가상기계 코드 최적화를 위한 프로파일링

신양훈[0] *, 이창환*, 오세만*
*동국대학교 컴퓨터공학과
{salz23[0], yich, smoh}@dongguk.edu

Profiling for Optimization of Virtual Machine Codes

Yang-Hoon Shin[0] *, Chang-Hwan Yi*, Se-Man Oh*
*Dept. of Computer Engineering, Dongguk University

요 약

가상기계(Virtual Machine)는 소프트웨어로 제작되어 논리적인 시스템 구성을 갖는 컴퓨터이기 때문에 그 수행 속도와 필요 저장 공간 측면에서 성능이 떨어질 수 밖에 없다. 따라서 가상기계의 성능에 있어서 보다 효율적인 코드로의 최적화가 중요하다.

본 논문에서는 가상기계 코드(Virtual Machine Code) 최적화를 위해 코드를 실행하여 얻을 수 있는 동적 정보인 프로파일링 데이터(Profiling Data)를 정의하고, 프로파일링 시스템을 설계하여 프로파일링 데이터를 가상기계 코드 최적화에 적용 할 수 있는 기반을 마련하였다. 나아가 EVM(Embedded Virtual Machine)에서 실행되는 SIL(Standard Intermediate Language) 코드를 대상으로 프로파일링 시스템을 구현하여 실제 가상기계 코드에 대하여 프로파일링 데이터를 추출하였다.

1. 서론

가상기계란 하드웨어로 이루어진 물리적 시스템과는 달리 소프트웨어로 제작되어 논리적인 시스템 구성을 갖는 개념적인 컴퓨터이다. 가상기계 기술을 이용하면 응용 프로그램 실행 환경인 프로세서나 운영 체제가 바뀌더라도 응용 프로그램을 수정하지 않고 사용할 수 있다. 하지만, 가상기계는 소프트웨어적인 컴퓨터이기 때문에 기존의 물리적인 시스템 위에서 직접 수행되는 것보다 그 수행속도와 필요 저장 공간 측면에서의 성능이 떨어질 수밖에 없다. 이러한 가상기계 환경에서의 최적화는 가상기계의 소프트웨어적인 측면을 보완하여 그 성능을 향상시키는 면에서 중요하다.

최적화 기법은 크게 코드를 정적으로 분석하여 최적화하는 정적 최적화 방법과 코드를 실행하여 얻어진 동적인 정보를 분석하여 최적화하는 동적 최적화 방법으로 구분할 수 있다.

정적 최적화 방법은 실행 시 코드 상태 정보가 고려되지 않는다는 면에서 동적 최적화 방법에 비해 성능 개선에 제한적이다. 하지만 동적 최적화 방법을 적용하기 위해서는 코드를 실행하여 얻을 수 있는 코드에 대한 동적 정보인 프로파일링 데이터가 필요하다.

본 논문에서는 가상기계 코드의 효과적인 최적화를 위한 프로파일링 데이터를 정의하였고, 실행 정보를 기록하는 일련의 코드인 프로브를 이용한 프로파일링 시스템을 설계하여 프로파일링 데이터를 가상기계 코드 최적화에 적용 할 수 있는 기반을 마련하였다. 나아가 EVM에서 실행되는 가상기계 코드인 SIL코드를 대상으로 프로파일링 시스템을 구현하여 가상기계 코드에 대한 프로파일링 데이터를 추출하여 프로파일링 데이터를 확인 하였다.

본 논문의 2장에서는 가상기계 코드에 대한 소개와 특징에 대하여 간략히 언급하고, 프로파일링의 종류와 실행 정보를 기록하는 일련의 코드인 프로브(probe)에 대해 설명한다. 그리고 본 논문에서 사용한 가상기계인 EVM과 EVM의 가상기계 코드인 SIL에 대하여 간략하게 소개한다. 또한 3장에서는 본 논문에서 제시하는 프로파일링 시스템과 가상기계 코드의 특징에 따른 가상기계 코드 최적화를 위한 프로파일링 데이터를 정의하고, 본 시스템에서 정의하고 사용할 데이터 형식을 기술한다. 4장에서는 구현된 시스템을 통하여 생성된 실제 프로파일링 데이터를 확인하고, 마지막으로 5장 결론에서는 본 논문에서 제시한 프로파일링 데이터의 유용성에 대하여 간략하게 요약하고 향후 연구 방향에 대하여 언급한다.

