Coarray Fortran Tutorial: Parallel Programming in Fortran 2018

Damian Rouson
Berkeley Lab

The International Conference for High Performance Computing, Networking, Storage, and Analysis 2023 Tutorial

go.lbl.gov/sc23

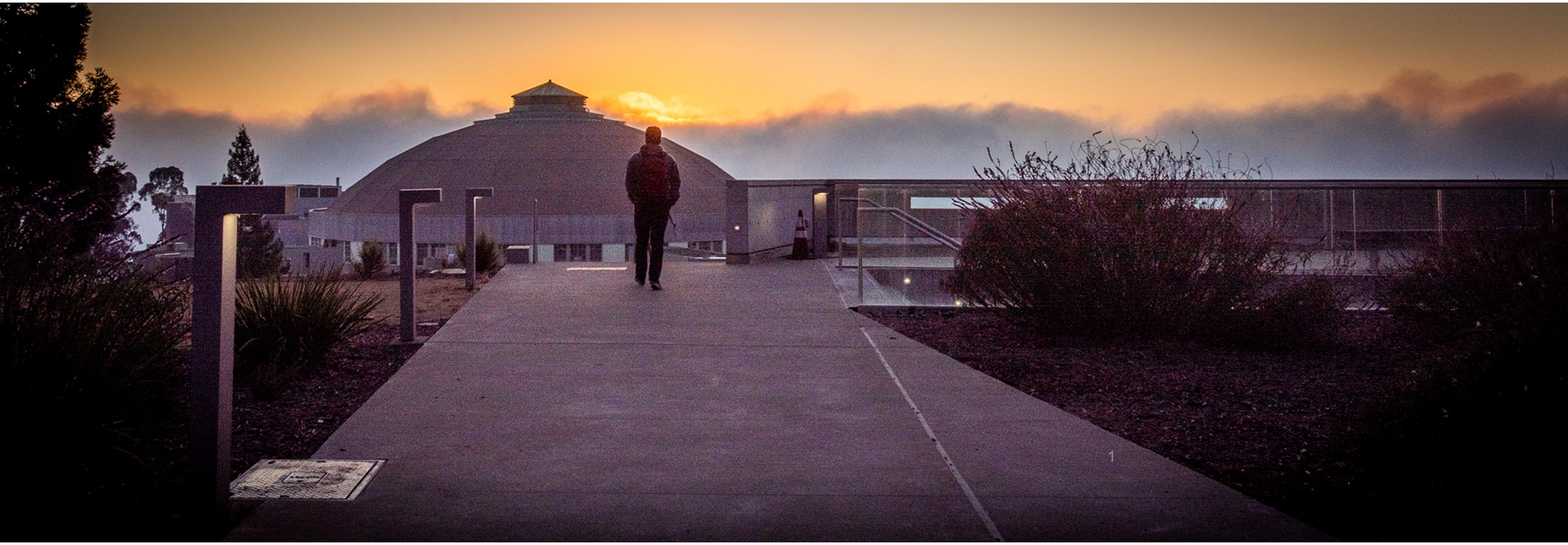
BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY
Office of Science

Coarray Fortran Tutorial

Damian Rouson
Computer Languages & System Software

SC23, November 12, 2023



Acknowledgements

This presentation includes efforts on the part of contributors to the GASNet-EX, Matcha, and OpenCoarrays software libraries and members of the Computer Languages and Systems Software (CLaSS) Group and our collaborators:

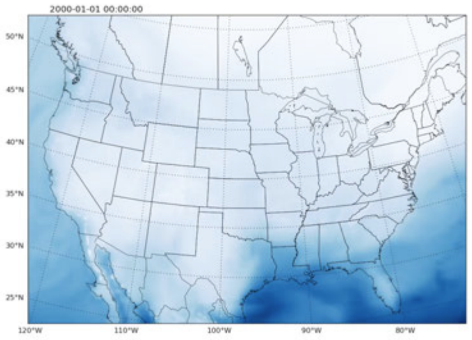
Amir Kamil, Dan Bonachea, Paul Hargrove, Tobias Burnus, Alessandro Fanfarillo, Daniel Ceils Garza, Ethan Gutmann, Jeff Hammond, Peter Hill, Paul Hargrove, Dominick Martinez, Katherine Rasmussen, Soren Rasmussen, Brad Richardson, Sameer Shende, David Torres, Andre Vehreschild, Jordan Welsman, Nathan Weeks, Yunhao Zhang

This research was supported in part by the **Exascale Computing Project** (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering and early testbed platforms, in support of the nation's exascale computing imperative.

This research used resources of the **National Energy Research Scientific Computing Center (NERSC)**, a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231, as well as This research used resources of the **Oak Ridge Leadership Computing Facility** at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

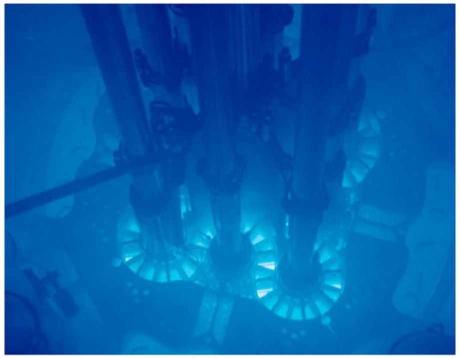
- Introduction to Coarray Fortran (“CAF”)
    - Motivation: Why Fortran, CAF philosophy
    - SPMD parallel execution: Images
    - PGAS data structures: Coarrays
    - Example Application: Matcha
    - Compiling and running “Hello, world!”
- Break
- A deeper dive

Why Fortran Matters



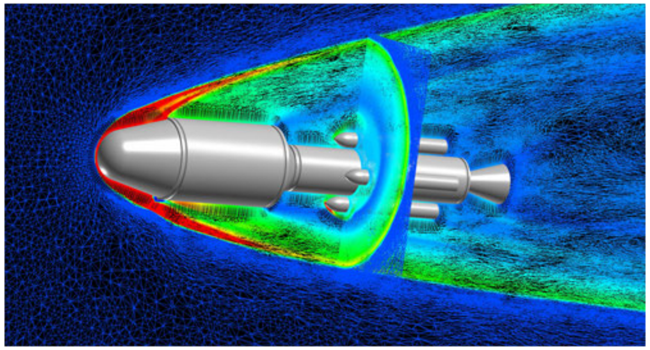
Intermediate Complexity Atmospheric Research (ICAR) Model
Courtesy of Ethan Gutmann, NCAR

Weather & Climate



U.S. Nuclear Regulatory Commission File Photo

Nuclear Energy



FUN3D Mesh Adaptation for Mars Ascent Vehicle, Courtesy of Eric Nielsen & Ashley Korzun, NASA Langley

Aerospace

CAF Philosophy

“The underlying philosophy of our design is to make the smallest number of changes to the language required to obtain a robust and efficient parallel language without requiring the programmer to learn very many new rules.”

Reid, J., & Numrich, R. W. (2007). Co-arrays in the next Fortran standard. *Scientific Programming*, *15*(1), 9-26.

Seminal paper:

Numrich, R. W., & Reid, J. (1998, August). Co-Array Fortran for parallel programming. In *ACM SIGPLAN Fortran Forum* (Vol. 17, No. 2, pp. 1-31). New York, NY, USA: ACM.

Single Program Multiple Data

BERKELEY LAB
Bringing Science Solutions to the World

```
cd fortran
make run-hi
```

Single Program Multiple Data (SPMD) parallel execution

— Synchronized launch of multiple “images” (process/threads/ranks)

— Asynchronous execution except where program explicitly synchronizes

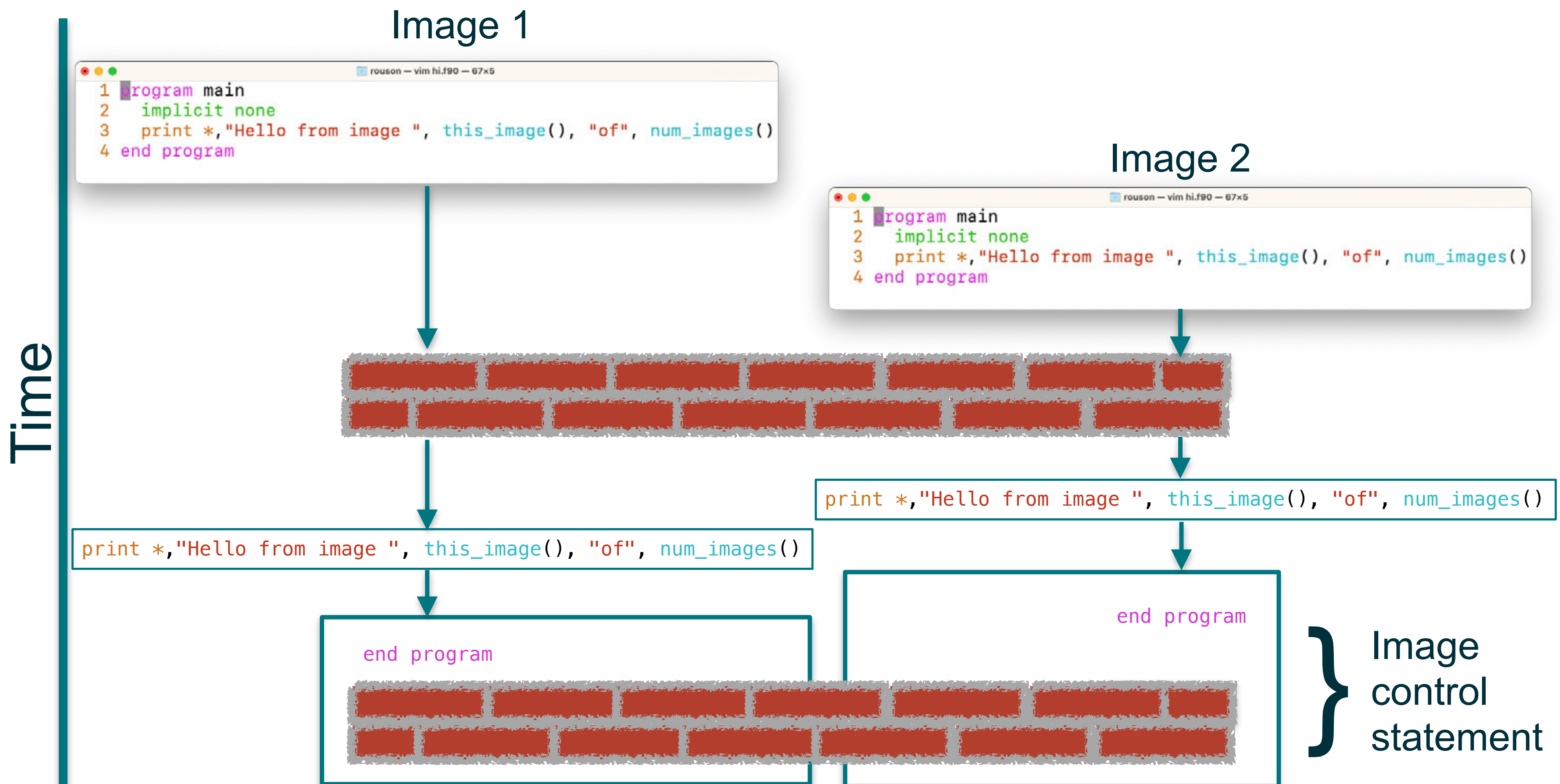
— Error termination or synchronized normal termination

rouson — vim hi.f90 — 67×5

```fortran
program main
  implicit none
  print *,"Hello from image ", this_image(), "of", num_images()
end program
```

