

# Introduction to Co-Array Fortran

Robert W. Numrich

Minnesota Supercomputing Institute  
University of Minnesota, Minneapolis

and

Goddard Space Flight Center  
Greenbelt, Maryland

[rwn@msi.umn.edu](mailto:rwn@msi.umn.edu)

University of Minnesota

# What is Co-Array Fortran?

- Co-Array Fortran is one of three simple language extensions to support explicit parallel programming.
  - Co-Array Fortran (CAF) Minnesota
  - Unified Parallel C (UPC) GWU-Berkeley-NSA-Michigan Tech
  - Titanium ( extension to Java) Berkeley
  - [www.pmodels.org](http://www.pmodels.org)

# Programming Model

- Single-Program-Multiple-Data (SPMD)
- Fixed number of processes/threads/images
- Explicit data decomposition
- All data is local
- All computation is local
- One-sided communication thru co-dimensions
- Explicit synchronization
- Is this model still adequate?

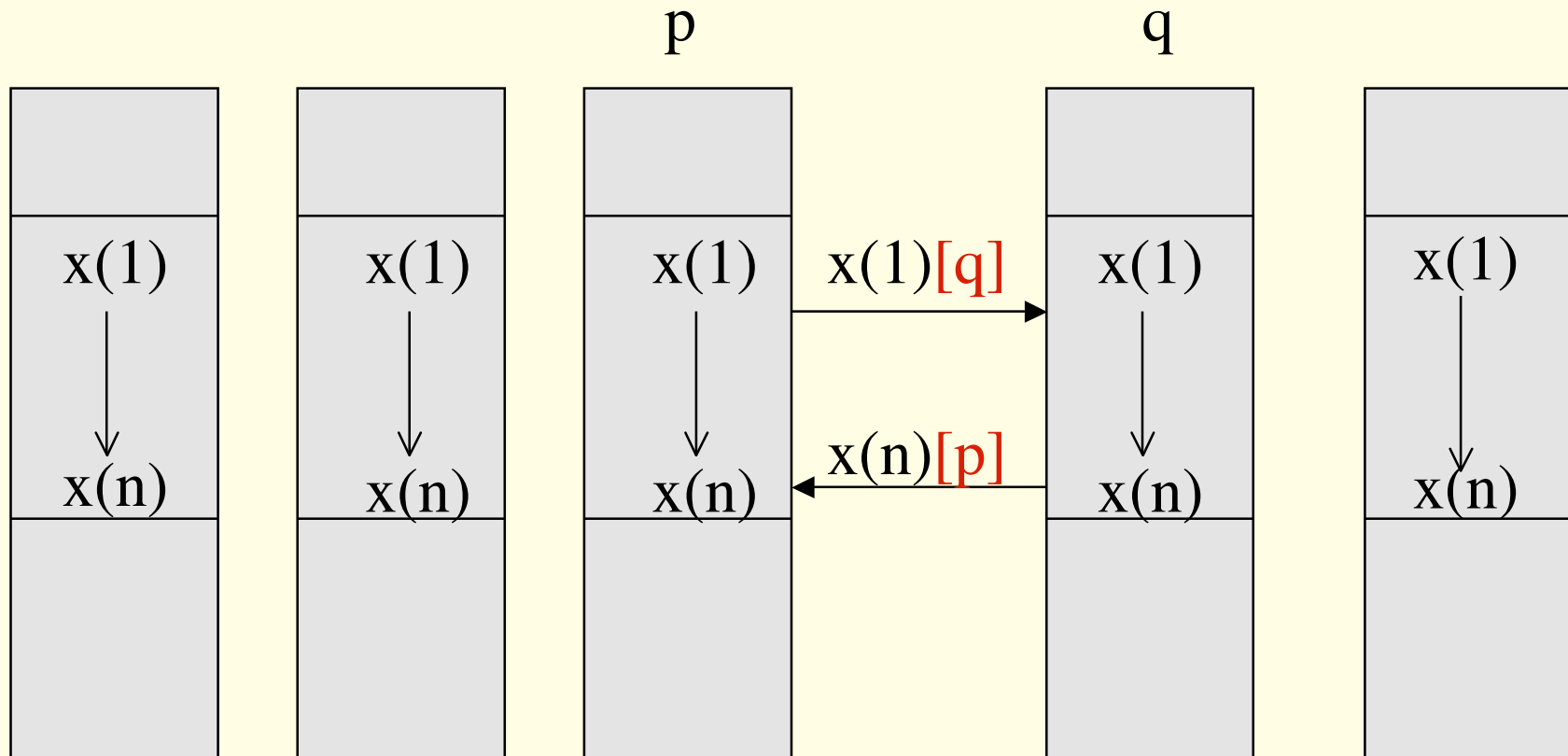
# Co-Array Fortran Execution Model

- The number of images is fixed and each image has its own index, retrievable at run-time:
  - $1 \leq \text{num\_images}()$
  - $1 \leq \text{this\_image}() \leq \text{num\_images}()$
- Each image executes the same program independently of the others.
- The programmer inserts explicit synchronization and branching as needed.
