Coarray Fortran (CAF)

- Global address space SPMD parallel programming model
  — one-sided communication
- Simple, two-level memory model for locality management
  — local vs. remote memory
- Programmer has control over performance critical decisions
  — data partitioning
  — data movement
  — synchronization
- Adopted in Fortran 2008 standard
Coarray Fortran 2.0 Goals

- Exploit multicore processors
- Enable development of portable high-performance programs
- Interoperate with legacy models such as MPI
- Facilitate construction of sophisticated parallel applications and parallel libraries
- Support irregular and adaptive applications
- Hide communication latency
- Colocate computation with remote data
- Scale to exascale
Coarray Fortran 2.0 (CAF 2.0)

• Teams: process subsets, like MPI communicators
  —formation using team_split
  —collective communication (two-sided)
  —barrier synchronization

• Coarrays: shared data allocated across processor subsets
  —declaration: double precision :: a(:, :)[*]
  —dynamic allocation: allocate(a(n, m)[@row_team])
  —access: x(:, n+1) = x(:, 0)[mod(team_rank() + 1, team_size())]

• Latency tolerance
  —hide: asynchronous copy, asynchronous collectives
  —avoid: function shipping

• Synchronization
  —event variables: point-to-point sync; async completion
  —finish: SPMD construct inspired by X10

• Copointers: pointers to remote data
Process Subsets: Teams

- Teams are first-class entities
  - ordered sequences of process images
  - namespace for indexing images by rank $r$ in team $t$
    - $r \in \{0..\text{team}_\text{size}(t) - 1\}$
  - domain for allocating coarrays
  - substrate for collective communication

- Teams need not be disjoint
  - an image may be in multiple teams

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

<table>
<thead>
<tr>
<th>Atmosphere</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>
Teams and Operations

• Predefined teams
  —团队_world
  —团队_default
    – 用于任何协程操作，缺乏显式团队指定
    – 使用WITH TEAM / END WITH TEAM动态分隔，块结构化

• Operations on teams
  —team_rank(team)
    – 返回当前图像在团队中的0-based相对排名
  —team_size(team)
    – 返回给定团队的图像数量
  —team_split (existing_team, color, key, new_team)
    – 供应相同颜色的图像分配到同一个团队
    – 每个图像在其新团队中的排名由（key, 父团队排名）的字典序决定
• Designed for scalability: representation is $O(\log S)$ per node for a team of size $s$

• Based on the concept of pointer jumping

• Pointers to predecessors and successors at distance $i = 2^j$, $j = 0 .. \lfloor \log S \rfloor$
Collective Example: Barrier

Dissemination algorithm

for $k = 0$ to $\lceil \log_2 P \rceil$

processor $i$ signals processor $(i + 2^k) \mod P$ with a PUT
processor $i$ waits for signal from $(i - 2^k) \mod P$
Collective Example: Broadcast

Binomial Tree

round 0  \[2^0\]  
round 1  \[2^1\]  
round 2  \[2^2\]  

[Diagram of a binomial tree showing data flow through rounds 0, 1, and 2.]
Accessing Coarrays on Teams

- Accessing a coarray relative to a team
  \[ x(i,j)[p@ocean] \quad \text{! } p \text{ names a rank in team ocean} \]

- Accessing a coarray relative to the default team
  \[ x(i,j)[p] \quad \text{! } p \text{ names a rank in team_default} \]
  \[ x(i,j)[p@team_default] \quad \text{! } p \text{ names a rank in team_default} \]

- Simplifying processor indexing using “with team”
  \[
  \text{with team atmosphere} \quad \text{! set team_default to atmosphere within}
  \quad \text{! } p \text{ is wrt team atmosphere, } q \text{ is wrt team ocean}
  \]
  \[ x(:,0)[p] = y(:)[q@ocean] \]
  \text{end with team}
Rich Set of Collectives

- TEAM_ALLGATHER()
- TEAM_ALLREDUCE()
- TEAM_ALLTOALL()
- TEAM_BARRIER()
- TEAM_BROADCAST()
- TEAM_GATHER()
- TEAM_SCAN()
- TEAM_SCATTER()
- TEAM_SHIFT()
- User-defined reductions

✓ Generally, should consider MPI 3.0 set
✓ Optional team argument uses TEAM_DEFAULT if not specified
✓ Compiler calculates sizes of buffers to simplify param lists
Redundancies

