データ並列言語の通信生成方式とマルチグリッド法での最適化評価

太田 寛[†] 西谷康仁[†]

HPF（High Performance Fortran）など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やシャドウ通信の融合などの最適化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 上で性能測定した．測定結果に基づいて各種の通信最適化の効果を定量的に明らかにする．本方式により，MG の HPF 版の実行時間を MPI 版（NPB2.3 β）の 1.18 倍まで短縮できることが示された．

Communication Generation for Data-parallel Languages and Evaluation of Optimizations with the Multigrid Method

Hiroshi Ohta[†] and Yasunori Nishitani[†]

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 optimization 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 performance evaluation on SR2201 shows the quantitative effectiveness of the communication optimizations. It is also shown that the execution time of the HPF version is reduced to 1.18 times as much as that of the MPI version (NPB2.3 β).

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）[13] など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),2)] 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[3)]，および，実行時の配列再マッピングを利用する方法[5),6)] 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よいかが明確ではなかった．また同文献では，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，その適用条件など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）[14] の MG（マルチグリッド法）のような複雑なストライド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．

† 株式会社日立製作所
Hitachi, Ltd.

以下，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．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．4 章では NPB/MG を用いた性能評価を示す．5 章で関連研究との比較を行い，6 章で全体をまとめる．

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．

なお本稿では，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[13]．また，計算マッピングは，代入文左辺の配列参照の割当て先プロセッサが計算を実行する，いわゆる Owner-Computes Rule に従うものとする．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．

図 1 のコードにおいて，2 次元配列 a，b はともに，**図 2** (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-Computes Rule に従うと，左辺配列参照 a(i1,i2) に合わせて図 2 (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の組 (i1,i2) 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．

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．

たとえば上記のコードでは，参照マッピングは，図 2 (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 2 (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 2 (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．

再マッピング通信におけるコンパイラ生成コードの

```
      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
```

図 1 例題コード
Fig. 1 Code example.

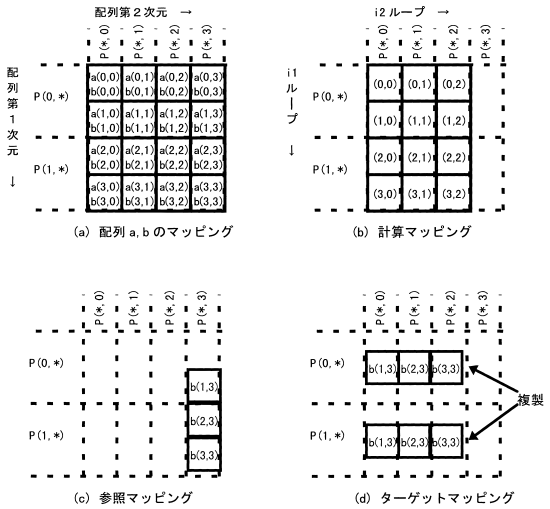

図 2 再マッピング通信の例
Fig. 2 Example of remapping communication.

処理概要は以下のようになる．再マッピング通信とはいっても，元の配列自体のマッピングを変更するわけではなく，ターゲットマッピングに相当する形状のテンポラリ配列を各プロセッサで確保し，元の配列の参照リージョンの内容をそのテンポラリ配列に転送する．この転送を実現するために再マッピングライブラリを用いる．またループ内では，元の配列を参照する代わりに上記テンポラリ配列を参照する．例として，図 1，図 2 の場合は，各プロセッサでテンポラリ配列 tmp(1,0:2) を確保し，b から tmp へ再マッピングライブラリによる転送を行う．そして，ループ内では配列参照 b(i2+1,3) の代わりに tmp(1,i2) を参照する．

2.2 マッピングの表現法

図 2 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[8]．マッピング標準形は**表 1** に示すパラメータから構成される．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ータの中から，本

表 1 マッピング標準形
Table 1 Mapping normal form.

(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（次元ごとに 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（次元ごとに 1 つ）
(c) プロセッサ次元ごとマッピング情報（各次元ごとに 1 セット）	
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) 配列次元ごとマッピング情報 *（各次元ごとに 1 セット）	
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) に分類される他のパラメータは使用しない．

表 2 proc_axis_type の内容
Table 2 Description of proc_axis_type.

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 の内容
Table 3 Description of proc_axis_info.

