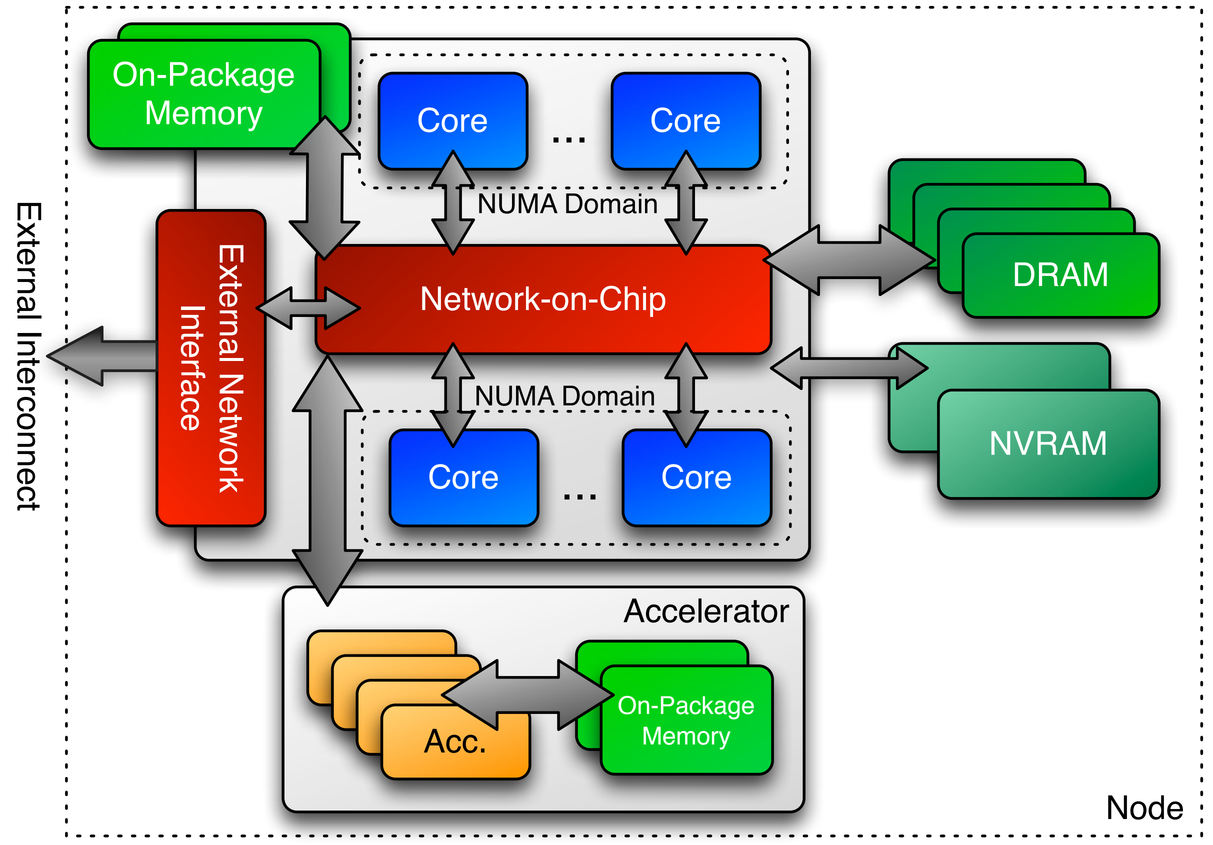
A QCD Performance Portability Study using Kokkos

Balint Joo - Jefferson Lab
T. Kurth, J. C. Deslippe - NERSC
D. Sunderland, D. Ibanez - Sandia National Laboratory
K. Clark - NVIDIA

IXPUG BOF, SC’17, Denver, Nov 14, 2017

Perf. Portability Study with Kokkos

- With hardware diversity and looking forward to new systems, performance portability is crucial.
- Kokkos is a C++ library providing parallel programming constructs, developed and maintained at Sandia National Lab. (see: github.com:kokkos/kokkos)
  - Node model: multicore CPUs, accelerator devices, several memory spaces
  - Patterns: parallel_for, reduction, scan, DAG/futures
  - Back Ends: OpenMP, CUDA, OpenMP-target, pthreads, …
- As Part of our ECP & NESAP work we investigated porting a Key QCD Kernel (Wilson Dslash) to Kokkos
  - work done mostly during 4 week visit to NERSC
  - and at a GPU Hackathon organized by OLCF & NASA
  - in collaboration with NERSC, NVIDIA and the Kokkos team



A schematic of a typical compute node modeled by Kokkos: Image from Kokkos Tutorial Slide Deck: KokkosTutorialARL2016.pdf

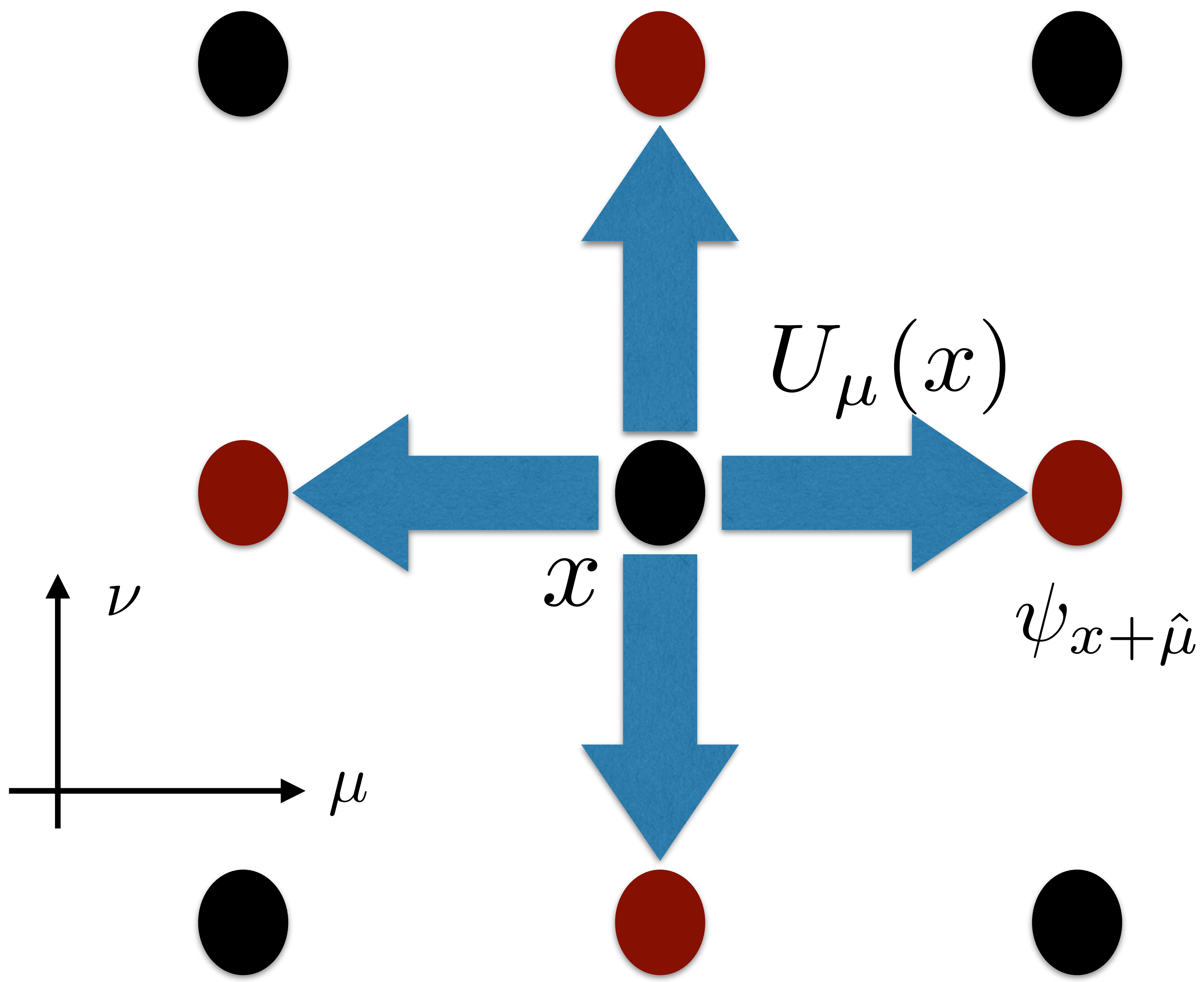
```
int N=…  // Some large number
View<double[N]> x("x");
View<double[N]> y("y");
double  a = 0.5;

parallel_for(N,
     KOKKOS_LAMBDA(const size_t i) {
              y(i) += a*x(i);
          }
);
```

