Clover Miscellany
(Random ramblings)

Bálint Joó, Jefferson Lab

USQCD All Hands 2019, Brookhaven National Lab

Contents

- (Clover) Gauge Generation
  - Multi-Grid in HMC using QUDA
  - Old Fashioned Hasenbuschery: rebooted (?)
- (Clover) Propagator calculations with MG-Proto on KNL, SKX, …
  - OpenMP and fine grained parallelism
- Software for the future:
  - End of the line for ‘conventional’ clusters?
  - What programming model to use for performance portability?
  - One suggestion to C++ folks: Use Kokkos for on-node parallelism

Big Shout Out to all involved!

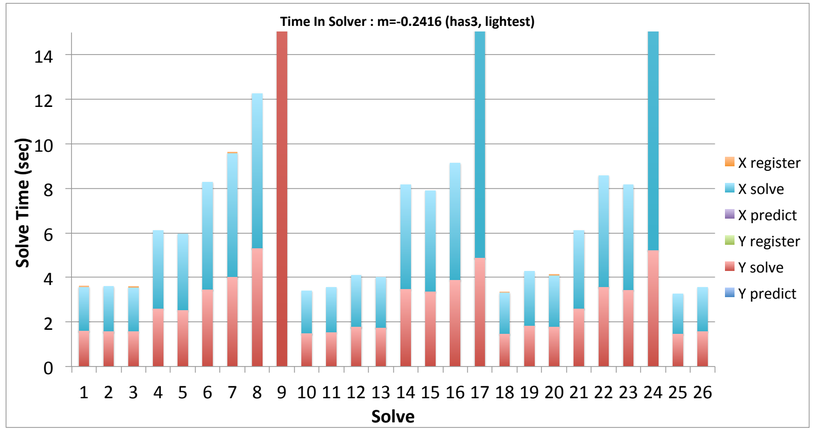
- Kate Clark, Evan Weinberg, Mathias Wagner (NVIDIA) for QUDA!!
  - Especially Kate also for algorithmic directions, MG support and MG in the gauge generation
- Frank Winter (JLab) for his work on QDP-JIT
- QPhiX collaborators over the years: M. Smelyanskiy (now Facebook), D. Kalamkar (Intel) K. Vaidyanathan (Intel), T. Kurth (NERSC), M. Ueding (U. Bonn), B. Kostrzewa (U. Bonn), P. Labus (U. Bonn), C. Urbach (U. Bonn), D. Deb (RENCI)
- Arjun S. Gambhir (now LLNL) for his work on QUDA Multigrid Integration.
- Boram Yoon (LANL) for his work on the FG Integrator
- OLCF (especially R. Budiardja, J. Hill) for SummitDev & Summit Use
- J. Deslippe, T. Kurth, R. Gerber (NERSC) & NESAP for a nice long visit to NERSC to work on Kokkos.

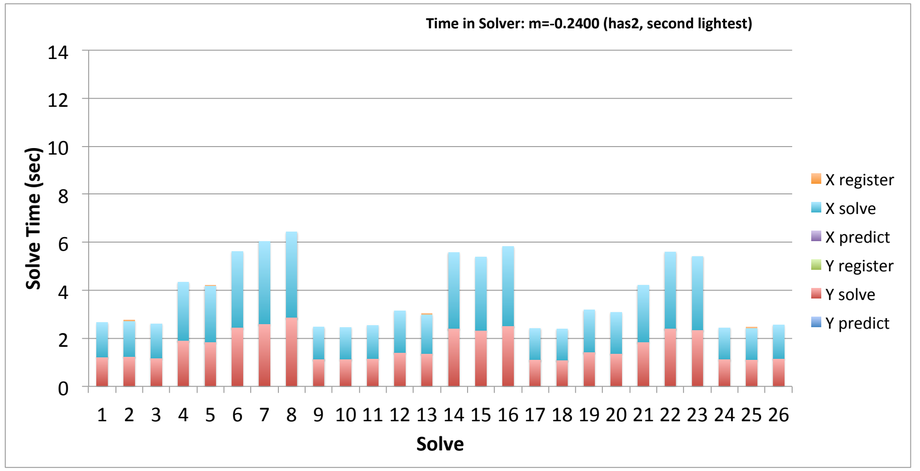
Multigrid in the HMC

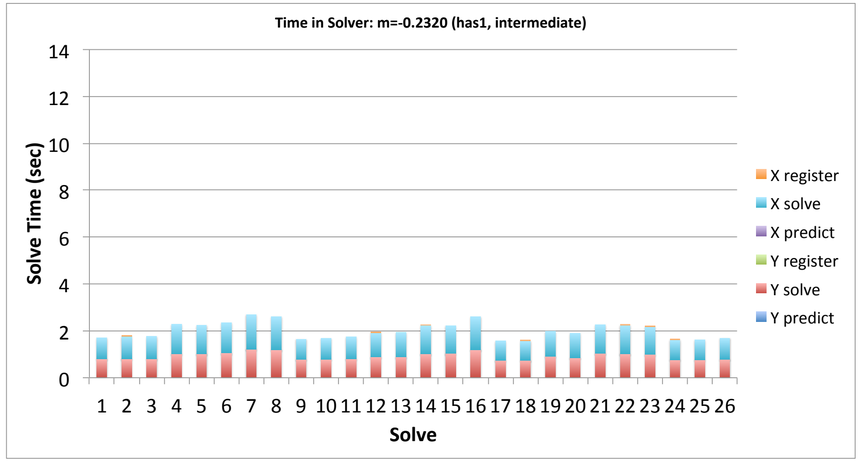
- We focus on the MG implementation in QUDA hooked into Chroma
- Primary Targets: Titan, Summit, Blue Waters and any other big GPU based system we can use
- MG Subspace Management
  - Keep Subspace in Chroma Named Object Store (reuse for solves with several masses)
  - Still need to recompute coarse operators: this must be fast
  - Subspaces lose effectiveness as fields evolve => Refresh when Iteration Threshold reached
  - Keep costs of Subspace Refresh low: use fixed number of iterations (rather than residuum)
  - Two new tunable parameters: Iteration Threshold & Refresh Iterations
- Numerical Experiments: 64^3x128 lattice (a~0.092fm, mpi ~ 172 MeV)
  - Production Isotropic Lattice

