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
https://go.lbl.gov/fortran

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

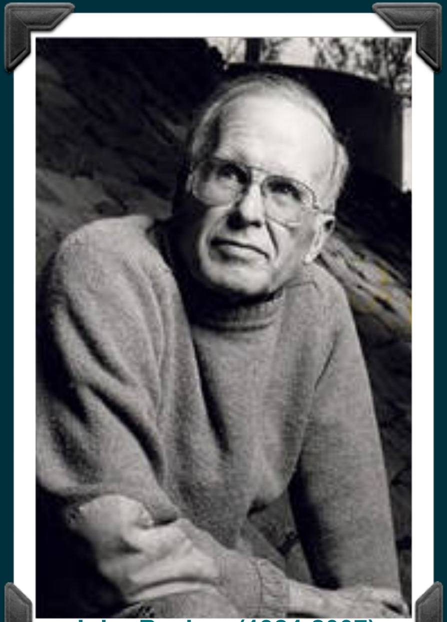
03

Use Cases:

Nooks, Crannies, and Pastures

04

Conclusions

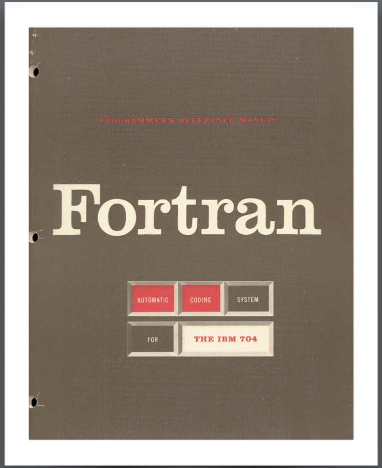


John Backus (1924-2007)

Pioneers in Science and Technology Series: John Backus, 1984

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

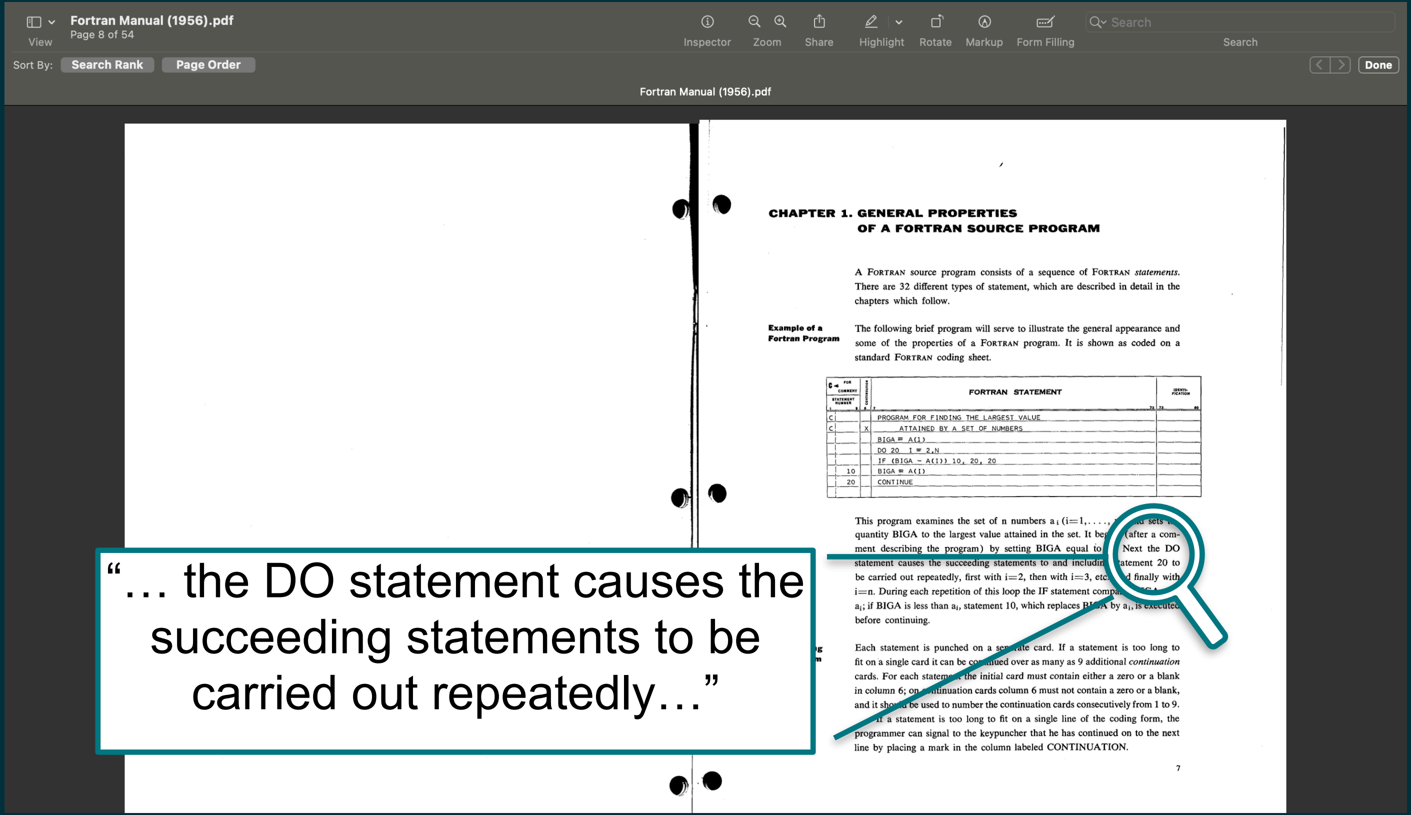
1956



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

BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY
Office of Science



CHAPTER 1. GENERAL PROPERTIES OF A FORTRAN SOURCE PROGRAM

A FORTRAN source program consists of a sequence of FORTRAN *statements*. There are 32 different types of statement, which are described in detail in the chapters which follow.

