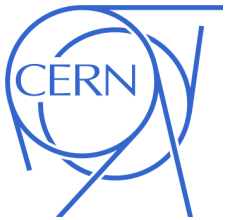
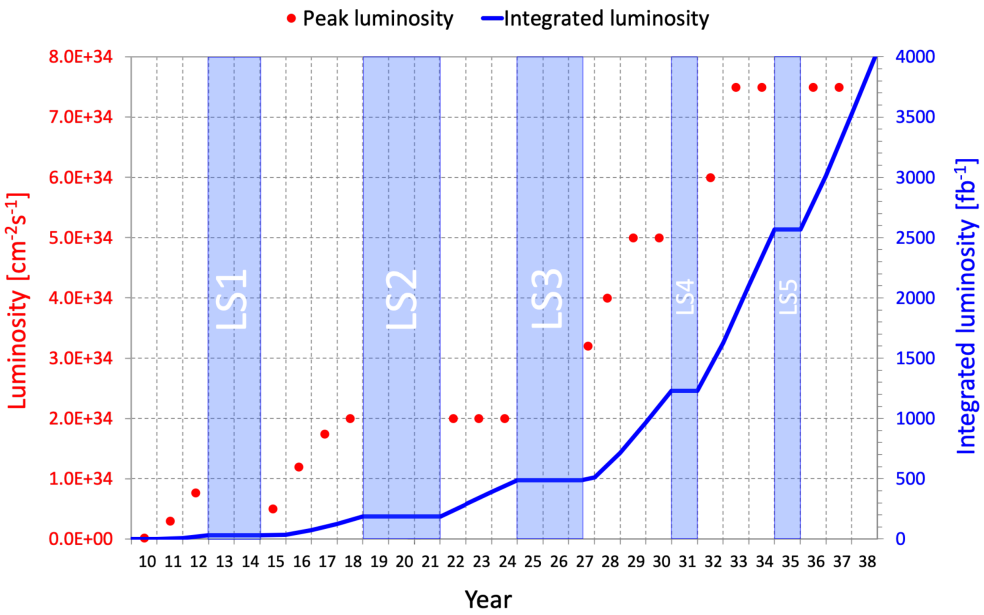
High performance Analysis for m_W Measurement

Josh Bendavid, Marco Cipriani, Marc Dünser (CERN)
Kenneth Long (MIT)



