다중스레드 모델의 스레드 코드를 자바 바이트 코드로 변환하기 위한 번역기 설계

김 기 태°, 이 갑 래*, 양 창 모**, 유 원 희
인하대학교 전자계산공학과
*김천과학대학
**청주 교육대학교
g1991251@inhavision.inha.ac.kr

Design of Translator for generating Java Bytecode from Thread code of Multithreaded Models

Ki-Tae Kim°, Kab-Lae Lee*, Chang-Mo Yang**, Weon-Hee Yoo
Department of Computer Science and Engineering, Inha University
*Department of Computer Science and Information Processing Kimcheon College of Science
**Chongju National University of Education

요 약

다중스레드 모델은 데이터플로우 모델의 내부적인 병렬성, 비동기적 자료 가용성과 폰 노이만 모델의 실행 지역성을 결합하여 병렬처리 시스템의 성능을 향상 시켰다. 이 모델은 프로그램의 실행을 위하여 컴파일러에 의해 생성된 스레드를 수행하며, 스레드의 생성 방법에 따라 자원 활용 빈도나 동기화 빈도와 같은 스레드의 질이 결정되는 특징이 있다. 하지만 다중스레드 모델은 실행 모델이 특정 플랫폼에 제한되는 단점을 가지고 있다. 이에 반해 자바는 플랫폼에 독립적인 특징을 가지고 있어 다중스레드 모델의 스레드 코드를 실행 단위인 자바 언어로 변환하면 다중스레드 모델의 특징을 여러 플랫폼에서 수정 없이 사용할 수 있게 된다.

자바는 원시 언어를 중간 언어 형태의 바이트 코드로 변환하여 각 아키텍처에 맞게 설계된 자바 가상 머신이 실치된 시스템에서 자바 언어를 수행한다. 이러한 자바 언어의 바이트 코드는 번역기의 중간 언어와 같은 역할을 수행하고, 자바 가상 머신은 번역기의 후위부와 같은 역할을 한다.

본 논문은 다중스레드 코드가 플랫폼에 독립적인 특성을 갖출 수 있도록 다중스레드 코드를 자바 가상 머신에서 실행 가능하도록 한다. 즉, 다중스레드 모델의 스레드 코드를 자바 바이트 코드로 변환하는 번역기를 설계, 구현하고, 자바 가상 머신의 실행을 분석한다.

1. 서 론

다중스레드 모델은 폰 노이만(von Neumann) 모델의 실행 지역성과 데이터플로우 모델의 비동기적 자료 가용성, 내재적 병렬성을 혼합한 모델로 확장형 병렬 기계(scalable parallel machine)를 구성하는데 적당한 실행 모델이다. 다중스레드 모델에서 계산 단위는 논리적으로 연관된 순차 명령으로 구성되는 스레드이다. 다중스레드 모델은 수행에 필요한 모든 값이 사용할 수 있을 때 동적으로 스케줄링 되는 데이터플로우 모델의 스케줄링 정책에 따라 스케줄링 되며, 스레드가 스케줄링 되면 스레드를 구성하는 모든 명령어들은 폰 노이만 모델의 수행 방식에 따라 스레드의 마지막 명령까지 중단 없이 수행된다. 하지만 이러한 특징을 가진 다중스레드 모델은 이 모델이 실행되는 플랫폼에 따라 다른 실행 모델을 가져야 한다. 그 이유는 레지스터의 개수나 명령어의 구조 등이 머신마다 모두 다르기 때문이다. 하지만 적절한 실행 모델을 머신마다 개발하기에는 많은 비용과 어려움이 따른다. 따라서 현대의 많은 컴파일러 개발자나 시스템 프로그래머, 프로그래밍 언어 개발자에게는 효과적인 중간 언어를 개발하는 것과 여러 아키텍처에 맞는 머신을 개발하는 것이 중요하다.

