High Energy Physics
Center for Computational Excellence

P. Calafiura
for the HEP-CCE project
(special thanks to C. Leggett and P. van Gemmeren)

SwiftHep/ExcaliburHep Workshop
Jan 14, 2021

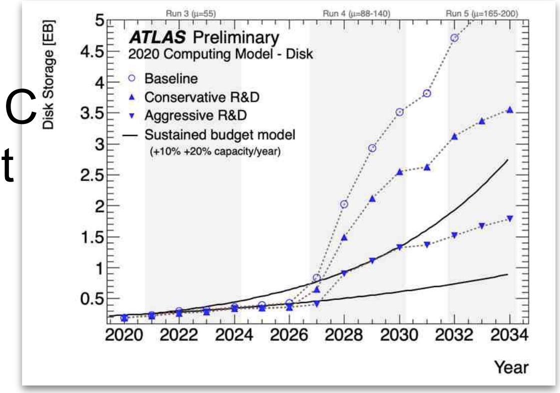
Why HEP-CCE?

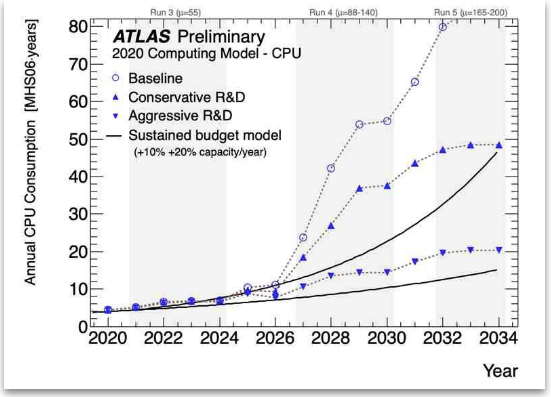
HEP computing resource challenges

Notably 10x more data, 10x more complexity @ HL-LHC

→ need **PetaFlops *sustained*** per experiment

Project CPU **and** Storage resource **shortage**





US long-term investment in HPCs

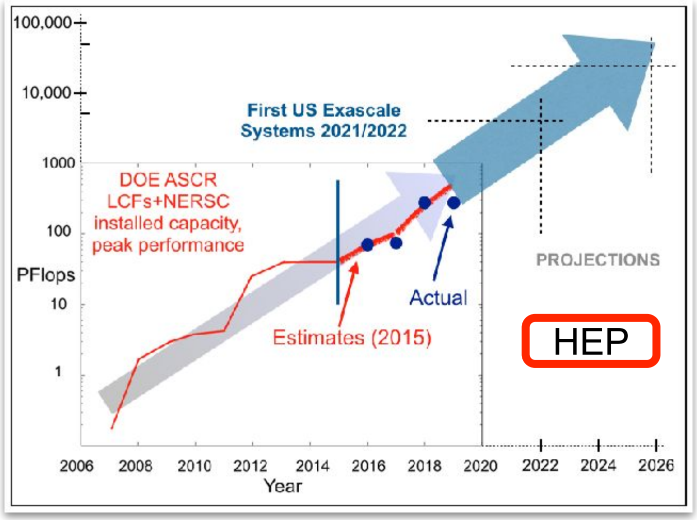
Platform of choice for simulations, and, more recently, data processing (light sources, LIGO, cosmology…)

Expect O(10) **ExaFlops *peak*** performance by 2026

→ Challenging to run at 10% of peak

DOE would very much like HEP to partake

→ Most of the cycles provided by accelerators



Why HEP-CCE? “Extreme Heterogeneity”

“A New Golden Age for Computer Architectures” (D. Patterson)

		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 Graviton2
- Google Cloud TPU
- Microsoft Azure
- Intel DevCloud

Opportunity to bring HEP into the “HPC family”

- Develop practical solutions to port hundreds of kernels, complex data models
- Collaborate with HPC community (including networking) on data intensive use cases

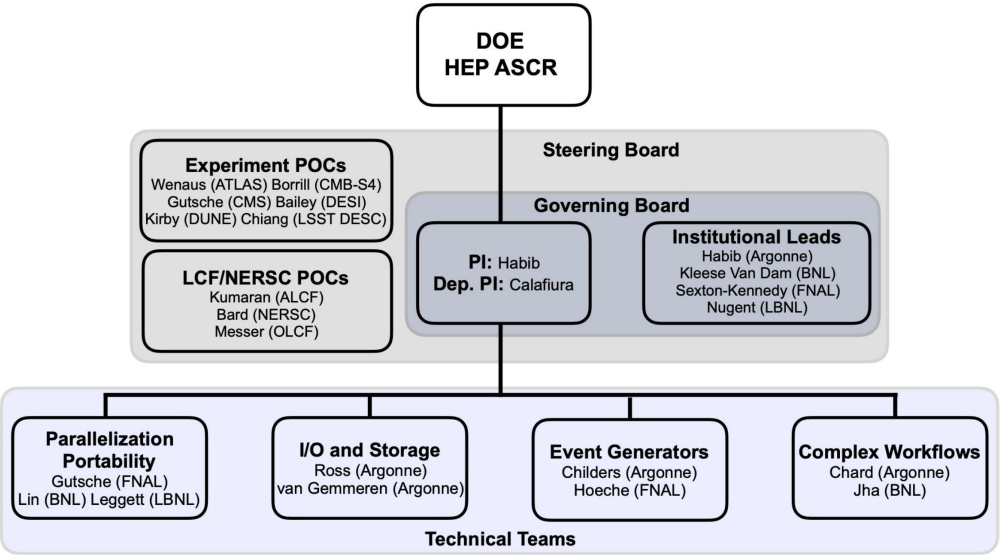
What is HEP-CCE?

