Fermilab U.S. DEPARTMENT OF ENERGY Office of Science

Application of performance portability solutions for GPUs and many-core CPUs to track reconstruction kernels

Martin Kwok(Fermilab)
On behalf of the p2r and p2z team

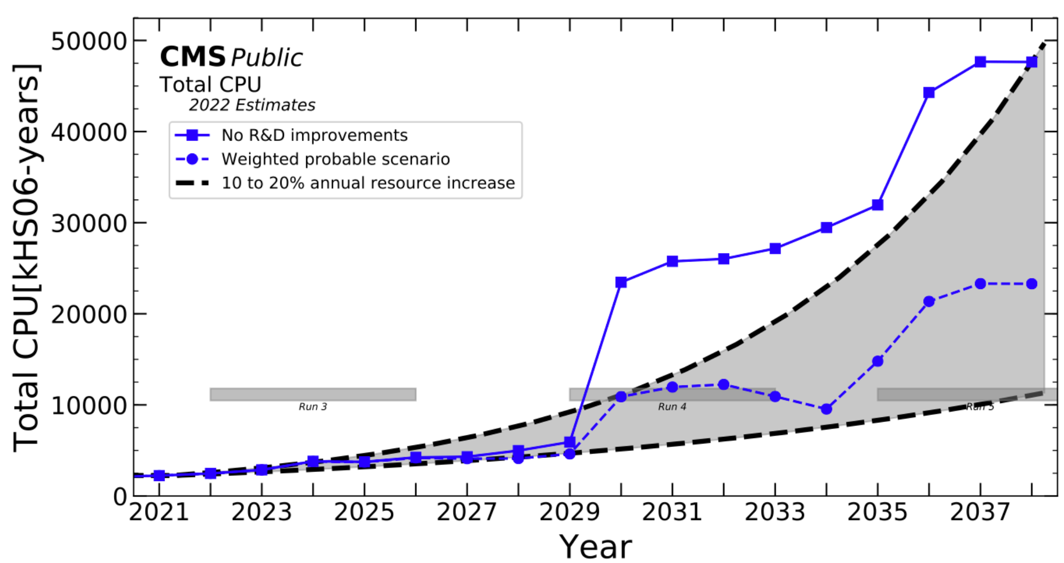
CHEP 2023
8 May, 2023

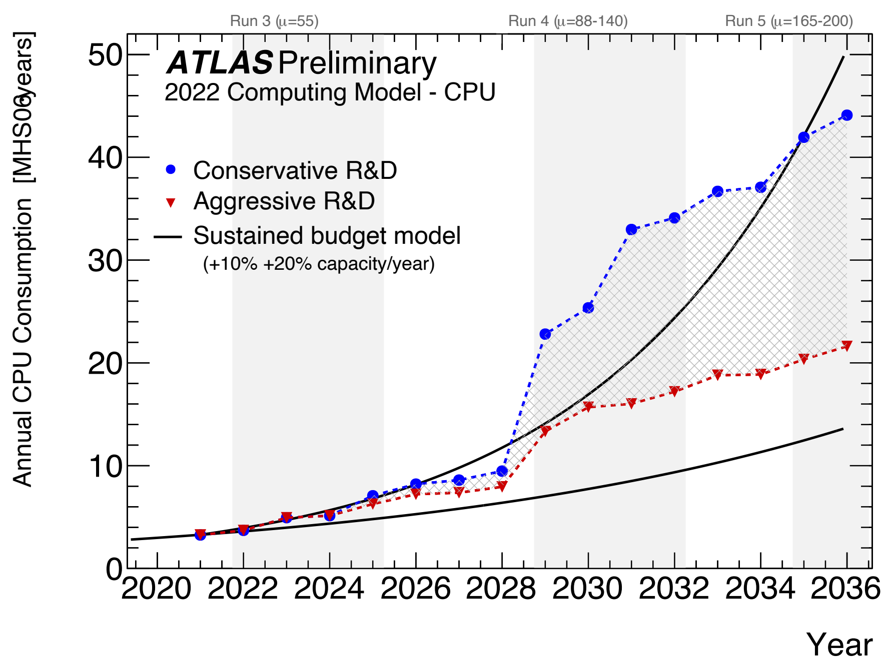
HEP-CCE

Argonne NATIONAL LABORATORY Brookhaven National Laboratory Fermilab BERKELEY LAB Bringing Science Solutions to the World

Performance portability

- Heterogenous computing is one of the key to meet the HL-LHC computing challenge
- Challenges of HEP computing:
  - Hundreds of computing sites (grid clusters + HPC systems + clouds)
  - Hundreds of C++ kernels (several million line of code, no hot-spots)
  - Hundreds of data objects (dynamic, polymorphic)
  - Hundreds of non-professional developers (domain experts)
- Portability:
  - Support multiple accelerator platforms with minimal changes to code base
- Performance portability:
  - Efficient use of CPU and GPU





Portability: Software landscape

HEP-CCE

- Rapidly changing ~O(month) portability solutions
  - New features/compiler supports/New backend
- Different approaches:
  - Compiler pragma-based approach
  - Libraries
  - Language extension
- HEP-CCE: Joint effort of major U.S. National labs involved in HEP
  - Investigate different portability solutions in HEP context

