BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY
Office of Science

Please, No More Loops (Than Necessary):

New Patterns in Fortran 2023

Damian Rouson
Computer Languages and Systems Software (CLaSS) Group

HPC Best Practices Webinar, 21 January 2026

Acknowledgements

The Berkeley Lab Fortran Team

Dan Bonachea, Brandon Cook, Paul Hargrove, Hugh Kadhem, Kate Rasmussen

Collaborators

Julienne: Katherine Rasmussen, Dan Bonachea

Fiats: Zhe Bai, Jeremiah Bailey, Baboucarr Dibba, Dan Bonachea, Ethan Gutmann, Brad Richardson, Sameer Shende, David Torres, Katherine Rasmussen, Kareem Jabbar Weaver, Jordan Welsman, Yunhao Zhang

Formal, MOLE: Dan Bonachea, Jose Castillo, Johnny Corbino, Joseph Hellmers

Flang, Caffeine, PRIF: Dan Bonachea, Kareem Ergawy, Jeff Hammond, Michael Klemm, Jean-Didier Pailleux, Etienne Renault, Katherine Rasmussen, Brad Richardson

Matcha: Dan Bonachea, David Torres, Dominick Martinez

Sponsors

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, and Office of Nuclear Physics. This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231. This research was supported by the Lawrence Berkeley National Laboratory Laboratory Research and Development (LDRD) program.

Overview

01

Background:

Deep origins

02

Features & Paradigms:

A walking tour from Fortran 90 to 202Y+

03

Use Cases:

Nooks, Crannies, and Pastures

04

Conclusions

Overview

01

Background:

Deep origins

02

Features & Paradigms:

A walking tour from Fortran 90 to 202Y+

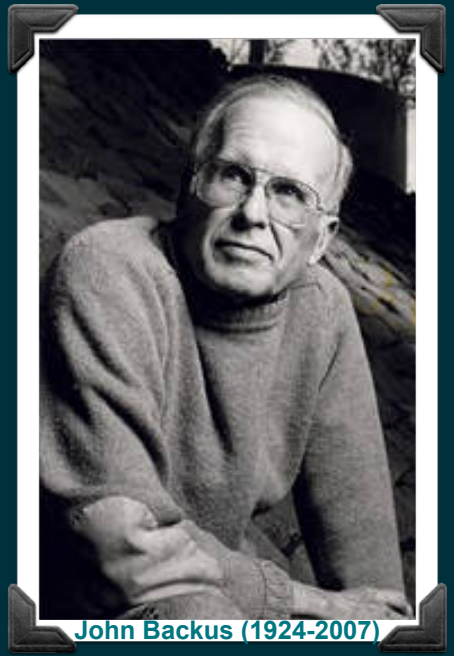
03

Use Cases:

Nooks, Crannies, and Pastures

04

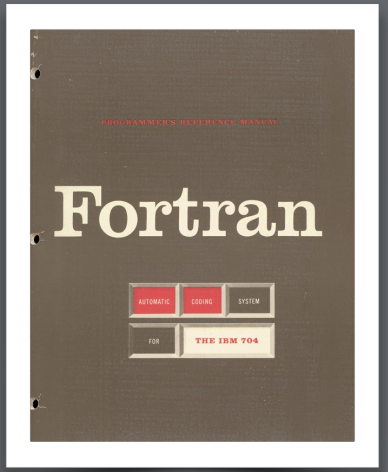
Conclusions



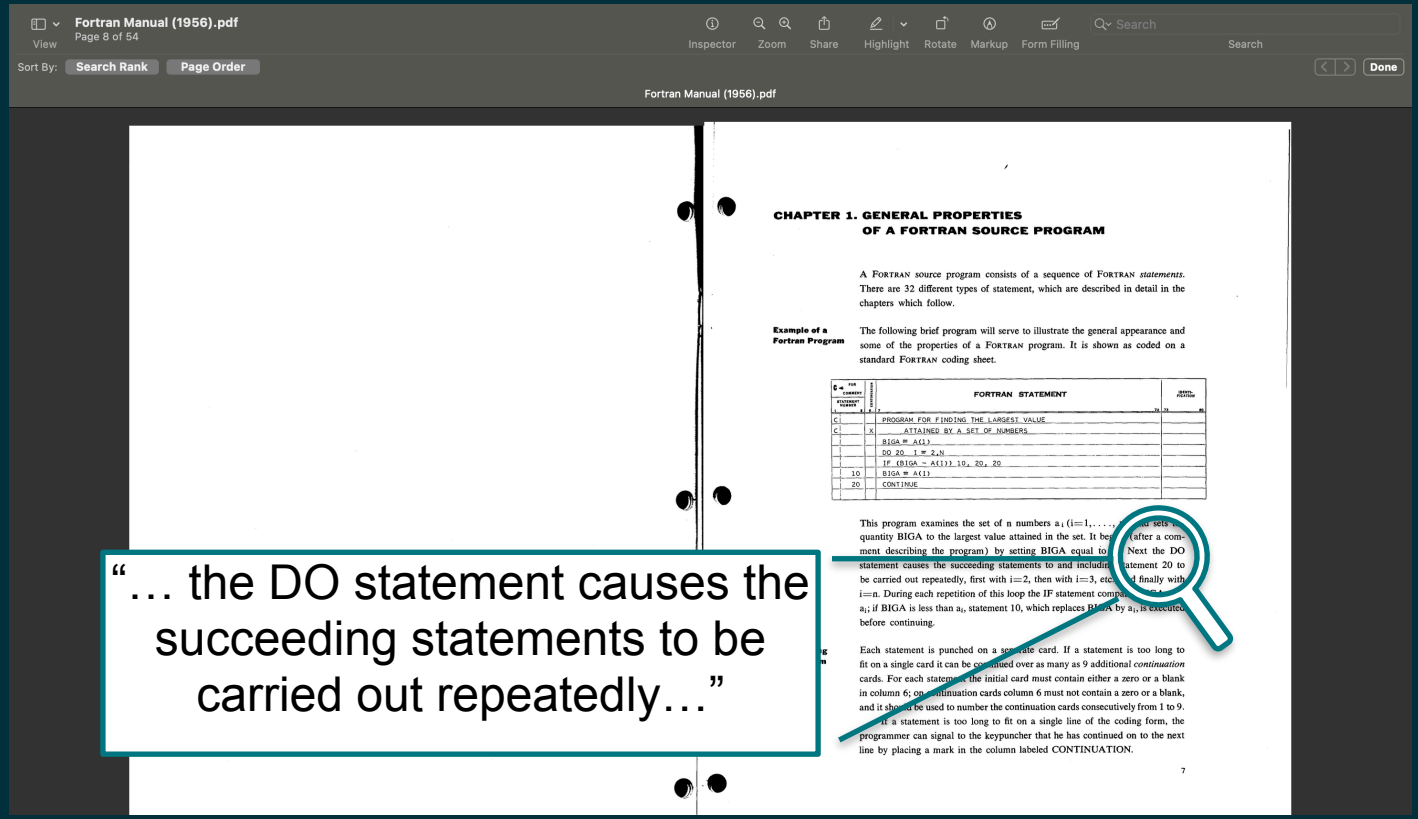
John Backus (1924-2007)

Pioneers in Science and Technology Series: John Backus, 1984
© City of Oak Ridge, Oak Ridge, TN 3783 (Public Domain)
https://cdm16107.contentdm.oclc.org/digital/collection/p15388coll1/

1956



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



"… the DO statement causes the succeeding statements to be carried out repeatedly…"

Overview

01

Background:

Deep origins

02

Features & Paradigms:

A walking tour from Fortran 90 to 202Y+

03

Use Cases:

Nooks, Crannies, and Pastures

04

Conclusions

New Features

90

Derived types

Structure constructors

Derived type input/output (I/O)

Data and procedure privacy

Pointers (required for dynamically allocating object components)

Defined operations and defined assignments

Recursive procedures

Allocatable arrays

Array statements, structure constructors, and intrinsic functions

Modules

Interface blocks & bodies, including generic interfaces

Structured branching and looping

Kind parameters and related intrinsic functions

Free-form source (.f90)

Paradigms

Object-Based Programming

Functional programming

Array programming

Modular programming

Structured programming

New Features

90

95

`pure` procedures

`elemental` procedures

`where` construct

`forall` construct (obsolescent)

