Results from the High Energy Physics Center for Computational Excellence and Future Plans

Charles Leggett and Peter van Gemmeren **for HEP-CCE**

HSF HCAF
June 14, 2023

What is HEP-CCE?

Three-year (2020-2023) pilot project

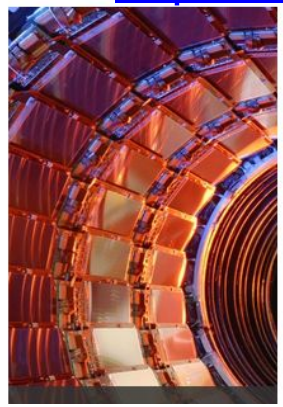
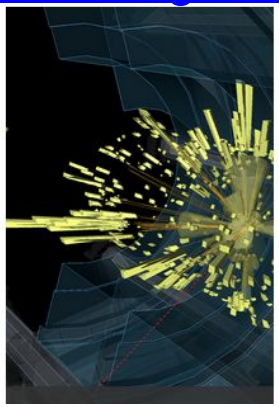
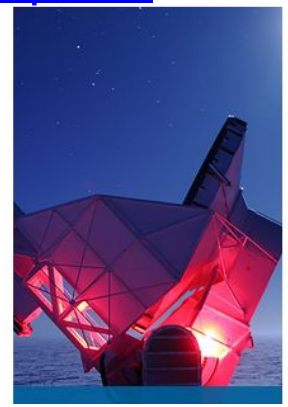
- Develop **practical** solutions to port hundreds of kernels to multiple platforms
- Collaborate with HPC & networking communities on **data-intensive** use cases

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, workflow instrumentation
3. EG: Optimizing event generators
4. CW: 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.
PI: Salman Habib (ANL), Co-PI: Paolo Calafiura (LBNL)

Portable Parallelization Strategies (PPS)

	CUDA	Kokkos	SYCL	HIP	OpenMP	alpaka	std::par
NVIDIA GPU			codeplay	hipcc			nvc++
AMD GPU			hipSYCL	hipcc			
Intel GPU			oneAPI				oneAPI:dpl
x86 CPU			oneAPI				gcc
FPGA							

circa 2019

Portable Parallelization Strategies (PPS)

	CUDA	Kokkos	SYCL	HIP	OpenMP	alpaka	std::par
NVIDIA GPU				*hipcc*	*nvc++ LLVM, Cray GCC, XL*		*nvc++*
AMD GPU			*openSYCL intel/llvm*	*hipcc*	*AOMP LLVM Cray*		
Intel GPU			*oneAPI intel/llvm*	CHIP-SPV: *early prototype*	*Intel OneAPI compiler*	*prototype*	*oneapi::dpl*
x86 CPU			*oneAPI intel/llvm openSYCL*	*via HIP-CPU Runtime*	*nvc++ LLVM, CCE, GCC, XL*		
FPGA				*via Xilinx Runtime*	*prototype compilers (OpenArc, Intel, etc.)*	*protytype via SYCL*	

***circa* today**

Portable Parallelization Strategies

Ported representative testbeds from ATLAS, CMS and DUNE to each portability layer.

	Kokkos	SYCL	OpenMP	Alpaka	std::par
Patatrack	Done	Done*	WIP	Done*	Done compiler bugs
Wirecell	Done	Done	Done	no	Done
FastCaloSim	Done	Done	Done	Done	Done
P2R	done	Done	OpenACC	Done	Done

Evaluated each porting experience according to a number of different objective and subjective metrics.

Metrics

- Ease of Learning
- Code conversion
  - From CPU to GPU and between different APIs
- Extent of modifications to existing code
  - Control of main, threading/execution model
- Extent of modifications to the Data Model
- Extent of modifications to the build system
- Hardware Mapping
  - Current and promised future support of hardware
- Feature Availability
- Interoperability
  - Interaction with external libraries, thread pools, C++ standards
- Address needs of large and small workflows
- Long term sustainability and code stability
  - Backward/forward compatibility of API and e.g. CUDA
- Compilation time
- Run time/Performance
- Ease of Debugging
- Aesthetics

HEP Testbeds

FastCaloSim

- ATLAS parametrized LAr calorimeter simulation
- 3 simple kernels (large workspace reset, main simulation, stream compaction)
- 1-D and 2-D jagged arrays
- small data transfer d->h at end of each event

Patatrack

- CMS pixel detector reconstruction
- 40 kernels of varying complexity and lengths (many are short)
  - good test for latency, concurrency, asynchronous execution, memory pools

