J. Plasma Fusion Res. Vol.83, No.2 (2007) 188-192

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

7．もっと活用しよう HPF(1)

林　康晴
NEC　第一コンピュータソフトウェア事業部

（原稿受付：2006年1月18日）

今回は，既存のFortranプログラムをHPF化する際の注意事項や，今まで触れられなかったHPFのデータマッピングの機能，および高速化手法の第一弾として，ループの高速化について説明します．

7.1　コードのクリーンナップ

HPFはFortranの拡張仕様なので，基本的にはFortranプログラムはそのままHPFプログラムとして実行できますが，配列が複数のプロセッサ間に分割配置される場合があるために，いくつか修正が必要な記述があります．これまでの連載でも折に触れて説明してきましたが，ここでまとめておくことにします．既存のFortranプログラムをHPF化する場合は，まずここに挙げた項目について修正した後，HPF指示文を挿入するとよいでしょう．なお，並列化する必要のない手続は，以前の連載で説明したEXTRINSIC機能を利用して，Fortranプログラムとして翻訳・リンクすれば下記のような修正を行わずに済みます．

1．アドレス渡しは部分配列に修正し，実引数として配列を渡すことを明示します（Fig. 1(a)）．
2．大きさ引継ぎ配列（擬寸法配列）は整合配列または形状引継ぎ配列に修正します（Fig. 1(b)）．
3．実引数と仮引数の次元数と各次元の寸法を一致させます．実引数として部分配列，仮引数として整合配列や形状引継ぎ配列を利用するとよいでしょう．

```
real a(n,n)                 real a(n,n)
call sub(a(1,i),n)          call sub(a(:,i),n)
    << 略 >>          →         << 略 >>
subroutine sub(a,n)         subroutine sub(a,n)
real a(n)                   real a(n)
```

(a)アドレス渡しを部分配列に修正

```
real a(100,20)              real a(100,20)
call sub(a)                 call sub(a,20)
    << 略 >>          →         << 略 >>
subroutine sub(a)           subroutine sub(a,n)
real a(100,*)               real a(100,n)
```

(b)大きさ引継ぎ配列を整合配列に修正

Fig. 1　アドレス渡し，大きさ引継ぎ配列の修正方法．

4．共通ブロックの宣言は，分散の指定も含めて，全ての出現で一致させます．INCLUDEファイルやモジュール中に宣言を記述すると書き忘れ・書き誤りが防げます．
5．分散配列に対するEQUIVALENCE文は削除します．大きな配列を宣言し，それを部分的に使用しているような場合，割付け配列など実行時に宣言範囲を決められる機能を使って，配列は実際に利用する形状と型で宣言するようにします．

7.2　配列の宣言とデータマッピング

本節では，配列の大きさを実行時に決めたい場合や，宣言範囲または参照パターンの異なる配列がループ中に混在する場合のデータマッピングの記述法を解説します．

7.2.1　配列の大きさを実行時に決めたい場合

プログラムの実行時に配列の大きさを決めたい場合，Fortran90仕様の自動割付け配列や割付け配列を利用するとよいでしょう．自動割付け配列は，Fig. 2の配列aのように，仮引数や共通ブロック変数を使って各次元の宣言範囲を指定できる局所変数です．また割付け配列は，Fig. 2の配列b1，b2のように，"allocatable"を付けて宣言した上で，実行時にALLOCATE文によって各次元の上下限を指定し，必要な領域を割り付ける配列です．

割付け配列をblock分散する際注意が必要なのは，宣言範囲が翻訳時に決まらないため，各配列要素がどのプロセッサに配置されるかは，一般に実行時までわからない，ということです．例えばFig. 2のdoループは，配列b1，b2の宣言範囲が同一なら通信なしで実行できますが，宣言範囲がb1(1:200)，b2(1:160)だとすると，Fig. 3(a)のように，例えばb1(100)はp(1)上に，b2(100)はp(2)上にそれぞれ配置されるため，代入文b1(100)=b2(100)の実行に通信が必要となります．たとえ実際には宣言範囲が同じであっても，翻訳時には不明の場合，DISTRIBUTE指示文だけで分割配置を指定すると，コンパイラは，通信が必要な場合も動作するよう効率の悪い実行コードを生成する可能性があります．このような場合に有効なのが**ALIGN指示文**[1]です．ALIGN指示文を使うと，複数の配

