異機種間分散共有メモリシステムのためのソフトウェアサポート

1L−11

川合 史朗，相田 仁，齊藤 忠夫
東京大学 工学部

1 概要

疎結合マルチプロセッサシステムでは，プロセッサ間の通信はメッセージパッシングによるものが主流である。しかし，ソフトウェアまたはハードウェアのサポートにより疎結合環境で共有メモリシステムを実現することができれば，プログラミングの容易さ，密結合システムとのソフトウェアの互換性など，利点は多いと考えられる[3]。

そのような観点から，分散共有メモリシステムの研究が盛んになってきているが，そのほとんどは同一機種・OS上での実装か，共有データの形を限定しているのが現状である。

しかし汎用性を求めるならば，異機種間でビットイメージによりメモリを共有できることが望ましい。本研究では，言語仕様の拡張とコンパイラシステムの拡張によって汎用的に異機種間共有メモリシステムのプログラミングをサポートすることを試みている。

なお，既に異機種間での分散共有メモリシステムを実現している例はいくつか報告されているが[1, 2]，抽象化されたオブジェクトの共有という形をとっている。

2 解決すべき問題点

機種が異なると，endian, alignment, 浮動小数点数の表現などの，メモリ上でのデータ表現形式が異なる場合がある。共有メモリへのアクセスの際に，何らかの方法でデータ表現を変換しなければならない。

さらに，共有メモリがアドレス空間のどこにマップされるかが各CPUによって異なる可能性がある。その場合，ポインタ変数をそのまま格納することができない。

また，コンパイル，リンクをそれぞれのプロセッサで独立して行なうとすると，共有変数のロケーションをどうやって決めれば良いか，といった問題もある。

3 方針

共有メモリにアクセスするための専用のプリミティブを用いれば表現形式の変換の問題は解決されるが，共有メモリの利点を活かすためには，共有変数も通常の変数と同様に扱えることが望ましい。そこで，コンパイラシステムを拡張することにより，上記の問題点の解決を試みる。

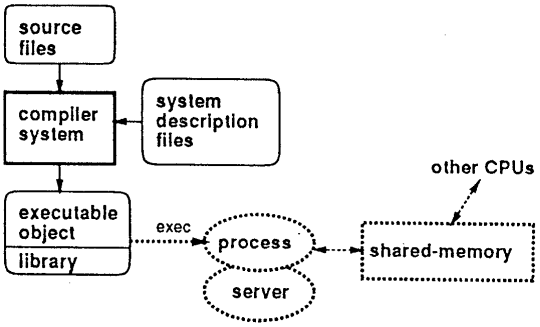

図 1: 本システムの処理の流れ

C言語に対し，共有変数を宣言できるように文法を拡張する。専用のコンパイラシステムを通すことにより，共有変数のアクセスの際の変換が正しく行なわれるようにする。コンパイラはコンパイルの際に，各CPUのデータ表現形式などを記述したシステム記述ファイルを読み込み，変換のためのコードや，共有メモリを管理するサーバと交信するためのコードを挿入する(図1)。

4 C言語の拡張

変数が共有メモリに格納されることを知らせるために記憶クラスを拡張し，ポインタの参照を正しく行なわせるために表現形式指定子を導入する。

記憶クラスの拡張

通常の記憶クラスに加えて，記憶クラス shared を導入する。shared と宣言された変数は共有メモリに置かれ，各プロセッサ上のプロセスから参照できる。

表現形式指定子

予約語 standard によって，オブジェクトの表現形式がプロセッサ間の共通形式であることを指定する。文法的には Type Qualifier とほぼ同じ扱いになる。

shared クラスの変数は全て standard 型であり，そうでない記憶クラスの変数は全て standard 型ではない。もちろんシステムによっては両形式が一致する場合もある。

単純な変数では宣言時の記憶クラスによって自動的に表現形式が決定されるので，この予約語は必要無いが，ポインタを用いる際にこれらの型の区別が重要になる。

"Software Support for Heterogeneous Distributed Shared Memory Systems"
Shiro Kawai, Hitoshi Aida, Tadao Saito
The University of Tokyo

