ISSN 1975-8359(Print) / ISSN 2287-4364(Online)
The Transactions of the Korean Institute of Electrical Engineers Vol. 62, No. 2, pp. 252~256, 2013
http://dx.doi.org/10.5370/KIEE.2013.62.2.252

SPEC 벤치마크 프로그램에 대한 매니코어 프로세서의 성능 연구

A Performance Study on Many-core Processor Architectures with SPEC Benchmark Programs

이 종 복*
(Jongbok Lee)

Abstract - In order to overcome the complexity and performance limit problems of superscalar processors, the multi-core architecture has been prevalent recently. Usually, the number of cores mostly used for the multi-core processor architecture ranges from 2 to 16. However in the near future, more than 32-cores are likely to be utilized, which is called as many-core processor architecture. Using SPEC 2000 benchmarks as input, the trace-driven simulation has been performed for the 32 to 1024 many-core architectures extensively. For 1024-cores, the average performance scores 15.7 IPC, but the performance increase rate is saturated.

Key Words : Many-core processor, Multi-core processor, Trace-driven simulation

1. 서 론

최근 30년간 마이크로프로세서 연구는 싱글코어 프로세서를 더욱 빠르게 실행하도록 설계하는 것을 목표로 하였으며, 슈퍼스칼라 프로세서로 발전하였다 [1]. 그러나, 수퍼스칼라 프로세서는 하드웨어의 복잡도에 비하여 성능 향상이 미미하여 한계에 봉착하였으므로, 이것을 해결하기 위하여 멀티코어 (multi-core) 프로세서가 대두되었다 [2-7]. 멀티코어 프로세서는 응용 프로그램을 병렬화할 수 있다는 것을 전제로 할 때, 프로세서의 성능을 높일 수 있는 돌파구로서 현재 광범위하게 쓰이고 있다. 이러한 멀티코어 프로세서는 전문 그래픽 분야를 제외하고는 주로 16 개 이하의 코어로 구성된다. 그러나 향후에는 32 개 이상의 코어로 구성되는 매니코어 (many-core) 프로세서 아키텍처가 대세가 될 전망이다.

본 논문에서는 매니코어 프로세서의 성능을 분석하기 위하여, 32-코어에서 1024-코어의 매니코어 프로세서에 대하여 SPEC 2000 벤치마크를 입력으로 모의실험을 수행하여 그 성능을 측정하였다. 그 결과, 코어의 개수가 증가할수록 매니코어 프로세서 아키텍처의 성능이 증가하였으나, 코어의 개수가 1024 개에 가까울수록 코어 간의 데이터 종속, 메모리 종속에 의한 통신 오버헤드(communication overhead)로 인하여 성능의 향상률이 급격하게 둔화되었다. 본 연구의 실험결과에 따라서, 매니코어 프로세서의 설계 초기 단계에서 필요한 성능과 코어의 개수에 대한 결정을 내릴 때 유용하다. 본 논문은 다음과 같이 구성된다. 2 장에서 매니코어 프로세서의 구조에 대하여 살펴보고, 3 장에서 모의실험기 및 모의실험 환경에 대하여 고찰한다. 4 장에서 모의실험 결과를 보이며, 5 장에서 결론을 맺는다.

* Corresponding Author : Dept. of Information and Communication Engineering, Hansung University, Korea
E-mail : jblee@hansung.ac..kr
Received : December 15, 2012; Accepted : January 25, 2013

2. 매니코어 프로세서 아키텍처

2.1 매니코어 프로세서의 구조

그림 1은 N 개의 코어로 구성되는 공유 메모리 (shared-memory) 매니코어 프로세서의 일반적인 구조를 나타낸 것이다. 각 코어는 1부터 N까지 구성되는데, 자체적으로 1 차 명령어 캐쉬와 1 차 데이터 캐쉬를 가진다. 또한 각 코어는 메인 메모리와 연결되는 공통의 2 차 통합 캐쉬 (unified cache)를 공유한다. 각 코어에 설치된 1 차 데이터 캐쉬의 일관성(cache-coherency)을 위하여 MESI 프로토콜을 이용한다. MESI 프로토콜은 캐쉬 블럭을 변경(Modified), 독점(Exclusive), 공유(Shared), 무효(Invalid)의 4 가지 상태로 관리한다. 이하 본 논문에서 기술하는 명령어 캐쉬와 데이터 캐쉬는 모두 1 차 캐쉬를 의미한다.

