ハイパフォーマンス
コンピューティング 79−12
（1999．12．3）

データ並列言語の通信生成方式とマルチグリッド法への適用

太田 寛　西谷 康仁
新情報処理開発機構マルチプロセッサコンピューティング日立研究室

HPF(High Performance Fortran)等のデータ並列言語のコンパイラにおける通信生成の一手法として，従来，配列の再マッピングを利用する方法が提案されている。本研究は，この方法の一般化および最適化強化により，様々な配列添字を持つループへの適用性を高めることを目的としている。まず，一般的なループに対する再マッピング通信の生成方法を定式化する。さらに，シャドウ通信や1対1通信などの高速通信の生成条件や通信の融合条件について述べる。提案方式を実装し，NAS Parallel ベンチマークのMG(マルチグリッド法)に適用してSR2201上で評価した。本方式により，MGのHPF版の実行時間が人手並列化版(NPB2.3β)の1.3倍以内に収まることが示された。

Communication Generation for Data-Parallel Languages and its Application to the Multigrid Method

Hiroshi OHTA, Yasunori NISHITANI
RWCP Multiprocessor Computing Hitachi Laboratory

Array remapping has been used as one of the methods for communication generation in data-parallel language compilers such as HPF (High Performance Fortran). This study aims at improving the applicability of the method to loops with various array subscripts, by generalization and more intensive optimization of the method. First, we formulate the algorithm for generating remappings for general loops. Then we describe the conditions for generating faster communications such as shadow communications and one-to-one communications. We also describe how we merge multiple shadow communications. We have implemented our method and applied it to MG (multigrid method) of the NAS Parallel Benchmarks. The evaluation on SR2201 shows that the execution time of the HPF version of MG is within a factor of 1.3 compared to that of the hand-parallelized version (NPB2.3 beta).

1. はじめに

分散メモリ型マルチプロセッサ向けのプログラミング言語として，HPF(High Performance Fortran)[8]等のデータ並列言語が提案されている。データ並列言語のコンパイラは，データマッピング(データの各プロセッサへの分散割り当て)と計算マッピング(計算処理の各プロセッサへの分散割り当て)に基づいて，プロセッサ間通信を生成しなければならない。

従来，通信生成方法としては，各プロセッサが所有する配列要素集合と参照する集合との差分をコンパイル時に計算する方法[1]や，これを一般化してコンパイル時に整数方程式を解く方法[2]，および，実行時の配列再マッピングを利用する方法[3]などが提案されている。

このうち最後の再マッピング利用方法は，通信を配列の再マッピング(プログラム実行中のマッピングの変更)と見なし，再マッピング用の実行時ライブラリによって通信を実行するというものである。しかし従来は，様々な配列添字パターンに対して，一般的にどのような再マッピングを生成すれば良いかが明確ではなかった。また，シャドウ通信などの特定の通信パターンに対しては，再マッピングの代わりに，通信パターンに特化した高速通信を生成する方法が提案されているが，その適用条件等も必ずしも明確には示されていなかった。

本研究は，再マッピング利用方法を一般化および最適化強化することによって，様々な配列添字を持つループへの適用性を高めることを目的としている。特に，NAS Parallel ベンチマーク(NPB)[9]のMG(マルチグリッド法)のようなストライド付のアクセスパターンに対して，効率的通信が生成できることを目的としている。

以下，第2章では一般的なループに対する再マッピング通信生成アルゴリズムを定式化する。第3章では特定パターン向け高速通信の適用条件や，最適化の条件を示す。第4章ではNPB/MGを用いた性能評価を示す。

2. 再マッピング通信の生成

2.1 再マッピング通信の概要

初めに，簡単な例を用いて再マッピングを利用した通信(**再マッピング通信**)の概念を説明する。

なお本報告では，配列マッピングのパターンとしては HPF2.0 の基本仕様で規定されているものを対象とする[8]。また，計算マッピングは，代入文左辺の配列参照の割り当て先プロセッサが計算を実行する，いわゆる Owner-Computes Rule にしたがうものとする。したがって，通信が必要となるのは右辺の参照のみということになる。