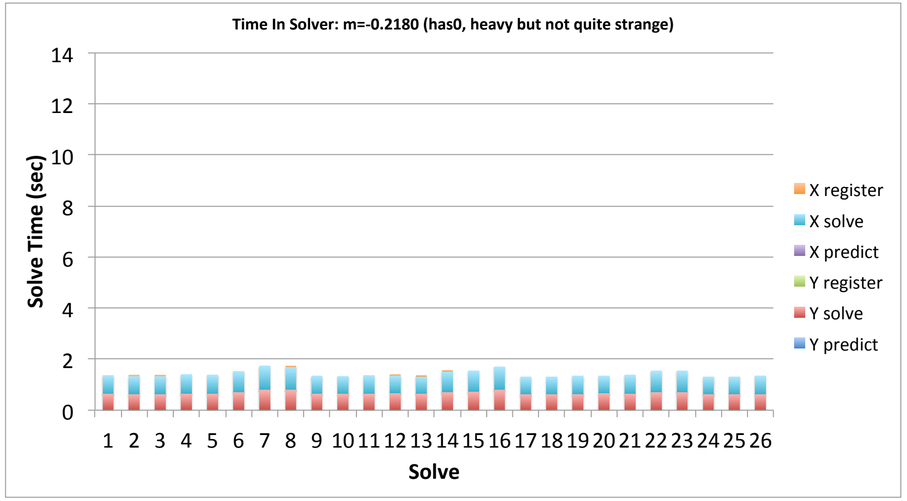
Multigrid Behavior

- Only the lightest mass really matters — as expected. And as shown before by Meifeng the preconditioner doesn't degrade much for heavier masses