Jan. 26, 2022
MIT Physics Computing Workshop

Introduction



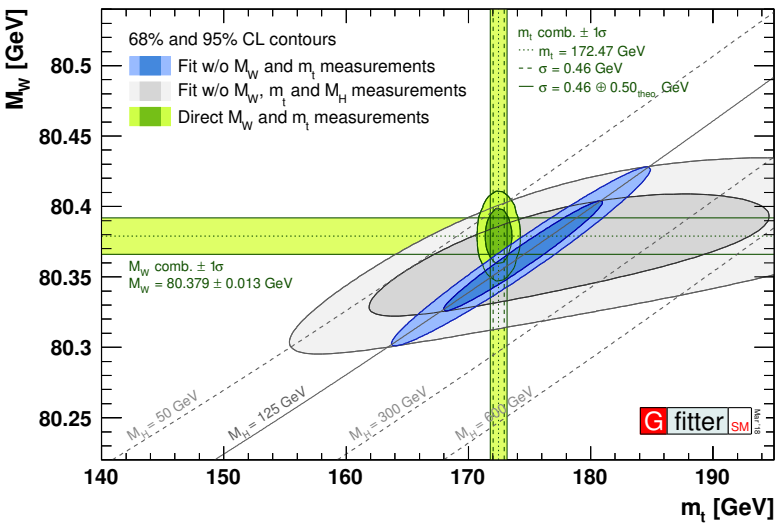
CMS-TDR-0122

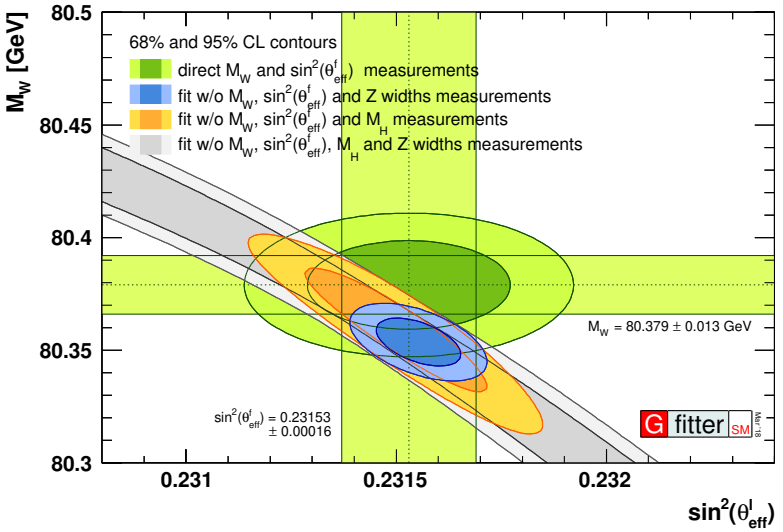
- **Huge** amount of data to be collected in HL-LHC-era, 20x increase over today
- Interplay between integrated luminosity, physics program, trigger strategy, but ∼**all searches and measurements across all final states/phase space regions will have significantly more data and MC to analyze**

Precision W measurements as a prelude to HL-LHC computing

- Personally working on precision W measurements in CMS
- Inclusive W production is among the highest cross section electroweak processes at the LHC $\rightarrow$ more than $3 \times 10^9 W \rightarrow \ell\nu$ produced per lepton flavour in LHC run 2 per experiment
- Example analysis **for** $1/4$ **of total run 2 integrated luminosity and one lepton flavour**:
  - 800M single lepton-triggered data events with little to no scope for skimming
  - 1.5B **Signal** Monte Carlo events with little to no scope for skimming
- For this type of analysis **HL-LHC is now**

Electroweak Parameters





Eur. Phys. J. C78, 675 (2018)

- Precise measurements of the Higgs mass enable more precise consistency tests of the Standard Model using m_W and $\sin^2\theta_W$

Analysis Needs

- What do we need from our software/hardware/workflows to do effective physics analysis?
  - **Fast** turnaround "as the physicist waits" $\rightarrow$ fast iteration time is essential for debugging experimental/theoretical/technical issues and for developing/improving the analysis
  - **Flexible** capabilities for binning/categorization/fitting/systematic uncertainties ~~"We know how to implement this in a more correct/robust/sensitive manner but it's not technically feasible."~~

Analysis Steps (LHC Run 2/3)

- Representative workflow for CMS:

1. Central generation/simulation/reconstruction on the grid of MC/data to MINIAOD output O(30kB/event), Root files with CMS reconstruction object structure $\rightarrow$ **months**
2. Central or private production on the grid of O(1kB)/event Root files with TTrees of basic types/arrays ("flat" is a misnomer) e.g. centrally supported/produced NANOAOD format $\rightarrow$ **days**
3. NANOAOD or similar $\rightarrow$ "final" histograms for plotting/statistical interpretation (or very condensed dataset for unbinned fits) **hours or less**
4. Visualization and statistical interpretation $\rightarrow$ **hours or less**

- (+ auxiliary workflows for calibrations/corrections)$\rightarrow$ **hours or less**

Analysis Steps (LHC Run 2/3)

- Representative workflow for CMS:

1. Central generation/simulation/reconstruction on the grid of MC/data to MINIAOD output O(30kB/event), Root files with CMS reconstruction object structure
2. Central or private production on the grid of O(1kB)/event Root files with TTrees of basic types/arrays ("flat" is a misnomer) e.g. centrally supported/produced NANOAOD format
3. **NANOAOD or similar $\rightarrow$ "final" histograms for plotting/statistical interpretation (or very condensed dataset for unbinned fits)**
4. Visualization and statistical interpretation

Analysis speed/flexibility for "Final" reduction step

- Something in the analysis looks problematic or could possibly be improved:
  1. Implement the change in easy to read/understand/maintain code representing the high level analysis logic
  2. Go for coffee or lunch
  3. Look at the results and consider next steps
- If you had to go for coffee or lunch during step 1 your code is too hard to maintain
- If the results aren't ready by the time you come back from step 2 your code/software/infrastructure is too slow
- **For O(billions) of events this means event throughput in the MHz, data throughput in the GBytes/sec**

What is needed to get there?

