CAF 2.0: A Next-generation Coarray Fortran

Laksono Adhianto, John Mellor-Crummey, and Bill Scherer

Department of Computer Science, Rice University

WPSE Workshop, Tsukuba, Japan

25 March 2009

QuickTime™ and a
decompressor
are needed to see this picture.

Outline

- Coarray Fortran 1.0 language recap
- Design Goals and Principles
- Design Feature Details
- Matters of Syntax
- Implementation Status

Coarray Fortran (CAF) 1.0

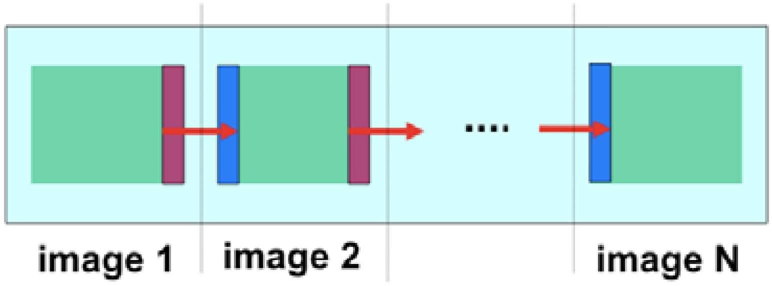
- Explicitly-parallel extension of Fortran 90/95
  - Defined by Numrich and Reid
- Global address space SPMD parallel programming model
  - One-sided communication
- Simple two-level memory model for locality management
  - Local vs. remote memory
- Programmer control over performance-critical decisions
  - Data partitioning
  - Communication
- Suitable for mapping to a range of parallel architectures
  - Shared memory, message passing, hybrid

CAF Programming Model Features

- SPMD process images
  - Fixed number of images during execution
  - Images operate asynchronously
- Both private and shared data
  - **real x(20, 20)** a private 20x20 array in each image
  - **real y(20, 20) [*]** a shared 20x20 array in each image
- Simple one-sided shared-memory communication
  - **x(:,j:j+2) = y(:,p:p+2) [r]** copy columns from p:p+2 into local columns
- Synchronization intrinsic functions
  - **sync_all** – a barrier and a memory fence
  - **sync_mem** – a memory fence
  - **sync_team([notify], [wait])**
    - **notify** = a vector of process ids to signal
    - **wait** = a vector of process ids to wait for, a subset of notify

Accessing Remote Co-array Data

```
me = this_image()
npe = num_images()
if (me < npe) notify(me+1)
if (me > 1) then
   query(me-1)
   a(:,1:2) = a(:,19:20)[me-1]
   notify(me-1)
endif
if (me < npe) query(me+1)
```



Recent Activity in CAF

- Effort to incorporate CAF features into Fortran 2008 standard as an extension of Fortran 2003 features
  - Features fall short of what is truly needed
  - We’ve published a detailed critique -- URL at end of the talk
    - Largely based on the CAF 1.0 design
    - Using the language of yesterday to solve the problems of tomorrow!
- This talk will focus on what we’ve been doing since then
  - New features
  - Support for new hardware
- This is work in progress!

Partitioned Global Address Space (PGAS)

- Global Address Space
  - One-sided communication (GET/PUT)
  - Simpler than message passing
- Programmer-controlled performance factors:
  - Data distribution and locality control
  - Computation partitioning
  - Communication placement
- Data movement and sync are language primitives
  - Enables compiler-based communication optimizations

The PGAS Model

- Data movement and synchronization are expensive
- Reduce overheads:
  - Co-locate data with processors
  - Aggregate multiple accesses to data
  - Overlap communication and computation

CAF 2.0 Design Goals

- Facilitate the construction of sophisticated parallel applications and parallel libraries
- Scale to emerging petascale architectures
- Exploit multicore processors
- Deliver top performance: enable users to avoid exposing or overlap communication latency
- Support development of portable high-performance programs
- Interoperate with legacy models such as MPI
- Support irregular and adaptive applications

CAF 2.0 Design Principles

- Largely borrowed from MPI 1.1 design principles
- Safe communication spaces allow for modularization of codes and libraries by preventing unintended message conflicts
- Allowing group-scoped collective operations avoids wasting overhead in processes that are otherwise uninvolved (potentially running unrelated code)
- Abstract process naming allows for expression of codes in libraries and modules; it is also mandatory for dynamic multithreading
- User-defined extensions for message passing and collective operations interface support the development of robust libraries and modules
- The syntax for language features must be convenient