Wirecell Toolkit

- LArTPC signal simulation
- 3 kernels: rasterization, scatter-add, FFT convolution

p2r

- CMS "propagate-to-r" track reconstruction in a single kernel

Kokkos

- Higher level of abstraction than CUDA
- Requires explicit initialization and finalization of runtime
- Separate compilation of library for different backends and features
  - implications for code distribution
- Can mix and match native and Kokkos kernels in same application

- Good performance for simple and long running kernels
  - hides overheads from initialization of data structures and kernel launches
- No native support for jagged arrays
- Concurrent kernels only with CUDA backend and CUDA specific features
  - concurrent calls to serial backend safe, but not efficient
- No common API to vendor optimized FFT, RNG (though some built-in)
- Poor interoperability with external concurrency mechanisms and thread pools (eg TBB)

- Very good developer support

Alpaka

- Verbose API, sparse documentation, steep learning curve
  - responsive and helpful developers
- very small user community
- Need either to wrap kernels in callable objects or to use lambdas, heavy use of typedefs
  - compiler error messages hard to decipher
- For memory transfers between host and device one can use either alpaka buffers (takes ownership of the allocation) or alpaka views (to copy already allocated memory)
  - Our attempts to use alpaka views inside FastCaloSim led to random crashes
  - Patatrack didn't have these issues
- Extensive configurability with CMake
- Can mix in native kernels and libraries in same application
- Performance comparable to native

SYCL

- Requires different compilers for different backends
  - oneAPI (Intel), llvm/sycl (Intel,NVIDIA,AMD), openSYCL (Intel,NVIDIA,AMD)
- Simplified code design as associates data dependencies with kernels for automatic data migration
  - can get better performance with explicit transfers
- No current support for concurrent kernels with any backend
  - though in theory supported by the standard
- Good interoperability with external concurrency layers like TBB, OMP, MPI
- Mostly seamless integration with build systems
- Near native performance (*cf* CUDA / HIP)
- Strongly supported by Intel

OpenMP Target Offload

- Widest difference of opinion in people's porting experience
  - some found it very easy to change serial code
  - some experienced large challenges
- Significant variation between performance characteristics of different compilers
  - compiler options
  - tuning of threads/teams
- Specialized operations unsupported (scan, memset) or much less performant (atomics) than CUDA

- Documentation is sparse, especially for more advanced features
- Debugging and profiling very challenging due to extra OpenMP code infrastructure and architecture-specific plugins loaded at initialization

std::par / nvc++

- Pure C++ - no learning curve (beyond using STL algorithms)
  - slightly convoluted indexed container access with C++17
- Not equivalent to CUDA/SYCL/Kokkos
  - no low level controls
  - not intended to be a CUDA replacement - rather a stepping stone to parallelism
- Support for NVIDIA GPUs (nvc++), Intel GPUs (oneAPI::dpl), and multicore CPUs (nvc++, g++)
- nvc++ is still immature
  - bugs (can't compile ROOT yet)
  - need workarounds to compile parts of projects w/ g++, parts with nvc++
  - any offloaded data must be allocated by code compiled with nvc++
  - compilation much slower than g++
  - unusual memory transfer speeds from AMD CPUs
- Very good performance for longer kernels that translate well into Thrust
- Path towards C++ standards based implementation of GPU offloading
  - C++26 std::exec (schedulers, senders, receivers), etc

Portable Parallelization Strategies: Recommendations

Software and hardware are still rapidly changing

- Lots of interactions with API developers in hackathons and to fix bugs
- Results remain preliminary

API recommendations are very application dependent

- All perform approximately equally for simple kernels
- Complex algorithms and chained kernels bring out weaknesses of all APIs
  - interaction with external libraries adds extra complexities
  - even compilation can be an issue

Learning curve / language complexity of APIs not all the same

- std::par → OMP → Kokkos / SYCL → alpaka → OMP
- subjective and dependent on code complexity and previous experience

