GVM SAL 코드 최적화*

김은경º, 윤성림, 오세만
동국대학교 컴퓨터공학과
E-Mail : {plusek, yslhappy, smoh}@dongguk.edu

GVM SAL Code Optimization

Eun-Kyoung Kimº, Sung-Lim Yun, Se-Man Oh
Dept. of Computer Engineering, Dongguk University

요 약

GVM(General Virtual Machine)은 무선 단말기 상에서 동적인 응용프로그램을 실행할 수 있는 가상 기계(Virtual Machine) 플랫폼이다. 가상 기계를 이용한 응용프로그램의 실행은 플랫폼 독립적인 실행이 가능하며 또한 효과적인 다운로드 솔루션을 통한 동적인 실행이 가능하다. GVM 은 SGS 파일을 다운로드 받아 실행되는 시스템이므로, 성능의 저하없이 실행되기 위해서는 효율적인 최적화와 실행 시스템이 요구된다.

본 논문은 SGS 파일이 시스템 리소스의 제한이 큰 무선 단말기 상에서 보다 효율적으로 실행되기 위해서 SAL 코드에 대한 최적화를 수행하였다. SAL 코드 최적화 단계를 수행한 SGS 파일은 부분적으로 SGS 파일의 최적화를 가져와 전체 SGS 파일의 크기를 줄이고, 실행될 때 수행 속도 면에서 좀 더 빠른 실행 속도를 가지게 된다. 존재하는 최적화 방법론에 관한 연구를 통하여 SAL 코드의 특성을 고려한 최적화 방법론을 제시하고, 최적화된 SAL 코드를 생성하기 위한 코드 최적화기에 관하여 설계하고 구현하였다.

1. 서론

GVM(General Virtual Machine)은 무선 단말기 상에서 동적인 응용프로그램을 실행할 수 있게 순수 국내 기술로 개발된 가상 기계(Virtual Machine) 플랫폼이다. 가상 기계를 이용한 응용프로그램의 실행은 플랫폼 독립적인 실행이 가능하며 또한 효과적인 다운로드 솔루션을 통한 동적인 실행이 가능하다. GVM은 중간 형태의 SAL(Sinji Assembly Language) 코드를 생성하고, SAL 어셈블러를 통해 단말기용 게임 플러그인에서 동작하는 바이너리 형태의 SGS(Sinji Game Script) 파일로 변환된다.

GVM은 SGS 파일을 다운로드 받아 실행되는 시스템이므로, 성능의 저하없이 실행되기 위해서는 효율적인 최적화와 실행 시스템이 요구된다. 일반적으로 컴파일러는 원시 프로그램에 대한 컴파일과정 중 최적화 단계에서 프로그램의 실행 속도를 개선시키고 코드 크기를 줄일 수 있는 다양한 최적화 기법을 수행한다. GVM에서는 SAL 코드의 최적화를 통하여 무선 단말기 상에 SGS 파일을 제공하고자 한다.

본 논문은 네트워크 상에서 동적으로 다운로드되는 SGS 파일을 SAL 코드 수준에서 최적화하였다. 최적화된 SAL 코드들이 시스템에서 적은 네트워크 로드를 가지고 실행될 수 있도록 하며, 효율적인 실행 속도를 보이도록 하는 것이다. 즉, SGS 파일이 시스템 리소스의 제한이 큰 무선 단말기 상에서 보다 효율적으로 실행되기 위해서 SAL 코드에 대한 최적화를 수행하여 실행 시간을 개선하고 전체 SGS파일의 크기를 줄이게 된다.

존재하는 최적화 방법론에 관한 연구를 통하여 SAL 코드의 특성을 고려한 최적화 방법론을 제시하였다. 또한 최적화된 SAL 코드를 생성하기 위한 코드 최적화기에 관하여 설계하고 구현하였다.

