Enzyme: High-Performance Automatic Differentiation of LLVM

William S. Moses (wmoses@mit.edu), Valentin Churavy (vchuravy@mit.edu)
MIT CSAIL

Existing Automatic Differentiation Tools

- Differentiable DSLs (TensorFlow, PyTorch, DiffAuto) provide a new language where everything is differentiable. Must rewrite code in DSL.
- Operator Overloading (Adopt, JAX) tools provide differentiable versions of existing language constructs (double \(\Rightarrow\) adouble, np.sum \(\Rightarrow\) jax.sum). May require rewriting to use non-standard utilities for support.
- Source Rewriting tools statically analyze code to produce a new gradient function in the source language. Requires all code available ahead of time and is difficult to use with external libraries.

Optimization and AD

All tools for existing code operate at the source level preventing optimizations before AD without reimplementing compiler analyzers and optimization into the AD tool. While historically not considered necessary, we demonstrate in Figure 1 how crucial optimization prior to AD can be.

Performing AD on low-level IR presents additional challenges as source-level information is lost. For example, differentiating memcpy in Figure 2 requires the underlying type of the data to select the correct gradient.

Challenges of AD on Low-Level Code

- To differentiate inputs to double, dst, void, src (memcpy(dst, src, 8));
- \#Assume double inputs

Figure 2. Top: Call to memcpy for an unknown 8-byte object. Left: Gradient for a memcpy of 8 bytes of double data. Right: Gradient for a memcpy of 8 bytes of float data.

Enzyme synthesizes derivatives by:
- Running Type and Activity Analysis
- Allocating shadows of active variables
- Creating a "reverse" copy of BasicBlock's in the original code that compute the adjoints of its instructions in reverse order.

```c
float mag(const float*); // Compute magnitude in O(N) 

void norm(float out, const float* in) {
    // LCM norm: mag = sqrt(\sum_x in_x^2)
    float out[\#in]; 
    for (int i = 0; i < N; i++) {
        out[i] = in[i] * in[i]; 
    }
    
    // AD then LCM, O(N^2)
    float out = sqrt magna_TIMER
    for (int i = 0; i < N; i++) {
        out[i] = in[i] * in[i]; 
    }
    
    // Reverse pass 
    double dres = 0;
    for (int i = 0; i < N; i++) {
        out[i] = in[i]; 
    }
    
    // Type Analysis: a new interprocedural analysis that derives the corresponding gradient function.
    return with a branch to the reverse pass.
```

Figure 3. Gradient synthesis of relu for (x > 0).

```c
void f(void* dst, void* src) { memcpy(dst, src, 8); }

// Assume double inputs

void g(double* dst, double* dsrc, double* ddsrc) {
    // Forward pass 
    memcpy(dst, src, 8); 
    // Reverse pass 
    ddsrc[0] = ddst[0];
    ddst[0] = 0;
}
```

```c
# Define the adjoint of a function \(f\).

@ddx
void adv(float out, const float* in) {
    // LCM: mag = sqrt(\sum_x in_x^2)
    float out[\#in]; 
    for (int i = 0; i < N; i++) {
        out[i] = in[i] * in[i]; 
    }
    
    // AD then LCM, O(N^2)
    float out = sqrt magna_TIMER
    for (int i = 0; i < N; i++) {
        out[i] = in[i] * in[i]; 
    }
    
    // Reverse pass 
    double dres = 0;
    for (int i = 0; i < N; i++) {
        out[i] = in[i]; 
    }
    
    // Type Analysis: a new interprocedural analysis that derives the corresponding gradient function.
    return with a branch to the reverse pass.
```

Evaluation

Performing AD after optimization yields a 4.5x speedup over AD before optimization. This accounts for much, but not all, of Enzyme's improvement over prior art (different cache and activity analysis implementations).

Figure 4. Relative speedup of AD systems on ADBench+ benchmarks, higher is better. A red X denotes programs that an AD system does not produce a correct gradient. A value of 1.0 denotes the fastest system, whereas 0.5 denotes taking twice as long.

Usage

A user can use gradient functions by calling \(\text{__enzyme_autodiff}\) with the function to be differentiated as the first argument. When the Enzyme optimization pass is run, it will replace any calls to \(\text{__enzyme_autodiff}\) with a call a newly-generated gradient function.

```
Enzyme
```

```
Enzyme
```

Figure 5. Convention for invoking Enzyme.

Acknowledgements

William S. Moses was supported in part by a DOE Computational Sciences Graduate Fellowship DE-SC0019923. Valentin Churavy was supported in part by the DARPA program PAPPA under Contract Number HR00112090016, and in part by NSF Grant OAC-1835443. This research was supported in part by LANL grant S31711. Research was sponsored by the United States Air Force Research Laboratory and was accomplished under Cooperative Agreement Number FA8750-19-2-1000, the views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.