並列化コンパイラ TINPAR における自動データ分割部の実現

辰巳尚吾[†]　窪田昌史[†]　五島正裕[†]
森眞一郎[†]　中島浩[†]　富田眞治[†]

本稿では、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 における自動データ分割部について述べる。owner computes rule に従った並列化を行う並列化コンパイラでは、データ分割によって、必要となる通信量や、並列化効率、プロセッサ間の負荷バランスなどが左右されるため、これらを考慮に入れた最適なデータ分割を決定する必要がある。我々の自動データ分割部は、最初に複数の配列変数どうしの相対的な配置関係を求め、それをもとに、分割する次元を決定する方法を採用する。各決定は、上記の 3 つを評価基準として行われる。

livermore kernel No.23 に対し自動データ分割を行い、それ以外のデータ分割との実行時間を比較したところ、自動生成した分割が、評価したプログラムに対して最も良いデータ分割を求められることを確認した。

An Implementation of the Automatic Data Distribution Phase of the Parallelizing Compiler TINPAR

Shogo Tatsumi,[†] Atsushi Kubota,[†] Masahiro Goshima,[†]
Shin-ichiro Mori,[†] Hiroshi Nakashima[†] and Shinji Tomita[†]

This paper prsents the automatic data distribution phase of TINPAR; a parallelizing compiler for message-passing multiprocessors. In compilers parallelizing according to the owner computes rule, a data distribution affects the amount of necessary messages, the effeciency of parallelizing and the load balance between processors. Thus, the data distribution phase has to decide the optimal data distribution considering those factors. First, our automatic data distribution phase obtains the relations about the distribution between all array variables. Then it decides the data distribution.

As a result of the comparison of the execution times of livermore kernel No.23 under several distribution patterns, we confirm that the distribution pattern automatically generated by our compilers is the optimal distribution for this program.

1. はじめに

現在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が盛んに研究されている。メッセージ交換型並列計算機において効率よく問題を解くためには、並列性の抽出のみならず、プロセッサ間の負荷バランスをとることや、大きなオーバヘッドを伴うメッセージ通信の量を減らすことも重要である。owner computes rule に従った並列化を行う場合、これらは、プログラム中の配列変数のプロセッサへの分割方法で決まる。即ち配列変数の分割は、計算の性能に大きな影響を与える重要な決定要素である。

現在は、プログラマにこのデータ分割を指定させ、その情報をもとに並列化を行うコンパイラが多く利用されているが、最適なデータ分割を決定するのは、プログラマにとって非常に困難なことである。

そこで我々は、逐次コードを解析し、並列化のための最適なデータ分割法を決定する自動データ分割部を開発し、既存の並列化コンパイラ TINPAR(TINy PARallelizer)[1),2)] に実装を行っている。

第 2 節で、データ分割部の概要について述べる。次に第 3 節で我々が実装したデータ分割部の詳細について述べる。続いて、第 4 節でデータ分割部の評価を行い、最後に第 5 節でまとめる。

2. データ分割部の概要

2.1 対象とする並列化コンパイラ

実装したデータ分割部は、次の条件を満足する並列化コンパイラを対象として設計を行った。

---

† 京都大学大学院工学研究科情報工学教室
Department of Information Science, Faculty of Engineering, Kyoto University

- owner computes rule に従った並列化を行う。
- データ分割として、配列変数の各次元それぞれに対し、ブロックサイクリック分割のサイズが指定できる。

