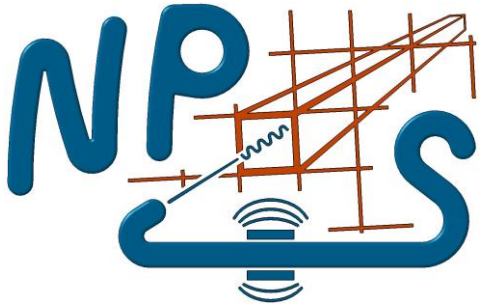
Multi-Threaded Waveform Fitting

MITCH KERVER

5/5/2025



Multi-Threading the Waveform Analysis

- Runs are broken up into multiple segments with up to ~500k events per segment.
- Original waveform fitting code estimated to take up to 25days for the longest segments!!
- Max wall time on the farm CPUs ~2 day -> jobs will fail

- Solutions:
  - Further subdivide each segment, run wf analysis, then recombine.
  - Run wf analysis across multiple CPU threads.
    - Give each job $\#\text{of cores} \approx \frac{Days\ expected}{2\ Day\ walltime}$

kinC_x25_4	4986	20 uA	LD2	PS6=0	260854	2.73	1360516	12.28	5.2156	0 days, 4 hours	8 days, 6 hours
kinC_x25_1	5919	20 uA	LD2	PS6=0	111547	6.61	3687865	11.28	33.0611	0 days, 11 hours	8 days, 12 hours
kinC_x50_0	5437	20 uA	LD2	PS6=0	295105	2.57	1374502	11.54	4.6577	0 days, 4 hours	8 days, 18 hours
kinC_x36_6	4714	20 uA	LD2	PS6=0	136582	7.31	2773186	16.47	20.3042	0 days, 12 hours	11 days, 13 hours
kinC_x50_0a	5249	20 uA	LD2	PS6=0	130320	9.55	2923283	20.12	22.4316	0 days, 16 hours	14 days, 9 hours

Original Code Structure – Wassim/Malek

Read in elastic reference waveforms and interpolate.
Loop over all events{
Loop over all blocks{

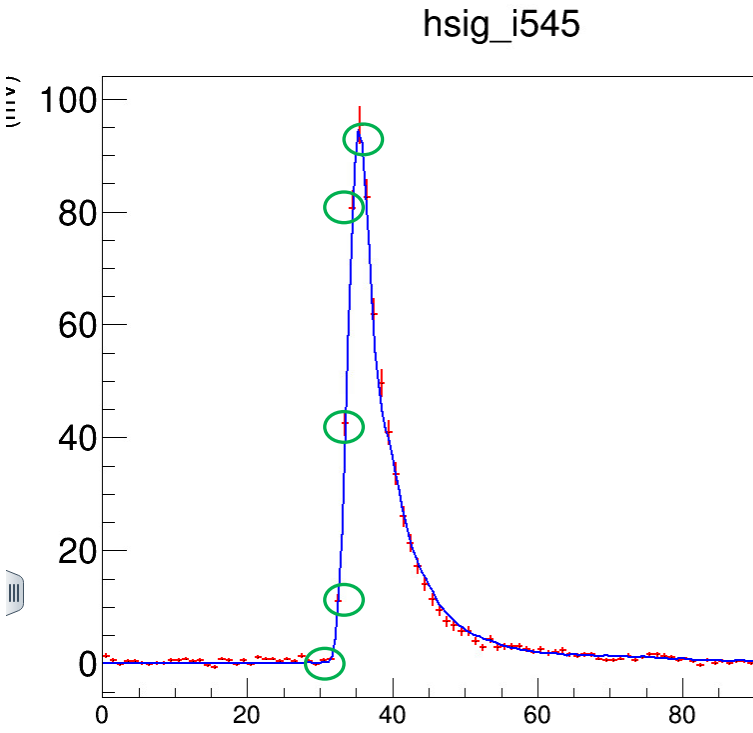
- Fill histogram with 110-sample ADC waveform.
- Scan for 5 **consecutively increasing** samples + 2 decreasing = pulse found.
- If pulse found use the peak amplitude and time as initial fit parameters.

- If no pulse found or pulses are outside coincidence window -> add new "pulse" with fit parameters =2mV at the expected coincidence time.

- Create a fit function: $f(t) = A \cdot R_{\text{interp}}(t - t_0)$
  $R_{\text{interp}}(t)$ = interpolated reference shape

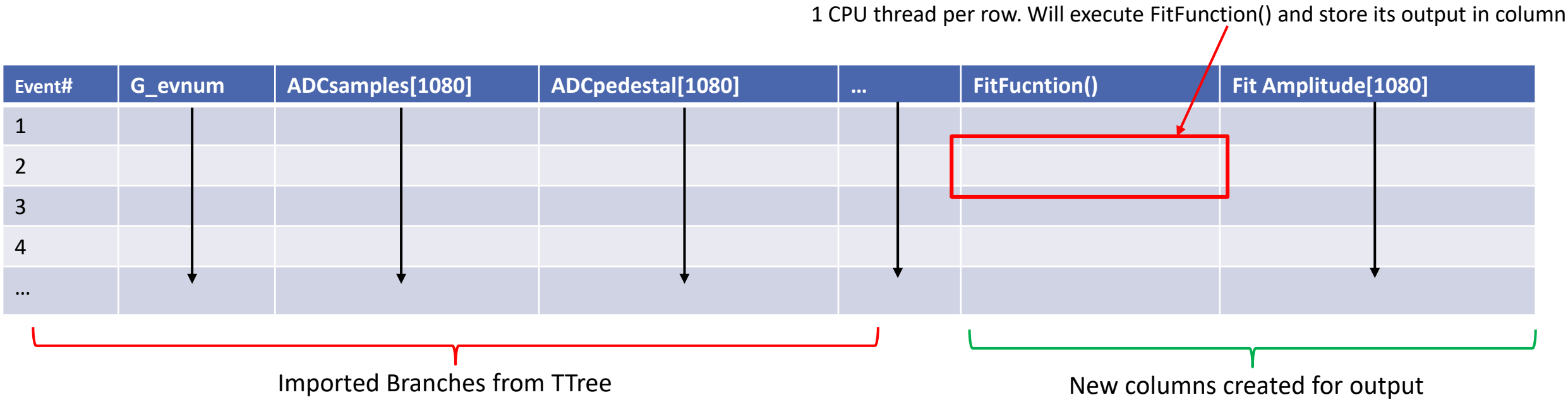
- Perform χ^2 minimization between fit function and histogram.
- Save the final amplitude and time fit parameters in ROOT branches

