# HPF処理系の実現と評価

蒲池恒彦[†] 草野和寛[†] 末広謙二[†]
妹尾義樹[†] 田村正典[††] 左近彰一[††]

本論文では我々が開発したHigh Performance Fortran (HPF) のための言語処理系の実現方法について述べる．本処理系は，Fortran 77プログラムにHPF指示行を挿入したHPFプログラムを並列化し，実行時ライブラリ呼び出しを挿入したSPMD (Single Program Multiple Data) プログラムに変換するHPFコンパイラと，実行時にプロセッサ間通信などを制御する実行時ライブラリからなる．HPFでは，配列のマッピングを「整列 (alignment)」と「分散 (distribution)」の2レベルマッピングで実現するが，本処理系では，ループのイタレーションマッピングも配列同様2レベルマッピングで実現する．通信解析においては，この2レベルマッピングの機能，特に整列の機能を利用し，プロセッサ間通信を配列の再マッピングと見なした通信パターンの検出を行う．この再マッピングを基本とした通信制御方式により，配列のマッピング方法がコンパイル時に不明の場合でも，既知の場合と同じ方式で効率良く通信解析を行うことが可能となる．また，ベンチマークプログラムを用いたCenju-3での評価結果により本処理系の有効性を示す．

# Design and Implementation of an HPF Compiler and Its Performance Results

TSUNEHIKO KAMACHI,[†] KAZUHIRO KUSANO,[†] KENJI SUEHIRO,[†]
YOSHIKI SEO,[†] MASANORI TAMURA[††] and SHOICHI SAKON[††]

This paper presents the design and implementation of an HPF compilation system for distributed-memory parallel machines. We introduce the concept of an iteration template corresponding to an iteration space. Our HPF compiler performs the loop iteration mapping through the two-level mapping of the iteration template in the same way as the data mapping is performed in HPF. Making use of this unified mapping model of the data and the loops, communication for nonlocal accesses is handled based on data-realignment between the user-declared alignment and the optimal alignment, which ensures that only local accesses occur inside the loop. This strategy results in effective means of dealing with communication for arrays with undefined mapping, a simple manner for generating communication, and high portability of the HPF compiler. Experimental results on the NEC Cenju-3 distributed-memory parallel computer demonstrate the effectiveness of our approach.

## 1. はじめに

近年，分散メモリマシン用のデータ並列記述言語としてHigh Performance Fortran (HPF) が注目を集めている．HPFは，Fortran 90をベースとしてデータのマッピング方法を指示行で記述できるように拡張した言語で，HPF Forum (HPFF) で標準化作業が進められている[1]．現在，各国の研究機関やベンダで処理系の研究開発が進められており[2)~5)]，一部商用化されているものもあるが，まだ実用レベルには達していない．

我々が開発したHPF処理系[6),7)]は，Fortran 77プログラムにHPF指示行を挿入したHPFプログラムを，実行時ライブラリ呼び出しを挿入したSPMD (Single Program Multiple Data) プログラムに変換するHPFコンパイラと，実行時にプロセッサ間通信や並列実行制御などを制御する実行時ライブラリからなり，以下の特長を有する．

(1) 独自指示行によるHPFの拡張：
HPF指示行に加えて独自に指示行を用意し，プログラマがより効率の良いプログラムを開発できるようにした．特に，計算処理マッピングのひとつである，イタレーションマッピングにおいては，イタレーションテンプレートと呼ぶ新たな概念を導入し，柔軟なマッ

† NEC C&C研究所
C&C Research Laboratories, NEC Corporation
†† NEC第一コンピュータソフトウェア事業部
1st Computers Software Division, NEC Corporation

ピングを可能とした.

(2) 高ポータビリティの実現:

データおよび計算処理の物理プロセッサへのマッピングや通信処理など,ターゲットマシンに依存する処理をすべて実行時ライブラリ内で行うようにし,高いポータビリティを実現した.

(3) サブセット以外の HPF 指示行の機能をサポート:

本 HPF コンパイラが受理する HPF の仕様はサブセット[1]を基本としているが,再マッピング指示行(**REALIGN**, **REDISTRIBUTE** および **DYNAMIC** 指示行),**INHERIT** 指示行,**FORALL** 構文および **EXTRINSIC** 関数もサポートしており,HPF が提供するほとんどの指示行の機能が利用可能である.

## 2. HPF 処理系の概要

### 2.1 HPF 処理系の構成

**図 1** に我々が開発した HPF 処理系の構成を示す.まず,HPF コンパイラが,逐次 Fortran 77 プログラムに HPF 指示行および本処理系独自の拡張指示行を挿入した HPF プログラムを並列化し,これに実行時ライブラリ呼び出しを挿入した SPMD (Single Program Multiple Data) プログラムに変換する.生成された SPMD プログラムはターゲットとなる並列システムでコンパイルされ,目的プログラムに変換される.この際,実行時にプロセッサ間通信や並列実行制御などを制御する実行時ライブラリがリンクされる.

現在までに我々は,図 1 に示すように,MPI (Message-Passing Interface)[8]を用いた MPI 版ライブラリと PARALIB/CJ を用いた PARALIB/CJ 版ライブラリの 2 種類の実行時ライブラリを開発した.MPI は現在標準化作業が進められているメッセージパッシング型の通信システムである.一方,PARALIB/CJ は Cenju-3[9]が提供する通信システムであり,他のプロセッサのローカルメモリへ直接アクセスすることが可能である.したがって,HPF コンパイラが生成した SPMD プログラムは,MPI が利用可能であれば Cenju-3 以外のシステムで動作可能である.

