第11章　HPF外来機能に関する公認拡張

この章の内容は、第6章に基づくものであり、LOCALやSERIAL (第11.1節) といった様々な並列処理モデル、及びHPF (第11.3節)、 C (第11.4節)、 Fortran (第11.5節)、Fortran 77 (第11.6節) といった様々な言語の、HPFに対する特別な引用仕様が定義されている。外来モデルの手続から利用可能なライブラリ手続は、第11.7節で定義されている。これらの引用仕様の定義は、Fortran言語による。C言語から引用できる同様のライブラリを定義して、実装してもよい。HPFの外来機能は、相互引用の機能を一般化するために定義されたものである。これらを、ここでは扱われていない引用仕様に対するモデルとして利用してもよい。いくつかの外来引用仕様が、付録E以降でさらに追加される。

HPF1.1では、 HPF、HPF_LOCAL、 及び HPF_SERIAL という特別な外来種別が定義されていた。また、F90_LOCAL に関する議論も少し述べられていた。本規格による、より一般的な仕様においても、互換性を保つために、HPF というキーワードは引続き採用されている。第 6 章で定義されているように、HPF のモデルはグローバルである。HPF_LOCAL は、LANGUAGE='HPF',MODEL='LOCAL' と同じであり、HPF_SERIAL は、LANGUAGE='HPF',MODEL='SERIAL' と同じである。F90_LOCAL 外来種別は、本規格では、LANGUAGE='FORTRAN',MODEL='SERIAL' 外来種別となった。

手続の呼出し側から見ると、「グローバル」 HPFプログラムからの 外来手続の引用は、通常の手続引用と同じ意味をもっている。呼ばれ側手続は、通常の手続とは異なった意味をもち得る。以下の節では、様々な外来種別のオプションをもつ呼ばれ側手続を、コーディングする際の規約を述べる。ある特定の HPF コンパイラでサポートされる 外来種別のオプションは、処理系依存である。それらは、この章で述べられているものに限定されない。さらに、外来副プログラムは、必ずしもHPFコンパイラで翻訳する必要はない。また、実行する外来プログラム単位のコードが、その言語用に定義された規約に従っているなら、LANGUAGE句に指定された言語のコンパイラである必要もない。これらの引用仕様は、移植性を高め、相互引用を促進するために定義されたものだが、実装者やプログラマは、その他のモデルや言語の組合せを定義してもよい。

11.1　様々な外来モデル : ローカルモデルとシリアルモデル

グローバルHPFプログラムでは、プロセッサの集合は、単一論理スレッドのプログラム制御上で、時には同期を取りながら、協調して動作している、と考えられる。第6章では、グローバルHPFから引用できる2つのモデルが定義された。その一つは、LOCALモデルである。このモデルは、単一プロセッサ「ノード」用コードであり、全ての活動プロセッサが実行に参加するが、それらは、物理プロセッサにマップされているデータだけを直接参照することができる。もう一つは、 SERIALモデルである。このモデルでは、単一のプロセッサだけが実行に参加する。必要な全てのデータは、シリアル副プログラムが起動される前に、呼出し側

によって、実行プロセッサに集められる。それぞれの利用例を挙げると、 LOCAL モデルは、データ転送のためのメッセージ通信が明示されているような低水準コードによるプログラムの場合に有効である。一方、SERIAL モデルは、システムライブラリや特別な I/O 手続を引用するために必要となる可能性がある。

LOCAL モデルも SERIAL モデルも、グローバル HPF プログラムから引用することができるが、一般的には、これらのモデルの引用を混合してはならない。ローカル手続やシリアル手続から グローバル HPF 手続を引用してはならない。また、ローカル手続から シリアル手続を引用してはならない。第 6.3.1 項では、様々なモデル同士の関係がどの様なものであるかを詳細に述べている。

グローバル HPF から引用される全てのローカル副プログラムとシリアル副プログラムには、以下のような制約が追加される。

- グローバル HPF から直接引用される副プログラムは、再帰的ではあってはならない。
- グローバル HPF から直接引用される副プログラムは、選択戻りを使用してはならない。

ローカル副プログラムやシリアル副プログラム内の I/O 文の動作は処理系依存である。

11.1.1 ローカルモデル

外来手続を、ローカル手続であると指定すれば、その手続が、各プロセッサ上で実行するためのコード (SPMD コード) であることを明示できる。この項では、呼出し側と EXTRINSIC(MODEL='LOCAL') 呼ばれ側手続との間の規約を述べる。外来手続 (HPF プログラムから引用される概念的な単一の手続要素) と個々のローカル手続 (各ノード上で実行される) とを、混同しないようにすることは重要である。外来手続の「起動」により、各プロセッサ上で独立に、ローカル手続が起動される。外来手続の「実行」は、各実行プロセッサ上でのローカル手続の並列実行から構成される。各ローカル手続は、RETURN 文の実行によっていつでも終了することができる。しかし、外来手続は、全てのローカル手続が終了して初めて、全体として終了する。つまり、プロセッサは、グローバル HPF 呼出し側に戻る前に、同期をとる。

ローカル手続から グローバルな呼出し側に戻る時を除いては、ローカルに実行しているプロセッサが暗黙的に同期をとる必要は全くない。ローカル手続内で宣言された変数は、各プロセッサの非公開なローカル記憶領域に割り付けられる。プロセッサ外のデータを参照するためには、ローカルモデルコードの実行前に、データをプロセッサにコピーするための通信を発行するか、若しくは独立に実行中のローカル手続のコピー間で、明示的に通信処理を行う必要がある。個々の実装では、例えば、メッセージ通信ライブラリや、共有メモリ機構など、処理系依存の通信手段を提供してもよい。そのような通信手段は、この規格の範囲外である。しかし、通信の機能がなくても、制御構造の独立性のみを要求するような多くの有益な移植性の高いアルゴリズムで、ローカル手続を利用できる、ということに注目されたい。

LOCAL モデルで仮定されているのは以下のことだけである。すなわち、非順序的な配列の次元が、直方体プロセッサ格子の次元に独立に対応づけられ、さらに配列の各次元は、プロセッサの高々1 つの次元に対応づけられ (「歪んだ」マッピングはない)、かつ配列の 2 つの次元がプロセッサの同じ次元に対応づけられることはない、という制約である。

この制約は、各物理プロセッサが所有する配列要素部分集合のローカルな構成を、直方体にできるための十分条件である。(もちろん、ある要素のローカルな添字が与えられた時に、そのグローバルな添字を計算することや、その逆変換は、複雑な計算であり得る。しかし、それは可能である。) サイクリック分散の場合、配列の一部分が複数個、ローカルなプロセッサにマップされ得る。

この節で述べられた規約に従わない外来種別を実装する場合、そのモデル名やキーワードは、LOCAL という語を含まないよう推奨する。

11.1.1.1 ローカル副プログラムの呼出し規約

スカラ仮引数、スカラ関数結果、及び順序的な配列は、各物理プロセッサ上に複製される。このようなマッピングを、引用仕様内で明示しても良い。(順序的な配列の場合は除く。これらは、明示的にマップしてはならない。) しかし、複製以外のマッピングを指定することは、HPF 規格合致でない。仮引数は、明示的にマップされた構造型であったり、明示的にマップされた構造体成分をもっていてはならない。

非外来副プログラムを引用する場合と同様に、実引数はどの様にマップされていてもよい。もし必要なら、外来手続引用の前後で自動的に、HPF 処理系が、実引数を正しくマップされた一時領域にコピーする。

ローカル副プログラムを引用するための規約は、 GLOBAL モデルと LOCAL モデル間の引用仕様にだけ適用されることに注意されたい。この規約は、LOCAL モデル手続からさらに呼出される副プログラムに対しては適用されない。

11.1.1.2 ローカル手続の呼出し手順

外来ローカルプログラムの実行は、以下に述べられた処理が、各プロセッサ上でのローカル手続の起動に先立って行われる「かの如くに」実装されなければならない。(関連する規約のリストと「かの如くに」の意味は、 第 6.3.2 項を参照のこと。) ここで要求されている任意の処理は、プログラマの責任ではなく、呼出し側プログラム単位のコンパイラによって行われるものであり、それらはローカル手続には影響を与えない。

1. プロセッサは同期をとる。言い換えれば、論理的に呼出しより前の全ての動作が完了する。
2. 各実引数は、外来手続に対して宣言された引用仕様内の (明示的又は暗黙的な) 指示文に従って、必要なら再マップされる。この場合、引用仕様内の HPF マッピング指示文は、強制力がある。つまり、コンパイラは、ローカル外来手続を起動する際に、これらの指示文に従わなければならない。(この規則が設けられた理由は、データのマッピングが、ローカル手続内であからさまに参照できるからである。) 順序的な配列やスカラ仮引数に対応する実引数は、全てのプロセッサ上に、(例えば、ブロードキャストによって) 複製される。明示的にマップされた構造体成分をもつ構造型や明示的にマップされた構造型のスカラを、グローバル HPF から引用されるローカル外来手続の引数に指定してはならない。
3. 呼ばれ側手続から参照可能な変数が複製されている場合、その全てのコピーは、原始プログラムを逐次的に実行した場合の、現在の正しい値をもつように、呼出し前に更新される。

これらが実行された後、ローカル手続が各プロセッサ上で起動される。ローカル手続起動中に利用できる情報は、以下の第 11.1.1.3 項で述べられる。

外来ローカル手続の実行が完了して、呼出し側に制御が戻った時、以下の処理が、既に行われた「かの如くに」呼出し側の実行が再開されなければならない。

1. 全てのプロセッサは戻り時に同期をとる。言い換えれば、呼出し側の実行再開前に、ローカル手続の全てのコピーの実行が完了している「かの如くに」、グローバルプログラム単位内の演算は実行される。
2. 必要ならば、引数 (と外来関数の結果) のマッピングが、呼出し前のマッピングに戻される。

【実装者への助言】 ローカル副プログラムから戻る前に、複製された変数が副プログラム内で同じ値に更新されたことを確認するように実装してもよい。しかし、そのように実装しなければならない訳ではないし、奨励すらされないだろう。これは、ローカル副プログラム側の責任であり、呼出し側の確認はどのようなものでも、スピードと、例えばデバッグのしやすさとのトレードオフとなる。【以上】

11.1.1.3 ローカル手続内で利用可能な情報

呼出し側が (グローバル) 外来手続引用仕様に対して渡す、各「グローバル引数」に対応して、各プロセッサ上で起動されるローカル手続には、「ローカル引数」が渡される。各グローバル引数は、HPF の配列又はスカラである。それに対応するローカル引数は、グローバル配列内のローカルに割り当てられている部分か、スカラ引数のローカルコピーか、全プロセッサに複製された順序的な配列である。HPF の配列が複製されている場合、各ローカル手続は、実引数全体のコピーを受け取る、ということに注意されたい。HPF 呼出し側が渡す実配列引数を、「グローバル配列」と呼ぶ。グローバル配列の部分格子であり、(そのプロセッサに割り当てられている、という理由で) ローカル手続の 1 つのコピーに渡される引数を、「ローカル配列」と呼ぶ。