}
}



RDataFrames

- ROOT's multi-thread friendly interface for data analysis built around a column/row structure.
- Rows represent the event number, and Columns represent a TTree branch, variable, or even a function!
- Fill the first columns with branches from the input ROOT file
- Make new columns containing logic/variables needed for WF fitting.
- No Looping over events anymore. ROOT handles filling all columns for each row on different threads.
- When all entries are filled…snapshot the Rdataframe into a normal Root Ttree for output!



First Approach

- Create a Rdataframe with a new column that is a function which contains ALL the logic within the original event loop (slide3)
- Whenever a thread evaluates a row of the Rdataframe it will call the function with a **unique** set of all its local variables (histograms, interpolator, fit function, etc)
- The output of that function is stored in that column for the corresponding row

Pros:

- Maintain exact same logic and data structures as original code
- Same fitting routine -> Outputs should be identical (with very small numerical differences)
- Avoid having to deal with Mutex and threads sharing variables

Cons:

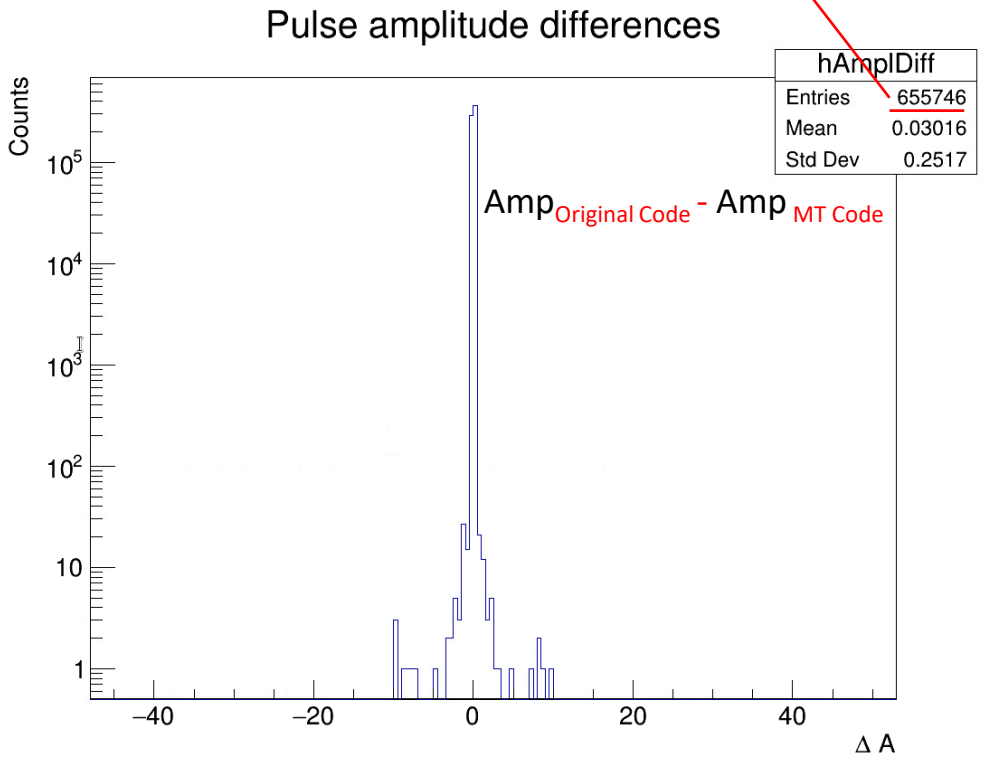
- Rdataframes doesn't support selecting subset of events with Multi-threading enabled;
  Either must run on entire ROOT file, or if just over a subset of events limit to only 1 thread. ☹

Initial Comparisons

- Compared to runs done by Wassim of just first 5k events (must run on 1 thread)
  - Fit amplitudes match very well!
  - χ^2 distributions match

- Compared a few small full segments on multi-threads
  - Fit amplitudes still agreeing!
  - Often seg faulted with higher core counts > 3 …hmm

$5k\ events\ \times\ (7 \times 7 blocks)\ \times\ \sim 2\ pulses \times \#clusters$



First Issue: Minuit

- Each thread has its own unique TF1::Fit() object. TF1::Fit() uses the Minuit minimizer.
- Turns out when the Fit() is called, the minimizer uses some shared global variables in the backend.

- Running the code with a low thread count (2) and small number of events less likely for threads to call Minuit simultaneously -> no errors
- Explains why crashed with higher thread count jobs!

- Solution:
  - Use Fit::Fitter() objects instead, that can utilize the newer Minuit2 minimizer, which is thread safe!

