Summary of CCE-PPS team

Martin Kwok
for the PPS team

HEP-CCE All-Hands Meeting
Apr 20, 2022

HEP Is (Slowly) Embracing Heterogeneous Computing

Challenges:

Hundreds of computing sites (grid clusters + HPC systems + clouds)

Hundreds of C++ kernels (several million LOC, no hot-spots)

Hundreds of data objects (dynamic, polymorphic)

Hundreds of non-professional developers (domain experts)

→ Can't rewrite code to target every HPC platform

Opportunity:

Scale of experiments and community provides significant R&D firepower

scores of active groups, will not attempt to list

Current Focus:

Online event filtering, offline pattern recognition, detector simulation

Portable Parallelization Strategies (PPS) Activities

Investigate a range of software **portability solutions**:

- Libraries
- Compilers
- Language extensions

Define a set of **metrics** to evaluate portability solutions, as applied to our testbeds

- Productivity, cross-platform performance, broader impact, long-term sustainability, *etc*

Port a small number of HEP **testbeds** to each portability solution

- Tracking
- Simulation

Make **recommendations** to the experiments

- Must address needs of both LHC style workflows (many modules and many developers), and smaller/simpler workflows

Portability Solutions: Software Support Chart

	NVIDIA CUDA	AMD HIP	dpc++ / SYCL	Kokkos	Alpaka	std::par	OpenMP Offload	Python
NVIDIA GPU								
AMD GPU			*hipSYCL intel/llvm*	*select GPUs*				
Intel GPU		HIPLZ: *early prototype*			*experimental*			
CPU								
FPGA					*experimental(via SYCL)*			

Supported

Under Development

3rd Party

Not Supported

All green cells in table are potential targets for our studies, details later

- **products are rapidly evolving**
- some hope of seeing emergence of industry standards at language level

Metrics for Evaluation of PPS Platform

Ease of learning (experts and novices) and extent of code modification

Code conversion

- CPU → PPL / CUDA → PPL / PPL → PPL

Impact on other existing code

- Event Data Model
- does it take over main(), does it affect the threading or execution model, *etc*

Impact on existing toolchain and build infrastructure

- do we need to recompile entire software stack?
- cmake / make transparencies

Hardware mapping

- evolving support for new hardware features
- new architectures

Feature availability

- reductions, kernel chaining, callbacks, *etc*
- concurrent kernel execution

Ease of Debugging

subjective and objective

Address needs of all types of workflows

- scaling with # kernels / application
- scaling with # developers
- compute vs memory bound

Long-term sustainability and code stability

- Support model of technologies ➜ stability of implementation if underlying libraries (CUDA) change
- CUDA is going to be around for a long time, what about the portability solutions?
- Long term support for technologies by vendors

Compilation time

- separate builds for different architectures?

Performance: CPU and GPU

- degradation of CPU code?

Interoperability

Aesthetics

- compatibility with C++ standards

→ more details

Testbed Applications

Detector simulation

- Full MC simulation (Geant4) large code base, hard to parallelize, resource intensive. Use to develop/train
  - Fast MC simulation: effective models (ML or parametrized by hand).
    - FastCaloSim (ATLAS) → arXiv:2103.14737 (HEP-CCE)
      - Compact, regular application, good initial target
    - WireCell LArTPC simulation → arXiv:2104.08265 (HEP-CCE)
      - 2D FFT Convolution-based LArTPC Simulation

Particle tracking

- sequence of complex, resource-intensive pattern recognition steps
- nested, dynamic data structures
- vibrant R&D on parallel algorithms targeting GPUs and FPGAs
  - Patatrack Pixel Tracking, p2r (CMS) → arXiv:2104.06573 (HEP-CCE)
  - ACTS (ATLAS, sPHOENIX, …): experiment-independent toolkit for track simulation and reconstruction

Event Generation

  - Experiment-independent applications, theory community, ~20% of projected cycles for HL-LHC
    - Platform-independent version of Madgraph5-GPU (details in the EG section)

Status of Ports for Testbeds

	CUDA	HIP	Kokkos	SYCL	OpenMP	Alpaka	std::par
Patatrack				*Planned*		*not by CCE*	
WireCell	*partial*						
p2R					*OpenACC*		
FastCaloSim							
ACTS							

We will go through some examples. Details in backup slides (→ go there)

ATLAS FastCaloSim Testbed

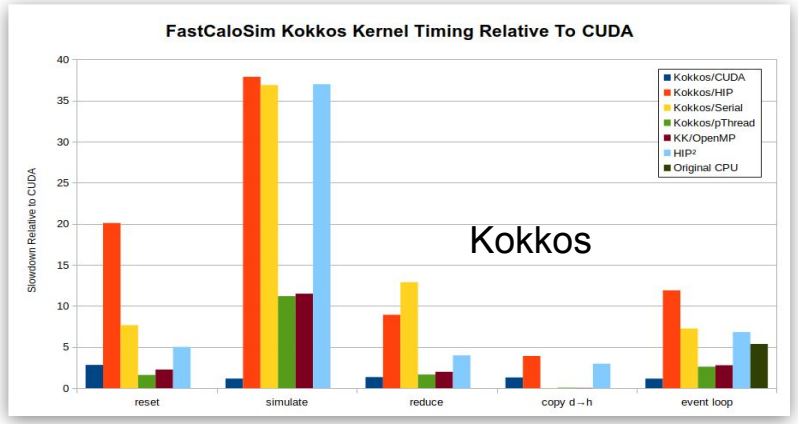
arXiv:2103.14737

HEP-CCE

Developed parallel version (CUDA) of ATLAS parameterized calorimeter simulation

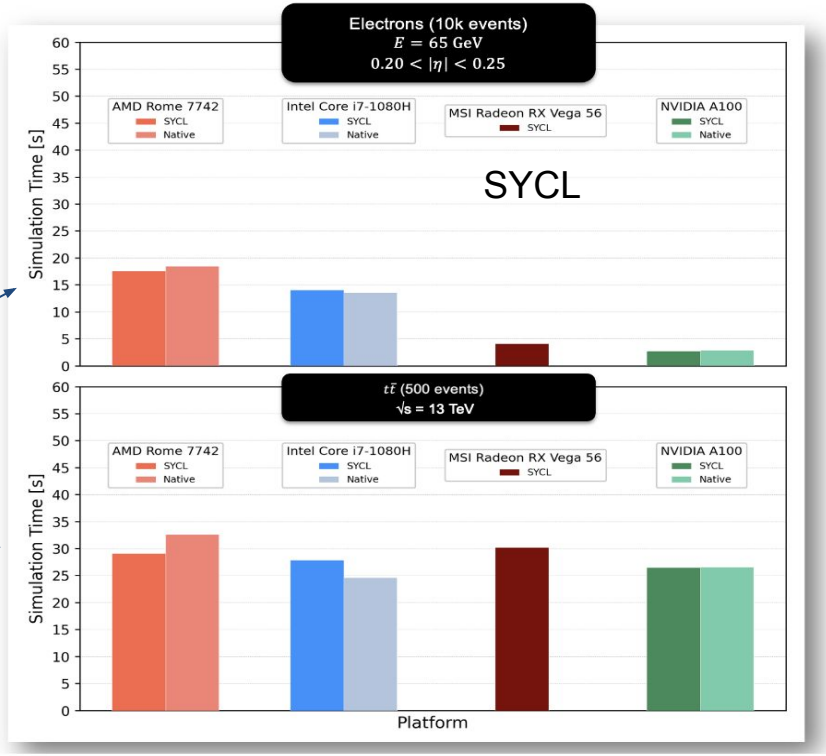
Ported to Kokkos, SYCL, std::par, OpenMP (in progress)

