ハイパフォーマンス
コンピューティング 57−19
（1995．8．25）

HPFにおける実行時の通信解析オーバヘッドの削減手法

小笠原 武史、石崎一明、小松 秀昭

日本アイ・ビー・エム（株）東京基礎研究所

配列分割を指定するデータ並列言語では、各ノードプロセッサが必要とする配列範囲を求める通信解析の必要があるが、解析に必要なパラメータが変数で実行時まで求められない場合が多い。この場合多様な分割と配列アクセスを考慮すると、ライブラリで実行時解析する手法が有効である。しかしながら各ループごとの通信解析を行なうライブラリの実行頻度は多く、ライブラリの実行時間はアプリケーションの実行時間に大きな影響を与える。本稿では、ライブラリの解析結果の再利用により、ライブラリのオーバヘッドを削減する手法を説明する。

A mothod for reducing the overhead of dynamic communication analysis

Takeshi Ogasawara, Kazuaki Ishizaki and Hideaki Komatsu

Tokyo Research Laboratory, IBM Japan,Ltd.

Dynamic communication analysis is needed when data-parallel compilers cannot resolve array regions that are accessed but are on remote processors at compile-time. Considering various combinations of data alignments and distributions, it is effective to use run-time library routines for dynamic communication analysis. However, since they are frequently called for each loop nest, their execution time has a serious impact on the performance of application programs. In this paper, we present a method to reduce the run-time overhead of dynamic communication analysis by reusing results of the analysis.

1 研究背景

一般にデータ並列言語用のコンパイラ [1, 2] は、ループ実行に必要なデータをループ実行に先だって各プロセッサがベクタメッセージ通信によりプリフェッチし、通信や同期なしにループを並列実行させることが重要である [3]。ベクタメッセージ通信を行なうには、送信プロセッサ、受信プロセッサ、通信する配列領域の3つ組の集合（通信セットと呼ぶ）を求める通信解析を行なうことが必要である [4]。

一般に通信解析はソースコード上に登場する、

- ループ
- ループ内の文
- 文中の右辺配列オペランド

の組合せ毎に必要である。以下ループは常に正規化されている、すなわちループが1から上限まで1とびで実行されるものとして考えると、1回の通信解析に必要なパラメータ（以下、通信解析パラメータ）は、

- ループインデックスの上限
- 左辺配列の各次元を分割するプロセッサ数、サイズ、分割方法、添字式係数
- 右辺配列の各次元を分割するプロセッサ数、サイズ、分割方法、添字式係数

である。例えば代表的なデータ並列言語である High Performance Fortran (HPF)[2] では `BLOCK(n)`, `CYCLIC(n)` を分割方法として指定でき、配列のある次元を n 個の配列要素毎にそれぞれ block, cyclic でプロセッサ配分する。また添字式係数とは、配列の添字式中の注目しているループのループインデックス変数以外をいう。

通信解析は通信する配列の各次元毎に行なう。図1は通信解析の概略である。並列化されたループ（$LoopNest_i$）対して、"owner computes rule" に基づいてプロセッサ配分されたインデックス空間（$LocalIterationSet_i$）が求められる [10]。$LoopNest_i$ のループ本体のある文（$Statement_j$）の右辺配列オペランド（$RhsArray_k$）について、ある次元の添字式（$SubscriptExpression_l$）と $LocalIterationSet_i$から、各プロセッサがアクセスする領域（$LocalReadSet_l$）が求められる。$LocalReadSet_l$とその次元の分割方法（$ArrayDecomposition_l$）から、各プロセッサが通信する領域（$CommunicationSet_l$）が求まる。通信解析パラメータは、$LocalIterationSet_i$、$SubscriptExpression_l$、$ArrayDecomposition_l$を 決定するパラメータである。

コンパイル時にこれらすべての通信解析パラメータが定数値として与えられる場合、コンパイラは通信解析できる。しかし1つでも変数であると、コンパイル時に通信解析を終了できない [5]。そして多くの場合プロセッサ数やループ上限等、一部の通信解析パラメータは変数である。

コンパイル時に定数値でない通信解析パラメータを使用する計算は、実行時に通信解析を行なわなければならない [6]。例えば HPF において `INHERIT` 記述子により、配列を分割するプロセッサ数や分割方法がすべて実行時に初めて与えられるような場合等である。

実行時通信解析を行なうには、通信解析コードをインライン生成する方法と、ライブラリルーチンを呼ぶ方法が考えられる。通信解析の複雑さ [7, 8] とそれを行なうルーチンのコードサイズを考慮すると、ライブラリによって実現する手法の方が有効であると考えられる。

