Journal of Korea Multimedia Society Vol. 16, No. 2, February 2013(pp. 198-205)
http://dx.doi.org/10.9717/kmms.2013.16.2.198

다중블럭을 실행하는 멀티코어 비순차 수퍼스칼라 프로세서의 성능 분석

이 종 복†

요 약

본 논문에서는 다중블럭 실행을 이용하는 멀티코어 비순차 수퍼스칼라 프로세서 아키텍쳐의 성능을 분석하였다. 이것을 위하여 SPEC 2000 벤치마크를 입력으로 하며, 윈도우 크기가 32와 64이고 1개에서 4개의 다중블럭을 실행하는 멀티코어 비순차 수퍼스칼라 프로세서에 대하여 1 코어에서 16 코어까지 광범위한 모의실험을 수행하였다. 모의실험 결과, 4개의 다중블럭을 실행하는 멀티코어 비순차 수퍼스칼라 프로세서는 같은 사양에서 단일 블럭을 실행할 때보다 평균 22.0%의 성능 향상을 가져왔다.

Performance Analysis of Multicore Out-of-Order Superscalar Processor with Multiple Basic Block Execution

Jong Bok Lee†

ABSTRACT

In this paper, the performance of multicore processor architecture is analyzed which utilizes out-of-order superscalar processor core using multiple basic block execution. Using SPEC 2000 benchmarks as input, the trace-driven simulation has been performed for the out-of-order superscalar processor with the window size from 32 to 64 and the number of cores between 1 and 16, exploiting multiple basic block execution from 1 to 4 extensively. As a result, the multicore out-of-order superscalar processor with 4 basic block execution achieves 22.0 % average performance increase over the same architecture with the single basic block execution.

Key words: Multicore Processor(멀티코어 프로세서), Multiple Basic Block Execution(다중블럭실행)

1. 서 론

최근 30년간 마이크로 프로세서 연구는 단일코어 프로세서를 더욱 빠르게 실행하도록 설계하는 것을 목표로 하였으며, 수퍼스칼라 프로세서가 그 대표적인 예이다[1]. 그러나, 이러한 수퍼스칼라 프로세서는 하드웨어의 복잡도를 높여도 성능이 그 이상 증가하지 않는 한계에 이르렀다. 이것을 해결하기 위하여 첫 번째 방안으로 수퍼스칼라 프로세서의 비순차 실행 및 다중분기 예측법이 이용되었다[2,3]. 다중분기 예측법은 한 싸이클에 두 개 이상의 분기명령어에 대한 예측을 수행하여 두 개 이상의 기본블럭을 인출한다. 따라서, 명령어가 스케줄링되는 범위가 넓어지므로 명령어의 병렬성을 높일 수 있다. 두 번째 방안으로 멀티코어 프로세서가 대두되었으며, 코어 간의 명령어 수준 병렬성을 확보하여 코어의 개수를 늘릴수록 성능 향상을 얻을 수 있으므로, 현재 널리 이용되고 있다[4,5].

※ 교신저자(Corresponding Author) : 이종복, 주소 : 서울시 성북구 삼선동 2가 389번지, 전화 : 02)760-4497, FAX : 02)760-4435, E-mail : jblee@hansung.ac.kr
접수일 : 2012년 8월 25일, 수정일 : 2012년 11월 12일
완료일 : 2012년 12월 6일
† 정회원, 한성대학교 정보통신공학과
※ 본 연구는 한성대학교 교내연구장려금 지원과제임.

이러한 멀티코어 프로세서는 여러 개의 코어를 이용한 병렬처리로 성능을 끌어올릴 수 있으므로, 시스템을 구성하는 단위 코어로 한 싸이클에 최대 한 개의 명령어만을 처리할 수 있는 간단한 RISC가 주로 쓰이고 있다. 그러나, 코어의 개수가 32개 미만인 소규모 멀티코어 프로세서 시스템에서, 그 성능의 극대화를 위하여 RISC보다 훨씬 성능이 뛰어난 고성능 수퍼스칼라 프로세서를 단위 코어로 활용한다면, 하드웨어 복잡도가 그리 높지 않으면서도 적은 개수의 코어를 가지고도 전체 시스템의 성능이 더욱 향상될 수 있다.

