論文2000-37CI-4-1

Bytecode로부터 재목적 코드 생성 기법을 이용한 Pentium 코드 생성에 관한 연구

(A Study on the Pentium Code Generation using Retargetable Code Generation Technique from Bytecode)

鄭 成 玉*, 高 光 萬*, 李 聖 周**

(Sung-Ok Jung, Kwang-Man Ko, and Sung-Joo Lee)

요 약

인터넷 및 WWW의 급속한 성장은 이 기종 기계 및 분산 네트워크 환경에서 수행될 수 있는 응용 소프트웨어를 위한 프로그래밍 언어에 대한 연구를 부각시키고 있다. 이러한 이기종 기계 및 분산 네트웍 환경에서 응용 소프트웨어 개발을 위해 개발된 Java 언어는 객체지향 특성을 지원하는 언어이며 Java 프로그래밍 언어 환경에서는 이식성, 번역성, 고성능, 및 단순성 등을 지원하고 있다. Bytecode는 Java 언어의 중간 코드로서 이 기종 기계 및 분산 네트웍 환경의 다중 플랫폼 환경에서 다양한 응용 소프트웨어의 개발을 가능하게 하고 있다. 하지만 Bytecode는 인터프리터 기법으로 실행되는 특성 때문에 많은 실행 시간을 소비하는 단점을 가지고 있다. 본 연구에서는 Bytecode로부터 정형화된 방법으로 다양한 목적기계에 대한 코드를 생성하기 위해 재목적 코드 생성 시스템을 설계하고 구현하고자 한다. 특히, Java 컴파일러로부터 생성된 Bytecode로부터 실질적으로 Pentium 코드를 생성하는 시스템을 구현한다. 본 연구의 원활한 수행을 위해 컴파일러 자동화 도구인 ACK의 코드 생성 시스템을 기반으로 한다.

Abstract

The massive growth of the internet and the world-wide-web leads us to research the programming languages for the development of applications in heterogeneous, network-wide distributed environments. Java is an object-oriented language for such a environment and the Java programming language environment provides a portable, interpreted, high-performance, simple programming language. Bytecode is an intermediate code for Java language and it enables the development of applications on multiple platform in heterogeneous, distributed networks. But it takes much time to execute Bytecode because of using an interpretation method.

In this paper, we design and implement a retargetable code generation system which can be systematically reconfigured to generate code for a variety of distinct target computers. From the system, we realize the code generation system which translates the Bytecode being produced by Java compiler into Pentium target code. We use ACK code generation system to do the work easily.

* 正會員, 光州女子大學校 컴퓨터學部
(Kwang-Ju Women's University, Division of Computer)

** 正會員, 朝鮮大學校 컴퓨터工學部
(Chosun University, Division of Computer Engineering)

接受日字:2000年 1月 12日, 수정완료일:2000年 7月4日

Ⅰ. 서 론

자바 프로그래밍 언어는 인터넷 및 분산 환경 시스템에서 효과적으로 응용 프로그램을 작성할 수 있도록 설계된 언어로서 이 기종간의 실행 환경에 적합하

도록 설계된 Bytecode를 중간 언어로 사용한다. Bytecode는 자바 프로그래밍 환경에서 사용하는 중간 코드로서 서로 다른 실행 환경에서 독립적으로 사용된다. 이러한 Bytecode를 실행하는 방법은 그림 1과 같이 인터프리터, Just-In-Time 방식 또는 컴파일 기법을 이용하여 특정 목적기계에 대한 목적기계 코드를 생성하는 Back-end 방식으로 구분된다. 최근에는 인터프리터 방식으로 수행되는 Bytecode에 대한 실행 속도의 한계와 네트워간의 속도를 개선하기 위해 Bytecode에 대한 목적기계 코드로의 변환과 실행 환경이전에 관한 연구가 진행중이다.[17][18]

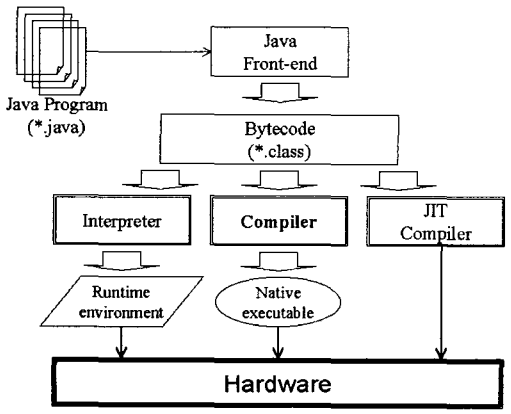

그림 1. Bytecode 실행 방식
Fig. 1. Execution model of Bytecode.

