High Energy Physics
Center for Computational Excellence

Peter van Gemmeren (ANL), Charles Leggett (LBNL), Saba Sehrish (FNAL)
for the HEP-CCE project

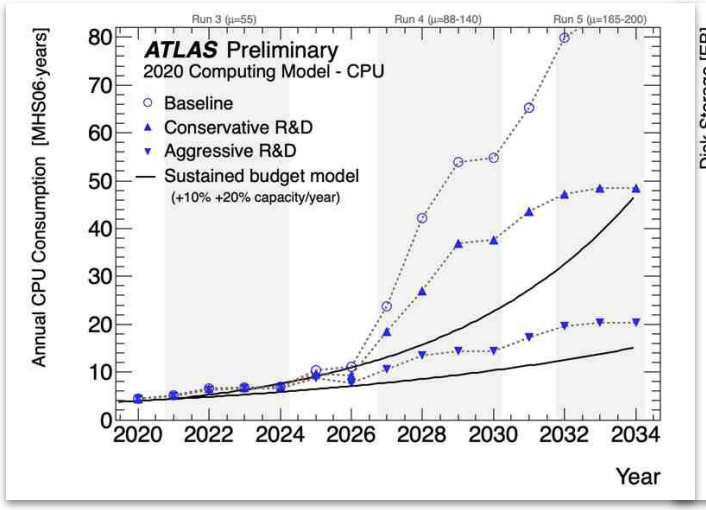
HSF - Compute Accelerator Forum
February 9 2022

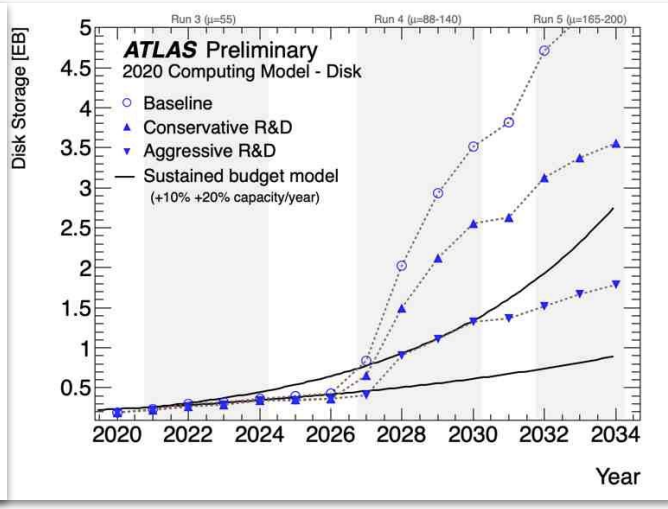
Why a HEP Center for Computational Excellence?

HEP computing resource challenges

10x data, 10x complexity @ HL-LHC

- **in 2030,** LHC experiments will need
  - **O(100) PFlops/s** sustained,
  - **O(1) Exabyte/year**