- Working with Fit::Fitter() has brought several complications:
  - The underlying fit method is no longer the same as original code
  - Fits often fail if waveform is noisy or flat
  - Very sensitive to initial parameters
  - Doesn’t respect set parameter limits

  - See how these manifest with some examples…

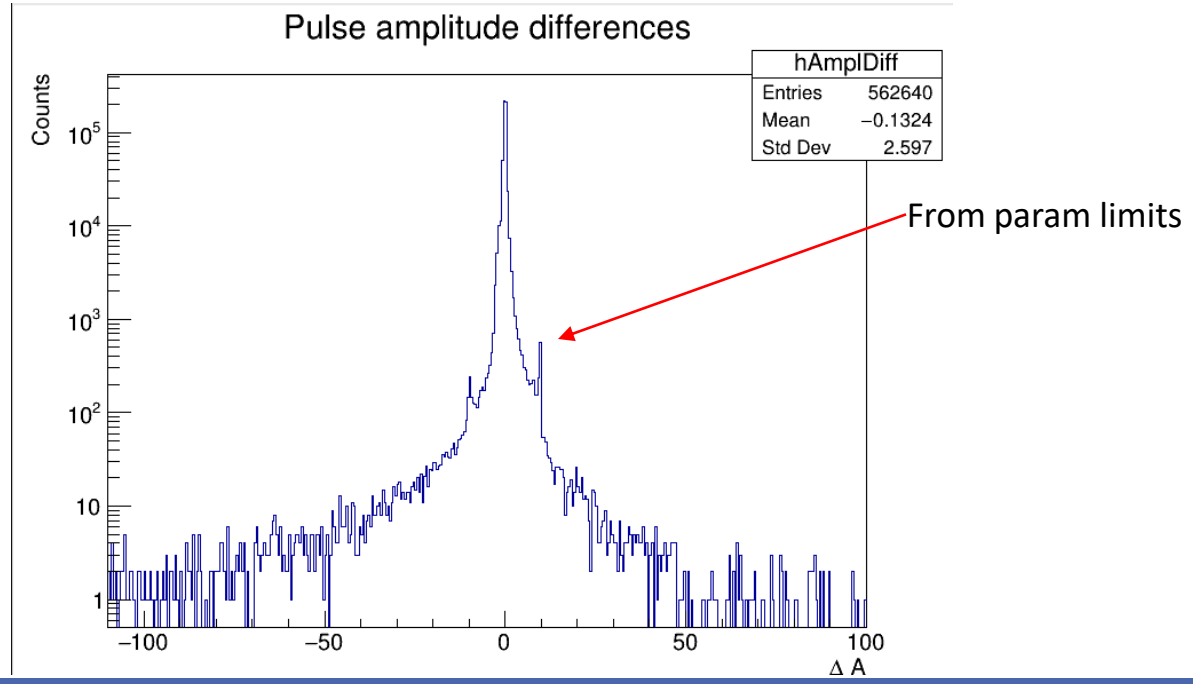
Reference Pulse Concerns: Fitting Noise

- No pulse found -> seed fit parameters with a 2mV pulse at the reference time AND still performs a fit
- Original code with TF1::Fit() can set strict fit parameter limits (0.05 , 10)

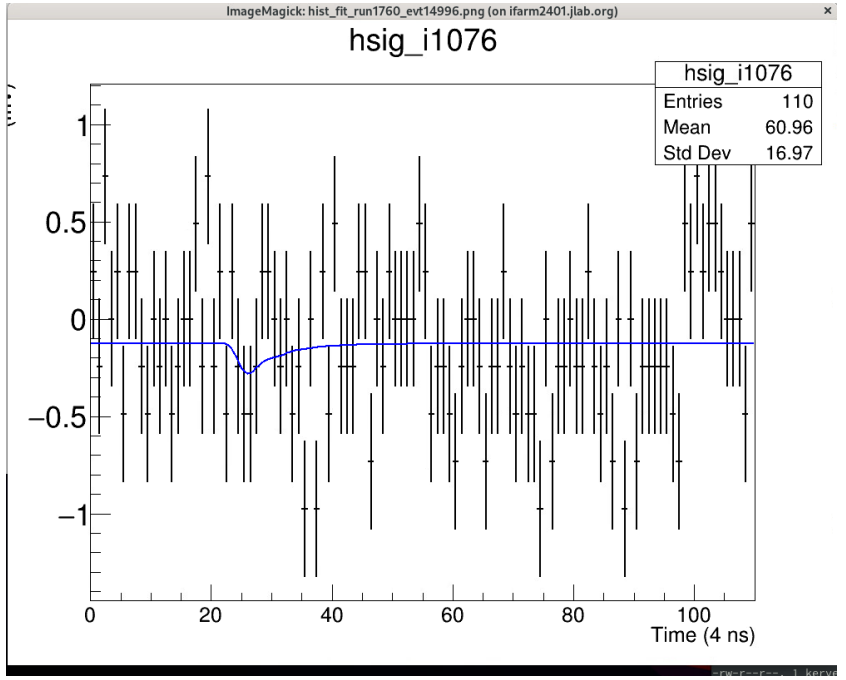
- Minuit2 will not stay bounded in order to minimize χ^2.

- Example:

TF1::Fit() is seeded with reference pulse Ampl=2



Minuit2 finds no pulse and fits noise (expensive) Ampl= -0.2

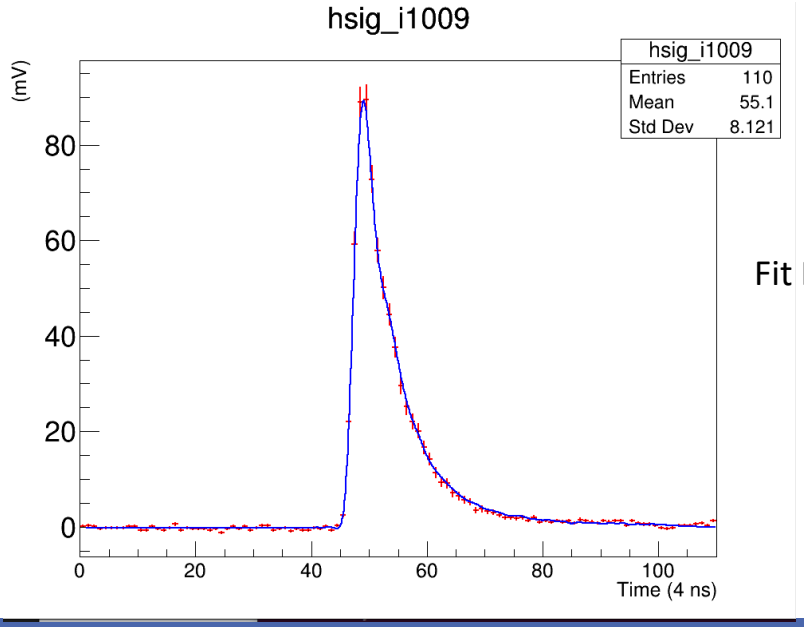


Reference Pulse Concerns: Another Example

- If a pulse is not within a certain distance of expected coincidence time, the fit is seeded with parameters of a new reference pulse of 2mV at the ref time.
- Original code with TF1::Fit() can set strict fit parameter limits (0.05 , 10)

