****

J3/02-298r1
Date: October 2002
To: J3
From: Aleksandar Donev
Subject: Multiple Nonzero-Rank Part References for Structure Components
Reference: J3-007R3, "Multiple Nonzero-Rank Part References for Structure Components in Fortran 95/2002", to appear in the Fortran Forum

****

Summary

I propose to delete the part of C613 (103:9) in 6.1.2 that prohibits multiple nonzero rank part-refs:
"In a data-ref, there shall be no more then one part-ref with nonzero rank." There is no justification for this constraint, and removing it would unleash a most useful capability which Fortran is uniquely capable of with its possibly non-contiguous arrays.

The constraint that "A part-name to the right of a part-ref with nonzero rank shall not have the ALLOCATABLE or POINTER attribute" should remain as this prohibits skewed arrays, which cannot be implemented with regular array descriptors. However, structure components with multiple nonzero rank part-refs *can* be implemented with regular array descriptors.

In fact, I have implemented extensions for the three compilers I use to be able to use such structure components in only a hundred lines of Fortran+C code or so. So I know for sure that this won't pose an implementation challenge, while I also know that it is a most useful functionality in scientific codes. It is also pretty trivial to incorporate the change into the standard.

J3 response

Your idea raises interesting possibilities that require further careful consideration. We will suggest that it be added to the list of features for consideration for the next revision of the Fortran standard.

Motivation

Take the simple example:

```
TYPE Point3D
    ! A point in 3D
    REAL :: coordinates(3), data(2)
END TYPE Point3D

type(point3D), dimension(10) :: points
   ! A collection of points
```

In Fortran,
```
    points[1:2]%coordinates[1]
```
produces a strided rank-1 array section (I will use this term more liberally then the actual standard) which contains the x coordinates of the first two points. This can, for example, be used as a target of a rank-1 array pointer.

However, the reference
```
    points[1:2]%coordinates[:]
```
is not allowed. In a user's mind, this would reference the xyz coordinates of the first two points, and can be thought of as an "array of arrays". But in fact, it can just as well be thought of as a rank-2 array of shape (/3,2/).

This is more then just a convenient convention. In fact, the memory layout of the collection of real numbers (coordinates) referenced by this array of array can be described by a regular strided array section, so that in fact it is almost trivial for any existing F95 compiler to implement the following nonstandard assignment of a rank-2 array pointer to this "array of arrays":

```
REAL, DIMENSION(:,:) :: selected_coordinates
selected_coordinates=>points[1:2]%coordinates[:]
```

Yet no compiler known to the author implements such an extension. It should be obvious to the reader that this kind of functionality would indeed be useful. For example, finding the centroid of the selected points would be performed with,

```
WRITE(*,*) "The centroid is", SUM(selected_points, DIM=2)
```

which requires no loops. Even more useful would be the ability to pass the coordinates of the selected points to a procedure (note that this procedure need not know that the coordinates came from an array of derived type point3D) as an actual argument associated with an assumed-shape array dummy argument.

Solution

Delete "In a data-ref, there shall be no more then one part-ref with nonzero rank" in C613. Then, modify 103:12-13 and add constraint (see C549 at 76:14):

The rank of a data-ref is the sum of the ranks of the part-refs with nonzero rank, if any; otherwise, the rank is zero.
...
Cxxx: The maximum rank of a data-ref is 7.

Also modify Note 6.6, 107:11-13 and 106:14-107:1 accordingly. In particular, modify 106:14+ to say something like:

The rank and shape of a nonzero rank part-ref are determined as follows. If the part-ref has no section-subscript-list, the rank and shape are those of part-name. Otherwise, the rank is the number of subscript triplets and vector subscripts in section-subscript-list, and the shape is the rank-1 array whose i-th element is the number of integer values in the sequence indicated bu the i-th subscript triplet or vector subscript. If any of these sequences is empty, the corresponding element in the shape is zero.

In an array-section, the rank of the array is the sum of the ranks of the nonzero rank part-refs. The shape of the array is the rank-1 array obtained by concatenating the shapes of the nonzero rank part-refs, in backward order, i.e., starting from the last one. If the shape has an element with the value of zero, the array section has size zero.

Problems and Alternatives

Since Fortran only guarantees that arrays of rank up to 7 will be supported by a conforming processor, the validity of having a total rank of more then 7, as in the reference,
level1(:,:,:)%level2(:,:,:)%level3(:,:)
will need to be either prohibited or left "processor-dependent". I believe this is a minor issue.

Another problem is that the Fortran order of specifying components, structure%component, as opposed to the alternative component%structure, is the opposite of the order of concatenation of the shapes of the non-zero rank references. For example, the reference:
level1(1:4,1:5,1:6)%level2(1:2,1:3)%level3(1:1)

represents an array section of shape (/1,2,3,4,5,6/), and not (/4,5,6,2,3,1/) as might be thought at first. Again, I believe this to be a mere inconvenience and not an sufficient reason to deny very useful functionality to Fortran programmers.

Edits

Will be written in a revision after some feedback is received.

Extended Example

Here is an illustration of the expressivity and power of the proposed feature:

```
! A type hierarchy of meterological data:
TYPE :: Hourly_Record
   REAL (KIND=r_wp) :: temperature (3) = 0.0
      ! Three temperature readings (water, air, soil)
   LOGICAL (KIND=l_byte) :: synny = .TRUE.
END TYPE
TYPE :: Daily_Record
   TYPE (Hourly_Record), DIMENSION (24) :: hourly_records
   INTEGER (KIND=i_sp) :: sunrise = 7, sunset = 18
END TYPE
TYPE :: Weekly_Record
   TYPE (Daily_Record), DIMENSION (7) :: daily_records
   REAL (KIND=r_sp) :: forecast_success (5)
END TYPE

! Weather data over a grid of observation points:
INTEGER, PARAMETER :: n_x = 100, n_y = 50
TYPE (Weekly_Record), DIMENSION (n_x, n_y), TARGET :: weekly_records

... ! Assign values to the weather data, do calculations, etc...

! Select the second temperature reading on Mondays and Wednesdays
! at 9:00 and 15:00 hours at grid point (3,1):
WRITE (*,*) "The selected temperatures are:", &
   weekly_records(3,1)%daily_records(1:3:2)%hourly_records(9:15:6)%temperature(2)

! EOF
```