Interactive Programming Documentation for Data Science

Organisation: Compiler Research Organisation – GSoD'23 Page

Description: We are a group of programming languages enthusiasts located at Princeton University and CERN. Our primary goal is research into foundational software tools helping scientists to program for speed, interoperability, interactivity, flexibility, and reproducibility.

Our current research focus is primarily in interpretative C/C++/CUDA, automatic differentiation tools, and C++ language interoperability with Python and beyond.

Authors: @QuillPusher, David Lange

Problem Statement

This Google Season of Docs project aimed to advance the documentation in the area of interactive analysis, both in the context of C++ and support for C++/Python interoperability. Our intent was to build the sort of documentation that enables user engagement while being easy to update as our codes continue to evolve and improve.

Our background is in enabling particle physics researchers to do cutting-edge data analysis. To make this happen, researchers have developed several unique software technologies in the area of data analysis. We developed an interactive, interpretative C++ interpreter (aka REPL) as part of the ROOT data analysis project, including "Cling", which is a REPL based on LLVM. Cling is a core component of ROOT and has been in production since 2014. Cling is also a standalone tool, which has a growing community outside our field. It is recognized for enabling interactivity, dynamic interoperability and rapid prototyping capabilities for C++ developers. For example, if you are typing C++ in a Jupyter notebook you are using the xeus-cling Jupyter kernel.

We are in the midst of an important project to address one of the major challenges to ensure Cling's sustainability and to foster that growing community: moving most parts of Cling into LLVM. Since LLVM version 13 we have released a version of Cling called Clang-Repl within LLVM itself. We subsequently focused on the language interoperability capabilities of Cling. One user-facing application of our libInterOp, together with Clang-Repl, is Xeus-Clang-Repl, which is a replacement for xeus-cling using these new codes. As we advance the implementation and generalise its usage we aim for improving the overall documentation experience in the area of interactive programming in various environments.

Proposal Abstract

The project proposed was to audit and extend the existing documentation for the Clang-Repl (interactive C++), Xeus-Clang-Repl (notebook-based C++ and Python platform) and libInterOp (bridging automatically C++ and Python). We aim to identify gaps in the information or presentation from the point-of-view of new, science-oriented users.

The anticipated scope of the work was:

- Improve user and developer documentation about xeus-clang-repl
- Write several tutorials demonstrating the current capabilities of clang-repl.
- Prepare a blog post (or posts) about clang-repl and xeus-clang-repl.
- Improve user and developer documentation about CppInterOp library API and usage.
- Develop a set of blog posts on how to use the CppInterOp library API and usage.
- Develop a set of blog posts on how to use the CppInterOp library together with higher-level tool kits such as Cppyy.

We aimed to advance the documentation in the area of interactive analysis, both in the context of C++ and support for C++/Python interoperability. Our intent was to build the sort of documentation that enabled user engagement while being easy to update as our codes continue to evolve and improve.

We also wanted to capture the Student Success Stories, including milestones achieved by student developers under the banner of Compiler Research Org's guidance and mentorship programs.

Project Description

Creating the proposal

Our organisation had internal discussion areas of recent development and how each would benefit from a technical writer of the sort we were likely to find through Google Season of Docs. We had the benefit of working with a successful GSoD writer previously and so had some ways upon which to differentiate appropriate ideas. From there the idea was socialised within our team during project meetings and revised given ideas brought forward.

Budget

The budget and corresponding work scope was estimated based on prior experience and availability schedules of each mentor (e.g., the reinterpretation work was constrained due to mentor unavailability once the fall semester started). The largest uncertainty was due to the regional variation in salary expectations of technical writers based on their location, which was uncertain at the time of the proposal. Nevertheless, the overall budget appeared to be appropriate given the time period and project scope that we envisioned.

Our budget turned out to be on target. Our expectations for technical writer productivity and costs roughly matched what we had planned for in our proposal. No additional funds outside of Season of Docs were used to fund the technical writers, however the project did build upon work of a previous GSoD writer and a Google Summer of Code student that contributed significant documentation infrastructure as part of their project.

