Wire-Cell PPS Status

Zhihua Dong, Meifeng Lin, Tianle Wang, Haiwang Yu
Brookhaven National Laboratory

for the HEP-CCE PPS Group

HEP-CCE All Hands Meeting, October 11-13, 2022
Lawrence Berkeley National Laboratory

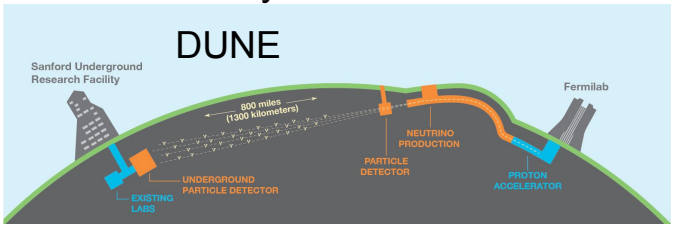
Motivation

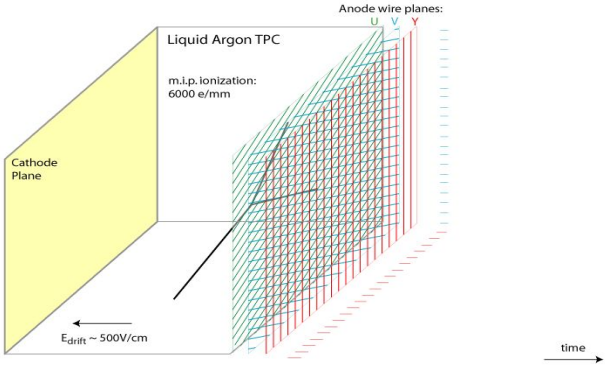
For ACAT presentation
Exascale architecture diversity
Need for fast processing, etc.

Liquid Argon TPC (LArTPC) and Wire-Cell Toolkit

LArTPC is a key detector technology for many next-gen neutrino experiments

- rich and precise topology info.
- calorimetry info.





LArTPC Signal Formation

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
- https://github.com/WireCell/wire-cell-toolkit

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

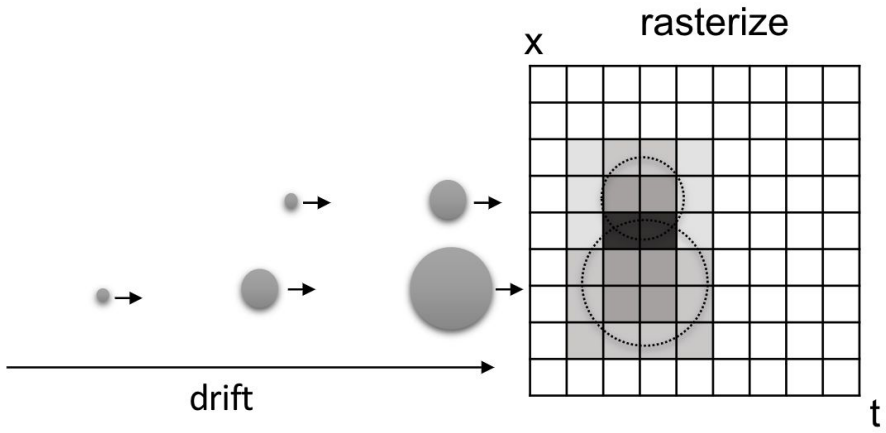
- modular design
- infrastructures + algorithms
- central hub of the LArTPC software community
- https://larsoft.org/

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

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

multiplication in frequency domain

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

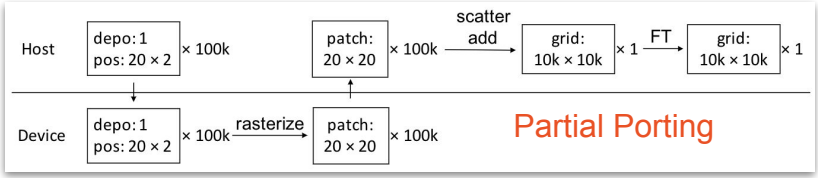
$$M(\omega_t,\omega_x) = R(\omega_t,\omega_x)\cdot S(\omega_t,\omega_x),$$

