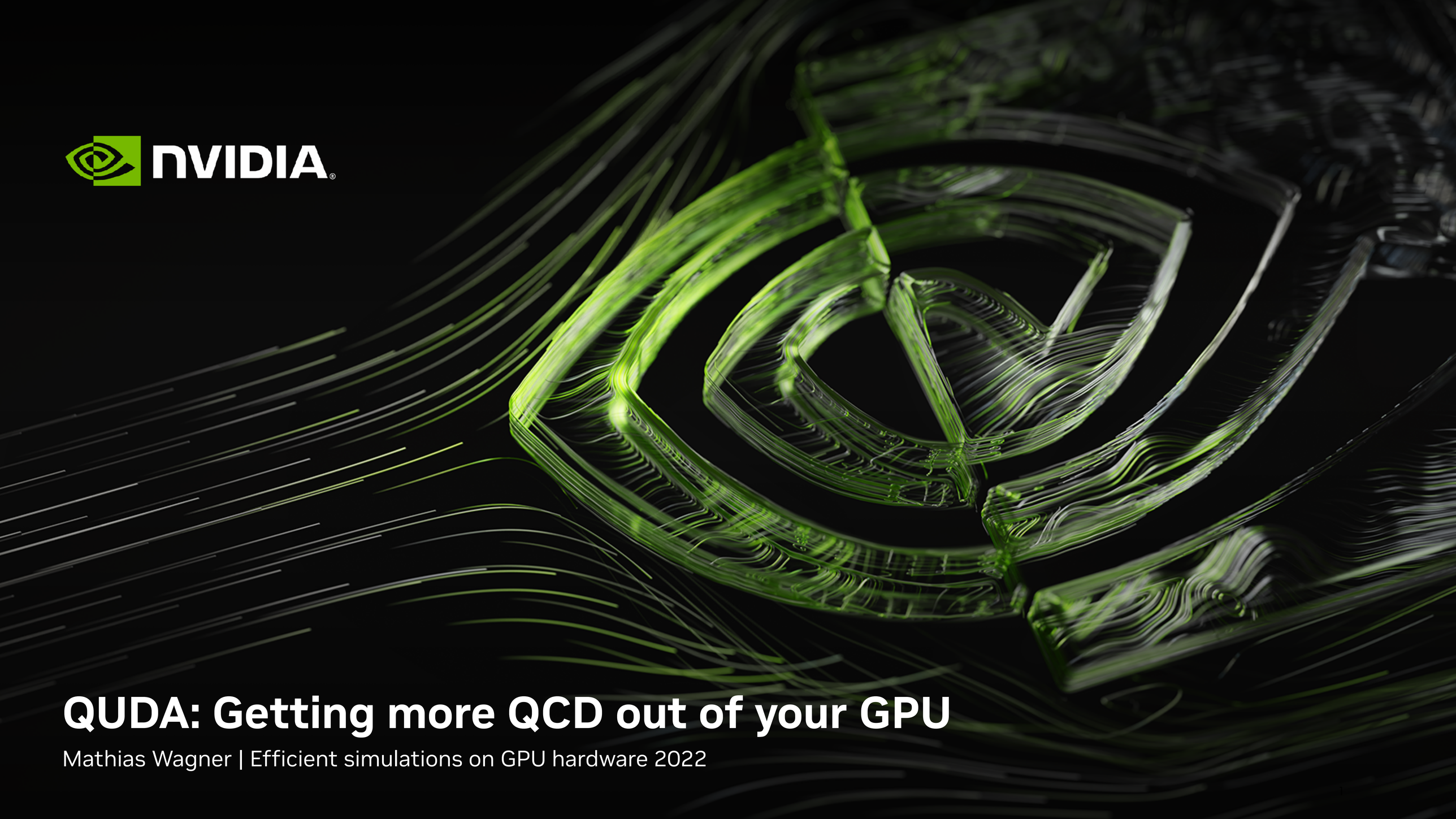


QUDA: Getting more QCD out of your GPU

Mathias Wagner | Efficient simulations on GPU hardware 2022

http://lattice.github.com/quda

QUDA

- Effort started at Boston University in 2008, now in wide use as the GPU backend for BQCD, Chroma, CPS, MILC, TIFR, tmLQCD, etc.
- Provides:
  - Various solvers for all major fermionic 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
  - Eigenvector and deflated solvers (Lanczos, EigCG, GMRES-DR)
  - Multigrid solvers for optimal convergence Multi-source solvers
  - Strong-scaling improvements
- Portability
  - Started on NVIDIA GPUs with CUDA (“QCD on CUDA”)
  - Added support for AMD through HIP (in current develop branch)
  - Ongoing work for Intel through SYCL and OpenMP Offload (open PR, work ongoing)

QUDA CONTRIBUTORS

10+ years - lots of contributors

Buck Babich (NVIDIA)
Simone Bacchio (Cyprus)
Kip Barros (LANL)
Rich Brower (Boston University)
Nuno Cardoso (NCSA)
Kate Clark (NVIDIA)
Michael Cheng (Boston University)
Carleton DeTar (Utah University)
Justin Foley (Utah -> NIH)
Joel Giedt (Rensselaer Polytechnic Institute)
Arjun Gambhir (LBL)
Steve Gottlieb (Indiana University)
Kyriakos Hadjiyiannakou (Cyprus)
Dean Howarth (LBL)
Xiao-long Jin (ANL)
Bálint Joó (ORNL)
Hyung-Jin Kim (BNL -> Samsung)
Bartek Kostrzewa (Bonn)
James Osborn (ANL)
Claudio Rebbi (Boston University)
Eloy Romero (William and Mary)
Hauke Sandmeyer (Bielefeld)
Guochun Shi (NCSA -> Google)
Mario Schröck (INFN)
Alexei Strelchenko (FNAL)
Jiqun Tu (NVIDIA)
Carsten Urbach (Bonn)
Alejandro Vaquero (Utah University)
Mathias Wagner (NVIDIA)
André Walker-Loud (LBL)
Evan Weinberg (NVIDIA)
Frank Winter (Jlab)
Yi-bo Yang (CAS)
➡ SOON *Your name here ?*