- Minuit2 will not stay bounded in order to minimize χ^2.

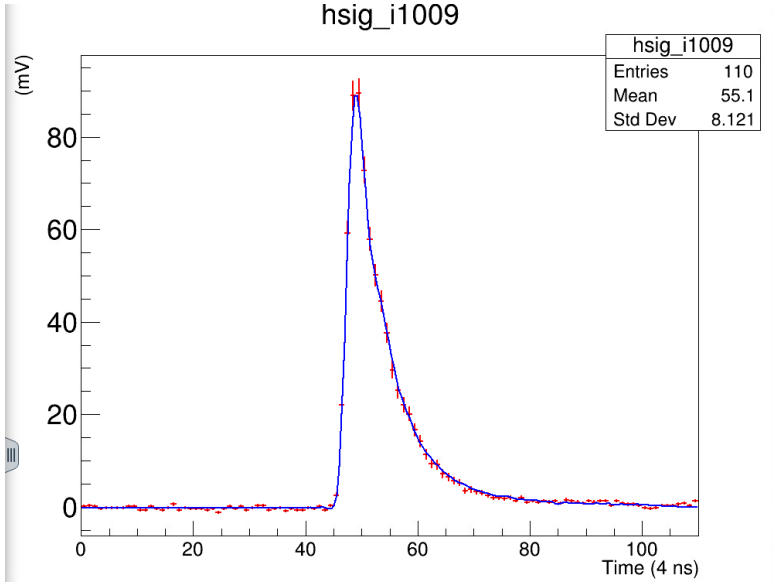
Example: Similar fit quality and both very good χ^2 . But got there by different means.
If only 1 set of parameters existed, it would give same result.



Fit Parameters:
89 , 0.05

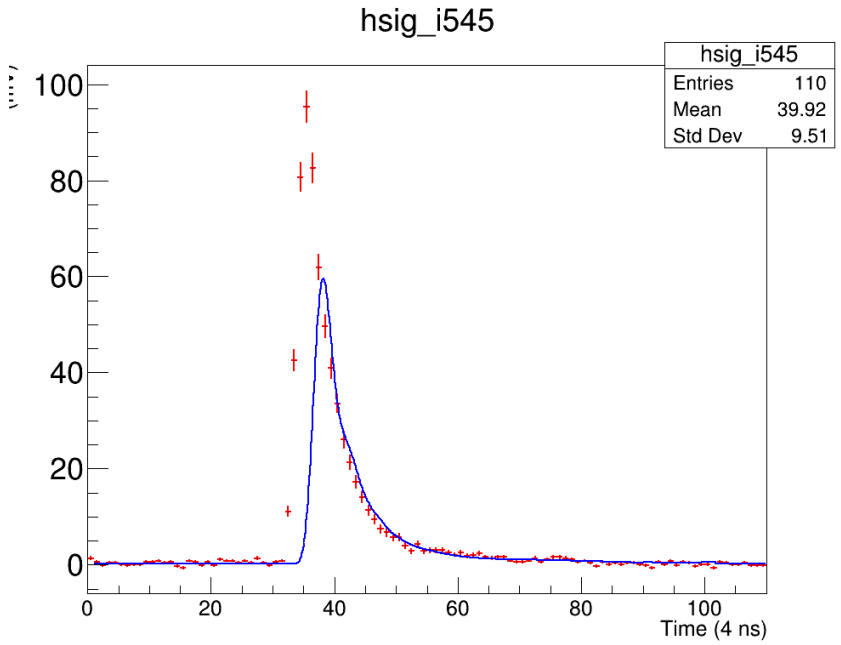
Second “pulse” parameter is bounded

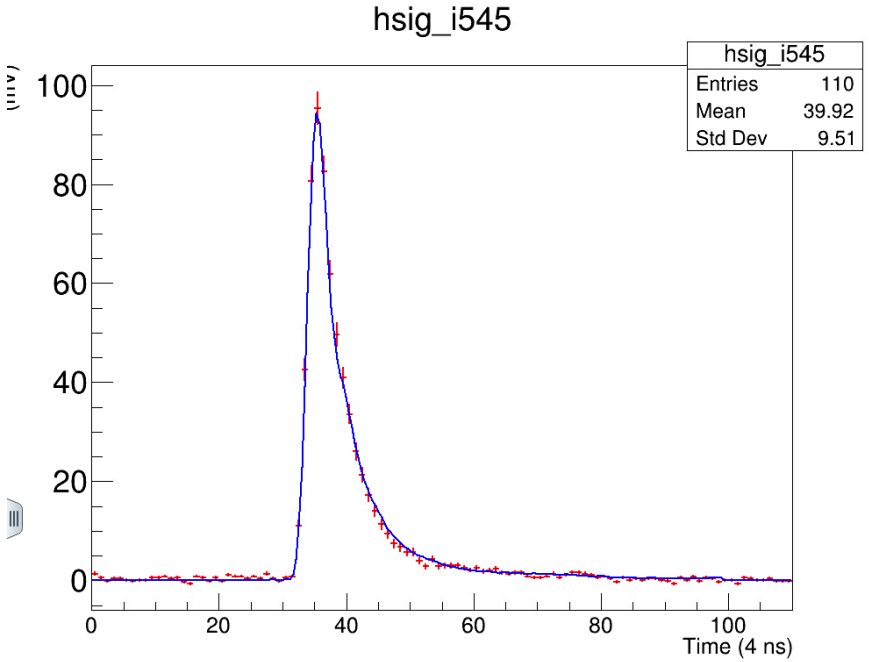
Fit Parameters:
308 , -219



Fit Failures

- Noticed in frequent instances of original code having strange fit behavior.
- Need to investigate with Wassim. Possibly due to the added reference pulse?
- Haven't noticed this behavior with the Minuit2 fits