번역기는 중간 언어를 기준으로 전위부와 후위부로 구분할 수 있는데, 전위부는 에러를 조작하기 위해 사용되고, 특별한 머신 구조에 맞게 코드를 생성하기 위해서는 후위부를 사용한다. 따라서 전위부에서 개발된 중간 언어는 후위부가 개발된 어떠한 머신에서도 동작할 수 있다[4, 5, 6, 7, 8]. 이러한 특성으로 인해 전위부에서 개발된 중간 언어는 후위부가 개발된 머신에서 머신 독립적으로 존재할 수 있고 컴파일러가 새로운 구조에 맞게 이식하는 과정을 쉽게 한다. 따라서 언어 개발자와 컴파일러 개발자는 올바른 중간 언어를 결정하는 것이 중요한 과제이다[5]. 이와 같은 구조를 갖는 언어에는 P-code와 선 마이크로시스템(Sun Microsystems. Inc.)에서 개발한 자바 언어(Java language) 또는 자바 바이트 코드(Java Bytecode) 등이 있다[1, 2, 3]. 자바는 원시 언어를 중간 언어 형태의 바이트 코드로 변환하여 각 아키텍처에 맞게 설계된 자바 가상 머신(Java Virtual Machine)이 설치된 시스템에서 자바 언어를 수행할 수 있도록 한다[2, 3]. 이러한 자바 언어의 바이트 코드는 번역기에 있는 중간 언어와 같은 역할을 수행하고, 자바 가상 머신은 번역기의 후위부와 같은 역할을 수행한다.

스레드 코드의 실행을 자바 바이트 코드로 변환할 수 있다면 다양한 플랫폼에서 추가적인 비용 없이 처리할 수 있다. 그래서 스레드 코드를 자바 바이트 코드로 변환하는 번역기를 구현하는 방법은 매우 중요하다.

2. 다중스레드 코드와 자바

2.1 다중스레드 코드

다중스레드 모델(multithreaded model)은 데이타플로우 모델의 단점을 해결하기 위한 방안으로 프로그램에 내재한 병렬성을 최대한 활용하는 데이타플로우 모델과 지역성을 이용하여

본 논문의 연구는 1999년도 정보통신부 대학기초 연구지원 사업에 의해 수행되었음.

순차 코드를 효율적으로 수행할 수 있는 폰 노이만 모델의 혼합형 계산 모델이다. 다중스레드 모델은 순차적으로 수행되는 명령들을 스레드 단위로 묶어 폰 노이만 방식으로 수행하고, 스레드의 스케줄링으로 데이타플로우 모델의 스케줄링 방식을 사용하는 모델이다. 계산의 입자 크기를 증가시킴으로써 지역성의 장점을 얻고, 동기화가 스레드의 수행이 시작될 때만 발생하게 하여 동기화 비용을 감소시키는 등 순차 수행의 이점을 얻을 수 있으며 스레드 간의 병렬성을 활용할 수 있다[9]. 다양한 프로그래밍 언어에 대하여 다중스레드 모델로 변환할 수 있는데 특히, 다중스레드 모델을 위하여 함수 언어를 번역할 때 중요한 것은 순차적으로 수행될 수 있는 명령의 집합을 찾아내어 스레드로 분할하고, 동적 스케줄링은 이들 스레드 사이에서만 발생하도록 하는 것이다. 다중스레드 명령어의 구문구조는 [표1]과 같다.

```
<multithreaded code> ::= <opcode><operand>
<opcode> ::= <basicop>|<memory-registerop>|<value-argumentop>|<synchronizationop>
<basicop> ::= <arithmeticop>|<relationop>|<booleanop>|<bitstringop>
<arithmeticop> ::= <add>|<sub>|<inc>|<dec>|<mul>|<div>|<neg>
<relationop> ::= <gt>|<lt>|<ge>|<le>|<et>
<boolean> ::= <and>|<or>|<nor>
<bitstring> ::= <shtl>|<shtr>|<rotr>|<rotl>
<memory-registerop> ::= <store>|<load>
<value-argumentop> ::= <receive>
<synchronizationop> ::= <initsc>|<incsc>|<decsc>
<operand> ::= <regn>|<Fm>|<thrdN>
<regn> ::= reg|[reg]
```

