The Nuon Model in TDataFrame (Update II)

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. Debug defined sections (not run when benchmarking)

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;};
```

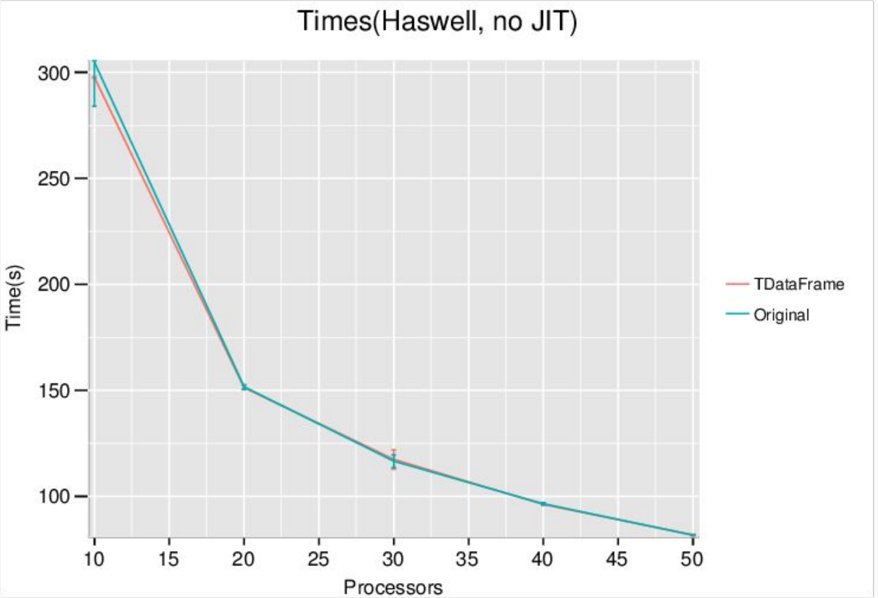
The Analysis

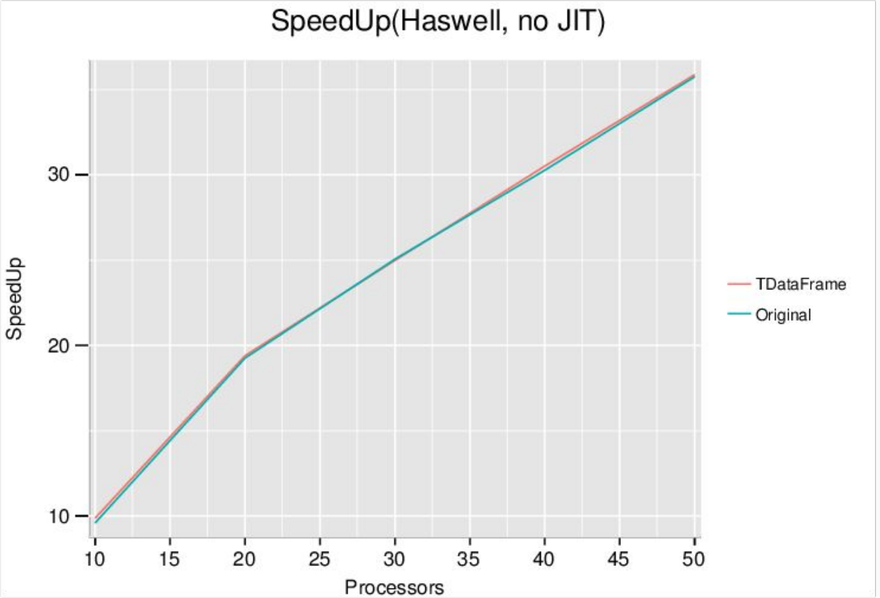
```
ROOT::Experimental::TDataFrame d(maxthreads*nEvents_p);
auto d2 = d.DefineSlot("collisionData", collideOps)
          .Define("hasCollided", [&](const collisionData &colData){return colData.hasCollided;}, {"collisionData"})
          .Define("roffset", [&](collisionData &colData){return colData.roffset;},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

