효율적인 이중 스레드 자바 프로세서 핵심

정 준 목°, 김 신 덕
연세대학교 컴퓨터과학과 병렬처리시스템 연구실

An Effective Dual Threaded Java Processor Core

Chun-Mok Chung and Shin-Dug Kim
Parallel Processing System Lab., Dept. of CS, Yonsei University

요 약

자바(Java)의 수행 성능을 향상시키기 위한 방법으로 자바 프로세서가 제안되었다. 그러나 현재의 자바 프로세서는 자바 가상 머신(Java Virtual Machine)의 구조만을 고려한 것이다. 본 논문에서는 기존 자바 프로세서의 성능을 향상시키는 방법으로 자바 프로그래밍에서 사용되는 다중스레드를 직접 지원하는 새로운 자바 프로세서인 동시 다중스레드 자바 칩(Simultaneous Multithreaded Java Chip, SMTJC)을 제안한다. SMTJC은 두 개의 독립적인 스레드를 동시에 수행함으로써, 자바 프로그램에서의 명령어 수준 병렬성(Instruction level parallelism)을 향상시킨다. 다중 스레드 수행을 위해 새로운 스택 캐쉬의 구조 및 운영 방법을 사용한다. JavaSim을 통한 시뮬레이션은 SMTJC이 기존 자바 프로세서에 비해 이중 스택 캐쉬와 추가적 처리 유닛들로 인해, 1.28~2.00의 전체적 수행 성능이 향상됨을 보여준다. 본 연구는 하드웨어와 소프트웨어의 상호 보완적인 기술적 경향을 배경으로 자바의 언어적 특성을 고려한 프로세서를 설계, 지원함으로써 자바 프로세서의 성능 향상을 도모하고 있다.

1. 서론

하드웨어 독립성을 제공하는 자바(Java)의 인터프리터(Iterpreter) 방식은 수행 성능이 기존 컴퓨터 언어로 제작된 프로그램들보다 떨어진다. 자바의 성능을 향상시키기 위해 자바 가상 머신을 하드웨어로 구현한 자바 프로세서를 사용하는 방법이 제안되었다 [1].

그러나 현재의 자바 프로세서는 자바 가상 머신(Java Virtual Machine) [9]의 특성만을 고려한 것이므로, 자바 언어의 다양한 특성을 고려하지 않은 단점을 가지고 있다. 본 논문에서는 기존 자바 프로세서의 간결한 구조를 기반으로 하며, 자바 언어의 특징이면서, 고성능 병렬 처리에 효율적인 자바 다중스레드(Multithread)를 직접적으로 지원하는 향상된 자바 프로세서의 구조를 제안하며, 실험을 통하여 성능 향상의 가능성을 제시한다.

2장에서는 관련 연구들을 알아보고, 3장에서는 본 논문에서 제안하는 동시 다중스레드 자바칩(Simultaneous Multithreaded JavaChip, SMTJC)의 구조를 설명한다. 4장에서는 실험을 통한 SMTJC의 성능을 평가한후, 끝으로 5장에서는 결론을 맺는다.

2. 관련 연구

본 절에서는 SMTJC 구조와 직접적으로 연관이 있는 자바 프로세서 관련 연구 및 다중스레드 프로세서 관련 연구들을 살펴본다.

J.Michael은 기존의 자바 가상 머신을 프로세서로 구현하는 작업을 수행했다 [1]. picoJava-I로 명명된 이 작고, 유연한 이 마이크로 프로세서 핵심은 자바 수행 성능을 기존의 인터프리티브다 20배 이상 향상시켰다 [1]. Lee-Ren은 picoJava-I의 구조를 근간으로 하여, 확장된 명령어 폴딩(Instruction folding) 방법을 적용한 자바 프로세서의 구조를 제안했다 [2].