- Modern/easy to learn/use software frameworks and interface
  $\rightarrow$ consensus building around python for user-facing interface
- **Smart** parallelism
  - Latency matters just as much as throughput
  - Avoid IO bottlenecks
  - Avoid serialized bottlenecks
  - Avoid processing tails
- **Probably bad:**
  - Submit 1000 single core jobs to condor batch system reading from mass storage, writing O(200MB) of histograms to afs
  - Resubmit the 30 or 40 jobs which failed the first time (and the 5 or 10 which failed the second time)
  - Merge the 20GB of histogram output
  - Move to the next step

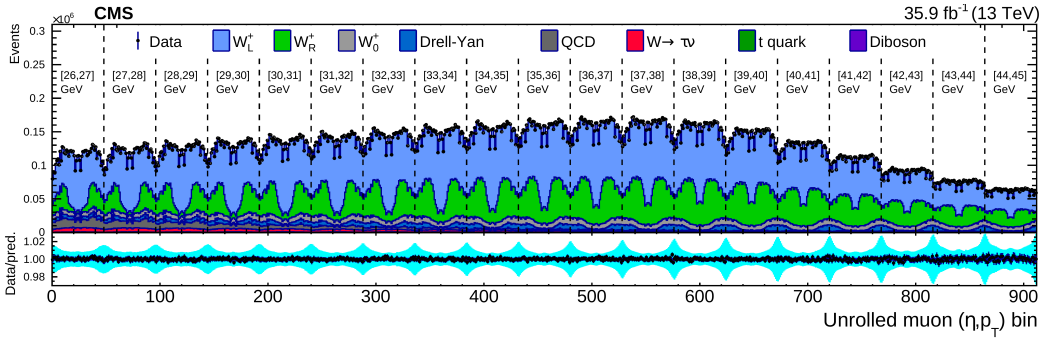
What is needed to get there: Parallelization

- **Probably better:**
  - **Multi-thread or multi-process parallelism on a single node** with shared memory/inter-process communication to merge results
    - Recent radical increases in CPU density and SSD performance allow single-node scaling to go quite far
    - Watch out for lock contention, Amdahl's law, python GIL
  - **Task-based multi-node scaling** with communication over the network (Spark, Dask, etc)
    - Can maintain "interactive-like" user-facing behaviour
    - Much more flexible scaling and provisioning, more efficient resource utilization, but more challenges for robustness and avoiding IO bottlenecks

How do we do this for m_W measurement?

- NANOAOD $\rightarrow$ histogram step in RDataFrame (python interface) with large interactive machine at CERN (single multithreaded process)
  - 128 cores/256 threads, 1TB memory, 100gbps network, 100Gbytes/sec sequential read bandwidth from disk (16 x gen4 NVME)
- SUBMIT provides similarly high performance resources, but spread over several medium-sized nodes
- **Challenge/Goal:** Get analysis running on SUBMIT in a distributed manner while maintaining the performance and latency of the single node case
  - Will rely on relatively new/in-progress support for Dask in RDF, but need to also maintain/add support for new/advanced RDF features
  - Multithreaded tasks are essential for memory reasons

In Practice



arXiv:2008.04174, (related example for illustration)

- For 2016 postVFP (16 fb^{-1}): 400M data events, 700M W/Z signal, 500M background (2TB total), O(1000) systematic variations
- Filling $\sim$ 70x 5D histograms with RDF (using **new HistoND functionality**)
- (p_T, η, charge, signal/control region idx, systematic idx) x (systematic groups, processes)
- 20 minutes to final histograms on high performance interactive machine (with significant further optimization possible via vectorized/array filling of systematic histograms, etc)
- **Already hit the per-thread memory limit with all processes simultaneously** $\rightarrow$ migrated to intermediate boost histograms with std::atomic<double> for concurrent filling

In Practice

- Technical optimizations have required bleeding edge versions of ROOT, gcc, boost, Eigen, numba, jax, tensorflow etc
- Containers (podman/singularity) are essential
- Heavy reliance on **just-in-time-compilation** of complex template instances using PyROOT (relying on recent/in-progress performance improvements in Cling)

