item number 18 **title** Defined assignment in FORALL

submitted by Henry Zongaro **email** zongaro@vnet.ibm.com **date** 05/04/95 **updated** 7/12/95

group Group C **status** status: in progress **action:** Action Chuck, Guy, Henry, Jerry

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

formal reply

original Q

Hello,

I was wondering whether there's not a problem with allowing defined assignment to appear within a FORALL. Consider the following example.

```
module mod
  integer :: a(3) = (/1,2,3/)
contains
  pure subroutine def_assign(lhs, rhs)
   integer, intent(inout) :: lhs
    character, intent(in) :: rhs

    lhs = a(ichar(rhs)+1)
  end subroutine def_assign
end module mod

program p
 use mod
  interface assignment(=)
    module procedure def_assign
  end interface

  forall (i = 1:2) a(i) = char(i)    ! A sneaky way of passing "i"
                                     ! to def_assign
end program p
```

The rules of forall specify that the right-hand side and the indices of the left-hand side are evaluated, in any order, prior to assignment, which also takes place in any order. In the above example, we have

```
a(1) = char(1)
a(2) = char(2)
```

as the two defined assignments which take place. Inside of def_assign, there's a host-associated reference to a, so what ends up happening is the following:

```
a(1) = a(2)
a(2) = a(3)
```

The order in which these assignments occurs affects the result. The value of a after the forall statement is executed could be (/2,3,3/) or (/3,3,3/).

Basically, the problem is that in defined assignment, completely evaluating the right-hand side for all active combinations does not necessarily let the compiler precompute everything which might also appear on the left-hand side.

Thanks,

Henry

s a

discussion

DISCUSSION AT MEETING:

CCI #18 ...Semantics of a forall are to evaluate all rhs and then store in all lhs, but if the assignment operator is user defined this is under user control, not compiler control. The user's definition might make the order of assignment important. Guy queried how an array assignment was handled in this case. Jerry will take this question to X3J3 about the status with respect to elemental functions. Guy pointed out that for WHERE F90 forbids the defined assignment. Chuck presented a proposal that sounded promising: that the evaluation is as if the rhs were assigned into a temp using the defined assignment operator and then the lhs is a direct copy of the already evaluated values. There was discussion of issues like the type of the temp (same as lhs or rhs?). Action Chuck, Guy, Henry, Jerry to circulate proposed wording for this definition.

item number 25 **title** Use of pure and independent

submitted by Matt Rosing **email** m_rosing@pnl.gov **date** 06/27/95 **updated**

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

doc change text **doc change status**

formal reply

original Q

I have some questions about HPF semantics and implementation. These are based on the HPF definition I have (dated 5/93).

I'm trying to determine if HPF can be used to implement a code we have and I have questions about spmd execution within HPF. The code has two phases, the first fits the data parallel model very well and the second does not. Within the second phase, independent operations are done on sections of a distributed array. These operations modify the array.

There are apparently two methods to describe non-data parallel code in HPF. This includes functions and subroutines declared pure, and loops declared independent. There are about five pages of constraints on the use of pure functions and it's not clear that I can use them. The main reason I believe they won't work is that pure functions are not allowed to have side effects and our code has each independent operation modifying a part of a distributed array. Is it true that pure functions can't be used for this?

Independent do loops, however, appear to have fewer constraints. The only constraint seems to be that multiple loop iterations can not write to the same location (I'm ignoring IO). I have a few questions to see how far I can push this:

1) Can an iteration modify data belonging to another processor?

2) If so, how do implementations handle the one-sided nature of the communication generated by this? The reason I ask is that, if done naively by buffering remote writes until the loops are done, this could require as much space for buffers as there is space for the data structure being operated on.

3) Is there any limitation on the types of routines that can be called from within an independent loop?

4) How is the scheduling done? If there are just a bunch of subroutine calls within the body of the loop, how does the compiler figure out which processor does what?

5) Is it possible to dynamically schedule the loops on the processors for load balancing? Although none of the iterations would interfere, the scheduling mechanism would.

Some of these questions are probably outside the scope of the language definition but I would still appreciate answers from any implementors. The reason being that the resulting performance might depend heavily on the implementation.