Agarwal은 다중스레드 프로세서의 구조의 성능을 분석하면서, 다중스레드 프로세서의 설계시 고려할 점들에 대해 조사, 분석했다 [3]. Hirata는 하나의 프로세서에 다중스레드 기능을 포함한 프로세서의 구조를 제안하였다 [4]. J.Eggers와 그의 팀은 프로세서 내에서의 병렬성을 향상 시킬 수 있는 방법으로, 동시 다중스레드(Simultaneous Multithread, SMT) 프로세서를 제안했다 [5, 6]. 명령어 인출(Fetch)을 위한 효율적 프로세서 구조와 다양한 명령어 인출(Instruction fetching) 방법을 제안했으며 [7], 컴파일러 최적화 방법을 함께 제시하고 있다 [8].

3. 동시 다중스레드 자바 칩

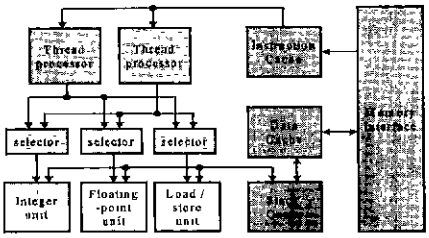
3.1. 프로세서의 구조

동시 다중스레드 자바 칩(Simultaneous Multithreaded Java Chip, SMTJC)은 picoJava-I의 구조를 바탕으로 하며, 추가적으로 이중 스레드를 지원하기 위한 구조를 가진다 [그림 1]. 각각의 스레드는 스레드 프로세서(Thread Processor, TP)에 의해 컨텍스트(Context)가 유지되며, 이중(Dual) 스레드 지원을 위해 SMTJC은 2개의 TP와 추가적인 정수용 ALU, 메모리 유닛을 등을 포함한다. 부동소수용 ALU는 공유하게 된다

명령어 캐쉬(Instruction cache), 데이터 캐쉬(Data cache), 스택 캐쉬(Stack cache)는 이중 스레드의 효율적 지원을 위해서, 스레드별로 분리된 구조를 가진다. 스택 캐쉬의 분리는 논리적 분리와 물리적 분리로 나뉘는 두 가지 구조를 지원한다

SMTJC은 picoJava-I [1]과 같은 인출, 디코드, 수행, 저장

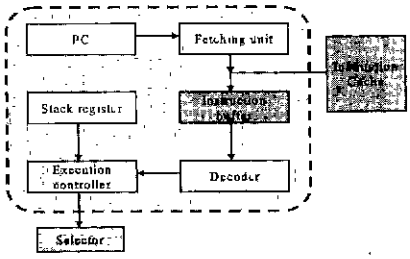
의 4단계 파이프라인을 지원한다.



[그림 1] 동시 다중스레드 자바칩의 구조

3.2 스레드 프로세서

TP는 하나의 컨텍스트를 유지하기 위한 구조를 가진다 [그림 2]. 현재 명령어 인출을 위한 명령어의 위치를 가리키는 하나의 pc를 가지며, 인출 유닛(Fetch unit)은 pc의 값에 따라 명령어를 명령어 캐쉬에서 읽어 명령어 버퍼(Instruction buffer)에 저장한다. 디코드 유닛(Decoding unit)은 명령어 버퍼의 내용을 디코드하며, 이는 수행 제어기(Execution controller)에 해당 신호를 보내게 된다. 스택 레지스터는 TP가 유지하는 컨텍스트가 사용하는 메모리 스택의 시작 위치를 가리키게 된다. 자바는 스택중심의 명령어 구조를 가지게 되므로, 하나의 컨텍스트에서는 하나의 메모리 스택이 유지되기 때문이다.



