第IV部

附属書

ここでは、High Performance Fortran 言語 バージョン 2.0 (第 I 部と第 II 部で記述) および、公認拡張仕様 (第 III 部で記述) の機能の文法と意味について記述を、参照利用のためにまとめる。ここは、厳密な意味での HPF 言語仕様書の一部ではない。

附属書 A　構文規則

この附属書は、本 High Performance Fortran 言語仕様書で定義される構文規則の一覧である。

A.2　記述法と構文

A.2.2　指示文の構文

H201	*hpf-directive-line*	**is**	*directive-origin hpf-directive*
H202	*directive-origin*	**is**	`!HPF$`
		or	`CHPF$`
		or	`*HPF$`
H203	*hpf-directive*	**is**	*specification-directive*
		or	*executable-directive*
H204	*specification-directive*	**is**	*processors-directive*
		or	*align-directive*
		or	*distribute-directive*
		or	*inherit-directive*
		or	*template-directive*
		or	*combined-directive*
		or	*sequence-directive*
H205	*executable-directive*	**is**	*independent-directive*

制約: *hpf-directive-line* には、他の文があってはならない。

制約: *specification-directive* は、宣言構文 (*declaration-construct*) が現れてもよい場所にのみ現れてもよい。

制約: *executable-directive* は、実行構文 (*executable-construct*) が現れてもよい場所にのみ現れてもよい。

制約: *hpf-directive-line* は、プログラム単位内の周囲の形式に従って、Fortran の自由形式 (F95:3.3.1.1) または固定形式 (F95:3.3.2.1) の注釈行の規則のどちらかに従う。(F95:3.3)

H206	*specification-directive-extended*	**is**	*processors-directive*
		or	*subset-directive*
		or	*align-directive*
		or	*distribute-directive*
		or	*inherit-directive*
		or	*template-directive*
		or	*combined-directive*
		or	*sequence-directive*
		or	*dynamic-directive*
		or	*range-directive*
		or	*shadow-directive*
H207	*executable-directive-extended*	**is**	*independent-directive*
		or	*realign-directive*
		or	*redistribute-directive*
		or	*on-directive*
		or	*resident-directive*
H208	*executable-construct-extended*	**is**	*action-stmt*
		or	*case-construct*
		or	*do-construct*
		or	*if-construct*
		or	*where-construct*
		or	*on-construct*
		or	*resident-construct*
		or	*task-region-construct*

A.3 データマッピング

A.3.2 データ整列とデータ分散指示文の構文

H301	*combined-directive*	**is**	*combined-attribute-list* `::` *combined-decl-list*
H302	*combined-attribute*	**is**	`ALIGN` *align-attribute-stuff*
		or	`DISTRIBUTE` *dist-attribute-stuff*
		or	`INHERIT`
		or	`TEMPLATE`
		or	`PROCESSORS`
		or	`DIMENSION (` *explicit-shape-spec-list* `)`
H303	*combined-decl*	**is**	*hpf-entity* [`(` *explicit-shape-spec-list* `)`]
		or	*object-name*

H304 *hpf-entity* **is** *processors-name*
or *template-name*

制約: 同じ種類の *combined-attribute* を、一つの *combined-directive* 中で 2 回以上指定してはならない。

制約: `DIMENSION` 属性が *combined-directive* に現れた場合、それが適用されるすべての言語要素は、HPF の `TEMPLATE` や `PROCESSORS` 型指定子で宣言されなければならない。

A.3.3 DISTRIBUTE 指示文

H305 *distribute-directive* **is** `DISTRIBUTE` *distributee dist-directive-stuff*

H306 *dist-directive-stuff* **is** *dist-format-clause* [*dist-onto-clause*]

H307 *dist-attribute-stuff* **is** *dist-directive-stuff*
or *dist-onto-clause*

H308 *distributee* **is** *object-name*
or *template-name*

H309 *dist-format-clause* **is** (*dist-format-list*)
or * (*dist-format-list*)
or *

H310 *dist-format* **is** `BLOCK` [(*scalar-int-expr*)]
or `CYCLIC` [(*scalar-int-expr*)]
or *

H311 *dist-onto-clause* **is** `ONTO` *dist-target*

H312 *dist-target* **is** *processors-name*
or * *processors-name*
or *

制約: *distributee* である実体名 (*object-name*) は、単純な名前でなければならず、部分実体特定子や成分名 (*component-name*) であってはならない。

制約: *distributee* である実体名 (*object-name*) は、*alignee* であってはならない。

制約: *distributee* である実体名 (*object-name*) は、`POINTER` 属性をもつことはできない。

制約: *distributee* である実体名 (*object-name*) は、`TARGET` 属性をもつことはできない。

制約: *distributee* がスカラであるとき、*dist-format-list*(およびそれを囲む丸括弧) は現れてはならない。この場合、文形式の指示文は、*dist-format-clause* が “*” である場合だけが許される。

制約: *dist-format-list* が指定されたら、その長さはそれぞれの *distributee* の次元数と等しくなければならない。

制約: *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、そのプロセッサ構成の次元数と等しくなければならない。

制約: *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、そのプロセッサ構成の次元数と等しくなければならない。

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* は仮引数でなければならない。

制約: `DISTRIBUTE` 指示文の *dist-format* 中の整数式 (*scalar-int-expr*) は、すべて宣言式 (*specification-expr*) でなくてはならない。

A.3.4 ALIGN 指示文

H313	*align-directive*	**is**	`ALIGN` *alignee* *align-directive-stuff*
H314	*align-directive-stuff*	**is**	(*align-source-list*) *align-with-clause*
H315	*align-attribute-stuff*	**is**	[(*align-source-list*)] *align-with-clause*
H316	*alignee*	**is**	*object-name*
H317	*align-source*	**is**	:
		or	*
		or	*align-dummy*
H318	*align-dummy*	**is**	*scalar-int-variable*

制約: *alignee* である実体名 (*object-name*) は、単純な名前でなければならず、部分実体特定子や成分名 (*component-name*) であってはならない。

制約: *alignee* である実体名 (*object-name*) は、*distributee* であってはならない。

制約: *alignee* である実体名 (*object-name*) は、`POINTER` 属性をもつことはできない。

制約: *alignee* である実体名 (*object-name*) は、`TARGET` 属性をもつことはできない。

制約: *alignee* がスカラであるとき、*align-source-list* (およびそれを囲む丸括弧) は現れてはならない。この場合、文形式の指示文は許されない。

制約: *align-source-list* が指定されたとき、その長さは *alignee* の次元数と等しくなければならない。

制約: *align-dummy* は、名前付き変数でなければならない。

制約: 実体は、 `INHERIT` 属性と `ALIGN` 属性の両方をもつことはできない。

H319	*align-with-clause*	**is**	`WITH` *align-spec*
H320	*align-spec*	**is**	*align-target* [(*align-subscript-list*)]
		or	* *align-target* [(*align-subscript-list*)]

H321	*align-target*	**is**	*object-name*
		or	*template-name*
H322	*align-subscript*	**is**	*int-expr*
		or	*align-subscript-use*
		or	*subscript-triplet*
		or	*
H323	*align-subscript-use*	**is**	[[*int-level-two-expr*] *add-op*] *align-add-operand*
		or	*align-subscript-use add-op int-add-operand*
H324	*align-add-operand*	**is**	[*int-add-operand* *] *align-primary*
		or	*align-add-operand* * *int-mult-operand*
H325	*align-primary*	**is**	*align-dummy*
		or	(*align-subscript-use*)
H326	*int-add-operand*	**is**	*add-operand*
H327	*int-mult-operand*	**is**	*mult-operand*
H328	*int-level-two-expr*	**is**	*level-2-expr*

制約: *align-target* である実体名 (*object-name*) は、単純な名前でなければならず、部分実体特定子や成分名 (*component-name*) であってはならない。

制約: *align-target* は、 `OPTIONAL` 属性をもつことはできない。

制約: `ALIGN` 指示文の *align-spec* が “*” で始まるときは、すべての *alignee* は仮引数でなければならない。

制約: *align-directive* 中の整数式 (*int-expr*)、*int-level-two-expr*、*int-add-operand* および *int-mult-operand* は、宣言式でなくてはならない。

制約: *align-directive* 中の *align-subscript* である添字三つ組 (*subscript-triplet*) の添字 (*subscript*) や刻み幅 (*stride*) は、宣言式でなくてはならない。

制約: *align-subscript-list* では、同一の *align-dummy* は高々1 回だけ現れることができる。

制約: *align-subscript-use* 式には、*align-dummy* は高々1 回だけ現れることができる。

制約: *align-dummy* として使われた *scalar-int-variable* は、上記の文法により明確に許された場所以外では *align-spec* の中には現れてはならない。言い代えれば、一つの *align-dummy* に、*align-dummy* を含まない整数宣言式を加数や乗数として加えることでのみ、*align-subscript-use* を構成することができる。

制約: *align-subscript* 中の添字 (*subscript*) には、いかなる *align-dummy* も現れてはならない。

制約: *int-add-operand*, *int-mult-operand* および *int-level-two-expr* は、整数型でなくてはならない。

A.3.6 PROCESSORS 指示文

H329 *processors-directive* **is** `PROCESSORS` *processors-decl-list*

H330 *processors-decl* **is** *processors-name*
[(*explicit-shape-spec-list*)]

A.3.7 TEMPLATE 指示文

H331 *template-directive* **is** `TEMPLATE` *template-decl-list*

H332 *template-decl* **is** *template-name* [(*explicit-shape-spec-list*)]

A.3.8 記憶列結合と順序結合

H333 *sequence-directive* **is** `SEQUENCE` [[::] *association-name-list*]
or `NO SEQUENCE` [[::] *association-name-list*]

H334 *association-name* **is** *object-name*
or / [*common-block-name*] /

制約: いかなる有効域においても、データ実体の名前と共通ブロック名は、*sequence-directive* の中に高々一度しか現れてはならない。

制約: *association-name-list* のない *sequence-directive* は、同じ有効域の中で一つしか許されない。

A.4 副プログラム境界でのデータマッピング

A.4.4 整列

H401 *inherit-directive* **is** `INHERIT` *inheritee-list*

H402 *inheritee* **is** *object-name*

制約: *inheritee* は仮引数でなければならない。

制約: *inheritee* は *alignee* であってはならない。

制約: *inheritee* は *distributee* であってはならない。

A.5 INDEPENDENT 指示文及び関連の指示文

A.5.1 INDEPENDENT 指示文

H501 *independent-directive* **is** `INDEPENDENT` [, *new-clause*]
[, *reduction-clause*]

H502 *new-clause* **is** `NEW` (*variable-name-list*)

H503 *reduction-clause* **is** `REDUCTION` (*reduction-variable-list*)

H504 *reduction-variable* **is** *array-variable-name*
or *scalar-variable-name*
or *structure-component*

制約: *independent-directive* の後の最初の非注釈行は、*do-stmt*、*forall-stmt*、または *forall-construct* でなければならない。

制約: *independent-directive* の後の最初の非注釈行が *do-stmt* である場合、その文は *do-variable* を含む *loop-control* オプションを持たなければならない。

制約: `NEW` 節または `REDUCTION` 節がある場合、指示文の後の最初の非注釈行は *do-stmt* でなければならない。

制約: `NEW` 節または `REDUCTION` 節内に指定する *variable* 及びそれらの成分及び要素は、次のものであってはならない。

- 仮引数
- `SAVE` 属性または `TARGET` 属性を持つもの
- `COMMON` ブロックに現われるもの
- `EQUIVALENCE` 文により他の実体と記憶列結合するもの
- 参照結合したもの
- 親子結合したもの
- 親子結合により他の有効域でアクセスされるもの

制約: *reduction-variable* として現われる変数は、同じ *independent-directive* の *new-clause* 中に現われてはならず、*independent-directive* が適用される後続の *do-stmt*、*forall-stmt* 及び *forall-construct* の範囲内 (すなわち、ソース上でのループ本体部) の *new-clause* 及び *r*eduction-clause に現われてはならない。

制約: *reduction-variable* 中の *structure-component* は、*subscript-section-list* を含んではならない。

制約: *reduction-variable*[1] として現われる変数は、組込み型でなければならない。また、`CHARACTER` 型であってはならない。

---

[1]原文は *reduction-var*

H505 *reduction-stmt* **is** *variable* = *variable mult-op mult-operand*
or *variable* = *add-operand* * *variable*
or *variable* = *variable add-op add-operand*
or *variable* = *level*-2-*expr* + *variable*
or *variable* = *variable and-op and-operand*
or *variable* = *and-operand and-op variable*
or *variable* = *variable or-op or-operand*
or *variable* = *or-operand or-op variable*
or *variable* = *variable equiv-op equiv-operand*
or *variable* = *equiv-operand equiv-op variable*
or *variable* = *reduction-function* (*variable* , *expr*)
or *variable* = *reduction-function* (*expr* , *variable*)

