What Happens to a Dream Deferred?

Chasing Automatic Offloading in Fortran 2023

Damian Rouson
Computer Languages and Systems Software Group

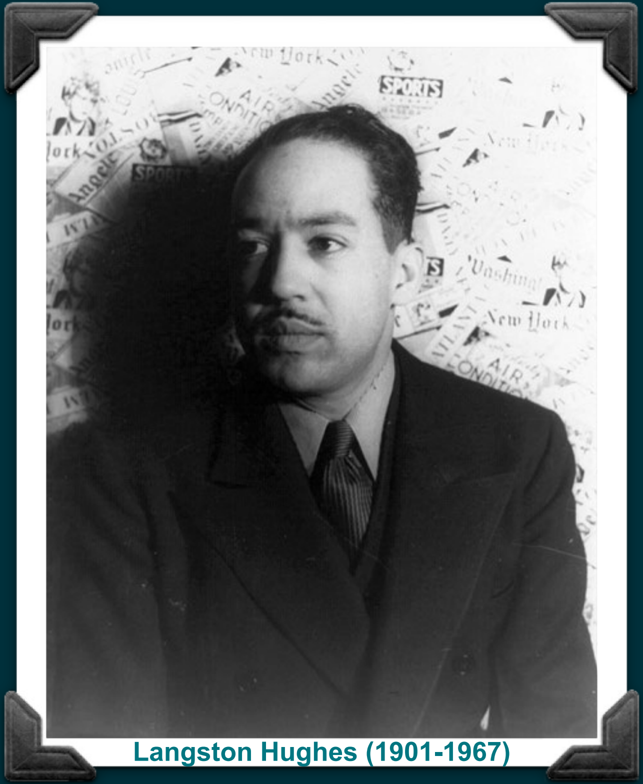
International Workshop on Automatic Performance Tuning (iWAPT), 31 May 2024

Overview

From Software Archaeology to Software Modernity

BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY
Office of Science



Langston Hughes (1901-1967)

Portrait by Carl Van Vechten, 1936. Public Domain.
Library of Congress Prints and Photographs Division Washington, D.C. 20540
http://hdl.loc.gov/loc.pnp/cph.3b38891

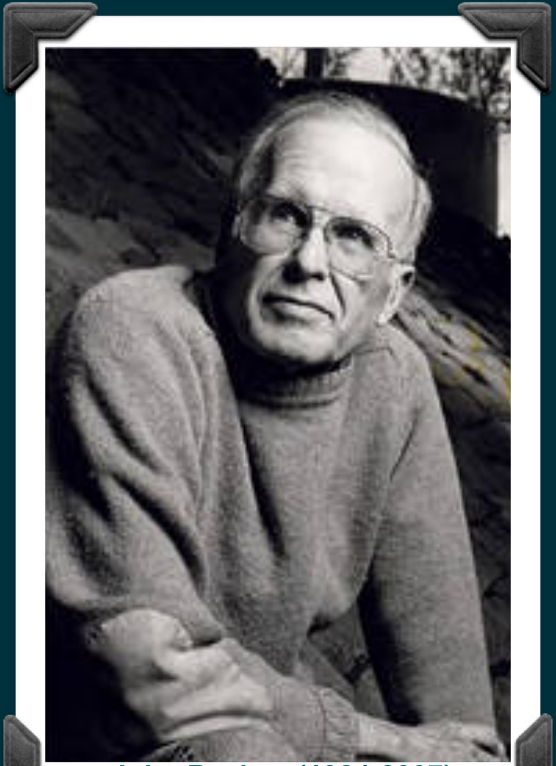
"Harlem"

By Langston Hughes, 1951

What happens to a dream deferred?
Does it dry up
like a raisin in the sun?
Or fester like a sore—
And then run?
Does it stink like rotten meat?
Or crust and sugar over—
like a syrupy sweet?

Maybe it just sags
like a heavy load.

Or does it explode?

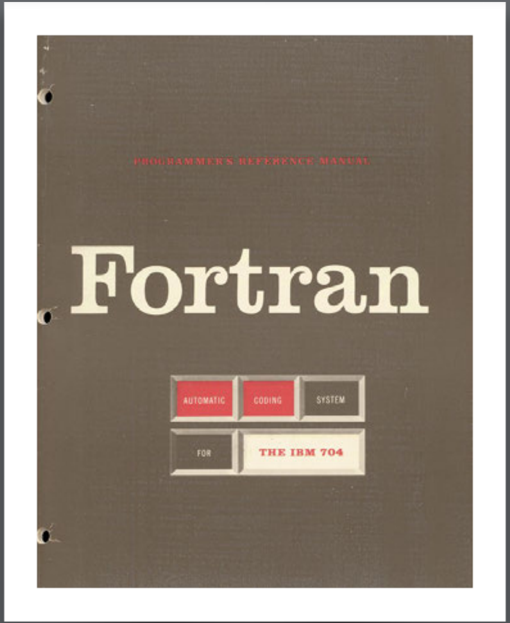


John Backus (1924-2007)

Pioneers in Science and Technology Series: John Backus, 1984

https://cdm16107.contentdm.oclc.org/digital/collection/p15388coll1/id/526

1956



The Fortran Automatic Coding System fort he IBM 704,
the first programmer's reference manual for Fortran
(Public Domain)

https://cdm16107.contentdm.oclc.org/digital/collection/p15388coll1/id/526

BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF
ENERGY
Office of Science

1961

"Fortran is a new and exciting language used by programmers to communicate with computers. It is exciting as it is the wave of the future."

Character of Dorothy Vaughan,
a NASA mathematician and programmer,
as played by Octavia Spencer in
Hidden Figures (20th Century Fox, 2016).

Overview

From Software Archaeology to Software Modernity

BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY
Office of Science

1977 Turing Award Lecture:

“Can Programming be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs”

1977 ACM Turing Award Lecture

Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs

John Backus
IBM Research Laboratory, San Jose

Backus, J., *Communications of the ACM*, August 1978, 21:8

BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY
Office of Science

Rumors of Fortran's Demise…

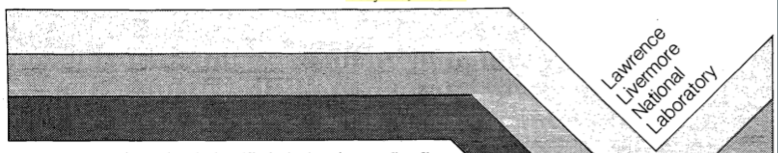
Retire Fortran? A Debate Rekindled

David Cann
Computing Research Group, L-306
Lawrence Livermore National Laboratory
P.O. Box 808, Livermore, CA 94550
cann@lll-crg.llnl.gov

Abstract

In the May 1984 issue of Physics Today, Jim McGraw debated David Kuck and Michael Wolfe on the question of retiring FORTRAN. They addressed such questions as: Is FORTRAN the best tool for decomposing

cause of today's software crisis. We bel… Graw in 1984, that increased producti… utility, portability, and performance … ble if programmers avoid the constra… tive languages and adopt a higher leve… We must escape the morass of imper…


July 24, 1991

This is a preprint of a paper intended for publication in a journal or proceedings. Since changes may be made before publication, this preprint is made available with the understanding that it will not be cited or reproduced without the permission of the author.

