


Consolidate and advance the GPU infrastructure in Clad

Project Proposal for Google Summer of Code 2026

Mentors: Vassil Vassilev, David Lange

Contents

1 Personal Details

Name: Vedant Goyal
Profiles: GitHub, Linkedin
Email: goyalvedant2005@gmail.com
Education: Electrical and Computer Science Engineering Undergraduate
Phone Number: +91 85710-27278

2 Project Details

Organization: CERN-HSF, Compiler Research Group
Duration: 350 hours

3 Abstract

Automatic Differentiation(AD) is a set of techniques to evaluate the derivative of functions specified by a computer program. Automatic Differentiation is different from symbolic differentiation and numerical differentiation. Symbolic differentiation is computationally expensive whereas Numerical differentiation suffers from round off errors. AD solves all these problems by applying chain rule systematically to compute gradients.

Clad is a Clang-based automatic differentiation tool that transforms C++ source code to compute derivatives. Unlike other AD tools which compute derivatives at runtime by operator overloading, Clad computes derivatives at compile time, by leveraging Clang's compiler infrastructure. It supports multiple differentiation modes like forward mode, reverse mode and hessian mode, making it suitable for a wide range of applications. Also, Clad have demonstrated promising support for GPU-based differentiation through prior efforts involving CUDA, Thrust, and integrations with applications such as XSBench and LULESH. However, all this work remains fragmented and inconsistently tested.

The project aims to consolidate the fragmented work and advance Clad's GPU infrastructure into a robust and consistent systems, and by doing this, Clad users will gain the ability to automatically generate gradients for the CUDA-accelerated code and make it applicable to real world applications. Ultimately, this work will bridge the gap between high performance GPU computing and AD.

4 Benefits to Community

- **To CERN-HSF & Scientific Community:** Strengthens the Clad's GPU infrastructure and enabling reliable AD for CUDA-accelerated programs. This would allow Clad users to efficiently compute derivatives in high-performance computing workloads.
- **To Open Source Community:** It establishes a strong foundation for the further development of such GPU-accelerated AD. It would inspire further efforts in this domain.

5 Deliverables

- Unified and up-streamed GPU infrastructure, consolidating prior work on CUDA support, Thrust API's and existing prototype integration.
- Complete differentiation pipeline for HPC-GPU applications such as LULESH, XSBench, and the in-tree CUDA raytracer, with verified correctness.
- A reproducible benchmark suite for the evaluation of Clad on GPU workloads, with comparisons against tools such as Enzyme.
- Add GPU-enabled continuous integration(CI) in `.github/workflows/` that runs `test/CUDA/` tests on GPU hardware.
- Comprehensive unit and integration tests covering GPU-related functionality.

- Optimization of gradient accumulation strategies on GPU to reduce reliance on atomic operations and improve performance.
- Improved user documentation and examples demonstrating differentiation of CUDA and Thrust-based programs.

6 Implementation Plan

Currently Clad's GPU support suffers from three core problems:

- Fragmented work across multiple unmerged branches:
  - Fork kchristin22 consists of code that would allow clad to differentiate of (CUDA) GPU kernels.
  - Fork ovdiiuv consists of activity analyzer that optimizes AD.
  - Fork a-elrawy consists of code that would allow clad to differntiate NVIDIA Thrust template functions.
  - Fork r-timmaraju consists of *cladtorch* a custom C++ tensor library for trainig LLM using Clad on GPU.
- Lack of correctness guarantee due to lack of consistent testing and reproducible benchmarking.
- Incomplete support for CUDA and Thrust API limiting Clad's ability to differentiate realistic GPU workloads.

This project aims to tackle these issues in a staged manner where initial stages would focus on auditing and recovering the existing work and then upstreaming the work by ensuring the correctness and performance. Later stages would focus on long-term maintainability via benchmarking and CI and also optimizing the implementation so that Clad is a practical AD solution for real-world GPU applications.