A basic example of a Kokkos Code using Views, parallel_for and C++ 11 lambdas

The Wilson Dslash Operator

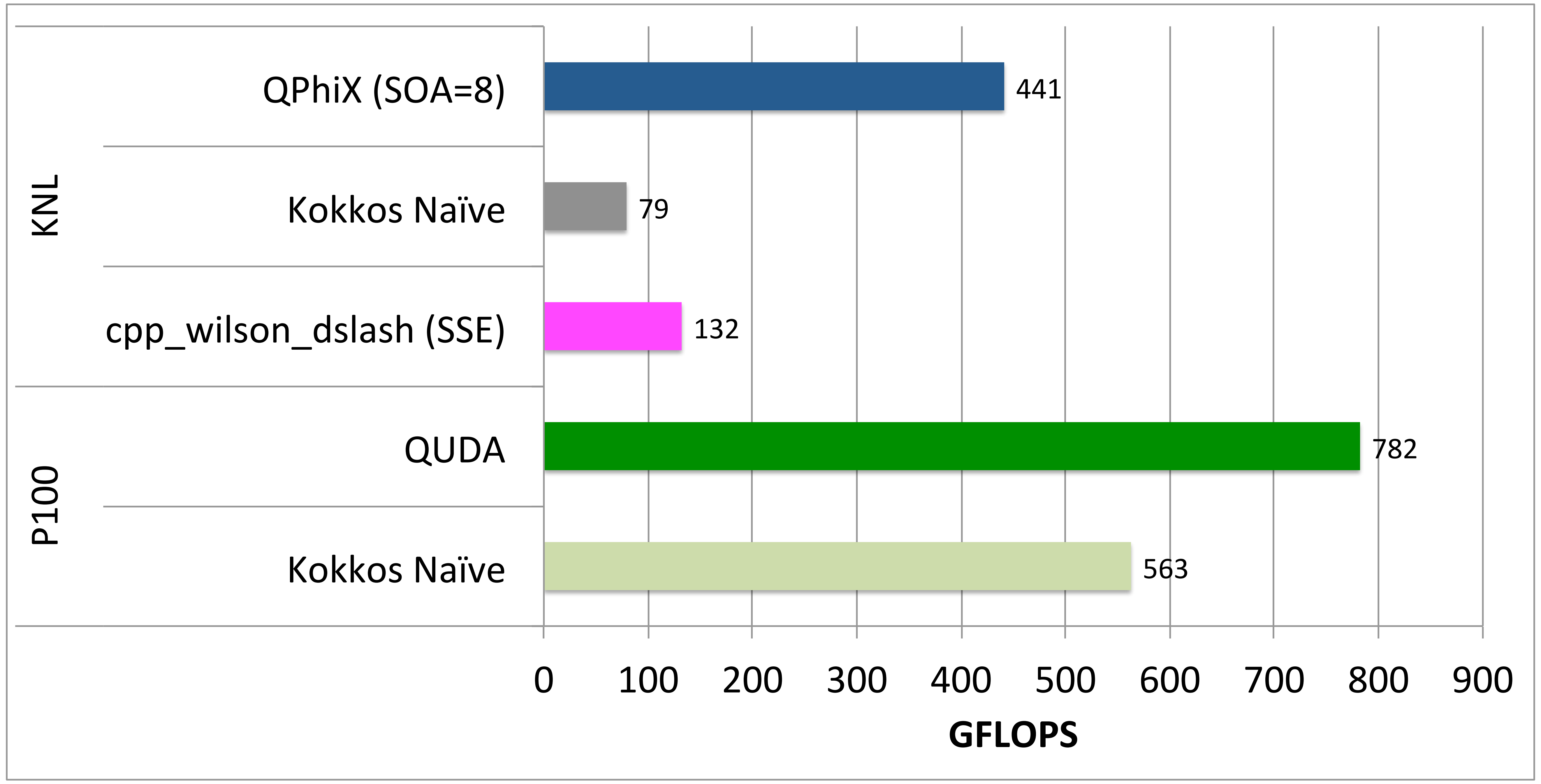
- 4D Sparse Matrix-Vector Operator: $\chi = \not{D}\, \psi$ (SRHS)
- For each neighboring site
  - project 4x3 spinor on the site to 2x3-spinor
  - multiply spinor components by a 3x3 matrix (2 multiplies)
  - reconstruct 4x3-spinor result and accumulate
- Both spinors and 3x3 matrix use complex numbers
- 3x3 matrices and vectors: **tricky to vectorize**
  - Vectorize over lattice sites: BFM, QPhiX, BNL-code…
  - Or over RHS vectors for: $\chi_i = \not{D}\, \psi_i$ i=1,2,…. (MRHS)
- MRHS allows reuse of gauge fields
  - useful even if one vectorizes over the sites (BNL, QUDA)



$$\not{D}_{x,y}\, \psi_y = \sum_\mu (1-\gamma_\mu)\, U_\mu(x)\, \psi_{x+\hat{\mu}} + (1+\gamma_\mu)\, U^\dagger_\mu(x-\hat{\mu})\psi_{x-\hat{\mu}}$$

Initial Kokkos Results (SRHS)

- KNL
  - Performance was very low
  - Even lower than previous legacy codes
  - ***Reason: no (auto) vectorization***
    - 3x3 matrices, 3 vectors
- GPU
  - Initially large amount of register spill to local memory
  - Needed to adjust CUDA launch bounds for kernels (Kokkos::LaunchBounds policy) - Kokkos team has since fixed this default
  - After this performance was pretty good

