Add support for consteval and constexpr functions in clad

Mihail Mihov
Email: mihovmihailp@gmail.com
GitHub: MihailMihov
Phone number: +359 889692277
April 2, 2024

Mentors:
Vassil Vassilev
Vaibhav Thakkar

Synopsis

In mathematics and computer algebra, automatic differentiation (AD) is a set of techniques to numerically evaluate the derivative of a function specified by a computer program. Automatic differentiation is an alternative technique to Symbolic differentiation and Numerical differentiation (the method of finite differences). Clad is based on Clang which provides the necessary facilities for code transformation. The AD library can differentiate non-trivial functions, to find a partial derivative for trivial cases and has good unit test coverage. Newer C++ versions provide the constexpr and consteval specifiers, but currently Clad does not abide by them. The aim of this proposal is to ensure that Clad generated derivative functions follow the semantics of the original functions.

Benefit to the community

C++ extensively uses constexpr and consteval to enable computation at compile time in the compiler frontend. This compile time programming is used to form idioms that follow the C++ language design principles to enable zero-cost abstractions. These techniques are quite well demonstrated in libraries such as STL and boost. The objective of supporting constexpr and consteval is to enable differentiable programming in the standard library. The high-level deliverable is to demonstrate that automatic differentiation can work in the compiler frontend.
during a constant evaluation context. A research aspect of the work is whether we can preserve the constant evaluation properties in the generated code.

Objectives

1. Implement forward mode differentiation for functions marked with consteval and constexpr.
2. Extend support to include reverse mode differentiation.
3. Add unit tests that include consteval and constexpr functions.
4. Write documentation and tutorials for using compile-time differentiation.

Expected Outcomes

By the end of this project, Clad will be able to differentiate functions marked with consteval and constexpr. This will be accompanied by thorough testing, documentation, and tutorials. The following code should build and behave as the user would expect:

```c++
#include <cstdio>
#include "clad/Differentiator/Differentiator.h"
constexpr double sq(double x) { return x*x; }
consteval double fn(double x, double y, double z) {
    double res = sq(x) + sq(y) + sq(z);
    return res;
}
int main() {
    auto d_fn = clad::gradient(fn);
    double dx = 0, dy = 0, dz = 0;
    d_fn.execute(3, 4, 5, &dx, &dy, &dz);
    printf("Gradient vector: [%-.2f, %-.2f, %-.2f]", dx, dy, dz);
    return 0;
}
```

Overview

Constexpr was an identifier introduced in C++11, which specifies that the value of a certain variable or function can appear in a constant expression. A constant expression can be evaluated at compile time and it has the added benefit that it can be used in some specific contexts that require constant expressions (template arguments, array sizes, etc.).

In C++20 consteval was introduced which declared that every evaluated call to such function must produce a constant expression. This is useful for functions that must never be evaluated at runtime.

Supporting these constructs in Clad will increase the percentage of functions that Clad can differentiate and will allow projects based on newer C++ language versions to make use of automatic differentiation using Clad.
The way that Clad works is that the user either uses clad::differentiate and specifies the variable w.r.t which to differentiate the function (forward mode) or uses clad::gradient (reverse mode) which by default differentiates w.r.t every input variable. Both of these functions return a clad::CladFunction, which contains methods to execute the original function, but that currently doesn't support any compile-time evaluation.

The first step in this project will be deciding on a new interface for CladFunction to make it constexpr-aware. It likely won't be possible to differentiate all consteval functions at compile time, so the users will somehow have to be alerted of these limitations.

Afterwards clad::differentiate and clad::gradient will need to be changed to use the new constexpr/consteval functionality in CladFunction.

**Implementation**

**Task 1: Support in Forward Mode**

Objective: Enable forward mode differentiation for constexpr and consteval functions.

Approach: Modify code generation to apply these specifiers to generated derivative functions.

**Task 2: Support in Reverse Mode**

Objective: Implement reverse mode differentiation for constexpr and consteval functions.

Approach: Ensure reverse mode respects the constexpr and consteval specifiers in derivative code.

**Task 3: Extend Unit Testing**

Objective: Ensure reliability of differentiation for compile-time evaluated functions.

Approach: Develop a set of unit tests for various compile-time scenarios.

**Task 4: Documentation and Tutorials**

Objective: Provide users with instructions to utilize compile-time differentiation.


**Candidate Background**

I believe that I’m capable of completing this project as I have experience both in working with C++ and also working with language parsers. I've worked on other compiler-related projects in the Rust language. In one of the projects I had to parse some parts of the AST to add code assists to the language server. This should have some overlap with what I’ll be doing in this project, as Clad works by parsing the AST generated by Clang. I have also already started solving some simpler issues in Clad, which I have detailed below.
Related Work

In order to familiarize myself with the project I have already solved an issue and have two merged pull requests.

- **PR #838**, where I fixed tests which were failing on 32-bit systems, due to size_t being unsigned int instead of unsigned long.
- **PR #840**, where I added more developer documentation, detailing how to use systemd-nspawn in order to make more reproducible local development environments.

Timeline

<table>
<thead>
<tr>
<th>Community Bonding Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>May 1 - May 26</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Coding period begins</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Week 1</strong> 03.06.2024-09.06.2024</td>
</tr>
<tr>
<td><strong>Deliverable</strong>: CladFunction can be used in constexpr contexts.</td>
</tr>
<tr>
<td><strong>Week 2</strong> 10.06.2024-23.06.2024</td>
</tr>
<tr>
<td><strong>Deliverable 1</strong>: clad::differentiate working on functions marked as consteval.</td>
</tr>
<tr>
<td><strong>Deliverable 2</strong>: Unit tests ensuring this functionality</td>
</tr>
</tbody>
</table>
| Week 5 | 24.06.2024-30.07.2024 | Extend support to cover more functions marked with constexpr.  
**Deliverable:** clad::differentiate working most functions from the cmath header file from STL. clad::gradient works on 50% of the functions from the cmath header file. |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Week 6</td>
<td>01.07.2024-07.07.2024</td>
<td>Buffer week and preparation for midterm evaluation.</td>
</tr>
<tr>
<td><strong>Midterm Evaluations</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Week 8 | 12.07.2024-21.07.2024 | Add support for generating consteval clad::CladFunction's in forward mode. Decide on the interface of clad::gradient. The new clad::CladFunction functionality from the former task should work here too.  
**Deliverable:** User-facing interface along with examples that should work in the end. |
| Week 9 | 22.07.2024-28.07.2024 | Research in which cases it will be possible for the generated code to also be compile time evaluated. Also figure out a way to make the user aware whether the clad generated functions are immediate.  
**Deliverable:** Technical report and a draft implementation |
| Week 10 | 29.07.2024-04.08.2024 | Decide whether the interface of clad::differentiate should remain unchanged for constexpr/consteval and evaluate any alternatives if available. |
| Week 11 | 05.08.2024-11.08.2024 | Write a blog post about compile-time programming and source transformation AD with Clad |
| Week 12 | 12.08.2024-18.08.2024 | Buffer Week |
| Week 13-14 | 19.08.2024-01.09.2024 | Extended testing, developing documentation, presenting the work.  
**Deliverable:** test cases, code examples to demonstrate the use of immediate functions; blog post about the achieved results; presentation at the compiler-research.org team meeting. |

**Contact Information**

- Email: mihovmihailp@gmail.com  
- GitHub: MihailMihov
Other Commitments

I won't have any other commitments during the standard Google Summer of Code coding period. I will be starting my first year at university during the first half of September, so that might reduce my availability for a week or two if I do proceed into the extended period, but that likely won't be necessary.