Google Summer of Code

PROPOSAL

Add support for functor objects in clad

By
Parth Arora

Table of Contents

1. Abstract

Differentiation support for functions is available in clad. But support for direct differentiation of functors and lambda expressions is missing. Many computations are modelled using functors and functors and lambda expressions are becoming more and more relevant in modern C++. Thus, there's a growing need for support for directly differentiating them in clad.

Clad is very fast and efficient but it currently lacks support for various C++ statements, syntax and the ability to differentiate non top-level declarations. These things are required to make clad more robust and convenient to use.

This proposal aims to add support for directly differentiating functors and lambda expressions and increase clad coverage and overcome its shortcomings by fixing various existing issues and adding support for various currently unsupported syntax.

2. Deliverables

- Add support for differentiating functors and lambda expressions using *clad::differentiate* and *clad::gradient*

- Add support for differentiating member functions using *clad::hessian* and *clad::jacobian*

- Add support for differentiating functors and lambda expressions using *clad::hessian* and *clad::jacobian*

- Fix various existing issues and improve clad coverage and overcome its shortcomings by adding support for currently unsupported syntax

- Proper documentation and tests for the above-mentioned components.

3. Project Details

The following list briefly explains and elucidates how the goals (deliverables) of this proposal will be achieved.

3.1 Add support for differentiating functors and lambda expressions using clad::differentiate and clad::gradient

For differentiating functors and lambda expressions, clad needs to do 2 things:

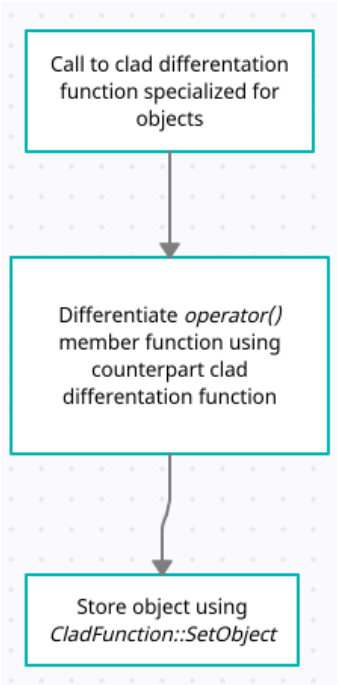
- Implicitly differentiate *operator()* member function whenever an object is passed
- Remember the passed object so that users do not have to explicitly pass it for future *CladFunction::execute* calls. CladFunction class is an ideal place to store the passed object.

If we can differentiate declarations that are not top-level declarations then this will be a trivial task. As we can do the mentioned 2 things in a different, newly created clad differentiation functions specialized for objects. But as of now, clad only supports differentiating top-level declarations.

We can add support for differentiation of all declarations (this can be very complicated), or just specifically allow differentiation of all declarations in the clad functions specialized for objects, this can be comparatively easily done.

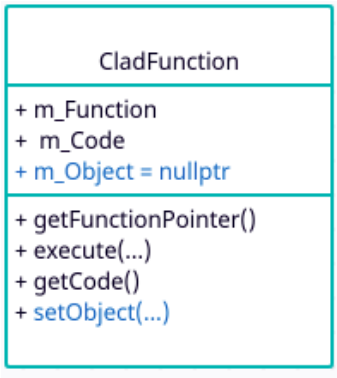
We will need to perform some AST transformations near the call site and modify *DiffPlanner* class to allow non top-level declaration differentiation in clad functions specialized for objects using this idea.

The diagram below shows the General workflow for clad differentiation functions specialized for objects :



All clad differentiation functions return an object of type *CladFunction* which provides information about and interface to the derived function as well as saves pointer to the passed object. If the user differentiates function, i.e. there is no functor object, then *m_Object* (pointer to functor) will have a default value of *nullptr*.

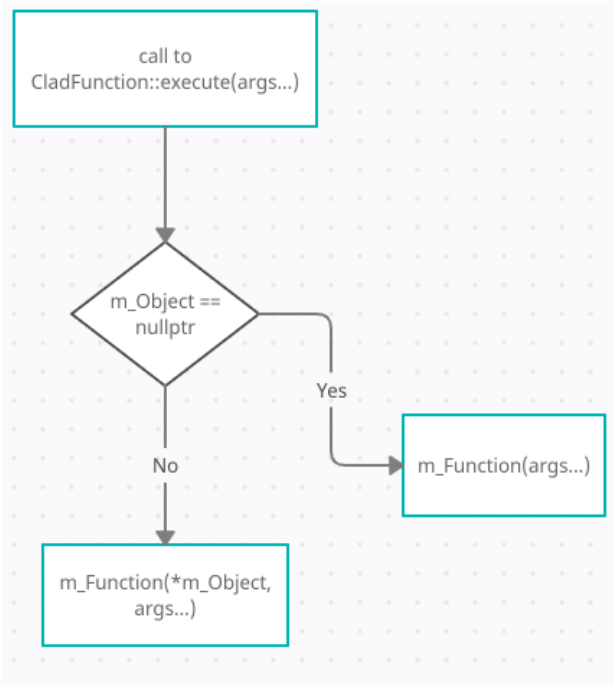
Thus, *CladFunction* class will need to be modified to accommodate these changes,



Attributes in blue need to be added

Now, *CladFunction::execute()* can check if we are differentiating a functor or function, and can thus make calls to the underlying derived function accordingly.

Diagram below shows general workflow for *CladFunction::execute()*



Here *m_Function* is a pointer to the actual derived function

We can also use just AST transformations rather than adding support for differentiating non top-level declarations to achieve this. To achieve this just using AST Transformations, we need to implicitly transform all codes like **1)** into code like **2)**

1)

```
Experiment E;
auto d_E = clad::differentiate(E, "i");
```

2)

```
Experiment E;
auto d_E = clad::differentiate(&(decltype(E)::operator()), "i");
d_E.setObject(E);
```

All these ideas will be studied and compared thoroughly, and the best approach will be selected for the implementation.

