CMS W-mass Performance Analysis

G. Amadio

ROOT

Data Analysis Framework

https://root.cern

Overview

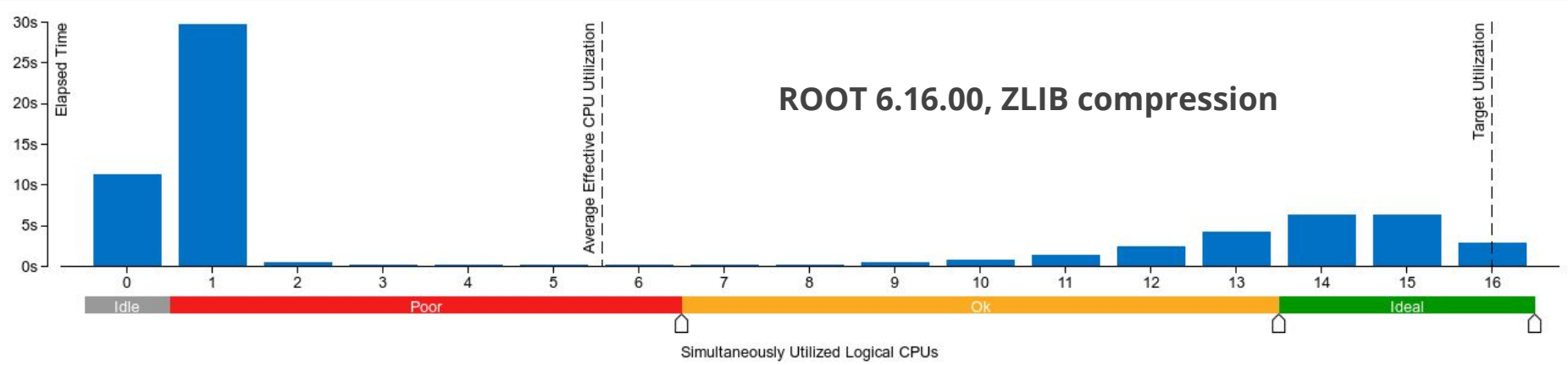
- CMS W-mass analysis from Elisabetta Manca
- Python 2.7 script using RDataFrame via PyROOT
- Run on titanx.cern.ch machine (Core i7 7820X @ 3.6GHz)
- Run with ROOT 6.16 and master branches
- ROOT compiled with -march=native -O2 -g
- Input files: ~9G (LZMA), ~12G (ZLIB)
- VTune 2018, perf (visualization with gprof2dot + flamegraph)
- No valgrind (does not work with AVX512)