이 때, 수퍼스칼라 프로세서는 순차 방식 (in-order) 또는 비순차 방식 (out-of-order)으로 실행할 수 있다. 비순차 방식의 수퍼스칼라의 하드웨어 구조가 순차 방식보다 복잡하지만, 순차 방식보다 훨씬 높은 성능을 낼 수가 있다. 이 때, 각 코어에서 실행할 수 있는 태스크는 1개 또는 여러 개의 기본블럭으로 구성될 수 있으며, 기본 블럭의 개수가 멀티코어 프로세서의 성능에 큰 영향을 끼친다[6]. 그러나, 태스크를 구성하는 기본블럭의 개수가 멀티코어 프로세서의 성능에 미치는 영향에 대하여 충분히 연구된 관련 문헌을 찾아볼 수 없으므로, 이것에 대한 연구가 필요하다.

본 논문에서는 각 코어에서 다중블럭 실행이 멀티코어 비순차 수퍼스칼라 프로세서 아키텍처에 미치는 영향을 분석하였다. 이것을 위하여, SPEC 2000 벤치마크를 입력으로, 기본블럭의 수가 1에서 4, 윈도우의 크기가 32와 64, 코어의 개수가 1 개에서 16개인 멀티코어 비순차 수퍼스칼라 프로세서에 대하여 광범위한 모의실험을 수행하여 그 성능을 측정하고 분석하였다.

본 논문은 다음과 같이 구성된다. 2장에서 멀티코어 비순차 수퍼스칼라 및 다중블럭의 실행에 대하여 고찰하고, 3장에서 모의실험기에 대하여 설명한다.

4장에서 모의실험 환경에 대하여 살펴보고, 5장에서 모의실험 결과를 분석한다. 마지막으로, 6장에서 결론을 맺는다.

2. 멀티코어 비순차 수퍼스칼라 및 다중블럭의 실행

2.1 멀티코어 비순차 수퍼스칼라 프로세서의 구조

그림 1은 N 차 멀티코어 비순차 수퍼스칼라 프로세서의 구조를 나타낸 것이다. 이 시스템은 1개부터 N 개까지의 코어로 구성되는데, 자체적으로 1차 명령어 및 1차 데이터 캐쉬를 가지며, 메인 메모리와 연결되는 공통의 2차 캐쉬를 공유한다.

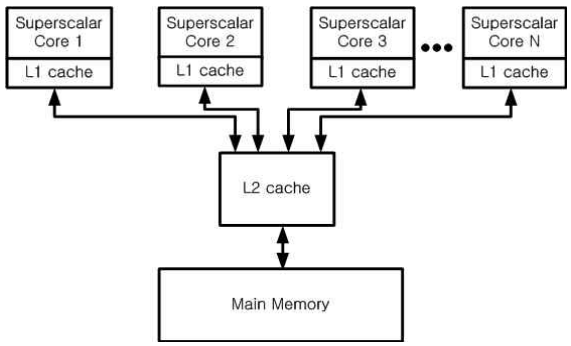


그림 1. 멀티코어 비순차 수퍼스칼라 프로세서의 구조

각 코어에 설치된 1차 데이터 캐쉬의 일관성 (cache-coherency)을 위하여 MESI 프로토콜을 이용한다. MESI 프로토콜에 의하여, 어떤 코어에 소속된 1차 데이터 캐쉬의 데이터가 다른 코어에 의하여 쓰기 작업이 되었을 때, 그 데이터는 무효화 (invalidate)된다.

본 논문에서는 단위 코어로서 명령어 윈도우에서 동적 스케줄링에 의하여 명령어가 이슈되는 비순차 수퍼스칼라를 기본으로 하였다. 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, 다중블럭 실행에 의하여 한 싸이클에 인출 가능한 블럭의 개수를 초과하거나 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. 명령어가 비순차 실행이 되더라도, 윈도우에서 이슈된 명령어는 재정렬버퍼(Reorder Buffer)에 삽입되어 완료(commit)되므로, 각종 캐쉬 미스, 태스크 예측 미스, 인터럽트 및 트랩에 대비하여 프로그램의 원순서를 보존할 수 있다.