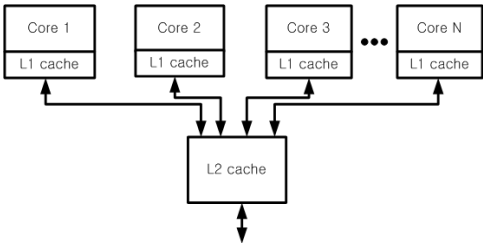

그림 1 매니코어 프로세서의 구조
Fig. 1 The many-core processor architecture

각 코어 내에서의 레지스터 재명명 (register renaming)은 기존의 레지스터 재명명 방법으로 수행된다. 한편, 서로 다른 코어 간의 레지스터 재명명을 위하여, 한 코어의 레지스터 값을 다른 코어로 전송한다. 만일에 어느 코어에서 레지스터 값이 새로 기록되지 않는다면 그대로 다음 코어로 전

송이 되지만, 기록된다면 이전 레지스터 값의 전송은 중단되고 그 대신 새로운 레지스터 값이 전송된다.

각 코어 내에서의 메모리 종속은 기존의 방법으로 수행되며, 상이한 코어 간의 메모리 종속은 주소 해결 버퍼(Address Resolution Buffer)를 이용하여 관리된다 [8]. 주소 해결 버퍼를 통하여, 같은 메모리 주소를 갖는 스토어 값들이 여러 코어에 존재하여 메모리 재명명을 수행할 수 있다. 주소 해결 버퍼의 값들은 각 코어가 완료할 때 업데이트 되거나 폐기된다. 로드에 대하여 올바른 스토어 데이터를 공급하기 위하여, 주소 해결 버퍼는 가장 근접한 스토어가 로드에게 데이터를 제공하도록 한다. 만일 생산자 스토어가 수행되기 전에 소비자 로드가 먼저 수행이 된다면, 주소 해결 버퍼가 이것을 감지하여 잘못된 로드를 수행한 코어를 원 상태로 복구한다.

2.2 매니코어 프로세서를 구성하는 캐쉬

기존의 RISC나 수퍼스칼라 프로세서의 명령어 캐쉬가 성능에 끼치는 영향력이 크므로, 전통적으로 2 차 이상의 연관도 (2-way set associative) 및 충분한 용량으로 구성하여 캐쉬 미스에 의한 성능의 손실을 최소화하였다. 이것을 매니코어 프로세서에서도 적용할 수 있으며, 2 차 이상의 연관도로 명령어 캐쉬를 구성하고 그 용량에 따라 충분한 캐쉬 히트율을 확보할 수 있다.

한편, 전통적인 RISC 및 수퍼스칼라 프로세서에서 이용되는 데이터 캐쉬는 명령어 캐쉬와는 달리, 직접 캐쉬(direct-mapped cache)로 구성해도 충분한 성능을 얻을 수가 있었다. 그러나, 매니코어 프로세서 시스템에서는 여러 코어 간의 캐쉬 일관성(Cache Coherency)을 위한 MESI 프로토콜의 적용으로 인하여 캐쉬의 데이터를 쓰기 무효화(write-invalidate)하는 경우가 빈번히 발생한다. 따라서, 직접 캐쉬로는 적정한 캐쉬 히트율을 확보할 수가 없어서 매니코어 프로세서의 성능 손실이 크므로, 데이터 캐쉬 역시 2 차 이상의 연관도로 구성해야한다.

