

Fortran Interoperability with Python and C

Daniel Topa
daniel.topa@hii.com

Huntington Ingalls Industries
Mission Technologies

December 12, 2024

Abstract

Modern Fortran is a powerful language for high-performance computing (HPC) and numerical analysis, offering features specifically designed for scientific and engineering applications. Its support for advanced numerics, such as high-precision arithmetic and robust intrinsic functions, makes it ideal for solving complex mathematical problems. With built-in tools for vectorization and array operations, Fortran enables concise, efficient code for handling large datasets. Coarrays introduce a simple yet effective parallel programming model, allowing developers to harness distributed-memory architectures with ease. Additionally, modern Fortran integrates seamlessly with C and Python, ensuring interoperability and access to extensive ecosystems. These features, combined with decades of optimization for numerical workloads, make modern Fortran a top choice for computational scientists and engineers.

Contents

| | | |
|----------|--|----------|
| 1 | About Fortran | 2 |
| 1.1 | Fortran: A Pioneering Language | 2 |
| 1.2 | Why Fortran? | 3 |
| 2 | Calling Fortran from Python | 4 |
| 2.1 | Why PyBind11? | 4 |
| 2.2 | Workflow Overview | 4 |
| 2.3 | Example Implementation | 4 |
| 2.3.1 | Fortran Code | 4 |
| 2.3.2 | C++ Wrapper | 5 |
| 2.3.3 | Python Code | 5 |
| 2.4 | Alternatives to PyBind11 | 6 |
| 3 | Fortran Interoperability With C | 6 |
| 3.1 | Fortran and C Interoperability | 6 |
| 3.2 | Key Features of Interoperability | 6 |
| 3.2.1 | Interoperable Data Types | 6 |
| 3.2.2 | The ISO_C_BINDING Module | 6 |

| | | |
|----------|---|----------|
| 3.2.3 | Procedure Interoperability | 6 |
| 3.2.4 | Interoperability of Global Data | 7 |
| 3.2.5 | Interfacing with C Pointers | 7 |
| 3.2.6 | Interoperability of Arrays | 7 |
| 3.3 | Historical Impact | 7 |
| 3.4 | Further Enhancements with TS 29113 | 7 |
| A | Fortran's iso_c_binding | 7 |
| A.1 | Key Features of Interoperability | 8 |
| A.1.1 | Interoperable Data Types | 8 |
| A.1.2 | The ISO_C_BINDING Module | 9 |
| A.1.3 | Procedure Interoperability | 9 |
| A.1.4 | Interoperability of Global Data | 9 |
| A.1.5 | Interfacing with C Pointers | 9 |
| A.1.6 | Interoperability of Arrays | 9 |
| A.2 | Historical Impact | 9 |
| A.3 | Further Enhancements with TS 29113 | 9 |
| A.4 | Additional Concepts from Chapter 20 | 9 |
| A.4.1 | Interoperability of Derived Types | 10 |
| A.4.2 | Procedure Interoperability | 10 |
| A.4.3 | Assumed-Type Dummy Arguments | 10 |
| A.4.4 | Optional Arguments | 10 |
| A.4.5 | Enumerations | 10 |
| A.5 | Final Notes | 10 |

1 About Fortran

Modern Fortran is an ideal choice for computationally intensive applications due to its efficient handling of arrays in numerical computations, vector formulation, advanced parallelism through coarrays, and seamless integration with C and other languages. Its object-oriented features enable large-scale software design, while array manipulation and intrinsic functions provide concise and readable code for mathematical and scientific applications. Fortran a robust, high-performance language for HPC, data analysis, and numerical modeling in the modern era.

1.1 Fortran: A Pioneering Language

Commodity-priced personal computers of the democratization of the Internet have spurred explosive evolution in computational science. At times, Fortran has spurred new paradigms, at other times responded. The watershed change was captured in the Fortran 90 standard. IBM, once guardian of the langue, saw their influence wane and they stumbled in the personal computer market. They fought to maintain the simplicity of the language and lost to those who wanted to include object oriented programming features, setting a new mindset over the language. Even more revolutionary, was the remake of Fortran into a vector language. While Cray supplied Fortran compilers with vector tools for Cray super computers, the new standard brought vector computing to the desktop, an astonishing and often overlooked breakthrough.

