cci list: 05/03/95

item number 1 **title** Partial processor shape

submitted by Eiji Nunohiro **email** nunohiro@hisoft.soft.hitachi.co.jp **date** 01/20/95

group Group C **status** new **action:**

doc change text **doc change status** doc status not known

formal reply

original Q

>1. If the processors available to the program are part of elements of the
>implementation-dependent hardware processor array, should
PROCESSORS_SHAPE()
>return the shape of the whole processor array, or otherwise ?

discussion

answer sent by chk:
The language definition states (Subsection 2.6.6.4, I'm reading the HTML version so I don't have the page number)

NUMBER_OF_PROCESSORS(DIM)
Optional Argument. DIM
Description. Returns the total number of processors available to the program or the number of processors available to the program along a specified dimension of the processor array.
Class. System inquiry function.
Arguments.

Based on the description, I would expect NUMBER_OF_PROCESSORS to return
information about the part of the hardware array that is available to the program. (I admit that this is slightly unclear in the text; we will examine it in the CCI part of the next HPFF meeting in May.)

item number 2 **title** Should there be a “undefined” result from shape???

submitted by Eiji Nunohiro **email** nunohiro@hisoft.soft.hitachi.co.jp **date** 01/20/95

group Group not assigned **status** status: new **action:**

doc change text **doc change status** doc status not known

formal reply

original Q

>In another case shown in figure.2,
>what is the return value of NUMBER_OF_PROCESSORS()
>and PROCESSORS_SHAPE() ?
>
>What is the return value of NUMBER_OF_PROCESSORS(DIM=2) ?

```
O O X
O X X
X O X
```

figure.2

discussion

chk answer: Clearly, NUMBER_OF_PROCESSORS()=4.
PROCESSORS_SHAPE() is something else.
The important point to understand is that the interpretation of a PROCESSOR is implementation-dependent. If the topology of the machine is not an array (for example, a tree or figure 2) then the system must provide some (hopefully reasonable) mapping to describe it as an array. In your example, either PROCESSORS_SHAPE()=[4] or PROCESSORS_SHAPE()=[2,2] seem reasonable. In either case, the compiler will have to translate between the shape it chooses and the hardware topology.

item number 3 **title** Pure / Allocatable

submitted by Henry Zongaro **email** zongaro@vnet.ibm.com **date** 03/30/95

group Group C **status** status: new **action:**

doc change text **doc change status** doc status not known

formal reply reply:

original Q

discussion

2) Consider the following example. Imagine that in a particular implementation, after 999 999 allocations and deallocations of dynamic storage, the memory manager is no longer able to fulfill a request for a 1000 element integer array.

```
program p
  integer :: a(1000000), i

  forall (i = 1:1000000) a(i) = f()
contains
  pure integer function f()
    integer, allocatable :: a(:)

    allocate(a(1000), stat=f)
    deallocate(a)
  end function f
end program p
```

For 999 999 references to f, the value returned will be 0. For one reference, the value returned will be some positive value. The order in which the references to f() for each value of i are executed will affect the value of a.

Is f a pure function? It doesn't violate any of the constraints, but should it be allowed to return different values based on the order in which the references to f() occur? More importantly, can a compiler assume, in the face of an example like the above, that two references to a pure function, with the same actual arguments and with no intervening changes to global data, will return the same value?

item number 4 **title** Why SIZE restriction in PURE?

submitted by Henry Zongaro **email** zongaro@vnet.ibm.com **date** 03/30/95

group Group C **status** status: new **action:**

doc change text **doc change status** doc status not known

formal reply reply:

original Q

discussion

3) In the HPF Spec., on pp. 75-76 (4.3.1.1), there is a long constraint which prohibits global variables and dummy arguments in pure functions from appearing as the variable in a SIZE= specifier. There is a similar constraint on page 79 (4.3.1.2) for pure subroutines. Is this constraint required?

