Clad

COMPILER C|R RESEARCH

Accelerating Large Scientific Workflows Using Source Transformation Automatic Differentiation

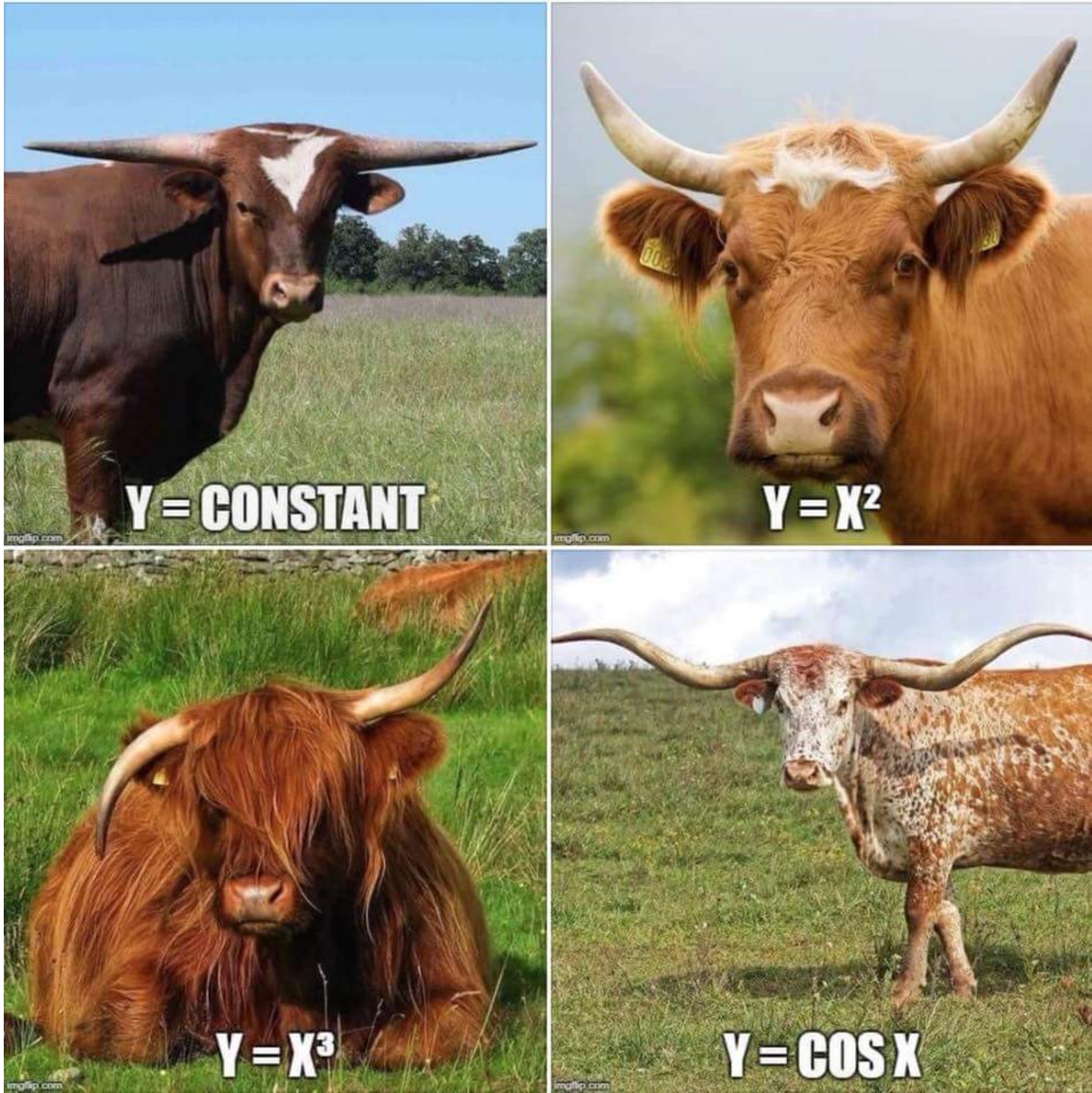
Vassil Vassilev, Princeton
compiler-research.org

This work is partially supported by National Science Foundation under Grant OAC- 2311471, *OAC- 1931408* and NSF (USA) Cooperative Agreement OAC-1836650

Motivation

Provide automatic differentiation for C/C++ that works with~~out~~ little code modification (including legacy code)

AD Basics



AD. Chain Rule

$$\frac{dz}{dx} = \frac{dz}{dy} . \frac{dy}{dx}$$

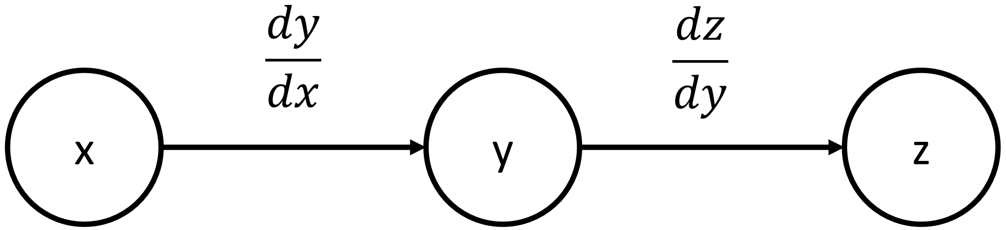
Intuitively, the chain rule states that knowing the instantaneous rate of change of *z* relative to *y* and that of *y* relative to *x* allows one to calculate the instantaneous rate of change of *z* relative to *x* as the product of the two rates of change.

“if a car travels twice as fast as a bicycle and the bicycle is four times as fast as a walking man, then the car travels 2 × 4 = 8 times as fast as the man.” G. Simmons

AD. Algorithm Decomposition

```
y = f(x)
z = g(y)

dydx = dfdx(x)
dzdy = dgdy(y)
dzdx = dzdy * dydx
```

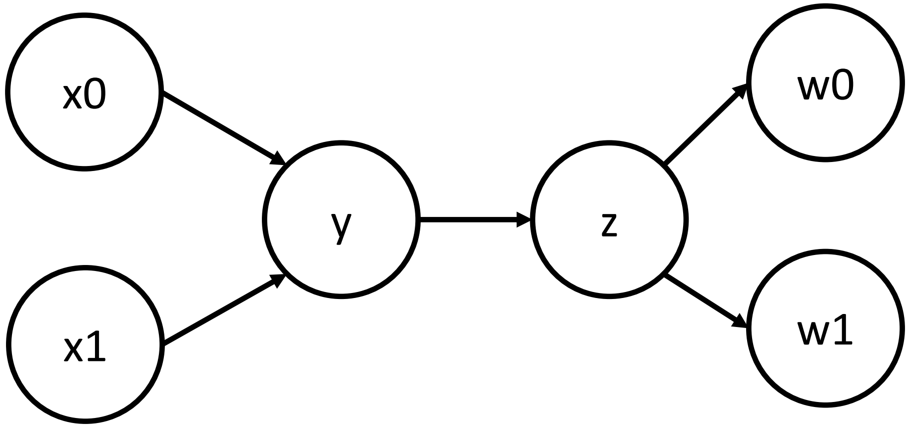


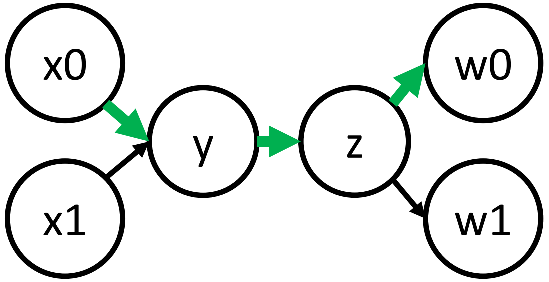
In the computational graph each node is a variable and each edge is derivatives between adjacent edges

