# 第III部

# HPF公認拡張仕様

ここでは、High Performance Fortran 公認拡張の機能の構文と意味について記述する。ほとんどの場合、これらの機能は HPF の概念に基づいて設計されている。そのため、背景となる情報を知るためには第 I 部や第 II 部を参照しなければならない。

# 第8章　データマッピングの公認拡張仕様

本章では、第 3 章で述べたデータマッピングに関する基本機能の能力を拡張するいくつかの機能について述べる。これらの拡張は、二つのカテゴリーに分けられる。

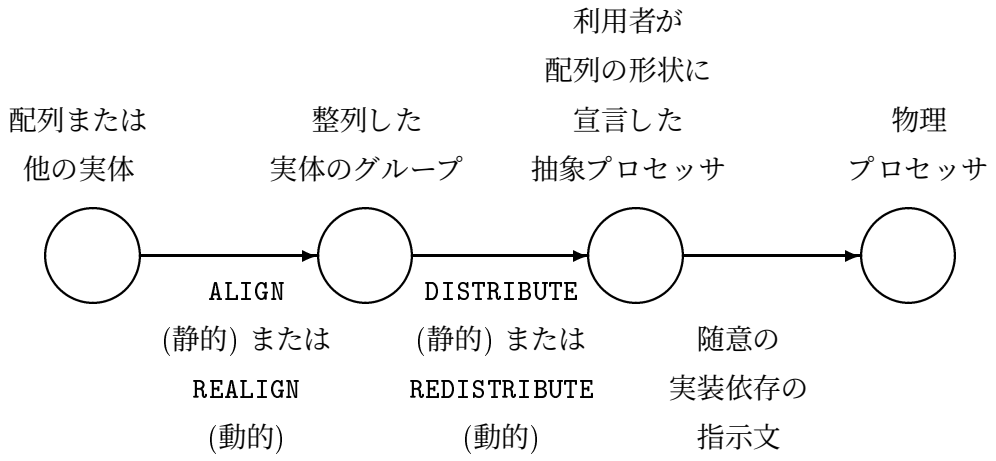
拡張の第一のカテゴリーでは、利用者がデータのマッピングに関してさらに多くの制御を行なうための機能を提供する。それらには、DYNAMIC と宣言したデータを実行時に再分散や再整列することができる動的再マッピングのための指示文が含まれる。DISTRIBUTE 指示文の ONTO 節は、プロセッサ集合の部分への分散を可能にするように拡張される。さらに、ポインタや構造型の成分の明示的マッピングも導入される。二つの新しい分散も含まれる: ブロック分散を一般化した GEN_BLOCK 分散、それに、個々の配列要素のマッピングをマッピング配列を使って指定することを可能にする INDIRECT 分散である。

拡張の第二のカテゴリーは、利用者が効率的なプログラムを作成するために、有益な情報をコンパイラに与えることに利用することができる。RANGE 指示文を使うと、動的に分散された配列、ポインタまたは仮引数の分散のレンジを指定することができる。SHADOW 指示文は、隣接計算 (nearest-neighbor computation) において、ローカルでない要素を収めるための追加スペースの量を指定するために利用することができる。

本章は、拡張について述べるため、第 3 章および第 4 章の一部を繰り返し述べる。その際、必要に応じて新しい構文規則を示し、古い構文規則を拡張する。特に、8.13、8.14 および 8.15 節は、ここで述べる公認拡張に基づいて第 3 章の対応する節を拡張する。

## 8.1　拡張されたモデル

抽象プロセッサへのデータの割当ての基本モデルは、第 3 章で述べたような二段階マッピングのままである。しかし、以下に図示されているように、データ実体の動的再マッピングを可能にするように拡張されている:



したがって、実行指示文 REALIGN および REDISTRIBUTE を使って実体を実行時に再マップすることができる。整列木の根である実体 (すなわち、他の実体に明示的に整列していない実体) は、明示的に再分散することができる。そのような実体を再分散することは、整列関係を保つために、その実体に最終的に整列しているすべての実体の再分散を引き起こす。

整列木の根でない実体は、明示的に再整列することができるが、明示的に再分散することはできない。このような再整列は、他の実体のマッピングを変更することはない。データの再マッピングは、プロセッサ間の通信を必要とすることに注意されたい。

HPF は、Fortran の ALLOCATABLE 属性への類推によって、DYNAMIC 属性を採り入れた。DYNAMIC と宣言されていない配列は、REALIGN することが許されない。同様に、DYNAMIC と宣言されていない配列やテンプレートは、REDISTRIBUTE することが許されない。

SAVE 属性付きの局所変数、共通ブロックの変数および参照結合により参照される変数は、暗黙的に再マップ (たとえば、可変的な分散形式をもつことによって、または可変的な分散形式をもつ言語要素に整列することによって起きる) されてはならない。これらの三つの種類の変数の中では、参照結合によりアクセスされる変数だけが DYNAMIC 属性をもつことができる。

3.1 節に示すように、実体は、その実体が宣言されている有効域にある HPF のマッピング指示文に現れた場合に、**明示的にマップされている**とみなされ、さもなければ**暗黙的にマップされている**とみなされる。3.1 節のマッピング指示文の定義は、次のように拡張される: マッピング指示文とは、ALIGN、DISTRIBUTE、INHERIT、DYNAMIC、RANGE、SHADOW 指示文、それに、整列や分散または INHERIT、DYNAMIC、RANGE、SHADOW 属性を与える指示文である。

## 8.2 拡張データマッピング指示文の属性形式の構文

他のマッピング指示文と同様に、実行指示文である REALIGN や REDISTRIBUTE にも二つの形式 (文形式と属性形式) がある。しかし、一つの指示文の中で他の属性と組み合わせることはできない。RANGE と SHADOW 属性は、一つの指示文の中で他の属性と組み合わせることができる。

H801 *combined-attribute-extended* **is** ALIGN *align-attribute-stuff*
**or** DISTRIBUTE *dist-attribute-stuff*
**or** INHERIT
**or** TEMPLATE
**or** PROCESSORS
**or** DIMENSION ( *explicit-shape-spec-list* )
**or** DYNAMIC
**or** RANGE *range-attr-stuff*
**or** SHADOW *shadow-attr-stuff*
**or** SUBSET

制約: SUBSET 属性は、プロセッサ構成にだけ適用できる。

SUBSET 属性については、第 9 章で論じる。他は下記で論じる。

## 8.3 REDISTRIBUTE 指示文

REDISTRIBUTE 指示文は、DISTRIBUTE 指示文と似ている。ただし、REDISTRIBUTE 指示文は実行指示文である。実体やテンプレートは、DYNAMIC と宣言されていればいつでも再分散することができる (8.5 節 参照)。配列 (またはテンプレート) に対し最終的に整列している他の実体は、その最終整列先が再分散されたときに、新しい分散を反映するために再マップされ、整列関係を維持する (3.4 節 参照)。(これらの操作は、実行時に多くの計算と通信の労力を必要とするため、プログラマは、この機能を使う場合には注意する必要がある。)

DISTRIBUTE 指示文は、有効域の宣言部にだけ現れることができる。REDISTRIBUTE 指示文は、有効域の実行部にだけ現れることができる。DISTRIBUTE と REDISTRIBUTE の主要な違いは、DISTRIBUTE は、分散形式 (BLOCK や CYCLIC など) の引数として、宣言式 (*specification-expr*) だけを含むことができ、一方 REDISTRIBUTE では、これらの引数はどんな整数式でも構わないことである。もう一つの違いは、DISTRIBUTE は属性であり、*combined-directive* の一部として他の属性と組み合わせることができるが、REDISTRIBUTE は属性でない。(ただし、REDISTRIBUTE 文は、"::" 区切りを使った属性形式で記述することができる。)

REDISTRIBUTE 指示文の構文は次のとおりである。

H802 *redistribute-directive* **is** REDISTRIBUTE *distributee dist-directive-stuff*
**or** REDISTRIBUTE *dist-attribute-stuff* :: *distributee-list*

制約: REDISTRIBUTE 指示文に現れる *distributee* は、DYNAMIC 属性をもたなければならない (8.5 節 参照)。

制約: REDISTRIBUTE 指示文の *distributee* は、ALIGN または REALIGN 指示文の *alignee* として現れてはならない。