並列化コンパイラ TINPAR は上記の条件を満たしており、本データ分割部は TINPAR に対して実装中である。

2.2 データ分割部の概要

2.2.1 データ分割を指定するパラメータ

データ分割の方法としてブロックサイクリック分割のみを考えると、次のパラメータによってデータ分割を指定することができる。

- 配列変数どうしの相対的な配置関係
- 配列変数のどの次元を分割するのか
- 分割のブロックサイズ

2.2.2 データ分割の方針

データ分割を行う際の評価項目として、以下の 3 つを考え、それぞれで述べる方針に従ってデータ分割を行う。

- 並列実行時のプロセッサ間の通信量
  ソースプログラム中のあるループボディに、そのループのループ変数を添字式に含む配列変数どうしの代入文があった場合、owner computes rule に従った並列化を行って、左辺の配列要素を持つプロセッサが、右辺の配列要素を持つプロセッサからそのデータを送信してもらわなければならない。そこで、ループ変数の変化に伴い、どの配列要素が同時に必要とされるのかを配列変数どうしの代入文から解析し、これらの配列要素を同じプロセッサに割り当てるような分割を求める。
- 抽出される並列度
  owner computes rule に従った並列化では、データ分割を決定した段階で、どのイタレーションを並列化するかが一意に定まってしまう。しかしながら、並列化されたイタレーション間に、解決できない依存関係が存在すると、十分な並列度が得られない。そこで、データ分割を決定する際に、イタレーション間の依存解析を行い、プログラム全体を通して最大限並列度を抽出できるデータ分割を選択する。
- プロセッサ間の負荷のバランス
  owner computes rule に従った並列化では、データ分割のブロックサイズを小さくすることによって、負荷バランス保つことが可能である。また、データ分割が決定した段階では、各データ分割に対してプロセッサでの負荷がある程度予測できる。このため、この予測をもとに、負荷バランスが保てるようなデータ分割 (特にブロックサイズ) を求める。

2.2.3 データ分割部の概要

本手法では、第 2.2.1節で述べたデータ分割を指定するパラメータを決定することによって、データ分割の決定を行なう。それぞれに対応して、以下の 4 つのフェーズにより構成した。各フェーズでは、第 2.2.2 節で述べた方針を考慮に入れて決定を行う。

(1) 配列次元アラインメントの決定
(2) ブロックサイズの制約の決定
(3) 分割する次元の決定
(4) ブロックサイズの決定

「配列変数どうしの相対的な配置関係」の決定は、上記 (1)(2) の 2 フェーズで実現する。最初の 2 フェーズで有意な相対的配置関係を求めておくことで、「分割する次元の決定」において各配列変数を分割する次元の組み合わせの数を、大幅に減らすことができる。

分割する次元の決定では、前のフェーズで決定された配置関係を保ちながら、配列変数のどの次元を分割するかの優先度を決定する。最後のフェーズでは、前のフェーズで求まった分割する次元に対して、それまでのフェーズでの決定を保ちながら、分割のブロックサイズを求める。

なお、実装中のデータ分割部では、現在のところ、以下の制限が設けられている。

- 配列変数のいずれか 1 次元のみを分割する。
- プログラムの途中でデータの再分散 (redistribution) は行わない。
- 解析の対象とする添字式は、いずれか 1 つのループ変数の 1 次式で表現されるものに限る。

3. データ分割部の詳細

3.1 配列次元アラインメントの決定

並列実行時のプロセッサ間の通信量を最小にするには異る配列変数の各次元を分割の際にそれぞれどのように対応づけられればよいか、という相対的な関係を求めることが、本フェーズでの目的である。それらの関係は、プログラム中に現れるすべての配列変数の次元のグループ分けとして表現される。このグループ分けを、本稿では、配列次元アラインメントと呼ぶ。配列次元アラインメントの決定は、配列変数の参照関係の解析によって行う。

本データ分割部では、Li らによって提案された、CAG (Component Affinity Graph) を利用した方法6) によって通信量を最小にする配列次元アラインメントを求める。

3.1.1 CAG の構成手順

まず、アフィン関数、アフィン関係、アフィン関係にある次元という用語を以下のように定義する[☆]。

- アフィン関数— $f(i) = a \times i + b$ の形であらわせる関数 f。ただし、a, b は整数、$a \neq 0$。

[☆] より一般的には、アフィン関数は、ベクトルから整数への写像として表され、アフィン関係もこのアフィン関数に対して定義される。

- アフィン関係— 変数 (上の関数 fでは i) が同じアフィン関数どうしを、互いにアフィン関係にあるという。
- アフィン関係にある次元— ある代入文において、左辺の配列変数の 1 つの次元の添字式と、右辺の配列変数の 1 つの次元の添字式がアフィン関係にあるとき、これらの次元をアフィン関係にあるという。

「配列変数の次元間にアフィン関係がある場合、それらの次元に対して同じ分割をする」、という分割についての相対的な関係を決めておけば、分割の有無に関わらず通信は不要となる。この事実を利用するために、CAG は以下の手順に従って作成される。

(1) プログラム中のすべての配列変数の各次元に対して 1 つのノードを作成する。
(2) プログラム中の配列変数どうしの代入文すべてに対して、以下の手続きを行う。
  (a) 代入文の左辺の配列変数と、右辺に現れる配列変数の次元のすべての組に対して、アフィン関係を調べる。
  (b) アフィン関係にある次元に対応するノードの間にエッジを作る。
  (c) エッジの重みとして、現在解析中の代入文の実行回数をつける。