- **Fortran 90:**
 - Modular programming with `MODULE`.
 - Dynamic memory allocation and allocatable arrays.
 - Whole-array and elemental operations for numerical computations.
 - User-defined types and array slicing.
 - Enhanced control structures like `DO WHILE` and `SELECT CASE`.
- **Fortran 95:**
 - `FORALL` and `WHERE` constructs for parallelism.
 - `PURE` and `ELEMENTAL` procedures for functional programming.
- **Fortran 2003:**
 - Full object-oriented programming with type extension, polymorphism, and type-bound procedures.
 - C interoperability with the `ISO_C_BINDING` module and `BIND(C)` attribute.
 - Asynchronous and stream I/O for better data handling.
- **Fortran 2008:**¹
 - Introduction of coarrays for native parallel programming.
 - `SUBMODULES` for modular program decomposition.
 - `DO CONCURRENT` for loop-level parallelism.
 - `BLOCK` construct for nested variable scoping.
- **Fortran 2018:**
 - Enhanced coarrays with teams and events for parallel synchronization.
 - Support for optional arguments and assumed-rank arrays in C interoperability.
 - `FAIL IMAGE` and error codes in `STOP` statements for debugging.
 - Bitwise operations and improved intrinsic functions.
- **Fortran 2023 (Upcoming):**
 - Improved object-oriented features, including finalization and procedure pointers.
 - Automatic deallocation of allocatable arrays.
 - Further enhancements to coarray parallelism and task-based concurrency.
 - Simplified syntax and improved performance diagnostics.

¹The infamous `DO LOOP` is now extinct.

1.2 Why Fortran?

Perhaps a better question is Why aren't you using a vector language with intrinsic parallelism? For mankind to progress, we must rise beyond the dogmatic adherence to scalar and serial principles embodied in the cultural inertia of Excel.

2 Calling Fortran from Python

PyBind11 is a bridge between Python and Fortran providing a way to blend Python's usability with Fortran's performance, an approach particularly valuable in HPC and scientific workflows where computational efficiency and ease of scripting are paramount.

Integrating Fortran with Python combines the computational efficiency of Fortran with the flexibility and ecosystem of Python. This section demonstrates a practical approach to achieve this using PyBind11, highlighting examples and best practices.

2.1 Why PyBind11?

PyBind11 is a high-level library for binding C++ code to Python. Although designed for C++, it serves as an effective bridge between Python and Fortran by leveraging C++ wrappers for Fortran routines. Benefits include:

- Minimal overhead, allowing efficient execution of Fortran routines.
- High-level abstractions to simplify binding.
- Direct integration with Python's ecosystem.

2.2 Workflow Overview

The workflow to call Fortran from Python using PyBind11 involves three main steps:

1. Writing computational routines in Fortran and compiling them into a shared library.
2. Creating a C++ wrapper to expose the Fortran routines.
3. Using PyBind11 to bind the C++ wrapper to Python.

2.3 Example Implementation

2.3.1 Fortran Code

Here is a simple Fortran module that adds two arrays:

```
1 module myfortran
2 contains
3   subroutine add_arrays(a, b, result, n)
4     real, intent(in) :: a(:), b(:)
5     real, intent(out) :: result(:)
6     integer, intent(in) :: n
7     integer :: i
8
9     do i = 1, n
10      result(i) = a(i) + b(i)
```

```

11         end do
12     end subroutine add_arrays
13 end module myfortran

```

Listing 1: Fortran routine to add arrays.

Compile the Fortran code into a shared library:

Listing 2: Fortran compilation into a shared object.

```
$ gfortran -shared -fPIC -o libmyfortran.so myfortran.f90
```

2.3.2 C++ Wrapper

Create a C++ wrapper to call the Fortran routine. Use the `extern "C"` directive to ensure compatibility:

```

1  #include <pybind11/pybind11.h>
2  #include <pybind11/numpy.h>
3
4  extern "C" {
5      void add_arrays_(float* a, float* b, float* result, int* n);
6  }
7
8  namespace py = pybind11;
9
10 void add_arrays(py::array_t<float> a, py::array_t<float> b, py::array_t<float> result) {
11     auto buf_a = a.request();
12     auto buf_b = b.request();
13     auto buf_result = result.request();
14
15     if (buf_a.size != buf_b.size || buf_a.size != buf_result.size) {
16         throw std::runtime_error("Array sizes must match.");
17     }
18
19     int n = buf_a.size;
20     add_arrays_(static_cast<float*>(buf_a.ptr), static_cast<float*>(buf_b.ptr),
21                 static_cast<float*>(buf_result.ptr), &n);
22 }
23
24 PYBIND11_MODULE(myfortran, m) {
25     m.doc() = "Python interface for Fortran routines";
26     m.def("add_arrays", &add_arrays, "Add two arrays using Fortran");
27 }

```

Listing 3: C++ wrapper for the Fortran routine.

