This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MCSE.2022.3159862, Computing in Science & Engineering

Computing in Science & Engineering
Volume 24, Number 2, March-April 2022, pp. 63-72
DOI: 10.1109/MCSE.2022.3159862

Feature Article
Editor: Konrad Hinsen, konrad.hinsen@cnrs.fr

The State of Fortran

Laurence Kedward
Department of Aerospace Engineering, University of Bristol, Bristol, UK
Corresponding author: laurence.kedward@bristol.ac.uk

Bálint Aradi
Bremen Center for Computational Materials Science, University of Bremen, Germany

Ondřej Čertík
Los Alamos National Laboratory

Milan Curcic
University of Miami, Miami, FL, USA

Sebastian Ehlert
Mulliken Center for Theoretical Chemistry, Institut für Physikalische und Theoretische Chemie, Universität Bonn, Germany

Philipp Engel
Institut für Geodäsie und Geoinformationstechnik, Technische Universität Berlin, Germany

Rohit Goswami
Quansight Labs

Michael Hirsch
Center for Space Physics, Boston University, Boston, Massachusetts, USA

Asdrubal Lozada-Blanco
São Carlos Institute of Physics, University of São Paulo, São Carlos, SP, Brazil

Vincent Magnin
Univ. Lille, CNRS, Centrale Lille, Univ. Polytechnique Hauts-de-France, IEMN, Lille, France

Arjen Markus
Deltares Research Institute, The Netherlands

Emanuele Pagone
Cranfield University, Sustainable Manufacturing Systems Centre, School of Aerospace Transport and Manufacturing, Cranfield, UK

Ivan Pribec
Chair of Brewing and Beverage Technology, Technical University of Munich, Germany

Brad Richardson
Archaeologic, Inc., CA, USA

Harris Snyder
Structura Biotechnology Inc., Toronto, Ontario, Canada

John Urban
HPC Consultant, USA

Jérémie Vandenplas
Animal Breeding and Genomics, Wageningen UR, P.O. 338, 6700 AH, Wageningen, The Netherlands

***Abstract*—A community of developers has formed to modernize the Fortran ecosystem. In this article, we describe the high-level features of Fortran that continue to make it a good choice for scientists and engineers in the 21st century. Ongoing efforts include the development of a Fortran standard library and package manager, the fostering of a friendly and welcoming online community, improved compiler support, and language feature development. The lessons learned are common across contemporary programming languages and help reduce the learning curve and increase adoption of Fortran.**

© 2022 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works
1521-9615 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

FORTRAN is a high-level programming language primarily used to solve scientific and engineering problems. It has been under active development since its inception under John Backus at IBM in 1954 to the present day. The initial goal was to ease the translation of mathematical formulas to optimized machine code instructions, a concept now known as compilation. The intuitive abstraction of mathematical procedures enabled rapid development of numerical solutions to scientific problems, at a time when most programs were still hand-coded in assembly language. Following the release of its first implementation in 1957, the language was adopted by the scientific and engineering communities for writing numerical programs. As a result, the language was quickly ported to several computer architectures such that Fortran is accepted as being the first cross-platform programming language.

The ISO Fortran Standard and its maintenance of backwards compatibility provide guarantees for language stability and code longevity. Consequently, there is a mature and well-established ecosystem of Fortran compilers and libraries—what is lesser known is how much the language has evolved since its beginnings. In this article, we first review the high-level features that continue to make Fortran a powerful and effective tool for high-performance, scientific computing. Second, we present recent progress and efforts towards uniting the Fortran community and improving the tooling and resources available to Fortran programmers. Finally, we provide an outlook on the future of Fortran and its burgeoning ecosystem.

WHY FORTRAN?

Fortran is perceived in some software development circles as archaic, lacking the features and conveniences of newer languages, and characterized by an obtuse syntax. However, such considerations typically stem from a lack of familiarity with Fortran standards later than Fortran 77 [1]. Similarly, later updates to the Fortran language competed with newer alternatives such as C, MATLAB and Python that built on the high-level concepts first introduced in Fortran.

The Fortran language has seen many revisions [2]–[5], with the latest international standard being Fortran 2018. Though a large body of Fortran code written in the 1970s to 1980s remains in use to this day, the language and its paradigms have evolved significantly. In the remainder of this section we present the key features that make Fortran an effective tool for high-performance numerical computing.