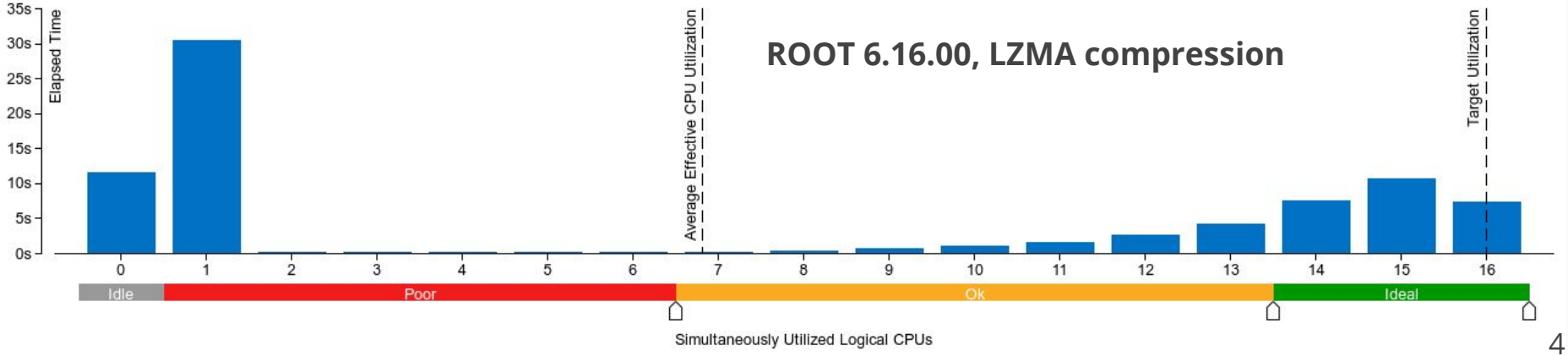
VTune Amplifier XE: Summary

Runtime	6.16.00	master@dc11db1b
ZLIB	Elapsed Time: 67.560s CPU Time: 435.480s Effective Time: 376.365s Spin Time: 1.970s Overhead Time: 57.145s Total Thread Count: 62 Paused Time: 0s	Elapsed Time: 143.813s CPU Time: 429.370s Effective Time: 367.029s Spin Time: 2.190s Overhead Time: 60.151s Total Thread Count: 59 Paused Time: 0s
LZMA	Elapsed Time: 79.754s CPU Time: 621.640s Effective Time: 543.836s Spin Time: 1.770s Overhead Time: 76.034s Total Thread Count: 62 Paused Time: 0s	Elapsed Time: 203.433s CPU Time: 626.260s Effective Time: 538.291s Spin Time: 5.420s Overhead Time: 82.549s Total Thread Count: 59 Paused Time: 0s

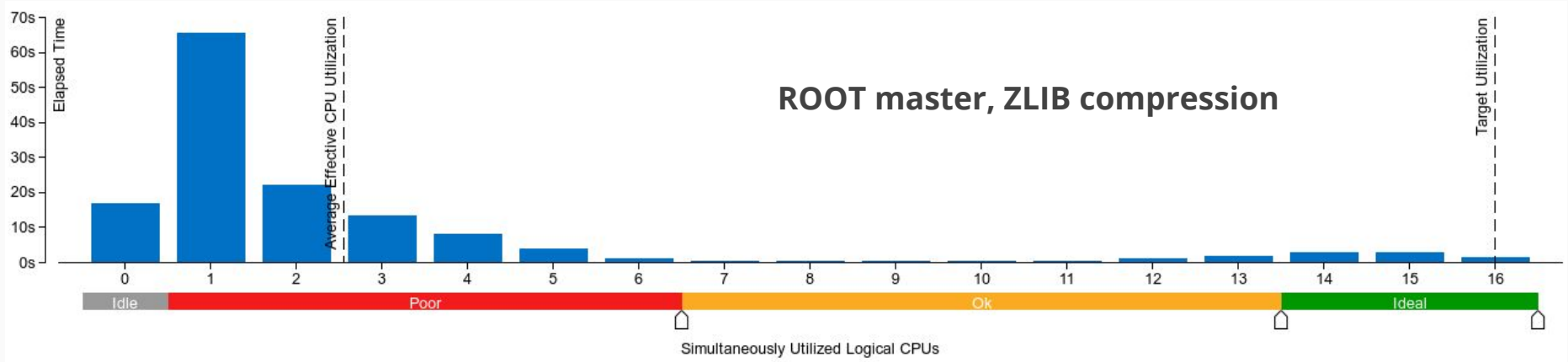
note: outside of VTune, the runtime is not that different between 6.16.00 and master

