The Transactions of the Korean Institute of Electrical Engineers Vol. 61, No. 10, pp. 1502~1507, 2012
http://dx.doi.org/10.5370/KIEE.2012.61.10.1502

멀티코어 비순차 수퍼스칼라 프로세서의 성능 연구

A Performance Study of Multi-core Out-of-Order Superscalar Processor Architecture

이 종 복*
(Jong-Bok Lee)

Abstract - In order to overcome the hardware complexity and power consumption 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 increased. In this paper, out-of-order superscalar processor is utilized for the multi-core processor architecture. Using SPEC 2000 benchmarks as input, the trace-driven simulation has been performed for the out-of-order superscalar cores between 2 and 16 extensively. As a result, the 16-core out-of-order superscalar processor for the window size of 16 resulted in 17.4 times speed up over the single-core out-of-order superscalar processor, and 50 times speed up over the single core RISC processor. When compared for the same number of cores on the average, the multi-core out-of-order superscalar processor performance achieved 3.2 times speed up over the multi-core RISC processor and 1.6 times speed up over the multi-core in-order superscalar processor.

Key Words : Multi-core processor, Superscalar processor, Out-of-order execution

1. 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 하드웨어의 복잡도를 높여도 성능이 그 이상 증가하지 않는 한계에 이르러 더 이상 프로세서의 동작주파수를 높이는데 난관에 봉착하였다. 이것을 해결하기 위하여 멀티코어 프로세서가 대두되어 널리 사용되고 있다 [2-7]. 멀티코어 프로세서는 응용 프로그램을 병렬화할 수 있다는 것을 전제로 할 때, 프로세서의 성능을 높이는 유일한 해결책이다. 이러한 멀티코어 아키텍처의 단위 코어 프로세서로는 한 싸이클에 최대 한 개의 명령어만을 처리할 수 있는 간단한 RISC가 주로 쓰이고 있는데, 그 이유는 코어의 개수를 늘릴 수록 성능 향상을 얻을 수 있기 때문에 굳이 단위 코어의 하드웨어를 복잡하게 할 필요가 없기 때문이다. 그러나, 코어의 개수가 32개 미만으로서 매니코어(many-core)가 아닌 멀티코어 프로세서 시스템에서 그 성능의 극대화를 위하여 RISC보다 훨씬 성능이 뛰어난 수퍼스칼라 프로세서를 단위 코어로 활용한다면, 하드웨어 복잡도가 그리 높지 않으면서도 적은 개수의 코어를 가지고도 전체 시스템의 성능이 더욱 향상될 수 있다.

이 때 멀티코어를 구성하는 수퍼스칼라 프로세서는 순차 방식과 비순차 방식으로 다시 나눌 수 있다. 비순차 방식의 수퍼스칼라의 하드웨어 구조가 순차 방식보다 복잡하지만, 훨씬 높은 성능을 낼 수가 있다. 또한 비순차 수퍼스칼라를 단위 코어로 채택하더라도, 메모리 관련 명령어를 순차 시행시키고 메모리 관련 명령어를 제외한 나머지 명령어들만 비순차시행 시킨다면 하드웨어의 복잡도가 그리 높지 않다.

본 논문에서는 비순차 수퍼스칼라 프로세서를 멀티코어 시스템의 단위 구조로 이용하는 아키텍처를 제안하였다. 또한, 멀티코어 비순차 수퍼스칼라 모의실험기를 개발하여, SPEC 2000 벤치마크에 대하여 윈도우의 크기가 4에서 16의 범위인 2-코어에서 16-코어의 비순차 수퍼스칼라 프로세서의 성능을 측정하고 분석하였다. 제안한 아키텍처의 성능의 우수성을 입증하기 위하여, 멀티코어 RISC 프로세서 및 멀티코어 순차 수퍼스칼라 프로세서와 성능을 비교하였다.

