Evaluating Performance Portability with the CMS Heterogeneous Pixel Reconstruction code

N. Andriotis[1], A. Bocci[2], E. Cano[2], L. Cappelli[3], M. Dewing[4], T. Di Pilato[5,6], J. Esseiva[7], L. Ferragina[8], G. Hugo[2],
M. Kortelainen[9], M. Kwok[9], J. J. Olivera Loyola[10], F. Pantaleo[2], A. Perego[11], W. Redjeb[2,12]
[1]BSC [2]CERN [3]INFN Bologna [4]ANL [5]CASUS [6]University of Geneva [7]LBNL [8]University of Bologna
[9]FNAL [10]ITESM [11]University of Milano Bicocca [12] RWTH

CHEP 2023 11 May 2021

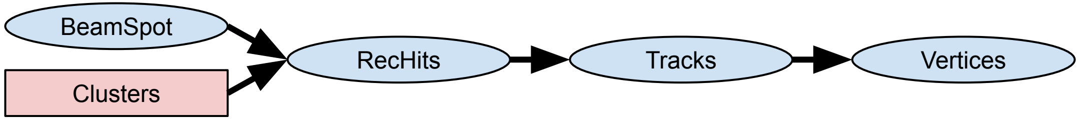
HEP-CCE

Introduction

- CMS uses GPUs as part of the High-Level Trigger farm in LHC Run 3
- GPU vendors provide their own APIs that also differ from programming the CPU
  - Want to minimize development and maintenance effort
  - CMS is moving to achieve portable code via Alpaka (A. Bocci Tuesday Track 2)
    - Want to be aware of the other technologies in the market to guide long term planning
- Used CMS heterogeneous pixel reconstruction (Patatrack) as a use case for a set of realistic algorithms utilizing GPU effectively
- Measure the performance of direct, Alpaka, Kokkos, and SYCL versions on CPU, NVIDIA GPU, and AMD GPU
  - All versions give the same results (within reproducibility accuracy)
  - Some grain of salt needed to interpret the results
    - The versions using different portability technologies have differences
- Report initial experience with `std::par` and OpenMP Target offload

CMS Heterogeneous Pixel Reconstruction

- About 40 kernels organized in 5 “framework modules”
  - arXiv:2008.13461
  - Kernels are short: few μs to ~1 ms, performance sensitive to overheads



- Raw pixel detector data (~250 kB/event) transferred to the GPU
- Only final results transferred back to the CPU: ~4 MB for tracks, ~90 kB for vertices
  - Not considered in throughput measurements in this talk
- Extracted into a standalone program to enable rapid prototyping
  - Flexible GNU Make -based build system
  - Simple framework mimicking CMSSW’s use of oneTBB tasks
  - Disk I/O contribution to time measurements is ignored
    - 1000 events from TTbar + pileup 50 simulation from CMS Open Data read at the beginning of the job and recycled

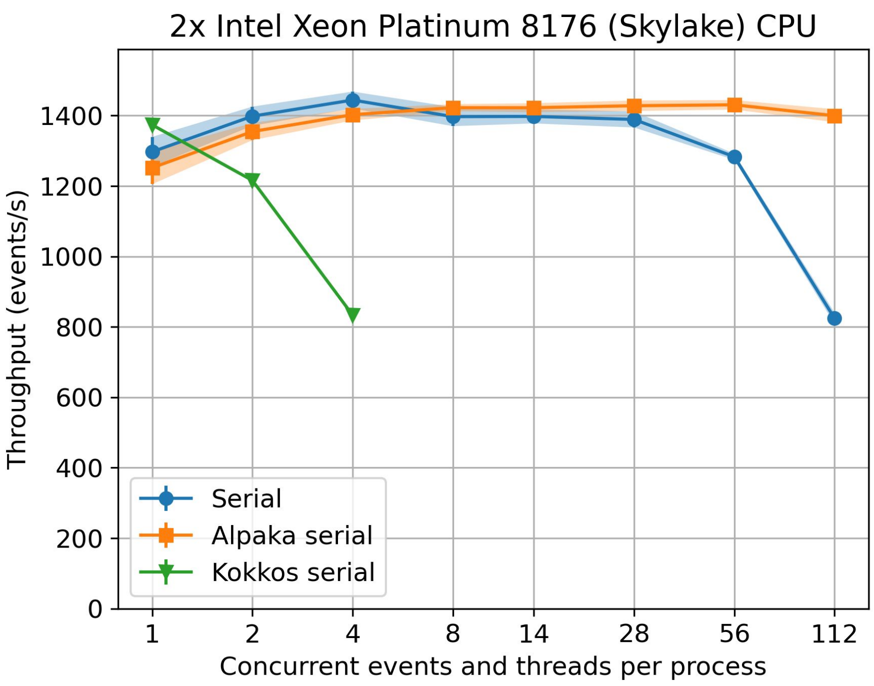
Alpaka and Kokkos versions are most mature