[그림 2] 스레드 프로세서의 구조

3.3. 스택 캐쉬

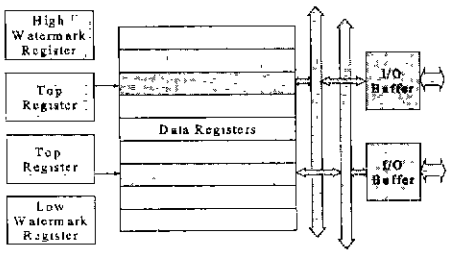
하나의 스레드를 지원하는 picoJava-I의 구조에서의 스택 캐쉬는 이중 스레드를 지원하는 SMTJC에서는 적합하지 않게 된다. 왜냐하면, 두 개의 스레드가 하나의 스택 캐쉬를 사용함에 따라 스택 캐쉬의 메소드 프레임(Method frame)간의 순서가 얽히게 되는 결과를 가져오기 때문이다 따라서 SMTJC에서는 이런 문제를 방지하기 위해 새로운 두 가지 방법을 제안한다.

3.3.1. 물리적 이중 스택 캐쉬

물리적 이중 스택 캐쉬(Physical dual stack cache, PDSC)의 구조는 2개의 독립적 스택 캐쉬를 지원하며, TP당 하나의 스택 캐쉬를 사용하므로, 두 개의 스레드가 동시에 스택 캐쉬를 참조할 수 있게 해준다. 각각의 스택 캐쉬는 picoJava-I에서와 같은 구조를 가지므로 운영방법은 기존과 같다. 두 개의 스택 캐쉬가 별도로 운영되므로, TP간의 충돌(Conflict)로 인한 지연시간이 없어 전체적 수행 시간을 단축시킬 수 있다 다만, 두 TP사이의 메소드 호출 빈도의 차이에 따라, 하나의 스택 캐쉬에서는 자주 참조 실패(Miss)가 발생하고, 다른 하나의 스택 캐쉬는 활용율이 낮아지는 불균형(Unbalancing)문제가 발생할 수 있다.

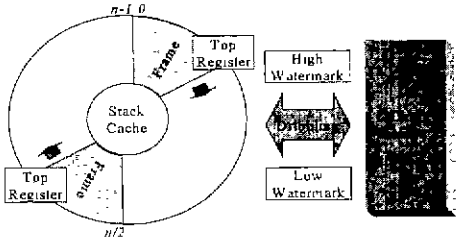
3.3.2. 논리적 이중 스택 캐쉬

논리적 이중 스택 캐쉬(Logical dual stack cache, LDSC)는 이중 스레드가 동시에 스택 캐쉬를 참조하기 위해서 두개의 포트를 제공하는 스택 캐쉬를 사용하며, 각 스레드별로 스택 캐쉬의 시작 주소를 다르게 하여, 각각을 별도로 운영하게 된다. LDSC는 4개의 전용 레지스터와 64개의 데이터 레지스터, 2개의 입출력 포트, 2개의 내부 버스로 구성된다 [그림 3]. 탑(Top) 레지스터는 스레드별로 유효 스택 탑을 각각 나타내며, 상위 표시(High watermark) 레지스터와 하위 표시(Low watermark) 레지스터는 드리블링(Dribbling)을 위해 사용된다. 64개의 데이터 레지스터에는 프레임의 내용이 저장되며, 2개의 독립적인 내부 버스에 각각의 데이터 레지스터가 연결되어 있어서, 동시에 서로 다른 데이터 레지스터의 내용을 참조 가능하게 한다 선택된 데이터 레지스터의 데이터는 2개의 입출력 포트를 통해 입출력이 이루어진다



[그림 3] 논리적 이중 스택 캐쉬의 구조

LDSC는 두 개의 스레드가 프레임의 시작 저장 위치를 다르게 하는 방법으로 하나의 데이터 레지스터들을 공유하여 사용하게 된다 즉, n개의 항을 가지는 스택 캐쉬를 가질 때, 스레드$_0$은 첫 번째 항에서부터 저장을 하며, 스레드$_1$은 n/2번째 항에서부터 저장을 시작한다 [그림 4] 각 스레드당 스택 캐쉬의 유효 크기는 평균적으로 n/2가 된다. 두개의 스레드가 전체 스택 캐쉬를 공유하게 되므로, 하나의 스레드가 메소드 호출이 빈번할 경우에도 전체적 스택 캐쉬의 활용도가 유지되는 장점을 가지게 되며, 스택 캐쉬를 두 개의 스레드가 동시에 이용하게 됨으로, 스택 캐쉬의 활용도가 picoJava-I에서보다 2배가량 높아지는 장점을 가지게 된다.



