異機種間分散共有メモリ用コンパイラシステムの性能評価

8L−1 川合 史朗, 相田 仁, 齊藤 忠夫
東京大学 工学部

1 概要

分散共有メモリ (Distributed Shared Memory: DSM) は，疎結合マルチプロセッサシステムにおいて各プロセッサにメモリを共有しているように見せる技術である。ページングを疎結合プロセッサ間に拡張する方式や，ハードウェア的にメモリ内容を複製する方式 (複製型 DSM) 等が提案されている。

機種間で互換性のないデータ表現形式同士の適切な変換を行えるならば，DSM を異機種間に拡張することができ，応用範囲が広がる。本稿では，変数の型の概念を拡張することにより複数のデータ表現形式を扱えるようにした異機種間 DSM 用の C コンパイラと，その動作実験について述べる。

2 異機種間 DSM 用 C コンパイラ

データ表現形式の非互換性

異機種間では，複数バイトのデータの配置順 (endian)，データの整列境界 (alignment)，文字コード体系，浮動小数点形式等に互換性がない場合がある。

共有メモリ上での表現形式は機種間で統一しておき，共有変数のアクセスの際に変換を行なうことが必要である。プロセッサによっては表現形式変換のためのインストラクションを備えているものがあり，それを直接利用できれば変換のオーバヘッドは低く抑えられる。

C 言語仕様の拡張

上記の問題を次のように抽象化する。各プロセッサは，自分にとって自然な表現と共通表現の二つのデータ表現形式を扱うことになる。そこでデータ表現形式を型属性の一種と考え，前者を local 型，共通表現を standard 型と呼ぶことにする。

通常の変数宣言では，変数はローカル型であるとみなされる。それに対し，表現形式指定子 standard を用いて，

```
standard int SharedVar;
```

のように宣言された変数は，“standard int” 型を持つ。standard 型の変数の値に対しては，全く同じ型の変数への代入と同じ型属性のローカル型への変換という二つの操作のみが許される。演算式中に standard 型変数を用いた場合はコンパイラによって暗黙のうちにローカル型へ変換される。

共有変数の宣言は記憶クラスに shared を指定することで行なう。共有変数はすべて standard 型である。表現形式と記憶クラスとを分けたのは，ローカル型でありながら standard 型の変数を指すポインタが存在するからで，

```
shared int *sptr; /* 共有メモリにあり，共有メモリ
上の integer (standard int 型) を指すポインタ */

standard int *lsptr; /* ローカルメモリにあり，
standard int 型を指すポインタ */

int *lptr; /* ローカルメモリにあり，ローカル型 を
指すポインタ */
```

のように使い分ける。

コンパイル時に standard 表現形式のフォーマットや変換インストラクションを記述したシステム記述ファイルを指定することにより，特定の standard 型をもつシステム用のオブジェクトが生成される。変換のコードはすべてインライン展開される。

コンパイラの仕様・動作についての詳細は文献[1]を参照されたい。なお，通常のコンパイラと区別してこの拡張コンパイラを xcc と呼んでいる。

3 異機種間 DSM の動作実験

実験システムの構成

複製型 DSM には，SCRAMNet という製品を用いた。伝送に光ファイバを用いており，実効データ転送速度は最大 7.2MByte/sec である。

プロセッサには，2 台の WS(SPARC station1+ 及び 2) と，1 台の PC(Canon AXi/V 486) を SCRAMNet で接続した (図 1)。sparc は big endian であり，i486 は little endian である。i486 には 1 サイクルで endian 変換を行なうインストラクションが用意されているが，sparc では専用命令はなく，long word の endian 変換にはアセンブラで 15 ステップ程を要する。OS はすべて Unix 互換 OS を用いた。

実験用アプリケーションとしては，並列ソートアルゴリズムを用い，ファイルに格納された long integer のデータを読み込んでソートするプログラムを使用した。負荷はプロセッサ間で静的・均等に分割するようにした。

並列化にあたって必要なソースの変更はほとんど密結合共有メモリマルチプロセッサの場合と同じであった。

“ Performance Evaluation of Heterogeneous Distributed Shared Memory and Its Compiler System”
Shiro Kawai, Hitoshi Aida and Tadao Saito
Faculty of Engineering, The University of Tokyo

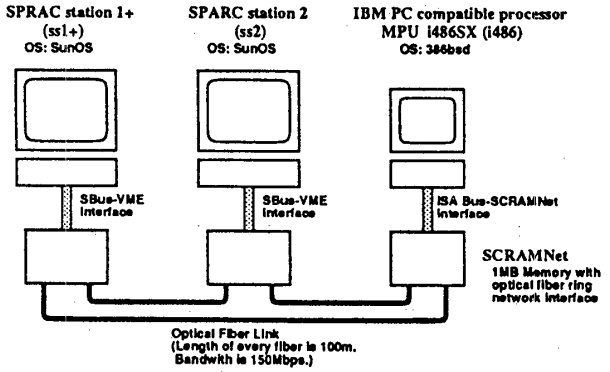

図 1: 実験システムの構成

```
shared long data[];
...
  if (data[i] > data[j])
    {
      standard long temp;
      temp = data[i];
      data[i] = data[j];
      data[j] = temp;
    }
```

Code for xcc

```
/* 実行時に共有メモリ上の
   データ配列を指すように
   初期化する                */
extern long *data;
...
  if (convert(data[i])
      > convert(data[j]))
    {
      int temp = data[i];
      data[i] = data[j];
      data[j] = temp;
    }
```

Code for ordinary cc

図 2: xcc 用のソースと通常のコンパイラ用のソース

比較のために，xcc を用いずに明示的に変換関数を呼び出し，通常の cc でコンパイルしたプログラムも用意した。xcc を用いない場合はデータの変換だけでなく，共有変数をポインタとして宣言しておき実行開始時に初期化するコードなども明示的に書く必要がある。なお，変換関数は C で書いた。両者のコードの一部を図 2に示す。

データ表現の変換のオーバヘッド

まず，データ表現形式の変換の実行速度への影響を見るために，単一ノードでの実行速度の測定を行なった。standard 表現形式がローカルと同じ場合を基準として，xcc による endian 変換のインストラクションが挿入された版と，明示的に変換関数を呼び出している版の実行速度を図 3に示す。

i486 プロセッサでは，xcc を用いた場合，変換命令がインラインで展開されるため，変換の影響は 2% 程に抑えられている。sparc プロセッサでは変換時の負担が大きく，11 ～ 16% の効率低下が起きているが，共有メモリのアクセスがかなり多いアプリケーションであることを考えると，通常のアプリケーションでの変換の影響はこの程度かより小さいことが予想される。

明示的に関数呼び出しを用いる版では，さらに効率が低下している。もちろん，変換関数をアセンブラで書いたり，関数のインライン展開ができるコンパイラを用いれば xcc の場合と同じ効果が得られることは予想される。

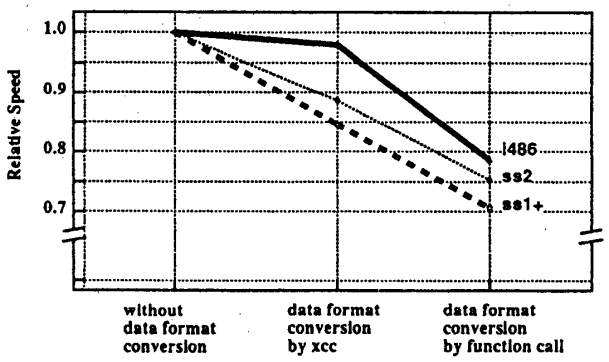

図 3: データ表現変換の実行速度への影響

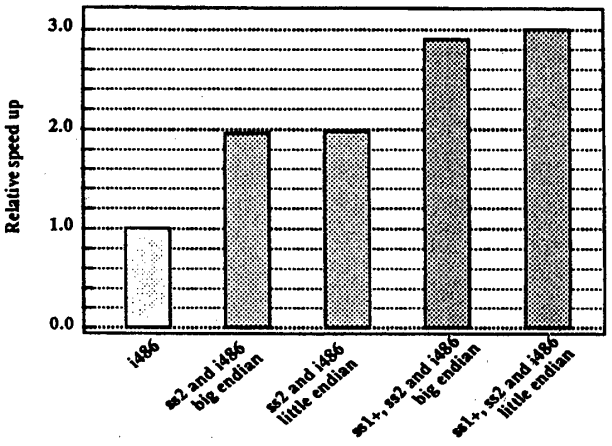

図 4: 異機種間並列処理による速度向上

並列処理による速度向上

i486 単体での実行速度を基準にした場合の 2 プロセッサ，3 プロセッサでの実行速度を図 4に示す。standard 型を little endian(i486 側) に設定した方が速くなっており，前節の結果と矛盾するようだが，これは i486 と sparc との実行速度差によるためと考えられる。単体性能で sparc の方が 2 倍以上速いため，均等に負荷を分割した今回のアプリケーションでは，sparc 側で表現形式変換のオーバヘッドを負担しても i486 の方が遅く，結果的に表現形式変換の影響が現れなかったのである。

4 まとめ

コンパイラでデータ表現の変換をサポートすることにより，異機種性をあまり意識することなくプログラムが書け，また専用インストラクションを用いれば実行時の効率低下も低く抑えられる。

参考文献

[1] 川合 史朗，相田 仁，齊藤 忠夫：異機種間分散共有メモリのためのコンパイラシステム．情処研報 92-DPS-58, Dec. 1992.