Paradigms

} Functional programming

} Array programming

} Parallel/vector programming

New Features

Type extension, type-bound procedures, polymorphism, and final subroutines

Generic bindings, including type-bound operators and assignments

Type/source allocation & automatic (re-)allocation via intrinsic assignment

`Allocatable` scalars and components

Parameterized derived types

Vector subscripts

Abstract types, abstract interfaces, and deferred bindings

`Associate` construct

`iso_c_binding` module

`Iso_fortran_env` module: `input_unit, error_unit,…`

Paradigms

} Object-Oriented Programming (OOP)

} Generic programming

} Array programming

} Object-Oriented Design Patterns (OOD)

} Functional programming

} (C Interoperability)

New Features

- images
- coarrays
- synchronization, `critical` blocks, locks
- Atomic subroutines and atomic kinds
- `do concurrent`
- `Impure elemental` procedures
- `Contiguous` array dummy arguments
- Pointer rank remapping
- `Mold` allocation and automatic type (re-)allocation via intrinsic assignment
- Submodules, module procedure interfaces

Paradigms

Parallel/GPU programming:
- SPMD
- PGAS
- Shared or distributed memory

Array programming

Modular programming

New Features

Kind parameters: `c_int32_t, ...`

Assumed-type (`type(*)`) and assumed-rank (`a(..)`)dummy arguments and rank guarding (select rank)

`ISO_Fortran_binding.h`:

Data structures: `CFI_cdesc_t, CFI_dim_t, CFI_rank_t, CFI_type_t, CFI_attribute_t`

Functions: `CFI_address(), CFI_allocate(), CFI_deallocate(), CFI_establish(), CFI_is_contiguous(), CFI_section()`

Collective subroutines: `co_sum, co_broadcast, co_min, co_max, co_reduce`

Events: `event_type, event_query(), event wait, event post`

Locality specifiers for `do concurrent`: `local, shared, default(none)`

Teams

Failed images

(C-Interoperability)

Parallel/vector programming

Paradigms

New Features

Paradigms

`do concurrent reduce` locality

put notifications: `notify_type`, `notify wait`

enumeration types

maximum statement length: 1 million characters

} Parallel/vector programming

New Features

Paradigms

Type-safe templates

Generic subroutines

} Generic programming

Enhanced collective subroutines: team collectives, asynchronous collectives

} Parallel programming

+ Standardized preprocessor: Part 2?

Nooks & Crannies

www.merriam-webster.com

NOOK Definition & Meaning - Merriam-Webster

nook

Definition | Synonyms | Example Sentences | Word History | Phra

nook noun

ˈnu̇k

Synonyms of *nook* ›

1 **chiefly Scotland** : a right-angled corner

2 **a** : an interior angle formed by two meeting walls

b : a secluded or sheltered place or part

searched every *nook* and cranny

c : a small often recessed section of a larger room

a breakfast *nook*

www.marthastewart.com/

22 Reading Nook Ideas for Turning Any Space Into a Cozy Escape

martha stewart

22 Reading Nook Ideas for Turning Any Space Into a Cozy Escape

Get inspired to find the perfect place to forget about your cares and slip away into a good book with these reading nook ideas.

By **Heather Bien** | Published on July 3, 2024

Credit: Barr Joinery & Lucy Walters

www.merriam-webster.com

CRANNY Definition & Meaning - Merriam-Webster

cranny

Definition | Synonyms | Example Sentences | Word History | Phra

cranny noun

cran·ny ˈkra-nē

plural **crannies**

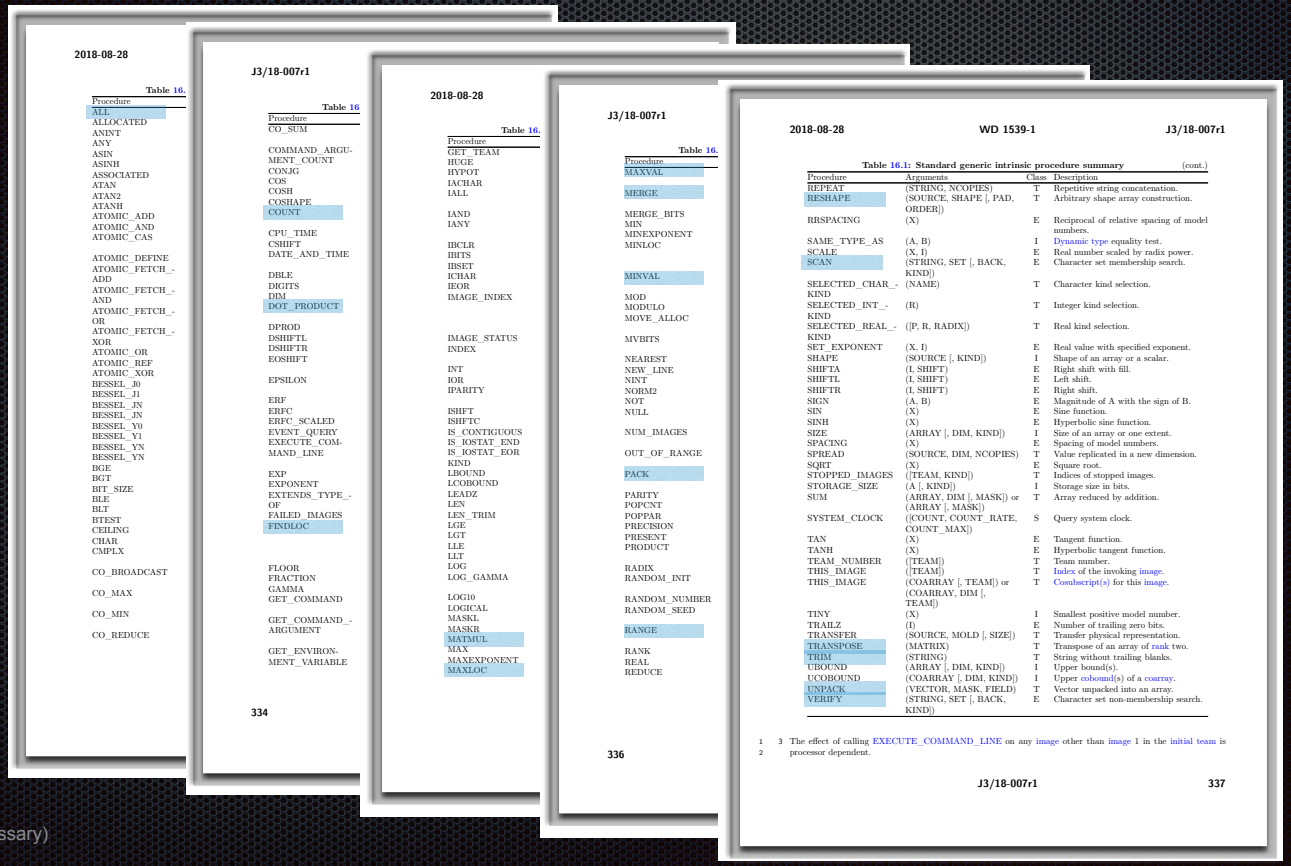
Synonyms of *cranny* ›

1 : a small break or slit : **CREVICE**

2 : an obscure nook or corner

crannied ˈkra-nēd adjective

Fortran 2023 Intrinsic Functions



Expansive Pastures

Davos, Switzerland, on the way to PASC23

Fortran 2023 Multi-Image Features

- Statements
  - Synchronization
    - Explicit: SYNC ALL, SYNC IMAGES, SYNC MEMORY, SYNC TEAM
    - Implicit: ALLOCATE, DEALLOCATE, STOP, END, MOVE_ALLOC
  - Events: EVENT POST, EVENT WAIT
  - Notify: NOTIFY WAIT
  - Error termination: ERROR STOP
  - Locks: LOCK, UNLOCK
  - Failed images: FAIL IMAGE
  - Teams: FORM TEAM, CHANGE TEAM
  - Critical sections: CRITICAL, END CRITICAL
- Coarray Accesses ([...])
- Intrinsic functions: NUM_IMAGES, THIS_IMAGE, LCOBOUND, UCOBOUND, TEAM_NUMBER, GET_TEAM, FAILED_IMAGES, STOPPED_IMAGES, IMAGE_STATUS, COSHAPE, IMAGE_INDEX