DO MORE SCIENCE

Reduce time to solution

Faster and / or more hardware (strong scaling)

More bandwidth

Lower latencies

More efficient use of hardware

More work with the same data

MAPPING THE DIRAC OPERATOR TO GPUS

Finite difference operator in LQCD is known as Dslash

Assign a single space-time point to each thread

V = XYZT threads, e.g., V = 24^4 => $3.3\text{x}10^6$ threads

Looping over direction each thread must

Load the neighboring spinor (24 numbers x8)

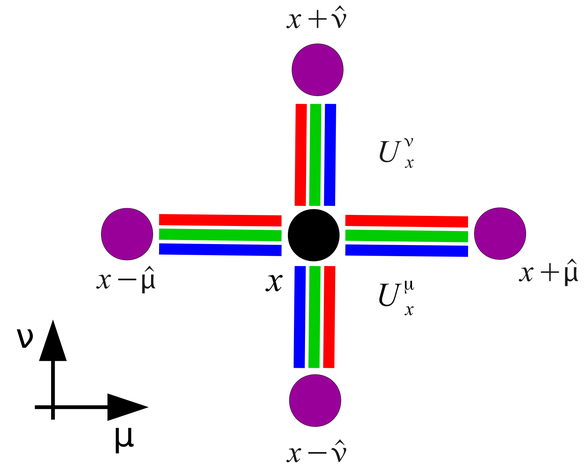
Load the color matrix connecting the sites (18 numbers x8)

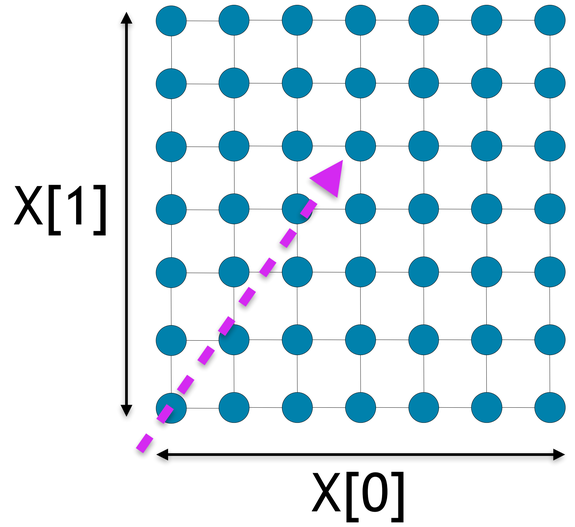
Do the computation

Save the result (24 numbers)

Each thread has (Wilson Dslash) 0.92 naive arithmetic intensity

$$D_{x,x'} =$$





ANNOUNCING H100

Unprecedented Performance, Scalability, and Security for Every Data Center

HIGHEST AI AND HPC PERFORMANCE

4PF FP8 (6X)| 2PF FP16 (3X)| 1PF TF32 (3X)| 60TF FP64 (3X)
3TB/s (1.5X), 80GB HBM3 memory

