The Nuon Model in TDataFrame

Xavier Valls

ROOT

Data Analysis Framework

https://root.cern

Rene’s Totem

Simulates random pairs of protons interactions (one per event) and writes the data into histograms.

Uses POSIX threads for parallelism.

Advantages of TDataFrame

- More intuitive expression of Filters
- Less clutter, no need for thread-management instructions
- No need to think about parallelism*.

Lacking vs Original

Lacking from the original:

1. Partial writing of files.
2. Debugging

Conversion to TDataFrame

```
class collisionData {
public:
   collisionData(){}
   collisionData(bool hC, Double_t &r,Double_t &w,Double_t &f):
      hasCollided(hC), roffset(r), wevent(w), fr(f){}

   bool hasCollided = false;
   Double_t roffset{};
   Double_t wevent{};
   Double_t fr{};
};
```

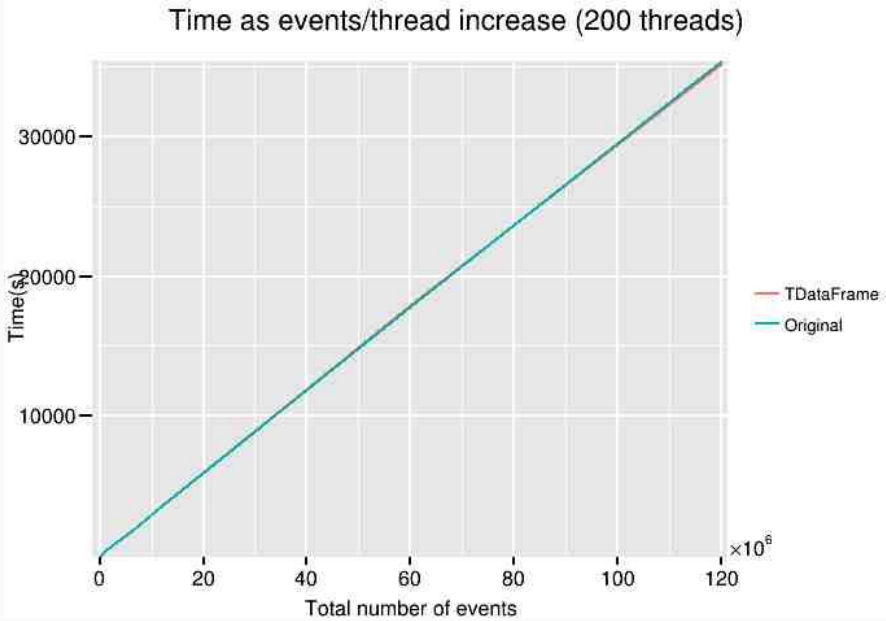
```
auto collideOps = [&](unsigned int slot) {
    // ... Intensive physics computations
}
```

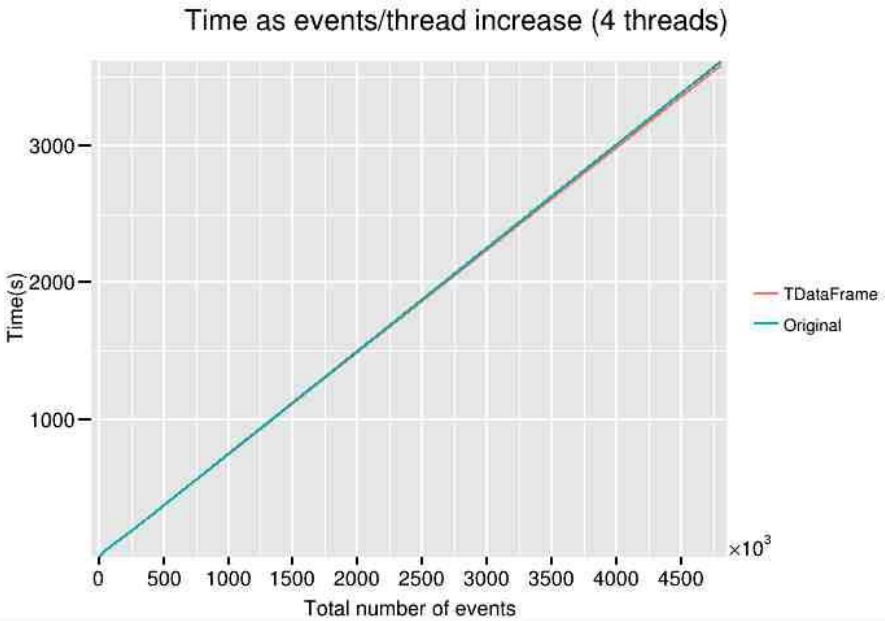
```
// Helpers for clarity
auto hasCollided = [](const bool &hasCollided){return hasCollided;};
auto hasNotCollided = [](const bool &hasCollided){return !hasCollided;};
```

