"Modern Fortran"

A contradiction in itself
or
A future-proof language?
ISO-IEC 1539-1: Horses vs. Dinosaurs

• this Dinosaur is in Use
  (... on both ends of the keyboard)

• Communities:
  Fortran is used for
  • data-intensive numerical calculations
    (need for optimization!)
  • nuclear weapons simulations
  • weather and climate modeling
  • quantum chemistry, biology
  • engineering (CFD, structural mechanics)
  • embedded components (R, python, ...)

• Learnability
  • good usefulness/effort ratio

• this Dinosaur has adapted

• Language evolution driven by
  ... 2 ... committees

  • feature definitions by
    ISO Fortran Language Committee
    („WG5“) [https://wg5-fortran.org/]

  • technical implementation by
    US National Body INCITS PL22.3
    („J3“) [https://j3-fortran.org/]

October 2022 | R. Bader (LRZ)
The Technical Literature does its best to keep up ...
What is "modern" about Fortran?

• Focus of FORTRAN 77
  • Portability
  • Performance
    (... for numerical algorithms)

• Focus today
  • Portability
  • Performance
    (... on scalable HPC systems)
  • Interoperability
  • Software Design and Engineering
  • Resiliency
    (... optional)

apart from cosmetic innovations ...
Fortran Standard
New Concepts in (and outside) Standard Generations

Fortran 90/95
- Module
  static Inheritance encapsulation
- Dynamic data
  Pointers allocatable Objects
- Type concept
  POD components dynamic components
- Array processing
- Parallelism ?
  FORALL OpenMP
  MPI (on crutches)

Fortran 2003
- Object orientation
  single dynamic inheritance type- and object-bound procedures
  abstract types and interfaces
- IEEE-754
- C-Interoperability
  C semantics in Fortran
- I/O extensions
  Asynchronous, Streams, UDDTIO
- Parallelism ??
  parameterized Data types (SoA vs AoS)
  OpenMP tasks

Fortran 2008
- Submodules
  Dependency inversion now for real !
- Parallelism !
  SPMD-PGAS: Coarrays
  DO CONCURRENT
  OpenMP simd
  OpenMP target
  MPI
  (still on crutches)

Fortran 2018
- C-Interoperability
  Fortran semantics in C
  Asynchronous extension
- Parallelism !!
  Coarray Teams
  collective intrinsics atomic intrinsics events
  DO CONCURRENT
  erweitert
  OpenMP
  MPI
  (nearly) without crutches!
- Resiliency
  optional

October 2022 | R. Bader (LRZ)
• important modules are used by **many** other program units → "structural" Dependency Inversion
• deep submodule hierarchies are permitted, but likely of only limited usefulness
Software Design and Engineering

Dependency Inversion (2) – Interface class

- Abstract type and abstract interface

```plaintext
TYPE, ABSTRACT :: handle
CONTAINS
  PROCEDURE(open_handle), DEFERRED :: open
END TYPE HANDLE

ABSTRACT INTERFACE
  SUBROUTINE open_handle(this)
    IMPORT :: handle
    CLASS(handle) :: this
  END SUBROUTINE
END INTERFACE

TYPE, EXTENDS(handle) :: file_handle
PRIVATE
  INTEGER :: unit
CONTAINS
  PROCEDURE :: open => open_file_handle
END TYPE file_handle
```

- Usage:

```plaintext
CLASS(handle), ALLOCATABLE :: my_handle
my_handle = ...
call my_handle%open()
```

- Observation: compile time access is only to abstract API
- "semantic" Dependency Inversion
• It would be nice if life were that easy ... but ...
What comes first – Hen or Egg?

• Creation of the polymorphic object ...

• How?
   not possible through a method
   ... since the latter needs the dynamic type of the object
   which is in turn determined by the creation process.

