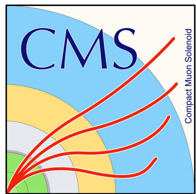


Analysis Benchmarks

Where are the bottlenecks?

Luke Kreczko (CMS/LZ)

24th March 2020

University of BRISTOL

First ECHEP workshop

During first workshop we had a lot of emphasis on processing speed

- Some known inefficiencies in generators (e.g. negative weights, lexical_cast)
- Benefits of SIMD, trading accuracy for speed in simulation (fast-sim)
- Alternative architectures (GPU, FPGA, etc) to accelerate software in HEP

Improvements usually identified through profiling & benchmarks

In some cases a lot of expert knowledge went into improvements

ECHEP Analysis summary slides (Ben Krikler)

One possible 6 month plan

- Step 1: Identify several benchmark analyses (1 month)
  - Open data, coordinate between experiments
- Step 2: Implement using existing tools (2.5 months)
  - Coffea or FAST-HEP
  - Directly with PARSL or DASK where data not amenable to columnar approach
  - **Deliverable: where do these existing tools struggle for different analyses / experiments**
- Step 3: Understand caching requirements (2.5 months)
  - Timing without caching
  - **Deliverable: Profiling for second run with caching**
  - **Deliverable: Size of caching required**
- Step 4 (extension): Explore caching between analyzers (?)

Python for accelerated development

(the near future is Python)

- High level programming → fast to try something
- Compact language → fewer lines of code, fewer bugs
- Quick refactoring
- Access to Big data tools → ML + distributed processing
- Full-Stack prototyping (even FPGA)
- Can be easy to change architectures (e.g. CPU → GPU with numba/tensorflow)

Numba: Python just-in-time compiler

- Few 'array-oriented' compilers though common use case and hardware optimizations exist.
- Wasn't possible few years ago, **Python faster than your C++ code.**

```
@vectorize
def sinc(x):
    if x==0.0:
        return 1.0
    else:
        return sin(x*pi)/(pi*x)
```

```
1  ; ModuleID = 'sinc_mod_7b29370'
2
3  define double @sinc(
4  Entry:
5    %0 = fcmp oeq doub
6    br i1 %0, label %B
7
8  BLOCK_12:
9    ret double 1.00000
10
11 BLOCK_16:                                         ; preds = %Entry
12   %1 = fmul double %x, 0x400921FB54442D18
13   %2 = call double @llvm.sin.f64(double %1)
14   %3 = fmul double %x, 0x400921FB54442D18
15   %4 = fdiv double %2, %3
16   ret double %4
17
18 BLOCK_47:                                         ; No predecessors!
19   ret double 0.000000e+00
20 }
21
22 declare double @llvm.sin.f64(double) nounwind readonly
```

Faster than C++?
... depends

- ROOT cling game changer here for JIT devs!!!!

https://zenodo.org/record/1418513#.XniKnoj7TAQ

Step 1: Identify several benchmark analyses

Start with some small subtasks for guidance

HEP Data analysis

Compared to dedicated efforts in generators and simulation, data analysis is more varied

- Analysis methods & algorithms can differ widely between groups
- Code is usually written by non-experts
- Mixture of using experiment software frameworks and standalone code

Difficult to utilize advanced techniques or new architectures

Analysis benchmarks

First two benchmarks implemented - meant to highlight big differences between individual analysis steps

- Selecting events with at least 4 jets with pt > 30 GeV and |eta| < 2.4 (e.g. a loose skim)
    - 0 to N jets per event → loops of depth 2
    - Uses CMS Top Quark open data (6,423,106 events)
- Categorise ttbar decay channel
    - 10 different decay channels (from full hadronic to full leptonic)
    - Need to traverse genparticle decay chain → loops of depth 3 & many comparisons
    - Uses private MC due to missing information in Open Data (1,441,999 events)

Processed on CERN GitLab CI, data are copied to local disk

Selection results (6,423,106 events)

method	HDD (CERN CI)	SSD	comment
numpy	18.5 s	8.4 s	Advanced python
Loop depth 1	53.2 s	30.5 s	Advanced python
Loop depth 2	214.5 s	128.2 s	Beginner python
C++ loop depth 2	6.6 s	4.4 s	Advanced ROOT

Advanced ROOT: using ROOT features that are not that common (yet) but recommended (e.g. TTreeReaderArray) - ROOT usage is stuck “in the past” due to inheritance of macros

(fresh) Decay channel results (1,441,999 events)

method	HDD (CERN CI)	SSD	**comment**
numpy	37.1 s	12.5 s	Advanced python
Loop depth 3	1436.8 s	822.4 s	Beginner python
C++ loop depth 3	8.9 s	4.3 s (18.5 s)	Advanced ROOT
C++ GetEntry	308.9 s	148.3 s	Beginner ROOT
C++ GetEntry + disabling unused branches	---	15.7 s (51.2s)	Beginner ROOT

Note 1: Arrays stored in NanoAOD do not work well with SetBranch method (output is wrong for some events))
Note 2: Using namespaces in the ROOT macro, increases processing time (???)

(fresh) Decay channel results (1,441,999 events)

method	HDD (CERN CI)	SSD	
numpy	37.1 s		
			Beginner ROOT

Depending on task & implementation performance can range across orders of magnitude

Note 1: Arrays stored in NanoAOD do not work well with SetBranch method (output is wrong for some events))
Note 2: Using namespaces in the ROOT macro, increases processing time (???)

