논문 2007-44SD-2-9

분기 동시 수행을 이용한 단일 칩 멀티프로세서의 성능 개선

(Performance Improvement of Single Chip Multiprocessor using Concurrent Branch Execution)

이 승 렬*, 김 준 식*, 최 재 혁*, 최 상 방**

(Seung Ryul Lee, Jun Shik Kim, Jae Hyeok Choi, and Sang Bang Choi)

요 약

프로세서 성능향상에 일반적으로 이용되어 오던 명령어 수준의 병렬성은 이제 그 한계를 드러내고 있다. 명령어 수준의 병렬성을 이용하는데 장애가 되는 요인 중에 하나는 분기문에 의한 제어 흐름의 변화이다. 단일 칩 멀티프로세서는 쓰레드 수준의 병렬성을 이용하는 프로세서이다. 그러나 다중 쓰레드를 고려하지 않고 작성된 프로그램을 수행하는 경우에는 단일 칩 멀티프로세서의 성능을 최대한 사용할 수 없는 단점이 있다. 이와 같은 두 가지 성능 저하 요인을 극복하기 위해 본 논문에서는 다중 경로 수행 기법을 단일 칩 멀티프로세서에 적용한 분기 동시 수행 기법을 제안한다. 제안된 방법에서는 유휴 중인 프로세서를 이용하여 조건 분기의 두 흐름을 모두 수행하게 한다. 이를 통하여 분기문에 의한 제어 흐름이 끊기는 것을 막고 유휴 시간을 줄여서 프로세서의 효율을 높일 수 있다. 시뮬레이션을 통하여 본 논문에서 제시한 분기 동시 수행의 효과를 분석한 결과 분기 동시 수행으로 약 20%의 유휴 시간이 감소하였고, 분기 예측 성공률은 최대 10% 향상 되었다. 전체적으로 일반적인 단일 칩 멀티프로세서에 비해 최대 39%의 성능 향상을 이루었고, 슈퍼스칼라 프로세서에 비해 최대 27%의 성능 향상을 이루었다.

Abstract

The instruction level parallelism, which has been used to improve the performance of processors, expose its limit. The change of a control flow by a branch miss prediction is one of the obstacles that restrict the instruction level parallelism. The single chip multiprocessors have been developed to utilize the thread level parallelism. However, we could not use the maximum performance of the single chip multiprocessor in case of executing the coded programs without considering the multi-thread. In order to overcome the two performance degradation factors, in this paper, we suggest the concurrent branch execution method that applies to the multi-path execution method at a single chip multiprocessor. We executes all two flows of the conditional branch using the idle core processor. Through this, we can improve the processor's efficiency with blocking the control flow termination by the branch instruction and reducing the idle time. We analyze the effects of concurrent branch execution proposed in this paper through the simulation. As a result of that, concurrent branch execution reduces about 20% of idle time and improves the maximum 10% of the branch prediction accuracy. We show that our scheme improves the overall performance of maximum 39% compared to the normal single chip multiprocessor and maximum 27% compared to the superscalar processor.

Keywords: Computer Architecture, Multiprocessors, Single Chip Multiprocessors, multipath execution

I. 서 론

지난 십여년간 프로세서의 성능 향상을 이끌어 온 기술은 좀 더 많은 명령어 수준 병렬성(Instruction Level Parallelism)을 찾고 이용하기 위한 기술들이었다[1][2]. 이러한 강력한 기술들을 모두 구현하여 현재 프로세서 시장을 점유하고 있는 아키텍쳐가 바로 슈퍼스칼라 아키텍쳐이다. 여전히 슈퍼스칼라 아키텍쳐에 기반한 더욱 진보된 기술들에 대한 연구가 이루어지고 있지만 몇 가지 명령어 수준 병렬성의 한계에서 오는 문제점들이 대두되고 있다. 그 중 대표적인 것들은 분기 예측 실패와 2차 캐쉬 실패로 인한 성능 저하, 명령어 수준의 병렬성의 특성상 제한된 하드웨어 자원으로 이용할 수 있

* 학생회원, ** 정회원, 인하대학교 전자공학과
(Department of Electronic Eng., INHA University)
접수일자: 2006년6월19일, 수정완료일: 2007년1월25일

는 명령어 수준의 병렬성은 그 양에 한계가 있다는 점을 들 수 있다[3].

이러한 명령어 수준 병렬성을 이용하는 성능 향상 방법의 한계를 다른 관점의 병렬성을 이용하는 방법으로 극복하고자 하는 노력들이 있어왔다. 기존의 명령어 수준의 병렬성 외에도 쓰레드 수준의 병렬성까지 이용하는 성능 향상 기법들이 많이 연구되고 있다. 쓰레드 수준의 병렬성을 이용하는 아키텍쳐로는 단일 칩 멀티프로세서(Single Chip Multi-Processor, CMPs)와 동시 쓰레드 프로세서(Simultaneous Multithreading Processor, SMT)가 대표적이다[4][5][6]. 단일 칩 멀티프로세서 구조는 여러 개의 완전한 프로세서를 하나의 칩에 집적시킨 멀티프로세서의 한 종류이다[4]. 단일 칩 멀티프로세서는 쓰레드 수준 병렬성을 활용하여 슈퍼스칼라 프로세서를 능가하는 성능 향상을 얻을 수 있다. 그러나 응용의 특성상 단일 쓰레드로 구현된 프로그램의 경우 단일 칩 멀티프로세서의 이득을 전혀 볼 수 없거나 오히려 동일한 수준의 슈퍼스칼라 프로세서를 밑도는 성능을 보이기도 한다.

이러한 단일 칩 멀티프로세서의 단점을 보완하기 위해 여러 가지 방법들이 제안되었다. 데이터 모험 수행(data speculation)을 이용하는 방법과 유휴 상태의 프로세서를 이용하여 선인출(prefetching)을 수행함으로써 효율을 높이는 방법이 있다[7][8]. 좀 더 효율적인 다중 쓰레드 코드를 만들어내기 위한 컴파일러 기술의 연구도 있다. 본 논문에서는 다중 경로 실행 기법[9]을 단일 칩 멀티프로세서에 적용시킨 분기 동시 수행을 제안한다. 단일 쓰레드로 구성된 프로그램을 수행할 때 낭비되는 프로세서를 이용하여 다른 경로의 프로그램 코드를 실행함으로써 단일 칩 멀티프로세서의 효율을 높이고 분기 예측 실패로 인한 손실을 제거하였다. 하나의 프로세서가 분기문을 수행할 때 다른 프로세서가 유휴 상태에 있다면 그 유휴 상태의 프로세서가 분기문의 다른 한 쪽 흐름을 수행한다. 또한 시뮬레이션을 통해 다중 쓰레드를 고려한 벤치마크 프로그램과 단일 쓰레드로 구현된 벤치마크 프로그램에 대해서 각각 제안된 기법의 성능을 측정하였다. 분기 동시 수행으로 약 20%의 유휴 시간이 감소하였고, 분기 예측 성공률은 최대 10%까지 향상 되었다. 전체적으로 일반적인 단일 칩 멀티프로세서에 비해 최대 39%의 성능 향상을 이루었고, 슈퍼스칼라 프로세서에 비해 최대 27%의 성능 향상을 이루었다.

