J. Plasma Fusion Res. Vol.82, No.11 (2006) 796-799

連載コラム High Performance Fortran で並列計算を始めよう

4．作ってみよう HPF プログラム(2)

岩下英俊，林 康晴[1)]，石黒静児[2)]
富士通　ソフトウェア事業本部，[1)]NEC　第一コンピュータソフトウェア事業部，[2)]核融合科学研究所

（原稿受付：2006年10月5日）

HPF/JA 仕様[1]の虎の巻，その2です．これで一通り，データ分散とループ並列化の，基本的な部分は終わりになります．

4.1　スカラ変数の使い方

4.1.1　スカラ変数は重複割付け

前回解説したとおり，HPF で並列化を行うためには，配列データは分散する必要があります．これに対して，スカラ変数は，原則として全プロセッサに同じものをコピーします．そうすれば，どのプロセッサでも通信なしで読むことができるからです．このような割付け方法を，分散に対して**重複**と呼びます．

重複割付けされた変数（重複変数）は，プロセッサ間で値がいつも一致しているように，自動的に制御されます．重複変数への代入があれば，普通は同じ書き込みが全プロセッサで行われます．ファイルから重複変数への入力があれば，全てのコピーに読み込まれます．こういった自動化によって，前回紹介した「グローバル空間」と「単一スレッド」のイメージが守られ，指示文を無視した逐次解釈との結果の一致が保障されます．

4.1.2　並列ループで更新されるスカラ変数

並列ループ実行中に，重複変数が更新されるときの振る舞いは少々複雑です．一時的に，それぞれのプロセッサが自分のコピーだけを更新し，プロセッサ間で値がばらつくという状況が起こるためです．

並列ループ内で更新できるスカラ変数は，種類は多くありません．DO 変数を除けば，NEW 変数か集計変数のどちらかであると考えてください．INDEPENDENT 指示文を記述する場合には，それぞれ，そのオプションである NEW 節と REDUCTION 節で宣言します．なお，並列ループ内で値が更新されない変数なら，これらの宣言は必要ありません．

集計変数については，次の節を割いて詳しく説明します．

NEW 変数は，ループの1回の繰り返しの中だけで読み書きして役目を終える変数です．Fig. 1 の例では，DO 変数 i から計算される変数（誘導変数）である k，内側 DO ループの DO 変数 j，そして一時的な作業変数 tmp が，NEW 変数として宣言されています．これらの変数の値は，並列ループ実行終了後に不定となることに注意してください．NEW節は，「1回の繰り返しを超えて値を保障する必要はない」という宣言ですから，繰り返しの外に出ると値は保障されません．

もし仮に，例えば変数 tmp について NEW 節による指示がないとすれば，繰り返しを跨いで（つまりプロセッサも跨いで）値を共有することになりますので，文法的には誤ったプログラムとなります[1]．しかし実際には，スカラの NEW 変数は頻繁に使いますし，いちいち指示するのは面倒です．そこで多くのコンパイラでは，NEW 節でも REDUCTION 節でも宣言されていないスカラ変数について，自動的に NEW 変数と見なす最適化が行われています．

並列 DO ループの DO 変数（同図の i）は，特別な扱いを受けます．ループ実行中にプロセッサごとにばらばらな値となることは NEW 変数と同じですが，ループ実行終了後の値は逐次解釈と一致するように保障されます．この例では，i のループ実行後，すべてのプロセッサ上の変数 i のコピーが，逐次解釈と同じ n+1 になります[2]．

```
!HPF$ INDEPENDENT,NEW(k,tmp,j)
      do i=m,n
       k=i+1
       do j=k,n
         tmp=a(i-1,k)+a(i,k)
         b(k)=tmp
       enddo
      enddo
```

Fig. 1　NEW 変数の例．

1　厳密には，繰り返しを跨ぐデータ依存をもつ DO ループを INDEPENDENT で指示したことになり，文法違反です．
2　i を NEW 節に加えれば，この処理が省略されて，i の値はループ実行後に不定になります．

