FortranCon 2020

Thursday 02 July 2020 - Saturday 04 July 2020

University of Zurich

Book of Abstracts

Contents

Welcome & Keynote / **1**

Registration

Welcome & Keynote / **2**

Welcome

Welcome & Keynote / **3**

Keynote: Fortran 2018...and Beyond

Steve Lionel, Convenor of the ISO/IEC Fortran Standard Committee, talks about how a Fortran standard is made and then gives an overview of what's new in Fortran 2018. Looking to the future, Steve then highlights features planned for the next revision of the standard, whose working title is Fortran 202X.

4

Closing notes

Corresponding Author: tiziano.mueller@chem.uzh.ch

Session B / **7**

Front-end optimization in gfortran

Author: Thomas König[1]

[1] *Gnu Fortran maintainer*

Corresponding Author: tk@tkoenig.net

The gfortran front end to gcc reads in the source code and converts it to an abstract syntax tree, which is then converted to an intermediate representation which gcc then further optimizes and converts into assembler. However, there are certain optimizations that require knowledge of Fortran semantics and which are better handled at the level of the abstract syntax tree because the rest of the compiler is language-agnostic. In other words, the compiler rewrites the source code representation to something that the user ought to have written.

This presentation gives an overview of what transformations are currently done, how their effect can be controlled by the user and what future developments might be. On that last point, input from the community is highly welcome.

Session C / **8**

Using R with High Performance Fortran on a Windows Laptop

Author: Erin Hodgess[1]

[1] *Western Governors University*

Corresponding Author: erin.hodgess@wgu.edu

We will discuss the integration of R with Fortran, using such tools as MPI, OpenACC and CUDA Fortran on a Windows laptop. We will consider how to create a package with all three of these tools. We will demonstrate the considerable speedup with ordinary kriging and other spatial functions. We can exploit the statistical functions by utilizing the NVIDIA graphics card in conjunction with the PGI Community Edition Fortran compiler. Our package can extend the current functions by extreme speedups.

Session G / **9**

Highly Parallel Fortran and OpenACC Directives

Authors: Jeff Larkin[1]; Michael Wolfe[1]

[1] *NVIDIA*

Corresponding Authors: jlarkin@nvidia.com, mwolfe@nvidia.com

Fortran has long been the language of computational math and science and it has outlived many of the computer architectures on which it has been used. Modern Fortran must be able to run on modern, highly parallel, heterogeneous computer architectures. A significant number of Fortran programmers have had success programming for heterogeneous machines by pairing Fortran with the OpenACC language for directives-based parallel programming. This includes some of the most widely-used Fortran applications in the world, such as VASP and Gaussian. This presentation will discuss what makes OpenACC a good fit for Fortran programmers and what the OpenACC language is doing to promote the use of native language parallelism in Fortran, such as do concurrent and Co-arrays.

Session D / **10**

Designing a Modern C++/Fortran Interface by Example

Author: Maximilien Ambroise[1]

[1] *Universität Heidelberg*

Corresponding Author: maximilien.ambroise@iwr.uni-heidelberg.de

In the world of quantum chemistry programs, Fortran reigns supreme. While there are packages available that are purely written in C++ and Python, it has become increasingly common to combine different languages. Common combinations include C++/Fortran or Python/C++.

Fortran 2003 introduced a standardized way to generate interoperable procedures and derived types with the C programming language, using the BIND(C) attribute. This was a necessary step, as interfaces from C to legacy Fortran code had compiler-specific problems which made portability a major issue. Following Fortran 2008 and 2018, more technical specifications were accepted that furthered interoperability.

In this presentation, we will discuss the development of a C/C++ interface to a modern Fortran library called DBCSR (Distributed Block Compressed Sparse Row), designed for efficient sparse matrix-matrix multiplication, among other operations. This interface is currently being incorporated in a novel sparse-tensor quantum chemistry program written in C++. Several points about C/Fortran interoperability will be addressed, such as memory management, derived types, handling arrays, and problems encountered therein, as well as how preprocessors may be used to help design a modern C++ interface with Fortran at its roots.

