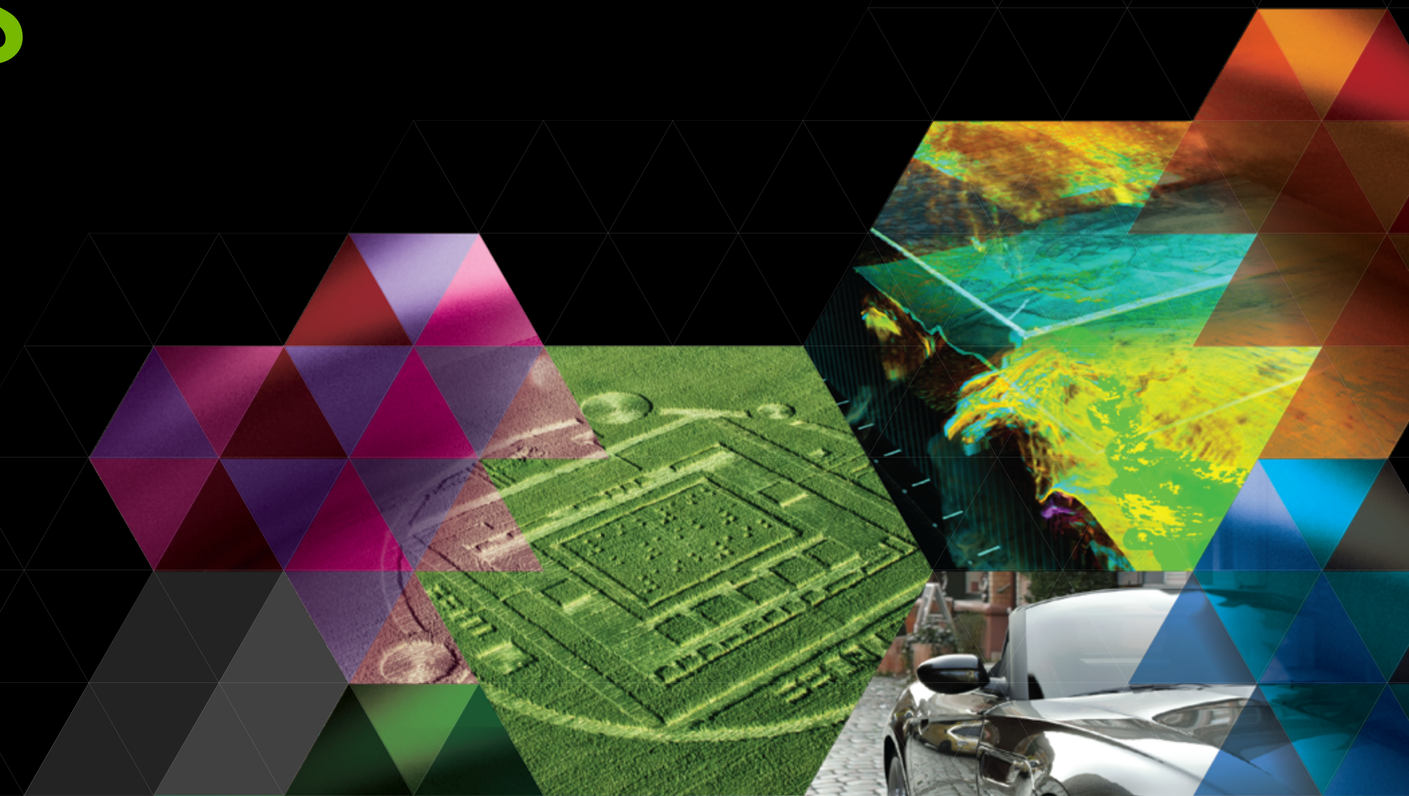
QUDA FEATURES, SCALING AND SOLVERS



NVIDIA

Introducing QUDA

- "QCD on CUDA" – http://lattice.github.com/quda (open source)
- Effort started at Boston University in 2008, now in wide use as the GPU backend for BQCD, Chroma, CPS, MILC, TIFR, etc.
- Provides:
  - Various solvers for all major fermonic discretizations, with multi-GPU support
  - Additional performance-critical routines needed for gauge-field generation
- Maximize performance
  - Exploit physical symmetries to minimize memory traffic
  - Mixed-precision methods
  - Autotuning for high performance on all CUDA-capable architectures
  - Domain-decomposed (Schwarz) preconditioners for strong scaling
  - Eigenvector solvers (Lanczos, EigCG, GMRES-DR) (see talk by A. Strelchenko)
  - Multigrid solvers for optimal convergence
- A research tool for how to reach the exascale

QUDA collaborators

- Ron Babich (NVIDIA)
- Kip Barros (LANL)
- Rich Brower (Boston University)
- Nuno Cardoso (NCSA)
- Michael Cheng (Boston University)
- Justin Foley (Utah -> NIH)
- Joel Giedt (Rensselaer Polytechnic Institute)
- Steve Gottlieb (Indiana University)
- Bálint Joó (Jlab)
- Hyung-Jin Kim (BNL)
- Claudio Rebbi (Boston University)
- Guochun Shi (NCSA -> Google)
- Mario Schröck (INFN)
- Alexei Strelchenko (FNAL)
- Alejandro Vaquero (INFN)
- Mathias Wagner (Indiana University -> NVIDIA)
- Frank Winter (UoE -> Jlab)

QUDA Features

- Latest release 0.7.1 (10th June 2015)
- Dirac operators supported
  - Wilson, Wilson-clover, staggered, improved staggered, twisted mass, non-degenerate twisted mass, twisted-clover, domain wall (4-d / 5-d), Mobius
- Mixed-precision solvers
  - CG, BiCGstab, GCR
  - double, single, half precision
  - 0.7 introduced a huge improvement in mixed-precision CG
- Link smearing and fermion force routines
- GPU-aware MPI and GPU Direct RDMA support
- Eigenvector solvers
- Improved strong scaling