- Intrinsic subroutines
  - Collective subroutines: CO_SUM, CO_MAX, CO_MIN, CO_REDUCE, CO_BROADCAST
  - Atomic subroutines: ATOMIC_ADD, ATOMIC_AND, ATOMIC_CAS, ATOMIC_DEFINE, ATOMIC_FETCH_ADD, ATOMIC_FETCH_AND, ATOMIC_FETCH_OR, ATOMIC_FETCH_XOR, ATOMIC_OR, ATOMIC_REF, ATOMIC_XOR
  - Other subroutines: EVENT_QUERY
- Types, kind type parameters, and values
  - Intrinsic derived types: EVENT_TYPE, TEAM_TYPE, LOCK_TYPE, NOTIFY_TYPE
  - Atomic kind type parameters: ATOMIC_INT_KIND and ATOMIC_LOGICAL_KIND
  - Values: STAT_FAILED_IMAGE, STAT_LOCKED, STAT_LOCKED_OTHER_IMAGE, STAT_STOPPED_IMAGE, STAT_UNLOCKED, STAT_UNLOCKED_FAILED_IMAGE

Overview

01

Background:

Deep origins

02

Features & Paradigms:

A walking tour from Fortran 90 to 2023 & beyond

03

Use Cases:

Nooks, Crannies, and Pastures

04

Conclusions

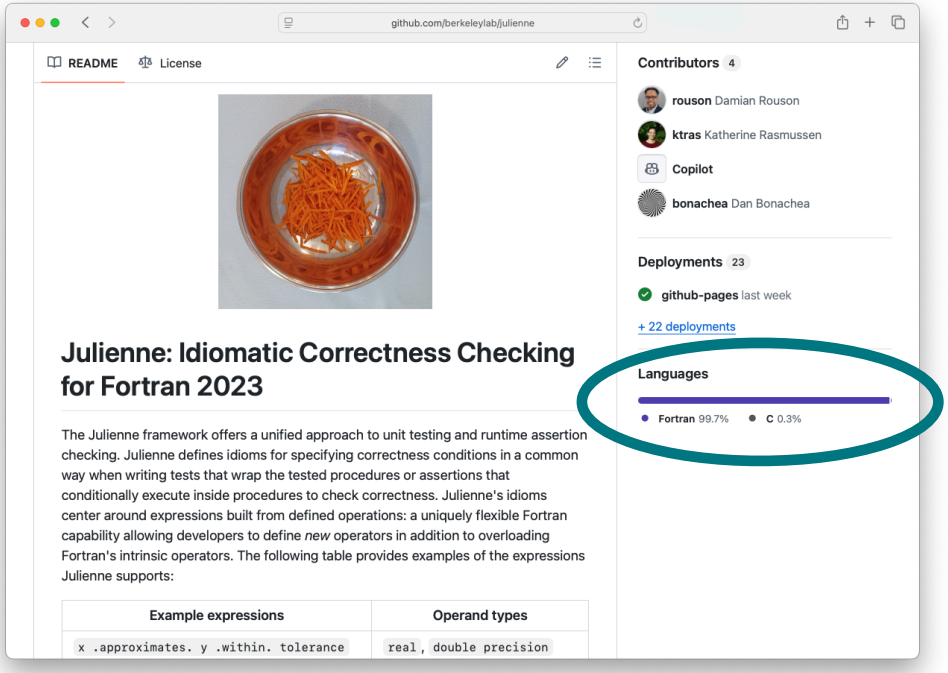
Correctness-Checking with Julienne

Unified Idioms for writing

- Unit tests
- Assertions

Support for Fortran 2023 native parallelism

- Multi-image testing: a collective reduction detects failure on a subset of images
- Assertions are `pure` procedures as required for invocation inside a `do concurrent` construct.



https://go.lbl.gov/julienne

Rouson, Bonachea, & Rasmussen, "Idiomatic Correctness-Checking via Julienne in Fortran 2023", *Proceedings of the US Research Software Engineering Conference*, October 2025. DOI: 10.25344/S4BG65

Julienne Idioms

Row	Example expressions	Supported operand types of the bold operator
1	`x .approximates. y .within. tolerance`	`real, double precision`
2	`x .approximates. y .withinFraction. tolerance`	`real, double precision`
3	`x .approximates. y .withinPercentage. tolerance`	`real, double precision`
4	`.all. ([i,j] .lessThan. k)`	`test_diagnosis_t`
5	`.all. ([i,j] .lessThan. [k,m])`	`test_diagnosis_t`
6	`.all. (i .lessThan. [k,m])`	`test_diagnosis_t`
7	`(i .lessThan. j) .also. (k .equalsExpected. m)`	`test_diagnosis_t`
8	`x .lessThan. y`	`integer, real, double precision`
9	`x .greaterThan. y`	`integer, real, double precision`
10	`i .equalsExpected. j`	`integer, character, type(c_ptr)`
11	`i .isAtLeast. j`	`integer, real, double precision`
12	`i .isAtMost. j`	`integer, real, double precision`
13	`s .isBefore. t`	`character`
14	`s .isAfter. t`	`character`
15	`(.expect. allocated(A)) // '(expected an allocated array "A")'`	`logical`

`Elemental` Operators:

- Defined as `pure` functions
- Binary operators accept conformable operands:
  - Same-shaped arrays
  - Scalar/array combinations

⇒

Any 1 elemental function defining a binary operator accepts 46 combinations of operands:

- 1 for scalar operands
- 15 for array operands of rank 1-15
- 30 for scalar/array combinations in either order

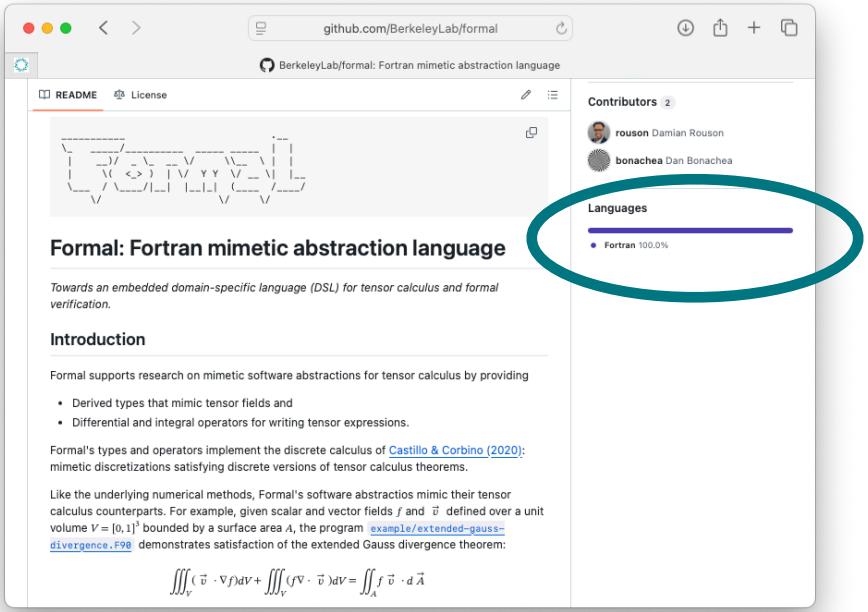
Writing PDE Solvers with Formal

Fortran mimetic abstraction language

- Derived types that mimic tensor fields
- Differential and integral operators
- Supported by a discrete calculus based on the mimetic discretizations of Corbino & Castillo (2020).

Future work

- An embedded domain-specific language (DSL) for tensor calculus
- Formal verification leveraging problem-specific proof by testing
- Tensor contractions for machine learning



https://go.lbl.gov/formal

Corbino, J., & Castillo, J. E. (2020). “High-order mimetic finite-difference operators satisfying the extended Gauss divergence theorem”. *Journal of Computational and Applied Mathematics*, 364, 112326.

Please, No More Loops (Than Necessary)

“When writing type-safe templates in Fortran, you can consider the requirements as defining a DSL for the template body. Such DSLs are extremely cheap to define, just a collection of derived declarations, and have no runtime overhead.”

Prof. Magne Haveraaen
Bergen Language Design Laboratory
University of Bergen

REQUIREMENT Construct

```
REQUIREMENT binop(op, T, U, V)
  DEFERRED TYPE :: T, U, V
  DEFERRED INTERFACE
    FUNCTION op(x,y) RESULT(z)
      TYPE(T), INTENT(IN) :: x
      TYPE(U), INTENT(IN) :: y
      TYPE(V) :: z
    END FUNCTION
  END INTERFACE
END REQUIREMENT
```

A requirement encapsulates a reusable relationship among deferred arguments.

