

A Proposal for Adding Team Objects to Co-Array Fortran

Robert W. Numrich

Minnesota Supercomputing Institute
University of Minnesota
Minneapolis, Minnesota USA
rwn@msi.umn.edu

UNIVERSITY OF MINNESOTA

- ▶ What is partitioned?
 - ▶ The physical processors?
 - ▶ The virtual processes?
 - ▶ The memory?
 - ▶ The work?
 - ▶ The data?
 - ▶ The threads?
 - ▶ The cores?
 - ▶ The nodes?
- ▶ Who is responsible for partitioning?
 - ▶ The programmer?
 - ▶ The compiler?
 - ▶ The operating system?
 - ▶ The library?
- ▶ Is the default view local or global?
 - ▶ The global composite of local objects?
 - ▶ The local component of the global object?

Suppose we have decided what a PARTITION is:

- ▶ What name do we assign to a partition?
- ▶ How many partitions are there?
- ▶ Is the number of partitions fixed?
- ▶ Can we make sub-partitions?
- ▶ Can we make super-partitions?
- ▶ What is the affinity between data and work for a partition?
- ▶ Can a thread assigned to one partition work on data assigned to a different partition?
- ▶ Is data in one partition visible to another partition?
- ▶ What is the memory consistency model?
- ▶ What are the synchronization mechanisms?
- ▶ Can a thread be reassigned to a different partition?
- ▶ Can data be reassigned to a different partition?

Comparing models

	Fortress	X10	Chapel	UPC	CAF	MPI
Partitions	Region	Place	Locale	Thread	Image	Communicator
Number	Dynamic?	Fixed	Fixed	Fixed	Fixed	Fixed
Name	Region	num_places	numLocales	Threads	num_images	Comm_size
Affinity	object.region()	here	Domain	myThread	this_image	Comm_rank
Distributions	yes	yes	yes	yes	no	no
Execution Model	data parallel recursive	data parallel async	data parallel	SPMD	SPMD hybrid	SPMD hybrid
Sub-partitions	at region	at clocks?	on locale	-	Team	Comm_group

The co-array model

- ▶ SPMD with a fixed number of *virtual* images
 - ▶ An image corresponds to a logical partition of global memory
 - ▶ `num_images()`, `this_image()`
 - ▶ The physical memory for each image is assigned to some local memory by the run-time system.
- ▶ Physical processors are assigned to work on a set of images whose memory is local to the processor.
 - ▶ one processor to one image
 - ▶ one processor to many images
 - ▶ many processors to one image
 - ▶ many processors to many images
- ▶ Images are dereferenced by multi-rank co-dimensions.
 - ▶ All variables are local to an image.
 - ▶ Only variables declared with co-dimensions are visible across images.
 - ▶ Co-indices are dereferenced relative to all the images.
- ▶ Allocate/deallocate of co-arrays is collective across all images.
- ▶ `sync all` is collective across all images.

Fortran is a modern language

- ▶ Fortran 2003
 - ▶ Object-oriented
 - ▶ Portable C interface
 - ▶ Parametrized derived types
 - ▶ Strong typing through interfaces
- ▶ Fortran 2008
 - ▶ Co-arrays
 - ▶ First parallel addition to the language
 - ▶ Open-source g95 has implemented co-arrays (but not objects).

- ▶ Object-oriented
 - ▶ Objects
 - ▶ User-defined derived types
 - ▶ Type-bound procedures
 - ▶ Type constructors
 - ▶ Type finalization
 - ▶ Abstract types
 - ▶ Inheritance
 - ▶ Deferred procedure bindings
 - ▶ Overloaded generic procedures
 - ▶ Polymorphism

- ▶ "MPI Communicator defined as an abstract object"
- ▶ A team is a subset of images.
 - ▶ Teams are created by constructors.
 - ▶ Different teams assigned to different applications.
 - ▶ Synchronization within a team or a subteam.
 - ▶ Allocation/deallocation of memory within a team.
 - ▶ Collectives within a team.
 - ▶ Synchronization within and between teams.
 - ▶ Split barrier (notify/wait).
- ▶ Dereference co-indices relative to the team
 - ▶ $y = x[p,q]$ refers to image $[p,q]$ relative to the team.
 - ▶ Co-arrays contain information identifying their team.
 - ▶ Information added to dope vector at allocation.