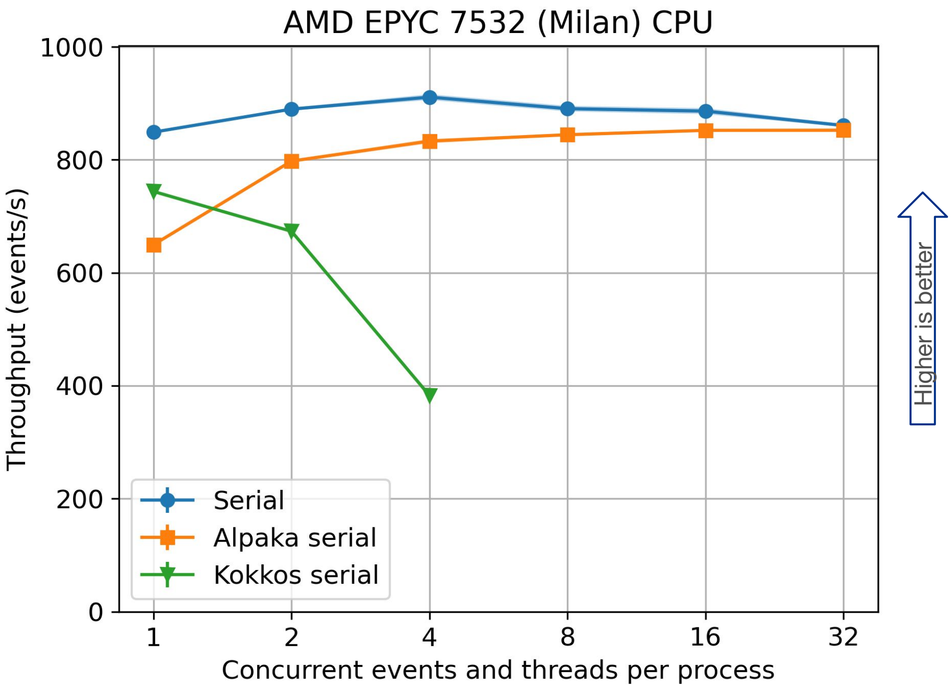
- Alpaka (earlier reported in ACAT 21: J. Phys. Conf. Ser. **2438** 012058)
  - Thin, header-only, templated C++ library, abstraction level similar to CUDA
    - Backends include serial, OpenMP 2, std::thread, TBB, CUDA, HIP, SYCL (experimental)
  - Flexible to work with
    - E.g. can build a single application that supports multiple GPU backends
  - Somewhat more verbose syntax compared to others
- Kokkos (earlier reported in vCHEP 21: EPJ Web. Conf. **251** 03034)
  - Templated C++ library, higher abstraction level than CUDA
    - Backends include serial, OpenMP, CUDA, HIP, HPX, OpenMP-Target, SYCL (experimental)
  - Provides parallel algorithms such as prefix scan, reduction, sorting
  - Provides multidimensional array with customizable layout (precursor to `std::mdspan`)
  - Constraints how to build the application code
  - Have had to understand what Kokkos does between developer and vendor API

Performance measurements

- Performance measurements done using the resources of the Joint Laboratory for System Evaluation at Argonne National Laboratory
- CPUs:
  - 2-socket Intel Xeon Platinum 8176 (Skylake): 28 cores and 56 threads x 2
  - 1-socket AMD EPYC 7532 (Milan): 32 cores and 32 threads
  - Measure total throughput of full node
    - N processes of M threads such that NxM = number of HW threads
- GPUs:
  - NVIDIA: **A100** (19.5 FP32 TFLOPS) and **A40** (37.4 FP32 TFLOPS)
  - AMD: **MI100** (32.1 FP32 TFLOPS) and **MI250** (90.5 FP32 TFLOPS)
  - Measure the throughput on a single GPU by increasing the number of concurrent events
  - Node has no other activity
- Take average of 4 executions

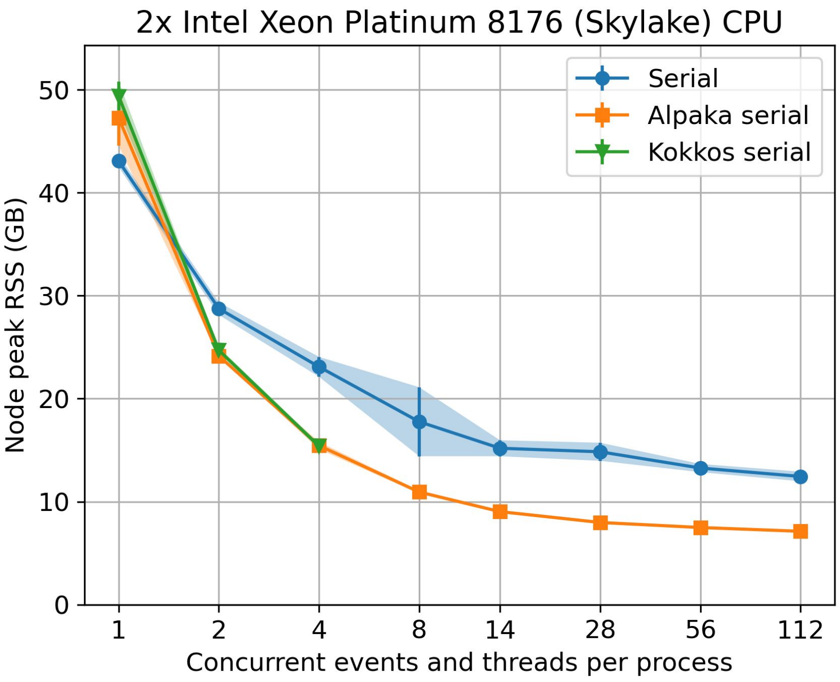
Event processing throughput on CPU "serial backends"

