Novel functional and distributed approaches to data analysis in ROOT

Guilherme Amadio (CERN) for the ROOT Team

ROOT
Data Analysis Framework

https://root.cern

Outline

- The problem we are trying to solve
- TDataFrame: a declarative approach to data analysis
- Performance figures
- Ongoing R&D: distributing ROOT based data analysis on Spark clusters
- Foreseen evolution

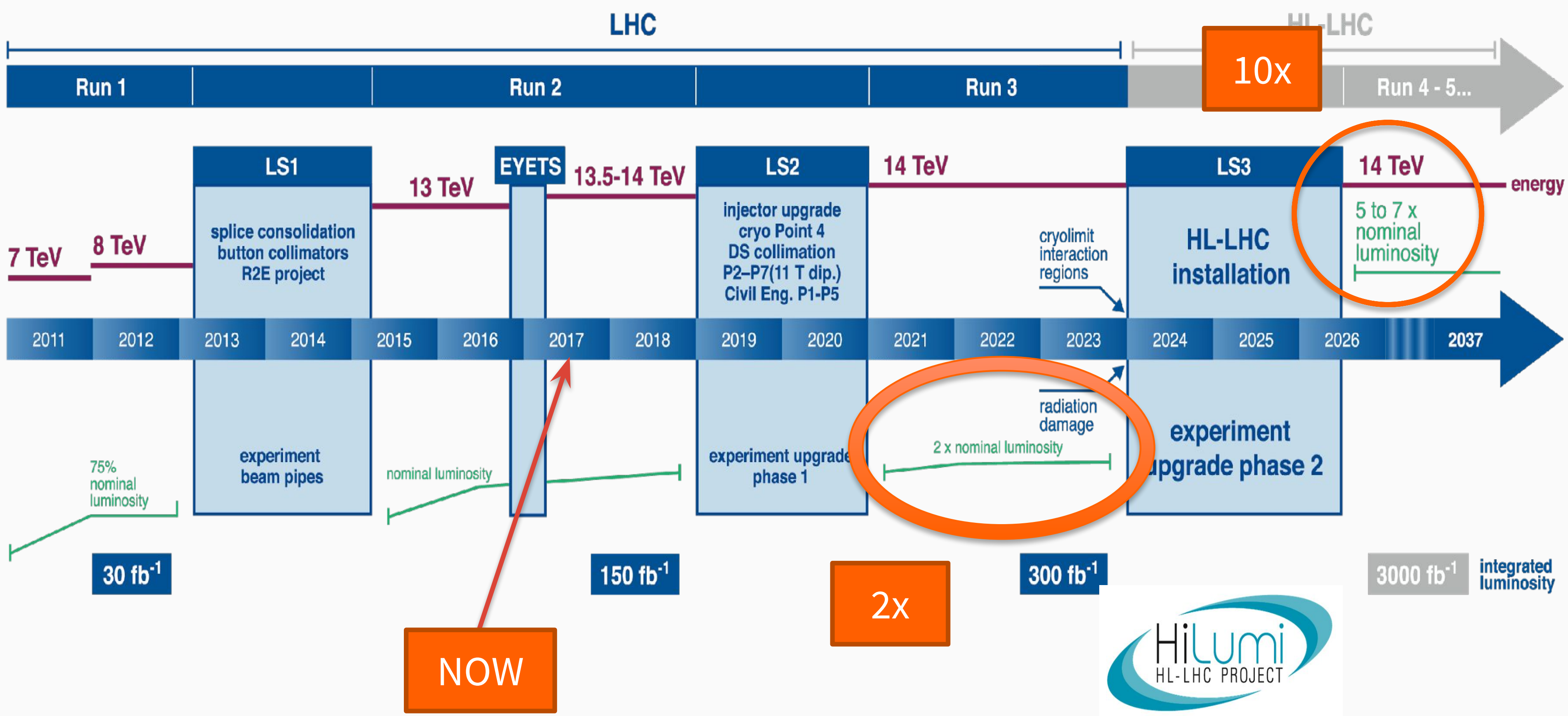
All available in ROOT 6.10!

See talk A quantitative review of data formats for HEP analyses by J. Blomer!

The challenge posed by the increase in luminosity

Even more events to analyse!

