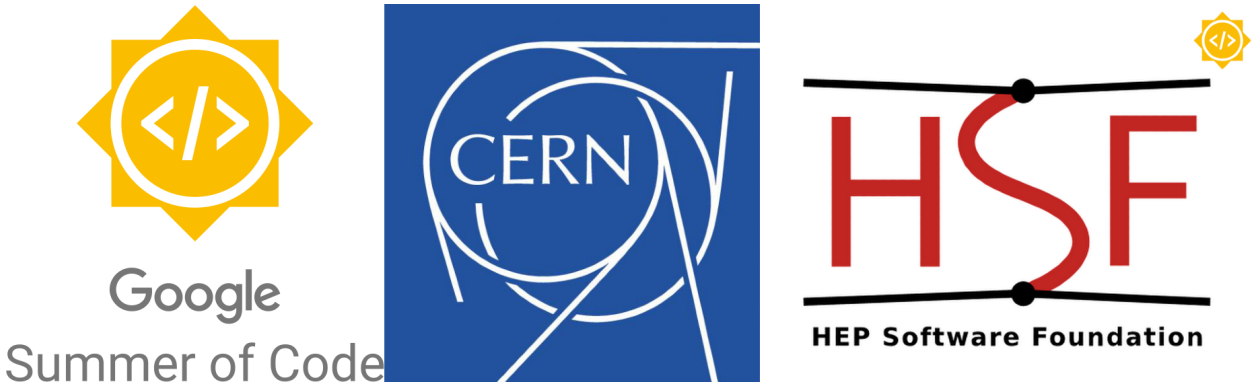
GSoC 2023 Project Proposal
CERN HSF
High Energy Physics Software Foundation



Implement vector mode in forward mode automatic differentiation in Clad

Mentors

- Vassil Vassilev
- Parth Arora

Personal Details

Vaibhav Thakkar
GitHub link: https://github.com/vaithak
Email: vaibhav.thakkar.22.12.99@gmail.com
Phone number: +91 95590-22960

Project Details

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. Clad is a Clang based plugin which can be used for differentiating user defined functions using source code transformation. The library is able to differentiate non-trivial functions and to find a partial derivative for trivial cases.

Vector mode support will facilitate the computation of gradients using the forward mode AD in a single pass and thus without explicitly performing differentiation n times for n function arguments. The major benefit of using vector mode is that computationally expensive operations do not need to be recomputed n times for n function arguments.

For example, if we want to compute df/dx and df/dy of a function f(x, y) using the forward mode AD in Clad, then currently we need to explicitly differentiate f two times. Vector mode will allow the generation of f_d(x, y) such that we will be able to get partial derivatives with respect to all the function arguments (gradient) in a single call.

Having a vectorized forward mode AD is better than using reverse mode as the latter requires storing all the inputs for every node in the computation graph thus consuming large amounts of memory. Also, vectorized forward mode AD can take utilize parallelization capabilities of modern processors..

Benefit to the community

The project will help in improving the efficiency of the clad library as computationally expensive statements are computed only once in the derived function instead of n times, which would be the case if we differentiate the function using Forward Mode AD n times, once with respect to each input parameter. I think providing this efficiency in forward mode AD will be a great advantage for the open source and research community using clad.

Project Details

- **Current Implementation - single input variable differentiation in forward mode**
    - Forward mode AD works on function ***f*** with single output and many inputs. Current implementation in clad computes the derivative of the output with respect to a single input as specified in the argument. If we need to differentiate the function with respect to all the inputs, we need to call ***clad::differentiate*** *n* number of times (n - number of inputs) which will essentially generate *n* different functions at compile time. During run time, this will cause many repeated computations which could have been shared

