RooFit in 2022

Jonas Rembser (CERN, EP-SFT) for the ROOT team

11 May 2022, ROOT Users Workshop

Introduction

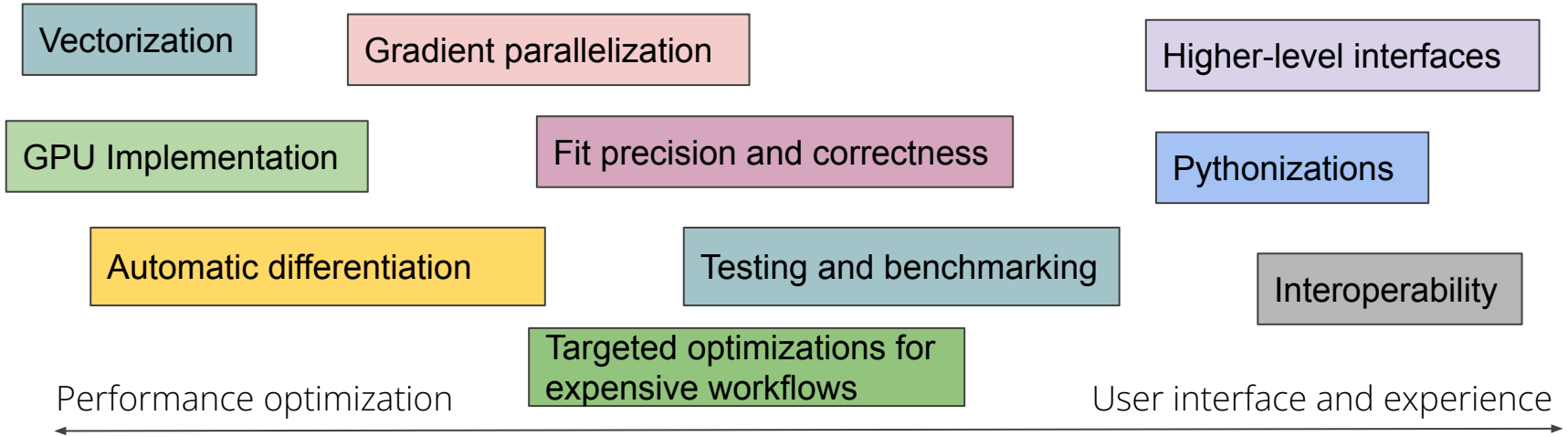
- **RooFit**: C++ library for statistical data analysis in ROOT
  - Model specification and fitting to data (baseline RooFit)
  - Implements common statistical tests (RooStats)
  - Includes tools to specify complex binned models (HistFactory)

- Recent development focused on:
  - **Performance** boost (preparing for larger datasets of **HL-LHC**)
  - More **user friendly** interfaces and high-level tools

- **Topics** of today:
  - **Overview** of development areas and recent highlights
  - **Outlook** on new developments

RooFit development areas

In which areas does RooFit evolve (besides bugfixes)?



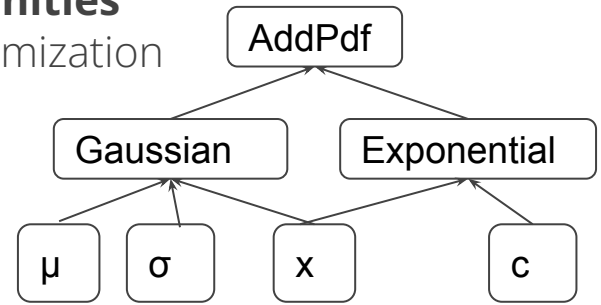
- Not all areas are covered with the same level of activity
- Some areas started to be covered only recently (*automatic differentiation, interoperability*)