본 논문에서는 위와 같은 이유 및 캐쉬의 하드웨어 비용을 절감하고 매니코어 프로세서 아키텍처 성능의 극대화를 위하여, 명령어 캐쉬와 데이터 캐쉬를 모두 2 차 연관도로 구성하였다. 그리고, 공정한 성능 비교를 위하여 코어의 개수가 늘어남에 따라 단위 명령어 캐쉬와 데이터 캐쉬의 용량이 줄어들도록 매니코어 시스템을 설계하였다.

3. 모의실험 환경

3.1 매니코어 프로세서 모의실험기

매니코어 프로세서는 제 1 단계 명령어 자취의 발생, 제 2 단계 명령어 자취에 대한 매니코어 프로세서의 실행으로 나누어진다. 제 1 단계에서 명령어 자취는 쓰레드 단위 병렬성이 임의의 차수의 매니코어에 대응시키는데 적합하도록 발생되었다. 제 2 단계에서 각 코어가 단순한 RISC 프로세서로 동작하여 매니코어 프로세서가 실행된다. 제 2 단계의 과정을 자세하게 기술하면 다음과 같다.

3.1.1 명령어 인출, 재명명 및 이슈

그림 2는 매니코어 프로세서 모의실험기의 순서도이다. Initialize 블럭에서 초기화 작업을 거친 후에, Grouping 블럭이 Create_Window 블럭을 부르고 이것은 다시 Fetch_One_Instr 블럭을 불러서 각 코어는 매 싸이클마다 1 개의 명령어를 인출 받는다. Get_Node 블럭에서 인출한 명령어는 Rename 블럭에서 재명명 작업을 거치면서 명령어 종속에 의한 타임스탬프(timestamp) 값을 설정 받는다. 타임스탬프 방식은 명령어 자취를 이용하는 모의실험에서 데이터 종속성을 신속하고 효율적으로 부여할 수 있는 핵심적인 방법이다. 레지스터 화일의 타임스탬프 값에 의하여, 매니코어 프로세서 명령어 간의 종속성이 유지되어 성능을 구하는데 반영된다. 이와 같이 재명명을 거친 명령어는 Insert 블럭에서 각 코어의 명령어 윈도우에 삽입된다. Issue 함수로 진행하면, 싸이클이 증가함에 따라서 윈도우 내의 명령어는 자체의 타임스탬프 값이 현재 싸이클보다 작거나 같고, 해당 연산유닛이 활용 가능할 때 삭제될 수 있다.

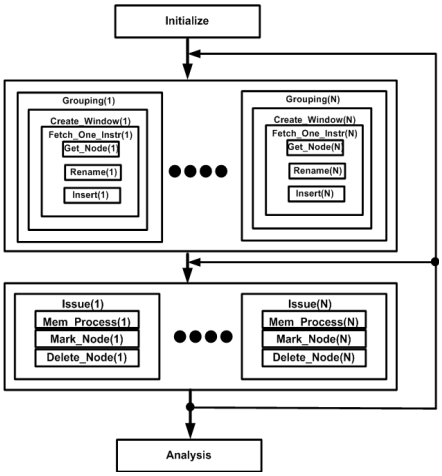

그림 2 매니코어 프로세서 모의실험기의 순서도

Fig. 2 The flow chart of many-core processor architecture simulator

3.1.2 매니코어 시뮬레이션

모든 N 개의 매니코어에 대하여 해당 코어의 윈도우 공간에 Grouping 블럭을 이용하여 한 개의 명령어를 인출해서 채우고, 역시 N 개의 매니코어에 대하여 Issue 블럭으로 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 명령어의 실행 및 삭제가 가장 오래 걸리는 코어가 해당 싸이클 수를 결정한다. 모의실험에 입력으로 쓰인 명령어의 총 개수를 처리하기 위하여 소요된 총 싸이클 수로 나누어, 매니코어 프로세서 시스템의 성능의 척도인 IPC(Instruction Per Cycle)를 계산할 수 있다.

