ハイパフォーマンス
コンピューティング 62－9
（1996．8．28）

High Performance Fortranトランスレータの機能概要

吉川　聡　　根岸　清　　佐藤　真琴*　　太田　寛*
黒澤　隆**　　小林　篤**　　会田　一弘　　布広　永示

（株）日立製作所ソフトウエア開発本部
*（株）日立製作所システム開発研究所
**日立東北ソフトウエア株式会社

並列計算機に対するプログラミング言語として High Performance Fortran(HPF) が提案されている。本稿では、HPFプログラムをFortranのSPMDプログラムに変換するトランスレータについて述べる。この処理系は、HPFの全仕様をサポートする。また、データ分散、再分散機能、計算分割機能、通信最適化機能を持ち、さらに処理系独自の拡張指示文をサポートすることにより、効率的な並列プログラムの作成を可能にする。

A summary of a High Performance Fortran Translator

Satoru Yoshikawa　　Kiyoshi Negishi　　Makoto Sato*　　Hiroshi Ohta*
Takashi Kurosawa**　　Atsushi Kobayashi**　　Kazuhiro Aida　　Eiji Nunohiro

Software Development Center, Hitachi Ltd.
*Systems Development Laboratory, Hitachi Ltd.
**Hitachi Tohoku Software Ltd.

High Performance Fortran(HPF) is proposed as a programming language for parallel computers. This paper describes a language processor which translates HPF programs into Fortran SPMD programs. It supports full HPF specifications. It has many features, such as data distribution, data redistribution, computation partitioning, and communication optimization. Furthermore, it supports original directives. Users can develop efficient parallel programs by using those features.

1．はじめに

プロセッサ単体の性能向上が物理的限界に近づきつつある近年、分散メモリ型超並列計算機に対する期待はますます大きくなっている。分散メモリ型超並列計算機上でのプログラミングは、データの分散、プロセッサ間の通信管理などが必要であり、従来の逐次処理を前提としたプログラミングとは異なった困難が伴う。プログラマの負担を緩和するため、並列計算機に対応した科学技術計算向けプログラミング言語仕様としてHigh Performance Fortran [1]（以下HPF）が提案されている。HPFは、Fortran90プログラムにデータ分散指示文を挿入することにより、並列プログラムを記述する言語仕様である。

本稿で紹介するHPF処理系は、HPFのフルセットを包含した言語仕様を入力とする言語処理系であり、通信関数を含むFortran90のSPMD (Single Program Multiple Data Stream) プログラムを出力する。

本稿では２節で処理系の特徴、３節で構成、４節で拡張仕様、５節で最適化について述べる。

2．処理系の特徴

（1）HPFのフルセットをサポート

全てのHPF指示文、FORALL文、HPFライブラリ等、HPF Language Specification Ver. 1.0で規定されているHPFのフルセット機能をサポートしている。

（2）Fortran90仕様のサポート

HPFはFortran90を拡張してデータ分散指示文やFORALL構文等を付け加えたものである。Fortran90の仕様をサポートする事により、配列演算機能、動的記憶割付機能、インタフェースブロック等、HPFの記述および処理系の実現に有効な機能を使用することができる。

（3）拡張仕様

本処理系では、HPFで定められた仕様を補完するために、独自の拡張仕様を設けている。これには、HPF2[2]で新たに提案されているような指示文が含まれる。

（4）トランスレート方式

本処理系はHPF指示文を含んだFortranプログラムを通信関数呼出しコードを含むFortran 90プログラムに変換するトランスレータである。

（5）再分散指示のトランスレート時解析

本処理系は、HPFのREALIGN/REDISTRIBUTE指示文やALLOCATE文を解析して、プログラムの任意位置での変数の分散情報をトランスレート時に得る。これにより、再分散指示文を含むプログラムに対しても、通信最適化などが適用できる。トランスレート時に分散情報が不明な場合は、分散情報を格納した構造体を用いて、実行時に再分散ルーチンで分散する。