Example of a Fortran Program

The following brief program will serve to illustrate the general appearance and some of the properties of a FORTRAN program. It is shown as coded on a standard FORTRAN coding sheet.

C FOR COMMENT / STATEMENT NUMBER	CONTINUATION	FORTRAN STATEMENT	IDENTIFICATION
C		PROGRAM FOR FINDING THE LARGEST VALUE	
C	X	ATTAINED BY A SET OF NUMBERS	
		BIGA = A(1)	
		DO 20 I = 2,N	
		IF (BIGA - A(I)) 10, 20, 20	
10		BIGA = A(I)	
20		CONTINUE	

This program examines the set of n numbers a_i (i=1,. . . ., [illegible] sets [illegible] quantity BIGA to the largest value attained in the set. It beg[illegible] (after a comment describing the program) by setting BIGA equal to [illegible] Next the DO statement causes the succeeding statements to and includin[illegible] statement 20 to be carried out repeatedly, first with i=2, then with i=3, etc. [illegible] finally with i=n. During each repetition of this loop the IF statement compa[illegible] a_i; if BIGA is less than a_i, statement 10, which replaces B[illegible] by a_i, is executed before continuing.

Each statement is punched on a se[illegible] card. If a statement is too long to fit on a single card it can be co[illegible] over as many as 9 additional *continuation* cards. For each stateme[illegible] the initial card must contain either a zero or a blank in column 6; on [illegible]tinuation cards column 6 must not contain a zero or a blank, and it sho[illegible] be used to number the continuation cards consecutively from 1 to 9. [illegible] a statement is too long to fit on a single line of the coding form, the programmer can signal to the keypuncher that he has continued on to the next line by placing a mark in the column labeled CONTINUATION.

7

"… 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 (Derived types; Structure constructors; Derived type input/output (I/O); Data and procedure privacy; Pointers)
- Functional programming (Defined operations and defined assignments; Recursive procedures)
- Array programming (Allocatable arrays; Array statements, structure constructors, and intrinsic functions)
- Modular programming (Modules; Interface blocks & bodies, including generic interfaces)
- Structured programming (Structured branching and looping)

New Features

Paradigms

90

95

- `pure` procedures } Functional programming
- `elemental` procedures } Array programming
- `where` construct } Array programming
- `forall` construct (obsolescent) } Parallel/vector programming

New Features

90 | 95 | 2003

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

90 | 95 | 2003 | 2008

- 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

Paradigms

90 | 95 | 2003 | 2008 | 2018

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()`

(C-Interoperability)

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

Parallel/vector programming

New Features — Paradigms

90 | 95 | 2003 | 2008 | 2018 | 2023

2023 ↓

`do concurrent reduce` locality

put notifications: `notify_type`, `notify wait`

} Parallel/vector programming

enumeration types

maximum statement length: 1 million characters

New Features → Paradigms

Type-safe templates
Generic subroutines
} Generic programming

Enhanced collective subroutines: team collectives, asynchronous collectives
} Parallel programming

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/r

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**

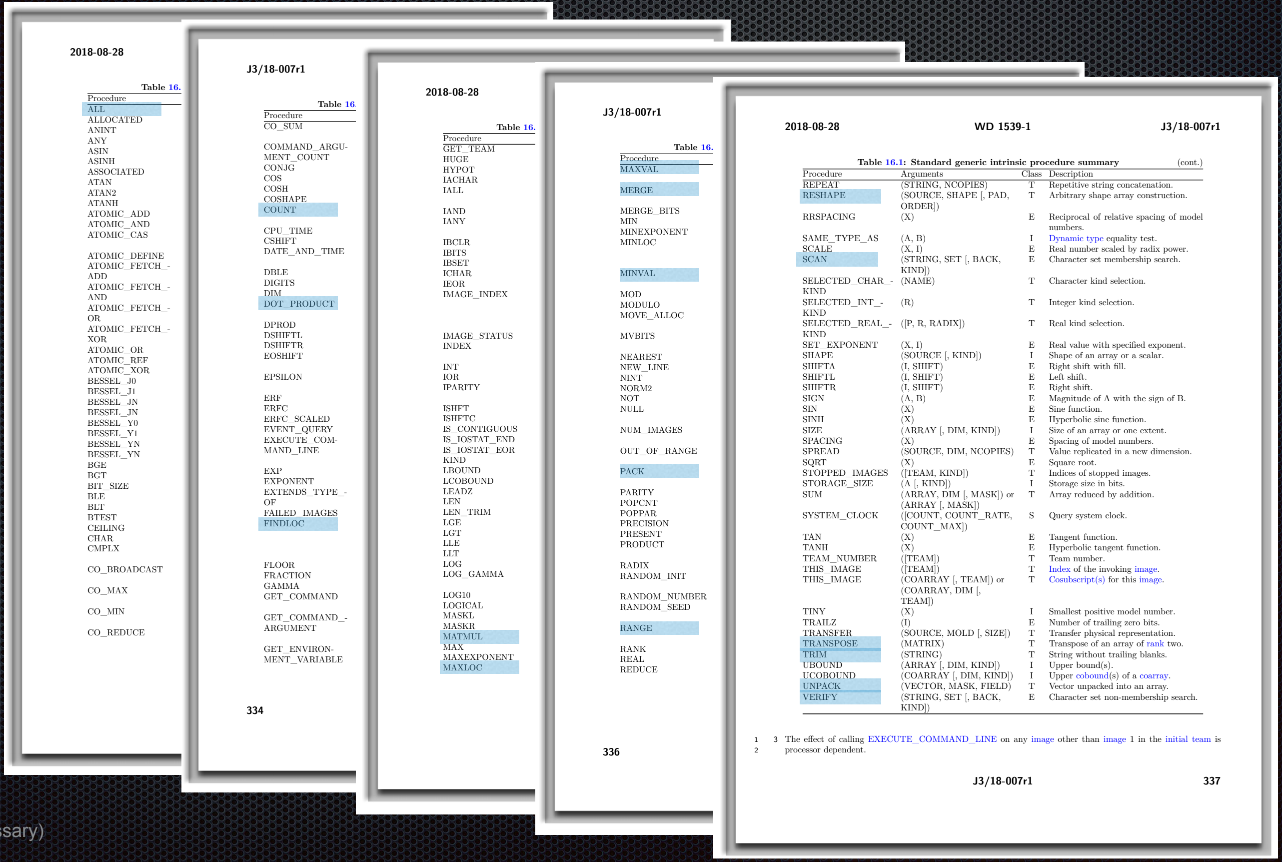
Synonyms of *cranny* >

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

2 : an obscure nook or corner

crannied ˈkra-nēd *adjective*

Fortran 2018 Intrinsic Functions



2018-08-28

Table 16.

