Proposal for GSoC 2022

Add Initial Integration of Clad with Enzyme

Manish Kausik H
Indian Institute of Technology, Bhubaneswar
mkh10@iitbbs.ac.in

Abstract

Clad is an open source plugin to the Clang compiler that detects from the parsed Abstract syntax tree, calls to differentiate a defined function, generates code that differentiates the function using the concept of Automatic Differentiation(AD) and modifies the Abstract Syntax Tree(AST) to insert the generated code. While clad works in the frontend of the compilation process, Enzyme, another LLVM based AD plugin works in the backend, where it takes in code in LLVM IR form and then differentiates the code.

This proposal aims to integrate Clad with Enzyme, and give the user the option of selecting Enzyme for Automatic Differentiation, based on his/her needs. This will give the user the same User Interface as clad for writing his/her code, but the option of using Enzyme as the backend with all its optimisations to calculate the Derivative/Gradient of the requested function. This proposal also briefly gives insights into how this can be achieved by tapping into the existing code base of Clad.

Deliverables

This project proposes to add the following features to clad:

1. Define the Enzyme configuration requirements and enable Clad to communicate efficiently with Enzyme.

2. Enable Clad to use Enzyme's AD feature when requested by the User.

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

Clad API

Clad provides the function, “clad::differentiate” to differentiate a function in forward mode and “clad::gradient” to differentiate a function in reverse mode.

Suppose in the following code snippet we want to differentiate function “foo”, then the corresponding calls to “clad::differentiate” and “clad::gradient” are described in the code snippet:

```
#include "clad/Differentiator/Differentiator.h"
#include <iostream>

double foo(double x) { return x * x; }

int main() {
  // Call clad to generate the derivative of foo wrt x.
  auto foo_dx = clad::differentiate(foo, "x");

  // Call clad to generate the gradient of foo
  auto foo_grad = clad::gradient(foo);
}
```

“foo_dx” and “foo_grad” are pointers to the functions generated by clad that contain code for differentiating foo with respect to x and foo’s gradient respectively. These functions are generated by parsing function foo’s Abstract Syntax Tree(AST) and for every operation represented by a node of the tree we find its derivative and propagate the pass forward or backward respectively. The functions generated when dumped, can show the step by step process involved in the derivative calculations.

Enzyme API

Enzyme asks the user to use the phrase “__enzyme_autodiff”(for Reverse Mode) and “__enzyme_fwddiff”(for Forward Mode) as part of any function name that is supposed to refer to the derivative of some other function. Suppose one wants to find the derivative of function foo from the previous code snippet, via Reverse Mode, then one has to make a call to a function called “__enzyme_autodiff_foo(foo)” or just “__enzyme_autodiff(foo)” wherever the derivative is needed. Enzyme recognises these calls and then replaces them with the code for Derivatives in the LLVM IR stage.

The following code snippet shows how one can use enzyme to obtain derivatives of the function foo:

```
#include <iostream>
extern double __enzyme_autodiff(void*, double);
double foo(double x) { return x * x; }
double dfoo(double x) {
    // This returns the derivative of square or 2 * x
    return __enzyme_autodiff((void*) square, x);
}

int main() {
    for(double i=1; i<5; i++){
        printf("foo(%f)=%f, dfoo(%f)=%f",i,foo(i),i,dfoo(i));
     }
}
```

When the following code snippet is passed to the frontend of clang compiler, it generates the LLVM IR of the code, with the function call "__enzyme_autodiff" used in dfoo. Now the LLVM IR is passed to the enzyme program, which then replaces the "__enzyme_autodiff" used in dfoo with the actual derivative code. The LLVM IR can be passed before optimizations or after optimization. Now the modified LLVM IR can be passed to the LLVM Backend to obtain the executable code.

Proposed Changes to Clad

The proposal aims to give the user an option of selecting Enzyme as the tool for AD, instead of the clad AST method. The following code snippet will denote that the user has requested clad to use Enzyme for calculating the Derivative/Gradient:

```
#include "clad/Differentiator/Differentiator.h"
#include <iostream>

double foo(double x) { return x * x; }

int main() {
  // Call clad to generate the derivative of foo wrt x, but use Enzyme as
backend.
```

```
  auto foo_dx = clad::differentiate<clad::opts::use_enzyme>(foo,"x");

  // Call clad to generate the gradient of foo, but use Enzyme as backend
  auto foo_grad = clad::gradient<clad::opts::use_enzyme>(foo);
}
```

A call to “clad::differentiate” or “clad::gradient” if asked to be used with Enzyme by the user, must trigger clad to initialize the correct data structures (where the derivatives can be stored by enzyme) and pass them to the enzyme_autodiff (for reverse mode) or enzyme_fwddiff (for forward mode) functions, which can then be replaced with the respective AD code by Enzyme. Thus Clad has to generate the following wrapper functions, when the user requests Clad to use Enzyme:

```
extern double __enzyme_fwddiff_foo_x(void*, double);
double foo_dx_forward(double x) {
    double seed[1] = {0}; //1 here represents number of params of foo
    seed[0] = 1; //This sets in which direction we want the derivative, 0
represents the index of x in the list of input params of foo

    // We pass the seed(direction) as well as the location at which
derivative is needed to Enzyme
    auto diff = __enzyme_fwddiff_foo_x((void*) foo, x, seed[0]);
    return diff;
}

extern double __enzyme_autodiff_foo(void*, double);
double foo_grad_backward(double x) {
    int enzyme_dup;
    double d_x[1];//Initializing data structure to store the result; 1
represents that the function has only 1 input param
    //We tell Enzyme that the passed arguments are of type "Duplicated"
with the llvm metadata "enzyme_dup".
    __enzyme_autodiff_foo((void*) foo, enzyme_dup, &x, &d_x[0]);
    return d_x;
}
```

The above code must be inserted by clad so that, when the LLVM IR is generated, Enzyme can act upon the requested functions to generate their Derivatives.