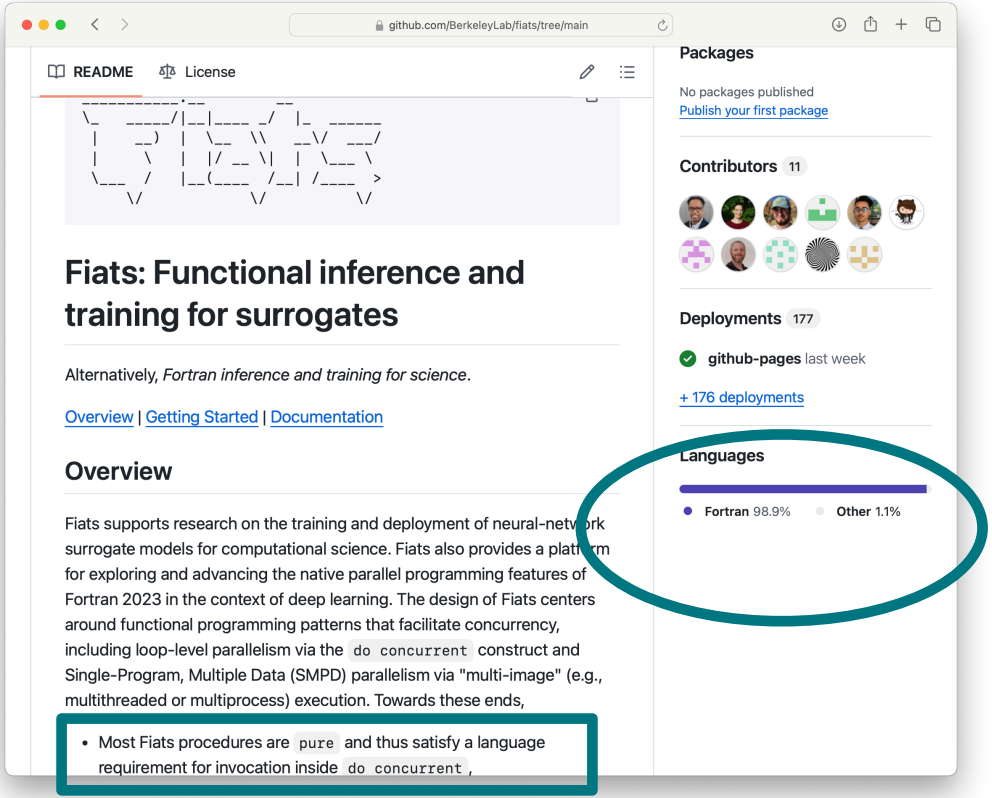
The **REQUIRE** statement enforces a REQUIREMENT

- Mismatch at template instantiation is compile-time error
- Transitively declares its arguments
- Can appear in template specification and requirement construct

```
TEMPLATE my_templ(T, U, plus, times)
   USE requirements_mod, only: binop

   REQUIRE binop(plus, T, U, U)  ! Real+complex -> complex
   REQUIRE binop(times, T, U, U) ! Real*complex -> complex
   …
END TEMPLATE
```

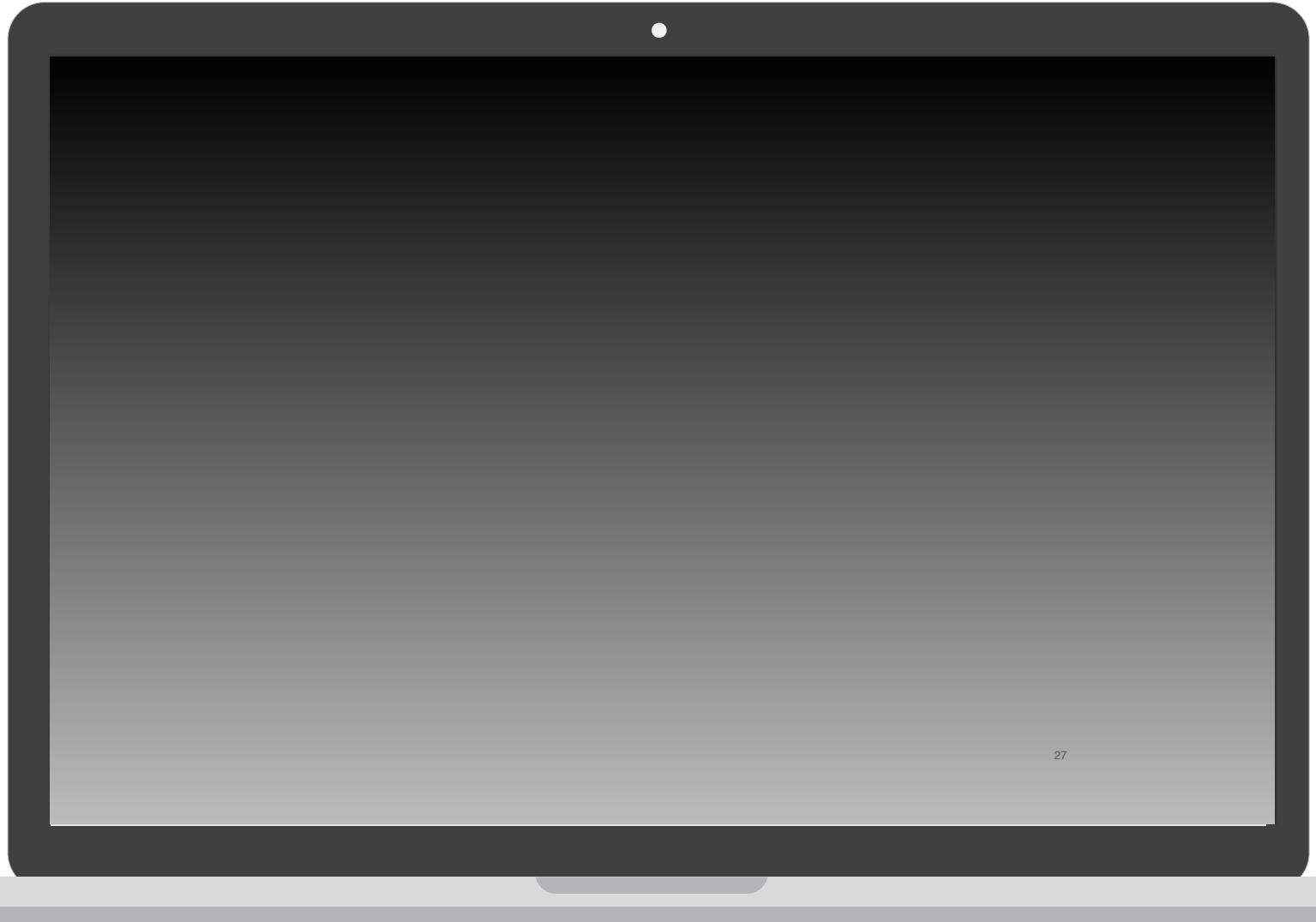
Deep Learning with Fiats



Please, No More Loops (Than Necessary)

BERKELEY LAB
Bringing Science Solutions to the World

Fiats: Inference



Fiats: Inference

example — vim concurrent-inferences.f90 — 90×9

```
129     !$omp parallel do default(none) shared(neural_network,inputs,outputs) collapse(3)
130     do j=1,lon
131       do k=1,lev
132         do i=1,lat
133           outputs(i,k,j) = neural_network%infer(inputs(i,k,j))
134         end do
135       end do
136     end do
```

Fiats: Inference

example — vim concurrent-inferences.f90 — 90×9

```
129     !$omp parallel do default(none) shared(neural_network,inputs,outputs) collapse(3)
130     do j=1,lon
131       do k=1,lev
132         do i=1,lat
133           outputs(i,k,j) = neural_network%infer(inputs(i,k,j))
134         end do
135       end do
136     end do
```

fiats — rouson@login38:~/fiats — login38 — vim example/concurrent-inferences.f90 — 103×5

```
122     do concurrent(i=1:lat, k=1:lev, j=1:lon) default(none) shared(outputs, neural_network, inputs)
123       outputs(i,k,j) = neural_network%infer(inputs(i,k,j))
124     end do
```

