BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY
Office of Science

Coarray Fortran Tutorial

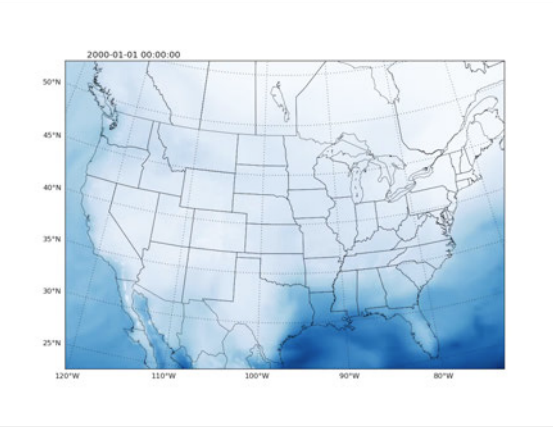
Damian Rouson
Computer Languages & System Software

Hosted by ECP, NERSC, and OLCF, 26-27 July 2023

Day 1

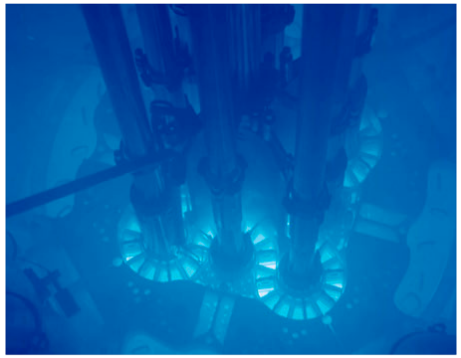
- Introduction to Coarray Fortran (“CAF”)
  - — Why Fortran Matters
  - — SPMD parallel execution
  - — PGAS data structures & RMA
- Heat Conduction Solver
  - — Compiling and running it
  - — Understanding it

Why Fortran Matters



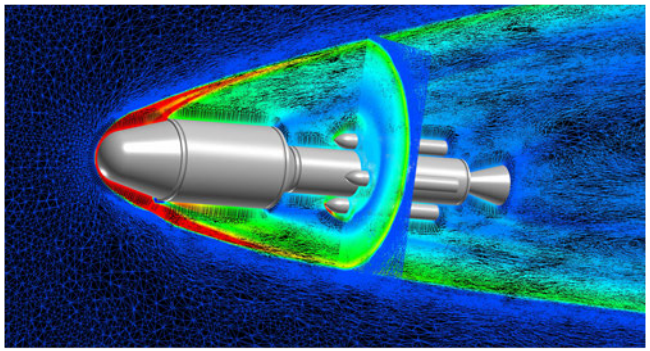
Intermediate Complexity Atmospheric Research (ICAR) Model
Courtesy of Ethan Gutmann, NCAR

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.

<u>Seminal paper:</u>

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

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

Compiling and Running `hi.f90`

BERKELEY LAB
Bringing Science Solutions to the World

rouson — -zsh — 64×19

```
cuf23-tutorial:
```

SPMD Execution Sequence

BERKELEY LAB
Bringing Science Solutions to the World

Time

Image 1

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

Image 2

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

```
print *,"Hello from image ", this_image(), "of", num_images()
```

```
print *,"Hello from image ", this_image(), "of", num_images()
```

```
end program
```

```
end program
```

Image control statement

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

```fortran
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
```

Compiling & Running `hello.f90`

BERKELEY LAB
Bringing Science Solutions to the World

cuf23-tutorial — -zsh — 66×15

```
cuf23-tutorial:
```

Compiling and Running the Heat Equation Solver

BERKELEY LAB
Bringing Science Solutions to the World

cuf23-tutorial — -zsh — 78×23

```
cuf23-tutorial:
```

Heat Equation

```
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

Class Diagram

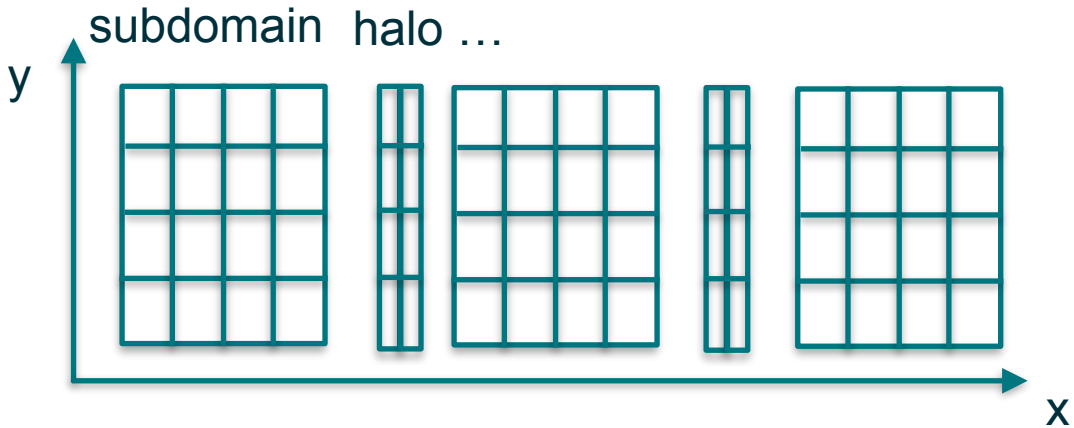
BERKELEY LAB
Bringing Science Solutions to the World

C subdomain_2D_t

s_ : real[]

define()
laplacian(rhs: subdomain_2D_t) : subdomain_2D_t
multiply(lhs : subdomain_2D_t, rhs : subdomain_2D_t) : subdomain_2D_t
add(lhs : subdomain_2D_t, rhs : subdomain_2D_t) : subdomain_2D_t
copy(lhs : subdomain_2D_t, rhs : subdomain_2D_t)
dx()
dy()
values()
exchange_halo()
allocate_halo_coarray()

Halo Exchange



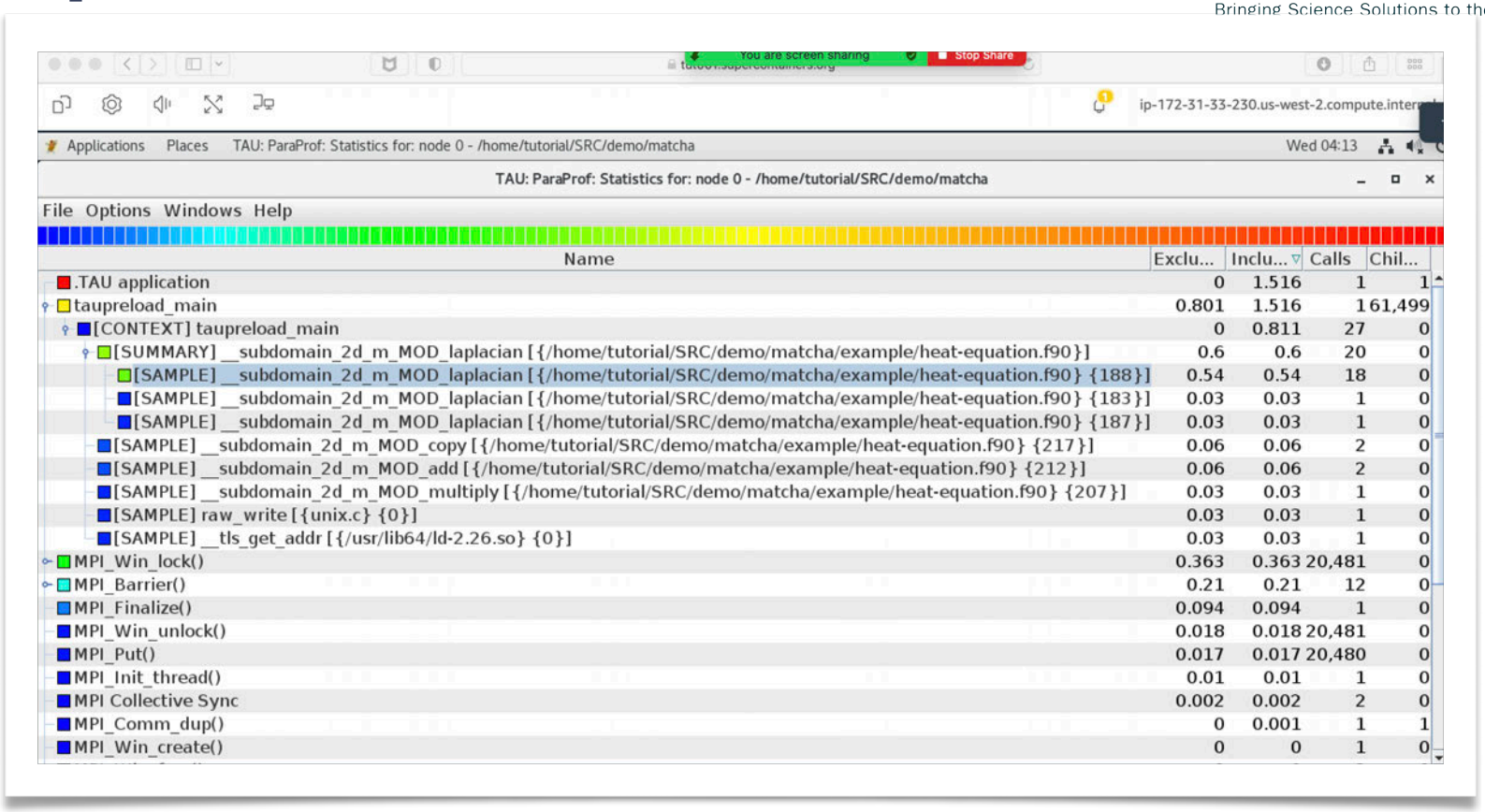

```
real(rkind), allocatable :: halo_x(:,:)[:]
integer, parameter :: west=1, east=2

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



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
[SAMPLE] raw_write [{unix.c} {0}]	0.03	0.03	1	0
[SAMPLE] __tls_get_addr [{/usr/lib64/ld-2.26.so} {0}]	0.03	0.03	1	0
MPI_Win_lock()	0.363	0.363	20,481	0
MPI_Barrier()	0.21	0.21	12	0
MPI_Finalize()	0.094	0.094	1	0
MPI_Win_unlock()	0.018	0.018	20,481	0
MPI_Put()	0.017	0.017	20,480	0
MPI_Init_thread()	0.01	0.01	1	0
MPI Collective Sync	0.002	0.002	2	0
MPI_Comm_dup()	0	0.001	1	1
MPI_Win_create()	0	0	1	0