Software →

Hardware ↓

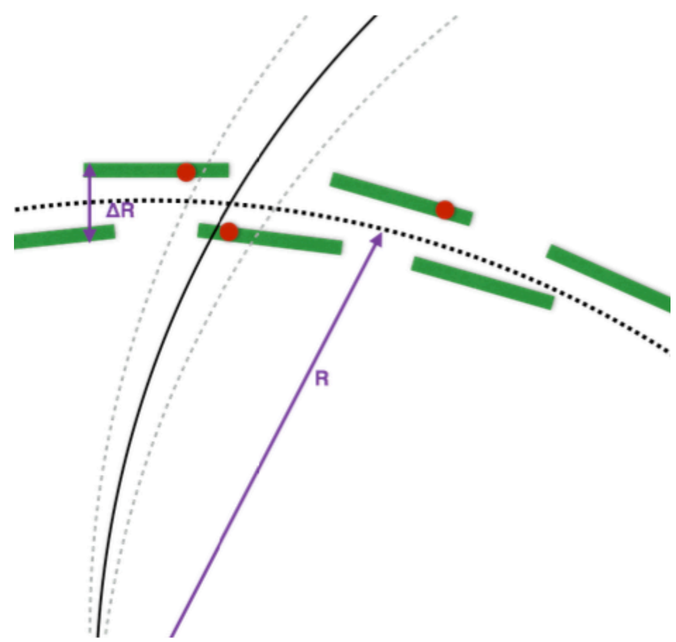
	CUDA	Kokkos	SYCL	HIP	OpenMP	alpaka	std::par
NVIDIA GPU			*intel/llvm compute-cpp*	*hipcc*	*nvc++ LLVM, Cray GCC, XL*		*nvc++*
AMD GPU			*openSYCL intel/llvm*	*hipcc*	*AOMP LLVM Cray*		
Intel GPU			*oneAPI intel/llvm*	CHIP-SPV: *early prototype*	*Intel OneAPI compiler*	*prototype*	*oneapi::dpl*
x86 CPU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

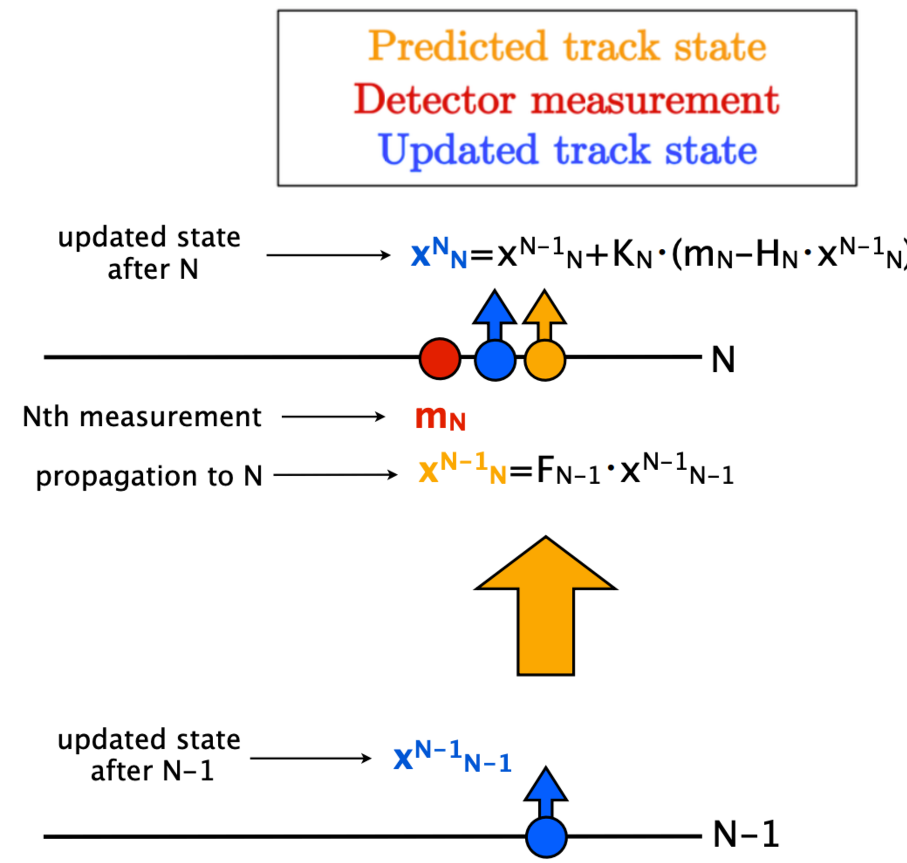
Stay tuned tomorrow for! [HEP-CCE result]

The p2r/p2z program

mkFit: https://arxiv.org/abs/2006.00071
p2r: https://github.com/cerati/p2r-tests
p2z:https://github.com/cerati/p2z-tests

- Track reconstruction is one of the most computational intensive task in collider experiments such as the LHC at CERN
- p2r & p2z are a standalone mini-app. to perform core math of parallelized track reconstruction
  - Build tracks in radial direction from detector hits (propagation +Kalman Update)
    - Different propagation matrix in R / Z direction
  - Lightweight kernel extracted from a more realistic application (mkFit, vectorized CPU track fitting)
- Together forms the backbone of track fitting kernels