본 논문은 다음과 같이 구성된다. Ⅱ장에서는 다중 경로 실행이 적용될 기반 아키텍쳐인 단일 칩 멀티프로세서를 먼저 소개하고, 단일 칩 멀티프로세서에서 수행되는 다중 경로 실행의 동작과 수정된 단일 칩 멀티프로세서 구조를 설명한다. Ⅲ장에서는 시뮬레이션을 통해 본 논문이 제안하는 구조의 성능을 평가한다. 마지막으로 Ⅳ장에서는 본 논문의 결론 및 향후 연구 과제를 제시하며 본 논문을 마친다.

Ⅱ. 본 론

1. 단일 칩 멀티프로세서

단일 칩 멀티프로세서는 명령서 수준의 병렬성 뿐만 아니라 쓰레드 수준 병렬성을 이용하도록 제안된 일종의 멀티프로세서 아키텍쳐이다. 기존의 일반적인 멀티프로세서와는 달리 하나의 칩에 멀티프로세서를 집적시킨 구조를 가진다. 또한 캐쉬와 주 메모리의 공유 구조에 따라 몇 가지 다른 구조의 단일 칩 멀티프로세서가 가능하다. 그림 1은 가장 일반적인 2차 캐쉬를 공유하는 단일 칩 멀티프로세서의 구조를 보여준다.

단일 칩 멀티프로세서는 동시에 많은 수의 명령어를 이슈하고 여러 개의 실행 유닛을 가지는 슈퍼스칼라 프로세서에 비해 다음과 같은 장점을 가진다. 첫 번째로 명령어 수준의 병렬성 뿐만아니라 쓰레드 수준의 병렬성을 이용할 수 있다. 두 번째 장점은 단일 칩 멀티프로

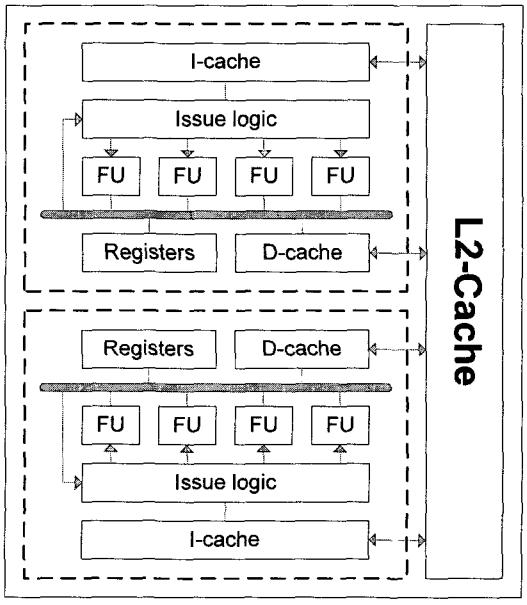


그림 1. 2차 캐쉬를 공유하는 단일 칩 멀티프로세서 구조

Fig. 1. Single chip multiprocessor with shared L2 cache.

세서를 구성하는 각각 하나의 프로세서는 슈퍼스칼라 프로세서보다 간단한 하드웨어 구성을 가지며 적은 하드웨어 자원을 가지게 설계될 수 있다는 점이다[10]. 정수 연산이 주를 이루는 어플리케이션 같은 경우 병렬성을 가지는 명령어들이 넓게 분포되는 경향이 있기 때문에 이를 찾아서 병렬적으로 수행하기 위해서는 명령어 윈도우(instruction window)가 매우 커야할 뿐만 아니라 분기 예측이 완벽해야 한다. 사실상 이것은 제한적인 하드웨어 리소스로 구현이 불가능하기 때문에 이런 특성을 가지는 어플리케이션에서 활용할 수 있는 명령어 수준의 병렬성은 크지 않다[4].

단일 칩 멀티프로세서는 기존의 일반적인 멀티프로세서를 하나의 칩에 집적함으로써 대칭형 멀티프로세서와 같은 아키텍처에 비해 다음과 같은 장점이 있다. 첫 번째로 다중 쓰레드 어플리케이션에서 많이 발생하는 쓰레드 간의 데이터 공유를 위한 통신 비용이 적다. 대칭형 멀티프로세서 아키텍처에서는 프로세스들 간의 데이터 공유를 위한 수단으로 공유 메모리를 통하기 때문에 서로 다른 프로세서에서 실행 중인 프로세스 간에 통신 지연 시간은 상당히 크다. 이에 비해 단일 칩 멀티프로세서는 그 구조에 따라 캐쉬를 통한 데이터의 공유가 가능하기 때문에 공유 데이터 통신에 드는 지연이 매우 작다. 두 번째는 단일 칩 멀티프로세서는 하나의 프로그램이 가지는 여러 쓰레드를 병렬적으로 수행시킬 수 있는 구조이다. 기존의 대칭형 멀티프로세서는 서로 다른 프로그램을 병렬적으로 수행시키는데 적합한 구조이다. 세 번째 장점은 적은 면적을 차지하고 상대적으로 하드웨어 구조가 간단해지기 때문에 더 높은 동작 속도를 가질 수 있으며 전력 소모를 줄일 수 있다.

2. 분기 동시 수행

본 논문에서 제안하는 분기 동시 수행은 단일 칩 멀티프로세서에 다중 경로 실행 기법을 적용한 것이다. 다음에 설명하는 내용들은 두 개의 슈퍼스칼라 프로세서를 가지는 단일 칩 멀티프로세서를 가정한다. 분기문으로 인한 두 경로의 제어 흐름을 단일 칩 멀티프로세서의 두 프로세서에서 각각 실행하게 한다. 분기의 두 제어 흐름을 수행 중에 분기의 경로가 결정되면 해당되는 프로세서는 계속 수행을 하고 다른 프로세서는 수행을 멈추게 된다.

가. 분기 동시 수행의 동작 모드

단일 칩 멀티프로세서의 각 프로세서는 세 가지 상태에서 동작할 수 있다. 프로세서가 자신에 할당된 쓰레드를 수행하는 일반적인 상태, 프로세서에 할당된 쓰레드가 없어서 유휴 중인 상태, 분기 동시 수행으로 두 프로세서가 서로 다른 제어 흐름을 수행 중인 상태이다.