TRANSFORMER MODEL OPTIMIZATIONS

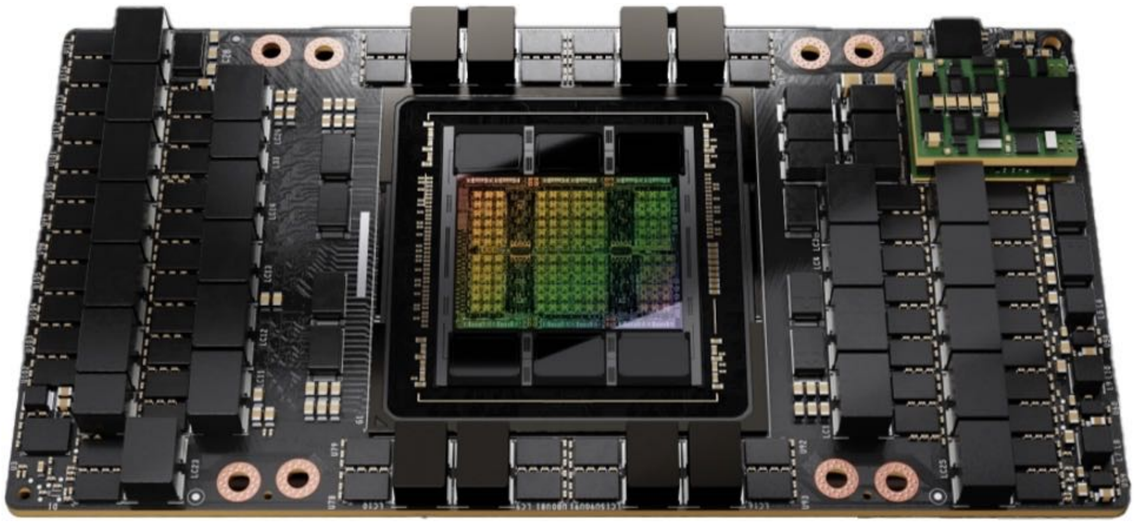
6X faster on largest transformer models

HIGHEST UTILIZATION EFFICIENCY AND SECURITY

7 Fully isolated & secured instances, guaranteed QoS
2nd Gen MIG | Confidential Computing

FASTEST, SCALABLE INTERCONNECT

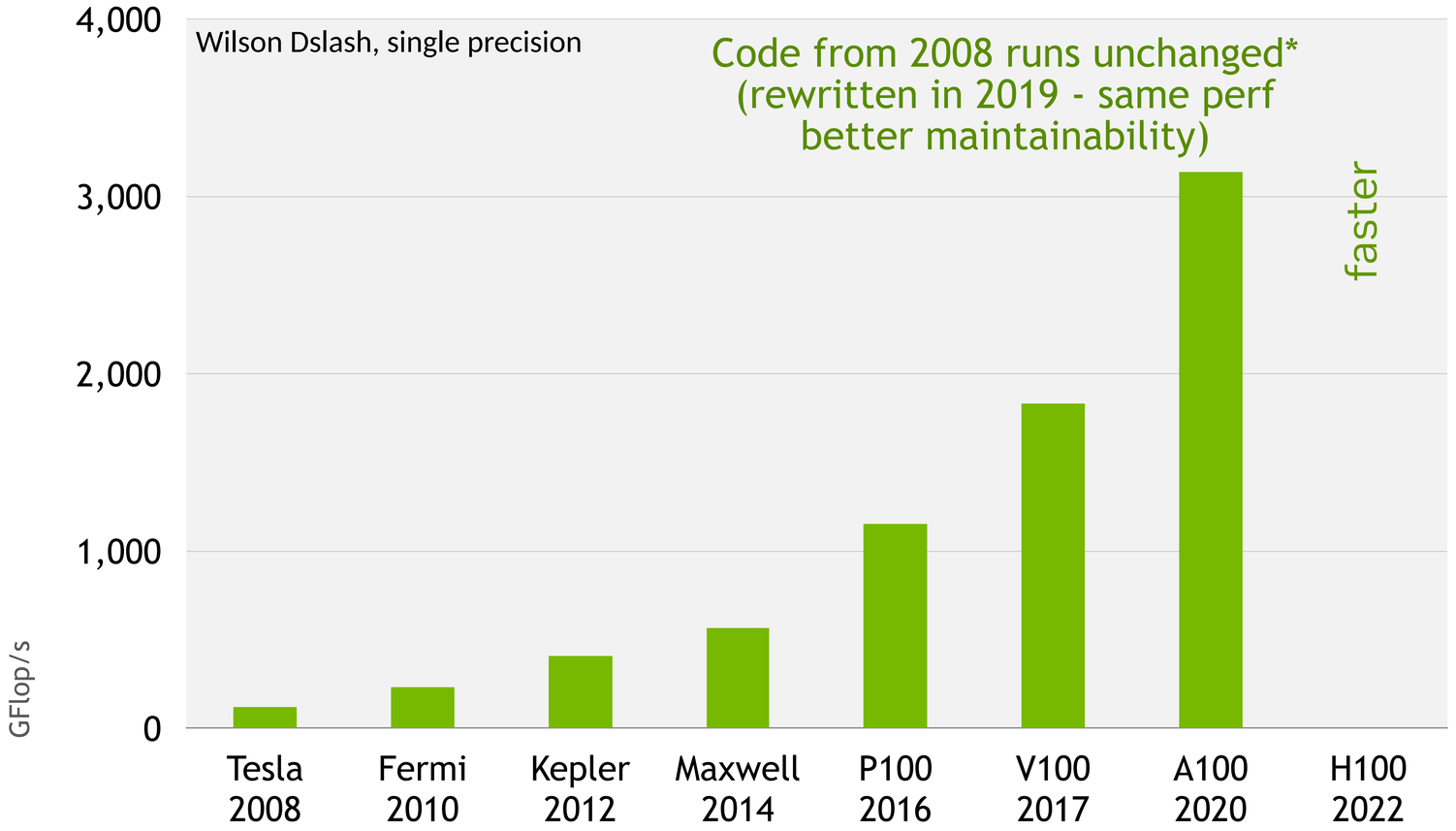
900 GB/s GPU-2-GPU connectivity (1.5X)
up to 256 GPUs with NVLink Switch | 128GB/s PCIe Gen5



