解 説 HPF言語の動向

2. HPF言語の現状と将来

Present and Future of High Performance Fortran (HPF) by Yoshiki SEO (C&C Research Laboratories, NEC Corporation).

妹尾 義樹[1]
1 NEC C&C研究所

1. はじめに

並列計算機を本当に実用レベルの計算ツールとするためには，並列プログラミングインタフェースの標準化が重要である．近年多数の並列計算機が商用化されているが，応用面での並列ソフトウェア技術の蓄積をはかるためには，プログラムのポータビリティーが不可欠だからである．そこで最近になって，HPF(High Performance Fortran)とMPI(Message Passing Interface)という2つのプログラミングインタフェースがデファクトスタンダードとして提案された．

MPIはメッセージパッシング，つまり，プロセッサ間のデータ転送を，ユーザがライブラリを用いて明示的に記述する枠組みである．SPMD(Single Program Multiple Data Stream)形式に基づいて，複数の制御の流れを意識したプログラミングが必要になる．これに対してHPFは従来のFortran言語に最小限の付加的指示で並列化を可能とすることを狙ったデータパラレル言語である．現在はまだ仕様そのものや処理系(コンパイラ)の完成度の面で発展途上であるが，大きな可能性を秘めるとともに，今後，分散メモリマシンを並列処理の専門家だけの研究ツールから実用ツールにするためには必須のプログラミングインタフェースである．

分散メモリシステム上の並列化において，最も重要となるのは，データ分割，つまり処理対象データの分散メモリ上への配置と，処理分割，つまり計算処理の各要素プロセッサ(以後PEと記す)への分配である．並列化の良し悪しは，並列性の抽出のみならず，いかにデータアクセスローカリティーを高め，プロセッサ間通信を減らすかに大きく依存する．HPFの本質的な考え方は，このローカリティー抽出のためのデータ分割をユーザが明示的に指示し，それ以外の仕事を処理系に任せようというものである．これにより，ユーザは煩わしいプログラムのSPMD化や通信管理から解放され，従来のFortranプログラミングと非常に親和性の高い方法で分散メモリシステムを利用することが可能となる．メッセージパッシングと異なり，HPFでは，プログラムはシングルストリームのセマンティックスで記述され，また，データ分割の指示は必要だが，データはすべて大域的に扱われる．

HPF言語の標準化活動は，デファクトスタンダードを目指して，ISOやANSIなどの公式な標準制定機関ではなく，Rice大学のKen Kennedy教授を中心とするプライベートな組織HPFF(HPF Forum)で行われた．ANSIを中心に行われたFortran90の仕様制定に10年以上かかった反省からである．言語仕様に組み込むべき新機能ごとにサブワーキンググループを作り，ここで仕様案が作られ，これらが参加者の投票を経て正式に仕様に組み込まれる．活動は1991年に開始され，1993年にHPF1.0が制定された[1]．その後もHPFF2において，仕様の曖昧性のチェックや，HPF1.0で不十分であった機能拡張が検討され，現在HPF2.0の仕様がレビューされているところである(1996年10月現在)．HPFFのホームページが，http://www.crpc.rice.edu/HPFF/ にあり，HPF言語の仕様や関連研究プロジェクトの情報が得られるので，興味のある読者はこちらを参照されたい．

本稿では，まず，HPF1.0の仕様および，そのコンパイラの処理概要を説明し，HPF1.0の言語

仕様の問題点について述べる．さらに現在検討されているHPF2.0の言語仕様案，そしてHPFの将来について概観する．

2. HPF言語の仕様

HPF言語の最大の特徴は，データの分散メモリへの配置の指定機能である．本章では，この配置の指定方法を中心にHPF1.0の言語仕様を説明する．

2.1 データ分散配置のモデル

データの分散配置を指定する目的は，データアクセス局所性の抽出により，並列処理におけるPE間通信オーバヘッドを減らすことである．次の簡単なプログラムの例を考える．

```
DO I=1,N
  A(I) = B(2*I) + C(I+1)
END DO
```

このループの並列化において，通信を考慮すると，A(I)，B(2*I)およびC(I+1)の対応する要素が同一のPEのメモリに割りつけられると都合がよい．このために，配列要素間の配置の関係，つまり，異なる配列のどの要素と，どの要素を同一のPEメモリに配置するかを記述するのが整列(Alignment)である．

```
DO I=2,N
  S = S + D(I) + D(I-1)
END DO
```

一方，この例では，配列Dの連続する要素が，なるべく同一PEに配置されるのがよい．このために，同一配列の各要素をどのように分散メモリにレイアウトするかを指定するのが分散(Distribution)である．

