# 패턴 매칭 기법을 이용한 SIL 코드 최적화기*

박성환º, 라황균, 오세만
동국대학교 컴퓨터공학과
e-mail : {hadler,amaranth,smoh}@dongguk.edu

# SIL Code Optimizer Using Pattern Matching Technique

Sung-Hwan Parkº, Hwang-Gyun La, Se-Man Oh
Dept. of Computer Engineering, Dongguk University

**요 약**

EVM(Embedded Virtual Machine)은 모바일 디바이스, 디지털 TV 등 임베디드 컴퓨팅 환경에서 동적인 응용프로그램을 실행할 수 있는 가상기계 플랫폼(Virtual Machine Platform)이다. 가상기계를 이용한 응용프로그램은 플랫폼 독립적인 실행 및 효과적인 다운로드 솔루션을 통한 동적인 실행이 가능하다. EVM을 위한 가상기계 코드인 SIL(Standard Intermediate Language)은 언어/기계 독립적으로 설계되었다.

본 논문은 SIL 코드가 시스템 리소스의 제한이 큰 임베디드 시스템상에서 보다 효율적으로 실행되기 위하여 최적화를 수행하였다. 기존의 최적화 방법론에 관한 연구를 통하여 SIL 코드 특성을 고려한 최적화 방법론을 제시하고, 최적화된 코드를 생성하기 위한 코드 최적화기를 설계하고 구현하였다. SIL 코드 최적화기는 컴파일러에 의해 생성된 SIL 코드를 입력으로 받아 효율적인 코드로 변환하여, 전체 코드의 크기를 줄이고 수행 속도의 개선효과를 얻을 수 있다.

## 1. 서론

가상기계는 하드웨어로 이루어진 물리적 시스템과는 달리 소프트웨어로 제작되어 논리적인 시스템 구성을 갖는 개념적인 컴퓨터이다. 가상기계 기술을 이용하면 응용 프로그램 실행 환경인 프로세서나 운영체제가 변경되더라도 응용 프로그램을 수정하지 않고 사용할 수 있는 장점을 가진다.

이러한 가상기계는 기존의 PC 환경뿐만 아니라 모바일 장치, 디지털 TV, 셋탑 박스 등의 임베디드 시스템으로 적용분야가 확장되고 있기 때문에 더욱 더 그 중요성이 부각되고 있다[6].

임베디드 시스템이란, 전용 동작을 수행하거나 특정 임베디드 응용 프로그램과 함께 사용되도록 설계된 컴퓨팅 시스템을 말한다. 임베디드 시스템을 위한 가상기계 기술은 모바일 장치와 디지털 TV 등에 탑재할 수 있는 핵심 기술로서 다운로드 솔루션에서는 꼭 필요한 소프트웨어 기술이다.

EVM 이라 명명된 임베디드 시스템을 위한 가상기계 솔루션은 언어/기계 독립적으로 설계된 SIL 이라는 가상기계 코드를 사용한다. SIL 코드는 임베디드 시스템을 위한 가상기계의 표준 중간 언어로서 다양한 프로그래밍 언어를 수용할 수 있으며, 객체지향 언어와 절차 언어를 모두 수용하기 위한 연산 코드 집합을 가지고 있다. SIL 은 연산코드가 객체지향적으로 설계되어 타입정보가 없었으나, 효율적인 실행을 위하여 연산 코드에 타입을 추가하여 확장된 SIL (Extended SIL)로 재정의 하였다[7].

가상기계에서 실행되는 SIL 이 갖춰야 할 가장 기본이 되는 필요충분조건은 '경량화'와 '고속화'라 말할 수 있다. 임베디드 시스템의 제한된 메모리와 저속 CPU 환경에서 원활히 수행되기 위해서는 가상기계가 끊임없이 경량화되고 고속 동작을 보장해야만 한다.

본 논문에서는 기존의 최적화 방법론에 관한 연구를 토대로 확장된 SIL 코드의 특성을 고려한 최적화 방법론을 제시하고, 최적화된 SIL 코드를 생성하기 위

* 본 연구는 한국과학재단 목적기초연구 (R01-2002-000-00041-0)지원으로 수행되었음.

한 코드 최적화기를 설계하고 구현하였다.