Timing Performance

- 2 Test runs with most recent Minuit2 code
  - Less than linear speed up with core count ☹
  - A lot of CPU time wasted trying to fit noisy events. If 1 failed fit, it retries with more iterations and different (more expensive) fit strategy

  - Once a decision is made on how to treat these events, the time per event should decrease significantly

Run	# Events	Time 5k events (sec)	Approx time for all events	Total CPU Time	Real time	Time Saving	Cores Used	
1812	83,269	1,943	32,358	37,685	10,140	3.2x	4	ok
1776	192,664	810	31,211	52,408	6,840	4.6x	8	Not as great

Past performance test with old Fit code were closer to linear, so I feel confident improvements here!

Summary

- Original implementation with Rdataframe failed from non-thread-safe TF1::Fit() method.
  - Output fits matched original code very well though.

- Changed to a new fit method Minuit2
  - Thread-safe!
  - Cannot rely on constraining fit parameters

- Seeing odd fit behavior from edge cases when using added “reference pulses”
  - Still thinking about how to handle these

- Time saving does not yet seem linear with core count
  - Optimistic fitting less noisy events will greatly speed up code

- The Good News:

It runs, it runs faster, and it produces good fits (good χ^2)

Thank You!

Past Analysis Meeting Slides

Performance

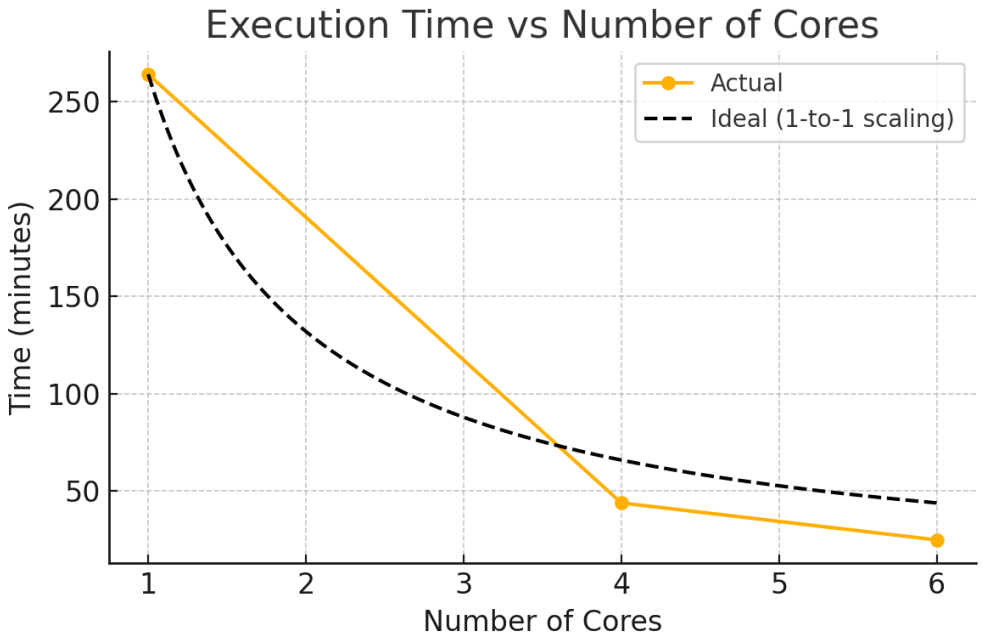
- Compared CPU time for a small segment of the kin_x25 pass2 replay. Run/segment 5137_10 (only 5012 events)

- When ran on an interactive farm node:
  - Original code = 4.4hr
  - MT code on 4 threads = 50min
  - MT code on 6 threads = 25min

- Better scaling than simply dividing by #cores because of small optimizations and not writing fitted waveform histos to output rootfile

- Job used ~1.25GB memory. Not sure how this scales with # of events? (more dataframe rows will grow memory footprint)

- Issue to investigate: Same exact code ran on the batch farm (swif2) took 2.5hr(4cores) and 1.25hr(6cores) . Same number of threads. Will reach out to Brad S. for ideas here.

