Automated Fortran–C++ Bindings for Large-Scale Scientific Applications
(the slides are available under “Presentation Materials” in the above URL)
Date: May 12, 2021
Presented by: Seth Johnson (Oak Ridge National Laboratory)

Q. Interesting slide (#6). Is Fortran losing ground as a "fast" efficient HPC language, or do you think it's more because people started pushing for writing new scientific and HPC codes in C++? Or is it mostly generic programming (which is missing in modern Fortran)

A. Yes to all three. Compiler capabilities and open source communities drive much of contemporary high performance computing, and Fortran lags behind in both due to its more limited usage compared to C++ and Python.

C. A bit off topic, but i’m curious what your thoughts are on the prospects of julia for HPC...

A. I think it's quite promising. It's positioned itself as a modern, agile alternative to Fortran as a domain-specific language for scientific computing. From the outset it's been focused on tackling today's crucial programming issues such as heterogeneous architectures, distributed memory parallelism, modularity, testability, and community-driven development. This is in contrast to Fortran which is forced to retrofit these considerations onto a language with 50 years of backward compatibility to maintain.

C. I think the Scientific Interface Definition Language (SIDL) deserves a mention. It was a nice effort in interoperability, somewhat forgotten by now.

A. Yes, thank you, that's the language I was trying to recall when the question about ROSE (below) was asked during the presentation. I did reference Babel in our CiSE article, but it takes a very different approach from SWIG+Fortran. Our original goal was to adapt an existing C++ interface to Fortran, but Babel/SIDL require an *ab initio* interface definition that existing C++ and Fortran code must then adapt to.

C. Is it possible to say a few words comparing/contrasting SWIG and ROSE (e.g. the ROSE compiler)

A. I've only given ROSE a cursory look or two, but it seems to have an entirely different approach from SWIG+Fortran. ROSE appears to be about source code *transformation*, actually converting C++ code to Fortran or vice versa, rather than generating code for library functions to be used across the language barrier. That being said, I think it could complement SWIG+Fortran in the context of converting a project from Fortran to C++: perhaps using SWIG to replace some Fortran with wrapped C++ library calls, then using ROSE to convert the remaining Fortran code to C++.

Q. Is SWIG currently an ECP funded ST project? We were not able to come across a SWIG poster during this year's ECP annual meeting.

A. The Fortran-targeted components of SWIG were developed as part of the ForTrilinos ST project, which has since been rolled into ALExa. Most of the initial development on SWIG+Fortran happened through 2019, and recent advancements to SWIG+Fortran have been in support of its use by other ECP libraries.

Q. Do you have an example or can you comment on string handling?

A. Yes, the SWIG Fortran manual section on strings gives a detailed breakdown of string handling, but in short a C string (`const char*`) input will generate a `character(len=*), target` Fortran interface, and an output will return `character(len=:), allocatable`. The Flibcpp wrapper for std::string includes additional capabilities.

Q. I see on GitHub that SWIG-Fortran is a fork of SWIG. Any plan to upstream the developments?

A. There is an open merge request to pull in the Fortran advancements. I continue to hope the maintainers will respond to it and collaborate with me on integration.

Q. How does Fortran-C/C++ interop work for shared-memory parallel programs? E.g., for calling a C code from an OpenMP Fortran loop?

A. Whoops, I misheard this question in the online discussion, apologies. I haven't personally played with OpenMP C++/Fortran integration, but I don't think it requires any special care from the Fortran/C++ bindings since all of the data conversion is thread-private. The only non-thread-safe data in generated Fortran code has to do with exception handling.

Q. Is there anything users can do or have to do for more efficient link time optimization?

A. Not to my knowledge at surface level. For the example Andrey did, it was just a matter of adding `-flto` to the C++ and Fortran compilers in addition to the linker. I would guess that improving LTO performance is highly dependent on the compiler implementation.

C. The BABEL tool at LLNL read a SIDL file to create language interoperability. It provided a hub and spokes approach where Fortran could call C++ and C++ call Fortran.

A. Yes, see the above comment—SWIG+Fortran takes a different approach, with a C++ library/codebase as the "producer" and a Fortran application/codebase primarily as a "consumer".

Q. Does row-major vs. col-major array storage order matter much in this context?

A. I can't tell the context from the question here but am assuming it's with regard to idiomatic Fortran. I think the most important piece will be defining and documenting how data is stored and accessed. With ISO-2003 Fortran it's not directly possible to pass a generic multi-D array between languages, so it's harder to accidentally transpose. Trilinos matrices define unambiguous row-based interfaces to achieve a balance.

Q. Any thoughts on f2018 further interoperability with C?

A. I think it's exciting and will definitely simplify the transfer of complicated numerical data across languages. We'll have to wait and see for broad compiler support, of course.