Fitting and Modeling in ROOT

L. Moneta, A. Tsang, X. Valls (CERN EP-SFT)

CHEP 2018
23RD INTERNATIONAL CONFERENCE ON COMPUTING IN HIGH ENERGY AND NUCLEAR PHYSICS
9-13 July 2018
National Palace of Culture
Sofia, Bulgaria

Outline

- Introduction
- New developments in TFormula class
- Composition of functions and convolution
- Parallelization via multi-threads and vectorization
- Performance tests
- Conclusions

Introduction: Fitting in ROOT

- Function modeling definition using TF1 and TFormula classes
  - can fit directly ROOT data objects (histograms and graphs)
  - simple and efficient but limited support for complex cases
- Model using RooFit package
  - powerful, can build model of arbitrary complexity
  - support for simultaneous fits
  - automatic normalisation of functions (pdf)
  - can be difficult to use and sometimes performances not optimal
- We will show recent improvements in TF1 and TFormula which make fitting directly in ROOT easier !

TF1 Class in ROOT

- TF1 is the class for defining parametric functions that can be used for fitting
- Can support both function defined directly in C++ code or as an expressions (compiled on the fly using Cling JIT)
  - using a C++ functor (e.g. a lambda):

```
auto myfunc = [](double *x, double *p){ return p[0]*sin(p[1]*x[0]);
TF1 f1("f1",myfunc,xmin,xmax,2);
```

  - using an expression (based on TFormula):

```
TF1 f2("f2","[0]*sin([1]*x)", 0., 10.);
```

New Formula developments

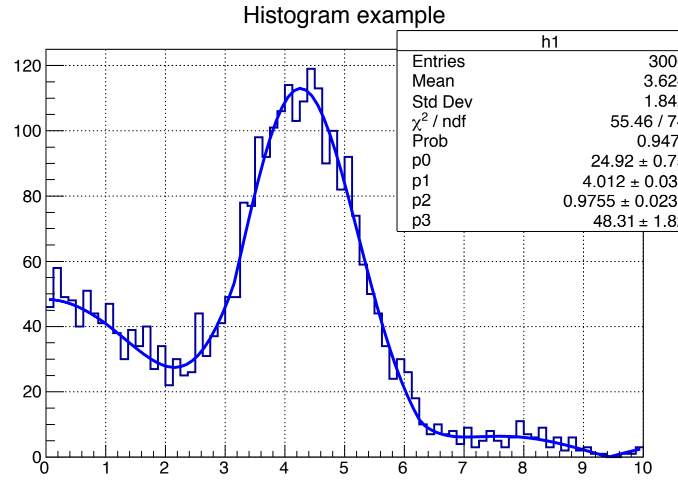
- **Argument parsing**
  - improve parsing when defining the functions in Formula
- **Function composition**
  - support normalised sums of functions
    - e.g. signal + background fits
  - support convolutions

- These new developments make modelling in ROOT much easier

Improved Argument parsing

- Better parameter definitions:
  can set names, define orders, etc..
  - `TF1 ("f1,"gaus( x,[0..2])+ gaus(x, [3],[4],[2])");`
  - `TF1("f1","gaus(x, [Constant],[Mean],[Sigma])");`
- Improved support for multi-dimensional functions
  - `TF2 f2("f2","gaus( x+y, [A],[M],[S])");`
- Function compositions by concatenating formula expressions
  - `TF1 fs("sigma","[0]*x+[1]");`
  - `TF1 f1("f1","gaus(x,[C],[Mean],sigma(x,[A],[B])");`