자바 가상 기계(Java Virtual Machine; JVM)에서 사용하는 Bytecode는 자바 가상 기계 명령어, 심벌 테이블, 그 외에 부수적인 정보를 가지고 있는 클래스 파일 형식으로서 보안성을 위하여 클래스 파일의 코드가 엄격한 형식과 구조적인 제약을 가지고 있도록 하고 있다.[15] Bytecode의 구성은 연산자를 기술하기 위해 1 바이트 크기를 갖는 "opcode" 부분과 연산자에 의해 사용되어질 인자 및 데이터를 기술하는 "operands"로 구성되어진다. Bytecode는 이 기종 기계 및 분산 환경에서 독립적으로 실행될 수 있는 장점을 갖지만 각 플랫폼에서 인터프리터 방식으로 실행되므로 실행 속도면 에서 단점을 가지고 있다. 하지만 Bytecode를 특정 기계에 대한 목적기계 코드로 변환하여 실행하면 번역시간이 소비되는 단점을 갖지만 번역이 완료된 후 실행 시간에서는 인터프리터 방식에 비해 10~50배 정도의 향상을 가져온다. 따라서 자바 프로그램을 효율적으로 실행하기 위해서는 인터프리터와 JIT 방식을 혼용함으로써 이 기종 기계에서 독립적으로 실행되며 인터프리터만을 사용하는 시스템에 비해 실행 속도가 증가하기 위해 중간 코드인 Bytecode를 특정기계 대한 목적기계 코드로의 변환 과정이 요구된다.[17][18]

재목적 가능 코드 생성 시스템(Retargetable code generation system)은 컴파일러 후단부에서 다양한 목적기계에 대해 원시 프로그램의 중간 코드를 목적기계 코드로 바꾸는 과정을 정형화 기법을 통하여 자동적으로 구성하는 것이다. 따라서 코드 생성 과정을 목적기계 의존적인 부분과 목적기계 독립적인 부분으로 나누어 정형화하는데 목적을 두고 있으며 템플리트 매칭(Template matching) 또는 테이블을 이용한 방법(Table-driven method)에 의해 목적기계에 대한 코드를 생성한다. 목적기계 의존적인 부분에서는 목적기계 표현 테이블을 입력으로 받아 목적기계에 대한 정보 및 코드 생성 규칙을 저장하고 있는 테이블을 출력한다. 목적기계 독립적인 부분에서는 원시 언어의 중간 코드 입력을 테이블을 참조하여 하나의 공통된 코드 생성 알고리즘을 이용하여 실질적으로 목적기계 코드를 생성하게 된다.[5][8][11]

최근에는 Java 언어를 실행 속도의 한계를 개선하기 위해 Java 언어의 중간 코드인 Bytecode(*.class)에 대한 특정 목적기계 코드로의 변환 기법에 관한 연구가 진행중이다. Bytecode를 특정 기계에 대한 목적기계 코드로 변환하여 실행하면 번역시간이 소비되는 단점을 갖지만 번역이 완료된 후 실행 시간에서는 인터프리터 방식에 비해 10~50배 정도의 향상을 가져온다. 따라서 자바 프로그램을 효율적으로 실행하기 위해서는 인터프리터와 JIT 방식을 혼용함으로써 이 기종 기계에서 독립적으로 실행되며 인터프리터만을 사용하는 시스템에 비해 실행 속도가 증가하기 위해 중간 코드인 Bytecode를 특정기계 대한 목적기계 코드로의 변환 과정이 요구된다.