Planned Timeline

Since the exact dates are not fixed yet and are flexible, here is a 10 week timeline for the proposed project (Tentative dates are mentioned):

Week No	***Tasks to be Completed***
Week 1 *(May 29th to June 4th)*	Explore the codebase, discuss with mentors the existing code architecture and identify locations to be modified in the codebase. Explore associated libraries like LLVM and Clang. Also learn about the Differentiation concepts that are used in the code base. Read about LLVM IR and how Enzyme works on it to generate the Derivative of a Function.
Week 2 *(June 5th to June 11th)*	[Coding Begins] Add the option of "clad::opts::use_enzyme" to the clad API. This involves adding a state variable for every differentiation request that stores whether the user has requested the use of Enzyme for differentiation. **Deliverable:** A working code snippet that shows that clad can correctly identify the user request for Enzyme backend and set the corresponding state variable in the differentiation request.
Week 3 *(June 12th to June 18th)*	Modify the "clad::ReverseModeVisitor::Derive" code to check if the differentiation request asks for the use of Enzyme. If such a request is made, then branch off from the main Derive function to generate a different code. **Deliverable:** Correctly recognise that one needs to generate Reverse Mode Enzyme code when requested by the user.
Week 4 *(June 19h to June 25th)*	Continue work from Week 3, work on generating code for Reverse AD that is enzyme compatible. Make sure the correct LLVM IR is generated for Enzyme to work on. **Deliverable:** A working code snippet that uses Enzyme to generate Gradients of a function with the interface call "clad::gradient"
Week 5 *(June 26th to July 2nd)*	Write unit tests to verify that the use of Enzyme for gradient code generation and calculation is correctly done. **Deliverable:** An exhaustive set of test cases that show that clad generates the correct code, compatible with Enzyme and the code evaluates the gradients correctly during runtime.
Week 6 *(July 3rd to July 9th)*	Buffer Period, Phase 1 evaluations
Week 7 *(July 10th to July 16th)*	Modify the "clad::ForwardModeVisitor::Derive" code to check if the differentiation request asks for the use of Enzyme. If such a request is made, then branch off from the main Derive function to generate a different code.

	Deliverable: Correctly recognise that one needs to generate Forward Mode Enzyme code when requested by the user.
Week 8 *(July 17th to July 23rd)*	Continue work from Week 7, work on generating code for Forward AD that is enzyme compatible. Make sure the correct LLVM IR is generated for Enzyme to work on. **Deliverable:** A working code snippet that uses Enzyme to generate Derivatives of a function with the interface call "clad::differentiate".
Week 9 *(July 24th to July 30th)*	Write unit tests to verify that the use of Enzyme for derivative code generation and calculation is correctly done. **Deliverable:** An exhaustive set of test cases that show that clad generates the correct code, compatible with Enzyme and the code evaluates the derivatives correctly during runtime.
Week 10 *(July 31st to Aug 6th)*	Buffer Period, Phase 2 evaluations

Personal Information

A. Basic Details

Name	Manish Kausik H
Email	hmanishkausik@gmail.com mkh10@iitbbs.ac.in
Mobile Number	+91 7760356001
Time Zone	IST (+5:30)
Github	Nirhar
University	Indian Institute of Technology, Bhubaneswar
Degree	B.Tech in Computer Science and Engineering and M.Tech in Computer Science and Engineering (Dual Degree)
Year	4th year Undergraduate (2023 expected)
Availability	A minimum of 20 hours per week during the coding period Preferred slot: June - August

B. *Why am I interested in this project?*

Ever since I studied principles of compiler design as a part of my coursework, I've always wanted to take up some project related to compiler design. During coursework, we were taught basics of LLVM and its use in the frontend design of a compiler. I was interested in exploring LLVM and I find this project a perfect opportunity to study tools like LLVM and Clang.

Moreover, I also find this project interesting because I find the whole idea of using a compiler frontend plugin to generate derivatives extremely cool! I also have an interest in understanding scientific applications in the real world, and I am curious to learn more about how various applications use Clad. I hope to learn more about the activities of the scientific community at CERN, by being a part of this project.

C. *My Contributions to Clad*

I am relatively new to clad, and have been exploring the codebase over the past month. Currently, I have raised 2 pull requests to solve 2 issues in Clad:

#388	Including cstring in a clad program throws errors [Solved by #398, Accepted]
#408	Incorrect result when calling clad::jacobian wrt given parameter [Solved by #422, Accepted]