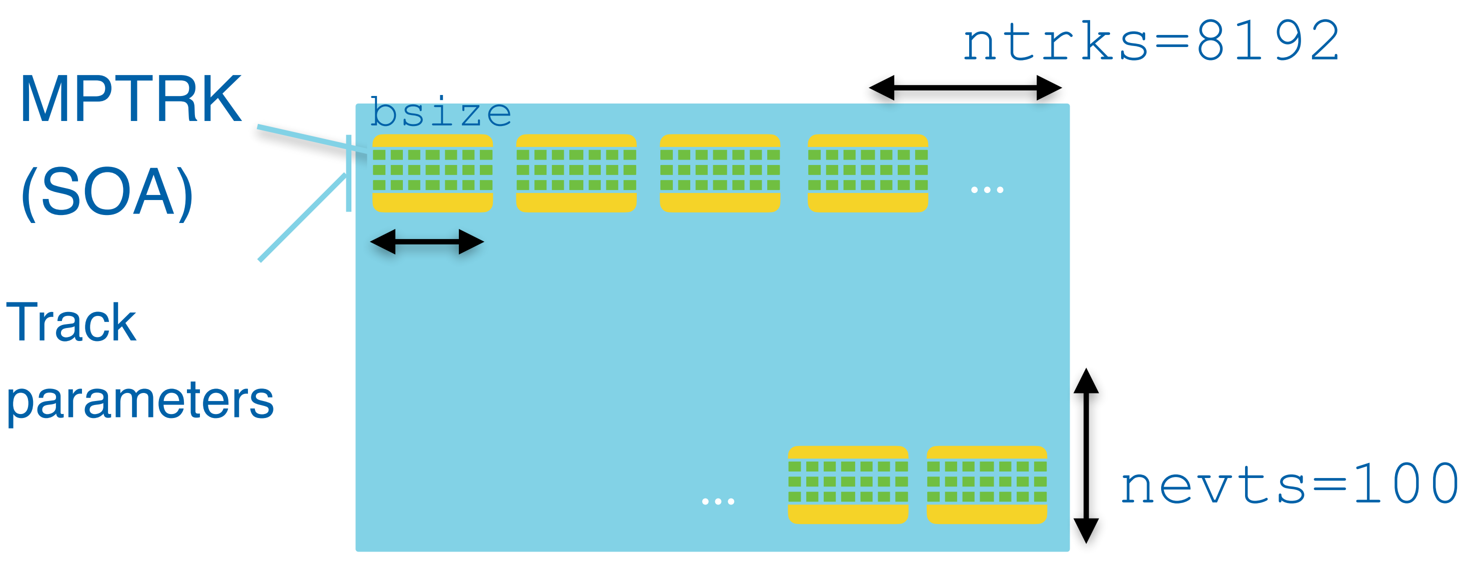




Fermilab

p2r / p2z program overview

- Simplified program workflow:
  - Fixed set of track parameters
  - Fixed number of events (`nevts`)
  - Fixed number of tracks in each event (`ntrks`)
  - Single GPU kernel:
    - Prepare data on CPU
    - Transfer to GPU compute
    - Transfer track data back to CPU
- p2r/p2z use Array-Of-Structure-Of-Array (AOSOA) as the main data structure
  - Total work of `ntrks` x `nevts`, tracks in an event are grouped into batch of `bsize`
  - Batch of tracks are put into the same data structure (MPTRK)



Fermilab

Overview of portability layers

- Explore different approches to portabilities:
  - Template Libraries : Alpaka, Kokkos
  - Compiler pragma-based approach: OpenMP, OpenACC
  - Language extension : SYCL, std::par
- Alpaka and Kokkos has different abstraction level
  - Alpaka is closer to CUDA-level
  - Kokkos aims to be more descriptive

Kokkos

```
template <int bSize, int layers, typename member_type>
KOKKOS_FUNCTION void launch_p2r_kernel(const member_type& teamMember){

     Kokkos::parallel_for(Kokkos::TeamThreadRange(teamMember,
                          teamMember.team_size()),[&] (const int& i_local){
     int i = teamMember.league_rank () * teamMember.team_size () + i_local;
     for(int layer = 0; layer < layers; ++layer) {
       //
      propagateToR<N>(…);
       KalmanUpdate<N>(…);
       //
  });
  return;
}

   Kokkos::parallel_for("Kernel",
       team_policy(team_policy_range,team_size,vector_size),
        KOKKOS_LAMBDA( const member_type &teamMember){
            launch_p2r_kernel<bsize, nlayer>()); // kernel for 1 track
        });
```

Alpaka

```
struct GPUsequenceKernel
{
public:
    template<typename TAcc>
    ALPAKA_FN_ACC auto operator()(
        TAcc const& acc,
        MPTRK* btracks_,
        MPHIT* bhits_,
        MPTRK* obtracks_
        ) const -> void
    {
       using Dim = alpaka::Dim<TAcc>;
       using Idx = alpaka::Idx<TAcc>;
       using Vec = alpaka::Vec<Dim, Idx>;

   for(int layer = 0; layer < nlayer; ++layer) {
      //
       propagateToR<N>(…);
       KalmanUpdate<N>(…);
       //
     }
};
//
   alpaka::enqueue(queue, taskKernel);
   alpaka::wait(queue);
```

Overview of portability layers

- SYCL is a *specification* of single-source C++ programming model for heterogeneous computing
  - "Native" support for Intel's hardware
  - Alpaka/Kokkos has/are developing a SYCL-backend to support intel GPUs
- Standard parallelization since C++17
  - Plain C++ code!
  - Limited to what the standard supports:
    No async operation, no launch parameters, need unified memory, etc
  - NVIDIA's advocated solution for portability:
    A closed source compiler(nvc++) for NVIDIA GPUs

SYCL

```
#include <CL/sycl.hpp>

auto p2r_kernels = [=,btracksPtr    = trcks.data(),
                     outtracksPtr  = outtrcks.data(),
                     bhitsPtr      = hits.data()] (sycl::id<1> i) {
                          propagateToR<N>(…);
                           KalmanUpdate<N(…);
                    };

cq.submit([&](sycl::handler &h){
   h.parallel_for(sycl::nd_range(global_range,local_range), p2r_kernels);
});
```

std::par

```
auto p2r_kernels = [=,btracksPtr    = trcks.data(),
                     outtracksPtr  = outtrcks.data(),
                     bhitsPtr      = hits.data()] (const auto i) {
                          propagateToR<N>(…);
                           KalmanUpdate<N(…);
                    };

std::for_each(policy,
              impl::counting_iterator(0),
              impl::counting_iterator(outer_loop_range),
              p2r_kernels);
```

