Accelerating X-ray Tracing for Exascale Systems using Kokkos

Felix Wittwer¹, Nicholas Sauter¹, Derek Mendez¹, Billy Poon¹, Aaron Brewster¹, James Holton¹, Michael Wall², William Hart³, Deborah Bard¹, Johannes Blaschke¹

¹Lawrence Berkeley National Lab
²Las Alamos National Lab
³Sandia National Lab
Serial femtosecond crystallography (SFX)
SFX Challenges

- Measurement time is scarce
- Samples are limited
- Is the collected data useful?
  - hit rate, crystal quality, beam problems, etc
- Move on to next sample?
- live feedback required!
- new instruments will produce huge amounts of data
  - from 100 images/s to more than a million images/s
- → use superfacility for fast data analysis
Superfacility

• send data directly from experiment to supercomputer
  o fast and robust data transfer of terabytes
• analyse, then send results back to experimental site
  o Perlmutter not always available → use Frontier or Aurora
• Perlmutter/Frontier/Aurora use different hardware (Nvidia/AMD/Intel)
  o code fragmentation?
• portability options: OpenACC, Kokkos, OpenMP target,...
Kokkos

• C++ programming model for performance portability
  o no new syntax (#pragma omp, <<<>>>, etc)
• provides abstract execution and memory spaces
  o possible spaces: CPU, GPU,...
• allows to specify hardware target only during compilation
  o make -DKOKKOS_ARCH=Ampere80 -DKOKKOS_DEVICES=CUDA
  o make -DKOKKOS_ARCH=Zen3 -DKOKKOS_DEVICES=OpenMP
• Introduction available:
  github.com/kokkos/kokkos-tutorials/wiki/Kokkos-Lecture-Series
Kokkos – execution patterns

• parallelize for-loop

```cpp
for (i=0; i<pixels; ++i) {
    result[i] = scattering(i, parameters);
}
```

Kokkos::parallel_for(pixels, [=] (const int i) {
    result(i) = scattering(i, parameters);
});

• three patterns: parallel_for, parallel_reduce and parallel_scan
Kokkos – memory management

- GPU has no direct access to system memory
  → requires data transfer between GPU and system memory
- Creating a zero array

```c
    double* cu_image;
    cudaMalloc((void**)&cu_image, sizeof(*cu_image)*size);
    cudaMemcpy((void*)cu_image, 0, sizeof(*cu_image)*size);
    ...
    cudaFree(cu_image);
```

```cpp
    Kokkos::View<double*> view_image("image", size);
```
nanoBragg

• simulate diffraction images at pixel level  
  → massively parallel, well suited for GPUs
• original code written in C++
• port to CUDA resulted in 20x speed-up
• port from CUDA to Kokkos:
  o replace kernels with parallel_for patterns
  o replace CUDA arrays with views
Performance

• How did the performance change by switching to Kokkos?
• Benchmark: simulate 100,000 images on 128 Perlmutter nodes

<table>
<thead>
<tr>
<th></th>
<th>CUDA</th>
<th>Kokkos</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>150 s</td>
<td>126 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(16% faster)</td>
</tr>
</tbody>
</table>

• Portability?
  o 128 Crusher nodes (Frontier test system) with AMD MI250X

<table>
<thead>
<tr>
<th></th>
<th>Perlmutter</th>
<th>Crusher</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>126 s</td>
<td>54 s</td>
</tr>
<tr>
<td></td>
<td>(57% faster)</td>
<td></td>
</tr>
</tbody>
</table>
Summary

- Kokkos allows to use GPUs without vendor lock-in
- Same code achieves performance on notebooks and HPC centers
- Porting from C++ or CUDA to Kokkos relative straightforward
- No new syntax or language, pure C++
- Limited CUDA library support
- Kokkos port improved performance by 20% on Nvidia hardware
- Portable performance for Nvidia, AMD or Intel GPU
  (switching to AMD hardware increased performance by 60%)
Acknowledgement

- ExaFEL is part of the Exascale Computing project
- Collaboration between SLAC, LBNL and LANL
- Special thanks to Johannes Blaschke, Deborah Bard, Nicholas Sauter, Aaron Brewster, Billy Poon, Iris Young, Derek Mendez, James Holton, Michael Wall, William Hart
Some more slides
Background

- Computational Crystallographic Toolbox (github.com/cctbx/cctbx_project)
- X-ray crystallography
Data Acquisition → Spinning Disk → XRooTD ~ 15 TB/day → SCRATCH, CFS, DataWarp

Users interact with data analysis in real-time → Workflow Coordination → MySQL → Cori Compute Nodes

(J. Blaschke)
nanoBragg – performance
## Kernel run-times

<table>
<thead>
<tr>
<th></th>
<th>nanoBraggSpots</th>
<th>addBackground</th>
<th>addArray</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA</td>
<td>8.28ms</td>
<td>1.87ms</td>
<td>0.13ms</td>
</tr>
<tr>
<td>Kokkos</td>
<td>6.98ms</td>
<td>1.76ms</td>
<td>0.12ms</td>
</tr>
<tr>
<td>Speed-up</td>
<td>+15.7%</td>
<td>+5.9%</td>
<td>7.7%</td>
</tr>
</tbody>
</table>
### nanoBraggSpots – nsight details

<table>
<thead>
<tr>
<th></th>
<th>CUDA</th>
<th>Kokkos</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run-time</td>
<td>8.28ms</td>
<td>6.98ms</td>
</tr>
<tr>
<td>Compute throughput</td>
<td>65.05%</td>
<td>77.42%</td>
</tr>
<tr>
<td>Memory throughput</td>
<td>21.05%</td>
<td>21.35%</td>
</tr>
<tr>
<td>Registers</td>
<td>130</td>
<td>116</td>
</tr>
<tr>
<td>Theoretical occupancy</td>
<td>18.75%</td>
<td>25%</td>
</tr>
<tr>
<td>Achieved occupancy</td>
<td>16.8%</td>
<td>24.74%</td>
</tr>
</tbody>
</table>
void Container::init() {
  Kokkos::parallel_for("init", size,
    KOKKOS_LAMBDA (const int& j) {
      data(j) = m_value;
    });
}

void Container::init() {
  Kokkos::parallel_for("init", size,
    KOKKOS_LAMBDA (const int& j) {
      this->data(j) = this->m_value;
    });
}