SPMD Execution Sequence

BERKELEY LAB
Bringing Science Solutions to the World



1. After the creation of a fixed number of images, each image’s first “segment” (sequence of statements) executes.
2. Image control statements totally order segments executed by a single image and partially order segments executed by separate images.

Partitioned Global Address Space (PGAS)

BERKELEY LAB
Bringing Science Solutions to the World

```
cd fortran
make run-hello
```

Coarrays:

— Distributed data structures — `greeting`

— Facilitate Remote Memory Access (RMA) — line 15

cuf23-tutorial — vim hello.f90 — 74×21

```
1 program main
2   !! One-sided communication of distributed greetings
3   implicit none
4   integer, parameter :: max_greeting_length=64, writer = 1
5   integer image
6   character(len=max_greeting_length) :: greeting[*] ! scalar coarray
7
8   associate(me => this_image(), ni=>num_images())
9
10    write(greeting,*) "Hello from image",me,"of",ni ! local (no "[]")
11    sync all ! image control
12
13    if (me == writer) then
14      do image = 1, ni
15        print *,greeting[image] ! one-sided communication: "get"
16      end do
17    end if
18
19  end associate
20 end program
```

Coarrays

BERKELEY LAB
Bringing Science Solutions to the World

- Non-allocatable (static):

```
character(len=max_greeting_length) :: greeting[*]
```

- Dynamically allocatable:

```
real(rkind), allocatable :: halo_x(:,:)[:]
```

- Derived type components:

```
type global_field_t
  real, allocatable :: values_(:)[:]
end type
```

- Local coarrays:

```
subroutine gather_image_numbers
  integer, allocatable :: images(:)[:]
  allocate(images(num_images())[*])
end subroutine
```

- Derived type coarrays:

```
type payload_list_t
  type(payload_t), allocatable :: payloads(:)
end type

type(payload_list_t), allocatable :: mailbox[:]
```

} Allow for asymmetric memory

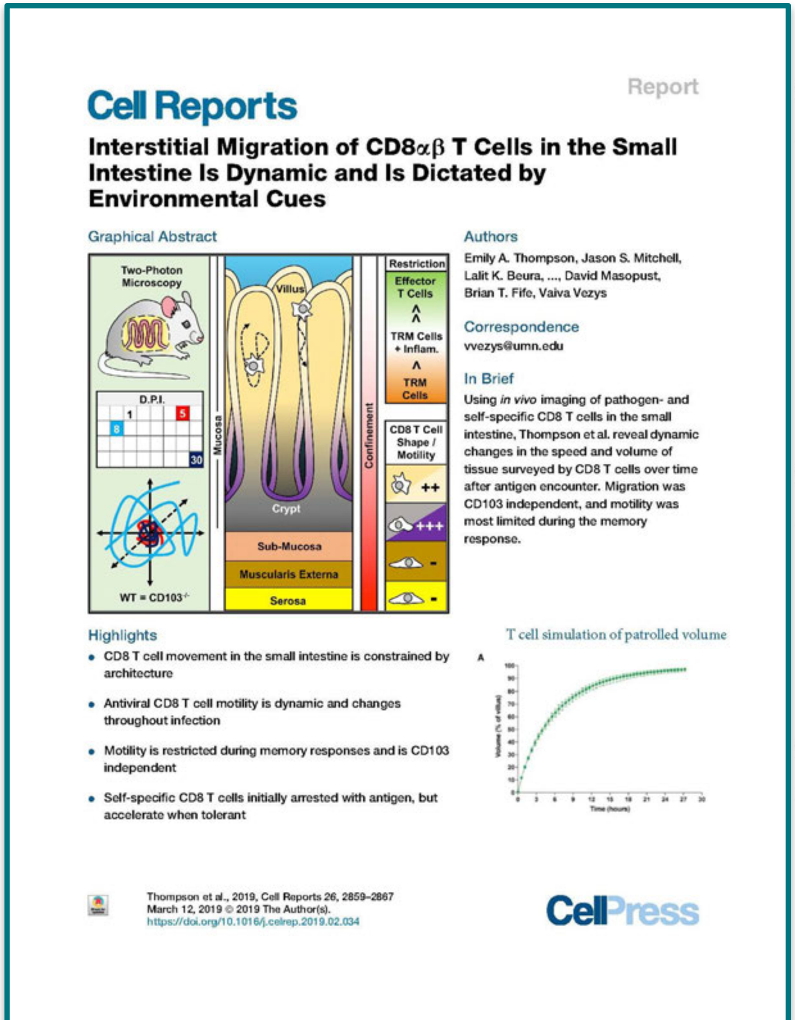
A coarray is a data entity that has nonzero corank; it can be directly referenced or defined by other images. It may be a scalar or an array.

For each coarray on an image, there is a corresponding coarray with the same type, type parameters, and bounds on every other image of a team in which it is established

=> Symmetric memory
if intrinsic-type coarray

New Frontiers: T-Cell Motility

BERKELEY LAB
Bringing Science Solutions to the World



Report

Cell Reports

Interstitial Migration of CD8αβ T Cells in the Small Intestine Is Dynamic and Is Dictated by Environmental Cues

Graphical Abstract

Authors

Emily A. Thompson, Jason S. Mitchell, Lalit K. Beura, ..., David Masopust, Brian T. Fife, Vaiva Vezys

Correspondence

vvezys@umn.edu

In Brief

Using *in vivo* imaging of pathogen- and self-specific CD8 T cells in the small intestine, Thompson et al. reveal dynamic changes in the speed and volume of tissue surveyed by CD8 T cells over time after antigen encounter. Migration was CD103 independent, and motility was most limited during the memory response.

Highlights

- CD8 T cell movement in the small intestine is constrained by architecture
- Antiviral CD8 T cell motility is dynamic and changes throughout infection
- Motility is restricted during memory responses and is CD103 independent
- Self-specific CD8 T cells initially arrested with antigen, but accelerate when tolerant

T cell simulation of patrolled volume

Thompson et al., 2019, Cell Reports 26, 2859–2867
March 12, 2019 © 2019 The Author(s).
https://doi.org/10.1016/j.celrep.2019.02.034

CellPress

Thompson, E. A., Mitchell, J. S., Beura, L. K., Torres, D. J., Mrass, P., Pierson, M. J., ... & Vezys, V. (2019). Interstitial migration of CD8αβ T cells in the small intestine is dynamic and is dictated by environmental cues. *Cell reports*, *26*(11), 2859-2867.

- Application:
  - — Matcha: Motility Analysis of T Cells in Activation
  - — Matching the speed & turning angle distributions to observed T cells, simulations can explore large spatial volumes and parameter spaces.
- Programming models:
  - — Coarray halo exchanges in a 3D diffusion PDE solver.
  - — `Do concurrent` for automatic GPU offloading
- Highlights:
  - — This tutorial's 2D heat equation solver was the prototype for the 3D diffusion solver.

https://go.lbl.gov/matcha

Compiling & Running `hello.f90`

BERKELEY LAB
Bringing Science Solutions to the World

cuf23-tutorial — -zsh — 66×15

```
cuf23-tutorial: 
```

- Introduction to Coarray Fortran ("CAF")
- Break
- A deeper dive
  - Heat equation:
    - Numerical algorithm
    - Abstract calculus design pattern
    - Halo exchanges
    - Performance analysis
  - CAF Overview:
    - Image enumeration
    - Synchronization
    - Collective subroutines
    - Events
    - Example: FEATS task scheduler

Heat Equation

BERKELEY LAB
Bringing Science Solutions to the World

```
cd fortran
make run-heat-equation
```

$$\frac{\partial T}{\partial t} = \alpha \nabla^2 T$$

$$\{T\}^{n+1} = \{T\}^n + \Delta t \cdot \alpha \cdot \nabla^2 \{T\}^n$$

```
T =  T + dt * alpha * .laplacian. T
```