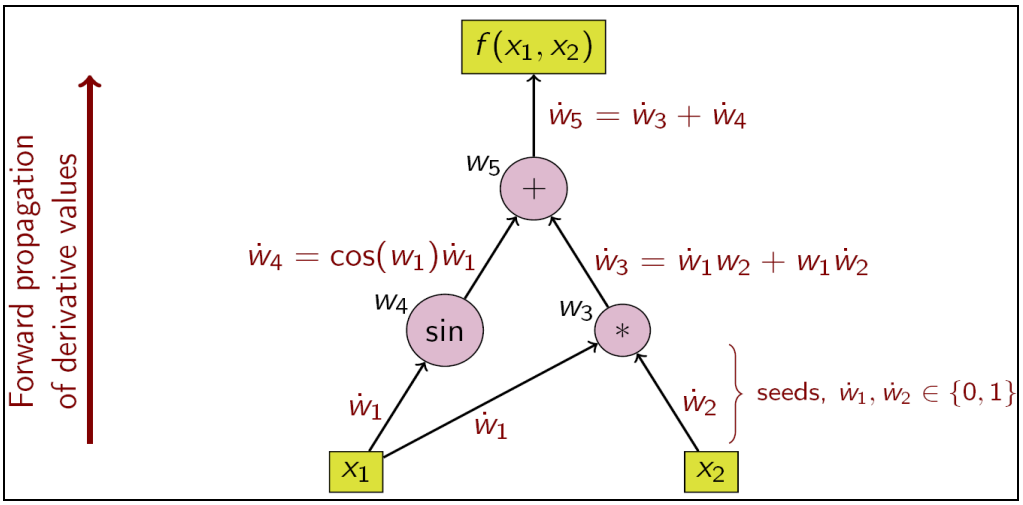
between functions runs for different inputs.

```
double f(double x, double y) {
  return x * y;
}

int main() {
  // Call clad to generate the derivative of f wrt x.
  auto f_dx = clad::differentiate(f, "x");
  // Execute the generated derivative function.
  std::cout << f_dx.execute(/*x=*/3, /*y=*/4) << std::endl;
}
```

- **Proposed Improvement - producing vector of all directional derivatives**
    - Current approach works by accumulating derivatives in the computation graph starting from the input nodes and going up to the output node. For each internal node, it essentially computes ***d (node_out) / d (input_var)*** where **node_out** is the output of the current internal node. For the leaf or input variable nodes, a seed of one hot vector is used.

      An example demonstrating this:



    - The above approach can be improved and modified to a vectorized mode so that every node outputs a vector of directional derivatives, where i^{th} element of the vector signifies the derivative of that node's output with

respect to i^{th} input variable. This means that there each input node will have a different one hot vector as seed (1 at i^{th} index for i^{th} input node).

- **Why will this be beneficial?**
    - If there is computationally expensive operation in the function **f** which is independent of any of the input variables, this can be shared while computing derivatives for all inputs and it will be computed only once (instead of *n* times in the current implementation). An example function:

```
double f(double x, double y) {
  double t = expensiveFunc();
  return x * y * t;
}
```

    - One more important point to add in the beneficial section is: vectorization of the differentiation operation. For example, consider a function with n inputs, x_0, x_1, x_2...., x_{n-1} and one output y. Assume the function has a statement:

```
u = v;
```

      If the function was being differentiation w.r.t only one input, say x_kThen the corresponding derivative statement would be:

```
du/dx_k = dv/dx_k;
```

      But in the vector forward mode, if we are differentiating w.r.t to each input, the corresponding derivative statements would be something like this:

```
      for (size_t i = 0; i < n; ++i
        du/dx_i = dv/dx_i;
```

      Vector parallelization can be easily utilized here and it also satisifes the requirements for SIMD parallelisation.

- **Required changes in clad**
    - To support the new implementation, we will need to finalize an interface of how this mode will be invoked from ***clad::differentiate***, one simple way can be that if the user invokes this without providing any args specifying the name / index / array index of the input variable, this will imply that the

intention is to compute all directional derivatives together. Thus, the call can simply be: `auto f_dx = clad::differentiate(f);`

- Another, major change will be generalizing the ForwardModeVisitor to handle vector of derivatives instead of a scalar for each node (including the input nodes, which will now be initialized by a one hot vector instead of a 0/1).
  This will also mean that also simple scalar operations into an iteration over the vector of derivatives, for example:

```
// example expression
u = v;


// earlier derived method code
du = dv // du = partial derivative of u w.r.t x
        // dv = partial derivative of v w.r.t x


// new derived method code
for (int i = 0; i < n; ++i) {
  du[i]= dv[i];// du[i] = partial derivative of u w.r.t x_i
               // dv[i] = partial derivative of v w.r.t x_i
}
```

Deliverables

This main delivarables for the project are as follows:

1. Add a new mode to the top-level clad interface `clad::differentiate` for vector mode.

2. Extend and generalize the ForwardModeVisitor to produce a single function with the directional derivatives.

3. Document the above features and write unit tests for them.

Timeline

Below is a tentative timeline for the proposed project

Duration	Work Details
May 5 - May 28 (3 weeks)	**Community Bonding period** ● Read articles and watch videos to get a better understanding of automatic differentiation, Clang's AST and debugging tools. ● Writing an initial blogpost explaining an high level overview of the clad project and providing an idea of where does my project fit in. ● Setting up a development environment.
May 29 - June 5 (1 week)	This week is mainly for brainstorming and finalizing the modified interface of ***clad::differentiate*** to provide a new mode for vectorized approach, and also discuss the plan of action to generalize ForwardModeVisitor.
June 6 - June 13 (1 week)	**Initial support for vectorized forward AD** Generalize the existing ForwardMode visitor to handle gradient of functions with a single return statement of mathematical operations. For ex. f(x, y, z, c) = c + (x+y)*z **Deliverable:** working implementation of vectorized forward mode AD for simple mathematical functions.
June 14 - June 20 (1 week)	Expand the vectorized mode to handle similar types of functions as supported by existing implementation of single variable derivative.
June 21 - June 28 (1 week)	Adding test cases to check generated code in case of the primal function containing some computationally expensive operation as explained above. **Deliverable:** working implementation of vectorized forward

	mode AD for general functions with good test coverage.
June 29 - July 6 (1 week)	Buffer time in case something from above spills over.
July 7 - July 14 (1 week)	Midterm evaluations.
July 15 - July 21 (1 week)	Adding techniques for improving efficiency by exploiting parallelization strategies.
July 22 - July 29 (1 week)	Adding test cases for checking the efficiency of the derived code. **Deliverable:** improved and efficiency mode utilizing vectorization and parallelization in processors.
July 30 - Aug 5 (1 week)	Buffer time for adding any further efficiency improvements with the help from mentors.
Aug 6 - Aug 13 (1 week)	**Documentation phase** This will be to improve examples, tutorials, README and other documentation related docs to explain the usage and benefit of this newly added functionality. **Deliverable:** completed documentation of the new mode.
Aug 14 - Aug 20 (1 week)	Buffer time.
Aug 21 - Aug 28 (1 week)	Publish the final blogpost concluding the project and detailing my experience and complete final evaluation report.

Why me ?

I am always on the lookout for challenging engineering and applied math problems to solve. My approach to technology is one of deep understanding - I love to get to grips with the first principles behind any system and, where possible, even implement it from scratch to ensure a complete understanding.

Regarding my educational background, I am a graduate from the Indian Institute of Technology, Kanpur with majors in Electrical Engineering and minors in Computer Science in the fields of Computer Systems, Theory of Computation, and Machine Learning.

I have also completed a GSoC project in 2021 which involved implementing scalable geometric algorithms in high dimensions using C++..

I find the problem of Automatic Differentiation very interesting. In fact, I have worked on a for-fun project related to this in the past, which involved implementing an autograd engine in Python. Similar to the one used in Pytorch, the aim was to perform differentiation on the dynamic computation graph formed by performing operations on a custom tensor class.

Although I have a basic understanding of LLVM and Clang and will learn more from mentors, but I have worked on a similar compiler plugin problem where I implemented a plugin for protocol buffer compiler to autogenerate custom Golang code for validation of grpc requests using annotations provided in the proto file. The interface was very similar to this open source project.

As part of my evaluation tasks, I have also contributed these PRs in the clad project:

- Fix reverse mode computation by adding missing gradient update during loop initialization PR #518
- Fix for missing nested namespace identifier: PR #524

I am confident that my technical abilities and experience, combined with my love of problem-solving, will make me an asset to this project. At the same time, I understand the importance of humility and recognize that there is always more to learn. I am excited to work with and learn from the mentors and other contributors on this project.

Why GSoC with Clad project in CERN HSF ?

I am extremely excited about the possibility of working with CERN software projects because of my deep interest in physics and the groundbreaking experiments that are conducted at the organization. For example, the LHC project has always been a source of fascination for me, and I would be thrilled to contribute to the development of the software that will be used by scientists and the research community working on such amazing projects.

In addition to my passion for physics, I am also highly motivated by applied research projects, and I believe that working on this project would allow me to explore how

theoretical algorithms can be implemented efficiently in practice. Overall, I am confident that this experience would be both challenging and rewarding, and I look forward to the opportunity to make meaningful contributions to CERN's groundbreaking work.

Other commitments during summer

I have been working as a programmer at a startup for the past 20 months and my work there has been pretty much streamlined, so I can dedicate the required 20-25 hours per week for this project.

Preferred medium of communication

I am perfectly fine with Email, Discord, Zoom or any other similar medium of communication. My preferred language for communication is English. I will try to have regular meetings and sync ups with mentors to share my progress updates, and remain active on an asynchronous chat platform to ensure I regularly discuss my doubts from the mentors instead of getting stuck on any issues for too long.