QUDA 0.8 (end of summer)

- Gauge Fixing
  - Overrelaxation and FFT methods
- Pure gauge evolution algorithms
  - Heatbath and overrelaxation
- 4-d multi-GPU random generator
- Improved strong scaling
- Improved performance of many algorithms

QUDA 0.9

- Mixed-precision eigenvector deflation
- Adaptive multigrid
- Multi-rhs solvers

QUDA

Improving Strong Scaling

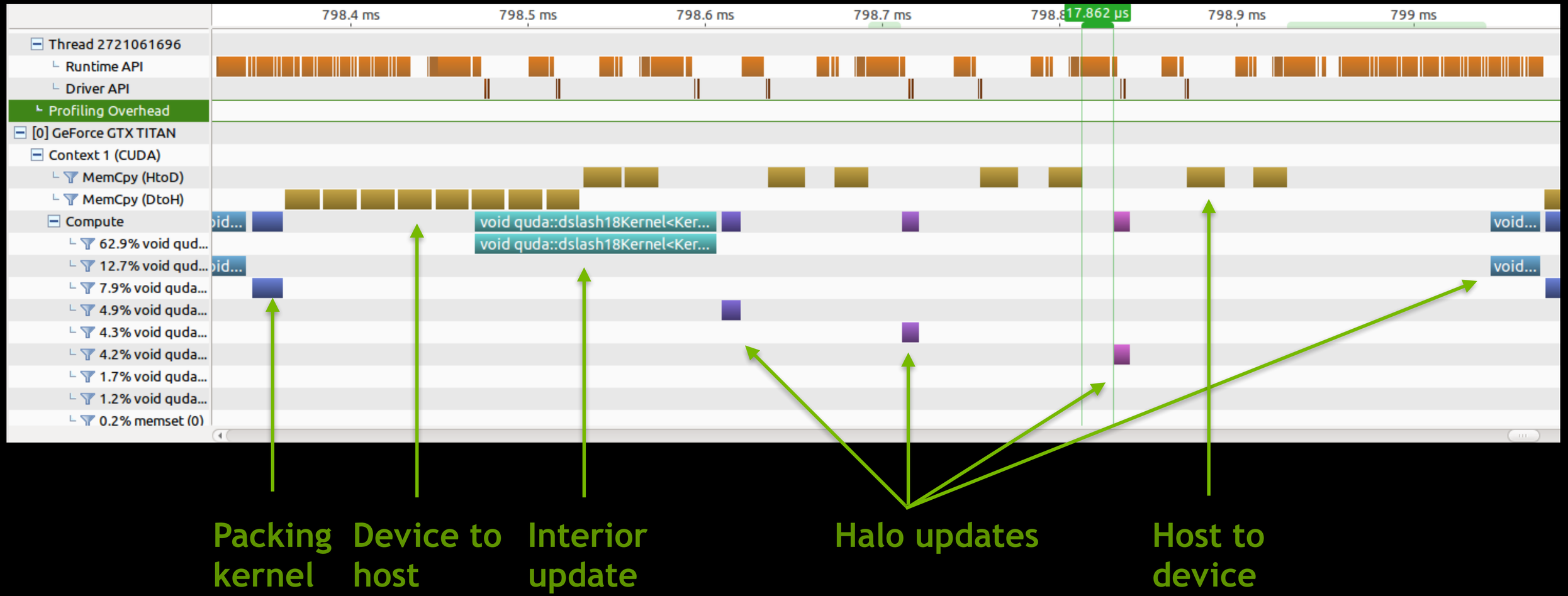
- Communication-reducing solvers available but
  - Not always applicable: staggered, multi-shift, etc.
  - Regular multi-GPU dslash becomes performance limiter
  - GPUs perhaps have a reputation of limited strong scaling

- Surprising observation (prior to QUDA 0.7)
  - PCIe generation 3: doubling in PCIe bandwidth
    - Negligible improvement in performance
  - GPUDirect RDMA: reduces latency by a factor of three
    - Negligible improvement in performance

- What is the performance limiter?

What is limiting strong scaling?

Screen shot from visual profiler (QUDA 0.6, K40, PCIe gen2, wilson dslash, single precision, $V=16^4$)

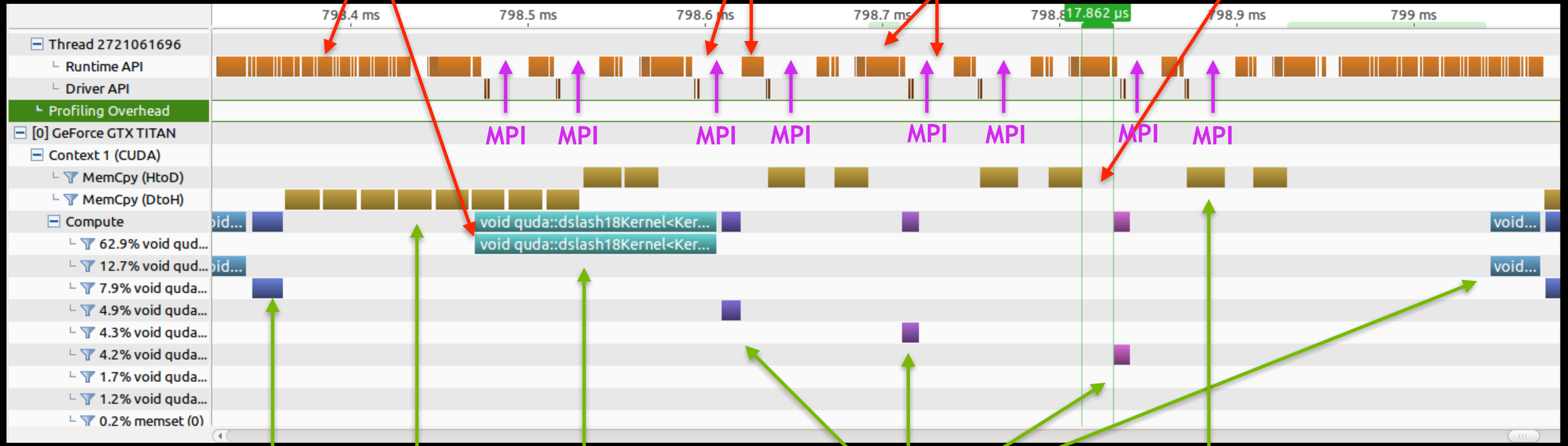


Interior kernel launch blocked by serialized cudaMemcpy API overhead

CUDA and MPI dependent interactions have to be synchronous on CPU

MPI blocked by independent CUDA API calls

Kernel launch latency



Packing kernel

Device to Host

Interior update

Halo updates (GPU under-occupied)

