計算機アーキテクチャ 130－7
（1998. 8. 5）

分散メモリ型並列計算機向けの自動データ分散並列化方式

西谷康仁† 太田寛†
布広永示† 菊池純男†

分散メモリ型の並列計算機でプログラムの性能を引き出すためには、適切なデータ分散を行なうことが必要である。従来データ分散の指示はユーザーにまかされていたが、最適なデータ分散は様々な要素に依存するため見つけるのが難しく、並列化プログラミングを困難にする一因となっていた。この問題を解決するため、並列化コンパイラによる自動データ分散が必要とされている。本稿では、分散パターンの候補を評価し、動的分散を考慮した最適な分散を並列化コンパイラが自動的に決定する自動データ分散方式について検討する。

Automatic data distribution and parallelization technique for distributed-memory multicomputers.

YASUNORI NISHITANI,† HIROSHI OHTA,† EIJI NUNOHIRO† and SUMIO KIKUCHI†

To achieve high performance on the distributed-memory parallel machines, it is necessary to decide proper data distribution. Usually data distribution is done by the user directive, but the best distribution depends on various factors and cannot be found easily, resulting that making parallel programs is very hard. To solve this probelm, automatic data distribution by the parallelizing compiler is needed. In this paper, we discuss the automatic data distribution method which the parallelizing compiler evaluates candidate distribution patterns and automatically decides the best distribution considering dynamic remapping.

1. はじめに

分散メモリ型並列計算機における並列化プログラミングの難しさを緩和するため、HPF[1] に代表されるデータ並列型言語が提案され、これに対応した並列化コンパイラが数多く開発されている。分散メモリ型の並列計算機でプログラムの性能を引き出すためには適切なデータ分散を行なうことが重要であるが、HPF ではデータ分散の決定はユーザー指示に任されている。しかし、最適なデータ分散を見つけるのは必ずしも容易なことではない。なぜなら、最適なデータ分散は、ターゲットマシンや問題サイズ、使用するコンパイラ等、様々な要素に依存するからである。特に、並列化コンパイラによって並列化方法、能力が異なるため、ユーザーはコンパイラがどのように並列化を行なうかを意識してデータ分散を決定しなければならず、HPF の特徴の一つであるポータビリティがあまり活かされていないのが現状である。

この問題を解決するためには、並列化コンパイラが自動的に最適なデータ分散を決定する技術が必要である。自動データ分散により、ターゲットマシンや問題サイズに最適化したデータ分散を行なうことが可能となる。また、コンパイラ自身が、どのような並列化を行なうかを考慮してデータ分散を決定するため、ユーザーはコンパイラの並列化について意識する必要がなくなる。

自動データ分散技術に関する研究はこれまでも数多く行なわれている[2)3)4)5)]。しかし、これらの多くは、配列の分散対象次元数が 1 次元に限られていた。また、配列間の整列について、基本的に可能な組み合わせ全てを評価するため、解析時間が増大するという問題があった。

そこで本稿では、配列の分散対象次元数として、複数の次元を扱うことが可能な自動データ分散方式を提案する。また本方式では、配列間の整列について、ルー

† 新情報日立研究室
RWCP Hitachi Laboratory

プ分散が可能になるように整列を決定することで、整列に関する解析時間の増加を抑える。

2. 自動データ分散コンパイラの構成

自動データ分散コンパイラの構成を図1に示す。本処理系は、配列データの分散形状を自動的に決定する自動データ分散決定部と、決定したデータ分散形状に基づいてプログラムを並列化するプログラム並列化部からなる。