New RooFit computation backend

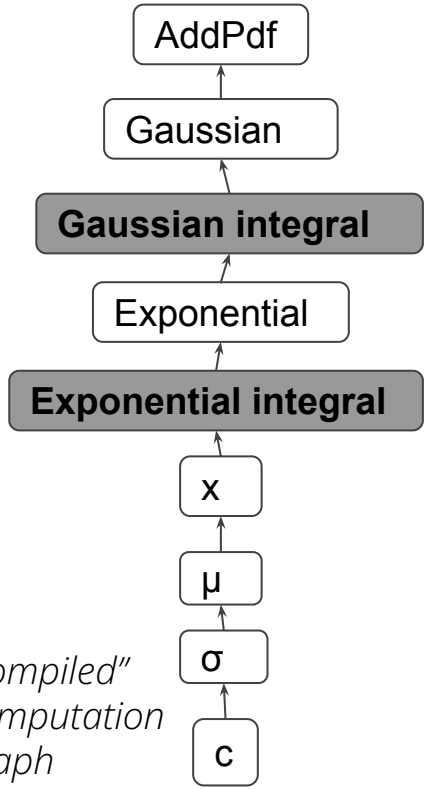
Vectorization

GPU Implementation

- Old way of evaluating RooFit models via recursion unsuitable for heterogeneous computing
- **New** `BatchMode("cpu")` and `"cuda"` computation backends for `pdf.fitTo()`
- RooFit **computation graph reorganized** as a sequence of functions with no side effects, evaluated by the `RooFitDriver`
  - Bypass internal caching in RooFit objects
  - Opened up **new opportunities** for parallelization and optimization
- **Broadcasting** of values to enable vectorization



original computation graph



"compiled" computation graph

Status of RooFit's BatchMode

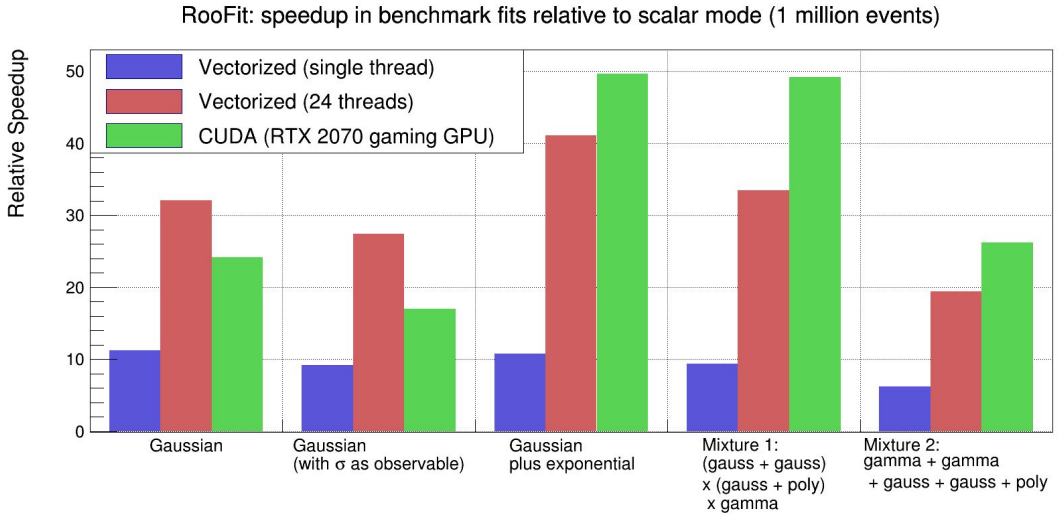
Vectorization

GPU Implementation

- Architecture-specific accelerator libraries for key functions
  - Optimal one loaded at runtime, given current architecture
  - Now also includes **GPU version**! Try it out with `pdf.fitTo(model, BatchMode("cuda"))`
- Multithreading via `ROOT::EnableImplicitMT()`

- **Huge speedup** for unbinned fits with many events

- For large computation graphs with few events, BatchMode still has larger overhead than recursive evaluation
- Goal for 6.28:
  Make BatchMode strictly Faster for any possible model