2. 멀티코어 비순차 수퍼스칼라 프로세서

2.1 멀티코어 비순차 수퍼스칼라 프로세서의 구조

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

본 논문에서는 그림 2와 같이 명령어 윈도우에서 동적 스케줄링에 의하여 명령어가 이슈되는 비순차 수퍼스칼라의 기본형을 이용하였다. 비순차 수퍼스칼라 프로세서는 매 싸이클마다 2개 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 만

* 정 회 원 : 한성대 공대 정보통신공학과 교수
E-mail : jblee@hansung.ac.kr
접수일자 : 2012년 8월 5일
최종완료 : 2012년 9월 20일

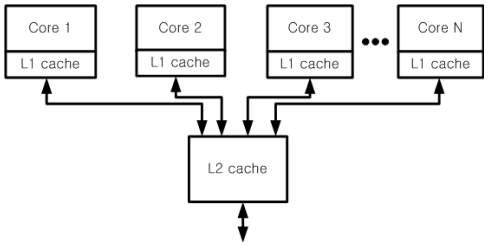

그림 1 멀티코어 수퍼스칼라 프로세서의 구조
Fig. 1 The multi-core superscalar processor architecture

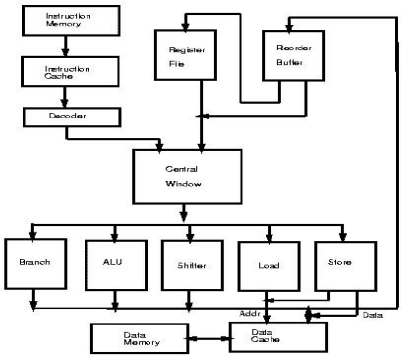

그림 2 수퍼스칼라 코어 프로세서의 구조
Fig. 2 The superscalar core processor

나거나 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 비순차 실행된다. 비순차 실행이 되더라도, 윈도우에서 이슈된 명령어는 재정렬버퍼(Reorder Buffer)에 삽입되어 완료(commit)되므로, 각종 캐쉬 미스, 분기 명령어 예측 미스, 인터럽트 및 트랩에 대비하여 프로그램의 순서를 보존할 수 있다.

2.2 멀티코어 비순차 수퍼스칼라 프로세서 모의실험기

멀티코어 비순차 수퍼스칼라 프로세서의 모의실험은 제 1 단계 명령어 자취의 발생, 제 2단계 명령어 자취에 대한 멀티코어 비순차 수퍼스칼라 프로세서의 실행으로 나누어진다. 제 1단계에서 명령어 자취는 임의의 차수의 멀티코어에 적합하도록 발생되었다. 제 2 단계 멀티코어 비순차 수퍼스칼라 프로세서의 실행은 그림 3에 나타낸 것과 같다. 제 2 단계의 과정을 명령어 인출, 명령어 재명명, 명령어 이슈 및 멀티코어 시뮬레이션으로 나누어 자세하게 기술하면 다음과 같다.