Overview of portability layers

- Compiler directive approach: OpenMP, OpenACC
  - Explicitly tells compiler how to execute the loop
    - Easy to write simple off-loading code
    - Can get complicated

OpenMP

```
#pragma omp target update to(trk[], hit[])\
 nowait depend(out:trk[])

#pragma omp target teams distribute parallel \
for num_teams(...) num_threads(...) collapse(2)\
  map(to: trk[...], hit[], outtrk[])\
 nowait depend(in:trk[]) depend(out:outtrk[])

for (size_t ib=0;ib<nb;++ib) { // loop over blocks
    for (size_t tIdx=0;tIdx<bsize;++tIdx) { // loop over threads
        ...
        #pragma unroll
        for(size_t layer=0; layer<nlayer; ++layer) {
            ..
            propagatetoz(...);
            kalmanupdate(...);
        }
    }
}
```

OpenACC

```
#pragma acc parallel loop gang worker collapse(2) \
     default(present) num_workers(NUM_WORKERS)     \
     private(errorProp, temp, rotT00, rotT01
    for (size_t ie=0;ie<nevts;++ie) { // loop over events
      for (size_t ib=0;ib<nb;++ib) { // loop over tracks
         const MPTRK* btracks = bTk(trk, ie, ib);
         MPTRK* obtracks = bTk(outtrk, ie, ib);
         for(size_t layer=0; layer<nlayer; ++layer) {
           const MPHIT* bhits = bHit(hit, ie, ib, layer);
           propagateToR(…);
           KalmanUpdate(…);
         }
       }
     }
   }
```

Measurement

- *p2r* measurement done on Joint Laboratory for System Evaluation (JLSE)
  - HPC Testbed system hosted at Argonne National Lab
  - Does not include time for data-transfer (~3x kernel time on a A100 GPU)
- All versions compiled with the same p2r parameters
  - Perform computation on ~800k tracks, repeated 5 times

- *p2z* performs similar measurements on Summit GPU node
  - Includes data-transfer time
  - Explores different compiler/implementations