Muon Calibration: Gradient Aggregation

- Aside from high level analysis, a number of auxiliary measurements of corrections and calibration constants
- Muon momentum scale calibrations are based on a high granularity correction for B-field, material and alignment residuals, approaching complexity of full tracker alignment
- In large debugging version which was being used for R&D, $\sim$130k parameters, 7TB of flat trees
- Taking the same workflow and running with tight selection of tracks on top of the large trees $\rightarrow$ reduced CPU load $\rightarrow$ **workflow becomes largely IO limited**
- Calibration turnaround time from trees with refit track paramters gradients $\rightarrow$ calibration parameters in O(1 hour)

Benchmark: Gradient Aggregation: IO Limits

CPU	Storage	Avg. Rate (GBytes/sec)
2 x Xeon (32C/64T)	eos/xrootd (eoscms) (25gbps)	1.64
2 x Xeon (32C/64T)	eos/xrootd (test inst.) (25gbps)	2.62
2 x Xeon (32C/64T)	CephFS HDD (25gbps)	2.60
2 x Xeon (32C/64T)	CephFS SSD (25gbps)	2.64
2 x Xeon (32C/64T)	Local SSD (16xSATA)	5.21

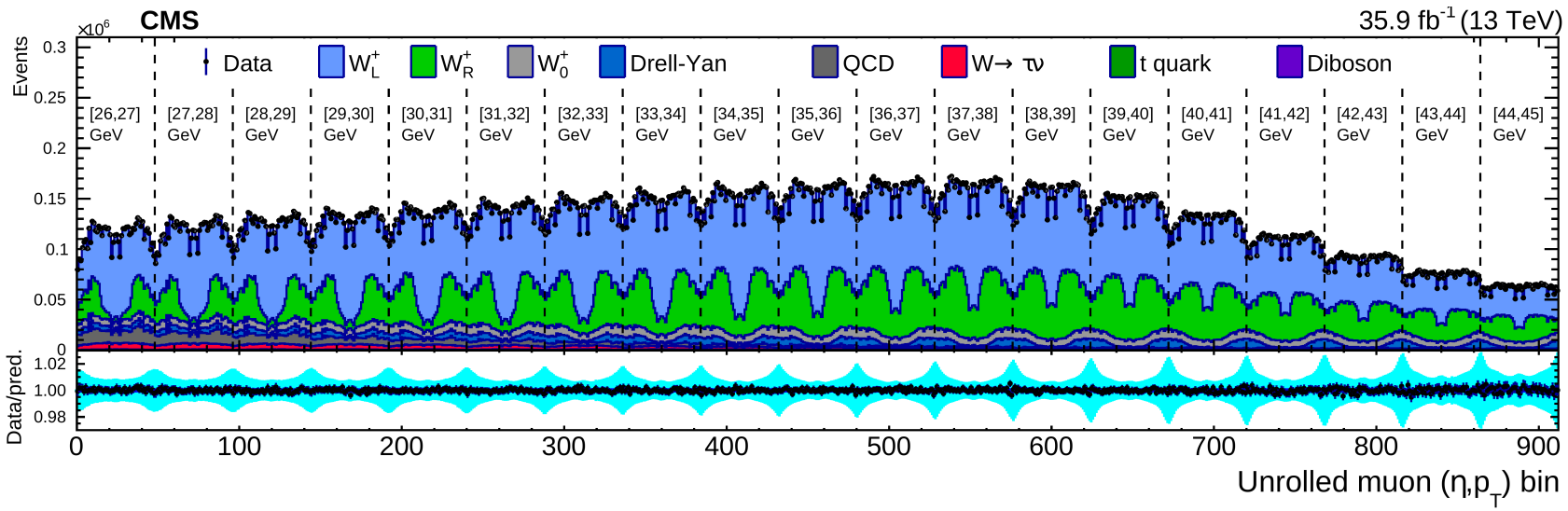
thanks to IT-ST group for help setting up some of these tests

- Need to set e.g. XRD_PARALLELEVTLOOP=16 to get good eos performance
- EOS+xrootd standard production instance not quite scaling up to network limits (possible xrootd client/ROOT bottlenecks?)
- Extremely good performance of EOS test instance, and CephFS (CentOS 8 kernel client), approaching limits of ethernet connection
- **Reach 5.2GBytes/sec from local SSDs, approaching limits of disk array - PCIE 3.0 8x SATA controller)**, to be tested on newer machine/SUBMIT disk server

