Performance Portability and Programming Models for C++ codes

Bálint Joó, Jefferson Lab

2019 NERSC User Group (NUG) Meeting

Rockville, MD

July 19, 2019

New Machines Coming! Look busy!

- AMD CPU + NVIDIA GPU
- Cray Slingshot Interconnect

If your code runs well on current
NVidia GPUs using CUDA
Or OpenACC you should be
Well positioned for
Perlmutter

- Intel CPU + Xe Technology

- AMD CPU + AMD GPU
- ROCm (HIP)

If your code runs well on
Summit with CUDA,
Converting to HIP
may be mostly
automated

What are some options out there?

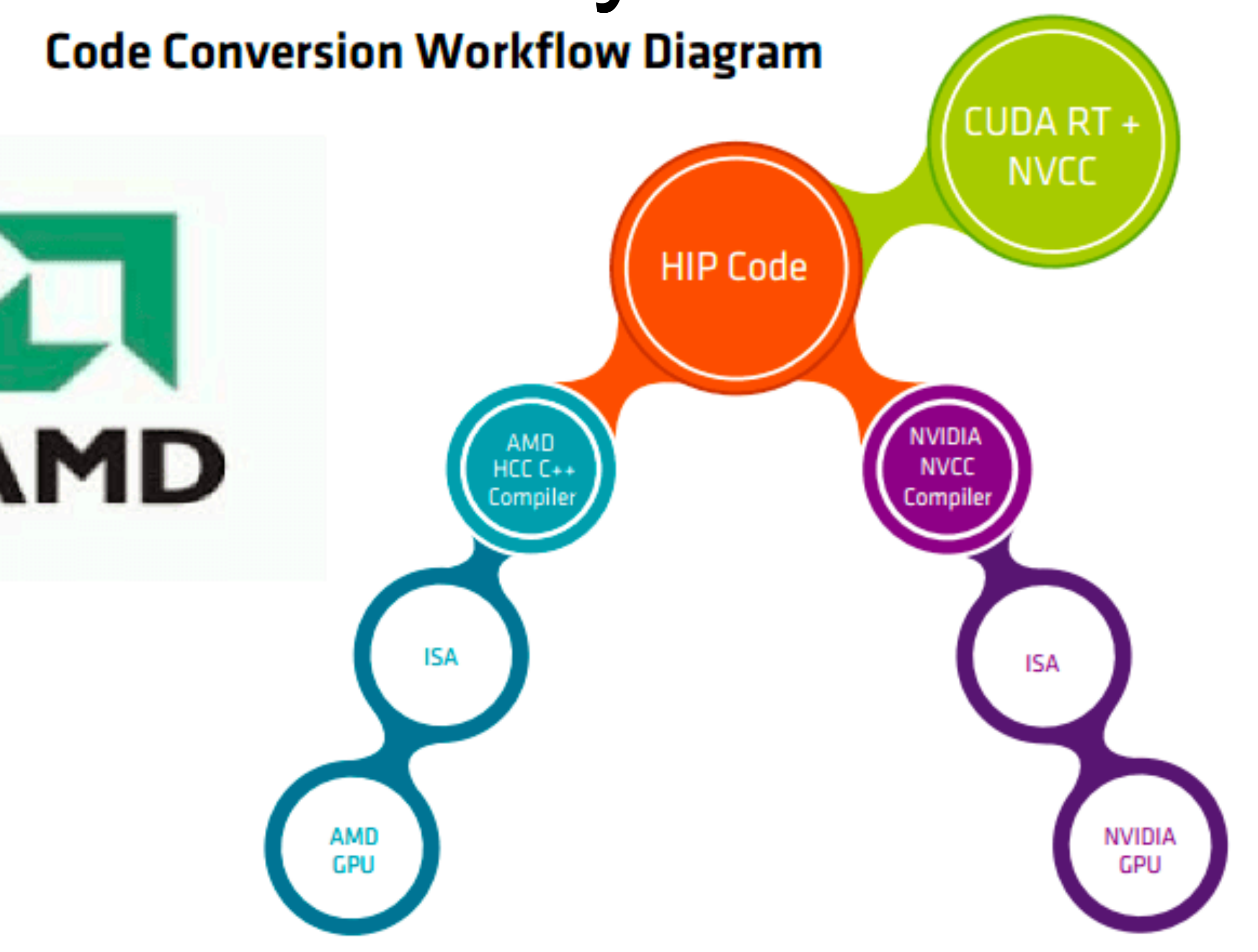
OpenMP & OpenACC

- #pragma directives for
  - Parallel for
  - Reduction
  - SIMD
  - Offload
  - Tasking
- Relies heavily on compiler