According to page 120, lines 26-27 and lines 22-24 (9.4.1) of the Fortran 90 standard, respectively, "A SIZE= specifier may be present only in an input statement that contains an ADVANCE= specifier with the value NO," and "An ADVANCE= specifier may be present only in a formatted sequential input/output statement. . . whose control information list does not contain an internal unit specifier."

Since a pure function or subroutine must not contain a READ statement which specifies an external unit or *, it follows that SIZE= can't appear in a pure function or subroutine, so the constraint isn't necessary.

item number 5 **title** Maxloc / rank definition problem

submitted by Henry Zongaro **email** zongaro@vnet.ibm.com **date** 03/30/95

group Group E **status** status: new **action:**

doc change text **doc change status** doc status not known

formal reply

original Q

discussion

4) There appears to be a serious problem in the definitions of MAXLOC and MINLOC. According to page 104, lines 17-25, the result of

MAXLOC(ARRAY, DIM[, MASK])

is a scalar if ARRAY has rank one. If the rank of ARRAY is greater than one, the value of element $(s_1, \ldots, s_{DIM-1}, s_{DIM}, \ldots, s_n)$ of the result is equal to

MAXLOC(ARRAY($s_1, \ldots, s_{DIM-1}$, :, $s_{DIM}, \ldots, s_n$),
[MASK($s_1, \ldots, s_{DIM-1}$, :, $s_{DIM}, \ldots, s_n$)]),

where n is the rank of ARRAY. Unfortunately, I believe the value of this is a rank one array whose size is equal to the rank of the first argument - how can the value of an *element* of the result of MAXLOC be an array?

The problem isn't immediately apparent when you look at the examples on lines 33-37, because in the two-dimensional case, each element of the result (according to the text) will be a rank one array of only one element - still invalid, but not obviously so.

item number 6 **title** Allocating multiple items

submitted by Henry Zongaro **email** zongaro@vnet.ibm.com **date** 03/30/95

group Group D **status** status: new **action:**

doc change text **doc change status** doc status not known

formal reply

original Q

discussion

5) According to page 39, lines 7-8 (3.6), "when an object X is created it may be aligned to another object Y only if Y has already been created or allocated." Fortran 90 does not specify an order for allocation or deallocation in ALLOCATE and DEALLOCATE statements. Does the text cited prohibit the following example,

```
      program p
        integer, allocatable :: a(:), b(:)
!hpf$   align b(:) with a(:)
        allocate(a(1), b(1))
      end program p
```

or is additional text required?

cci list: 05/03/95

item number 7 **title** INHERIT dummy shape

submitted by Yasuharu Hayashi **email** hayashi@lang1.bs1.fc.nec.co.jp **date** 02/27/95

group Group D **status** status: new **action:**

doc change text **doc change status** doc status not known

formal reply

original Q

```
>      I have a question about "High Performance Fortran Language
>      Specification November 10,1994 Version 1.1".
>      example;
>           PROGRAM SEQ
>      !HPF$ SEQUENCE A
>           REAL A(2,2)
>             :
>           CALL SUB(A)
>             :
>           END
>
>           SUBROUTINE SUB(B)
>           REAL B(4)

>     !HPF$ SEQUENCE B
>      !HPF$ INHERIT B
>             :
>
>      Since it could be argued that B is an aggregate cover of itself,
>      the INHERIT directive for B is HPF-conforming.
>      (the text on p.152,l.13 says,
>      "A sequential variable may not be explicitly mapped unless it is a
>       scalar or rank-one array,and is an aggregate cover. ...")
>
>      And according to the text p.46,l.40,
>      "If the dummy argument is not explicitly aligned and does have the
>      INHERIT attribute,then the template is "inherited" from the actual
>      argument according to the following rules:
>       -If the actual argument is a whole array ,the template of the dummy
>      is a copy of the template with which the actual argument is ultimately
>      aligned. ..."
>
>      In this example,the template of B have the same shape of [2,2] as A,
>      But it seems that the appropriate alignment of B with the shape of [2,2]
>      can not be defined.
>      Isn't it necessary to add the following description to the definition
>      of "inherited template" ?;
>      "If the dummy argument has INHERIT attribute and differs from the
>      actual argument in shape,then the shape and distribution of the
>      template may be chosen arbitrarily by the language processor."
>      Thanks,
>                              Yasuharu Hayashi
>                              NEC Corporation
>                              hayashi@lang1.bs1.fc.nec.co.jp
++++++++++++++++++++++++++++++++++++
Beginning of discussion
reply to answer sent by Rob:
Hello,
Thank you for your answer.
> I just reread your question about a dummy that has the inherit
> attribute and has a different shape than the whole-array actual; both
> dummy and actual have the sequence attribute.
>
> Very good point.   Your proposed solution, which is that the
> dummy's mapping can be chosen by the compiler, would work here.
> I am concerned that this may not cover all the possible problems.
>
> Would another possible interpretation be the following:  The mapping
> of B in your example is indeed inherited, with
>        b(1) mapped to the same processors as a(1,1)
>        b(2) mapped to the same processors as a(2,1)
>        b(3) mapped to the same processors as a(1,2)
>        b(4) mapped to the same processors as a(2,2)
```