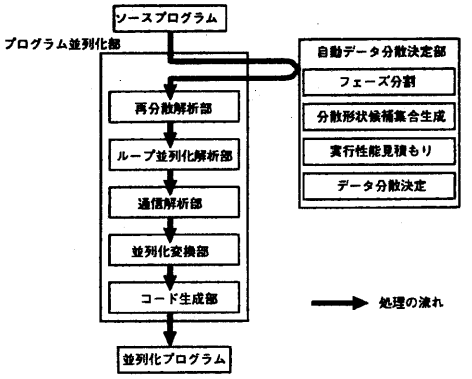

図1 自動データ分散コンパイラの構成

データ分散決定部は、プログラムの構造やデータフローを解析して、最適なデータ分散形状を自動的に決定する。

プログラム並列化部は、HPF コンパイラ（トランスレータ）[6] と同じ構成、機能を持つ。自動データ分散決定部が加わることによって、HPF 指示を含まない単なる逐次プログラムから、自動的に並列化されたプログラムを生成することが出来るようになる。

3. 自動データ分散方式

3.1 機能概要

本節では、自動データ分散方式の機能概要を述べる。

自動データ分散方式を検討するにあたって考慮しなければならない基本的な問題点として、以下の2つが挙げられる。

- 最適なデータ分散の評価基準
  並列プログラムの性能は、並列性、負荷バランス、通信コスト、同期コスト等様々な要素により決定される。これらの要因は、互いに関係しあっている。例えば分散する次元数を増やして並列性を高めると、通信コストが増大するといった具合にである。そのため、各々を独立に最適化することは出来ず、最適なデータ分散を行なうにはこれらのトレードオフを考えなければならない。これらは単位の異なる量であるから、直接比較することは出来ないので、実行時間で比較するのが最も適当と考えられる。すなわち、コンパイル時の性能見積もりが重要になる。
- 動的な分散形状の変更 (再分散)
  各ループにおいて最適な分散形状は、ループ間での再分散を考慮すると、プログラム全体については必ずしも最適とは限らない。再分散のオーバーヘッドを含めたプログラム全体の実行時間が最小となるように、各ループでの分散形状を決定する必要がある。

これらをふまえて、自動データ分散並列化方式を以下の4つの機能で構成する。

(1) フェーズ分割
プログラムを、フェーズという単位に分割する。フェーズとは、配列全体に対して同じアクセスパターンが続くようなひとまとまりの計算であり、データ分散はフェーズを単位にして解析する。

(2) 分散形状候補集合生成
各フェーズ毎に、そのフェーズに含まれる全ての配列についての分散形状の候補を生成する。
まず、フェーズ内の配列の整列関係を決定する。整列関係は、フェーズ内でユニークなテンプレートに全配列を整列させることで表現する。その後に、テンプレートの分散を行ない、分散形状の候補集合を生成する。

(3) 実行性能見積もり
(2) で生成した分散形状候補のそれぞれについて、そのフェーズの実行時間を見積もる。

(4) データ分散決定
(1) から (3) の情報を基にして、DLG (Data Layout Graph) を生成する。DLG のノードは各フェーズでの分散形状候補を表し、エッジはノード間でのリマッピングを表す。ノードとエッジは (3) で求めた実行時間により重み付けされる。
DLG が出来上がると、最適なデータ分散は、DLG 上の最短経路を求めることによって得られる。

3.2 フェーズ分割

データ分散を考える単位として、プログラムをフェーズに分割する。フェーズとは、「配列全体に対して同じアクセスパターンが続くようなひとまとまりの計算」であり、「フェーズ中に含まれるいずれかの配列の添字として現われるインダクション変数を定義するような最大のループネスト」として定義する [3]。

