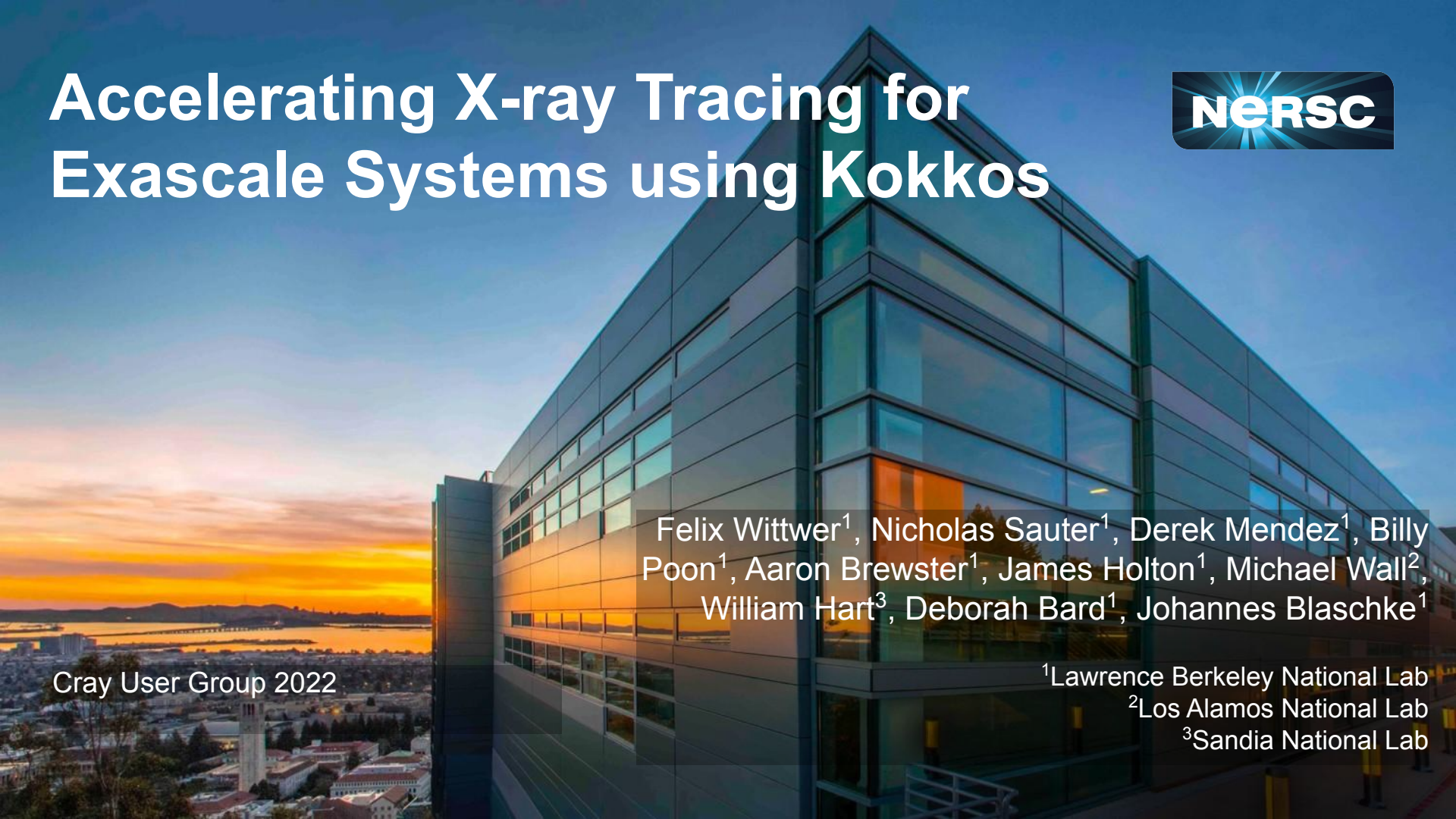
Accelerating X-ray Tracing for Exascale Systems using Kokkos