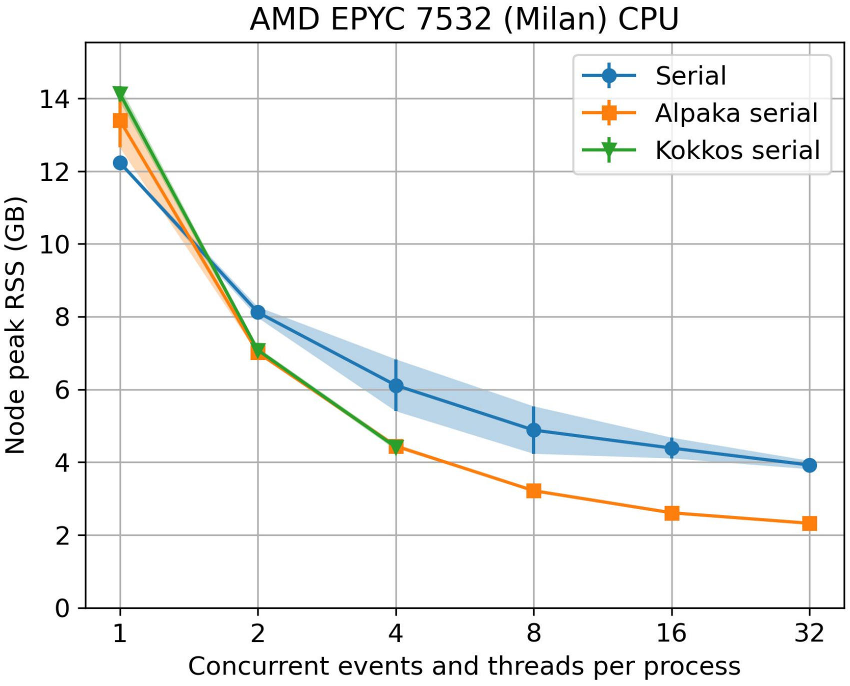




"Serial" = replica of the backend for each concurrent event, no parallelism within each algorithm

Peak node memory usage on CPU “serial backends”