複雑な通信解析を行なうライブラリの実行時間は、超並列環境においてループ実行時間が短縮化されるに従い、オーバヘッドとしてアプリケーションの性能に大きく影響する。したがってライブラリによって消費される時間の削減が重要な問題となる。

一般に通信解析パラメータに変化がなければ、実行時に一度計算された通信セットは再利用できる [9]。また、たとえ通信解析パラメータが変更されても、変更された通信解析パラメータを使用する通信解析だけを行なって、その他の通信解析に必要な計算のほとんどを省略することができる場合もある。

これら再利用による最適化手法（以下**通信セット再利用**と呼ぶ）により実行時に行なわなければならない通信解析の計算量を削減し、前回と同じ無駄な計算を行なわないことによって実行時通信解析オーバーヘッドを削減し、コンパイル時に通信解析パラメータがすべて定数値で与えられている場合に比べ、すべてが変数であっても遜色のない性能が得られることが確認できた。

今回、我々が開発した HPF コンパイラに通信セット再利用技術を実装してその有効性の評価を行なった。以下、2章で通信セット再利用を実現するために必要なコンパイラとライブラリの技法を説明する。3 章でベンチマークプログラムにより性能改善を評価する。

```
foreach Statement_j in LoopNest_i
  foreach RhsArray_k in Statement_j
    foreach SubscriptExpression_l in RhsArray_k
      LocalReadSet_i = computeLocalReadSet (SubscriptExpression_l, LocalIterationSet_i)
      CommunicationSet_i = computeCommunicationSet (LocalReadSet_i, ArrayDecomposition_k)
    end
  end
end
```

図 1: 通信セットの計算

2 通信セット再利用アルゴリズム

実行時通信解析は、ループ中の文に登場する各右辺配列オペランドを対象として行なわれる。通信セット再利用は次のように、まずコンパイラで処理され、実行時にライブラリで行なわる。

1. コンパイラでの作業
   (a) 通信解析パラメータの分類と再利用性判断
   (b) 実行時解決のためのコード生成
2. ライブラリでの作業
   (a) 再利用可否チェック
   (b) 通信セット再計算

以下、各作業を説明する。

2.1 コンパイラでの作業

2.1.1 通信解析パラメータ分類と再利用性判断

ソースコード上のある通信解析対象となるループ、文、右辺オペランドの3つ組に対して、コンパイラは通信解析パラメータのそれぞれについて、

1. 定数値である
2. 注目しているループのループ不変変数である

のどちらかの条件を満たすかを調べる。すべての通信解析パラメータがどちらかの条件を満たすならば、ベクタメッセージ通信可能と認識し [10]、この通信解析対象セットに対して計算された通信セットは再利用可能であると判断する。

2.1.2 実行時解決コード生成

実行時にライブラリがコンパイル時と同等の通信解析が必要になる場合に備えて、コンパイラは通信解析パラメータ等実行時通信解析に必要な情報を、解析の単位毎にメモリブロック（以下パラメータブロック）に展開するコードを生成する。通信セットは通信する配列の各次元毎に計算されるため、パラメータブロック内では次元毎の計算に必要な通信解析パラメータがグループ化されている。

パラメータブロックは次の内容から構成される。

- CommSetP
- PrevParamVij
- CurrParamVij
- ReuseCommSet$i$

CommSetP は通信セットへのポインタである。PrevParamVijは通信する配列の第i次元に関する通信セットを計算するための通信解析パラメータのj番目の通信解析パラメータの前回の値、CurrParamVijは同じく現在の値をそれぞれ保持する。ReuseCommSet$i$は通信セットのi次元目の再利用可否を示す。

パラメータブロックは実行可能モジュールのロード時に部分的に初期化される領域である。通信解析パラメータのうち定数値のものは、この初期化の際にその値がパラメータブロックに格納される。変数のものは場所だけが確保される。CommSetP はコンパイル時に計算できた通信セットを保持するか、あるいはコンパイル時に全く計算できない場合は 0 に初期化されて最初の一度だけ通信解析が必要なことを示す。ReuseCommSet$i$ についても同様である。

コンパイラは初期化されない場所を実行時の通信解析パラメータの値で埋めるコードを生成する。したがって通信セットの第i次元の実行時計算は、コンパイル時にそれらが定数値であった場合の計算と同じ計算が行なわれる。