본 논문에서는 Java 언어의 Bytecode로부터 정형화된 코드 생성 기법을 적용하여 다양한 목적기계 코드를 생성할 수 있는 재목적 코드 생성 시스템을 설계하고 구현하고자 한다. 특히, 실질적인 목적기계 코드 생성 과정을 구현하기 위해 Bytecode로부터 Pentium 코드를 생성하는 코드 생성 시스템을 구현한다. 이를 위해 Bytecode에 대한 Pentium 코드 생성 정보를 기술한 코드 생성 규칙을 정형화된 방법으로 기술하며

양질의 코드 생성을 위한 코드 생성 기법을 제시한다. 본 연구를 원활히 수행하기 위해 컴파일러 자동화 도구인 ACK의 코드 생성 시스템과 코드 확장기 모델에 기반을 둔다.

ACK에서는 EM 중간 코드를 사용하여 다양한 목적기계에 대한 코드를 생성한다. ACK의 중간 코드로 사용되는 EM 코드는 가상 스택 기계에 기반을 둔 EM 기계의 어셈블리 언어이다. 후단부는 목적기계에 대한 코드를 생성하는 부분으로서 ACK의 코드 생성 시스템은 코드-생성기 생성기와 코드 생성기로 구성되어 있다. 본 논문에서는 코드-생성기 생성기와 코드 생성기로 구성된 ACK 후반부 모델을 적용하며 EM 중간 코드를 Java 언어의 중간 코드인 Bytecode로 대체한다. 따라서 각각의 Bytecode에 대한 목적기계 코드 생성 정보를 기술하며 생성된 목적기계 코드 생성 정보를 이용하여 컴파일러 후단부의 핵심인 코드 생성기가 Java 언어로 작성된 원시 프로그램의 Bytecode(*.class)를 입력을 받아 목적기계 코드를 생성한다.

본 논문의 구성은 제 2장에서 재목적 코드 생성 시스템의 특징 및 재목적 코드 생성 기법에 대해 고찰한다. 제 3장에서는 실질적으로 Bytecode로부터 Pentium 코드 생성하는 모델 및 생성 과정을 기술한다. 제 4장에서는 결론 및 향후 연구 방향에 대해 기술한다.

Ⅱ. 재목적 코드 생성 기법

1. 재목적 코드 생성 시스템

재목적 코드 생성 시스템(retargetable code generation system)은 컴파일러 후단부에서 다양한 목적기계에 대해 원시 프로그램의 중간 코드를 목적기계 코드로 변환하는 과정을 정형화된 방법을 통하여 자동적으로 구성하는 것이다. 따라서 재목적 코드 생성 시스템에서는 코드 생성 과정을 목적기계 의존적인 부분과 목적기계 독립적인 부분으로 나누어 정형화하는데 목적을 두고 있으며 템플리트 매칭 또는 테이블을 이용한 방법에 의해 코드를 생성한다. 목적기계 의존적인 부분에서는 목적기계 표현 테이블을 입력으로 받아 목적기계 정보 및 중간 코드에 대한 목적 코드 생성 정보를 저장하고 있는 테이블을 출력해 준다. 목적기계에 대해 독립적인 부분에서는 중간 코드를 입력으로 받아 테이블 정보를 참조하여 하나의 공통된 코드 생성 알고리즘을 이용하여 실질적인 목적기계 코드를 생성한다.[5][8][11]

재목적 코드 생성 시스템은 목적기계 표현 테이블, 코드-생성기 생성기, 코드 생성기로 구성되어 있다. 목적기계 표현 테이블에는 특정 기계에 대한 정보 및 중간 코드에 대한 목적 코드 생성 규칙을 기술한다. 코드-생성기 생성기는 목적기계 표현 테이블을 입력으로 받아 코드 생성 시에 참조될 수 있는 정보를 테이블 형태로 생성한다. 코드 생성기는 중간 코드를 입력으로 받아 실질적으로 목적기계 코드를 생성하는 부분으로서 하나의 공통된 코드 생성 알고리즘을 이용한다. 재목적 후단부를 구현하기 위해 제시된 목적기계 코드 생성 기법으로는 해석적 코드 생성, 패턴 매칭 코드 생성, 테이블을 이용한 코드 생성 방법으로 구분된다.

