Checking memory usage of WCT Signal Processing in *art*

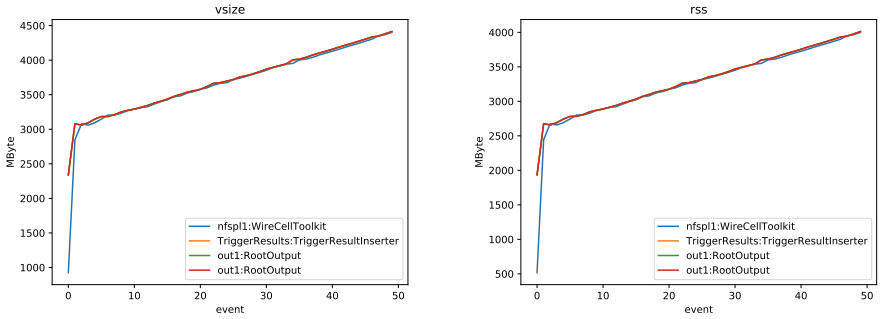
Brett Viren

December 27, 2018

1 Starting point

Starting point was "greater than 4GB" usage which caused problems with production jobs failing. Two tools were used to study the problem.

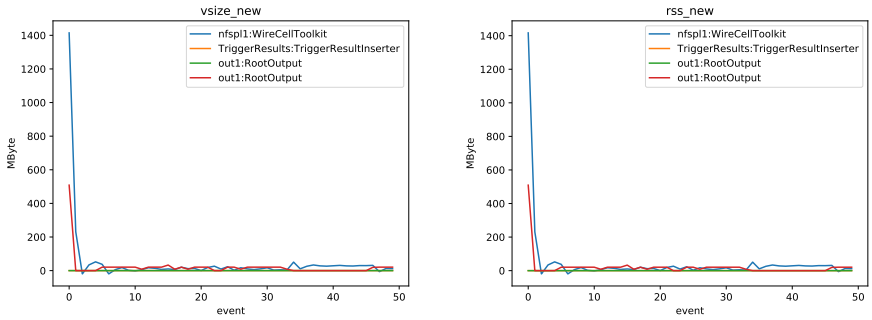
First, the "memcheck" feature of *art* was used to produce SQLite3 database files and some Python code was developed to plot the results. The plots shown here are from that code.



As shown in these plots, *art* samples its own Vsize and RSS. The former includes virtual memory usage while the latter is more representative of how much RAM is in use. The sampling occurs just before and just after each module executes as well as just before and after data is written to file. The *art* Module's label and type are shown in the legend. The output is shown twice as it both executes and outputs.

Note that each sample point attributed to a particular *art* Module reflects any memory allocated by all other sources prior to that sample. In addition to plotting the usage at the **start** of an execution (or output) as above, the

change in memory usage between the start and the end of the execution of an *art* Module can be plotted:



The second tools used is Valgrind. In particular its `memcheck` and `massif` sub-tools. The first finds memory errors and obvious leaks and latter can find more subtle leaks. Both proved useful here. The leaks are describe below and in addition a buffer overrun was found and fixed.

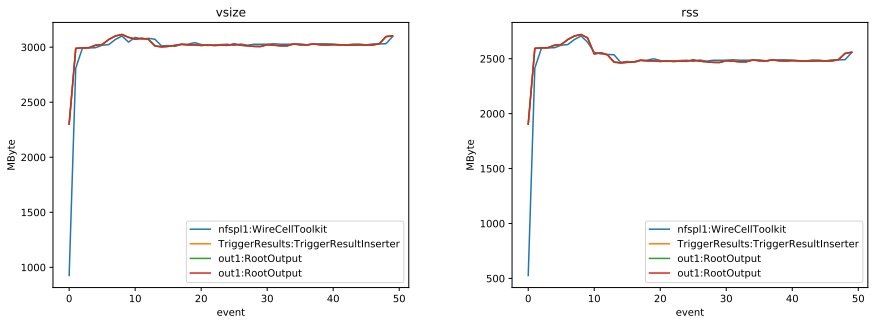
2 Fix memory leaks

Using Valgrind's Memcheck and Massif tools some leaks were found:

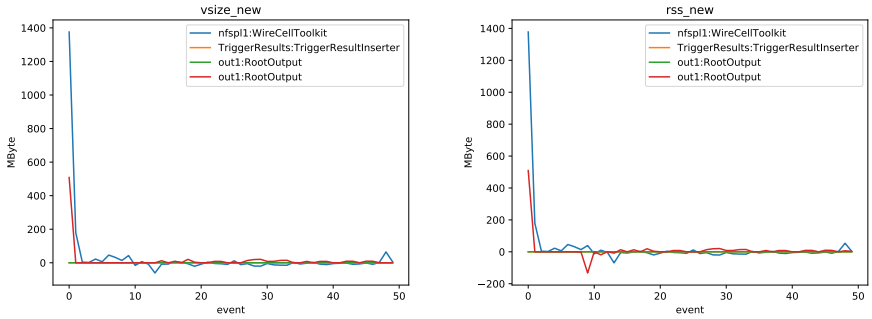
`PeakFinding` a few vector `new []` were paired with a scalar `delete`. This leak was responsible for the bulk of the linear rise.