Host to Device

QUDA

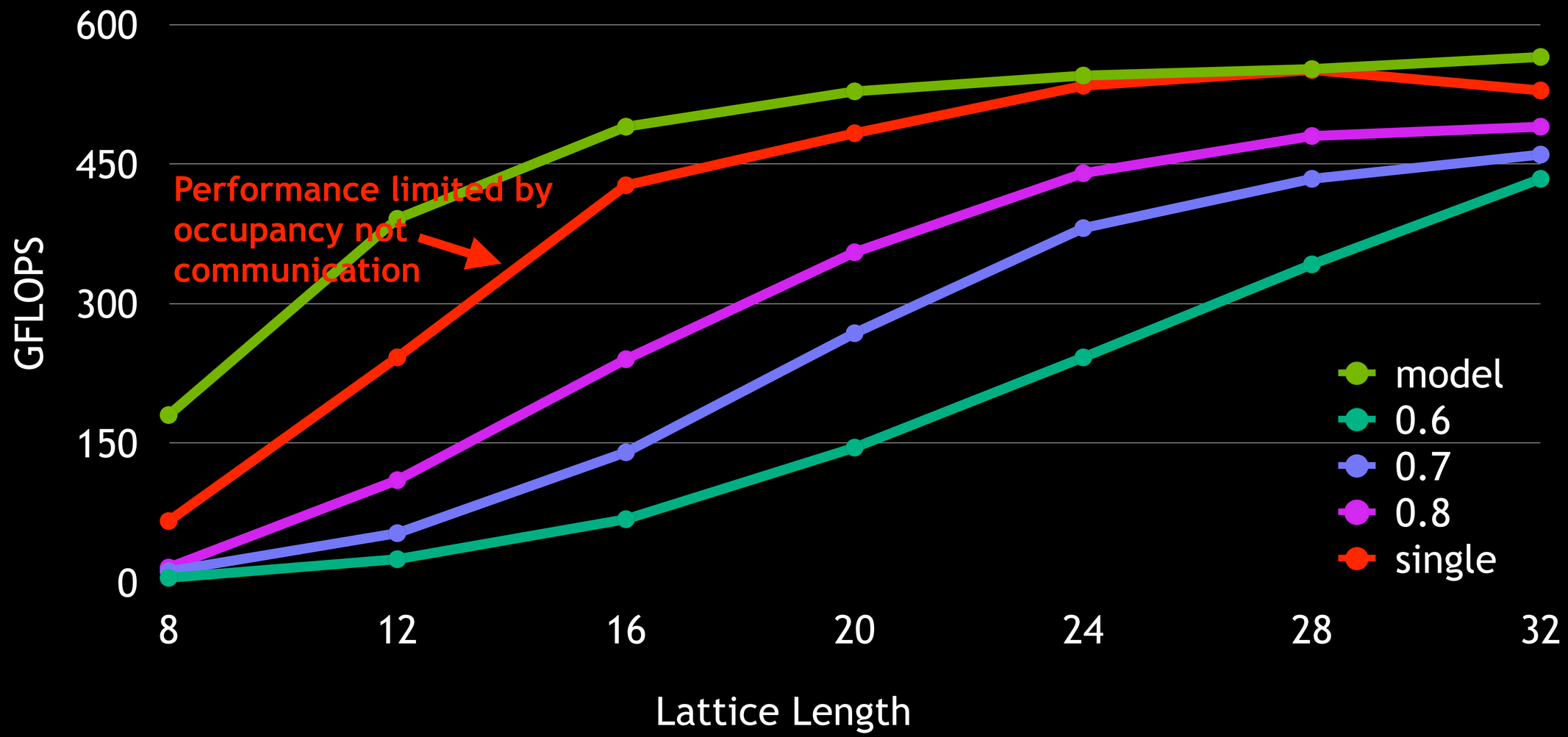
What is limiting strong scaling?

- GPU kernel / memcpy launch overhead (4-10 us)
  - Reduce number of kernels / memcpy
  - Use a single kernel for all halo regions (6->3 kernel calls) (0.7)
  - Fuse norm ghost with main ghost (half precision) (0.8)
- Halo region kernels don’t saturate the GPU
  - Use a single kernel for all halo regions (4x threads) (0.7)
- MPI / CUDA can block each other from progressing
  - Use pthreads to parallelize CUDA and MPI calls (0.8)
- MPI / CUDA have to interact synchronously via CPU
  - Watch this space…