Heat Equation

BERKELEY LAB
Bringing Science Solutions to the World

```
cd fortran
make run-heat-equation
```

$$\frac{\partial T}{\partial t} = \alpha \nabla^2 T$$

$$\{T\}^{n+1} = \{T\}^n + \Delta t \cdot \alpha \cdot \nabla^2 \{T\}^n$$

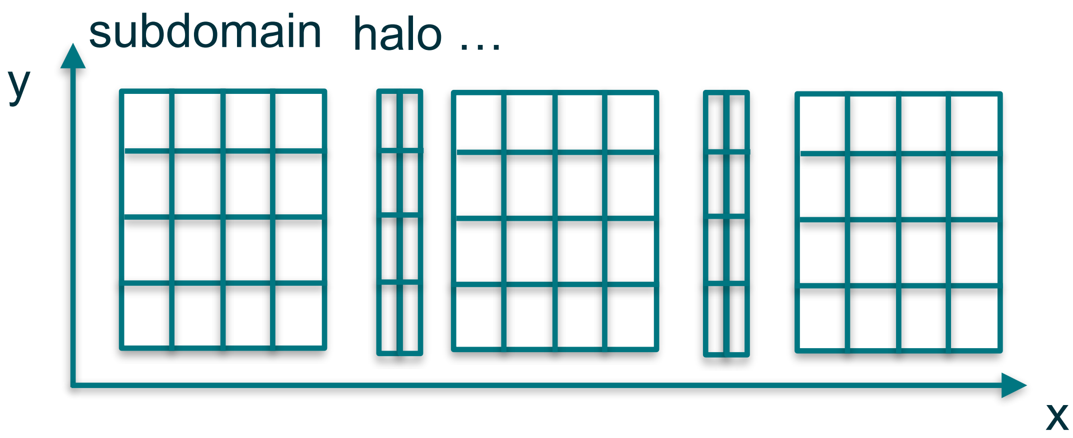
```
T = T + dt * alpha * .laplacian. T
```

local objects

pure user-defined operators

Halo Exchange

BERKELEY LAB
Bringing Science Solutions to the World



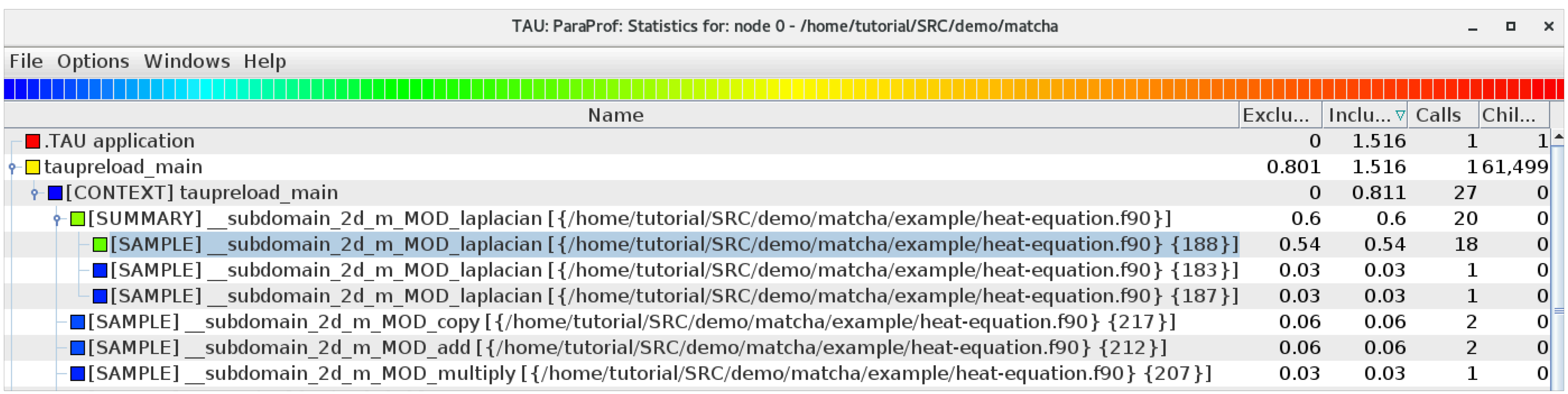

```fortran
real(rkind), allocatable :: halo_x(:,:)[:]
integer, parameter :: west=1, east=2

me = this_image()
num_subdomains = num_images()
my_nx = nx/num_subdomains + merge(1, 0, me <= mod(nx, num_subdomains))

subroutine exchange_halo(self)
  class(subdomain_2D_t), intent(in) :: self
  if (me>1) halo_x(east,:)[me-1] = self%s_(1,:)
  if (me<num_subdomains) halo_x(west,:)[me+1] = self%s_(my_nx,:)
end subroutine
```

Loop-Level Parallelism

BERKELEY LAB
Bringing Science Solutions to the World

line continuation

```
188 do concurrent(j=2:ny-1)
189   laplacian_rhs%s_(i, j) = &
      (halo_left(j)   - 2*rhs%s_(i, j) + rhs%s_(i+1,j  ))/dx_**2 + &
190   (rhs%s_(i, j-1) - 2*rhs%s_(i, j) + rhs%s_(i  ,j+1))/dy_**2
191 end do
```



Name	Exclu...	Inclu...	Calls	Chil...
.TAU application	0	1.516	1	1
taupreload_main	0.801	1.516	1	61,499
[CONTEXT] taupreload_main	0	0.811	27	0
[SUMMARY] __subdomain_2d_m_MOD_laplacian [{/home/tutorial/SRC/demo/matcha/example/heat-equation.f90}]	0.6	0.6	20	0
[SAMPLE] __subdomain_2d_m_MOD_laplacian [{/home/tutorial/SRC/demo/matcha/example/heat-equation.f90} {188}]	0.54	0.54	18	0
[SAMPLE] __subdomain_2d_m_MOD_laplacian [{/home/tutorial/SRC/demo/matcha/example/heat-equation.f90} {183}]	0.03	0.03	1	0
[SAMPLE] __subdomain_2d_m_MOD_laplacian [{/home/tutorial/SRC/demo/matcha/example/heat-equation.f90} {187}]	0.03	0.03	1	0
[SAMPLE] __subdomain_2d_m_MOD_copy [{/home/tutorial/SRC/demo/matcha/example/heat-equation.f90} {217}]	0.06	0.06	2	0
[SAMPLE] __subdomain_2d_m_MOD_add [{/home/tutorial/SRC/demo/matcha/example/heat-equation.f90} {212}]	0.06	0.06	2	0
[SAMPLE] __subdomain_2d_m_MOD_multiply [{/home/tutorial/SRC/demo/matcha/example/heat-equation.f90} {207}]	0.03	0.03	1	0

Comments

BERKELEY LAB
Bringing Science Solutions to the World

- Coarray Fortran began as a syntactically small extension to Fortran 95:
  - — Square-bracketed “cosubscripts” distribute & communicate data
- Integration with other features:
  - — Array programming: colon subscripts
  - — OOP: distributed objects
- Minimally invasive:
  - — Drop brackets when not communicating
- Communication is explicit:
  - — Use brackets when communicating