Normalized Additions

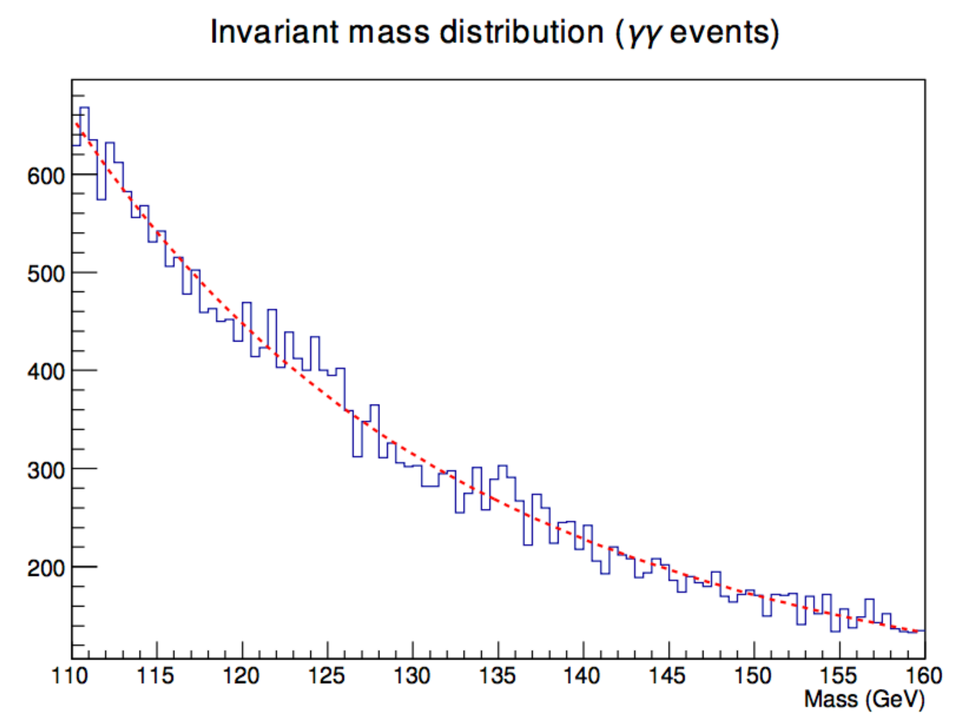


- Many typical HEP fits consists of sums of functions modelling different processes with separate components (e.g. signal + background)
- Fitting often used to determine fractions or number of events for each process
  - from number of events -> cross-sections, discovery significances, etc..
- To fit directly for number of events need to normalise the different model functions
    - otherwise can integrate functions afterwards, but difficult to estimate uncertainties due to correlations (e.g. using TF1::IntegralError)
- **Provide now in ROOT functionality for performing fits with normalised sum:**
  - special operator **NSUM** that can be used to create composite TF1 function objects from formula based functions
    - based on the TF1NormSum class, that can be used for compiled functions

Fitting with normalised sums

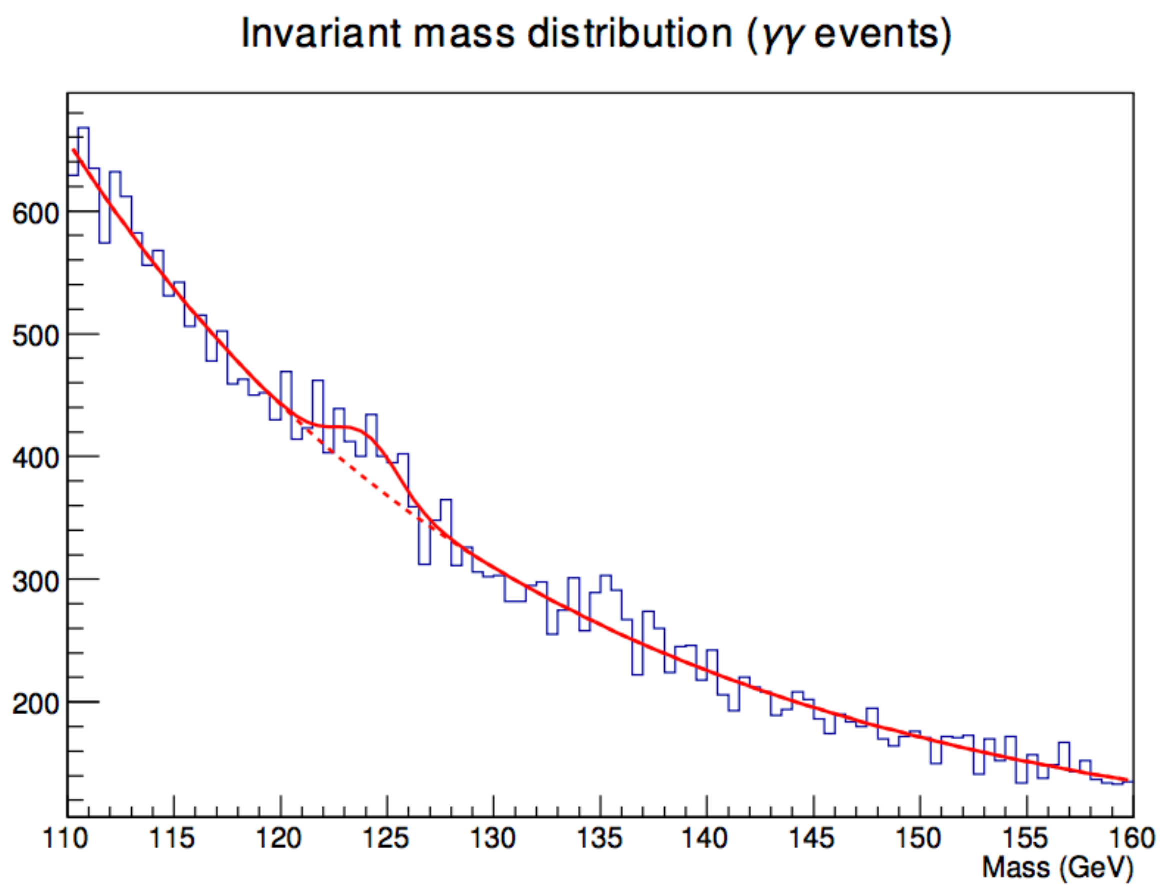
- Example: Gaussian signal plus exponential background fit
- We define first the background as a double exponential

```
TF1 *expo2 = new TF1("expo2","[Constant]*exp([A0]*x + [A1]*x*x)",110,160);
expo2->SetParameters(-8e-2, 2e-4, 5e5);
histo->Fit("expo2", "L"); // binned Likelihood Fit
```