We recursively apply the rules until we encounter an elementary function such as addition, substraction, multiplication, division, sin, cos or exp.

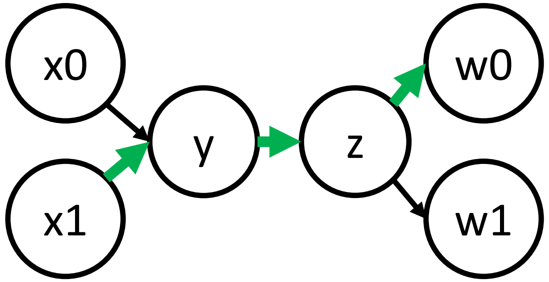
AD. Chain Rule

```
y = f(x0, x1)
z = g(y)
w0, w1 = l(z)
```

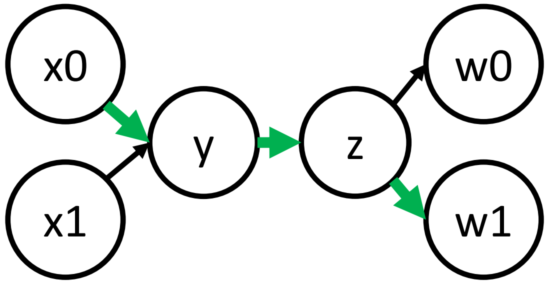




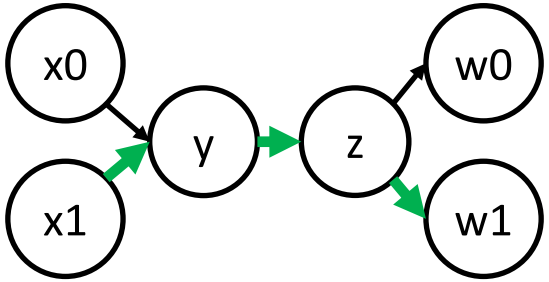
$$\frac{\partial w0}{\partial x0} = \frac{\partial w0}{\partial z}\frac{\partial z}{\partial y}\frac{\partial y}{\partial x0}$$



$$\frac{\partial w0}{\partial x1} = \frac{\partial w0}{\partial z}\frac{\partial z}{\partial y}\frac{\partial y}{\partial x1}$$



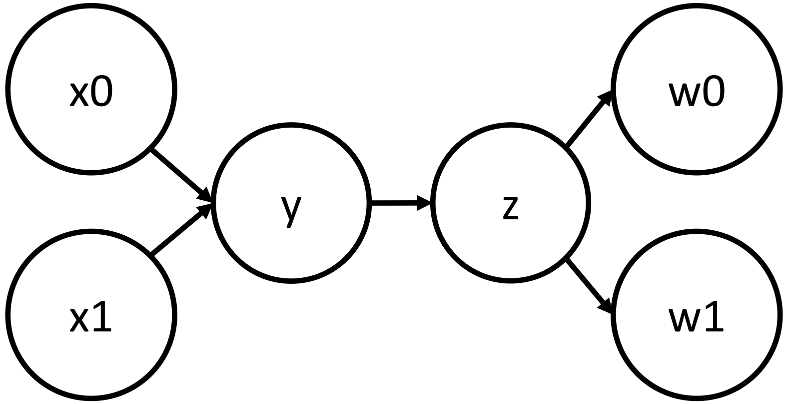
$$\frac{\partial w1}{\partial x0} = \frac{\partial w1}{\partial z}\frac{\partial z}{\partial y}\frac{\partial y}{\partial x0}$$



$$\frac{\partial w1}{\partial x1} = \frac{\partial w1}{\partial z}\frac{\partial z}{\partial y}\frac{\partial y}{\partial x1}$$

AD step-by-step. Forward Mode

```
dx0dx = {1, 0}
dx1dx = {0, 1}
y = f(x0, x1)
dydx = df(x0, dx0dx, x1, dx1dx)
z = g(y)
dzdx = dg(y, dydx)
w0, w1 = l(z)
dw0dx, dw1dx = dl(z, dzdx)
```



AD step-by-step. Reverse Mode

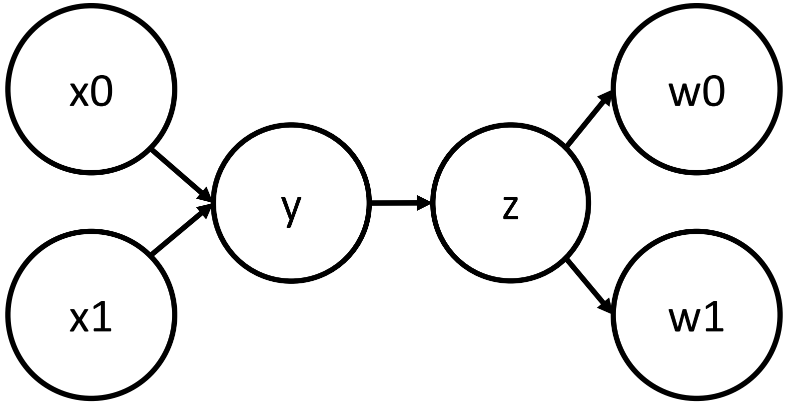
```
y = f(x0, x1)
z = g(y)
w0, w1 = l(z)

dwdw0 = {1, 0}
dwdw1 = {0, 1}

dwdz = dl(dwdw0, dwdw1)

dwdy = dg(y, dwdz)

dwx0, dwx1 = df(x0, x1, dwdy)
```



AD. Cheap Gradient Principle

- The computational graph has **common subpaths** which can be precomputed
- If a function has a single input parameter, no mater how many output parameters, **forward mode** AD generates a **derivative** that has the **same time complexity** as the original function
- More importantly, if a function has a **single output** parameter, **no matter how many input** parameters, reverse mode AD generates **derivative** with the **same time complexity** as the original function.