2.2.1 명령어 인출, 재명명 및 이슈

본 모의실험기는 Initialize 함수에서 초기화 작업을 거친 후에, Grouping 함수가 Create_Window 함수를 부르고 이것

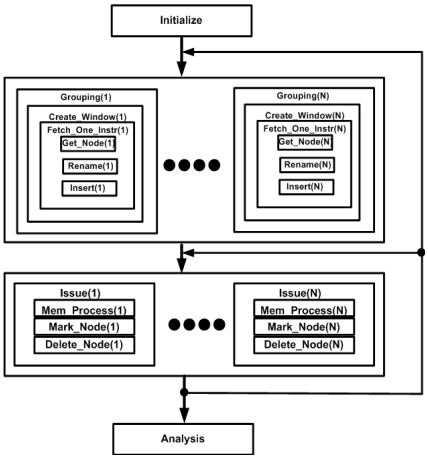

그림 3 멀티코어 프로세서 모의실험기
Fig. 3 The multi-core processor architecture simulator

은 다시 Fetch_One_Instr 함수를 불러서 각 코어는 매 싸이클마다 새로운 명령어를 인출받는다. 한 싸이클에 2개 이상의 명령어를 인출받을 때, 분기명령어를 만나거나 캐쉬 미스가 발생하면 해당 싸이클에서는 그 이상의 인출을 하지 않고 인출을 멈춘다. Get_Node 함수에서 인출한 명령어는 Rename 함수에서 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. 모든 명령어는 인출 초기에 현재 싸이클에 명령어 자체의 지연 싸이클을 더한 값을 타임스탬프로 설정받는다. 이후의 명령어 해독 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 같은 이름을 갖는 레지스터가 존재하고 현재 명령어의 타임스탬프 값보다 소오스 레지스터 1 또는 2의 타임스탬프 값이 더 크다면, 이 명령어는 선행하는 명령어에 종속이 발생한 것이다.

따라서, 이 명령어의 타임스탬프는 해당 소오스 레지스터의 타임스탬프에 명령어 자체의 지연 사이클 수를 더한 값으로 대체된다. 또한, 레지스터 화일에 존재하는 명령어의 목적 레지스터의 타임스탬프도 같은 값으로 정정하여, 추후에 명령어 간 종속으로 인하여 이 목적 레지스터를 소오스 레지스터로 참조하는 명령어에 대하여 올바른 타임스탬프 값을 가질 수 있도록 한다. 레지스터 화일의 타임스탬프 값에 의하여, 멀티코어 프로세서 명령어 간의 종속성을 이용하여 명령어 병렬성을 구하는데 반영된다. 이와 같이 재명명을 거친 명령어는 insert 함수에서 각 코어의 명령어 윈도우에 삽입된다.

Issue 함수로 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. 명령어는 위와 같은 조건에 의하여 비순차 실행되지만, 명령어들은 재정렬버퍼(Reorder Buffer)에 프로그램 원래의 순서대로 삽입되어 완료된다.

2.2.2 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 각 윈도우에서 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 함수를 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.

2.3 모의실험 환경

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

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 아키텍쳐의 사양을 나타낸 것이다. 멀티코어의 개수는 1개, 2개, 4개, 8개, 16개를 대상으로 하였다. 각 코어는 비순차 수퍼스칼라 방식으로 운영되므로, 윈도우의 크기 4, 8, 16에 대하여 매 싸이클 마다 2개, 4개 및 8개의 명령어를 인출한다. 각 코어의 연산유닛은 정수형 유닛, 로드 및 스토어 유닛, 그리고 분기명령어 유닛으로 구성되며, 각각 2개에서 4개로 구성된다.

1차 명령어 캐쉬와 1차 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 캐쉬는 연관매핑(associative-mapping)을 이용하지만 데이터 캐쉬는 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 1차 명령어 캐쉬 및 1차 데이터 캐쉬는 모두 2차 연관도(set associativity)를 갖도록 설계하였다. 1차 캐쉬에서 미스가 발생하였을 때 10 싸이클이 소요된다. 분기 명령어는 2단계 적응형 분기 예측 방식을 적용하였으며, 14비트 전역 히스토리 방식을 채택하였고 분기 어드레스 캐쉬는 2048개의 엔트리를 갖는다 [9].

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 1차 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),		
재정렬버퍼	64		

2.4 모의실험 및 결과

본 논문에서 제안하는 멀티코어 비순차 수퍼스칼라 프로세서와의 성능을 비교하기 위하여, 멀티코어 RISC 프로세서 및 멀티코어 순차 수퍼스칼라 프로세서의 모의실험 결과를 차례대로 살펴보면 다음과 같다.