독립 수행 모드(private mode). 단일 칩 멀티프로세서의 프로세서가 할당된 쓰레드를 수행 중인 일반적인 상태이다. 두 프로세서가 동시에 독립 수행 모드에서 동작하는 시간이 많을수록 단일 칩 멀티프로세서의 성능을 최대한 이끌어 낼 수 있다. 다중 쓰레드가 효율적으로 작성된 프로그램들은 두 프로세서가 대부분의 실행 시간을 독립 수행 모드에서 동작하게 한다. 단일 칩 멀티프로세서의 두 프로세서가 모두 독립 수행 모드에서 동작하고 있다면 유휴 중인 프로세서가 없기 때문에 각 쓰레드를 수행 중인 프로세서가 조건 분기문을 만나도 분기 동시 수행을 할 수 없다.

대기 모드(wait mode). 일반적인 단일 칩 멀티프로세서는 각 프로세서에 할당된 쓰레드가 종료되었거나 할당된 쓰레드가 없는 경우에 해당 프로세서가 유휴 상태로 아무 동작을 수행하지 않는다. 즉, 프로세서 자원의 절반이 낭비된다. 분기 동시 수행을 하는 단일 칩 멀티프로세서에서 할당된 쓰레드가 없는 프로세서는 대기 모드로 들어간다. 대기 모드에서 동작 중인 프로세서는 앞으로 발생할 수 있는 분기 동시 수행을 위해 레지스터 파일을 복제하고 1차 캐쉬를 선인출을 수행한다. 이 두 가지 동작은 뒤에서 자세히 설명한다. 대기 모드로 진입한 프로세서는 레지스터 파일 복제와 1차 캐쉬 선인출 작업은 수행하다가 독립 수행 모드에서 수행 중인 프로세서가 조건 분기문을 수행하게 될 때 분기 수행 모드로 전환된다. 또한 대기 모드에서 분기 동시 수행을 수행하기 전에 새로운 쓰레드를 할당 받게 되면 독립 수행 모드로 전환된다.

분기 수행 모드(branch mode). 단일 칩 멀티프로세서의 두 프로세서가 각각 독립 수행 모드와 대기 모드에서 수행 중 일 때, 독립 수행 모드의 프로세서가 조건 분기문을 수행하게 되면 대기 모드에 있는 프로세서는 분기 수행 모드로 전환된다. 두 프로세서는 각자의 제어 흐름을 따라 같은 쓰레드를 독립적으로 수행한다. 이렇게 독립 수행 모드와 분기 수행 모드에서 동작 중인 두 프로세서는 조건 분기문의 경로가 결정되면 바른 경로를 수행 중인 프로세서는 독립 모드로 돌아가서 수행을 계속하게 되고, 잘못된 경로를 수행 중인 프로세서는 쓰레드의 수행을 중단하고 대기 모드로 돌아간다. 이렇게 대기 모드에서 수행 중인 프로세서가 없는 경우

(분기 동시 수행 중인 경우)에 프로세서가 또 다른 분기문을 만나게 되면 분기 동시 수행을 하지 않고 슈퍼스칼라 단일 프로세서가 분기문을 처리하는 방법대로 동작한다.

나. 분기 동시 수행의 흐름

분기 동시 수행을 하는 단일 칩 멀티프로세서의 두 프로세서는 각각 앞서 설명한 세 가지 동작 모드를 바

```
1:    spawning_thread(thread1_func)
      ⋮
2:    if a = b
3:        then path ← func_ppath()
4:    else
5:        for each elements (e) of array_A
6:            e ← func2(e)
7:        path ← func_cpath()

8:    spawning_thread(thread2_func)
```

그림 2. 분기 동시 수행을 위한 예제 코드
Fig. 2. Example code for concurrent branch execution.

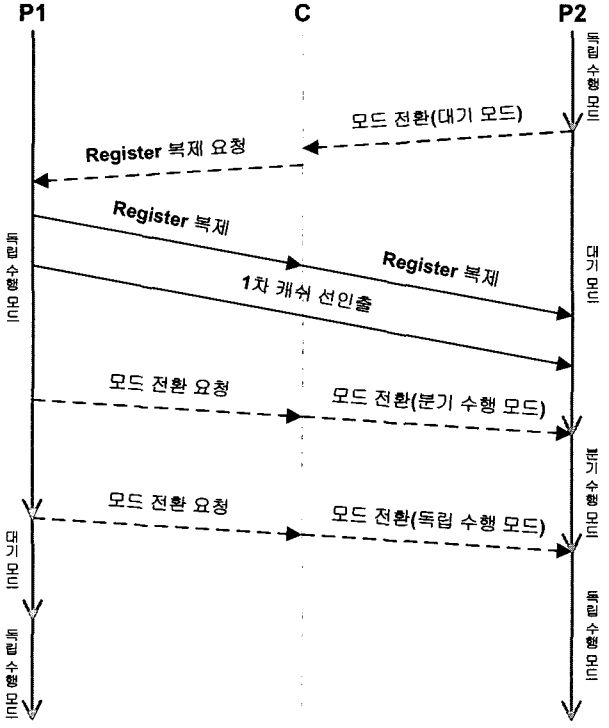


그림 3. 그림 2의 예제 코드를 수행하는 동안 프로세서의 상태 변화
Fig. 3. Processors's state transition during executing Fig 2 code.

꾸어 가면서 필요한 시점에서 분기 동시 수행을 한다. 프로세서의 동작 모드가 어떻게 전환되는지 간단한 예제 코드를 통해서 살펴본다.

그림 2는 단일 칩 멀티프로세서가 수행하게 될 프로그램 코드이다. 그림 3은 단일 칩 멀티프로세서의 두 프로세서의 상태 전이를 보여주는 그림이다. 초기에 프로세서 1이 그림 2의 코드를 수행한다고 가정하면 프로세서 1이 라인 1을 수행하면 새로운 쓰레드가 프로세서 2에 할당된다. 그림 3에서 두 프로세서의 초기 상태가 독립 수행 모드임을 볼 수 있다. 잠시 후에 프로세서 2의 쓰레드가 종료된다고 가정하면 프로세서 2는 분기 동시 수행 제어기에 대기 모드로 전환함을 알리고 대기 모드로 들어가게 된다. 대기 모드에서는 분기 동시 수행 제어기를 통해서 레지스터 복제와 1차 캐쉬 선인출이 이루어진다. 프로세서 1이 라인 2의 if-분기문을 수행하게 되면 제어기에 이를 알리게 되고 대기 모드의 프로세서는 분기 수행 모드로 전환된다. 프로세서 2가 라인 5의 else 블록을 수행한다고 가정하면 프로세서 2는 for-분기문을 다시 만나게 되는데 이미 분기 수행 모드에서 수행 중이므로 상태 전이는 발생하지 않는다. 이렇게 두 프로세서가 분기 수행 모드에서 각 제어 흐름을 수행 중에 분기의 결과가 else 블록으로 결정되면 프로세서 1은 이를 제어기에 알리고 대기 모드로 들어간다. 프로세서 2는 독립 수행 모드로 전환되어 수행을 계속한다. 이후에 프로세서 2가 라인 8을 수행하게 되면 대기 모드에 있는 프로세서 1은 독립 수행 모드에서 새로운 쓰레드를 수행하게 된다.

