Evaluation of Portable Programming Models to Accelerate LArTPC Detector Simulations

Zhihua Dong[*], Kyle Knoepfel[†], Meifeng Lin[*a], Brett Viren[*], Haiwang Yu[*], Kwangmin Yu[*]

[*]Brookhaven National Laboratory, Upton, NY 11973, USA

[†]Fermi National Accelerator Laboratory, Batavia, IL 60510, USA

https://www.anl.gov/hep-cce

Abstract

The Liquid Argon Time Projection Chamber (LArTPC) technology is widely used in high energy physics experiments, including the upcoming Deep Underground Neutrino Experiment (DUNE). Accurately simulating LArTPC detector responses is essential for analysis algorithm development and physics model interpretations. Accurate LArTPC detector response simulations are computationally demanding, and can become a bottleneck in the analysis workflow. Compute accelerators such as General-Purpose Graphics Processing Units (GPGPUs) have the potential to speed up the simulations significantly compared to the traditional CPU-only processing, often at the cost of specialized code refactorization and porting. With the rapid evolution and increased diversity of the computer architecture landscape, it is highly desirable to have a portable solution that also maintains reasonable performance. We report our ongoing effort in evaluating using Kokkos as a portable programming model for LArTPC simulations in the context of the Wire-Cell Toolkit, a new C++ library for LArTPC simulations, data analysis, reconstruction and visualization.

Introduction

The recorded digitized TPC signal from each wire plane can be modeled as a two-dimensional (2D) convolution of the distribution of the arriving ionization electrons and the impulse detector response:

$$M(t,x) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} R(t-t', x-x') \cdot S(t', x') dt' dx' + N(t,x), \quad (1)$$

- t: sampling time; x: wire position
- $M(t,x)$: a measurement, such as an analog-to-digital converter (ADC) value at a given x and t.
- $R(t-t', x-x')$: impulse detector response, including both the field response that describes the induced current by a moving ionization electron and the electronics response from the shaping circuit.
- $S(t', x')$: the charge distribution of the arriving ionization electrons,
- $N(t,x)$: electronics noise.

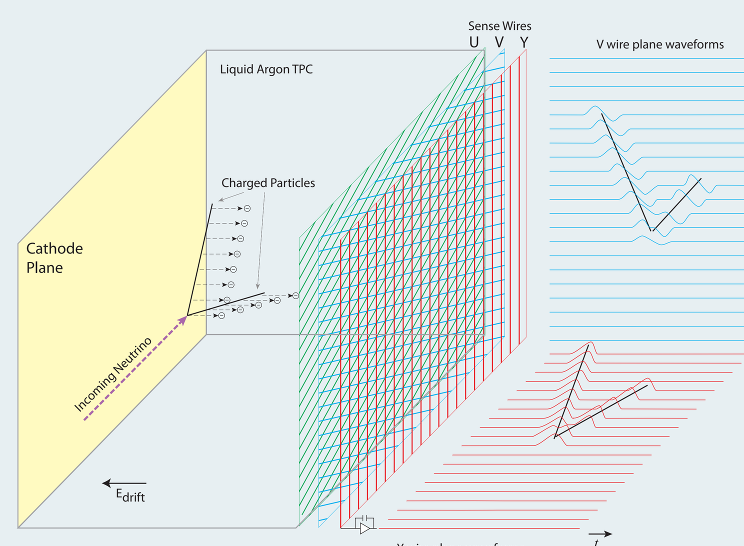

Figure 1: Illustration of a three-wire plane LArTPC and its signal formation.

We focus on the signal simulation part highlighted in red since it is more computationally demanding than the noise term. In the Wire-Cell Toolkit [1], the 2D convolution is calculated by Fourier transforming $S(t,x)$ to the frequency domain, applying a multiplicative correction, and then performing inverse Fourier transform back to the time-space domain:

$$\begin{aligned} S(t,x) &\xrightarrow{\textbf{FT}} S(\omega_t, \omega_x), \\ M(\omega_t, \omega_x) &= R(\omega_t, \omega_x) \cdot S(\omega_t, \omega_x), \\ M(\omega_t, \omega_x) &\xrightarrow{\textbf{IFT}} M(t,x), \end{aligned} \quad (2)$$

The calculation consists of three key computational tasks:

- **Rasterization**: Energy deposition is rasterized into small patches of size $\sim 20 \times 20$.
- **Scatter-Adding**: Summation of energy depositions into a larger grid of $\sim 10,000 \times 10,000$.
- **Convolution**: Add detector responses using Eq. (2) with Fourier Transformations.

