Benefits of continuing Fortran standardisation survey: final report

Anton Shterenlikht
Standards Officer, BCS Fortran Specialist Group

14th January 2019

1 Introduction

This survey has been developed by the committee of the BCS Fortran Group to quantify the value of modern Fortran standards to organisations and individuals. The Fortran Group wanted to know how newer Fortran standards have increased the quality of users' code, cut development costs, increased portability or performance, or whether users could attach any monetary value to the benefits enabled by modern Fortran standards.

The Fortran language has been steadily developing since its origins in 1957. Many people have been working on revising the Fortran specification, resulting in Fortran 77, 90, 95, 2003, 2008 and 2018 standards. This survey was designed to find out exactly what benefits newer Fortran standards bring to the community.

The results of the survey will help the Group justify continuing involvement in Fortran standardisation efforts. The results of the survey will also be shared with the ISO Fortran standardisation committee.

The survey opened 30-JUN-2018 and closed on 3-JAN-2019. 427 respondents participated in the survey. All questions were optional, hence the number of responses varies for different questions. The percentages for each question were calculated based on the number of responses for that particular question.

For fields where the respondents could enter any text, the responses are given verbatim, one response per paragraph. Multiple identical responses in such fields are indicated with numbers in brackets after such responses.

Efforts have been made to preserve the original formatting in the longer responses.

These are raw results as presented by Google Forms. No analysis of the results has been done.

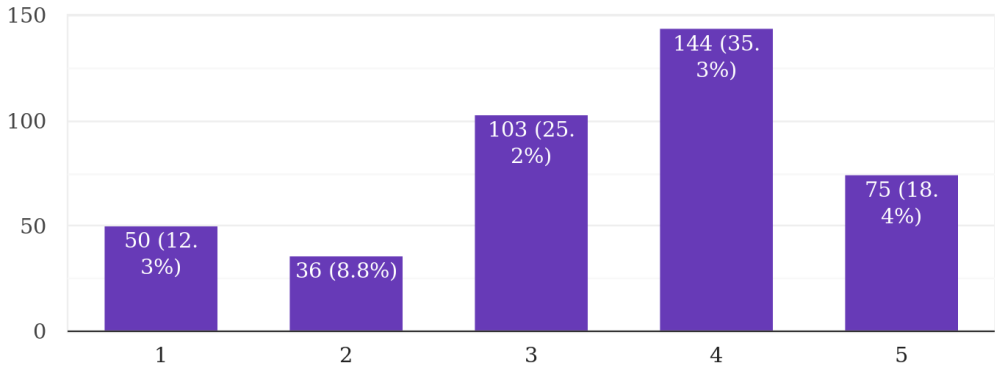
2 Have newer Fortran standards brought you any of the following benefits?

The possible answers to all questions in this section are from 1 to 5, where 1 means "No cost saving", and 5 means "Huge cost saving":

	1	2	3	4	5	
No cost saving	○	○	○	○	○	Huge cost saving

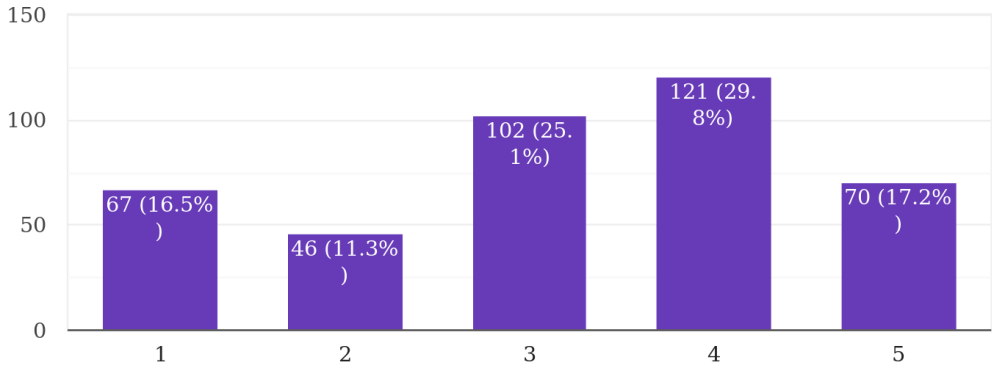
Cut development costs, e.g. via more powerful language features

408 responses



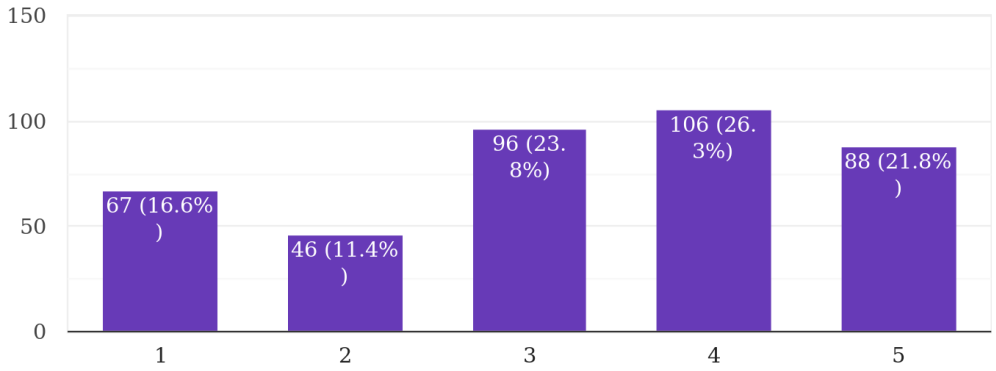
Cut deployment costs, e.g. via improved portability

406 responses



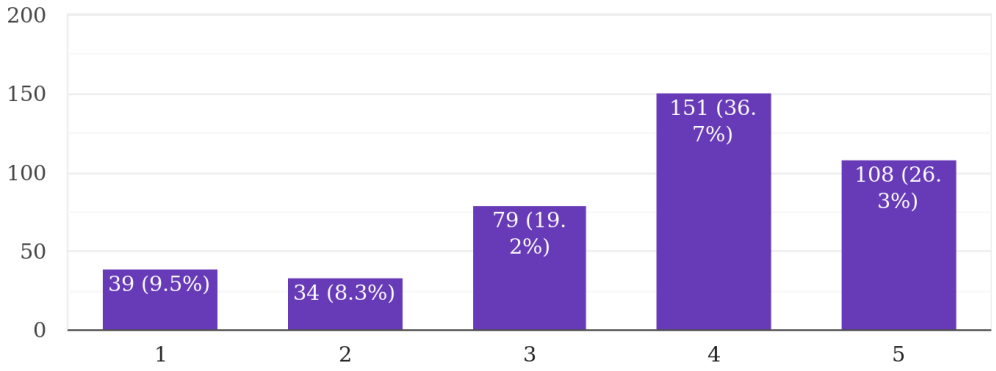
Ability to target new architectures, e.g. parallel computers and HPC

403 responses



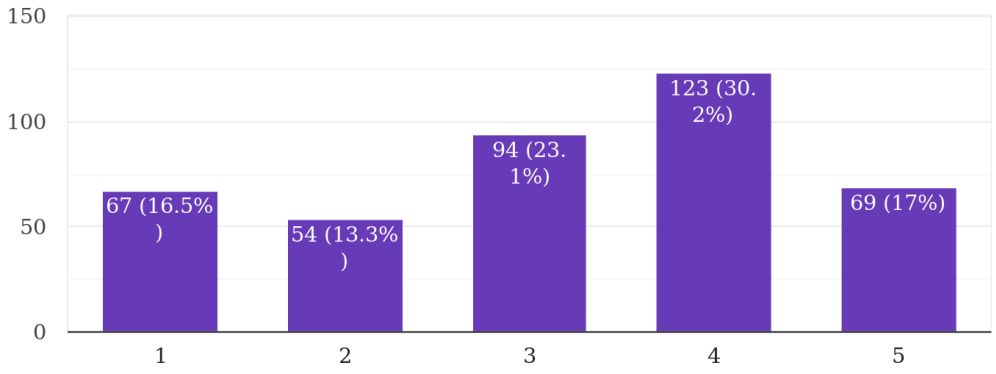
Cut debugging costs, e.g. via more stringent rules discouraging "sloppy' code

411 responses



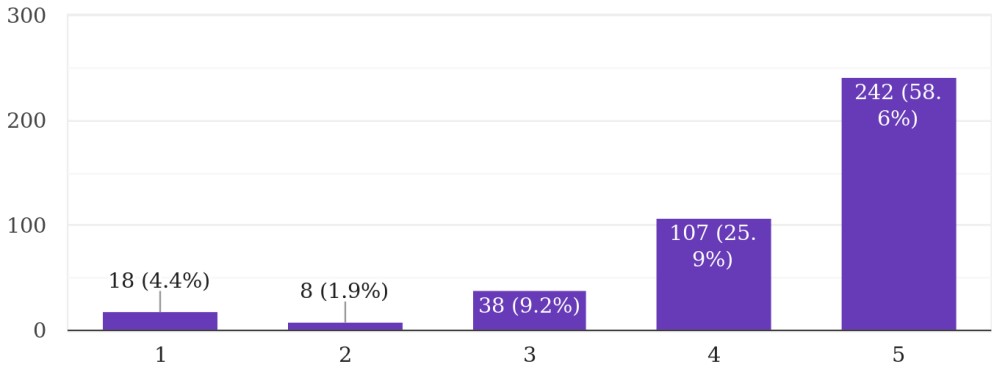
Cuts maintenance costs, e.g. with object oriented features