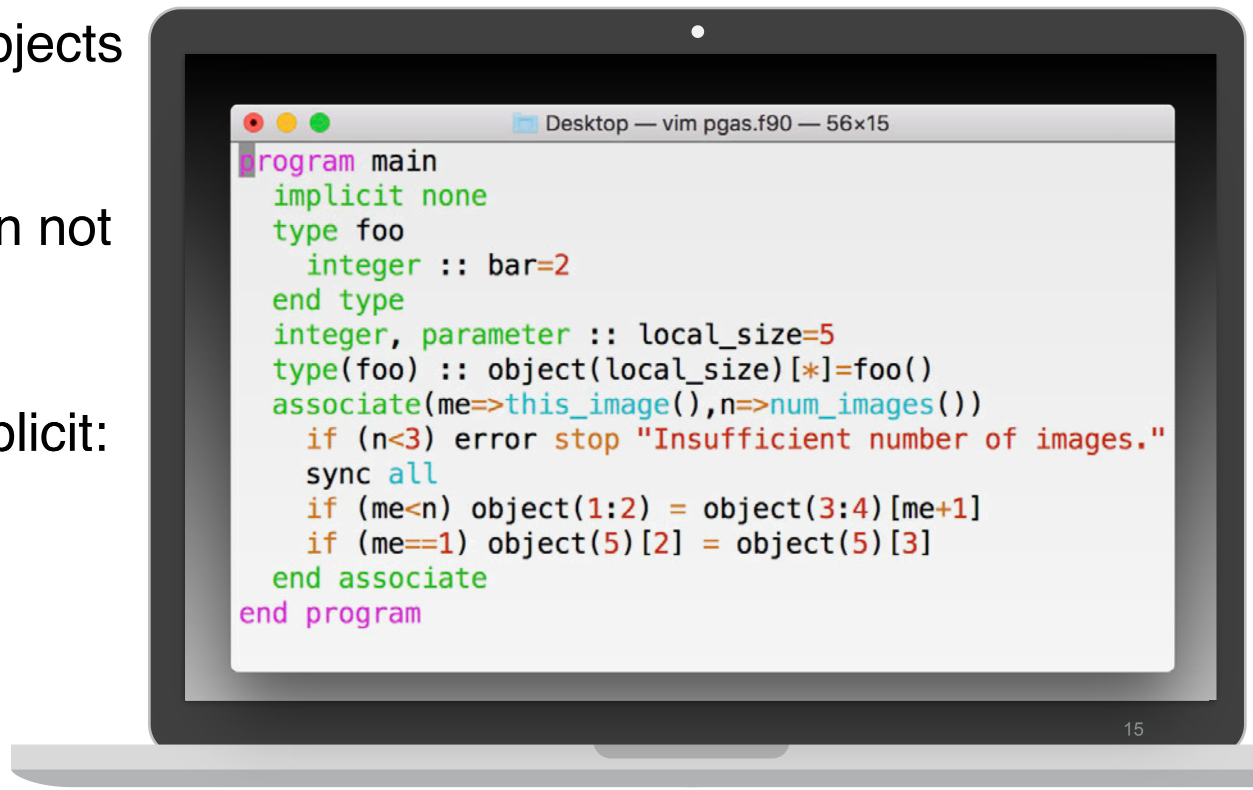


Image Enumeration

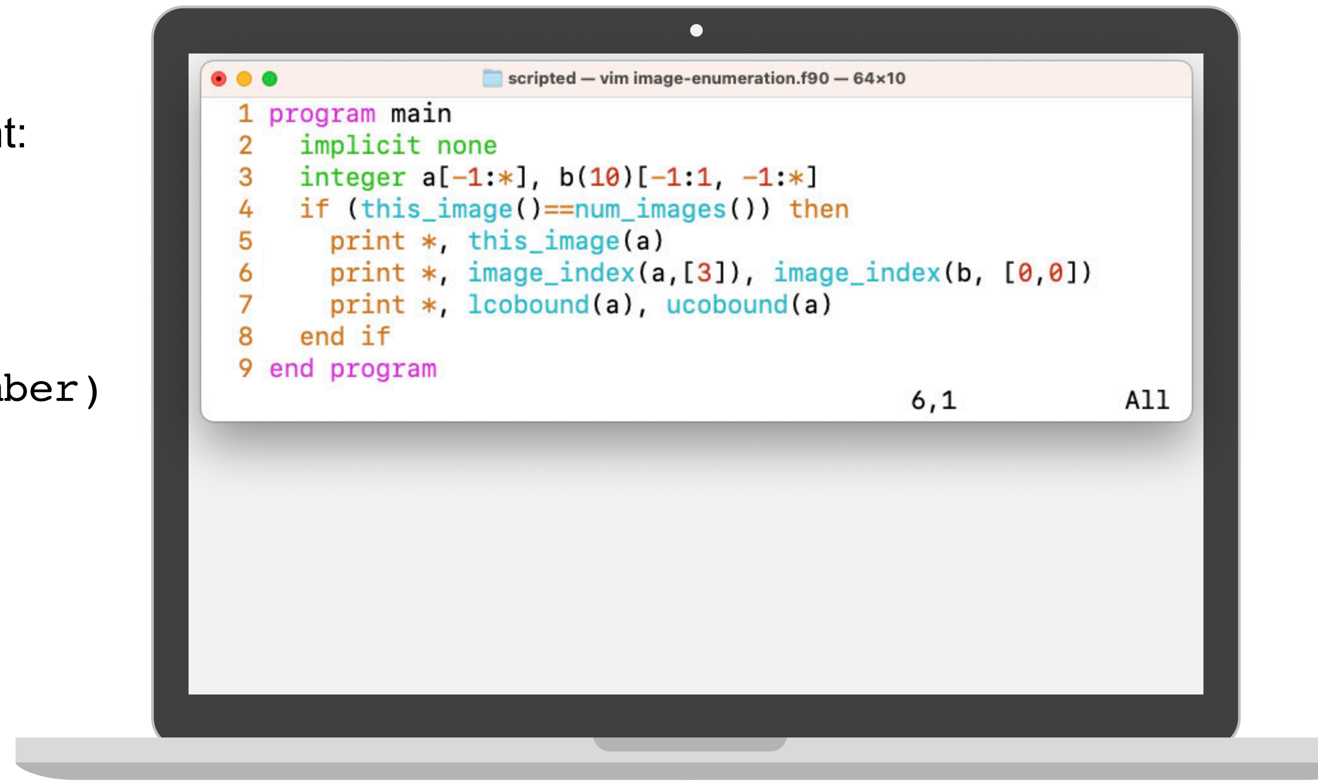
BERKELEY LAB
Bringing Science Solutions to the World

- Obtaining an image index:

```
this_image([team])                   image_index(coarray, sub, team_number)
this_image(coarray [,team])          image_index(coarray, sub, team)
this_image(coarray, dim [,team])     image_index(coarray, sub)
```

- Obtaining an image count:

```
num_images()
num_images(team)
num_images(team_number)
```



Synchronization

BERKELEY LAB
Bringing Science Solutions to the World

- Image barriers ("meet-ups"):

  `sync all(stat, errmsg)`

  `sync images(image-set, stat, errmsg)`

  `allocate()`

  `deallocate()`

  } for coarrays only, including implicit (de)allocation at end of a block or procedure

  `stop stop_code` (integer or character codes allowed)

  `end program`

  `call move_alloc(from,to)` with coarray arguments.

  Any statement causing an implicit coarray deallocation by completing a block or procedure.

- Deprecated by Metcalf, Reid & Cohen (2018):

  `sync memory(stat, errmsg)`

Other Image Control Statements

- Locks:

```
lock(lock-variable, errmsg)
unlock(lock-variable, stat, errmsg)
```

A lock variable is a coarray object of the extensible intrinsic type `lock_type` with private components.

- Critical blocks:

```
critical(stat, errmsg)
end critical
```

- Teams

```
form team(team_number, team_variable)
change team(team_value, …)
end team
```

- Events

```
event post(event-variable, stat, errmsg)
event wait(event-variable, stat, errmsg)
```

An event variable is a coarray object of the extensible intrinsic type `event_type` with private components.

Collective Subroutines

BERKELEY LAB
Bringing Science Solutions to the World

- Behavior:
  - Successful execution of a collective subroutine performs a calculation on all the images of the current team and assigns a computed value on one or all of them.
  - If it is invoked by one image, it shall be invoked by the same statement on all active images of its current team in segments that are not ordered with respect to each other
  - Corresponding references participate in the same collective computation.
- Complete list:
  - co_sum(a, result_image, stat, errmsg)
  - co_max(a, result_image, stat, errmsg)
  - co_min(a, result_image, stat, errmsg)
  - co_broadcast(a, source_image, stat, errmsg)
  - co_reduce(a, operation, result_image, stat, errmsg)

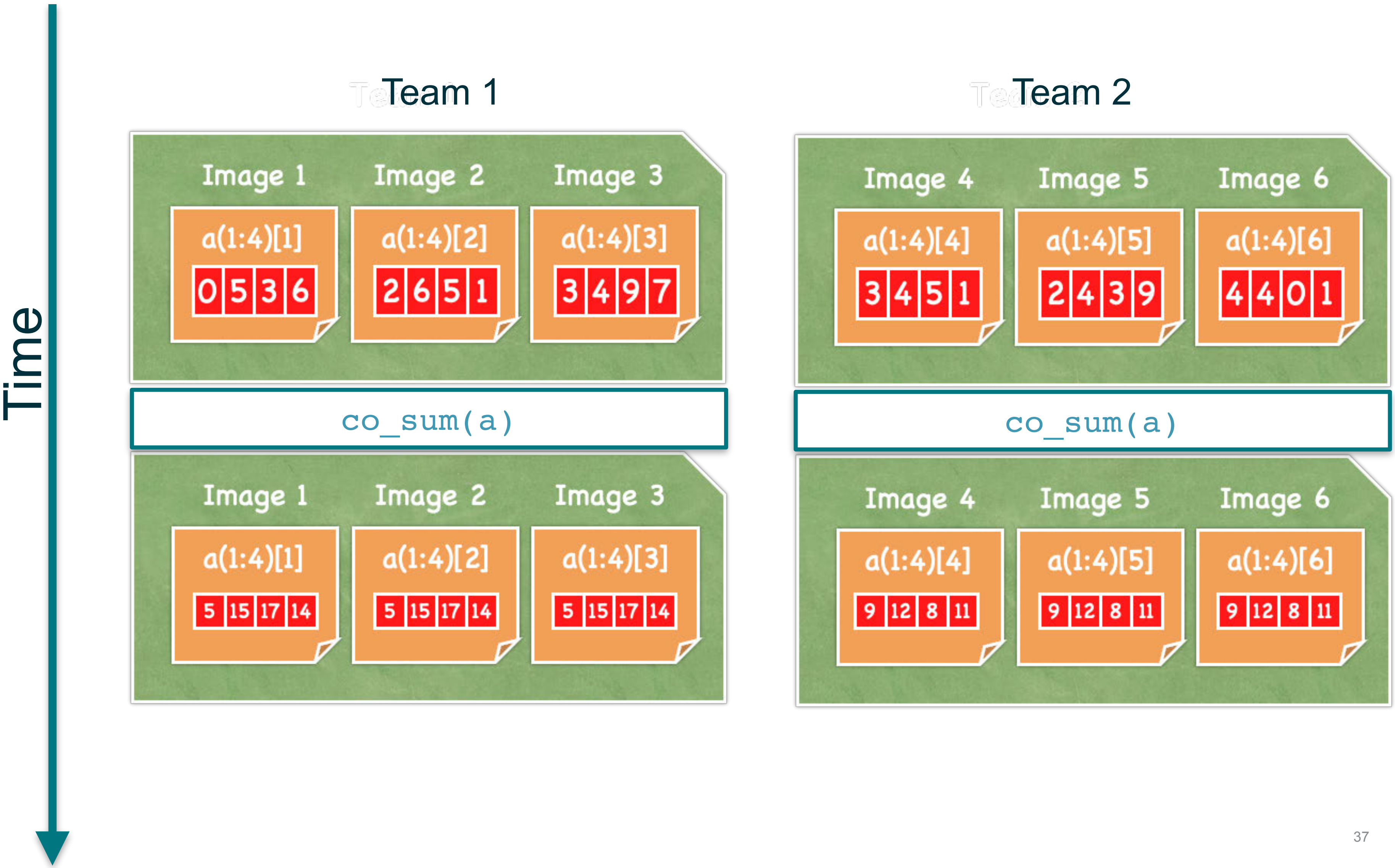
co_sum

BERKELEY LAB
Bringing Science Solutions to the World

```
co_sum(a, result_image, stat, errmsg)
```

- Argument `a`
  - — shall be of numeric type,
  - — shall have the same shape, type, & type parameter values, in corresponding references.
  - — shall not be a coindexed object
  - — is an `intent(inout)` argument
- Argument `result_image` (optional)
  - — shall be of scalar type `integer`
  - — is an `intent(in)` argument
  - — If present, it shall be present on all images of the current team, have the same value on all images of the current team, and shall be an image index of the current team

co_sum


Time
Team 1
Image 1
Image 2
Image 3
a(1:4)[1]
0 5 3 6
a(1:4)[2]
2 6 5 1
a(1:4)[3]
3 4 9 7
co_sum(a)
Image 1
Image 2
Image 3
a(1:4)[1]
5 15 17 14
a(1:4)[2]
5 15 17 14
a(1:4)[3]
5 15 17 14
Team 2
Image 4
Image 5
Image 6
a(1:4)[4]
3 4 5 1
a(1:4)[5]
2 4 3 9
a(1:4)[6]
4 4 0 1
co_sum(a)
Image 4
Image 5
Image 6
a(1:4)[4]
9 12 8 11
a(1:4)[5]
9 12 8 11
a(1:4)[6]
9 12 8 11