Thanks,

Matt
(m_rosing@pnl.gov)

s a

discussion

Chuck Replies 6/29/95
There is a new draft of the HPF spec, available through http://www.erc.msstate.edu/hpff/home.html. The changes are minor (basically, corrections and clarifications) from the version you're looking at.
The entire purpose of PURE is to ensure that those functions do not have any side effects. So, at face value, this is correct.
You could rewrite the functions to return arrays (this assumes each function modifies one array). You could then call the PURE functions, assigning the returned values into the separate array sections. For example, the final code calling the functions might look like

```
FORALL ( I=1:NUM_BLOCKS )
  X(ILOW(I):IHI(I)) = FOO( A, B, C )
END FORALL
```

This would probably mean a lot of modification for the code, and it might break the current generation of compilers. But it would be one way to use PURE in this context.
>1) Can an iteration modify data belonging to another processor?
Yes. Distribution of data has no effect on the semantics of the program.
Note that HPF has no notion of the processor that an iteration executes on. (Of course, the underlying compiler should have some such idea...)

>2) If so, how do implementations handle the one-sided nature of
....
As you suggest below, this is something outside the scope of the language definition. Very good compilers will do something efficient (like strip-mining the loops) to ensure reasonable-sized buffers. Bad compilers will run out of memory and generate core dumps.

>3) Is there any limitation on the types of routines that can be called
>from within an independent loop?
Not in the language. The only limitation is in how called routines behave, i.e. that they don't violate the independence conditions.

>4) How is the scheduling done? If there are just a bunch of subroutine
....
As with 2 above, this is beyond the scope of the language definition. You are correct that this will be difficult to do well.
Incidentally, discussions of specifying scheduling mechanisms for parallel loops are going on in the HPFF 2 meetings. If you have opinions on the subject (and I know you do, Matt :-), please feel free to contribute them. The final language will be better for it.

>5) Is it possible to dynamically schedule the loops on the processors
....
A dynamic scheduling mechanism would be a valid HPF implementation. (Of course, it is not the only one.) Independence is not a problem here because any interference is between system variables; INDEPENDENT only makes an assertion about the user code.
Again, discussion in the HPFF 2 meetings may be relevant to this question in the future.
=====Rik Littlefield replies 6/29/95=========
Matt,

You write:
> 2) If so, how do implementations handle the one-sided nature of
....
If you're talking about the electronic structure codes, say creating the Fock matrix, then the problem is even worse. There are only O(N^2) data elements but something like O(N^4) updates to those elements. (At best, I think it's O(N^3) with clever tricks.)

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 **updated** 7/13/95

group Group C **status** status: waiting for text **action:** Rob will clarify processor shape

doc change text **doc change status** doc text needed

formal reply No language change needed, but RSS will clarify Procssor_Shape

original Q **discussion** **s** d

>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

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.

====
May meeting - subgroup discussed, No language change. Rob will clarify processors_shape

item number 3 **title** Pure / Allocatable

submitted by Henry Zongaro **email** zongaro@vnet.ibm.com **date** 03/30/95 **updated** 7/13/95

group Group C **status** status: waiting for text **action:** chk to prepare text change

doc change text **doc change status** doc text needed

text needed.

formal reply ALLOCATE and DEALLOCATE will be illegal in a pure procedure

original Q

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?

s d

discussion

Minutes from meeting ...

CCI#3 - forall (i =) a(i) = f() where f is pure but does an allocate with "stat = " that might fail. If there were no stat= it would be ok , the program would stop, but with stat, it might return a different result. Two proposals were discussed:
(a) "stat" in allocate is illegal in a pure procedure (b) allocate is illegal in a pure procedure (since the user can, after all, use a local variable). This was amended to be that ALLOCATE and DEALLOCATE are both illegal in a pure procedure. This proposal passed 20-0-6. Chuck will prepare draft language for the document.

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

submitted by Henry Zongaro **email** zongaro@vnet.ibm.com **date** 03/30/95 **updated** 7/13/95

group Group C **status** status: waiting for text **action:**

doc change text **doc change status** doc text needed

formal reply reply: no change - CHK will add note.

original Q **s d**

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.

discussion

May meeting subgroup discussion:
No change - Chuck will add note..

item number 1 **title** Partial processor shape

submitted by Eiji Nunohiro **email** nunohiro@hisoft.soft.hitachi.co.jp **date** 01/20/95 **updated** 7/13/95

group Group C **status** status: resolved **action:**

doc change text **doc change status** no doc change needed

formal reply resolved - no change required

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 **s z**

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.)

=====
May meeting - subgroup discussed - no change required.

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 **updated** 07/12/95

group Group D **status** status: in progress **action:** is there an action item?

doc change text **doc change status** no doc change needed

formal reply CCI # 10 - not cci but a request for implementation practice.