## 2. 배경 연구

### 2.1 SIL

SIL 은 임베디드 시스템을 위한 가상기계의 표준 중간 언어로 설계되었으며 인터프리테이션을 목적으로 정의된 코드이다. SIL 의 특징은 다양한 프로그래밍 언어를 수용하기 위해서 기존의 가상기계 어셈블리 언어들의 분석을 토대로 정의하였으며, 객체지향 언어와 순차적인 언어를 모두 수용하기 위한 연산 코드 집합을 가지고 있다.

SIL 은 의사코드와 연산코드로 이루어져 있다. 의사코드는 클래스 등 특정 작업의 수행을 의미하고, 연산코드는 가상기계에서 실행되는 실제 명령어로 스택 기반의 명령어 집합이며, 특정 프로그래밍 언어에 종속되지 않는 언어 독립성과, 하드웨어 및 플랫폼 독립적인 하드웨어 독립성을 가지고 있다. 또한 연산 코드의 니모닉은 특정 하드웨어나 소스 언어에 종속되지 않는 추상적인 형태를 지니고 있다.

SIL 코드는 객체지향 언어와 순차적 언어를 모두 수용하기 위한 연산 코드 집합으로 이루어져 있으며 어셈블리 언어 수준의 디버깅을 용이하게 하기 위해 일관성 있는 이름 규칙을 적용하여 코드의 가독성을 확보하고 있다[6].

SIL 은 연산코드가 객체지향적으로 설계되어 타입정보를 가지지 않았었다. 그러나 코드의 효율적인 실행을 위하여 연산 코드에 타입을 추가하여, 확장된 SIL 로 재 정의하였다. 본 논문에서는 확장된 SIL 을 가지고 최적화를 수행하였다.

### 2.2 최적화 방법론

코드 최적화란 주어진 입력 프로그램과 의미적으로 동등하면서 효율적인 코드로 바꾸는 것을 의미한다. 따라서, 코드 최적화기는 가급적 계산의 횟수를 줄이고, 보다 빠른 명령을 사용하여 실행 시간이 짧은 코드를 생성해야 하며, 기억 장소의 요구량을 최소화해야 한다.

원시 프로그램에 대한 컴파일과정 중 최적화 단계에서는 프로그램의 실행 속도를 개선시키고 코드 크기를 줄일 수 있는 다양한 최적화 기법을 수행한다. 일반적으로 최적화는 컴파일 과정에서 선택적인 단계로 필요 시에만 컴파일러 옵션으로 선택하여 실행할 수 있다. 최적화를 수행하는 주요한 기준은 첫째, 반드시 입력과 의미적으로 동등해야 한다. 둘째, 평균적으로 속도가 빨라져야 한다. 셋째, 최적화에 들인 노력에 대한 Tradeoff 가 있어야 한다[8].

대표적인 최적화 방법론이 패턴 매칭 방법이다. 패턴 매칭 방법은 이식성을 위한 최적화 방법으로서, 최적화 알고리즘 자체로부터 기계 표현을 분리하는데 중점을 두었다. 이 방법의 장점은 모든 기종에 대하여 하나의 최적화 알고리즘을 사용할 수 있는 점이다.

패턴 매칭 방법을 좀 더 개선한 방법이 트리를 이용한 패턴 매칭 방법이다. 트리 패턴 매칭 방법은 Weingart 에 의해 처음으로 소개되었다. 목적 기계 명령어 집합을 패턴 매칭하는데 쉽게 하기 위해 트리를 이용하여 패턴 매칭을 한다. 트리 패턴 최적화기는 패턴을 받고 동등한 패턴을 찾기 위해 패턴 트리를 검색하는 일종의 트리 운행기이다.

## 3. SIL 코드 최적화 시스템 설계

### 3.1 자료구조

SIL 코드는 크게 데이터 부분과 코드부분으로 나누어 볼 수 있다. 데이터 부분은 함수의 선언등과 같은 선언부에 대한 정보를, 코드 부분은 함수 안의 문장부 및 함수 안에서 선언된 정보를 가지고 있다. 본 논문에서 제안하는 자료구조는 SIL 코드 한 라인의 정보를 모두 저장하고, 저장된 정보들을 순서 있게 저장하기 위하여 이중 연결 리스트 구조를 사용하였다. [그림 1]은 정해진 자료구조를 도식화 한 것이다.