Design Features Overview: Orthogonal Concerns

- Participation: Teams of processors
- Organization: Topologies
- Communication: Co-dimensions
- Mutual Exclusion: Extended support for locking
- Multithreading: Dynamic processes
- Coordination: Events
- Collective Synchronization: Barriers and team-based reductions

Teams and Groups

- Partitioning and organizing images for computation
  - Teams are local notions; groups are shared
  - Creating a group from a team is a collective operation
  - Groups are immutable once created; teams may be modified freely
  - Collective operations work with groups
- Predefined teams (immutable):
  - CAF_WORLD: contains all images, numbered with rank 1..NPE
  - CAF_SELF: contains just the local image; size is always 1
- Creating new teams
  - Splitting or subsetting an existing team
  - Intersection or union of existing teams
  - Reordering images based on topology information
- Implementation note: team representation
  - If each team member stores a vector of the process images in the team, quadratic space overhead, which is not scalable
  - Distributed representation, caching of team members?

Splitting Teams

- TEAM_Split (team, color, key, team_out)
  - team: team of images (handle)
  - color: control of subset assignment. Images with the same color are in the same new team
  - key: control of rank assigment (integer)
  - team_out: receives handle for this image's new team
- Example:
  - Consider p processes organized in a q ✕ q grid
  - Create separate teams each row of the grid

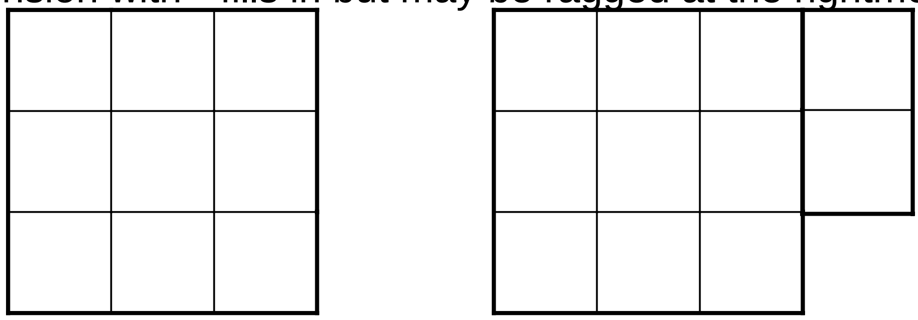
```
IMAGE_TEAM team
integer rank, row
rank = this_image(TEAM_WORLD)
row = rank/q
call team_split(TEAM_WORLD, row, rank, team)
```


Topologies

- Permute the indices of a team or of all processors
- ZPL-style movement for programmer convenience
  - Really just functions on the processor numbers
  - Binary tree example:
    - Parent = MYPE/2; Left = MYPE*2; Right = MYPE*2 + 1
    - x(i,:)[Left()] = x(:,i)[Right()] ! transpose x between siblings
- Cartesian topology is "just" a special case
  - Very important in traditional HPC apps
  - Modern apps are increasingly chaotic
    - Irregular/unstructured mesh, AMR
- Graph topology to support the general case
  - Arbitrary connectivity between processor nodes
- Dynamic modification of topologies (by changing teams) supports dynamic/adaptive applications

Co-dimensions

- Declaration:
  - real :: X(:,:)[3,*]
- Fortran constraint: all leading co-dimensions MUST be constants (unless allocatable)
- Dimension with * fills in but may be ragged at the rightmost edge



- When is this useful?
  - only provides right abstraction for dense arrays, simple boundaries
  - only useful in practice when MOD(npe,3) == 0: brittle software
- Can effect the same functionality via topologies

Mutual Exclusion

- Critical section from draft spec
  - Named critical regions
  - Static names - doesn't work for fine-grained locking of dynamic data structures
- Built-in LOCK type

```
CAF_LOCK L
LOCK(L)
!…use data protected by L here…
UNLOCK(L)
```

Lock Sets: Safer Multi-locking

- Big problem with locks: Deadlock
  - Results from lock acquisition cycles
- Take a cue from two-phase locking
  - Acquire all locks as one logical processing step
  - Total ordering over locks avoids cycles between processes during acquisition
- Lockset abstraction supports this idiom for programmer convenience
  - Add or remove individual locks to a runtime set
  - Acquire operation on the set acquires individual locks in canonical order