본 논문의 구성은 2장에서는 본 연구의 기반이 되는 GVM, SAL 그리고 최적화 기법에 대한 관련 연구를 소개한다. 3장에서는 본 연구에서 설계하고 구현한 SAL 코드 최적화 시스템에 대해서 기술한다. 4장에서

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

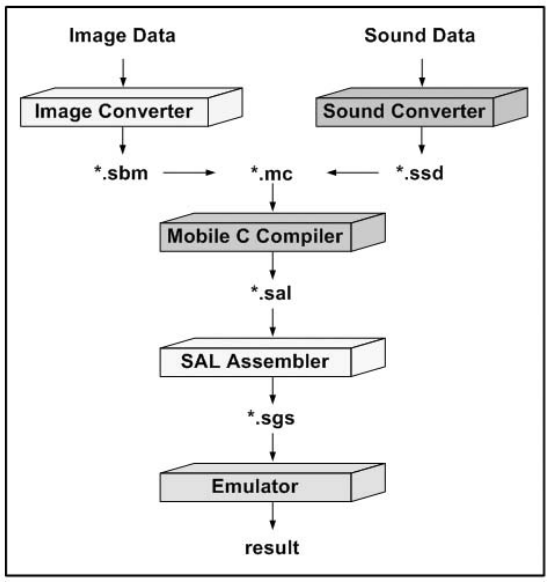
는 최적화 알고리즘의 성능을 평가하며 마지막으로 5장에서는 결론과 향후 연구 과제에 대해 기술한다.

2. 관련연구

2.1 GVM

GVM은 Mobile C로 개발한 프로그램을 단말기에 다운로드 받아 무선 단말기 상에서 동적인 응용프로그램을 실행할 수 있게 순수 국내 기술로 개발된 가상 기계 플랫폼이다. Mobile C는 표준 C언어를 기반으로 제한된 모바일플랫폼에 맞추어 응용프로그램을 작성할 수 있는 언어이다.

일반적인 방법으로 제작된 이미지와 사운드는 GVM SDK (GVM Software Development Kit)의 이미지와 사운드 제작도구를 통해 GVM 규격으로 변환한다. 변환된 소스를 Mobile C 소스파일(*.mc)에 포함시킨 후 Mobile C 컴파일러를 통해 중간 형태의 코드(*.sal)를 생성하고, SAL 어셈블러를 통해 가상 기계인 GVM에서 실행될 수 있는 코드(*.sgs)로 변환된다. [그림 1]은 이와 같은 과정을 나타내고 있다.



[그림 1] SGS 파일 작성 과정

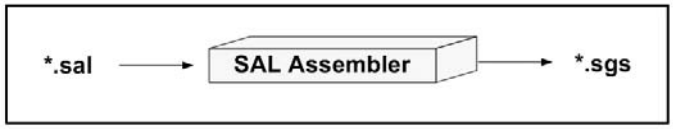
이렇게 작성된 모바일 응용 프로그램은 서버에 업로드 시킨 후 GVM이 내장된 단말기로 다운로드되어 실행되게 된다.

2.2 SAL

SAL은 GVM의 어셈블리 언어에 해당된다. 심볼 정의 및 프로그램 코드와 함께 이미지, 오디오 데이터를 들여오는(import) 기능을 포함하며, SGS의 헤더 생성을 위한 각종 데이터 정의 기능도 포함되어 있다.

SAL은 실제 실행이 되는 연산 코드(operation code)와 이에 대한 선언과 데이터를 지칭하는 의사 코드(pseudo code)로 분류된다. SAL의 연산코드는 218개이며 크게 5개의 카테고리로 나눌 수 있다. 각 카테고리는 스택 관련 연산, 산술/논리 연산, 배정(assignment) 연산, 제어 연산, 간접 연산이다.

SAL 프로그램은 SAL 어셈블러를 통해 단말기용 게임 플러그인에서 동작하는 바이너리 형태의 SGS 파일로 변환된다[그림 2].



[그림 2] SAL 어셈블러

2.3 최적화 방법론

코드 최적화란 의미적으로 동등하면서 좀 더 효율적인 코드를 생성해 내는 작업을 말한다. 즉, 가급적 계산의 횟수를 줄이고, 보다 빠른 명령을 사용하여 실행시간이 짧은 코드를 생성해야 하며, 기억용량의 요구량을 최소화하도록 해야 한다.

