Status of the Patatrack use case

Mark Dewing (ANL), Julien Esseiva (LBNL), **Matti Kortelainen** (FNAL)

HEP-CCE All Hands Meeting

11 October 2022

Introduction

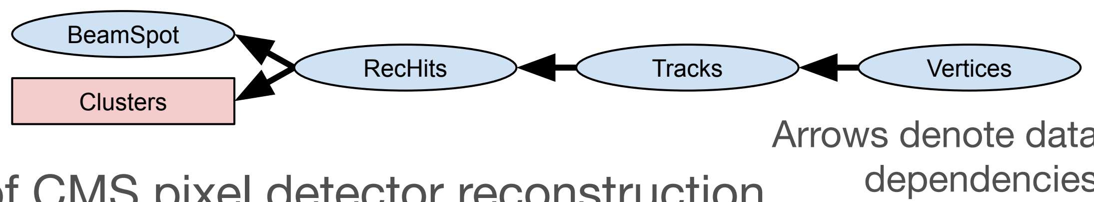
- Patatrack
  - A frozen, standalone version of CMS heterogeneous pixel track and vertex reconstruction
    - “End-to-end”, with mock framework and build system
  - Current status

	Implementations							
	CPU Serial	CUDA (original)	HIP	Kokkos	Alpaka (by CERN team)	std::par	SYCL (by CERN team)	OpenMP
NVIDIA			Eigen does not support					
AMD				Crashes randomly				
Intel				Does not compile (Eigen)				
CPU				Serial, POSIX threads	Serial, TBB			

Completed
In progress
Not started

Fermilab

Introduction (reminder)



- Origin in CUDA implementation of CMS pixel detector reconstruction from raw data up to tracks and vertices (arXiv:2008.13461)
  - Extracted into standalone repository from CMSSW https://github.com/cms-patatrack/pixeltrack-standalone/
    - Mimics most important aspects of CMSSW framework and build system
- Code consists of 39 CUDA kernels organized in 5 “algorithm modules” scheduled by the mock framework (about ~14kSLOC)
  - Not all kernels are perfect fit for GPU, strategy is to maximize the code run on GPU and minimize overheads (e.g. data transfers)
- Raw pixel detector data (~250 kB/event) transferred to the GPU
  - Optionally transfer tracks (~4 MB/event) and vertices (~90 kB/event) back to host
- Figure of merit is event processing throughput
  - Disk I/O contribution is ignored

Kokkos version

T. Childers, M. Kortelainen, M. Kwok, A. Strelchenko, Y. Wang

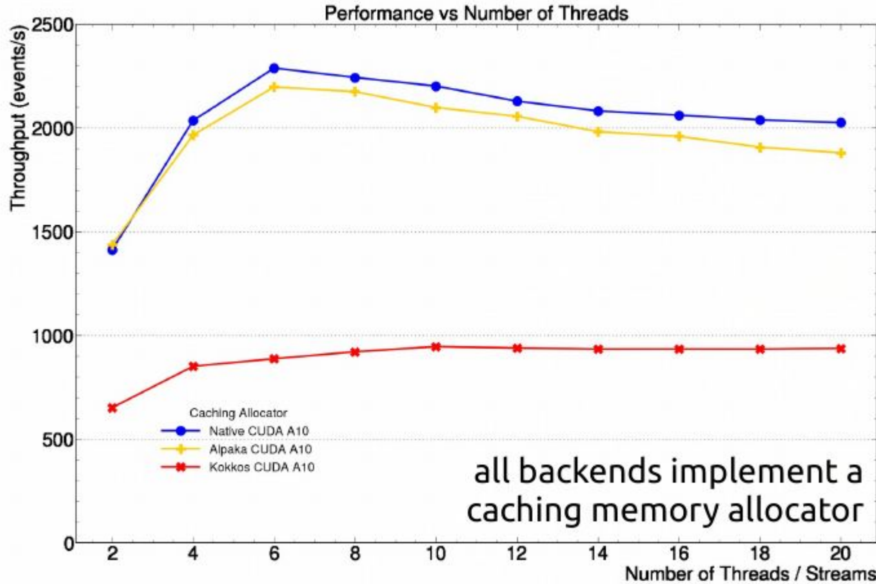
- Kokkos was the first portability layer the code was ported to
  - Conversion done by migrating CUDA code piece by piece
- First versions were very slow, performance has been improved greatly since then
  - Hackathon last year with Kokkos developers was very useful
- Current version considered complete
  - Supports “advanced GPU optimizations”: asynchronous execution, multiple CUDA streams, caching allocator
  - Only test Kokkos version updates and their impact on performance
- Experience and preliminary results documented in vCHEP21 (arXiv:2104.06573)
- Study was part of CMS’ decision process to choose between Kokkos, Alpaka, and in-house “solution”
  - Main reasons to reject
    - Host-serial backend serializes all execution from concurrent calls outside of Kokkos
    - On GPU slower than competition

