OAK RIDGE National Laboratory

An Overview of The Fortran Standard: Fortran 2023 and Beyond

Reuben D. Budiardja
Acting Group Leader,
Advanced Computing for Nuclear, Particles, and Astrophysics
Oak Ridge Leadership Computing Facility
INCITS/Fortran Technical Committee Chair

ORNL is managed by UT-Battelle LLC for the US Department of Energy

U.S. DEPARTMENT OF ENERGY

Abstract

While it has been around for many decades, Fortran is still the preeminent language suitable for scientific and numerical computation, making up a large share of applications that are running in supercomputing facilities such as OLCF. The latest Fortran standard, informally known as Fortran 2023, was just recently released. In this talk, I will give an overview of the Fortran standardization process, followed by a high-level summary of the new features in Fortran 2023. I will also discuss future improvements being considered to make Fortran an even more productive language in the era of heterogeneous HPC.

Fortran Standardization Efforts

https://j3-fortran.org/

- Standards are published by International Organization for Standardization (ISO); ISO JTC1/SC22/**WG5** is the ISO working group for Fortran
  - consists of multiple "national bodies" (NB)
- INCITS is the U.S. standard NB, serving as Technical Advisory Group to the JTC1
  - INCITS/Fortran Technical Committee is responsible for the technical development of the standards https://www.incits.org/committees/pl22.3
  - Formerly known as ANSI X3J3
- ***In practice***, WG5 provides general directions & advice, while INCITS/Fortran does the technical work
  - Members outside the U.S. participate in INCITS/Fortran

Fortran Standardization Efforts

- Performance and numerical centric
  - work hard such that feature specifications does not hamper optimization
- Continues evolution to accommodate hardware development in HPC
  - while trying to avoid chasing passing fads
- Backward compatibility
  - previous standards is proper subset of current standards

Fortran Standards Evolution

… and their compelling features in the exascale world

- FORTRAN 66
  - The first standardized version by American Standard Association (now ANSI) known as "USA Standard FORTRAN."
  - main program, subroutine, function, "intrinsic" data types

- FORTRAN 77
  - Added significant features to address shortcomings of FORTRAN66
  - Block if-statement
  - do-loop extensions
  - **implicit** statement

Fortran Standards Evolution (2)

… and their compelling features in the exascale world

- Fortran 90

my first introduction to the language

  - Notable name change from FORTRAN to Fortran
  - The first version with an international (ISO) standard; one document for both ISO and ANSI standard
  - A major revision to the standards, including new features such as:
    - free-form source
    - modules
    - generic procedures & operator overloading
    - user-defined derived (structure) data types
    - compile-time checking interfaces
    - portable, user-specified numerical precision
    - array operations with array syntax (sections or whole array)

Facilities for encapsulation, composition, abstraction, & polymorphism → managing complexities for large programs.

Fortran Standards Evolution (3)

… and their compelling features in the exascale world

- Fortran 95
  - Minor revision: mostly clarifications and correcting defects
  - `forall` statement and construct
  - `pure` and `elemental` procedures
  - pointer initialization and structure default initialization

Fortran Standards Evolution (4)

… and their compelling features in the exascale world

- Fortran 2003
  - Another major revision, including new features such as:
    - Object-oriented programming support: type extension, accessibility control, dynamic type allocation, inheritance, type-bound procedures, polymorphism
    - Enhancements to derived type
    - Procedure pointers
    - C Interoperability
  - By now all major compilers have implemented most if not *all* of Fortran 2003 standards

Fortran Standards Evolution (5)

… and their compelling features in the exascale world

- Fortran 2008
  - Relatively minor revision to the standard
    - as decided by the committee to allow time for vendors to implement Fortran 2003 (and users to learn to use it)
  - New features
    - Coarrays (parallel programming for distributed & shared-memory architecture … *more on this later*)
    - Submodules - modularization of large modules, another layer of encapsulation
    - Performance enhancement: `do concurrent`*, `contiguous`*
      (*a potential path for GPU programming directly from the base language more later*)
    - Enhancements to data objects, I/O, and execution control