Session G / **12**

ParaMonte: Plain Powerful Parallel Monte Carlo Library

Authors: Amir Shahmoradi[1]; Fatemeh Bagheri[1]; Joshua Osborne[1]; Shashank Kumbhare[1]

[1] *The University of Texas*

Corresponding Authors: a.shahmoradi@uta.edu, bagheri.fateme@gmail.com, shashankkumbhare8@gmail.com, joshuaalexanderosborne@gmail.com

We present the ParaMonte library, a pure-modern-Fortran open-source software for serial and parallel stochastic sampling and integration of high-dimensional mathematical objective functions of arbitrary shapes and dimensions. The principal design goals of the ParaMonte library are: 1. full automation of the entire build process of the library as well as all Monte Carlo simulations, 2. interoperability of the core library with multiple programming languages, including C/C++/MATLAB/Python/..., via the C-interoperability features of the Fortran language, 3. high-performance 4. parallelizability and scalability of the simulations, 5. virtually zero-dependence on external libraries, 6. fully-deterministic reproducibility and continuation of all stochastic simulations, 7. automatic comprehensive-reporting and post-processing of the simulation results. The library is Fortran-2018 standard compliant and the parallelization of the code relies on the MPI and Coarray parallelism paradigms within the Fortran programming language. We discuss how these design goals can help the ParaMonte users readily and efficiently solve a variety of machine learning and scientific inference problems on a wide range of platforms, from Jupyter notebooks on personal laptops to supercomputers. We also discuss how the modern features of the Fortran language simplified software development and what new language features would be desired from the developer perspective.
https://www.cdslab.org/paramonte/

Session D / **13**

Shroud: generate Fortran wrappers for C and C++ libraries

Author: Lee Taylor[1]

[1] *Lawrence Livermore National Laboratory*

Corresponding Author: taylor16@llnl.gov

Fortran application often need to access libraries which are written in C or C++. The interoperability with C features introduced in Fortran 2003 standardized access to symbols and types. But to access all of the features of C++ libraries additional wrapper code must be written, often by the author of the C++ library who may not be familiar with modern Fortran features.
Shroud is a tool to create an idiomatic Fortran interface for a C++ library. The user creates a YAML file with the C/C++ declarations to be wrapped along with annotations to provide semantic information and code generation options.
Many C++ features will map directly to Fortran such as classes,

overloaded function, default arguments and template instantiation.
Shroud has successfully been used by several projects over the past four years.
Shroud is written in Python and available at github.com/llnl/shroud with a BSD license.

Session G / **14**

Parallelization of a Legacy Software through Fortran 2018 Standard

Author: Nicolas Netto[1]

[1] *Electrical Energy Research Center (Cepel)*

Corresponding Author: nicolasrln@cepel.br

Anatem is a software developed by the Brazilian Electrical Energy Research Center (Cepel) that aims to evaluate electromechanical disturbances on large power systems, being the most used software in Brazil for that matter. Its development dates from the late '80s through today, being a legacy code mostly written in Fortran 77 standard and before object-oriented programming mindset even existed. In the face of a challenge to achieve real-time simulation, parallel techniques were deployed employing Fortran 2018 standard with Intel Fortran 2020 compiler, reaching desirable results in less than six months of work.

This presentation discusses how coarrays and collective functions could be applied to obtain fast implementations, with less coding than alternatives like the MPI library, the upsides, and the downsides for the before-mentioned approach.

Session D / **15**

Connecting Fortran to the Internet of Things

Author: Philipp Engel[None]

Corresponding Author: pengel@hs-nb.de

At first glance, Fortran may not be easily associated with the emerging Internet of Things and its new networked services, data formats, and protocols. In fact, the language features of modern Fortran make it possible to inter-connect with existing third-party libraries, written in C, Lua, Python, and other languages. With the toolset given by the ISO C binding module, Fortran applications gain access to RESTful web services, NoSQL databases, ZeroMQ message queues, MQTT-based pub/sub middleware, or IoT sensor networks. The talk will give an overview of existing technologies and how to access them from Fortran.