Procedure
ALL
ALLOCATED
ANINT
ANY
ASIN
ASINH
ASSOCIATED
ATAN
ATAN2
ATANH
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
BESSEL_J0
BESSEL_J1
BESSEL_JN
BESSEL_JN
BESSEL_Y0
BESSEL_Y1
BESSEL_YN
BESSEL_YN
BGE
BGT
BIT_SIZE
BLE
BLT
BTEST
CEILING
CHAR
CMPLX
CO_BROADCAST
CO_MAX
CO_MIN
CO_REDUCE

J3/18-007r1

Table 16

Procedure
CO_SUM
COMMAND_ARGU-MENT_COUNT
CONJG
COS
COSH
COSHAPE
COUNT
CPU_TIME
CSHIFT
DATE_AND_TIME
DBLE
DIGITS
DIM
DOT_PRODUCT
DPROD
DSHIFTL
DSHIFTR
EOSHIFT
EPSILON
ERF
ERFC
ERFC_SCALED
EVENT_QUERY
EXECUTE_COM-MAND_LINE
EXP
EXPONENT
EXTENDS_TYPE_-OF
FAILED_IMAGES
FINDLOC
FLOOR
FRACTION
GAMMA
GET_COMMAND
GET_COMMAND_-ARGUMENT
GET_ENVIRON-MENT_VARIABLE

334

2018-08-28

Table 16.

Procedure
GET_TEAM
HUGE
HYPOT
IACHAR
IALL
IAND
IANY
IBCLR
IBITS
IBSET
ICHAR
IEOR
IMAGE_INDEX
IMAGE_STATUS
INDEX
INT
IOR
IPARITY
ISHFT
ISHFTC
IS_CONTIGUOUS
IS_IOSTAT_END
IS_IOSTAT_EOR
KIND
LBOUND
LCOBOUND
LEADZ
LEN
LEN_TRIM
LGE
LGT
LLE
LLT
LOG
LOG_GAMMA
LOG10
LOGICAL
MASKL
MASKR
MATMUL
MAX
MAXEXPONENT
MAXLOC

J3/18-007r1

Table 16.

Procedure
MAXVAL
MERGE
MERGE_BITS
MIN
MINEXPONENT
MINLOC
MINVAL
MOD
MODULO
MOVE_ALLOC
MVBITS
NEAREST
NEW_LINE
NINT
NORM2
NOT
NULL
NUM_IMAGES
OUT_OF_RANGE
PACK
PARITY
POPCNT
POPPAR
PRECISION
PRESENT
PRODUCT
RADIX
RANDOM_INIT
RANDOM_NUMBER
RANDOM_SEED
RANGE
RANK
REAL
REDUCE

336

2018-08-28 WD 1539-1 J3/18-007r1

Table 16.1: Standard generic intrinsic procedure summary (cont.)

Procedure	Arguments	Class	Description
REPEAT	(STRING, NCOPIES)	T	Repetitive string concatenation.
RESHAPE	(SOURCE, SHAPE [, PAD, ORDER])	T	Arbitrary shape array construction.
RRSPACING	(X)	E	Reciprocal of relative spacing of model numbers.
SAME_TYPE_AS	(A, B)	I	Dynamic type equality test.
SCALE	(X, I)	E	Real number scaled by radix power.
SCAN	(STRING, SET [, BACK, KIND])	E	Character set membership search.
SELECTED_CHAR_-KIND	(NAME)	T	Character kind selection.
SELECTED_INT_-KIND	(R)	T	Integer kind selection.
SELECTED_REAL_-KIND	([P, R, RADIX])	T	Real kind selection.
SET_EXPONENT	(X, I)	E	Real value with specified exponent.
SHAPE	(SOURCE [, KIND])	I	Shape of an array or a scalar.
SHIFTA	(I, SHIFT)	E	Right shift with fill.
SHIFTL	(I, SHIFT)	E	Left shift.
SHIFTR	(I, SHIFT)	E	Right shift.
SIGN	(A, B)	E	Magnitude of A with the sign of B.
SIN	(X)	E	Sine function.
SINH	(X)	E	Hyperbolic sine function.
SIZE	(ARRAY [, DIM, KIND])	I	Size of an array or one extent.
SPACING	(X)	E	Spacing of model numbers.
SPREAD	(SOURCE, DIM, NCOPIES)	T	Value replicated in a new dimension.
SQRT	(X)	E	Square root.
STOPPED_IMAGES	([TEAM, KIND])	T	Indices of stopped images.
STORAGE_SIZE	(A [, KIND])	I	Storage size in bits.
SUM	(ARRAY, DIM [, MASK]) or (ARRAY [, MASK])	T	Array reduced by addition.
SYSTEM_CLOCK	([COUNT, COUNT_RATE, COUNT_MAX])	S	Query system clock.
TAN	(X)	E	Tangent function.
TANH	(X)	E	Hyperbolic tangent function.
TEAM_NUMBER	([TEAM])	T	Team number.
THIS_IMAGE	([TEAM])	T	Index of the invoking image.
THIS_IMAGE	(COARRAY [, TEAM]) or (COARRAY, DIM [, TEAM])	T	Cosubscript(s) for this image.
TINY	(X)	I	Smallest positive model number.
TRAILZ	(I)	E	Number of trailing zero bits.
TRANSFER	(SOURCE, MOLD [, SIZE])	T	Transfer physical representation.
TRANSPOSE	(MATRIX)	T	Transpose of an array of rank two.
TRIM	(STRING)	T	String without trailing blanks.
UBOUND	(ARRAY [, DIM, KIND])	I	Upper bound(s).
UCOBOUND	(COARRAY [, DIM, KIND])	I	Upper cobound(s) of a coarray.
UNPACK	(VECTOR, MASK, FIELD)	T	Vector unpacked into an array.
VERIFY	(STRING, SET [, BACK, KIND])	E	Character set non-membership search.