Same source code runs on four different platforms
(x86 CPU, NVIDIA, AMD, Intel GPU)



Main results: (→ more details)

- NVIDIA CUDA best performance, used as reference
- Performance limited by GPU offloading overhead
  - need to increase GPU work size, e.g. batching together particles from many events
- GPU performance depends on physics process
- SYCL introduces little to no overhead
- Kokkos adds overhead particularly with AMD GPUs



U.S. DEPARTMENT OF ENERGY | Office of Science

Argonne NATIONAL LABORATORY

Brookhaven National Laboratory

Fermilab

BERKELEY LAB Bringing Science Solutions to the World

CMS Patatrack Pixel Tracking Testbed

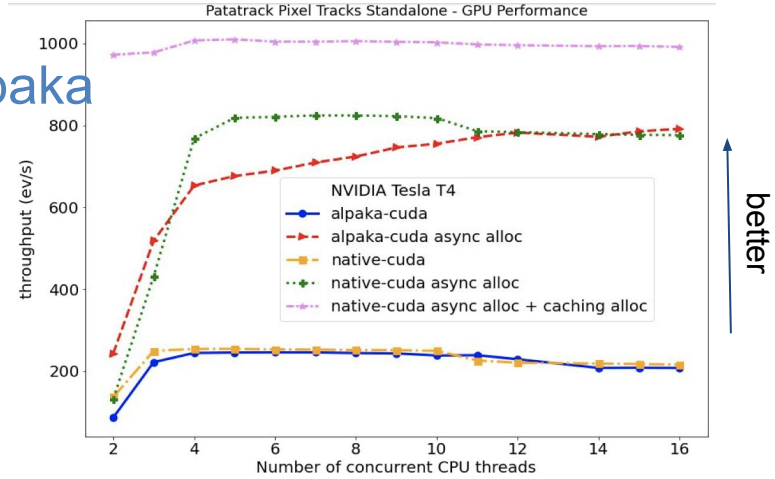
A frozen, standalone version of CMS Heterogeneous pixel track and vertex reconstruction

1. Copy the raw data to the GPU
2. Run multiple kernels to perform the various steps
3. Take advantage of the GPU computing power to improve physics
   a. fit the track parameters (Riemann fit, broken line fit) and apply quality cuts
   b. reconstruct vertices

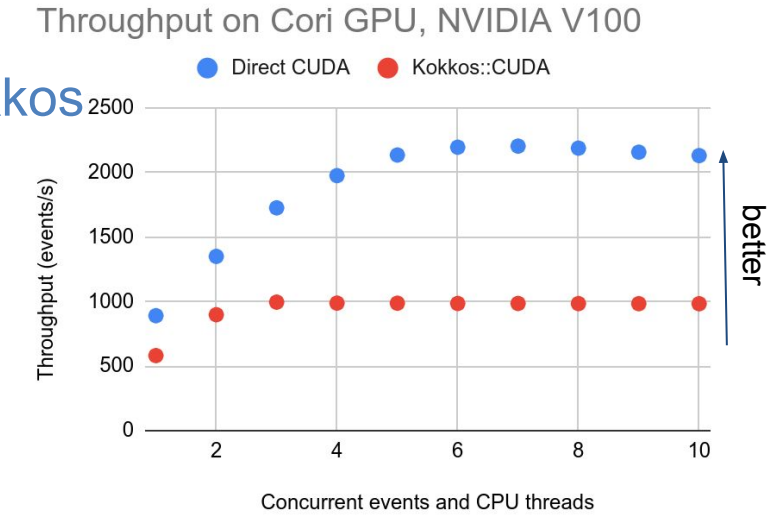
Parallelized with CUDA, HIP, and through Kokkos (plus Alpaka, @CERN)

- Run on x86 CPUs, AMD+NVIDIA GPUs

Alpaka



Kokkos



Main results: (→ more details)

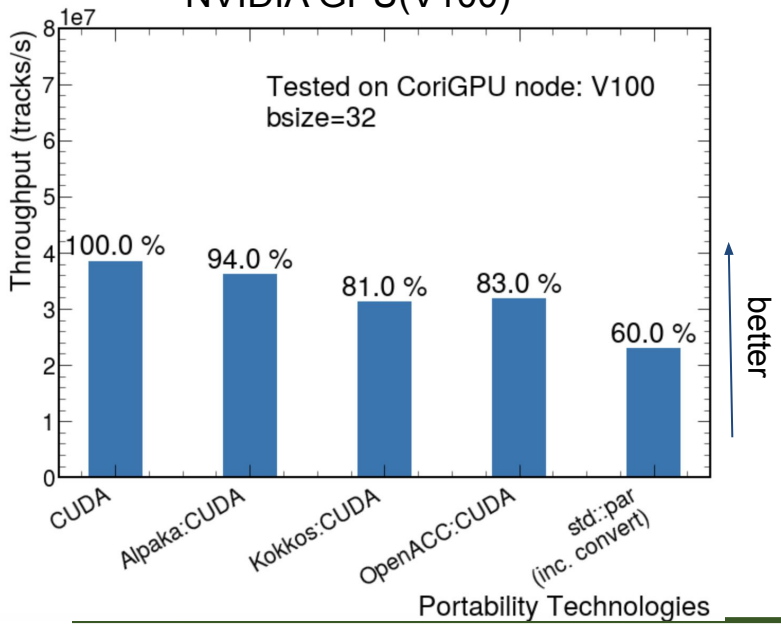
- NVIDIA V100 ~4.5x faster than Intel Skylake
- Kokkos versions 1.5-3x slower than direct CUDA
- Alpaka has similar performance to native CUDA
- Automatic memory management ("CUDA unified memory")
  3x slower than explicit GPU transfers

P2R (Propagate-to-Radial)

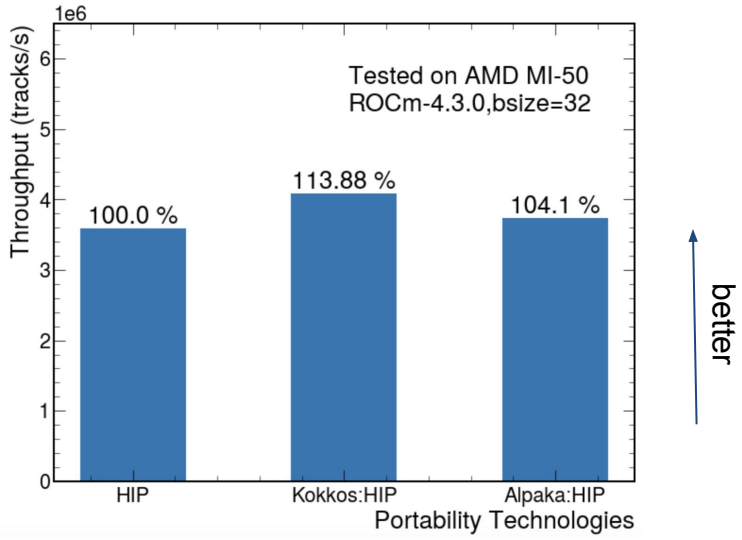
- A miniapp (~1k lines of standalone code) running “backbone” functions for track fitting
  - Kernels for track propagation and Kalman update in the radial direction
  - Extracted from a full application (mkFit)
- Intend to explore more technologies with a lightweight program
  - TBB, CUDA, HIP,, Kokkos, Alpaka, std::par, SYCL, OpenACC
- Performance compared on NVIDIA V100, AMD MI-50 and Intel GPUs
  - Same source code to run on all platforms