- we then model the full spectrum summing with a Gaussian representing the signal

```
TF1 *model = new TF1("model","NSUM(expo2, gaus)", 110, 160); // new!
model->SetParameter(0, 1e4); // size of background
model->SetParLimits(1, 0, 1e3); // size of signal
model->SetParLimits(4, 115, 140); // mean
model->SetParLimits(5, .3, 6); // sigma
histo->Fit("model", "L");
```



Note that the functions are normalised in the given range. This is [110,160] in this case

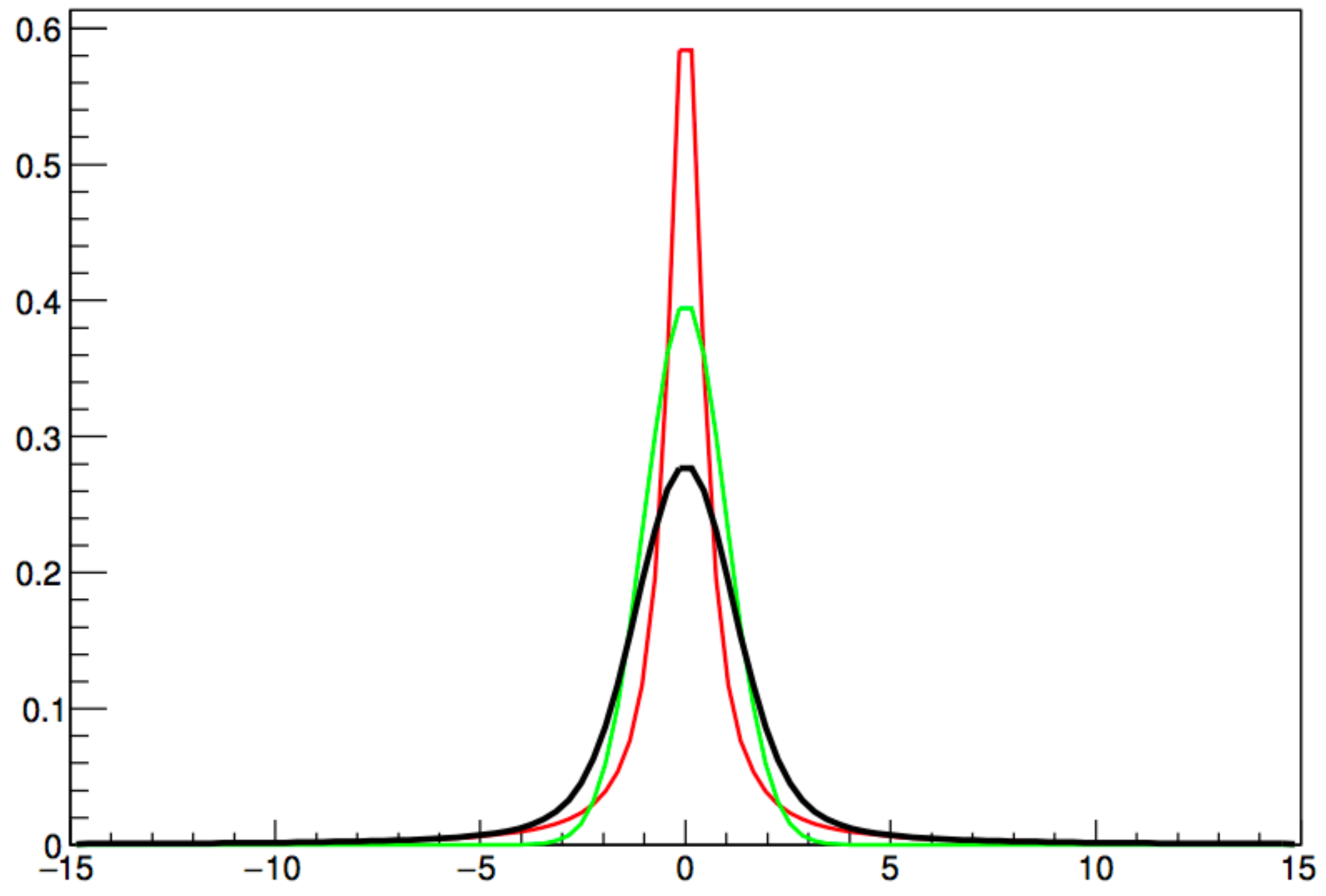
Convolutions

- The observed measured process results from a theoretical distribution f(x) smeared by a resolution function g(x)

$$(f * g)(x) = \int_{-\infty}^{\infty} f(\xi) g(x - \xi) d\xi$$

- Can build in ROOT TF1 function objects representing convolution using the **CONV** operator
  - Example: Breit-Wigner * Gaussian

```
TF1 *bw = new TF1("bw", "breitwigner", -15, 15);
bw->SetParameters(1, 0, 1);
TF1 *mygausn = new TF1("mygausn", "gausn", -15, 15);
mygausn->SetParameters(1, 0, 1);
TF1 *voigt = new TF1("voigt", "CONV(bw, mygausn)", -15, 15);
```