Fortran Standards Evolution (6)

… and their compelling features in the exascale world

- Fortran 2018
  - Another minor revision with a few major enhancements:
    - Further interoperability with C
    - Enhancement to parallel features (coarrays)
    - locality clause to `do concurrent`

Fortran - C Interoperability (Fortran 2003, 2018)

- Also known as "`iso_c_binding`" (for the Fortran module that provides entities related to this feature).
- A standardized means to reference entities from/by C:
  - calling C function from Fortran; calling Fortran subroutine / function from C
  - defines what — and the conditions under which — entities are **interoperable**
  - manipulations of Fortran data objects (e.g. allocatable arrays, pointers) from C (or C-like) language (Fortran 2018)
- Probably one of the most useful feature in "heterogenous" world
  - For e.g., providing bindings for libraries such as MPI, OpenMP, and GPU libraries (HIP, HIPFort, CUDA, …)
- *OLCF works with compiler vendors to ensure strong support for this feature*

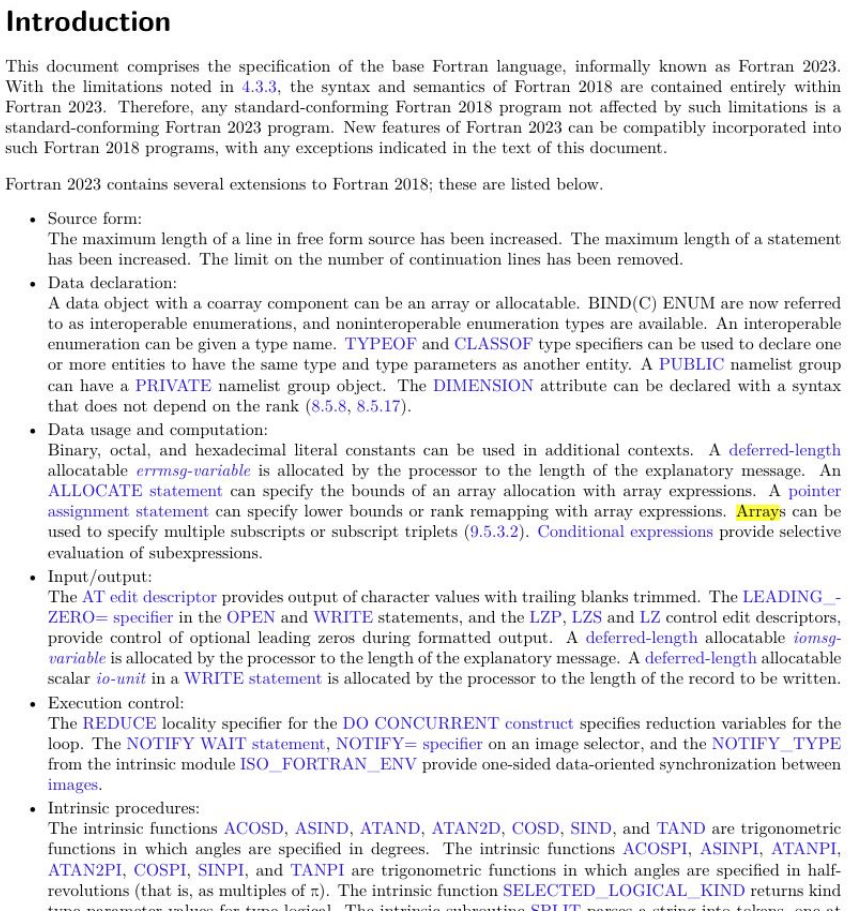
The Current Standard: Fortran 2023

… and their compelling features in the exascale world

- Fortran 2023
  - Published in October 2023
  - ISO publication: https://www.iso.org/standard/82170.html
  - Committee's "Interpretation Document" https://j3-fortran.org/doc/year/24/24-007.pdf