- An “object” has the same name in each image.
- Each image works on its own local data.
- An image moves remote data to local data through, and only through, explicit co-array syntax.

# Declaration of a Co-Array

```
real :: x(n)[*]
```

# Co-Array Fortran Memory Model



# What Do Co-Dimensions Mean?

real :: x(n)[p,q,\*]

1. Replicate an array of length n, one on each image.
2. Build a map so each image knows how to find the array on any other image.
3. Organize images in a logical (not physical) three-dimensional grid.
4. The last co-dimension acts like an assumed size array: \*  $\Rightarrow$  num\_images()/(pxq)

# Examples of Co-Array Declarations

**real :: a(n)[\*]**

**complex :: z[0:\*]**

**integer :: index(n)[\*]**

**real :: b(n)[p, \*]**

**real :: c(n,m)[0:p, -7:q, +11:\*]**

**real, allocatable :: w(:)[:]**

**type(field) :: maxwell[p,\*]**

# Communication Using CAF Syntax

$$\mathbf{y}(\cdot) = \mathbf{x}(\cdot)[\mathbf{p}]$$

$$\mathbf{x}(\mathbf{index}(\cdot)) = \mathbf{y}[\mathbf{index}(\cdot)]$$

$$\mathbf{x}(\cdot)[\mathbf{q}] = \mathbf{x}(\cdot) + \mathbf{x}(\cdot)[\mathbf{p}]$$

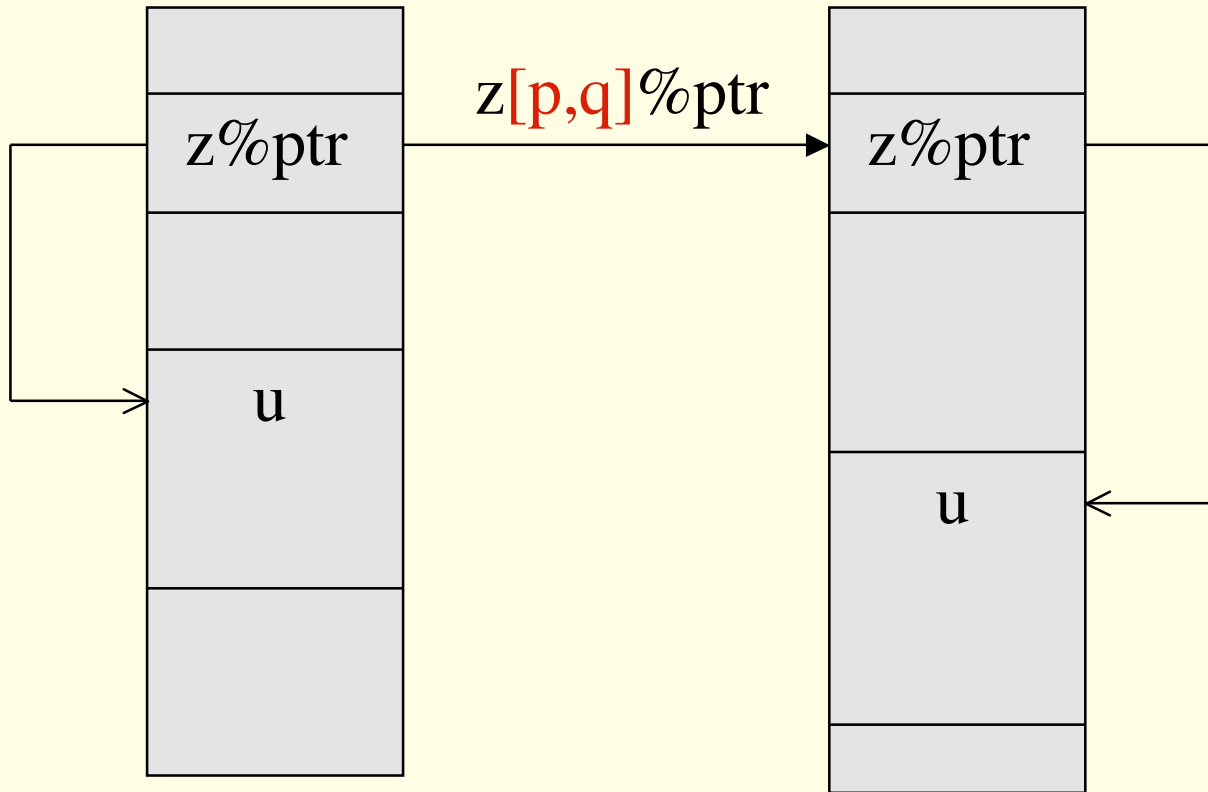
Absent co-dimension defaults to the local object.

