Accelerating RooFit with GPUs …and other RooFit news

Jonas Rembser for the RooFit team, presenting developments by Emmanouil Michalainas, Stephan Hageböck, Lorenzo Moneta, Jonas Rembser, Patrick Bos, Wouter Verkerke, Carsten Burgard, Harshal Shende, Rahul Balasubramanian

1 Dec 2021, ACAT 2021

Introduction

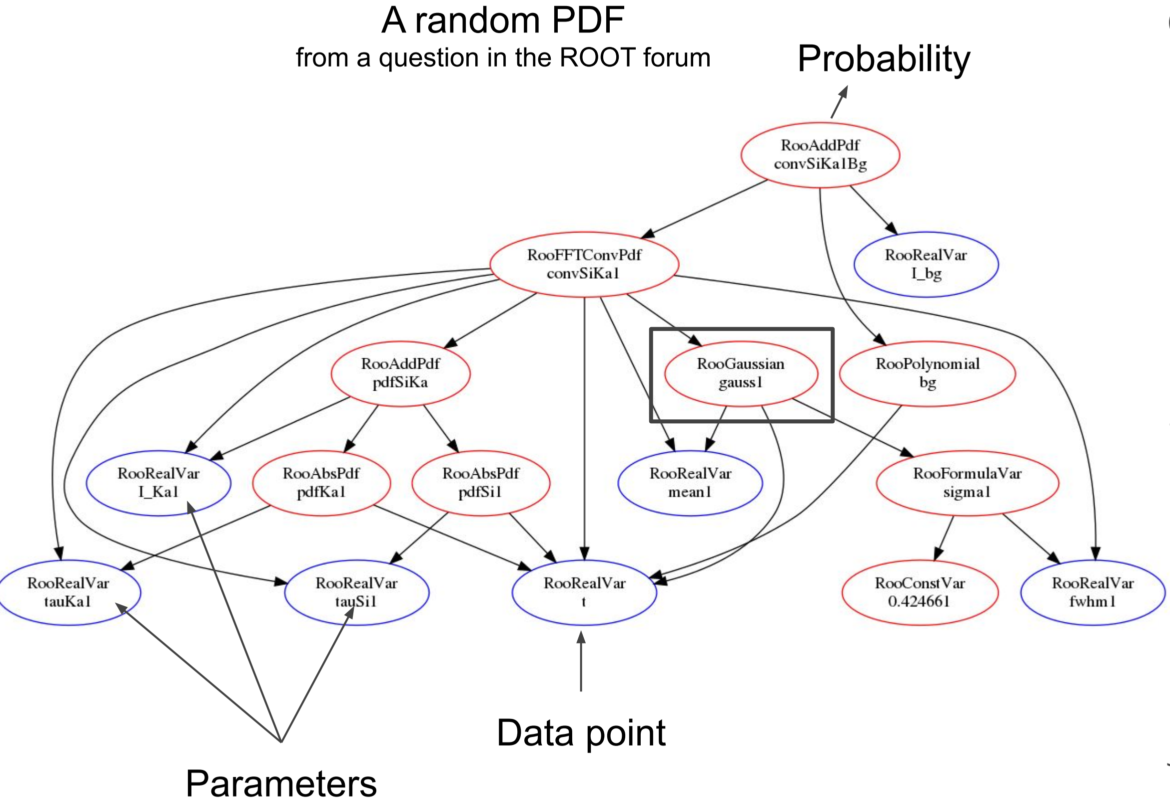
- **RooFit**: C++ library for statistical data analysis in ROOT
  - provides tools for model building, fitting and statistical tests
- Recent development focused on:
  - **Performance** boost (preparing for larger datasets of **HL-LHC**)
  - More **user friendly** interfaces and high-level tools

In **this presentation**:

- Report on new **vectorized RooFit** interface with **GPU support** (aka *BatchMode*)
  - **CPU**: up to 10x speed up, **GPU**: up to 50x!
- Overview on other new RooFit features in the upcoming ROOT release *v6.26*
  - Highlight: **pythonizations**!
- Outlook on planned developments

1. Faster vectorized RooFit: Now with GPU support

Why vectorizing RooFit



Current (**scalar**) RooFit computation:

1. Load a single data point into variables
2. Walk whole expression tree (minus cached branches)
3. Obtain one probability. Repeat at 1. with next data point.