3.2 Add support for differentiating member functions using *clad::hessian* and *clad::jacobian*

As of now, *clad::hessian* and *clad::jacobian* do not support differentiation of member functions. To differentiate functor objects and lambda expressions, we first need to be able to differentiate member functions.

For this, I will need to work on and modify *HessianModeVisitor* and *JacobianModeVisitor* in the *DerivativeBuilder.cpp* file to add support for differentiating member functions.

I need to research in more detail for its implementation.

3.3 Add support for differentiating functors and lambda expressions using *clad::hessian* and *clad::jacobian*

The process to achieve this goal is very similar to the process of achieving goal 1.

3.4 Fix various existing issues and improve clad coverage by adding support for currently unsupported syntax.

I will also solve existing issues including but not limited to:

#10	Implement differentiation of templated functions.
#168	Support of differentiation of multi-arg calls in the forward mode
#128	Fix potential issues with side effects in conditional operators in forward mode
#119	Improve diagnostic messages for nested calls differentiation

#116	Fix nested function calls in the forward mode
#91	Emit hard error when differentiation visits unsupported statement
#210	function qualifiers of derived functions in reverse differentiation are inconsistent with function qualifiers of derived functions in forward differentiation

I will also add support for currently unsupported syntax in functions to be differentiated including but not limited to:

- Range-based loop
- *While* loops
- Switch statements

4. Schedule of Deliverables

- There are 3 major milestones:
    1) Add support for differentiating functors and lambda expressions.
    2) Add support for differentiating member functions in *clad::hessian* and *clad::jacobian*
    3) Fix existing issues and improve clad coverage by adding support for currently unsupported syntax

- If more things come up or plans change, they can also be incorporated.

- Few days before the evaluation period have been left free for completing any remaining work (if any). This provides sufficient buffer-time for making sure that the timeline is followed.

Date	Work
April 14 - May 17	I will study the codebase of **clad** properly so that I'm able to complete everything I have proposed smoothly
Community Bonding period begins	
May 17 - June 7	During this time I will get in touch with my mentors and will discuss and design the basic layouts for the project
Community Bonding period ends	
June 7 - June 17	Add support for differentiating functor objects in *clad::differentiate*
June 17 - June 27	Fix existing bug of differentiating member functions in *clad::gradient* and add support for differentiating functors in *clad::gradient*
June 23 - July 3	Add support for differentiating member functions in *clad::hessian* and *clad::jacobian*
July 3 - July 10	Add support for differentiating functor objects in *clad::hessian* and *clad::jacobian*
July 10 - August 1	Fix existing issues listed in section 3.4
August 1 - August 16	Add support for currently unsupported syntax listed in section 3.4
August 16 - August 23	Buffer-time. Test and submit final code and project summaries. Prepare documentations and remove any bugs.

5. Obligations

I will be available to work full time (a minimum of 40 hours per week) during the GSoC period and have no other personal or professional commitments during this period.

6. General Notes

I believe that communication is a vital aspect of GSoC and to ensure that the status of the project is communicated properly, I will be undertaking the following steps:

- I will publish a blog every week detailing:
    - Tasks completed in that week.
    - Hurdles faced to complete the tasks.
    - How I overcame the hurdles.

- I will contact mentors daily on gmail to update them how the work is progressing.

7. About Me

7.1 Personal Information and Contact Details

Name: Parth Arora

Email: partharora99160808@gmail.com

Github: parth-07

Linkedin: parth-r07

Resume: link

Phone number: +91 9354826906

Time Zone: +5:30 GMT

University Name: Guru Gobind Singh Indraprastha University, New Delhi, India

Current year: 3rd year

7.2 Why Me

I am an enthusiastic person who is always motivated to learn new things. I am proficient in C++ and use it almost every day. I am also proficient in C, python and javascript. My interest lies in contributing to the field of science and technology by helping to develop libraries, tools and framework that can help the scientific community with their research. I am also very fascinated by the core idea used in clad of using AST to differentiate functions at compile time, and thus want to contribute and work on clad.

I regularly participate in many competitive programming contests and have secured good ranks in them. I am well versed in data structures and algorithms and always enthusiastic to learn about new techniques to make the code more efficient while still being easy to maintain and understand.

I care a lot about the efficiency, readability and maintainability of the code that I write. I have made many projects in web development and scripting in the past. I have also made an efficient generic header-only template library for data structures and algorithms in C++ that are commonly used in competitive programming. (Repository link: https://github.com/parth-07/ds-and-algos)

I've been working with clad for around a month now and I am very much familiar with it's codebase. I will be very excited to contribute much more if given the opportunity to. I've raised following issues in this time period:

- #198 : Clad is not considering truncation of variables inside function for differentiation
- #199: Compile time error while differentiating constant member functions
- #209: Differentiating variadic functions in reverse, hessian and jacobian modes

- #210: function qualifiers of derived functions in reverse differentiation is inconsistent with function qualifiers of derived functions in forward differentiation

- #211: Error when using function pointer variable to pass function address in differentiation calls

I have implemented and fixed the following issues:

- #200: Support member functions with qualifiers in differentiation calls. This PR fixes #199, and have been merged into the master branch.

- #212: Add support for passing fns using fn pointers in differentiation calls. This PR fixes #211, and this PR has not yet been merged into the master branch.

I have started working on more issues and will be sending more PRs in the coming weeks. I'll continue sending PR's and further improve clad long after GSoC.