AD. Implementation Approaches

AD tools can be categorized by how much work is done before program execution

- *Tracing/Operator Overloading/Dynamic Graphs/Taping* -- Records the linear sequence of computation operations at runtime into a tape
- *Source Transformation* -- Constructs the computation graph and produces a derivative at compile time

Automatic vs Symbolic Differentiation

$$f(x) = e^{e^{e^{e^{e^x}}}}$$

Symbolic via Wolfram Alpha

$$\frac{d}{dx}\left(e^{e^{e^{e^{e^x}}}}\right) = e^{x+e^{e^{e^{e^x}}}+e^{e^{e^x}}+e^{e^x}+e^x}$$

Figure out the analytical fn

Handcode

```cpp
// f(x)=e^(e^(e^(e^(e^x))))
#include <cmath>
double f (double x, int N=5) {
  double result = x;
  for (unsigned i = 0; i < N; i++)
    result = std::exp(result);
  return result;
}
```

AD

Handcode

```cpp
double f_dx(double x, int N=5) {
  double result = x;
  double d_result = 1;
  for (unsigned i = 0; i < N; i++) {
    result = std::exp(result);
    d_result *= result;
  }
  return d_result;
}
```

AD. Gradient Generation

- Control Flow and Recursion fall naturally in forward mode.
- Extra work is required for reverse mode in reverting the loop and storing the intermediaries **in general**.

```
double f_reverse (double x, int N=5) {
  double result = x;
  std::stack<double> results;
  for (unsigned i = 0; i < N; i++) {
    results.push(result);
    result = std::exp(result);
  }
  double d_result = 1;
  for (unsigned i = N; i; i--) {
    d_result *= std::exp(results.top());
    results.pop();
  }
  return d_result;
}
```

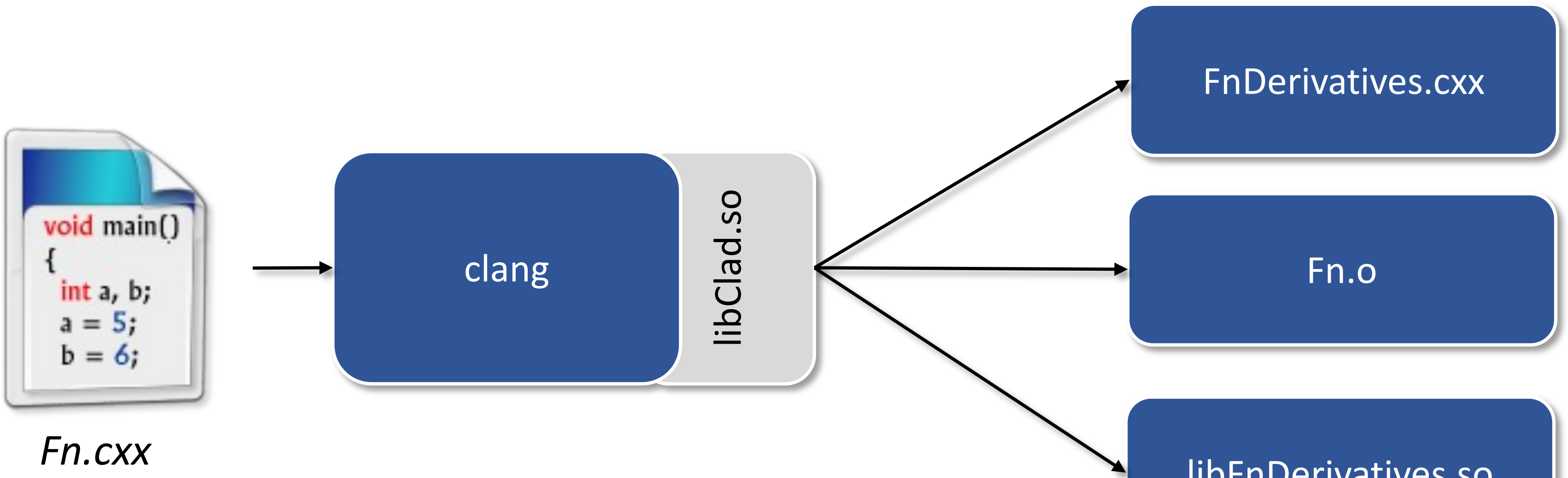
Cla∂


LLVM
Compiler Infrastructure
COMPLEXITY OF COMPILER DESIGN
POWER
ON OFF
BRIGHT
"NFC" COMMITS
ADDRESS SANITIZER

Clad. Design Principles

- ~~Look Ma’ I can make a compiler generate a derivative!~~
- Make AD a first-class citizen to a high-performance language such as C++
- Support idiomatic C++ (compile-time programming such as constexpr, consteval)
- Infrastructure reuse – employ our compiler engineering skills
- Lower contribution entry barrier
- **Diagnostics**

High-Level Data Flow



- Compiler module, very similar to the template instantiator by idea and design.
- Generates f' of any given f using source transformation at compile time.

Programming Model

```cpp
// clang++ -fplugin libclad.so —Iclad/include ...

// Necessary for clad to work include
#include "clad/Differentiator/Differentiator.h"
double pow2(double x) { return x * x; }
double pow2_darg0(double);

int main() {
  auto dfdx = clad::differentiate(pow2, 0);

  // Function execution can happen in 3 ways:
  // 1) Using CladFunction::execute method.
  double res = cladPow2.execute(1);

  // 2) Using the function pointer.
  auto dfdxFnPtr = cladPow2.getFunctionPtr();
  res = cladPow2FnPtr(2);

  // 3) Using direct function access through fwd declaratic
  res = pow2_darg0(3);
  return 0;
}
```

The body will be generated by Clad

Result via Clad's function-like wrapper

Result via function pointer call

Result via function forward declaration

Programming Model. Differential Operators

User-defined substituti

```
// MyCode.h
float custom_fn(float x);

namespace custom_derivatives {
  float custom_fn_dx(float x) {
    return x * x;
  }
}

float do_smth(float x) {
  return x * x + custom_fn(x);
}

int main() {
  clad::differentiate(do_smth, 0).execute(2); // will return 6
  return 0;
}
```