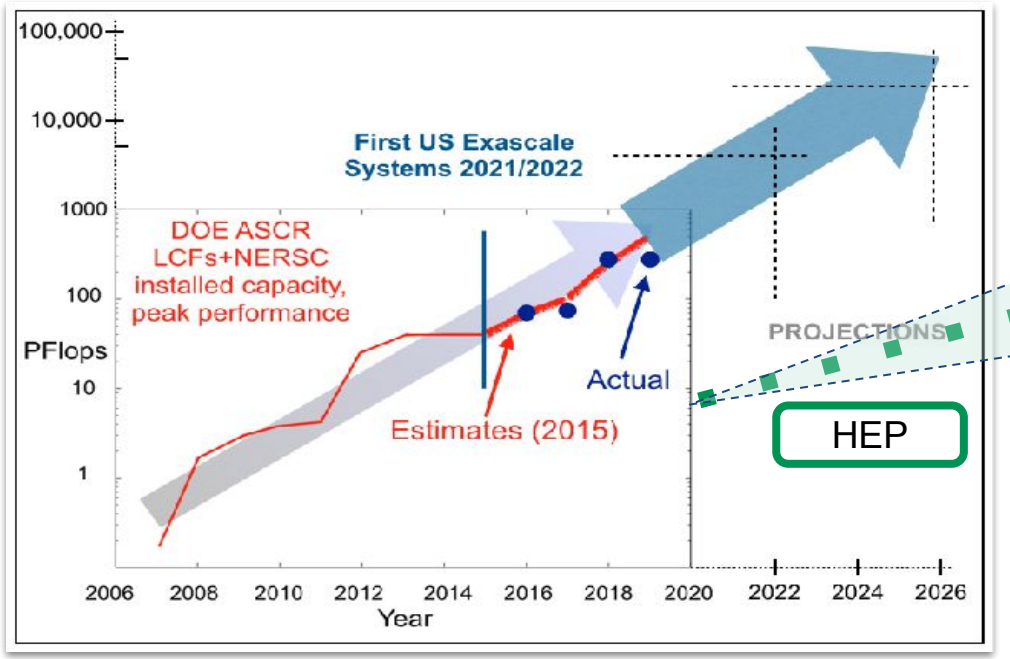


HEP long-term investment in HPC

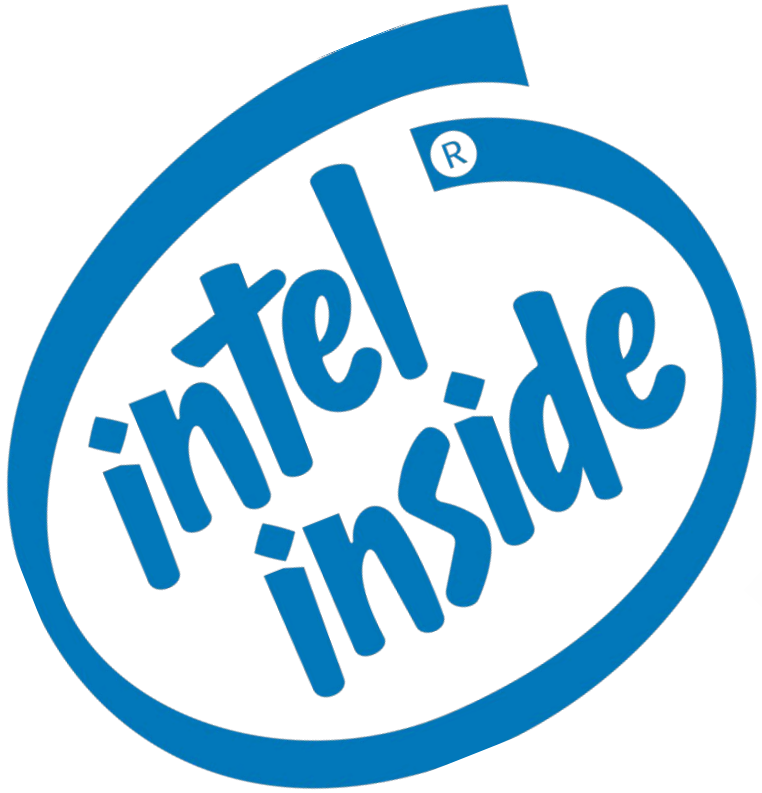
- Platform of choice for Cosmic Frontier
- LHC experiments among top users of NERSC Cori (~10% cycles in 2021)

Challenges:

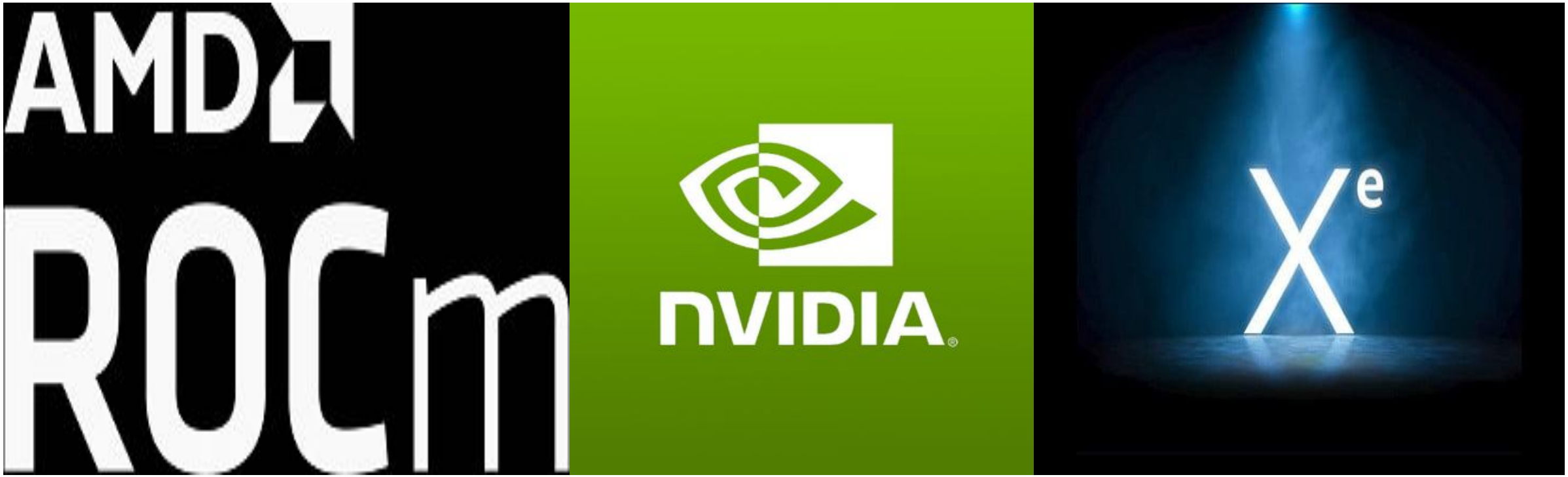
- Run multiple HEP workflows at O(10)PF/s sustained on multiple heterogeneous Exascale systems
- Match HEP I/O requirements to HPC filesystems and networking



The HEP Computing Universe Today



The HPC Computing Universe Today



HEP is (slowly) Embracing Heterogeneous Computing

Challenges:

Hundreds of computing sites (grid clusters + HPCs + clouds)

Hundreds of C++ kernels (several million LOC)

Hundreds of data objects (dynamic, polymorphic)

Hundreds of non-professional developers (domain experts)

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

What is HEP-CCE?

Three-year (2020-2023) pilot project

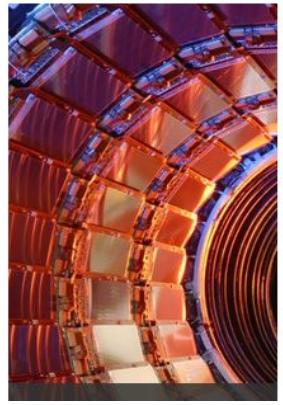
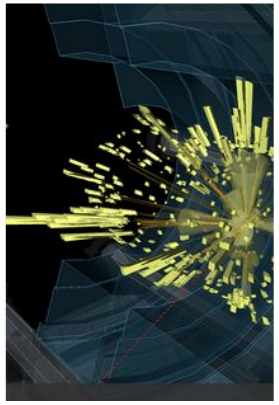
- Develop **practical** solutions to port hundreds of kernels to multiple platforms
- Collaborate with HPC & networking communities on **data-intensive** use cases