2. ACK 코드 생성 시스템

ACK(Amsterdam Compiler Kit)[11][12]는 컴파일러의 후단부를 자동화하기 위한 도구의 하나로서 이식성과 재목적성이 매우 높은 컴파일러를 만들기 위한 실질적인 도구이며 다양한 특성을 갖는 컴파일러를 구성하기 위해 여러 단계의 조합으로 구성된다. ACK의 중간 코드로 사용되는 EM 코드는 가상 스택 기계에 기반을 둔 EM 기계의 어셈블리 언어이다. 후단부는 목적기계에 대한 코드를 생성하는 부분으로서 ACK의 코드 생성 시스템은 그림 2와 같이 코드-생성기 생성기와 코드 생성기로 구성되어 있다. 코드-생성기 생성기는 EM 코드 패턴에 대한 목적 코드 생성 규칙을 기술한 목적 기계 표현 테이블을 입력으로 받아 코드 생성시에 필요한 정보를 생성한다. 코드 생성기는 실질적인 EM 코드를 입력으로 받아 목적 기계에 대한 코드를 생성하는 부분으로서 EM 코드 패턴에 해당되는 목적 코드

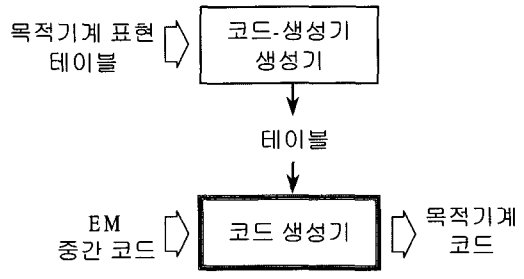

그림 2. ACK 코드 생성 시스템
Fig. 2. ACK Code Generation System Model.

생성 정보를 결정하기 위해 스트링 패턴 매칭 코드 생성 알고리즘(string pattern matching code generation algorithm)을 적용한다. 이러한 매칭 기법은 지나친 검색 동작을 수행하므로 비효율적이다[11].

Ⅲ. 재목적 코드 생성 기법을 이용한 Pentium 코드 생성 시스템

1. 시스템 개요

본 논문에서는 ACK 코드 생성 시스템 모델을 기반으로 해서 현재 인터넷 분산 환경 시스템에서 널리 사용되고 있는 Java 언어에 대한 중간 코드(*.class)인 Bytecode로부터 특정 목적기계에 대한 코드를 생성기법을 제시하고 실질적으로 Pentium 기계에 대한 코드를 생성한다. 특히, Bytecode는 스택 지향 구조로서 목적기계 코드 생성시에 가상 스택(fake stack)을 이용하며 각각의 명령어에 대해 목적기계 코드를 생성하는 규칙을 정형화하게 기술한다. 전체적인 시스템의 구성은 그림 3과 같이 Bytecode에 대한 Pentium 코드 생성 규칙을 기술한 목적기계 표현 테이블(Machine Description Table; MDT), 코드-생성기 생성기(Code-Generator Generator; CGG), 코드 생성기(Code Generator; CG)로 구성된다.

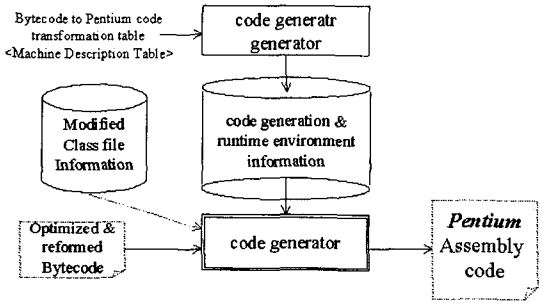

그림 3. Pentium 코드 생성 시스템
Fig. 3. Pentium Code Generation System Model.

목적기계 표현 테이블에는 Bytecode를 목적기계 코드로 변환시켜 주는데 필요한 목적기계 특성 및 코드 생성 규칙을 기술한다. 즉, Bytecode와 목적기계 코드 간의 변환을 정의하는 부분으로서 각각의 Bytedcode에 대한 Pentium 생성 정보를 기술한다. 코드-생성기 생성기는 목적기계 표현 테이블을 입력으로 받아 코드 생성기가 목적기계 코드 생성시에 필요한 정보를 테이블 형태로 출력하며 단일 패스로 처리된다. 테이블을 생성하기 위해서는 먼저 입력으로 들어오는 목적기계 표현 테이블에 대한 구문 분석과 의미 분석을 수행한 후 테이블 정보를 파일에 직접 생성한다. 코드 생성기는 Bytecode를 입력으로 받아 코드-생성기 생성기에 의해 생성된 테이블 정보를 참조하여 실질적으로 Pentium 목적기계 코드를 생성한다.