3.2. 벤치마크 및 매니코어 프로세서의 사양

표 1은 모의실험에 이용된 SPEC 2000 정수형 벤치마크 프로그램이다. Simplescalar를 통하여 MIPS IV 10억 개의 명령어 자취를 구하고, 명령어 자취의 쓰레드 단위 병렬성을 코어 단위 병렬성으로 대응시켜서 매니코어 모의실험기에 입력하였다 [9].

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 매니코어 프로세서 아키텍처의 사양을 나타낸 것이다. 매니코어의 개수는 32 개, 64 개, 128 개, 256 개, 512 개 및 1024 개를 대상으로 하였다. 각 코어는 RISC 방식으로 운영되므로, 매 싸이클마다 1 개의 명령어를 인출, 이슈, 실행 및 기록한다. 각 코어의 연산유닛은 정수형 유닛, 로드 스토어 유닛, 그리고 분기명령어 유닛으로 구성된다.

표 2 모의실험에 이용된 매니코어 프로세서 아키텍처 하드웨어의 사양
Table 2 The architecture specification of each core

항목	값					
매니코어의 수	32	64	128	256	512	1024
명령어 캐쉬 및 데이터 캐쉬의 용량 (KB)	32	16	8	4	2	1
연산유닛 사양	산술논리(1), 분기(1), 로드(1), 스토어(1)					
태스크 어드레스 캐쉬	2 K 엔트리					
태스크 예측기	2 단계 14 비트 전역 히스토리 방식 미스 페널티 6 싸이클					
이슈 지연 싸이클	산술논리(1), 분기(1), 로드(1), 스토어(1)					
결과 지연 싸이클	산술논리(1), 분기(1), 로드(1), 스토어(1)					

명령어 캐쉬와 데이터 캐쉬는 각 코어마다 설치되는데, 32 코어에서 32 KB, 64 코어에서 16 KB, ... , 1024 코어에서 1 KB의 용량을 갖도록 설정하였다. 그 이유는 코어의 개수가 2 배로 증가할수록 캐쉬의 개수 역시 2 배로 증가하여 그 용량이 증가하기 때문에, 공정한 성능 비교를 하기 위해서이다. 각 캐쉬는 2 차 연관도(2-way set associative) 방식을 통하여 접근된다. 각 캐쉬의 블럭 크기는 16 B로 정하였으며, 1 차 캐쉬 미스가 발생하였을 때는 20 싸이클의 페널티를 갖는다. 그러나, 메인 메모리는 별도로 모델링하지 않았기 때문에, 모든 코어에 의하여 공유되는 2 차 통합 캐쉬는 충분한 용량으로 인하여 100 % 히트가 난다고 가정하였다. 태스크의 예측을 위하여, 2 단계 적응형 분기 예측 방식을 응용한 2 단계 적응형 태스크 예측 방식을 적용하였으며, 태스크 어드레스 캐쉬는 2048 개의 엔트리를 갖는다 [10].

4. 모의실험 및 결과

그림 3(a)부터 3(h)까지 32-코어에서 1024-코어 프로세서에 대하여, 8 개의 정수형 SPEC 벤치마크를 입력으로 하는 매니코어 프로세서의 모의실험 결과를 보였다. 32-코어일 때 *gcc*가 최저인 1.4 IPC를, *twolf*가 최고인 4.4 IPC를 기록하였다. *gcc*의 성능이 낮은 이유는 프로그램의 불규칙성에 의하여 명령어 캐쉬 히트율이 상대적으로 저조하기 때문이다. 256-코어의 경우 *gcc*가 여전히 가장 낮은 6.6 IPC를 나타냈고, *twolf*가 13.9 IPC로 최고값을 기록하였다. 1024-코어일 때는 *mcf*가 코어 단위별 줄어든 캐쉬 용량의 크기에 민감하여 최저인 11.8 IPC를 나타냈고, 19.3 IPC를 기록한 *twolf*를 제치고 *parser*가 22.3 IPC로 최고값을 나타냈다.