1. PPS: Portable parallelization strategies
   - exploit massive concurrency
   - portability requirements
2. IOS: HEP I/O and HPC storage issues
   - new data models (memcpy-able, SOA,...)
   - fine-grained I/O, event batching (XPU offloading)
3. EG: Optimizing event generators
4. CW: Running complex workflows on HPCs

Open collaboration

https://indico.fnal.gov/category/1053/

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


Four US labs, six experiments, ~12 FTE over ~35 collaborators. Salman Habib PI, Paolo Calafiura co-PI

Handoff to Peter and Saba for IOS

Portable Parallelization Strategies

Investigate a range of software portability solutions:

- Kokkos / Raja / Alpaka
- SYCL / dpc++ / hipSYCL
- OpenMP / OpenACC
- std::parallel::execution (std::par)

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

Port a small number of HEP testbeds to each portability solution

- Patatrack Pixel Tracking (CMS) → arXiv:2008.13461, arXiv:2104.06573
- p2r (CMS) → whitepaper
- WireCell Toolkit (DUNE)
- FastCaloSim(ATLAS) → arXiv:2103.14737
- ACTS

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

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

Make recommendations to the experiments

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

CCE/PPS: Software Support Chart

	OpenMP Offload	Kokkos	dpc++ / SYCL	HIP	CUDA	Alpaka
NVIDIA GPU			*codeplay and intel/llvm*			
AMD GPU		*experimental (feature complete)*	*via hipSYCL and intel/llvm*			
Intel GPU		*prototype*		HIPLZ: *very early development*		*prototype*
CPU						
Fortran						
FPGA						*possibly via SYCL*

Supported

Under Development

3rd Party

Not Supported

time to add python!

Platform support still a moving target: this chart is updated often!

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?

Validation

Aesthetics

- compatibility with C++ standards

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

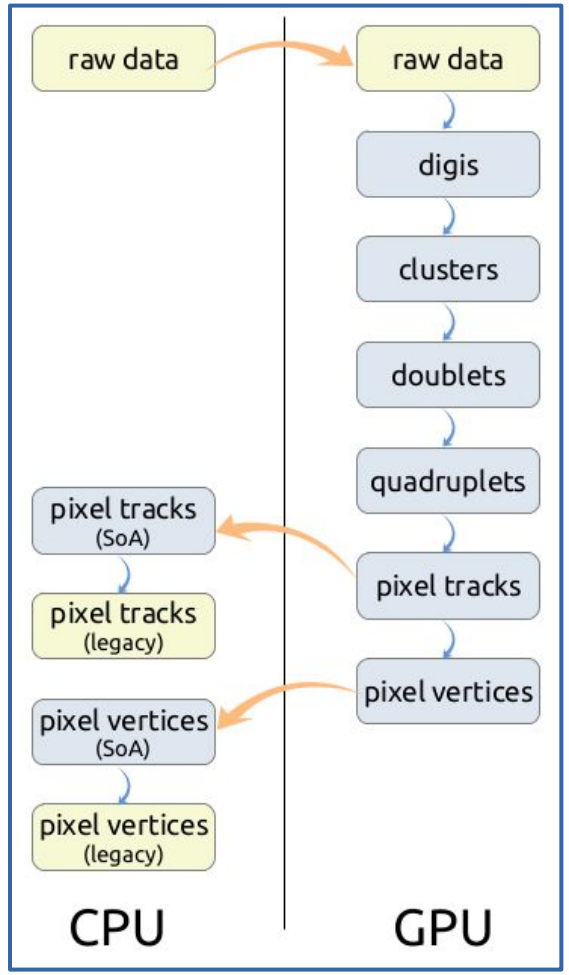
- decode the pixel raw data
- cluster the pixel hits (SoA)
- form hit doublets
- form hit quadruplets (or ntuplets) with a Cellular automaton algorithm
- clean up duplicates

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

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

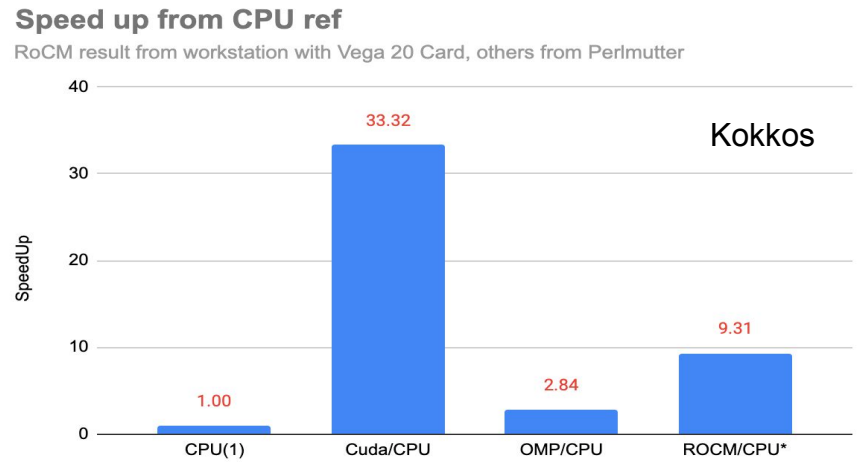
Three major steps - a representative workflow