Fiats: Inference

example — vim concurrent-inferences.f90 — 90×9

```
129     !$omp parallel do default(none) shared(neural_network,inputs,outputs) collapse(3)
130     do j=1,lon
131       do k=1,lev
132         do i=1,lat
133           outputs(i,k,j) = neural_network%infer(inputs(i,k,j))
134         end do
135       end do
136     end do
```

fiats — rouson@login38:~/fiats — login38 — vim example/concurrent-inferences.f90 — 103×5

```
122     do concurrent(i=1:lat, k=1:lev, j=1:lon) default(none) shared(outputs, neural_network, inputs)
123       outputs(i,k,j) = neural_network%infer(inputs(i,k,j))
124     end do
```

example — vim concurrent-inferences.f90 — 50×5

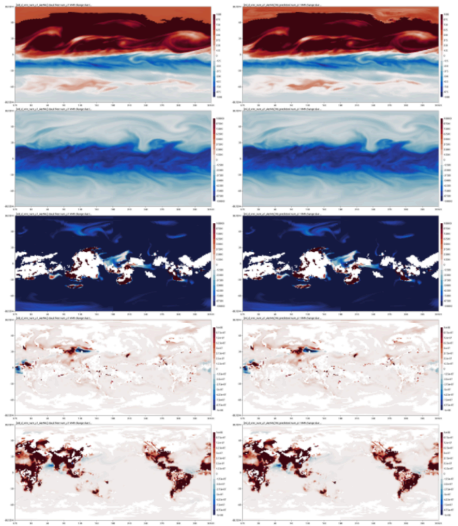
```
73      !$omp workshare
74      outputs = neural_network%infer(inputs)
75      !$omp end workshare
```

Automatic Parallelization on Perlmutter CPU

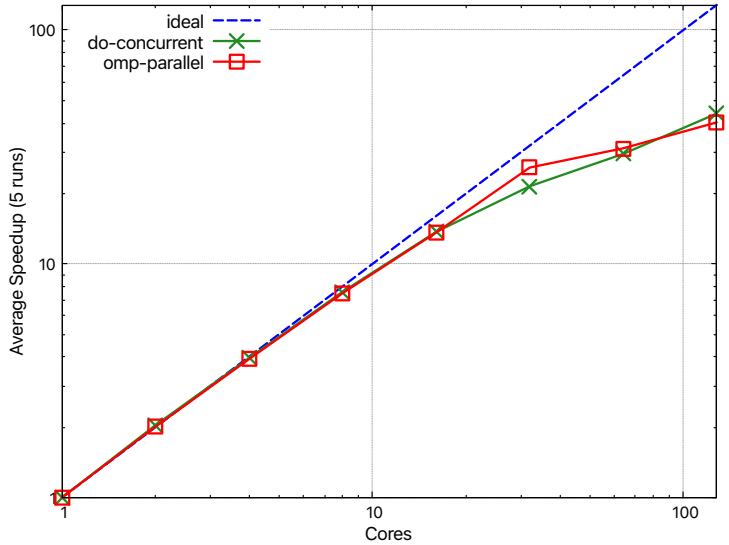
Automatically parallelizing batch inference on deep neural networks using Fiats and Fortran 2023 "do concurrent"

Damian Rouson[1], Zhe Bai[1], Dan Bonachea[1], Kareem Ergawy[2], Ethan Gutmann[3], Michael Klemm[2], Katherine Rasmussen[1], Brad Richardson[1], Sameer Shende[4], David Torres[5], and Yunhao Zhang[1]

[1] Lawrence Berkeley National Laboratory, Berkeley CA 94720, USA



applications have evaluated or adopted deep neural networks as surrogate models. At least two categories of solutions have emerged to satisfy the inference and training needs of Fortran applications: (1) application programming interfaces (APIs) that expose functionality provided by software packages written




```
OMP_NUM_THREADS=128 fpm run \
  --example concurrent-inferences \
  --runner "srun --cpu_bind=cores -c 128 -n 1" \
  — --network model.json
```

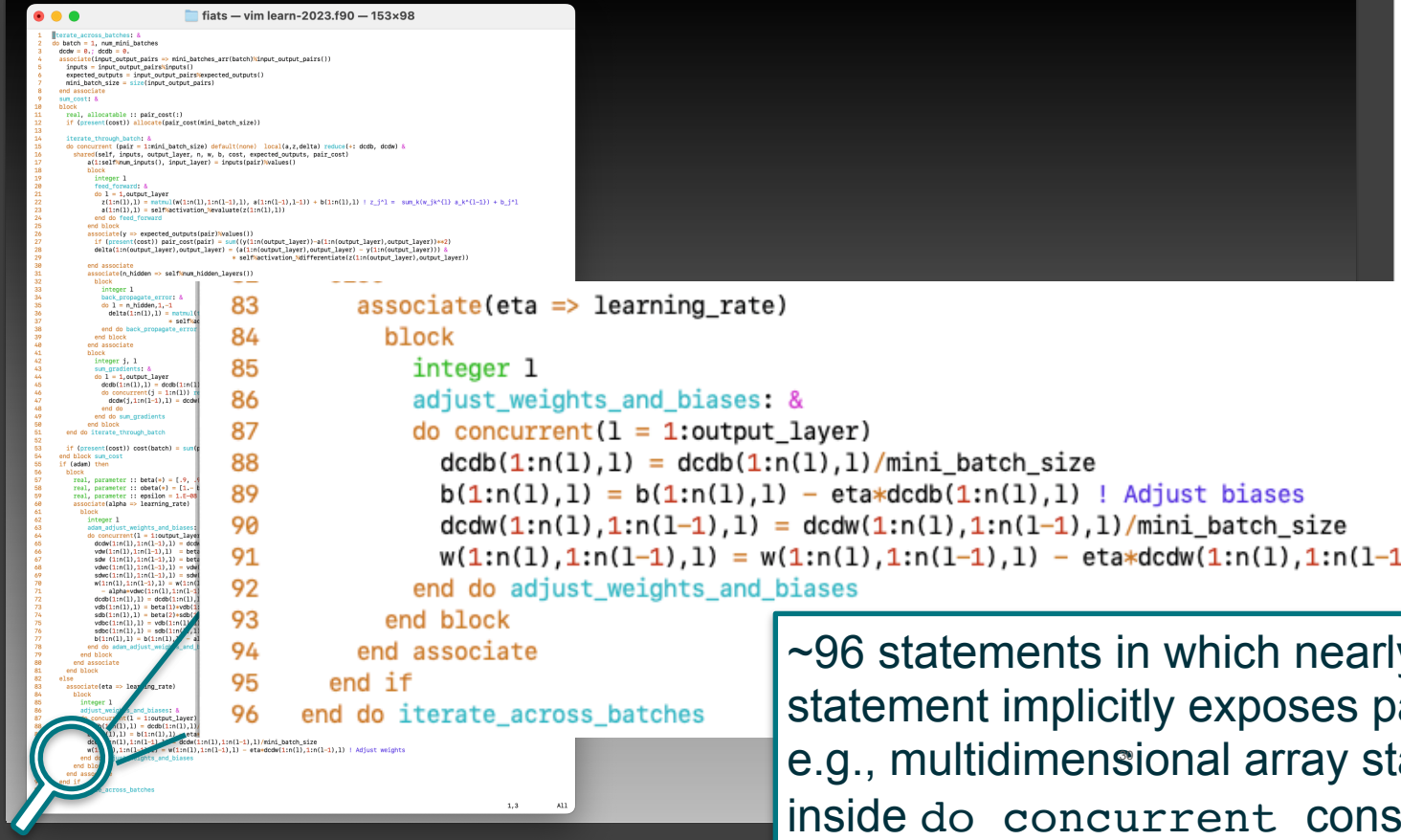
Rouson, Bai, Bonachea, Ergawy, Gutmann, Klemm, Rasmussen, Richardson, Shende, Torres, and Zhang (2025). Automatically parallelizing batch inference on deep neural networks using Fiats and Fortran 2023 "do concurrent". In *5th International Workshop on Computational Aspects of Deep Learning*, Hamburg, Germany.