Three-year (2020-2023) **pilot** project

four US labs, six experiments, ~12 FTE, ~30 collaborators

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

1. Portable parallelization strategies
   - exploit massive concurrency
   - portability requirements
2. Fine-grained I/O and related storage issues
   - new data models (zero-copying, SOA,...)
   - event batching (XPU offloading)
3. Optimizing event generators
4. Running complex workflows on HPCs
   - main use case: cosmology surveys



DOE
HEP ASCR

Steering Board

Experiment POCs
Wenaus (ATLAS) Borrill (CMB-S4)
Gutsche (CMS) Bailey (DESI)
Kirby (DUNE) Chiang (LSST DESC)

LCF/NERSC POCs
Kumaran (ALCF)
Bard (NERSC)
Messer (OLCF)

Governing Board

PI: Habib
Dep. PI: Calafiura

Institutional Leads
Habib (Argonne)
Kleese Van Dam (BNL)
Sexton-Kennedy (FNAL)
Nugent (LBNL)

Parallelization Portability
Gutsche (FNAL)
Lin (BNL) Leggett (LBNL)

I/O and Storage
Ross (Argonne)
van Gemmeren (Argonne)

Event Generators
Childers (Argonne)
Hoeche (FNAL)

Complex Workflows
Chard (Argonne)
Jha (BNL)

Technical Teams

Open collaboration

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

CCE/PPS: Goals and Year 1 priorities

Investigate a range of software portability solutions:

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

- products are rapidly evolving
- we have some hope of seeing the emergence of industry standards at the C++ language level

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*

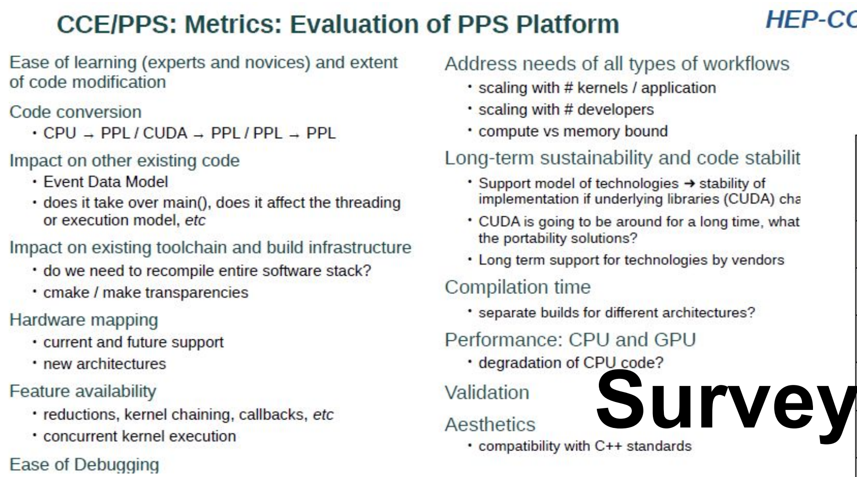
Make recommendations to the experiments

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

CCE/PPS: Year 1 achievements

HEP-CCE

See Nov 2020 HEP-CCE AHM
for details



CCE/PPS: Metrics: Evaluation of PPS Platform

HEP-CCE

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

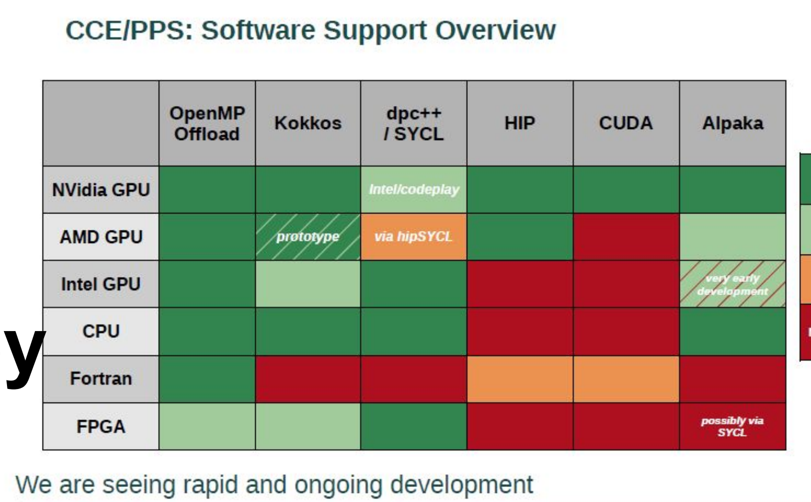
Compilation time
- separate builds for different architectures?

Performance: CPU and GPU
- degradation of CPU code?