Session A / **16**

Fortran Package Manager

Authors: Brad Richardson[None]; Milan Curcic[1]; Ondřej Čertík[None]

[1] *University of Miami*

Corresponding Authors: everythingfunctional@protonmail.com, caomaco@gmail.com, ondrej@certik.us

While Fortran is the oldest high level language, it has done quite well in keeping up with the times in terms of features and capabilities of the language itself. However, modern practices and developers have become accustomed to tools and ecosystems which provide many conveniences in a programming environment. Unfortunately, Fortran has not kept pace with such tooling and ecosystems. One such tool which has become popular is a package manager. A package manger is a tool that manages the dependencies of a project on other libraries. This is accomplished by keeping track of the dependencies, with specifiable version constraints, and automating the process of fetching them - including transitive dependencies - for use in the compilation of the project. Often included are the facilities for compiling, running, and testing the project, as well as searching for available open source libraries, and generating a template for new projects. This paper describes the development of just such a tool for Fortran, aptly named the Fortran Package Manager (FPM).

Session F / 18

Applying context-free grammar to hierarchically organized and variably shaped arrays

Authors: Robert Schweppe[1]; Stephan Thober[1]; Sebastian Müller[1]; Luis Samaniego[1]

[1] *Helmholtz-Centre for Environmental Research Leipzig - UFZ*

Corresponding Authors: stephan.thober@ufz.de, robert.schweppe@ufz.de, luis.samaniego@ufz.de, sebastian.mueller@ufz.de

In the community of environmental modelling, the advent of hyper-resolution Earth observations and datasets in conjuncture with growing computational resources lead to an increase in model resolution.
The mathematical representations of biogeophysical processes need to be solved for billions of grid cells and thousands of time points.
Each process requires parameters that cannot be easily and sensibly set fix nor calibrated.
Instead they need to be inferred directly from the land surface properties through transfer functions.
In a simple form, they follow a context free grammar which follows the Fortran language.
The transferred effective parameters have a hierarchical interdependency forming a tree structure.
Yet finally, the shape of the arrays containing the land surface properties does usually not conform with the shape of the array of process parameters of the model.
This necessitates multiple array broadcasting, slicing and remapping steps.
The scientific approach - the Multiscale Parameter Regionalization (MPR) concept - is now available as an object-oriented and flexible Fortran library (https://git.ufz.de/chs/MPR).

In this presentation, we discuss the design of the MPR library, show implementation details and highlight major difficulties. We are strongly interested in community feedback on the implementation.

Session F / 19

Interfacing with OpenCL from Modern Fortran for Highly Parallel Workloads

Author: Laurence Kedward[1]

[1] *University of Bristol, UK*

Corresponding Author: laurence.kedward@bristol.ac.uk

OpenCL is a well-established and widely-supported standard for executing parallel workloads on accelerator devices such as conventional multicore CPUs as well as GPUs and FPGAs.

In this presentation, detail is given on a modern Fortran library which wraps calls to the OpenCL API with a higher abstraction level aimed at scientists and engineers looking to execute highly-parallel OpenCL kernels from Fortran. Modern Fortran features, including derived types, generics, operator-overloading and the iso c binding, are exploited to bring the Fortran style to OpenCL by: abstracting away pointers; providing a level of type-safety for device memory; detecting and handling program errors in a user-friendly manner; and providing a concise but feature-rich interface.

Device kernels are written in the OpenCL C dialect and the Fortran library provides routines to: initialize the accelerator, allocate device memory, enqueue kernels for execution, perform memory transfers and manage device synchronisation.

