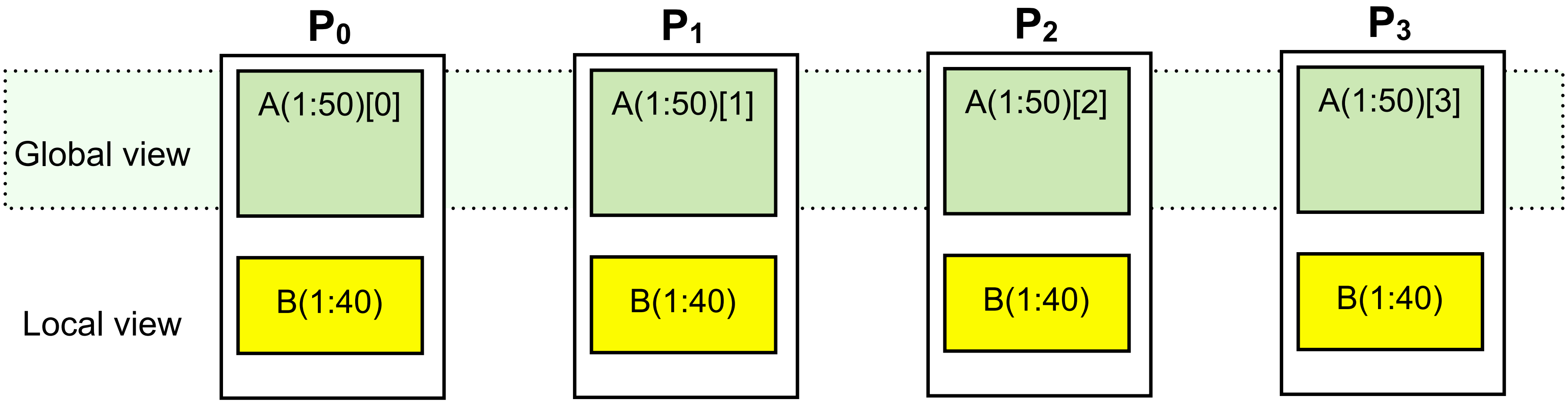
Coarray Fortran 2.0

John Mellor-Crummey, Karthik Murthy,
Dung Nguyen, Sriraj Paul, Scott Warren,
Chaoran Yang

Department of Computer Science
Rice University

DEGAS Retreat 4 June 2013

Coarray Fortran (CAF)


P_0 P_1 P_2 P_3

Global view

A(1:50)[0] A(1:50)[1] A(1:50)[2] A(1:50)[3]

Local view

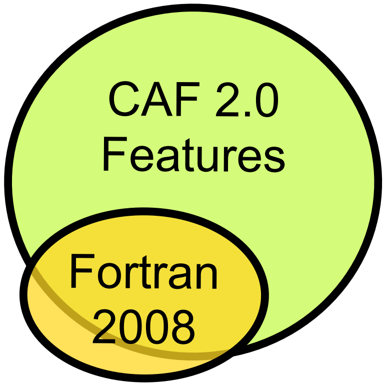
B(1:40) B(1:40) B(1:40) B(1:40)

