Center for Scalable Application Development Software

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

SciDAC Scientific Discovery through Advanced Computing

RICE

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

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 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
  ...
  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
```

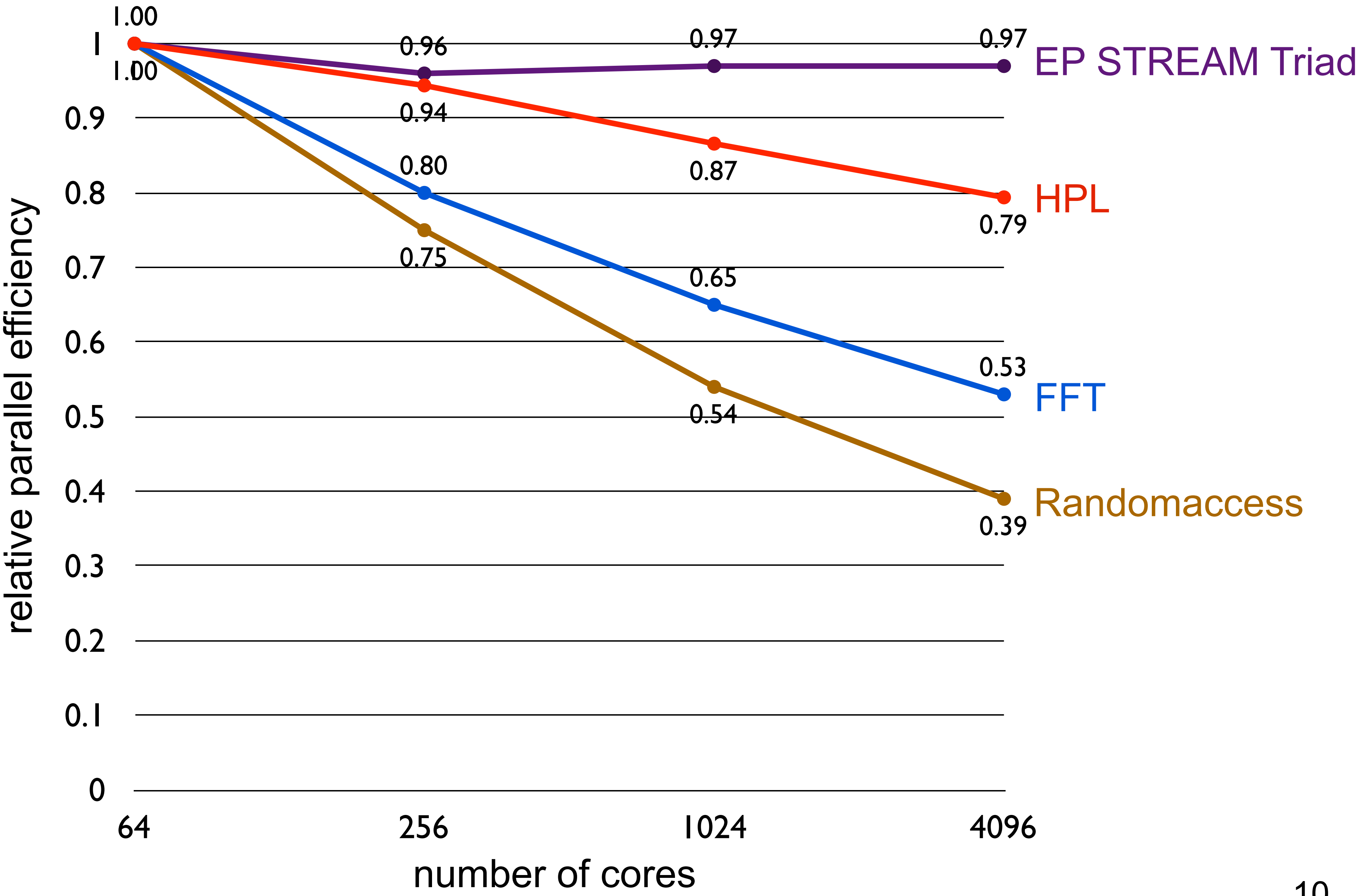
FFT

- Radix 2 FFT implementation
- Block distribution of array “c” across all processors
- Computation
  - —permute elements: c = (/ c(bitreverse(i), i = 0, n-1 /)
    - – 3 parts: pack data for all-to-all; 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 ⇒ 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

CAF 2.0 HPCC Relative Parallel Efficiency



Productivity = Performance / SLOC

Performance (Cray XT4)

# of cores	HPC Challenge Benchmark: STREAM Triad † (TByte/s)	RandomAccess* (GUP/s)	Global HPL † (TFlop/s)	Global FFT † (GFlop/s)
64	0.14	0.08	0.36	3.66
256	0.54	0.24	1.36	11.7
1024	2.18	0.69	4.99	38.2
4096	8.73	2.01	18.3	125

*Measured on Jaguar † Measured on Franklin

Source lines of code

HPC Challenge Benchmark	Source Lines of Code	Reference SLOC
Randomaccess	409	787
EP STREAM Triad	58	329
Global HPL	786	8800
Global FFT	439	1130

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)