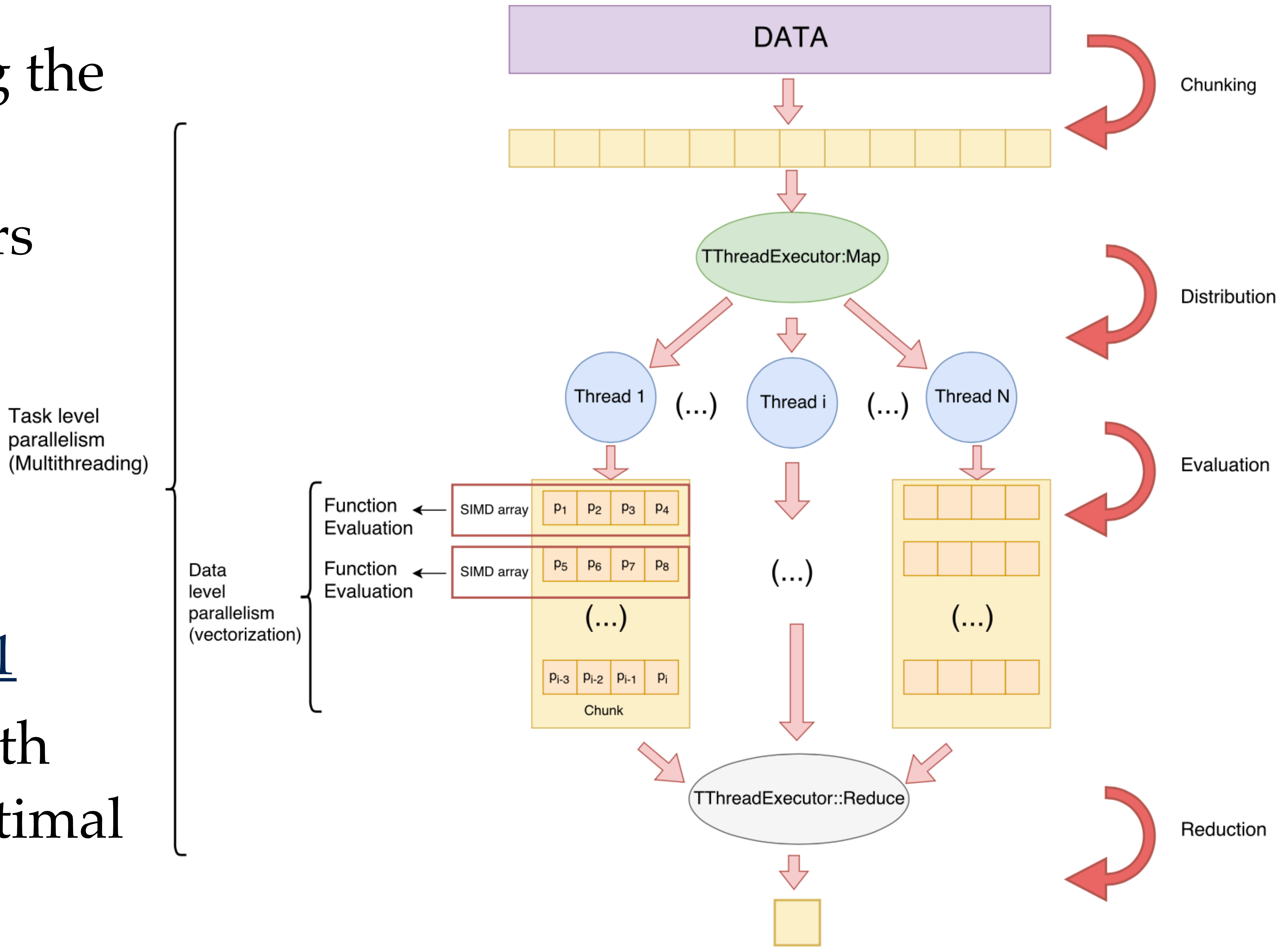
- Convolutions is performed by using FFT (default) or numerical integration.
- The TF1Convolution class is used internally and can be used for compiled functions

