Add Support for differentiating functor objects in Clad

Student: Parth Arora  Organisation: CERN-HSF  Mentors: Vassil Vassilev, David Lange
Project repository: Clad  Project proposal: Proposal link

Table of Contents

- Clad overview
- Project overview
- Project Deliverables
- Project Results
  - Added support for differentiating functor objects and lambda expressions.
  - Extend clad by adding support for differentiating more C++ syntax and constructs.
  - Made clad more robust by adding an automatic testing framework and fixing various existing issues.
  - Added support for building clad Doxygen documentation.
- Contributions
  - Pull requests
  - Issues opened
- Conclusion
- Acknowledgement

Clad overview

Clad is an automatic differentiation clang plugin for C++. It can differentiate mathematical functions represented as C++ functions. For each function that is to be differentiated, Clad creates another function that computes its derivative.
Clad analyses the abstract syntax tree (AST) produced by the clang compiler to differentiate the functions using automatic differentiation (AD). Automatic differentiation avoids all the usual disadvantages of symbolic and numerical differentiation. Clad provides an interface to execute the differentiated function and obtain its source code. A simple example to demonstrate clad usage:

```cpp
double fn(double i, double j) {
    return i*j;
}

int main() {
    // Create a function that computes derivative of 'fn' w.r.t 'i'
    auto d_fn = clad::differentiate(fn, "i");

    // Computes derivative of 'fn' w.r.t 'i' when (i, j) = (3, 4)
    double res = d_fn.execute(3, 4); // res is equal to 4

    // 'getCode' returns string representation of the generated derived function.
    const char* derivative_code = d_fn.getCode();
}
```

**Project overview**

Many computations are modelled using functor objects and lambda expressions. These constructs are becoming more and more popular and relevant in modern C++. This project adds support to differentiate functor objects and lambda expressions in Clad.

This project also focuses on general improvements to Clad by adding support for differentiating more statements and making Clad more robust by adding support for additional testing and fixing various existing bugs.

All the contributions are made in the project main repository: [Clad](https://github.com/clad)

**Project Deliverables**

- Add support for differentiating functor objects and lambdas in both forward and reverse mode automatic differentiation in Clad.
- Extend Clad by adding support for differentiating more C++ syntax and constructs.
- Make Clad more robust by adding an automatic testing framework and fixing various existing issues.
- Add support for building Clad Doxygen documentation.

**Project Results**

**Added support for differentiating functor objects and lambda expressions.**

Now, Clad supports differentiating functor objects and lambda expressions in both forward and reverse mode automatic differentiation.

```cpp
// An object of class type that has defined call operator ('operator()')
class Experiment;
```
In brief, the core idea used for the differentiation of functor objects and lambda expressions is as follows:

- Implicitly differentiate the overloaded call operator defined by the functor (or lambda expression) type.
- Store the address of the functor object in the resulting CladFunction object.
  (CladFunction class stores all the necessary information about the derived function to conveniently access, execute and debug them.)
- Now, CladFunction automatically takes the stored functor object address to execute the derived function. Thus, there's no need to explicitly pass the functor object (or lambda expression) while executing the derived function through the CladFunction object.

Some of the challenges faced during its implementation:

- The Implementation of differentiating functors and lambda expressions depended on the differentiation of member functions using clad differentiation functions.

Clad differentiation functions are as follows:

- `clad::differentiate` - differentiates function with respect to a single parameter using forward mode automatic differentiation.
- `clad::gradient` - differentiates function with respect to multiple parameters using reverse mode automatic differentiation.
- `clad::hessian` - compute hessian matrix of a function with respect to multiple parameters using both forward and reverse mode automatic differentiation.
- `clad::jacobian` - compute jacobian matrix of a function with respect to multiple parameters using reverse mode automatic differentiation.

