Portable Parallelization Strategies - Overview and Metrics

Martin Kwok(Fermilab)

On behalf of the CCE-PPS team

IRIS-HEP topical meeting

04 August, 2021

What is HEP-CCE?

- HEP-CCE (Center for Computational Excellence)
  - A 3-year (2020-2023) pilot project, funded by US DOE
  - Formed by 4 US labs, 6 experiments, covering cosmic/intensity/energy frontiers
- Develop common strategies to common computational challenges for HEP community
  - Specifically to efficiently run HEP software on HPCs

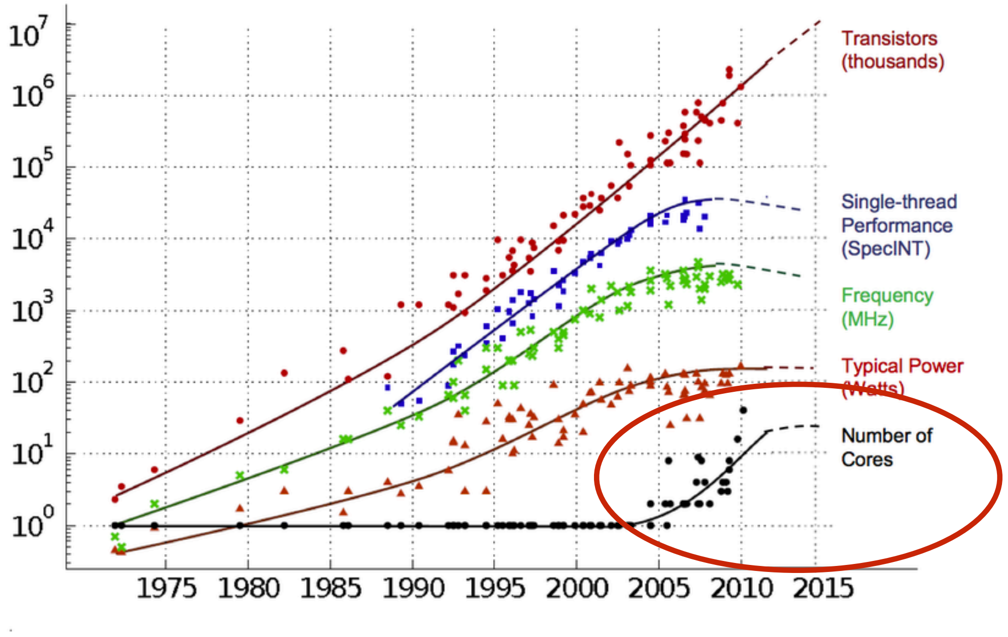
HEP-CCE

Argonne NATIONAL LABORATORY | BROOKHAVEN NATIONAL LABORATORY | Fermilab | BERKELEY LAB Bringing Science Solutions to the World

1. PPS: Portable Parallelization Strategy ← Focus of today
2. IOS: HEP I/O and HPC storage issues
3. EG: Event Generation
4. CW: Complex Workflow on HPC