Muon Calibration Validation

- Validation of muon momentum scale requires thousands of likelihood fits (5000 bins in pt and eta, fit scale and resolution in each one)
- Parallelized Optimized Fitting with Jax (vectorize over multiple fits) $\rightarrow$ full set of validation fits in minutes

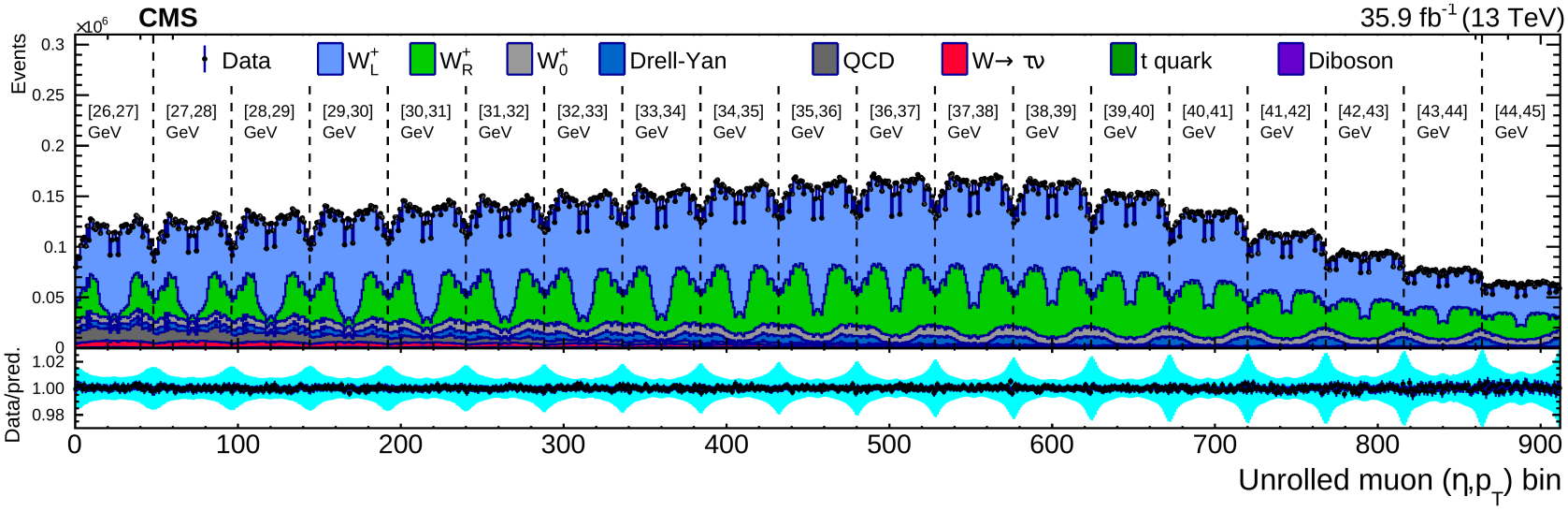
Maximum Likelihood Fit



arXiv:2008.04174

- Maximum likelihood fit for m_W is technically complex
- 1000's of detector-level bins
- O(1000) nuisance parameters
- 10's of thousands of histograms entering the fit
- gradients in minimization need to be known accurately
- good convergence behaviour of fit must be maintained
- time/memory of minimization need to be kept under control
- uncertainties covariances need to be accurately computed

Maximum Likelihood Fit



arXiv:2008.04174

- Maximum likelihood fit for m_W is technically complex
- 1000's of detector-level bins
- O(1000) nuisance parameters
- Optimized maximum likelihood fit using "combinetf" Tensorflow implementation of CMS higgs combination tool, dedicated minimization algorithms
- Full likelihood fit in minutes, full results and uncertainties in $\sim$ 30 mins

Maximum Likelihood Fit

	Likelihood	Likelihood+Gradient	Hessian
Combine, TR1950X 1 Thread	10ms	830ms	-
TF, TR1950X 1 Thread	70ms	430ms	165s
TF, TR1950X 32 Thread	20ms	71ms	32s
TF, 2x Xeon Silver 4110 32 Thread	17ms	54ms	24s
TF, GTX1080	7ms	13ms	10s
TF, V100	4ms	7ms	8s