Validation

Aesthetics
- compatibility with C++ standards

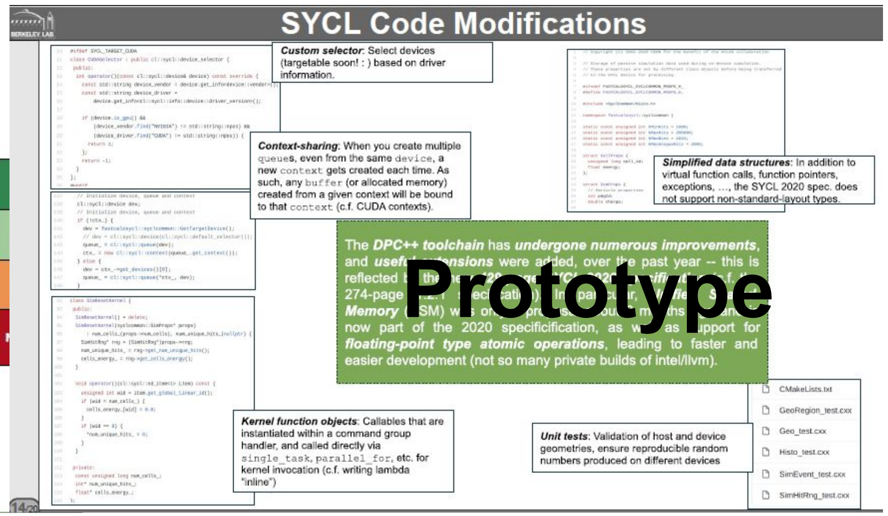
Survey



CCE/PPS: 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*

We are seeing rapid and ongoing development



SYCL Code Modifications

```
#ifdef SYCL_TARGET_CUDA
class CUDASelector : public cl::sycl::device_selector {
 public:
  int operator()(const cl::sycl::device& device) const override {
    const std::string device_vendor = device.get_info<device::vendor>();
    const std::string device_driver =
        device.get_info<cl::sycl::info::device::driver_version>();

    if (device.is_gpu() &&
        (device_vendor.find("NVIDIA") != std::string::npos) &&
        (device_driver.find("CUDA") != std::string::npos)) {
      return 1;
    };
    return -1;
  }
};
#endif
```

```
  // Initialize device, queue and context
  cl::sycl::device dev;
  // Initialize device, queue and context
  if (!ctx_) {
    dev = fastcalosycl::syclcommon::GetTargetDevice();
    // dev = cl::sycl::device(cl::sycl::default_selector());
    queue_ = cl::sycl::queue(dev);
    ctx_ = new cl::sycl::context(queue_.get_context());
  } else {
    dev = ctx_->get_devices()[0];
    queue_ = cl::sycl::queue(*ctx_, dev);
  }
```

Custom selector: Select devices (targetable soon! :) based on driver information.

Context-sharing: When you create multiple `queue`s, even from the same `device`, a new `context` gets created each time. As such, any `buffer` (or allocated memory) created from a given context will be bound to that `context` (c.f. CUDA contexts).

Simplified data structures: In addition to virtual function calls, function pointers, exceptions, …, the SYCL 2020 spec. does not support non-standard-layout types.

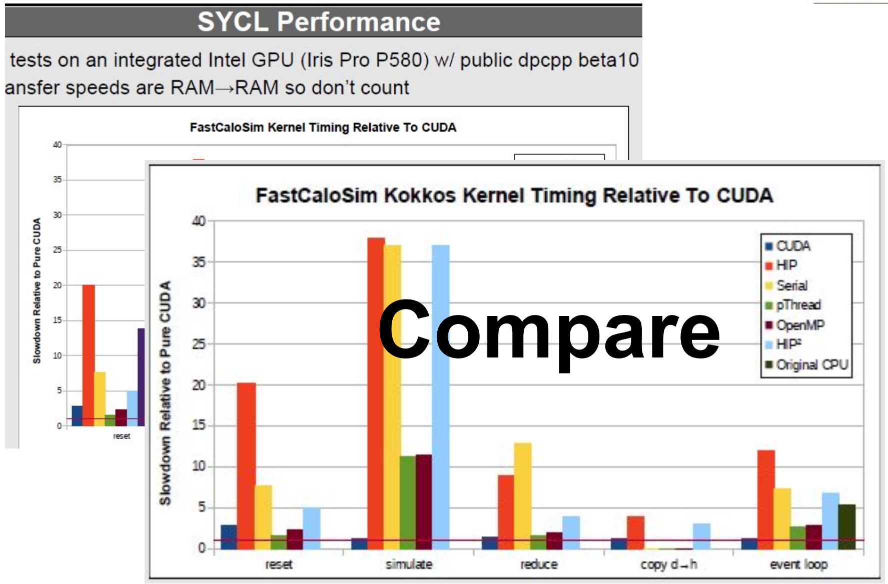
The ***DPC++ toolchain*** has ***undergone numerous improvements***, and ***useful extensions*** were added, over the past year -- this is reflected [illegible] 274-page [illegible] ***Memory*** (USM) [illegible] now part of the 2020 specificification, as w[illegible] as [illegible]upport for ***floating-point type atomic operations***, leading to faster and easier development (not so many private builds of intel/llvm).

Prototype

Kernel function objects: Callables that are instantiated within a command group handler, and called directly via `single_task`, `parallel_for`, etc. for kernel invocation (c.f. writing lambda "inline")

Unit tests: Validation of host and device geometries, ensure reproducible random numbers produced on different devices

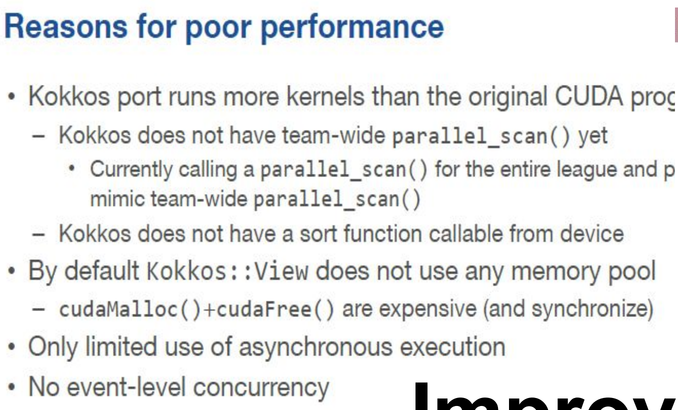
- CMakeLists.txt
- GeoRegion_test.cxx
- Geo_test.cxx
- Histo_test.cxx
- SimEvent_test.cxx
- SimHitRng_test.cxx



SYCL Performance

tests on an integrated Intel GPU (Iris Pro P580) w/ public dpcpp beta10
ansfer speeds are RAM→RAM so don't count

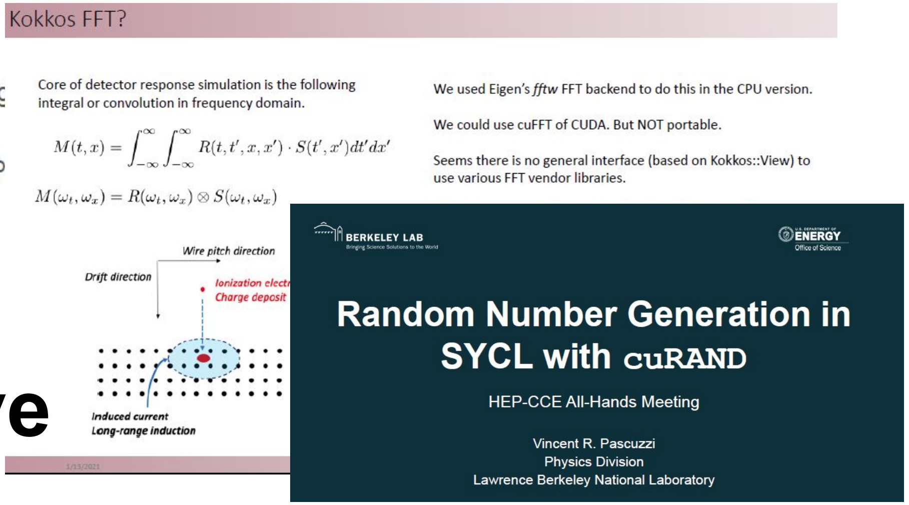
Compare



Reasons for poor performance

- Kokkos port runs more kernels than the original CUDA prog
  - Kokkos does not have team-wide `parallel_scan()` yet
    - Currently calling a `parallel_scan()` for the entire league and p mimic team-wide `parallel_scan()`
  - Kokkos does not have a sort function callable from device
- By default `Kokkos::View` does not use any memory pool
  - `cudaMalloc()+cudaFree()` are expensive (and synchronize)
- Only limited use of asynchronous execution
- No event-level concurrency

Improve



Kokkos FFT?

Core of detector response simulation is the following integral or convolution in frequency domain.

$$M(t,x) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} R(t,t',x,x') \cdot S(t',x')dt'dx'$$

$$M(\omega_t,\omega_x) = R(\omega_t,\omega_x) \otimes S(\omega_t,\omega_x)$$

We used Eigen's *fftw* FFT backend to do this in the CPU version.

We could use cuFFT of CUDA. But NOT portable.

Seems there is no general interface (based on Kokkos::View) to use various FFT vendor libraries.

Random Number Generation in SYCL with cuRAND

HEP-CCE All-Hands Meeting

Vincent R. Pascuzzi
Physics Division
Lawrence Berkeley National Laboratory

BERKELEY LAB
Bringing Science Solutions to the World
U.S. DEPARTMENT OF ENERGY
Office of Science

CCE/IOS: Goals

IOS will focus and concentrate effort on:

- **Parallel serialization/de-serialization** of HEP data models
  - both single node and multi-node access patterns
- **Persistable data representations** tuned for HPC storage systems.
  - Connection to PPS exploration of portable parallelization libraries
  - can benefit from Write-Once/Read-Many HEP access models
- **Accessing partial, partitioned or sub-event data blocks**
  - matched to specific algorithm consumption requirement
- **Runtime memory mapping of data**
  - exploit batched, vectorized, and data parallel operations and transforms on columnar data.
  - taking into account CPU-XPU communication

CCE/IOS: Year 1 Activities

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 beneficial.

Testing framework for understanding scalability and performance of HEP output methods

- An ability to simulate HEP output of specific data products (e.g., RECO, AOD, miniAOD) in different scenarios prepares us for deeper analysis of intermediate data storage options.

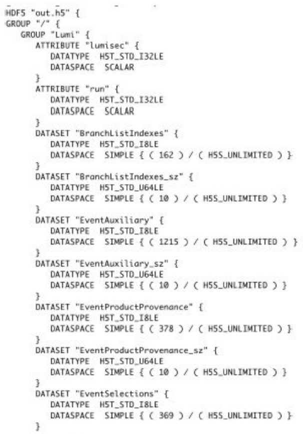
CCE/IOS: Year 1 Achievements

in collaboration with SCIDAC-4 HEPonHPC and HEPnOS projects

Mapping HEP data to HDF5

There is no direct mapping to HDF5 way of organizing

- map a subrun/lumi to a group and add lumi and ru attributes
- map each data product to multiple data sets; first (product itself, and the second dataset for event ID offset or size.
  - Different options explored
- read and write performance will depend on access patterns

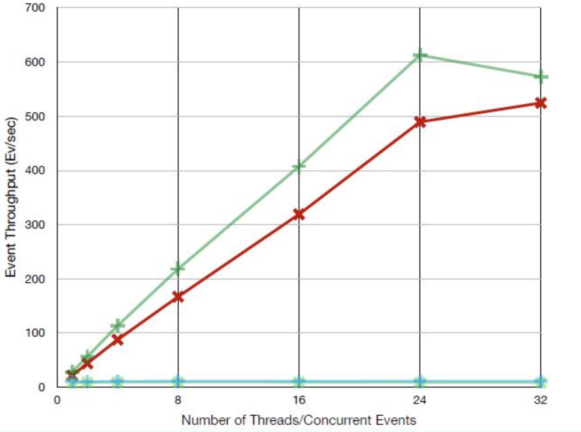


```
HDF5 "out.h5" {
GROUP "/" {
   GROUP "Lumi" {
      ATTRIBUTE "lumisec" {
         DATATYPE  H5T_STD_I32LE
         DATASPACE  SCALAR
      }
      ATTRIBUTE "run" {
         DATATYPE  H5T_STD_I32LE
         DATASPACE  SCALAR
      }
      DATASET "BranchListIndexes" {
         DATATYPE  H5T_STD_I8LE
         DATASPACE  SIMPLE { ( 162 ) / ( H5S_UNLIMITED ) }
      }
      DATASET "BranchListIndexes_sz" {
         DATATYPE  H5T_STD_U64LE
         DATASPACE  SIMPLE { ( 10 ) / ( H5S_UNLIMITED ) }
      }
      DATASET "EventAuxiliary" {
         DATATYPE  H5T_STD_I8LE
         DATASPACE  SIMPLE { ( 1215 ) / ( H5S_UNLIMITED ) }
      }
      DATASET "EventAuxiliary_sz" {
         DATATYPE  H5T_STD_U64LE
         DATASPACE  SIMPLE { ( 10 ) / ( H5S_UNLIMITED ) }
      }
      DATASET "EventProductProvenance" {
         DATATYPE  H5T_STD_I8LE
         DATASPACE  SIMPLE { ( 378 ) / ( H5S_UNLIMITED ) }
      }
      DATASET "EventProductProvenance_sz" {
         DATATYPE  H5T_STD_U64LE
         DATASPACE  SIMPLE { ( 10 ) / ( H5S_UNLIMITED ) }
      }
      DATASET "EventSelections" {
         DATATYPE  H5T_STD_I8LE
         DATASPACE  SIMPLE { ( 369 ) / ( H5S_UNLIMITED ) }
      }
```