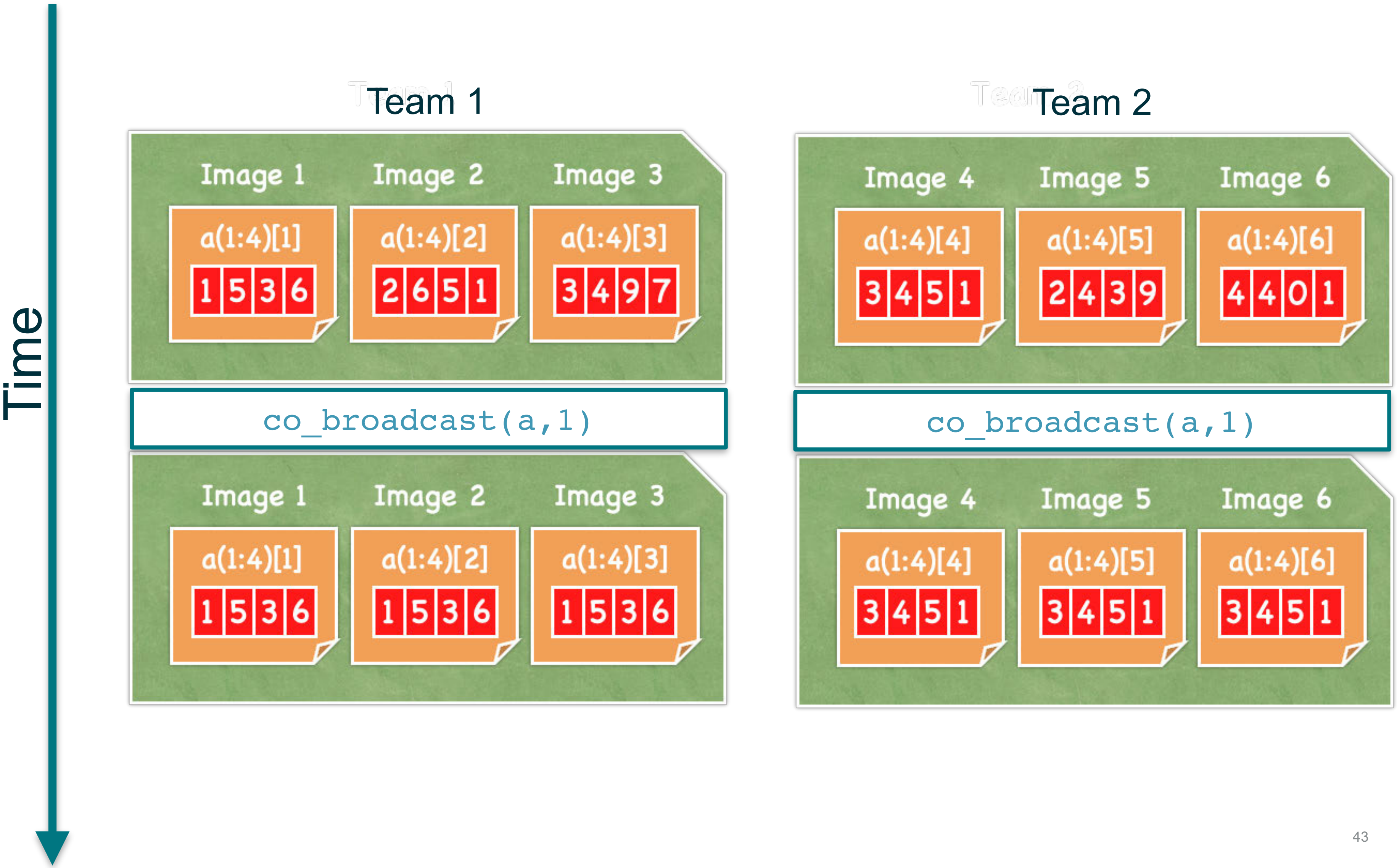
co_broadcast

```
co_broadcast(a, source_image, stat, errmsg)
```

- Argument `a`
  - — shall have the same shape, dynamic type, & type parameter values, in corresponding references.
  - — shall not be a coindexed object
  - — is an `intent(inout)` argument
  - — successful execution causes a to become defined as if by intrinsic assignment on all images in the current team with the value of a on the source_image
- Argument `source_image`
  - — shall be of scalar type `integer`
  - — is an `intent(in)` argument
  - — If present, it shall be present on all images of the current team, have the same value on all images of the current team, and shall be an image index of the current team

co_broadcast

BERKELEY LAB
Bringing Science Solutions to the World



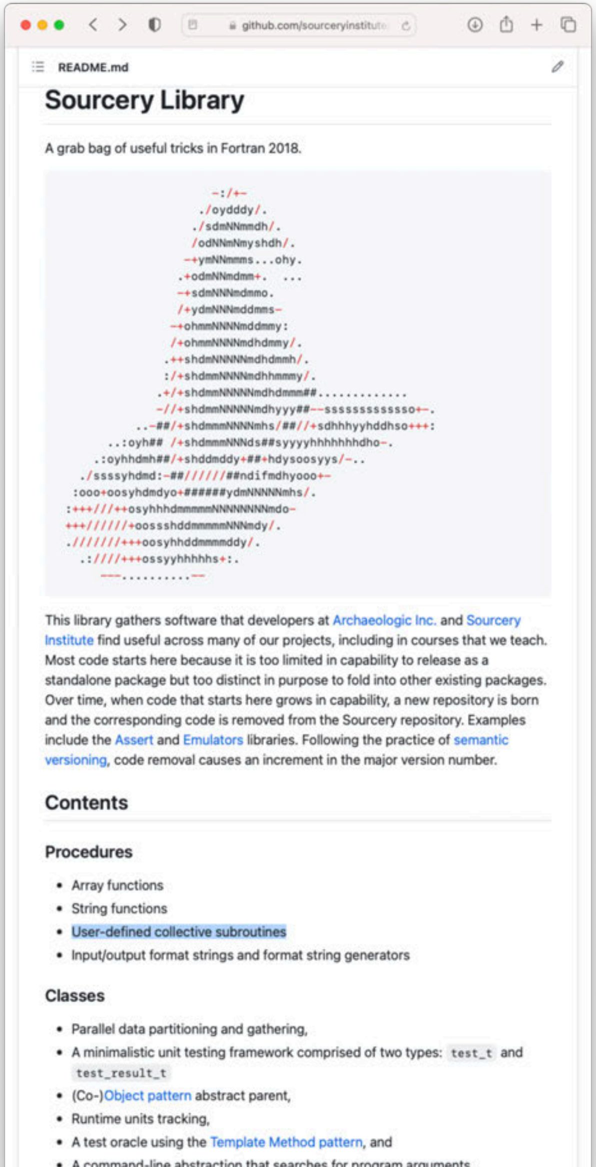
co_reduce

BERKELEY LAB
Bringing Science Solutions to the World

```
co_reduce(a, operation, result_image, stat, errmsg)
```

- Argument `a`
  - — shall be `intent(inout)`, non-polymorphic and not coindexed
  - — shall have the same shape, dynamic type, & type parameter values, in corresponding references.
  - — becomes the result of applying the reduction operation to values of a in the corresponding references, and likewise on an element-wise basis if a is an array
- Argument operation
  - —shall implement an associative operation via a `pure` function with two arguments
- Argument `result_image`
  - —shall be of scalar `integer, intent(in) argument`
  - —if present, it shall have the same value on all images of the current team and shall be an image index of the current team

Hands-on co_reduce


README.md

Sourcery Library

A grab bag of useful tricks in Fortran 2018.

This library gathers software that developers at Archaeologic Inc. and Sourcery Institute find useful across many of our projects, including in courses that we teach. Most code starts here because it is too limited in capability to release as a standalone package but too distinct in purpose to fold into other existing packages. Over time, when code that starts here grows in capability, a new repository is born and the corresponding code is removed from the Sourcery repository. Examples include the Assert and Emulators libraries. Following the practice of semantic versioning, code removal causes an increment in the major version number.

Contents

Procedures

- Array functions
- String functions
- User-defined collective subroutines
- Input/output format strings and format string generators

Classes

- Parallel data partitioning and gathering,
- A minimalistic unit testing framework comprised of two types: `test_t` and `test_result_t`
- (Co-)Object pattern abstract parent,
- Runtime units tracking,
- A test oracle using the Template Method pattern, and
- A command-line abstraction that searches for program arguments.

https://github.com/sourceryinstitute/sourcery

```fortran
module co_all_m
  implicit none

  interface
    module subroutine co_all(a)
      implicit none
      logical, intent(inout) :: a
    end subroutine
  end interface

end module

submodule(co_all_m) co_all_s
  implicit none
contains
  module procedure co_all
     call co_reduce(a, and)
  contains
     pure function and(lhs, rhs) result(lhs_and_rhs)
       logical, intent(in) :: lhs, rhs
       logical lhs_and_rhs
       lhs_and_rhs = lhs .and. rhs
     end function
  end procedure
end submodule

program main
  use co_all_m, only : co_all
  implicit none
  logical :: operand = .true.

  associate(me=>this_image())
    call co_all(operand)
    if (me==1) print *, operand
    if (me==num_images()) operand = .false.
    call co_all(operand)
    if (me==1) print *, operand
  end associate
end program
```