Code extracts and results are presented for two fluid dynamics codes implementing a lattice Boltzmann method and a multigrid finite volume Euler solver. Background is given on the challenges and design choices for programming GPU hardware from a Fortran perspective, followed by discussion on the future of accelerator offloading from the Fortran language.

Session F / 20

A Fortran-Keras Deep Learning Bridge for Scientific Computing

Authors: Jordan Ott[1]; Mike Pritchard[1]; Natalie Best[2]; Erik Linstead[2]; Milan Curcic[3]; Pierre Baldi[1]

[1] *UC Irvine*

[2] *Chapman University*

[3] *University of Miami*

Corresponding Authors: caomaco@gmail.com, linstead@chapman.edu, mspritch@uci.edu, pfbaldi@ics.uci.edu, best120@mail.chapman.edu, jott1@uci.edu

Implementing artificial neural networks is commonly achieved via high-level programming languages like Python, and easy-to-use deep learning libraries like Keras. These software libraries come pre-loaded with a variety of network architectures, provide autodifferentiation, and support GPUs for fast and efficient computation. As a result, a deep learning practitioner will favor training a neural network model in Python where these tools are readily available. However, many large-scale scientific computation projects are written in Fortran, which makes them difficult to integrate with modern deep learning methods. To alleviate this problem, we introduce a software library, the Fortran-Keras Bridge (FKB). This two-way bridge connects environments where deep learning resources are plentiful, with those where they are scarce. The library a number of unique features offered by FKB, such as customizable layers, loss functions, and network ensembles. We apply FKB to address open questions about the robustness of an experimental approach to global climate simulation, in which subgrid physics are outsourced to deep neural network emulators. In this context, FKB enables a hyperparameter search of one hundred plus candidate models of subgrid cloud and radiation physics, initially implemented in Keras, to then be transferred and used in Fortran to assess their emergent behavior.

Session F / 21

Code::Blocks: open source, cross platform IDE for Fortran

Author: Darius Markauskas[1]

[1] *TU Berlin*

Corresponding Author: markauskas@tu-berlin.de

While it is possible to write Fortran code with a simple text editor, many programmers prefer to use an IDE (Integrated Development Environment) for their work. In addition to simply highlighting text in many editors, Code::Blocks offers Fortran users a grouping of their code files into projects, compiling code with the selected compiler directly from the IDE, code navigation, code completion and debugging with GDB debugger and more. Code::Blocks IDE is an open source project that has been continuously developed in C ++ by many developers from different countries since 2004. The source code of the IDE is organized in the core and in many plugins. Most of the functions specific to Fortran are implemented in the FortranProject plugin. The presentation explains many features of this IDE that are useful for programming in Fortran. It shows how to create a project and how to add existing code to this project, how to navigate the code and how to debug it. Some new features and ongoing work are also shown.

Session B / **22**

Flang: The LLVM Fortran Front-End

Authors: Gary Klimowicz[1]; Steve Scalpone[1]

[1] *NVIDIA*

Corresponding Author: gklimowicz@nvidia.com

We'll present the goals, current status and future plans for Flang, the LLVM Fortran front-end.

Flang is the new LLVM-based Fortran front-end supporting full Fortran 2018. It is being written from scratch in modern C++ and will make extensive use of existing LLVM tools (MLIR, LLVM IR, utility libraries, and so on). We expect this to be the last Fortran front-end that will ever need to be written.

The primary goals of Flang are
- Build a robust Fortran compiler using modern compiler techniques (parser combinators, multi-level intermediate representations) that encompasses all of the Fortran 2018 standard.
- Develop an active community of developers (now DOE, NVIDIA, Arm, AMD and others).
- Provide support for Fortran tool development.
- Provide support for Fortran language experimentation for features proposed for future Fortran standards.

60,000+ lines of the compiler have been upstreamed to the flang/ directory of the LLVM monorepo. Its build integrates with the rest of LLVM (as an optional component). This covers parsing, semantic analysis and the start on lowering to an MLIR dialect called FIR.

We will also briefly discuss the next steps for the project.

Session C / **23**

Copernicus Spacecraft Trajectory Design and Optimization Program

Author: Jacob Williams[1]

[1] *NASA Johnson Space Center*

Corresponding Author: jacob.williams-1@nasa.gov

Copernicus is a spacecraft trajectory design and optimization application developed at the NASA Johnson Space Center. Copernicus is written in Fortran and uses many features of the latest language

standards. The tool is used for a wide range of projects at NASA, including the upcoming Artemis missions to flight test the Orion spacecraft and then return humans to the Moon. This presentation will give a brief overview of the software, its history, how it was designed, and how it is used.

Session E / **24**

Program flow control using scripting languages

Authors: Nick Papior[1]; Alberto Garcia[2]

[1] *DTU DCC*

[2] *ICMAB*

Corresponding Author: nickpapior@gmail.com

Controlling program flow from scripting languages is a tractable extension which lowers barriers for extending functionality in lower level languages (such as fortran).

In this presentation we show how a set of required features enables end-users to change and control the program flow using Lua scripts.
The following features are necessary for minimal code in the hosting program:
- dictionaries for seamless data exchange with little coding effort
The dictionaries allows (pointers to) data to be referenced via characters and ensures no intermediate copying of data.
- running a Lua interpreter
- creating interaction points in program
This is the most problematic part since you have to expose *break* points where the hosting program stops and calls Lua. Another approach would be to build your entire program around scripting.

In our density functional theory program (Siesta) we expose Lua for users to implement their own molecular dynamics engines, change convergence criteria/properties based on self consistent cycles, and more.

This coding effort revealed substantial insight on how to control and expose data structures for end users in scripting languages.
We envision that other scripting languages are better suited since the communities are not experienced Lua coders/users, e.g. Python, nim.

[aotus,flook,fdict]

Session C / **25**

EIS2 - A mathematically rich input file processor for HPC applications

Author: Christopher Brady[1]

Co-author: Heather Ratcliffe [1]

[1] *University of Warwick*

Corresponding Authors: c.s.brady@warwick.ac.uk, h.ratcliffe@warwick.ac.uk

Allowing users to control software using input control files has substantial benefits for ease of use, avoidance of error and reproducibility of results. Evaluation of mathematical expressions provides a powerful way of allowing user control of even the most complex codes while remaining natural

and easy-to-learn for the end user. Derived from the input control system of the EPOCH particle-in-cell plasma code, EIS2 allows a developer to add rich text file based controls to a large scientific code. EIS2 combines a mechanism for reading structured, hierarchical text files with a mechanism for evaluting mathematical expressions while allowing the host code to provide contextual information as the expression is evaluated. This means that a text expression provided by a user such as (1+sin(x))*(1+cos(y)) can be easily evaluated over a range of x and y values and used to set values as required by the host code. By using an approach that is optimised for performance rather than generality evaluating these expressions is many times faster than a general purpose parser such as Python.

EIS2 is written in standard Fortran 2003 (with optional 2008 extensions) and has a C interoperable interface for other languages (releasing soon).

https://github.com/csbrady-warwick/EIS-2

Session A / **26**

Fortran Standard Library

Authors: Jeremie Vandenplas[1]; Bálint Aradi[2]; Izaak Beekman[3]; Ondřej Čertík[None]; Milan Curcic[4]; Pierre de Buyl[5]; Juan Fiol[None]; Michael Hirsch[6]; Yvan Pribec[7]; Nathaniel Shaffer[8]

[1] *WUR*

[2] *University of Bremen*

[3] *ParaTools Inc.*

[4] *University of Miami*

[5] *KU Leuven*

[6] *SciVision Inc.*

[7] *Technical University of Munich*

[8] *Los Alamos National Laboratory*

Corresponding Authors: caomaco@gmail.com, jeremie.vandenplas@wur.nl, ivan.pribec@tum.de, contact@izaakbeekman.com, pdebuyl@pdebuyl.be, nrshaffer@protonmail.com, info@scivision.dev, ondrej@certik.us, aradi@uni-bremen.de

The Fortran Standard, as published by the International Organization for Standardization (ISO), does not include a Standard Library. The language can be extended with new intrinsic procedures and modules, but these must be formally standardized and then implemented by compiler vendors before becoming available to users. Therefore, the goal of this project is to provide a community driven and agreed upon de facto "standard" library for Modern Fortran, called the Fortran Standard Library (stdlib; https://github.com/fortran-lang/stdlib). This library aims to provide to the community a set of procedures for science, engineering, and mathematics. The overall scope of the Fortran Standard Library is therefore similar to the one of SciPy or to the default built-in Matlab scientific environment. Currently the library includes procedures for catching and handling errors, handling optional arguments, facilitating I/O operations, linear algebra, numerical integration, and descriptive statistics. Started a few months ago, no less than 15 people already contributed to the development of the Fortran Standard Library and its documentation. Many other programmers are also involved in active discussions about its development through GitHub issues. The development of this library is part of the Fortran-lang project and aims to collaborate with the Fortran Standards Committee.

Session B / **27**

LFortran: Interactive LLVM-based Fortran Compiler for Modern Architectures

Authors: Ondřej Čertík[1]; Nikhil Maan[2]; Ankit Pandey[3]; Milan Curcic[4]; Peter Brady[1]; Zach Jibben[1]; Neil Carlson[1]; Rohit Goswami[5]; Amir Shahmoradi[6]; Arjen Markus[7]

[1] *Los Alamos National Laboratory*

[2] *Amity University, India*

[3] *Grinnell College*

[4] *University of Miami*

[5] *Science Institute, University of Iceland, VR-III, 107, Reykjavik, Iceland and Department of Chemistry, IIT Kanpur, India*

[6] *the University of Texas Arlington*

[7] *Deltares, The Netherlands*

Corresponding Authors: caomaco@gmail.com, zjibben@lanl.gov, nnc@lanl.gov, ptb@lanl.gov, pandeyan@grinnell.edu, ondrej@certik.us, rog32@hi.is, arjen.markus@deltares.nl, nikhilmaan22@gmail.com, shahmoradi@utexas.edu

We are developing a modern open-source Fortran compiler called LFortran (https://lfortran.org/). This front-end compiler will enable the interactive execution of code in environments like Jupyter. This will allow exploratory work (much like Python, MATLAB or Julia) which is currently not feasible. The interactivity of our compiler does not impede compilation of binaries with the goal to run user's code on modern architectures such as multi-core CPUs and GPUs, which is an essential requirement for wider Fortran adoption that current Fortran compilers do not address well. A Live demo of the compiler with a Jupyter notebook will be shown. The compiler itself is written in C++ for robustness and speed with optional Python wrappers to improve inter-operability. It parses Fortran code to an Abstract Syntax Tree (AST) and transforms it to an Abstract Semantic Representation (ASR). LFortran has several backends that transform the ASR to machine code via LLVM, or to C++, or to provide automatic Python wrappers. More backends are planned. The compiler has been designed to be modular so that data can be extracted/inserted between the different stages, which is an important feature that would support an ecosystem of tools that otherwise would be hard with a monolithic compiler.

Session A / **28**

Toward a thriving open source Fortran community

Author: Milan Curcic[1]

Co-authors: Ondřej Čertík ; Laurence Kedward [2]; Vincent MAGNIN [3]; Ivan Pribec [4]; Brad Richardson ; Jeremie Vandenplas [5]

[1] *University of Miami*

[2] *University of Bristol, UK*

[3] *Univ. Lille, CNRS, Centrale Lille, Yncréa ISEN, Univ. Polytechnique Hauts-de-France, UMR 8520 - IEMN, F-59000 Lille, France.*

[4] *Technical University of Munich*

[5] *WUR*

Corresponding Authors: everythingfunctional@protonmail.com, caomaco@gmail.com, laurence.kedward@bristol.ac.uk, jeremie.vandenplas@wur.nl, ivan.pribec@tum.de, ondrej@certik.us, vincent.magnin@univ-lille.fr

A thriving community around a programming language is essential for onboarding new users and retaining existing ones. Besides a few discussion boards and mailing lists, Fortran has not had a healthy online community in the modern internet era, like many other programming languages have. Despite the developments of the language, its user base has been declining as a result. Many new software projects are being started in languages like C++, Python, or Julia, despite Fortran being better suited for the task at hand. In this talk, we will discuss the need for and the ongoing development of the new fortran-lang (https://fortran-lang.org) community and its suite of projects that currently include a standard library, package manager, and a website. Fortran-lang aims to provide a central place for Fortran users, beginning and expert alike, to find recommended

tools, libraries, tutorials, and discussion venues that are inclusive and welcoming of newcomers. We believe that an organized community and a home of Fortran on the internet, in addition to the Standards Committee and compiler developers, are essential for the long-term thriving—and not just mere survival—of Fortran. Finally, we will discuss the next steps, the one-year vision, and the 10-year vision for going forward.

Session D / **29**

gtk-fortran: a GTK / Fortran binding

Author: Vincent MAGNIN[1]

Co-authors: James TAPPIN [2]; Jens HUNGER [3]; Jerry DeLisle [4]

[1] *Univ. Lille, CNRS, Centrale Lille, Yncréa ISEN, Univ. Polytechnique Hauts-de-France, UMR 8520 - IEMN, F-59000 Lille, France.*

[2] *RAL Space, STFC Rutherford Appleton Laboratory, Harwell Campus, Didcot,Oxfordshire OX11 0QX, United Kingdom*

[3] *Technische Universität Dresden, Department of Chemistry and Food Chemistry, Dresden, Germany*

[4] *GFortran Team, USA*

Corresponding Authors: vincent.magnin@univ-lille.fr, jhunger@protonmail.com, james.tappin@stfc.ac.uk, jvdelisle@charter.net

The gtk-fortran binding (https://github.com/vmagnin/gtk-fortran/wiki), developed since 2011, is based on the ISO_C_BINDING module, introduced in the Fortran 2003 standard, which is used to interface fortran programs with the functions of the GTK libraries (mainly written in C). gtk-fortran is multi-platform: Linux, FreeBSD, MacOS, Windows (via MSYS2), and even Raspberry Pi (ARM processor)…

GTK being a collection of libraries (GTK, Pango, GDK, ATK, Cairo, GdkPixbuf, GLib…), a python script parses the hundreds of GTK header files and generates around 10000 Fortran / C interfaces. A supplementary High Level library can ease programming, and PLplot can also be used in gtk-fortran. The user can learn to create Graphical User Interfaces using its wiki documentation and the commented Fortran examples.

We will finally speak of the new GTK 4 development branch, our objective being to be ready for the release of that next major version around the end of the year. We will also present some ideas for the future and a SWOT analysis of the project.

Session G / **30**

F2PY: Bringing fast code into the future

Authors: Melissa Weber Mendonça[1]; Pearu Peterson[1]

[1] *Quansight*

Corresponding Authors: melissawm@gmail.com, pearu.peterson@quansight.com

F2PY is a tool (authored by Pearu Peterson in 1999) to generate custom CPython
extension modules to interface high-performance Fortran or C libraries to high-level Python code. The F2PY tool, currently packaged with NumPy, is one of the most fundamental packages in the scientific Python ecosystem. On the other hand, F2PY has not kept up with modern Fortran standards, as evidenced by the lack of support for user-defined types or derived types, for example. Adding these new features will improve significantly the applicability range of F2PY to interfacing modern

Fortran libraries to Python code. Since Fortran is often, but unfairly, considered a niche programming language, finding contributors and building a community around this package has proven difficult, in spite of its importance and relevance in modern scientific code.