- Full exploitation of the LHC: highest priority in the European Strategy for Particle Physics, adopted by the CERN Council and integrated into the ESFRI Roadmap.
- Major LHC upgrade in ~2020: increase luminosity by 10x beyond the original design.



http://hilumilhc.web.cern.ch

How to cope with this?

An opportunity to improve our analysis toolset

Requirements:

1. Exploit modern, parallel architectures, including accelerators, *for data analysis*
   - Leverage the experience accumulated parallelising centralised data processing
2. Offer an **easy programming model** to scientists
   - Obtain more results with less effort

Functional Chains R&D

- We are constantly looking for opportunities to apply implicit parallelism in ROOT
- "Functional Chains" R&D being carried out
  - Functional programming principles: no global states, no for/if/else/break
  - Analogy with tools like ReactiveX*, R dataframe, Spark
  - Gives room for optimising operations internally

Can this be a successful model for our physicists?

```
import ROOT
f = ROOT.TFile("aliDataset.root")
aliTree = f.Events
dataFrame = TDataFrame(aliTree)

dataFrame.filter(sel1).map(func2).cache().filter(sel3).histo('var1:var2').Draw('LEGO')
```

Express analysis as a chain of functional primitives.

TDataFrame: A Declarative Approach to Data Analysis in ROOT

"The comfort of the big data tools, with the speed of ROOT."

TDataFrame: A Declarative Approach to Data Analysis in ROOT

“The comfort of the big data tools, with the speed of ROOT.”

Available since ROOT 6.10 (June ‘17)

TDataFrame: Declarative Analysis

- New way to interact with ROOT columnar data format
  - Inspiration from Pandas, Spark, and others
  - Similar ideas proposed in the past (e.g. LINQToROOT by G. Watts)
- Analysis is a graph of:
  - **Transformations**: filter, add a column, …
  - **Actions**: Fill an histogram, a profile, count events, …
- Specify **what** you want and let ROOT choose **how** to do it
  - Computation triggered lazily
  - Several optimisations (e.g. partitioning, caching, reordering, parallelisation)

DOI 10.5281/zenodo.260230

TDataFrame: Less Boilerplate Code

Full control of the loop with TTreeReader, but

- Needs boilerplate code
- Not easily parallelisable
- Simple operations implemented over and over again

TTreeReader

```
TTreeReader data(tree);
TTreeReaderValue<A> x(data, "x");
TTreeReaderValue<A> y(data, "y");
TTreeReaderValue<A> z(data, "z");
while(data.Next())
 if(IsGoodEvent(*x, *y, *z))
   h.Fill(*x)
```

TDataFrame

```
TDataFrame tdf(tree);
auto h =
 tdf.Filter(IsGoodEvent, {"x","y","z"})
    .Histo1D("x");
```

TDataFrame: Trivial Parallelisation

- A single line change to enable implicit parallelisation in ROOT
  - Parallelises not only TDataFrame, but also ROOT I/O, etc

Sequential Code

```
TDataFrame tdf(tree, {"x","y","z"});
auto h = tdf.Filter(IsGoodEvent)
            .Histo1D();
```

Parallel Code

```
ROOT::EnableImplicitMT();
TDataFrame tdf(tree, {"x","y","z"});
auto h = tdf.Filter(IsGoodEvent)
            .Histo1D();
```

Parallelism at the reach of anyone!

Easy Programming Model via JITing

- TDataFrame is heavily templated C++ code
  - Performance, type safety
- JIT compilation at runtime for type deduction

Can write this / JIT / **Instead of this**

```
d.Histo1D("myCol");
```

JIT

```
d.Histo1D<float>("myCol");
```

```
d.Define("v1v2","v1*v2");
```

JIT

```
d.Define("v1v2",
 [](T &v1, T &v2){return v1*v2;},
 {"v1","v2"});
```

A string to replace a callable, <u>no DSL but C++ (jitted!)</u>

Example: Cuts and Histograms

- All actions are executed in the same loop
- Type inference using just-in-time compilation