[표 1] 다중스레드 명령어의 구문구조

기본연산에는 사칙연산과, 관계연산, 불린연산, 비트 연산에 관한 명령이 포함된다. 또 기본 연산의 오퍼랜드 부분에서는 2개 또는 3개의 레지스터를 사용한다. 메모리와 레지스터, 값과 인자, 동기화 연산에 대한 명령은 오퍼랜드 부분에서 레지스터와 프레임 메모리 그리고 스레드를 위한 동기화 카운터를 사용한다[9]. 이러한 다중스레드 모델에서 수행되는 스레드 코드는 [그림 1]과 같다.

```
# sum

thrd0:  receive   arg0
        stop
thrd1:  receive   arg1
        stop
thrd2:  load      arg0      R0
        load      arg1      R1
        add       R0        R1        R0
        store     R0        l_var0
        send      l_var0    pfp
        endf
```

[그림 1] 다중스레드 코드

[그림 1]의 스레드 코드는 함수언어로 작성한 두 수의 합을 구하는 프로그램을 스레드 코드로 변환한 것이다. [그림 1]은 세 개의 스레드로 구성된다. thrd0와 thrd1은 인자를 받아들이는 부분이고, thrd2는 앞의 두 개의 스레드에서 받아들인 인자를 이용해서 계산을 하는 스레드 코드이다.

2.2 자바와 자바 바이트 코드

자바 개발 환경에서는 컴파일 과정 후에 생성된 목적 코드가 모든 플랫폼에 적용될 수 있도록 코드를 생성하는데 이를 바이트 코드라 한다. 바이트 코드의 형태를 보면 다음과 같다.

No	opcode	operand

예를 들면 20 if_icomple 12 이다. 20번째 위치에 있는 바이트 코드는 정수형의 숫자를 비교해서 같으면 12바이트 뒤에 있는 위치로 분기하는 것이다. 이 형태는 앞에서 설명한 다중스레드 코드와 비슷한 형태이다. 자바는 원시 언어를 중간 언어 형태인 바이트 코드로 변환하여 자바 가상 머신에서 수행할 수 있도록 한다.

여러 가지 특징 중에 특히 바이트 코드의 번역, 다중스레드 지원, 플랫폼에 독립적으로 사용할 수 있는 특징이 다중스레드 모델의 스레드 코드를 자바 바이트 코드로 변환하는 동기가 된다. 하지만 위의 다중스레드 코드를 자바 가상 머신에서 수행하려면 동기화를 위한 클래스와 객체를 생성하는 클래스를 부가적으로 추가해주어야 한다.

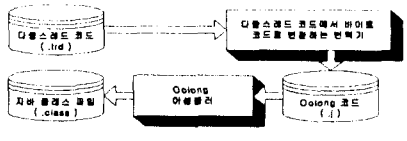
2.3 Oolong

Oolong은 자바 가상 머신을 위한 어셈블리어이고, 이 어셈블리어는 존 메이어(Jon Meyer)의 자스민(Jasmin) 언어를 기반으로 만들어진 언어이다[3]. Oolong 언어는 바이트 코드를 일일이 분석할 필요 없이 바이트 코드 수준에서 프로그래밍을 할 수 있도록 디자인 되어있다.

자바의 가상 프로세서는 스택을 기반으로 실행하고 실행 결과를 저장하기 위해 지역변수를 사용한다. 반면 다중스레드 코드는 레지스터를 기반으로 스레드 코드를 생성한다. 그래서 이 레지스터들은 자바 가상 머신에서 지역 변수로 대체 해야 하고, 이러한 과정을 통해서 .class 파일을 생성하는데 특별한 문제는 없다.

3. 스레드 코드를 자바 바이트 코드로 변환하는 번역기 설계

다중스레드 모델의 스레드 파일을 모든 플랫폼에 독립적으로 실행시키기 위해서는 스레드 코드를 바이트 코드로 변환해야 한다. 스레드 코드를 바이트 코드로 변환하는 단계는 다음과 같다. 우선, 생성된 스레드 코드를 읽어들여서 Oolong 코드로 변환한다. 그 후에, 생성된 Oolong 코드를 Oolong 컴파일러를 이용해서 자바 바이트 코드의 표현 형식인 .class 파일을 생성한다. [그림 2]는 다중 스레드 파일을 최종적으로 자바 클래스 파일로 변환되는 전체적인 과정도 이다.