[그림 4] 논리적 이중 스택 캐쉬 운영 방법

4. 성능 평가

4.1. 실험 환경

본 연구에서는 자체 개발된 자바 칩 시뮬레이터인 JavaSim를 사용하여 실험을 하였으며, 본 논문의 모든 수행 결과는 이 시뮬레이터를 통한 데이터이다. JavaSim은 자바 명령어를 직접 수행하는 수행 기반(Execution-driven) 시뮬레이터로 자바칩의

설계 및 성능 분석을 위해 개발되었다.

실험은 [표 1]에서와 같은 세 가지 변수에 대해 각각 수행하였다. ALU는 이중 다중스레드 수행을 위해 단일에 ALU를 추가한 것을 의미하며, 스택 캐쉬 크기는 항의 수를 의미한다

[표 1] 시뮬레이션 설정 변수들

유닛	설정	
ALU	단일	이중
스택 캐쉬 구조	PDSC	LDSC
스택 캐쉬 크기	64항	128항

4.2. 벤치마크

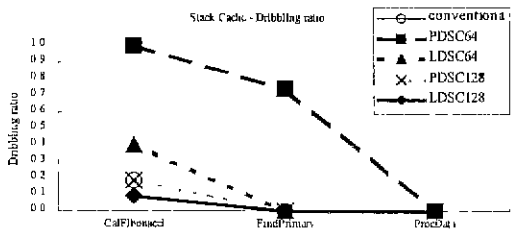
본 논문에서는 기존 알고리즘을 선택하여, 자바로 벤치마크를 직접 제작하여. 이들 벤치마크로 시뮬레이션을 수행하였다. 벤치마크들은 자바 프로그램의 특성을 고려하여 선택되었으며, 대다수의 자바 프로그램에 포함되어 있고, 자바의 가장 뚜렷한 특성인 오브젝트(메모리) 관련, 메소드 관련, 계산 관련 기능들을 고려하였다. 이들 세 가지 분야의 특징을 지니는 프로그램들 중에서 대표로 선정된 벤치마크들은 [표 2]와 같다

[표 2] 시뮬레이션 벤치마크들

벤치마크	중심 부분
소수 검색	산술 연산
멀티미디어 데이터 처리	메모리(오브젝트)
피보나치 순열 계산	메소드 호출

4.3. 이중 스택 캐쉬의 성능

드리블링은 스택 캐쉬와 데이터 캐쉬사이의 데이터 이동을 필요로 한다. 그러므로, 드리블링을 처리하는 동안에는 명령어의 수행이 잠시 중단된다. 즉, 드리블링의 발생 횟수는 프로세서의 수행 성능 저하율로 나타난다. 따라서 스택 캐쉬의 성능 비교의 척도로써 드리블링 발생 빈도를 측정하였다



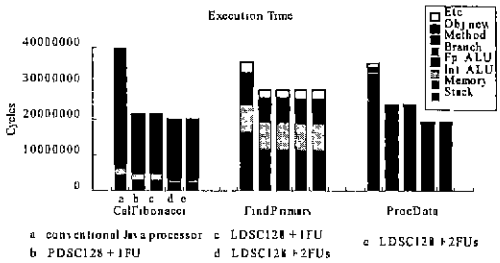
[그림 5] 이중 스택 캐쉬의 수행 성능

LDSC의 경우, 데이터 레지스터의 크기를 기존 picoJava-I의 크기로 유지할 경우, 드리블링 횟수가 증가하게 되지만, PDSC보다는 우수한 성능을 보여준다 [그림 5] 이는 LDSC에서 표시기가 PDSC와는 달리 두 개의 스레드가 함께 사용함으로, 드리블링이 발생할 확률이 감소하기 때문이다 12항 크기의 데이터 레지스터를 가지는 LDSC의 경우에도 PDSC와 같은 성능을 보여준다 이는 두 스레드에서 각 각 64항 이상의 스택 캐쉬를 사용하는 경우에 대한 드리블링이 발생하기 때문이다.