HPFでは，このように，整列と分散の2段階の枠組みを用いて，配列データをユーザ定義の論理プロセッサ配列にマッピング(Mapping)する．つまり，図-1に示すように，ALIGN指示行により複数の配列を代表配列またはテンプレート(Template)と呼ばれる仮想配列に整列し，配列相互の対応関係を記述する．次にDISTRIBUTE指示行により，代表配列またはテンプレートの分散メモリへのレイアウトを指定する．プロセッサ配列は，PROCESSORS指示行により，配列の宣言と同様に任意次元の矩形形状のものを宣言する．論理プロセッサと物理プロセッサとのマッピングはシステムに任されている．

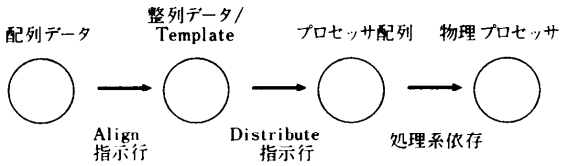

図-1 HPFにおけるデータマッピングのモデル

2.2 データの整列

文法規則などの仕様の詳細は，文献1),2)を参照していただくとして，ここでは例を用いて指示行の機能について説明する．

```
!HPF$ ALIGN A(I) WITH B(I)
```

この例は，Iという添字を介して，1次元配列AとBの配置を同一にすることを指示する．Aを**被整列変数**(Alignee)，Bを**整列対象変数**(Align Target)，Iを**ダミー変数**(Align Dummy)と呼ぶ．

整列対象変数に用いられるダミー変数が単純引用の場合には，これを「:」で置き換えて，

```
!HPF$ ALIGN A(:) WITH B(:)
```

のように記述してもよい．この指示行の意味は先の例と同一である．

```
!HPF$ ALIGN C(I) WITH B(2*I-1)
```

のように書くと，配列Cの宣言範囲のすべてのIについて，C(I)とB(2*I-1)が整列される．たとえば，C(1)とB(1)，C(2)とB(3)，C(3)とB(5),...といった対応となる．

```
!HPF$ ALIGN A(:,:) WITH B(:,:)
```

複数の「:」がある場合には，左から順に対応がとられる．これは2次元配列Aを変換なしにBに整列することを意味する．ただし，転置した整列関係を表すには，少なくとも1つのダミー変数を導入して，

```
!HPF$ ALIGN A(I,J) WITH B(J,I)
```

または

```
!HPF$ ALIGN A(I,:) WITH B(:,I)
```

と記述する必要がある．

また，「*」を用いて，次元の異なる配列間の整列を指定できる．

```
!HPF$ ALIGN D(I,*) WITH B(I)
```

この例において，被整列変数に示された「*」は**縮退**(Collapsing)を示し，Dの2次元目の添字が整列に無関係であることを意味する．配列D

の1次元目の添字とBの添字が同じであれば，Dの2次元目の添字の値にかかわらず同じPEにマッピングされる．つまり，配列Dのそれぞれの行に対してBの各要素が整列されることになる．

また，整列対象の方に「*」を記述すると，これは**複製**(Replication)になる．

```
!HPF$ ALIGN A(:) WITH B(:,*)
```

は，2次元配列Bに1次元配列Aを整列する際に，Aの複製をBの2次元目のサイズの個数だけ作成し，Aを2次元配列に拡張した後，その各要素をBに対応させる．複製は，READ ONLYのデータについて，各プロセッサにコピーをもたせて通信を削減したいときに便利である．

2.3 データの分散

DISTRIBUTE指示行は配列またはテンプレートの各次元に対して，BLOCK, CYCLIC,「*」を指定することにより，配列データの分散メモリ上のレイアウトを指定する．

```
   REAL A(100,100)
!HPF$ PROCESSORS PROC(10)
!HPF$ DISTRIBUTE A(BLOCK,*) ONTO &
!HPF$ PROC
```

この例では2次元配列Aを1次元方向で等分割しプロセッサ配列PROCにマッピングする．CYCLICが指定された場合には，ラウンドロビン的な割当てを行う．「*」は，指定された次元での分割を行わないことを意味する．**図-2**に2次元配列の種々の分割例を示す．通常，偏微分方程式を近傍参照モデルで解くような場合にはBLOCK分割が適している．一方，LU分解など，計算の進行とともに，計算対象領域が変化するような場合には，負荷均衡化をはかるためにCYCLIC分割が有効な場合がある．また，CYCLIC(4)などのように，分割のスライス幅を明示的に指定することもできる．