3.1.2 CAG と配列次元アラインメント

CAG が与えられると、プログラムに現れる配列変数の最高次元数を n_max とした時、以下の条件を満足する n_max 個の排他的なグループを作成可能である。

- CAG のすべてのノードはいずれか 1 つのグループに属する
- 1 つのグループに同一配列変数の次元に対するノードは 1 つのみ存在可

この n_max 個の排他的なグループの構成が配列次元アラインメントに対応する。

最適な配列次元アラインメントの決定は、CAG において、上記のノードのグループ分けのうち、グループ間を結ぶエッジの重みの和が最小となるようなグループ分けの決定に相当する。

Li らによって提案されている手法[6]では、エッジの重みとして、1 または、$\epsilon (0 < \epsilon \ll 1)$ をつけている。我々は、より正確なエッジの重みづけを行うため、その両端のノードが、ノードのグループ分けで同じグループに属さなかった場合に被る不利益を評価した値、即ち、そのエッジの存在を導いた代入文によるデータの移動量を考慮する。具体的には、重みとして代入文の実行回数を採用した。

```
        for j = 0 , n - 1 do
            for i = 0 ,n/2 - 1 do
S1:             A(j,2*i) = B(i,j)
            endfor
        endfor
                  ⋮
        for j = 0 , n - 1 do
            for i - 0 , n - 1 do
S2:             A(i,j) = B(i,j)
            endfor
        endfor
```

図 1 CAG による配列次元アラインメント決定の例

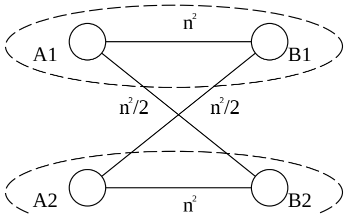


図 2 図 1 のプログラムの CAG

今、図 1のプログラムを考えてみる (対応する CAG は図 2) となる)。配列変数 `A` に対して `A` の `i` 次元目を $\mathtt{A}_{\mathtt{i}}$ と表現する[3),6)]と、文 S_1から、A_1と B_2、A_2と B_1というグループ分けの候補が、文 S_2から、A_1と B_1、A_2と B_2というグループ分けの候補が生成される。エッジの重みの大小関係から、この図 2の点線で囲んだ次元どうしをグループにするような配列次元アラインメントが最適であると求まる。

3.2 ブロックサイズの制約の決定

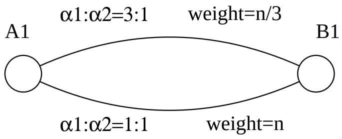
本フェーズは、並列実行時のプロセッサ間の通信量を最適にするために、配列変数間の各次元の分割のブロックサイズについての関係を求めることが目的である。配列次元アラインメントの決定フェーズで求まった配列変数の次元のグループ内の各次元の添字式を解析することにより、グループ内でのブロックサイズの制約をブロックサイズの比として求める。

あるグループ内の 2 つの次元の添字式が$\alpha_1 * \mathtt{i} + \beta_1$、$\alpha_2 * \mathtt{i} + \beta_2$ であるとする。このとき、ループ変数 `i` のとりえるすべての値に対して通信なく実行を進めるための、2 つの次元間のブロックサイズについての関係は、それぞれのブロックサイズを $|\alpha_1| : |\alpha_2|$ にするということである。すべての代入文から、これらの関係を求めればプログラム全体としてのブロックサイズの制約を求めることができる。

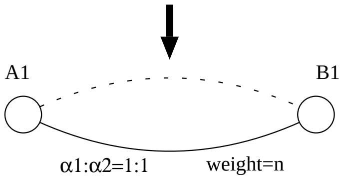
しかし、プログラム全体として、矛盾なくブロックサイズの制約を求められない場合もある。そういう場合であっても、最終的には、各代入文に対して定められたブロックサイズの比を、ある 1 つ次元のグループにつき唯一の比として求めなければならない。この問題を、BCG(Block-size Constraint Graph) と呼ばれる無向グラフを利用して解く方法が、Gupta らによって提案されている[3)]。ノードは配列変数の次元を表す。

```
          for i = 0 , n - 1 do
$S_1$:        A(i) = B(i)
          endfor
                  ⋮
          for i = 0 , n/3 - 1 do
$S_2$:        A(3*i) = B(i)
          endfor
```