HPF では,データのマッピングは HPF で定義される抽象プロセッサ上で表現され,抽象プロセッサから物理プロセッサへのマッピングは処理系に委ねられている.本処理系では,HPF コンパイラが,計算処理のマッピングおよびプロセッサ間通信の生成を抽象プロセッサを対象として行い,抽象プロセッサから物理プロセッサへのマッピングは実行時ライブラリで行う.このため HPF コンパイラは,ターゲットとなるシステムの物理プロセッサ構造や,提供される低レベルな通信システムのインタフェースを意識することなく,一貫した解析を行うことができる.したがって,ターゲットとなる並列システムで利用可能な通信システムを用いて実行時ライブラリを構築すれば,HPF コンパイラが生成した SPMD プログラムは複数のシステムで実行可能となり,高いポータビリティを得ることができる.

### 2.2 HPF コンパイラの構成

HPF コンパイラは**図 2** に示すように 5 つのフェーズからなる.以下各フェーズでの処理を簡単に説明する.

フロントエンド

Fortran 77, HPF 指示行および拡張指示行の構文/字句解析を行い,入力プログラム情報を整理分類して,各文の種類に対応するテーブルと変数/定数情報からなるデータベースを生成する.このとき,**FORALL** か

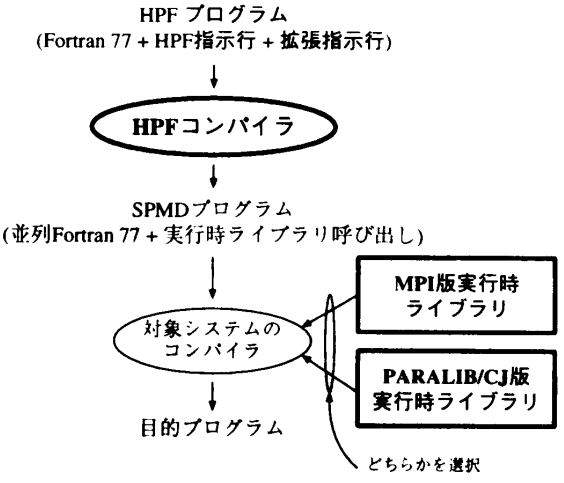



**図 1** HPF 処理系の構成
Fig. 1 HPF compilation system.

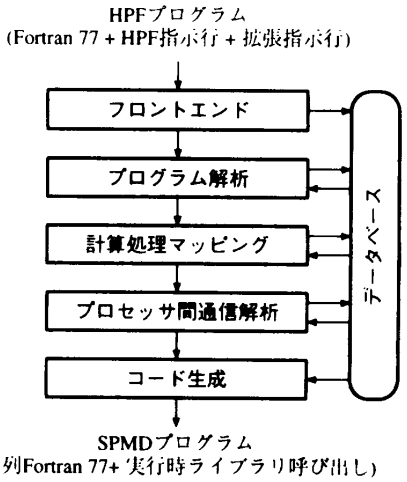



**図 2** HPF コンパイラの構成
Fig. 2 Comfiguration of the HPF compiler.

らDOループへの変換も同時に行う．また，HPF指示行あるいは指示行の一部が省略された場合は，既定値に従って省略された部分を補う指示行を生成する．

**プログラム解析**

データベースを参照してプログラム情報を取得し，基本ブロック分割，データ依存関係解析，制御/データフロー解析，シンボル参照関係解析，リダクション演算の検出，データマッピング解析，配列のアクセス範囲解析，並列実行可能なDOループの抽出を行った後，これらの解析結果をデータベースに付加する．

**計算処理マッピング**

プログラム解析情報をもとに計算処理のマッピング方法を決定し，これを実現するためにDOループの変形などプログラムの変換を行う．

**プロセッサ間通信解析**

プログラム解析情報と計算処理マッピング情報から必要となるプロセッサ間の通信パターンを検出し，検出したパターンに対応した通信コードを生成する．

**コード生成**

以上のフェーズにより変換されたデータベースからソースプログラムを生成する．生成されるプログラムは並列化されたFortran 77プログラムに実行時ライブラリ呼び出しが挿入されたSPMDプログラムである．

## 3. 計算処理マッピング

計算処理マッピングフェーズでは以下の2種類のマッピングを決定する[7]．

(1) イタレーションマッピング
(2) 逐次ブロックマッピング

### 3.1 イタレーションマッピング

イタレーションマッピングとは，ループの各繰返し（イタレーション）をプロセッサに割り付ける（マッピングする）ことである．これを実現するため，我々はイタレーションテンプレートと呼ぶ新たな概念を導入した．イタレーションテンプレートとは，HPFにおけるテンプレートの概念をイタレーション空間に適用したものであり，配列やテンプレートと同様に扱うことができる．本処理系では，**図3**に示すように，イタレーションテンプレートを配列と同様に2レベルマッピングによってマッピングし，イタレーションマッピングを実現している．以下の例を考えてみる．

```
DO I = 1, M
   DO J = 1, N
      A(I,J) = B(I,J)
```

IおよびJのループは，それぞれMおよびN個のイタレーションからなる．したがって，DOループに対応するイタレーションテンプレートをLとすると，これはL(M,N)と表現することができる．本処理系では，イタレーションテンプレートを当該ループ内に出現する配列に整列させることで *owner computes rule*[10] を基本としたイタレーションマッピングを行う．すなわち，例ではイタレーションテンプレートLは，HPFの文法で表現すると「ALIGN L(I,J) WITH A(I,J)」という関係で配列Aに整列される．これにより，LにはAと同じ分散が適用されるため，Aの要素とこれをアクセスするイタレーションが同一のプロセッサへマッピングされることになり，結果として *owner computes rule* が実現できる．