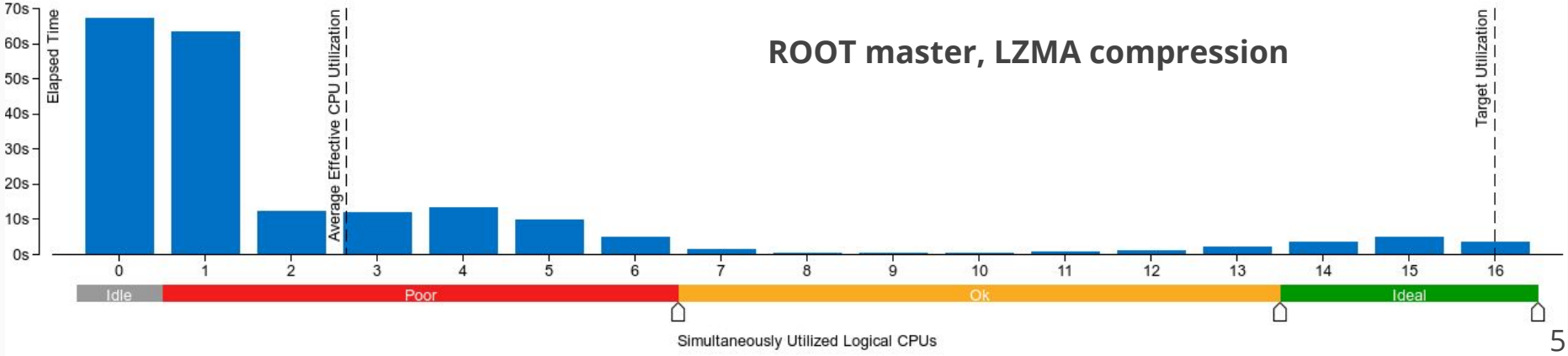
VTune Amplifier XE: CPU Utilization





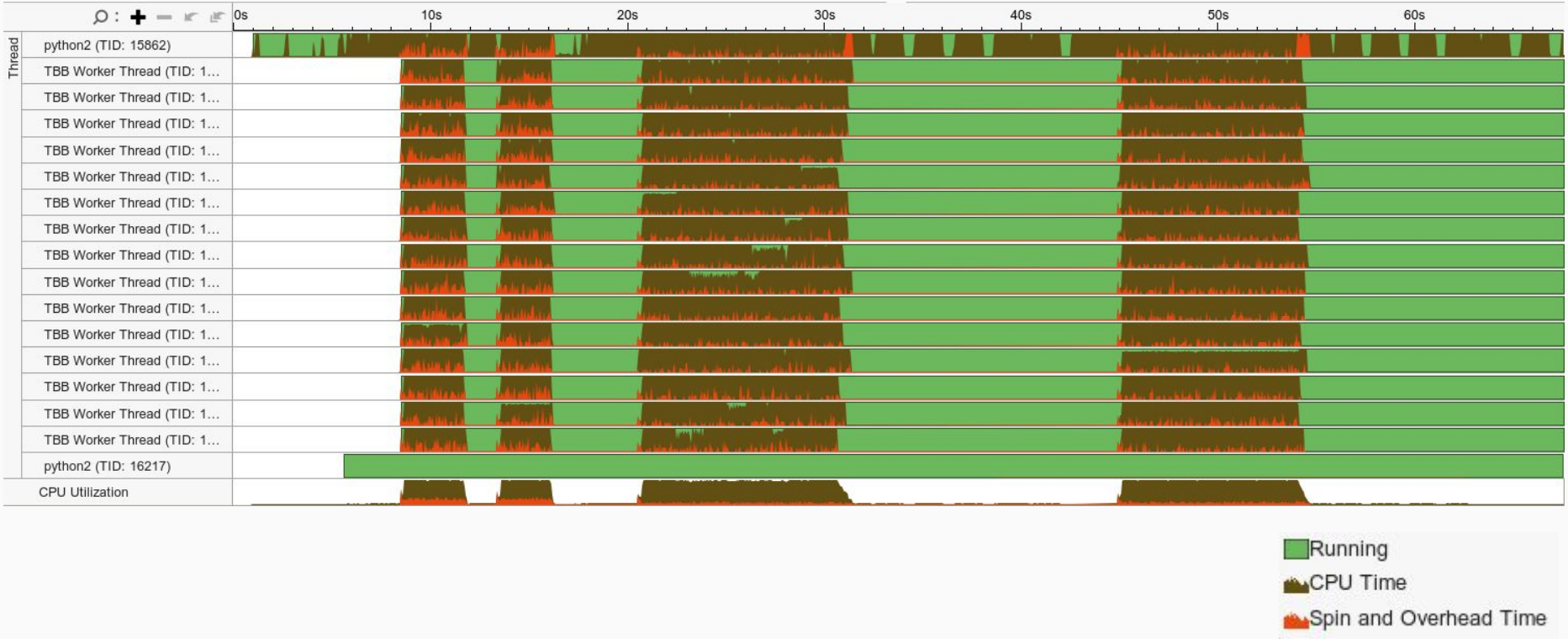
VTune Amplifier XE: CPU Utilization





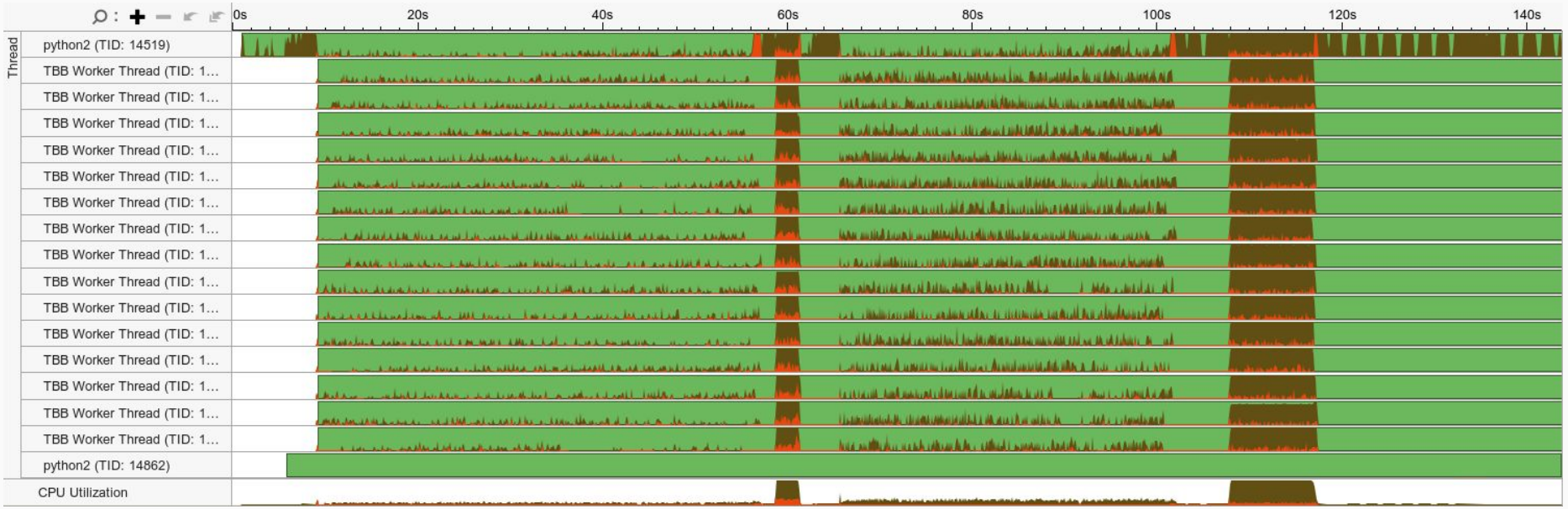
VTune Amplifier XE: Platform Overview

ROOT 6.16.00, ZLIB compression



VTune Amplifier XE: Platform Overview

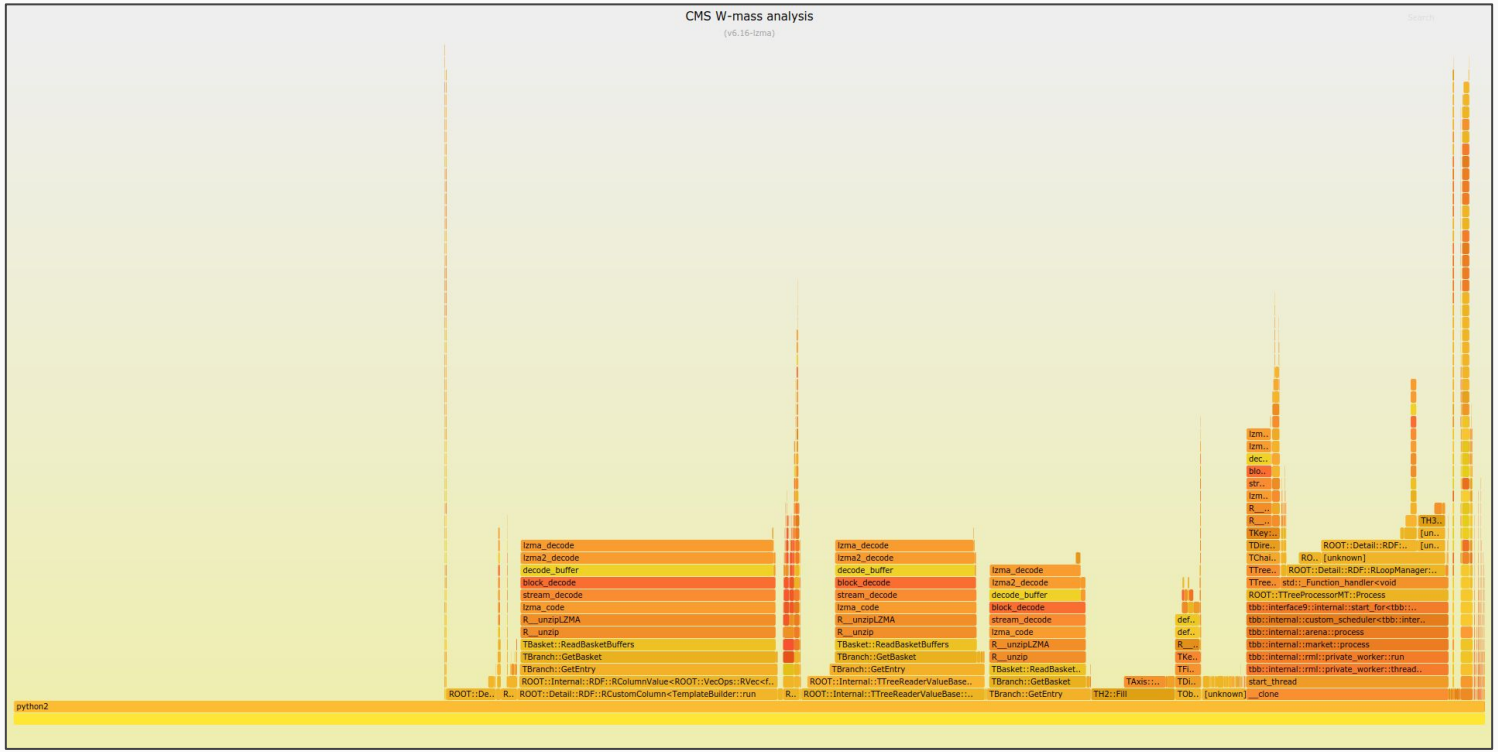
ROOT master, ZLIB compression



Module / Function / Call Stack ▲	CPU Time: Difference	CPU Time: hs-master-lzma			CPU Time: new-6.16-lzma		
		Effective Time by Utilization (Idle Poor Ok Ideal Over)	Spin Time	Overhead Time	Effective Time by Utilization (Idle Poor Ok Ideal Over)	Spin Time	Overhead Time
[Unknown]	-30.443s	131.355s	0.010s	0.040s	161.757s	0s	0.090s
AngCoeff.so	-1.619s	8.355s	0s	0s	9.974s	0s	0s