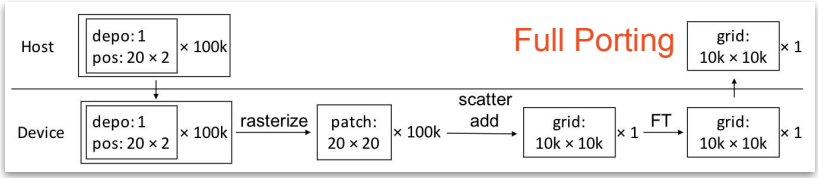
$$M(\omega_t,\omega_x) \xrightarrow{IFT} M(t,x).$$

Recap: Kokkos Porting Strategies and Results

Two stage porting strategy

1. Started with partial CUDA porting [1]: Only **rasterization part was ported GPU**
   a. Feasibility test and baseline performance
   b. Not performant
2. Full porting [2]: All three components ported to GPU
   a. more workloads for parallelization
   b. batched device-host data transfer





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.

References:

[1] Z. Dong, K. Knoepfel, M. Lin, B. Viren, H. Yu and K. Yu, vCHEP 2021, arXiv: 2104.08265

[2] Z. Dong, K. Knoepfel, M. Lin, B. Viren, H. Yu and K. Yu, ACAT 2021 poster, arXiv:2203.02479

Recap: Kokkos port challenges and code changes

- Wire-Cell Toolkit and its associated tests (relied ton LArSoft) have many dependencies
  - Use **Docker containers** to package the dependencies, compilers and Kokkos builds
- Wire-Cell uses task-based programming model: need to retain the flexibility that some tasks may not run on the GPUs
  - Added a C++ **KokkosEnv** context manager component to initialize and finalize Kokkos
- Kokkos does not provide a wrapper API for optimized vendor FFTs (FFTW, cuFFT, etc.)
  - Implemented own FFT wrapper similar to the Synergia group
- Numerous code refactoring and reorganization to make it more GPU friendly
  - Improved RNG usage (also improved CPU performance significantly)
  - Data layout transformation to use dense matrix representation instead of sparse vectors.
  - …

FY22 Activities: Porting WCT to OpenMP and SYCL

Describe OpenMP and SYCL
Compare OpenMP to Kokkos to SYCL

FY22 Activities: Porting WCT to OpenMP and SYCL

SYCL is a programming model and (Khronos) standard that brings support for heterogeneous programming to C++ .
Single source , different backend enable parallel execution on a range of hardwares CPUs GPUs, DSPs, FPGAs…

OpenMP is an API for multithreading, and it starts to support “target offloading” on heterogeneous architectures since OpenMP 4.0. It now supports several programming and memory models, including shared-memory parallelism, task parallelism, and host-device heterogeneous computing.

SYCL

```
#include <CL/sycl.hpp>

int main() {
 cl::sycl::queue Queue;
 unsigned long N=1024*1024 ;
 float a_h[N] ;
 auto a_d=cl::sycl::malloc_device<float>(N,Queue) ;
 Queue.parallel_for(
       cl::sycl::range<1>(N), [=]( auto item) {
            int id = item.get_id(0) ;
            a_d[id] = cl::sycl::sqrt((float)id ) ;
      });
  Queue.wait() ;
  Queue.memcpy(a_h, a_d, N*sizeof(float)).wait() ;
  ....
}
```

Kokkos

```
#include <Kokkos_Core.hpp>

int main() {
  Kokkos::initialize( argc, argv );
  {
  unsigned long N=1024*1024 ;
  typedef Kokkos::View<double*>   ViewVectorType;
  ViewVectorType a_d( "A", N );
  Kokkos::parallel_for("A2" , N, KOKKOS_LAMBDA (int i ) {
            a_d[i] = sqrt((double)i ) ;
      });
   Kokkos::fence();
   auto a_h = Kokkos::create_mirror_view( a_d );
   Kokkos::deep_copy(a_h, a_d, N*sizeof(double)) ;
   ....
   }
   Kokkos::finalize() ;
}
```

OpenMP

```
#include<omp.h>
#include<math.h>

int main()
{
  unsigned long N = 1024 * 1024;
  double *a = (double*)malloc(sizeof(double) * N);
#pragma omp target enter data map(to:a[0:N])
#pragma omp target teams distribute parallel for
  for(auto i=0; i<N; i++)
    a[i] = sqrt(a[i]);
#pragma omp target exit data map(from:a[0:N])
}
```

