Performance Optimization

Justin Anguiano

August 1, 2019

Goal — Improve Physics analysis performance

Use different C/Python approaches to obtain a low programming overhead but high performance

Task to be optimized: Read File(s) containing a TTree, produce histograms from elements of the tree(s), write histograms to a TFile

Three histograms are produced: TH1D: track p_T weighted by $1/p_T$, TH1D: track p_z, TH2D, track p_x vs p_y

First attempt using ROOT6 parallelization on local machine (laptop) methods:

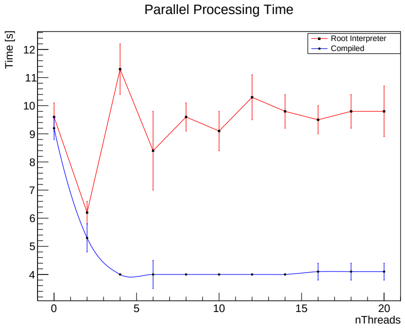
- Sequential run over a test file with TTreeReader (Compiled and Interpreted)
- Parallelized run over a test file with TTreeReader (Compiled and Interpreted)

Test File Details

- code adapted from example: `https://root.cern.ch/doc/v612/imt101__parTreeProcessing_8C.html`
- uses fake event file containing 48000 events; each with some number of tracks per event
- total tracks overall $\approx 2.4e+7$

First Test Results:

- Time is Mean time $\pm$ stdev over 10 trials
- no guarantee of system releasing requested resources (nthreads) so perform multiple trials
- data point at nThreads $= 0$ is the basic sequential program
- high threadcount performance possibly bottlenecked by system?
- Takeaway: Compiling is important! $\approx$Factor of 2 improvement



Second attempt using ROOT6 parallelization, classic(Make Class) parallelization, MultiProcessing on t3.unl.edu
methods:

- Sequential run over multiple test files with TTreeReader (Compiled and Interpreted)
- Parallelized run over multiple test files with TTreeReader (Compiled and Interpreted)
- Sequential run over multiple test files with Make Class (Compiled and Interpreted)
- Parallelized run over multiple test files with Make Class (Compile and Interpreted)

Test File set Details:

- using 10 files from Single Muon 2018 dataset
- X events each with at least 1 conversion per event
- total tracks overall