HPF/VPP 入門 (2)

浅岡香枝 *　平野彰雄 *

1　はじめに

VPP800 の HPF コンパイラ HPF/VPP を使うための解説記事として、前号から「HPF/VPP 入門」というタイトルで連載を始めています。

初回は、 HPF の概要と HPF 指示文、コンパイルおよび実行方法について解説しました。今回は、手続きの扱いについて解説します。

2　手続き間でのデータマッピング

2.1　手続き毎の分散指定と手続引用仕様宣言

通常、プログラムは複数の手続きで記述され、手続き単位に分割コンパイルされます。配列のデータマッピングは、手続き毎に指定しますので、分散配列を引数で渡す場合には、実引数に対応する仮引数のマッピング状態を手続きの呼出し元に教えるために、**手続引用仕様宣言 (interface block)** を記述する必要があります。

手続きの呼出しを含む例をプログラム例 1 に示します。

```
!-------- プログラム例 1 ----------!
      program main
      integer,parameter :: n=100
      integer :: i
      real(8) :: a(n)
!hpf$ processors p1(4)
!hpf$ distribute a(block) onto p1
            :
!!!!! 手続引用仕様宣言 !!!!!
      interface
       subroutine sub(a)
        integer,parameter :: n=100
        real(8) :: a(n)
!hpf$ processors p1(4)
!hpf$ distribute a(block) onto p1
       end subroutine sub
      end interface
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
            :
!hpf$ independent,new(i)
      do i=1,n
!hpf$ on home(a(i)) begin
        a(i)=real(i,8)
!hpf$ end on
        end do
            :
      call sub(a)
            :
      end program main

      subroutine sub(a)
      integer,parameter :: n=100
      integer :: i
      real(8) :: a(n)
!hpf$ processors p1(4)
!hpf$ distribute a(block) onto p1
            :
!hpf$ independent,new(i)
      do i=1,n
!hpf$ on home(a(i)) begin
        a(i)=a(i)+1.0d0
!hpf$ end on
      end do
      end subroutine sub
!--------------------------------!
```

手続き呼出し元の `main` にある `interface` と `end interface` で囲まれた部分が手続引用仕様宣言であり、呼出し先サブルーチン `sub` の引用仕様が明記されています。手続引用記述宣言には、配列のデータマッピングを指定する HPF 指示文も記述します。

2.2　不必要な再マッピングの防止

手続きの呼出し境界では、**手続きの呼出し時**と**手続きからの復帰時**に、配列データの再マッピングが行われます。

本来、実引数と仮引数の分散が同じであれば、再マッピングは必要ありません。しかし、 HPF/VPP では、実引数と仮引数の分散が同じであるかないかに関わらず、再マッピングが発生します。

実引数と仮引数の分散が同じである場合、再マッピングを防ぐには次のことをします。

プログラム例 1 の仮引数に対する `distribute` 指示文の書き方は、**指令的**といいますが、まずこれを**記述的**な書き方に変更します。

* あさおかかえ、ひらのあきお (京都大学 学術情報メディアセンター)

```
!hpf$ distribute a *(block) onto *p1
```

変更点は、分散形式とプロセッサ構成の前に*(アスタリスク) がついていることです。

次に、コンパイル時に -Wh のサブオプションとして -noremap オプションをつけてコンパイルします。

```
% frt -Wh,-noremap sample.hpf
```

プログラム例１について、記述的な distribute 指示文により仮引数の分散を指定したものをプログラム例２に示します。

```
!-------- プログラム例２ ----------!
      program main
      integer,parameter :: n=100
      integer :: i
      real(8) :: a(n)
!hpf$ processors p1(4)
!hpf$ distribute a(block) onto p1
            :
!!!!! 手続引用仕様宣言 !!!!!
      interface
       subroutine sub(a)
        integer,parameter :: n=100
        real(8) :: a(n)
!hpf$ processors p1(4)
!hpf$ distribute a *(block) onto *p1
       end subroutine sub
      end interface
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
            :
!hpf$ independent,new(i)
      do i=1,n
!hpf$ on home(a(i)) begin
       a(i)=real(i,8)
!hpf$ end on
      end do
            :
      call sub(a)
            :
      end program main

      subroutine sub(a)
      integer,parameter :: n=100
      integer :: i
      real(8) :: a(n)
!hpf$ processors p1(4)
!hpf$ distribute a *(block) onto *p1
            :
!hpf$ independent,new(i)
      do i=1,n
!hpf$ on home(a(i)) begin
        a(i)=a(i)+1.0d0
!hpf$ end on
      end do
      end subroutine sub
!---------------------------------!
```

