LArSoft - Support #15100

Need faster NearestWire look up method

01/11/2017 01:30 PM - Brian Rebel

Status:	Closed	**Start date:**	01/11/2017
Priority:	High	**Due date:**	
Assignee:	Gianluca Petrillo	**% Done:**	100%
Category:	Geometry	**Estimated time:**	0.00 hour
Target version:		**Spent time:**	11.00 hours
Experiment:	DUNE, MicroBooNE	**Co-Assignees:**	

Description

The current methods to map a position to the nearest corresponding readout wire are very slow for large detectors with high rates (i.e. on the surface). It would be good to examine how those methods work and see if they can be made more efficient.

Both MicroBooNE and the DUNE ND LArTPC option are reporting heavy overheads and have traced them to the use of this method in the LArVoxelReadout::DriftIonizationElectrons method.

I know that Wes is working on a feature branch that strips the mapping of a group of ionization electrons to the channel out of that method and will instead simply create a collection of TDCIDEs that can be consumed by a downstream module to create the SimChannels.

Doing so only shoves the problem to a different location, unfortunately, so a more permanent fix needs to be implemented.

Additionally, it might make sense to rename the methods NearestChannel rather than NearestWire as some proposed TPCs do not use wires.

Related issues:

Related to LArSoft - Bug #15131: LArG4: diffusion causes errors in SpaceCharg...	**Closed**	**01/12/2017**
Blocked by LArSoft - Feature #14364: Build a geometry infrastructure for supp...	**Closed**	**11/04/2016**

History

#1 - 01/11/2017 01:32 PM - Brian Rebel

- Priority changed from Normal to High

This is actually an urgent issue as it interferes with MC production taking place in a timely manner. I heard that uBooNE takes 120s per interaction to simulate...and the beam spills are every 2 seconds so we will never be able to produce more MC than data at that ratio (excepting having a boatload of nodes chugging away).

#2 - 01/11/2017 03:53 PM - Gianluca Petrillo

- Category set to Geometry

Can you document the test you did to identify the problem in NearestWire?
I want to reproduce it.

P.S. I do have an alternative implementation in feature/gp_protoDUNEDP_trueOrientation, but I am not even sure it's faster.

#3 - 01/11/2017 04:21 PM - Brian Rebel

Hi Gianluca

I don't have any data on it, but Wes and James do, so I am going to let them chime in.

#4 - 01/11/2017 06:18 PM - Gianluca Petrillo

In the meanwhile, I reread the code in LArVoxelReadout::DriftIonizationElectrons(). Every time I do, I bleed a bit.

My recurring question is: does it make real sense to split O(10k) electrons in groups of 20, ask 500 times "which is the nearest wire for this slightly smeared position?", and get 50% of the time one answer, and 50% of the times another?
I think the optimisation is not in making NearestWire() faster, but in making less calls to it.
If the original position (x;y;z) has wire coordinate wc 7.8 and we project the smearing vector on the "wire coordinate direction" (that is the direction that the wires measure), we know that everything that gets a smearing between -0.3 and +0.7 stays on that wire, with +0.7 to +1.7 on the next wire, -1.3 to -0.3 on the previous... int(wc + smear) or something like that. With a single WireCoordinate() call rather than 500 NearestWire().

Am I missing anything?
I could go further into questioning the fineness of the smearing, but that is trickier in 3D (I think it is important along drift direction, not as much on the wire coordinate direction, not at all on the wire direction).

#5 - 01/11/2017 10:21 PM - Brian Rebel

Hi Gianluca

So there is both transverse and longitudinal diffusion, with the transverse diffusion being about twice the size of the longitudinal diffusion. The scale of the transverse diffusion is about 2 mm over a 2 ms drift time. That is of course the 1 sigma value, so you can have a change, as you say of potentially up to one channel, depending on the direction of the diffusion and the channel pitch.

I like the direction your thinking is going - can we determine the closest channel for the energy deposition location projected to the readout and then smear from there with a less intensive calculation? I guess you could determine the magnitude of the distance of the smeared location from the central value and the direction in the YZ plane. Then if using wire readout one could compare that direction to the orientation of the wires for the desired plane and if the direction weighted difference in positions is large enough, you put that amount of charge on a different channel. Of course, in the case of wire readout, most of the time the central point will be between two wires and then you will have to handle that properly.

#6 - 01/12/2017 04:37 AM - Wesley Ketchum

Hi.

So, based on v06_18_01, there is a 'feature/wketchum_SplitLArG4' branch pushed up to lardataobj and larsim. Lardataobj contains a new "SimEDep" product to hold information on energy depositions (and the info needed for the DriftIonizationElectrons function). In LArG4_module and the underlying PhysicsList/LArVoxelREadout code, I now create a collection of SimEDep products when the DisableWireplanes parameter is set to 'true' in LArG4Parameters service. One should not need any additional changes...(so, use standard_larg4_uboone.fcl plus the override).

