven Simulation for Multi-core Processor Architectures

이종복*

Jongbok Lee

요 약 최근에 이르러, 과거 수퍼스칼라 프로세서의 하드웨어 복잡도와 전력소모 문제를 극복하기 위하여 멀티코어 프로세서가 상용화 되어 널리 이용되고 있다. 이러한 멀티코어 프로세서의 설계 초기 단계에서는 광범위한 모의실험을 수행하는 것이 매우 중요하다. 그러나 기존의 실행 위주(execution-driven)의 멀티코어 프로세서 모의실험기는 속도가 느리다는 단점이 있다. 본 논문에서는 이것을 극복하기 위하여 빠른 속도를 갖는 명령어 자취형 (trace-driven) 멀티코어 프로세서 모의실험기를 개발하였으며, 이것을 이용하여 2 개에서 16 개까지의 멀티코어 프로세서에 대하여 SPEC 2000 벤치마크를 입력으로하여 모의실험을 수행하였다. 모의실험 결과, 16개의 코어를 이용하는 멀티코어 프로세서에서 평균 4.1 IPC의 성능과 단일코어 대비 13.3 배의 성능 향상을 기록하였다.

Abstract In order to overcome the complexity and power problems of superscalar processors, the multi-core architecture has been prevalent recently. Although the execution-driven simulation is wide spread, the trace-driven simulation has speed advantages over the execution-driven simulation. We present a methodology to simulate multi-core architecture using trace-driven simulator. Using SPEC 2000 benchmarks as input, the trace-driven simulation has been performed for the cores ranging from 2 to 16 extensively. As a result, the 16-core processor resulted in 4.1 IPC and 13.3 times speed up over single-core processor on the average.

Key Words : multi-core processor, trace-driven simulation.

Ⅰ. 서 론

최근 30년간 마이크로 프로세서 연구는 싱글코어 프로세서를 더욱 빠르게 실행하도록 설계하는 것을 목표로 하였으며, 수퍼스칼라 프로세서가 그 대표적인 예라고 할 수 있다[1]. 그러나, 이러한 수퍼스칼라 프로세서는 하드웨어가 지나치게 복잡하고 전력 소모가 과다한 수준에 이르렀고, 따라서 더 이상 동작주파수를 높이는데 한계에 봉착하였다.

이것을 해결하기 위하여 멀티코어 프로세서가 대두되었다[2-5]. 멀티코어 프로세서는 응용 프로그램을 병렬화 할 수 있다는 것을 전제로 할 때, 프로세서의 성능을 높이는 유일한 해결책이다. 이러한 멀티코어 프로세서를 설계하기 위한 초기 단계에 광범위한 모의실험을 실행하

*정회원, 한성대학교 정보통신공학과
접수일자 2012년 5월 15일, 수정완료 2012년 6월 1일
게재확정일자 2012년 6월 8일

Received: 15 May 2012 / Revised: 1 June 2012 /
Accepted: 8 June 2012
*Corresponding Author: jblee@hansung.ac.kr
Dept. of Information and Communications Engineering, Hansung University, Korea

는 것이 매우 중요하다. 모의실험에는 실행 위주(execution-driven) 방식과 명령어 자취(trace-driven) 방식이 널리 쓰인다[6,7]. 실행 위주 방식은 정확하지만 시간이 많이 걸린다는 단점이 있다. 본 논문에서는 멀티코어 프로세서의 성능을 빠르고 효율적으로 측정할 수 있는 명령어 자취형 모의실험기를 제안하였다. 또한 본 모의실험기를 이용하여, SPEC 2000 벤치마크에 대하여 2-코어에서 16-코어까지의 성능을 측정하였다.

본 논문은 다음과 같이 구성된다. 2 장에서 멀티코어 프로세서 및 모의실험기에 대하여 살펴보고, 3 장에서 모의실험 환경에 대하여 고찰한다. 4 장에서 모의실험 결과를 보이며, 5 장에서 결론을 맺는다.

II. 멀티코어 프로세서 및 모의실험기

1. 멀티코어 프로세서의 구조

그림 1은 일반적인 N 차 멀티코어 프로세서 구조를 나타낸 것이다. 각 코어는 1부터 N까지 구성되는데, 자체적으로 1 차 캐쉬를 가지며, 메인 메모리와 연결되는 공통의 2 차 캐쉬를 공유한다. 각 코어에 설치된 1 차 캐쉬의 일관성(cache-coherency)을 위하여 MESI 프로토콜을 이용한다.

