RooFit's new heterogeneous computing backend

Jonas Rembser

8 May, CHEP 2023

ROOT
Data Analysis Framework
https://root.cern

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
- Today: **20th anniversary** of RooFit first presented at a conference (CHEP 2003)

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
- Today: **20th anniversary** of RooFit first presented at a conference (CHEP 2003)

In **this presentation**:

- Report on new **vectorized RooFit** interface with **GPU support** (aka *BatchMode*)
- Follow-up on ACAT 2021 talk with preliminary prototype results
- Today's benchmark results obtained with **ROOT *6.28.04*!**

Other RooFit presentations to follow:

- **Garimas Singhs** presentation on applying **automatic differentiation** to RooFit
- **Zef Wolffs** presentation on **configurable parallelization** in RooFit

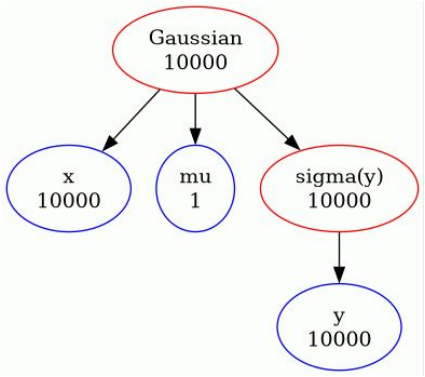
Computation graphs in RooFit

RooFit evaluates expression trees many times for different parameter values to find NLL minima.

Computation graphs in RooFit

RooFit evaluates expression trees many times for different parameter values to find NLL minima.

Expression tree with observables x and y for 10000 data points:
Gaussian(x|mu,sigma(y))




```
RooRealVar x{"x", "x", 0.0, -20.0, 20.0};
RooRealVar y{"y", "y", 0.0, 0.0, 1.0};

RooRealVar mu{"mu", "mu", 0.0, -20.0, 20.0};
RooFormulaVar sigma{"sigma", "1.0 + 2.0 * y", {y}};

RooGaussian gauss{"gauss", "gauss", x, mu, sigma};
```

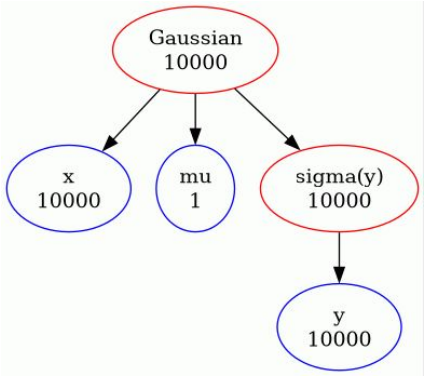
Computation graphs in RooFit

RooFit evaluates expression trees many times for different parameter values to find NLL minima.

Why rewriting RooFit NLL evaluation backend:

- Old RooFit computation: re-evaluate expression tree of *for each event*
- Lots of function calls, **no vectorization possible**

Expression tree with observables x and y for 10000 data points:
Gaussian(x|mu,sigma(y))




```
RooRealVar x{"x", "x", 0.0, -20.0, 20.0};
RooRealVar y{"y", "y", 0.0, 0.0, 1.0};

RooRealVar mu{"mu", "mu", 0.0, -20.0, 20.0};
RooFormulaVar sigma{"sigma", "1.0 + 2.0 * y", {y}};

RooGaussian gauss{"gauss", "gauss", x, mu, sigma};
```

Computation graphs in RooFit

RooFit evaluates expression trees many times for different parameter values to find NLL minima.