図 3　BCG によるブロックサイズの制約の決定の例



(a)図5のBCG



(b)制約の競合の解決

図 4　図 3 に対応する BCG

エッジはその両端のノードに対応する配列変数の次元間にブロックサイズの制約があることを表す。エッジには、「(1) エッジの両端のノードに対応する配列変数の次元のブロックサイズの比」と「(2) その比に決定した時に得られる利益」を表す値を持たせる。BCG にサイクルが存在しない場合、各代入文に対して定められたブロックサイズの比をすべて満足するように、配列変数の次元の組に対して唯一の比を求めることが可能である。サイクルが存在し、かつ、制約が矛盾している場合には、BCG のいくつかのエッジを切って、サイクルをなくすことで比を唯一に定めることができる。最適なブロックサイズの制約を求めることは、BCG において、残されたエッジの重みの和が最大になるようなエッジ集合を求めることに帰着される。

我々は、(2) の利益を、エッジに対応する代入文が実行される回数で表す。

図 3のプログラムに対する BCG を図 4(a) に示す。この例では、S_1とS_2の間で矛盾が生じている。図4(b) では、この矛盾を解決するため、S_2から得られるエッジ (制約) が切られたことを表している。この結果、A_1,B_1を 1:1 の比のブロックサイズにするという制約が最終的に求まる。

3.3　分割する次元の決定

本フェーズは、ループの並列性を、並列化されたプログラムに反映できるような配列変数の分割次元を求めることを目的とする。

本稿では、あるループ文をイタレーション単位でプロセッサに割り当てたときの並列実行の可能性を、そのループに対するループ分割適性度、配列次元アラインメント決定フェーズで求まったアラインメントを構成する各グループに対して、そのグループに属する各配列変数の次元で分割を行った際に予測される性能の良さを、そのグループに対する配列次元分割適性度とそれぞれ呼ぶことにする。本フェーズでは、配列次元分割適性度を求め、適性度の大きい次元から優先的に分割する次元として決定する。

本手法は、配列次元分割適性度を求めるために、まず、ループのイタレーション間にわたるデータ依存解析4) を行い、ループ分割適性度を求める。

イタレーション間にフロー依存がある場合、依存関係を保存することを考えると、そのループは逐次実行しかできない。そのため、イタレーション間にフロー依存があるループを分割した場合、依存のため、データ待ちが生じ、プロセッサが有効に利用されない。一方、イタレーション間に逆依存があるループを分割した場合、代入文での配列変数の参照がプロセッサをまたがる事があり、このときに通信が必要となるが、並列実行は可能である。また、イタレーション間に上記の依存関係がまったくなかった場合、そのループは完全に並列実行が可能である。

これらの事実から、本手法では、ループ間にわたる依存関係の解析の結果として、フロー依存があった場合のループ分割適性度を$-1\times$(ループボディの実行時間)$\times$(ループのイタレーション数) とし、その他の場合のループ分割適性度を 0 とした。ただし、ループボディの実行時間は、ループボディ中の演算回数に比例させた値とする。逆依存があるループを分割した場合は、依存関係のないループを分割した場合よりも通信量が必要な分だけ実行時間が遅くなるが、その差は、フロー依存のあるループを分割した場合の性能の悪さに比べて十分小さいと考えたため、逆依存のあるループに対するループ分割適性度も 0 とした。

さて、次に、上記で定めたループごとのループ分割適性度から、配列次元分割適性度を求める。owner computes rule に従って並列化する場合、代入文の左辺の配列変数の分割する次元の添字式に現れるループ変数を変数とするループが分割されることになる。この事実から、あるループを分割するためには、そのループ変数が添字式に現れる配列変数の次元を分割すればよいことが分かる。しかし、このようにして分割する次元を配列変数ごとに決定した場合、以下の 2 つの点が問題になる。

- ある配列変数が複数のループのボディ内に現れ、それぞれのループで最適分割となる次元が異なる場合がある。
- 異なる配列変数間で、配列次元アラインメントによって求まった関係に矛盾するような分割次元が

求まる場合がある。

これらの問題を解決するために、ループ分割適性度を利用する。具体的には次のようにして配列次元分割適性度を一意に決定する。

(1) プログラム中の代入文で、左辺に配列変数が現われるすべての代入文の各次元について、次の手続きを行う。
    - 現在調べている次元の添字式に含まれるループ変数を変数とするループのループ分割適性度を得る。
    - 現在調べている次元に対応するCAGのノードに対して、得られたループ分割適性度を重みとして足す。

