Unlocking the Power of C++ as a Service:
Uniting Python's Usability with C++'s Performance

Vassil Vassilev, compiler-research.org

12.10.2023
Motivation

Is there a way to combine the expressiveness of Python and the power of C++ without creating a new programming language?
Leverage the exploratory programming infrastructure developed in the field of high energy physics and make it available to other scientific domains via LLVM and open source.
In-Tree Support for Incremental Compilation With Clang-Repl
Positive outcome for our LLVM community outreach. Adapting mainline LLVM infrastructure started shortly after.
Support For Incremental Compilation. Clang-Repl

Initial version of the incremental compilation infrastructure landed in LLVM and was released in LLVM 13. Gradual improvements in every release.

Since LLVM 13, approximately 30 developers have contributed in that area.
Clang-Repl Helped Upstreaming Tech. Debt

Clang-Repl provided an environment which helps explain and test the custom patches developed in the domain of High-Energy Physics (HEP). Most patches are released via LLVM17.

* During the project we have upstreamed the essential patches relevant for incremental compilation

* That lead to faster llvm upgrade cycles in HEP. Time for upgrades went down from approximately 1 year (llvm5->llvm9) to several months from (llvm9->llvm13) to several weeks (llvm13->llvm16).
Developments Related to Clang-Repl (1)

Clang-Repl drove several new developments:

- Automatic completion at the prompt improving the overall user experience (will be released in LLVM18). See F. Fu’s student talk later today.
- Implement shared memory manager for JITLink enabling efficient out-of-process execution to improve system stability (LLVM15)
- Program reoptimization. See S. Kim’s talk from yesterday.
Developments Related to Clang-Repl (2)

JITLink is a library for JIT Linking. That is a component enabling re-use of LLVM as an in-memory compiler by adding an in-memory link step to the end of the usual compiler pipeline.

- Develop Windows Support (COFF in LLVM16)
- Develop ARM64 Unix Support (Aarch64 in LLVM16)
- Develop ARM32 Unix Support based on our ARM64 infrastructure — external contribution
- Develop RISCV JIT Support (LLVM16)
- Develop PowerPC Support (ppc64 in LLVM18) — contributed by IBM/Sycomp
Interactive CUDA Support

Implemented a novel approach in interpreting CUDA codes where the PTX is passed through the virtual file system (LLVM17)

The CUDA engine in Clang-Repl helped discover issues in the mainstream CUDA support in Clang.
Automatic Language Interoperability
Interoperable, Interactive C++ in Jupyter

Crossing the language barrier is expensive.

Our Compiler-As-A-Service Approach solves that.

In [1]:
```cpp
struct S { double val = 1.; }
```

In [2]:
```python
from libInterop import std
python_vec = std.vector(S)(1)
```

In [3]:
```python
print(python_vec[0].val)
```
1

In [4]:
```python
class Derived(S):
def __init__(self):
    self.val = 0
res = Derived()
```

In [5]:
```c
__global__ void sum_array(int n, double *x, double *sum) {
    for (int i = 0; i < n; i++) *sum += x[i];
}
```
```c
// Init N=1M and x[i] = 1.f. Run kernel on 1M elements on the GPU.
sum_array<<<1, 1>>>(N, x, &res.val);
```

compiler-research.org's Compiler-As-A-Service Project Final Goal. Shown in the live demo.
Clang-Repl in Data Science With Xeus

Xeus is a protocol that enables executors to connect to the Jupyter infrastructure:

- Xeus-Clang-Repl enables incremental C++ with interoperability extensions in Jupyter by implementing the Xeus kernel protocol
Automatic Language InterOp With Python

**CPPYY** is a CPython/PyPy Extension using their C API. It offers automatic, on-demand mapping of Python to C++ concepts.

Every unsuccessful lookup can be completed by a C++ entity connected to a python class wrapper.

```
val = std.vector[int]((1,2,3))
```

While parsing we can associate each construct with a C++ entity. The approach does not require the project maintainer to bother providing static bindings.
Moving CPPYY to LLVM Orbit With CppInterOp

