Portability: A Necessary Approach for Future Scientific Software

Meghna Bhattacharya (Fermilab)
for HEP-CCE

Snowmass 22, Computational Frontier 1
July 20, 2022

High Performance Computing- Gen Z DOE Supercomputers

HEP-CCE

- Today's world of scientific software for High Energy Physics (HEP) powered by x86 code

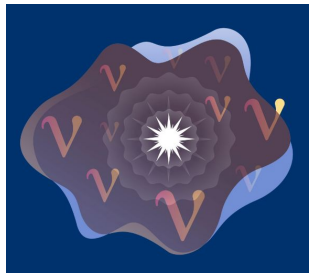
- Future HEP Experiments -
  - Order of magnitude increase in data rate
    - Data & processing complexity within existing frameworks
    - "Buy more CPUs" - not cost effective

- High Performance Computing

  - large installations of hardware using GPUs and other accelerators provide more processing power for the same energy consumption as with x86-based supercomputers.

  - multiple different GPU and CPU vendors are available to optimize the hardware to our research problems.

- Challenge - writing efficient scientific code and getting the science out a lot more difficult



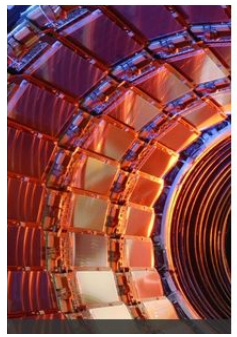
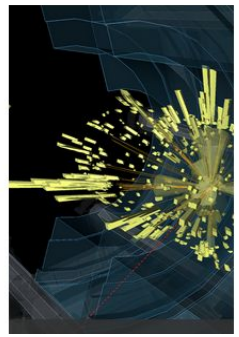
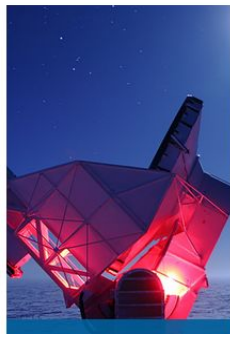
High Performance Computing (HPC) - HEP CCE Efforts

HEP-CCE

- **HEP-CCE (Centre for Computational Excellence) (2020 - 2023) 3 year pilot project**
  - **6 Experiments, 4 National labs across US**
  - **Intensity, Energy and Cosmic Frontiers**

- **Goal - Exploit features of HPCs efficiently**
  - **Develop and test strategies to overcome HEP community wide computational challenges**
    - **PPS: Portable Parallelization Strategy**
    - **IOS: I/O and Storage on HPC Platforms**
    - **EG: Event Generators**
    - **CW: Complex Workflow on HPC**

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

Argonne NATIONAL LABORATORY
BROOKHAVEN NATIONAL LABORATORY
Fermilab
BERKELEY LAB Lawrence Berkeley National Laboratory

U.S. DEPARTMENT OF ENERGY | Office of Science
Argonne NATIONAL LABORATORY
Brookhaven National Laboratory
Fermilab
BERKELEY LAB Bringing Science Solutions to the World

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

Portability requirement -

- single source code to be compiled for and executed on multiple different heterogeneous architectures with few or no changes

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

Programming Models Evaluated

- Most of the HEP codes are C++ based, so the programming models we investigate are those with good C++ support.
- **Kokkos**
  - A C++ abstraction layer (library) that supports parallel execution of the code and data management for different **host** and **accelerator** architectures. https://github.com/kokkos/kokkos/wiki
- **SYCL**
  - SYCL is a **specification** for a cross-platform C++ abstraction layer https://www.khronos.org/sycl/
  - Implementations are provided by different vendors/organizations to support different architectures.
- **OpenMP/OpenACC** https://www.openmp.org/ and https://www.openacc.org/
  - Directive-based programming models
  - **Specifications** for parallel execution on different **host** and **accelerator** architectures
- **Others**
  - **Alpaka:** C++ abstraction layer similar to Kokkos https://alpaka-group.github.io/alpaka/
  - **std::par:** language-based parallelism from the C++ 17 Standard
  - **HIP:** AMD's abstraction layer for AMD and NVIDIA backends https://github.com/ROCm-Developer-Tools/HIP

