HPC-BP Webinar Q&A

Please, No More Loops (Than Necessary): New Patterns in Fortran 2023

Date: 21 January 2026
Presented by: Damian Rouson (Berkeley Lab)

(The slides are available via the link in the page's sidebar.)

Q:Is object based the same as object oriented?

A: I've seen a couple of different definitions of object-based programming. I described object-based programming as object-oriented programming (OOP) minus x, where I think people choose x as what they consider to be essential to OOP. Most commonly x = inheritance. By that definition, Fortran became object-oriented in the 2003 standard with the introduction of type extension.

Q: Is fortran 90 (and later versions) made to be more like other oop languages like C++? If so, why use fortran and not just other oop? Is the reason to be able to run old fortran codes? I assume one reason to use fortran 90 is that it is fast.

A: Speed is a common reason. There are aspects of Fortran that make it easier for the compiler to optimize, e.g., the anti-aliasing rules and the design of pointers, which are different from C or C++ pointers in several regards. But there are many additional features of Fortran – which was named as a portmanteu for "Formula translator" – that make Fortran especially suitable for writing numerical algorithms. In my view, most notable among these are support for array statements and parallel programming, including shared- and distributed-memory programming and for programming graphics processing units (GPUs). Moreover, people are increasingly recognizing the utility of Fortran in writing non-mathematical utilities that support Fortran. Examples include the Fortran Package Manager (fpm), Julienne correctness-checking framework, and the new fpx.f Fortran pre-processor in Fortran, which I think is pretty exciting. And Fortran's facilities for matrix computations also make it especially useful for writing deep learning libraries such as our Fiats library.

Q: What practical steps can the community take to get compiler developers (GCC,NV,Intel,AMDflang,flang) to put more time and urgency into implementing the full Fortran standard and fixing all bugs? It currently feels very slow…

A: What I believe helps are

1. Submit feature requests and bug reports.
2. If possible, submit the request via a paid license – especially one associated with a big customer of the vendor, e.g., an HPC center such as nersc.gov.
3. If you run jobs at an HPC center that acquires large systems, ask the HPC center to include the desired feature as a requirement during major acquisitions.

   Regarding item 1 above, there's a chicken-and-egg problem, wherein I've heard compiler vendors cite lack of user interest as a reason for not implementing certain features, but the lack of compiler support often explains the lack of user interest. Please let your preferred vendor(s) know about features you want. Additionally,

Q: Given that the features you're highlighting are relatively unknown, have you found that AI is able to effectively use them in generated code? Have you considered developing a Fortran MCP?

A: AI is able to generate fairly modern code, including features like `do concurrent`. AI is getting better at generating code, but it is still important to check it carefully. For a problem asking for about 40 lines of code, based on knowledge that a first- or second-year graduate student would know, only one of four AI models would generate compilable code, and in doing several trials, it regressed. Today's AI models can't get you all the way there in terms of correctness, but they do seem to know about modern language features.

Q: With so many compiler bugs that you are submitting - how are you able to reliably run your code in production?

A: I develop everything first with LLVM Flang, which is the most robust and feature-complete compiler that I've used in at least the past 20 years. In most cases, I can work around other compilers' bugs or missing features. In some cases, however, I've had to drop support for other compilers until the bugs get fixed or features added. Also, because my primary role is leading a programming languages research group,

innovative programming is a higher priority for me than writing production code, which I recognize is different from most people. On several of my software projects, I'm more concerned with getting things to work and less concerned with peak performance or portability. Nonetheless, some of our projects are production projects (e.g., Caffeine and Julienne). In both cases, we put in the effort to make everything work with as many compilers as feasible.

Q: What is the recommended way to compile modern Fortran or the recommended toolchain?

A: I highly recommend the Fortran Package Manager (fpm), which was named when it targeted Fortran almost exclusively, but now does a good job of building projects that include C and C++. For example, I built trivial build systems for SuperLU, which is mostly C, and Armadillo, which is C++. If you can get fpm to work for your project(s), it is likely to require orders of magnitude less time upfront and over time than other build systems. As for compilers, I recommend LLVM flang unless squeezing out the best performance is critical, in which case the answer really depends on many factors so test other compilers.