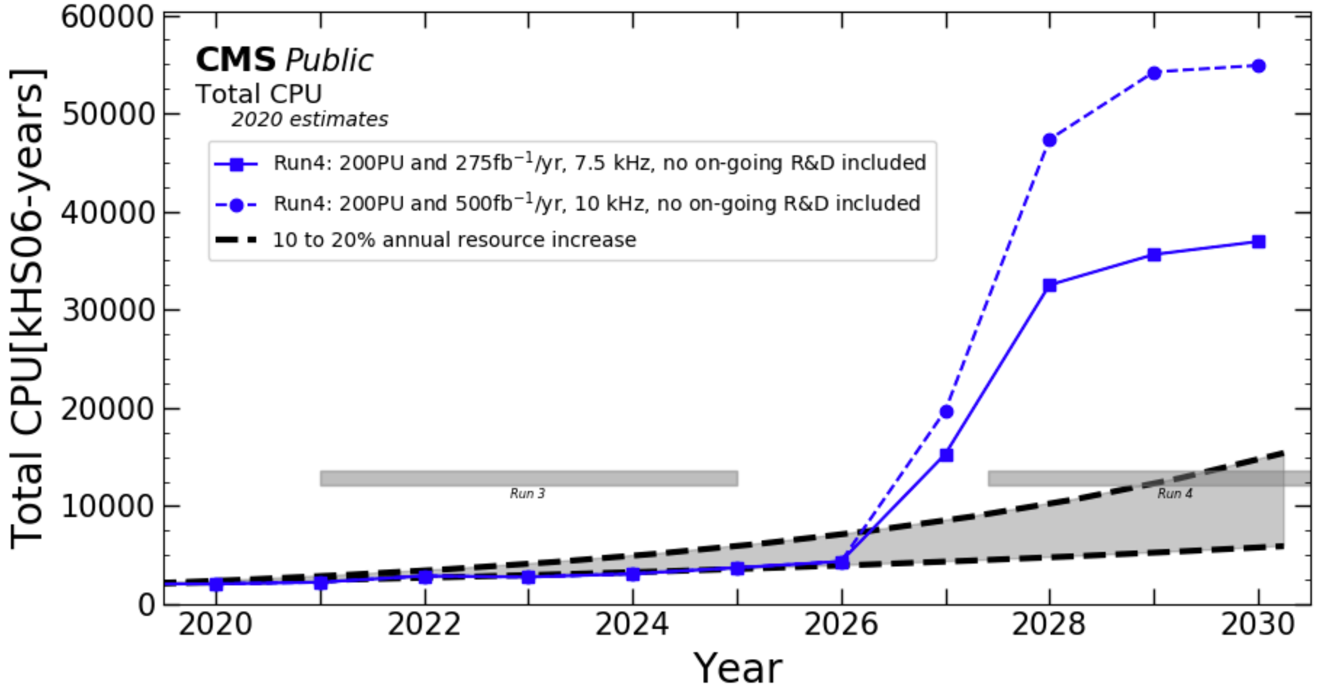
Parallelization: Common challenge to HEP

- Demand for processing power from HEP experiment will outgrow expected resource increase
  - "Buy more CPU" does not work
  - In 2030, LHC experiments will need O(100) PFlops/s
- *Parallelism* is one of the key to meet growing computing power
  - Lesson from the past: single-core CPU frequency has plateaued since ~2005
  - Multi-core CPU flourished



Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore

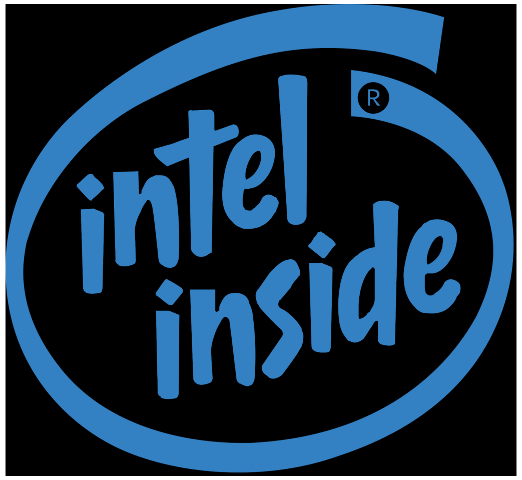
arxiv:1604.00319

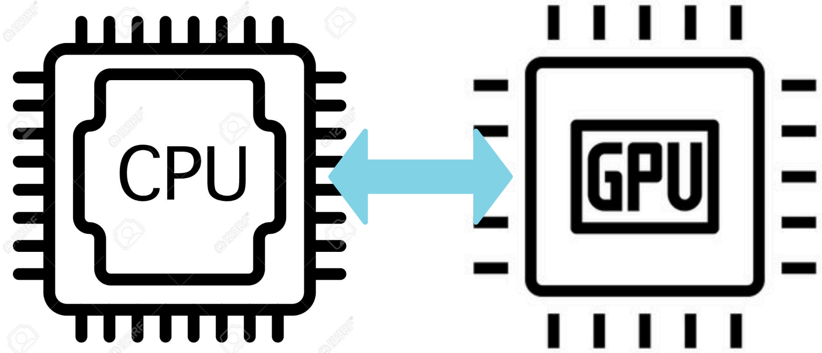


Parallelization: Common challenge to HEP

- Extend the parallelism:
  - Offload computational intensive tasks to a custom architecture for parallel computation (an accelerator)
  - Could be a GPU or even a FPGA
- Exascale HPC systems could provide the computing power
  - Taking advantage of CPU + GPU
- Mis-match between software and hardware:
  - HEP software predominantly written for x86 platforms
  - Offloading to another architecture is a different programming paradigm
    - Needs more effort to use accelerator efficiently

HEP software





Perlmutter Phase 1 @NERSC

Ranked 5 in the Top-500 list

700k+ cores, ~65 PFlops/s

Portability: HPC landscape

- Accelerator architectures are proliferating
  - Main GPU manufacturers: NVidia, Intel, AMD
  - FPGA is possible but is less mature
- We need portable solutions
  - Cannot afford to re-write the code for each architecture
  - Aim at maintaining a single code-base for all target architectures

		Accelerators				
		Intel	NVIDIA	AMD	FPGA	Other
CPU	Intel	Aurora	WLCG (HEP) Cori GPU Piz Daint Tsukuba MareNostrum		Tsukuba	
	AMD		WLCG (HEP) Perlmutter	Frontier El Capitan		
	IBM		Summit Sierra MareNostrum			
	Arm		Alps			Astra
	Fujitsu					Fugaku

Portability: Software Support

- No universally accepted best way to write performance, portable GPU code
  - Each portability solution has strength and weakness

	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

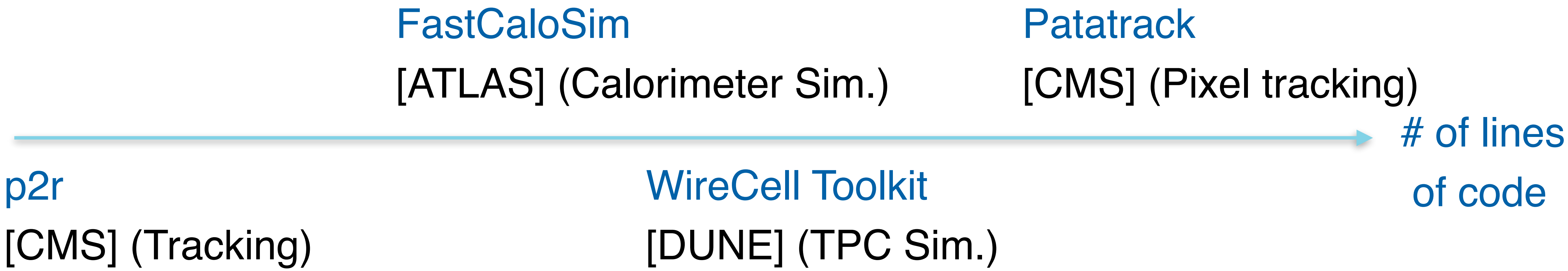
🟩	Supported
◐	Under Development
🟧	3rd Party
🟥	Not Supported

The technologies are still evolving, platform support is a moving target

Strategy: The plan for PPS

- Investigate a range of software portability solutions
  - Kokkos
  - SYCL/ dpc++
  - Alpaka
  - OpenMP/OpenACC
- Experiment each solution with various HEP applications
  - Covers different HEP experiments, code sizes, physics problems
  - Diversity of the test application tells us different aspects of the technology
- Evaluate each solution with a set of metrics
  - Capture the full porting experience (Porting, building, performance etc.)
- Make recommendations to the experiments
  - Addresses different needs for different workflows