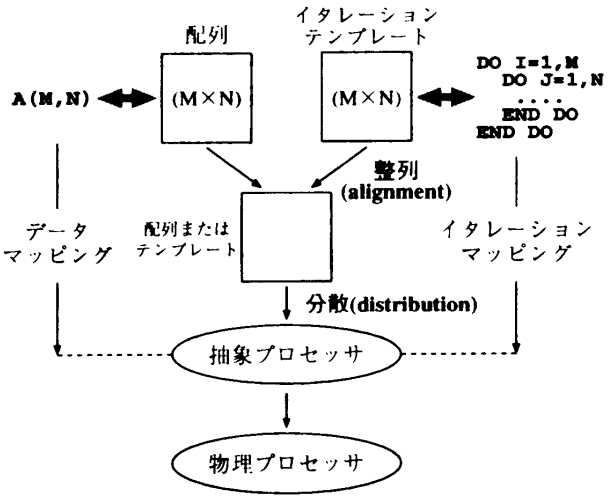



**図3** データ/イタレーションマッピングモデル
Fig. 3 Data and iteration mapping model.

また，イタレーションマッピングは，以下のように拡張指示行を用いて明示的に指示することも可能である．

```
!HPX$ ITEMPLATE L(M,N)
!HPX$ IALIGN L(I,J) WITH A(I,J)

!HPX$ ITERATIONS L(I,J)
      DO I = 1, M
         DO J = 1, N
            A(I,J) = B(I,J)
```

まず，ITEMPLATE指示行でイタレーションテンプレートLを宣言し，IALIGN指示行でイタレーションテンプレートを配列Aに整列させる．イタレーションテンプレートとDOループの対応付けはITERATIONS指示行を用いて行う．また，例では用いていないが，IDISTRIBUTE指示行でイタレーションテンプレートを直接分散することも可能である．このように，本処理系では，HPFにおける配列の分散指示と同様の方法でイタレーションマッピングを指示することができる．

### 3.2 逐次ブロックマッピング

逐次ブロックマッピングとは逐次実行部分のマッピングである．本処理系では拡張指示行を用いて，特定のプロセッサ1台で実行するSINGLEブロック，すべてのプロセッサで実行するALLブロック，特定データを所有しているプロセッサ群で実行するOWNERブロックの3種の実行形態を指示することができる．なお，明示的な指示がない場合には入出力文はSINGLEブロック，それ以外はALLブロックとなる．

## 4. プロセッサ間通信解析

HPFコンパイラは，基本的にはループ，逐次実行ブロックおよび基本ブロックの各実行単位ごとに通信解析を行って必要となるプロセッサ間通信パターンを検出し，これを実現する通信コードを生成する．特にループに対する解析においては，配列を再マッピングするという考え方に基づいて通信パターンを検出する[11]．

### 4.1 DOループに対する通信解析

#### 4.1.1 基本方針

ループの並列実行にともなって必要となる通信は，イタレーションテンプレートのマッピング方法と，ループ内でアクセスされるデータのマッピング方法の不整合に起因するデータの再マッピングと考えることができる．本処理系では，まずループ実行中に他プロセッサのローカルメモリへのアクセスが生じない最適なデータマッピングを導き出し，次に，ユーザが定義したデータマッピングと最適なデータマッピングとの間での再マッピングと見なした通信パターンを検出する．

HPFでは，データのマッピングを「整列（alignment）」と「分散（distribution）」の2レベルのマッピングで定義する[1]．本処理系では，この2レベルマッピングの機能，特に整列の機能を利用して最適データマッピングを抽出する．最適なデータマッピングとは，「あるイタレーションとそのイタレーションでアクセスされる配列要素が同一プロセッサへ配置されるマッピング」であり，これは整列関係のみで実現することができる．以下の例を考えてみる．

```
!HPF$ ALIGN B(I) WITH A(I)
      DO I = 1, N-1
         A(I) = B(I+1)
```

配列Bの最適マッピングとは，A(I)とB(I+1)が同一のプロセッサへマッピングされるマッピング，すなわち，HPFの文法で表現すると「!HPF$ ALIGN B(I) WITH A(I-1)」であり，配列Aの分散方法とは無関係にBとAの間の整列関係のみで保証することができる．つまり，上記の例におけるBの通信とは「!HPF$ ALIGN B(I) WITH A(I)」から「!HPF$ ALIGN B(I) WITH A(I-1)」への整列の変更，すなわち再整列（realignment）を行うことに等しい．このような再整列に基づく通信方式では，以下の利点が得られる．

(1) マッピング方法がコンパイル時に不明な配列に対しても，既知の場合と同じ方式で効率良く通信解析を行うことができる．
(2) ループ実行にともなう通信用の実行時ライブラリと，HPFの再マッピング指示行に対応する実行時ライブラリで主要な機能を共有することができる．
(3) 通信パターンが高レベルな集団通信パターンとなるため，高いポータビリティが得られる．

1番目の利点は特に重要である．なぜなら，HPFでは指示行で記述された再マッピングが実行されるか否か決定できない場合や，仮引数がINHERIT指示行で宣言された場合など，配列のマッピング方法がコンパイル時に不明な場合があるからである．

以上述べた基本方針に基づいて，HPFコンパイラは以下の手順で通信解析を行う．

(1) 整列情報に基づく最適マッピングの抽出
(2) 最適マッピングとユーザが定義したマッピング間での再マッピングの順序決定
(3) 分散情報を利用した定型再マッピングパターンの検出
(4) 通信コードの生成

