Portable Parallelization Strategies

Charles Leggett

CCE Kickoff Meeting, ANL
March 9 2020

Heterogenerous Architectures

- Existing and future Heterogeneous HPCs

		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			
	Fujitsu					Fugaku

Amazon EC2 P3
Google Cloud TPU
Microsoft Azure

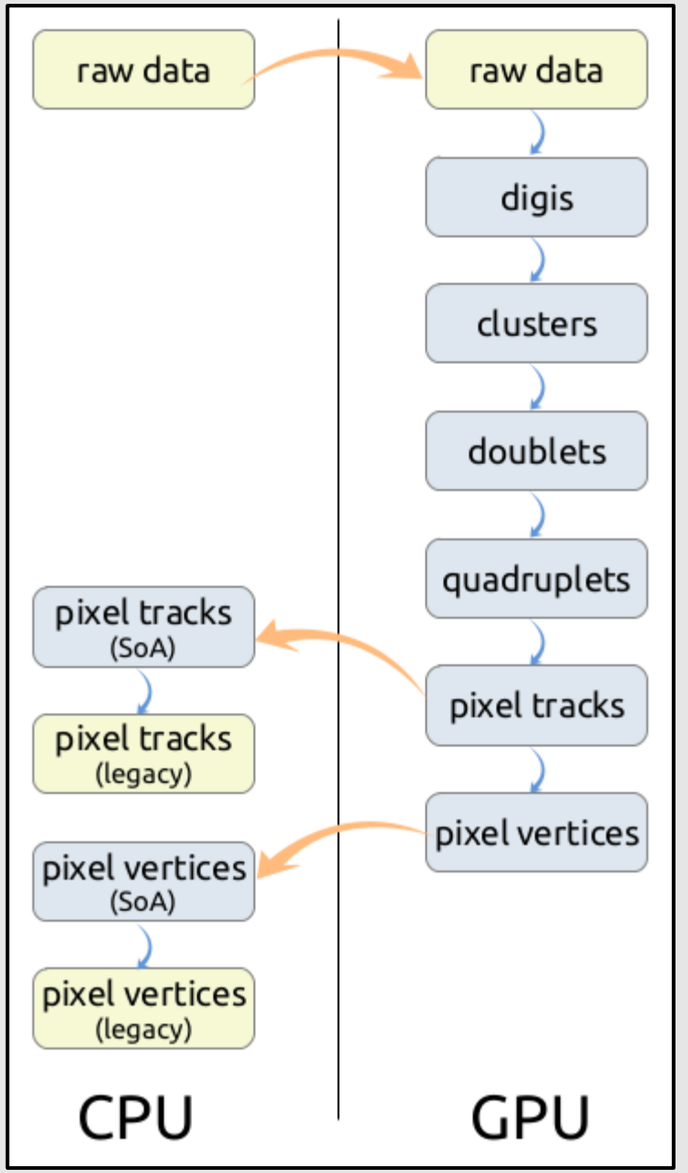
- Other accelerator flavors too: TPU, IPU, CSA ...
- We will have more CPU+GPU+FPGA/other in one machine in the future

CCE-PPS Mandate

- Investigate a range of portability solutions such as
  - Kokkos / Raja
  - SyCL
  - Alpaka
  - OpenMP / OpenACC
- Port a small number of HEP testbeds to each language
  - Patatrack (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*
  - see discussion tomorrow in parallel session
- Make recommendations to the experiments
  - must address needs of both LHC style workflows with many modules and many developers, and smaller/simpler workflows