# An Example from the UK Met Unified Model

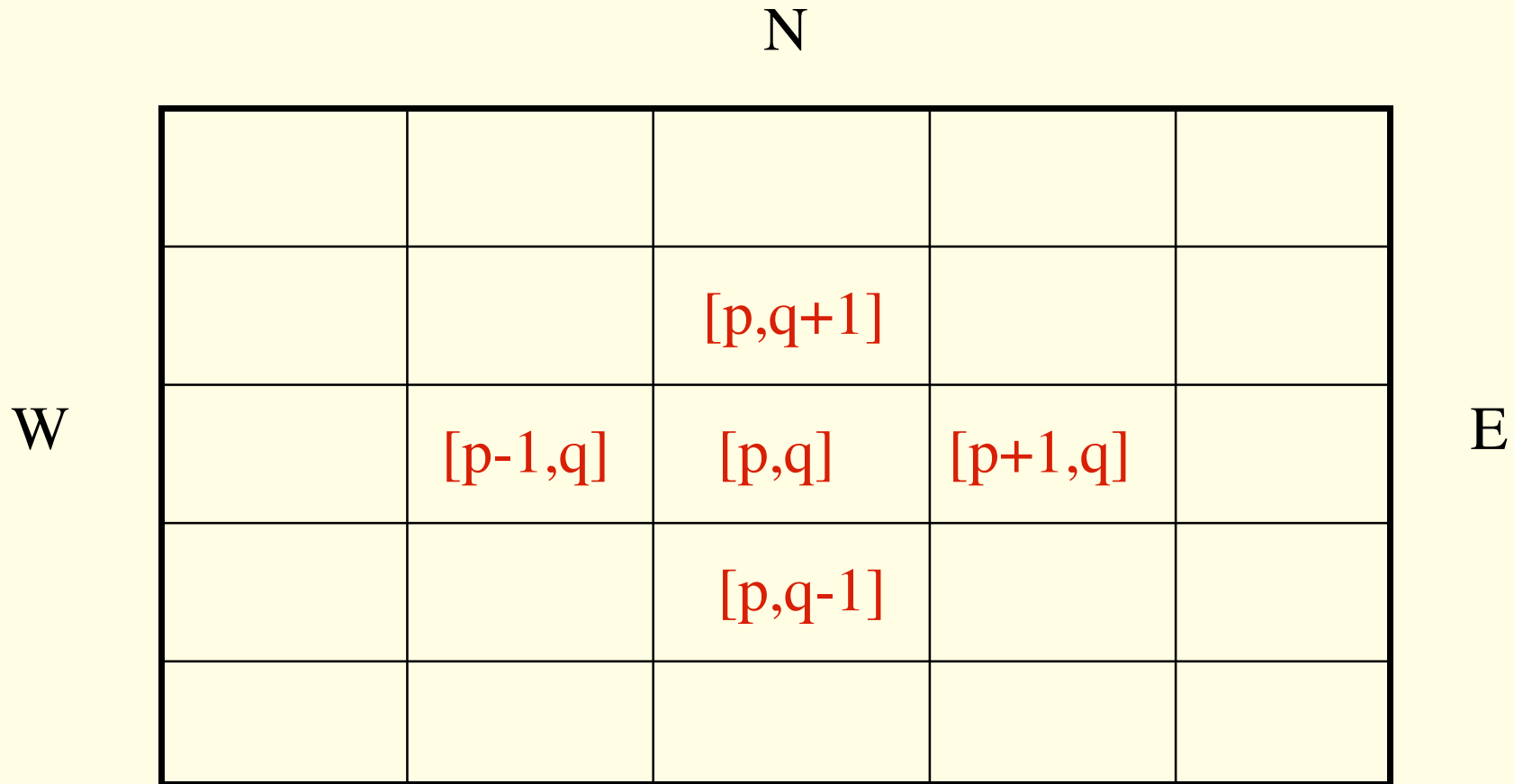
# Alias to Local Fields

- real, allocatable :: u(:, :, :)
- allocate( u(0:m+1, 0:n+1, levels))
- type(field) :: z[p, \*]
  
- z%ptr => u

# Alias to Remote Data



# Problem Decomposition and Co-Dimensions



S

# Cyclic Boundary Conditions East-West Direction

real,dimension [p,\*] :: z

myP = `this_image(z,1)`                   !East-West

West = myP - 1

if(West < 1) West = nProcEW           !Cyclic

East = myP + 1

if(East > nProcEW) East = 1           !Cyclic

# East-West Halo Swap

- Move last row from west to my first halo

$u(0,1:n,1:lev) = z[\text{West,myQ}]\%ptr(m,1:n,1:lev)$

- Move first row from east to my last halo

$u(m+1,1:n,1:lev)=z[\text{East,myQ}]\%Field(1,1:n,1:lev)$

# Total Time (s)

PxQ	SHMEM	SHMEM w/CAF SWAP	MPI w/CAF SWAP	MPI
2x2	191	198	201	205
2x4	95.0	99.0	100	105
2x8	49.8	52.2	52.7	55.5
4x4	50.0	53.7	54.4	55.9
4x8	27.3	29.8	31.6	32.4

# Co-Array Fortran and “Object-Oriented” Programming Methodology

# CafLib: A Parallel “Class Library” for Co-Array Fortran

- Combine the object-based features of Fortran 95 with co-array syntax to obtain an efficient parallel numerical class library that scales to large numbers of processors.