discussion

```
Discussion continued:
> The compiler has chosen a mapping for A that is consistent with the
> fact that A is sequential.  Therefore, it is a mapping that could be
> described as the mapping of a one-dimensional cover of A.   Therefore,
> it is sensible for B to have this mapping too.  In other words, maybe
> the template of a sequential variable is one-dimensional regardless of
> its rank?
>
> The following is also possible:
>           PROGRAM SEQ
>      !HPF$ SEQUENCE M
>           REAL M(2,2)
>             :
>           CALL SUB(M(1,2))
>             :
>           END
>
>           SUBROUTINE SUB(N)
>           REAL N(2)
>      !HPF$ SEQUENCE N
>      !HPF$ INHERIT N
>             :
> Should this be legal?   It cannot be reconciled with the language:

According to "High Performance Fortran Language Specification
November 10,1994 Version 1.1", p.46,l.40,

"If the dummy argument is not explicitly aligned and does have the
INHERIT attribute,then the template is "inherited" from the actual
argument according to the following rules:
-If the actual argument is a whole array ......
-If the actual argument is an array section of array A where
no subscript is a vector subscript .....
-If the actual argument is any other expression ,the shape and
 distribution of the template may be chosen arbitrarily by the
 language processor(and therefore the programmer cannot know
 anything a priori about its distribution) "

I thought the third case described above was applied to this example.
In general, unless the actual argument is a whole array or a regular
array section, the alignment of the corresponding dummy argument which
have INHERIT attribute with the template of it cannot be described
by the ordinary HPF's alignment.
So I thought the third definition of the inherited template was applied
to all such cases. Isn't it correct?
And even though the actual argument is a whole array or a regular array
section ,if it is sequential (i.e. if the actual argument differs from
the dummy argument in shape), the same problem may occur.
And in such cases your proposed solution, which is that
the template of a sequential variable is always one-dimensional,
would go well too.

Please give me any comment.
>     "The INHERIT attribute specifies that a dummy argument should be
>      aligned to a copy of the template of the corresponding actual
>      argument in the same way that the actual argument is aligned."
>
> One could call this nonconforming, rather than saying it must
> be made to work by having the compiler choose the mapping of the dummy.
>
> These are just some ideas.  I dont know the right answer to this problem.
> Rob Schreiber
```

item number 8 **title** Dummy assertion asterisk

submitted by Yasuharu Hayashi **email** hayashi@lang1.bs1.fc.nec.co.jp **date** 04/25/95

group Group D **status** status: new **action:**

doc change text **doc change status** doc status not known

formal reply

original Q

I have a question about the interpretation of the assertion asterisk when the template of a dummy argument is a natural template. According to High Performance Fortran Language Specification November 10 ,1994 Version 1.1 p.51,l.31,

"If the dummy argument has a natural template (no INHERIT attribute) then things are more complicated. In certain situations the programmer is justified in inferring a preexisting distribution for the natural template ......"

