Porting CMS Heterogeneous Pixel Reconstruction to Kokkos

Taylor Childers\textsuperscript{1}, Matti J Kortelainen\textsuperscript{2}, Martin Kwok\textsuperscript{2}, Alexei Strelchenko\textsuperscript{2}, Yunsong Wang\textsuperscript{3}

\textsuperscript{1}ANL  \textsuperscript{2}FNAL  \textsuperscript{3}LBNL

vCHEP 2021  18 May 2021
Introduction

- CMS will use 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 → looking for a technology enabling portable code between CPU and different GPUs (at least)

- Used Patatrack heterogeneous pixel reconstruction from the CMS experiment as a use case for a set of realistic algorithms utilizing GPU effectively
- In this talk we report preliminary experience of porting the heterogeneous pixel reconstruction code from CUDA to Kokkos

- For more reports on the use of Kokkos by HEP-CCE see
  - FastCaloSim (add link)
  - WireCell (add link)
CMS Heterogeneous Pixel reconstruction

• About 40 kernels organized in 5 “framework modules”
• Raw pixel detector data (~250 kB/event) transferred to the GPU
• Only final results transferred back to the host: ~4 MB for tracks, ~90 kB for vertices
  – Not considered in this talk
• Standalone program to enable rapid prototyping
  – Flexible GNU Make-based build system
  – Simple framework mimicking CMSSW’s use of TBB tasks
  – Disk I/O contribution to time measurements is ignored
    • Recycling 1000 events from TTbar + pileup 50 simulation from CMS Open Data
Introduction to Kokkos

- Kokkos is a C++ library for writing performance portable applications
  - Single-source implementation, descriptive programming model
- Supported backends in Kokkos 3.3.1
  - Host serial
  - Host parallel: OpenMP, POSIX threads
  - Device parallel: CUDA, HPX, HIP, OpenMP Target, SYCL 2020
- We have tested serial, threads, CUDA, and HIP backends
- High-level API
  - `parallel_for`, `parallel_scan`, `parallel_reduce`; can be nested
  - Details of iteration and operations controlled with a policy
  - Kokkos::View<T> as an N-dimensional array of type T
    - Works similar to `std::shared_ptr`
    - Memory layout can be controlled, default layout depends on the backend
Porting experience

• Building
  – Kokkos requires a runtime library, only subset of backends can be enabled at a time
  – Spent a lot of time to figure out build rules when used in conjunction with shared libraries
    • For CUDA had to build Kokkos as a static library with -fPIC
    • For HIP could (had to) build Kokkos as a dynamic library

• Algorithms
  – Mostly straightforward to port from CUDA

• Data structures, i.e. Kokkos::View
  – Useful for unified memory allocation interface and smart pointer
  – Not particularly useful building block for Structures-of-Arrays of runtime size
  – By default initializes the data on device memory, can be expensive if not needed

• Able to run on CPU and NVIDIA GPUs, same results within numerical precision
  – Run-time failures with AMD GPUs, being investigated further
Performance comparison (CPU)

- On Cori GPU nodes at NERSC
- Pinned on one socket of Xeon Gold 6148 ("Skylake")
  - 20 cores, 40 threads
- Filled the socket with CPU intensive work
- Repeated 8 times on random nodes, jobs run 5-10 minutes at a time
- Different parallelization strategies
  - “CPU version”: process events concurrently, one event per thread
  - “Kokkos Threads backend”: one event in flight, parallelize each `parallel_X` call

<table>
<thead>
<tr>
<th>Test</th>
<th>Throughput (events/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU version, 1 thread</td>
<td>13.5 ± 0.2</td>
</tr>
<tr>
<td>Kokkos version, Serial backend</td>
<td>8.5 ± 0.2</td>
</tr>
<tr>
<td>CPU version, 40 threads</td>
<td>539 ± 9</td>
</tr>
<tr>
<td>Kokkos with Threads backend, peak at 18 threads</td>
<td>28 ± 1</td>
</tr>
</tbody>
</table>
### Performance comparison (NVIDIA GPU)

<table>
<thead>
<tr>
<th>Test</th>
<th>Throughput (events/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA version</td>
<td></td>
</tr>
<tr>
<td>9 concurrent events and threads (peak)</td>
<td>1840 ± 20</td>
</tr>
<tr>
<td>1 concurrent event</td>
<td>720 ± 20</td>
</tr>
<tr>
<td>1 concurrent event, caching allocator disabled</td>
<td>159 ± 1</td>
</tr>
<tr>
<td>Kokkos version</td>
<td></td>
</tr>
<tr>
<td>CUDA backend</td>
<td>115.7 ± 0.3</td>
</tr>
</tbody>
</table>

- On Cori GPU nodes, with 1 NVIDIA V100 GPU, repeated 8 times
- No concurrent events or memory pool with Kokkos, so disable those in CUDA version for comparison
Conclusions

• Successfully ported Patatrack heterogeneous pixel reconstruction to Kokkos
• Kokkos provides an API that is at higher level than CUDA
  – Potentially easier to use by physicists
• Achieved almost full portability between CPU, CUDA, and HIP
  – Runs on CPU and on NVIDIA GPU, produces same results within numerical precision
• Current Kokkos port has still a large overhead compared to native implementations
• Kokkos could work well for a project that compiles code separately for each target architecture, does not use much shared libraries, uses CMake as build system, and does not rely on concurrent work outside Kokkos