第6章　外来プログラム単位

HPF のグローバルな計算モデルは、Fortran を拡張 (および制限) することによって、広範なハードウェアアーキテクチャにおいて効率的に実装することのできる Fortran の計算モデルを、プログラマに提供する。対象とするアーキテクチャは、一般に、複数プロセッサ、非一様アクセス特性を持つ複数メモリ、及び複数の結合網を持つ。この計算モデルは、論理的に単一の制御スレッドを提供しており、配列構文、FORALL 文などの Fortran のデータ並列機能、および、Fortran の有効域規則により定義されるデータ可視性を含んでいる。特にこのモデルは、このようなアーキテクチャを利用するための低レベルの機能、例えばスレッドライブラリや明示的メッセージ通信などを要求しない。プログラマは、HPF コンパイラが、HPF の機能を手助けとして、与えられたハードウェアアーキテクチャに対してデータや計算のマッピングを行い、効率の良いコードを生成すると期待できる。

この章では、HPF で記述されたプログラム単位から、HPF のグローバルモデルを使用しない非 HPF プログラム単位を使用するための、外来 (*extrinsic*) 機構を定義する。非 HPF 手続に対する明示的引用仕様をどのように記述するかを述べ、分散および複製されたデータの扱いが、その引用仕様によって呼出し元でどのように仮定されるかを定義する。これにより、プログラマは、HPF 以外の言語の機能を利用できるようになる。例えば、HPF では効率的に扱えない問題に対しての低レベル機能の利用、重要なカーネル部分の人手チューニング、および最適化されたライブラリの呼出しなどが可能となる。このインタフェースは、C のような他言語と HPF とのインタフェースとしても利用できる。

6.1　概要

ある HPF プログラムから、他のプログラミングモデルや異なる言語で実装された手続を呼び出したい場合があるかもしれない。手続の**プログラミングモデル** (*programming model*) には、以下のようなものが与えられる可能性がある。

- 単一制御スレッドで、概念上は手続の一つのコピーが実行され、プログラムテキストには制御の単一の軌跡が存在するもの。このモデルは前提となるターゲットハードウェアが (潜在的に) 複数のプロセッサあるいはメモリを持っている場合にグローバル *(global)* と呼ばれ、前提となるターゲットハードウェアが単一プロセッサ (あるいはマルチプロセッサの中の 1 つのノード) である場合に、シリアル (*serial*) と呼ばれる。

- 複数制御スレッド (プロセッサ毎に 1 つ) で、それぞれのスレッドが同じ手続を実行するもの。このモデルは ローカル (*local*)、より一般的には SPMD(Single Program, Multiple Data) と呼ばれる。

- ここでは論じない、何等かの他のモデル。例えば、複数制御スレッドであり、プロセッサに対するループ繰返しの動的割り当てや、明示的なプロセス生成を備えているような

物である。呼出時の最初の時点で、概念上実行される手続のコピーが一つあるが、このコピーはプログラムテキスト内に複数の制御の軌跡を (おそらくは時間とともに数を変えながら) 生み出す可能性がある。

プログラミング言語 (*programming language*) は、ある特定の文法 (言語仕様)、語義 (意味)、及び実用性 (目的) を提供する。プログラミング言語の中には、Fortran(ANSI 及び ISO 規格。最新版は 1997 年に認可されるはずである)、HPF(Fortran に拡張と制約を加えた仕様)、Fortran 77(以前の ANSI 及び ISO 規格)、C 、C++ 、Java、Visual Basic、 そして COBOL が含まれる。

プログラム単位の言語とモデルは、全体でその外来種別 (*extrinsic kind*) を構成する。この外来種別は、外来プレフィックス (*extrinsic prefix*) によって明示的に、またはコンパイラの選択やある一連のコンパイラオプションによる起動によって暗黙に指定することができる。従って、コンパイラを Fortran で定義されている**親有効域** (*host scoping unit*) を与えるものとしてみることができる。例えば、HPF コンパイラでコンパイルされたあるプログラムは、`HPF` という外来種別に属することになる。また、プログラムの外来種別は `EXTRINSIC(HPF)` あるいは `EXTRINSIC(LANGUAGE='HPF',MODEL='GLOBAL')` のように明示的に外来プレフィックスを使用して指定することができる。