Heat Equation Solver

BERKELEY LAB
Bringing Science Solutions to the World

cuf23-tutorial — vim heat-equation.f90 — 110×39

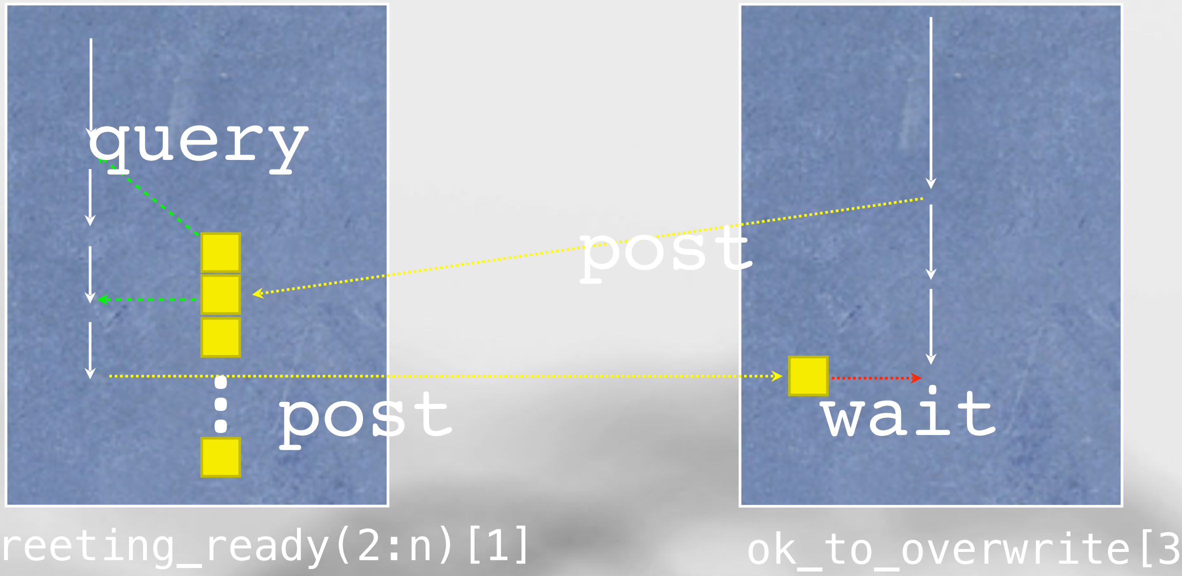
```fortran
program heat_equation
  !! Parallel finite difference solver for the 2D, unsteady heat conduction partial differential equation
  use subdomain_2D_m, only : subdomain_2D_t
  use iso_fortran_env, only : int64
  use kind_parameters_m, only : rkind
  implicit none
  type(subdomain_2D_t) T
  integer, parameter :: nx = 4096, ny = nx, steps = 50
  real(rkind), parameter :: alpha = 1._rkind
  real(rkind) T_sum
  integer(int64) t_start, t_finish, clock_rate
  integer step

  call T%define(side=1._rkind, boundary_val=1._rkind, internal_val=2._rkind, n=nx)! Initial/boundary cond.
  call T%allocate_halo_coarray ! implicit synchronization

  associate(dt => T%dx()*T%dy()/(4*alpha)) ! set time step

    call system_clock(t_start)

    do step = 1, steps
      call T%exchange_halo ! put subdomain boundary values on neighboring images
      sync all
      T =  T + dt * alpha * .laplacian. T ! asynchronous parallel user-defined operators
      sync all
    end do

  end associate

  T_sum = sum(T%values()) ! local sum
  call co_sum(T_sum, result_image=1) ! distributed collective sum

  call system_clock(t_finish, clock_rate)
  if (this_image()==1) then
    print *, "walltime: ", real(t_finish - t_start, rkind) / real(clock_rate, rkind)
    print *,"T_avg = ", T_sum/(nx*ny)
  end if
end program
```

Compiling and Running the Heat Equation Solver

BERKELEY LAB
Bringing Science Solutions to the World

Events

Hello, world!



Performance-oriented constraints:
– Query and wait must be local.
– Post and wait are disallowed in do concurrent constructs.

Pro tips:
– Overlap communication and computation.
– Wherever safety permits, query without waiting.

Segment Ordering: Events

An intrinsic module provides the derived type `event_type`, which encapsulates an `atomic_int_kind` integer component default-initialized to zero.

- An image increments the event count on a remote image by executing `event post`.
- The remote image obtains the post count by executing `event_query`.

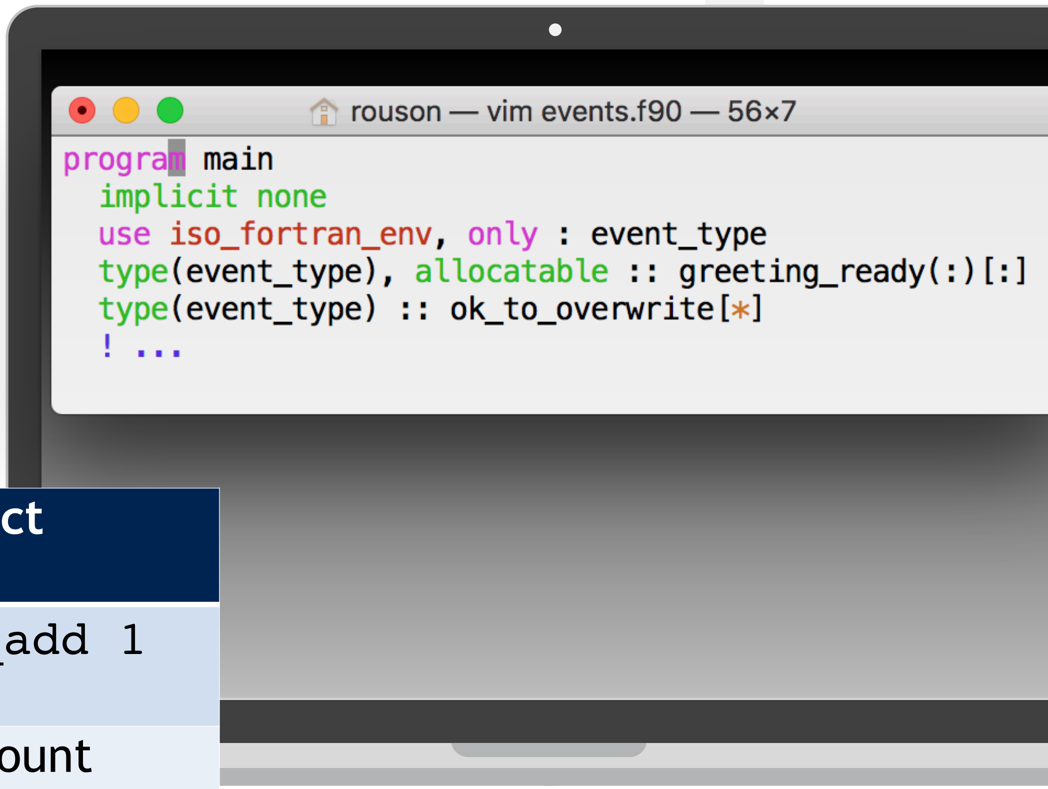


	Image Control	Side Effect
event post	☒	atomic_add 1
event_query		defines count
event wait	☒	atomic_add -1

FEATS:

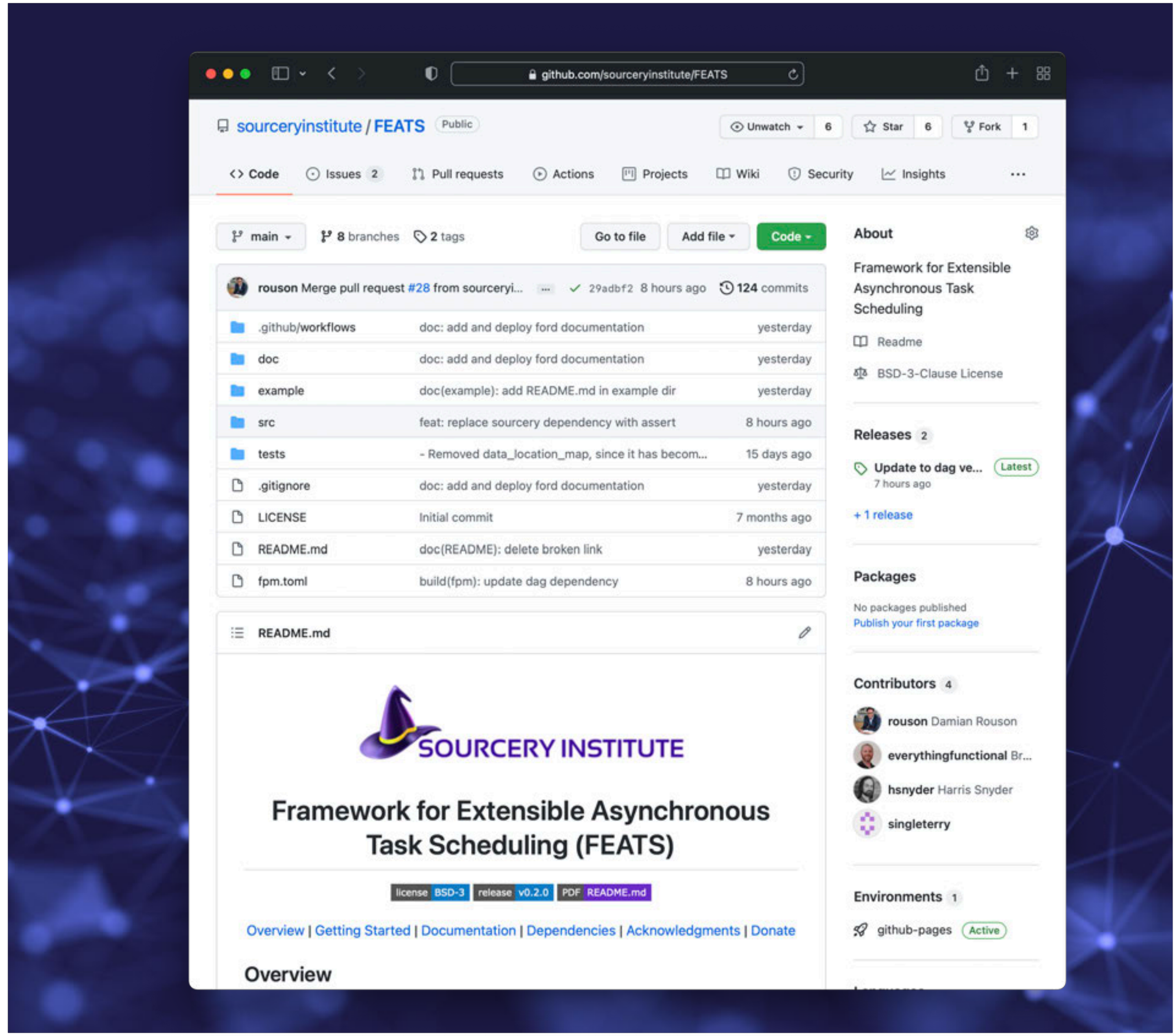
Framework for Extensible Asynchronous Task Scheduling

Execution:

- In each team, establish one scheduler image and one or more compute images.
- Schedulers post task_assigned events to compute images in an order that respects dependencies in a directed acyclic graph (DAG).
- Compute images post ready_for_next_task events to scheduler.
- A task_payload_map_t abstraction maps task task identifiers to locations in a payload_t mailbox coarray.