- Fit can exploit multiple CPU cores or GPU's
- Up to 100x speedup "as the physicist waits"

Conclusions

- Ongoing m_W measurement is extremely challenging from both a physics and technical standpoint
- Innovative and cutting edge technical solutions and requirements at each step of the analysis workflow
- Up to now have focused on large interactive node use at CERN
- Interesting technical challenge/demonstrator to reach similar performance in a **distributed** manner on submit with fast network/disk across several-many nodes

Backup

Analysis Flexibility for "Final" reduction step

- **NANOAOD or similar $\rightarrow$ "final" histograms for plotting/statistical interpretation (or very condensed dataset for unbinned fits)**
- What kind of information can/should we store in O(1kB/event)?
- What kind of logical operations can be (re-)done "on-the-fly" at this stage?

Analysis Flexibility for "Final" reduction step

- What kind of information can/should we store in O(1kB/event)?
- Variable length arrays for 4-vectors and summarized properties of high level objects (electrons/muons/jets/etc)
  - Muon isolation sum ✓
  - Detailed info on isolation constituents ✗
  - Jet substructure variables τ_{21}, DNN tagger output, etc ✓
  - Detailed jet constituent information (clusters, PFCandidates) ✗

Analysis Flexibility for "Final" reduction step

- What kind of logical operations can be (re-)done "on-the-fly" at this stage?
- Object selection (e.g. lepton identification/isolation) ✓
- Composite object combinatorics ($Z \to \mu\mu$, $H \to \gamma\gamma$, $t\bar{t} \to \ell jjbb\not{E}_T$) ✓
- Lepton energy/momentum scale/resolution corrections + **systematic variations** ✓
- Jet energy/resolution corrections + **systematic variations** ✓
- Recomputation of isolation sums ✗
- Reclustering of jets ✗
- Re-calculation of jet substructure variables ✗
- Machine learning inference on low-level detector information ✗
- Machine learning inference on high level object/event quantities ✓

Analysis Flexibility for "Final" reduction step

- **NANOAOD or similar → "final" histograms for plotting/statistical interpretation (or very condensed dataset for unbinned fits)**
- What kind of information can/should we store in O(1kB/event)?
- What kind of logical operations can be (re-)done "on-the-fly" at this stage?
- O(1kB/event) data format can always be reproduced on the grid in days-weeks (for now)
  - Need an alternate calculation of some high level variable to improve sensitivity/mitigate some experimental/theoretical effect
  - "Forgot" an important variable
  - Found a bug
- Some intermediate cases (slow calculations) can be handled with caching results ("friend trees")

Recent Advances in Analysis Software for HEP

- **ROOT**
  - Flexible interface to high performance C++ (jitting + python bindings from Cling/PyROOT)
  - Multi-threaded computational graph (RDataFrame)
- **"Python ecosystem"**
  - e.g. Uproot + Awkward Array to analyze Root trees with numpy-like syntax $\rightarrow$ **Columnar Analysis**
  - Coffea for high level functionality
  - Boost histogram python bindings
  - +++
- Both of the above can be used **today** for high performance analysis, with different caveats/limitations/optimal environments

Recent Advances in Analysis Software for HEP: ROOT Cling/PyROOT

- When I was a grad student using ROOT 5.x, older/wiser people told me "If you care about robustness or performance, compile your (C) ROOT macros"
- Real llvm/clang-based jitting with Cling in Root 6 has eliminated robustness/language feature gap for some time
- PyROOT provides a much friendlier interface, and interoperability with python ecosystem tools
- (Very) Recent developments (**PR#7283**) just about **close the performance gap with compiled code** (one small but important PyROOT case remains **#9112**))

Recent Advances in Analysis Software for HEP: ROOT RDataFrame

- RDF + PyROOT provide a quite user-friendly interface for complex analysis logic with high performance
- Multi-thread parallelization on a single node is $\sim$ fully transparent to the user and extremely convenient if your analysis can fit on the hardware you have access to
- Recent/ongoing work to enable similarly convenient Spark/Dask parallelization (**Poster**)
- Significant ongoing work on functionality/convenience/performance (**Poster**, **PPP talk**)