Portability Solutions: Software Support Chart

	NVIDIA CUDA*	Kokkos	Alpaka	AMD HIP	std::par	SYCL	OpenMP
NVIDIA GPU				*hipcc*	*nvc++*	*intel/llvm compute-cpp*	*nvc++ LLVM, Cray GCC, XL*
AMD GPU				*hipcc*		*hipSYCL intel/llvm*	*AOMP LLVM Cray*
Intel GPU			*prototype*	HIPLZ: *early prototype*	*oneapi::dpl*	*oneAPI intel/llvm*	*Intel OneAPI compiler*
x86 CPU				*via HIP-CPU Runtime*		*oneAPI intel/llvm computecpp*	*nvc++ LLVM, CCE, GCC, XL*
FPGA			*possibly via SYCL*	*via Xilinx Runtime*			*prototype compilers (OpenArc, Intel, etc.)*
ARM						*computecpp + pocl*	*ARM, Cray GCC, LLVM Fujitsu*

All green cells in table are potential targets for our studies.

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

* As a reference

Supported / Partially Supported

Not Supported

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

→ 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

Status of Ports for Testbeds

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

Done
Under Development
Not Started

ATLAS FastCaloSim Testbed

arXiv:2103.14737

HEP-CCE

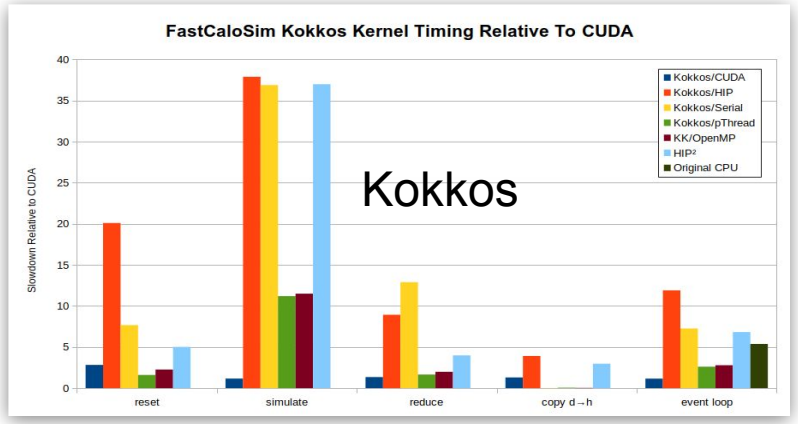
Developed parallel version (CUDA) of ATLAS parameterized calorimeter simulation

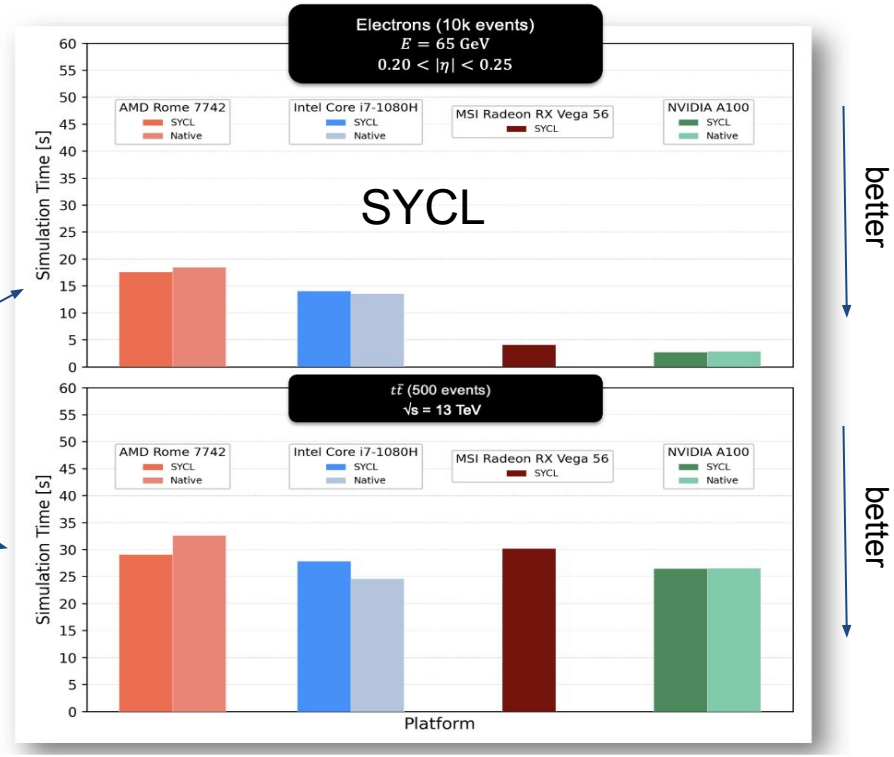
Ported to Kokkos, SYCL, std::par, OpenMP (in progress)