RooFit pythonizations

Pythonizations

- PyROOT bindings **more pythonic** in *6.26*
- Now you can for example:
  - use **Python keyword arguments** instead of RooFit command arguments
  - pass around **Python sets or lists** instead of `RooArgSet` or `RooArgList`
  - pass **Python dictionaries** to functions that take `std::map<>`
  - implicitly convert floats to `RooConstVar` in `RooArgList/Set` constructors
- All pythonizations are documented
- Some Pythonizations to help with C++/Python lifetime issue
  - Still there are memory leaks when returning owning pointers
- *See also this ROOT meeting presentation*

Example code from the rf316_llratioplot.py tutorial showcasing the pythonizations:

```
# Create background pdf poly(x)*poly(y)*poly(z)
px = ROOT.RooPolynomial("px", "px", x, [-0.1, 0.004])
py = ROOT.RooPolynomial("py", "py", y, [0.1, -0.004])
pz = ROOT.RooPolynomial("pz", "pz", z)
bkg = ROOT.RooProdPdf("bkg", "bkg", [px, py, pz])

# Create composite pdf sig+bkg
fsig = ROOT.RooRealVar("fsig", "signal fraction",
                       0.1, 0., 1.)
model = ROOT.RooAddPdf("model", "model",
                       [sig, bkg], [fsig])

data = model.generate((x, y, z), 20000)

# Make plain projection of data and pdf on x observable
frame = x.frame(Title="Projection on X", Bins=40)
data.plotOn(frame)
```

Implementing RooFit pythonizations

Pythonizations

- All the RooFit pythonization code is located in one directory in the PyROOT bindings
- For RooFit, we wrote an abstraction of the cppyy Pythonization engine using **Python mirror classes**
- All attributes of the mirror classes are **transferred** to the actual RooFit classes
  - The **original attributes** are available with an **underscore** prefix

It's important that power users are aware of how **easy** that is, to **contribute** pythonizations with high user impact!

Example code showing the pythonization of RooRealVar:

```
class RooRealVar(object):
    def bins(self, range_name=None):
        """Return the binning of this RooRealVar as a
        NumPy array."""

        # you can use the function name with an
        # underscore prefix to access the original C++
        # overload if it exists, e.g., by calling
        # self._bins

        # code skipped here
        …

        return bin_array
```

RooFit with NumPy, Pandas, and RDF

Pythonizations | Interoperability

- ROOT *v6.26* **new converters** between NumPy arrays/Pandas dataframes and **RooDataSet/RooDataHist**
  - No translation from RooDataHist to dataframe because histograms are in general multi-dimensional
  - Tutorial in Python
- New `RooRealVar.bins()` function to get RooFit **bin boundaries** as NumPy array
- Creating **RooFit datasets** from **RDataFrame**
  - Works for both `RooDataSet` and `RooDataHist`
  - Weighted filling still needs to be implemented
  - Tutorial in C++ and Python

Example of exporting RooDataSet to Pandas:

```
from ROOT import RooRealVar, RooCategory, RooGaussian

x = RooRealVar("x", "x", 0, 10)
cat = RooCategory("cat", "cat",
                  {"minus": -1, "plus": +1})

mean = RooRealVar("mean", "mean",
                  5, 0, 10)
sigma = RooRealVar("sigma", "sigma",
                   2, 0.1, 10)

gauss = RooGaussian("gauss", "gauss",
                    x, mean, sigma)


data = gauss.generate((x, cat), 100)

df = data.to_pandas()
```

	x	cat
0	6.997865	-1
1	7.211196	-1
2	3.198248	1
3	5.015824	1
4	7.782388	1
...	...	...
95	6.878027	-1
96	0.475900	1
97	4.451101	-1
98	3.481015	-1
99	4.010105	-1