2.4 動的再配置

HPFでは，プログラムの実行中にデータのマッピングを動的に変更するために，REALIGN, REDISTRIBUTEという指示行が用意されている．これらの機能は，それぞれALIGN, DISTRIBUTE指示行と同様だが，実行文として扱われ，指示行が挿入されている時点でそれぞれ整列，分散が変更される．整列先となっている変数またはテンプレートがREDISTRIBUTEされる場合には，これらの被整列変数も整列関係を保持するようにマッピングが変更される．これらの指示行により動的にマッピングが変更される配列は，DYNAMIC指示行により明示的に指定する必要がある．動的再配置は，ADI(Alternate Direction Integrate)法のように多次元配列を，順次方向を変えて走査(sweep)するような例で有効である．

2.5 手続き間インタフェース

HPFでは，分割コンパイルを前提として設計されている．そこで，処理系が手続き間でのデータマッピングをできるだけ効率よく制御できるように，仮引数のデータマッピングについて，**記述的**(descriptive)，**模写的**(transcriptive)，**指令的**(prescriptive)の3種類の指定方法が用意されている．

記述的指定は，DISTRIBUTEやALIGNでのマッピング指示に「*」を付加することにより，実引数のマッピングがそれらと同一であることを明示的に示すものである．

```
!HPF$ DISTRIBUTE A*(BLOCK) ONTO &
!HPF$ PROC
```

この例では，(BLOCK)およびPROCの前に「*」をつけることにより，配列Aに対応する実引数もブロック分割され，PROCと同一形状のプロセッサ配列にマッピングされていることを示す．この指定により処理系は手続き間でのマッピング一致チェックを省略できる．

模写的指定は，明示的に仮引数のマッピングを指定せず，INHERIT指示行により，実引数のそれと合致させることを指示する．手続きインタフェースでデータ転送が不要だが，コンパイル時に

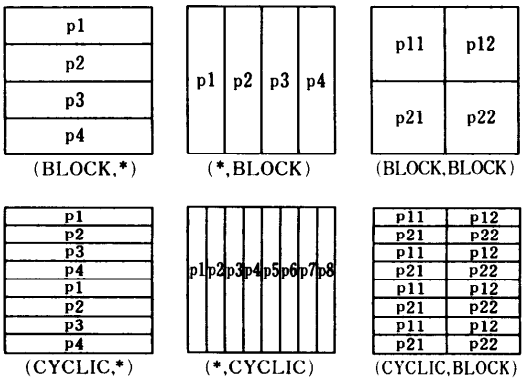

図-2 DISTRIBUTE指示行による2次元配列の分割例

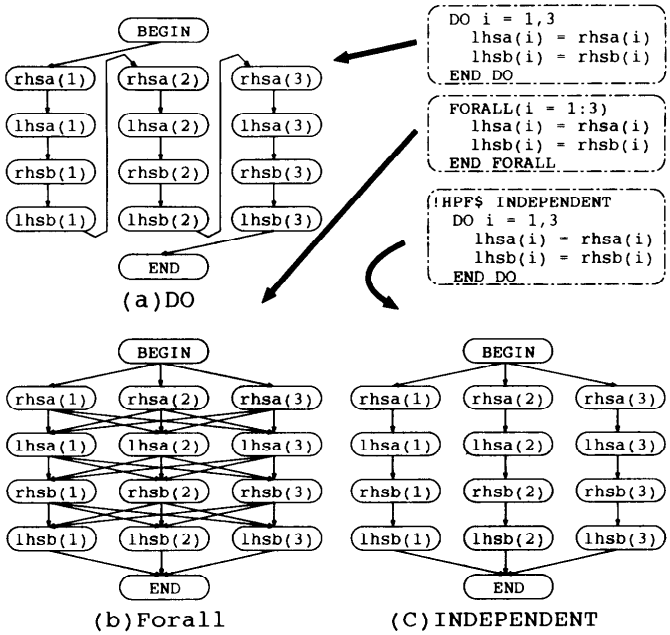

図-3 Forall/INDEPENDENTによる実行順序の制約

静的にマッピングがわからないため，記述的指定に比べると，生成されるコードの効率が低下する場合がある．

指令的指定は，上記2つの方法によらず，対応する実引数のマッピングとは無関係に仮引数のマッピングを指定する方法であり，この両者のマッピングが異なれば，処理系により自動的に必要な通信が生成される．

2.6 その他の機能

HPFは上記のほかにも，以下のような機能を用意している．

• FORALL構文, INDEPENDENT指示行

両方とも，ループの並列実行を促進するための枠組みで，通常のDOループよりデータ依存関係を弱める働きをもつ．図-3に，DOループ，FORALLループ，INDEPENDENT指示のあるループのそれぞれの文の実行順序についての制約を示す．DOループでは，ループ1周ごとに各文の右辺の評価，左辺の更新を順に行うが，FORALLでは，各文について，配列の複数要素を同時処理するセマンティックスになっている．実行イメージは，Fortran90の複数の配列演算を順に処理していくものと同等であり，とくに，ベクトルマシンやSIMDマシンに適した構文である．INDEPENDENT指示は，DOループとFORALLループの両方に指定が可能で，ループの各繰り返しを独立に行えることを示す．