Times (Xeon, 28 cores)

Number of threads	TDataFrame (seconds)	Original (seconds)
1	2932.951	2919.44
10	297.663	304.7534
20	151.3235	151.6379
30	117.3833	116.5535
40	96.16202	96.4905
50	81.75429	81.65801

Results: 28 physical cores (Xeon)

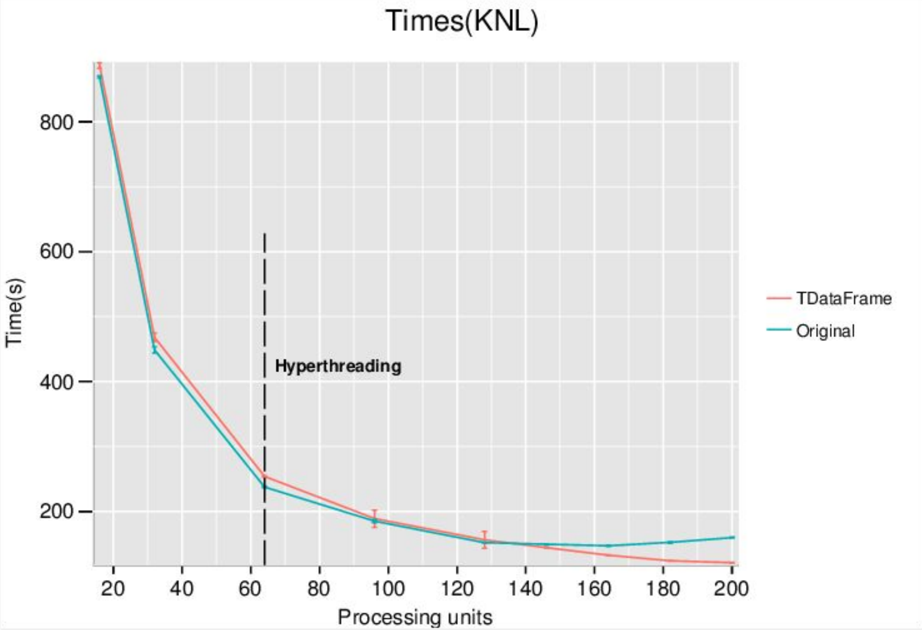


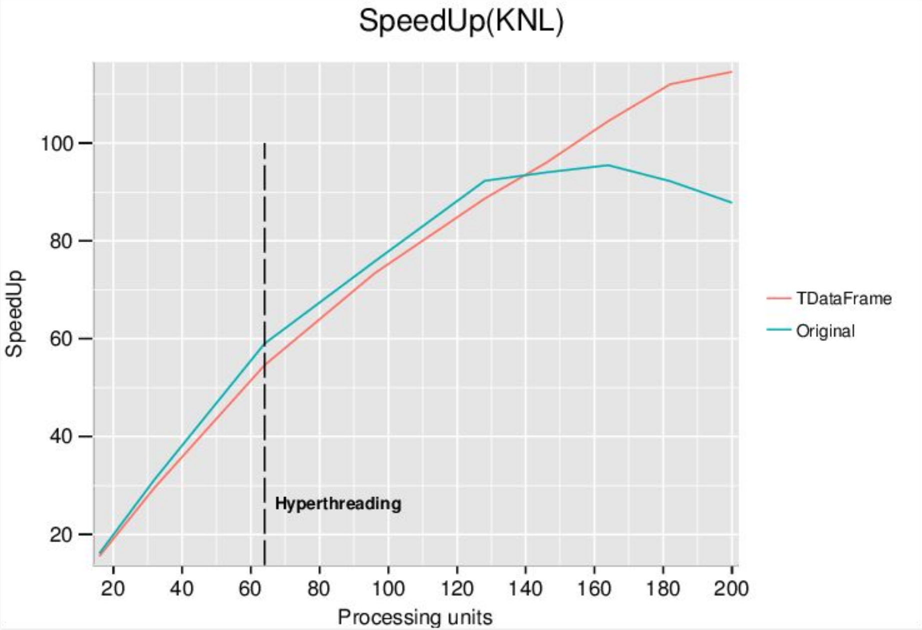


Times (Xeon Phi 64, cores)