6.2 外来プログラム単位の宣言

6.2.1 FUNCTION 文及び SUBROUTINE 文

外来プレフィックス (*extrinsic-prefix*) は、(Fortran の規格で定義された) FUNCTION 文、及び SUBROUTINE 文の中のキーワード `RECURSIVE`、`PURE`、及び `ELEMENTAL` と同じ位置に記述することができる。これは、1995 年 5 月の Fortran95 規格のドラフトの中で、*prefix-spec* に関する規則 R1219 の拡張で規定されている。FUNCTION 文 (*function-stmt*) に関する規則 R1217、*prefix* に関する規則 R1218、及び SUBROUTINE 文 (*subroutine-stmt*) に関する規則 R1222 には変更はないが、参考のためにここに再掲する。

H601 *function-stmt* **is** [*prefix*] `FUNCTION` *function-name*
([*dummy-arg-name-list*])
[`RESULT` (*result-name*)]

H602 *subroutine-stmt* **is** [*prefix*] `SUBROUTINE` *subroutine-name*
[([*dummy-arg-list*])]

H603 *prefix* **is** *prefix-spec* [*prefix-spec*] ...

H604 *prefix-spec* **is** *type-spec*
or `RECURSIVE`
or `PURE`
or `ELEMENTAL`
or *extrinsic-prefix*

制約: 任意の HPF の外部副プログラム内で、どの内部副プログラムも、その親と同じ外来種

別でなければならず、また、明示的に外来種別が与えられていない内部副プログラムに対しては、その親と同じ外来種別が仮定される。

F95:12.2 で与えられている**手続の特性**の定義を拡張して、手続の外来種別を含むものとする。

6.2.2 PROGRAM 文、MODULE 文、及び BLOCK DATA 文

外来プレフィックス (*extrinsic-prefix*) は、PROGRAM 文、MODULE 文、及び BLOCK DATA 文の先頭にも記述することができる。以下の構文は、Fortran95 の構文規則 R1102(PROGRAM 文 (*program-stmt*))、R1105(MODULE 文 (*module-stmt*))、及び R1111(BLOCK DATA 文 (*block-data-stmt*)) の拡張である。

H605 *program-stmt* **is** [*extrinsic-prefix*] `PROGRAM` *program-name*

H606 *module-stmt* **is** [*extrinsic-prefix*] `MODULE` *module-name*

H607 *block-data-stmt* **is** [*extrinsic-prefix*] `BLOCK DATA`
[*block-data-name*]

制約: 任意の HPF のモジュール内で、どのモジュール副プログラムも、その親と同じ外来種別でなければならず、また明示的に外来種別が与えられていないモジュール副プログラムに対しては、その親と同じ外来種別が仮定される。

制約: HPF の任意の主プログラムあるいはモジュール副プログラム内で、どの内部副プログラムも、その親と同じ外来種別でなければならず、また明示的に外来種別が与えられていない内部副プログラムに対しては、その親と同じ外来種別が仮定される。

6.2.3 外来プレフィックス

H608 *extrinsic-prefix* **is** `EXTRINSIC (` *extrinsic-spec* `)`

H609 *extrinsic-spec* **is** *extrinsic-spec-arg-list*
or *extrinsic-kind-keyword*

H610 *extrinsic-spec-arg* **is** *language*
or *model*
or *external-name*

H611 *language* **is** [`LANGUAGE =`]
scalar-char-initialization-expr

H612 *model* **is** [`MODEL =`]
scalar-char-initialization-expr

H613 *external-name* **is** [`EXTERNAL_NAME =`]
scalar-char-initialization-expr

制約: *extrinsic-spec-arg-list* の中には、*language*、*model*、あるいは *external-name* の少なくとも 1 つが指定されなければならず、またどれも 2 回以上指定することはできない。

制約: もし、LANGUAGE=を使わずに *language* を指定する場合、*language* は *extrinsic-spec-arg-list* 中の最初の要素でなければならない。もし、MODEL=を使わずに *model* を指定する場合、LANGUAGE=のない *language* が *extrinsic-spec-arg-list* 中の最初の要素であり、*model* が 2 番目の要素でなければならない。もし、EXTERNAL_NAME=を使わずに *external-name* を指定する場合、LANGUAGE=のない *language* が *extrinsic-spec-arg-list* 中の最初の要素であり、MODEL=のない *model* が 2 番目の要素でなければならない。

制約: LANGUAGE=、MODEL=、EXTERNAL_NAME=を伴う形は、上で禁止された場合を除いて、いかなる順番で書いても構わない。

これらの *extrinsic-spec-arg-list* に関する規則は、あたかも EXTRINSIC が、LANGUAGE、MODEL、EXTERNAL_NAME という、それぞれが OPTIONAL であるような *dummy-arg-list* を使用した明示的引用仕様を伴った手続であるかのようなものであることに注意されたい。

