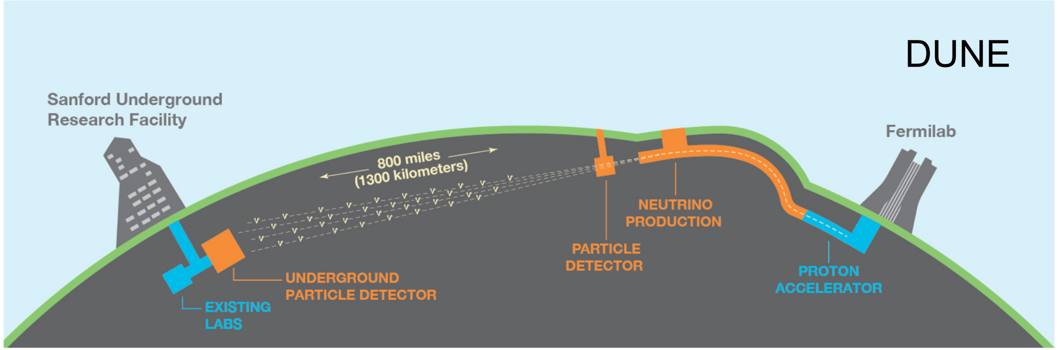
Portable Acceleration Solutions for LArTPC Simulation Using Wire-Cell Toolkit

Haiwang Yu[1], Zhihua Dong[2], Kyle Knoepfel[3], Meifeng Lin[2], Brett Viren[1] and Kwangmin Yu[2]

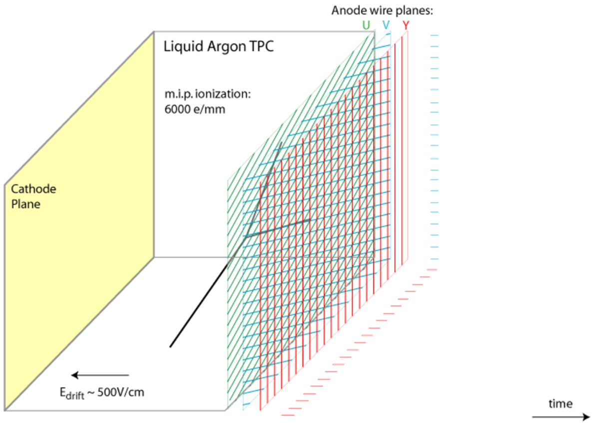
[1] Department of Physics, Brookhaven National Laboratory
[2] Computational Science Initiative, Brookhaven National Laboratory
[3] Scientific Computing Division, Fermi National Accelerator Laboratory

Liquid Argon TPC (LArTPC)

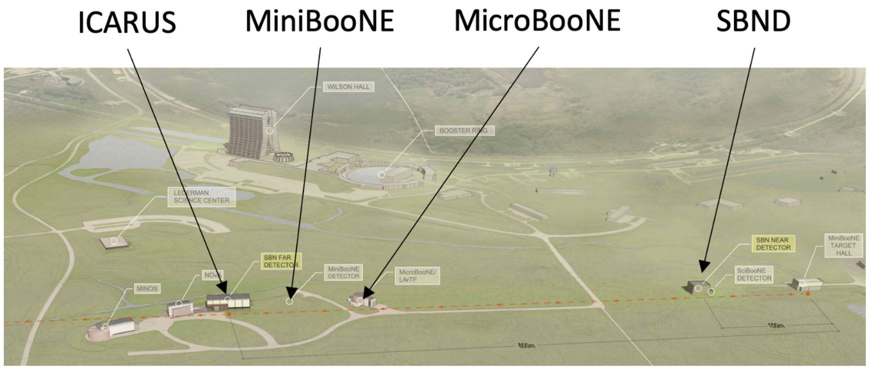
LArTPC is a key detector technology for many next-gen neutrino experiments
- rich and precise topology info.
- calorimetry info.



LArTPC Signal Formation



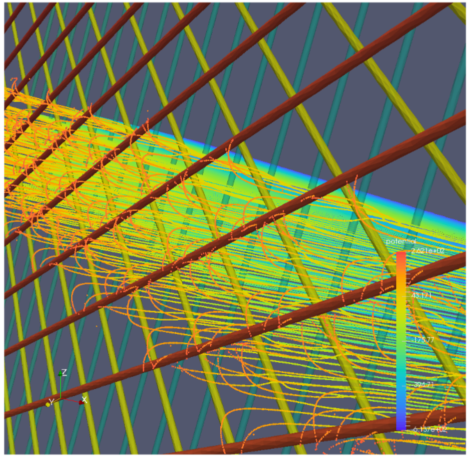
SBN Program



LArTPC Simulation

$$\boxed{\text{Ramo's theorem: } i = -q\,\vec{E}_w \cdot \vec{v}_q}$$

2D: approximate translational symmetry along the wire direction



LArTPC wire-readout measures induced charge $\otimes$ response

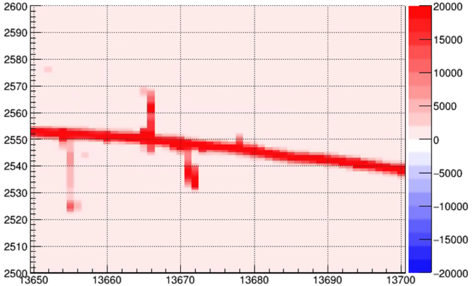
$$M(t',x') = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} R(t,t',x,x')\cdot S(t,x)dtdx + N(t',x')$$

Energy depo + diffusion + rasterization

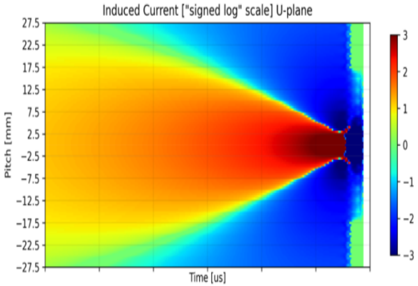
Long-range and position-dependent field response