High-Level Language Features

Key to the early popularity and continued use of Fortran are the high-level abstractions provided by the language (Figure 1). Intrinsic support for multidimensional arrays in combination with safe dynamic allocation and expressive array slicing, makes for a readable and easy-to-learn language suitable for high-performance numerical applications. Strong, static typing helps avoid common programming pitfalls via meaningful compile-time errors without sacrificing runtime performance. Similarly, modules and submodules allow for code organisation and reuse with automatic checking of routine signatures and types. Fortran `allocatable` arrays enable dynamic allocation without memory leaks and are guaranteed to be non-aliasing. As a result, efficient machine code can be generated by the compiler handling `allocatable` arrays, without requiring additional annotations such as the C `restrict` keyword. Moreover, the array-oriented design of the language means that operator overloading with object arrays can be optimized by the compiler without requiring complex expression templates as in C++. Additional language keywords that aid compiler optimization and ensure compile-time correctness include: the `intent` attribute for indicating whether procedure arguments are inputs, outputs, or bidirectional; the `pure` attribute indicates procedures that have no external side effects; and the `elemental` attribute maps a procedure to each element of an N-dimensional array or scalar, useful for rank polymorphism and akin to the map operation in functional programming languages.

Fortran natively supports arrays up to rank 15, complex numbers, object-oriented programming, and syntax for expressing shared-memory and distributed-memory parallelism. The Fortran native parallelism offers compiler-enforceable safeguards against many pitfalls of parallel programming. Shared-memory parallelism for ‘do‘ loops, which has been supported through the OpenMP

© 2022 IEEE
Published by the IEEE Computer Society

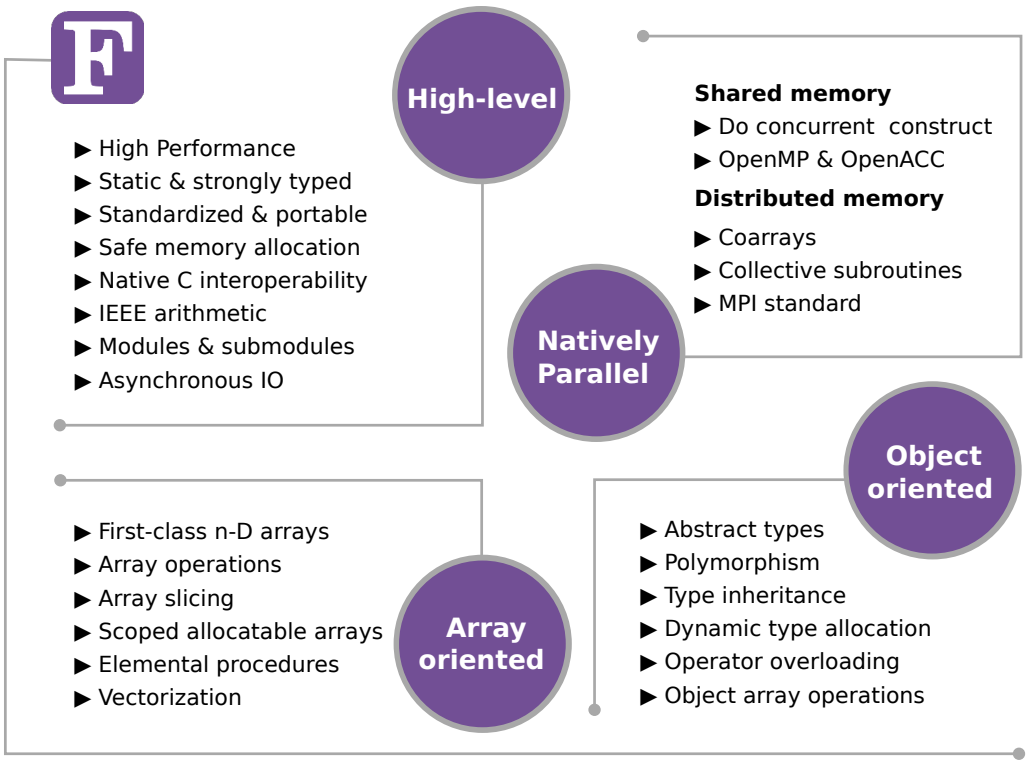


Figure 1. High-level language features of Fortran

Listing 1. Do concurrent example

```
do concurrent (i = 1:n, j = 1:m)
  u(i,j) = a * v(j,i)
end do
```

standard for many years, can now be expressed with the new `do concurrent` syntax to inform the compiler that loop iterations may be executed in any order using threading, SIMD (single instruction, multiple data), or any other parallel implementation available to the compiler. For example, the freely available NVIDIA *nvfortran* compiler can automatically offload standard `do concurrent` loops for execution to NVIDIA GPUs. Listing 1 shows a two-dimensional loop using `do concurrent`.