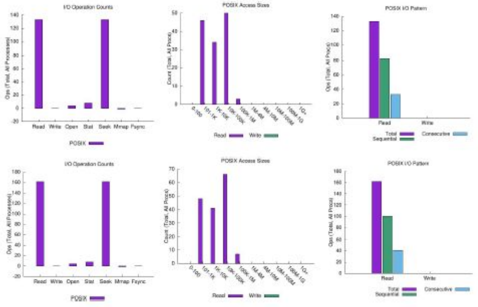
Testing Framework Purpose

- Mimic the characteristics of a HEP data processing framework
  - Similar multi-threaded behavior
  - Similar I/O behavior
  - Should reasonably behave like CMS, ATLAS anc
- Easily try different I/O implementations
  - Choose what to use via command line argument
- Experiment agnostic
  - With ability to read actual experiment ROOT files
    - ROOT will dynamically load serialization/deserializa
- Make it easier to perform performance me
  - I/O performance
  - threaded scaling performance

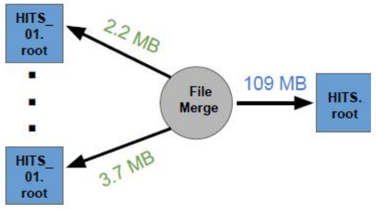
- Use ROOT C++ object serialization concurrently
  - Allow each serialized Event to be compressed concurrently



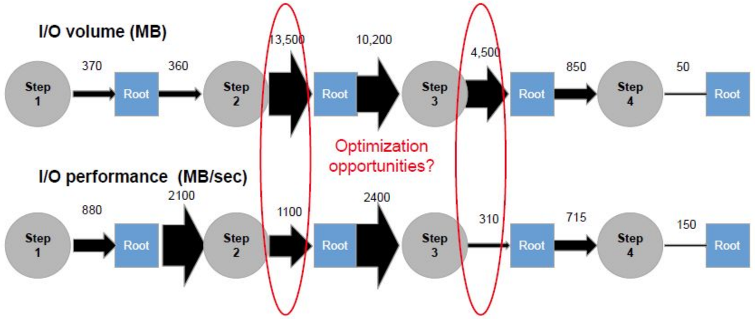
Darshan analysis of ATLAS I/O



Equal numbers of reads/seeks, reads mostly sized at O(10s of KB), I/O noticeably less sequential than write phase

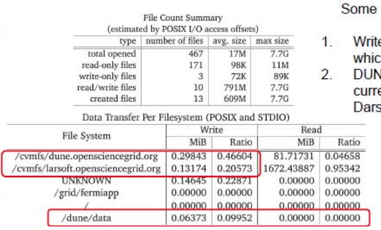


CMS workflow I/O: Step 4 final output



DUNE workflow I/O analysis

- ❖ Similar in nature to CMS workflow pattern described prev detector simulation + reconstruction)
  - ➢ Similar dependence on technologies like cvmfs, XRootD, etc.
- ❖ Preliminary Darshan results for standard reconstruction of ProtoDUNE-SP raw data on DUNE interactive machines
  - ➢ Raw inputs read from NFS for these tests (rather than traditional XRootD reading of data)
  - ➢ Run 5 events (input file contains O(100) events)



File Count Summary
(estimated by POSIX I/O access offsets)

type	number of files	avg. size	max size
total opened	467	17M	7.7G
read-only files	171	98K	11M
write-only files	3	72K	89K
read/write files	10	791M	7.7G
created files	13	609M	7.7G

Data Transfer Per Filesystem (POSIX and STDIO)

File System	Write		Read	
	MiB	Ratio	MiB	Ratio
/cvmfs/dune.opensciencegrid.org	0.29843	0.46604	81.71731	0.04658
/cvmfs/larsoft.opensciencegrid.org	0.13174	0.20573	1672.43887	0.95342
UNKNOWN	0.14645	0.22871	0.00000	0.00000
/grid/fermiapp	0.00000	0.00000	0.00000	0.00000
/	0.00000	0.00000	0.00000	0.00000
/dune/data	0.06373	0.09952	0.00000	0.00000

Some open questions:

1. Write activity to cvmfs, which is read-only?
2. DUNE output files not currently captured by Darshan?

CCE/EG: Event Generators

Why improve event generators?