- **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**

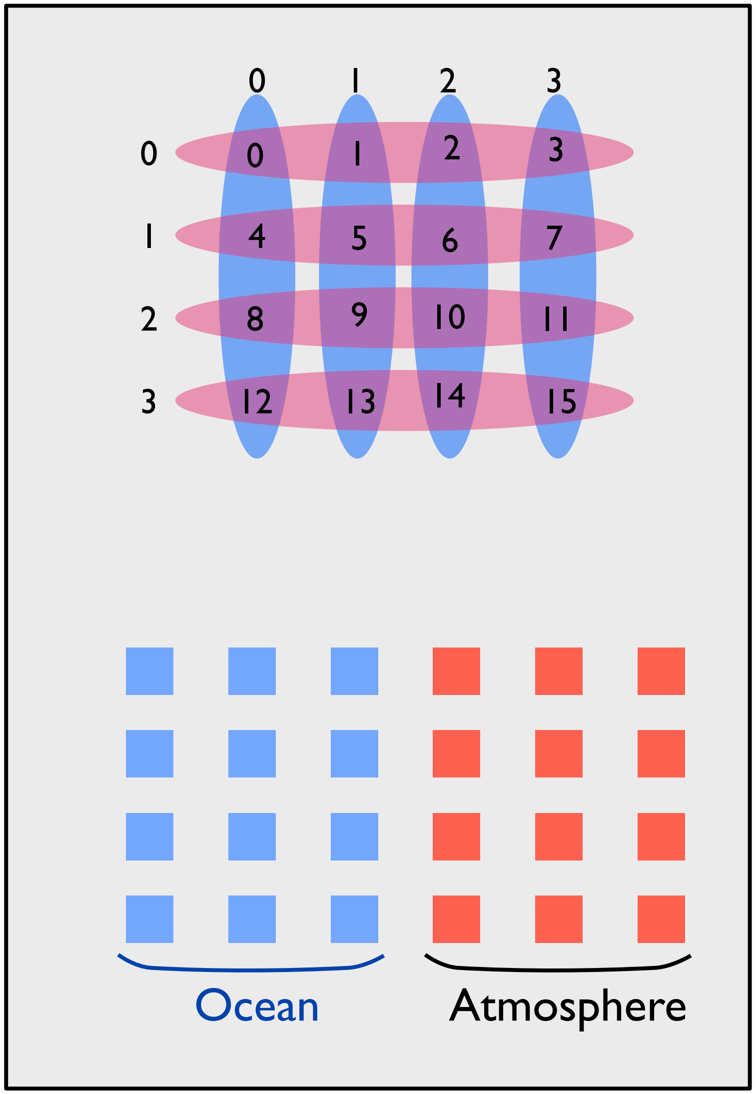
Coarray Fortran 2.0 (CAF 2.0)



- **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 ∈ {0..team_size(t) - 1}
  - —domain for allocating coarrays
  - —substrate for collective communication
- **Teams need not be disjoint**
  - —an image may be in multiple teams

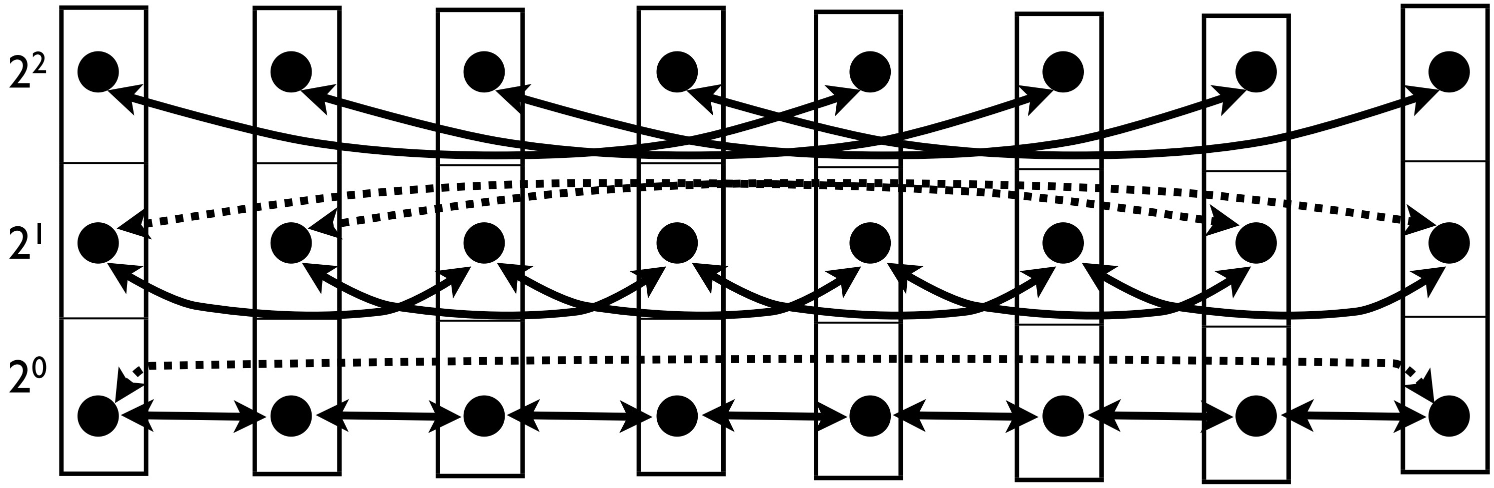


Teams and Operations

- **Predefined teams**
  - **team_world**
  - **team_default**
    - used for any coarray operation that lacks an explicit team specification
    - set via `WITH TEAM` / `END WITH TEAM`