Distributed-memory parallelism has typically been implemented in Fortran using the MPI standard. Fortran 2008 introduced intrinsic abstractions for distributed-memory parallelism in the form of *coarrays*. Parallel Fortran programs follow the SPMD (single program, multiple data) paradigm, where multiple copies of the same executable are started as different processes, with each process working on a different piece of the problem. These processes (*images*, in Fortran terminology) have independent address spaces but can exchange data using coarrays or collective subroutines, introduced in Fortran 2018. If a variable is declared as a coarray, an image can read and write the value of that variable on other images with a simple syntax that is similar to array indexing (an image number is used instead of an array index). Synchronizing images is as simple as issuing a `sync all` statement. More fine-grained control is possible with subsets of workers known as *teams*, and event-based scheduling with the *events*. Coarrays, collectives, teams, and events abstract away much of the verbosity required by MPI thereby helping programmers to focus on their application rather than the low-level details of interprocess data

exchange. Several coarray implementations are based on MPI, allowing Fortran programmers to leverage the excellent hardware interconnects supported by MPI libraries. Reid *et al.* summarize the evolution of Fortran's parallel features [6].

The `iso_c_binding` module is an essential feature introduced in Fortran 2003 and has been further extended in subsequent versions. It provides standardized interoperability with the C programming language to access procedures, structures, and global variables to and from Fortran. Nearly any programming language that has a C interface such as Python or C++ can interact with compiled Fortran code in a platform-agnostic manner. Foreign functions are bound through interface declarations that often can be generated automatically and are called like any other native Fortran procedure. The resulting application is linked against the corresponding C library or object file. Consequently, the programmer can reuse source code in a mixed-language environment or can outsource tasks like connecting to hardware via OEM drivers written in C. A broad range of interface libraries is already available, including common use-cases: system libraries (libc, POSIX); toolkits (GTK, Motif); abstraction layers (OpenGL, SDL 2.0, Curses); scripting languages (Lua, Python, MATLAB); and networks (cURL, FastCGI). The `iso_c_binding` does not cover all C99 features—for instance, variadic functions which have no equivalent in Fortran—however the binding continues to be under development for future Fortran revisions.

Additionally, Fortran procedures can be exposed to C or any other programming language with foreign function interface, such as Python, Lua, or Rust. Interoperable functions and subroutines are indicated by the `bind(c)` attribute and the standardized `ISO_Fortran_binding.h` header allows calling Fortran procedures from C or other languages using a C structure to interface with Fortran array descriptors.

Stability and Reliability

As an actively used language with a long history, Fortran and its associated compilers have long maintained excellent backward compatibility and language stability. This is important for many well-established and large Fortran projects that are critical for numerically intensive tasks. For such large projects, a fast-developing language with breaking API changes would incur significant costs to the maintainers in person-hours of redevelopment and revalidation time. By contrast, Fortran offers inherent guarantees of code longevity and indeed many projects in use today are largely unchanged from their original release despite being written under Fortran standards dating from decades ago.

Importantly, anyone writing Fortran software today can have confidence not only in the longevity of their code but also in that of any Fortran libraries and applications they use. This inherent future-proofing of Fortran code therefore protects against so-called *code rot* or *software collapse* [7] whereby code becomes unusable due to breaking changes in its dependencies or its implementation language.

Mature

A key accomplishment of Fortran 90 and 95 was to incorporate dynamic arrays and *de facto* procedures and syntax widely adopted by compilers of the time. Along with the C interoperability mentioned previously, Fortran 2003 brought improved operations for interacting with the environment, such as reading environment variables, reading command-line arguments, and running external programs. Remaining gaps in systems programming such as filesystem operations are being addressed in the growing Fortran standard library described in the following section. While some projects have recently moved from Fortran to C++ or other languages, the underpinnings of numerical computing remain implemented in Fortran. Often a user may not be aware they are using Fortran via another language such as the Python NumPy or SciPy libraries which are widely used among the scientific community.

There are several mature Fortran compilers, many of which support the latest Fortran standards. The OpenMP and OpenACC standards provide directives for parallel execution in a platform-agnostic manner. NVIDIA promotes CUDA Fortran and OpenACC in the `nvfortran` compiler for offloading of parallel constructs to the GPU. The MPI-3 standard continues to support modern Fortran syntax with polymorphic interfaces scalable from multi-core

embedded systems to the largest supercomputers.

New books continue to be written on modern Fortran, major funding agencies across the globe continue to fund upgrades and new projects in Fortran, and graduate students in many STEM disciplines continue to learn Fortran. The Fortran community, as discussed in the next section, has renewed vigor and is rapidly growing the Fortran ecosystem using best practices of modern software development.