2. 클래스 파일 및 목적기계 표현 테이블

클래스 파일은 16비트, 32비트, 64비트 크기는 2개, 4개, 8개의 연속적인 바이트를 읽어서 8비트 스트림으로 구성된다. 여러 개의 바이트로 구성된 데이터 아이템은 상위 바이트가 먼저 오는 순서로 저장되며 구조체를 이용하여 그림 4와 같이 하나의 ClassFile 구조체로 표현된다.

```
ClassFile {
    u4 magic;
    // . . .
    u2 constant_pool_count;
    cp_info constant_pool[constant_pool_count-1];
    // . . .
    attribute_info attributes[attributes_count];
}
```

그림 4. 클래스 파일 구조체
Fig. 4. Class File Structure.

클래스 파일에서 상수 기억 장소인 constant_pool은 Class, Fieldref, Methodref와 같은 타입을 가지고 있다. 속성 테이블인 attributes 필드에는 클래스 파일에서 미리 정의된 Sourcefile, Costantvalue, Code, Exceptions, Linenumbertable, Localvariabletable에 대한 구조체로 구성되어 있다. 클래스나 인터페이스의 메소드에 대해 기술되어 있는 method_info 타입의 구조체에는 메소드에 대한 접근 권한을 갖는 access_flags와 메소드의 이름을 나타내기 위해 상수 기억 장소에 대한 인덱스를 갖는 name_index, 자바 메소드 기술자를 나타내기 위해 상수 기억 장소에 대한 인덱스를 갖는 descriptor_index가 있다. 또한, 메소드가 갖는 attribute 테이블의 수를 나타내는 attributes_count, 속성 테이블이 기술되어 있는 attributes로 구성되어 있으며, 이 attributes에 있는

Code 구조체에 Bytecode가 저장되어 있다. Code 속성을 갖는 구조체는 그림 5와 같다.

```
Code_attribute{
    u2 attribute_name_index;
    // . . .
    u2 exception_table_length;
    {
        // . . .
    }
    // . . .
    attribute_info attributes[attributes_count];
}
```

그림 5. 코드 속성 구조체
Fig. 5. Code Attribute Structure.

Bytecode의 피연산자는 컴파일 시간에 결정되며 실행 시간에 실질적으로 계산되어 결과값이 스택에 저장되는 스택 지향적 구조를 가진다. Bytecode의 데이터 형에는 정수형에 속하는 byte, short, int, long, char 형이 있고, 실수형에 속하는 float, double 형이 있다.

Bytecode 테이블은 코드-생성기 생성기에 의해 참조되어 중간 코드인 Bytecode를 목적기계 코드인 Pentium 코드로 확장하는데 필요한 정보로 구성된다. Bytecode에 대한 코드 생성 규칙을 정형화된 방법으로 기술하기 위해 그림 6과 같은 규칙을 따른다. 목적기계 표현 테이블 기술시에 Bytecode의 스택 지향 구조인 특성을 레지스터 지향 구조인 Pentium 코드로 변환하기 위해 레지스터 할당을 위한 정보와 생성될 목적 코드가 기술된다.

```
Table ::= (RULE)*
RULE ::= C_instr(COND_SEQUENCE|SIMPLE)
COND_SEQUENCE ::= (condition SIMPLE)*
"default" SIMPLE
SIMPLE ::= "==>" ACTION_LIST
ACTION_LIST ::= [ACTION( ; ACTION)*] .
ACTION ::= AS_INSTR | function_call
AS_INSTR ::= """[labe":"] [INSTR]"""
INSTR ::= mnemonic[operand( , operand)*]
```

그림 6. 목적기계 표현 테이블 문법
Fig. 6. Grammar of Machine Description Table.

목적기계 표현 테이블 기술 문법을 적용하여 그림 7과 같이 실질적으로 테이블의 일부분을 기술하였다.

```
/* Pushing Constants onto the Stack      */
C_bipush ==>
code_combiner(
    ;
    push_const($1))
.
/* Loading Local Variable onto the Stack */
C_iload ==>
code_combiner(
;
    {
        reg_t S1;
        if (S1 = find_local($1, 0)) {
            soft_alloc_reg(S1);
            push_reg(S1);
        } else {
            soft_alloc_reg(reg_lb);
            push_reg(reg_lb);
            inc_tos($1);
            push_const(4);
            C_los(4);
        }
    })
```

그림 7. 목적기계 표현 테이블 예
Fig. 7. Example of Machine Description Table.