H506 *reduction-function* **is** MAX
or MIN
or IAND
or IOR
or IEOR

制約: *reduction-stmt* 中に *variable* が 2 つ現われる場合、その 2 つは字面的に等しくなければならない。

A.6 外来プログラム単位

A.6.2 外来プログラム単位の宣言

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。

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。

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。

A.8 データマッピングの公認拡張仕様

A.8.2 拡張データマッピング指示文の属性形式の構文

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。

A.8.3 REDISTRIBUTE 指示文

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。

A.8.4 REALIGN 指示文

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。

A.8.5 DYNAMIC 指示文

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。

A.8.7 部分プロセッサへのマッピング

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。

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。

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。

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。

A.8.9 構造体成分のマッピング

H807	*distributee-extended*	**is**	*object-name*
		or	*template-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。

A.8.10 新しい分散形式

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。

A.8.11 RANGE 指示文

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。

A.8.12 SHADOW 指示文

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。

A.9 データとタスク並列に対する公認拡張

A.9.1 活動プロセッサ集合

H901 *subset-directive* **is** `SUBSET` *processors-name*

A.9.2 ON 指示文

H902 *on-directive* **is** `ON` *on-stuff*

H903 *on-stuff* **is** *home* [, *resident-clause*] [, *new-clause*]

H904 *on-construct* **is**

directive-origin block-on-directive
block
directive-origin end-on-directive

H905 *block-on-directive* **is** `ON` *on-stuff* `BEGIN`

H906 *end-on-directive* **is** `END ON`

H907 *home* **is** `HOME` (*variable*)
or `HOME` (*template-elmt*)
or (*processors-elmt*)

H908 *template-elmt* **is** *template-name* [(*section-subscript-list*)]

H909 *processors-elmt* **is** *processors-name* [(*section-subscript-list*)]

A.9.3 RESIDENT 節、指示文、構文

H910 *resident-clause* **is** `RESIDENT` *resident-stuff*

H911 *resident-stuff* **is** [(*res-object-list*)]

H912 *resident-directive* **is** `RESIDENT` *resident-stuff*

H913 *resident-construct* **is**

directive-origin block-resident-directive
block
directive-origin end-resident-directive

H914 *block-resident-directive* **is** `RESIDENT` *resident-stuff* `BEGIN`

H915 *end-resident-directive* **is** END RESIDENT

H916 *res-object* **is** *object*

A.9.4 TASK_REGION 構文

H917 *task-region-construct* **is**

directive-origin block-task-region-directive
block
directive-origin end-task-region-directive

H918 *block-task-region-directive* **is** TASK_REGION

H919 *end-task-region-directive* **is** END TASK_REGION

A.10 非同期入出力に関する公認拡張

or ASYNCHRONOUS

or ID = *scalar-default-int-variable*
or ASYNCHRONOUS

制約: ASYNCHRONOUS 指定子又は ID=指定子のどちらか一方を指定した場合、その両方を指定しなければばならない。

制約: ASYNCHROUNOUS 指定子を指定した場合、REC=指定子を指定しなければならず、かつ「書式」を指定してはならず、かつ「変数群名」を指定してはならない。

制約: ASYNCHRONOUS 指定子を指定した場合、データ転送文中のどの式においても、関数を引用してはならない。

or ID = *scalar-default-int-variable*
or PENDING = *scalar-default-logical-variable*

制約: INQUIRE 文中に FILE=指定子を指定した場合、ID=指定子や PENDING=指定子を指定してはならない。

制約: ID=指定子又は PENDING=指定子のどちらか一方を指定する場合、その両方を指定しなければばならない。

A.10.1 WAIT 文

H1001 *wait-stmt* **is** WAIT (*wait-spec-list*)

H1002 *wait-spec* **is** UNIT = *io-unit*
or ID = *scalar-default-int-expr*
or ERR = *label*
or IOSTAT = *label*

制約: *wait-spec-list* には、UNIT=指定子をちょうど 1 つ、ID=指定子をちょうど 1 つと、他の各指定子を高々1 つ、指定しなければならない。

A.11 HPF 外来機能に関する公認拡張

A.11.2 外来言語との結合仕様

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。

附属書B　構文記号の索引

この附属書は、構文規則で用いられる記号の索引を示す。H で始まる識別番号は、本 High Performance Fortran 言語仕様書の構文規則を表し、その規則の全体は附属書 A に示されている。R で始まる識別番号は、Fortran 言語規格 (“Fortran 95”) の構文規則を表す。

B.1　構文規則の左辺に現れる非終端記号

記号	定義箇所	参照箇所
action-stmt	R216	H208
add-op	R710	H323　H505
add-operand	R706	H326　H505
align-add-operand	H324	H323　H324
align-attribute-stuff	H315	H302　H801　H803
align-directive	H313	H204　H206
align-directive-stuff	H314	H313　H803
align-dummy	H318	H317　H325
align-primary	H325	H324
align-source	H317	H314　H315
align-spec	H320	H319
align-subscript	H322	H320
align-subscript-use	H323	H322　H323　H325
align-target	H321	H320
align-target-extended	H809	
align-with-clause	H319	H314　H315
alignee	H316	H313　H803　H805
alignee-extended	H808	
alignee-or-distributee	H805	H804
allocate-object	R625	
allocate-stmt	R622	
and-op	R720	H505
and-operand	R715	H505
array-constructor	R432	
array-spec	R513	H1102
assignment-stmt	R735	
association-name	H334	H333
attr-spec	R503	

attr-spec-extended	H1102	H1101
block	R801	H904 H913 H917
block-data-stmt	H607	
block-on-directive	H905	H904
block-resident-directive	H914	H913
block-task-region-directive	H918	H917
call-stmt	R1211	
case-construct	R808	H208
combined-attribute	H302	H301
combined-attribute-extended	H801	
combined-decl	H303	H301
combined-directive	H301	H204 H206
data-stmt	R532	
deallocate-stmt	R631	
directive-origin	H202	H201 H904 H913 H917
dist-attribute-stuff	H307	H302 H801 H802
dist-directive-stuff	H306	H305 H307 H802
dist-format	H310	H309
dist-format-clause	H309	H306
dist-onto-clause	H311	H306 H307
dist-target	H312	H311
distribute-directive	H305	H204 H206
distributee	H308	H305 H802 H805
distributee-extended	H807	
do-construct	R816	H208
dummy-arg	R1223	H602
dynamic-directive	H804	H206
end-function-stmt	R1220	
end-on-directive	H906	H904
end-resident-directive	H915	H913
end-subroutine-stmt	R1224	
end-task-region-directive	H919	H917
entity-decl	R504	H1101
equiv-op	R722	H505
equiv-operand	R717	H505
executable-construct	R215	
executable-construct-extended	H208	
executable-directive	H205	H203
executable-directive-extended	H207	
execution-part	R208	
explicit-shape-spec	R514	H302 H303 H330 H332 H801
expr	R723	H505

extended-dist-format	H810						
extended-dist-target	H806						
external-name	H613	H610					
extrinsic-kind-keyword	H614	H609					
extrinsic-prefix	H608	H604	H605	H606	H607		
extrinsic-spec	H609	H608					
extrinsic-spec-arg	H610	H609					
function-reference	R1210						
function-stmt	H601						
function-subprogram	R1216						
high-width	H823	H820					
home	H907	H903					
hpf-directive	H203	H201					
hpf-directive-line	H201						
hpf-entity	H304	H303					
if-construct	R802	H208					
independent-directive	H501	H205	H207				
inherit-directive	H401	H204	H206				
inheritee	H402	H401					
input-item	R914						
int-add-operand	H326	H323	H324				
int-expr	R728	H310	H322	H810	H821	H822	H823
int-level-two-expr	H328	H323					
int-mult-operand	H327	H324					
int-variable	R607	H318					
interface-body	R1205						
internal-subprogram-part	R210						
io-unit	R901	H1002					
kind-selector	R506						
label	R313	H1002					
language	H611	H610					
layout-spec	H1104	H1102					
level-2-expr	R707	H328	H505				
low-width	H822	H820					
map-to-spec	H1103	H1102					
mask-expr	R743						
model	H612	H610					
module-stmt	H606						
mult-op	R709	H505					
mult-operand	R705	H327	H505				
namelist-stmt	R544						
new-clause	H502	H501	H903				

nullify-stmt	R629		
on-construct	H904	H208	
on-directive	H902	H207	
on-stuff	H903	H902	H905
or-op	R721	H505	
or-operand	R716	H505	
output-item	R915		
pass-by-spec	H1105	H1102	
pointer-assignment-stmt	R736		
pointer-object	R630		
prefix	H603	H601	H602
prefix-spec	H604	H603	
processors-decl	H330	H329	
processors-directive	H329	H204	H206
processors-elmt	H909	H907	
program-stmt	H605		
range-attr	H815	H814	
range-attr-stuff	H813	H801	H811
range-directive	H811	H206	
range-dist-format	H816	H815	
range-distribution	H814	H813	
ranger	H812	H811	
read-stmt	R909		
realign-directive	H803	H207	
redistribute-directive	H802	H207	
reduction-clause	H503	H501	
reduction-function	H506	H505	
reduction-stmt	H505		
reduction-variable	H504	H503	
res-object	H916	H911	
resident-clause	H910	H903	
resident-construct	H913	H208	
resident-directive	H912	H207	
resident-stuff	H911	H910	H912 H914
section-subscript	R618	H806	H908 H909
sequence-directive	H333	H204	H206
shadow-attr-stuff	H819	H801	H817
shadow-directive	H817	H206	
shadow-spec	H820	H819	
shadow-target	H818	H817	
specification-directive	H204	H203	
specification-directive-extended	H206		

specification-expr	R734	
specification-part	R204	
stat-variable	R623	
stop-stmt	R840	
stride	R620	
structure-component	R614	H504 H807 H808 H809
subroutine-stmt	H602	
subscript	R617	
subscript-triplet	R619	H322
subset-directive	H901	H206
target	R737	
task-region-construct	H917	H208
template-decl	H332	H331
template-directive	H331	H204 H206
template-elmt	H908	H907
type-declaration-stmt	R501	
type-declaration-stmt-extended	H1101	
type-spec	R502	H604 H1101
variable	R601	H505 H907
wait-spec	H1002	H1001
wait-stmt	H1001	
where-construct	R739	H208
where-stmt	R738	
width	H821	H820
write-stmt	R910	

B.2 構文規則の左辺に現れない非終端記号

記号	参照箇所
access-spec	H1102
array-variable-name	H504
block-data-name	H607
char-initialization-expr	H611 H612 H613 H1103 H1104 H1105
common-block-name	H334
component-name	H807 H808 H809 H818
default-int-expr	H1002
dummy-arg-name	H601
function-name	H601
int-array	H810
intent-spec	H1102
module-name	H606

object	H916
object-name	H303 H308 H316 H321 H334 H402 H807 H808 H809 H812 H818
processors-name	H304 H312 H330 H806 H901 H909
program-name	H605
result-name	H601
subroutine-name	H602
template-name	H304 H308 H321 H332 H807 H809 H812 H908
variable-name	H502 H504

B.3 終端記号

記号	**参照箇所**
`!HPF$`	H202
`(`	H302 H303 H309 H310 H314 H315 H320 H325 H330 H332 H502 H503 H505 H601 H602 H608 H801 H806 H810 H814 H816 H819 H907 H908 H909 H911 H1001 H1102
`)`	H302 H303 H309 H310 H314 H315 H320 H325 H330 H332 H502 H503 H505 H601 H602 H608 H801 H806 H810 H814 H816 H819 H907 H908 H909 H911 H1001 H1102
`*`	H309 H310 H312 H317 H320 H322 H324 H505 H806 H810 H816
`*HPF$`	H202
`+`	H505
`,`	H501 H505 H903 H1101
`/`	H334
`:`	H317 H820
`::`	H301 H333 H802 H803 H1101
`=`	H505 H611 H612 H613 H1002
`ALIGN`	H302 H313 H801
`ALL`	H815
`ALLOCATABLE`	H1102
`BEGIN`	H905 H914
`BLOCK`	H310 H607 H810 H816
`CHPF$`	H202
`CYCLIC`	H310 H810 H816

附属書C　HPF 1.1 Subset

As part of the definition of the previous version of the High Performance Fortran language, HPF 1.1, a subset language was formally defined, based on the Fortran 77 language. The goal was to permit more rapid implementations of a useful subset of HPF that did not require full implementation of the new ANSI/ISO standard Fortran ("Fortran 90").