GetWeights.so	-2.898s	10.068s	0s	0s	12.966s	0s	0s
ld-linux-x86-64.so.2	-0.010s	0.090s	0s	0s	0.100s	0s	0s
libc-dynamic.so	0.010s	0.276s	0s	0s	0.266s	0s	0s
libc.so.6	-0.030s	0.140s	0.010s	0s	0.180s	0s	0s
libCling.so	1.402s	1.402s	0s	0s			
libCling.so	-1.150s				1.150s	0s	0s
libCore.so.6.16	-0.220s				0.220s	0s	0s
libCore.so.6.17	0.100s	0.100s	0s	0s			
libdl.so.2	0.007s	0.381s	0s	0s	0.374s	0s	0s
libHist.so.6.16	-16.970s				16.970s	0s	0s
libHist.so.6.17	14.428s	14.428s	0s	0s			
libImt.so.6.16	-40.253s				0s	0.020s	40.233s
libImt.so.6.17	44.933s	0s	0s	44.933s			
libLLVMBitReader.so.5	-0.010s				0.010s	0s	0s
libLLVMCore.so.5	0.020s	0.030s	0s	0s	0.010s	0s	0s
libLLVMRuntimeDyld.so.5	-0.050s	0.270s	0s	0s	0.320s	0s	0s
libLLVMSelectionDAG.so.5	0.060s	0.100s	0s	0s	0.040s	0s	0s