It is for these reasons—performance, ease-of-use, productivity, portability, stability and longevity—that we advocate for Fortran. In the remainder of the paper, we describe the shortcomings of Fortran and its ecosystem, and present the ongoing community efforts to address them.

FORTRAN-LANG

A New Community for Fortran Users

Despite recent revisions to the language and continuing to be a large share of the software executed on major High Performance Computing (HPC) systems, Fortran's ecosystem has stagnated across multiple fronts. First, the lack of a standard library, a common resource in modern programming languages, makes mundane general-purpose programming tasks difficult. Second, building and distributing Fortran software has been relatively difficult, especially for newcomers to the language. Third, Fortran does not have a community maintained compiler like Python, Rust or Julia has, that can be used for prototyping new features and is used by the community as a basis for writing tools related to the language. Finally, Fortran has not had a prominent dedicated website—an essential element for new users to discover Fortran, learn about it, and get help from other Fortran programmers. In the same way, Fortran is no longer widely taught to university students or valued as a useful skill by industry. As a consequence, adoption of new users has been stagnating, large scientific Fortran projects have been migrating to other languages, and the communities of Fortran programmers remained scattered and isolated.

To address these issues, a new open source Fortran community called *Fortran-lang* was formed in December 2019 [8]. Its initial conception came about in the J3 Fortran Proposals repository on GitHub, described in a later section, where it became clear that there was a need for modern tooling and an improved web presence for the Fortran community. We describe these projects in more detail in the rest of this section.

Online channels

Modern computer languages generally have a website, offering resources to new and existing users all in one place. Several online resources have existed for some time—notably, the *comp.lang.fortran* Usenet group, the Fortran Wiki[1], and the https://fortran90.org site—however, there has never been a central web presence run by the community until now. The new https://fortran-lang.org website provides learning resources, a list of open source and commercial compilers, guides to contribute to the community projects, a Fortran package index, and a monthly newsletter.

Besides the main website, Fortran-lang manages a traditional mailing list for news and announcements, a @fortranlang Twitter account, and the Fortran Discourse[2] which provides a friendly and welcoming online discussion board for all Fortran programmers—existing and aspiring alike. The new Discourse forum has proved a popular success, totaling more than 300 users in one year, with threads covering a wide variety of interesting Fortran-related topics.

At the core of the new Fortran-lang community have been its flagship projects—the Fortran Standard Library and the Fortran Package Manager—which were identified early on as critical for improving the Fortran ecosystem and tooling. The development of these projects has been performed openly and collaboratively thanks to the distributed version control software *Git* and the popular hosting platform *GitHub*. GitHub allows everyone to clone an open source project, test it, modify the code and offer it to the community via a "Pull Request" to be reviewed. Everyone can therefore easily contribute to the projects of the Fortran-lang GitHub organization[3] and, at the time of writing, there are 164 contributors to the main Fortran-lang projects through code contributions and discussions. This includes six

[1] http://fortranwiki.org
[2] http://fortran-lang.discourse.group
[3] https://github.com/fortran-lang

student interns, funded by the Google Summer of Code initiative, to work on Fortran-lang projects in 2021. In the following sections, we present the new standard library and package manager projects.

THE FORTRAN STANDARD LIBRARY

Motivation

Although the Fortran Standard specifies a number of built-in procedures and modules (collectively called *intrinsics*), it does not define a standard library that is common to languages like C, C++, Python, or Rust. Currently, Fortran 2018 defines 168 intrinsic procedures and a small number of constants and derived types, but no widely used data structures like linked lists and dictionaries.

Thus, commonly used features—string and error handling, high-level I/O, statistical methods—are programmed by users in-house, creating many redundant implementations with different levels of correctness and rigor. Proliferation of tailor-made, common functionalities is inefficient and error-prone, and discourages the adoption of the language in emerging fields, reducing its future relevance. For example, Fortran has not seen much application in the computationally-intensive area of neural networks for machine learning despite being a performant parallel array-oriented language. Fortran neural-network frameworks do exist [9], but the majority of deep learning is performed in Python. Though not originally designed for high-performance numerical work, Python has attracted machine learning practitioners with its ecosystem of packages and rich standard library. Furthermore, Fortran intrinsics have often been formally standardized before being prototyped in compilers, thus slowing down the modernization of the language and creating opportunities for poor API design.

The Fortran standard library (*stdlib*) project aims at providing community-developed, robust and reusable reference implementations of commonly used procedures, in a versioned and well-tested library. Consequently, *stdlib* intends to reduce practitioners duplication of effort and to help lower the bar for Fortran's adoption. A future aim is to collaborate with the Fortran Standard Committee and compiler developers to incorporate relevant procedures into the language intrinsics and provide optimized implementations alongside compilers.