Noise Spectrum

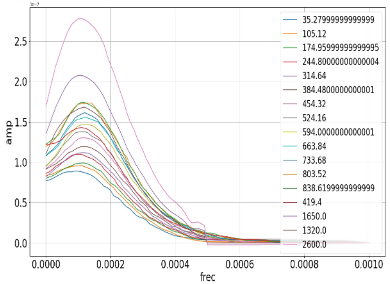
Final Signal



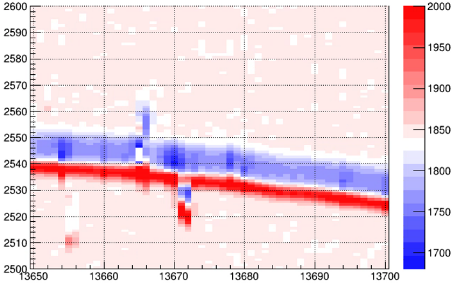
$\otimes$



$\oplus$



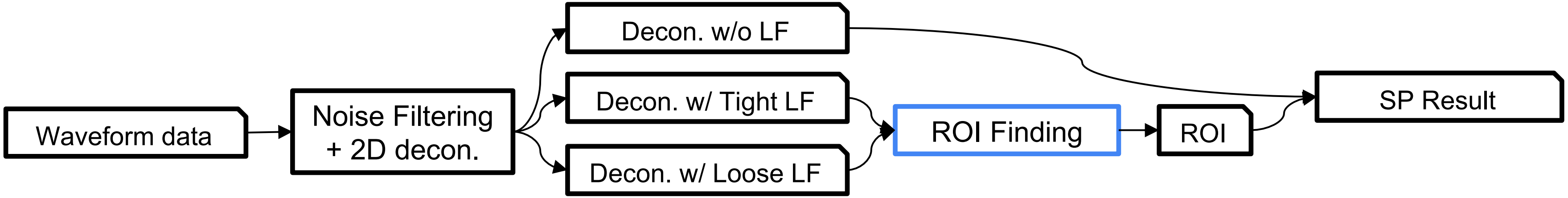
$\rightarrow$



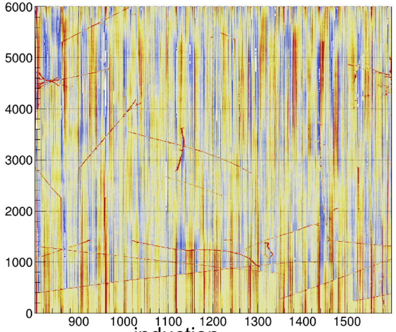
LArTPC Signal Processing

Signal Processing (SP) of LArTPC resolves charge from the original measurement:

$$S(\omega_t, \omega_x) \sim \frac{F(\omega_t, \omega_x) \cdot M(\omega_t, \omega_x)}{R(\omega_t, \omega_x)} \xrightarrow{IFT} S(t, x)$$

