Center for Scalable Application Development Software

2011 HPC Challenge Class II Submission: **Coarray Fortran 2.0**

John Mellor-Crummey, Laksono Adhianto

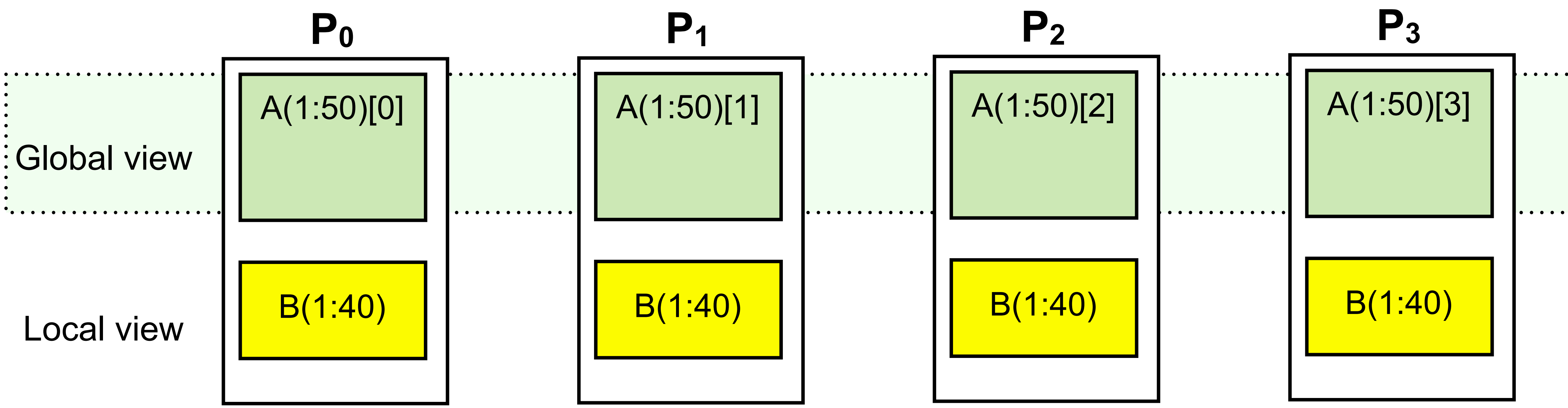
Mark Krentel, Guohua Jin, Karthik Murthy,

William Scherer III, Chaoran Yang

Department of Computer Science

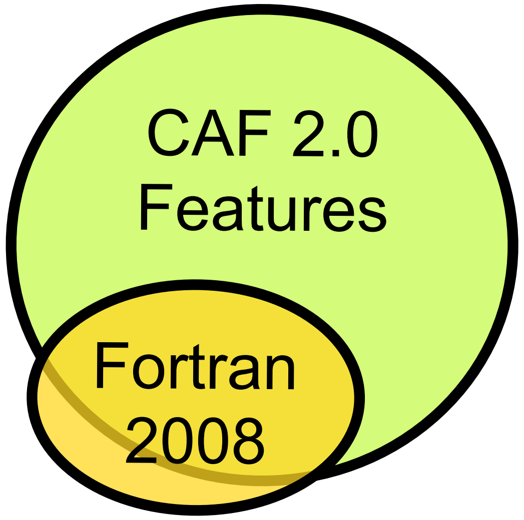
Rice University

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 (CAF 2.0)