Scope

In addition to standardizing common interfaces for core functionalities—like input and output (I/O), containers, and filesystem access *etc.*—the scope of *stdlib* is akin to that of NumPy [10], SciPy [11], and MATLAB, thus providing procedures useful for science, engineering, and mathematics. As of July 2021, *stdlib* contains 17 modules that contain procedures for handling errors and optional arguments, working with bitsets, facilitating I/O operations, linear algebra, logging, numerical integration, descriptive statistics, sorting, and strings. More information can be found in full API documentation online (https://stdlib.fortran-lang.org) generated by the FORD tool [12].

Development

To provide generic interfaces for all supported types, kinds, and ranks, *stdlib* uses the Fypp[4] preprocessor to generate code from templates. This is required since generic programming in Fortran is currently limited to mandate for every combination of type, kind and rank an implementation. However, all such procedures are then usually invoked using a common generic name.

To ensure cross-platform support, *stdlib* exploits GitHub's Continuous Integration (CI) pipeline to test all procedures, derived types, and modules on the main branch and on any changes proposed in pull requests. The CI pipeline is run across a combination of different compilers and operating systems (Table 1); beyond this, *stdlib* should be supported on any platform with Python (for Fypp) and a Fortran 2008 compliant compiler.

THE FORTRAN PACKAGE MANAGER

Motivation

The translation of a compiled language source code to machine instructions in a large project, involves scheduling source files for compilation, taking into account any inter-file dependencies, and linking the resulting objects together and

[4]https://github.com/aradi/fypp

Table 1. Operating systems and compilers on which the Fortran Standard Library is regularly tested

Operating system	Architecture	Compiler	Version
MacOS Catalina 10.15	x86_64	GCC Fortran	9, 10, 11
		Intel oneAPI classic	2021.1
Ubuntu 20.04	x86_64	GCC Fortran	9, 10, 11
		Intel oneAPI classic	2021.1
Microsoft Windows Server 2019	x86_64	GCC Fortran (MSYS)	10
		GCC Fortran (MinGW)	10

with any third-party dependencies. A typical Fortran project relies on hand-coded Makefiles (not portable), CMake (complicated to learn and use), or custom build scripts (difficult to maintain) to perform the build process. Thus, a new Fortran programmer not only needs to learn Fortran and how to apply it to solve their problem, but they also need to learn one or more build systems. Moreover, such language-agnostic build systems have no easy way to use existing Fortran libraries as dependencies in new projects. For experienced Fortran programmers, this has been the way of life—but for newcomers it is a serious barrier to entry.

Newer languages like Rust (cargo), Racket (raco), Haskell (cabal) and Python (distutils) have simplified the process of fetching dependencies and building/running the ensemble by abstracting much of the aforementioned complexity through a language-specific build system or package manager.

The goal for the Fortran Package Manager (*fpm*) is to have a Fortran-specific build system and package manager to simplify compiling Fortran code and using third-party dependencies. *Fpm* removes the need to learn about and maintain complex build systems thereby simplifying the learning curve for newcomers.

Design Considerations

Fpm's functionality is accessed by an intuitive command-line interface modeled after Rust's package manager Cargo. It covers common Fortran project tasks including: creating new projects; compiling and linking; running applications and tests; and fetching third-party dependencies to use within the current project (Figure 2).

The minimal requirement for packages to be used with *fpm* is the inclusion of the package manifest file, written using the TOML[5] format. It allows *fpm* to automatically find sources, executables, and unit tests, if they are placed within the default folder structure. The manifest file can contain additional metadata (author, maintainer, license, copyright, *etc.*) or options for finer control of *fpm*'s behavior. *Fpm* is written in Fortran and built by itself or from a single source file with just a Fortran compiler.

A key feature of *fpm* is the ability to easily reuse code between projects. Reusing compiled Fortran modules has been difficult due to missing Application Binary Interface (ABI) compatibility between compilers, and even between versions of the same compiler. *Fpm* solves this by providing first-class support for handling dependencies between Fortran projects while keeping builds with different compilers and options separate. Additionally, complex compiler configuration is abstracted away from the user while retaining access to commonly used debugging options for development and recommended optimization settings for production.

Fpm's first class support for dependency management makes it straightforward to specify dependencies and control their exact version. If a project and all of its dependencies are handled by *fpm*, stronger guarantees are possible for effortless and reproducible builds. Other aspects that can be controlled are the compiler, the compiler version, the build flags, and the operating system. With a project's dependencies and compiler flags fully specified, one only needs to define an operating system, compiler and its version to reproduce a specific build, which is manageable even for new users. This solves what has historically been an intractable problem for many Fortran programmers and their projects.