No subset language is defined as part of the current version, HPF 2.0. This Annex is included in the HPF 2.0 language document as a convenient summary of the HPF 1.1 Subset, which has served as a minimum requirement for HPF implementations.

C.1 Fortran 90 Features in the HPF 1.1 Subset

The features of the HPF 1.1 subset languages are listed below. For reference, the section numbers from the Fortran 90 standard are given along with the related syntax rule numbers:

- All FORTRAN 77 standard conforming features, except for storage and sequence association.
- The Fortran 90 definitions of MIL-STD-1753 features:
  - `DO WHILE` statement (8.1.4.1.1 / R821)
  - `END DO` statement (8.1.4.1.1 / R825)
  - `IMPLICIT NONE` statement (5.3 / R540)
  - `INCLUDE` line (3.4)
  - scalar bit manipulation intrinsic procedures: `IOR`, `IAND`, `NOT`, `IEOR`, `ISHFT`, `ISHFTC`, `BTEST`, `IBSET`, `IBCLR`, `IBITS`, `MVBITS` (13.13)
  - binary, octal and hexadecimal constants for use in `DATA` statements (4.3.1.1 / R407 and 5.2.9 / R533)
- Arithmetic and logical array features:
  - array sections (6.2.2.3 / R618–621)
    - subscript triplet notation (6.2.2.3.1)
    - vector-valued subscripts (6.2.2.3.2)
  - array constructors limited to one level of implied `DO` (4.5 / R431)
  - arithmetic and logical operations on whole arrays and array sections (2.4.3, 2.4.5, and 7.1)

- array assignment (2.4.5, 7.5, 7.5.1.4, and 7.5.1.5)
- masked array assignment (7.5.3)
  * WHERE statement (7.5.3 / R738)
  * block WHERE . . . ELSEWHERE construct (7.5.3 / R739)
- array-valued external functions (12.5.2.2)
- automatic arrays (5.1.2.4.1)
- ALLOCATABLE arrays and the ALLOCATE and DEALLOCATE statements (5.1.2.4.3, 6.3.1 / R622, and 6.3.3 / R631)
- assumed-shape arrays (5.1.2.4.2 / R516)

• Intrinsic procedures:

The list of intrinsic functions and subroutines below is a combination of (a) routines which are entirely new to Fortran and (b) routines that have always been part of Fortran, but have been extended here to new argument and result types. The new or extended definitions of these routines are part of the subset. If a FORTRAN 77 routine is not included in this list, then only the original FORTRAN 77 definition is part of the subset.

For all of the intrinsics that have an optional argument DIM, only actual argument expressions for DIM that are initialization expressions are part of the subset. The intrinsics with this constraint are marked with †in the list below.

- the argument presence inquiry function: PRESENT (13.10.1)
- all the numeric elemental functions: ABS, AIMAG, AINT, ANINT, CEILING, CMPLX, CONJG, DBLE, DIM, DPROD, FLOOR, INT, MAX, MIN, MOD, MODULO, NINT, REAL, SIGN (13.10.2)
- all mathematical elemental functions: ACOS, ASIN, ATAN, ATAN2, COS, COSH, EXP, LOG, LOG10, SIN, SINH, SQRT, TAN, TANH (13.10.3)
- all the bit manipulation elemental functions : BTEST, IAND, IBCLR, IBITS, IBSET, IEOR, IOR, ISHFT, ISHFTC, NOT (13.10.10)
- all the vector and matrix multiply functions: DOT_PRODUCT, MATMUL (13.10.13)
- all the array reduction functions: ALL†, ANY†, COUNT†, MAXVAL†, MINVAL†, PRODUCT†, SUM†(13.10.14)
- all the array inquiry functions: ALLOCATED, LBOUND†, SHAPE, SIZE†, UBOUND†(13.10.15)
- all the array construction functions: MERGE, PACK, SPREAD†, UNPACK (13.10.16)
- the array reshape function: RESHAPE (13.10.17)
- all the array manipulation functions: CSHIFT†, EOSHIFT†, TRANSPOSE (13.10.18)
- all array location functions: MAXLOC†, MINLOC†(13.10.19)

- all intrinsic subroutines: DATE_AND_TIME, MVBITS, RANDOM_NUMBER, RANDOM_SEED, SYSTEM_CLOCK (3.11)

- Declarations:
  - Type declaration statements, with all forms of *type-spec* except *kind-selector* and TYPE(type-name), and all forms of *attr-spec* except *access-spec*, TARGET, and POINTER. (5.1 / R501-503, R510)
  - attribute specification statements: ALLOCATABLE, INTENT, OPTIONAL, PARAMETER, SAVE (5.2)

- Procedure features:
  - INTERFACE blocks with no *generic-spec* or *module-procedure-stmt* (12.3.2.1)
  - optional arguments (5.2.2)
  - keyword argument passing (12.4.1 /R1212)

- Syntax improvements:
  - long (31 character) names (3.2.2)
  - lower case letters (3.1.7)
  - use of "_" in names (3.1.3)
  - "!" initiated comments, both full line and trailing (3.3.2.1)

C.2 HPF 1.1 Directives and Language Extensions in the HPF 1.1 Subset

The following HPF 1.1 directives and language extensions to Fortran 90 were included in the HPF 1.1 Subset:

- The basic data distribution and alignment directives: ALIGN, DISTRIBUTE, PROCESSORS. and TEMPLATE.
- The *forall-statement* (but not the *forall-construct*).
- The INDEPENDENT directive.
- The SEQUENCE and NO SEQUENCE directives.
- The system inquiry intrinsic functions NUMBER_OF_PROCESSORS and PROCESSORS_SHAPE.
- The computational intrinsic functions ILEN, and the HPF extended Fortran intrinsics MAXLOC and MINLOC, with the restriction that any actual argument expression corresponding to an optional DIM argument must be an initialization expression.

For a discussion of the rationale by which features were chosen for the HPF 1.1 Subset, please consult HPF Language Specification Version 1.1.

附属書D Previous HPFF Acknowledgments

The current HPF 2.0 document would not have been possible without the contributions of the previous series of HPFF meetings. Following are the acknowlegements for those efforts.

D.1 HPFF Acknowledgments

Technical development for HPF 1.0 was carried out by subgroups, and was reviewed by the full committee. Many people served in positions of responsibility:

- Ken Kennedy, Convener and Meeting Chair;
- Charles Koelbel, Executive Director and Head of the FORALL Subgroup;
- Mary Zosel, Head of the Fortran 90 and Storage Association Subgroup;
- Guy Steele, Head of the Data Distribution Subgroup;
- Rob Schreiber, Head of the Intrinsics Subgroup;
- Bob Knighten, Head of the Parallel I/O Subgroup;
- Marc Snir, Head of the Extrinsics Subgroup;
- Joel Williamson and Marina Chen, Heads of the Subroutine Interface Subgroup; and
- David Loveman, Editor.

Geoffrey Fox convened the first HPFF meeting with Ken Kennedy and later led a group to develop benchmarks for HPF. Clemens-August Thole organized a group in Europe and was instrumental in making this an international effort. Charles Koelbel produced detailed meeting minutes that were invaluable to subgroup heads in preparing successive revisions to the draft proposal. Guy Steele developed LaTeX macros for a variety of tasks, including formatting BNF grammar, Fortran code and pseudocode, and commentary material; the document would have been much less aesthetically pleasing without his efforts.

Many companies, universities, and other entities supported their employees' attendance at the HPFF meetings, both directly and indirectly. The following organizations were represented at two or more meetings by the following individuals (not including those present at the first HPFF meeting in January of 1992, for which there is no accurate attendee list):
Alliant Computer Systems Corporation .... David Reese

Amoco Production Company .... Jerrold Wagener, Rex Page
Applied Parallel Research .... John Levesque, Rony Sawdayi, Gene Wagenbreth
Archipel .... Jean-Laurent Philippe
CONVEX Computer Corporation .... Joel Williamson
Cornell Theory Center .... David Presberg
Cray Research, Inc. .... Tom MacDonald, Andy Meltzer
Digital Equipment Corporation .... David Loveman
Fujitsu America .... Siamak Hassanzadeh, Ken Muira
Fujitsu Laboratories .... Hidetoshi Iwashita
GMD-I1.T, Sankt Augustin .... Clemens-August Thole
Hewlett Packard .... Maureen Hoffert, Tin-Fook Ngai, Richard Schooler
IBM .... Alan Adamson, Randy Scarborough, Marc Snir, Kate Stewart
Institute for Computer Applications in Science & Engineering .... Piyush Mehrotra
Intel Supercomputer Systems Division .... Bob Knighten
Lahey Computer .... Lev Dyadkin, Richard Fuhler, Thomas Lahey, Matt Snyder
Lawrence Livermore National Laboratory .... Mary Zosel
Los Alamos National Laboratory .... Ralph Brickner, Margaret Simmons
Louisiana State University .... J. Ramanujam
MasPar Computer Corporation .... Richard Swift
Meiko, Inc. .... James Cownie
nCUBE, Inc. .... Barry Keane, Venkata Konda
Ohio State University .... P. Sadayappan
Oregon Graduate Institute of Science and Technology .... Robert Babb II
The Portland Group, Inc. .... Vince Schuster
Research Institute for Advanced Computer Science .... Robert Schreiber
Rice University .... Ken Kennedy, Charles Koelbel
Schlumberger .... Peter Highnam
Shell .... Don Heller
State University of New York at Buffalo .... Min-You Wu
SunPro and Sun Microsystems .... Prakash Narayan, Douglas Walls
Syracuse University .... Alok Choudhary, Tom Haupt
TNO-TU Delft .... Edwin Paalvast, Henk Sips
Thinking Machines Corporation .... Jim Bailey, Richard Shapiro, Guy Steele
United Technologies Corporation .... Richard Shapiro
University of Stuttgart .... Uwe Geuder, Bernhard Woerner, Roland Zink
University of Southampton .... John Merlin
University of Vienna .... Barbara Chapman, Hans Zima
Yale University .... Marina Chen, Aloke Majumdar

Many people contributed sections to the final language specification and HPF Journal of Development, including Alok Choudhary, Geoffrey Fox, Tom Haupt, Maureen Hoffert, Ken Kennedy, Robert Knighten, Charles Koelbel, David Loveman, Piyush Mehrotra, John

Merlin, Tin-Fook Ngai, Rex Page, Sanjay Ranka, Robert Schreiber, Richard Shapiro, Marc Snir, Matt Snyder, Guy Steele, Richard Swift, Min-You Wu, and Mary Zosel. Many others contributed shorter passages and examples and corrected errors.

Because public input was encouraged on electronic mailing lists, it is impossible to identify all who contributed to discussions; the entire mailing list was over 500 names long. Following are some of the active participants in the HPFF process not mentioned above:

N. Arunasalam	Werner Assmann	Marc Baber
Babak Bagheri	Vasanth Bala	Jason Behm
Peter Belmont	Mike Bernhardt	Keith Bierman
Christian Bishof	John Bolstad	William Camp
Duane Carbon	Richard Carpenter	Brice Cassenti
Doreen Cheng	Mark Christon	Fabien Coelho
Robert Corbett	Bill Crutchfield	J. C. Diaz
James Demmel	Alan Egolf	Bo Einarsson
Pablo Elustondo	Robert Ferrell	Rhys Francis
Hans-Hermann Frese	Steve Goldhaber	Brent Gorda
Rick Gorton	Robert Halstead	Reinhard von Hanxleden
Hiroki Honda	Carol Hoover	Steven Huss-Lederman
Ken Jacobsen	Elaine Jacobson	Behm Jason
Alan Karp	Ronan Keryell	Anthony Kimball
Ross Knippe	Bruce Knobe	David Kotz
Ed Krall	Tom Lake	Peter Lawrence
Bryan Lawver	Bruce Leasure	Stewart Levin
David Levine	Theodore Lewis	Woody Lichtenstein
Ruth Lovely	Doug MacDonald	Raymond Man
Stephen Mark	Philippe Marquet	Jeanne Martin
Oliver McBryan	Charlie McDowell	Michael Metcalf
Charles Mosher	Len Moss	Lenore Mullin
Yoichi Muraoka	Bernie Murray	Vicki Newton
Dale Nielsen	Kayutov Nikolay	Steve O'Neale
Jeff Painter	Cherri Pancake	Harvey Richardson
Bob Riley	Kevin Robert	Ron Schmucker
J.L. Schonfelder	Doug Scofield	David Serafini
G.M. Sigut	Anthony Skjellum	Niraj Srivastava
Paul St.Pierre	Nick Stanford	Mia Stephens
Jaspal Subhlok	Xiaobai Sun	Hanna Szoke
Bernard Tourancheau	Anna Tsao	Alex Vasilevsky
Stephen Vavasis	Arthur Veen	Brian Wake
Ji Wang	Karen Warren	D.C.B. Watson
Matthijs van Waveren	Robert Weaver	Fred Webb
Stephen Whitley	Michael Wolfe	Fujio Yamamoto
Marco Zagha		

The following organizations made the language draft available by anonymous FTP access and/or mail servers: AT&T Bell Laboratories, Cornell Theory Center, GMD-I1.T (Sankt Augustin), Oak Ridge National Laboratory, Rice University, Syracuse University, and Thinking Machines Corporation. These outlets were instrumental in distributing the document.

