Portable Parallelization Strategies in the Era of Heterogeneous Architectures

C. Leggett for HEP-CCE

HSF WLCG Virtual Workshop on New Architectures, Portability, and Sustainability

May 11 2020

Current and Near Future Heterogeneous HPCs

Complex ecosystem

		Accelerators				
		Intel	NVidia	AMD	FPGA	Other
CPU	Intel	Aurora	Cori Piz Daint Tsukuba MareNostrum		Tsukuba	
	AMD		Perlmutter	Frontier El Capitan		
	IBM		Summit Sierra MareNostrum			
	Arm		Wombat			Astra*
	Fujitsu					Fugaku

Amazon EC2 P3
Google Cloud TPU
Microsoft Azure
Intel DevCloud

Can we make our code run everywhere without significant platform customization?

HEP Center for Computational Excellence: Mandate

Address shortcomings of current HEP computational model due to demise of Dennard scaling and order of magnitude increases of experimental data throughput across Cosmic, Energy and Intensity Frontiers.

3 year pilot project of collaborative research with DOE ASCR HPC community investigating:

- portable parallelization strategies
- fine-grained I/O and related storage issues
- optimizing event generators
- running complex workflows on HPC systems.

Address essential tasks that are needed to assess whether, and in what way, HPC systems can meet the needs of the experiments, and to study portable solutions that can be deployed across multiple platforms.

Involves close collaboration between the domain scientists, the ASCR experts and the wider set of users who work with the experimental data, to ensure that the proposed solutions do in fact properly address the needs of the experiments.

HEP Center for Computational Excellence: Mandate

Address shortcomings of current HEP computational model due to demise of Dennard scaling and order of magnitude increases of experimental data throughput across Cosmic, Energy and Intensity Frontiers.

3 year pilot project of collaborative research with DOE ASCR HPC community investigating:

- **portable parallelization strategies** ←
- fine-grained I/O and related storage issues
- optimizing event generators
- running complex workflows on HPC systems.

Focus of this talk

Address essential tasks that are needed to assess whether, and in what way, HPC systems can meet the needs of the experiments, and to study portable solutions that can be deployed across multiple platforms.

Involves close collaboration between the domain scientists, the ASCR experts and the wider set of users who work with the experimental data, to ensure that the proposed solutions do in fact properly address the needs of the experiments.

Plan for CCE-PPS

Investigate a range of software portability solutions:

- Kokkos / Raja
- SYCL / dpc++
- Alpaka
- OpenMP / OpenACC

Port a small number of HEP testbeds to each language

- Patatrack Pixel Tracking (CMS)
- WireCell Toolkit (DUNE)
- FastCaloSim (ATLAS)

Define a set of metrics to evaluate the ports, and apply them

- Ease of porting, performance, code impact, relevance, *etc*

Make recommendations to the experiments

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

Software Support Overview

	OpenMP Offload	Kokkos	dpc++ / SYCL	HIP	CUDA	Alpaka
NVidia GPU	🟩	🟩	🟢 *Intel/codeplay*	🟩	🟩	🟩
AMD GPU	🟩	🟩 *prototype*	🟧 *via hipSYCL*	🟩	🟥	🟢
Intel GPU	🟩	🟢	🟩	🟥	🟥	🟢 *very early development*
CPU	🟩	🟩	🟩	🟥	🟥	🟩
Fortran	🟩	🟥	🟥	🟧	🟧	🟥
FPGA	🟢	🟢	🟩	🟥	🟥	🟥 *possibly via SYCL*

- 🟩 Supported
- 🟢 Under Development
- 🟧 3rd Party
- 🟥 Not Supported

We are seeing rapid and ongoing development

Kokkos

Kokkos provides portability via various backends: OpenMP, CUDA, (tbb), *etc*

Single source C++, standard dependent on backend (nvcc limits to C++14)