• PURE属性

手続き，とくに関数の属性として指定し，これらに副作用のないことを宣言する．関数呼出しを含むDOループの並列化の際に効果がある．

• INTRINSIC手続き(HPF組込み関数)

実行プロセッサ数を返すNUMBER_OF_PROCESSORS()などのシステムの状態を得る関数や，配列のマッピング状態を問う関数，総和，SHIFTなどの配列演算などが用意されている．

• EXTRINSICインタフェース

HPFプログラムの一部をMPIなどのメッセージパッシングやほかの並列処理パラダイムを使って記述するためのインタフェースが用意されている．

• SEQUENCE指示行

古いコードや，ベクトル化チューニングを意図したコードには，ある複数の配列がメモリの連続番地に割りつけられることを期待するものや，手続きインタフェースで形状の異なる配列を実/仮引数として結合させるものなどがある．これらは，HPFの最適化により，不正動作の原因となり得る．そこで，本指示行により，指定した配列に対する最適化の抑制を明示的に指示することがで

きる．

3. HPF処理系の実装について

3.1 HPF処理系の基本処理

HPF処理系の主要な処理は，自動並列化を含む計算マッピングおよび通信の生成である．ユーザ指定のデータマッピングを利用して，通信ができるだけ少なくなるように計算処理を分割し，必要な通信を生成する．HPF処理系が生成するSPMD形式の並列コードのイメージを図-4に示す．

代入文の左辺の変数を所有するPE(Owner)が計算を実行するように計算処理を分割する方法を，Owner Computes Ruleと呼ぶ．最も簡単な実装では，図に示すように，各文に実行ガードと呼ばれるIF文を挿入する．SPMD形式で実行に参加する前PEの制御が実行ガードに到達し，このIF文でループ処理を選択的に行うことにより並列実行が可能となる．右辺で，自PEが所有しないデータが参照されると，これに対する通信が生成される．以下にHPF処理系で必要となる主な処理について説明する．

• 並列化ループの決定

データ依存関係解析により，各ループの並列実行可能性を判定する．並列化不可ループについては，必要となる通信量により，全PEで実行するか，特定のPEで実行するかを判断する．並列化可能ループについては，必要となるデータ転送を考慮して，実際に並列化するか否かを決定する．また，総和，最大値などのリダクション演算を認識し，各PEでローカルに求めた値をまとめる処理を生成する．

• 通信生成

ループまたは基本ブロックを単位として，各配列に対するアクセス範囲を求める．そして，データマッピングの情報と比較して，自分の所有していない範囲のデータアクセスに対して通信を生成する．この通信範囲の計算や，リモートデータを所有するプロセッサの計算はシンボリックに行う必要があるが，これを汎用的に支援する処理系構築ツールとして，Maryland大学で開発されたOMEGAライブラリ(http://www.cs.umd.edu/projects/omega参照)がある．

• バッファ管理

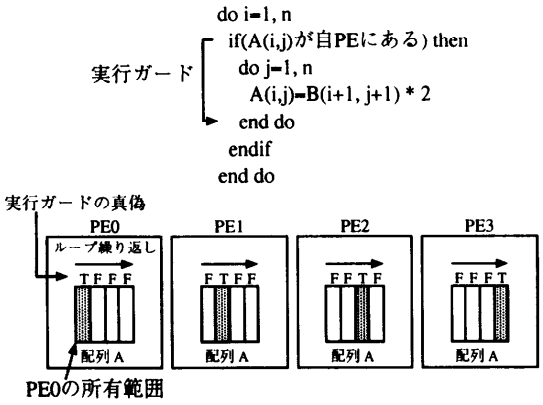

図-4 Owner Computes Ruleによる並列化

リモートメモリアクセスのために他PEから転送したデータをいったん保持するメモリ領域やデータの動的再配置のための領域の確保／解放を管理する．

• アドレス変換

HPFプログラム上では，配列アクセスはグローバルアドレスで表現される．一方で各PEは配列の部分しか保持しないため，この上でアクセスするためのローカルアドレスに変換する必要がある．BLOCK分割の場合は，比較的簡単だが，とくにスライス幅をもつCYCLIC分割の場合は，各PEが非線形な領域を保持する(たとえば，A(1), A(2), A(17),A(18)・・・など)ため，変換が繁雑になる．

• 動的マッピング管理／手続き間リマッピング管理

REALIGN/REDISTRIBUTEや手続き間での配列データの再配置を行うためには，実行時に各配列のマッピング情報を管理する必要がある．通常は，再配置前後のマッピング情報を引数で渡すことにより，実行時ライブラリで処理する．