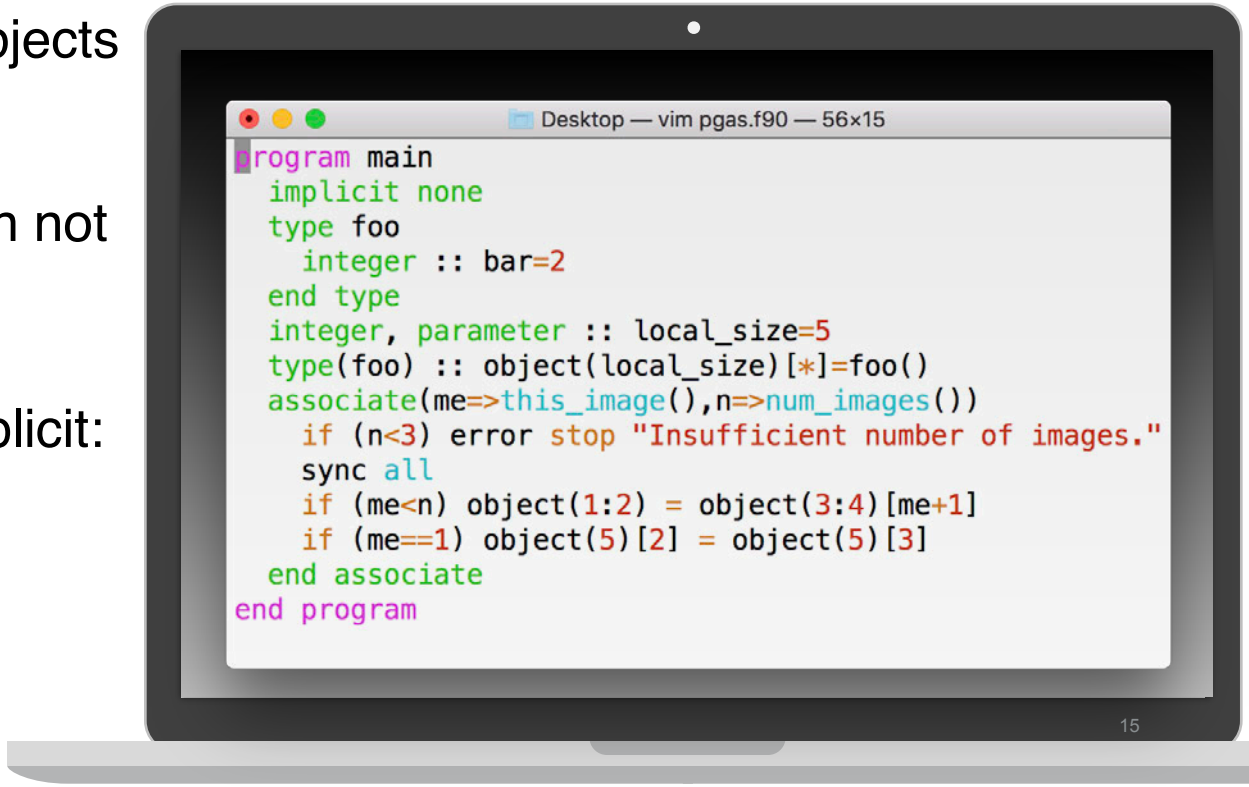
line continuation

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

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



BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY
Office of Science

Acknowledgements

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

Dan Bonachea, Jeremiah Bailey, Tobias Burnus, Alessandro Fanfarillo, Daniel Ceils Garza, Ethan Gutmann, Jeff Hammond, Paul Hargrove, Peter Hill, Dominick Martinez, Tan Nguyen, Katherine Rasmussen, Soren Rasmussen, Brad Richardson, Sameer Shende, Robert Singleterry, Harris Snyder, 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.

Day 2

- CAF at Scale
- Teams
- Image enumeration
- Synchronization
- Collective Subroutines
- Coarrays
- Events

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

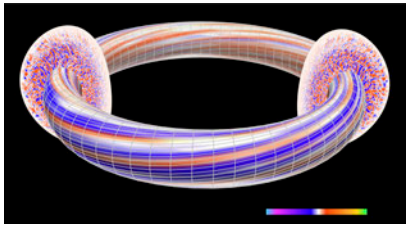


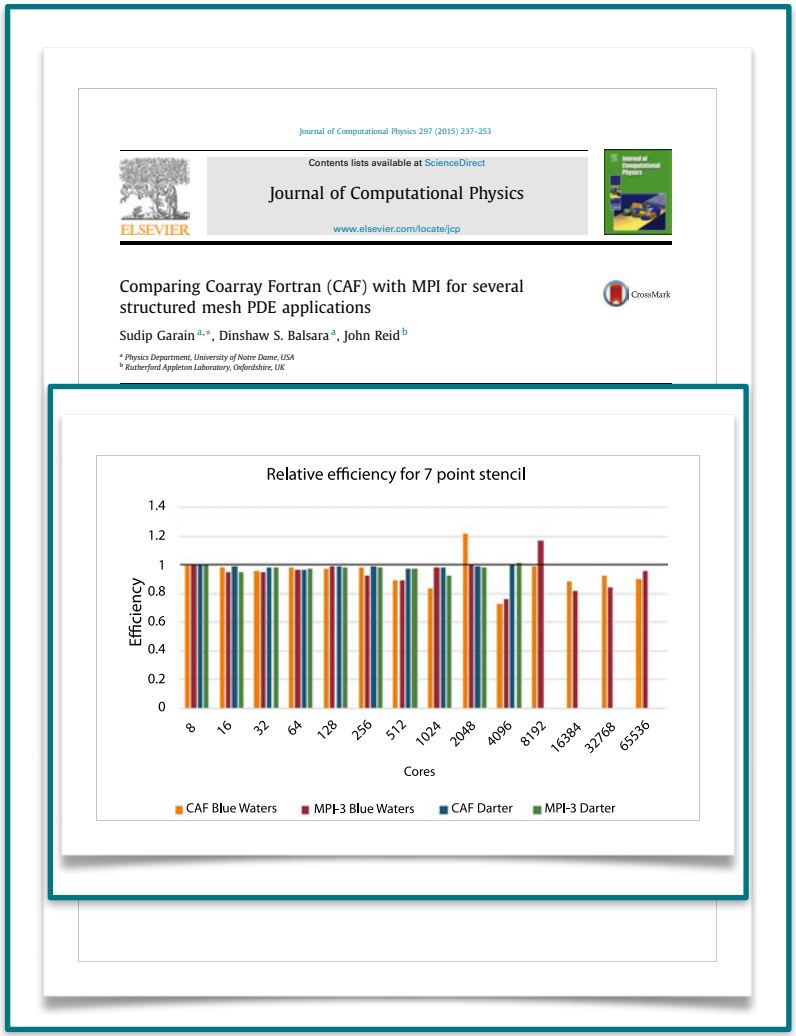
Figure 2: GTS field-line following grid & toroidal domain decomposition. Colors represent isocontours of the quasi-two-dimensional electrostatic potential

- 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



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.

- 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…"

CAF at Scale: Weather

BERKELEY LAB
Bringing Science Solutions to the World

Article

International Journal of HIGH PERFORMANCE COMPUTING APPLICATIONS

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
2015, Vol. 29(3) 261–273

DOI: 10.1177/1094342015576773
hpc.sagepub.com
SAGE

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

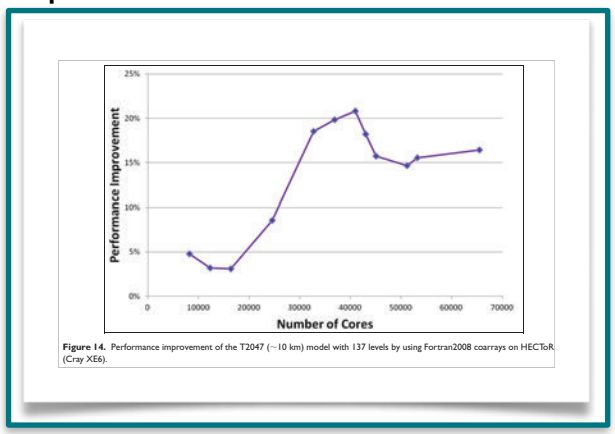


Figure 14. Performance improvement of the T2047 (~10 km) model with 137 levels by using Fortran2008 coarrays on HECToR (Cray XE6).

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.

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.

Application:

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

Programming models studied:

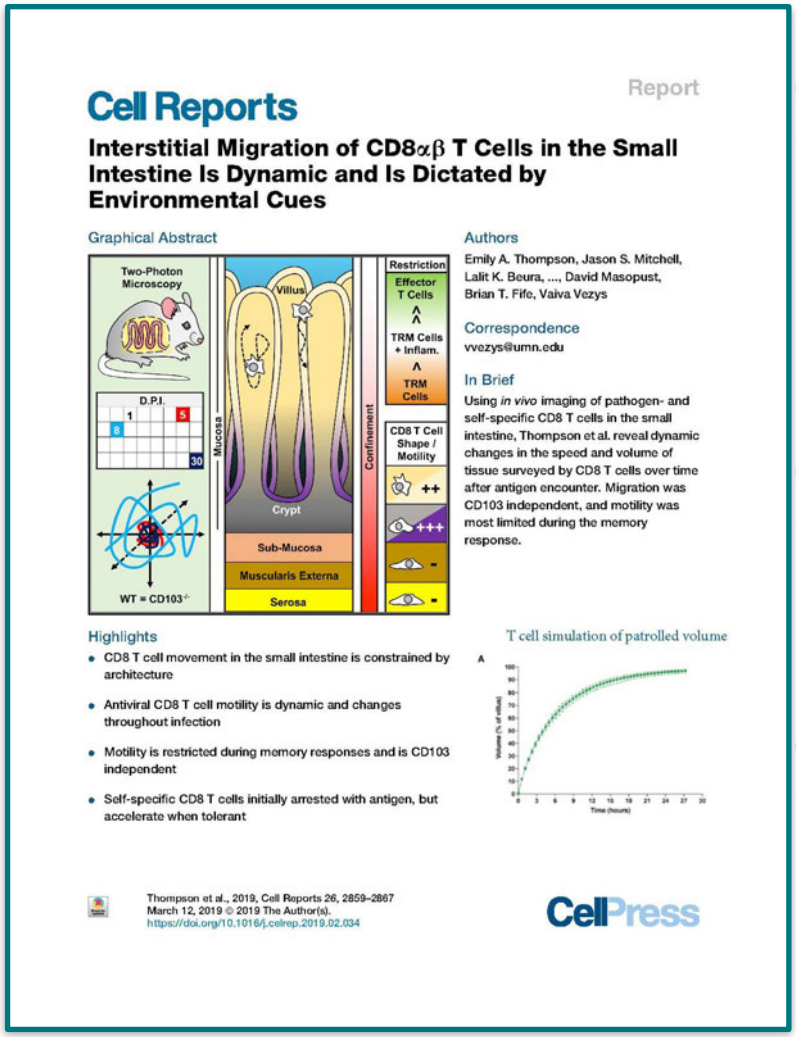
- CAF over one-sided MPI
- CAF over OpenSHMEM
- Two-sided MPI
- Cray CAF

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."

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

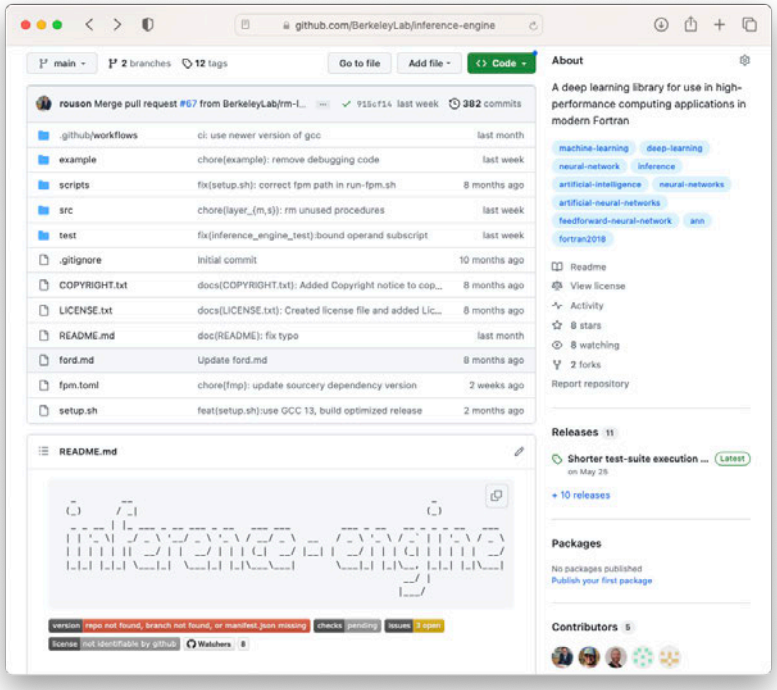
Thompson, E. A., Mitchell, J. S., Beura, L. K., Torres, D. J., Mrass, P., Pierson, M. J., ... & Vezys, V. (2019). Interstitial migration of CD8αβ T cells in the small intestine is dynamic and is dictated by environmental cues. *Cell reports*, *26*(11), 2859-2867.

