Preparation

- **Log in to mirage**

  `$ ssh -l login mirage[0-2].ucar.edu`

  Use CryptoCard or Yubikey

- **Copy the example source files**

  `$ cp –r /glade/home/dnagle/Fortran-III .`

Modern Fortran III for Computational Scientists

Consulting Services Group
Dan Nagle (Presenter)
May 24, 2012

Outline

- Documents
- Coarrays
- Trial Problems
- Implementations

Documents

- WG5 paper N1824.pdf (www.nag.co.uk/sc22wg5)
- Rice University (caf.rice.edu)
- FDIS J3/10-007r1.pdf (www.j3-fortran.org)
- <u>Modern Fortran Explained</u> by Metcalf, Reid, Cohen

PGAS

- Partitioned Global Address Space
- Examples include:
  - UPC (upc.gwu.edu)
  - Titanium (titanium.cs.berkeley.edu)
- Original Paper at ftp://ftp.numerical.rl.ac.uk/pub/reports/nrRAL98060.ps.gz

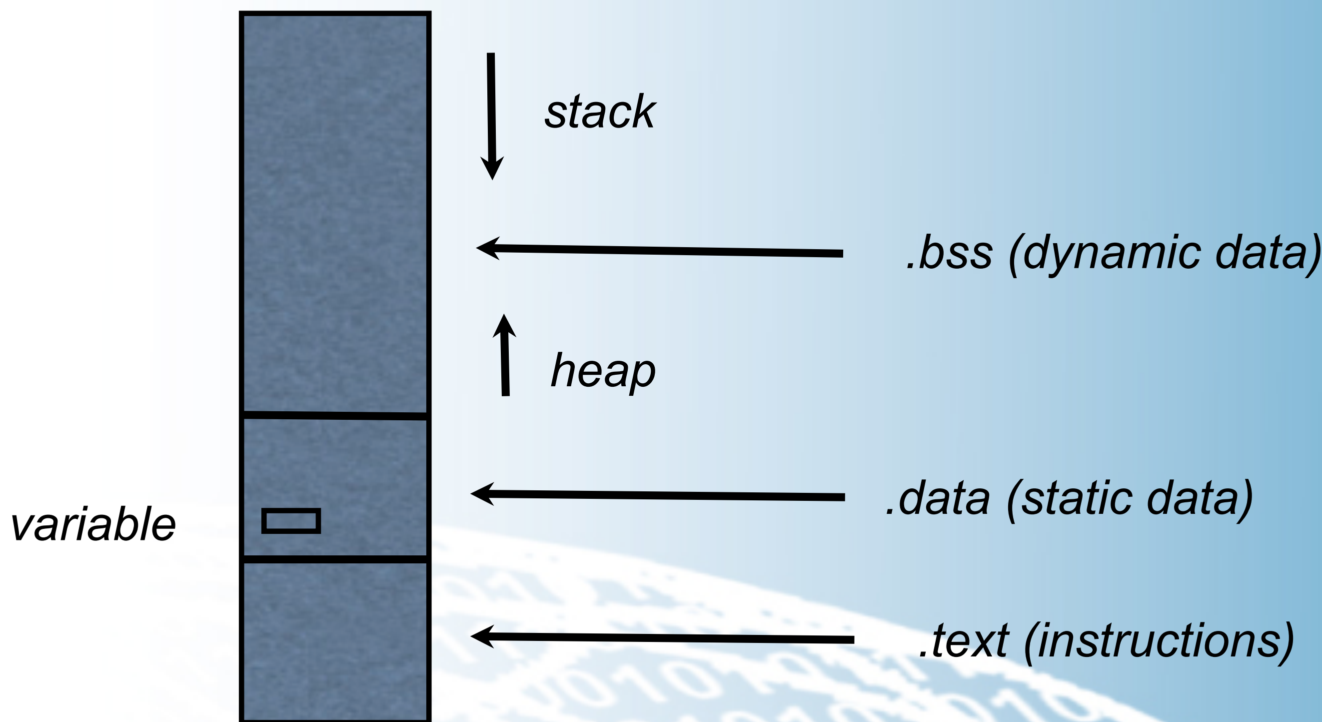
Brief History

- Agreed at Delft 2005
- Discussed at Fairfax 2006
- Again at London 2007
- Las Vegas Compromise 2008 (core & more)
- Tokyo Further Compromise 2008
- Finally Agreed Las Vegas 2010

