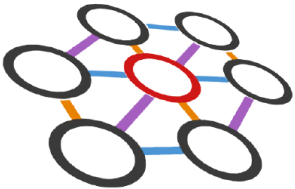
Threads of parallelism with the Wire-Cell Toolkit

Brett Viren for the Wire-Cell team



LArSoft Multithreading and Acceleration Workshop
March 2023

Topics

Overview of the Wire-Cell Toolkit (WCT).

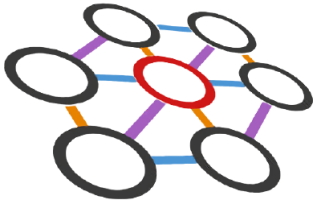
- With a focus on multithreading (MT).

Describe the main ways we use WCT.

- With *art* + LArSoft or “stand alone”.
- Implications on for exploiting MT.

MT strategies to key LArTPC software problems.

- How WCT can contribute.

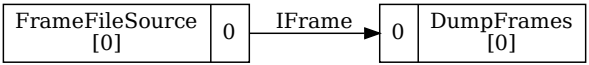


Wire-Cell Toolkit (WCT) in a nutshell

WCT produces a set of C++ shared libraries providing:

- Software development patterns such as: **user configuration, plugins, component factory, interface classes, code-aggregation methods, and support for application, library and package building**.
- A collection of general-purpose and LArTPC-specific **utility functions**.
- A taxonomy of algorithm and data **interfaces** (abstract base classes) relevant to LArTPC data processing and which are the building blocks of the toolkit architecture.
- A **multithreaded** (MT) and **single-threaded** (ST) *data flow programming paradigm* (DFP) **graph execution engine** providing WCT's primary *execution model*.
- Key **LArTPC algorithms** and the **detailed configuration** to apply them to most of today's major and prototype LArTPC detectors.
- A **research and development platform** for network-distributed applications, hardware-acceleration, high-performance computing and (production-quality) AI/ML inference.

WCT implements data flow programming (DFP) paradigm



In general, DFP aggregates code units as **nodes** in a **graph** sharing data over **edges**.

- Graphs are **executed**: node body methods are run, typed data is consumed and produced on node **ports** and transferred between ports along **edges**.
- A WCT nodes, ports and data are strongly typed via C++ **interface** classes.
- A nodes may be an `IConfigurable` to receive user configuration.

The DFP graph forms the basis for WCT's primary MT mechanism.

WCT provides two graph execution engines

`Pgrapher` an ST engine which **minimizes memory usage**.

- Executes nodes in reverse order of topological sort.
- An emergent "peaked wave" of data flows through the graph.

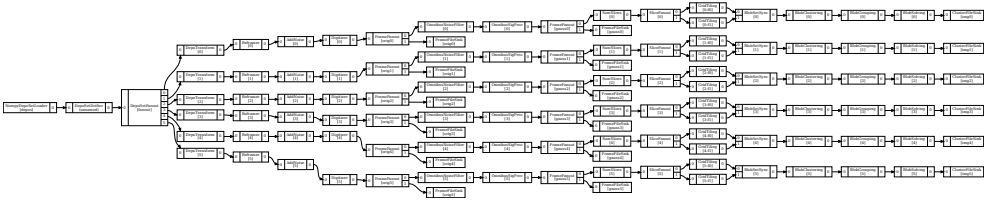
`TbbFlow` an MT engine which **maximizes CPU utilization**.

- Executes nodes concurrently up to a configurable maximum thread count via TBB `flow_graph`.
- A node will not execute concurrently with itself.

Both engines receive identical user-configuration to build the graph.

- A trivial configuration change is needed to switch between engines.

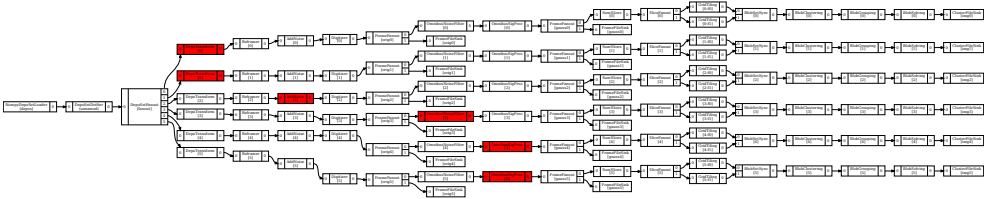
Larger graph example: sim, sigproc, 3D imaging, file I/O



6 APAs of ProtoDUNE-SP reflected into 6 pipelines of nodes.

The `TbbFlow` engine produces two types of **emergent MT patterns** $\longrightarrow$

Transverse parallelism

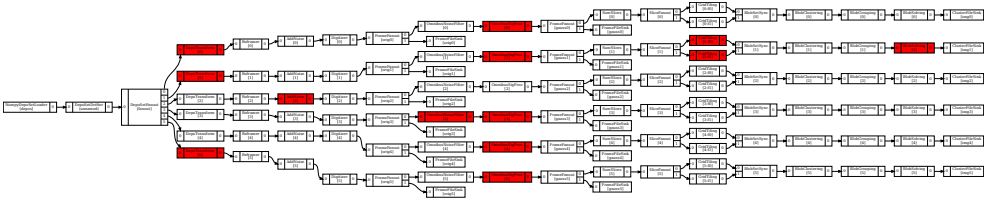


Example of 6 cores being active (red nodes).