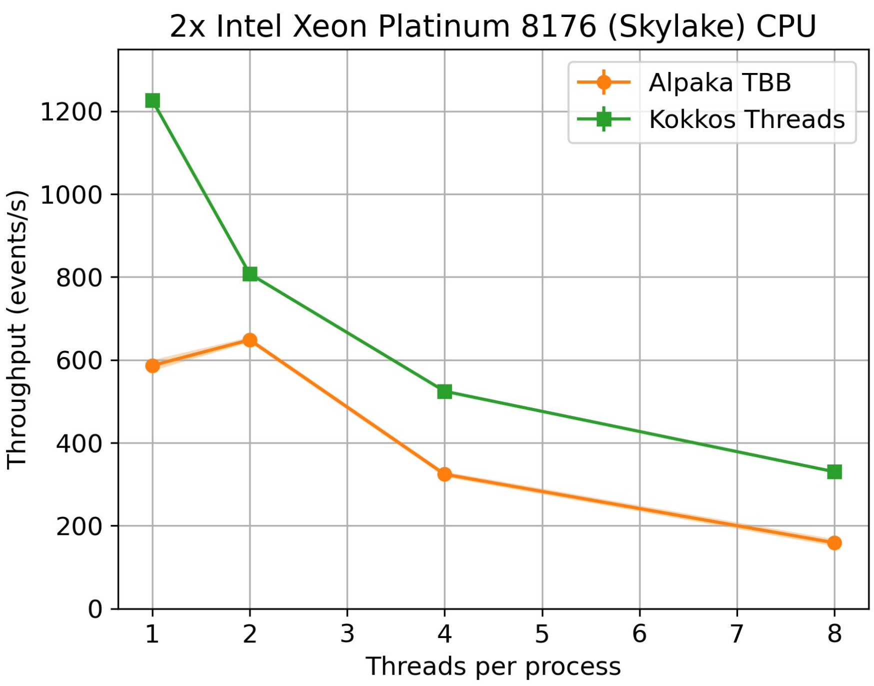


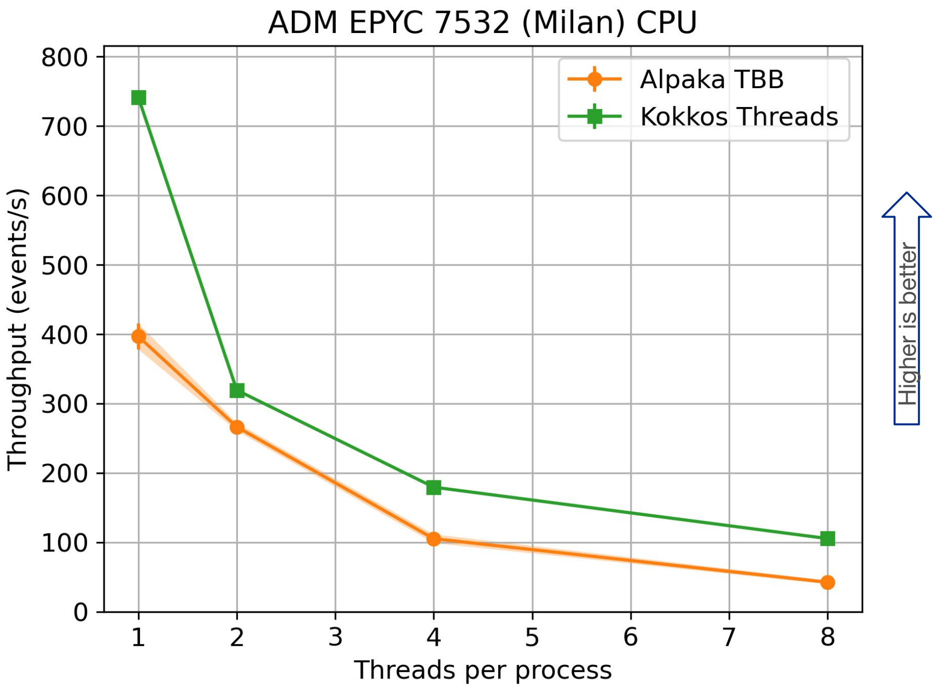


Lower is better

HEP-CCE

Event processing throughput on CPU “parallel backends”

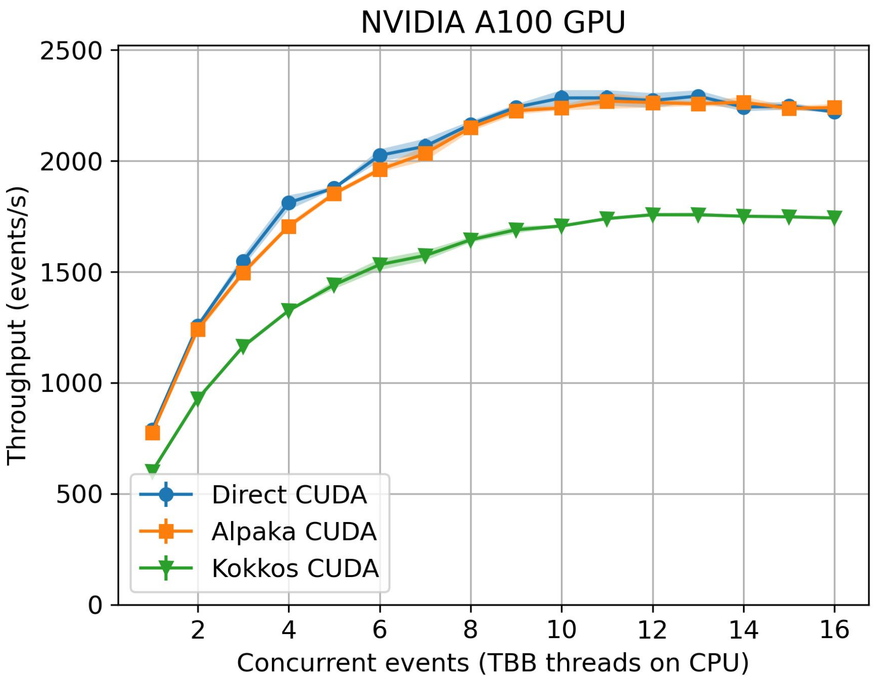


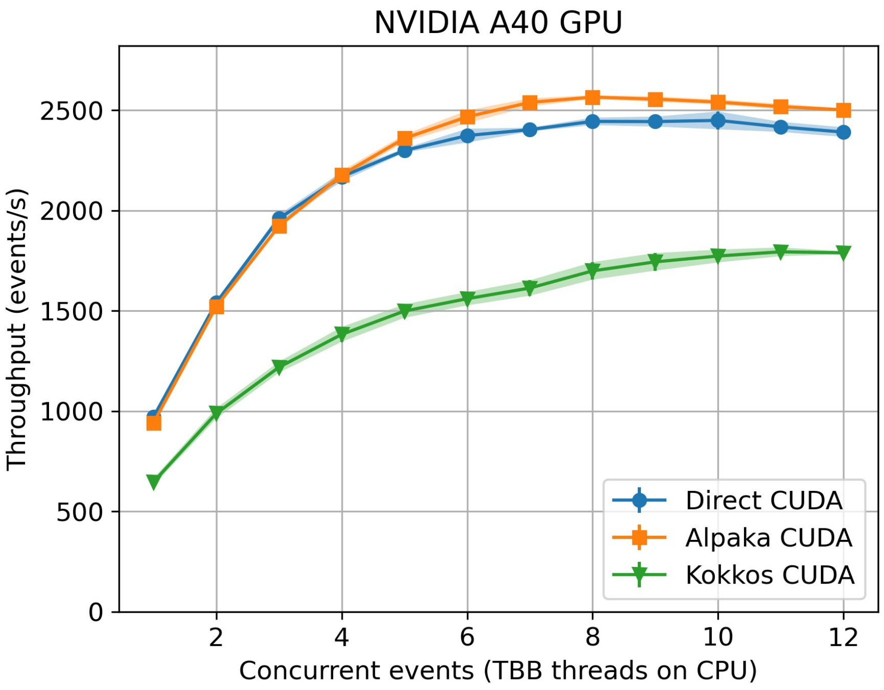


One event in flight → concurrent event processing is more useful than intra-algorithm parallelism in this case