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 **s a**

Notes from May meeting
CCI # 10 - not cci but a request for implementation practice.

item number 11 **title** pointer with sequence

submitted by Henry Zongaro **email** zongaro@VNET.IBM.COM **date** 02/16/95 **updated** 7/13/95

group Group D **status** status: in progress **action:** needs more research

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

formal reply

original Q

s a

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

Meeting minutes: needs more research

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 **updated** 7/12/95

group Group D **status** status: in progress **action:** "think about this one"

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?

s a

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.)

FOR MORE CLARIFICATION / DISCUSSION, SEE #27

============
REPORT AT THE MAY HPFF MEETING:
CCI # 12 is a related implicit forward reference. The subcommittee thought that the user should get the statements in the right order. This is somewhat different than the compiler-sort-it-out. The group recommended that we think about this.

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

submitted by Henry Zongaro **email** zongaro@VNET.IBM.COM **date** 02/16/95 **updated** 7/13/95

group Group D **status** status: in progress **action:** needs more research

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.

s a

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 17 **title** Defaults for distribution

submitted by A.C. Marshall **email** adamm@liv.ac.uk **date** 04/18/95 **updated** 07/11/95

group Group D **status** status: in progress **action:** needs more research

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

s a

discussion

notes from May meeting: needs more research:

Scott Baden and Chuck Koelbel reply 07/07/95

This is the way things are "supposed to be"
Consult the relevant text (page 30, line 20-21, v. 1.1)

... To prevent syntactic ambiguity, the dist-format-clause
must be present in the statement form [of a distribute spec]

Chuck Koelbel adds that the "syntactic ambiguity" referred to here is due to the problem of non-significant blanks in Fortran:

Consider

```
>
> !HPF$ DISTRIBUTE PRONTO ONTO LOGY
> !HPF$ DISTRIBUTE PR ONTO ONTOLOGY
>
```

or the following example:

```
>
> !HPF$ ALIGN TWITHEADS WITH A
> !HPF$ ALIGN T WITH EADSWITHA
>
> Disallowed for the same reason.
```

Chuck also continues:

```
>
> It's not clear that his example has "the same effect". For example,
> consider
>
> !HPF$ PROCESSORS P(NUMBER_OF_PROCESSORS())
> !HPF$ PROCESSORS Q(4,NUMBER_OF_PROCESSORS()/4)
> !HPF$ DISTRIBUTE :: A
> Does A have a 1-dimensional or 2-dimensional distribution? (Yeah, this
> assumes that NUMBER_OF_PROCESSORS() is divisible by 4...)

>
> The reason for requiring at least one of the clauses is, "What information
> are you giving if you leave both out?" In effect,
>
> !HPF$ DISTRIBUTE :: A
>
> would be a no-op, and we didn't think there was a need for that.
```

Scott Baden
Chuck Koelbel

====

cci list: 05/09/95

item number 19 **title** Mapping function results

submitted by Henry Zongaro **email** zongaro@vnet.ibm.com **date** 05/04/95 **updated** 7/13/95

group Group D **status** status: in progress **action:** needs more research

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

formal reply

original Q

s a

I have a couple of questions related to specification of mappings for function results.

1) Consider the following program fragment

```
      program prog
        interface
          function f()
            integer f(100)
!hpf$       processors p(number_of_processors())
!hpf$       distribute f(block) onto p
          end function f
        end interface

        call sub(f())
      end program prog

      subroutine sub(i)
        integer :: i(100)
!hpf$   processors p(number_of_processors())
!hpf$   distribute i *(block) onto *p
      end subroutine sub
```

Is the above HPF conforming? Does distribution of a function result variable affect the distribution of the expression returned? The text on page 53, lines 27-28 indicates that the alignment of an expression is, in general, unpredictable, except in the case of arrays and array sections, so I believe the answer to my question is "No". However, this is actually spurred by another question relating to the SEQUENCE directive.

According to page 151, line 47 of the 1.1 HPF Spec., an <association-name> can be a <function-name>. When I first read this, I thought the explicit reference to <function-name> was there to include result variables. Now a co-worker has suggested an alternate interpretation, and we were wondering which is correct. Her suggestion was that this is trying to allow something like the following:

```
      program p
        integer, external :: f
!hpf$   sequence :: f

        i = f()
      end program p
```

Is this correct? Will this make the result of the function sequential? If so, that brings up another question:

```
      program p
        interface
          function f()
            integer :: f(10)
!hpf$       sequence :: f
          end function f
        end interface

        call sub(f())
      end program p

      subroutine sub(a)
        integer a(2, 5)
!hpf$   sequence a
      end subroutine sub
```

According to page 155, lines 33-35, an array valued expression cannot be specified to be sequential, but if specifying f to be a sequential function makes its result value sequential, this would be a contradiction.

discussion

May meeting minutes:
needs more research

cci list: 7/19/95

item number 21 **title** Question about derived type mappings and documentation.

submitted by Michael Hennecker **email** hennecke@rz.uni-karlsruhe.de **date** 05/11/95 **updated**

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

doc change text **doc change status**

formal reply

original Q

Hello,

I have some questions regarding data mapping of objects of derived type:

(1) Is it possible to DISTRIBUTE / ALIGN objects of derived type,
or are the data mapping attibutes restricted to intrinsic types?