[例 1]

```
   real a(0:3,0:3), b(0:3,0:3)
!HPF$ processors p(0:1,0:3)
!HPF$ distribute (block,block) onto p :: a,b
   do i1 = 0,3
     do i2 = 0,2
       a(i1,i2) = b(i2+1,3)
     enddo
   enddo
```

このコードにおいて，2 次元配列 a，b は共に，図 1(a)に示すように2次元のプロセッサ構成pの上にマッピングされている。マス目内の(0,0)などは配列要素のインデックスである。また，このループの計算マッピングは，Owner-Compute Rule に従うと，左辺配列参照a(i1,i2)に合わせて図 1(b)に示すようになる。マス目内の(0,0)などはループインデックス(i1,i2)の組を表す。この2重ループが実行される間に右辺配列参照 b(i2+1,3)によって参照される配列bの範囲はb(1:3,3)である。これを**参照リージョン**と呼ぶことにする。

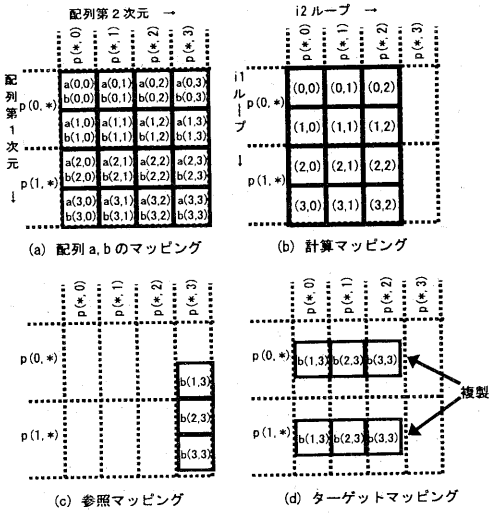

図 1 再マッピング通信の例

再マッピング通信の基本的な考え方は，配列の再マッピングライブラリを利用して，参照リージョンのマッピングを動的に変更し，計算マッピングと一致させるというものである。再マッピング通信の前後での参照リージョンのマッピングを，それぞれ，**参照マッピング**および**ターゲットマッピング**と呼ぶことにする。

例えば上記のコードでは，参照マッピングは，図 1(a)において参照リージョン b(1:3,3)のみに注目することにより，図 1(c)のようになる。一方，イタレーション(i1,i2)で参照される配列要素b(i2+1,3)が，そのイタレーションを実行するプロセッサに割り当てられるためには，参照リージョンのマッピングは図 1(d)のようになっていなければならない。すなわち，参照リージョンの第 1 次元がプロセッサ構成の第 2 次元に沿って整列し，プロセッサ構成の第 1 次元に関しては参照リージョンが複製されていなければならない。このマッピングがターゲットマッピングである。

2.2 マッピングの表現法

図 1に示されるように，本方式では，配列マッピング，計算マッピング，参照マッピング，ターゲットマッピングの 4 種類のマッピングを使用する。これらの各種マッピングの統一的な表現方法として，**マッピング標準形**を用いる[4]。マッピング標準形は表 1に示すパラメタから構成される。これは，HPF で配列マッピングを指定するための各種パラメタの中から，本質的なものだけを抽出して整理したものである。また，表 2，表 3に proc_axis_type および proc_axis_info パラメタの意味を示す。計算マッピングを表現する場合は，表中の「配列」を文の「インスタンス空間」と読み替える。

なおマッピング標準形は，分散形式が block か cyclic かを表す明示的パラメタを含まない。これは blocksize などの他のパラメタから求められるからである。

2.3 再マッピングの決定アルゴリズム

上記マッピング標準形を用いて，一般の多重ループ内の配列参照に対する再マッピング決定アルゴリズムを定式化する。なお，本報告では計算マッピングが与えられていることを前提とする。計算マッピングの決定方法自体については文献[4]に述べられている。