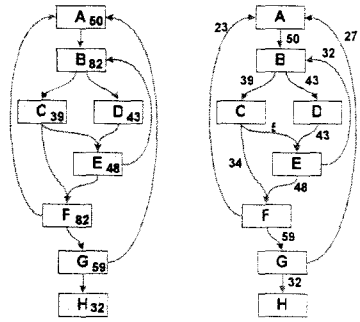
2. 관련연구

2.1. 프로파일링 (Profiling)

프로파일링은 프로파일을 하는 방법에 따라 크게 노드기반(Node-based) 프로파일링과 아크기반(Arc-based) 프로파일링으로 구분할 수 있다.

노드기반 프로파일링은 소스 코드를 기본 블록(Basic block)으로 나누어 각 기본 블록의 정보를 수집하는 프로

파일링으로서 기본블록의 특성을 분석하는데 중점을 둔 기법이다. 또한, 아크기반 프로파일링은 소스 코드를 기본 블록으로 나누고, 기본 블록들 사이의 정보를 수집하는 프로파일링으로서 기본블록 간의 관계를 분석하는데 중점을 둔 것이다. <그림 1>은 노드기반 프로파일링과 아크기반 프로파일링의 예를 보여준다.



<그림 1> 프로파일링의 종류

2.2. 프로브(Probe)

프로브(Probe)란 실행 정보를 기록하는 일련의 코드로서 기본 블록과 함수 블록의 성질을 나타내는 정보를 기록한다. 프로파일링을 시작하기 전에 가상기계 코드에 프로브를 삽입하는 것은 프로파일링 데이터를 추출하는데 중요한 부분이다. 이렇게 프로브가 삽입된 코드를 실행하여 각 기본 블록과 함수 블록의 특성을 동적으로 파악하고 이 정보를 이용하여 프로파일링 데이터를 출력한다. 이렇게 수집된 정보는 각 기본 블록과 함수 블록의 최적화 여부, 최적화 강도 결정, 최적화 기법의 종류를 결정하는 정보로 사용되게 된다.

2.3. 가상기계 코드 (Virtual Machine Code)

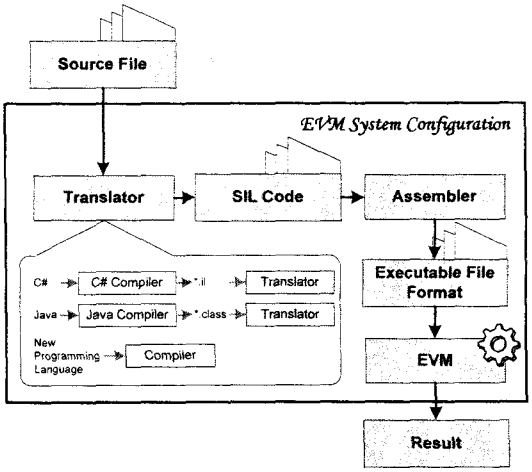
가상기계 코드란 가상기계의 어셈블리 코드로서, 가상기계에서 즉시 실행될 수 있는 이진 코드와 1:1로 대응되는 명령어의 집합을 말하며, 가상기계의 어셈블리 코드라 할 수 있다.

가상기계 코드는 연산 방식에 따라서 스택 기반 코드와 레지스터 기반 코드로 구분할 수 있으며, 일반적으로 복잡한 레지스터 할당 구현의 불필요, 코드 생성용이, 인터프리터 구현용이 등의 이유로 스택 기반 코드로 설계된다. 가상기계 코드의 구조는 크게 데이터부(data section)와 코드부(code section)로 구분할 수 있다. 데이터부에는 연산에 이용될 데이터 정보가 지시어로서 표현되어 있으며, 코드부에는 가상기계 코드가 실행되는 연산 정보가 명령어로서 표현되어 있다.

2.4. EVM (Embedded Virtual Machine)

EVM은 모바일 디바이스(Mobile device), 셋톱 박스(Set-top box), 디지털 TV(Digital TV)등에 탑재되어 동적 응용 프로그램을 다운로드하여 실행할 수 있는 가상기계 솔루션이다.

