Coarray: a parallel extension to Fortran

Jim Xia
IBM Toronto Lab
jimxia@ca.ibm.com

SCINET compiler workshop | February 17-18, 2009

Agenda

- Coarray background
- Programming model
- Synchronization
- Comparing coarrays to UPC and MPI
- Q&A

Existing parallel model

- MPI: de facto standard on distributed memory systems
  - Difficult to program
- OpenMP: popular on shared memory
  - Lack of data locality control
  - Not designed for distributed systems

Coarray background

- Proposed by Numrich and Reid [1998]
  - Natural extension of Fortran's array language
  - Originally named F-- (as jokey reference to C++)
- One of the Partitioned Global Address Space languages (PGAS)
  - Other GAS languages: UPC and Titanium
- Benefits
  - One-sided communication
  - User controlled data distribution and locality
  - Suitable for a variety of architectures: distributed, shared or hybrid
- Standardized as a part of Fortran 2008
  - Expected to be published in 2010

Programming model

- Single Program Multiple Data (SPMD)
  - Fixed number of processes (images)
- “Everything is local!” [Numerich]
  - All data is local
  - All computation is local
- Explicit data partition with one-sided communication
  - Remote data movement through codimensions
- Programmer explicitly controls the synchronizations
  - Good or bad?

Coarray syntax

- CODIMENSION attribute

```
double precision, dimension(2,2), CODIMENSION[*] :: x
```

- or simply use **[ ]** syntax

```
double precision :: x(2,2)[*]
```

- a coarray can have a corank higher than 1

```
double precision :: A(100,100)[5,*]
```

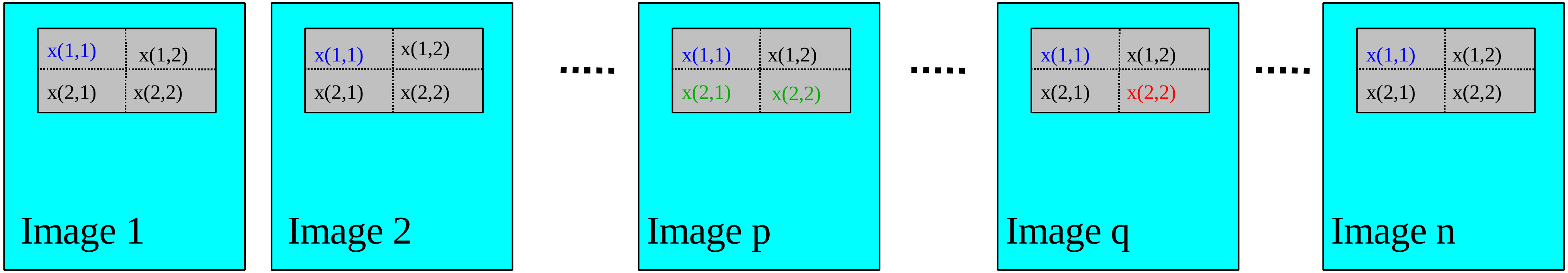
- from ANY single image, one can refer to the array x on image Q using **[ ]**

```
X(:,:)[Q]
e.g. Y(:,:)    = X(:,:)[Q]
     X(2,2)[Q] = Z
```

- Coindexed objects
  - Normally the remote data
- Without **[ ]** the data reference is local to the image

```
X(1,1) = X(2,2)[Q]
!LHS is local data; RHS is a coindexed object, likely a
!remote data
```

