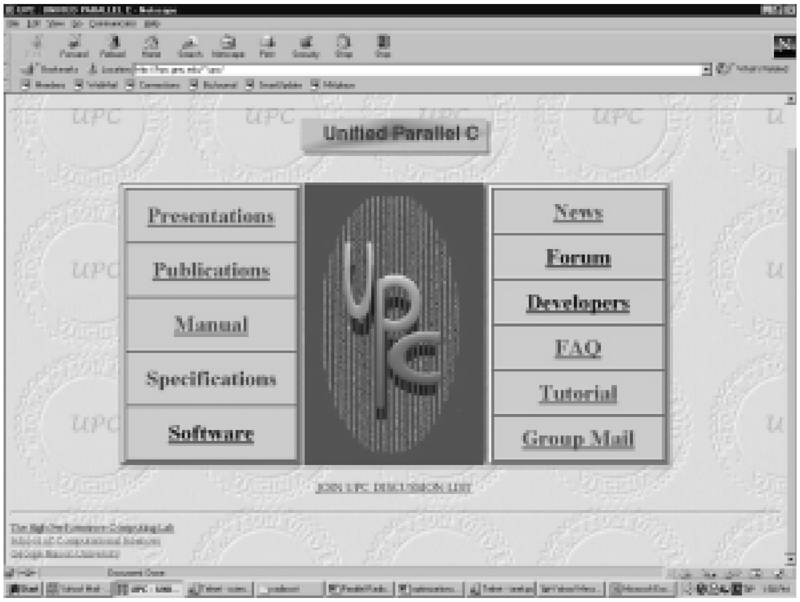
http://upc.gwu.edu



A Co-Array Fortran Tutorial

Robert W. Numrich
Cray Inc.

Outline

1. Philosophy of Co-Array Fortran
2. Co-arrays and co-dimensions
3. Execution model
4. Relative image indices
5. Synchronization
6. Dynamic memory management
7. Example from UK Met Office
8. Examples from Linear Algebra
9. Using “Object-Oriented” Techniques with Co-Array Fortran
10. I/O
11. Summary

1. The Co-Array Fortran Philosophy

The Co-Array Fortran Philosophy

- What is the smallest change required to make Fortran 90 an effective parallel language?
- How can this change be expressed so that it is intuitive and natural for Fortran programmers to understand?
- How can it be expressed so that existing compiler technology can implement it efficiently?

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

Co-Array Fortran on the T3E

- CAF has been a supported feature of Fortran 90 since release 3.1
- f90 -Z src.f90
- mpprun -n7 a.out

Non-Aligned Variables in SPMD Programs

- Addresses of arrays are on the local heap.
- Sizes and shapes are different on different program images.
- One processor knows nothing about another’s memory layout.
- How can we exchange data between such non-aligned variables?

Some Solutions

- **MPI-1**
  - Elaborate system of buffers
  - Two-sided send/receive protocol
  - Programmer moves data between local buffers only.
- **SHMEM**
  - One-sided exchange between variables in COMMON
  - Programmer manages non-aligned addresses and computes offsets into arrays to compensate for different sizes and shapes
- **MPI-2**
  - Mimic SHMEM by exposing some of the buffer system
  - One-sided data exchange within predefined windows
  - Programmer manages addresses and offsets within the windows

Co-Array Fortran Solution

- Incorporate the SPMD Model into Fortran 95 itself
  - Mark variables with co-dimensions
  - Co-dimensions behave like normal dimensions
  - Co-dimensions match problem decomposition not necessarily hardware decomposition
- The underlying run-time system maps your problem decomposition onto specific hardware.
- One-sided data exchange between co-arrays
  - Compiler manages remote addresses, shapes and sizes

The CAF Programming Model

- Multiple images of the same program (SPMD)
  - Replicated text and data
  - The program is written in a sequential language.
  - An “object” has the same name in each image.
  - Extensions allow the programmer to point from an object in one image to the same object in another image.
  - The underlying run-time support system maintains a map among objects in different images.

2. Co-Arrays and Co-Dimensions

What is Co-Array Fortran?