制約: *language* の中では、*char-initialization-expr* の値は、以下のものが許される。

- 'HPF'　HPF 言語を指す。もし *model* が明示的に指定されていない場合、*model* には'GLOBAL'が暗黙に仮定される。
- 'FORTRAN' ANSI または ISO 規格の Fortran 言語を指す。もし *model* が明示的に指定されていない場合、*model* には'SERIAL'が暗黙に仮定される。
- 'F77' 以前の ANSI または ISO 規格である FORTRAN 77 言語を指す。もし *model* が明示的に指定されていない場合、*model* には'SERIAL'が暗黙に仮定される。
- 'C' ANSI 規格の C 言語を指す。もし *model* が明示的に指定されていない場合、*model* には'SERIAL'が暗黙に仮定される。
- 実装依存の値。暗黙の *model* は実装に依存する。

ほとんどの実装にとって、'C'は、引用仕様本体 (*interface-body*) 中に記述された FUNCTION 文か SUBROUTINE 文でしか許されないことに注意されたい。

制約: *language* が指定されていない場合、親有効域と同じものが仮定される。

制約: *model* において、*char-initialization-expr* の値は、以下のものが許される。

- 'GLOBAL' グローバルモデルを指す。
- 'LOCAL' ローカルモデルを指す。
- 'SERIAL' シリアルモデルを指す。
- 実装依存の値。

制約: *model* が指定されず、*language* の指定から暗黙に仮定されない場合、親有効域と同じものが仮定される。

制約: 名称が HPF の 3 文字から始まる全ての *language* 及び *model* は、本仕様及びその後継仕様の現在あるいは将来における定義のために予約されている。

制約: *external-name* において、*scalar-char-initialization-expr* の値は、その用途が外来種別によって決定される文字列である。例えば、ある外来種別は、*external-name* を、その手続が C の手続から参照された場合の名前を指定するために使用するかもしれない。そのような実装では、ユーザはコンパイラに、その名前が C コンパイラに理解できるように変換を行なうことを期待するであろう。もし *external-name* が指定されていない場合、その値は実装依存となる。

H614 *extrinsic-kind-keyword* **is** HPF
or HPF_LOCAL
or HPF_SERIAL

制約: EXTRINSIC(HPF) は EXTRINSIC('HPF','GLOBAL') と同値である。*extrinsic-prefix* が存在しないとき、HPF コンパイラはコンパイル単位を外来種別 HPF に属するかのように解釈する。従って、HPF コンパイラにとって、EXTRINSIC(HPF) あるいは EXTRINSIC('HPF', GLOBAL') と指定するのは冗長である。しかし、このような明示的な指定は、複数の外来種別をサポートしているコンパイラを使用する場合に必要となるかもしれない。

制約: EXTRINSIC(HPF_LOCAL) は EXTRINSIC('HPF','LOCAL') と同値である。外来種別が HPF_LOCAL であるような主プログラムは、外来種別が HPF で、実行部がそのサブルーチンの呼出しだけから構成された主プログラムから引数なしで呼び出される、外来種別 HPF_LOCAL のサブルーチンであるかのように振る舞う。

制約: EXTRINSIC(HPF_SERIAL) は EXTRINSIC('HPF','SERIAL') と同値である。外来種別が HPF_SERIAL であるような主プログラムは、外来種別が HPF で、実行部分がそのサブルーチンの呼出しだけから構成された主プログラムから引数なしで呼び出される、外来種別 HPF_SERIAL のサブルーチンであるかのように振る舞う。

制約: 名称が HPF の 3 文字から始まる全ての *extrinsic-kind-keyword* は、本仕様及びその後継仕様の現在あるいは将来における定義のために予約されている。

【実装者への助言】

他の *language* あるいは *model* を、コンパイラベンダが定義、提供することができる。それらは、この HPF 仕様の一部ではないとはいえ、HPF の外来種別の規則と精神に添うものであるよう求められる。

