Wire-Cell Toolkit

Architectural Highlights

Brett Viren

October 17, 2022

Topics

Goal: give a taste of some of the technology in and around WCT to perhaps spark your own ideas.

- Basic usage and provided major algorithms.
- Toolkit layers, execution model and multi-threading.
- GPU acceleration.
- AI/ML efforts.
- HPC exploitation.

Major LArTPC algorithms in WCT:

- Simulation: ionization electron drift, induced electrode current using 2D field response model, electronics response, coherent and incoherent noise and ADC digitization.
- Noise filtering: coherent and incoherent, generic and detector-specific covering many of the current LArTPC detectors.
- Signal-processing: deconvolution and signal-region-of-interest using 2D response model.
- 3D imaging: Wire-Cell's namesake tomographic inversion algorithm, L1-regularized ionization electron charge solving.

Clustering, pattern recognition and other advanced algorithms in the MicroBooNE *Wire-Cell Prototype* are in the process of being optimized and generalized to other LArTPC detectors and ported to WCT.

Wire-Cell Toolkit design layers

In order of increasing structure:

1. Utility function and data structure library.
   - User directly codes/uses in ad-hoc manner.
2. Abstract interface base classes.
   - User provides concrete method implementations, application free to aggregate them in any manner.
3. Data flow programming (DFP) paradigm.
   - Dynamic aggregation of implemented interfaces provided in shared library plugins is driven by user configuration and results in an executable "data flow graph".
4. Command line or application interfaces.
   - Embed data flow graph or smaller WCT constructs into some larger program.

Next, some details about 4 then 3 $\longrightarrow$

Application-level interfaces to WCT

Provided command line interface

```
wire-cell --help
wire-cell [command and config options] -c mycfg.jsonnet
```

- Jsonnet is a purely-functional programming language using a super-set of JSON syntax to write programs that create data structures.
- Job may load "core" WCT plugins and any WCT "user" plugins.