Parallelization

- The computation of the fitting objective function (likelihood, least square function, etc..) is computed in parallel by dividing the data points in n-chunks
- Parallelization is performed using the TThreadExecutor class of ROOT
  - task oriented multi-thread Map-Reduce:
    - Map: evaluate chunks of the objective functions by parallel
    - Reduce: sum all computed contributions
- TThreadExecutor provides a very convenient API for multi-threading parallelism in ROOT
  - Map, Reduce, Foreach and chunked mapping with partial reduction
  - used also in TMVA (BDT and Deep Neural network training), I/O and RDataFrame
    - see CHEP18 contribution #346

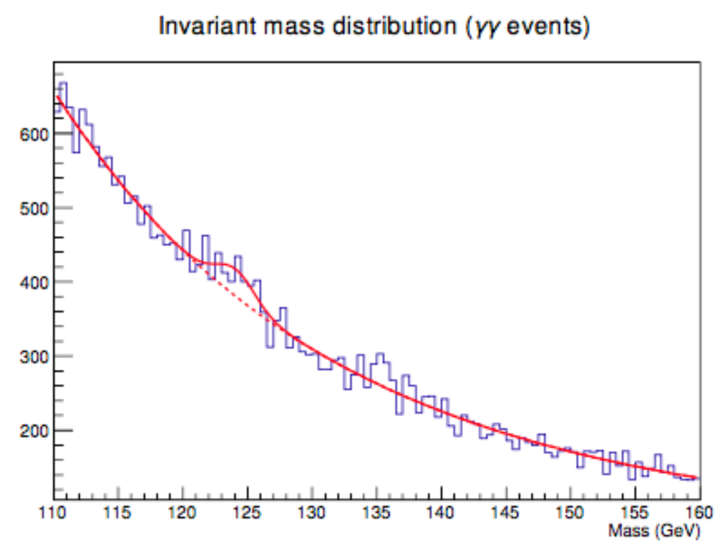
Parallelisation and Vectorization

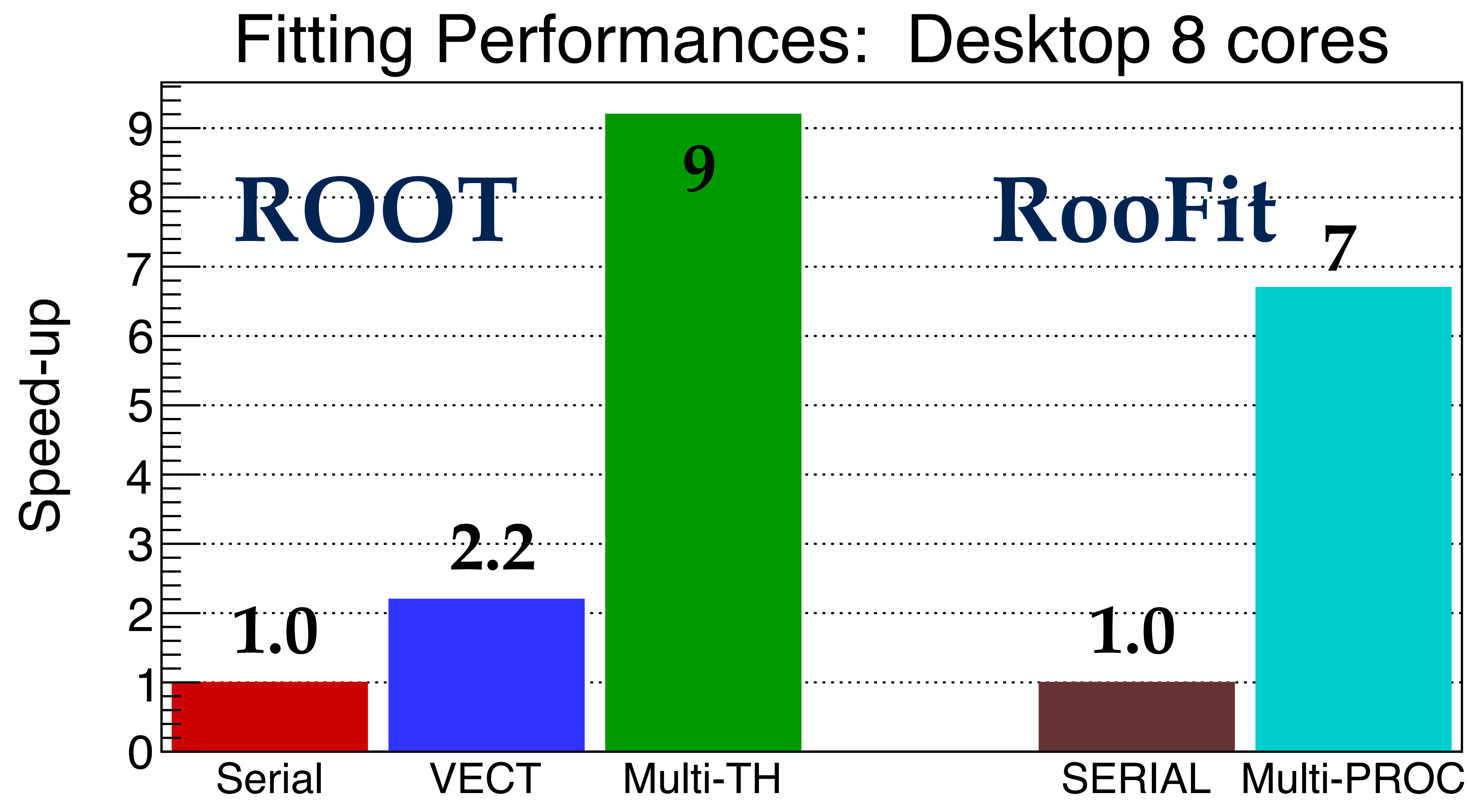
- Model function is evaluated in vectorised mode when computing the fitting objective function
  - organise the input data in vectors (with `ROOT::Double_v`)
  - use vectorised API of TF1 and internal function interfaces
  - TFormula is also vectorised
    - see CHEP18 presentation: #371
- Vectorization can be combined with multithreading parallelism for optimal speed-up