This is problematic:

- Simple profiling:
  50% L1/L2 cache misses
- No chance to vectorize computations

See RooFit presentation at ICHEP 2020

From scalar to vector computations

- **Evaluate** every RooFit object **once per** fitting **iteration**
  - Iterate over dataset entries in inner loop
- Better compiler optimizations, cache efficiency, **vectorization**, less virtual function calls
- For maximum efficiency:
  - Run multiple CUDA kernels, CPU and GPU computations **concurrently**
- To do this: **RooBatchCompute** library and **`RooFitDriver`** class explained next

#9004

Implementation of Gaussian in old RooFit code and new RooBatchCompute library compared:

```
double RooGaussian::evaluate() const {
  double arg = x - mean;
  return std::exp(-0.5*arg*arg/(sigma*sigma));
}
```

```
__global__ void computeGaussian(Batches batches)
{
  auto x=batches[0], mean=batches[1], sigma=batches[2],
normVal=batches[3];
  for (size_t i=BEGIN; i<batches.getNEvents(); i+=STEP) {
    double arg = x[i]-mean[i];
    double halfBySigSq = -0.5 / (sigma[i]*sigma[i]);
    batches._output[i] = fast_exp(arg*arg*halfBySigSq)/normVal[i];
  }
}
```

The RooBatchCompute library

- A new ROOT **library** containing the code for the vector computations + some few extras
- Compiled multiple times for different target **architectures** (generic, SSE4.1, AVX, AVX2, AVX512 and **CUDA**)
  - We **reuse code** for CPU and CUDA implementations (modulo some `#define`)
  - Uses thrust, the CUDA C++ template library

- **Automatic** hardware inspection and **loading** of the right library **at runtime**
  - No need to recompile ROOT for your system!
- Handles **broadcasting** of scalar values to arrays with minimal overhead
- Also supports **multithreading** via `ROOT::EnableImplicitMT()`

- Try it out by passing `"CPU"` or `"GPU"` to the `BatchMode()` argument of `fitTo()`:
  - `pdf.fitTo(*data, RooFit::BatchMode("GPU"));`

The RooFitDriver approach

- Heterogeneous computing hard to implement with **recursive** interface (e.g. virtual calls to `RooAbsReal::getValV(const RooArgSet* normSet)`)
- We need logic to:
  - **analyze** the computation graph, figure out sizes of result arrays
  - handle **memory** both on the host and the CUDA device
  - choose correct instance of **RooBatchCompute** per call
  - **evaluate** RooFit objects in the correct order
  - manage CUDA streams, **synchronize** results, …
- The new **RooFitDriver** class is responsible for all of that
- Improved **thread-safety** by bypassing result caching in RooFit objects

Game changer in RooFit implementation!

Computation graph analyzed

- Not all RooFit classes support GPU evaluation
  - RooFitDriver manages **concurrent evaluation** on CPU and GPU
- Host ⇄ device copying times and CPU/GPU evaluation times are measured
  - This can be done in the first two minimization iterations
  - Decide if an object should be evaluated on the GPU given the **copying overhead**
- Together with multithreading capabilities of RooBatchCompute:
  => we can can **use all CPU cores and GPU** device for RooFit evaluations!