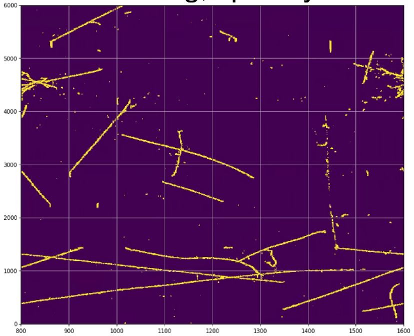


Decon. w/o LF filter
Waveform ⟶ charge, dense

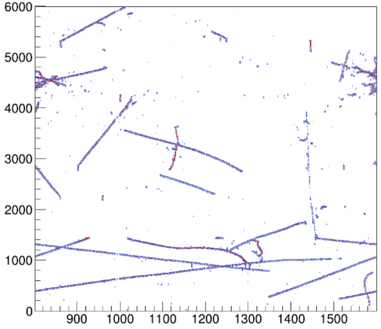


induction plane

ROI:
Hit finding, sparsify

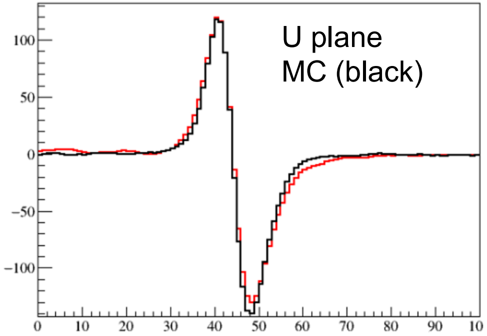


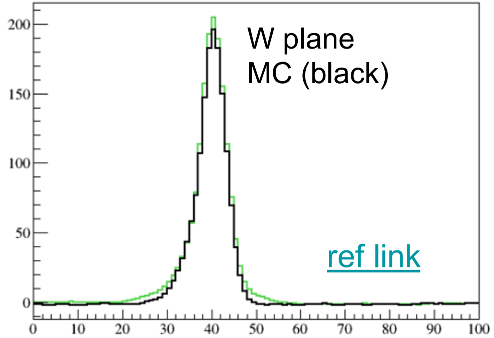
SP result:
Sparse, charge



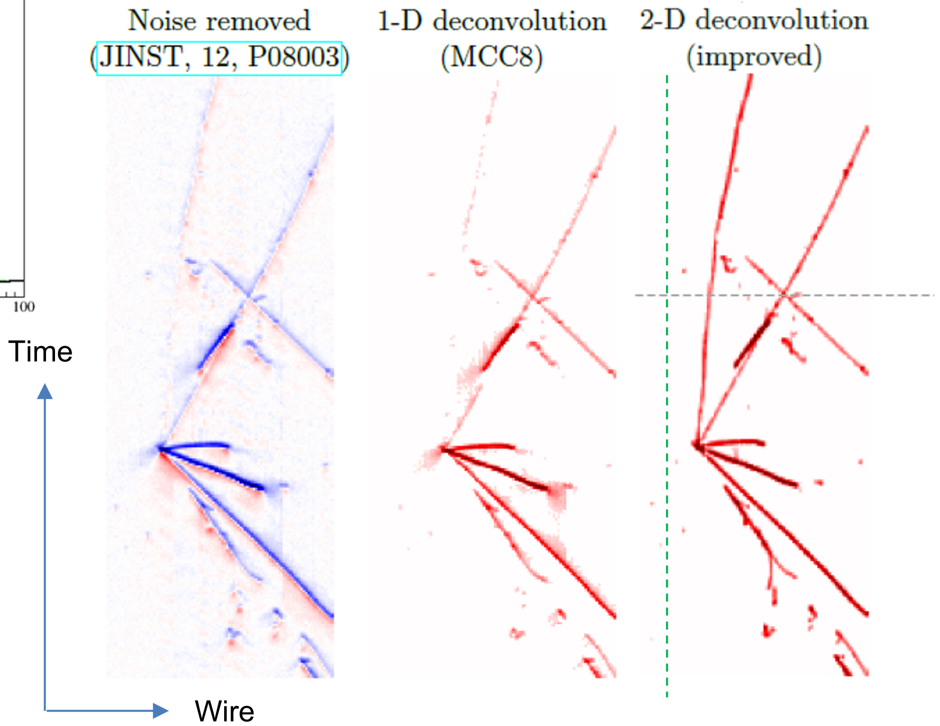
Validation and Performance

ProtoDUNE-SP
Average raw waveform: data vs. MC





MicroBooNE Data, Induction plane



Related publications
JINST 12 P08003
JINST 13 P07006
JINST 13 P07007
JINST 16 P01036

Wire-Cell Toolkit and LArSoft

Wire-Cell Toolkit (WCT) is a software package initialized for LArTPC

- algorithms: **simulation, signal processing**, reconstruction and visualization.
- data-flow programming paradigm
- modular design; can port different modules relatively independently
- works in both standalone mode and as plugin of LArSoft

LArSoft is a C++ software framework for many neutrino experiments using LArTPCs

- modular design
- infrastructures + algorithms
- central hub of the LArTPC software community

Accelerating Needs

Offline: both traditional and ML based

ML analysis examples:

- DNN ROI finding: JINST 2021 16 P01036
  - 500 APA data
  - total 16 hours
  - **Sim & SigProc - 9.4 CPU*hour**
- DUNE CVN, Phys.Rev.D 102 (2020) 9, 092003
  - 3 million APA data/9 million images

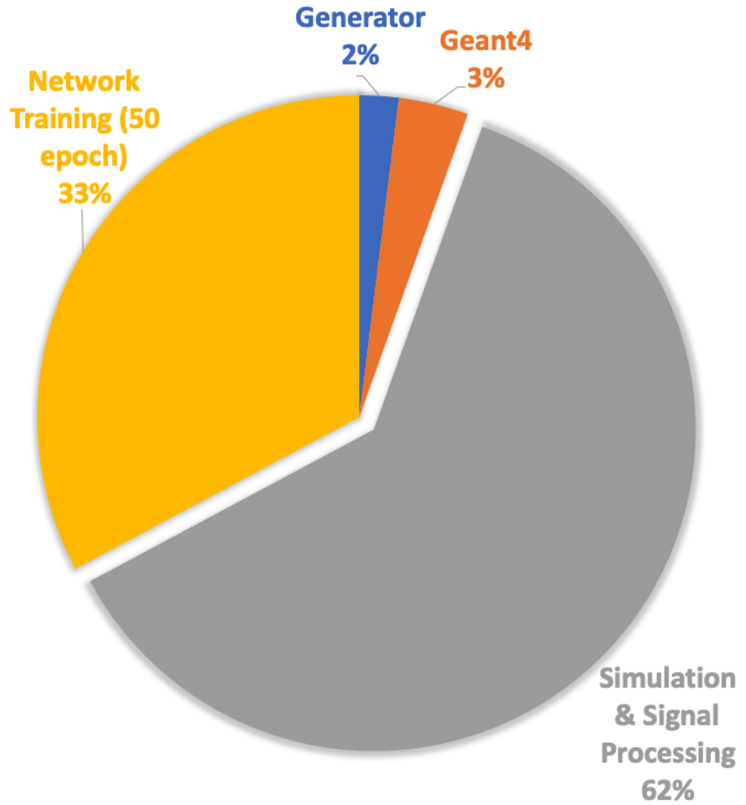
Online: TPC based online trigger

- supernova neutrino burst detection

Motivation to search for heterogeneous computing solutions, e.g. HPC

Refer to P. Laycock's talk for more on DUNE computing

Computing time breakdown for the DNN ROI finding task

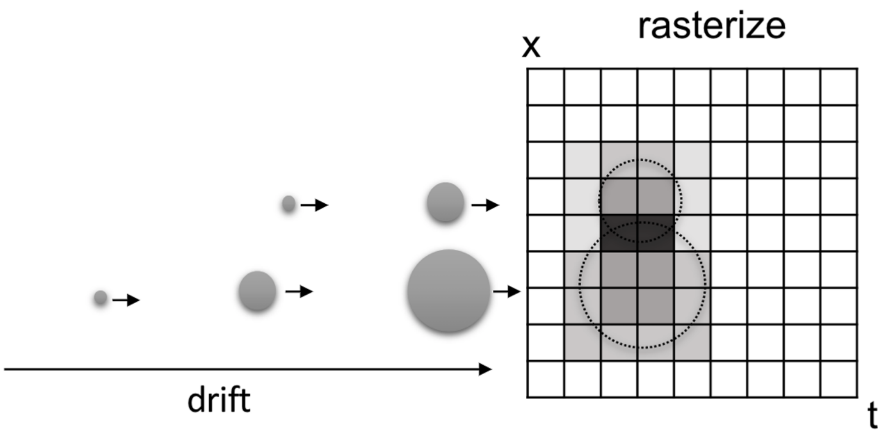


Wire-Cell Simulation Major Steps

Three major steps of LArTPC simulation with Wire-Cell - a representative workflow