2.2 다중블럭의 실행

그림 2는 다중블럭을 실행하는 멀티코어 프로세

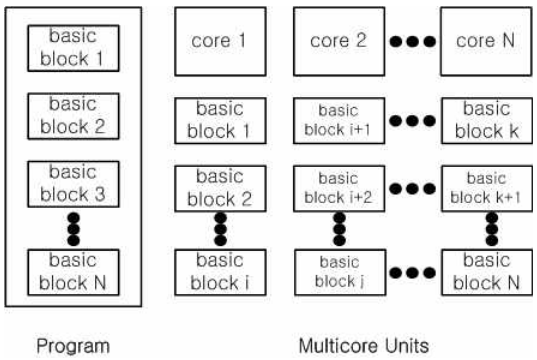


그림 2. 멀티코어 프로세서 단위 코어에서의 다중블럭의 실행

서를 나타낸다. 임의의 프로그램은 동적 명령어의 흐름 basic block 1, basic block 2, ..., basic block N으로 구성된다. 이 때 각 코어마다 최대 i 개의 다중 블럭이 할당되어 실행된다. 각 코어 내에서 명령어들은 비순차 실행되지만 순차 완료되며, 각 코어에 대해서도 순차완료된다. 또한, 동적으로 코어 내부 및 코어 간 레지스터 종속과 메모리 모호성 제거를 검사하여 올바른 실행을 할 수 있다. 단일블럭 실행은 한 싸이클에 한 개의 기본블럭을 각 코어에 인출하여 실행한다. 반면에 다중블럭 실행은 한 싸이클에 두 개 이상의 기본블럭을 각 코어에 인출하여 실행한다. 이렇게 함으로써 명령어가 스케줄링되는 범위가 확대되어 명령어 수준 병렬성이 향상된다.

코어 당 다중블럭 실행을 위한 태스크 할당 알고리즘으로 채택한 방법은 2단계 적응형 분기 예측법을 태스크 예측에 응용한 것이다[7]. 2단계 적응형 태스크 예측법은 태스크 히스토리 레지스터에 최근의 태스크 경로 결과를 기록하고, 본 레지스터의 값으로 태스크 패턴 히스토리 테이블을 접근하여 2 비트 포화 카운터의 값에 따라 태스크 경로의 채택 여부를 예측한다. 이 때, 각 태스크의 시작 어드레스를 기록하기 위하여 태스크 어드레스 캐쉬를 운용한다.

일반적으로, M개의 다중 태스크 예측을 수행할 때, 첫 번째, 두 번째, ..., M번째 태스크에 대하여, 태스크 히스토리 레지스터의 k 비트, k-1 비트, ..., k-M+1 비트로 태스크 패턴 히스토리 테이블을 각각 인덱싱하여 다음 태스크의 시작 어드레스를 예측한다.