EVM은 크게 C#이나 자바 등의 고급 프로그래밍 언어로 작성된 프로그램을 형태로 번역하는 부분, SIL 코드를 가상기계에서 실행 가능한 형태인 *.evm 포맷으로 변환하는 어셈블러 부분, 실제 하드웨어에 탑재되어 *.evm 파일을 실행하는 가상기계 부분으로 구성된다. EVM 시스템 구성도는 <그림 2>와 같다.



<그림 2> EVM 시스템 구성도

SIL(Standard Intermediate Language)은 EVM의 어셈블리 언어로서 EVM에서 실행될 수 있는 이진 코드와 대응되는 명령어 집합이다. 이는 어셈블러에 의해 *.evm 형태로 변환되고 시스템의 운영 체제나 구조에 상관없이 EVM에 의해 실행된다.

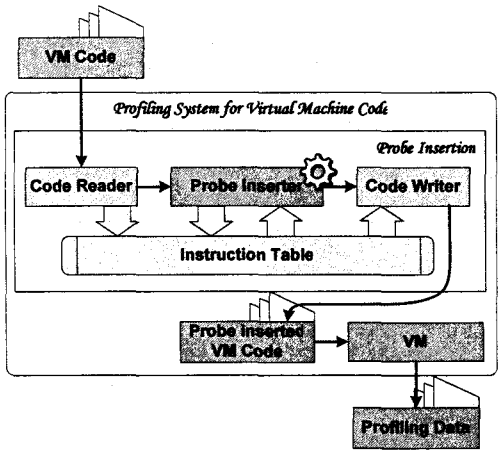
SIL은 임베디드 시스템을 위한 가상기계의 표준 중간 언어로 설계되었다. 따라서, 다양한 프로그래밍 언어를 수용하기 위해서 Oolong, .NET IL등 기존에 널리 사용되고 있는 가상기계 어셈블리 언어들의 분석을 토대로 정의하였다. SIL은 클래스 선언 등 특정 작업의 수행을 의미하는 의사 코드와 가상기계에서 실행되는 실제 명령어에 대응되는 연산 코드로 이루어져있다. 연산 코드는 스택 기반의 명령어 집합이며 특정 프로그래밍 언어에 종속되지 않는 언어 독립성과 하드웨어 및 플랫폼 독립성을 갖고 있다. 따라서 연산 코드의 니모닉은 특정 하드웨어나 소스 언어에 종속되지 않는 추상적인 형태를 지닌다.

3. 프로파일링 시스템

본 장에서는 논문에서 설계하고 구현한 프로파일링 시스템에 대하여 설명하고 가상기계 코드의 특징에 따른 프로파일링 데이터를 살펴보고, 본 시스템에서 정의하고 사용할 데이터 형식에 대해 언급한다.

3.1. 시스템 구성

프로파일링 시스템은 <그림 3>와 같이 구성된다. 또한 시스템은 크게 프로브삽입 (Probe Insertion) 부분과 가상기계 부분으로 세분화 되며, 프로브 삽입기, 가상기계가 핵심적인 역할을 한다.



<그림 3> 프로파일링 시스템 구성도

3.1.1. 프로브 삽입 (Probe Insertion)

프로브 삽입 부분은 코드 입력기와 출력기(Code reader / writer), 프로브 삽입기(Probe inserter)로 구성되고, 가상기계 코드를 입력으로 받아 그 가상기계 코드에 대하여 프로파일 데이터를 산출하기 위한 일련의 코드인 프로브를 삽입하는 역할을 한다. 프로브는 프로그램 단위영역의 특징을 추출하기 위하여 특정 명령어 다음이나, 기본블록의 시작점 또한 함수의 호출시점이나 함수의 시작점 등에 삽입된다.

3.1.2. 가상기계 (EVM)

가상기계 부분은 내부적으로 실행엔진(Execution Engine)과 프로파일링 정보 분석기 구성된다. 프로브 삽입기에 의해 프로브가 삽입된 가상기계 코드는 가상기계를 통하여 실행되어, 가상기계 코드에 대한 실행 정보를 산출한다. 산출된 실행정보는 프로파일링 정보 분석기를 통해 분석되어 최종적으로 가상기계 코드에 대한 프로파일링 데이터를 생성한다.

