

dunetpc - Task #23810

Improve AdcPedestalFitter_tool.cc

12/27/2019 04:49 PM - Tingjun Yang

Status:	Closed	Start date:	12/27/2019
Priority:	Normal	Due date:	
Assignee:	David Adams	% Done:	0%
Category:		Estimated time:	0.00 hour
Target version:			
Description			
The following tool is used in ProtoDUNE-SP dataprep to fit pedestals for each channel: https://cdcvs.fnal.gov/redmine/projects/dunetpc/repository/revisions/develop/entry/dune/DataPrep/Tool/AdcPedestalFitter_tool.cc			
We would like to improve the speed of this code. Gianluca made the following suggestions:			
<pre>You can definitely use `hit::CompiledGausFitCache<1U>`, which works great as long as do not attempt multithreading with it, or else you'll need to take the usual multithreading precautions with complicate caching by a lot. But that helps only if creating the TF1 is the task that is devouring your processing time.</pre>			
<pre>This may end up not being your case. Just taking a look at the code, it seems it's doing a lot of things: allocating and filling two new ROOT histogram, creating a ROOT function (a Gaussian), cloning it, running a fit, and possibly opening a ROOT file, cloning an histogram, writing it into that file, and cleaning up.</pre>			
<pre>The best action depends on what of this is essential and what is not. For example, histogramming may be done without ROOT for some gain in resource usage, and simple single-Gaussian fit may be done with alternative methods (e.g. using `lar::util::GaussianFit` if its requirements are satisfied). As you can well imagine, each choice comes with a price.</pre>			
I am also adding Kyle and Saba for their inputs on any other potential improvements.			

History

#1 - 10/21/2020 12:14 PM - David Adams

- Assignee set to David Adams

- Status changed from New to Work in progress

In a test job referenced by Tingjun, dataprep takes 93 of 557 seconds. Far from dominant but still too large.

When I run dataprep only on dunegpvm15, dataprep takes 42 seconds/event with 21 seconds/event spent the pedestal fitter referenced here. I am looking into reducing that.

#2 - 10/26/2020 09:08 AM - David Adams

I have done a lot of work with the tool. Many (8?) seconds per event (starting from 21 sec) are saved by dropping the unneeded call to Sumw2, avoiding copies of TH1 and TF! and doing block fills of the histogram instead of calling Fill for each tick.

Another 2.9 sec/event (for my one test event) are saved by using a cached TF1 Gaussian fit function instead of creating a new instance for each channel. This comes at the price of adding state to the tool and complicating thread safety (which is not yet implemented).

Most of the remaining time is spent in the fitter when a Gaussian fit is performed for each channel. A likelihood fit was being performed to cope with a few channels for which the least-squares fit failed. There is now a fcl parameter to control the fitter with the following options:

```
// FitOpt - 0: Use histogram mean (no fit)
//           1: Chi-square fit
//           2: Likelihood fit
//           3: Chi-square fit followed by likelihood fit if chi-square fit fails
```

For the test event the time in sec for each of these is

Opt	Time
0	1.9
1	5.9
2	10.3
3	5.9

I propose moving to option 3 for now. It should show little or no difference in the pedestal or pedestal RMS compared to option 2 which is similar to what we do now. We have to check and likely fix the RMS calculation if we want to go to option 0.

#3 - 10/26/2020 09:59 AM - David Adams

There is also a new fcl parameter FitPrecision which is passed to to TFit::SetPrecision to control the speed and precision of the least squares fits. It was set to 1.0 for the above results. Here is a summary table:

Prec	Time [sec]
0.0001	6.1
0.001	6.1
0.01	6.1
0.1	6.0
1.0	5.9
10	5.8
100	5.7

I propose to leave this at 1.0 for now.

#4 - 10/26/2020 10:24 AM - Tingjun Yang

Hi David,

Thanks for the updates. All look very interesting.

By the way, Mike Wang implemented a Levenberg-Marquardt algorithm to replace the root Gaussian fit used in the GausHitFinder module, which improves the speed of GausHitFinder by a factor of ~8:

https://github.com/LArSoft/larrecoblob/develop/larrecoblob/HitFinder/HitFinderTools/PeakFitterMrqdt_tool.cc

Is it possible to use this tool for the pedestal finder?