Detailed results
In Haiwang's talk



Kokkos

https://github.com/kokkos/kokkos

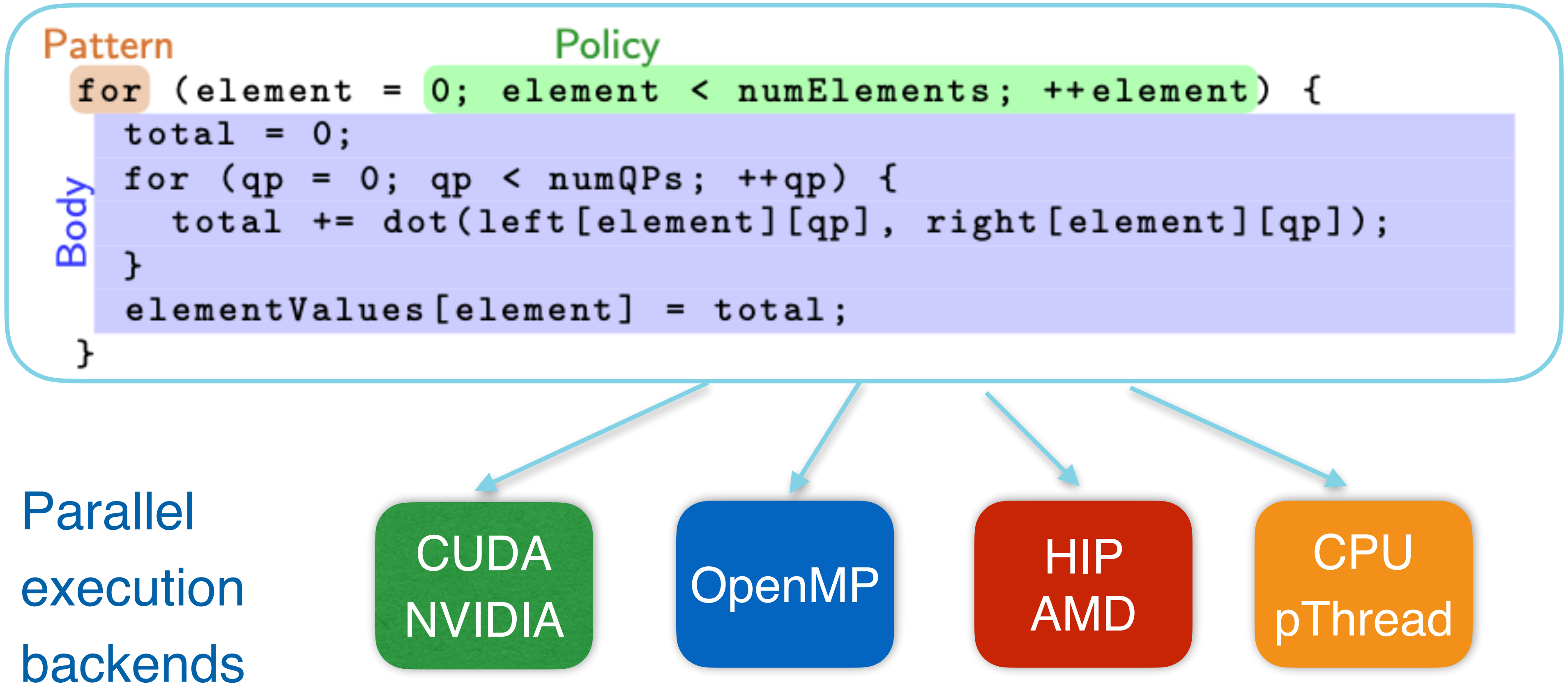
- Single source C++ template library
- Aims to be descriptive, not prescriptive
  - Kokkos handles the backend (relatively high-level API)
  - Abstraction layer provides handle for efficient data layout for both GPU/CPU + more
- One host(CPU)+parallel (GPU/CPU) backend chosen at compile time
  - Supports NVIDIA/AMD GPU
  - Write once, compile for different architectures
- Active Kokkos development, and user support
  - Open source