Initial target applications:

- NASA's Online Tool for the Assessment of Radiation in Space (OLTARIS)
- NCAR's Intermediate Complexity Atmospheric Research (ICAR) model: work-sharing/work-stealing.
- Fortran Package Manager: parallel builds.



FEATS:

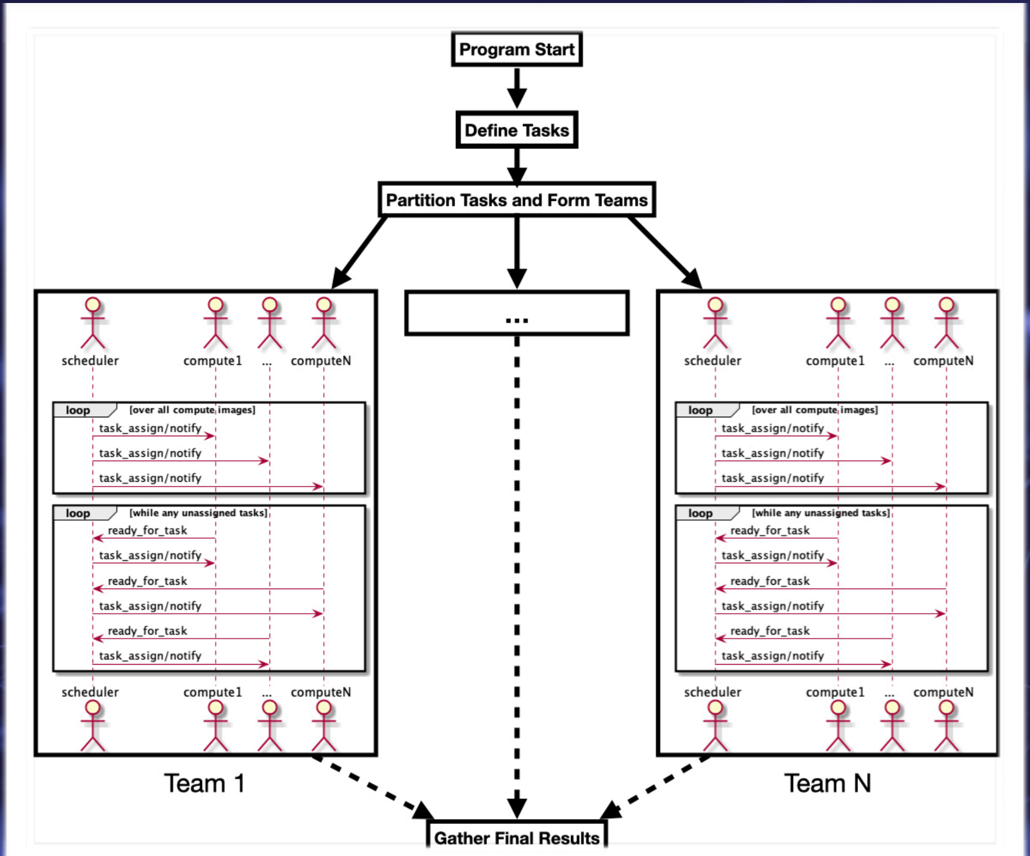
Framework for Extensible Asynchronous Task Scheduling

Execution:

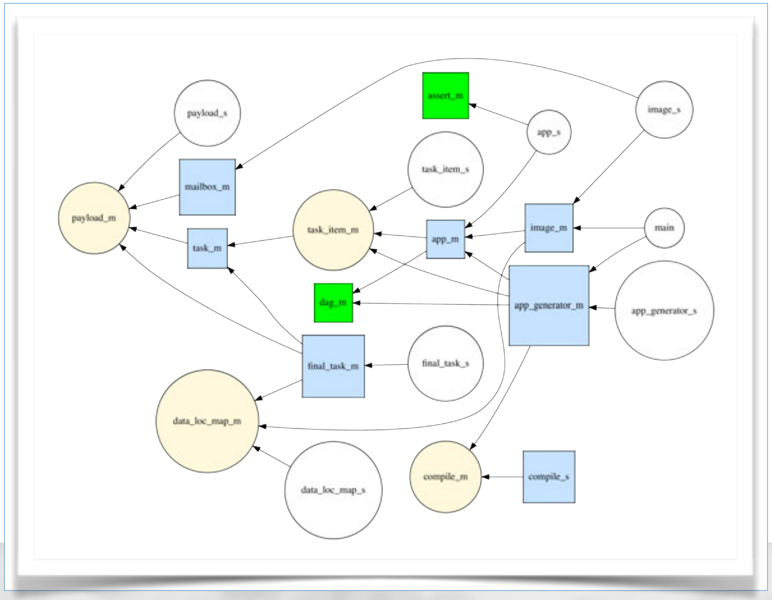
- In each team, establish one scheduler image and one or more compute images.
- Schedulers post task_assigned events to compute images in an order that respects dependencies in a directed acyclic graph (DAG).
- Compute images post ready_for_next_task events to scheduler.
- A task_payload_map_t abstraction maps task task identifiers to locations in a payload_t mailbox coarray.

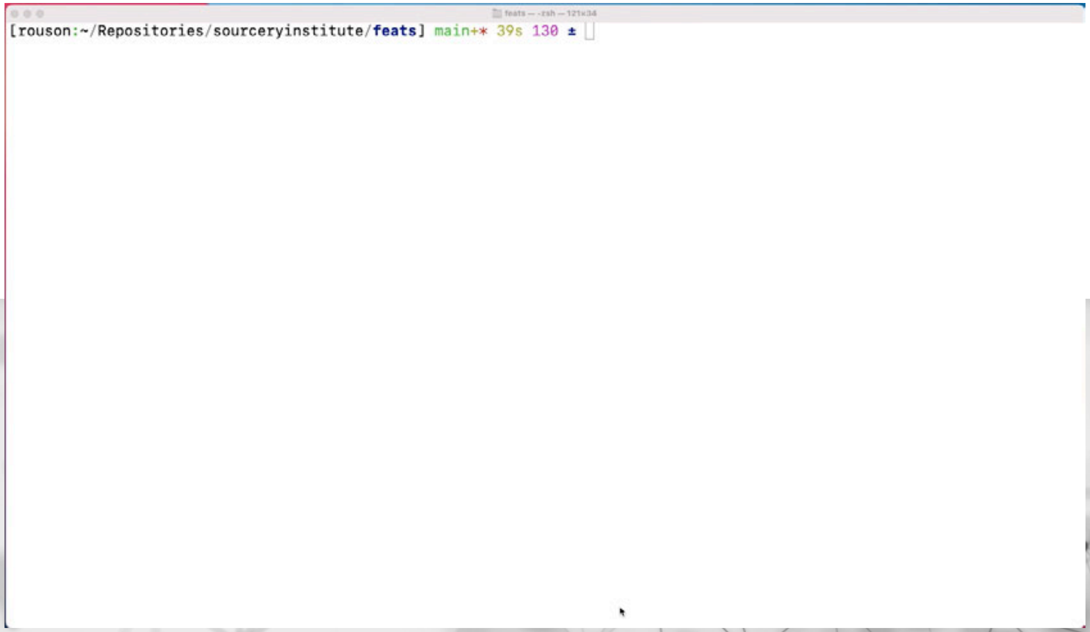
Initial target applications:

- NASA's Online Tool for the Assessment of Radiation in Space (OLTARIS)
- NCAR's Intermediate Complexity Atmospheric Research (ICAR) model: work-sharing/work-stealing.
- Fortran Package Manager: parallel builds.



Demo


payload_s
assert_m
image_s
app_s
task_item_s
mailbox_m
payload_m
task_item_m
app_m
image_m
main
task_m
dag_m
app_generator_m
app_generator_s
final_task_m
final_task_s
data_loc_map_m
data_loc_map_s
compile_m
compile_s


[rouson:~/Repositories/sourceryinstitute/feats] main+* 39s 130 ±

CAF at Scale: Magnetic Fusion

BERKELEY LAB
Bringing Science Solutions to the World

Multithreaded Global Address Space Communication Techniques for Gyrokinetic Fusion Applications on Ultra-Scale Platforms

Robert Preissl
Lawrence Berkeley National Laboratory
Berkeley, CA, USA 94720
rpreissl@lbl.gov

Nathan Wichmann
CRAY Inc.
St. Paul, MN, USA, 55101
wichmann@cray.com

Bill Long
CRAY Inc.
St. Paul, MN, USA, 55101
longb@cray.com

John Shalf
Lawrence Berkeley National Laboratory
Berkeley, CA, USA 94720
jshalf@lbl.gov

Stephane Ethier
Princeton Plasma Physics Laboratory
Princeton, NJ, USA, 08543
ethier@pppl.gov

Alice Koniges
Lawrence Berkeley National Laboratory
Berkeley, CA, USA 94720
aekoniges@lbl.gov

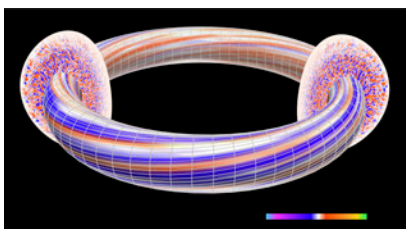


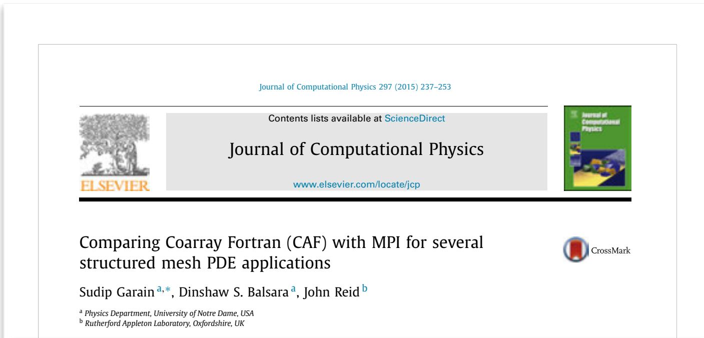
Figure 2: GTS field-line following grid & toroidal domain decomposition. Colors represent isocontours of the quasi-two-dimensional electrostatic potential

- Application focus:
  - — The shift phase of charged particles in a tokamak simulation code
- Programming models studied:
  - — CAF + OpenMP or
  - — Two-sided MPI + OpenMP
- Highlights:
  - — Experiments on up to 130,560 processors
  - — 58% speed-up of the CAF implementation over the best multithreaded MPI shifter algorithm on largest scale
  - — “the complexity required to implement … MPI-2 one-sided, in addition to several other semantic limitations, is prohibitive.”