원시 프로그램에 대한 컴파일과정 중 최적화 단계에서는 프로그램의 실행 속도를 개선시키고 코드 크기를 줄일 수 있는 다양한 최적화 기법을 수행한다. 컴파일러 개발 과정에서 최적화 기법의 수행은 목적 기계 독립적인 중간코드 최적화(intermediate code optimization)와 목적 기계 의존적인 목적 코드 최적화(target code optimization)로 구분할 수 있다. 또한 중간 코드 최적화는 기본 블록을 기준으로 최적화하는 지역최적화와 프로그램 전체의 제어 흐름을 분석하여 수행하는 전역 최적화로 구분한다. 이러한 기법들에 대한 연구는 이루어져 있으나 SAL 코드의 특성을 고려한 최적화에 대해서는 좀 더 연구가 필요한 상황이다.

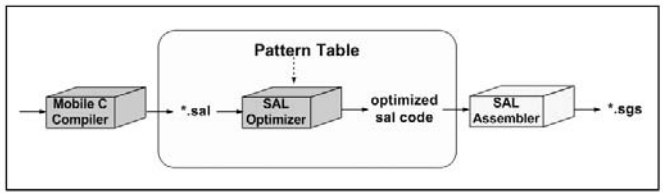
ACK에서는 패턴 테이블 생성기와 핍홀 최적화기에서 스트링 패턴 매칭 기법을 이용하여 EM 중간 코드에 대한 최적화 코드를 생성한다. 패턴 테이블 생성기에는 패턴 묘사 테이블의 내용을 파싱하고 (패턴:대치)쌍으로 구성된 입력을 받아들여 패턴 테이블을 출력한다. 패턴 테이블은 최적화기가 효율적으로 이용할 수 있도록 패턴 묘사 테이블을 바이트 배열 형태로 갖고 있는 테이블이다. 핍홀 최적화기에서는 구성된 패턴 테이블을 참조하여 입력으로 들어오는 EM 명령어에 대해 스트링 패턴 매칭 방법을 통해 최적화된 EM 명령어로 생성한다.

3. SAL 코드의 최적화

GVM 위에서 실행되는 SAL 코드는 기계 독립적인 특징을 가지고 있으므로 SAL 코드의 패턴을 분석하여 좀 더 코드의 크기를 줄이면서, 실행 효율을 높일 수 있다. 본 절에서는 하나의 SAL 코드로부터 새로운 최적화된 SAL 코드를 생성하는 데 있어서 SAL 최적화기의 시스템 구성 모델을 제시한다.

SAL 최적화기는 전단부에서 생성된 SAL 코드를 입력으로 받는다. 그리고, SAL 코드의 특성을 고려하여 작성된 패턴 테이블을 참조해서 일련의 비효율적인 코드를 좀 더 효율적인 코드로 대치하는 패턴 매칭

기법을 통해 최적화된 SAL 코드를 생성한다[그림 3].



[그림 3] SAL 코드 최적화 시스템 구성도

3.1 SAL 코드의 최적화 알고리즘

SAL 코드 최적화에 적용되는 최적화 알고리즘은 패턴을 탐색하는 부분과 기존 최적화 알고리즘을 적용하는 부분으로 구성된다.

SAL 최적화기의 목적은 기존 최적화 방법인 핍홀 최적화에 의하여 Mobile C 컴파일러가 일관성있게 생성한 코드들을 조작하여 좀 더 작은 크기의 SAL 코드 집합과 좀 더 실행 효율이 높은 코드들로 바꾸어 생성하는 데 있다. 또한 SAL 최적화기는 SAL 코드에 대응하는 최적의 패턴을 찾기 위해 패턴 매칭 기법을 적용하여 동등한 의미의 보다 효율적인 코드를 생성해 낸다.

특히 Mobile C의 특징적 요소를 사용하기 위하여 여러 가지 접근 방법을 사용하였다. 일반적으로 최적화에 사용되는 최적화 기법과 핍홀 최적화, 전역 최적화 방법들을 사용하여 직접 SAL 코드만을 분석하고 최적화하였다.