外来手続が関数の場合、ローカル手続もまた関数であり、その返値はスカラでなければならない。各ローカル関数の返値は、同一でなければならない。

グローバル HPF プログラムが、実配列引数 `X` をもつローカル副プログラム `A` を呼出し、`A` が、配列 `X` の一部を仮引数 `P` として受け取った場合、`A` は、`P` 又は `P` の一部を実引数として、別のローカル副プログラム `B` を呼び出すことができる。

各ローカル関数内で、各ローカル引数に関して、対応するグローバル引数のマッピングを知り、グローバルな添字からローカルな添字への変換及びその逆変換ができるように、実行時のインタフェースにより、十分な情報が提供されなければならない。このような情報を提供する特別な一連の手続が、第 11.7.1 項に、HPF ローカルライブラリとして述べられている。

このような情報をローカルプログラム単位から利用できるようにする手段は、実装と、ローカルプログラム単位に利用されるプログラミング言語に依存する。

11.1.2 シリアルモデル

この節では、副プログラムの分身が (一つ以上かもしれないプロセッサのうち) 一つのプロセッサだけで実行されるコードを書くための規約を定義する。プログラム単位の外来モデルが `SERIAL` の場合、HPF コンパイラは、その副プログラムが、単一プロセッサ上で実行され

るようにコーディングされている、と仮定しなければならない。グローバル HPF 呼出し側からみると、SERIAL 手続は、同様にコーディングされた HPF 手続と同じように振舞う。それらの違いは、(性能のような) 処理系依存の動作だけである。

今のところ、どのプロセッサが HPF_SERIAL 手続を実行するのかを指定する方法はない。

11.1.2.1 シリアル手続呼出し手順

グローバル HPF から SERIAL 手続を起動する前に、以下の処理が行われた「かの如くに」、プログラムは動作する。

1. プロセッサは同期をとる。論理的に呼出しより前の全ての動作が完了する。
2. 全ての実引数は、実際に SERIAL 手続を実行するプロセッサに再マップされる。SERIAL 手続は、各引数を順序的な引数とみなす。

SERIAL 手続は、一つのプロセッサ上で実行されている「かの如くに」動作する。グローバル HPF から起動された SERIAL 手続の分身の実行完了後、以下の処理が行われた「かの如くに」、プログラムは動作する。

1. 全てのプロセッサが、呼出しの後に同期をとる。
2. 実引数のマッピングが、呼出し前のマッピングに戻される。

11.2 外来言語との結合仕様

前節では、HPF の外来機能に対して定義されている実行モデルに関して、規約と考察を述べた。HPF の外来引用仕様は、引用された副プログラムに使用されている言語の結合仕様を、コンパイラに伝える目的でも利用される。ここでは、4 つの言語との結合仕様が定義されている。それらは、HPF、 Fortran、 F77、及び C である。実装者は、他の引用仕様をサポートしてもよいし、ユーザーが引用仕様をカスタマイズできるようにしてもよい。この節では、特別な外来種別である HPF_LOCAL と HPF_SERIAL を定義する。

言語引用仕様の重要な機能は、明示的な外来引用仕様における、仮引数の属性の拡張である。これにより、プログラマは、外来種別の異なる手続間で、引数を渡す際の全ての要素を制御できる。この機構は、 C と Fortran77 に対する引用仕様で多用されるが、他の言語の引用仕様にも適用可能なので、ここでより一般的な文脈において定義されている。

11.2.1 引数の制御

外来種別の手続の仮引数に対する特別なデータ属性として、MAP_TO、LAYOUT、及び PASS_BY がある。これらは、適切な外来種別の手続に対する明示的引用仕様内で、仮引数の型宣言文中にのみ指定できる。特に、これら全ての属性は、LANGUAGE ='C' (第 11.4 節) という種別の外来引用仕様に対して定義されており、後の 2 つの属性は、 LANGUAGE ='F77' (第 11.6 節) という種別の外来引用仕様に対して定義されている。

この言語拡張の目的は、異なるプログラミング言語で記述された手続間で、引数の受け渡しを可能にするための EXTRINSIC 引用仕様を、より柔軟な機構にすることである。これら 3 つの属性は、様々な引数渡しの規約やオプションを考慮したものであり、またほとんど等価なデータ型表現や配列の配置方法の間で、引数渡しを行う便利な方法を提供している。しか

し、これらの機構は、様々な言語のデータ型や実装間で完全な等価性を提供する、という問題を解決するものではない、ということに注意されたい。

MAP_TO 属性は、本質的には同じだが、必ずしも同じ値の範囲や等価なマシン表現をもたない (異なる言語の) データ型間における、引数渡しの標準的、汎用的な機構を提供するために設計された。実引数の値が、新たなデータ型で適切に表現されるかどうか、また、最初に新たな言語へ変換され、外来手続内で演算が行われ、そして元の言語へ暗黙的に逆変換される、という一連の操作によって、その値が変化するかどうか、ということに関しては、(言語の実装者ではなく) プログラマに責任がある。LAYOUT 属性は、異なる言語の手続間で、引数として配列を渡す時に、一つ又はそれ以上のプロセッサ内での配列要素順序を変える必要がある場合に使用される。最後の PASS_BY 属性は、言語の実装毎の差異を考慮して、引数渡しの機構をより詳細に制御できるように設計されており、それにより、HPF 以外の言語で提供されている引数渡しのオプションを選択したり、HPF のマッピング情報を、データの値と一緒に HPF 以外の手続へ渡したい時に、処理系依存のデータ構造を渡すことができる。

これらの属性は、 Fortran 規格の *attr-spec* に関する構文規則 R503 を拡張することにより定義されている。*type-declaration-stmt* の規則 R501 は、拡張された *attr-spec* を参照することを除き変更されない。以下の最初の 2 つの制約は、Fortran 規格からの変更はないが、確認のため改めて述べる。というのは、それらは全ての属性に適用される一般的なものであるからだ。Fortran 規格の規則 R501 から R506 までの残りの制約は、規格中で既に定義されている属性に特有のものであり、ここでもそれは仮定されているが、繰り返さない。このような Fortran 構文に対する変更は、言語間の相互引用性が Fortran 社会全体の興味の対象であるため、次改訂で言語規格に採用されることを期待して行われた。

H1101 *type-declaration-stmt-extended* **is** *type-spec* [[, *attr-spec-extended*] ... ::] *entity-decl-list*

H1102 *attr-spec-extended*
is PARAMETER
or *access-spec*
or ALLOCATABLE
or DIMENSION (*array-spec*)
or EXTERNAL
or INTENT (*intent-spec*)
or INTRINSIC
or OPTIONAL
or POINTER
or SAVE
or TARGET
or MAP_TO (*map-to-spec*)
or LAYOUT (*layout-spec*)
or PASS_BY (*pass-by-spec*)

H1103 *map-to-spec* **is** *scalar-char-initialization-expr*

H1104 *layout-spec* **is** *scalar-char-initialization-expr*

H1105 *pass-by-spec* **is** *scalar-char-initialization-expr*

制約: 同じ *attr-spec-extended* を、一つの *type-declaration-stmt* 中で 2 回以上指定してはならない。

制約: 一つのデータ要素には、一つの有効域内で、どの属性も 2 回以上明示的に指定してはならない。

制約: MAP_TO 属性、LAYOUT 属性、及び PASS_BY 属性は、これらの属性が明示的に定義されている外来種別の有効域内で、仮引数に対してだけ指定できる。

F95:12.2.1.1 で定義され、第 8.15 節で拡張された**仮データ実体の特性**の定義は、仮データ実体の MAP_TO 属性、 LAYOUT 属性、及び PASS_BY 属性を含むように、さらに拡張される。

MAP_TO 属性における *map-to-spec* の値には、名前付き実引数のデータ型を、外来手続内の仮引数のデータ型にどのように変換するかを指定する。その例は、第 11.4.2.1 項で示される。

MAP_TO 属性を指定できる外来種別に対しては、*map-to-spec* として許される値の集合が定義される。型と変換先の型との間で、許される値の範囲が異なり、実引数又は実引数の部分実体の値が、変換先の型で許される範囲にない場合、それと結合する仮引数又はその部分実体の値は不定となる。逆に、仮引数や仮引数の部分実体の値が、それと結合する実引数の値として許される範囲内になく、実引数が変数の場合、結合する実引数又は実引数の部分実体の値は、不定となる。

仮引数の型と仮引数の変換可能な型との間で、精度、表現方式、許される値の範囲、又は記憶列の不一致がある場合、外来手続の引用中、仮引数の表現が、変換可能な型の実体に対する呼ばれ側手続の想定に適合することを、コンパイラは保証する。手続からの戻り時には、コンパイラは、変数である実引数の値が、宣言された型と種別に戻ることを保証する。

【利用者への助言】 この規則は、相互引用機能の移植性を保証するために定められた。しかし、大きな実体の場合、データ型に用いられる表現方式と変換可能な型に用いられる表現方式が一致していない場合、大きなオーバヘッドが生じる可能性があることに注意すべきである。【以上】

LAYOUT 属性において、ある外来引用仕様に対して許される *layout-spec* の値により、名前付き実引数のデータ配置から、外来手続の仮引数のデータ配置への変換方法を指定できる。その一例が、第 11.6.3 項で示される。仮配列引数に対する LAYOUT 属性が省略されている場合、データ配置は、他の暗黙のデータ配置が、与えられた外来種別に対して定義されていない限り、同じモデルの HPF 手続へ渡された場合と同じになる。

PASS_BY 属性において、ある外来引用仕様に対して許される *pass-by-spec* の値により、名前付き実引数を、外来手続内の仮引数と結合する際に使用される機構を選択できる。その例が、第 11.6.3 項 と第 11.4.2.1 項で示される。PASS_BY 属性が省略された場合、引数結合の機構は、外来言語処理系に関するコンパイラの知識に基づくものとなり、それは処理系依存である。

11.3 HPF 言語との結合仕様

プログラム単位は、既定値では、HPF 言語で記述されているとみなされる。HPF 言語の場合、明示的引用仕様内で、MAP_TO 属性、 LAYOUT 属性、又は PASS_BY 属性を指定する必要はない。副プログラムを結合するのに必要な全ての情報は、Fortran 規格の明示的引用仕様から得ることができる。

Fortran 規格の第 14.7 節で述べられている規則は、Fortran 言語に基づく SERIAL 及び LOCAL プログラム単位の有効域で宣言された変数に適用される。特に、ある変数の定義状態、結合状態、割り付け状態が、Fortran 副プログラムの RETURN 文や END 文の実行時に定義されている場合、SERIAL 副プログラムや LOCAL 副プログラム内のそのような変数もまた、RETURN 文や END 文の実行時に定義される。

異なるモデルの外来副プログラムで行われる I/O や、グローバル HPF 内で使用されるファイル名及び装置番号と、ローカル又はシリアル副プログラムコード内で使用されるそれらとの対応は、処理系依存である。

