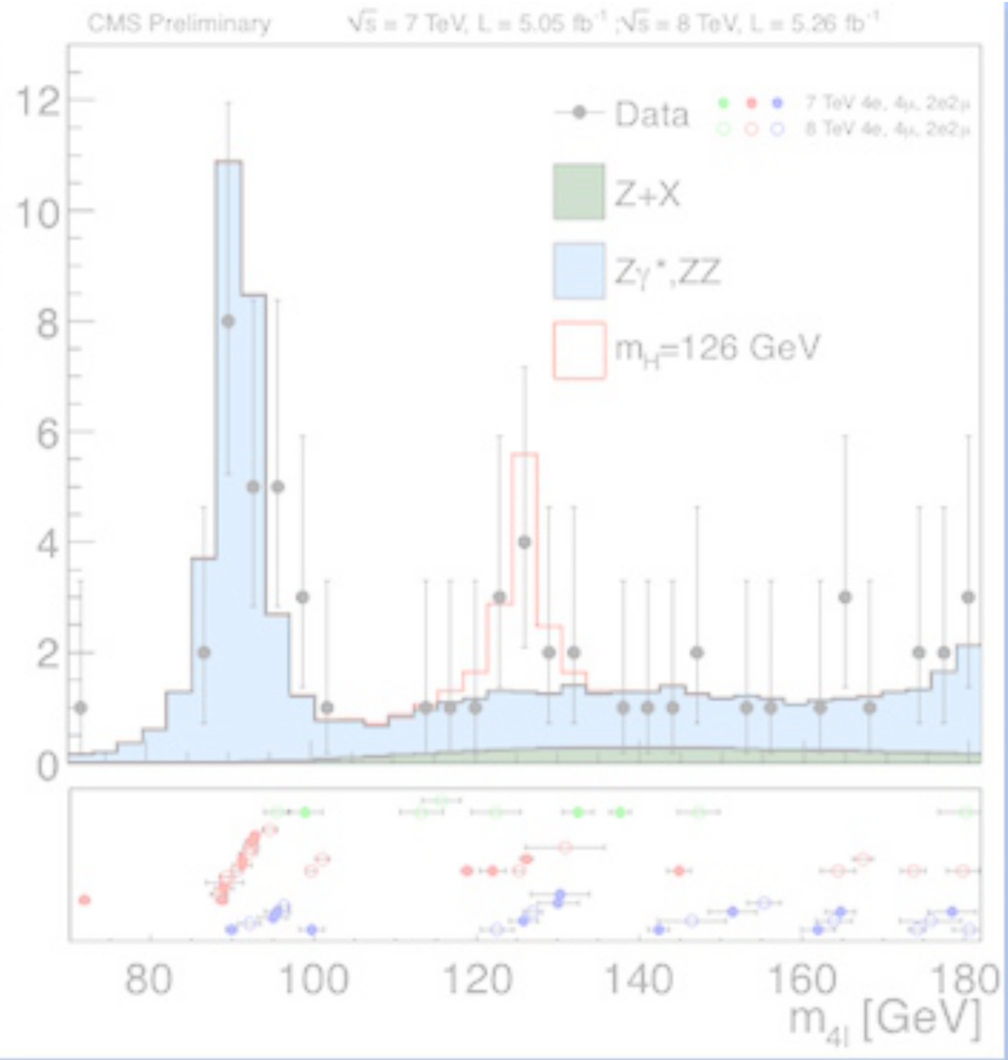


IPCC ROOT
Princeton/Intel Parallel Computing Center

Progress Report

Vassil Vassilev, PhD

18.04.2017

Status 29.03-2017 — 18.04.2017

- Enable vectorization for ROOT through VecCore
    - PR#393 (+116 −29 💬 30); PR#497 (+158 −31 💬 2);
- Enables contributors to submit code employing the vectorization capabilities of the CPUs.
- Misc
    - more…

clad: Automatic Differentiation Library in ROOT

- Foreseen piece of work for Q3 of this year
- clad already got some interest by Intel people

clad: Integration plan

- Enable the use of the library within ROOT, connecting it to the cling interpreter (also Clang/LLVM based), etc.

- Update to the latest compiler versions, debug, etc.

- Integrate AD into specific non-trivial examples in Minuit (used for numerical minimization in ROOT) and TMVA (multivariate analysis) in ROOT.