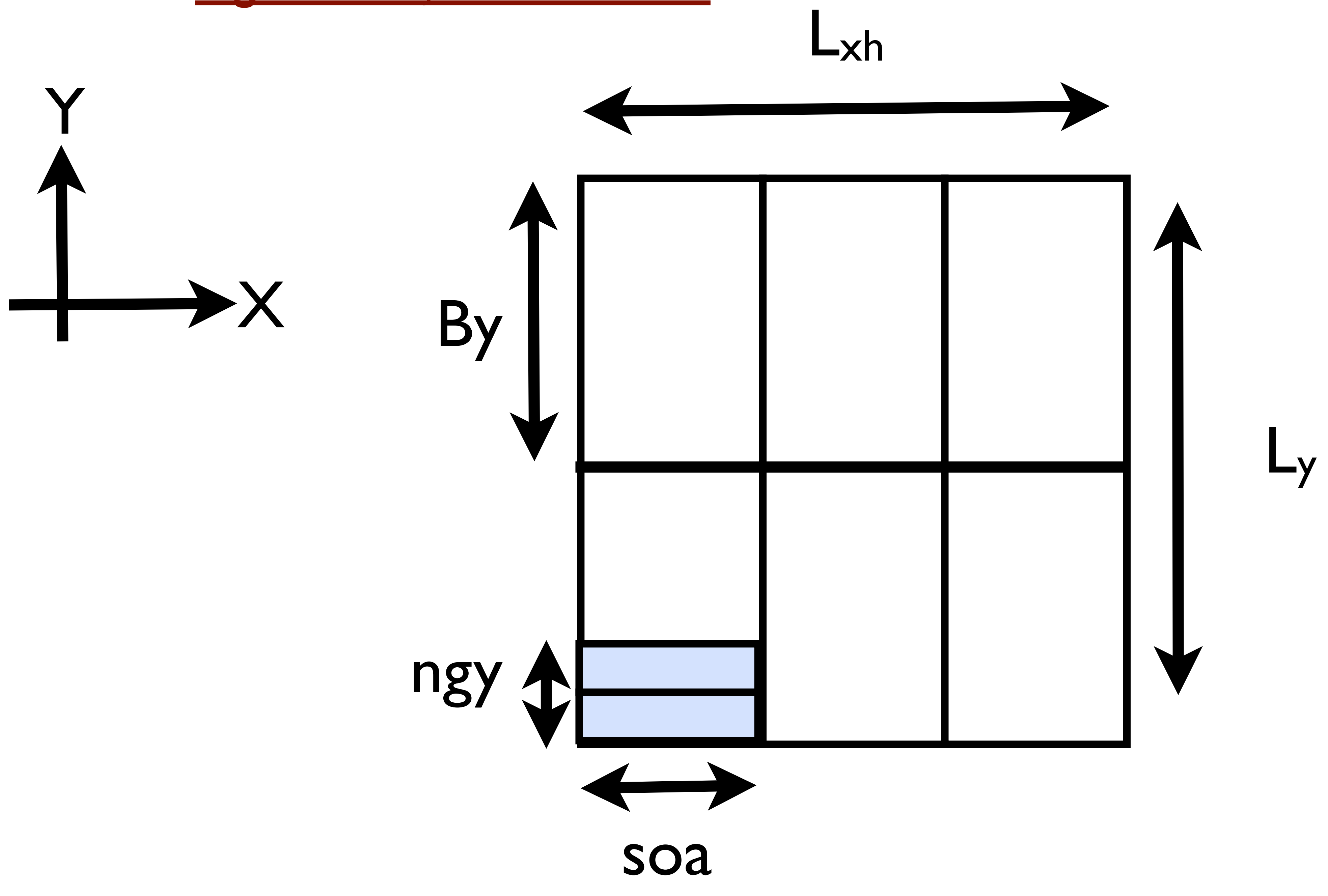


Single RHS Dslash Performances: Vol=32x32x32x32 sites

Site-wise vectorization schemes

X-Y Tiling, e.g. QPhiX (idea by D. Kalamkar)

e.g. B. Joo, et. al. ISC'13

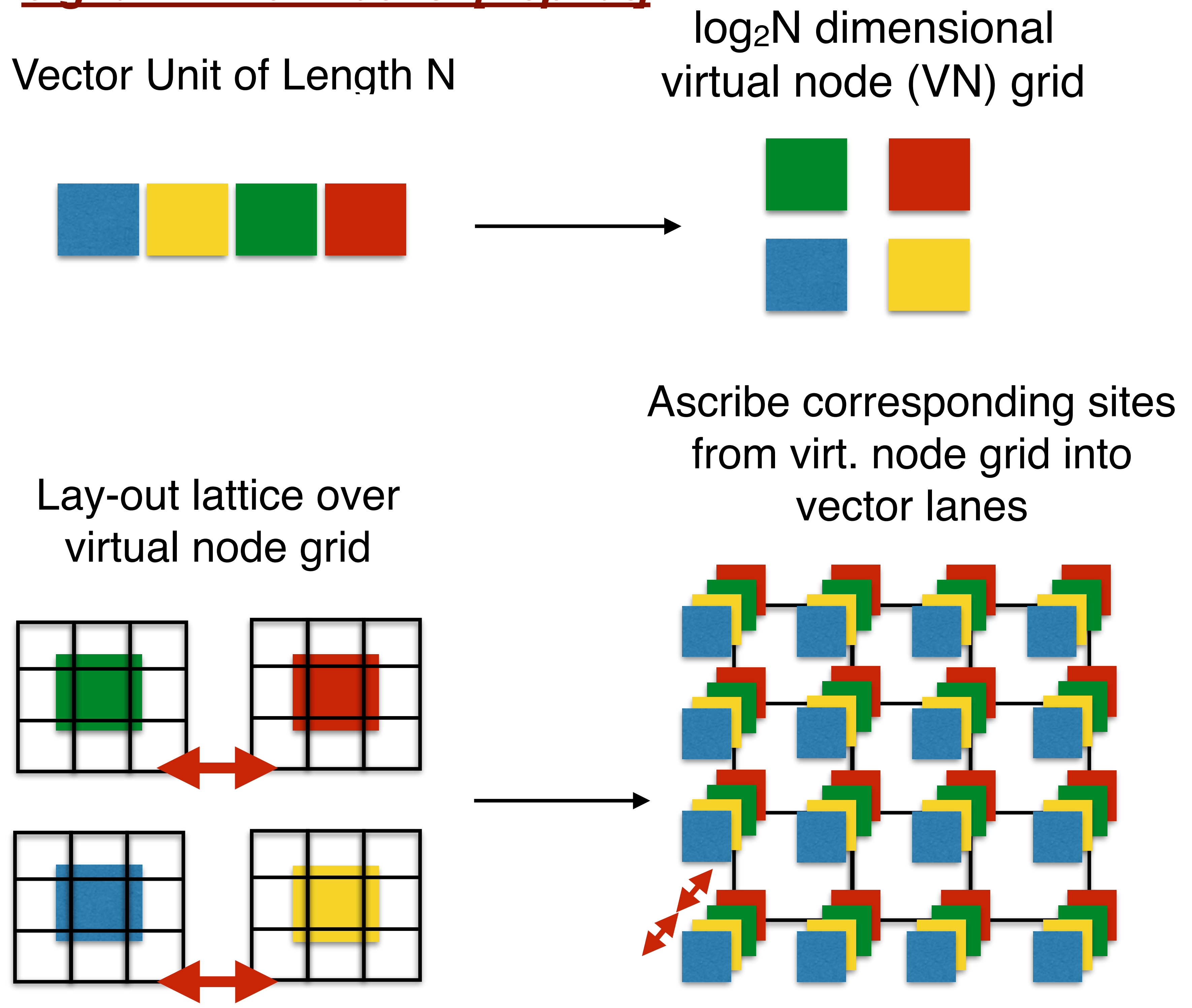


Assemble full Vector from ngy x soa pieces:

- *e.g. ngy=4, soa=4, ngy=2 soa=8, or general gather*
- *unaligned loads for some neighbors in x-y plane*
- *user now has to choose soa to suit problem*

Virtual Node Vectorization (P. Boyle, e.g. in Grid, BFM)