*All throughput exclude data-transfer time

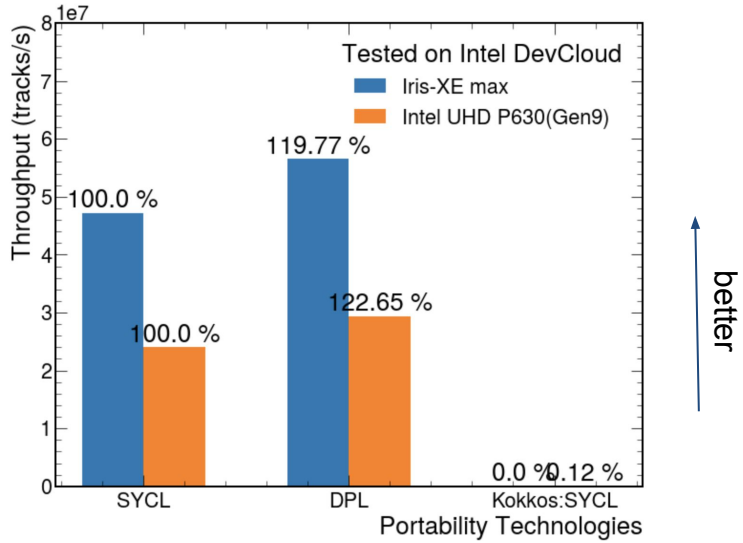
NVIDIA GPU(V100)



AMD GPU (MI-50)



Intel GPUs



Wire-Cell : LArTPC Simulation

Parallelized 2D FFT Convolution-based LArTPC Simulation:

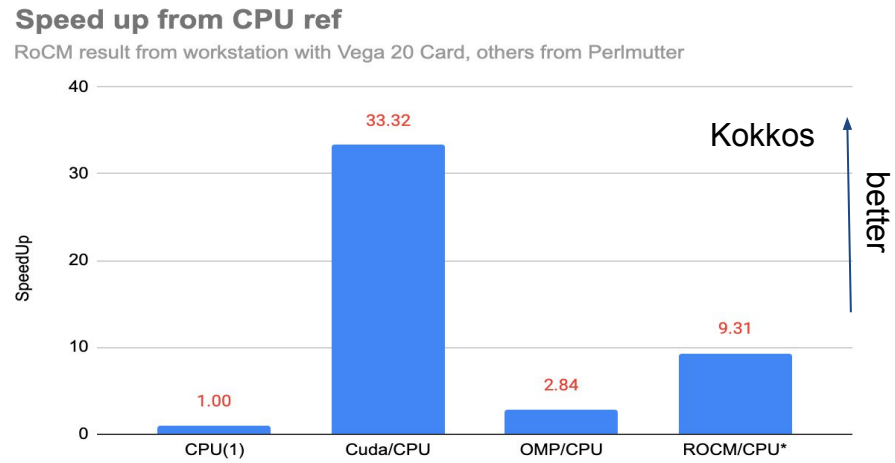
- part of Wire-Cell Toolkit (WCT) C++17 software package for Liquid Argon Time Projection Chamber (TPC) simulation, signal processing, reconstruction and visualization.

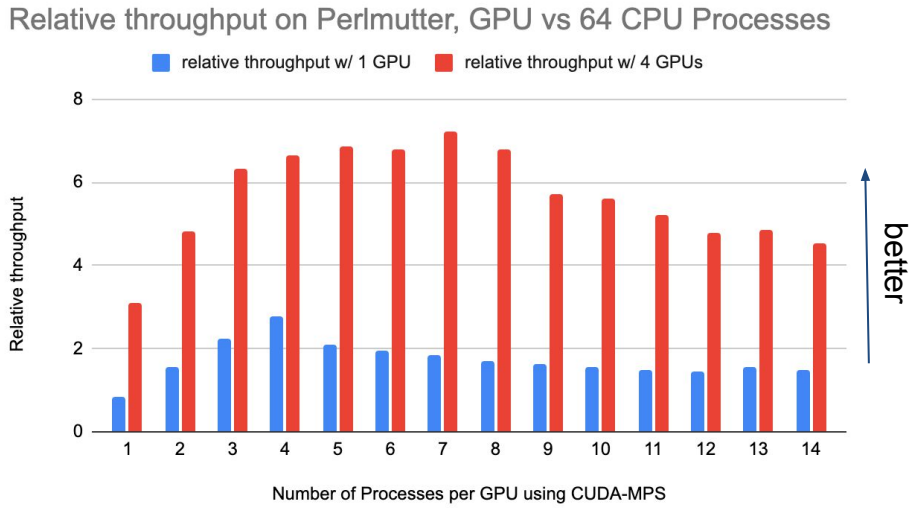
Three major steps - a representative workflow

- Rasterization: depositions ⟶ patches
- Scatter adding: patches ⟶ grid
- FFT: convolution with detector response

Main results: (⟶ more details)

- ➢ **Kokkos implementation achieved moderate speedups** cf. original CPU on multicore CPU, AMD and NVIDIA GPUs when running single process
- ➢ **Further speedups by running multiple processes** to share the GPUs
  - GPU under-utilized with 1 process





Interim Experiences With Portability Layers

Kokkos

- provides high-level abstraction of parallel hardware
- mature, well supported, good hardware support
- not the best performer

(→ more details)

Alpaka

- low-level abstraction layer (extensive templating)
- good hardware support
- close to native CPU/GPU performance

SYCL

- single source running on four hardware platforms
- actively developed, growing feature set and hardware support
- close to native CPU/GPU performance
- supported by Intel, pushing it as part of C++ standard

std::par

- simple, clean programming model → best usability
- built on C++ standard → best (hope of) long-term support by compilers, vendors
- current performance on GPU significantly inferior to other parallelization solution
- supported by NVIDIA

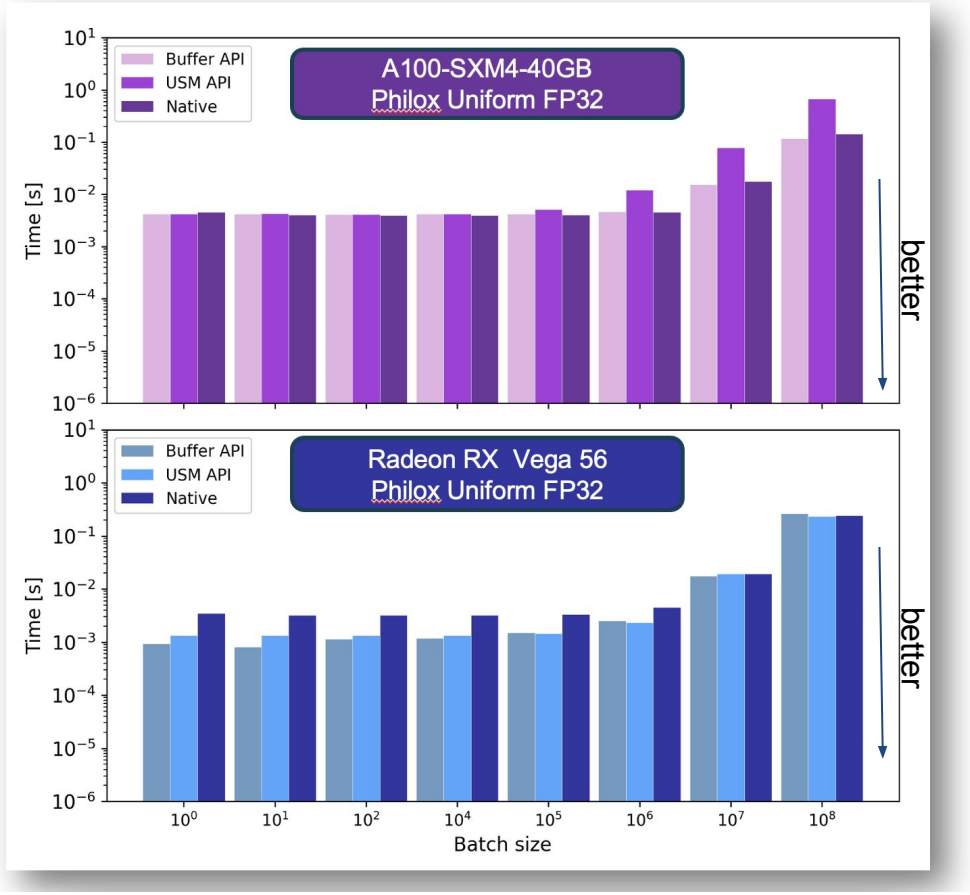
Broader Impacts: Portable Random Number Generation