Participants

We had numerous technical writers contact us after our project was approved to be part of the 2023 GSoD program. We organised a small application process including open-ended questions like “how would you recommend improving this doc page” and received nearly 40 expressions of interest. From those, we interviewed six to make final decisions. One stood out, QuillPusher, who we extended an offer to. We were very happy that he accepted (having multiple organisers interested) and started working with us right away. QuillPusher’s role was to work with our team to improve, or in some cases to develop from scratch, doc pages, blog posts, and small blobs highlighting accomplishments of our younger team members.

We found regular communication to be (as always) important. We set up weekly meetings and used Slack for regular communications. Furthermore, we believe that this led to an efficient process for information exchange and for addressing questions (even if answers were not always available).

Timeline

Our estimated end date was October 31st, 2023. While we have covered all the major documentation tasks originally planned for this project, we also drifted a bit from the original plan, based on documentation audit and available resources.

For example, one of the original tasks included Xeus-Clang-Repl documentation, but we also wanted to make sure to avoid duplication. Since this topic was covered by a different student under GSoC 2023, we opted to work on other time-sensitive tasks, for example, profiling our top student contributors that were soon leaving their respective programs, and capturing their knowledge for their successors in the form of developer documentation.

A few of the PRs are in review and haven't been closed as of October 31st, but QuillPusher has agreed to stay on and resolve these before the program is finally concluded.

Results

As part of our Google Season of Docs 2023 project, one of the main goals for us was to highlight the amazing work done by our various contributors in the domain of Data Science, and for these features to be understood by a wider audience through detailed technical documentation.

We had the opportunity to develop and improve user and developer facing documentation of various projects that the Compiler Research Organization has contributed to in the recent past. The rest of this section describes each of the results of our Season of Docs work.

The development of the Clang-REPL component of LLVM. Clang-Repl is an interactive C++ interpreter that allows for incremental compilation. The **Execution Results Handling** features in **Clang-REPL** that were developed for the upstream **LLVM** repository have been documented here (D156858).

The development and application of the Clad tool. Clad enables automatic differentiation (AD) for C++. It is based on LLVM compiler infrastructure and is a plugin for Clang compiler. Clad is a source code transformation and automatic differentiation tool: given C++ source code of a mathematical function, it can automatically generate C++ code for the derivative(s) of the function. Specifically this project created documentation on:

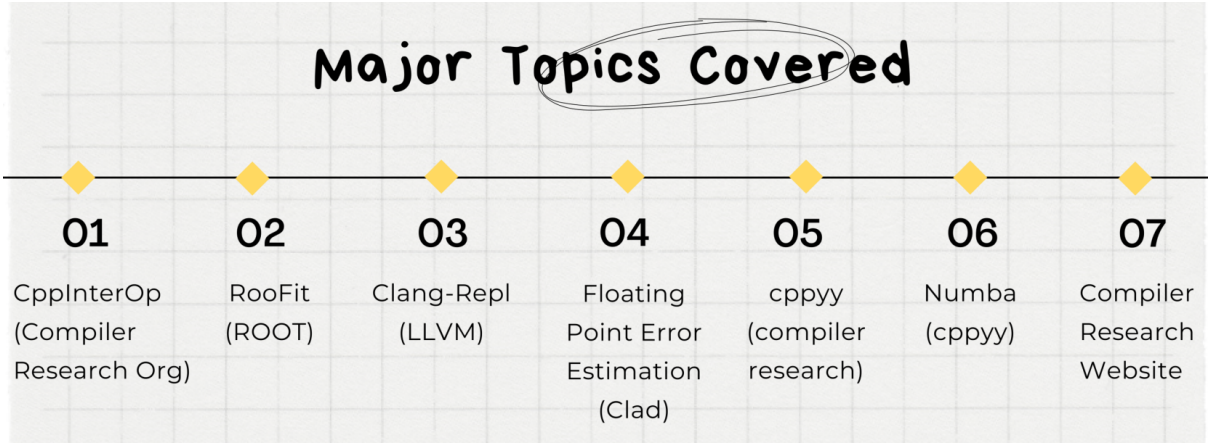
- **RooFit** enhancements based on **Clad** that were developed for the upstream **ROOT** repository. We worked with RooFit/ROOT developers to review technical information and were able to contribute to the documentation against the features contributed by our team. The main document can be found here (PR1, PR2).
- The CHEF-FP tool for estimating floating-point errors. CHEF-FP is a flexible, scalable, and easy-to-use source-code transformation tool based on Automatic Differentiation (AD) for analysing approximation errors in HPC applications. The main document can be found here (PR).