Step 2: Implement using existing tools

Moving from implementation to description

FAST-HEP tools offer another approach

- Describe what you need, tools will do the rest
- Optimizations can happen in the background
  - This includes architecture selection & parallel processing (local & distributed)
- What is the fraction of covered use-cases?
  - We know it works for some CMS, LZ and DUNE analyses
    - Including a range of distributed backends (e.g. from Coffea project)
  - Performance issues with nested types (e.g. vector<vector<vector>>)
    - Awkward-array 1.0 will fix these
  - Multi-tree input (e.g. CMS L1 trigger prototyping, standard LZ analyses) not yet functional

CMS Efforts overview

CMS Analysis Facility/real-time data query system

Steps 3 & 4 have certainly overlap

- Analysis with Apache Spark
  - Caching capabilities
- Real-time data query system
  - Explores fast data query and caching
  - Shows big difference depending on how the data are accessed
  - code transformation performance should be similar to TTreeReaderArray
- Now all under ServiceX
  - Use Kafka for streaming
  - Cache results for instant replay

https://arxiv.org/abs/1711.01229

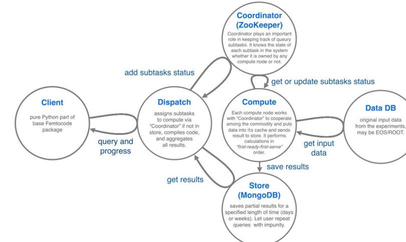
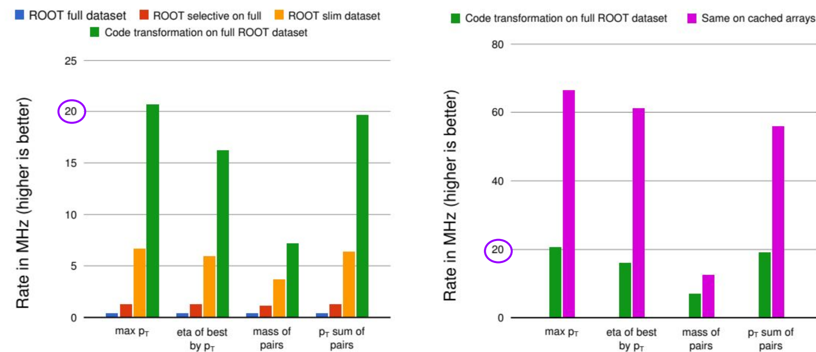


Figure 2. Schematic for distributed query processing to minimize cache misses (see text).



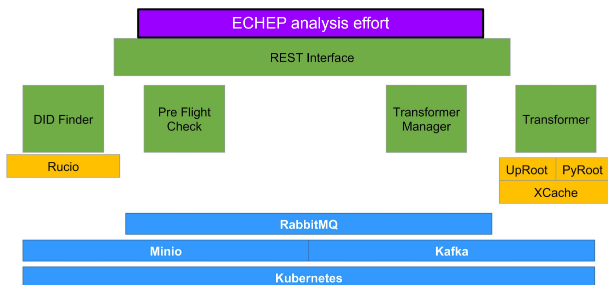
New CMS effort

Software and Computing R&D and code modernization/performance

- Just started: First meeting on the 12th of March
- Many areas targeted
  - Not sure how much of it is public knowledge (or allowed to be)
- But, due to the nature of the effort:
  - Significant overlap with ECHEP

Summary

Questions and Outlook



First benchmarks implemented

- What is the best analysis to highlight current bottleneck?
- Are these the same between ATLAS & CMS?
- Do we have something similar for LHCb (and other experiments)?

Next: implement with existing (high-level) tools

- FAST-HEP + Coffea (Parsl backend)
- Current example highlight the difficulty of replacing for-loops

Other efforts exist

- Mostly going in a similar direction
  - Minimize overlap, maximize syngery
- User → ServiceX for steps 3+4 might be an option

Backup slides

Other thoughts

(as a by-product of step 2)

Expertise vs automation

Other than being an expert in C++, Python & <distributed computing system of your experiment, department, funding agency's choice> what can be done?

Automation:

- E.g. CERN CI has access to EOS
  - can a mini-version of the analysis be run in CI (tests & profiling)?
  - Do we have to provide skeleton analysis or training for this?
- Is there a way to test this in the submission systems?
  - CI → submission system for scaling?

TODO

Do we have to worry about producing plots?

- It can be time-consuming, but usually is not

Statistical analysis (RooFit, PyHF, Minuit) can take a lot of time

- Which tool is the best (fast & reliable)?

Putting everything into ECHEP context:

Analysis is fraction X of total computing budget, following route Y would reduce usage by Z. → does this translate into analysis improvements, more resources for other analyses or just saving money?

Other items

HSF simulation on non-LHC simulation requirements:
https://indico.cern.ch/event/899153/

Draft roadmap

Attempt to measure analysis bottlenecks

- Focus on specific algorithms within different analyses
    - Compare SIMD/GPU to typical implementations
- Provide training material for advanced algorithms
    - Translating an analysis into SIMD or different architectures is not trivial
- Advertise tools (e.g. FAST-HEP, Coffea) that can improve transitions for non-experts

Focus on Python (compare to C++) as it is a friendlier language