C++ as a service — rapid software development and dynamic interoperability with Python and beyond

Interactive C++: cling and clang-repl

Vassil Vassilev

03.11.2022

Status. Cling

- Continuing to rebase cling on top of llvm13, fixing Windows.
- Added initial version of readthedocs-based documentation

Status. Clang-Repl

- More progress on parsing statements on the global scope: D127284
- Started a clang-repl based service for Jupyter called xeus-clang-repl

The goal is to provide better stability and robustness which can later cling can reuse.

Status. InterOp

- We can execute basic cppyy workflows
- Working on simplifying CallFunc and moving it in libInterOp: PR10850

Status. Clad

- Released v0.1 and integrated in ROOT and the xeus-cling binder

Documentation

- Working on preparing blog posts for our summer internships

Upstreaming Patches

- Spreadsheet tracking the progress here.
- Total amount of upstreamed cling patches 26 out of 52 upstreamable.

CaaS Open Projects

- Open projects are tracked in our open projects page.

Next Meetings

- Monthly Meeting — 8th Dec, 1700 CET/0800 PDT
  - Tentative talk by Sunho Kim on clang-repl and orcv2 jit-link infrastructure

If you want to share your knowledge/experience with interactive C++ we can include presentations at an upcoming next meeting

GSoC 2022

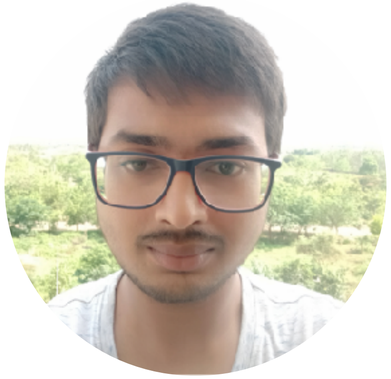
Contributors

Surya Somayyajula

IRIS-HEP Fellow, University of Wisconsin-Madison, USA

Improve Cling's packaging system: Cling Packaging Tool

(May 2022-Sep 2022)



Manish Kausik H

GSoC22, Computer Science and Engineering(Dual Degree), Indian Institute of Technology Bhubaneswar

Add Initial Integration of Clad with Enzyme

(May 2022-Sep 2022)

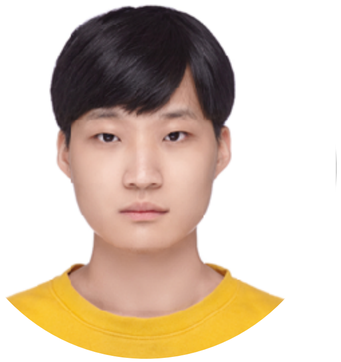
Matheus Izvekov

GSoC22, Computer Science

Preserve type sugar for member access on template specializations

(May 2022-~~Sep~~Nov 2022)

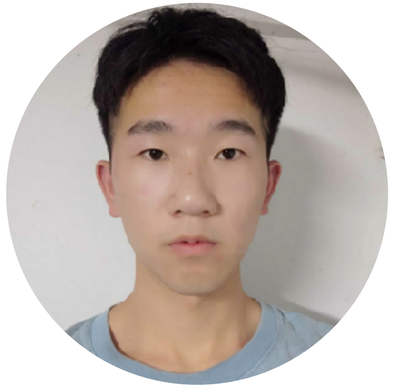
People



Sunho Kim

GSoC22, De Anza College, Cupertino, USA

Write JITLink support for a new format/architecture (ELF/AARCH64)
(May 2022-Sep 2022)



Jun Zhang

GSoC22, Anhui Normal University, WuHu, China

Optimize ROOT use of modules for large codebases
(May 2022-Sep 2022)

Anubhab Ghosh

GSoC22, Indian Institute of Information Technology, Kalyani, India

Shared Memory Based JITLink Memory Manager
(May 2022-Sep 2022)

Surya Somayyajula

IRIS-HEP Fellow, University of Wisconsin-Madison.

Improve Cling's packaging system: Cling Packaging Tool (May 2022-Sep 2022). Slides: here.

Project Objectives

- Improvements to be made
  - Fixing platform issues
    - This mostly entails fixing builds with LLVM on Linux and Mac OS
    - Debian packaging creation
    - Fixing Windows builds
  - Rewriting the CPT itself
    - A full rewrite of the CPT, fixing old features as well as adding new features, and getting rid of non-functional options
  - Rewriting documentation
    - Adding new documentation for rewrite and fixes, as well as rewriting old documentation for overriding variables
  - Fixing miscellaneous issues
    - Fixing specific software dependency issues

Rewriting the CPT

- Using a different program execution starting point
  - I added a new if name block separate from all the program functions
- Revamping the argument parser
  - I added an option to only build Cling and not package it, as users want this option
  - I added some dependent arguments so that errors would be caught before any building is done
  - I also renamed some arguments for consistency
- I added a feature to specify the number of CPU cores to use when building Cling
- Reduced global variable mutation
  - Implemented parameter passing style for a couple of global variables where possible, as most of the global variables are deeply embedded in the CPT
- Made the CPT more flake8 compliant, although almost all of the flake8 errors are due to the lines being longer than 79 characters

Manish Kausik H

GSoC22, Indian Institute of Technology Bhubaneswar

Add Initial Integration of Clad with Enzyme
(May 2022-Sep 2022). Slides: here and here. Final report. Blog Post.