制約: REDISTRIBUTE 指示文の *dist-format-clause* や *dist-target* のどちらも、"*" で始まってはならない。

実体は、INHERIT 属性と DISTRIBUTE 属性の両方をもってはならないものの、INHERIT 属性をもっているいないにかかわらず、DYNAMIC 属性をもち、ALIGN や REALIGN 指示文の *alignee* に現れない限り、REDISTRIBUTE 指示文の *distributee* に現れることができることに注意されたい。

RANGE 指示文 (8.11 節 参照) が、*distributee* で許される分散形式を限定するために使われた場合、新しいマッピングは、RANGE 指示文で指定された分散形式の一つにマッチしなければならない。

REDISTRIBUTE 指示文の文形式は、一つだけの *distributee* を指定しようとした属性形式の短縮形と考えることができる。

```
!HPF$ REDISTRIBUTE distributee ( dist-format-list ) ONTO dist-target
```

は、以下と同じである。

```
!HPF$ REDISTRIBUTE ( dist-format-list ) ONTO dist-target :: distributee
```

## 8.4 REALIGN 指示文

REALIGN 指示文は、ALIGN 指示文と似ている。ただし、REALIGN 指示文は実行指示文である。配列 (またはテンプレート) は、DYNAMIC と宣言されていればいつでも再整列することができる (8.5 節 参照)。再分散 (8.5 節 参照) と違い、データ実体の再整列は、他の実体の再マッピングを引き起こさない。(しかし、たった一つの実体の再整列でも、それが大きければ実行時に多くの計算と通信の労力を必要とするため、プログラマはこの機能を使う場合には注意する必要がある。)

ALIGN 指示文は、有効域の宣言部にだけ現れることができる。REALIGN 指示文は、有効域の実行部にだけ現れることができる。ALIGN と REALIGN の主要な違いは、ALIGN は、添字 (*subscript*) や添字三つ組 (*subscript-triplet*) の中に宣言式 (*specification-expr*) だけを含むことができ、一方 REALIGN では、添字 (*subscript*) や添字三つ組 (*subscript-triplet*) の中の式は、宣言式 (*specification-expr*) である必要はないことである。もう一つの違いは、ALIGN は属性であり、*combined-directive* の一部として他の属性と組み合わせることができるが、REALIGN は属性でない。(ただし、REALIGN 文は、"::" 区切りを使った属性形式で記述することができる。)

REALIGN 指示文の構文は次のとおりである。

H803 *realign-directive* **is** REALIGN *alignee align-directive-stuff*
**or** REALIGN *align-attribute-stuff* :: *alignee-list*

制約: REALIGN 指示文に現れる *alignee* は、DYNAMIC 属性をもたなければならない (8.5 節 参照)。

制約: *align-with-clause* に指定された *align-target* が DYNAMIC 属性をもつ場合、*alignee* も同様に DYNAMIC 属性をもたなければならない。

制約: REALIGN 指示文の *alignee* は、DISTRIBUTE や REDISTRIBUTE 指示文の *distributee* であってはならない。

実体は、INHERIT 属性と ALIGN 属性の両方をもってはならないものの、INHERIT 属性をもっているいないにかかわらず、DYNAMIC 属性をもち、DISTRIBUTE や REDISTRIBUTE 指示文の *distributee* に現れない限り、REALIGN 指示文の *alignee* に現れることができることに注意されたい。

RANGE 指示文 (8.11 節 参照) が、*alignee* で許される分散形式を限定するために使われた場合、新しいマッピングは、RANGE 指示文で指定された分散形式の一つにマッチしなければならない。

## 8.5 DYNAMIC 指示文

DYNAMIC 属性は、実体を動的に再整列や再分散してもよいことを指定する。

H804 *dynamic-directive* **is** DYNAMIC *alignee-or-distributee-list*

H805 *alignee-or-distributee* **is** *alignee*
**or** *distributee*

制約: 共通ブロックの実体は、DYNAMIC と宣言することはできず、また、DYNAMIC である実体 (またはテンプレート) に整列することはできない。(ここで制約されているようなことをしたいならば、共通ブロックの代わりにモジュールを使わなければならない。)

制約: 構造型の成分は、POINTER 属性をもつ場合だけ DYNAMIC 属性をもつことができる。(詳しくは、8.9 節を参照されたい。)

制約: SAVE 属性をもつ実体は、DYNAMIC と宣言することはできず、また、DYNAMIC である実体 (またはテンプレート) に整列することはできない。

REALIGN 指示文は、DYNAMIC 属性をもたない *alignee* に適用することはできない。REDISTRIBUTE 指示文は、DYNAMIC 属性をもたない *distributee* に適用することはできない。

DYNAMIC 指示文は、Fortran の属性に関する文法がそうであるように、他の指示文や任意の順序で並べた属性と組み合わせることができる。

例:

```
!HPF$ DYNAMIC A,B,C,D,E
!HPF$ DYNAMIC:: A,B,C,D,E
!HPF$ DYNAMIC, ALIGN WITH SNEEZY:: X,Y,Z
!HPF$ ALIGN WITH SNEEZY, DYNAMIC:: X,Y,Z
!HPF$ DYNAMIC, DISTRIBUTE(BLOCK, BLOCK) :: X,Y
!HPF$ DISTRIBUTE(BLOCK, BLOCK), DYNAMIC :: X,Y
```

最初の二つの例は、両方とも全く同じことを意味する。その次の二つの例は、両方とも全く同じことを意味する。最後の二つの例は、両方とも全く同じことを意味する。

以下の三つの指示文は、

```
!HPF$ TEMPLATE A(64,64),B(64,64),C(64,64),D(64,64)
!HPF$ DISTRIBUTE(BLOCK, BLOCK) ONTO P:: A,B,C,D
!HPF$ DYNAMIC A,B,C,D
```

次のように一つの指示文に組み合わせることができる。

```
!HPF$ TEMPLATE, DISTRIBUTE(BLOCK, BLOCK) ONTO P,   &
!HPF$   DIMENSION(64,64),DYNAMIC :: A,B,C,D
```

割付け実体も、DYNAMIC 属性をもってもよい。ALLOCATE 文の直後に REDISTRIBUTE や REALIGN 指示文が続く場合、その文法上の意味は、もし静的に宣言されたマッピングがあるならば、まずそれを使って配列が生成され、その後すぐに配列は再マップされる。しかし実際には、次のような明らかな最適化が可能である。すなわち、一回のステップで、配列を再マップされようとしているプロセッサで生成することである。HPF 実装者には、この最適化を実装することを勧告し、HPF プログラマには、この最適化に依存することを奨励する。ここで例を示す:

```
      REAL,ALLOCATABLE(:,:) :: TINKER, EVERS
!HPF$ DYNAMIC :: TINKER, EVERS
```

```
      REAL, ALLOCATABLE :: CHANCE(:)
!HPF$ DISTRIBUTE(BLOCK),DYNAMIC :: CHANCE
      ...
      READ 6,M,N
      ALLOCATE(TINKER(N*M,N*M))
!HPF$ REDISTRIBUTE TINKER(CYCLIC, BLOCK)
      ALLOCATE(EVERS(N,N))
!HPF$ REALIGN EVERS(:,:) WITH TINKER(M::M,1::M)
      ALLOCATE(CHANCE(10000))
!HPF$ REDISTRIBUTE CHANCE(CYCLIC)
```

CHANCE はデフォルトにより常に BLOCK 分散で割り付けられるが、直ちに CYCLIC 分散に再マップされることをコンパイラは知ることができなければならない。同様な所見は、TINKER や EVERS にも適用される。( EVERS は TINKER に、まばらに伸ばす方法でマップされることに注意されたい。EVERS の隣り合った要素は、TINKER の刻み幅 M で離された要素にマップされる。EVERS(1,1) は TINKER(M,1) にマップされるため、このまばらに伸ばすマッピングは、TINKER の左下隅に置かれる。 )

⇑ 5.1 節には、DO ループの中で実行された場合、ループの繰返しが互いに干渉することになり、それによって、ループが INDEPENDENT であると特徴づけることを妨げる操作の一覧が示されている。その一覧には、以下の操作も加わる:

- ループの中で実行される REALIGN および REDISTRIBUTE 指示文は、同じデータへの参照や同じデータの他の再マッピングと干渉する。

  【仕様の根拠】 REALIGN および REDISTRIBUTE は、配列要素をもつプロセッサを変更することができるため、その要素の代入や引用 (use) と干渉する。同様に、同時に起る複数の再マッピング操作は、ある要素を複数の場所 (location) に配置してしまう。【以上】

## 8.6 再マッピングと副プログラム・インタフェース

副プログラムの仮引数が DYNAMIC 属性をもっていたならば、その副プログラムに関しては明示的引用仕様が必要である (8.14 節 参照)。REALIGN および REDISTRIBUTE 指示文の副プログラム引数インタフェースとの相互作用のルールは、以下のとおりである:

1. 仮引数は、DYNAMIC と宣言することができる。ただし、それは、結合するテンプレートを表すための、配列名の利用に関する一般的な制約を受ける。
   手続が呼出し元に戻った後の仮引数の再分散に関する影響は、実引数の属性に依存する。仮引数と結合している実引数が、同様に DYNAMIC と宣言されていたならば、仮引数の明示的再マッピングは、手続から戻った後で呼出し元においても反映されている。RANGE 指示文 (8.11 節 参照) が、実引数で許される分散形式を限定するために使われた場合、新しいマッピングは、RANGE 指示文で指定された分散形式の一つにマッチしなければならない。
   DYNAMIC 属性をもつ実引数と結合する仮引数は、結合する実引数が部分配列ではなく全体配列である場合に限り、*alignee* や *distributee* として REALIGN および REDISTRIBUTE に

指定することができる。
仮引数と結合する実引数が DYNAMIC と宣言されていないならば、呼出しからの戻り時に実引数の元々のマッピングに戻されなければならない。副プログラムから戻り、呼出し元が実行を再開するときに、呼出し元で参照でき DYNAMIC と宣言されていないすべての実体は、呼出し前と全く同じようにマップされている。

2. 配列またはその部分が二つまたはそれ以上のパスで参照可能な場合、それらのパスのいずれかを使って再マップすることは、HPF 規格合致ではない。たとえば、配列が実引数として引き渡された場合、副プログラムがその呼出しから戻るまでの間は、その配列を再整列することや、その配列が呼出し時に整列している配列やテンプレートを再分散することは禁止されている。これは、ぞっとするような別名問題を防止する。例を示す:

```
      MODULE FOO
      REAL A(10,10)
!HPF$ DYNAMIC ::  A
      END

      PROGRAM MAIN
      USE FOO
      CALL SUB(A(1:5,3:9))
      END

      SUBROUTINE SUB(B)
      USE FOO
      REAL B(:,:)
!HPF$ DYNAMIC ::  B
      ...
!HPF$ REDISTRIBUTE A            ! 規格合致でない
      ...
      END
```

「規格合致でない」と記された文の位置での A への代入が禁止されているのと同じ理由で、このような場合も禁止されている。一般に、別名の理由により変数への代入が規格合致でない**任意の**場面において、その変数を REALIGN や REDISTRIBUTE 指示文を使って明示的に再マップすることは、同様に禁止されている。
変数が、一つのパスでのみ参照可能であり DYNAMIC と宣言されていれば、副プログラムの中で親子結合や参照結合されていても、再マップすることが許されていることに注意されたい。そのような再マッピングは、副プログラムから呼出し元に戻った後でさえも有効である。

## 8.7 部分プロセッサへのマッピング

この拡張は、プロセッサが記述できる場所 (例えば DISTRIBUTE 指示文の中) に、部分プロセッサを指定することを認めることで、実体を直接的に部分プロセッサへ分散することを可能に

する。この指定された部分は、プロセッサ構成の適当な部分集合でなければならない。

拡張された *dist-target* の構文は次のとおりである。

H806 *extended-dist-target* **is** *processors-name* [ ( *section-subscript-list* ) ]
**or** * *processors-name* [ ( *section-subscript-list* ) ]
**or** *

制約: 部分配列添字並び (*section-subscript-list*) の部分配列添字 (*section-subscript*) は、ベクトル添字 (*vector-subscript*) であってはならず、添字 (*subscript*) または添字三つ組 (*subscript-triplet*) でなければならない。

制約: 部分配列添字並び (*section-subscript-list*) では、部分配列添字 (*section-subscript*) の数は、*processor-name* の次元数と等しくなければならない。

制約: DISTRIBUTE 指示文の中では、それぞれの部分配列添字 (*section-subscript*) は、宣言式 (*specification-expr*) でなければならない。

制約: DISTRIBUTE および REDISTRIBUTE 指示文の中で、*dist-format-list* と *dist-target* の両方が指定されたとき、"*" でない *dist-format-list* の要素の数は、指定されたプロセッサ構成の添字三つ組 (*subscript-triplet*) の数と等しくなければならない。

制約: DISTRIBUTE および REDISTRIBUTE 指示文の中で、*dist-format-list* がなく *dist-target* が指定されたとき、それぞれの *distributee* の次元数は、指定されたプロセッサ構成の添字三つ組 (*subscript-triplet*) の数と等しくなければならない。

```
!例 1
!HPF$ PROCESSORS P(10)
      REAL A(100)
!HPF$ DISTRIBUTE A(BLOCK) ONTO P(2:5)



!例 2
!HPF$ PROCESSORS Q(10,10)
      REAL A(100,100)
!HPF$ DISTRIBUTE B(BLOCK,BLOCK) ONTO Q(5:10,5:10)
```

例 1 では、配列 A は、プロセッサ P(2) から P(5) にかけてブロック分散で分散されている。二つ目の例では、配列 B は、プロセッサ構成 Q の右下の 4 分の 1 に分散されている。

【利用者への助言】 この拡張は、独立な複数の計算段階をプロセッサの別々の部分で同時に実行することを可能にするタスキング構文 (9.4 節 参照) と一緒に使うと特に有用である。このような状況は、プログラムが並列に計算することができる複数のデータ構造を利用する場合に発生し、このとき、それぞれの独立なデータ構造の計算もさらに並列性を示す。ある種の流体力学計算で使われるマルチブロック格子における多重ブロックが一例である。ここでは、それぞれの独立なブロックは、並列性の両方のレベルを活かすために部分プロセッサ上に分散されていなければならない。【以上】

## 8.8 ポインタ

### 8.8.1 マップされたポインタ

HPF の公認拡張として、ポインタと指示先は、明示的にマップすることができる。形式的には、それぞれ 3.3 節および 3.4 節で述べた次の制約を除かなければならないことを意味する: ⇑
*distributee* や *alignee* は、POINTER または TARGET 属性をもつことはできない。

割付け実体の場合もそうであったように、ポインタへのマッピング指示は、すぐには効果を表さない。しかし、ポインタが割付けやポインタ代入によって指示先と結合したポインタとなったときに、役割を果たす。

明示的なマッピングをもつポインタが ALLOCATE 文で使われたとき、そのデータは指定されたマッピングで割り付けられる。

例えば:

```
      REAL, POINTER, DIMENSION(:) :: A, B
!HPF$ ALIGN B(I) WITH A(I)
!HPF$ DISTRIBUTE A(BLOCK)
      ...
      ALLOCATE(A(100))
      ALLOCATE(B(50))
      ...
      ALLOCATE(B(200))                    ! 規格合致でない
```

ポインタ A は、BLOCK 分散をもつと宣言され、一方、ポインタ B は、A と同値に整列すると宣言されている。A が割り付けられるときには、ブロック分散で生成される。B が割り付けられるときには、A の最初の 50 要素に整列する。実体が整列する先の実体がすでに生成されたり割り付けられた場合にだけ実体は整列できるため、ALLOCATE 文の順序を入れ換えてはならないことに注意されたい。また、より大きな実体 (ここでは B) は、より小さな実体 (ここでは A) に整列することはできないため、B の二番目の割付けは規格合致ではない。