The High Performance Fortran Forum also received a great deal of volunteer effort in nontechnical areas. Theresa Chatman and Ann Redelfs were responsible for most of the meeting planning and organization, including the first HPFF meeting, which drew over 125 people. Shaun Bonton, Rachele Harless, Rhonda Perales, Seryu Patel, and Daniel Swint helped with many logistical details. Danny Powell spent a great deal of time handling the financial details of the project. Without these people, it is unlikely that HPF would have been completed.

HPFF operated on a very tight budget (in reality, it had no budget when the first meeting was announced). The first meeting in Houston was entirely financed from the conferences budget of the Center for Research on Parallel Computation, an NSF Science and Technology Center. DARPA and NSF have supported research at various institutions that have made a significant contribution towards the development of High Performance Fortran. Their sponsored projects at Rice, Syracuse, and Yale Universities were particularly influential in the HPFF process. Support for several European participants was provided by ESPRIT through projects P6643 (PPPE) and P6516 (PREPARE).

D.2 HPFF94 Acknowledgments

The HPF 1.1 version of the document was prepared during the HPFF94 series of meetings. A number of people shared technical responsibilities for the activities of the HPFF94 meetings:

- Ken Kennedy, Convener and Meeting Chair;
- Mary Zosel, Executive Director and head of CCI Group 2;
- Richard Shapiro, Head of CCI Group 1;
- Ian Foster, Head of Tasking Subgroup;
- Alok Choudhary, Head of Parallel I/O Subgroup;
- Chuck Koelbel, Head of Irregular Distributions Subgroup;
- Rob Schreiber, Head of Implementation Subgroup;
- Joel Saltz, Head of Benchmarks Subgroup;
- David Loveman, Editor, assisted by Chuck Koelbel, Rob Schreiber, Guy Steele, and Mary Zosel, section editors.

Attendence at the HPFF94 meetings included the following people from organizations that were represented two or more times.

Don Heller ... Ames Laboratory
Jerrold Wagener ... Amoco Production Company
John Levesque ... Applied Parallel Research
Ian Foster ... Argonne National Laboratory
Terry Pratt ... CESDIS/NASA Goddard
Jim Cowie ... Cooperating Systems
Andy Meltzer, Jon Steidel ... Cray Research, Inc.
David Loveman ... Digital Equipment Corporation
Bruce Olsen ... Hewlett Packard
E. Nunohiro, Satoshi Itoh ... Hitachi
Henry Zongaro ... IBM
Piyush Mehrotra ... Institute for Computer Applications in Science & Engineering
Bob Knighten, Roy Touzeau ... Intel SSD
Mary Zosel, Bor Chan, Karen Warren ... Lawrence Livermore National Laboratory
Ralph Brickner ... Los Alamos National Laboratory
J. Ramanujam ... Louisiana State University
Paula Vaughan, Donna Reese ... Mississippi State University and NSF ERC
Shoichi Sakon, Yoshiki Seo ... NEC
P. Sadayappan, Chua-Huang Huang ... Ohio State University
Andrew Johnson ... OSF Research Institute
Chip Rodden, Jeff Vanderlip ... Pacific Sierra Research
Larry Meadows, Doug Miles ... The Portland Group, Inc.
Robert Schreiber ... Research Institute for Advanced Computer Science
Ken Kennedy, Charles Koelbel ... Rice University
Ira Baxter ... Schlumberger
Alok Choudhary ... Syracuse University
Guy Steele ... Thinking Machines Corporation, Sun Microsystems
Richard Shapiro ... Thinking Machines Corp., Silicon Graphics Inc.
Scott Baden, Val Donaldson ... University of California, San Diego
Robert Babb ... University of Denver
Joel Saltz, Paul Havlak ... University of Maryland
Nicole Nemer-Preece ... University of Missouri-Rolla
Hans Zima, Siegfried Benkner, Thomas Fahringer ... University of Vienna

An important activity of HPFF94 was the processing of the many items submitted for comment and interpretation which led to the HPF 1.1 update of the language document. A special acknowlegement goes to Henry Zongaro, IBM, for many thoughtful questions exposing dark corners of language design that were previously overlooked, and to Guy Steele, Thinking Machines/Sun Microsystems for his analysis of, and solutions for some of the thornier issues discussed. And general thanks to the people who submitted comments and interpretation requests, including:

David Loveman, Michael Hennecke, James Cownie, Adam Marshall, Stephen Ehrlich, Mary Zosel, Matt Snyder, Larry Meadows, Dick Hendrickson, Dave Watson, John Merlin, Vasanth Bala, Paul.Wesson, Denis.Hugli, Stanly Steinberg, Henk Sips, Henry Zongaro, Eiji_Nunohiro, Jens Bloch Helmers, Rob Schreiber, David B. Serafini, and Allan Knies.

Other special mention goes to Chuck Koelbel at Rice University for continued maintenance of the HPFF mailing lists, to Donna Reese and staff at Mississippi State University for establishing and maintaining a WWW home-page for HPFF, and to the University of Maryland for establishing a benchmark FTP site.

Theresa Chatman and staff at Rice University were responsible for meeting planning and organization and Danny Powell continued to handle financial details of the project.

HPFF94 received direct support for research and administrative activities from grants from ARPA, DOE, and NSF.

関連図書

[1] American National Standards Institute, Inc., 1430 Broadway, New York, NY. *American National Standard Programming Language FORTRAN*, ANSI X3.9-1978, approved April 3, 1978.

[2] American National Standards Institute, Inc., 1430 Broadway, New York, NY. *American National Standard for Information Systems Programming Language FORTRAN*, S8 (X3.9-198x) Revision of X3.9-1978, Draft S8, Version 104, April 1987.

[3] High Performance Fortran Forum. *High Performance Fortran Language Specification* Scientific Programming, 2,1, 1993. Also published as: CRPC-TR92225, Center for Research on Parallel Computation, Rice University, Houston, TX, 1992 (revised May. 1993). Also published as: Fortran Forum, 12,4, Dec. 1993 and 13,2, June 1994.

[4] High Performance Fortran Forum. *High Performance Fortran Language Specification, version 1.0* CRPC-TR92225, Center for Research on Parallel Computation, Rice University, Houston, TX, 1992 (revised May. 1993).

[5] US Department of Defense. *Military Standard, MIL-STD-1753: FORTRAN, DoD Supplement to American National Standard X3.9-1978*, November 9, 1978.

附属書E Policy and Mechanism for Recognized Extrinsic Interfaces

HPF defines certain extrinsics such as `HPF_LOCAL`, and `HPF_SERIAL` as interfaces that HPFF believes are useful to the HPF community. But there are many more such extrinsic interfaces beyond those maintained by HPFF. HPFF has a adopted a policy of formally recognizing certain extrinsic interface definitions, where the interface, and its addition to the HPF document is considered to be a service to the HPF community. Examples are language bindings to HPF or library packages.

E.1 Extrinsic Policy

To be considered for HPFF recognition, a proposed extrinsic must demonstrate the following things. It should be noted, however, that meeting these criteria does not guarentee acceptance of a proposed interface by HPFF.

- conformance to HPF rules for calling extrinsics,
- significant new functionality,
- existing practice such as users, implementations, etc.,
- institutional backing with evidence of ongoing support,
- coherent documentation,
- non-proprietary interface definition, and
- copyright goes to HPFF for interface, with permission to use (royalty free).

If a proposed extrinsic is accepted by HPFF, then:

- HPFF will recogize the interface and reference it in documentation, but HPFF does not assume responsibility for the extrinsic or its interface.
- The sponsor of the extrinsic must continue to conform to the HPF interface rules for extrinsics. The interface HPFF approves must not change without HPFF approval.
- The sponsor must assume responsibility for any CCI requests concernting the extrinsic.

A list of recognized extrinsic interfaces will be included in HPF documentation, with the following guidelines:

- There should be a single page introduction to the extrinsic which contains:
  - the name of the extrinsic,
  - a brief abstract of functionality,
  - a brief and informal description of the interface,
  - information about platform and system availability, and
  - reference and contacts for formal documentation, continued responsibility, and additional information (e.g. compiler availability).
- There should be about two pages with short examples of usage.
- A short paper with the formal definition of the interface and an informal description of the functionality of the extrinsic.

E.2 Extrinsic Interface Mechanism

The HPF www-home page will have instructions for submission of an extrinsic interface. For HPFF consideration, the sponsor prepares a proposal that includes:

- a statement of what significant new functionality is provided,
- a description of existing practice,
- a statement of institutional backing with evidence of ongoing support,
- a copy of the complete documentation or a reference to an online version of the documentation,
- a draft of the text (described above) that would be included in the HPFF documentation, and
- a statement justifying the claim that the interface follows HPF conventions for calling extrinsics.

If the proposed extrinsic interface is approved by HPFF, the sponsor then submits:

- a formal statement for HPFF records that the interface definition is non-proprietary and that the copyright of the interface belongs HPFF,
- the formal contact for CCI and continued maintenance of the interface, and
- a copy of the interface documentation formatted for HPFF use, including a copy in the current document and web mark-up languages.

附属書F　HPF_CRAFT

HPF_CRAFT is a hybrid language, combining an SPMD execution model with high performing HPF features. The model combines the multi-threaded execution of HPF_LOCAL and the HPF syntax. The goal of HPF_CRAFT is to attain the potential performance of an SPMD programming model with access to HPF features and a well-defined extrinsic interface to HPF.

F.1 Introduction

HPF_CRAFT is a hybrid language, combining an SPMD execution model with high performing and portable HPF features. The model combines the multi-threaded execution of HPF_LOCAL and the HPF syntax and features. The goal of HPF_CRAFT is to attain the potential performance of an SPMD programming model with access to HPF features and a well-defined extrinsic interface to HPF. It is built on top of the HPF_LOCAL extrinsic environment.

SPMD features and a multi-threaded model allow the user to take advantage of the performance and opportunity for low level access of a more general purpose programming model. Including HPF data distribution features gives the programmer access to high performing aspects of both models, but with the added responsibility of working with a more low-level execution model. HPF_CRAFT is best suited for platforms that support one way communication features, but is consistent with HPF and easily targeted for platforms that have HPF and can support SPMD programming styles.

The HPF features included in HPF_CRAFT are a subset of the full HPF language chosen for their performance and their broad portability and ease of use. HPF_CRAFT contains additional features to support SPMD programming styles. There are some differences from HPF, however. For example, I/O causes differences; in HPF_CRAFT different processors are allowed to read from different files at the same time, in HPF the processors must all read from the same file. The differences in the models are principally caused by the multi-threaded execution model and the introduction of HPF_LOCAL data rules.

HPF_CRAFT allows for the notion of *private data*. Data defaults to a mapping in which data items are allocated so that each processor has a unique copy. The values of the individual data items and the flow of control may vary from processor to processor within HPF_CRAFT. This behavior is consistent with the behavior of HPF_LOCAL. In HPF_CRAFT a processor may be individually named and code executed based upon which processor it is executing on. HPF_CRAFT also allows for the notion of *private loops*. A private loop is executed in entirety by each processor.

The rules governing the interface to HPF_CRAFT subprograms are similar to those for the HPF_LOCAL interface. Dummy arguments use a hybrid of the interfaces between HPF and itself and that of HPF and HPF_LOCAL. Explicitly mapped dummy arguments behave just as they do in HPF, while default (private) dummy arguments use the HPF_LOCAL calling convention.

HPF_CRAFT will be initially made available on Cray MPP systems and may also be available on Cray vector architectures. Future versions of HPF_CRAFT are possible on other vendor's architectures as well.

HPF_CRAFT is being implemented for Cray Research by The Portland Group, Inc. For Cray systems, HPF_CRAFT may be obtained through the Cray Research Inc. Orderdesk,

Cray Research Inc.
orderdsk@cray.com
(612) 683-5907

Additional formal documentation, requests, and suggestions can be made to

The Portland Group
9150 SW Pioneer Ct., Suite H
Wilsonville, OR 97070
(503) 682-2806
trs@pgroup.com

F.2 Examples of Use

HPF_CRAFT is intended for use in circumstances where greater control and performance are desired for MIMD style architectures. Since data may be declared to be private, local control is made more available and since processor information is available message passing and direct memory access programming styles can be seamlessly integrated with explicitly mapped data.