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

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 パラメータの意味を示す．計算マッピングを表現する場合は，表中の「配列」をループの「イタレーション空間」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ータを含まない．これは block-size などの他のパラメータから求められるからである．

例として，図 2 (a) の配列 a，b のマッピングに対するマッピング標準形は，ともに**図 3** に示すようにな

proc_rank	2	
proc_size	2	4
proc_axis_type	NORMAL	NORMAL
proc_axis_info	1	2
rank	2	
size	4	4
is_collapsed	FALSE	FALSE
axis_map	1	2
align_lb	0	0
align_stride	1	1
blocksize	2	1

図 3 配列 a，b のマッピング標準形
Fig. 3 Mapping normal form for the arrays a and b.

proc_rank	2	
proc_size	2	4
proc_axis_type	NORMAL	NORMAL
proc_axis_info	1	2
rank	2	
size	4	3
is_collapsed	FALSE	FALSE
axis_map	1	2
align_lb	0	0
align_stride	1	1
blocksize	2	1

図 4 計算のマッピング標準形
Fig. 4 Mapping normal form for the computation.

る．プロセッサ構成や配列の各次元のパラメータは第 1 次元を先頭にして横に並べてある．また，図 2 (b) の計算マッピングに対するマッピング標準形を**図 4** に示す（この例の場合，図 3 の配列マッピングとの違いは第 2 次元の size パラメータが 3 になっている点だけである）．

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，本研究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 8) 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．

全体を通しての基本的な戦略は，以下のとおりである．

(1) 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．

(2) 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，(1) 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 (2) 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，(1) 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．

2.4 適 用 例

付録のアルゴリズムを例 1 のループに適用した場合の処理の進行を以下に示す．入力は，右辺の配列参照 b(i2+1,3)，配列 b のマッピング標準形（図 3），および，計算のマッピング標準形（図 4）である．

参照リージョンは次のように決定される．配列参照 b(i2+1,3) を入力とする．図 11（付録参照）の手順において，配列第 1 次元については，添字が i2+1 なので手順の 2，3 行目が適用され，三つ組は (1:3:1) となる．配列第 2 次元については，添字が定数 3 なので手順の 4，5 行目が適用され，三つ組は (3:3:1) となる．結局，参照リージョンは (1:3:1,3:3:1) となる．

参照マッピングは次のように決定される．配列 b のマッピング標準形（図 3）を元にして，図 12（付録参照）の手順に従って修正が行われる．まず，配列の両方の次元について，手順の 3 行目で size が変更される．また，配列第 1 次元については，手順の 10 行目から 12 行目が適用されて，整列パラメータが変更される．配列第 2 次元については，手順の 5 行目から 9 行目が適用されて配列次元は非分散となり，また，対応するプロセッサ第 2 次元はシングルマッピングとなる．この結果として得られる参照マッピング標準形を**図 5** に示す．

proc_rank	2	
proc_size	2	4
proc_axis_type	NORMAL	SINGLE
proc_axis_info	1	3
rank	2	
size	3	1
is_collapsed	FALSE	TRUE
axis_map	1	-
align_lb	1	-
align_stride	1	-
blocksize	2	-

図 5 参照マッピング標準形
Fig. 5 Mapping normal form for the reference mapping.

proc_rank	2	
proc_size	2	4
proc_axis_type	REPLICATED	NORMAL
proc_axis_info	-	1
rank	2	
size	3	1
is_collapsed	FALSE	TRUE
axis_map	2	-
align_lb	0	-
align_stride	1	-
blocksize	1	-

図 6 ターゲットマッピング標準形
Fig. 6 Mapping normal form for the target mapping.

ターゲットマッピングは次のように決定される．プロセッサ形状（proc_rank，proc_size）は計算マッピングから，配列形状（rank，size）は参照マッピングからコピーされる．図 13（付録参照）の最初の for ループにおいて，イタレーション空間の第 2 次元（i2 ループ）について手順の 3 行目の条件が成立する（配列第 1 次元の添字が i2+1）．そこで，配列第 1 次元のマッピングはイタレーション空間の第 2 次元のマッピングと同一となる．配列第 2 次元については，手順の 9 行目において非分散となる．プロセッサ第 1 次元については，19 行目が適用されて REPLICATED となり，プロセッサ第 2 次元については，12 行目から 14 行目が適用されて NORMAL となる．この結果として得られるターゲットマッピング標準形を**図 6** に示す．