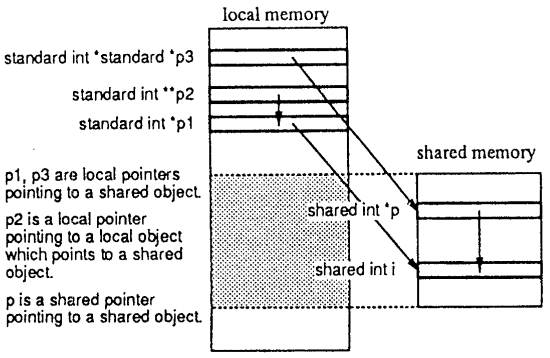

図 2: ポインタ

コンパイル時の式の評価中に表現形式の違いが検出された場合，型変換のためのコードが挿入される。基本的に，変数は評価される前に CPU のローカルな表現形式に変換されなければならない。ただしゼロとの比較など特殊な場合には可能な限り変換は抑制される。

ポインタ

共有メモリに置かれるポインタには，ローカル CPU のアドレスそのものではなく，共有メモリ先頭からのオフセットが格納される。ただし変換は自動的に行なわれるので，プログラマには通常のポインタと同じように見える。

ローカル変数領域に置かれるポインタ (ローカルポインタ) には，ローカル変数を指すものと共有変数を指すものの 2 種類があり，区別してやらねばならない (図 2)。

異なる表現形式を指すポインタ間の代入は，不正な参照を引き起こすので，原則として禁止される。

構造体・共用体

構造体のメモリ上での表現は，alignment の違いにより，ローカル領域と共有領域で異なる場合がある。表現形式の異なる構造体への代入は，メンバごとに変換・代入を行なうコードに展開される。

関数の引数

standard 表現形式とプロセッサ内部の表現形式が一致する場合は，明示的なキャストによって standard 型へのポインタを関数に渡すことができる。多くの CPU で文字列の表現方法は同じなので，strcat() などの文字列を扱う標準ライブラリに共有メモリ上の文字列へのポインタを渡すことが可能になる。

5 実装

システム記述ファイル

システム記述ファイルとしては，ローカルな表現形式を記述したファイルと，代表的な他の形式への変換を行なうコードを記述したファイルが CPU ごとに用意される。どの CPU を用いるかを指定することにより，適した standard 表現形式が選択される。standard 表現形式の endian 等を指定することもできる。

ソースファイルとは別にこれらのファイルを用意しておくことにより，同一のソースを異なる構成のシステムに利用することができる。

処理系の詳細

現在の実装では，コンパイラ本体は各 CPU の OS 附属のものを用い，前処理系とリンカによって拡張に対応している。

前処理系は，通常の前処理の後，もう一度ソースを構文解析し，共有変数の参照，代入の箇所に変換のためのコードを挿入する。コードは可能な限りインライン展開され，インラインアセンブルが可能ならばそれが用いられる。また，共有変数の宣言は拾い出されて共通のテーブルが作られる。

リンカは前処理系によって作成されたテーブルをもとに，共有変数の配置を決定する。

OS によっては共有メモリのマップアドレスが実行時まで分からない場合がある。そのようなシステムでは，前処理によって共有メモリの参照がポインタ参照に置き換えられ，実行時のスタートアップルーチンによってポインタが初期化される。

6 まとめ

このシステムは，抽象化されたデータ表現を扱うような言語に比べると，かなりハードウェアに近いレベルでメモリを扱っている。C の拡張ということで，既存のソフトウェアの移植は容易であろうが，高度な並列処理を書くのには必ずしも適してはいない。むしろ，より抽象化された並列処理言語や分散システムを記述するために用いるのがよいと考えている。

今後は，実際に分散アプリケーションを開発しての性能評価や，他言語 (C++ 等) への拡張を予定している。

参考文献

[1] S. Ahuja, N. Carriero, and D. Gelernter. Linda and friends. *IEEE Computer*, 19(8):26–34, Aug. 1986.

[2] R. Bisiani and A. Forin. Multilanguage parallel programming of heterogeneous machines. *IEEE Trans. Comput.*, 37(8):930–945, Aug. 1988.

[3] B. Nitzverg and V. Lo. Distributed shared memory: A survey of issues and algorithms. *IEEE Computer*, 24(8):52–60, Aug. 1991.