Even Easier Analysis with TDF

Danilo Piparo for the ROOT Team

ROOT

Data Analysis Framework

https://root.cern

What was Achieved

TDataFrame implements a powerful interaction with columnar data

- Declarative, read, write, transformations, actions
- Parallelism

We have done a lot: can we get even nearer to analysis needs?

The Problem to Solve, in Terms of TTree::Draw

```
Draw("Muon_pt", "Muon_eta> 1")
Draw("Muon_pt", "Muon_eta[0] > 1")
Draw("Muon_pt[0]", "Muon_eta[0] > 1")
Draw("Muon_pt[1]", "Muon_eta[0] > 1")
Draw("Muon_pt[0]", "Sum$(Muon_pt*(Muon_eta > 1)) > 30")
Draw("Muon_pt", "Sum$(Muon_pt*(Muon_eta > 1)) > 30")
Draw("hg[2][][36]:timesamp[]+(dacinj/4096):dacinj")
```

People do this, we need to help them

Some High Level Guidelines

We need easy paths for:

- Implicit (nested) for loops
- Operations between same size collections resulting in a collection
- Operations on collections resulting in a collection or a number
  - E.g. calling a method element by element and storing results, Sum

Challenging but *opportunity for more optimisations and data parallelism*

Example Opportunity

Sum$(Muon_pt*(Muon_eta > 1))

This is a cut + a sum over elements in a collection

- Parallelise multiplications
- Parallelise on the accumulation

Autovectorisation, veccore... Details.

Example Opportunity

Sum$(Muon_pt*(Muon_eta > 1))

This is a cut + a sum over elements in a collection

- Parallelise multiplications
- Parallelise on the accumulation

Autovectorisation, veccore... Details.

Proposals for Concrete Improvements

Operations on Colls Returning A Coll

Problem: Multiply element by element two collections, return the collection of products

Proposal: Mult<T, V=T> (const T&, const V&)

- This holds for other operations: Add, Divide …
- It works in compiled code (all types must be specified)
- Shows its full power in Jitted code

Operations on Colls Returning A Coll

```
auto f=[](const T1& col1, const T2& col2, const T3& col3) {..};
tdf.Define("results", f, {"col1", "col2", "col3"});
```

Or

```
tdf.Define("results", "Add(col1, Mult(col2, col3))");
```

The same technique works for collection to scalar functions (e.g. Sum)

Calling Methods of Objects in Containers

Problem: column holding vector<T>. Want a column with vector<R> where R is the type of the result of T::MyMethod()

Proposal:

```
vector<R> ApplyToVec<R, T>(R(T::*m()))
```

This returns a lambda: `[](const vector<T>& v) {...};`

Usage:

```
tdf.Define("results", ApplyToVec(&T::MyMethod), {"myTs"})
```

Embed Value in Histograms w/o Define

Problem: Fill a histogram with a sophisticated value created only for that and not used anywhere else.

Proposal:

Histo1D(*model,* myExpr, {"col1", "col2"})

where model could {“name”, “title”, 64, -4, 4}

Advantages:

- Smaller runtime, more concise syntax
- Interplay with previous solutions

Open Questions

Draw("hg[2][][36]:timesamp[]+(dacinj/4096):dacinj")