Q: What is the fault model for multiple images and/or coarray programming?

A: There is an ability to detect failed images and collect failed images into teams (i.e., to isolate them). You'll still need to provide your own checkpoint/restart or neighbor data replication to be able to recover, but there is a basic capability to continue the program in the face of failure. Interestingly, a few years ago, we heard a lot about fault tolerance, but lately not so much.

Q At my organization, we get frequent requests from management to consider translating our millions-of-lines code base from Fortran to some other language, like C++ or Python (with a DSL for HPC, e.g., gt4py). Some reasons are GPU porting/performance, possibly declining support for Fortran compilers, and attracting software engineers. How would you respond to this?

Community Answer:
For GPUs, there are many examples of using Fortran on GPUs on multiple compilers. See https://ieeexplore.ieee.org/document/10820592 for an example with NVIDIA and Intel GPUs.

Q: Many of us need to run code on multiple production machines, each with different compilers. Are there references or guidance for us on what features are supported and stable by the most used compilers? And following up, is there a list of compilers that are sufficiently maintaining and debugging the standard?

Community Answer:
List (not updated since 2019) on Fortran wiki:
https://fortranwiki.org/fortran/show/Compiler+Support+for+Modern+Fortran

Fortran lang list of compilers that is maintained, but doesn't specify feature support:
https://fortran-lang.org/compilers/

Q: In your opinion, what drives developers to write new code/projects in languages such as C++ instead of modern Fortran? There are several large-scale projects in the Earth System modeling (and GeoSciences community in general) where "modernization" in one definition is to rewrite legacy code (predominantly in Fortran) in C++.

A: First, if "modernization" is defined as translating to C++, then one has to account for all the ways in which Fortran is decades ahead of C++ in support for array programming, parallel programming, and GPU programming and will soon be ahead of C++ in template features – while the reverse is true if one looks at other features. It's also really important to address sociological issues, which can cut either way. One issue is that most universities aren't teaching Fortran in the U.S. so on-the-job training is required, but there are several options for bringing someone in to teach short courses and I've done a lot of that. Another issue, however, is productivity: many people feel more productive when writing Fortran. These are all human issues that have to be said out loud and addressed. Seeing the decision as purely technical (as many people often do) is limiting.

Q: In your opinion, will Fortran be able to not lose positions and maybe even get more popular in 5 years, with possible popularity for younger generations?

A: Follow the Tiobe index. In that ranking, Fortran's popularity peaked around number 2 in the 1990s, bottomed out around 31 (roughly where Julia is now) around in the 2010s, but recovered and reached as high as 9 over the past five years. I suspect one reason for the recovery is Fortran's suitability for math and the increasing importance of linear

algebra for neural network training and inference. I hope that at least stabilizes support for Fortran even though it's not likely to rise a lot higher.

Q: Slide 27: does do concurrent impact potential under the cover optimization (vectorization, optimization with respect to memory hierarchy, etc)?

A: Yes, the NVIDIA, Intel, HPE (Cray) compilers can automatically parallelize `do concurrent` on CPUs via multithreading or automatically offload `do concurent` to a GPU. The LLVM flang compiler can automatically parallelize `do concurrent` and work on automatic offloading is under way. The GFortran compiler can vectorize `do concurrent`.

Also, I believe there is benefit in what `do concurrent` communicates to developers. It communicates about 1.5 pages of constraints (in the language standard) that the code must satisfy. That's a lot of information. At a high level, it means that the iterations can happen in any order, including potentially in parallel. Also, the discipline that those constraints impose (e.g., that every procedure called inside a `do concurrent` construct must be `pure` and that those procedures' arguments can't be modified inside the procedure if the procedure is a function) is very useful when refactoring code. Knowing that every procedure in a block of code is `pure` can make life a lot easier – for example, greatly easing code movement.

Q: Is the compiler allowed to fuse implicit loops using the array syntax (with elemental or intrinsic functions)? Any advice on best practices for optimization from a memory bound perspective?