Data movement
(CPU to GPU)
~3x kernel time

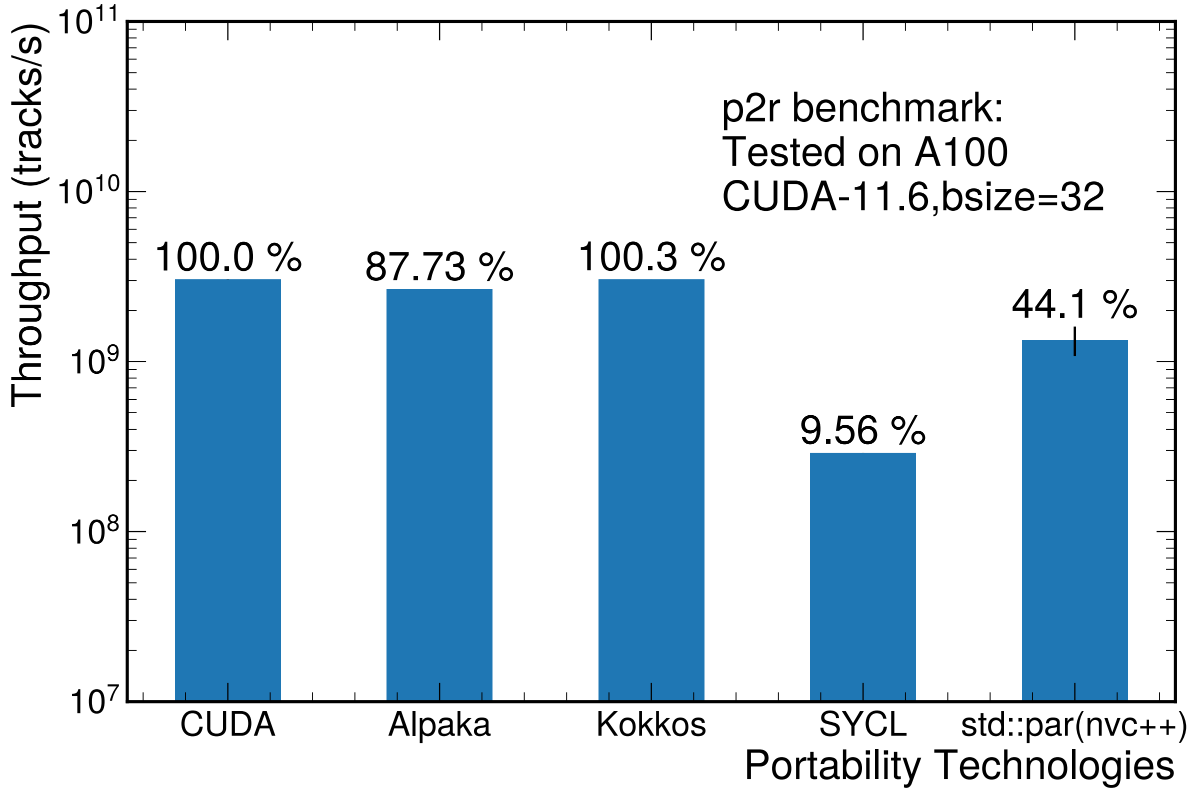
CUDA Kernel

Typical p2z/p2r GPU timeline w/ single stream

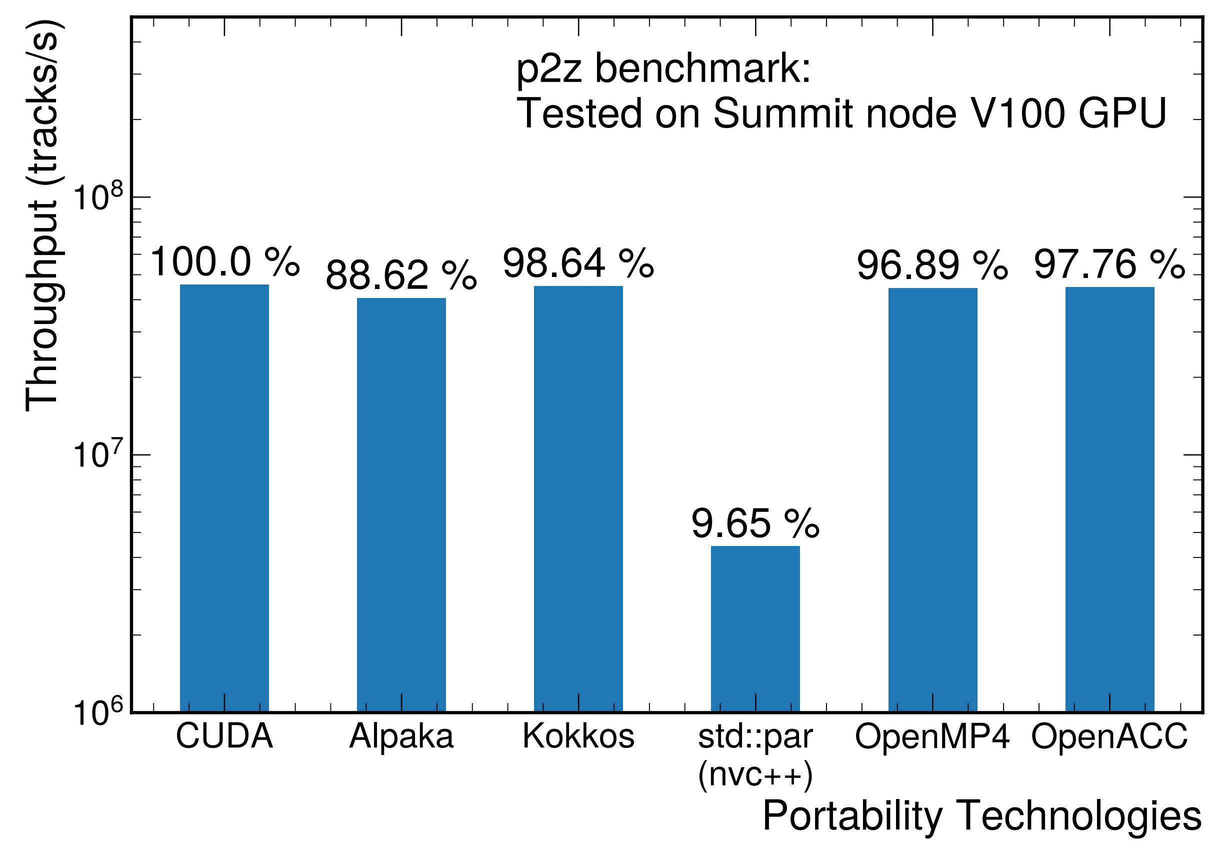
GPU Results - NVIDIA

- p2r's measurement more sensitive changes to kernel execution
  - p2z measurement is sensitive to overheads related to data movement
- Kokkos and Alpaka both managed to produce close-to-native performance
- Unclear what is causing the slowdown in SYCL/std::par in p2r versions
  - Profiling shows significant branching in SYCL version