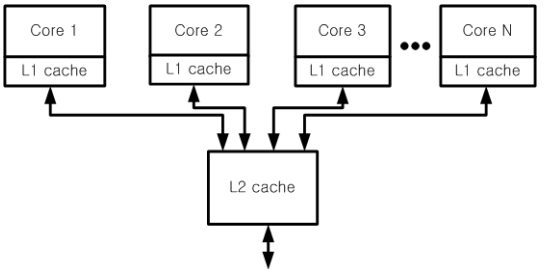

그림 1. 멀티코어 프로세서 구조
Fig. 1. The multi-core processor architecture

2. 멀티코어 프로세서 모의실험기

멀티코어 프로세서는 제 1 단계 명령어 자취의 발생, 제 2 단계 명령어 자취에 대한 멀티코어 프로세서의 실행으로 나누어진다. 제 1 단계에서 명령어 자취는 임의의 차수의 멀티코어에 적합하도록 발생되었다.

제 2 단계 멀티코어 프로세서의 실행은 다음과 같다. 본 모의실험기는 각 코어가 단순한 RISC 프로세서 뿐만 이 아니라, 수퍼스칼라 프로세서로도 동작할 수 있도록 개발되었다. 따라서 2 개 이상의 명령어를 인출받을 수 있다. 제 2 단계의 과정을 자세하게 기술하면 그림 2에 나타낸 것과 같다.

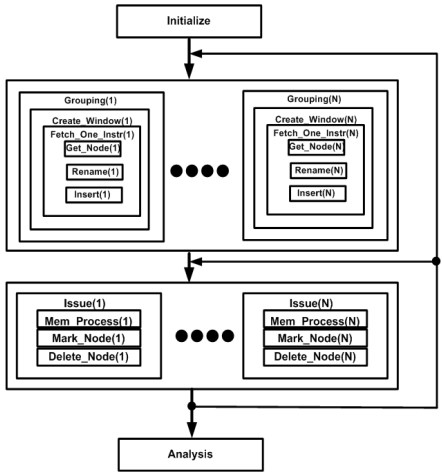

그림 2. 멀티코어 프로세서 모의실험기
Fig. 2. The multi-core processor architecture simulator.

(1) 명령어 인출 및 재명명

Initialize 함수에서 초기화 작업을 거친 후에, Grouping 함수가 Create_Window 함수를 부르고 이것은 다시 Fetch_One_Instr 함수를 불러서 각 코어는 매 싸이클마다 새로운 명령어를 인출받는다. 한 싸이클에 1 개의 명령어를 인출받는 RISC 프로세서와 달리, 수퍼스칼라 프로세서에서 만일 2 개 이상의 명령어를 인출받을 때, 분기명령어를 만나면 그 이상의 인출을 하지 않고 인출을 멈춘다. Get_Node 함수에서 인출한 명령어는 Rename 함수에서 재명명 (renaming) 작업을 거치면서 명령어 종속에 의한 타임스탬프(timestamp) 값을 설정받는다. 타임스탬프 방식은 명령어 자취를 이용하는 모의실험에서 데이터 종속성을 신속하고 효율적으로 부여할 수 있는 핵심적인 방법이다. 모든 명령어는 인출 초기에 현재 싸이클과 명령어 자체의 지연 싸이클을 더한 값을 타임스탬프로 설정 받는다. 이후의 디코드 및 재명명 단계에서, 각 명령어의 소오스 레지스터 1과 소오스 레지스터 2에 대하여 레지스터 화일을 검색하여 같은 이름을 갖는 레지스터를 찾아서 해당하는 타임스탬프 값을 읽는다. 만일 현재 명령어의 타임스탬프 값보다 소오스 레지스터 1

또는 소오스 레지스터 2의 타임스탬프 값이 더 크다면, 명령어 자체의 타임스탬프는 그 값으로 대체된다. 또한, 명령어의 목적레지스터의 타임스탬프도 같은 값으로 대체되어, 추후에 이 목적 레지스터를 소오스 레지스터로 참조하는 명령어에 대하여 올바른 타임스탬프 값을 가질 수 있도록 한다. 레지스터 화일의 타임스탬프 값에 의하여, 멀티코어 프로세서 명령어 간의 종속성이 유지되어 성능을 구하는데 반영된다. 이와 같이 재명명을 거친 명령어는 insert 함수에서 각 코어의 명령어 윈도우에 삽입된다.

(2) 명령어 이슈

Issue 함수로 진행하면, 싸이클이 증가함에 따라서 윈도우 내의 명령어는 자체의 타임스탬프 값이 현재 싸이클 보다 작거나 같을 때 삭제될 수 있다. 그러나, 첫 단계인 Mark_Node 함수에서 즉각 삭제하지 않고 삭제 가능 표시만 하며, 다음 단계인 Delete_Node 함수에서 삭제 가능 표시를 한 명령어를 실제로 삭제한다. 이 과정에서 만일 삭제 가능 표시를 하지 않은 명령어를 만나면 그 이후의 명령어는 삭제를 즉각 종료한다. 이렇게 함으로써, RISC 프로세서가 아닌 비순차실행(out-of-order execution) 방식의 수퍼스칼라 프로세서코어인 경우에도 순차종료(in-order completion)를 보장할 수 있다.

(3) 멀티코어 시뮬레이션

모든 N개의 멀티코어에 대하여 해당 코어의 윈도우 공간에 Grouping 함수를 이용하여 명령어를 인출해서 채우고, 역시 N개의 멀티코어에 대하여 Issue 함수로 각 코어에 대하여 명령어를 실행하면서 종속성에 의하여 부여된 명령어의 타임스탬프가 충족되면 삭제한다. 이 과정은 입력으로 주어진 벤치마크 프로그램의 모든 명령어가 소진될 때까지 반복된다.

위 과정이 한번 실행될 때 마다 싸이클이 증가하므로, 매 싸이클 당 명령어 실행 및 삭제가 가장 오래 걸리는 코어가 해당 싸이클 수를 결정한다. 모의실험에 입력으로 쓰인 명령어의 총 개수를 처리하기 위하여 소요된 총 싸이클 수로 나누어, 멀티코어 프로세서 시스템의 성능의 척도인 IPC(Instruction Per Cycle)를 계산할 수 있다.

Ⅲ. 모의실험 환경

표 1은 모의실험에 이용된 SPEC 2000 정수형 벤치마크 프로그램이다. Simplescalar를 통하여 MIPS IV 10억 개의 명령어 자취를 임의의 차수의 멀티코어 프로세서에 적합하도록 발생시켜서 모의실험기에 입력하였다[8].

표 1. SPEC 2000 정수형 벤치마크 프로그램
Table 1. SPEC 2000 integer 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
paser	워드 프로세서
twolf	배선 및 배치 모의실험기

표 2는 모의실험에 이용된 멀티코어 프로세서 아키텍쳐의 사양을 나타낸 것이다. 멀티코어의 개수는 1 개, 2 개, 4 개, 8 개 및 16 개를 대상으로 하였다. 각 코어는 RISC 방식으로 운영되므로, 윈도우의 크기는 1이며,매 싸이클 마다 1 개의 명령어를 인출, 이슈, 실행 및 기록한다. 각 코어의 연산유닛은 정수형 유닛, 로드 스토어 유닛, 그리고 분기명령어 유닛 각 1 개로 구성된다.

표 2. 모의실험에 이용된 멀티코어 프로세서 아키텍쳐 하드웨어의 사양
Table 2. The architecture specification of each core

항목	값
멀티코어 수	1, 2, 4, 8, 16
1차 명령어 캐쉬 및 데이터 캐쉬의 공통 사항	64 KB, 2 차 연관, 16 B 미스 페날티 10 싸이클
명령어 윈도우의 크기	1
인출율, 이슈율, 퇴거율	1
연산유닛 사양	산술논리(1), 분기(1), 로드(1), 스토어(1)
분기 어드레스 캐쉬	2 K 엔트리
분기 예측기	2 단계 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)

1 차 명령어 캐쉬와 1 차 데이터 캐쉬는 각 코어마다 설치되며, 64 KB의 용량을 갖도록 설정하였으며, 2 차 연관도(2-way set associativity) 방식을 통하여 접근된다. 그러나, 모든 코어에 의하여 공유되는 2 차 캐쉬는 충분한 용량으로 인하여 100 % 히트가 난다고 가정하였다. 분기 명령어는 2 단계 적응형 분기 예측 방식을 적용하였으며 분기 어드레스 캐쉬는 2048 개의 엔트리를 갖는다.[9]

본 연구에서 개발한 모의실험기는 각 코어가 RISC 또는 순차 및 비순차실행 방식의 수퍼스칼라로 동작할 수 있으나, 본 모의실험에서는 간단한 코어를 지향하기 위하여 각 코어마다 RISC 방식으로 한 싸이클에 1 개의 명령어만을 인출하고 실행할 수 있도록 하였다.

Ⅳ. 모의실험 및 결과

그림 3에서 1-코어에서 16-코어 프로세서에 대하여 8 개의 정수형 SPEC 벤치마크를 입력으로 하는 모의실험 결과를 보였다. 모의실험 결과에서 알 수 있듯이, 각 벤치마크 프로그램 별로 코어의 개수가 증가할 수록 성능이 증가함을 알 수 있다. 단일코어의 성능을 기준으로 하였을 때, 16-코어에서 gap이 32 배로 최고의 성능 향상을 기록하였으나, bzip2는 프로그램의 복잡도로 인하여 8.2 배에 그쳤다. 벤치마크 프로그램 중, gcc의 1 차 명령어 캐쉬 히트율이 약 42 %로 저조하여 성능에 큰 손실을 가져왔다. 1 차 데이터 캐쉬를 직접매핑(direct-mapping) 방식 으로 구현하였을 때는 MESI 캐쉬 일관성 프로토콜에 의하여 캐쉬 히트율이 저조하였으나, 2 차 연관 캐쉬로 구성하여 95 % 이상의 데이터 캐쉬 히트율을 기록함으로써, 성능 하락을 방지하였다.