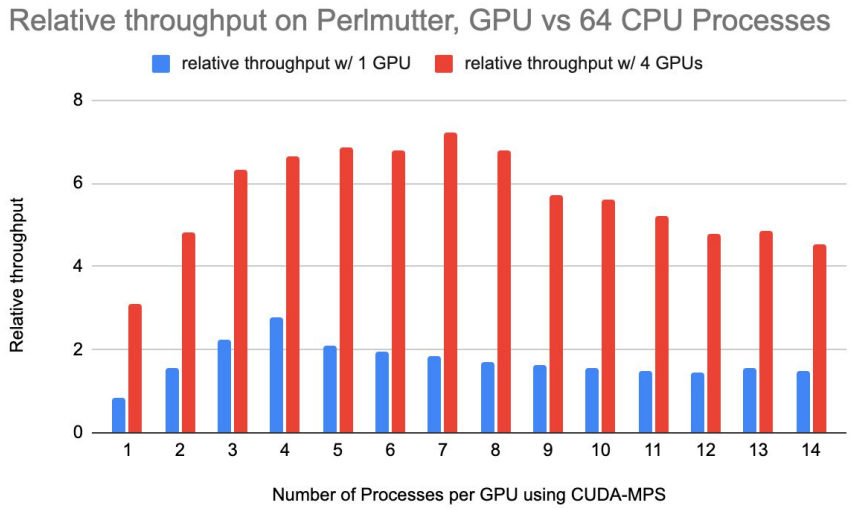
- Rasterization: depositions ⟶ patches (small 2D array, ~20×20), # depo ~100k for cosmic ray event
- Scatter adding: patches ⟶ grid (2D array, 10k×10k)
- FFT: convolution with detector response

➢ **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)

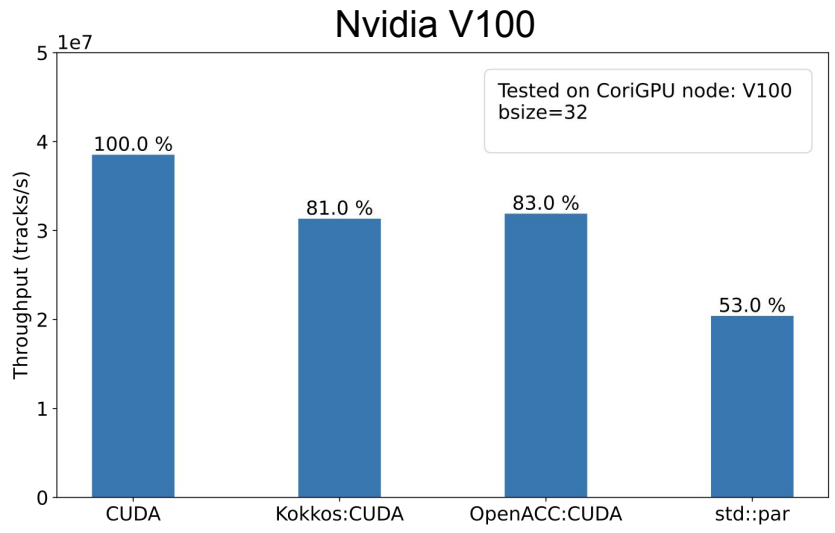
- GPU under-utilized with 1 process

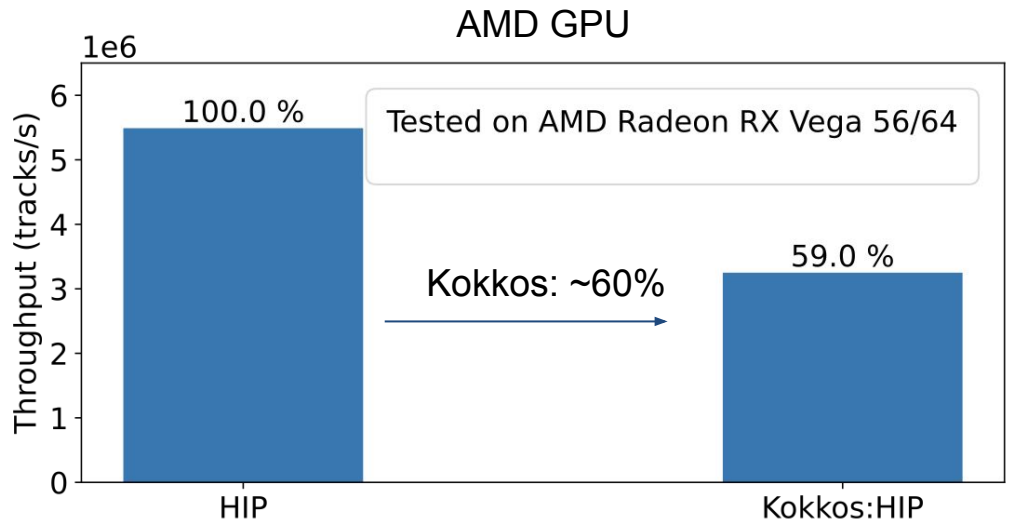




P2R (Propagate-to-Radial)

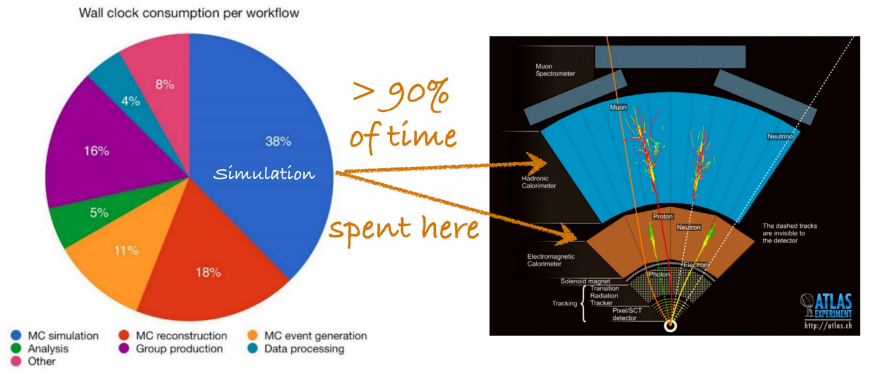
- A miniapp (~1k lines of standalone code) running “backbone” functions for track fitting
  - Kernels for track propagation and Kalman update in the radial direction
  - Extracted from a full application (mkFit)
  - Intend to explore more technologies with a lightweight program
- Versions implemented:
  TBB, CUDA, Kokkos, OpenACC, std::par (nvc++), HIP, Alpaka
- Performance compared on NVIDIA V100 and AMD Vega 64
  - Throughput measured with the input data already on the GPU