3. 멀티코어 비순차 수퍼스칼라 프로세서 모의실험기

멀티코어 비순차 수퍼스칼라 프로세서의 모의실

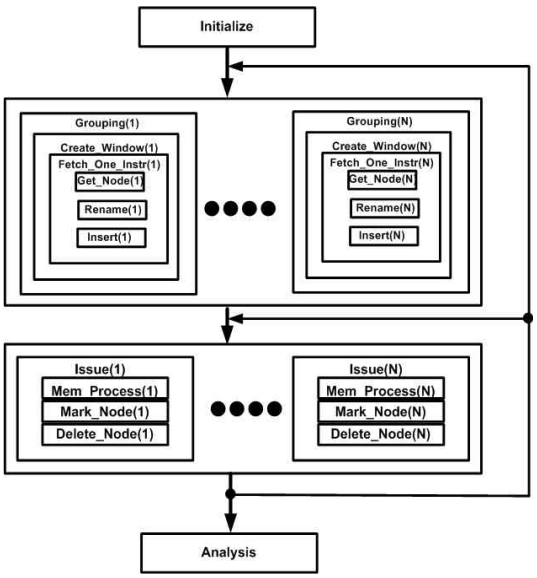


그림 3. 멀티코어 프로세서 모의실험기의 순서도

험기는 실행위주 방식(execution-driven)과 명령어 자취 방식(trace-driven)으로 나뉠 수 있다[8]. 본 논문에서는 실행 위주 방식보다 속도가 빠르고 효율적인 명령어 자취 방식을 채택하였으며, Intel Core i5 데스크탑 컴퓨터의 리눅스 3.3.4 운영체제하에서 gcc 4.7.2를 이용하여 자체 개발하였다. 본 연구에서 개발한 모의실험기는 미리 발생시킨 명령어 자취를 입력으로 하여 수행되며, 그 구성은 그림 3에 나타낸 것과 같다. 이 과정을 단위코어 내에서 발생하는 명령어의 인출, 명령어 재명명 및 명령어 이슈 및 멀티코어 시뮬레이션으로 나누어 자세하게 기술하면 다음과 같다.

3.1 단위코어에서의 명령어의 인출, 재명명 및 이슈

본 모의실험기는 Initialize 블럭에서 초기화 작업을 거친 후에 Grouping 블럭이 Create_Window 블럭을 부르고, 이것은 다시 Fetch_One_Instr 블럭을 호출하여, 각 코어는 매 싸이클마다 새로운 명령어를 인출받는다. 한 싸이클에 2개 이상의 명령어를 인출받을 때, 다중블럭의 개수가 초과하거나 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. 모든 명령어는 인출 초기에 현재 싸이클에 명령어 자체의 지연 싸이클을 더한 값을 타임스탬프로 설정 받는다. 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 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. 명령어는 위와 같은 조건에 의하여 비순차 실행되지만, 명령어들은 재정렬버퍼(Reorder Buffer)에 프로그램 원래의 순서대로 삽입되어 완료된다.

3.2 멀티코어 시뮬레이션

모든 N개의 멀티코어에 대하여 해당 코어의 윈도우 공간에 Grouping 블럭을 이용하여 명령어를 인출해서 채우고, 각 코어에 대하여 Issue 블럭으로 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 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.

4. 모의실험 환경

표 1은 모의실험에 이용된 SPEC 2000 벤치마크 프로그램이다. SimpleScalar를 통하여 SPEC 벤치마크 프로그램을 컴파일하고 수행하여 MIPS IV 10억 개의 명령어 자취를 임의의 차수의 멀티코어에 적합하도록 발생시켰다[9]. 이렇게 해서 얻은 명령어 자취를 본 연구에서 개발한 모의실험기에 입력하여 성능을 측정하였다. 명령어 자취의 발생 및 모의실험도 모의실험기와 같이 리눅스 3.3.4 운영체제하의 Intel Core i5에서 실행되었다.

표 2는 모의실험에 이용된 멀티코어 수퍼스칼라 프로세서 아키텍쳐의 하드웨어 사양을 나타낸 것이다. 멀티코어의 개수는 1개, 2개, 4개, 8개, 16개를 대상으로 하였으며, 각 코어는 비순차 수퍼스칼라 방식으로 운영된다. 윈도우의 크기가 작을 때는 다중블럭 실행을 충분히 활용할 수 없으므로, 윈도우의 크기를 32와 64로 설정하였다. 매 싸이클 마다 각 윈도우의 크기에 따라 8개 또는 16개의 명령어를 프로세서로 인출한다. 각 코어의 연산유닛은 정수형 유닛, 분기 유닛, 로드 및 스토어 유닛으로 구성되며, 각각 2개에서 16개의 범위로 구성된다.