(2) If mapping of objects of derived type is not possible, shouldn't the v1.0
and v1.1 specs for HPF_ALIGNMENT, HPF_DISTRIBUTION and
HPF_TEMPLATE

"ALIGNEE may be of any type." (5.7.15, 5.7.16)
"DISTRIBUTEE may be of any type." (5.7.17)

read "may be of any intrinsic type." ?

Best regards,
Michael

s a

discussion

Reply by CHK 5/15/95
...
It is possible to map objects of derived type.
(It is currently not possible to map components of derived type objects;
this is being discussed in the HPFF 95 meetings.)

Second question is moot, given the first answer.

Thanks for asking.

Chuck Koelbel

cci list: 7/19/95

item number 22 **title** Implementor note about processor distributions?

submitted by Henry Zongaro **email** zongaro@VNET.IBM.COM **date** 6/15/95 **updated**

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

doc change text **doc change status**

formal reply

original Q **discussion** **s a**

Hello,

We came across something that didn't seem immediately obvious here, and might not be immediately obvious to others, so we were wondering whether a note
to users and/or implementers might be justified.

On page 30 of the 1.1 Language Spec., it's stated that if the ONTO clause of a DISTRIBUTE directive is omitted, an arbitrary processor arrangement is chosen for each distributee. In some cases, there may be no suitable arrangement; I assume such a program would not be HPF-conforming. For example,

```
      program p
        integer :: a(10, 10)
!hpf$   distribute a(block(5), block(5))
      end program p
```

Here, the processor arrangement created would have to have an extent of at least two in each dimension (which, by the way, constrains how arbitrary the selection of a processor arrangement can be), so this program could not run in an environment in which the number of processors was fewer than four.

Does a note seem worthwhile here, or do others feel such a case is immediately obvious?

Thanks,

Henry

item number 23 **title** Regularize section 3 wording: object/array/template

submitted by Michael Hennecke **email** hennecke@rz.uni-karlsruhe.de **date** 05/15/95 **updated**

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

doc change text **doc change status**

formal reply

original Q

Hello!
I would suggest to use the words object, array and template in HPF v1.1
section 3 more consistently. Formally we have:

```
 H307  <distributee>    is  <object-name>
                  or  <template-name>

 H316  <alignee>        is  <object-name>

 H321  <align-target>   is  <object-name>
                  or  <template-name>

 H336  <template-name>  is  <object-name>
```

A. Comments concerning formal syntax rules:

1. By virtue of H336, the ``or'' clause in H307 and H321 seems to be
redundant: although TEMPLATEs are not <object>s (since they only =
appear in F90 comments), the <template-name>s are equivalent to =

``real'' <object-name>s.

Proposed edits:
o delete ``or <template-name>'' in H307 and H321

2. Is it intended in H316 that <alignee> shall not be a TEMPLATE?
If yes, I would add a constraint that forbids that such objects
be of ``type'' TEMPLATE (in view of comment A.1).

Proposed edits (if yes):
o Add to H316:
``Constraint: an <object-name> mentioned as an <alignee> =

shall not have the TEMPLATE type specifier.''

B. Comments concerning the section text:

1. The whole section uses ``array'' and ``object'' more or less freely
interchangeable. But ``object'' may mean ``scalar object'' as well
as ``array object''. In my understanding, it is _possible_ to =

distribute scalars (though perhaps not _useful_ since nothing more
than the default replication can be specified, right?).
=

Proposed edits:
o section 3.0 should explain these relationships:
formally we distribute a ``data object'', abbreviated ``object'',
meaning ``array'' or ``scalar'', being (most) useful for "array"=
=2E
o all those occurrences of ``array'' should read ``object'' in the
following subsections. =