Class AbstractTeam

```
module ClassAbstractTeam
  Type,public,abstract :: AbstractTeam
  contains
    procedure,public,pass(this),deferred :: newTeam
    procedure,public,pass(this),deferred :: isMyTeam
    procedure,public,pass(this),deferred :: getTeamList
    procedure,public,pass(this),deferred :: getParent
    procedure,public,pass(this),deferred :: getTeamSize
    procedure,public,pass(this),deferred :: getTeamIndex
    procedure,public,pass(this),deferred :: run
    procedure,public,pass(this),deferred :: allocate
    procedure,public,pass(this),deferred :: deallocate
    procedure,public,pass(this),deferred :: sync
    procedure,public,pass(this),deferred :: notify
    procedure,public,pass(this),deferred :: wait
    procedure,public,pass(this),deferred :: sum
    procedure,public,pass(this),deferred :: max
    procedure,public,pass(this),deferred :: min
  end Type AbstractTeam
  abstract interface
    logical function isMyTeam(this)
    import :: AbstractTeam
    Class(AbstractTeam),intent(in) :: this
  end interface
end module ClassAbstractTeam
```

Default AllTeam Extends AbstractTeam

```
Type(AllTeam) :: all  
real,allocatable :: x[:,:]
```

```
call all%newTeam()      ! all = AllTeam()
```

```
call all%sync()
```

```
call all%allocate(x[p,*])
```

```
s=all%sum(x)
```

```
Type(SomeTeam) :: this  
Type(RowTeam) :: rowX  
real,allocatable :: x[:,:]
```

```
allocate%this(x[p,*])
```

```
call rowX%newTeam(x,q)
```

```
s = rowX%max(x)
```

ContiguousTeam Extends AbstractTeam

```
Type(ContiguousTeam) :: air, ocean
```

```
call air%newTeam(1,num_images()/2)
```

```
call ocean%newTeam(num_images()/2+1, num_images/2)
```

```
if(air%isMyTeam()) then
```

```
  call air%run()
```

```
elseif(ocean%isMyTeam()) then
```

```
  call ocean%run()
```

```
endif
```

Synchronization among teams

```
Type(AllTeam),pointer :: airParent,oceanParent
airParent => air%getParent()
oceanParent => ocean%getParent()
if(air%isMyTeam()) then
  call air%sync()
  call air%sync(parent)
  call air%sync(ocean)
  call air%notify(ocean)
elseif (ocean%isMyTeam()) then
  call ocean%sync()
  call ocean%sync(parent)
  call ocean%sync(air)
  call ocean%wait(air)
endif
```

ocean%run() procedure

```
subroutine run(this,air,buffer)
  Type(ContiguousTeam),intent(in) :: this,air
  real :: buffer(1:)[*]
  real,allocatable :: oceanData(:,:)[:,:]
  !——local arrays——
  this%allocate(oceanData(m,n)[p,*])
  airImage = air%getTeamList(1)
  if(this%getTeamIndex() == 1) buffer(:) = oceanBoundaryConditions(:)
  call this%sync(air)
  airBoundayConditions(:) = buffer(:)[airImage]
  !——some ocean work——
  if(this%getTeamIndex() == 1) buffer(:) = oceanBoundaryConditions(:)
  call this%sync(air)
  airBoundayConditions(:) = buffer(:)[airImage]
  !——some more ocean work——
  this%deallocate(oceanData)
end subroutine run
```

- ▶ A team is an abstract object something like an MPI communicator.
- ▶ It allows for clean definitions for collectives.
- ▶ It resolves issues concerning synchronization.
- ▶ It allows programmers to write applications that work the same for all images or for a team of images.
- ▶ It requires adding team state to co-array objects.
- ▶ What would a team object look like in other languages?

- ▶ J. Reid and R.W. Numrich, Co-arrays in the next Fortran Standard, *Scientific Programming*, 15(1), pp. 9-26, 2007.
- ▶ R.W. Numrich, A Parallel Numerical Library for Co-Array Fortran, *Proceedings PPAM05*, pp. 960-969, 2005.
- ▶ R.W. Numrich, Parallel numerical algorithms based on tensor notation and Co-Array Fortran syntax, *Parallel Computing*, 31, pp. 588-607, 2005.
- ▶ R.W. Numrich, CafLib Users' Manual: Release 1.2, technical report.