When a actual argument is a whole array, the text on p.51,l.35 states only

"In all these situations, the actual argument must be a whole array or array section, and the template of the actual must be coextensive with the array along any axis having a distribution format other than "*". If the actual argument is a whole array, then the pre-existing distribution of the natural template of the dummy is identical to that of the actual argument".

I think this description is ambiguous. For example :

```
      PROGRAM EX
      REAL A(10,10),B(5,5)
!HPF$ PROCESSORS P(5)
!HPF$ DISTRIBUTE A(BLOCK,*) ONTO P
!HPF$ ALIGN B(*,I) WITH A(2*I,*)
      CALL SUB(B)
        :
      END
      SUBROUTINE SUB(BB)
      REAL BB(5,5)
!HPF$ PROCESSORS P(5)
!HPF$ DISTRIBUTE *(*,BLOCK(1)) ONTO *P :: BB
        :
      END
```

Is the assertion asterisks for BB in SUB HPF-conforming ?

Isn't it necessary to add the list as follows which shows what assertion asterisks for a natural template are legal when a actual argument is a whole array ?

"1.If n th axis of a actual argument which is a whole array corresponds to T(n) th axis of the template of it and j > i, T(j) must be larger than T(i).

2.If the situation is not described below ,no assertion about the distribution of the natural template of a dummy is HPF-conforming.

(a) If the alignment of the actual array axis with its template is collapsed, then * should appear in the distribution for the corresponding axis of the natural template of the dummy.

(b) If the actual array is aligned with the axis of its template by replication (or "replication-triplet") and that template axis is distributed * , then no entry should appear in the distribution for the natural template of the dummy.

(c) If the actual array is aligned with the axis of its template by int-expr and that template axis is distributed * , then no entry should appear in the distribution for the natural template of the dummy.

(d) If the alignment of the actual array axis with the axis of its template is subscript triplet l:u:s and that axis of its template distributed *, then * should appear in the distribution for the corresponding axis of the natural template of the dummy.

(e) If the alignment of the actual array axis with the axis of its template is subscript triplet l:u:s and that axis of its template distributed BLOCK(n) and LB is the lower bound for that axis of the template, then BLOCK(n/s) should appear in the distribution for the natural template of the dummy, provided that s divides n evenly and that l - LB < s.

(f) If the alignment of the actual array axis with the axis of its template is subscript triplet l:u:s and that axis of its template distributed CYCLIC(n) and LB is the lower bound for

discussion

Question continued:

(f) If the alignment of the actual array axis with the axis of its template is subscript triplet l:u:s and that axis of its template distributed CYCLIC(n) and LB is the lower bound for that axis of the template, then CYCLIC(n/s) should appear in the distribution for the natural template of the dummy , provided that s divides n evenly and that l - LB < s."

(g) If the alignment of the actual array axis with the axis of its template is subscript triplet l:u:s ,s must be positive.

Or it might be better to forbid the use of any assertion asterisks in DISTRIBUTE directive in case that a dummy argument doesn't have INHERIT attribute and the corresponding actual argument isn't ultimately aligned with itself since it seems that this solution makes things far simplified and cause little actual inconvenience (the same effect can also be achieved by ALIGN directive).

Discussion:

cci list: 05/03/95

item number 9 **title** Scopeing of attributes in Modules

submitted by S. Uzuhara **email** uzuhara@slab.sfc.keio.ac.jp **date** 10/20/94

group Group D **status** status: new **action:**

doc change text **doc change status** doc status not known

formal reply reply:

original Q

```
>   Question. Is the following code HPF conforming?
>
>     module ABC
>      real A(100,100)
>      ...
>     end module
>
>     module A_WITH_B
>     use ABC
>      real B(100,100)
>     !HPF$ align A(:,:) with B(:,:)
>      ...
>     end module
>
>     module A_WITH_C
>     use ABC
>      real C(100,100)
>     !HPF$ align A(:,:) with C(:,:)
>      ...
>     end module
>
>   ABC contains a variable A, and A_WITH_B and A_WITH_C use ABC.
>   A_WITH_B aligns A with B, but A_WITH_C aligns A with C.
>   (In ABC, no directive for A is specified).
```

discussion

>   A1. data objects arranged by HPF directives in a scoping unit
>   are local entity in that scoping unit, as objects
>   specified in type declaration statement? (if so, the above
>   code is wrong)
>
> Yes, this would be my interpretation. The intent of HPF is that ALIGN
> and DISTRIBUTE function as if they were Fortran 90 attributes. All the