1. Rasterization: depositions ⟶ patches (small 2D array, ~20×20)
    - # depo ~100k for cosmic ray event
2. Scatter adding: patches ⟶ grid (large 2D array, ~10k×10k)
3. FFT: convolution with detector response

rasterization and scatter adding



Convolution theorem:
convolution in time/space domain

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

$$\downarrow$$

multiplication in frequency domain

$$S(t,x) \xrightarrow{FT} S(\omega_t, \omega_x),$$

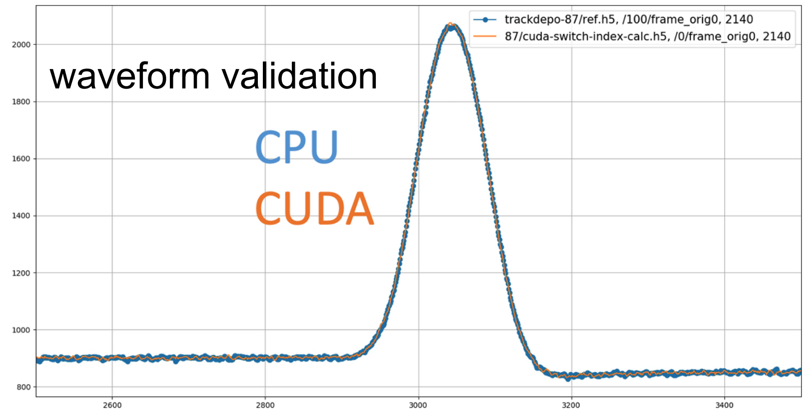
$$M(\omega_t, \omega_x) = R(\omega_t, \omega_x) \cdot S(\omega_t, \omega_x),$$

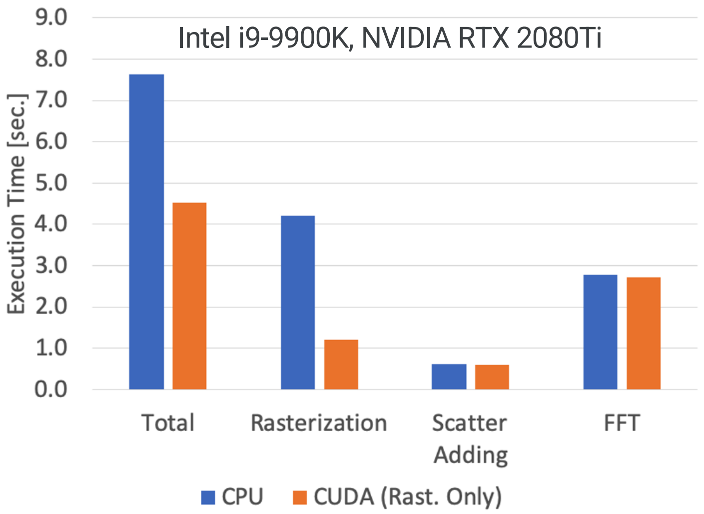
$$M(\omega_t, \omega_x) \xrightarrow{IFT} M(t,x).$$

Initial CUDA porting

First CUDA porting focused on the Rasterization step:

- 3× speedup for the Rast. step
  - parallelization at single patch level
  - RNG factored out ⟶ random number pool
- simulation results statistically consistent with CPU version

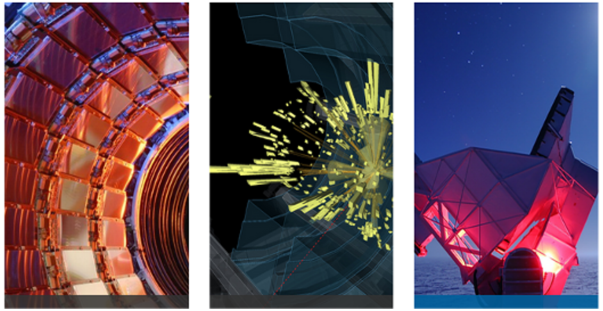




HEP-CCE and PPS

- HEP-CCE: High Energy Physics Center for Computational Excellence
    - A 3-year pilot US DOE project to develop solutions for HEP experiments to efficiently utilize diverse HPC resources
    - Covers 6 experiments in Cosmic, Intensity and Energy Frontiers.
    - Involves four US DOE labs: ANL, Fermilab, LBNL and BNL.
- PPS: Portable Parallelization Strategies
    - Focused on performance portability
    - Evaluation of **Kokkos, SyCL, OpenMP**, etc. as potential portability solutions for HEP
    - Use cases cover ATLAS, CMS and **DUNE**
- Started with **Kokkos** as the potential portability layer
    - Targets C++ applications
    - Supports multiple hardware architectures through different backends
    - Supports manual data management

For more details, see C. Leggett's Monday PM Plenary.



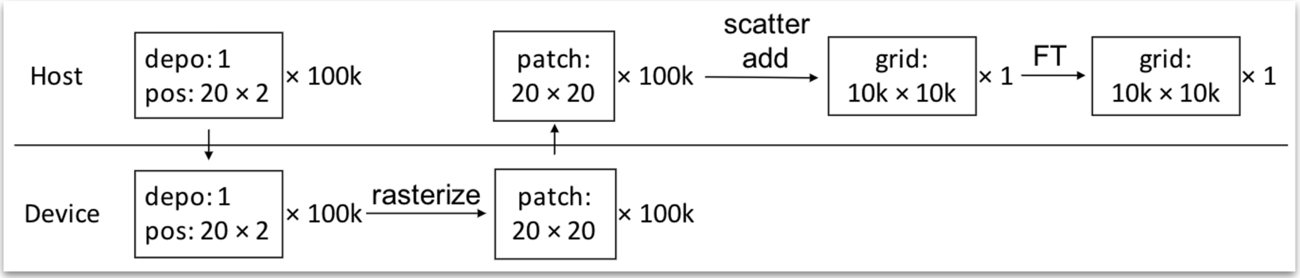
https://www.anl.gov/hep-cce

Kokkos Porting Plan

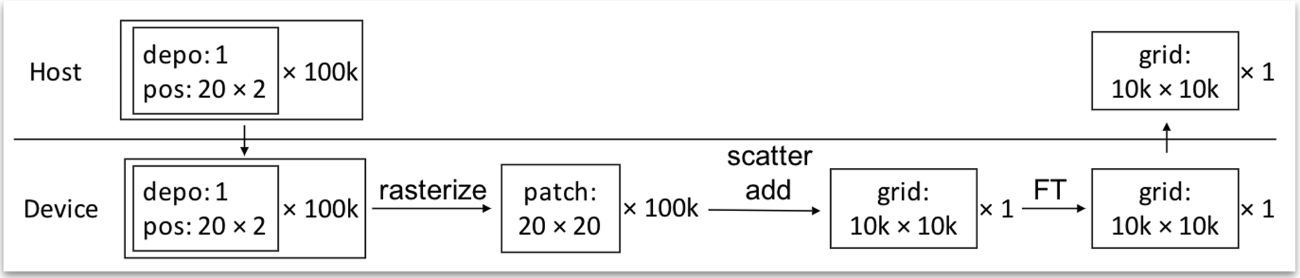
Two stage porting strategy

1. partial porting - port only step 1, rasterization
2. full porting
   a. more workloads for parallelization
   b. batched device-host data transfer

stage 1



stage 2



Developing Environment Setup

Standalone package: [wire-cell-gen-kokkos]
- clear interface to main Wire-Cell Toolkit and LArSoft
- minimum amount of code needs to be ported

Input data is provided in one of two ways:
- As JSON-serialized data to the standalone wire-cell executable
- Through LArSoft's larwirecell package as a plugin to the *art* event-processing framework

The framework solution allows a more realistic presentation of input data to the signal-processing algorithms.