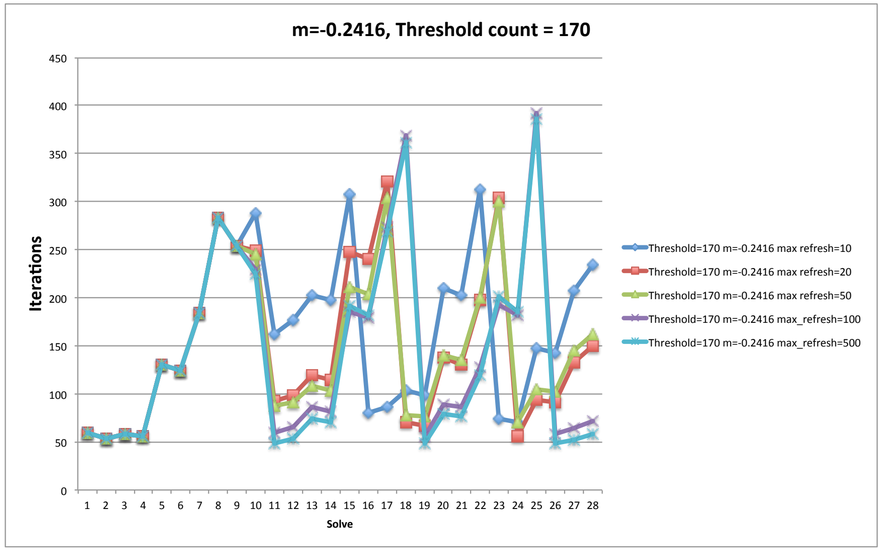


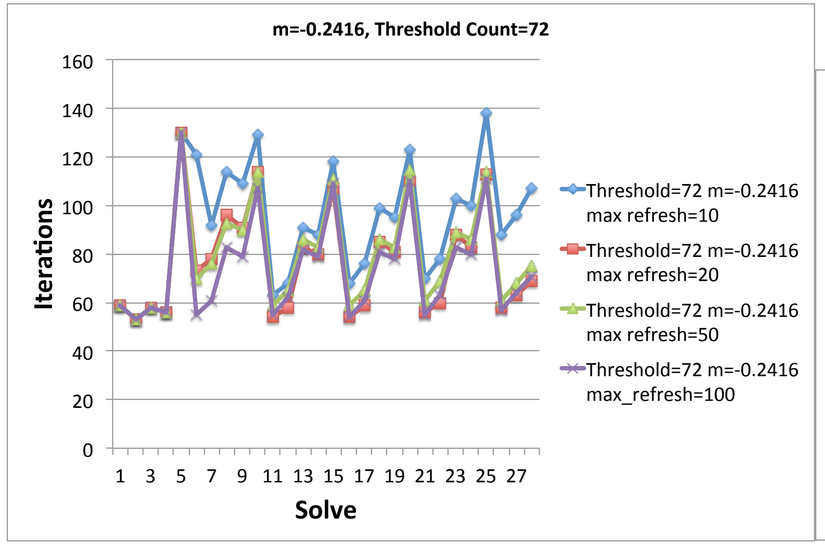


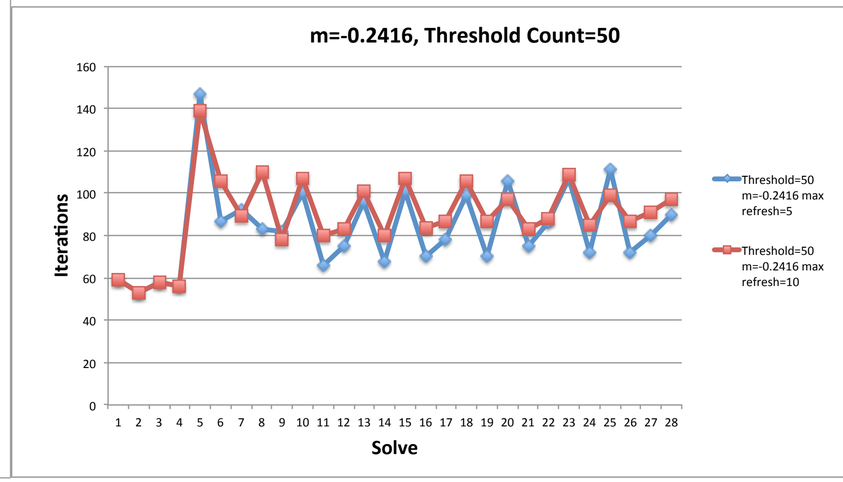




Multigrid Tuning

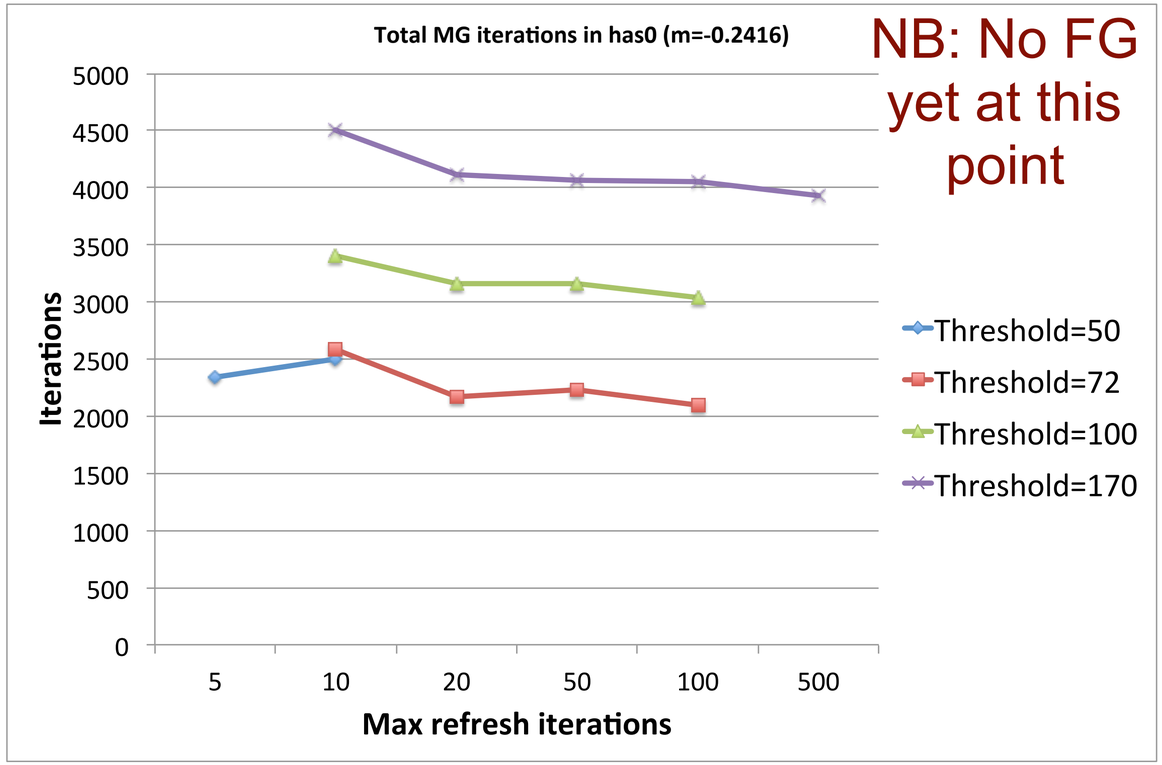


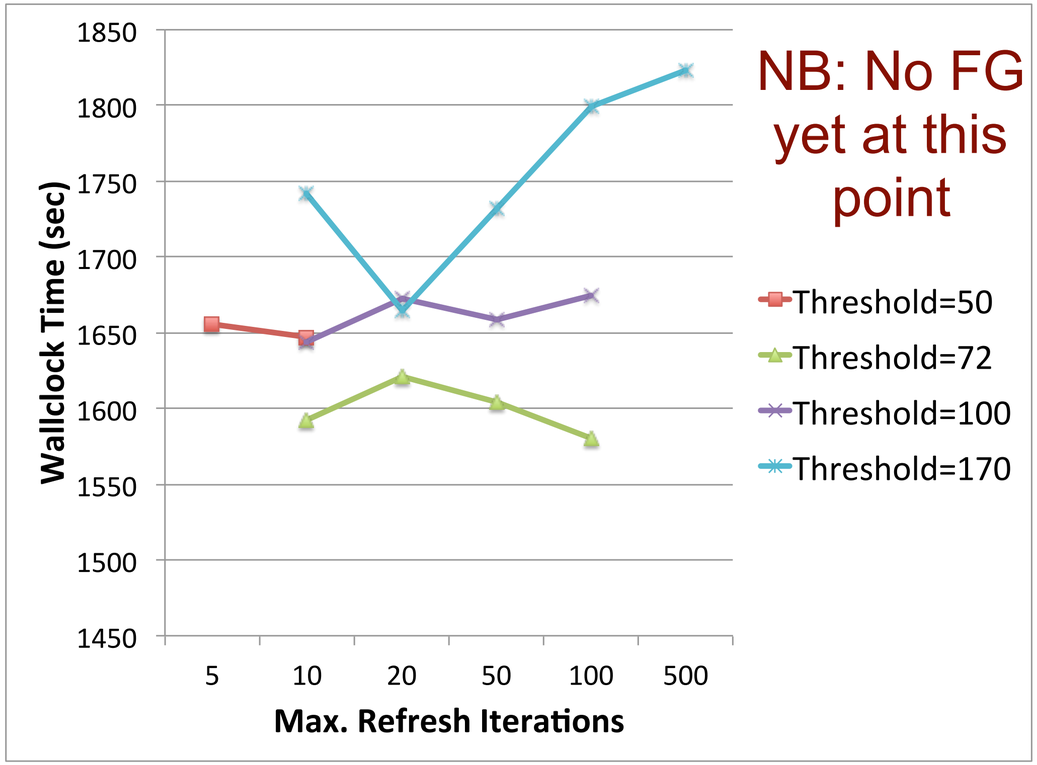




- Threshold count effectively regulates frequency or refreshes
- Maximum number of refresh iterations controls cost of refreshes
  - But the refreshed subspace may not be good enough if too few refresh iterations are used.
- NB: Until first refresh, subspace is made with wrong mass
  - created by 2nd lightest mass in PF refresh of lightest Hasenbusch field, but seemingly not a big deal