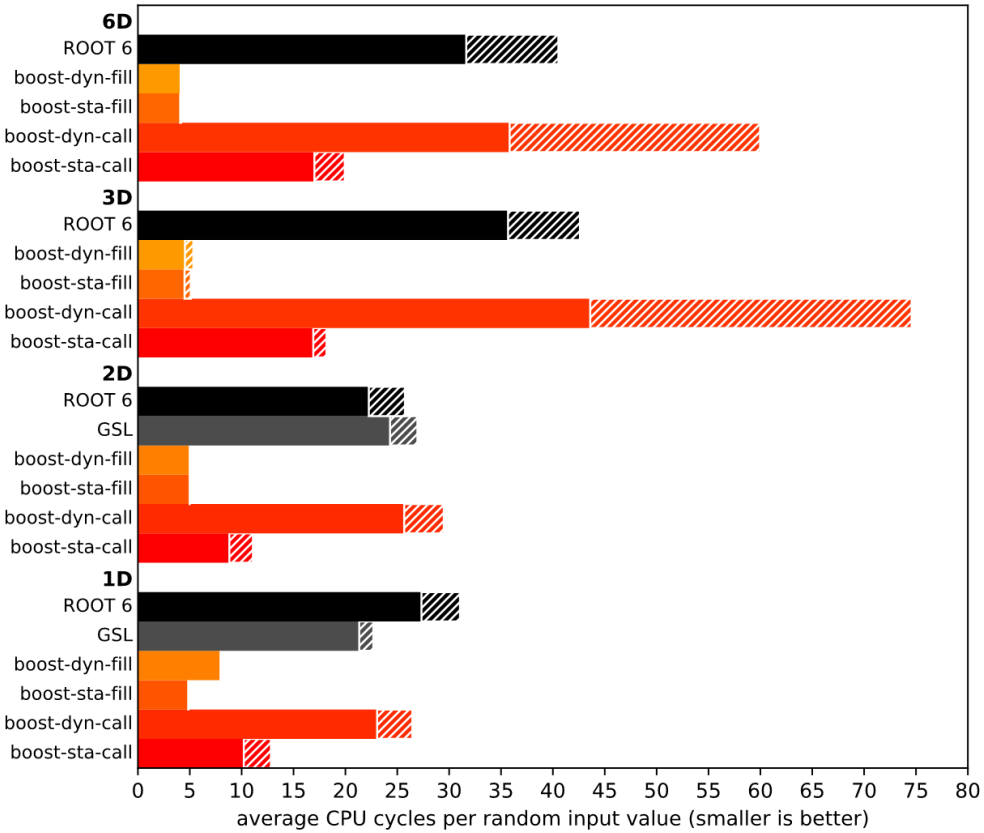
"Python Ecosystem" and Columnar Analysis

- Many industry-standard tools beyond HEP based on python ecosystem (numpy, scipy, etc)
- Variable length arrays e.g. in Root trees are stored as (flattened) contents + offsets
- E.g. three collision events with respectively 2, 3, 1 muons might store for Muon_pt:
  - Contents: [23.1, 10.4, 15.6, 10.8, 45.1, 39.9]
  - Offsets: [0, 2, 5]
- This can be naturally (and technically!) mapped onto Awkward array structures ("jagged" numpy-like arrays where the outer axis indexes events, and the inner axes index objects/properties within each event)
- Numpy-like syntax can be used to perform operations on batches of events
- Uproot: Python re-implementation of (subset of) ROOT I/O
- Coffea: High level analysis tools around the above
- **Multi-process** parallelization on a single node, significant focus on efficient scaling with Spark/Dask type infrastructure
- e.g. relevant talks in parallel sessions (**talk**, **talk**)

Event Batches vs Event Loop

- Operating on large batches of events can render "call overhead" irrelevant, much more easily exposes vectorization (**really** slow stuff in the python interpreter can be fine if it's called once per 100k events)
- "Per event" logic is much more in line with how HEP analysis has been done up to this point
- **C++ is not immune from call overheads**
  - vtable lookups, clobbering the branch predictors
  - dynamic memory allocation with std::vector-like objects
- With aggressive inlining, the **event loop can also expose vectorization potential**
  - RDF already avoids copying array contents per event (RVec as a view)
  - Work ongoing exploring "full inlining" of the graph, using optimized jitting to "undo" type erasure where necessary, with some interesting features to be improved/worked around (**pathological example**)