3. 코드 생성기

코드 생성기는 코드-생성기 생성기에서 생성된 Bytecode에 대한 Pentium 코드 생성 정보를 참조하여 입력으로 들어오는 Bytecode에 대한 Pentium 코드를 실질적으로 생성하는 부분이다. 현재 구현된 시스템은 중간 코드로 들어오는 Bytecode 각각에 해당하는 Pentium 코드로 변환되는 기법(1:1 mapping)을 적용하고 있다. 따라서 각각의 Bytecode에 대해 하나 이상의 Pentium 코드로 그림 8과 같이 변환된다.

목적기계 테이블에는 각각의 Bytecode에 대한 Pentium 코드 생성 규칙 및 정보가 기술되며 코드-생성기 생성기는 이러한 정보를 입력으로 받아 코드 생성시에 필요한 정보를 생성한다. 그림 8에서는 "iload" Bytecode에한 Pentium 코드 생성 정보를 입

력으로 받아 코드 생성시에 필요한 정보 및 루틴을 생성한다. 실질적인 Pentium 코드 생성은 코드 생성기가 중간 코드인 "iload 0" Bytecode를 입력을 받아 "Pop R1" Pentium 코드를 생성한다. 이와 같은 방법으로 코드 생성기는 각각의 Bytecode로 구성된 *.class 파일을 입력을 받아 각각의 Bytecode에 대한 Pentium 코드를 생성한다.

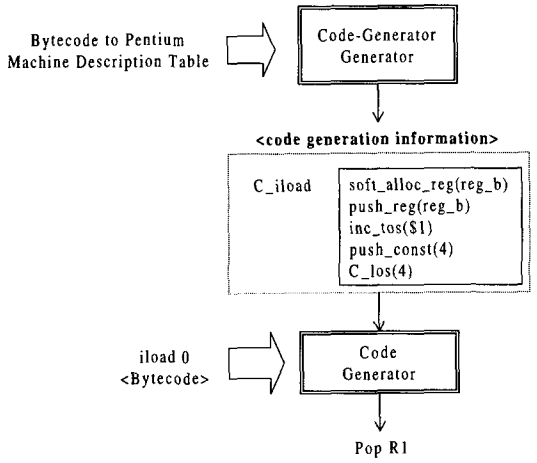

그림 8. Bytecode로부터 Pentium 생성
Fig. 8. Pentium Code Generation from Bytecode.

Pentium 목적기계에 종속적인 루틴 작성시 자바 가상 기계는 스택 구조에 기반을 둔 가상 기계이므로 스택을 이용하여 매개변수를 전달한다. 특히, 자바 가상 기계는 다양한 목적기계로의 이식을 용이하게 하기 위해 레지스터를 사용하는 대신 스택과 지역 변수를 사용한다. 하지만 Pentium 목적기계는 레지스터 기반 기계이므로 상태 정보를 유지하기 위해서 특정 레지스터를 지정하여 이용한다. 따라서 코드 생성시에 스택 포인터 레지스터와 프레임 포인터 레지스터의 상대적 주소 값을 계산하여 적재와 저장 연산을 수행하는 스택으로 대체한다.