CCE/PPS: ATLAS FastCaloSim

Calorimeter-dominated



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

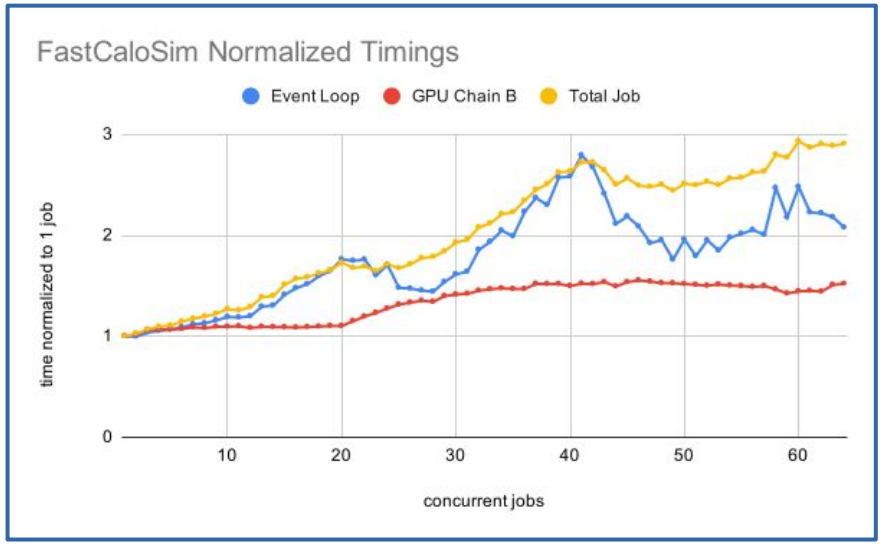
Fast calorimeter simulation uses parametrization

- less accurate, but much faster than Geant4

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

Initial CUDA port from BNL group

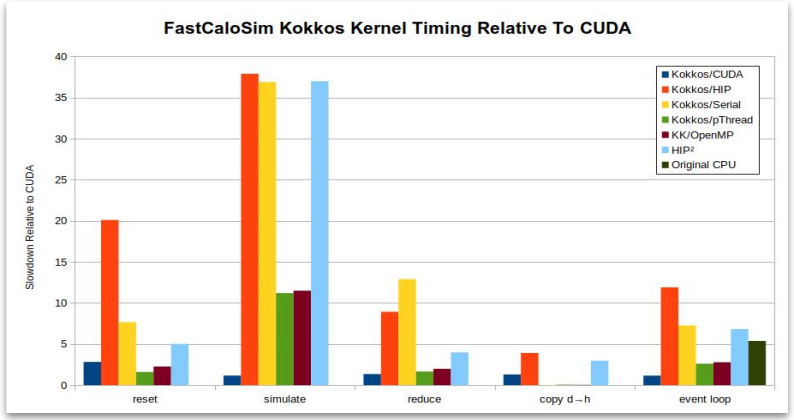
- modify/flatten data structures (eg Geometry) to offload to GPU
- multi-stage CUDA kernels to generate histograms
- current efficiency hampered by small work sizes
- need to use more particles or gang events



