Co-Array Fortran
What is it? Why should you put it on BlueGene/L?

Robert W. Numrich

Minnesota Supercomputing Institute
University of Minnesota

rwn@msi.umn.edu
The Guiding Principle behind Co-Array Fortran

- 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?
What’s the Problem with SPMD?

• One processor knows nothing about another’s memory layout.
  – Local variables live on the local heap.
  – Addresses, sizes and shapes are different on different program images.

• How can we exchange data between such non-aligned variables?
Co-Array Fortran Extension

• Incorporate the SPMD Model into Fortran 90
  – Multiple images of the same program
  – Text and data are replicated in each image
• Mark some variables with co-dimensions
  – Co-dimensions behave like normal dimensions
  – Co-dimensions express a logical problem decomposition
  – One-sided data exchange between co-arrays using a Fortran-like syntax
• Require the underlying run-time system to map the logical problem decomposition onto specific hardware.
The CAF Execution Model

- The number of images is fixed and each image has its own index, retrievable at run-time:
  
  \[ 1 \leq \text{num}_\text{images}() \]
  
  \[ 1 \leq \text{this}_\text{image}() \leq \text{num}_\text{images}() \]

- 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 CAF syntax.
What is Co-Array Syntax?

• Co-Array syntax is a simple 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 [ ].
Examples of Co-Array Declarations

real :: s[*]
real :: a(n)[*]
complex :: z[*]
integer :: index(n)[*]
real :: b(n)[p, []]
real :: c(n,m)[0:p, -7:q, 11:[]]
real, allocatable :: w(:)[::]
type(field) :: maxwell[p,[]]
CAF Memory Model

University of Minnesota
One-to-One Execution Model

One Physical Processor
Many-to-One Execution Model

Many Physical Processors

\[
\begin{align*}
&x(1) \\
&\downarrow \\
&x(n)
&x(1) \\
&\downarrow \\
&x(n)
&x(1) \\
&\downarrow \\
&x(n)
&x(1) \\
&\downarrow \\
&x(n)
&x(1) \\
&\downarrow \\
&x(n)
\end{align*}
\]

\[x(1)[q]\]
\[x(n)[p]\]
One-to-Many Execution Model

One Physical Processor
Many-to-Many Execution Model

Many Physical Processors
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: \( \text{num\_images()}/(pxq) \)
- A specific implementation could choose to represent memory hierarchy through the co-dimensions.
<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>

\[
x[4,*] \quad \text{this}_\text{image}() = 15 \quad \text{this}_\text{image}(x) = (3,4)\]

University of Minnesota
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>-5</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>13</td>
</tr>
<tr>
<td>-4</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>-2</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
</tr>
</tbody>
</table>

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

University of Minnesota
Relative Image Indices (IV)

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

\[x[0:1,0:]*\]  \[\text{this\_image()} = 15\]  \[\text{this\_image}(x) = (\langle 0,7 \rangle)\]
Communication Using CAF Syntax

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

\[ \text{myIndex}(:) = \text{index}(:) \]
\[ \text{yourIndex}(:) = \text{index}(:)[\text{you}] \]

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

\[ x(:,[q]) = x(:) + x(:)[p] \]

Absent co-dimension defaults to the local object.
Irregular and Changing Data Structures
Matrix Multiplication

real, dimension(n, n)[p, q] :: a, b, c
(/myP, myQ/) = this_image(c)
Matrix Multiplication

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

do k=1,n
  do q=1,p
    c(i,j)[myP,myQ] = c(i,j)[myP,myQ] + a(i,k)[myP, q]*b(k,j)[q,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
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.
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”
  – \( L1 (\cdot,\cdot) = a(\cdot,\cdot) \ [\text{myP},p] \)
  – \( U1(\cdot,\cdot) = a(\cdot,\cdot) \ [p,\text{myQ}] \)
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.
A Parallel “Class Library” for CAF

• Combine the object-based features of Fortran 90 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.

• Based on Vector Maps designed to support redistribution of data for load balancing, adaptive mesh refinement, etc.
Run-time System Support for CAF

• Compiler decodes CAF syntax and determines the processor (thread, process, node) where the data lives
• Compiler hands this information to a communication protocol
  – Global virtual address space: use load/store instructions
  • Higher-order bits in address: remote = local + shift(p)
  • Virtual offset: remote = local + offset(p)
  • Table lookup: remote = remote(p)
  – Implement on one BG/L compute node as proof-of-concept?
  – Interface to a one-sided communication library
    • Armci, Shmem, Lapi, Quadrics elan, Myrinet GM-2, MPI-2, Active messages
• Dynamic memory management for co-arrays
• Fast barriers
• Cache coherence (invalidate on sync?)
• Optimal logical to physical mapping (simulated annealing?)
The Co-Array Fortran Standard

• Co-Array Fortran is defined by:

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