I found the following on a VERY limited number of MicroBooNE cosmics events:
---With DisableWireplanes set to true but doing none of the LArVoxelReadout steps (only doing the ParticleActionList), events took ~20 seconds to process through LArG4. (This you can test without the above changes in the branches, and just changing the parameter in the fcl).
---With DisableWireplanes set to true, but with my changes above where I now readout the EDeps in the LArVoxelReadout methods, it takes about 50 seconds to process through LArG4.
---With DisableWireplanse set to false, which should replicate the standard behavior, it takes about 100 seconds to process.

You can test on a file like the one here:
/pnfs/uboone/scratch/users/uboonepro/mcc8_val/v06_18_01/sim/prodcosmics_corsika_cmc_uboone_noDDR/0/1/1/16067129_0/prodcosmics_corsika_cmc_uboone_noDDR_0_20170106T045935_gen0.root. Or any in that general area. Or generate your own corsika cosmics!

I will say that I've done nothing that says NearestWire is the problem: the whole DriftIonizationElectrons function should be under suspicion, and I would agree with Gianluca's assessment that the diffusion seems suboptimally done, both in terms of execution but also, likely, in terms of physics output.

Any quick fix/idea is certainly appreciated, as it could have impact now as a patch for MicroBooNE's simulation campaign. However, that should be limited to replicating the behavior of the existing code. There's likely a longer discussion to be had on the right way to do this longer term. Likely, one should have code that will take an EDep/electron bunch and determine the size after diffusion at some point approaching the first wire plane. Then, the model of the field lines should probably be used to determine the signals on the wires themselves, which implies tighter coupling with code that is in detsim. I have ideas on doing that efficiently, but that's another discussion. IMO, LArG4 output should likely be the EDeps, and the rest moved to DetSim, with the SimChannels as an intermediate stage towards the RawDigit. Again, another discussion.

Let me know if more info is needed.

#7 - 01/12/2017 08:46 AM - Brian Rebel

Hi Wes

I think a fuller discussion of this issue is probably needed in a LArSoft coordination meeting (or whatever we are calling those these days).

I am not opposed to your idea of using a new data product, and I agree that it probably makes sense to output something like your new data product or a TDCIDE collection at the LArG4 stage rather than SimChannels. The SimChannels (if they are still needed at all) could be made either in a separate downstream module or in the same module that is currently doing the detector simulation. If using your new data product, we should be careful not to use more precision than necessary (i.e. positions probably only need to be floats, times may need to be doubles, but maybe not) as there will be a whole lot of those products made and we don't want to bloat the file size unnecessarily.

The other concern I have is that we need to ensure we don't break downstream functionality, specifically the BackTracker code. I know a lot of people (at least in LArIAT) are using that code and it depends on the sim::IDE portion of the SimChannels.

We also have to be sure that we can still use already produced files, so it may be that some translating module needs to be created that changes sim::IDEs to your new data products, if that is the way we end up going.

#8 - 01/12/2017 08:13 PM - Gianluca Petrillo

Brian, I need you or James hand me a precise case with:

1. input file
2. FHiCL configuration file

3. description of the observed behaviour (that is, how long it takes)
4. desired behaviour (that is, how long you think it should take)

I can't use MicroBooNE as a test case for LArIAT: it is two different issues. I have profiled MicroBooNE job that Wesley handed to me:

- it took about 150 seconds
- disabling space charge, it took 100 seconds
- Geant4 processed more than 500k particles
- photon library cost 10 seconds to load (I think it's just once, but I profiled a single event)

Neither of these apply for LArIAT, at the best of my knowledge.

About MicroBooNE profiling: apart from space charge, the processing time is split about evenly between core Geant4 and LArSoft hooks. If LArSoft became instantaneous, that job would be still ~100 seconds. Meaning: the problem is not in LArSoft code speed. Possibly, instead, it is in the choices that are coded in.
NearestWire() is actually not called, NearestChannel() is and it accounts for 3% of the processing. Some time may be wasted because if diffusion moves the points outside the range of space charge map, the latter returns nan and NearestChannel() is asked which channel is closest to (-0.3, nan, nan), as amply documented by the thousand error messages on the log. It throws an exception, that is slow. Anyway, not really an actor in the optimisation.
In fact, the "problem" is that there is no single component that is accountable for a large slow down by itself. True is that LArVoxelReadout::DriftIonizationElectrons() takes 16% of the time, which is something. With some effort, that might be halved, but it won't change the balance.

