# Wire-Cell Toolkit

A Snowmass 2021 Letter of Interest in Experimental Algorithm Parallelization (CompF1)

Brett Viren for the Wire-Cell team

July 13, 2020

## 1 Overview

As the name implies, the Wire-Cell Toolkit (WCT) takes a bottom-up approach as contrasted with the top-down approach that is usually identified with software frameworks. Instead, the WCT C++ libraries provide modular layers of functionality giving an application developer choices in how much or how little to adopt.

WCT infrastructure is general purpose and may be used in many domains. Due to its birth in the domain of liquid argon time projection chamber (LArTPC) detectors including MicroBooNE[1], those in the SBN program[2], DUNE{Abi:2020loh} and R&D on novel LArTPC readout technologies[3] it has many algorithm-level "batteries included" relevant to LArTPC detector R&D and operation. These include state-of-the-art LArTPC noise and signal simulation, digital noise filtering[4] and signal processing[5] and an implementation of a 3D tomographic method[6] to reconstruct TPC ionization patterns (which provided the original "Wire-Cell" name).

For the remainder of this letter, the main aspects of the toolkit are described at a high level. Many more details may be found online[1].

## 2 Interfaces

Strongly influenced by Gaudi[7], a core layer of WCT consists of a set of pure, abstract base classes, aka *interfaces*. A plugin/factory method allows implementations of interfaces to be dynamically constructed and later located given a concrete type name and optional instance name.

There are two distinct categories of interfaces in WCT: *components* and *data*. Components make up the "verbs" and data the "nouns" of a WCT application vocabulary. Component methods that accept data interfaces do so via shared pointer such that all data outside the context of the method body is immutable and data memory is automatically and properly managed.

## 3 Component composition and execution

The developer may chose how to compose component classes into an application. For example, one may "hard-code" a composition of concrete component and data implementations. This is inflexible and more effort than the alternative but is often used in unit tests.

The primary composition idiom that the toolkit supports is that of a *data flow graph. The components form graph nodes which are connected through identified /ports* (associated with component

[1] `https://wirecell.bnl.gov/`

method arguments) by graph edges that represent the passing of data of specific type. The graph construction driven by user configuration in a way that assures graph completeness and type-safety.

The execution of components in a WCT data flow graph is itself performed through an abstract interface. This allows different graph execution engines to be developed and WCT provides two.

The first engine (`pgrapher`) provides single-threaded execution and a graph traversal policy which minimize memory usage. It is preferred in jobs where non-WCT portion of the job is substantial and does not utilize multiple threads.

The second engine (`tbbflow`) makes use of TBB[2] flow graph library to execute a number of WCT components concurrently. The graph traversal policy of this engine allows for data pipelining. That is, multiple "events" may be "in flight" through the graph at any given moment which can help to avoid idle threads.

This component-level parallelism allows WCT to fill in a "mid-grained" gap in the granularity spectrum of parallel processing. That is, individual WCT components may be single-threaded or internally utilize GPU, SIMD or loop-level parallelism, the `tbbflow` engine allows for component-level parallelism and of course, multiple WCT jobs may run in the usual "embarrassingly parallel" manner.

# 4 Configuration

At the C++ level, configuration objects are currently represented as JsonCPP[3] objects. The application may manually construct a configuration objects and dispatch it to the appropriate instance of a "configurable" or the WCT configuration manager may be used. It supports loading compressed or plain JSON files and files written in the Jsonnet[4] data templating language.

WCT provides Jsonnet support code. Graph construction functions allow defining subgraphs which may then be used as apparently simple nodes to build larger subgraphs until a final graph may be represented as a single node. Jsonnet itself is a LISP-like functional language with many features such as modules, iteration, comprehensions, branches, etc which allow complex configurations to be easily constructed.

# 5 Application interface

The WCT provides a high-level interface called "app" which represents high-level execution. It is the job of a WCT app to "do" something, typically by composing components and executing them. The two data flow graph engines described above are each implementations of this interface. A number of "apps" may be run in one full application although typical user jobs employ only one.

At the penultimate highest layer is the WCT `Main` component. It implements a configuration protocol using simple strings, such as may be provided to `main()` and delegates to plugin and configuration managers and then executes any apps.

At the highest level of the toolkit are external interfaces. The first is the command line interface program called `wire-cell` which merely constructs a `Main` and passes to it the command line arguments. Through this program the user may control WCT logging facility (based on `spdlog`[5]), indicate plugins and most importantly provide configuration files and additional configuration parameters.

[2] `https://www.threadingbuildingblocks.org/`

[3] `https://github.com/open-source-parsers/jsoncpp`

[4] `https://jsonnet.org`

[5] `https://github.com/gabime/spdlog`

Another external interface connects to the *art*[8] framework and the LArSoft[9] data model. It exists as part of the LArSoft project in the `larwirecell` package. It provides an *art* "tool" class which delegates to the WCT `Main` class where the string arguments are provided to the tool by *art* configuration facilities. The tool may be used directly and in addition a general purpose *art* module is provided which connects the tool to the larger *art* execution cycle. The `larwirecell` package also includes WCT components which depend on LArSoft data classes and which provide conversion to and from WCT data interfaces.