1991

2 Programming Alternatives

In 1984, McGraw noted that by all indications future supercomputers would be multiprocessors. Today, most supercomputer users and vendors agree. But can programmers take advantage of the horse-

model to the imperative model of FORTRAN. To begin, we list the desired characteristics of a true parallel programming language [1]:

1. The language must insulate the programmer from the underlying machine. Deriving and expressing a parallel algorithm is hard enough; one should not have to reprogram it for each new machine.

2. Parallelism must be implicit in the semantics of the language. The compilation system should not have to unravel the behavior of the computation.

3. When a programmer desires determinancy, the language should guarantee it. Regardless of the conditions of execution, a program that realizes a determinate algorithm should yield the same results for the same data.

Of the three items, the last is an issue only when automatic parallelizing compilers are not available and the programmer is responsible for expressing and managing parallelism. Programmers will make mistakes, and these mistakes may remain hidden until system activity changes the rate of execution. This is all we will say about determinancy, as most parallel machines support automatic parallelizing compilers.

Regarding the first two items, however, imperative languages fail to meet the requirements. Remember that languages like FORTRAN were designed to exploit von Neumann machines. As such their computational model assumes that a single program counter will step

For example, consider the following FORTRAN excerpt:

```
A = Foo(X)
B = Goo(Y)
```

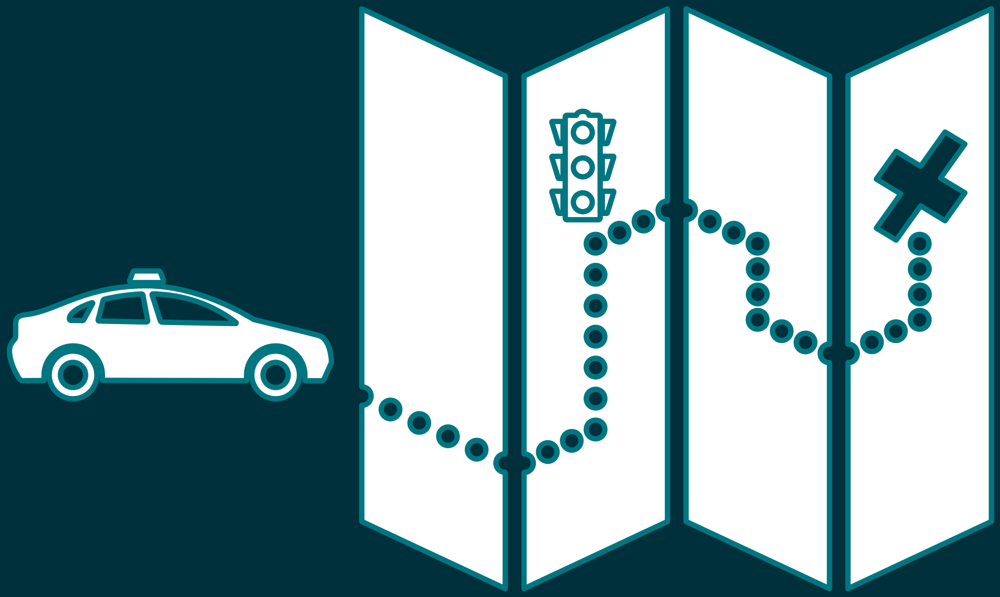
Determining if these statements can execute in parallel requires a full understanding of both functions. Because of COMMON blocks, they might share data. Further, because of aliasing, some combination of X, Y, A, or B might represent the same memory cell. Hence the parallelism in this excerpt is not immediately obvious, and its discovery requires interprocedural analysis or function expansion.

Functional languages, on the other hand, meet all the requirements listed above and do not require analysis for the discovery of parallelism [1,11,13,14]. A functional program is a collection of mathematically sound expressions comprised of both intrinsic and user defined functions. These functions are *well defined* and *determinate*. That is, they define a unique mapping between their domain and their range. A function passed the same set of values will yield the same results regardless of the environment of invocation. This establishes *referential transparency*, which implies that the evaluation of an expression, or the sharing of its subexpressions, does not change the value it denotes. Consequently, expressions are *side effect free*. The concept of a FORTRAN COMMON block does not exist. In the absence of side effects, programmers cannot see the target machine; the concept of *data* replaces memory, and the concept of *creation* replaces update. Further, in the absence of side effects, programs are implicitly parallel.

BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY
Office of Science

Or a Roadmap for Fortran's Future?



Overview

From Software Archaeology to Software Modernity

01
Background

02
Motivation

03
Parallelism in Fortran 2023

04
AI

05
HPC

06
Ruminations

Explicit Parallelsim in Fortran 2023

BERKELEY LAB
Bringing Science Solutions to the World

Single Program Multiple Data (SPMD) parallel execution

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

— Asynchronous execution except where program explicitly synchronizes

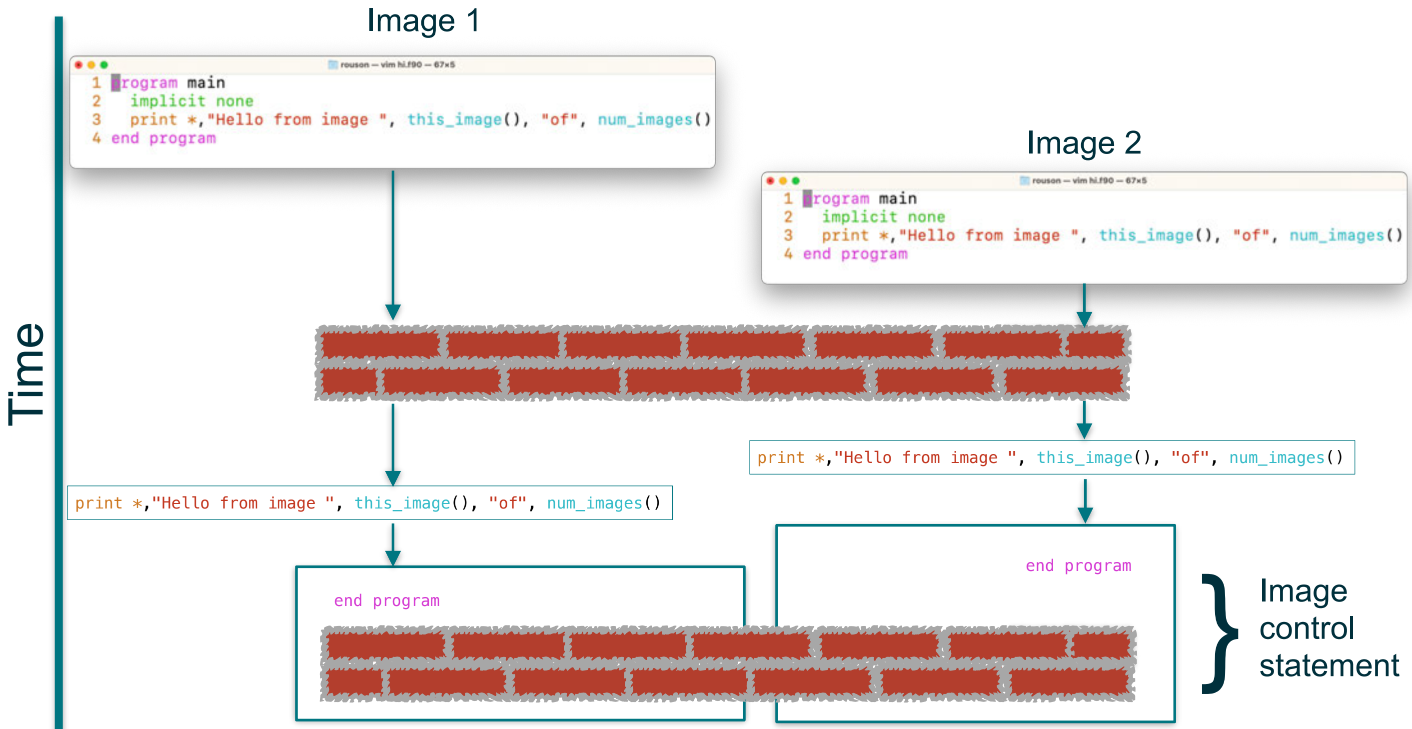
— Error termination or synchronized normal termination