Let Us Start Parallel Processing Using High Performance Fortran ! 4. Let Us Make HPF Program (2)
IWASHITA Hidetoshi, HAYASHI Yasuharu and ISHIGURO Seiji
corresponding author's e-mail: iwashita.hideto@jp.fujitsu.com

4.2 集計変数と集計計算

4.2.1 集計計算とは

集計変数(リダクション変数)は，総和や最大値など，集計演算のためだけに使用される特別な変数です．Fig. 2 のプログラム例を見ていきましょう．

同図の DO ループは，変数 asum について，繰り返しを跨いだ値の継承が必要です．これまで説明した範囲では，並列化できないループに分類されます．しかし，このループは変数 asum に対して特別なパターンの計算—**集計計算**—を構成しているので，REDUCTION 節で asum の指定があれば，コンパイラは asum を集計変数として扱って，ループを並列化することができます．

Fig. 3 は 2 プロセッサ実行のときのイメージを示しています．DO ループは並列化され，変数 a の分散 (cyclic) に合わせて，4 行目のようなループ処理の分担が行われています．集計変数に関わる生成コードは，コンパイラによって多少異なるでしょうが，概ね太字で示したようになります．

Fig. 2 の逐次解釈では，asum に a の要素を a(1) から a(n) の順序で足し込んでいく計算になっていますが，順序を守ったままでは並列化できません．そこで Fig. 3 では，

(1) 現在の値を退避し，0 で初期化（2，3 行目）
(2) 並列ループ内で，自分の持つデータだけを合計
(3) プロセッサ間で合計（7 行目）
(4) 退避していた値を加えて終了（8 行目）

という順序に変更することで，ループを並列化しています[3]．

集計変数は，配列であっても構いません．その場合は，すべての配列要素に対して集計演算が行われます．この例は**4. 3. 3**項で紹介します．

4.2.2 集計演算の種別の指定

集計計算（集計演算を含む一連の計算）を構成する条件は，HPF/JA 仕様[1]で定義されています．大まかにはこう理解してください．集計変数に対する演算は，交換則と結合則の成り立つもの（加算，乗算，最大・最小値，論理和・積，ビット和・積など）に限られ，足し込むとか，より大きな値に置き換えるなど，同じ演算を繰り返していくパターンに限られます．

HPF/JA 仕様では，

```
REDUCTION(MAX:amax,bmax)
REDUCTION(+:asum,bsum)
```

のように，変数名の並びの前に演算子や組込み関数名を″:″で区切って書き，集計演算の種別を指定することができますが，Fig. 2では省略しています．指定が省略できる条件は，大雑把に言えば，集計変数の更新が以下の形をした文（集計文）だけで行われる場合です[4]．

r = r op expr　または　r = expr op r
r = f(r, expr)　または　r = f(expr, r)

ここで，r は集計変数，op と f はそれぞれ集計演算として許されている演算子と組込み関数（Fig. 9 参照），expr は r を含まず，かつ op よりも先に計算される式です．

種別の指定は，省略できる場合には省略する方が安全です．なぜなら，コンパイラは指定があれば無条件に従いますが，指定がなければ，集計演算のパターンに合致しない場合にエラーメッセージを表示してくれるからです．そのときはまずプログラムミスの可能性を疑ってみて，問題ないと自信があれば，種別を指定して並列化を強制すればいいでしょう．なお，INDEPENDENT 指示文を書くなら，すべての集計変数に対して REDUCTION 節による指示が必要です．並列化を自動に任せる場合には，INDEPENDENT 指示文ごと全部省略してください．

4.2.3 最大・最小とその位置

最大値または最小値を探す処理では，その位置（配列添字など）や，同じ位置の別の情報も同時に知りたいことが多くあります．HPF/JA 仕様では，そういった計算を行うループの並列化も指定することができます．

最大を取る場所が複数ある場合を考慮して，「ループを逐次実行したときに最初に見つかる最大値」には FIRSTMAX，「最後に見つかる最大値」には LASTMAX という集計種別が用意されています．同様に，最小値については，FIRSTMIN と LASTMIN が用意されています．これらの集計種別は，REDUCTION 節で指定を省略することはできません．

