내장형 자바를 위한 클래스 파일의 바이트 코드 압축

이영민, 맹혜선, 강두진, 김신덕, 한탁돈
연세대학교 병렬처리 연구실

Bytecode Compression Method for Embedded Java System

Young-Min Lee, Hye-Seon Maeng, Doo-Jin Kang, Shin-Dug Kim, Tack-Don Han
Parallel Processing Lab. , Yonsei Univ.

요 약

본 논문에서는 추후 여러 내장형 기기들을 대체하게될 내장형 자바가상머신에서 효율적으로 메모리를 사용하기 위해 바이트코드 압축 방법을 제시하고 있다. 이 압축 방법은 기본 코드 블럭을 내장형 자바가상머신에서 사용하지 않는 명령어군과 한 바이트의 인덱스를 이용하여 사전을 구축하고, 사전에 등록되어 있는 반복되는 기본 코드 블럭들을 이 두 바이트로 대체함으로써 압축하는 것이다. 그러나, 압축하는데 있어서 압축효율 뿐만 아니라 바이트 코드의 수행 속도도 고려하여야 하므로 압축으로 인한 수행 오버헤드를 최소화하여 수행시간에 영향을 적게 주도록 압축 방법을 단순화하여 설계하였다. 본 논문에서 제시하고 있는 압축 방법을 사용하여 실제 사용되는 자바 API(Application Programming Interface)등을 압축함으로써 메모리에 적재되는 바이트 코드를 최대 36%까지 줄이는 결과를 얻어낼 수 있다

1. 서 론

내장형 자바가상머신[1]은 기본적으로 자바가상머신 명세[2]를 따른다. 그러나 실제적으로 구현될 때에는 내장형 기기들이 가지는 제약점이 고려되어야 한다. 내장형 자바가상머신은 내장형 시스템의 제약 위에서 수행되어야 하기 때문이다. 따라서 많은 메모리를 이용하여 빠른 수행 시간을 얻어낼 수 있는 구조보다는 적은 메모리 상에서 효율적으로 수행할 수 있는 구조가 내장형 자바가상머신에 적합하다.

내장형 자바가상머신을 제한된 크기의 메모리 상에서 수행하기 위하여 클래스 파일의 바이트 코드를 압축하여 저장하고 이를 이용할 수 있다. 클래스 파일의 바이트 코드는 스택 기반 머신 코드이기 때문에 특정한 블록이 자주 반복되는 특성을 가진다. 이 반복되는 코드들을 압축 코드로 표현하므로써 메모리에 적재되는 바이트 코드의 크기를 감소시킬 수 있다

본 논문에서는 바이트 코드 명령어 집합 중에서 내장형 자바가상머신이 사용하지 않는 quick 명령어군을 이용하여 압축 코드를 표현하는 방법을 제안하였다. Quick 명령어에 한 바이트 크기의 연산자를 추가하므로써 충분한 개수의 압축된 블록을 표현할 수 있을 뿐만 아니라 향후 자바가상머신의 바이트 코드의 명령어 집합이 확장되더라도 충돌을 일으키지 않을 수 있다. 압축될 코드 블록을 저장하는 방식으로는 간접 참조가 필요 없이 직접 접근이 가능한 배열 구조로 사전을 구성하도록 하였다. 이 방법은 압축된 코드를 접근할 때 발생하는 메모리 접근 회수를 줄이므로써 런타임시의 수행 오버헤드를 최소화한다

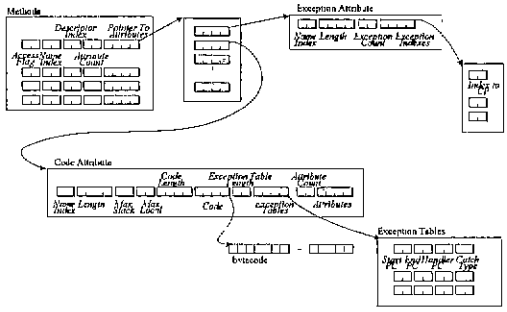
본 논문에서 제안한 방법을 썬마이크로시스템즈(Sun Microsystems)의 퍼스널 자바(Personal Java) API와 트랜스버추얼(Transvirtual)의 까페(Kaffe) API에 적용한 결과 클래스들이 메모리에 적재되는 바이트 코드 크기를 최대 30%이상 줄이는 결과를 얻을 수 있었다.