- 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

New Frontiers: Deep Learning

BERKELEY LAB
Bringing Science Solutions to the World



https://go.lbl.gov/inference-engine

- Application:
  - Inference-Engine
  - *In situ* neural network training and large-batch inference for HPC applications
- Language-based parallel & GPU programming:
  - Extensive use of array statements, `elemental` procedures, `do concurrent`
  - Functional programming pattern:

    Every procedure is `pure` except those that create and consume JSON file objects.
  - *Coming soon:*

    Parallel mini-batch training via `co_sum`

Implicitly Parallel Training

BERKELEY LAB
Bringing Science Solutions to the World

inference-engine — vim src/inference_engine/trainable_engine_s.f90 — 132×55

```
136       w = 0.; b = 0.e0 ! Initialize weights and biases
137
138       iterate_across_batches: &
139       do iter = 1, size(mini_batches)
140
141         cost = 0.; dcdw = 0.; dcdb = 0.
142
143         associate(input_output_pairs => mini_batches(iter)%input_output_pairs())
144           inputs = input_output_pairs%inputs()
145           expected_outputs = input_output_pairs%expected_outputs()
146           mini_batch_size = size(input_output_pairs)
147         end associate
148
149         iterate_through_batch: &
150         do pair = 1, mini_batch_size
151
152           a(1:num_inputs,0) = inputs(pair)%values()
153           y = expected_outputs(pair)%outputs()
154
155           feed_forward: &
156           do l = 1,output_layer
157             z(1:n(l),l) = matmul(w(1:n(l),1:n(l-1),l), a(1:n(l-1),l-1)) + b(1:n(l),l)
158             a(1:n(l),l) = self%differentiable_activation_strategy_%activation(z(1:n(l),l))
159           end do feed_forward
160
161           cost = cost + sum((y(1:n(output_layer))-a(1:n(output_layer),output_layer))**2)/(2.e0*mini_batch_size)
162
163           delta(1:n(output_layer),output_layer) = &
164             (a(1:n(output_layer),output_layer) - y(1:n(output_layer))) &
165             * self%differentiable_activation_strategy_%activation_derivative(z(1:n(output_layer),output_layer))
166
167           back_propagate_error: &
168           do l = n_hidden,1,-1
169             delta(1:n(l),l) = matmul(transpose(w(1:n(l+1),1:n(l),l+1)), delta(1:n(l+1),l+1))
170             delta(1:n(l),l) = delta(1:n(l),l) * self%differentiable_activation_strategy_%activation_derivative(z(1:n(l),l))
171           end do back_propagate_error
172
173           sum_gradients: &
174           do l = 1,output_layer
175             dcdb(1:n(l),l) = dcdb(1:n(l),l) + delta(1:n(l),l)
176             do concurrent(j = 1:n(l))
177               dcdw(j,1:n(l-1),l) = dcdw(j,1:n(l-1),l) + a(1:n(l-1),l-1)*delta(j,l)
178             end do
179           end do sum_gradients
180         end do iterate_through_batch
181
182         adjust_weights_and_biases: &
183         do l = 1,output_layer
184           dcdb(1:n(l),l) = dcdb(1:n(l),l)/mini_batch_size
185           b(1:n(l),l) = b(1:n(l),l) - eta*dcdb(1:n(l),l) ! Adjust biases
186           dcdw(1:n(l),1:n(l-1),l) = dcdw(1:n(l),1:n(l-1),l)/mini_batch_size
187           w(1:n(l),1:n(l-1),l) = w(1:n(l),1:n(l-1),l) - eta*dcdw(1:n(l),1:n(l-1),l) ! Adjust weights
188         end do adjust_weights_and_biases
189       end do iterate_across_batches
                                                                                    136,9          72%
```

“Loop” Structure

BERKELEY LAB
Bringing Science Solutions to the World

inference-engine — vim src/inference_engine/trainable_engine_s.f90 — 132×55

```fortran
136         w = 0.; b = 0.e0 ! Initialize weights and biases
137
138         iterate_across_batches: &
139         do iter = 1, size(mini_batches
140
141           cost = 0.; dcdw = 0.; dcdb =
142
143           associate(input_output_pairs
144             inputs = input_output_pairs
145             expected_outputs = input_ou
146             mini_batch_size = size(inpu
147           end associate
148
149           iterate_through_batch: &
150           do pair = 1, mini_batch_size
151
152             a(1:num_inputs) = inputs
153             y = expected_outputs(pair)
154
155             feed_forward: &
156             do l = 1,output_layer
157               z(1:n(l),l) = matmul(w(1
158               a(1:n(l),l) = self%diffe
159             end do feed_forward
160
161             cost = cost + sum((y(1:n(
162
163             delta(1:n(output_layer),ou
164               (a(1:n(output_layer),out
165               * self%differentiable_ac
166
167             back_propagate_error: &
168             do l = n_hidden,1,-1
169               delta(1:n(l),l) = matmul(transpose(
170               delta(1:n(l),l) = delta(1:n(l),l) *
171             end do back_propagate_error
172
173             sum_gradients: &
174             do l = 1,output_layer
175               dcdb(1:n(l),l) = dcdb(1:n(l),l) + c
176               do concurrent(j = 1:n(l))
177                 dcdw(j,1:n(l-1),l) = dcdw(j,1:n(l
178               end do
179             end do sum_gradients
180           end do iterate_through_batch
181
182           adjust_weights_and_biases: &
183           do l = 1,output_layer
184             dcdb(1:n(l),l) = dcdb(1:n(l),l)/mini_batch_size
185             b(1:n(l),l) = b(1:n(l),l) - eta*dcdb(1:n(l),l) ! Adjust biases
186             dcdw(1:n(l),1:n(l-1),l) = dcdw(1:n(l),1:n(l-1),l)/mini_batch_size
187             w(1:n(l),1:n(l-1),l) = w(1:n(l),1:n(l-1),l) - eta*dcdw(1:n(l),1:n(l-1),l) ! Adjust weights
188           end do adjust_weights_and_biases
189         end do iterate_across_batches
                                                                    136,9          72%
```

```fortran
136         w = 0.; b = 0.e0 ! Initialize weights and biases
137
138         iterate_across_batches: &
139         do iter = 1, size(mini_batches)
140
141           cost = 0.; dcdw = 0.; dcdb = 0.
142
143           associate(input_output_pairs => mini_batches(iter)%input_output_pairs())
144             inputs = input_output_pairs%inputs()
145             expected_outputs = input_output_pairs%expected_outputs()
146             mini_batch_size = size(input_output_pairs)
147           end associate
148
149           iterate_through_batch: &
150           do pair = 1, mini_batch_size
```

Iterating sequentially across and within mini-batches of input/output pairs facilitates *in situ* training at application runtime, potentially eliminating the export of large training data sets or at least making it so that the resulting network can be trained off-line in fewer iterations.

“Loop” Structure

BERKELEY LAB
Bringing Science Solutions to the World

inference-engine — vim src/inference_engine/trainable_engine_s.f90 — 132×55

```
136       w = 0.; b = 0.e0 ! Initialize weights and biases
137
138       iterate_across_batches: &
139       do iter = 1, size(mini_batches)
140
141         cost = 0.; dcdw = 0.; dcdb = 0.
142
143         associate(input_output_pairs => mini_batches(iter)%input_output_pairs())
144           inputs = input_output_pairs%inputs()
145           expected_outputs = input_output_pairs%expected_outputs()
146           mini_batch_size = size(input_output_pairs)
147         end associate
148
149         iterate_through_batch: &
150         do pair = 1, mini_batch_size
151
152           a(1:num_inputs,0) = inputs(pair)%values()
153           y = expected_outputs(pair)%outputs()
154
155           feed_forward: &
156           do l = 1,output_layer
157             z(1:n(l),l) = matmul(w(1:n(l),1:n(l-1),l), a(1:n(l-1),l-1)) + b(1:n(l),l)
158             a(1:n(l),l) = self%differentiable_activation_strategy_%activation(z(1:n(l),l))
159
160
161                                                                          **2)/(2.e0*m
162
163
164
165                                                                          utput_layer)
166
167
168
169             delta(1:n(l),l) = matmul(transpose(w(1:n(l+1),1:n(l),l+1)), delta(1:n(l+1),l+1))
170             delta(1:n(l),l) = delta(
171           end do back_propagate_erro
172
173           sum_gradients: &
174           do l = 1,output_layer
175             dcdb(1:n(l),l) = dcdb(1:
176             do concurrent(j = 1:n(l)
177               dcdw(j,1:n(l-1),l) = c
178             end do
179           end do sum_gradients
180         end do iterate_through_batch
181
182         adjust_weights_and_biases: &
183         do l = 1,output_layer
184           dcdb(1:n(l),l) = dcdb(1:n(
185           b(1:n(l),l) = b(1:n(l),l)
186           dcdw(1:n(l),1:n(l-1),l) = dcdw(1:n(l),1:n(l-1),l)/mini_batch_size
187           w(1:n(l),1:n(l-1),l) = w(1:n(l),1:n(l-1),l) - eta*dcdw(1:n(l),1:n(l-1),l) ! Adjust weights
188         end do adjust_weights_and_biases
189       end do iterate_across_batches
                                                                    136,9          72%
```

The only other sequential logic is the (mostly) necessary stepping through layers:

All other logic is implicitly parallel array statements or `do concurrent` blocks:

```
173           sum_gradients: &
174           do l = 1,output_layer
175             dcdb(1:n(l),l) = dcdb(1:n(l),l) + delta(1:n(l),l)
176             do concurrent(j = 1:n(l))
177               dcdw(j,1:n(l-1),l) = dcdw(j,1:n(l-1),l) + a(1:n(l-1),l-1)*delta(j,l)
178             end do
179           end do sum_gradients
180         end do iterate_through_batch
```

Fast-GPT

BERKELEY LAB
Bringing Science Solutions to the World

ondrejcertik.com/blog/2023/03/fastgpt-faster-than-

Ondřej Čertík

FASTGPT: FASTER THAN PYTORCH IN 300 LINES OF FORTRAN

March 14, 2023

Authors: Ondřej Čertík, Brian Beckman

In this blog post I am announcing fastGPT, fast GPT-2 inference written in Fortran. In it, I show

1. Fortran has speed at least as good as default `PyTorch` on Apple M1 Max.
2. Fortran code has statically typed arrays, making maintenance of the code easier than with Python
3. It seems that the bottleneck algorithm in GPT-2 inference is matrix-matrix multiplication. For physicists like us, matrix-matrix multiplication is very familiar, unlike other aspects of AI and ML. Finding this familiar ground inspired us to approach GPT-2 like any other numerical computing problem.
4. Fixed an unintentional single-to-double conversion that slowed down the original Python.
5. I am asking others to take over and parallelize `fastGPT` on CPU and offload to GPU and see how fast you can make it.