다. 레지스터 파일 복제

분기 동시 수행을 하기 위해서는 대기 모드에서 동작하는 프로세서의 레지스터 파일을 독립 수행 모드에서 수행 중인 프로세서의 레지스터 파일에 일치시키는 작업이 필요하다. 두 프로세서는 서로 다른 쓰레드를 각자 실행하고 있었으므로 레지스터 파일의 내용이 다르게 된다. 분기 동시 수행은 하나의 쓰레드에서 조건 분기문이 만드는 두 제어 흐름을 수행하는 것이기 때문에 반드시 레지스터 파일이 일치해야 정확한 결과를 얻을 수 있다. 따라서 대기 모드에서는 분기 동시 수행을 시작하거나 새로운 쓰레드를 할당 받아 독립 수행 모드로 전환될 때까지 지속적으로 레지스터 파일을 일치시키는 작업을 하게 된다.

레지스터 파일이 전용의 통신 채널을 통해서 복제되는 동안에도 독립 수행 모드의 프로세서는 명령어를 계

클럭	명령어	전체복제	변경복제
1	ADD r0, r1, r2	r0, r1, r2	
2	SUB r10, r7, r1	r3, r4, r5	r0
3	SUB r4, r2, r3	r6, r7, r8	
4	AND r9, r4, #0	r9, r10, r11	r4
5	SUB r0, r10, r7		r9
6			r0

그림 4. 두 프로세서 간에 레지스터 파일 복제를 위한 예제 프로그램

Fig. 4. Example code of register file duplication between the two processors.

	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11
P1	7	3	10	4	5	20	21	34	12	16	20	11
P2	X	X	X	X	X	X	X	X	X	X	X	X

(a) 두 프로세서의 초기 레지스터 값

	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11
P1	13	3	10	4	5	20	21	34	12	16	20	11
P2	7	3	10	X	X	X	X	X	X	X	X	X

(b) 두 프로세서의 2번째 클럭에서 레지스터 값

	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11
P1	13	3	10	4	5	20	21	34	12	16	31	11
P2	13	3	10	4	5	20	X	X	X	X	X	X

(c) 두 프로세서의 3번째 클럭에서 레지스터 값

	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11
P1	13	3	10	4	6	20	21	34	12	16	31	11
P2	13	3	10	4	5	20	21	34	12	X	X	X

(d) 두 프로세서의 4번째 클럭에서 레지스터 값

	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11
P1	13	3	10	4	6	20	21	34	12	0	31	11
P2	13	3	10	4	6	20	21	34	12	16	31	11

(e) 두 프로세서의 5번째 클럭에서 레지스터 값

	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11
P1	-3	3	10	4	6	20	21	34	12	0	31	11
P2	13	3	10	4	6	20	21	34	12	0	31	11

(f) 두 프로세서의 6번째 클럭에서 레지스터 값

	r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11
P1	-3	3	10	4	6	20	21	34	12	0	31	11
P2	-3	3	10	4	6	20	21	34	12	0	31	11

(g) 두 프로세서의 7번째 클럭에서 레지스터 값

그림 5. 두 프로세서 간에 레지스터 파일 복제 과정

Fig. 5. Procedure of register file duplication between the two processors.

속 수행하고 있기 때문에 레지스터의 값은 계속 변경된다. 따라서 통신 채널을 통한 레지스터 파일의 복제는 두 가지 형태의 레지스터 값을 한 쌍으로 하여 이루어져야 한다. 한 가지는 아키텍쳐 레지스터 전체를 복사하는 형태이고 다른 한 가지는 가장 최근에 변경된 레지스터를 복사하는 형태이다. 일단 동일한 쓰레드를 두 프로세서에서 수행하기 위해 모든 아키텍쳐 레지스터가 복제 되어야 함은 위에서 언급하였다. 대기 모드의 프로세서는 실제로 명령어를 수행하는 것이 아니므로 레지스터 값이 변경될 때마다 복제해 주어야 한다. 따라서 모든 아키텍쳐 레지스터가 한 번 복제될 때까지는 두 가지 종류의 복제가 일어나고 이후에는 변경되는 레지스터의 값만이 복제된다. 그림 4의 예제 코드를 수행할 때 두 프로세서간의 레지스터 파일 복제가 일어나는데 그림 5가 이 과정을 보여준다.

여기서 두 프로세서의 아키텍쳐 레지스터는 12개이고, 모든 명령어는 1클럭 사이클만에 수행되며 통신 채널을 통한 전송에 따른 지연 시간은 1클럭 사이클로 가정했다. 그림 5.(a)에서 처음 4클럭 동안 레지스터 파일 전체를 한번 복제(전체복제)하고 나면 그 이후에는 실시간으로 변경이 되는 레지스터만 복제(변경복제)된다. 그림 5.(c)에서 r10의 값이 변경되었지만 r10이 레지스터 파일 전체 복제 과정을 거치지 않았기 때문에 다음 클럭인 그림 5.(d)에 복제하지 않는다. 변경된 r10의 값은 그 다음 클럭인 그림 5.(e)에서 레지스터 파일 전체 복제 과정을 통해 복제된다. 이러한 과정을 거쳐서 최종적으로 그림 5.(g)에서 두 프로세서의 레지스터 파일 복제가 완료되었다.

통신 채널에서 발생하는 약간의 지연으로 인해 독립 수행 모드 프로세서의 레지스터 값 변경은 몇 클럭 사이클 이후에 대기 모드 프로세서의 레지스터에 반영된다. 따라서, 분기 수행 모드로의 진입도 두 프로세서 간에 시간차가 존재한다.

라. 1차 캐쉬 선인출

분기 동시 수행을 위해 대기 모드에서 필요한 또 다른 사전 작업은 1차 캐쉬의 선인출(prefetch)이다. 앞서 설명한 레지스터 파일과 마찬가지로 서로 다른 쓰레드를 수행했던 두 프로세서의 1차 캐쉬의 내용은 다르기 때문에 유휴 상태에 있다가 분기 동시 수행을 시작하는 프로세서에서는 1차 캐쉬 실패가 증가하게 된다. 물론, 2차 캐쉬를 공유하는 구조이기 때문에 1차 캐쉬 실패를 발생시키는 데이터 요청은 2차 캐쉬에 존재할 확률이

높다. 따라서 이것은 레지스터 파일의 일치와는 달리 반드시 필요한 작업은 아니다. 하지만, 1차 캐쉬 실패로 인한 성능 저하를 줄이기 위해 대기 모드에서 동작하는 동안 지속적으로 1차 캐쉬의 선인출을 수행한다. 선인출은 독립 수행 모드에 있는 프로세서가 2차 캐쉬로 요청하는 데이터를 대기 모드의 프로세서도 같이 가져가는 방법으로 수행된다.