p2r: NVIDIA GPU (A100)
Kernel-only

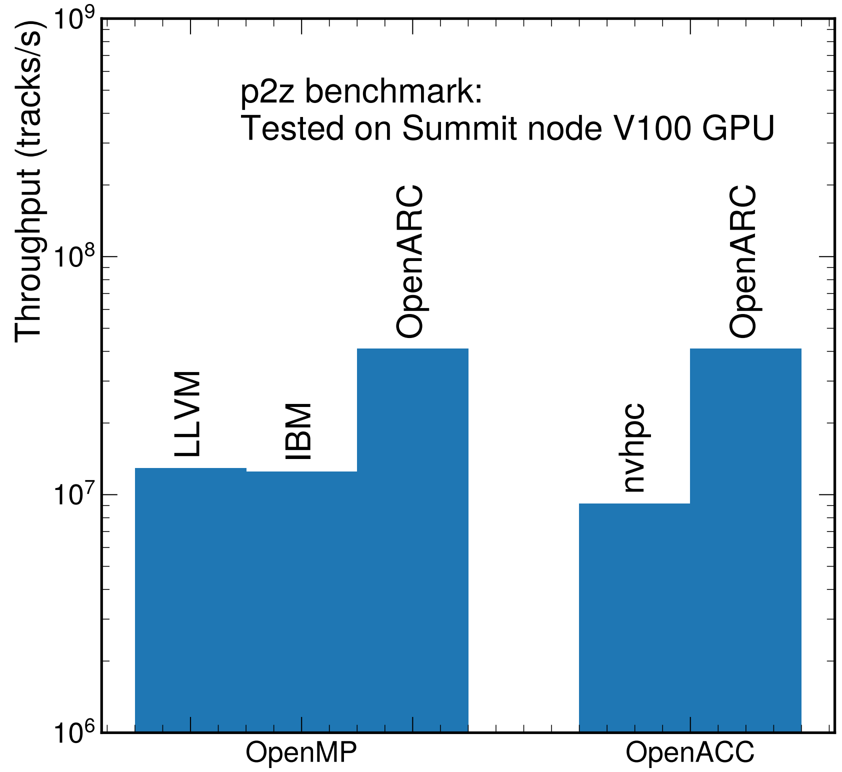


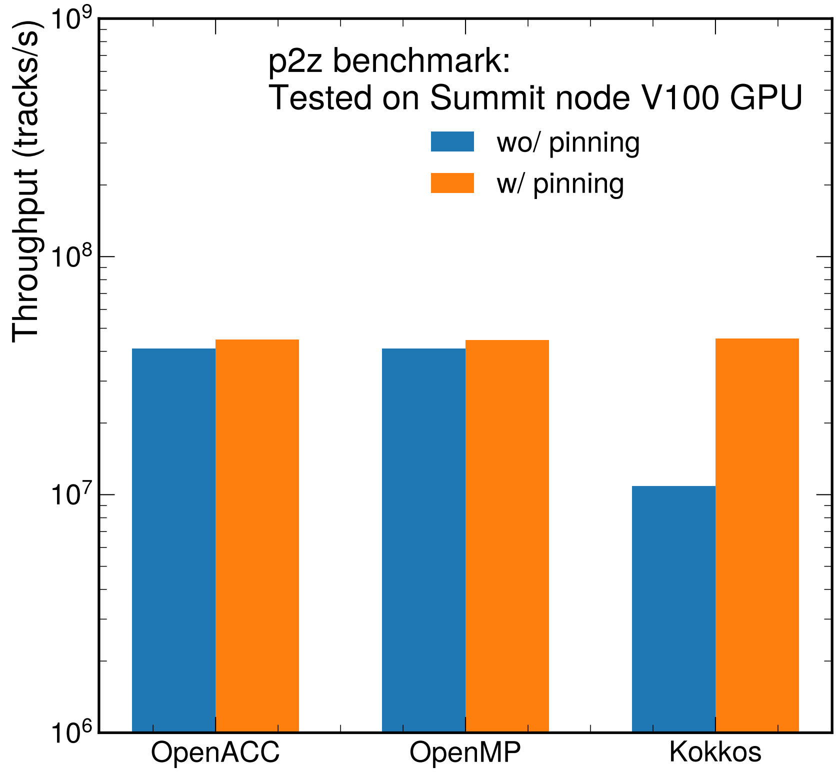
p2z: NVIDIA GPU (V100)
Data movement + kernel



GPU Results - NVIDIA

- Performance can varies a lot — due to varies issues
  - Compilers matters — especially for directive-based portability
  - Memory pinning
  - Data layout, temporary data placement (local memory/shared memory)





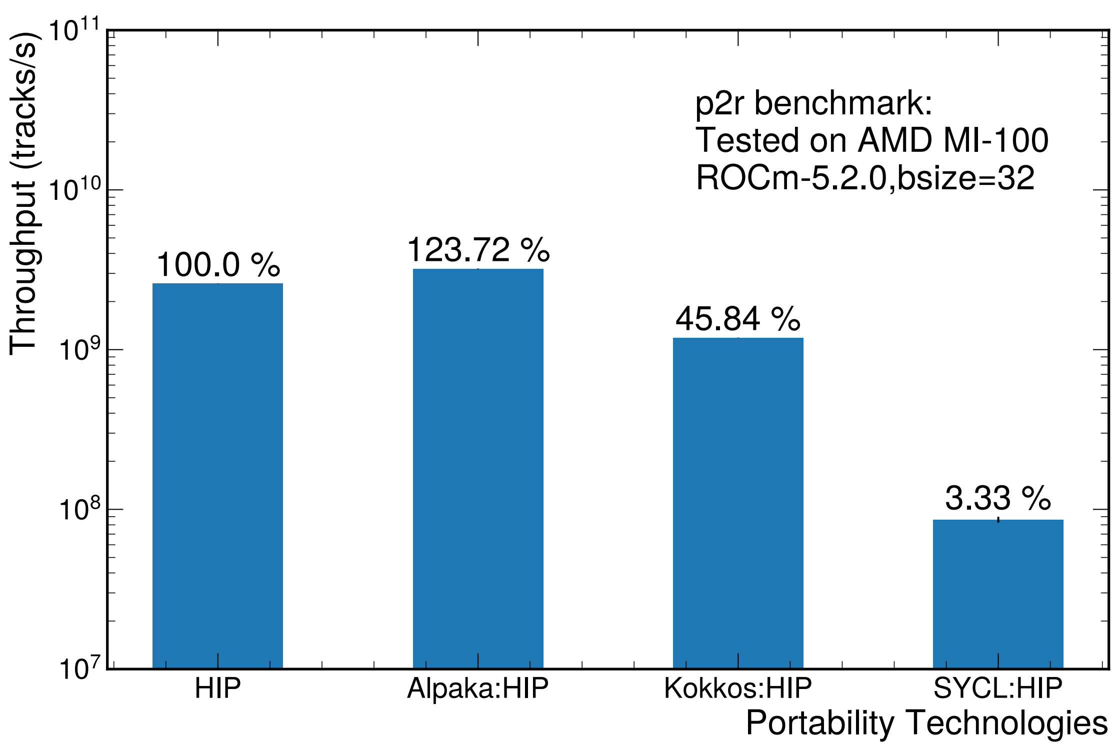
- Optimized performance is not easy to achieve
  - Even for a simple, single-kernel application like p2r/p2z
  - Iteration between profiling & implementation