본 논문에서 일반적으로 사용되는 최적화 기법에는 도달할 수 없는 코드의 제거, 수행에 불필요한 코드의 제거, 중복된 코드의 제거, 상수들로 구성된 연산의 제거, 제어 관련 최적화 등이 있다. 그리고, SAL 코드 의존적인 최적화 기법으로는 스택 관련 최적화, 배정 관련 최적화 등을 수행하였다. 스택 관련 최적화는 스택 기반 언어인 SAL 코드가 수행한 스택에 대한 연산에 관련된 최적화를 뜻한다. 또한, Mobile C에서는 문자열, 이미지, 오디오, 보코더 데이터 처리를 새로운 media 데이터 타입으로 처리하며, media 타입의 초기값은 문자열을 제외하고 모두 배열 형식을 갖는다. 이러한 배열의 배정 연산자를 위한 특별한 명령어가 많은 점을 고려하여, 여러 개로 구성되어 있는 배정 표현을 하나의 코드로 변경하는 배정 관련 최적화를 수행하였다.

본 최적화기에서 작업을 수행하는 기본적인 흐름은 [그림 4]와 같다.

```
// 최초에 비교할 instrution을 판별
ptr = nextNode(head);
while(ptr != head)
{
    replace = 0;
    // 그에 해당하는 함수를 찾아서 호출
    switch (ptr->instr->opcode)
    {
        case A:
            // 나머지 instruction 판별해 최적화 수행
            replace = ptr_is_A();
            break;
            ...
            ...
            ...
    }
    if(replace == 0) ptr = nextNode(ptr);
}
```

[그림 4] SAL 최적화 알고리즘

본 최적화기는 프로그램의 흐름상 도달할 수 없는 코드를 먼저 제거한다. 그리고 SAL코드 특성을 고려하여 작성된 패턴 테이블을 참조해서 일련의 비효율적인 코드를 그에 상응하는 보다 효율적인 코드로 대체한다. 또한 수행에 불필요하거나 중복된 코드가 있으면 이를 찾아내어 제거하도록 하였다.

3.2 최적화 패턴

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

패턴에 반영된 최적화 내용은 도달할 수 없는 코드(unreachable code)의 제거, 스택 관련 연산, 산술/논리 연산, 배정 연산, 제어 연산 등을 포함하는 패턴으로 구성되어 있다. [그림 5]는 최적화에 사용될 SAL 코드 패턴을 보여주고 있다.

```
// 상수
    [pushc|pushw]c1 [pushc|pushw]c2 [add|sub|mult|div|mod]
    → [pushc|pushw] [c1+d2| c1-d2| c1*d2| c1/d2| c1%d2]
    ...
// 스택
    pushc c ldrz A add → ldric A c
    ldric A c ixa → pushic A c
    pushz B ldrz A ixa → pushi A B
    ...
// 배정
    pushc c popz A → zsetc A c
    pushz B popz A → zset A B
    pushic B c popz A → zsetn A B c
    ...
// 제어
    $$1 nop ujp $$2 → nop ujp $$2
    ujp $$1 $$1 nop → $$1 nop
    ...
```

[그림 5] SAL 코드에 대한 패턴

3.3 최적화 SAL 코드 생성

SAL 코드 최적화 단계를 수행한 SGS 파일은 부분적으로 SGS 파일의 최적화를 가져온다. 원시 SAL 코드로부터 SAL 최적화기를 통해 생성된 최적화된 SAL 코드 자신은 SGS 파일을 생성하는데 문제가 없다. 하

지만, 추가 코드 정보를 통해서 SAL 코드에서 SGS 파일로 변환하는 동안 변수명을 축약하고 SGS 파일을 생성한다. 이로써 전체 SGS 파일의 크기를 줄이고, 실행될 때 수행 속도 면에서 좀 더 빠른 실행 속도를 가지게 된다.

예를 들면, 생성된 NOP 명령어는 실행에 영향을 주지 않으므로 제거하여 최적화한다. 또한 여러 개로 구성되어 있는 배정 표현은 하나의 코드로 변경된다.

[그림 6]은 SAL 최적화기를 통해 생성된 최적화된 SAL 코드이다.

```
70              GetTime
71              ClearWhite
72              call    perfect:
73              Flush
74              ldrz    24
75              GetTime
76              call    Time:
77              end
78 perfect:     zsetc   16   2
79 $$0:         pushz   16
80              pushw   500
81              le
82              fjp     $$1:
83              zsetc   20   0
84              pushz   16
85              pushc   2
86              div
87              popz    18
88              zsetc   17   1
89 $$2:         pushz   17
90              pushz   18
91              le
92              fjp     $$3:
93              pushz   16
94              pushz   17
95              mod
96              stnz    19
97              nejp    0   $$4:
98              pushz   20
99              pushz   17
100             add
101             popz    20
102 $$4:        incz    17   1
103             ujp     $$2:
104 $$3:        pushc   1
105             pushc   10
106             pushc   0
107             DrawStr
108             pushz   16
109             pushz   20
110             eq
111             fjp     $$5:
112             incz    21   15
113             pushz   22
114             pushc   1
115             pushz   16
116             MakeStr1
117             pushc   1
118             pushc   10
```