      dynamically scoped, block structured
- **Operations on teams**
  - **team_rank(team)**
    - returns the 0-based relative rank of the current image within a team
  - **team_size(team)**
    - returns the number of images of a given team
  - **team_split (existing_team, color, key, new_team)**
    - images supplying the same color are assigned to the same team
    - each image's rank in the new team is determined by lexicographic order of (key, parent team rank)

CAF 2.0 Team Representation

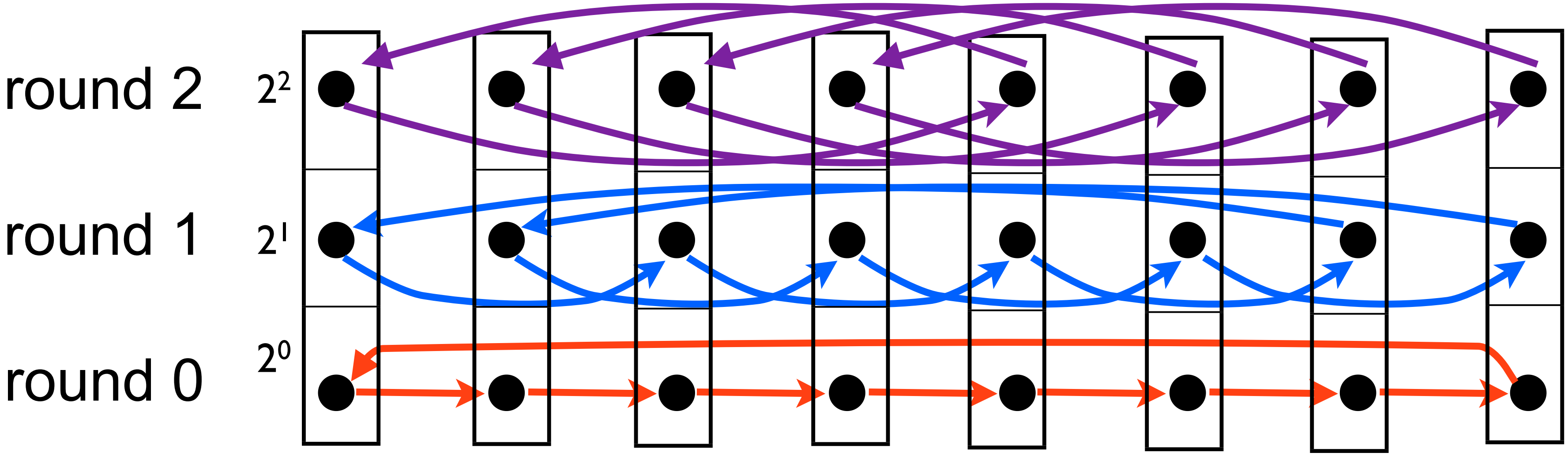


- **Designed for scalability: representation is O(log S) per node for a team of size s**
- **Based on the concept of pointer jumping**
- **Pointers to predecessors and successors at distance i = 2^j, j = 0 .. ⌊log S⌋**

