Performance Optimization

Justin Anguiano

August 8, 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:

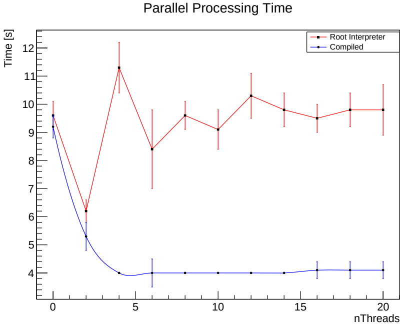
- Sequential run over a test file with TTreeReader (Compiled and Interpreted)
- Parallelized run over a test file with TTreeReader (Compiled and Interpreted)

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 versus classic(Make Class) sequential program (my current approach) on t3.unl.edu
Methods:

- Sequential run over multiple test files with TTreeReader (Compiled Only)
- Parallelized run over multiple test files with TTreeReader (Compiled Only)
- Sequential run over multiple test files with MakeClass (Compiled and Interpreted)

Test File set Details:

- using 9 files from Dataset: `/SingleMuon/Run2018D-PromptReco-v2/AOD`
  - Physics group: NoGroup Creation time: 2018-08-01 13:16:41 Status: VALID Type: data Dataset size: 150070502441346 (150.1TB) Number of blocks: 267 Number of events: 511823047 Number of files: 45330
- 1576737 events each with at least 1 conversion per event
- total tracks overall (exactly 2 per conversion) = 14190956

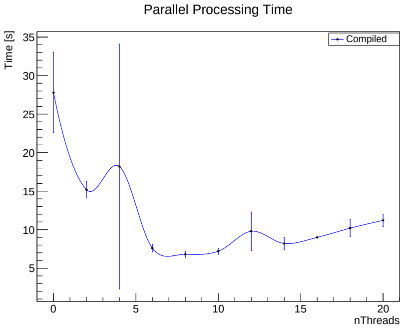
File Sequence Details:

- Run2018_100.root
  - contains 193388 events
  - contains 1547322 unique conversion tracks
- Run2018_110.root
  - contains 140502 events
  - contains 950444 unique conversion tracks
- Run2018_120.root
  - contains 99085 events
  - contains 602330
- Run2018_130.root
  - contains 62289 events
  - contains 347156 unique conversion tracks
- Run2018_141.root
  - contains 218339 events
  - contains 1958272 unique conversion tracks
- Run2018_155.root
  - contains 269483 events
  - contains 2968340 unique conversion tracks
- Run2018_166.root
  - contains 172409 events
  - contains 1299172 unique conversion tracks
- Run2018_176.root
  - contains 127117 events
  - contains 832024 unique conversion tracks
- Run2018_193.root
  - contains 294125 events
  - contains 3685896 unique conversion tracks

First Cluster Test Results:

(Using full 9 file dataset)

- Time is Mean time $\pm$ stdev over 5 trials
- data point at nThreads $=0$ is the basic sequential program
- high threadcount again not optimal, chokes up the system
- Classic sequential approaches using MakeClass (Compiled and Interpreted)–
  - 265.4 ± 20.6 [s] (Interpreted)
  - 171.0 ± 23.7 [s] (Compiled)
- ROOT6 mutlithreading is very good!



Second Cluster Test: The impact of data size on performance

- Time is Mean time $\pm$ stdev over 5 trials
- nTracks is the accumulated number of tracks looped over for a given number of files that are run in a preserved order
- Parallel is run with the previously optimal number of threads (8)
- Parrellization is faster and more consistent at any size of dataset
- As data gets larger the parallel benefit becomes more significant