11.3.1 HPF_LOCAL に関する特記事項

ローカルなプロセッサ毎の手続を作成する際、どのような HPF の機能を利用できるかに関して考察する。

ローカルプログラム単位は、REDISTRIBUTE 指示文と REALIGN 指示文を除く、全ての HPF 構文を使用できる。ただし、DISTRIBUTE 指示文、 ALIGN 指示文、及び INHERIT 指示文は、仮引数にだけ適用できる。すなわち、全ての *alignee* と *distributee* は、仮引数でなければならず、全ての *align-target* は、テンプレートか仮引数でなければならない。ローカル HPF プログラム単位内のマッピング指示文は、それらが グローバル HPF コード内で指定されたかのように、グローバル配列に対して適用されていて、その一部が各プロセッサに渡されている、というグローバルな意味をもっているものとみなされる。(そのようなマッピング指示文が主として使用されるのは、グローバル HPF モジュールから引用される HPF_LOCAL モジュール内である。)

HPF_ALIGNMENT、 HPF_TEMPLATE、及び HPF_DISTRIBUTION マッピング問合せライブラリサブルーチンは、非順序的なローカル配列に適用することができる。それらの結果は、単一ノードに全ての要素が割り当てられているグローバル配列の場合と同じである。

第 6.3.1 項で述べたように、ローカル HPF プログラム単位は、実引数を通して参照可能なデータ以外のグローバル HPF 内のデータを、直接又は間接に参照してはならない。特に、ローカル HPF プログラム単位は、 グローバル HPF 内の COMMON ブロックを参照できない。すなわち、ローカル HPF プログラム単位に現れる COMMON ブロックは、グローバル HPF 手続の COMMON ブロックとは異なる。同じ実行プログラムにおいて、ローカル HPF プログラム単位の COMMON ブロックと HPF プログラム単位の COMMON ブロックに、同じ名前を指定してはならない。

ローカル副プログラム内の COMMON ブロックは、ローカル副プログラム内の局所変数と同様に、各プロセッサにローカルな記憶領域に割り付けられたローカルデータである。この記憶領域は、ローカルにしか参照できず、一般に、各プロセッサ上で異なる値をもつ。さらに、ローカルな COMMON ブロックの大きさは、各プロセッサ毎に異なる可能性もある。

Fortran 規格によれば、有効域外の COMMON ブロックは、SAVE 属性をもたない限り保存されない。これは、ローカルな COMMON ブロックの場合も同様であり、ローカル副プログラムの呼出し間で値を保存したい場合には、SAVE 属性を指定しなければならない。

明示的にマップされた構造型スカラを、HPF_LOCAL 副プログラムの引数に指定してはならない。

仮引数の属性 (型、種別、次元数、省略可能性、授受特性) は、明示的引用仕様内の対応する仮引数の属性と一致していなければならない。EXTRINSIC('HPF','LOCAL') 手続の仮引数に、手続名を指定してはならない。

EXTRINSIC('HPF','LOCAL') 手続の仮引数に、POINTER 属性を指定してはならない。

EXTRINSIC('HPF','LOCAL') 手続の非順序的な仮配列引数は、引用仕様内で形状が明示してある場合でも、形状引継ぎ配列でなければならない。仮配列引数の形状は、実行プロセッサが 1 台でない限り、一般に、対応する実引数の形状とは異なる、ということに注意されたい。

仮引数に対する明示的なマッピング指示文を、ローカル手続内で指定してもよい。そのような指示文は、グローバル配列に対して適用されて、その各プロセッサ上にあるローカルな部分が、実引数又は関数結果として渡されている、とみなされる。そのような指示文が指定された場合、対応するマッピング指示文が、グローバル HPF 呼出し側から参照可能でなければならない。このためには、呼出し側に引用仕様宣言を記述するか、又は外来種別が HPF_LOCAL のモジュール内にローカル手続を記述し、ローカル手続を呼び出すグローバル HPF プログラム単位から、それを引用すればよい。

EXTRINSIC('HPF','LOCAL') 手続は、FORALL 構文の本体や INDEPENDENT ループの本体から、直接的にも間接的にも起動してはならない。ローカル手続は、複数の ENTRY をもっていてもよい。グローバル HPF 呼出し側には、HPF プログラムから起動できる各入口に対して、別々の外来引用仕様を記述しなければならない。

11.3.2 引数の結合

EXTRINSIC('HPF','LOCAL') 手続の仮引数がスカラの場合、ローカル手続の対応する仮引数は、同じ型及び型パラメタのスカラでなければならない。スカラは、組込み型、又は明示的にマップされていない構造型の場合だけ、グローバルプログラム単位から HPF_LOCAL 手続への引数に指定できる。外来手続が起動された時、ローカル手続にはスカラのローカルコピーが渡される。このコピーは、正当な HPF のスカラである。

EXTRINSIC('HPF','LOCAL') 手続の仮引数が配列の場合、ローカル手続内で宣言された仮引数は、同じ次元数、型、及び型パラメタの配列でなければならない。

外来引用仕様内の配列が順序的の場合、対応する実引数は、スカラ引数が渡される場合と同様に、全てのプロセッサ上に複製されて渡される。各ローカル仮引数は、実配列引数全体のコピーと結合する。外来引用仕様内の仮引数と、ローカル手続の宣言内の対応する仮引数は、同じ形状の形状明示配列であってよい。EXTRINSIC('HPF','LOCAL') 手続へ、複製されて渡される全ての順序的な仮引数は、INTENT(IN) 引数であるか、又は全プロセッサ上で、同じ値に更新されなければならない。

仮引数が非順序的な配列の場合、ローカル仮引数は、外来手続の起動時に、グローバル配列内のローカルに割り付けられた部分格子であるローカル配列と結合する。このローカル配列は、正当な HPF の配列である。

EXTRINSIC('HPF','LOCAL') 手続が関数の場合、ローカル手続は、HPF の外来関数と同じ型、型パラメタのスカラを返す関数である。ローカルに起動される各関数の返値は、同一でなければならない。

各物理プロセッサは、各 HPF 変数の、高々1 つのコピーをもつ。

次のような外来引用仕様を考えてみよう。

```
INTERFACE
  EXTRINSIC('HPF','LOCAL') FUNCTION MATZOH(X, Y) RESULT(Z)
    REAL, DIMENSION(:,:) :: X
    REAL, DIMENSION(:) :: Y
    REAL Z
    !HPF$ ALIGN WITH X(:,*) :: Y(:)
      ! この宣言は、size(Y) = size(X,1) であることを主張している。
    !HPF$ DISTRIBUTE X(BLOCK, CYCLIC)
  END FUNCTION
END INTERFACE
```

対応するローカル HPF 手続は、以下のように定義されているとする。

```
EXTRINSIC('HPF','LOCAL') FUNCTION MATZOH(XX, YY) RESULT(ZZ)
  REAL, DIMENSION(:,:) :: XX
  REAL, DIMENSION(5:) :: YY ! 下限が 5 の形状引継ぎ配列
  REAL ZZ
  NX1 = SIZE(XX, 1)
  LX1 = LBOUND(XX, 1)
  UX1 = UBOUND(XX, 1)
  NX2 = SIZE(XX, 2)
  LX2 = LBOUND(XX, 2)
  UX2 = UBOUND(XX, 2)
  NY  = SIZE(YY, 1)
  LY  = LBOUND(YY, 1)
  UY  = UBOUND(YY, 1)
  ...
END FUNCTION
```

2×2 のプロセッサ構成を用いた 4 プロセッサマシン上で (物理プロセッサ 1 つ当たり、1 つの抽象プロセッサが対応していると仮定する)、形状が 3×3 の (グローバル) 実配列引数 X、及び長さ 3 の実ベクトル引数 Y と共に、関数が起動されると仮定する。

すると、ローカルに起動された関数 MATZOH の各々は、以下のような実引数を受け取る。

Processor (1,1)	Processor (1,2)
X(1,1) X(1,3)	X(1,2)
X(2,1) X(2,3)	X(2,2)
Y(1)	Y(1)
Y(2)	Y(2)

Processor (2,1)	Processor (2,2)
X(3,1) X(3,3)	X(3,2)
Y(3)	Y(3)

各プロセッサが、 NX1、LX1、 UX1、 NX2、 LX2、 UX2、 NY、LY、及び UY に設定する値は、以下の通りである。

Processor (1,1)	Processor (1,2)
NX1 = 2 LX1 = 1 UX1 = 2	NX1 = 2 LX1 = 1 UX1 = 2
NX2 = 2 LX2 = 1 UX2 = 2	NX2 = 1 LX2 = 1 UX2 = 1
NY = 2 LY = 5 UY = 6	NY = 2 LY = 5 UY = 6

Processor (2,1)	Processor (2,2)
NX1 = 1 LX1 = 1 UX1 = 1	NX1 = 1 LX1 = 1 UX1 = 1
NX2 = 2 LX2 = 1 UX2 = 2	NX2 = 1 LX2 = 1 UX2 = 1
NY = 1 LY = 5 UY = 5	NY = 1 LY = 5 UY = 5

外来手続への実引数は、ポインタであってもよい。対応する仮引数には、POINTER 属性を指定できないので、仮引数は、グローバル HPF 内のポインタ指示先と結合することになる。ローカルなポインタを、グローバル HPF 内の指示先と結合させることはできない。したがって、実引数は、ポインタ成分を含む構造型であってはならない。

【仕様の根拠】 グローバルなポインタ変数は、少なくとも分散メモリマシン上では、ローカルなポインタ変数とは違う内部表現をもっていると考えられる。というのは、グローバルなアドレスの割り振りのために、付加的な情報をもつ必要があるからである。この制限は、将来取り除かれる可能性がある。【以上】

ALLOCATED や PRESENT のような問合せ組込み手続も、仕様通りの動作をすべきである。グローバル配列がローカル手続へ渡される時、受け取る配列要素の集合が空であるプロセッサもあり得る、ということに注意されたい。

11.3.3 HPF_SERIAL に関する特記事項

ここでは、HPF_SERIAL 副プログラムに適用される制約を述べる。

specification-directive、 *realign-directive*、又は *redistribute-directive* を、HPF_SERIAL 副プログラムやその引用仕様本体に指定してはならない。

【仕様の根拠】 HPF のマッピング指示文は、HPF_SERIAL 副プログラム内では意味をもたない。しかし、*independent-directive* を、HPF_SERIAL 副プログラム内で指定しても

よい、ということに注意されたい。というのは、 それにより、DO ループ、FORALL 文、又は FORALL 構文に関する、コンパイラにとって重要な情報が提供され得るからである。【以上】

HPF_SERIAL 手続の任意の仮データ実体と関数結果変数は、順序的だとみなされる。

HPF_SERIAL 副プログラム内では、HPF 又は HPF_LOCAL プログラム単位で宣言された共通ブロックと同じ名前の共通ブロックを宣言してはならない。さらに、HPF 又は HPF_LOCAL プログラム単位内に無名共通ブロックの宣言がある場合、HPF_SERIAL 副プログラム内では、無名共通ブロックを宣言してはならない。