2.4.1 멀티코어 RISC 프로세서의 모의실험 결과

그림 4에 단위코어를 RISC로 채택한 멀티코어 RISC 프로세서의 모의실험 결과를 나타냈다. 각 단위 코어 프로세서는 한 싸이클에 오직 한 개의 명령어만을 처리할 수 있으며, 수퍼스칼라 단위 프로세서와 동일하게 64 KB의 1차 명령어 캐쉬와 1차 데이터 캐쉬를 갖는다. 단일코어의 경우 성능의 기하평균은 0.31 IPC를 기록하였으며, 2-코어에서 0.66 IPC로 성능이 2.2배가 되었다. 4-코어에서는 1.31 IPC, 8-코어에서 2.50 IPC를 나타냈다. 마지막으로 16-코어의 경우 4.06 IPC의 성능을 가져왔다. 16-코어프로세서는 1-코어 프로세서 대비 13.3배의 성능 향상을 기록하였다.

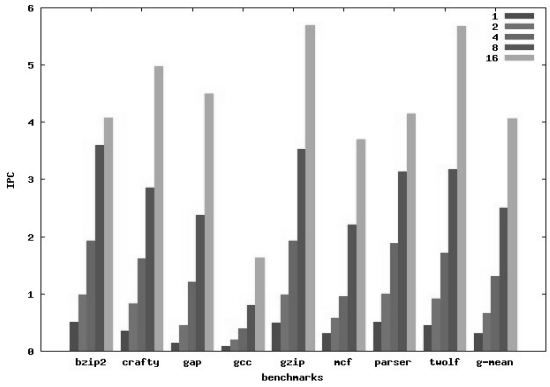

그림 4 1~16 개 멀티코어 RISC 프로세서의 성능 결과

Fig. 4 Performance of multi-core RISC processor architecture for 1 to 16 cores.

2.4.2 멀티코어 순차 수퍼스칼라 프로세서의 모의실험 결과

그림 5부터 그림 7까지 윈도우의 크기가 각각 4, 8, 16일 때 1-코어부터 16-코어로 구성되는 멀티코어 순차 수퍼스칼라 프로세서에 대한 모의실험 결과를 보였다. 멀티코어 순차 수퍼스칼라 프로세서에서는 모든 명령어들을 순차적으로 실행하며, 프로그램의 순서상 나중에 있는 명령어가 실행 가능해도 실행하지 않는다. 윈도우의 크기가 4일 때, 1-코어, 2-코어, 4-코어, 8-코어, 16-코어 순차 수퍼스칼라 프로세서 성능의 기하평균은 각각 0.55, 1.13, 2.03, 3.35, 5.46 IPC를 기록하였다. 윈도우의 크기가 8로 증가하면, 각 성능은 0.66, 1.18, 2.35, 3.78, 6.30 IPC로 향상되었다. 마지막으로 윈도우의 크기가 16일 때, 0.74, 1.39, 2.71, 4.65, 7.73 IPC로 크게 증가하였다. 각 벤치마크 프로그램 별로 순차 수퍼스칼라 코어의 개수가 2배가 되면 성능이 1.8배 향상되었고, 윈도우의 크기가 2배가 되면 성능이 1.2배 증가하였다. 윈도우의 크기가 16인 16-코어 순차 수퍼스칼라 프로세서는 1-코어 순차 수퍼스칼라 프로세서보다 10.5 배의 성능이 향상되었다.

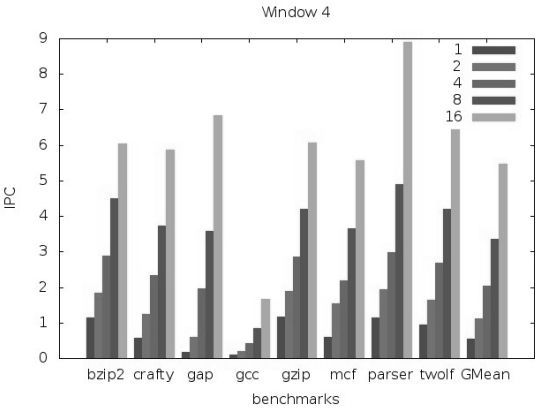

그림 5 윈도우의 크기가 4일 때 1~16 개 멀티코어 순차 수퍼스칼라 프로세서의 성능 결과

Fig. 5 Performance of 1 to 16-core in-order superscalar processor architectures for the window size of 4