2.3　異なるデータマッピングの指定

例えば、配列の各次元方向に依存関係のあるループが交互に現れるようなプログラムでは、ある特定の手続きだけ並列化のために配列の分散を変更したい場合があります。このような場合は、指令的な distribute 指示文を用いてデータマッピングを指定します (プログラム例３)。

この例は、手続きの呼出し元の main では、配列 a を 2 次元目で分散していますが、サブルーチン sub では、配列 a の 2 次元目方向に依存関係のあるループがあるために 1 次元目での分散を指定しています。手続きの呼出し時と手続きからの復帰時には、配列 a について再マッピングが起こり、指定したとおりにマッピングが変更されます。

```
!-------- プログラム例３ ----------!
      program main
      integer,parameter :: n=100
      integer :: i,j
      real(8) :: a(n,n)
!hpf$ processors p1(4)
!hpf$ distribute a(*,block) onto p1
      interface
        subroutine sub(a)
         integer,parameter :: n=100
         real(8) :: a(n,n)
!hpf$ processors  p1(4)
!hpf$ distribute a(block,*) onto p1
        end subroutine
      end interface
            :
!hpf$ independent,new(i,j)
      do j=1,n
!hpf$ on home(a(:,j)) begin
        do i=2,n-1
          a(i,j)=a(i-1,j)+a(i,j) &
                        +a(i+1,j)
        end do
!hpf$ end on
      end do
            :
      call sub(a)
            :
      end program main
      subroutine sub(a)
      integer,parameter :: n=100
      integer :: i,j
      real(8) :: a(n,n)
!hpf$ processors  p1(4)
!hpf$ distribute  a(block,*) onto p1
            :
!hpf$ independent,new(i,j)
      do i=1,n
!hpf$ on home(a(i,:)) begin
        do j=2,n-1
           a(i,j)=a(i,j-1)+a(i,j) &
                          +a(i,j+1)
         end do
!hpf$ end on
      end do
      end subroutine sub
!---------------------------------!
```

3　外来手続き

3.1　手続き種別とその宣言

HPF では、通常の HPF プログラムだけでなく、HPF 以外の言語で記述された手続きや異なる実行モデルの手続きを呼出すために、外来手続き呼出し機能があります。

外来手続きの種別は、記述言語と実行モデルの組み合わせからなり、 HPF/VPP で扱うことができる種別は次の 3 つです。

- **HPF_GLOBAL**
- **HPF_LOCAL**
- **FORTRAN_LOCAL**

GLOBAL モデルは、通常の HPF プログラムの実行モデルで、 LOCAL モデルは、実行が各プロセッサに閉じた実行モデルです。デフォルトの手続き種別は、 HPF_GLOBAL です。

手続き種別の宣言は、 extrinsic という外来プレフィックスを用いて宣言し、手続きの SUBROUTINE 文または FUNCTION 文の先頭に記述します。

各手続き種別の宣言は次のようになります。

- HPF_GLOBAL

  `extrinsic（'HPF','GLOBAL'）`
- HPF_LOCAL

  `extrinsic（'HPF','LOCAL'）`
- FORTRAN_LOCAL

  `extrinsic（'FORTRAN','LOCAL'）`

では次に、 HPF_LOCAL 手続きと FORTRAN_LOCAL 手続きについて例をあげて解説します。

3.2　HPF_LOCAL 手続き

HPF_LOCAL 手続きは、各プロセッサがそれぞれ自分のメモリ上にマッピングされているデータに対して処理を行います。分散配列を引数とした場合、各プロセッサ上にマッピングされている部分が手続きに渡り、渡された配列は手続き内においてグローバルインデックスではなくローカルインデックスでアクセスされます。

例えば、 a(100,100) の配列が 2 次元目で 4 つのプロセッサに分割されている場合、HPF_GLOBAL の手続きでは図 1のように見えますが、 HPF_LOCAL の手続きには、図 2に示すように、それぞれ a(100,25) の配列として渡され、各プロセッサ上でローカルなインデックス空間を用いてアクセスされます。