Custom 4N TSMC Process | 80 billion transistors

SINGLE GPU PERFORMANCE

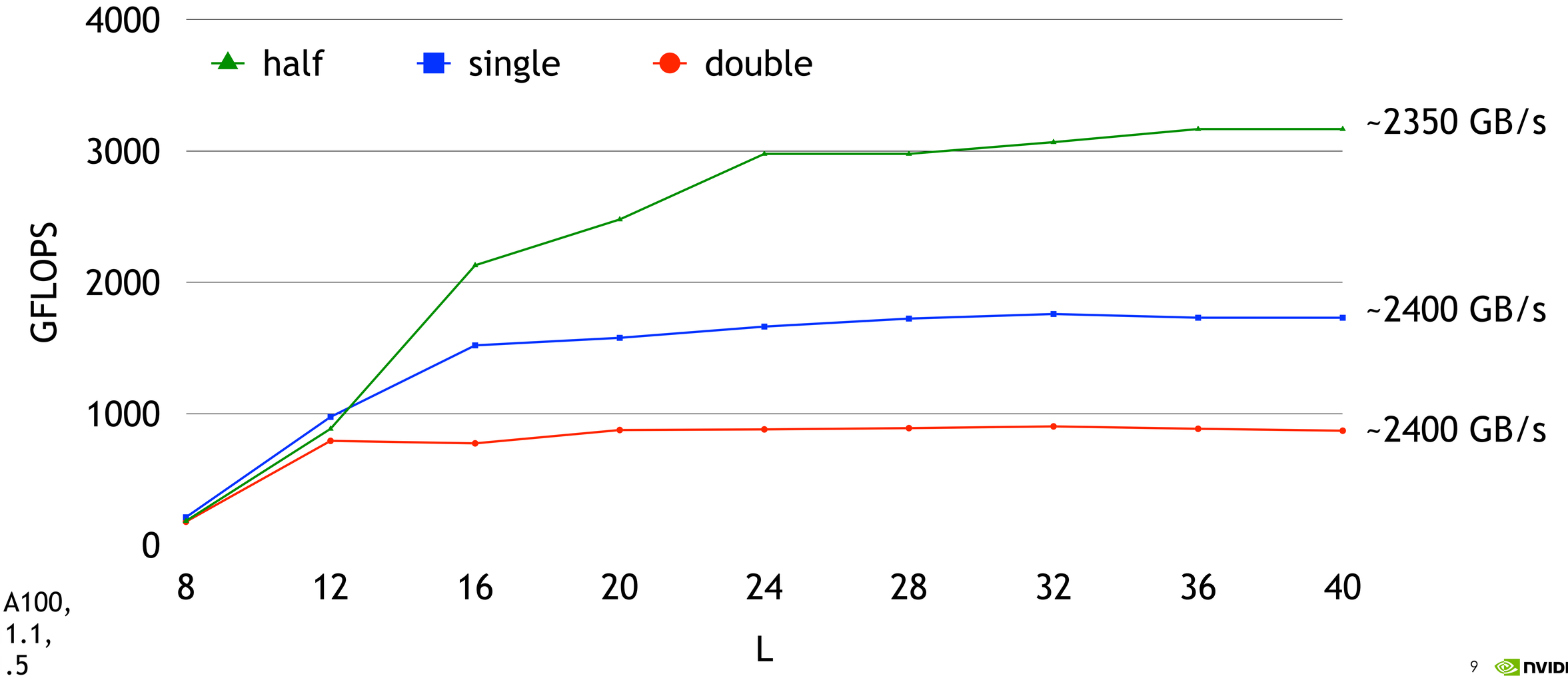
Wilson Dslash Kernel



EFFICIENT USE OF BANDWIDTH

SINGLE GPU PERFORMANCE

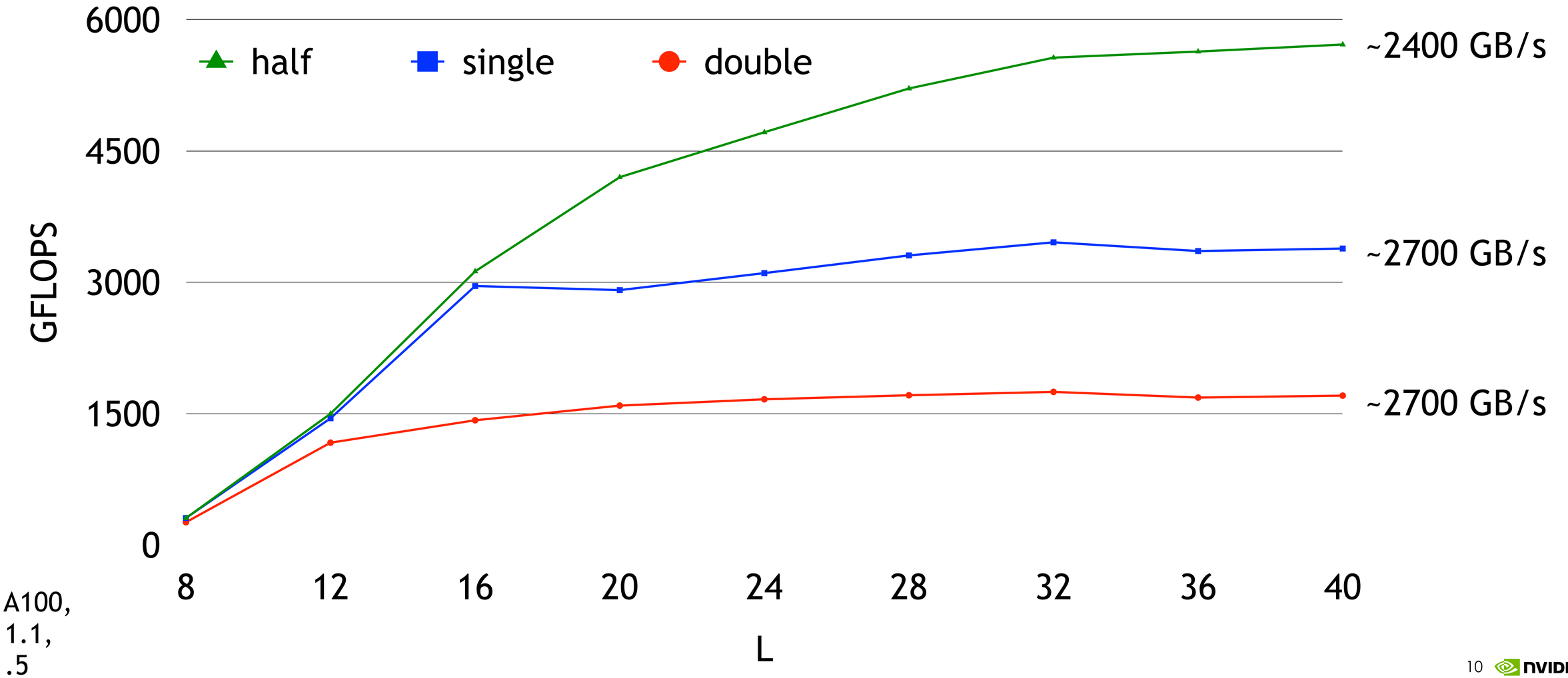
HISQ stencil (MILC, A100-80)



NVIDIA A100,
CUDA 11.1,
GCC 11.5

SINGLE GPU PERFORMANCE

Wilson-clover stencil (Chroma, A100-80)



NVIDIA A100,
CUDA 11.1,
GCC 11.5

IEEE FLOATING-POINT NUMBERS

```
struct float32_t {
  unsigned int mantissa : 23;
  unsigned int exponent :  8;
  unsigned int sign     :  1;
};
```

$$(-1)^{b_{31}} \times 2^{(b_{30}b_{29}\ldots b_{23})_2 - 127} \times (1.b_{22}b_{21}\ldots b_0)_2$$

FP32

- 32-bits per real
- 24-bit mantissa => Precision $\epsilon \sim 5 \times 10^{-8}$
- 8-bit exponent => Range $\in [1 \times 10^{-38}, 3 \times 10^{38}]$

FP64

- 64-bits per real
- 53-bit mantissa => Precision $\epsilon \sim 1 \times 10^{-16}$
- 8-bit exponent => Range $\in [2 \times 10^{-208}, 2 \times 10^{308}]$

QUDA “HALF” PRECISION

Gauge Field

- Element range $\in [-1,1]$
- No need to store exponent
- Store the matrix elements in 16-bit fixed-point

Fermion fields

- No *a priori* bound on the elements range
- For each site vector store max element to set range