• Where?
   in the module that defines the abstract API
   ... in order to preserve the structural Dependency Inversion

• Interface

```plaintext
FUNCTION create_handle(htype) result(h)
  USE mod_file_handle, ONLY : file_handle
  CHARACTER(len=*) , INTENT(in) :: htype
  CLASS(handle), ALLOCATABLE :: h
  ...

  SELECT CASE(htype)
    CASE 'file_handle'
      ALLOCATE( file_handle :: h )
    ...
    CASE default
      STOP 'Unsupported extension of handle'
  END SELECT
```

• Possible implementation

```plaintext
... SELECT CASE(htype)
  CASE 'file_handle'
    ALLOCATE( file_handle :: h )
  ...
  CASE default
    STOP 'Unsupported extension of handle'
END SELECT

  USE access to definition of file_handle is needed  ➔ circular USE❗
```
Solution: combine structural and semantic Dependency Inversion

**Beware:**
- USE association overrides host association
- Use e.g., ONLY to avoid side effects
Parallelism

• Within the Standard

  ➢ Compiler-driven

    ```
    DO CONCURRENT
    ...
    END DO
    ```

    – additional clauses (locality, reductions)
    – hardware independent

  ➢ Coarrays

    – Replication of serial program image (SPMD)
    – Scalable model

• Outwith the Standard

  ➢ OpenMP - portable directives

    – Threaded execution in coherent shared memory
    – Vector units (simd)
    – Accelerators (offload)

  ➢ Message Passing (MPI)

    – Replication of serial program image
    – Under control of communication library
    – Scalable model

    – Standard conforming?
Asynchronous processing

- overlap computation with independent data transfers

Assumption:
- additional system resources are available for processing the extra activity or even multiple activities (without incurring significant overhead)
Realization – non-blocking writes in MPI

• Specifications

```
USE mpi

REAL :: buf(NDIM)
INTEGER :: count, ierr, tag, req
INTEGER :: status(MPI_STATUS_SIZE)
```

• MPI rank 0

```
CALL mpi_isend(buf, count, MPI_REAL, &
   dest=1, tag, &
   MPI_COMM_WORLD, req, ierr)

: Code that does not modify buf

CALL mpi_wait(req, status, ierr)
```

• MPI rank 1

```
CALL mpi_recv(buf, count, MPI_REAL, &
   source=0, tag, &
   MPI_COMM_WORLD, status, ierr)

: Do anything with buf
```

• Beware:
  ➢ Implementation is not **obliged** to overlap communication and computation!
Unfortunately, lots of bad things can happen ...

- Race conditions appear mysteriously (with a new and improved compiler and/or MPI library)
- Transmission of array section fails

- And, while we're at it,
  - none of the MPI calls with buffers are type-safe
  - reason: MPI buffers can be of any type and rank, but Fortran (up to 2008) lacks a concept for this

- So, apart from not really working, the MPI interface is not standard-conforming.
  - crutches: VOLATILE, MPI_F_SYNC_REG, directives for suppressing signature check

```fortran
CALL mpi_wait(req, status, ierr)
buf = ...
```
```fortran
CALL mpi_isend(buf(:,stride), ...)
```

(optimizer's code motion)

dummy argument is assumed size DIMENSION (*), causing copy-in
Resolution – use Fortran 2018 and the new MPI interface (> 3.1)

- Specifications

```fortran
USE mpi_f08
REAL :: buf(NDIM)
INTEGER :: count, tag
TYPE(mpi_request) :: req
TYPE(mpi_status) :: status
```

- MPI rank 0

```fortran
BLOCK
  ASYNCHRONOUS :: buf
  CALL mpi_isend(buf, count, MPI_REAL, &
    dest=1, tag, &
    MPI_COMM_WORLD, req)
  
  CALL mpi_wait(req, status, ierr)
END BLOCK
```

- buf dummy argument for `mpi_isend`

```fortran
TYPE(*), ASYNCHRONOUS, INTENT(in) :: buf(..)
```

inhibits code motion

assumed-type (really the same as `*void`)

assumed-rank

just to make sure

- Array sections:

```fortran
IF (MPI_SUBARRAYS_SUPPORTED) THEN
  CALL mpi_isend(buf(::stride), ...)
ELSE
  ...
END IF
```

Code that does not modify `buf`
The machinery behind assumed-rank ...