Let Us Start Parallel Processing Using High Performance Fortran ! 7. Let Us Practice HPF (1)

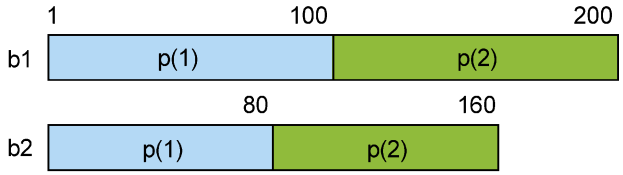
HAYASHI Yasuharu
author's e-mail: hayashi@hpc.bs1.fc.nec.co.jp

```
      subroutine ex1(n)
      common /com/m
      real a(n,m)                    ! 自動割付け配列
      real,allocatable :: b1(:),b2(:) ! 割付け配列
!HPF$ PROCESSORS p(2)
!HPF$ DISTRIBUTE (BLOCK) ONTO p :: b1,b2
      allocate(b1(n1),b2(n2))
      do i=1,100
        b1(i) = b2(i)
      enddo
```

Fig. 2　自動割付け配列と割付け配列.

(a) DISTRIBUTE指示文のみ

```
!HPF$ DISTRIBUTE (BLOCK) ONTO p :: b1,b2
```



(b) ALIGN指示文+DISTRIBUTE指示文

```
!HPF$ ALIGN b2(i) WITH b1(i)
!HPF$ DISTRIBUTE (BLOCK) ONTO p :: b1
```

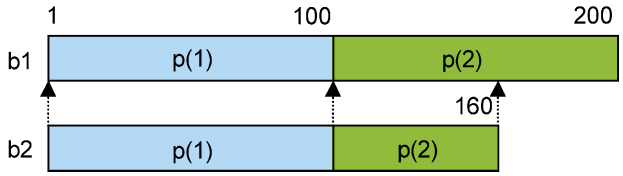


Fig. 3　ALIGN 指示文の効果.

列間の相対的位置関係（**整列**）を指定できます．例えば，Fig. 3(b)の ALIGN 指示文は，配列 b1 を基準として，b2(i) を b1(i) と同一のプロセッサに配置する，ということを意味しています．この基準配列を**整列先**と呼びます．整列先 b1 を DISTRIBUTE 指示文で分散すれば，b2 の分割配置は b1 との相対的な位置関係により自動的に決まることになります．このように，ALIGN 指示文を利用すると，実行時に宣言範囲が決まる場合でも，配列同士の添字の対応により通信が不要であることが翻訳時にわかるので，効率の良い実行コードが生成されます．同様に，形状引継ぎ配列や見かけ上宣言範囲の異なる自動割付け配列（Fig. 4）に対しても，ALIGN 指示文が有効です．

7. 2. 2　配列の宣言範囲がさまざまな場合

Fig. 5(a)のように，配列によって宣言範囲が異なる場合も ALIGN 指示文が有効です．Fig. 6(a)のように，宣言範囲が異なる配列を block 分散で均等に分割すると，各プロセッサに配置される範囲も少しずれてしまうため，ループの実行時に通信が必要となります．そこで Fig. 6(b)のように ALIGN 指示文を指定すれば，対応する配列要素を同一のプロセッサに配置でき，通信の発生を避けられます．