art "module" and "tool" interfaces

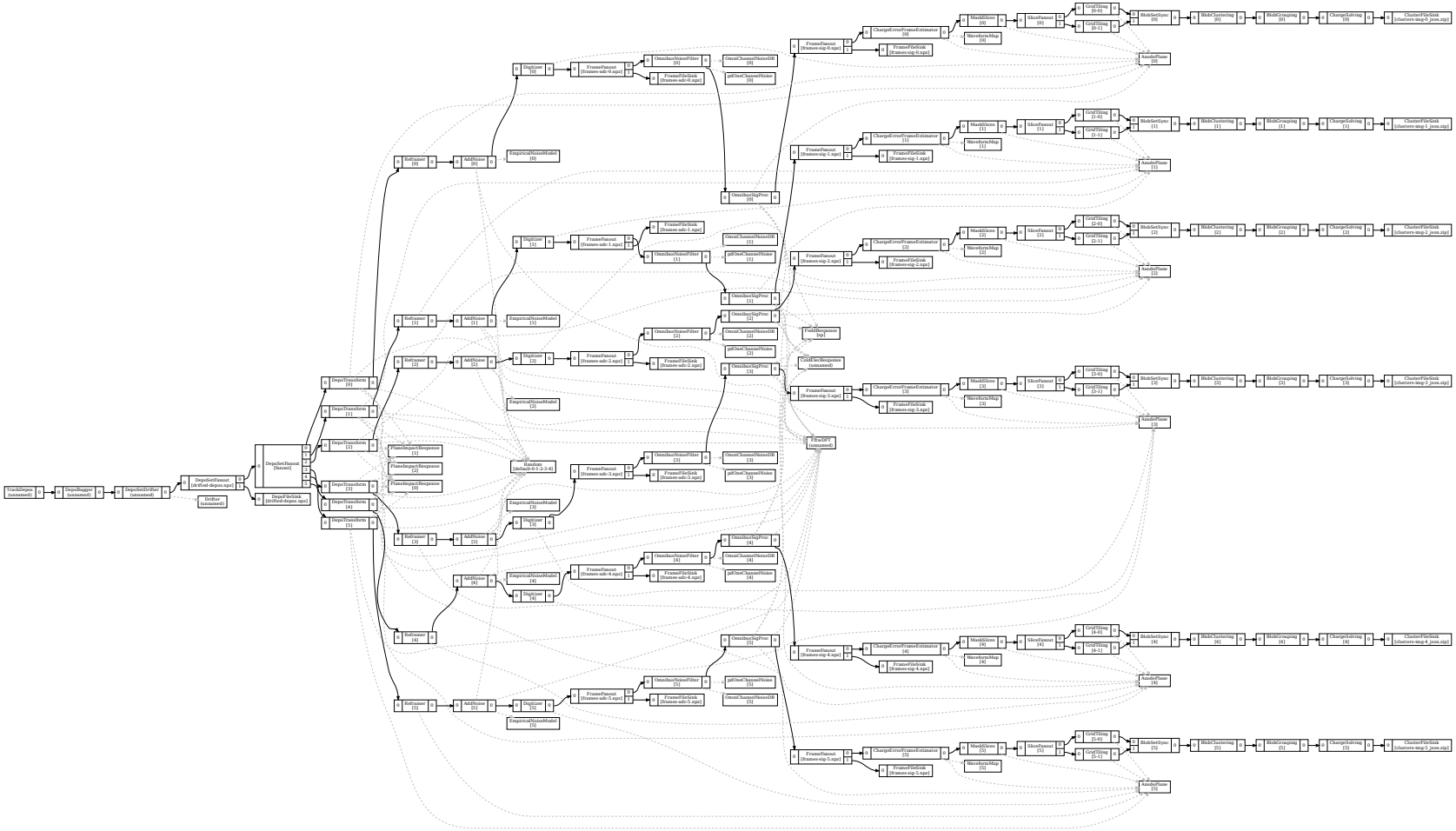
- The LArSoft package `larwirecell` provides *art* components `WCLS_tool` and `WireCellToolkit_module` and WCT components that convert between WCT and LArSoft data products.
- Design is general to any WCT + *art* job, dynamically extensible, loose coupling between *art*, LArSoft and WCT versions.
- WCT / *art* configuration interface is exactly analogous to the `wire-cell` CLI.
  - Provide Jsonnet file name and "command line arguments" in *art* FHiCL.

WCT I/O support

- Python Numpy arrays and structured JSON via container streams.
  - Supported formats: `.zip/.npz`, `.tar`, `.tar.gz`, etc.
  - WCT Python / CLI converters to Paraview/VTK and Bee/JSON.
- ROOT histogram format for validation/debugging.
  - (ROOT is not a core dependency)
- WCT object transport via ZeroMQ.
- HDF5 file format.
  - New WIP: reading DUNE DAQ files, B. Chowdhury, ANL.
- WCT/LArSoft object converters.