The following examples show some of the capabilities of HPF_CRAFT that are different from those of HPF. Others, such as integrated message passing and synchronization primitives are not shown. Much of HPF can also be used within HPF_CRAFT.

Example 1 illustrates the difference between the default distribution for data and the distribution of mapped data.

```
!  Example 1

      INTEGER PRIVATE_A(100, 20), PRIVATE_B(12, 256), PRIVATE_C
      INTEGER MAPPED_A(100, 20), MAPPED_B(12, 256), MAPPED_C
!HPF$  DISTRIBUTE MAPPED_A(BLOCK, BLOCK), MAPPED_B(BLOCK, *), MAPPED_C
```

In the above example, given 8 processors, there would be 8 * 100 * 20 (or 16,000) elements in the array PRIVATE_A. Each processor contains an entire array named PRIVATE_A. The elements of PRIVATE_A on processor 1 cannot be referenced using implicit syntax by any other processor. There are only 100 * 20 (or 2000) elements of array MAPPED_A, however, and these elements are distributed about the machine in a (BLOCK, BLOCK) fashion.

The difference between the PRIVATE_A declaration in HPF_CRAFT and that in HPF is the most instructive. In HPF_CRAFT each processor contains one copy of the array, and the values of the elements of the array may vary from processor to processor. HPF implementations are permitted to make one copy of the array per processor the default, but the values of these copies must remain coherent across all processors. In HPF there is no way to write a conforming program in which different processors have different values for the same array.

Example 2 shows the usefulness of the ON clause for the INDEPENDENT loop as well as giving an example of how private data may be used.

```
!  Example 2

      PRIVATE_C = 0
!HPF$  INDEPENDENT (I, J) ON MAPPED_B(I, J)
      DO J=1,256
         DO I=1,12
            MAPPED_B(I, J) = MAPPED_B(I, J) + 5
            PRIVATE_C = PRIVATE_C + MAPPED_B(I, J)
         ENDDO
      ENDDO
```

In this example, each iteration is executed on the processor containing the data that is mapped to it. The user was allowed to specify this.

In addition, the private variable PRIVATE_C is used to compute a total for each processor. At the end of execution of the loop, the values of PRIVATE_C may be different on each processor depending upon the values in the elements of the array on each processor. This data may be used as is, or it can be quickly summed using a barrier or an ATOMIC UPDATE.

Example 3 shows the final total value being combined into the variable MAPPED_C whose value is available to all processors.

```
!  Example 3

      MAPPED_C = 0
!HPF$  ATOMIC UPDATE
      MAPPED_C = MAPPED_C + PRIVATE_C
```

Example 4 shows how the language allows private data to vary from processor to processor.

```
!  Example 4

      IF (MY_PE() .EQ. 5) THEN
         PRIVATE_C =  some-big-expression
      ENDIF
```

In this example, `PRIVATE_C` on processor 5 will have the result of *some-big-expression*. Each processor can do distinctly different work and communicate through mapped data.

The code fragment in Example 5 is from an application and shows a few features of the language.

```
!  Example 5

!HPF$ GEOMETRY G(*, CYCLIC)
      REAL FX(100,100), FY(100,100), FZ(100,100)
!HPF$ DISTRIBUTE (G) ::  FX,FY,FZ
      REAL FXP(100,16,100), FYP(100,16,100)
!HPF$ DISTRIBUTE FXP(*,*, BLOCK) FYP(*,*, BLOCK)
      INTEGER CELL, ATOM, MAP(1000), NACELL(1000)

!HPF$ INDEPENDENT (CELL) ON FX(1,CELL)
      DO CELL=1,100
         JCELL0 = 16*(CELL-1)
         DO NABOR = 1, 13
            JCELL = MAP(JCELL0+NABOR)
            DO ATOM=1, NACELL(CELL)
               FX(ATOM, CELL) = FX(ATOM, CELL) + FXP(ATOM, NABOR, JCELL)
               FY(ATOM, CELL) = FY(ATOM, CELL) + FYP(ATOM, NABOR, JCELL)
            ENDDO
         ENDDO
      ENDDO
```

The `GEOMETRY` directive allows the user to generically specify a mapping and use it to apply to many arrays (they need not have the same extents.)

Example 5 has a single `INDEPENDENT` loop which is the outer loop. It executes 100 iterations total. Within this loop the private value of `JCELL0` is set for each processor (ensuring that it is a local computation everywhere.) Nested inside the `INDEPENDENT` loop is a private loop; this loop executes 13 times *per* processor. Inside this loop `JCELL` is computed locally on each processor, minimizing unnecessary communication. Finally the innermost loop is also private.

F.3 External Interface

This section describes the behavior when an HPF_CRAFT routine is called from HPF.

The calling convention and argument passing rules for HPF_CRAFT are a hybrid of those for HPF calling HPF_LOCAL and HPF calling HPF. Explicit interfaces are required. Where dummy arguments are private (default) storage, the HPF calling HPF_LOCAL conventions are used. Where dummy arguments are explicitly mapped, the calling convention matches HPF calling HPF.

There are a number of constraints on HPF_CRAFT routines that are called from HPF. The following is a list of restrictions placed on HPF_CRAFT routines called from HPF:

- Recursive HPF_CRAFT routines cannot be called from HPF.
- HPF_CRAFT routines called from HPF may only enter the routine at a single place (no alternate entries).
- An HPF_CRAFT supprogram may not be invoked directly or indirectly from within the body of a `FORALL` construct or within the body of an `INDEPENDENT DO` loop that is inside an HPF program.
- The attributes (type, kind, rank, optional, intent) of the dummy arguments in a supprogram called by HPF must match the attributes of the corresponding dummy arguments in the explicit interface.
- A dummy argument of an HPF_CRAFT supprogram called by HPF
  - must not be a procedure name.
  - must not have the `POINTER` attribute.
  - must not be sequential, unless it is also `PE_PRIVATE`.
  - must have assumed shape even when it is explicit shape in the interface.
  - if scalar, it must be mapped so that each processor has a copy of the argument.
- The default mapping of scalar dummy arguments and of scalar function results when an HPF program calls an HPF_CRAFT routine is that it is replicated on each processor.

If a dummy argument of an `EXTRINSIC('HPF_CRAFT')` routine interface block is an array and the dummy argument of the HPF_CRAFT supprogram has the default private mapping, then the corresponding dummy argument in the specification of the HPF_CRAFT procedure must be an array of the same rank, type, and type parameters. When the extrinsic procedure is invoked, the dummy argument is associated with the local array that consists of the subgrid of the global array that is stored locally.

If the dummy argument of the HPF_CRAFT supprogram is explicitly mapped, it must have the same mapping as the dummy argument of the EXTRINSIC('HPF_CRAFT') supprogram. Note that this restriction does not require actual and dummy arguments to match and is no more stringent than saying that mappings of dummy arguments in interface blocks must match those in the actual routine.

F.4 Execution Model

HPF_CRAFT is built upon the fundamental execution model of HPF_LOCAL, augmented with data mapping and work distribution features from HPF. It is also augmented with explicit low-level control features, many taken from Cray Research's CRAFT language.

In HPF_CRAFT there is a single task on each processor and all tasks begin executing in parallel, with data defaulting to a private distribution, the same default distribution used in HPF_LOCAL. Each processor gets a copy of the data storage unless specified otherwise by the user. Consequently I/O works identically to I/O in HPF_LOCAL and message passing libraries are easily integrated.

Simply stated, the execution model is that of HPF_LOCAL.

To provide correct behavior when explicitly mapped data is involved, this model defines implicit barrier points at which the execution model requires that all processors must stop and wait for the execution of all other processors before continuing. These barriers add additional semantics to the HPF_LOCAL behavior. An implementation may remove any of these barriers that are deemed unnecessary, but *every* processor must participate in the barriers at each one of these points.

The points where there are implicit barriers are conceptually after those instances in which the processors in the HPF_CRAFT program are executing cooperatively, as if in an HPF program (e.g., after an INDEPENDENT loop). An HPF_CRAFT program treats operations on explicitly mapped objects as if they were operations in an HPF program and it treates operations on private data as if they were executed within the HPF_LOCAL framework. It is occasionally useful for an advanced programmer to indicate to the compilation system where barriers are not needed; HPF_CRAFT has syntax to allow this capability.

F.5 HPF_CRAFT Functional Summary

HPF_CRAFT contains a number of features not available in HPF, and restricts the usage of many of the features currently available. The following is a concise list of the differences.

- INDEPENDENT has been extended to better support an ON clause.
- There are new rules defining the interaction of explicitly mapped and private data.
- Parallel inquiry intrinsics IN_PARALLEL() and IN_INDEPENDENT() have been added.
- Serial regions (MASTER / END MASTER) have been added.
- Explicit synchronization primitives are provided.

- The `ATOMIC UPDATE`, `SYMMETRIC`, and `GEOMETRY` directives have been added.
- Many other compiler information directives have been added to assist the compiler in producing good quality code.

F.5.1 Data Mapping Features

Data mapping features provided are those that have been found useful most often. When data is explicitly mapped, only one copy of the data storage is created unless the explicit mapping directs otherwise. The value of explicitly mapped replicated data items must be consistent between processors as is the case in HPF. Storage and sequence association for explicitly mapped arrays is not guaranteed in HPF_CRAFT. For private data, storage and sequence association follows the Fortran 90 rules.

A new directive is included for completeness: `PE_PRIVATE`, which specifies that the data should conform to the default behavior. The values of private varaibles may vary on different processors.

F.5.2 Subprogram Interfaces

The behavior and requirements of an HPF_CRAFT program at subprogram interfaces may be divided into three cases. Each case is also available using some combination of HPF and HPF_LOCAL. For dummy arguments that are explicitly mapped, the behavior is identical to that of HPF. All processors must cooperate in a subprogram invocation that remaps or explicitly maps data. In other words, if an explicit interface is required (by the HPF rules) or the subprogram declares explicitly mapped data, the subprogram must be called on all processors. Processors need not cooperate if there are only reads to non-local data. The `INHERIT` attribute may only be applied to explicitly mapped data.

Data that has the default private mapping (case two) the behavior of an HPF_CRAFT subprogram at subprogram interfaces is identical to that of HPF_LOCAL. Data is passed individually on every processor and the processors need not interact in any way.

When a subprogram is passed actual arguments that are a combination of both explicitly mapped data and private data, the explicitly mapped data follows the HPF rules and the private data follows the HPF_LOCAL rules.

In case three, the user has the option of passing data with explicitly mapped actual arguments to dummy arguments that are not explicitly mapped (i.e., private.) The mapping rules for this data are identical to the mapping rules when HPF calls an HPF_LOCAL subprogram. The data remains "in-place." All HPF arrays are logically carved up into pieces; the HPF_CRAFT procedure executing on a particular physical processor sees an array containing just those elements of the global array that are mapped to that physical processor. There is implicit barrier synchronization after an `INDEPENDENT` loop. Transfer of control into or out of an `INDEPENDENT` loop is prohibited.

Finally, it is undefined behavior when an actual argument is private and the dummy argument is explicitly mapped. A definition could be supplied for this interaction, but it is the same solution that one might propose for a calling sequence when HPF_LOCAL subprograms call HPF subprograms.

F.5.3 The INDEPENDENT Directive

The `INDEPENDENT` directive is part of HPF_CRAFT with the same semantics as in HPF. However, within `INDEPENDENT` loops the values of private data may vary from processor to processor. `INDEPENDENT` applied to `FORALL` has identical syntax and semantics as in HPF.

An HPF independent loop optionally may have a `NEW` clause. The `NEW` clause is not required by HPF_CRAFT for default (not explicitly mapped) data. In HPF_CRAFT data defaults to private so values may differ from processor to processor.

Private data has slightly different behavior than data specified in the `NEW` clause. The value of a private datum on each processor can be used beyond a single iteration of the loop. Private data may be used to compute local sums, for example. The values of data items named in a `NEW` clause may not be used beyond a single iteration. The `NEW` clause asserts that the `INDEPENDENT` directive would be valid if new objects were created for the variables named in the clause for each iteration of the loop. The semantics of the `NEW` clause are identical in HPF_CRAFT and HPF.

The semantics of an `INDEPENDENT` applied to loops containing private data references changes with respect to the private data. The change can be summarized to say that instead of indicating that iterations have no dependencies upon one-another, with respect to the private data, iterations on different processors have no dependencies upon one-another.