Software dependencies
- LArSoft and Wire-Cell require a number of software packages (Boost, Geant, Python, etc.)
- This portability exercise was to be studied across several computing platforms (NERSC's Cori, BNL clusters, and private machines)
  - Argues for a package delivery system that is portable.

Use Docker containers to run on multiple platforms.

Kokkos container images

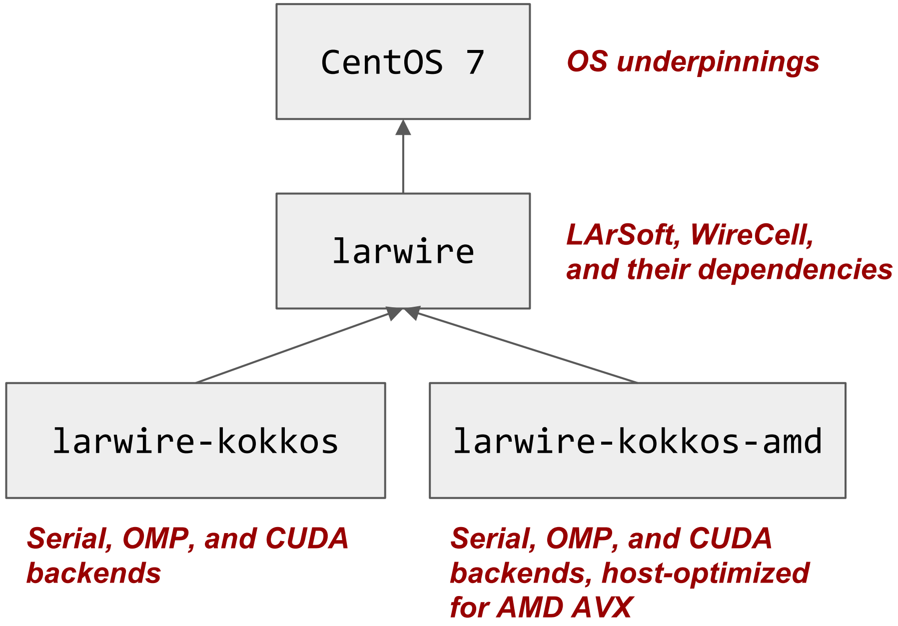
Docker containers/images

- Images contain installations of all required dependencies.
- A container is like an instance of an image.
- Images are layered in ways that allow for extensibility.
- The most derived layer has Kokkos/CUDA installations, possibly optimized for the host architecture.

Development workflow

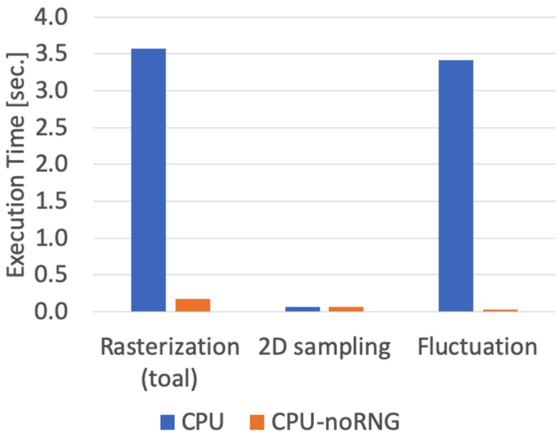
- Docker images are published to dockerhub, and converted to Singularity or Shifter (Cori) images.
- Kernels to be run on the GPU are compiled inside the container.

Image hierarchy



- Each job is run inside a container with a computing environment that suits the platform under study.

Random Number Generator (RNG)



Original serial CPU (ref-CPU) version:

- <random>
- gcc default: std::minstd_rand0
- Generate 1 number per use

CUDA:

- curandGenerateNormal
- Random number pool

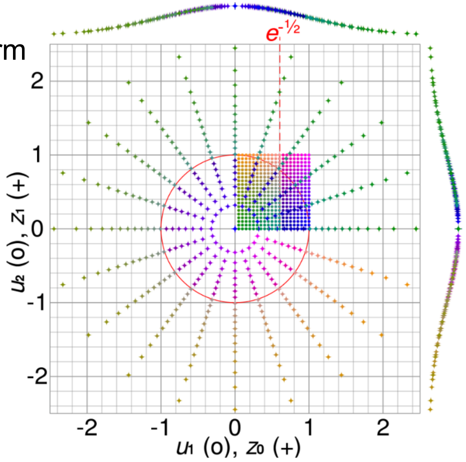
Kokkos:

- [Kokkos Xorshift RNG]
- Random number pool
- Box-Muller transform: Uniform ⟶ Gaussian

❑ curand and Kokkos RNG (CUDA) are much faster than CPU one

❑ in principle, we do not need unique RN for each patch, a large enough pool should also do the job