3 The effect of calling EXECUTE_COMMAND_LINE on any image other than image 1 in the initial team is processor dependent.

J3/18-007r1 337

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

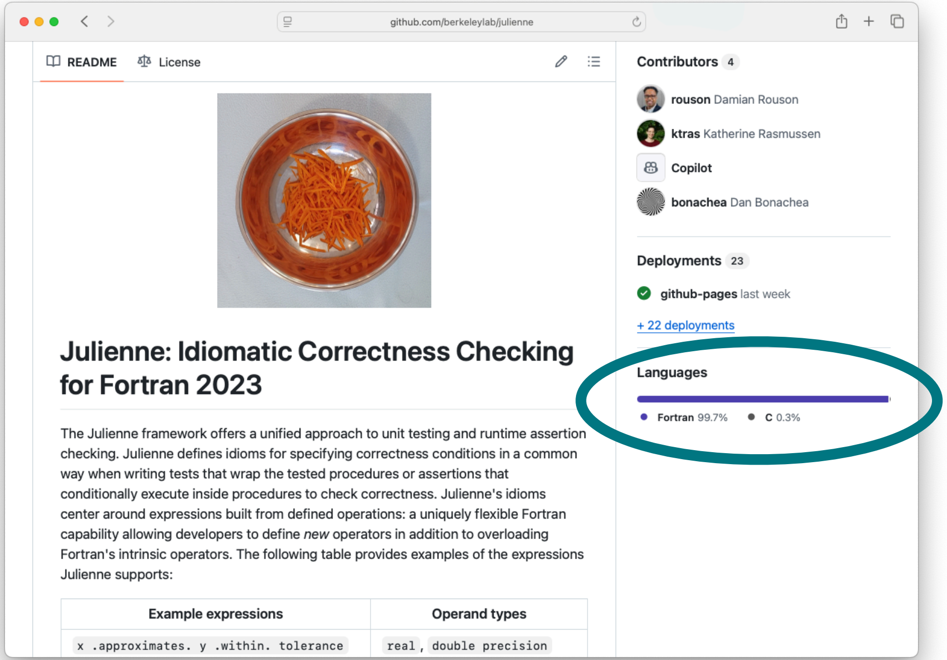
Correctness-Checking with Julienne

Unified Idioms for writing

- Unit tests
- Assertions

Support for Fortran 2023 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.



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](#)

https://go.lbl.gov/julienne

Please, No More Loops (Than Necessary)

BERKELEY LAB
Bringing Science Solutions to the World

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

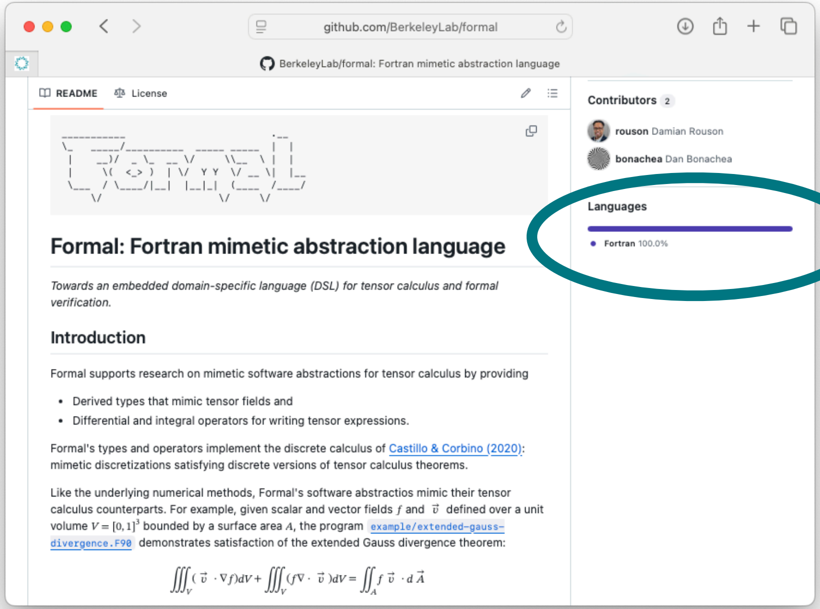
Writing PDE Solvers with Formal

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 <u>problem-specific</u> 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.

BERKELEY LAB
Bringing Science Solutions to the World

LaTeXiT-1
$$\iiint_V (\vec{v} \cdot \nabla f) dV + \iiint_V (f \nabla \cdot \vec{v}) dV + \iint_A f \vec{v} \cdot d\vec{A}$$
\iiint_V (\vec{v} \cdot \nabla f) dV + \iiint_V (f \nabla \cdot \vec{v}) dV + \iint_A f \vec{v} \cdot d\vec{A}
Auto
Align
Display
Inline
Text
Font size : 36.00 pt
Color :
LaTeX it!
formal — vim tmp.f90 — 64x10
% fpm run --example extended-gauss-divergence
f = (x**2)/2 ! <-- scalar function
v = x ! <-- vector function
.SSS. (v .dot. .grad. f) * dV = 0.33333333302059365
.SSS. (f * .div. v) * dV = 0.16666666739857122
-.SS. (f .x. (v .dot. dA)) = -0.50000000041916492
sum = 0.00000000000000000 (residual)
9,2
All