Ⅳ. 결론 및 향후 연구 과제

자바 프로그래밍 언어는 인터넷 및 분산 환경 시스템에서 효과적으로 응용 프로그램을 작성할 수 있도록 설계된 언어로서 이기종 기계간의 실행 환경에 적합하도록 설계된 Bytecode를 중간 언어로 사용한다. Bytecode는 자바 프로그래밍 환경에서 사용하는 중간 코드로서 서로 다른 실행 환경에서 독립적으로 사용된다. 최근에는 인터프리터 방식으로 수행되는 Bytecode에 대한 실행 속도의 한계와 네트워간의 속도를 개선하기 위해 Bytecode에 대한 목적기계 코드로의 변환과 실행 환경이전에 관한 연구가 진행중이다. Bytecode를 특정 기계에 대한 목적기계 코드로 변환하여 실행하면 번역시간이 소비되는 단점을 갖지만 번역이 완료된 후 실행 시간에서는 인터프리터 방식에 비해 10~50배 정도의 향상을 가져온다. 따라서 Java 프로그램을 효율적으로 실행하기 위해서는 인터프리터와 JIT 방식을 혼용함으로써 이 기종 기계에서 독립적으로 실행되며 인터프리터만을 사용하는 시스템에 비해 실행 속도가 증가하기 위해 중간 코드인 Bytecode를 특정기계 대한 목적기계 코드로의 변환 과정이 요구된다.

재목적 가능 코드 생성 시스템(Retargetable code generation system)은 컴파일러 후단부에서 다양한 목적기계에 대해 원시 프로그램의 중간 코드를 목적기계 코드로 바꾸는 과정을 정형화 기법을 통하여 자동적으로 구성하는 것이다. 따라서 코드 생성 과정을 목적기계 의존적인 부분과 목적기계 독립적인 부분으로 나누어 정형화하는데 목적을 두고 있으며 템플리트 매칭(Template matching) 또는 테이블을 이용한 방법(Table-driven method)에 의해 목적기계에 대한 코드를 생성한다. 목적기계 의존적인 부분에서는 목적기계 표현 테이블을 입력으로 받아 목적기계에 대한 정보 및 코드 생성 규칙을 저장하고 있는 테이블을 출력한다. 목적기계 독립적인 부분에서는 원시 언어의 중간 코드 입력을 테이블을 참조하여 하나의 공통된 코드 생성 알고리즘을 이용하여 실질적으로 목적기계 코드를 생성하게 된다.

본 논문에서는 Java 언어의 Bytecode로부터 코드 생성 기법을 적용하여 다양한 목적기계 코드를 생성할 수 있는 재목적 코드 생성 시스템을 설계하고 구현하고자 한다. 특히, 실질적인 목적기계 코드 생성 과정을 구현하기 위해 Bytecode로부터 Pentium 코드를 생성하는 코드 생성 시스템을 구현하였다. 이를 위해 Bytecode에 대한 Pentium 코드 생성 정보를 기술한 코드 생성 규칙을 기술하며 목적기계 코드 생성을 위한 코드 생성 기법을 제시한다. 본 연구를 원활히 수행하기 위해 컴파일러 자동화 도구인 ACK의 코드 생성 시스템과 코드 확장기 모델에 기반을 두었다.

앞으로 생성된 코드의 실행을 위해 자바의 실행 환

경을 각 기계의 실행 환경으로 변환이 요구되며 이를 위한 다양한 연구를 진행할 예정이다. 특히, 현재까지는 단순한 프로그램에 대해서만 Pentium 코드 생성을 실험하였지만 보다 많은 코드 생성 규칙을 목적기계 표현 테이블에 기술하여 복잡한 프로그램에 대해서도 실험할 예정이다. 또한 생성된 코드의 질을 평가하기 위한 성능 분석 실험을 통해 보다 양질의 코드 생성을 위한 연구를 진행할 예정이다. 본 연구를 통해서 Java 언어를 기존의 인터프리터 수행 방식에 비해 특정 목적기계에서 효율적으로 수행할 수 있는 장점을 갖을 수 있다.

참 고 문 헌

[1] Alfred V. Aho, Mahadevan Ganapathi, Steven W. K. Tjiang, "Code Generation Using Tree Matching and Dynamic Programming," ACM TOPLAS, Vol. 11, No. 4., pp.491-516, Oct., 1989.

[2] Anant Agrawal, Robert B. Garner and Donald C. Jackson, "SPARC:An ASIC Solution for High-Performance Micropro- cessors," The SPARC Technical papers, Sum Microsystems Inc., 1991.

[3] R. G. G. Cattell, "Automatic Derivation of Code Generators from Machine Descriptions," ACM TOPLAS, Vol. 2, No. 2, pp.173-190, Apr., 1980.

[4] Christoph M. Hoffmann and Michael J. O'Donnell, "Pattern Matching in Trees," Journal of the Association for Computing Machinery, Vol. 29, No.1, pp. 68-95, Jan., 1982.