Perform computation in FP32

16-bit local precision $\epsilon \sim 3 \times 10^{-5}$ with global FP32 range

- cf IEEE FP16: $\epsilon \sim 5 \times 10^{-4}$

3x3 Link matrix

```
struct matrix {
  int16_t v[18];
};
```

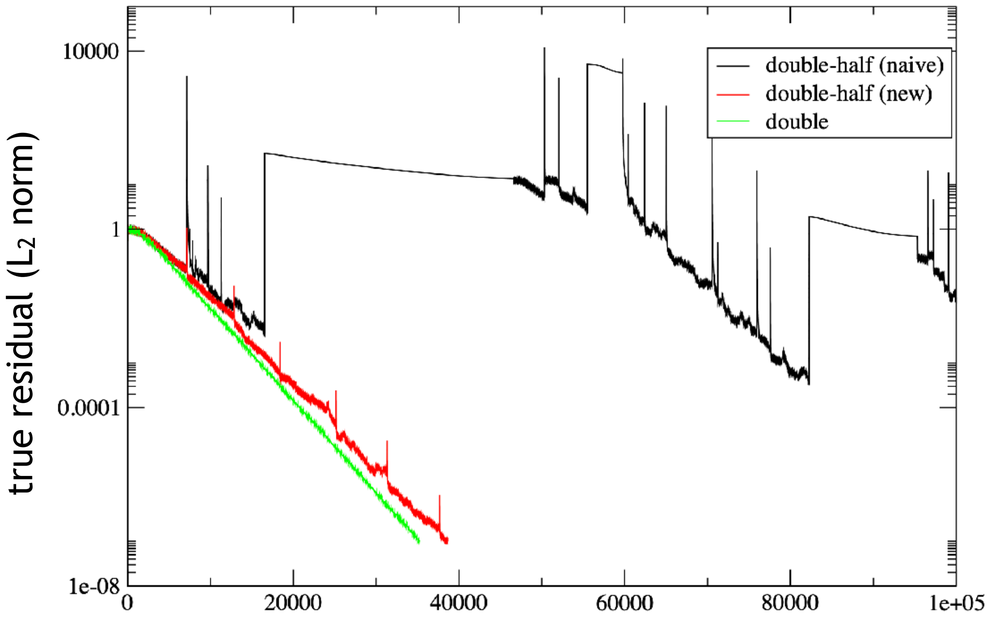
Staggered fermion

```
struct vector3 {
  int16_t v[6];
  float max;
};
```

MIXED PRECISION

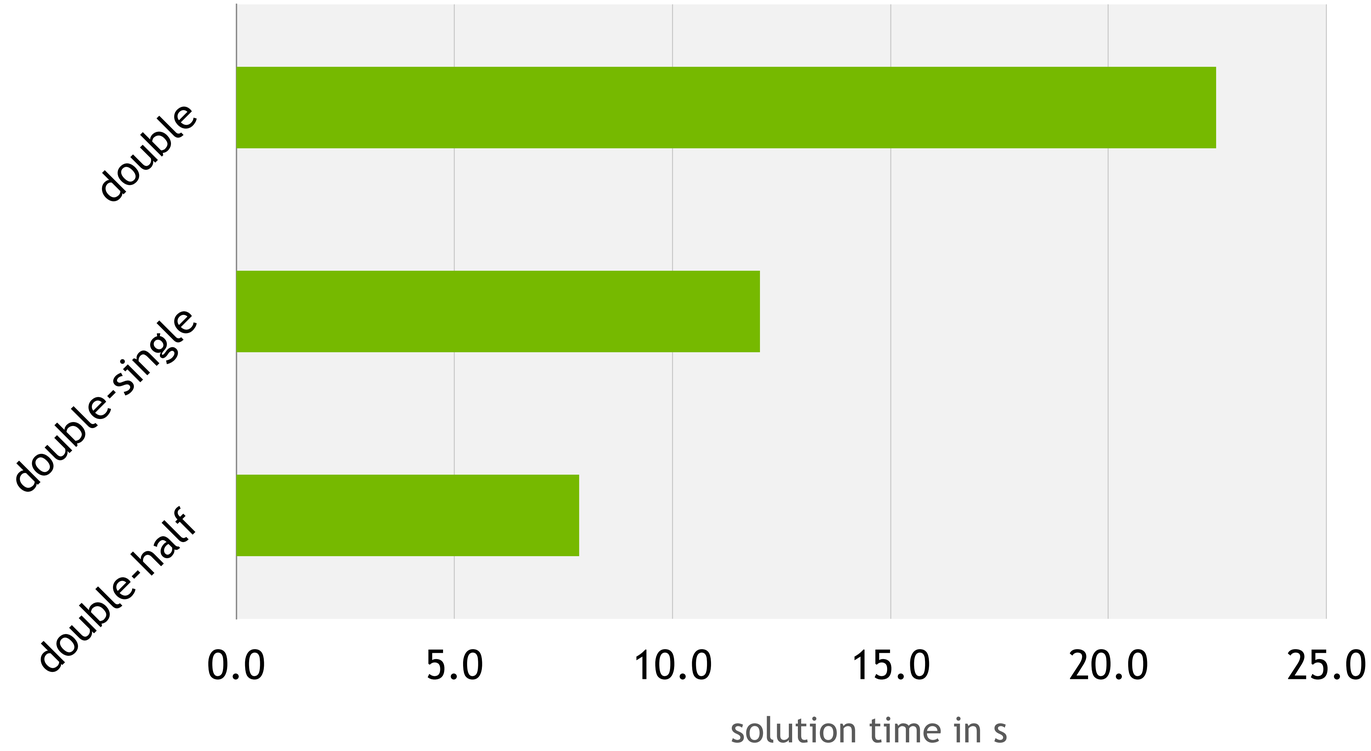
Using your bits wisely

MILC/QUDA HISQ CG, mass = 0.001 => $\kappa \sim 10^6$



Tesla V100,
CUDA 10.1,
GCC 7.3,
QUDA 1.0

MILC/QUDA HISQ CG solver



MIXED-PRECISION CG

double-half

- Reliable update: periodic replacement of the residual with true residual in high precision