（6）計算分割および通信の最適化

種々の通信最適化に加え、DOACROSSループに対する通信最適化や、ループ内のIF文を削減する計算分割の最適化を行い、処理系による並列化は困難なループに対しても高速化を実現する。以下に、最適化項目の一覧を挙げる。

- ●ループ範囲縮小
- ●ループ分割
- ●ループインデックス範囲分割
- ●ガードのループ追い出し
- ●通信ベクトル化
- ●通信融合
- ●タイリング
- ●ブロックストライド通信の利用
- ●冗長通信の削除

3．処理系の構成

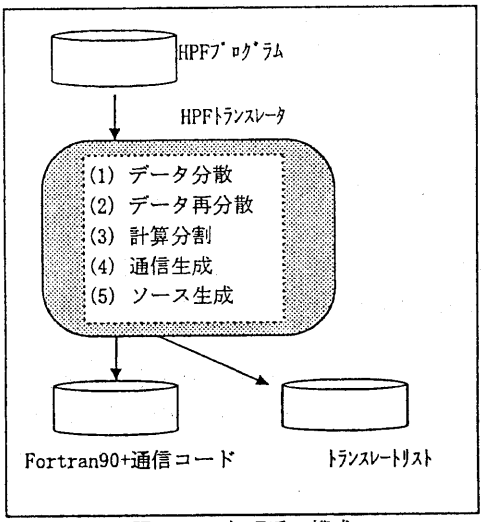

図１　HPF処理系の構成

図１は、HPF処理系の構成である。

（1）　データ分散

並列化指示文に従って、配列データが各プロセッサに分散されるよう変換する。
手続き間での分散形状の引き継ぎは、分散形状に関する情報を保持する構造体を各変数毎に生成し、手続きの引数を拡張してこの構造体を渡す操作を行うことによって実現する。

（2） データ再分散

HPFのREALIGN/REDISTRIBUTE指示文やALLOCATE文を解析して、プログラムの任意位置での変数の分散情報を得る。

（3）計算分割

DOループやFORALLを基本的にowner compute rule [3] (以下、OCRと呼ぶ)に従い分散する。また、総和計算などのイディオム認識を行う。

（4）通信生成

データ分散の結果、各プロセッサで必要になるプロセッサ間通信を解析し、通信コードを生成する。通信形態に応じて、SEND/RECEIVE型通信とBROADCAST型通信を使い分ける。各種の通信最適化も行う。

（5）ソース生成

解析結果を元にソースプログラムを出力する。出力されるソースは、通信ライブラリの呼び出しコードを含むFortran90のSPMDプログラムである。また、トランスレート時に得られた情報をトランスレートリストとして出力する。ユーザはこれを使用して、より効率的な並列プログラムを作成できる。

4．拡張仕様

本処理系では、HPFで定められた指示文の他に、表1に示す指示文をサポートしている。

表1 HPFトランスレータの拡張指示文

OVERLAP指示文	配列のオーバーラップ領域のサイズを指示。
REDUCTIONオプション	independent指示ループ中のリダクションを指示。
LOCAL_PURE指示文	当該手続きが他プロセッサの分散データにアクセスしないことを指示。

以下の説明で、拡張指示文は「!PFD$」で始まる行に記述する。

（1）OVERLAP指示文

隣接通信型プログラムでは、配列を各プロセッサにブロック分散し、各プロセッサのローカル配列の端に、データ受信用領域(オーバーラップ領域)を設ける手法がよく用いられる。OVERLAP指示文はこの領域の大きさを指示する。

例えば、2次元配列をブロック分散し、第1次元は両端に1要素ずつ、第2次元は下端に2要素のオーバーラップ領域を設けるときには、次のような指示を記述する。

```
!HPF$ DISTRIBUTE A(BLOCK,BLOCK)
!PFD$ OVERLAP A(1:1,2:0)
```