rouson — vim hi.f90 — 67×5

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

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
```

Additional Parallel Features

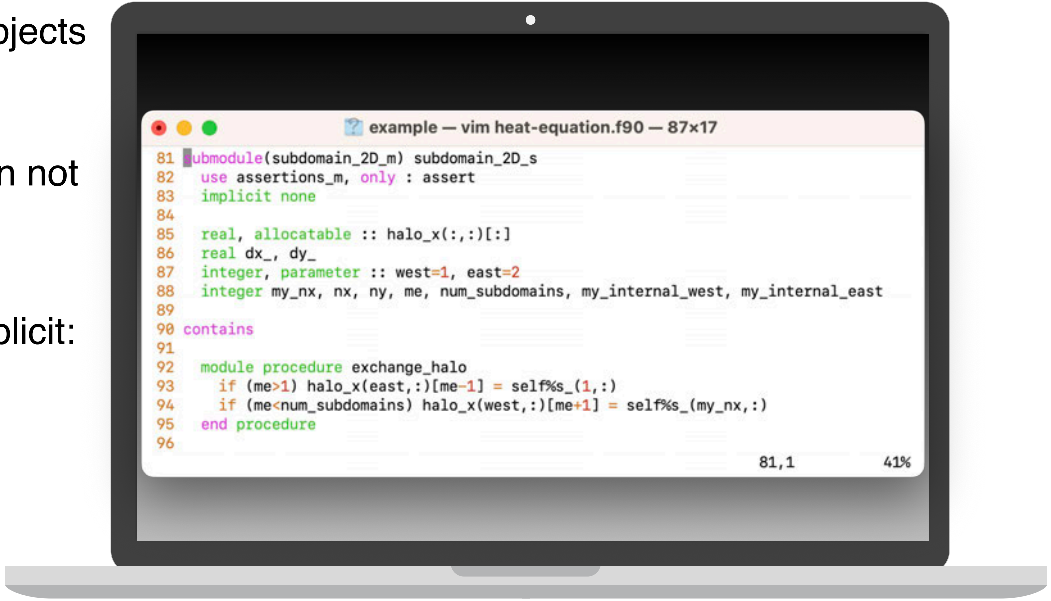
BERKELEY LAB
Bringing Science Solutions to the World

- Teams of images can be formed at runtime.
- Collective subroutines: `co_{broadcast, sum, max, min, reduce}`
- Atomic subroutines:
  - `atomic_{define,ref,add,fetch_add,…}`
  - Events: counting semaphores with post/wait/query operations
- Failed/stopped image detection, locks, critical sections, …

Explicit Parallelsim: Coarray Fortran

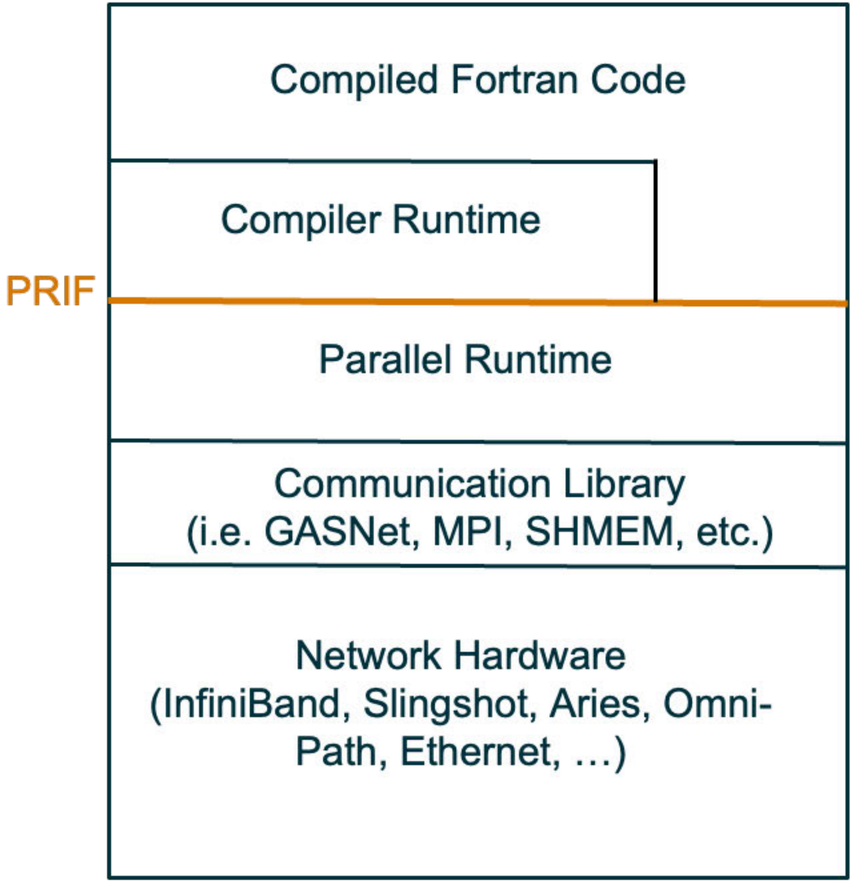
BERKELEY LAB
Bringing Science Solutions to the World

- 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



PRIF

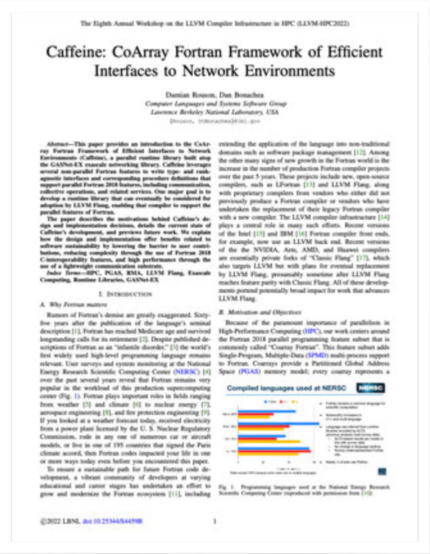
- Enable a compiler to target multiple implementations of PRIF
  - I.e. enable a vendor to supply their own parallel runtime
- Enable a PRIF implementation to be used by multiple compilers
- Isolate a compiler's support of the parallel features of the language from any particular details of the communication infrastructure
- Our group's experience with UPC and OpenCoarrays has shown this to be valuable



Caffeine

Co-Array Fortran Framework of Efficient Interfaces to Network Environments

- Caffeine supports the parallel features of Fortran 2018 for compilers.
- Caffeine leverages GASNet-EX, a high-performance networking middleware that undergirds a broad ecosystem of languages, libraries, frameworks, and applications.



The Eighth Annual Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC2022)

Caffeine: CoArray Fortran Framework of Efficient Interfaces to Network Environments

Damian Rouson, Dan Bonachea
Computer Languages and Systems Software Group
Lawrence Berkeley National Laboratory, USA

Abstract—This paper provides an introduction to the CoArray Fortran Framework of Efficient Interfaces to Network Environments (Caffeine), a parallel runtime library built atop the GASNet-EX exascale networking library. Caffeine leverages several non-parallel Fortran features to write type- and rank-agnostic interfaces and corresponding procedure definitions that support parallel Fortran 2018 features, including communication, collective operations, and related services. One major goal is to develop a runtime library that can eventually be considered for adoption by LLVM Flang, enabling that compiler to support the parallel features of Fortran.