- Assume `mpi_isend()` has a BIND(C) interface:

```c
#include <ISO_Fortran_binding.h>
void mpi_isend(CFI_cdesc_t *buf, ...);
```

- C descriptor for Fortran object

- Actual argument is a complete array (0:1,3)

- Actual argument is an array section (0::2,1::3) of (0:2,9)

- Not meant for tampering — must use API

- All "orange" storage units are not part of the object, but are exposed by descriptor — do not touch!
Implement Fortran `mpi_isend()` in terms of C `MPI_Isend()`

```c
void mpi_isend( CFI_cdesc_t *buf, int count, MPI_Datatype datatype, int dest, int tag,
               MPI_Comm comm, MPI_Request *request, int *ierror ) {
    int ierror_local;
    MPI_Datatype disc_type;
    if ( CFI_is_contiguous( buf ) ) {
        ierror_local = MPI_Isend( buf->base_addr, count, datatype,
                                dest, tag, request, comm );
    } else {
        ... /* use descriptor information to construct disc_type
             from datatype (e.g. via MPI_Type_create_subarray) */
        ierror_local = MPI_Isend( buf->base_addr, count, disc_type,
                                dest, tag, request, comm );
        ... /* clean up disc_type */
    }
    if (ierror != NULL) *ierror = ierror_local;
}
```

Notional - I'm not saying it is done exactly this way ...

Using this was what you needed to do in "old" Fortran anyway 😎

C API always uses contiguous buffer
PGAS Programming with coarrays (Fortran 2008)

- Asynchronous execution (SPMD)
  - **Images** 1, 2, …, num_images()
  - each image has its own data

- Coarrays
  - Declaration with `CODIMENSION` attribute marks data as communicable
  - such data are **symmetric**: same type, rank and shape on all images
  - image-local accesses: as for non-communicable data

- Data transfer and synchronisation
  - **Segment ordering:**
    - \( q_1 \) before \( p_2 \), \( p_1 \) before \( q_2 \)
    - \( q_j \) vs. \( p_j \) unordered

- Data transfer and synchronisation
  - Segment ordering:
    - \( q_1 \) before \( p_2 \), \( p_1 \) before \( q_2 \)
    - \( q_j \) vs. \( p_j \) unordered
Coarray teams (Fortran 2018)

- Compose parallel programs
- Data exchange across team boundaries

- is possible only for coarrays \((\text{bd}[*])\) that already exist prior to \text{change team} execution
- the current team context must be preserved (no deallocation of \text{df}\[**\])!
- new syntax is therefore needed for coindexing, e.g.

\[
... = \text{bd}(::)[4, \text{TEAM}=\text{parent\_team}]
\]

- cross-team synchronisation done with \text{sync team} (not shown here)

\textbf{fluid und \textit{structure} are sibling teams}
Usage scenarios for teams

• MPMD-style parallelism

• hybrid parallelism
  • team images within shared memory as potential optimization
  • recursive team decomposition (e.g. for “divide and conquer“ algorithms)

• Resiliency

• optional feature: continue execution in the face of partial hardware failure (non-impacted images)
• Defining a spare image set in a team decomposition allow the programmer to implement resilient applications
• Requires significant programming effort to do properly, including reworking of parallel library components.
Combining parallel programming models?

• Yes, but ...
  • a multitude of design options (you need to decide on one early on)
  • high implementation dependency (portability suffers)
• MPI
  • MPI standard does not deal with coarrays and potential interactions
• OpenMP
  • currently no coarray support
  • separating to procedures should work
  • NAG does not permit this combination, however permits MPI „outside“ and coarrays „inside“

• MPI-coarrays one-to-one

• Hierarchical setups
## Implementations

### As of 2022

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Platform</th>
<th>Implementation</th>
<th>distributed memory?</th>
<th>Resiliency?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Fortran („classic“)</td>
<td>x86</td>
<td>MPI based</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>GCC gfortran</td>
<td>multiple</td>
<td>MPI based (Opencoarrays)</td>
<td>yes</td>
<td>partial</td>
</tr>
<tr>
<td>GCC gfortran</td>
<td>multiple</td>
<td>shared memory, not all features</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>NAG Fortran</td>
<td>x86</td>
<td>shared memory (exclusive alternative to OpenMP)</td>
<td>no</td>
<td>yes (semantics)</td>
</tr>
<tr>
<td>Cray Fortran</td>
<td>Cray / HPE</td>
<td>low-level communication library</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