データ分散は、フェーズ内では固定とする。よって、

フェーズ内のそれぞれの配列に対して一つの分散形状を決定することが目標となる。再分散は、フェーズとフェーズの間でのみ行なわれる。

フェーズというまとまった単位で分散形状を考えることにより、ループ単位に解析を行なうよりも解析を簡単にすることが出来る。

例えば、図2のようなプログラムでは、DO 10、DO 20、DO 30 の 3 つのループネストがフェーズとして認識される。DO 40 のループは、内側にその制御変数 itr を添字に持つ配列がないため、フェーズを構成するループとはならない。

```
      REAL x(N,N,N),c(N,N,N)
      DO 40 itr=1,MAXITR
        DO 10 k=1,N
          DO 10 j=1,N
            DO 10 i=2,N
              x(i,j,k)=x(i,j,k)-x(i-1,j,k)*c(i,j,k)
10      CONTINUE
        DO 20 k=1,N
          DO 20 j=2,N
            DO 20 i=1,N
              x(i,j,k)=x(i,j,k)-x(i,j-1,k)*c(i,j,k)
20      CONTINUE
        DO 30 k=2,N
          DO 30 j=1,N
            DO 30 i=1,N
              x(i,j,k)=x(i,j,k)-x(i,j,k-1)*c(i,j,k)
30      CONTINUE
40    CONTINUE
      ENDDO
```

図 2　フェーズの例

フェーズが決定されたら、それを基に PCFG (Phase Control Flow Graph) を構成する。これは、各フェーズ間の制御の流れを表すグラフである。具体的には、フェーズをまたぐループ構造や、分岐構造に応じて、フェーズ間をエッジで結合していく。各エッジには、ループの繰り返し回数や、分岐確率を情報として蓄えておく。この情報を基に、後のフェーズで DLG の重み付けを行なう。

図 2 のプログラムを PCFG で表すと図 3 のようになる。

3.3　分散形状候補集合生成

3.2 節で生成した各フェーズに対し、そのフェーズ内で参照される配列に対する分散形状の候補集合を生成する。候補生成は、整列解析と分散解析の 2 段階に分けて行なう。

3.3.1　整 列 解 析

フェーズ内の配列の整列関係を解析する。配列間の整列の組み合わせの数は無限に近いので、全ての整列

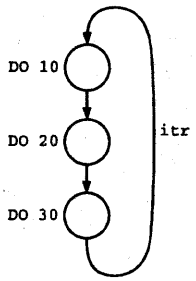

図 3　PCFG の例

パターンを考えるのは不可能である。しかし、現実のプログラムでは、ループを分散することを考えると、可能な整列というのは配列添字の形から一意に決ってしまう場合がほとんどである。

整列は、次元間の整列と次元内の整列に分けられる。次元間の整列とは、配列のある次元を別の配列（テンプレート）のどの次元に整列させるかという問題である。次元内の整列は、ストライドとオフセットの 2 種類がある。例えば、

```
!HPF$ align x(i) with t(a*i+b)
```

と書いた時、a がストライドで、b がオフセットである。この内、オフセットについては今回は考慮しないことにする。理由は、オフセットを考慮することにより削減できる通信は隣接通信（HPF でいう shadow 領域の通信）がほとんどであり、これは比較的オーバーヘッドが小さいからである。また、オフセットまで考慮すると分散形状の候補の数が多くなってしまうという問題もある。

以上の前提に基づき、整列を以下のように決定する。

(1)　テンプレートを、フェーズ毎に一つ作成する。テンプレートは、フェーズ内に現われる全ての配列と、フェーズを構成するループが作るイタレーション空間（ループのイタレーション全体からなる空間）のうちで、最高の次元数を持つものと同じ次元数を持つ。

(2)　テンプレートに、イタレーション空間を、テンプレートの 1 次元目から順にマッピングする。

(3)　フェーズ内の配列について、テンプレートへのマッピングを決定する。この時、整列する次元を分散した時に出来るだけ通信が少なくなるようにする。このため、まず配列添字がループ制御変数の 1 次式で表される次元について、その次元を、対応するイタレーション空間の次元と同じテンプレートの次元に整列する。その結果、どのテンプレート次元とも対応しなかった次元については、整列を COLLAPSED（折り畳み）とし、分散の対象から外す。また、ど

```
アルゴリズム: フェーズ内配列の整列決定方法
1  イタレーション空間の内側次元から外側次元までを、テンプレートの 1 次元目から順にマッピングしていく
2  for （フェーズ内の各配列について） {
3    for （イタレーション空間の外側次元から順に） {
4      for （配列の外側次元から順に） {
5        if （配列のその次元の添字が a*i+b の形である） {
6        /* i はイタレーション空間のその次元に対応するループ制御変数、a,b は定数 */
7          配列のその次元を、対応するイタレーション空間の次元と同じテンプレートの次元に、
           ストライド a でマッピングする (必要であればテンプレート全体を a 倍に拡大する)。
8          break;
9        }
10     }
11     if （イタレーション空間のその次元が配列のどの次元とも対応しない） {
12       対応するテンプレートの次元について整列を REPLICATED（複製）とする。
13     }
14   }
15   for （配列の各次元について） {
16     if （どのテンプレート軸とも対応しない） {
17       その次元の整列を COLLAPSED（折り畳み）とする。
18     }
19   }
```

図 4　整列のアルゴリズム

の配列の次元とも対応しなかったテンプレートの次元については、整列を REPLICATED（複製）とする。

一つの配列について、フェーズ内に複数の参照点がある場合は、その中の任意の一つの参照点を選んで解析する。オフセット付の整列を考えない場合はこれで十分である。

以上のアルゴリズムをまとめたものをを図 4 に示す。

本アルゴリズムによれば、フェーズ内の配列の整列関係は唯一通りに決定してしまう。これは、実プログラムにおいてはそのような場合がほとんどであるという根拠に基づくものであるが、場合によっては、複数の整列候補を考えることによって最適化の可能性が増えることもあるかもしれない。この可能性については、今後の本方式の実プログラムへの適用を通じて評価する予定である。

3.3.2 分散解析

前項で生成したフェーズ毎のテンプレートについて、可能な分散のパターンを列挙して候補集合を生成する。

プログラムから最大限の並列性を引き出すためには、配列のいずれか 1 次元のみを分散するだけでは不十分な場合がある。そこで、本方式では、配列の任意の複数次元を分散することが出来るようにする。各次元の分散は、BLOCK または COLLAPSED とする。

分散形状候補生成の手順を以下に示す。

(1)　テンプレートの各次元について、その次元を分散した場合にループが分散可能かどうか調べる。ループ分散が不可になるようなテンプレートの次元は、分散を COLLAPSED に固定する。この理由は、分散メモリ向けの並列化コンパイラの場合、ループ分散が不可能となるように配列が分散された場合、配列の一要素ずつについて通信を行なうようなコードとなり、性能が極端に劣化するからである。

(2)　残った分散可能な次元について、各次元を BLOCK 又は COLLAPSED にする全ての組み合わせを列挙して、分散形状候補を生成する。

3.4 実行性能見積もり

各フェーズについて、3.3 節で生成した分散形状の候補毎に、その候補に対する実行性能の見積もりを行なう。実行性能は、相対的な時間が分かれば良い。

実行性能見積もりのため、まずある分散形状の候補について、ループ分散解析や通信解析を行ない、ループの並列化情報や必要となる通信を求める。これは、フェーズ内の各配列について、分散形状を候補となる分散形状に仮定し、実際に並列化コンパイラの処理を途中まで実際に行なってみることで得ることが出来る。

コンパイラによる並列化は、いくつかの並列化形態に分類出来る。並列化形態としては、

- DO ALL 型
- DO ACROSS 型

- リダクション型

等が挙げられる。性能見積もりのため、これらの並列化形態について、実行時間をパラメータを用いて表現する。パラメータとしては、

- 計算量（ループ中の演算数 × ループ長）
- プロセッサ数
- 通信起動オーバーヘッド
- 通信スループット

などが考えられる。実機での評価により、これらのパラメータを基にして実行性能をモデル化し、見積もりを行なう。

実行性能見積もりにおいて問題となるのは、評価のためのパラメータにコンパイラにとって未知のものが含まれる場合である。本方式では、プロセッサ数、問題サイズ（配列サイズ、ループ長）はコンパイル時に判明していることを前提としている。しかしこれは、実プログラムにおいては厳しい制限であり、実際にはこれらのパラメータが変数として与えられている場合も多い。また、プログラムに分岐が含まれる場合、実行性能を見積もるためには分岐確率を正確に予測することが必要となるが、これはコンパイル時には不可能な場合が多い。