2.5 通信不要判定

データ並列言語では，通信生成に先立って，そもそも通信が必要か否かの判定が非常に重要である．本方式では，これは参照マッピングとターゲットマッピングの比較によって簡単に判定できる．双方のマッピング標準形の対応するパラメータどうしがすべて一致

するならば通信は不要である．また完全に一致しなくても，参照マッピングに REPLICATED（複製）や COLLAPSED（非分散）を含む場合は，通信不要な場合がある．具体的には，参照マッピングとターゲットマッピングの間で次の条件がすべて満たされたときに，通信不要と判定する．

(1) プロセッサ構成の形状（proc_rank，proc_size）が一致する．
(2) プロセッサ次元ごと情報（proc_axis_type, proc_axis_info）が，各次元ごとに次のいずれかを満たしている．
  (a) 一致する．または，
  (b) 参照マッピングにおいて proc_axis_type の値が REPLICATED である．すなわち，当該プロセッサ次元に沿って複製されている．
(3) 配列次元ごと情報（is_collapsed，axis_map，align_ lb，align_stride，blocksize）が，各次元ごとに次のいずれかを満たしている．
  (a) 一致する．または，
  (b) 参照マッピングにおいて is_collapsed の値が TRUE である．すなわち，当該配列次元は分散されていない．

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，ターゲットマッピングに相当する大きさのテンポラリ配列を確保して，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，実プログラムによ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．本章では，そのような特定パターン通信として，シャドウ通信におよび 1 対 1 通信について述べる．

3.1 シャドウ通信の生成

実プログラムによく現れる隣接参照型のプログラムでは，シャドウ領域[13] 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．

(1) align_lb 以外のパラメータがすべて一致する．
(2) 配列の各次元ごとに，(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（シャドウ領域の幅は shadow 指示文によって与えられているものとする）．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，

$$-lshadow \leq offset \leq ushadow \qquad (1)$$

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（各次元ごと）は，

$offset$ が正のとき，上側シャドウに $\lceil offset \rceil$ 個
$offset$ が負のとき，下側シャドウに $\lceil -offset \rceil$ 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 $\lceil\ \rceil$ 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，

(下側幅 ： 上側幅)

のように記述する．

[例]

NPB/MG には，**図 7** 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 8** 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）．たとえば 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．

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従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_lb 以外のパラメータはすべて一致し，配列第 3 次元については，$lbr = 2$，$lbt = 1$ となる．この次元について，align 指示文と shadow 指示文より $sta = 2$，$lshadow = 1$，$ushadow = 1$ であるから，上記の式 (1) は $-1 \leq 0.5 \leq 1$ ($offset = 0.5$) 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． □

シャドウ通信生成による効果を以下にまとめておく．

再マッピング通信を用いた場合は，2.1 節で述べたように，参照リージョン全体の内容をテンポラリ配列に移動する必要がある．たとえば図 7 では，z(i1,i2,i3+1) に対する参照リージョンは z(1:129,1:129,2:130) であり，ほぼ配列全体に等しい．このデータ移動の内訳と

```
     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 7 NPB/MG からのコード片
Fig. 7 Code fragment of NPB/MG.

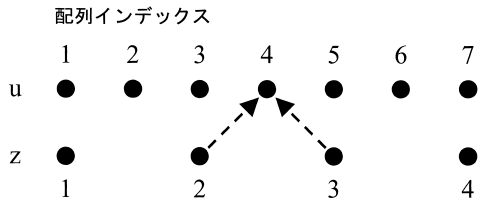

図 8 MG における配列の整列関係
Fig. 8 Alignment of arrays in MG.

しては，プロセッサ間で転送されるデータはプロセッサ境界付近のデータのみであり，残りの大部分はプロセッサ内のメモリコピーである．配列の 1 次元の寸法を N，プロセッサ数固定として考えると，プロセッサ間転送データ量が $O(N^2)$ であるのに対して，メモリコピーのデータ量は $O(N^3)$ であり非常に大きい．

一方，シャドウ通信を用いた場合は，データ移動はシャドウ領域へのデータ転送のみである．プロセッサ間で転送されるデータ量としては再マッピング通信の場合と等しいが，プロセッサ内でのメモリコピーが不要な分，実行時間が大幅に短縮する．また，テンポラリ配列の確保や解放のオーバヘッドもない．さらに，シャドウ領域内の配列要素をアクセスするためのは添字は，ローカルな（本来自プロセッサに割り当てられた）配列要素とまったく同じでよく，余分な添字変換オーバヘッドが生じることもない．具体的には，今の場合はブロック分散なので，元の添字から定数を差し引くだけでよい．

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．これは，Fortran D [2] などで行われている message coalescing に相当するものであるが，これまで述べてきたシャドウ通信の枠組みの中では以下のように定式化できる．

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．

(1) プログラム内での生成位置が等しい．すなわち，同一ループの外側に生成される．
(2) たかだか 1 つの配列次元を除いて，各配列次元に対して（分散/非分散にかかわらず），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．
(3) 上の条件 (2) を満たさない唯一の次元について，以下のいずれが成り立つ．
    (a) 一方の参照リージョン三つ組が他方のそれを含む．
    (b) 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 (3) 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．

たとえば，図 7 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 (3) (b) を満たし，他の次元について条件 (2) を満たすので融合される．

なお，条件 (3) を満たす次元を 1 つだけしか許さない理由は，複数の次元が条件 (3) 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．たとえば，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．