(2) 配列次元アラインメントの決定のフェーズで得られた次元のグループごとに、そのグループに属するすべての次元に対応するCAGのノードの重みを加える。得られた重みを、そのグループに対する配列次元分割適性度とする。

この結果求まった配列次元分割適性度のなかで、最も大きい値を持つ配列変数の次元のグループで分割を行う。

3.4 ブロックサイズの決定

本フェーズは、プロセッサ間の負荷バランス、通信量を最適にするための最適なブロックサイズの決定を行うことを目的とする。

分割のブロックサイズが小さい場合、負荷バランス、通信量という点において、次のようなことがいえる。

- ループのイタレーションごとの演算量の差の有無に関わらず、常に負荷バランスが保てる。
- 配列変数の近傍の要素どうしが頻繁に参照されるプログラムに対しては、ブロックサイズが大きい場合と比べて通信量が非常に多くなる。

逆に、分割のブロックサイズが大きい場合は、次のようなことがいえる。

- ループのイタレーションごとの演算量に差があった場合、負荷バランスが保てないことがある。
- 配列変数の近傍の要素どうしが頻繁に参照されるプログラムに対しては、ブロックサイズが小さい場合と比べて通信量が少なくなる。

このことから、ブロックサイズの決定において解決すべきプログラムの性質として以下の2点が考えられる。

- 配列変数の近傍の要素どうしが頻繁に参照される
- ループのイタレーション間の負荷バランスが悪い

上記の2つの性質のうち、負荷バランスが悪いという性質だけを持つプログラムに対しては、小さいブロックサイズ、近傍の要素どうしの参照が頻繁にあるという性質だけを持つプログラムに対しては大きいブロックサイズとすると、最適な分割が求まる。

しかし、上記の両方の性質を持ち合わせたプログラムに対しては、それぞれに対し何らかの評価関数を用いてブロックサイズを求める必要がある。

本データ分割部では、現在のところ、上記の2つの性質のうち、どちらか1つの性質しか持っていないプログラムのみの存在を仮定し、負荷バランスが悪いと判断した場合はできる限り小さいブロックサイズ、それ以外の場合はできる限り大きいブロックサイズを本フェーズの出力とする。

4. 性能評価

本データ分割部をが採用した方針でデータ分割を行った場合と、それ以外のデータ分割を行った場合について、並列化コンパイラTINPARによって並列化したプログラムの並列計算機AP1000上での実行時間を測定した。

評価用プログラムとして、livermore kernel No.23のプログラムを利用した。図5にlivermore kernel No.23のリストを示す。ただし、`m=n=512`とした。

4.1 自動データ分割

本データ分割部を適用すると、各フェーズで次のような決定が行われる。

配列次元アラインメントの決定

za_1とzr_1、zb_1、zu_1、zv_1、zz_1のそれぞれの組がアフィン関係にある。同様に、za_2とzr_2、zb_2、zu_2、zv_2、zz_2のそれぞれの組がアフィン関係にある。これらの組の間には競合がないため、それぞれの次元の組が対応づけられる。

ブロックサイズの制約の決定

アフィン関係にある次元の組はすべて、その添字式中のループ変数への係数が1である。従って、ブロックサイズの制約としては、すべてのアフィン関係にある次元の組に対して1:1となる。

分割する次元の決定

`j`、`k`いずれのループも、フロー依存、逆依存の両方が存在する。`k`ループのループ分割適性度は$-(\mathrm{n}-2)$、`j`ループのループ分割適性度は$-(\mathrm{n}-2)*(\mathrm{m}-2)$となる。$-(\mathrm{n}-2) > -(\mathrm{n}-2)*(\mathrm{m}-2)$であるから、分割するループは、`k`ループがよいと判断される。これにより、すべての配列変数の1次元目を分割するという結果を得る。

ブロックサイズの決定

両方のループとも一定の回数実行されるので、どちらのループを分割しても、負荷は均等であると決定できる。従ってブロックサイズはブロックサイズの制約のなかでとりえる最も大きなサイズ、即ち、ブロック分割と決定される。

