A Parallelised ROOT for Future HEP Data Processing

D. Piparo (CERN, EP-SFT) for the ROOT team

ROOT
Data Analysis Framework

https://root.cern

This talk

- Challenges ahead of us
- ROOT's approach to parallelism
- Parallel components
- The future
  - Distributed analysis
  - Heterogeneous platforms

Unless explicitly stated, everything available in ROOT 6.14!

Opportunities Ahead of Us

- HL-LHC: Challenge for data processing and analysis SW
    - In both areas ROOT is a key component
- Parallelism: not the solution, a prerequisite
- Find and create opportunities for parallelism in ROOT
    - Replace components for which evolution is not possible
    - Provide programming model which makes scientists productive - cannot require too broad technical skill set from neophytes

ROOT’s Approach to Parallelism

Parallelism in ROOT

Implicit parallelism: operations run in parallel w/o user's intervention

- Just invoke `ROOT::EnableImplicitMT()`
- Task based backed by multithreading, TBB library in the backend
  - Must not overcommit node, can share pool with other libraries
- Data parallelism: 1st class citizen (VecOps) with vectorisation support

Explicit parallelism: user expresses parallelism, ROOT provides low level tools to do that efficiently

- Map, MapReduce helpers (`T{Process,Thread}Executor`)
- Forking based multiprocessing backend in ROOT
- Mutexes, async launcher

Classic Interfaces

```
TFile f(filename);
TTreeReader tr(treename, &f);
TTreeReaderArray<double> px(tr, "px");
TTreeReaderArray<double> py(tr, "py");
TTreeReaderArray<double> E(tr, "E");

TH1F h("pt", "pt", 16, 0, 4);

while (tr.Next()) {
   for (auto i=0U;i < px.GetSize(); ++i) {
      if (E[i] > 100) h.Fill(sqrt(px[i]*px[i] + py[i]*py[i]));
   }
}
h.Draw();
```

- Get a dataset in a file
  - Columns: px, py, E (collections)
- Fill a histogram with square sum of px and py for entries where E >100

Imperative way, explicit double loop

Ergonomic Interfaces

Runs in parallel!

Collections, operations vectorised

A cut

```
ROOT::EnableImplicitMT();
RDataFrame f(treename, filename);
f.Define("good_pt", "sqrt(px*px + py*py)[E>100]")
 .Histo1D({"pt", "pt", 16, -.5, 3.5}, "good_pt")->Draw();
```

Declarative, type safe, jit to simplify, task parallelism, vectorised operations on collections

See E.Guiraud [RDataFrame: Easy Parallel ROOT Analysis at 100 Threads](), Track 6, Hall 9, 10th July 11:45

Parallelised Components

What is Implicitly Parallelised?

ROOT::EnableImplicitMT() activates parallelised:

- RDataFrame event-loop
- TTree::GetEntry read of multiple branches
- TTree::FlushBaskets write of baskets
- TTreeCacheUnzip decompression of TTreeCache content
- TH{1,2,3}::Fit evaluation of the objective function over the data
- TMVA::DNN trains NN in parallel

And **more** to come!

IMT Effect On CMS Data Processing

32 thread RECO-AOD-MINIAOD

Module	Total Loop Time	Total Loop CPU	CPU Utilization	Events/ Second	RSS
Standard w/o IMT	1		62	2.94	9454
Standard w/IMT	1		34	4.21	8981
Parallel 6x6x3				4.47	13817
Parallel 1x6x3				4.59	10745
NoWrite			96	4.65	12140
NoFill			91	5.41	7201

Activate Implicit MT: +43% Throughput

Add Parallel Writing: +13% Throughput

18% away from asymptotic value (not filling nor writing output datasets)

D. S. Riley, [CMS And ROOT IO](), ROOT IO Workshop, 20 June 2018, CERN
See G. Amadio [Writing ROOT Data in Parallel with TBufferMerger]() Track 5, Hall 3, 10th July 12:15