```
!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 9 NPB/MG における境界条件設定コード
Fig. 9 Code fragment for boundary condition in NPB/MG.

(1) 少なくとも 1 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．
(2) その他のパラメータがすべて一致する．

[例]

NPB/MG には，**図 9** 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従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 がともに 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．他のパラメー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． □

1 対 1 通信を生成することの効果を以下に述べる．1 対 1 通信では，受信領域としてテンポラリ領域を確保する点や，参照リージョン全体の内容をそのテンポラリ領域に移動する点などは，再マッピング通信と同様である．しかし，転送すべき配列範囲の計算や転送データのバッファリング処理などを 1 対 1 通信に特化して最適化できるので，一般的な再マッピングライブラリを用いた場合よりも高速に実行できる．

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[9] 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[14] をベースにして作成した．NPB2.3-serial からの主な変更点は以下である．

- マルチグリッド法ではサイズの異なる複数の 3 次元配列を使用する（図 7，図 8 参照）．1 次元あたりの寸法は $2^L + 2$（$L = 1, 2, \ldots, L_{\max}$）となっており，$L_{\max}$ の値は問題サイズが Class A および B の場合は 8，Class C の場合は 9 である．NPB2.3-serial では，メインプログラムにおいてはこれらの配列を巨大な 1 次元の作業配列として宣言しており，サブルーチン側においてはこの作業配列の部分領域を 3 次元配列として受けている．このように手続きにまたがって配列の宣言形状が異なっていると，HPF で適切なデータ分散指示文を記述するのが困難である．そこで，我々の HPF 版ではメインプログラムでの配列宣言を各 3 次元配列ごとの別々の宣言に変更した．
- 図 9 に示した境界条件設定コードは，NPB2.3-serial においては，右辺の配列要素を 1 次元の作業配列にバッファリングしてから左辺に代入するという処理になっている（おそらく MPI 版でのメッセージ通信コードの名残り）．しかしこのままでは，1 次元作業配列に適切なデータ分散指示を与えたり，HPF コンパイラが効率的なコードを生成したりするのが困難である．そこで，我々の HPF 版では図 9 に示すような単純なコードに変更した．これは，NASA 提供の旧バージョンである NPB1.0 で用いられていたものとほぼ同じである．
- データ分散指示文としては，図 7 に示すように，align 指示文によってサイズの異なる配列を整列させ，distribute 指示文によって第 2，第 3 次元をブロック分散した．また，shadow 指示文によって上下ともに幅 1 のシャドウ領域を明示的に指定した（実際には shadow 指示文とほぼ同機能の独自指示文である overlap 指示文[9] を用いている）．

この HPF 版を，以下の 4 通りの通信生成方法でコンパイルして実行した．

remap：すべての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：さらにシャドウ通信の融合を実施
1to1：さらに 1 対 1 通信を生成

ここで shadow や 1to1 は，そのような特定パターン通信の生成条件が満たされる場合には，再マッピングの代わりに特定パターン通信を用いるという意味である．プロセッサ数は 16，32，64 台，問題サイズは Class A および B の場合を測定した．また，比較の対象として，MPI を用いた人手並列化版である NPB2.3 β（以下，MPI 版）についても測定を行った．