Thank you so much for my question. This question program was derived from a code of NAS parallel benchmarks(90/HPF version). The author of the programs tested them by several F90 compilers and DEC F90 compiler which supports some HPF constructs (I'm not sure of DEC's compiler). So, I searched through my bible, "The High Performance Fortran Handbook"(MIT Press), for the description of such a scoping rule, but I couldn't find it.

> attributes of an object must be specified in the module that defines
> the object. So any ALIGN directive for A must appear in module ABC.
> (But module A_WITH_B could contain a REALIGN statement in its code,
> but only if module ABC contains a DYNAMIC declaration for A.)

REALIGN or REDISTRIBUTE statements can be specified in a module, since they are executable(?).

> --Guy

Thanks in advance,

--Shigeru

cci list: 05/03/95

item number 10 **title** Permutations in HPF

submitted by Kenth Engo **email** kenth@imf.unit.no **date** 01/11/95

group Group D **status** status: new **action:**

doc change text **doc change status** doc status not known

formal reply

original Q

I have a general question about the way HPF deals with permutations of data on the different parallel architectures.

In many applications in MIMD and SIMD computations today, one often encounter the need to just perform a permutation of the data distributed on the parallel computer, i.e. a one-to-one mapping of the data set onto itself. It is then very important that the routing of the data is done in such a fashion that traffic contetions are eliminated in the interconnecting network. One does not want a situation where 10 processors are communicating data to the very same processor. This would make all but one processor idle, since no more than one processor is allowed to communicate with the destination processor at the same time.

What I have in mind is examplified by the following HPF code:

```
 C 8 PROCESSORS AND AN ARRAY OF 32 ELEMENTS
 C
 !HPF$ PROCESSORS SEDECIM(8)
       REAL CENTURY(32)
 C
 C THE ARRAY IS DISTRIBUTED BY BLOCK.
 C
 !HPF$ DISTRIBUTE CENTURY(BLOCK) ONTO SEDECIM
 C
 C THE ELEMENTS ARE DISTRIBUTED WITH ELEMENTS 1,2,3,4 ON
PROC
 C #1, ELEMENTS 5,6,7,8 ON PROC. 2 AND SO ON.
```

Suppose one want to redistribute the elements during execution to the following arrangement

```
 C DATA REDISTRIBUTED CYCLIC ON THE PROCESSORS
 C
 !HPF$ REDISTRIBUTE CENTURY(CYCLIC) ONTO SEDECIM
 C
 C THE ELEMENTS ARE NOW REDISTRIBUTED WITH ELEMENTS
1,9,17,25 ON
 C PROC #1, ELEMENTS 2,10,18,26 ON PROC. 2 AND SO ON.
```

During this redistribution a permutation of the data set is performed. How is this permutation implemented, and what is actually done. Is there a strategy/theory for generally doing this permutation optimally, that is; without any traffic contetions in the network?

I will be grateful if someone could answer this email, and possibly send or give me references to literature or people where I can find out more about how HPF implements the permutations.

Best regards,

Kenth Engo

discussion

item number 11 **title** pointer with sequence

submitted by Henry Zongaro **email** zongaro@VNET.IBM.COM **date** 02/16/95

group Group D **status** status: new **action:**

doc change text **doc change status** doc status not known

formal reply

original Q

Hello,

Things have been quiet here lately, so I thought I'd send a few questions that I've been hoarding. All page and line references are relative to the 1.0 HPF Language Spec.

I'd like to hear other people's opinions on these, especially the 2nd and 3rd items.