図2は、このときのローカル配列の状態を示す。

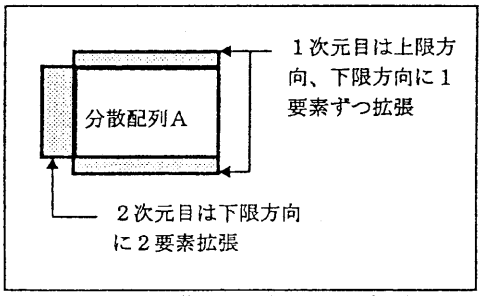

図2 OVERLAP指示文による配列の拡張

OVERLAP指示文は、手続き間にまたがる配列に対して特に効果がある。本指示文がないとき、トランスレータの解析結果によって確保されるオーバーラップ領域の大きさは、手続き毎に異なる可能性がある。このため、トランスレータは手続きの先頭でリマッピングなどの処理を行う必要がある。

OVERLAP指示文によって、ユーザはオーバーラップ領域の大きさを手続きにまたがって一致させることができ、リマッピングなどのオーバーヘッドを削減することができる。

なお、HPF2では、DISTRIBUTE指示文の一部として、オーバーラップ領域を指示する仕様が提案されている。その場合、一つのテンプレートにマッピングされている複数の配列に対して、必ず同じ大きさのオーバーラップ領域を取ることになる。一方、本処理系のOVERLAP指示文は、同一テンプレートにマッピングされている個別の配列毎に、異なる大きさを指示することができる。

（2）REDUCTIONオプション

REDUCTIONオプションはINDEPENDENT指示文のオプションとして使用する。HPFで定められているINDEPENDENT指示文は、直後のループの各繰り返しが独立であることを指示する。しかし、ループ内に総和などのリダクション演算が含まれていると、繰り返しにまたがった依存関係があるため、INDEPENDENT指示ができない。

本オプションはリダクション演算以外の処理に関しては、ループの各繰り返しが独立であることを指示する。この指示により、トランスレータは、各繰り返しを独立に実行した後で、大域演算用の通信関数を使用してリダクション演算を実現するコードを出力する。

例えば、変数Sに対する総和を除けばループが独立である場合、次のような指示を記述する。

```
!PFD$ INDEPENDENT, REDUCTION:SUM(S)
      DO I=･･･
         :
        S=S+･･
      ENDDO
```

なおHPF2でも、INDEPENDENT指示文に対するREDUCTIONオプションが提案されている。
本処理系のREDUCTIONオプションは、リダクションの種類をユーザが指示できる点が異なる。これにより、解析によってリダクションの種類を特定するのが困難な場合であっても、処理系は適切なコードを出力することができる。

（3）LOCAL_PURE 指示文

HPFで規定されているPURE手続きは、その手続きの呼び出しによって他に副作用を与えない手続きである。しかし、PURE手続き内での分散配列に対する参照は可能であり、手続き内で通信が必要となる可能性がある。メッセージパッシング型の並列計算機に対しては、このような手続き呼び出しを含むループを処理系が分散するのは困難である。

そこで、分散されたデータを参照しないPURE手続きを示す指示文として、LOCAL_PURE指示文を導入した。指示文の形式は次の通りである。

```
!PFD$ LOCAL_PURE
```

本指示文は、当該手続きの宣言部、および、呼び出し元手続きにおける当該手続きに対するインタフェースブロックに記述する。

5．最適化機能

本節では、2節で紹介した最適化機能の内、本処理系に特徴的なものを説明する。
5．1では、ループ内のIF文を削減する2つの最適化方法を説明する。5．2では、通信自身を削除する、あるいは、通信オーバヘッドを削減する4つの最適化を説明する。

5．1 計算分割最適化

計算分割は、データ分割に従ってループ計算を各プロセッサに分散させる処理である。
計算分割は基本的にOCRを利用する。このため、ループ内代入文の左辺配列の ownerプロセッサが、全てのループインデックスに対して互いに等しければ、各ループイタレーションは一つのプロセッサが実行することになる。このとき、さらにループイタレーションとそれを実行するプロセッサ番号との間に規則性があると、ループが各プロセッサに分散される。