- Encapsulate all the hard stuff in modules using named objects, constructors, destructors, generic interfaces, dynamic memory management.

# CAF Parallel “Class Libraries”

**use BlockMatrices**

**use BlockVectors**

**type(PivotVector) :: pivot[p,\*]**

**type(BlockMatrix) :: a[p,\*]**

**type(BlockVector) :: x[\*]**

**call newBlockMatrix(a,n,p)**

**call newPivotVector(pivot,a)**

**call newBlockVector(x,n)**

**call luDecomp(a,pivot)**

**call solve(a,x,pivot)**

# LU Decomposition

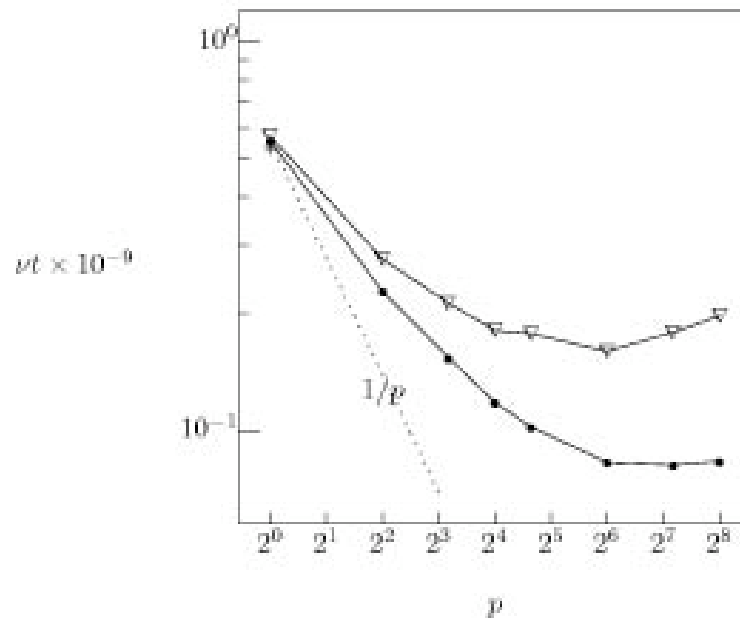


Figure 6: Time as a function of the number of processors  $p = q \times r$  for block-cyclic LU decomposition. The matrix size is  $1000 \times 1000$  with blocks of size  $48 \times 48$ . Time is expressed in dimensionless giga-clock-ticks,  $\nu t \times 10^{-9}$ , as measured on a CRAY-T3E with frequency  $\nu = 300\text{MHz}$ . The dotted line represents perfect scaling. The curve marked with bullets ( $\bullet$ ) is code written in Co-Array Fortran. The curve marked with triangles ( $\nabla$ ) is SCALAPACK code.