Fermilab

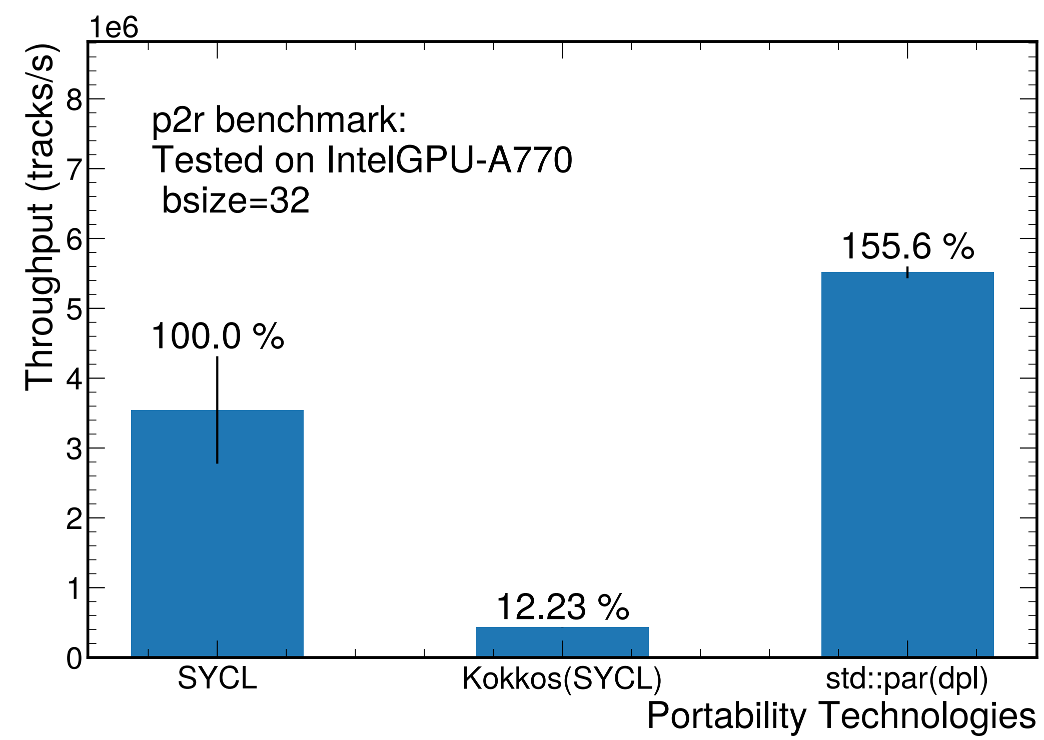
GPU Results - AMD/Intel

- Portability technologies are expanding towards AMD/intel GPU supports
  - Results are more preliminary
  - Switching backends for Alpaka and Kokkos are relatively seamless
- HIP backend:
  - Alpaka and Kokkos has reasonable performance
- Intel's A770 GPU do not support double-precision
  - Results obtained with DP emulation, could have significant performance impact
  - Plans to revisit Kokkos's result with the latest SYCL v4.0.1
  - Alpaka is working on a SYCL backend

