VecOps: Express easily common operations on collections

Danilo Piparo, Enric Tejedor, Enrico Guiraud

ROOT
Data Analysis Framework
https://root.cern

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

From Last meeting, https://indico.cern.ch/event/607858/

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*

From Last meeting, https://indico.cern.ch/event/607858/

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.

From Last meeting, https://indico.cern.ch/event/607858/

Proposals for Concrete Improvements

Minimal Set of Elements Needed

1) A library allowing easy operations (math, math functions etc.) between collections, collections and scalars
2) Upgrade TDF to avoid Define nodes for histogramming
   - tdf.Histo1D(*model,* myExpr, {"col1", "col2"}) instead of tdf.Define("q", myExpr, {"col1", "col2"}).Histo1D(model, "q")
   - tdf.Histo1D(model, "myExpr") instead of tdf.Define("q", "myExpr").Histo1D(model, "q")

Today we focus on 1)

VecOps

A library that:

- Allows to do things like `sqrt(v0*v0+v1*v1)/3` where v0 and v1 are collections
- Main item: TVec<T>
  - Same interface of a std::vector (it is a vector, with a special allocator)
  - Contiguous in memory (yes, to vectorise)
  - Operations such as *,/,-,+,>,==,< & co are possible
  - Math functions are implemented
  - Owns its content but can be a view on a contiguous memory region (to wrap TTreeReaderArrays for example)
- This exists, it's *VecOps* https://github.com/dpiparo/VecOps

VecOps

Up to here two aspects mentioned:

- Easy, vectorised operations on collections, *per se*
- Integrated in TDF for making analysis easier and more efficient

VecOps

Up to here two aspects mentioned:

- Easy, vectorised operations on collections, *per se*
- Integrated in TDF for making analysis easier and more efficient

Operations on Collections: Examples

```
std::cout << "Initialiser list ctor:" << std::endl;
TVec<float> v0{0, 1, 2, 3};
std::cout << v0 << std::endl;

std::cout << "Size ctor:" << std::endl;
TVec<int> v1(4);
std::cout << v1 << std::endl;
```

```
We start from some constructors
Initialiser list ctor:
{ 0, 1, 2, 3 }
Size ctor:
{ 0, 0, 0, 0 }
```

https://github.com/dpiparo/VecOps/blob/master/test/intro.C

Operations on Collections: Examples

```
std::cout << "Sum with scalar (3):" << std::endl;
TVec<float> v0{0, 1, 2, 3};
auto res0 = v0 + 3;
std::cout << res0 << std::endl;

std::cout << "Division by scalar (3.):" << std::endl;
TVec<int> v1{0, 1, 2, 3};
auto res1 = v1 / 3.;
std::cout << res1 << std::endl;

std::cout << "Greater than a scalar (2, note the return type, TVec<int>):" << std::endl;
TVec<double> v2{0, 1, 2, 3};
auto res2 = v2 > 2.;
std::cout << res2 << std::endl;
```

```
Sum with scalar (3):
{ 3, 4, 5, 6 }
Division by scalar (3.):
{ 0, 0.333333, 0.666667, 1 }
Greater than a scalar (2, note the return type, TVec<int>):
{ 0, 0, 0, 1 }
```

https://github.com/dpiparo/VecOps/blob/master/test/intro.C

Operations on Collections: Examples

```
v0 = { 1, 2, 3 }
v1 = { 7, 8, 9 }
v2 = { 3, 3, 4 }
v0 + 1 = { 2, 3, 4 }
v1 - v2 = { 4, 5, 5 }
(v1 - v2) / 3. = { 1.33333, 1.66667, 1.66667 }
v0 + 1 + (v1 - v2) / 3. = { 3.33333, 4.66667, 5.66667 }
(v0 + 1 + (v1 - v2) / 3.) > 4  { 0, 1, 1 }
```

```
TVec<float> v0{1, 2, 3};
TVec<char> v1{7, 8, 9};
TVec<int> v2{3, 3, 4};
std::cout << "v0 = " << v0 << std::endl;
std::cout << "v1 = " << v1 << std::endl;
std::cout << "v2 = " << v2 << std::endl;
std::cout << "v0 + 1 = " << v0 + 1 << std::endl;
std::cout << "v1 - v2 = " << v1 - v2 << std::endl;
std::cout << "(v1 - v2) / 3. = " << (v1 - v2) / 3. << std::endl;
std::cout << "v0 + 1 + (v1 - v2) / 3. = " << v0 + 1 + (v1 - v2) / 3. << std::endl;
std::cout << "(v0 + 1 + (v1 - v2) / 3.) > 4  " << ((v0 + 1 + (v1 - v2) / 3.) > 4) << std::endl;
```