Collective Example: Barrier

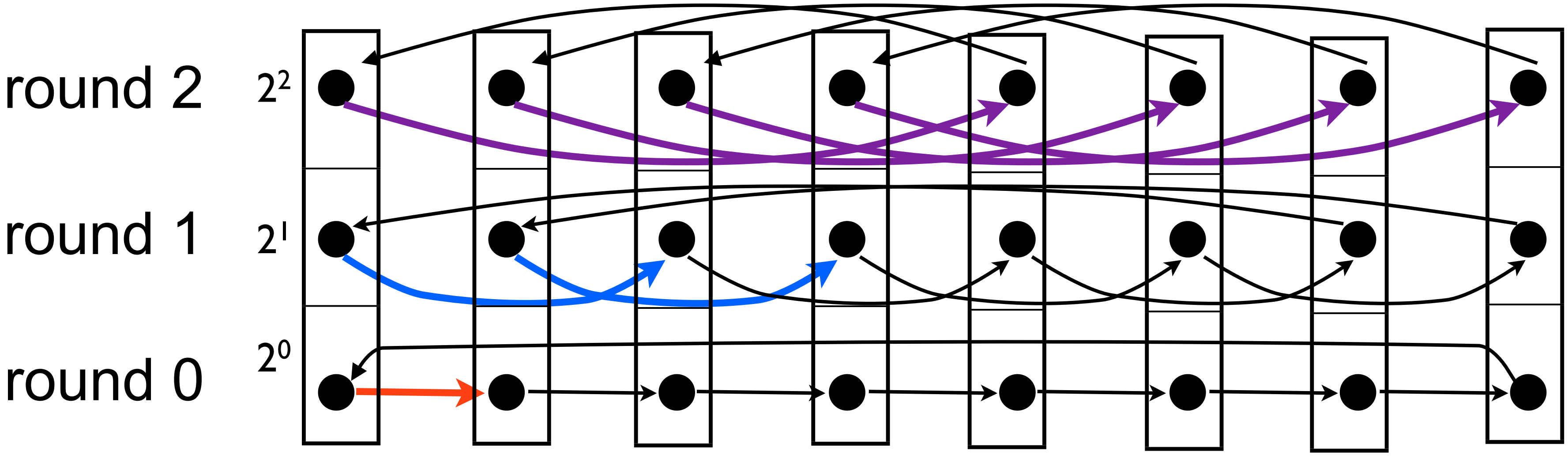
Dissemination algorithm

```
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



Accessing Coarrays on Teams

- **Accessing a coarray relative to a team**
  - x(i,j)[p@ocean] *! p names a rank in team ocean*

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

- **Simplifying processor indexing using "with team"**

