Fortran 2008: what's in it for high-performance computing

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

Fortran 2008 has been completed and is about to be published.

The biggest change is the addition of coarrays, summarized at RAPS 2008.

The aim of this talk is to remind you about coarrays and explain other additions that have been designed for enhanced performance.

RAPS Workshop,
ECMWF, Reading,
3 November 2010.

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; have same bounds on all images
- Images mostly execute asynchronously
- Programmer responsible for synchronization: `sync all`, `sync images`, `critical`, `lock`
- Intrinsics: `this_image`, `num_images`, `image_index`

Full summary: WG5 N1824
(Google WG5 N1824)

Examples of coarray syntax

```
real :: r[*], s[0:*] ! Scalar coarrays
real,save :: x(n)[*] ! Array coarray
type(u),save :: u2(m,n)[np,*]
! Coarrays always have assumed
! cosize (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 object
x(:)[p] = x(:)
u2(i,j)%b(:) = u2(i,j)[p,q]%b(:)
```

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*`)`

Limit execution to one image at a time

```
     critical
        :
     end critical
```

Limit execution in a more flexible way

```
     lock(lock_var[6])
        p[6] = p[6] + 1
     unlock(lock_var[6])
```

These are known as **image control statements**.

Execution segments

On an image, the sequence of statements executed before the first image control statement or between two of them is known as a **segment**.

The image control statements produce a partial ordering of the segments: for any two segments, one may precede the other or they may be unordered.

Important rule: if a variable is defined in a segment, it must not be referenced, defined, or become undefined in a another segment unless the segments are ordered.

Dynamic coarrays

Only dynamic form: the allocatable coarray.

```
real, allocatable :: a(:)[:], s[:,:]
    :
allocate ( a(n)[*], s[-1:p,0:*] )
```

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, cobounds, and length parameters must not vary between images.

A coarray is not allowed to be a pointer.

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.

A dummy argument may be a coarray. The actual argument must also be a coarray and there are rules to ensure that copy-in copy-out is never needed.

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.

Input/output

Default input (`*`) is available on image 1 only.

Default output (`*`) and error output are available on every image. The files are separate, but their records will be merged into a single stream or one for the output files and one for the error files.

The `open` statement connects a file to a unit on the executing image only.

Whether a file on one image is the same as a file with the same name on another image is processor dependent.

Planned extensions

The following features were part of the proposal but have moved into a planned 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 default output or default error.

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.

A subset has been implemented by Cray for some ten years. Coarrays have been added to the g95 compiler, are being added to gfort, and for Intel ‘are at the top of our development list’.

Enhanced module facilities (TR)

If a huge module is split into several modules:

- Internal parts exposed
- Any change leads to compilation cascade

Solution:

- Submodules contain definitions of procedures whose interfaces are in the module itself
- Users have access these procedures, but no recompilation of user code needed if submodule changes
- Submodules have full access by host association
- Submodules can be compiled independently

Enhanced performance

- `contiguous` attribute

  Arrays need not be contiguous in Fortran, e.g. the section `a(1:n:2)`. Can lead to performance loss for pointer and assumed-shape arrays. Users can now promise not to let this happen.

- `do concurrent`
  Iterations of the loop are independent. Allows low-level optimizations such as vectorization.

References

Reid, John (2010). *Coarrays in the next Fortran Standard.* ISO/IEC/JTC1/SC22/ WG5 N1824, see `ftp://ftp.nag.co.uk/sc22wg5/N1801-N1850`

Reid, John (2010). *The new features of Fortran 2008.* ISO/IEC/JTC1/SC22/ WG5 N1828, see `ftp://ftp.nag.co.uk/sc22wg5/N1801-N1850`

WG5(2010). *FDIS revision of the Fortran Standard.* ISO/IEC/JTC1/SC22/ WG5 N1830, see `ftp://ftp.nag.co.uk/sc22wg5/N1801-N1850`