図 10 に実行時間を示す．また**表 4** に，HPF 版と MPI 版との実行時間比を示す．

remap 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，16 台のときはメモリ不足で測定できなかった．32 台や 64 台のときは測定できたが，

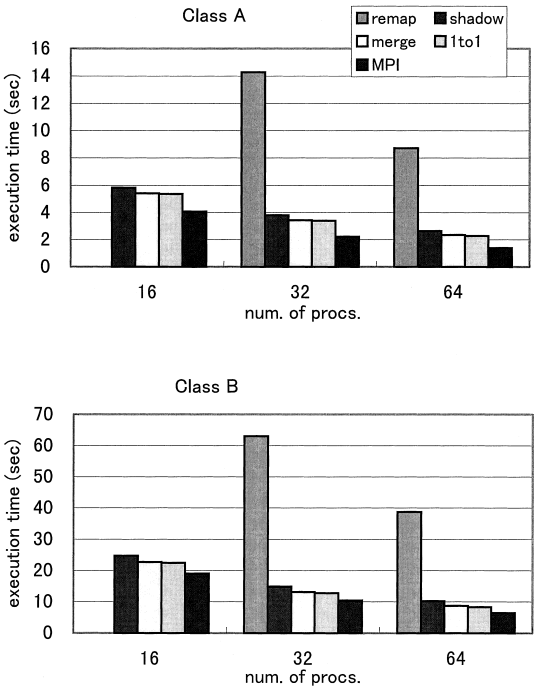

図 10 MG の実行時間
Fig. 10 Execution time of MG.

表 4 対 MPI 実行時間比
Table 4 Execution time ratio to MPI.

Optimization	class A			class B		
	num. of procs.			num. of procs.		
	16	32	64	16	32	64
remap	n.a.	6.43	6.31	n.a.	6.07	6.02
shadow	1.43	1.71	1.92	1.30	1.43	1.59
merge	1.33	1.55	1.70	1.19	1.26	1.35
1to1	1.31	1.53	1.64	1.18	1.23	1.29

通信データ量が大きいため，他の通信方法と比べて非常に実行時間が長い．

shadow で示されるように，シャドウ通信の生成によって実行時間は大幅に短縮される．これは 3.1 節で述べたように，再マッピング通信の場合は，参照リージョン全体の配列要素を受信領域のテンポラリ配列に移動しなければならないのに対して，シャドウ通信では，プロセッサ境界の配列要素のみを移動すればよいからである．表 4 より，最適化としてシャドウ通信の生成を行っただけでも，MPI 版の 2 倍以内の実行時間が達成できることが分かる．

merge で示されるように，シャドウ通信の融合を行うことによってさらに実行時間が短縮される．shadow と比べて 1〜2 割の改善となっており，シャドウ通信の生成自体よりは効果が小さいが，それでも最適化の効果としてはかなり大きいものである．なお融合されたシャドウ通信の数は，ループによって異なるが，おおむね 4〜16 個の通信が融合されている．

1to1 で示されるように，1 対 1 通信を生成することによってもさらに実行時間が短縮される．ただし，1 対 1 通信による改善は数%程度であり他の最適化に比べると比較的小さい．これには，主に 2 つの理由が考えられる．1 つは，1 対 1 通信が生成されるような条件の下では，参照リージョンは配列全体よりも次元数が低いのでデータ転送量が元々少ないことである．もう 1 つは，1 対 1 通信のデータ転送量は再マッピング通信と同じであり，1 対 1 通信の利点は，特化したライブラリを用いることによる転送前後のオーバヘッドの削減のみだからである．このように 1 対 1 通信の効果は比較的小さいが，一方で数%の改善は有意であるということができ，十分に価値のある最適化であると考えている．

表 4 に示されるように，1to1 までの全最適化を行った場合に，HPF 版と MPI 版との実行時間比は最小で 1.18 倍（16 台，Class B）であり，最大でも 1.64 倍（64 台，Class A）に収まっている．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[10),11)]，今回表 4 の実行時間比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．

なお筆者らの以前の報告[12)] では，MG の HPF 版/MPI 版の実行時間比として 1.7 倍を達成しているが，そこではストライド付きの整列関係を回避するために 3 次元格子点を 5 次元配列で表現するという不自然なソースコードを用いていた．一方今回の研究では 3 次元配列を用いた自然なソースコードを用いており，実行時間比もいっそう改善されている．