Fiats: Training

fiats — vim neural_network_s.F90 — 101×36

```
907 #if F2023_LOCALITY
908               iterate_through_batch: &
909               do concurrent (pair = 1:mini_batch_size) local(a,z,delta) reduce(+: dcdb, dcdw)
910
911 #elif F2018_LOCALITY
912
913               reduce_gradients: &
914               block
915                 real reduce_dcdb(size(dcdb,1),size(dcdb,2),mini_batch_size)
916                 real reduce_dcdw(size(dcdw,1),size(dcdw,2),size(dcdw,3),mini_batch_size)
917                 reduce_dcdb = 0.
918                 reduce_dcdw = 0.
919
920                 iterate_through_batch: &
921                 do concurrent (pair = 1:mini_batch_size) local(a,z,delta)
922
923 #else
924
925               reduce_gradients: &
926               block
927                 real reduce_dcdb(size(dcdb,1),size(dcdb,2),mini_batch_size)
928                 real reduce_dcdw(size(dcdw,1),size(dcdw,2),size(dcdw,3),mini_batch_size)
929                 reduce_dcdb = 0.
930                 reduce_dcdw = 0.
931
932                 iterate_through_batch: &
933                 do concurrent (pair = 1:mini_batch_size)
934
935                   iteration: &
936                   block
937
938                     real a(maxval(self%nodes_), input_layer:output_layer) ! Activations
939                     real z(size(b,1),size(b,2)), delta(size(b,1),size(b,2))
940 #endif
941
                                                                    910,0-1       89%
```

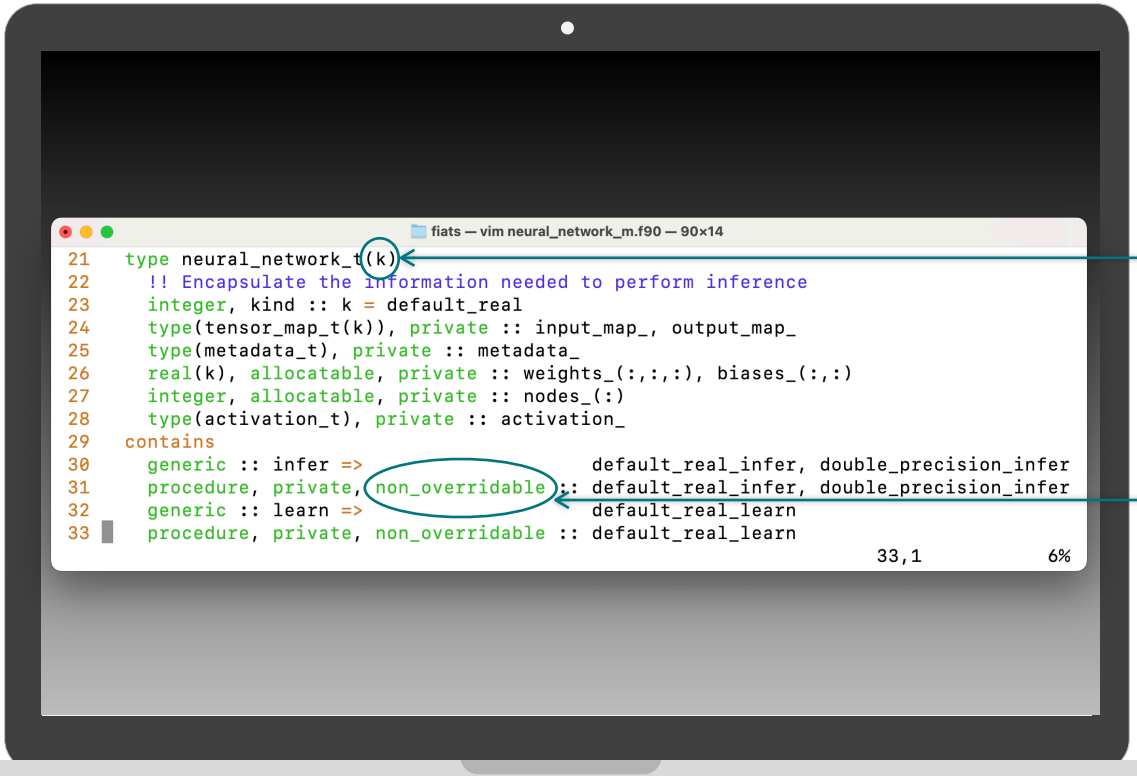
Stochastic Gradient Descent + Adam Optimizer




```
83        associate(eta => learning_rate)
84          block
85            integer l
86            adjust_weights_and_biases: &
87            do concurrent(l = 1:output_layer)
88              dcdb(1:n(l),l) = dcdb(1:n(l),l)/mini_batch_size
89              b(1:n(l),l) = b(1:n(l),l) - eta*dcdb(1:n(l),l) ! Adjust biases
90              dcdw(1:n(l),1:n(l-1),l) = dcdw(1:n(l),1:n(l-1),l)/mini_batch_size
91              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
92            end do adjust_weights_and_biases
93          end block
94        end associate
95      end if
96    end do iterate_across_batches
```

~96 statements in which nearly every statement implicitly exposes parallelism, e.g., multidimensional array statements inside `do concurrent` constructs

Deep Learning with Fiats

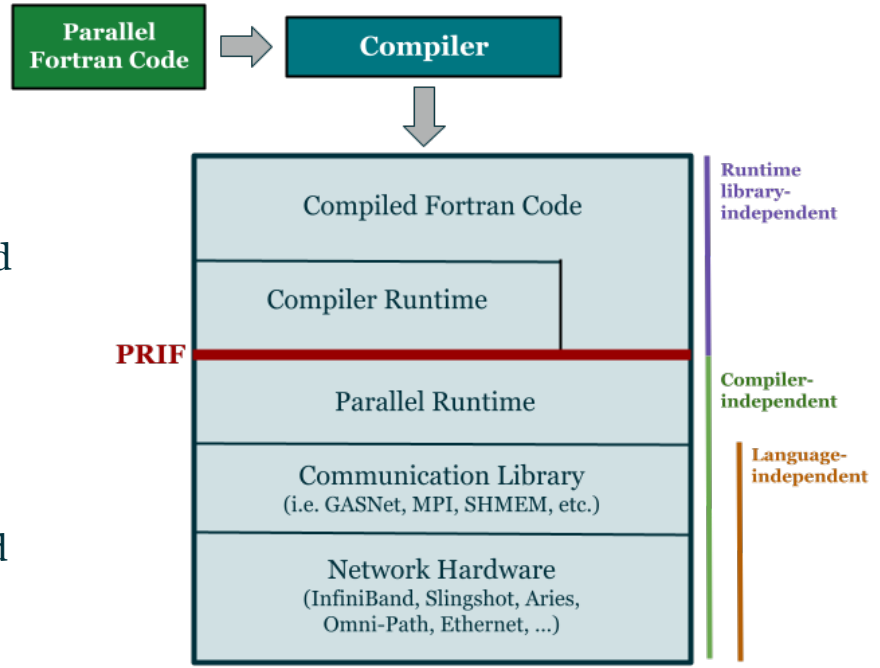


Kind type parameter allows us to set an object's precision in its declaration *without* recompiling.

`Non_overridable` attribute prevents dynamic dispatch, thereby facilitating *future* GPU execution.

Parallel Runtime Interface for Fortran (PRIF)

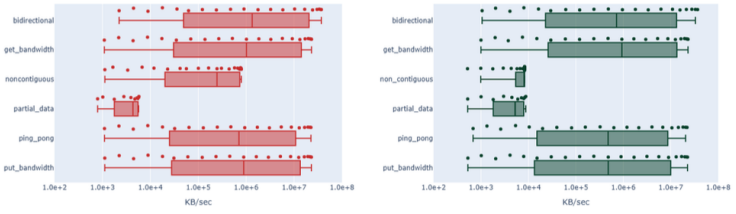
- Compiler- and runtime-agnostic interface to support multi-image parallel Fortran features
- A runtime interface written *in Fortran:* prif module
- Tight correspondence between PRIF procedures and Fortran's multi-image parallel features, e.g.,
  - num_images ⟶ prif_num_images
  - real x(N)[*] ⟶ prif_allocate_coarray
- For more information, please see go.lbl.gov/prif and fortran.lbl.gov.



D. Bonachea, K. Rasmussen, B. Richardson, D. Rouson, "Parallel Runtime Interface for Fortran (PRIF): A Multi-Image Solution for LLVM Flang", *Tenth Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC2024)*, Nov. 2024. doi:10.25344/S4N017.

