Optimizing automatic differentiation using activity analysis

Andriichuk Maksym, 10.07.2024

Short introduction to Clad

Clad is a Clang plugin designed to provide automatic differentiation (AD) for C++ mathematical functions.

It generates code for computing derivatives modifying abstract syntax tree using LLVM compiler features. AD breaks down the function into elementary operations and applies chain rule to compute derivatives of intermediate variables. Clad supports forward- and reverse-mode differentiation that are effectively used to integrate all kinds of functions

```cpp
double square(double x){
  double result = x*x;
  return result;
}
```

```
FunctionDecl 0x14988afa0 <my.cpp:12:1, line:15:1> line:12:8 square 'double (double)'
|-ParmVarDecl 0x14988aed0 <col:15, col:22> col:22 used x 'double'
`-CompoundStmt 0x14988b200 <col:24, line:15:1>
  |-DeclStmt 0x14988b1a0 <line:13:3, col:22>
  | `-VarDecl 0x14988b0a8 <col:3, col:21> col:10 used result 'double' cinit
  |   `-BinaryOperator 0x14988b180 <col:19, col:21> 'double' '*'
  |     |-ImplicitCastExpr 0x14988b150 <col:19> 'double' <LValueToRValue>
  |     | `-DeclRefExpr 0x14988b110 <col:19> 'double' lvalue ParmVar 0x14988aed0 'x' 'double'
  |     `-ImplicitCastExpr 0x14988b168 <col:21> 'double' <LValueToRValue>
  |       `-DeclRefExpr 0x14988b130 <col:21> 'double' lvalue ParmVar 0x14988aed0 'x' 'double'
  `-ReturnStmt 0x14988b1f0 <line:14:3, col:10>
    `-ImplicitCastExpr 0x14988b1d8 <col:10> 'double' <LValueToRValue>
      `-DeclRefExpr 0x14988b1b8 <col:10> 'double' lvalue Var 0x14988b0a8 'result' 'double'
```

So what is activity analysis(AA)?

First a bit of motivation…

Sometimes Clad produces adjoints that are useless for the desired final derivative. Let’s call those variables ***passive***. Otherwise, the variable is called ***active***. Now Clad assumes all variables are active, but we can do much better using AA.

Lets see the example:

code	forward mode	fm+aa

f(a, b, c):
  x = a*b
 	
  d = a*c
 	
  return x
	f_darg0(a, b, c):
	  d_a=1
	  d_b=0
	  d_c=0
	
	  d_x = d_a * b + a * d_b
	  x = a*b
	
	  d_d = d_a * c + a * d_c
	  d = a*c
	
	  return d_x
		f_darg0(a, b, c):
		  d_a=1
		  d_b=0
		
		  d_x = d_a * b + a * d_b
		  x = a*b
		
		  d = a*c
		
		  return d_x

AA is the combination of a forward and a backward analysis.

It propagates forward the **Varied** set of the variables that depend in a differentiable way on some independent input. Similarly, it propagates backwards the **Useful** set of the variables that influence some dependent output in a differentiable way.

Since the relation "depends in a differentiable way of" is transitive on code sequences, the essential equations of the propagation are:

$$Varied^{+}(I) \;=\; Varied^{-}(I) \;\times\; Diff - depp(I)$$

$$Useful^{-}(I) \;=\; Diff - dep(I) \;\times\; Useful^{+}(I)$$

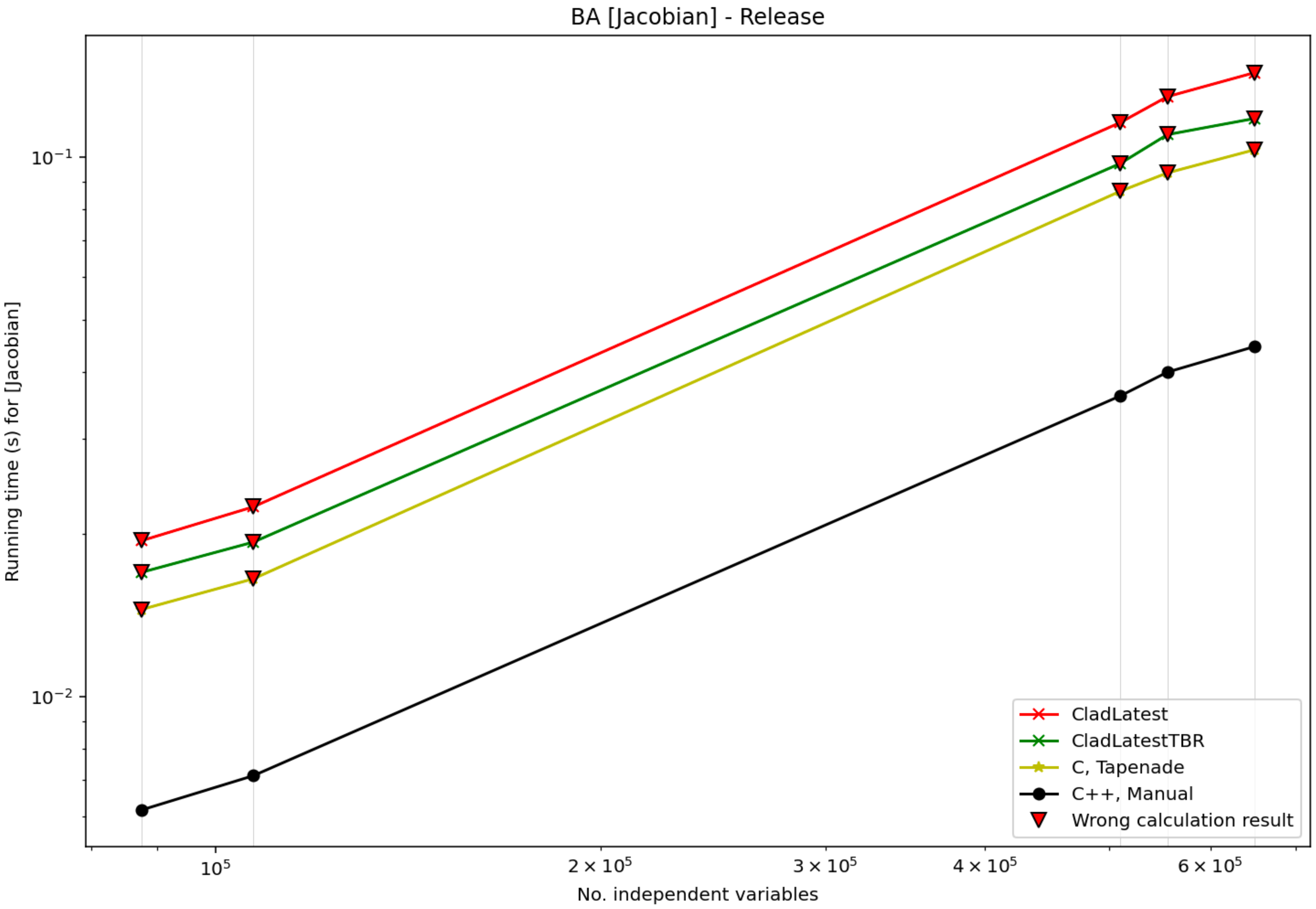
Where $Varied^{-}(I)$, $Varied^{+}(I)$ are sets of **Varied** variables before and after $I - th$ instruction,

$(v_1, v_2) \in Diff - dep(I)$ iff v_2 depends on v_1 after $I - th$ instruction,

$v_2 \;\in\; S \;\times\; Diff - dep(I) \Longleftrightarrow \exists v_1 \;\in\; S, (v_1, v_2) \;\in\; Diff - dep(I)$

And finally we define the set of all *active* variables as follows:

$$Active^{+}(I) \;=\; Varied^{+}(I) \;\cap\; Useful^{+}(I)$$


BA [Jacobian] - Release
Running time (s) for [Jacobian]
No. independent variables
10^{-1}
10^{-2}
10^5
2×10^5
3×10^5
4×10^5
6×10^5
CladLatest
CladLatestTBR
C, Tapenade
C++, Manual
Wrong calculation result

Note:

After AA is implemented and both AA and TBR analysis are default, there is a potential in modifying TBR using AA.

References

[1] L.Hascoët, V.Pascual. The Tapenade Automatic Differentiation Tool: Principles, Model, and Specification. *ACM Transactions on Mathematical Software 39(3):20:1-20:43*.