Multigrid Tuning

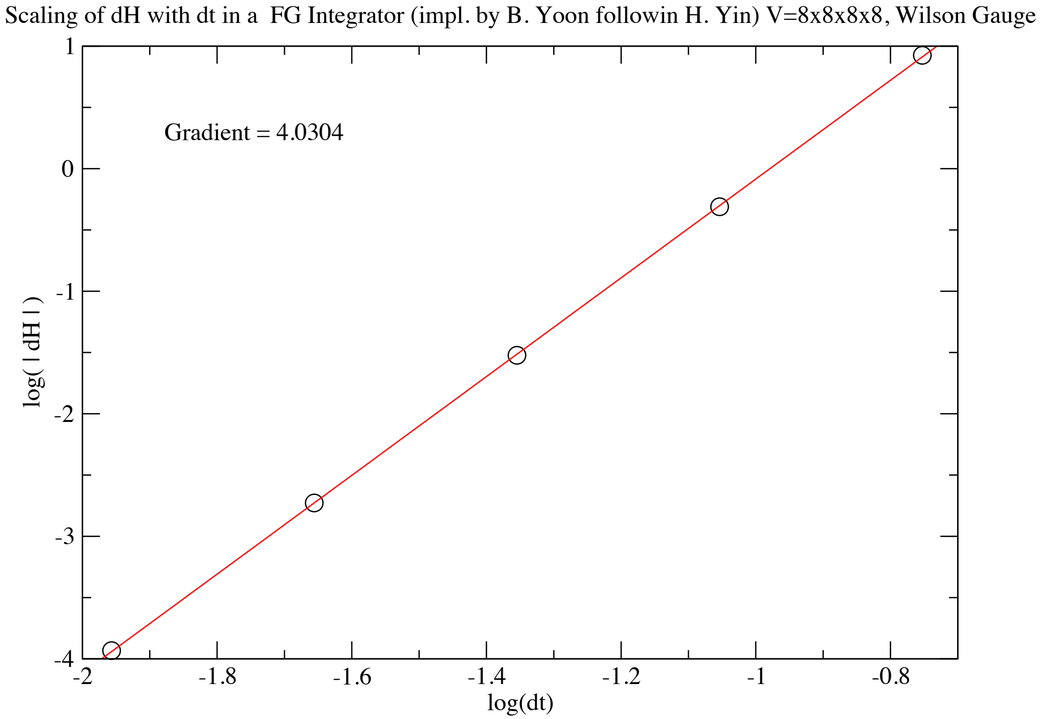




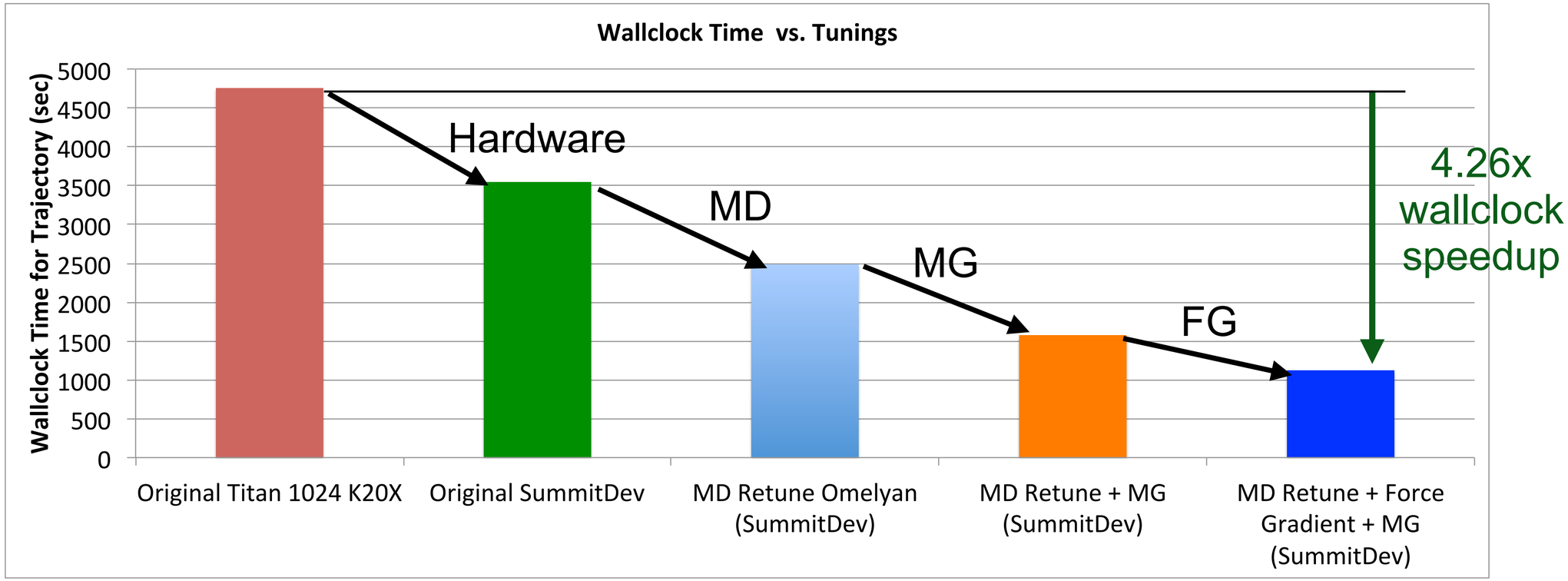
- Outer MG iterations insensitive to refresh iterations once refresh iters > 20 or so
- In terms of time, it is not immediately clear where the point is…
  - For Threshold=170 it is what I expect: there is a nice minimum (fewer iters=>worse space, more iters=> expensive)
  - For Threshold=72, we refresh sufficiently often to keep a low iteration count?

Force Gradient Integrator

- Standard 4th order integrator following Omelyan requires 5 force evaluations per step (4MN5FV version)
- Omelyan 2nd order intergrator requires 3 force evaluations per step
- Force gradient integrator (Clark, Kennedy, Silva) following H. Yin and Mawhinney's exponential trick needs 3 force evaluations + 1 auxiliary force gradient evaluation, but is 4th order
  - Saves on solves compared to 4MN5FV
  - 4th order so volume scaling of cost is $V^{9/8}$.



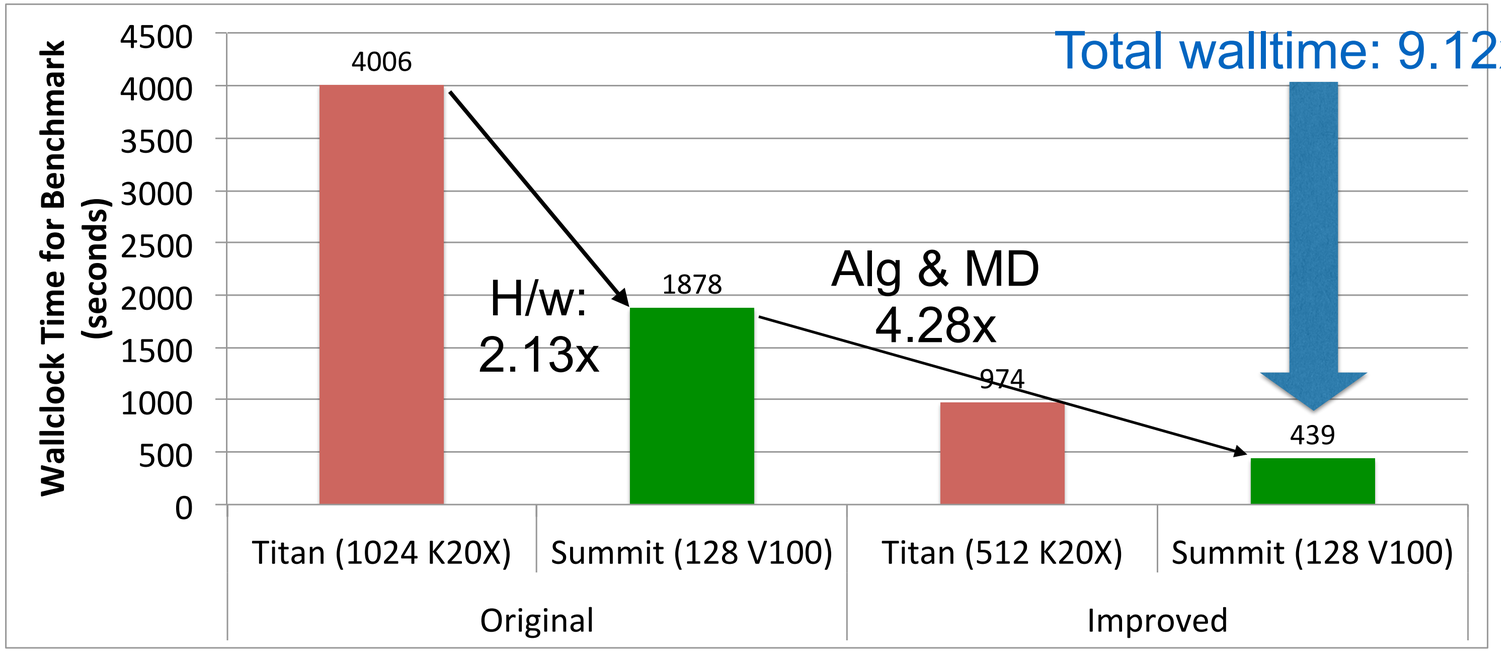
Summit Dev Gains.