CMS Patatrack

- 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



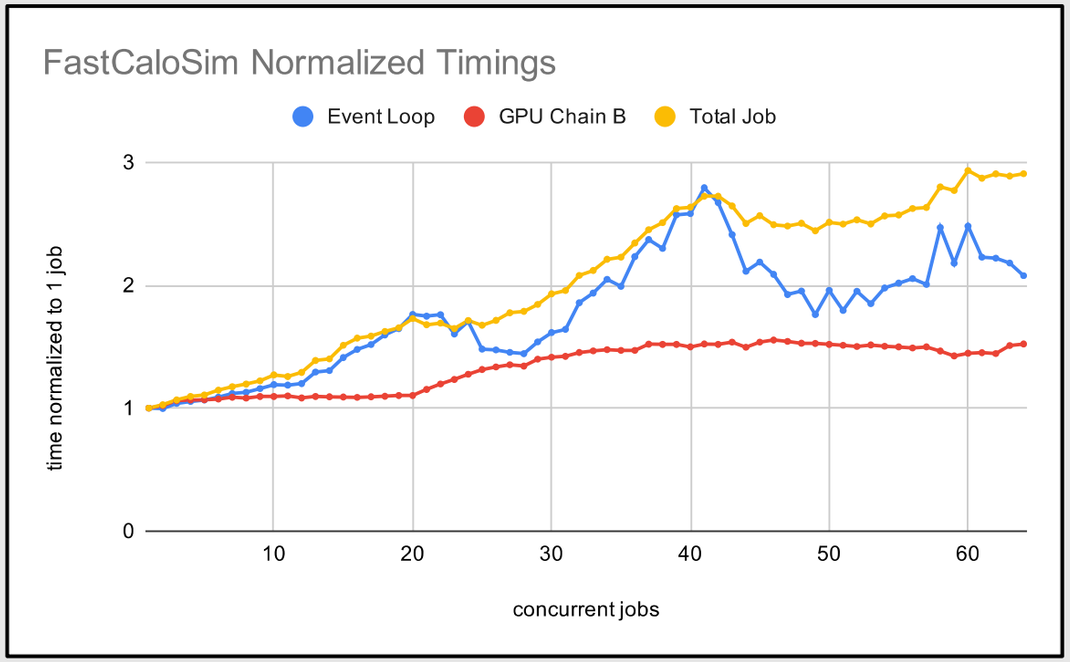
Wire-Cell Toolkit

- 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
  - Currently single-threaded, but can be multi-threaded
  - Can be interfaced to LArSoft
- 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



Software Support

- Compilers / Software Support Matrix

	OpenMP Offload	Kokkos / Raja	dpc++ / SyCL	HIP	CUDA	Alpaka
NVidia GPU						
AMD GPU						
Intel GPU						
CPU						
Fortran						
FPGA						

Supported

Under Development

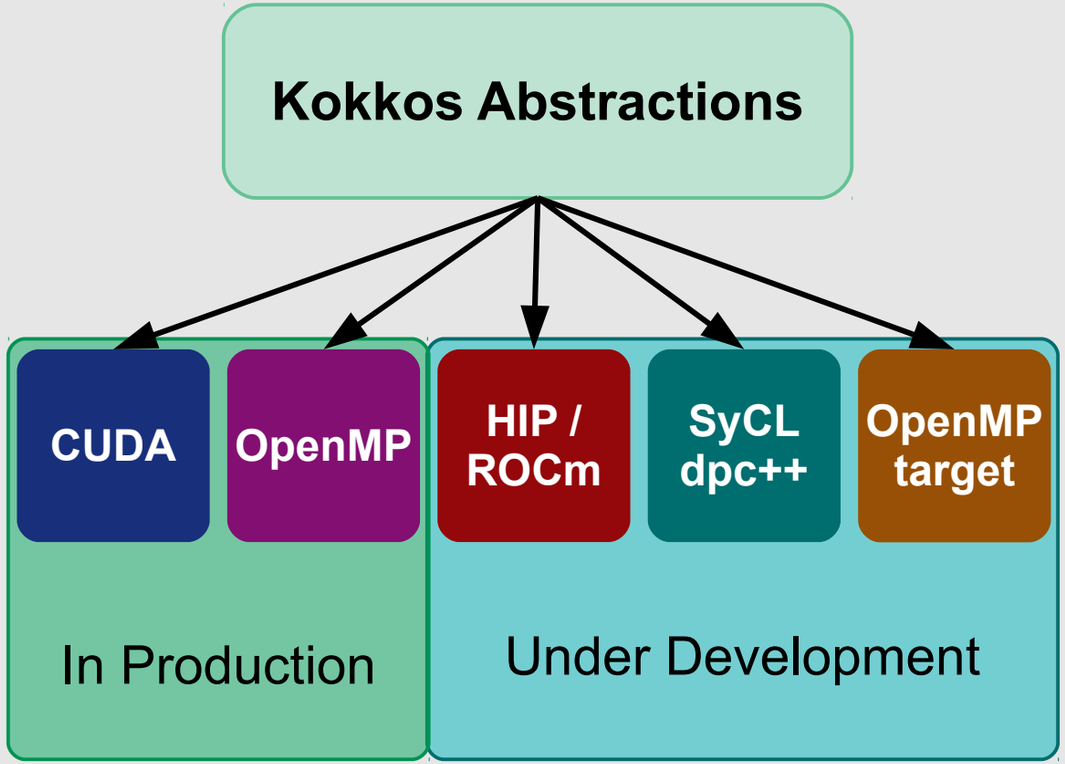
3rd Party

Not Supported

- We are seeing ongoing and rapid evolution

Kokkos

- Kokkos provides portability via various backends: OpenMP, CUDA, (tbb), *etc*
  - backend must be selected at compile time
- Single source
- 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
  - select back end via a policy