각 코어의 차수별 기하평균을 구하면 단일코어의 경우 0.31 IPC를 기록하였으며, 단일코어 대비 2-코어에서 0.66 IPC 성능이 2.2 배가 되었다. 같은 기준으로 4-코어에서는 4.3 배인 1.31 IPC, 8-코어에서 8.2 배인 2.5 IPC를 나타냈다. 마지막으로 16-코어의 경우 4.06 IPC로 단일코어 대비 13.3 배의 성능을 가져왔다. 위 결과는 4-코어 때 약 4 배, 8-코어 때 약 8 배, 16-코어 때 10 배~15 배의 성능향상을 기록하는 기존의 실행 위주 멀티코어 프로세서 모의실험기를 통하여 얻은 결과와 일치한다[10][11].

멀티코어 프로세서의 모의실험에 명령어 자취형 방식을 채택함으로써, 실행위주 방식보다 결과를 10 배 이상 빠르게 얻을 수 있었다. 또한, 간단한 RISC 코어를 이용한 멀티코어 프로세서 아키텍쳐라도 기존의 싱글 코어 수퍼스칼라 프로세서 아키텍쳐로는 도달할 수 없었던 높은 성능을 얻을 수 있음을 알 수 있다.

Ⅴ. 결 론

본 논문에서는 현재 널리 이용되고 있는 멀티코어 프로세서 아키텍처에 대하여 적용할 수 있는 빠른 명령어 자취형 모의실험기를 개발하였다. 또한 2 개부터 16 개까지의 멀티코어 프로세서에 대하여 SPEC 벤치마크를 입

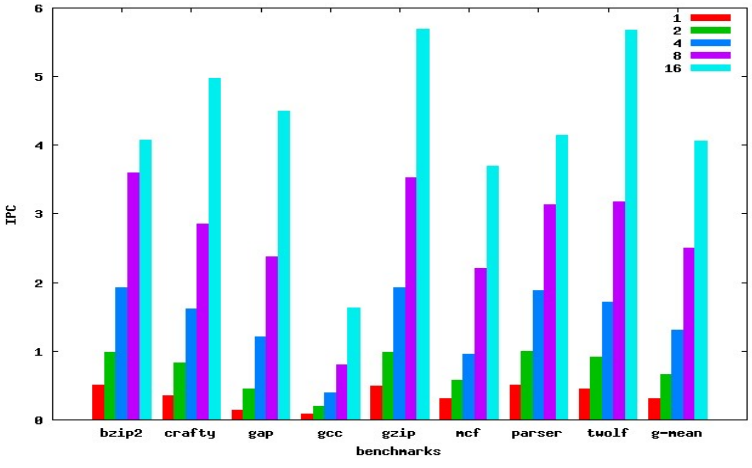

그림 3. 멀티코어 수에 따른 SPEC 벤치마크의 IPC 성능
Fig. 3. IPC of SPEC benchmarks for different number of multi-cores

력으로 하여 그 성능을 모의실험을 통하여 측정하였다. 그 결과, 16 개의 코어에서 단일코어 대비 13.3 배의 성능 향상을 가져왔으며, 평균 4.1 IPC의 성능을 기록하였다.

추후로, 각 코어에 대하여 RISC 방식이 아닌, 순차 및 비순차 실행을 하는 수퍼스칼라를 채택하고 각 코어의 윈도우의 크기를 변화시키면서 모의실험을 통하여 그 성능을 연구할 계획이다. 이 때, 다중 분기 예측(multiple branch prediction)을 도입하였을 때 멀티코어 프로세서의 성능에 나타나는 효과 역시 검토할 수 있다. 또한, 동질 코어(homogeneous core)가 아닌 비동질 코어(heterogeneous core)를 채택하는 비대칭 칩 멀티프로세서(asymmetric chip multiprocessor) 구조에 대한 연구 및 모의실험도 필요하다. 더 나아가, 최신 경향인 코어 수 64개 이상의 매니코어 (many-core) 아키텍쳐에 대한 연구 및 모의실험도 수행할 예정이다.

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

[9] 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.

[10] S. Biswas, et. al, "Multi-Execution : Multicore Caching for Data-Similar Executions," Proceedings of the 36th Annual International Symposium on Computer Architecture, pp. 164-173.

[11] M. Monchiero, et. al, "How to Simulate 1000 Cores," ACM SIGARCH Computer Architecture News archive, Vol. 37, Issue 2, May 2009, pp. 10-19

저자 소개

이 종 복(정회원)

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

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

※ 본 연구는 한성대학교 교내연구장려금 지원과제 임.