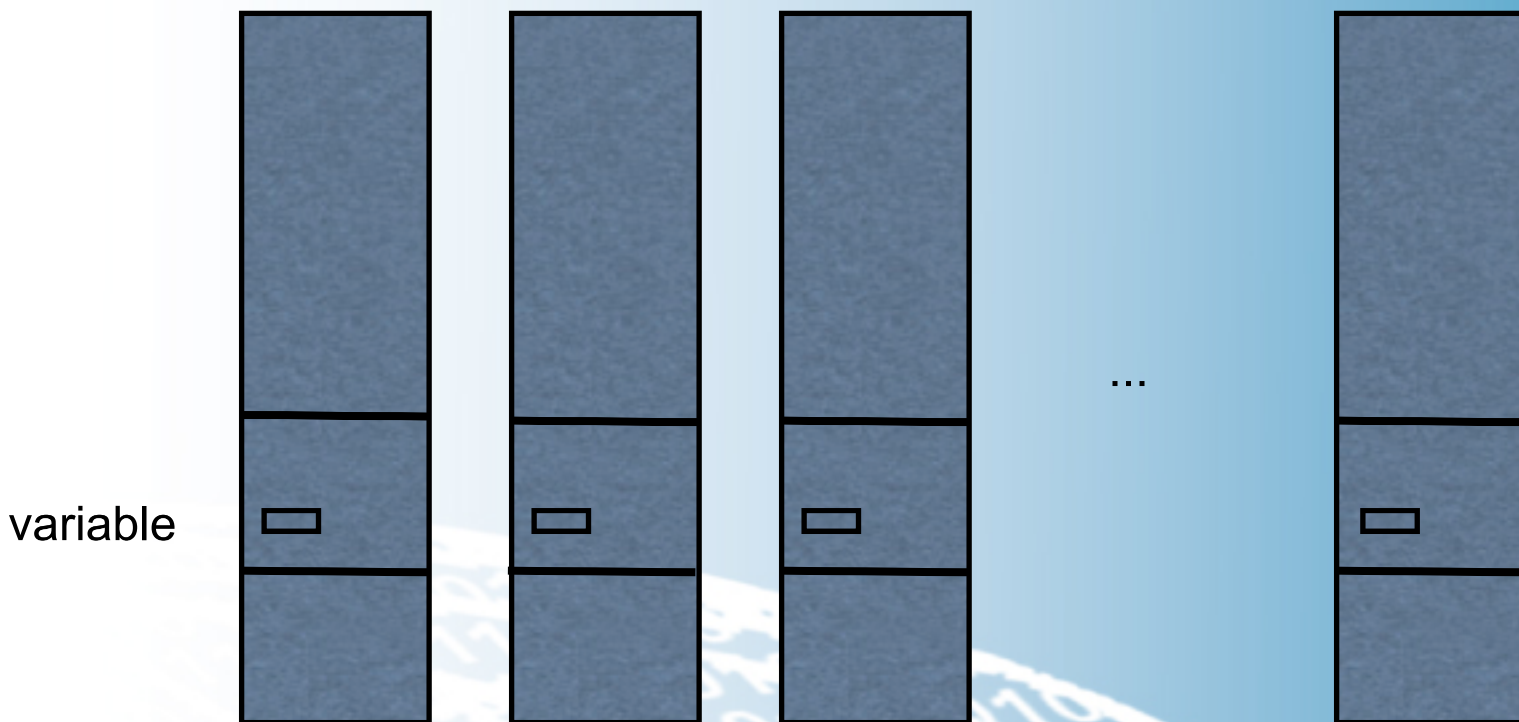
Coarray Concepts

- A program is treated as if it were replicated at the start of execution, each replication is called an image.
- Each image executes asynchronously.
- A coarray is indicated by trailing [ ].
- A data object without trailing [ ] is local.
- Explicit synchronization statements are used to maintain program correctness.