Porting Wire-Cell-Gen to SYCL

Zhihua Dong

Strategies:

- SYCL syntax is very similar to Kokkos, porting from Kokkos is straightforward.
- Create Array1D, Array2D classes (pointer and sizes with few methods) to replace KokkosArray(warpper of Kokkos::view) —-> Minimum code change.

```
Kokkos::deep_copy(sps_f, spf_h);
auto  sp_ts = KokkosArray::idft_cr(sp_fs,1) ;
```

→

```
sp_fs.copy_from(sps_h);
auto  sp_ts = SyclArray::idft_cr(sp_fs,1) ;
```

- SYCL does not have RNG as Kokkos, we use wrapper (cuRAND,rocRAND,random123)
  https://github.com/GKNB/test-benchmark-OpenMP-RNG.git (Tianle Wang)
- FFT part similar to Kokkos , use wrapper (cuFFT,rocFFT, host original based on FFTW)

Things to be careful :

- SYCL kernels and memory operation always async by default. Eg. sycl::memcpy()
- Some Kokkos functions put extra fence e.g. deep_copy()

Porting Wire-Cell-Gen to SYCL

Zhihua Dong

Issues:

- Various SYCL Compilers some fast developing, each have some issues for WireCell Gen code
  - Intel distributed oneAPI (dpcpp) → does not support AMD/Nvidia GPU
    - Host backend use tbb by default
    - Long delay ~1s for 1st kernel launch (?)
  - github.com/intel/llvm —> works for Nvidia/AMD,
    but hostbackend not full feature supported. (e.g group level collective like scan,sum)
  - hipSYCL —> We only got OMP backend work, others have run time error, or build error
    - Also strict in Syntax

```
q.parallel_for(N0, [=] ( auto i0) { ptr[i0] /= N0 ; } ) ;  // Won't compile on hipSYCL
```

- Due to lack of working standalone Wirecell running envrionment , we have to run under LarSoft framework which we need container for run on different platforms.
  Have not successfully build intel/llvm compiler for AMD backend within container.

Porting Wire-Cell-Gen to OpenMP

Tianle Wang

Strategies:

- When porting using OpenMP, we simply add #pragma for data movement and kernel, we don't need to change the CPU code a lot.
- However, the original CPU code is not suitable for GPU, so we port from the Kokkos implementation.
- Use one dimensional array to represent all the data.
- Manually perform data movement using #pragma omp target data map to remove unnecessary data movement and lower peak memory usage.
- OpenMP does not support GPU scan (prefix) operation, so we do padding, at the cost of small extra memory.
- OpenMP does not have RNG as Kokkos, we use wrapper (cuRAND,rocRAND,random123) https://github.com/GKNB/test-benchmark-OpenMP-RNG.git
- For FFT part, similar to Kokkos, we use wrapper (cuFFT,rocFFT, host original based on FFTW).
- Use #pragma omp atomic for scattering add.

Porting Wire-Cell-Gen to OpenMP

Tianle Wang

Issues:

- For most of the kernels, a simple porting can give a decent performance on both CPU and GPU. However, there is one kernel (set_sampling_bat) that needs different parallelism pattern.
- Different from Kokkos and Sycl, it is difficult to change the data layout. Also there is no easy-to-use data structure (e.g. multi-dimensional array).
- Compiler is still developing for better performance (e.g. atomic operation).
- Currently can not compile the project with the latest clang compiler (clang-15) and nvc++ compiler.
- Currently can not find GPU inside container, so we can not test our code on other platforms which requires container.
- Data movement speed is 1.5-3 times slower than that in Kokkos.
- We don’t need to initialize and finalize OpenMP like Kokkos, but the first data allocation on GPU requires a long time (~60 ms).
- OpenMP usually use more registers then CUDA for the same kernel.