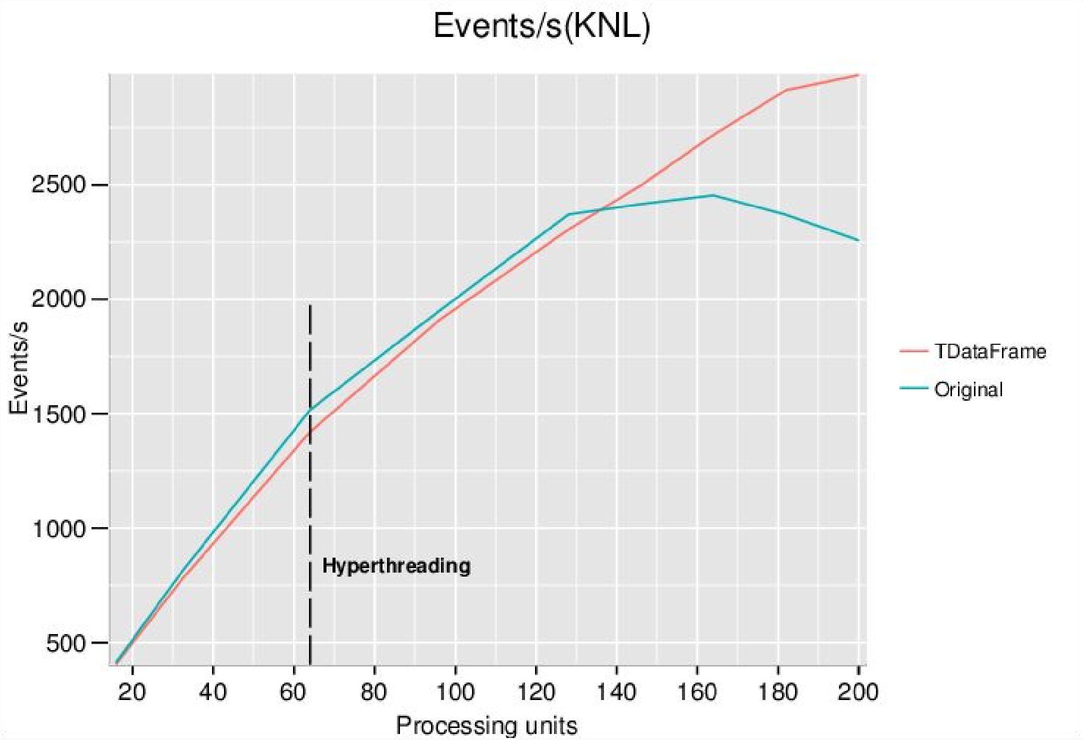
Number of threads	TDataFrame (seconds)	Original (seconds)
1	13842.96	14011.58
16	887.4904	869.3472
32	468.1272	448.5831
64	253.5476	237.5605
96	188.6633	184.8104
128	156.2084	151.8857
146	144.2199	149.0883
164	132.4937	146.7849
182	123.5787	151.9972
200	120.8362	159.6057

Results: 64 physical cores (Xeon Phi)





Results: 64 physical cores (Xeon Phi)



Timing one event

Time per event executing 3000 events		
	TDataFrame	Original
Desktop	0.0051	0.0046
Xeon	0.0094	0.00834
Xeon Phi	0.043	0.038

Time per event executing 1 event		
	TDataFrame	Original
Desktop	1.923	1.00154
Xeon	3.59735	1.0017
Xeon Phi	14.9339	1.00366

Things left to check

1. Profiling.
2. Time writing against generation
3. Jitted-not jitted histograms (depends on 2)
4. Reduce the grain of the problem to improve balancing (If 1. shows unbalance, depends on 2)