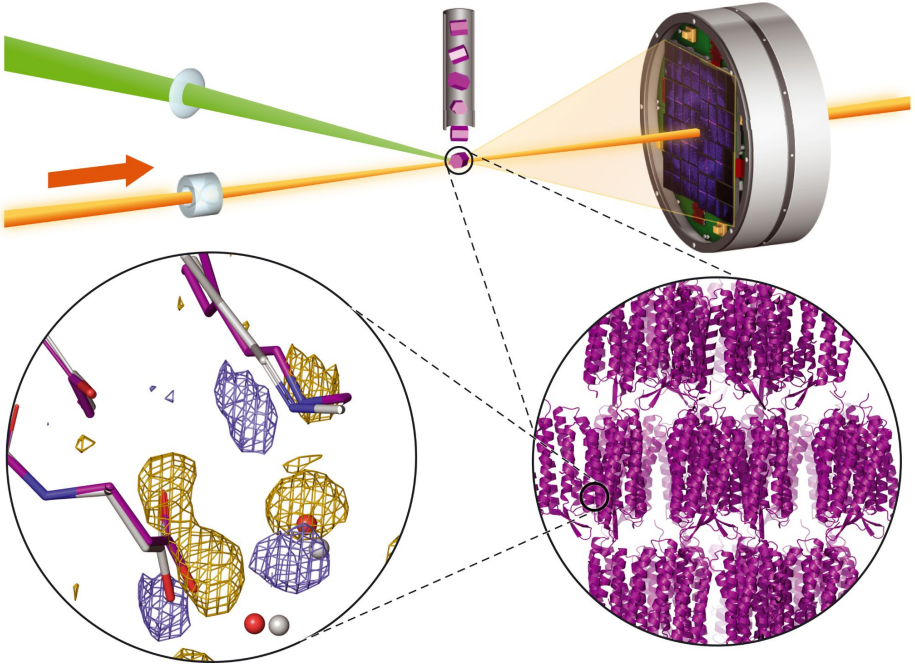


Felix Wittwer[1], Nicholas Sauter[1], Derek Mendez[1], Billy Poon[1], Aaron Brewster[1], James Holton[1], Michael Wall[2], William Hart[3], Deborah Bard[1], Johannes Blaschke[1]

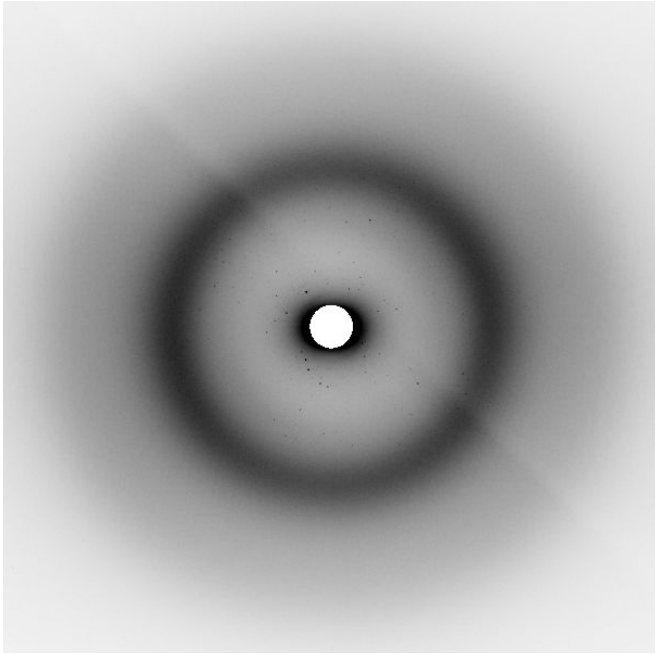
[1]Lawrence Berkeley National Lab
[2]Los Alamos National Lab
[3]Sandia National Lab