Performance Comparison (SYCL vs. Kokkos vs. OpenMP)

Benchmark :
Code was run on same workstation with:

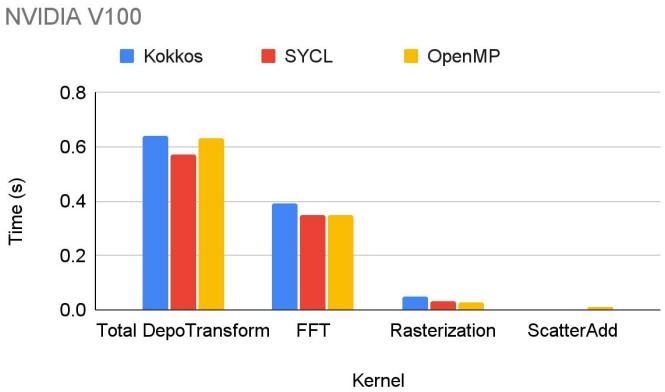
- 24 core AMD Ryzen Threadripper 3960X, 48 HT
- Nvidia V100 GPU
- AMD Raedon Pro VII

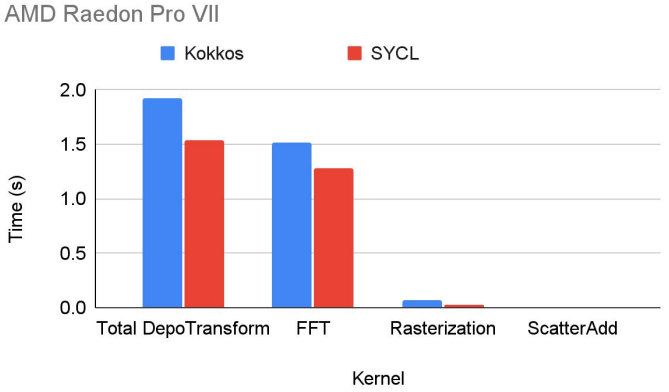
Total DepoTrasform time as well as Rasterization, AcatterAdd and FFT parts are measured for

- SYCL with CUDA backend , SYCL with HIP(AMD) backend, SYCL Host (hipSYCL & dpcpp)
- OpenMP with CUDA backend, OpenMP Host with clang-13 compiler
- Compared with Kokkos CUDA backend and Kokkos HIP(AMD) backend

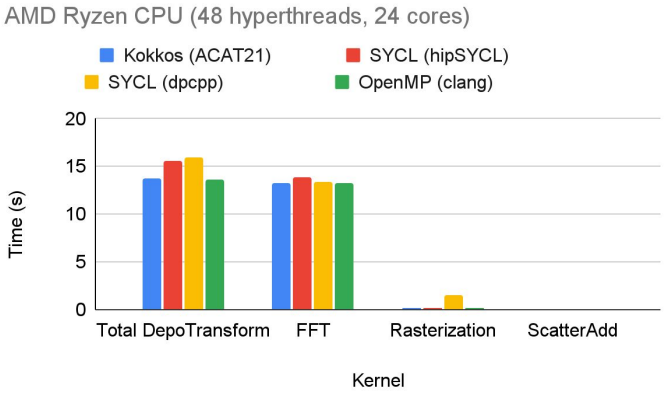
Time(s)	SYCL CUDA	SYCL HIP(AMD)	Kokkos cuda	Kokkos HIP(AMD)	SYCL omp48 (hipSYCL)	SYCL 48 Threads (dpcpp)	OpenMP cuda	OpenMP 48 threads
Total DepoTransform	0.57	1.54	0.64	1.92	15.6	15.9	0.63	13.6
FFT:	0.35	1.28	0.39	1.52	13.9	13.4	0.35	13.2
ScatterAdd:	0.003	0.006	0.0006	0.006	0.023	0.036	0.009	0.064
Rasterization:	0.030	0.026	0.050	0.065	0.18	1.55	0.027	0.15