Provides interoperability primitives for C++ to enable crosstalk in automatic way with Python but also for D, Julia. The library allows replacing the cppyy backend with a specialized and more robust InterOp, moving it closer to LLVM orbit to allow.
Tutorials & Community Outreach

https://github.com/compiler-research/pldi-tutorials-2023
Community Outreach

- Open, Virtual Weekly Team Meetings
- Open, Virtual Monthly Meetings
  - 13 invited talks by speakers from institutions such as Apple, HZDR, QuantStack, Max-Planck, LBL, CERN and EA
- Student mentoring
  - 2 Unpaid Contributors
  - 2 CERN Interns
  - 4 IRIS-HEP Fellows
  - 15 Google Summer of Code
- 3 Technical Documentation Writers via Google Season of Docs

https://compiler-research.org/vacancies/
https://compiler-research.org/team/
https://compiler-research.org/meetings/
Compiler (C++) As A Service

Static Compiler

In-Process Compiler As A Service

More in S. Kim’s talk, yesterday
Automatic program reoptimization support in LLVM ORC JIT

Continuous Optimization
(LLVM’s OrcV2)

Ahead of Time
Just-in-Time

Binary started
Binary execution

Deferred
Produced binary

Abstract machine
Abstract user

Concrete user(s)
Target machine

develop
deploy
start
execute

optimize

LLVM Dev Meeting, Oct, 2023
Unlocking the Power of C++ as a Service, V. Vassilev, 12-Oct-2023
Future Work

The funding period is finished but we have plenty of interesting things to pursue in this area:

- Continue the open meetings policy
- Continue bug fixing and stabilizing Clang-Repl
- Continue developing tutorials
- Reach out to other scientific domains to inform their communities for the new possibilities offered by our innovative software stack!
Live Demo

https://youtu.be/be89sF0WLrc
C++: Create a C++ Struct `S``
Python: Create a wrapper class over std::vector instantiated with `S`
Python: Print the value of `S`
Python: Derive from `S`
CUDA: Perform a sum over array and record the result into res.
Demo: OpenMP Hello World

- Run OpenMP codes in Jupyter
Demo: Image Processing. Mixing Python/C++/CUDA

Use Pillow and NumPy interactively with C++ and CUDA execution

- CUDA: Apply underexposure to pixels based on a threshold value
- C++: Data conversion
- Python: Plotting
Demo: Kalman Filter. Mixing Python/C++/CUDA

Use PyYaml and Matplotlib interactively with C++ and CUDA execution

- CUDA: Compute fast matrix and vector operations
- C++: Set of efficient CUDA function abstractions
- Python: Data processing and plotting
A Note of Gratitude

This multiyear, multi person effort would not have been possible without YOU!

The compiler-research team would like to express its deepest gratitude to the various people who contributed intellectual work in the area over the years!
Thank you!
Exploratory programming with C++
CaaS. Programming Model

/// Call an interpreted function using its symbol address.
void callInterpretedFn(clang::Interpreter& interp) {
  // Declare a function to the interpreter. Make it extern “C”
  // to remove mangling from the game.
  clang::Value V;
  interp.ParseAndExecute("extern \"C\" int sq(int x) { return x * x; }\n"sq(12)\n", &V);
  printf("From JIT: square(12)=%d\n", V.getInt());
  auto SymAddr = ExitOnErr(Interp->getSymbolAddress("sq"));
  auto squarePtr = SymAddr.toPtr<int(*)(int)>();
  printf("From compiled code: sq(13)=%d\n", squarePtr(13));
}

// caas-demo.cpp
// g++ ... caas-demo.cpp; ./caas-demo
int main(int argc, const char* const* argv) {
  clang::IncrementalCompilerBuilder CB;
  CB.SetCompilerArgs({"-std=c++20"});
  auto I = Interpreter::create(std::move(CB.CreateCpp()));
  callInterpretedFn(I);
  return 0;
}
Interactive C++. Key Insights

- Incremental Compilation
- Handling errors
- Syntactic
- Semantic
- Execution of statements
- Displaying execution results
- Entity redefinition

```cpp
#include <vector>
std::vector<int> v = {1,2,3,4,5};
std::sort(v.begin(), v.end());
input_line_1:1:1: error: unexpected namespace name 'std': expected expression
std::sort(v.begin(), v.end());

std::sort(v.begin(), v.end());
std::string v = "Hello World";
```
Demo: JupyterLite

- Demonstrate Clang-Repl in browser
Broader Impact

The project developed technical and human capital in the intersection of compiler and data science. It connected domain scientists to the LLVM community via core technologies fostering synergies and collaborations with industry.

The project helped develop 27 young professionals from 11 different countries some of who went to prestigious academic and industrial institutions such as UCSD, ETH Zurich, CERN, Pittsburgh U, IIT and Qualcomm.