Task1: Audit and reproduce the existing work

In this task we will do a systematic audit of existing GPU-related contributions since the work is distributed across multiple forks and branches with varying level of completeness and compatibility with upstream codebase and based on this each contribution will be categorized as:

- Direct upstreamable with minimal changes.
- Unsuitable for integration in its current form.

The correctness of the gradients will be verified against Finite-difference approximations:

$$\frac{f(\theta+\varepsilon)-f(\theta-\varepsilon)}{2\varepsilon}, \varepsilon = 10^{-4}$$

Formula for a finite difference

Example code for the verification:

```
// Gradient computed by Clad
float clad_g = d_grad_vec[idx];

// Compute the expected gradient using Central Finite Difference on the
    target function
float fd_g = finite_diff(target_gpu_function, original_input_vec, idx);

// Calculate relative error with an epsilon to prevent division by zero
float relative_error = std::abs(clad_g - fd_g) / (std::abs(fd_g) + 1e
    -10f);
```

A relative error above 10^3 is treated as failure and requires investigation.

The outcome of this audit establishes a clear baseline and determines which components can be reliably integrated into Clad, forming the foundation for subsequent tasks.

Task2: Integration of GPU workloads

Prior efforts have demonstrated GPU-based differentiation using Clad using applications such as LULESH and XSBench but they are not part of the main codebase and lacks consistent validation. Building on the Task1 this task focuses on integrating GPU applications like LULESH and XSBench by adapting it to the current Clad infrastructure.

Task3: Extend Thrust API support

Clad currently supports only a limited subset of Thrust operations, which restricts its ability to differentiate real-world GPU programs. Many commonly used primitives such as `thrust::reduce_by_key`, `thrust::scan_by_key`, and `thrust::sort_by_key` are not supported. Although prior work has implemented support for operations like `thrust::reduce`, `thrust::transform`, `thrush::inner_product` etc,and that too remains fragmented across multiple branches and is not integrated into the main codebase yet.

This task focuses on extending Clad's support for Thrust missing and existing primitives into a unified and manageable framework. Instead of differentiating through the internal implementation, custom derivative function handlers will be defined for each operation.

```
//manual forward computation
thrust::device_vector<float> compute_square() {

thrust::device_vector<float> input(100, 2.0f);
thrust::device_vector<float> output(input.size());

thrust::transform(input.begin(), input.end(), output.begin(),
[](float x) { return x * x; });
return output;
}
```

```
//derivative handler for thrust::transform
namespace clad::custom_derivatives::thrust{
    template<typename T>
    void transform_grad(T input, T grad_output, T grad_input, size_t n)
        {

        for (size_t i = 0; i < n; ++i)
            grad_input[i] += 2 * input[i] * grad_output[i];
    }
}
```

Example code for the custom derivative handler

Task4: Reproducible Benchmarking and Comparison with Enzyme

Clad currently lacks a standardized and reproducible benchmarking framework for GPU-based differentiation. Prior work lacks consistent testing. Additionally, there is no direct comparison with Enzyme, which serves as a key baseline for CUDA-based AD. A reproducible benchmarking suite under `benchmark/`, covering representative GPU workloads such as LULESH, XSBench, and the in-tree CUDA raytracer. Each benchmark will evaluate both forward execution and gradient computation, measuring metrics such as execution time and throughput. Equivalent implementations will be evaluated using both Clad and Enzyme under identical configurations.

Task 6: GPU-Aware CI and Documentation

Clad's current CI pipeline runs only CPU-based tests, leaving GPU functionality effectively untested in automated workflows. Although GPU tests exist in `test/CUDA/`, they are not executed regularly, making it difficult to detect regressions and ensure correctness of CUDA-related features over time.

GPU-aware continuous integration to ensure reliable testing of CUDA-based differentiation will be implemented. A dedicated CI job will be added to `.github/workflows/` to execute tests from `test/CUDA/` on GPU-enabled infrastructure. To maintain efficiency, the CI workflow will be selectively triggered only when CUDA-related components are modified. This ensures continuous validation of GPU functionality without incurring unnecessary computational cost.