Why rewriting RooFit NLL evaluation backend:

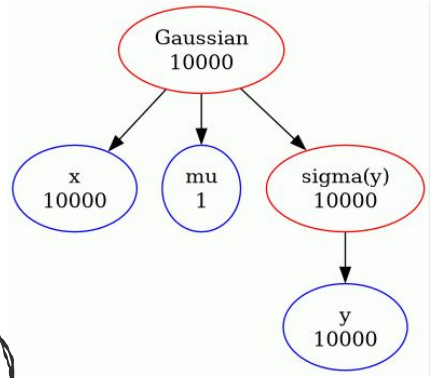
- Old RooFit computation: re-evaluate expression tree of *for each event*
- Lots of function calls, **no vectorization possible**

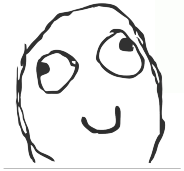
Should have been easy to improve and do on GPU?!

1. Allocate memory for results
2. Call vectorized function/CUDA kernel for each node[1] in topological order if values of children have changed

[1] `RooAbsArg` in RooFit

Expression tree with observables x and y for 10000 data points:
Gaussian(x|mu,sigma(y))





```
RooRealVar x{"x", "x", 0.0, -20.0, 20.0};
RooRealVar y{"y", "y", 0.0, 0.0, 1.0};

RooRealVar mu{"mu", "mu", 0.0, -20.0, 20.0};
RooFormulaVar sigma{"sigma", "1.0 + 2.0 * y", {y}};

RooGaussian gauss{"gauss", "gauss", x, mu, sigma};
```

Computation graphs in RooFit

RooFit model evaluation is not straight forward:

- Nodes often own other nodes that they evaluate
- These *internal nodes* are not registered in the graph
- Sometimes these nodes are even clones of entire subgraphs

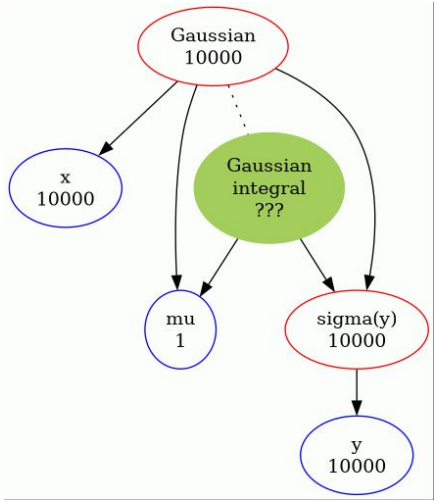
Computation graphs in RooFit

RooFit model evaluation is not straight forward:

- Nodes often own other nodes that they evaluate
- These *internal nodes* are not registered in the graph
- Sometimes these nodes are even clones of entire subgraphs

Typical example: **normalization integrals**
(still harmless compared to other cases, but good for illustration)

Evaluating model for given normalization observables dynamically extends computation graph, adding new disconnected nodes




```
gauss.getVal(/*normSet=*/x);
```

Computation graphs in RooFit

RooFit model evaluation is not straight forward:

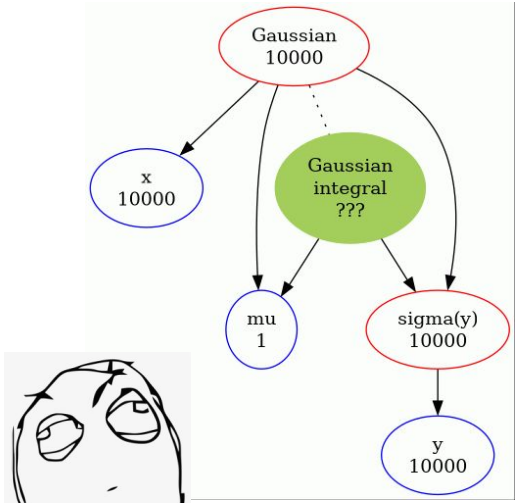
- Nodes often own other nodes that they evaluate
- These *internal nodes* are not registered in the graph
- Sometimes these nodes are even clones of entire subgraphs

Typical example: **normalization integrals**
(still harmless compared to other cases, but good for illustration)

Dynamic nature of computation graphs in RooFit makes organizing data flow and computations in a heterogeneous computing environment a challenge.

In other words: data structure for model *building* not completely suitable for *evaluation*.