HEP MC simulations need billions of high-quality, reproducible pseudo-random numbers

arxiv:2109.01329

Contributed to oneMKL a cross-platform SYCL random number generator (RNG) API

- Run our SYCL test applications **single-source** across all major CPU and GPU platforms
- relies on platform-specific RNG implementations (e.g., cuRAND and rocRAND), and SYCL interoperability API
  - **nearly zero impact on performance**
    - **future work** may include writing pure SYCL RNG kernels to achieve **sequence reproducibility** across platforms
      - at the expense of not leveraging pre-existing vendor-specific optimizations



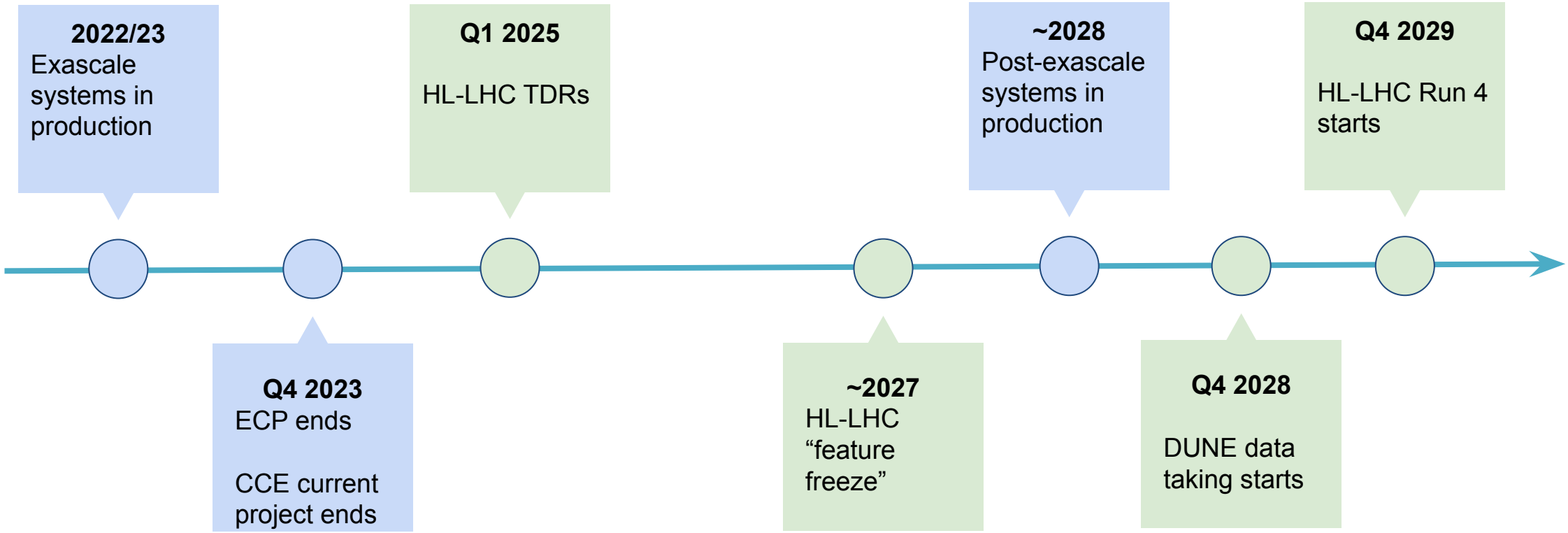
Summary

What we have learnt:

- Multiple testbeds (of different code sizes/applications/experiments) implemented with different technologies
  - Gained substantial experience with Kokkos, SYCL
- Initial GPU implementation takes the biggest effort
  - Math library support could be crucial to performance

Ideas for next Steps:

- More in these 3 days!
- New testbed applications (ACTS), more parallelization solutions
  - possibly ML/python testbeds
  - (std::par, possibly alpaka, openmp/acc)
- Sharing results with experiments, provide feedback to solutions developers
  - Presenting to collaboration meetings, HSF, ECP, CompHEP conferences
    - Snowmass whitepapers from IOS and PPS in preparation.
  - Excellent interactions with both software and hardware providers
- Summarizing experience in the metric



Thanks!

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

hep-cce@anl.gov

Cast of Characters

PPS

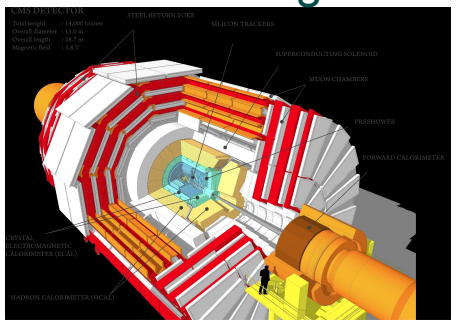
- Taylor Childers (ANL)
- Mark Dewing (ANL)
- Zhihua Dong (BNL)
- Meifeng Lin (BNL)
- Vincent Pascuzzi (BNL)
- Haiwang Yu (BNL)
- Meghna Bhattacharya (FNAL)
- Oli Gutsche (FNAL)
- Michael Kirby (FNAL)
- Matti Kortelainen (FNAL)
- Martin Kwok (FNAL)
- Kyle Knopfel (FNAL)
- Liz Sexton-Kennedy (FNAL)
- Charles Leggett (LBNL)
- Peter Nugent (LBNL)
- Yunsong Wang (LBNL)
- Sam Williams (LBNL)
- Beomki Yeo (LBNL)