Fig. 4 に，FIRSTMAX の例を示します．これは，配列 a の要素中の絶対値の最大値 aamax，そのときの配列添字 i1，そして a(i1) の値 a1 を得る計算です．6 行目と 7 行目

```
1       real function asum(a,n,a0)
2       real a(n),a0
3  !HPF$ DISTRIBUTE a(CYCLIC)
4
5       asum=a0
6  !HPF$ INDEPENDENT,REDUCTION(asum)
7       do i=1,n
8        asum=asum+a(i)
9       end do
10      return
11      end
```

Fig. 2　プログラム例 2 ：集計演算．

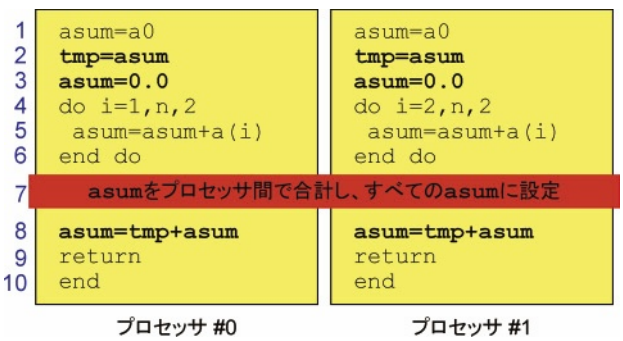

Fig. 3　プログラム例 2 の実行イメージ．

3　集計変数が浮動小数点型の場合には，計算順序の違いにより，逐次実行と並列実行で結果が異なる場合があります．
4　種別の指定は，HPF/JA[1]で可能になった機能です．種別が省略できる条件は，HPF2.0 仕様書[2]で REDUCTION 節が書ける条件として厳密に定義されています．

```
1       parameter(m=100)
2       real a(m)
3 !HPF$ DISTRIBUTE a(BLOCK)
4
5       aamax=-0.0
6 !HPF$ INDEPENDENT, REDUCTION
7 !HPF$&  (FIRSTMAX:aamax/i1,a1/)
8       do i=1,m
9         if (aamax < abs(a(i))) then
10          i1 = i
11          a1 = a(i1)
12          aamax = abs(a1)
13        end if
14      end do
15
16      write(*,*) aamax,i1,a1
```

Fig. 4　プログラム例 3 ：集計種別 FIRSTMAX.

```
1       subroutine mmul(A,B,C,m,n,l)
2       real A(m,l),B(l,n),C(m,n)
3 !HPF$ DISTRIBUTE (*,BLOCK) :: A,B,C
4
5       C=0.0
6       do j=1,n
7        do i=1,m
8         do k=1,l
9          C(i,j)=C(i,j)+A(i,k)*B(k,j)
10        end do
11       end do
12      end do
13      return
14      end
```

Fig. 5　プログラム例 4 ：行列積.

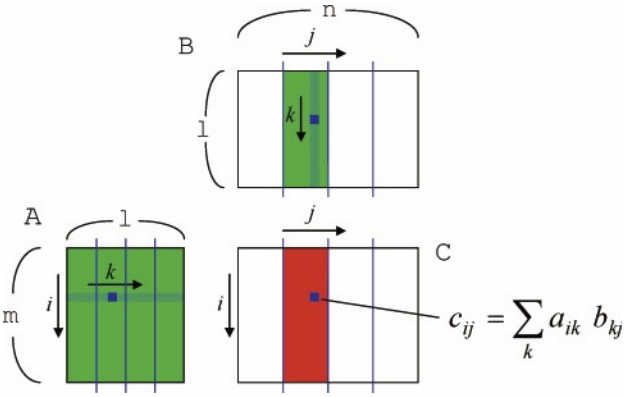

Fig. 6　例 4 のデータ分散とアクセスパターン.