The paper describes the motivations behind Caffeine's design and implementation decisions, details the current state of Caffeine's development, and previews future work. We explain how the design and implementation offer benefits related to software sustainability by lowering the barrier to user contributions, reducing complexity through the use of Fortran 2018 C-interoperability features, and high performance through the use of a lightweight communication substrate.

Index Terms—HPC, PGAS, RMA, LLVM Flang, Exascale Computing, Runtime Libraries, GASNet-EX

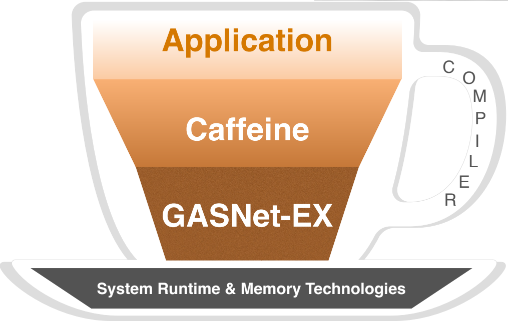
I. Introduction

A. Why Fortran matters

B. Motivation and Objectives

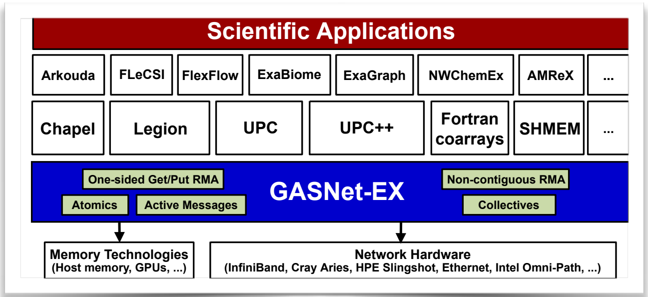
Compiled languages used at NERSC

LLVM for HPC Workshop

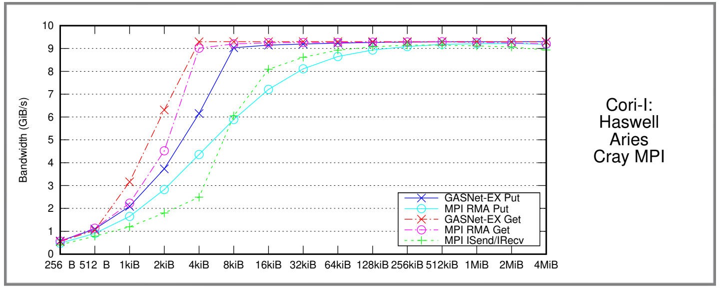


GASNet-EX

GASNet-EX Ecosystem



Microbenchmark: GASNet-EX vs MPI



D. Bonachea and P. H. Hargrove, "GASNet-EX: A High- Performance, Portable Communication Library for Exascale," in *Proceedings of Languages and Compilers for Parallel Computing (LCPC'18)*, ser. LNCS, vol. 11882. Springer, October 2018, doi:10.25344/S4QP4W.

Overview

From Software Archaeology to Software Modernity

Implicit Parallelism

BERKELEY LAB
Bringing Science Solutions to the World

In addition to the SPMD/PGAS features that work in shared or distributed memory, several features facilitate expressing unordered sets of calculations amenable to multithreading, vectorization, or accelerator offloading:

- `do concurrent` + `pure` procedures, including `elemental` procedures

```fortran
    integer row, col
    integer, parameter :: window=4, time=1

    associate(rows => size(distance%body,1), cols => size(distance%body,2))
      do concurrent(row=1:rows, col=1:cols)
        associate(first_row => max(1, row-window), last_row=>min(row+window, rows))
          distance%body(row,col) = minval(hypot( &
            this%body(first_row:last_row, time) - rhs%body(row, time), &
            this%body(first_row:last_row,  col) - rhs%body(row,  col) &
          ))
        end associate
      end do
    end associate
```

- `where` statement