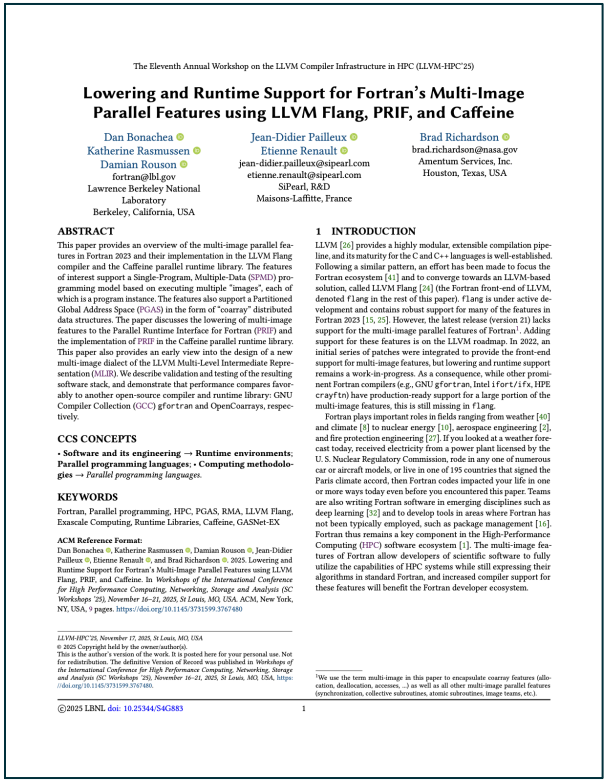
LLVM-HPC Workshop at SC25 Paper

- Paper highlights the increased LLVM Flang compiler support for Fortran's multi-image features, a subset of which has now been upstreamed, thanks to the support of the NERSC/CLaSS collaboration
- Perlmutter runs in distributed memory show LLVM Flang is comparable with Cray's long extant multi-image Fortran support
- Cray ftn compiler bug encountered while compiling a coarray benchmark on Perlmutter: [NERSC ticket INC0241058]



(a) `flang` and Caffeine (b) HPE Cray Fortran

Figure 5: Results from running Caf-testsuite in distributed memory across two nodes of the NERSC Perlmutter supercomputer, connected over an HPE Slingshot 11 network



The Eleventh Annual Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC'25)

Lowering and Runtime Support for Fortran's Multi-Image Parallel Features using LLVM Flang, PRIF, and Caffeine

Dan Bonachea
Katherine Rasmussen
Damian Rouson
fortran@lbl.gov
Lawrence Berkeley National Laboratory
Berkeley, California, USA

Jean-Didier Pailleux
Etienne Renault
jean-didier.pailleux@sipearl.com
etienne.renault@sipearl.com
SiPearl, R&D
Maisons-Laffitte, France

Brad Richardson
brad.richardson@nasa.gov
Amentum Services, Inc.
Houston, Texas, USA

ABSTRACT

This paper provides an overview of the multi-image parallel features in Fortran 2023 and their implementation in the LLVM Flang compiler and the Caffeine parallel runtime library. The features of interest support a Single-Program, Multiple-Data (SPMD) programming model based on executing multiple "images", each of which is a program instance. The features also support a Partitioned Global Address Space (PGAS) in the form of "coarray" distributed data structures. The paper discusses the lowering of multi-image features to the Parallel Runtime Interface for Fortran (PRIF) and the implementation of PRIF in the Caffeine parallel runtime library. This paper also provides an early view into the design of a new multi-image dialect of the LLVM Multi-Level Intermediate Representation (MLIR). We describe validation and testing of the resulting software stack, and demonstrate that performance compares favorably to another open-source compiler and runtime library: GNU Compiler Collection (GCC) `gfortran` and OpenCoarrays, respectively.

CCS CONCEPTS

• **Software and its engineering** → **Runtime environments**; **Parallel programming languages**; • **Computing methodologies** → *Parallel programming languages*.

KEYWORDS

Fortran, Parallel programming, HPC, PGAS, RMA, LLVM Flang, Exascale Computing, Runtime Libraries, Caffeine, GASNet-EX

ACM Reference Format:

Dan Bonachea, Katherine Rasmussen, Damian Rouson, Jean-Didier Pailleux, Etienne Renault, and Brad Richardson. 2025. Lowering and Runtime Support for Fortran's Multi-Image Parallel Features using LLVM Flang, PRIF, and Caffeine. In *Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC Workshops '25), November 16–21, 2025, St Louis, MO, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3731599.3767480

1 INTRODUCTION

LLVM [26] provides a highly modular, extensible compilation pipeline, and its maturity for the C and C++ languages is well-established. Following a similar pattern, an effort has been made to focus the Fortran ecosystem [41] and to converge towards an LLVM-based solution, called LLVM Flang [24] (the Fortran front-end of LLVM, denoted `flang` in the rest of this paper). `flang` is under active development and contains robust support for many of the features in Fortran 2023 [15, 25]. However, the latest release (version 21) lacks support for the multi-image parallel features of Fortran[1]. Adding support for these features is on the LLVM roadmap. In 2022, an initial series of patches were integrated to provide the front-end support for multi-image features, but lowering and runtime support remains a work-in-progress. As a consequence, while other prominent Fortran compilers (e.g., GNU `gfortran`, Intel `ifort`/`ifx`, HPE `crayftn`) have production-ready support for a large portion of the multi-image features, this is still missing in `flang`.

Fortran plays important roles in fields ranging from weather [40] and climate [8] to nuclear energy [10], aerospace engineering [2], and fire protection engineering [27]. If you looked at a weather forecast today, received electricity from a power plant licensed by the U. S. Nuclear Regulatory Commission, rode in any one of numerous car or aircraft models, or live in one of 195 countries that signed the Paris climate accord, then Fortran codes impacted your life in one or more ways today even before you encountered this paper. Teams are also writing Fortran software in emerging disciplines such as deep learning [32] and to develop tools in areas where Fortran has not been typically employed, such as package management [16]. Fortran thus remains a key component in the High-Performance Computing (HPC) software ecosystem [1]. The multi-image features of Fortran allow developers of scientific software to fully utilize the capabilities of HPC systems while still expressing their algorithms in standard Fortran, and increased compiler support for these features will benefit the Fortran developer ecosystem.

[1]We use the term multi-image in this paper to encapsulate coarray features (allocation, deallocation, accesses, ...) as well as all other multi-image parallel features (synchronization, collective subroutines, atomic subroutines, image teams, etc.).

LLVM-HPC'25, November 17, 2025, St Louis, MO, USA

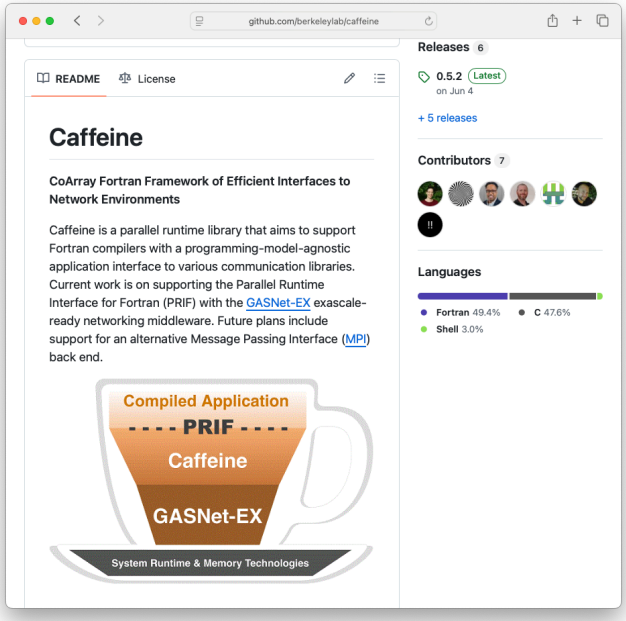
©2025 LBNL. doi: 10.25344/S4G883

Dan Bonachea, Katherine Rasmussen, Damian Rouson, Jean-Didier Pailleux, Etienne Renault, Brad Richardson. "Lowering and Runtime Support for Fortran's Multi-Image Parallel Features using LLVM Flang, PRIF, and Caffeine", Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC Workshops '25), November 16–21, 2025, St Louis, MO, USA. ACM, New York, NY, USA, 9 pages. https://doi.org/10.25344/S4G883

Caffeine: Co-Array Fortran Framework of Efficient Interfaces to Network Environments