QUDA

Wilson Dslash Strong Scaling

K40, half precision, 8-way communication, 12 reconstruct



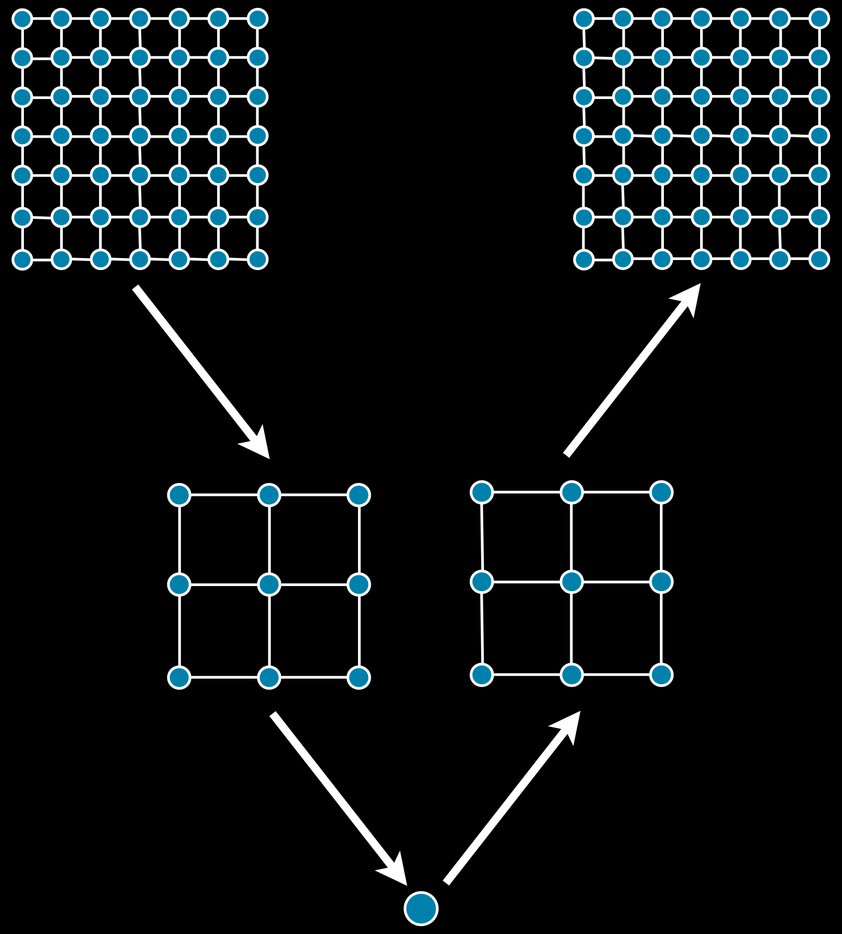
QUDA

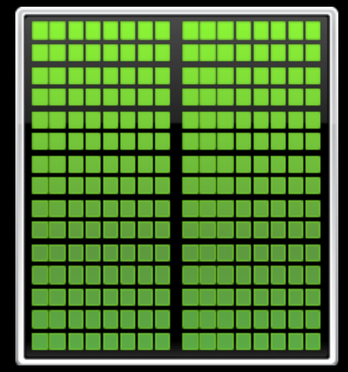
GPU-Driven Communication (0.9+)

- For multi-GPU nodes remove MPI overhead
  - Use the peer-2-peer CUDA API
  - Completely asynchronous execution model
- Ultimate goal is single-kernel multi-GPU dslash
  - Directly read / write from neighboring GPU
  - Maximizes GPU occupancy and minimizes all sources of latency
- QUDA is one of the applications being investigated in the DOE's "DesignForward 2" Exascale programming investigation
- Asynchronous multi-node communication…
- Negligible gap between single and multi-GPU performance
  - Performance limited by parallelism and not communication
  - Expose more parallelism!

QUDA

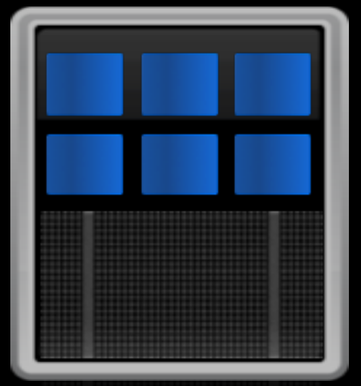
Multigrid





GPU

Thousands of cores
for parallel processing



CPU

Few Cores optimized
for serial work

Multigrid

- QUDA now supports both Wilson and Wilson-clover multigrid
  - Other wilson-like actions trivial to add

- Lack of time has meant much left to be done
  - CPU routines need to be optimized
  - Define an interface and plug this into Chroma
  - End of summer for initial production

- Staggered multigrid efforts ramping up (A. Strelchenko and R. Brower)
  - Fairly easy: delete loops over spin loops
  - Careful thought about Hermiticity requirements