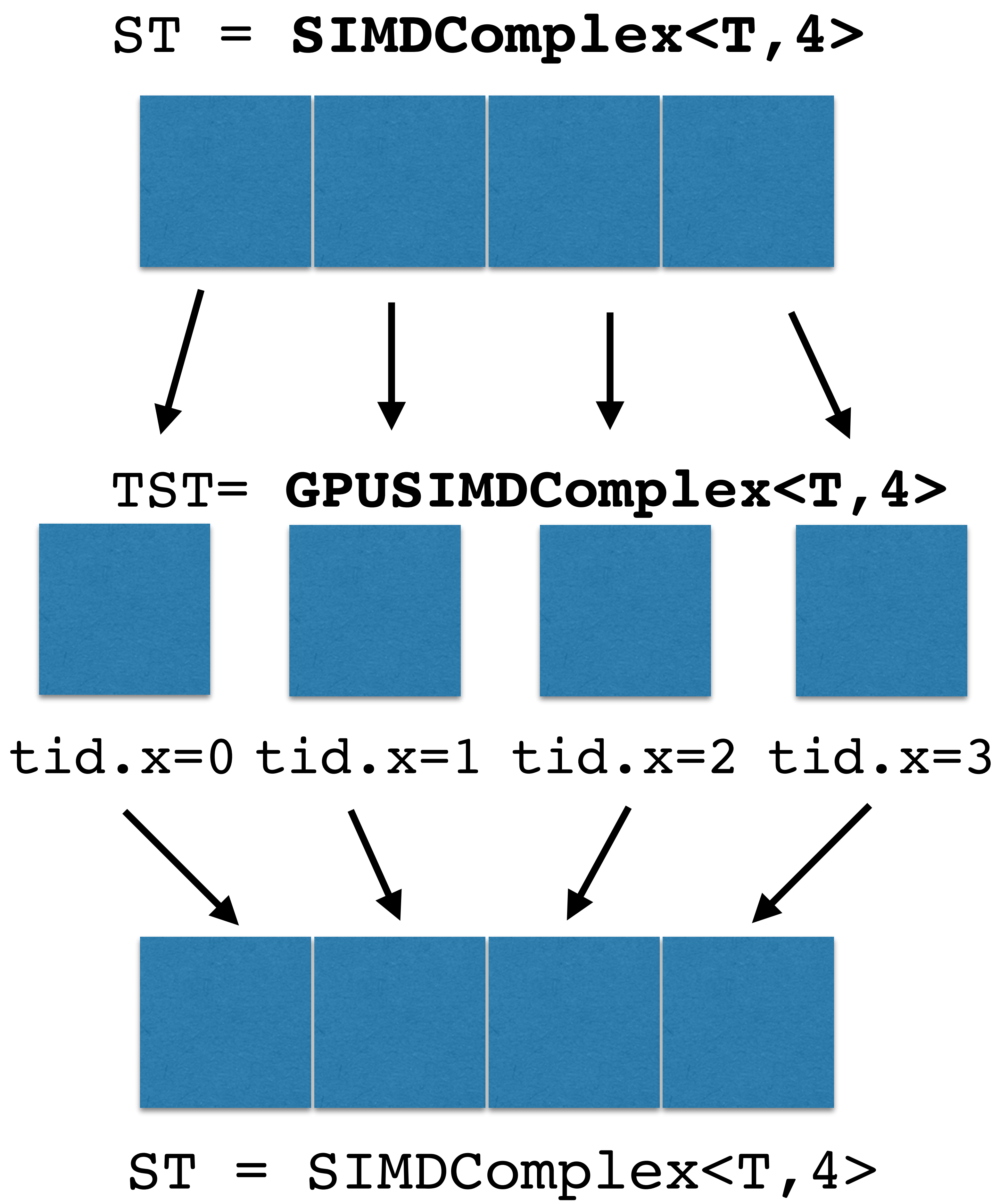
e.g. arXiv:1512.03487[hep-lat]



Nearest neighbor from other VN :
data from other lane => lane permutations

Vectorization And GPUs

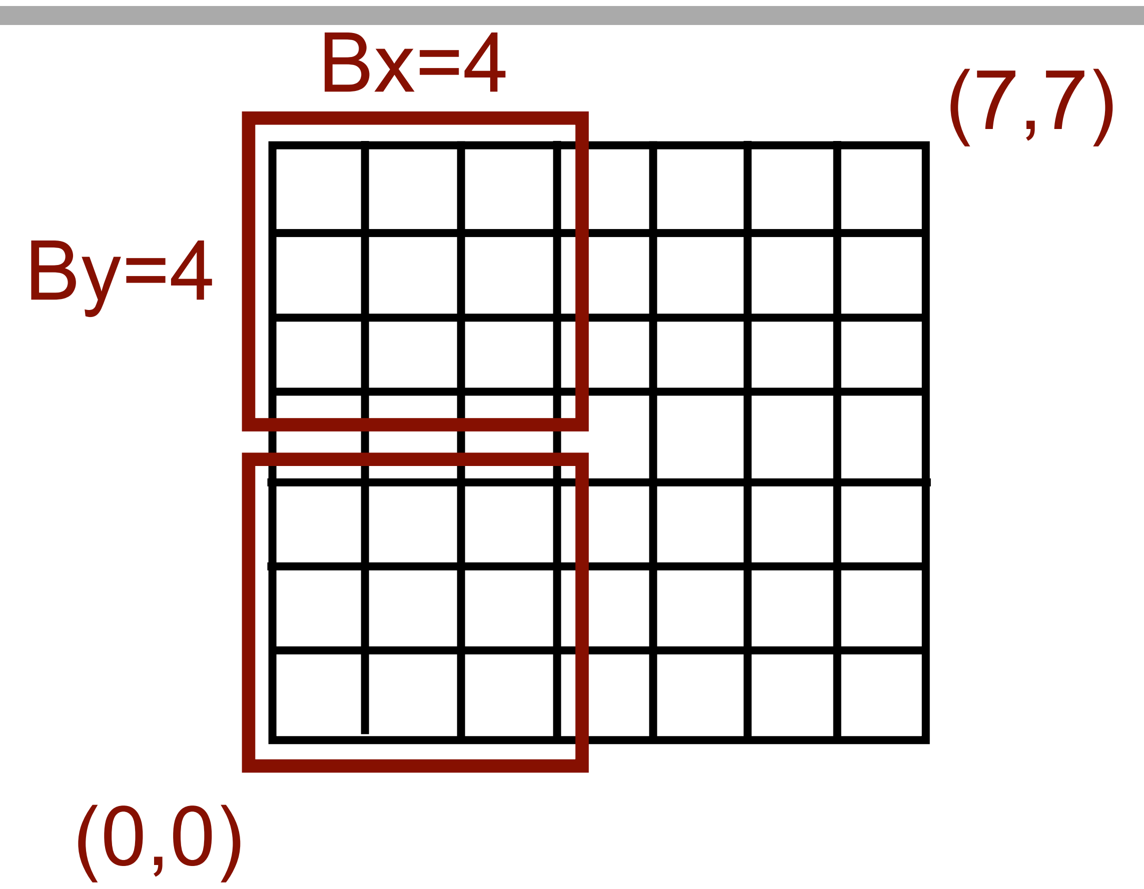
- What does it mean to `vectorize' for GPUs ?
- Approach 1: Use warp threads as vector lanes
  - Kokkos implements with "ThreadVectorRange"
    - x-dimension of thread block for CUDA
    - loop + maybe #pragma ivdep fo OpenMP
- Approach 2: GPU Vector Length is 1
  - Vector code reduces to unvectorized on GPU
  - This is fine, since 'unvectorized' was already good.
- Implemented Approach 1 for MRHS case
- Implemented Approach 2 for SRHS case
  - used 'Grid'-like Virtual Node approach