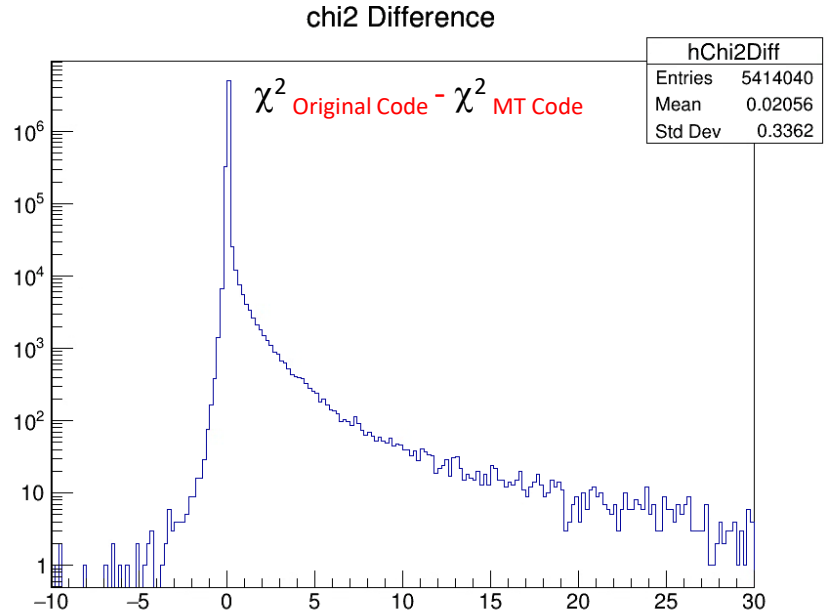


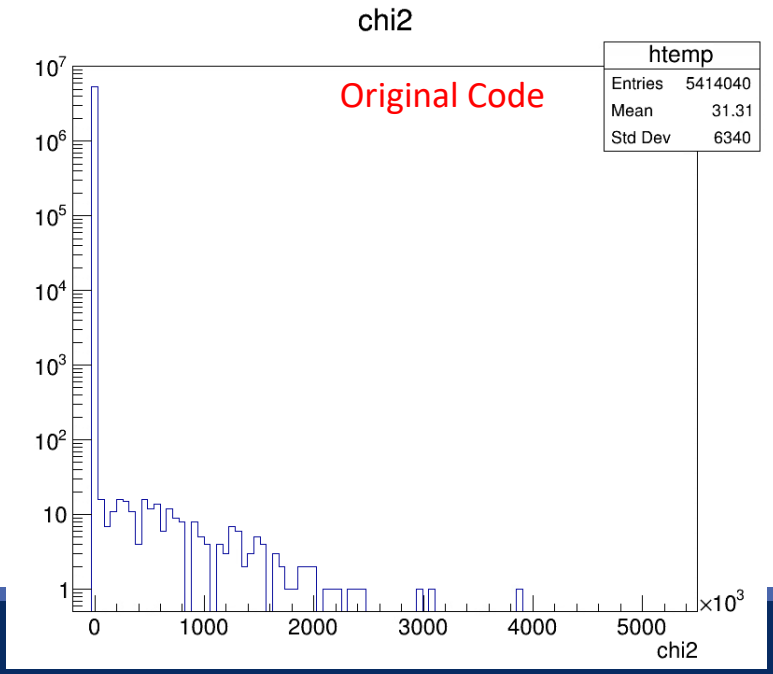
Waveform Results

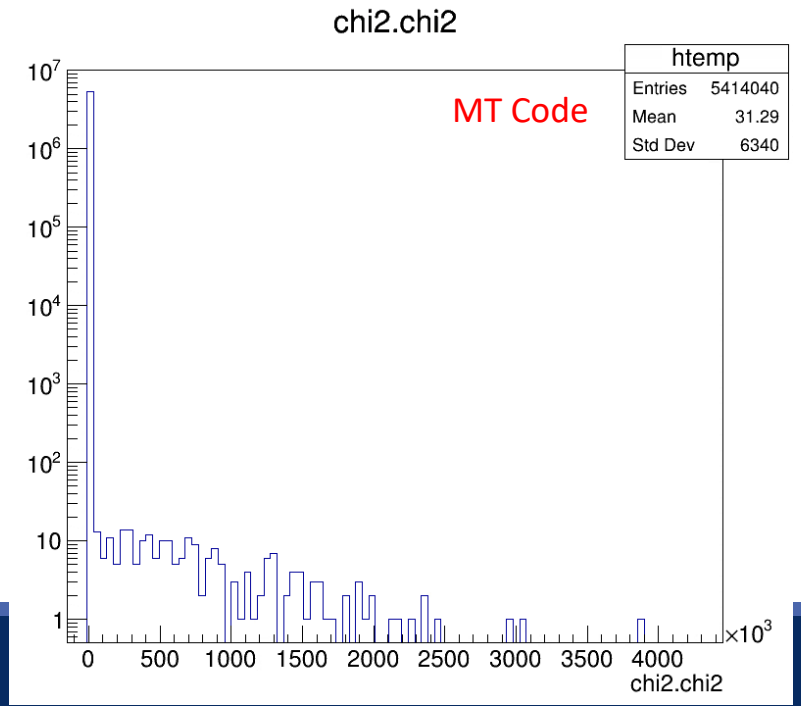
Tried to limit changing any logic of original code. Fits should be **exact** same.

Compare the distributions of χ^2 , wf amplitudes, and time shift of pulses w.r.t the reference waveform.

Distributions look similar, but not exactly the same ->Look at event-per-event difference