C++ and Python Interoperability: CppInterOp is a library developed to simplify and debloat the interoperability features between C++ and Python (PR1, PR2). It also helps in reducing dependencies for faster and more efficient interoperability in **cppyy** (PR).

An important link in this chain is Numba, the just-in-time compiler for Python that works best on code that uses NumPy arrays and functions, and loops. Our developers had a recent contribution to cppyy that enables Numba (when used with cppyy) to digest and JIT compile C++ code. These efforts were also documented to increase community awareness (PR).

Success Stories: One of our greatest joys is working with future scientists and innovators during their foundational years. We started a series of Success Stories to highlight student developers that were able to add significant contributions to the world of Open Source software. These are documented here (PR).

Following is an illustration of the major documentation efforts:



These were concluded based on expected timeline, except for some final reviews and unmerged PRs (listed below).

Notable PRs

- CppInterOp ReadMe: https://github.com/compiler-research/CppInterOp/pull/91
- CppInterOp Doxygen: https://github.com/compiler-research/CppInterOp/pull/85
- Success Stories: https://github.com/compiler-research/compiler-research.github.io/pull/110
- Execution Results Handling: https://reviews.llvm.org/D156858
and https://github.com/llvm/llvm-project/pull/65650
- Roofit: https://github.com/root-project/root/pull/13929
and https://github.com/root-project/root/pull/14018
- Floating Point Error Estimation: https://github.com/vgvassilev/clad/pull/648
- Numba: https://github.com/wlav/cppyy/pull/199
- CPPYY: https://github.com/compiler-research/CppInterOp/pull/160
- CR Website Setup:
https://github.com/compiler-research/compiler-research.github.io/pull/129

Metrics

The primary metrics used were around the number of distinct contributions made and towards “user” satisfaction and “developer” community acceptance. While it is too early to really judge the user metric, the other two have been met and exceeded. QuillPusher contributed content into more communities than we anticipated and had good success in getting this content accepted and integrated. In that sense, these two metrics correlate well with the desired outcomes we had. The user satisfaction metric will wait for follow-up surveys as it takes time to gather feedback.

Analysis

We would consider the project successful, on account of the major documentation contributions in upstream LLVM, ROOT, and other repositories, where our developers had previously contributed code.

Going into this project, we had a vision of what we wanted to achieve, but over several brainstorming sessions, we identified several areas that we thought needed improvement.

Due to the limited time frame that we were working with, we prioritised the tasks based on what was urgent and what was important. There’s still a wish list of items that we discovered during these discussions, these are, however, out of the scope of this GSoD project, so we’ll tackle them over time.

Summary

Our overall experience with this round of Season of Docs has been very positive. We comment on some specifics of our experience. Different from our previous experience, we were well-prepared for the recruitment process. Despite many more applications than previous (2-3x more!), the fact that we had prepared a process meant that we were ready and organised for these applications. Talking to all the leading candidates was lots of fun, and in reality we had three or four candidates that would have been excellent for our project.

The content development process itself was also a positive experience. Because our software lands in quite a number of different repositories, QuillPusher contributed documentation to numerous open source projects, matching and exceeding our expectations for the project. He was able to integrate into the “style” of existing documentation systems and improve our own sites. As things are wrapping up, it is clear that the end results are going to be extremely valuable to our project and partner websites. Results are (we believe) sustainably integrated with the codebases. Our project leaders found frequent communication, including informal means such as chat, important for their project components.