HEP-CCE

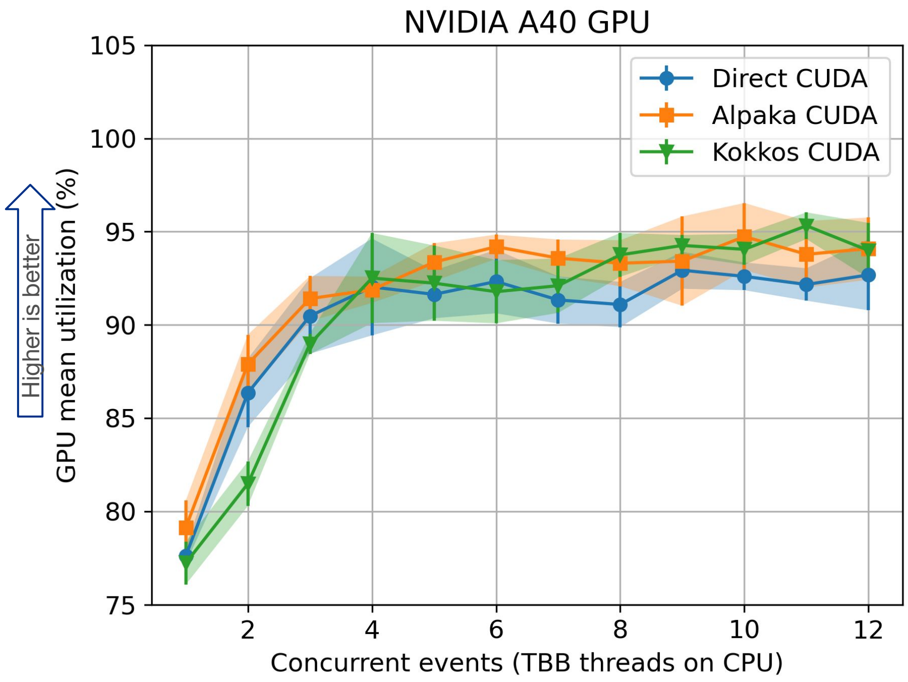
Event processing throughput on NVIDIA GPU