407 responses



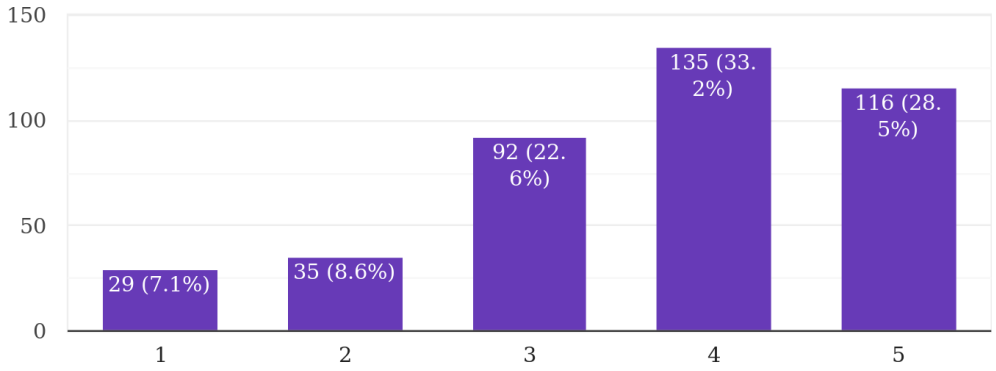
Better code expressiveness, e.g. via free format, long variable names, whole array operations, etc.

413 responses



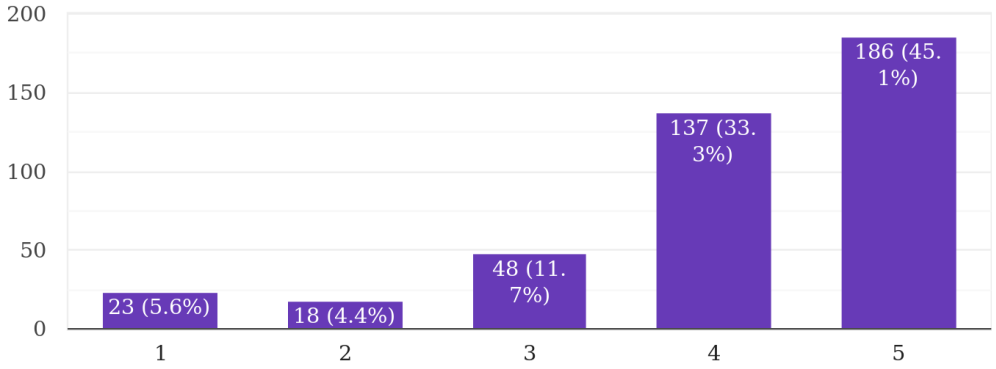
Better optimised code, e.g. more structured code, easier for compilers to optimise

407 responses



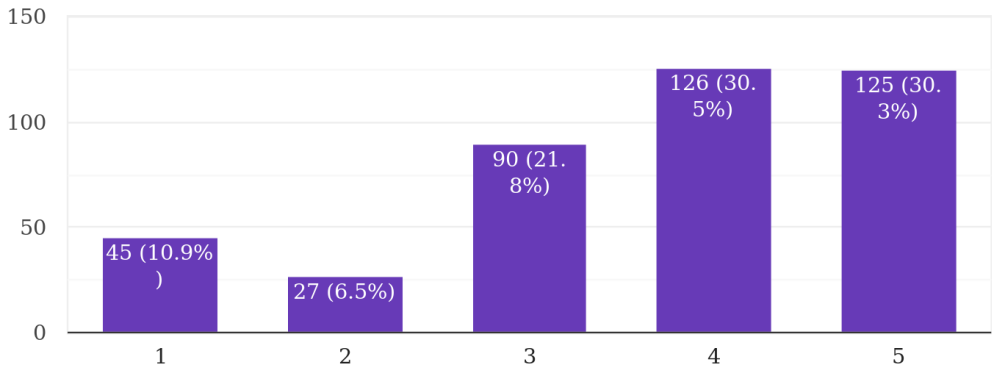
Improved code modularity

412 responses



Better interoperability with other programming languages

413 responses



Other benefits not mentioned above

51 responses

Improved error handling; improved support for pointers (allowing more flexible data structures); improved vectorisation hints (do concurrent)
support generic programming, at least support generic containers
Removal of archaic 'gotchas' - level 4
The mentioned huge savings cost are however offset by compiler bugs and workarounds required to make things actually work.
Allocatable arrays, variable length strings, etc. significantly reduce development time and enhance code reuse through simpler interfaces.
The benefits of improved Fortran standards can't be equated directly to financial cost/benefit in my organisation, so the negative answers above do not reflect the true benefit. Also I maintain legacy code and rarely compose code from scratch so I can't take advantage of the most modern features. Benefits do include improved interoperability with the operating system, and I expect to write Fortran code to interact with SQL databases in due course.
The coarray model for parallel programming / HPC without MPI/OpenMP.

Easier to program, in line with modern programming practice.

Ability to return to code months or years later and still understand the syntax and organization -- iff some OO features are not used (OO seriously degrades "returnability").

But for Fortran 2003 and later revisions, Fortran will not be used in my organization.

It is useful to have GPU coding features in some Fortran versions, would be great to have this as standard, including for non-NVIDIA GPUs.

It's not a benefit, however the standard did pointers all wrong.

Improved code correctness, it can be very difficult if not impossible to determine correctness of old fortran code for different datatypes etc.

Need compilers to keep up with standards. Often a disconnect between debuggers, compilers and fortran standard making development difficult without substantial costs for commercial tools or workarounds when musing multiple machines (i.e there is still poor portability when debugging MPI codes that require use of specific machines).

Ability to target GPUs with CUDA Fortran and automatic kernel generation

Small improvements such as open(newunit=...), unlimited format descriptor etc. make life a lot easier!

F08/18 could possibly allow for development of new kinds of parallel algorithms

Fortran Coarray is impressive, and I personally believe its development should continue.

Backward compatability gets a score of 4. I can still use math software libraries I developed starting in 1974.

Please keep f77 alive and well!

I use legacy fortran code, and so have not experienced the new standards.

I would like to be able to say that all of the new features implemented by the recent standards have enabled me to make a quantuum leap in my ability to write to write more modular, more efficient, and more maintainable code but I can't because off the lack of a compiler that everyone has unrestricted access to that supports all of the current standard (Fortran 2008) and is bulletproof. I can't tell how useful a new feature is if I don't have access to a compiler that supports it. This is the biggest issue facing Fortran today. Granted writing a compiler is a very complicated endeavor but that still is no excuse for the lag between when a standard is released and when the new features are available in a compiler everyone has access to. I'm afraid we have reached the point where the language will die not because its inferior to other languages but because the people in charge of defining the language have no clue how to lead. The standard committee is too inbred with compiler developers who only see the language from the inside out and lacking in users who know what features they need for their particular application space. While things like coarrays etc are great ideas, the average user would have preferred having something like the C++ STL so we don't have to write our own ADTs and containers for things like lists, maps, etc. that are almost mandatory for modern programming in areas like FEM and CFD. As of today the only two things I can think of that will save Fortran is for Intel to follow NVIDIA/Portland Groups example and release a community edition (Ie free) version of their compiler and for the NVIDIA backed flang project to supplant gfortran as the "open source" compiler. Just my 2 cents

Flexibility of data sets due to allocated arrays

The built-in integer and floating-point arithmetic system intrinsics.

Protected values for input constants. Definition of INTENT IN/OUT.
Language defined features that previously relied on external libraries are a giant bonus.
Ability to transfer coding paradigms from other languages (python) to compiled programs
The standards are quite good actually. The problem is the lack of implementation in many (especially commercial) compilers. Especially as a scientist in an HPC environment, where it isn't easy to change compilers, this means that we are still stuck to F90 + selected F2003 features to ensure portability.
None!
Biggest improvement was the expressiveness
Actually not a benefit, but a real failure in the standardization. There is no standard for the Fortran modules file format. This is terrible. One has to compile Fortran libraries with all sorts of compilers to be able to link them to main code compiled with the corresponding compilers. The Fortran standards committee should have adrressed this horribly lax policy a long time ago. Long overdue.
Some of the above benefits might have been in the standard but took another ten years to be widely available with compilers.