- Co-Array Fortran (CAF) is a simple parallel extension to Fortran 90/95.
- It uses normal rounded brackets ( ) to point to data in local memory.
- It uses square brackets [ ] to point to data in remote memory.
- Syntactic and semantic rules apply separately but equally to ( ) and [ ].

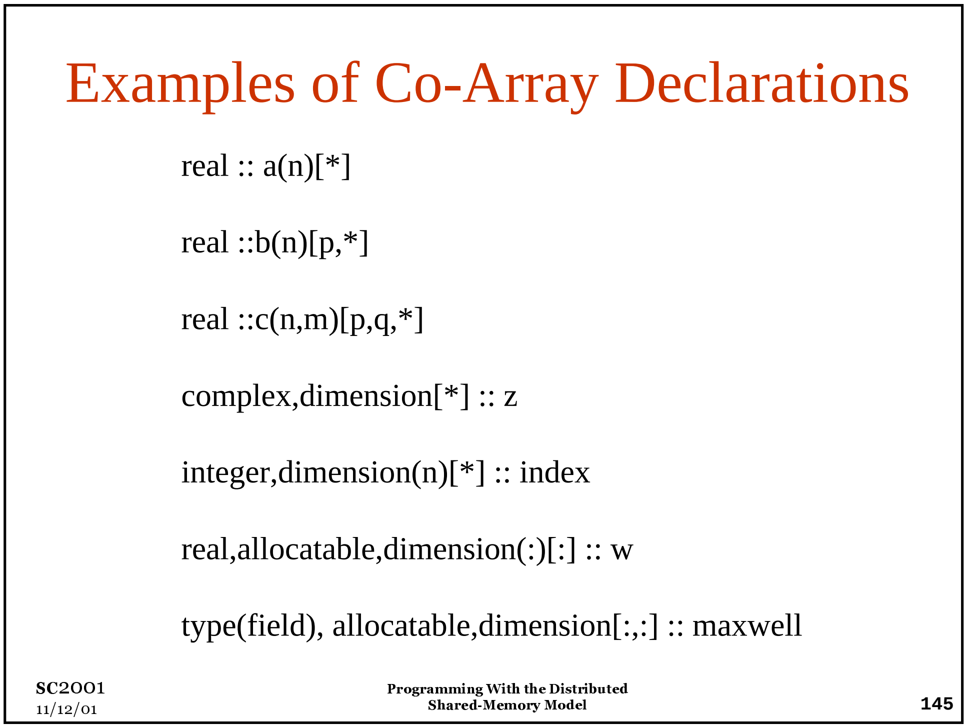
What Do Co-dimensions Mean?