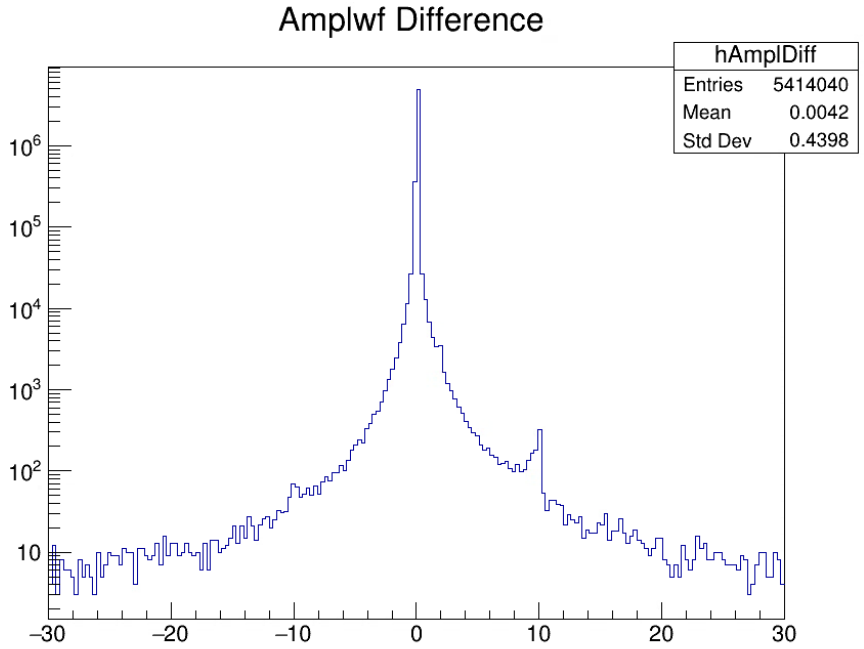


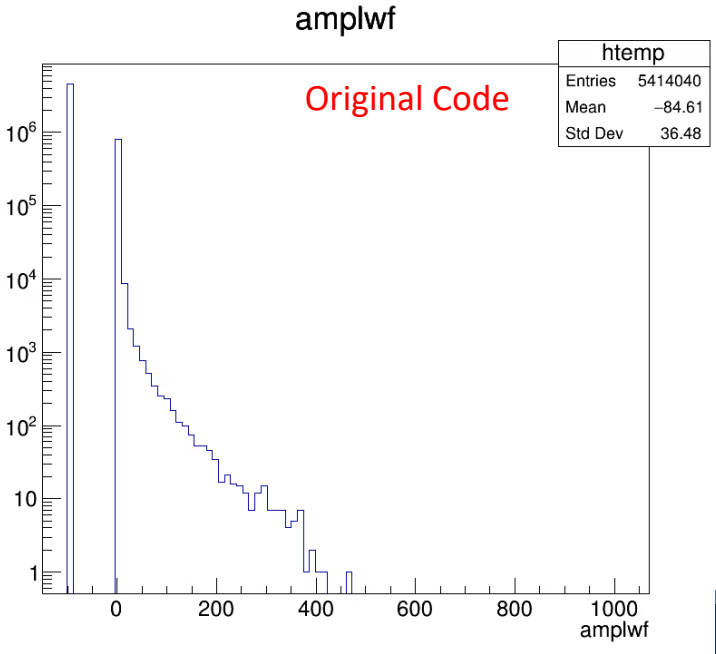
Waveform Results

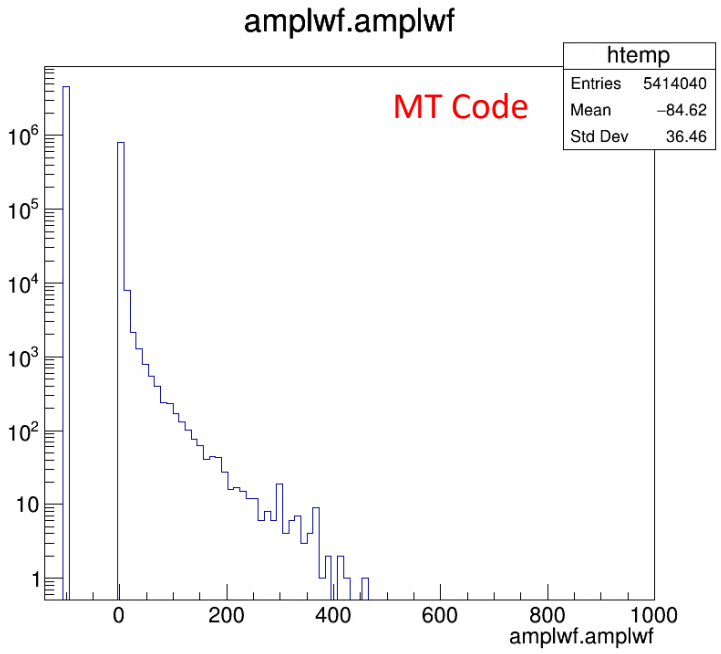
Tried to limit changing any logic of original code. Fits should be **exact** same.

Compare the distributions of χ^2, wf amplitudes, and time shift of pulses w.r.t the reference waveform.

Distributions look similar, but not exactly the same ->Look at event-per-event difference