Abstractions usually provided via C++ header libraries

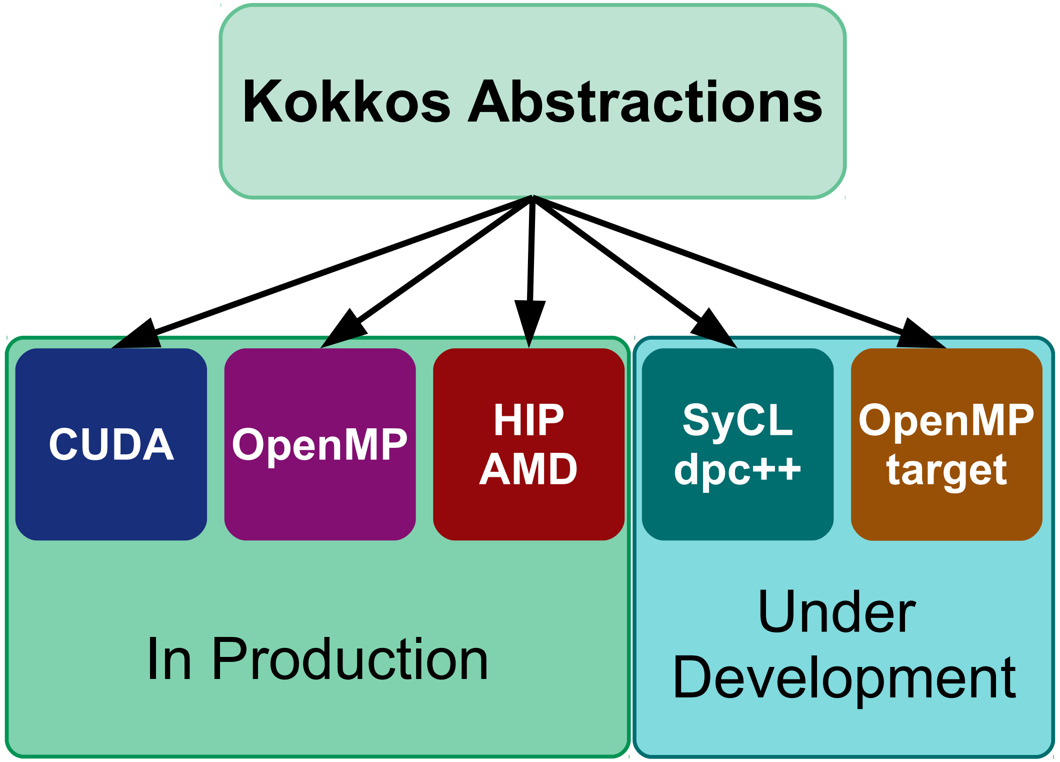
- parallel_for, reduction, scans

Data interaction *via* **`Kokkos::Views<...>`**

- explicit memory movement
- memory space selected via policy
- impact on C++ code

Execution is bound to an Execution Space

- device backend must be selected at compile time



SYCL / dpc++

SYCL 1.2.1 plus Intel extensions (some of which are from SYCL 2.X)

Single source C++ (understands C++17)

Explicit memory transfers not needed

- builds a DAG of kernel/data dependencies, transfers data when needed on device

Executes on all platforms (or at least will "soon")