- Event generation might consume more resources at HL-LHC that we currently extrapolate
- Generators code generally not well written ➜ scaling issues even on HTC resources, never mind HPC

Building on previous improvements

- Parallelized Pythia particle-level event generation
- Improved performance of Sherpa, particularly I/O (HDF5)
- Novel integrator using Neural Networks and Normalizing Flows

CCE plans

- Complete rewrite of matrix-element generator for CPUs & GPUs
- Paradigm shift from “best of scaling” to “best for computation”
- Exploration of different frameworks (low-level, Kokkos, ...)
- Alternatively try to revive existing implementations (HELAS/MG)
- Coordinate with HSF and worldwide effort

CCE/CW: Complex Workflows

Why complex workflows?

- Workflows enable representation and execution of analyses composed of heterogeneous components
  - Single node codes, multi-node MPI applications, scripts, binaries, glue code, etc.
- HPC systems and environments are not designed for such workloads
  - Millions of tasks, diverse and varying requirements (cores, duration, CPU/GPU), co-scheduling and dynamic sizing

▪ CCE plans

- Support development and use of workflows within target domains
- Apply a modular, interoperable, and inclusive approach
  - Leverage/combine components from existing workflow systems
- Democratize access to advanced workflow capabilities and reduce overheads on small collaborations
  - Enable use of accelerators, heterogeneous hardware, without knowledge of low-level programming
  - R&D related to portability, scalability, performance, and reproducibility

Year 2 Priorities

PPS

Added use cases:

- CMS track follower/fitter: math-intensive kernel
- ACTS: experiment-independent end-to-end tracking package

New activity: Event Batching

- in collaboration with IOS

IOS

New activity: Data Model and storage for CPU/XPU communication

  - Cross-educate on ASCR and HEP activities
    - awkwardarray, dataframes, ATLAS/CMS columnar AODs

EG

- Madgraph port to CUDA/Kokkos in collaboration with Madgraph team.

Outlook

PPS and IOS have made significant progress in Year 1

Project presented in multiple venues and conferences with positive feedback

- very good interactions with experiments and tool developers
    - Will target vCHEP next.

Ramping up effort (big challenge these days!)

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

Very hopeful that there will be a 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

pcalafiura@lbl.gov

Cast of Characters

PPS

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

not official list: taken from those who call into weekly meetings

IOS

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

CCE/PPS: Software Support Chart

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

🟩	Supported
▨	Under Development
🟧	3rd Party
🟥	Not Supported

Platform support still a moving target: Charles Leggett updates this chart 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

- 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: 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

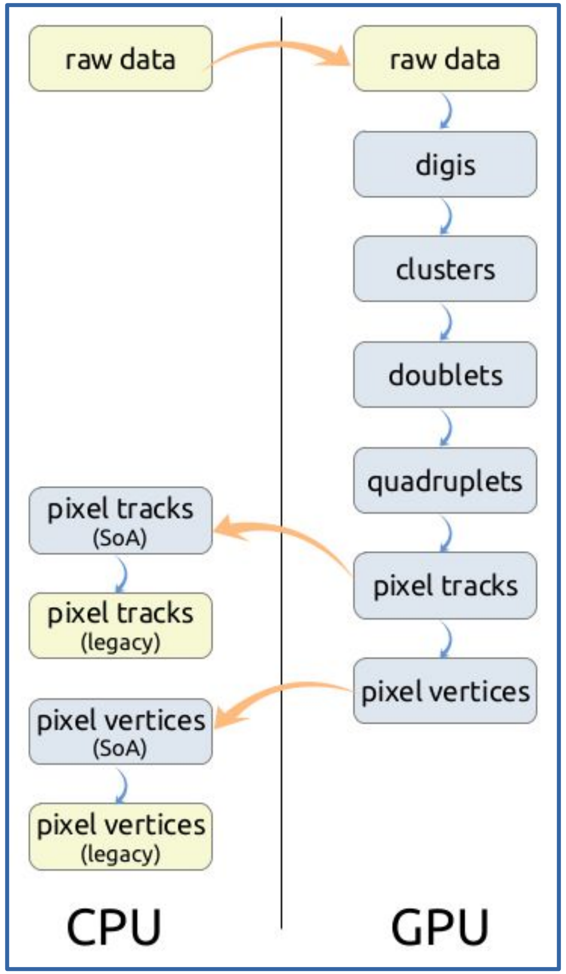
- decode the pixel raw data
- cluster the pixel hits (SoA)
- form hit doublets
- form hit quadruplets (or ntuplets) with a Cellular automaton algorithm – clean up duplicates

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



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

CCE/PPS: 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

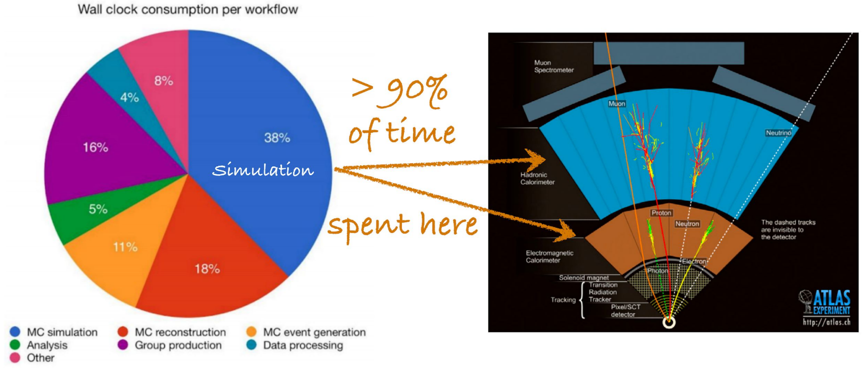
- less accurate, but much faster than Geant4