Running newer code on older clusters is very tricky subject, good example is vasp544
Improved I/O error checking improves execution robustness.
Using FORTRAN 77 with FORTRAN 2008 is still a nightmare: there is no function for the leading dimensions of an array.
Large backward compatibility
powerful build system with first-class Fortran support
Allows adding new capabilities while still keeping legacy code working.
We use f90 since it has dynamic array allocation; otherwise we are basically writing f77. We avoid modules and all other fortran features since then. OpenMP is fine for our parallelization for such codes.
Better-performing array syntax in compilers, which has improved over time
Convinced tool writers that it is not a dead language.

Some Fortran features make it ideal for some kinds of numerical computing. Newer features mostly seem to enable interoperability with similar new language features, mostly in C, C++. Co-array Fortran still seems not to have widespread support - unclear if it provides a low enough level or if other language designs will supplant it as a fourth generation programming language. Cuda Fortran is a very well designed accelerator programming model, would be nice if something based on this ended up in Fortran for general accelerators.
Honestly, I think Fortran should be retired
lack of compiler supports is the main issue.
Just a general remark: I once read a quote from Gauss, if I remember it correctly, I have never been able to find it again, about primes. A contemporary of his complained that making progress with the mathematical laws gouverning prime numbers was difficult because "we lack a good way to notate them". Gauss's answer was that we need notions, not notations. That is true as much for programming languages as it is for mathematics. We need to clearly express our ideas, the notation (syntax) is merely a vehicle. Of course a clumsy notation makes it difficult to clearly express your ideas. In my opinion most of the new features of Fortran make us express our ideas more clearly.
My experience is that many of the newer Fortran features have actually increased cost, as earlier standards (77/90) often used in legacy code were much simpler and more reliable and maintainable.
Less necessity to use C for features not implemented in Standard Fortran.
Allow modern code design to still exploit older libraries
Code integrity is improved with the modern standards.
More attractive code: expressing ideas elegantly More modern look/feel: defending Fortran against "modern" languages
The savings in my case are not so much in cost as in programming time and effort. A benefit not mentioned above is improved clarity of code for programmers new to the code.

3 Please tick any features which you use, or for F2018, are planning to use

In this section the features are sorted by popularity. The percentages are rounded (nearest) to integer values.

3.1 Fortran 95, 408 responses

3.1.1 Pre-set responses

Feature	NUM	%
do/end do loop	383	94
whole array operations	368	90
implicit none	368	90
dynamic memory allocation	367	90
modules	367	90
free form syntax	365	90
array sections	331	81
module procedures	312	77
exit, cycle	310	76
intrinsic procedures for arrays	289	71
optional arguments	285	70
select case	281	69
allocatable components	271	66
pointers	252	62
generic interfaces	242	59
internal and recursive procedures	241	59
cpu_time	222	54
operator overloading	181	44
null	178	44
parametric intrinsic types	152	37
multibyte characters	63	15

3.1.2 Additional responses

1. WHERE and FORALL constructs are important too.
2. the key nature of F90/95: array operations on nested derived types (object-based array programming)
3. Standard date and time functions
4. do concurrent, "accessor" pointer-valued functions, g0 descriptor
5. FORALL

3.2 F2003, 355 responses

3.2.1 Pre-set responses

Feature	NUM	%
interoperability with C	256	72
OS: envars, command line, etc.	205	58
inheritance	184	52
dynamic type allocation	181	51
type extension	176	50
type–bound procedures	172	49
polymorphism	164	46
procedure pointers	158	45
flush	151	43
input_unit, output_unit, error_unit	143	40
IEC 60559 floating point exceptions	136	38
stream IO	108	30
parametrised derived types (PDT)	106	30
deferred type parameters	97	27
explicit type in array constructor	95	27
finalisers	82	23
asynchronous IO	63	18
derived type IO (DTIO)	56	16
control of rounding modes	53	15
volatile	28	8

3.2.2 Additional responses

1. protected attribute, bit manipulation functions
2. ASSOCIATE
3. ASSOCIATE, ABSTRACT types and interfaces, IMPORT, GENERIC binding, almost all of Fortran 2003 is critical to modern code
4. get_command, command_argument_count, move_alloc
5. increased length for names to 63 characters
6. Interoperability with C should be mentioned at least twice!!

3.3 F2008, 310 responses

3.3.1 Pre-set responses

Feature	NUM	%
int8, int16, int32, int64, real32...	189	56
64-bit integer	161	48
Bessel and err. func, e.g. BESSEL_J0	127	38
submodules	124	37
do concurrent	121	36
execute_command_line	120	36
c_size_of	110	33
contiguous	108	32
coarrays + coarray intrinsics	106	31
findloc - array searching	106	31
newunit	104	31
bit manipulation func. bitwise comp...	102	30
compiler_version, compiler_options	91	27
block construct	91	27
new complex intrinsics: ACOS, ACOSH...	86	25
storage_size	76	23
HYPOT, NORM2 for 2-norms	75	22
%re, %im shorthands for real and im...	75	22
more complex intrinsics	71	21
initial pointer association	71	21
impure elemental procedures	54	16
atomics	53	16
critical	52	15
locks	48	14
max array rank of 15	41	12

3.3.2 Additional responses

1. intrinsic assignment for class(*) variables
2. Implied shape array, allocatable components of a recursive type (for stack type of data structures), kind of a DO CONCURRENT index, polymorphic assignment, pointer functions, MOLD in ALLOCATE, G0 edit descriptor, unlimited format item, recursive I/O, on and on with so-called miscellaneous enhancements per Modern Fortran Explained.
3. Mold=x in allocate statement
4. Might use coarrays in the near future.
5. error stop in pure procedures. note that I avoid inheritance - I only use type extension to classify my procedures and their arguments more precisely rather than relying on a variable name alone.
6. error stop
7. (The compilers I can use are only beginning to support coarrays in a convenient way. Our programs are not yet taking advantage of them)

8. defining functions in subroutine

9. Much of 2008 is still a little bit too new for released code

3.4 Fortran 2018 (previously known as 2015), 180 responses

3.4.1 Pre-set responses

assumed rank: select rank	83	46%
assumed type	77	43%
improved IEEE floating point support...	75	42%
ISO_Fortran_binding.h, CFI_establish...	73	41%
C descriptor for assumed shape dummy	66	37%
collectives: co_broadcast, co_max...	48	27%
new atomics: atomic_add, atomic_and...	37	21%
events: post, wait, event_query	34	19%
teams: form team, change team/end te...	33	18%
image failure: failed_images, stoppe...	24	13%

3.4.2 Additional responses

1. All of this is too recent to trust for deployment (2)

2. Not used 2018 Fortran but would like to try in next project.

3. Most of the items in "Removal of deciencies and discrepancies" per John Reid's "What's New in Fortran 2018", especially enhancements to ERROR STOP.

4. planning to use parallel-programming support, required features not yet determined

5. error stop in pure procedures

6. exceptions

7. (Very limited support in the compilers I use ...)

8. 2018 is much too new to consider using in released code

4 Future of the Fortran Language

4.1 If you think Fortran is lacking particular features which would help you, please detail them here, 150 responses

Templates
generic programming
None
Protected components of derived types (modifiable in the module where they are defined, viewable but not modifiable elsewhere), for the reasons explained in N2147, page 12, third bullet point. Exception handling.
An in-place version of "reshape", to allow an array to be addressed as if it had a different shape, but without causing a memory copy. At the moment I do this in a nasty way by casting to a C pointer and then back to a differently-shaped Fortran pointer. A way to pass the lower and upper bounds of an array in and out of subroutines (without having to have the array "allocatable"). Native support for heterogeneous computing, particularly with regard to using alternative RAM (e.g. HBM or SSD) and Accelerators; e.g. CUDA Fortran has extra attributes to specify whether arrays live on the "host" or the "device".
matrix operations
generic programming, generic containers
Some symbolic maths tools/library would be nice, or a latex type print function for existing expressions.
Packages (as in Java) to allow namespace management and "friend" types; type-bound procedures for arrays of defined types (e.g. as in python/numpy).
multiple inheritance would avoid some present workarounds.
type initialisation procedure (similar to finalisation procedure)
Unsigned integers
Better run-time error trapping
A native or embedded MPI support might be a step forward for Fortran. How about automatic shared CPU/GPU memory blocks for fast coding and optimized GPU support? Like viewing memory blocks at CPU and GPU with same arrays.
- inline functions as part of the standard (even if compiled in other third party shared library). This also goes for derived-types to implement getter and setter functions without penalty.

- chaining of functions as is possible in most other languages:
e.g.
call A%get_B()%execute()
where "get_B()" is a member function of derived_type instance A that returns a derived_type instance B, and we don't
need to add boiler plate including the type of B. We also don't need to create a "copy" of B to do this. So functions should be able to return a value by reference or pointer, as is possible in C++.

- operator() overloading of derived type (relying on inlining mentioned above)
e.g.
type(LookupTable) :: lookup_table
integer :: i
...
i = lookup_table(row, col)
the operator(row,col) may be an inline statement like:
return array(offset(col) + row)

- compiler independent module standard.

- namespaces to avoid module name clashes.