[5] https://toml.io

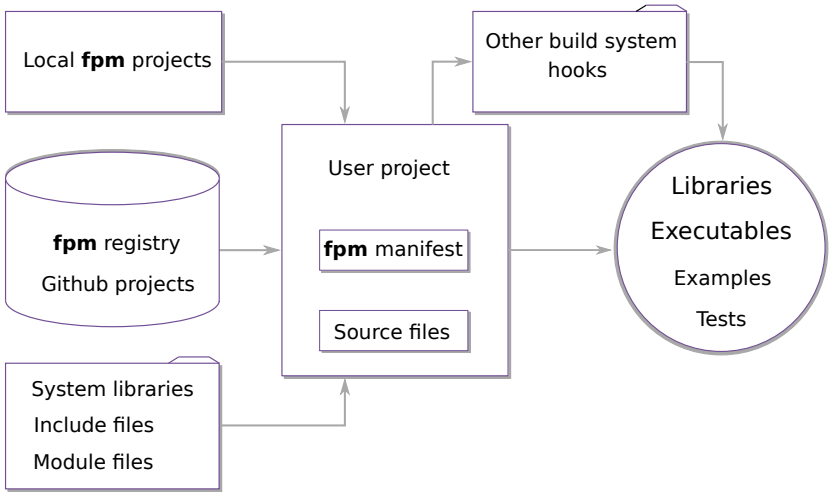


Figure 2. Fortran package manager workflow

Future features

Ongoing work is currently focused on providing fine-grain control of compiler flags under different build *profiles* within the manifest file. This process is complicated by the many different available Fortran compilers (each with their own flags), the intricate semantics that are associated with certain flags, and linking with platform-specific resources. *Fpm*'s end goal in this respect is to abstract away all of the complications in a compiler-agnostic manner. Furthermore, *fpm* will eventually provide native support for using features such as *MPI*, *OpenMP*, *OpenACC*, and *Coarrays*.

To help newcomers who are unfamiliar with command-line interfaces, a cross-platform graphical user interface is in development. Alongside common tasks such as building and installing, the prototype allows setting build options, such as "debug" or "release" as well as browsing the online *fpm* registry and navigating to the webpage of a selected package.

Finally, peripheral features common to other language-specific package managers are planned, including the ability to query and download packages from one or more central registries with the command line, and integrated support for documentation generation.

THE FUTURE IS BRIGHT

In addition to the *stdlib* and *fpm* projects, there are several other ongoing community efforts to improve the Fortran language, its compilers, and its outreach.

Evolving the language as a community

Development of the Fortran language standard follows the ISO standardization workflow which is led by national Fortran standards committees and the working group WG5 of the ISO/IEC JTC 1/SC22 standardization subcommittee. Committee members are typically delegated by participating organizations, such as national labs, research institutes, or compiler and hardware vendors. While the formal standardization procedure is beneficial for maintaining quality and robustness of the language, the period for a proposed feature to become part of the standard is consequently very long as compared to popular non ISO-standardized languages like Python or Rust. Moreover, the language development is quite decoupled from the wider community of Fortran programmers.

To address this shortcoming, two members of the U.S. Fortran Standards Committee suggested establishing a public Git repository[6] for language proposals. The repository was created in October 2019 on GitHub and became an immediate suc-

[6]https://github.com/j3-fortran/fortran_proposals

cess, highlighting a long-standing desire within the community to participate in developing the language. Community members are now able to easily propose and discuss additions to the language which, after discussion and consensus, can be made into formal proposals for discussion by the committee. More than 190 issues and 25 pull requests have been created so far, and some of the suggested features are expected to become part of the Fortran future 202Y standard. Similarly, the highly discussed topic of improving generic programming in Fortran has had a separate working group and public repository[7] set up to address this widely sought after functionality.

Compiler development

Fortran has several free-to-use compilers (Table 2), of which GFortran—part of GCC—is the most advanced open source option (licensed under GPL). It supports most of the latest Fortran standards and is the recommended option for a production compiler today.

Three new Fortran compilers, based on LLVM, are currently under development: Flang, which is part of the LLVM project, ifx by Intel and LFortran[8].

LFortran is a modern open-source (BSD licensed) interactive Fortran compiler built on top of LLVM. Its goal is to become a community supported compiler that can be used to prototype new features and develop new tools for Fortran. It can execute user's code interactively for exploratory work at the command line or in a Jupyter notebook (much like Python, MATLAB or Julia) as well as compiling to binaries. LFortran is currently under active development with the aims to support all of Fortran 2018 and target modern architectures such as multi-core CPUs and GPUs.

