Introduction to Co-Array Fortran

Robert W. Numrich
Minnesota Supercomputing Institute
University of Minnesota, Minneapolis
and
Goddard Space Flight Center
Greenbelt, Maryland
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
What is Co-Array Syntax?

• Co-Array syntax is a simple parallel extension to normal Fortran syntax.
  – 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 [ ].
Declaration of a Co-Array

real :: x(n)[*]
CAF Memory Model
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.
Synchronization Intrinsic Procedures

\texttt{sync\_all()}
  Full barrier; wait for all images before continuing.

\texttt{sync\_all(wait(:))}
  Partial barrier; wait only for those images in the wait(:) list.

\texttt{sync\_team(list(:))}
  Team barrier; only images in list(:) are involved.

\texttt{sync\_team(list(:),wait(:))}
  Team barrier; wait only for those images in the wait(:) list.

\texttt{sync\_team(myPartner)}
  Synchronize with one other image.
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), zxcvbxcvballocaatable :: maxwell[::]
Communication Using CAF Syntax

\[ y(\cdot) = x(\cdot)[p] \]

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

\[ x(\cdot)[q] = x(\cdot) + x(\cdot)[p] \]

Absent co-dimension defaults to the local object.
Problem Decomposition and Co-Dimensions

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>[p-1,q]</td>
<td>[p,q]</td>
</tr>
<tr>
<td></td>
<td>[p,q+1]</td>
<td>[p+1,q]</td>
</tr>
<tr>
<td>S</td>
<td>[p,q-1]</td>
<td></td>
</tr>
</tbody>
</table>

10
What Do Co-Dimensions Mean?

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

- Replicate an array of length n, one on each image.
- Build a map so each image knows how to find the array on any other image.
- Organize images in a logical (not physical) three-dimensional grid.
- The last co-dimension acts like an assumed size array:  * ⇒ num_images()/(pxq)
Relative Image Indices (1)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>6</td>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>7</td>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
</tr>
</tbody>
</table>

\[ x[4,*] \quad \text{this}_\text{image}() = 15 \quad \text{this}_\text{image}(x) = (/3,4/) \]
Relative Image Indices (II)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>13</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>6</td>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>7</td>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
</tr>
</tbody>
</table>

\[ x[0:3,0:*] \quad \text{this\_image()} = 15 \quad \text{this\_image(x)} = (/2,3/) \]
### Relative Image Indices (III)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>-5</strong></td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>13</td>
</tr>
<tr>
<td><strong>-4</strong></td>
<td>2</td>
<td>6</td>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td><strong>-3</strong></td>
<td>3</td>
<td>7</td>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td><strong>-2</strong></td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
</tr>
</tbody>
</table>

\[ x[-5:-2,0:*] \text{this\_image}() = 15 \]
\[ \text{this\_image}(x) = (-3, 3/) \]
Relative Image Indices (IV)

\[
x[0:1,0:*] \quad \text{this\_image()} = 15 \quad \text{this\_image}(x) = (/0,7/)
\]
Matrix Multiplication

\[
\text{myP} \times \text{myQ} = \text{myP} \times \text{myQ}
\]
Matrix Multiplication

\[
\text{real, dimension}(n, n)[p, *] :: a, b, c
\]

do k = 1, n
  do q = 1, p
    \[
    c(i, j)[\text{myP, myQ}] = c(i, j)[\text{myP, myQ}] + a(i, k)[\text{myP, q}] \times b(k, j)[q, \text{myQ}]
    \]
  enddo
enddo
Matrix Multiplication

real,dimension(n,n)[p,*] :: a,b,c

do k=1,n
   do q=1,p
      c(i,j) = c(i,j) + a(i,k)[myP, q]*b(k,j)[q,myQ]
   enddo
endo
endo
Block Matrix Multiplication
Using “Object-Oriented” Techniques with Co-Array Fortran

- Fortran 95 is not an object-oriented language.
- But it contains some features that can be used to emulate object-oriented programming methods.
  - Allocate/deallocate for dynamic memory management
  - Named derived types are similar to classes without methods.
  - Modules can be used to associate methods loosely with objects.
  - Constructors and destructors can be defined to encapsulate parallel data structures.
  - Generic interfaces can be used to overload procedures based on the named types of the actual arguments.
Cyclic-Wrap Distribution

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>4</th>
<th>7</th>
<th>2</th>
<th>5</th>
<th>3</th>
<th>6</th>
</tr>
</thead>
</table>

|   | 1 | 4 | 7 |
|---|---|---|

|   | 2 | 5 |
|---|---|

|   | 3 | 6 |
Irregular and Changing Data Structures

```
z%ptr
  
  u
  
  z[p,q]%ptr
```

```
  z%ptr
  
  u
  ```
Ocean Objects

```fortran
  type Ocean
    type(ObjectMap) :: rowMap
    type(ObjectMap) :: colMap
    type(Cell), allocatable :: cells(:, :)
  end type Ocean

  type Cell
    type(Fish) :: fish
    type(Shark) :: shark
  end type Cell
```
type(Ocean), allocatable :: atlantic[::,::]
coDim(1:2) = factor_num_images(2)
allocate(atlantic[coDim(1),*])
call newOcean(atlantic,rowCells,colCells)
do t=1,nIter
    call sync_all()
    call swimFishes(atlantic)
    call sync_all()
    call swimSharks(atlantic)
endo
Summary