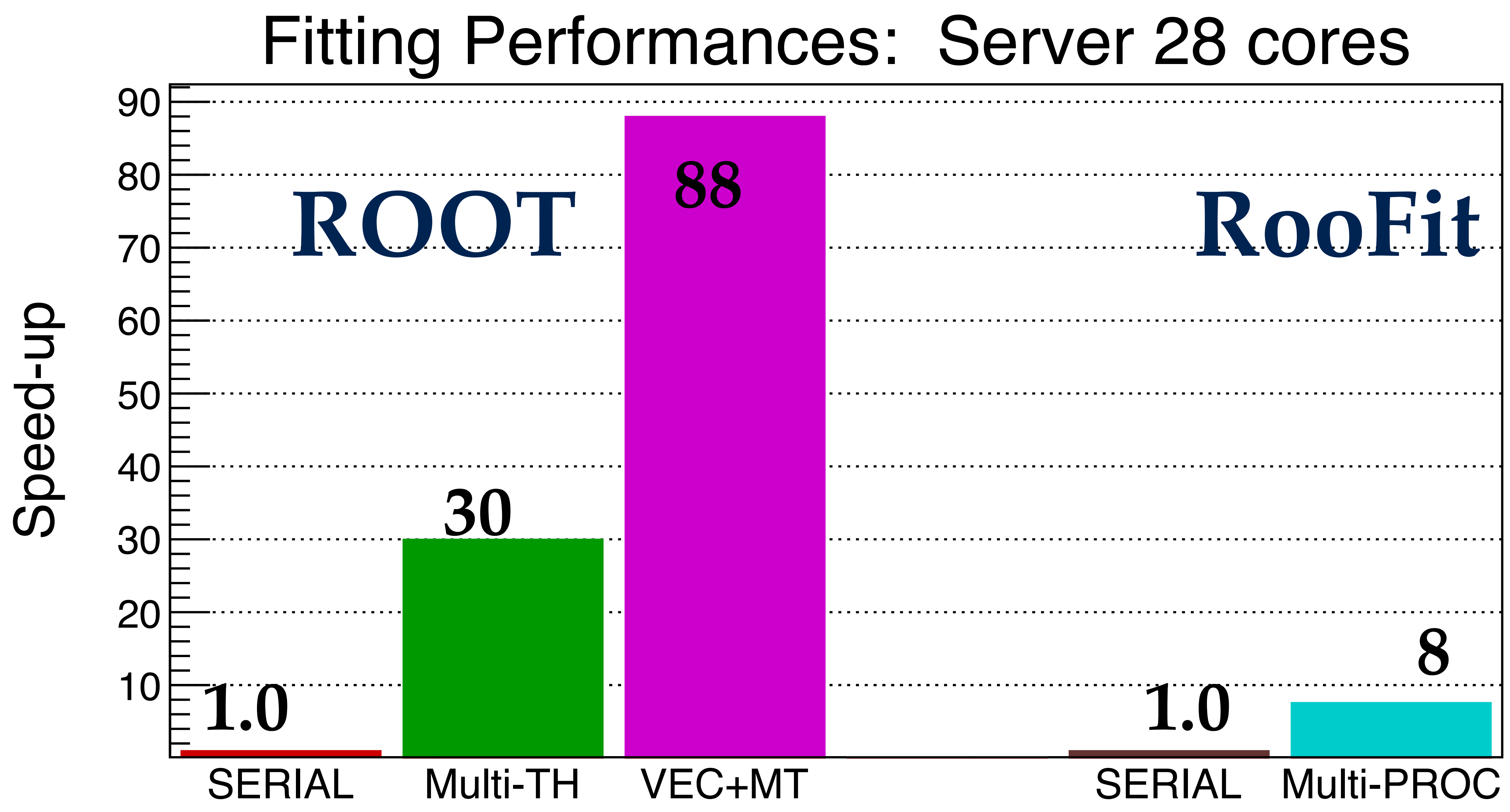
Fitting Performances

- Measure CPU performances in a typical HEP fitting
  - fit invariant mass spectrum to determine significance and location of the signal (e.g. H -> gg)
- Test using ~ 1 M data points in an unbinned fit
  - **ROOT only vs RooFit**