グローバル HPF から引用される HPF_SERIAL 手続の仮引数や関数結果変数には、POINTER 属性を指定してはならない。グローバル HPF から引用される HPF_SERIAL 手続の仮引数や関数結果の部分実体にも、POINTER 属性を指定してはならない。

グローバル HPF から引用される HPF_SERIAL 手続の仮引数やその部分実体には、TARGET 属性を指定してはならない。

HPF_SERIAL 手続の仮手続引数は、HPF_SERIAL 手続でなければならない。

```
      PROGRAM MY_TEST
        INTERFACE
          EXTRINSIC('HPF','SERIAL') SUBROUTINE GRAPH_DISPLAY(DATA)
            INTEGER, INTENT(IN) :: DATA(:, :)
          END SUBROUTINE GRAPH_DISPLAY
        END INTERFACE

        INTEGER, PARAMETER :: X_SIZE = 1024, Y_SIZE = 1024

        INTEGER DATA_ARRAY(X_SIZE, Y_SIZE)
!HPF$   DISTRIBUTE DATA_ARRAY(BLOCK, BLOCK)

!  DATA_ARRAY を計算する
        ...
        CALL DISPLAY_DATA(DATA_ARRAY)
      END PROGRAM MY_TEST

! 画像表示用副プログラムの定義
! この手続は、なんらかの実装依存の方式で、
! DATA 中のデータのグラフをプロットする。

      EXTRINSIC('HPF','SERIAL') SUBROUTINE GRAPH_DISPLAY(DATA)
        INTEGER, INTENT(IN) :: DATA(:, :)
        INTEGER :: X_IDX, Y_IDX

        DO Y_IDX = LBOUND(DATA, 2), UBOUND(DATA, 2)
          DO X_IDX = LBOUND(DATA, 1), UBOUND(DATA, 1)
```

```
      ...
      END DO
    END DO
  END SUBROUTINE GRAPH_DISPLAY
```

11.4 C 言語との結合仕様

Fortran のユーザが直面する共通の問題は、他の言語、特に C や C プロトタイプで記述できる引用仕様をもった言語の手続を、呼び出す必要に迫られる、ということだろう。多くの Fortran の実装では、この問題を解決する方法が提供されているが、これらの方法には、ほとんどの場合移植性がない。

この節では、相互引用性に対する一般的な障害のほとんどを解消する一方で、移植性も維持できるような、C 言語で定義された手続に対する引用仕様の指定方法を述べる。

11.4.1 C 言語で定義された手続に対する引用仕様

手続が C 言語で定義されている、ということを、ユーザが指定したい場合には、第 6 章で規定されているように、`LANGUAGE = 'C'` という *extrinsic-spec-arg* 、又は `C` という *extrinsic-kind-keyword* を指定すればよい。

`EXTRINSIC (LANGUAGE = 'C')` が指定された C 言語の副プログラムに対しては、*attr-spec-extended*(H1102) の構文に関する制約が、以下のように拡張される。

制約: `LANGUAGE = 'C'` 関数の結果は、整数型、実数型、又は倍精度実数型のスカラでなければならない。

制約: `LANGUAGE = 'C'` 手続の仮引数は、形状引継ぎ配列であってはならない。また、`LANGUAGE = 'C'` 手続の仮引数に、`POINTER` 属性や `TARGET` 属性を指定したり、その部分実体に `POINTER` 属性を指定してはならない。

制約: `LANGUAGE = 'C'` 手続の仮引数の上下限は、定数宣言式でない宣言式であってはならない。また、仮引数の文字長パラメタは、定数宣言式でない宣言式であってはならない。

制約: `LANGUAGE = 'C'` サブルーチン の *dummy-arg-list* 中の *dummy-arg* として、*や仮手続を指定してはならない。

`EXTERNAL_NAME` 指定子中の *scalar-char-initialization-expr* の値により、C 言語における手続の名前が指定される。この値は、*function-stmt* や *subroutine-stmt* で宣言された手続名と同じである必要はない。`EXTERNAL_NAME` が省略された場合、手続名を小文字で記述した値が指定されているものとみなされる。

> 【利用者への助言】 `EXTERNAL_NAME` 指定子には、必ずしも、リンカが手続を認識する際の名前を指定する訳ではない、ということに注意されたい。そこに指定するのは、C 言語のプログラムから引用された場合に、手続が認識される名前であり、HPF コンパイラが、C コンパイラと同様の変換を行わなければならない名前である。

ユーザが、EXTERNAL_NAME 指定子に指定する名前は、例えば、最初の文字が下線の名前、大文字と小文字を区別しなければならない名前、といった HPF コンパイラに対しては許されない名前でもよい。

【以上】

LANGUAGE = 'C'という *extrinsic-spec-arg* により、コンパイラは、手続が C で定義されていることを認識できるので、C コンパイラが要求する方法による手続の起動を保証するための、適切な手順を踏むことができる。

【実装者への助言】 ベンダは、システムで利用可能な様々な C コンパイラが、様々な手続呼出し規約やデータ型の大きさを提供している場合、2 つ以上の C コンパイラをサポートしなければならない、と考えるかもしれない。ベンダは、例えば、LANGUAGE=指定子に、GNU_C という値を提供してもよいし、コンパイラへのオプションによりサポートしてもよい。【以上】

11.4.2 C 言語に対するデータ型変換の仕様

MAP_TO 属性、LAYOUT 属性、及び PASS_BY 属性からなる外来手続の仮引数の特性は、Fortran プログラムから C で定義された手続を引用できるようにするための主要な機能である。また、これらの属性により、ユーザは、引用された手続の実引数と仮引数とを結合するために必要な変換を指定できる。MAP_TO 属性により、コンパイラが、HPF のデータを C 手続のどのデータ型に変換するのかを指定できる。また、PASS_BY 属性により、仮引数への C ポインタが渡される必要があるか否かを指定できる。また、LAYOUT 属性により、配列を、Fortran 言語の配列要素順序から C 言語のそれに変換する必要があるか否かを指定できる。

C 言語に対しては、*attr-spec-extended*、*map-to-spec*、 *layout-spec*、及び *pass-by-spec* (H1102–H1105) に関する制約が、以下のように拡張される。

制約: MAP_TO 属性は、LANGUAGE = 'C'手続に対する明示的引用仕様内の、全ての仮引数と関数結果変数に指定しなければならない。

制約: 仮引数に対して指定する *map-to-spec* は、仮引数の型と適合していなければならない。(以下で述べる適合規則を参照せよ)

制約: LAYOUT 属性は、仮引数が配列の場合だけ指定できる。

制約: LAYOUT 属性を、大きさ引継ぎ配列に指定してはならない。

コンパイラが大文字と小文字を区別できる場合でも、*map-to-spec*、*layout-spec*、及び *pass-by-spec* の値に対しては、その区別は無い。コンパイラは、*map-to-spec*、*layout-spec*、及び *pass-by-spec* の値の識別の際、それらの中の空白を全て無視する。

実装者は、ISO_C というモジュールを提供して、その中で、C_VOID_POINTER という構造型を定義しなければならない。C_VOID_POINTER 型の成分は、非公開である。

【利用者への助言】 C_VOID_POINTER 型により、プログラム内で、void *ポインタを使用できるようになる。しかし、ユーザには、プログラムの Fortran 部分でその様なポイ

ンタを操作するいかなる方法も与えられない。というのは、非公開成分をもつ実体に対して、その型を定義しているモジュール外で I/O を行うことはできないし、その様な構造型の成分や構造体構成子を参照することもできないからだ。【以上】

LANGUAGE = 'C'に対する *map-to-spec* の値として許されているものは、'INT'、'LONG'、'SHORT'、'SIGNED_CHAR'、'FLOAT'、'DOUBLE'、'LONG_DOUBLE'、'CHAR'、'CHAR_PTR'、'VOID_PTR'、又はこれらの値を任意にカンマで区切って並べて、括弧で括ったものである。これらと適合する HPF の型は、以下の表に示されている。

値の並びが括弧で括られた *map-to-spec* は、並び中の各値が構造型の対応する成分と適合する場合、構造型仮引数と適合する。

PASS_BY 属性が指定された場合、*pass-by-spec* として許される値は、'VAL'、'*'、又は'**'である。PASS_BY 属性が省略された場合、PASS_BY ('VAL') が仮定される。*pass-by-spec* として、VAL という値が指定された場合、仮引数には、 INTENT(OUT) 属性も INTENT(INOUT) 属性も指定してはならない。*pass-by-spec* として、'*'又は'**'という値が指定された場合、結合する実引数は、変数でなければならない。

値が C である LANGUAGE=指定子が指定された手続の引用仕様本体内で、仮引数に対して指定する *map-to-spec* の値は、変換可能な型の少なくとも 1 つが、手続の C 言語による定義における、対応する仮引数の (C 言語の) データ型と同じでなければならない (あるいは、変換可能な型の 1 つと適合する型でなければならない)。手続の C 言語による定義における、関数の (C 言語の) データ型は、値が C である LANGUAGE=指定子が指定された関数の引用仕様本体内で、関数結果変数に対して指定された変換可能な型の 1 つでなければならない (あるいは、変換可能な型と等価な型でなければならない)。サブルーチンに、値が C である LANGUAGE=指定子が指定されている場合、手続の C 言語による定義には、void というデータ型を指定しなければならない。

スカラ仮引数が、組込み型、又は構造型 C_VOID_POINTER である場合の変換可能な型は、以下の表に示されている。

MAP_TO	**適合する Fortran の型**	各 PASS_BY に対応する **C 言語の型**		
		'VAL'	'*'	'**'
'INT'	INTEGER	int	int*	int**
'LONG'	INTEGER	long	long*	long**
'SHORT'	INTEGER	short	short*	short**
'SIGNED_CHAR'	INTEGER	signed char	signed char*	signed char**
'FLOAT'	REAL	float	float*	float**
'DOUBLE'	REAL	double	double*	double**
'LONG_DOUBLE'	REAL	double	double*	double**
'CHAR'	CHARACTER(1)	char	char*	char**
'CHAR_PTR'	CHARACTER	char*	char**	char***
'VOID_PTR'	C_VOID_POINTER	void*	void**	void***

配列の変換可能な型は、各次元に対して、左角括弧 ([)、仮引数の対応次元の寸法、右角括弧 (]) が指定されている同じ型のスカラ変数の変換可能な型と等しい。LAYOUT 属性が省略されている場合、実引数の各次元に対して、仮引数の次元は、右から左へと対応する。一方、

LAYOUT 属性に C_ARRAY という値が指定されている場合、実引数の各次元に対して、仮引数の次元は、左から右へと対応する。

LANGUAGE = 'C'の *layout-spec* として許されている値は、C_ARRAY である。

構造型スカラ変数の変換可能な型は、対応する成分が、構造型成分の変換可能な型のうちの 1 つであるような構造型である。

仮引数の型と、仮引数の変換可能な型との間で、精度、表現方式、許される値の範囲、又は記憶列の不一致がある場合、コンパイラは、C という値の LANGUAGE=指定子が宣言された手続の引用中、変換可能な型の実体に対して C 言語処理系が想定している表現と、仮引数の表現が適合していることを保証しなければならない。手続から戻る時には、コンパイラは、変数である実引数の値が、宣言された型と種別に戻ることを保証しなければならない。