#9 - 01/12/2017 08:22 PM - Gianluca Petrillo

- Related to Bug #15131: LArG4: diffusion causes errors in SpaceCharge computation added

#10 - 01/13/2017 09:09 AM - Brian Rebel

Hi Gianluca

LArIAT is not concerned with this issue given its small size. I raised it on behalf of James and Wes who I heard were having problems.

Thanks for looking into the issue. I think there is still a definite problem with the amount of time that it takes to process a single event, especially in a high rate beam like the LBNF beam. We should not settle for 100s per beam spill to process a single MC event.

I would suggest that tracking down the causes of this long processing time should become a priority of the project.

Thanks

#11 - 01/23/2017 10:38 AM - Lynn Garren

- Blocked by Feature #14364: Build a geometry infrastructure for support of arbitrary drift direction added

#12 - 01/23/2017 10:38 AM - Lynn Garren

- Status changed from New to Assigned

- Assignee set to Gianluca Petrillo

We do not believe that this is a problem in current code. However, we expect new code in 2-4 weeks. Once the new code is available we will profile again. If no problem appears, we will close this ticket.

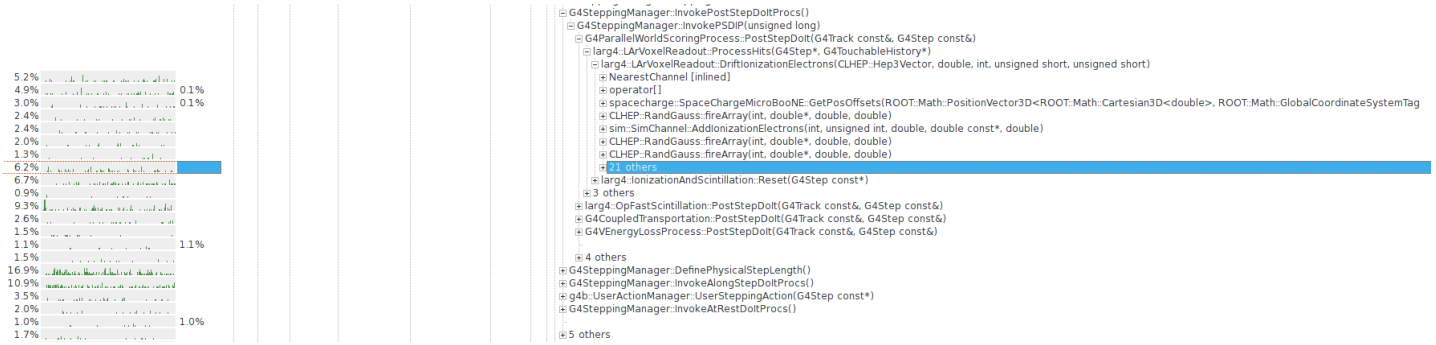
#13 - 11/01/2017 06:03 PM - Gianluca Petrillo

- File LArG4profiling.png added

I have run the allinea profiler on a LArG4-only job based on standard_g4_uboone.fcl with uboonecode v06_55_00.
Apparently the profiler collects only 1000 samples, that means the resolution of each figure is 0.1% and the (Poisson) uncertainty on a 5% is about 0.7%.
 <LArG4profiling.png
The screenshot shows a detail of the results.
Off screen, the time spent in LArG4::produce() is 98.3%. Half of it is more or less related to LArSoft, on third (34.1%) specifically from ProcessHits().
In there, NearestChannel() takes about 5% of the time (roughly the same as map::operator[]).
While there is room for improvement, in particular when the TVector3 interface is dropped for geo::Point_t, I don't consider a 5% important enough to act on reducing it.

#14 - 11/01/2017 06:04 PM - Gianluca Petrillo

I have run the allinea profiler on a LArG4-only job based on standard_g4_uboone.fcl with uboonecode v06_55_00.
Apparently the profiler collects only 1000 samples, that means the resolution of each figure is 0.1% and the (Poisson) uncertainty on a 5% is about 0.7%.



The screenshot shows a detail of the results.
Off screen, the time spent in LArG4::produce() is 98.3%. Half of it is more or less related to LArSoft, on third (34.1%) specifically from ProcessHits().
In there, NearestChannel() takes about 5% of the time (roughly the same as map::operator[]).
While there is room for improvement, in particular when the TVector3 interface is dropped for geo::Point_t, I don't consider a 5% important enough to act on reducing it.

#15 - 11/01/2017 06:05 PM - Gianluca Petrillo

- Status changed from Assigned to Resolved

- % Done changed from 0 to 100

#16 - 11/09/2017 11:38 AM - Gianluca Petrillo

- Status changed from Resolved to Closed

Files

LArG4profiling.png	94.8 KB	11/01/2017	Gianluca Petrillo