- including CPU, GPU, FPGA
- selectable at runtime (mostly)
- complex ecosystem

Intel wants to push into llvm main branch

- OneAPI to become an open specification

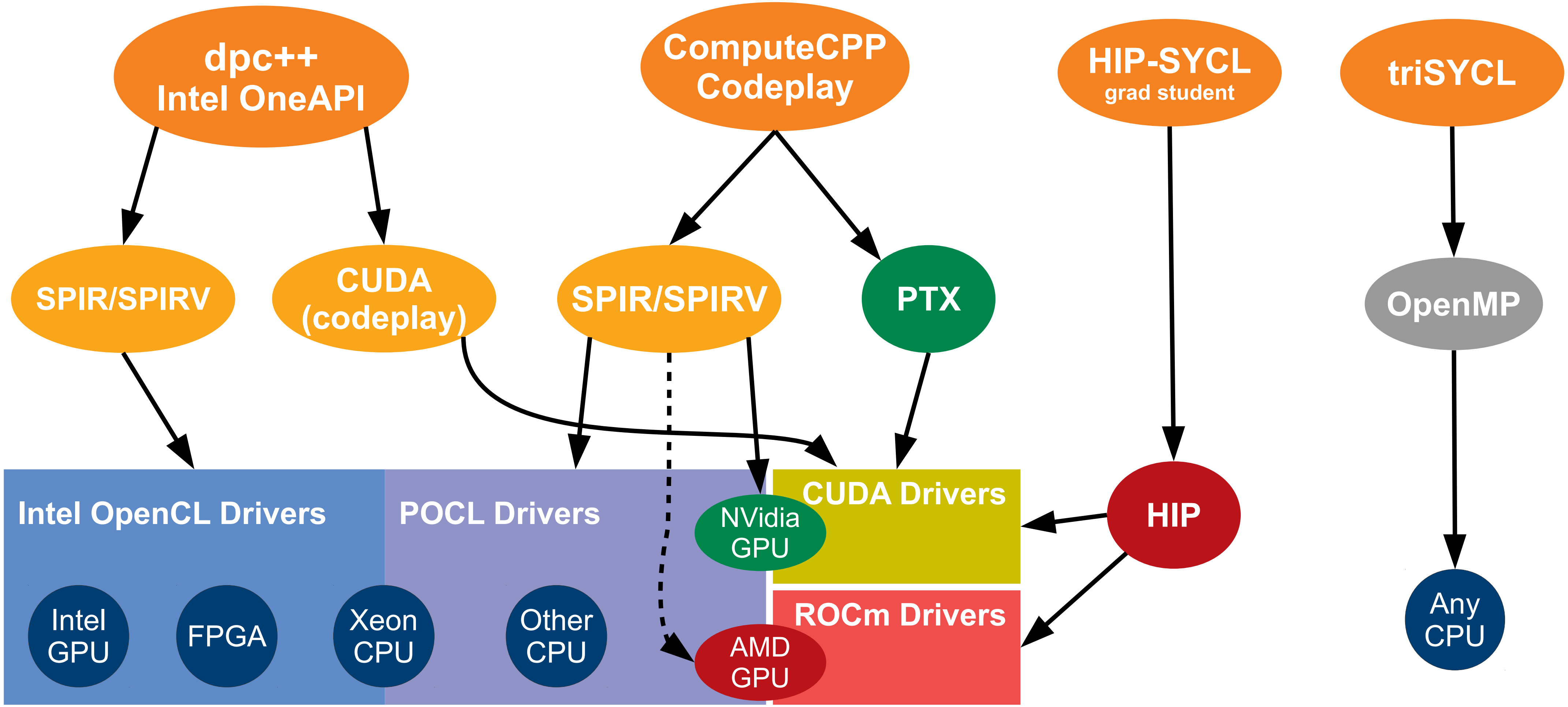
Long term OpenCL IR layer in question: Intel is replacing it with "LevelZero"

- OpenCL 1.2 standard is too limiting

Concurrent kernels don't work, for ANY backend (and won't for the foreseeable future)

- except CPU, where you can get 2 threads/core concurrently

SYCL Ecosystem



OpenCL 3.0

Newly released provisional specification by Khronos group

- Makes many of the "improvements" of the 2.X standard over 1.2 optional, enabling much broader range of devices to be 3.0 compliant, allowing use of extra features if they're available/applicable. "Less is More"
- Allows more gradual adoption of extra features by vendors. Move away from "all or nothing" approach of 2.X
- OpenCL C++ abandoned in favor of clang based C++ for OpenCL (allows most C++17)

Strong-ish statements of support from various industry groups:

> NVIDIA will ship a conformant OpenCL 3.0 when the specification is finalized
>
> *Compute product manager at NVIDIA*

> Codeplay is excited to enable hardware vendors to support OpenCL 3.0 and to take advantage of the flexibility provided
>
> *Founder and CEO of Codeplay Software*

> Intel strongly supports cross-architecture standards being driven across the compute ecosystem such as in OpenCL 3.0 and SYCL
>
> *Vice President, Intel Architecture, Graphics and Software*

Alpaka

Single source C++ kernels, standard limited by backend

Platform decided at compile time

CUDA like, multidimensional set of threads

- grid / block / thread / element

Maps abstraction model to desired acceleration back ends

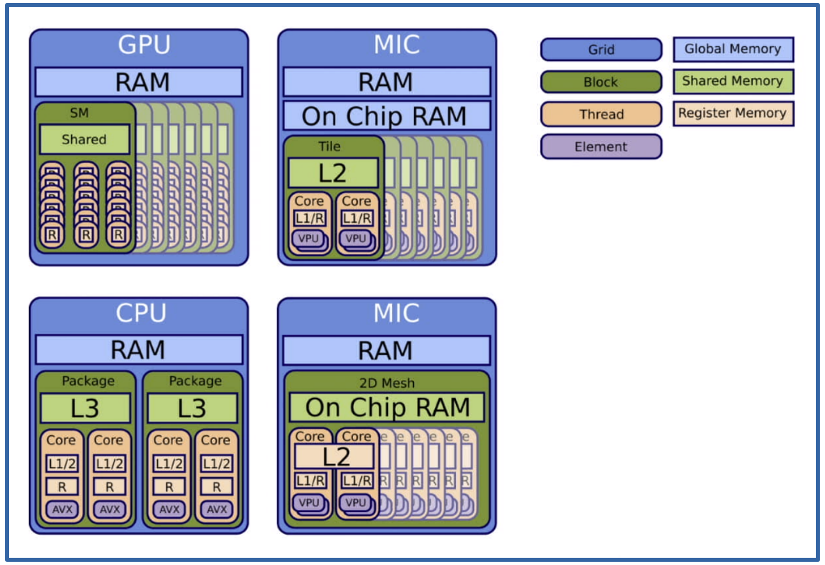
Data agnostic memory model:

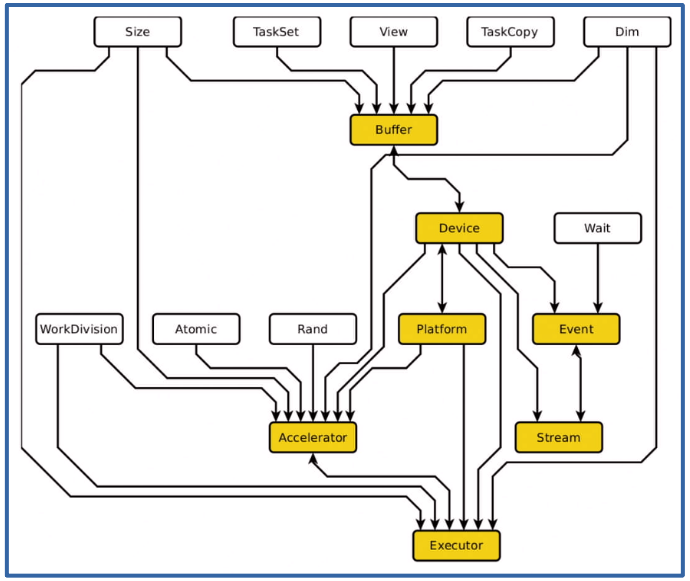
- allocates memory for you, but needs directives for hardware mapping
- same API for allocating on host and device

Uses templates for a "Zero Overhead Abstraction Layer"

Straightforward porting of CUDA kernels using cupla

- only include and kernel calls need to be changed





OpenMP / OpenACC

Two similar mechanisms for annotating code to direct the compiler to offload bits of code to other devices

- uses `#pragmas`

OpenMP was really developed for MP on HPC

- very large and complex standard
- recently extended to target GPUs
- very prescriptive: need to tell compiler exactly how to unroll loops
  - have to modify pragmas when move to different GPU architecture

OpenACC developed explicitly for accelerators

- lets compiler make intelligent decision on how to decompose problems
- is a standard that describes what compilers ***should*** do, not *must*
  - different compilers interpret ***should*** very differently
- very strong support in Fortran community

Best supported on HPC

Metrics: 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

- current and future support
- new architectures

Feature availability

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

Ease of Debugging

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 Status

Currently porting Patatrack Pixel Tracking, FastCaloSim and WireCell Toolkit to Kokkos v3.1

- build system integration
    - integrating with existing make / cmake build systems can be challenging
- kernel and data structure identification
    - replacing CUDA malloc/memcpy with **`Kokkos::Views`** is intrusive
- preliminary porting of kernels

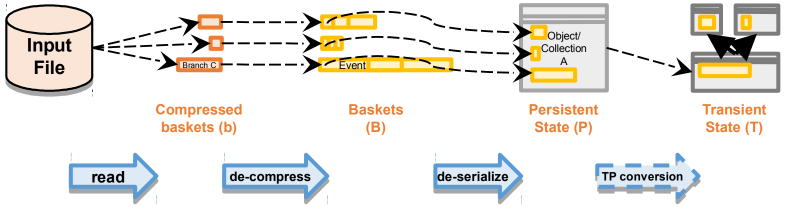
General observations

- some Execution spaces have features that are not supported by others. May require use of **`#ifdef`** if these features are used

CCE-IOS

Goals

- Efficient serialiation/de-serialization of data representations under parallel I/O models.
- Development of persistable data representations that can be tuned for access on HPC storage systems and optimized for HEP I/O patterns.
- Optimization of reads of partial, partitioned or sub-event data blocks from storage which are matched to specific algorithm consumption requirement.
- Optimization of runtime memory mapping of data to exploit batched, vectorized, and data parallel operations and transforms on columnar data.