not an official list: open collaboration

“Typical” HEP Experiment Workflow

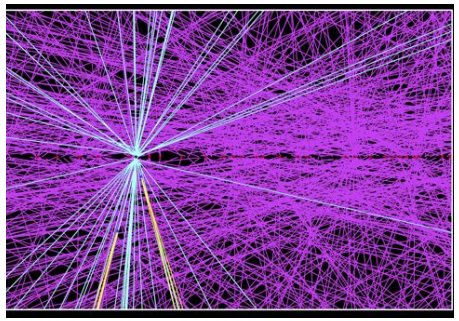
HEP-CCE

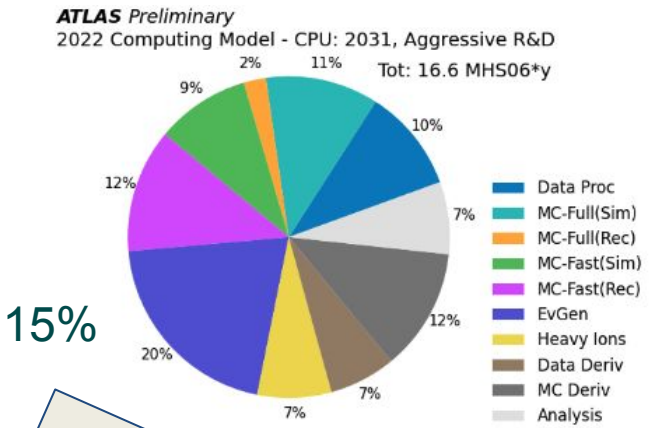
Edge Processing



Online Event Filtering

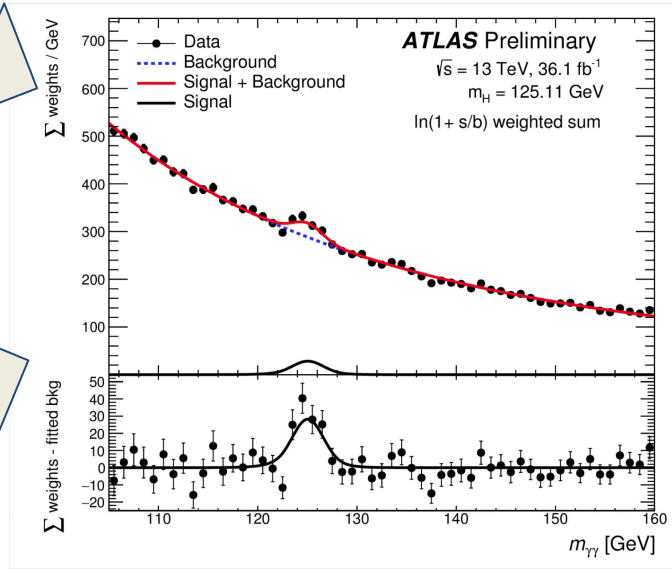
Offline Event Reconstruction





15%

Data

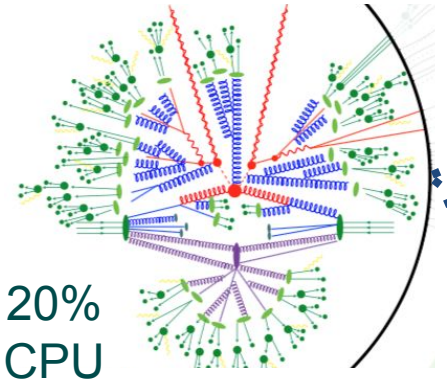


Simulation

25% CPU

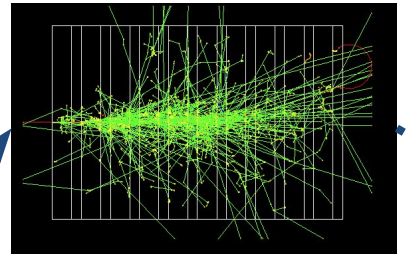
Data Analysis 10% CPU

MC Event Generation



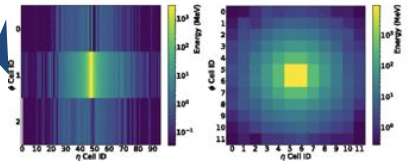
20% CPU

MC Detector Simulation



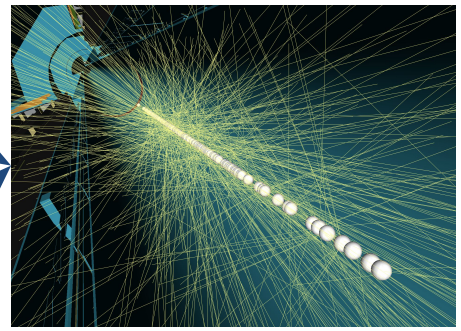
Full

20%



Fast

Pile-up + MC reconstruction



Portable Parallelization is Key for HEP

(in **bold,** systems HEP is running on or targeting)

		Accelerators				
		Intel	NVIDIA	AMD	FPGA	Other
CPU	Intel	Aurora	WLCG (HEP) Cori GPU Piz Daint Tsukuba MareNostrum		Tsukuba	
	AMD		WLCG (HEP) Perlmutter	Frontier El Capitan		
	IBM		Summit Sierra MareNostrum			
	Arm		Alps			Astra*
	Fujitsu					Fugaku

- Amazon Graviton2
- Google Cloud TPU
- Microsoft Azure
- Intel DevCloud

x86+NVIDIA main target now

Will NVIDIA Grace change the equation?

Details on PPS studies

Testbed: ATLAS FastCaloSim

ATLAS Calorimeter simulation measures the energy deposition of O(1000) particles after each collision

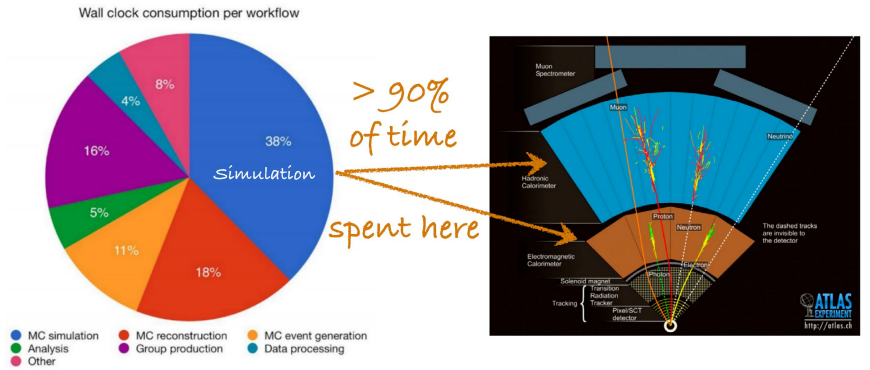
Full detailed simulation uses Geant4

- very slow due to complex LAr Geometry

Fast calorimeter simulation uses parametrization

- less accurate, but much faster than Geant4

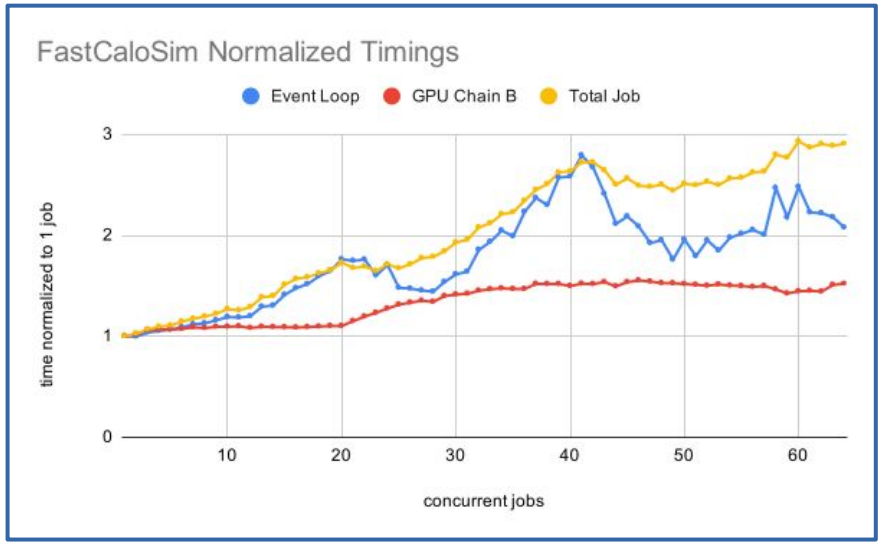
Calorimeter-dominated



FastCaloSimV2: a relatively self-contained code base for fast ATLAS parametrized calorimeter simulation