3. 분기 동시 수행의 하드웨어 구현

단일 칩 멀티프로세서에 다중 경로 실행 기법을 적용하기 위해서는 그 구조에 대한 몇 가지 수정이 필요하다. 필요한 수정은 레지스터 파일 복제를 위한 두 프로세서간의 통신 채널 추가와 1차 캐쉬 선인출을 위한 버스 구조의 변경이다. 그림 6은 분기 동시 수행을 위해 수정된 단일 칩 멀티프로세서의 구조를 보여준다.

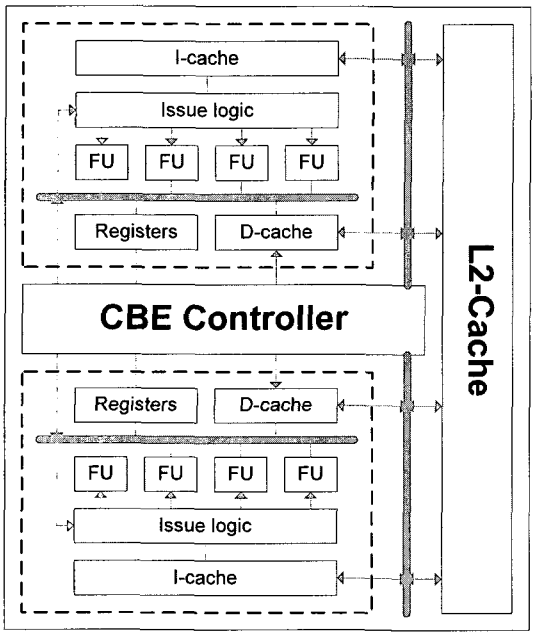


그림 6. 분기 동시 수행을 위한 단일 칩 멀티프로세서 구조

Fig. 6. Single chip multiprocessor architecture for concurrent branch execution.

가. 프로세서간 통신 채널

대기 모드에서 레지스터 파일을 복제하기 위해 두 프로세서 간에 통신 채널이 필요하다. 두 프로세서가 단일 칩에 집적되어 있기 때문에 통신 채널로 레지스터 파일을 복제하는데 그리 큰 지연 시간이 걸리지 않는다. 또한 레지스터 복제를 담당하는 전용의 제어기를 두어 레지스터 복제를 위해 프로세서가 낭비되지 않도록 한다. 이 제어기에는 복제를 위한 버퍼가 있으며 레

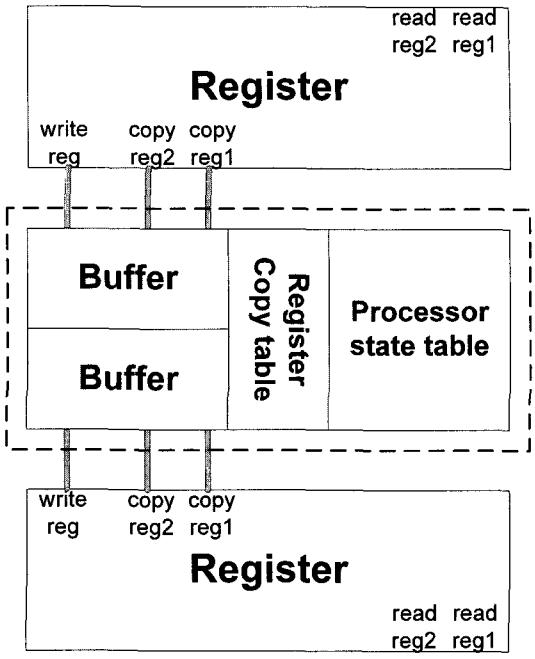


그림 7. 프로세서간 통신을 위한 하드웨어 구조

Fig. 7. Hardware structure for inter-processor communication.

지스터 복제에 걸리는 시간을 단축하기 위해 현재 상태에서의 아키텍쳐 레지스터(architectural register) 만을 복사한다.

일반적인 레지스터 파일 구조에 통신 채널의 대역폭에 따라 몇 개의 읽기 포트가 추가된다. 이 추가된 읽기 포트는 분기 동시 수행 제어기가 레지스터 파일의 값을 읽어 가기위해 사용된다. 그러나 레지스터 파일에 추가의 쓰기 포트는 필요하지 않은데 기존의 쓰기 포트를 통해 레지스터 값의 복제가 가능하다. 대기 모드의 프로세서는 코드를 수행하지 않는 상태이므로 레지스터 파일의 쓰기 포트가 사용되지 않는다. 따라서, 분기 동시 수행 제어기는 레지스터 파일의 쓰기 포트를 통해서 대기 모드 프로세서의 레지스터에 값을 쓰게된다. 또한 어떤 레지스터까지 복제가 완료되었는지에 대한 정보를 저장하는 테이블(register copy table)이 추가되며 두 프로세서의 현재 동작 모드를 저장하기 위한 테이블(processor state table)도 포함된다. 그림 7은 이러한 분기 동시 수행 제어기의 레지스터 복제에 관련된 부분의 블록도를 보여준다.

나. 1차 캐쉬 선인출을 위한 버스 구조

2차 캐쉬를 공유하는 단일 칩 멀티프로세서 구조에서는 대기 모드에서 분기 수행 모드로 전환되는 프로세서의 1차 캐쉬의 적중률이 떨어질 수 있다. 대기 모드

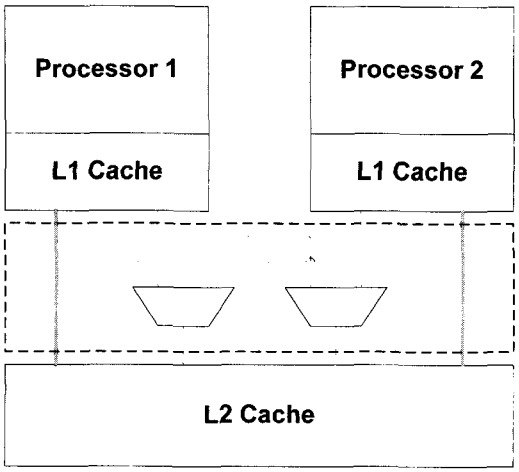


그림 8. 1차 캐쉬 선인출을 위한 개략적인 캐쉬 구조
Fig. 8. Cache structure for prefetching the L1 cache.