本アルゴリズムの入出力は以下である。

[入力]
- ループ内の配列参照
- その配列のマッピング
- その配列参照を含む文の計算マッピング

[出力]
- 参照リージョン
- 参照マッピング
- ターゲットマッピング

本アルゴリズムは，これら3種類の出力を順に決定する3段階から構成される。各段階の詳細は付録に示すこととし，ここでは，全体の考え方について述べる。

表 1 マッピング標準形

(a) プロセッサ構成の形状		
	proc_rank	プロセッサ構成の次元数。
	proc_size	プロセッサ構成の各次元の寸法(次元毎に1つ)。
(b) 配列の形状		
	rank	配列の次元数。
	size	配列の各次元の寸法(次元毎に1つ)。
(c) プロセッサ次元毎マッピング情報 (各次元毎に1セットの情報を持つ)		
	proc_axis_type	NORMAL, REPLICATED, SINGLE のいずれかの値。意味は表 2を参照
	proc_axis_info	proc_axis_type の値により，表 3の意味を持つ。
(d) 配列次元毎マッピング情報 *) (各次元毎に1セットの情報を持つ)		
	is_collapsed	当該次元が分散されていなければ TRUE，分散されていれば FALSE。
	axis_map	当該次元のマッピング先であるプロセッサ次元。
	align_lb	当該次元の最初の要素が整列するテンプレート要素のインデックス。ただしテンプレートの下限を0とする。
	align_stride	当該次元のテンプレートへの整列ストライド。
	blocksize	当該次元に対するテンプレート次元の分散ブロックサイズ。

*) is_collapsed が TRUE のときは(d)に分類される他の情報は使用しない。

表 2 proc_axis_type の値と意味

値	意味
NORMAL	当該プロセッサ次元に沿って，配列のある次元が分散されている。
REPLICATED	当該プロセッサ次元に沿って，配列が複製マッピングされている。
SINGLE	当該プロセッサ次元上の単一のプロセッサに，配列がシングルマッピングされている。

表 3 proc_axis_info の意味

proc_axis_type の値	proc_axis_info の意味
NORMAL	対応する配列次元。
REPLICATED	使用せず。
SINGLE	配列を持つプロセッサのインデックス。ただし下限を0とする。

全体を通しての基本的な戦略は，以下の通りである。

(a) 添字がループ制御変数の一次式または定数になっている配列次元については，添字に基づいて，参照リージョン，参照マッピング，およびターゲットマッピングを正確に求める。

(b) そうでない配列次元については，配列の宣言範囲全体を参照リージョンと見なし，ターゲットマッピングは非分散(is_collapsed=TRUE)とする。また，(a)で配列次元と対応づけられなかったプロセッサ次元については，ターゲットマッピングにおいて，そのプロセッサ次元に沿って配列を複製する(proc_axis_type = REPLICATED)。

これは，以下の考え方に基づいている。原理的には，すべての次元について(b)を適用してしまえば，ターゲットマッピングにおいて全プロセッサ上に配列全体が複製され，いずれのプロセッサにおいても任意の配列要素が参照できることになる。しかし，それでは必要メモリ量やデータ転送量が非常に大きくなってしまう。そこで，(a)のように配列添字が一次式や定数の場合は，できるだけ参照リージョンやマッピングを正確に求めて，本当に必要な配列要素だけを転送しようというのが，本アルゴリズムの考え方である。

この考え方に基づいて詳細化したアルゴリズムを付録に示す。これにより，任意の配列添字を含むループに対して再マッピング通信を生成できる。

3. 特定パターン通信の生成

再マッピング通信は，任意の配列添字に対して適用可能である。しかし参照リージョン全体のデータを移動することになるので，特にNPB/MGのような隣接参照型のプログラムでは効率が悪い。そこで，実プログラムに良く現れる特定のパターンについては，そのパターンに特化した，より高速な通信を用いる必要がある[3]。本章では，そのような特定パターン通信として，シャドウ通信および1対1通信について述べる。