型と変換先の型との間で、許される値の範囲が異なっていて、実引数やその部分実体の値が、変換先の型で許される範囲にない場合、それと結合する仮引数やその部分実体の値は、不定となる。逆に、仮引数やその部分実体の値が、結合する実引数の値として許される範囲になく、実引数が変数である場合、実引数やその部分実体の値は、不定となる。

【利用者への助言】 これらの規則は、相互引用機能の移植性を保証するために定められた。しかし、大きな実体の場合、データ型に用いられる表現方式と、変換可能な型に用いられる表現方式が一致していない場合、大きなオーバヘッドが生じる可能性があることに注意すべきである。【以上】

【利用者への助言】 引用された手続内で値が更新されないにもかかわらず、実引数の値が変わってしまう場合があり得る。例えば、

```
PROGRAM P
  INTERFACE
    EXTRINSIC(LANGUAGE='C') SUBROUTINE C_SUB(R,I)
      REAL(KIND(1.0D0)), MAP_TO('FLOAT'), PASS_BY('*') :: R
      INTEGER, MAP_TO('INT'), PASS_BY('*') :: I
    END SUBROUTINE C_SUB
  END INTERFACE
  REAL(KIND(0.0D0)) RR

  RR = 1.0D0 + 1.0D-10
  I = 123456789
  PRINT *, RR
  CALL C_SUB(RR, I)
  PRINT *, RR
END PROGRAM P

void c_sub(float *r, int *i)
{
}
```

の結果は、*r の値が、c_sub 内で更新されないにもかかわらず、以下のようになる可能性がある。

```
1.00000000010000000
1.00000000000000000
```

同様に、I の値は、更新されないにもかかわらず、c_sub の引用後不定となる可能性がある。

基本実数型以外の任意の型の仮引数に対して、変換先の型に float を指定しないようにしたり、倍精度実数型以外の任意の型の仮引数に対して、 変換先の型に double を指定しないようにするのは、良い方法であるが、変換先の型に int、又は long を指定する必要がある実体に対して、適切な仮引数の型を選択することは、あまり簡単ではないかもしれない。【以上】

仮配列引数に対して、*layout-spec* が省略された場合、配列要素順序は、Fortran で規定されている順序と同じである。指定された *layout-spec* の値が C_ARRAY の場合、手続引用中の、その配列の配列要素順序は、転置されることになる。

11.4.2.1 データ型変換の例

与えられた Fortran プログラム内の引用仕様本体に対して、どのような種類の C 言語の手続定義が許されるのかを明らかにするために、幾つかの例が役に立つだろう。例えば、以下のような引用仕様本体は、

```
INTERFACE
  EXTRINSIC('C') SUBROUTINE C_SUB(I, R, DARR, STRUCT)
    INTEGER, MAP_TO('INT') :: I
    REAL, MAP_TO('FLOAT'), PASS_BY('*') :: R
    REAL(KIND(1.0D0)), MAP_TO('DOUBLE') :: DARR(10)
    TYPE DT
      SEQUENCE
      INTEGER :: I, J
    END TYPE DT
    TYPE(DT), MAP_TO('(INT, LONG)'), PASS_BY('*') :: STRUCT
  END SUBROUTINE C_SUB
END INTERFACE
```

以下のようなプロトタイプ宣言をもつ C 言語の手続に対応し得る。

```
void c_sub(int i, float r*, double darr[10], struct {int i, long j} *)
```

以下の例のような LAYOUT 属性が指定された場合、

```
PROGRAM P
  INTERFACE
    EXTRINSIC('C') SUBROUTINE C_SUB(A, B)
      INTEGER, MAP_TO('INT') :: A(2,2)
      INTEGER, MAP_TO('INT'), LAYOUT('C_ARRAY') :: B(2,2)
    END SUBROUTINE C_SUB
  END INTERFACE

  INTEGER :: AA(2,2), BB(2,2)
  CALL C_SUB(AA, BB)
END PROGRAM P

void c_sub(int a[2][2], b[2][2])
```

AA と a の要素、及び BB と b の要素の対応は、以下のようになる。

```
AA(1,1)   a[0][0]              BB(1,1)   b[0][0]
AA(2,1)   a[0][1]              BB(2,1)   b[1][0]
AA(1,2)   a[1][0]              BB(1,2)   b[0][1]
AA(2,2)   a[1][1]              BB(2,2)   b[1][1]
```

11.5 Fortran 言語との結合仕様

外来種別の宣言で指定された言語が Fortran の場合、結合仕様は基本的に HPF に対するものと同じである。というのは、HPF 規格は、Fortran 規格に基づいているからである。この場合に考慮すべき点が少しある。

- Fortran 構文のみを使用すべきである。非同期 I/O や HPF ライブラリのような機能はサポートされない場合がある。
- この目的で利用される Fortran 言語処理系は、HPF_LOCAL マッピング問合せライブラリ、及び HPF_LOCAL_LIBRARY モジュールの引用をサポートするよう拡張されていることが望ましい。
- Fortran 言語処理系で外来プログラム単位をコンパイルするつもりなら、それらは、HPF ソースコードと一緒のファイルではなく、別のファイルに記述すべきである。
- プログラマは、HPF 指示文は全て無視され得る、と考えるべきである。

11.6 Fortran 77 言語との結合仕様

Fortran 77 は、言語の引用仕様に関しては、現在も本質的に ANSI/ISO Fortran 規格の部分集合である。従って、HPF からの Fortran 手続呼出しに関連するほとんどの事項は、HPF から Fortran 77 外来手続を呼び出す際にも当てはまる。しかし、Fortran と Fortran 77

との間には、以下のように、2 つの重要な違いがあるため、HPF からの引用仕様である EXTRINSIC(LANGUAGE='F77') の規格が、特にローカルモデルの場合に複雑になっている。

- 引数の渡し方が通常は異なる。典型的な Fortran 77 の実装では、副プログラムへの引数をアドレス (参照渡し) により渡す。つまり、データ型、大きさ、マッピングといった、実引数に関するその他の情報は何も渡されない。これに対して、HPF の実装では、しばしば記述子経由で変数を渡し、副プログラム内でそれらの情報を参照できるようにしている。

- 配列要素の、記憶領域への配置方法に関して利用可能な情報が、Fortran 77 と HPF のプログラマの間では、かなり異なる。

  Fortran 77 の場合、配列の宣言により、配列要素と線形に並んだ記憶領域との対応が正確に規定される。HPF の場合、複数のプロセッサへの配列要素のマッピングを (例えば、DISTRIBUTE 指示文や ALIGN 指示文によって) 制御したり、(例えば、HPF_ALIGNMENT、HPF_DISTRIBUTION 、又は HPF_TEMPLATE によって) 問合せたりすることはできるが、任意のプロセッサにローカルな、連続した記憶領域内で、配列要素がどの様に配置されているかに関する情報は全く無い。Fortran 90 の場合ですら、例えば、形状引継ぎ配列の場合などは、Fortran 77 配列の記憶列結合と順序結合の規則に従っているとは限らない。

  実際のところ、HPF コンパイラが異なれば、ローカルなデータの配置方法は異なる可能性がある。例えば、定型パタンを 最適化する為に、境界要素を余分に割り付けているかもしれないし、全てのプロセッサ上で、部分格子の大きさを等しくするために、ダミー要素の埋め込みを行っているかもしれない。実際、HPF コンパイラは、特定の方法でローカルなデータを配置するように義務づけられては「いない」し、例えば、SMP の場合など、配列宣言イメージ通りの配置順序を選択する HPF コンパイラもあり得る。

11.6.1 F77_LOCAL に関する特記事項

EXTRINSIC(F77_LOCAL) 引用仕様は、 HPF_LOCAL 及び FORTRAN_LOCAL の外来引用仕様を、Fortran 77 プログラマの必要に合わせて拡張したものである。

この EXTRINSIC 種別においては、先に述べたような、外来副プログラム引用のための構文が用いられる。これは、より正確には、EXTRINSIC(LANGUAGE='F77',MODEL='LOCAL') 引用仕様と記述できる。グローバルプログラム単位とローカル手続との間の制御の移行に関する基本的な規約は、先に第 11.1 節で述べたものが適用される。

引数の渡し方とマッピング情報に関するオプションを付加することによって、そのような引用仕様を使用するさまざまな動機を与えると共に、これら 2 つの言語の、引数の渡し方とデータマッピングという点における違いが扱いやすくなる。このようなオプションは、LAYOUT 属性と PASS_BY 属性により提供される。

11.6.2 F77_LOCAL 手続への引数の渡し方

典型的な Fortran 77 の実装では、引数は参照渡しであり、通常、最初のデータ要素の記憶領域のベースアドレスが渡される。また、その様な引数の順序結合も仮定されている。このような事情により、マップされたデータ構造を、HPF から F77_LOCAL 手続に渡す暗黙の方法と

しては、ローカルな記憶領域において、引数が割り付けられている場所のベースアドレスを渡すのが最も実際的である。実引数と仮引数の順序結合を可能にするために、全てのプロセッサ上で、データ並びに対する並べ変えや圧縮、又はその両方が必要となる場合がある。この方法は、マップされたデータを EXTRINSIC(F77_LOCAL) 手続に渡す最も安全な方法なので、暗黙の方法であるべきだが、性能上のコストが極めて大きくなりがちである。

引数渡しの 2 つ目の選択肢は、グローバル HPF 手続からローカル F77 手続へ、仮引数の順序結合は保証せずに、マップされた配列データを、「そのまま」渡す方法であり、この場合、プロセッサにローカルな配列の要素列を圧縮したり並べ変えるためのデータの移動を、ローカルに行う必要がない。言い換えれば、他の HPF 手続に、同じ引数を渡す場合より多くのデータ移動は行わなくてよい。仮引数の順序結合に対する保証は、実引数のローカルな要素列を並べ変えたり圧縮したりしなくて済む性能上のメリットのために犠牲にされる。ローカル手続のプログラマには、グローバル HPF プログラムにおける処理系依存の配置順序が利用可能でなければならない。

3 つ目の選択肢は、 EXTRINSIC(F77_LOCAL) 手続呼出しでも、HPF_LOCAL 風のローカル手続のプログラミングを可能にするためのものであり、HPF の実装や Fortran 90 の形状引継ぎ配列で典型的に行われているように、記述子やハンドル経由で配列を渡す方法である。ローカル手続から、この仮引数の要素を直接参照してはならず、特別なユーティリティ手続に対してそれを渡して、ローカル又はグローバルなマッピング情報などを得ることだけができる。

以下の属性は、EXTRINSIC(F77_LOCAL) 手続へ引数を渡す際の、上記 3 つの選択肢をサポートするのに十分である。

- LAYOUT('F77_ARRAY') ローカルな連続した記憶領域内で、配列が FORTRAN 77 の順序結合をなすことを示す。

  多くのコンパイラは、例えば、定型パタンの最適化のために境界要素を余分に付け加えたり、全てのプロセッサが各配列に対して等しい大きさの記憶領域を割り付けるために、特定のプロセッサ上では、配列内にダミー要素をもたせたりしている。