明示的マッピングをもつポインタ P は、次に述べる条件の下で、ポインタ代入文により指示先 T と結合しているポインタとなることができる。

1. T のマッピングは、P のマッピングの特殊化である (特に、T は全体配列でなければならない)。さらに、
2. P が明示的に整列しているとき、その最終整列先は転写的でなく完全に指定された分散をもたなければならない。さらに、
3. P と T は、共に DYNAMIC であるか、共にそうでないかのどちらかである。

いくつかの例を示す:

```
      REAL, POINTER, DIMENSION(:,:) :: P
!HPF$ DISTRIBUTE P(BLOCK,BLOCK)
      REAL, TARGET, DIMENSION (100, 100) ::  B, C, D
!HPF$ DISTRIBUTE B(BLOCK, BLOCK)
!HPF$ DISTRIBUTE C(BLOCK, CYCLIC)
```

```
      ...
      P => B               ! 規格合致である
      P => B(1:50, 1:50)   ! 規格合致でない
                           !    : 指示先は全体配列でなければならない。
      P => C               ! 規格合致でない: 第 2 次元の分散が一致しない。
      P => D               ! 規格合致でない: D は明示的にマップされていない。
      ...
```

上記のポインタ代入 `P => B(1:50,1:50)` が規格合致でない直観的な理由は、53 ページの例 (`INHERIT A` と `DISTRIBUTE A * ONTO *`の違いを示している) が規格合致でない理由と似ている: 例えば、配列 B が、$2 \times 2$ のプロセッサ構成に分散されているとしよう。このとき、部分配列 `B(1:50, 1:50)` は、すべてプロセッサ $(1, 1)$ 上に存在する。そのため、このマッピングは、P に対する `(BLOCK, BLOCK)` 分散では正しく記述できない。

次のポインタ代入文は、ポインタに対してプロセッサ構成が指定されていないにも関わらず、有効である。この例では、B のマッピングは P のマッピングの特殊化である。

```
      REAL, POINTER, DIMENSION(:) :: P
      REAL, TARGET, DIMENSION(100) ::  B
!HPF$ PROCESSORS PROC(NUMBER_OF_PROCESSORS())
!HPF$ DISTRIBUTE P(BLOCK)
!HPF$ DISTRIBUTE (BLOCK) ONTO PROC :: B
      ...
      P => B               ! 規格合致である
      ...

      REAL, POINTER, DIMENSION(:) :: P
!HPF$ DISTRIBUTE * :: P
      REAL, TARGET, DIMENSION(100) ::  B, C
!HPF$ DISTRIBUTE B(BLOCK), C(CYCLIC)
      ...
      P => B               ! 規格合致である
      P => C               ! 規格合致である
      P => C(1:50)         ! 規格合致でない
                           !    : 指示先は全体配列でなければならない。
      ...
```

ここで、ポインタ P に対して、転写的分散を示すために、*が使われている。そのため、それぞれ BLOCK や CYCLIC で分散された指示先 B および C とポインタ結合するポインタになることができる。しかし、それでも `C(1:50)` のような部分配列を指すために使うことはできない。そうするためには、ポインタは INHERIT 属性をもたなければならない。

```
      REAL, POINTER, DIMENSION(:) :: P
!HPF$ INHERIT :: P
      REAL, TARGET, DIMENSION(100) ::  B, C
```

```
!HPF$ DISTRIBUTE B(BLOCK), C(CYCLIC)
      ...
      P => B              ! 規格合致である
      P => C              ! 規格合致である
      P => C(1:50)        ! 規格合致である
      ...
```

ポインタが転写的分散をもつことを可能にするために 3.3 節で明記された *dist-format-clause* の制約を次のように変更しなければならない (変更部分は下線を付けて示す)。

制約: DISTRIBUTE 指示文で、*dist-format-clause* または *dist-target* のどちらかが "*" で始まるときは、すべての *distributee* は、*distributee* が POINTER 属性をもつ場合を除き、仮引数でなければならない。

3.4 節で明記された *align-spec* への制約は、次のように変更されなければならない:

制約: ALIGN 指示文の *align-spec* が "*" で始まるときは、すべての *alignee* は、*alignee* が POINTER 属性をもつ場合を除き、仮引数でなければならない。

4.4.2 項で明記された *inheritee* への制約は、次のように変更されなければならない:

制約: *inheritee* は、*alignee* が POINTER 属性をもつ場合を除き、仮引数でなければならない。

そのような転写的マッピングをもつポインタが ALLOCATE 文で使われた場合、コンパイラは、割り付けられたデータに対して任意のマッピングを選んでよい。RANGE 宣言 (8.11 節 参照) を分散形式の集合を限定するために使ってもよい。

ポインタが DYNAMIC 属性をもっている場合、そのポインタが結合している指示先 (したがって、それは DYNAMIC 属性をもっていなければならない) は、以下の制約の範囲において、REALIGN または REDISTRIBUTE 指示文を使って再マップすることができる。

ポインタは、部分配列ではなく全体配列と結合している場合に限り、REALIGN または REDISTRIBUTE 指示文において *alignee*、 *align-target*、 または *distributee* として使ってもよい。

実体が再マップされたとき、その実体と結合しているどのポインタを介しても、新しいマッピングが反映されていることに注意されたい。

### 8.8.2 ポインタと副プログラム

ポインタ仮引数が明示的にマップされていない場合、実引数も明示的にマップされていてはならない。

ポインタ仮引数が明示的なマッピングをもつとき、次の一つの例外を除いて、実引数は上述したポインタ代入のルールに従わなければならない: 実引数が DYNAMIC 属性をもつ場合、対応する仮引数は DYNAMIC 属性をもつ必要はない。すなわち、153 ページの項目 3 は、次のように弱められる。

3. ポインタ仮引数が DYNAMIC 属性をもつ場合、対応する実引数は DYNAMIC 属性をもたなければならない。

RANGE 宣言 (8.11 節 参照) を、実引数の分散形式の集合を限定するために使うことができる。

ポインタ仮引数は、DYNAMIC 属性をもつことができる。この場合、実引数も DYNAMIC 属性をもたなければならない。仮引数と結合している指示先は、一つ前の節で述べた制約の範囲で、再分散されても構わない。Fortran での規則に従って、実引数が (親子結合や参照結合により) 仮引数以外を通して参照できる場合、指示先は仮引数を通してだけ再分散してもよい。仮引数が再分散されたならば、手続から戻ったとき実引数は新しいマッピングをもつ。このような場合、新しいマッピングは、実引数のレンジ制限 (もしあるならば) を守らなければならない。

### 8.8.3 ポインタや指示先に関する制限事項

手続 P の呼出しで、(a) 仮引数が TARGET 属性をもっている、かつ、(b) 対応する実引数が TARGET 属性をもっていてベクトル添字をもつ部分配列ではない (すなわち、実体 A または配列 A の部分である) 場合、プログラムは以下の条件を満たす場合だけが HPF 規格合致である。

1. 呼出しの間に実引数の再マッピングが起きない。または、
2. 次の制約を満たすために、プログラムの実行の残りの部分で影響がない。
   (a) 呼出しの前に A のいずれかの部分と結合しているポインタは、P の入口でポインタ結合状態が不定となり、P の実行中に再代入されない限り、出口で入口直前のポインタ結合状態にもどされる。
   (b) P の実行中に仮引数のいずれかの部分や A のいずれかの部分と結合しているポインタは、P の出口ではポインタ結合状態が不定になる。

【利用者への助言】 再マッピングが起きないことを保証する一つの方法は、仮引数に INHERIT 属性を与えることである。【以上】

【仕様の根拠】 これらの制限事項は、副プログラム境界での暗黙的な再マッピングにおいて、Fortran 規格 (F95:12.4.1.1) の次のような部分をサポートするために設けられた。

> 手続を呼び出した時、仮引数が TARGET または POINTER 属性をもっていない場合には、実引数と結合しているポインタは、対応する仮引数とは結合しない。
>
> 仮引数が TARGET 属性をもっていて、対応する実引数も TARGET 属性をもっていて、実引数がベクトル添字をもつ部分配列でない場合には:
>
> 1. 手続を呼び出した時、実引数と結合しているポインタは、対応する仮引数と結合する。
> 2. 手続の実行が終了した時点で、仮引数と結合しているポインタは、実引数との結合を保持する。
>
> 仮引数が TARGET 属性をもっていて、対応する実引数が TARGET 属性をもっていない、または実引数がベクトル添字をもつ部分配列である場合には、手続の実行が終了した時点で、仮引数と結合しているいずれのポインタも不定になる。