に跨がる INDEPENDENT 指示文は[5]，REDUCTION 節によって，集計種別 FIRSTMAX，集計変数 aamax と位置変数 i1，a1 を指定しています．位置変数とは，aamax が最大値を得て更新されるときに同時に書き込まれるようにプログラムされた変数で，集計変数の後に 2 つの"/"で囲んで記述します．

逐次解釈でプログラムを読んでみてください．これは最初に見つかる最大値を得るプログラムなので FIRSTMAX を指定します．9 行目の IF 文の条件節の"<"を"<="に変えれば，最後に見つかる最大値を得るプログラムに変わりますが，その場合にはLASTMAXを指定します．同じように，最初に見つかる最小値なら">"と FIRSTMIN，最後に見つかる最小値なら">="と LASTMIN になります．

4. 3　多次元配列の分散と重複割付け

4. 3. 1　多次元配列のプログラム

Fig. 5 の例は，行列積を計算するサブルーチンの例です．[m,l] 行列 A，[l,n] 行列 B とサイズ l，m，n を引数として与え，結果を [m,n] 行列 C に格納しています．DISTRIBUTE指示文の構文については前回紹介しましたが，3 行目の指示文では変数 A，B，C について，1 次元目では分散せず，2 次元目でblock分散することを指示しています．ループ並列化は自動に任せていますが，このケースならまず間違いなく，ループ内の左辺の変数 C の分散に合うように，6 行目の j のループが並列化されるでしょう．

この分散と配列アクセスのパターンを Fig. 6 に示します．同図で配列を縦に切っている線は，4 並列の場合のデータ分散の様子を示し，あるプロセッサ（2 番目のプロセッサ）から見て読むだけの配列要素は緑で，読んで書く配列要素は赤で示しています．B と C については，データ分散と読み書きする配列要素がぴったり一致しているので，通信は生じません．しかし A については，全要素を読むので，他のプロセッサが持つデータを通信によって持ってくる必要があります．コンパイラはそのために必要な通信を，並列ループの前に自動で生成します．

4. 3. 2　重複配列の使用

配列 A のように読むだけのデータであれば，配列変数であっても重複割付けとすることで，通信をなくすことができます．分散の指示を書かなければ，重複割付けになると思ってください[6]．つまり，Fig. 5 の 3 行目を

```
!HPF$ DISTRIUBTE (*,BLOCK) :: B,C
```

と修正すれば，A は重複配列になり，内部で通信の起こらないサブルーチンができます．

一般に，重複配列の特徴は，主に以下の 3 点でしょう．

- 読み出しは高速です．なぜなら通信が不要だからです．
- 書き込みは分散配列より時間がかかります．1 プロセッサから書き込むと放送（broadcast）通信が発生しますし，それぞれのプロセッサで書き込んだとしても並列性が出ません．逐次実行と同じ速さが上限です．
- メモリを食います．プロセッサが増えても，プロセッサ当りのメモリ消費量が減っていきません．

つまり，読み出しを主にしてうまく利用すれば，プログラムの効率が上がることがありますが，メモリの浪費には気をつける必要があります．また，分散配列にしなければ並列化のトリガにならないので，主要な配列変数は分散することを考えてください．

5　Fortran の規則と同じで，固定形式のソースプログラムでは，6 カラム目に何か文字があれば(Fig. 4 では 7 行目の"&")，前の行に継続します．自由形式では継続される方の行の最後に"&"を書きます．

6　分散指示のない変数の分散をどうするかはコンパイラが決めますが，重複割付け以外が選択されることはまずありません．

4.3.3 重複配列への代入

同じ行列積のプログラムですが，Fig. 7に示す例は，書き込み先のCを重複としています．AとBは，それぞれ2次元目と1次元目で分散しています．ループ交換により並列化されるkのループを一番外側に移動し，通信の発生回数を削減しています．

このプログラムの並列化の特徴は，以下の2点です．

- ON指示文は，代入文の左辺でなく右辺の配列Aの2次元目の分散に合わせた処理の分担を指示しています．アクセスパターンはFig. 8に示すようになります．
- 配列変数Cを集計変数とする集計演算が行われます．これは，**4.2.1**項で説明した集計演算の配列版です．