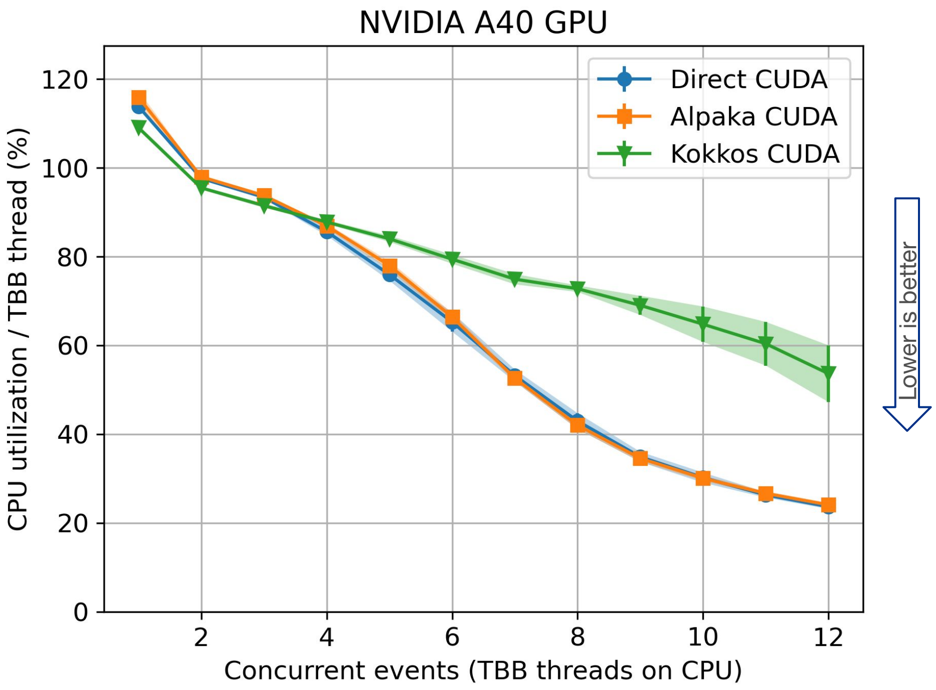


Higher is better

Mean GPU and CPU utilization on NVIDIA A40 GPU

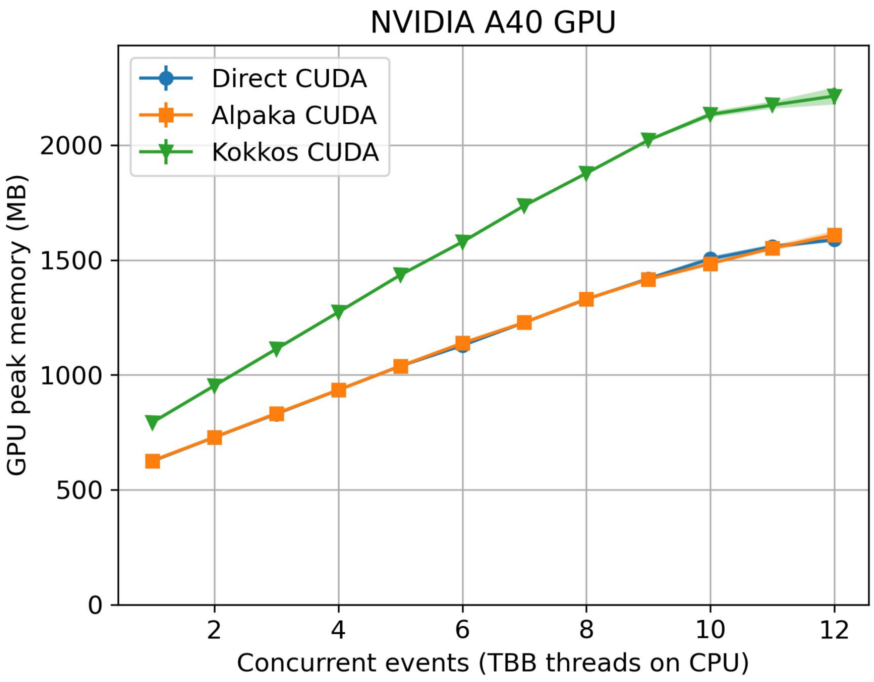


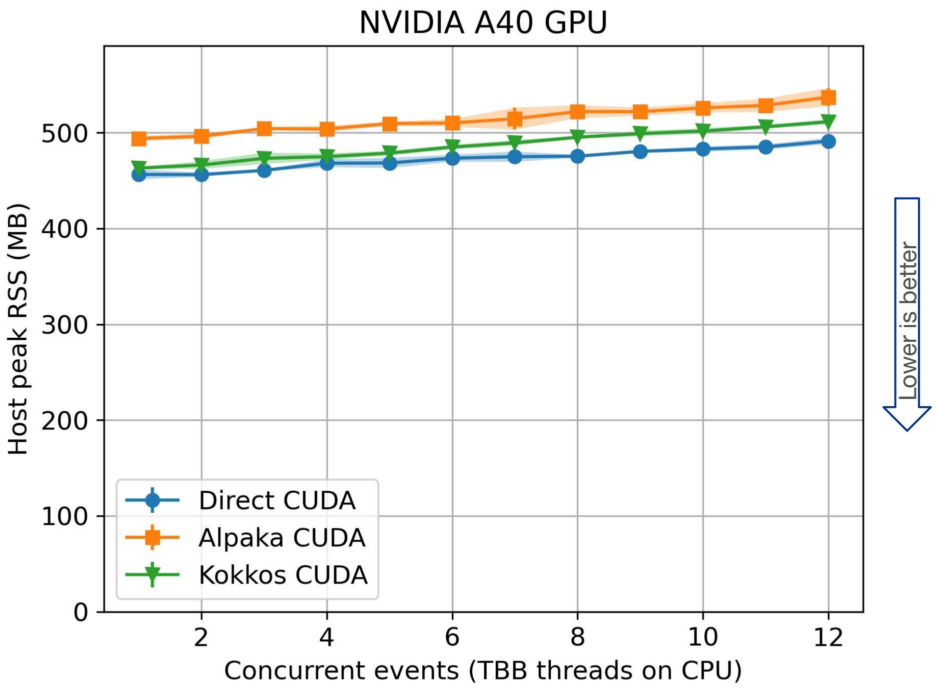
As reported by `nvidia-smi`, mean from samples