F.5.4 The ON Clause

In addition to the version of `INDEPENDENT` available from HPF, a new version of `INDEPENDENT` is included that incorporates the `ON` clause. There are a number of differences between the versions of `INDEPENDENT` with and without the `ON` clause.

The new version of the `INDEPENDENT` directive may be applied to the first of a group of tightly nested loops and may apply to more than one of them. This more easily facilitates the use of the `ON` clause. The current `INDEPENDENT` directive applies only to a single loop nest. The `INDEPENDENT` directive is extended so that multiple loop nests can be named. The general syntax for these new independent loops is as follows:

```
!HPF$ INDEPENDENT (I1,I2,...,In) ON array-name(h1(I1),h2(I2),...,hn(In))
      DO I1 = L1, U1, S1
            DO I2 = L2, U2, S2
              ...
                DO In = Ln, Un, Sn
                  ...
                END DO
              ...
            END DO
      END DO
```

The syntax and semantics of INDEPENDENT with the ON clause are different from its syntax and semantics without the ON clause. With the ON clause the directive states that there are no cross-processor dependencies, but there may be dependencies between iterations on a processor. There is an implicit barrier synchronization after an INDEPENDENT loop. Transfer of control into or out of an INDEPENDENT loop is prohibited.

The iteration space of an INDEPENDENT nest must be rectangular. That is, the lower loop bound, the upper loop bound, and the step expression for each loop indicated by the INDEPENDENT induction list must be invariant with regard to the INDEPENDENT nest. Each index expression of *array-name* in the ON clause (the functions h_i above,) must be one of the following two forms:

```
[ a * loop_control_variable + ] b
[ a * loop_control_variable - ] b
```

where a and b must be integer values; they can be expressions, constants, or variables. The values of a and b must be invariant with regard to the INDEPENDENT loop nest. For example, specifying A(I,J,K) is valid. Specifying A(3,I+J,K) is not valid. Specifying A(I,I,K) is not valid because I appears twice. Division is prohibited in any index expression of the ON clause.

F.5.5 Array Syntax

Array syntax is treated identically in HPF_CRAFT as in HPF for explicitly mapped objects. For private objects the behavior is identical to that of HPF_LOCAL. When private objects and explicitly mapped objects are combined the rules are as follows:

$$result = rhs_1 \ \mathrm{op}_1 \ rhs_2 \ \mathrm{op}_2 \ \ldots \ \mathrm{op}_m \ rhs_n$$

- If *result* is explicitly mapped and all *rhs* arrays are explicitly mapped, the work is distributed as in HPF.
- If *result* is private and all *rhs* arrays are private the computation is done on all processors as an HPF_LOCAL program would do it.
- If *result* is private and all *rhs* arrays are explicitly mapped, the work is distributed as in HPF and the values of the results are broadcast to the *result* on each processor.
- If *result* is explicitly mapped and *not* all *rhs* arrays are explicitly mapped, the results of the operation are undefined, unless all corresponding elements of all private *rhs* arrays have the same values.
- If *result* is private and some, but not all *rhs* arrays are explicitly mapped, the value is computed on each processor and saved to the local *result*.

All processors must participate in any array syntax statement in which the value of an explicitly mapped array is modified, and there is implicit barrier synchronization after the statement executes.

F.5.6 Treatment of FORALL and WHERE Statements

The `FORALL` and `WHERE` statements are treated exactly as in HPF when data is explicitly mapped. When private data is modified, the statement is executed separately on each processor. Finally, when data in a `FORALL` or `WHERE` are mixed, the rules for array syntax apply. If any explicitly mapped data item is modified in a *forall-stmt* or *where-stmt* then arrays in the *forall-header* or *where-header* must be explicitly mapped. In a `FORALL` construct, if any explicitly mapped array is modified, all modified arrays must be explicitly mapped. There is an implicit barrier synchronization after `FORALL` and `WHERE` statements if any arrays in the *forall-header* or *where-header* are explicitly mapped.

F.5.7 Synchronization Primitives

A number of synchronization primitives are provided. These primitives include:

Barriers (test, set, wait)
Locks (test, set, clear)
Critical Sections
Events (test, set, wait, clear)

Barriers provides an explicit mechanism for a task to indicate its arrival at a program point and to wait there until all other tasks arrive. A task may test and optionally wait at an explicit barrier point. In the following example, a barrier is used to make sure that *block3* is not entered by any task until all tasks have completed execution of *block1*.

```
block1
CALL SET_BARRIER()
block2
CALL WAIT_BARRIER()
block3
```

The following example performs a similar function as above. However, while waiting for all tasks to arrive at the barrier, the early tasks perform work within a loop.

```
block1
CALL SET_BARRIER()
DO WHILE (.NOT. TEST_BARRIER())
      block2
END DO
block3
```

Locks are used to prevent the simultaneous access of data by multiple tasks.

The `SET_LOCK(`*lock*`)` intrinsic sets the mapped integer variable *lock* atomically. If the lock is already set, the task that called `SET_LOCK` is suspended until the lock is cleared by another task and then sets it. Individual locks may be tested or cleared using *result* = `TEST_LOCK(`*lock*`)` and `CLEAR_LOCK(`*lock*`)` respectively.

A *critical section* protects access to a section of code rather than to a data object. The CRITICAL directive marks the beginning of a code region in which only one task can enter at a time. The END CRITICAL directive marks the end of the critical section. Transfer of control into or out of a critical section is prohibited.

```
!HPF$ CRITICAL
      GLOBAL_SUM = GLOBAL_SUM + LOCAL_SUM
!HPF$ END CRITICAL
```

Events are typically used to record the state of a program's execution and to communicate that state to another task. Because they do not set locks, as do the lock routines described earlier, they cannot easily be used to enforce serial access of data. They are suited to work such as signalling other tasks when a certain value has been located in a search procedure. There are four routines needed to perform the event functions, and each requires a mapped argument.

The SET_EVENT(*event*) routine sets or *posts* an event; it declares that an action has been accomplished or a certain point in the program has been reached. A task can post an event at any time, whether the state of the event is cleared or already posted. The CLEAR_EVENT(*event*) routine clears an event, the WAIT_EVENT(*event*) routine waits until a particualr event is posted, and the *result* = TEST_EVENT(*event*) function returns a logical value indicating whether a particular event has been posted.

F.5.8 Barrier Removal

You can explicitly remove an implicit barrier after any INDEPENDENT loop, or after any array syntax statement that modifies explicitly mapped arrays, by using the NO BARRIER directive.

```
!HPF$ NO BARRIER
```

F.5.9 Serial Regions

It is often useful to enter a region where only one task is executing. This is particularly useful for certain types of I/O. To facilitate this, two directives are provided. In addition, one may optionally attach a COPY clause to the END MASTER directive which specifies the private data items whose values should be broadcast to all processors. The syntax of this directive is:

```
!HPF$ MASTER
      sequential region
      ...
!HPF$ END MASTER [, COPY( var1 [, var2, ..., varn ])]
```

where *var* is `SYMMETRIC` private data to be copied to the same named private data on other processors.

If a routine is called within a serial region, the routine executes serially; there is no way to get back to parallel execution within the routine. All explicitly mapped data is accessible from within routines called in a serial region, but a routine called from within a serial region cannot allocate explicitly mapped data or remap data. All processors must participate in the invocation of the serial region. Transfer of control into or out of a serial region is not permitted.

F.5.10 Libraries

The HPF Local Routine Library is available in HPF_CRAFT. The HPF_LOCAL extrinsic environment contains a number of libraries that are useful for local SPMD programming and a number of libraries that allow the user to determine global (rather than local) state information. These library procedures take as input the name of a dummy argument and return information on the corresponding global HPF actual argument. They may only be invoked by an HPF_CRAFT procedure that was directly invoked by global HPF code. They may be called only for private data. The libraries reside in a module called `HPF_LOCAL_LIBRARY`.

The HPF Library is available to HPF_CRAFT when called with data that is explicitly mapped and all processors are participating in the call. In addition, as in HPF_LOCAL, the entire HPF Library is available for use with private data. Mixing private and explicitly mapped data in calls to the HPF library produces undefined behavior.

F.5.11 Parallel Inquiry Intrinsics

These intrinsic functions are provided as an extension to HPF. They return a logical value that provides information to the programmer about the state of execution in a program.

```
IN_PARALLEL()
IN_INDEPENDENT()
```

F.5.12 Task Identity

`MY_PE()` may be used to return the local processor number. The physical processors are identified by an integer in the range of 0 to *n-1* where *n* is the value returned by the global HPF_LIBRARY function `NUMBER_OF_PROCESSORS`. Processor identifiers are returned by `ABSTRACT_TO_PHYSICAL`, which establishes the one-to-one correspondence between the abstract processors of an HPF processors arrangement and the physical processors. Also, the local library function `MY_PROCESSOR` returns the identifier of the task executing the call.

F.5.13 Parallelism Specification Directives

These directives allow a user to assert that a routine will only be called from within a parallel region, a serial region, or from within both regions. Without these directives an implementation might be required to generate two versions of code for each routine, depending upon implementation strategies. The directives simply make the generated code size smaller and remove a test.

```
!HPF$ PARALLEL_ONLY
!HPF$ SERIAL_ONLY
!HPF$ PARALLEL_AND_SERIAL
```

The default is PARALLEL_ONLY.

F.5.14 The SYMMETRIC Directive

SYMMETRIC variables are private data that are guaranteed to be at the same storage location on every processor. The feature is beneficial to implementations that provide one-way communication functionality. One task can either *get* or *put* data into another task's symmetric data location, without involving the other task. There is an implicit barrier synchronization after SYMMETRIC data is allocated.

```
      REAL PRIV1(100), PRIV2
!HPF$ SYMMETRIC PRIV1, PRIV2
```

F.5.15 The RESIDENT Directive

The RESIDENT directive can be specified at the loop level and at the routine level. It is an assertion that the references to particular variables in the routine (or loop) are only references to data that are local to the task making the assertion. In the following loop, all references to arrays A, B, and C are local to the task executing each iteration.

```
      REAL A(100), B(100), C(100)
      INTEGER IX(100)
!HPF$ DISTRIBUTE A(BLOCK), B(BLOCK), C(BLOCK)
!HPF$ RESIDENT A

      ...
!HPF$ INDEPENDENT (I) ON B(I) RESIDENT(C)
      DO I = 1, 100
            A(IX(I)) = B(I) + C(IX(I))
      END DO
```

F.5.16 The ATOMIC UPDATE Directive

In HPF_CRAFT, the ATOMIC UPDATE directive tells the compiler that a particular data item or the elements of a particular array for a specified operation must be updated atomically. This can be used within loops or in array syntax and applies to both the elements of an array with an assignment of a permutation and the elements of an array within a loop.

In the following example, all references to R(IX(I)) occur atomically, thus eliminating the possibility that different iterations might try to modify the same element concurently.

```
      REAL R(200), S(1000)
      INTEGER IX(1000)
!HPF$ DISTRIBUTE R(BLOCK), S(BLOCK), IX(BLOCK)

!HPF$ INDEPENDENT (I) ON S(I)
      DO I = 1, 1000
!HPF$       ATOMIC UPDATE
            R(IX(I)) = R(IX(I)) + S(I)
      END DO
```

F.5.17 The GEOMETRY Directive

The GEOMETRY directive is simliar to a typedef in C, only it is for data mapping. It allows the user to conveniently change the mappings of many arrays at the same time. It is similar in many ways to the TEMPLATE directive, but since it is bound to no particular extent it is sometimes easier to apply.

```
!HPF$ GEOMETRY geom(d1 [, d2, ..., dn])
!HPF$ DISTRIBUTE ( geom ) [::] var1[, var2, ..., varm]
```

Where d_i indicates one of the allowable distribution formats.

```
!HPF$ GEOMETRY GBB(BLOCK, CYCLIC)
      REAL A(300,300), B(400,400)
!HPF$ DISTRIBUTE (GBB) ::  A, B
!     if GBB  changes then both A  and B  change
```

附属書G The FORTRAN 77 Local Library

The HPF standard now describes an EXTRINSIC(LANGUAGE='F77',MODEL='LOCAL') interface, or EXTRINSIC(F77_LOCAL) to use the keyword identification (see Section 11.6 for its description), similar in characteristics to the EXTRINSIC(LANGUAGE='HPF',MODEL='LOCAL') and EXTRINSIC(LANGUAGE='FORTRAN',MODEL='LOCAL') interfaces. This section describes a set of library routines to make it easier to make use of the F77_LOCAL interface when passing distributed array data. These library routines can facilitate, for example, a portable blend of global data parallel code with preexisting FORTRAN 77-based code using explicit message passing calls for interprocessor communication. The FORTRAN 77 Local Library interface described in this section was originally developed as part of Thinking Machines TMHPF and is now supported by Sun Microsystems Inc. For suggestions, requests, or corrections concerning this interface, please contact