Porting from Serial CPU code → GPU concepts is the biggest hurdle

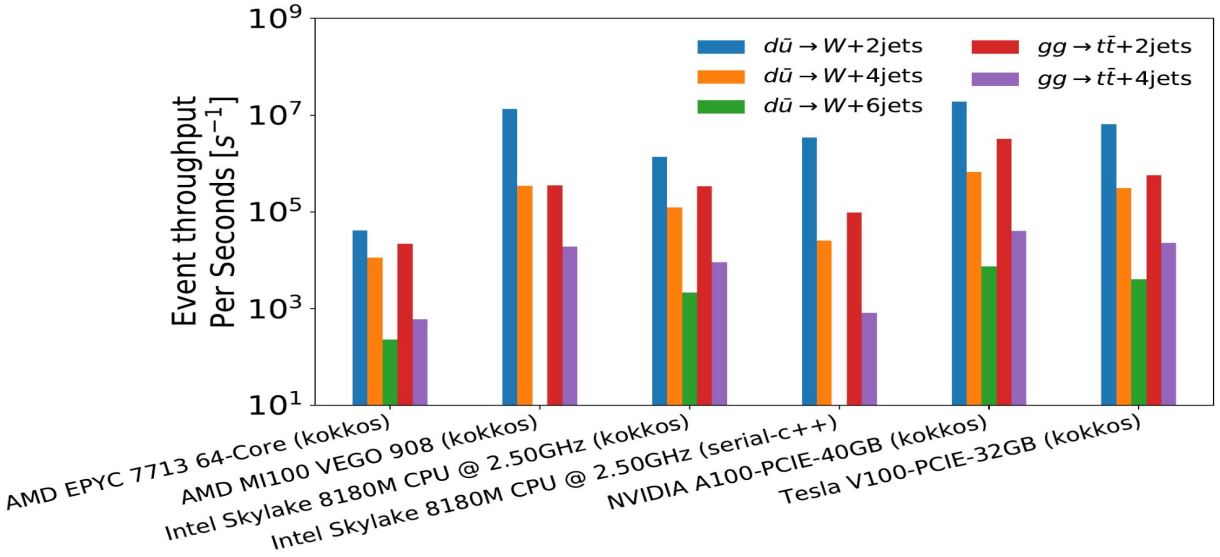
- starting with