Pattern: nature of work

Execution Policy: How computations are executed

Body: Unit of work

Kokkos Abstraction



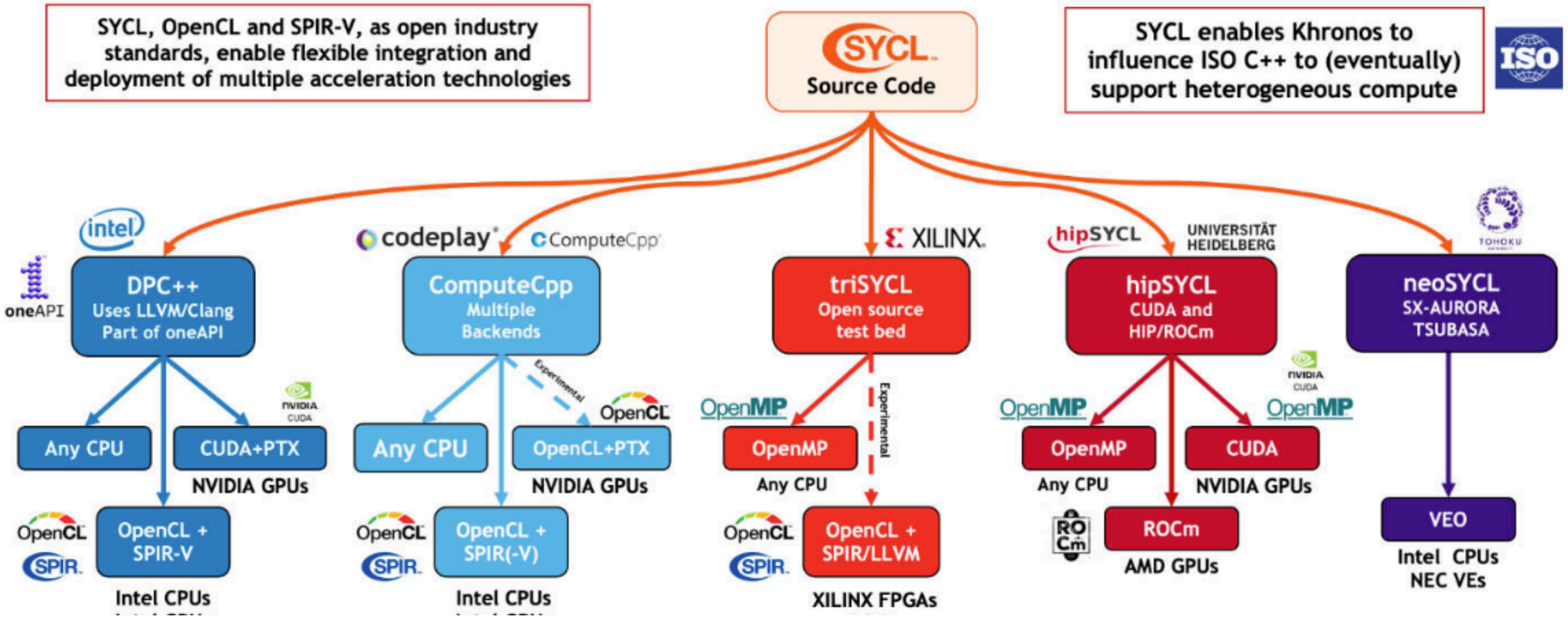
```
for (element = 0; element < numElements; ++element) {
    total = 0;
    for (qp = 0; qp < numQPs; ++qp) {
      total += dot(left[element][qp], right[element][qp]);
    }
    elementValues[element] = total;
}
```

Parallel execution backends

SYCL / Data Parallel C++ (DPC++)

- SYCL is a specification of single-source C++ programming model for heterogeneous computing
  - Different compilers implements the specification
  - Intel's oneAPI and Data Parallel C++ (DPC++) supports Intel CPU, GPU, FPGA
  - Parallelism based on C++17 standard
- Execute on most architectures
  - Including CPU, GPU, FPGA
- Exact behavior of program depends on compiler
  - Compilers are evolving rapidly
- C++ standard for heterogeneous computing is still evolving
  - Features are growing

Complex ecosystem



SYCL, OpenCL and SPIR-V, as open industry standards, enable flexible integration and deployment of multiple acceleration technologies