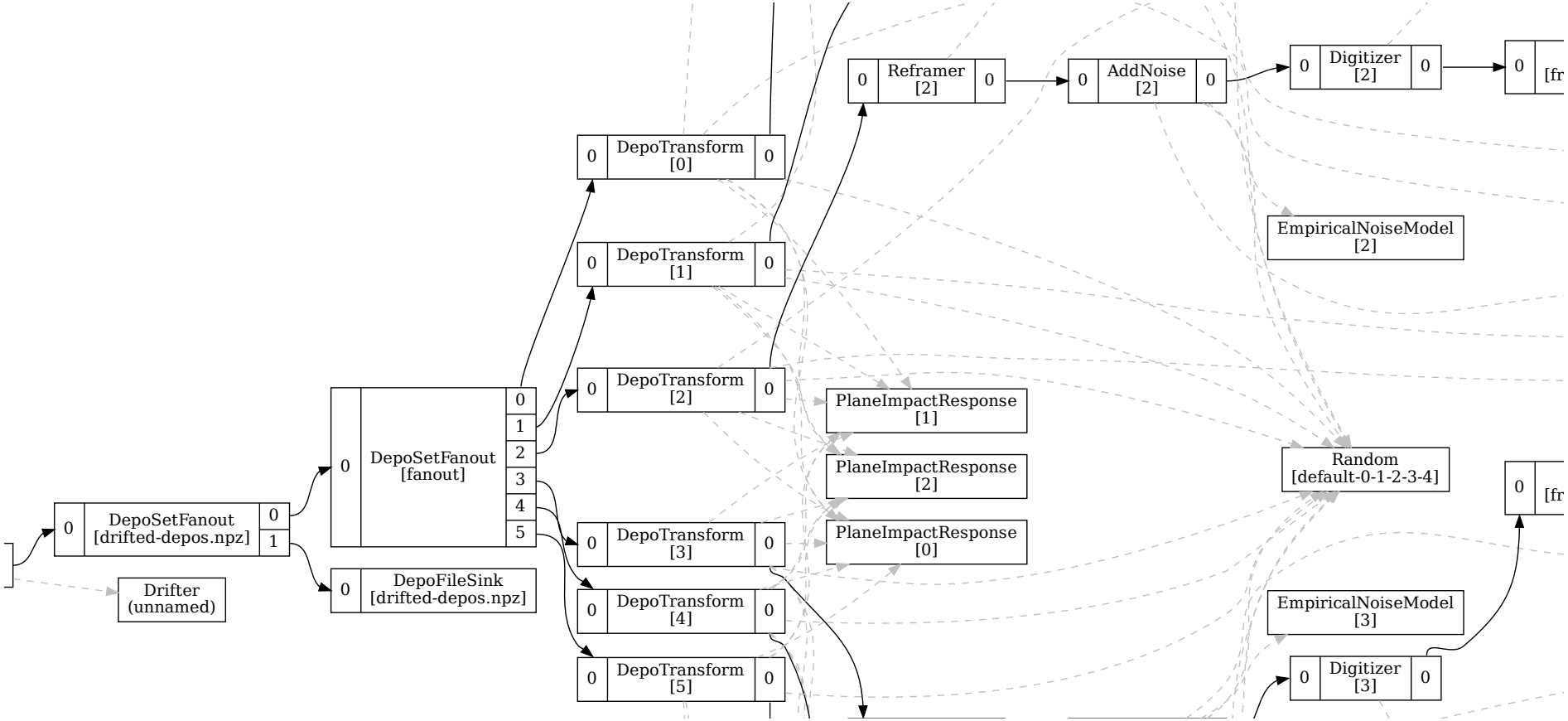
WCT *data flow processing* execution paradigm naturally supports parallel file streams and the mixing of different file streams. This is important for reading DUNE FD DAQ files, simulating variety of physics, production of sim/data "overlay", etc.

WCT jobs are organized as *data flow processing* graphs.



Data flow graph of WCT job performing simulation + signal processing + 3D imaging.
Major graph structure is six parallel pipelines, each for one APA of ProtoDUNE-SP.

Port graph representation



Flow graph vertices/nodes have *ports*:

- Each port passes data of a particular type and direction.
- Input/output *port sets* have well defined cardinality, may pass data of a variety of type.
- Categories: **synchronous**: *source*, *sink*, *functional*, *split* / *join*, *fanin* / *fanout*, **asynchronous**: *branch* / *merge*, *hydra*.

Flow graph edges: data objects sent between ports

A WCT flow graph **data object** is …

- … an implementation of abstract interface class.
  - Exposes only methods, may implement "smarts" like lazy-loading
    - ★ (saved the day for PDSP memory usage).
  - Interface defines the data type for flow graph *ports* and *edges*.
- … held as **const** and **shared** pointer:
  - Automated memory cleanup, consistent view of data
  - No data races in multi-thread context.
  - Re-aggregation of sub-data is fast and conserves memory.

Multi-thread (MT) graph execution

WCT flow graph provides two data flow graph *execution engines*:

`PGraph`

- Custom, single-threaded (ST) execution based on a topological sort of the graph and node execution strategy that minimizes memory usage.

`TbbFlow`

- Multi-threaded (MT), driven by TBB `flow_graph`
  - TBB's "push/pull" execution strategy optimizes CPU utilization over memory conservation.
- Threading issues are not exposed to developer of nodes, serial coding patterns are fine.
  - MT use in node is also allowed, but best to use TBB to share thread pool.
- Synergy with *art* which also uses TBB.