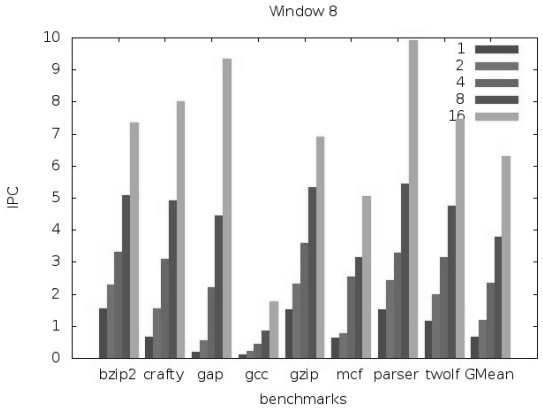

그림 6 윈도우의 크기가 8일 때 1~16 개 멀티코어 순차 수퍼스칼라 프로세서의 성능 결과

Fig. 6 Performance of 1 to 16-core in-order superscalar processor architectures for the window size of 8

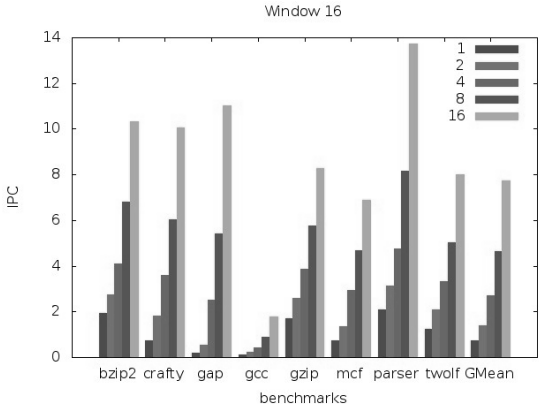

그림 7 윈도우의 크기가 16일 때 1~16 개 멀티코어 순차 수퍼스칼라 프로세서의 성능 결과

Fig. 7 Performance of 1 to 16-core in-order superscalar processor architectures for the window size of 16

2.4.3 멀티코어 비순차 수퍼스칼라 프로세서의 모의실험 결과

그림 8부터 그림 10까지 본 논문에서 제안하는 윈도우의 크기가 4, 8, 16일 때 1-코어부터 16-코어로 구성되는 멀티코어 비순차 수퍼스칼라 프로세서에 대한 모의실험 결과를 각각 보였다. 멀티코어 비순차 수퍼스칼라 프로세서의 경우, 윈도우의 크기가 4일 때, 1-코어, 2-코어, 4-코어, 8-코어, 16-코어 수퍼스칼라 프로세서 성능의 기하평균은 각각 0.67, 1.41, 3.13, 5.65, 10.61 IPC를 기록하였다. 윈도우의 크기가 8로 증가하면, 각 성능은 0.85, 1.76, 3.74, 7.56, 12.97 IPC로 향상되었다. 마지막으로 윈도우의 크기가 16일 때, 성능이 0.89, 1.86, 4.36, 8.56, 15.49 IPC로 크게 증가하였다. 각 벤치마크 프로그램 별로 비순차 수퍼스칼라 코어의 개수가 2 배가 되면 성능이 2배 향상되었으며, 성능의 증가율은 순차 수퍼스칼라의 경우인 1.8배보다 크다. 한편, 윈도우의 크기가 2배가 되면 성능이 1.2 배 증가하였고 이것은 순차 수퍼스칼라와 근사한 수준이다. 윈도우의 크기가 16인 16-코어 비순

차 수퍼스칼라 프로세서는 1-코어 비순차 수퍼스칼라 프로세서보다 17.4 배의 높은 성능을 보였다.