`ROI formation` a few ROI `delete` were simply neglected. This leak was too small to be found by Valgrind's `memcheck` tool but were found with its `massif` tool.

After the fixes, the memory usage profile is reduced.



The lack of any apparent leak is clear. And, here are the *art* Module's per-execution deltas.



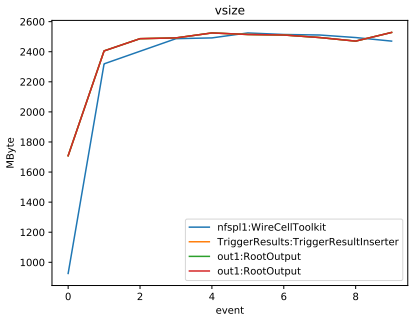
3 Memory usage

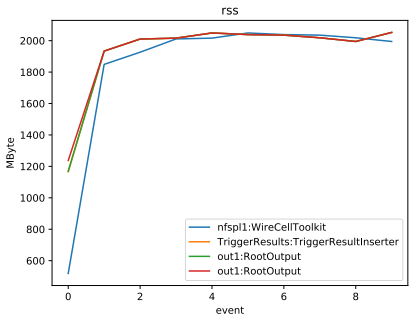
Beyond the leaks, there are two main contributors to using more memory than strictly needed:

1. An "event" in art/LArSoft currently spans 6 APAs (in PDSP). WCT signal processing is inherently per-APA so strictly speaking we should not have to load all APAs in to memory at once. This change is deferred until some coherent plan is made. See section 5 below for some discussion.

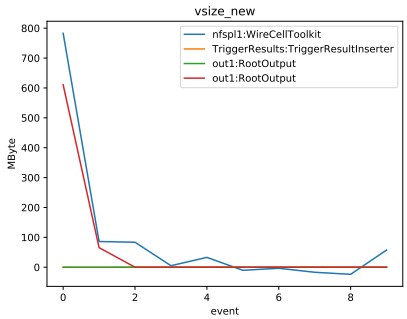
2. The output of SP is inherently "sparse" as the SP ROIs select out relatively tiny portions of waveforms and from only a minority of channels. However, the previous SP code was producing dense, zero-padded waveforms. Mitigating this somewhat on the LArSoft side, the WCT IFrame was being zero-suppressed when converted to recob::Wire.

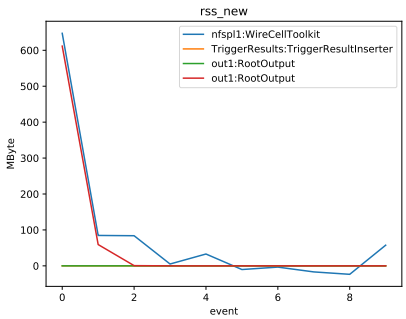
Adding to WCT SP the ability to directly produce a sparse output produces this memory profile. Only first 10 events are shown.