본 논문이 제안하는 자료구조에서 다양한 형태의 SIL 코드를 구별하기 위하여 codeShape 라는 변수를 두고 있다. 0~6 번까지는 컴파일러가 생성한 코드를, 7~199 번까지는 최적화 코드를, 200 번은 파일의 맨 끝을 알리는데 사용하고 있다.

```
typedef struct silNode {
    char label[5]; // label 정보
    char instruction[10]; // instruction 정보
    char type[5]; // instruction type 정보
    int baseFirst, offsetFirst, baseSecond, offsetSecond, baseThird,
    offsetThird, baseFourth, offsetFourth;
    // 주소 정보를 기억하기 위한 공간
    // (2 ~ 4번은 최적화를 위한 부분)
    int value; // code value에 대한 정보
    char jumpLabel[10000]; // 점프 label 및 디버깅 정보
    int codeShape; // sil code의 형태정보를 가리킴
    // 출력형태를 결정하기 위해 사용되는 정보
    struct silNode *pre; // 전 노드를 가리킴
    struct silNode *next; // 다음 노드를 가리킴
} SIL;
```

**[그림 1] 최적화기의 자료구조**

### 3.2 최적화 알고리즘

SIL 코드의 최적화에 적용되는 최적화 알고리즘은 패턴을 탐색하는 부분과 최적화 패턴을 적용하는 부분으로 구성되어 있다.

SIL 코드 최적화기의 목적은 컴파일러가 생성한 코드들을 조작하여, 동등한 의미를 가지면서 실행 효율이 높은 코드들로 바꾸는데 있다. SIL 최적화기는 SIL 코드에 대응하는 최적의 패턴을 찾는 방법론으로 패턴 매칭을 적용하고 있다.

본 논문에서 일반적으로 사용되는 최적화 기법에는 도달할 수 없는 코드의 제거, 수행에 불필요한 코드의 제거, 중복 된 코드의 제거, 상수들로 구성된 연산의

제거, 제어 관련 최적화 등이 있다. 또한 SIL 코드 의존적인 최적화 기법으로는 스택 관련 최적화, 배정 관련 최적화, 배열관련 최적화 등을 수행하였다.

### 3.3 최적화 패턴

패턴 매칭으로 수행되는 최적화기의 성능은 패턴 내용에 크게 좌우되기 때문에 패턴 작성은 매우 중요하며, 최대한의 최적화 효과를 반영할 수 있도록 구성되어야 한다. 최적화 패턴은 입력된 SIL 코드 명령어 집합을 향상된 SIL 코드로 교체하기 위한 수단으로써 사용된다.

패턴에 반영된 최적화 내용은 도달할 수 없는 코드의 제거, 스택 관련 연산, 산술/논리 연산, 배정 연산, 제어 연산 등을 포함하는 패턴으로 구성되어 있다. [그림 2]는 최적화에 사용될 SIL 코드 패턴을 보여주고 있다.

```
// 산술 연산
    [lod.s] c1 [lod.s] c2 [add.s]: [addv.s] c1 c2
……

// 배정
    [ldc.l] c1 [str.l] c2: [setvc.l] c2 c1
……

// 제어
    [lod.ui] c1 [lod.ui] c2 [eq.i] [fjp] $$x
     : [lod.ui] c1 [lod.ui] c2 [jne.ui] $$x
……

// 배열
    [lda] c1 [ldc.i] c2 [ldc.i] c3 [mul.i] [cvi.ui] [cvui.p] [add.p]
[ldi.i]
     : [ldele.i] c1 c2
……

// 파라메터 최적화
    [ldc.i]c1: [ldc.i.1] (단, c1의 값이 1인 경우)
```