```
with team atmosphere ! set team_default to atmosphere within
   ! p is wrt team atmosphere, q is wrt team ocean
   x(:,0)[p] = y(:)[q@ocean]
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**

```
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

FFT

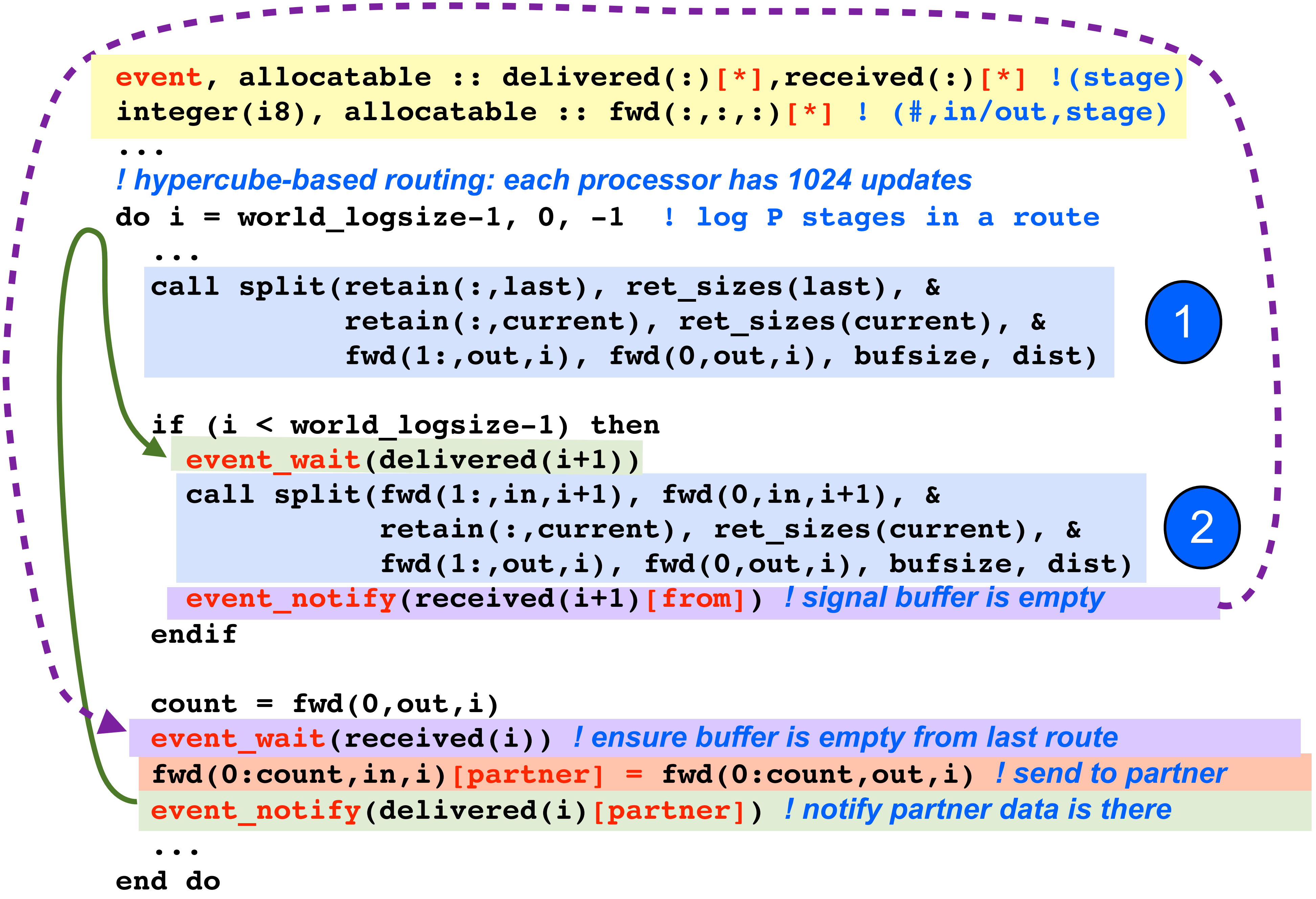
- **Radix 2 FFT implementation**
- **Block distribution of coarray “c” across all processors**
- **Sketch in CAF 2.0:**