Fortran 2023 New Features

- The Introduction section of the standard provides the comprehensive list (with forward references)
- ~20 new features were added to the language + smallish features + "fixes" & clarification (from corrigenda, interpretation request, etc.)
  - New features informally named "CC XX [titles / short desc]"
- Only going to highlight some in this talk
- Excellent summary by John Reid at WG5:
  - https://wg5-fortran.org/N2201-N2250/N2212.pdf



Introduction

This document comprises the specification of the base Fortran language, informally known as Fortran 2023. With the limitations noted in 4.3.3, the syntax and semantics of Fortran 2018 are contained entirely within Fortran 2023. Therefore, any standard-conforming Fortran 2018 program not affected by such limitations is a standard-conforming Fortran 2023 program. New features of Fortran 2023 can be compatibly incorporated into such Fortran 2018 programs, with any exceptions indicated in the text of this document.

Fortran 2023 contains several extensions to Fortran 2018; these are listed below.

- Source form:
  The maximum length of a line in free form source has been increased. The maximum length of a statement has been increased. The limit on the number of continuation lines has been removed.
- Data declaration:
  A data object with a coarray component can be an array or allocatable. BIND(C) ENUM are now referred to as interoperable enumerations, and noninteroperable enumeration types are available. An interoperable enumeration can be given a type name. TYPEOF and CLASSOF type specifiers can be used to declare one or more entities to have the same type and type parameters as another entity. A PUBLIC namelist group can have a PRIVATE namelist group object. The DIMENSION attribute can be declared with a syntax that does not depend on the rank (8.5.8, 8.5.17).
- Data usage and computation:
  Binary, octal, and hexadecimal literal constants can be used in additional contexts. A deferred-length allocatable *errmsg-variable* is allocated by the processor to the length of the explanatory message. An ALLOCATE statement can specify the bounds of an array allocation with array expressions. A pointer assignment statement can specify lower bounds or rank remapping with array expressions. Arrays can be used to specify multiple subscripts or subscript triplets (9.5.3.2). Conditional expressions provide selective evaluation of subexpressions.
- Input/output:
  The AT edit descriptor provides output of character values with trailing blanks trimmed. The LEADING_ZERO= specifier in the OPEN and WRITE statements, and the LZP, LZS and LZ control edit descriptors, provide control of optional leading zeros during formatted output. A deferred-length allocatable *iomsg-variable* is allocated by the processor to the length of the explanatory message. A deferred-length allocatable scalar *io-unit* in a WRITE statement is allocated by the processor to the length of the record to be written.
- Execution control:
  The REDUCE locality specifier for the DO CONCURRENT construct specifies reduction variables for the loop. The NOTIFY WAIT statement, NOTIFY= specifier on an image selector, and the NOTIFY_TYPE from the intrinsic module ISO_FORTRAN_ENV provide one-sided data-oriented synchronization between images.
- Intrinsic procedures:
  The intrinsic functions ACOSD, ASIND, ATAND, ATAN2D, COSD, SIND, and TAND are trigonometric functions in which angles are specified in degrees. The intrinsic functions ACOSPI, ASINPI, ATANPI, ATAN2PI, COSPI, SINPI, and TANPI are trigonometric functions in which angles are specified in half-revolutions (that is, as multiples of π). The intrinsic function SELECTED_LOGICAL_KIND returns kind type parameter values for type logical. The intrinsic subroutine SPLIT parses a string into tokens, one at

Conditional Expressions

… provide selective evaluation of subexpressions.

General form:

```
( condition ? expression [: condition ? expression ]... : expression )
```

- Each *expression* shall have the same declared type, kind, and rank.
- Each *condition* is evaluated in succession until either
  - one with the value true is found, in which case the expression following the condition is taken
  - all found to be false, in which case the value of the final expression is taken

Example:

```
if ( a > 1.0 ) then
  value = a
else
  value = 0.0
end if
```

```
value = ( a > 1.0 ? a : 0.0 )
```

Conditional Arguments

… provide actual argument selection in a procedure reference.