Kokkos & Raja

- More C++ oriented
  - Modern C++
  - Parallelism via functors & lambdas (forall, …)
  - Policy driven (execution & memory spaces)
- Many back ends
  - OpenMP, CUDA, ROCm,…

HIP by AMD



- AMDs Accelerator API
- GPU focus
  - Use OpenMP on CPU?
- (Very) Similar to CUDA
- Can target
  - AMD GPUs
  - NVIDIA GPUs

SyCL (Khronos)

- C++-11 extensions
- Functors & Lambdas
- Portable via lower Khronos layers (OpenCL)

OpenMP

- Offloaded axpy in OpenMP

```
#pragma omp target teams distribute parallel for simd
   map(to:z[:N]) map(a,x[:N],y[:N])
for(int i=0; i < N; i++) // N is large
{
   z[i] = a*x[i] + b[i];
}
```

- Collapses:
  - omp target - target the accelerator,
  - omp teams - create a league of teams
  - omp distribute - distribute the works amongst the teams
  - omp parallel for simd - perform a SIMD-ized parallel for
  - map a, x and y to the accelerator and map resulting z back out.

Kokkos

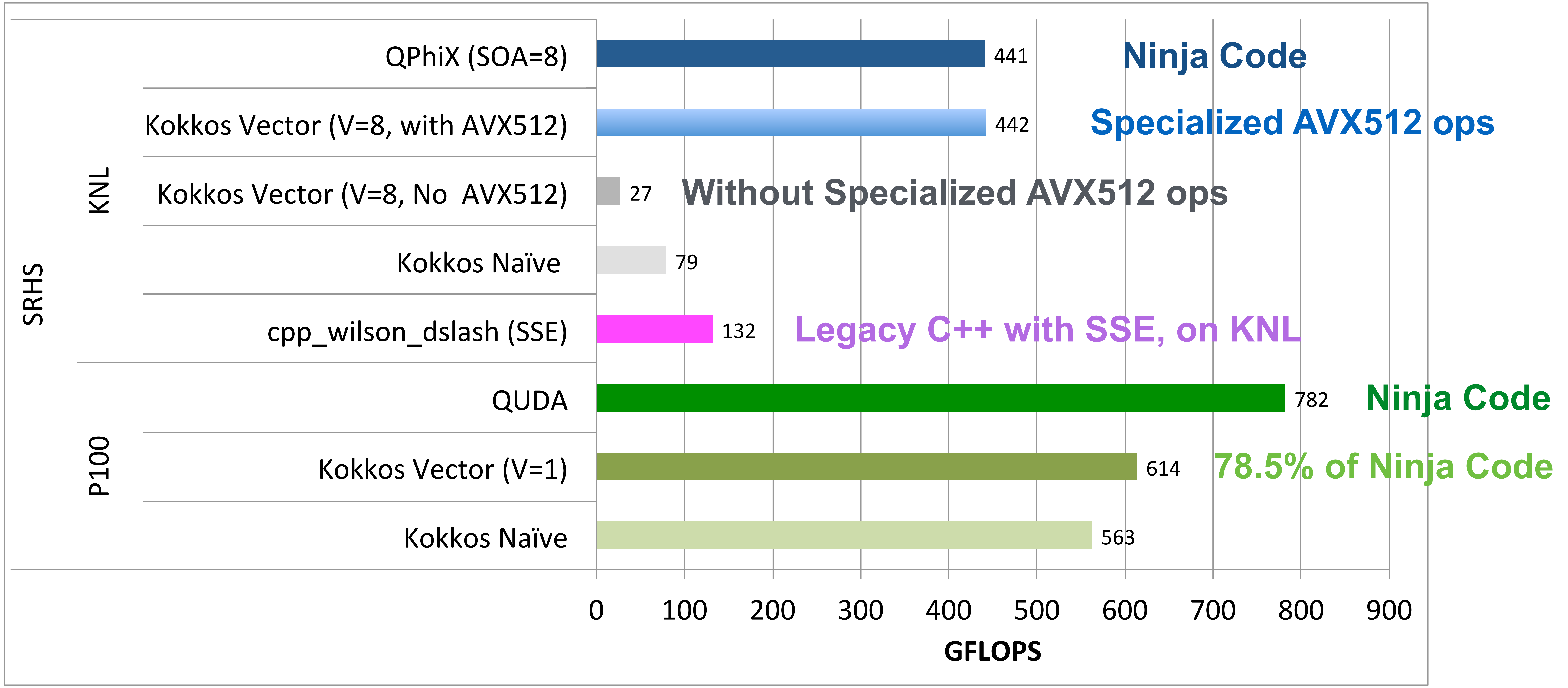
```
Kokkos::View<float[N],LayoutLeft,CudaSpace> x(“x”); // N is large
Kokkos::View<float[N],LayoutLeft,CudaSpace> y(“y”);
Kokkos::View<float[N],LayoutLeft,CudaSpace> z(“z”);

float a=0.5;

Kokkos::parallel_for(“zaxpy", N, KOKKOS_LAMBDA (const int& i) {
  z(i) = a*x(i) + y(i);   // view provides indexing operator()
});
```