2.2 ライブラリでの作業

2.2.1 再利用可否チェック

ライブラリは通信セットの各次元iについて、通信セットがすでに計算されていればそれが再利用可能かどうかをチェックする。

ReuseCommSet$i$が0すなわち一度も通信解析されていない場合、第i次元について通信解析を行なう（2.2.2）。すべてjについてPrevParamVijをCurrParamVijにコピーする。ReuseCommSet$i$を1にする。アルゴリズムはここで終了する。

ReuseCommSet$i$が1である場合、前回計算した通信セットがある。ライブラリはすべてのjについてPrevParamVijと CurrParamVijの同一性をチェックする。前回と異なる値をとる通信解析パラメータが1つでもあれば通信セットの再利用は不可能と判断し、通信解析を行なう（2.2.2）。この場合、すべてのPrevParamVijをCurrParamVijにコピーする。最終的にすべて同一であれば再利用可能と判断する。

2.2.2 通信セット再計算

実行時通信解析が必要であると判断された場合、図1で示される計算をコンパイル時と同様に行なって通信セットの各次元を計算し、結果を次回の再利用可否チェックのために保存する。

```
chpf$ template cols(idim,jdim)
chpf$ align with cols::c1,c2,c3,c4,c5,c6,c7,c8,c9,c10
chpf$ align with cols::u1,u2.
chpf$ distribute cols(*, block)
      tm1=second()
      do n=1,nt,2
       if(n .lt. npsor) u2(is,js)=u2(is,js)+fai(n)
       do j=3,jmax-2
       do i=3,imax-2
       u1(i,j)=c1(i,j)*u2(i-2,j)+c2(i,j)*u2(i-1,j)+
     1         c3(i,j)*u2(i+1,j)+c4(i,j)*u2(i+2,j)+
     2         c5(i,j)*u2(i,j-2)+c6(i,j)*u2(i,j-1)+
     3         c7(i,j)*u2(i,j+1)+c8(i,j)*u2(i,j+2)+
     4         c9(i,j)*u2(i,j)-c10(i,j)*u1(i,j)
       enddo
       enddo
       if (n    .lt. npsor) u2(is,js)=u2(is,js)-fai(n)
       if (n+1 .lt. npsor) u1(is,js)=u1(is,js)+fai(n+1)
       do j=3,jmax-2
       do i=3,imax-2
       u2(i,j)=c1(i,j)*u1(i-2,j)+c2(i,j)*u1(i-1,j)+
     1         c3(i,j)*u1(i+1,j)+c4(i,j)*u1(i+2,j)+
     2         c5(i,j)*u1(i,j-2)+c6(i,j)*u1(i,j-1)+
     3         c7(i,j)*u1(i,j+1)+c8(i,j)*u1(i,j+2)+
     4         c9(i,j)*u1(i,j)-c10(i,j)*u2(i,j)
       enddo
       enddo
       if (n+1 .lt. npsor) u1(is,js)=u1(is,js)-fai(n+1)
      enddo
      tm2=second()
```

図 2: X42 の核ループ

3 評価

3.1 評価環境

我々は、RISC システム/6000 (RS/6000) クラスタ、RS/6000 SP を対象とした、AIX XL Fortran コンパイラ/6000 をベースに HPF コンパイラを研究開発している [11, 12, 13, 10, 14, 15, 16, 17, 18]。現在までに Subset HPF[2] 準拠コンパイラ（以下 TRL-HPF コンパイラ）を開発した。TRL-HPF コンパイラは実行モジュールとして、AIX Parallel Environment (PE) Version 1.2.1 下で動作する SPMD コードを生成する。

今回、通信セット再利用技術を TRL-HPF コンパイラに実装し、その効果を評価した。TRL-HPF コンパイラの実行時通信解析ルーチンはコンパイラと同じものを使用している。コンパイラは SPMD コードを生成してすべてのプロセッサで使用されるコードを出すため、コンパイル時に生成されるおよび実行時に再計算される通信セットはすべてのプロセッサについての通信情報を表現する。通信最適化 [3] が実行時に行なわれる場合、最適化結果が同様に再利用される。