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

Main results:

- 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
- std::par kernels run 2-3x slower than CUDA for small kernels, but 30% faster than CUDA for large ones
  - memory ops to/from AMD **hosts** 20-50x worse than Intel





CMS Patatrack Pixel Tracking Testbed

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

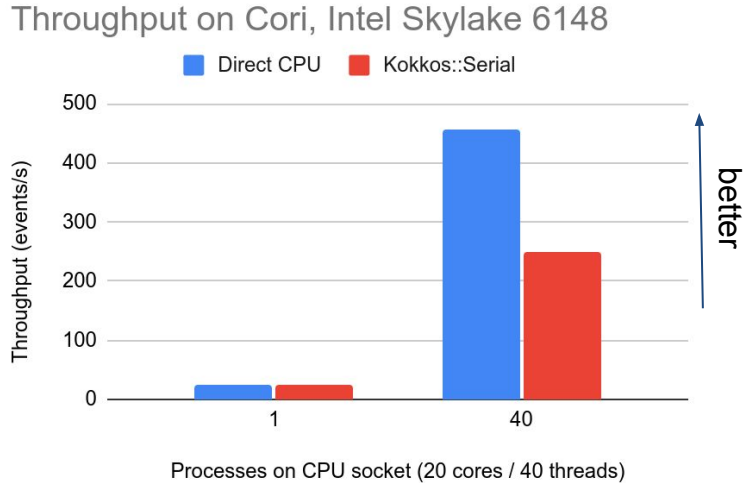
1. Copy the raw data to the GPU
2. Run multiple kernels to perform the various steps
3. Take advantage of the GPU computing power to improve physics
   a. fit the track parameters (Riemann fit, broken line fit) and apply quality cuts
   b. reconstruct vertices

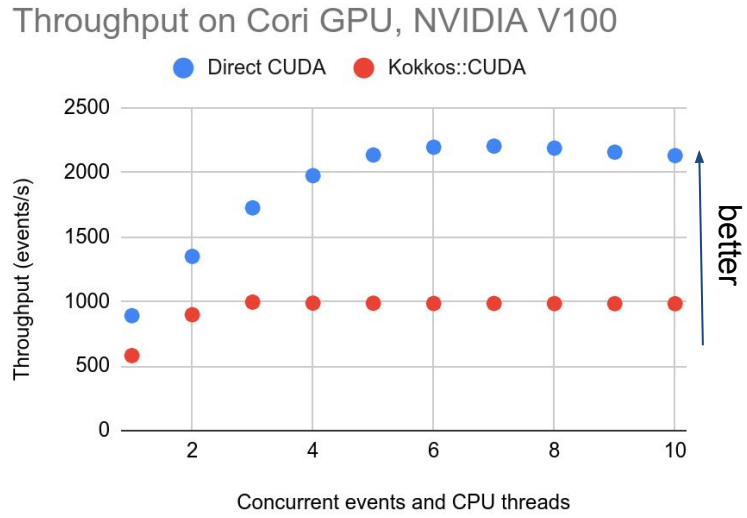
Parallelized with CUDA, HIP, and through Kokkos (plus Alpaka, @CERN)

- Run on x86 CPUs, AMD+NVIDIA GPUs

Main results:

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

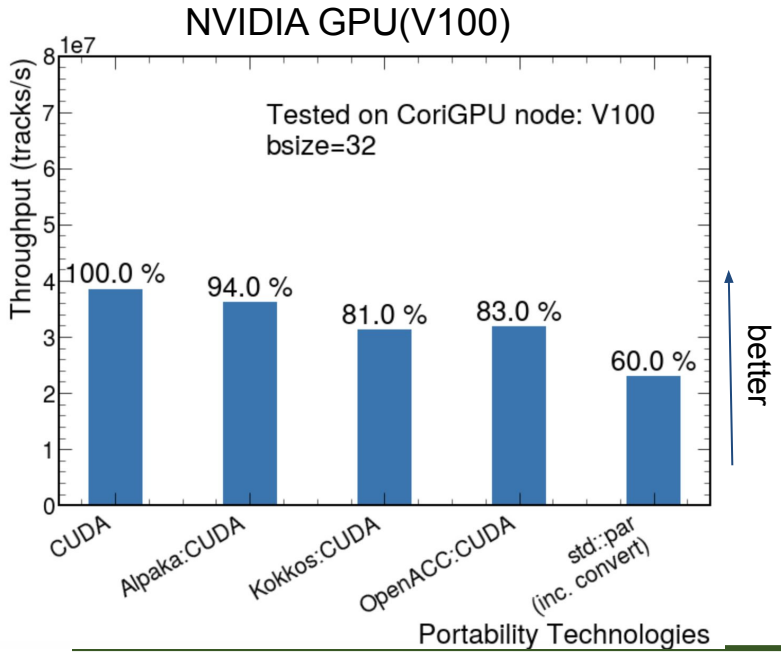


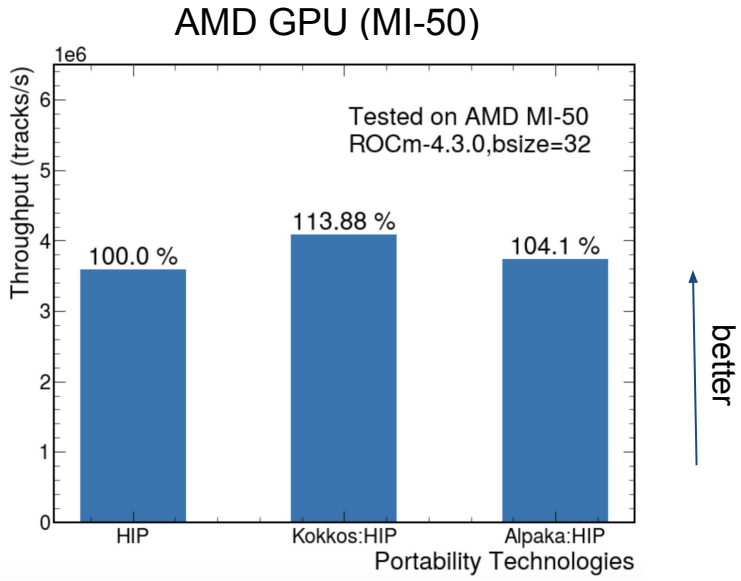


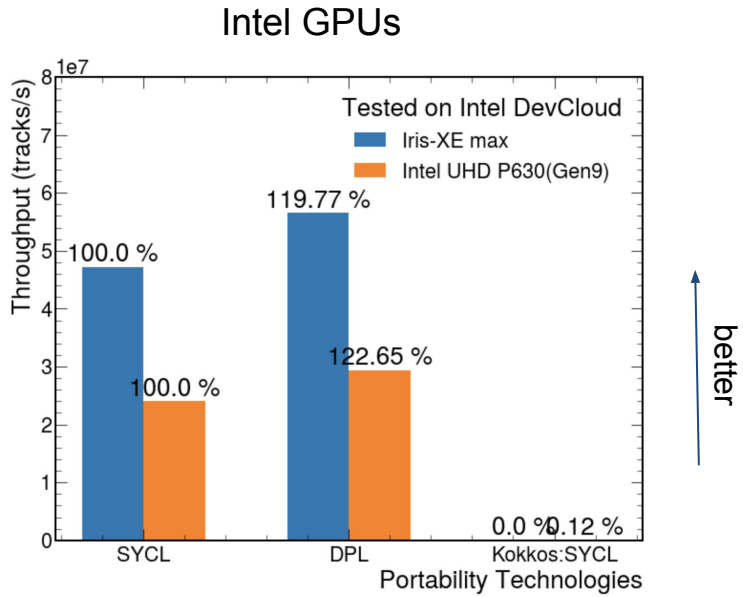
P2R (Propagate-to-Radial)