この他にも，通信ライブラリの初期化，プロセッサ台数の管理，入出力を特定プロセッサで実行するための制御などを行うコードを挿入したり，組み込み関数のサポート，FORALL文の処理などを行う必要がある．

3.2 最適化技法

[Owner Computes Ruleの緩和]

ループの中に実行ガードを挿入するのはオーバヘッドが大きい．また，ループの同じ繰返しで実行される複数の文の間に順依存のデータ依存関係

があり，これらの実行が別々のプロセッサに割りあてられると，ループ内での通信が必要になる．したがって，可能な限り，ループ1周の処理を同一PEで実行できるよう，ループ処理を分割する．通常は，ループ内の配列アクセスを対象配列のマッピングにより分類し，最もアクセス頻度の多いものについて通信が必要ないようにループを分割する．また，規則的分割の場合には，IFガードを削除し，代わりに各PEのループの繰返し範囲をそれぞれの担当領域に置き換える．

[通信最適化]

• Message Vectorization

連続するリモートアクセスをまとめてループの外に出すことにより，メッセージサイズを大きくする．

• Message Coalescing: 複数の同一配列に対する通信をまとめる．たとえば，A(1:10)とA(3:12)に対する通信をA(1:12)にまとめる．

• Message Aggregation: 別々の通信をまとめてメッセージサイズを大きくする．たとえば，A(1:10)とB(1:10)の通信を行う際に，転送範囲が連続アドレスとなるように，これらをいったんバッファエリアにコピーし，それから転送する．

ほかの通信最適化としては，ループ間／手続き間にまたがる通信最適化[3]や，SHIFT型の通信で必要になる通信バッファを動的に確保しないですむように，あらかじめ配列領域を広めに確保するといったものがある．

[その他の最適化]

• Inspector/Executorによる非定型通信の最適化

間接添字など，非線形添字の配列アクセスに対する通信は，単純な実装をすると，ループ中での1語ずつの転送か，全配列のブロードキャストが必要となり効率が悪い．このような場合に，通信をまとめる手段として，INSPECTOR/EXECUTORという方法がある[4]．これは，並列ループの処理を**Program Slicing**により転送のアドレス計算をする部分(Inspector)と，通信および実際の計算をする部分(Executor)に分ける方法である．Inspectorのオーバヘッドが大きいため，外側にさらなる繰り返し構造があり，Inspectorで生成した通信が再利用できる場合にだけ効果がある．処理系のコード生成で利用できる汎用の実行ライブラリとして，Maryland大学のJoel Saltzのグループが開発したCHAOSライブラリがある．(http://www.cs.umd.edu/projects/hpsl.html参照)

• パイプライニング[5]

2重ループで，並列化可能なループのアクセス方向と，データが分割されている次元が同じ場合，通常はデータの転置が必要となる．このような場合，図-5に示すように，並列化対象のループをスライスして，流れ作業的に並列化することにより，通信がSHIFTパターンになり，しかも演算と通信をオーバラップできる．

• イタレーションスプリッティング

SHIFTパターンの転送が必要になる際に，ループの繰り返しを，プロセッサ境界のリモートアクセスが必要な部分とそうでない部分に分けることにより，アドレス変換のオーバヘッドを削減する．リモートアクセスが現れない方のループでは，アドレス変換が不要になる．

4. HPF1.0の問題点

HPF1.0の問題点は，言語仕様として，並列化／通信最適化についての記述力の不足と，これを補うだけの処理系による自動最適化が現状の技術では困難なため，並列化の適用範囲が制限されることである．具体的には以下のような問題点がある．

• 不規則問題の記述

有限要素法など，配列の間接参照や不規則データ構造をもつ問題に対してデータ分割をうまく記述できない．

• 通信最適化

前章で，種々の通信最適化の技法について説明

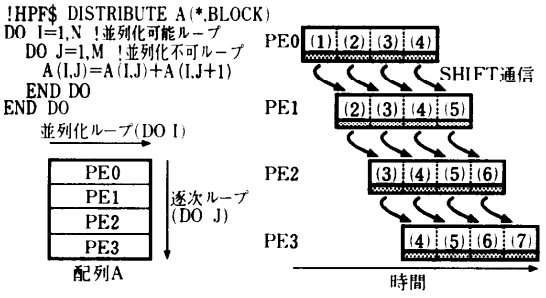

図-5 パイプライニングによる通信最適化

したが，複雑なケース，特に複数の手続きにまたがってこれらの最適化を適用するのが非常に困難である．一方で，HPFでは，計算処理のマッピングは，処理系に任され，ユーザから制御できない．したがって，この計算マッピングに依存する通信の最適化をユーザから指示することができない．