QUDA

Multi-rhs solvers

- Instead of solving a single system Ax = b, solve many

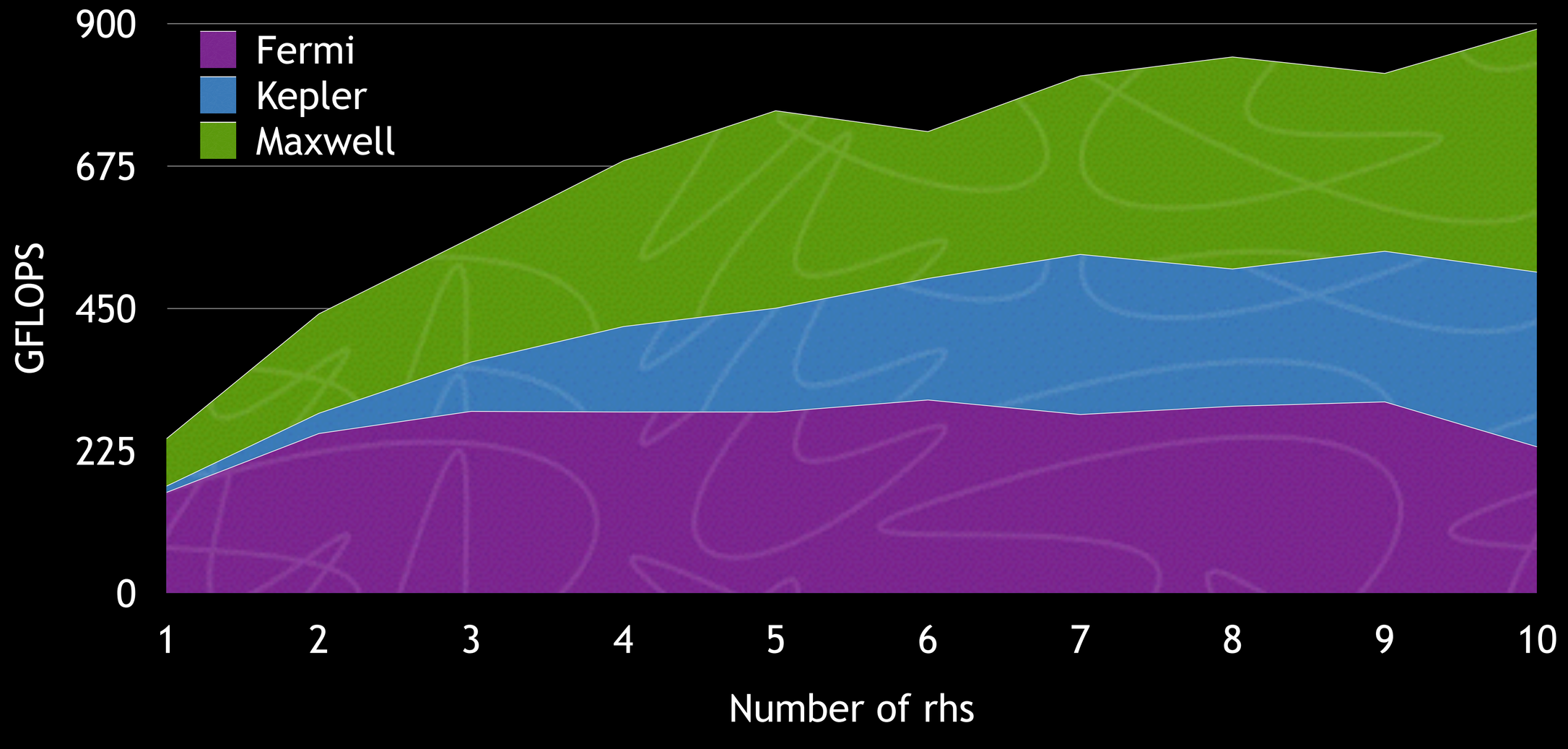
$$Ax_i = b_i$$

- Extremely easy to implement
  - Treat right-hand index i as 5^{th} dimension
  - Gauge field remains 4 dimensional
- Keep rhs index i local to each thread block
  - Gauge field locality through texture cache
- Note this is very similar to domain wall

QUDA

Improved Staggered Multi-rhs Dslash

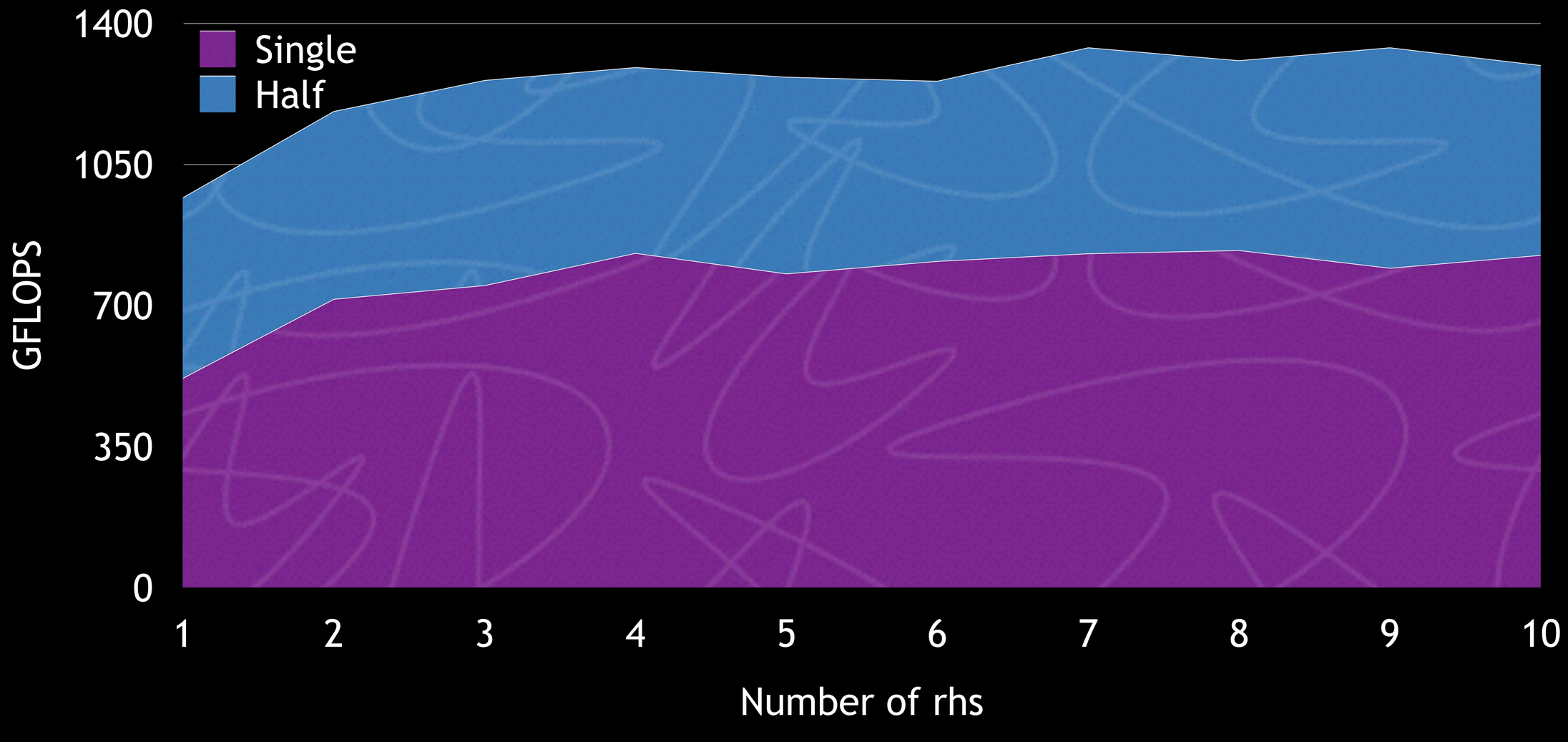
Volume = 24^4, single precision, no reconstruction