본 논문의 구성은 다음과 같다. 2 장에서는 클래스 파일 내에서 바이트 코드가 차지하는 영역에 대해 살펴보도록 한다. 3 장에서는 바이트 코드를 압축하기 위한 압축 코드의 구성 및 사전 구성 방법을 설명한다 4 장에서는 실험을 통하여 자바 API에 대해서 코드 압축 방법을 적용하였을 때 얻을 수 있는 실제적인 압축효과를 살펴 보도록 한다. 마지막으로 5장에서는 연구에 대한 결론과 함께 향후 계획을 제시하도록 한다.

2. 자바 클래스 파일

자바 언어로 작성된 프로그램을 자바 컴파일러를 이용하여 컴파일을 수행하면 클래스 파일을 얻을 수 있다. 클래스 파일은 자바 소스 프로그램에 대한 정보와 프로그램 수행에 필요한 컨스턴트 정보들 그리고 자바 프로그램에 정의된 필드와 메소드 등에 관한 정보를 가진 이진 파일이다.

클래스 파일 내에서 컨스턴트풀 정보는 평균적으로 클래스 파일의 약 50%를 차지한다. 다음으로 클래스 파일 내에서 많은 영역을 차지하는 것은 메소드에 관한 정보이다. 메소드 정보는 [그림 1]과 같은 구조로 구성된다 메소드 기술 정보, 코드 속성 정보, 바이트 코드, 메소드 디버깅 정보가 모두 메소드에 관한 정보이다 이에 더하여 예외 처리에 관한 정보가 더 포함된다. 메소드 정보에 속하는 디버깅 정보 또한 클래스 파일 크기의 감소를 위해서 삭제하는 것이 가능하다. 내장형 시스템의 경우에는 런타임시에 디버깅 정보를 필요로 하지 않기 때문에 내장형 자바 가상 머신에서는 디버깅 정보를 제거하고 클래스 파일 로딩을 수행할 수 있다



[그림 1] 메소드 정보의 구성

모든 메소드는 고정된 길이의 메소드 기술 정보와 코드 속성 정보를 가진다 가변적인 크기를 가지는 정보로는 디버깅 정보와 예외 처리에 관한 정보 그리고 바이트 코드에 대한 정보가 있다 본 논문에서는 메소드 정보 중에서 가장 많은 영역을 차지하는 바이트 코드 부분을 압축하는 것을 목표로 한다. 클래스 파일의 바이트 코드에는 여러 종류의 코드 블록들이 자주 반복된다 반복되는 코드 블록들을 테이블로 구성하고 해당 코드 블록을 압축된 코드임을 의미하는 코드들로 대치하므로써 바이트 코드의 크기를 감소시킬 수 있다. 다음 절에서는 코드 영역에서 반복되는 코드를 압축하는 방법을 자세하게 설명하도록 한다.

3. 바이트 코드의 압축

자바 가상 머신은 스택 머신 구조를 가지기 때문에 바이트 코드들이 수행하는 모든 연산은 스택을 통하여 이루어진다. 자바 가상 머신의 또다른 특징은 수행 중간 값의 저장을 위한 레지스터가 존재하지 않는다는 것이다. 대신 지역 변수 영역을 두어 이를 레지스터 대용으로 이용하도록 한다.