Generic containers

string handling

Expanded generic programming / polymorphism: currently there is no way to write e.g. a sort function which takes an array of unknown type and a comparison function and returns the sorted list. Similarly, if a parent type has a+b overloaded (a,b and a+b are all of type parent), it would be very useful to be able to have a child type which extend parent inherit a+b such that a,b and a+b are all of type child.

It would be useful if all of the intrinsic features of Fortran could be used with classes. e.g. a(i:j) is a very standard operation on intrinsic arrays, but there is no way of writing an array-like type which can be sliced using the a(i:j) syntax.

Lambda functions

Generic programming, structured exceptions handling, unsignef integers (all planned for Fortran 202X

Better string handling. A mature collections library would be good (or bindings to C++ collections would be fine)

exceptions and templates

Handling of strings is poor. Built-in dictionaries would be handy.

unsigned integer
1) Generics e.g., ability to compactly compose subprograms that can operate on any type or a set of stated types; ability to efficiently design containers for data of any type or a set of stated types such as lists, stacks, maps, queues, etc. 2) Scoped enumerations (reference: C++ or Microsoft .NET), a must for modern code in order to advance beyond the ordinary named constants; 3) Derived type and OO enhancements including SEALED (NON_OVERRIDABLE) classes, MOVE semantics, clear concept of namespaces with a third option in mind other than PRIVATE/PUBLIC attributes of type components and bound procedures so that everything useful with respect to extension types.
GPU programming as standard feature
templates, put-with-notify (to a different image)
Facilities for proper generic programming, like parametric types in Julia, or templates in C++.
Generic programming (templates) and namespaces
The software I develop is limited to Fortran 95 for portability reasons; new standards should not get too far in front of compilers.
bit type

templates
GPU acceleration: the Fortran language is outdated with respect to C/C++, that with OpenCL can be used with very little effort on heterogeneous hardware.
Fortran's approach to generics requires significant code rewriting for many cases in which the structure of the code is identical e.g. writing an interface explicitly for all numerical types when constructing an I/O routine for many dimensional matrices. This is avoided in C++ for example through templating, and in other languages through actual generics. Fortran still lags behind in requiring the coder to be far too explicit about a great deal of type information. Try, for example, writing a generic vector type in fortran, that can take arbitrary objects and compare that with C++ or even C. It is very difficult to write safe, correct code without repeating it many times or writing your own (error prone) code-generation program.
Cross platform support for debugging. I.e. windows and linux, command line debuggers can be cumbersome and are not intuitive.
Something like C++ template feature would have been nice to make the code more compact
Arrays of pointers

An efficient way to work with outer products of small (say length 3) vectors
More consistent implementation of compiler support.
A simpler way to approach generic programming
execute_command_line able to return a non-error type character message
generic programming support (templates), exception handling (try-catch), native heterogeneous arrays (e.g., arrays of character strings w/ non-uniform length)
F77 was easy to each and could/CAN program anything, but lack of dynamic memory required disc-based memory algorithms.
Improved string handling, *generic programming* like e.g. Java. Useful datastructures, e.g. linked lists and hash maps, with implementations of common operations on these. Coming from Python, Java and C++, I really wish there were a standard library.
templates, better exception handling

Fortran, apart from legacy maintenance, is becoming irrelevant to me. I want to use OO features, but it is so clunky and verbose in Fortran relative to C++. The big missing features are a) anything equivalent to C++ templates. Writing a type-generic module often means explicitly coding for each possible type. b) defined binary implementations, as in C, for things like binary files, subroutine arguments, etc. Yes, I know that iso_c_binding helps with this, but still... c) C-style pointer, and C++-style reference type attributes d) C++-style const (although parameter is nearly the same apart from (e)) and auto e) remove the insanity of having to define types before any executable statements. Yes, I know about block, but frustratingly we could not use it for many years because Intel were too lazy to implement quickly. f) somehow persuade compiler vendors to actually implement full standards quickly
Support for default argument values for optional arguments
- Better ways of declaring generic procedures - A better way of setting default values for optional arguments - Reduced precision reals

Generic programming, as widely noted. It would be good if this allowed us to not just avoid 'trivial rewrites' (e.g. like we do with #include) , but also to write 'high efficiency' code as well. For instance it is often said that the C++ 'sort' can be faster than qsort in C, because the former can inline the comparison operator. I suppose similar optimizations should be available e.g. for numerical integration libraries. I hope this kind of optimization will be straightforward in fortran's (future) generic programming.

Furthermore, recently we see people writing template based libraries in C++ that seem like they might be able to solve the 'hardward portability' issue -- i.e. writing code that runs and is reasonably efficient on both GPU and CPU. This kind of thing seems likely to make inroads to the climate/ocean communities. I'm no expert, but it looks to be facilitated by the strong generic programming capabilities of c++. I'd like it if future variants of fortran have this degree of flexibility.

No. Fortran is big enough and risk language bloat. Already problems with compilers taking long time to fully implement all features of F2008 even though 10 years old. Don't break it.

The main issues I see is the lack of genetics and the lack of interfaces (as in Java).

Subscripting on the fly an array function result

template meta-programming

1. Ability to read strings from files without declaring a "max line length", directory into allocatable character variables. "character(len=:), allocatable :: line" then "read(fileunit, '(A)')" line.
2. Ability to use allocatable and/or pointer arrays and that the "contiguous" attribute enables SIMD-type optimization on the related array operations.

Generics, exceptions

proper generic programming (or templates); unsigned integers; standardized ABI; exceptions; built-in unit testing; ready to use standard library with generic tree, map, list, etc

Generic programming

addition of some sort of template meta-programming would be great

Checking and (explicitly) converting units of measurement. Exception handling. Coroutines and iterators. Generic programming. Support for containers. I have sixty pages of small things collected from 600 colleagues during the last half century.

Plotting

As I mentioned previously, the biggest thing missing from Fortran is something like the STL with predefined containers, iterators etc for standard ADTs. Based on my experience looking at a lot of C++ implementations of FEM and CFD codes, the STL is used more than the full blown templating capability (and full blown OOP). My preference would be the ADTs implemented as intrinsic types with associated methods

I would like to see the feature implemented in F08 where the TYPE statement can be used to define intrinsic types (ie TYPE(Real(REAL64)) etc.) be modfied to define the type just by a KIND parameter (ie. TYPE(KIND=REAL64)). This would make parameterized types infinitly more useable. ie. we could then do something like.

```
Type :: genericArrays(akind, bkind, blen)

Integer, kind :: akind, bkind
Integer, len :: blen
Type(KIND=akind), ALLOCATABLE :: A(:,:)
Type(KIND=bkind) :: B(blen)

End Type
```

and makes PDTs closer to templates.

There are a couple of cosmetic changes I would like to make.
The first is to make the CALL keyword optional for referencing subroutines.
This would bring Fortran inline with other langauges for referencing void
procedures. The second thing I would like to see is the restriction of numeric only statement labels removed and fully alphanumeric labels allowed. I
think this would increase the readability of code that makes extensive use of
labels for format statements and GO TO (which does have its uses for
jumping to an error controller etc.)

ie Write(8, array1D) A

array1D: Format(10F10.4)

Obviously too late for this but I would like to be able to use a POLYMORPHIC attribute on a TYPE definition to define a polymorphic dummy argument or derived type component instead of CLASS (ie. Type, POLYMORPHIC). The current use of CLASS is confusing to people coming from other languages. Frankly, I would have never allowed the word to appear in the language at least in its current form.
Finally, I would like to see the use of assume type and assumed rank dummy arguments expanded beyound C-interoperability.

converting strings to variable names?
inherent memory alignment?

Support of half-precision (FP16) - including intrinsics

better link and support of graphics

More access to system level operations

Parsing the compiled .o and .mod files somehow to see precisely what changes were applied upon compilation. Not sure if this is possible. Also, I would appreciate built-in facilities to parse text formats, such as XML or JSON, that would generate the appropriate derived data type on run time. There ought to be more functionality to inquire into the allocation of variables usage at run time, say call display_allocated_memory(). This would print out to the terminal each variable, its allocated rank and sizes, and its memory usage. These memory values could even be returned as optional outputs, or could be "hidden type-bound procedures" like %re or %im on complex values (not sure the proper nomenclature for those features). Anyway, something like a %memory_alloc call that would return a value, say 10e5, whose units are bytes. Just a pipe dream ^^. And one more thing: if assignment is made to a variable of class foo from a variable of type foo, the instance of class foo obtains all of the components and methods of the instance of type foo. When I try this in gfortran, I get errors that I believe exist because it may be ambiguous if the class is extended; but if the compiler could check to see that no type extends type foo, then class foo ought to be able to be assigned without err. Maybe I'm misunderstanding the language facilities here...

C pointers much more helpful that Fortran pointers

no

Interop with other languages

Not really

free & easily accessed libraries of technical methods (numerical recipes and similar) - this is a major advantage of R, for example, in spite of how slow & obtuse R is

GPU interoperability. Support with new tools such as ML frameworks.