  ローカルな並べ変えはどんなものでも、INTERFACE 宣言内の DISTRIBUTE 指示文、又は ALIGN 指示文によって指定できるグローバルなデータの再マッピングに加えて行われる。

  LAYOUT 属性が省略された場合は、LAYOUT('F77_ARRAY') が仮定される。

- LAYOUT('HPF_ARRAY') ローカルなデータの並べ変えなしで、グローバル HPF 手続の場合と全く同じように、配列引数が渡されることを意味する。

  このオプションは、ローカルな並べ変えのオーバヘッドをなくしたい場合か、グローバル HPF コンパイラによる (境界要素付加、プロセッサ間の等サイズ割り付けといった) 配置方法のある特性を、ローカル手続においても保存したい場合に必要である。このオプションを指定する場合、ローカル手続のプログラマは、グローバル HPF コンパイラが採用する処理系依存の方法で、ローカルなデータを参照しなければならない。

さらに、INTERFACE 宣言内の各引数には、PASS_BY 属性を指定でき、それにより以下のことを指定できる。即ち、Fortran 77 風のデータ参照を行うために参照渡しにするか、特別なハンドル (恐らく、HPF の変数を渡す際に用いられる記述子経由の引数渡し) により、グローバル HPF 呼出し側が特別なマッピング情報を渡して、それをローカル Fortran 77 手続内で利用できるようにするかである。

- PASS_BY('*') ローカル手続が、ローカルな仮引数を、参照渡しによる F77 の変数と同様に参照できることを意味する。

- PASS_BY('HPF_HANDLE') ローカル手続が受け取るのは、グローバルプログラム単位の記述子であり、特別な問合せ手続と共にそれを用いると、有効なマッピング情報を得ることができる、ということを意味する。

ここで、暗黙の仮引数の属性は、LAYOUT('F77_ARRAY')、及び PASS_BY('*') であるから、順序結合は保証され、引数はその記憶領域へのポインタ経由で渡される。

【実装者への助言】

優れた EXTRINSIC(F77_LOCAL) の実装では、引数の渡し方とデータの並べ変えに関するオプションの提供に加えて、形状引継ぎ配列として宣言するのではなく、HPF の場合と同様に、任意の大きさのローカルな部分格子を宣言して、それらを参照できるようにする、という問題が解決されているべきである。

外来手続呼出しによって起動される各ローカル手続内で、グローバルにマップされたデータのローカルな配置方法に対応することも、Fortran 90 の配列問合せ関数や、HPF ライブラリの問合せサブルーチン無しでは難しい。付録 G で提案されている Fortran 77 ライブラリ関数の引用仕様の様に、グローバルにもローカルにも呼出せる特別の問合せ関数を、EXTRINSIC(F77_LOCAL) 手続インタフェースに追加して、より広い範囲のグローバル HPF のデータマッピングを、柔軟かつ効率的に利用できる様にすることが望ましい。

【以上】

11.6.3 F77_LOCAL 手続のプログラミング例

11.6.3.1 LAYOUT('F77_ARRAY') と PASS_BY('*')

ここでは、既定値である LAYOUT('F77_ARRAY') 属性と PASS_BY('*') 属性を用いた F77_LOCAL 手続のプログラミング例と、LAYOUT('HPF_ARRAY') 属性を用いた、ローカル手続内での問合せ手続の利用例を述べる。

- 呼出し側 HPF プログラム単位

```
      PROGRAM EXAMPLE

! 配列データと、値をチェックするためのコピーの宣言
      INTEGER, PARAMETER :: NX = 100, NY = 100
      REAL, DIMENSION(NX,NY) :: X, Y
!HPF$ DISTRIBUTE(BLOCK,BLOCK) :: X, Y

! 全体の和は、各プロセッサ上の部分和を
! 結合することにより計算できる。
```

```
      REAL PARTIAL_SUM(NUMBER_OF_PROCESSORS())
!HPF$ DISTRIBUTE PARTIAL_SUM(BLOCK)

! ローカルな部分格子である引数は、各プロセッサ毎に、
! 2次元配列として宣言される。
      INTEGER, DIMENSION(NUMBER_OF_PROCESSORS(),2) ::
     & LB, UB, NUMBER
!HPF$ DISTRIBUTE(BLOCK,*) :: LB, UB, NUMBER

! 引用仕様宣言
      INTERFACE

        EXTRINSIC(F77_LOCAL) SUBROUTINE LOCAL1
     &   ( LB1, UB1, LB2, UB2, X , X_DESC )
        INTEGER, DIMENSION(:) :: LB1, UB1, LB2, UB2
        REAL,DIMENSION(:,:),LAYOUT('HPF_ARRAY') :: X
        REAL,DIMENSION(:,:),LAYOUT('HPF_ARRAY'),     &
               PASS_BY('HPF_HANDLE')           :: X_DESC

!HPF$   DISTRIBUTE(BLOCK) :: LB1, UB1, LB2, UB2
!HPF$   DISTRIBUTE(BLOCK,BLOCK) :: X, X_DESC
        END

        EXTRINSIC(F77_LOCAL) SUBROUTINE LOCAL2(N,X,R)
        INTEGER N(:)
        REAL X(:,:), R(:)
! 既定値:
!       LAYOUT('F77_ARRAY')      行優先の連続した記憶領域
!       PASS_BY('*')             参照渡し (ローカルなアドレス)
!HPF$   DISTRIBUTE N(BLOCK)
!HPF$   DISTRIBUTE X(BLOCK,BLOCK)
!HPF$   DISTRIBUTE R(BLOCK)
        END

      END INTERFACE

! グローバル HPF だけを用いて、結果を求める。

      ! 初期化
      FORALL (I=1:NX,J=1:NY) X(I,J) = I + (J-1) * NX

      ! 全体の和の計算と報告
```

```
      PRINT *, 'Global HPF result: ',SUM(X)

! ローカルサブルーチンを用いて、結果を求める。

      ! 初期化 ( 整列の刻み幅は、1 と仮定する )
      CALL HPF_SUBGRID_INFO( Y, IERR, LB=lb, UB=UB )
      IF (IERR.NE.0) STOP 'ERROR!'
      CALL LOCAL1( LB(:,1), UB(:,1), LB(:,2), UB(:,2), Y , Y )

      ! 全体の和の計算と報告
      NUMBER = UB - LB + 1
      CALL LOCAL2 ( NUMBER(:,1) * NUMBER(:,2) , Y , PARTIAL_SUM )
      PRINT *, 'F77_LOCAL result #1 : ',SUM(PARTIAL_SUM)

      END
```

- 呼ばれ側 FORTRAN 77 手続

```
      SUBROUTINE LOCAL1( LB1, UB1, LB2, UB2, X , DESCRX )

      REAL X ( LB1 : UB1 , LB2 : UB2 )
      INTEGER DESCRX ( * )

! 最初の次元の、グローバル手続内での寸法を得る。
! これは、接頭文字列'F77_'をもつ HPF_LOCAL 種別の問合せ関数である。
      CALL F77_GLOBAL_SIZE ( NX , DESCRX , 1 )

! 配列要素の初期化
      DO J = LB2, UB2
        DO I = LB1, UB1
          X(I,J) = I + (J-1) * NX
        END DO
      END DO

      END



      SUBROUTINE LOCAL2(N,X,R)
! ここでは、グローバル手続内での添字との対応は重要ではない
! 部分格子の総数だけが渡される。
      REAL X(N)

```

```
      R = 0.
      DO I = 1, N
        R = R + X(I)
      END DO

      END
```

11.6.3.2 LAYOUT('HPF_ARRAY') と PASS_BY('HPF_HANDLE')

この例では、上の例の初期化部分だけを行う。F77_LOCAL 手続内で、部分格子問合せ関数を利用するための、HPF 風記述子又はハンドルを渡すために、PASS_BY('HPF_HANDLE') を用いる例と共に、HPF のマップされた配列を、再マップすることなく渡すために、LAYOUT('HPF_ARRAY') 属性を利用する例を述べる。また、「埋め込み配列」に特有のデータ参照法についても例示する。

- 呼出し側 HPF プログラム単位

```
      PROGRAM EXAMPLE

      INTEGER, PARAMETER :: NX = 100, NY = 100
      REAL, DIMENSION(NX,NY) :: Y
!HPF$ DISTRIBUTE(BLOCK,BLOCK) :: Y

! ローカルな部分格子である引数は、プロセッサ毎に、
! 2 次元配列として宣言される。
      INTEGER, DIMENSION(NUMBER_OF_PROCESSORS(),2) ::
     & LB, UB, LB_EMBED, UB_EMBED
!HPF$ DISTRIBUTE(BLOCK,*) :: LB, UB, LB_EMBED, UB_EMBED


! 引用仕様宣言

      INTERFACE

        EXTRINSIC(F77_LOCAL) SUBROUTINE LOCAL1(
     &   LB1, UB1, LB_EMBED1, UB_EMBED1,
     &   LB2, UB2, LB_EMBED2, UB_EMBED2, X, X_DESC )
        INTEGER, DIMENSION(:) ::
     &   LB1, UB1, LB_EMBED1, UB_EMBED1,
     &   LB2, UB2, LB_EMBED2, UB_EMBED2
!             既定値で、X は参照渡しとなる。
        REAL, DIMENSION(:,:), LAYOUT('HPF_ARRAY') :: X
!             X_DESC は、記述子又は「ハンドル」により渡される。
```

```
        REAL, DIMENSION(:,:), LAYOUT('HPF_ARRAY'),
     &                    PASS_BY('HPF_HANDLE') :: X_DESC
!HPF$ DISTRIBUTE(BLOCK) :: LB1, UB1, LB_EMBED1, UB_EMBED1
!HPF$ DISTRIBUTE(BLOCK) :: LB2, UB2, LB_EMBED2, UB_EMBED2
!HPF$ DISTRIBUTE(BLOCK,BLOCK) :: X
        END

      END INTERFACE

! 初期化
! ( 整列の刻み幅を 1 とし、次元の置換はないと仮定する )

      CALL HPF_SUBGRID_INFO( Y, IERR,
     & LB=LB, LB_EMBED=LB_EMBED,
     & UB=UB, UB_EMBED=UB_EMBED)
      IF (IERR.NE.0) STOP 'ERROR!'

      CALL LOCAL1(
     & LB(:,1), UB(:,1), LB_EMBED(:,1), UB_EMBED(:,1),
     & LB(:,2), UB(:,2), LB_EMBED(:,2), UB_EMBED(:,2), Y, Y )

      END
```