Thanks,
Tingjun

#5 - 10/26/2020 10:54 AM - Tingjun Yang

Mike also pointed me to the document on the Marquardt-based Fitter:

<https://home.fnal.gov/~mwang/larsoft/gshfomt-v1.0.pdf>

#6 - 10/26/2020 10:57 AM - David Adams

I had a quick look at MarqFitAlg used by the code you reference. It looks like it is for fitting with multiple Gaussians.

#7 - 10/26/2020 11:49 AM - David Adams

I added the RMS evaluation (TH1::GetRMS) for fitting option 0 so it now gives reasonable pedestal and pedestal RMS. Should we get to the point where 4 sec/event is a big part of our reco budget, we might want to switch to that option.

I will commit to dune_tpc after a few more check and after updating to dune_tpc v09_08_00.

#8 - 10/26/2020 01:10 PM - Michael Wang

Hi,
MarqFitAlg was originally meant to be used only with GausHitFinder. But the Levenberg-Marquardt algo used is not tied specifically to Gaussians. So, in principle, MarqFitAlg can be adapted to support any fitting function/s.

#9 - 10/26/2020 02:48 PM - Tingjun Yang

Hi David,

There is an error in the latest CI test:

<https://dbweb5.fnal.gov:8443/LarCI/app/ns:dune/storage/docs/2020/10/26/stderr%23y2iC9sC.log>

31: %MSG-s ArtException: Early 26-Oct-2020 13:40:30 CDT JobSetup

32: cet::exception caught in art

33: ---- OtherArt BEGIN

34: ServiceCreation

```

35: ---- Configuration BEGIN
36:  make_tool:
37:  ---- Can't find key BEGIN
38:    FitOpt
39:  ---- Can't find key END
40:  Exception caught while processing plugin spec: AdcPedestalFitter
41:  ---- Configuration END
42:  cet::exception caught during construction of service type ToolBasedRawDigitPrepService:
43:  ---- OtherArt END
44:  %MSG

```

#10 - 10/26/2020 04:08 PM - David Adams

I neglected to push the update of the configuration. The new develop version should fix this. Sorry.

#11 - 10/26/2020 04:43 PM - David Adams

To respond to Michael's point above, first thank you. I cannot judge without significant effort whether the algorithm you reference will be faster than Minuit. If I had an implementation, I would try. Otherwise, it will be while before this rises to the top of my priority queue. --da

#12 - 10/26/2020 05:01 PM - Tingjun Yang

In the latest CI test, the dataprep does take less time:

450: decode:caldata:DataPrepByApaModule	55.8334	55.8334	55.8334	55.8334	0	1
compared with the previous CI test:						
450: decode:caldata:DataPrepByApaModule	87.7081	87.7081	87.7081	87.7081	0	1

However, the memory footprint seems to increase. In the latest CI test

```

494: Peak virtual memory usage (VmPeak) : 3278.57 MB
495: Peak resident set size usage (VmHWM): 2144.67 MB
496: Details saved in: 'mem.db'

```

In the previous CI test:

```

494: Peak virtual memory usage (VmPeak) : 3462.42 MB
495: Peak resident set size usage (VmHWM): 1866.52 MB
496: Details saved in: 'mem.db'

```

#13 - 10/26/2020 05:21 PM - David Adams

With all the going back and forth with caching histograms, I may have forgotten to clean up my temporaries, i will have a look.

#14 - 10/26/2020 05:32 PM - David Adams

I fixed one obvious problem. See if that helps. I will try to check the dataprep sequence with valgrind tomorrow.

#15 - 10/27/2020 08:34 AM - David Adams

I see more leaks and am working on them.

#16 - 10/27/2020 09:30 AM - David Adams

Leaks are plugged in the version I just committed. Sorry for the trouble.

#17 - 10/27/2020 12:28 PM - Tingjun Yang

The memory footprint seems to be back to normal in the latest CI test:

<https://dbweb5.fnal.gov:8443/LarCI/app/ns.dune/storage/docs/2020/10/27/stderr%23RXhDNSe.log>

```

494: Peak virtual memory usage (VmPeak) : 3055.46 MB
495: Peak resident set size usage (VmHWM): 1915.83 MB
496: Details saved in: 'mem.db'

```

#18 - 10/27/2020 12:29 PM - David Adams

- Status changed from Work in progress to Closed

Thanks for confirming. I close this ticket.