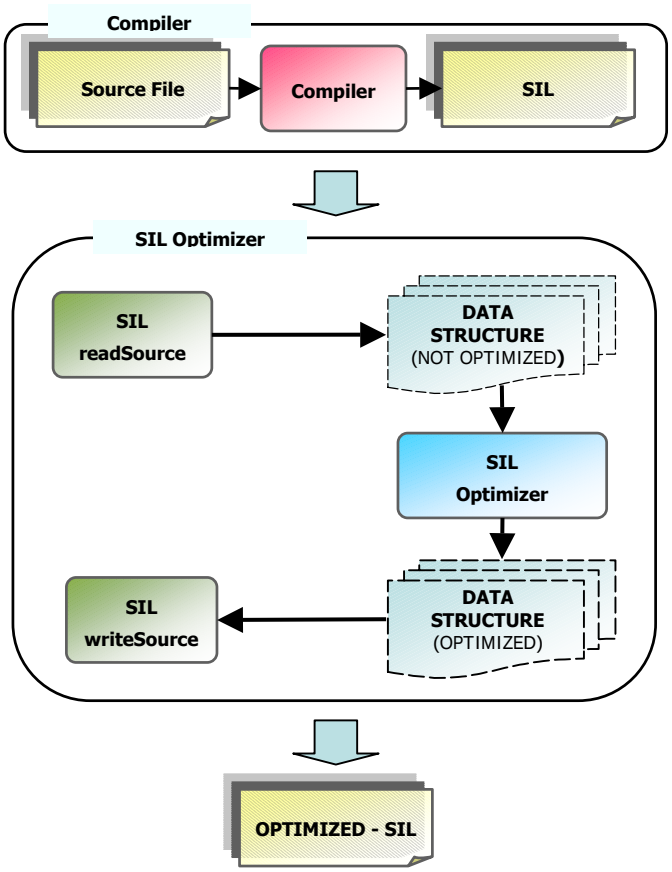
**[그림 2] SIL 코드에 대한 패턴**

## 4. SIL 코드 최적화 시스템 구현

본 절에서는 SIL 코드 패턴을 찾고, 찾은 패턴을 최적화된 코드로 바꾸는 SIL 코드 최적화기의 시스템 구성 모델에 대해 언급하고 있다.

### 4.1 시스템 구성도

소스 프로그램을 컴파일러가 동등한 의미를 가지는 SIL 코드로 변환한다. SIL 코드 최적화기는 컴파일러의 결과물로 나온 SIL 을 입력으로 받아, 동등한 의미를 가지면서 원본 코드보다 '저용량'과 '고속성'을 가지는 최적화된 SIL 코드로 변환한다. [그림 3]는 이러한 과정을 도식화한 것이다.




**[그림 3] SIL 코드 최적화기 구성도**

### 4.2 최적화기 시스템

코드최적화 시스템은 read, optimize, write 부분으로 구성되어 있다. read 부분은 1 라인의 SIL 코드를 읽어서 정해진 자료구조에 1 라인의 모든 정보를 저장한다. 이렇게 저장된 정보는 linker()에 의해 입력 순서대로 정렬이 된다. optimize 부분은 패턴을 찾고, 패턴부분을 패턴에 해당하는 최적화 코드로 패턴부분을 대치한다. 최적화 패턴에 해당하는 부분을 찾으면, 최적화 코드를 정해진 자료구조에 저장한다. 패턴에 해당하는 코드들의 집합을 제거하고 적용한 최적화 코드부분을 연결한다. 이러한 패턴 탐색 및 변환동작을 코드의 끝까지 반복한다.

write 부분은 이렇게 최적화된 자료구조를 읽어 들여 파일로 출력하는 역할을 한다. [그림 4]는 최적화기를 들어가기 전의 SIL 파일이고, [그림 5]는 최적화기를 통하여 나온 최적화된 SIL 파일이다.

```
24 %Line 17 : disc = stack[from][top[from]--];
25   lda 0 3
26   lod.i 1 0
27   ldc.i 256
28   mul.i
29   cvi.ui
30   cvui.p
31   add.p
32   lda 0 771
33   lod.i 1 0
34   ldc.i 4
35   mul.i
36   cvi.ui
37   cvui.p
38   add.p
39   ldi.i
40   dup
41   ldc.i 1
42   sub.i
43   lda 0 771
44   lod.i 1 0
45   ldc.i 4
46   mul.i
47   cvi.ui
48   cvui.p
49   add.p
50   sti.i
51   ldc.i 4
52   mul.i
53   cvi.ui
54   cvui.p
55   add.p
56   ldi.i
57   str.i 1 8
```