CPU efficiency normalized by number of threads

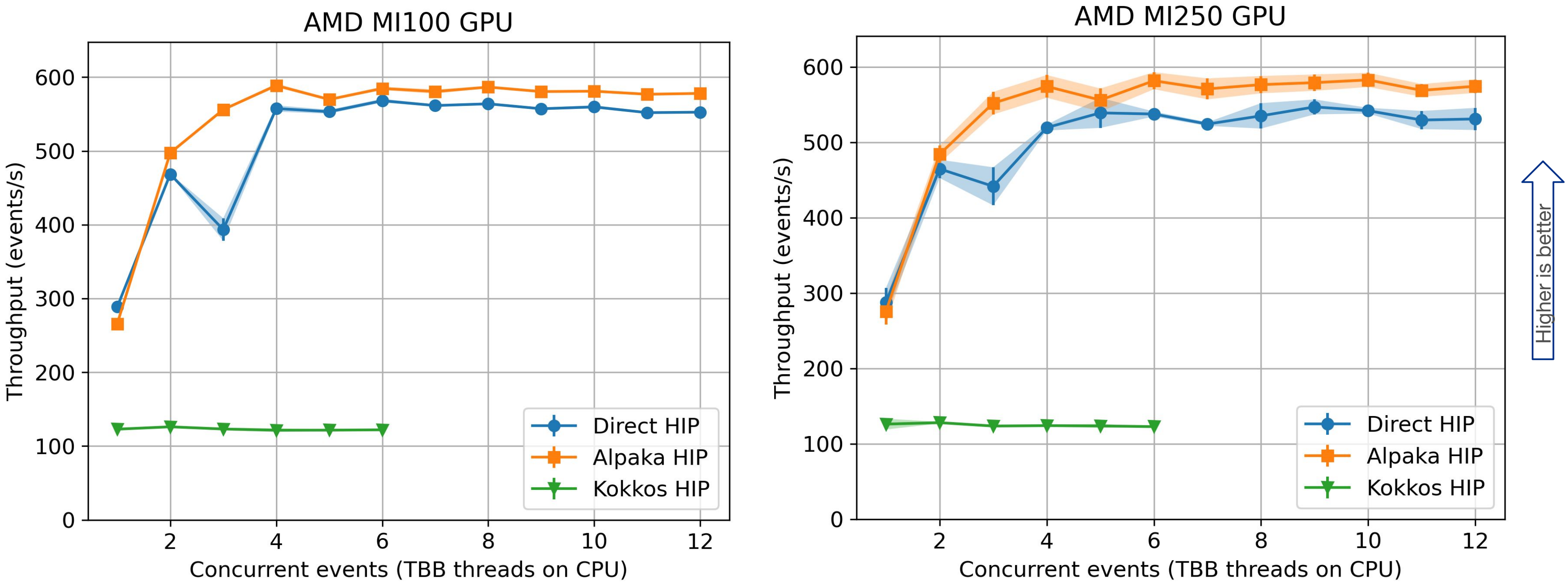
Peak memory usage on NVIDIA A40 GPU





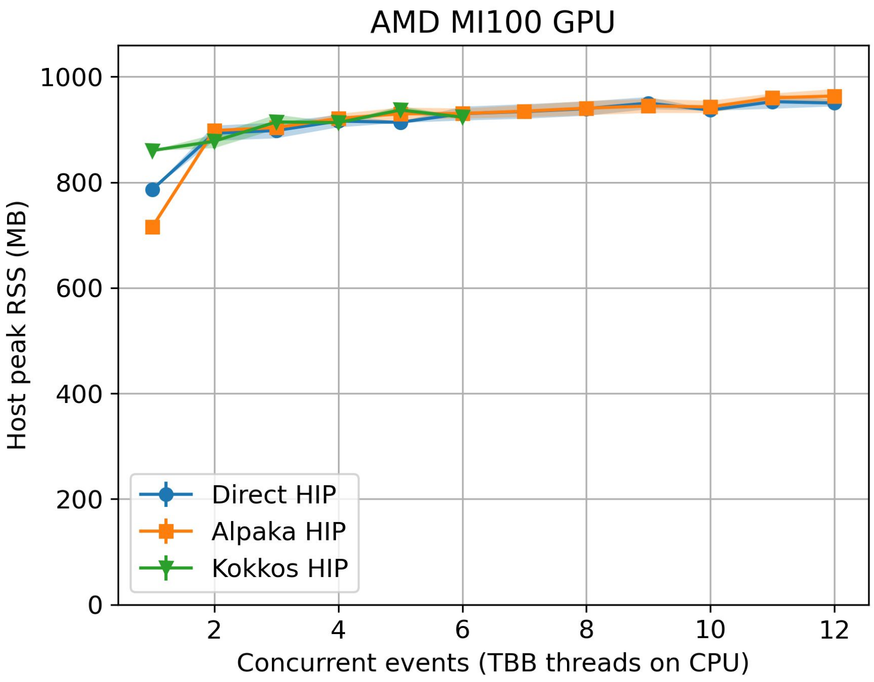
As reported by `nvidia-smi` and `/proc/<PID>/status`. A100 shows similar behavior.

