자바가상기계에서 데이터 이동 분석

양희재

경성대학교 컴퓨터공학과

Analysis of Data Transfers in Java Virtual Machine

Heejae Yang

Dept of Computer Engineering, Kyungsung University

E-mail: hjyang@star.ks.ac.kr

요 약

스택 기반 구조를 갖는 자바가상기계(JVM)에서는 전체 동작의 상당 부분이 데이터 이동에 소요되는 것으로 알려지고 있다. 따라서 효율적인 JVM의 개발을 위해서는 JVM 내부에서 데이터가 어떻게 이동되는지를 분석할 필요가 있다. 본 논문에서는 오퍼랜드 스택, 지역변수배열, 힙, 그리고 상수 풀 사이에서 데이터의 이동에 대해 바이트코드 수준에서 분석 조사하였다.

ABSTRACT

It is widely known that most operations performed in JVM belongs to data transfers at all times as JVM is based on abstract stack machine. Hence it is necessary to analyze the fashion of internal data transfers in JVM to develop a more efficient machine. We have analyzed in this paper the data transfer operations between operand stack, local variable array, heap, and constant pool in bytecode level.

키워드

자바, 자바가상기계, 바이트코드, 데이터 처리

I 서 론

자바가상기계(JVM)의 메모리 영역은 오퍼랜드 스택, 지역변수배열, 그리고 힙 등 세 가지로 나눌 수 있다 [1][2]. 자바 프로그램의 실행에 따라 수많은 데이터가 생성되며, 이들 데이터는 오퍼랜드 스택, 지역변수배열, 힙 등 세 가지 영역 사이를 빈번히 이동한다. 데이터의 이동은 하드웨어적으로 메모리에 대한 접근을 필요로 하며, 메모리에 대한 접근은 시간적 지체와 전력 소비를 야기한다 [3]. 따라서 자바가상기계의 성능을 향상시키고 전력 소비를 최소화하기 위해서는 무엇보다도 자바가상기계 내에서 데이터가 이동하는 형태에 대한 면밀한 분석이 필요하며, 이 분석에 따라 최적의 데이터 이동 경로를 설계하는 것이 필요하다.

본 논문에서는 자바가상기계의 데이터 이동 형태, 즉 어느 영역에서 어느 영역으로의 이동이 빈번한지, 데이터 이동의 방향은 어떠한지 등에 대해 바이트코드 수준에서 분석한다. JVM 규격에서 개별 바이트코드의 기능에 대해 설명하고 있지만, 아직까지 데이터 이동과 관련하여 바이트코드를 분류한 자료나 논문은 발견되지 않았다. 본 연구의 결과는 향후 효율적인 자바가상기계의 설계에 적용될 수 있을 것으로 기대된다.

II 자바 메모리

그림 1은 JVM의 사용 메모리를 개념적으로 나타낸 것이다. 클래스 영역은 클래스 정보, 즉 메소드를 이루는 바이트코드 등이 들어가 있는 읽기 전용 영역이며, 오퍼랜드로 사용될 상수 값도 이곳에 위치한 상수 풀(constant pool)에 저장된다. 네이티브 메소드 스택은 C 등 네이티브 언어로 작성된 프로그램이 사용하는 스택 부분을 의미하며, 바이트코드가 실행되는 영역이 아니므로 본 연구의 범위와는 무관하다.

본 연구의 관심은 자바 스택 메모리와 힙 메모

† 본 연구는 한국학술진흥재단 지역대학우수과학자 사업지원으로 수행되었음.

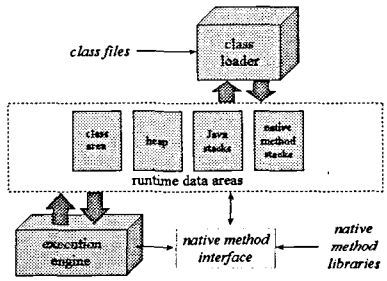

그림 1. 자바가상기계의 메모리 사용

리다. 자바 스택 메모리는 수많은 자바 스택 프레임으로 구성되며, 각 스택 프레임은 다시 오퍼랜드 스택과 지역변수배열로 나뉘어진다. 본 연구에서는 상수 풀과 오퍼랜드 스택, 지역변수배열, 그리고 힙 사이의 데이터 이동에 대해 조사해본다.