About one month ago, I read the blogpost GPT in 60 Lines of NumPy, and it piqued my curiosity. I looked at the corresponding code (picoGPT) and was absolutely amazed, for two reasons. First, I hadn't known it could be so simple to implement the GPT-2 inference. Second, this looks just like a typical computational physics code, similar to many that I have developed and maintained throughout my career.

https://tinyurl.com/fastgpt-by-certik

Teams

BERKELEY LAB
Bringing Science Solutions to the World

An ordered set of images created by execution of a **`form team`** statement, or the initial ordered set of all images.

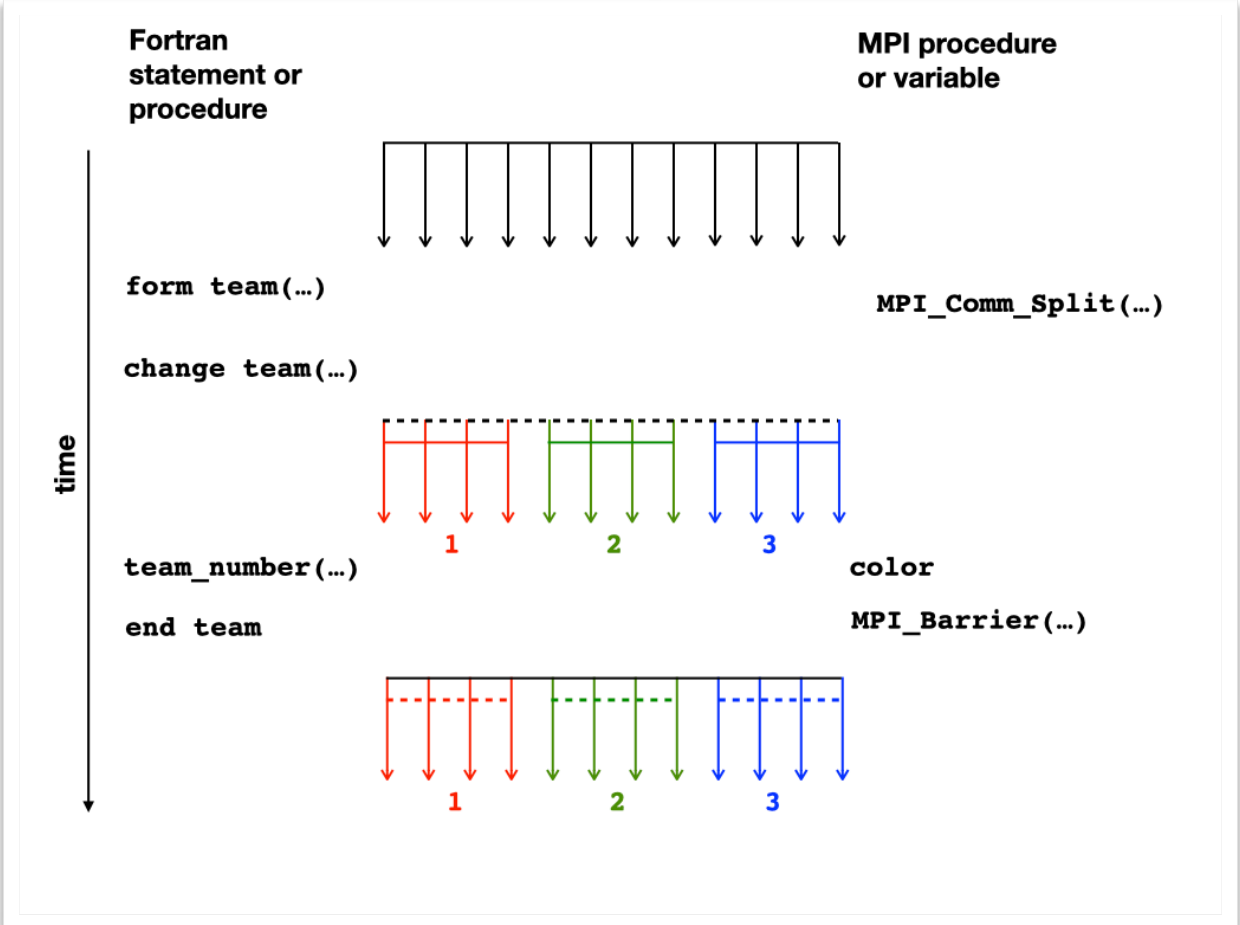
Teams facilitate the execution of an image sets independently from other image sets, e.g., a **`sync all`** statement synchronizes the current team only.

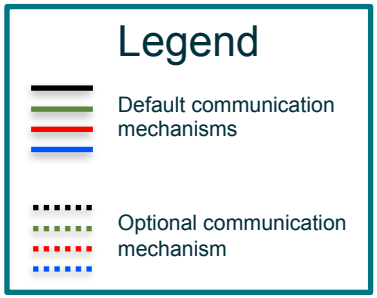
An extensible derived type `team_type` with private components describes a team after the successful execution of a **`form team`** statement.

CAF/MPI Rosetta Stone

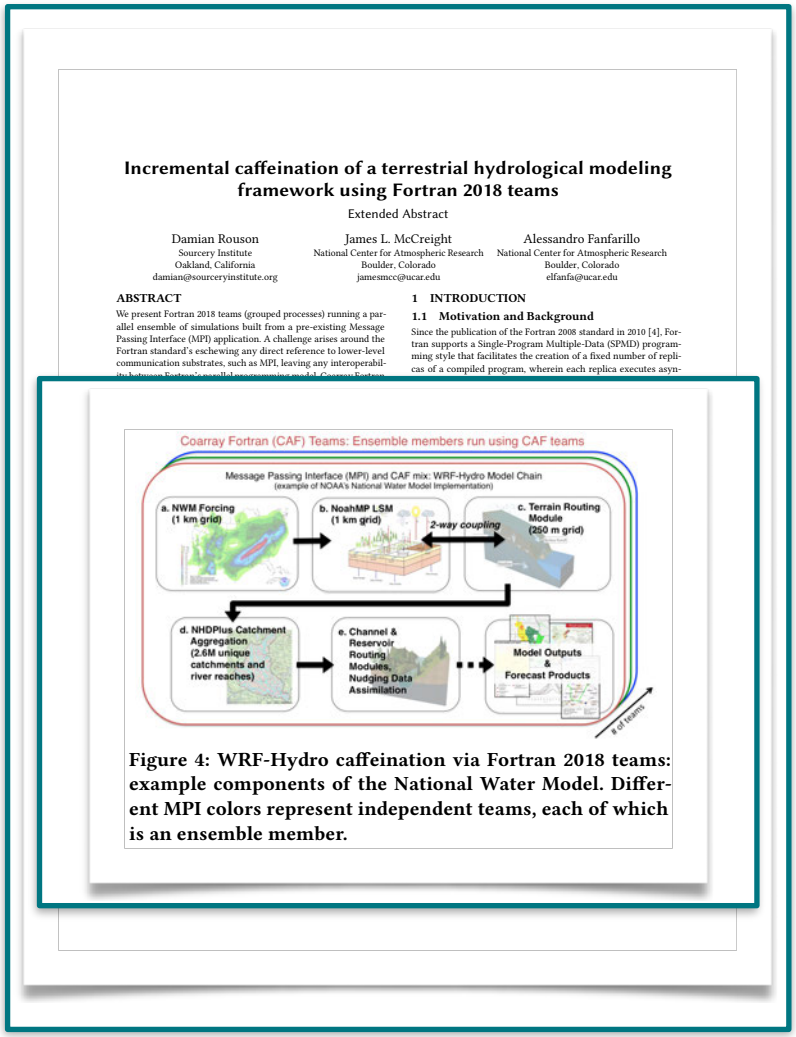
BERKELEY LAB
Bringing Science Solutions to the World

Program execution sequence over time (left axis) in 12 images (top) initially globally and then within subgroups.





Teams Test Code



Incremental caffeination of a terrestrial hydrological modeling framework using Fortran 2018 teams

Extended Abstract

Damian Rouson, Sourcery Institute, Oakland, California, damian@sourceryinstitute.org

James L. McCreight, National Center for Atmospheric Research, Boulder, Colorado, jamesmcc@ucar.edu

Alessandro Fanfarillo, National Center for Atmospheric Research, Boulder, Colorado, elfanfa@ucar.edu

ABSTRACT

We present Fortran 2018 teams (grouped processes) running a parallel ensemble of simulations built from a pre-existing Message Passing Interface (MPI) application. A challenge arises around the Fortran standard's eschewing any direct reference to lower-level communication substrates, such as MPI, leaving any interoperability between Fortran's coarray programming model, Coarray Fortran

1 INTRODUCTION

1.1 Motivation and Background

Since the publication of the Fortran 2008 standard in 2010 [4], Fortran supports a Single-Program Multiple-Data (SPMD) programming style that facilitates the creation of a fixed number of replicas of a compiled program, wherein each replica executes asyn-

Figure 4: WRF-Hydro caffeination via Fortran 2018 teams: example components of the National Water Model. Different MPI colors represent independent teams, each of which is an ensemble member.

```fortran
program main
  !! Test team_number intrinsic function
  use iso_fortran_env, only : team_type
  use assertions_module , only : assertions

  implicit none

  integer , parameter :: standard_initial_value = -1
  type(team_type), target :: home

  call assert(team_number() == standard_initial_value)

  associate(my_team=>mod(this_image(),2) + 1)

    form team(my_team,home) ! Map even|odd images->teams 1|2
    change team(home)
      call assert(team_number() == my_team)
    end team

    call assert(team_number() == standard_initial_value)

  end associate

  sync all

  if (this_image() == 1) print *, "Test passed."

end program
```


Rouson, D., McCreight, J. L., & Fanfarillo, A. (2017, November). Incremental caffeination of a terrestrial hydrological modeling framework using Fortran 2018 teams. In *Proceedings of the Second Annual PGAS Applications Workshop* (pp. 1-4).