3.2. 가상기계 코드 최적화를 위한 프로파일링 데이터

가상기계 코드의 최적화를 위한 프로파일링 데이터가 최적화에서 효과적으로 사용되기 위해서는 가상기계 코드의 특성을 연구하여 그에 알맞은 프로파일링 데이터를 정의하는 것이 중요하다.

가상기계 코드는 프로그램 작성의 편의보다는 프로그램의 하드웨어의 독립성을 확보 할 수 있도록 설계된 코드로서 단계적 컴파일을 통한 컴파일러 제작의 용이성을 확보 할 수 있다. 이로 인하여 가상기계 코드는 직접 하드웨어를 접근하여 수행되기 보다 가상기계 내의 논리적인 메모리영역인 레지스터(Register), 스택(Stack) 과 힙(Heap)영역에서 간접적으로 처리하게 된다. 따라서 프로파일링 데이터는 기본적인 기본블록 실행 수 (basic block count), 기본블록 크기 (size of basic block), 함수 호출 수 (function call count)와 더불어 논리적인 메모리 영역을 접근하는 스택 읽기 수 (stack read count), 스택 쓰기 수 (stack write count), Memory 읽기 수 (memory [heap] read count), Memory 쓰기 수 (memory[heap] write count), 레지스터 읽기 수 (register read count), 레지스터 쓰기 수 (register write count), 등으로 표현할 수 있다.

<표 1> 프로파일링 데이터

이름	Bytes	설명
Function Name	128	Function name
Block ID	8	Basic block ID
Start Line	8	Starting point of function or block
Size	8	Size of function or block
Count	8	Execution count of function or block
Stack Read	8	Stack read count
Stack Write	8	Stack write count
Heap Read	8	Memory read count
Heap Write	8	Memory write count

Profiling Data File Format

```
<file> ::= '%File ' <file_name> { <function_def> }
        '%Enc-File '<file_name>

<function_def> ::= <function_info> { <block_info> }

<function_infc> ::= '%Function '<function_name>' '
<start_line>' '<size>' '<count>' '<stack_read>' '
<stack_write>' ' <heap_read>' ' <heap_write>

<block_infc> ::= <block_ID>' '<start_line>' '
<size>' '<count>' '<stack_read>' '<stack_write>' '
<heap_read>' '<heap_write>

<file_name> ::= '$identifier'
<function_name> ::= '$identifier'
<block_ID> ::= '$identifier'
<start_line> ::= $integer
<size> ::= $integer
<count> ::= $integer
<stack_read> ::= $integer
<stack_write> ::= $integer
<heap_read> ::= $integer
<heap_write> ::= $integer
```

<그림 4> 프로파일링 데이터 파일형식 (EBNF)

3.3. 프로파일링 데이터 형식

본 시스템에서 정의하고 사용할 가상기계 코드 최적화를 위한 프로파일링 데이터는 <표 1>과 같고, 프로파일링 데이터 파일형식을 EBNF로 표현한 것은 <그림 4>과 같다.

4. 실험결과

시스템은 ANSI-C 호환 컴파일러인 Visual C++ 6.0 컴파일러를 사용하여 구현하였고, 실행환경으로 EVM을 사용하여 프로파일링 데이터를 산출하였다.

실험과정은 먼저 예제소스를 컴파일러를 사용하여 컴파일하여 SIL코드를 출력하고, 출력된 SIL코드를 probe 삽입기를 통하여 SIL코드에 probe를 삽입한다. 마지막으로 probe가 삽입된 SIL 코드를 EVM을 통해 실행하여 프로파일링 데이터를 산출한다. 테스트를 위한 예제는 " The N-queen problem" 소스를 사용하였고 이에 대한 프로파일링 데이터를 산출한 결과는 <그림 5>와 같다.