5. 関 連 研 究

本稿の冒頭でも述べたように，再マッピングを利用した通信生成手法は蒲池ら[5),6)] によって提案されている．また同著者らにより，基本的なシャドウ通信生成やその最適化についても示されている．本研究の基本的な考え方はこれらの文献と共通であるが，様々な配列添字を持つループに対する一般的な通信生成アルゴリズムを定式化し，特に，添字が一次式の場合にできるだけ正確なマッピングを求めるようにしたことが，本研究の特徴である．また，配列要素「半個分」のシフトに相当するシャドウ通信の生成や，1 対 1 通信の生成などの通信最適化も上記文献にない特徴である．これらの特徴により，両辺の配列の整列ストライドが

異なる場合にもシャドウ通信などの高速通信が生成可能となり，NPB/MG のようなマルチグリッド法プログラムの効率的実行を実現している．

他の従来研究の多くは，各プロセッサが通信すべき配列要素集合をコンパイル時に計算する方法を用いている[1),2)]．これらの方法には，配列宣言やループ範囲などにコンパイル時に確定しない変数が含まれていると，通信集合を十分な精度で計算するのが困難であるという問題があった．線形不等式系ソルバを用いて強力に通信集合の解析を行う方法も提案されているが，プロセッサ数か可変の場合に非線形式が現れて特殊な扱いが必要になるなどの問題があった[3)]．

また別の方法として，添字のパターンマッチングにより，シフトやブロードキャストなどの特定パターン通信を生成するものも知られている[4)]．しかしこの方法では，効率的通信を生成できる場合が単純な添字パターンのみに限られてしまう．

本研究のように再マッピングに基づいた方式では，コンパイル時には，プロセッサ全体から見た通信パターンを特定するパラメータ（マッピング標準形や転送シャドウ幅など）のみを求める．各プロセッサの通信集合は，実行時に変数の値が確定した状態で，ライブラリ内で計算される．その結果，より高精度に通信集合を計算できる機会が増える．また，マッピング標準形の利用によって複雑な添字でも標準化して扱うことができるため，単純な添字のパターンマッチングと比較して，コンパイル時の特定パターン通信認識が容易になるという利点もある．

小笠原ら[7)]は，通信パターン認識の一部を実行時に行う方法を提案している．しかし，基本的には文献 1) などと同様に，プロセッサ全体ではなく各プロセッサごとの通信集合を計算するという考え方に基づいているため，ストライドアクセスなどに対する通信パターン認識が困難になっているものと考えられる．

6. 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．また，ストライドアクセスパターンに対して，効率的なシャドウ通信が生成できる．

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 に適用して性能評価を行った．測定結果に基づき，本方式における各種通信最適化の効果を定量的に示した．これらの最適化により，従来は数倍もの差があった HPF 版と MPI 版の実行時間比を，本研究では 1.18 倍にまで縮めることができた．

今後の課題としては，まず初めに，さらに多くのベンチマークや実応用プログラムによる本方式の評価があげられる．その結果次第では，特定パターン通信の種類や最適化についていっそうの改善が必要となる可能性もある．次に，本研究で言及しなかった DOACROSS 型の通信について，本方式の枠組みに収められるように検討していきたい．また，本研究では Owner-Computes Rule を前提として右辺配列のみを通信対象としているが，左辺配列に対して通信を生成する場合には，複数のプロセッサ間での整合性問題に対処する必要がある．最後に，NPB/MG において，人手並列化版との差をさらに縮めるための性能分析と対策も進めていきたいと考えている．

謝辞 本研究の機会を与えてくださった（株）日立製作所の菊池純男氏，布広永示氏，また本方式の実装に関して多くのご助言をいただいた日立東北ソフトウェア（株）の黒澤隆氏に感謝いたします．本研究の一部は著者らが新情報処理開発機構の一員として行ったものである．

