Status of gfortran, the gcc fortran compiler

Paul Thomas for the gfortran contributors[1]
pault@gcc.gnu.org

[1] https://gcc.gnu.org/onlinedocs/gcc/Contributors.html identifies all the gfortran contributors. Authors of gfortran ChangeLog entries in 2025 are:

H Anlauf, P-A Arras , R Biener, J Brown, T Burnus, J Delisle, D Kochergin, M Jambor, J Jelinek, S Kargl, T Koenig, S Loosemore, H Lu, Y Ma, D Malcolm, J Melcr, M Mohite, M Morin, A Pinski, D Rouson, T Schwinge, P Thomas, T Trnka, A Vehreschild, M Vollweiler, M Wielaard and K Yeung

Credit should also be given to the bug reporters (especially those that prepare small reproducers and post to gcc Bugzilla ☺)

Introduction

- The purpose of the GNU Fortran (GFortran) project is to develop the Fortran compiler front end and run-time libraries for GCC, the GNU Compiler Collection.

- The gfortran is compliant with Fortran 95 and includes legacy F77 support.
- Most F2003 and F2008 plus an increasing number of F2018 and F2023 features are implemented.
- Of particular importance are the support for OMP and coarrays to support parallel computation.
- See the gfortran wiki for more details: https://gcc.gnu.org/wiki/Gfortran.

- The presentation will describe the status of implementation of the new or modified features of each of the newer standards, parallel processing and single processor performance.

The gfortran “front end”

- The gfortran “front end” is, in principle, written in C++. However, only small snippets of C++ have crept in and, for the main part, it is written in Kerighan and Ritchie C.

- The front end has three main stages:

1. A single pass parser that converts the fortran source into an intermediate representation;
2. A resolution stage, which includes some optimisation passes. The resulting intermediate representation code can be exposed using the compiler option –fdump-parse-tree; and
3. A translation stage, where the intermediate representation is converted to TREE-SSA use by the gcc infrastructure to produce executable code. The translation output can be exposed with the compiler option –fdump-tree-original. The output appears in a file with the suffix *.original and looks distinctly C-like.

- While the front end code is syntactically straightforward, the single pass compiler, legacy support and complexity of the fortran language from F2003 onwards have caused significant bloat.

Standards compliance

- The following slides are based on the tables to be found in the gfortran wiki.
- The wiki entries are tabulated as in the sections/subsections of John Reid's "The new features of Fortran 20xx".
- I have concentrated on those areas in where considerable progress has been made in the development version gfortran-16.0.0 and should appear in the 2026 release.
- Additionally conditional expressions and unsigned integers (-funsigned) have been implemented.

F2003 compliance

ISO TRs	
15580: IEEE Arithmetic	**Yes** (since 5.0)
15581: Allocatable Enhancements	**Yes** (since 4.2)
Data enhancements and object orientation	
Parameterized derived types	**Yes** (since 8.0, 2017-09-09), bugs PR82173
Procedure pointers	**Yes** (partial since 4.4, components since 4.5)
Finalization	**Yes** (since 4.9: PR37336 3 bugs left)
Procedures bound by name to a type	**Yes** (partial since 4.4, 2008-08-31)
The PASS attribute	**Yes** (since 4.5, 2009-07-25)
Procedures bound to a type as operators	**Yes** (since 4.5, 2009-08-27)
Type extension	**Yes** (since 4.4, 2008-07-29)
Overriding a type-bound procedure	**Yes** (since 4.4, 2008-08-28)
Enumerations	**Yes** (since 4.1)
ASSOCIATE construct	**Yes** (since 4.6: PR87477 3 bugs left)
Polymorphic entities	**Yes** (partial since 4.5, arrays since 4.7, unlimited since 4.8)
SELECT TYPE construct	**Yes** (since 4.5, 2009-11-30)
Deferred bindings and abstract types	**Yes** (since 4.4, 2008-09-02; deferred binding since 2009-03-29)
Allocatable scalars	**Yes** (since 4.5, 2009-09-30)
Allocatable character length	**Yes** (since 4.6, as components since 4.9, bugs: PR68241)

Your's truly made a mess of the original PDT implementation ☹

2025 PDT campaign has reduced number of (known) bugs by 24, of which 11 were found during the campaign. 11 to go…..

Work on the associate construct was funded by an STF grant (sovereign.tech/de/tech/gfortran)

Otherwise F2003 coverage is in good shape.

F2018 compliance (ii)

Additional parallel features in Fortran (TS 18508)	
Teams	**Yes** (partial 6.0 - complete 16.0)
Image failure	**Yes** (in 16.0)
Form team statement	**Yes** (in 16.0)
Change team construct	**Yes** (in 16.0)
Coarrays allocated in teams	**Yes** (in 16.0)
Critical construct	**Yes** (in 16.0)
Lock and unlock statements	**Yes** (in 16.0)
Events	**Yes** (since 6.0)
Sync team statement	**Yes** (in 16.0)
Image selectors	**Yes** (in 16.0)
Procedure calls and teams	No
Intrinsic functions get_team and team_number	**Yes** (in 16.0)
Intrinsic function image_index	**Yes** (in 16.0)
Intrinsic function num_images	**Yes** (in 16.0)
Intrinsic function this_image	**Yes** (in 16.0)
Intrinsic function move_alloc	**Yes** (in 16.0)
Fail image statement	**Yes** (since 7.0)
Detecting failed and stopped images	**Yes** (since 7.0)
Collective subroutines	**Yes** (since 5.0)
New and enhanced atomic subroutines	**Yes** (since 5.0)
Failed images and stat= specifiers	**Yes** (in 16.0)

The coarray features added in the current development branch, 16.0.0, were developed by Andre Vehreschild.