100 rows × 2 columns

Many new fitting options

Fit precision and correctness

- `IntegrateBins(double precision)`: integrate the PDF over the bins instead of using the probability density at the bin center
- `RecoverFromUndefinedRegions(double strength)`: when PDF is invalid (e.g. negative), add penalty to likelihood to direct the minimizer away from undefined region
- `AsymptoticError()`: use the asymptotically correct approach to estimate errors in the presence of weights, slower but more accurate than `SumW2Error()` (https://arxiv.org/abs/1911.01303)

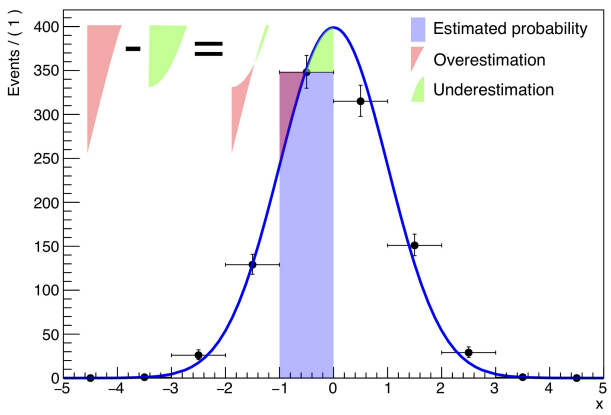


Illustration of bias in binned fits when not integrating PDF over bins

Higher-level interfaces

- `GlobalObservablesSource()`: which source to prioritize for global observable values, which can now be conveniently stored in `RooDataSet`/`RooDataHist`

Parallelized gradient calculation

Gradient parallelization

- For many parameters, most fitting time is spent for the **numeric gradient computation** (re-evaluation after varying each parameter one at a time)
- Distributing the **gradient calculation over multiple processes** is a very general way to speed up fitting (see ACAT 2019 presentation)
- Gradient parallelization is part of ROOT *6.26*
- It comes together with **new likelihood classes** with improved performance for parallelization over entries

More info in this talk on the same workshop

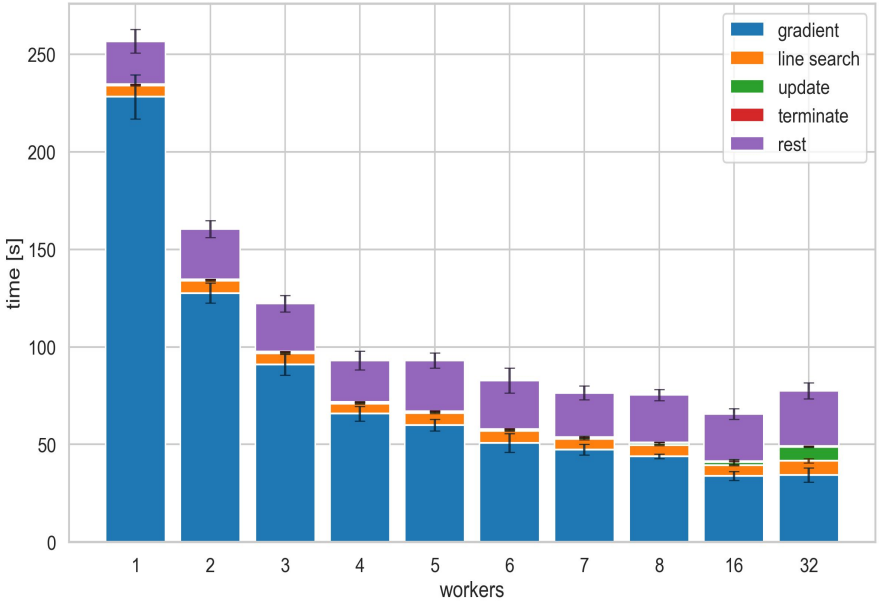


Figure from the ACAT 2019 presentation showcasing the scaling of the gradient parallelization for an ATLAS Higgs combination fit