비순차 실행에서 메모리 관련 명령어인 로드와 스토어가 30% 이상을 차지하며 프로그램의 순차성을

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

표 2. 모의실험에 이용된 멀티코어 수퍼스칼라 프로세서 아키텍쳐 하드웨어의 사양

항 목	값	
멀티코어 수	1, 2, 4, 8, 16	
다중블럭 수	1, 2, 3, 4	
1차 명령어 캐쉬 및 1차 데이터 캐쉬	64KB, 2 차 연관, 16 B 미스 페날티 10 싸이클	
명령어 윈도우의 크기	32	64
인출율, 이슈율, 퇴거율	8	16
연산유닛 사양	ALU(8), load(2), store(1), branch(1-4)	ALU(16), load(4), store(2), branch(1-4)
태스크 어드레스 캐쉬	2 K 엔트리	
다중블럭 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	32	64

보존해야하므로 그 처리가 중요하다. 로드와 스토어가 동시에 스케줄링되는 경우는 총 4 가지의 조합이 발생하는데, 이 때 후속 명령어의 실행 여부가 비순차 수퍼스칼라 프로세서의 성능에 큰 영향을 준다. 로드-로드 쌍의 경우에 데이터 값을 변형시키지 않으므로 후속 로드는 동시 실행 가능하며, 로드-스토어의 쌍이면서 유효 주소가 일치할 때는 선행하는 로드가 먼저 수행되어야 하므로, 후속 스토어를 로드와 동시에 실행시키지 않는다. 스토어-로드 쌍일 때 로드는 스토어 값을 스토어 버퍼(store buffer)에서 참조하여 동시 실행이 가능하다. 마지막으로 스토어-스토어 쌍은 유효주소가 일치할 때 후속 스토어가 데이터를 쓰면 안되므로 이를 실행시키지 않는다.

1차 명령어 캐쉬와 1차 데이터 캐쉬의 용량이 작으면 멀티코어 프로세서의 성능을 충분히 끌어올릴 수 없으므로, 명령어 캐쉬와 데이터 캐쉬는 64 KB의 용량을 갖도록 설정하였다. 기존의 단일 코어 프로세서의 경우, 명령어 캐쉬는 연관매핑 (associative-mapping)을 이용하지만 데이터 캐쉬는 직접매핑 (direct-mapping)을 이용하더라도 높은 캐쉬 히트율을 얻을 수 있었다. 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(2-way set associative)로 설계하였다.

단, 모의실험의 복잡도를 줄이기 위하여 DRAM으로 구성되는 메인 메모리를 모델링하지 않았기 때문에, 2차 통합 캐쉬 (unified cache)를 별도로 모델링하지 않고 100% 히트가 난다고 가정하였다. 다중블럭 태스크 예측기는 2단계 적응형 태스크 예측 방식을 적용하였으며, 14 비트 전역 히스토리 방식을 채택하였고 태스크 어드레스 캐쉬는 2048개의 엔트리를 갖는다.

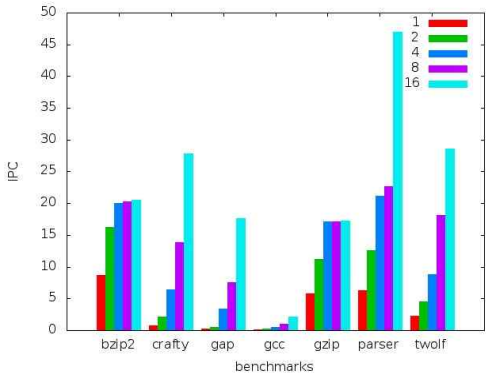
5. 모의실험 및 결과

그림 4(a)부터 4(h)에 본 논문에서 분석하는 다중블럭 실행을 이용하는 멀티코어의 비순차 수퍼스칼라 프로세서에 대한 모의실험 결과를 각각 나타냈다.

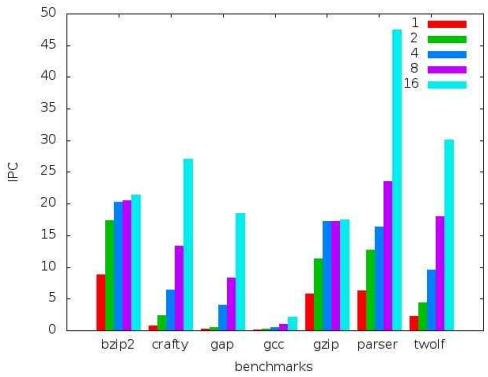
이 때, 윈도우의 크기는 각각 32, 64이고, 멀티코어는 1개부터 16개의 범위를 갖는다. 이론적으로 다중블럭 실행은 N개의 기본블럭에 대하여 실행할 수 있으나, 본 실험에서는 멀티코어 프로세서에 활용하는 만큼 하드웨어의 복잡도를 줄이기 위하여 한 싸이클에 1개에서 4개의 기본블럭을 실행하였다.