The Analysis

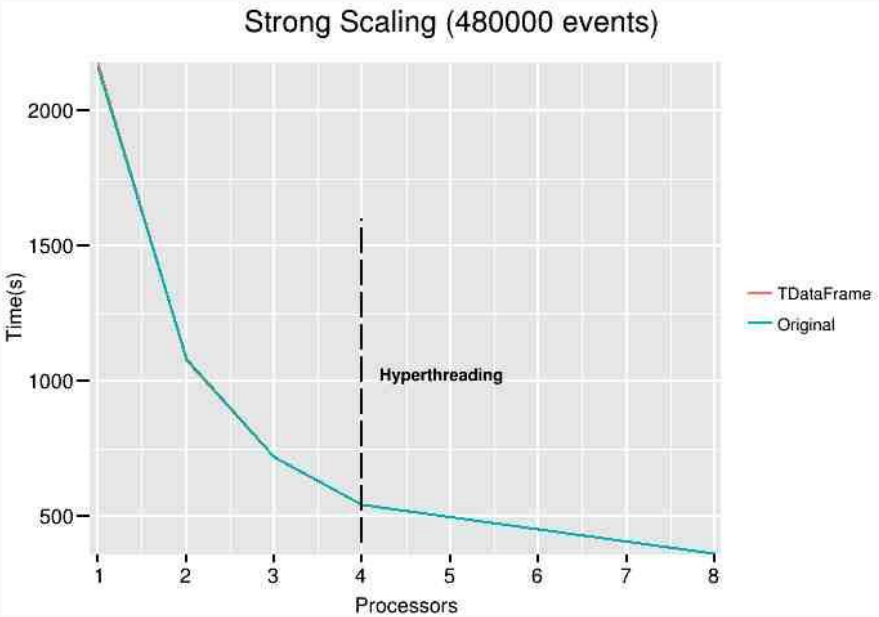
```
ROOT::Experimental::TDataFrame d(maxthreads*nEvents_p);
auto d2 = d.DefineSlot("collisionData", collideOps)
          .Define("hasCollided", [&](const collisionData &colData){return colData.hasCollided;}, {"collisionData"})
          .Define("roffset", [&](collisionData &colData){return colData.roffset;}, {"collisionData"})
          .Define("wevent", [&](const collisionData &colData){return colData.wevent;}, {"collisionData"})
          .Define("fr", [&](const collisionData &colData){return colData.fr;}, {"collisionData"});

auto helast = d2.Filter(hasNotCollided, {"hasCollided"})
                .Filter([](const double &offset){return offset < 1.01;}, {"roffset"})
                .Histo1D(TH1D("helast","elastic collisions in proton radius range",100,distmin,1.01), "roffset");

auto hfr = d2.Filter(hasNotCollided, {"hasCollided"})
             .Histo1D(TH1D("hfr","p-p elastic", ntbins, tbins), "fr", "wevent");

auto hfr1 = d2.Filter(hasNotCollided, {"hasCollided"})
              .Filter([](const double &offset){return offset < 1;}, {"roffset"})
              .Histo2D(TH2D("hfr1", "hfr1", 100, 0, 0.5, 100, distmin, 0.7), "fr", "roffset", "wevent");
auto hfr2 = d2.Filter(hasNotCollided, {"hasCollided"})
              .Histo2D(TH2D("hfr2", "hfr2", 100, tmin, tmax, 100, distmin, 1), "fr", "roffset", "wevent");

auto hcoll = d2.Filter(hasCollided, {"hasCollided"})
               .Filter([](const double &offset){return offset < 1.01;}, {"roffset"})
```