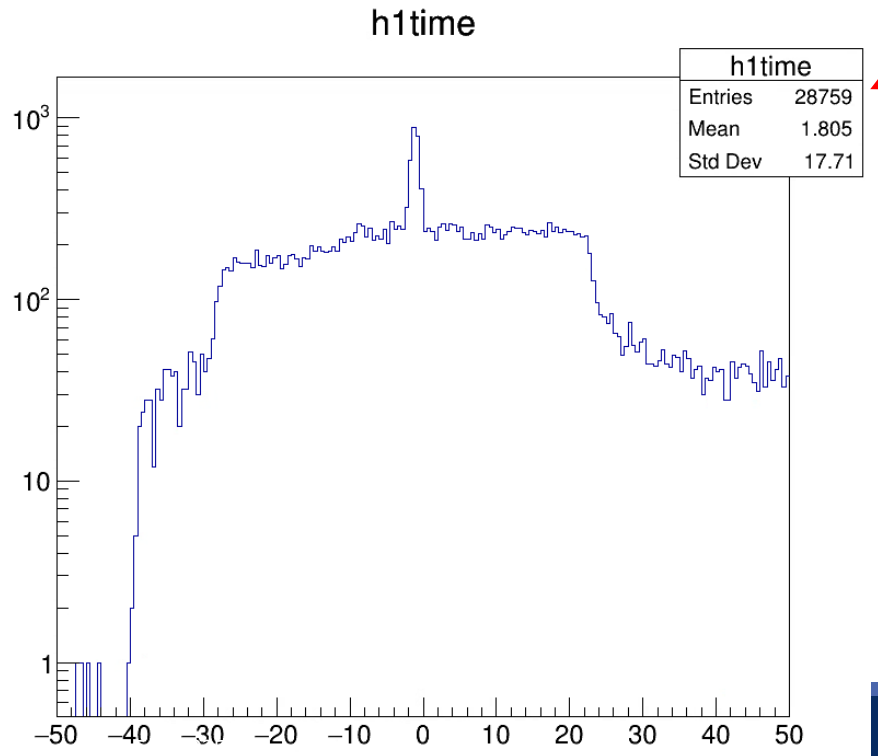


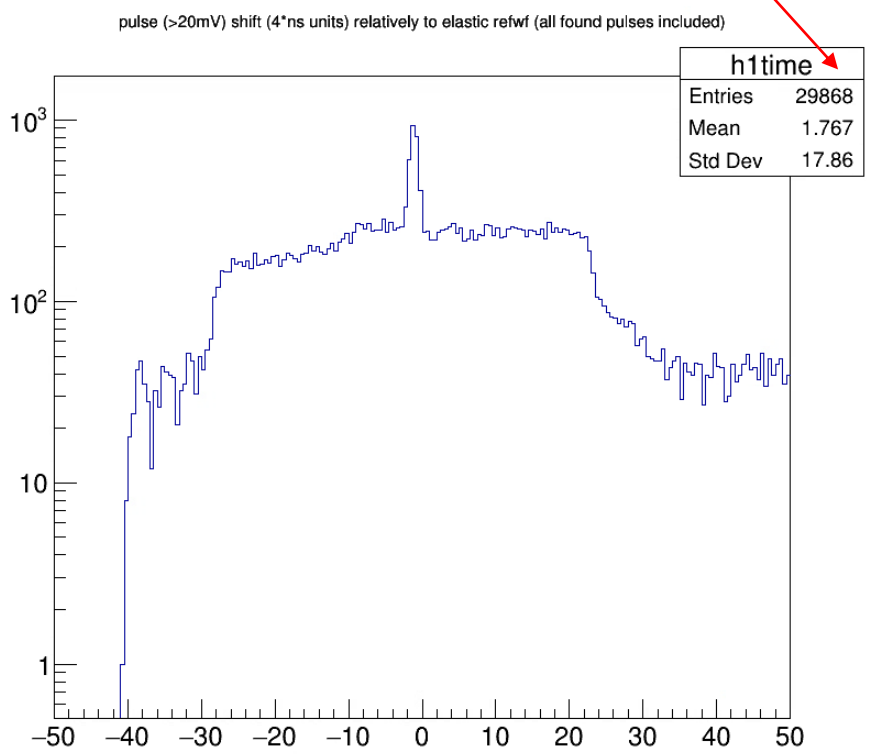
Waveform Results

Tried to limit changing any logic of original code. Fits should be **exact** same.

Compare the distributions of χ^2, wf amplitudes, and time shift of pulses w.r.t the reference waveform.

Difference in entries comes from a *new* restriction on the pulse selection in Wassim's code. Is now added to MT code.



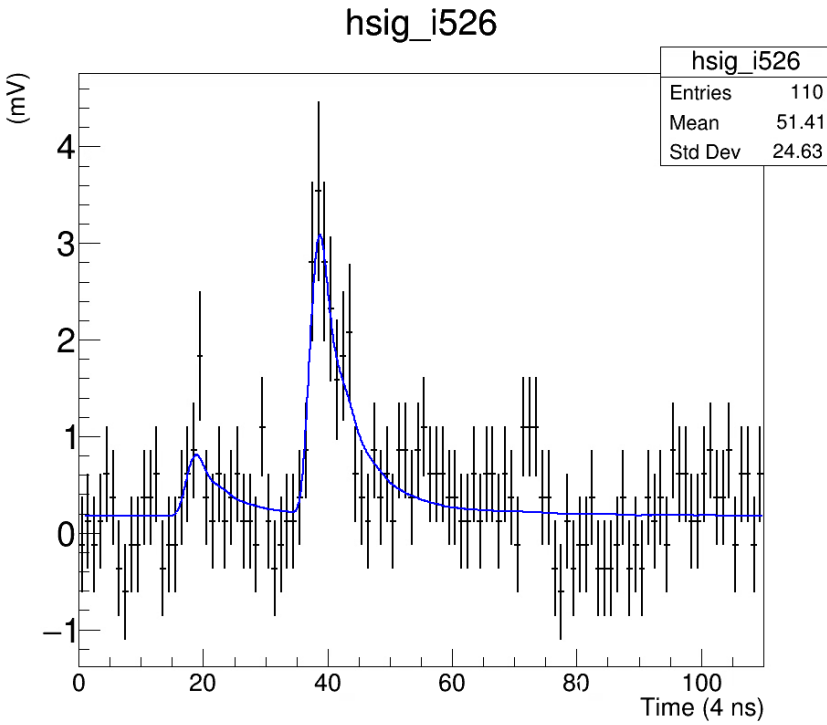


Waveform Fit Comparison?

Diagnostic interest to compare the WF fits for conflicting events

Several challenges in the way:

- ROOT rDataFrames does not support multi-threaded I/O
  - Each thread cannot simultaneously write histograms of the fitted waveform to the same rootfile
  - Storing the histograms in the dataframe would blow-up memory usage. Unlike serial event loops where the histos are written from memory to disk then deleted, the dataframe stores all histos in memory then “snapshots” everything to disk at the end (# histos in memory = 1080 blocks x millions of events!)
  - As a work around, each thread can write it’s event’s histos to it’s *own* rootfile. End up with many tiny root files of one events worth of histos. (Maybe can combine them in a post process?)
- Wassim’s original code only stores the last event histos in the rootfile. He would have to update his code and rerun it on
- Initial inspection: Selected 5 events with “large” χ^2 difference. Look at them by eye, fits look okay.



Summary

Multi-threaded version of waveform code runs and offers time saving proportional to number of cores used. Can significantly lower Walltime for large segments

Waveform fitting seems to be good (acceptable χ^2 and h1time distributions)

Fits don't seem to be **exactly** the same as Wassim's original code.

Doing event-by-event fit comparisons would take some time.

Still want to investigate memory usage scaling for very large run segments

Code found at: https://github.com/mkerv/nps-waveform-analysis/tree/timing-update