에서 최대한 선인출을 하여 1차 캐쉬를 일치시킨다. 독립 수행 모드에서 동작 중인 프로세서가 1차 캐쉬 실패를 발생시키면 2차 캐쉬에 데이터 요청을 보내고 데이터를 받아 1차 캐쉬를 갱신시킨다. 이 때, 대기 모드의 프로세서도 그 요청에 대한 데이터를 받아 자신의 1차 캐쉬를 갱신시키도록 하면 두 프로세서의 1차 캐쉬는 점차 일치하게 된다. 이러한 선인출을 가능하게 하기 위해서는 두 프로세서의 1차 캐쉬와 2차 캐쉬 간의 주소 버스를 멀티플렉싱하여 동일한 주소의 값을 가져오게 하는 것이다.

그림 8에 개략적인 구조가 나타나 있다. 점선으로된 사각형 영역은 분기 동시 수행 제어기의 한 부분이다. 굵은 선은 주소 버스를 나타내고 멀티플렉서를 통해 교차 연결된 라인은 데이터 버스를 나타낸다. 두 프로세서의 1차 캐쉬에서 공유 2차 캐쉬로의 주소 버스가 서로 교차하여 연결되어 있음을 알 수 있다. 프로세서의 상태가 독립 수행 모드나 분기 수행 모드일 경우 2차 캐쉬로 연결되는 주소 버스는 각자 자신의 주소와 연결되어 두 프로세서가 서로 다른 메모리 주소의 값을 캐쉬로 읽어 갈 수 있다. 그러나, 대기 모드에서는 독립 수행 모드에서 수행 중인 프로세서의 주소 버스가 2차 캐쉬로 연결되어 독립 수행 모드의 프로세서가 읽어가는 주소의 값을 1차 캐쉬로 가져오게 된다.

III. 시뮬레이션 및 성능 분석

본 논문에서 제안하는 방법의 성능을 시뮬레이션을 통해 다른 아키텍쳐들의 성능과 비교, 분석한다. 시뮬레이터로 심플스칼라(SimpleScalar) 툴셋[11]을 사용하고

표 1. 시뮬레이션에 사용된 벤치마크 프로그램
Table 1. Used benchmark program in the simulation.

벤치마크 프로그램	정수/실수	특성	설 명
bzip2	CINT2000	단일 쓰레드	Data compression utility
gcc	CINT2000	다중 쓰레드	C compiler
gzip	CINT2000	단일 쓰레드	Data compression utility
twolf	CINT2000	단일 쓰레드	Place and route simulator
vortex	CINT2000	다중 쓰레드	Object Oriented Database
vpr	CINT2000	단일 쓰레드	FPGA circuit placement and routing
applu	CFP2000	단일 쓰레드	Parabolic/elliptic partial differential equations
equake	CFP2000	다중 쓰레드	Finite element simulation; earthquake modeling
mgrid	CFP2000	다중 쓰레드	Multi-grid solver in 3D potential field

표 2. 시뮬레이션에 사용된 아키텍쳐
Table 2. Used architecture in the simulation.

	Superscalar	CMPs	CMPs with CBE
프로세서 수	1	2	2
아키텍쳐 레지스터 수 (정수 / 실수)	32 / 32	32 / 32	32 / 32
총 레지스터 수 (정수 / 실수)	40 / 40	20 / 20	20 / 20
기능 유닛 수	3	1	1
명령어 이슈 큐 (엔트리)	32	8	8
명령어 캐쉬	16KB	8KB	8KB
데이터 캐쉬	16KB	8KB	8KB
2차 캐쉬	256KB	128KB	128KB
2차 캐쉬 접근 지연 (클럭 사이클)	4	4	4
메모리 접근 지연 (클럭 사이클)	50	50	50
통신 채널 지연 (클럭 사이클)	N/A	N/A	2
통신 채널 폭	N/A	N/A	128bit
통신 채널 버퍼	N/A	N/A	20 x 32bit

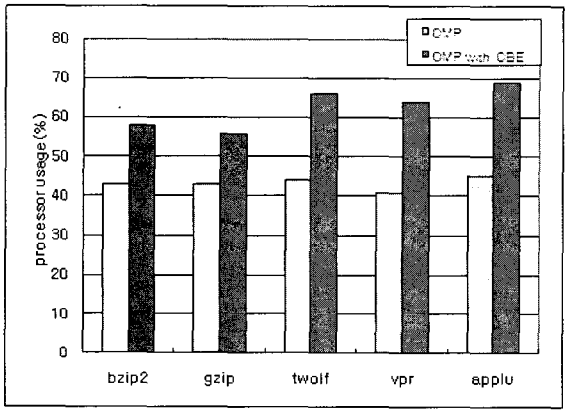
SPEC2000 벤치마크 프로그램을 사용하여 시뮬레이션을 수행하였다. 표 1에 시뮬레이션에 사용된 벤치마크 프로그램들의 특성을 요약하였다. 표 2는 본 논문에서

제안하는 구조에 해당하는 시뮬레이션 모델과 비교를 위해 사용하는 슈퍼스칼라 모델, 단일 칩 멀티프로세서 모델의 사양을 보여준다.

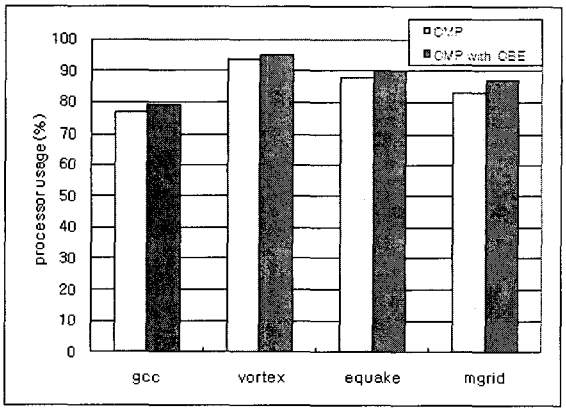
가. 단일 칩 멀티프로세서의 효율

그림 9는 벤치마크 프로그램을 수행하는 동안 단일 칩 멀티프로세서(CMPs)와 분기 동시 수행을 하는 단일 칩 멀티프로세서(CMPs with CBE)의 프로세서 사용률을 보여준다.

그림 9.(a)에서 단일 칩 멀티프로세서의 하나의 프로세서는 프로그램 실행 시간의 대부분을 유휴 상태로 보냄을 알 수 있다. 이러한 단일 칩 멀티프로세서에 분기 동시 수행을 적용한 결과는 대략 20%의 시간을 프로그램 수행에 더 사용하는 것을 보여준다. 이 시간은 분기 동시 수행 기법에 의해 조건 분기의 두 흐름을 수행하는데 사용될 것이다. 그림 9.(b)에서는 이미 프로그램의 다중 쓰레드 특성으로 단일 칩 멀티프로세서가 충분한 효율을 보인다. 이 상태에서는 두 프로세서가 모두 독립 수행 모드에서 동작하는 시간이 많으므로 분기 동시 수행이 거의 발생하지 않는다. 분기 동시 수행을 하는 프로세서의 수치가 거의 차이가 없음을 알 수 있다.