実装において、プログラマに対するある程度の制約を課すことがあるかもしれない。さらに、それぞれの外来種別が、一連の異なる制約を求めることがありうる。

例えば、並列プロセッサでの実装では、スカラ引数を複製して全てのプロセッサにコピーを提供するほうが便利な場合がある。これは、このプロセスが呼出し元に見えない限り許される。これを達成するための方法の一つは、副プログラムから戻るときに、該当のスカラ引数の全てのコピーが同じ値を持っていなければならないという制約をプログラマに課すことである。これは、仮引数が INTENT(OUT) 属性を持っていれば、副プログラムの戻りの時点までには全てのコピーが矛盾なく更新されていなければならないということを暗黙に意味している。【以上】

6.3 HPF 外来副プログラムの呼出し

外来手続の呼出しは、HPF で記述されている呼出し側プログラムから見て、副プログラムが HPF で記述されている場合と厳密に同じように振る舞う。HPF の外来種別を持つプログラム単位から呼び出された関数またはサブルーチンが、呼出し元からみて明示的引用仕様を持たない場合、呼出し元と同じ外来種別を持つと仮定される。

呼出し元と異なる外来種別を持つ副プログラムを呼び出すためには、副プログラムは、呼出し元に認識できる明示的引用仕様を持たねばならない。そして副プログラムは、呼出し元から見て、呼出し元と同じ外来種別のコードで記述された場合とおおむね同じように振る舞うことが期待される。この要請を満たすための責任のいくらかはコンパイラにかかっており、またいくらかはプログラマにかかっている。この引用仕様は外来手続の「HPF への見え方」を定義する。

HPF 以外のモデル、あるいは言語で記述された手続は、ローカル手続実行モデルを使用しているか否かにかかわらず、その手続を呼び出す HPF プログラムの中で、`EXTRINSIC` として宣言されていなければならない。外来プレフィックスは、指定された副プログラムを呼び出すときにどの種類のインタフェースを使用するかを宣言する。もし明示的な指定がなければ、ユーザは実装依存のインタフェースの正当性について全ての責任を負わねばならない。

HPF の外来種別をもつプログラム単位中の引用仕様宣言に現れる FUNCTION 文または SUBROUTINE 文は、言語の実装によってサポートされている任意の外来種別に対する外来プレフィックスを持つことができる。その様な FUNCTION 文または SUBROUTINE 文に、外来プレフィックスが現れない場合、引用仕様宣言が現れたプログラムと同じ HPF の外来種別であることが仮定される。

引用仕様本体によって定義されている手続の特性は、手続の定義におけるそれと一致していなければならない。

外来引用仕様を持つ手続についての定義および規則は、HPF の範疇外である。しかし、その様な手続に対する明示的引用仕様は、HPF の仕様に従っていなければならない。個々の HPF の実装は、外来種別をどのように選択してサポートしてもよく、また全く何もサポートしなくてもかまわない (HPF の実装において明らかにサポートされなければならない外来種別「`HPF`」自体を除いて)。

6.3.1 型、手続、データに対するアクセス

一般に、ある外来種別を与えられたプログラム単位は、Fortran の有効域の規則に従って、同じ外来種別を持つ他のプログラム単位中の、型、手続、データの名前を使用することができる。

異なる外来種別を持つ他のプログラム単位中の型、手続、データの名前を使用することには、表 6.1 に示される、以下で述べるような制限が加えられる。

外来種別 HPF を持つあるモジュール X が、他の外来種別 HPF を持つプログラム単位 Y で使用されている場合、X の項目の内で、Y が使用または呼び出すことを許可されたものの名前だけが参照結合できることに注意されたい。すなわち、Y が使用できないすべての項目については X が非公開属性とするか、Y の `USE` 文が、Y が使用可能な項目の名前だけを並べた `ONLY` 句を持っているかの何れかでなければならない。

		使用されるモジュールの外来種別		
		HPF	HPF_SERIAL	HPF_LOCAL
使用する	HPF	**T P D**	**T P**	**T P**
プログラムの	HPF_SERIAL	**T**	**T P D**	**T**
外来種別	HPF_LOCAL	**T**	**T**	**T P D**

T = 構造型の定義
P = 手続及び手続の引用仕様
D = データ実体

表 6.1: それぞれの HPF 外来種別に従って、モジュールからの引用が可能な言語要素

6.3.1.1 型

明示的にマップされた成分を持たない構造型の定義は「外来種別なし (extrinsic kind neutral)」と考えることができる。外来種別 HPF を持つ任意のプログラム単位は、外来種別 HPF を持つどのようなモジュールからでも構造型の定義を使用することができる。公認拡張では、構造型の成分のマッピングが許可されていることに注意されたい。