以下それぞれの手順について具体的に述べる．

#### 4.1.2 最適マッピングの抽出

HPFでは配列間の整列関係は以下の3つの要素によって定義される．

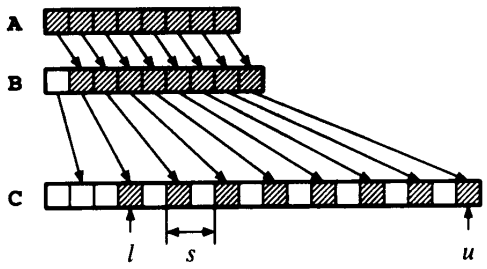
(1) 最終整列ターゲット：最終的に整列されるテンプレートまたは配列．図4の例では，配列Aの最終整列ターゲットは配列Bではなく配列Cである．
(2) 整列次元：配列の各次元が整列される最終整列ターゲットの次元．
(3) 整列三つ組：最終整列ターゲット上での整列の範囲．図4(b)に示すように，$l$（下限），$u$（上限），$s$（ストライド）の3種のパラメータを用いて表す．図4(a)の例におけるAのC上での整列三つ組は[4:18:2]である．

ここで最適マッピングとは，上記の要素において以下の条件を満たすマッピングである．

(1) 配列の最終整列ターゲットがイタレーションテ

```
      DEMENSION A(8),B(9),C(18)
!HPF$ ALIGN A(I) WITH B(I+1)
!HPF$ ALIGN B(I) WITH C(2*I)
```

(a) 整列の記述例




(b) 整列三つ組

図 4 整列三つ組による整列関係の表現
Fig. 4 Alignment representation with align-triplet.

ンプレートのそれと同じである.

(2) 配列の各次元とこれをアクセスするイタレーションテンプレートの次元が最終整列ターゲット上で同一次元に整列される.

(3) 配列の各要素とこれをアクセスするイタレーションテンプレートの要素が最終整列ターゲットの同一要素に整列される.

上記条件の (3) を満たす整列三つ組を最適整列三つ組と呼び, これは以下に述べるようにイタレーションテンプレートの整列三つ組と配列のアクセス範囲情報より求めることができる. なお, アクセス範囲情報は RSD (Regular Section Descriptor)[12] によって表現する. すなわち, 整列三つ組と同様に下限, 上限, ストライドの 3 種のパラメータで表す.

まず, ある配列の要素 $i$ が整列される最終整列ターゲットの要素 $F(i)$ は, 整列三つ組の 3 つのパラメータを用いて以下の式で求めることができる ($LB$ および $UB$ はそれぞれ配列の宣言の下限と上限を表す).

$$F(i) = (i - LB) \times s + l \quad \{ i \mid LB \leq i \leq UB \}$$

ここで, イタレーションテンプレートの整列三つ組を $[l_{lp}{:}u_{lp}{:}s_{lp}]$, 最適整列三つ組を $[l_{opt.}{:}u_{opt.}{:}s_{opt.}]$, アクセス範囲を $[l_{ac}{:}u_{ac}{:}s_{ac}]$ とすると, これらには以下の関係が成立する.

$$\begin{aligned} l_{lp} &= F(l_{ac}) \\ &= (l_{ac} - LB) \times s\_opt. + l\_opt. \quad (1) \\ u_{lp} &= F(u_{ac}) \\ &= (u_{ac} - LB) \times s\_opt. + l\_opt. \quad (2) \\ s_{lp} &= s_{ac} \times s\_opt. \quad (3) \end{aligned}$$

式 (1)～(3) より最適整列三つ組は以下の式によって求めることができる.

$$s\_opt. = s_{lp}/s_{ac} \quad (4)$$

$$l\_opt. = l_{lp} - (l_{ac} - LB) \times s\_opt. \quad (5)$$

$$u\_opt. = (UB - LB) \times s\_opt. + l\_opt. \quad (6)$$

図 5 (a) の例を考えてみる. ここで, 図 5 (b) に示すように, イタレーションテンプレートは整列三つ組 [2:16:2] で配列 A に整列されているとする. また, A, B はそれぞれ自分自身に整列三つ組 [1:16:1] で整列されていると考える. この例では, B のアクセス範囲は [1:15:2] であるため, 上記式 (4)～(6) より B の最適整列三つ組は [2:17:1] となる. したがって, B はループの実行前に「B への整列三つ組 [1:16:1] による整列」から「A への整列三つ組 [2:17:1] による整列」への再整列が必要となる. 図 5 (c) に B の再整列の様子を示す. なお, 図中 L はイタレーションテンプレートを表している.

#### 4.1.3 再マッピングの順序決定

ユーザが定義したマッピングと, コンパイラが抽出した最適マッピングとの間での再マッピングの順序は, 対象となる配列がループ中で *rhs* (*right-hand side*), *lhs* (*left-hand side*) のどちらに出現するかによって以下のように異なる.

**rhs**: ループ実行前にユーザ定義マッピングから最適マッピングへ再マッピングする.

**lhs**: ループ実行後に最適マッピングからユーザ定義マッピングへ再マッピングする.

さらに, 最終整列ターゲットの分散形式 (BLOCK または CYCLIC) や分散サイズ (分散される次元の各プロセッサに割り付けられるサイズ) など, 配列の分散情報が既知の場合は, 次項で述べるように, 再整列を基本とした再マッピングパターンを定型的な再マッピングパターンに特化できる場合がある.

#### 4.1.4 定型再マッピングパターンの検出

分散情報が既知の場合は, これを利用して再整列パターンに基づいて抽出された再マッピングのパターンから以下に示す定型的な再マッピングのパターンを検出する. 定型的な再マッピングパターンを適用することで, 結果的には通信ライブラリでは規則的な通信パターンを利用した高速化が可能となる.

**REDISTRIBUTE**

最適マッピングとユーザ定義マッピング間で分散方法のみが異なる場合の再マッピングである. 分散される次元や分散形式が変更される.

**SHIFT**

最適マッピングとユーザ定義マッピング間で整列三つ組のみが異なる場合の再マッピングであり, 整列関係の差分だけ整列関係をシフトするパターンとなる.