- Benchmark those applications

clad: Integration plan. Scope

- In this first step we are not dealing with OpenCL or parallelization. The latter still has to come from the end user applications.
- After Q3 (Y1) we would likely use this infrastructure in RooFit in Y2. RooFit is ROOT's the data modeling and fitting package which is being reengineered.

clad: In a Nutshell

clad neither employs the slow symbolic nor inaccurate numerical differentiation. It uses the fact that every computer program can be divided into a set of elementary operations (-,+,*,/) and functions (sin, cos, log, etc). By applying the chain rule repeatedly to these operation, derivatives of arbitrary order can be computed.

C/C++ to C/C++ language transformer implementing the chain rule from differential calculus. For example:

```
constexpr double MyPow(double x) { return x*x; }
```

↓

```
constexpr double MyPow_darg0(double x) { return (1. * x + x * 1.); }
```

clad: Advantages over Numerical Differentiation

```
#include <cmath>

double MyCos(double x) { return std::cos(x); }
double MySin(double x) { return std::sin(x); }
constexpr double MyPow(double x) { return x*x; }

typedef double (*SigF)(double);

// Simple finite differences numerical differentiator.
double derive(SigF f, double a, double h=0.01, double epsilon = 1e-7){
  double f1 = (f(a+h)-f(a))/h;
  double f2 = 0.;
  while (1) {
    h /= 2.;
    f2 = (f(a+h)-f(a))/h;
    double diff = std::abs(f2-f1);
    f1 = f2;
    if (diff < epsilon)
      break;
  }
  return f2;
}
```

clad: Advantages over Numerical Differentiation

```
#include <cmath>

double MyCos(double x) { return std::cos(x); }
double MySin(double x) { return std::sin(x); }
constexpr double MyPow(double x) { return x*x; }

// The derivatives are provided by clad but hardcoded here for simplicity, i.e.
// you can run this example without installing clad.
double MyCos_darg0(double x) { return -std::sin(x) * (1.); }
double MySin_darg0(double x) { return std::cos(x) * (1.); }
constexpr double MyPow_darg0(double x) { return (1. * x + x * 1.); }
```

clad: Advantages over Numerical Differentiation

```
// No clad, using the simple numerical differentiator
int main () {
  printf("MyCos' at 30 is %f\n", derive(MyCos, 30));
  // For every point we need to iterate :( This causes
  // not only slow execution but precision loss!
  printf("MyCos' at 31 is %f\n", derive(MyCos, 31));
  printf("MySin' at 30 is %f\n", derive(MySin, 30));

  // Even if MyPow is a compile-time foldable we still loop!
  printf("MyPow' at 2 is %f\n", derive(MyPow, 2));


  // From math we know that sinx' = cosx
  // Let's check if this was true.
  if (derive(MySin, 30) == MyCos(30))
    printf("No precision loss!\n");
  else
    printf("Precision loss!\n");

  // Output:
  //  MyCos' at 30 is 0.988032
  //  MyCos' at 31 is 0.404038
  //  MySin' at 30 is 0.154252
  //  MyPow' at 2 is 4.000000
  //  Precision loss!
  return 0;
}
```

Lines of assembly code

	-O0	-O3
gcc 6.1	223	141
clang 4	206	226
icc 17	279	283

```
// Using clad, employing automatic differentiation techniques
int main () {
  printf("MyCos' at 30 is %f\n", MyCos_darg0(30));
  // For every point we just need to call a function pointer!
  printf("MyCos' at 31 is %f\n", MyCos_darg0(31));
  printf("MySin' at 30 is %f\n", MySin_darg0(30));

  // The compile-time foldable MyPow folds away!
  printf("MyPow' at 2 is %f\n", MyPow_darg0(2));


  // From math we know that sinx' = cosx
  // Let's check if this was true.
  if (MySin_darg0(30) == MyCos(30))
    printf("No precision loss!\n");
  else
    printf("Precision loss!\n");

  // Output:
  //  MyCos' at 30 is 0.988032
  //  MyCos' at 31 is 0.404038
  //  MySin' at 30 is 0.154251
  //  MyPow' at 2 is 4.000000
  // No precision loss!

  return 0;
}
```

Lines of assembly code

	-O0	-O3
gcc 6.1	150	63
clang 4	154	65
icc 17	181	129

Thank you!

References:

[1] clad — Automatic Differentiation with Clang, http://llvm.org/devmtg/2013-11/slides/Vassilev-Poster.pdf
[2] clad Official GitHub Repository https://github.com/vgvassilev/clad
[3] clad demos https://github.com/vgvassilev/clad/tree/master/demos
[4] clad showcases https://github.com/vgvassilev/clad/tree/master/test
[5] More automatic differentiation tools http://www.autodiff.org/