Alternatively, section 3.0 may say that the later text always spea=
ks
of ``array'', but means also ``scalar''. In this case change all =

those ``object''s to ``array''s. (1st alternative preferred.)
o except 3.0, replace all ``data object''s with ``object''s.
o figure in 3.1, very left circle: ``other objects'' can only
be ``scalars'' (see F90 2.4.3.1).
Either substitute this or replace the whole text with ``objects''.=

s a

discussion

CONTINUTATION OF ORIGINAL QUESTION:
2. The problem that TEMPLATEs are _treated_ as local entities of class (=
1)
but cannot _be_ true <object>s (since they are only comments) means
they need special treatment. In effect, every instance of ``object''
as a <distributee>, <alignee> (?) or <align-target> should also say
that ``template'' is equally possible. (This does not apply to the
names if my comment A.1 is right.)
Discussing TEMPLATEs, it should also be made clear (as with comment B=
=2E1)
that they, too, may be scalar (is this right?), and that it is even m=
ore
useless than with objects to have them...

Proposed edits:
o section 3.0 should explain this wording problem with TEMPLATEs, an=
d
say that writing ``object=B4=B4 later in the text always is to be
interpreted as including ``template''. (Beware comment A.2.)
o delete ``template'' where it _is_ mentioned together with ``object=
''.

C. Related comments on <object>s' TYPE:

The role of intrinsic versus derived types is not discussed in great
detail (though formally well-defined). It should be explained at the =

beginning of the section, explicitly saying that distribution of objects=

of derived type is possible, and useful only with _arrays_ of derived ty=
pe
because it is not possible in HPF v1.1 to distribute structure component=
s.

Best regards,
Michael Hennecke

PS: I'd volunteer to submit a draft containing these edits,
should HPFF agree with my interpretations/comments.

==

item number 24 **title** Mapping directives with module variables

submitted by Eiji_Nunohiro **email** nunohiro@hisoft.soft.hitachi.co.jp **date** 06/26/95 **updated**

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

doc change text **doc change status**

formal reply

original Q

Dear, HPFF members,

The next questions are about HPF Language Specification Version 1.1

1. mapping directive of module variable

According to p.26,line 17, the DISTRIBUTE directive may appear only in the specification-part of a scoping unit. Judging from it, static mapping directive of module variable can only be written in the module. Is that right ?

```
     MODULE MD1
     REAL a(100),b(100)
!HPF$ DISTRIBUTE a(BLOCK)  ! legal
     END MODULE

     USE MD1
!HPF$ DISTRIBUTE b(BLOCK)  ! illegal
```

2. scope of mapping directive of module variable

Assume that a variable in a module is distributed by a directive in the module and declared to be dynamic. Then in a subroutine, the module variable is remapped. If another subroutine is called and the routine uses the module, what is the mapping of the variable ?

```
     MODULE MD2
     REAL a(100)
!HPF$ DISTRIBUTE a(BLOCK)
!HPF$ DYNAMIC a
     END MODULE

     SUBROUTINE SUB1
     USE MD2
     ...
!HPF$ REDISTRIBUTE a(CYCLIC)
     CALL SUB2
     ...

     SUBROUTINE SUB2
     USE MD2
     ...
     a(i)=...   ! is the mapping of A BLOCK or CYCLIC ?
```

If the mapping of A in SUB2 is CYCLIC is right, the compiler should not use the mapping information in the module when compiling SUB2. In other words, mapping of A in SUB2 is "DISTRIBUTE a * onto *". Is that right ?

Sincerely, Dr. Eiji Nunohiro.

s a

discussion

From Henry Zongaro 6/27/95
about question 2::
The DYNAMIC directive has two constraints which prohibit members of COMMON
blocks and objects with the SAVE attribute from being given the DYNAMIC attribute. I've wondered why objects declared in the specification part of a module do not have the same restriction.

A typical way of implementing module data is to place the public data into something like COMMON blocks - perhaps one large block for all of the objects,
or one block per object. Similarly, implementations probably often place private module data in static storage - effectively treating it as if it had the SAVE attribute. Presumably, whatever makes it difficult to handle DYNAMIC for objects in COMMON or with the SAVE attribute, probably makes it difficult
to handle for objects declared in the scoping unit of a module.

So, should data objects declared in the scoping unit of a module be prohibited from having the DYNAMIC attribute?

Thanks,
Henry
=======
Rob replies to Henry's reply:

I dont think so! The declaration of the module data is visible to all who use it, so the fact that it is dynamic is unmistakable. Not so with common blocks that are declared locally. I dont understand why SAVE and DYNAMIC are exclusive. Can someone remember?