DO syntax consistent with DO CONCURRENT and allowing declaring the loop variable, operators such as +=, f0.x format descriptor, revisiting formats for improved readibility (maybe giving format strings a special format so that editors know how to highlight them, while keeping them internally represented as strings. For example, print "(i0,2x,f20.5)", 10, 12.5 could be changed into print /i0,2x,f20.5/, 10, 12.5 or print format(i0,2x,f20.5), 10, 12.5. Having formats with embedded strings such as "(i0, 1x, 'has value', 1x, i0)" all colored as character literal is hell.), being able to declare variables anywhere in the program, removing the hellish implicit save (integer :: i = 3 has implicitly SAVE attribute if declared in the procedure) that forces one to use two separate lines for declaration and initialization and introduces hard to find bugs, IMPLICIT NONE should be default (it's not that much job to paste one IMPLICIT line into old codes or implement a compiler switch for backwards compatibility), force PURE

attribute on all functions so that the "function side effects and the evaluation order" discussion can finally be closed without constraining compiler's optimizing capability. Here I didn't mention two major features from Fortran survey, that is exceptions and templates, which I agree are the most burning need at the moment, but also very challenging to implement -- so I am afraid it might take a long time before compiler vendors catch up. Syntax improvements are small things that can be easily done in a couple of months time without rewriting most of the compiler's code. I believe that Fortran standard committee should strive to keep language as modern as possible and get rid of old features (so nobody has excuse to use them anymore). Compiler vendors will support anything up to f77 for a long time so the standarization committee should worry about breaking 40 years old codes as this will most likely not happen. Compatibility-breaking changes like removing "implicit save" or forcing "implicit none" can be easily handled by compiler vendors by introducing an adequate switch. I believe the fact that there are new codes developed with 50 year old obsolete syntax is the main reason why Fortran has the bad reputation in the progressive part of the scientific community and is dumped in favor of C++ despite it being less suitable for the job. Fortran's place is to be as easy and trouble-free as Python and as fast as C/C++. I advertise Fortran to my colleagues as "Code clean, compute fast, save time on debugging". Unfortunately, I see that many members of the Fortran community are still too lazy to update their programming style and they want to keep old broken features of the language forever, which clearly restricts introduction of new features and syntax improvements. I hope that the commitee members do not listen to them and soon Fortran will regain its dominance!

parallel IO

Generics, macro language, template

Assignment operators (i.e. i += 1 in lieu of i = i + 1), OS identification without preprocessor directives, ideally another symbol for accessing derived type properties (maybe `->`, maybe support `.` where it's not ambiguous, I'm not sure, but % makes for incredibly ugly code).

I use Matlab a lot, but it is too slow sometimes. I would like Fortran to have a standard library like C++, so I do not have to write every function by myself. Now I am using Julia now.

Run on JVM

Generics.

Better string manipulation, improved generics or templates

Hash or associative arrays): A("name") = "Fred"

Generics, in order to make function more portable.

clear and well-defined C interoperability, routines that make compiler-independent I/O with identical results possible without resorting to C functions, a standardized interface to check which features are available and correctly implemented in the compiler, some form preprocessor that works better than the currently used C preprocessor (multi-line support etc.), polymorphism often does not produce code that's fast enough for production on HPC systems

There are enough languages out there that, if I need to do something Fortran can't easily do, I'll just interface to something else. Fortran's weakness is how much code you have to write yourself as there's no centralised, efficient means to obtain libraries.
c-style const pointers
I have been tasked to redevelop Fortran applications in other languages and to be honest I can't see much use in keeping it as a current language. It creates roadblocks because it is so infrequently used now and therefore difficult to read.
More flexible parametrisation (eg, specifying parameter arrays in a similar manner like DATA statements); now that you have do concurrent and contiguous, perhaps do vectorise to force vdctorisation?
Proper aliases
PL/I like "on <condition>" and "revert <condition>"
Explicit vectorization of arithmetic operations instead of relying on the compiler for deciding whether vectorization might useful or not. Also it would be nice to have sorting intrinsics
Framework
Fortran is a dead language and its use should be banned by an act of Parliament
memory allocation attribute (e.g. pinned memory)
I use f77
Fortran module file format must be standardized. It is still compiler-dependent. This is completely against any concept of code portability and interoperability. This is a long overdue issue never addressed by the Fortran standard committe.
Specification for very widely used but non-standard features. For example, INTEGER*4 is accepted by every compiler we know. The standard should say what it means. STRUCTURE-MAP-UNION is supported by several systems. The standard should comment - how should this be interpreted?
High order functions, functional programming
template
Support from compiler makers.
better object scoping/encapsulation
OTHER (44)

1. Templates
2. Threads. Committee members complain about how OpenMP extends Fortran and yet does nothing to provide a better alternative. Coarrays and DO CONCURRENT are *not* reasonable alternatives to threading.
3. A "short" sleep to yield the process. This is useful for MPI functionality in non-blocking messaging. sleepqq() will do the job for ifort, but microsecond capability with C's usleep( ) to the standard would be handy–millisecond fine too. This is easy to add for any Fortran with a tiny C routine compiled and linked in. But, it has to be added.

4. High level I/O - native interfaces (not subroutines) to HDF5, NetCDF, PNG, etc.
5. Protected attribute in type attributes
6. Fortran needs STL as same as C++
7. Threading support.
8. Overloading the Array-Element Access operator to implement array-like objects, as in C++.
9. better inter-language operation for derived types including pointers and allocatable arrays
10. No sparse matrices
11. Even better and more automatic inteorperability with C. Why some ISO_C_BINDING intrinsic procedures are not PURE?
12. powerful macros or template system as first citizien in Fortran
13. My usage of Fortran relies primarily on "old" feature. I do "modern" CS practices with high-level C++ and don't see any need to do all this stuff in Fortran.
14. I'd like a swap function, e.g. to exchange A and B, A<=>B, or A,B = B,A, or ??
15. unsigned int , Template class , Standard math library
16. don't add features. Keep the language stable and simple.
17. None
18. I think Fortran has too many features. I think it has lost its appeal and core purpose of providing simple, unambiguous encoding of mathematical formulae. By pursuing advanced features that try to compete with modern OO languages such as C++ and not having a large enough community to justify industry investment in high-quality implementations of new standards, Fortran codes lose portability for anything other than features that were part of F95 and, only recently F2003. I would like to see F2021 be a substantial feature reduction release.
19. templates or equivalent (compile time variables)
20. Information on language features is not lacking per se, but information on ability of compilers to optimise code would be helpful - at present Fortran 77 produces very fast codes on some compilers which more advanced language features may hinder on some architectures. It would be good to have more knowledge on this.
21. forget old cards and move to stream orientation requiring end of statement ;
22. Exception handling is the one thing that comes to mind - even if the main reason is that Fortran is often criticised for not having them

23. Templates

24. (1) Allowing data members of a class to be accessible by subclasses but not elsewhere (like 'protected' in Java). (2) Default values for optional arguments.

25. List directed reads should stop at the first error so that next record is defined.

26. Fortran90 compilers introduce bugs with some whole array operations

27. Chained lists, packing of derived types

28. More features to manage memory hierarchy and NUMA, to deal with accelerators (GPU)

29. N/A

30. package management, like maven or npm

31. It would be great if one in an associate context could refer to things defined previously in the same associate context. For instance,

    associate (a => long_array_name, s => size(a)).

32. Standard implementations of some standard data types would be very nive, e.g. linked lists, balanced trees, hash tables

33. While I realize that this is unlikely to ever happen, Fortran should move to case-dependence (to *dramatically* improve interoperability with other languages).

34. easy -to-use pointers (like C). Flexibility in passing arguments of different types to procedures. The issue is not lack of features. The issue is the lack of compiler support and compilers are buggy with new features.

35. A normally-distributed random number generator would be great, though it's the kind of feature that might fundamentally be better off implemented by users.

36. control over inlining of procedures

37. Templates or something similar to simplify the writing of generic procedures. Functional programming features e.g. anonymous functions.

38. Binary io of unsigned or compressed data. Must use c instead.

39. Exceptions for error handling, co-array member of a derived type that doesn't have to be declared as a co-array (for use with abstract distributed data structures where a co-array implementation is just one of several versions), derived types with allocatable components that can be bound to c structs, a string intrinsic with procedures to manipulate strings, an abstract class for numerical data types (integer, real , complex) that have generics for standard numerical operations so I can implement an algorithm once.

40. 1) Dynamic pointer downcast (like dynamic_cast):

```
class(base), intent(inout), target:: object
class(derived), pointer:: pd=>NULL()
pd=>object !NEW: Dynamic cast in one line
if(associated(pd)) then
DO STUFF WITH pd POINTING TO object AS derived TYPE
else
object IS NOT OF derived TYPE, DO NOTHING
endif
```

In other words, to replace clumsy SELECT TYPE in simple cases (single case). That is, above "pd=>object" would be a simple shortcut for clumsy

```
select type(object)
class is(derived)
pd=>object
class default
pd=>NULL()
end select
```

P.S. derived extends(base) here.