Kokkos

Kokkos [2] is a C++ abstraction library targeting performance portability. It supports several different node architectures and memory models by allowing users to define their own execution and memory spaces. It maps C++ source code to different *backends* during build time to achieve portability.

- **Serial** backend, which executes single-threaded on a host device.
- **Host-parallel** backend, which executes multithreaded on the host device.
- **Device-parallel** backend, which executes on an external device (e.g. a GPU).

For the host-parallel backends, Kokkos currently supports OpenMP or POSIX threads (pthreads) for CPUs. The device-parallel backends include CUDA for NVIDIA GPUS, HIP for AMD GPUs, OpenMP target offloading and SYCL.

Implementation Details

To better experiment with the Kokkos abstraction layer, we developed the standalone Wire-Cell-Gen-Kokkos module [3]. In our vCHEP21 presentation [4], we showed results for a partial porting as a demonstration. Here we report the results of a full porting that implements all the main computational tasks in Kokkos, with the data flow between host (CPU) and device (GPU) shown below.

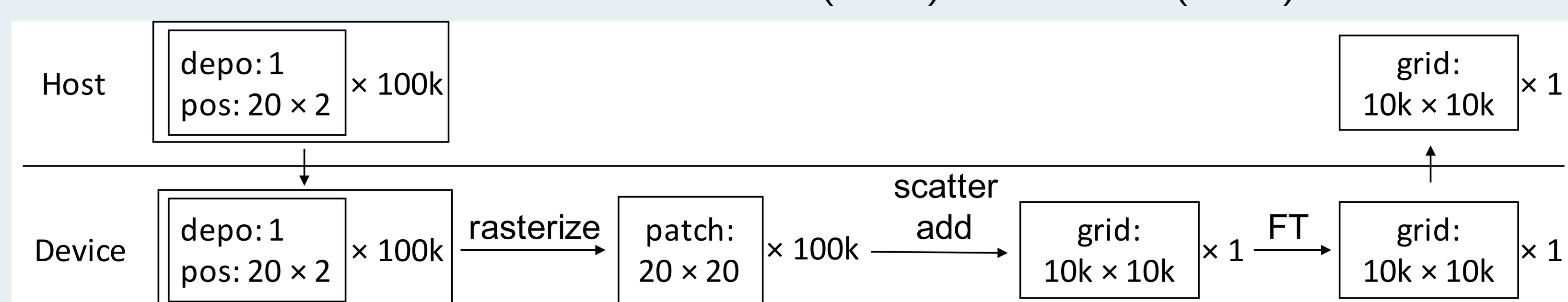

Figure 2: Data flow for the full Kokkos porting.

- **Extensions to the toolkit:** We added a C++ `KokkosEnv` context manager component to initialize and finalize Kokkos as well as special build system support.
- **Changes to data representation:** Data layout transformation to use dense matrix representation instead of sparse vectors.
- **FFT Wrapper API:** Since Kokkos does not provide an API to optimized vendor FFT libraries (`FFTW`, `cuFFT`, etc.), we implemented our own FFT wrapper similar to the Synergia group [5].
- **Code example:**

```
Kokkos::deep_copy(resp_f_w_k, resp_f_w_k_h); //data copying
resp_f_w_k = KokkosArray::dft_cc(resp_f_w_k, 1); // Discrete Fourier transform

auto data_c = KokkosArray::dft_rc(f_data, 0);
data_c = KokkosArray::dft_cc(data_c, 1);

// S(f) * R(f)
Kokkos::parallel_for(
    Kokkos::MDRangePolicy<Kokkos::Rank<2, Kokkos::Iterate::Left>>({0, 0}, {data_c.extent(0), data_c.
     extent(1)}),
    KOKKOS_LAMBDA(const KokkosArray::Index& i0, const KokkosArray::Index& i1) {
            data_c(i0, i1) *= resp_f_w_k(i0, i1);});

// transfer wire to time domain
data_c = KokkosArray::idft_cc(data_c, 1);

// extract M(channel) from M(impact)
Kokkos::parallel_for(
    Kokkos::MDRangePolicy<Kokkos::Rank<2, Kokkos::Iterate::Left>>({0, 0}, {acc_data_f_w.extent(0),
     acc_data_f_w.extent(1)}),
    KOKKOS_LAMBDA(const KokkosArray::Index& i0, const KokkosArray::Index& i1) {
        acc_data_f_w(i0, i1) = data_c((i0 + 1) * 10, i1);});
```

Result Validation and Performance Benchmarking

Validation: Waveforms after convolution with field response are tapped out from CPU reference implementation and Kokkos porting with different backends, shown in Figure 3. For both CUDA and OMP backends, the differences are at 0.01% level, which are likely caused by subtle implementation differences and should not affect any downstream analyses.