Vassil Vassilev/ACAT14 1.09.14

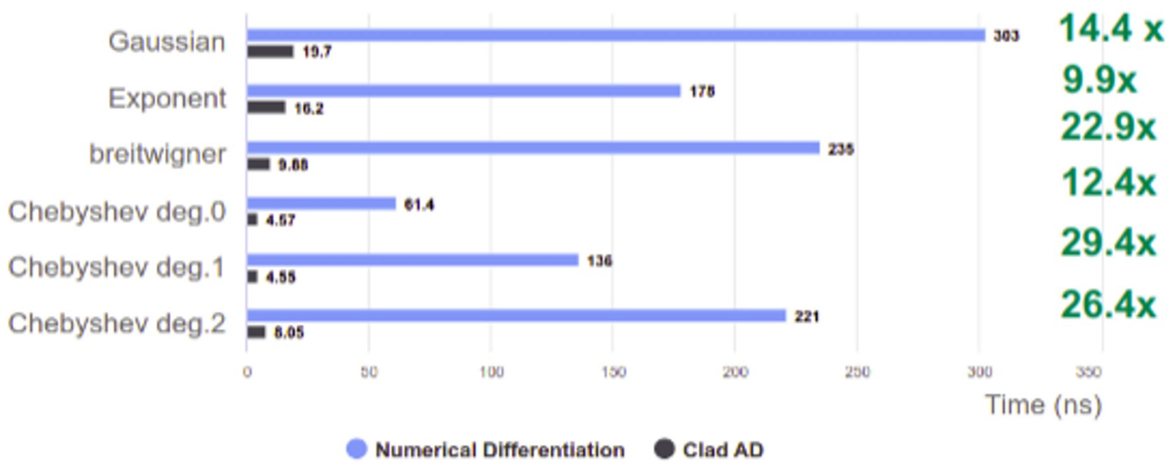
```
template <typename T1, typename T2>
CUDA_HOST_DEVICE ValueAndPushforward<decltype(::std::pow(T1(), T2())),
                                     decltype(::std::pow(T1(), T2()))>
pow_pushforward(T1 x, T2 exponent, T1 d_x, T2 d_exponent) {
  auto val = ::std::pow(x, exponent);
  auto derivative = (exponent * ::std::pow(x, exponent - 1)) * d_x;
  // Only add directional derivative of base^exp w.r.t exp if the directional
  // seed d_exponent is non-zero. This is required because if base is less than or
  // equal to 0, then log(base) is undefined, and therefore if user only requested
  // directional derivative of base^exp w.r.t base -- which is valid --, the result would
  // be undefined because as per C++ valid number + NaN * 0 = NaN.
  if (d_exponent)
    derivative += (::std::pow(x, exponent) * ::std::log(x)) * d_exponent;
  return {val, derivative};
}
```

Clad in High-Energy Physics

A data analysis framework used to process EB data

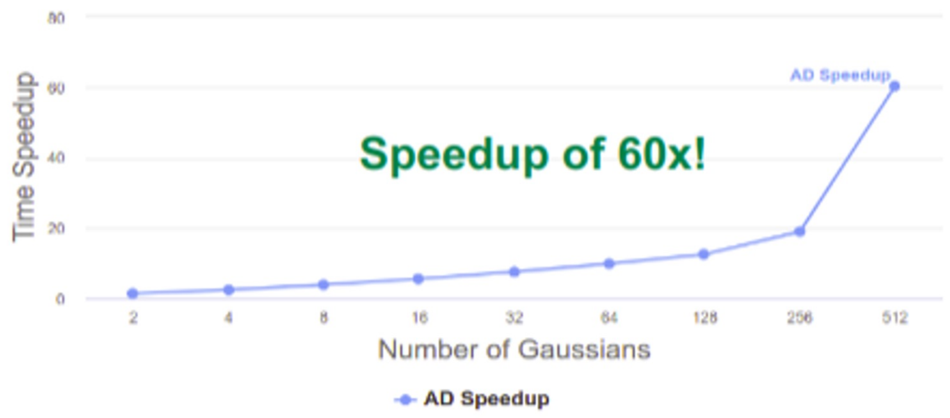
- We have seen some promising results (in ROOT) already!

Performance Comparison of Generation in TFormula



TFormula benchmarks of gradient generation time from numerical differentiation and clad AD.

Performance Speedup of a Multi-Gaussian Fit (10000 bins)

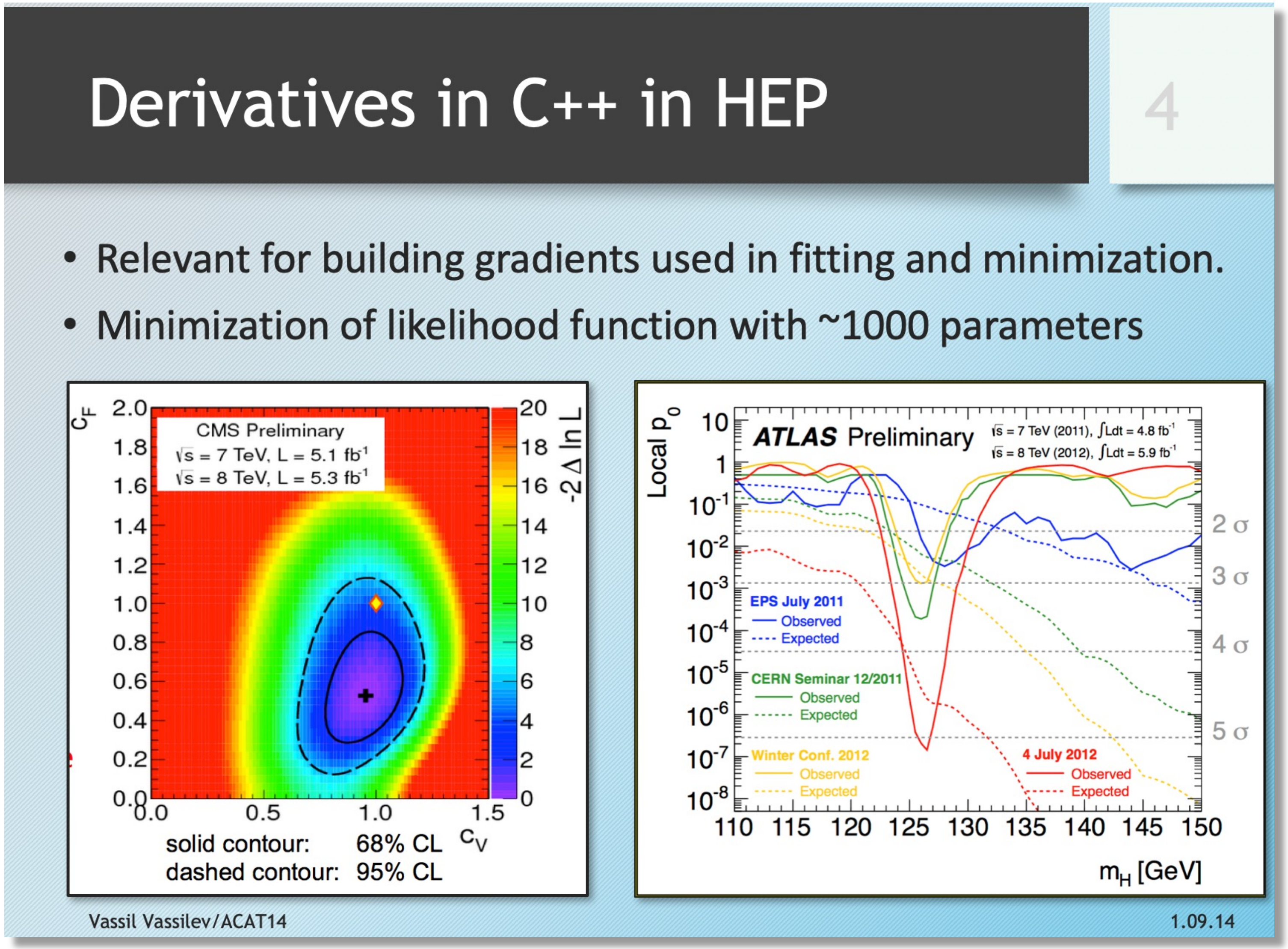


TF1 based benchmarks. TF1 is the TFormula fitting interface for fitting histograms.