1) The response to CCI item 6.3 indicated that variables with the POINTER attribute must be distributed or aligned. A related question - can they be given the SEQUENCE attribute? Or can a pointer be associated with both sequential and nonsequential targets?

discussion

cci list: 05/03/95

item number 12 **title** reference before definition

submitted by Henry Zongaro **email** zongaro@VNET.IBM.COM **date** 02/16/95

group Group D **status** status: new **action:**

doc change text **doc change status** doc status not known

formal reply

original Q

Hello,

Things have been quiet here lately, so I thought I'd send a few questions that I've been hoarding. All page and line references are relative to the 1.0 HPF Language Spec.

I'd like to hear other people's opinions on these, especially the 2nd and 3rd items.

2) This one is related to the response to CCI 7.3. I was wondering whether we need to tighten up the language even further. The response to 7.3 was to change part of the definition of specification expressions to the following:

> If a specification expression includes a reference to the value of an element of an array specified in the same specification-part, the array bounds must be specified in a prior declaration and there may be no subsequent static mapping directive affecting the mapping of the array.

With this change, we avoided references to the values of array elements before the array bounds have been fully specified and array mapping has occurred.

This text was based upon text in 7.1.6.2 of the Fortran 90 standard. The sentence before this in the standard states the following:

> If a specification expression includes a reference to an inquiry function for a type parameter or an array bound of an entity specified in the same specification-part, the type parameter or array bound must be specified in a prior specification of the specification-part.

The intent of this is that there should be no forward reference to array bounds, character lengths or kinds. In HPF we have situations in which there are implicit dependencies on array bounds, but which don't occur in specification expressions. For example,

```
!hpf$ distribute a(block) onto p
!hpf$ processors p(10)
      integer a(100)
```

The distribute directive has dependencies on the bounds of a and p, which don't appear in specification expressions. Similarly, in the following example, the align directive has dependencies on the bounds of a and b.

```
!hpf$ align a(:) with b(:)
      integer a(100), b(100)
```

Should we restrict entities from appearing in align or distribute directives before their bounds (if any) have been specified?

discussion

Rob replies ....
2. The Digital HPF/F90 compiler succesfully compiles programs in which array declarations occur after hpf mapping statments involving them. (I just tried it!) We should not make any restrictions along these lines.

and Henry replies ...

With due respect to Digital's product, I'd like to point out that vendors have been known to provide extensions before. I just wanted to make sure that we do
things that are in keeping with the spirit of the base language, Fortran 90. We should be very careful about advocating a change in the spirit of declarations processing (and I maintain that the fact that the following is invalid in Fortran 90

```
      integer b(ubound(a))
      integer a(10)
```

while the following might be valid in HPF

```
!hpf$ processors p(10)
!hpf$ distribute a(block) onto p
      integer a(10)
```

when both have forward dependence on the bounds of "a", is such a change in the
spirit of the base language.)

cci list: 05/03/95

item number 13 **title** scope of processor arrangements

submitted by Henry Zongaro **email** zongaro@VNET.IBM.COM **date** 02/16/95

group Group D **status** status: new **action:**

doc change text **doc change status** doc status not known

formal reply

original Q

Hello,
All page and line references are relative to the
1.0 HPF Language Spec.
I'd like people's opinions on these, especially the 2nd and 3rd items.

3) Consider the following example:

```
       module mod1
         integer :: i
       end module mod1

       module mod2
         use mod1
   !hpf$   processors p(i, number_of_processors()/i)
         integer a(100, 100)
   !hpf$   distribute a(block, block) onto p
       end module mod2

       program prog
         use mod1
         i = 2
         call sub
         i = 0
         call sub
       end program prog

       subroutine sub
         use mod2
       end subroutine sub
```

Is this example valid? The text on page 41, lines 28-37 states the following:

> Returning from a subprogram causes all processor arrangements declared local to that subprogram to become undefined. It is not HPF-conforming for any array or template to be distributed onto a processor arrangement at the time the processor arrangement becomes undefined unless at least one of the two conditions holds:
>
> o The array of template itself becomes undefined at the same time be virtue of returning from the subprogram.
>
> o Whenever the subprogram is called, the processor arrangement is always locally defined in the same way, with identical lower bounds, and identical upper bounds.