Profiling Data - N queen problem

```
%File N_queens c sil
%     Functior  mair
%               $$C 2C 4 1 C E C 1
%               $$1 24 4 9 18 18 E C
%               $$2 28 4 E 16 16 E E
%               $$3 32 3 E E C E E
%               $$4 35 3 1 C C C 1
%               $$5 38 4 16 32 32 16 C
%               $$6 42 E 15 75 75 6C 30
%               $$7 50 3 15 15 C 15 15
%               $$E 53 E 1 C 2 2 2
%     End-Functior  mair 2C 41 1 13 18 1' 9
%     Functior  rn_queens
%               $$9 61 10 1965 786C 9825 1965 589E
%               $$10 71 E 1768E 53055 53055 53055 C
%               $$1' 76 6 1572C 1572C 31440 31440 31440
%               $$12 82 44 205E 74016 74016 22616 616E
%               $$13 126 2 92 C C C 92
%               $$14 128 4 828 1656 1656 1656 C
%               $$15 132 21 736 736 4416 220E 736C
%               $$16 153 3 736 736 C 736 736
%               $$17 156 6 92 92 92 C 184
%               $$18 162 E 1964 3928 785E 3928 3928
%               $$19 17C 4C 205E 67848 67848 18504 616E
%               $$2C 210 1 1572C C C C C
%               $$21 211 E 1572C 94320 78600 4716C 1572C
%               $$22 219 3 1965 C C 393C C
%     End-Functior
            rn_queens 61 161 1965 17685C 190605 76635 55020
%     Functior  positior_ok
%               $$23 222 16 1572C 172920 18864C 62880 31440
%               $$24 238 12 550E 60588 60588 27540 C
%               $$25 250 3 1572C 1572C 1572C 1572C C
%               $$26 253 14 342C 4446C 4446C 1368C C
%               $$27 267 4 1572C 31440 31440 31440 C
%     End-Functior
          positior_ok 222 49 1572C 597360 613080 251520 31440
%End-File N_queens c sil
```

<그림 5> 프로파일링 데이터 예

5. 결론 및 향후 연구

가상기계가 소프트웨어적인 컴퓨터이기 때문에 가상기계 코드의 실행 시 상태 정보를 가상기계 코드 최적화에 반영하는 것은 중요하다. 이에 본 논문에서는 가상기계 코드의 특징을 분석하였고, 그 특징에 맞는 프로파일링 데이터(Profiling Data)를 정의하였다. 또한 프로파일링 데이터를 추출하는 프로파일링 시스템을 설계하여 최적화에 적용 할 수 있는 기반을 마련하였고, EVM의 가상기계 코드인 SIL을 통해 실험하여 결과인 프로파일링 데이터를 확인하였다. 이 프로파일링 데이터의 유용성은 가상기계 코드의 특징을 나타낼 수 있고, 그 특징을 가상기계 코드 최적화기에 적용할 수 있는데 있다.

향후에는 본 논문에서 제안한 프로파일링 데이터를 가상기계 코드 최적화기에 적용할 것이고, 가상기계 코드를 위한 더욱 효과적인 프로파일링 기법을 연구할 것이다. 또한, 프로파일링 알고리즘을 더욱 개선하여 프로파일링 오버헤드를 경감 시킬 예정이다.

참고문헌

[1] James E. Smith, Ravi Nair, Virtual Machines - Versatile Platforms for Systems and Processes, Morgan Kaufmann, 2005.

[2] Michael D. Bond, Kathryn S. McKinley, "Continuous Path and Edge Profiling," IEEE/ACM International Symposium on Micro-architecture table of contents, pp.130-140, 2005.

[3] Microsoft Corporation, MVM Instruction Set Specification, 2000.

[4] Pohua P. Chang, Scott A. Mahlke, Wen-mei W. Hwu, "Using Profile Information to Assist Classic Code Optimizations," IEEE Software Practice and Experience, Vol.21, No.12, pp.1301-1321, 1991.

[5] Thomas Bell, "The Concept of Dynamic Analysis," ACM SIGSOFT international symposium on Foundations of software engineering table of contents, pp.216-234, 1999.

[6] Thomas Ball, Peter Mataga, Mooly Sagiv, "Edge Profiling versus Path Profiling," ACM Symposium on Principles of Programming Languages, pp.134-148, 1998.

[7] Tim Lindholm, Frank Yellin, The Java Virtual Machine Specification, Second Edition, Addison Wesley, 1999.

[8] Vasanth Bala, Evelyn Duesterwald, Sanjeev Banerjia, "Dynamo: a transparent dynamic optimization system," ACM SIGPLAN conference on Programming language design and implementation, pp.1-12, 2000.

[9] 오세만, 컴파일러 입문 개정판, 정익사, 2005.