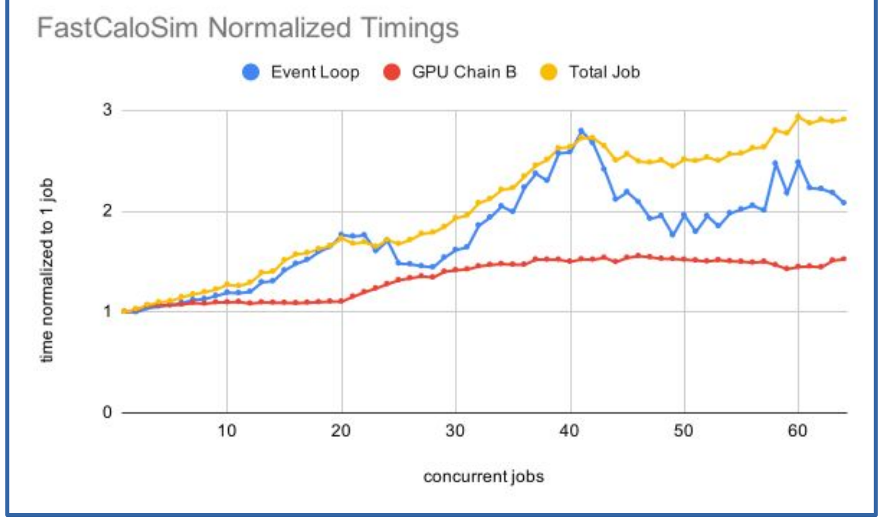
FastCaloSimV2: a relatively self-contained code base for fast ATLAS parametrized calorimeter simulation