Image Enumeration

BERKELEY LAB
Bringing Science Solutions to the World

- Obtaining an image index:

```
this_image([team])
this_image(coarray [,team])
this_image(coarray, dim [,team])
```

```
image_index(coarray, sub, team_number)
image_index(coarray, sub, team)
image_index(coarray, sub)
```

- Obtaining an image count:

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

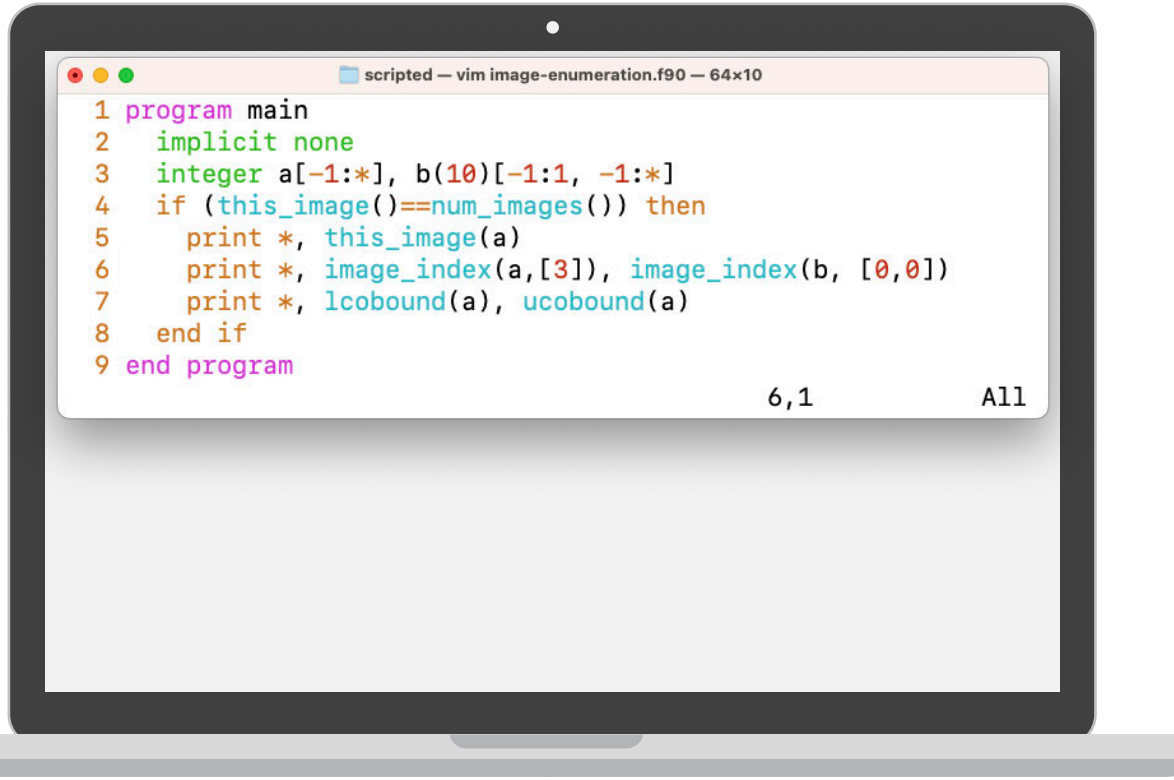


Image Enumeration

BERKELEY LAB
Bringing Science Solutions to the World

	[-1,1]	[0,1]	[1,1]
b(:)	[-1,0]	[0,0]	[1,0]
	[-1,-1]	[0,-1]	[1,-1]

a	[0]	[1]	[2]	[3]	[4]

scripted — vim image-enumeration.f90 — 64×10

```fortran
program main
  implicit none
  integer a[-1:*], b(10)[-1:1, -1:*]
  if (this_image()==num_images()) then
    print *, this_image(a)
    print *, image_index(a,[3]), image_index(b, [0,0])
    print *, lcobound(a), ucobound(a)
  end if
```

All

scripted — -zsh — 64×10

```
cuf23-tutorial: cafrun -n 5 ./image-enumeration
           3
           5           5
          -1           3
cuf23-tutorial: ^5^1
cafrun -n 1 ./image-enumeration
          -1
           0           0
          -1          -1
cuf23-tutorial:
```

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

BERKELEY LAB
Bringing Science Solutions to the World

- 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)

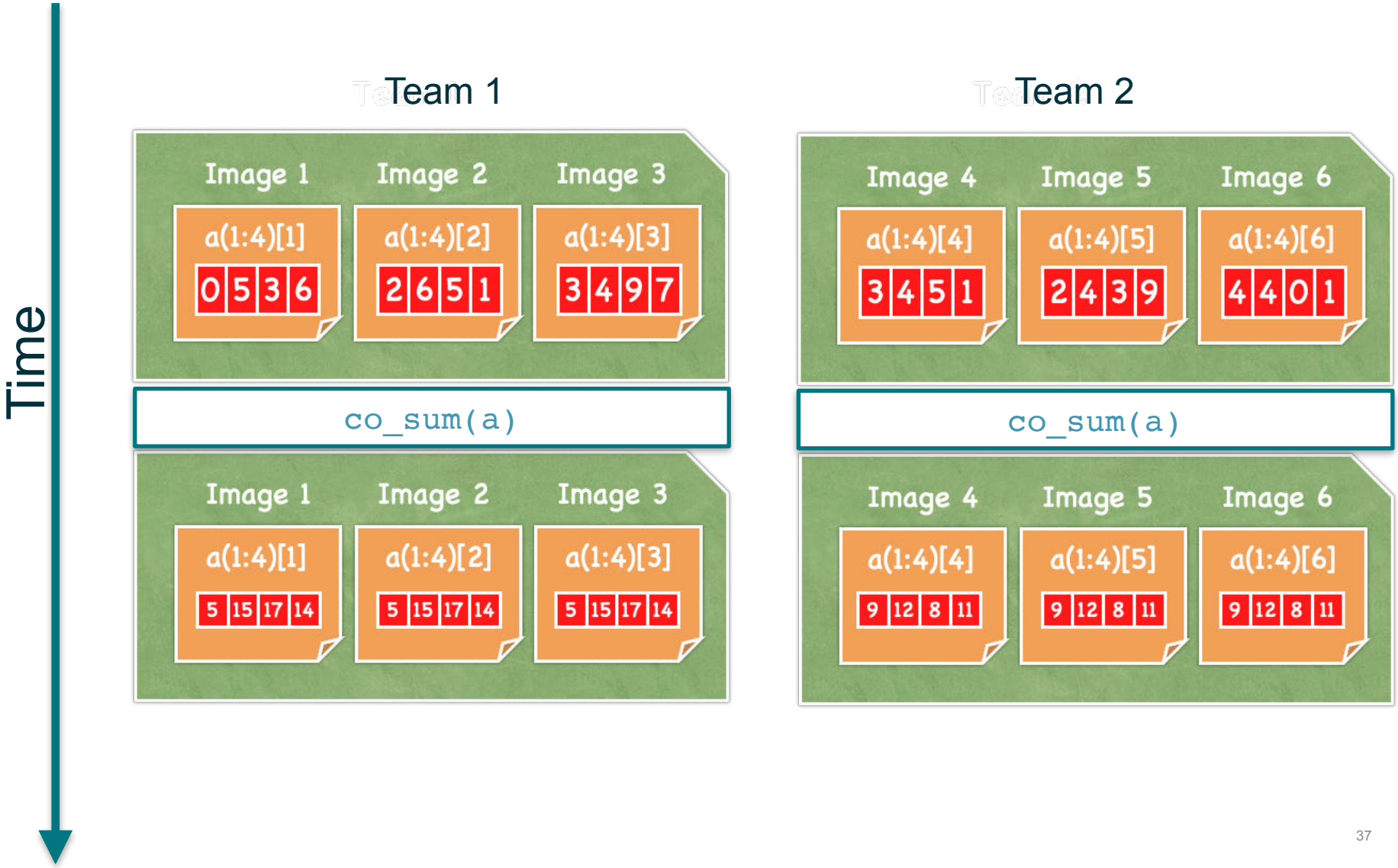
co_sum

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
Team 2
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
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)
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
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