Status

- Darshan for ROOT I/O in HEP workflows on HPC.
  - ROOT I/O is central to all HEP experiments. Measurements of its performance on HPC using tools like Darshan, could give valuable insights for possible improvements.
- Investigate HDF5 as intermediate event storage for HPC processing
  - In some workflows, such as the ATLAS EventService, temporary data is written to ROOT files. Moving this data to a parallel file format such as HDF5 could be benefitial.

fin

CMS Patatrack Pixel Tracking

Goal is demonstrate that part of the CMS HLT Pixel local reconstruction can be efficiently offloaded to a GPU

- 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

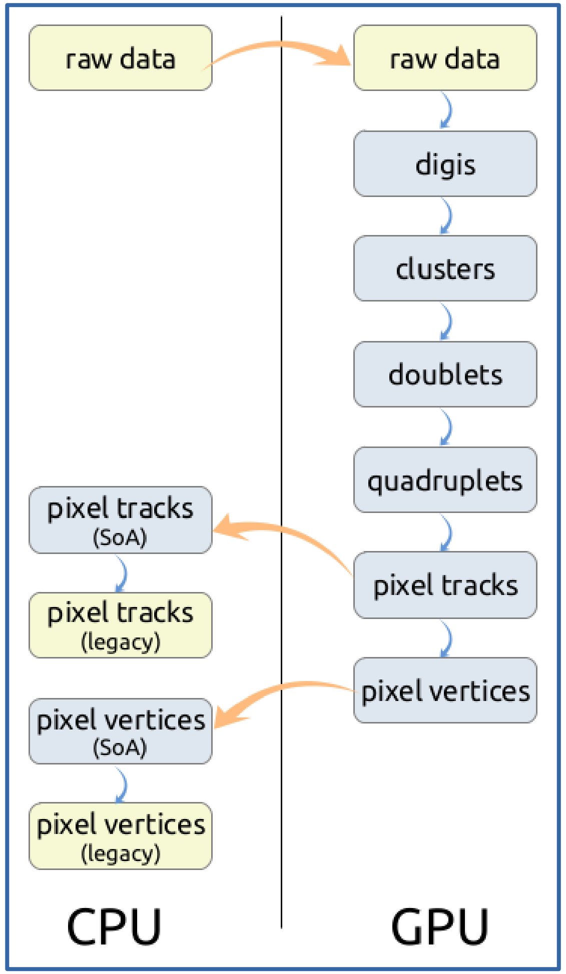
- decode the pixel raw data
- cluster the pixel hits (SoA)
- form hit doublets
- form hit quadruplets (or ntuplets) with a Cellular automaton algorithm – clean up duplicates

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



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 Camber (TPC) simulation, signal processing, reconstruction and visualization.

- Written in 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

ATLAS FastCaloSim

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

Full detailed simulation uses Geant4, which is very slow

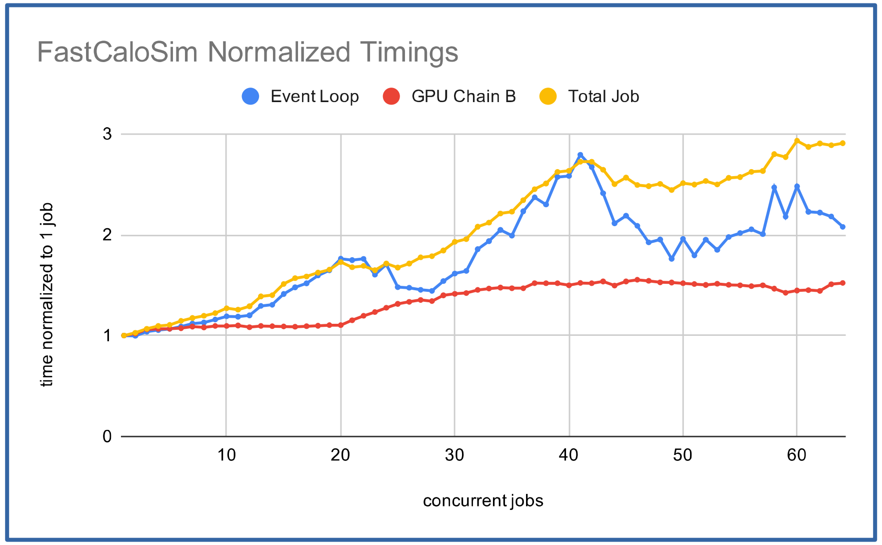
Fast calorimeter simulation uses parametrization of the calorimeter

- less accurate, but much faster than Geant4

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

BNL group has already created a CUDA port

- modify data structures (eg Geometry) to offload to GPU
- multi-stage CUDA kernels to generate histograms
- current efficiency hampered by small work sizes


FastCaloSim Normalized Timings
● Event Loop
● GPU Chain B
● Total Job
time normalized to 1 job
concurrent jobs