Sun Microsystems Inc.
High Performance Computing
M/S UCHLO5-104
5 Omni Way
Chelmsford, MA 01824
f77-local-library@sun.com

G.1 Introduction

The basic constraints for the local model (Section 11.1) together with the F77_LOCAL-specific argument passing options (Section 11.6) define the nature of the F77_LOCAL interface: how control is to be transferred from a global HPF procedure to a set of local procedures described by an EXTRINSIC(F77_LOCAL) procedure interface and how data can be passed between these two types of procedures: by reference or by descriptor, and with or without temporary local reordering of data to satisfy FORTRAN 77 provisions for sequential, contiguous storage of array data in Fortran array element order. These alternative methods of argument passing can be obtained by use of the two special-purpose attributes for extrinsic dummy arguments defined for LANGUAGE='F77' routines: LAYOUT('F77_ARRAY') (the default) vs. LAYOUT('HPF_ARRAY'), and PASS_BY('*') (the default) vs. PASS_BY('HPF_HANDLE'). However, to take advantage of the option allowing one to pass global HPF array “handles” to local FORTRAN 77 procedures and then obtain information locally about how the local portion of a given parallel array is actually distributed requires special inquiry routines comparable to the HPF Local Library of functions. Since this library is not only described as a module, but uses many features such as array-valued functions and optional arguments not available in FORTRAN 77 code, it is recommended that a modified FORTRAN 77 interface to this library be provided in the manner described below. Furthermore, there is the problem of describing local portions of parallel arrays in the FORTRAN 77 code used in each local routine called from a global HPF one. Since assumed-shape syntax may not be used, explicit shape arrays are required. But it is common for global distribution of arbitrary sized arrays to result in local portions of arrays that do not have constant shapes on all processors, and the actual extents in each processor cannot necessarily be predicted in advance. In order to allow programmers to obtain axis extent information at run time from the HPF global caller, a special HPF-callable subgrid inquiry subroutine is provided. A FORTRAN 77 callable version of the same routine is also described below, for flexibility in programming.

G.2 Summary

- One HPF-callable subgrid inquiry subroutine

    HPF_SUBGRID_INFO

- A set of FORTRAN 77-callable inquiry subroutines

    F77_SUBGRID_INFO

    F77_GLOBAL_ALIGNMENT

    F77_GLOBAL_DISTRIBUTION

    F77_GLOBAL_TEMPLATE

    F77_ABSTRACT_TO_PHYSICAL

    F77_PHYSICAL_TO_ABSTRACT

    F77_LOCAL_TO_GLOBAL

    F77_GLOBAL_TO_LOCAL

    F77_LOCAL_BLKCNT

    F77_LOCAL_LINDEX

    F77_LOCAL_UINDEX

    F77_GLOBAL_SHAPE

    F77_GLOBAL_SIZE

    F77_SHAPE

    F77_SIZE

    F77_MY_PROCESSOR

G.3 Global HPF Subgrid Inquiry Routine

The F77_LOCAL library interface includes only one global HPF subroutine, HPF_SUBGRID_INFO, whose implementation should be added as an extension to the standard HPF Library module. Its purpose is to provide per-processor information about the local subgrids of distributed arrays. This information is often critical when passing such arrays to local procedures written in FORTRAN 77, where array argument shapes must be stated explicitly in the local procedure (except in the last dimension; there are "assumed size" but no "assumed shape" arrays) , but may be expressed in terms of arguments passed at run time ("adjustable shape arrays"). Thus the subgrid parameters obtained from this subgrid inquiry routine can be passed as arguments to the local routines and used there to describe the extents of the locally visible portions of global HPF arrays, as the example in Section G.5 will demonstrates.

HPF_SUBGRID_INFO (ARRAY, IERR, DIM, LB, UB, STRIDE, LB_EMBED, UB_EMBED, AXIS_MAP)

Description. Gives local information about local subgrid allocation onto each processor of a distributed array; callable from a global HPF routine.

Class. Inquiry subroutine.

Arguments.

ARRAY — is a nonsequential array of any type, size, shape, or mapping. It is an INTENT (IN) argument.

IERR — is a scalar integer of default kind. It is an INTENT (OUT) argument. Its return value is zero upon successful return and nonzero otherwise. Errors result if local subgrids cannot be expressed as array sections of ARRAY.

If any of the optional arguments LB_EMBED, UB_EMBED, or AXIS_MAP is present, then a nonzero value is also returned if the compiler does not organize the local data in serial memory by sequence associating a larger "embedding" array (see Section G.3.1 below for more explanation).

DIM (optional) — is a scalar integer of default kind. It is an INTENT (IN) argument. DIM indicates the axis along which return values are desired. If DIM is not present, values are returned for all axes.

LB (optional) — is an INTENT (OUT), default integer array. If this argument is present, and if the value returned in IERR is zero, the values returned in array LB are the lower bounds in global coordinates of each processor's subgrid, along one (if DIM is present) or each dimension of ARRAY.

UB (optional) — is an INTENT (OUT), default integer array. If this argument is present, and if the value returned in IERR is zero, the values returned in array UB are the upper bounds in global coordinates of each processor's subgrid, along one (if DIM is present) or each dimension of ARRAY.

STRIDE (optional) — is an INTENT (OUT), default integer array. If this argument is present, and if the value returned in IERR is zero, the values returned in array STRIDE are the strides in local memory between elements of each processor's subgrid, along one (if DIM is present) or each dimension of ARRAY.

LB_EMBED (optional) — is an INTENT (OUT), default integer array. If this argument is present, and if the value returned in IERR is zero, the values returned in array LB_EMBED are the lower bounds in global coordinates of the actual global array elements allocated on each processor, possibly a superset of the user-visible subgrid, along one (if DIM is present) or each dimension of ARRAY.

UB_EMBED (optional) — is an INTENT (OUT), default integer array. If this argument is present, and if the value returned in IERR is zero, the values returned in array UB_EMBED are the upper bounds in global coordinates of the actual global array elements allocated on each processor, possibly a superset of the user-visible subgrid, along one (if DIM is present) or each dimension of ARRAY.

AXIS_MAP (optional) — is a rank 2, INTENT (OUT), default integer array. If this argument is present, its shape must be at least [n,r], where n is the number of processors and r is the rank of ARRAY.

If the value returned in IERR is zero, the values returned in AXIS_MAP(i,1:r) represent the numbers of the axes of the subgrid on processor i from fastest varying to slowest varying, and form a permutation of the sequence 1,2,...,r.

For the last six arguments, LB, UB, STRIDE, LB_EMBED, UB_EMBED, and AXIS_MAP, each array has a first axis of extent at least n, where n is the number of processors, and the first n indices of that axis of each array must be distributed (perhaps via an explicit CYCLIC or BLOCK distribution) one index per processor. If a second dimension is needed, it should be a collapsed axis of extent at least equal to the rank of ARRAY.

If HPF_SUBGRID_INFO is called, and the elements of ARRAY that are local to any particular processor are not representable as an array section of the global user array, then a nonzero value is returned for IERR. Otherwise, if any of the optional arguments LB, UB, or STRIDE is present, then the lower bounds, upper bounds, or strides, respectively, that describe the local array sections are returned in terms of one-based, global coordinates.

G.3.1 Subgrid Inquiries Involving Embedding Arrays

In the common case in which the elements of each local subgrid of the global array argument are distributed across processors, with no overlap, and allocated in local memory like a local FORTRAN 77 array, as a contiguous sequence of elements in Fortran array element order, these three last optional arguments would not be required.

However, some implementations may choose less common layouts in local memory, that involve "embedding" these elements in a larger array section of equal rank that *is* sequence-associated in serial memory. For example, alignment of axes of arrays in different orders may result in a permuting embedding of the subgrid. Or axes of subgrids map be padded with ghost cells, either for stencil optimizations or to achieve same-size subgrids on all nodes.

In variations such as these, we may still view the subgrid as being "embedded" in a sequence associated array which may be accessible in F77_LOCAL operations, if the permutation of axes, shape of any embedding array, and offsets into that array can be obtained at runtime. The last three arguments of HPF_SUBGRID_INFO are provided to allow programmers to obtain this information when it is appropriate, with the help of the IERR flag to signal when this is not the case.

In this mapping, local memory has been allocated for a larger array section, with coordinates (LB_EMBED : UB_EMBED : STRIDE). The coordinates of the *actual* computational elements are limited to the subset (LB : UB : STRIDE). The sequence association is generalized to an arbitrary mapping of axes. Here, AXIS_MAP numbers the axes from fastest varying to slowest varying. If LB_EMBED, UB_EMBED, or AXIS_MAP is specified in a call to HPF_SUBGRID_INFO but ARRAY does not satisfy the assumptions of this mapping model, then a nonzero value is returned for IERR.

G.4 Local FORTRAN 77 Inquiry Routines

Here the F77-callable inquiry subroutines are described briefly. These provide essentially the same capability as the combination of the HPF intrinsic array inquiry functions such as SHAPE and SIZE, together with the HPF LOCAL LIBRARY inquiry routines. The subroutine F77_SUBGRID_INFO serves as a local counterpart to the globally callable subroutine HPF_SUBGRID_INFO described above. In all of the following:

- ARRAY is a dummy argument passed in from a global HPF caller using the LAYOUT ('HPF_ARRAY') attribute and declared within the FORTRAN 77 local subroutine as a scalar integer variable. It is an INTENT (IN) argument.

- DIM is a scalar integer of default kind. It is an INTENT (IN) argument. This argument specifies a particular axis of the global array associated with ARRAY or, if DIM = -1, inquiry is for all axes.

- An "inquiry result" is an INTENT (OUT) argument. If DIM = -1, it is a rank-one array of size equal to at least the rank of the global array associated with ARRAY, returning information associated with all axes. If DIM is positive, the "inquiry result" is a scalar, returning information only for the axis indicated by DIM.

- The arguments are defined in the same way as for the corresponding HPF or HPF_LOCAL routines unless otherwise noted. See the description of HPF_SUBGRID_INFO above and Section 11.7.1 for full specifications of the similarly-named HPF_LOCAL_LIBRARY procedures.

F77_SUBGRID_INFO (ARRAY, IERR1, IERR2, DIM, LB, UB, STRIDE, LB_EMBED, UB_EMBED, AXIS_MAP)

Description. This is a FORTRAN 77-callable version of the HPF subroutine HPF_SUBGRID_INFO.

Arguments.

IERR1 — is a scalar integer of default kind. It is an INTENT (OUT) argument. Its return value is zero if LB, UB, and STRIDE were determined successfully and nonzero otherwise.

IERR2 — is a scalar integer of default kind. It is an INTENT (OUT) argument. Its return value is zero if LB_EMBED and UB_EMBED were determined successfully and nonzero otherwise.

LB, UB, STRIDE, LB_EMBED, UB_EMBED, AXIS_MAP are "inquiry results" of default integer type. They are the lower and upper bounds and strides of the array sections describing the local data (in terms of global indices), the lower and upper bounds of the embedding arrays (again, in terms of global indices), and the axes of the embedding arrays to which the axes of ARRAY are mapped.

F77_GLOBAL_ALIGNMENT (ALIGNEE, LB, UB, STRIDE, AXIS_MAP, IDENTITY_MAP, DYNAMIC, NCOPIES)

Description. This is a FORTRAN 77-callable version of the HPF_LOCAL subroutine GLOBAL_ALIGNMENT. All but the first are INTENT (OUT) arguments whose return values are as specified by the corresponding HPF routine.

Arguments.

ALIGNEE — is a dummy argument passed in from global HPF. It is an INTENT (IN) argument.

LB, UB, STRIDE, AXIS_MAP — are integer arrays of rank one. Their size must be at least equal to the rank of the global HPF array associated with ALIGNEE.

IDENTITY_MAP, DYNAMIC — are scalar logicals.

NCOPIES — is a scalar integer of default kind.

F77_GLOBAL_DISTRIBUTION (DISTRIBUTEE, AXIS_TYPE, AXIS_INFO, PROCESSORS_RANK, PROCESSORS_SHAPE)

Description. This is a FORTRAN 77-callable version of the HPF_LOCAL subroutine GLOBAL_DISTRIBUTION. All but the first are INTENT (OUT) arguments whose return values are as specified by the corresponding HPF routine.

Arguments.

DISTRIBUTEE — is a dummy argument passed in from global HPF. It is an INTENT (IN) argument.

AXIS_TYPE — is a CHARACTER*9 array of rank one. Its size must be at least equal to the rank of the global HPF array associated with DISTRIBUTEE.