6.3.1.2 手続

HPF のグローバルなプログラムまたは手続は、グローバル、ローカル、あるいはシリアルな他の HPF 手続を呼び出すことができる。

HPF のローカルなプログラムまたは手続は、他のローカルな HPF 手続だけを呼び出すことができる。グローバルまたはシリアルな手続は呼び出せない。

HPF のシリアルなプログラムまたは手続は、他のシリアルな HPF 手続だけを呼び出すことができる。グローバルまたはローカルな手続は呼び出せない。

6.3.1.3 データ

HPF 種別を持つプログラム単位の任意の名前つき共通ブロック (COMMON) は、もし存在すれば、同じ外来種別を持つ他の全てのプログラム単位の同じ名前の共通ブロックと結合する。無名共通ブロックも同様である。(このような共通ブロックの記憶領域は該当の外来種別を持つプログラムの中で宣言されたデータ実体と同じ振る舞いをする。特に、HPF_LOCAL なプログラムでは、各プロセッサに共通ブロックのコピーが存在することになるであろう。)

どのような共通ブロック名も、単一のプログラム中で異なる HPF 種別を持つプログラム単位内で使用されてはならない。同様に、単一のプログラム中で異なる HPF 種別を持つプログラム単位内で、無名共通ブロックを使用することはできない。

6.3.2 呼出しの効果

外来手続の呼出しは、その引数を再マップしない通常の HPF 手続の呼出しと意味的に同じでなければならない。従って、外来手続の呼出しは、以下のように動作したかの**如くに** (*as if*)

振る舞わなければならない。HPF の技術用語で**如くに** (*as if*) とは、記述された動作が、あたかもそれが指定された順序で実際に行なわれたかのようにユーザには見える、ということを意味する。実装では、ユーザに正しい結果を提供するように、任意の動作を任意の順序で行ってかまわない。

1. 副プログラムの呼出しに先行する呼出し元の全ての動作は、副プログラムの実行のいかなる動作よりも前に完了していなければならない。また、副プログラムの全ての動作は、呼出し元の副プログラムの呼出しに後続するいかなる動作よりも前に完了していなければならない。
2. 全ての実引数は、必要ならば、外来手続の引用仕様宣言の (明示的あるいは暗黙の) 指示に従って再マップされる。従って、引用仕様に現れる HPF のマッピング指示は拘束力がある。コンパイラは、ローカル外来手続の呼出しにおいてこれらの指示に従わなければならない。非外来副プログラムの場合と同様に、実引数はどのようにマップされていてもかまわない。もし必要なら、それらは呼出しの前に正しくマップされた一時領域に自動的にコピーされ、外来手続の戻りの前に実引数にコピーバックされる。スカラ仮引数やスカラ関数の戻り値は、各プロセッサで複製されたかのように振る舞う。これらのマッピングは、引用仕様中に明示されてもかまわないが、それ以外のいかなる明示的マッピングも、HPF 規格合致ではない。
3. `IN`、`OUT`、及び `INOUT` の授受特性の制約は、守られなければならない。
4. HPF 変数 (HPF variable) は、同じ明示的引用仕様を持つ HPF 手続によって更新されることはありうるが、それ以外は更新されない。`HPF_LOCAL` および `HPF_SERIAL` 手続は HPF のグローバルデータを参照することも更新することもできないが、他の種類の外来手続は HPF 手続と同程度にそれが可能かもしれないことに注意されたい。
5. ある手続から戻り、呼出し元が実行を再開するとき、呼出しの後に呼出し元が使用できる全ての実体は呼出し前の状態に正確にマップされていなければならない。特に、必要であれば、引数のもとのマッピングは復元される。
6. 副プログラムの呼出しの前後では、HPF の実行環境から見て厳密に同じプロセッサ集合になる。

【実装者への助言】

呼出しよりも論理的に先行する動作の完了を保証するために、複数のプロセッサが呼出しの前に同期処理を行う必要があるかもしれない。

呼び出された手続で使用可能な変数のマッピングが複製である場合、全てのコピーは、呼出しに先だって、原始プログラムの逐次的な意味に従った正確な値に更新されなければならないかもしれない。

複製された変数が、もし手続の中で更新されているならば、矛盾しないように更新されなければならない。より正確には、ある手続が使用可能な変数のマッピングが複製であり、(一つのあるいは複数の複製された) 手続によって更新された場合、最後のプロセッサがローカル手続から戻った時点でその複製された変数の全てのコピーは同じ値を持っていなければならない。