【以上】

本章での制限事項を表した例を示す:

```
      INTEGER, TARGET, DIMENSION (10) :: ACT
      INTEGER, POINTER, DIMENSON (:) :: POINTS_TO_ACT, POINTS_TO_DUM
!HPF$ DISTRIBUTE ACT(BLOCK)

      POINTS_TO_ACT => ACT
      CALL F(ACT)
      POINTS_TO_DUM(1) = 1             ! 不正

      CONTAINS
        SUBROUTINE F(DUM)
          INTEGER, TARGET, DIMENSION(10) :: DUM
        !HPF$ DISTRIBUTE DUM(CYCLIC)

          POINTS_TO_DUM => DUM
          POINTS_TO_ACT(1) = 1         ! 不正
        END SUBROUTINE
      END
```

POINTS_TO_DUM(1) への代入は、項目 2b に違反しているために、不正である。項目 2a に違反しているために、POINTS_TO_ACT(1) への代入も、不正である。

## 8.9 構造体成分のマッピング

ALIGN 指示文、DISTRIBUTE 指示文、および DYNAMIC 指示文は、構造型定義の内部で成分定義文が現れることができるところであればどこにでも現れることができる。これらの指示文に現れるすべての *alignee* と *distributee* は、その構造型定義の内部で定義される成分の名前でなければならない。構造型の成分のマッピングを可能にするために、構文規則を以下のように拡張する。

H807 *distributee-extended* **is** *object-name*
**or** *template-name*
**or** *component-name*
**or** *structure-component*

いずれかの成分が明示的にマップされている場合、または、いずれかの成分が明示的にマップされた型である場合、その構造型は、**明示的にマップされた型**であると言う。

制約: 構造型の成分は、その型が明示的にマップされていない場合に限って明示的に分散することができる。

制約: 構造型のデータ実体は、その構造型が明示的にマップされた型でない場合に限って明示的に分散することができる。

制約: DISTRIBUTE 指示文の *distributee* は、構造体成分 (*structure-component*) であってはならない。

制約: 構造型定義の中に現れる DISTRIBUTE 指示文の *distributee* は、構造型の成分の成分名 (*component-name*) でなければならない。

制約: 構造型定義の中に現れる DISTRIBUTE 指示文に限って、*distributee* を成分名 (*component-name*) とすることができる。

制約: REDISTRIBUTE 指示文の中でだけ *distributee* を構造体成分にすることができる。このとき、右端を除くすべての部分参照はスカラ (0 次元) でなければならない。構造体成分の右端の部分参照は、DYNAMIC 属性をもっていなければならない。

H808 *alignee-extended* **is** *object-name*
**or** *component-name*
**or** *structure-component*

制約: 構造型の成分は、その型が明示的にマップされていない場合に限って明示的に整列することができる。

制約: 構造型のデータ実体は、その構造型が明示的にマップされた型でない場合に限って明示的に整列することができる。

制約: ALIGN 指示文の *alignee* は、構造体成分 (*structure-component*) であってはならない。

制約: 構造型定義の中に現れる ALIGN 指示文の *alignee* は、構造型の成分の成分名 (*component-name*) でなければならない。

制約: 構造型定義の中に現れる ALIGN 指示文に限って、*alignee* を成分名 (*component-name*) とすることができる。

制約: REALIGN 指示文の中でだけ *alignee* を構造体成分 (*structure-component*) にすることができる。このとき、右端を除くすべての部分参照はスカラ (0 次元) でなければならない。構造体成分の右端の部分参照は、DYNAMIC 属性をもっていなければならない。

H809 *align-target-extended* **is** *object-name*
**or** *template-name*
**or** *component-name*
**or** *structure-component*

制約: 構造型定義の中に現れる ALIGN 指示文に限って、整列先を成分名 (*component-name*) とすることができる。

制約: *align-target* が構造体成分 (*structure-component*) であるとき、その右端を除くすべての部分参照はスカラ (0 次元) でなければならない。

上記の制約で言及されているように、構造型の定義の中で構造型の成分をマップしておくと、その型のデータ実体を生成するときに、その成分は指定したとおりにマップされる。

以下の例を考える。

```
      TYPE DT
        REAL C(100)
!HPF$   DISTRIBUTE C(BLOCK) ONTO P
      END TYPE DT

      TYPE (DT) :: S1
      TYPE (DT) :: S2(100)
```

構造型の中の配列 C は、ブロック分散で宣言されている。したがって、構造型 DT のスカラ変数 S1 がもつ構造体成分 S1%C は、プロセッサ構成 P にブロック分散される。同様に、配列 S2 のすべての要素について、成分 C はプロセッサ配列 P にブロック分散される。

構造型の定義の中の ALIGN 指示文によって、構造型の成分を同じ構造型の別の成分に対して整列させたり、他のデータ実体に対して整列させたりすることができる。構造体成分は、構造型の成分などの他のデータ実体の整列先として使用することができる。

例:

```
!HPF$ TEMPLATE T(100)
!HPF$ DISTRIBUTE T(CYLIC)

      TYPE DT
        REAL, DIMENSION(100) :: A, B, C
!HPF$   ALIGN WITH A :: B
!HPF$   DISTRIBUTE (BLOCK) :: A
!HPF$   ALIGN WITH T :: C
      END TYPE DT
```

ここでは、構造型 DT の変数は、以下のように生成される。

- 成分 B は A に対して整列する。
- A はブロック分散される。
- 成分 C はテンプレート T に対して整列する。T は構造型定義の外で定義される。

構造型の成分のマッピングの指定が不完全な場合、その成分のマッピングをどう補うかはコンパイラに任されていることに注意されたい。しかし実際には、コンパイラはそのような構造型の成分のマッピングを、変数によらずすべて同じにすると思ってよい。例えば、上のコードで成分 A の分散がなかった場合について考えてみる。B と A は整列するように宣言されているが、A の分散は与えられていない。そのような場合、構造型 DT のすべての変数は、成分 A(または言い換えると成分 B)が同じ分散となるように生成されると考えてよい。

構造型成分のマッピングに関する制約により、構造体変数のマッピングは 1 レベルだけが許されている。例として、構造型の成分が構造型であるような以下のコードを考える。

```
      TYPE SIMPLE
        REAL S(100)
!HPF$   DISTRIBUTE S(BLOCK)
      END TYPE SIMPLE

!HPF$ TEMPLATE, DISTRIBUTE(BLOCK, *) :: HAIRY_TEMPLATE(47,73)

      TYPE COMPLICATED
        INTEGER SIZE
        REAL RV(100,100), KV(100,100), QV(47,73)
! 配列 RV, KV, QV はマップできる。
!HPF$   DISTRIBUTE (BLOCK, BLOCK):: RV, KV
!HPF$   ALIGN WITH HAIRY_TEMPLATE :: QV
        TYPE(SIMPLE) SV(100)
! SIMPLE は明示的にマップされた型なので、次の指示文は誤り。
!HPF$   DISTRIBUTE SV(BLOCK)
      END TYPE COMPLICATED

      TYPE(COMPLICATED) LOTSOF(20)

! COMPLICATED は明示的にマップされた型なので、次の指示文は誤り。
!HPF$ DISTRIBUTE LOTSOF(BLOCK)
```

ここでは、構造型の成分 SIMPLE はマップされている。このような型のデータ実体 (例えば COMPLICATED 型の SV) は、分散することができない。配列 LOTSOF も同じ理由で分散できない。