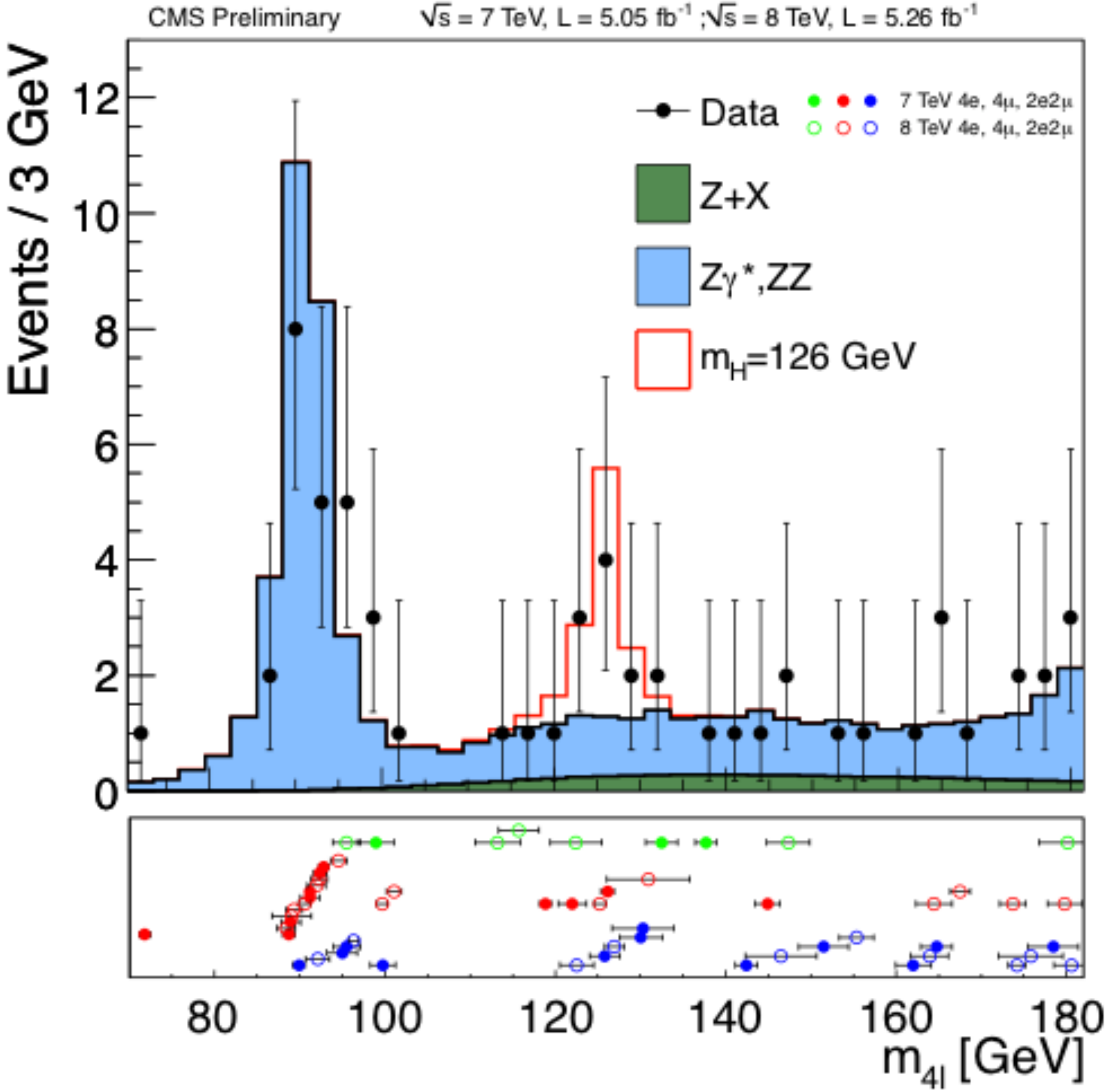
There and Back Again

Social Engineering, Progress, Social Engineering…

In the meanwhile: Cling, ROOT6, C++ Modules, IPCC-ROOT, compiler-research.org, Clang-Repl …



Statistical Modelling



Automatic Differentiation in RooFit

G. Singh

RooFit represents all mathematical formulae as RooFit objects which are then brought together into a compute graph. This compute graph makes up a model on which further data analysis is run.

Math Notations		RooFit Object
variable	x	RooRealVar
function	$f(x)$	RooAbsReal
PDF	$f(x)$	RooAbsPdf
space point	$\hat{x}$	RooArgSet
integral	$\int_a^b f(x)$	RooRealIntegral
list of space points	$\hat{x}_1, \hat{x}_1, \hat{x}_1 \ldots$	RooAbsData

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Gaussian Probability Distribution Function (pdf)

```
//Obj represents f(x) here
RooGaussian obj(x, mu,
sigma);
```

Equivalent Code in C++ with RooFit

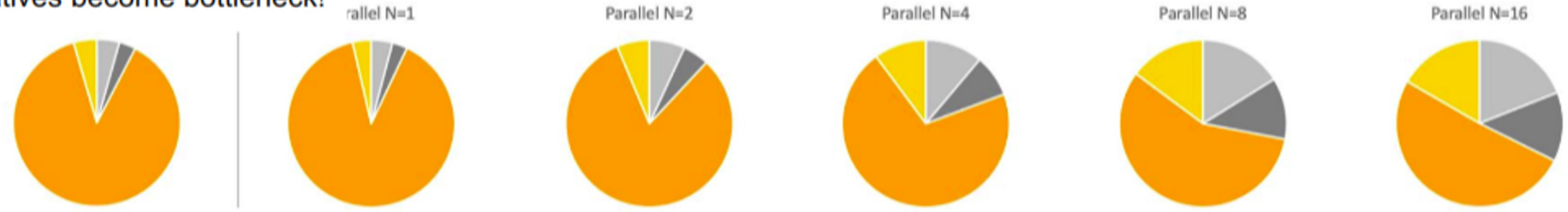
Programmers/users know this relationship. But how do we connect these two together when a connection is not obvious in code?