## 6 Packaging

The core portion of WCT software is provided as a git repository[6] with a number of subpackages, each providing a shared library. Dependencies between these packages and with third party software is strictly managed. The build system is based on Waf[7] which allows variant builds to produced which selects a subset of the dependency graph. For example, core functionality does not depend on ROOT[8] while TBB is only required for the multi-threaded graph execution engine.

Waf is also exploited to provide for so called WCT "user packages" (WCUP). These are simple to produce and skeletons may be generated from templates[9]. A WCUP closely resembles the core subpackages and this enables novel work to be started easily later if it becomes accepted it may be easily merged into the core repository.

## 7 Ongoing Development

The Wire-Cell team have solved crucial infrastructure and algorithm level problems in the current WCT. Novel development continues in both of these directions.

Various lines of development at the algorithm level are ongoing. The existing signal processing components are being factored to reduce their granularity to increase parallelism. A Deep Learning model has been developed to increase the signal processing efficiency for the very challenging case where ionization tracks are near normal to LArTPC wire planes. For both signal processing and simulation, effort which is part of HEP CCE-PPS[10] has ported computational intense algorithms for offload to GPU via direct CUDA implementations and more recently device-portable layer based on Kokkos[11].

The WCT infrastructure development is attacking the scaling problem that GPU offload creates. Depending on the amount of computation offloaded to the GPU the overall job may require 100s or 1000s of cores to avoid GPU idling. This has motivated development of multi-process and multi-host applications which communicate via high performance message passing. The ZeroMQ-based ZIO[12]

[6] `https://github.com/wirecell/wire-cell-toolkit`
[7] `https://waf.io`
[8] `https://root.cern`
[9] `https://github.com/brettviren/moo`
[10] See dedicated LOI regarding HEP CCE-PPS.
[11] `https://github.com/kokkos/kokkos`
[12] `https://brettviren.github.io/zio`

## References

[1] R. Acciarri et al. "Design and Construction of the MicroBooNE Detector". In: *JINST* 12.02 (2017), P02017. DOI: 10.1088/1748-0221/12/02/P02017. arXiv: 1612.05824 [physics.ins-det].

package is used to develop GPU and file I/O services that may be as asynchronously accessed by multiple threads, processes and hosts. The ZIO library is also a candidate to supply self-trigger infrastructure to the DUNE far detector data acquisition system and thus its use in WCT represents a paradigm shift where large-scale offline applications begin to resemble highly integrated, large scale online distributed data processing systems.


[2] M. Antonello et al. "A Proposal for a Three Detector Short-Baseline Neutrino Oscillation Program in the Fermilab Booster Neutrino Beam". In: *arXiv:1503.01520* (2015). arXiv: 1503.01520 [physics.ins-det].

[3] B. Baibussinov et al. "Operation of a LAr-TPC equipped with a multilayer LEM charge readout". In: *JINST* 13.03 (2018), T03001. DOI: 10.1088/1748-0221/13/03/T03001. arXiv: 1711.06781 [physics.ins-det].

[4] R. Acciarri et al. "Noise Characterization and Filtering in the MicroBooNE Liquid Argon TPC". In: *JINST* 12.08 (2017), P08003. DOI: 10.1088/1748-0221/12/08/P08003. arXiv: 1705.07341 [physics.ins-det].

[5] C. Adams et al. "Ionization electron signal processing in single phase LArTPCs. Part I. Algorithm Description and quantitative evaluation with MicroBooNE simulation". In: *JINST* 13.07 (2018), P07006. DOI: 10.1088/1748-0221/13/07/P07006. arXiv: 1802.08709 [physics.ins-det].

[6] Xin Qian et al. "Three-dimensional Imaging for Large LArTPCs". In: *JINST* 13.05 (2018), P05032. DOI: 10.1088/1748-0221/13/05/P05032. arXiv: 1803.04850 [physics.ins-det].

[7] G. Barrand et al. "GAUDI - A software architecture and framework for building HEP data processing applications". In: *Comput. Phys. Commun.* 140 (2001), pp. 45–55. DOI: 10.1016/S0010-4655(01)00254-5.

[8] Robert K. Kutschke. "art: A framework for new, small experiments at Fermilab". In: *J. Phys. Conf. Ser.* 331 (2011). Ed. by Simon C. Lin, p. 032019. DOI: 10.1088/1742-6596/331/3/032019.

[9] E.L. Snider and G. Petrillo. "LArSoft: Toolkit for Simulation, Reconstruction and Analysis of Liquid Argon TPC Neutrino Detectors". In: *J. Phys. Conf. Ser.* 898.4 (2017). Ed. by Richard Mount and Craig Tull, p. 042057. DOI: 10.1088/1742-6596/898/4/042057.