바이트 코드 내에서 특정 연산을 수행하는 경우, 연산자들은 일반적으로 지역 변수에 위치한다. 따라서 지역 변수에 저장된 값을 스택으로 푸쉬(push)한 후 연산을 수행하고 그 결과값을 다시 지역 변수에 팝(pop)하는 형태로 연산이 수행된다. 따라서 유사한 연산을 수행하는 경우에는 연산 수행에 대한 명령어 전후의 명령어 집합이 같은 명령어들로 구성되는 것이 일반적이다

바이트 코드 수행 중에 메소드를 호출하는 경우에도 유사한 상황이 발생한다 메소드 호출 시에 필요한 패러미터들은 일반적으로 지역 변수에 저장되어 있다. 메소드 호출을 위해서는 지역 변수에 저장된 값들을 스택으로 푸쉬하고 메소드를 호출하는 명령어를 수행시킨다. 또한 메소드 호출 후에 반환되는 결과값은 스택에서 다시 지역 변수로 팝하여 사용한다 따라서 메소드 호출시마다 발생하는 명령어 집합들은 빈번하게 유사한 형태를 보이게 된다.

본 논문에서는 위와 같이 바이트 코드에서 보여지는 명령이 나열 중 반복되는 코드 블록들을 사전의 엔트리로 구성한 후, 실제 바이트 코드 내에서는 코드 블록을 삭제하고 압축된 코드를 의미하는 명령이와 사전 엔트리에 대한 인덱스 정보로 구성하므로써 바이트 코드의 크기를 줄일 수 있는 방법을 제시하도록 한다

3.1 압축 코드의 표현

자주 반복되는 코드 블록을 제거하고 압축된 코드로 표현하기 위해서는 압축 코드를 표현하기 위한 방법이 제시되어야 한다. 압축 코드를 표현하기 위한 방법은 텍스트 압축 방법에서부터 다양하게 제시되어왔다.

압축 코드를 표현하는 방법을 크게 두 가지로 구분하면 압축 코드의 크기가 고정된 크기로 표현되는 방법과 변화 가능한 크기로 표현되는 방법이 있다. 텍스트 압축 방법의 경우에는 변화 가능한 크기로 표현하므로써 압축의 효과를 더욱 증대시키기도 한다 그러나 이 방법의 경우 복원시의 오버헤드가 증가하는 단점을 가진다. 이진 코드를 압축하는 경우에는 이진 코드의 기본 원형을 유지하는 범위에서 압축 코드를 구성하는 방법이 보다 합리적이다. 본 논문에서는 복원시의 오버헤드를 최소화시키기 위하여 자바 바이트 코드 중에서 사용되지 않는 영역을 압축된 코드를 표현하는 방법을 사용하도록 한다

자바 바이트 코드는 한 바이트씩 구분 되어서 의미를 가진다. 자바 바이트 코드의 명령어는 한 바이트, 8 비트에 비트 정보를 변화시켜서 서로 다른 명령어를 의미하도록 한다 모두 256개의 opcode가 존재할 수 있으나 실제로 의미를 가지는 코드는 0번부터 201번까지이다. 자바 가상 머신 명세에 따르면 203번부터 228번까지의 코드는 quick 명령어로 정의하였다. Quick 명령어는 동적 로딩을 수행하였을 때 간접 참조를 직접 참조로 변화한 정보를 가지고 명령어를 수행할 수 있도록 재정의 하는데 이용된다. 그러나 내장형 자바가상머신에서는 동적 로딩을 수행하지 않으므로 인해서 간접 참조를 직접 참조를 바꾸는 작업이 런타임 전에 이루어지며 따라서 quick 명령어를 사용하지 않는다.

본 논문에서는 내장형 자바가상머신에서 사용되지 않는 quick 명령어를 압축된 코드로 이용하는 방법을 사용한다. 바이트 코드 중 quick 명령어 영역 뿐만 아니라 미사용으로 정의된 코드 영역을 모두 압축 코드를 표현하는 방법도 가능하지만 이 방법의 경우에는 향후 자바 바이트 코드의 명령어 집합이 확장되는 경우에는 명령어 충돌이 발생할 수 있는 문제점이 존재한다

제안하는 압축 코드 표현 방법은 quick 명령어에 피연산자를 1 바이트 추가하므로써 보다 많은 사전 엔트리를 접근할 수 있도록 한다. 압축 코드에는 총 26개의 quick 명령어 중에서 25개가 이용되는데 마지막 quick 명령어는 압축된 코드 블록이 끝남을 표시하는 정보로 이용한다. quick 명령어에 한 바이트의 피연산자 정보를 부가 시키서 얻을 수 있는 서로 다른 코드 정보의 개수는 다음과 같다.