• タスク並列性

HPFで記述できる並列性の指定は，INDEPENDENT指示行で記述できるDOALL型のループ処理とFORALL型のループ処理および配列演算だけである．これ以外のDO ACROSS型の並列化や，制御並列を含むようなタスク並列性の指定ができない．

• 並列I/O

大規模演算においては，大規模データ(ファイル)を扱うことが多く，入出力が並列化の妨げになることが多い．これを解決するための並列I/Oを記述する方法がない．

• 手続き間解析の問題

分散メモリ上の並列化は，グローバルな情報が必要であるが，これには手続き間解析が不可欠である．大学などで研究用に開発された処理系や一部の商用システムでは，すでに手続き間解析が採用されているが，まだまだ課題は多い．たとえば，これらのシステムでは，大規模ソフトウェアを開発する際に，1つのファイルを更新すると，全ファイルを再コンパイルする必要がある．また，詳細で正確な手続き間解析には，長時間の解析時間と大量のメモリが必要になる．

• コンパイル時に不明な変数，データサイズ

並列化においては，最適化の方法や，最適化のパラメータがデータサイズやループ長に依存することが多い．選択肢が2,3個の場合には，マルチバージョン方式といって，複数の最適化コードを生成しておき，実行時にこれらを選択実行する手法がとられるが，選択肢が多数あったり，複数のパラメータが相互にからみあう場合には，使えない．

• アドレス変換オーバヘッド

ある配列が，ループの繰返しとともに，リモートデータ／ローカルデータをまたがってアクセスされ，かつ，これらのデータを静的に連続アドレス領域に確保できない場合，配列アクセスごとに必要なアドレス変換にIF文による分岐が必要になるか，またはローカルデータを移送する必要が生ずる．イタレーションスプリッティングである程度はカバーできるが十分ではない．

• 繰返し構造の把握

INSPECTOR/EXECUTOR方式は，INSPECTORのオーバヘッドが非常に大きいため，この適用には，プログラムに繰り返し構造があり，通信スケジュールが，繰り返し再利用できる必要がある．しかし，このためには，スケジュールの再利用可能性をコンパイラに知らせる手段が必要となるが，現在のHPFにはこの機能がない．

5. HPF2の言語拡張案

HPF2.0の正式な仕様は，一部の例外を除いて，HPF1.0のサブセットとなっている．これは，仕様を縮小し，安定した処理系の供給を促進する狙いである．一方で，前章で述べた種々の問題に対応するため，公認拡張仕様(Approved Extensions)を定めている．現時点では，1996年10月21日に公開された仕様案Version 2.0.δ [6]が最も新しいが，11月のSupercomputing 96で正式に発表される予定である．以下では，この公認拡張仕様について説明する．

5.1 拡張分散指示

不規則問題の記述のために，各PEに割りつける分割の幅を任意に指定できるGENERALIZED BLOCKと，割りつけリストを用いて，任意のマッピングが記述できるINDIRECTの2つのマッピング方法が定められた．

```
  PARAMETER (S=/2,25,20,0,8,45/)
!HPF$ PROCESSORS P(6)
  REAL A(100)
!HPF$ DISTRIBUTE A(GEN_BLOCK(S)) &
!HPF$ PROC
```

この例は，配列AをGENERALIZED BLOCKを用いて分割する例であり，定数配列Sの各要素S(i)でプロセッサiに割りつけられる要素数が指定される．

```
!HPF$ PROCESSORS P(4)
  REAL A(100),B(50)
  INTEGER map1(100)
  PARAMETER (map1=/1,3,4,3,3,2,1,4,.../)
!HPF$ DISTRIBUTE A(INDIRECT(map1)) &
```

!HPF$ ONTO P

こちらは，割りつけリストを用いて分散を指定する例であり，A(i)の要素がプロセッサ配列Pの要素P(map1(i))にマッピングされる．

また，プロセッササブセットを定義する機能とこれへのマッピングの指定，およびSHADOW WIDTHと呼ばれる，PE境界のデータをオーバラップしてもたせる機能が定められた．プロセッササブセットの定義機能は，後述のタスク並列性の記述においても重要な役割を果たす．

5.2 コンピュテーションマッピングの明示的指定

ON節(ON Clause)を用いて演算処理を実行するプロセッサを指定することができる．

!HPF$ ON HOME(A(I))

本指定により，後続の文が，A(I)を所有するPEで実行される．また，

!HPF$ ON PROC(I)

は，後続の文が論理プロセッサPROC(I)で実行されることを示す．上記2つの例において，Iの値は，実行時に指示行の場所に制御が移った時点で評価される．また，ON節を，BEGIN ～ ENDで囲まれたブロックに対して指定することもできる．

