4H－10

# 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†

足立原 直　奥平 雄吾　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

異機種混成環境におけるプロセス移送 (*process migration*) は、プロセスをアーキテクチャやOSが異なる計算機に移す実装技術である。これまでに、異機種混成環境におけるプロセス移送を可能にする様々な研究 [2] がされ、利用できるプログラミング言語を特定の型安全言語に限定したEmerald [3]、Javaにおけるスレッド移送 [5]、型安全でない言語 (ANSI-C) を利用できるが一部のデータが再現されないTui System [4] などが実現された。しかし任意の言語を利用可能にするシステムは、非型安全言語における移送前後でのデータの再現性の問題と、データの再現のための処理にかかる実行時間の問題があるため、実用に至るものはまだない。

本稿では、実計算機レベル命令セットをもつ仮想計算機 (VM) [1] を用い、異機種混成環境において、任意のプログラミング言語を利用可能なプロセス移送システムを提案する。

VMは、特定の高級プログラミング言語に依存しないように、実計算機命令に近い命令セットをもつ。また異機種混成環境に対応するため、メモリ上のデータについて、バイト順 (endian) を意識せずに参照できる命令を備える。

本システムは [1] に、移送命令の追加と、すべての計算機上でのメモリ上のデータの表現と配置を同一にするという、2つの拡張をおこなう。同一のデータ表現と配置により、メモリ上の実行時データをそのまま移すことで、すべてのデータの値を変化させることなくプロセス移送を可能にする。

## 2　本システムの概略

本システムの構成は図1のようになる。任意の言語で記述されたソースプログラムは、コンパイラにより最適化されたVMコードに変換される。システムはVMコードを実計算機コードに変換し、実行を開始する。その後移送命令が実行されると、システムはプログラムの実行を中断し、機種非依存であるVMコードと、実行時データを移送先へ移す。移送先ではVMコードを新たに実計算機コードに変換し、実行を再開する。

## 3　本システムのプロセス移送方式

プロセスの移送は、プロセスのコードと実行時状態を移すことである。実行時状態は、プログラムカウンタ (pc)、

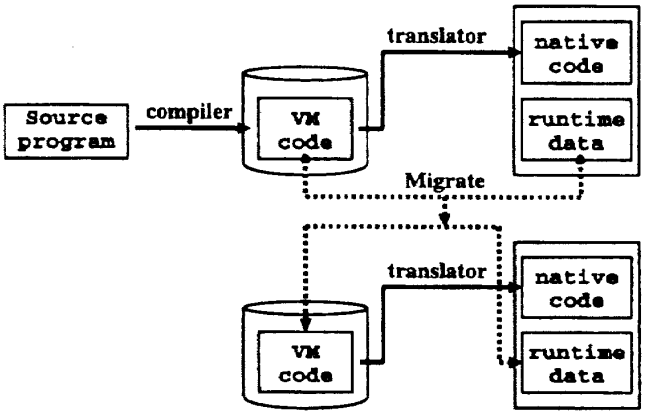


図 1: プロセス移送システム全体図

レジスタ、スタック、ヒープ、静的データ領域の5つから構成される。異機種混成環境では、これらすべてを機種に依存しない方法で移すことが必要である。本節では本システムにおける、コードと実行時状態の各要素 (pc、レジスタ、スタック、ヒープ、静的データ領域) の移送方式を述べる。

**コードについて**　コードの移送はVMコードを用いる。VMコードは実計算機コードに変換されるため、高速な実行が可能である。すべてのVMの命令が、実計算機命令に1対1で変換されるわけではないので、後述するpcやコードを指すポインタの処理のために、VMコード上の命令の位置 (ラベル) と、実計算機コードにおけるアドレスとの対応表 (ラベル・アドレス対応表) を、実計算機コード変換時に作成する。

**pcについて**　pcについては、移送前の実計算機コードにおけるpcに対応する、VMコード上の位置を求め、移送後に再び実計算機コードにおけるpcに変換する。本システムでは、プロセス移送はプログラム中の任意の位置に記述できるプロセス移送命令の実行によってのみ起きるため、すべての移送命令にラベルを付けておくことにより、移送が起きた命令のラベルと対応する実アドレスをラベル・アドレス対応表から求めることができる。

**レジスタについて**　レジスタについては、移送前にレジスタに乗っている値をメモリ上に待避し、移送後は必要に応じてメモリ上から回復するようにして、事実上レジスタの移送を行わない。

**スタック、ヒープ、静的データ領域について**　本システムで利用する仮想計算機の命令セットは実計算機命令に近い。実計算機命令レベルで考えた場合、メモリ上における機種毎のデータ表現の違いは、endianだけである。そこで本システムでは、スタック、ヒープ、静的データ

†Process Migration for a Low Level Instruction Set Virtual Machine in a Heterogeneous Environment,
T. Adachihara, Y. Okudaira, K. Wakita, and M. Sassa,
Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology.

```
load                        [memory], reg1
data_to_machine_endian      reg1
            ........
machine_to_data_endian      reg1
store                       [memory], reg1
```