ここで，ALIGN 指示文の整列先 c3 の宣言範囲は，整列を指定する配列 c1，c2 の対応次元の宣言範囲を包含している必要があることに注意してください．整列先に対して“はみ出す”要素があると，その要素はプロセッサとの対応

```
      real a(100)
!HPF$ DISTRIBUTE a(BLOCK)
      call ex2(a,100,100)
      end
      subroutine ex2(a,n,m)
      real a(:)       ! 形状引継ぎ配列
      real b(n),c(m)  ! 自動割付け配列
!HPF$ ALIGN (i) WITH a(i) :: b,c
!HPF$ DISTRIBUTE a(BLOCK)
      do i =1,n
        a(i) = b(i) + c(i)
      enddo
```

Fig. 4　形状引継ぎ配列と自動割付け配列の整列.

```
      real c1(0:10), c2(11), c3(0:11)
!HPF$ PROCESSORS p(2)
!HPF$ DISTRIBUTE (BLOCK) onto p :: c3
!HPF$ ALIGN (i) WITH c3(i) :: c1,c2
      do i =1,10
        c1(i) = c2(i) + c3(i)
      enddo
```

(a)変数 c3 に対する整列

```
      real d1(0:10), d2(11)
!HPF$ TEMPLATE t(0:11)
!HPF$ ALIGN (i) WITH t(i) :: d1,d2
!HPF$ PROCESSORS p(2)
!HPF$ DISTRIBUTE t(BLOCK) onto p
```

(b)テンプレート t に対する整列

Fig. 5　宣言範囲が異なる場合.

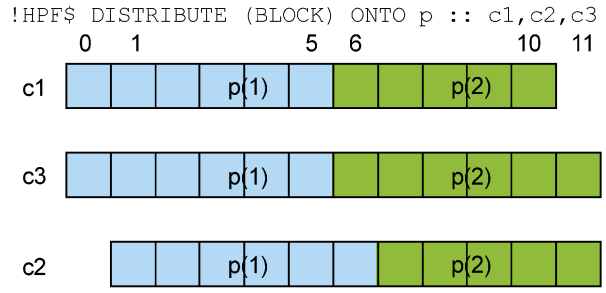
が取れないので，翻訳時または実行時エラーとなるからです．では，Fig. 5(b)のように，他の配列の宣言範囲を包含する配列が存在しない場合はどうしたらよいでしょう？　配列 d1 と d2 は，どちらを整列先として ALIGN 指示文を指定しても“はみ出す”要素が発生してしまいます．Fig. 5(a)の配列 c3 のような，全ての配列の宣言範囲を包含する配列を別に宣言すればよいのですが，マッピングの指定のためだけにメモリを浪費したくはないですよね．

このような場合に有効なのが **TEMPLATE 指示文**[1]です．TEMPLATE 指示文を使うと，メモリを消費しない仮想配列（**テンプレート**）を宣言できます．Fig. 5(b)のように，全ての配列の分散次元の宣言範囲を包含するようなテンプレート t を宣言し，それを整列先として各配列に ALIGN 指示文を指定した上で，テンプレート t を分散すれば，Fig. 6(b)のように，各配列の対応する要素を同じプロセッサに配置できます．

7. 2. 3　ループ中の配列添字に不一致がある場合

Fig. 7(a)のループは，配列 e1，e2 の分散次元である 2 次元目の添字が 1 つずれているため，block 分散だけではやはり通信が発生しますが，Fig. 7(b)のように，ALIGN 指示文で e1(:,i) と e2(:,i+1) を対応させれば通信なしで実行が可能になります．この例でも，e1(:,i) と e2(:,i+1) を直接整列させると“はみ出す”要素が発生してしまうので，両配列を包含する範囲でテンプレートを宣言し，ALIGN 指示文の整列先としています（Fig. 8 参照）．なお，ALIGN 指示文中で，各配列の 1 次元目に“*”が指定されて

