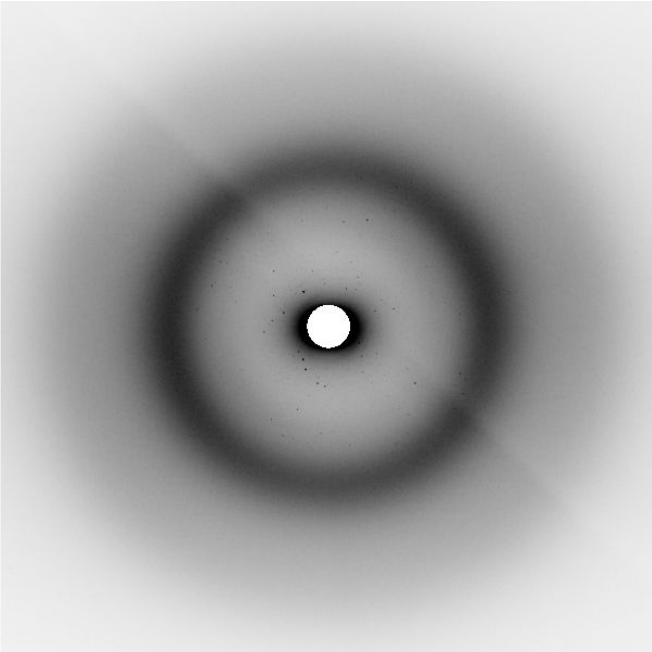
Accelerating X-ray Tracing for Exascale Systems using Kokkos

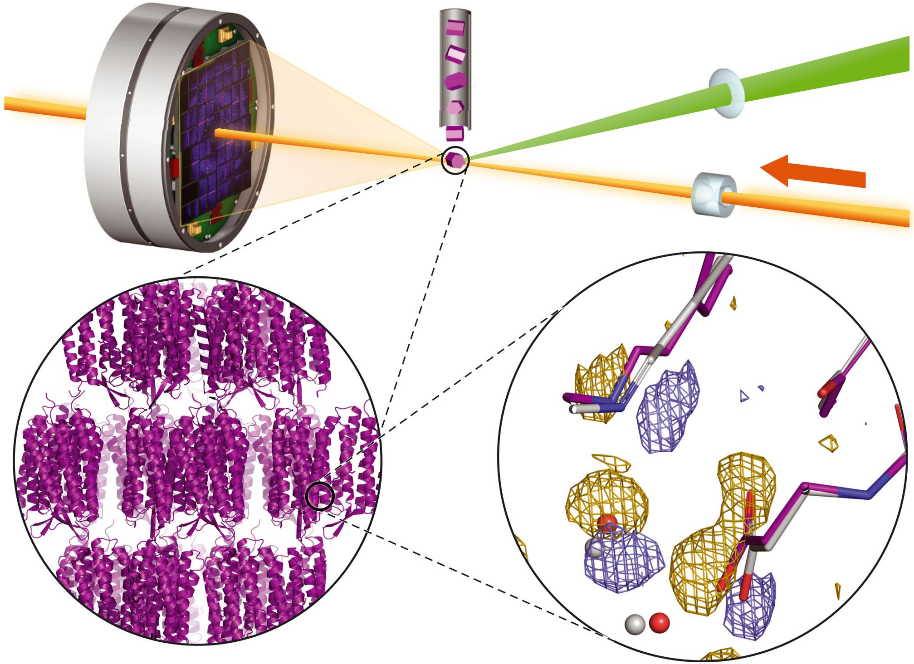
Felix Wittwer[1], Nicholas Sauter[1], Derek Mendez[1], Billy Poon[1], Aaron Brewster[1], James Holton[1], Michael Wall[2], William Hart[3], Deborah Bard[1], Johannes Blaschke[1]

[1]Lawrence Berkeley National Lab
[2]Las Alamos National Lab
[3]Sandia National Lab

GPUs for Science day

Serial femtosecond crystallography (SFX)





(Brändén, Science 2021)