- View - multi-dimensional array, index order specified by Layout, location by MemorySpace policy. Layout allows appropriate memory access for CPU/GPU
- Parallel for dispatches a C++ lambda
- **Portable:** Parallel for done with back end: OpenMP, CUDA, ROCm, …
- Kokkos developers on C++ standards committee - work to fold features into C++

Kokkos Performance Summary

Lattice QCD Wilson Dslash Operator (Sparse MV)

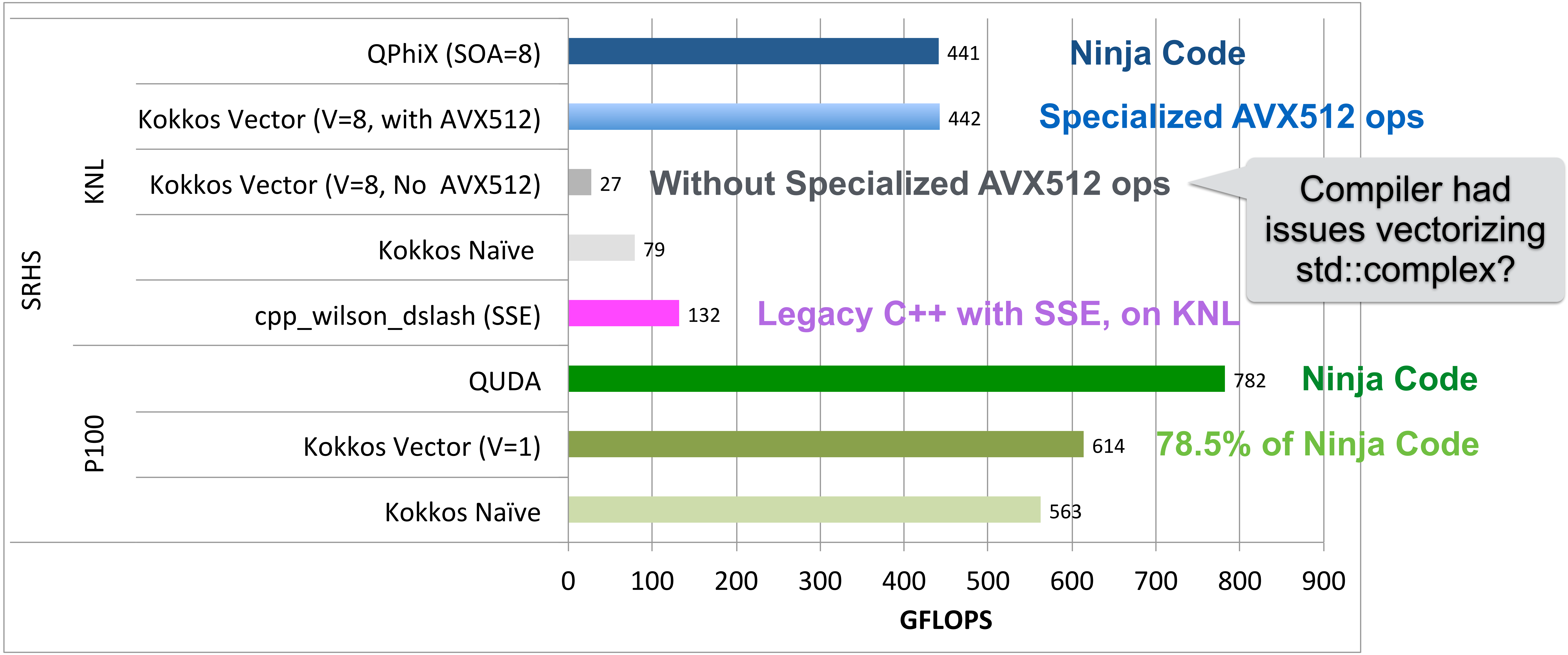
Vol=32x32x32x32 sites



Kokkos Performance Summary