(a) 단일 쓰레드 벤치마크



(b) 다중 쓰레드 벤치마크

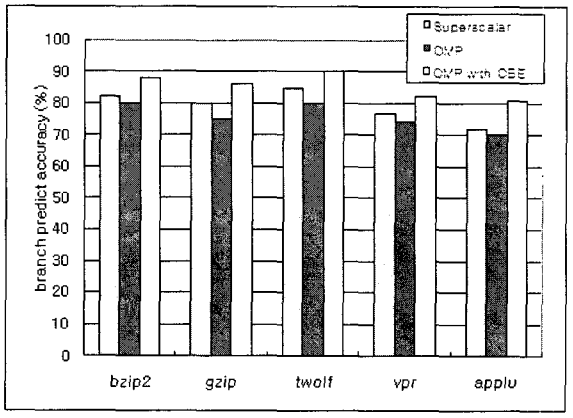
그림 9. 단일 칩 멀티프로세서의 효율

Fig. 9. Efficiency of single chip multiprocessor.

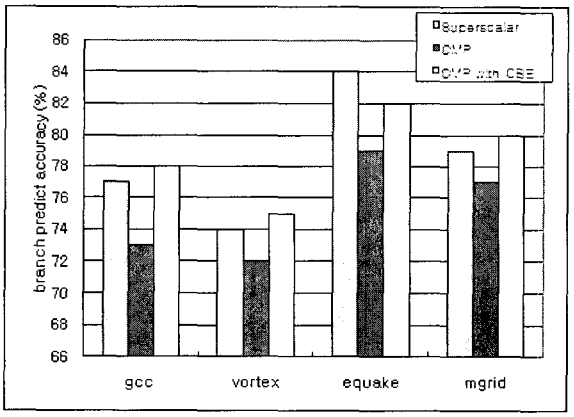
나. 분기 동시 수행에 따른 분기 예측률

그림 10은 분기 동시 수행으로 향상된 분기 예측률을 보여준다. 세 가지 프로세서에 대해서 측정한 분기 예측률을 비교하였다.

분기 동시 수행은 올바른 분기의 흐름이 끊어지지 않고 계속되므로 이상적인 분기 예측기를 가지는 프로세서로 볼 수 있다. 그러나 본 논문에서 제안된 분기 동시 수행은 중첩을 허락하지 않기 때문에 중첩된 조건 분기가 발생시 분기 예측기가 분기를 예측하게 된다. 따라



(a) 단일 쓰레드 벤치마크



(b) 다중 쓰레드 벤치마크

그림 10. 슈퍼스칼라, 단일 칩 멀티프로세서, 분기 동시 수행 프로세서의 분기 예측률 비교

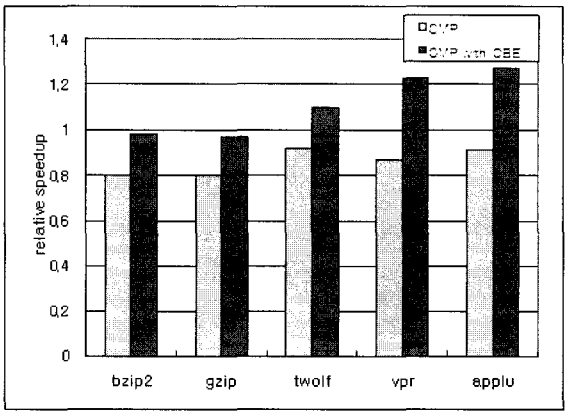
Fig. 10. Comparison of a branch prediction accuracy in a superscalar processor, a single chip multiprocessor, and a single chip multiprocessor with concurrent branch execution.

서 이상적인 분기 예측기와 같은 성능을 나타낼 수는 없다. 분기 동시 수행으로 향상될 수 있는 예측률은 하나의 프로세서가 유휴 상태에 있어서 분기 동시 수행이 가능한 경우에 해당되는 양 만큼이다. 기본적으로 분기 예측기의 예측률이 뛰어나기 때문에 단일 쓰레드 프로그램과 다중 쓰레드 프로그램들 간에 예측률 향상의 차이는 크게 나타나지 않았음을 볼 수 있다.

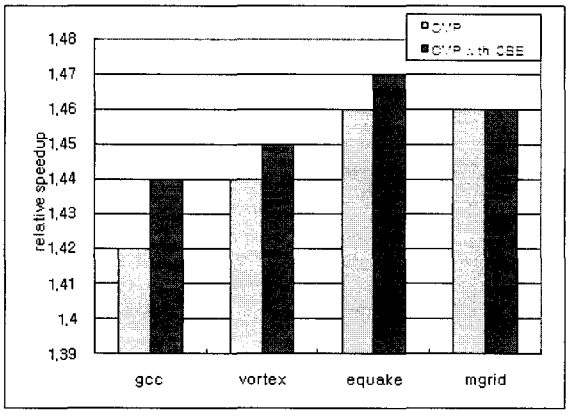
다. 성능 비교

그림 11은 세 가지 프로세서의 전반적인 성능을 비교한다. 슈퍼스칼라 프로세서의 성능을 1로 보았을 때, 두 프로세서의 성능을 비교한다.

그림 11.(a)의 단일 쓰레드 프로그램에 대하여 일반적인 단일 칩 멀티프로세서는 비슷한 성능의 슈퍼스칼라 프로세서의 성능을 밑도는 것으로 나타났다. 이유는



(a) 단일 쓰레드 벤치마크



(b) 다중 쓰레드 벤치마크

그림 11. 슈퍼스칼라, 단일 칩 멀티프로세서, 분기 동시 수행 프로세서간의 성능 비교

Fig. 11. Comparison of performance in a superscalar processor, a single chip multiprocessor, and a single chip multiprocessor with concurrent branch execution.

단일 쓰레드를 수행하는 하나의 프로세서만 고려했을 때 슈퍼스칼라 프로세서가 더 많은 하드웨어 자원을 가지기 때문이다. 그러나 분기 동시 수행을 하였을 때는 최대 27%까지의 성능 향상을 보였다. applu 프로그램이 좀 더 큰 성능 향상을 보인 것은 다른 프로그램에 비해 2차 캐쉬 실패율이 높기 때문인 것으로 분석된다. 이 프로그램은 24.26%의 2차 캐쉬 실패율을 보였다. 2차 캐쉬 실패율이 높다는 것은 조건 분기문과 의존성이 있는 데이터의 2차 캐쉬 실패율도 당연히 높을 수 밖에 없기 때문이다. 이런 경우에 분기 동시 수행으로 상당히 많은 손실을 줄일 수 있기 때문에 성능 향상이 더욱 크게 나타난 것으로 보인다.