Bottlenecks

- One goal - Make RooFit Faster. Results from a Higgs-combination fit:

	serial old		parallel N=1		parallel N=2		parallel N=4		parallel N=8		parallel N=16	
	setup_roofit	313	setup_roofit	327	setup_roofit	315	setup_roofit	315	setup_roofit	312	setup_roofit	327
	minuit_init	230	minuit_init	231	minuit_init	231	minuit_init	231	minuit_init	231	minuit_init	231
	gradient_calc	6289	gradient_calc	7102	gradient_calc	3734	gradient_calc	1997	gradient_calc	1107	gradient_calc	879
	line_search	323	line_search	287	line_search	287	line_search	287	line_search	287	line_search	287

Derivatives become bottleneck!



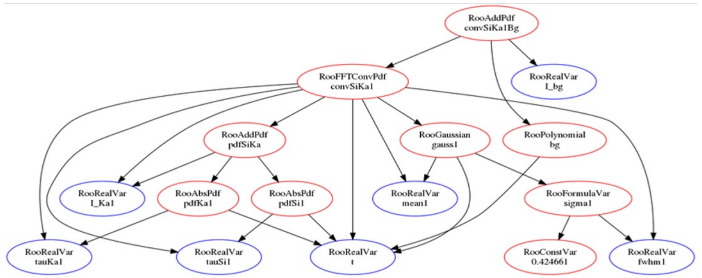
ICHEP 2022 - Zeff Wolffs - https://agenda.infn.it/event/28874/contributions/169205/attachments/93887/129094/ICHEP_RooFit_ZefWolffs.pdf

- Good results, but still use numerical differentiation.
- Potential next step – use Automatic Differentiation to compute the gradients.

Image ref: Automatic Differentiation of Binned Likelihoods With Roofit and Clad - *Garima Singh, Jonas Rembser, Lorenzo Moneta, Vassil Vassilev*, ACAT 2022

Automatic Differentiation in RooFit

What that we want to differentiate



Some way to expose differentiable properties of the graph as code.

C++ code the AD tool can understand

C++ code the AD tool can understand

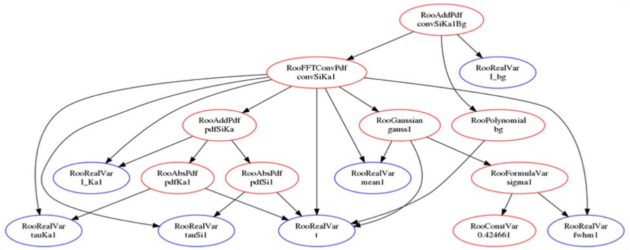
The AD tool

Cla∂

Derivative code of the model!

Automatic Differentiation in RooFit. Approach

What that we want to differentiate



Define 2 Functions in RooFit

C++ code the AD tool can understand

Stateless function enabling differentiation of each class.

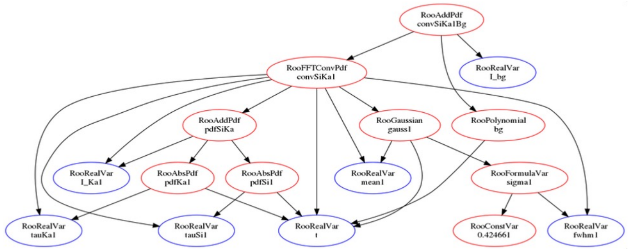
```
double ADDetail::gauss(double x, double mean, double sigma) {
 const double arg = x - mean;
 const double sig = sigma;
 return std::exp(-0.5 * arg * arg / (sig * sig));
}
```

The “glue” function enabling graph squashing.

```
void RooGaussian::translate(...) override {
 result = "ADDetail::gauss(" +
                              _x->getResult() +
                              " ," + _mu->getResult() +
                              " ," + _sigma->getResult() + ")";
}
```

Automatic Differentiation in RooFit. Approach

What that we want to differentiate



Define 2 Functions in RooFit

C++ code the AD tool can understand

`RooGaussian::evaluate()`

The RooFit call to evaluate a gaussian

- Bookkeeping

& caching

`ADDetail::gauss(x, mu, sig)`

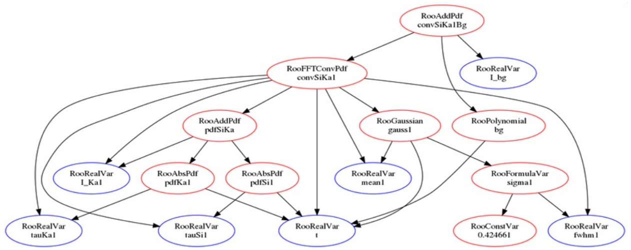
The equivalent code generated

`ADDetail::gauss(x, mu, sig) / ADDetail::gaussIntegral(...)`

The equivalent code generated
(given the class supports analytical integrals)