- 4.25x wallclock speedup on SummitDev
- On Titan, after all algorithmic optimizations on 512 nodes: ~974 sec per traj
- On Titan on 256 nodes: ~1hour
  - superlinear slowdown: suspect node-device traffic from QDP-JIT cache

Summit Gains

SciDAC Highlight In 2018!



Can We Get To 100x ?

- Combined Hardware x Algorithm & MD Retuning Walltime gain: **9.12x**
- 8x reduction in GPUs: Integrated gain: **73x**. (Contribution to 2018 ECP FOM)
- Updated: more QUDA optimizations (reduced prec chrono vecs etc.): 392.6 sec
  - **10.2x walltime x 8-fold GPU reduction => 81.6x (Contribution to 2019 ECP FOM)**

Hasenbuschery Rebooted (?)

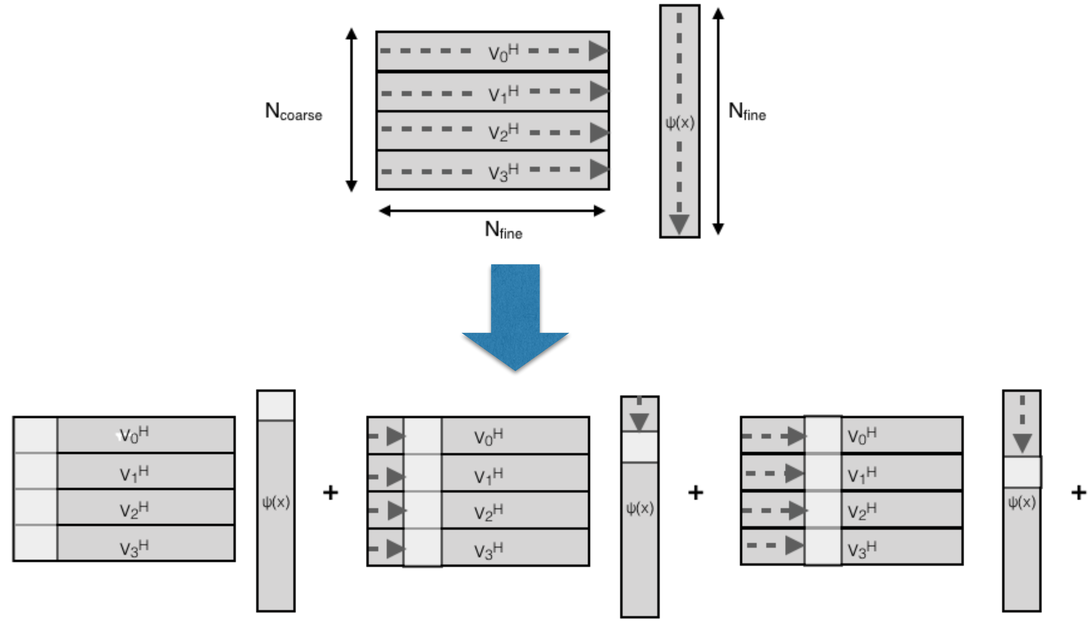
- Work in Progress (K. Clark, B. Joo, W. Sun)
- Observations from current runs:
  - Ratio terms are on same time-scale, MG Iterations weakly sensitive to mass
- Kate: Use a multi-Hasenbusch Term !!
  - Hasenbusch with twisted mass, Krylov subspaces for twists are related
  - Add nonzero μ_1 & accept/reject without or reweight => tame ΔH spikes (?)
  - Multi-RHS Solver (different twist each right hand side)
  - Increase Gauge Reuse & More Comms B/W bound, Better scaling(?)
- Current status:
  - Chroma side: multi-RHS solver interface and multi-Hasenbusch Monomial written
  - SRHS Prec. Clover op with twisted mass written in QUDA using new Dslash infrastructure.
  - SRHS MG for twisted operator in QUDA (not same as regular TWM) needs sorted out next
  - Once that is all working move to multi-RHS pseudo block solver

$$\frac{\det(M_1^2)}{\det(M_2^2)} \times \frac{\det(M_2^2)}{\det(M_3^2)} \cdots$$

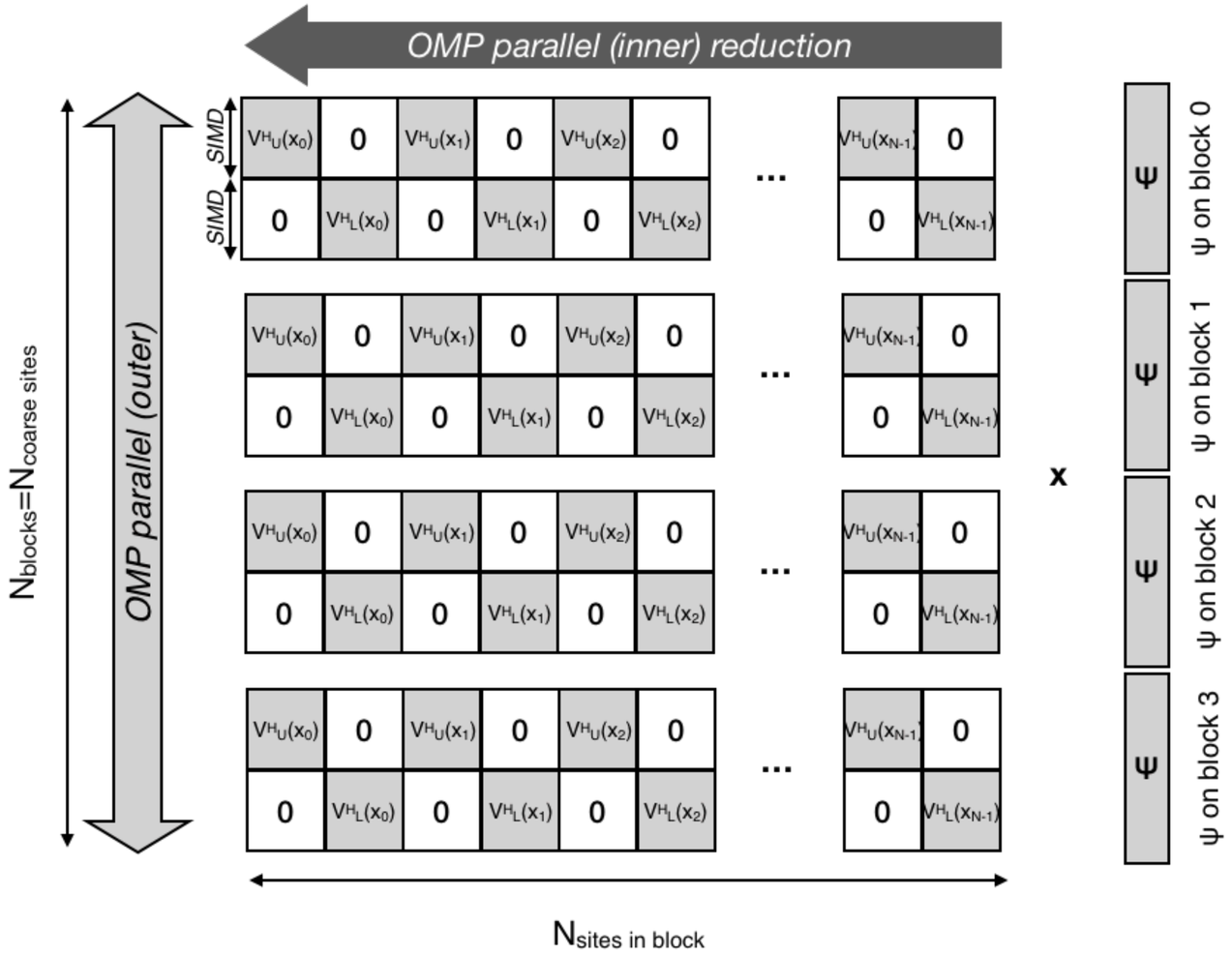
$$\frac{\det(M_1^2+\mu_1^2)}{\det(M_1^2+\mu_2^2)} \frac{\det(M_1^2+\mu_2^2)}{\det(M_1^2+\mu_3^2)} \cdots$$