General form:

```
( condition ? consequent [: condition ? expression ]... : consequent )
```

- Each *consequent* is an expression, a variable, or `.nil.` to specify absence
- Each *condition* is evaluated in succession:
  - one with the value true is found, in which case the consequent following the condition is taken
  - all found to be false, in which case the value of the final consequent is taken
- Nesting is not allowed, i.e. *`consequent`* cannot be a conditional argument, but can be a conditional expression

Conditional Arguments

Example:

```
call MySub ( ( x > 0 ? x  :  y > 0 ? y  :  z, &
             ( edge > 0 ? edge  :  mode == 3 ? 1.0  : .nil. ) &
             some, other, arguments )
```

where the interface of MySub is:

```
subroutine MySub ( x, bnd )
  real, intent ( inout ) :: x
  real, intent ( in ), optional :: bnd
```

Fortran Parallelism

```
real :: a, b, x(n)
a = 0
b = -huge (b)
do concurrent (1 = 1, n) reduce (+:a) reduce(max:b)
  a = a + x(i)
  b = max(b, x(i)
end do
```

- `do concurrent`:
  - first introduced in Fortran 2008
  - tells the compiler there are no data dependencies between iterations
  - compiler may optimize with, e.g. vectorization, unrolling, multi-threadings
  - There is a set of restrictions that must be satisfied for do-concurrent
- Meant to standardized available directives recognized by compilers (sometime with different exact meanings)
- Fortran 2018 adds locality clause: (`local, local_init, shared, default(none)`)
- Fortran 2023 adds reduction specifier for intrinsic operator:
  `+`, `*`, `.and.`, `.or.`, `.eqv.`, `max`, `min`, `iand`, `ieor`, `ior`
- *Some compilers allow offloading of `do concurrent` to GPU*

Using Arrays to Specify Array Subscripts (1)

Recap: an array section `A` in Fortran is addressed with:

```
A ( [lbound] : [ubound] : [stride] [, subscript-triplet]... )
```

Example:
Suppose an array is declared as `A(5, 4, 6)`. The array-section `A ( 3:5, 2, 1:5:2 )` is a rank-2 array of shape (3, 3) the following elements from **A**:

```
A ( 3, 2, 1)        A ( 3, 2, 3)        A (3, 2, 5)
A ( 4, 2, 1)        A ( 4, 2, 3)        A (4, 2, 5)
A ( 5, 2, 1)        A ( 5, 2, 3)        A (5, 2, 5)
```

Using Arrays to Specify Array Subscripts (2)

multiple-subscript can be used to specify a sequence of subscripts

Examples of references to parts of arrays using one-dimensional arrays to specify sequences of subscripts or sequences of subscript triplets, assuming V1, V2, and V3 are rank-one arrays, are:

```
A(@[3,5])           ! Array element, equivalent to A(3, 5)
A(6, @[3,5], 1)     ! Array element, equivalent to A(6, 3, 5, 1)
A(@[1,2]:[3,4])     ! Array section, equivalent to A(1:3, 2:4)
A(@:[4,6]:2, :, 1) ! Array section with stride, equivalent to A(:4:2, :6:2, :, 1)
A(@V1, :, @V2)      ! Rank-one array section, the rank of A being
                    ! SIZE (V1) + 1 + SIZE (V2).
B(@V1, :, @V2:)     ! Rank 1 + SIZE (V2) array section, the rank of B being
                    ! SIZE (V1) + 1 + SIZE (V2).
C(@V1, :, @::V3)    ! Rank 1 + SIZE (V3) array section, the rank of C being
                    ! SIZE (V1) + 1 + SIZE (V3).
```

provides a way to write code to access array in a rank-agnostic manner

Arrays to Specify Rank and Bound

```
integer, dimension(3) :: lb_array, ub_array

real, dimension(lb_array-1 : ub_array+1) :: grid !-- rank-3 array

allocate ( x ( : ub_array), y (lb_array: ub_array) )
```

Integer Constant to specify Rank

```
integer, dimension(10, 10, 10) :: x0

logical, rank(3), allocatable :: x1

real, rank(rank(x0)), allocatable :: x2
```

(Very) Brief Introduction / History to Coarrays

- Started with a simple idea: we can calculate the address of arrays in remote processors from the local array
- First implemented as simple **put** and **get** operations
- Add subscript (with [ ]) to array to indicate processor grid. Operation with a *codimension* indicate to programmer that it (potentially) involves remote access.
- Standardized in Fortran 2008

```
if (p==p1) then
   call MPI_send (array1,size(array1),MPI_real,p2,tag,comm,ierr)
else if (p==p2) then
   call MPI_recv (array2,size(array1),MPI_real,p1,tag,comm,status,ierr)
end if
```

MPI Send & Recv

```
if (p==p1) array2(:)[p2] = array1(:)
```

Fortran coarrays

J. Reid, B. Long, J. Steidel, *"History of Coarrays and SPMD Parallelism in Fortran"*

(Very) Brief Introduction / History to Coarrays (2)

- Coarrays provides parallel programming capability for Fortran on distributed- and shared-memory systems.
- Program is replicated, each replication is called an *image*.
- An additional set of subscripts provide access to data on another image. Additional statements provide image control.
- Compiler can optimize both execution and communication between images images
- Coarrays is supported on Frontier with HPE/Cray compiler
- Recent Coarray tutorial at OLCF https://www.olcf.ornl.gov/calendar/introduction-to-high-performance-parallel-distributed-computing-using-chapel-upc-and-coarray-fortran/

US 12 Array of Coarray

- Fortran 2018:
  C825 An entity whose type has a coarray ultimate component shall be a nonpointer nonallocatable scalar, shall not be a coarray, and shall not be a function result.

- Use case to relax the constraint https://j3-fortran.org/doc/year/18/18-280r1.txt :

  Boundary-data communication exchange … may be encapsulated into derived type, e.g.

```
type vector
    real, allocatable :: component(:),component_buffer(:)[:]
end type
type(vector) :: bundle, field
```

  Unfortunately, Fortran 2018 rules fix the number of such data objects in the program. Fixing the number of data objects at compile time is undesirable … because it prevents those same objects from being reused as components of other higher-level objects in a sufficiently flexible way … would need to be recompiled to allow for changing the partitioning of the problem into subdomains.

- Fortran 2023 allows object with coarray component to be array or allocatable:

```
type(vector), dimension ( :, :, :), allocatable :: bundle, field
```

  C825 An entity whose type has a coarray potential subobject component shall not be a pointer, shall not be a coarray, and shall not be a function result.

`simple` Procedures

- Fortran 95 introduced **`pure`** procedure: procedure that does not have side effect. e.g. it changes the variables through its argument (function return, or intent(inout/out) args for subroutine.
  - allowing it to be used in parallel construct / concurrency

- Fortran 2023 introduces **`simple`** procedure
  - must satisfy all requirements of pure procedure
  - plus additional requirements to ensure an entirely local calculation
  - allow compiler to better optimized for threads / concurrency
  - all intrinsic functions are simple

- Example

```
real simple elemental MyFunction ( a1 )
  …
end function MyFunction
```

Fortran 2023 Features Not Covered

- US 01 & 92: length statement
- US 14: Automatic allocation of length of character
- US 16: `typeof` and `classof`
- US 23: binary, octal, hexadecimal constants
- US 03: `split` and `tokenize` (token extractions from string)
- US 04, 05: Trig functions
- US 07, 08: Additional named constants for `kind`s
- UK 01: c_f_pointer can specify lower bound
- US 09: C - Fortran string conversion
- US 10, 11: edit descriptor enhancements
- US 21: Enumeration enhancements
- Other miscellaneous enhancements and clarifications

Fortran 202Y? (with 'Y' TBD)