Differentiation of member functions using reverse mode differentiation had a bug that used to cause the resulting program to crash at runtime. The cause of the bug was inconsistencies in the clang Abstract Syntax Tree in the updated call expression site. It took us quite a while to solve this bug in an optimal way. It was fixed in PR #252.
And also, in the reverse mode, the derived member functions did not preserve original member function qualifiers. This was essential to resolve to reduce unnecessary restrictions while differentiating CVR qualified functor objects. It was fixed in PR #249.

clad::hessian did not support differentiating member functions before. Hence, the support for differentiating member functions in clad::hessian was added in PR #250.

- Obtaining the correct CVR qualified functor type while allowing functor objects to be passed both by pointers and references was interesting to implement, and had its fair share of complexities.

Extend clad by adding support for differentiating more C++ syntax and constructs.

Added support for differentiating while and do-while statements in both the forward and the reverse mode automatic differentiation.

It was thought-provoking and exciting to handle all the corner cases associated with applying automatic differentiation to while and do-while statements.

With this added support, clad can now differentiate, in both the forward and the reverse mode automatic differentiation, functions containing while and do-while statements.

For example,

```cpp
while(int index = counter) {
    res += i;
    counter-=1;
}
```

In forward differentiation mode, while statement displayed above gets differentiated to,

```cpp
while (int index = counter)
{
    int _d_index = _d_counter;
    _d_res += _d_i;
    res += i;
    _d_counter -= 0;
    counter -= 1;
}
```

Whereas in reverse differentiation mode, same while statement gets differentiated to,

```cpp
// forward pass
_t0 = 0;
while (int index = counter)
{
    _t0++; // forward pass
    res += i;
    counter -= 1;
```
// reverse pass
while (_t0)
{
{
{
    int _r_d1 = _d_counter;
    _d_counter += _r_d1;
    _d_counter -= _r_d1;
}
{
    double _r_d0 = _d_res;
    _d_res += _r_d0;
    *_d_i += _r_d0;
    _d_res -= _r_d0;
}
}
_t0--;
{
    _d_counter += _d_index;
    _d_index = 0;
}
}

Added support for differentiating switch statement in the forward mode automatic differentiation.

It was fascinating to handle switch statements because of the interesting scope rules associated with the case labels, C++17 switch init statements and the sudden change in the flow of control depending on the switch condition.

With this added support, clad can now differentiate functions containing switch statements in the forward mode differentiation.

For example,

```c
switch(int choice = index) {
    case 0:
        res += i;
        break;
    default:
        res += j;
}
```

in forward differentiation mode, switch statement displayed above gets differentiated to,

```c
{
    int _d_choice = _d_index;
    switch (int choice = index)
    {
        case 0:
        {
```
Made clad more robust by adding an automatic testing framework and fixing various existing issues.

Automatic testing of the reverse mode differentiation using the forward mode differentiation

To ensure clad is producing consistent derivative results using both forward and reverse mode automatic differentiation, now we can optionally enable additional testing of derivatives produced by the reverse mode using the forward mode automatic differentiation. This optional testing can be enabled by running clad with `-fenable-reverse-mode-testing` compile-time flag.

The primary goal of the automatic testing is to make clad more robust by making it easier to find any inconsistencies and incorrect derivatives produced in the forward and reverse mode AD.

The automatic testing modifies the computed derived gradient function as follows:

```cpp
double fn_grad(double i, double j) {
    // make copy of all the arguments.
    // These will be used to call the forward mode differentiated functions.
    double _p_i = i;
    double _p_j = j;
    ...
    // usual code to calculate the function gradient
    ...
    // Verify derivative w.r.t 'i' is equal using both reverse and forward mode.
    clad::
        VerifyResult("_d_i, fn_darg0(_p_i, _p_j)", /*assertMessage=*/
            "Inconsistent differentiation result with respect to the "
            "parameter 'i' in forward and reverse differentiation mode",
            "FileName.cpp", "fn_grad");
    // Verify derivative w.r.t 'j' is equal using both reverse and forward mode.
    clad::
        VerifyResult("_d_j, fn_darg1(_p_i, _p_j)", /*assertMessage=*/
            "Inconsistent differentiation result with respect to the "
            "parameter 'j' in forward and reverse differentiation mode",
            "FileName.cpp", "fn_grad");
}
```
clad::VerifyResult verifies that both the derivative values are equal, if the verification fails, then it prints an "Assertion failed" message and aborts the program.