optimized code is extra challenging

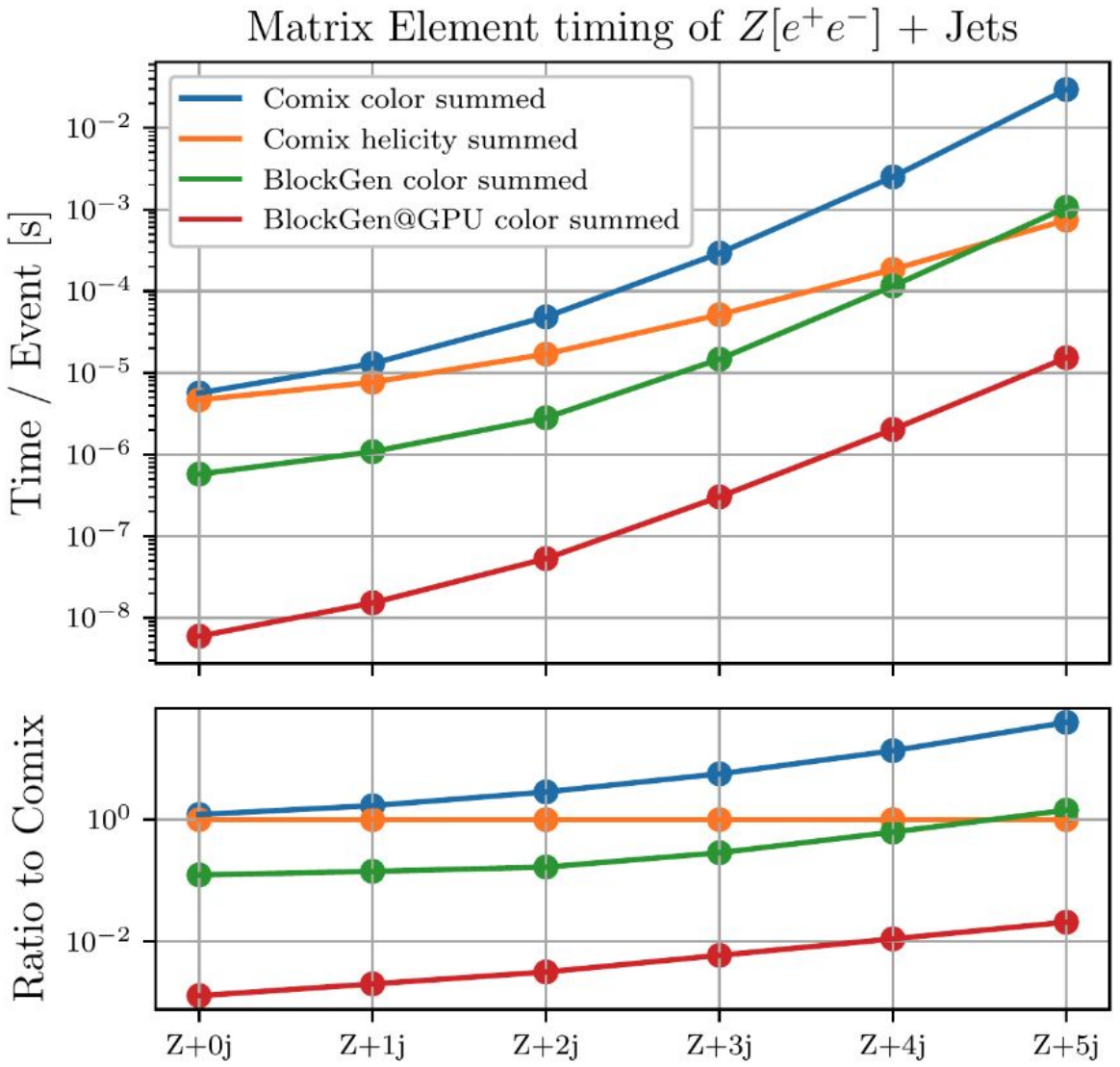
Very hard to extrapolate to next five years or beyond