この問題に対処する方法としては、以下の2つが考えられる。

- 対話的ツールとの連携により、ユーザーからヒントをもらう
- 実行プロファイル情報を収集し、コンパイラにフィードバックすることで情報を補う

これらは今後検討していくべき課題であるが、現段階では、本方式の妥当性を検証することを優先し、未知のパラメータについてはコンパイラが適当な値を仮定するという方法で対処することにする。

3.5 データ分散決定

これまでに得られた解析結果を基にして、各フェーズ毎のデータ分散を決定する。ここで言うデータ分散とは、再分散を含めた動的な分散形状のことである。

最適な分散をプログラム全体について決定するために、3.2 節で作成した PCFG と、3.3 節で生成した分散形状候補集合を基に、DLG（Data Layout Graph）を構成する。

DLG は、PCFG の各ノードにそのフェーズにおける分散形状候補を対応させ、PCFG のエッジに沿って、各分散形状候補を結んで作成する。DLG の各ノードには、そのノードに対応する分散形状に対する実行性能見積もりの結果を持たせる（通信コストを含む）。DLG の各エッジには、エッジの両端の分散形状間で起こり得るリマッピングのコストを重みとして持たせる。

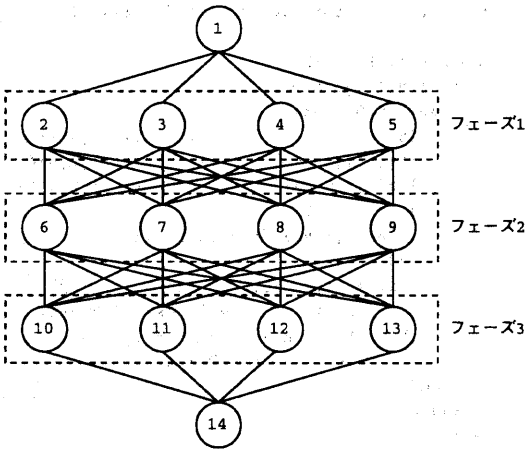

図 5 DLG の例

例えば、フェーズ 1〜フェーズ 3 の 3 つのフェーズからなり、各フェーズ毎に 4 つの分散形状候補が存在するような DLG は図 5 のようになる。ここで、ノード 1 とノード 14 は、開始状態と終了状態を表すために設けた仮想的なノードで、このノードと他のノードを結ぶエッジの重みは 0 とする。

DLG が構成されると、最適な分散形状は、DLG の開始ノードから終了ノードに至る最短経路を求めることにより得られる。

最短経路探索問題の解法としては様々なものが考案されており、ここでは特に述べない。

4. 適 用 例

本節では、これまで述べた方式の適用例を、サンプルプログラムを例にして説明する。

図 6 は、NAS Parallel Benchmarks の SP より、カーネル部分の一部を切り出して簡略化したものである。このプログラムに本方式を適用すると、以下のようになる。

(1) フェーズ分割

フェーズ分割により、do k, do j, do i, do m の 4 つのループからなるループネストが 1 つのフェーズとなる。

(2) 分散形状候補集合生成

3.3 節に従って分散形状の候補集合を生成する。まず、ループネスト数は 4、配列の最大次元数も 4 であるので、4 次元のテンプレートが作られる。これに、図 4 のアルゴリズムを適用すると、配列 lhs は、1,2,3 次元目が、それぞれ i,j,k ループに対応するテ

```
 double precision
>     lhs (-2:isize,0:isize,0:isize-1, 15),
>     rhs (-2:isize,0:isize,0:isize-1, 5)

 do k = 1, ksize-2
   do j = 1, jsize-2
     do i = 0, iend-2
       fac1              = 1.d0/lhs(i,j,k,n+3)
       lhs(i,j,k,n+4)    = fac1*lhs(i,j,k,n+4)
       lhs(i,j,k,n+5)    = fac1*lhs(i,j,k,n+5)
       do m = 1, 3
         rhs(i,j,k,m) = fac1*rhs(i,j,k,m)
       end do
       lhs(i+1,j,k,n+3) = lhs(i+1,j,k,n+3) -
>           lhs(i+1,j,k,n+2)*lhs(i,j,k,n+4)
       lhs(i+1,j,k,n+4) = lhs(i+1,j,k,n+4) -
>           lhs(i+1,j,k,n+2)*lhs(i,j,k,n+5)
     end do
   end do
 end do
```