QUDA

Wilson Multi-rhs Dslash

Maxwell (M6000), Volume = 24^4, single precision, 12 reconstruction

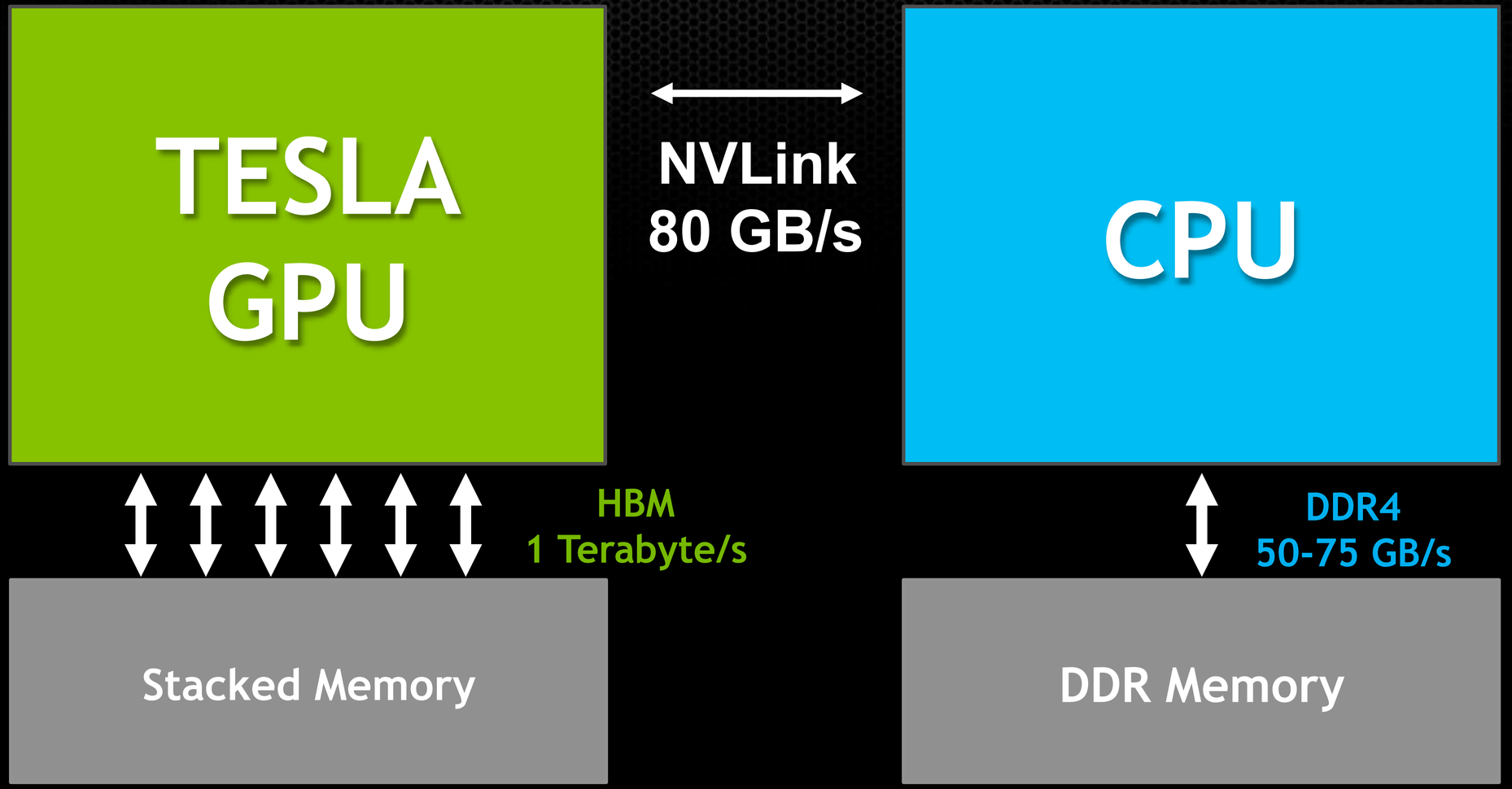


QUDA

Multi-rhs solvers

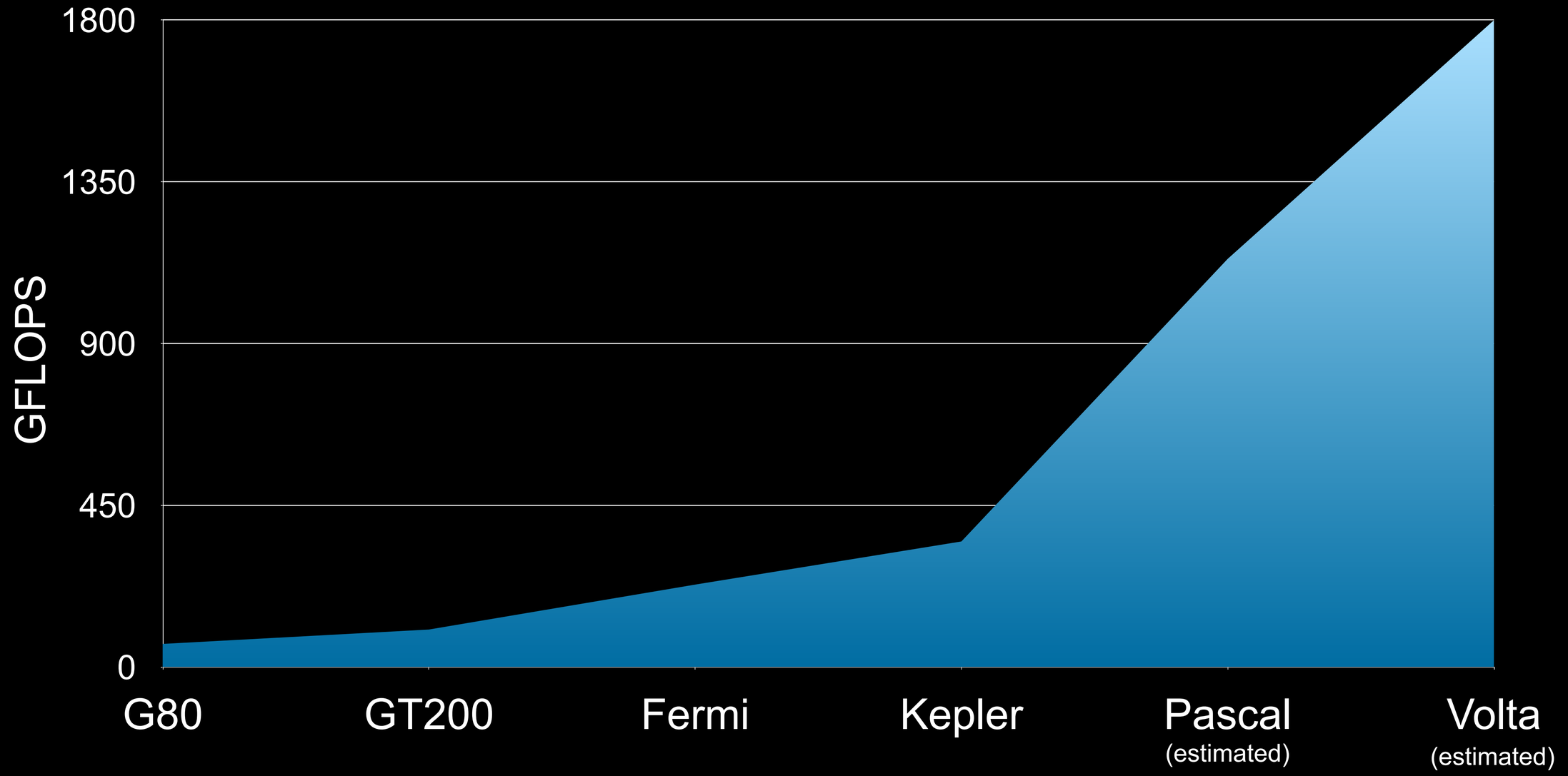
- Multi-rhs Wilson CG (double-half) on 4x M6000 GPUs
  - 3 TFLOPS sustained
- Increasing #rhs improves strong scaling (latency reduction)
- Combine with deflation methods for bigger speedups
  - Eigen-vector deflation 5-10x speedup for multi-rhs systems
  - Utilize with multi-rhs solver 1.5-4x speedup
  - Overall speedup of 8-40x versus serial solver
  - Huge benefit for analysis workloads
- Expect similar benefits for multigrid
  - Gain for multigrid may be even greater since increasing rhs exposes more data parallelism in the coarse grids

GPU Computing in 2016



LQCD Performance with GPU generation

Single Precision Wilson-Dslash performance, $V=24^4$



QUDA

GPU Computing in 2016

- Massive disruption in LQCD performance

- High Bandwidth Memory (HBM)
  - Memory bandwidth increases up to 1 TB/s
  - Out-of-the-box >3x speedup in dslash performance
  - Up to 32 GiB HBM per GPU

- NVLink-enabled servers
  - No drawback from building dense GPU nodes
  - PCIe bandwidth used exclusively for NIC
  - A single node sustaining 10-20+ TFLOPS solver performance
  - “Modest” clusters will achieve 1 PFLOPS of solver performance