Measured run times using `RooFitDriver`*:*

```
------Copying times------
h2dTime=618us   d2hTime=597us
------Nodes------                        CPU time:          CUDA time:
nll        CUDA      0x5575eabb0600      3123 us            355 us
mean       CPU       0x5575eacc4090      0 us (param.)      nan
gauss      CUDA      0x5575eacef950      5560 us            376 us
sigma      CPU       0x5575e8f94e90      0 us (param.)      nan
```

Benchmark setup

- **CPU**: AMD Ryzen 9 3900 12-Core Processor (**24 Threads**)
- **GPU**: NVIDIA GeForce **RTX 2070** SUPER
  - *Note*: **gaming GPU** not optimized for double precision (single precision to double precision register ratio 32:1)
  - Much better results expected in a data-center/scientific GPU
- Set up: Perform a full fit
  - Not easy to get benchmarks that represent vast variety of models in the wild
- Multithread results generated using all 24 threads

Some **caveats**:

- **Kahan summation** of log-likelihoods switched **off** (not implemented on GPU yet)
- Recovery from invalid parameters switched off (`RecoverFromUndefinedRegions(0.0)`)

Benchmark results

Benchmark 1 million events	**Scalar time**	**Vector-ST** (Speedup vs scalar)	**Vector-MT** (Speedup vs scalar)	**CUDA** (Speedup vs scalar)
Gaussian with one observable `gauss(x)`	2632 ms	234 ms (11x)	82 ms (32x)	109 ms (24x)
Gaussian with two observables `gauss(x,s)`	1069 ms	116 ms (9x)	39 ms (27x)	63 ms (17x)
Gaussian plus exponential `f × gauss(x) + (1-f) × exp(x, c1)`	9784 ms	908 ms (11x)	238 ms (41x)	197 ms (50x)
Addition benchmark 1 `(fx × gauss(x) + (1-fx) × gauss(x)) × (fy × gauss(y) + (1-fy) × poly(y)) × gamma(z)`	112 s	12 s (9x)	3.35 s (33x)	2.28 s (49x)
Addition benchmark 2 `ns1 × gamma(x) + ns2 × gamma(x) + ng2 × gauss(x) + ng3 × gauss(x) + npol × poly(x)`	93 s	15 s (6x)	4.80 s (19x)	3.55 s (26x)

ST = single thread, MT = multithreading

2. Other new RooFit features

RooFit pythonizations

- PyROOT bindings **more pythonic** in *v6.26*
- 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 in the [reference guide]
- *See also this [ROOT meeting presentation]*

PR list

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)
```

Interoperability with NumPy/Pandas

- ROOT *v6.26* **new converters** between NumPy arrays/Pandas dataframes and **RooDataSet/RooDataHist**:
  - `RooDataSet.from_numpy()`
  - `RooDataSet.to_numpy()`
  - `RooDataSet.from_pandas()`
  - `RooDataSet.to_pandas()`
  - `RooDataHist.from_numpy()`
  - `RooDataHist.to_numpy()`

#9346

#8784

- No translation from RooDataHist to dataframe because histograms are in general multi-dimensional
- New `RooRealVar.bins()` function to get RooFit **bin boundaries** as NumPy array

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

Parallelized gradient calculation

- For many parameters, most fitting time is spent for the **numeric gradient computation** (re-evaluatoin 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)
- The gradient parallelization will be part of ROOT *v6.26* (PR list , last PR #9349)
- It comes together with **new likelihood classes** with improved performance for parallelization over entries

- Next year: work on faster gradient calculation with **automatic differentiation (AD)**

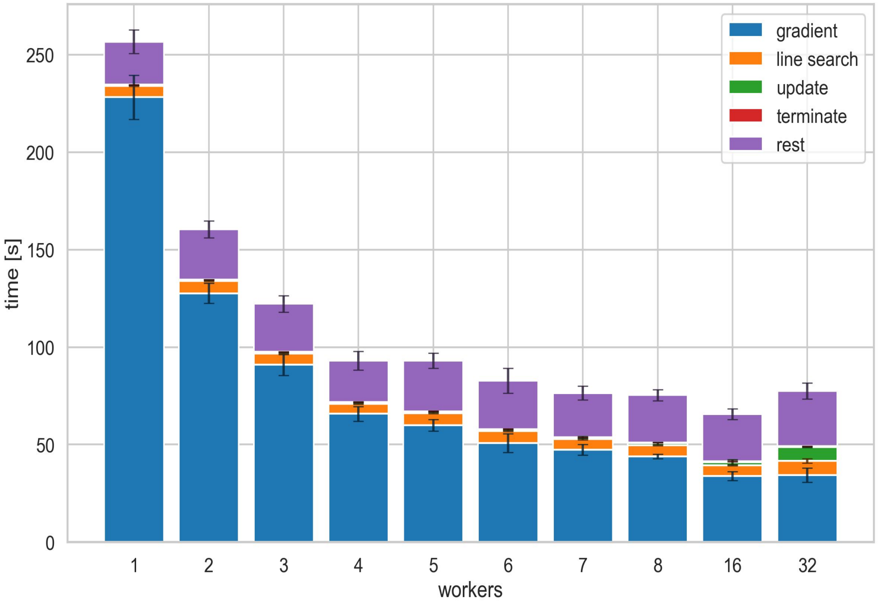


Figure from the ACAT 2019 presentation showcasing the scaling of the gradient parallelization for an ATLAS Higgs combination fit

RooWorkspace ⇄ JSON/YAML

- There are higher-level tools to build RooFit models in 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 *(v6.26)* 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 }
}
```