- Maintain solution vectors in high precision
  - Including the partial accumulator

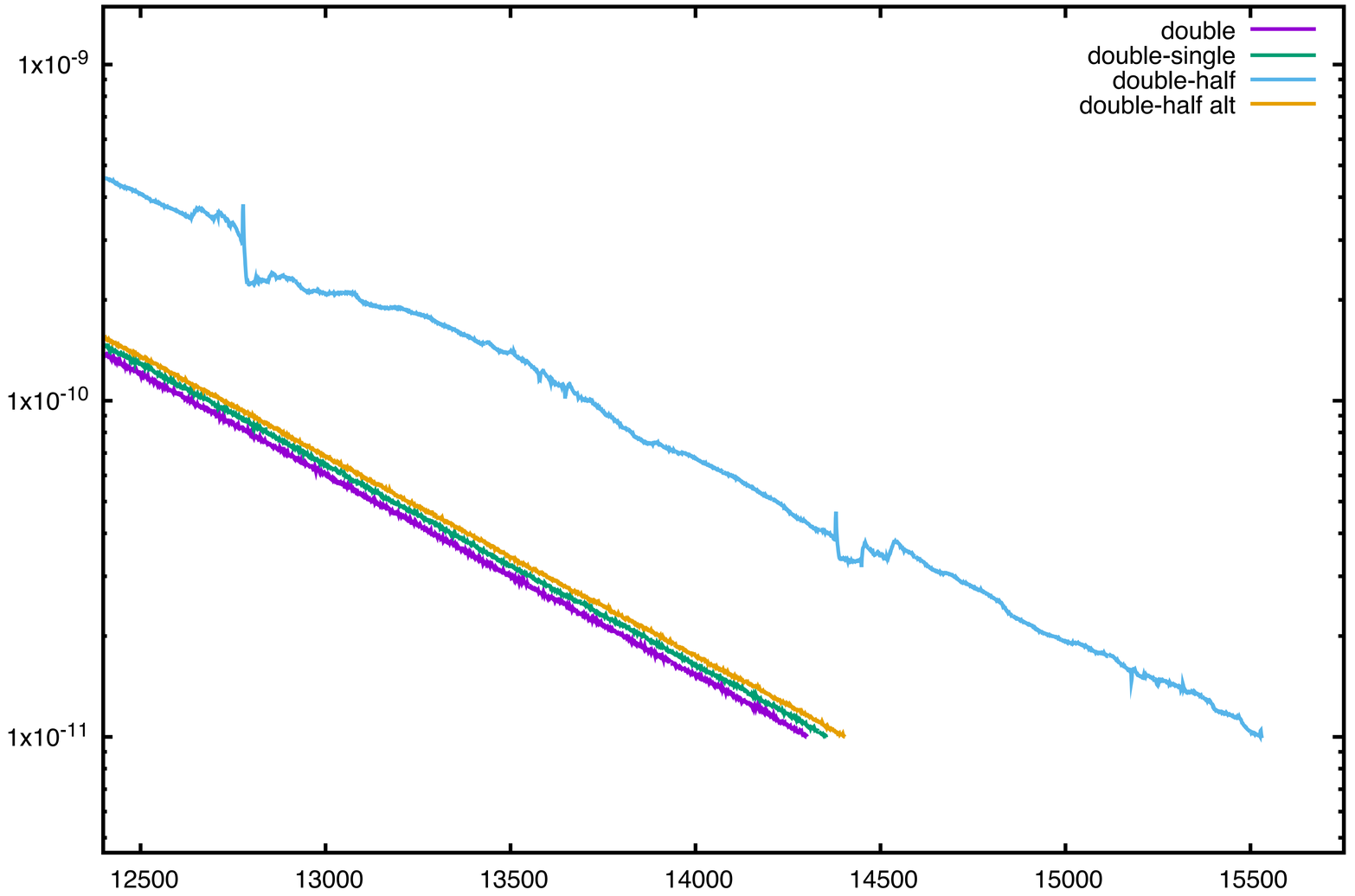
- When true residual is injected, re-project the direction vector

- Use Polak-Ribière formula

$$\beta_k := \frac{\mathbf{z}_{k+1}^{\mathsf{T}} \left(\mathbf{r}_{k+1} - \mathbf{r}_k \right)}{\mathbf{z}_k^{\mathsf{T}} \mathbf{r}_k}$$

double-half alt

- Residual replacement strategy of van der Worst and Ye



MORE PRECISION AT CONSTANT BITS

Using your bits even more wisely

$\epsilon \sim 3 \times 10^{-5}$

```
struct vector3_half {
  int16_t v[6];
  float max;
};
```

128 bits

$\epsilon \gtrsim 3 \times 10^{-6}$

```
struct spinor_20 {
  int20_t v[6];
  uint8_t exponent;
};
```

```
struct spinor3_fp32 {
  float v[6];
};
```