Simple Analysis

```
TDataFrame d("myTree", "myFile.root");
auto hp = d.Filter("theta > 0.0").Histo1D("pt");
auto hn = d.Filter("theta < 0.0").Histo1D("pt");

hp->Draw(); // Overloaded ->: Event Loop runs once here
hn->Draw("Same"); // No need to re-run here
```

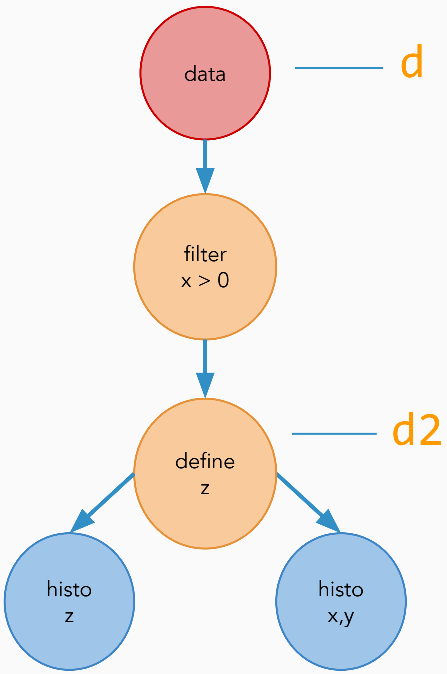
Describe all calculations first, run all of them at once later.

A Functional Graph

- More than a simple chain, a graph of actions and transformations

```
// d2 is a new data-frame, it is
// a transformed version of d
auto d2 = d.Filter("x > 0")
           .Define("z", "x*x + y*y");

// make multiple histograms out of it
auto hz = d2.Histo1D("z");
auto hxy = d2.Histo2D("x","y");
```



Complex control flows can be expressed easily

A New Way of Writing TTrees

- TDataFrame Snapshot Action
- Read data, add custom columns, write out
- Uses new TBufferMerger internally

```
TDataFrame d("myTree", "myFile.root");
auto d2 = d.Filter("0 == b1 % 2");
           .Define("b1_square", "b1 * b1");

// Write selected columns in a TTree on a TFile
d2.Snapshot("myNewTree", "myNewFile.root",
            {"b1_square", "b1"});
```

One line to write out a dataset, <u>it works in parallel too.</u>

Good performance! See also Increasing Parallelism in ROOT I/O for more information

Transformations and Actions

Transformations

- Define
- Filter
- Range

Actions

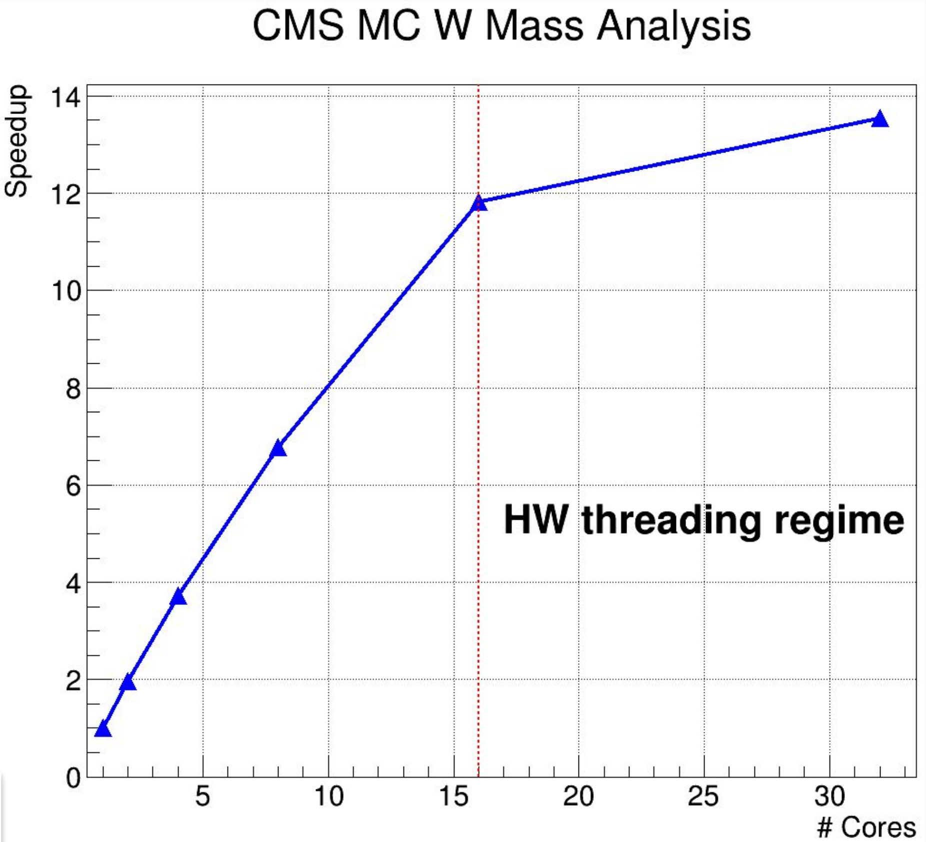
- Histograms
- Min, Max, Mean
- Profile
- Reduce
- Snapshot