2) Simultaneous overload/override: Currently, if the base type has a generic type-bound interface (generic type-bound overload for multiple concrete type-bound procedures), there is no way to have the same-name generic type-bound interface in the derived type which would represent the overload of the overriden type-bound procedures.

Something like:

```
type base
contains
procedure:: p1=>pb1
procedure:: p2=>pb2
generic:: p=>pb1,pb2
end type base

type, extends(base):: derived
contains
procedure:: p1=>pd1
procedure:: p2=>pd2
generic:: p=>pd1,pd2
end type derived
```

Here, pd1 overrides pb1, pd2 overrides pb2, but generic overload p cannot be overriden, although it is properly associated with two overriden procedures.

41. Operations with NaN, more than one return values of function (similar to C++ or Python)
42. Maps, templates as in C++
43. Case sensitivity
44. A robust and flexible pre-processor.

45. Standardized format for .mod files, guaranteeing portability across different compiler versions, and compilers.

46. C++ - FortranOOP mixed programming standardized

47. Implement the concept of either friend classes or protected member variables of C++ in FortranOOP

48. Extend the current support for parameterized data types such that one can also parameterize a data type in terms of a user-defined data type (in line with generic programming in C++, a.k.a. templates)

49. A support library in line with the Standard Template Library (STL) of C++

4.2 What Fortran compiler(s) are you using? And does it (do they) support the standard features you want to use? 371 responses

1. GFortran, Intel

2. Mainly ifort. Supports most features, but performance is not always as it should be.

3. gfortran, intel fortran, cray. GNU is usually slow on the uptake of new fortran features from newer standards (c.f. co-arrays etc)

4. Gfortran, ifort, pgi

5. Intel (most features supported, if not all), CRAY (most features supported, if not all), gfortran (many features supported, but %re and %im missing - really looking forward to this, lack of progress in PR40196 for over 9 years is very disappointing), NAG (mostly used as an additional way to check correctness of code, some features missing), PGI (too many features missing, and this limits our development choices due to our need to support OpenACC through it).

6. Intel, GNU and Portland Group. They are very slow to catch up with the standards, which is particularly frustrating for easy things like "do concurrent" and the "contiguous" attribute.

7. fortran, Intel

8. gfortran, ifort

9. ifort, gfortran, nag - yes

10. GNU Fortran, Intel Fortran. They do not (fully or enough) support generic programming

11. Intel, Flang, GCC, Cray

12. The gfortran compiler

13. gfortran 8; yes for the most part (deferred-length characters not fully implemented).

14. gfortran, ifort

15. gfortran

16. NAg

17. gfortran, good enough

18. Nag (mostly), Intel (mostly), gfortran (lacking)

19. ifort, gfortran

20. gfortran, Intel fortran

21. Intel, Nag, Gfortran, PGI, Oracle, g95

22. Intel yes (eventually)

23. Intel18+ + gcc7+ - support is usually good

24. Various and no

25. Intel Fortran and GNU Fortran Compilers. Up to now they are OK.

26. gfortran, ifort

27. GNU(mostly), PGI(mostly), Intel(mostly), Cray(mostly)

28. All compilers tested have bugs preventing me to use some features that I want to use. Work is required for all compilers to give consistent results. Perhaps the standard is too ambiguous in some cases. In particular in the case of finalisation, every compiler calls final on left-hand-side and right-hand-side at different moments, and a different number of times.

29. ifort, yes

30. gfortran

31. Gnu, Intel, Cray, NAG, Portland. They do not support the latest standards quickly enough

32. Intel Visual Fortran. Most, but not all standard features of interest to me are supported. It is quite possible that as new features are added we would make use of some of them.

33. gfortran and yes

34. GNU Gfortran - yes.

35. Gfortran 5.4.0, Nagfor 6207, ifort 17.0.5

    None of these compilers supports the full (up to 2008) standard, and it is frustrating using trial and error to find the subset of useful features that they do all support. All three claim to support parts of the standard which they actually don't. Among other problems, gfortran incorrectly handles elemental functions on arrays and scalars, e.g. f(array, g(scalar)) fails for any elemental functions f and g; nagfor incorrectly handles implicit re-allocations such as array = [array, scalar], and Ifort refuses valid constructor syntax such as array=[String::] when String is a user-defined type.

36. gfortran, flang, openuh

37. Gfortran

38. Intel

39. Intel (most) , gfortran (for Linux). Intel

40. GNU, Cray, Intel, PGI. Support for F2008 and beyond is poor. I don't need OO features either as there are better languages for that.

41. intel ifort largely but I do use others

42. ifort, gfortran, yes

43. Lahey (old); gfortran

44. intel fortran

45. GNU, Intel

46. gfortran, ifort

47. Intel Fortran - supports ALL of Fortran 2003 plus lots of Fortran 2018.

48. Intel, GCC, PGI

49. Cray (yes), Intel (most), gfortran (most), PGI/NVIDIA (not really)

50. Ifort - yes. Gfortran - mostly.

51. gfortran

52. Gfortran. Yes

53. gfortran, intel

54. gfortran (gcc), ifort, nagfor, Cray

55. GCC, PGI, Intel, Cray mostly support the required features

56. Cray, Intel, PGI and GNU

57. intel, pgi, gfortran. We only use features well-supported by all three.

58. ifort, gfortran

59. GCC, Intel, both OK (though both buggy sometimes)

60. Intel Fortran Compiler and Gnu Fortran Compiler: both support standard features that I want to use, but GFortran seems to have a better support of CAF wrt IFort

61. gfortran (generally quite good, takes a very long time for some features, e.g. only just got submodules)

62. ifort (generally ok for new standards, but its code generation is much more frequently incorrect)

63. nag (very very standards compliant)

64. pgi (lags way behind standards)

65. cray (very hard to test without access to supercomputer infrastructure, but from my understanding it is generally up to date)

66. gfortran, Intel, Cray. Need to have similar features (not partial standard implementation). Debugging large parallel codes need work, genrally end up with print statement debugging.

67. NAG, Intel, gfortran NAG does not support coarrays om more than one image.

68. gfortran, intel

69. gnu, intel

70. intel/gnu yes

71. I use mostly intel fortran for large pieces of code and occasionally gfortran for smaller projects. In our work we've started to use the oop techniques, i.e. F2003 some 6 years ago. Since then ifort has improved massively with regards to the support for F2003 - earlier versions such as v12 were riddled with bugs. When I started using oop essentially only ifort had some of F2003 implemented. I guess gfortran has improved since then but I don't know what the current status is.

72. gfortran 8.1 , not full support of F2008

73. Sun Solaris short of 2003; gfortran has most of what I want to use

74. GNU and Intel

75. ifort, gfortran yes

76. Open Watcom F77. Supports F77 with a lot of F90 additions.

77. PGI (NVIDIA) and Intel, yes for both

78. Intel/gfortran/cray - Some but for a large code features need to be supported by all.

79. ifort, fortran

80. Ifort, gfortran, nagfor. yes.

81. NAG for diagnosis and testing, gfortran and ifort for final product. They support most of features I use most of the time.

82. ifort, gfortran: yes. Sun: most. g95: many

83. Intel, gnu. Intel support most desired features. Would have preferred an LLVM based compiler; the return of Lahey/Fujitsu. The biggest issue is the shrinking vendor base.

84. gfortran on both Windows and Linux. Yes they are fine

85. gfortran, yes

86. nagfor, gfortran, ifort, pgfortran. (gfortran, ifort: generally ok; nagfor: missing support for submodules; pgfortran: too buggy for production use)

87. F77 and F95. Have no need for any higher level

88. GCC, iFort

89. gfortran - yes (although it would be nice if coarrays were better integrated, i.e. not having to use the opencoarray library explicitly). ifort - yes.

90. Intel Visual Fortran, NAG Fortran, GFortran: most features are supported

91. Gfortran, Intel Fortran

92. Intel Visual Fortran

93. Gfortran/OpenCoarrays and ifort. Yes, they (will) do support the required features.

94. Ifort 17.0.4, Gfortran 6.3.0 both support the features I currently use.

95. gfortran and (decreasingly, because the performance advantage is slighter than formerly) ifort

96. Intel, GNU, Cray. Most features supported.

97. gnu, nag

98. Intel, NAG, gfortran. Not fully

99. gfortran, intel. Mostly, but poor support for finalisation

100. GFortran, Intel Fortran 2013; Yes

101. gfortran 7, gfortran 8, ifort 2017, pgfortran 2017. They support some features, but not all.

102. mostly gfortran, sometimes ifort.

    I have tried PGI recently, but it's support of fortran 2003 seemed very poor – even my basic test codes did not run. I use f03 features all the time, so this is a dealbreaker.

    A key current limitation is coarray support

    I have had to work-around not-so-good coarray support in ifort (critical parts of my code were rewritten in MPI because ifort 18's coarrays were still not comparably fast). Also I had to provide alternatives to the coarray collectives.

    gfortan + opencoarrays seems quite good for a 'basic but functional' subset of coarrays (e.g. using allocatable coarrays inside a module, with typical point-to-point communication, and collectives).