- 呼ばれ側 Fortran 77 手続

```
      SUBROUTINE LOCAL1(
     & LB1, UB1, LB_EMBED1, UB_EMBED1,
     & LB2, UB2, LB_EMBED2, UB_EMBED2, X, X_DESC )

! 部分格子は、「埋め込まれた」形で渡されている。
      REAL X ( LB_EMBED1 : UB_EMBED1 , LB_EMBED2 : UB_EMBED2 )

! この引数は、問合せ関数への入力としてのみ利用される。
      INTEGER X_DESC

! 最初の次元の、グローバル手続内での寸法を得る。
! これは、接頭文字列'F77_'をもつ HPF_LOCAL 種別の問合せ手続である。
      CALL F77_GLOBAL_SIZE(NX,X_DESC,1)

! 配列要素の初期化
! ループの範囲は、実引数として渡される配列要素上のみである。
      DO J = LB2, UB2
```

```
      DO I = LB1, UB1
        X(I,J) = I + (J-1) * NX
      END DO
    END DO

    END
```

11.7 外来手続のためのライブラリ

以下では、外来副プログラムのプログラミングに有用な手続の、Fortran 言語による結合仕様を述べる。

11.7.1 ローカル HPF 手続のためのライブラリ

ローカル HPF 手続内では、任意の HPF 組込み手続、ライブラリ手続を引用することができる。

> 【実装者への助言】そのような手続への引数は、ローカル配列である。実装方法に依存して、ローカル HPF 手続から引用される組込み手続、ライブラリ手続の実際のコードは、グローバル HPF コードから引用された場合のコードと同じであってもよいし、異なっていてもよい。【以上】

さらに、ローカルライブラリ手続 GLOBAL_ALIGNMENT、GLOBAL_DISTRIBUTION、及び GLOBAL_TEMPLATE が提供されているので、外来関数に対する実引数のグローバルなマッピングを問合せることができる。その他のローカルライブラリ手続は、実引数の大きさ、形状、及び配列の上下限を問合せるために提供される。これらのライブラリ手続は、仮引数の名前を入力として、対応するグローバル HPF の実引数に関する情報を返す。これらは、グローバル HPF コードから直接起動されたローカル手続内でのみ引用することができる。もしモジュールの機能が利用可能であれば、それらは、HPF_LOCAL_LIBRARY というモジュール内で定義される。それらを引用するローカル手続には、以下の文

```
USE HPF_LOCAL_LIBRARY
```

あるいは、これに代わる適切な機能をもつ文を記述しなければならない。

HPF ローカル手続ライブラリは、各物理プロセッサを 0 から $n-1$ までの整数値で識別する。ここで、n は、グローバル HPF 組込み関数 NUMBER_OF_PROCESSORS が返す値である。プロセッサ識別子は、ABSTRACT_TO_PHYSICAL の返値であるが、それにより、HPF のプロセッサ構成である抽象プロセッサと、物理プロセッサとの間に、一対一対応が定められる。さらに、ローカルライブラリ関数 MY_PROCESSOR は、それを呼び出したプロセッサの識別子を返す。

ローカル HPF ライブラリ手続のある引数が、グローバル HPF の実引数と結合するローカル手続の仮引数でなければならないときにはどんな場合でも、その結合は推移的ではない。つまり、ローカル手続の仮引数は、グローバル HPF から引用された手続の仮引数でなければならず、他のローカル副プログラムから引用された手続の仮引数であってはならない。

11.7.1.1 ブロック単位での仮引数参照

グローバル HPF 配列の物理プロセッサへのマッピングによって、1 つ、あるいはそれ以上の「**ブロック**」が、各プロセッサ上へ割り当てられる。ここで、**ブロック**とは、添字が連続している要素の集合のことである。あるプロセッサにマップされているブロックの数は、そのプロセッサにマップされている、各次元の添字が連続しているブロックの数を掛け合わせたものである。例えば、1 次元配列 X が、CYCLIC(4) で分散されている場合、それは、4 要素からなるブロックをもつ。ただし、最後のブロックは、SIZE(X) mod 4 要素になる可能性もある。一方、X が、CYCLIC(4) で分散されているテンプレート又は配列に整列しており、整列の刻み幅が 1 でない場合 (例えば、!HPF$ ALIGN X(I) WITH T(3*I))、そのブロックに含まれる要素は、4 より少ない。整列の刻み幅が 3 で、そのテンプレートが、各ブロック 4 要素のブロック-サイクリック分散の場合には、X の各ブロックは、1 要素又は 2 要素からなる。整列の刻み幅が 5 の場合、X の全てのブロックは、ちょうど 1 要素からなる。というのは、配列要素が全く整列されていないテンプレートのブロックは、ブロック数を数える際には無視されるからである。

HPF_LOCAL 副プログラムにおいて、仮引数と結合するグローバル配列引数の一部分は、ブロック毎に参照することができる。3 つのローカルライブラリ手続 LOCAL_BLKCNT、LOCAL_LINDEX、及び LOCAL_UINDEX により、ローカルな記憶領域内の特定のブロックを容易に参照できる。以下の例で、この様な目的のためにこれらの手続を利用する方法が示されている。そこでは、ローカルなデータは、一度に 1 ブロックずつ初期化されている。

```
EXTRINSIC(HPF_LOCAL) SUBROUTINE NEWKI_DONT_HEBLOCK(X)
REAL X(:,:,:)
INTEGER BL(3)
INTEGER, ALLOCATABLE :: LIND1(:), LIND2(:), LIND3(:)
INTEGER, ALLOCATABLE :: UIND1(:), UIND2(:), UIND3(:)

BL = LOCAL_BLKCNT(X)

ALLOCATE (LIND1(BL(1)))
ALLOCATE (LIND2(BL(2)))
ALLOCATE (LIND3(BL(3)))

ALLOCATE (UIND1(BL(1)))
ALLOCATE (UIND2(BL(2)))
ALLOCATE (UIND3(BL(3)))

LIND1 = LOCAL_LINDEX(X, DIM = 1)
UIND1 = LOCAL_UINDEX(X, DIM = 1)

LIND2 = LOCAL_LINDEX(X, DIM = 2)
UIND2 = LOCAL_UINDEX(X, DIM = 2)
```

```
LIND3 = LOCAL_LINDEX(X, DIM = 3)
UIND3 = LOCAL_UINDEX(X, DIM = 3)

DO IB1 = 1, BL(1)
  DO IB2 = 1, BL(2)
    DO IB3 = 1, BL(3)
      FORALL (I1 = LIND1(IB1) : UIND1(IB1),  &
              I2 = LIND2(IB2) : UIND2(IB2),  &
              I3 = LIND3(IB3) : UIND3(IB3) ) &
                X(I1, I2, I3) = IB1 + 10*IB2 + 100*IB3
    ENDDO
  ENDDO
ENDDO
END SUBROUTINE NEWKI_DONT_HEBLOCK
```

GLOBAL_ALIGNMENT(ARRAY, ...)

HPF の問合せサブルーチン HPF_ALIGNMENT と引用仕様や動作は同じ。ただしローカル配列に関する情報ではなく、ローカル仮引数 ARRAY と結合するグローバル HPF の実配列引数に関する情報を返す。

GLOBAL_DISTRIBUTION(ARRAY, ...)

HPF の問合せサブルーチン HPF_DISTRIBUTION と引用仕様や動作は同じ。ただしローカル配列に関する情報ではなく、ローカル仮引数 ARRAY と結合するグローバル HPF の実配列引数に関する情報を返す。

GLOBAL_TEMPLATE(ARRAY, ...)

HPF の問合せサブルーチン HPF_TEMPLATE と引用仕様や動作は同じ。ただしローカル配列に関する情報ではなく、ローカル仮引数 ARRAY と結合するグローバル HPF の実配列引数に関する情報を返す。

GLOBAL_SHAPE(SOURCE)

機能. HPF_LOCAL 手続の配列又はスカラ仮引数と結合するグローバル HPF の実引数の形状を返す。

分類. 問合せ関数。

引数.

SOURCE　任意の型でよい。配列でもスカラでもよい。グローバル HPF の実引数と結合する HPF_LOCAL 手続の仮引数でなければならない。

結果の型、型パラメタ、及び形状. 結果は、基本整数型 1 次元配列であって、大きさが SOURCE の次元数と同じとする。

結果の値. 結果の値は、SOURCE と結合するグローバル実引数の形状とする。

例示. A が、 INTEGER A(3:100, 200) と宣言されていて、B と結合する引数であるとき、GLOBAL_SHAPE(B) の値は、(/98, 200/) になる。B が、部分配列 A(5:10, 10) と結合する引数のとき、GLOBAL_SHAPE(B) の値は、(/6/) になる。

GLOBAL_SIZE(ARRAY, DIM)

省略可能な引数. DIM

機能. HPF_LOCAL 手続の仮配列引数と結合するグローバル HPF の実配列引数の、指定された次元の寸法を返す。

分類. 問合せ関数

引数.

ARRAY 任意の型でよい。スカラであってはならない。グローバル HPF の実引数と結合する HPF_LOCAL 手続の仮引数でなければならない。

DIM (省略可能) 整数型スカラであって、$1 \leq$ DIM $\leq n$ の範囲の値でなければならない。ここで、n は ARRAY の次元数とする。

結果の型、型パラメタ、及び形状. 基本整数型スカラとする。

結果の値. 結果の値は、ARRAY と結合する実引数の DIM 次元目の寸法と同じであるか、又は DIM が省略されたとき、ARRAY と結合する実引数の要素の総数と同じとする。

例示. A が、 INTEGER A(3:10, 10) と宣言されていて、B と結合する引数であるとき、GLOBAL_SIZE(B, 1) の値は、8 になる。B が、部分配列 A(5:10, 2:4) と結合する引数のとき、GLOBAL_SIZE(B) の値は、18 になる。

ABSTRACT_TO_PHYSICAL(ARRAY, INDEX, PROC)

機能. グローバル実配列引数に対して指定されている抽象プロセッサに対応する物理プロセッサの、プロセッサ識別子を返す。

分類. サブルーチン。

引数.

ARRAY 任意の型でよい。グローバル HPF の実配列引数と結合する仮配列でなければならない。INTENT(IN) 引数とする。

INDEX　整数型 1 次元配列であって、その値は、グローバル HPF において、配列がマップされている抽象プロセッサの座標でなければならない。INTENT(IN) 引数とする。INDEX の大きさは、抽象プロセッサの次元数と同一でなければならない。i 番目の要素は、1 から e_i までの範囲の値でなければならない。ここで、e_i は、抽象プロセッサの i 次元目の寸法とする。

PROC　整数型スカラでなければならない。INTENT(OUT) 引数とする。INDEX によって指定された抽象プロセッサに対応する物理プロセッサの識別子を受け取る。

PHYSICAL_TO_ABSTRACT(ARRAY, PROC, INDEX)

機能. 指定された物理プロセッサに対応する、グローバル実配列引数がマップされている抽象プロセッサの座標を返す。

分類. サブルーチン。

引数.

