Improve automatic differentiation of object-oriented paradigms using Clad

Petro Zarytskyi
Julius-Maximilians-Universität, Würzburg

Mentors:
Vassil Vassilev, David Lange
June, 2025

About myself

Email: petro.zarytskyi@gmail.com
Github: PetroZarytskyi
Location: Germany
Citizenship: Ukraine
Timezone: CET, UTC+1 (Berlin)

I am Petro Zarytskyi, and I'm a Ukrainian Mathematics student at the University of Würzburg, Germany. I've been programming in C++ for the past 4 years, and I'm especially interested in compiler research and static analysis. I have a lot of experience contributing to the Clad project and working closely with the team. I'm very familiar with Clang and LLVM.

Related Prior Experience

- Contributing to Clad
- Technical student at CERN
- IRIS-HEP Fellowship, CERN

Project Overview

Clang is a powerful compiler for C, C++, and Objective-C. Clad is a Clang plugin that enables automatic differentiation (AD) for C++ mathematical functions. It modifies the abstract syntax tree using LLVM's compiler capabilities to generate derivative computing code. The great advantage of Clad is the ability to be integrated into existing codebases, as it does not require any code modifications. Clad supports both forward and reverse mode differentiation. The reverse mode is efficient for computations of derivatives with respect to multiple outputs, which makes Clad well-suited for Machine Learning, inverse problems, and other applications involving backpropagation.

Automatic differentiation in the reverse mode involves two passes over the function: the forward pass to compute and store all intermediate variable values, and the reverse pass to compute the derivatives. Clad supports storing intermediate values between all basic operations and data-flow. However, Clad does not fully support storing function call arguments, only trivially copyable types are supported. While Clad can cover many use cases, this limitation stops it from fully supporting C-style arrays and non-copyable types. This poses a major constraint on using Clad in Object-Oriented Programming (OOP) as it means that non-copiable types like unique pointers cannot be modified with method calls.

Implementation Details

Sometimes Clad needs to modify functions to use them in the forward sweep of the reverse mode. This functionality is used to properly handle memory allocation of adjoint objects, which is necessary when working with object or pointer types. The plan is to modify the existing reverse sweep functions generated by Clad and make them capable of storing intermediate values.

The main objective during the project will be to find the best generic solution. However, with memory handling in C++, it is not always possible to determine which expressions are modified at compile-time. This means that the general solution might involve tracking memory locations. Working with raw memory is often inefficient and results in suboptimal code. This brings the necessity to work on a simplified solution to cover a big portion of use cases while producing efficient and user-friendly code.

There is a type of analysis in automatic differentiation called To-Be-Recorded (TBR) analysis. Its purpose is to determine which expressions need to be stored in the forward sweep. It is already partly implemented in Clad. Currently, it only operates in separate functions and is unable to efficiently handle nested function calls, making it unsuitable in this context. The other bottleneck TBR poses on reliable usage in this case is the limited support for pointers: pointer reassignments and address operators are not supported. Therefore, the next goal in the project will be to enhance TBR analysis and enable it to find cases where memory is handled in a predictable way, so that Clad can generate more optimal and concise code. Having this system successfully implemented will not only add support for storing non-copyable types but will also make storing copyable structures more efficient because only specific parts of them will need to be stored.

Finally, the stretch goal of the project will be to investigate how often the fallback from optimal behaviour occurs by studying use cases. Then, if the difference is significant, see whether it's possible to make the system more flexible and limit the amount of suboptimal code.

Proposed Timeline

Community Bonding Period:
Research the topic by comparing Clad to other AD tools and investigating existing use cases and bug reproducers. Meet with mentors to discuss approaches to the project.

Week 1:
Determine pros and cons of different approaches to modify reverse sweep functions, i.e., introducing global tapes and extending the functions' signature with tape parameters. Consider different solutions to distinguish function local variables from the ones that need to be restored outside of it in the reverse pass, e.g., tracking memory locations. **Deliverable:** Prepare a report for the Clad team listing missing features and proposing solutions with pros and cons.

Weeks 2-3:
Implementing the previously chosen solution. Adding tests. **Deliverable:** Having a pull request that succeeds all tests.

Week 4:
Buffer week.

Week 5:
Running benchmarks to make sure the performance stays high. **Deliverable:** Prepare a report on the performance of the solution in benchmarks and Clad use cases.

Week 6:
Study the use cases to determine the common characteristics of where optimizations can be made. Develop a solution to improve the existing

To-Be-Recorded analysis implementation, including nested function call handling and improving pointer/reference support. **Deliverable:** Having a report on the planned solution.

Midterm Evaluations

Weeks 7-9:
Improve the To-Be-Recorded analysis and attach the new features to the AST generating engine. Write tests for the new capabilities. **Deliverable:** Having a pull request that succeeds all tests.

Week 10:
Run benchmarks and compare the newly To-Be-Recorded analysis system with the fallback one implemented earlier. Also, run benchmarks to see if there is an improvement in performance in existing Clad use cases. **Deliverable:** A report listing all mentioned performance comparisons.

Week 11:
Buffer week.

Week 12:
Investigate how often the fallback to suboptimal behavior occurs in use cases. If the room for optimization turns out to be significant, attempt to make the system more flexible.

Week 13:
Improve documentation. Prepare the final presentation. **Deliverable:** Present the work, including benchmarks and use cases. Point out possible next steps or issues that need to be addressed in the future.

Candidate’s Other Commitments

My only other commitment is going to be the studies at the University of Würzburg until June. The final exams will take place in early July. This is not going to pose a limitation on my availability throughout the project timeline, as attendance is not mandatory. Moreover, this semester, I have deliberately picked non-time-consuming subjects so that I can focus more on GSoC.