모의실험 결과를 각 벤치마크 프로그램에 대하여 고찰하면, 멀티코어 비순차 수퍼스칼라 프로세서의 성능은 명령어 캐쉬, 데이터 캐쉬, 태스크 어드레스 캐쉬에 의한 미스 및 다중 블럭 예측 오류에 의하여 감소하였다. 이 때, 명령어 캐쉬와 데이터 캐쉬 미스에 의한 성능의 손실이 가장 컸다. *Parser*는 명령어 캐쉬 및 데이터 캐쉬의 히트율이 높기 때문에 성능의 손실이 적어서 최고의 성능을 나타냈으며, 이와 반대로 G*cc*는 명령어 캐쉬 미스와 데이터 캐쉬 미스에 의하여 각각 96%와 47%의 성능 손실을 가져와 최저의 성능을 기록하였다.

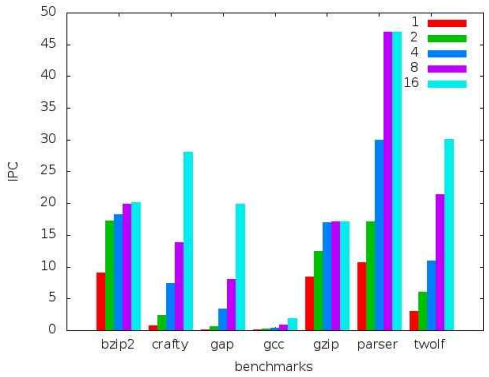
코어의 개수가 2배가 될 때마다 명령어의 병렬처리가 증가하여 성능이 평균 1.8배 증가하였다. 이 때, 성능은 1-코어에서 2-코어로 증가할 때 최대 2.3배를 기록하였고 8-코어에서 16-코어로 증가할 때 최저 1.4배를 기록하여, 코어의 개수가 증가할 수록 성