- **NUM.Images**
  — same as `TEAM_SIZE(TEAM_WORLD)`

- **SYNC TEAM, SYNC ALL**
  — both supplanted by `TEAM_BARRIER()`
Events

- First-class event variables
  - support safe synchronization space
- Uses
  - point-to-point synchronization
  - signal the readiness or completion of asynchronous operations
Coping with Latency

• Asynchronous operations for latency tolerance
  — predicated asynchronous copy
  — collectives
  — split-phase synchronization
    – barriers
    – events

• Function shipping for latency avoidance
  — co-locate data with computation
Predicated Asynchronous Copy

• Issue
  —want communication/computation overlap like MPI_Isend/MPI_Irecv for a one sided model

• Approach: predicated asynchronous copy

• Unified synchronization through events
  —when copy may begin
  —when source data may be overwritten
  —when destination data may be read

• COPY_ASYNC(var_dest, var_src [,ev_dr][,ev_cr][,ev_sr])
  —ev_dr = destination ready (write complete)
  —ev_cr = copy ready (copy may start)
  —ev_sr = source ready (source safe to overwrite)
Asynchronous Collectives

- Interface is same as proposed synchronous collectives
  — one extra parameter: completion event
- Upon completion of collective, signal the supplied event
- Note: asynchronous barrier is the same as a split-phase barrier
- Unified synchronization through events
Copointers

• **Pointers to remote coarray sections or remote shared data**

```fortran
integer, dimension(:), copointer :: p1, p2  ! copointer to array of integer
integer, dimension(10), cotarget :: a1[*]   ! coarray of array of integer

...  
p1 => a1       ! copointer to a1’s local coarray section  
p2 => a1[9]    ! copointer to a1 on image 9    
p1(6) = 1      ! assigns sixth element of local section of a1  
p2(6)[] = 42    ! assigns sixth element of a1 on image 9
```

**Cotarget** = allocated in shared space

• **Accesses to remote data explicitly use [ ]**
  —conforms to spirit of coarray Fortran extensions  
  —visual cues to mark remote operations
Other Features

• Atomic operations
  —CAS, ADD, AND, OR, XOR, FADD, FAND, FOR, FXOR

• Team-based storage allocation

• Topologies: cartesian, graph

• Inter-team communication and coupling
  —setup and utilization
  —synchronization
    – m X n collectives
  —one-sided access to extra-team data
    – normal coarray-style access
    – m-gather-from-n
      interpolated or variable-sized results for doing own

• Function shipping
  —call spawn

• Finish
HPC Challenge Benchmarks

- Priorities, in order
  - performance, performance, performance
  - source code volume
- Productivity = performance / (lines of code)
- Implementation sketch
  - FFT
    - use global transposes to keep computation local
  - EP STREAM Triad
    - outline a loop for best compiler optimization
  - Randomaccess
    - batch updates and use software routing for higher performance
  - HPL
    - operate on blocks to leverage a high performance DGEMM
  - Unbalanced Tree Search (UTS)
    - evaluate how CAF 2.0 supports dynamic load balancing
    - use function shipping to implement work stealing and work sharing
• Radix 2 FFT implementation

• Block distribution of coarray “c” across all processors

• Sketch in CAF 2.0:

```fortran
complex, allocatable :: c(:,2)[*], spare(:)[*]
```

```
! permute data to bit-reversed indices (uses team_alltoall)
call bitreverse(c, n_world_size, world_size, spare)
```

```
! local FFT computation for levels that fit in the memory of an image
do l = 1, loc_comm-1 ...
```

```
! transpose from block to cyclic data distribution (uses team_alltoall)
call transpose(c, n_world_size, world_size, spare)
```

```
! local FFT computation for remaining levels
do l = loc_comm, levels ...
```

```
! transpose back from cyclic to block data distribution (uses team_alltoall)
call transpose(c, n_world_size, n_local_size/world_size, spare)
```
double precision, allocatable :: a(:)[*], b(:)[*], c(:)[*]

! each processor in the default team allocates their own array parts
allocate(a(local_n)[], b(local_n)[], c(local_n)[])

! perform the calculation repeatedly to get reliable timings
do round = 1, rounds
  do j = 1, rep
    call triad(a,b,c,local_n,scalar)
  end do
  call team_barrier() ! synchronous barrier across images in the default team
end do