ARRAY　任意の型でよい。グローバル HPF の実配列引数と結合する仮配列でなければならない。INTENT(IN) 引数とする。

PROC　基本実数型スカラでなければならない。INTENT(IN) 引数とする。物理プロセッサの識別子を指定する。

INDEX　整数型 1 次元配列でなければならない。INTENT(OUT) 引数とする。INDEX の大きさは、グローバル HPF において配列がマップされている抽象プロセッサの次元数と同一でなければならない。INDEX は、PROC により指定された物理プロセッサに対応する抽象プロセッサの座標を受け取る。i 番目の要素は、1 から e_i までの範囲の値でなければならない。ここで、e_i は、抽象プロセッサの i 次元目の寸法とする。

この手続は、抽象プロセッサと物理プロセッサが 1 対 1 に対応するシステム上でだけ引用することができる。この対応が 1 対多であるようなシステムでは、等価な処理系依存の手続が提供される。

LOCAL_TO_GLOBAL(ARRAY, L_INDEX, G_INDEX)

機能. ローカル仮配列のローカルな座標を、それと結合するグローバル HPF の実配列引数の、対応するグローバルな座標に変換する。

分類. サブルーチン。

引数.

ARRAY　任意の型でよい。グローバル HPF の実配列引数と結合する仮配列でなければならない。INTENT(IN) 引数とする。

L_INDEX　整数型 1 次元配列であって、大きさが ARRAY の次元数と同一でなければならない。INTENT(IN) 引数とする。ローカル仮配列 ARRAY の要素の座標をもつ。i 番目の要素は、1 から e_i までの範囲の値でなければならない。ここで、e_i は、ARRAY の i 次元目の寸法とする。

G_INDEX　整数型 1 次元配列であって、大きさが ARRAY の次元数と同一でなければならない。INTENT(OUT) 引数とする。L_INDEX により指定されたローカル配列の要素に対応する、グローバル HPF の実配列引数の座標を受け取る。i 番目の要素は、1 から e_i までの範囲の値とする。ここで、e_i は、ARRAY と結合するグローバル HPF の実配列引数の i 次元目の寸法とする。

GLOBAL_TO_LOCAL(ARRAY, G_INDEX, L_INDEX, LOCAL, NCOPIES, PROCS)

省略可能な引数. L_INDEX, LOCAL, NCOPIES, PROCS

機能. グローバル HPF の実配列引数のグローバルな座標を、それと結合するローカル仮配列の、対応するローカルな座標に変換する。

分類. サブルーチン。

引数.

ARRAY　任意の型でよい。グローバル HPF の実配列引数と結合する仮配列でなければならない。INTENT(IN) 引数とする。

G_INDEX　整数型 1 次元配列であって、大きさが ARRAY の次元数と同一でなければならない。INTENT(IN) 引数とする。ローカル仮配列 ARRAY と結合する、グローバル HPF の実配列引数の要素の座標をもつ。i 番目の要素は、1 から e_i までの範囲の値でなければならない。ここで、e_i は、ARRAY と結合するグローバル HPF の実配列引数の i 次元目の寸法とする。

L_INDEX (省略可能)　整数型 1 次元配列であって、大きさが ARRAY の次元数と同一でなければならない。INTENT(OUT) 引数とする。G_INDEX によって指定された、グローバル実配列引数の要素に対応する、ローカル仮配列の座標を受け取る。(同一のグローバル配列要素のコピーを所有するプロセッサ上では、これらの座標は同一である。) i 番目の要素は、1 から e_i までの範囲の値とする。ここで、e_i は、ARRAY の i 次元目の寸法とする。

LOCAL (省略可能)　論理型スカラでなければならない。INTENT(OUT) 引数とする。ローカル配列がグローバル配列要素のコピーをもつとき、.TRUE. に、そうでないとき、.FALSE. に設定されるとする。

NCOPIES (省略可能)　整数型スカラでなければならない。INTENT(OUT) 引数とする。指定されたグローバル実配列の要素のコピーを所有するプロセッサ数が設定される。

PROCS (省略可能) 整数型 1 次元配列であって、大きさが、グローバル実配列の指定された要素のコピーを所有するプロセッサ数以上でなければならない。そのようなプロセッサの識別子が、PROCS に設定される。その出現順序は、処理系依存とする。

MY_PROCESSOR()

機能. 呼び出した物理プロセッサの識別子を返す。

分類. ピュア関数。

結果の型、型パラメタ、及び形状. 結果は、基本整数型スカラとする。

結果の値. 呼出しを行った物理プロセッサの識別子を返す。$0 \leq$ MY_PROCESSOR $\leq n-1$ の範囲の値とする。ここで、n は、NUMBER_OF_PROCESSORS が返す値とする。

LOCAL_BLKCNT(ARRAY, DIM, PROC)

省略可能な引数. DIM, PROC.

機能. 与えられたプロセッサ上に割り当てられている配列の、各次元の要素のブロック数を返すか、又は指定された次元における要素のブロック数を返す。

分類. ピュア関数。

引数.

ARRAY 任意の型でよい。グローバル HPF の実配列引数と結合する仮配列でなければならない。

DIM (省略可能) 整数型スカラであって、$1 \leq$ DIM $\leq n$ の範囲の値でなければななならない。ここで、n は、ARRAY の次元数とする。対応する実引数は、省略可能な仮引数であってはならない。

PROC (省略可能) 整数型スカラでなければならない。有効なプロセッサ番号でなければならない。

結果の型、型パラメタ、及び形状. 結果は、基本整数型とする。DIM が指定されたときスカラとする。そうでなければ、結果は 1 次元配列であって、大きさが n とする。ここで、n は、ARRAY の次元数とする。

結果の値.

場合 *(i)*: LOCAL_BLKCNT(ARRAY, DIM, PROC) の値は、プロセッサ PROC にマップされている ARRAY の最終整列先の、DIM 次元目のブロック数とする。ARRAY の要素が、少なくとも 1 つは整列されているブロックだけを数えるものとする。

場合 *(ii)*: LOCAL_BLKCNT(ARRAY, DIM) は、LOCAL_BLKCNT(ARRAY, DIM, PROC=MY_PROCESSOR()) と同じ値を返す。

場合 *(iii)*: LOCAL_BLKCNT(ARRAY) の値は、$i = 1, \ldots, n$ に対して、i 番目の成分が、LOCAL_BLKCNT(ARRAY, i) と同じとする。ここで、n は、ARRAY の次元数とする。

例示. 以下のように宣言されたとき、

```
      REAL A(20,20), B(10)
!HPF$ TEMPLATE T(100,100)
!HPF$ ALIGN B(J) WITH A(*,J)
!HPF$ ALIGN A(I,J) WITH T(3*I, 2*J)
!HPF$ PROCESSORS PR(5,5)
!HPF$ DISTRIBUTE T(CYCLIC(3), CYCLIC(3)) ONTO PR
!HPF$ CALL LOCAL_COMPUTE(A, B)
      ...
      ...
      ...
      EXTRINSIC(HPF_LOCAL) SUBROUTINE LOCAL_COMPUTE(X, Y)
      USE HPF_LOCAL_LIBRARY
      REAL X(:,:), Y(:)
      INTEGER NBY(1), NBX(2)
      NBX = LOCAL_BLKCNT(X)
      NBY = LOCAL_BLKCNT(Y)
```

PR(2,4) に対応する物理プロセッサにおいて、NBX の値は、(/4,3/) になり、NBY の値は、(/1/) になる。

LOCAL_LINDEX(ARRAY, DIM, PROC)

省略可能な引数. PROC.

機能. 仮配列引数の与えられた次元における、プロセッサ上の全てのブロックのローカルな添字の下限を返す。

分類. ピュア関数。

引数.

ARRAY　任意の型でよい。グローバル HPF の実配列引数と結合する仮配列でなければならない。

DIM　整数型スカラであって、$1 \leq$ DIM $\leq n$ の範囲の値でなければならない。ここで、n は、ARRAY の次元数とする。

PROC (省略可能) 整数型スカラでなければならない。有効なプロセッサ番号でなければならない。

結果の型、型パラメタ、及び形状. 結果は、基本整数型 1 次元配列であって、大きさは b とする。

ここで、b は、LOCAL_BLKCNT(ARRAY, DIM [, PROC]) の返値とする。

結果の値.

場合 *(i):* LOCAL_LINDEX(ARRAY, DIM, PROC) の i 番目の成分の値は、プロセッサ PROC における、ARRAY の DIM 次元目の、i 番目のブロックの最初の要素のローカルな添字とする。i 番目の要素は、1 から e_i までの範囲の値とする。ここで、e_i は、ARRAY の i 次元目の寸法とする。

場合 *(ii):* LOCAL_LINDEX(ARRAY, DIM) は、LOCAL_LINDEX(ARRAY, DIM, PROC=MY_PROCESSOR()) と同じ値を返す。

例示. LOCAL_BLKCNT の例と同じ宣言があるとき、PR(2,4) に対応する物理プロセッサにおいて、

LOCAL_LINDEX(X, DIM=1) の値は、(/1,2,3,4/) になる。LOCAL_LINDEX(X, DIM=2) の値は、(/1,3,4/) になる。

LOCAL_UINDEX(ARRAY, DIM, PROC)

省略可能な引数. PROC.

機能. 仮配列引数の与えられた次元における、プロセッサ上の全てのブロックのローカルな添字の上限を返す。

分類. ピュア関数。

引数.

ARRAY 任意の型でよい。グローバル HPF の実配列引数と結合する仮配列でなければならない。

DIM 整数型スカラであって、$1 \leq$ DIM $\leq n$ の範囲の値でなければならない。ここで、n は、ARRAY の次元数とする。

PROC (省略可能) 整数型スカラでなければならない。有効なプロセッサ番号でなければならない。

結果の型、型パラメタ、及び形状. 結果は、基本整数型 1 次元配列であって、大きさは b とする。ここで、b は、LOCAL_BLKCNT(ARRAY, DIM [, PROC]) の返値とする。

結果の値.

場合 *(i)*: LOCAL_UINDEX(ARRAY, DIM, PROC) の i 番目の成分の値は、プロセッサ PROC における、ARRAY の DIM 次元目の、i 番目のブロックの最後の要素のローカルな添字とする。i 番目の要素は、1 から e_i までの範囲の値とする。ここで、e_i は、ARRAY の i 次元目の寸法とする。

場合 *(ii)*: LOCAL_UINDEX(ARRAY, DIM) は、LOCAL_UINDEX(ARRAY, DIM, PROC=MY_PROCESSOR()) と同じ値を返す。

例示. LOCAL_BLKCNT の例と同じ宣言があるとき、PR(2,4) に対応する物理プロセッサにおいて、

LOCAL_UINDEX(X, DIM=1) の値は、(/1,2,3,4/) になる。LOCAL_UINDEX(X, DIM=2) の値は、(/1,3,4/) になる。

11.7.2 シリアル外来種別からのライブラリ引用

SERIAL 副プログラムは、HPF_ALIGNMENT、HPF_DISTRIBUTION、 HPF_TEMPLATE を除く、任意の HPF_LIBRARY 手続や HPF 組込み関数を引用できる。SERIAL プログラム単位の有効域内で、HPF_LOCAL_LIBRARY モジュールを引用してはならない。

組込み関数 NUMBER_OF_PROCESSORS や PROCESSORS_SHAPE の返値は、グローバル HPF 内で引用された場合と同一である。