- Vendors are pushing in different directions (but towards standards)
- Increasing proximity of CPU / GPU / memory will have significant impact

Event Generators

Pepper: A New Leading-Order Matrix Element Integrator



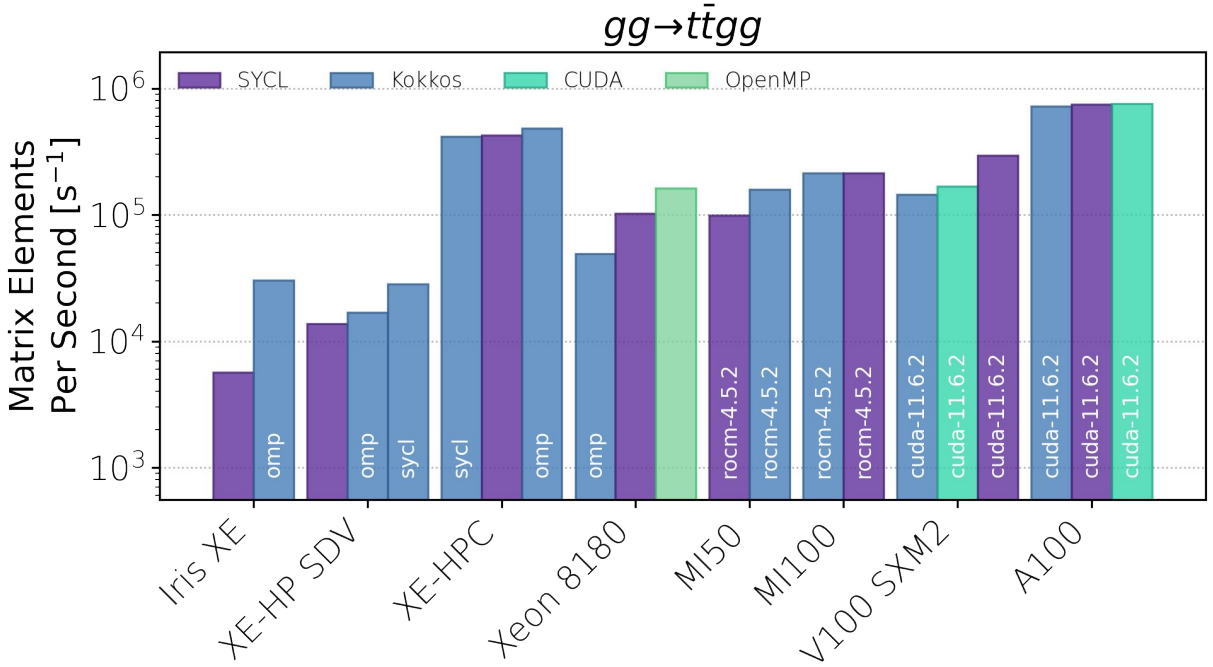


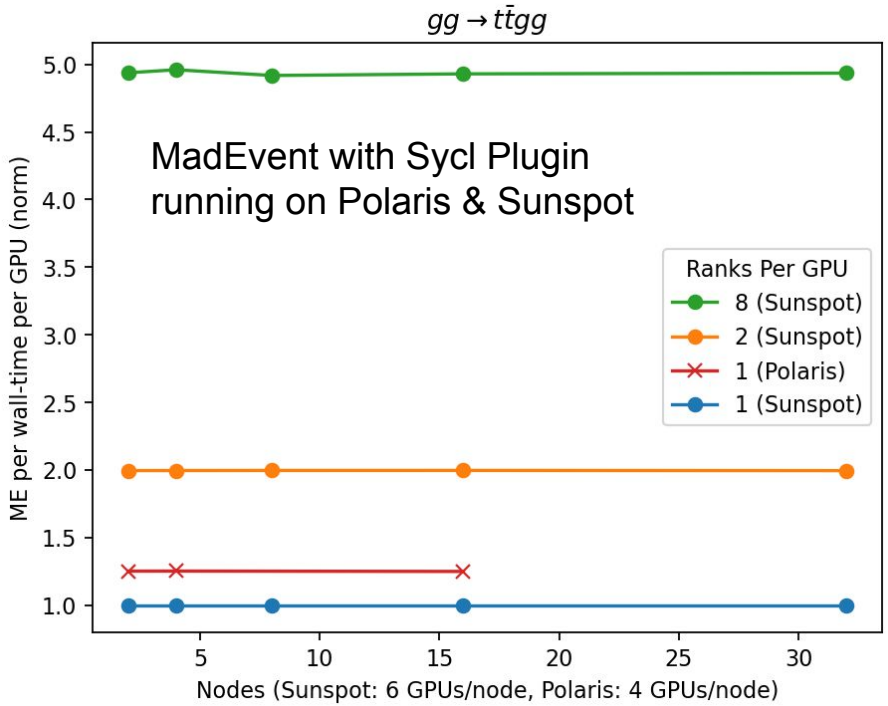
New LO generator developed from the ground up for parallel architectures [alg-paper]

- Strong performance for CUDA over a wide range of processes

HEP-CCE facilitating portability via Kokkos port of the algorithm with excellent results on multiple modern hardware

MadGraph: Portable MadEvent





- Working with a team of developers from CERN and UCLouvain on a GPU version of the LO MadEvent generator with Matrix Element (ME) calculations ported to Kokkos/SYCL
- HEP-CCE developed Kokkos version and took over the Sycl version
- Kokkos and SYCL have enabled running on a broad variety of architectures including multicore CPUs, AMD and NVIDIA GPUs, and new Intel GPUs on Sunspot (Aurora) including oversubscription models, with good performance characteristics (now)

Next Steps for HEP-CCE in EvGen

- Generally would like to move on to NLO event generation on GPUs
- There are several challenges with this:
  - some technical:
    - often NLO libraries require quad precision (not supported on GPUs, in fact AI pushing 16- and 8-bit precision)
    - NLO algorithms are very divergent (not GPU friendly)
  - some political:
    - loop libraries are developed by non-DOE, non-US collaborations
    - Newly awarded SciDAC includes porting the only (?) NLO code, MCFM, where the developer is US-based (and DOE-based).