本節では、ループ内代入文の左辺配列のownerプロセッサが、必ずしも等しくない場合と、全てのループインデックスに対してただ一つに決まる場合の最適化を説明する。

5．1．1 ループインデックス範囲分割

ループ内に複数の代入文があり、それらの左辺配列の owner プロセッサが、必ずしも等しくない場合、ループ分散は一般に困難になる。このような場合、全てのインデックス範囲を実行するループ内で、各文を実行すべきか否かを判定するIF文を生成するコード生成方式がある。これに対し、ループインデックス範囲分割は、各文が実行されるインデックス範囲を解析してループのインデックス範囲を細分し、細分された各範囲内ではIF文が不要になるようにする処理である[4]。

図3は、左辺配列の分散方法が互いに異なるループの例である。

図4は、図1のプログラムを単純に並列化した結果である。

図5は、図1のプログラムにループインデックス範囲分割を適用した結果である。
図5では、プロセッサによってループインデックス範囲が異なること、3ループの中の実行文が異なることに注意されたい。

```
      REAL A(100),B(100)
!HPF$ PROCESSORS P(10)
!HPF$ DISTRIBUTE (BLOCK) ONTO P :: A, B
      DO J = 1, 99
        A(J) =
        B(J+1) =
      ENDDO
```

図3　ownerが異なる分散配列への代入式の例

```
      REAL A(10),B(10)
      DO J = 0,10
        IF(0 < J) THEN
          A(J) =
        ENDIF
        IF(J < 10) THEN
          B(J+1) =
        ENDIF
      ENDDO
```

図4　図3のプログラムの並列化例

```
      REAL A(10),B(10)

      IF (PE = 0) THEN
        LOW1 = 1; UPP1 = 0
      ! ループは実行されない (LOW1 > UPP1)
        LOW2 = 0; UPP2 = 0
        LOW3 = 0; UPP3 = 0
      ENDIF
      IF (0 < PE < 9) THEN
        LOW1 = 1; UPP1 = 9
        LOW2 = 1; UPP2 = 9
        LOW3 = 1; UPP3 = 9
      ENDIF
      IF (PE = 9) THEN
        LOW1 = 10; UPP1 = 10
        LOW2 = 10; UPP2 = 10
        LOW3 =  1; UPP3 =  0
      ! ループは実行されない (LOW3 > UPP3)
      ENDIF

      DO J = LOW1, UPP1
        B(J+1) =
      ENDDO
      DO J = LOW2, UPP2
        A(J) =
        B(J+1) =
      ENDDO
      DO J = LOW3, UPP3
        A(J) =
      ENDDO
```

図5　ループインデックス範囲分割

5．1．2　ガードのループ追い出し

ループ内代入文の左辺配列の OWNER プロセッサが、全てのループインデックスに対してただ一つに決まる場合、このループ自体、一つのプロセッサが実行することになる。
この時、本処理はプロセッサ判定用のIF文をループの外側に出して、IF文の判定オーバヘッドを削減する。

図6は、ループ内代入文の左辺配列の OWNER プロセッサが、ただ一つに決まる例である。

図7は、図1のプログラムにガードのループ追い出しを適用した結果である。

```
!HPF$ PROCESSORS MP(10)
!HPF$ DISTRIBUTE (*,BLOCK) ONTO MP :: A
      DO J = 1, N
        A(J,1) = A(J,N)
      ENDDO
```

図6　特定プロセッサ上の配列への代入

```
      IF (PE = 9) THEN
        SEND(A(1:N, N), 0)
      ENDIF
      IF (PE = 0) THEN
        RECV(TMP(1:N), 9)
        DO J = 1, N
          A(J,1) = TMP(J,9)
        ENDDO
      ENDIF
```

図7　ガードのループ追い出し適用例

5．2 通信最適化について

通信オーバヘッドは数百マシンサイクル以上かかるため、通信最適化は非常に重要である。本節では、冗長な通信の削除、通信回数の削減を行う最適化を説明する。

5．2．1　冗長通信の削除

一度、受信した配列要素は、新たに値が定義されなければ繰り返し使用することが可能である。冗長通信の削除は、一旦、全ての通信を内部的に生成し、同じ位置にある通信で、送受信プロセッサが同じものについて、それらの通信データの一方が他方を含む場合、含まれる方の通信を削除するものである。