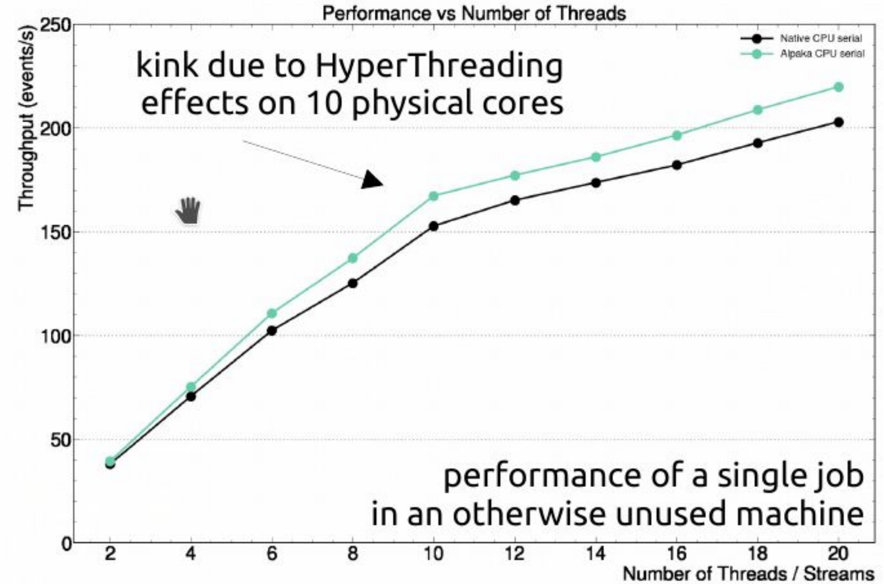
Alpaka version

M. Kortelainen, CERN team

- Alpaka version was developed mainly by CERN team
  - Matti got the technical part started and reviewed code
  - Conversion done by converting CUDA code piece by piece
- Current version considered complete
  - At least what concerns performance
  - Some prototyping supporting the Alpaka integration within CMSSW might still come
- Some highlights
  - Supports asynchronous execution, multiple CUDA streams and caching allocator
  - Can support both CUDA and HIP backends in the same build, and can run code of both backends in the same process on a machine that has both NVIDIA and AMD GPU
    - Likely not very useful for production, but demonstrates flexibility

Kokkos vs. Alpaka vs. native on GPU and CPU





- From 3/9/2022 Compute Accelerator Forum

SYCL version

- I’ve been told the porting is mostly done by the CERN team, but I have not seen any code yet
  - Can’t wait to review ~14kSLOC Pull Request…
  - Also interesting to learn how they dealt with Eigen

Fermilab

std::par version

J. Esseiva

- Different porting approach
  - Start from CUDA Unified Memory version, convert code piece by piece to std::par
  - Use NVIDIA's std::par implementation (nvc++)
- 3 modules left to port to std::par
  - RecHits, Tracks, Vertices
  - Data structures, memory management, framework architecture, tests have been ported
- nvc++ still very new, encountering a lot of bugs
  - Link error with Eigen when used in device context
  - Compiling with optimizations turned on fail for some CUDA kernels
  - Atomic operations (e.g. min), implementing it in terms of CAS operation doesn't work in device contexts
- Expect first completed version to be slow
  - More kernels, unified memory, more memory traffic, no asynchronous execution, …

OpenMP Target status

M. Dewing

- Different porting approach
  - Start from serial version, decorate code piece-by-piece with OpenMP pragmas
- So far ported various unit tests and clusterizer. Currently working on vertex fits.
- Encountered various problems
  - LLVM Compiler crash, apparently caused by running out of stack. Increased stack size.
  - LLVM and OpenMP code bases evolve rapidly
    - Encountered another crash in LLVM 15, apparently fixed in main branch
      - But random updates from main branch may or may not work
  - Eigen: Had to find the right preprocessor macros to turn off many optimized and processor-specific features (vectorization, CPU id, CUDA, …)
    - Still needs some code changes in Eigen to compile on NVIDIA
    - AMD GPU compilation still results in a compiler assertion failure in Eigen

OpenMP Target status (2)

M. Dewing

- Multiple compilers
  - AMD AOMPCC is a script that drives clang
    - Doesn't recognize `.cc` suffix as C++
    - Script will not pass through arguments that take values (eg. `-isystem /usr/include`)
  - NVIDIA HPC SDK
    - Does not support critical sections in GPU code, need to convert to atomics
    - Dependency file outputs (-MMD) are named differently than GCC. Must specify file name (-MF)
  - Intel One API (icpx)
    - SPIRV does not allow zero-sized arrays

OpenMP Target status (3)

M. Dewing

- Able to use OpenMP Target offloading from an application that uses TBB for "outermost-loop" concurrency
- Very first performance look, only on raw-to-cluster module, on a laptop
  - Serial: 30 events/s
  - OpenMP (using critical sections): 6 events/s
  - OpenMP (using atomics): 25 events/s
  - Data is copied to and from GPU on each kernel call
  - Used first critical sections where CUDA version uses atomics
    - Just to get something running correctly

Fermilab

Other versions

- Serial (M. Dewing, M. Kortelainen)
  - Original CUDA code came with an in-house hack to compile the CUDA code on CPU for some part of the code
  - First Serial version was to use that hack for all code, codebase still has CUDA look&feel
  - Mark reduced the CUDA look&feel for Serial version to be a better base for the OpenMP version (using Comby tool to express the code transformations)
- Direct HIP (M. Kortelainen)
  - Straightforward conversion from CUDA with `hipify-perl` to have a native version for AMD GPU
  - Can't be used for HIP-on-NVIDIA because of Eigen not supporting that mode
- CUDA with Unified Memory (M. Kortelainen, M. Kwok)
  - Demonstrate the cost of unified memory (40-45 % reduction in throughput)
  - Reported in vCHEP21 (doi:10.1051/epjconf/202125103035)

Fermilab

Outlook

- Performance measurements at JLSE
  - Still in setting up and testing phase
  - Compare all possible cases on CPUs and NVIDIA, AMD, Intel GPUs
    - How to get public Intel GPU results?
  - Results to be reported in CHEP23
    - Need to go through CMS, as we did for vCHEP21
- Additional manpower for profiling could be useful to answer some remaining mysteries
  - Why Alpaka (and Kokkos) versions yield better performance than serial version on CPU?
  - Why Kokkos is so much slower than Alpaka and native CUDA?
  - …