The declaration

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

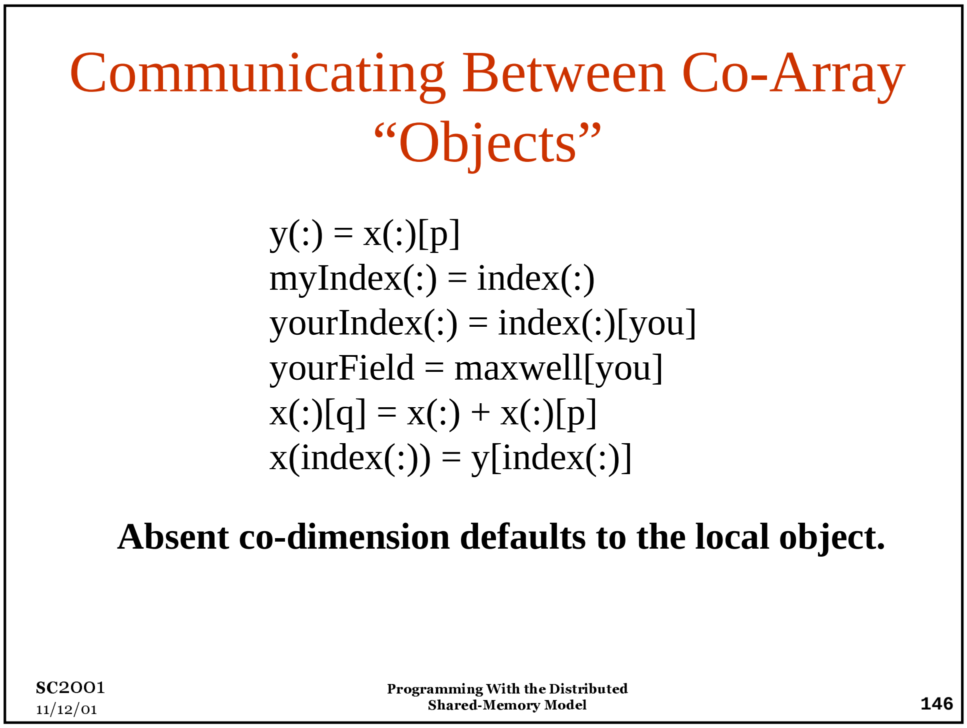
means

1. An array of length n is replicated across images.
2. The underlying system must build a map among these arrays.
3. The logical coordinate system for images is a three dimensional grid of size
4. (p,q,r) where r=num_images()/(pq)



Examples of Co-Array Declarations

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

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

real ::c(n,m)[p,q,*]

complex,dimension[*] :: z

integer,dimension(n)[*] :: index

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

type(field), allocatable,dimension[:,:] :: maxwell
```

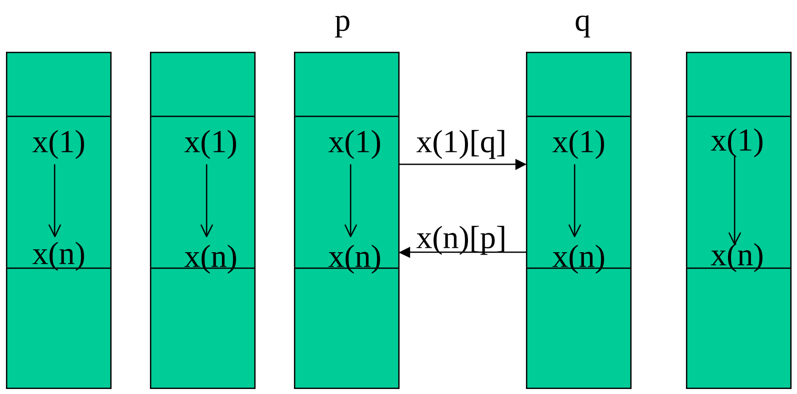


Communicating Between Co-Array “Objects”

```
y(:) = x(:)[p]
myIndex(:) = index(:)
yourIndex(:) = index(:)[you]
yourField = maxwell[you]
x(:)[q] = x(:) + x(:)[p]
x(index(:)) = y[index(:)]
```

Absent co-dimension defaults to the local object.

CAF Memory Model



Example I: A PIC Code Fragment