III 데이터 이동 분석

JVM 규격에 따르면 현재 201개의 바이트코드들이 제공되고 있다. 이들 바이트코드들을 분석한 결과 데이터의 이동 경로는 그림 2와 같이 주어짐을 알 수 있었다.

그림 2에서 볼 수 있듯이 데이터 이동 경로의 중심은 오퍼랜드 스택이다. 한 메모리 영역에서 다른 메모리 영역으로 옮겨갈 때 항상 오퍼랜드 스택을 경유하게 됨을 발견할 수 있다. 이 구성은 오퍼랜드 스택에서 데이터 이동에 따른 병목 현상이 발생할 수 있음을 암시해주는 것이다.

다음 분석은 자바 언어로 작성된 프로그램을 디스어셈블하여 어떤 경우에 각 영역에서 다른 영역으로의 데이터 이동이 발생하는지를 알아본 것이다.

3.1 상수 풀에서 오퍼랜드 스택으로

상수 풀(constant pool)에는 자바 프로그램 내에서 사용되는 각종 상수들이 들어있으며, 상수에 대한 접근은 상수 풀의 인덱스 값으로 이루어진다. 상수는 읽기 동작만 이루어지므로 그림 2의 1번 화살표와 같이 상수 풀에서 오퍼랜드 스택으로의 데이터 이동만 일어날 뿐 그 반대 방향은 발생하지 않는다.

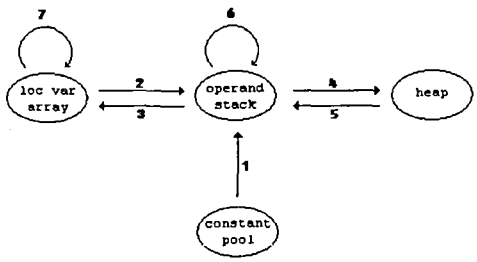

그림 2 자바가상기계의 데이터 이동 경로

자바 프로그램에서 상수 풀에서 오퍼랜드 스택으로의 데이터 이동은 다음과 같은 경우에 일어난다.

```
int i, j, k, l;
i = 50000; j = 0; k = 100; l = 500;
```

여기서 변수 i, j, k, l 의 위치가 지역변수배열이든 힙 영역의 필드든 관계없이 그림 2에 보인 것과 같이 항상 오퍼랜드 스택에 해당 상수 값이 놓이는 것을 알 수 있다. 바이트코드로는 다음과 같이 주어진다.

```
ldc #n
```

ldc 뒤의 인수는 해당 상수가 들어있는 상수 풀의 인덱스 값이다. ldc 외에도 ldc_w, ldc2_w 등의 명령도 이같은 데이터 이동에 사용된다.

0에서 5사이의 정수, 또는 -1 등 빈번히 사용되는 작은 크기의 정수형 상수는 상수 풀에 두지 않고 명령어의 의미 내에 포함한다 (즉 implied addressing mode를 사용한다). 예를 들어 0을 오퍼랜드 스택에 넣기 위해서는 iconst_0 명령을 사용한다. iconst_m1, iconst_*n* (단, *n* 은 0부터 5 사이 값)등이 모두 이 같은 데이터 이동에 사용된다.

8비트로 나타내어질 수 있는 정수형 상수 역시 상수 풀에 두지 않고 명령어 내에 포함한다 (즉 immediate addressing mode를 사용한다). 예를 들어 100을 오퍼랜드 스택에 넣기 위해서는 bipush 100 명령을 사용한다. 마찬가지로 16비트로 나타내어질 수 있는 정수형 상수도 상수 풀에 두지 않고 명령어 내에 포함한다. 예를 들어 500을 오퍼랜드 스택에 넣기 위해서는 sipush 500 명령을 사용한다.