FastCaloSim : Results

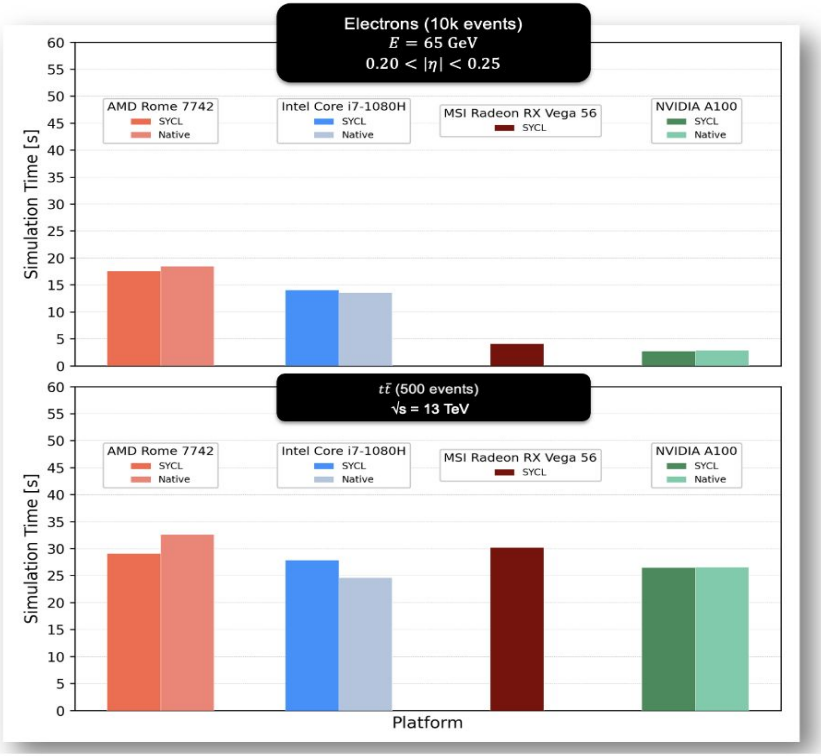
Kokkos

- 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

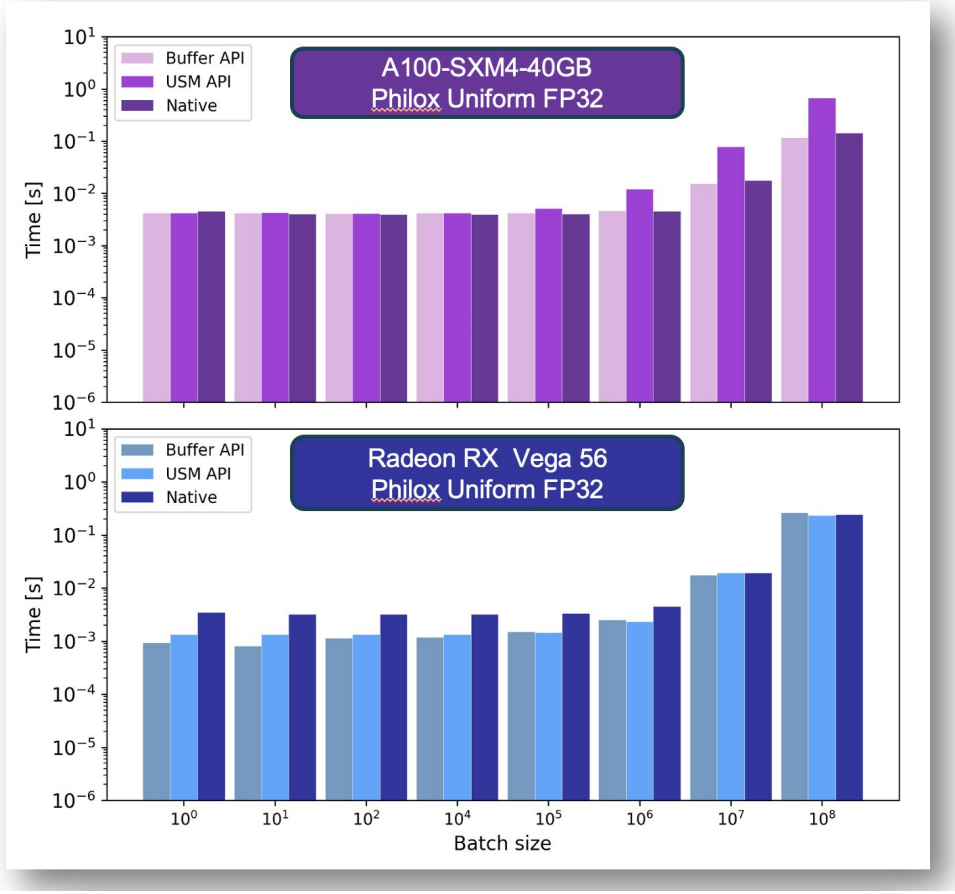
Broader Impacts: Portable Random Number Generation

arxiv:2109.01329

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

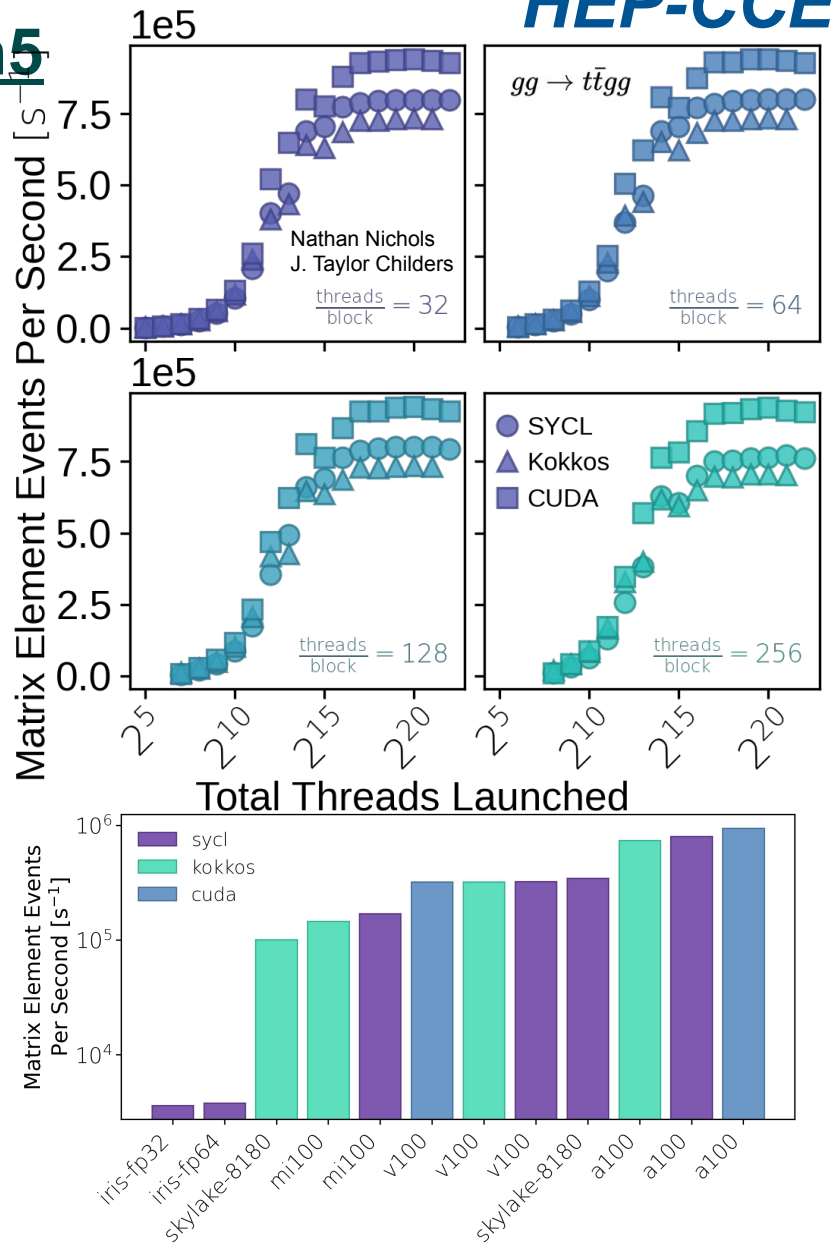
Contributed to oneMKL a cross-platform SYCL random number generator (RNG) API