Implementation Ideas

2. Reverse Mode Differentiation Code Generation

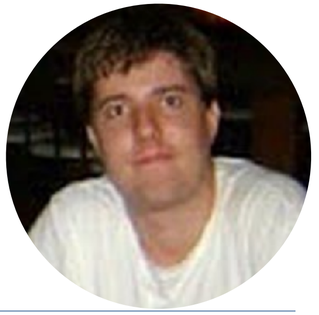
- **DiffCollector::VisitCallExpr** must set a variable in the DiffRequest Object, that states whether the user wants to use enzyme or not.
- **ReverseModeVisitior::Derive** must create a new branch for Enzyme DiffRequests, with a constant template code
- Must link the Code generated by **ReverseModeVisitor::Derive** with the **CladFunction** class (Need to explore this)
- How can **DiffCollector::VisitCallExpr** recognise the request for use of enzyme based on a template parameter? (Need to explore this)

Integrating Enzyme Reverse Mode with Clad

1. Identifying a request for using Enzyme with Clad (PR #460)
2. Integrating Enzyme as a static library in Clad (PR #466)
3. Generating code for Enzyme Reverse mode with clad (PR #486)
4. Verifying Enzyme generated derivatives with clad(PR #488)

```
clad::gradient(f) //Normal Calling convention
clad::gradient<clad::opts::use_enzyme>(f) //Calling Convention for using Enzyme within Clad
```

Matheus Izvekov



Preserve type sugar for member access on template specializations
(May 2022-~~Sep~~Nov 2022). Slides: here and here.

GSoC22

In simplest terms, with an example, we want this to work:

```
template<class T> struct foo { using type = T; };

struct Baz {};

using Bar [[gnu::aligned(64)]] = Baz;

using type = typename foo<Bar>::type;

// Clang as it stands will fail below assert
// as the foo template will only be instantiated
// with the structural part of the argument,
// which the Bar alias is not.
// So it only sees Baz, the aligned attribute is never seen.
static_assert(alignof(type) == 64);
```

Accomplishments

We submitted the RFC at https://discourse.llvm.org/t/rfc-improving-diagnostics-with-template-specialization-resugaring/64294.

- We have positive feedback, people want to see this implemented
- We got one extra volunteer for reviewing.
- We got feedback that this work might influence debug info.
- We linked to the WIP patch in phabricator.
  However, the patch is too big and we must work on splitting it up

Sunho Kim

GSoC22, De Anza College, Cupertino, USA

Write JITLink support for a new format/architecture (ELF/AARCH64).
Slides: here and here. Final report.

Issues of Old JIT Linker

- Some horrors
  - https://github.com/llvm/llvm-project/blob/main/llvm/lib/ExecutionEngine/RuntimeDyld/RuntimeDyldELF.cpp#L1217 (RuntimeDyldELF::processRelocationRef)
  - ```
    No test case: Unfortunately RuntimeDyldELF's GOT building mechanism (which
    uses a separate section for GOT entries) isn't compatible with
    RuntimeDyldChecker. The correct fix for this is to fix RuntimeDyldELF's GOT
    support (it's fundamentally broken at the moment: separate sections aren't
    guaranteed to be in range of a GOT entry load), but that's a non-trivial job.

    llvm-svn: 279182

    main
    llvmorg-15-init ... 2020.06-alpha
    ```

4

My project

- Problem: lack of platform/architecture support in JITLink to make it a viable replacement for old JIT infrastructures.

	Linux (ELF)	Mac (MachO)	Windows (COFF)
ARM64	O	O	X
X86_64	O	O	O
RISCV	O	X	X

4
    ```

Anubhab Ghosh

Shared Memory Based JITLink Memory Manager.
Slides: here. Final report.

GSoC22, Indian Institute of Information Technology, Kalyani,

The plan

- A MemoryMapper interface with implementations based on
  - Shared memory
    - When both executor and controller process share same physical memory
  - Regular memory allocation APIs
    - When the resultant code is executed in the same process
    - Useful for unit tests
  - EPC
    - Required when the executor and controller process run with different physical memory
    - Resultant code is transferred to the executor process over the EPC channel
- A JITLinkMemoryManager implementation that can use any MemoryMapper
  - It will allocate large chunks of memory using MemoryMapper and divide into smaller chunks
  - Better support for small code model by keeping everything close in memory

Design and Implementation

- `orc::MemoryMapper` interface: This is an interface to perform memory allocation, deallocation, setting memory protections etc. that handles most platform-specific operations. This abstraction allows us to decouple the transport for generated code from heap management making it simple for clients to use different transport mechanisms. (D127491)
  - `orc::InProcessMemoryMapper`: This implementation is used when running code in the same process where the JIT is running and uses `sys::Memory` API. (D127491)
  - `orc::SharedMemoryMapper`: This implementation is used when transferring code to a different executor process and uses POSIX or Win32 shared memory APIs. (D128544)
- `orc::MapperJITLinkMemoryManager`: This class implements the `jitlink::JITLinkMemoryManager` interface and handles all allocations within a slab. (D130392)
- Memory coalescing to join two consecutive free ranges and reuse them. (D131831)
- `llvm-jitlink` tool integration:
  - `MapperJITLinkMemoryManager` with an `InProcessMemoryMapper` is used by default when executing the code in the same process as the JIT. (D132315)
  - `MapperJITLinkMemoryManager` with a `SharedMemoryMapper` can be optionally used when `--use-shared-memory` is passed. (D132369)

Thank you!