```fortran

    where(rhs_filtered/=0._rkind)
      distance%body = distance%body/rhs_filtered
    elsewhere
      distance%body = 0.
    end where

```

- Array statements + `elemental` procedures (intrinsic or user-defined): `matmul`, `reduce`, `transpose`, `dot_product`, `merge`, `pack`, `unpack`, `count`, `any`, `all`, `findloc`, …

Inference-Engine

Use case:

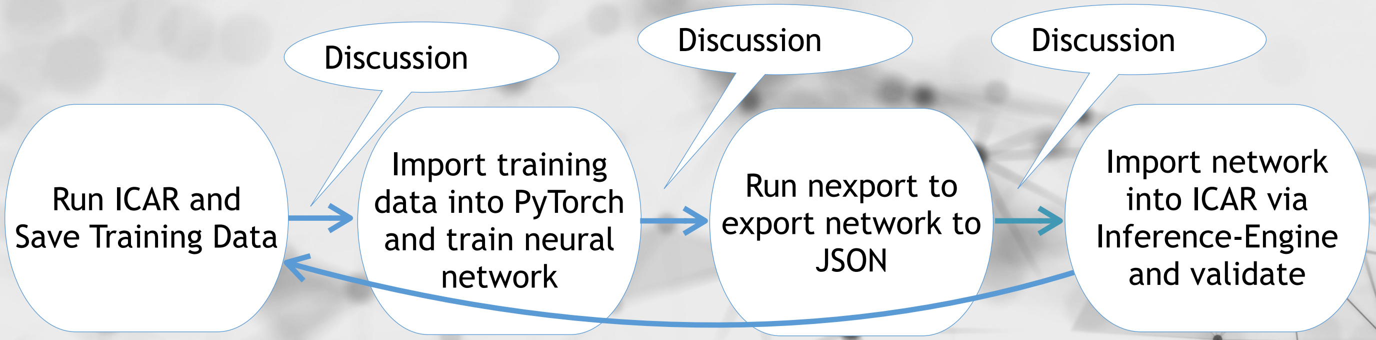
- Large-batch, concurrent inference and *in situ* training of neural networks for high-performance computing applications in modern Fortran.

Goals:

- To explore language-based parallelism, including GPU offloading.
- To simplify the workflow for training neural networks, i.e., eliminate the telephone game.

How:

- A functional programming style that facilitates concurrent inference across a large collection of inputs using multiple specialized neural networks.
- A training algorithm that squeezes out most unnecessary programmer-imposed ordering of



Inference-Engine

BERKELEY LAB
Bringing Science Solutions to the World

Use case:

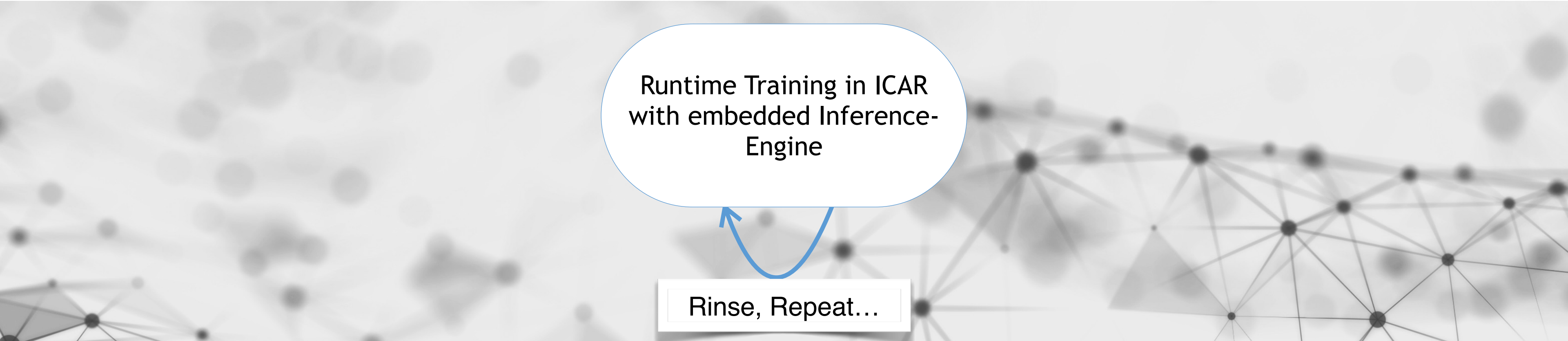
- Large-batch, concurrent inference and *in situ* training of neural networks for high-performance computing applications in modern Fortran.

Goals:

- To explore language-based parallelism, including GPU offloading.
- To simplify the workflow for training neural networks, i.e., eliminate the telephone game.

How:

- A functional programming style that facilitates concurrent inference across a large collection of inputs using multiple specialized neural networks.
- A training algorithm that squeezes out most unnecessary programmer-imposed ordering of



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

AI

```fortran
do k=1,lev
  do j=1,lon
    do i=1,lat
      outputs(i,j,k) = inference_engine%infer(inputs(i,j,k))
    end do
  end do
end do
```

```fortran
do concurrent(i=1:lat, j=1:lon, k=1:lev)
  outputs(i,j,k) = inference_engine%infer(inputs(i,j,k))
end do
```

```fortran
outputs = inference_engine%infer(inputs)  ! elemental
```

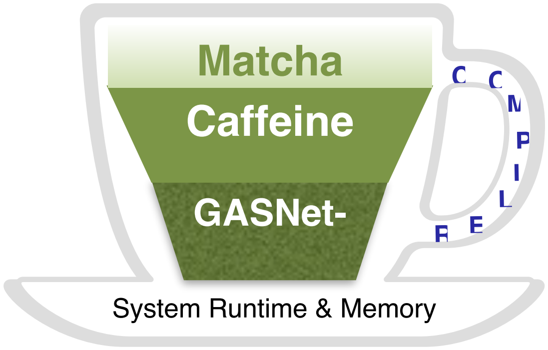
Motility Analysis of T-Cell Histories in Activation (Matcha)

A parallel virtual T-cell model.

- Matcha tracks the stochastic T-cell motions according to multiple distributions of speeds and angles, accounting for the dependence of speed on the turning angle and on the previous speed.
- T cells must mount a coordinated attack in order to avoid overwhelming the host tissue.
- The study of T-cell/T-cell interactions remains in its infancy [1].
- Some communication occurs via secreting soluble mediators, e.g., cytokines and chemokines.
- Matcha models mediator spread via a 3D diffusion equation:

$$\phi_t = D\nabla^2\phi$$

where $\phi_t = \partial\phi/\partial t$.



[1] L.F. Uhl and A. Ge´rard A. “Modes of communication between T cells and relevance for immune responses.” *Int. J. Mol. Sci.* **2020**, *21*, 2674; doi:10.3390/ijms21082674

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

U.S. DEPARTMENT OF ENERGY
Office of Science

A Functional Programming Pattern

test — vim subdomain_test_m.f90 — 68×28

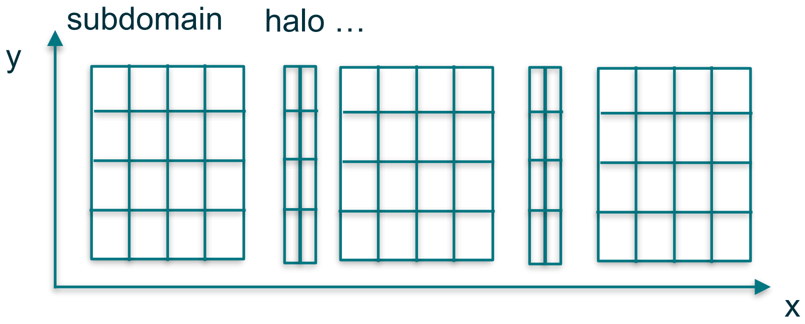
```fortran
function functional_matches_procedural() result(test_passes)
  logical test_passes
  integer, parameter :: steps = 6000, n=32
  real, parameter :: tolerance = 1.E-06, alpha = 1.
  real, parameter :: side=1., boundary_val=1., internal_val=2.
  associate( T_f => T_functional(), T_p => T_procedural())
    associate(L_infinity_norm => maxval(abs(T_f - T_p)))
      test_passes = L_infinity_norm < tolerance
    end associate
  end associate
contains
  function T_functional()
    real, allocatable :: T_functional(:,:,:)
    type(subdomain_t), save :: T[*]
    integer step

    call T%define(side, boundary_val, internal_val, n)

    associate(dt => T%dx()*T%dy()/(4*alpha))
      do step = 1, steps
        sync all
        T =  T + dt * alpha * .laplacian. T
      end do
    end associate