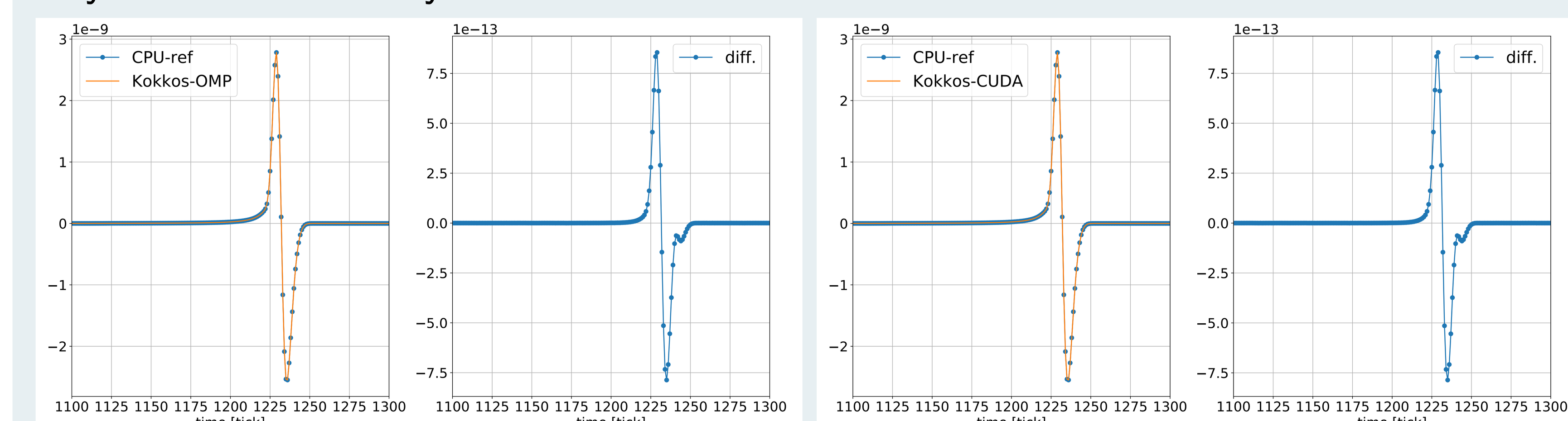

Figure 3: Waveform comparisons from CPU-ref and Kokkos with OpenMP and CUDA backends.

Benchmarking: The same code was tested on three different architectures: 24-core AMD Ryzen Threadripper 3960X for reference CPU implementation (CPU-ref) and Kokkos with OpenMP backend running 48 threads (Kokkos-OMP48), NVIDIA V100 GPU for Kokkos-CUDA and AMD Raedon Pro VII for Kokkos-HIP.

Computation [secs]	CPU-ref	Kokko-CUDA	Kokkos-HIP	Kokkos-OMP48
Rasterization	10.45	0.05	0.04	0.15
ScatterAdd	1.14	0.0006	0.007	0.013
FFT	5.44	0.71	2.50	13.3
Total Time	18.04	0.99	2.77	13.7

Table 1: Timing for the main computational tasks on different architectures averaged over 10 runs each.

- FFT is not parallelized for CPU-ref or Kokkos-OMP48. The slowdown may be due to implementation difference. Detailed investigation is ongoing.
- We get an overall speedup of $18\times$ on V100, and $7\times$ on Raedon Pro VII.
- The GPUs are still under utilized and can be shared by several parallel processes to gain further speedup using *e.g.* CUDA MPS.

Conclusions / Future Plans

- We have implemented the LArTPC signal simulation in Kokkos for portability across different architectures.
- Speedups have been achieved using GPUs with room for further improvements.
- We will look into other portable programming models, *e.g.* OpenMP and SYCL.

References

[1] https://github.com/WireCell/wire-cell-toolkit

[2] H.C. Edwards, C.R. Trott, *2013 Extreme Scaling Workshop (xsw 2013)* (2013), pp. 18–24

[3] https://github.com/WireCell/wire-cell-gen-kokkos

[4] Haiwang Yu *et al.*, EPJ Web Conf., 251 (2021) 03032

[5] https://web.fnal.gov/sites/Synergia/

[a]Speaker. Email: mlin@bnl.gov. Work supported by US Department of Energy, Office of Science, Office of High Energy Physics under the High Energy Physics Center for Computational Excellence (HEP-CCE), a collaboration between Argonne National Laboratory, Brookhaven National Laboratory, Fermilab and Lawrence Berkeley National Laboratory.

Argonne National Laboratory | Brookhaven National Laboratory | Fermilab | Berkeley Lab

ACAT2021, 11/28/2021-12/03/2021