```
type(Pstruct) particle(myMax),buffer(myMax)[*]
myCell = this_image(buffer)
yours = 0
do mine =1,myParticles
    If(particle(mine)%x > rightEdge) then
        yours = yours + 1
        buffer(yours)[myCell+1] = particle( mine)
    endif
enddo
```

Exercise: PIC Fragment

- Convince yourself that no synchronization is required for this one-dimensional problem.
- What kind of synchronization is required for the three-dimensional case?
- What are the tradeoffs between synchronization and memory usage?

3. Execution Model

The Execution Model (I)

- The number of images is fixed.
- This number can be retrieved at run-time.

num_images() >= 1

- Each image has its own index.
- This index can be retrieved at run-time.

1 <= this_image() <= num_images()

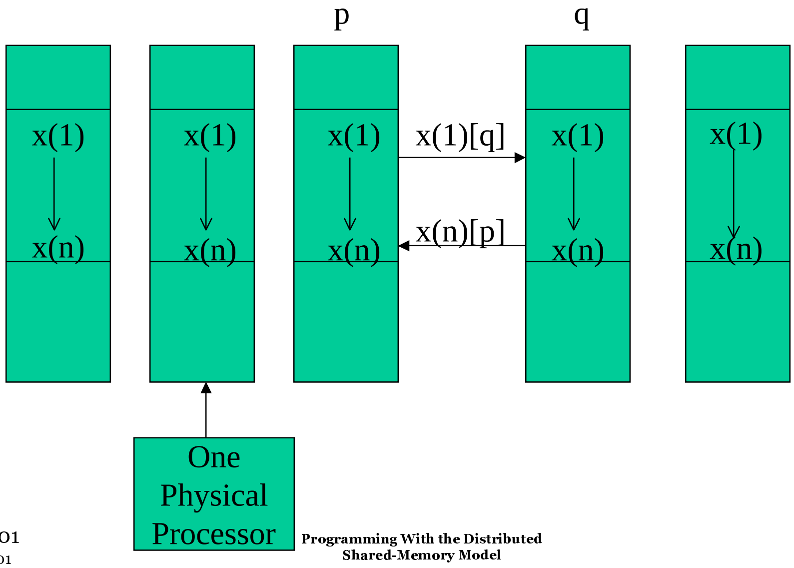
The Execution Model (II)

- Each image executes independently of the others.
- Communication between images takes place only through the use of explicit CAF syntax.
- The programmer inserts explicit synchronization as needed.

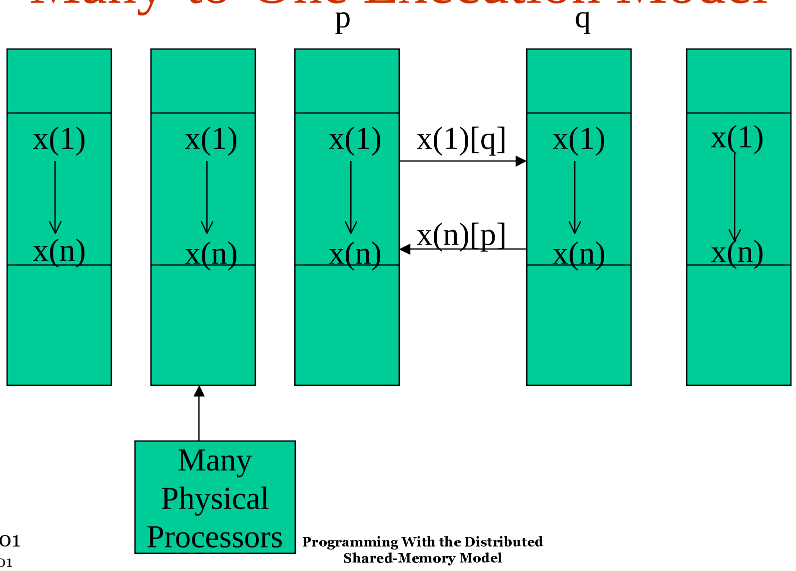
Who Builds the Map?

- The programmer specifies a **logical** map using co-array syntax.
- The underlying run-time system builds the **logical-to-virtual** map and a **virtual-to-physical** map.
- The programmer should be concerned with the logical map only.

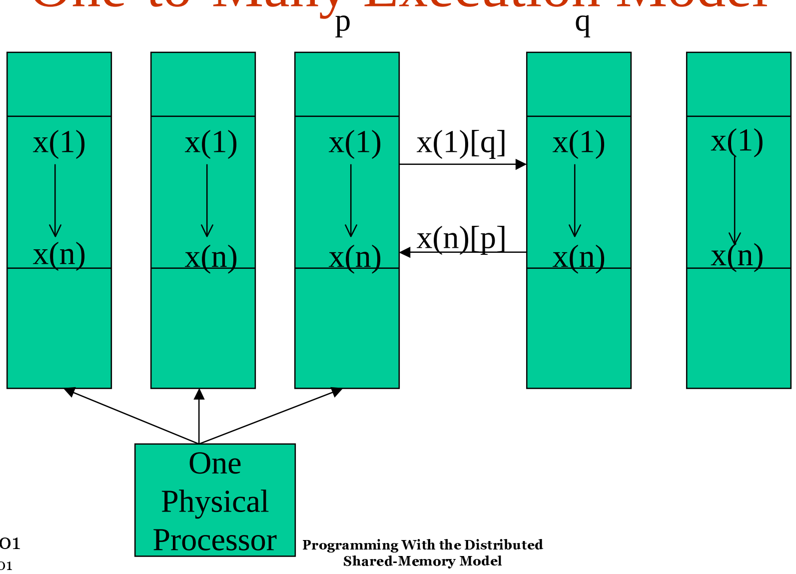
One-to-One Execution Model



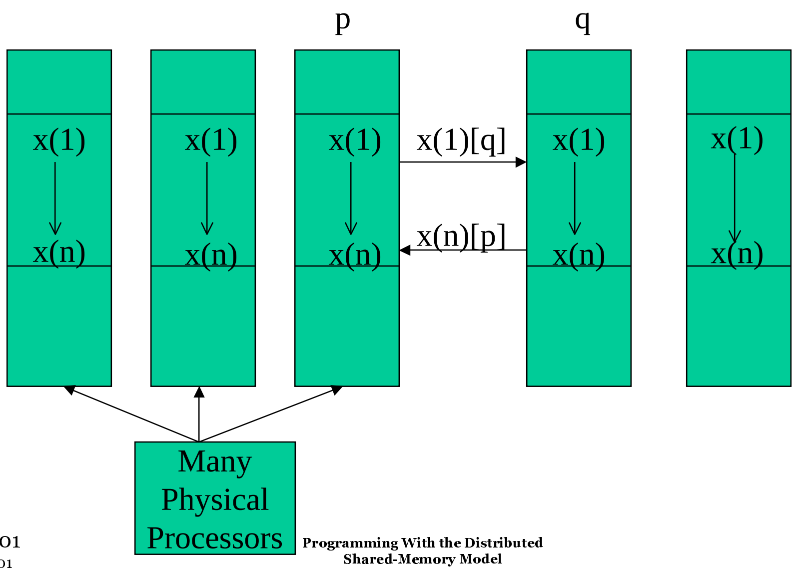
Many-to-One Execution Model



One-to-Many Execution Model



Many-to-Many Execution Model



---

4. Relative Image Indices

Relative Image Indices

- Runtime system builds a map among images.
