Parallel programming in Fortran with Coarrays

John Reid, ISO Fortran Convener,
JKR Associates and
Rutherford Appleton Laboratory

The ISO Fortran Committee has decided to include co-arrays in Fortran 2008.

Aim of this talk: introduce co-arrays and explain why we believe that they will lead to easier development of parallel programs, faster execution times, and better maintainability.

RAPS Workshop,
ECMWF, Reading,
5 November 2008.

Design objectives

Coarrays are the brain-child of Bob Numrich (Minnesota Supercomputing Institute, formerly Cray).

The original design objectives were for

- A simple extension to Fortran
- Small demands on the implementors
- Retain optimization between synchronizations
- Make remote references apparent
- Provide scope for optimization of communication

A subset has been implemented by Cray for some ten years.

Coarrays have recently been added to the g95 compiler.

Summary of coarray model

- SPMD – Single Program, Multiple Data
- Replicated to a number of **images** (probably as executables)
- Number of images fixed during execution
- Each image has its own set of variables
- Coarrays are like ordinary variables but have second set of subscripts in [ ] for access between images
- Images mostly execute asynchronously
- Synchronization: `sync all`, `sync images`, `sync memory`, `allocate`, `deallocate`, `critical` construct
- Intrinsics: `this_image`, `num_images`, `co_lbound`, `co_ubound`, `image_index`.

Full summary: Reid (2008)

Examples of coarray syntax

```
real :: r[*], s[0:*] ! Scalar co-arrays
real :: x(n)[*]      ! Array co-array
type(u) :: u2(m,n)[np,*]
! Co-arrays always have assumed
! co-size (equal to number of images)

real :: t               ! Local
integer p, q, index(n)  ! variables
     :
t = s[p]
x(:) = x(:)[p]
! Reference without [] is to local part
x(:)[p] = x(:)
u2(i,j)%b(:) = u2(i,j)[p,q]%b(:)
```

Implementation model

Usually, each image resides on one processor.

However, several images may share a processor (e.g. for debugging) and one image may execute on a cluster (e.g. with OpenMP).

A coarray has the same set of bounds on all images, so the compiler may arrange that it occupies the same set of addresses within each image.

On a shared-memory machine, a coarray may be implemented as a single large array.

On any machine, a coarray may be implemented so that each image can calculate the memory address of an element on another image.

Synchronization

With a few exceptions, the images execute asynchronously. If syncs are needed, the user supplies them explicitly.

Barrier on all images

```
        sync all
```

Wait for others

        `sync images(`*image-set*`)`

For hand coding, e.g. spin loops

```
        sync memory
```

Critical construct

```
    critical
       p[6] = p[6] + 1
       :
    end critical
```

Limits execution to one image at a time.

Non-coarray dummy arguments

A coarray may be associated as an actual argument with a non-coarray dummy argument (nothing special about this).

A coindexed object (with square brackets) may be associated as an actual argument with a non-corray dummy argument. Copy-in copy-out is to be expected.

These properties are very important for using existing code.

Dynamic coarrays

Only dynamic form: the allocatable coarray.

All images synchronize at an `allocate` or `deallocate` statement so that they can all perform their allocations and deallocations in the same order. The bounds must not vary between images.

Automatic arrays or array-valued functions would require automatic synchronization, which would be awkward.

An allocatable coarray may be a component of a structure provided the structure and all its ancestors are scalars that are neither pointers nor coarrays.

Coarray dummy arguments

A dummy argument may be a coarray. It may be of explicit shape, assumed size, assumed shape, or allocatable:

```
subroutine subr(n,w,x,y,z)
  integer :: n
  real :: w(n)[n,*]  ! Explicit shape
  real :: x(n,*)[*]  ! Assumed size
  real :: y(:,:)[*]  ! Assumed shape
  real, allocatable :: z(:)[:,:]
```

Where the bounds or cobounds are declared, there is no requirement for consistency between images. The local values are used to interpret a remote reference. Different images may be working independently.

There are rules to ensure that copy-in copy-out of a coarray is never needed.

Co-Arrays and SAVE

Unless allocatable or a dummy argument, a co-array must be given the SAVE attribute.