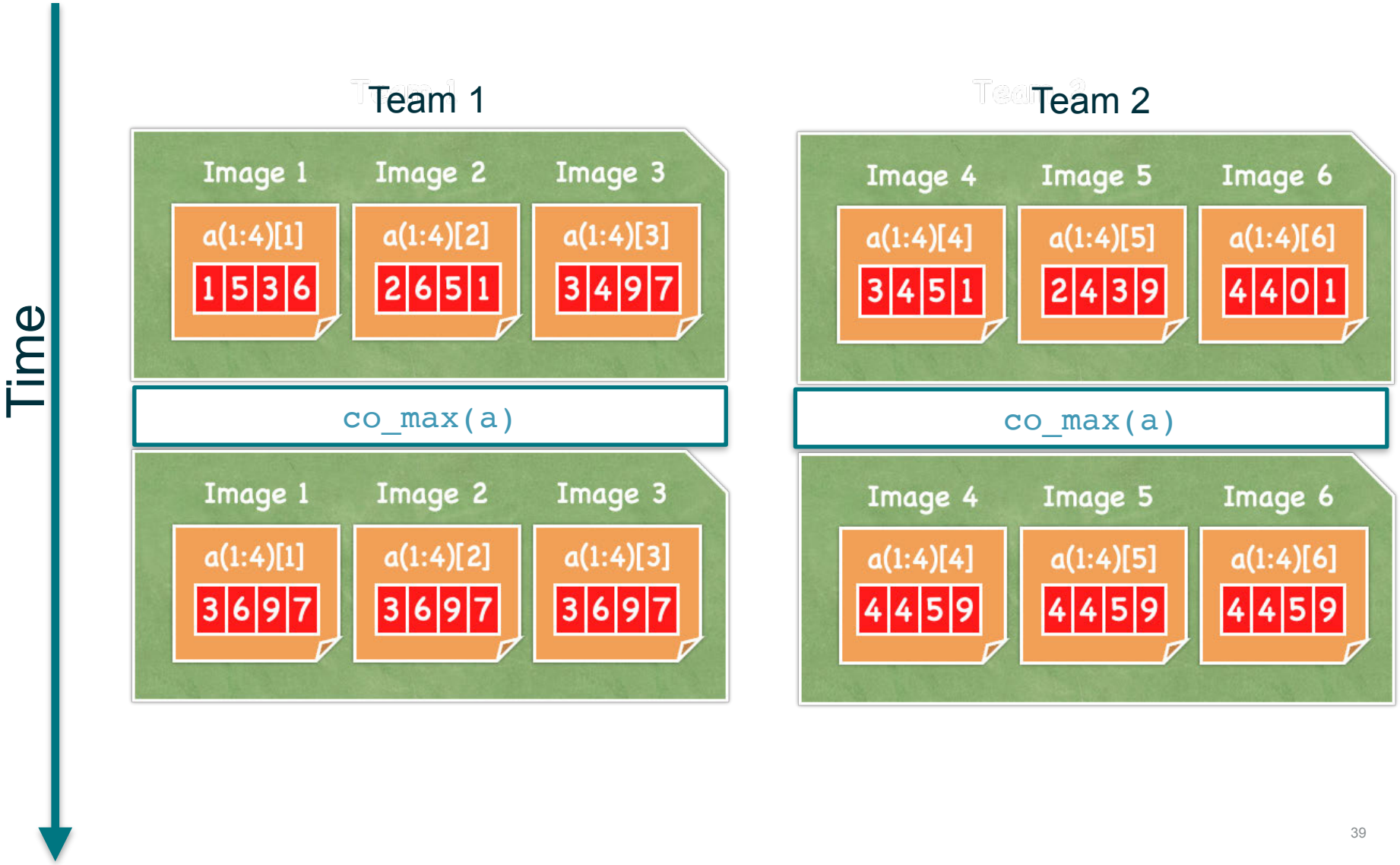
co_max

BERKELEY LAB
Bringing Science Solutions to the World

```
co_max(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_max


Time
Team 1
Team 2
Image 1
Image 2
Image 3
a(1:4)[1]
1536
a(1:4)[2]
2651
a(1:4)[3]
3497
Image 4
Image 5
Image 6
a(1:4)[4]
3451
a(1:4)[5]
2439
a(1:4)[6]
4401
co_max(a)
co_max(a)
Image 1
Image 2
Image 3
a(1:4)[1]
3697
a(1:4)[2]
3697
a(1:4)[3]
3697
Image 4
Image 5
Image 6
a(1:4)[4]
4459
a(1:4)[5]
4459
a(1:4)[6]
4459

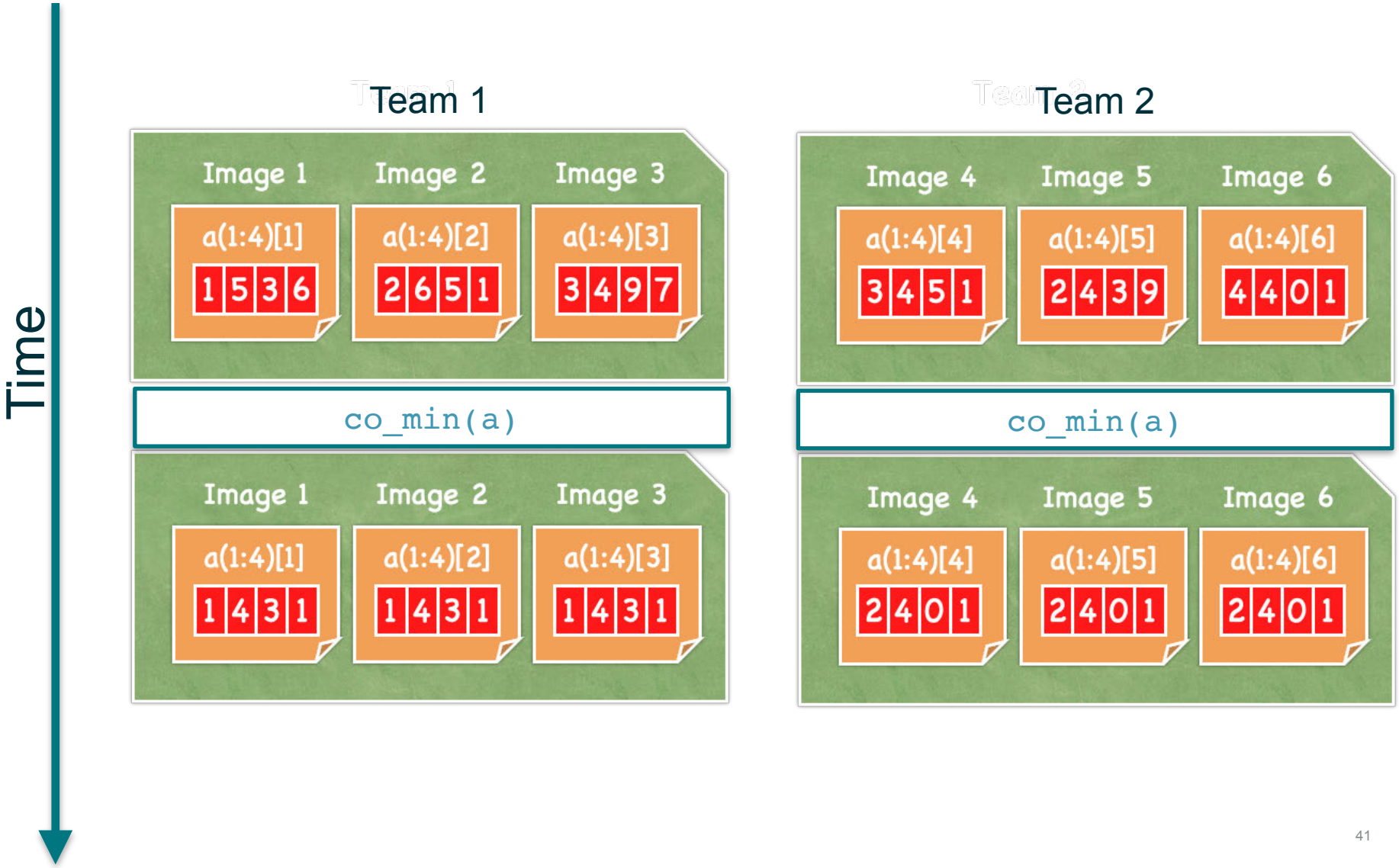
co_min

BERKELEY LAB
Bringing Science Solutions to the World

```
co_min(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_min


Time
Team 1
Image 1
a(1:4)[1]
1536
Image 2
a(1:4)[2]
2651
Image 3
a(1:4)[3]
3497
co_min(a)
Image 1
a(1:4)[1]
1431
Image 2
a(1:4)[2]
1431
Image 3
a(1:4)[3]
1431
Team 2
Image 4
a(1:4)[4]
3451
Image 5
a(1:4)[5]
2439
Image 6
a(1:4)[6]
4401
co_min(a)
Image 4
a(1:4)[4]
2401
Image 5
a(1:4)[5]
2401
Image 6
a(1:4)[6]
2401

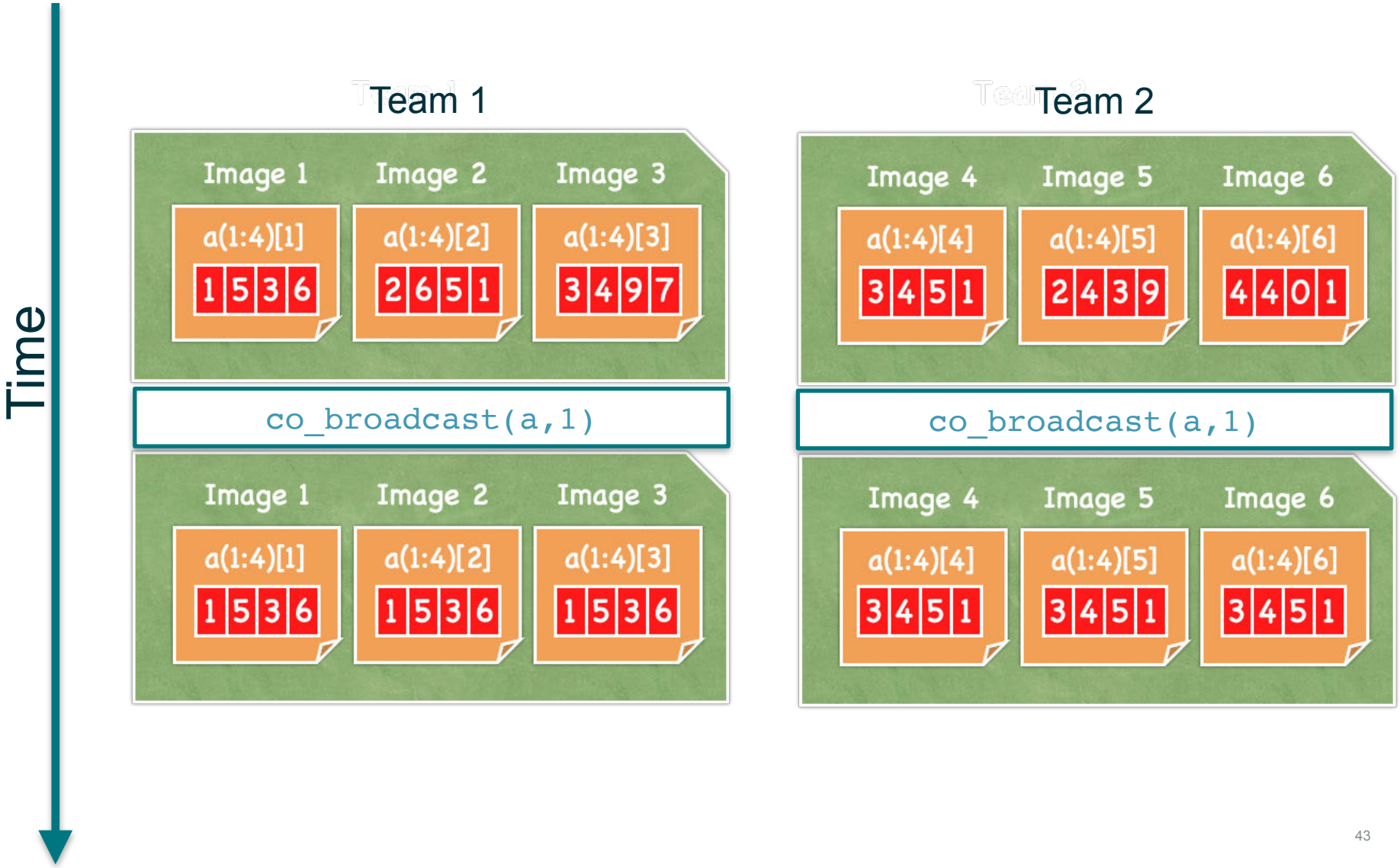
co_broadcast

BERKELEY LAB
Bringing Science Solutions to the World

```
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



co_reduce

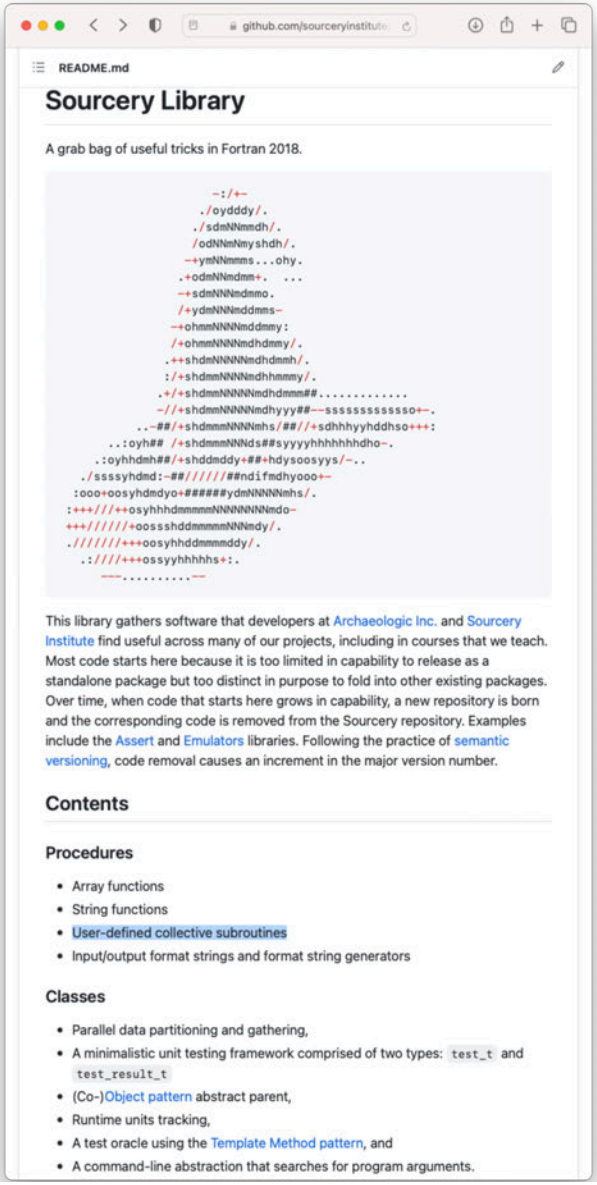
BERKELEY LAB
Bringing Science Solutions to the World

```
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

BERKELEY LAB
Bringing Science Solutions to the World



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
```

Hands-On Heat Equation

BERKELEY LAB
Bringing Science Solutions to the World

github.com/rouson/cuf23-tutorial#heat-equation-exercise

README.md

Heat Equation Exercise

In addition to demonstrating parallel features of Fortran 2018, this example shows an object-oriented, functional programming style based on Fortran's user-defined operators such as the `.laplacian.` operator defined in this example. To demonstrate the expressive power and flexibility of this approach, try modifying the modifying the main program to use 2nd-order Runge-Kutta time advancement:

```
T_half = T + 0.5*dt*alpha* .laplacian. T
call T%exchange_halo
sync all
T = T + dt*alpha* .laplacian. T_half
call T%exchange_halo
sync all
```

You'll need to append `, T_half` to the declaration `type(subdomain_2D_t) T`. With some care, you could modify the main program to use any desired order of Runge-Kutta algorithm without changing any of the supporting code.

This example also demonstrates a benefit of Fortran's facility for declaring a procedure to be `pure`: the semantics of `pure` procedures essentially guarantees that the above right-hand-side expressions can be evaluated fully asynchronously across all images. No operator can modify state that would be observable by another operator other than via the first operator's result. This would be true even if an operator executing on one image performs communication to *get* data from another image via a coarray. To reduce communication waiting times, however, each image in our example proactively *puts* data onto neighboring images. Puts generally outperform gets because the data can be shipped off as soon the data are ready. With the exception of one coarray allocation in the `define` procedure, all procedures are asynchronous and all image control is exposed in the main program.

Coarrays

BERKELEY LAB
Bringing Science Solutions to the World

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

- 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

Abstract Calculus Pattern

User-defined, purely functional operators

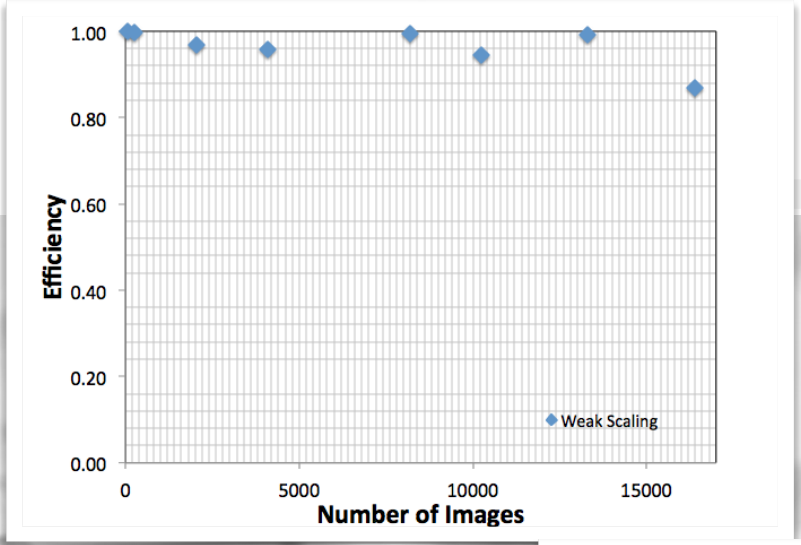
```
u_t = -(.grad.p)/rho + nu*(.laplacian.u) -(u.dot.(.grad.u))
```

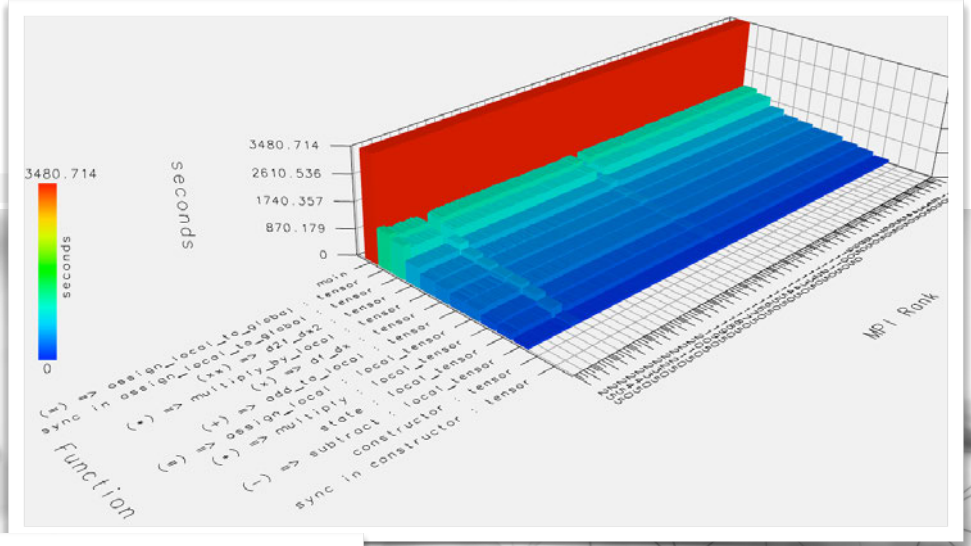
Distributed objects

$$\vec{u}_t = -\frac{1}{\rho}\nabla p + \nu\nabla^2\vec{u} - \vec{u}\cdot\nabla\vec{u}$$