RooWorkspace ⇄ JSON/YAML

Interoperability

- Tools to build RooWorkspaces (e.g. **HistFactory** or CMS **Higgs combination** tool) require descriptive languages to define the model (like **XML** for HistFactory)
- **JSON** or **YAML** is more readable and more standard nowadays
- The new RooFit *(6.26)* includes a new `RooJSONFactoryWSTool` to **import/export** `RooWorkspaces` to JSON or YAML
- This can ease interoperability also with other statistics frameworks such as **pyhf** an **zfit**

Example on the right: *JSON for Gaussian signal with* ***`RooArgusBG`*** *background*

```
"pdfs": {
        "signal": {
            "type": "Gaussian",
            "x": "mes", "mean": "sigmean", "sigma": "sigwidth"
        },
        "background": {
            "type": "ARGUS",
            "mass": "mes", "resonance": 5.291,
            "slope": "argpar", "power": 0.5
        },
        "model": {
            "type": "pdfsum",
            "summands": [
                "signal",
                "background"
            ],
            "coefficients": [
                "nsig",
                "nbkg"
            ],
            "tags": [
                "toplevel"
            ]
        }
},
"variables": {
        "mes": { "value": 5.25, "min": 5.2, "max": 5.3 },
        "sigmean": { "value": -5.28, "min": 5.2, "max": 5.3 },
        "nsig": { "value": 200, "min": 0, "max": 10000 },
        "argpar": { "value": -20, "min": -100, "max": -1 },
        "nbkg": { "value": 800, "min": 0, "max": 10000 }
}
```

More info in this talk on the same workshop

RooFit in the ROOT Plan of Work 2022

RooFit plans from ROOT **plan of work** 2022 slides (public),

priorities **super high**, medium high, fairly high:

- **Prototype usage of automatic differentiation**
- **Consolidate work on batch mode and GPU support**
- **Roll out parallel gradient likelihood and parallel Hessian computation**
- Further optimize HistFactory implementation for speed
- Stabilize RooWorkspace to JSON conversion tools
- **More benchmarks with recent experiment workflows**
- Further pythonizations

..plus addressing the **requests from experiments**!

Automatic differentiation (AD) in RooFit

Automatic differentiation

- **Gradient** of RooFit model essential for minimization
  - RooFit uses numeric derivatives, varying one parameter at the time
  - Using analytic gradients is much more efficient for many parameters
  - We can use **automatic differentiation** techniques to get these gradients

- No code merged yet, but we investigate different implementation paths:
  - Extend `RooAbsReal` with gradient interface and evaluate the gradient with the **chain rule**
  - **Squashing** RooFit model to one function **and automatically differentiate** with **clad**

- For both approaches, we can build on top of the new `BatchMode()` evaluation backend
- We are focusing on *HistFactory* models in our prototype work:
  - Limited set of RooFit objects and many parameters

Benchmarks with experiment workflows

Testing and benchmarking

Targeted optimizations for expensive workflows

- The focus of this year so far was consolidating the BatchMode and prototyping for AD

- We also want to improve **monitoring** of major user code and **cutting-edge workflows**
- We have access to recent **CMS** and **ATLAS Higgs combination** workspaces
  - Already use them for guiding improvements:
    - performance optimizations
    - Interface extensions
  - Structured benchmarks in rootbench will follow later

- It is **important** that experiment code is compatible with newest ROOT version for benchmarking new RooFit developments!
  - We are also happy to help with this, as we do for example for Higgs combing (CMS)

Summary

- RooFit is **evolving** steadily
  - Support and development from **ROOT team** at CERN
  - Many new features developed by **external contributors**
- Highlights of the recent version *6.26* are the **GPU BatchMode** and the **Pythonizations**
- Future developments will focus on **automatic differentiation** and **general speedups**
  - In particular for *HistFactory-style* binned fits with many parameters
- It is important to know about experiment workflows for targeted **optimizations**
- Your input is always welcome!