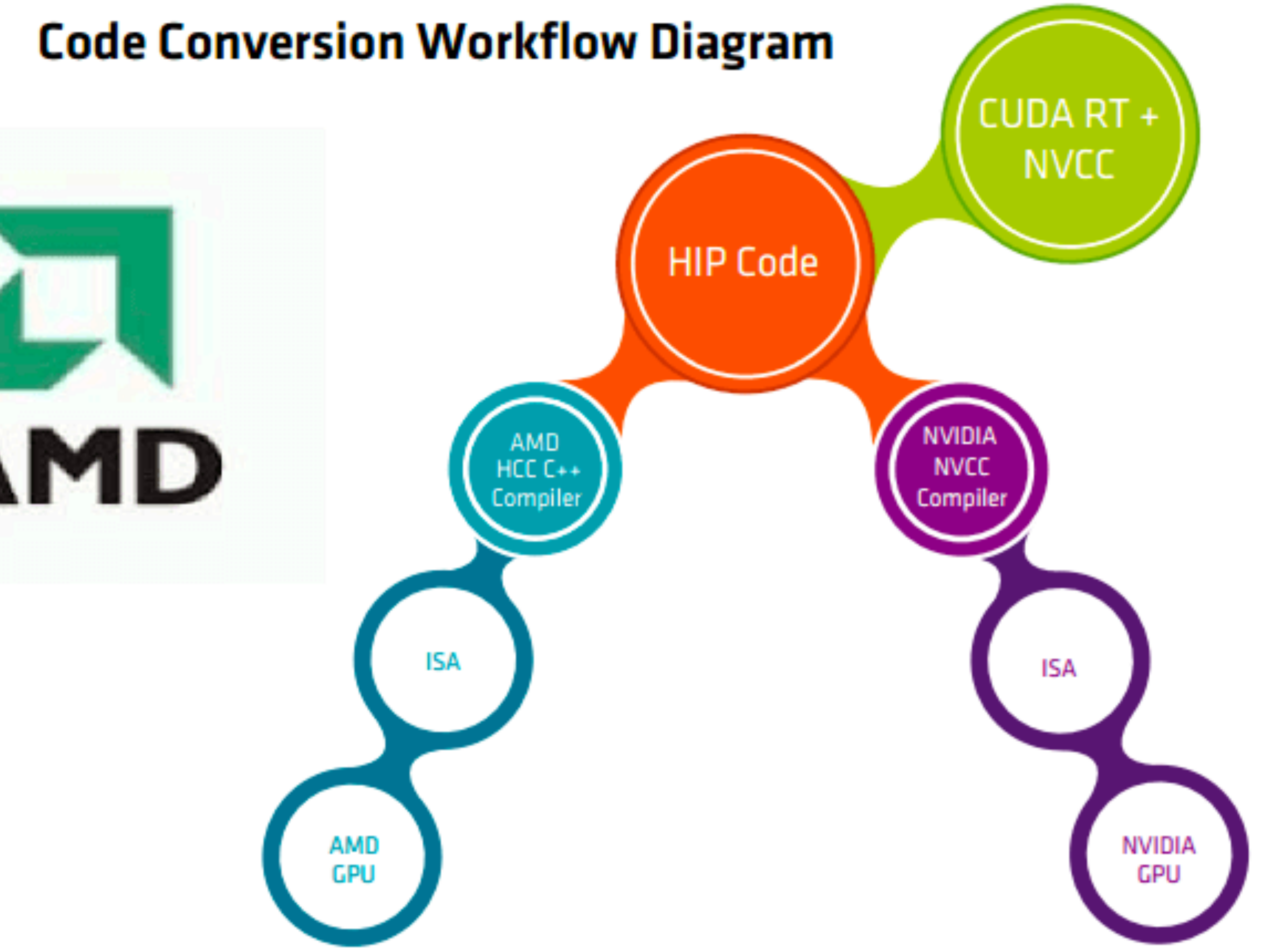
Lattice QCD Wilson Dslash Operator (Sparse MV)

Vol=32x32x32x32 sites



HIP

- HIP is AMD's "C++ Heterogeneous-Compute Interface for Portability"
- Take your CUDA API and replace 'cuda' with 'hip':
  - `cudaMemcpy()` -> `hipMemcpy()`
  - `kernel<<>>( )` -> `hipLaunchKernel(kernel,…)`
  - and other slight changes.
  - You can use ***hipify*** tool to do first pass of conversion automatically
- Open Source
- Portability between NVIDIA and AMD GPUs only.