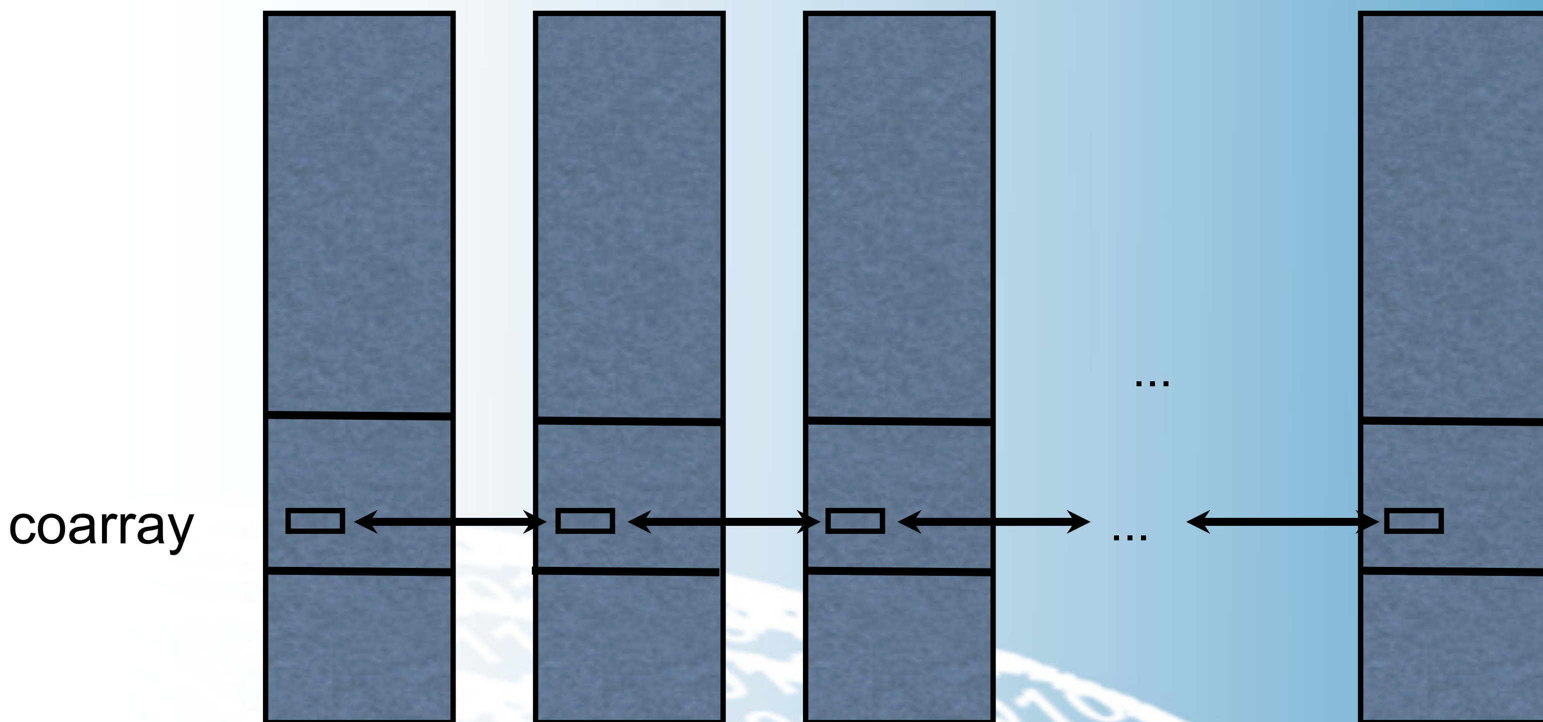
Memory Basics (Single Image)