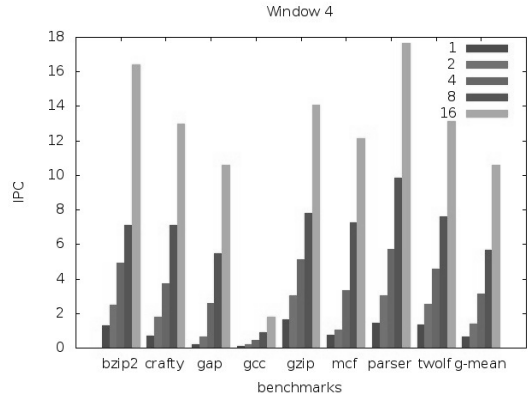

그림 8 윈도우의 크기가 4일 때 1~16개 멀티코어 비순차 수퍼스칼라 프로세서의 성능 결과

Fig. 8 Performance of 1 to 16-core out-of-order superscalar processor architectures for the window size of 4

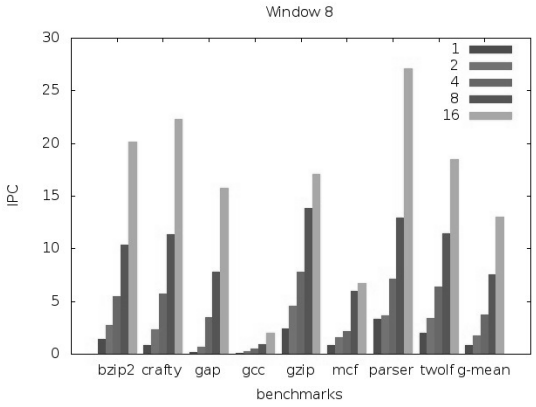

그림 9 윈도우의 크기가 8일 때 1~16 개 멀티코어 비순차 수퍼스칼라 프로세서의 성능 결과

Fig. 9 Performance of 1 to 16-core out-of-order superscalar processor architectures for the window size of 8

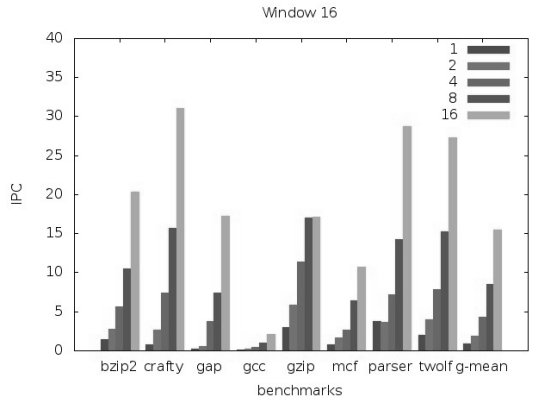

그림 10 윈도우의 크기가 16일 때 1~16 개 멀티코어 비순차 수퍼스칼라 프로세서의 성능 결과

Fig. 10 Performance of 1 to 16-core out-of-order superscalar processor architectures for the window size of 16.

2.4.4 멀티코어 RISC 프로세서, 멀티코어 순차 및 비순차 수퍼스칼라 프로세서의 성능 비교

윈도우의 크기가 16일 때, 본 논문에서 제안하는 멀티코어 비순차 수퍼스칼라 프로세서의 성능을 같은 개수의 멀티코어 RISC 프로세서의 성능과 수평 비교하면 평균 3.2 배의 성능을 가져왔다. 이것은 수퍼스칼라 프로세서를 코어로 채택했을 경우, 단지 ⅓의 코어 개수를 가지고도 멀티코어 RISC 프로세서 이상의 성능을 나타낼 수 있다는 것을 의미한다. 한편, 윈도우의 크기가 16인 16-코어 비순차 수퍼스칼라 프로세서를 1-코어 RISC 프로세서와 비교하면 성능이 무려 50배 향상되었다. 같은 윈도우의 크기와 코어의 개수에서, 멀티코어 비순차 수퍼스칼라 프로세서는 멀티코어 순차 수퍼스칼라 프로세서에 비하여 평균 1.6배의 성능 향상을 기록하였다.