“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
```

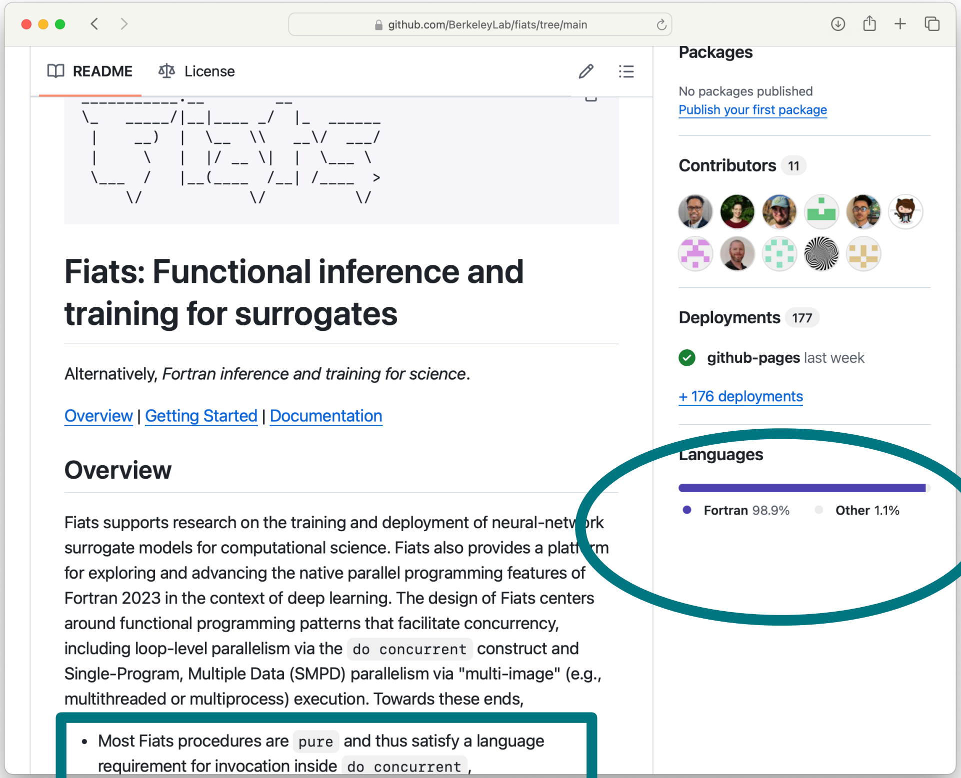
A requirement encapsulates a reusable relationship among deferred arguments.

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
```