今回は POWER2 用のコードを生成し、RS/6000 SP（POWER2-66MHz シン・ノード 32 ノード）上で評価を行なった。

3.2 実験結果

3.2.1 通信セット再利用の評価

今回、Applied Parallel Research, Inc (APR) が公表しているベンチマークプログラム（以下 APR-Bench）[1] の X42 を使用した。このプログラムは核ループについての計算時間を計っている。図 2は核ループのソースコードである（$imax, jmax$ 共に 512）。6行目のループによって8行目と19行目の2つの2重ループが繰り返し実行される。TRL-HPF コンパイラは8行目の2重ループと19行目の2重ループを並列化し、ベクタメッセージ通信のプリフェッチコードを各ループ直前に生成する。プロセッサ数がコンパイル時に未定であるため、実行時通信解析が必要である。

実行時通信解析オーバヘッドはプロセッサ数に関係なくほぼ一定であり、プロセッサ数が多くなって計算実行時間が減るほどオーバーヘッドの占める割合が高まる（図 3）。

通信セット再利用がある場合とない場合について核

[1] URL=ftp://ftp.infomall.org/tenants/apri/Bench/Benchmarks.tar.Z

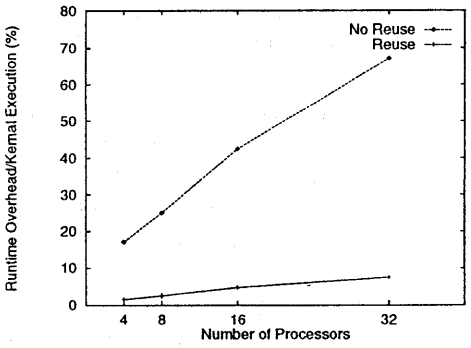

図 3: 核ループにおける通信解析オーバヘッド比率

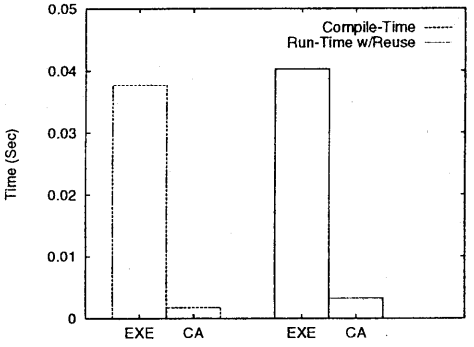

図 5: コンパイル時通信解析との比較

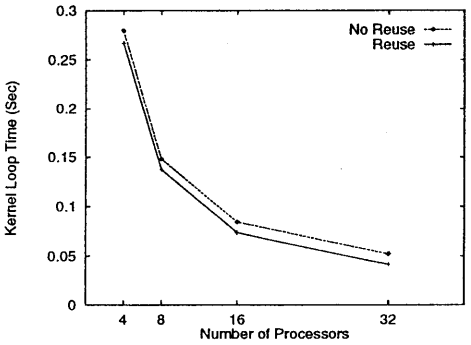

図 4: 核ループにおける再利用効果

ループの時間を比較した（図 4)。実行時通信解析のオーバヘッドはプロセッサ数に依存せず固定のため、プロセッサ数が多いほど通信セット再利用の効果が高いことが現れている。APR-Bench では X42 以外にも Tomcatv, Shallow が同様に高い効果を示す。

3.2.2 静的通信解析との比較

3.2.1で使用した X42 プログラムで実行時通信解析（通信セット再利用あり）を行なう場合と、同じプログラムを基に静的に通信解析パラメータを与えてコンパイル時に通信解析が終了するように変更した場合について、32 台のプロセッサを使用して核ループの実行時間を比較した（図 5)。

図 5には核ループ実行時間（EXE）とその一部である通信解析ルーチン実行時間（CA）が示されている。TRL-HPF では通信セットはすべてのプロセッサに対する表現となっており、実行時に各プロセッサは自プロセッサに関連する情報を抽出する必要があるため、コンパイル時に通信解析が終了する場合でも CA が若干存在する。

通信セット再利用によって実行時通信解析のオーバーヘッドを削減した結果、32 台のプロセッサでは通信解析パラメータを静的に与えた場合に比べて、核ループ実行時間を約 7%増にとどめることができた。

HPF の対象となるような実アプリケーションを考えた場合、配列サイズおよび繰り返し回数がベンチマークプログラムに比べ大きいため、核ループの実行時間はより長くなる。したがって通信セット再利用によって実行時通信解析オーバヘッドが削減された場合、オーバヘッドがループ実行時間に占める比率は非常に小さくなる。

4 まとめ

本稿では、通信解析パラメータがコンパイル時にループ不変な変数であることが原因で実行時通信解析が必要な場合について、コンパイラとライブラリ両方による最適化である、実行時通信解析の結果を再利用する最適化手法について説明した。同じループ実行の繰り返し回数が多く、しかも通信セットが毎回再利用される場合、通信解析ルーチンが消費する時間は 1 回の実行時間だけとなって実行時解析オーバヘッドは激減し、コンパイル時に通信解析パラメータがすべて定数値で与えられる場合に比べても遜色がなくなる。