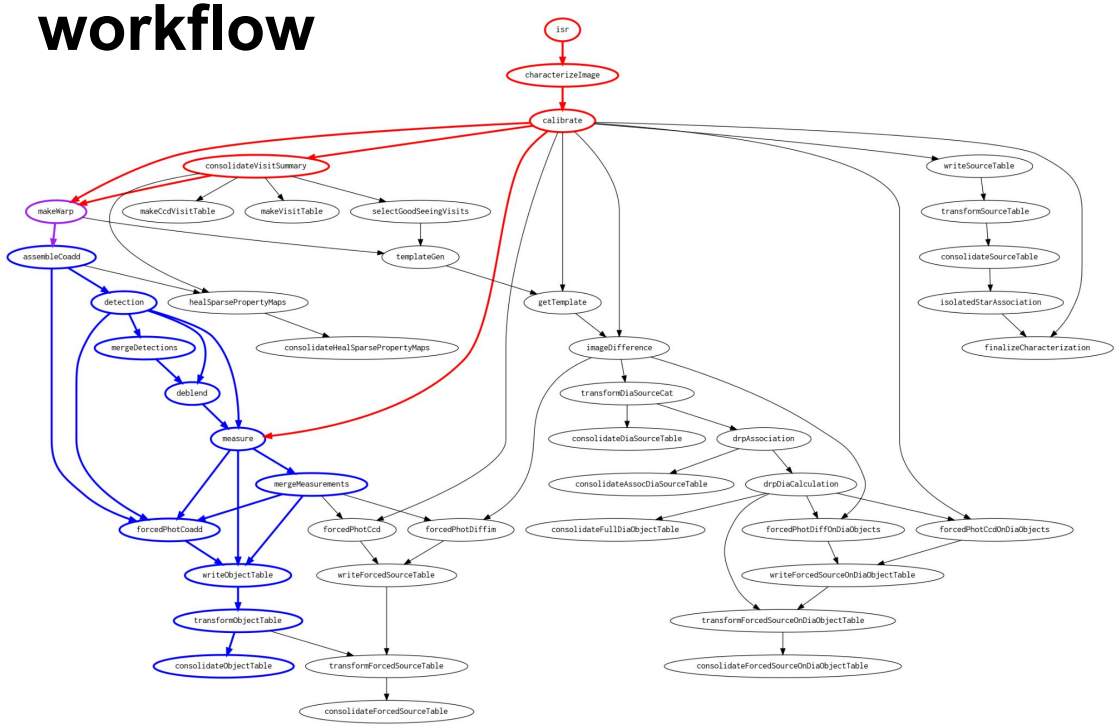
- Some options for moving forward:
  - Continue work with MadGraph (CERN+UCLouvain) group who are discussing porting one (Ninja) of the four loops libraries MadGraph is compatible with.
  - Complementary to the MCFM port would be making a GPU version of Pythia for showering, benefits are Pythia devs are US+DOE based.

Complex Workflows

HEP Experiments require HPC workflows

- HEP has many workflow technologies, serving specific use-cases, that are monolithic, and difficult to extend
- Performance on leadership platforms is complex even for simple workflows; HPC workflows will become more important, but increasingly harder
- Opportunity to harmonize use of experiment-agnostic components, integrable solutions for extensibility and modularity on leadership platforms.
- HEP-CCE workflows group (including workflows and experiments) identified common challenges and prototyped cross-workflow approaches
  - Interoperability between workflow systems
  - Common task graph representation
  - Streamlined remote execution
  - Diverse monitoring information

Rubin LSST image processing workflow



Graph showing dependencies between task types for Rubin image processing. red operate on CCD-visits, blue tasks on patches, and purple on both.

Plans for CCE-2

The following are tentative ideas

We are seeking **community input** for directions to go and challenges to explore

Applying Lessons Learned by PPS

Work with experiments to develop tailored recommendations

- experiment size, resources and timescale
- existing code and libraries
- data and data structures

Develop experiment-agnostic algorithm examples

- cookbooks - if "scenario == this"; then do "that"
- training examples

Develop pre-packaged mini-apps for ASCR facility testing and next gen planning

- could integrate into HEP benchmarks like SPECHPC or HEPScore
- ability to influence design and parameters of next gen facilities

Continue monitoring evolving hardware and software

- landscape is still changing rapidly - evolutionary or revolutionary?
- use testbeds/mini-apps to track changes

From Portable Applications to Portable Workflows

Increasing use of HPCs and HTCs in our computing workflows

- very challenging to integrate
- each platform is almost a one-off

Need to develop tools, services and policies to homogenize and simplify this process

- identity management
- software delivery and container management
- computing and storage resource brokering with focus on resource availability and overall throughput
- scalable, distributed execution engines (Dask, HPX, etc)
  - FaaS (funcX / parsl), Inference as a service (Sonic / Triton)
- data cataloging, delivery and access (xrootd, Globus, rucio)
- edge services
  - pilot submission (Harvester), remote logging and reporting, database access

Machine Language Training for HEP

Current training times for ML models like anomaly detection, pattern rec, detector simulation require enormous resources

- multiple days to train
- large datasets, huge memory requirements

Future models are orders of magnitude more resource intensive

- PB training data, TB of memory, thousands of hours to train

Similarly to CCE/PPS, identify and develop suitable scalable training solutions for HEP applications

- evaluate performance for HEP applications of distributed data-parallel ML training components
  - PyTorch DistributeDataParallel
  - Tensorflow MultiWorkerMirroredStrategy
- optimizing tools
  - Horovod, Raytune,
  - HEP-developed iDDS

à continuer...

This work was supported by the U.S. Department of Energy, Office of Science, Office of High Energy Physics, High Energy Physics Center for Computational Excellence (HEP-CCE). This research used resources at Argonne Leadership Computing Facility, NERSC and BNL Scientific Data and Computing Center.