Integer Java Virtual Machine のエミュレータ試作

The construction of the emulator of Integer Java Virtual Machine

http://www.tani.cs.chs.nihon-u.ac.jp/g-2004/karaika/

谷 研究室　森　考史
TAKASHI MORI

概要

Java の実行の高速化のために Java Virtual Machine のサブセットである Integer Java Virtual Machine (IJVM) のエミュレータを作成した。またその可視化を行なった。

1　はじめに

Java 言語で記述されたプログラムは、バイトコードコンパイラで Class ファイルへと変換され JVM(Java Virtual Machine) という仮想マシン上で実行される。そのため、各プラットフォームに対応した JVM さえあれば Java 言語で記述されたプログラムはどんなプラットフォーム上でも実行できる。Java 言語の特徴の一つである、プラットフォーム非依存ということを実現するのが JVM である。

様々なアプリケーションの実行時における JVM の各命令の使用頻度や使用順序の調査は、JVM の高速化につながると予想される。例えば、頻度の高い命令を高速化することによって低コストで JVM のパフォーマンス向上が期待できる。本研究では、その足がかりとして JVM のサブセットである Integer Java Virtual Machine(IJVM) の各命令の使用頻度や使用順序を調査するエミュレータを作成した。

2　JVM

Class ファイルフォーマットに従った Class ファイルを読み込み、JVM の命令を解釈し実行する。実在する計算機のように、命令セットが用意され、命令の実行時に様々な記憶領域の操作が行われる。

2.1　Class ファイル

Class ファイルフォーマットに従ったバイナリファイル。８ビットのバイトストリームからなっている。各 Class ファイルには、クラスやインターフェースの定義が一つ保持されている。通常は Java 言語で記述されたコードをバイトコードコンパイラで生成する。しかし、Class ファイルの生成方法については決められているわけではない。

2.1.1　Class ファイルフォーマット

Class ファイルが従わなければならない形式。ハードウェアやオペレーティング・システムに依存しないバイナリフォーマット。

2.2　JVM の命令

実行する操作を定義した１バイトのオペコードに続いて、引数、あるいは操作が用いるデータとなるオペランドをゼロ個以上続けたものから構成されている。オペコードは約２００種類ある。命令の多くはオペランドを持たず、オペコードのみで構成されている。

2.3　Java バイトコード

JVM の命令の集合によって記述された実行形式のプログラム。Class ファイルフォーマットの中のメソッドの情報が記述されている部分の属性の種類の一つである Code 属性のなかに記述されている。

2.4　JVM の構造

JVM の定義は「クラスファイルを正しく実行できること」であるので、これが満たされる限り、JVM の実装は自由である。ここで記述している JVM の構造は、たくさんある JVM の実装の仕方の中の１つでしかない。

2.4.1　メソッドエリアとヒープ

JVM が Class ファイルを読み込むと、クラスをメモリに展開する。メモリ中のクラスの置き場所をメソッドエリアと呼ぶ。そのクラスのインスタンスを生成すると、そのインスタンスはヒープという場所に置かれる。

2.4.2 Java スタック

JVM の各スレッドには、スレッドと同時に生成される非公開の Java スタックが保持される。Java スタックにはフレームが格納される。ローカル変数や中間的な結果を保持したり、メソッドの起動やリターンの一部を担っている。

2.4.3 フレーム

フレームとは、ダイナミックリンクの実行、メソッドからの値のリターン、例外のディスパッチ、データや中間的な結果を格納するために使用される。メソッドが起動されるたびに新たなフレームが生成される。フレームは、そのメソッドが終了する際に、正常終了または途中終了する（キャッチされない例外がスローされたか）に関わらず破棄される。各フレームには、自身のローカル変数配列（変数の配列）、自身のオペランドスタック (last-in-first-out のスタック)、カレントメソッドに対する実行時コンスタントプールへの参照が保持される。

2.4.4 ローカル変数配列

フレームにおけるローカル変数配列の長さは、コンパイル時に決定され、クラスやインタフェースーバイナリ表現中にある該当フレームに対応づけられたメソッドのコードとともに提供される。ローカル変数は、それ一つで boolean 型、byte 型、char 型、short 型、int 型、float 型、reference 型、returnAddress 型のいずれかの値を保持することができる。そして、ローカル変数のペアによって、long 型、double 型のいずれかの値を保持することができる。ローカル変数はインデックスによってアドレス付けられる。

2.4.5 オペランド・スタック

オペランド・スタックは、それが保持されるフレームの生成時点では空になっている。JVM にはオペランド・スタックからオペランドを取得し、操作を行なった結果をオペランド・スタックに再び格納する命令が用意されている。また、オペランド・スタックはメソッドに引き渡されるパラメータを準備したり、メソッドの結果を受け取るためにも用いられる。

3 Integer Java Virtual Machine

Structured Computer Organization (Tanenbaum, 1998) のセクション 4 で導入された Java バイトコードのサブセットのためのアセンブラおよびインタープリタから成るもの。

3.1 Integer Java Virtual Machine の命令セット

20 種類の命令がある。その 20 種類の命令の詳細は別途表参照。

4 今後の課題

JVM の他の命令（IJVM に存在しない命令）も実装し JVM のエミュレータを作成する。さらに JVM の高速化のために、様々なアプリケーションの実行時における JVM の各命令の使用頻度及び使用順序の調査を行う。