LFortran has separate Abstract Syntax Tree (AST) and Abstract Semantic Representation (ASR) structures which allows for greater flexibility in the compiler. For example, the AST representation can be used as an automatic code formatter. Likewise, the standalone ASR can be the input of multiple backends for code generation. Indeed, LFortran has an LLVM backend, a fast x86 one, and a code generator for C++.

[7] https://github.com/j3-fortran/generics
[8] https://lfortran.org

Research and dissemination

A great landmark for Fortran was the inaugural FortranCon 2020[9], an international conference which took place on July 2–4, 2020, in Zürich, Switzerland. Lead by Tiziano Müller (Department of Chemistry, University of Zürich) and Alfio Lazzaro (Hewlett Packard Enterprise, Switzerland), the conference attracted more than 270 participants for two full days of talks and a half-day workshop on object-oriented programming in Fortran. The keynote talk *"Fortran 2018… and Beyond"* was delivered by Steve Lionel giving an overview of the recent standardization efforts; this was followed by talks covering community projects, scientific applications, compilers, programming tools, parallel programming, and interoperability with other languages. FortranCon 2021 was held in September 2021, with similar scope of presentation and attendance as the inaugural one. We hope that future FortranCon events continue to amplify the work of Fortran practitioners, enhance community-building, and increase language adoption.

CONCLUSIONS

The Fortran language has evolved enormously since its invention over sixty years ago such that it continues to offer a wealth of useful high-level abstractions for engineers and scientists. Moreover, the language still retains its high performance through its array oriented design, strong static guarantees, and its native support for shared memory and distributed memory parallelism.

In this article we have exposed the lesser-known, high-level features of Fortran and highlighted the key advantages for code longevity and stability. Similarly, we have identified the main limitations to using Fortran in the modern age, and have described recent efforts by the new Fortran-lang online community at addressing these. In particular are the provision of a central online web presence with a moderated discussion board; the collaborative development of a Fortran standard library and Fortran package manager; and community led efforts to develop the language and its compilers.

With a burgeoning ecosystem of tooling and open source code, a growing online community,

[9] https://tcevents.chem.uzh.ch/event/12/

Table 2. Summary of free-to-use multi-platform Fortran compilers

gfortran	Open source, GPLv3	Full support for F2003, partial support for F2008 and F2018
Classic flang	Open source, Apache-2.0	Full support for F2003, to be superseded by LLVM flang
LLVM flang	Open source, Apache-2.0	Under development, full support for parsing F2018
LFortran	Open source, BSD-3-Clause	Under development, full support for parsing F2018
Intel Classic (ifort)	Proprietary, Intel	Full support for F2018
Intel LLVM (ifx)	Proprietary, Intel	Beta development, full support for F95
nvfortran	Proprietary, NVIDIA	Full support for F2003, partial support for F2008

and ongoing development of the language and its compilers, Fortran is on track for continued and increased growth into the future.

ACKNOWLEDGMENT

We gratefully acknowledge all contributors to the Fortran-lang open source projects, Fortran Discourse participants, compiler developers, and Fortran Standard Committees. We thank Pierre De Buyl and two anonymous reviewers whose comments helped improve this article.

REFERENCES

1. V. K. Decyk, C. D. Norton, and H. J. Gardner, "Why Fortran?" *Computing in Science Engineering*, vol. 9, no. 4, pp. 68–71, 2007. [Online]. Available: https://doi.org/10.1109/MCSE.2007.89
2. M. Metcalf, "The seven ages of Fortran," *J. Comput. Sci. Technol.*, vol. 11, 01 2011.
3. J. Reid, "The new features of Fortran 2003," vol. 26, no. 1, p. 10–33, Apr. 2007. [Online]. Available: https://doi.org/10.1145/1243413.1243415
4. ——, "The new features of Fortran 2008," vol. 33, no. 2, p. 21–37, Jul. 2014. [Online]. Available: https://doi.org/10.1145/2647342.2647344
5. ——, "The new features of Fortran 2018," *SIGPLAN Fortran Forum*, vol. 37, no. 1, p. 5–43, Apr. 2018. [Online]. Available: https://doi.org/10.1145/3206214.3206215
6. J. Reid, B. Long, and J. Steidel, "History of coarrays and SPMD parallelism in Fortran," *Proceedings of the ACM on Programming Languages*, vol. 4, no. HOPL, pp. 1–30, 2020. [Online]. Available: https://doi.org/10.1145/3386322
7. K. Hinsen, "Dealing with software collapse," *Computing in Science Engineering*, vol. 21, no. 3, pp. 104–108, 2019. [Online]. Available: https://doi.org/10.1109/MCSE.2019.2900945
8. M. Curcic, O. Čertík, B. Richardson, S. Ehlert, L. Kedward, A. Markus, I. Pribec, and J. Vandenplas, "Toward modern Fortran tooling and a thriving developer community," *arXiv preprint arXiv:2109.07382*, 2021. [Online]. Available: https://doi.org/10.48550/arXiv.2109.07382
9. M. Curcic, "A parallel Fortran framework for neural networks and deep learning," in *ACM SIGPLAN Fortran Forum*, vol. 38, no. 1. ACM New York, NY, USA, 2019, pp. 4–21. [Online]. Available: https://doi.org/10.1145/3323057.3323059
10. C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2
11. P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright *et al.*, "SciPy 1.0: fundamental algorithms for scientific computing in Python," *Nature methods*, vol. 17, no. 3, pp. 261–272, 2020. [Online]. Available: https://doi.org/10.1038/s41592-019-0686-2
12. C. MacMackin, "FORD," Jun. 2018. [Online]. Available: https://doi.org/10.5281/zenodo.1422473