MG Proto: Props on Intel Xeon & KNL

- QUDA MG Available since 2016
- Needed MG Capability on KNL and X86
- I wanted to learn about what goes into an MG Implementation - use experience gained on GPUs
- Result: MG Proto (prototype MG library)
  - Use QPhiX on Fine Level (efficient AVX,AVX2,AVX512)
  - Vectorize Coarse Mat-Vec ops. In Coarse Op.
  - Investigate threading of coarse op (e.g. fine grained)
    - Fine grained threading doesn't seem to work well with OpenMP - IXPUG 18 paper with T. Kurth
  - Performance not embarrassing…
  - Production use at NERSC, JLab, Meltemi, Stampede2



Nested Parallelism in Restrictor



- Restrictor has several sources of parallelism
  - Outer (coarse-site/block) parallelism
  - Inner Parallel reduction (inner product)
  - SIMD (e.g. over null-vectors)
- Explicit intrinsic based SIMD in reductions needed OMP-4 custom reduction
- Various approaches to parallelizing inner matvecs
  - OMP nested parallelism
  - Manually managed (via thread ID)

KNL Thread Scaling Results

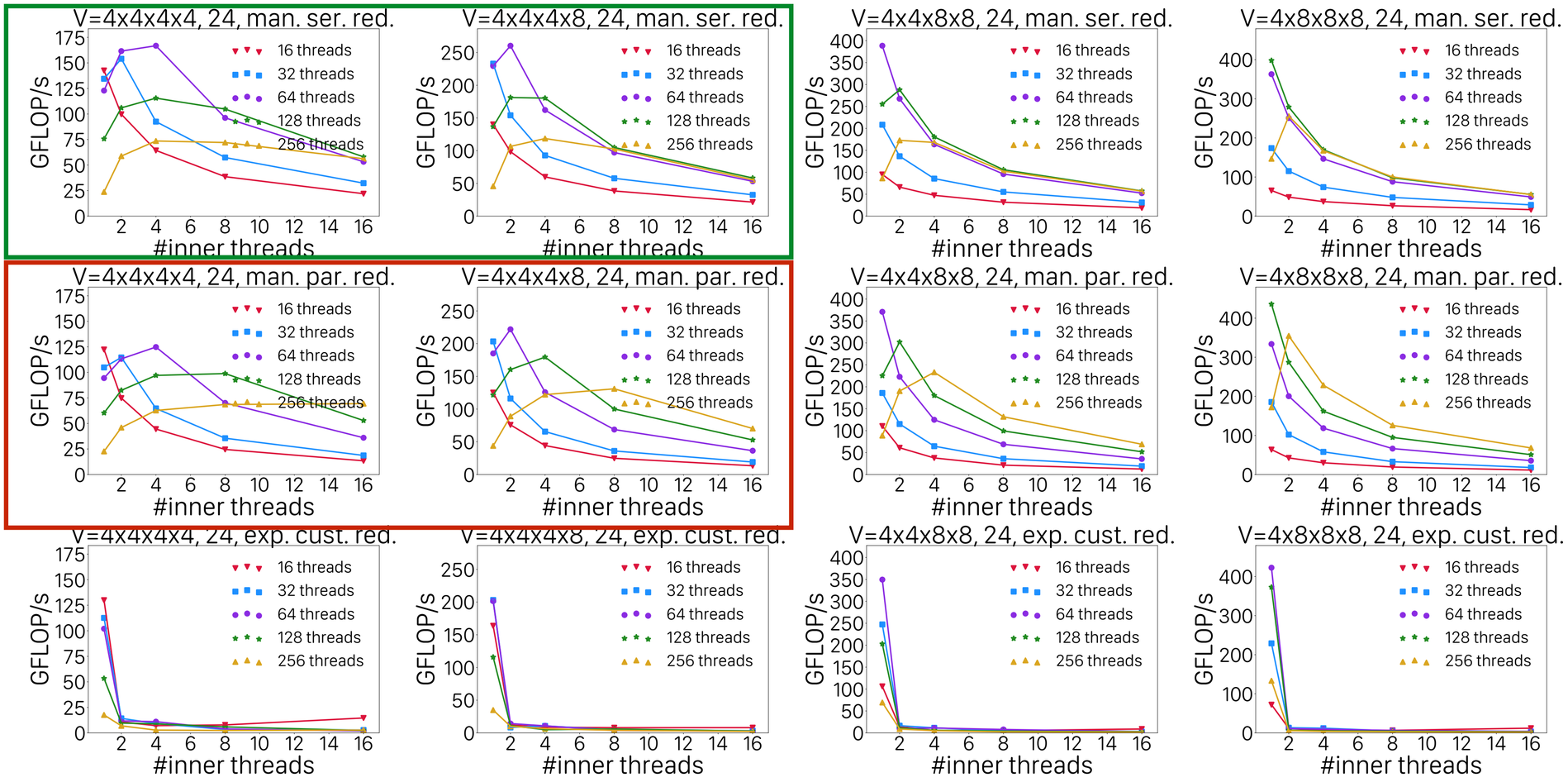
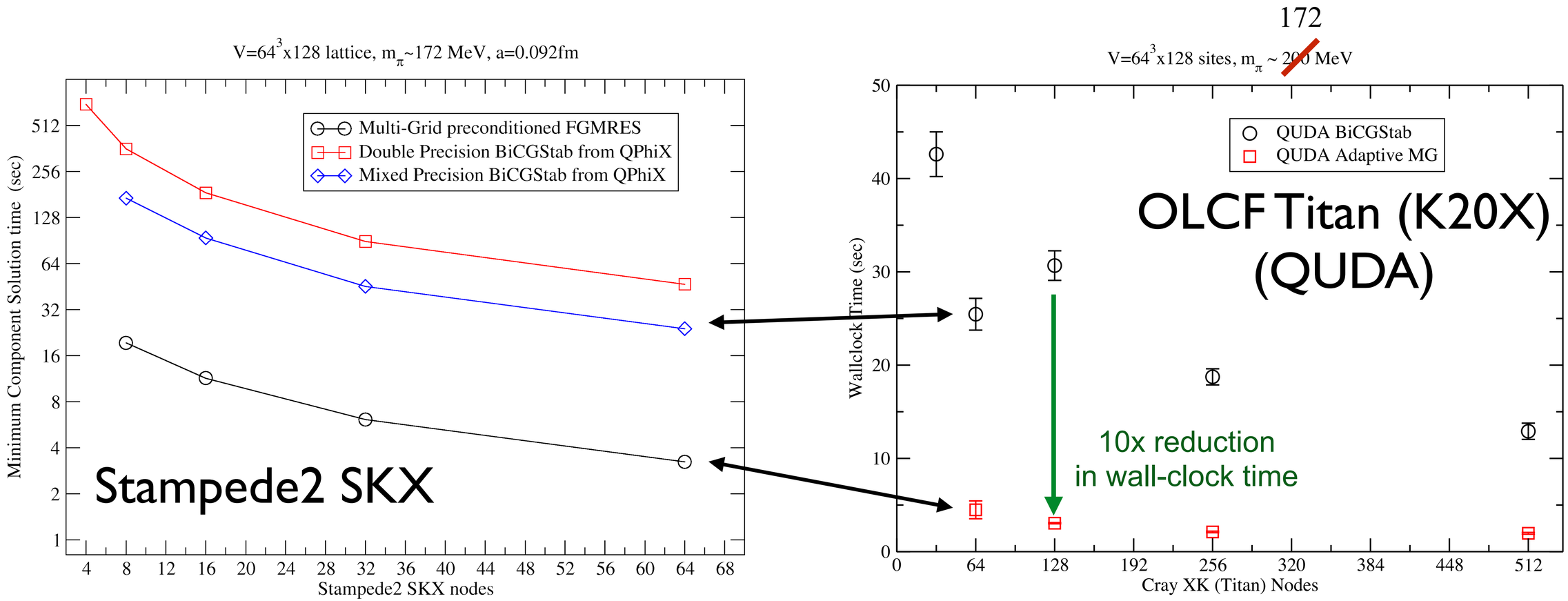


