高速実行可能な低レベル命令セット仮想計算機の設計†

4H－9

奥平 雄吾　足立原 直　脇田 建　佐々 政孝

東京工業大学 数理計算科学専攻

1 はじめに

ネットワーク上に混在する様々なアーキテクチャのマシンを利用して、分散計算、移動計算あるいはプログラムの移植をしたいという要求は高い。異機種マシン間で一つのコードを実行するためには、コードに可搬性を持たせることが必要である。

P-codeを中間コードとするPascal、Smalltalk、Javaなどの言語処理系は、コードに可搬性を持たせるために、仮想計算機(VM)を用いる。VMには、インタプリタにより逐次実行するものと、JITにより一括変換し実行するものがある。前者は、各VM命令をソフトウェアでfetch、decodeするため、実マシンの上でコンパイルしたコードと比べ実行速度が10倍程度遅い。後者は、実行時の(1)**最適化が困難**、(2)**変換のオーバヘッドが無視できないほど大きくなる**などの問題がある。また、前者後者ともVMの扱う命令が高級なため、(3)**言語依存度が大きく言語ごとにVMを用意しなくてはならない**。(1)(2)(3)の問題は、特定プログラミング言語に特化して設計された**高レベルVM**である点に起因する。

一方、OmniVM[1]は、RISCのような命令を扱う低レベルVMであるため、プログラミング言語に特化しない。また、OmniVMはコンパイラにより最適化済のVM用コード(VMコード)を実マシンコードに一括変換し実行することで高速実行を実現している。OmniVMでは、実マシン上でコンパイルされたコードを実行する場合と比べ、10 ～ 20 %程度のオーバヘッドで実行可能である。

本研究では、10 %程度のオーバヘッドによる実行、アーキテクチャ依存の最適化、64bitアーキテクチャへの対応、VMコード量の削減、など、OmniVMより高性能なVMを目指し、言語非依存で高速実行可能な低レベルVMを設計する。

2 システムの概要

2.1 VM命令セット

本VMは、実マシン命令セットのようなレジスタやスタックフレームを扱う低レベルな命令セットを持つ。アーキテクチャの差異を吸収するため、実マシン命令より抽象度をあげた命令も含まれるが、基本的にはRISCのような単純な命令セットである。

2.2 システム構成

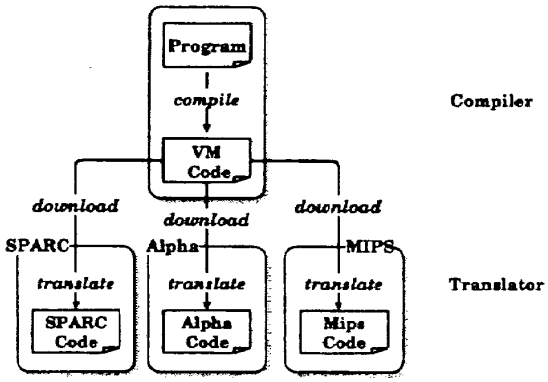

図 1: システムの構成

本システムの構成を図1に示す。本システムはコンパイラとVMの実現からなる。コンパイラは、様々な言語のプログラムをVMコードに変換する。VMを実現するために、SPARC、Alpha、MIPSなど様々なアーキテクチャ上でトランスレータがVMコードを実マシンコードに変換し、実マシンが実マシンコードを実行する。この設計により、1節で挙げた3つの問題点は以下のように解決される。

(1) 既存の最適化技術の利用　VMコードが低レベルなため、コンパイラを作成する際には、lccなどの既存のretargetableなコンパイラにVM用コード生成器を追加するだけですむ。コンパイラは、既存の最適化技術を利用して、コードにアーキテクチャ独立な最適化(定数伝播、演算の強さの軽減、ループの最適化、レジスタ割付、等)を施す。実装にかかる工数は、VM用コード生成器を追加するだけなので少ない。

(2) 高速に実マシンコードへ変換　コンパイラによりアーキテクチャ独立の最適化は完了しているので、トランスレータはアーキテクチャ依存の変換を行うだけでよい。また、VMコードが低レベルなため、トランスレータは多くのVM命令を1対1で実マシン命令に変換することができる。よって、トランスレータはVMコードを高速に変換することが可能であり、変換のオーバヘッドは少なくなる。

(3) プログラミング言語独立性　様々な言語で書かれたプログラムが、全て同じVMのコードに変換されるため、各言語ごとにVMを用意する必要がない。実マシンコードへの変換に伴う諸問題は3 章の方法で克服できる。

VMの実行性能　トランスレータはコンパイラにより最適化されたVMコードを実マシンコードにほぼ1対1変換するため実マシンコードの実行速度は速い。また、(2)より変換のオーバヘッドは少なく変換速度も速い。

---

†Design of High Performance Virtual Machine with Low Level Instruction Set

Y. Okudaira, T. Adachihara, K. Wakita, and M. Sassa

Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology

3 VM アーキテクチャ

レジスタ長、命令セット、スタックフレームの構成などはアーキテクチャにより異なる。様々なアーキテクチャのマシン上で VM を動かすためには、実マシンコードに変換する時に整合性がとれるように VM のアーキテクチャを設計しなくてはならない。以下、設計する VM のアーキテクチャについて述べる。