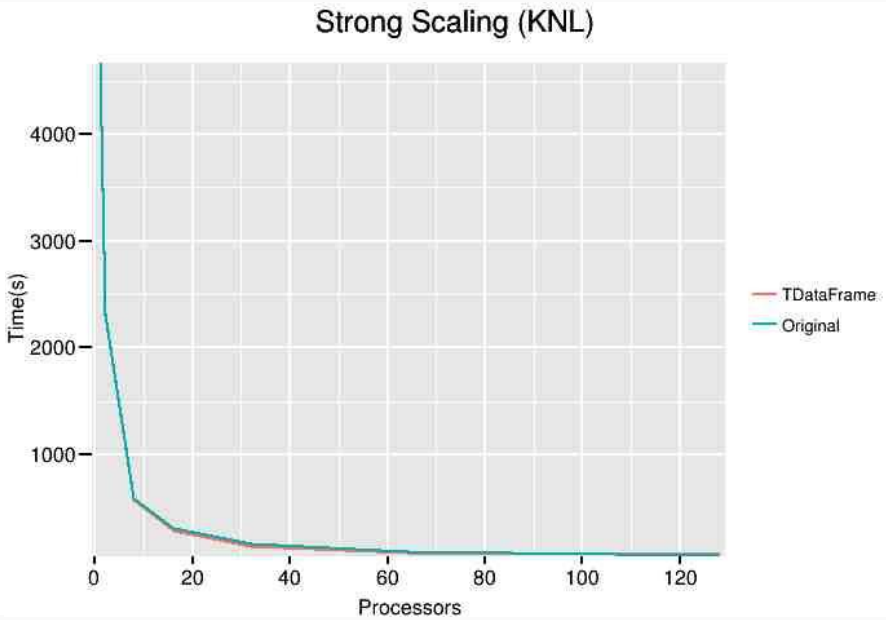
Where is my line



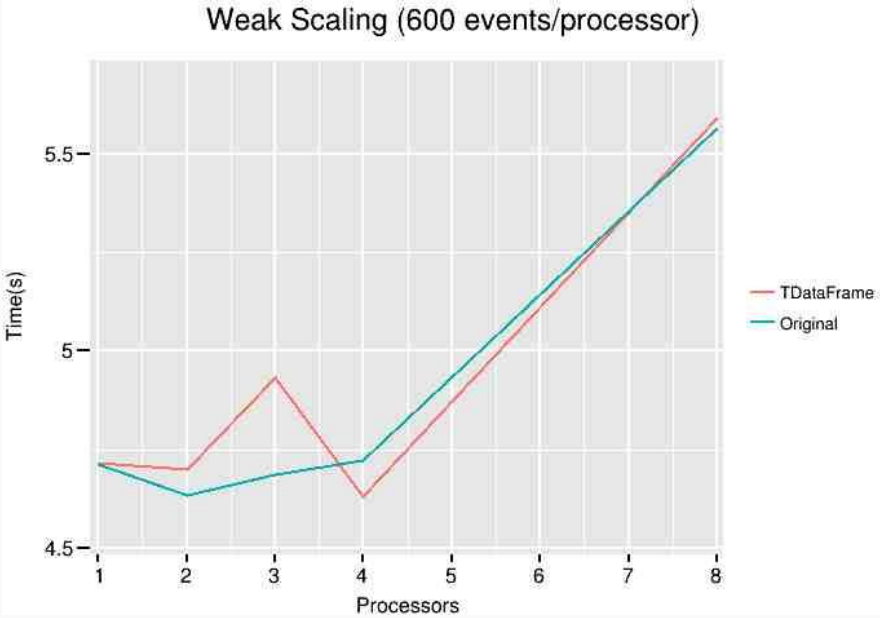


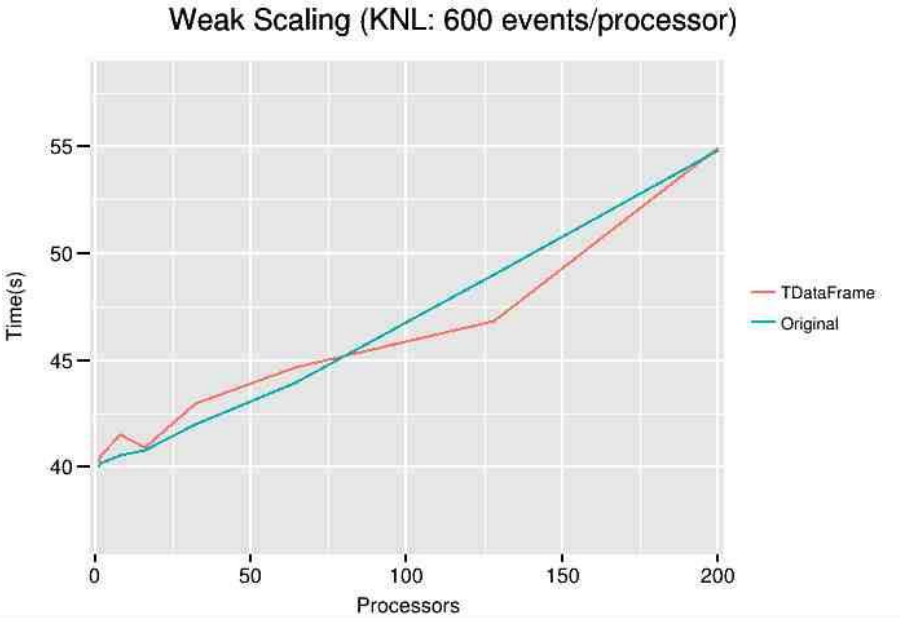
Strong Scaling



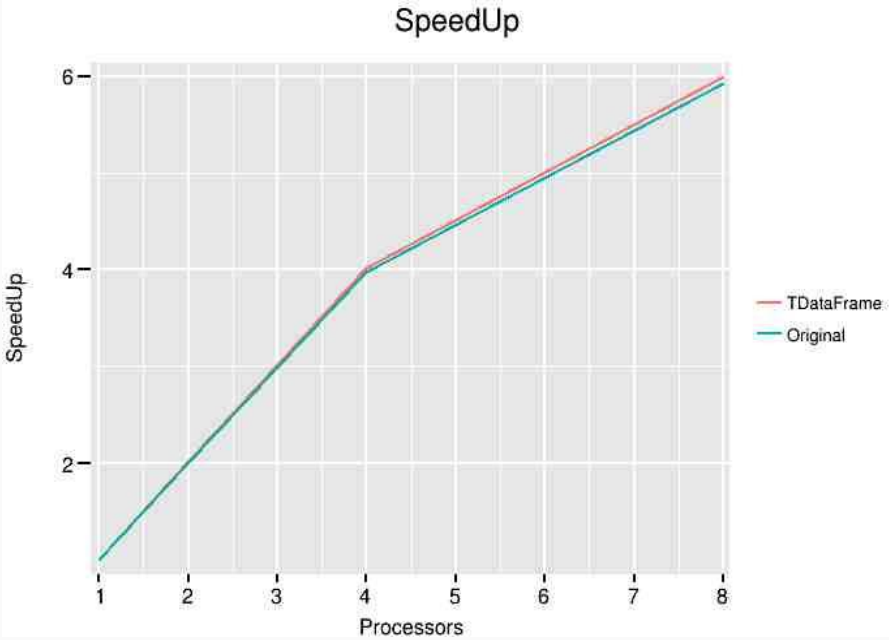


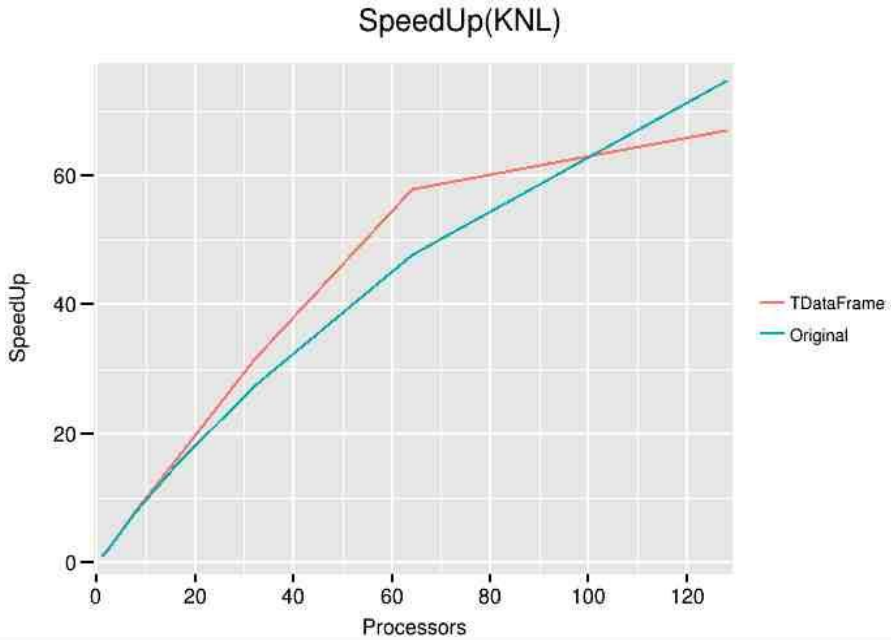