$$(26(\text{quick instruction}) - 1) \times 256 = 6400$$

자바 가상 머신에서 명령어를 읽어서 해독하는 시점에서 quick 명령어 계열의 명령어가 발견되면 다음 바이트에 존재하는 피연산자를 읽어서 실제 코드가 저장된 영역을 계산한다 본 논문에서 제안하는 방법에서는 코드 블록을 고정된 크기의 엔트리에 저장하도록 구성하므로써 코드 블록들이 배열의 형태로 저장되도록 구성한다. 따라서 실제 코드 블록을 접근하기 위해서 자바 가상 머신의 인터프리터는 엔트리의 위치 정보 계산을 위한 1회의 연산만을 추가적으로 수행한다

3.2 코드 블록의 선택

반복되는 부분을 삭제하고 압축을 의미하는 코드로 대치하므로써 압축의 효과를 얻어내는 방법은 일찍이 텍스트 압축에서부터 사용되어 왔다. 또한 내장형 RISC 프로세서에서 제어 프로그램을 압축하는 방법도 널리 연구되어 왔다. 모든 방법들은 공통적으로 반복되는 부분을 효과적으로 선택할 수 있어야 한다는 조건을 가진다. 실제로 압축의 최대 효과를 얻

도록 반복 구문을 선택하는 것은 NP 복잡도를 가진다.

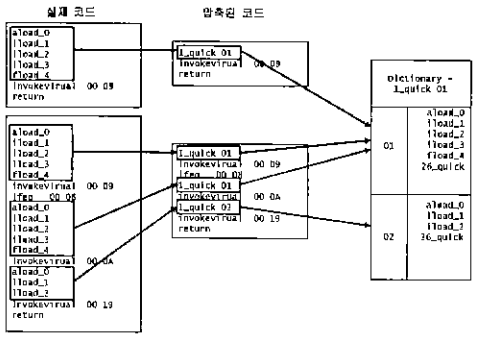
본 논문에서는 반복되는 코드 블록의 범위를 기본 코드 블록(Basic Code Block)으로 한정한다. 기본 코드 블록은 분기 명령을 포함하지 않으며 분기 명령의 대상이 되는 명령을 포함하지 않는 블록을 의미한다. 분기 명령이 압축될 코드 블록에 포함되면 압축을 위해서 고려해 주어야 하는 사항에 대한 복잡도가 크게 증가하고 이를 위한 처리에 런타임 비용이 소요되기 때문에 본 연구에서는 기본 코드 블록만을 이용하도록 한다.

기본 코드 블록들에 대한 리스트를 생성하고 각각의 코드 블록들이 반복되는 회수를 저장하도록 한다 코드 블록을 선택하는 방식으로는 코드 블록이 반복되는 회수와 코드 블록을 압축 코드로 대치하므로써 얻을 수 있는 이득을 계산하여 가장 높은 대치 효과를 얻을 수 있는 블록으로부터 선별하는 방법을 사용하도록 한다.

3.3 사전의 구성 및 이용

선택된 코드 블록들을 사전으로 구성하는 방법은 다음과 같다 각각의 코드 블록들은 사전의 엔트리로 등록되어 순차적으로 저장된다 [그림 2]는 반복되는 기본 코드 블록을 사전으로 구성하고 이들에 대한 인덱스 정보를 저장하는 방법을 종합적으로 보여준다.

특징적인 것은 사전의 엔트리들이 공통적으로 26_quick 명령어를 코드 블록의 맨 마지막에 가진다는 것이다. 이것은 압축 코드 정보를 해석하고 사전으로 분기하여 사전 내의 코드를 수행하고 있는 도중 다시 본래의 코드로 돌아갈 수 있는 정보를 주기 위한 것이다. 만약 이러한 코드를 사용하지 않는다면 사전의 엔트리에 존재하는 명령어를 수행할 때마다 비교 명령을 통해서 다음 명령을 사전 내에서 찾아야할 지 아니면 본래의 코드에서 찾아야할 지를 결정해야할 것이다