! perform the calculation with top performance
! assembly code is identical to that for sequential Fortran

subroutine triad(a, b, c, n, scalar)
  double precision :: a(n), b(n), c(n), scalar
  a = b + scalar * c ! EP triad as a Fortran 90 vector operation
end subroutine triad
Randomaccess Software Routing

```fortran
! hypercube-based routing: each processor has 1024 updates
do i = world_logsize-1, 0, -1  ! log P stages in a route
  call split(retain(:,last), ret_sizes(last), &
            retain(:,current), ret_sizes(current), &
            fwd(1:,in,i), fwd(0,in,i), bufsize, dist)
  if (i < world_logsize-1) then
    event_wait(delivered(i+1))
    call split(fwd(1:,in,i+1), fwd(0,in,i+1), &
               retain(:,current), ret_sizes(current), &
               fwd(1:,out,i), fwd(0,out,i), bufsize, dist)
    event_notify(received(i+1)[from])  ! signal buffer is empty
  endif
  count = fwd(0,out,i)
  event_wait(received(i))  ! ensure buffer is empty from last route
  fwd(0:count,in,i)[partner] = fwd(0:count,out,i)  ! send to partner
  event_notify(delivered(i)[partner])  ! notify partner data is there
end do
```
Experimental Setup

• Rice Coarray Fortran 2.0
  — source to source translation from CAF 2.0 to Fortran 90
    • generated code compiled with Portland Group’s pgf90
  — CAF 2.0 runtime system built upon GASNet (versions 1.14 .. 1.17)
  — scalable implementation of teams, using $O(\log P)$ storage

• Experimental platforms: Cray XT4, XT5, and XE6
  — systems
    • Franklin - XT4 at NERSC
      2.3 GHz AMD “Budapest” quad-core Opteron, 2GB DDR2-800/core
    • Jaguar - XT4 at ORNL
      2.1 GHz AMD quad-core Opteron, 2GB DDR2-800/core
    • Jaguar - XT5 at ORNL
      2.6 GHz AMD “Istanbul” hex-core Opteron, 1.3GB DDR2-800/core
    • Hopper - XE6 at NERSC
      2.1 GHz AMD dual-twelve cores Magnycours, 1.3GB DDR3-1333/core
  — network topologies
    • XT4, XT5: 3D Torus based on Seastar2 routers; XE6: Gemini
Unbalanced Tree Search (UTS)

- Exploration of an unbalanced implicit tree
- Fixed geometric distribution, depth 18, 270 billion nodes

```fortran
! while there is work to do
do while(queue_count .gt. 0)
  call dequeue_back(descriptor)
  call process_work(descriptor)
  ...
  ! check if someone needs work
  if ((incoming_lifelines .ne. 0) .and. &
      (queue_count .ge. lifeline_threshold)) then
    call push_work()
  endif
enddo

! attempt to steal work from another image
victim = get_random_image()

spawn steal_work()[victim]

! set up lifelines on hypercube neighbors
do index = 0, max_neighbor_index-1
  neighbor = xor(my_rank, 2**index)
  spawn set_lifelines(my_rank, index)[neighbor]
enddo
```

- Slope shows all PE working
- Tight grouping of lines shows good load balance

Separate line for each of 128 PEs

Cray XT5, 12 cores/node
HPL

- Block-cyclic data distribution
- Team based collective operations along rows and columns
  — synchronous max reduction down columns of processors
  — asynchronous broadcast of panels to all processors

```fortran
module paneltype
  type(paneltype) :: panels(1:NUMPANELS)
  event, allocatable :: delivered(:)[*]
end module

pp = 0
PROBLEMSIZE = ...
BLKSIZE = ...
do j = pp, PROBLEMSIZE - 1, BLKSIZE
  cp = mod(j / BLKSIZE, 2) + 1
  event_wait(delivered(3-cp))
  if (mycol == cproc) then
    if (ncol > 0) then ! update part of the trailing matrix
      call fact(m, n, cp) ! factor the next panel
    endif
    call team_broadcast_async(panels(cp)%buff(1:ub), panels(cp)%info(8), &
                               delivered(cp)) ! update rest of the trailing matrix
    if (nn-ncol>0) call update(m, n, col, nn-ncol, 3 - cp)
  endif
end do
```
Productivity = Performance / SLOC

Performance (on Cray XT4 and XT5)