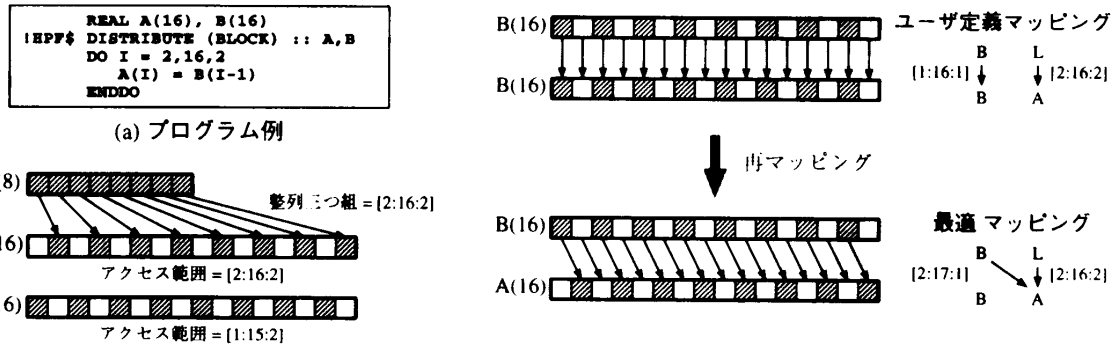



図5 再マッピングの例
Fig. 5 Example of remapping.

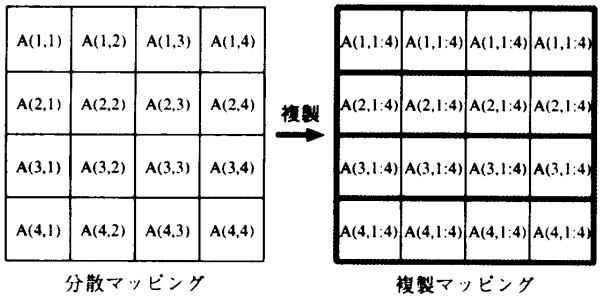



図6 REPLICATE パターンの再マッピング
Fig. 6 Remapping of REPLICATE pattern.

表1 主な通信ライブラリ関数
Table 1 Communication library routines.

| 関数名 | 機能 |
|---|---|
| LREALIGN | ループ実行にともなう再整列 |
| LRDIST | ループ実行にともなう再分散 |
| PRE_SHIFT | 指定された次元の参照前シフト |
| POST_SHIFT | 指定された次元の更新後シフト |
| REPLICATE | 指定された次元の複製 |
| REPLICATE_ALL | 全次元の複製 |
| BROADCAST | 全コピーデータの一貫性保持 |
| GATHER | 特定プロセッサへのデータ収集 |
| SCATTER | 特定プロセッサからデータの所有プロセッサへ分配 |

たとえば，図5の例の場合，配列Bのための通信はユーザ定義マッピングから最適マッピングへの再整列であり，かつ，配列A，Bの分散方法が等しいため，シフトパターンが適用される．

**REPLICATE**

ある次元に関して，BLOCKあるいはCYCLIC形式で分散される分散マッピングから，同一のデータがコピーされる複製マッピングへ変更する再マッピングである．**図6**に例を示す．この例では，2次元ともに分散されている配列Aの2次元目が，プロセッサ配列の2次元目方向に属するプロセッサ群にコピーされる複製マッピングへ再マッピングされる．

### 4.1.5 通信コードの生成

HPFコンパイラは，再マッピングパターンを検出した後，検出したパターンに対応した通信ライブラリ呼び出しを挿入する．**表1**に主な通信ライブラリを示す．これらの通信パターンは抽象プロセッサ上での通信パターンであり，ライブラリ中で物理プロセッサへのマッピングが施され，物理プロセッサ間で通信が行われる．また，すべてのライブラリは，実行時に管理されるデータマッピング記述子（5章参照）を参照し，通信の対象となる配列のマッピング情報を取得する．以下，主な通信ライブラリについて具体的な機能を説明する．

```
!HPF$ INHERIT B
!HPF$ DISTRIBUTE A(BLOCK,BLOCK)

      call LREALIGN(DMD_B,opt_mapping,range)
      call LOOP_BOUND(DMD_A,st1,end1,...)
      call LOOP_BOUND(DMD_A,st2,end2,...)
      DO I = st1, end1
         DO J = st2, end2
            A(I,J) = B(J,I)
         END DO
      END DO
```

図7 LREALIGN通信の例
Fig. 7 Example of LREALIGN communication.

**LREALIGN**

**図7**に再整列マッピングパターンに対応するLREALIGNを用いた通信コードの生成例を示す．例では配列BがINHERIT指示行で宣言されているため，コンパイル時にはマッピング方法が不明である．し

```
!HPF$ DISTRIBUTE (BLOCK) :: A,B

      call PRE_SHIFT(1,1,range,DMD_B,bound)
      call LOOP_BOUND(DMD_A,st,end,...)
      DO I = st, end
         A(I) = B(I-1)
      END DO
```

図8 SHIFT 通信の例
Fig. 8 Example of SHIFT communication.

```
!HPF$ PROCESSORS P(4,4)
!HPF$ ALIGN A(I,*) WITH B(I,*)
!HPF$ DISTRIBUTE B(BLOCK,BLOCK) ONTO P

      call REPLICATE(2,range,DMD_B,bound)
      call LOOP_BOUND(DMD_A,st,end,...)
      DO I = st, end
         DO J = 1, N
            A(I,J) = B(I,J)
         END DO
      END DO
```

図9 REPLICATE 通信の例
Fig. 9 Example of REPLICATE communication.