[그림 2] 번역기 수행 단계

스레드 코드를 자바 바이트 코드로 변환하기 위해서는 스

[알고리즘 1] 다중스레드 코드를 Oolong 코드로 변환하는 알고리즘

Input : Multi-threaded code
Output : Oolong code
Temporary : Symbol table

Procedure
1. Repeat
2. *Read a token from Multi-threaded code*
3. *Store Symbol table with token*
4. if synchronization count = 0 then
5. if token = **thread number** then
6. *MyConsumer Oolong code initialization*
7. *MyProducer Oolong code initialization*
8. elseif token = **'receive'** then
9. *Make* ***synchronized*** *MyProducer thread*
10. elseif token = **'stop'** then
11. synchronization count = -1
12. elseif token = **'load'** then
13. *Make* ***synchronized*** *MyConsumer thread*
14. elseif token = **op-code** then
15. *Translate op-code to Oolong code*
16. elseif token = **'endf'** then
17. *stop*
18. Until end of input is reached

End-Procedure

레드 코드에 대응되는 자바 바이트 코드를 우선 정의하여야 한다. 대부분의 코드들이 비슷한 구문과 의미를 갖고 있지만 스레드 코드와 바이트 코드의 구문상의 차이로 몇 가지의 명령어는 새롭게 정의해주어야 한다. 스레드 코드를 바이트 코드로 변환하는 번역기의 알고리즘은 [알고리즘1]과 같다.

[알고리즘 1]에서 synchronization count는 각 스레드의 동기화 계수이고, 각 스레드는 동기화 계수가 0일 경우 실행된다. 0이 아닐 경우는 계속 대기한다. MyConsumer와 MyProducer 스레드는 ***synchronized***로 동기화를 갖게 하였다. [그림 1]에서 생성된 스레드 코드인 .thd를 Oolong파일인 .j로 만들어주고, 이것을 Oolong 어셈블러를 이용하여 .class 파일로 변환한다. 이렇게 해서 얻어낸 .class 파일은 일반적인 .class 파일과 동일한 형식으로 구성된다.

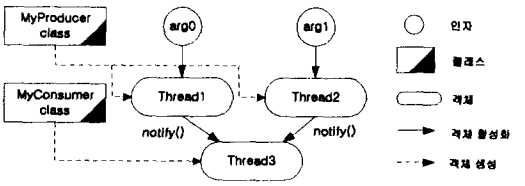
[알고리즘1]을 이용해서 구현한 본 논문의 번역기에서는 Oolong 어셈블리어 코드를 사용하였다. 읽어드린 스레드 코드를 이진수인 클래스 코드로 직접 변환할 수도 있으나 저급 수준의 최적화, 부가적인 작업환경 등을 고려할 수 있도록 Oolong 어셈블리어로 변환하는 단계를 거치도록 하였다. 그래서 자바 가상 머신 어셈블러인 Oolong을 사용하여 바이트 코드 수준에서 변환을 수행하도록 하였다.

```
.class super MyObject                    //MyObject 클래스
.super java/lang/Object
.method public <init> ()V
...
.end method
.method public synchronized setValue (I)V
...
.end method
.method public synchronized getA ()I
...
.end method
.method public synchronized getB ()I
...
.end method
.method public synchronized setC (I)V
...
.end method
.class super MyProducer                  //MyProducer 클래스
.super java/lang/Thread
.method public <init> (LMyObject;I)V
...
.end method
.method public run ()V
...
.end method
.class super MyConsumer                  //MyConsumer 클래스
.super java/lang/Thread
.method public <init> (LMyObject;)V
...
.end method
.method public run ()V
...
.end method
.class public super TAdd                 // 메인 클래스를 생성하는 부분
.super java/lang/Object
.method public <init> ()V
...
.end method
.method public static main ([Ljava/lang/String;)V
.limit stack 4
.limit locals 5
    new MyObject
    new MyProducer
    invokespecial MyProducer/<init> (LMyObject;I)V
    new MyProducer
    invokespecial MyProducer/<init> (LMyObject;I)V
    new MyConsumer
    invokespecial MyConsumer/<init> (LMyObject;)V
    invokevirtual java/lang/Thread/start ()V
    invokevirtual java/lang/Thread/start ()V
    invokevirtual java/lang/Thread/start ()V
.end method
```