Memory Basics (Multiple Images)



Memory Basics Coarrays



Computational & Information Systems Laboratory
CISL
NCAR

Memory Organization

- A static variable is stored at a fixed memory location.
- The fixed location is independent of the number of replications of the program.
- The coarray attribute connects the storage locations on each replication.
- The remote image may use the same address as the local image to access a data item.

Using Dynamic Coarrays

- A coarray may have the same address on all images.
- A coarray program must synchronize when a coarray is allocated or deallocated, or when an automatic local variable in a referenced subprogram is a coarray.

Image Indexes

- An image has an image index, that is a number between one and the number of images, inclusive.
- The images may be addressed as a rectangular array.
- The mapping from coindexes to image index is the same as an ordinary array.

Coarray Declarations

```
! a scalar coarray
real :: a[ *]
! an array coarray
real, dimension( n) :: ar[ *]
! a scalar coarray
real :: ca[ 10, 10, *]
! an array coarray
real :: caa( m, n)[ 10, 10, *]
```

Coarray Declarations

```
! a coarray with corank > 1
real :: a[ 10, *]
! sum of rank + corank <= 15
real, dimension( n) :: ar[ *]
! another scalar coarray
real :: ca[ -10: 10, *]

! the last coextent is always *
```

Simple Coarray Usage

```
! get a value from another image
a = b[ 4]
! send a value to another image
b[ n] = a
! get an array value
c( 1: n) = ca( 1: n)[ i]
! put an array value
caa( 1: n)[ j] = aa( 1: n)
```

Coarrays as Actual Arguments

```
! a coarray may be used
! as an ordinary actual argument
! may have intent in, out, in out

call my_sub( b[ 4])
x = f( y[ j])
```

Coarrays as Dummy Arguments

```
! a coarray dummy argument
! needs an explicit interface
! intent may be in, out, in out

subroutine my_sub( b)
real, intent( in out) :: b[ *]

pure function func( y)
real, intent( in) :: y( :) [ *]
```

Segments

- A <u>segment</u> is a piece of code between synchronization points.
- A compiler is free to apply all its optimizations within a segment.
- Segments are <u>ordered</u> by synchronization statements or dynamic memory actions.
- Segments may be <u>ordered</u> or <u>unordered</u>.

Synchronization

```
! synchronize all the images
sync all

! synchronize with a set of images
sync images( this_image() + 1)
sync images( list_images( i: j))
sync images( *)

! wait for memory quiet
sync memory
```

Coarray Local Variables

```
! function has coarray local variables
function f( x)
…
  real :: local_a[ *]
…
end function f

! reference f
! this causes two synchronizations:
! one at the call, one at the return
y = f( x1)
```

Allocatable Coarrays

```
! declaring allocatable coarrays
real, allocatable, dimension( :) :: ac[ :]

! allocate as usual
! this causes a synchronization
allocate( ac( n)[ *], stat= … )

! deallocate as usual
! this causes a synchronization
deallocate( ac, stat= … )
```

Derived Types with Coarrays

```
! a derived type with a coarray component
type :: co_array_t
   real, dimension( :) :: dtc[ *]
end type co_array_t

! a variable of type co_array_t
! this must be a scalar
type( co_array_t) :: my_dt_coarray

! a coarray of derived type
! my_type has no coarray components
type( my_type), dimension( :) :: dtc[ *]
```

Inheriting Derived Types with Coarrays

```
! a derived type with a coarray component
type :: parent_t
   real, dimension( n) :: dtc[ *]
end type parent_t

! inherit from parent_t
! if child_t has coarray components,
! so must the parent_t
type, extends( parent_t) :: child_t
   real, dimension( n) :: adtc[ *]
end type child_t
```