たがって，定型再マッピングパターンの検出ができず **LREALIGN** が適用される．**LOOP_BOUND** は各プロセッサが実行するループの境界を計算するライブラリ関数である．例では省略されているが **opt_mapping** には再整列のための情報が，また **range** には各次元のアクセス範囲情報がセットされる．なお，**DMD_A**，**DMD_B** は，それぞれ実行時に管理される配列 **A**，**B** のデータマッピング記述子のアドレスである．

**PRE_SHIFT/POST_SHIFT**

両ライブラリともシフトパターンに対応する通信ライブラリであり，分散されている次元の分散境界にある要素を抽象プロセッサ上で隣接するプロセッサへ転送する機能を持つ．**PRE_SHIFT** および **POST_SHIFT** はそれぞれ *rhs*，*lhs* に出現する配列に対して適用される．図 **8** に **PRE_SHIFT** が生成された例を示す．

**REPLICATE**

**REPLICATE** は特定の次元の要素を複製する通信ライブラリである．複製とは，プロセッサ配列のある次元方向のプロセッサ群が同一データを持つ状態である．つまりこの通信は，特定次元に属するプロセッサ群でのマルチキャスト通信となる．図 **9** に **REPLICATE** が生成された通信コードの例を示す．この例では，配列 **B** がプロセッサ配列 **P** の 2 次元目方向のプロセッサ群に複製される．

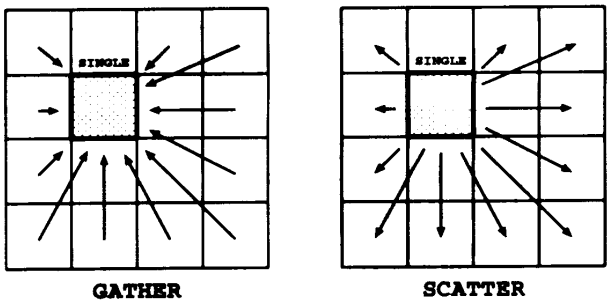



図10 GATHER/SCATTER 通信
Fig. 10 GATHER/SCATTER communication.

### 4.2 逐次実行ブロックに対する通信解析

逐次実行ブロックでは，実行前に参照されるデータをそのブロックを実行するプロセッサ群が収集し，実行後に更新されたデータをそのデータが割り付けられているプロセッサへ転送する必要がある．これを実現するために，それぞれ **GATHER**，**SCATTER** の 2 種類の通信ライブラリを開発した．図 **10** に SINGLE ブロックに適用した例を示す．図中の 2 次元格子はプロセッサ配列を表し，矢印はデータの転送方向を表す．

## 5. 実行時のデータマッピング情報管理

4.1 節で述べたように，HPF ではデータのマッピングがコンパイル時に不明な場合がある．そこで我々は，実行時に配列ごとにマッピング情報を記述した記述子 (DMD: Data Mapping Descriptor) を生成し，これを管理する機構を構築した．すべての実行時ライブラリは DMD を参照してデータマッピング情報を取得する．また，DMD は手続き呼び出しの際に，呼び出す手続きから呼び出される手続きに渡される．HPF では各手続き内でのマッピングは独立であるため，手続きの入口では，引き渡された DMD と手続き内のマッピングが比較され，必要であれば再マッピング処理が行われる．また，手続きの出口では入口と逆の処理が行われる．

## 6. 通信最適化

通信のオーバヘッドを最小化することは，高性能の並列プログラムを生成するためには必要不可欠である．現在までいくつかの最適化手法が提案されているが[13),14)]，本 HPF 処理系では，以下のようにコンパイル時および実行時の両方で最適化を行っている．

### 6.1 コンパイル時の最適化

(1) Message coalescing

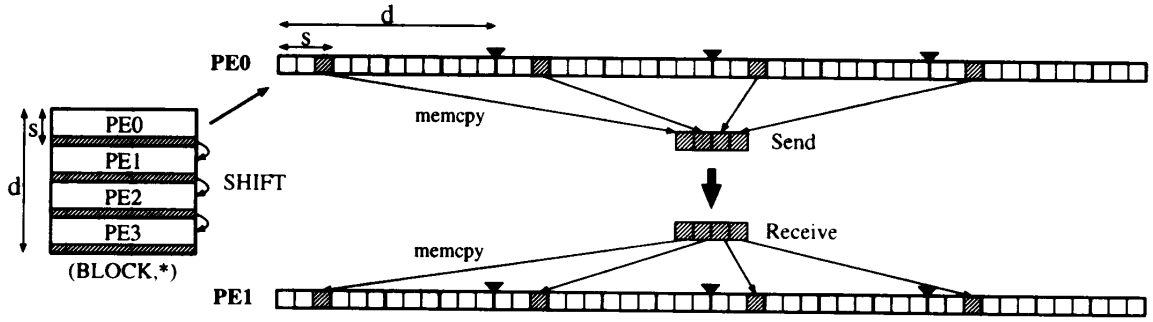



図11 SHIFTライブラリにおける self aggregation の適用
Fig. 11 Self aggregation in SHIFT function.

同一配列に対する異なったアクセスの範囲を融合し，重複した通信を削除するとともに，不連続なアクセスをできるだけ連続アクセスとすることで，通信ライブラリ関数の呼び出し回数を最少化する．

(2) Message aggregation

本最適化は同一プロセッサ間で転送される不連続なデータをいったんバッファに格納し，これを一括して転送することで通信の回数を減らし，オーバヘッドを削減しようというものである．本処理系では以下の2種類の aggregation 最適化を採用している[14]．

- self aggregaion：同一配列へのアクセスに対する aggregation
- group aggregation：異なる配列へのアクセスに対する aggregation

どちらの aggregation 最適化処理も実際には通信ライブラリ内で行われるが，group aggregaion を行うために，HPF コンパイラは同一の通信ライブラリで通信処理が可能なデータの集合を生成し，この集合に属するデータ群に対して1回の通信ライブラリ関数呼び出しを生成する．