Rob
=========
And Henry replies:
But objects in common can be explicitly mapped or given the SEQUENCE attribute, even though they are declared only locally. Using the language of Section 7 of the HPF Language Spec., every occurrence of a nonsequential COMMON block must have exactly the same number of components. If a component
is nonsequential in one occurrence it must be nonsequential and have indentical type, shape and mapping attributes in every component. If a component is sequential and explicitly mapped in one occurrence, it must be sequential and explicitly mapped with the same mapping attributes in every occurrence.

After placing all those other requirements on a user, requiring that, if a component is dynamic in one occurrence of a COMMON block, it be so in all occurrences doesn't seem so unreasonable.

If dynamic objects can be handled in the specification part of a module, it should be possible for implementations to use the same (or a very similar) mechanism to handle dynamic objects in common.

cci list: 07/19/95

item number 26 **title** Conditional realignment

submitted by Fabien COELHO **email** coelho@chailly.ensmp.fr **date** 07/01/95 **updated**

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

doc change text **doc change status**

formal reply

original Q

Hi out there,

Is this kind of thing allowed in HPF ?

```
! align A with T
  if (some runtime condition)
! realign A with T'
  endif
! redistribute T ...
```

after the redistribution, array A mapping is not known. It depends on the runtime condition. I cannot remember anything that may forbid this. I guess it is not very nice for the compiler...
should/could be forbidden ? Or I may be wrong ?

Fabien.

discussion **s a**

Chuck Koelbel replies 7/3/95:
Yes, this is allowed. This was the intent of HPF REALIGN and REDISTRIBUTE
- to allow the user to make run-time decisions about data mapping. Allowing run-time remapping will indeed require substantial support in the run-time system. We discussed this tradeoff, and the concensus was that users had valid reasons for wanting this capability, therefore it should be in the language. The difficulties with implementation were one reason that REALIGN and REDISTRIBUTE were not put in Subset HPF.

In short, you are right that this is legal and hard to implement. You are wrong that it is/should be forbidden.

Chuck

item number 27 **title** Order of mappings

submitted by Will Denissen **email** den-wja@tpd.tno.nl **date** 07/04/05 **updated**

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

doc change text **doc change status**

formal reply

original Q

I can remember that at our last HPF-meeting, there was a cci (12 reference before definition) on the declaration order of HPF mappings. As far as I can remember we decided that mapping specifications may not contain forward references. So the program below WAS correct before the cci vote, and IS incorrect after the cci vote.

```
      SUBROUTINE ONE (D, E)
      INTEGER, DIMENSION (3), INTENT (IN)   :: D, E

 !HPF$ ALIGN D(J) WITH E(J)

 !HPF$ DISTRIBUTE E (BLOCK)


      print *, D(3)

      END SUBROUTINE ONE
```

I'm I right?

Best regards,

-- Will Denissen.

================
BEGINNING OF DISCUSSION
================
From David Presberg 07/05/95

HPFF Folks --

I read with some concern Will Denissen's request for a "stiff" interpretation of an order-of-appearance issue concerning HPF directives among other HPF directives. He refers to actions at the May 95 meeting which I did not attend.

Concerning the substantive issue in Will Denissen's posting, my reading of Henry Zongarro's CCI request focussed on the order of Fortran declarations (and thus specific compiler-determinable numbers that are parameters describing array dimension bounds, etc.) versus some HPF directive that needs information from the declaration to "elaborate" the directive. I did _not_ read into Henry's question any discussion of the particular ordering among the HPF directives themselves, so long as, taken as a whole, they are symbolically "well founded". If Will's report that "mapping specifications may not contain forward references" is an accurate reflection of your May deliberations in that you determined (paraphrasing Henry's question) "... we restrict entities from appearing in align or distribute directives before their bounds (if any) have been specified", I regret that I was not there to argue against that decision. (See below for the basis of an objection.) But to further assume that this decision imposes an ordering within HPF directives seems to be unduly and unwisely extending that interpretation.

s a

discussion

Discussion continue

Can someone please comment on the actual May actions and the debate?

-- Pres

p.s. The reasons for _not_ imposing "declarative" ordering restrictions on HPF directives or related Fortran declarations that all are within the immediate scope of a single Fortran compilation unit is that this is a significant departure from common Fortran practice. It may be burdonsome on some compiler writers, but it has long been a known Fortran compiler technology to determine the type, kind, etc., status of some variables from evidence collected from the entire compilation unit, without regard to the order in which that evidence appeared (so long as it followed statement-form ordering rules well described in all Fortran standards). That is, one might supply the type of a variable long before (in appearance of statements) it was declared an array, or vice versa. (There are even some obscure cases in which the "final" status of some identifier is only determinable at the very end of the text of the compilation unit.) Yes, this "order of declarative reference is irrelevant; collect evidence over the entire module" form of language definition is archaic and contributes to how "interesting" Fortran compiler writing can be, but, no, I don't believe any Fortran standards committee has attempted to write it out of the language, yet.