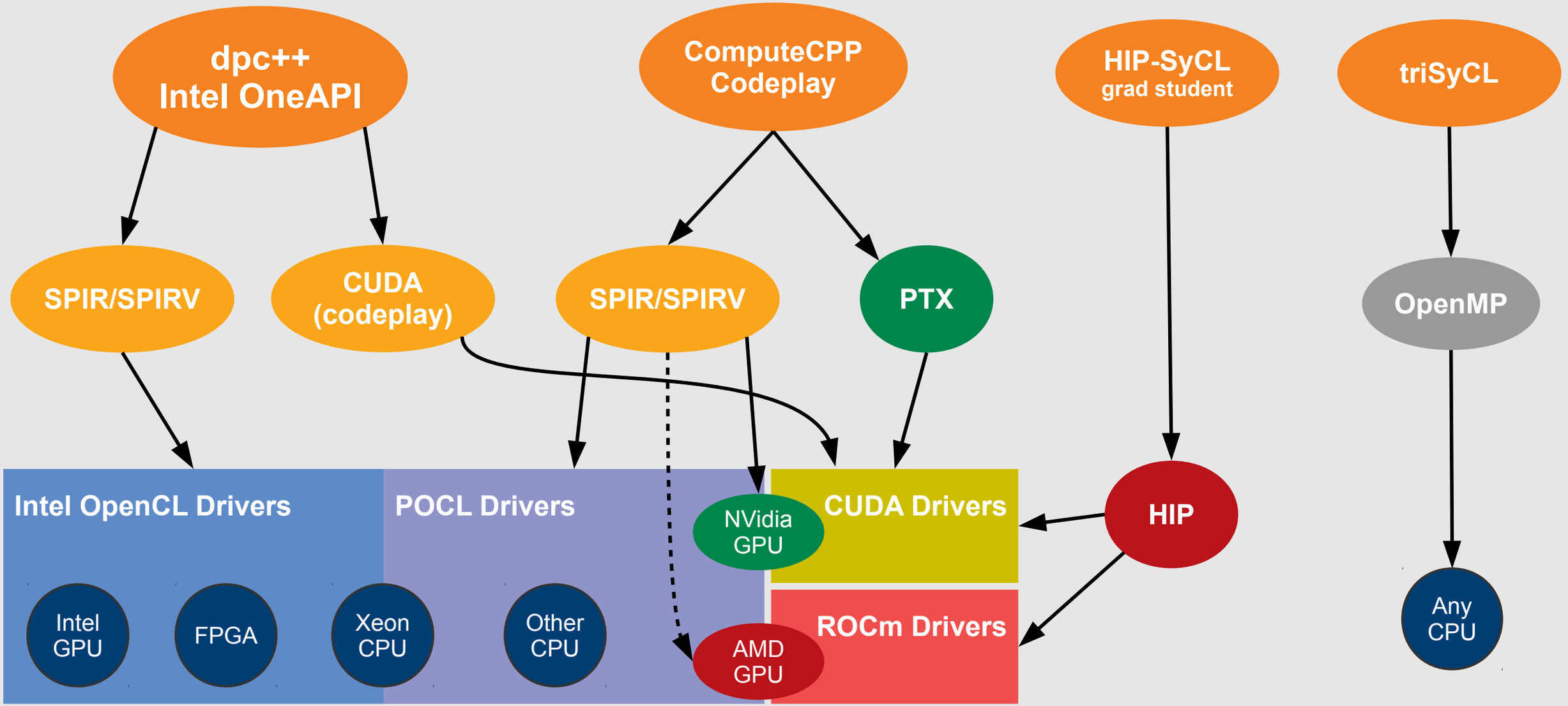


SyCL / Data Parallel C++ (dpc++)

- 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 as needed
- Executes on all platforms
  - or at least will "soon"
  - including CPU, FPGA
  - selectable at runtime (mostly)
  - complex ecosystem
- Intel wants to push into llvm main branch
  - become an open standard
- 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
  - except CPU, where you can get 2 threads/core concurrently