- 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







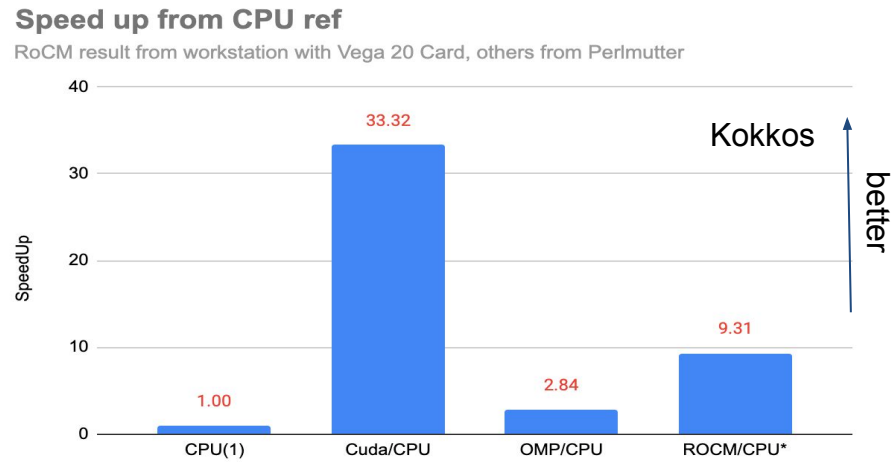
Wire-Cell : LArTPC Simulation

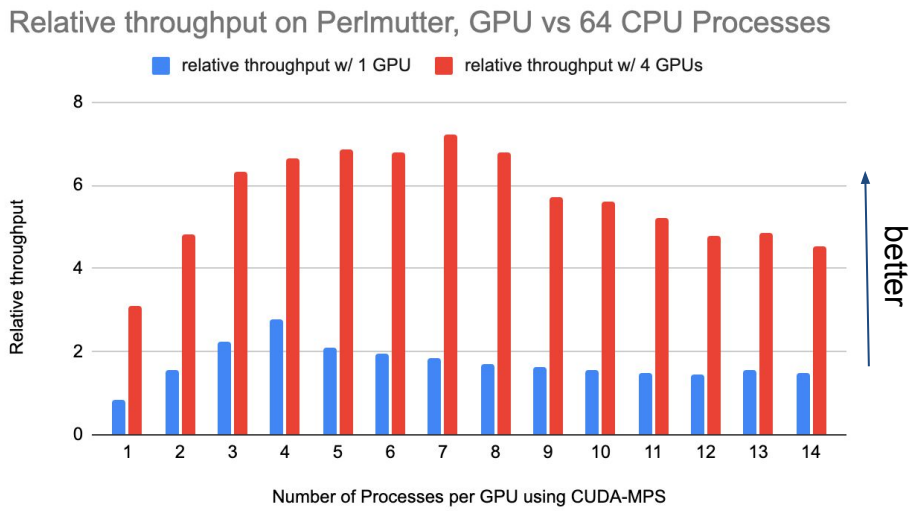
Parallelized 2D FFT Convolution-based LArTPC Simulation:

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

Main results:

- **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
  - because GPUs are under-utilized with one process
- **SYCL** implementation achieved similar performance to Kokkos on NVIDIA GPUs.
  - tests on AMD GPUs are ongoing





Conclusion

- The evolving computing landscape utilizing heterogeneous architectures (GPUs, FPGAs etc.) poses challenges for HEP workflows.
  - Development of scientific code is at a crossroad leaving the convenient era of x86 only code
  -
- Portability is a major consideration for such software adaptation.