다음은 상수 풀에서 오퍼랜드 스택으로 상수 값을 옮기는데 사용되는 바이트 명령어들을 모은 것이다.

```
ldc, ldc_w, ldc2_w, aconst_null, iconst_m1,
iconst_0, iconst_1, iconst_2, iconst_3,
iconst_4, iconst_5, lconst_0, lconst_1,
fconst_0, fconst_1, fconst_2, dconst_0,
dconst_1, bipush, sipush
```

3.2 오퍼랜드 스택과 지역변수배열 사이

오퍼랜드 스택과 지역변수배열은 동일한 자바 스택 프레임 내에 위치하고 있으며, 그림 2와 같이 양방향으로 데이터 이동이 일어난다.

먼저 지역변수배열에서 오퍼랜드 스택으로의 데이터 이동에 대해 알아보자. 이런 이동은 자바 프로그램에서 지역변수들에 대해 연산을 실행할 때 주로 일어난다. 예를 들어 지역변수 i 와 j를 더하기 위한 자바 프로그램은 i+j 와 같으며, 바이트코드로는 다음과 같이 번역된다.

```
iload #n
```

```
iload #m
iadd
```

처음 두 줄의 iload #n 과 iload #m 은 각각 지역변수배열의 n번째 항목과 m번째 항목을 오퍼랜드 스택으로 가져오는 명령이다 (변수 i 와 j가 각각 지역변수배열의 해당 위치에 있는 것으로 가정하였다).

간단한 메소드인 경우 지역변수의 개수가 4개를 넘지 않는다는 점에 착안하여 4개까지의 지역변수배열은 iload 명령 뒤에 별도 파라미터를 두지 않고 명령어 의미 내에 포함하도록 했는데, 예를 들어 0번째 지역변수배열 항목을 오퍼랜드 스택에 넣기 위해서는 iload_0 명령을 사용하며, 3번째 항목은 iload_3 명령을 사용한다.

다음은 지역변수배열에서 오퍼랜드 스택으로 데이터 값을 옮기는데 사용되는 바이트 명령어들을 모은 것이다.

```
iload, lload, fload, dload, aload, iload_0,
iload_1, iload_2, iload_3, lload_0,
lload_1, lload_2, lload_3, fload_0,
fload_1, fload_2, fload_3, dload_0,
dload_1, dload_2, dload_3, aload_0,
aload_1, aload_2, aload_3
```

지역변수배열에서 오퍼랜드 스택으로의 데이터 이동은 필드를 접근하거나 메소드를 호출할 때도 발생한다. 다음 자바 코드에서 f는 필드, 즉 인스턴스 변수며, m은 메소드라고 하자.

```
f = 0;
m(f);
```

이 코드는 다음 바이트코드로 번역된다. 여기서 i, j 는 각각 필드 f와 메소드 m을 접근하기 위한 필드 테이블 및 메소드 테이블의 인덱스 값이다.

```
aload_0
iconst_0
putfield #i
aload_0
dup
getfield #i
invokevirtual #j
```

위 바이트코드들에서 알 수 있듯이 필드를 접근할 때나 메소드를 호출할 때마다 지역변수배열 0번째 항목에 들어있는 현재 객체를 가리키는 참조자 값이 오퍼랜드 스택으로 이동된다.