#8944

Other new features for *v6.26*

- **Creating RooFit datasets from RDataFrame** #7317
  - Works for both `RooDataSet` and `RooDataHist`
  - Weighted filling still needs to be implemented
  - Tutorial in C++ and Python

Last two features requested by **ATLAS!**

- **Global observables in RooFit datasets** #8788
  - Convenient way to store global observable values in `RooDataSet`/`RooDataHist`
- **Bin integration for simultaneous fits** #9358
  - *v6.24* introduced bin integration option to avoid biases in binned fits
  - ATLAS Higgs combination effort wanted to set the bin integration parameter
  - *v6.26*: set bin integration parameter **separately** for **each pdf** in **`RooSimultaneous`**

3. Outlook and summary

Outlook on planned developments

- Continue development on **faster** vectorized **RooFit**
  - Implement more RooFit operations on the **GPU** (e.g. analytic integrals, convolutions)
  - **Concurrent evaluation** of RooFit objects as alternative to parallelizing over entries

- Improve higher-level RooFit tools like **HistFactory**
  - Targeted **performance improvement** for binned fits
  - Make it easier to use and more stable

- Speed up gradient computation with **automatic differentiation**

- Continue to improve **Python** bindings
  - Solve object ownership issues

- **Your idea** here!

Talk summary

- RooFit is **evolving** steadily
  - Support and development from **ROOT team** at CERN
  - Many new features developed by **external contributors**

- Updated **vectorized** RooFit interface (`BatchMode()`) in *v6.26*
  - New computation library for multiple architectures
  - Significant refactor of RooFit computation graph
  - Up to 10x faster on a single **CPU** thread
  - Up to 50x faster on a gaming **GPU**

- New **pythonizations** and exclusive functions for Python interface, other new features

Please try it out and get in touch with us!

Useful Links

Useful Links

- RooFit tutorials (Recommended! There are notebooks!)
- RooFit documentation
- Release notes (yet to come for ROOT *v6.26*)
- Test the faster batch mode:
    - `auto result = pdf.fitTo(*data, RooFit::BatchMode("GPU"), RooFit::Save());`
    - Note: old scalar computation is the default
- Feature request or bug report? https://github.com/root-project/root/issues
    - A new PDF that should go into RooFit?
    - Ideas for Python interfaces?
- Tricky problems? https://root-forum.cern.ch/c/roofit-and-roostats
- Think that a certain workflow should be part of ROOT's tests? jonas.rembser@cern.ch