US to Build Two Flagship Supercomputers

Powered by the Tesla Platform

OAK RIDGE National Laboratory

Lawrence Livermore National Laboratory

IBM

NVIDIA

100-300 PFLOPS Peak

10x in Scientific App Performance

IBM POWER9 CPU + NVIDIA Volta GPU

NVLink High Speed Interconnect

40 TFLOPS per Node, >3,400 Nodes

2017

Major Step Forward on the Path to Exascale

QUDA

Summary

- QUDA is a production library for GPU-accelerated LQCD
  - Coverage for most common LQCD algorithms
- Recent focus on improving strong scaling
  - Improves the performance of all solvers
  - More improvement to come
- Ongoing efforts in multigrid
- Multi-rhs solvers give significant speedup
- GPU Computing in 2016+ will be disruptive

QUDA

Mixed-precision solvers

- QUDA has had mixed-precision from the get go
- Almost a free lunch where it works well (wilson/clover)
  - Residual injection / reliable updates mixed-precision BiCGstab
  - 2 Tflops sustained in workstation (4 GPUs)
- Did not work well for CG (staggered / twisted mass / dwf)
  - double-single has increased iteration count
  - double-half non convergent
- Why is this?
  - CG recurrence relations much more intolerant
  - BiCGstab noisy as hell anyway
- Need to make CG more robust
  - Make double-half work
  - Less polishing in mixed-precision multi-shift solver