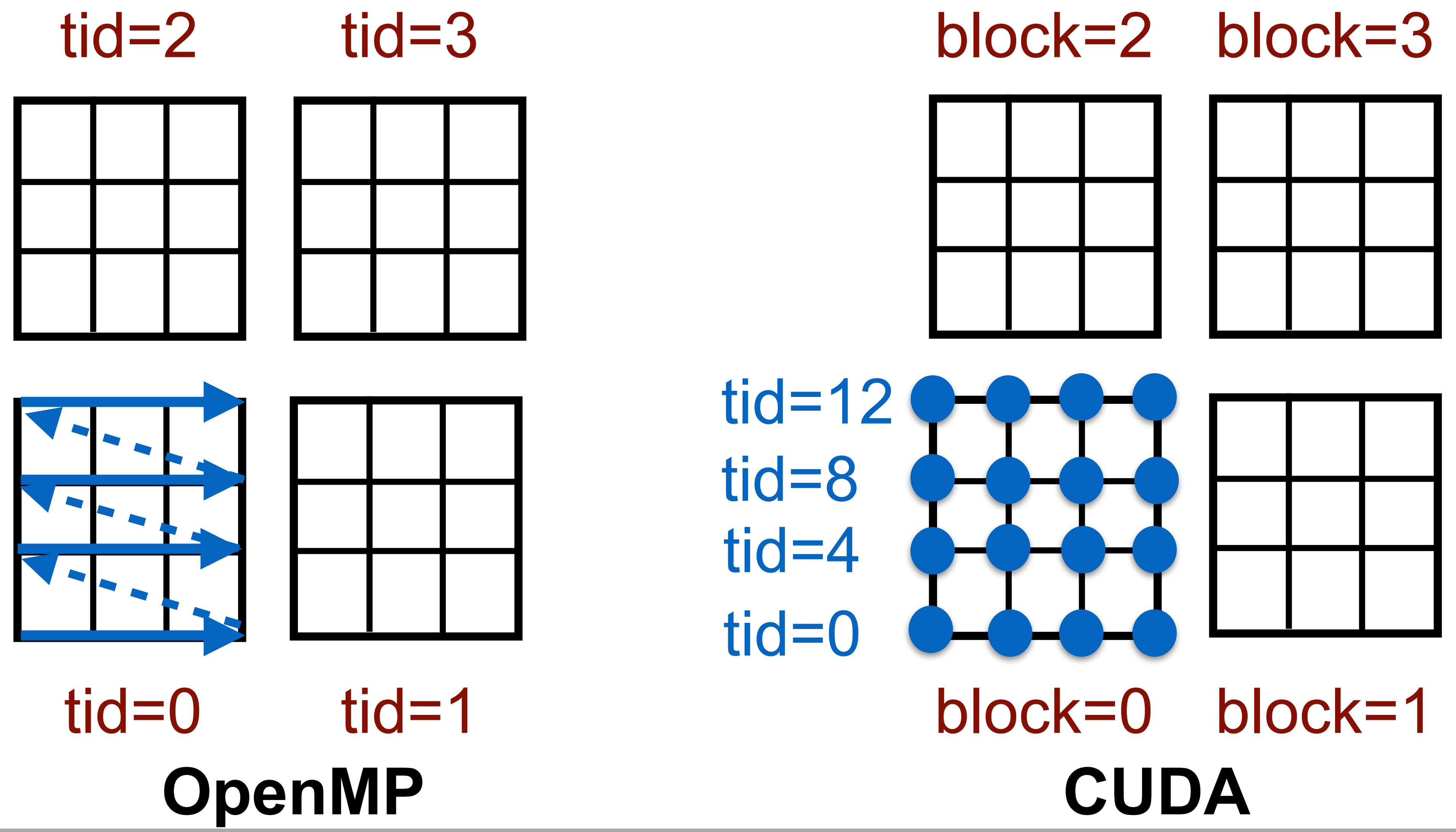
Schematic of Approach 1 on GPU: TST is a type for thread-local objects.

Other Optimizations

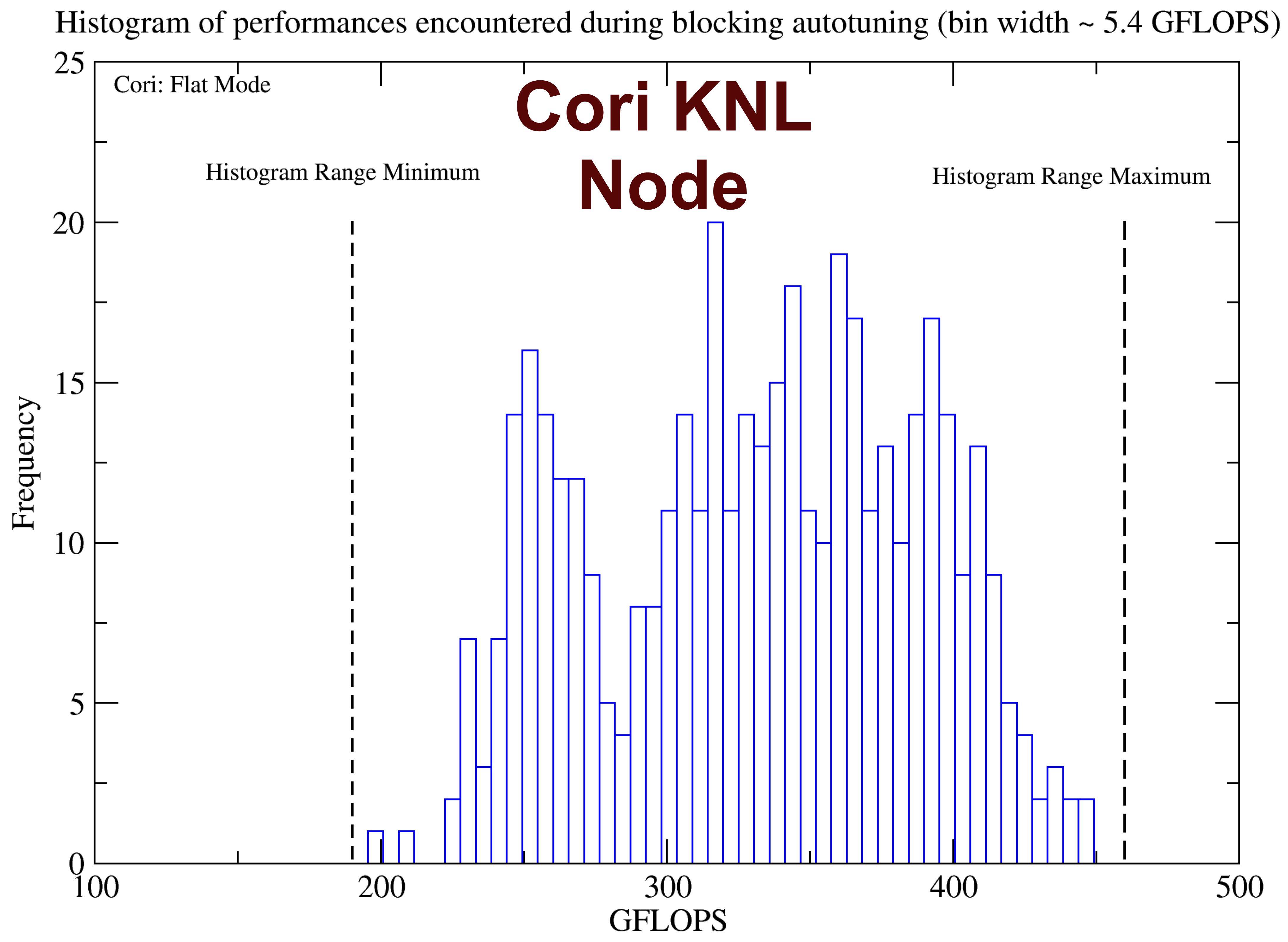
- Vectorize VN permutes using AVX512 intrinsics
  - using `_m512_permutexvar_ps()`
- spin <-> color loop interchange (help L1 locality)
- 4D Blocking using Kokkos::MDRange exec. policy
  - tune block size for performance
- Gauge Field Access:
  - keep copies of back pointing links => unit stride access for gauge fields
  - KNL: pre-permute links from back neighbor: no gauge permute in Dslash
- GPU: derived complex number class from float2