Figure from IXPUG ISC talk by T. Kurth (NERSC)

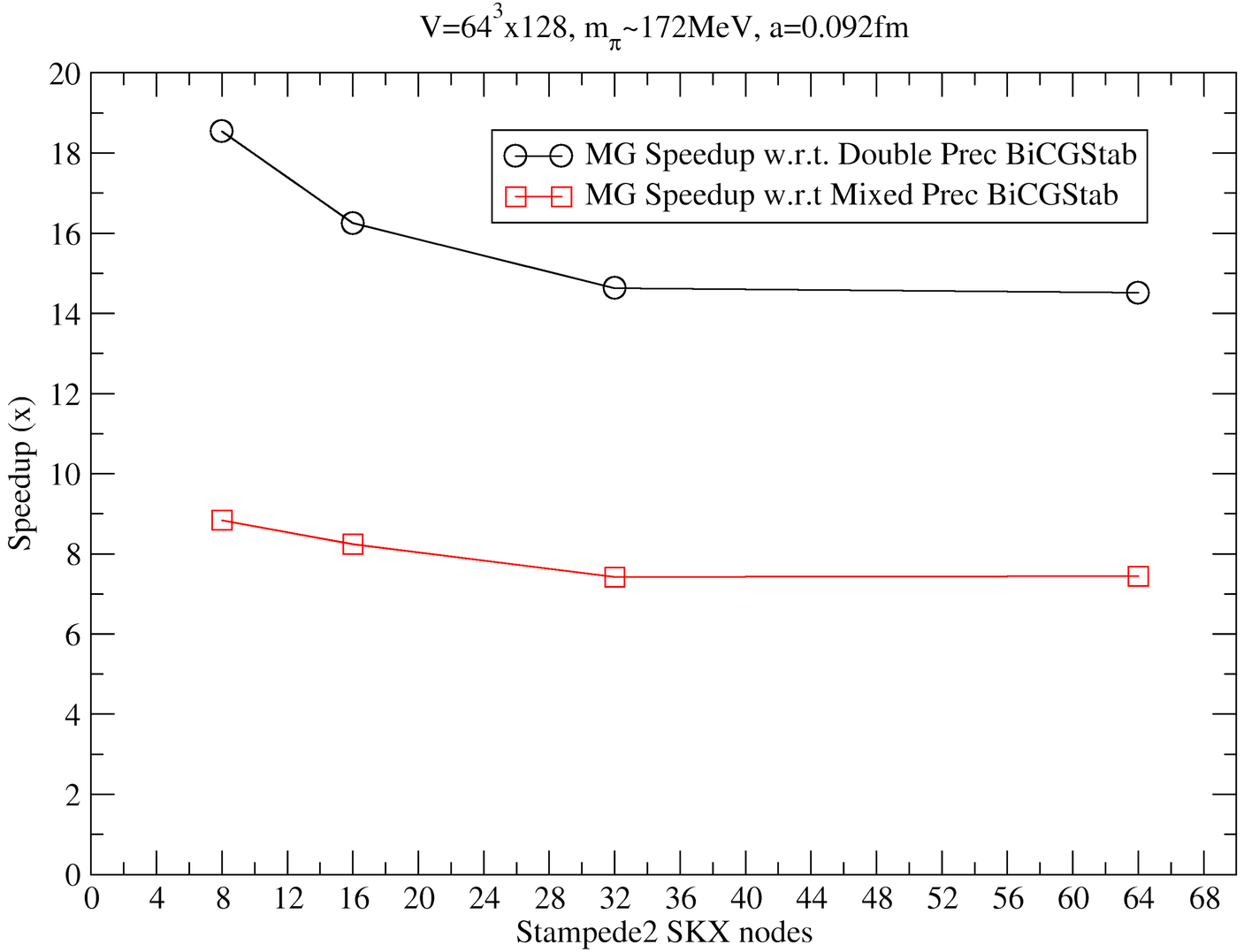
- Explicit nested (exp) OpenMP was always worse than manually (man) managed nesting.
- “Inner” threads only helped on the smallest blocks (2-4 inner threads)
- On smallest blocks, serial reductions worked better than parallelized ones

MG Scaling on Stampede2 SKX nodes



- Stampede SKX performance appears comparable to an OLCF Titan NVIDIA K20X node
  - Fine print: the Titan result was from our SC'16 paper, QUDA performance will have improved since then

MG Speedup on SKL



- Compared to the Mixed precision BiCGStab, the speedup is around 8x.
- Similar speed to what we observed on Titan…