(Stable) Mixed-precision CG

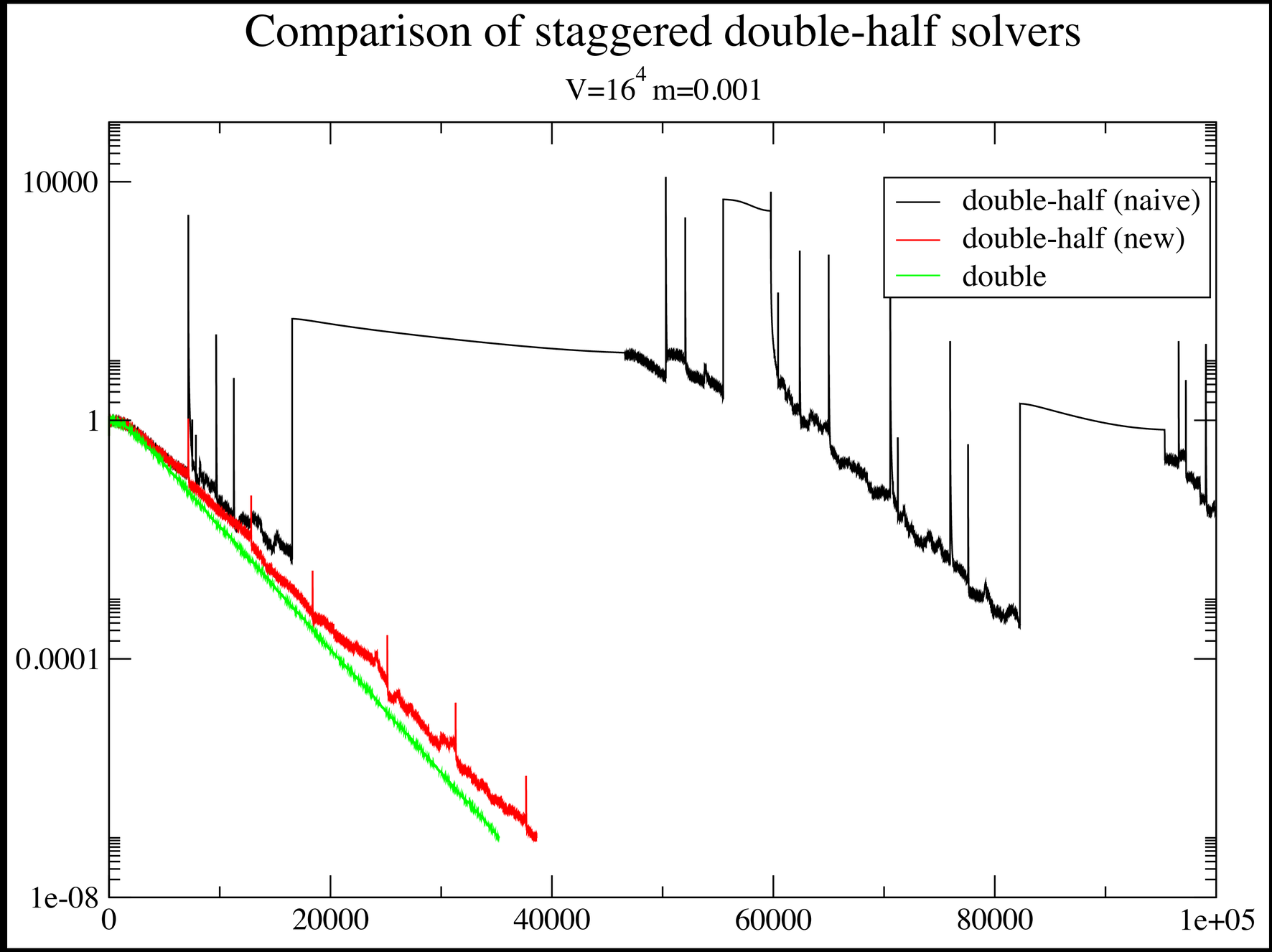
- CG convergence relies on gradient vector being orthogonal to residual
  - Re-project when injecting new residual
- α chosen to minimize $|e|_A$
  - True irrespective of precision of p, q, r
  - Solution correction is truncated if we keep low precision x
  - Always keep solution vector in high precision
- β computation relies on $(r_i,r_j) = |r_i|^2 \delta_{ij}$
  - Not true in finite precision
  - Polak-Ribière formula is equivalent and self-stabilizing through local orthogonality

$$\beta_k = \alpha(\alpha(q_k,q_k) - (p_k,q_k))/(\mathbf{r}_{k-1},\mathbf{r}_{k-1})$$

- Further improvement possible
  - Mining the literature on fault-tolerant solvers…

```
while (|r_k|> ε) {
    β_k = (r_k,r_k)/(r_{k-1},r_{k-1})
    p_{k+1} = r_k - β_k p_k
    q_{k+1} = A p_{k+1}
    α = (r_k,r_k)/(p_{k+1}, q_{k+1})
    r_{k+1} = r_k - α q_{k+1}
    x_{k+1} = x_k + α p_{k+1}
    k = k+1
}
```

QUDA


Comparison of staggered double-half solvers
V=16^4 m=0.001
10000
1
0.0001
1e-08
0
20000
40000
60000
80000
1e+05
double-half (naive)
double-half (new)
double

QUDA

Halo Region Updates (QUDA 0.7)

- Best way to reduce latency all round is kernel fusion
  - Reduces API calls
  - Reduces kernel launch overhead
  - Increases GPU occupancy
- Previous multi-GPU dslash had 6 kernels
  - pack (all faces), interior, halo_t, halo_z, halo_y, halo_x
- This puts a lower bound on the minimum time taken regardless of the speed of the GPU execution
- Fused multi-GPU dslash now has 3 kernels halving lower bound
  - pack (all faces), interior, halo (all faces)
- Scope for further fusing if we consider D_{eo} D_{oe} together
  - pack -> interior -> halo -> pack -> interior -> halo

QUDA

Other improvements

- Double buffering of QMP/MPI receive buffers (QUDA 0.7)
  - Early pre-posting of MPI Receive
- Dslash has been rewritten using pthreads to parallelize between independent MPI and CUDA API calls (QUDA 0.8)
  - Parallelize between CUDA -> MPI and MPI -> CUDA dependent operations. E.g., waiting on MPI in t dimension while waiting on device -> host copy in z dimension
- Improvement to half-precision latency (QUDA 0.8)
  - Previously norm field was stored in separate halo region
  - Now store in same array as main quark field
    - Halves host -> device API calls
    - Increases message size for improved throughput