- Community solutions for portability are needed to continue writing scientific code efficiently with a large and not always professionally trained user community

- Experiences with several representative testbeds and portable programming models indicate that
  - Different portability solutions have their own pros and cons.
  - There is an overhead in implementing the portability layers in the code, but being able to run the same code across different architectures may be worth the effort.
  - Best portability solutions may be use case dependent.

- We think without them, software development could be costly to be able to run on available hardware infrastructure

- In future, as a community, we need to request and work on portability solutions with a very low entry bar for users, maybe even as an extension to C++ standards

For More Details

- Childers, Taylor, et al. “Porting CMS Heterogeneous Pixel Reconstruction to Kokkos.” *vCHEP 2021.* arXiv:2104.06573v1. Slides.
- Dong, Zhihua, et al. “Porting HEP Parameterized Calorimeter Simulation Code to GPUs.” *Frontiers in Big Data*. arXiv:2103.14737v2. Slides.
- Kortelainen, Matti J., et al. “Performance of CUDA Unified Memory in CMS Heterogeneous Pixel Reconstruction.” vCHEP 2021. Paper. Slides.
- Pascuzzi, Vincent R., Goli, Mehdi. “Achieving Near Native Runtime Performance and Cross-Platform Performance Portability for Random Number Generation Through SYCL Interoperability.” arXiv:2109.01329
- Yu, Haiwang, et al. “Evaluation of Portable Acceleration Solutions for LArTPC Simulation Using Wire-Cell Toolkit.” *vCHEP 2021.* arXiv:2104.08265v1. Slides.
- HEP-CCE Collaboration, Portability: A Necessary Approach for Future Scientific Software, Snowmass White Paper

Backup

Interim Experiences With Portability Layers

Kokkos

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

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 inferior to other parallelization solution for small kernels, but better than CUDA for longer ones. Odd performance with AMD hosts is not understood
- supported by NVIDIA

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

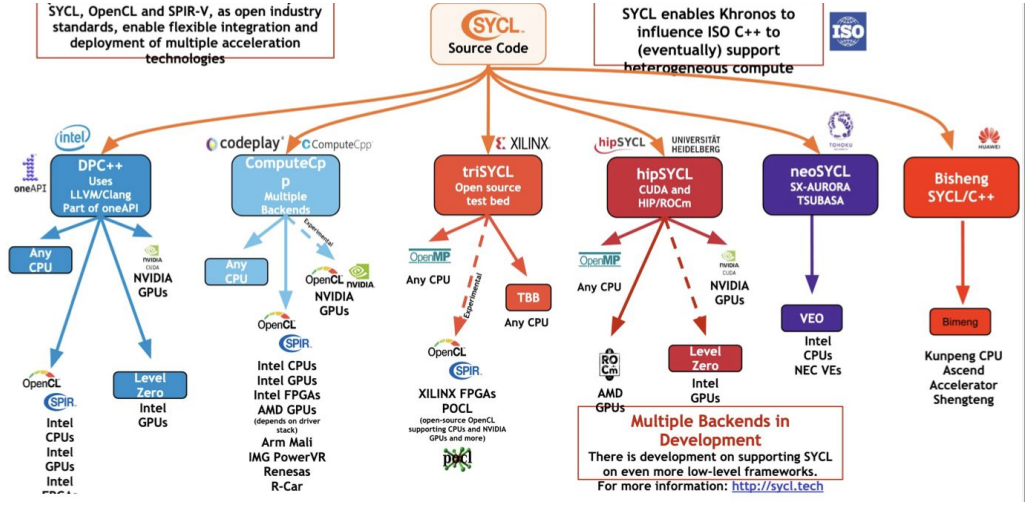
Demonstrated ability to run same source on four major vendor hardware

- 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
- Small kernels not as performant as CUDA
  - impact of USM? thrust? immature compiler?
  - also slower build time
- Large kernels sometimes 30% faster than CUDA
- Memory ops with AMD hosts much slower than Intel
- Similar speed to original CPU
  - sometimes slightly faster!