#19 - 10/27/2020 12:36 PM - Tingjun Yang

Also thanks David for adding the information on the processing time for dataprep tools:

612: ToolBasedRawDigitPrepService:dtor:	digitReader:	0.32 sec/event
613: ToolBasedRawDigitPrepService:dtor:	pdsp_sticky_codes_ped:	0.07 sec/event
614: ToolBasedRawDigitPrepService:dtor:	pd_adcPedestalFit:	13.02 sec/event
615: ToolBasedRawDigitPrepService:dtor:	adcSampleCalibration:	1.11 sec/event

616: ToolBasedRawDigitPrepService:dtor:	pdsp_gainCorrection:	0.12 sec/event
617: ToolBasedRawDigitPrepService:dtor:	pdsp_adcMitigate:	0.83 sec/event
618: ToolBasedRawDigitPrepService:dtor:	pdsp_timingMitigate:	0.01 sec/event
619: ToolBasedRawDigitPrepService:dtor:	pdspTailPedRemovalZKe:	6.29 sec/event
620: ToolBasedRawDigitPrepService:dtor:	pdsp_noiseRemovalKe:	21.21 sec/event
621: ToolBasedRawDigitPrepService:dtor:	adcKeepAllSignalFinder:	0.25 sec/event
622: ToolBasedRawDigitPrepService:dtor:	adcScaleKeToAdc:	0.17 sec/event
623: ToolBasedRawDigitPrepService:dtor:	pdsp_RemoveBadChannels:	0.01 sec/event

#20 - 10/27/2020 06:26 PM - Tingjun Yang

The latest CI report:

611: ToolBasedRawDigitPrepService:dtor:	Time report for 12 tools.	
612: ToolBasedRawDigitPrepService:dtor:	digitReader:	0.21 sec/event
613: ToolBasedRawDigitPrepService:dtor:	pdsp_sticky_codes_ped:	0.06 sec/event
614: ToolBasedRawDigitPrepService:dtor:	pd_adcPedestalFit:	15.32 sec/event
615: ToolBasedRawDigitPrepService:dtor:	adcSampleCalibration:	0.77 sec/event
616: ToolBasedRawDigitPrepService:dtor:	pdsp_gainCorrection:	0.13 sec/event
617: ToolBasedRawDigitPrepService:dtor:	pdsp_adcMitigate:	0.89 sec/event
618: ToolBasedRawDigitPrepService:dtor:	pdsp_timingMitigate:	0.01 sec/event
619: ToolBasedRawDigitPrepService:dtor:	pdspTailPedRemovalZKe:	5.72 sec/event
620: ToolBasedRawDigitPrepService:dtor:	pdsp_noiseRemovalKe:	22.38 sec/event
621: ToolBasedRawDigitPrepService:dtor:	adcKeepAllSignalFinder:	0.25 sec/event
622: ToolBasedRawDigitPrepService:dtor:	adcScaleKeToAdc:	0.19 sec/event
623: ToolBasedRawDigitPrepService:dtor:	pdsp_RemoveBadChannels:	0.01 sec/event

Could be just fluctuation.

#21 - 10/28/2020 08:16 AM - David Adams

I get stabler (RMS ~ 1%) and much smaller values running on (lightly loaded) dunegpvm15:

ToolBasedRawDigitPrepService:dtor:	Event count:	1
ToolBasedRawDigitPrepService:dtor:	Call count:	6
ToolBasedRawDigitPrepService:dtor:	Time report for 12 tools.	
ToolBasedRawDigitPrepService:dtor:	digitReader:	0.08 sec/event
ToolBasedRawDigitPrepService:dtor:	pdsp_sticky_codes_ped:	0.03 sec/event
ToolBasedRawDigitPrepService:dtor:	pd_adcPedestalFit:	6.21 sec/event
ToolBasedRawDigitPrepService:dtor:	adcSampleCalibration:	0.36 sec/event
ToolBasedRawDigitPrepService:dtor:	pdsp_gainCorrection:	0.04 sec/event
ToolBasedRawDigitPrepService:dtor:	pdsp_adcMitigate:	0.38 sec/event
ToolBasedRawDigitPrepService:dtor:	pdsp_timingMitigate:	0.01 sec/event
ToolBasedRawDigitPrepService:dtor:	pdspTailPedRemovalZKe:	2.93 sec/event
ToolBasedRawDigitPrepService:dtor:	pdsp_noiseRemovalKe:	9.97 sec/event
ToolBasedRawDigitPrepService:dtor:	adcKeepAllSignalFinder:	0.14 sec/event
ToolBasedRawDigitPrepService:dtor:	adcScaleKeToAdc:	0.04 sec/event
ToolBasedRawDigitPrepService:dtor:	pdsp_RemoveBadChannels:	0.00 sec/event