- CAF syntax is a *logical* expression of this map.
- Current image index:

  1 <= this_image() <= num_images()
- Current image index relative to a co-array**:**

  lowCoBnd(x) <= this_image(x) <= upCoBnd(x)

Relative Image Indices (1)

	1	2	3	4
1	1	5	9	13
2	2	6	10	14
3	3	7	11	15
4	4	8	12	16

x[4,*] this_image() = 15 this_image(x) = (/3,4/)

Relative Image Indices (II)

	0	1	2	3
0	1	5	9	13
1	2	6	10	14
2	3	7	11	15
3	4	8	12	16

x[0:3,0:*] this_image() = 15 this_image(x) = (/2,3/)

Relative Image Indices (III)

	0	1	2	3
-5	1	5	9	13
-4	2	6	10	14
-3	3	7	11	15
-2	4	8	12	16

x[-5:-2,0:*] this_image() = 15 this_image(x) = (/-3, 3/)

Relative Image Indices (IV)

	0	1	2	3	4	5	6	7
0	1	3	5	7	9	11	13	15
1	2	4	6	8	10	12	14	16

x[0:1,0:*]   this_image() = 15   this_image(x) =(/0,7/)

---

5. 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(myPartner)

Synchronize with one other image.

Events

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

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

Example: Global Reduction

```
subroutine glb_dsum(x,n)
real(kind=8),dimension(n)[0:*] :: x
real(kind=8),dimension(n)   :: wrk
integer n,bit,i, mypartner,dim, me, m
dim = log2_images()
if(dim .eq. 0) return
m = 2**dim
bit = 1
me = this_image(x)
do i=1,dim
  mypartner=xor(me,bit)
  bit=shiftl(bit,1)
  call sync_all()
  wrk(:) = x(:)[mypartner]
  call sync_all()
  x(:)=x(:)+wrk(:)
enddo
end subroutine glb_dsum
```

Exercise: Global Reduction

- Convince yourself that two sync points are required.
- How would you modify the routine to handle non-power-of-two number of images?
- Can you rewrite the example using only one barrier?