This work was funded by an STF grant (sovereign.tech/de/tech/gfortran)

Of the rest of the features introduced in F2018, “Further interoperability of Fortran with C (TS 29113) is in good shape” and the wiki entries for the other features appear to be in need of updating.

Coarrays

- GNU Fortran currently supports three **coarray** modes, which can be selected via the **-fcoarray=** flag:
  - **none**: The default, which prints an error when a coarray construct is encountered
  - **single**: Optimized version for a single image, which allows for fast serial programs
  - **lib**: A communication-library-based coarray version; either MPI or GASNet.
  - *A shared-memory version is under development.*

- See https://gcc.gnu.org/wiki/CoarrayLib for details.

Open MP and GCC

- GNU Fortran implements all of the OpenMP Application Program Interface v4.5, and many features from later versions of the OpenMP specification. *See OpenMP Implementation Status in GNU Offloading and Multi Processing Runtime Library*, for more details about currently supported OpenMP features.

- To enable the processing of the OpenMP directive !$omp in free-form source code; the c$omp, *$omp and !$omp directives in fixed form; the !$ conditional compilation sentinels in free form; and the c$, *$ and !$ sentinels in fixed form, *gfortran needs to be invoked with the -fopenmp option*. This option also arranges for automatic linking of the OpenMP runtime library. See GNU Offloading and Multi Processing Runtime Library.

- See: https://gcc.gnu.org/onlinedocs/gfortran/OpenMP.html for details.

Polyhedron Fortran Testsuite https://fortran.uk/

Linux – Runtime Benchmarks 64-Bit Fortran								
	Absoft 19.0	Absoft(AP) 19.0	gfortran 7.40	Intel 2019.5	Intel(AP) 2019.5	NAG 6.2	PGI 19.4	PGI(AP) 19.4
ac	3.80	3.76	13.99	3.32	3.47	14.05	5.50	5.56
aermod	6.05	6.21	5.12	5.61	6.01	7.85(3	—(6	—(6
air	1.73	0.75	2.45	1.71	1.11	1.86	1.14	1.35
capaci	10.75	10.92	11.72(7	9.67	10.92	11.94(3	9.55(4,5	9.86(4,5
channe	53.13	46.27	49.32(7	45.79	48.06	69.21	50.70(4,5	50.05(4,5
doduc	8.98	8.99	7.87	5.45	5.38	10.03	7.77(4,5	9.33(4,5
fatigu	37.81	38.57	44.16	31.70	31.63	87.94	63.94	64.77
gas_dy	32.30	29.44	48.59	45.27	44.82	54.97	31.54	32.80
induct	19.93	12.67	24.13	18.87	19.76	63.66	30.59	30.58
linpk	3.56	3.58	2.82	2.40	2.47	2.90	2.50	4.35
mdbx	5.40	4.61	4.51	3.51	2.22	5.14	3.92	4.24
mp_pro	81.44	27.76	160.46	32.57	6.08	158.88	32.43	32.44
nf	5.57	5.50	4.13	3.57	3.67(8	5.70	4.74	4.80
protei	14.18	13.79	13.76	14.30	14.35	13.22	13.63	14.51
rnflow	9.27	9.19	18.35	7.09	5.02	18.82	16.37	11.57
test_f	36.02	27.20	47.16	12.37(8	17.10(8	49.86	23.07	22.48
tfft2	31.73	25.65	23.26	26.04	25.94	30.25	29.80	24.56
Geometric Mean	12.78	10.57	14.98	9.76	8.56	18.20	12.15	12.60

No recent, published results that I could find.

Did some runs with

- gfortran 15.2.1
- ifx 2025.3.0
- flang 21.1.4

The purpose of these runs was not to compare the compilers but to uncover relative changes in gfortran since version 7.4.

Polyhedron fortran testsuite runs 31/10/25

	gfortran 15.2.1	ifx 2025.3.0	flang 21.1.4
ac	4.8	3.7	4.3
aermod	2.6	3.3	3.0
air	1.2	1.2	1.4
capaci	5.0	6.6[1]	7.6
channe	38.9	38.8	38.6
doduc	4.3	4.2	4.3
fatigu	31.4	34.8	30.0
gas_dy	31.5	19.2	Seg fault[2]
induct	16.4	12.1	22.6
linpk	2.0	2.0	1.9
mdbx	3.5	3.2	3.5
mp_pro	56.7	17.3	29.4
nf	4.0	4.3[1]	4.0
protei	8.3	8.4	7.8
rnflow	11.5	7.9	Seg fault[3]
test_f	23.8	18	15.9
tft2	14.2	18.3	17.9
Geo mean	9.7	na	na

- Tests run on Core I9-12900HKx20 with 32Gbytes
- Fedora Linux 43 (Work Station Edition)
- All runs with –O3 and no fast math
- Algorithm in rnflow.f90(function genui) replaced by “call random_number (genui)”.

[1] Segfault in libcp required use of –heap-arrays 4000000 for run.
[2] Incompatible dummy data object shape in subroutine ‘KEEL’ causes segfault.
[3] Segfault. gdb reveals location as line 3430 and “error reading variable: value requires 524288 bytes which is more than max-value-size”.

Final Remarks

The gfortran maintainers/developers tend either to be fortran users, gcc maintainers or working under contract, typically funded by governmental grants.

Many contributors from previous years not mentioned here. The same is also true of financial contributions.

Even though the volunteer, fortran users are in the minority in the contributor list, their presence over the years has given rise to an ethos that appears to be discouraging of financial support. This is far from being the case, although the terms of the GPL have to be heeded.

If you hit a problem with gfortran, please feel free to try to fix it yourself and contribute to the gfortran mailing list. Given that this is a minority taste ☺, please submit problem reports to Bugzilla, preferably with a reduced testcase.