Preissl, R., Wichmann, N., Long, B., Shalf, J., Ethier, S., & Koniges, A. (2011, November). Multithreaded global address space communication techniques for gyrokinetic fusion applications on ultra-scale platforms. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 1-11).

CAF at Scale: CFD, FFTs, Multigrid

BERKELEY LAB
Bringing Science Solutions to the World


Journal of Computational Physics 297 (2015) 237–253

Contents lists available at ScienceDirect

Journal of Computational Physics

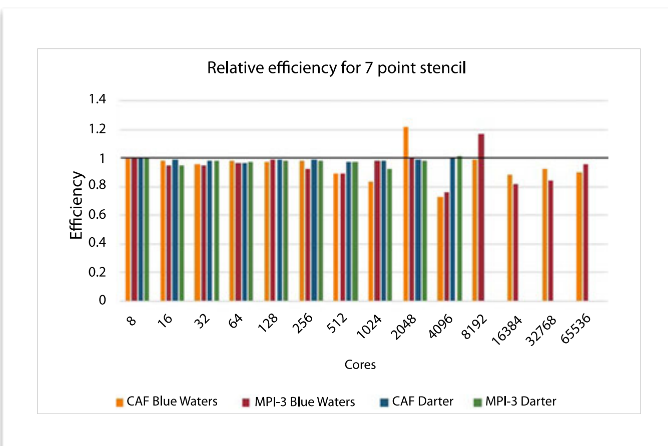
www.elsevier.com/locate/jcp

ELSEVIER

Comparing Coarray Fortran (CAF) with MPI for several structured mesh PDE applications

Sudip Garain[a,*], Dinshaw S. Balsara[a], John Reid[b]

[a] Physics Department, University of Notre Dame, USA
[b] Rutherford Appleton Laboratory, Oxfordshire, UK



- **Applications studied:**
  - — Magnetohydrodynamics (MHD)
  - — 3D Fast Fourier Transforms (FFTs) used in infinite-order accurate spectral methods
  - — Multigrid methods with point-wise smoothers requiring fine-grained messaging
- **Programming models studied:**
  - — CAF or
  - — One-sided MPI-3
- **Highlights:**
  - — Simulations on up to 65,536 cores
  - — "… CAF either draws level with MPI-3 or shows a slight advantage over MPI-3."
  - — "CAF and MPI-3 are shown to provide substantial advantages over MPI-2.
  - — "CAF code is of course much easier to write and maintain…"

Garain, S., Balsara, D. S., & Reid, J. (2015). Comparing Coarray Fortran (CAF) with MPI for several structured mesh PDE applications. *Journal of Computational Physics*, *297*, 237-253.

CAF at Scale: Weather

BERKELEY LAB
Bringing Science Solutions to the World

Article

International Journal of HIGH PERFORMANCE COMPUTING APPLICATIONS

The International Journal of High Performance Computing Applications
2015, Vol. 29(3) 261–273

DOI: 10.1177/1094342015576773
hpc.sagepub.com
SAGE

A Partitioned Global Address Space implementation of the European Centre for Medium Range Weather Forecasts Integrated Forecasting System

George Mozdzynski, Mats Hamrud and Nils Wedi

Figure 7. EQ_REGIONS partitioning of grid-point space, showing a partition at the poles and then an increasing number of partitions as we approach the equator.

- Application:
  - European Centre for Medium Range Weather Forecasts (ECMWF) operational weather forecast model
- Programming models studied:
  - CAF or
  - Two-sided MPI
- Highlights:
  - Simulations on > 60K cores
  - performance improvement from switching to CAF peaks at 21% around 40K cores

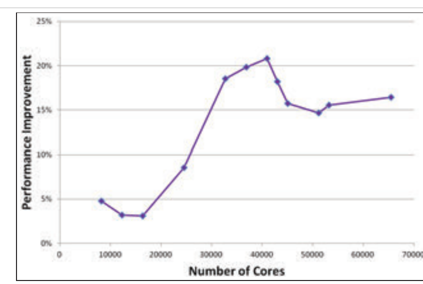
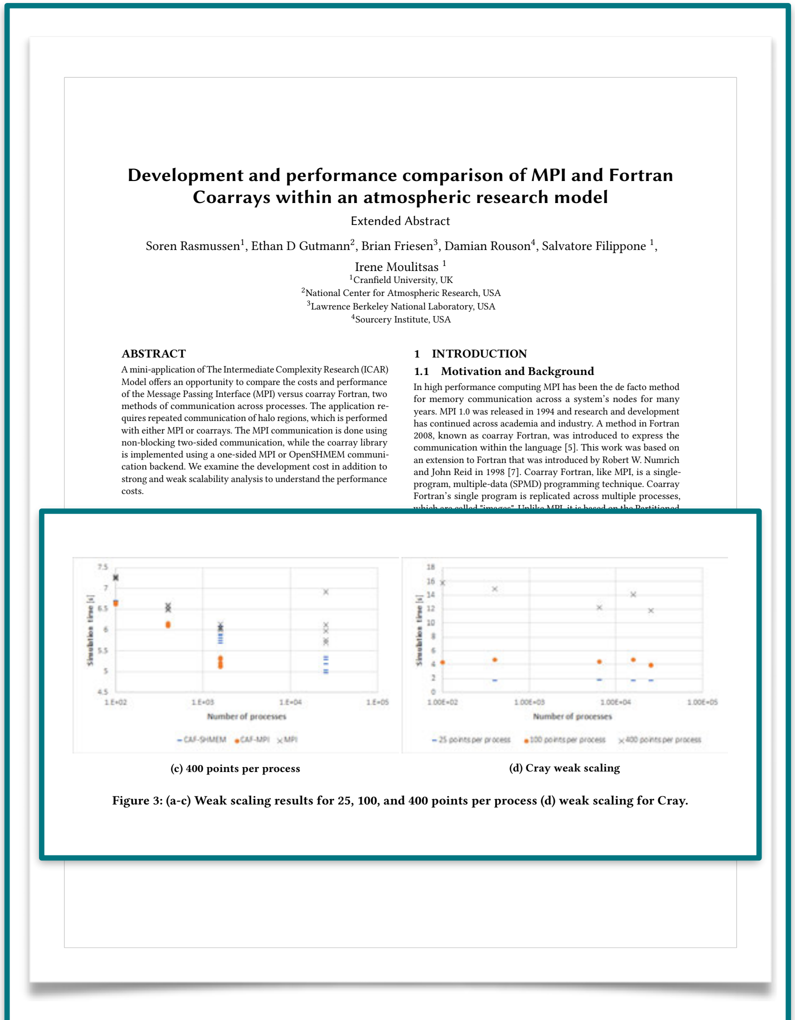


Figure 14. Performance improvement of the T2047 (~10 km) model with 137 levels by using Fortran2008 coarrays on HECToR (Cray XE6).

Mozdzynski, G., Hamrud, M., & Wedi, N. (2015). A partitioned global address space implementation of the European centre for medium range weather forecasts integrated forecasting system. *The International Journal of High Performance Computing Applications*, *29*(3), 261-273.

CAF at Scale: Climate

BERKELEY LAB
Bringing Science Solutions to the World



Development and performance comparison of MPI and Fortran Coarrays within an atmospheric research model

Extended Abstract

Soren Rasmussen[1], Ethan D Gutmann[2], Brian Friesen[3], Damian Rouson[4], Salvatore Filippone [1], Irene Moulitsas [1]

[1]Cranfield University, UK
[2]National Center for Atmospheric Research, USA
[3]Lawrence Berkeley National Laboratory, USA
[4]Sourcery Institute, USA

ABSTRACT

A mini-application of The Intermediate Complexity Research (ICAR) Model offers an opportunity to compare the costs and performance of the Message Passing Interface (MPI) versus coarray Fortran, two methods of communication across processes. The application requires repeated communication of halo regions, which is performed with either MPI or coarrays. The MPI communication is done using non-blocking two-sided communication, while the coarray library is implemented using a one-sided MPI or OpenSHMEM communication backend. We examine the development cost in addition to strong and weak scalability analysis to understand the performance costs.

1 INTRODUCTION

1.1 Motivation and Background

In high performance computing MPI has been the de facto method for memory communication across a system's nodes for many years. MPI 1.0 was released in 1994 and research and development has continued across academia and industry. A method in Fortran 2008, known as coarray Fortran, was introduced to express the communication within the language [5]. This work was based on an extension to Fortran that was introduced by Robert W. Numrich and John Reid in 1998 [7]. Coarray Fortran, like MPI, is a single-program, multiple-data (SPMD) programming technique. Coarray Fortran's single program is replicated across multiple processes,

(c) 400 points per process

(d) Cray weak scaling

Figure 3: (a-c) Weak scaling results for 25, 100, and 400 points per process (d) weak scaling for Cray.

Application:

- Intermediate Complexity Atmospheric Research (ICAR) model
- Regional impacts of global climate change

Programming models studied:

- CAF over one-sided MPI
- CAF over OpenSHMEM
- Two-sided MPI
- Cray CAF

Highlights:

- "… we used up to 25,600 processes and found that at every data point OpenSHMEM was outperforming MPI."
- "The coarray Fortran with MPI backend stopped being usable as we went over 2,000 processes… the initialization time started to increase exponentially."

Rasmussen, S., Gutmann, E. D., Friesen, B., Rouson, D., Filippone, S., & Moulitsas, I. (2018). Development and performance comparison of MPI and Fortran Coarrays within an atmospheric research model. *Parallel Applications Workshop - Alternatives to MPI+x (PAW-ATM)*, Dallas, Texas, USA.

Coming Soon to a Computer Screen Near You

- Fortran 2023
  - Reductions in `do concurrent`
  - Notified access for remote coarray data
- Fortran 202Y (Y ~ 8)
  - Type-safe generic programming
  - Task-based parallel programming