異機種間プロセス移送のための再配置可能スタックフレームの構成

5R−4

長澤 育範　相田 仁　齊藤 忠夫

東京大学 工学部

1　はじめに

プロセス移送は、負荷分散によるスループットの向上、信頼性・可用性の向上などを目的として多く実現されてきたが、これらは同機種間のものであり、プロセス移送の有用性をさらに向上させるには、異機種間でもプロセス移送を行なうことができるのが望ましい．しかし、異機種間でプロセス移送を行なうには、アドレス空間の各領域の開始番地の相違やデータ表現の相違に対処してやらねばならない．

本稿では、ポインタのアドレス変換処理を行なって、関数フレームを再配置可能な形にすることで、筆者らがこれまでに提案してきた異機種間プロセス移送の方式をさらに拡張する方法について述べる．

2　現在までの方式の概要と問題

筆者らは現在まで、異機種間のプロセス移送の際に生じる問題に対して、C言語のコンパイラシステムの拡張やライブラリの変更により

- データ領域、ヒープ領域は全機種で各データが同一アドレスに割り付けられるようにする．
- スタック領域に関しては各データの関数フレーム内オフセットが全機種で同じになるようにする．
- 実行再開位置は、ソースプログラム中のラベル付けによって、全機種で認識可能にする．

といった処理を行なうことにより、データをある単位ごとにビットイメージで転送し、異機種間プロセス移送を行なう方法を提案・試作してきた[1][2]．

データ・ヒープ領域に関しては、全機種で同一アドレスに割り付けられているのでその領域を指すポインタは移送前後でそのまま利用可能である．しかし、スタック領域については移送時に各フレームが積み直されて、移送前後でローカル変数は異なるアドレスに割り付けられるため、そのアドレスを値として持つポインタをそのまま移送しても意味をなさない．

さらに一般的なプログラムに対応できるようにするためには、ローカル変数へのポインタ(以後、ローカルポインタと呼ぶ)は移送前後で変換を行なえるように、つまり各フレームを再配置可能にする必要がある．

3　ポインタ変数のアドレス変換

ポインタのアドレス変換のために、ポインタ変換表をグローバルポインタ変数のものはデータ領域に、ローカルポインタ変数のものは各関数のフレーム内に付加する．

ポインタ変換表の各エントリは、

- ポインタ変数の位置情報
  - グローバル変数は絶対アドレス
  - ローカル変数はフレーム内オフセット

  で表現することで、全機種で認識可能になる．
- ポインタ変数のタイプ
  どの領域を指すポインタであるかの種別．
- 指しているフレームのID
  これは、ローカルポインタの場合のみ利用される．
  指しているフレームを識別するのに用いられる．

からなる．

このポインタ変換表は、移送時に移送元で共通表現に変換し、移送先で逆変換する際に用いられる．その際、ローカルポインタ自体はフレーム内オフセットに変換されて転送される．通常の実行中には、各マシン上の表現になっているため、ポインタのアクセス速度は低下しない．

4　再配置可能フレームの構造と移送用処理

4.1　フレームのデータ構造

本方式で用いる各関数フレームの構造は図1のようである．

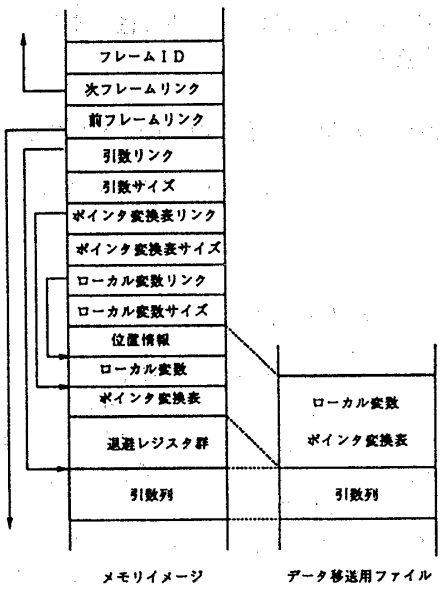

図1: 各関数のフレーム

呼び出し中の関数のフレームは、フレームIDによって一意に識別される．これは、ポインタのアドレス変換の際に使用される．

A relocatable stack frame for heterogeneous process migration
Ikunori Nagasawa, Hitoshi Aida, Tadao Saito
The University of Tokyo

このようなフレームに対して、関数呼び出し / 復帰時、移送時には以下に述べるような処理を行なう.

