Adopting CppInterOp in cppyy

Contact Details

Name: Vipul Cariappa
Email: vipulcariappa@gmail.com
GitHub: https://github.com/Vipul-Cariappa/
Website: https://www.vipulcariappa.xyz/
Resume: https://www.vipulcariappa.xyz/assets/My-Resume.pdf
Timezone: India Standard Time (GMT+5:30)
Supervisors:
- Dr. Vassil Vassilev (Princeton University, USA), vvasilev@cern.ch
- Dr. S M Hari Krishna (Ramaiah University of Applied Sciences, India), harikrishna.cs.et@msruas.ac.in

About My Project

Abstract

cppyy is a Python library that provides fully automatic, dynamic Python-C++ bindings using the cling based C++ interpreter/incremental compiler. Cling itself is based on the LLVM toolchain. Compiler Research maintains its own fork of cppyy. The main motivation of this fork is to directly use the LLVM's clang-REPL as C/C++ interpreter/incremental compiler and runtime reflection library. This reduces the burden on the developers to maintain a separate fork of LLVM like the cling interpreter. The Compiler Research's cppyy fork's migration to using the upstream clang-REPL is an ongoing effort. The migration is incomplete, and many features are missing in the Compiler Research's fork. I aim to implement as many of these missing features as possible to complete the transition to using the clang-REPL compiler backend.
Once we get most of the high priority features working using the clang-REPL backend, I plan on implementing a multi-language Jupyter kernel on top of xeus-cpp. This multi-language kernel will support using both C++ and Python for code execution and will provide the ability for the users to seamlessly use symbols defined in each other.

The compiler research's fork of cppyy, uses LLVM clang-REPL through the CppInterOp library. CppInterOp is developed and maintained by compiler research. It is a high level abstraction of the underlying LLVM API. CppInterOp is designed to be simple and minimalistic. It is built for the purpose of language interoperability, incremental C++ compilation, and runtime C++ reflection. During this process of migration from using the cling backend to CppInterOp, we will also be extending CppInterOp's feature set.

Some of the missing features in the cppyy are;

- Use of smart pointers like `std::unique_ptr` and `std::shared_ptr`.
- Usability of Templated classes and functions.
- Lookup of anonymous fields and enums, global operators, and overloaded functions and methods
- Ability to run multi-threaded or multi-processed code.
- Cross-Exception handling

As we are adapting an existing code base to work with a different backend library, we will also need to rethink the methodology behind the optimizations that were previously built in. For example; the caching mechanism fails more than 50% of the time, which leads to increased memory usage and memory leaks, and time spent on creating new objects.

xeus-cpp is a Jupyter Kernel for C++. It uses CppInterOp library for incremental compilation and execution of code. The idea behind the multi-language kernel is to use both Python and C++ in a single notebook. The end user should be able to use symbols across languages seamlessly. The kernel would be responsible for any necessary type conversions.
Cppyy can be used for this purpose. It already has the necessary building blocks for incremental compilation and interoperability with Python. We would adapt xeus-cpp to integrate with cppyy for the Python code execution.
Cppyy can not resolve functions and classes defined in Python for use in C++, we will require a new mechanism to resolve the symbols defined in Python to be used in C++. We will also need to factor in the differences in the type systems of the two languages for a seamless end user experience.

Benefit to the Community

If we complete the migration to clang-REPL we will not be required to maintain a separate LLVM's fork, which we do through cling. We can directly use the upstream LLVM, and use all of its new features, speed improvements, and bug fixes. The current implementation of cppyy using the cling library for C++ interpretation uses lots of string manipulation. This string manipulation increases the performance overhead and affects the code readability and debuggability. While using clang-REPL, these string manipulations can be avoided, mitigating the drawbacks mentioned.
Python is the go-to language for data science and machine learning. But Python is slow because it is interpreted and dynamically typed. Whereas C++ is compiled and statically typed, C++ offers much better performance than Python. Most of the compute intensive Python libraries are written in C/C++ or Fortran. Building a multi-language Jupyter kernel will provide users with the advantages of both worlds. Easy to write Python, and performance of C++.

Tentative Timeline

I will approximately be spending 40 hours a week on this project.

- Week 1 & 2: Implement the necessary changes/features to enable the lookup of anonymous fields and enums, global operators, and overloaded functions and methods.
- Week 3 & 4: Implement the necessary features to use templated functions and classes
- Week 5 & 6: Implement the necessary feature to use C++ smart pointers. Then extend the work to include the use of other templated classes and functions from the Standard Template Library (STL).
- Week 7 & 8: Work on features related to cross-inheritance, i.e. inheriting from C++ class in Python.
- Week 9 & 10: Make cppyy and CppInterOp thread-safe to enable multi-threading and multi-processing workflows.
- Week 11 & 12: Work on xeus-cpp, implementing the multi-language kernel.

The above mentioned timeline should be considered as an estimate.

Note: Work on CppInterOp will be prioritized over the multi-language Jupyter Kernel.

Work done so far

- Fixed an issue in CppInterOp related to a missing feature; to get the include paths; through this PR.
- Implemented the ability to lookup anonymous `enum` constants (PR).
- Implemented the ability to lookup data members/fields of anonymous `struct` and `union` (PR).
- Implemented the ability to read and write to static class fields/data members (PR).

Biographical Information

I am currently pursuing my bachelor's degree in computer science and engineering in Bangalore, India. I am a two time GSoC contributor. In 2023, I contributed to GNU Octave. Octave is a programming language designed for compute intensive and scientific applications. My project was on interoperability between Octave and Python programming languages. In 2024, I contributed to LPython under the Python Software Foundation. LPython is a statically typed, compiled version of Python. I built a REPL (read-evaluate-print-loop) shell for LPython, Jupyter Kernel, and worked on interoperability with CPython.