    One thing that does not yet have good support in gfortran/opencoarrays is allocatable coarrays inside derived types. It seems like this would be very natural for many problems. In general, I like to 'wrap up' my code inside derived types to keep the high-level structure simple and easy to generalise. But thus far I've avoided doing this with coarrays due to compiler limitations, and I'm aware that this is making parts of my code harder to 'cleanly re-use'.

    Other

    Early experiences with object oriented programming in gfortran (4.8-ish) made me avoid crucial elements (e.g. inheritance). Similarly, seeing so many issues with memory leaks / polymorphism discussed on comp.lang.fortran makes me cautious about using it. Thus I tend to be writing 'object-based' codes.

103. gfortran ifort
104. Gnu, Intel
105. gfortran
106. intel gfortran openmp
107. Intel, gfortran, NAG, PGI. Good support for all F95 and 2003. Slowly adopting 2008.
108. I use Nag and gfortran. Nag is a bit behind in terms of features so we can't use it all the time.
109. GNU, Intel, GNU still lacking support/buggy in allocatable character strings
110. gfortran, Intel Fortran. Yes (F2003).
111. gfortran and ifort. good support so far.
112. mostly GNU Fortran 8 (good support for new features), but also Intel Fortran 17+ (worse support for new features)
113. gfortran, intel, cray, I try to stick to widely supported features.
114. gfortran,yes— watfor, only some
115. Gfortran, Intel. Yes
116. ifort, gfortran
117. gfortran, intel ifort. +
118. gfortran and ifort: they both do not completely and reliably implement the features that I'd like to use
119. Intel, GNU
120. Intel ifort 17, NAG nagfor 6.2 (still lacks a few features I'd like to use)
121. Intel Fortran
122. ifort, gfortran
123. f90
124. Intel
125. GFORTRAN, IFORT, PGF90
126. ifort. Yes.
127. gfortran - supports most features I want - parameterized derived types not fully supported (I think)
128. g77, gfortran
129. gfortran, yes
130. I use gnu fortran (and until recently, Absoft Fortran). As you can see from my answers, I'm not a sophisticated user, and make do with what I have in my Fortran 77/95 world! I would like improved handling of character strings, but I suspect that's already in one of the current updates.
131. gnu fortran
132. gfortran-fsf-4.9
133. f77, g77
134. gfortran, ifort
135. Intel Fortran has everything I need
136. gfortran, ifort
137. PGI, GCC - they support everything I need at this time.
138. Intel, gfortran - yes
139. gfortran, ifort
140. gfortran and Intel Fortran
141. gfortran, f95
142. PGI, Intel and GCC. Only Intel supports FINDLOC
143. Intel Fortran. Yes.
144. Intel Fortran
145. gfortran, Absoft
146. g77

147. gfortran - missing further interoperability with C

148. intel

149. gfortran and FTN95

150. gfortran supports most of what I'm interested in, except submodules and error stop in pure procedures.

151. gfortran, ifort; support is okay for the most part

152. Intel, gcc

153. gfortran

154. Cray, GNU, Intel mostly support. PGI, XL, NEC still rather abysmal

155. gfortran and ifort, yes for both

156. Lahey, Absoft, Intel, gfortran.

157. Intel

158. gfortran

159. Ifort and gfortran

160. GNU

161. intel (supports all features), gnu (supports most features)

162. Intel, PGI, GNU, Cray

163. Intel

164. gfortran (yes/no), Intel Fortran (yes)

165. ifort and gfortran. ifort is slightly better in this matter but seems to have more bugs in new Fortran features.

166. Intel

167. intel

168. ifort, yes

169. Intel and Lahey, if I update

170. gfortran (GNU), ifort (Intel), pgf90 (Portland), nagfort (occaisionally)

171. ifort 18, gfortran

172. I am using GCC, it only has very limited intrinsic functions

173. Intel visual fortran

174. Intel and PGI. They supports most of the features what I want.

175. Intel Fortran Compiler

176. Intel Fortran

177. Intel - gfortran

178. Intel and gfortran. Most but not all.

179. Intel mostly

180. ifort and gfortran

181. Intel Fortran, yes

182. gfortran, mostly does what I need

183. I typically have Intel and sometimes gnu.

184. gfortran

185. GNU , Intel, PGI and Flang

186. gfortran 7

187. gfortran. Standard features I use are supported

188. Plato 95 and Microsoft Visual Studio with Fortran 2018. Yes, they support the standards I want to use.

189. gfortran, g95, intel

190. Intel, PGI, Cray, IBM, Gnu. Of these, Intel and Gnu seem to have the most robust support for F2008. In principal I think they all support the features I want but in practice there are bugs...

191. GFortran (supports most new features), PGI doesn't support Coarray which is a major issue!

192. gnu

193. Gfortran, yes

194. gfortran (most features, but the available versions on HPC clusters are often very old), ifort (many features, but often buggy implementations), xlf (many features, but

195. gfortran and ifort. Their current versions support everything I've tried to use, but the versions installed on scientific clusters barely support F2008 so I've had to remove some of those features from my code.

196. PGI

197. gfortran 5.4.0, ifort 2012 vintage

198. gcc, intel

199. gnu, intel. nag for testing and debugging. yes, they do.

200. Intel Fortran Compiler. Yes.

201. gfortran

202. gfortran and ifort

203. GNU Fortran 7, Intel Fortran 14, PGI Fortran 13

204. gfortran, intel

205. gfortran, ifort

206. Intel, Gnu (does not), Pgi

207. gfortran, ifort – both approach the support of the most useful features of 2008

208. gcc 4.8 to 7, intel 17; yes

209. Gfortran,nagfor,ifort,g95

210. Portland, Intel, gfortran

211. nagfor, ifort, gfortran

212. Intel, Sun, gfortran, flang, Cray, PGI, NAG. Some are lacking support for 2003/2008 features, preventing adoption of selected newer features for reasons of portability.

213. gnu, intel, cray: yes they support all I need

214. gfotran

215. I am learning fortran IV for use in pdp-8 emulation. The algebraic context is quite helpful. Fortran IV has relatively few features. A more free, choose-your-code system would make fortran popular and accessible. From an entry standpoint such a platform would create engineering access to competative computer programming.

216. PGI, XLF, gfortran, Cray – none support all

217. I stopped using it in 1982

218. Cray Compiler (near support); GCC (near support); XL Fortran (near support)

219. Cray, PGI, Gnu, Intel. They all have bugs in some of the F2003 features we want to use. For some features there are workarounds required to get them to work.

220. Gfortran, ifort, nagfor

221. gfortran

222. 77

223. F77

224. Gfortran, Intel, PGI

225. gfortran, ifort, pgfortran, SunStudio, g95, CVF, FTN95: All except g95 support everything we currently need for most applications.

226. GNU, Intel

227. Gfortran mostly. It still needs full compatibility with coarrays.

228. GNU Fortran

229. Intel, gfortran, pgi, craympi

230. ifort

231. gfortran. No

232. intel and gnu and yes

233. ifort, gfortran

234. IFORT, if you can afford the newest compiler there is no issue

235. gfortran, nagfor, ifort (yes they do)

236. gfortran, yes

237. gfortran

238. gfortran and NaG compiler

239. gfortran, ifort, pgi, cray, IBM. All support all core features that we use, but we do not try to use features not supported in all these compilers

240. gfortran

241. Intel and GCC. Both support all of the Fortran 2008+ features I use.

242. gfortran

243. Intel ifort, Gnu gfortran, but sometimes Cray's Fortran. Cray's compiler is somewhat older now as is the Cray machine used recently, a few minor problems fixed by reverting back, e.g. don't equivalence to a SEQUENCE struct. The essential "newer" features were fine on all 3, namely allocatable arrays and structures.

244. gnu, ifort. Yes.

245. Intel Fortran 11. Looking at gFortran to move beyond F95.

246. PGI, Intel, Cray

247. Using Intel, NAG, and gfortran. The commercial compilers support most of the standard features I want to use.

248. Intel fortran - yes

249. Intel

250. Ifort, gfortran. I don't really know if they support what I use. I've been programming in C for my job since most folks prefer that.

251. Gfortran and it does

252. gfortran (does not support), Intel ifort (partial support)

253. gfortran, ifort. Yes, except coarrays.

254. intel, gfortran, pgi, xlf

255. gfortran, flang, g77

256. g++, xlf, ifort

257. Intel and fortran; mostly

258. ifort; gfortran; nag - main problem relates to lack of portability of .mod file

259. gfortran (Is OK)

260. gfortran, ifort

261. gfortran and Intel fortran - Yes

262. gfortran, intel

263. gfortran, ifort ... no they do not have powerful macros

264. Intel, Absoft

265. ifort, gfortran, flang

266. GNU, Lahey

267. Intel Fortran, yes

268. Intel

269. Intel, GNU

270. I use gfortran, and it seems to have most of the features I need. The only exceptions have been external math libraries that I have to wrap in a C function.