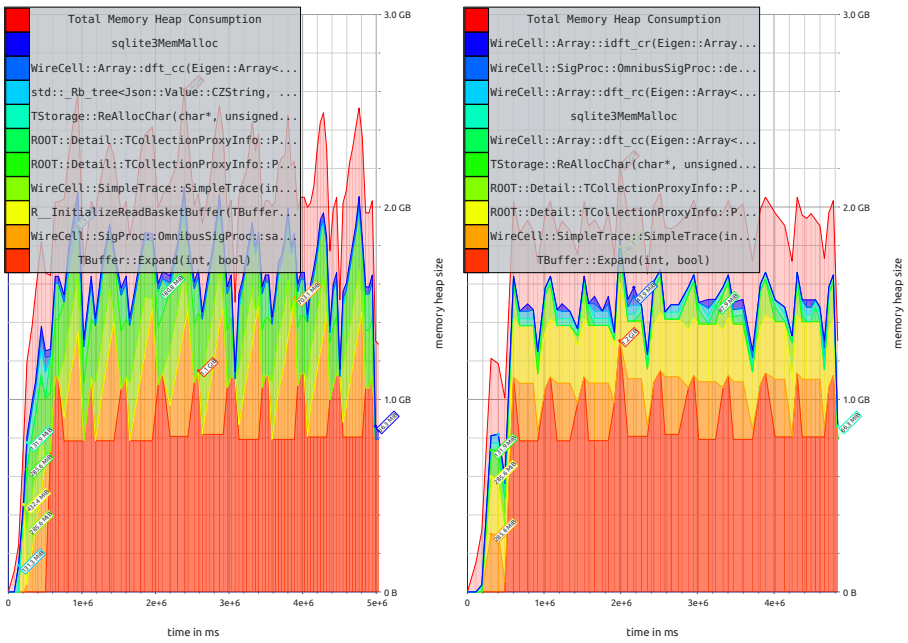


And the deltas.





The Valgrind `massif` tool takes more fine grained samples than does the build in "memcheck" tool of *art*. It apparently measures RSS. Its results may be visualized as in the following two plots which represent ten events before and after enacting sparse SP output.



In both, the orange regions correspond to the internal `IFrame` storage. In both, the underlying red regions are unchanged and attributed to ROOT IO. It turned out that we have always been producing sparse `recob::Wire` despite using a "dense" intermediate `IFrame`. So, making the intermediate sparse understandably doesn't bring any additional benefit.

It is unknown why ROOT requires such a large amount of RAM. Running this job with no output file will reduce the approximately 800 MB "baseline" to near zero. Note, the green `ROOT::Detail::...` in the "before" becomes the yellow in the after.

The peak usage attributed to WCT SP drops from about 800 MB to about 300 MB. Unfortunately the job still broaches the 2 GB limit which keeps it in the category of a 2-CPU grid allocation.

Finally, a large number of small memory users fill in the gap between the highest (blue) entry and the total. The `massif-visualizer` allows their display but they are varied and not low-hanging fruit. After this reduction ROOT I/O is even more the main offender.

4 Summary

GB:	Vsize	RSS	comment
start	3.1	2.7	25 MB/event leak
leak fix	3.0	2.5	varies over time
sparse	2.5	2.1	

The WCT SP memory leaks were untenable and have been fixed. One was blatant and found easily, the other more gentle but located and fixed. Valgrind's `massif` tool attributed at least 600 MB out of 2.5 GB to the dense intermediate WCT `IFrame` objects. Using sparse intermediary versions of these `IFrame` objects brings this usage down to about 300 MB. The primary offender is now (and was before) ROOT I/O.

5 Per-APA Discussion

Significant modifications to WCT as well as *art* and possibly downstream LArSoft modules are needed to achieve further memory reduction. This is helpful for the $N = 6$ APA single-phase protoDUNE detector but is **absolutely required** for the $N = 150$ APA single-phase DUNE FD. Per-APA execution is likely helpful/needed for dual-phase detectors.

To be successful the per-APA execution strategy must be enacted starting at the point of reading data from an input file. This requires *art* to have some input module that iterates over APAs w/in one "trigger", loading only one at a time[1]. Thus each subsequent `art::Event` produced in memory will consist of just one APA and many APAs will be "seen" by the *art* Modules which are all of the same trigger[2].

In general *art* Modules must adapt to this new execution strategy by expecting different APAs. I particular, they must in some way "swap in" appropriate configuration where any is APA dependent. This will likely be an issue for noise filtering and signal processing modules which by their nature tend to erase APA-specific features.

For WCT in particular, there are several "layers" where one may differentiate between APAs.

[1] In the case of parallel processing it may make sense to load more than one APA along different *art* paths.

[2] For supernova neutrino bursts, things get even more complicate as a single APA over many tens of seconds can not be processed in one go. Likely some "framing" procedure must be enacted at the read-in stage. Frame edge boundaries will need special attention.

- if *art* has one path for each APA then one *art* Module per APA may be used (this is generically true for non-WCT modules).
- in a single *art* Module multiple `WCLS` *art* Tools, one per APA may be constructed and the Module may then execute the appropriate `WCLS` *art* Tool when any given APA data is encountered.
- in WCT (assuming *pgrapher* execution engine) appropriate graph structure can accomodate per-APA execution. For example a *fanout* to N pipelines, one per APA may be constructed. Current *pgrapher* support would require a special *fanout* which produces one "real" APA frame and $N-1$ placeholders in order to assure proper sync. A dynamic *fanout* and *fanin* can be constructed to avoid this.

The *art*, LArSoft and WCT experts should form some discussion to better understand this problem and form some plan for how to proceed.