Programming for the Future

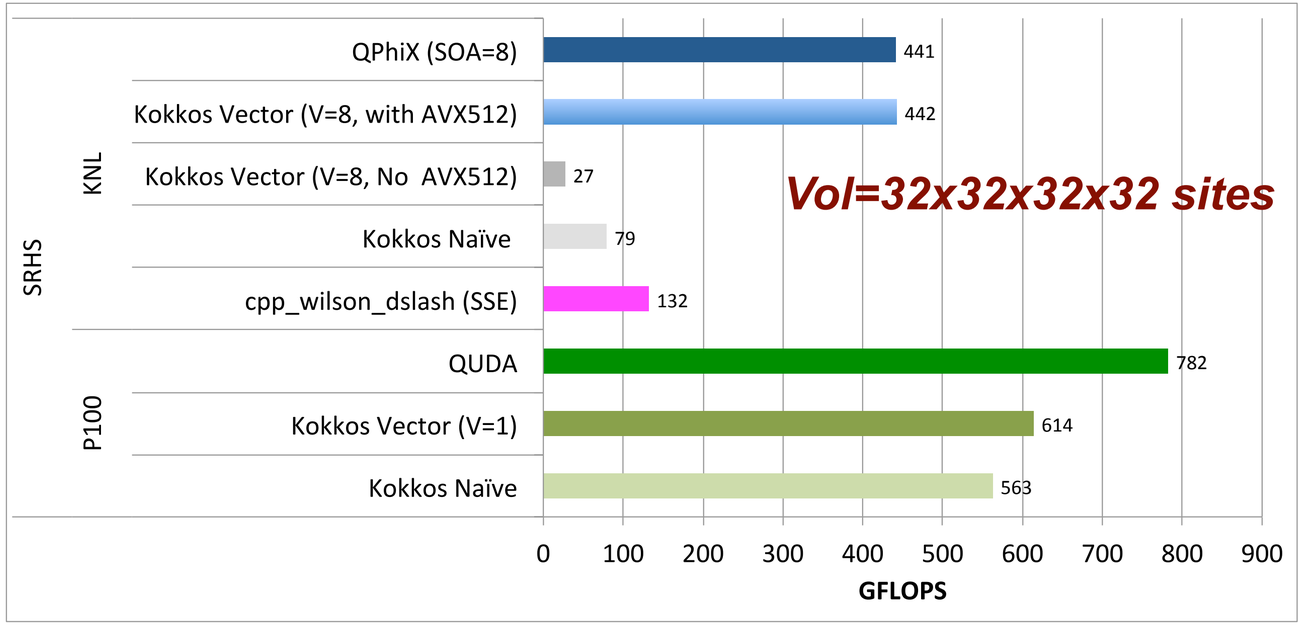
- Very few new ‘conventional clusters’ out there…
  - Frontera at TACC in the U.S. (Xeon Server)?
- Essentially all leadership systems have some form of highly parallel processor
  - Either KNL, or NVIDIA GPU currently.
  - This trend will likely continue into the future: Perlmutter, Aurora, Frontier, …
- Not being able to use Accelerators in your code is a serious handicap
- Not being able to move your code to new hardware can be a disadvantage
  - Worth some amount of strategic planning
- My recommendation: Try Kokkos!

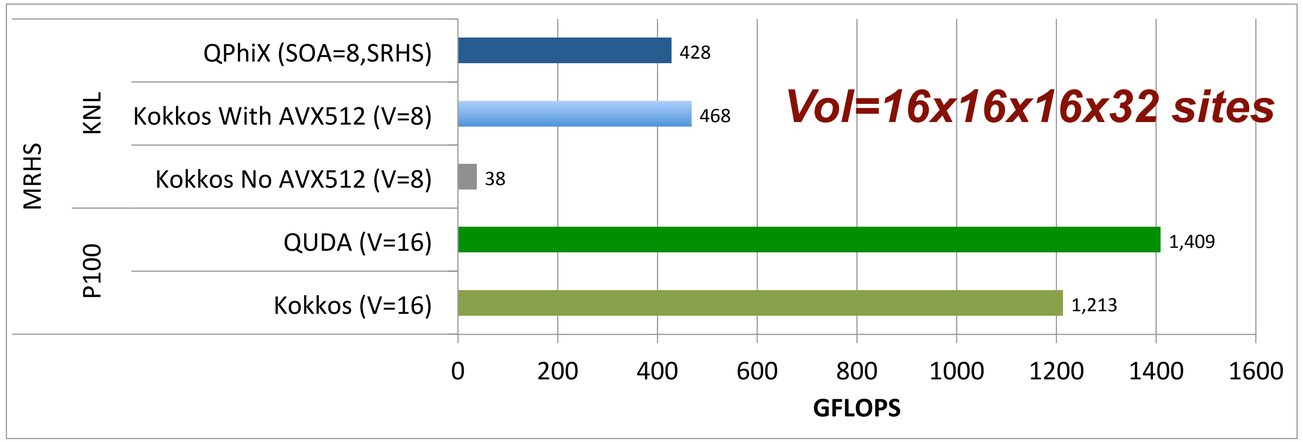
Kokkos Main Features

- Kokkos is an on-node programming model providing
  - Parallel Execution Patterns: parallel for, reduction, scan
  - Parallel Kernels encoded as C++ Functors or C++ Lambdas
  - Separate Execution Spaces (Backends): Host, OpenMP, CUDA, …
  - Separate Memory Spaces: Host, Device, UVM, …
  - Views (multi-dimensional array) to control index order using policies
- Currently best supported: CUDA and OpenMP
- Aurora Support is scheduled for CY20
- Kokkos team members are on C++ standard committee
- I found Kokkos quite useable — modulo vectorizing complex arithmetic on CPUs
- Could be a good option for accelerating Analysis/Contraction codes…

Kokkos Performance Summary

- SRHS Dslash 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 Dslash 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)***





Summary

- Multigrid is in the mainstream in the Clover world:
  - Super efficient GPU implementation and a not-embarrasingly efficient x86 version available
  - MG has now been incorporated into gauge generation in production and is a big boost to the USQCD ECP FOM.
  - MG proto is in production use now on KNL and SKX system (NERSC, TACC, JLab, W&M)
  - MG proto is a good starting point for performance portability tests (Kokkos, OpenMP, …)
- Multi-Hasenbusch should hopefully improve Gauge Generation further
  - More efficient solvers using multi-RHS approaches
  - Stabilize Delta-H spikes in MD with auxiliary twisted masses
- I still think Kokkos could be a good place to start writing performance portable code, especially for new/analysis codes which do not leverage other current performance portability approaches (ie. Not written in Grid, or QDP-JIT, etc.)

Acknowledgements

- Once again, I'd like to acknowledge everyone who has worked and continues to work with me on these projects (See Slide 1)
- 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 Programe (1,2,3, and 4) and from the Exascale Computing Project
- I'd like to acknowledge all the Computer Systems used for this work at
  - NERSC (Edison, Cori), ALCF (Theta), OLCF (Titan, Summit, SummitDev), TACC (Stampede 2), NCSA (Blue Waters), and of course clusters at Jefferson Lab (16p,18p, 12k, 14g).