POINTER 属性をもつ構造体成分は、DYNAMIC 宣言されている場合には、REALIGN 指示文または REDISTRIBUTE 指示文で再マップすることができる。例えば以下のコードは、それ自身がブロック格子である複数のブロック (ここでは SUBGRID と呼ぶ) を割り付けてマップするために使用される。

```
!HPF$ PROCESSORS P( number_of_processors() )

      TYPE SUBGRID
        INTEGER SIZE
        INTEGER LO, HI          ! 分散先となる部分プロセッサの範囲
        REAL, POINTER BL(:)
!HPF$   DYNAMIC BL
      END TYPE SUBGRID

      TYPE (SUBGRID), ALLOCATABLE :: GRID(:)

      READ (*,*) SUBGRID_COUNT
```

```
      ALLOCATE GRID(SUBGRID_COUNT)
      DO I = 1, SUBGRID_COUNT
        READ(*,*) GRID(I)%SIZE
      END DO

! それぞれの subgrid に対する部分プロセッサを計算し、
! LO と HI の値を設定する。
      CALL FIGURE_THE_PROCS ( GRID, number_of_processors())
! それぞれの subgrid を割り付け、計算した部分プロセッサに対して分散する。
      DO I = 1, SUBGRID_COUNT
        ALLOCATE( GRID(I)%BL( GRID(I)%SIZE ) )
!HPF$   REDISTRIBUTE GRID(I)%BL(BLOCK) ONTO P( GRID(I)%LO : GRID(I)%HI )
    END DO
```

【仕様の根拠】 構造型の成分は、それが DYNAMIC 属性だけでなく POINTER 属性ももっているときに限って再マップすることができる。この制約は、HPF 指示文を使って直接に指定できないマッピングを禁止するために置かれている。例えば、以下のコードを考えてみる。

```
  !HPF$ PROCESSORS P(4)

        TYPE DT
          REAL C(100)
  !HPF$   DISTRIBUTE C(BLOCK) ONTO P
  !HPF$   DYNAMIC C                    ! 規格合致でない
        END TYPE DT

        TYPE (DT) :: S(10)
         ...
        J = 3
         ...
  !HPF$ REDISTRIBUTE S(J)%C(CYCLIC) ONTO P

          ... S(:)%C(2) ...
```

ここでは、構造型 DT の成分 C は DYNAMIC と宣言されている。したがって、配列変数 S は 10 個の配列要素から成り、それぞれの配列要素は最初はブロック分散された成分 C をもつ構造体となる。REDISTRIBUTE 指示文により、S の第 J 要素の構造体成分 C はサイクリック分散になるように再マップされる。ここで、式 S(:)%C(2) によって参照されるデータ実体のマッピングについて考えてみる。これは、配列変数 S を構成する 10 個の構造体のそれぞれから、2 番目の要素を取り出したものである。S の 1 要素 (この場合は 3 番目の要素) を再分散しているため、データ実体の他のすべての要素はプロセッサ P(1) 上に存在するが、第 3 要素だけはプロセッサ P(2) に存在する。このような分散は、HPF 指示文では直接に宣言することはできない。

Fortran 規格では、成分が POINTER 属性をもつ場合のこのような式を禁止している。すなわち、データ参照中の部分名 (*part-name*) が POINTER 属性をもつ場合、それよりも左側の部分参照 (*part-ref*) はスカラでなければならない (F95:6.1.2)。そこで、我々は以下のような規約によって上記のような状態を避ける。

- POINTER 属性をもたない構造体成分の再マッピングは禁止する。
- POINTER 属性をもつ構造体成分について、上記のような式を禁止するために、Fortran 規格の規約に従う。

【以上】

## 8.10 新しい分散形式

本章では、二つの新しい分散形式について述べる。構文は以下のように拡張される。

H810 *extended-dist-format* **is** BLOCK [ ( *int-expr* ) ]
**or** CYCLIC [ ( *int-expr* ) ]
**or** GEN_BLOCK ( *int-array* )
**or** INDIRECT ( *int-array* )
**or** *

制約: DISTRIBUTE 指示文または REDISTRIBUTE 指示文の *extended-dist-format* の中に現れる整数型配列 (*int-array*) は、整数型の一次元配列でなければならない。

制約: DISTRIBUTE 指示文の *extended-dist-format* の中に現れる整数型配列 (*int-array*) は、制限式 (*restricted-expr*) でなければならない。

制約: GEN_BLOCK 分散に現れる整数型配列 (*int-array*) の大きさは、分散先のプロセッサ構成の対応する次元の寸法と等しくなければならない。

制約: INDIRECT 分散に現れる整数型配列 (*int-array*) の大きさは、その分散が適用される *distributee* の対応する次元の寸法と等しくなければならない。

ブロック分散を「一般化」した GEN_BLOCK は、配列を不均等な大きさの連続区間に分けてプロセッサにマップさせる。各区間の大きさは利用者定義の整数型のマッピング配列の値によって指定される。つまり、マッピング配列の i 番目の要素は、分散先プロセッサ構成の i 番目のプロセッサに配置されるブロックの大きさを指定する。そのため、マッピング配列の値は負でない数に制約され、その総和は分散配列の対応する次元の寸法と比較して等しいか大きくなければならない。

マッピング配列は、DISTRIBUTE 指示文で使用される場合には制限式でなければならないが、REDISTRIBUTE 指示文では配列変数であってもよい。後者の場合、マッピング配列の値が指示文が実行された後で変化しても、分散配列のマッピングは変化しない。

ここで、$l$ と $u$ をそれぞれ *distributee* のある次元の下限値と上限値、$MAP$ をマッピング配列とし、*BS(i):BE(i)* を分散先であるプロセッサ構成の対応する次元の i 番目のプロセッサ

にマップされる要素とする。すると、次式が成り立つ。

$$
\begin{aligned}
BS(1) &= l, \\
BE(i) &= \min(BS(i) + MAP(i) - 1, u), \\
BS(i) &= BE(i-1) + 1.
\end{aligned}
$$

例:

```
      PARAMETER (S = (/2,25,20,0,8,65/))
!HPF$ PROCESSORS P(6)
      REAL A(100), B(200), new(6)
!HPF$ DISTRIBUTE A( GEN_BLOCK( S) ) ONTO P
!HPF$ DYNAMIC  B
        ...
      new = ...
!HPF$ REDISTRIBUTE B( GEN_BLOCK(new) )
```

上記の宣言で、配列要素 A(1:2) は P(1) にマップされ、A(3:27) は P(2) にマップされ、A(28:47) は P(3) にマップされ、P(4) にマップされるものはなく、A(48:55) は P(5) にマップされ、A(56:100) は P(6) にマップされる。配列 $B$ の分散は、実行時に値が計算される配列 $new$ によって決まる。

> 【実装者への助言】 一般化されたブロック分散を使った分散配列の要素のアクセスのためには、実行時にマッピング配列の値をアクセスしなければならない。しかし、マッピング配列の大きさはプロセッサ構成の大きさと等しい (あまり大きくない) ので、ほとんどの場合にはすべてのプロセッサに複製してしまえばよい。
>
> 動的配列に対しては、マッピング配列のコピーをシステム内で個々に管理して、マッピング配列の値が変化しても分散配列のアクセスに影響を与えないようにしなければならない。【以上】

問題領域の構造が Fortran のデータ構造に直接対応付けできないような科学分野のアプリケーションは数多くある。例えば、多くの CFD アプリケーションでは、問題領域の表現に構造化できないメッシュ (2 次元での三角形、3 次元での四面体など) が使用される。このようなメッシュの格子点は一般に 1 次元配列で表現され、それらの相互の接続関係の表現にも 1 次元配列が使われる。この配列のマッピングに BLOCK や CYCLIC のような構造化された分散の機構を用いると、同じプロセッサ上に無関係な要素がマップされる結果となる。これは言い換えると、不必要な通信を大量に発生させることになる。必要なのは、関係の深い配列要素の集合を任意に同じプロセッサ上にマップさせる機構である。INDIRECT 分散は、そのような機構を提供する。

INDIRECT 分散は、データ配列のある次元の要素から分散先プロセッサ構成のある次元への多対一のマッピングを可能にする。分散する配列次元の個々の要素と分散先プロセッサの対応を指定するために、一つの整数型配列が使用される。すなわち、このマッピング配列の i

番目の要素は、i 番目の配列要素がマップされる先のプロセッサ番号を表している。マッピング配列は配列要素からプロセッサ要素へのマッピングを示すので、マッピング配列の寸法は分散される配列のその次元の寸法と一致しなければならない。また、マッピング配列の値は、分散先となるプロセッサ構成の次元の下限値と上限値の間になければならない。