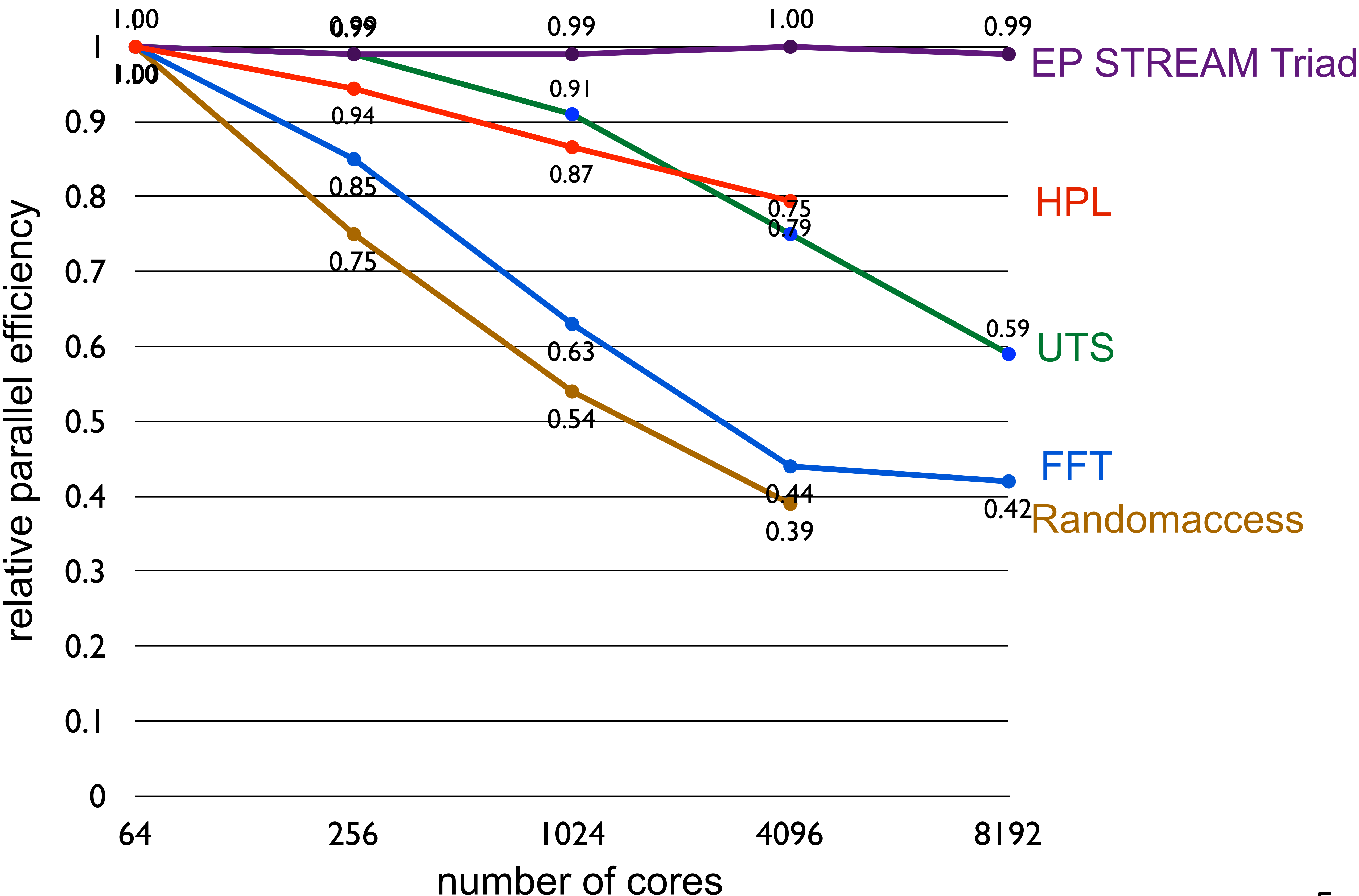


- **Teams: process subsets, like MPI communicators**
  - —formation using team_split (like MPI_Comm_split)
  - —collective communication (two-sided)
  - —barrier synchronization
- **Coarrays: shared data allocated across processor subsets**
  - —declaration: double precision :: a(:,:)[*]
  - —dynamic 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**

Our HPC Challenge Goal: Productivity

- Priorities, in order
  - —performance, performance, performance
  - —source code volume
- Productivity = performance / (lines of code)
- Implications for our implementations
  - —FFT (revised implementation for this year)
    - – 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 <u>work stealing</u> and <u>work sharing</u>