Compile the C++ wrapper with PyBind11:

Listing 4: Compiling the PyBind11 wrapper.

```
c++ -O3 -Wall -shared -std=c++17 -fPIC $(python3 -m pybind11 --includes) \
    wrapper.cpp -o myfortran$(python3-config --extension-suffix) -L. -lmyfortran
```

2.3.3 Python Code

Finally, use the Fortran routine from Python:

```

1 import numpy as np
2 import myfortran
3
4 a = np.array([1.0, 2.0, 3.0], dtype=np.float32)
5 b = np.array([4.0, 5.0, 6.0], dtype=np.float32)
6 result = np.empty_like(a)
7
8 myfortran.add_arrays(a, b, result)
9 print("Result:", result)

```

Listing 5: Calling the Fortran routine in Python.

2.4 Alternatives to PyBind11

While PyBind11 is versatile, other options exist for calling Fortran from Python:

- **F2PY:** A Fortran-to-Python interface generator that is part of NumPy. Ideal for pure Fortran-Python workflows, [1, 2].
- **ctypes or CFFI:** Directly interact with shared libraries using Python’s built-in libraries. Suitable for simple use cases.
- **Cython:** Provides fine-grained control over bindings but involves more boilerplate code than PyBind11.
- **pyfort:** Provides fine-grained control over bindings but involves more boilerplate code than PyBind11.

3 Fortran Interoperability With C

3.1 Fortran and C Interoperability

Fortran 2018 introduced significant advancements in interoperability with the C programming language, marking a major breakthrough in combining the strengths of these two languages. This section highlights the key features of Fortran-C interoperability as defined in the Fortran 2018 standard.

3.2 Key Features of Interoperability

Fortran 2018 provides robust mechanisms to facilitate seamless integration between Fortran and C, ensuring efficient and consistent data exchange and procedure calling conventions. The following are the main features of Fortran-C interoperability:

3.2.1 Interoperable Data Types

The standard defines a set of data types that are compatible between Fortran and C. These data types ensure seamless translation and interpretation of data structures when shared across both languages.

3.2.2 The ISO_C_BINDING Module

The ISO_C_BINDING intrinsic module introduces named constants and derived types that map Fortran types to their C counterparts. This module ensures consistent interpretation and compatibility of data types across both languages.

3.2.3 Procedure Interoperability

Fortran procedures can be made accessible to C, and vice versa, by using the BIND(C) attribute. This attribute specifies the linkage convention and optionally the external name to be used in the C environment, enabling seamless procedure calls between the languages.

3.2.4 Interoperability of Global Data

Global variables can be shared between Fortran and C by applying the BIND(C) attribute. This allows both languages to access and modify the same global data structures, ensuring consistency across the codebase.

3.2.5 Interfacing with C Pointers

The ISO_C_BINDING module includes derived types such as C_PTR and C_FUNPTR, which facilitate the interaction with C pointers and function pointers. These types ensure that pointer operations remain compatible and error-free.

3.2.6 Interoperability of Arrays

Guidelines are provided in the Fortran 2018 standard for handling array descriptors. These guidelines ensure that arrays, whether passed by Fortran or C, are correctly interpreted and manipulated across the languages.

3.3 Historical Impact

The introduction of these features in Fortran 2018 marked a significant milestone in scientific and engineering computing. By enabling seamless interoperability with C, Fortran retained its dominance in numerical computing while leveraging the extensive ecosystem of C libraries and tools. This advancement greatly simplified the process of integrating Fortran with modern software stacks.

3.4 Further Enhancements with TS 29113

The Technical Specification TS 29113, titled *Further Interoperability of Fortran with C*, builds upon the Fortran 2018 standard to offer extended capabilities and clarifications. Notable additions include support for assumed-shape arrays and optional dummy arguments in interoperable procedures. For more details, refer to <https://j3-fortran.org/doc/year/12/12-119.pdf>.

Fortran and C interoperability continues to empower developers to combine the computational efficiency of Fortran with the versatility of C, enabling powerful solutions for modern scientific and engineering challenges.

A Fortran's `iso_c_binding`

Fortran 2018 introduced significant advancements in interoperability with the C programming language, marking a major breakthrough in combining the strengths of these two languages. This section highlights the key features of Fortran-C interoperability as defined in the Fortran 2018 standard the following is excerpted and adapted from *Modern Fortran Explained: Incorporating Fortran 2023, 3rd Edition* by Metcalf, Reid, Cohen, and Bader (2024).

A.1 Key Features of Interoperability

Fortran 2018 provides robust mechanisms to facilitate seamless integration between Fortran and C, ensuring efficient and consistent data exchange and procedure calling conventions. The following are the main features of Fortran-C interoperability:

A.1.1 Interoperable Data Types

The standard defines a set of data types that are compatible between Fortran and C. These data types ensure seamless translation and interpretation of data structures when shared across both languages.

A.1.2 The `ISO_C_BINDING` Module

The `ISO_C_BINDING` intrinsic module introduces named constants and derived types that map Fortran types to their C counterparts. This module ensures consistent interpretation and compatibility of data types across both languages.