マッピング配列は、DISTRIBUTE 指示文で使用される場合には制限式でなければならないが、REDISTRIBUTE 指示文では配列変数であってもよい。後者の場合、マッピング配列の値が指示文が実行された後で変化しても、分散配列のマッピングは変化しない。

例:

```
!HPF$ PROCESSORS P(4)
        REAL A(100), B(50)
        INTEGER map1(100), map2(50)
        PARAMETER (map1 = (/1,3,4,3,3,2,1,4, ..../))
!HPF$ DYNAMIC B
!HPF$ DISTRIBUTE A( INDIRECT(map1) ) ONTO P
!HPF$ DISTRIBUTE map2(BLOCK) ONTO P

        map2 = ...
!HPF$ REDISTRIBUTE B( INDIRECT(map2) ) ONTO P
        ....
```

ここで、配列 A は定数配列 map1 を使って静的に分散されている。したがって、以下のようになる。

A(1) は P(1) へマップされる
A(2) は P(3) へ
A(3) は P(4) へ
A(4) は P(3) へ
A(5) は P(3) へ
A(6) は P(2) へ
A(7) は P(1) へ
A(5) は P(4) へ
...

配列 B は動的であると宣言され、マッピング配列 map2 を使って再分散されている。

【実装者への助言】 一般に、INDIRECT 分散は、マッピング配列となる配列変数をもった REDISTRIBUTE 指示文の形で使用されることになるだろう。また、マッピング配列の大きさは分散される配列の大きさと同じなので、それ自身もほとんどの場合 BLOCK 分散などで分散されるだろう。これには、いくつかの問題点がある。この分散を正しく実装するためには、実行時システムはマッピング配列の (分散された) コピーを管理して、プログラムがマッピング配列を変更しても分散が変化しないようにしなければならない。マッピング配列として配列変数を使用することは、配列のそれぞれの要素の配置が実行時まで分からないということを意味する。したがって、指定された配列要素の配置を計算するために、通信が必要となるであろう。【以上】

## 8.11 RANGE 指示文

RANGE 属性は、DYNAMIC 属性または転写的な分散形式をもつデータ実体 (ポインタを含む) とテンプレートについて、あり得る分散形式を限定するために使用される。

H811 *range-directive* **is** RANGE *ranger range-attr-stuff*

H812 *ranger* **is** *object-name*
**or** *template-name*

H813 *range-attr-stuff* **is** *range-distribution-list*

H814 *range-distribution* **is** ( *range-attr-list* )

H815 *range-attr* **is** *range-dist-format*
**or** ALL

H816 *range-dist-format* **is** BLOCK [ ( ) ]
**or** CYCLIC [ ( ) ]
**or** GEN_BLOCK
**or** INDIRECT
**or** *

制約: 少なくとも以下のいずれかが成り立たなければならない。

- *ranger* は DYNAMIC 属性をもつ。
- *ranger* は INHERIT 属性をもつ。
- *ranger* は DISTRIBUTE 指示文か *combined-directive* で指定され、その *dist-format-clause* は*である。

制約: *range-attr-list* の長さはそれぞれ、*ranger* の次元数と等しくなければならない。

制約: *ranger* は ALIGN 指示文または REALIGN 指示文の *alignee* であってはならない。

*range-attr-list* の長さはそれぞれ *ranger* の次元数と等しいので、*range-distribution* 中の *range-attr* ($R$ とする) は *ranger* の次元 ($D$ とする) にそれぞれ順番に対応する。つまり、次元 $D$ は *ranger* が最終的に整列するテンプレートの座標軸 $A$ に対応するか (この場合順番が対応するとは限らない)、あるいは、テンプレートのどの座標軸とも対応しない。

この記法を使って、*ranger* に対する RANGE 属性の意味を表現すると以下のようになる:

*range-distribution-list* 中の少なくとも一つの *range-distribution* について、すべての *range-attr* $R$ は以下のどちらかである。

- 対応する座標軸 $A$ が存在する場合、その座標軸の分散形式と適合している。
- 対応する座標軸が存在しない場合、*または ALL である。

この適合性は、*ranger* が *distributee* として現れるどんな DISTRIBUTE 指示文や REDISTRIBUTE 指示文に対しても維持されなければならない。また、*ranger* が POINTER 属性をもっていて転写的に分散されている場合、*ranger* が結合されるすべての指示先に対しても、この適合性が維持されなければならない。

ここで言う適合とは、分散形式によって以下のように規約される。

1. BLOCK は、*range-dist-format* の中の BLOCK、BLOCK() または CYCLIC() と適合する。
2. BLOCK(n) は、*range-dist-format* の BLOCK() または CYCLIC() と適合する。
3. CYCLIC は、*range-dist-format* の CYCLIC または CYCLIC() と適合する。
4. CYCLIC(n) は、*range-dist-format* の CYCLIC() と適合する。
5. GEN_BLOCK(a) は、*range-dist-format* の GEN_BLOCK と適合する。
6. INDIRECT(a) は、*range-dist-format* の INDIRECT と適合する。
7. *は、*range-dist-format* の*と適合する。

すべての分散形式は、*range-dist-format* の ALL と適合する。

*combined-directive* についての構文規則 H301 により、以下の形式の RANGE 指示文も許されることに注意されたい。

!HPF$ RANGE *range-attr-stuff-list* :: *ranger-list*

これは、構文規則 H801 によって定義される *combined-attribute-extended* を用いている。

例:

```
!HPF$    DISTRIBUTE T(BLOCK)
!HPF$    ALIGN A(I,J) WITH T(I)

         CALL SUB(A)
            ....

         SUBROUTINE SUB(X)
!HPF$    INHERIT X
!HPF$    RANGE X (BLOCK, *), (CYCLIC, *)
```

X の最終整列先 (この場合は継承されたテンプレート T) は 2 次元目をもたないので、X の *range-distribution* の 2 次元目には*か ALL だけを使うことができる。

```
         REAL A(100, 100, 100)
!HPF$    DISTRIBUTE A(BLOCK, *, CYCLIC)

         CALL SUB( A(:,:,1) )          ! 規格合致である
         CALL SUB( A(:,1,:) )          ! 規格合致でない
         CALL SUB( A(1,:,:) )          ! 規格合致でない
            ....
```

```
      SUBROUTINE SUB(X)
      REAL X(:, :)
!HPF$ INHERIT X
!HPF$ RANGE X (BLOCK, *)
```

サブルーチン SUB に与えられた RANGE 指示文に対して、SUB の最初の呼出しだけが規格合致である。しかし、RANGE 指示文を以下のものに置き換えるなら、この三つの呼出しをすべて規格合致にすることができる。

```
!HPF$ RANGE (BLOCK, *), (BLOCK, CYCLIC), (*, CYCLIC) :: X
```

## 8.12 SHADOW 指示文

隣接計算をするコード — 例えば、偏微分方程式の離散化や畳込みの計算 — においては、各プロセッサ上にローカルな部分配列を格納する領域を割り付けるときに、隣接したプロセッサから移動して来る要素のための余分なスペースを付加しておくという技巧が一般的に使われる。この余分な領域は、「シャドウ (shadow edge)」と呼ばれる。概念的に、配列の各次元に二つずつのシャドウがある。一つはローカルな部分配列の下端にあり、もう一つは上端にある。

一個の手続内であれば、コンパイラはどの配列がシャドウを必要としているかを知って、それに従って余分なスペースを確保することができる。しかし、シャドウ領域の幅は使用する計算方法に依存するので、別の手続では同じ配列に対して異なるシャドウ幅が必要とされることがある。したがって、手続をまたぐ解析がない場合、手続呼出しの度に、適切なシャドウ幅をもつスペースに配列引数をコピーしなければならないことがある。同様なデータ移動は、サブルーチンから出る時にデータを元の場所に戻すためにも必要となるだろう。この不必要なデータ移動は、必要なシャドウ幅を利用者が配列の宣言時に指定できるようにすることで避けられる。