3.1 シャドウ通信の生成

実プログラムに良く現れる隣接参照型のプログラムでは，シャドウ領域[8]を受信領域として用いることによって，効率的な通信が実現可能である。これを**シャドウ通信**と呼ぶことにする。本方式では，参照マッピングとターゲットマッピングとの間で次の条件が満たされたときに，再マッピング通信の代わりにシャドウ通信を生成する。

(a) align_lb 以外のパラメタがすべて一致する。

(b) 配列の各次元毎に，(align_lb の差)/(配列マッピングの align_stride)が，シャドウ領域に収まる。厳密には，

lbr: 参照マッピングの align_lb
lbt: ターゲットマッピングの align_lb
sta: 配列マッピングの align_stride
lshadow: 配列の下側シャドウ幅

ushadow: 配列の上側シャドウ幅

と定義したときに,

$-lshadow \leqq offset \leqq ushadow$ (式1)

ただし, $offset = (lbr - lbt)/sta$

が成立する。ただし除算は数学的な意味の実数除算である。

転送する要素数(各次元毎)は,

offset が正のとき, 下側シャドウに⌈*offset*⌉個

offset が負のとき, 上側シャドウに⌈-*offset*⌉個

とする。これを**転送シャドウ幅**と呼ぶことにする。ここで⌈ ⌉は整数への切り上げを表す。なお以下では両側の転送シャドウ幅をまとめて,

(下側の幅 : 上側の幅)

のように記述する。

[例 2]

NPB/MG には, 図 2のようなコードが含まれる。このコードにおいて, 3 行目の align 指示文により, 配列 u と配列 z の要素は図 3に示すように 1 個おきに対応している(1次元のみを示している)。例えば u(4)の値を計算するために, z(2)と z(3)の値が用いられる。

```
  real*8 z(130,130,130),u(258,258,258)
!HPF$ processors p(4,4)
!HPF$ align z(i,j,k) with u(2*i-1,2*j-1,2*k-1)
!HPF$ distribute (*,block,block) onto p ::u
!HPF$ shadow(1:1,1:1,1:1) :: z, u
  do i3 = 1,129
    do i2 = 1,129
      do i1 = 1,129
        u(2*i1-1,2*i2-1,2*i3) =
 >          z(i1   ,i2,i3+1)+z(i1  ,i2,i3  )
 >        + z(i1+1,i2,i3+1)+z(i1+1,i2,i3  )
      enddo
    enddo
  enddo
```

図 2 NPB/MG からのコード片

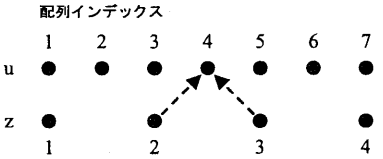

図 3 MG における配列の整列関係

ループ内の代入文の右辺の z(i1,i2,i3+1)に対して付録のアルゴリズムにしたがって参照マッピングとターゲットマッピングを決定すると, 配列第3次元のalign_1b以外のパラメタはすべて一致し, 配列第3次元については, *lbr*=2, *lbt*=1 となる。同次元について, *sta*=2, *lshadow*=1, *ushadow*=1 であるから, 上記の(式1)は $-1 \leqq -0.5 \leqq 1$ (*offset* =-0.5)となり成立する。したがって, 第3次元について転送シャドウ幅が (0:1)であるようなシャドウ通信を生成する。同様に右辺のz(i1,i2,i3)に対しては, 転送シャドウ幅が(1:0)であるようなシャドウ通信を生成する。右辺の他の 2 個の参照についても同様である。

3.2 シャドウ通信の融合

2 個のシャドウ通信の通信範囲に重なりがあるとき, それらを融合して 1 個のシャドウ通信とした方が効率が良い。本方式では, 同一配列に対する 2 個のシャドウ通信が以下の条件をすべて満たすときにそれらを融合する。