Evaluating model for given normalization observables dynamically extends computation graph, adding new disconnected nodes




```
gauss.getVal(/*normSet=*/x);
```

Computation graphs with fixed normalization

New mechanism to “compile” the graph for a given normalization set to fulfill condition ⟶

Each RooAbsArg involved in the evaluation must be connected to the top node via RooFits client-server relations.

Computation graphs with fixed normalization

New mechanism to "compile" the graph for a given normalization set to fulfill condition ⟶

Each RooAbsArg involved in the evaluation must be connected to the top node via RooFits client-server relations.

If your RooFit classes don't fulfill this yet, you should consider overriding:
`RooAbsArg::compileForNormSet()`

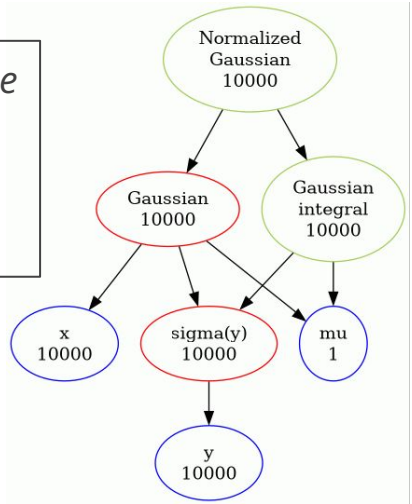
- Function called recursively in NLL creation when using `BatchMode()`
- Result is ready for heterogeneous eval.
- Mechanism also used for the C++ code generation from RooFit models that enables automatic differentiation *(see next talk by Garima Singh)*

This function can also be used to hook in graph **optimizations**.

Computation graphs with fixed normalization

New mechanism to "compile" the graph for a given normalization set to fulfill condition ⟶

Each RooAbsArg involved in the evaluation must be connected to the top node via RooFits client-server relations.



If your RooFit classes don't fulfill this yet, you should consider overriding:
`RooAbsArg::compileForNormSet()`

- Function called recursively in NLL creation when using `BatchMode()`
- Result is ready for heterogeneous eval.
- Mechanism also used for the C++ code generation from RooFit models that enables automatic differentiation *(see next talk by Garima Singh)*

This function can also be used to hook in graph **optimizations**.

```
auto nll = gauss.createNLL(
    *data,
    ConditionalObservables(y),
    BatchMode("cpu")
); // create NLL object

nll->Print("v"); // get some info on the graph evaluation order
```

```
--------------------------------------------------------------------------------
| Idx | Name                    | Class            | Size    | From Data |
--------------------------------------------------------------------------------
| 1   | y                       | RooRealVar       | 10000   | 1         |
| 2   | sigma                   | RooFormulaVar    | 10000   | 0         |
| 3   | mu                      | RooRealVar       | 1       | 0         |
| 4   | x                       | RooRealVar       | 10000   | 1         |
| 5   | gauss                   | RooGaussian      | 10000   | 0         |
| 6   | gauss_Int[x]            | RooRealIntegral  | 10000   | 0         |
| 7   | gauss_over_gauss_Int[x] | RooNormalizedPdf | 10000   | 0         |
| 8   | nll                     | RooNLLVar        | 1       | 0         |
--------------------------------------------------------------------------------
```

The vectorized evaluation functions

- The BatchMode backend uses **new functions** in RooAbsReal that you can **override** to add support for CPU and GPU of your class:
  - `RooAbsReal::canComputeWithCuda()`
  - `RooAbsReal::computeBatch()`

