2010 HPC Challenge Class II Submission: Coarray Fortran 2.0

John Mellor-Crummey, Laksono Adhianto
Mark Krentel, Guohua Jin, William Scherer III, Chaoran Yang

Department of Computer Science
Rice University

SC 2010
Coarray Fortran (CAF)

Explicitly-parallel extension of Fortran 95 (Numrich & Reid)

- 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
  - computation partitioning
  - data movement
  - synchronization

Emerging in Fortran 2008
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
  — source code volume

• Productivity = performance / (lines of code)

• Implications
  — EP STREAM Triad
    – outlined a loop to assist compiler optimization
  — Randomaccess
    – used software routing for higher performance
  — FFT
    – blocked packing/unpacking loops for bitreversal (8x gain for packing kernel)
  — HPL
    – tuned code to make good use of the memory hierarchy
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 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
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:,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

! notify partner data is there
...
• 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
  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
        ...
        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
```
• Radix 2 FFT implementation

• Block distribution of array “c” across all processors

• Computation
  — permute elements: $c = (c(\text{bitreverse}(i), i = 0, n-1)$
    - 3 parts: pack data for all-to-all; \underline{team collective all-to-all}; unpack data locally
  — FFT is log N stages
    - first $(\log N - \log P)$ stages are local
    - remaining log P stages are non-local
      - each processor has a partner; each partner does half the work
      - partner is ready $\Rightarrow$ \underline{fetch half its data using multiple asynchronous copies}
      - as the data arrives, perform your part of the computation
      - return half of your results to your partner with asynchronous copies
      - synchronize with partner to complete the stage

• Verification
  — use same code to perform inverse FFT
Experimental Setup

• Coarray Fortran 2.0 by Rice University
  — source to source compilation from CAF 2.0 to Fortran 90
    – generated code compiled with Portland Group’s pgf90
  — CAF 2.0 runtime system built upon GASNet (version 1.14.2)
  — scalable implementation of teams, using $O(\log P)$ storage

• Experimental platform: Cray XT
  — systems
    – Franklin at NERSC
      2.3 GHz AMD “Budapest” quad-core Opteron, 2GB DDR2-800/core
    – Jaguar at ORNL
      2.1 GHz AMD “Budapest” quad-core Opteron, 2GB DDR2-800/core
  — network topology
    – 3D Torus based on Seastar2 routers
    – OS provides an arbitrary set of nodes to an application
Productivity = Performance / SLOC

Performance (Cray XT4)

<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>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>0.14</td>
<td>0.08</td>
<td>0.36</td>
<td>3.66</td>
</tr>
<tr>
<td>256</td>
<td>0.54</td>
<td>0.24</td>
<td>1.36</td>
<td>11.7</td>
</tr>
<tr>
<td>1024</td>
<td>2.18</td>
<td>0.69</td>
<td>4.99</td>
<td>38.2</td>
</tr>
<tr>
<td>4096</td>
<td>8.73</td>
<td>2.01</td>
<td>18.3</td>
<td>125</td>
</tr>
</tbody>
</table>

*Measured on Jaguar
† Measured on Franklin

Source lines of code

<table>
<thead>
<tr>
<th>HPC Challenge Benchmark</th>
<th>Source Lines of Code</th>
<th>Reference SLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Randomaccess</td>
<td>409</td>
<td>787</td>
</tr>
<tr>
<td>EP STREAM Triad</td>
<td>58</td>
<td>329</td>
</tr>
<tr>
<td>Global HPL</td>
<td>786</td>
<td>8800</td>
</tr>
<tr>
<td>Global FFT</td>
<td>439</td>
<td>1130</td>
</tr>
</tbody>
</table>

Notes
- EP STREAM: 66% of memory B/W peak
- Randomaccess: high performance without special-purpose runtime
- HPL: 49% of FP peak at @ 4096 cores (uses dgemm)