```
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)
```

EP STREAM Triad

```
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



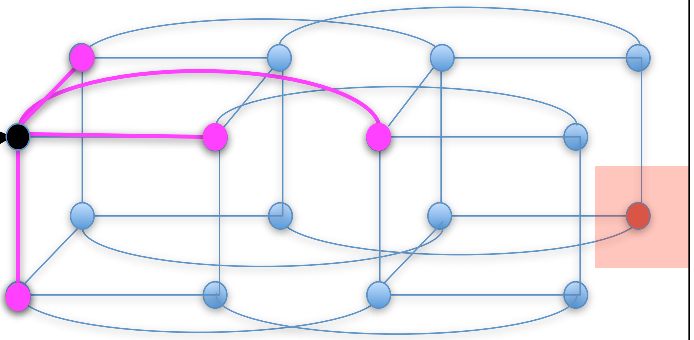
```
event, allocatable :: delivered(:)[*],received(:)[*] !(stage)
integer(i8), allocatable :: fwd(:,:,:)[*] ! (#,in/out,stage)
...
! hypercube-based routing: each processor has 1024 updates
do i = world_logsize-1, 0, -1  ! log P stages in a route
  ...
  call split(retain(:,last), ret_sizes(last), &
             retain(:,current), ret_sizes(current), &
             fwd(1:,out,i), fwd(0,out,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

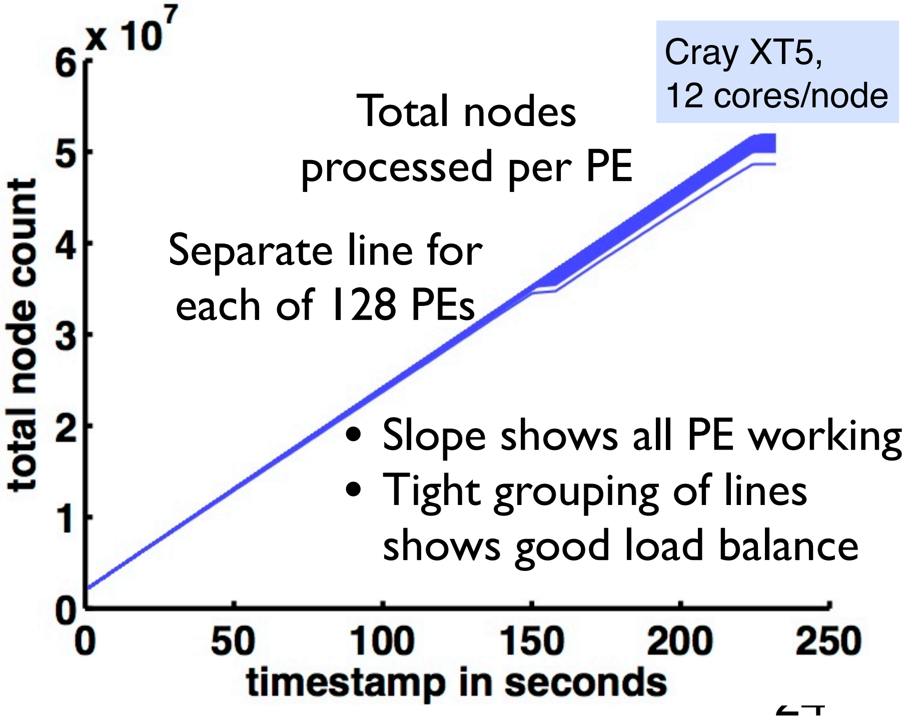
- Rice Coarray Fortran 2.0
    - source to source translation from CAF 2.0 to Fortran 90
        - generated code compiled with Portland Group's pgf90
    - CAF 2.0 runtime system built upon GASNet (versions 1.14 .. 1.17)
    - scalable implementation of teams, using O(log P) storage
- Experimental platforms: 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**



```
! 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
```



Cray XT5, 12 cores/node

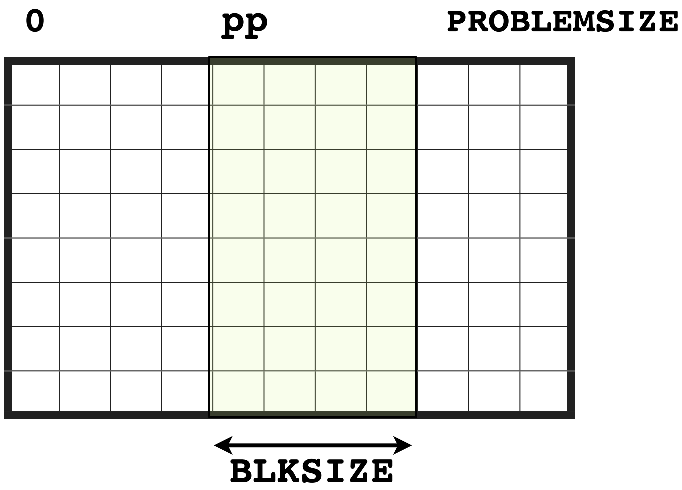
Total nodes processed per PE

Separate line for each of 128 PEs

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

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