p2r: AMD MI-100



p2r: Intel GPU A770*



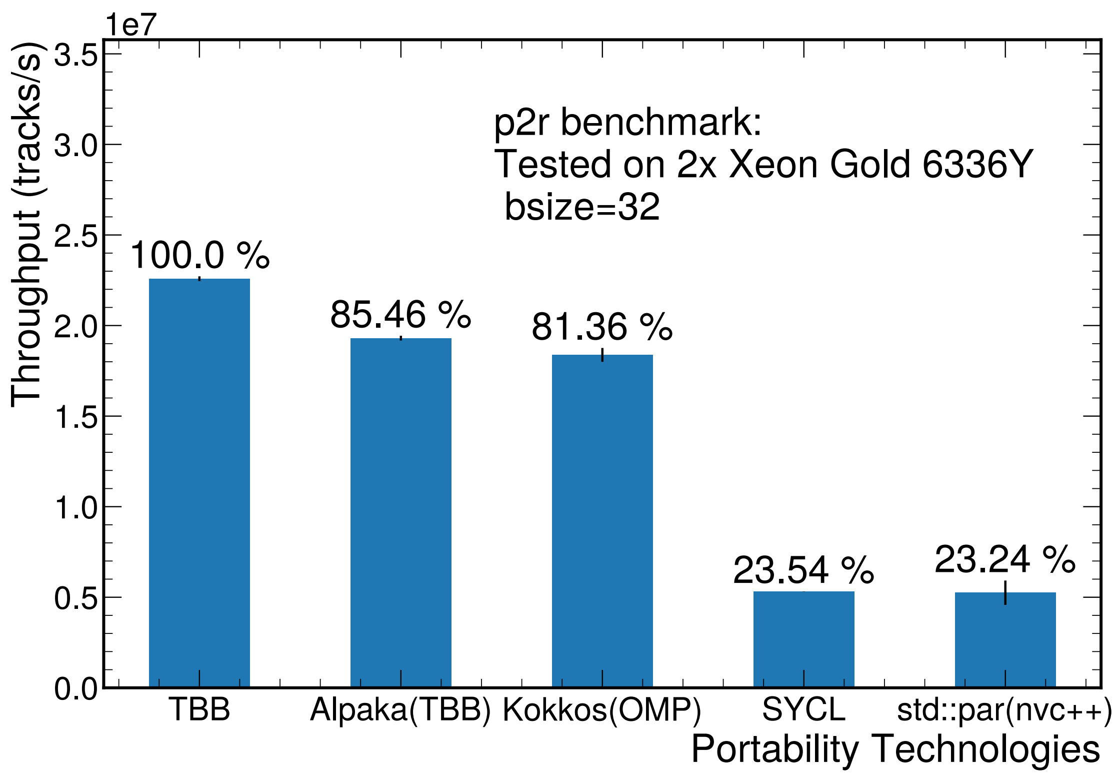
*not measured on JLSE

Fermilab

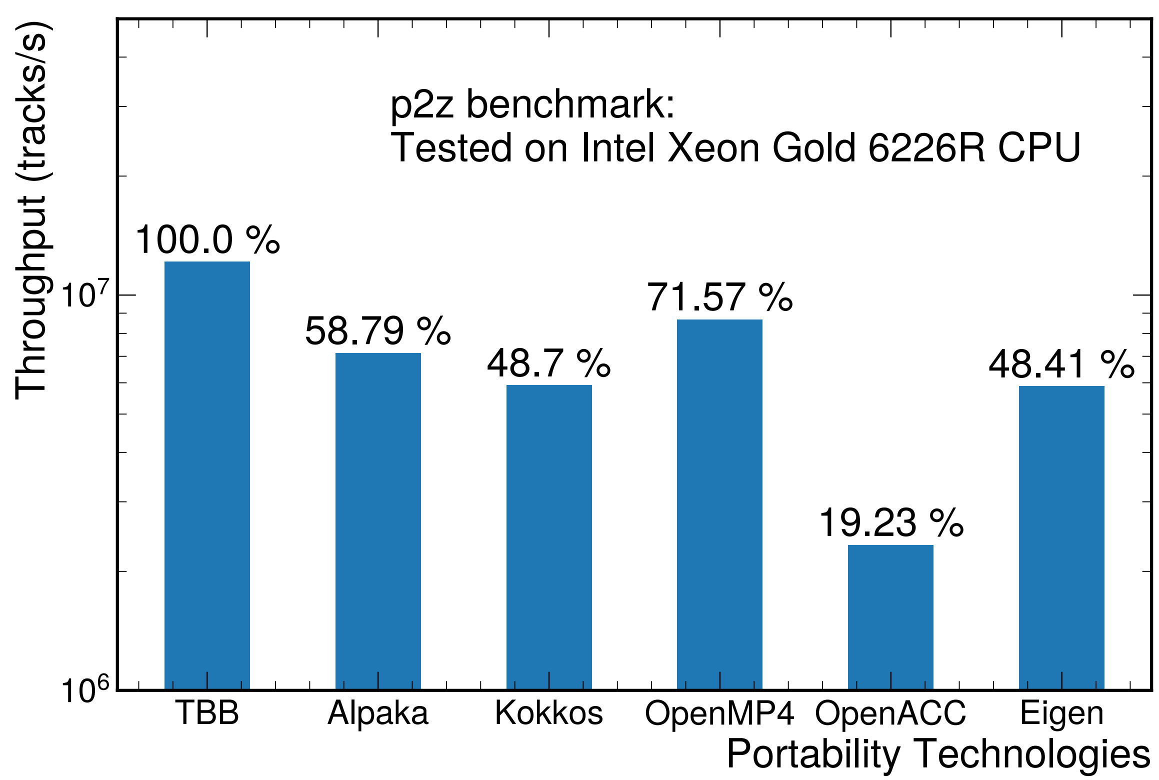
CPU results

- Native implementation done with TBB
  - Multi-threaded and vectorized
- Non-trivial to have efficient CPU & GPU performance with same code base
  - Data layout issue
  - Make sure loops are vectorized
- Portability layers can achieve ~50-80% native performance

p2r: Intel Xeon Gold 6336Y CPU



p2z: Intel Xeon Gold 6226R CPU



Summary and outlook

- Explored major portability solutions in a HEP-testbed application
  - Alpaka, Kokkos, SYCL, std::par, OpenMP

- Most solutions can give reasonable performance on NVIDIA GPUs
- Support for HIP/Intel GPUs are less mature

- Looking forward:
  - Summarize the porting experience towards HEP-CCE final report
  - “Best” solution will probably depend on application/situation

Acknowledgement:
We thank the Joint Laboratory for System Evaluation (JLSE)
for providing the resources for the performance measurements
used in this work.

Back up

Software versions used in p2r results

	TBB	CUDA	HIP	Alpaka (v 0.9.0)	Kokkos (3.6.1)	SYCL	std::par
NVIDIA GPU	-	cuda/11.6.2	cuda/11.6.2 rocm/5.2.0	cuda/11.6.2 gcc/9.2.0 nvcc	cuda/11.6.2 gcc/8.2.0 nvcc	cuda/11.6.2 intel/llvm-sycl [1]	dpcpp/ 2022.1.0
AMD GPU	-	N/A	rocm/5.2.0	rocm/5.1.3 gcc/9.2.0 hipcc	rocm/5.1.3 gcc/9.2.0 hipcc	cuda/11.6.2 intel/llvm-sycl [1]	N/A
Intel GPU	-	N/A	N/A	N/A	Kokkos/4.0 dpcpp/ 2023.0.0	dpcpp/2023.0.0	dpcpp/ 2023.0.0 dpl/2022.0.0
CPU(x86)	gcc/XXX	N/A	N/A	[TBB]	gcc/11.1.0	dpcpp/2023.0.0	nvc++/22.7

[1] intel/llvm sycl branch commit 70c2dc6