(3) 不要通信の除去

並列実行されるループ中でスカラ変数が更新される場合，ループ実行終了時には各プロセッサが異なる値を保持している可能性がある．このため，最後に更新したプロセッサがその値をブロードキャストする必要がある．しかしながら，スカラ変数が並列ループの出口で死んでいる（ループ実行後に参照されないか，あるいは参照される前に更新される）場合はブロードキャストは不要である．HPF コンパイラは，live-variable 解析[15]により並列ループの出口で死んでいる変数を検出し，これらの変数に対するブロードキャストを削除する．

### 6.2 実行時の最適化

(1) 定型再マッピングパターンの検出

配列のマッピングが不明な場合は，HPF コンパイラは LREALIGN を用いた通信コードを生成する．しかしながら LREALIGN では，DMD のマッピング情報をもとに HPF コンパイラと同様の解析を行い，定型再マッピングパターンが適用可能な場合は，これに対応する通信ライブラリを呼び出して高速な通信を実現する．

(2) Message aggregation

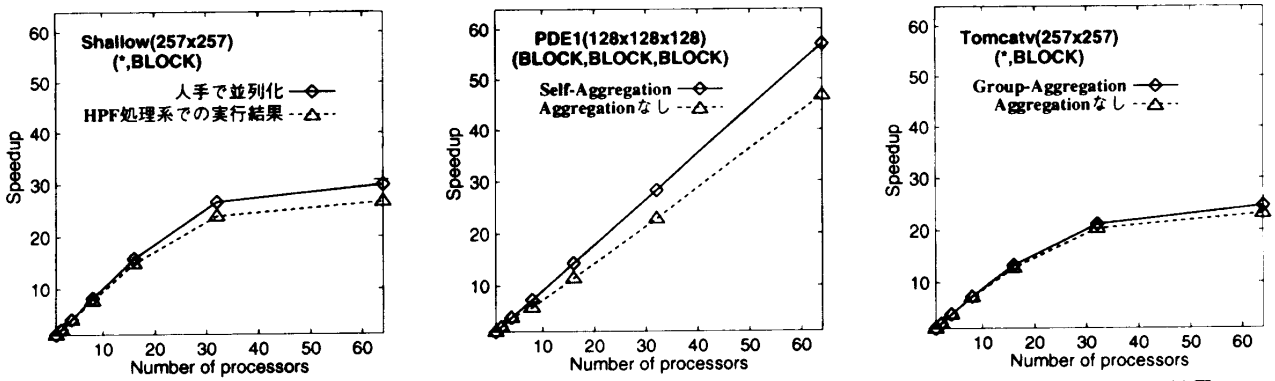
図11 に，SHIFT において，PE0（送信側）と PE1（受信側）間で self aggregation が適用された例を示す．例では，配列の1次元目が分散されているため，1次元目方向の通信で転送すべき配列の要素はメモリ上で不連続となるが，self aggregation を適用することによって不連続な要素を一括して転送することが可能となる．

## 7. 評 価

本 HPF 処理系の評価を NCAR ベンチマークの Shallow，SPEC ベンチマークの Tomcatv および GENESIS ベンチマークの PDE1 を用いて，64台構成の分散メモリ型並列コンピュータ Cenju-3[9]上で行った．結果のグラフはオリジナルの逐次プログラムを1台で実行した時間に対する速度向上率を示している．また，いずれの評価も MPI 版の実行時ライブラリを用いて行った．

### 7.1 Shallow

Shallow を人手で並列化した場合と本 HPF 処理系を用いて並列化した場合の比較を行った．結果を図12(a)に示す．本処理系での結果は人手で並列化した場合の10%以内と良い結果が得られている．Shallow では，主に SHIFT を用いた通信コードが生成された．SHIFT 中では通信処理のほかに，DMD からの配列のマッピング情報の取得やエラーチェックなどを行っているが，これらの通信ライブラリ内でのオーバヘッドや DMD の生成および管理のためのオーバヘッドを考慮しても，本処理系では許容できる性能が得られている．これにより，本処理系におけるプロセッサ間通信の生成方式




(a) Shallowの実行結果 (b) PDE1の実行結果 (c) Tomcatvの実行結果

**図 12** ベンチマークプログラムの実行結果
Fig. 12 Benchmark results.

およびDMDを用いたデータマッピング情報管理方式の有効性が分かる.

### 7.2 PDE1

PDE1は SHIFT で self aggregation 最適化が適用された例である. 最適化を適用した場合としなかった場合の結果を図12(b)に示す. 最適化によって約1.2倍の性能向上が得られた. PDE1では3次元配列のすべての次元を分散したので, 分散した3次元すべてに SHIFT が適用された.

### 7.3 Tomcatv

Tomcatv は SHIFT で group aggregation 最適化が適用可能な例である. 最適化を適用した場合としなかった場合の結果を図12(c)に示す. 2個の配列に対して最適化が適用された. PDE1での self aggregation ほどの性能向上は得られていないが, group aggregation が適用可能な配列が多くなれば大きな効果が期待できる.

## 8. 関連研究

Rice大は HPF のベースとなった Fortran D 用の並列化コンパイラを開発しているが, 1次元分散のみのサポートなど制限が多い[10]. HPF 処理系に関しては, 富士通研究所[2], IBM[5], PGI (Portland Group Inc.), APR (Applied Parallel Research), Syracuse大[3], GMD[4]等が開発を進めている. 富士通研究所のコンパイラは VPP Fortran を中間表現としており, 実行時の再マッピングなど VPP Fortran にはない HPF の機能はサポートされていない. PGI, Syracuse大, GMD のコンパイラはいずれも forall のみを並列化の対象としている. また, いずれのコンパイラもコンパイル時にマッピングが不明な配列の通信コード生成に関しては言及されていない. 本処理系では, データの再マッピングという考え方に基づいた通信方式を採用し, されに実行時にデータ分散情報を管理する機構を構築したことでこの問題に対処した.