```
type(paneltype) :: panels(1:NUMPANELS)
event, allocatable :: delivered(:)[*]
...
do j = pp, PROBLEMSIZE - 1, BLKSIZE
  cp = mod(j / BLKSIZE, 2) + 1
  ...
  event_wait(delivered(3-cp))
  ...
  if (mycol == cproc) then
    ...
    if (ncol > 0) ...    ! 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)
  ...
end do
```

Productivity = Performance / SLOC

Performance (on Cray XT4 and XT5)

	HPC Challenge Benchmark				
# of cores	STREAM Triad* (TByte/s)	RandomAccess‡(GUP/s)	Global HPL † (TFlop/s)	Global FFT † (GFlop/s)	UTS* (MNode/s)
64	0.17	0.08	0.36	6.69	163.1
256	0.67	0.24	1.36	22.82	645.0
1024	2.66	0.69	4.99	67.80	2371
4096	10.70	2.01	18.3	187.04	7818
8192	21.69			357.80	12286

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

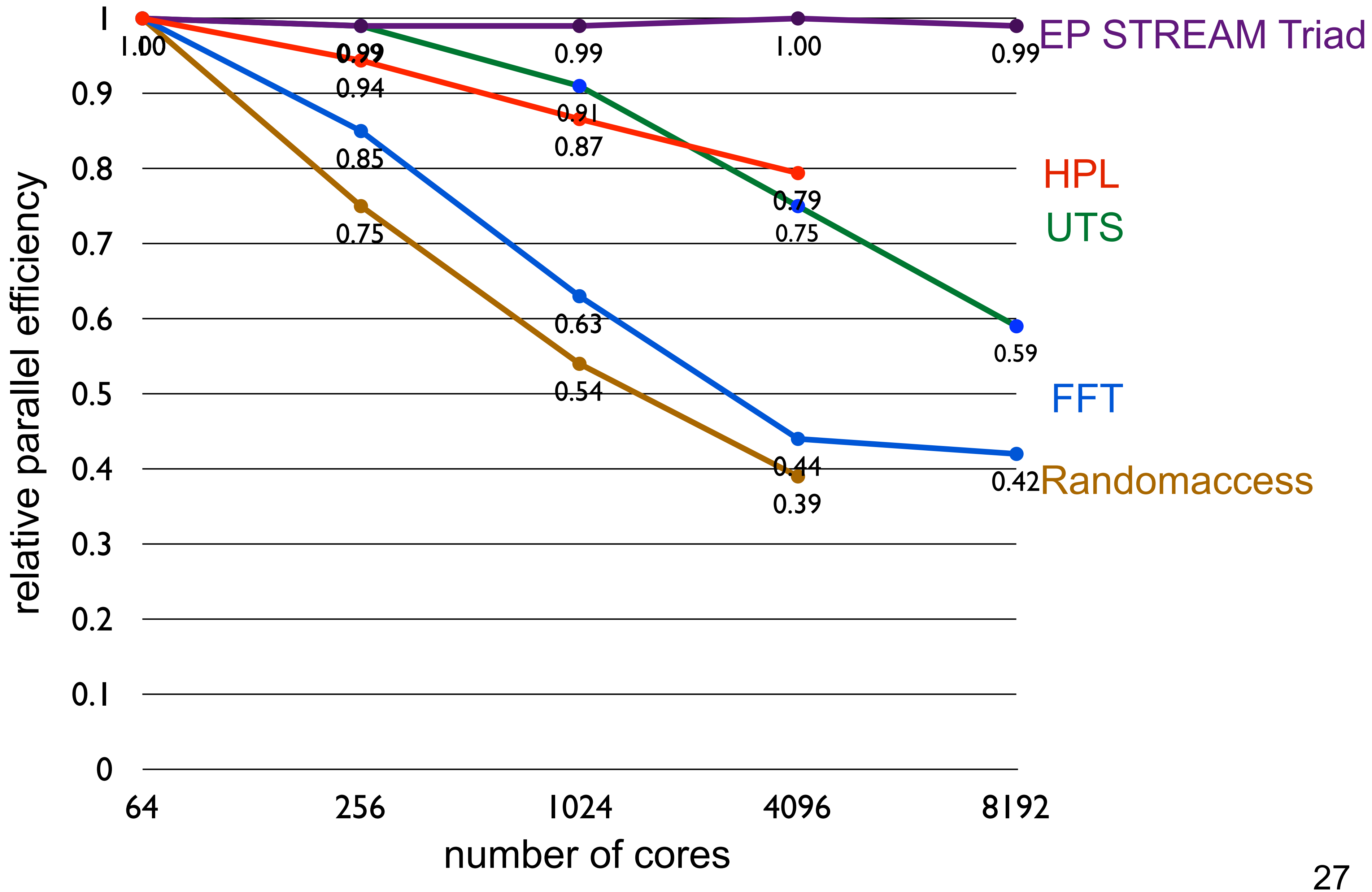
Source lines of code

Benchmark	Source Lines	Reference SLOC	Reduction
Randomaccess	409	787	48%
EP STREAM	63	329	81%
Global HPL	786	8800	91%
Global FFT	450	1130	60%
UTS	544	n/a	n/a

Notes

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

Relative Parallel Efficiency



MPI

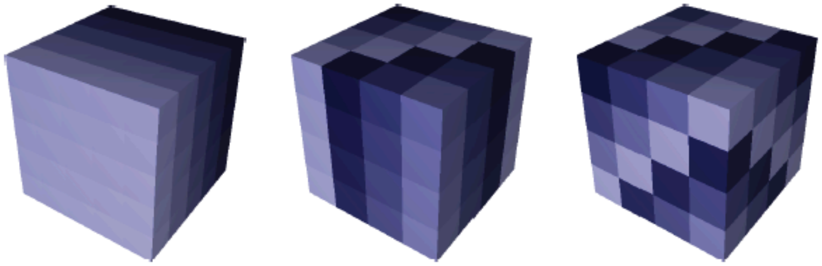
```
! post a receive
do n=1,in_bndy%nmsg_ew_rcv
   bufsize = ny_block*nghost*in_bndy%nblocks_ew_rcv(n)
   call MPI_IRECV(buf_ew_rcv(1,1,1,n), bufsize, mpi_dbl,   &
                  in_bndy%ew_rcv_proc(n)-1,                &
                  mpitag_bndy_2d + in_bndy%ew_rcv_proc(n), &
                  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)