図 6 サンプルプログラム

ンプレートの次元に整列し、4 次元目については対応するループがないので整列は COLLAPSED になる。配列 rhs は、1,2,3,4 次元目が、それぞれ i,j,k,m ループに対応するテンプレートの次元に整列する。整列が決定したら、テンプレートの分散を行なう。ここで、テンプレートの do i ループに対応する次元を分散すると、配列 lhs の 1 次元目の添字がループ分散に適さないため、ループ分散が不可能になってしまう。よって、テンプレートの do i ループに対応する次元は分散を COLLAPSED に固定する。残りの 3 つの次元は、それぞれ分散を BLOCK と COLLAPSED の 2 通りとし、その組み合わせで計 8 通りの分散形状候補を生成する。

(3) 実行性能見積もり

3.4 節に基づいて各分散形状候補に対する実行性能見積もりを行なう。

(4) データ分散決定

3.5 節に基づいてデータ分散を決定する。この例では、フェーズが 1 つしかないので実行性能見積もりの結果が一番優れていた候補がそのまま選択される。実行性能見積もりの結果、do m ループに対応する次元は分散しても良い並列性が得られないので分散されない。do k, do j ループに対応する次元の分散は、パラメーター isize とプロセッサ数 p の関係により異なる。

p<isize が成り立つ場合、do k ループに対応する次元が分散される。これは、外側ループで並列化した方が並列化のオーバーヘッドが少ないからである。この結果は、HPF を用いた人手による並列化の結果と一致する。

p>isize が成り立つ場合、2 つの次元を並列化した方が並列性が高くなる。その結果、do k と do j の 2 つのループに対応する次元が分散される。

このように複数次元の分散を考慮することにより、プログラムから最大限の並列性を引き出すことが可能になった。

5. おわりに

最適なデータ分散を自動的に行なう並列化コンパイル方式について検討した。

自動データ分散技術に関しては、これまで様々な手法が提案されており、どの方法が優れているかというのは一概には言えない状況である。自動データ分散技術の実用化のためには、実アプリケーションへの適用評価を多く行なうことが必要と考えられる。

今後の予定としては、試作中の本方式に基づいた自動並列化コンパイラの完成後、広範囲な実プログラムに適用し、その妥当性について検証を行なう予定である。また、3.4 節で述べたように、対話的ツールとの連携、実行プロファイル情報のフィードバックによる解析精度の向上についても今後検討する必要がある。

参考文献

1) High Performance Fortran Forum: *High Performance Fortran Language Specification Version 2.0* (1997).
2) Anderson, J. M. and Lam., M. S.: Global Optimizations for Parallelism and Locality on Scalable Parallel Machines, *Proceedings of the ACM SIGPLAN'93 Conference on Programming Language Design and Implementation* (1993).
3) K.Kennedy and U.Kremer: Automatic Data Layout for High Performance Fortran, *Proceedings of Supercomputing'95*, San Diego, CA (1995).
4) Palermo, D. J. and Banerjee, P.: Automatic Selection of Dynamic Data Partitioning Schemes for Distributed-Memory Multicomputers, *Proceedings of the 8th Workshop on Language and Compilers for Parallel Computing* (1995).
5) J.Garcia, E.Ayguadé and J.Labarta: A Framework for Automatic Dynamic Data Mapping, *Proceedings of the 8th IEEE Symposium on Parallel and Distributed Processing* (1996).
6) 佐藤真琴, 太田寛, 布広永示: HPF トランスレータ "Parallel FORTRAN" の開発と評価, 情報処理, Vol. 38, No. 2, pp. 105–108 (1997).