4.3.4 分散方法の比較

同じ行列積のプログラムに対して，(1)全ての配列を2次元目でblock分散する方法，(2)配列Aを重複配列とする方法，(3)配列Cを重複配列とする方法，の3種類の分散方法を解説しました．これらの実行性能を比較すると，通信の全くない(2)が最も良く，次いで(1), (3)の順となるでしょう．(1)では，ループの実行前に，分散配列Aの全要素について，全プロセッサへの放送が起こります．(3)では，ループの実行後に，重複配列Cの全要素について，各プロセッサがもつ部分和を互いに足し合わせながら配り合う集計演算が実行されます．

```
1       subroutine mmul2(A,B,C,m,n,l)
2       real A(m,l),B(l,n),C(m,n)
3 !HPF$ DISTRIBUTE A(*,BLOCK)
4 !HPF$ DISTRIBUTE B(BLOCK,*)
5
6       C=0.0
7 !HPF$ INDEPENDENT,NEW(j,i),REDUCTION(C)
8       do k=1,l
9 !HPF$  ON HOME(A(:,k))
10       do j=1,n
11        do i=1,m
12         C(i,j)=C(i,j)+A(i,k)*B(k,j)
13        end do
14       end do
15      end do
16      return
17      end
```

Fig. 7 プログラム例5：行列積その2．

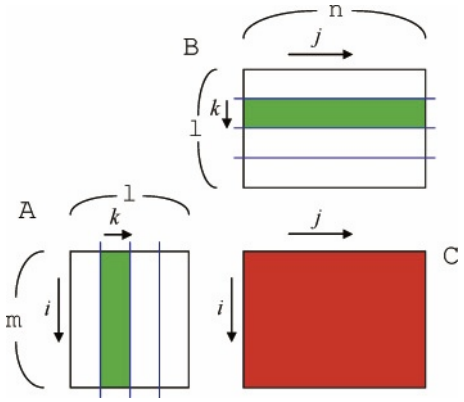

Fig. 8 例5のデータ分散とアクセスパターン．

集計(リダクション)計算を伴う並列ループ
!HPF$ INDEPENDENT [, 〈節〉] ...
〈節〉は、new節またはreduction節
reduction節は
REDUCTION([〈種別1〉:] r_1, r_2, ...)
REDUCTION(〈種別2〉: $r_1/i_{11}, i_{12}, .../, ...$)
〈種別1〉は + * .AND. .OR. .EQV. .NEQV. MAX MIN IAND IOR または IEOR
〈種別2〉は FIRSTMAX FIRSTMIN LASTMAX または LASTMIN
r_1 や r_2 は集計変数(変数名)
i_{11} や i_{12} は位置変数(スカラ変数名)

Fig. 9 今回解説した構文のまとめ．

一方で，メモリの使用量を考えてみましょう．変数の割付けに必要な領域は，3変数とも分散している(1)が最小です．しかし実行時には，分散配列Aの放送を受けるために，Aの全配列の大きさに相当する作業領域が必要なので，動的には大きな領域を確保します（少なくとも並列ループ実行中は）．(3)も動的な領域を使用しますが，大きさはコンパイラによります．(2)のような通信の起こらないプログラムでは，大きな動的な領域確保は起こりません．

このように，同一のプログラムでも分散方法によって，性能やメモリ量は大きく変わることがあります．

なお，この節では局所的な性能やメモリ効率を考えましたが，変数A，B，Cはサブルーチン引数なので，呼び出し側の実引数との関係も考える必要があります．手続を跨いだデータ分散や性能については，次回考えていくことにしましょう．

4.4 演習

- 姫野ベンチHPF版[3]でREDUCTION節の使い方を確認してください．結果出力から，集計演算の結果が並列数を変えると変化することも確認できるでしょう．
- 行列積でj方向とk方向の並列化を紹介しましたが，実はi方向でも並列化できます．書いてみてください．

4.5 まとめ

今回までで，HPFの基本的な書き方を一通り網羅しました．今回登場した集計計算の構文をFig. 9にまとめます．次回は，手続単位を跨いだプログラム全体の考え方を紹介します．

参 考 文 献

［1］HPF/JA言語仕様書 Version 1.0
(http://www.hpfpc.org/jahpf/spec/jahpf-j.html)
［2］High Performance Fortran言語仕様書 Version 2.0
(http://www.hpfpc.org/jahpf/spec/hpf-j.html)
［3］HPF推進協議会（http://www.hpfpc.org/）