参考文献

(i) ティム リンドホルム (著), フランク イェリン (著), Tim Lindholm (原著), Frank Yellin (原著), 村上雅章 (翻訳), Java 仮想マシン仕様 The Java series, 2001

(ii) ジョン メイヤー (著), 鷲見 豊 (著), トロイ ダウニング (著), Jon Meyer (原著), Troy Downing (原著) ,JAVA バーチャルマシン THE JAVA SERIES, 1998

付録

Class ファイル構造体：ClassFile {

```
u4 magic;
u2 minor_version;
u3 major_version;
u2 constant_pool_count;
cp_info constant_pool[constant_pool_count-1];
u2 access_flags;
u2 this_class;
u2 super_class;
u2 interfaces_count;
u2 fields_count;
field_info fields[fields_count];
u2 interfaces[interfaces_count];
u2 fields_count;
field_info fields[fields_count];
u2 methods_count;
method_info methods[methods_count];
u2 attributes_count;
attribute_info attributes[attributes_count];
```

}

Class ファイル構造体の項目：

magic ： Class ファイルフォーマットを識別するマジックナンバーが保持されている。

minor_version, major_version ： この Class ファイルにおけるマイナーバージョン番号およびメジャー・バージョン番号。

constant_pool_count ： この項目の値は constant_pool テーブル中のエントリ数よりも 1 つ大きいものとなる。

constant_pool[] ： Class ファイル構造体およびその部分構造体内で参照されるさまざまな文字列定数、クラス名やインタフェース名、フィールド名、その他の定数を表現するための構造体テーブル。

access_flags ： アクセス許可、および該当のクラスやインタフェースにおける属性を記述するために用いられるマスク・フラグである。

this_class ： この項目の値は、constant_pool テーブルに対する有効なインデックスとなっていなければならない。該当インデックスによって参照される constant_pool 中のエントリは、この Class ファイルによって定義されているクラスやインターフェースを表現したものでなければならない。

super_class ： クラスの場合における項目 super_class の値は、ゼロあるいは constant_pool テーブルに対する有効なインデックスでなければならない。直接のスーパークラスの情報を保持。

interfaces_count ： この項目の値によって、このクラス型やインタフェース型に対する直接のスーパーインタフェースの数が示される。

interfaces[] ： 直接のスーパーインタフェースの情報を保持。

fields_count ： この項目の値によって、fields テーブルにおけるフィールドの数が示される。

fields[] ： 自クラスで定義されたフィールドについてのすべての情報を保持。

methods_count ： この項目の値によって、methods テーブルにおけるメソッドの数が示される。

methods[] ： 自クラスで定義されたメソッドについてのすべての情報を保持する。継承してオーバーライドしていないものは含まない。

attributes_count ： この項目の値によって、このクラスの attributes テーブルにおけるアトリビュートの数が示される。

attributes[] ： その他の様々な情報をほじするもの。定義済み属性と、それ以外のユーザー定義属性に分かれます。命名規則とフォーマットに従う限り、ユーザーは自由に新たな属性を使用して構わない。

Integer Java Virtual Machine の命令

オペコード	ニーモニック	操作内容
0x10	BIPUSH byte_exp	byte_exp をオペランド・スタックへとプッシュする
0x59	DUP	オペランドスタックの先頭にある値がコピーされ、オペランドスタックへとプッシュされる
0xA7	GOTO label	無条件分岐を行う
0x60	IADD	オペランドスタックから 2 つの値をポップして、それらの合計をプッシュします

オペコード	ニーモニック	操作内容
0x7E	IAND	オペランドスタックから 2 つの値をポップして、それらの論理積をプッシュします
0x99	IFEQ label	値をオペランドスタックからポップし、それが 0 なら分岐する
0x9B	IFLT label	値をオペランドスタックからポップし、それが 0 未満なら分岐する
0x9F	IF_ICMPEQ label	２つの値をオペランドスタックからポップし、それらが等しいならば分岐する
0x84	IINC varnum_exp, byte_exp	vamum_exp で示されたローカル変数に、byte_exp を int へ符号拡張したものを加算する
0x15	ILOAD vamum_exp	vamum_exp で示されたローカル変数をオペランドスタックへとプッシュする
0xB6	INVOKEVIRTUAL method	クラスに基づくディスパッチを行い、インスタンスメソッドを起動する
0x80	IOR	オペランドスタックから 2 つの値をポップして、それらの論理和をプッシュします
0xAC	IRETURN	メソッドから int をリターンする
0x36	ISTORE varmum_exp	値をオペランドスタックからポップし、それを varmum_exp で示すローカル変数にセットする
0x64	ISUB	オペランドスタックから 2 つの値をポップして、それらの差をプッシュする
0x13	LDC_W constant_exp	constant_exp で示される定数をオペランドスタックへとプッシュする
0x00	NOP	何もしない
オペコード	ニーモニック	操作内容
0x57	POP	オペランドスタックの先頭にある値をポップする
0x5F	SWAP	オペランドスタックの先頭にある 2 つの値を交換する
0xC4	WIDE	補助バイトを使用したローカル変数インデックスの拡張をする