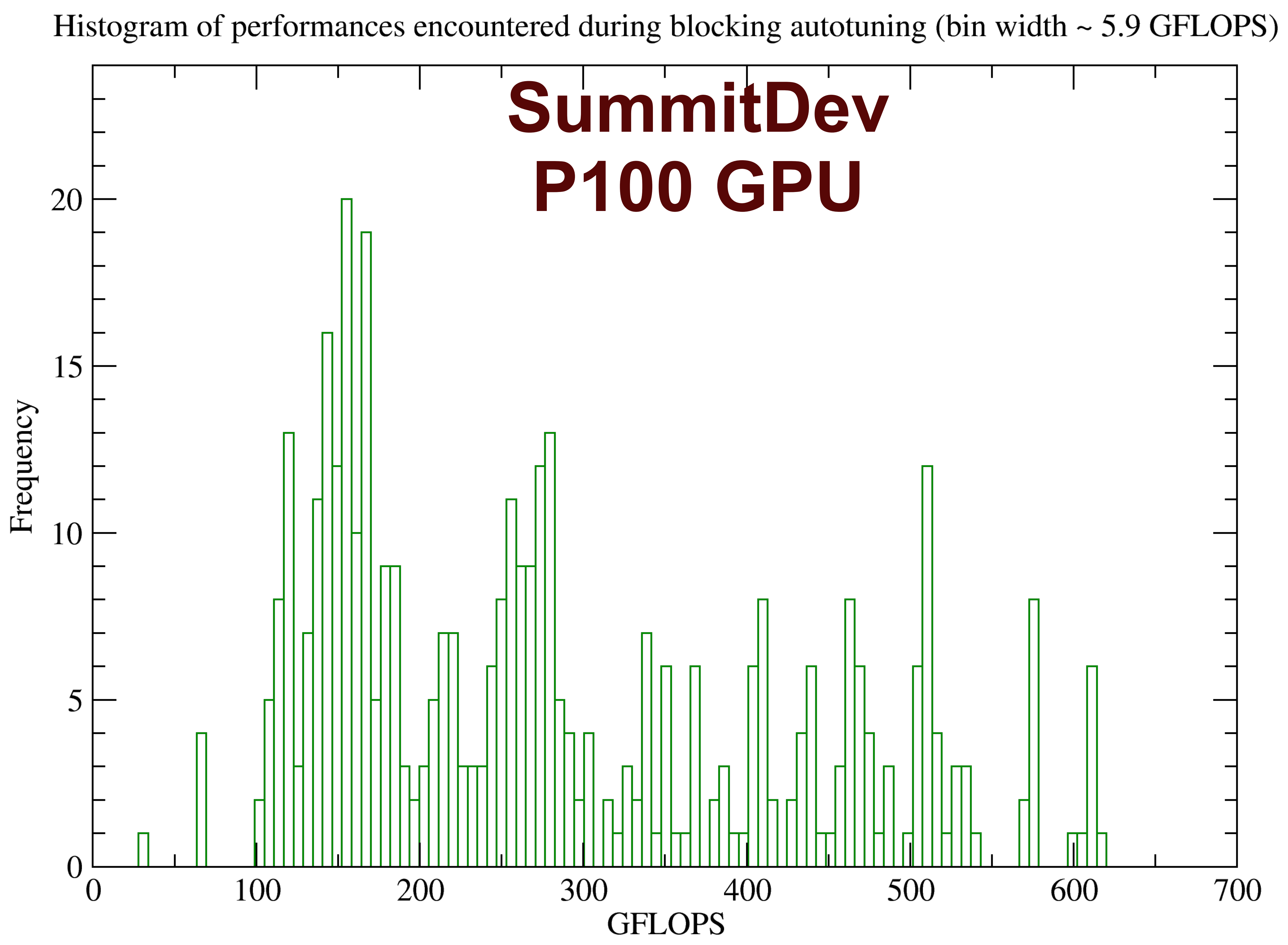



```
MDRangePolicy<2,IterateLeft,IterateLeft>
                    policy({0,0},{8,8},{4,4});
```



Block Tuning for MDRange

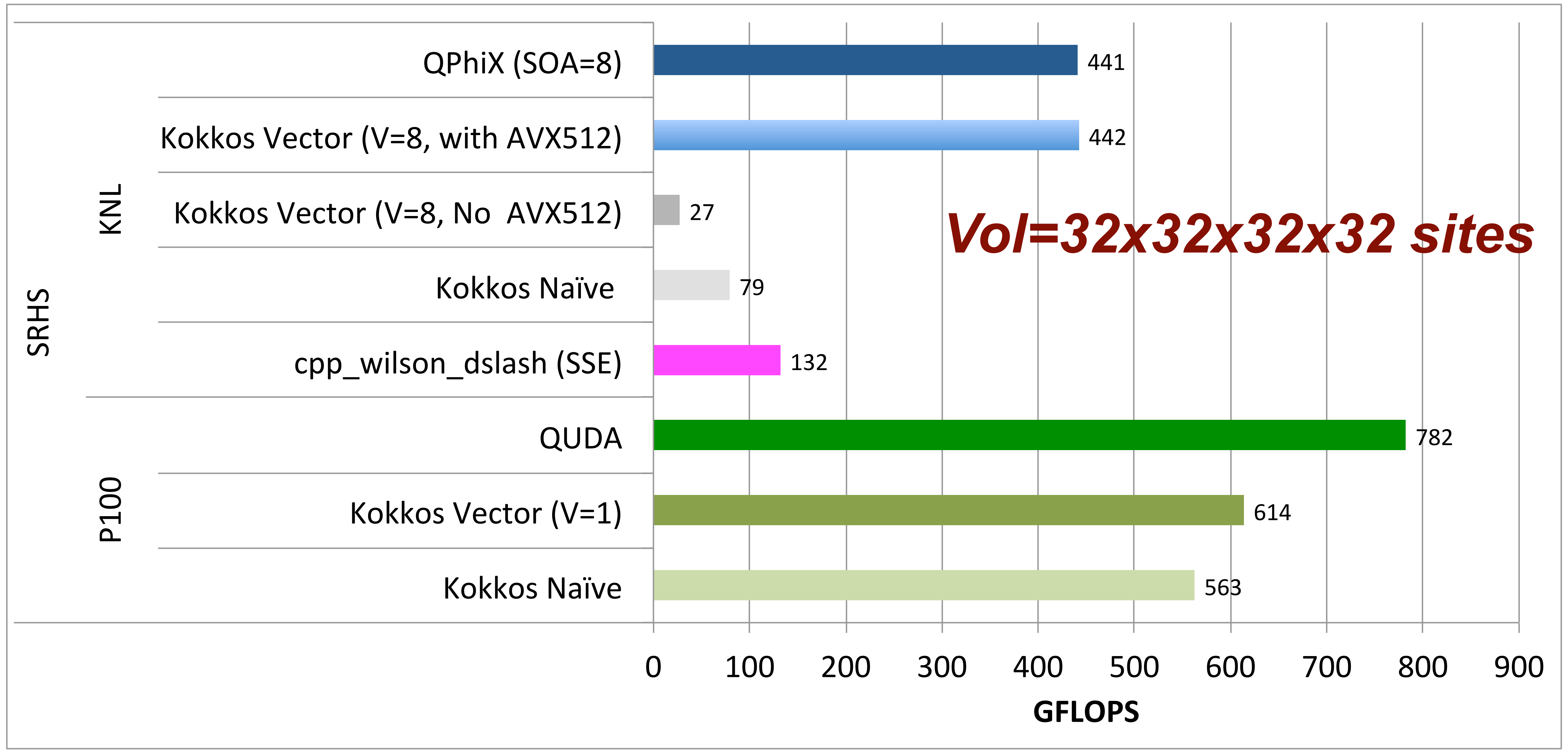


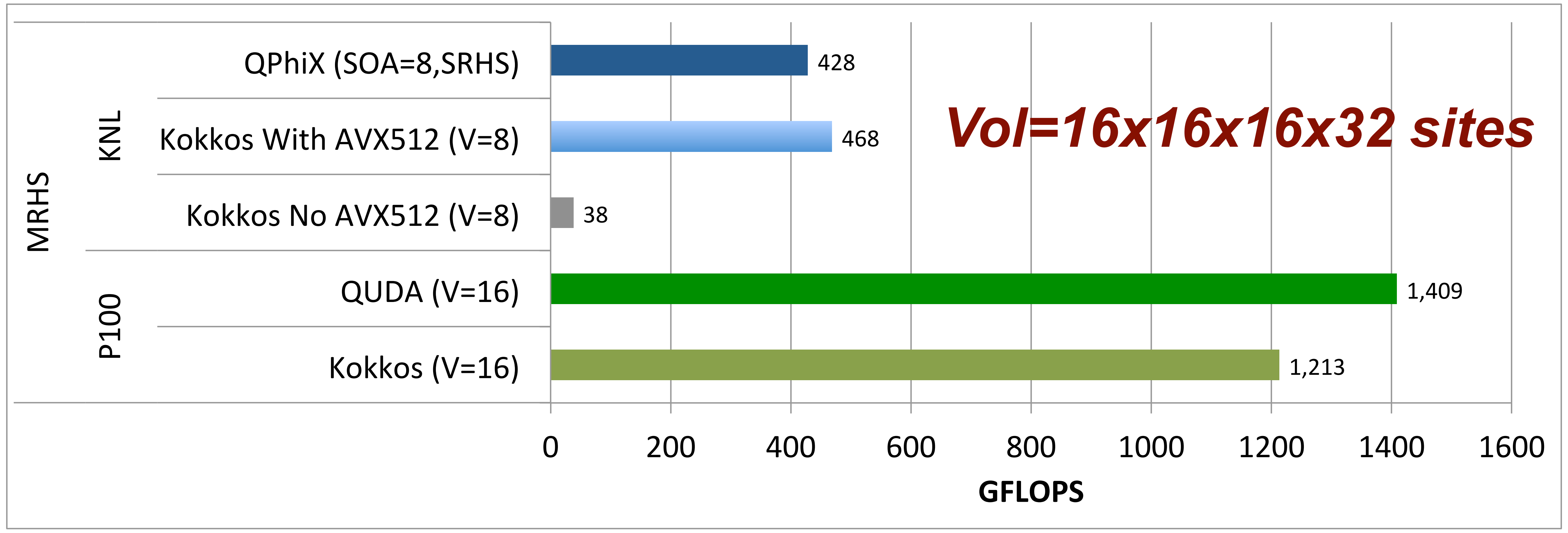


- Block Tuning gave broad performance distributions, with few (<10) tunings giving the highest performance. Autotuning is a must & unfortunately the space is big (4D).