5.3 通信最適化の指定

通信最適化を記述するためにRESIDENT指示行が用意されている．これは，ON指示行に付加する形で記述し，後続文がON指示行によって指定されたプロセッサで実行するかぎり，RESIDENT以下で指定される配列のアクセスに対して通信が必要ないことを示す．

(RESIDENTの例)

```
!HPF$ INDEPENDENT
  DO J=1,N
  !HPF$ ON HOME(IX(J)),RESIDENT(Y)
  X(J) = Y(IX(J)) - Y(IY(J))
  END DO
```

この例では，ループの各繰返しは，IX(J)を所有するプロセッサで実行され，配列Yの参照が通信なしに行えることを示している．

ただし，場合によっては，ある配列名ではなく，特定の配列参照に対して，通信が不要であることを示したい場合がある．このため，RESIDENT指示行では，RESIDENT以下に特定の配列アクセスを指定することができる．この場合，ON節が有効なブロックに対して，RESIDENT以下で指定された配列参照と字面上の一致が検索され，これらに対して，RESIDENT指示が有効となる．

5.4 タスク並列性の指定

タスク並列性は，前節で述べたON節とRESIDENTを用いて指定する．TASK REGION ～ END TASK REGION指示行で囲まれたブロックでは，ON節で指定された単位(ON BLOCKと呼ぶ)をタスクとして並列性が記述できる．すなわち，各PEが，自分が実行すべきON BLOCKを選択的に実行することにより，並列実行が実現できる．各ON BLOCKにおいてアクセスできるデータは，ローカルデータのみである．これは，RESIDENT指示行により，明示的に指示する必要がある．ここでローカルデータとは，ON BLOCKを実行するプロセッサ集合が，参照データについては，少なくとも，1つの複製を所有するもので，更新データについては，本プロセッサ集合だけが所有するもののことである．特定のプロセッサ間でデータの通信をしたい場合には，この両者を含むON BLOCKを定義すれば，この中で可能である．また，プロセッササブセットへデータをマッピングすることにより，タスクローカルなデータを定義することができる．TASK REGIONを入れ子構造にすることにより，木構造の探索問題などの並列性を記述できる．また，パイプライン並列性の記述も可能である．詳細については，仕様書[6]を参照されたい．

5.5 そ の 他

上記以外のHPF2の特徴には，以下のものがある．

- 動的再配置機能

HPF1.0の動的再配置の機能が，HPF2の基本仕様から削られ，Approved Extensionの機能となった．

- 手続き間データマッピング

手続きごとの分割コンパイルで手続き間のデータマッピングを最適化できるよう，特定の場合を除いて，呼出側の手続きで，呼び出す手続き内の仮引数のマッピングを明示的に指定することが義務づけられた．指定は，Fortran90のInterface BlockやModuleの機能を用いて行う．

- NEW属性による，並列化ループ内のワーク変

数の明示的指定.（基本仕様）
- Reduction 演算の明示的指示(基本仕様)
- 非同期 I/O (Approved Extension)

6. HPF の将来

HPF 言語は，1991 年に HPFF が組織され，言語仕様の検討がスタートしてからすでに約 5 年経過し，ようやく実用レベルの処理系が利用できる状況となった．これまでは，PGI, APR, DEC, PSR, NA-Software といった欧米のソフトウェアベンチャーが製品化の主役であったが，国産スーパーコンベンダ各社も，製品化もしくは，それに近いフェーズにある．規則性の高いプログラムは，これらを用いて簡単に並列化できる．

一方で，現在の HPF は，広範な実応用プログラムの並列化に適用するには，まだまだ機能が不足しているのも事実である．とくに，有限要素法などで必要となる非定型な問題の処理，ループ以外の並列性の抽出，手続き呼出しをまたがって記述された並列性の抽出および PE 間通信の最適化などが広範な実用問題への適用を阻害している．言語仕様の拡張とコンパイル技術の両面からさらなるブレークスルーが必要である．

当面の課題は，公認拡張仕様を含めた HPF2.0 の処理系の早急な開発である．HPF2.0 の安定した処理系が使えるようになれば，実用プログラム並列化への適用性はかなり向上する．また，デバッガや性能アナライザなどのプログラミングツールの充実も重要である．現在のところ，HPF を Fortran へのトランスレータの形式で実装しているシステムが多いため，HPF のソースレベルでのデバッガやツールの開発が遅れているが，これらを早急に整備する必要がある．