• Co-dimensions match your logical problem decomposition
  – Run-time system matches them to hardware decomposition
  – Explicit representation of neighbor relationships
  – Flexible communication patterns
• Code simplicity
  – Non-intrusive code conversion
  – Modernize code to Fortran 95 standard
• Code is always simpler and performance is always better than MPI.
The Co-Array Fortran Standard

• Co-Array Fortran is defined by:

• Additional information on the web:
  – www.co-array.org
  – 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
Vector Objects

type vector
  real, allocatable :: vector(:)
  integer :: lowerBound
  integer :: upperBound
  integer :: halo
end type type vector
Block Vectors

type BlockVector

  type(VectorMap) :: map

  type(Vector), allocatable :: block(:)

  --other components--

end type type BlockVector
Block Matrices

type BlockMatrix
  type(VectorMap) :: rowMap
  type(VectorMap) :: colMap
  type(Matrix), allocatable :: block(:, :)
  --other components--
end type type BlockMatrix
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)
5. Where Can I Try CAF?
Co-Array Fortran on Other Platforms

- Rice University is developing an open source compiling system for CAF.
  - Runs on the HP-Alpha system at PSC
  - Runs on SGI platforms
  - We are planning to install it on Halem at GSFC
- IBM may put CAF on the BlueGene/L machine at LLNL.
- DARPA High Productivity Computing Systems (HPCS) Project wants CAF.
  - IBM, CRAY, SUN
Why Language Extensions?

- Programmer uses a familiar language.
- Syntax gives the programmer control and flexibility.
- Compiler concentrates on local code optimization.
- Compiler evolves as the hardware evolves.
  - Lowest latency and highest bandwidth allowed by the hardware
  - Data ends up in registers or cache not in memory
  - Arbitrary communication patterns
  - Communication along multiple channels
The Guiding Principle

• 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?
• How can it be expressed so that existing compiler technology can implement it easily and efficiently?
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
One-to-One Execution Model
Many-to-One Execution Model

Many Physical Processors
One-to-Many Execution Model

One Physical Processor
Many-to-Many Execution Model

Many Physical Processors
Exercise 1: Global Reduction

subroutine globalSum(x)
real(kind=8),dimension[0:*] :: x
real(kind=8) :: work
integer n,bit,i, mypal,dim,me, m
dim = log2_images()
if(dim .eq. 0) return
m = 2**dim
bit = 1
me = this_image(x)
do i=1,dim
doit=1,dim
    mypal=xor(me,bit)
    bit=shf1(bit,1)
    call sync_all()
    work = x[mypal]
    call sync_all()
    x=x+work
endo
end subroutine globalSum
Events

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

sync_team(list(:),list(you:you))    wait event
Other CAF Intrinsic Procedures

sync_memory()
   Make co-arrays visible to all images

csync_file(unit)
   Make local I/O operations visible to the global file system.

start_critical()

end_critical()
   Allow only one image at a time into a protected region.
Other CAF Intrinsic Procedures

\textbf{log2\_images()}

Log base 2 of the greatest power of two less than or equal to the value of \texttt{num\_images()}

\textbf{rem\_images()}

The difference between \texttt{num\_images()} and the nearest power-of-two.
Block Matrix Multiplication

Figure 4: Time as a function of the number of processors $p = q \times r$ for block matrix multiplication. The matrix size is $1000 \times 1000$ with blocks of size $1000/q \times 1000/r$. Time is expressed in dimensionless giga-clock-ticks, $\nu t \times 10^{-9}$, as measured on a CRAY-T3E with frequency $\nu = 300$MHz. The dotted line represents perfect scaling.
2. An Example from the UK Met Unified Model
Incremental Conversion to Co-Array Fortran

- Fields are allocated on the local heap
- One processor knows nothing about another processor’s memory structure
- But each processor knows how to find co-arrays in another processor’s memory
- Define one supplemental co-array structure
- Create an alias for the local field through the co-array field
- Communicate through the alias
CAF Alias to Local Fields

- real :: u(0:m+1,0:n+1,lev)
- type(field) :: z[p,*]
- z%ptr => u
- u = z[p,q]%ptr
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)

<table>
<thead>
<tr>
<th>PxQ</th>
<th>SHMEM</th>
<th>SHMEM w/CAF SWAP</th>
<th>MPI w/CAF SWAP</th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>2x2</td>
<td>191</td>
<td>198</td>
<td>201</td>
<td>205</td>
</tr>
<tr>
<td>2x4</td>
<td>95.0</td>
<td>99.0</td>
<td>100</td>
<td>105</td>
</tr>
<tr>
<td>2x8</td>
<td>49.8</td>
<td>52.2</td>
<td>52.7</td>
<td>55.5</td>
</tr>
<tr>
<td>4x4</td>
<td>50.0</td>
<td>53.7</td>
<td>54.4</td>
<td>55.9</td>
</tr>
<tr>
<td>4x8</td>
<td>27.3</td>
<td>29.8</td>
<td>31.6</td>
<td>32.4</td>
</tr>
</tbody>
</table>
3. CAF and “Object-Oriented” Programming Methodology
A Parallel “Class Library” for CAF

• 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 (●) is code written in Co-Array Fortran. The curve marked with triangles (▽) is SCALAPACK code.
Communication for LU Decomposition

• Row interchange
  – temp(,:) = a(k,:)
  – a(k,:) = a(j,:) [p,myQ]
  – a(j,:) [p,myQ] = temp(:)

• Row “Broadcast”
  – L0(i:n,i) = a(i:,n,i) [p,p] i=1,n

• Row/Column “Broadcast”
  – L1 (:,:) = a(:,:)[myP,p]
  – U1(:,:) = a(:,:) [p,myQ]
6. Summary