Performance Comparison

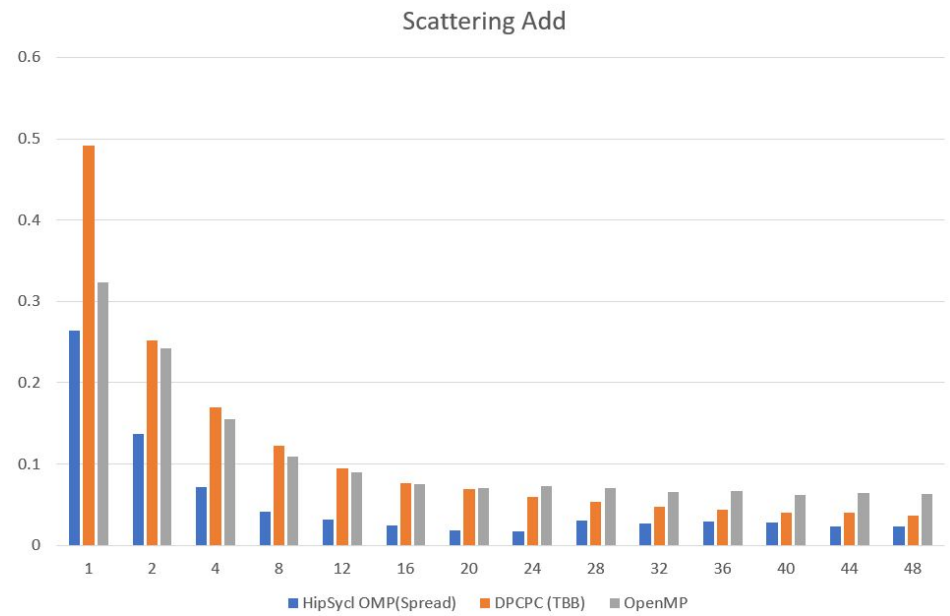


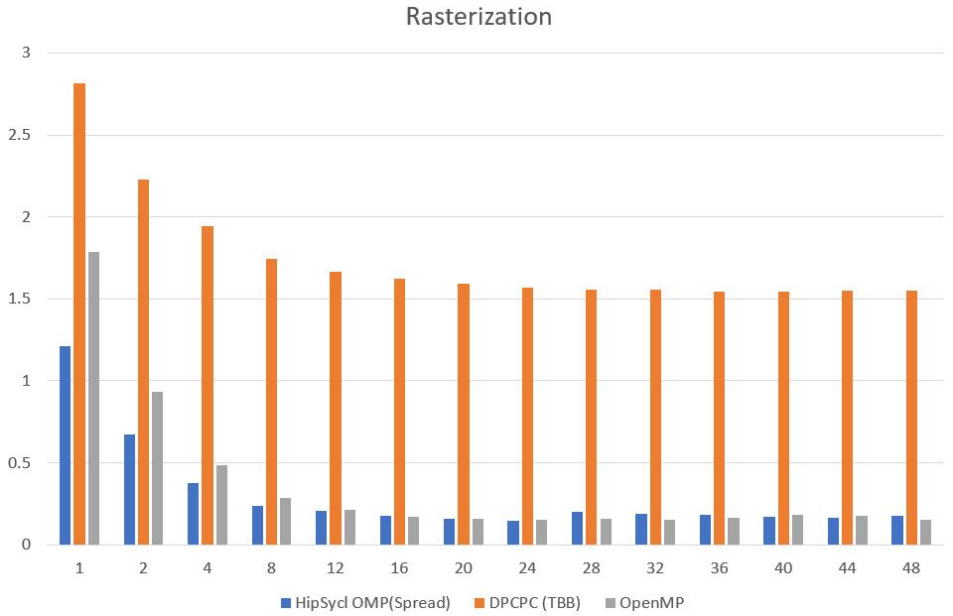


- **FFT :** SYCL/OpenMP better than Kokkos:
  - Due to optimized normalization kernel (fft) and S*F etc kernels from 2D to 1D Compared to Kokkos code
- **Rasterization:** SYCL/OpenMP better than Kokkos
  - Mainly due to RNG . Other kernels are similar in timing
- **ScatterAdd:**
  - SYCL 5x slower than Kokkos for CUDA backend
  - OpenMP 15x slower than Kokkos for CUDA backend (should be much better in clang-15)
  - SYCL same as Kokkos for HIP(AMD)



CPU Scaling Comparison (SYCL vs. OpenMP)





OMP24+ rebound
Due to OMP thread arrangements

DPCPC 1st kernel have extra 1s+ time

Summary and Plans

- Porting the Kokkos implementation of WCT to SYCL was relatively straightforward due to similarities of their syntaxes.
- Porting to OpenMP was also relatively easy, but getting the most of the performance required a bit more optimization work.
- Compiler support for both SYCL and OpenMP is still under development. Performance across different architectures is variable.
- NVIDIA GPUs are the best supported (in terms of performance) by all three programming models.
- Lack of a universal API for portable optimized libraries (such as FFT and RNG) is a common issue for all the portable programming models.

Future Plans

- Finish the HIP backend for the OpenMP implementation
- Write up a paper with Kokkos, SYCL and OpenMP experiences
- If time allows, investigate Alpaka