シャドウ幅を宣言するための構文は以下のとおりである。

| | | | |
|---|---|---|---|
| H817 | *shadow-directive* | **is** | SHADOW *shadow-target shadow-attr-stuff* |
| H818 | *shadow-target* | **is** | *object-name* |
| | | **or** | *component-name* |
| H819 | *shadow-attr-stuff* | **is** | ( *shadow-spec-list* ) |
| H820 | *shadow-spec* | **is** | *width* |
| | | **or** | *low-width* : *high-width* |
| H821 | *width* | **is** | *int-expr* |
| H822 | *low-width* | **is** | *int-expr* |
| H823 | *high-width* | **is** | *int-expr* |

制約: *width*、 *low-width* または *high-width* として現れる整数式 (*int-expr*) は、0 以上の値をもつ宣言式 (*specification-expr*) でなければならない。

*shadow-spec* に *width* を指定することは、*shadow-spec* に *width*:*width* を指定することと等価である。

```
!HPF$    DISTRIBUTE (BLOCK) :: A
!HPF$    SHADOW (w) :: A
```

このコードでは、配列 A を BLOCK で分散し、両側に幅 w のシャドウをもつことを宣言している。A が仮引数である場合、これはコンパイラに対して、手続呼出し時の不必要なデータ移動を避けるのに十分な情報を与えている。

また、一つの次元の下端と上端に異なるシャドウ幅を指定することもできる。

```
         REAL, DIMENSION (1000) :: A
!HPF$    DISTRIBUTE(BLOCK), SHADOW(1:2) ::  A
         ....
         FORALL (i = 2, 998)
            A(i) = 0.25 * (A(i) + A(i-1) + A(i+1) + A(i+2))
         END FORALL
```

このコードでは、下端には一つだけローカルでない要素が必要で、上端には二つ必要であることを宣言している。

## 8.13 マッピングの集合における同値と半順序

新しい分散、SHADOW 属性、および構造型の成分のマッピングが公認拡張として導入されたが、これらに対応するため 4.5 節を変更する必要がある。新しい文面は以下のとおりである (追加部分は下線を付けて示す)。

⇑

最初に、*dist-format* の指定について同値の概念を定義する。

1. 「同値である」という表現に記法 $\equiv$ を用いると、

| | | | |
|---|---|---|---|
| BLOCK | $\equiv$ | BLOCK | |
| CYCLIC | $\equiv$ | CYCLIC | |
| $*$ | $\equiv$ | $*$ | |
| BLOCK($n$) | $\equiv$ | BLOCK($m$) | ただし、$m$ と $n$ は値が等しい |
| CYCLIC($n$) | $\equiv$ | CYCLIC($m$) | ただし、$m$ と $n$ は値が等しい |
| CYCLIC | $\equiv$ | CYCLIC(1) | |
| GEN_BLOCK($v$) | $\equiv$ | GEN_BLOCK($w$) | ただし、$v$ と $w$ の対応する要素の値は等しい |
| INDIRECT($v$) | $\equiv$ | INDIRECT($w$) | ただし、$v$ と $w$ の対応する要素の値は等しい |

2. これら以外で、字面上異なる *dist-format* の指定は同値でない。

これは、数学的な意味で普通に用いられる同値関係である。

次に、SHADOW 属性 (文法は 8.12 節 参照) についての同値の概念を定義する。

1. *shadow-spec* の式 $w_1$ と $w_2$ は、同じ値をもつ場合、またその場合に限って同値である。
2. *shadow-spec* $w$ は、*shadow-spec* $w$:$w$ と同値である。

3. *shadow-spec* $l_1$:$h_1$ と *shadow-spec* $l_2$:$h_2$ は、$l_1$ が $l_2$ と同値であり、かつ $h_1$ が $h_2$ と同値である場合、またその場合に限って同値である。
4. これら以外で、字面上で異なる *shadow-spec* 同士は同値でない。

そして、二つの SHADOW 属性は、それらの *shadow-spec-list* が要素ごとに互いに同値である場合、またその場合に限って同値であると言える。

今度は、マッピングの半順序を定義する。ここで、S (“special”) と G (“general”) をデータ実体とする。

以下のいずれかを満たす場合、またその場合に限って、S のマッピングは G のマッピングの特殊化であると言う。

1. G は INHERIT 属性をもつ。
2. S は INHERIT 属性をもたず、かつ以下のすべての制約が満たされる:
   (a) S は名前付きデータ実体または構造体成分である。
   (b) S と G は、最終整列先の形状が一致する。
   (c) S と G の対応する次元は、それぞれの最終整列先の対応する次元にマップされる。また、S と G の対応する要素は、それぞれの最終整列先の対応する要素に整列される。
   (d) 以下のいずれかである。
      i. S と G の最終整列先は明示的に分散されていない。
      ii. S と G の最終整列先は明示的に分散されている。この場合、G の最終整列先に対する分散の指示文は、以下の条件の中のいずれか一つを満たしていなければならない。
         A. *dist-onto-clause* がない。
         B. *dist-onto-clause* が “ONTO *” である。
         C. *dist-onto-clause* で指定されているプロセッサ構成の形状が、S の最終整列先に対する分散の指示文で明示されているものと一致する。

      また、以下の条件についても、いずれか一つを満たしていなければならない。
         A. *dist-format-clause* がない。
         B. *dist-format-clause* が “*” である。
         C. すべての *dist-format* が、S の最終整列先に対する分散の指示文で明示されている *dist-format-clause* の中の対応する位置の *dist-format* と、それぞれ同値 (上記で定義) である。
   (e) S と G は、どちらも SHADOW 属性をもっていないか、同値の SHADOW 属性をもっている。

## 8.14 明示的引用仕様を省略できる条件

4.6 節に示した条件は、DYNAMIC 属性があってもよいことを考慮して以下のように拡張される (追加部分は下線を付けて示す)。 ⇑

以下のすべての条件が満たされる場合を除いて、明示的引用仕様が必要である。

1. Fortran 規格で必要としていない。
2. 転写的に分散された仮引数はない。また、INHERIT 属性をもつ仮引数はない。
3. DYNAMIC 属性をもつ仮引数はない。
4. 対応する実引数と仮引数は、以下のどちらかである。

(a) どちらも暗黙にマップされている。
(b) どちらも明示的にマップされていて、
  i. 実引数のマッピングは、仮引数のマッピングの特殊化であり、
  ii. 実引数と仮引数の最終整列先がどちらも明示的に分散されている場合には、それぞれの *dist-onto-clause* には同じ形状のプロセッサ構成が指定されている。

5. 対応する実引数と仮引数は、
(a) どちらも順序的であるか、
(b) どちらも非順序的である。

## 8.15 手続の特性

SHADOW 属性と DYNAMIC 属性は、仮引数と手続の戻り値の HPF 特性になる。厳密には、4.7 節での定義は以下のように書き換えられる (追加部分は下線を付けて示す)。

- プロセッサ構成は一つの HPF 特性をもつ。それは形状である。
- テンプレートは以下の高々三つの HPF 特性をもつ。
  1. 形状。
  2. 分散。だだし、それが明示的に宣言されている場合。
  3. 分散先のプロセッサ構成の HPF 特性 (すなわち形状)。だだし、それが明示的に宣言されている場合。
- 仮データ実体は以下の HPF 特性をもつ。
  1. 整列、および整列先のすべての HPF 特性。ただし、それが明示的に宣言されている場合。
  2. 分散、および分散先のプロセッサ構成の HPF 特性 (すなわち形状)。ただし、それが明示的に宣言されている場合。
  3. SHADOW 属性。ただし、それが明示的に宣言されている場合。
  4. DYNAMIC 属性。ただし、それが明示的に宣言されている場合。
- 関数結果は仮データ実体と同じ HPF 特性をもつ。すなわち、以下の HPF 特性をもつ。
  1. 整列、および整列先のすべての HPF 特性。ただし、それが明示的に宣言されている場合。
  2. 分散、および分散先のプロセッサ構成の HPF 特性 (すなわち形状)。ただし、それが明示的に宣言されている場合。
  3. SHADOW 属性。ただし、それが明示的に宣言されている場合。
  4. DYNAMIC 属性。ただし、それが明示的に宣言されている場合。