[5] Mahadevan Ganapathi, Charles N. Fischer, John L. Hennessy, "Retargetable Compiler Code Generation," ACM Computing Surveys, Vol. 14, No. 4, pp.573-592, Dec., 1982.

[6] Susan L. Graham, "Table-driven Code Generator", IEEE Computer, Vol. 13, No. 8, pp25-34, Aug., 1980.

[7] Philip Homburg, Raymond Michiels, "A Fast Backend for SPARC Processor," report-81, Netherlands Vrije Universiteit, 1989.

[8] Frans Kaashoek, Koen Langendoen, "The Code Expander Generator," Dept. of Mathematics and Computer Science, Vrije Universiteit, 1989.

[9] Karen A. Lemone, Design of Compilers : Techniques of Programming Language Translation, CRC Press, 1992.

[10] Richard P. Paul, "SPARC Architecture, Assembly Language Programming, and C," Prentice-Hall, 1994.

[11] Hans van Staveren, "The table driven code generator from ACK 2nd. Revision," report-81, Netherlands Vrije Universiteit, 1989.

[12] Andrew S. Tanenbaum, Hans van Staveren and Johan W. Stevenson, "A Practical Tool Kit for Making Portable Compilers," CACM, Vol. 26, No. 9, Sep., 1983.

[13] Jon Meyer, Troy Downing, "Java Virtual Machine", O'REILLY, 1997.

[14] Peter van der Linden, "just JAVA", Prentice-Hall, 1997.

[15] Tim Lindholm, Frank Yellin, "The Java Virtual Machine Specification" ADDISON-WESLEY, 1996.

[16] Christopher W. Fraser, David R. Hanson, Todd A. Proebsting, "Engineering a Simple, Efficient Code Generator", ACM Letters on Programming Languages and System, pp. 231-226, Sep., 1982.

[17] Wen-Mei W. Hwu, "IMPACT: An Architecture Framework for Multiple-Instruction Issue Processors", Proceeding of the 18th International Symposium on Computer Architecture, pp. 266-275, 1991.

[18] Wen-Mei W. Hwu, "Java Bytecode to Native Code Translation: The Caffeine Prototype and Preliminary Results", The Procedding of the 29th Annual International Symposium on Microarchitecture, Dec., 1996.

[19] William A. Wulf, "An Overview of the Production Quality Compiler-Compiler Project", IEEE Computer, Vol. 13, No. 8, pp. 38-49, Aug., 1980.

저 자 소 개

鄭 成 玉(正會員)

1958년 3월 27일생. 1987년 2월 조선대학교 전자계산학과 졸업(이학사). 1989년 2월 조선대학교 전자계산학과 졸업(이학석사). 2000년 조선대학교 전자계산학과(박사과정 수료). 1992년 3월~1997년 2월 광주여자전문대학 전산정보처리과 조교수. 1997년 3월~현재 광주여자대학교 컴퓨터학부 조교수 재직. 주관심분야는 소프트웨어 공학, 객체지향 소프트웨어, 시스템 소프트웨어, 프로그래밍 언어론, 시스템분석 설계

李 聖 周(正會員)

1970년 2월 한남대학교 물리학과 졸업(이학사). 1992년 2월 광운대학교 전자계산학과 졸업(이학석사). 1998년 2월 대구효성카톨릭대학교 전자계산학과 졸업(이학박사). 1988년~1990년 조선대학교 전자계산소 부소장. 1995년~1997년 조선대학교 정보과학대학장. 1981년~현재 조선대학교 전자계산학과 교수 재직. 주관심분야는 소프트웨어 공학, 프로그래밍 언어 객체지향 시스템, 러프집합

高 光 萬(正會員)

1968년 2월 14일생. 1991년 2월 원광대학교 전자계산공학과 졸업(공학사). 1993년 2월 동국대학교 컴퓨터공과 졸업(공학석사). 1998년 2월 동국대학교 컴퓨터공학과 졸업(공학박사). 1998년 3월~현재 광주여자대학교 컴퓨터공학부 전임강사 재직. 주관심분야는 객체지향 프로그래밍 언어, 병행 프로그래밍 언어, 원도우즈 프로그래밍 언어, 시각 프로그래밍 언어, 컴파일러 구성론