192 bits

```
struct spinor_30 {
  int30_t v[6];
  uint8_t exponent;
};
```

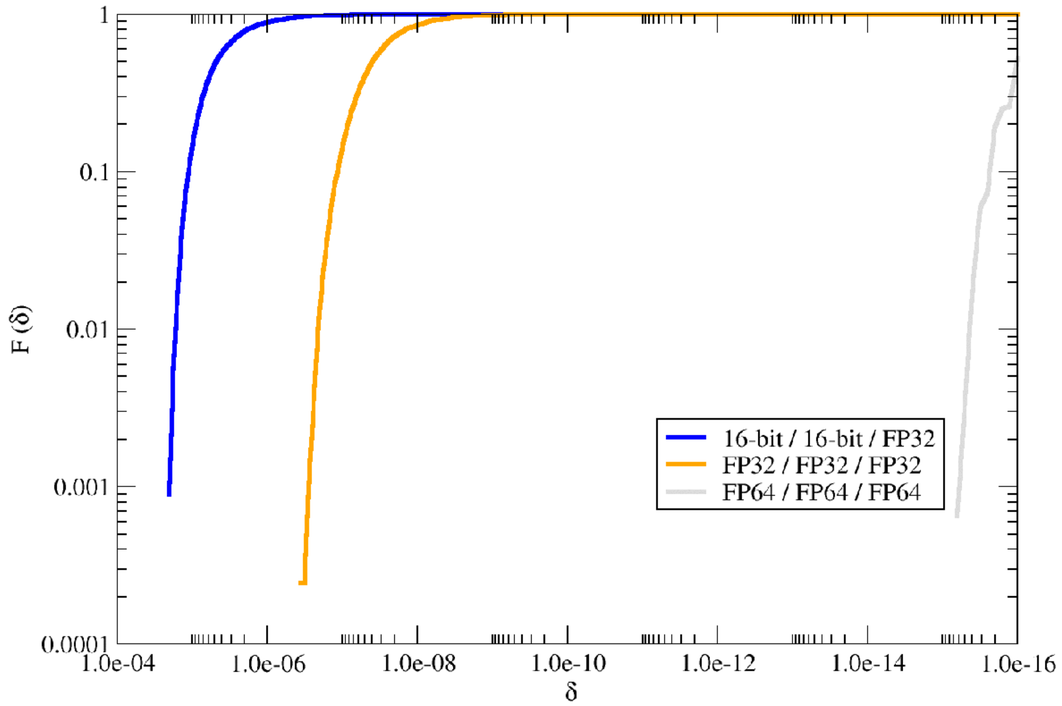
$\epsilon \sim 1 \times 10^{-7}$

$\epsilon \gtrsim 2 \times 10^{-9}$

HOW WELL DOES IT WORK?

Precision: gauge / fermion / compute

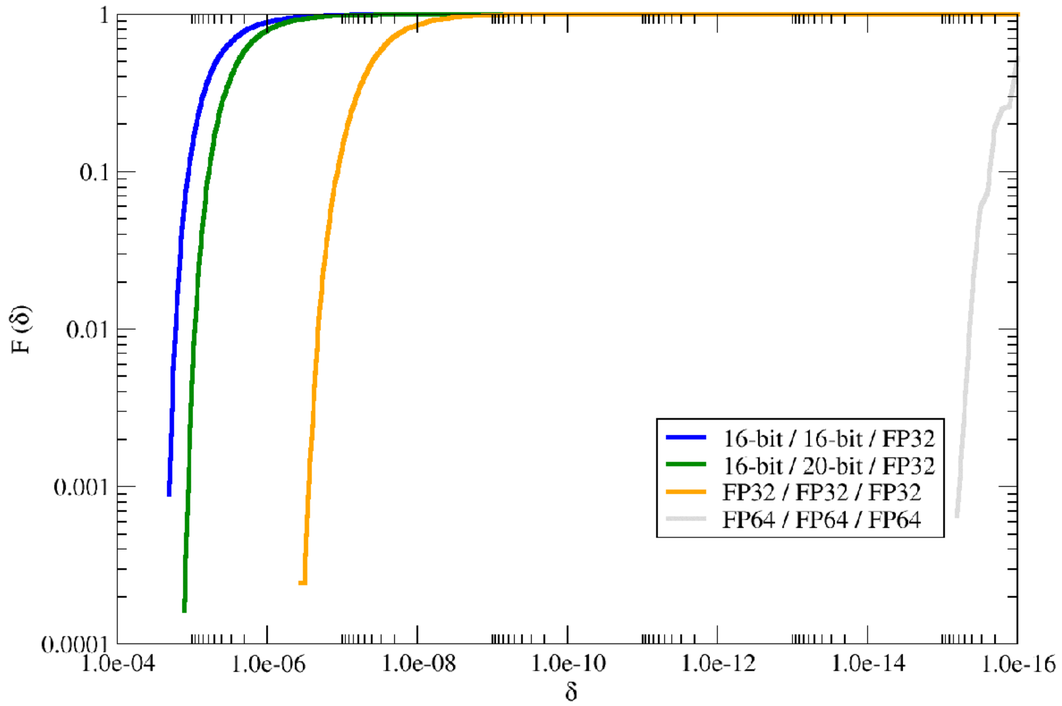
HISQ Dslash element-by-element absolute deviation CDF vs FP64 reference



HOW WELL DOES IT WORK?

Precision: gauge / fermion / compute

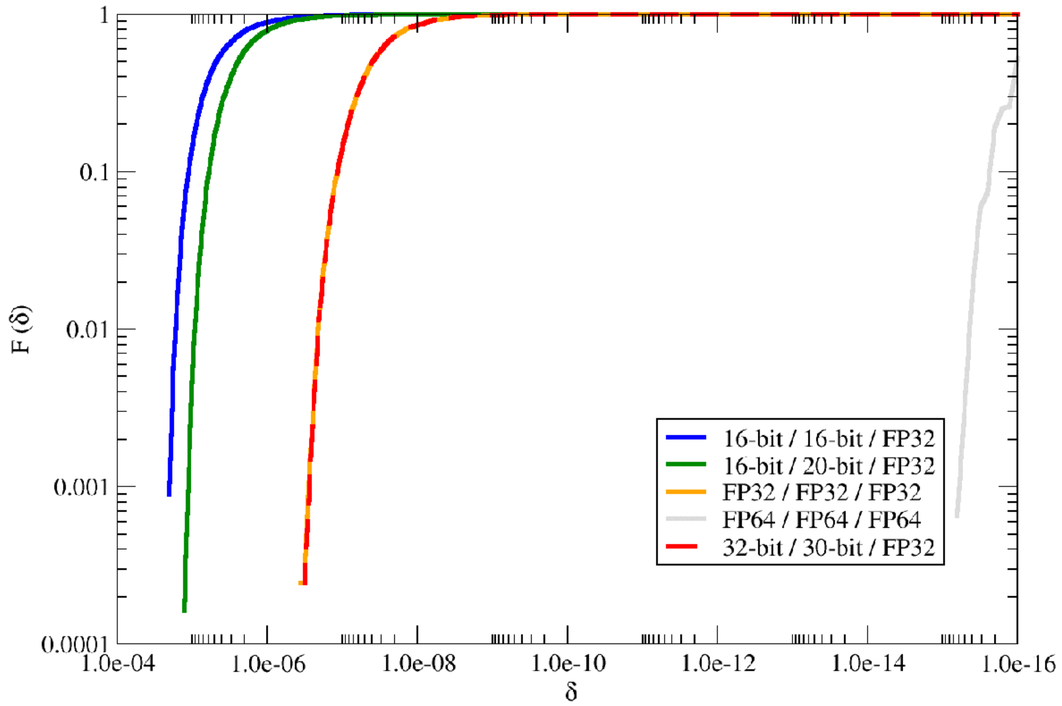
HISQ Dslash element-by-element absolute deviation CDF vs FP64 reference



HOW WELL DOES IT WORK?