SyCL

- C++11 based standard by Khronos group.
- Follows concepts of OpenCL
  - buffers, command queues, kernels, etc
- ‘Single Source File’ compilation
  - OpenCL kernels were in separate files
- much less verbose than OpenCL

```
Queue myQueue;
buffer<float,1> x_buf(LARGE_N);
buffer<float,1> y_buf(LARGE_N);
buffer<float,1> z_buf(LARGE_N);

// … fill buffers somehow …
float a = 0.5;
{
myQueue.submit([&](handler& cgh) {
  auto x=x_buf.getAccess<access::mode::read>(cgh);
  auto y=y_buf.getAccess<access::mode::read>(cgh);
  auto z=z_buf.getAccess<access::mode::write>(cgh);

  cgh.parallel_for<class zaxpy>(LARGE_N,[=](id<1> id){
      auto i = id[0];
      z[i]=a*x[i] + y[i];
  });
});
}
```

SyCL

- SYCL manages buffers
- Only access buffers via accessors
- can track accessor use and build data dependency graph to automate data movement
- What does this mean for non SyCL Libraries with pointers? (e.g. MPI)

```
Queue myQueue;
buffer<float,1> x_buf(LARGE_N);
buffer<float,1> y_buf(LARGE_N);
buffer<float,1> z_buf(LARGE_N);

// … fill buffers somehow …
float a = 0.5;
{
myQueue.submit([&](handler& cgh) {
   auto x=x_buf.getAccess<access::mode::read>(cgh);
   auto y=y_buf.getAccess<access::mode::read>(cgh);
   auto z=z_buf.getAccess<access::mode::write>(cgh);

   cgh.parallel_for<class zaxpy>(LARGE_N,[=](id<1> id){
        auto i = id[0];
        z[i]=a*x[i] + y[i];
   });
});
}
```