Coarray memory model



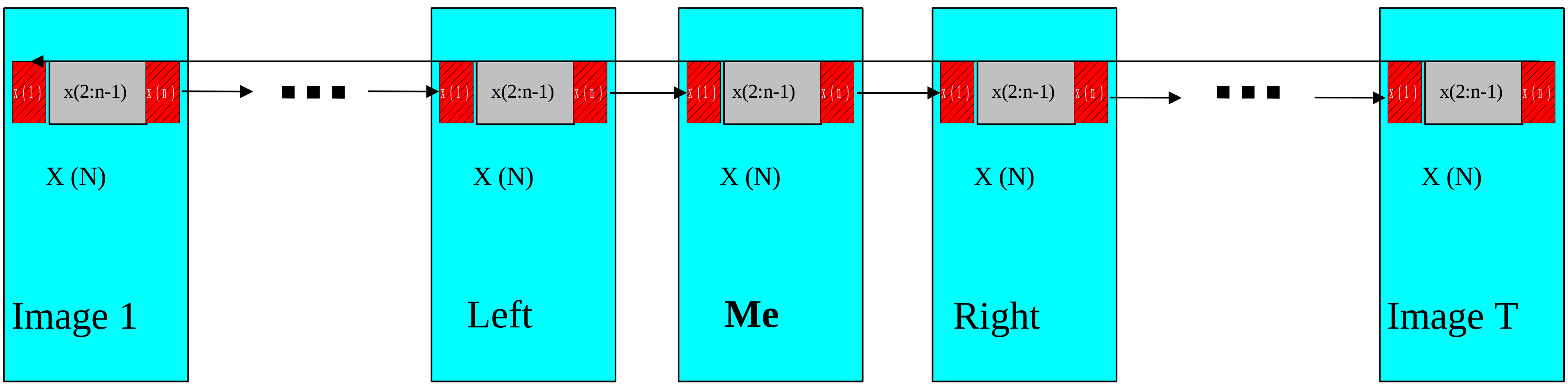
Logical view of coarray X(2,2)[*]

- A fixed number of images during execution
  - Each has a local array of shape (2 x 2)

examples of data access: local data and remote data

```
X(1,1)    =  X(2,2)[q] !assignment occurs on all images
if (this_image() == 1) X(2,2)[q] = SUM(X(2,:)[p])
                      !computation of SUM occurs on image 1
```

Example: circular shift by 1



Real :: X(N)[*]

- image indexing

```
me = this_image()
if (me == 1) then
   left = num_images()
else
   left = me - 1
end if
```

- Execute the shift

```
SYNC ALL
temp = x(n-1)
x(2:n-1) = x(1:n-2)
x(1) = x(n)[left]
SYNC ALL
x(n) = temp
```

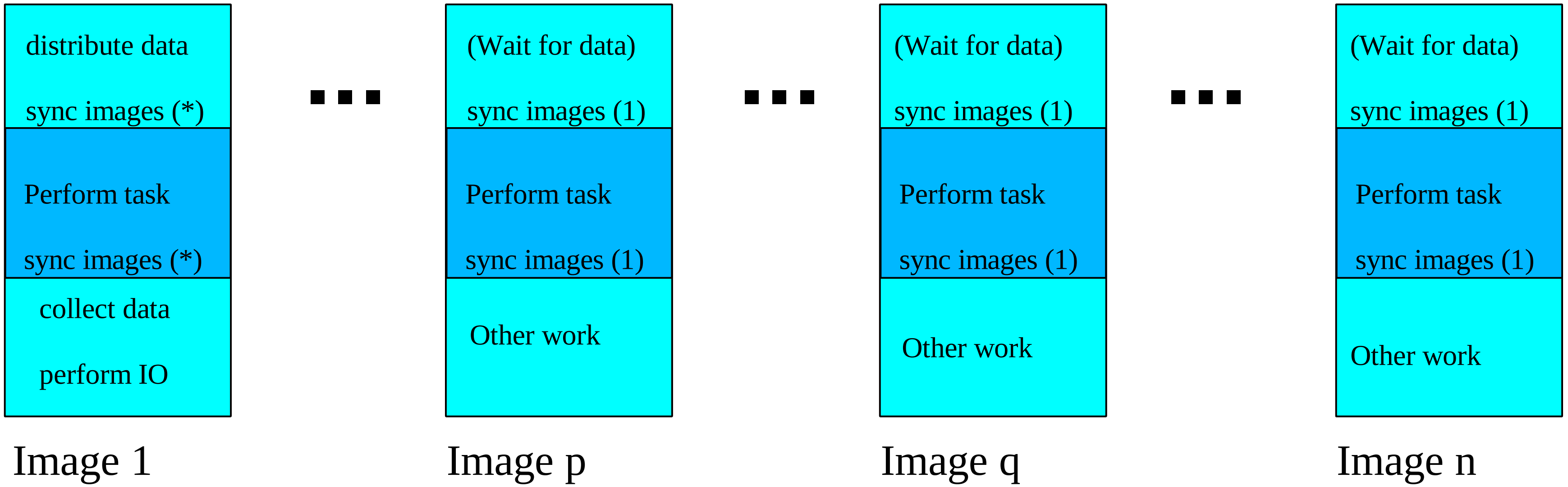
- “Global view” on coarray
- Fortran intrinsic CSHIFT only works on local arrays
- this_image(): index of the executing image
- num_images(): the total number of images