- The committee is already working on the next standard, informally called F202Y
- An initial list of features was approved by NB at 2023 WG5 ISO Fortran Meeting, including
  - Generic programming with Template
  - Generic subprograms
  - Standardize Fortran preprocessor
  - Asynchronous tasking
  - Improved rank-independent functionality
  - etc. see https://wg5-fortran.org/N2201-N2250/N2222.txt for current list
- The next WG5 Fortran meeting (June 2024) will approve additional features

Generic Programming with Templates (F202Y **Preview**)

Disclaimer: details are subject to change

A motivating _AXPY example:

```
subroutine axpy(a, x, y)
  real, intent(in) :: a
  real, intent(in) :: x(:)
  real, intent(inout) :: y(:)
end subroutine axpy
```

Generic Programming with Templates (F202Y **Preview**)

Disclaimer: details are subject to change

```
requirement bin_op(T, op)
  type, deferred :: T
  elemental function op(a, b)
    type(T), intent(in) :: a, b
    type(T) :: op
  end function
end requirement
```

```
template axpy_tmpl(T, plus, times)
  private
  public :: axpy
  requires bin_op(T, plus)
  requires bin_op(T, times)
  interface axpy
    procedure axpy_
  end interface
contains
  subroutine axpy_(a, x, y)
    type(T), intent(in) :: a
    type(T), intent(in) :: x(:)
    type(T), intent(inout) :: y(:)

    y = plus(times(a,  x), y)
  end subroutine
end template
```

```
integer, parameter :: sp = kind(1.0), dp = kind(1.d0)
instantiate axpy_tmpl(real(sp), operator(+), operator(*))
instantiate axpy_tmpl(real(dp), operator(+), operator(*))
instantiate axpy_tmpl(integer, operator(+), operator(*))
real(sp) :: a, x(10), y(10)
real(dp) :: da, dx(10), dy(10)
integer :: ia, ix(10), iy(10)
…
call axpy(a, x, y)
call axpy(da, dx, dy)
call axpy(ia, ix, iy)
```

credit: Tom Clune (NASA), Brad Richardson (NERSC), & INCITS/Fortran Generic Subgroup

Fortran in Heterogeneous Computing World

Fortran remains an excellent language platform:

- first class array handling
- relatively simple language
  - can do sophisticated execution control and data with OOP, but straightforward computational kernels for better optimization
- easy for compilers to optimize
- run in many architectures (x86-64, Power, ARM, …)
- backward compatibility
  - previous standards are proper subset of current standards

Everything in Fortran

- Current practice
  - MPI to manage multiple processes and inter-process communications (e.g. one process per node, per NUMA-domain, …)
  - OpenMP for shared-memory, multithreading execution (threading on multi- & many-cores)
  - Heterogeneous programming models for offloading computation to GPUs (with one or multiple GPU per process)