Burgers Eq. Solver

$$u_t = \nu u_{xx} - \left(\frac{u^2}{2}\right)_x$$





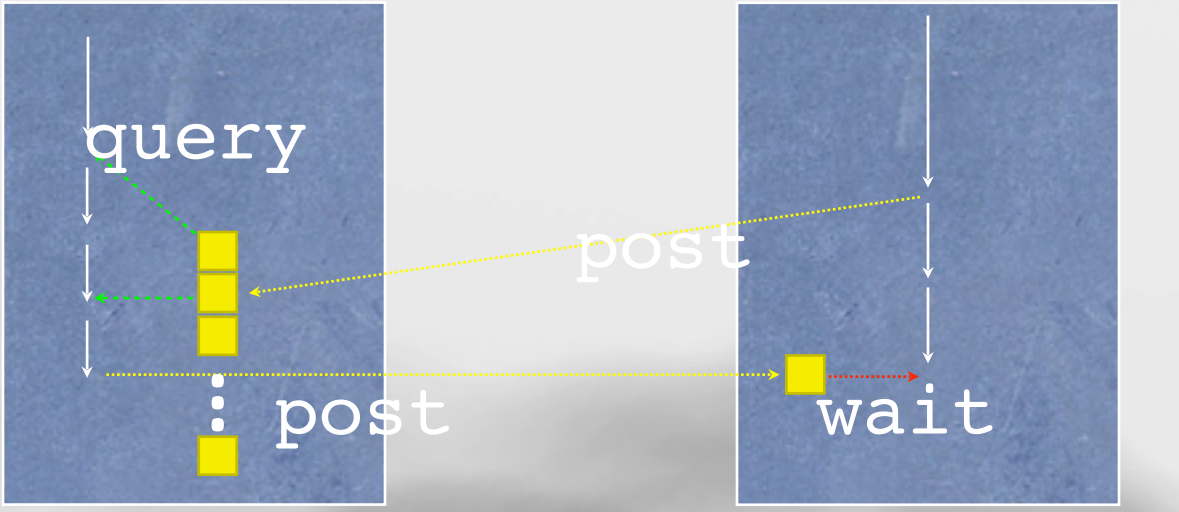
Burgers, J. M. (1948). A mathematical model illustrating the theory of turbulence. *Adv. Appl. Mech.* (1), 25-27.

Platform: Cray XE6 (Hopper at NERSC)

Rouson, Xia, & Xu (2011). *Scientific Software Design: The Object-Oriented Way.* Cambridge University Press.

Events

Hello, world!



greeting_ready(2:n)[1]

ok_to_overwrite[3

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`.

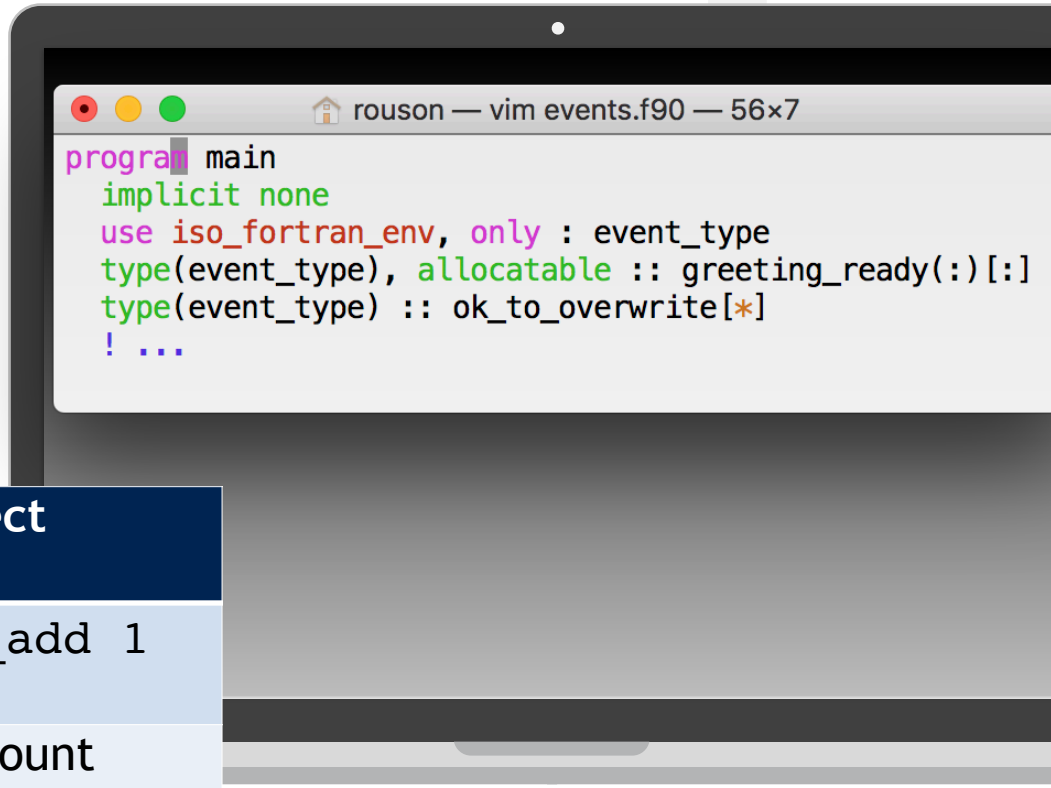
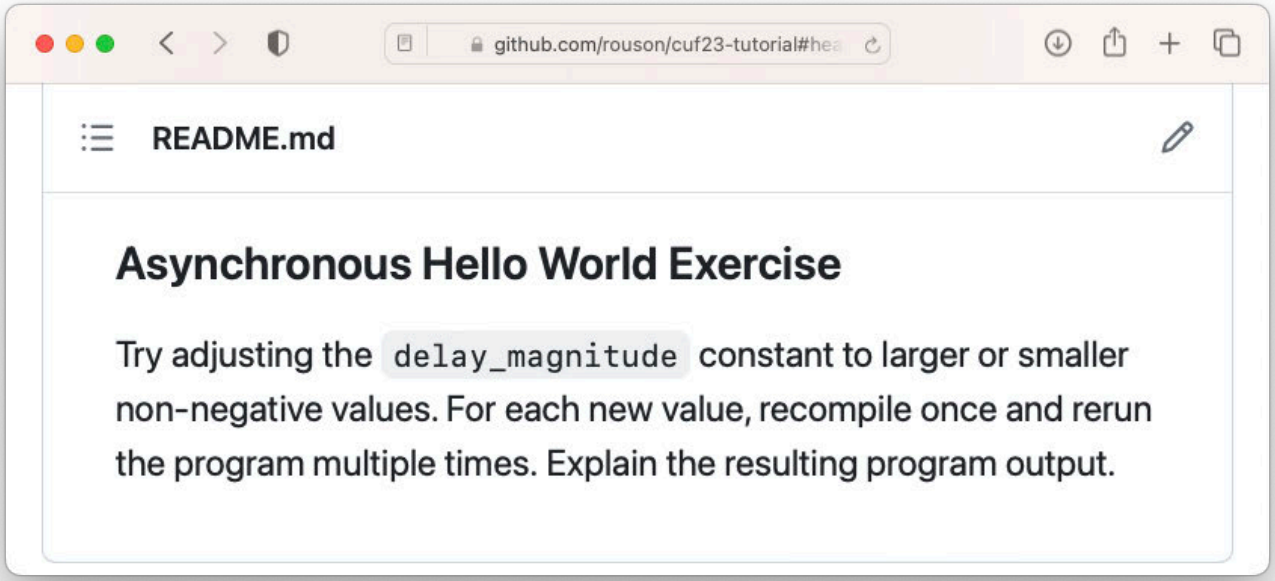


	Image Control	Side Effect
event post	☒	atomic_add 1
event_query		defines count
event wait	☒	atomic_add -1

Hands-On Asynchronous “Hello, World!”

BERKELEY LAB
Bringing Science Solutions to the World



FEATS:

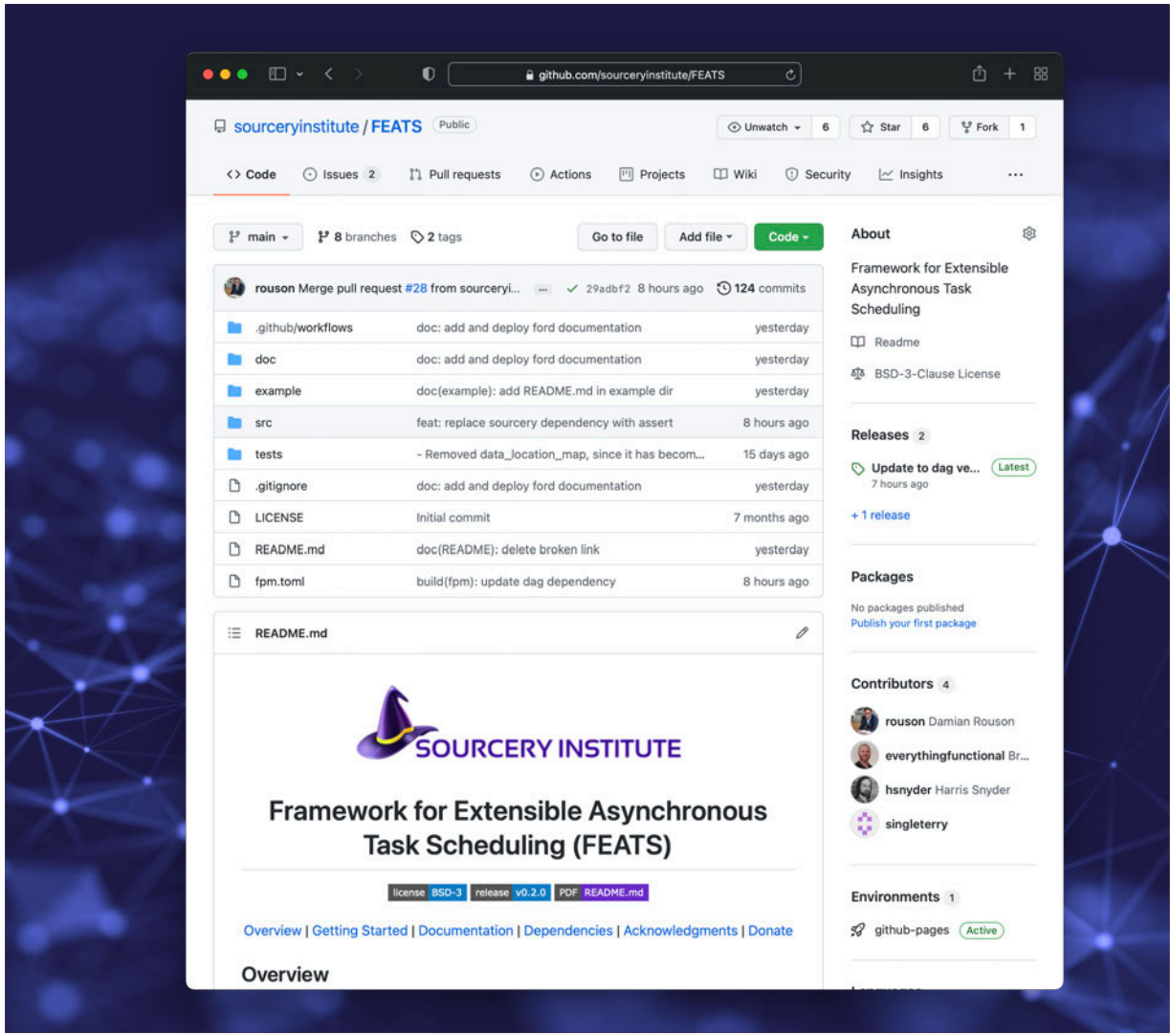
Framework for Extensible Asynchronous Task Scheduling

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:

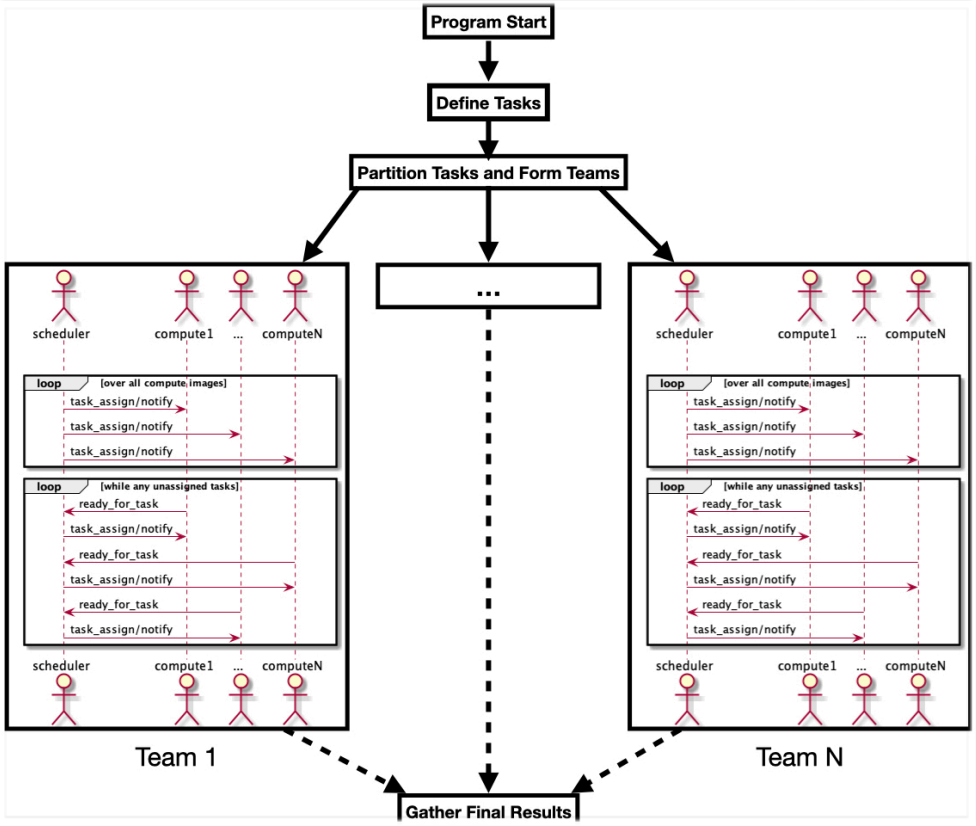
Framework for Extensible Asynchronous Task Scheduling

Execution:

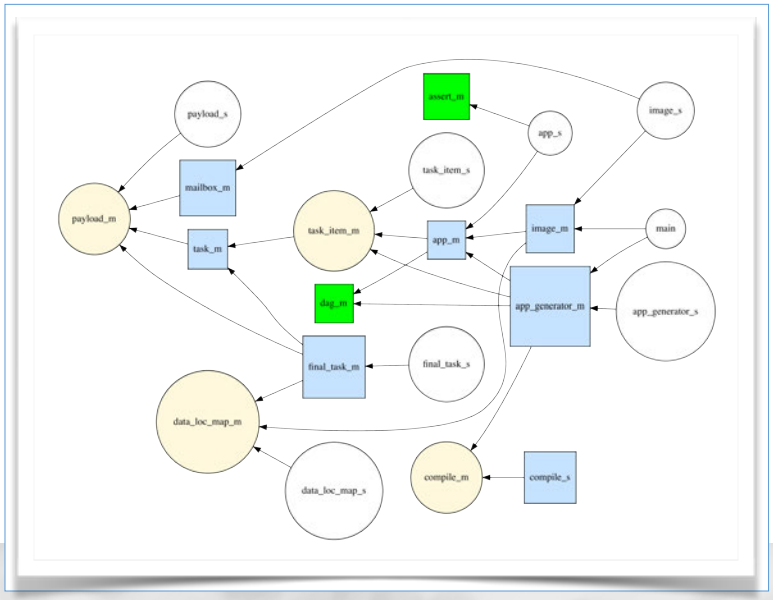
- In each team, establish one scheduler image and one or more compute images.
- Schedulers post task_assigned events to compute images in an order that respects dependencies in a directed acyclic graph (DAG).
- Compute images post ready_for_next_task events to scheduler.
- A task_payload_map_t abstraction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

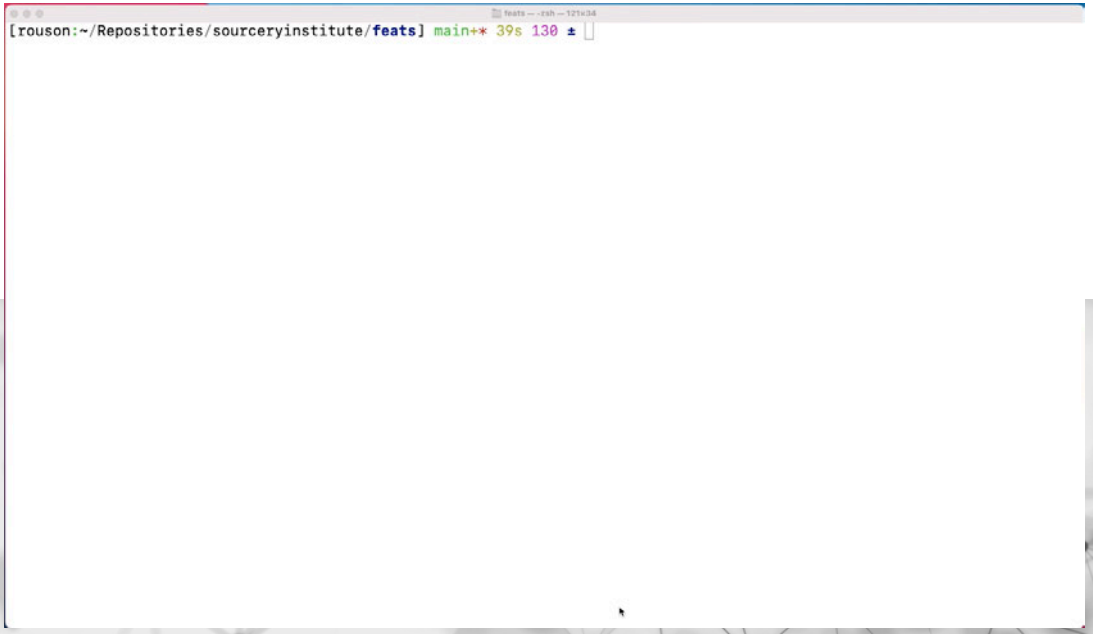
Initial target applications:

- 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
compile_m
compile_s
data_loc_map_s


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

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