- One node in each pipeline executes due to the fan-out.
- We exploit this with algorithms that are fine-grain “across” the detector.
  - Note 6-way per-APA and 12-way per-APA-face.
- High core utilization up to number of major pipelines.
  - Trivial **exploitation of "extra" cores** (over)allocated just to get their memory.
  - More cores than pipelines will be wasted.

But $\longrightarrow$

Longitudinal parallelism

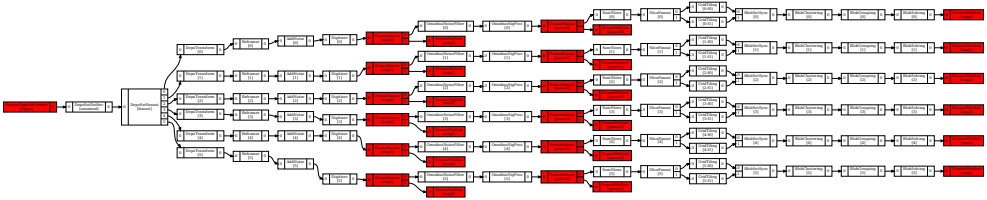


Example of 13 cores being active.

Multiple nodes in a single pipeline may execute in parallel with MT when **new data is input while older data still "in flight"**.

- However, free flowing input may fill graph, exhaust memory.
  - Simple solution: limit total input size to be “small enough”.
  - Future: “dynamic backpressure”, designs exist, need implementations.
- WCT file sources nodes will induce longitudinal parallelism.
  - The *art* event-based execution model will not (transverse parallelism, yes).

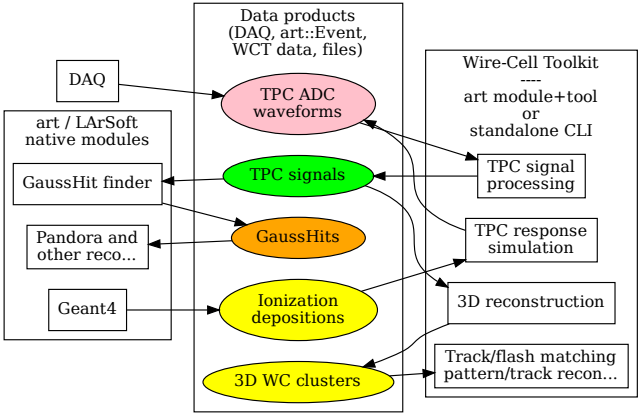
Parallel data I/O patterns



Example of nodes providing parallel file I/O.

- Input: multiple input (here just one) may *asynchronously fan-in* to implement an *event mixing* pattern.
- Taps: any graph edge may be “broken” to insert a *tap pattern* allowing pass-through and consumption of data.
- Output: pipelines may indivdually terminate in a sink, eg producing per-APA files here.

Wire-Cell Toolkit usage context



Consumers: shared? shared LArSoft Wire-Cell

Command line WCT

```
$ wire-cell --help
$ wire-cell -t 16 -l my-job.log -L debug \
            -A detector=uboone -c my-job.jsonnet
```

The CLI lifetime:

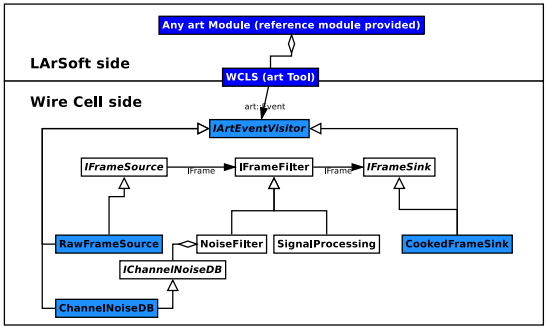
- set number of threads, configure logging streams,
- injects command-line options to and compiles and interprets the configuration,
- loads plugin libraries, instantiates named objects, applies their configuration,
- executes one or more special “app” objects (eg the graph execution engine).

Beyond that, the CLI is totally “policy free” with no preconceived data nor execution models and the entire behavior is fully determined by configuration.