This is to avoid the need for synchronization when co-arrays go out of scope on return from a procedure.

Structure components

A coarray may be of a derived type with allocatable or pointer components.

Pointers must have targets in their own image:

```
q => z[i]%p        ! Not allowed
allocate(z[i]%p) ! Not allowed
```

Provides a simple but powerful mechanism for cases where the size varies from image to image, avoiding loss of optimization.

Optimization

Most of the time, the compiler can optimize as if the image is on its own, using its temporary storage such as cache, registers, etc.

There is no coherency requirement except on synchronization.

It also has scope to optimize communication.

Main changes since the 1998 draft

- cosubscripts limited to scalars
- The critical section replaces the intrinsic subroutines, `start_critical` and `end_critical`
- Rank plus corank limit made 15.
- coarrays of a type with allocatable components
- Allocatable coarray dummy arguments
- No parallel i/o features

Recent changes

A substantial reduction was proposed by the US at the February meeting and accepted.

It is to separate parallel programming features into a ‘core’ set that remain in Fortran 2008 while the following features are moved into a separate Technical Report on ‘Enhanced Parallel Computing Facilities’:

1. The collective intrinsic subroutines.

2. Teams and features that require teams.

3. The `notify` and `query` statements.

4. File connected on more than one image, unless preconnected to the unit specified by `output_unit` or `error_unit`.

It was also decided to remove hyphens from the words ‘co-array’, ‘co-rank’, etc., (cf ‘cosine’ and ‘cotangent’).

Comparison with MPI (i)

MPI is the de-facto standard but is awkward to program. Here is an example due to Jef Dawson of AHPCRC-NCSI.

With coarrays, to send the first `m` elements of an array from one image to another:

```
real ::  a(n)[*]
me=this_image()
if ( me.eq.2 ) a(1:m)=a(1:m)[1]
sync_all
```

and with MPI:

```
real :: a(n)
call mpi_comm_rank(mpi_comm_world,  &
                  myrank, errcode)
if (myrank.eq.0) call mpi_send      &
      (a,m,mpi_float,1,tag1, &
       mpi_comm_world,errcode)
if (myrank.eq.1) call mpi_recv      &
      (a,m,mpi_float,0,tag1, &
       mpi_comm_world,status,errcode)
```

Comparison with MPI (ii)

Experience on the Cray vector computers with the Cray compiler suggests that there is a performance advantage as the number of processes increases.

For example, Dawson (2004) reports speed-up of 60 on 64 processors of the Cray X1 for a stencil update code, compared with 35 for MPI.

Comparison with MPI (iii)

A colleague of mine (Ashby, 2008) recently converted most of a large code, SBLI, a finite-difference formulation of Direct Numerical Simulation (DNS) of turbulance, from MPI to coarrays using a small Cray X1E (64 processors).

Since MPI and coarrays can be mixed, he was able to do this gradually, and he left the solution writing and restart facilites in MPI.

Most of the time was taken in halo exchanges and the code parallelizes well with this number of processors. He found that the speeds were very similar.

The code clarity (and maintainability) was much improved. The code for halo exchanges, excluding comments, was reduced from 176 lines to 105 and the code to broadcast global parameters from 230 to 117.

Advantages of coarrays

- Easy to write code – the compiler looks after the communication
- References to local data are obvious as such.
- Easy to maintain code – more concise than MPI and easy to see what is happening
- Integrated with Fortran – type checking, type conversion on assignment, ...
- The compiler can optimize communication
- Local optimizations still available
- Does not make severe demands on the compiler, e.g. for coherency.

References

Ashby, J.V. and Reid, J.K (2008). Migrating a scientific application from MPI to coarrays. CUG 2008 Proceedings. RAL-TR-2008-015, see `http://www.numerical.rl.ac.uk/reports/reports.shtml`

Dawson, Jef (2004). *Coarray Fortran for productivity and performance*. In Army HPC Research Center Bulletin, 14, 4.

Reid, John (2008). *Coarrays in the next Fortran Standard*. ISO/IEC/JTC1/SC22/ WG5-N1747, see `ftp://ftp.nag.co.uk/sc22wg5/N1701-N1750`