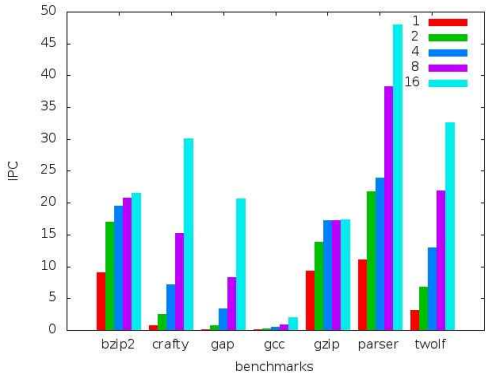
(a) 윈도우=32, 블럭 수=1



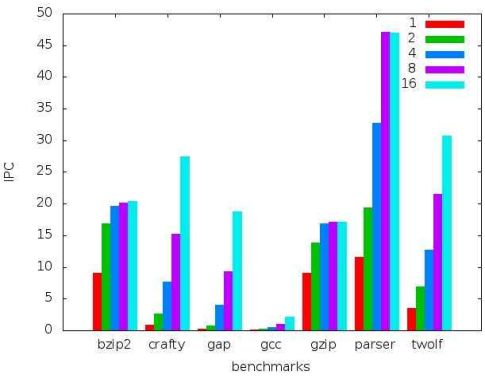
(b) 윈도우=64, 블럭 수=1



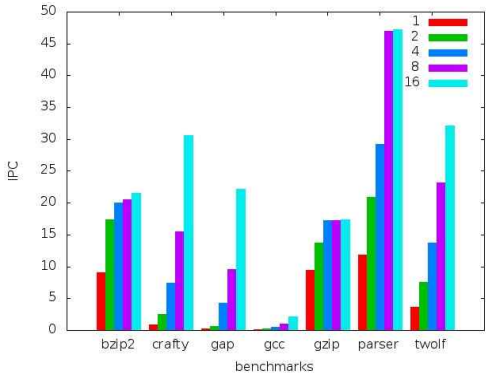
(c) 윈도우=32, 블럭 수=2



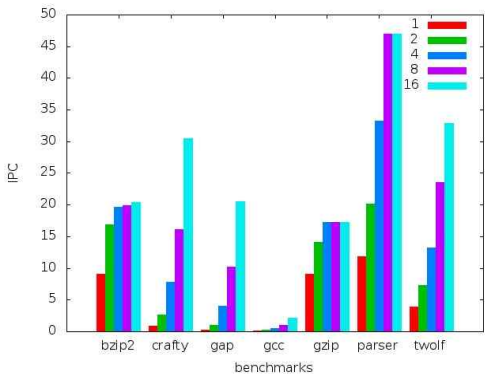
(d) 윈도우=64, 블럭 수=2



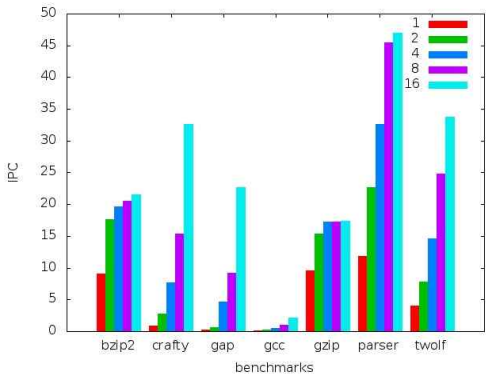
(e) 윈도우=32, 블럭 수=3



(f) 윈도우=64, 블럭 수=3



(g) 윈도우=32, 블럭 수=4



(h) 윈도우=64, 블럭 수=4

그림 4. 다중블럭을 실행하는 멀티코어 비순차 수퍼스칼라 프로세서의 모의실험 결과

능의 향상률은 둔화되었다. 또한, 윈도우의 크기 32보다 64의 경우에 성능이 더높은 것을 알 수 있다.

다중블럭 실행에 의하여 블럭의 개수가 증가할 수록 역시 성능이 향상되었다. 이것은 다중블럭 실행으로 인하여 각 코어 단위로 명령어 수준 병렬성을 더욱 광범위하게 추출할 수 있기 때문이다. 성능의 향상률은 윈도우의 크기가 32일 때, 1-블럭에서 2-블럭으로 증가할 때 평균 7.6%, 3-블럭에서 4-블럭으로 증가할 때 평균 2.7%를 기록하였다. 윈도우의 크기가 64일 때는 그 값이 더욱 증가하여 1-블럭에서 2-블럭으로 증가할 때 평균 13.7%, 3-블럭에서 4-블럭으로 증가할 때 평균 3.0%를 기록하였다. 다중블럭의 개수가 한 개 증가할 수록 성능은 평균 5.6%씩 증가하지만, 성능의 향상률은 역시 둔화되는 것을 알 수 있다. 또한, 윈도우의 크기가 클 수록 다중블럭 실행에 의한 성능이 높다는 것을 알 수 있다.

멀티코어 비순차 수퍼스칼라 프로세서에서 4-블럭에 대한 단일블럭의 성능을 비교하면, 코어의 개수에 따라서 최저 5.3%에서 최고 34.9%를 기록하여, 평균 22.0%의 성능향상을 가져왔다.

한편, 4개의 다중블럭 실행을 이용하는 1-코어 비순차 수퍼스칼라 프로세서를 기준으로 다른 개수의 코어와 성능을 비교하였을 때, 2-코어는 2.2배, 4-코어는 4.5배, 8-코어는 7.1배, 마지막으로 16-코어는 10.3배의 성능향상을 가져왔다. 위 결과는 4-코어 때 약 4배, 8-코어 때 약 8배, 16-코어 때 10배~15배의 성능향상을 기록하는 기존의 멀티코어 프로세서 모의실험기를 통하여 얻은 결과와 근접하다[10,11].

위 모의실험결과를 종합하면, 멀티코어 비순차 수퍼스칼라 프로세서에서 코어의 개수가 증가하고, 윈도우의 크기가 크며, 다중블럭을 실행할 수록 성능의 향상을 가져오는 것을 알 수 있다.

