Add Initial Integration of Clad with Enzyme

Manish Kausik H, GSoC 2022

Contents

1. Introducing clad and enzyme

2. A Brief overview of Clad and its API

3. A Brief overview of Enzyme and its API

4. Integrating Enzyme with Clad

5. Some Implementation ideas

Clad and Enzyme

- Both are libraries to perform Automatic Differentiation
- Clad:
  - A plugin to the Clang compiler
  - Specific to C++ Language
  - Works on the frontend - Modifies the AST
  - Has: Forward mode, Reverse Mode, Hessian, Jacobian, Error Estimation, Numerical Diff
- Enzyme:
  - Works on the Backend - AD on the LLVM IR
  - Applicable to multiple languages
  - Has: Forward mode, Reverse Mode

Clad API

```
#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);
}
```

Enzyme API

```
#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*) foo, x);
}

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

Integrating Enzyme with Clad

```
#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);
}
```

The above code must transform to ….

Integrating Enzyme with Clad

```
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;
}
```

Disclaimers!

- New to Template Metaprogramming in C++
- Aware of just the basics of STL
- Claims made can be wrong/improved
- Feel free to point out mistakes

Implementation Ideas

1. Implementing ***clad::opts::use_enzyme***

```
namespace clad{
      namespace opts{
            struct use_enzyme{};
      }
}
```

```
template <typename ArgSpec = const char*, typename F, typename enzyme,
          typename DerivedFnType = GradientDerivedFnTraits_t<F>,
          typename = typename std::enable_if<std::is_same<enzyme,
clad::opts::use_enzyme>::value>::type,
          typename = typename std::enable_if<
!std::is_class<remove_reference_and_pointer_t<F>>::value>::type>
 CladFunction<DerivedFnType, ExtractFunctorTraits_t<F>, true> __attribute__((
     annotate("G"))) CUDA_HOST_DEVICE
 gradient(F f, ArgSpec args = "",
          DerivedFnType derivedFn = static_cast<DerivedFnType>(nullptr),
          const char* code = "") {
   assert(f && "Must pass in a non-0 argument");
   return CladFunction<DerivedFnType, ExtractFunctorTraits_t<F>, true>(
       derivedFn /* will be replaced by gradient*/, code);
}
```

Implementation Ideas

2. Reverse Mode Differentiation Code Generation

- **DiffCollector::VisitCallExpr** must set a variable in the DiffRequest Object, that states whether the user wants to use enzyme or not.
- **ReverseModeVisitior::Derive** must create a new branch for Enzyme DiffRequests, with a constant template code
- Must link the Code generated by **ReverseModeVisitor::Derive** with the **CladFunction** class (Need to explore this)
- How can **DiffCollector::VisitCallExpr** recognise the request for use of enzyme based on a template parameter? (Need to explore this)

Thank You!