- 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



Event Generators: Platform Independent MadGraph5

- MadGraph5 is one of the primary event generators used by ATLAS & CMS and will be responsible for generating a large portion of simulated collisions in the next decade.
- MadGraph5 developers (Olivier Mattelaer, UCLouvain & CERN) are working to add CUDA support.
- HEP-CCE is adding backends that use Kokkos, Intel OneAPI, and Sycl, broadening reach to architectures expected on future Exascale machines.
- This effort benefits from an Aurora Early Science Project award received by the ATLAS group at Argonne which provides early access to Intel GPUs and support from Intel/LCF experts.
- This work aims to enable running MadGraph matrix element calculations on NVidia, Intel, and AMD GPUs
- The codes are being developed here, still a work in progress: https://github.com/madgraph5/madgraph4gpu



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

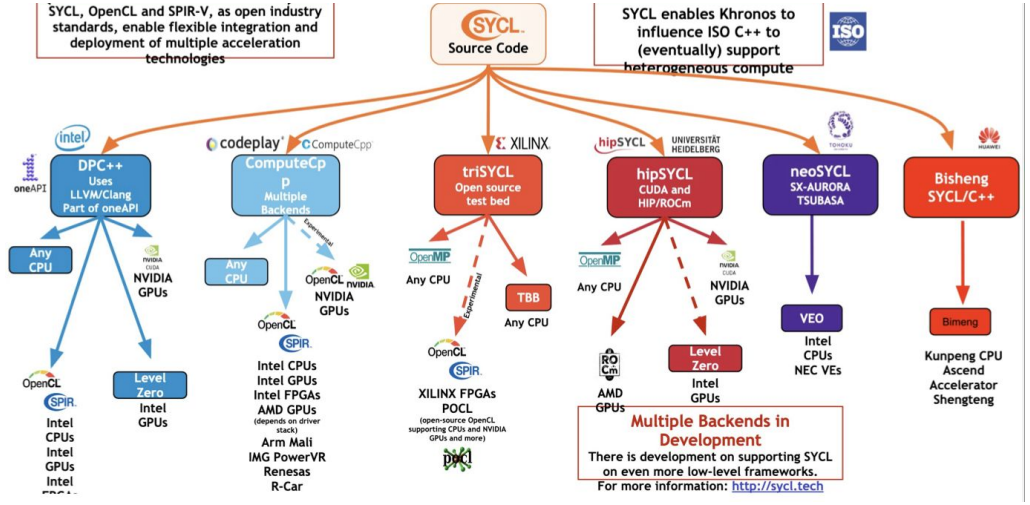
Demonstrated ability to run same source on four major vendor hardware

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

Numerous new features in 2020 specification (tested)

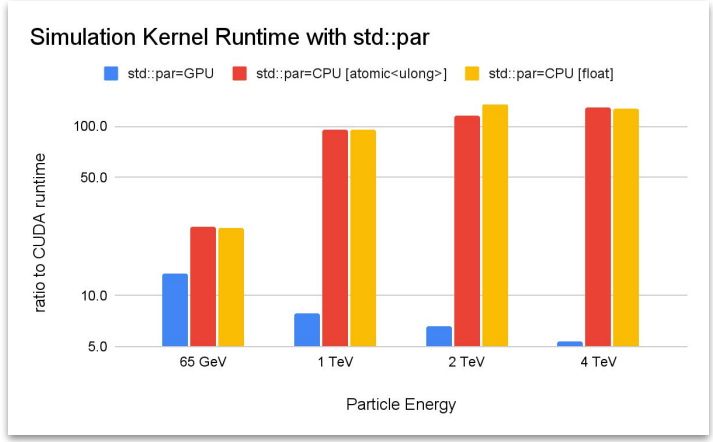
- Built-in optimized parallel reductions
- Work-group and sub-group algorithms for efficient parallel operations between work-items
- Sub-devices (currently limited to CPU with OCL but could prove extremely useful)
- Atomic operations aligned with C++20
- Improved interoperability for more efficient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

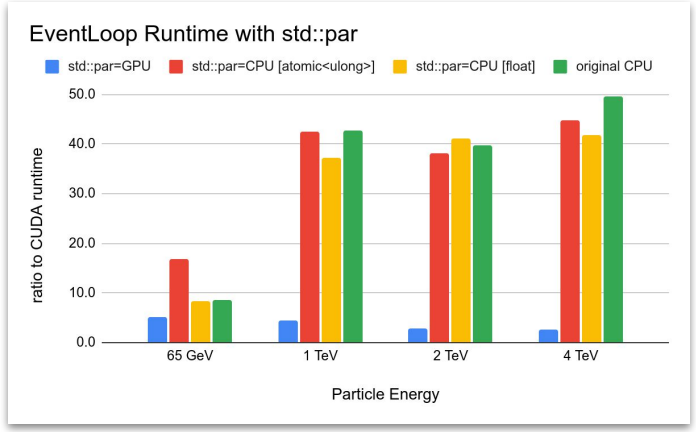
Still growing ecosystem (as of 31/10/21)



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!





Status of Ports for Testbeds

	Kokkos	SYCL	OpenMP	Alpaka	std::par
Patatrack					
WireCell					
p2R					
FastCaloSim					
ACTS					

Outlook

Prioritized PPS and IOS for initial work, EG and CW are taking off

- Significant developments and results, with many valuable lessons learned

Project work presented at multiple venues and conferences with positive feedback, good response from experiments

- Excellent interactions with both software and hardware providers

Ramping up effort (big challenge these days!)