In this talk, we will discuss the current status of the F2PY project and possible ways forward. This will include a discussion on how to create extension modules and F2PY's approach to doing this, with concrete examples for those unfamiliar with the tool. In addition, we will discuss current efforts to modernize this tool to extend its capabilities and make sure that projects that depend on it are supported in the future.

Session E / **31**

Experimental Fortran Programming

Author: Arjen Markus[1]

[1] *Deltares*

Corresponding Author: arjen.markus@deltares.nl

While Fortran is usually used for serious work, number crunching for instance, it does not mean that that is the only way to use it. Far from it, the features offered by modern Fortran allow all manner of experimentation with other programming paradigms. Emulating, say, prototype-based object-oriented programming may not lead to an efficient implementation, but it offers the benefits of demonstrating what the possibilities are of such a paradigm. In this talk I would like to present three cases of extending the language beyond the obvious:

- Using user-defined operators to stay close to the mathematical notation of differential equations
- An alternative way of looking at object-oriented programming
- Introducing "lambda expressions"or anonymous functions

As stated, the implementation probably will not lead to very efficient programs, but thinking beyond the traditional will bring new possibilities to light.

Session C / **32**

Evolving Fortran for Emerging Architectures: Lessons from the ICON-GPU Atmospheric Model

Author: William Sawyer[1]

[1] *Swiss National Supercomputing Centre*

Corresponding Author: wsawyer@cscs.ch

For decades Fortran has been on the forefront of high performance computing. As new architectures emerged, the Fortran standard added constructs to exploit them, but not always with complete success. For example, F90 added array syntax to describe vectorization, yet vectorizing compilers found it easier to translate DO loops with appropriate directives. Co-arrays were added in F08 and extended in F18 to support distributed memory, but this is usually addressed by the Message-Passing Interface (MPI). Other libraries, e.g., BLAS, LAPACK or PLASMA cover optimized numerical calculations outside of the language.

The advent of General Purpose Graphics Processing Units (GPGPUs) has created another conundrum. They can be programmed with an appropriate language (CUDA, CUDAFortran, or OpenCL) or with

directives (e.g., OpenMP4.5 or OpenACC3.0), each with disadvantages. In this talk, we outline the lessons learned in porting the ICON atmospheric model to GPUs with OpenCL, CUDAFortran and, finally, OpenACC, with the latter now in production at the Swiss National Supercomputing Centre (CSCS). Now that we understand the programming challenges, it is possible to consider new extensions to the Fortran standard to address GPUs, which are clearly not going away any time soon. We attempt to give some future perspectives.

Session E / **33**

The Futile project: an embedded DSL to simplify the treatment of low-level operation in large Fortran programs

Author: Luigi Genovese[1]

[1] *CEA Grenoble*

Corresponding Author: luigi.genovese@cea.fr

While writing a FORTRAN code, the developer is often obliged to increase code smell (see https://en.wikipedia.org/wiki/Code_sm due to the intrinsic characteristic of the programming language.
This problem not only generate inaesthetically long code, but is also responsible of the great majority of side-effects and bugs.
The FUTILE project is an attempt to simplify developer's life by taking care of some of these operations.
By definition, this suite of modules is conceived for Fortran programmers. The Fortran standard used is F95, with only minor extensions to F2003 standard, in the aim of increase portability with older codes.
In addition to that, FUTILE package might be also seen as a framework, which helps traditional Fortran developers to "think differently" and to write code subprograms which are similar to those of higher level, "object-oriented" programming.

Session E / **34**

Generic Programming Techniques

Author: Patrick Seewald[1]

[1] *University of Zurich*

Corresponding Author: patrick.seewald@chem.uzh.ch

Different strategies towards generic programming in Fortran are discussed. The Fypp preprocessor is presented as a versatile tool for condititional compilation and template metaprogramming.