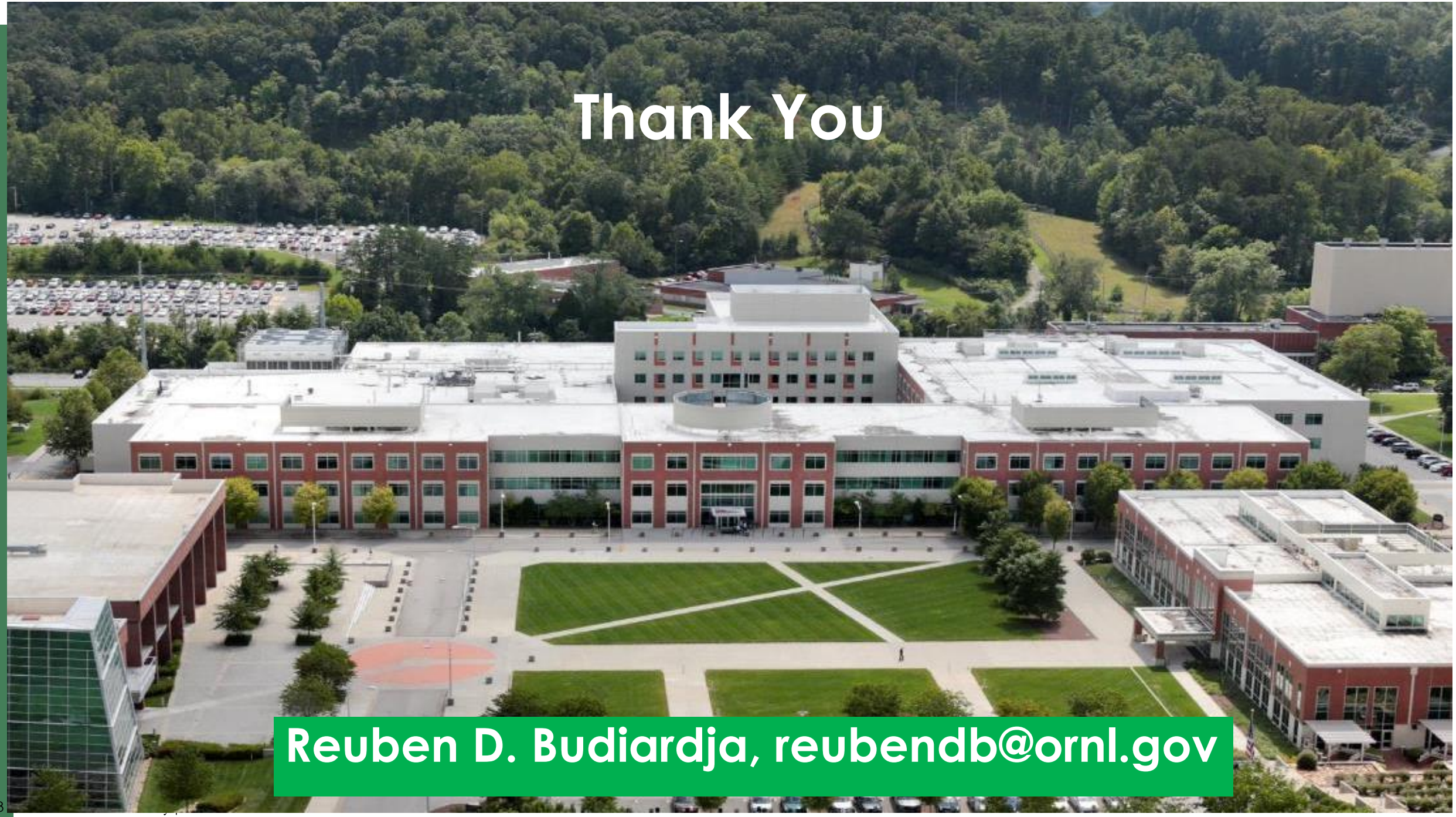
- Potential (near) future for Fortran applications?
  - coarrays to manage multiple "images" (run as processes)
  - do concurrent for multithreading and offloading to GPUs
  - Some technical challenges still to overcome
  - Advantages: everything is done in the language, potentially simpler for developer and better optimization opportunities.

Community Building

- SC23 BoF, and likely SC24 BoF
- https://fortran-lang.discourse.group/
- https://github.com/j3-fortran/
- ORNL / OLCF actively participates in many standardization efforts:
  - Fortran, C++
  - OpenMP
  - MPI
- OLCF works with vendor partners to prioritize feature implementations of new standards based on user / program needs

Concluding Remarks

- Fortran remains important to industries and agencies
  - it will continue to be supported at computing facilities
- Fortran standards continues to evolve as a modern language.
  - what important features you would like to see?
- Consider joining the standards committee
  - or reach out to ORNL / OLCF representatives
- How do we make sure we continue to have strong compilers?
  - use features that can improve / benefit your code
  - include standard tracking as part of procurement
- Need robust supports in tools, programing models, and ecosystems
- Strong workforce development is needed


Thank You
Reuben D. Budiardja, reubendb@ornl.gov