    T_functional = T%values()
  end function
```

179,23

- Explicitly **pure** procedures
  - Side-effect free: no I/O, no **stop**, no image control, etc.
  - Functions: **intent(in)** arguments
  - Subroutines: specified argument **intent**
  - Deterministic in most cases (Fortran 202X simple removes most non-determinism)
- Implicitly **pure** procedures: **elemental**
- **Associate**
  - Define immutable state by associating with an expression, e.g., function reference.
- Only **pure** procedures may be invoked inside a **do concurrent** block.
  - Every intrinsic function is **pure**
- Error termination in **pure** procedures
  - Variable **stop** codes
- Use objects to encapsulate multiple entities in one function results.

Halo Exchange

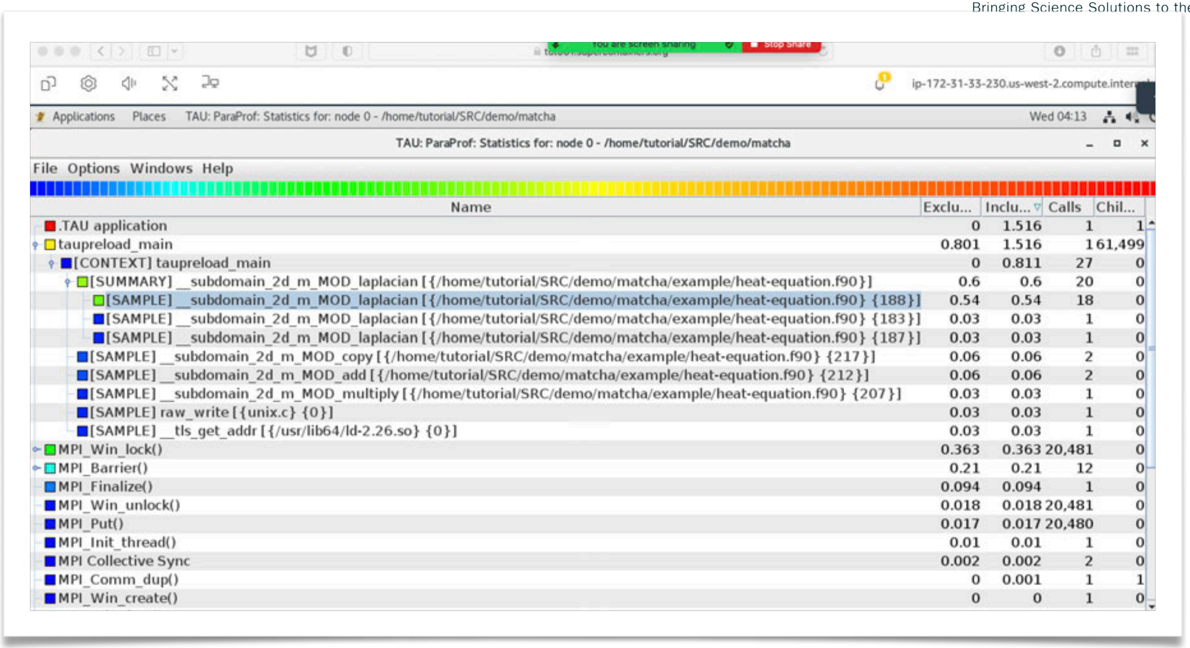
BERKELEY LAB
Bringing Science Solutions to the World




```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



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

```
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
```

Repositories — vim nse.f90 — 63×12

```
program main
  use vector_field_m, only : vector_field_t
  use scalar_field_m, only : scalar_field_t
  implicit none
  type(vector_field_t) u, u_t
  type(scalar_field_t) p
  real, parameter :: rho = 1.23, nu=1.65E-05

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

end program
```

Purely functional parallel algorithms (user-defined operators) operating on distributed objects (derived type coarrays) with automatic GPU offloading via do concurrent.

Compiler Status

Supporting CAF features:

- Cray
- Intel
- GNU
- NAG

Automatic offloading of do concurrent:

- NVIDIA
- Intel
- Cray

LLVM Flang:

- Parses and verifies CAF syntax and semantics
- Does not yet lower CAF features
- Berkeley Lab develops