Synchronization primitives

- Multi-image synchronization
  - SYNC ALL
    - Synchronization across all images
  - SYNC IMAGES
    - Synchronization on a list of images
- Memory barrier
  - SYNC MEMORY
- Image serialization
  - CRITICAL (“the big hammer”)
    - Allows one image to execute the block at a time
  - LOCK: provide fine-grained disjoint data access
    - Simple lock support
- Some statements may imply synchronization
  - SYNC ALL implied when the application starts

Example: SYNC IMAGES

- Master image to distribute and collect data




```
if (this_image() == 1) then
   call distributeData ()
   SYNC IMAGES (*)
   call performTask ()
   SYNC IMAGES (*)
   call collectData ()
   call performIO ()
else
   SYNC IMAGES (1)
   call performTask ()
   SYNC IMAGES (1)
   call otherWork ()
end if
```

- Good:
  - Image q starts performTask once its own data are set – no wait for image p
  - Works well on a balanced system
    - Bad if the load is not balanced
  - Efficient if collaboration among small set of images

Atomic load and store

- Two atomic operations provided for spin-lock-loop
  - ATOMIC_DEFINE and ATOMIC_REF

```
LOGICAL(ATOMIC_LOGICAL_KIND),SAVE :: LOCKED[*] = .TRUE.
LOGICAL :: VAL
INTEGER :: IAM, P, Q
IAM = THIS_IMAGE()
IF (IAM == P) THEN
    ! preceding work
    SYNC MEMORY
    CALL ATOMIC_DEFINE (LOCKED[Q], .FALSE.)
    SYNC MEMORY
ELSE IF (IAM == Q) THEN
    VAL = .TRUE.
    DO WHILE (VAL)
        CALL ATOMIC_REF (VAL, LOCKED)
    END DO
    SYNC MEMORY
    ! Subsequent work
END IF
```

CAF implementation and Performance studies

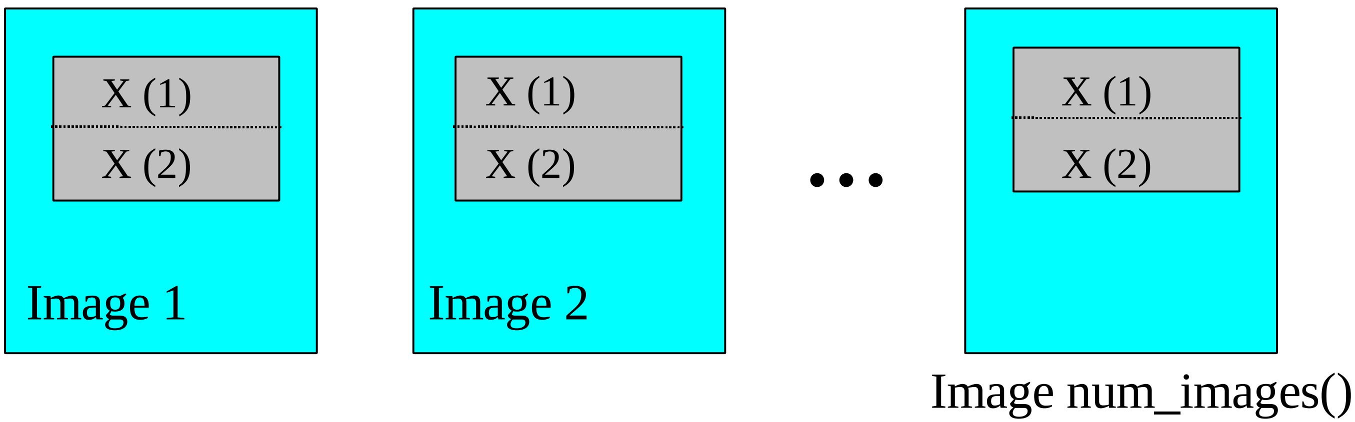
- Existing coarray implementations
  - Cray
  - Rice University
  - G95
- Coarray applications
  - Most on large distributed systems
  - e.g. ocean modeling
- Performance evaluation
  - A number of performance studies have been done
  - CAF Fortran 90 + MPI
- IBM is implementing coarrays
  - CAF and UPC on a common run-time