[그림 4] 최적화되기 전의 SIL 파일

```
24 %Line 17 : disc = stack[from][top[from]--];
25   lda 0 3
26   lod.i 1 0
27   ldc.i 256
28   mul.i
29   cvi.ui
30   cvui.p
31   add.p
32   ldele.i 0 771 1 0
33   dup
34   ldc.i.1
35   sub.i
36   sti.i
37   ldc.i 4
38   mul.i
39   cvi.ui
40   cvui.p
41   add.p
42   ldi.i
43   str.i 1 8
44 %Line 18 : stack[to][++top[to]] = disc;
45   lda 0 3
46   lod.i 1 4
47   ldc.i 256
48   mul.i
49   cvi.ui
50   cvui.p
51   add.p
52   ldele.i 0 771 1 4
53   ldc.i.1
54   add.i
55   sti.i
56   lda 0 771
57   lod.i 1 4
58   ldc.i 4
59   mul.i
```

[그림 5] 최적화된 SIL 파일

## 5. 실험결과

본 논문에서 구현한 SIL 코드 최적화기의 성능을 평가하기 위한 실험으로 SIL 코드의 사이즈 및 SIL 코드 라인 수를 비교 측정하였다. 가상기계의 특징은 단순한 코드 여러 개보다 복잡한 코드 한 개를 실행하는 게 더 빠르다. 따라서 최적화 코드의 라인 수도 최적화율을 측정하는 중요한 기준이 된다.

다음 [표 1]은 최적화 전의 SIL 코드와 최적화 후의 SIL 코드의 크기 및 라인 수를 비교한 것이다.

[표 1] 최적화 전· 후의 비교 테이블

| *.sil | Code Size | | Line Count | |
|---|---|---|---|---|
| | 전 | 후 | 전 | 후 |
| Des.sil | 36,029 | 31,672 | 2,601 | 2,405 |
| Hanoi.sil | 3,938 | 3,498 | 243 | 197 |
| Hash.sil | 15,201 | 10,373 | 652 | 588 |
| Heap.sil | 7,543 | 4,922 | 340 | 201 |
| Magic.sil | 4,438 | 3,719 | 241 | 196 |
| Merge.sil | 6,491 | 5,195 | 421 | 265 |
| Perfect.sil | 1,847 | 1,565 | 77 | 64 |
| Prime.sil | 1,806 | 1,593 | 74 | 65 |
| Qsort.sil | 3,435 | 2,877 | 205 | 142 |
| Span.sil | 15,444 | 12,059 | 1,003 | 627 |

실험 결과를 분석하면 실험에 이용되었던 SIL 파일들은 크기 및 라인 수에서 20~30% 경감율을 보여주었다. 특히, 배열연산이 많은 소스 프로그램일수록 다른 프로그램에 비해서 좋은 최적화율을 보여주고 있다.

## 6. 결론 및 향후 연구

본 논문에서는 기존의 최적화 기법들의 분석을 기반으로 하여, EVM 을 위한 중간 언어인 SIL 의 특성을 고려한 최적화를 수행하였다. SIL 코드 최적화기는 SIL 코드에 대한 각 명령어의 특성 및 EVM 에서 실행되는 동작을 분석하고, SIL 코드에 대한 최적화 기법들을 이용하여 공통 식의 제거, 연산 강도 경감, 루프 불변 코드 이동등과 같은 최적화를 진행하였다. 특히, SIL 코드에 의존적 최적화 기법인 배정, 스택, 배열에 관련된 최적화를 수행하였다.

향후 연구 방향은 SIL 코드 최적화기에 사용되는 코드 최적화 알고리즘을 보다 효율적으로 보완하여, 양질의 최적화된 코드를 생성하는데 있다.

## 참고 문헌


[1] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman, Compiler Principles, Techniques, and Tools, Addison-Wesley, 1985.

[2] Christopher Earnest, “Some Topics in Code Optimization”, Journal of the Association for Computing Machinery, Vol. 21, No.1, pp-76-102, Jan., 1974.

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

[4] 김정숙, 트리패턴매칭기법을 이용한 중간코드 최적화 시스템의 설계 및 구현, 동국대학교 박사학위 논문, 1998.

[5] 김은경, 패턴 매칭을 이용한 GVM SAL 코드 최적화, 동국대학교 석사학위 논문, 2005.

[6] 남동근, 가상기계를 위한 어셈블리 언어의 설계, 동국대학교 석사학위 논문, 2004.

[7] 남동근, “SIL Specification”, 신지소프트, Feb., 2005.

[8] 오세만, 컴파일러입문(개정판), 정익사, 2004.