A.1.3 Procedure Interoperability

Fortran procedures can be made accessible to C, and vice versa, by using the `BIND(C)` attribute. This attribute specifies the linkage convention and optionally the external name to be used in the C environment, enabling seamless procedure calls between the languages.

A.1.4 Interoperability of Global Data

Global variables can be shared between Fortran and C by applying the `BIND(C)` attribute. This allows both languages to access and modify the same global data structures, ensuring consistency across the codebase.

A.1.5 Interfacing with C Pointers

The `ISO_C_BINDING` module includes derived types such as `C_PTR` and `C_FUNPTR`, which facilitate the interaction with C pointers and function pointers. These types ensure that pointer operations remain compatible and error-free.

A.1.6 Interoperability of Arrays

Guidelines are provided in the Fortran 2018 standard for handling array descriptors. These guidelines ensure that arrays, whether passed by Fortran or C, are correctly interpreted and manipulated across the languages.

| Type | Named Constant | C Type or Types |
|-----------|-----------------------|----------------------------|
| integer | c_int | int |
| | c_short | short int |
| | c_long | long int |
| | c_long_long | long long int |
| | c_signed_char | signed char, unsigned char |
| | c_size_t | size_t |
| | c_int8_t | int8_t |
| | c_int16_t | int16_t |
| | c_int32_t | int32_t |
| | c_int64_t | int64_t |
| | c_int_least8_t | int_least8_t |
| | c_int_least16_t | int_least16_t |
| | c_int_least32_t | int_least32_t |
| | c_int_least64_t | int_least64_t |
| | c_int_fast8_t | int_fast8_t |
| | c_int_fast16_t | int_fast16_t |
| | c_int_fast32_t | int_fast32_t |
| | c_int_fast64_t | int_fast64_t |
| | c_intmax_t | intmax_t |
| | c_intptr_t | intptr_t |
| real | c_float | float |
| | c_double | double |
| | c_long_double | long double |
| complex | c_float_complex | float _Complex |
| | c_double_complex | double _Complex |
| | c_long_double_complex | long double _Complex |
| logical | c_bool | _Bool |
| character | c_char | char |

Table 1: Named constants for interoperable kinds of intrinsic Fortran types. Adapted from Metcalf, Cohen, and Reid, Table 19.1.

A.2 Historical Impact

The introduction of these features in Fortran 2018 marked a significant milestone in scientific and engineering computing. By enabling seamless interoperability with C, Fortran retained its dominance in numerical computing while leveraging the extensive ecosystem of C libraries and tools. This advancement greatly simplified the process of integrating Fortran with modern software stacks.

A.3 Further Enhancements with TS 29113

The Technical Specification TS 29113, titled *Further Interoperability of Fortran with C*, builds upon the Fortran 2018 standard to offer extended capabilities and clarifications. Notable additions include

support for assumed-shape arrays and optional dummy arguments in interoperable procedures. For more details, refer to <https://j3-fortran.org/doc/year/12/12-119.pdf> [.]

A.4 Additional Concepts from Chapter 20

The following are additional interoperability concepts excerpted and adapted from *Modern Fortran Explained, 5th Edition* by Metcalf, Reid, Cohen, and Bader (2024):

A.4.1 Interoperability of Derived Types

Derived types can be made interoperable with C structures by using the `BIND(C)` attribute. For example, a Fortran type can interoperate with a C structure if they have the same number of components, and each component is interoperable. Components must be declared in the same order.

A.4.2 Procedure Interoperability

Fortran procedures can interoperate with C function prototypes if they declare arguments and results that match the corresponding C types. A Fortran explicit-shape or assumed-shape array can interoperate with a C array.

A.4.3 Assumed-Type Dummy Arguments

The assumed-type syntax `TYPE(*)` allows interoperability with C types where no type information is specified. Assumed-type dummy arguments can be scalar, assumed-shape, or rank-agnostic arrays.

A.4.4 Optional Arguments

Fortran allows optional arguments to interoperate with C. If an argument is not present, Fortran passes a null pointer to the C procedure, enabling flexible interoperability with optional parameters in C.

A.4.5 Enumerations

Enumerations declared with `BIND(C)` enable Fortran to interoperate with C `enum` types. For example, a C enumeration can map directly to a Fortran kind value.

A.5 Final Notes

Fortran and C interoperability continues to empower developers by combining the computational efficiency of Fortran with the extensive library ecosystem of C, enabling robust solutions for scientific and engineering challenges.

References

- 1 NumPy Developers. *F2PY Documentation (Stable)*. Accessed: 2024-12-11. 2024. URL: <https://numpy.org/doc/stable/f2py/>.
- 2 NumPy Developers. *F2PY Usage Guide*. Accessed: 2024-12-11. 2024. URL: <https://numpy.org/doc/2.1/f2py/usage.html>.