Standardization status

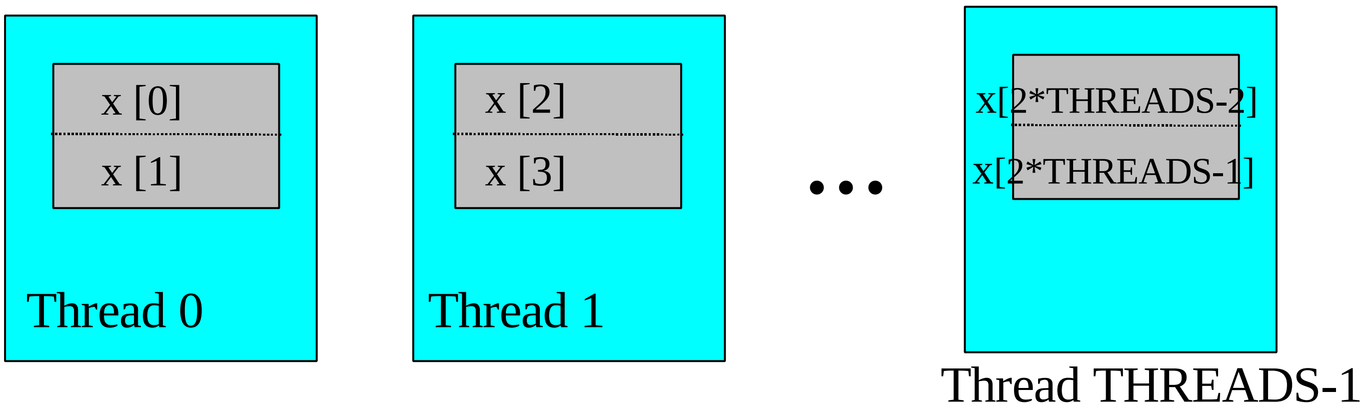
- Coarray is in base language of Fortran 2008
  - Could be finalized this May
  - Standard to be published in 2010
  - Fortran to be the first general purpose language to support parallel programming
- The coarray TR (future coarray features)
  - TEAM and collective subroutines
  - More synchronization primitives
    - notify / query (point – to – point)
  - Parallel IO: multiple images on same file

Comparison between CAF and UPC

CAF : REAL :: X(2)[*]



UPC : shared [2] float x[2*THREADS]



Coarrays and MPI

- Early experience demonstrated coarrays and MPI can coexist in the same application
- Migration from MPI to coarray has shown some success
  - Major obstacle: CAF is not widely available
- Fortran J3 committee willing to work with MPI forum
  - Two issues Fortran committee is currently working on to support:
    - C interop with void *

```
void * buf; (C)
TYPE(*), dimension(...) :: buf (Fortran)
```

    - MPI nonblocking calls: `MPI_ISEND, MPI_IRECV and MPI_WAIT`

MPI:
```
if (master) then
  r(1) = reynolds
  ...
  r(18) = viscosity
  call mpi_bcast(r,18,real_mp_type,
                 masterid,
                 MPI_comm_world,
                 ierr)
else
  call mpi_bcast(r, 18,
                 real_mp_type,
                 masterid,
                 MPI_comm_world,
                 ierr)
  reynolds = r(1)
   ...
   viscosity = r(18)
endif
```

(Ashby and Reid, 2008)

CAF:
```
sync all
if (master) then
  do i=1, num_images()-1
    reynolds[i] = reynolds
    ...
    viscosity[i] = viscosity
    end do
end if
sync all
```

Or simply:

```
sync all
reynolds = reynolds[masterid]
...
viscosity = viscosity[masterid]
```

Q & A