  -- Frontend unit tests for CAF features

  -- Frontend bug fixes

  -- Caffeine: a candidate parallel runtime

  -- PRIF: a specification

The World's Shortest Bug Reproducer

```
end
```

Overview

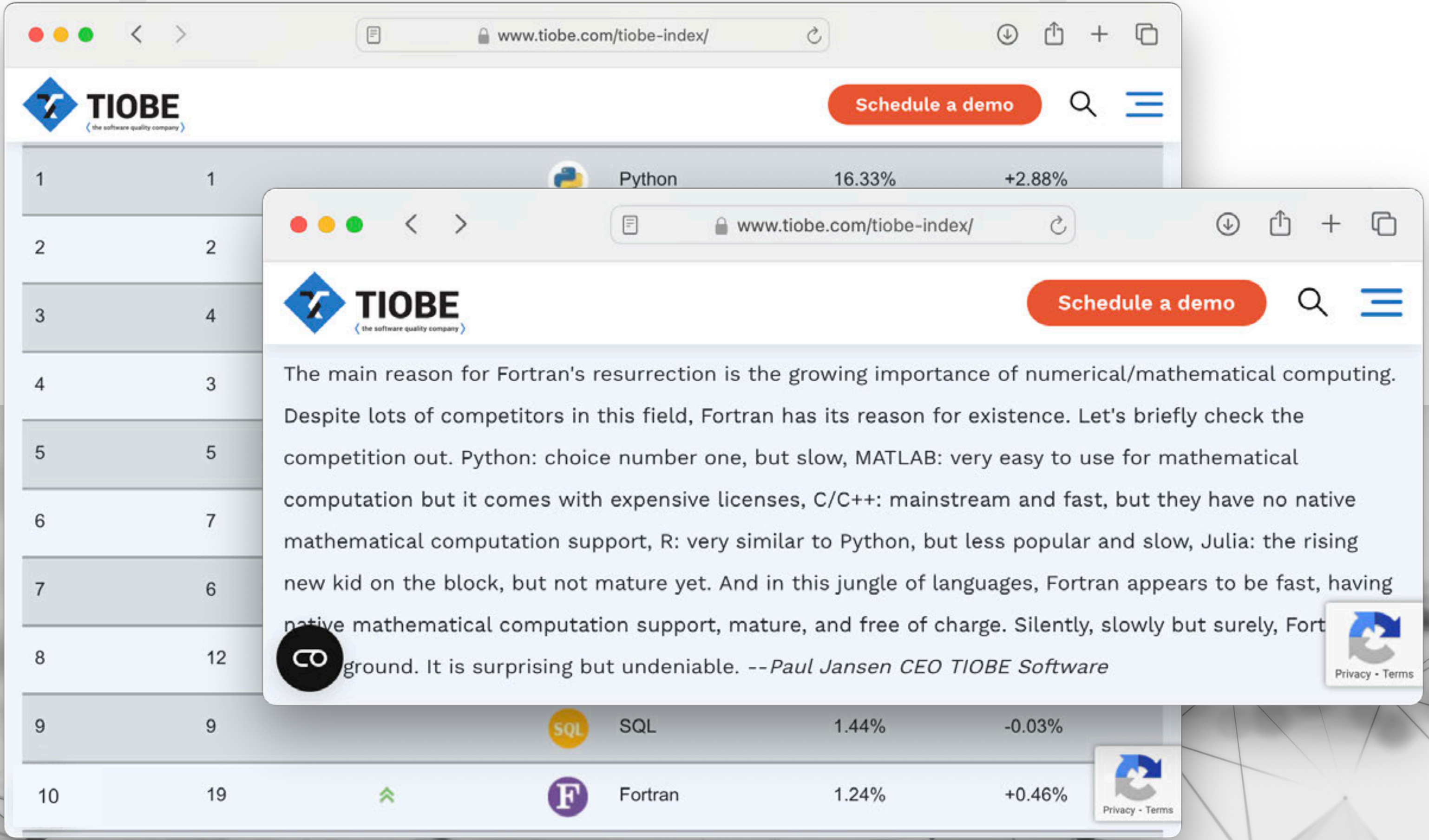
From Software Archaeology to Software Modernity

www.tiobe.com/tiobe-index/

TIOBE
the software quality company

Schedule a demo

1	1		Python	16.33%	+2.88%
2	2		C	9.98%	-3.37%
3	4	↑	C++	9.53%	-2.43%
4	3	↓	Java	8.69%	-3.53%
5	5		C#	6.49%	-0.94%
6	7	↑	JavaScript	3.01%	+0.57%
7	6	↓	Visual Basic	2.01%	-1.83%
8	12	⇈	Go	1.60%	+0.61%
9	9		SQL	1.44%	-0.03%
10	19	⇈	Fortran	1.24%	+0.46%



www.tiobe.com/tiobe-index/
TIOBE
the software quality company
Schedule a demo
1 1 Python 16.33% +2.88%
2 2
3 4
4 3
5 5
6 7
7 6
8 12
9 9 SQL 1.44% -0.03%
10 19 Fortran 1.24% +0.46%
Privacy - Terms
www.tiobe.com/tiobe-index/
TIOBE
the software quality company
Schedule a demo
The main reason for Fortran's resurrection is the growing importance of numerical/mathematical computing. Despite lots of competitors in this field, Fortran has its reason for existence. Let's briefly check the competition out. Python: choice number one, but slow, MATLAB: very easy to use for mathematical computation but it comes with expensive licenses, C/C++: mainstream and fast, but they have no native mathematical computation support, R: very similar to Python, but less popular and slow, Julia: the rising new kid on the block, but not mature yet. And in this jungle of languages, Fortran appears to be fast, having
ground. It is surprising but undeniable. --Paul Jansen CEO TIOBE Software
Privacy - Terms

Ruminations

What Happens to a Dream Deferred?

01

Sometimes it sags like a heavy burden.

02

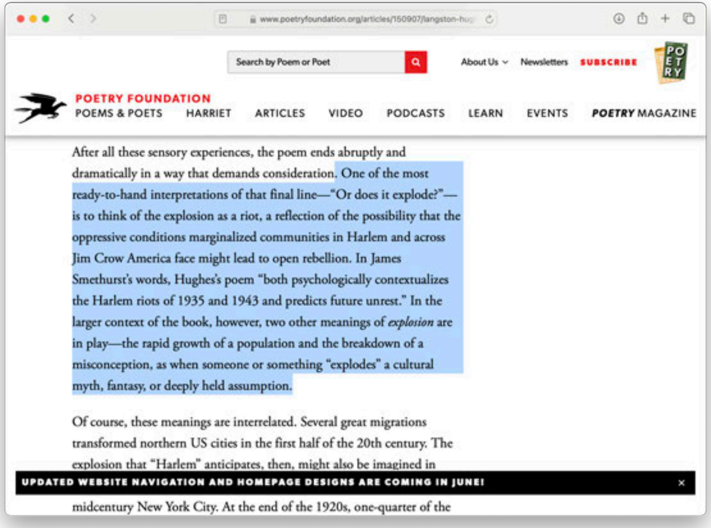
Sometimes it explodes in a segmentation fault!

03

Sometimes it explodes in popularity.

04

Let's hope the popularity maintains and realizes the dream.



After all these sensory experiences, the poem ends abruptly and dramatically in a way that demands consideration. One of the most ready-to-hand interpretations of that final line—"Or does it explode?"—is to think of the explosion as a riot, a reflection of the possibility that the oppressive conditions marginalized communities in Harlem and across Jim Crow America face might lead to open rebellion. In James Smethurst's words, Hughes's poem "both psychologically contextualizes the Harlem riots of 1935 and 1943 and predicts future unrest." In the larger context of the book, however, two other meanings of *explosion* are in play—the rapid growth of a population and the breakdown of a misconception, as when someone or something "explodes" a cultural myth, fantasy, or deeply held assumption.