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

Kokkos

- C++ programming model for performance portability
  - no new syntax (#pragma omp, <<<>>>, etc)
- provides abstract execution and memory spaces
  - possible spaces: CPU, GPU,...
- allows to specify hardware target only during compilation

```
make -DKOKKOS_ARCH=Ampere80 -DKOKKOS_DEVICES=CUDA
make -DKOKKOS_ARCH=Zen3 -DKOKKOS_DEVICES=OpenMP
```

- Introduction available: github.com/kokkos/kokkos-tutorials/wiki/Kokkos-Lecture-Series

Kokkos – execution patterns

- parallelize for-loop

policy

body

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

C++

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

Kokkos

- 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

```
double* cu_image;
cudaMalloc((void**)&cu_image, sizeof(*cu_image)*size);
cudaMemset((void*)cu_image, 0, sizeof(*cu_image)*size);
…
cudaFree(cu_image);
```

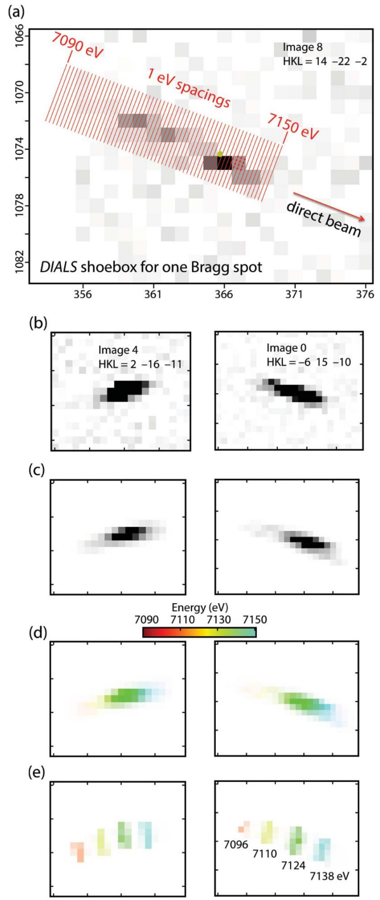
Cuda

```
Kokkos::View<double*> view_image(“image”, size);
```

Kokkos

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:
  - replace kernels with parallel_for patterns
  - replace CUDA arrays with views



Performance

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

CUDA	Kokkos
150 s	126 s (16% faster)

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

Perlmutter	Crusher
126 s	54 s (57% faster)