Laurence J. Kedward is a research associate in the department of aerospace engineering at the University of Bristol, UK where his research interests are computational fluid dynamics and numerical optimization.

Bálint Aradi is a senior researcher at the Bremen Center of Computational Materials Science at the University of Bremen, Germany where he develops and applies quantum mechanical simulation methods to explore materials properties on the atomistic scale.

Ondřej Čertík is a scientist at Los Alamos National Laboratory. He is the original author of the SymPy, SymEngine, and LFortran open source projects, and co-founder of Fortran-lang.

Milan Curcic is an assistant scientist at the University of Miami. He specializes in ocean wave physics and numerical weather prediction.

Sebastian Ehlert is a PhD student at the University of Bonn (Germany), in the Mulliken Center for Theoretical Chemistry, where he is working on the development of fast and robust semiempirical electronic structure methods.

Philipp Engel is a PhD student in Geodesy at the Technical University of Berlin (Germany), where his research includes sensor networks, deformation monitoring, and spatial data analysis.

Rohit Goswami maintains f2py within NumPy as a software engineer at Quansight Labs.

Michael Hirsch is a research scientist with the Center for Space Physics at Boston University. He uses Fortran to model sensor and system performance from subsurface to space including the atmospheric and ionospheric response.

Asdrubal Lozada-Blanco is a postdoctoral researcher in the São Carlos Institute of Physics at the University of São Paulo (Brazil), where he develops simulation methods in statistical mechanics.

Vincent Magnin is an associate professor at the University of Lille (France), in the Institut d'Electronique, de Microélectronique et de Nanotechnologie, where his interests are the numerical modeling and optimization of optoelectronics and photonics devices.

Arjen Markus is a senior consultant at the Deltares research institute, where he is responsible for several numerical modelling programs, specifically related to water quality. He has contributed to the ACM Fortran Forum and in 2012 he published the book "Modern Fortran in practice".

Emanuele Pagone is Research Fellow in Sustainable Manufacturing Modelling at Cranfield University where he simulates energy and manufacturing systems to support environmentally sustainable multidisciplinary decisions.

Ivan Pribec is a PhD student in the TUM School of Life Sciences at the Technical University of Munich, where he develops and applies lattice Boltzmann methods to explore and optimize chemical engineering unit operations.

Brad Richardson is a consultant and scientific software developer at Archaeologic, Inc. His efforts focus on developer productivity through the development and use of better tools, and the discovery and application of better software design principles.

Harris Snyder is a scientific computing engineer at Structura Biotechnology, where he works on data processing software for cryo-electron microscopy.

John Urban currently provides HPC consultation and support at the Bettis Atomic Power Laboratory, for the design and development of nuclear power for the U.S. Navy. His career has ranged from Engineering commercial power at Westinghouse to work at Cray, Digital, Compaq, HP, and SGI.

Jérémie Vandenplas is a researcher in Animal Breeding and Genomics at Wageningen University & Research, where his interests are quantitative genetics and genomic prediction related to livestock.



https://dspace.org/

The state of Fortran

Kedward, Laurence; Aradi, Balint; Certik, Ondrej; Curcic, Milan; Ehlert, Sebastian ...

2022-03-16
IEEE
https://doi.org/10.1109/MCSE.2022.3159862
Kedward L, Aradi B, Certik O, et al., (2022) The state of Fortran. Computing in Science and Engineering, Volume 24, Number 2, March-April 2022, pp. 63-72

Downloaded from a DSpace repository. https://dspace.org/