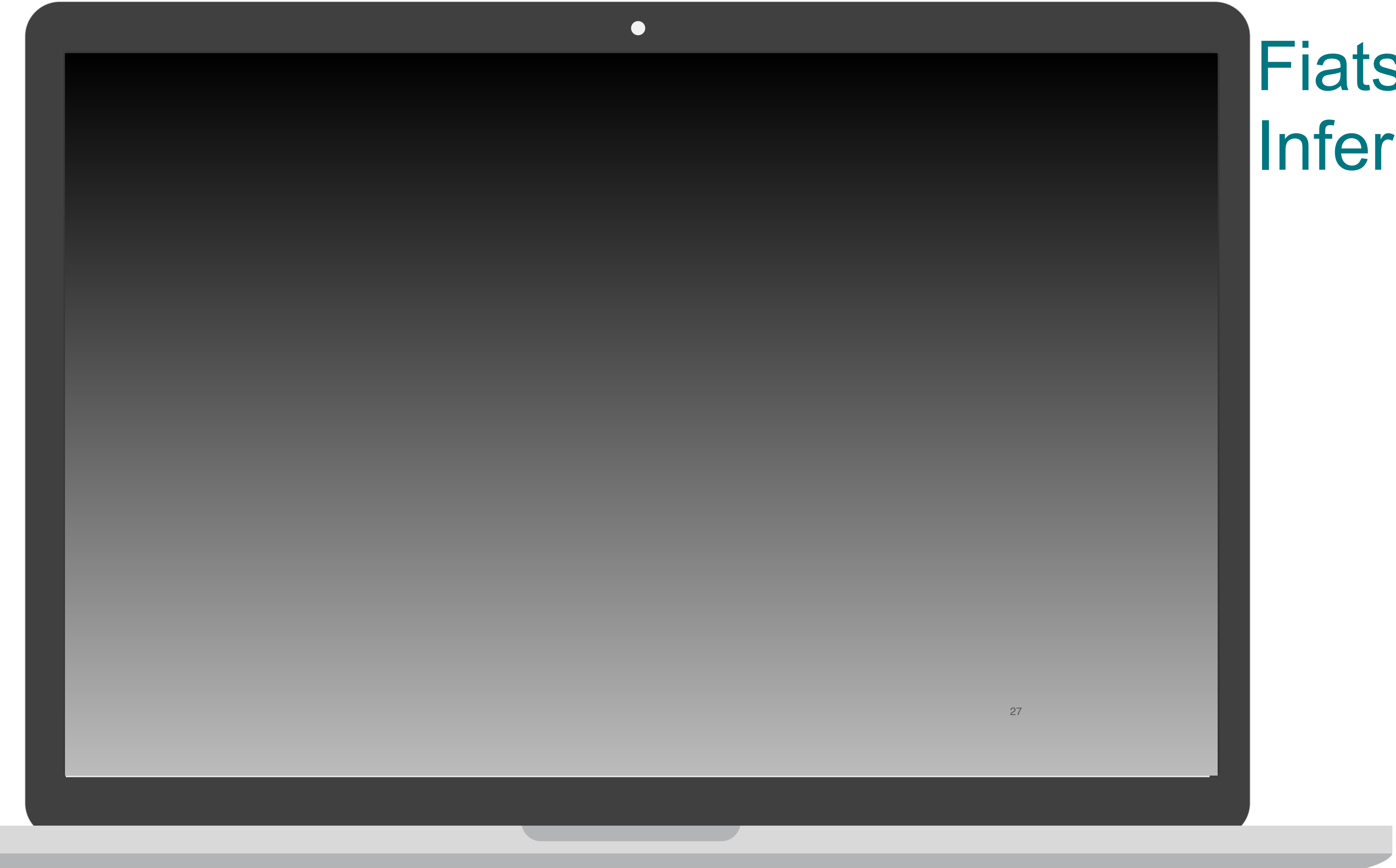
Deep Learning with Fiats



https://go.lbl.gov/fiats

BERKELEY LAB
Bringing Science Solutions to the World

Fiats: Inference



Fiats: Inference

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

```fortran
    !$omp parallel do default(none) shared(neural_network,inputs,outputs) collapse(3)
    do j=1,lon
      do k=1,lev
        do i=1,lat
          outputs(i,k,j) = neural_network%infer(inputs(i,k,j))
        end do
      end do
    end do
```

Fiats: Inference

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

```fortran
    !$omp parallel do default(none) shared(neural_network,inputs,outputs) collapse(3)
    do j=1,lon
      do k=1,lev
        do i=1,lat
          outputs(i,k,j) = neural_network%infer(inputs(i,k,j))
        end do
      end do
    end do
```

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

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

Fiats: Inference

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

```fortran
    !$omp parallel do default(none) shared(neural_network,inputs,outputs) collapse(3)
    do j=1,lon
      do k=1,lev
        do i=1,lat
          outputs(i,k,j) = neural_network%infer(inputs(i,k,j))
        end do
      end do
    end do
```

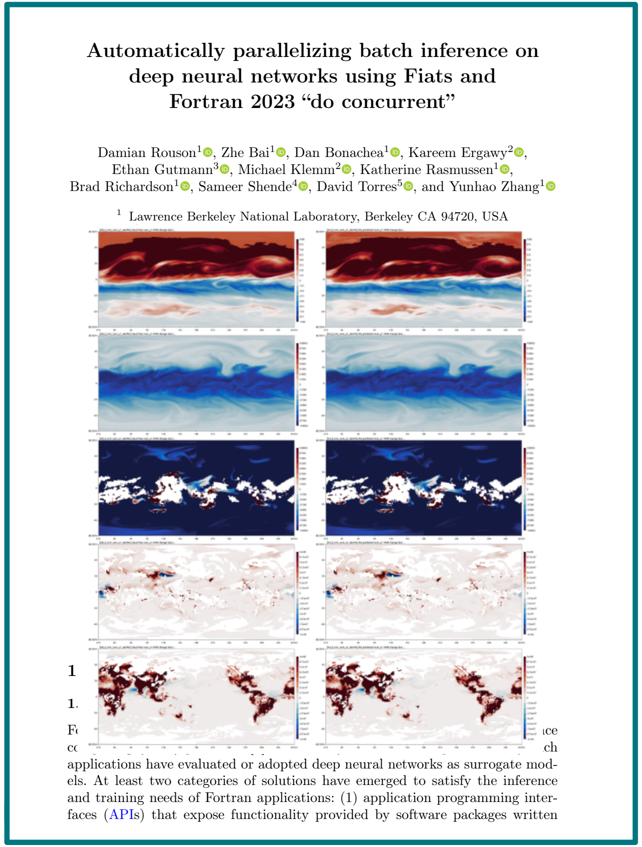
fiats — rouson@login38:~/fiats — login38 — vim example/concurrent-inferences.f90 — 103×5

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

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

```fortran
    !$omp workshare
    outputs = neural_network%infer(inputs)
    !$omp end workshare
```

Automatic Parallelization on Perlmutter CPU



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

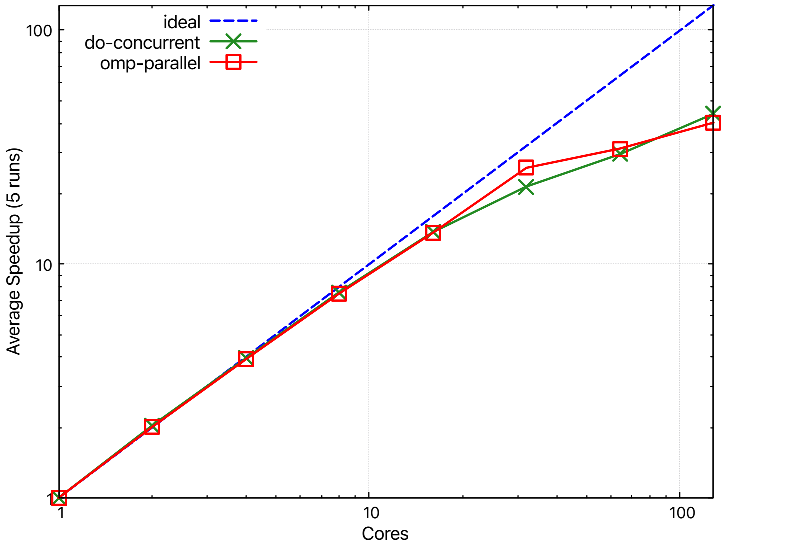
Damian Rouson[1], Zhe Bai[1], Dan Bonachea[1], Kareem Ergawy[2], Ethan Gutmann[3], Michael Klemm[2], Katherine Rasmussen[1], Brad Richardson[1], Sameer Shende[4], David Torres[5], and Yunhao Zhang[1]

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

1

1.

Fo ... ice
co ... ch
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.

BERKELEY LAB
Bringing Science Solutions to the World

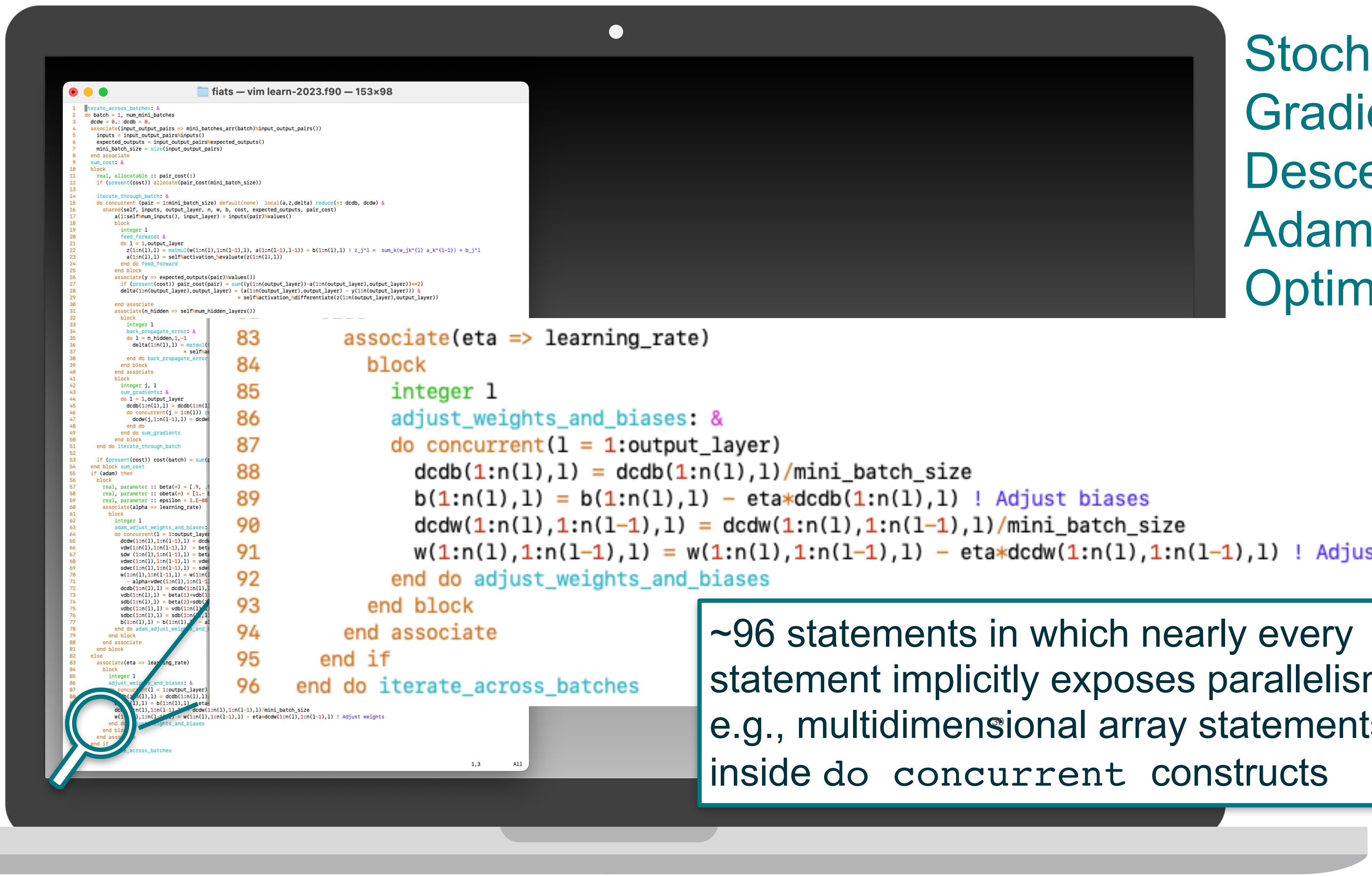
Fiats: Training

fiats — vim neural_network_s.F90 — 101×36

```
#if F2023_LOCALITY
              iterate_through_batch: &
              do concurrent (pair = 1:mini_batch_size) local(a,z,delta) reduce(+: dcdb, dcdw)

#elif F2018_LOCALITY

              reduce_gradients: &
              block
                real reduce_dcdb(size(dcdb,1),size(dcdb,2),mini_batch_size)
                real reduce_dcdw(size(dcdw,1),size(dcdw,2),size(dcdw,3),mini_batch_size)
                reduce_dcdb = 0.
                reduce_dcdw = 0.

                iterate_through_batch: &
                do concurrent (pair = 1:mini_batch_size) local(a,z,delta)

#else

              reduce_gradients: &
              block
                real reduce_dcdb(size(dcdb,1),size(dcdb,2),mini_batch_size)
                real reduce_dcdw(size(dcdw,1),size(dcdw,2),size(dcdw,3),mini_batch_size)
                reduce_dcdb = 0.
                reduce_dcdw = 0.

                iterate_through_batch: &
                do concurrent (pair = 1:mini_batch_size)

                  iteration: &
                  block

                    real a(maxval(self%nodes_), input_layer:output_layer) ! Activations
                    real z(size(b,1),size(b,2)), delta(size(b,1),size(b,2))
#endif
```

910,0-1 89%

Stochastic Gradient Descent + Adam Optimizer




```fortran
  associate(eta => learning_rate)
    block
      integer l
      adjust_weights_and_biases: &
      do concurrent(l = 1:output_layer)
        dcdb(1:n(l),l) = dcdb(1:n(l),l)/mini_batch_size
        b(1:n(l),l) = b(1:n(l),l) - eta*dcdb(1:n(l),l) ! Adjust biases
        dcdw(1:n(l),1:n(l-1),l) = dcdw(1:n(l),1:n(l-1),l)/mini_batch_size
        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
      end do adjust_weights_and_biases
    end block
  end associate
end if
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

fiats — vim neural_network_m.f90 — 90×14

```fortran
type neural_network_t(k)
  !! Encapsulate the information needed to perform inference
  integer, kind :: k = default_real
  type(tensor_map_t(k)), private :: input_map_, output_map_
  type(metadata_t), private :: metadata_
  real(k), allocatable, private :: weights_(:,:,:), biases_(:,:)
  integer, allocatable, private :: nodes_(:)
  type(activation_t), private :: activation_
contains
  generic :: infer =>                     default_real_infer, double_precision_infer
  procedure, private, non_overridable :: default_real_infer, double_precision_infer
  generic :: learn =>                     default_real_learn
  procedure, private, non_overridable :: default_real_learn
                                                  33,1           6%
```

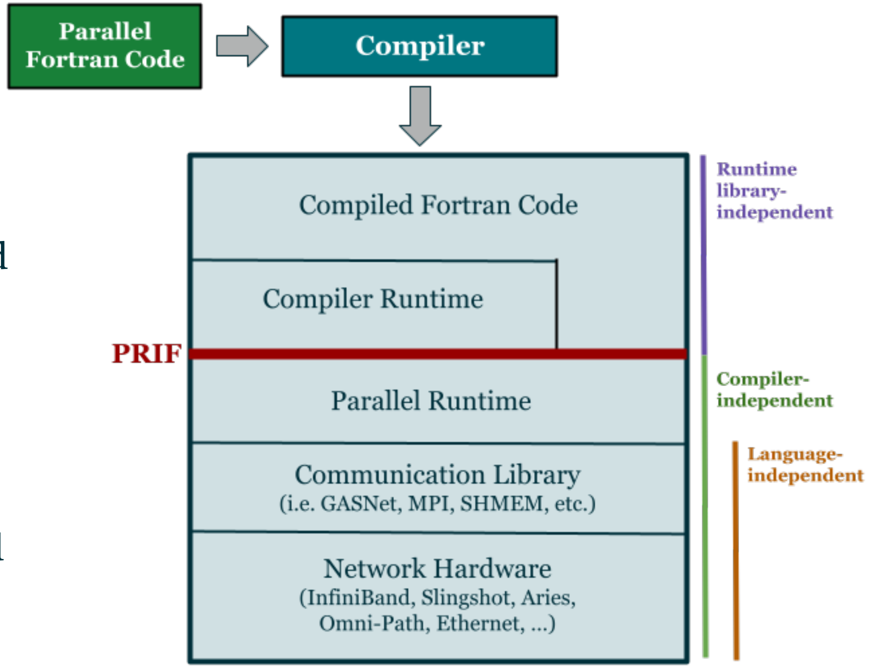
Kind type parameter allows us to set an object’s precision in its declaration *without* recompiling.

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

https://doi.org/10.25344/S4VG6T

Parallel Runtime Interface for Fortran (PRIF)

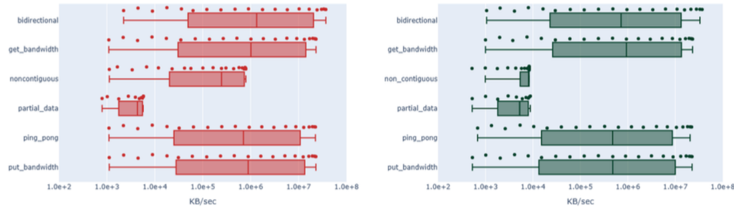
- Compiler- and runtime-agnostic interface to support multi-image parallel Fortran features
- A runtime interface written *in Fortran:* prif module
- Tight correspondence between PRIF procedures and Fortran's multi-image parallel features, e.g.,
    - num_images ⟶ prif_num_images
    - real x(N)[*] ⟶ 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

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

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.).

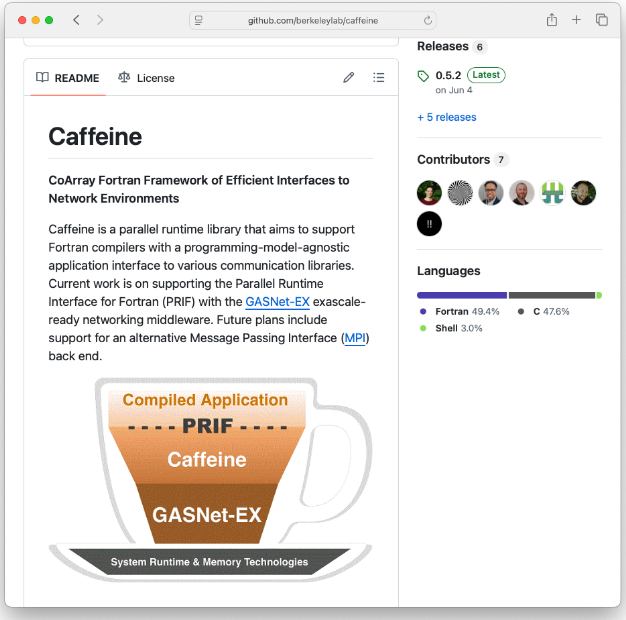
 doi: 10.25344/S4G883 1

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

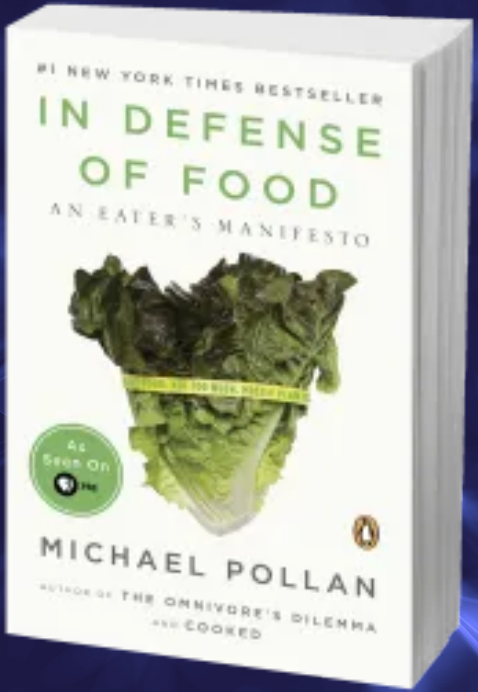
04

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!

https://go.lbl.gov/fortran