이제는 반대 방향, 즉 오퍼랜드 스택에서 지역변수배열로의 데이터 이동에 대해 알아보자. 이런 이동은 자바 프로그램에서 지역변수들의 값을 변경할 때 주로 일어난다. 그림 2에서 보인 바와 같이 상수 값을 지역변수에 할당할 때나 또는 어떤 연산의 결과를 지역변수에 할당할 때, 그리고 힙 영역에 있는 필드값을 지역변수에 할당할 때 모두 오퍼랜드 스택에서 지역변수배열로의 데이터 이동이 발생된다. 다음 자바 코드를 보자. 여기서 i, j 는 지역변수들이며, f는 필드를 의미한다고 가정했다.

```
i = 0; i = i+j; i = f;
```

이것을 바이트코드로 번역하면 다음과 같다.

```
iconst_0
istore_1
iload_1
iload_2
iadd
istore_1
aload_0
getfield #i
istore_1
```

오퍼랜드 스택에서 지역변수배열로의 데이터 이동에는 istore 등의 명령이 사용된다. 간단한 메소드인 경우 지역변수의 개수가 4개를 넘지 않는다는 점에 착안하여 4개까지의 지역변수배열은 istore 명령 뒤에 별도 파라미터를 두지 않고 명령어 의미 내에 포함하도록 했는데, 예를 들어오퍼랜드 스택의 내용을 0번째 지역변수배열 항목에 넣기 위해서는 istore_0 명령을 사용하며, 3번째 항목은 istore_3 명령을 사용한다.

다음은 오퍼랜드 스택에서 지역변수배열로 데이터 값을 옮기는데 사용되는 바이트 명령어들을 모은 것이다.

```
istore, lstore, fstore, dstore, astore,
istore_0, istore_1, istore_2, istore_3,
lstore_0, lstore_1, lstore_2, lstore_3,
fstore_0, fstore_1, fstore_2, fstore_3,
dstore_0, dstore_1, dstore_2, dstore_3,
astore_0, astore_1, astore_2, astore_3
```

3.3 오퍼랜드 스택과 힙 사이

먼저 오퍼랜드 스택에서 힙으로의 데이터 이동에 대해 알아보자. 이런 이동은 자바 프로그램에서 필드의 값을 변경할 때 일어나며, 3.2절에서 살펴본 바와 같이 바이트코드로는 putfield 명령에 의해 이루어진다. 이 명령을 내리기 전에 객체를 가리키는 참조자 값과 변경할 값이 먼저 오퍼랜드 스택 상에 놓여져 있어야 한다.

배열 값의 변경도 오퍼랜드 스택에서 힙 영역으로의 데이터 이동을 필요로 한다. JVM 규격에 따르면 배열은 힙 영역에 위치한다. 배열 값을 변경시키는 다음 자바 코드를 보자.

```
buf[10] = 0;
```

이것을 바이트코드로 번역하면 다음과 같다.

```
aload_1
bipush 10
iconst_0
iastore
```

즉 배열 항목을 위한 오퍼랜드 스택에서 힙 영역으로의 데이터 이동에는 iastore 등 명령이 사용되며, 변경할 값은 물론 변경할 배열을 가리키는 참조자와 인덱스 값도 미리 오퍼랜드 스택에 있어야 한다.

다음은 오퍼랜드 스택에서 힙 영역으로 데이터 값을 옮기는데 사용되는 바이트코드 명령어들을

모은 것이다.

```
putfield, aastore, bastore, castore,
dastore, fastore, iastore, lastore, sastore
```

이제는 반대 방향, 즉 힙에서 오퍼랜드 스택으로의 데이터 이동에 대해 알아보자. 이런 이동은 자바 프로그램에서 필드의 값을 조회할 때 일어나며, 3.2절에서 살펴본 바와 같이 바이트코드로는 getfield 명령에 의해 이루어진다. 이 명령을 내리기 전에 객체를 가리키는 참조자 값이 먼저 오퍼랜드 스택 상에 놓여져 있어야 한다.

배열 값의 조회도 힙 영역에서 오퍼랜드 스택으로의 데이터 이동을 필요로 한다. 배열 값을 조회하는 다음 자바 코드를 보자.

```
buf[10];
```

이것을 바이트코드로 번역하면 다음과 같다.

```
aload_1
bipush 10
iaload
```

즉 배열 항목을 위한 힙 영역에서 오퍼랜드 스택으로의 데이터 이동에는 iaload 등 명령이 사용되며, 변경할 배열을 가리키는 참조자와 인덱스 값도 미리 오퍼랜드 스택에 있어야 한다.