for this fit serial ROOT is also ~ 50% faster



Intel Xeon CPU E5-2683 with 28 physical cores

Future Outlook

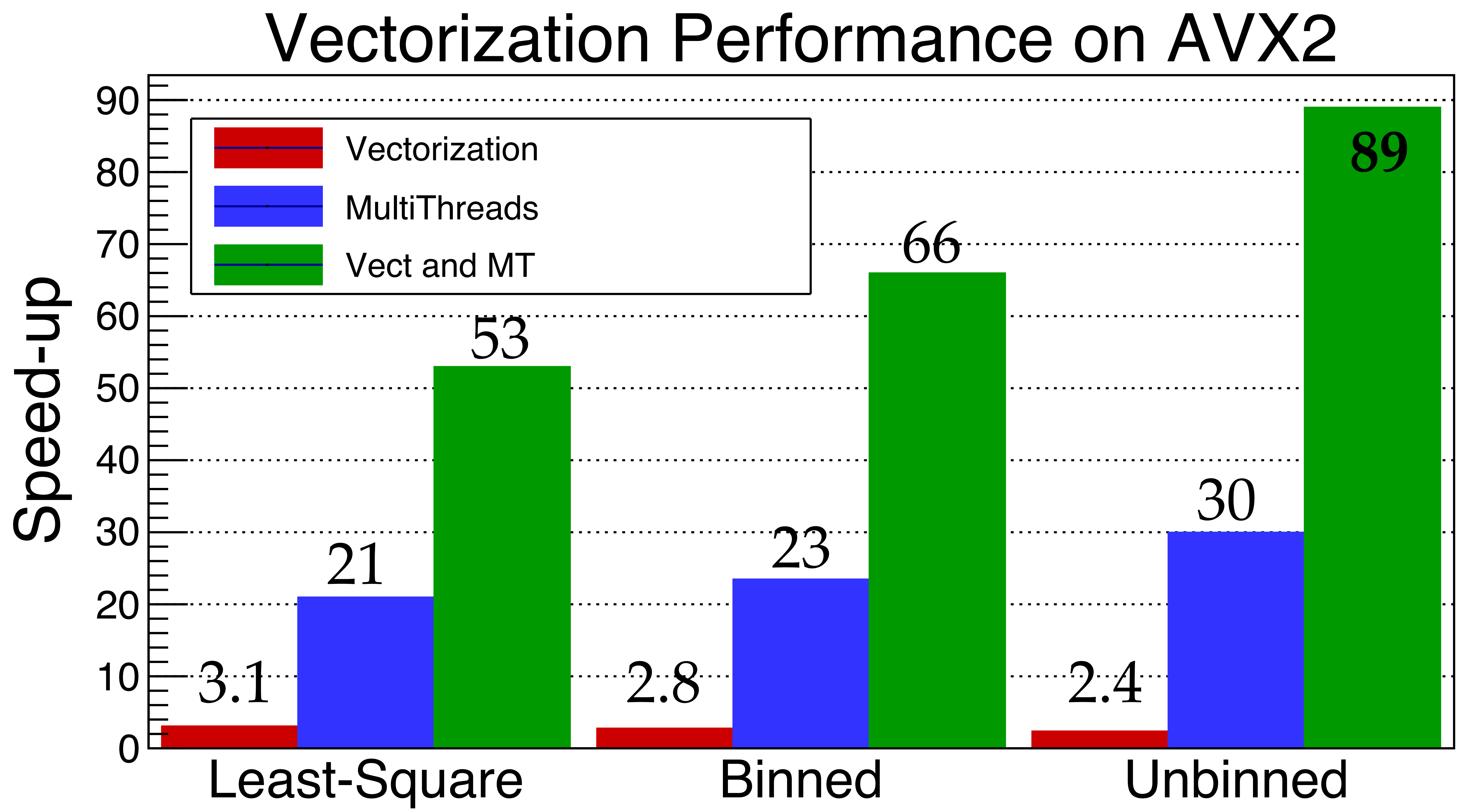
- Improve support for modelling more complex use cases
  - support in ROOT Fitter class constraint fits and simultaneous fits
    - e.g fitting multiple histograms with common functions
- Provide interfaces for fitting new ROOT 7 histograms
- Investigate developing new back-ends for fitting
  - given a model definition (e.g. via a RooWorkspace, or the HistFactory) use alternative implementation back-ends
    - e.g. pure ROOT or based on external packages (Tensorflow)
    - integrate with auto-differentiation for computing gradients
- Porting to GPU using CUDA