A: It's important to understand that array statements are not implicit loops so I don't think that's the right way to approach this issue. First, the entire right-hand side must be evaluated before the assignments to any left-hand-side arrays can happen. Second, I believe there's no implied ordering in the accesses to the elements on the right-hand side of an array statement. Those accesses can happen in any order that gives the correct result. Important caveat: your mileage may vary. I'm not a performance expert, but I will say that it's important to approach any performance discussion with data. Profile the code. (Most of my experience with profiling is with TAU.) Unless the code in question is the performance bottleneck in the application, I would always advise focusing on code clarity over performance.

Q: How do we learn about all this stuff? I try to keep up on the Fortran language but a lot of this stuff is completely new to me. Are there textbooks or something?

Community Answer: The Fortran Lang website and Discourse are wonderful resources. The "Learn" section of the website could be very helpful and the Discourse is very active for when there are questions that others might answer. The speaker mentioned the utility of Fortran intrinsics, and the website has a section dedicated to the Fortran intrinsics. And there are also well regarded textbooks out there about Fortran as well. For example:

> Metcalf, M., J. Reid, M. Cohen, and R. Bader (2018). Modern Fortran Explained: Incorporating Fortran 2023 Oxford University Press.
>
> Modern Fortran - Milan Curcic
>
> *Scientific Software Design: The Object-Oriented Way* – Rouson, Xia, and Xu (Cambridge University Press, 2011)

Q: Since Fortran will never support threads, what is the best way to take advantage of multiple cores and sockets that now number in hundreds?

A: I'm not sure what is meant by "Fortran will never support threads." If the question is regarding whether there will be features in the language that *explicitly* reference a specific operating-system feature or hardware design feature, then I suspect you're right. But that's because operating systems and hardware platforms evolve and some that are dominant in one era are non-issues in a later era. (At the hardware level, the one that most frequently comes to mine for me is floating-point co-processor, which were common in the 1990s).

As mentioned in the community answer below, for communicating across, I would recommend coarrays or collective subroutines.

Community Answer:
The most compatible way is to use MPI across sockets/nodes and OpenMP for threads. Alternatively, one can use Fortran CoArrays across nodes and Fortran "do concurrent" (DC) for threads if your available compilers support it. I personally use MPI+DC.

Q: The reason why python is so popular in AI is that there are a lot of AI packages. Can we say the same for fortran?

A: There's a vibrant and growing ecosystem of open-source modern Fortran tools, libraries, and utilities. I suspect it would be hard to compete with the ecosystem of Python – although it's worth noting that some of the most popular Python packages exist at least in part to add features to Python that exist in the languages standard in Fortran so there is at least to some extent less need for packages. I suspect that performance and backwards compatibility are two of the more compelling reasons to choose Fortran for production code, but those will never be enough to displace Python for prototyping or tasks that aren't performance critical or scripting tasks for which the code is not likely to hang around long enough for backwards compatibility to be an issue.

Q: Regarding pure functions. Are there any guides on how to get around the error messaging associated with IO? I believe you covered this a little bit vis-a-vis Julienne, but any other tid-bits here would be helpful.

A: If the intent is to detect and report fatal errors during a pure function, a good approach is to use something like the Assert or Julienne, which allow you to enforce program invariants and output diagnostic information when they are violated.

Q: Is there a way to treat array indices as first-class objects? If I want to write a function that behaves like an array (example `y=func(1:4)`), other languages have implemented some form of `index` and `range` objects for such occasion.

A: I think I would need at least a slightly more detailed example to answer this question. I suspect that there's less of a need for this functionality in Fortran, but more information would be helpful. In case you'd like to explain more, I have a blanket policy of accepting all calendar invitations for times that show as open at https://go.lbl.gov/damian-calendar.

Q: How good are coding agents with modern Fortran and which one should I use?

A: In my limited experience, I've found Claude Code best.

Q: What are the best resources for learning the Fortran 2023 multi-image features?

A: Bob Numrich, who invented Co-Array Fortran, published a book: *Parallel Programming with Co-Arrays*. I haven't read it in great detail, but it's likely the best place to start: