並列化コンパイラ TINPAR における専用通信ライブラリの実装と性能評価

前山浩二, 後藤慎也, 三吉郁夫, 森眞一郎, 中島浩, 富田眞治

京都大学工学部

1　はじめに

我々はメッセージ交換型並列計算機のための並列化コンパイラ TINPAR を開発している [1]。TINPAR は owner computes rule に従って、拡張 Tiny language で書かれた逐次プログラムから C 言語による SPMD プログラムを生成する。

メッセージ交換型並列計算機のプログラムは非所有データの参照のためにメッセージ通信を行う。しかし、通信処理の起動コストが一般に大きいため、その回数を減らす工夫が必要である。そこで TINPAR では、動的ベクトル化機能を持つ専用通信ライブラリを用いて、これに対処している。

本稿では、TINPAR の専用通信ライブラリに対する要件、実装法を述べ、さらに AP1000 上で簡単な数値演算プログラムを用いて評価を行った。

2　専用通信ライブラリによる動的ベクトル化

メッセージ通信は起動コストが一般に大きい。この起動コストによるオーバーヘッドを軽減する最適化手法の一つにメッセージのベクトル化がある。これは送信データをまとめて一つの長いメッセージとして送信する手法である [2]。

メッセージのベクトル化を行う場合、データが連続領域にある、あるいは一定距離毎にデータがあるならば、専用ハードウェアによるデータ領域からの直接通信が可能になり、高速な処理が可能である。しかし、データの配置がこの条件を満たさない場合、送信データをベクトル化するには、バッファにパッキングする処理ならびに、その適切なバッファの管理を必要とする。

TINPAR では専用通信ライブラリを用い、これにより提供される通信ランタイムシステムによって動的ベクトル化を行う。すなわち、TINPAR のオブジェクトコード中の (SEND,BROADCAST)/RECV 文がランタイムシステムを起動し、パッキング/アンパッキングを行うと共に、適切な長さのメッセージの送受信を行う。

送信のタイミングはバッファに空エントリがなくなった場合である。したがって必ずしも最適なタイミングで送出が行われるとは限らない。そこで、最適化の手段として送信バッファの強制フラッシュ機能を用意する。

TINPAR は専用通信ライブラリを用いることで専用ハードウェアによるベクトル化ができないデータに対して効率の良い通信を行うことができる。またこの際、後者のデータに対してコンパイル時にはベクトル化を意識する必要がない。

送信 ··· SEND_INTEGER(26) , SEND_REAL(28)
受信 ··· RECV_INTEGER(21) , RECV_REAL(23)
放送 ··· BROADCAST_INTEGER(18) ,
BROADCAST_REAL(20)
メッセージ・フラッシュ ···
MSG_BUFFER_FLUSH ,
ALL_MSG_BUFFER_FLUSH
バリア同期 ··· SYNC

() の中の数字はパッキング/アンパッキングのための命令数

表 1: 専用通信ライブラリのインターフェース

```
if (BC_buff != empty) {
  flush_BC_buff
}
data_packing
if (send_buff == full) {
  send_message
}
```

```
if (recv_buff == empty) {
  if (message == not_ready) {
    flush_send_buff
  }
  recv_message
}
data_unpacking
```

図 1: 通信ライブラリの概要

3　実装

TINPAR の専用通信ライブラリは現在表 1の機能を提供している。本章では AP1000 での実装について述べる。

3.1　バッファの管理

AP1000 は標準通信ライブラリにより連続領域のデータを送受信することができる。このため、AP1000 に対する TINPAR の専用通信ライブラリの実装は、図 1に示すように AP1000 の通信ライブラリにパッキング/アンパッキング処理とバッファ管理を行うルーチンを付加する形で実現できる。

表 1に示すように専用通信ライブラリには一対一通信と放送の二種類の送信機能が用意されている。各々のプロセッサは一本の放送用バッファと相手のプロセッサの数だけの送受信バッファを持つ。

送信、放送のどちらで通信が行われたかに関わらず、メッセージは同一の受信バッファに格納される。(図 2参照) そのため、送信/放送を併用する場合にはデータの受信順序を保証する必要がある。どちらかのバッファを常に空になるように管理すれば、受信順序を保証できる。

一方、受信側ではデッドロックを回避するために、データ未着による受信待ち状態に入る前に、全ての送信待ちデータを送出する。

3.2　パッキングとデータ型

TINPAR は数値処理プログラムをターゲットとしており、現在扱えるデータ型は整数型と実数型である。この

Implementation and Performance Evaluation for Communication Libraries for TINPAR
Koji Maeyama, Shin-ya Goto, Ikuo Miyoshi, Shin-ichiro Mori, Hiroshi Nakashima, Shinji Tomita
: Faculty of Engineering, Kyoto University

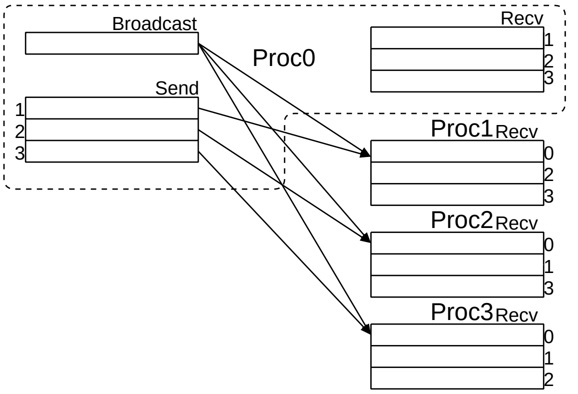

図 2: 送受信処理

ため、データ長の異なる二種類のデータをパッキングする必要がある。パッキングに際し、それらのデータを共用体を用いて固定長として扱う。これにより整数型データを送信する場合、送受信バッファの利用効率は落ちるが、高速処理が可能となる。

4 性能評価

AP1000 を用いて専用通信ライブラリの評価を行う。評価には行列積、ガウス消去法、SOR 法の三つのプログラムを TINPAR により 64 プロセッサ用に並列化したものを用いた。これらのソースプログラムは文献 [1] による。

4.1 動的ベクトル化の効果

表 1の括弧の中の数字は送信、受信、放送のパッキング/アンパッキング処理に必要な命令数である。これらの処理時間は最大でも 1ms 以下で、最低でも数十 ms かかる送受信関数の実行時間に比べ十分に短いため、数個のデータをベクトル化しただけでも十分に効果が現れる。SOR 法を例にとるとパッキングしない場合に対して、バッファサイズ (ベクトル長)2 の場合 1%、3 の場合 10%の速度向上が見られた。

4.2 専用通信ライブラリの実行時間への影響

各プログラムの実行時間を、AP1000 の通信ライブラリ、通信ランタイムシステム、通信以外の演算時間に分類し、図 3に示す。それぞれのプログラムに対し、グラフの右側は専用通信ライブラリを用いたものであり、左側はプログラムの静的解析に基づいた結果、明示的にパッキングを行い適切なタイミングで通信を行った場合に相当する。ここで、それぞれの時間は理想的な実行時間、すなわち (逐次版の実行時間)/64 を 1 として正規化したものである。

まず、専用通信ライブラリを用いた場合、それぞれの加速率は行列積 47.7 倍、ガウス消去法 39.8 倍、SOR 法 29.3 倍となり、良好な結果が得られた。それに対し、パッキングされるデータが連続領域に存在する時、AP1000 の送信関数で一括して送信するようにプログラムを変更したところ、その結果は行列積 49,8 倍、ガウス消去法 49.7 倍、SOR 法 33.3 倍となった。

最適化や実行時のタイミングの若干の違いにより少しの誤差があるが、動的ベクトル化を行った場合の通信以外の演算時間や AP1000 の通信ライブラリの実行時間は静的解析を行った場合のそれとほぼ等しい。実行時間の差はランタイムシステムの実行によるオーバーヘッドによるものであると分かる。

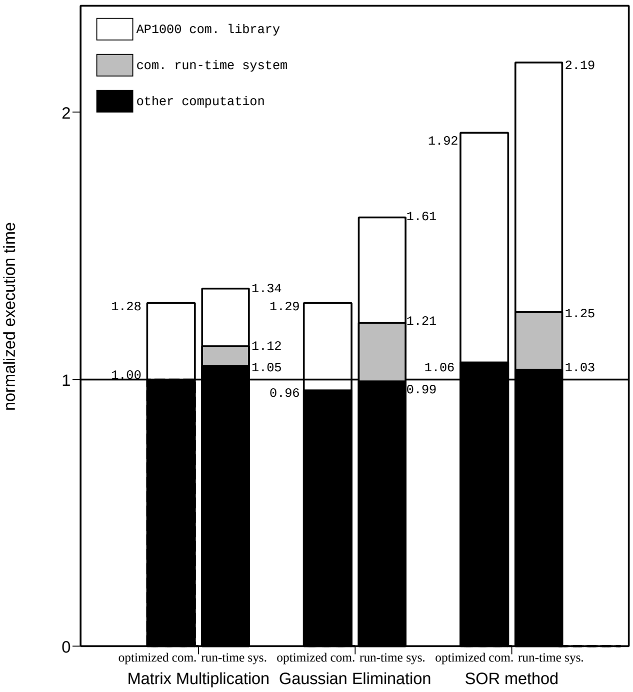

図 3: オブジェクトコードの実行時間の内訳

しかしながら、そのオーバーヘッドは静的解析を行った場合の通信時間の 20% ~ 30%増し程度であり、ベクトル化を行わなかった場合の通信時間全体の増加 (数倍 ~ 数 100 倍) に比べて十分に少ない。

5 おわりに

本稿では、現在我々が開発中であるメッセージ交換型並列計算機のための並列化コンパイラ TINPAR の専用通信ライブラリについて述べた。動的ベクトル化機能を持つ専用通信ライブラリを用いることにより、効率の良い通信処理可能であることを示した。

TINPAR の専用通信ライブラリが提供する動的ベクトル化機能は静的解析だけでは十分にベクトル化できない場合等に、その効果を発揮できる。

今後はこのようなベクトル化と、コンパイラの静的解析とハードウェアの専用通信機能を利用したベクトル化との併用について検討を行う予定である。

参考文献

[1] 三吉, 前山, 後藤, 森, 中島, 富田: メッセージ交換型並列計算機のための並列化コンパイラ TINPAR — 最適化手法と性能評価 —, 情処研報 94-HPC-54, pp.45-52, 1994

[2] Reinhard von Hanxleden, Ken Kennedy:GIVE-N-TAKE—A Balanced Code Placement Framework, ACM SIGPLAN '94 Conf. on PLDI, pp.107-120, June,1994