Calorimeter Waveform Processing Module

Timothy Rinn

September 2022

Abstract

This internal note discusses the basics and implementation of the calorimeter waveform processing module. The module is designed to be passed the measured waveform for any number of channels and returns the amplitude, time and pedestal information for each channel. The current implementation enables both traditional template fitting and onnx model processing, though additional functionality can be added down the road.

Contents

1 Code Location

The key piece of code discussed in this note is located at:
https://github.com/sPHENIX-Collaboration/coresoftware/tree/master/offlineoffline/packages/CaloReco/CaloWaveformProcessing.cc (.h)

The relevant wrapper function for the implementation of onnx within this framework can be found at:
https://github.com/sPHENIX-Collaboration/coresoftware/tree/master/offline/framework/phool/onnxlib.C (.h)

2 Module Use

2.1 Basic description

The module is designed to act as an interface between the user (reconstruction code) and the various waveform processing algorithms. By default, after initialization, it is designed to be passed a vector of vectors of floats, where the outer vector corresponds to a list of channels (arbitrary size) and the inner vector corresponds to the waveform for a given channel. It then returns a vector of vectors, in the same order, containing the extracted amplitude, arrival time, and pedestal value.

2.2 Initialization

During the init stages of a Fun4All module one must initialize the waveform fitting module with the desired parameters. Below find listed some of the key commands that can be used to constrain the behavior of the module. The effect of these commands is summarized in the following sub-subsections.

1. set_processing_type(CaloWaveformProcessing::process)
2. set_template_file(const std::string)
3. set_model_file(const std::string)
4. set_nthreads(int)

If these commands are not set by the user the module will default to utilizing template fitting in single threaded mode using templates produced from test beam data.

After setting the parameters for the module one simply needs to initialize the module by calling *initialize_processing*(). This will cause the module to load the relevant files and prepare for signal processing.

2.2.1 set_processing_type

This command is designed to allow the user to specify the method which they would like to use to process the waveform. As of writing the module is designed

for both a multithreaded template fitting procedure as well as using ONNX models. These can be selected by passing the following arguments:

- Template Fitting: CaloWaveformProcessing::TEMPLATE
- ONNX Models: CaloWaveformProcessing::ONNX

The ONNX interface is behind the scenes controlled by an ONNX wrapper which interfaces with the ONNX package. This is described in section 3.

2.2.2 set_template_file

This command is designed to allow the user to specify a file in the calibrations repository to be read in for use in the template fitting. By default the module will load the current template file from the conditions database.

2.2.3 set_model_file

This command is designed to allow the user to specify a file in the calibrations repository to be read in for use in ONNX processing. By default the module will load the current ONNX file from the conditions database.

2.2.4 set_nthreads

This command controls the number of threads to use during the multi threaded template fitting procedure. By default the module operates in single threaded mode, however any number of threads can be used. Through testing on sPHENIX fully occupied production hardware it was seen that beyond 4 threads no time benefit was gained by increasing the number of threads further. Though this may change based on the types of jobs running, it is a fair rule of thumb.

2.3 Processing waveforms

The waveform processing itself should get called during the process_event stage of a Fun4All module. The interface at this stage is quite simple and controlled by a single call.

- process_waveform(std::vector< std::vector<float> >)

This call takes as argument a vector of vectors of floats, with the outer vector consisting of any number of channels the user desires to process in a batch and the inner vector consisting of the waveforms for each of those vectors. This enables flexibility in how the module is implemented into a respective Fun4All code as it permits individual calls for each channel or batching of the entire 24,576 channels of the EMCal into a single operation.

These waveforms are packaged and passed to the corresponding processing methods as defined during the initialization stage (I.E. template fitting or ONNX model). These methods characterize the waveform into simply 3 parameters: pulse amplitude over pedestal, pulse time, and pedestal.

Finally, the function returns a std::vector < std::vector <float > > where each entry in the outer vector corresponds to the same channel as was passed to the module and the inner vector contains (in order) the extracted amplitude, pulse time, and pedestal. In the future, additional parameters may be appended to this vector to provide quantification of fit quality.

3 ONNX wrapper

In order to process ONNX models in a convenient manner a simple wrapper is used. These can be found at:
https://github.com/sPHENIX-Collaboration/coresoftware/blob/master/offline/framework/phool/onnxlib.cc (.h) The wrapper contains two main pieces of functionality, an initialization method and a processing method.

1. onnxSession(std::string &modelfile)
2. onnxInference(Ort::Session *session,std::vector<float> &input, int N, int Nsamp, int Nreturn)

The *onnxSession* method simply loads in the ONNX model from a provided file name and initializes the onnx session accordingly.

The *onnxInference* method gets passed the session initialized in the first command as well as a vector of waveforms (&input), the number of channels being processed (N) in a batch, the number of samples per waveform (Nsamp), as well as the number of return parameters returned by the model (Nreturn). This parses the information into a format that ONNX needs and processes the waveforms accordingly.