図 2: endian 変換命令の挿入

領域におけるデータの表現と配置を、全ての計算機上で同一にし、これら3つをそのままの表現と配置で移す。

この方法は、データの再現のために型情報を利用しないため、非型安全言語におけるデータの再現性の問題が解決される。例えばC言語において型変換代入された値でも、移送前後で値が変わることがない。また移送時にデータ表現を機種非依存の形に変換する必要が無いため、移送時のコストが低い。しかし、データの endian と機種本来の endian が異なるという問題と、移送後に書き換えなくてはならないデータの管理についての問題と、スタックやヒープのデータ表現と配置に関する取り決め (ABI) が機種本来で定められたものと異なるという問題の、3つの問題がある。それらの解決方法を以下に述べる。

**データの endian と機種本来の endian の相違**　メモリ上のデータをそのままの表現で移すことで、データの endian と機種本来の endian が異なる可能性がある。この場合はメモリ参照毎に endian を変える必要があるため、実計算機コード変換時に、ロード命令の直後にデータを機種本来の endian にする命令を挿入し、ストア命令の直前にデータ本来の endian にする命令を挿入する (図2)。endian 変換による実行時コストの増加については、改善案を 4 節で述べる。

**移送後に書き換えなくてはならないデータの管理**　スタック、ヒープ、静的データ領域に含まれるデータのうち、コードを指すポインタと、関数呼出しの戻り番地は、移送後に実計算機コードになった時点で決定する新しい値に書き換える必要がある。この処理を行うため、ラベル・アドレス対応表を用いて、コードを指すポインタと戻り番地が、表中の対応するエントリを指すようにする。つまりコードを指すポインタと戻り番地は表のエントリを経由しての間接参照になる。ラベル・アドレス対応表自体の配置は全ての計算機上で同一にする。

**機種本来の ABI との違い**　スタックやヒープ中のデータの表現と配置は、機種本来で定められた ABI と異なるため、機種本来の ABI に従うシステムコールや、libc 等の機種標準ライブラリを利用することができない。このためシステムコールやライブラリについては、ABI の違いを吸収するため、本システム専用ライブラリを各機種毎に用意する。

## 4 バイト順を変える実行時コスト

機種本来の endian とデータの endian が異なる場合に、全てのメモリとレジスタ間のロード／ストア時に endian 変換命令を実行することは、実行効率のボトルネックになる。例えば、32 bit の little-endian データを、big-endian 表現に変換するためには SPARC で 10 命令を要する。この問題の改善案を 3 つ示す。

**命令スケジューリング**　実計算機コード変換時に、endian を変える命令列をスケジューリングすることで、実行に必要なクロック数を減らす。

**不必要な変換命令生成の抑制**　プロセス移送命令がない葉の関数における局所変数など、プロセス移送が実行されない区間でのみ参照されるデータ領域へのアクセスは、endian を変える必要がない。そのようなデータ領域へのアクセスについて、コンパイル時に解析できる個所は、endian を変えない専用ロード／ストア命令を出力し、実計算機コード変換時に endian を変更する命令を挿入しないようにする。

**ソフトウェアキャッシュメモリの導入**　一度機種本来の endian に変更したデータはキャッシュ上に値を保存しておき、再度アクセスがあった時にキャッシュから値を読み出すようにする。キャッシュがあふれた時とプロセス移送時に、キャッシュ上の値をデータ本来の endian に変えて、データ領域へ書き戻す。

## 5 まとめ

異機種混成環境において、任意のプログラミング言語を利用可能なプロセス移送システムを提案した。全ての計算機上でメモリ上のデータの表現と配置を同一にすることで、非型安全言語におけるデータの再現性の問題を解決する。実計算機コードによる実行、低移送コスト、実行時 endian 変換のコスト削減により、プロセスの移送を含めた全処理の実行時間が最も短いシステムを目指す。

今後はシステムを実装し、従来のシステムとの実行時間、移送コストの比較や、実行時コスト削減のための3つの案の効果を計測する予定である。また計算機資源管理やセキュリティなど、システムの実用化を進めるつもりである。

## 参考文献


[1] 奥平雄吾ほか: 高速実行可能な低レベル命令セット仮想計算機の設計, 第 57 回情報処理学会全国大会論文集, 4H-09, 1998.

[2] A. Fugetta, *et al*: Understanding Code Mobility, IEEE Trans. Software Eng., Vol.24, No.5, pp. 342-361, May 1998.

[3] A. Black, *et al*: Distribution and Abstract Types in Emerald, IEEE Trans. Software Eng., Vol. SE-13, No. 1, pp. 65-76, January 1987.

[4] P. Smith and N. Hutchinson: Heterogeneous Process Migration: The Tui System, Tech. Rep. TR96-04, Dept. of Computer Science, Univ. of British Columbia, 1996.

[5] 首藤一幸, 村岡洋一: Java 上のスレッド移送システムと移動エージェントへの応用, JSPP'98 論文集, pp. 152, 1998.