Initial CUDA port from BNL group

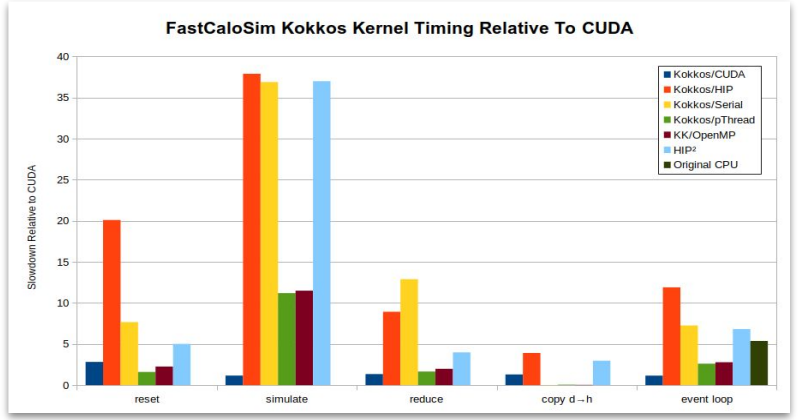
- modify/flatten data structures (eg Geometry) to offload to GPU
- multi-stage CUDA kernels to generate histograms
- current efficiency hampered by small work sizes
- need to use more particles or gang events



FastCaloSim : Results

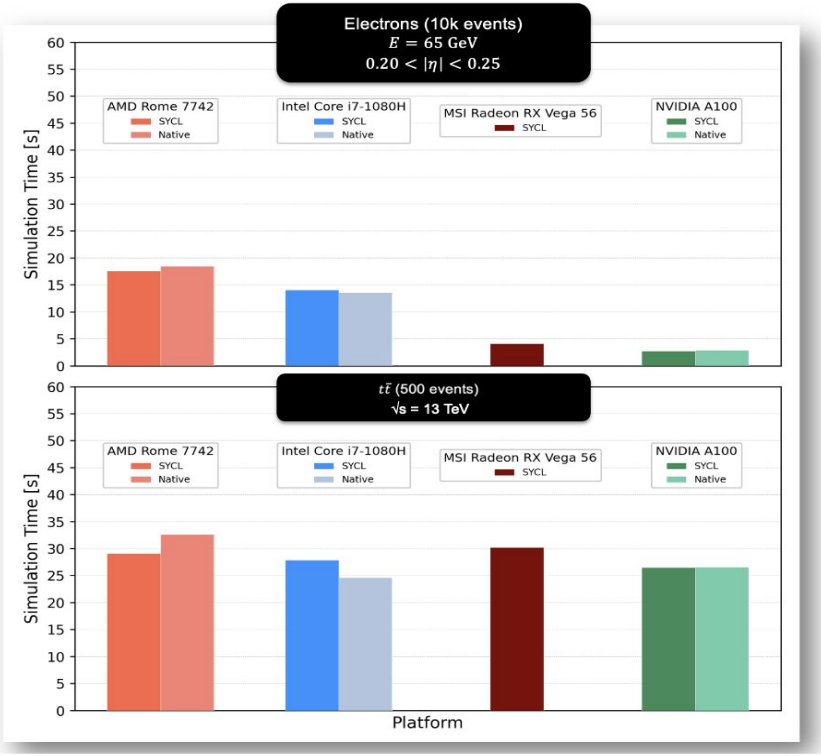
Kokkos

- exercised all backends: CUDA, HIP, Intel, pThread, OpenMP
- Kokkos/CUDA performs similarly to pure CUDA
  - 5x faster event loop for 65GeV electrons
  - 40x faster for 4TeV electrons
- increased penalties from launch latencies and memory init
- hip/AMD considerably slower than CUDA
  - Kokkos/HIP performs similarly to HIP
- Kokkos/OMP has 2.5x perf of original CPU at 12 threads



SYCL

- Ten 'runs' of single-electron and top quark pair production simulations
  - AMD CPU host backend (TBB, on OpenCL)
  - Intel CPU with OpenCL backend
- ~4x faster than CPU for single electrons when executed using GPU offload
- Top quark simulations achieve no gains on GPU due to lack of inter event parallelism and run-time loading of parametrizations on host (more secondaries)



Same source runs on 4 different platform

CCE/PPS: CMS Patatrack Pixel Tracking

A frozen, standalone version of CMS Heterogeneous pixel track and vertex reconstruction

- https://github.com/cms-patatrack/pixeltrack-standalone/
- reconstruct pixel-based tracks and vertices on the GPU
- leverage existing support in CMSSW for threads and on-demand reconstruction
- minimize data transfer

Copy the raw data to the GPU

Run multiple kernels to perform the various steps

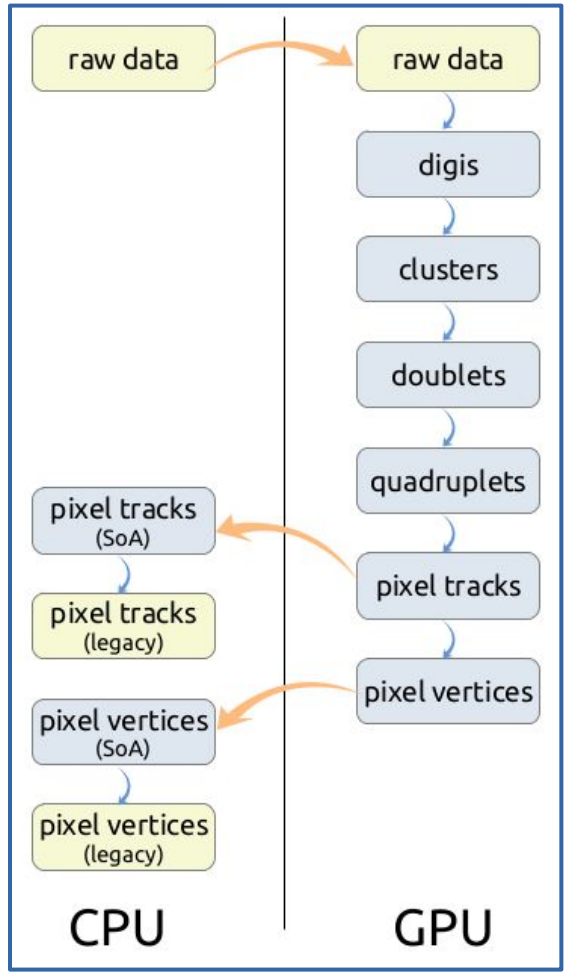
- decode the pixel raw data
- cluster the pixel hits (SoA)
- form hit doublets
- form hit quadruplets (or ntuplets) with a Cellular automaton algorithm
- clean up duplicates

Take advantage of the GPU computing power to improve physics

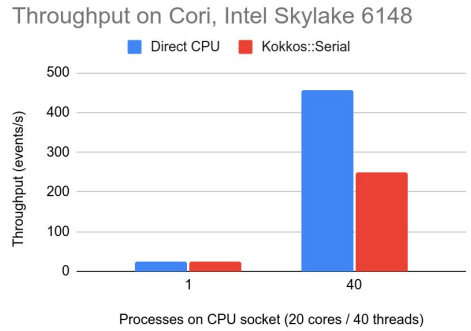
- fit the track parameters (Riemann fit, broken line fit) and apply quality cuts
- reconstruct vertices