Box-Muller transform
from wikipedia



Scatter Adding

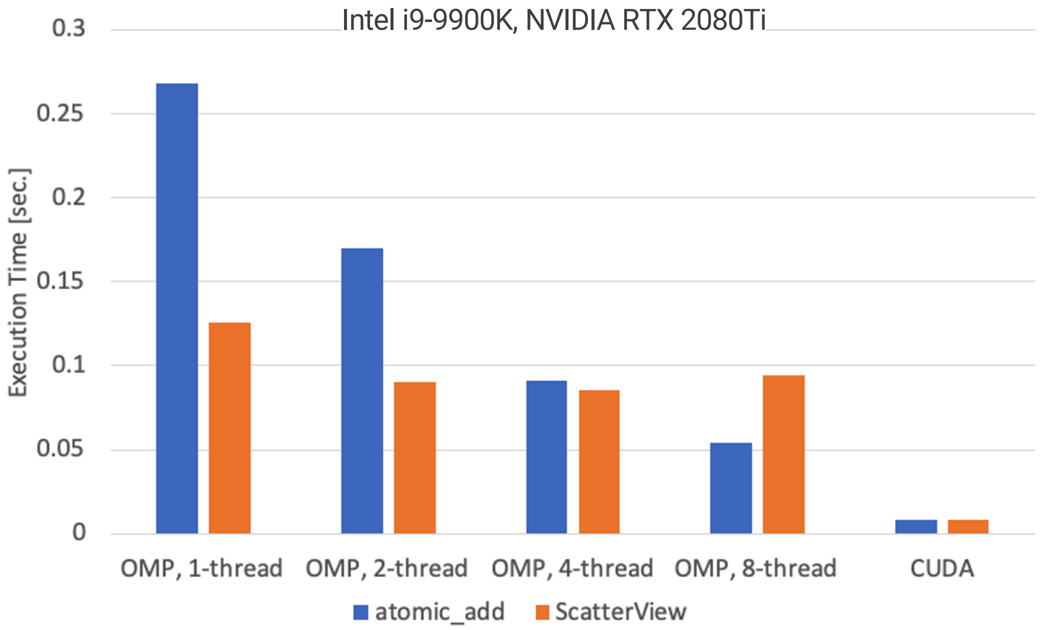
Initial attempts to do scatter adding with Kokkos

1. Kokkos::atomic_add
2. Kokkos::ScatterView

- atomic_add scales better with OMP threads
- ScatterView has better ST performance

- equal performance for CUDA

Unit test:
grid size: 1000 × 6000
patch size: 15 × 30
50k patches, avg. time for 10 executions



FFT

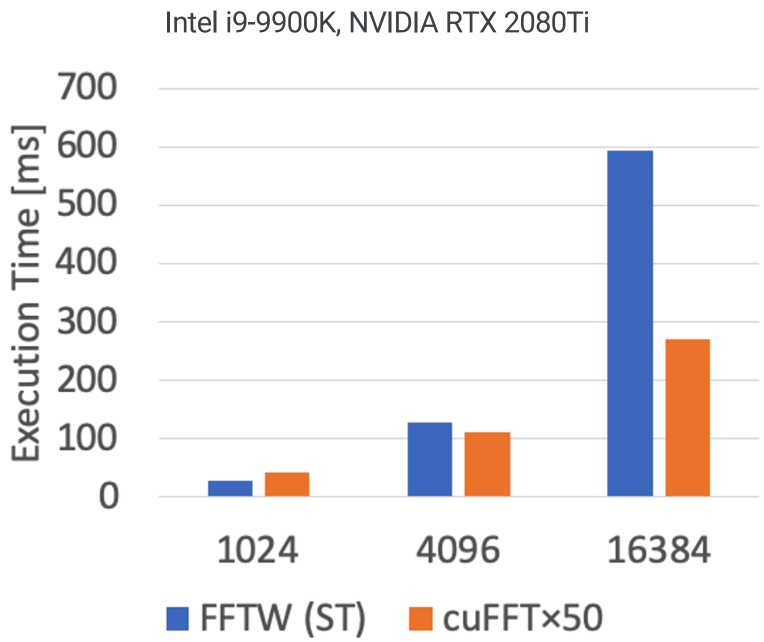
Kokkos wrapper for FFTW and cuFFT
Unit test: 1D FFT for 1024 arrays per operation
x-axis is length of array
using cufftPlanMany

Kokkos does not have a native FFT implementation or official interface to the optimized vendor libraries (FFTW, cuFFT) $\Longrightarrow$ **Use wrappers**

- Thanks to the synergia group for the helpful advices

cuFFT with **cufftPlanMany** performs 20× - 100× faster than FFTW on the test platform.

- Most of the times we need to do batches of 1D FFTs
- Previous FFTW version perform each one sequentially

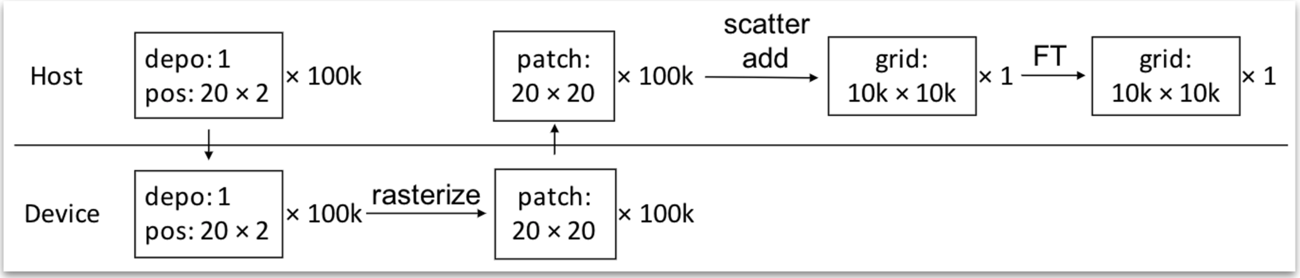


Kokkos Porting Plan

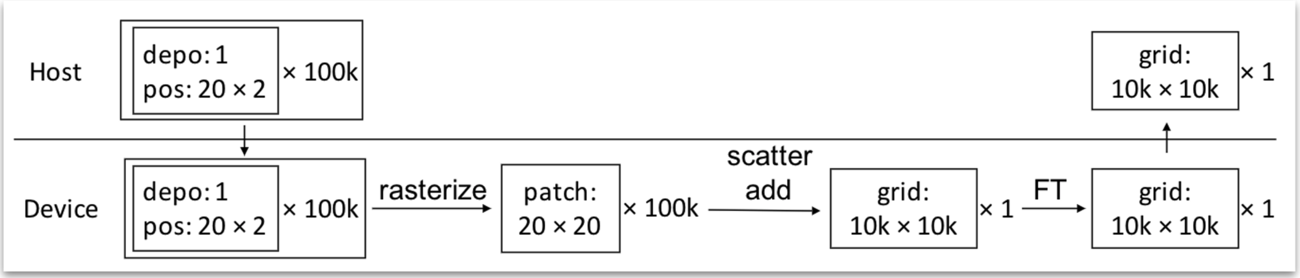
Two stage porting strategy

1. partial porting - port only step 1, rasterization
2. full porting
   a. more workloads for parallelization
   b. batched device-host data transfer