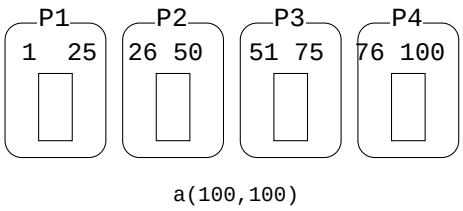

図 1. グローバルインデックスでのアクセス

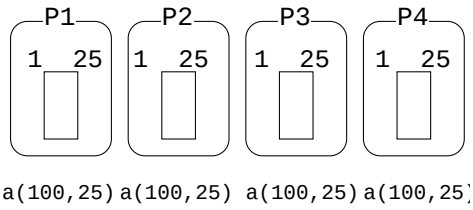

図 2. ローカルインデックスでのアクセス

次に HPF_LOCAL 手続きの例をプログラム例 4 に示します。この例は、 local_timer という HPF_LOCAL 手続きを定義し、手続き内ではタイマールーチンの system_clock を呼出しています。そして、 local_timer を使って、各プロセッサで独立してプログラムの一部分を計時し、最後に計時時間の全プロセッサでの合計および平均を求めています。

プロセッサ数と同じ大きさの配列 stim、 etim、 rate を宣言して分散し、 HPF_LOCAL 手続きに渡し、その配列に各プロセッサ毎の system_clock の結果を代入しています。

```
!---------- プログラム例４ ----------!
    program sample
    integer,parameter :: proc=4
    real(8) :: all_tim,avr_tim
    integer(4) :: stim(proc), &
                  etim(proc),rate(proc)
!hpf$ processors p(proc)
!hpf$ distribute stim(block) onto p
!hpf$ distribute etim(block) onto p
```

表 1. 各手続種別の呼出し関係

呼出し元	呼出し先		
	HPF_GLOBAL	HPF_LOCAL	FORTRAN_LOCAL
HPF_GLOBAL	○	○	○
HPF_LOCAL	—	○	○
FORTRAN_LOCAL	—	—	○

```
!hpf$ distribute rate(block) onto p
      interface
        extrinsic ('HPF','LOCAL') &
         subroutine local_timer(tim,rate)
         integer,dimension(:) :: tim,rate
         integer,parameter :: proc=4
!hpf$ processors p(proc)
!hpf$ distribute tim *(block) onto *p
!hpf$ distribute rate *(block) onto *p
        end subroutine local_timer
      end interface
      stim=0;etim=0;rate=0
     !!!! タイマー 起動 !!!
      call local_timer(stim,rate)
           : (計時範囲)
     !!!! タイマー 終了 !!!
      call local_timer(etim,rate)
      etim=etim-stim
      all_tim=real(sum(etim),8)/&
              real(rate(1),8)
      avr_tim=real(sum(etim),8)/&
       real(rate(1),8)/real(proc,8)
      write(*,*)"Sum:",all_tim,"sec"
      write(*,*)"Average:",avr_tim,"sec"
      end program sample

      extrinsic('HPF','LOCAL') &
         subroutine local_timer(tim,rate)
       integer,dimension(:) :: tim,rate
       call system_clock(tim(1),rate(1))
      end subroutine local_timer
  !-----------------------------------!
```

3.3 FORTRAN_LOCAL 手続き

FORTRAN_LOCAL 手続きは、Fortran で記述され、HPF/VPP コンパイラでなく Fortran コンパイラでコンパイルされた手続きです。

例をプログラム例５に示します。

```
  !---------- プログラム例５ ----------!
      program sample
      integer,parameter :: n=100
      integer :: i,t
      real(8) :: a(n)
!hpf$ processors p(4)
!hpf$ distribute a(block) onto p
      interface
       extrinsic ('FORTRAN','LOCAL') &
          real(8) function my_func(i)
          integer :: i
       end function my_func
      end interface
!hpf$ independent ,new(i,t)
      do i=1,n
        t=2*i
!hpf$ on home (a(i)) begin
        a(i)=my_func(t)
!hpf$ end on
      end do
      end program sample

      real(8) function my_func(i)
      integer :: i
      my_func=real(i,8)**2
      end function my_func
!-------------------------------------!
```

この例では、手続きの呼出し元に、各プロセッサ毎に独立して実行可能な do ループがあります。

do ループの中から Fortran で記述された関数 my_func を呼出しており、実引数 i と関数の結果が格納される t はループに閉じた変数なので、new 変数として指定することで各プロセッサごとに独立した領域がとられます。

このような関数は FORTRAN_LOCAL 手続きとして宣言することができます。これにより my_func を各プロセッサで独立して呼出すことができ、do ループが並列化されます。

4 おわりに

今回は、手続きに関する部分について紹介しました。すこしややこしくなりましたが、概要がおわかりいただけたでしょうか。

次回は、HPF 組込み手続き、HPF ライブラリ、また、ローカル HPF 手続きのためのライブラリについて紹介する予定です。