Cray User Group 2022

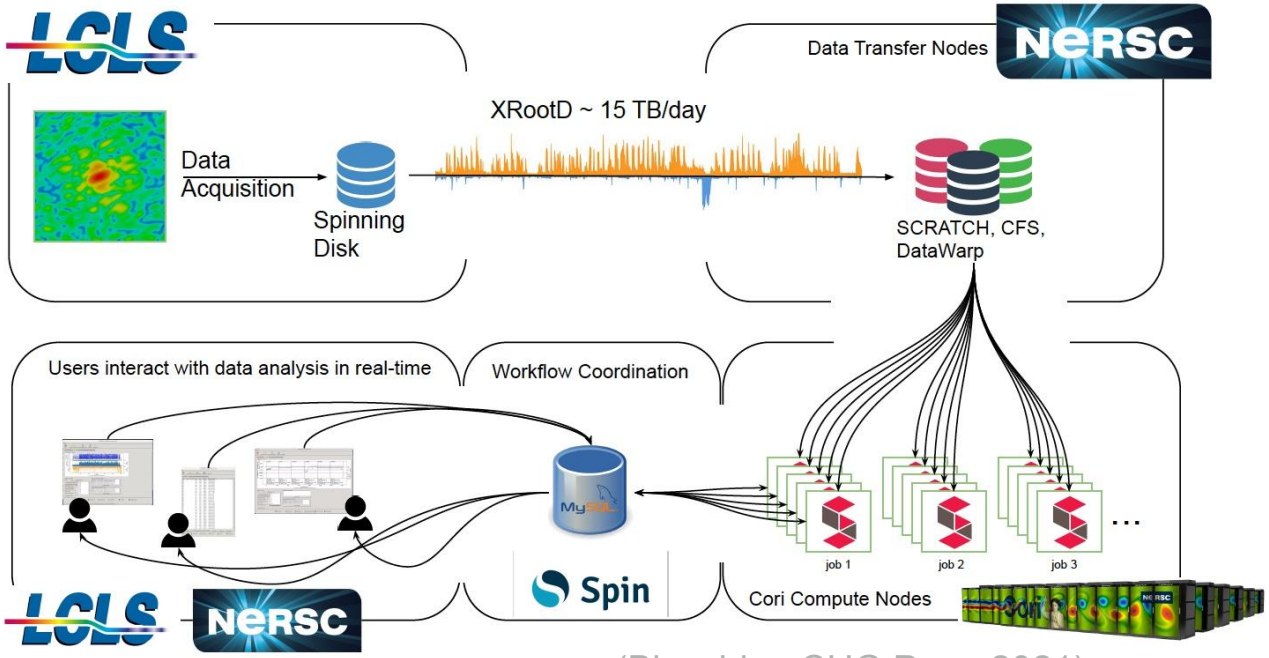
Serial Femtosecond Crystallography (SFX)



(Brändén, Science 2021)



Superfacility



(Blaschke, CUG Proc. 2021)

Experimental 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?

Computing challenges

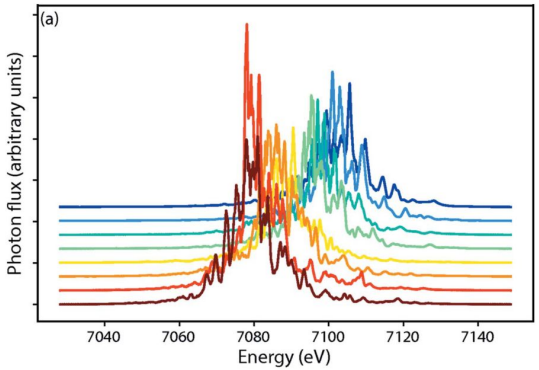
- LCLS-II HE upgrade will generate up to 1000x more data
  - How to ensure quick feedback?
- Upcoming Exascale systems
  - Frontier (Oakridge) based on AMD MI250X GPUs
  - Aurora (Argonne) based on Intel Xe GPUs
- Code must run on diverse hardware
  - options: OpenMP offloading, OpenACC, Kokkos, etc