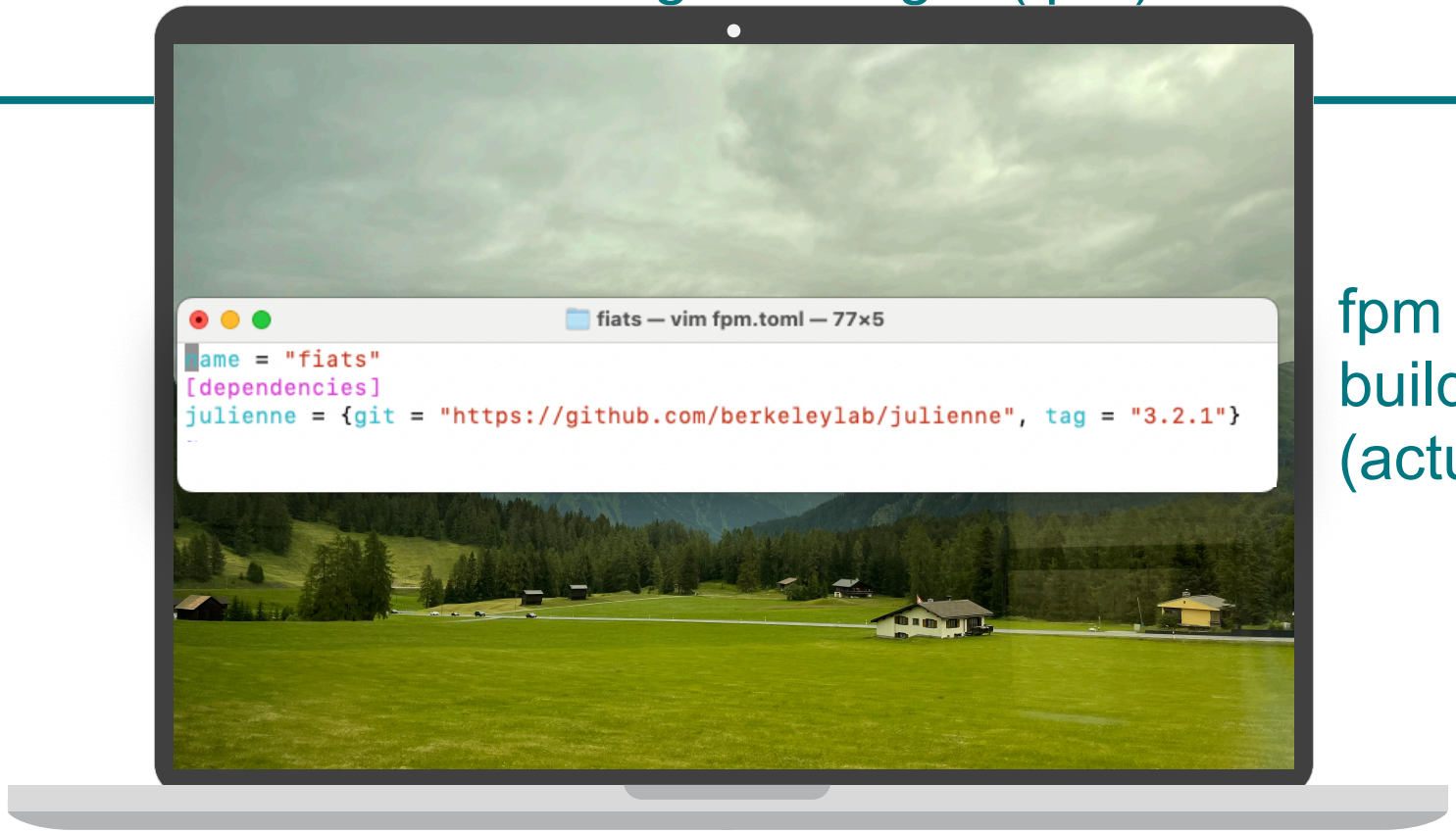
- Caffeine is written mostly in (serial) Fortran
- Invokes GASNet-EX communication library
- PRIF implementation status: go.lbl.gov/caffeine-status

Multi-image Fortran Feature	Status
Program startup and shutdown (incl. normal and error termination): `STOP`, `ERROR STOP`, `END PROGRAM` statements	yes
Collective subroutines: `CO_{BROADCAST,SUM,MIN,MAX,REDUCE}`	yes
Image queries: `THIS_IMAGE`, `NUM_IMAGES`, etc, intrinsic functions	yes
Synchronization: `SYNC {ALL,IMAGES,MEMORY,TEAM}` statements	yes
Storage management: Coarray allocation, deallocation and coarray aliases	yes
Coarray Queries: `LCOBOUND`, `UCOBOUND`, `COSHAPE`, etc.	yes
Contiguous and strided coarray access: Coarray puts and gets	yes
Teams: `TEAM_TYPE` intrinsic type and `{FORM,CHANGE,END} TEAM` statements	yes
Events: `EVENT_TYPE` intrinsic type, `EVENT_QUERY` subroutine and `EVENT {POST,WAIT}` statements	yes
Notifications: `NOTIFY_TYPE` intrinsic type and `NOTIFY WAIT` statement	yes
Atomics: `ATOMIC_{INT,LOGICAL}_KIND` kind parameters and `ATOMIC_{DEFINE,REF,...}` subroutines	yes
Critical construct: `CRITICAL` and `END CRITICAL`	no
Locks: `LOCK` and `UNLOCK` statements	no
`FAIL IMAGE` statement	no



go.lbl.gov/caffeine

Fortran Package Manager (fpm)



fpm
build system
(actual size)

Overview

01

Background:

Deep origins

02

Features & Paradigms:

A walking tour from Fortran 90 to 202Y+

03

Use Cases:

Nooks, Crannies, and Pastures

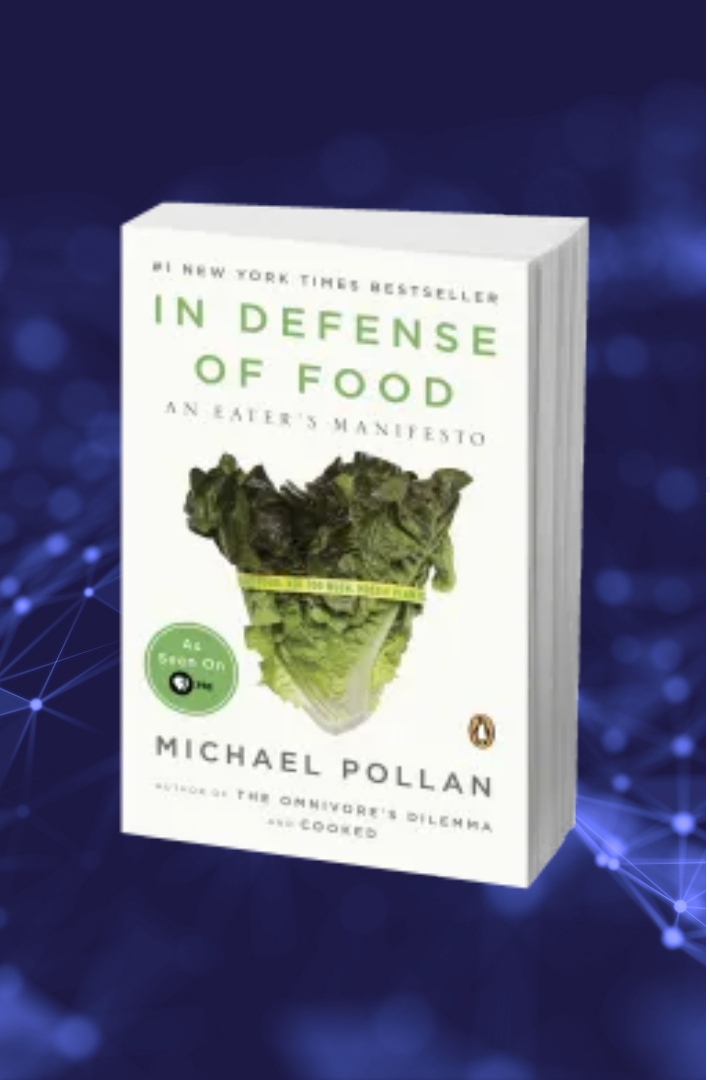
04

Conclusions

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).



In Defense of Food: An Eater's Manifesto

Eat food.

Not too much.

Mostly plants.

In Defense of Software: A Developer's Manifesto

Write software.

Not too much.

Mostly pure functions.

Conclusions

"Fortran [2023] is a new and exciting language used by programmers to communicate with [each other]."

Several underutilized feature sets facilitate writing

- Compact code:
  - 46-fold savings in supporting binary operators with elemental functions
  - State-of-the-art neural network training in fewer than 100 lines of code
- Parallel programs:
  - Multi-image execution for SPMD/PGAS programming
  - Automatic loop-level multithreading or offloading to a GPU
- Functional programming patterns:
  - Pure procedures
  - Immutable state: associate construct
- Expressive abstractions:
  - Natural language idioms
  - Textbook forms of partial differential equations

Thank You!