(a) DISTRIBUTE指示文のみ



(b) ALIGN指示文+DISTRIBUTE指示文

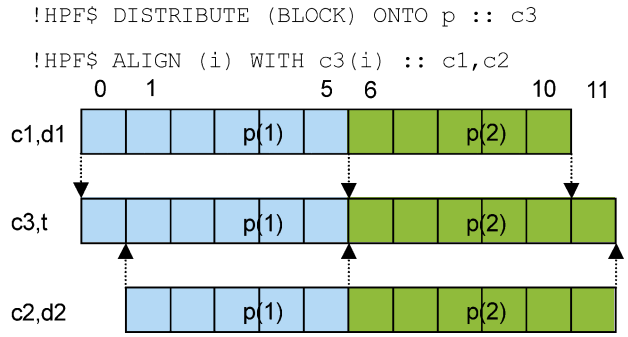


Fig. 6 対応する要素を ALIGN 指示文で整列する.

```
      real e1(100,10), e2(100,10)
!HPF$ DISTRIBUTE (*,BLOCK) :: e1,e2
      do i=1,9
        do j=1,100
          e1(j,i) = -e2(j,i+1)
        enddo
      enddo
```

(a) DISTRIBUTE 指示文のみの指定

```
      real e1(100,10), e2(100,10)
!HPF$ TEMPLATE t(11)
!HPF$ DISTRIBUTE (BLOCK) :: t
!HPF$ ALIGN (*,i) WITH t(i+1) :: e1
!HPF$ ALIGN (*,i) WITH t(i) :: e2
```

(b) ALIGN 指示文+DISTRIBUTE 指示文(テンプレート利用)

Fig. 7 添字に不一致がある場合.

いることに注意してください．この例のように整列先と次元数の異なる配列を整列する場合，分散しない次元には通常"*"を指定します．

7.2.4 HPF のデータマッピングまとめ

HPF では，配列のマッピングは，DISTRIBUTE 指示文と ALIGN 指示文の組合せで指定することができます．一般には，整列先とする配列またはテンプレートに対する相対的な位置関係を ALIGN 指示文で指定した上で，その整列先に対して DISTRIBUTE 指示文を指定すれば，全ての配列のデータマッピングを決定できます．ALIGN 指示文と TEMPLATE 指示文の構文は，末尾にまとめておきましたのでご参照ください．

7.3 不均等なデータマッピングの使い方

これまで，最も汎用的な block 分散を主に解説してきましたが，本節では不均等な分割配置を行うと効率の良い例をご紹介します．

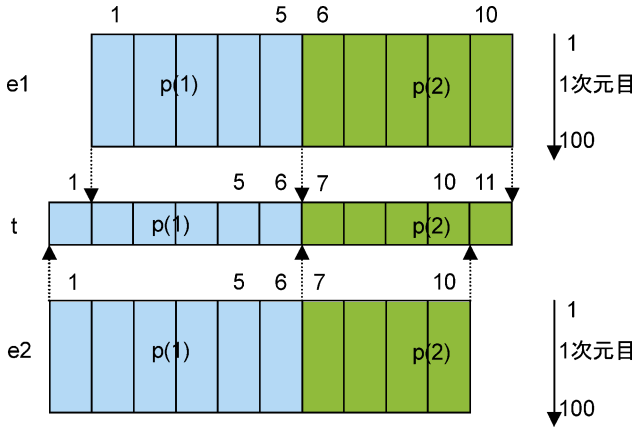


Fig. 8 2 次元配列の 1 次元配列への整列.

```
      dimension a(14),b(14)
      integer,parameter :: map=(/4,(3,i=1,2),4/)
!HPF$ PROCESSORS p(4)
!HPF$ DISTRIBUTE (GEN_BLOCK(map)) onto p :: a,b
      do i=2,13
        a(i) = b(i-1) + b(i) + b(i+1)
      enddo
```

Fig. 9 block 分散では負荷バランスの悪い例.