libLLVMSupport.so.5	0.070s	0.070s	0s	0s			
libLLVMX86CodeGen.so.5	0.020s	0.060s	0s	0s	0.040s	0s	0s
libLLVMX86Desc.so.5	-0.010s				0.010s	0s	0s
liblzma.so.5	0.030s	0.100s	0s	0s	0.070s	0s	0s
libm.so.6	0.099s	3.080s	0s	0s	2.981s	0s	0s
libpin3dwarf.so	-0.036s	0.064s	0s	0s	0.100s	0s	0s
libpthread.so.0	0.117s	0.297s	0.070s	0s	0.210s	0.040s	0s
libPyROOT.so	0.100s	0.100s	0s	0s			
libPyROOT.so	-0.130s				0.130s	0s	0s
libpython2.7.so.1.0	-0.091s	22.920s	0s	0s	23.011s	0s	0s
libROOTDataFrame.so.6.16	-7.006s				7.006s	0s	0s
libROOTDataFrame.so.6.17	7.119s	7.119s	0s	0s			
libROOTVecOps.so.6.16.00	-0.430s				0.430s	0s	0s
libROOTVecOps.so.6.17	0.260s	0.260s	0s	0s			
libstdc++.so.6	-0.010s	0.020s	0s	0s	0.030s	0s	0s
libtbb.so.2	6.584s	0.120s	5.330s	37.576s	0.030s	1.600s	34.812s
libThread.so.6.17	0.010s	0.010s	0s	0s			
libTree.so.6.16	-0.790s				0.790s	0s	0s
libTree.so.6.17	1.550s	1.550s	0s	0s			
libTreePlayer.so.6.16	-241.738s				241.738s	0s	0s
libTreePlayer.so.6.17	292.339s	292.339s	0s	0s			
TemplateBuilder.so	-22.784s	43.189s	0s	0s	65.973s	0s	0s

Perf data visualizations

Data available at https://cern.ch/amadio/wmass



Conclusions and Summary

- No clear bottleneck, but there is scheduling overhead
- zlib vs lzma: 15% faster, at the cost of 30% larger files
- Something seems strange from 6.16 to master
- Not all VTune analysis types work, had to use a basic one
- Next step is to make changes to master and re-measure
- Would be nice to have JIT with debugging symbols
- Also ran with python2 -m cProfile, but no useful info