stage 1



stage 2



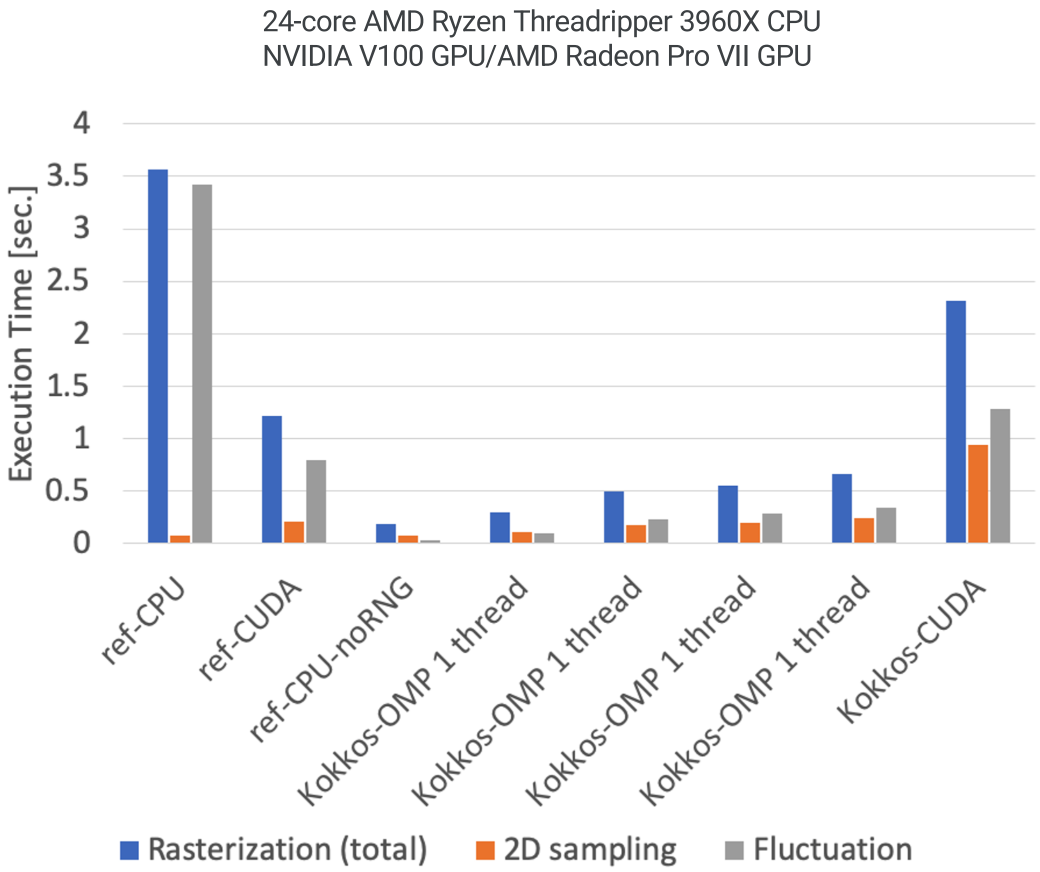
Stage 1

Initial Kokkos porting follows original CUDA porting.

- no need for major refactoring
- # concurrent workloads is small ~400
- results were not ideal

Nsight timeline analysis:

1. in between kernel and API calls, Kokkos has extra CudaDeviceSynchronization and CudaStreamSynchronization
2. Kokkos parellel_reduce() kernels are almost 3 times slower than CUDA reduction kernels in this version
    - too small workload only one block



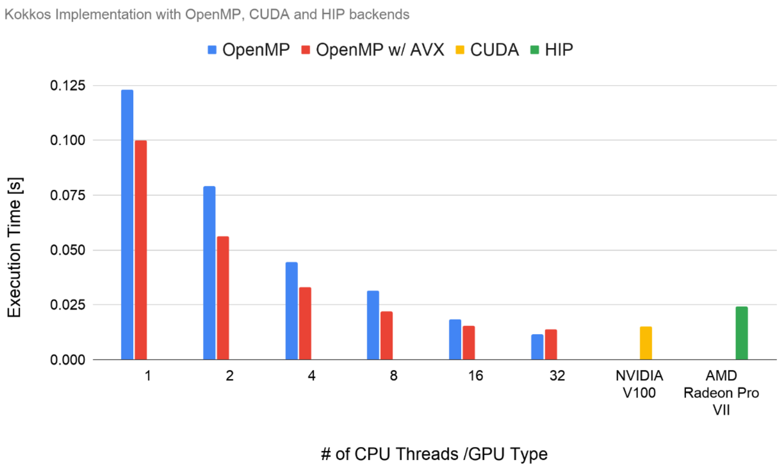
Stage 2 Milestone

24-core AMD Ryzen Threadripper 3960X CPU
NVIDIA V100 GPU/AMD Radeon Pro VII GPU

Boundle ~100k rasterizations together
(original total raserizations tasks become 2 parts)

- set_sampling_pre()
  - Prepare for 2D sampling
  - Currently serial ~0.085s
  - Working on parallelizing it
  - Expect good improvement in performance
- set_sampling_bat()
  - Single kernel for 2D samplings and fluctuations.
  - Include host/device data transfer
  - Use Kokkos *TeamThreadsRange* for CUDA
  - Use Kokkos *ThreadVectorRange* to enable SIMD on OMP backend.
- CUDA backend ~10x better than before

Timing for set_sampling_bat()



For GPUs, the actual kernel time are very small (<1ms)
Most time are on data transfers which will be absorbed in next step when scatter_add and FFTs parts are implemented on device with kokkos.