See online documentation for more information

TDataFrame: Performance Figures

CMS W Mass Analysis

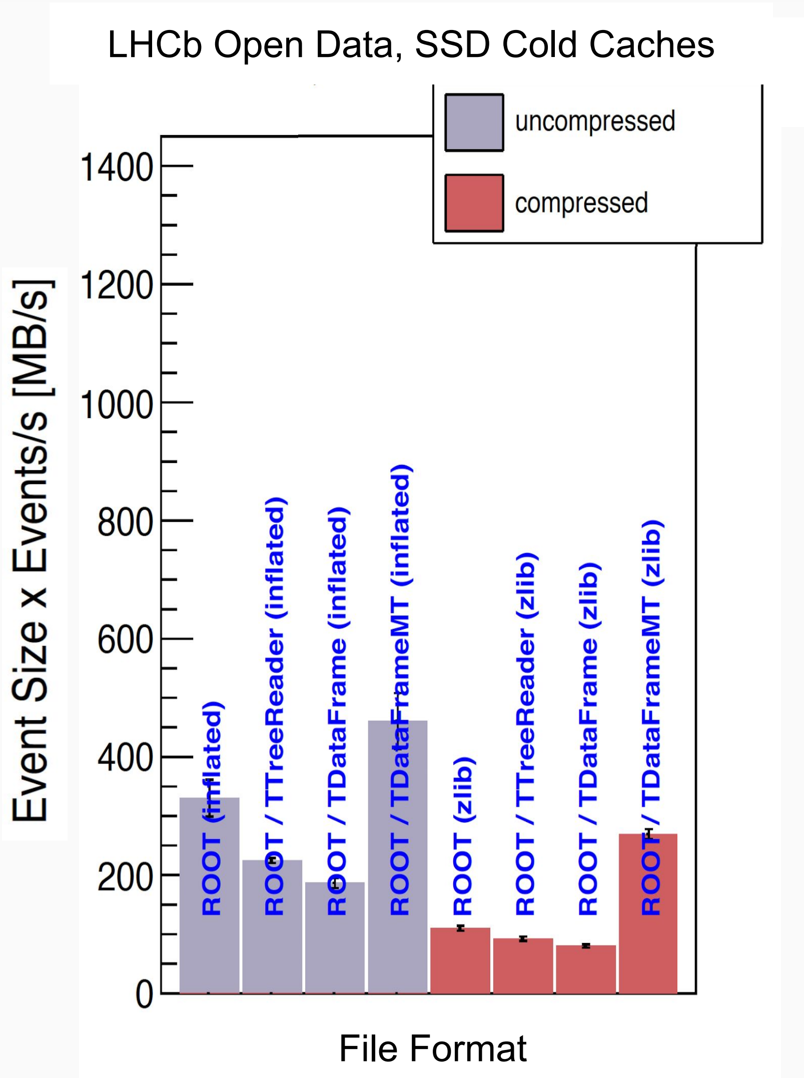
- Xeon(R) CPU E5-2650 v2 @ 2.60GHz
- 32 logical cores, 2 NUMA domains
- 2.5 GB input file (95 clusters, ~2M events)
- CMS MC analysis ntuple (smeared)
- Filling 1.1k TH3F with 70x10x10 bins
- 8 kinematic and quality cuts
- Timings include merging of histograms



12x speedup with 16 cores,
NUMA effects and merging included!

