Further coarray features

John Reid, ISO Fortran Convener,

JKR Associates and

Rutherford Appleton Laboratory

Items removed in 2008

In February 2008, it was decided to move the following features into a Technical Specification (a mini Standard) on *Additional Parallel Features in Fortran*

- Teams and features that require teams.
- The collective intrinsic subroutines.
- The **`notify`** and **`query`** statements.
- File connected on more than one image, unless preconnected to the unit specified by **`output_unit`** or **`error_unit`**.

The Technical Specification (TS)

In 2011, it decided that the choice of features should be reviewed while not altering the overall size of the addition.

This choice was made during the 2012 WG5 meeting and modified during the 2013 WG5 meeting.

I will describe the set of additional features now proposed. As voting takes place, there may be more changes.

Teams

Needed for independent computations on subsets of images.

Code that has been written and tested on whole machine should run on a team.

- Therefore, image indices need to be relative to team.
- Collective activities and syncs, need to be relative to team.

team_type and **form team**

The intrinsic module **iso_fortran_env** contains a derived type **team_type**. A scalar object of this type identifies a team of images.

The same **form team** statement must be executed on all images of a team to form subteams

```
form team(id,new_team)
```

The images with the same value of **id** form the new team.

All images of the current team synchronize.

change team construct

```
change team (team)
   : ! Block executed as a team
   if(team_id()==1) then ! New intrinsic
      : ! Code for team 1
   else
      :
end team
```

The new teams synchronize at the **change team** and **end team** statements.

Changing teams is likely to be costly – avoid doing it frequently.

Example

This code splits images into two teams.

```
use iso_fortran_env
integer :: i,ne
ne = num_images()
type(team_type),save :: team
i = 2
if (this_image()<=ne/2) i = 1
form team(i,team)
change team (team)
   :
end team
```

Accessing sibling team

```
   :
real, save :: a(n)[*]
type(team_type) :: initial, block
me = this_image()
initial = get_team() ! New intrinsic
i = ...
form team(i,block)
change team (block)
   :
   sync team(initial) ! New statement
   a(k) = a(1)[initial::me+1]
end team
```

Collectives

The collective subroutines are reduced in number, but a general reduction is added. They are

`co_broadcast, co_max, co_min, co_sum, co_reduce.`

Invoked by the same statement on all images of the team and involve synchronization within them, but not at start and end.

`co_broadcast` and **`co_max`**

`call co_broadcast (source, source_image)`

Copy **`source`** from **`source_image`** to all images of the current team.

`call co_max (source)`

On all images, replace **`source`** by maximum value of **`source`** on all images of the current team.

`call co_max (source, result_image)`

On **`result_image`**, replace **`source`** by maximum value of **`source`** on all images of the current team.

co_min, **co_sum**, **co_reduce**

co_min and **co_sum** are just like **co_max.**

co_reduce is also just like **co_max** but has extra argument

```
call co_reduce (source,operator)
```
or
```
call co_reduce (source, operator, &
                result_image)
```

operator is a pure function with two arguments of the same type as **source.** Applied just like **max, min, sum.**

Keeping source for collectives

If you want to retain the source, it is easy:

```
result = source
call co_max (result)
```

End of first half of lecture