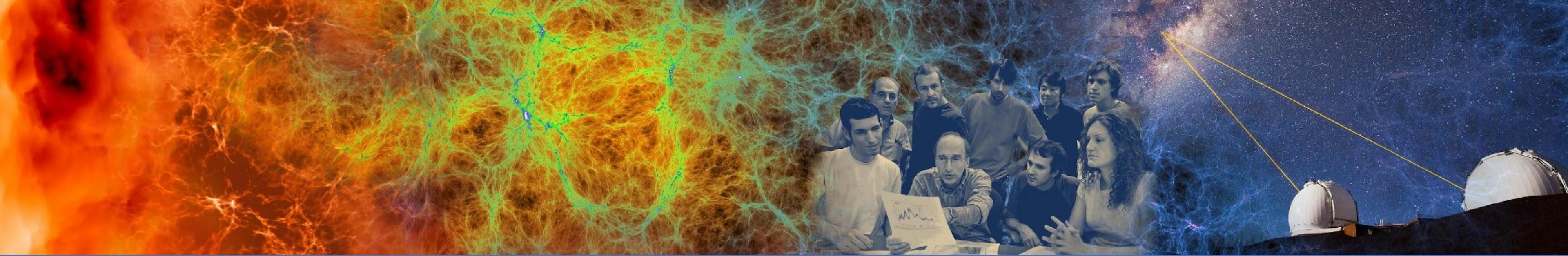
Summary

- 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

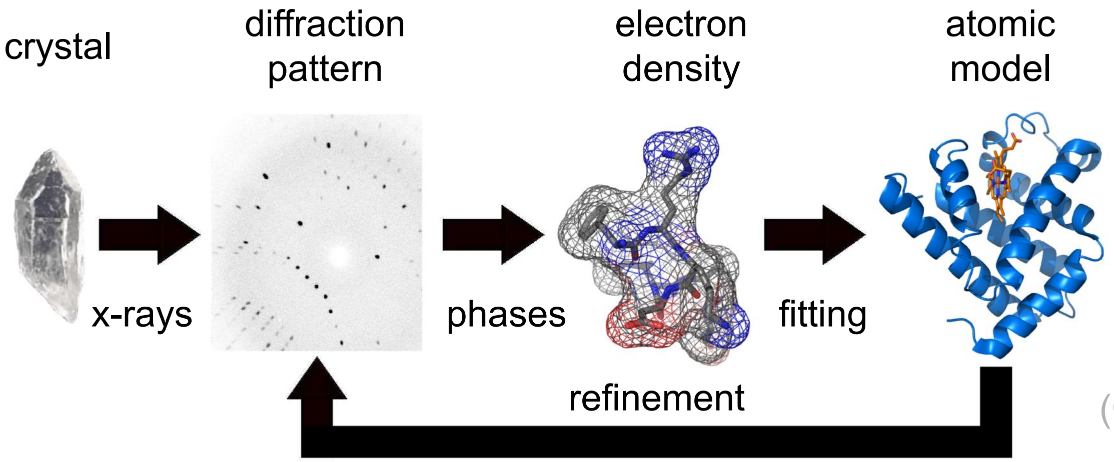
EXASCALE COMPUTING PROJECT

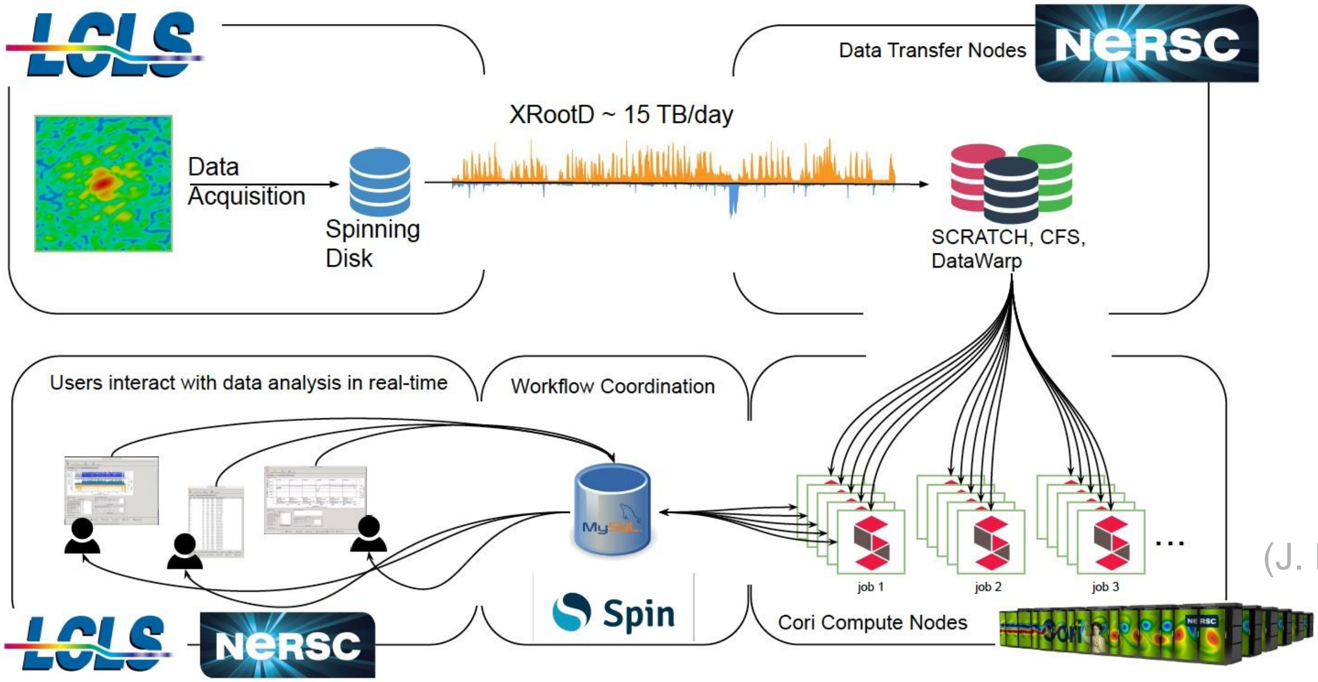


Some more slides

Background

- **C**omputational **C**rystallographic **Tool**box (github.com/cctbx/cctbx_project)
- X-ray crystallography

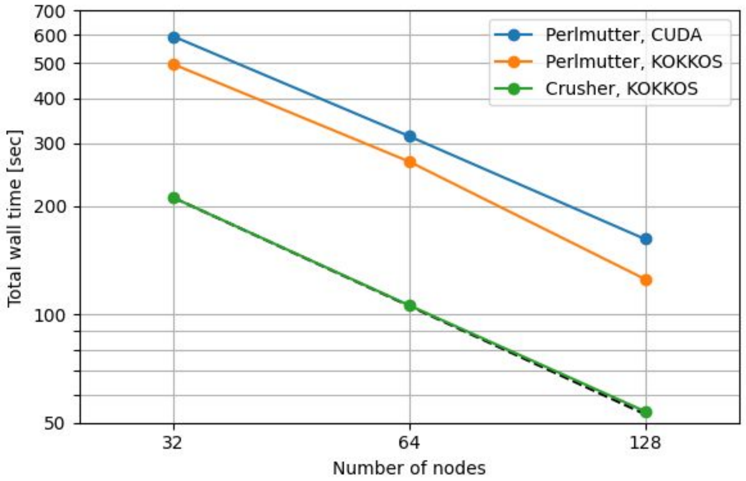


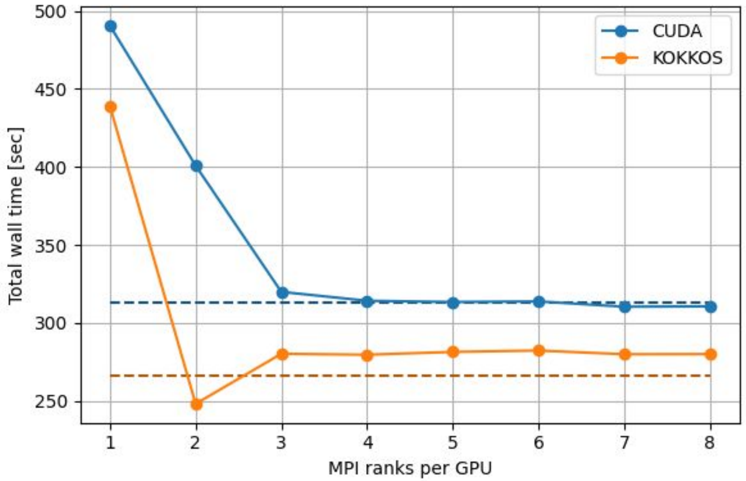


Data Transfer Nodes
XRootD ~ 15 TB/day
Data
Acquisition
Spinning
Disk
SCRATCH, CFS,
DataWarp
Users interact with data analysis in real-time
Workflow Coordination
Spin
job 1
job 2
job 3
Cori Compute Nodes

(J. Blaschke)

nanoBragg – performance





Kernel run-times

	nanoBraggSpots	addBackground	addArray
CUDA	8.28ms	1.87ms	0.13ms
Kokkos	6.98ms	1.76ms	0.12ms
Speed-up	+15.7%	+5.9%	7.7%

nanoBraggSpots – nsight details

	CUDA	Kokkos
Run-time	8.28ms	6.98ms
Compute throughput	65.05%	77.42%
Memory throughput	21.05%	21.35%
Registers	130	116
Theoretical occupancy	18.75%	25%
Achieved occupancy	16.8%	24.74%

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

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