Thanks to M. Dunser

LHCb OpenData



- Laptop, 8 logical cores
- Simplified analysis
- TDataFrame: little overhead with respect to TTreeReader
  - Mostly due to Filters, optimizations under development

ImplicitMT and TDataFrame: same code, parallelism for free!

https://github.com/jblomer/iotools

See talk A quantitative review of data formats for HEP analyses by J. Blomer!

Distributing Work on Spark Resources: R&D

Parallelising ROOT with Spark

- Analyse ROOT data with PyROOT + PySpark
- Minimal interface: Map-Reduce pattern to process TTrees
- Relies on shared filesystems on the driver and worker nodes
    - For example, CVMFS and fuse-mounted EOS

```
# ROOT imports
import ROOT
from DistROOT import DistTree

# Build the DistTree
dTree = DistTree(filelist = ["myFile1", "myFile2"],
                 treename = "myTree",
                 npartitions = 8)

# Trigger the parallel processing
myHistos = dTree.ProcessAndMerge(fillHistos, mergeHistos)
```

Promising R&D

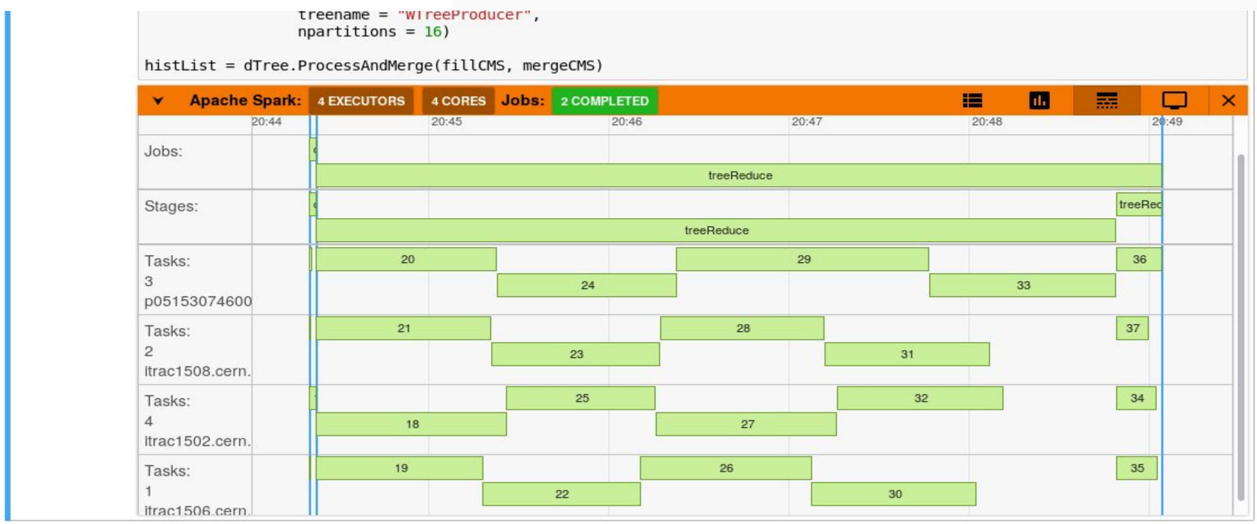
Tested on CERN infrastructure in collaboration with IT-DB and IT-ST groups.

https://github.com/etejedor/root-spark

Integrated Monitoring Infrastructure

Monitoring ROOT and other workflows on Spark clusters (Krishnan R., GSoC student)

The SWAN service (Service for Web based ANalysis) will be interfaced to CERN Spark resources.



https://github.com/krishnan-r/sparkmonitor

Bottomline

ROOT now supports declarative data analysis in C++ with TDataFrame

- PyROOT already partially supported
- Friendly programming model
- Same result with less lines of code
- Seamless implicit parallelisation
- Can be used to write datasets too!
- Distributed analysis with PyROOT and Spark
- Will be available in SWAN at CERN

Some forthcoming improvements (targeting ROOT 6.12 - November):

- Provide adapters for formats also other than ROOT (xAOD, csv, Parquet)
- Improve TDataFrame integration with PyROOT, e.g. using Python callables
- Refine writing procedure for improved performance