# Communication for LU Decomposition

- Row interchange
  - $\text{temp}(:) = a(k,:)$
  - $a(k,:) = a(j,:) [p, \text{myQ}]$
  - $a(j,:) [p, \text{myQ}] = \text{temp}(:)$
- Row “Broadcast”
  - $L0(i:n,i) = a(i:,n,i) [p,p] \quad i=1,n$
- Row/Column “Broadcast”
  - $L1(:, :) = a(:, :) [\text{myP}, p]$
  - $U1(:, :) = a(:, :) [p, \text{myQ}]$

# CAF I/O for Named Objects

**use BlockMatrices**

**use DiskFiles**

**type(PivotVector) :: pivot[p,\*]**

**type(BlockMatrix) :: a[p,\*]**

**type(DirectAccessDiskFile) :: file**

**call newBlockMatrix(a,n,p)**

**call newPivotVector(pivot,a)**

**call newDiskFile(file)**

**call readBlockMatrix(a,file)**

**call luDecomp(a,pivot)**

**call writeBlockMatrix(a,file)**

# What about Synchronization?

# Synchronization Intrinsic Procedures

## **sync\_all()**

Full barrier; wait for all images before continuing.

## **sync\_all(wait(:))**

Partial barrier; wait only for those images in the wait(:) list.

## **sync\_team(list(:))**

Team barrier; only images in list(:) are involved.

## **sync\_team(list(:),wait(:))**

Team barrier; wait only for those images in the wait(:) list.

## **sync\_team(myPal)**

Synchronize with one other image.

# Events

`sync_team(list(:),list(me:me))` post event

`sync_team(list(:),list(you:you))` wait event

# New Sync Intrinsic Procedures ?

- call `teamNotify(teamList(:))`
- call `teamWait(teamList(:))`
- `readyList(:) = teamReady(teamList(:))`

# Co-Array Fortran and Component Frameworks

# Component Frameworks

- Component frameworks provide interfaces between different applications.
- The Common Component Architecture (CCA) is supported by DoE
- The Earth System Modeling Framework (ESMF) is supported by NASA/NOAA
- Can Co-Array Fortran support component frameworks?

# Team Context

- call `setTeam(teamList(:))`
- `real :: x[*]`  $\Rightarrow$  `real :: x[teamList(:)]`
- `allocate(x[*])`  $\Rightarrow$  `allocate(x[teamList(:)])`
- call `syncAll()`  $\Rightarrow$  call `syncTeam(teamList(:))`

# Co-Procedures

```
type x
  real :: a(n)
contains
  method
end type
```

```
type(x) :: y[*]
b(:) = y[p]%a(:)
b(:) = y[p]%method()
```

# Where Can I Try Co-Array Fortran?

# The Co-Array Fortran Standard

- Co-Array Fortran is defined by:
  - R.W. Numrich and J.K. Reid, “Co-Array Fortran for Parallel Programming”, ACM Fortran Forum, 17(2):1-31, 1998
- Additional information on the web:
  - [www.co-array.org](http://www.co-array.org)
  - [www.pmodels.org](http://www.pmodels.org)

# CRAY Co-Array Fortran

- CAF has been a supported feature of Cray Fortran 90 since release 3.1
- CRAY T3E
  - f90 -Z src.f90
  - mpprun -n7 a.out
- CRAY X1
  - ftn -Z src.f90
  - aprun -n7 a.out

# Co-Array Fortran on Other Platforms

- Rice University is developing an open source compiling system for CAF.
  - Runs on the HP-Alpha systems at PSC and GSFC
  - Runs on SGI platforms
- IBM may put CAF on the BlueGene/L machine at LLNL.
- DARPA High Productivity Computing Systems (HPCS) Project wants CAF.
  - IBM, CRAY, SUN