Thoughts on DUNE "framework" Requirements

Brett Viren

January 9, 2020(v0.2)

1 First stage processing of large, dense raw data

The first stage processing of DUNE FD data involves applying software noise filters (NF) and digital signal processing (SP) algorithms. This stage must operate on the full dense array of TPC ADC samples readout from the FD detector single- and dual-phase modules (leaving pixelated for elsewhere) and is expected to achieve an data reduction on order 1000 with high efficiency to retain relevant physics information.

The TPC input to first stage processing is in the form of a set of 2D arrays, one array per "view" and per sub-detector (eg one APA wire plane or one CRO strip set) with on order 1k views per module. The 2D arrays span channels and sample times. The number of channels spanned per view is on order 1000. For any given detector module, the arrays span two very different time periods. A *nominal* readout array spans somewhat more than one maximum drift time and an *extended* readout spans on order 100 seconds. Thus, it may span about 10 k-samples or about 200 M-samples. Due to practical data handling limitations it is likely that nominal readouts and certain that extended readouts from one trigger can not be stored contiguously.

Given this, the software which applies the first stage processing algorithms must satisfy at least the following requirements. Requirements are stated and for each, example use cases are given.

1. It shall be possible to load into system memory data just one view, or just one portion of one view, independently from the loading of data from other views or their portions.

    - The "U" plane view of a nominal SP readout is 10k samples (5ms) by 800 channels and its have NF+SP algorithms are independently from other views. Loading "V" and "W" views into system

memory while "U" is being processed can lead to the RAM footprint exceeding computing facility limits.

- A single view of an extended readout will require on order 10GB and is prohibitively (and needlessly) large for the many of the NF+SP algorithm primitives. Loading in and processing some fraction of this readout is required.

2. The software shall not introduce artificial artifacts.

   - When fragmenting an extended readout in time (as above), a small overlap time must be duplicated at the fragmentation boundaries. This is required in order to account for the practical extent of the electrostatic field response. An overlap on order 100us is ample. Without this overlap, signal efficiency will drop at the fragmentation border.

3. The software shall avoid accumulation of unwanted intermediate data products.

   - First stage processing produces dense intermediate data products which are larger than the input due to inflation of 12 bit ADC to 32 bit quantities. These objects need to be exchanged between algorithms but should not be persisted beyond this scope in order to minimize system memory usage.

2 FD Simulation

Simulating activity and noise in a FD module can require prohibitive amount of computational resources if done naively. Similar fragmentation-related requirements described above for first stage processing apply to simulation. In addition, avoiding expending resources to simulate data that will be immediately discarded can be avoided.

1. It shall be possible to simulate a portion of a FD detector module which may be selected dynamically during processing.

   - The location of ionization activity generated by tracking particles through Geant4 can a'priori determine regions of detector channel and readout time that will be rejected by trigger emulation and analysis selection with effectively 100% certainty. It shall be

possible to avoid substantial computation to simulate signal and noise in these regions.

2. Fragmentation of simulation shall not compromise correctness.

   - In simulating a SNB, an individual interaction may coincide with some arbitrary fragmentation in space or time (such as to satisfy the above). The implementation shall not allow such fragmentation to lead to producing incorrect results.

3 Software Performance

The computing performance needs of DUNE are modest compared to, say, ATLAS/CMS. However, they are substantial compared to past neutrino experiments. DUNE must likely pursue novel computing facility architectures, if not just to seek their promises of high performance then because they may become the norm as traditional "grid style" facilities pursue HPC configurations. This section gives some discussion and requirements if DUNE software shall follow suit.

3.1 Multi-threading and pipeline

To amortize use of limited RAM across multiple cores and to exploit high core multiplicity facilities, development of software to support a multi-threaded execution model has become a goal of many projects.

The granularity of the execution context of such software impacts the number of threads that can be active and more importantly the variability of this number over the run time of the software. If this variability is high it will lead to overall inefficient use of CPU resources. For example, a global synchronization point will reduce the job from utilizing N threads to just one.

One way to avoid this inefficiency it so support pipelining. This is where a new unit of data enters processing while a previous unit is still being processed.

A requirements that can be derived from this discussion: DUNE software must provide means by which allocated threads may remain engaged with high utilization efficiency in the case where different portions of processing employ different numbers of threads and points of synchronization multiple threads may occur.

3.2 Example Study: Wire-Cell Toolkit

One real-world example that motivates the above requirement is the Wire-Cell Toolkit (WCT). This software is used by multiple LArTPC experiments and provides components to perform software noise-filtering, signal-processing (detector response deconvolution plus per-channel signal region-of-interest identification), signal and noise simulation, 3D activity imaging, etc. Although a toolkit, WCT provides a way to organize tools (WCT "components") into a directed graph with execution driven by exchange of data messages. Components may execute together on a single thread or simultaneously on multiple threads, depending on configuration. The graph execution also supports pipelining. Data may enter the graph while previous input data is still working its way through to the output. When points of synchronization in the graph reduce the number of threads that can be active, the idle threads may be utilized to begin processing new data. This continues until all threads are utilized or until all graph edges are "full" of data. WCT graph node and edge multiplicity are typically higher than the number of threads made available.

When WCT is run from an "event processing framework" (eg *art*) this pipelining mechanism is defeated. The results from one "event" must be output by the WCT graph and given to the framework before the "next event" can be delivered into the WCT graph for its processing. This represents a global synchronization point and thus leads to N-1 threads going idle, assuming the framework itself is not keeping those threads active on some other task.