In addition, documentation will be expanded to include clear examples of differentiating CUDA and Thrust-based programs.

7 Timeline

Community Bonding Period (May 1 – May 24)	
May 1 – May 24	Deepen understanding of Clad's and CUDA/Thrust programming and engage with community
Coding Period (May 25 – August 16)	

Week 1 May 25 – May 31	This week is mainly for brainstorming and analyzing existing GPU-related branches and forks from **Task1**. **Deliverable:** Initial audit report outlining available branches, features, and integration status.
Week 2 June 1 – June 7	Rebase selected branches onto current Clad and attempt builds on GPU-enabled systems. Begin correctness validation using finite-difference checks. **Deliverable:** Finalized audit report
Week 3 June 8 – June 14	Resolving issues in the selected branches which are unable to merge and integration of GPU applications (LULESH / XS-Bench). **Deliverables:** Complete consolidation of fragmented GPU work.
Week 4 June 15 – June 21	Complete end-to-end differentiation for integrated applications and validate gradients against finite differences. **Deliverable:** Verified gradient computation for real-world GPU application.
Week 5 June 22 – June 28	Adding test cases for checking the performance and correctness of the newly added feature.
Week 6 June 29 – July 5	Buffer Week
Mid-Term Evaluations	
Week 8 July 11 – July 17	Extend Clad to support missing Thrust primitives using custom derivative rules. Focus on commonly used operations. **Deliverable:** Differentiation support for key Thrust primitives with unit tests.
Week 9 July 18 – July 24	Integrate support for CUDA-specific constructs such as kernel launches, device lambdas, and pointer-based activity analysis. **Deliverable:** Extended differentiation support for CUDA-specific programming patterns.
Week 10 July 25 – July 31	Performance benchmarks
Week 11 August 1 – August 9	Buffer week

Week 12 August 10 – Aug 16	Developing documentation, extend testing and presentation of the work.

8 Background and Motivation

I am an Electrical and Computer Science undergrad currently in my 3rd year of engineering. I have prior experience with C++ and Python and Julia. Over the past two years, I have been working heavily on AI and ML projects. I recall a time when I was trying to implement a computer vision model entirely in C++ from scratch for better understanding, that is when I came to know about the Clad. What initially began as a curiosity quickly turned into a deeper interest and I started reading the codebase and understanding its working principles and it strongly resonated with my interests.

My Contributions in Clad:

PR	**Description**	**Status**
#1716	Implement Multilayer Disk Offloading feature to the tape	Merged
#1729	Adds benchmark to compare Tapenade and Clad performance	Merged
#1734	Implements Dual mode tape memory manager and custom_tape specialization hook	Merged
#1708	Corrected wrong scope of variables declared inside Lambda function	Merged
#1668	Removed unnecessary underscores in generated variable names	Merged
#1741	Fixes the CI failure caused by fetching the wrong commit hash	Merged
#1657	Fix: For issue-1654	Merged
#1692	Fixes XFail test (ArrayErrors.C)	Merged
#1755	Fix: For issue-1750	Merged
#1688	Parallelism in clad::restore_tracker using multithreading	Open
#1685	Fix: For issue-1074	Open

I found the project exciting, as it aligns closely with my interests in optimization, memory-efficient design, and parallel computing. It offers me an opportunity to deepen my under-

standing of compiler- based programs. I am excited to learn a lot from the mentors and other contributors.

9 Availability

I have no conflicting academic or professional commitments during the summer, I can devote 35 hours per week for this project. I am perfectly fine in communicating over Email, Zoom, Discord or any other platform preferred by mentors. Given the scope of consolidating multiple years of fragmented GPU work, I understand this project may extend beyond the standard GSoC period. I will remain available beyond August and committed to continuing contributions until the work is fully integrated and upstreamed.iI will remain active and make sure to share the updates on the projects regularly and discuss my doubts with the mentors.