実装においては、ローカルな副プログラムの戻りの前に、複製された変数が副プログラムによって矛盾なく更新されているか判定するかも知れない。しかし、実装でそれが要求されているわけではないことに注意されたい。これは単に、例えば実行性能とデバッグの容易性との兼ね合いである。

グローバル HPF の副プログラムにおいては、副プログラムからの戻り時にその効果が取り消される限り、実引数はどのように複製あるいは再マップされてもよいことに注意されたい。

呼出し元の処理が再び実行されるよりも前に、手続の全ての動作の論理的な完了を保証するために、複数のプロセッサが呼出しの後で同期処理を行わなければならないかもしれない。

【以上】

6.4 外来手続の例

以下の例について考える。

```
PROGRAM DUMPLING
  INTERFACE
    EXTRINSIC('HPF','LOCAL') SUBROUTINE GNOCCHI(P, L, X)
      INTERFACE
        SUBROUTINE P(Q)
          REAL Q
        END SUBROUTINE P
        EXTRINSIC('COBOL','LOCAL') SUBROUTINE L(R)
          REAL R(:,:)
        END SUBROUTINE L
      END INTERFACE
      REAL X(:)
    END SUBROUTINE GNOCCHI
    EXTRINSIC('HPF','LOCAL') SUBROUTINE POTSTICKER(Q)
      REAL Q
    END SUBROUTINE POTSTICKER
    EXTRINSIC('COBOL','LOCAL') SUBROUTINE LEBERKNOEDEL(R)
      REAL R(:,:)
    END SUBROUTINE LEBERKNOEDEL
  END INTERFACE
  ...
  CALL GNOCCHI(POTSTICKER, LEBERKNOEDEL, (/ 1.2, 3.4, 5.6 /) )
  ...
END PROGRAM DUMPLING
```

主プログラム DUMPLINGは、HPF コンパイラでコンパイルされるときは、暗黙に外来種別 HPF に所属する。3 つの外来サブルーチン GNOCCHI、POTSTICKER、および LEBERKNOEDEL の引用仕様が宣言されている。最初の 2 つは外来種別 HPF_LOCAL に所属し、3 番目は COBOL_LOCAL に所属する。GNOCCHI は、2 つの仮手続を引数とすることができるので、引用仕様はそのための宣言をしなければならない。仮引数 P には、外来プレフィックスが与えられていないので、その外来種別は親有効域の外来種別になり、サブルーチン GNOCCHI の宣言から、それは外来種別 HPF_LOCAL を持つことになる。結合する実引数の宣言から、 POTSTICKER は明示的な外来プレフィックスを持つ必要がある。なぜならば、その親有効域はプログラム DUMPLING であり、外来種別 HPF に属しているからである。

もう少し例を挙げる。最初の例では、BAGEL の 100 という明示的な大きさの宣言は、そのグローバルな大きさを表し、ローカルな大きさではないことに注意されたい。

```
INTERFACE
  EXTRINSIC('HPF','LOCAL') FUNCTION BAGEL(X)
    REAL BAGEL(100)
    REAL X(:)
      !HPF$ DISTRIBUTE (CYCLIC) :: BAGEL, X
  END FUNCTION
END INTERFACE
```

次の例では、ALIGN 文により、X、Y、及び Z が全て同じ形状を持っていることに注意されたい。

```
INTERFACE OPERATOR (+)
  EXTRINSIC('C','LOCAL') FUNCTION LATKES(X, Y) RESULT(Z)
    REAL, DIMENSION(:,:), INTENT(IN) :: X
    REAL, DIMENSION(:,:), INTENT(IN) :: Y
    REAL, DIMENSION(SIZE(X,1), SIZE(X,2)) :: Z
      !HPF$ ALIGN WITH X :: Y, Z
      !HPF$ DISTRIBUTE (BLOCK, BLOCK) X
  END FUNCTION
END INTERFACE
```

次の最後の例での引用仕様宣言では、2 つの外部手続は、一方は外来手続で、もう一方はそうではないが、同じ総称名に結合されており、その配列引数と同じ型のスカラを戻す。

```
INTERFACE KNISH
  FUNCTION RKNISH(X)                          !通常の HPF の引用仕様
    REAL X(:), RKNISH
  END RKNISH
  EXTRINSIC('SISAL') FUNCTION CKNISH(X)   !外来引用仕様
    COMPLEX X(:), CKNISH
  END CKNISH
END INTERFACE
```