Automatic Differentiation in RooFit. Approach

What that we want to differentiate



‘Squash’ the graph into code

`Roo*::translate()`

C++ code the AD tool can understand

C++ code the AD tool can understand

+

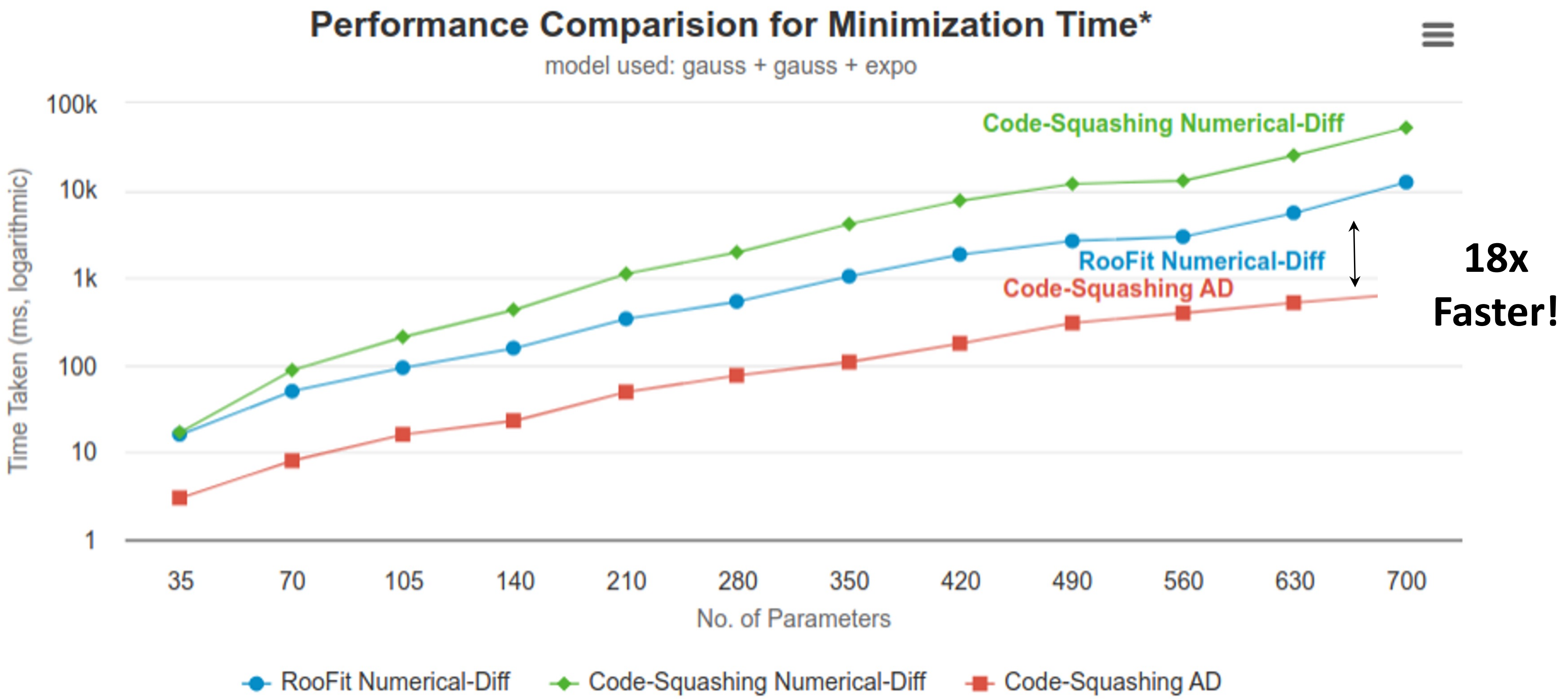
The AD tool

Cla∂

=

Derivative code of the model!

Basic RooFit Example With Binned Fit of Analytical Shapes



Tested on ROOT master as of May 2023.

*Excludes the seed generation time

Large Analysis Benchmark Describing Workflows in HEP

*Fitting Time (s)**

N Channels	RooFit ND	RooFit AD	Speedup
1	0.03	0.01	2x
5	1.19	0.26	2.5x
10	2.22	0.36	5.2x
20	7.38	1.17	5.3x

Link to paper: https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PAPERS/HIGG-2018-51/

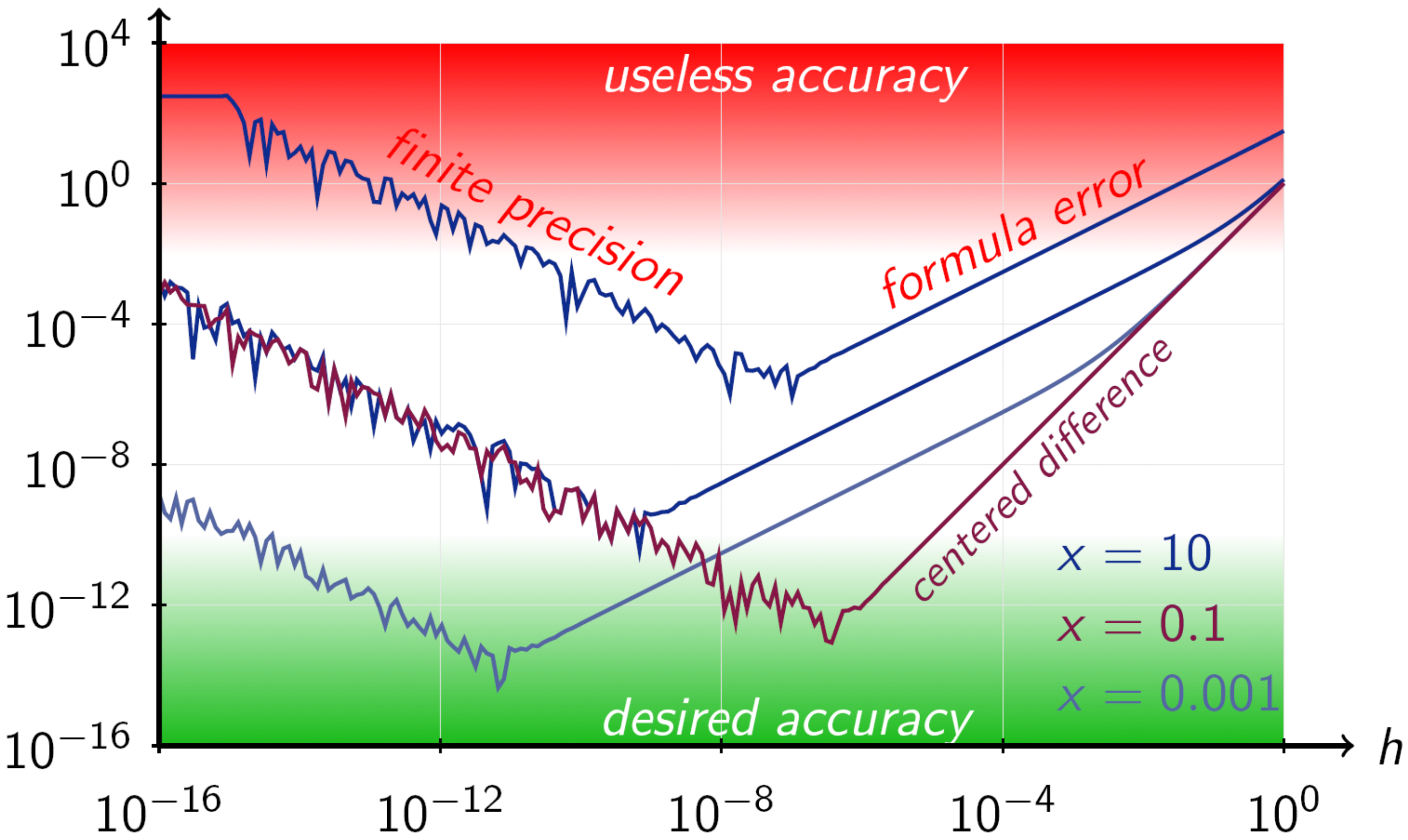
***Excludes the seed generation time, more info**

Large Analysis Benchmark Compile Times

Mode	JIT	gcc10	clang-13
-O0	16s	1.15s	0.82
-O1	17s	4.46s	6.00s
-O2	17s	9.24s	8.57s
-O3	17s	10.69s	8.88s

The generated code is suboptimal for the optimization pipelines.
We know how to fix this.