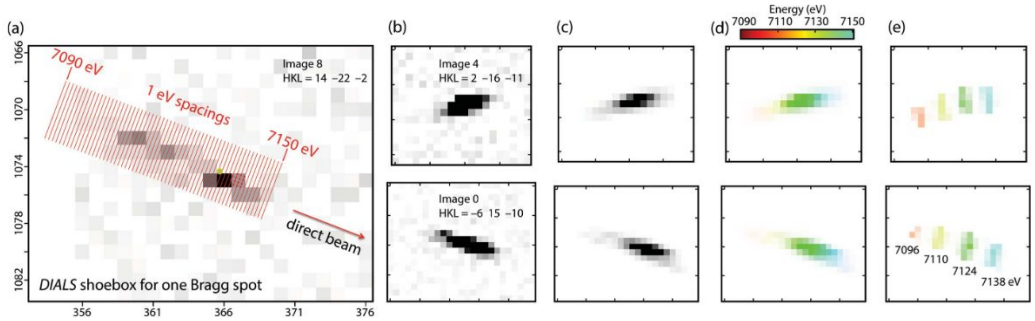
Kokkos

- C++ programming model
- portability through abstractions
  - memory spaces, data structures
  - execution spaces, execution patterns
- Library, not a new language or language extension
- Details
  - paper (DOI: 10.1109/TPDS.2021.3097283)
  - https://github.com/kokkos/kokkos

nanoBragg

- simulates diffraction images at pixel level
- includes X-ray spectrum, crystal mosaicity, orientation
- Details in Sauter *et al*, Acta Cryst. D 2020





nanoBragg

- Written in CUDA
- Main calculations are done in three kernels
  - nanoBraggSpots
    most complex kernel; simulates Bragg spots
  - addBackground
    second most complex; simulates air and water scattering
  - addArray
    simple kernel; adds results from other two kernels

Porting CUDA to Kokkos I

- Replace CUDA array with Kokkos::View
  - Creating a zero array in CUDA

```
double* cu_image;
cudaSafeCall(cudaMalloc((void**)&cu_image, sizeof(*cu_image)*image_size));
cudaSafeCall(cudaMemset((void*)cu_image, 0, sizeof(*cu_image)*image_size));
```

  - And in Kokkos

```
Kokkos::View<double*> view_image("image", image_size);
```

- Similar situation when transferring data from host to GPU

Porting CUDA to Kokkos II

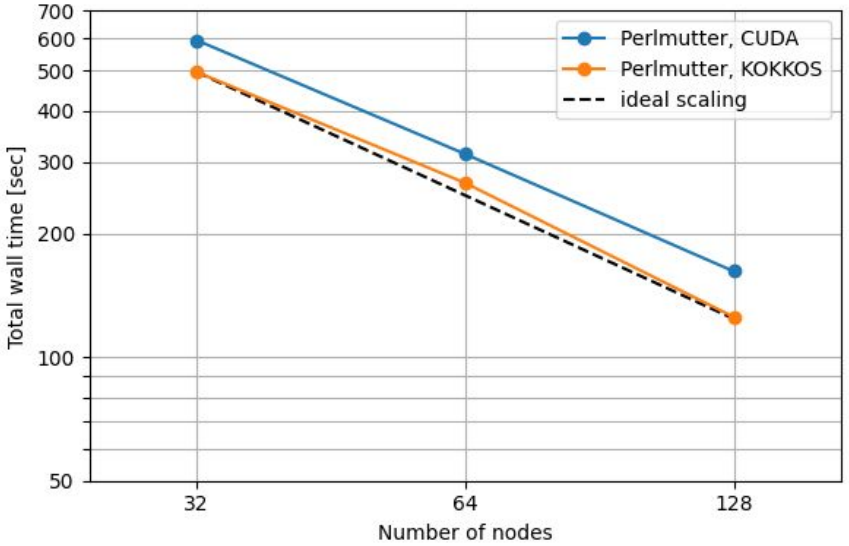
- Convert kernels to parallel_for patterns

```
__global__ void addArrayCUDA(double* lhs, float* rhs, int size) {
      int j = blockDim.x * blockIdx.x + threadIdx.x;
      if (j<size) {
            lhs[j] = lhs[j] + (double) rhs[j];
      }
}
addArrayCUDA<<<numBlocks, threadsPerBlock>>>(cu_image, cu_bg, image_size);
```

```
void addArray(Kokkos::View<double*> lhs, Kokkos::View<float*> rhs, int size) {
      Kokkos::parallel_for(size, KOKKOS_LAMBDA (const int& j) {
            lhs(j) = lhs(j) + (double) rhs(j);
      });
}
addArray(view_image, view_bg, image_size);
```