모의실험 결과에서 알 수 있듯이, 각 벤치마크 프로그램별로 코어의 개수가 증가할수록 성능이 증가하였다. 항목별 성능 결과에 대하여 기하평균값을 구하면, 코어의 개수가 2 배가 될 때 최저 1.2 배에서 최고 1.6 배 성능이 향상되었다. 대부분의 매니코어 프로세서에서 코어의 개수가 128에서 256으로 증가하는 구간에서 성능의 그래프가 가장 가파르게 나타났다. 따라서, 본 모의실험 결과에 의하면 256-코어로 구성되는 매니코어 프로세서의 가격대 성능비가 가장 높다.

각 코어의 차수별 기하평균을 구하면 32-코어의 경우 2.5 IPC를 기록하였으며, 64-코어에서 4.1 IPC로 성능이 1.6 배가 되었다. 128-코어에서는 6.1 IPC, 256 코어의 경우 9.6 IPC, 512-코어일 때 13.7 IPC를 각각 기록하였다. 마지막으로 1024-코어에서 최고 성능인 15.7 IPC를 나타냈다. 코어의 개수가 증가할수록 성능이 증가하였으나, 코어의 개수가 1024에 접근함에 따라 성능의 향상률이 급격히 둔화되었다. 그 이유는 캐쉬의 용량이 감소하였고, 동시에 코어 간의 데이터 종속 및 메모리 종속에 의하여 통신 오버헤드가 증가하였기 때문이다. 위 결과는 기존에 발표된 매니코어 프로세서 모의실험 연구 결과와 맥락을 같이 한다 [11-12].

5. 결 론

본 논문에서는 32 개부터 1024 개까지의 매니코어 프로세서 아키텍처에 대하여 SPEC 벤치마크를 입력으로 하여 모의실험을 통하여 성능을 측정하고 분석하였다. 그 결과, 코어의 개수가 증가할수록 매니코어 프로세서의 성능이 증가하여 1024-코어에서 평균 15.7 IPC를 기록하였다. 그러나, 코어의 개수가 1024 개에 근접할수록 코어 간의 통신 오버헤드로 인하여 성능의 향상률이 크게 둔화되었다.