[그림 3] [그림1]의 다중스레드코드를 번역기를 통해 변환한 Oolong 코드

[그림 3]의 코드는 [그림 1]의 다중스레드 코드를 [알고리즘 1]에 적용하여 변환한 Oolong 코드이다. 프로그램 처음부분에 정의되어있는 .class와 .super는 Oolong 프로그램을 정의하는 부분이고, method<init>는 코드를 초기화하는 부분 이다. .method<main>부분이 실제 프로그램이 작성되는 부분이다. MyObject 클래스는 객체 생성을 위해 만들어졌고, MyProducer와 MyConsumer 클래스는 Synchronized로 동기화된 계산을 처리하는 부분이다.

이렇게 생성된 코드를 Oolong 어셈블러를 이용해서 클래스 파일을 생성하면 [그림 1]의 코드와 같은 역할을 하는 .class 파일을 얻을 수 있다. 이렇게 생성된 .class 파일은 자바 언어의 특성을 가지며 플랫폼에 독립적인 특징을 가진다.



[그림 4] 번역된 클래스 파일의 실행 과정

[그림 4]는 생성된 클래스 파일이 실행되는 과정이다. 실제 여러 플랫폼에서 동작할 때 [그림 4]와 같이 세 개의 스레드가 생성되며 이것은 [그림 1]과 같은 결과를 산출한다. 이것은 차후 몇 가지의 조작을 하면 웹에서도 사용 가능하다.

4. 결론 및 향후 과제

본 논문은 프로그램에 내재한 병렬성을 최대한 활용하는 데이터플로우 모델과 지역성을 이용하여 순차 코드를 효율적으로 수행할 수 있는 폰 노이만 모델의 혼합형 계산 모델인 다중스레드 모델의 스레드 코드를 여러 플랫폼에 독립적인 자바 바이트 코드로 변환하는 번역기를 설계, 구현하였다. 자바 언어의 바이트 코드는 번역기에 있는 중간 언어와 같은 역할을 수행하고, 자바 가상 머신은 번역기의 후위부와 같은 역할을 수행하였다. 스레드 코드를 자바 바이트 코드로 변환할 경우 효율적인 코드 생성을 위해 Oolong 자바 가상 머신 어셈블러를 사용하였다. 이렇게 해서 생성된 .class 파일은 자바 가상 머신인 설치된 어떤 플랫폼에서도 실행가능 하였다.

본 논문에서 제안된 번역기는 다중스레드 코드를 자바 바이트 코드로 변환하는 부분에 중점을 두었다. 하지만, 속도를 위한 자바 바이트 코드 내에서의 최적화 부분에 대해서는 취급하지 못하였다. 이 부분은 향후 연구에서 개선되어야 한다.

<참고문헌>

[1] James Gosling, "Java Intermediate Bytecodes," ACM SIGPLAN Workshop on Intermediate Representations, 1995.

[2] Jon Meyer and Troy Downing, Java Virtual Machine, O'REILLY, 1997

[3] Joshua Engel , Programming for the Java Virtual Machine, Addison Wesley, 1999

[4] Gary Meehan, "Compiling Functional Programs to Java Byte-code," Department of Computer Science, University of Warwick, 1997

[5] Jonathan C. Hardwick and Jay Sipelstein, "Java as an Intermediate Language," School of Computer Science Carnegie Mellon University Pittsburgh, 1996

[6] Gray Meehan, Mike Joy, "Compiling Lazy Functional Programs to Java Bytecode," Department of Computer Science, University of Warwick, 1999

[7] Jack Pien "C Compiler Targeting the Java Virtual Machine," Computer Science Technical Report PCS-TR98-334, 1998

[8] Nick Benton, Andrew Kennedy, George Russell, "Compiling Standard ML to Java Bytecodes," Persimmon IT, Inc. Cambridge, U.K, 1998

[9] 유원희, 양창모, 주형석, 이갑래, "다중스레드 모델을 위한 비평가인자 함수 언어 번역기 설계 및 구현", 정보통신부 최종 결과 보고서, 1999