Derived Types with Coarrays

```
! a derived type with a coarray component
type :: has_coarray_t
   real, dimension( :) :: dtc[ *]
end type has_coarray_t

! a variable of type co_array_t
! this must be a scalar
type( has_coarray_t) :: my_dt_coarray

! the coarrayness exists at only one level
```

Derived Types with Coarrays

```
! derived type without coarray components
type :: no_coarray_t
   real :: vx, vy, vz
end type no_coarray_t

! a coarray of derived type
! no_coarray_t has no coarray components
type( no_coarray_t) :: dtc( n)[ *]

! the coarrayness exists at only one level
```

Coarrays & Pointers

```
! a coarray cannot be a pointer
! so make a derived type
type :: cptr
 real, dimension( :), pointer :: p
end type cptr
type( cptr) :: a, b[ *]
! all components assigned
a = b
! a% p is undefined
a = b[ j]
```

Coarray Intrinsics

```
! number of images
nim = num_images()
! my image number
me = this_image()
! cobounds
l = lcobound( ca, dim= 1)
u = ucobound( ca, dim= 2)
! index of cosubscripts
i = image_index( ca, subs)
```

Locks

```
! declare a lock coarray
type( lock_type) :: work_lock[ *]

! lock a critical resource
lock( work_lock)
…
unlock( work_lock)

! lock a neighbor's lock
lock( work_lock[ me + 1])
…
unlock( work_lock[ me + 1])
```

Critical Section

```
! one image at a time
critical

! update the critical resource
   i_crit = i_crit + 1

! end one image at a time
end critical
```

Atomic Intrinsics

```
integer( atomic_int_kind) :: iflag[ *]
logical( atomic_logical_kind) :: al[ *]

!  define iflag on image i to be 42
call atomic_define( iflag[ i], 42)

!  get value of al on image j
call atomic_ref( local_flag, al[ j])
```

Simple Example

```
program small_hello_world
  integer :: me
continue

 me = this_image()
 write( unit= *, fmt= '( a, i0, a)') &
   'Image ', me, ' says "Hello, world!"'

stop 'normal completion'
end program small_hello_world
```

Monte Carlo Toy

- Using four images, compute an approximation of pi as four times the ratio of points within a circle to points within the containing square.
- Image one should read the number of trials per image and print the sum of all the image's estimates.
- Remember to seed the random number generator differently on each image.

MxM

- Image one reads two NxN matrices.
- Decide upon a storage scheme among the four images, and distribute the matrix.
- Decide upon a computation scheme (that is, a blocking scheme), and multiply the two matrices.
- Image one writes the product matrix.

Digits Home

- Image one reads 16 four-digit numbers, where each digit is in the range [1-4].
- Distribute one fourth of the numbers to each of four images.
- Shuffle the numbers to where the ones digit is equal to the image number where it is stored.
- Repeat for the tens, hundreds, thousands digits.
- Print from each image at each step.

3d-fft

- Image one reads a rank-three array.
- Distribute to each of four images.
- Compute a three dimensional FFT in place.
- Image one prints the result.

Reduction

- Each image computes a value from its image number (for example, the image number squared).
- Sum the computed numbers in a logarithmic number (of images) of steps.
- Without a broadcast, the sum is on each image.
- Verify the correctness of the sum on each image, and print the sum from image one.