Especially MPI based implementations: leave headroom for scalability and performance
Parameterized data types
a yet-hidden treasure of object based programming

• „Array of structures“

```fortran
type :: body
  character(len=4) :: units
  real :: mass
  real :: pos(3), vel(3)
end type body
```

```fortran
type(body), allocatable :: traj(:)
  allocate( traj(3) )
```

• „Structure of arrays“ (Fortran 2003)

```fortran
type :: body_p( k, ntraj )
  integer, kind :: k = kind(1.0)
  integer, len :: ntraj = 1
  character(len=4) :: units
  real(kind=k) :: mass(ntraj)
  real(kind=k) :: pos(ntraj,3), vel(ntraj,3)
end type body_p
```

```fortran
type( body_p (ntraj=:) ), allocatable :: dyn_traj :
  allocate( body_p (ntraj=3) :: dyn_traj )
```
Memory Layout

- **AoS with 3 field elements**
  - Memory accesses are typically non-contiguous
    → loss of "spatial locality“, independent of field size
- **SoA with LEN parameter value 3**
  - Supports contiguous access for all components
    → vectorizable
    → accelerators: efficient offload

Actual memory layout may differ in details.
Performance comparison for momentum transfer
(Intel Skylake 2.3 GHz)
Alternative: user allocatable components?

- **Pro**
  - Vectorization works
  - Implementations are mature

- **Contra**
  - Use as coarray is rather limited (avoid remote allocation)
  - Performance impact for data transfers through coarrays (additional latency for accessing unsymmetric memory)

Example: scalar coarray of parameterized type

- symmetric memory!

```fortran
  type( body_p (ntraj=:) ), allocatable :: dyn_traj[:]
  allocate( body_p (ntraj=256) :: dyn_traj[*] )
  dyn_traj[p]%vel(:,:,)= ... sync all
```

Implementations are still immature
• Coarray extensions:
  ➢ notify/wait mechanism for data transfer
  ➢ arrays of a type with coarray components
• DO CONCURRENT:
  ➢ REDUCE clause
• Declare object based on another object’s type
  ➢ TYPEOF / CLASSOF
• SIMPLE procedures
  ➢ more restricted than PURE
  ➢ PURE intrinsics are all SIMPLE
• Additional intrinsics (esp. for strings) and ISO_FORTRAN_ENV constants
• Extensions to array syntax
  ➢ multi-subscripts, rank-agnostic DIMENSIONing
• Conditional expressions
  ➢ both RHS expressions and actual arguments in procedure calls
• Enumeration types
  ➢ both Fortran and C-compatible (more restricted), with slightly different syntax
• Additional edit descriptors
• Auto-allocation of error strings and internal records
Fortran futures (2) – Fortran 202Y

Already decided

- Templates (most likely) for generic programming
  - can be nested
  - parameters can be types, procedures ...
  - restrictions on parameters can be specified
  - no metaprogramming!
- https://github.com/j3-fortran/generics/tree/main/examples

Potential further features

- Still under consideration!
  - extensions to collective intrinsics
  - extend data access to assumed-rank
  - ... many other small features
- Current collection of requests (!) is at https://j3-fortran.org/doc/year/22/22-176r5.docx

Feature is under active development
Trying to answer the question from the cover page

• modern ... a contradiction in itself?

→ No

✓ the language has been (and will be) re-honed to keep up with scientist's needs
✓ relevant software and engineering concepts have been added
✓ use best practices in new codes to avoid (problematic) legacy features

• ... future proof?

→ Maybe

✓ performance portability is becoming more difficult to achieve
✓ insufficient support for accelerator hardware

„future“ is also a technical term:

```
future(id, device) : a = process(b, c) : ! do other stuff
wait(id)
```