7.3.1 block 分散では負荷バランスが悪い場合

block 分散の場合，1 つのプロセッサに配置される要素数は，配列の分散次元の寸法を N，プロセッサ数を P とすると，(N−1)/P+1 で計算されます．ここで，例えば寸法が14の次元を 4 プロセッサ上に block 分散すると，上記の式によりプロセッサあたりの要素数は 4 となるため，最後のプロセッサだけ 2 要素しか配置されないことになります．そのため，Fig. 9 の配列 a，b を block 分散で 4 プロセッサ上に分割配置し，do ループの各繰返しを，a(i)が配置されているプロセッサが担当するよう並列化した場合，Fig. 10(a)のように，p(1)は 3 回，p(2)，P(3)は 4 回の繰返しを実行するのに対して，p(4)は 1 回の繰返ししか実行しないことになり，負荷のアンバランスにより効率が低下します．このような場合，Fig. 9 のように DISTRIBUTE 指示文中で gen_block 分散を指定します．gen_block 分散では，map(i)の値によって，プロセッサ p(i)に配置する配列要素の数を指定しています(この配列 map を**マッピング配列**といいます)．gen_block 分散を利用すると，Fig. 10(b)のように，全てのプロセッサが 3 回の繰返しを実行することになり，負荷を均等化することができます．

gen_block 分散においては，各プロセッサに配置する要素数やプロセッサ数を実行時に決めたい場合がよくあります．そのような時には，Fig. 11 のように，呼出し側のプログラムでマッピング配列を設定し，それを引数として受け取るとよいでしょう．

7.3.2 疎行列の行列ベクトル積

Fig. 12 のような，疎行列の零成分を除いて一次元配列 s に圧縮し，非零成分の行番号，列番号をそれぞれ配列 rst，

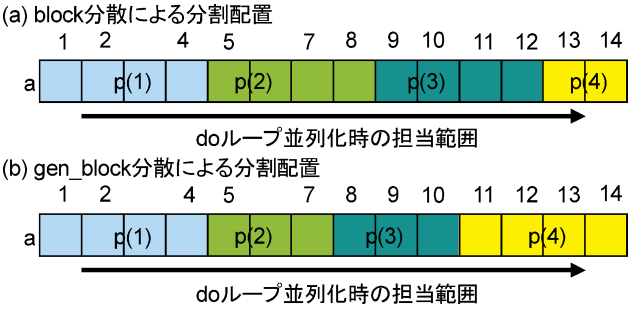


Fig. 10 gen_block 分散による負荷均等化の効果.

```
      program dummy_main
      integer, allocatable :: map(:)
      allocate(map(number_of_processors()))
      read(10)map
      call real_main(map,number_of_processors())
      end
      subroutine real_main(map,np)
      integer map(np)
      real a(100)
!HPF$ DISTRIBUTE (GEN_BLOCK(map)) :: map
```

Fig. 11 呼出し側でのマッピング配列の設定.

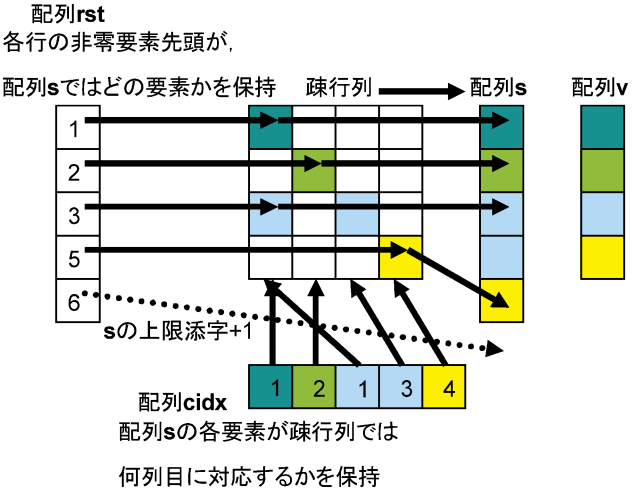


Fig. 12 疎行列の CRS 形式.