図8は、再使用可能な参照を含むループの例である。

図9は、図1のプログラムに冗長通信の削除を適用した結果である。

```
    REAL A(100,100),B(100,100),C(100,100)
!HPF$ PROCESSORS MP(10)
!HPF$ DISTRIBUTE (BLOCK,*) ONTO MP :: A,B,C
    DO J = 1, 100
      DO I = 1, 100
        B(I,J) = A(I+1,J)
        C(I,J) = A(I+1,1)
      ENDDO
    ENDDO
```

図8　再使用可能参照を含むプログラム

```
    SEND(A(0,1:100), PE-1)
    RECV(A(11,1:100),PE+1)
    DO J = 1, 100
      DO I = 1, 10
        B(I,J) = A(I+1,J)
        C(I,J) = A(I+1,1)
      ENDDO
    ENDDO
```

図9　冗長通信削除適用例

5.2.2 通信融合

複数の通信を1つの通信に融合することにより、通信回数を削減する。複数の配列に対する通信の融合と、1つの多次元配列内の非連続な領域に対する通信の融合の2つの場合に適用する。いずれも一時配列をバッファとして用いる。

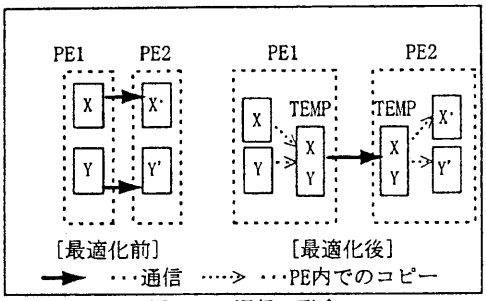

図10 通信の融合

5.2.3 タイリング

ループの繰り返しの間に実行順序関係のあるDO ACROSSループを並列化すると、ループ内で通信を行いながら、パイプライン的に実行を行う。このとき、複数の繰り返しをまとめて適切な粒度を選択することにより、通信回数の削減と、それによる実行の逐次化のトレードオフを解決する[5]。

タイリングは、完全交換可能な密多重ループに対して、分散/非分散ループがそれぞれ、少なくとも一つある場合に適用する。

図11は、DOACROSSループの例である。

図12は、図11のプログラムにタイリングを適用した結果である。

```
!HPF$ PROCESSORS MP(10)
!HPF$ DISTRIBUTE (BLOCK,*) ONTO MP :: A
    DO J = 1, N
      DO I = 1, N
        A(I,J) = A(I-1,J)+...
      ENDDO
    ENDDO
```

図11 DOACROSSループの例

```
      DO JT = 1, N, TILE
       JUB=MIN(JT+TILE-1,N)
       RECV(A(0,JT:JUB),PE-1)
       DO J = JT, JUB
         DO I = 1, N/10
           A(I,J) = A(I-1,J)+...
         ENDDO
       ENDDO
       SEND(A(10,JT:JUB),PE+1)
     ENDDO
```

図12 タイリング適用例

6. 評価

並列計算機の代表的ベンチマークであるNAS Parallel Benchmarks を本処理系によって並列化し、評価を行ったところ、良好なスケーラビリティを得た。

7. おわりに

本稿では、HPFトランスレータについて概説した。特に、効率的な並列プログラムを作成するために必要な3つの拡張指示文と、計算分割や通信生成における最適化を中心に説明した。

参考文献

[1]High Performance Fortran Forum. "High Performance Fortran Language Specification Version 1.0," May, 1993

[2] High Performance Fortran Forum. "High Performance Fortran Language Specification Version 2.0.α.2," May, 1996

[3]S. Hiranandani, K.Kenedy, C.-W. Tseng. "Compiling Fortran D for MIMD Distributed-Memory Machines", Communications of the ACM, Vol.35, No.8, August 1992.

[4]M. Sato, T. Hirooka, K. Wada, F. Yamamoto. "Program Partitioning Optimizations in an HPF Prototype Compiler", Compsac'96 (to appear).

[5]H. Ohta, Y. Saito, M. Kainaga, H. Ono. "Optimal Tile Size Adjustment in Compiling General DOACROSS Loop Nests", ICS'95, pp.270-279, 1995.