(a) プログラム内の位置が等しい。

(b) 高々一つの配列次元を除いて, 各配列次元に対して(分散/非分散に関わらず), 参照リージョンの当該次元の三つ組が一致し, 転送シャドウ幅も一致する。

(c) 上の条件(b)を充たさない唯一の次元について, 以下のいずれが成り立つ。

(c1) 一方の参照リージョン三つ組が他方のそれを含む。

(c2) 両者の参照リージョン三つ組に重なりがある。

この条件を満たす2個のシャドウ通信を融合し, 1個のシャドウ通信に置き換える。このとき, 上記の条件(c)に該当する次元に対して, 新たな参照リージョンは両者の元の参照リージョンの和集合とする。また転送シャドウ幅は, 上側下側それぞれにつき, 両者の元の転送シャドウ幅のうち大きい方とする。

例えば, 図 2の右辺の z(i1,i2,i3+1)に対するシャドウ通信は,

参照リージョン : (1:129, 1:129, 2:130)

転送シャドウ幅 : (0:0, 0:0, 0:1)

であり, z(i1+1,i2,i3+1)に対しては,

参照リージョン : (2:130, 1:129, 2:130)

転送シャドウ幅 : (0:0, 0:0, 0:1)

である。これらのシャドウ通信は, 第 1 次元について条件(c2)を満たし, 他の次元について条件(b)を満たすので融合される。

なお, 条件(c)を満たす次元を一つだけしか許さない理由は, 複数の次元が条件(c)を満たす場合, 融合したことによって余分な通信(それまでは通信する必要のなかった配列要素が通信されること)が発生する可能性があるからである。例えば, 2 次元配列において, 縦(第 1 次元)方向のシャドウ通信と, 横(第 2 次元方向)のシャドウ通信を融合すると, 斜め方向の通信が余分に必要になってしまう。

3.3 1対1通信の生成

差分法のプログラムでは, 周期的境界条件を設定するために, 配列の一方の端の列をもう一方の端の列にコピーする処理がよく現れる。この処理は, 並列機においては, 両端のプロセッサ間の 1 対 1 の通信となる。

本方式では, 参照マッピングとターゲットマッピングとの間で次の条件が満たされたときに, 再マッピング通信の代わりに1対1通信を生成する。

(a) 少なくとも一つのプロセッサ次元につき，proc_axis_type が両者ともに SINGLE であり，かつ，proc_axis_info(マッピング先プロセッサ)が一致しない。
(b) その他のパラメタがすべて一致する。

[例 3]

NPB/MG には，図 4のようなコードが含まれる。ループ内の代入文の右辺のu(i1,i2,255)に対して，付録のアルゴリズムにしたがって参照マッピングとターゲットマッピングを決定すると，プロセッサ第 2 次元において， proc_axis_type が共に SINGLE, proc_axis_info が参照マッピングでは 3，ターゲットマッピングでは 0 となる。他のパラメタはすべて一致する。したがって上記の条件が満たされ，コンパイラは 1 対 1 通信を生成する。

```
!HPF$ processors p(4,4)
!HPF$ distribute (block,block) onto p ::u
      do i2=1,256
        do i1=1,256
          u(i1,i2,1) = u(i1,i2,255)
        enddo
      enddo
```

図 4 NPB/MG における 1 対 1 通信

.4. 性能評価

提案方式のプロトタイプを我々の開発した HPF コンパイラ[5]上に実装し，実機で NPB/MG の性能評価を行った。使用したマシンは SR2201 である。HPF 版のソースプログラムは，逐次版である NPB2.3-serial[9]をベースにして作成した。

この HPF 版を，以下の 4 通りのオプションでコンパイルして実行した。

remap: 全通信に対して再マッピング通信のみを生成
shadow: シャドウ通信を生成
merge: シャドウ通信の融合を実施
1to1: 1 対 1 通信を生成