Dynamic Multithreading

- Spawn
  - Create local or remote asynchronous threads by calling a procedure declared as a co-function
    - Simple interface for function shipping
  - Local threads can exploit multicore parallelism
  - Remote threads can be created to avoid latency when manipulating remote data structures
- Finish
  - Terminally strict synchronization for (nested) spawned sub-images
  - Orthogonal to procedures (like X10 and unlike Cilk)
    - Exiting a procedure does not require waiting on spawned sub-images

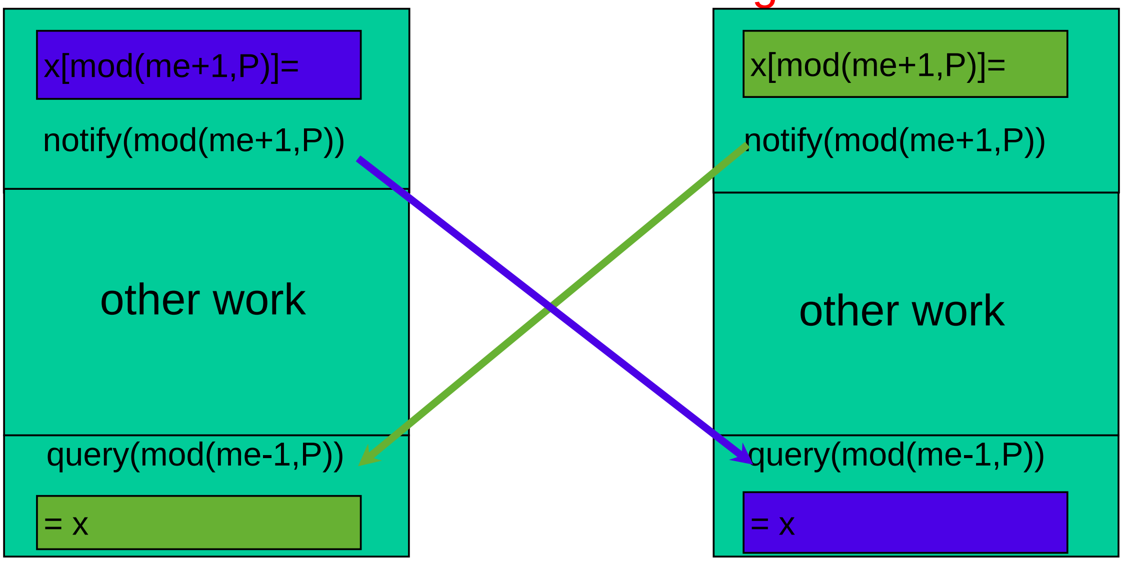
Safe Communication Spaces

- Event object for anonymous pairwise coordination
- Safe synchronization space: can allocate as many events as possible
- Notify: nonblocking, asynchronous signal to an event; a pairwise fence between sender and target image
- Wait: blocking wait for notification on an event or event set
- Waitany: return the ready event in an event set