Relative Parallel Efficiency



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

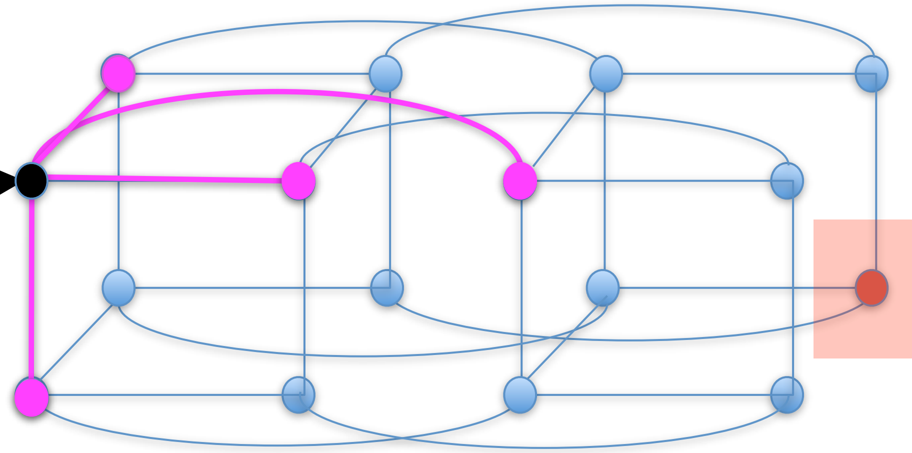
Benchmark	Source Lines	Reference SLOC	Reduction
Randomaccess	409	787	48%
EP STREAM	63	329	81%
Global HPL	786	8800	91%
Global FFT	450	1130	60%
UTS	544	n/a	n/a

Notes

- STREAM: 82% of peak memory bandwidth
- HPL: 49% of FP peak at 4096 cores (uses dgemm)
- Issues with GASNet 1.17 runtime for 4K and more processors on XE6

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
```

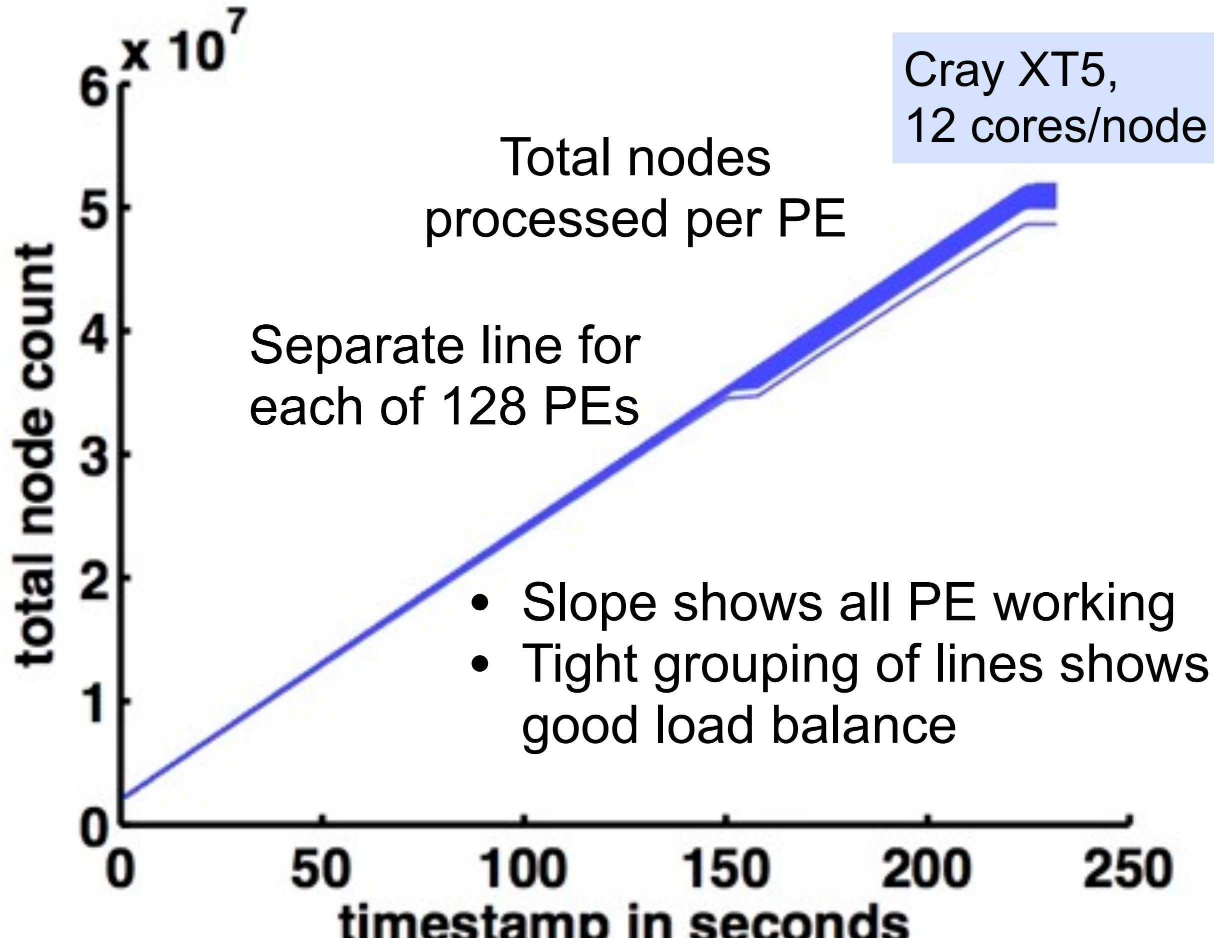
! attempt to steal work from another image