4.2 関数呼び出し / 復帰時の処理

関数呼び出し

関数呼び出しには、

- 通常呼び出し
- 移送時実行再開呼び出し

の2つのモードが存在する. 関数呼び出し時には、次の処理を行なう.

呼び出し側

1. 関数内位置情報 (ラベル番号) を退避する.
2. 引数サイズをグローバル変数にセットする.
3. 関数の呼び出し

呼び出された側

1. フレームリンク、変数サイズ、引数リンク、引数サイズ、トップフレームポインタ等の設定.
2. 移送時実行再開呼び出しの時は
   - その関数に相当するフレームをファイルから読み込む.
   - この関数が移送時にトップにあった関数であれば、ポインタ変換処理を行なう.
   - 関数内位置情報の値にしたがって、それに相当する位置にジャンプする.
3. 実行 (実行中には、ラベル位置で関数内位置情報を退避する.)

関数からの復帰時 (関数内の return を含む)

- フレームリンク、トップフレームポインタの再設定

を行なう.

4.3 移送時の処理

移送時には、まずポインタ変数の共通表現への変換を行なう. そしてデータ領域、ヒープ領域をファイルに書き出した後、スタックの底の側からフレームリンクにそって順番に各フレームのローカル変数、引数、ポインタ変換表をファイルに書き出す.

移送先の再開時には、移送時に呼び出し中であった各関数は、上で述べたようにスタックの底の側から順番に実際に呼び出される (移送時実行再開モード) ので、各関数の先頭で各リンクを設定し、ファイルからローカル変数、引数、ポインタ変換表の読み込みを行なう.

移送時にスタックのトップにあった関数が呼び出された時点で、ポインタ変数の逆変換処理が行なわれて、移送時実行再開モードは通常実行モードに移行する.

5 実装

以上述べた方式の実装は、、前処理系を拡張してソースコード変換を行ない、Cコンパイラ本体は各OS付随のものを用いる形で行なっている. 前処理系を用いているのは移植性を高めるためである.

現在のところ、Sun-3(Motorola MC68020)、Sun-4(Sun SPARC) の間で試作し、実際に稼働している.

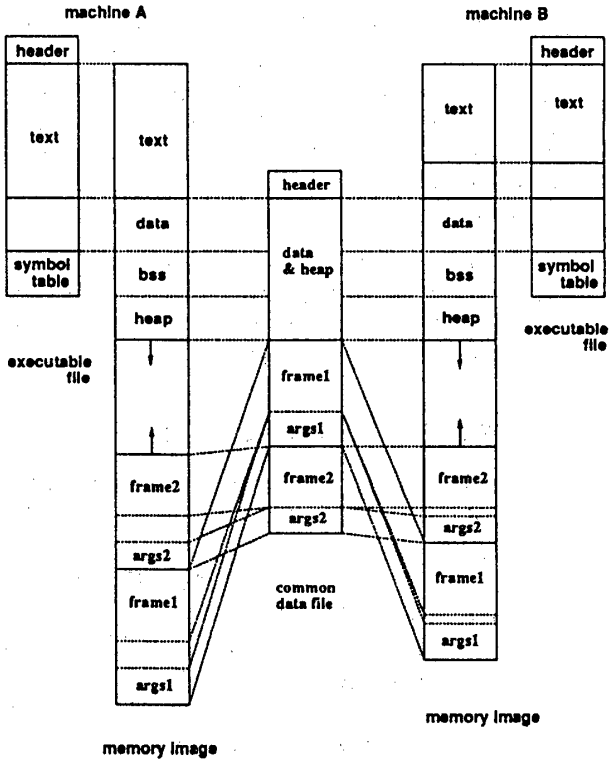

図 2: 本方式のアドレス空間のイメージ

6 まとめ

本稿では、ポインタのアドレス変換を加えて各フレームを再配置可能にすることによって、ローカル変数へのポインタを用いたプログラムを異機種間で移送する方法について述べた.

現在のところ、ポインタ変数のアドレス変換は、ヒープ領域中のポインタ変数には行なっていない. これを行なうには、ポインタ変換表をメモリ獲得時に動的に与える専用のライブラリ関数が必要があると思われる. これについては今後、検討して実装していく予定である.

参考文献

[1] 長澤、大胡、相田、齊藤：“異機種間プロセスマイグレーションの一手法,” 情報処理学会第 45 回全国大会予稿集 1P-3 (October, 1992).

[2] 長澤、相田、齊藤：“異機種間プロセス移送メカニズムの試作,” 情報処理学会研究報告, Vol.92, No.91, 92-DPS-58-22, pp.173-180 (November, 1992).