Further coarray features

John Reid

In 2008, WG5 decided to move some coarray features from Fortran 2008 to a Technical Specification (TS).

In 2011, WG5 decided that the choice of features should be reviewed, while not increasing the overall size. The choice was made in 2012 and revised this year.

We will review the recent changes and describe the full set of additions now proposed.

BCS Fortran Specialist Group
London, 26 September 2013.

2013 Changes

- Teams provide a minimal portable mechanism for continued execution in the presence of failed images
- When executing in a team, an image has access to data of ancestor teams
- At the beginning and end of a `change team` construct, each new team synchronizes
- Waiting for an event to be posted shall involve a local event variable

Collectives (1)

```
call co_broadcast (source, source_image)
```

Copies `source` on coarray `source_image` to all other images

```
call co_max (source)
```

```
call co_min (source)
```

```
call co_sum (source)
```

Replaces `source` elementally by max, min, or sum of values on all images. `source` need not be a coarray.

```
call co_reduce (source, operator)
```

Similar for user's binary operator, which must be a pure elemental function

Collectives (2)

- A collective must be invoked by same statement on all images of the team.
- No sync on entry or return, but syncs inside.
- `co_max`, `co_min`, `co_sum`, and `co_reduce` have optional arguments `result` and `result_image`.

  If `result` is present, the result is put in `result` and `source` is not changed.

  If `result_image` is present, `result` is defined on this image and is undefined on other images.

Teams

Needed for independent computations on subsets of images.

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

Image indices are relative to team.

Collective activities, including syncs, need to be relative to team.

NB Old feature did not do this.

Team variables

- Team variables are scalar values of type `team_type` from the intrinsic module `ISO_Fortran_env`.
- Formed by executing on all images of the current team

```
form subteam(id,team)
```

where *`id`* is an integer with a positive value and *`team`* is a team variable. Images with same value of *`id`* are put in the same subteam. Example:

```
use ISO_Fortran_env
type(team_type) all_images
form subteam(1,all_images)
```

Form subteam statement

- The same `form subteam` statement must be executed on all images of the current team and involves synchronization. Allows the implementation to store data in the team variable for efficient execution as a team.
- Another example:

```
use ISO_Fortran_env
type(team_type) odd_even
integer :: i
i = 2-mod(this_image(),2)
form subteam (i, odd_even)
```

Change team construct

The team is changed by the construct

```
change team (team)
    block
end team
```

Within *block*, each image executes as part of the subteam defined by *team*.

The subteams synchronize at the `change team` and `end team` statements.

Ancestor teams

- Nested `change team` constructs lead to the concept of **ancestor** teams and **team depth** from the original team of all images.
- Intrinsic functions:

```
team_depth()

subteam_id(distance) ! integer

this_image(distance)

num_images(distance)
```

Symmetric memory

To preserve symmetric memory, any coarray that is allocated in a change team block and is still allocated on leaving the block is deallocated automatically.

Access outside team

- Can access a coarray of an ancestor:

```
a[ancestor: i,j]
```

- Can synchronize as a member of any team:

```
sync_team(team)
```

Standard input

Standard input only on root image 1.

Failed images

- Once identified as failed, an image remains so.
- All the statements that have a `stat=` specifier can return `stat_failed_image` from the intrinsic module `ISO_Fortran_env` if a failed image is detected.
- The intrinsic function `failed_images()` returns an integer array of image indices in the current team.

Failed images example

Form a new team for continued execution:

```
if (num_images(failed=.true.) > 0 ) then
   form subteam(1, recover, stat=st)
            ! Returns stat_failed_image
   change team (recover)
       : ! Execute as a subteam
   end team
end if
```

Second failed images example

Perform calculation redundantly with checks for both halves having failed

```
change team (half)
        ! Perform whole calculation
            :
        sync team(partner, stat = st)
        if (st==stat_failed_image) then
        ! Partner has failed. Check my team.
           sync memory(stat=st)
           if (st==stat_failed_image) error stop
        end if
          :
    end team
```

Events

One-sided ordering of execution segments sometimes needed.

For example, suppose image I executes segments I1, I2,... and image J executes segments J1, J2,... there might be a need for I1 to precede J2 without the need for J1 to precede I2.

`NOTIFY/QUERY` did this but relied on counts of number of executions.

Tagged events give much clearer associations. Important, for a library routine, for example.

Events (2)

Events are held in scalar coarrays of type `event_type` from the intrinsic module `ISO_Fortran_env`.

```
use ISO_Fortran_env
type(event_type) :: event[*]
    :
event post(event[i])
    :
event wait(event)
    :
call event_query(event,count)
```

More atomics

Integer add

Bitwise and

Bitwise or

Bitwise xor

Compare and swap cas

Parallel Input/Output

A feature for parallel I/O was removed in 2008.

It was decided not to restore this or design a new feature.

This makes a worthwhile reduction in the size of the TS, which should mean that its construction is easier, quicker, and less controversial than it would have been.