Conclusions

- Several improvements applied for defining model fitting functions in ROOT
  - easier to create functions with formula
  - support for convolutions and normalised sums
- Optimal performances in computing likelihood's
  - using parallelisation and vectorisation (also who using TFormula)
- Advantages with respect to other packages, such as Roofit
  - capability of performing bin integrals fits is not available in RooFit
  - better performances and scalability to many cores
- **Users feedback is very much welcomed !**

Fitting Speedups

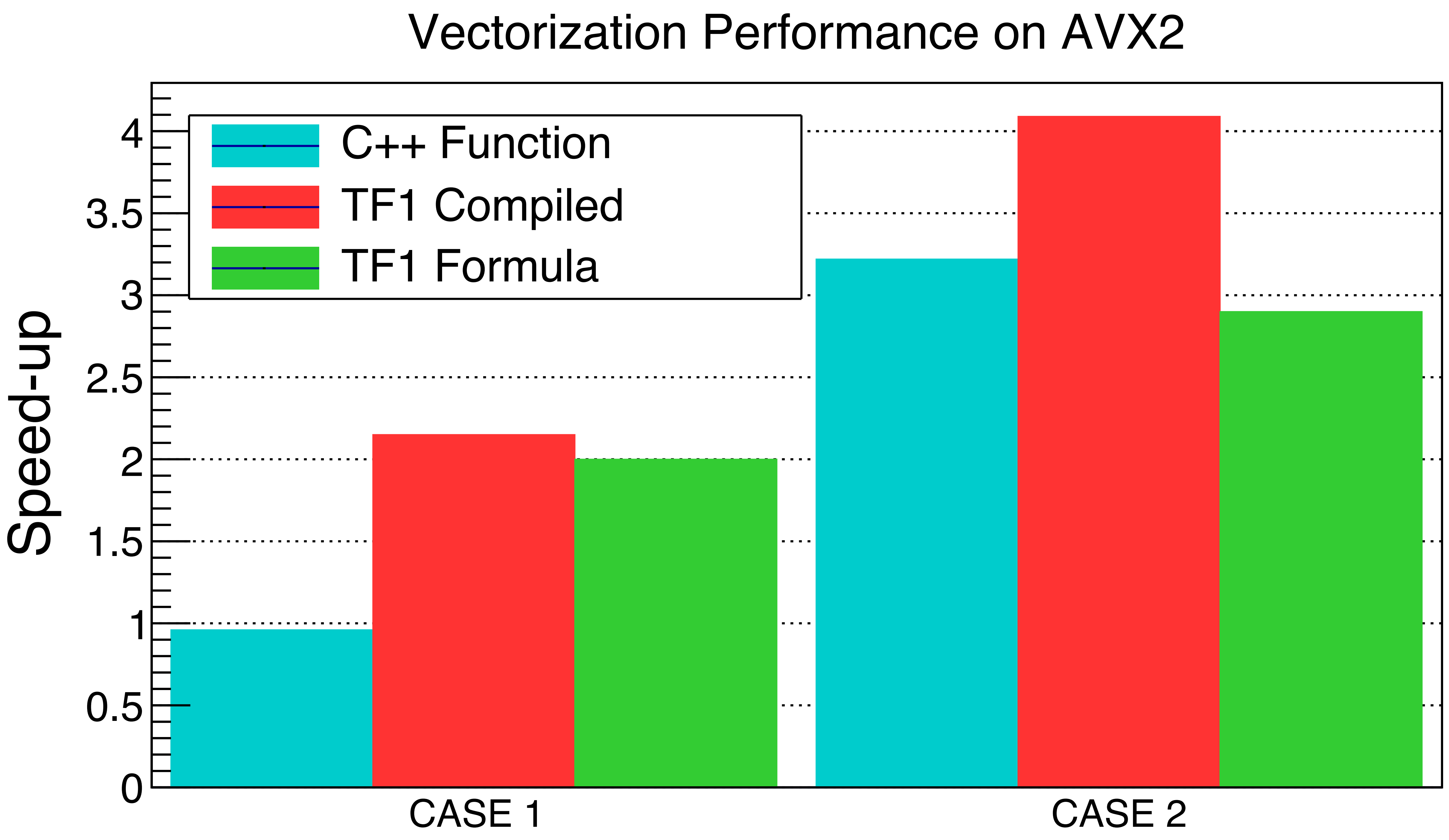
- Measure CPU performances in a typical HEP fitting
  - Speedups by combining vectorisation and parallelisation



Intel Xeon CPU E5-2683 with 28 physical cores

Vectorized TFormula Perfomances

- Performance results evaluating a math expression using a free C++ function with `TF1` and `TF1` based on `TFormula`
- Study the speed-up by using vectorisation on AVX



simple function   complex function

1. 2nd degree polynomial
2. exponential + gaussian