SYCL Source Code

SYCL enables Khronos to influence ISO C++ to (eventually) support heterogeneous compute

DPC++ Uses LLVM/Clang Part of oneAPI → Any CPU; CUDA+PTX NVIDIA GPUs; OpenCL + SPIR-V Intel CPUs

ComputeCpp Multiple Backends → Any CPU; OpenCL+PTX NVIDIA GPUs (Experimental); OpenCL + SPIR(-V) Intel CPUs

triSYCL Open source test bed → OpenMP Any CPU; OpenCL + SPIR/LLVM XILINX FPGAs (Experimental)

hipSYCL CUDA and HIP/ROCm → OpenMP Any CPU; CUDA NVIDIA GPUs; ROCm AMD GPUs

neoSYCL SX-AURORA TSUBASA → VEO Intel CPUs NEC VEs

First year of CCE-PPS

- Projects typically start with parallel CPU/ CUDA implementation
  - Target a portable solution
    - Could take a long time, depending the size of project
  - Benchmarking/Optimize implementation
    - See results from Haiwang's slides

- Designed metric for evaluating different solutions
  - Lots of useful lessons learned from hands-on experience
  - Collecting results among ourselves from different projects

		Implementation			
	Projects	***CPU***	***CUDA***	***SYCL***	***Kokkos***
1	FastCaloSim	🟩	🟩	🟩	🟩
2	Patatrack	🟩	🟩		🟩
3	Wirecell	🟩	🟩		🟨

□ Planned 🟨 In progress 🟩 Completed

Metrics: Evaluation

1. Ease of learning and extend of code modification
2. Code conversion
3. Impact on existing code
   - Control of main(),
   - Threading/execution model
4. Impact on EDM/ data
5. Impact on existing toolchain and build infrastructure
6. Hardware mapping
   - Current and future backend support
7. Feature availability
   - Reduction, concurrent kernels etc.
   - Interface to commonly used math libraries
8. Ease of debugging
9. Address needs of all workflows
   - Scaling with # kernels / applications
10. Long-term sustainability and code stability
11. Compilation time
12. Performance/Run time
    - Compares with native implementation (e.g. CUDA for NVidia GPUs)
13. Interoperability
    - Run different technologies in same application?
14. Aesthetics

Metric progress

🟥 Needs input

🟨 Some input there

🟩 Good amount of content

	Topics	*Metric formulation*	*SYCL*	*Kokkos*
1	Ease of Learning Language	🟩	🟩	🟩
2	Code conversion	🟩	🟩	🟩
3	Extent of modification to existing code	🟩	🟩	🟩
4	Extent of modifications to EDM / Data	🟨	🟩	🟨
5	Extent of modifications to build rules	🟩	🟩	🟩
6	Hardware Mapping	🟩	🟩	🟩
7	Feature Availability	🟨	🟨	🟨
8	Address needs of all workflows	🟥	🟥	🟥
9	Long term sustainability and code	🟩	🟨	🟨
10	Compilation time	🟩	🟩	🟩
11	Run time	🟨	🟨	🟨
12	Ease of debugging	🟩	🟨	🟨
13	Aesthetics	🟩	🟨	🟨
14	Flexibility	🟩	🟥	🟥

Summary and moving forward

- PPS stands for Portable Parallelization Strategy
  - Develop a common strategy for parallelization of HEP software
  - One of the key ingredients to meet the demand for HEP computing in the next 10 years
  - Fruitful results from 1st year: See Haiwang's talk up next

- What's next for PPS?
  - New use-case: A Common Tracking Software (ACTS)
    - HEP common software, instead of experiment-specific application
  - Propagate-to-R (P2R)
    - Very light-weight tracking application [O(1000) lines] to speed-up exploration of technologies
  - Explore new technologies
    - Alpaka: Europe-backed C++ parallel programming library, similar to Kokkos
    - `std::par`: Parallel execution policy in the new C++ standard
  - Summarize experience from difference projects into metrics/recommendations
    - Documentation of specific details could be useful for future decision making