https://github.com/dpiparo/VecOps/blob/master/test/intro.C

Operations on Collections: Examples

```
std::cout << "Dot of 2 TVecs of different type:" << std::endl;
TVec<int> v30{0, 1, 2, 3};
TVec<float> v31{0, 1, 2, 3};
auto res3 = Dot(v30, v31);
std::cout << res3 << std::endl;

std::cout << "Square root of a TVec:" << std::endl;
TVec<float> v40{0, 1, 2, 3};
auto res4 = sqrt(v40);
std::cout << res4 << std::endl;
```

```
Dot of 2 TVecs of different type:
14
Square root of a TVec:
{ 0, 1, 1.41421, 1.73205 }
```

https://github.com/dpiparo/VecOps/blob/master/test/intro.C

VecOps

Up to here two aspects mentioned:

- Easy, vectorised operations on collections, *per se*
- Integrated in TDF for making analysis easier and more efficient

TDF Integration: Examples

```
TDataFrame tdf(8);
tdf.Define("px", rndmVector)
   .Define("py", rndmVector)
   .Snapshot<std::vector<double>,std::vector<double>>("t", "dataset.root", {"px", "py"});

auto f = TFile::Open("dataset.root");
TTreeReader myReader("t", f);
TTreeReaderArray<double> px(myReader, "px");
TTreeReaderArray<double> py(myReader, "py");

// So far so good. Now the serious stuff
TH1F h("myhisto", "The Histo", 64, 0, 2);
while (myReader.Next()) {
   auto pxpp = (double**)px.GetAddress();
   auto pypp = (double**)py.GetAddress();
   ROOT::Detail::VecOps::TVecAllocator<double> allpx(*pxpp, px.GetSize());
   ROOT::Detail::VecOps::TVecAllocator<double> allpy(*pypp, py.GetSize());
   const TVec<double> pxv(px.GetSize(), double(), allpx);
   const TVec<double> pyv(py.GetSize(), double(), allpy);
   std::cout << pxv << " " << pyv << std::endl;
   std::cout << pxv*pyv << std::endl;
}
```

```
auto rndmVector = []() {
   std::vector<double> v(8);
   for (auto &&e : v) {
      e = gRandom->Gaus();
   }
   return v;
};
```

It is also a *view*, <u>until one reallocates!</u>
Copy performed, then “normal” container

https://github.com/dpiparo/VecOps/blob/master/test/intro.C

TDF Integration: Examples

```
auto rndmVector = []() {
   TVec<double> v(8);
   for (auto &&e : v) {
      e = gRandom->Gaus();
   }
   return v;
};

TDataFrame tdf(8);
auto df = tdf.Define("Muons_px", rndmVector).Define("Muons_py", rndmVector);
auto h = df.Define("Muon_pt", "sqrt(Muons_px*Muons_px + Muons_py*Muons_py)").Histo1D("Muons_pt");

::TCanvas c;
h->Draw();
c.Print("myHist.png");

// ROOT-8865
// Draw("Muons_px","Muons_py[0] > 1")
auto h0 = df.Define("q0", "Filter(Muons_px, Muons_py[0] > 1)").Histo1D("q0");
// Draw("Muons_px","Sum$(Muons_px*(Muons_py > 1)) > 30")
auto h3 = df.Define("q3", "Filter(Muons_px, (Sum(Muons_px*(Muons_py > 1)) > 30))").Histo1D("q3");
```

Integrated with ROOT in a private branch
https://github.com/dpiparo/root/tree/vecopsIntegration

Implementation Detail

```
27  namespace VecOps {
28
29  template<typename T>
30  using TCallTraits = typename ROOT::TypeTraits::CallableTraits<T>;
31
32  template <typename T>
33  using TVec = std::vector<T, ROOT::Detail::VecOps::TVecAllocator<T>>;
34
35  } // End of Experimental NS
36
37  } // End of VecOps NS
38
```

TVec<T> is a vector with a special allocator.