Stage 2 Full Prototype

Batched rasterization

Scatter Adding: major refactoring

- **sparse ⟶ dense (Kokkos::View)**
- **60× speed up**
- no extra HtoD needed for FFT

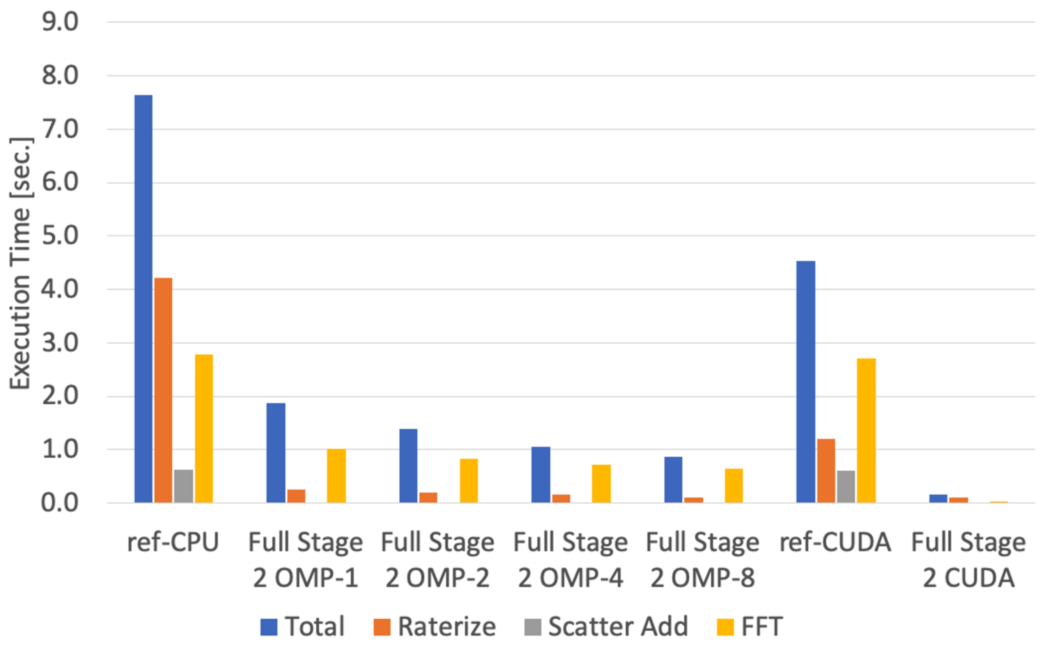
Kokkos FFT wrappers:

- cuFFT (cufftPlanMany): **88× (CUDA)**

Total speedup: **46×** (CUDA)

**Full Stage 2 porting is not completely finished yet, some tuning undergoing. But we expect the general trend should stay for the final version.*

Intel i9-9900K, NVIDIA RTX 2080Ti



Full Stage 2 Prototype

Batched rasterization

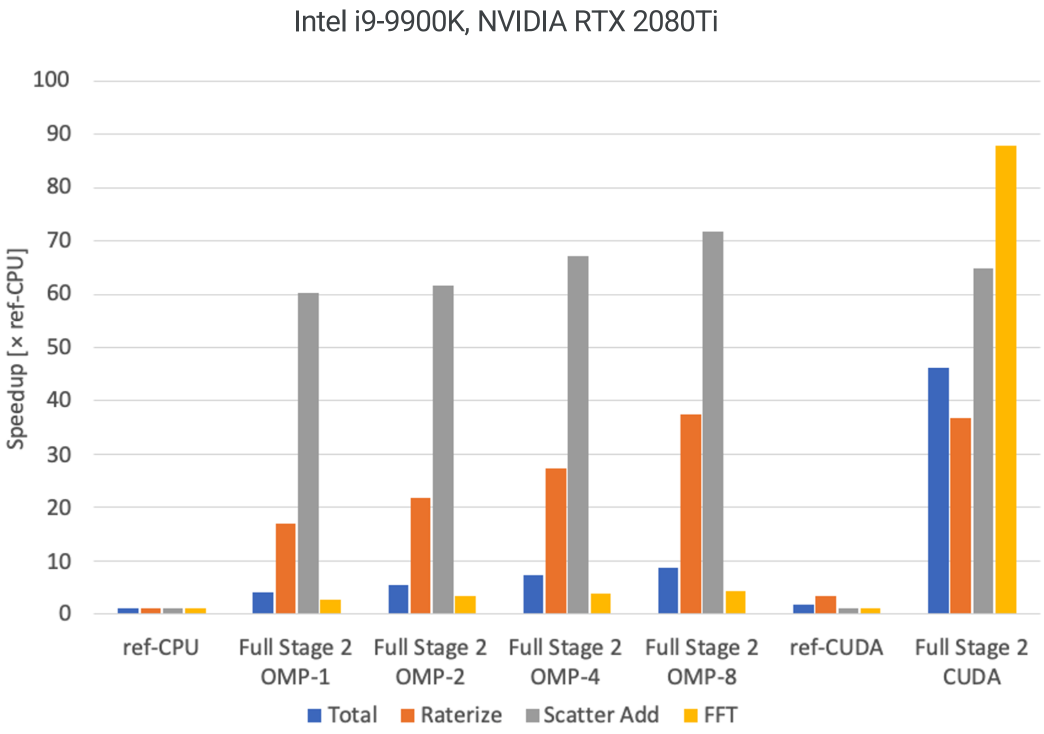
Scatter Adding: major refactoring

- sparse ⟶ dense (Kokkos::View)
- **60× speed up**
- no extra HtoD needed for FFT

cuFFT (cufftPlanMany): **88× (CUDA)**

Total speedup: **46×** (CUDA)

**Full Stage 2 porting is not completely finished yet, some tuning undergoing. But we expect the general trend should stay for the final version.*



Porting Experience

- Found general optimization directions even without accelerating in the game
  - factor out RNG
  - sparse ⟶ dense
- Major refactoring may be needed for code not initially designed considering parallel accelerating
  - significant improvement
  - benefit for both portable and non-portable solutions
    - larger workload, better data coalescence, less D-H transfer
    - (portable in a different sense?)
- Well organized D-H transfer is not as scary
- Containerized development making the environment setup really easy for multiple platforms

Future Plan

- Finish Kokkos porting for Wire-Cell Simulation
  - optimizations
  - validations
- Port Wire-Cell Signal Processing
- Better GPU utilization
  - data batching
  - Multi-Process Service
- Explore more backends and portability solutions
  - HIP
  - SYCL, Parallel C++ STL
- Applications:
  - production with suitable hardware
  - collaboration with online analyses

Backups