추후로, 동질 코어(homogeneous core)가 아닌 비동질 코어(heterogeneous core)를 채택하는 비대칭 칩 멀티프로세서

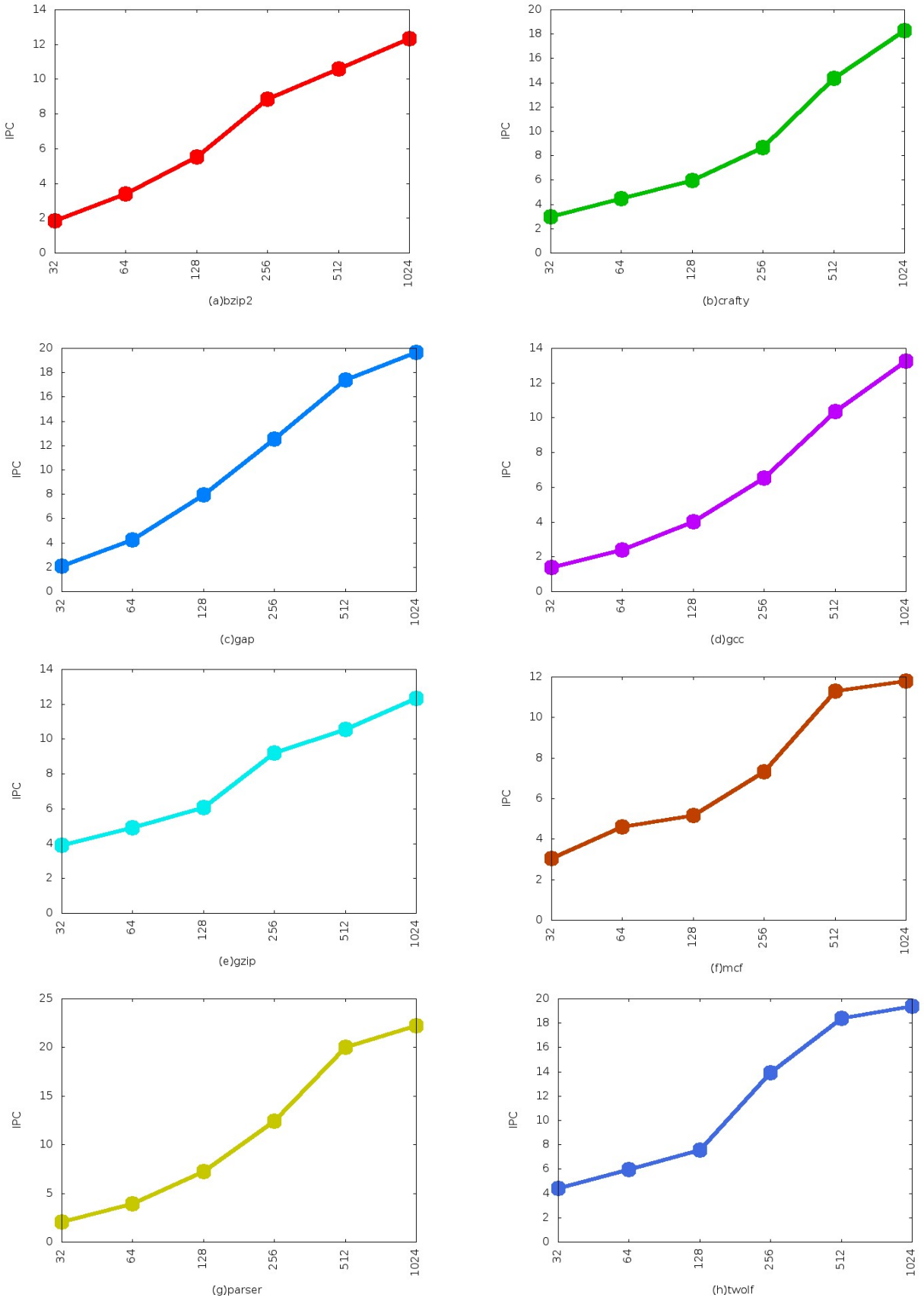

그림 3 매니코어 아키텍처의 모의실험 결과

Fig. 3 Simulation results for the many-core architectures from 32 to 64 cores

(asymmetric chip multiprocessor) 구조의 성능에 대한 연구가 필요하다. 또한, 매니코어 프로세서 아키텍처에서의 전력 소모량을 계산하여, 코어의 개수에 따라 성능과 전력을 트레이드 오프할 수 있는 최적화된 하드웨어 사양에 대한 연구를 수행할 예정이다.

감사의 글

본 연구는 한성대학교 교내연구장려금 지원과제 임.

References

[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] M. Frankilin, G. S. Sohi, "ARB: A Hardware Mechanism for Dynamic Reordering of Memory References," IEEE Transactions on Computers, Vol. 45, No. 5, May 1996.

[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.

[10] 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.

[11] S. Biswas, et. al, "Multi-Execution : Multicore Caching for Data-Similar Executions," International Symposium on Computer Architecture, Jun. 2009.

[12] M. Monchiero, et. al, "How to Simulate 1000 Cores," ACM SIGARCH Computer Architecture News archive, Vol. 37, Issue 2, May 2009, pp. 10-19 [1] A. Ghosh, S. Devadas, K. Keutzer and J. White, "Estimation of Average Switching Activity in Combinational and Sequential Circuits," ACM/IEE Design Automation Conf., pp. 253-259, 1992.

저 자 소 개

이 종 복 (李 鍾 馥)

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