Weak Scaling



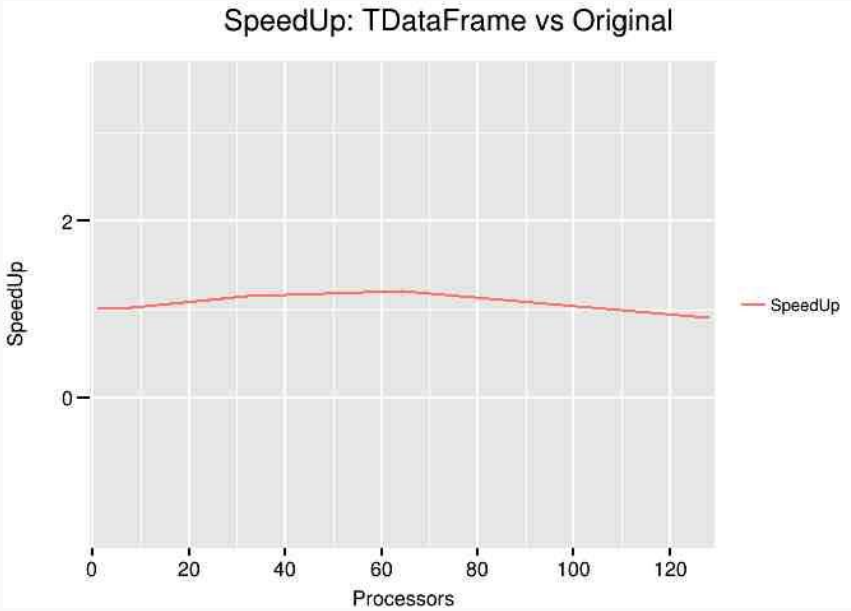


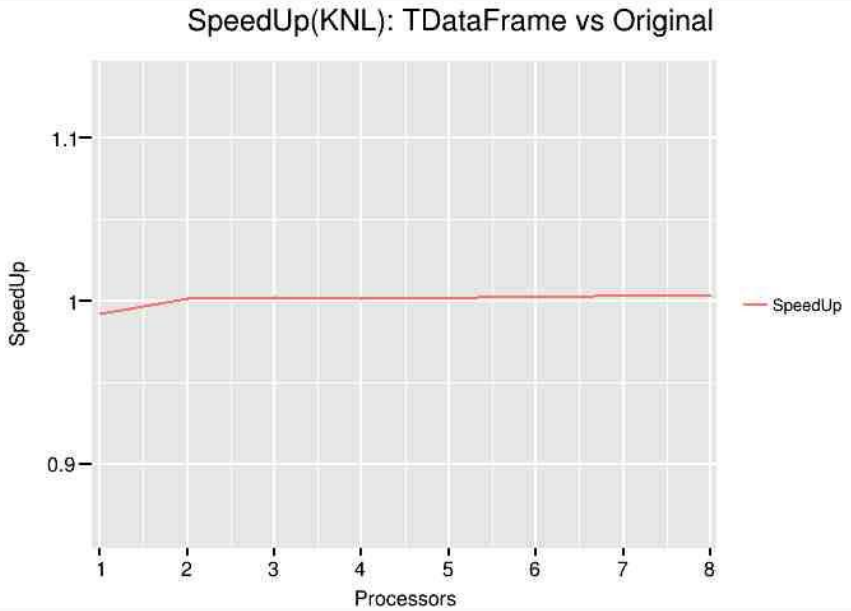
Speed Up


SpeedUp
SpeedUp
Processors
TDataFrame
Original
1
2
3
4
5
6
7
8
2
4
6


SpeedUp(KNL)
SpeedUp
Processors
TDataFrame
Original
0
20
40
60
80
100
120
0
20
40
60

Speed Up between implementations





Things left to check

1. Profiling.
2. Jitted against not jitted histograms
3. Reduce the grain of the problem to improve balancing (If 1. shows unbalance)