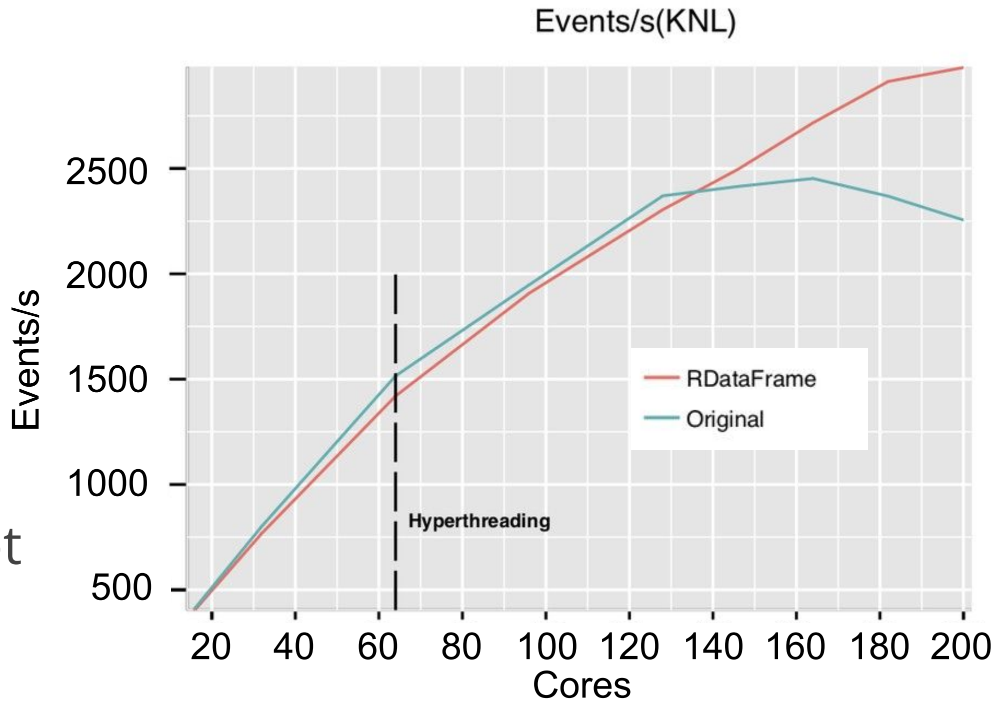
Extreme Architectures

No IO, KNL, 64 Physical cores

Monte Carlo QCD low-pt events generation+analysis on the fly

Ad-hoc implementation (patched ROOT5 & POSIX threads) Vs RDataFrame

Original code by experienced developer (R. Brun), intentionally not thread safe (*RDF always is*)



ROOT Can Scale Well on Extreme Architectures

Spotlight on TMVA parallelism

BDT's: implicit parallelism

- Specific operations in tree construction process
- x 1.6 speedup for 4 threads

Cross Validation: multiprocessing based

- Evaluate each fold independently
- Almost linear scaling!

And of course, Cuda based implementation of DNNs in TMVA

See K.Albertsson New ML Developments in ROOT/TMVA , Track 6, Hall 9, 9th July 11:15

Spotlight on Math parallelism

Evaluation of objective functions is parallelised and vectorised

- Adapt TF1, TFormula, fitting internal classes
- Leverage ROOT::Double_v SIMD type - based on VecCore
- AVX2, 2x2 cores: factor 10x not uncommon!

Introduced `ROOT::RVec<T>`: vectorised operations made easy

- `std::vector` like interface, ergonomic support of analysis operations
- Can adopt memory or own it
- Vectorised arithmetic operations, math functions

DOI 10.5281/zenodo.1253756

See also L. Moneta Vectorisation of ROOT Mathematical Libraries , Track 5, Hall 3, 9th July 15:45

ROOT::RVec<T> In Action

```
RVec<double> mus_pt {15., 12., 10.6, 2.3, 4., 3.};
RVec<double> mus_eta {1.2, -0.2, 4.2, -5.3, 0.4, -2.};
RVec<double> good_mus_pt = mus_pt[mus_pt > 10 && abs(mus_eta) < 2.1];
```

```
RVec<float> vals = {2.f, 5.5f, -2.f};
RVec<float> sin_vals = sin(vals);
```