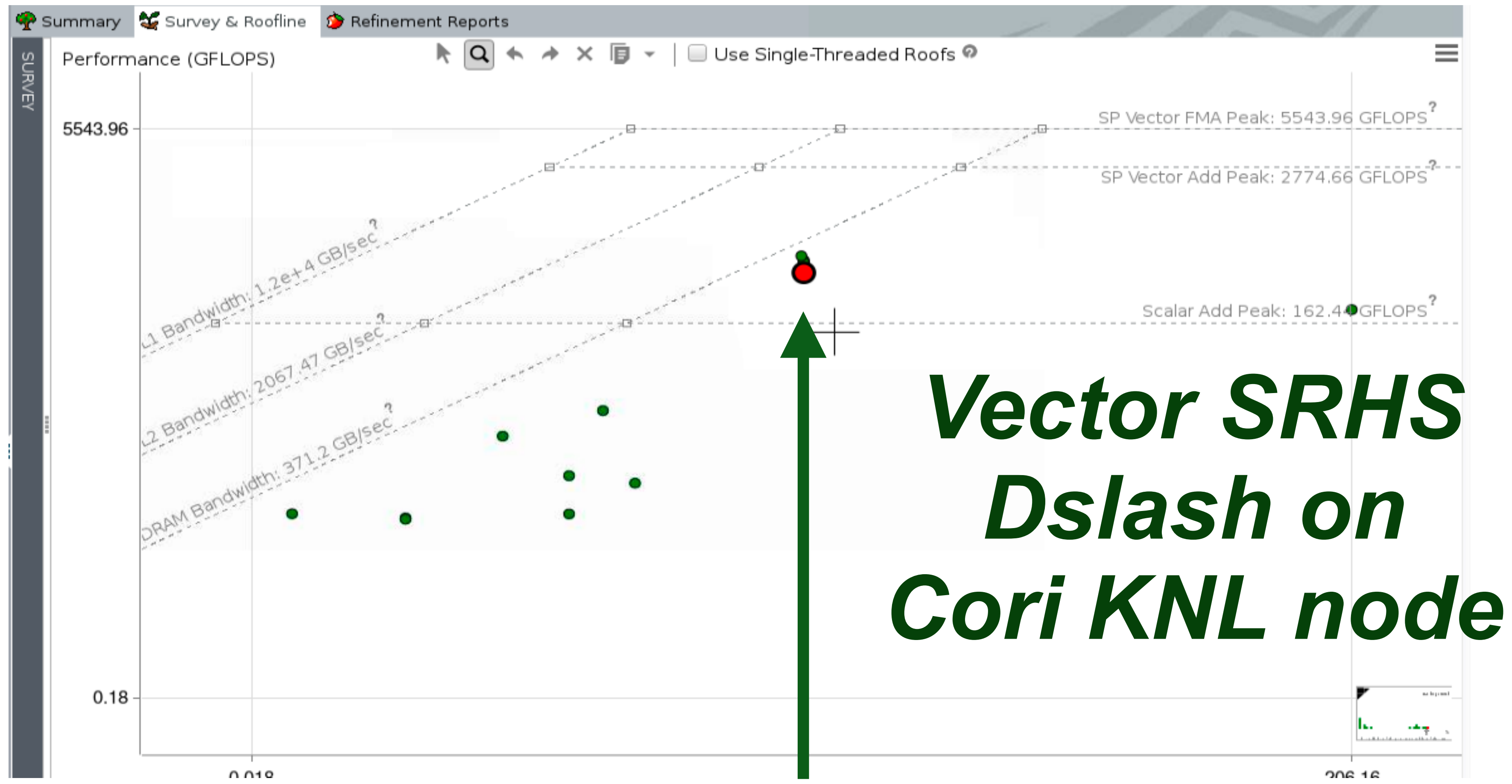
Current Performance Summary

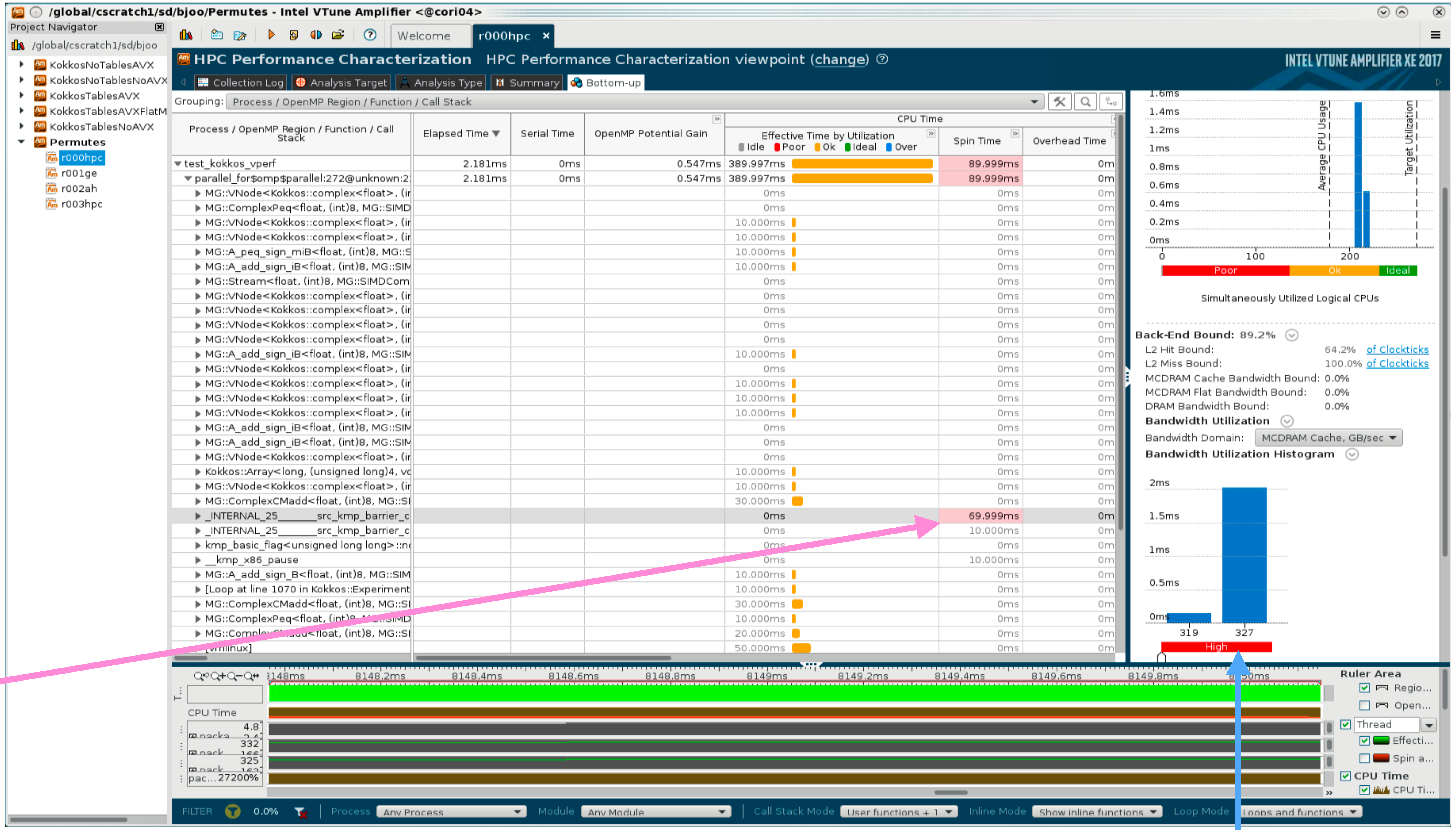
- SRHS Case:
  - ***Kokkos Vectorized Dslash with AVX512 and tuned blocking matches QPhiX on Cori KNL node (68 cores, 272 threads)***
  - Unvectorized & No AVX cases are slow
  - Kokkos Naive CUDA version is 72% of QUDA on P100 (SummitDev)
  - ***Vectorized (but V=1) QUDA version benefits from block tuning, memory & locality optimizations and md_parallel_for: 79% of QUDA on P100 (SummitDev)***
- MRHS Case:
  - ***Kokkos With AVX512 exceeds corresp. QPhiX SRHS performance on Cori KNL node for 8 RHS***
  - Kokkos Without AVX512 is very slow
  - ***Kokkos CUDA version is 86% of QUDA for 16 RHS on SummitDev (P100)***