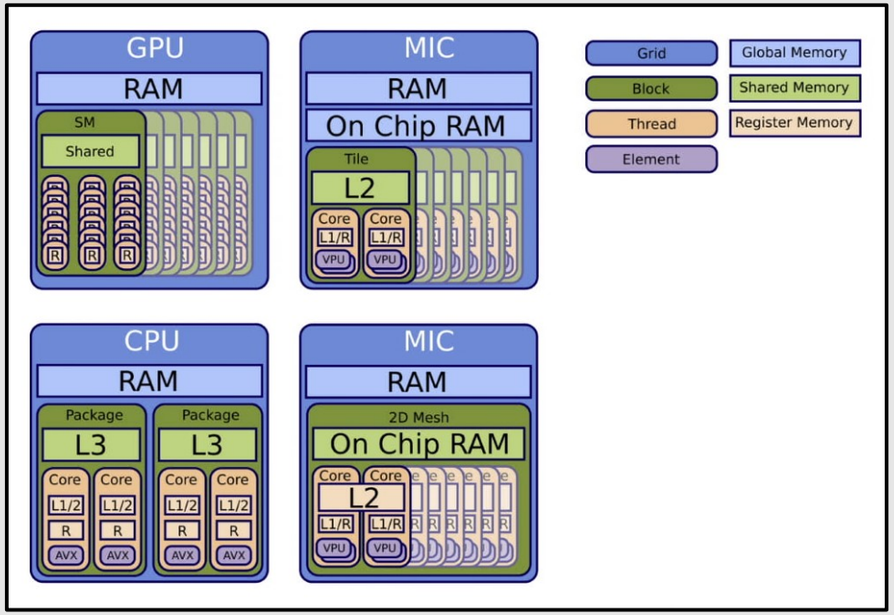
SyCL Ecosystem

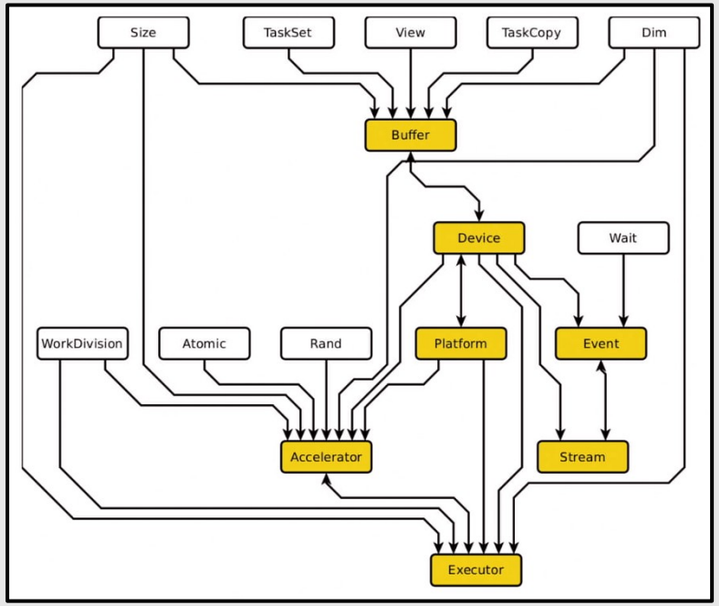
- complicated ecosystem



Alpaka

- Single source kernels
  - C++14
- 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"
- Trivial 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 and extent of code modification
- Impact on existing EDM
- Impact on other existing code
  - 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
- Feature availability
  - reductions, kernel chaining, callbacks, *etc*
  - concurrent kernel execution
- Address needs of all types of workflows
- Long-term sustainability and code stability
- Compilation time
- Performance: CPU and GPU
- Aesthetics

Longer discussion tomorrow

Timeline

- Phase 1: Preparation
  - Q1: Deliverable: matrix of benchmarks of the unaltered use cases
- Phase 2: First Implementation
  - Q4: Deliverable: choose one use case, implement in all selected parallelization technologies and benchmark them.
  - Determine the best 3 technologies according to the metrics.
  - Q5-7: Deliverable: implement all use cases in one of the three chosen technologies.
- Phase 3: Consolidate benchmarking results
  - Q8: Deliverable: write-up summarizing the benchmarking results of the use cases and recommending parallelization strategies for the HEP community.
- Phase 4: Fully-functional prototypes
  - provide recommendations on portability strategies to experiments
  - Q12: Deliverable: fully-functional prototypes made available to the experiments.

Next Steps

- Start investigation of portability solutions by porting testbeds to Kokkos
  - do all three codebases
  - start with Patatrack since it already has a Kokkos implementation for one kernel in the standalone application
- Coordinate closely with I/O CCE group to design data structures that address requirements of accelerators as well as parallel I/O
- Select a subset of Patatrack modules with a few chained kernels
  - implement Kokkos Views to manage data
  - rewrite kernels in Kokkos

fin