Thus, I would have argued strongly against any such ordering constraints in response to Henry Zongarro's CCI question (#12), and will argue even more strongly against Will Dennison's extension, if that has not already been decided by HPFF in May.
=================

Chuck Koelbel replies to Pres 7/07/05
Your reading is consistent with my interpretation of Henry's original question, and my memory of the HPFF discussion. In short, the restrictions that were agreed to were there to prevent "circular" orderings such as this:

```
!HPF$ DISTRIBUTE A( CYCLIC(B(10)) )
!HPF$ DISTRIBUTE B( CYCLIC(A(10)) )
```

They do not say anything about the other (benign) case that Will presents:

```
!HPF$ ALIGN D(J) WITH E(J)
!HPF$ DISTRIBUTE E (BLOCK)
```

I do not have the meeting notes (see above...), but I believe that the restriction is against using the _value_ of an object before its bounds and distribution were declared. This is similar in spirit and effect to the existing Fortran rules for PARAMETER declarations. For example:

```
PARAMETER ( N = 2*M )
PARAMETER ( M = N-1 )
```

This is not allowed (although one can clearly see that N=2, M=1 :-). The point you made in your PS was also brought up in the course of the debate, and was persuasive.

Chuck

item number 7 **title** INHERIT dummy shape

submitted by Yasuharu Hayashi **email** hayashi@lang1.bs1.fc.nec.co.jp **date** 02/27/95 **updated** 7/12/95

group Group D **status** status: in progress **action:** needs an answer sent

doc change text **doc change status** no doc change needed

formal reply ok as is - anther case of not being able to talk about an inherited template of different rank.
QUESTION - MZ- Does this introduce a new requirement somewhere?

original Q s a

> 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:
> example from original repeated here ...
> :
> 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

===========

Notes from May HPF meeting:
CCI #7 - ok as is - anther case of not being able to talk about an inherited
template of different rank.

============

Comment from Mary ... I left this “in progress” ... wondering (a) is this a case
of a cci that should produce an action for HPF2 language change and
(b) if it is ... how do we want to keep track of these?

item number 8 **title** Dummy assertion asterisk

submitted by Yasuharu Hayashi **email** hayashi@lang1.bs1.fc.nec.co.jp **date** 04/25/95 **updated** 7/13/95

group Group D **status** status: in progress **action:** study and discuss again in July

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.

s a

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: Rob replies ...

This example is nonconforming because axis 2 of B is NOT coextensive (one-to-one and onto mapping to) axis 1 of the template to which B is aligned.

> ... example from original message

Now let the example be this:

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

This example is correct. The replication over the second axis of the template of the actual is not a problem because that is an axis whose distribution format is *. B is not coextensive with that axis because it has a one-to-many association with it, but since the template axis has a * distribution, coextension is not a requirement. Is this reasonable?

Rob Schreiber

=============

meeting discussion

CCI #8 ...Semantics of a forall are to evaluate all rhs and then store in all lhs, but if the assignment operator is user defined this is under user control, not compiler control. The user's definition might make the order of assignment important. Guy queried how an array assignment was handled in this case. Jerry will take this question to X3J3 about the status with respect to elemental functions. Guy pointed out that for WHERE F90 forbids the defined assignment. Chuck presented a proposal that sounded promising: that the evaluation is as if the rhs were assigned into a temp using the defined assignment operator and then the lhs is a direct copy of the already evaluated values. There was discussion of issues like the type of the temp (same as lhs or rhs?). Action Chuck, Guy, Henry, Jerry to circulate proposed wording for this definition.

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 **updated** 7/13/95

group Group D **status** status: waiting for text **action:** example needed

doc change text **doc change status** doc text needed

formal reply CCI # 15 Interesting! No language change but add an examples.

original Q **s** d

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?

discussion

From May meeting minutes:
CCI # 15 Interesting! No language change but add an examples.

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 **updated** 07/12/95

group Group D **status** status: waiting for text **action:** ???who

doc change text **doc change status** doc text needed

formal reply CCI # 9 - a textual change is needed - directives must be in the same module scoping unit.