Implementations

- Cray
- IBM (PPC-only, Beta-only)
- g95 (but is it maintained?)
- Intel (12.1+)
- gfortran (4.6 single-image only)

Future Coarrays

- John Reid's Summary N1824
- Bill Long's Draft of the TS N1858 = 11-176
  - Pre Garching
- Metcalf, Reid, & Cohen, Modern Fortran Explained
  - http://www.amazon.com/Explained-Numerical-Mathematics-Scientific-Computation/dp/0199601429/ref=sr_1_1?s=books&ie=UTF8&qid=1326999081&sr=1-1

Computational & Information Systems Laboratory

CISL

NCAR

Basic Ideas

- Collective functions
  - Changed between 08-131r1 and 11-256r2
- Teams
  - Original team proposal versus Rice U proposal
- Notify/Query – How Much Like Events ?
- Parallel I/O – Features and Limitations ?
- Global Data Structures – coscalars or pointers ?
- Atomic Operations

Documents with Suggestions

- 08-131r1 (the original list of deferred features)
- 10-166 (Bill Long's 2010 draft of the Further Coarrays TS based on 08-131r1)
- N1835 (John Reid's 2010 summary list)
- N1856 (Rice U CAF group 2011 list)
- N1883 Comments on list (post Garching)
- 11-256r2 (John Reid's 2011 summary list) **

The Size of the Coarray Additions

- Competing desires:
  - to manage the workload on compiler suppliers
  - to provide useful tools to applications programmers as quickly as possible
- What's missing ?
  - Must judge without extensive application programmer experience with f08 coarrays
- "Useful" is application-dependent

Collective Procedures

- Which ones ?
  - Similar to the existing reduction intrinsics ?
  - Similar to MPI reduction procedures ?
- Synchronous or Asynchronous ?
  - If synchronous, what affect on performance ?
  - If asynchronous, how ?
    - Signal completion via event variables ?

The Original Set

- From 08-131r1 (see also 07-007r3)
  - co_all, co_any, co_count
  - co_maxloc, co_maxval
  - co_minloc, co_minval
  - co_product, co_sum
  - co_findloc ?
- Most Had Arg Lists (source, result [, team])
- Synchronous (Image Control Statements)

The Current Favorites

- As per 11-256r2
- co_bcast, co_max, co_min, co_reduce, co_sum
- asynchronous ?
- copy_async ?
- max-and-copy, min-and-copy ?
- apply to non-coarrays ?

Teams

- Teams were originally lists of image indexes
- Teams might be subdivided from a parent team via the scheme proposed by the Rice U CAF group
- What should teams do ? (synchronize and …)
  - label blocks ? (the **with team** construct)
  - procedures ? (intrinsic or external ?)
  - I/O ? (**team=** on control list)

Computational & Information Systems Laboratory

NCAR

Parallel Input/Output

- Sequential Access or Direct Access or Both ?
  - If sequential, control record order ?
  - If direct, control record access ?
    - how to manage races ?
- Linked with Teams how ?
  - **team=** on control list ?

Notify/Query

- As is (identified by image number), or as first class events (identified by event variables) ?
- Synchronous or queued ?
- Applied to a team or global ?
- What about libraries ?
  - Do library writers prefer event variables ?

If First Class Events, Details ?

- Apply to collective intrinsics ?
  - copy_async: Rice U CAF proposal has 3 events: source ready, destination ready, copy ready
- Apply to asynchronous I/O ?
- Apply to any other wait operation ?
- Do events queue or signal ?
  - If they queue, how to distinguish individual events ?

Global Data Structures

- Coscalars (Germany)
- Copointers (Rice U CAF)
  - and cotarget
- Pointer attribute allowed on coarrays (IBM)

Any other business

- Asynchronous copy (Rice U CAF)
- More atomic operations (Cray)
  – compare-and-swap is likely
  – the rest of Cray’s current set ?
    - add, and, or, xor
    - and corresponding fetch-and-*op* versions
- Support for irregular grids ?
  – affinity for teams ?

The Process Needs Input

- In February, set the initial feature list to start the discussion for the International meeting
  - might be viewed as the opening US position
- The June International meeting at IBM Toronto Labs will set the work list
- What are your concerns ?
- Discussion at http://sea.ucar.edu/forums/updates-fortran-2008

Modern Fortran III for Computational Scientists

Thanks for Attending!

May 24, 2012