Copy only the final results back to the host (optimised SoA)

- convert to legacy format if requested



Patatrack : Results



- Versions:
  - Direct: CPU, CUDA, HIP
  - Kokkos: CPU (Serial, POSIX Threads), CUDA, HIP
  - Alpaka: CPU (Serial, TBB), CUDA
    - Developed by CERN group, first results shown in ACAT21 poster
- Snapshot of performance comparison of direct vs Kokkos versions on Cori GPU
  - Intel Xeon Gold 6148 (Skylake, 20 cores, 2 threads/core) + NVIDIA V100

Running 1-thread processes	Direct CPU	Kokkos Serial
1 process (node free)	25.2 ± 0.4 events/s	23.9 ± 0.4 events/s
40 processes (full socket)	460 ± 10 events/s	260 ± 10 events/s

1 process	Direct CUDA	Kokkos CUDA
1 concurrent event	891 ± 5 events/s	582 ± 6 events/s
3 concurrent events	1725 ± 4 events/s	996 ± 4 events/s
7 concurrent events	2202 ± 9 events/s	985 ± 1 events/s

- Showed also in vCHEP21 that the throughput with CUDA Unified Memory was about 3x smaller than with explicit memory management

P2R status

- Completed a set of consistent measurement for NVIDIA GPU (V100)
  - Expanding measurements to AMD/Intel GPUs
- Plan:
  - Gain more experience with Intel GPUs
  - Consolidate measurements (esp. with CPU results)
  - Gain understanding of results via profiling

	Implementations								
	TBB	CUDA	HIP	Kokkos	Alpaka	Std::par (PSTL)	SYCL	DPL	OpenACC
NVDIA									
AMD									
Intel									
CPU									

Completed
In progress
Under investigation
Not started

CCE/PPS: Wire-Cell Toolkit

Much of DUNE software is based on LArSoft, which is single-threaded and has high memory usage.

Wire-Cell Toolkit (WCT) is a new standalone C++ software package for Liquid Argon Time Projection Chamber (TPC) simulation, signal processing, reconstruction and visualization.

- Written in modern C++ (C++17 standard)
- Follows data flow programming paradigm
- Supports both single-threaded and multi-threaded execution with the choice determined by user configuration.
  - MT graph execution supports pipelining, more than one "event" may be in flight through the flow graph.
- Runs from stand-alone command line program or from a LArSoft module.

WCT includes central elements for DUNE data analysis, such as signal and noise simulation, noise filtering and signal processing

- CPU intensive; currently deployed in production jobs for MicroBooNE and ProtoDUNE
- Some algorithms may be suited for GPU acceleration

Preliminary CUDA port of the signal processing and simulation modules show promising speedups

Wire-Cell : LArTPC Simulation

2D Convolution based LArTPC Simulation:

- satisfying data-simulation consistency
- more computing workload than 1D version
- large simulation samples needed for AI/ML

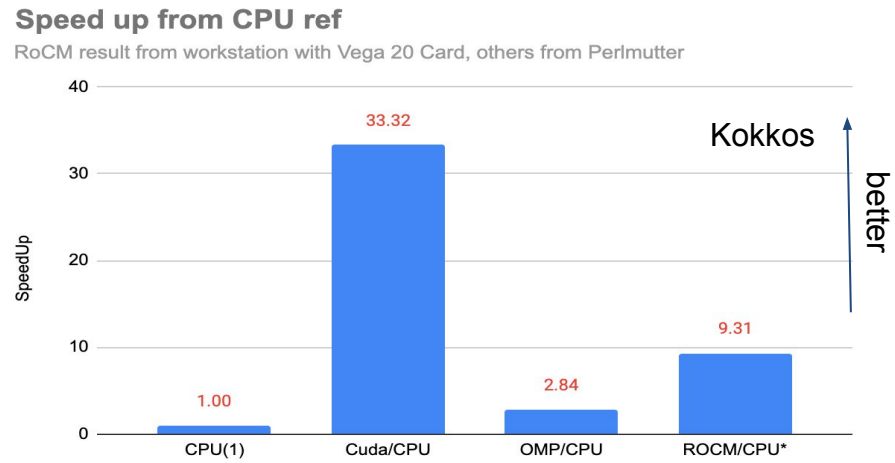
Three major steps - a representative workflow

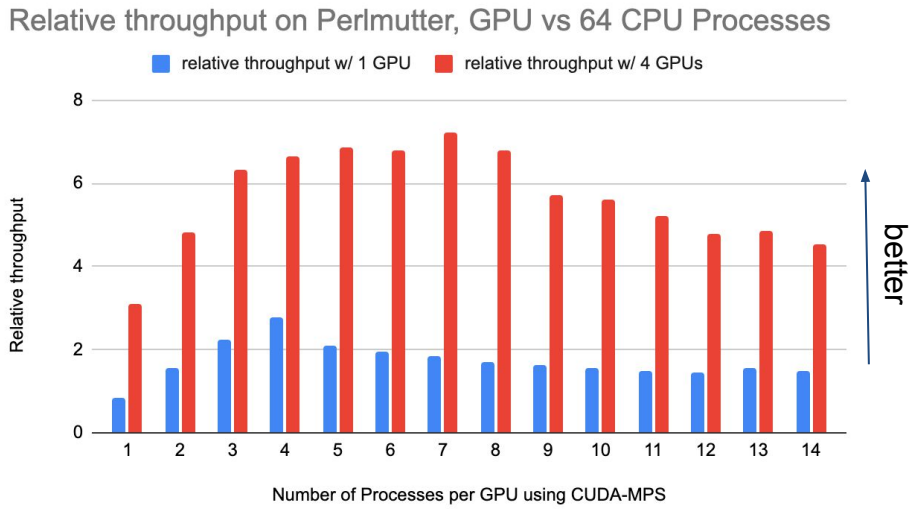
- Rasterization: depositions ⟶ patches (small 2D array, ~20×20), # depo ~100k for cosmic ray event
- Scatter adding: patches ⟶ grid (2D array, 10k×10k)
- FFT: convolution with detector response

➢ **Kokkos implementation achieved moderate speedups** cf. original CPU on multicore CPU, AMD and NVIDIA GPUs when running single process