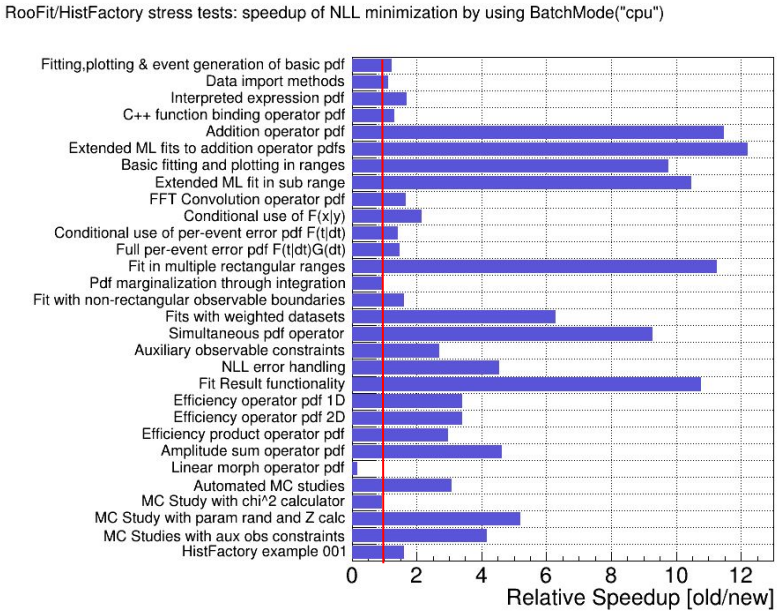
- Implementation of RooFit classes in ROOT uses `RooBatchCompute` library to implement `computeBatch()`:
  - **Architecture-specific accelerator libraries** for key functions
  - **Optimal one loaded at runtime**, given current architecture
  - More details in the ACAT 2021 talk

- Add the FastEvaluations stream to the RooMsgService the get **info printouts when** your RooAbsArgs **don't support** the **new backend**:

```
RooMsgService::instance().addStream(
        RooFit::Info, Topic(RooFit::FastEvaluations)
);
```

Benchmarking the RooFit test suite

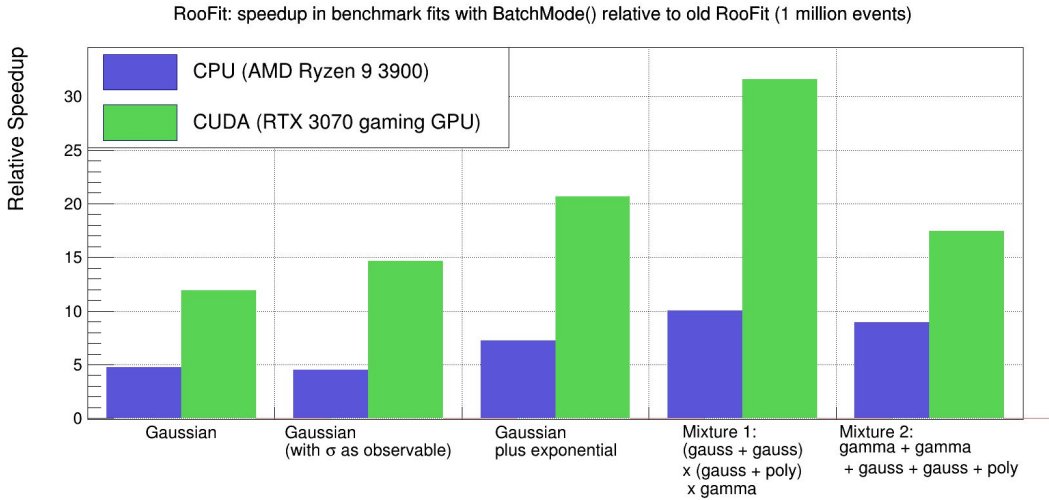
- Plot shows relative time spent for minimizations in [stressRooFit tests] for `BatchMode("cpu")` and `"off"`
- Significant speedup for almost all tests from a combination of:
  a. **Vectorized** evaluation
  b. Optimized computation graphs
  c. Less function calls
- Average speedup of **4.4x**



Results obtained with ROOT 6.28.04
Compare also to [ICHEP 2022] results, showing less drastic speedups in the middle of ROOT 6.28 development