Already integrated with RDataFrame

```
ROOT::EnableImplicitMT();
RDataFrame f(treename, filename);
f.Define("good_pt", "sqrt(px*px + py*py)[E>100]")
 .Histo1D({"pt", "pt", 16, -.5, 3.5}, "good_pt")->Draw();
```

The Future

Distributed Analysis

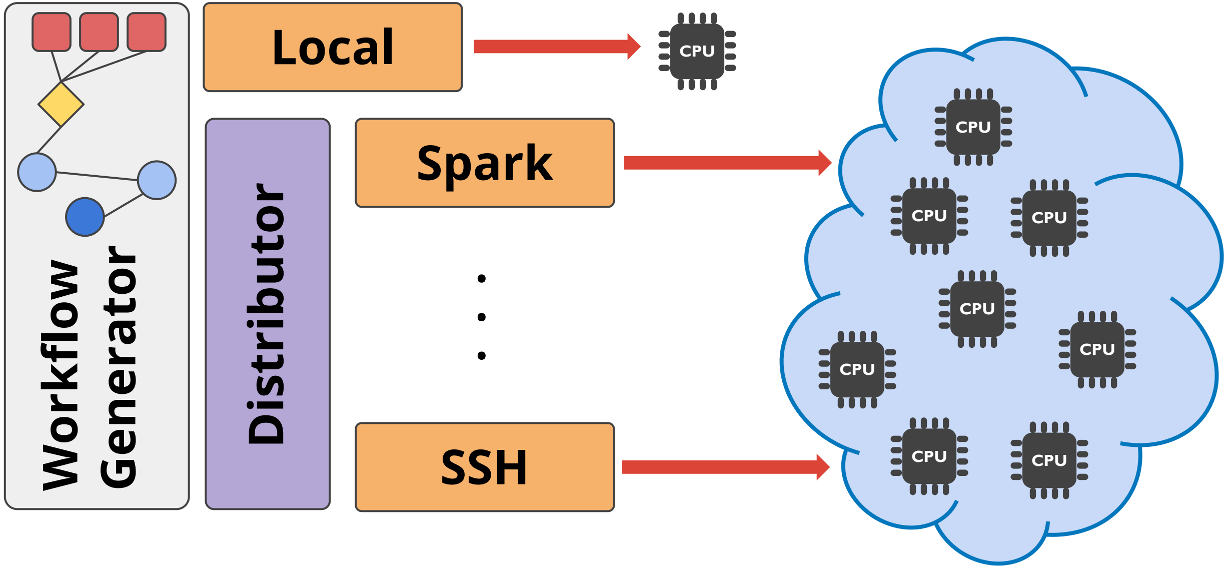
Investigate and prototype a complement to PROOF

- Parallelism on many nodes
- Transparent distribution
- Support several different backends

Minimal/No change in analysis code

```
d = RDataFrame ("t", dataset)
f = d.Define(...)
     .Define(...)
     .Filter(...)

h1 = f.Histo1D(...)
h2 = f.Histo1D(...)
h3 = f.Histo1D(...)
```



[shravan97/PyRDF](https://github.com/shravan97/PyRDF)

Not in 6.14
Working prototype available!

Support for Heterogeneity

Key element of future HEP software and computing

- TMVA already takes advantage of CUDA (DNNs)

Work ongoing to access NVidia devices from ROOT's interpreter:

- Allow to interpret CUDA code
  - `gKernel1<<<1,1>>>(deviceOutput1);`
  - More than plans: pieces already in ROOT master branch!
- Supports templates, runtime shared memory

Thanks to Simeon Ehrig for diving into the Cling-CUDA integration work!

Conclusions

ROOT: getting ready for the HL-LHC, also through parallelisation

- Emphasis on programming model, runtime and scaling

Substantial parallelism delivered in ROOT 6.14

- Scaling at the level of ad-hoc solutions written by experts
- Boost CMS amount of evts/s processed
- Parallelism in TMVA and fitting: factors can be achieved

Many opportunities ahead of us

- Provide a distributed system to further scale
- Embrace heterogeneity
- Drive renovation of ROOT with natively parallel components only