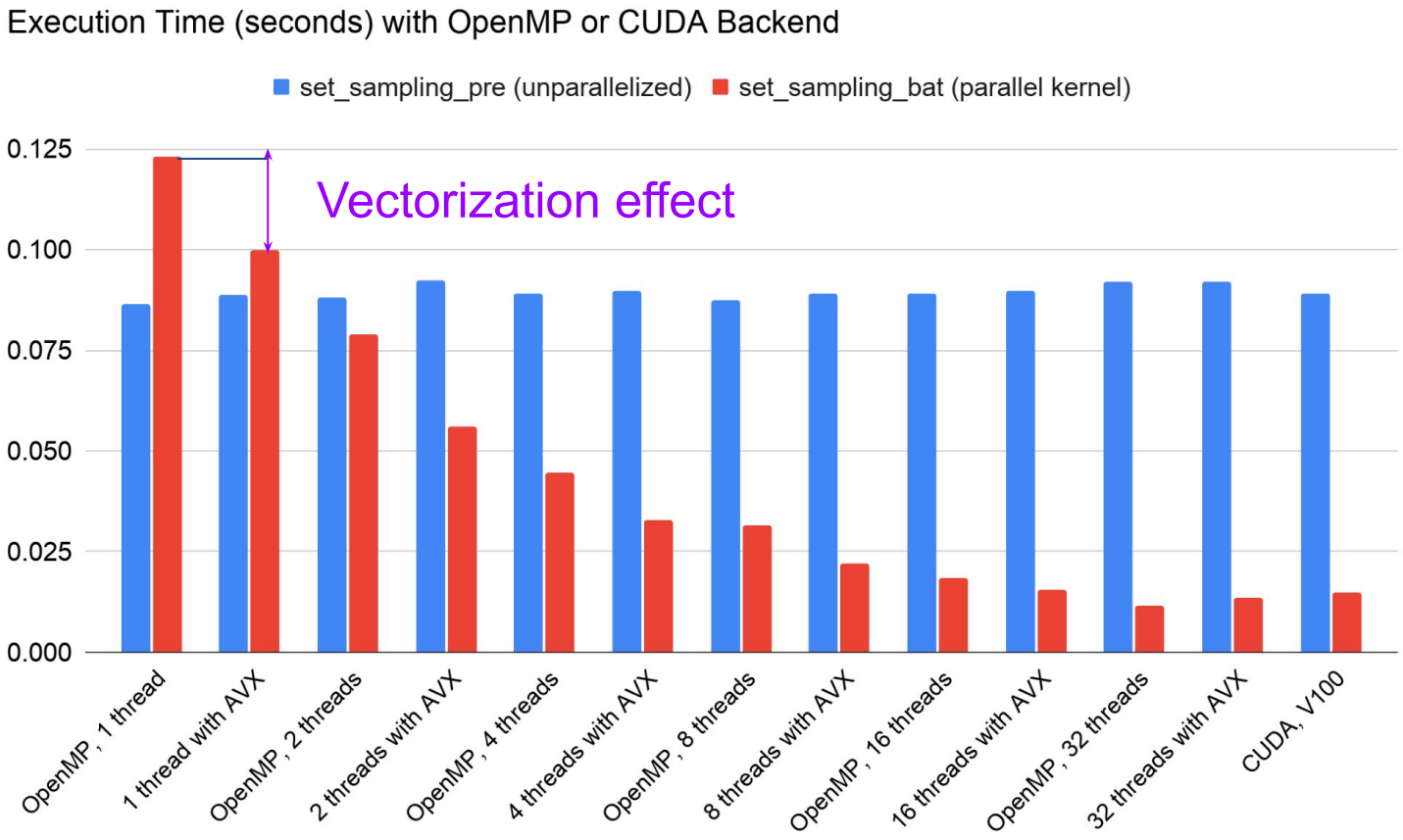
➢ **Further speedups by running multiple processes** to share the GPUs (HTC mode)

- GPU under-utilized with 1 process





Performance Evaluation of Kokkos Implementation in wire-cell-gen



- Initial Kokkos implementation based off previous CUDA implementation
- Code now runs on both multi-core CPU and NVIDIA GPU
- CPU SIMD vectorization also enabled; moderate performance gain
- GPU run time is not better than multi-threaded CPU
  - **Not enough work for the GPU**
  - **Execution time now dominated by data initialization (unparallelized)**

Test platform: 24-core AMD Ryzen Threadripper 3960X CPU and one NVIDIA V100 GPU

Kokkos: Interim Experiences

- High level programming model
  - Could be able to give reasonable performance out of the box on new architectures different from CPU vector units or GPUs
- Backends for NVIDIA, AMD and Intel GPUs, pThreads and OpenMP, Serial CPU
- APIs of earlier versions have been very stable
- Responsive developer community
- Depending on complexity of code, speed can approach that of native backend
  - but usually falls short as complexity and feature use increases
- Current challenges for use in HEP data processing frameworks
  - Requires a compiled runtime library that supports exactly one device architecture
  - CPU Serial backend is thread safe but not thread efficient (one mutex to rule them all)
    - Efficiency is being improved
  - Provides multidimensional array data type, but no special support for structured data
    - I.e. no help for crafting (Ao)SoAs, jagged arrays
  - No unified, portable interface for FFT algorithms
    - Such interface is being worked on

SYCL : Interim Experience and Feedback

C++-based API makes translation/code-conversion relatively straightforward

- Single-source (CPU, GPU code together)
- dpct (CUDA, HIP -> SYCL) conversion tool

DAG-based runtime satisfies inter-kernel dependencies (buffers)

- USM requires more explicit control from developer, but generally more performant

Integrates well with existing Makefile and CMake projects

- Compile SYCL code separately as libraries and link
- No need to recompile full stack

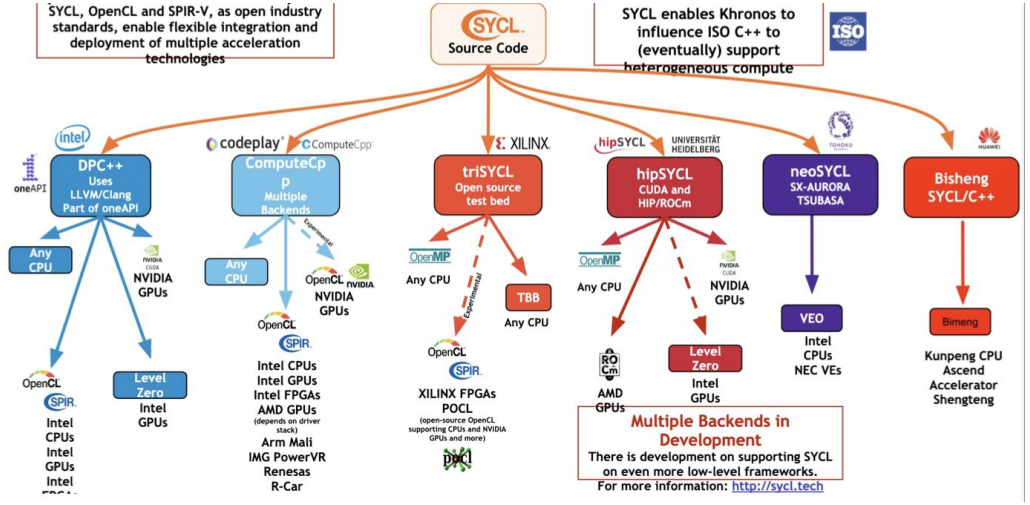
Demonstrated ability to run same source on four major vendor hardware

- Even without OCL or Level-Zero backends
- No experience yet with FPGAs

Numerous new features in 2020 specification (tested)

- Built-in optimized parallel reductions
- Work-group and sub-group algorithms for efficient parallel operations between work-items
- Sub-devices (currently limited to CPU with OCL but could prove extremely useful)
- Atomic operations aligned with C++20
- Improved interoperability for more efficient acceleration of third-party libraries (open or proprietary)

Still growing ecosystem (as of 31/10/21)



std::par : Preliminary Investigations

- NVIDIA nvc++ compiler is new and undergoing continuous development
  - can't compile ROOT yet
  - not well integrated with cmake - requires wrapper scripts to fix
  - some things work in standalone examples don't work in more complex environments with multiple shared libraries built with different compilers
  - could not exercise multicore backend
- Offers very interesting upgrade / sidegrade path
  - CPU -> GPU and multicore
  - GPU (CUDA) -> CPU/multicore
- Very simple changes to CUDA code
  - requires memory allocation on host by nvc++ for USM
  - kernel launch syntax
- Not as performant as CUDA
  - impact of USM? thrust? immature compiler?
  - also slower build time
- Similar speed to original CPU
  - sometimes slightly faster!