Integration of WCT with *art* and LArSoft

Integration code lives in `larwirecell` under the `larsoft` umbrella, depending on `wirecell`.

- `WCLS`, an *art* tool, calls the main WCT CLI object with info from FHiCL.
- `WireCellToolkit`, provides a minimal *art* module using a `WCLS` tool.
- `IArtEventVisitor` provides one part of a multi-interface WCT component.
- Various WC/LS implementations are included which adapt data products and services.



Strategy for large-scale simulation

In the past we output WCT sim ADC waveforms to `art::Event` as `RawDigit`'s.
But:

- Monolithic ADC data products are not feasible beyond ProtoDUNE scale (also not required!).
- The only(?) consumer of ADC in production simulation is WCT signal processing.

Elements of solution:

- **Combine WCT sim + sigproc** in one graph, **ADC's become transient** inside WCT.
- Apply WCT sim + sigproc "**sparse mode**", process only "interesting" APAs.

But, there are special cases needing simulated ADCs:

- Access full statistics? Run as WCT component during production processing.
- Small statistics? Run as special job in high-memory, output ADCs to `art::Event`.

People should understand if there are show stoppers here!

Strategy for signal processing

For sim, combine as just described.
But for **real detector data**:

- Expect dense, per-APA ADC files from DAQ.
- Single-APA files: **no transverse parallelism**.
- Using *art* for input: **no longitudinal parallelism**.
  - WCT must perform the input benefit over-allocated cores.
- Use art + LArSoft for **output** to benefit from provenance features.

Strategy for merging/mixing

WCT supports asynchronous *branch* and *merge* pattern.

- Each branch/merge node type is developed custom to perform some task driven by data.
- A merge can be fed by parallel file inputs. (merge all fragments from common trigger).
- A branch can feed parallel file outputs. (send fragment i to output stream `file_$i.tar.gz`)

Merging across APA's not required at **ADC** nor **signal** data tiers but perhaps needed at `GaussHits` and WCT 3D cluster data tiers (or soon after).

- These data tiers are vastly smaller than TPC ADC and so whole-detector merge will not suffer prohibitive memory overhead in making a monolithic output.

Strategy for GPU acceleration

sim Kokkos/GPU version of WCT sim for $18\times$ speedup ($\approx$ 1 sec / APA)

- Haiwang Yu, HEP-CCE/PPS

sigproc DNNROI AI/ML inference, FFTs.

- Most still CPU-only. Porting to GPU, possible but challenging.
- Promising R&D: replace parts with AI/ML inference on GPU.

Numerology: sim/sigproc needs ≈ 100 CPU cores to feed 1 GPU at $\approx 100\%$.

- HPC has 4-8 GPU / compute-unit (box), 50-100 CPU core.
  - Need much more CPU $\rightarrow$ GPU porting to fully utilize this GPU density.
- GRID has 0-1 GPU / box, 10-20 CPU core.
  - Simply live with $\approx 10\%$ GPU utilization?
  - WCT/ZIO/ZeroMQ GPU-as-service to share one GPU across ≈ 10 boxes.

Summary

Wire-Cell Toolkit:

- executes many ST-coded nodes in parallel across a MT'ed data flow graph.
- can run stand-alone or as a part of *art* + LArSoft.
- has GPU acceleration today, opportunities for more and methods to adapt to a range of GPU/CPU ratios.
- can flexibly adapt to different hardware resource limitations.
- provides parallel I/O patterns to mitigate memory overheads, avoid data monoliths and provide a number of useful functional features.

$\mathcal{FIN}$

Prompt signal processing:

- Want to run signal processing on raw data while it resides on **tape input buffer disk**.
- Save money and time to avoid full read-back ADC from tape.
- Signals are the input to “everything”, are smaller and can reside on disk for frequent re-reading.
- Enables ad-hoc prompt data access, release testing, calibration, quality monitoring.

Strategy for prompt supernova neutrino burst pointing

Locate source of SNB, promptly tell astro, better point telescopes.

- Ideas exist to use DAQ Trigger Primitives and raw ADC
  - All are limited in accuracy and precision due to bipolar nature of induction and we must use **signal processing to achieve best results**.
  - 10 minutes still useful/achievable but Super-Kamiokande already beats this.
- Run sigproc on **shared or dedicated HPC** with CPU+GPU.
  - DUNE DAQ sends SNEWS input message also to service on HPC.
  - HPC launches large job $N_{threads} \approx \mathcal{O}(10\text{k})$ over many processes.
  - DUNE DAQ streams sparse TPC data continuing activity directly to service on HPC. Must have priority over the $\approx$ 10 hour egress of full SNB data.
  - Promptly stream results to SNEWS 2.0

BNL proposed this as part of a recent SciDAC. Proposal was complimented but declined. Now up to DUNE / future funding if we want to contribute in this manner.