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 **s** d

> 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
=========
Notes from meeting May
CCI # 9 - a textual change is needed - directives must be in the same module scoping unit.

item number 6 **title** Allocating multiple items

submitted by Henry Zongaro **email** zongaro@vnet.ibm.com **date** 03/30/95 **updated** 7/12/95

group Group D **status** status: resolved **action:**

doc change text **doc change status** no doc change needed

formal reply Decided the compiler should “sort it out”.

original Q

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?

s z

discussion

Meeting discussion - subgroup decision reported by Guy Steele.

CCI #6 If B is aligned with A, allocatable, and the ALLOCATE is called with both A and B in the same statement, did A happen before B?

- ☐- could say both examples wrong
- ☐- left to right
- ☐- let the compiler sort it out <- this was the preferred answer.

item number 20 **title** Declaring array-valued functions

submitted by Henry Zongaro **email** zongaro@vnet.ibm.com **date** 05/04/95 **updated** 7/13/95

group Group E **status** status: waiting for text **action:** needs more text in document

doc change text **doc change status** doc text needed

formal reply Yes, this is a non trivial problen, but No change, put in extra text.

original Q

2) This second question is one that Rich Shapiro was asking towards the end of the October, 94 HPFF meeting. Consider the following program.

```
      program prog
        interface
          extrinsic(hpf_local) function f()
            integer f(10)
!hpf$       processors p(number_of_processors())
!hpf$       distribute f(block) onto p
          end function f
        end interface

        print *, f()
      end program prog

      extrinsic(hpf_local) function f()
        integer f(?)
      end function f
```

How should f be declared in the local subroutine? If this program runs on a number of processors which divides 10, the answer is simple. But what if the program is run on four processors? Three processors have three elements each, but the fourth has one element. There are ways in which you could specify this. However, there's no way for the user to know *which* processor is the one which will have one element of the result variable. Dummy array arguments of HPF_LOCAL procedures don't have this problem, because they must be assumed shape. Perhaps function results have to be distributed in such a way that the same number of elements of the result will be mapped onto all processors.

A related problem: how can an array valued function be specified in an HPF_LOCAL module?

```
      extrinsic(hpf_local) module mod
      contains
        function f()
          integer f(?)
!hpf$     processors p(number_of_processors())
!hpf$     distribute f(block) onto p
        end function f
      end module mod
```

The declaration of f here has to do double duty. It has to provide the shape of the result of f for any reference to the function, and it has describe the result variable mapped onto a single particular processor. Can array valued functions be permitted to appear in HPF_LOCAL modules?

Thanks,

Henry

s d

discussion

subgroup report: - only thing group e could com up with was that:

- a "local" function can only return a scalar
- the corresponding extrinsic function return value must be either a scalar or a rank one array of size number_of_processors()
- Anything more general would not be cci ... and what to do is not obvious.

====

CCI #20 - this poses a non-trivial problem about how the size of a function result is known and / or declared. Fortran 90 doesn't have any way of saying that the result is "assumed shape/size". After extended discussion, the subgroup decided to propose that a local function can only return a scalar, and that the corresponding extrinsic function return value must be either a scalar or an array that is made up from the scalars returned by the locals. In full group discussion, Guy pointed out that it is the DISTRIBUTE directive that is the problem - it lies. There was a lot of discussion, with a basic trend that there should be no change, but rather explanatory text in the document. To bring the issue to close there was a vote on the proposal for no change, but with extra text - (choosing whatever reason among several that the individual liked best for no change)/ This passed 14 - 1 - 6.

cci list: 05/03/95

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

submitted by Henry Zongaro **email** zongaro@vnet.ibm.com **date** 03/30/95 **updated** 7/13/95

group Group E **status** status: waiting for text **action:** needs document text

doc change text **doc change status** doc text needed

formal reply

original Q

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.

s d

discussion

May meeting minutes:
X3J3 already addressed and resolved recently. We will just copy their text.

Yes, it is wrong, the fix is that the example should include DIM=1

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

submitted by Henry Zongaro **email** zongaro@VNET.IBM.COM **date** 02/16/95 **updated** 7/13/95

group Group E **status** status: resolved **action:**

doc change text **doc change status** no doc change needed

formal reply No not two directives.

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 s z

May meeting discussion:
resolved: no not two hpf directives.
part one: the spec explicitly says no, p19, lines 25-43

part 2 - it is “legal” to have comment text that looks like an HPF directive following another statement or directive, but it does not represent another directive.

item number 16 **title** document update completion

submitted by Henry Zongaro **email** **date** 02/26/95 **updated** 7/12/95

group Group not assigned **status** status: postponed **action:** David and Henry to make plan ...

doc change text **doc change status**

formal reply

original Q **discussion** **s a**

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