GPU acceleration

- WCT is one of two test cases for HEP/CCE-PPS R&D.
  - (next slide)
- WCT provides an abstract FFT engine, choice at run-time of:
  - FFTW3 (CPU), Torch (CPU or GPU) and CUDA (GPU).
- Execute ML inference models via Python'esque "torch script".
  - May configure `libtorch` to execute on CPU or GPU.
- GPU resource management
  - Nominally, MT WCT graph nodes may submit unbounded number of GPU tasks. WCT protects from GPU resource exhaustion via MT-safe semaphore.

HEP-CCE/PPS and WCT

Evaluate methods to parallelize existing serial code in terms of software development **practices** and computing resource **performance**.

- WCT-related: Haiwang and our colleagues in BNL CSI.
- Technologies considered: CUDA, Kokkos, SYCL and OpenMP.
- Focus on WCT **signal simulation** with poses a mix of "trivial" and "difficult" algorithms to parallelize.
  - Eg: Kokkos achieves 18x speed up needing less than one second to simulate one APA-readout.

More info:

- Kokkos-based sim: https://github.com/WireCell/wire-cell-gen-kokkos
- Kokkos results: https://arxiv.org/abs/2203.02479
- CUDA + Kokkos results: https://indico.cern.ch/event/948465/contributions/4323675/

AI/ML efforts in the WCT arena

- **DNN ROI** increases signal processing efficiency for elongated tracks.
  - Otherwise, nearly invisible due to overlapping bipolar waveforms.
  - Uses and drove WCT's "torch script" support.
- **LS4GAN** to minimize, quantify and propagate systematic differences between simulated and real detector data.
  - First, use WCT simulation with two different field response models. Then, map between simulated and real detector domains.
  - Training is **unpaired** but infers **paired** "nominal" and "more real" simulated events: $\{E_i, E'_i\}$.
  - Given an arbitrary downstream reco/ana chain: $\mathcal{F}(\cdot)$ then an example simulation systematic estimate may be $\frac{1}{N}\sum_i^N |\mathcal{F}(E_i) - \mathcal{F}(E'_i)|$.
- **GNN**, "blob deghosting" based on graph neural network
  - WCT simulation, signal-processing and imaging for training.
  - Determine which elements of WC tomographic solution likely contain no true charge.
  - Expected to be extensible to other feature categorizations.

HPC: distributed-WCT

A grand-challenge: triggered, prompt processing for SNB pointing:

1. Long-lived service which is resident at a friendly HPC, directly accepting a DUNE SBN trigger (eg, sibling of SNEWS 2.0).
2. SBN trigger results in $\mathcal{O}(10k)$ core jobs starting immediately
   - possibly preempting running jobs! (politically impossible?)
3. Jobs receive data rich in SBN activity directly from a DUNE FD DAQ service.
   - readout chunks corresponding to DAQ SNB "trigger-activity" regions.
4. Run WCT NF+SP+imaging+clustering, get high-fidelity direction information.
5. Send results to global community, eg via SNEWS 2.0.

Latency goal: results to community w/in **10 minutes** of trigger.

- Interesting challenges in many areas: DAQ, HPC, software, networking, protocols, resource politics.

More details available in the (unfortunately declined) SciDAC-5 proposal.

Wire-Cell related documentation and source

Main page, documentation, blog and links:

- https://wirecell.bnl.gov/ or https://wirecell.github.io/

Core Toolkit C++ Source:

- https://github.com/wirecell/wire-cell-toolkit

Python utility helpers:

- https://github.com/WireCell/wire-cell-python

Detector support (large config files):

- https://github.com/WireCell/wire-cell-data

Spack recipes for full-stack software builds:

- https://github.com/wirecell/wire-cell-spack

Support for full-stack Docker/Singularity containers:

- https://github.com/WireCell/wire-cell-containers

Wire-Cell Prototype:

- https://github.com/BNLIF/wire-cell

Wire-Cell Toolkit / LArSoft interface:

- https://github.com/LArSoft/larwirecell