このことを TRL-HPF コンパイラと RISC システム/6000 SP 上で、APR のベンチマークプログラムを使用して確認した。

謝辞

本研究にあたり、日本IBM東京基礎研究所の、中谷登志男氏、郷田修氏、大澤暁氏、菅沼俊男氏に感謝致します。

参考文献

[1] Philip J.Hatcher and Michael J.quinn. *Data-Parallel Programming on MIMD Computers.* The MIT Press, 1991.

[2] Charles H.Kowlbel, David B.Loveman, Robert S.Schreiber, Guy L.Steele Jr., and Mary E.Zosel. *The High Performance Fortran Handbook.* The MIT Press, 1994.

[3] Chau-Wen Tseng. An Optimizing FORTRAN D Compiler for MIMD Distributed Memory Machines. Technical Report CRPC-TR93291-S, Rice University, Jan 1993.

[4] Ken Kennedy Seema Hiranandani and Chau-Wen Tseng. Compiler support for machine-independent parallel programming in fortran d. Technical Report CRPC-TR 91132, Center for Research on Parallel Computing, Rice University, 1991.

[5] Charles Koelbel. Compile-time generation of regular communication patterns. In *Proceedings of Supercomputing '91*, pp. 101–110, Albuquerque, NM, November 1991.

[6] Piiyush Mehrotra Charles Koelbel and John Van Rosendale. Supporting shared data structures on distributed memory architectures. In *Proceedings of the Second ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming*, pp. 177–186, Seattle, WA, March 1990.

[7] Fred J. E. Long Siddhartha Chatterjee, John R. Gilbert, Robert Schreiber, and Shang-Hua Teng. Generating local addresses and communication sets for data-parallel programs. *Journal of Parallel and Distributed Computing*, Vol. 26, No. 1, pp. 72–84, April 1995.

[8] John Mellor-Crummey Seema Hiranandani, Ken Kennedy and Ajay Sethi. Compilation techniques for block-cyclic distributions. Technical Report CRPC-TR 95521-S, Center for Research on Parallel Computing, Rice University, 1995.

[9] Craig M. Chase and Anthony P. Reeves. Data remapping for distributed-memory multicomputers. In *Proceedings of Scalable High Performance Computing Conference SHPCC-92*, pp. 137–144, Los Alamitos, CA, USA, April 1992.

[10] 石崎一明, 小松秀昭. 分散並列計算機のための並列性抽出法. 電子情報通信学会技術研究報告, Vol. 94, No. CPSY-163, pp. 97–104, 7月 1994.

[11] 中谷登志男. Compiling HPF for A Cluster of Workstations. 並列処理シンポジウム JSPP'93, pp. 1–6, 5月 1993.

[12] 大澤暁, 小松秀昭. ワークステーション・クラスタにおける動的なデータ交信／実行管理方法. 情報処理学会第48回全国大会講演論文集 (4), pp. 73–74, 3月 1994.

[13] 菅沼俊夫, 小松秀昭. マルチプロセッサシステムにおけるリダクションオペレーション. 情報処理学会第48回全国大会講演論文集 (5), pp. 69–70, 3月 1994.

[14] 小笠原武史, 小松秀昭. コンパイル時未定義ループ不変値を添字式に持つループの最内ループ並列化の1手法. 情報処理学会第49回全国大会講演論文集 (5), pp. 65–66, 9月 1994.

[15] 小笠原武史, 小松秀昭. データ依存不確定ループの最内ループ並列化. 電子情報通信学会技術研究報告, Vol. 94, No. CPSY-384, pp. 57–64, 10月 1994.

[16] Kazuaki Ishizaki and Hideaki Komatsu. Loop Parallelization Algorithm for HPF Compiler. In *Eighth Annual Workshop on Language and Compilers for Parallel Computing*, pp. 12.1–15, Ohio, August 1995.

[17] 石崎一明, 小松秀昭. ＨＰＦコンパイラにおける並列化手法. Vol. 95,, 8月 1995.

[18] 郷田修, 大澤暁, 小松秀昭, 菅沼俊夫, 小笠原武史, 石崎一明, 中谷登志男. ＨＰＦコンパイラの実装と評価. Vol. 95,, 8月 1995.