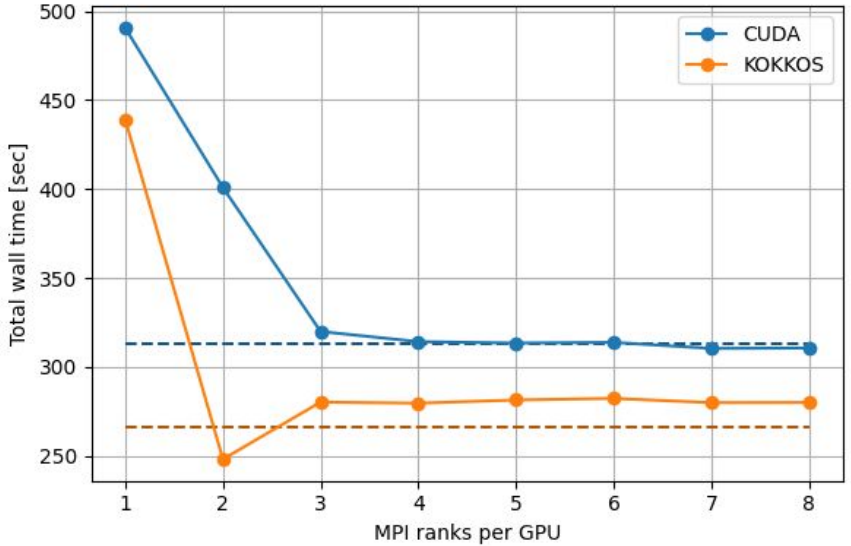
Kokkos performance

- Benchmark: time to simulate 100,000 scattering images (less is better)
- nearly ideal strong scaling
- Kokkos 15% faster than Cuda baseline
  - until now no focus on optimization
  - no use of advanced Kokkos features (e.g. teams)



Kokkos performance

- Benchmark: time to simulate and write 100,000 scattering images (less is better)
- Multi-tenancy allows to hide I/O latency



Kokkos profiling

- Use Nsight profile and Nsight compute
- Kernel times

	NanoBraggSpots	addBackground	addArray
Cuda	8.28 ms	1.87 ms	0.13 ms
Kokkos	6.98 ms	1.76 ms	0.12 ms
Speed-up	15.7%	5.9%	7.7 %