Benchmarking basic unbinned fits

- Benchmarking unbinned bit with 1 million events
- The CPU BatchMode runs on a single thread
- The CUDA kernels are launched with **128 thread blocks** with 1024 threads each
- Plot shows speedup relative to the old scalar evaluation interface

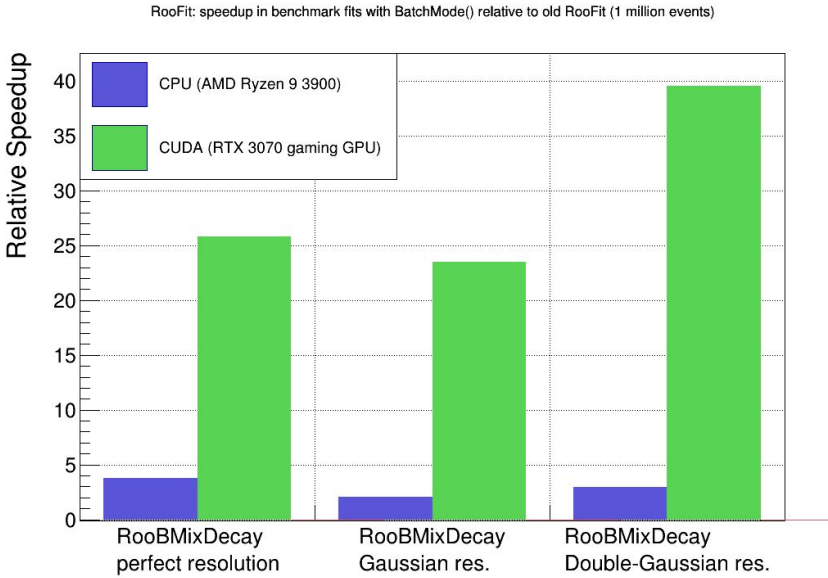


benchRooFitBackends in rootbench repo, plotting script is in same directory. Try it yourself with ROOT 6.28.04! Remember to use a ROOT build with -Dcuda=ON

New benchmarks for analytical convolution fits

- New benchmarks based on analytical convolutions of [RooBMixDecay] with resolution functions
  - With **perfect resolution** (`RooTruthModel`)
  - **Gaussian resolution** (`RooGaussModel`)
  - **Double-Gaussian** resolution (`RooAddModel` of `RooGaussModel`s)
- Describes the decay of B mesons with the effects of B0/B0bar mixing
- Quite an involved fit:
  double-Gauss fit takes 1 min with old backend
- GPU speedup **up to 40x**!
  - Larger speedups than for previously benchmarked simple models

Plan to also **do numeric integrals on GPU in the future** to support more B-physics usecases, i.e. amplitude fits.



RooFitUnBinnedBenchmarks in rootbench repo, plotting script is in same directory. Try it yourself with ROOT 6.28.04! Remember to use a ROOT build with -Dcuda=ON

How to use the new NLL evaluation backend

- Try it out by passing `"cpu"` or `"cuda"` to the `BatchMode()` argument of `RooAbsPdf::fitTo()` / `RooAbsPdf::createNLL()`:
  - `pdf.fitTo(data, RooFit::BatchMode("cuda"))`

It's a **one-line change**!

See also the RooAbsPdf documentation.

Conclusions and next steps

- RooFits new vectorized NLL evaluation backend (aka. BatchMode) is now production ready
  - **All RooFit tests pass** if enabled by default, which might happen in next ROOT release
  - If your model doesn't benefit from speedup yet, please open a bug report
  - Average speedup of about **4x** compared the old RooFit evaluation backed
- Revised CUDA backend in **ROOT 6.28.04**!
  - Gives you great speedup for wide range of unbinned fits with many events
  - Average speedup of **25x** (up to **40x!**) in fits with 1M events on **GeForce RTX 3070**
- The new backend relies on mechanism to fix computation graph that you might need to implement in custom RooFit classes
- **Next steps** *(CERN openlab summer student project)*:
  - Support even more PDFs with CUDA backend
  - **Numeric integration** also on the GPU