Event Batches vs Event Loop



https://www.boost.org/doc/libs/1_77_0/libs/histogram/doc/html/histogram/benchmarks.html

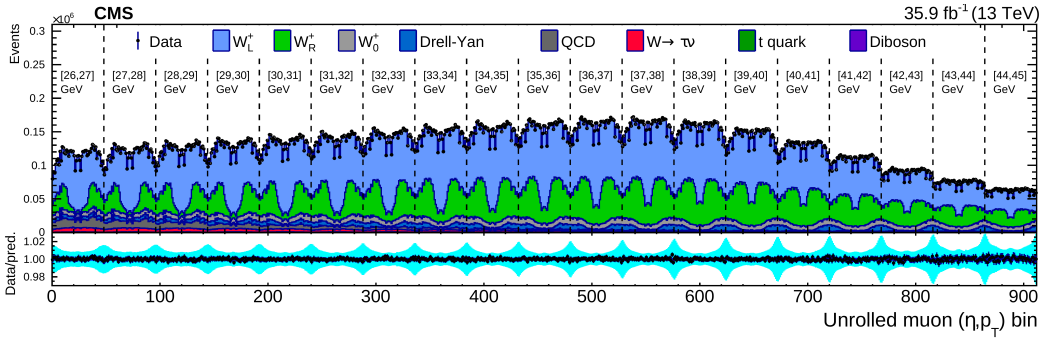
- Boost histograms (in C++) provide an interesting example where static/inlining optimizations matter for the per-event case, but largely irrelevant for large batches

Multi-threading vs Multi-processing

- One important distinction which is often overlooked: **shared memory**
- A typical server/batch slot in LHC computing infrastructure provides 2GB/thread of memory
- If the aggregated data (e.g. #of histogram bins x 16 bytes) exceeds 2GB, then *you are having a bad problem and you will not go to space today*
- Simplest way to mitigate this: thread-safe shared memory histograms with std::atomic internal storage **already possible with Boost histograms in C++** (and this can be used with RDF as well)
- Other possibilities: sparse histograms, buffered/locked filling, distributed histograms
- **Python tools can multi-thread too with care over the GIL**

  (python-jitting tools like Numba, Jax can aid in this)

A practical example



arXiv:2008.04174, (related example for illustration)

- Example with 800M data events, 1.4B W/Z signal, 1B background (4TB total), O(1000) systematic variations
- Filling $\sim$ 70x 5D histograms with RDF (using **in-progress HistoND functionality**)
- (p_T, η, charge, signal/control region idx, systematic idx) x (systematic groups, processes)
- 45 minutes to final histograms on 32 core/64 thread machine with SATA SSD's (still more potential for single node scale-up)
- **Already hit the per-thread memory limit with all processes simultaneously** (evaluating shared memory options)
- (n.b. auxiliary calibration workflows on the same machine exceed 5Gbytes/sec in real life use cases, limited by SATA controller PCIE link)
- See also **CERN EP Software Seminar**

Some thoughts on interoperability

- Having multiple sets of tools with parallel developments and innovations is a Good Thing
- Interoperability is also a Good Thing
- Related and rather frustrating examples (related to use of cppyy vs pybind11 for C++ python bindings):
  - Mutually incompatible python representations of TH1 in uproot and PyROOT
  - Not straightforward to use boost histogram python bindings with RDF
- More pragmatic approach to dependencies might be beneficial in some cases (do we really want/need to (always) read from Root files with zero Root dependencies? tighter integration could speed the path to real multi-threading in python ecosystem)

Conclusions

- HL-LHC will pose unprecedented challenges including for physics analysis
- **Fast** and **Flexible** analysis software/infrastructure/workflows are needed to get the best physics out of the data
- Significant development over the last few years in ROOT and in python ecosystem tools provide a serious head-start to addressing this **and are already essential for some classes of measurements**
- Some challenges to address leveraging the best use of thread-level and multi-node parallelism together, and optimizing storage and IO patterns (mass storage + client optimization/new mass storage systems/local or intermediate caching layers/HDDs vs SSDs)
- Core functionality/performance/ease of use improvements will no doubt continue on all fronts
- (n.b. visualization, fitting, statistical interpretation also important, much progress towards leveraging modern python tools/autograd for this: **PyHEP talk**, **PyHF**, **ZFit**/**poster**)