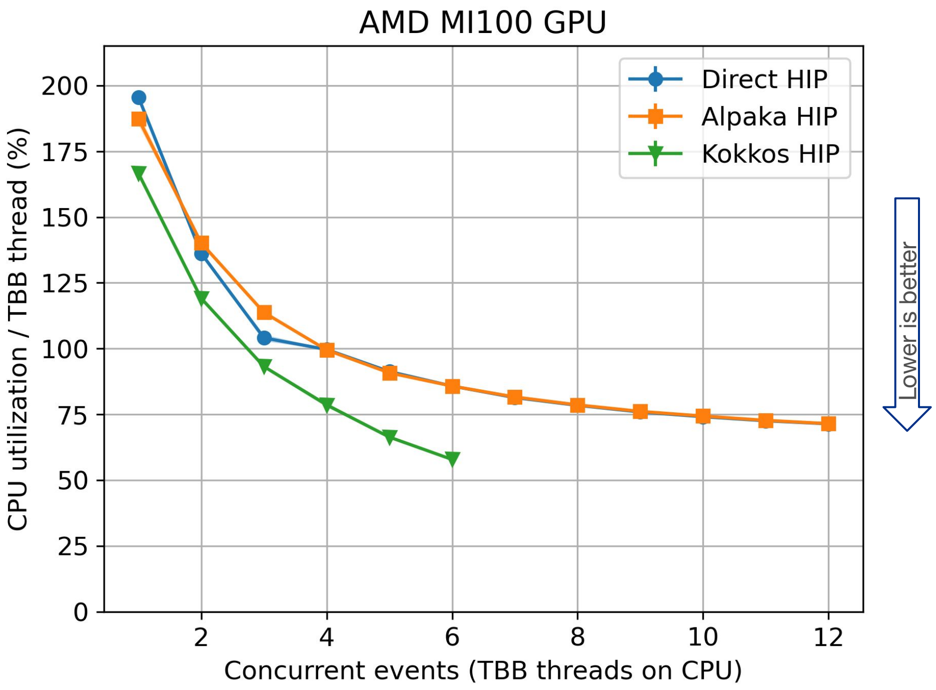
Event processing throughput on AMD GPUs



HEP-CCE

Host memory and CPU utilization on AMD MI100 GPU

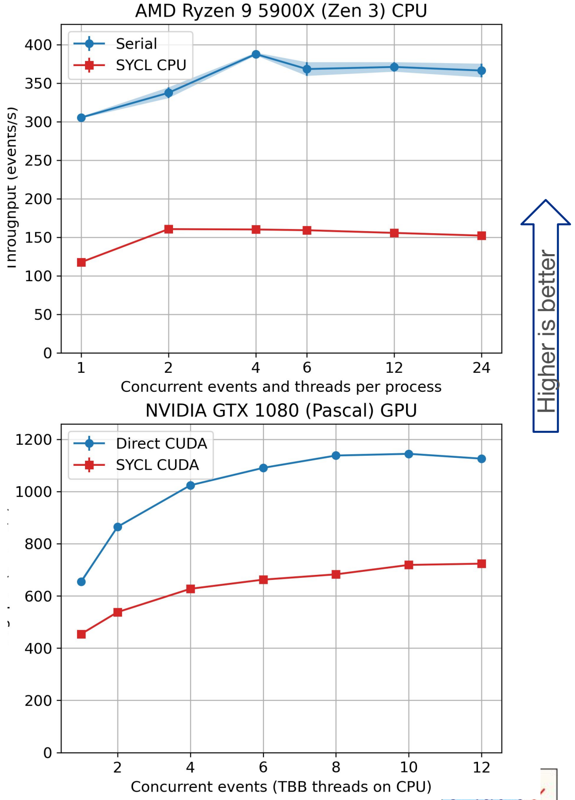




MI250 shows similar behavior

SYCL version: complete and runs on some hardware

- SYCL: Specification by the Khronos Group
  - Some notable implementations:
    - Intel's oneAPI DPC++ and open-source LLVM
    - Open SYCL (not tested)
  - Allows simultaneous use of multiple backends
- Development of SYCL version revealed many bugs in the Intel LLVM
  - E.g. collective operations on CPU, block shared variables
- Was not able to replicate the setup that would result in a working executable on other machines with e.g. A100
  - Also did not succeed to compile for AMD GPUs
- Some kernels are slower than in CUDA, every operation creates a SYCL event, SYCL events can not be reused



`std::par` version: technically complete

- STL parallel algorithms as implemented by NVIDIA in their HPC SDK
  - Relies on unified memory
- `std::par` version is complete, but testing is difficult because of compiler bugs
- Abstraction level much higher than Alpaka/Kokkos/SYCL
  - Low barrier for using GPUs in a new codebase
  - Converting a large and optimized CUDA application is easier to map to Alpaka/Kokkos/SYCL
    - `std::par` requires some algorithmic changes and/or more kernels
    - Hierarchical parallelism, e.g. synchronizing threads of a block, not supported
      - Have to split or rework such kernels
    - No access to CUDA shared memory, need to use global memory and use atomics
- Must compile the whole program with `nvc++` when offloading for NVIDIA GPU
  - To avoid One Definition Rule violations with e.g. `std::vector`

OpenMP Target offload: in progress

- Compiler pragma-based approach, popular for multithreading e.g. in HPC
- Can use `#omp target offload` in conjunction with multithreading with oneTBB
- Porting done with LLVM (15, 16, `main`), targeting NVIDIA and AMD GPU backends
  - Some of the encountered problems were fixed very quickly
- Tested also with other compilers (experienced many problems)
  - NVIDIA HPC SDK: compiles, fails at run time
  - AMD (AOMP, AFAR; `amdclang` underneath): compiler crashes
  - Intel oneAPI (`icpx`): compiles, but not pursued further yet
- Preliminary look on performance of some individual kernels with Nsight Systems
  - OpenMP kernels are slower than corresponding CUDA kernels
  - Much more data movement in OpenMP version compared to direct CUDA version

Conclusions

- We have compared the performance of various versions of CMS Heterogeneous Pixel Reconstruction
  - Direct, Alpaka, Kokkos, SYCL on x86 CPU, NVIDIA GPU, and AMD GPU
- Overall the best performance was achieved with Alpaka
- For this use case, Alpaka was also the easiest to work with
  - Flexible, little constraints added on top of the vendor APIs
- Kokkos: no concurrent instances of Serial backend (yet), often need to understand what Kokkos does in between developer and vendor API
- SYCL: compilation problems, overheads
- `std::par`: compilation problems, crashes, leads to many more kernels
- OpenMP Target offload: compilation problems, data movement is a concern

Fermilab

Related contributions

- M. Kortelainen: "Performance of Heterogeneous Algorithm Scheduling in CMSSW", Track X Tuesday 15:15
- A. Bocci: "Adoption of the alpaka performance portability library in the CMS software", Track 2 Tuesday 17:00
- Other portability studies from HEP-CCE
  - M. Kwok: "Application of performance portability solutions for GPUs and many-core CPUs to track reconstruction kernels", Track X Monday 11:00
  - M. Atif: "Porting ATLAS FastCaloSim to GPUs with OpenMP Target Offloading", Tuesday poster session
  - V. Tsulaia: "Porting ATLAS FastCaloSim to GPUs with std::par and with Alpaka", Tuesday poster session
  - C. Leggett "Porting ATLAS FastCaloSim to GPUs with Performance Portable Programming Models", Track X Tuesday 15:00
  - C. Leggett "Results from HEP-CCE", Track X Tuesday 11:00

Spares

Software versions

	Direct	Alpaka b518e8c9	Kokkos 3.5 or 4.0	SYCL Intel LLVM tag 2022-09 (0f579ba)
x86 CPU	GCC 11.1	GCC 11.1	GCC 11.1 Kokkos 3.5	GCC 8.5
NVIDIA GPU	GCC 11.1 CUDA 11.6.2	GCC 11.1 CUDA 11.6.2	GCC 11.1 CUDA 11.6.2 Kokkos 3.5	GCC 8.5 CUDA 11.8
AMD GPU	GCC 12.2 ROCm 5.4	GCC 12.2 ROCm 5.4	GCC 12.2 ROCm 5.4 Kokkos 4.0	