参 考 文 献

1) Koelbel, C.: Compile-Time Generation of Regular Communications Patterns, *Supercomputing'91*, pp.101–110 (1991).
2) Tseng, C.-W.: An Optimizing Fortran D Compiler for MIMD Distributed-Memory Machines, Ph.D. Thesis, Rice University (1993).
3) Adve, V. and Mellor-Crummey, J.: Using Integer Sets for Data-Parallel Program Analysis and Optimization, *SIGPLAN '98 Conf. on Programming Language Design and Implementation*, pp.186–198 (1998).
4) Bozkus, Z., Choudhary, A., Fox, G., Haupt, T., and Ranka, S.: Fortran 90D/HPF Compiler for Distributed Memory MIMD Computers: Design, Implementation and Performance Results, *Supercomputing '93* (1993).
5) 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).
6) Kamachi, T., Kusano, K., Suehiro, K., Seo, Y., Tamura, M. and Sakon, S.: Generating Realignment-Based Communication for HPF Programs, *IPPS '96*, pp.364–371 (1996).

7) 小笠原武史，小松秀昭：データ並列言語における集団通信の実行時認識手法，情報処理学会論文誌，Vol.40, No.11, pp.4117–4126 (1999).
8) 太田 寛，西谷康仁：データ並列言語における多重ループの統一的計算分散方式，情報処理学会論文誌，Vol.41, No.5, pp.1439–1447 (2000).
9) 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).
10) Saini, S. and Bailey, D.H.: NAS Parallel Benchmark (Version 1.0) Results 11-96, Report NAS-96-018, NASA Ames Research Center (1996).
11) Frumkin, M., Jin, H. and Yan, J.: Implementation of NAS Parallel Benchmarks in High Performance Fortran, *IPPS'99* (1999).
12) 太田 寛, 西谷康仁, 小林 篤, 布広永示：HPF 処理系 Parallel FORTRAN による NAS Parallel ベンチマークの並列化，情報処理学会論文誌, Vol.38, No.9, pp.1830–1839 (1997).
13) High Performance Fortran Forum: *High Performance Fortran Language Specification Version 2.0*, Rice University (1997).
14) NASA: 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 1 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] **図 11** の手順による．

第 2 段階 参照マッピング決定

[入力]

右辺の 1 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**図 12** 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ータは，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ータは，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 (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 11 参照リージョン決定手順
Fig. 11 Procedure to decide the reference region.

```
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
11:     /* 整列パラメータを変更する．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．*/
12:     align_lb(da) +=
          (L_reg - L_ary)*align_stride(da);
13:     align_stride(da) *= S_reg;
14:   endif
15: endfor
```

図 12 参照マッピング決定手順
Fig. 12 Procedure to decide the reference mapping.

第 3 段階 ターゲットマッピング決定

[入力]

右辺の 1 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ループのイタレーション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ータのうち，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，

```
1:  for (イタレーション空間の各次元 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;
6:    endif
7:  endfor
8:  for (マッピング未定の各配列次元 da につき) do
9:    is_collapsed(da) = TRUE;
10: endfor
11: for (各プロセッサ次元 dp につき) do
12:   if (axis_map(da) == dp であるような配列次元
        da が既にある) then
13:     proc_axis_type(dp)=NORMAL;
14:     proc_axis_info(dp)=da;
15:   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
16:     proc_axis_type(dp) = SINGLE;
17:     proc_axis_info(dp) = proc_axis_info_c(dp);
18:   else
19:     proc_axis_type(dp)=REPLICATED;
20:   endif
21: endfor
```

図 13 ターゲットマッピング決定手順
Fig. 13 Procedure to decide the target mapping.

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．その他のパラメータは**図 13** の手順で決定する．図中で，is_collapsed$_c$(ds) のように下付き添字 'c' を持つパラメータは計算マッピングに対するものであり，そうでないパラメータはターゲットマッピングに対するものである．

(平成 12 年 8 月 31 日受付)
(平成 13 年 3 月 9 日採録)

太田 寛（正会員）

1962 年生．1987 年東京大学大学院理学系研究科地球物理学専門課程修了．同年，(株）日立製作所入社．同社システム開発研究所を経て，現在，同社情報コンピュータグループ事業企画本部主任技師．入社以来，論理型言語の研究を経て，並列化コンパイラの研究に従事．並列処理ソフトウェア全般，並列アーキテクチャに興味を持つ．電子情報通信学会，ACM，IEEE 各会員．工学博士．

西谷 康仁（正会員）

1971 年生．1994 年東京大学工学部電子工学科卒業．同年，(株）日立製作所入社．現在，同社ソフトウェア事業部勤務．並列化コンパイラの研究開発に従事．