また，比較の対象として，MPI を用いた人手並列化版である NPB2.3β についても測定を行った。プロセッサ数は 16，問題サイズは Class A とした。

オプション	実行時間 (sec)	対 MPI 版 実行時間比
HPF remap	no mem.	no mem.
shadow	5.62	1.42
shadow + merge	5.19	1.31
shadow + merge + 1to1	5.14	1.29
MPI (NPB2.3β)	3.97	1.00

表 4 MG の性能

表 4に HPF 版の実行時間，および MPI 版に対する実行時間比を示す。remap の場合は，受信領域として，巨大なテンポラリ配列が何面も必要となるため，メモリ不足(no mem.)で測定できなかった。shadow 以降，特定パターン通信による最適化を強化するにつれて実行時間が短縮されていき，1to1 までの全最適化を行うことにより，MPI 版の 1.3 倍以内に収まった。各最適化の効果を比較すると，シャドウ通信生成の効果が最も大きく，1 対 1 通信生成の効果は比較的小さかったことが分かる。

これまでに知られている報告ではいずれも，MG の HPF 版は MPI 版に比べて数倍の実行時間を要しており[7][8]，今回 1.3 倍以内を達成したことにより本方式の有効性が示されたと考える。

5. おわりに

分散メモリ向けデータ並列言語のコンパイラにおける，通信生成方式について述べた。本方式は，再マッピング利用方式をベースとして，その一般化および最適化強化を行っている。これにより，様々な配列添字を持つループへの適用性を向上させることができる。

本方式のプロトタイプを我々の開発した HPF コンパイラ上に実装し，マルチグリッド法のプログラムである NPB/MG に適用して評価を行った。本方式により，HPF 版の実行時間を MPI を用いた人手並列化版の 1.3 倍以内に収めることができた。

参考文献

[1] C. Koelbel, "Compile-Time Generation of Regular Communications Patterns," Supercomputing'91, pp. 101-110.

[2] V. Adve and J. Mellor-Crummey, "Using Integer Sets for Data-Parallel Program Analysis and Optimization," SIGPLAN '98 Conf. on Programming Language Design and Implementation, pp. 186-198.

[3] 蒲池恒彦，草野和寛，末広謙二，妹尾義樹，田村正典，左近彰一，"HPF 処理系の実現と評価，" 情報処理学会論文誌，Vol. 37, No. 7, pp. 1255-1264, 1996.

[4] 太田寛，西谷康仁，"データ並列言語における多重ループの計算分散方式，" JSPP'99, pp. 79-86.

[5] 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.

[6] S. Saini and D. H. Bailey, "NAS Parallel Benchmark (Version 1.0) Results 11-96," Report NAS-96-018, NASA Ames Research Center, 1996.

[7] M. Frumkin, H. Jin and J. Yan, "Implementation of NAS Parallel Benchmarks in High Performance Fortran," IPPS'98.

[8] High Performance Fortran Forum, "High Performance Fortran Language Specification Version 2.0", Rice University, 1997.

[9] The NAS Parallel Benchmarks, http://www.nas.nasa.gov/Software/NPB/

付録

2.3節の再マッピング決定アルゴリズムの各段階の詳細について述べる。

第1段階　参照リージョン決定

[入力]

ループ内の代入文右辺の一つの配列参照。

[出力]

その配列参照に対する参照リージョン。これは，各次元 *da* に対する(下限:上限:増分)の三つ組によって表現する。これを triplet(*da*)と書く，

[方法] 以下の手順による。

```
1: for(各配列次元 da につき) do
2:   if(添字が F*I+D の形, かつ, I ループの下限,
       上限, 増分が定数) then
       /* ここで I はあるループの制御変数,
          F,D は定数, F≠0 */
3:     triplet(da) =
           (F*L_lp+D : F*U_lp+D : F*S_lp);
       /* ここで, L_lp, U_lp, S_lp はループの
          下限, 上限, 増分 */
4:   else if(添字が定数 C) then
5:     triplet(da) = (C:C:1);
6:   else
7:     triplet(da) = (L_ary,:U_ary:1);
       /* ここで, L_ary, U_ary は配列の当該
          次元の宣言範囲の下限, 上限 */
8:   endif
9: endfor
```