그림 11은 멀티코어 RISC 프로세서, 멀티코어 순차 수퍼스칼라 프로세서, 멀티코어 비순차 수퍼스칼라 프로세서의 3가지 성능의 평균을 윈도우의 크기 4, 8, 16에 대하여 1-코어부터 16-코어까지 그래프로 일목요연하게 표시한 것이다. 그림에서 알 수 있듯이, 멀티코어 RISC 프로세서의 성능이 가장 낮고, 멀티코어 순차 수퍼스칼라 프로세서의 성능이 중간을 차지하며, 멀티코어 비순차 수퍼스칼라 프로세서의 성능이 가장 높다는 것을 알 수 있다. 이 때, 윈도우의 크기와 상관없이 멀티코어 비순차 수퍼스칼라 프로세서의 성능이 멀티코어 순차 수퍼스칼라 프로세서의 성능보다 높은 점이 주목된다. 따라서, 수퍼스칼라 프로세서를 멀티코어 시스템의 단위 코어로 이용하는 경우에는 큰 윈도우의 순차 방식보다는 작은 윈도우의 비순차 방식을 채택하는 것이 유리하다.

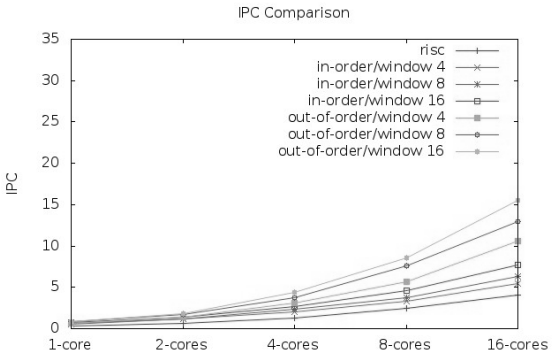

그림 11 각 멀티코어 프로세서의 성능 비교 결과

Fig. 11 Performance comparison of RISC, in-order and out-of-order multi-core superscalar processor architectures

3. 결 론

본 논문에서는 현재 널리 이용되고 있는 멀티코어 프로세서 아키텍처의 단위 코어로서, 비순차 실행 방식의 수퍼스칼라 프로세서의 채택을 제안하였다. 또한 8 개의 SPEC 2000 벤치마크에 대하여 윈도우의 크기 4부터 16, 코어의 개수 1

개부터 16 개까지인 멀티코어 프로세서에 대한 성능을 광범위한 모의실험을 통하여 측정하였다. 이 때, 제안하는 멀티코어 비순차 수퍼스칼라 프로세서의 성능을 멀티코어 RISC 프로세서 및 멀티코어 순차 수퍼스칼라 프로세서의 성능과 비교하였다. 그 결과, 코어의 개수가 같을 때, 멀티코어 비순차 수퍼스칼라 프로세서는 멀티코어 RISC 프로세서에 비하여 평균 3.2 배, 멀티코어 순차 수퍼스칼라 프로세서보다 평균 1.6배의 성능 향상을 기록하였다.

추후의 연구과제로, 각 비순차 수퍼스칼라 코어에 대하여 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. 또한, 동질 코어(homogeneous core)가 아닌 비동질 코어(heterogeneous core)를 채택하는 비대칭 칩 멀티프로세서(asymmetric chip multiprocessor) 구조에 대한 연구를 들 수 있다. 더 나아가, 최신 경향인 코어 수 100개 이상의 매니코어(many-core) 아키텍처에 대하여 연구를 수행할 예정이다.

감사의 글

본 연구는 한성대학교 교내연구장려금 지원과제 임.

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] David E. Culler and Jaswinder Pal Singh, "Parallel Computer Architecture", Morgan Kauffmann Publishers, Inc. Aug. 1998.

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

[4] Theo Ungerer, Borut Robic, and Jurij 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] Davy Genbrugge and Lieven Eeckhout, “Chip Multiprocessor Design Space Exploration through Statistical Simulation”, IEEE Transactions on Computers 58(12), pp.1668-1681, Dec. 2009.

[7] Alejandro Rico, Alejandro Duran. Felipe Cabarcas, Yoav Etsion, Alex Ramirex, and Mateo 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.

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