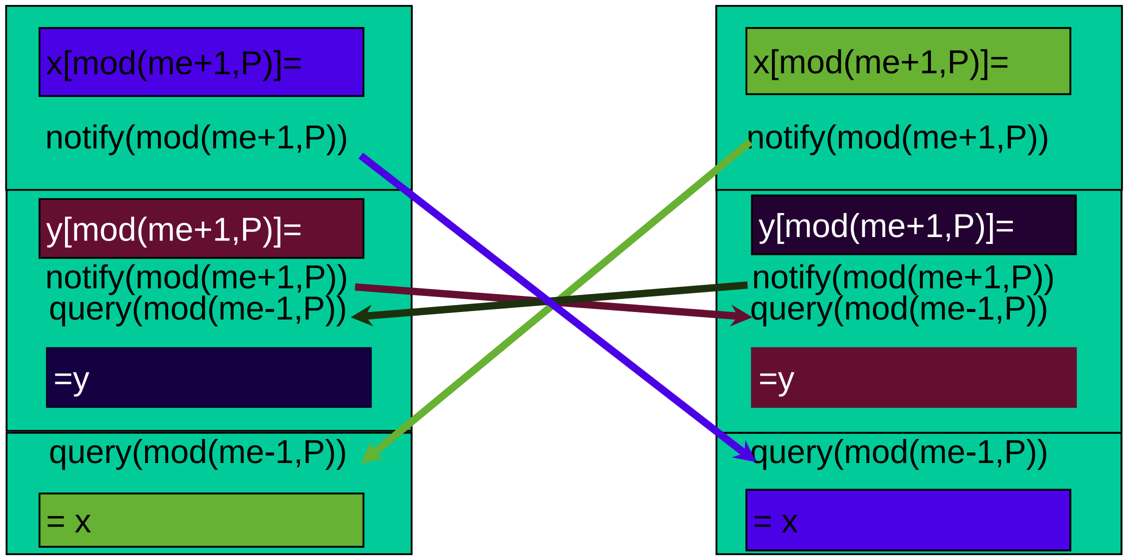
Multiple Communication Channels

- Multiple Communication Channels

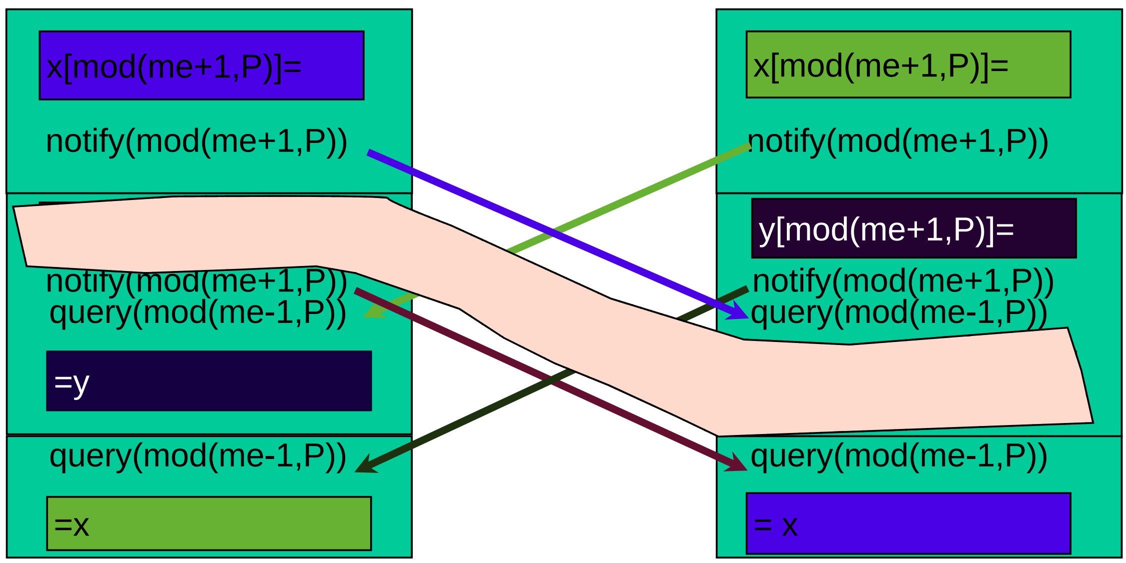
Consider the following



What if “Other Work” was Synchronized



Lack of Encapsulation Leads to Races



Collective Communication

- Sync: barrier within a team
- All the standard collective operations
  - sum, product, maxloc, maxval, minloc, minval
  - any, all, count, alltoall
- Coreduce: collective communication within a team
- User-defined reductions for extensibility

Syntactic Convenience: Critical Sections

- Structured critical section construct for mutual exclusion

```
critical (Lock | Lockset)
    … ! Critical region here
end critical
```

- Impossible to miss releasing a lock
- Does not support hand-over-hand locking
- Names vs. locks: static vs. dynamic
  - Cannot implement dynamic data structures with a lock in each node if the set of locks is static!

Syntactic Convenience: Team Namespaces

- Specify a default team for data access
- Retain ability to override with explicit team specifier

```
with team (air) ! sets default team
    a(:)[1] = b(:)[2@ocean]
    ! Image 1 from the air team gets data
    ! from image 2 of the ocean team
end with
```


Implementation Status

- CAF 1.0 compiler was based on Open64 framework
  - Very large and fragile codebase
  - Difficult to modify one piece without breaking something else
- New front-end compiler based on Rose
- Compiler and runtime library implementation in progress
  - Thinnest possible runtime for maximal performance
- GasNet substrate for interprocess communication in the runtime
- Strategy for dope vector management through judicious use of Cray pointers

Thank you!

- Our critique of coarray support in the Fortran 2008 draft standard may be found online at:
  - http://www.j3-fortran.org/doc/meeting/183/08-126.pdf
- For more information:
  - Laksono Adhianto: laksono@rice.edu
  - John Mellor-Crummey: johnmc@rice.edu
  - Bill Scherer (me): scherer@rice.edu
- More details soon -- stay tuned!