Pull request for this is currently in review as of 20th August and is planned to be merged soon.

Fix various existing issues
Accuracy and robustness are really important for a mathematical library like clad. I spent good amount of time fixing existing bugs in the GSoC second coding phase.

I fixed the following issues:

<table>
<thead>
<tr>
<th>Issue link</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#253</td>
<td>Support reference variables in reverse mode differentiation</td>
</tr>
<tr>
<td>#265</td>
<td>Reverse mode do not create derived variables for all parameters</td>
</tr>
<tr>
<td>#277</td>
<td>Derived variables of variables defined in loop are not reset to 0 at each iteration in reverse mode</td>
</tr>
<tr>
<td>#292</td>
<td>Gradient overloaded function does not do perfect forwarding for the <code>this</code> object</td>
</tr>
</tbody>
</table>

Added support for building Clad Doxygen documentation.
I modified Doxygen configuration file, added support for building Clad Doxygen documentation using CMake and added configuration script for hosting the documentation on ReadTheDocs.

Two new cmake flags were added for the purpose of creating documentation.

- CLAD_INCLUDE_DOCS: Generate targets of all the enabled documentation tools
- CLAD_ENABLE_DOXYGEN: Enables the generation of browsable HTML documentation using doxygen. Defaults to OFF.

Clad Doxygen documentation can be found here

Contributions
All the contributions are made in the project main repository: Clad

Pull requests

<table>
<thead>
<tr>
<th>PR link</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#235</td>
<td>Add support for building doxygen documentation using cmake</td>
</tr>
<tr>
<td>#240</td>
<td>Add support for differentiating functors in forward mode</td>
</tr>
<tr>
<td>#243</td>
<td>Remove dependency of how the fn is passed to clad diff functions</td>
</tr>
<tr>
<td>#249</td>
<td>Preserve original member function qualifiers in the derived function</td>
</tr>
<tr>
<td>#250</td>
<td>Modify clad::hessian to support differentiating member functions.</td>
</tr>
</tbody>
</table>
Modify clad diff fns signature to have separate arg for derived fn
Add support for reference variables in reverse mode
Add support for computing gradient of functors using clad::gradient
Add support for differentiating functors using clad::hessian
Add support for differentiating functors using clad::jacobian
Fix building of nested name specifiers in getArg function
Modify code to use effective fn name to create derived fn identifiers
Create derived variables for parameters which are not independent
Add support for differentiating switch statement in forward mode
Add support for differentiating while loops in forward mode
Modify doxygen configuration
Reset derived variables of loops local variables to 0 at each iteration
Add tests and demo for differentiating template functors
Add support for differentiating while and do-while stmts in reverse mode
Update code to fix clad build warnings
Add support for diff mem variables while diff functors in forw mode
Add support for automatic testing of reverse mode using forward mode
Modify BuildCallToMemFn to do perfect forwarding of *this object

All my contributions to clad can be found here

Some of the PRs are currently in review as of 20th August and are planned to be merged soon.

Issues opened
The whole list of issues created by me during the GSoC time period can be found here

Conclusion
I was able to complete all of the goals that were decided for the project. I want to thank my mentors for helping me in making this possible. I will continue contributing to the project since it aligns with my area of interest, and there are so many more exciting things that I want to implement in Clad. We will make Clad the best automatic differentiation tool for C++ :).

Overall, participating in Google Summer of Code was a wonderful experience for me. The major reason for the great experience is the amazing mentoring by my mentors. I will cherish this experience forever. While working on the project, I improved my knowledge of clang and llvm, learned a lot about open-source culture, writing good quality codes, working in a team environment and managing my time better.
Acknowledgement

I am incredibly grateful to my mentors, Vassil Vassilev and David Lange, for their constant guidance, support, reviews and help. I have learned so much from them this summer.

I am also very thankful to Google and CERN-HSF for providing me the opportunity to work on this amazing project, which helped me learn a lot in such a short period of time.