Absolute Performance is good too

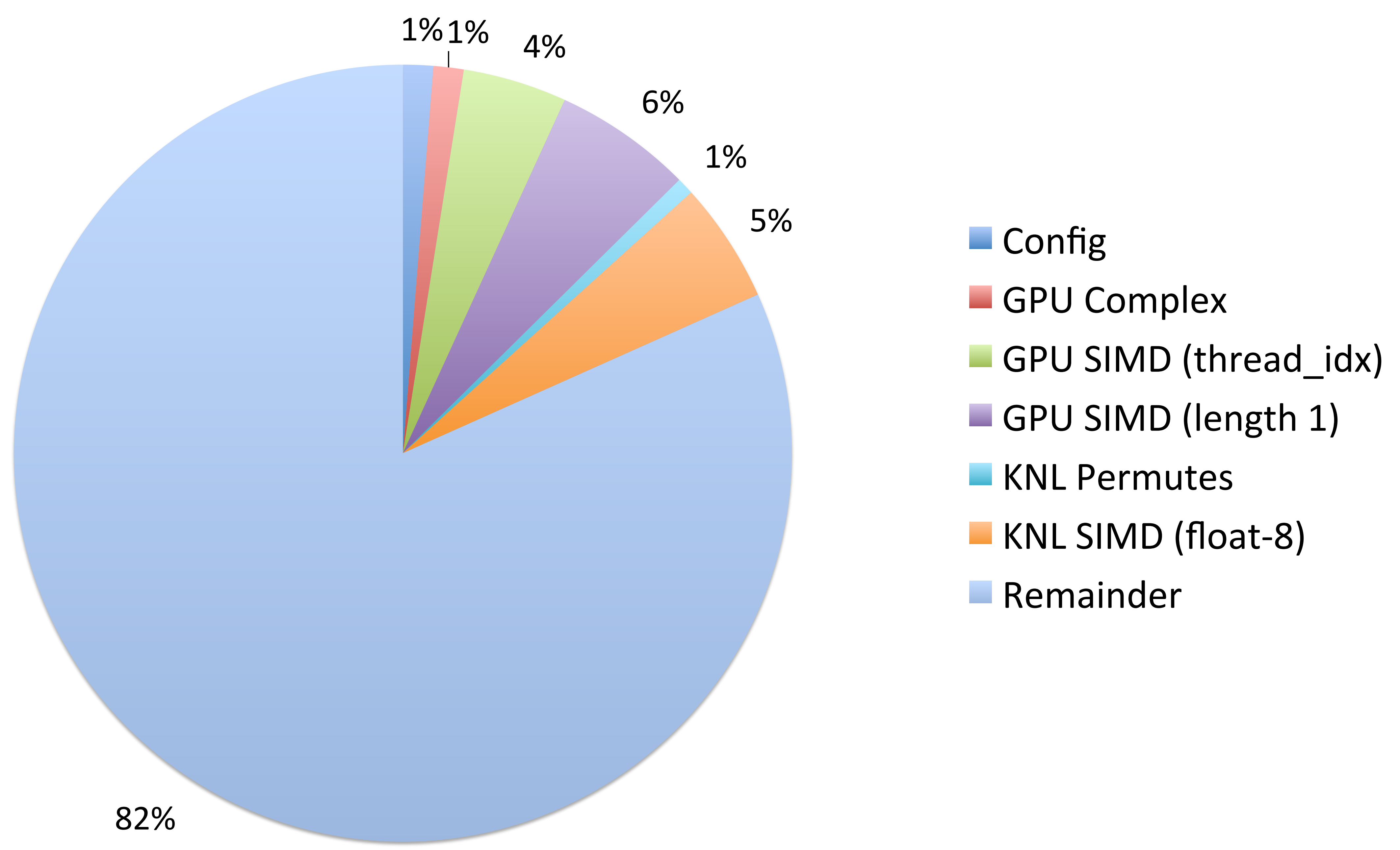




- Little Below Roofline. Reported L1 AI=1.97
- Vtune is concerned with spin wait in barrier
- Good CPU usage and mem BW Usage

How much Non-portable stuff is there?

- Total code is 4545 lines of code
  - excluding unit tests which bring it to 7100
- KNL Specific (AVX512) : 274 lines
  - permutes for Virtual Node 34 lines
  - complex ops for float-8 SIMD: 240 lines
- GPU Specific:
  - Derive complex from float2: 60 lines
  - Warp SIMD type & ops (threadIdx.x): 207 lines
  - Length=1 SIMD ops: 275 lines (repeat of scalar)
- Select GPU/CPU Layout, types etc: 60 lines



Line counts are 'approximate'
Carried out by outlining machine specific parts and using SLOCCount

Summary and Conclusions

- Kokkos freed us from the need to deal directly with OpenMP pragmas, CUDA etc
- It did not save us from having to optimize.
- On KNL we had to vectorize our complex arithmetic and permutations with intrinsics
  - ~274 LOC out of ~4500 and we would need to repeat for double prec, other AVX flavors, etc.
  - Kokkos implementing an optimized complex SIMD type could move this burden to Kokkos
- We needed a performance portable algorithm
  - a vector friendly algorithm, i.e. MRHS or Virtual Node vectorization over sites
  - on GPUs use Warp threads as Vector Lanes or have GPU Vector length = 1.
- Excellent performance on both KNL and GPU after optimzations and block tuning.
- Future work: Parallelize over nodes with MPI, interface with Trilinos, …

Acknowledgments

- Jefferson Lab is operated by Jefferson Science Associates LLC under U.S. DOE Contract No. DE-AC05-06OR23177
- B. Joo gratefully acknowledges funding from the U.S. Department of Energy, Office of Science, Offices of Nuclear Physics, High Energy Physics and Advanced Scientific Computing Research under the SciDAC programs: SciDAC, SciDAC 2, SciDAC 3 and SciDAC 4.
- B. Joo gratefully acknowledges funding from the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under the USQCD Exascale Computing Project (Lattice QCD)
- B. Joo gratefuly acknowledges travel funding from NERSC for a summer Affiliate Appointment for work on Kokkos.
- The 2017 ORNL Hackathon at NASA was a collaboration between and used resources of both the National Aeronautics and Space Administration and the Oak Ridge Leadership Computing Facility at Oak Ridge National Laboratory. Oak Ridge Nation Laboratory is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.
- We gratefully acknowledge use of computer time at JeffersonLab (SciPhi XVI cluster), K80 Development node, NERSC Cori and OLCF SummitDev.