- Recruiting more developers from the experiments and DOE/ASCR experts
- Several hires of postdocs and summer students

Potential for significant impact on the experiments

- HEP-CCE will produce ***strategies*** tested on ***prototypes***
- Production-level implementations will require direct experiment involvement

Thanks!

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

hep-cce@anl.gov

Special thanks to **Paolo Calafiura, Taylor Childers, Stefan Hoeche, Matti Kortelainen, Martin Kowk, Meifeng Lin, Vince Pascuzzi, Shane Snyder, Peter van Gemmeren** for the material they contributed. All mistakes and misrepresentations are mine only.

Cast of Characters

PPS

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

not an official list:
open collaboration

EG

- Enrico Bothmann (Goettingen),
- Taylor Childers (ANL),
- Walter Giele (FNAL),
- Stefan Hoeche (FNAL),
- Joshua Isaacson (FNAL),
- Max Knobbe (Goettingen)

IOS

- Doug Benjamin (BNL)
- Jakob Blomer (CERN)
- Suren Byna (LBNL)
- Philippe Canal (FNAL)
- Matthieu Dorier (FNAL)
- Chris Jones (FNAL)
- Kenneth Herner (FNAL),
- Patrick Gartung (FNAL).
- Rob Latham (ANL),
- Rob Ross (ANL)
- Liz Sexton-Kennedy (FNAL)
- Saba Sehrish (FNAL)
- Shane Snyder (ANL)
- Peter van Gemmeren (ANL)
- Torre Wenaus (BNL)

Portable Parallelization is Key for HEP

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

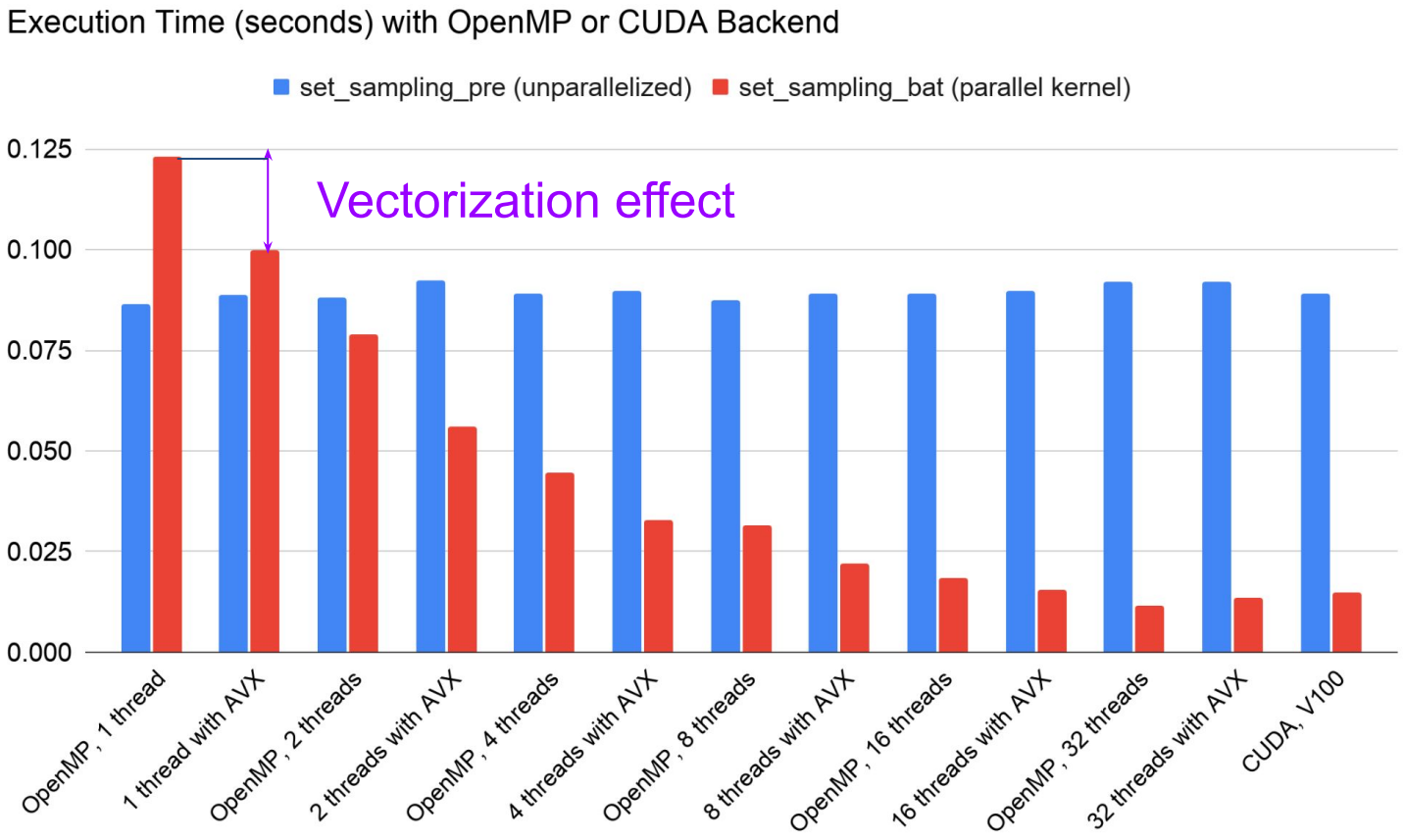
		Accelerators				
		Intel	NVIDIA	AMD	FPGA	Other
CPU	Intel	Aurora	WLCG (HEP) Cori GPU Piz Daint Tsukuba MareNostrum		Tsukuba	
	AMD		WLCG (HEP) Perlmutter	Frontier El Capitan		
	IBM		Summit Sierra MareNostrum			
	Arm		Alps			Astra*
	Fujitsu					Fugaku

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

x86+NVIDIA main target now

Will NVIDIA Grace change the equation?

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