그림 11.(b)의 다중 쓰레드 프로그램에서는 단일 칩 멀티프로세서가 슈퍼스칼라 프로세서에 비해 가지는 장점으로 인한 성능 향상이 나타났음을 알 수 있다. 예상대로 이미 다중 쓰레드를 이용할 수 있는 프로그램들이기 때문에 분기 동시 수행으로 얻는 이득은 미비함을 볼 수 있다.

Ⅳ. 결 론

본 논문은 기존의 단일 칩 멀티프로세서의 낭비되는 자원을 이용하고 분기 예측 실패시 발생하는 손실을 줄이고자 단일 칩 멀티프로세서에 다중 경로 실행을 적용한 분기 동시 수행 기법을 제안하였다. 또한 분기 동시 수행을 단일 칩 멀티프로세서에서 구현하기 위한 알고리즘과 하드웨어 구조를 제안하였고, 시뮬레이션을 통해 기존의 일반 단일 칩 멀티프로세서와 성능을 비교하였다.

제안된 분기 동시 수행 기법은 쓰레드 수준의 병렬성이 적은 응용에서 성능 향상을 꾀하였다. 시뮬레이션은 본 논문에서 목표로 하는 상황에 적합한 벤치마크 프로그램들과 그러한 특성이 크게 나타나지 않는 일반적인 벤치마크 프로그램들을 모두 사용하여 본 논문이 가진 목적에 결과가 얼마나 부합되는지를 확인하였다.

시뮬레이션을 통해 슈퍼스칼라 프로세서와 제안된 구조를 비교하였다. 제안된 구조는 다중 쓰레드 특성을 보이는 벤치마크 프로그램에 대해서 슈퍼스칼라 프로세서 보다 약 40% 높은 성능을 보여주었다. 동일한 벤치마크 프로그램들에 대해서 슈퍼스칼라 프로세서 대비 단일 칩 멀티프로세서로 얻는 성능 향상이 38% 수준인 점을 고려하면 분기 동시 수행을 통한 성능 향상은 미비함을 알 수 있다. 하지만 단일 쓰레드 특성을 보이는

벤치마크 프로그램에 대해서 분기 동시 수행 기법은 단일 칩 멀티프로세서보다 큰 성능 향상을 보여 주었다. 이 수치는 2차 캐쉬 실패가 빈번한 벤치마크 프로그램에서 좋아지는 경향을 보였다.

참 고 문 헌

[1] O. Mutlu, J. Stark, C. Wilkerson, and Y. N. Patt, "Runahead execution: an effective alternative to large instruction windows," IEEE Transaction on Micro, Vol. 23, Issue. 6, pp. 20-25, Nov. 2003.

[2] R. D. Barnes, J. W. Sias, E. M. Nystrom, S. J. Patel, J. Navarro, and W.W Hwu, "Beating in-order stalls with "flea-flicker" two-pass pipelining," IEEE Transactions on Computers, Vol. 55, Issue 1, pp. 18-33, Jan. 2006.

[3] J. Hennessy, and D. Patterson, *Computer Architecture : A quantitative Approach, 3rd edition*, Morgan Kaufmann, pp. 172-209, 2002.

[4] K. Olukotun et al., "The Case for a Single Chip Multiprocessor," Proc. 7th Int'l Conf. Architectural Support for Programming Languages and Operating Systems, ACM Press, pp. 2-11, New York, 1996.

[5] S. J. Eggers, J. S. Emer, H. M. Leby, J. L. Lo, R. L. Stamm, and D. M. Tullsen, "Simultaneous multithreading: a platform for next-generation processors," IEEE Transaction on Micro, Vol.17, Issue 5, pp. 12-19, Sept. 1997.

[6] W. S. Lee, J. J. Kim, J. C. Lee, and S. B. Choi, "Flying Interrupt: A Fast Way to Execute Interrupt Service Routines Using Simultaneous Subordinate Microthreading," International Technical Conference on Circuits/Systems, Computers and Communications, Sendai/ Matsushima, Japan, July 2004.

[7] L. Hammond, M. Willey, and K.Olukotun, "Data Speculation Support for a Chip Multiprocessor," Proc. 8th ACM Conf. Architectural Support for Programming Languages and Operating Systems, San Jose, California, Oct. 1998.

[8] I. Ganusov, and M. Burtscher, "Future execution: A Hardware Prefetching Technique for Chip Multiprocessors," Proc. 14th Int'l Conf. Parallel Architectures and Compilation Techniques, pp. 350-360, Sept. 2005.

[9] A. K. Uht, V. Sindagi, and K. Hall, "Disjoint eager execution: an optimal form of speculative execution," Proc. 28th Int'l Symp. Microarchitecture, pp. 313-325, Dec. 1995.

[10] B. A. Nayfeh, and K. Olukotun, "A single-chip multiprocessor," IEEE Transaction on Computer, Vol.30, Issue 9, pp. 79-85, Sept. 1997.

[11] *D. Burger, and T. Austin, "The Simplescalar Tool* Set, Version 2.0," Technical Report CS-TR97-1342, Univ. of Wisconsin, Madison, June. 1997.

저 자 소 개

이 승 렬(학생회원)

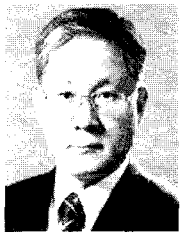
2004년 인하대학교 물리학과 학사 졸업.
2006년 인하대학교 전자공학과 석사 졸업.
<주관심분야 : 프로세서 구조, 임베디드 시스템, SoC 설계>

최 재 혁(학생회원)

2001년 인하대학교 전자공학과 학사 졸업.
2004년 인하대학교 전자공학과 석사 졸업.
2004년 인하대학교 전자공학과 박사 과정 재학.
<주관심분야 : 프로세서 구조, 임베디드 시스템, SoC 설계>

김 준 식(학생회원)

2005년 인하대학교 전자공학과 학사 졸업.
2007년 인하대학교 전자공학과 석사 졸업.
<주관심분야 : 멀티미디어, 무선 네트워크, QoS>



최 상 방(정회원)

1981년 한양대학교 전자공학과 학사 졸업.
1988년 Univ. of Washington, Dept. of Computer Eng. (M.S.)
1990년 Univ. of Washington, Dept. of Computer Eng. (Ph.D)
1981년~1986년 LG 정보통신 주식회사
1990년~1991년 Univ. of Washington / Postdoctoral Research Associate
1998년~1999년 Stanford University / Visiting Associate Professor
1991년~현재 인하대학교 전자공학과 교수
<주관심분야 : 컴퓨터 네트워크, 컴퓨터 구조, 임베디드 시스템, 병렬 및 분산 처리>