Of course, these meanings are interrelated. Several great migrations transformed northern US cities in the first half of the 20th century. The explosion that "Harlem" anticipates, then, might also be imagined in

UPDATED WEBSITE NAVIGATION AND HOMEPAGE DESIGNS ARE COMING IN JUNE!

midcentury New York City. At the end of the 1920s, one-quarter of the

https://www.poetryfoundation.org/articles/150907/langston-hughes-harlem

Acknowledgements

The Berkeley Lab Fortran Team

Dan Bonachea, Hugh Kadhem, Brad Richardson, Kate Rasmussen

Past and Present Collaborators

Jeremy Bailey, David Torres, Kareem Jabbar Weaver, Jordan Welsman, Yunhao Zhang

BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY
Office of Science

The Problem is Not Fortran

Damian Rouson

Computer Languages and Systems Software (CLaSS) Group ( )

NUCLEI Meeting, 29 May 2024

Popularity and Use

- Tiobe Index
- NERSC Data
- Open-Source: fpm, Caffeine, Veggies, Rojff
- Growth in Compilers: LFortran, LLVM Flang, …

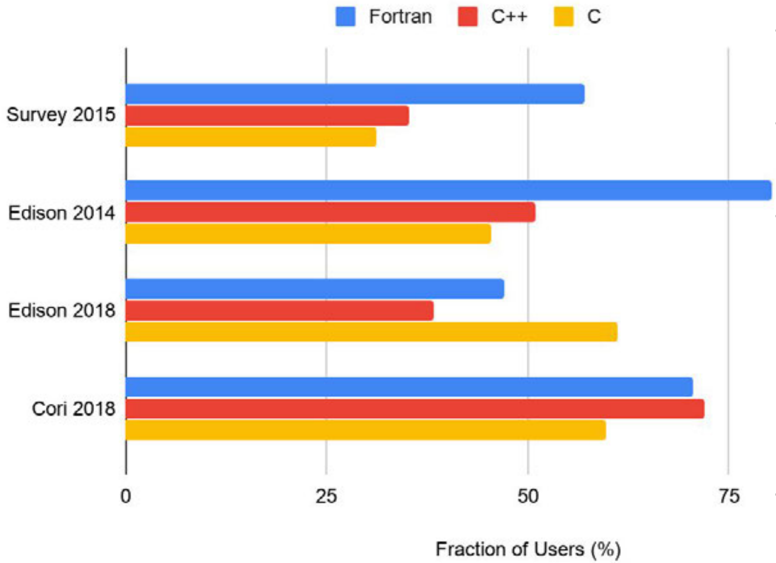
Fortran 2023 by Example

- Fusion
- Weather
- Climate
- FFTs, Multigrid, etc.

So what are the Problems?

- Perception
- Geography/Culture
- State of Practice
- State of Compilers

Compiled languages used at NERSC



Totals exceed 100% because some users rely on multiple languages.

- Fortran remains a common language for scientific computation.
- Noteworthy increases in C++ and multi-language
- Language use inferred from runtime libraries recorded by ALTD. (previous analysis used survey data)
  - ALTD-based results are mostly in line with survey data.
  - No change in language ranking
  - Survey underrepresented Fortran use.
- Nearly ¼ of jobs use Python.

Source: B. Austin et al., NERSC-10 Workload Analysis, 2020, doi:10.25344/S4N30W.

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

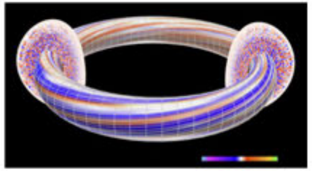


Figure 2: GTS field-line following grid & toroidal domain decomposition. Colors represent isocontours of the quasi-two-dimensional electrostatic potential

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).

- 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."

CAF at Scale: Weather

BERKELEY LAB
Bringing Science Solutions to the World


Article

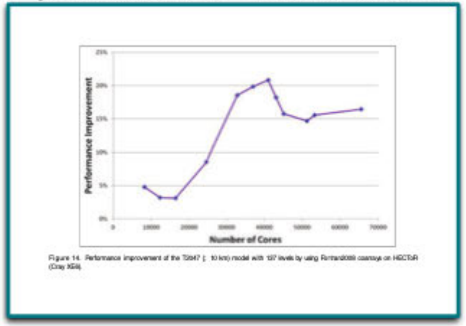
A Partitioned Global Address Space implementation of the European Centre for Medium Range Weather Forecasts Integrated Forecasting System

George Mozdzynski, Mats Hamrud and Nils Wedi

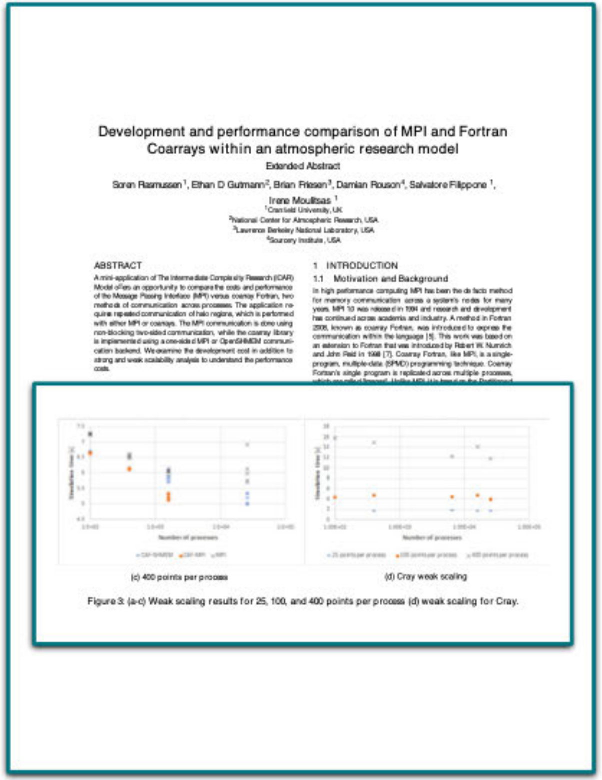
Figure 7. EQ_REGIONS partitioning of grid-point space, showing a partition at the poles and then an increasing number of partitions as we approach the equator.

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.

- **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



CAF at Scale: Climate


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

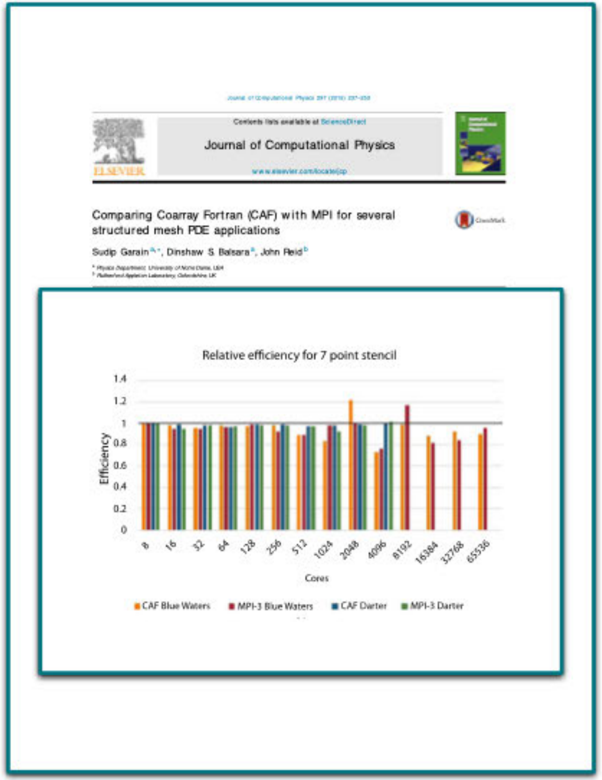
Figure 3: (a-c) Weak scaling results for 25, 100, and 400 points per process (d) weak scaling for Cray.

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."

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…"