BNL group has already created a CUDA port

- 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

Calorimeter-dominated





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

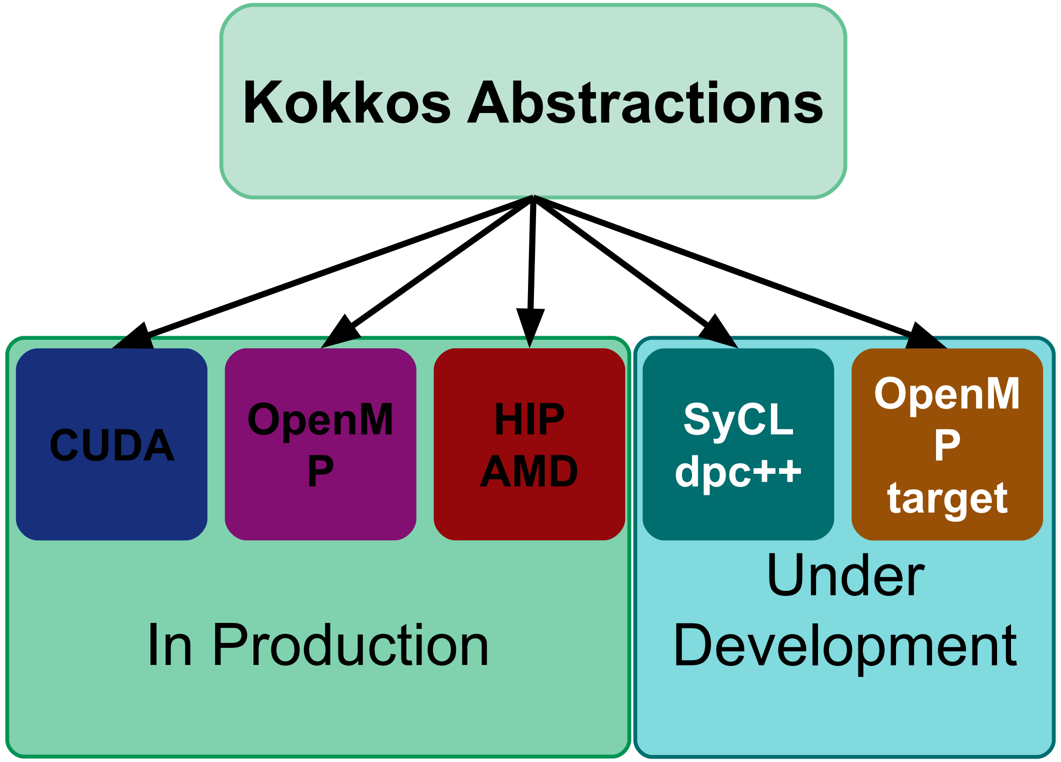
- parallel_for, reduction, scans

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

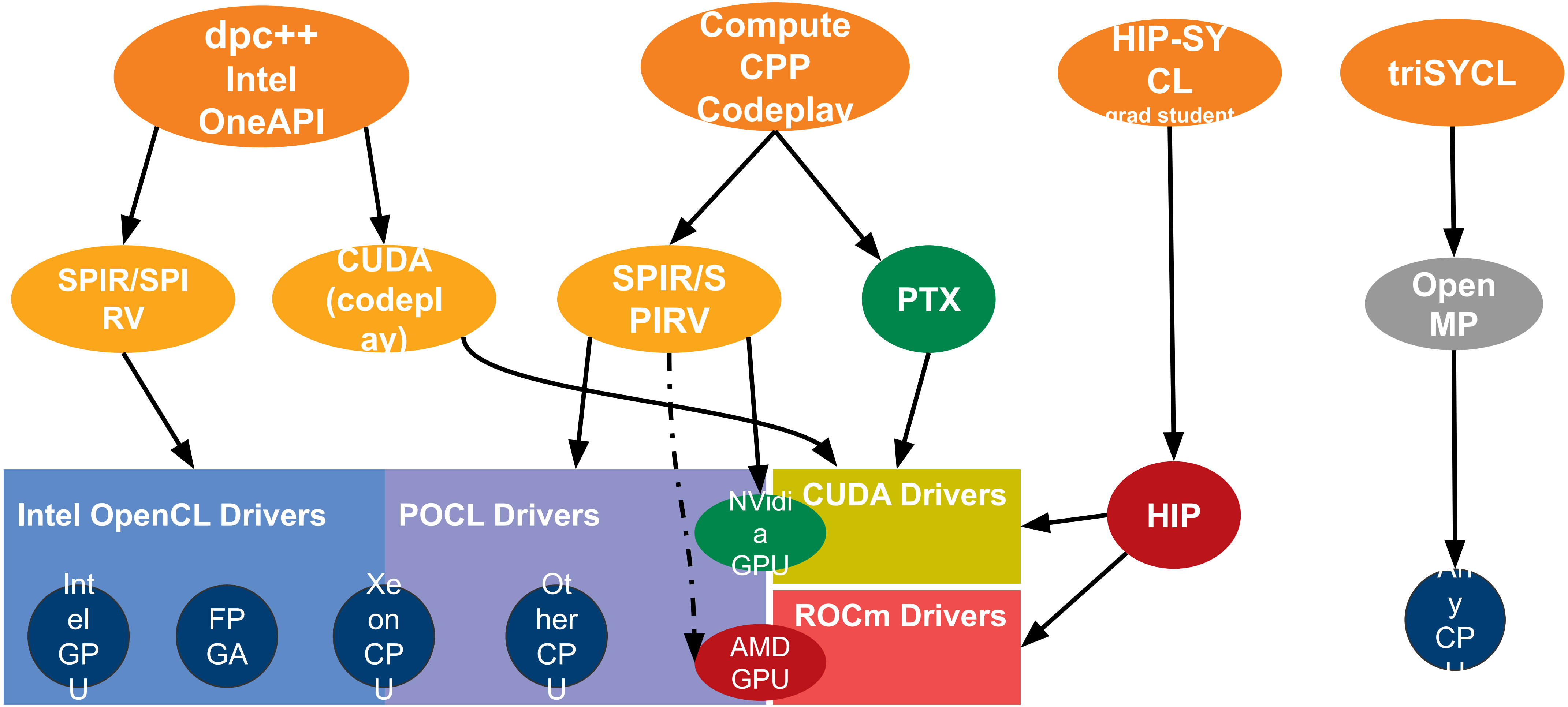
Long term OpenCL IR layer in question: Intel is replacing it with "LevelZero"

- 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



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

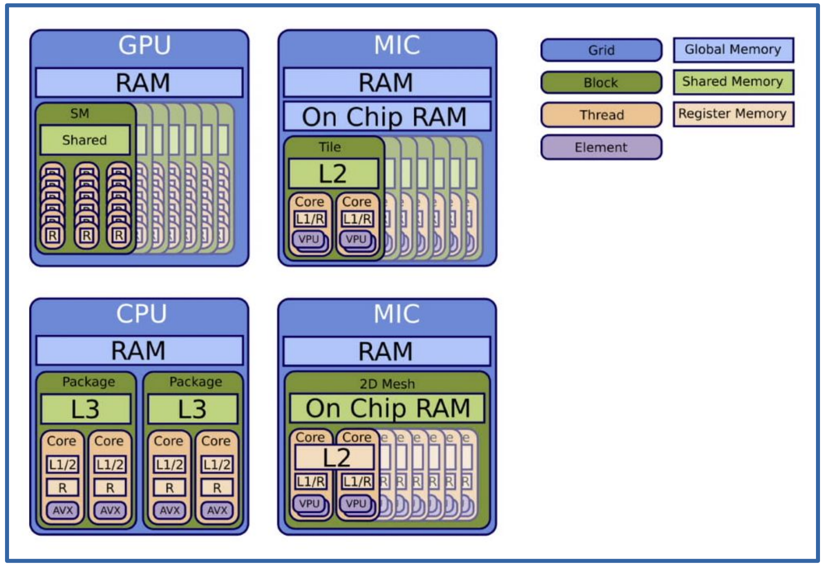
Data agnostic memory model:

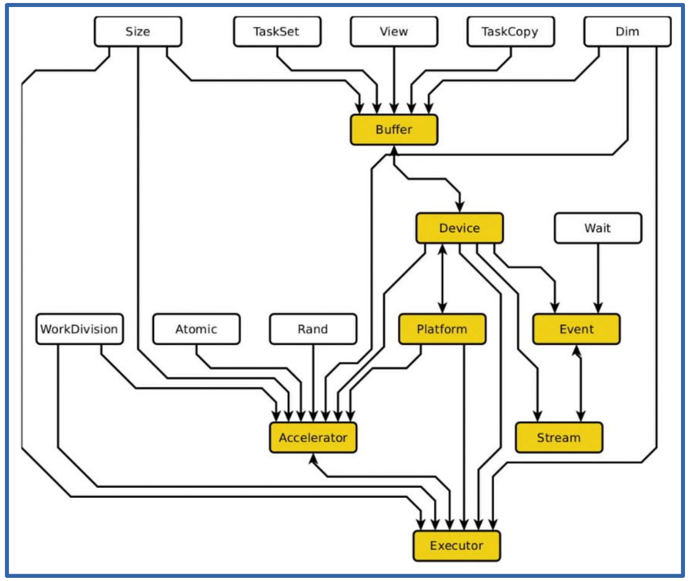
- allocates memory for you, but needs directives for hardware mapping
- same API for allocating on host and device

Uses templates for a "Zero Overhead Abstraction Layer"

Straightforward porting of CUDA kernels using cupla

- only include and kernel calls need to be changed


GPU
RAM
SM
Shared
R
MIC
RAM
On Chip RAM
Tile
L2
Core
L1/R
VPU
Grid
Global Memory
Block
Shared Memory
Thread
Register Memory
Element
CPU
RAM
Package
L3
Core
L1/2
R
AVX
MIC
RAM
2D Mesh
On Chip RAM
Core
L2
L1/R
VPU


Size
TaskSet
View
TaskCopy
Dim
Buffer
Device
Wait
WorkDivision
Atomic
Rand
Platform
Event
Accelerator
Stream
Executor