In Fortran 90, when a module is not being used anywhere in the current call chain, the variables in the module become undefined. Does the same thing happen with processor arrangements (and templates for that matter)? In the example above, after the program returns from the first call to sub, does the processor arrangement p become undefined? If so, on the second call to sub, are the bounds of processor arrangement re-evaluated with the new value of i, leading to a division by zero? I can think of two likely implementation strategies for processor arrangements and templates in the scope of a module, one of which would result in a division by zero, the other of which would always give p the same bounds as it had on the first use of the module - in this case p(2, number_of_processors()/2).

Fortran 90 doesn't have this problem, because arrays and character variables with non-constant bounds or lengths are not allowed to appear in the scope of a module.

discussion

Rob replies:
3. In Henry's example, on the second call to sub the processors arrangement will be given the shape [0, Infinity], which ought to bother at least some compi lers, but that wasnt his point.

I think this program is otherwise fine. The processors arrangement dies at the end of each call, together with the local that is mapped to it. The shape of p is different on each call. Now *this* should be illegal. Someone tell me, pleas e, why it already is.

```
      module mod1
      integer i
      data i/1/
      end module mod1

      module mod2
        use mod1
!hpf$     processors p(i, number_of_processors()/i)
         integer a(10,10), b(i, 2*i)
!hpf$     distribute a onto p
      end module mod2

       program wierd
      use mod2
      use mod1
      i = 2
      call sub
      i = 3
      call sub
      end

      subroutine sub
        use mod2
      end subroutine sub
```

Note that the problem is a Fortan 90, not an HPF one. What is the shape of b in sub?
----
and Henry replies ....
The above isn't a valid Fortran 90 program. The bounds and length specifications in the scoping unit of a module must be constant specification expressions, so your declaration

```
     integer a(10,10), b(i, 2*i)
```

is not valid. However, I expect that it was the intent of HPFF that the bounds specifications of processor arrangements be allowed to be non-constant in the scoping unit of a module. For example,

```
        module mymod
     !hpf$   processors p(number_of_processors())
        end module mymod
```

seems desirable. But then that opens the door to my original example,

```
        module mod1
          integer :: i
        end module mod1

        module mod2
          use mod1
     !hpf$   processors p(i, number_of_processors()/i)
        end module mod2
```

(this is just a preliminary response)

cci list: 05/03/95

item number 14 **title** multiple directives on a line

submitted by Henry Zongaro **email** zongaro@VNET.IBM.COM **date** 02/16/95

group Group E **status** status: new **action:**

doc change text **doc change status** doc status not known

formal reply

original Q

Hello,
All page and line references are relative to the
1.0 HPF Language Spec.

4) Can two HPF directives appear on a single line separated by ';'?
For example,

```
!hpf$ processors p(10);  template t(100)
```

From the syntax on page 19, lines 25-43, the line above appears to be invalid, but I just wanted to make certain this was the intent.

5) The constraint on page 19, lines 44-45, states "An hpf-directive-line cannot be commentary following another statement on the same line."
What about following another directive? For example, was it intended that the following should be valid?

```
!hpf$ processors p(10)   !hpf$ template t(100)
```

discussion

cci list: 05/03/95

item number 15 **title** equivalence and nonlinear mappings

submitted by Yasuharu Hayashi **email** hayashi@lang1.bs1.fc.nec.co.jp **date** 02/22/95

group Group D **status** status: new **action:**

doc change text **doc change status** doc status not known

formal reply

original Q

discussion

I think that HPF is designed to be able to describe all the mapping by alignment and distribution in principle.
But for example;

```
     REAL B(10)
     COMPLEX A(5)
     EQUIVALENCE (A,B)
!HPF$ PROCESSORS P(4)
!HPF$ TEMPLATE T(16)
!HPF$ ALIGN A(I) WITH T(3*I-1)
!HPF$ DISTRIBUTE T(BLOCK) ONTO P
      :
```

In this case,

P(1) has B(1),B(2).
P(2) has B(3),B(4),B(5),B(6).
P(3) has B(7),B(8).
P(4) has B(9),B(10).