## 9. まとめ

以上, 我々が開発した HPF 処理系について述べ, ベンチマークプログラムを用いた Cenju-3 上での評価結果により有効性を示した. 本 HPF 処理系はほとんどの HPF 指示行をサポートし, かつ高いポータビリティを実現した. また, イタレーションマッピングにおいてイタレーションテンプレートと呼ぶ新しい概念を導入し, 通信解析においてはこれを利用することで, データの再マッピングで表現される通信パターンを適用する. これによって実行時にデータのマッピングが不明の場合でも効率の良い通信コードを生成することが可能になった. 今後も引続き他のベンチマークプログラムで評価を行っていくとともに, 我々が拡張した指示行を HPF Forum へ提案していく予定である.

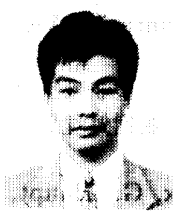
**謝辞** 本研究の機会を与えていただいた第一コンピュータソフトウェア事業部 片山事業部長代理, C&C 研究所 山本所長, 同コンピュータ・システム研究部 中崎部長, 同 中田課長に感謝いたします.

## 参考文献

1) High Performance Fortran Forum: High Performance Fortran Language Specification Version 1.1 (1994).
2) 進藤達也ほか: FLoPS: 分散メモリ型並列計算機を対象とした並列化コンパイラ, 並列処理シンポジウム, pp.377-384 (1995).
3) 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, *Proc. Supercomputing '93* (1993).
4) Brandes, T. and Zimmermann, F.: Adaptor -

A Transformation Tool for HPF Programs, *Working Conference of the IFIP WG 10.3* (1994).
5) Gupta, M., et al.: An HPF Compiler for the IBM SP2, *Proc. Supercomputing '95* (1995).
6) 蒲池恒彦ほか:HPF 処理系の実現と Cenju-3 での評価, 並列処理シンポジウム, pp.361–368 (1995).
7) 末広謙二ほか：HPF 処理系における計算処理マッピング, 並列処理シンポジウム, pp.369–376 (1995).
8) Message Passing Interface Forum: MPI: A Message-Passing Interface Standard (1995).
9) 広瀬哲也ほか：並列コンピュータ Cenju-3 のプロセッサ間通信方式とその評価, 並列処理シンポジウム, pp.241–248 (1995).
10) Tseng, C.-W.: An Optimizing Fortran D Compiler for MIMD Distributed-Memory Machines, Ph.D. Thesis, Rice University (1993).
11) Kamachi, T., et al.: Generating Realignment-Based Communication for HPF Programs, *Proc. 10th International Parallel Processing Symposium* (1996).
12) Havlak, P. and Kennedy, K.: Experience with Interprocedural Analysis of Array Side Effects, *Proc. Supercomputing '91* (1991).
13) Hiranandani, S., Kennedy, K. and Tseng, C.-W.: Evaluation of Compiler Optimizations for Fortran D on MIMD Distributed-Memory Machines, *Proc. ACM International Conference on Supercomputing* (1992).
14) Amarasinghe, S. and Lam, M.S.: Communication Optimization and Code Generation for Distributed Memory Machines, *Proc. ACM SIGPLAN '93 Conference on Programming Language Design and Implementation* (1993).
15) Aho, A.V., Sethi, R. and Ulman, J.D.: *Compilers:Principles, Techniques, and Tools*, Addison-Wesley (1986).

(平成 7 年 9 月 1 日受付)
(平成 8 年 4 月 12 日採録)

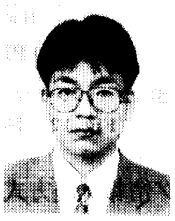


**蒲池 恒彦**（正会員）

1964 年生. 1988 年九州大学工学部情報工学科卒業. 1990 年同大学院総合理工学研究課情報システム学専攻修了. 同年 NEC 入社. 現在, C&C 研究所コンピュータシステム研究部主任. 並列計算機アーキテクチャ, 自動並列化コンパイラ, 並列化支援ツールなどに興味を持つ.

**草野 和寛**（正会員）

1965 年生. 1989 年九州大学工学部情報工学科卒業. 1991 年同大学院総合理工学研究科情報システム学専攻修了. 同年 NEC 入社. 現在, C&C 研究所コンピュータシステム研究部勤務. 分散メモリマシンのための並列化支援システムの研究に従事.



**末広 謙二**（正会員）

1965 年生. 1988 年京都大学工学部電気工学科卒業. 1990 年同大学院情報工学専攻修士課程修了. 1993 年同博士課程単位取得退学. 同年 NEC 入社. 現在, C&C 研究所コンピュータシステム研究部勤務. 並列計算機の基本ソフトウェアの研究開発に従事.

**妹尾 義樹**（正会員）

1961 年生. 1984 年京都大学工学部情報工学科卒業. 1986 年同大学院修士課程修了. 同年 NEC 入社. 以来 C&C 研究所にてスーパーコンピュータの研究開発に従事. 特に並列処理アーキテクチャ, 分散メモリマシンのための並列化支援システム, 並列アルゴリズムに興味を持つ. 現在 C&C 研究所主任. 工学博士. 1988 年本会論文賞受賞.

**田村 正典**

1962 年生. 1983 年国立詫間電波工業高等専門学校卒業. 同年 NEC 入社. 以来汎用コンピュータ ACOS シリーズおよびスーパコンピュータ SX シリーズの FORTRAN 言語処理系の開発に従事. 現在, 第一コンピュータソフトウェア事業部言語開発部主任.

**左近 彰一**（正会員）

1959 年生. 1982 年京都大学工学部数理工学科卒業. 昭和 1984 年同大学院工学研究科数理工学専攻修士課程修了. 同年 NEC 入社. 現在, 第一コンピュータソフトウェア事業部言語開発部にてスーパーコンピュータの言語処理系の研究開発に従事.