4.4. 전체적 성능

실험 결과를 종합해 보면, SMTJC은 LDSC나 PDSC의 구조에서 128항으로 구성될 때 1.27~1.85의 성능 향상으로 기장 좋은 성능을 보여주었다 [그림 6] 이들은 자바 프로그램의 메소드와 스택관련 명령어들의 수행 시간을 감소시켜준다. 추가적인 처리 유닛은 메모리 참조와 정수 연산의 수행 시간을 줄여주며, 이는 전체 수행 성능을 5~30%정도 향상 시켜준다.

자바 프로그램에서 가장 빈번하게 발생하는 부분들에 중심을 둔 세 가지 벤치마크에 대한 시뮬레이션을 통해, 이중 스레드를 동시에 수행하는 SMTJC은 자바 프로그램에서 기존 프로세서보다 1.28~2.00의 성능 향상이 있음을 알 수 있다.



[그림 6] 동시 다중스레드 자바칩의 수행 성능

5. 결론

기존 자바 프로세서의 성능을 향상시키는 방법으로 다중스레드를 직접 지원하는 새로운 자바 프로세서로 SMTJC을 제안했다. SMTJC은 명령어 캐쉬는 서로 독립적인 두 개의 스레드를 동시에 수행함으로써, 자바 프로그램에서의 명령어 수준 병렬성(Instruction level parallelism)을 향상시킨다. 다중 스레드 수행을 위해 새로운 스택 캐쉬의 구조 및 운영 방법을 사용한다. JavaSim을 통한 시뮬레이션은 SMTJC은 기존 자바 프로세서에 비해 이중 스택 캐쉬와 추가적 처리 유닛들로 인해, 1.28~2.00의 전체 수행 성능 향상이 있음을 보여준다.

본 연구는 하드웨어와 소프트웨어의 상호 보완적인 기술적 경향을 배경으로 자바의 언어적 특성을 고려한 프로세서를 설계, 지원함으로써 자바 프로세서의 성능 향상을 도모하고 있다

참고 문헌

[1] J.Michael et al., "picoJava-I : The Java Virtual Machine In Hardware." *IEEE Micro*, pp 45-53, March/April 1997

[2] Lee-Ren Ton et al., "Instruction Folding in Java Processor." 1997 *Int'l Conf on Parallel and Distributed Systems*, pp 138-143, Dec 1997

[3] Anant Agarwal, "Performance Tradeoffs in Multithreaded Processors," *IEEE Tran on Parallel and Distrib Sys*, Vol 3, No 5, pp 525-539, Sep 1992

[4] Hiroaki Hirata et al., "An Elementary Processor Architecture with Simultaneous Instruction Issuing from Multiple Threads." *19th Ann Int'l Sym on Comp Arch*, pp. 136-145, May 1992

[5] Susan J. Eggers et al., "Simultaneous Multithreading : A Platform for Next-Generation Processors," *IEEE Micro*, pp.12-19, Sep/Oct 1997

[6] Dean M. Tullsen et al, "Simultaneous Multithreading : Maximizing On-Chip Parallelism," *22nd Ann Int'l Sym on Comp Arch*, pp. 392-403, June 1995

[7] Dean M. Tullsen et al., "Exploiting Choice : Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor," *23rd Ann Int'l Sym on Comp Arch*, pp 191-202, May 1996

[8] Jack L Lo et al, "Tuning Compiler Optimizations for Simultaneous Multithreading," *30th Ann Int'l Sym on Microarchitecture*, pp 114-124, Dec 1997.

[9] Tim Lindholm, Frank Yellin, "The Java Virtual Machine Specification," *Addison-Wesley*, 1996