The mapping of B is unable to be described by any alignment and distribution because of the nonlinearity of the alignment of B.
Why isn't it constrained to map the variable associated by EQUIVALENCE with other variables whose length of the element is different from it?

item number 16 **title** document update completion

submitted by Henry Zongaro **email** **date** 02/26/95

group Group not assigned **status** status: new **action:**

doc change text **doc change status**

formal reply

original Q

discussion

This item is a place holder to be expanded and assigned to groups later when the associated CCI items for HPFF94 are added in. Henry has observed that not all of the CCI items made it into the document correcly, or completely. Next round of edit fixes needs to include resolving these. -mz-

----

I've got the following list which I believe to be incomplete. As you can see they're scattered about the document. Some of them are things that I think were partially dealt with. For a few, I couldn't tell whether the required change had been made.

1.2 a) Example font was not repaired - Section 4 (pp. 63-64, 70-73)

3.4 b) Parts of constraints in 4.3.1.1 and 4.3.1.2 were never deleted. (p. 75, lines 44-45; p. 79, lines 10-11)

5.1 Text in Section A (p. 165, line 30) was not deleted.

5.5 a) Section A - I couldn't tell whether this change was applied.

5.8 This change to Section A was completed, although the subsection numbering isn't consistent with the numbering of the other HPF_LOCAL_LIBRARY procedures.

6.3 This change to Section 3 was incomplete in that 3.6, which deals with mappings of pointers and allocatable arrays still indicates that pointers can appear as alignees or distributees.

6.10 and 6.12 Changes to syntax and constraint for INDEPENDENT in Section 4.4 were not applied. See page 83, lines 31, 44, 45.

7.9 The term "regular array section" was replaced in Section 3.10 (p. 46, line 21), but not in Section 3.9 (p. 45, line 35).

7.12 The grammar change that allowed constants to be sequential was applied to Section 7.1.3 (p. 151, line 46), but most of the chapter talks about variables rather than data objects.

8.2 I'm not sure whether the change that would define "implementation dependent" was applied to Section 1 or 2. (I couldn't spot it, but I'll admit I didn't look too, too hard.)

9.6 A constraint should have been added to Section 7.1.3 to prohibit the appearance in a single scoping unit of more than one sequence directive with no association-name-list. I think the constraint added (p. 152, line 4-5) was mistyped.

10.6 I'm not sure whether it was decided to more fully describe the interfaces of GLOBAL_ALIGNMENT, GLOBAL_DISTRIBUTION and GLOBAL_TEMPLATE, in Section A.2.3.2 - A.2.3.4 (pp. 169 - 170).

Thanks,

Henry

cci list: 05/03/95

item number 17 **title** Defaults for distribution

submitted by A.C. Marshall **email** adamm@liv.ac.uk **date** 04/18/95

group Group D **status** status: new **action:**

doc change text **doc change status** doc status not known

formal reply

original Q

Forgive me for being dim (and only having v1.0 of the draft standard) but...

Looking at the syntax rules for DISTRIBUTE (p24 & 26) it would appear to me that:

```
!HPF$ DISTRIBUTE A(BLOCK) !H303/5/8
!HPF$ DISTRIBUTE ONTO P :: A !H301/2/6/10
```

are valid but that

```
!HPF$ DISTRIBUTE A
!HPF$ DISTRIBUTE :: A
```

are not. Is this just me or is this how things are supposed to be, and if so why is it not possible to use default distribution and processor grid in the same statement, after all

```
!HPF$ PROCESSORS P(NUMBER_OF_PROCESSORS())
!HPF$ DISTRIBUTE ONTO P :: A
```

is valid and has the same effect.

Adam Marshall

discussion