다음은 힙 영역에서 오퍼랜드 스택으로 데이터 값을 옮기는데 사용되는 바이트코드 명령어들을 모은 것이다.

```
getfield, aaload, baload, caload, daload,
faload, iaload, laload, saload
```

3.4 오퍼랜드 스택에서 오퍼랜드 스택으로

그림 2의 6번 화살표에서 볼 수 있듯이 오퍼랜드 스택에서 오퍼랜드 스택으로의 데이터 이동도 발생된다.

대표적 경우로 오퍼랜드 스택의 값을 중복시키는 dup, dup_x1, dup_x2를 들 수 있으며, 64비트 크기의 값에 대한 중복 명령인 dup2, dup2_x1, dup2_x2 도 여기에 해당된다. 그밖에도 오퍼랜드 스택의 상위 2개의 항목 값을 서로 맞바꾸는 swap 명령도 해당된다.

또다른 경우는 오퍼랜드 스택에 놓인 값들에 대해 산술논리연산을 수행하는 것을 들 수 있다. 산술논리연산은 연산의 대상이 되는 오퍼랜드를 오퍼랜드 스택에서 취하며, 연산의 결과도 오퍼랜드 스택에 저장한다. 여기에는 iadd를 비롯한 많은 산술논리연산 바이트코드들이 해당된다.

다음은 오퍼랜드 스택에서 오퍼랜드 스택으로 데이터 값을 옮기는데 사용되는 바이트코드 명령어들을 모은 것이다.

```
dup, dup_x1, dup_x2, dup2, dup2_x1,
dup2_x2, swap, iadd, ladd, fadd, dadd,
isub, lsub, fsub, dsub, imul, lmul, fmul,
dmul, idiv, ldiv, fdiv, ddiv, irem, lrem,
frem, drem, ineg, lneg, fneg, dneg, iand,
land, ior, lor, ixor, lxor, ishl, lshl,
ishr, lshr, iushr, lushr, i2b, i2c, i2s,
i2d, i2l, i2f, f2d, f2l, f2i, l2d, l2f,
l2i, d2l, d2f, d2i
```

3.5 지역변수배열에서 지역변수배열로

마지막으로 그림 2의 7번 화살표에서 나타낸 지역변수배열에서 지역변수배열로의 데이터 이동에 대해 알아보자.

지역변수배열의 값을 바꾸는 거의 모든 명령은 반드시 오퍼랜드 스택을 경유하며, 지역변수배열에서 바로 지역변수배열로 데이터가 이동(또는 변경)하는 경우는 iinc 명령이 유일하다. iinc 는 지역변수배열의 특정 항목 값을 증가 또는 감소시키는 목적으로 사용된다.

IV 결 론

자바가상기계에서 일어나는 동작의 많은 부분이 데이터 이동에 관한 것이다. 보다 효율적인 JVM의 개발을 위해서는 데이터 이동에 대한 연구가 필수적이다. 본 논문에서는 JVM이 정의하는 바이트코드를 분석하여 오퍼랜드 스택, 지역변수배열, 힙, 그리고 상수 풀 간에 일어나는 데이터 이동에 대해 고찰하였다. JVM에서 일어나는 데이터 이동의 중심이 오퍼랜드 스택임을 확인하였고, 자바 프로그램이 바이트코드로 번역될 때 각 문장에 대해 어떤 데이터 이동이 일어나는지에 대해 분석하였다.

참 고 문 헌

[1] T. Lindholm and F Yellin, *The Java Virtual Machine Specification*, Second Edition, Addison-Wesley, 1999.

[2] 양희재, 자바가상기계, 한국학술정보(주), 2001년 3월, ISBN 89-5520-342-4

[3] H. Yang, "Memory Access Pattern in Java Virtual Machine," *4th Asia Pacific Int'l Symposium on Information Technology (APIS2004)*, 2005. 1.