Floating Point Error Analysis



Floating point errors

	Value	Error
Input number:	0.3	-
Representation in float:	0.30000001192092895508	1.19e-08
Representation in double:	0.29999999999999998890	1.11e-17

Let's try a simple addition operation: 0.3 + 0.3

	Value	Error
Operation output:	0.6	-
Representation in float:	0.60000002384185791016	2.38e-08
Representation in double:	0.59999999999999997780	2.22e-17

Link to code for these numbers

Classical Formula for Error Estimation

The maximum floating-point error (h_{max}) in x as allowed by IEEE is $|x| \cdot \varepsilon_M$, where ε_M is the machine epsilon.

$$\Delta f_x \approx |f'(x) \cdot |x| \cdot \varepsilon_M|$$

The general representation of the error estimation formula is:

First order partial derivatives

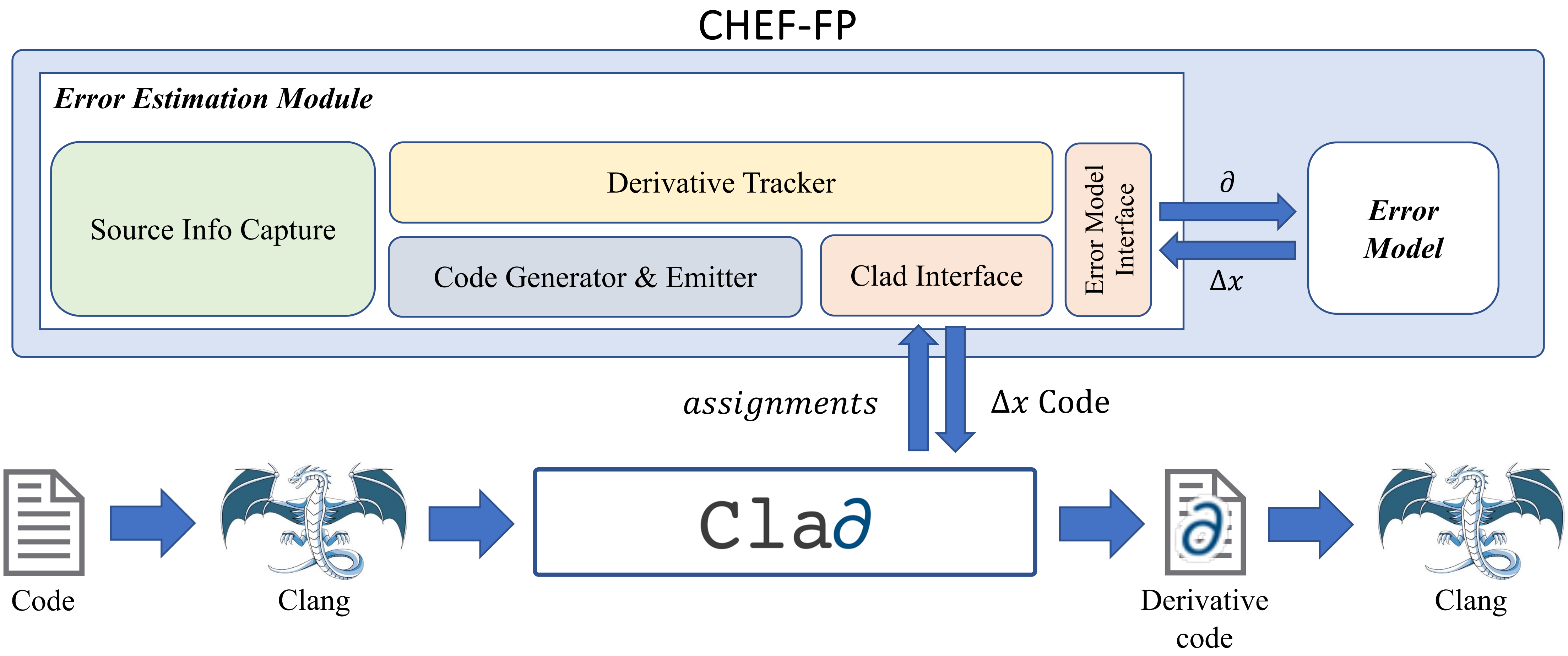
Input and intermediate variables

Machine Epsilon

$$\Delta f \equiv \sum_{i=0}^{n} \left| \underbrace{\frac{\partial f}{\partial x_i}}_{\text{Derivative w.r.t that variable}} \cdot \underbrace{|x_i| \cdot \varepsilon_M}_{\text{Error in variable}} \right|$$

Link to paper: https://arxiv.org/abs/2304.06441

Clad in FP Error Analysis: CHEF-FP



CHEF-FP Usage

```
double func(double x, double y) {
  double z = x + y;
  return z;
}
```

```
#include "clad/Differentiator/Differentiator.h"
#include "../PrintModel/ErrorFunc.h"

// Call CHEF-FP on the function
auto df = clad::estimate_error(func);
```

Execute the CHEF-FP object to get the error

```
double x = 1.95e-5, y = 1.37e-7;
double dx = 0, dy = 0;
double fp_error = 0;

df.execute(x,y, &dx, &dy, fp_error);

std::cout << "FP error in func: " << fp_error;
// FP error in func: 8.25584e-13

// Print mixed precision analysis results
clad::printErrorReport();
```

USER GENERATED CODE

```
void func_grad(double x, double y,
    clad::array_ref<double> _d_x,
    clad::array_ref<double> _d_y,
    double &_final_error) {
  double _d_z = 0, _delta_z = 0, _EERepl_z0;
  double z = x + y;
  _EERepl_z0 = z;
  double func_return = z;
  _d_z += 1;
  * _d_x += _d_z;
  * _d_y += _d_z;
  _delta_z +=
    clad::getErrorVal(_d_z, _EERepl_z0, "z");
  double _delta_x = 0;
  _delta_x +=
    clad::getErrorVal(* _d_x, x, "x");
  double _delta_y = 0;
  _delta_y +=
    clad::getErrorVal(* _d_y, y, "y");
  _final_error +=
    _delta_y + _delta_x + _delta_z;
}
```

The function generated by CHEF-FP to estimate the errors

AUTO GENERATED CODE

Plans

- Grey box AD
  - Enhance the pushforward/pullback mechanisms to avoid common AD pitfalls
- Further advancements and applications on floating point error estimation
  - Controlling the error limits helps the energy efficiency of algorithms
- Robust activity analysis
- A research platform AD in C/C++
  - Combines all power of Clang Static Analyzer, LLVM Optimization Passes, Control Flow Graphs

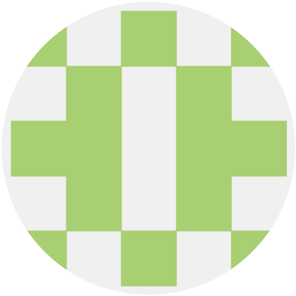
Violeta Ilieva
Initial prototype, Forward Mode

Vassil Vassilev
Conception, Mentoring, Bugs, Integration, Infrastructure

Martin Vassilev
Forward Mode, CodeGen

Alexander Penev
Conception, CMake, Demos, Jupyter

Aleksandr Efremov
Reverse Mode



Jack Qui
Hessians

Roman Shakhov
Jacobians

Oksana Shadura
Infrastructure, Co-mentoring

Pratyush Das
Infrastructure

Garima Singh
FP error estimation, RooFit, Bugs

Ioana Ifrim
CUDA AD

Parth Arora
Initial support classes, functors, pullbacks

Baidyanath Kundu
Array Support, ROOT integration

Vaibhav Thakkar
Forward Vector Mode

Thank you!