6. 결 론

본 논문에서는 현재 널리 이용되고 있는 멀티코어 프로세서 아키텍처의 단위 코어로서 다중블럭 실행을 이용하는 비순차 수퍼스칼라 프로세서의 성능을 분석하였다. 이것을 위하여, 윈도우의 크기가 32와 64이고 코어의 개수가 1에서 16의 범위에 있는 멀티코어 비순차 수퍼스칼라 프로세서에서 최대 4개의 다중블럭에 대하여 SPEC 2000 벤치마크를 입력으로 하여 광범위한 모의실험을 통하여 그 성능을 측정하였다.

그 결과, 본 멀티코어 프로세서는 코어의 개수가 증가할 수록, 윈도우의 크기가 클 수록, 그리고 다중블럭의 개수가 증가할 수록 성능이 증가하였다. 윈도우의 크기가 32와 64인 비순차 수퍼스칼라 프로세서에서 다중블럭을 실행할 때, 블럭의 개수가 한 개 증가할 때마다 평균 5.6%의 성능 향상을 가져와서, 4개의 다중블럭의 실행은 단일블럭의 경우보다 성능이 평균 22.0% 개선되었다.

추후로, 동질 코어(homogeneous core)가 아닌 비동질 코어(heterogeneous core)를 채택하는 비대칭 칩 멀티프로세서(asymmetric chip multiprocessor) 구조 및 100개 이상의 코어로 구성되는 매니코어(many-core) 아키텍쳐에 대한 연구가 필요하다.

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, 1994.

[2] T. Yeh, D. T. Marr, and Y. N. Patt, "Increasing the Instruction Fetch Rate via Multiple Branch Prediction and a Branch Address Cache," *The 7th International Conference on Supercomputing*, pp. 67-76, 1993.

[3] 이종복, "대형 윈도우에서 다중 분기 예측법을 이용하는 수퍼스칼라 프로세서의 프로화일링 성능 모델," 대한전기학회논문지, 제58권, 제7호, pp. 1443-1449, 2009. 7.

[4] T. Ungerer, B. Robic, and J. Silk, "Multi-threaded Processors," *The Computer Journal*, Vol. 45, No. 3, pp. 320-348, 2002.

[5] 박상수, "다중 멀티미디어 스트리밍을 위한 멀티코어 시스템 기반의 실시간 스케줄링 기법," 한국멀티미디어학회논문지, 제14권, 제11호, pp. 1478-1490. 2011년 11월.

[6] T. N. Vijaykumar and G. S. Sohi, "Task Selection for a Multiscalar Processor," *31st*

International Symposium on Microarchitecture, Dec. pp. 81-92, 1998.

[7] T-Y. Yeh and Y. N. Patt, "Alternative Implementations of Two-Level Adaptive Branch Prediction," *Proceedings of the 19th International Symposium on Computer Architecture*, pp. 124-134, 1992.

[8] A. Rico, A. Duran. F. Cabarcas, Y. Etsion, A. Ramirex, and M. Valero, "Trace-Driven Simulation of Multithreaded Applications," *ISPASS*, pp. 87-96, Apr. 2011.

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

[10] S. Biswas, D. Franklin, A. Savage, R. Dixon, T. Sherwood, and F. T. Chong, "Multi-Execution : Multicore Caching for Data-Similar Executions," *Proceedings of the 36th Annual International Symposium on Computer Architecture*, pp. 164-173, 2009.

[11] M. Monchiero, J. H. Ahn, A. Falcon, D. Ortega, and P. Faraboschi, "How to Simulate 1000 Cores," *ACM SIGARCH Computer Architecture News Archive*, Vol. 37, Issue 2, pp. 10-19, 2009.

이 종 복

1988년 2월 서울대학교 컴퓨터공학과 졸업(공학사)
1990년 2월 서울대학교 대학원 컴퓨터공학과 졸업(공학석사)
1998년 2월 서울대학교 대학원 전기공학부 졸업(공학박사)
2000년 3월~현재 한성대학교 정보통신공학과 교수
관심분야 : 마이크로 프로세서, 멀티코어 프로세서 등