[그림 2] 사전을 이용한 바이트코드 압축

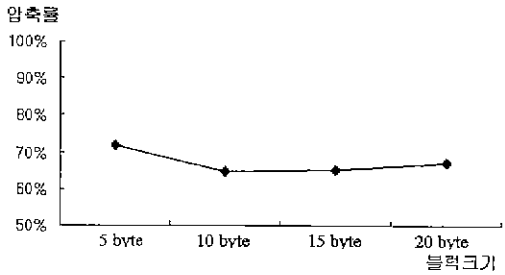
4. 코드 압축 실험 및 결과

앞서 제시한 코드 압축 방법을 적용하므로써 얻을 수 있는 압축 효과를 알아보기 위해서 썬마이크로시스템즈사의 퍼스널 자바 API와 트랜스비추얼사의 까페 API에 적용한 후 압축률을 계산하였다 압축 비율은 다음 식에 의해서 구하여 진다

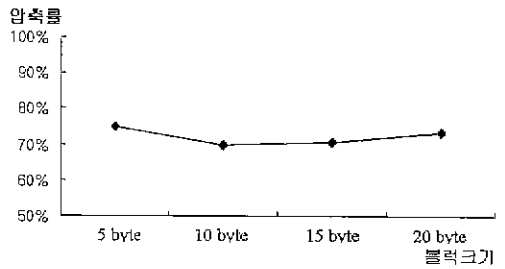
$$\text{압축비율} = \frac{\text{압축전 크기} - \text{압축된 크기}}{\text{압축전 크기}}$$

압축된 크기는 압축된 코드와 사전의 크기를 포함한다 바이트 코드를 압축하기 이전의 퍼스널 자바 API에 대한 바이트 코드가 383,494 바이트이며 까페 API에 대한 바이트 코드가 183,289 바이트이다.

[그림 3]과 [그림 4]는 각각 퍼스널 자바 API에 대한 압축 비율과 까페 API에 대한 압축 비율을 보여준다. 각각의 실험은 사전 엔트리의 크기를 5, 10, 15, 20 바이트로 변화시키면서 수행하였다. 압축률은 사전 엔트리의 크기가 10 바이트인 경우에 가장 높은 값을 나타내는데 기본 코드 블록의 크기가 10 바이트 주변에 몰려 있기 때문이다. 퍼스널 자바 API의 경우 사전 엔트리의 크기가 10 바이트인 경우 압축률이 64%에 이른 결과를 보여주었다.



[그림 3] 퍼스널 자바 API 압축 비율



[그림 4] 까페 API 압축 비율

5. 결 론

본 논문에서는 내장형 자바가상머신에서 효율적으로 메모리를 사용하기 위한 방법으로서 바이트 코드 압축 방법을 제안 하였다 바이트 코드 명령어 집합 중에서 내장형 자바가상머신에서 사용하지 않는 quick 명령어 군을 압축된 코드를 표현하는데 이용하므로써 압축률이 높고 호환성에 문제가 없이 바이트 코드를 압축하는 것이 가능하였다 실험을 통하여 퍼스널 자바와 까페 API에 압축 방법을 적용한 결과 최대 34%의 코드 압축률을 얻을 수 있었다 향후 연구 계획으로는 런타임 수행을 통하여 압축 코드의 해석으로 인한 오버헤드가 전체 수행 시간에 미치는 영향을 분석하고 이를 최소화할 수 있는 방법을 제시하므로써 높은 압축률을 가지고 효율적으로 수행될 수 있는 방법으로 발전시킬 계획이다

참고문헌

[1] Tim Lindholm, Frank Yellin, *The Java Virtual Machine Specification Second Edition*, Addison Wesley, 1999
[2] "EmbeddedJava Application Environment Specification," http://java.sun.com/80/products/embeddedjava/spec/, 1999
[3] T. C. Bell, J G. Cleary, and I. H. Witten, *Text Compression*, Prentice Hall, 1990
[4] L. R. Clausen, U. P. Schultz C. Consel, and G. Muller, "Java Bytecode Compression for Embedded Systems," IRISA Publication Interne N1213