レジスタ 本 VM が所有するレジスタは、64 bit 整数レジスタ 32 個、64 bit 浮動小数点レジスタ 32 個である。このうち何個かはスタックポインタ、グローバルポインタなどシステム予約として使用される。

データサイズ毎の命令 本 VM では、扱うデータのサイズにより、8, 16, 32, 64 bit の命令を用意する。データのサイズごとに命令が異なるので、実マシンのレジスタ長が VM のレジスタ長である 64 bit と異なっていても、以下のように実マシンに合わせた効率の良いマシンコードに変換できる。

- VM コードからレジスタ長 32 bit の実マシンコードへ変換、64bit データのロードは 2 命令に変換

```
LD32 [%vr1+0], %vr2   ⇒   LD [%r1+0], %r2
LD64 [%vr1+4], %vr3   ⇒   LD [%r1+4], %r3
                          LD [%r1+8], %r4
```

エンディアン中立 本 VM のアドレス空間は 32 bit で表される。ワード内の下位バイトへのアクセス命令は、エンディアンにより異なるアドレスをアクセスすることになる。VM コードを、実マシンのエンディアンに合わせた実マシンコードに変換するため、VM ではエンディアンの概念をなくすことにする [2]。このため、VM のメモリアクセス命令に、下位バイトロード命令などを用意し、エンディアン中立を保つ。下位バイトロードはエンディアンにより以下のように変換される。

- 1 ワードデータの下位 1 バイトをロードする

```
LDLOW8OF32 [%vr1+ 0], %vr2
        little endian⇒ LDB [%r1+ 0], %r2
           big endian ⇒ LDB [%r1+ 3], %r2
```

アーキテクチャ独立な命令 アーキテクチャにより異なる仕様をもつものが存在する。例えば、SPARC ではステータスレジスタが存在し、条件判定の結果を保存するフィールドを持つ。条件判定に続く分岐命令では、ステータスレジスタを参照し分岐する。一方、MIPS などではステータスレジスタは存在せず、汎用レジスタを参照して分岐する。本 VM は test & jump をひとまとめにした命令を提供し、特殊なレジスタの有無に関わらず正しく実マシンコードに変換することを可能とした。

ABI 中立 ある種の処理をするときに推奨されている命令セット、関数呼出しの際の引数の受渡し方、レジスタの使用法、メモリ・スタックフレームの構成などの仕様を ABI (Application Binary Interface) という。

ABI はアーキテクチャや OS によって異なるため、トランスレータは VM コードを実マシンの ABI に合わせて変換しなくてはならない。そこで本 VM では、特定の ABI に依存しない ABI を提供し、VM の ABI に従ったコードを各種 ABI に従ったコードに変換する。

本 VM のロード、ストア命令は「ベースレジスタ + オフセット」を唯一のアドレッシングモードとする。大域変数はグローバルポインタ相対でアクセスする。

Calling Convention、即ち関数呼出しの際の引数の受渡し、レジスタの使い方などについて、本 VM ではスタックやレジスタを使用せず、専用命令により行う。本 VM の Calling Convention は ABI 中立であるため、実マシンコードに変換する時には各アーキテクチャの Calling Convention に合わせることになる。また、専用命令を用意したことにより、VM の関数のスタックフレームにはローカル変数と戻り番地を保存することになる。

- 呼びだし側 (VM コード側)

```
SETARGS %vr4, [%vr2+8], %vf1
                        // 3 つが引数
CALL foo                // 関数 foo 呼び出し
```

- 呼び出された側 (VM コード側)

```
CALLEESAVE %vr5, %vr1, %vf3
                        // レジスタ Save
GETARGS 1, %vr5         // 第 1 引数を %vr5 へ
GETARGS 2, %vr1         // 第 2 引数を %vr1 へ
GETARGS 3, %vf3         // 第 3 引数を %vf3 へ
```

関数の返り値については専用レジスタで渡す。返り値が構造体の場合は渡し方がアーキテクチャにより異なるので本 VM では専用命令を使用する。呼び出す側では、スタック上で返り値を書き込む位置を指定する。呼ばれた側では、そのアドレスに返り値をしまう。

関数が呼びだし側に戻る際には関数の先頭で退避したレジスタを CALLEERECOVER 命令により回復し、呼びだし側に戻る。

命令長可変 本 VM の命令セットは上述の CALLEESAVE のような可変長のオペランドをもつ命令があるために、命令長は可変となる。これにより、VM の各命令は命令フィールドの無駄な領域が少なくなり、VM コード量を減らすことが可能となる。

4 まとめ

異機種混成環境でプログラムを実行するために、言語独立で高速実行可能な VM を設計した。VM の扱う命令セットを低レベルにしたことで、変換速度、実行速度とも効率がよいものとなる。

今後の課題は、コンパイラでのレジスタ割り付けをトランスレータ側に持ってきた場合などと比較し、現在の実行性能が充分であるかどうか検討することである。また、実用に向けて、グローバルな最適化の実装、セキュリティ機構の導入、様々な言語のコンパイラの作成、様々な機種用の VM トランスレータの作成なども行っていく予定である。

参考文献

[1] A. Adl-Tabatabai, G. Langdale, et al: Efficient and Language-Independent Mobile Programs, PLDI '96, May 1996, pp. 127-136.

[2] 足立原 直, 奥平 雄吾, 他: 低レベル命令セット仮想計算機を利用した混成環境におけるプロセス移送, 第 57 回情報処理学会全国大会論文集 4H-10(1998).