各フェーズの決定から、「すべての配列変数の1次元目をブロック分割」するという結果を得る。

4.2 評価結果

本データ分割部によって決定された分割による実行時間を図6の(1)に示す。(2)は、すべての配列変数の

```
        for j=1,m-2 do
          for k=1,n-2 do
$S_1$:        qa = za(k,j+1)*zr(k,j)+
                   za(k,j-1)*zb(k,j)+
                   za(k+1,j)*zu(k,j)+
                   za(k-1,j)*zv(k,j)+zz(k,j)
$S_2$:        za(k,j)=za(k,j)+.175*(qa-za(k,j))
          endfor
        endfor
```

図 5　livermore kernel No.23 のリスト

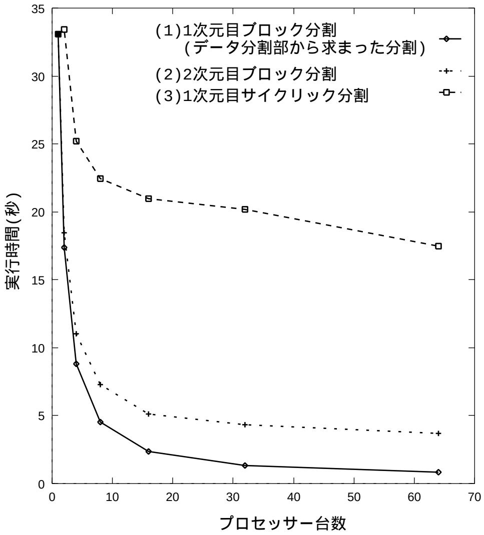


図 6　livermore kernel No.23 の実行時間

2 次元目をブロック分割したものと、(3) は、すべての配列変数の 1 次元目をサイクリック分割したものを表す。

(2) と (1) の実行時間を比べることにより、本手法の分割する次元の決定の結果が正しく反映されていることが分かる。

また、(3) と (1) の実行時間を比べることにより、本手法のブロックサイズの決定の結果が正しく反映されていることが分かる。

5. おわりに

本稿では、現在我々が開発中であるTINPARのデータ分割部について述べた。データ分割部は4 つのフェーズで構成され、各フェーズでは、プロセッサ間の通信量や、並列化効率、負荷バランスなどを評価項目として、分割を求めていく。本データ分割部はまず最初に、配列変数どうしの相対的な配置関係を求め、それをもとに、分割する次元を決定する。

さらに、livermore kernel No.23 のプログラムに対し、本手法で求まった分割とそれ以外の分割の実行時間を比べたところ、本手法では、評価したプログラムに対して最も良いデータ分割を求められることが明らかになった。

本手法では、各フェーズにおいて、プロセッサ間の通信量や、負荷バランス、実行時間など、さまざまな値を簡単な評価関数で見積もっている。しかし、演算と通信がオーバラップされている場合のループ分割適性度や、プロセッサ間の通信量などは、十分に正確であるとはいえない。そこで、今後の予定として、各フェーズにおいて、静的な詳しい解析結果を用いる手法を考えていきたい。また、データの再分散を許した場合のデータ分割の決定手法に関する研究も進めていく必要がある。

謝辞　日頃より御討論いただく京都大学工学部情報工学教室富田研究室の諸氏に感謝致します。また、並列計算機 AP1000 の実行環境を提供して頂いた (株) 富士通研究所に感謝致します。

参 考 文 献

1) 三吉 郁夫, 前山 浩二, 後藤慎也, 森 眞一郎, 中島 浩, 富田 眞治: 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 –最適化手法と評価–, 情処研報 94-HPC-54, pp.45-52, 1994
2) 三吉 郁夫, 前山 浩二, 後藤 慎也, 森 眞一郎, 中島 浩, 富田 眞治: 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, JSPP'95 論文集, pp.51-58, 1995
3) M.Gupta and P.Banerjee: *PARADIGM: A Compiler for Automatic Data Distribution on Multicomputers*, Proc. of 1993 ACM International Conference on Supercomputing, pp.87-96, Jul 1993.
4) 笠原 博徳: 並列処理技術(コロナ社,1991),pp128-pp.135
5) K.Kennedy and U.Kremer: *Automatic Data Layout for High Performance Fortran*, Proc. of Supercomputing, Dec 1995.
6) J.Li and M.Chen: *Index Domain Alignment: Minimizing Cost of Cross-Referencing Between Distributed Arrays*, The 3rd Symposium on the Frontiers of Massively Parallel Computation, pp.424-433, Oct 1990.