また，NUMA システム(Non-Uniform Memory Access の略．物理的には分散メモリマシンだが，他 PE のメモリを，低速ながら自 PE メモリと同一インタフェースでアクセスできるシステム)に HPF などのデータパラレル言語を適用する研究が最近各地で行われている．代表的なものに，Stanford 大学の SUIF コンパイラ[7]がある．これらのシステムは，純粋に共有メモリマシンとして用いると，データ転送が頻発して期待された性能を得られないことが多いが，ユーザ指定の分散メモリ上のデータレイアウト情報を利用することで通信を削減し，並列実行の効率を高めることができる．要するに，通信の最適化をコンパイラと NUMA システムが提供する共有メモリアクセス機構で分担することにより，両者の利点を生かそうというアプローチである．

さらに，現状では，MPI と HPF は，独立した並列プログラミングインタフェースとして存在しているが，これらを統合したプログラミング環境の整備も重要である．性能的にクリティカルな部分だけを MPI で記述することにより，HPF/MPI のそれぞれの欠点，すなわち実行効率およびプログラムの記述容易性を相補できる．HPF では，このために，HPF_LOCAL などの HPF から MPI プログラムを呼び出すためのインタフェースを提供しているが，まだまだ十分ではない．プログラミングツールを含めた両者統合環境の開発が待たれる．

7. むすび

分散メモリ型の並列計算機を簡単に使えるためのプログラミング言語 HPF について，その基本仕様，処理系に要求される機能，現在検討がすすんでいる HPF2.0 の仕様，そして，将来展望について述べた．

HPF のベースとなる Fortran-D や Vienna Fortran の言語仕様は，自動並列化コンパイラの研究者たちにより作成された．この目的は，自動並列化でどうしてもうまくいかない部分をユーザに指定させることにあった．この流れをうけて，HPF の言語仕様も，並列化コンパイラの研究開発を行う技術者を中心として検討されてきた．このため，HPF の仕様が，本当にユーザに使いやすいものになるためには，ユーザが HPF を実際に使ってみて，利用に際して不十分な点を，言語仕様や，処理系にフィードバックする必要がある．とくに，現時点では，ユーザが，言語仕様のみならず，処理系の動作を熟知していないと効率のよい並列プログラムを書けないことが多い．また，記述方法によって，同一アルゴリズムのプログラムの性能が大きく変わることも稀ではない．この問題は，初期のベクトル化コンパイラがそうであったように，ユーザと処理系開発者がいっしょになって実用プログラムの並列化経験を重ねることにより段階的に解決していく必要がある．ヨーロ

ッパでは，この目的で，PHAROSやHPF+(http:// www.vcpc.univie.ac.at/ 参照)といったECのESPRITプロジェクトがスタートしている．

冒頭でも述べたが，分散メモリ並列システムを並列処理専門家から，一般ユーザの計算ツールとするためには，HPFのような，記述レベルの高いプログラミングインタフェースが不可欠である．また，ユーザによるデータレイアウト指定＋コンパイラの並列化で分散メモリ上並列化を実現するというHPFの基本思想は，非常に自然なものである．Fortranの規格を検討するISO SC22/WG5やANSI X3J3では，2000年に改正予定のFortran規格にHPFの主要機能を取り込む検討を開始した．まだまだ課題は多いが，HPFの研究に携わるものの1人として，HPFが1日でも早く，真の意味での標準インタフェースとなることを願ってやまない．

参考文献

1) High Performance Fortran Forum: High Performance Fortran Language Specification, Version 1.0, Technical Report CRPC-TR92225, Center for Research on Parallel Computation, Rice University (1992).
2) Koelbel, C. et al.: The High Performance Fortran Handbook, 329 p., The MIT Press, Cambridge, Mass. (1992).
3) Gupta, M., Schonberg, E. and Srinivasan, H. : A Unified Data-Flow Framework for Optimizing Communication, Proceedings of the 7th Workshop on Languages and Compilers for Parallel Computing, Ithaca, NY (Aug. 1994).
4) Das, R., Saltz, J. and von Hanxleden, R. : Slicing Analysis and Indirect Access to Distributed Arrays, Proceedings of the 6th Workshop on Languages and Compilers for Parallel Computing, Portland, OR (Aug. 1993).
5) Tseng, C.-W. : An Optimizing Fortran D Compiler for MIMD Distributed-Memory Machines, Ph.D Thesis, Rice University (Jan. 1993).
6) High Performance Fortran Forum: High Performance Fortran Language Specification, Version 2.0.γ (1996).(http://www.crpc.rice.edu/HPFF/hpf2/index.html)
7) Amarasinghe, S. P., Anderson, J. M., Lam, M.S. and Tseng, C. W. : The SUIF Compiler for Scalable Parallel Machines, Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing (Feb. 1995).

(平成8年11月10日受付)

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．とく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言語環境，並列アルゴリズムに興味をもつ．現在C&C研究所主任．工学博士．1988年本会論文賞受賞．e-mail: seo@csl.cl.nec.co.jp