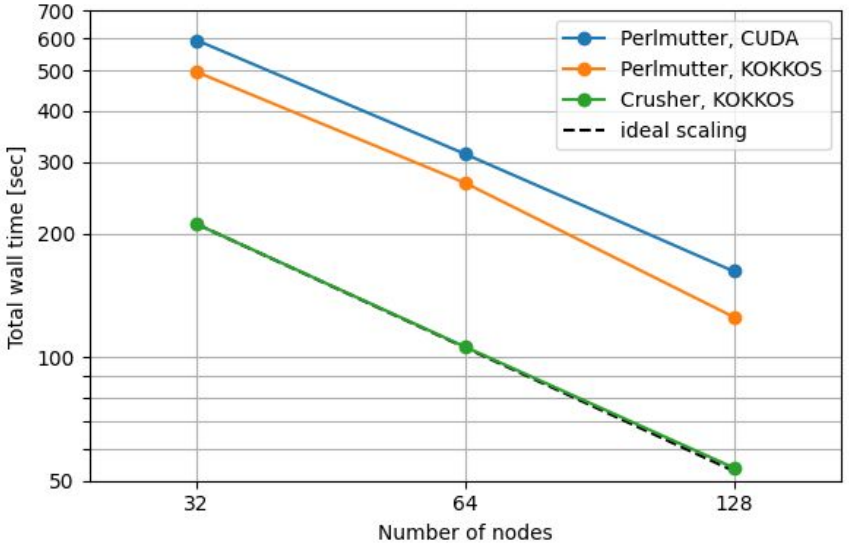
Kokkos profiling

- Use Nsight profile and Nsight compute
- Details for nanoBraggSpots kernel

	Registers	theoretical Occupancy	achieved Occupancy	Runtime
Cuda	130	18.75%	16.8%	8.28 ms
Kokkos	116	25%	24.74%	6.98 ms

Kokkos portability

- same benchmark
- Frontier testbed system
  - 4 AMD MI250X GPUs / node
- same code, different backend
- only change compiler flags
- 60% faster



Testing the Kokkos Port

```
[Me@Host:bin] srun nanoBragg_kokkos
terminate called after throwing an instance of 'std::runtime_error'
what():  cudaMemcpy(dst, src, n, cudaMemcpyDefault) error(
cudaErrorIllegalAddress): an illegal memory access was encountered
```

Testing the Kokkos Port

```
[Me@Host:bin] srun nanoBragg_kokkos
terminate called after throwing an instance of 'std::runtime_error'
what():  cudaMemcpy(dst, src, n, cudaMemcpyDefault) error(
cudaErrorIllegalAddress): an illegal memory access was encountered
```

```
void MyClass::init() {
     Kokkos::parallel_for(size, KOKKOS_LAMBDA (const int& j) {
          m_data(j) = m_value;
     });
}
```

Testing the Kokkos Port

```
[Me@Host:bin] srun nanoBragg_kokkos
terminate called after throwing an instance of 'std::runtime_error'
what():  cudaMemcpy(dst, src, n, cudaMemcpyDefault) error(
cudaErrorIllegalAddress): an illegal memory access was encountered
```

```
void MyClass::init() {
     Kokkos::parallel_for(size, KOKKOS_LAMBDA (const int& j) {
          this->m_data(j) = this->m_value;
     });
}
```

Testing the Kokkos Port

```
[Me@Host:bin] srun nanoBragg_kokkos
terminate called after throwing an instance of 'std::runtime_error'
what():  cudaMemcpy(dst, src, n, cudaMemcpyDefault) error(
cudaErrorIllegalAddress): an illegal memory access was encountered
```

```
void MyClass::init() {
      auto temp_data = m_data;
      auto temp_value = m_value;
      Kokkos::parallel_for(size, KOKKOS_LAMBDA (const int& j) {
            temp_data(j) = temp_value;
      });
}
```

Summary

- porting to Kokkos mostly straightforward
  - Lambda captures are useful, but can be problematic
- identical code runs on Cuda, HIP and OpenMP
- increased performance even on identical hardware
- performance on exascale test systems is promising
- future plans
  - continue to port more parts of CCTBX
  - test Kokkos-kernels as substitute for Cuda libraries

Acknowledgments

- NERSC
  - Deborah Bard
  - Johannes Blaschke
- LBL
  - Aaron Brewster
  - Billy Poon
  - Derek Mendez
  - James Holton
  - Nick Sauter
- LANL - Michael Wall
- Sandia - William Hart

- Exascale Computing Project
- NERSC resources
- OLCF resources

Thank you!