<table>
<thead>
<tr>
<th># of cores</th>
<th>STREAM Triad* (TByte/s)</th>
<th>RandomAccess‡(GUP/s)</th>
<th>Global HPL † (TFlop/s)</th>
<th>Global FFT † (GFlop/s)</th>
<th>UTS* (MNode/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>0.17</td>
<td>0.08</td>
<td>0.36</td>
<td>6.69</td>
<td>163.1</td>
</tr>
<tr>
<td>256</td>
<td>0.67</td>
<td>0.24</td>
<td>1.36</td>
<td>22.82</td>
<td>645.0</td>
</tr>
<tr>
<td>1024</td>
<td>2.66</td>
<td>0.69</td>
<td>4.99</td>
<td>67.80</td>
<td>2371</td>
</tr>
<tr>
<td>4096</td>
<td>10.70</td>
<td>2.01</td>
<td>18.3</td>
<td>187.04</td>
<td>7818</td>
</tr>
<tr>
<td>8192</td>
<td>21.69</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Jaguar - XT5 ‡Jaguar - XT4 †Franklin - XT4

Source lines of code

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Source Lines</th>
<th>Reference SLOC</th>
<th>Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Randomaccess</td>
<td>409</td>
<td>787</td>
<td>48%</td>
</tr>
<tr>
<td>EP STREAM</td>
<td>63</td>
<td>329</td>
<td>81%</td>
</tr>
<tr>
<td>Global HPL</td>
<td>786</td>
<td>8800</td>
<td>91%</td>
</tr>
<tr>
<td>Global FFT</td>
<td>450</td>
<td>1130</td>
<td>60%</td>
</tr>
<tr>
<td>UTS</td>
<td>544</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>

Notes
- STREAM: 82% of peak memory bandwidth
- HPL: 49% of FP peak at @ 4096 cores (uses dgemm)
Relative Parallel Efficiency

- EP STREAM Triad
- HPL
- UTS
- FFT
- Randomaccess

The graph illustrates the relative parallel efficiency of different applications or benchmarks as a function of the number of cores. The efficiency is measured on a scale from 0 to 1, where 1 represents perfect efficiency. The x-axis represents the number of cores, ranging from 64 to 8192, and the y-axis represents the relative parallel efficiency.
! post a receive
do n=1,in_bndy%nmsg_ew_rcv
    bufsize = ny_block*nghost*in_bndy%nblocks_ew_rcv(n)
    call MPI_IRECV(buf_ew_rcv(1,1,1,n), bufsize, mpi_dbl, &
                    in_bndy%ew_rcv_proc(n)-1, &
                    mpitag_bndy_2d + in_bndy%ew_rcv_proc(n), &
                    in_bndy%communicator, rcv_request(n), ierr)
end do

! pack data and send data
do n=1,in_bndy%nmsg_ew_snd
    bufsize = ny_block*nghost*in_bndy%nblocks_ew_snd(n)
    partner = in_bndy%ew_snd_proc(n)-1
    do i=1,in_bndy%nblocks_ew_snd(n)
        ib_src    = in_bndy%ew_src_add(1,i,n)
        ie_src    = ib_src + nghost - 1
        src_block = in_bndy%ew_src_block(i,n)
        buf_ew_snd(:,:,i,n) = ARRAY(ib_src:ie_src,:,src_block)
    end do
    call MPI_ISEND(buf_ew_snd(1,1,1,n), bufsize, mpi_dbl, &
                   in_bndy%ew_snd_proc(n)-1, &
                   mpitag_bndy_2d + my_task + 1, &
                   in_bndy%communicator, snd_request(n), ierr)
end do

! local updates
! wait to receive data and unpack data
call MPI_WAITALL(in_bndy%nmsg_ew_rcv, rcv_request, rcv_status, ierr)

! wait send to finish
! notify each partner that my face is ready
! when each partner face is ready
!     copy one of my faces to a partner’s face
!     notify my partner’s event when the copy is complete
! wait for all of my incoming faces to arrive

Open Issues - I

• What hierarchical teams features are needed?
  —split for shared memory domain?
  —split for UMA domain?

• What about mapping teams onto whole systems?

![Mapping applications with collectives over sub-communicators on torus networks. Bhatele et al. (LLNL) Proceedings of SC12](image)

Fig. 8: Untilted, once-tilted, and twice-tilted 3D boxes

• What should be the focus of DEGAS CAF compiler efforts?
  —incorporate support for DSL for CA algorithms?
  —generate code for throughput-oriented cores
    – CUDA/OpenCL?
Open Issues - II

• Irregular computation
  —global view?
  —partitioning
  —repartitioning
  —communication/synchronization schedules
  —computation schedules

• Multithreading
  —managing asynchronous communication and progress
  —managing function shipping

• Resilience