[그림 6] 최적화된 SAL 파일

4. 실험 결과

본 논문에서 구현한 SAL 최적화기의 성능을 평가하기 위한 실험은 실행속도와 SAL 코드 크기의 변화를 비교 측정하였다. 실험 환경으로는 Intel Pentium-Ⅲ 1GHz, RAM 256MB, Windows XP Professional, GVM SDK version 1.52를 사용하였다. 실험 대상 데이터는 무작위로 추출한 일상적인 Mobile C 프로그램들과 본 최적화기의 패턴 테이블에서 중점을 둔 배열의 배정 관련 최적화를 테스트하기 위한 프로그램들을 선정해 사용하였다.

다음 [표 1]은 각각 Mobile C 컴파일러를 통해 생성된 SAL 코드와 SAL 최적화기를 통해 생성된 최적화된 SAL 코드를 입력으로 받아 생성된 SGS 파일을 비교 분석한 것이다. SGS 코드 크기의 경감과 실행 속도의 경감을 통한 비교 실험을 하였다.

[표 1] 최적화 전· 후 비교 테이블

(*.sgs)	Code Size [byte]			Execution Time [msec.]		
	전	후	경감율	전	후	경감율
perfect	288	269	7 %	47	44.3	6 %
bubblesort	486	345	29 %	119	114	4 %
prime	248	233	6 %	32	31	3 %
matmul	412	329	21 %	103	95.1	8 %
calculator	203	202	1 %	21	21	0 %
infinityarith	97	64	36 %	14	12.6	1 %
biroperator	88	81	8 %	11	11	0 %
eightqueens	512	396	23 %	123	112	9 %
multitest	633	437	31 %	137	120	13 %
breakst	124	122	2 %	17	16	5 %
assigntest	569	370	35 %	131	117	11 %

실험 결과 분석에서 실험에 이용되었던 최적화된 SGS 파일들은 코드 크기 면에서 약 20~30% 경감의 결과를 보여주었다. 실행 속도 면에서도 각 단위 예제에 대한 100회 반복 실험을 통하여 약 9~10% 정도의 향상을 나타내었다. 특히, 본 최적화기의 패턴 테이블에서 중점을 둔 배열의 배정 관련 최적화를 테스트에서는 두드러지게 향상되었다.

5. 결론 및 향후연구

본 논문에서는 기존의 최적화 기법들의 분석을 기반으로 하여, GVM 을 위한 중간 언어인 SAL 의 특성을 고려한 최적화를 수행하였다. SAL 최적화기는 SAL 코드에 대한 각 명령어의 특성 및 GVM 에서 실행되는 동작을 분석하고, SAL 코드에 대한 최적화 기법들을 이용하여 공통식의 제거, 연산 강도 경감, 루프 불변 코드 이동 등과 같은 최적화를 진행하였다. 특히 SAL 코드에 의존적 최적화 기법인 배정, 스택에 관련된 최적화를 수행하였다. 본 최적화는 패턴 매칭 기법을 이용하였다.

향후 연구 방향은 SAL 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, Compilers Principles, Techniques, and Tools, Addison-Wesley, 1985.

[2] B. J. Mckenzie, “Fast Peephole Optimization Techniques”, Software Practice and Experience, Vol. 19, No. 2, pp.1151-1162, Dec., 1989.

[3] Sinjisoft, GVM & GNEX Developer Guide Homepage, http://www.gnexclub.com

[4] Sinjisoft, GVM SDK version 1.52 Manual, 2002.

[5] Sinjisoft, The Sinji Assembly Language Specification 108, 2001.

[6] W. M. Mckeeman, “Peephole Optimization”, CACM, Vol.8, No.8, pp.443~444, 1970.

[7] 오세만, 컴파일러입문(개정판), 정익사, 2000.