do n=1,in_bndy%nmsg_ew_rcv
   partner = in_bndy%ew_rcv_proc(n) - 1
   do k=1,in_bndy%nblocks_ew_rcv(n)
      dst_block = in_bndy%ew_dst_block(k,n)
      ib_dst = in_bndy%ew_dst_add(1,k,n)
      ie_dst = ib_dst + nghost - 1
      ARRAY(ib_dst:ie_dst,:,dst_block) = buf_ew_rcv(:,:,k,n)
   end do
end do

! wait send to finish
call MPI_WAITALL(in_bndy%nmsg_ew_snd, snd_request, snd_status, ierr)
```

CAF 2.0

```
type :: outgoing_boundary
   double, copointer :: remote(:,:,:)[*]
   double, pointer :: local(:,:,:)
   event :: snd_ready[*]
   event, copointer :: snd_done[*]
end type

type :: incoming_boundary
   event, copointer :: rcv_ready[*]
   event :: rcv_done[*]
end type

type :: boundaries
   integer :: rcv_faces, snd_faces
   type(outgoing_boundary) :: outgoing(:)
   type(incoming_boundary) :: incoming(:)
end type

! initialize outgoing boundary
!    set remote to point to a partner's incoming boundary face
!    set local to point to one of my outgoing boundary faces
!    set snd_done to point to rcv_done of a partner's incoming boundary

! initialize incoming boundary
!    set my face's rcv_ready to point to my partner face's snd_ready

! notify each partner that my face is ready
do face=1,bndy%rcv_faces
   call event_notify(bndy%incoming(face)%rcv_ready[])
end do

! 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
do face=1,bndy%snd_faces
     copy_async(bndy%outgoing(face)%remote[], &
                bndy%outgoing(face)%local,  &
                bndy%outgoing(face)%snd_done[], &
                bndy%outgoing(face)%snd_ready)
end do

! wait for all of my incoming faces to arrive
do face=1,bndy%rcv_faces
   call event_wait(bndy%incoming(face)%rcv_done)
end do
```

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?**



```
Z, Y, X = 0, 1, 2   # Assign names to dimensions
net = box([4,4,4]) # Create a box
net.tilt(Z, X, 1)   # Tilt Z (XY) planes along X
net.tilt(X, Y, 1)   # Tilt X (YZ) planes along Y
```

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

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

- **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**