cidx に保持しておくようなデータ構造において，Fig. 13 のように，行列ベクトル積を行うプログラムを考えます．結果を格納する配列 v を 4 プロセッサ上に block 分散する場合，Fig. 12 で色分けしたように配列 s，cidx を分散すると通信なしで i のループを並列化できます（配列 rst，x は複数の i の繰返しで参照されるため重複配列とします）．

このような不均等な分散は，Fig. 13 のように，gen_block 分散により指定できます．ただし，gen_block 分散された配列と block 分散された配列が 1 つのループ中で混在する場合，コンパイラが通信の有無を翻訳時に判定することは一般的には困難です．そのため，効率良く並列実行を行うためには，配列 s，cidx に対する通信の必要がないことを，前回説明した ON-HOME-LOCAL 指示文で明示する必要があることに注意してください．

```
      real s(5),v(4),x(4)
      integer rst(5),cidx(5)
      integer, parameter :: m(4)=(/1,1,2,1/)
!HPF$ PROCESSORS p(4)
!HPF$ DISTRIBUTE (BLOCK) ONTO p :: v
!HPF$ DISTRIBUTE (GEN_BLOCK(m)) ONTO p :: s,cidx
!HPF$ INDEPENDENT
      do i = 1,4
!HPF$   ON HOME(v(i)), LOCAL(s,cidx) BEGIN
        v(i) = 0. 0
        do j = rst(i),rst(i+1)-1
          v(i) = v(i) + s(j) * x(cidx(j))
        enddo
!HPF$   ENDON
      enddo
```

Fig. 13 疎行列の行列ベクトル積.

7.4 HPF プログラムの高速化(1)

本節では，HPF プログラムの高速化手法その 1 として，ループ実行時の通信を削減することによって，性能向上を図る手法をご紹介します．

7.4.1 境界処理のループ

境界処理においては，しばしば両端の特定のプロセッサ上のデータのみで計算が行われます．このような場合，ON-HOME-LOCAL 指示文を指定して通信が不要であることを明示すれば，効率よく実行できることが多々あります．例えば，Fig. 14(a)のようなループを考えてみます．100 プロセッサで実行するような場合を考えると，一般には a(i,1) と a(i,2)，あるいは a(i,n-1) と a(i,n) が必ず同一のプロセッサに配置されているとは限りませんので，HPF コンパイラは通信に備えたコードを生成します．しかし，実際にはほとんどの場合，これら両端の要素は同一のプロセッサ上に配置されています．そこで，例えば10プロセッサで実行する場合，Fig. 14(b)のように，通信が不要であることを ON-HOME-LOCAL 指示文で明示すると，通信に備えた効率の悪い実行コードを避けられます．

7.4.2 ループ中の配列参照パターンに不一致がある場合

ループ中で参照される配列の分散や添字に不一致がある

```
      parameter(n=100)
      real a(n,n)
!HPF$ DISTRIBUTE a(*,BLOCK)
      do i = 1, n
        a(i,1) = a(i,2)
        a(i,n-1) = a(i,n)
      enddo
```

(a)

```
      do i = 1, n
!HPF$   ON HOME(a(:,1)), LOCAL BEGIN
        a(i,1) = a(i,2)
!HPF$   ENDON
!HPF$   ON HOME(a(:,n)), LOCAL BEGIN
        a(i,n-1) = a(i,n)
!HPF$   ENDON
      enddo
```

(b)

Fig. 14 境界処理のループ.

```
      real,dimension(10,10) :: u,v,p
!HPF$ DISTRIBUTE (*,BLOCK) :: u,v,p
!HPF$ INDEPENDENT
      do j=1,9
        do i=1,9
          u(i+1,j) = -p(i+1,j+1)
          v(i,j+1) = p(i+1,j+1)
        enddo
      enddo
```

(a)

```
!HPF$ DISTRIBUTE (*,BLOCK) :: u,v,p
!HPF$ SHADOW p(0,0:1)
!HPF$ REFLECT p
!HPF$ INDEPENDENT
      do j=1,9
!HPF$   ON HOME(u(:,j)), LOCAL BEGIN
        do i=1,9
          u(i+1,j) = -p(i+1,j+1)
        enddo
!HPF$   ENDON
      enddo
      do j=1,9
        do i=1,9
          v(i,j+1) = p(i+1,j+1)
        enddo
      enddo
```

(b)

Fig. 15 添字に不一致がある場合のループ分割.