AXIS_INFO — is a default integer array of rank one. Its size must be at least equal to the rank of the global HPF array associated with DISTRIBUTEE.

PROCESSORS_RANK — is a scalar of default integer type.

PROCESSORS_SHAPE — is an integer array of rank one. Its size must be at least equal to the value returned by PROCESSORS_RANK.

F77_GLOBAL_TEMPLATE (ALIGNEE, TEMPLATE_RANK, LB, UB, AXIS_TYPE, AXIS_INFO, NUMBER_ALIGNED, DYNAMIC)

Description. This is a FORTRAN 77-callable version of the HPF_LOCAL subroutine GLOBAL_TEMPLATE. All but the first are INTENT (OUT) arguments whose return values are as specified by the corresponding HPF routine.

Arguments.

ALIGNEE — is a dummy argument passed in from global HPF. It is an INTENT (IN) argument.

TEMPLATE_RANK	is a scalar integer of default kind.
LB, UB, AXIS_INFO	are integer arrays of rank one. Their size must be at least equal to the rank of the align-target to which the global HPF array associated with ALIGNEE is ultimately aligned.
AXIS_TYPE	is a CHARACTER*10 array of rank one. Its size must be at least equal to the rank of the align-target to which the global HPF array associated with ALIGNEE is ultimately aligned.
NUMBER_ALIGNED	is a scalar integer of default kind.
DYNAMIC	is a scalar logical.

F77_ABSTRACT_TO_PHYSICAL(ARRAY, INDEX, PROC)

Description. This is a FORTRAN 77-callable version of the HPF_LOCAL subroutine ABSTRACT_TO_PHYSICAL.

Arguments.

INDEX	is a rank-one, INTENT (IN), integer array.
PROC	is a scalar, INTENT (OUT), integer.

F77_PHYSICAL_TO_ABSTRACT(ARRAY, PROC, INDEX)

Description. This is a FORTRAN 77-callable version of the HPF_LOCAL subroutine PHYSICAL_TO_ABSTRACT.

Arguments.

PROC	is a scalar, INTENT (IN), integer.
INDEX	is a rank-one, INTENT (OUT), integer array.

F77_LOCAL_TO_GLOBAL(ARRAY, L_INDEX, G_INDEX)

Description. This is a FORTRAN 77-callable version of the HPF_LOCAL subroutine LOCAL_TO_GLOBAL.

Arguments.

L_INDEX	is a rank-one, INTENT (IN), integer array.
G_INDEX	is a rank-one, INTENT (OUT), integer array.

F77_GLOBAL_TO_LOCAL(ARRAY, G_INDEX, L_INDEX, LOCAL, NCOPIES, PROCS)

Description. This is a FORTRAN 77-callable version of the `HPF_LOCAL` subroutine `GLOBAL_TO_LOCAL`.

Arguments.

`G_INDEX`	is a rank-one, `INTENT (IN)`, integer array.
`L_INDEX`	is a rank-one, `INTENT (OUT)`, integer array.
`LOCAL`	is a scalar, `INTENT (OUT)`, logical.
`NCOPIES`	is a scalar, `INTENT (OUT)`, integer.
`PROCS`	is a rank-one, integer array whose size is at least the number of processors that hold copies of the identified element.

F77_LOCAL_BLKCNT(L_BLKCNT, ARRAY, DIM, PROC)

Description. This is a FORTRAN 77-callable version of the `HPF_LOCAL` function `LOCAL_BLKCNT`.

Arguments.

`L_BLKCNT`	is an "inquiry result" of type integer.
`PROC`	is a scalar integer of default kind. It must be a valid processor number or, if `PROC = -1`, the value returned by `F77_MY_PROCESSOR()` is implied.

F77_LOCAL_LINDEX(L_LINDEX, ARRAY, DIM, PROC)

Description. This is a FORTRAN 77-callable version of the `HPF_LOCAL` function `LOCAL_LINDEX`.

Arguments.

`L_LINDEX`	is a rank-one, integer array of size equal to at least the value returned by `F77_LOCAL_BLKCNT`.
`DIM`	may not be -1.
`PROC`	is a scalar integer of default kind. It must be a valid processor number or, if `PROC = -1`, the value returned by `F77_MY_PROCESSOR()` is implied.

F77_LOCAL_UINDEX(L_UINDEX, ARRAY, DIM, PROC)

Description. This is a FORTRAN 77-callable version of the `HPF_LOCAL` function `LOCAL_UINDEX`.

Arguments.

`L_UINDEX`	is a rank-one, integer array of size equal to at least the value returned by `F77_LOCAL_BLKCNT`.
`DIM`	may not be -1.
`PROC`	is a scalar integer of default kind. It must be a valid processor number or, if `PROC = -1`, the value returned by `F77_MY_PROCESSOR()` is implied.

F77_GLOBAL_SHAPE(SHAPE, ARRAY)

Description. This is a FORTRAN 77-callable version of the `HPF_LOCAL` function `GLOBAL_SHAPE`.

Arguments.

`SHAPE`	is a rank-one, integer array of size equal to at least the rank of the global array associated with `ARRAY`. Its return value is the shape of that global array.

F77_GLOBAL_SIZE(SIZE, ARRAY, DIM)

Description. This is a FORTRAN 77-callable version of the `HPF_LOCAL` function `GLOBAL_SIZE`.

Arguments.

`SIZE`	is a scalar integer equal to the extent of axis `DIM` of the global array associated with `ARRAY` or, if `DIM = -1`, the total number of elements in that global array.

F77_SHAPE(SHAPE, ARRAY)

Description. This is a FORTRAN 77-callable version of the HPF intrinsic `SHAPE`, as it would behave as called from `HPF_LOCAL`.

Arguments.

`SHAPE`	is a rank-one, integer array of size equal to at least the rank of the subgrid associated with `ARRAY`. Its return value is the shape of that subgrid.

F77_SIZE(SIZE, ARRAY, DIM)

Description. This is a FORTRAN 77-callable version of the HPF intrinsic SIZE, as it would behave as called from HPF_LOCAL.

Arguments.

SIZE — is a scalar integer equal to the extent of axis DIM of the subgrid associated with ARRAY or, if DIM = -1, the total number of elements in that subgrid.

F77_MY_PROCESSOR(MY_PROC)

Description. This is a FORTRAN 77-callable version of the HPF_LOCAL function MY_PROCESSOR.

Arguments.

MY_PROC — is a scalar, INTENT (OUT), integer. Its value is the identifying number of the physical processor from which this call is made.

G.5 Programming Example Using HPF_SUBGRID_INFO

G.5.1 HPF Caller

```
      PROGRAM EXAMPLE
! Declare the data array and a verification copy
      INTEGER, PARAMETER :: NX = 100, NY = 100
      REAL, DIMENSION(NX,NY) :: X, Y
!HPF$ DISTRIBUTE(BLOCK,BLOCK) :: X, Y
! The global sum will be computed
! by forming partial sums on the processors
      REAL PARTIAL_SUM(NUMBER_OF_PROCESSORS())
!HPF$ DISTRIBUTE PARTIAL_SUM(BLOCK)
! Local subgrid parameters are declared per processor
! for a rank-two array
      INTEGER, DIMENSION(NUMBER_OF_PROCESSORS(),2) ::
     & LB, UB, NUMBER
!HPF$ DISTRIBUTE(BLOCK,*) :: LB, UB, NUMBER
! Define interfaces
      INTERFACE
        EXTRINSIC(F77_LOCAL) SUBROUTINE LOCAL1
     &   ( LB1, UB1, LB2, UB2, NX, X )
! Arrays LB1, UB1, LB2, UB2, and X are passed by default
```

```
! as LAYOUT('F77_ARRAY') and PASS_BY('*')
        INTEGER, DIMENSION(:) :: LB1, UB1, LB2, UB2
        INTEGER NX
        REAL X(:,:)
!HPF$ DISTRIBUTE(BLOCK) :: LB1, UB1, LB2, UB2
!HPF$ DISTRIBUTE(BLOCK,BLOCK) :: X
        END
        EXTRINSIC(F77_LOCAL) SUBROUTINE LOCAL2(N,X,R)
! Arrays N, X, and R are passed by default
! as LAYOUT('F77_ARRAY') and PASS_BY('*')
        INTEGER N(:)
        REAL X(:,:), R(:)
!HPF$ DISTRIBUTE N(BLOCK)
!HPF$ DISTRIBUTE X(BLOCK,BLOCK)
!HPF$ DISTRIBUTE R(BLOCK)
        END
      END INTERFACE

! Determine result using only global HPF
      ! Initialize values
      FORALL (I=1:NX,J=1:NY) X(I,J) = I + (J-1) * NX
      ! Determine and report global sum
      PRINT *, 'GLOBAL HPF RESULT: ',SUM(X)
! Determine result using local subroutines
      ! Initialize values ( assume stride = 1 )
      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), NX, Y )
      ! Determine and report global sum
      NUMBER = UB - LB + 1
      CALL LOCAL2 ( NUMBER(:,1) * NUMBER(:,2) , Y , PARTIAL_SUM )
      PRINT *, 'F77_LOCAL RESULT #1 : ',SUM(PARTIAL_SUM)
      END
```

G.5.2 FORTRAN 77 Callee

```
      SUBROUTINE LOCAL1( LB1, UB1, LB2, UB2, NX, X )
! The global actual arguments passed to LB1, UB1, LB2, and UB2
! have only one element apiece and so can be treated as scalars
! in the local Fortran 77 procedures
      INTEGER LB1, UB1, LB2, UB2
! NX contains the global extent of the first dimension
```

```
      ! of the global array associated with local array X
            INTEGER NX
      ! Note that X may have no local elements.
            REAL X ( LB1 : UB1 , LB2 : UB2 )
      ! Initialize the elements of the array, if any
            DO J = LB2, UB2
              DO I = LB2, UB2
                X(I,J) = I + (J-1) * NX
              END DO
            END DO
            END

            SUBROUTINE LOCAL2(N,X,R)
      ! Here, the rank of the original array is unimportant
      ! Only the total number of local elements is needed
      !     INTEGER N
            REAL X(N), R
      ! If N is zero, local array X has no elements, but R
      ! still computes the correct local sum
            R = 0.
            DO I = 1, N
              R = R + X(I)
            END DO
            END
```

G.6 Programming Example Using F77-Callable Inquiry Subroutines

This example performs only the initialization of the above example. It illustrates use of the F77-callable inquiry routines on descriptors passed from HPF, as well as the addressing of uncompressed local subgrid data in terms of "embedding arrays."

G.6.1 HPF Caller

```
            PROGRAM EXAMPLE
            INTEGER, PARAMETER :: NX = 100, NY = 100
            REAL, DIMENSION(NX,NY) :: X
      !HPF$ DISTRIBUTE(BLOCK,BLOCK) :: X
      ! Local subgrid parameters are declared per processor
      ! for a rank-two array
            INTEGER, DIMENSION(NUMBER_OF_PROCESSORS(),2) ::
           & LB, UB, LB_EMBED, UB_EMBED
      !HPF$ DISTRIBUTE(BLOCK,*) :: LB, UB, LB_EMBED, UB_EMBED
      ! Define interfaces
```

```
      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
! X is passed twice, both times without local reordering.
! First, it is passed by reference for accessing array elements.
        REAL, DIMENSION(:,:), LAYOUT('HPF_ARRAY'),
     &          PASS_BY('*')                    :: X
! It is also passed by descriptor for use in F77 LOCAL
! LIBRARY subroutines only.
        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
! Initialize values
! ( Assume stride = 1 and no axis permutation )
      CALL HPF_SUBGRID_INFO( X, 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), X, X )
      END
```

G.6.2 FORTRAN 77 Callee

```
      SUBROUTINE LOCAL1(
     & LB1, UB1, LB_EMBED1, UB_EMBED1,
     & LB2, UB2, LB_EMBED2, UB_EMBED2, X, X_DESC )
      INTEGER LB1, UB1, LB_EMBED1, UB_EMBED1
      INTEGER LB2, UB2, LB_EMBED2, UB_EMBED2
! The subgrid has been passed in its 'embedded' form
      REAL X ( LB_EMBED1 : UB_EMBED1 , LB_EMBED2 : UB_EMBED2 )
! Locally X_DESC is declared as an INTEGER
      INTEGER X_DESC
```

```
! Get the global extent of the first axis
! This is an HPF_LOCAL type of inquiry routine with an
! 'F77_' prefix
      CALL F77_GLOBAL_SIZE(NX,X,1)
! Otherwise, initialize elements of the array
! Loop only over actual array elements
      DO J = LB2, UB2
        DO I = LB2, UB2
          X(I,J) = I + (J-1) * NX
        END DO
      END DO
      END
```