図 5 参照リージョン決定手順

第2段階　参照マッピング決定

[入力]

右辺の一つの配列参照に対する，参照リージョン，および，その配列のマッピング。

[出力]

その配列参照に対する参照マッピング。

[方法]

配列のマッピング標準形を次の手順に従って修正したものを，参照マッピングとする。図中で，*da* を添字とするパラメタは，配列の第 *da* 次元に対するものであり，*dp* を添字とするパラメタは，プロセッサ第 *dp* 次元に対するものである。

```
 1: for(各配列次元 da につき) do
 2:   if(triplet(da)が宣言範囲全体) continue;
 3:   size(da) = triplet(da)の長さ;
 4:   if(分散次元でない) continue;
 5:   if(triplet(da)が(C:C:1)の形) then
        /* 非分散, シングルとする */
 6:     is_collapsed(da) = TRUE;
 7:     dp = axis_map(da);
          /* dp はマッピング先プロセッサ次元 */
 8:     proc_axis_type(dp) = SINGLE;
 9:     proc_axis_info(dp) =
          添字 C に対するプロセッサインデックス;
10:   else
        /* 整列パラメタを変更する。以下で,
           L_reg, S_reg は triplet(da)の下限,
           増分。L_ary は配列の宣言下限。*/
11:     align_lb(da) +=
            (L_reg - L_ary)*align_stride(da);
12:     align_stride(da) *= S_reg;
13:   endif
14: endfor
```

図 6 参照マッピング決定手順

第3段階　ターゲットマッピング決定

[入力]

右辺の一つの配列参照，その参照マッピング，および，その配列参照を含む文のインスタンス空間の計算マッピング。

[出力]

その配列参照に対するターゲットマッピング。

[方法]

ターゲットマッピングを構成する各パラメタのうち，プロセッサ形状(proc_rank, proc_size)は，計算マッピングのプロセッサ形状と同一にする。また，配列形状(rank, size)は参照マッピングの配列形状と同一にする。その他のパラメタは次の手順で決定する。図中で，is_collapsedc(ds)のように下付き添字 'c' を持つパラメタは計算マッピングに対するものであり，そうでないパラメタはターゲットマッピングに対するものである。

```
 1: for(インスタンス空間の各次元 ds につき) do
 2:   if(次元 ds に対応するループの下限, 上限,
        増分が定数でない) continue;
 3:   if(添字が F*I+D の形であるような配列次元が
        ある) then
        /* ここで I はループの制御変数,
           F,D は定数, F≠0 */
 4:     該当する配列次元から1個を選ぶ
        (これを da と書く);
 5:     配列次元 da のマッピング情報を,
        計算マッピングの次元 ds のマッピング
        情報と同一にする */
10:   endif
11: endfor
12: for(マッピング未定の各配列次元 da につき) do
13:   is_collapsed(da) = TRUE;
14: endfor
15: for(各プロセッサ次元 dp につき) do
16:   if(axis_map(da) == dp であるような配列次元
        da が既にある) then
17:     proc_axis_type(dp)=NORMAL;
18:     proc_axis_info(dp)=da;
19:   else if(計算マッピングにおいて,
          (proc_axie_type_c(dp) == SINGLE
          かつ proc_axis_info_c(dp)が定数)) then
20:     proc_axis_type(dp) = SINGLE;
21:     proc_axis_info(dp) = proc_axis_info_c(dp);
22:   else
23:     proc_axis_type(dp)=REPLICATED;
24:   endif
25: endfor
```

図 7 ターゲットマッピング決定手順