```
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
```



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

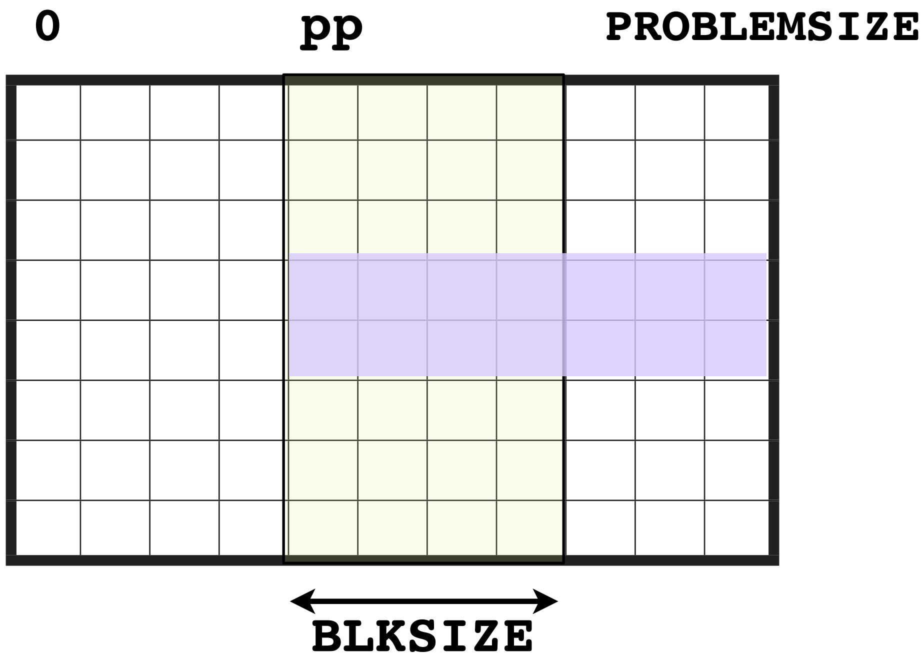
```
event, allocatable :: delivered(:)[*],received(:)[*] !(stage)
integer(i8), allocatable :: fwd(:,:,:)[*] ! (#,in/out,stage)
...
! hypercube-based routing: each processor has 1024 updates
do i = world_logsize-1, 0, -1  ! log P stages in a route
  ...
  call split(retain(:,last), ret_sizes(last), &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

  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)      2
    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
```

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
```

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

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)
- Issues with GASNet 1.17 runtime for 4K and more processors on XE6

Source Code Breakdown

	STREAM	RandomAccess	FFT	HPL	UTS
Computation	32	188	180	536	267
Communication & synchronization	1	12	4	46	17
Declaration	17	118	103	109	151
Comments & spaces	13	91	163	95	109
Total	63	409	450	786	544