ために，1つの繰返し中で異なるプロセッサに配置されている配列要素を参照してしまい，通信が発生する時には，ループ分割によって効率の良い実行が可能になる場合があります．

Fig. 15(a)のループを見てみましょう．配列 u の分散次元の添字が j であるのに対して，v，p は j+1 であるため，隣接プロセッサ間での通信が必要です．添字のずれが読出し側（右辺）だけで出現しているのなら，前回解説したような SHADOW 指示文，REFLECT 指示文，および ON-HOME-LOCAL 指示文の組合せによる袖領域への一括通信が可能ですが，本例のように書込み側（左辺）の場合，そのまま適用することはできません．また，コンパイラが自動的に通信を行うと，配列 u 全体が一時配列に置き換えられてしまい，効率が低下する可能性があります．そこで，

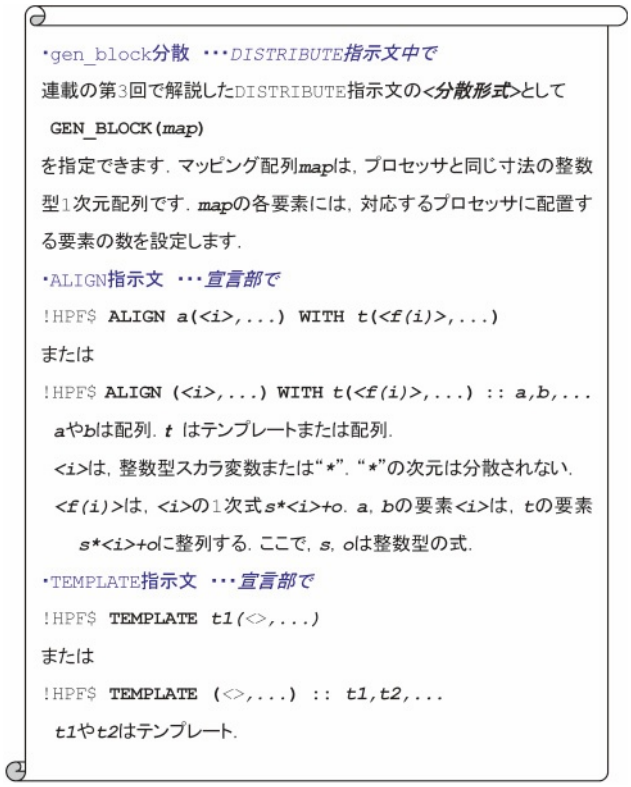


Fig. 16 今回紹介した構文のまとめ.

Fig. 15(b)のようにループを 2 つに分割し，各ループの左辺に出現する配列の分散次元の添字を揃えてやると，袖領域への一括転送を利用した効率の良い並列化が可能になります．実はこの例の場合，Fig. 7(b)のような方法で，u(:,j) と v(:,j+1)，p(:,j+1) とを ALIGN 指示文により対応させれば，ループを分割せずに通信なしで実行することができます．どちらの方法がよいかは，プログラムの他の部分での参照パターンを考慮して，全体として効率の良くなる方を選択するとよいでしょう[2]．

参 考 文 献

[1] High Performance Fortran 言語仕様書 Version 2.0 (http://www.hpfpc.org/japhpf/spec/hpf-j.html).

[2] 森井宏幸，坂上仁志，新居学，高橋豊：HPF の性能評価と応用に関する研究，情報処理学会研究報告，2003-HPC-95, 143-148 (2003).