271. Intel

272. gfortran, ifort

273. gcc/gfortran

274. Intel, PGI, GNU

275. Intel Visual Fortran XE 2013

276. ifort, gfortran

277. gfortran, It seems to provide for my needs and it is free!

278. Intel

279. Gnu

280. gfortran (yes)

281. intel fortran , gnu fortran , PGI fortran. Yes, they have.

282. gfortran, mostly. yes.

283. g77, g95

284. Intel, PGI and some g95. Intel and PGI support all the features I use.

285. Mainly Nec Fortran Compiler. It's behind in adoption of newer standards.

286. GNU Fortran and Intel Fortran most features I need are supported

287. gfortran; yes.

288. gfortran and ifort

289. gfortran

290. Intel, Gnu, Cray, Flang

291. fortran, mostly

292. gfortran, Intel Fortran, PGI, flang, dragonegg

293. gfortran, mostly

294. intel and PGI

295. Intel, gnu, nag

296. Intel, PGI(Nvidia)

297. Intel Fortran, gfortran - coarrays are the one feature that I dearly want to get more acquainted with in a convenient way. Open Coarrays is definitely going to help.

298. Intel composer 2018. Yes.

299. Gfortran

300. gcc, Intel

301. intel, gnu, nag, pgi. Most have relatively good Fortran 2003 support, but pgi support is buggy.

302. Absoft, Intel, Gfortran

303. Gfortran, ifort, xlf. They all do

304. gfortran, Nag, Intel. Needing to support these (and others such as PGI) means only using the common denominator of bug-free features which severely limits the Fortran 2008 features we can use.

305. IBM XLF, gfort etc. I have the features I need.

306. Ifort Cray

307. Intel Fortan, G95, gfortran, HP Fortran, g77

308. gnu fortran 7, Intel Fortran. Not all features supported.

309. ifort, gfortran

310. intel and gnu

311. Intel Fortran Compiler (ifort), GNU Fortran compiler (gfortran). Generally everything is supported promptly in ifort but takes a little while for gfortran to catch up.

312. GNU Fortran

313. intel gnu pgi none support the full set of features in the same way.

314. ifort

315. gfortran, ifort, xlf, ...

316. ifort and to lesser extent gfortran - both ok in terms of standards support (but older versions with incomplete standards support are still in use, sometimes hindering adoption of our new code)

317. xlf/IBM, ifort/INTEL, gfortran/GNU, pgi/PORTLAND

318. GCC, Intel, PGI

319. gfortran, portland group, yes

320. gfortran

321. Gcc and Intel. Yes they support everything we need

322. N/A

323. gfortran, yes

324. GNU & Intel. They claim to support for example procedure pointers, but both exhibit ICE and invalid runtime code on edge cases.

325. GFortran and IFort. At least one does not support %re and %im. At least one does not support hyperbolic functions (atanh etc).

326. GCC gfortran, Intel

327. Intel Fortran, gfortran, pgi

328. Intel, GFortran

329. GNU, IBM and Intel compilers.

330. Intel; yes

331. gnu, intel, nag, pgi if absolutely force to

332. Intel, GNU, PGI, IBM. They do not always support all the features I would want to use

333. GCC, Intel

334. gfortran; yes

335. gfortran, nagfor, ifort. They all support the features I use.

336. gfortran ifort

337. gcc (gfortran)

338. gcc, pgi

339. Gfortran, Intel and PGI. No they don't.

340. gfortran ifort

341. gfortran

342. gfortran not all

343. Intel18

344. intel fortran 2018, absoft pro fortran 2018

345. gfortran: I'm a minor contributor to one project that distributes binaries of 7.2 for consistent results across systems. Earlier versions don't support enough of the new features. My personal work tends not to use many new Fortran features so I've been fine with versions as old as 4.8 (3-5 years old).

346. The thread-safe random number generation (version 7 on) will be critical in planned development, though.

347. Intel, GFortran; yes

348. gfortran, intel, pgi

349. Intel Fortran

350. PGI Fortran

351. GNU Fortran

352. Oracle Fortran

353. PathScale Fortran

354. Intel 18.0, gcc 7.2.0

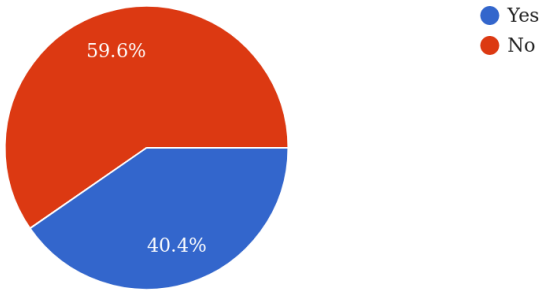
355. Gnu, Intel

356. Intel16.0 Intel 18.0 Gcc/7.2.0

357. intel, xlf, gfortran, partially (but all slightly different)
358. Intel
359. GNU, Intel. They support them.
360. Gfortran, yes
361. gfortran - supports what I use and features I want to start using, but haven't gotten around to yet
362. pgi - In theory is supports what I use, but very buggy and requires lots of workarounds- especially for object-oriented features.
363. cray - Same as pgi above
364. gcc/8.1.0+, Intel 18+, IBM XL 16.1.1+: They do mostly, sometimes buggy
365. gfortran (planning also to use PGI Fortran)
366. Intel, gfortran
367. Gfortran
368. gfortran 6.4,mostly
369. gfortran
370. gfortran, ifort (but we do not use the latest features as our user base is very heterogeneous and compiler versions in use might not support them)
371. GNU and INTEL. They do not actually support all standard features we want to use. Indeed, fragile/partial, and/or late support of some standard features is awful.

Would you like to know more about the work of the Fortran standards committee? If you answer yes, give your name and email so that we can get in touch. In return you will be sent an email outlining how you can best engage with the standard-making process.

374 responses

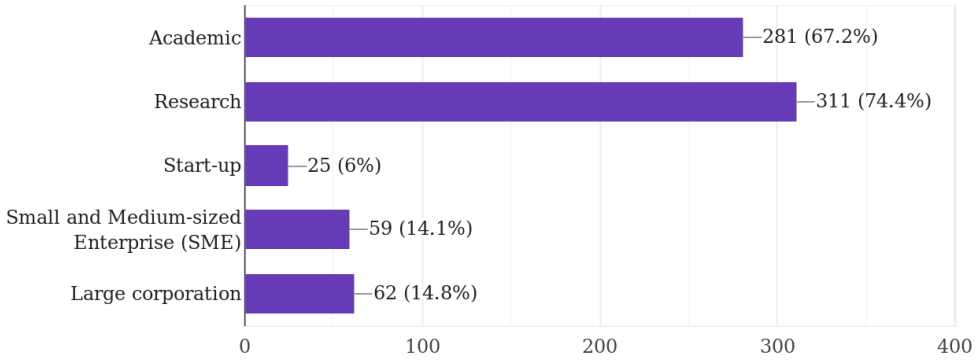


5 About you

Finally some questions about you

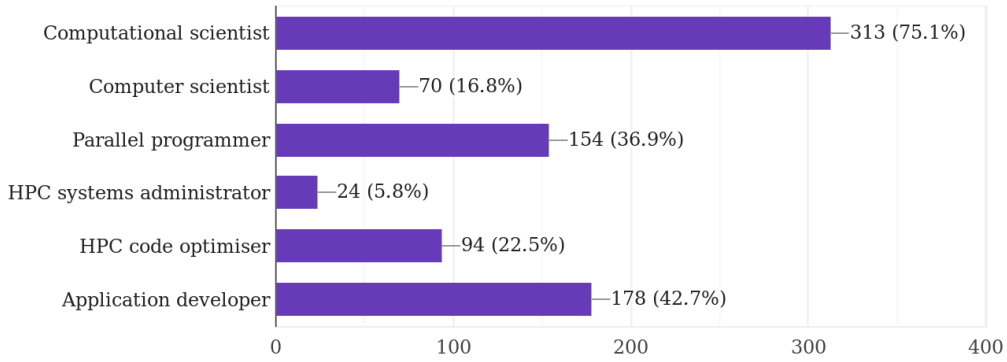
Select the industries that apply to you:

418 responses



What option(s) best describes your role?

417 responses



5.1 Which country are you based in? 384 responses

NUM	%	Country
120	31	United States
117	30	United Kingdom
27	7	Germany
17	4	Netherlands
14	4	France
8	2	Canada
7	2	Italy
6	2	Spain
5	1	Brazil, Russia, Sweden
4	1	Australia, New Zealand
3	<1	Argentina, Belgium, China, India, Norway, Poland
2	<1	Denmark, The Gambia, Greece, Switzerland, Vietnam
1	<1	Algeria, Ashmore and Cartier Islands, Cayman Islands, Chile, Czech Republic, Estonia, Finland, Iran, Ireland, Luxembourg, Malaysia, Romania, Saudi Arabia, Slovenia, South Africa, Turkey, Ukraine
384	Total	