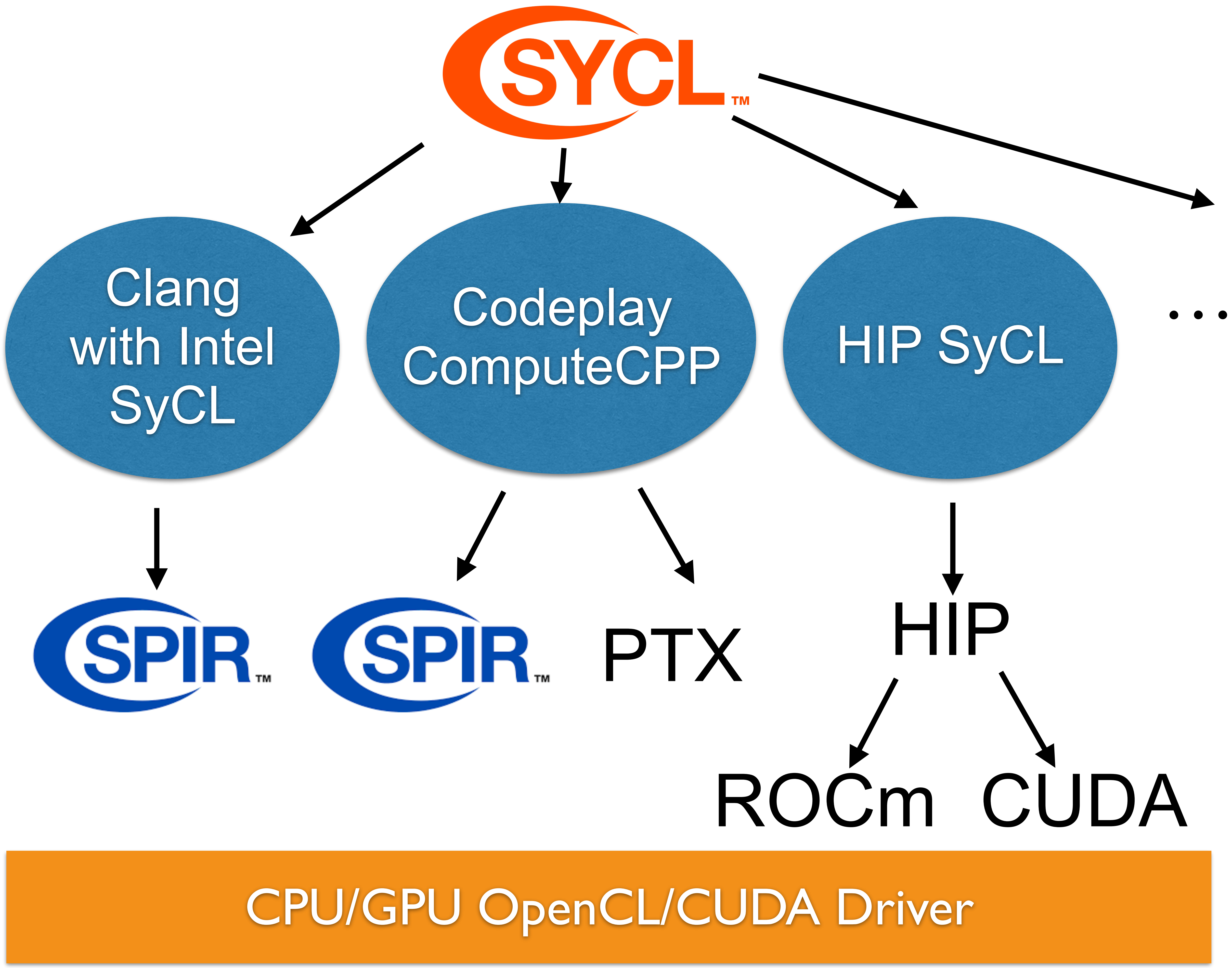
SyCL runtime manages data in buffers

access buffer data via accessors in command group (cgh) scope or host accessor

kernels must have a unique name in C++

How is SyCL portable?

- Essentially SyCL compiler creates OpenCL kernels from the Command Group Kernel functors/lambdas
- These can (in principle) compile into
  - SPIR: The Khronos Group's 'Standard Portable Intermediate Representation'
  - PTX: for NVIDIA GPUs
  - HIP and/or GCN ISA for AMD
- The final result can be consumed by the target machine runtime.
- Many SyCL implementations are available



Summary

- Lots of options: OpenMP, Kokkos, RAJA, SyCL, HIP etc.
  - but will there be one that works well on all of Perlmutter, Aurora and Frontier?
- There are similarities, with differences between Kokkos, Raja, and SyCL
  - Express parallelism via functors/lambdas
  - Data Movement: Views v.s. Buffers, Explicit v.s. Implicit movement, Accessor Scope.
- Parallel features are also being incorporated into standard C++
  - parallel algorithms, pSTL, std::simd, etc…
  - Kokkos developers & others are members on the C++ standards committee
- I have had some very positive experiences with Kokkos
  - Performance was within 20% or so of hand tuned code on P100 (SummitDev) even better on Volta.
  - Performance matched hand tuned code on KNL - after manual vectorization of complex. Kokkos::simd will fix this(?)
- Need to repeat these experiments with OpenMP & SyCL
  - ongoing current work. I hope to have more results on this soon.
  - as always: your application's mileage may vary.

References

- OpenMP: https://www.openmp.org/
- Kokkos: https://github.com/kokkos/kokkos
- RAJA: https://github.com/LLNL/RAJA
- HIP: https://gpuopen.com/compute-product/hip-convert-cuda-to-portable-c-code/
- SyCL: https://www.khronos.org/sycl/
- Intel One API
- Performance Portability: https://performanceportability.org/

Acknowledgements

- I'd like to acknowledge funding from US DoE Office of Nuclear Physics, Office of High Energy Physics and the Office of Advanced Scientific Computing Research through the SciDAC Program (1,2,3, and 4) and from the Exascale Computing Project
- I'd like to acknowledge and thank all my collaborators who worked with me on the Kokkos port so far: Jack Deslippe, Thorsten Kurth, Daniel Sunderland, Dan Ibanez, and thank NERSC for supporting a month long sabbatical in 2017 to work on Kokkos.
- Thank the Kokkos team for all their help in particular: C. Trott. D. Sunderland, D. Ibanez, S. Rajamanickam
- I'd like to acknowledge the systems used for the data shown here, including NERSC Cori, OLCF Summit-Dev, JLab KNL and GPU clusters.