Precision: gauge / fermion / compute

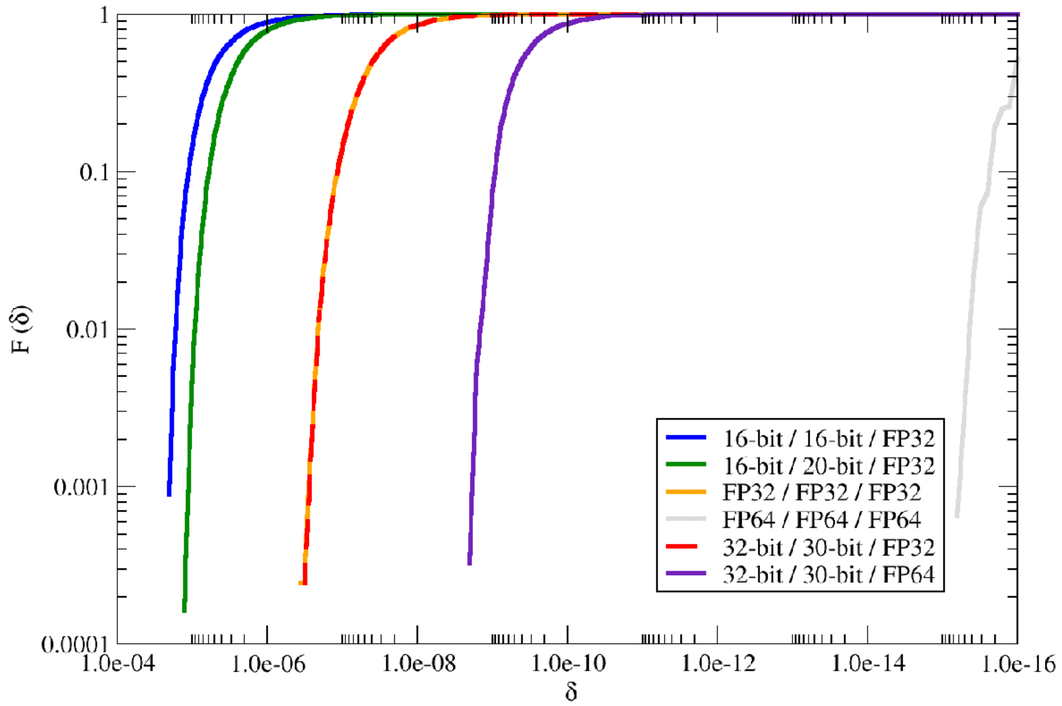
HISQ Dslash element-by-element absolute deviation CDF vs FP64 reference



HOW WELL DOES IT WORK?

Precision: gauge / fermion / compute

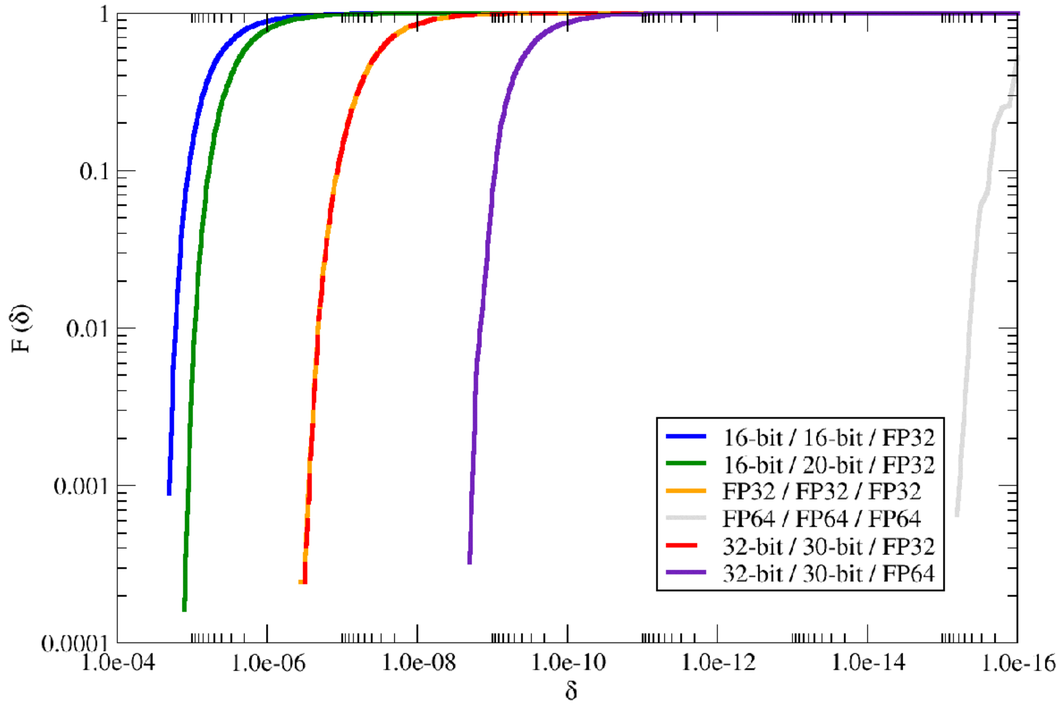
HISQ Dslash element-by-element absolute deviation CDF vs FP64 reference



HOW WELL DOES IT WORK?

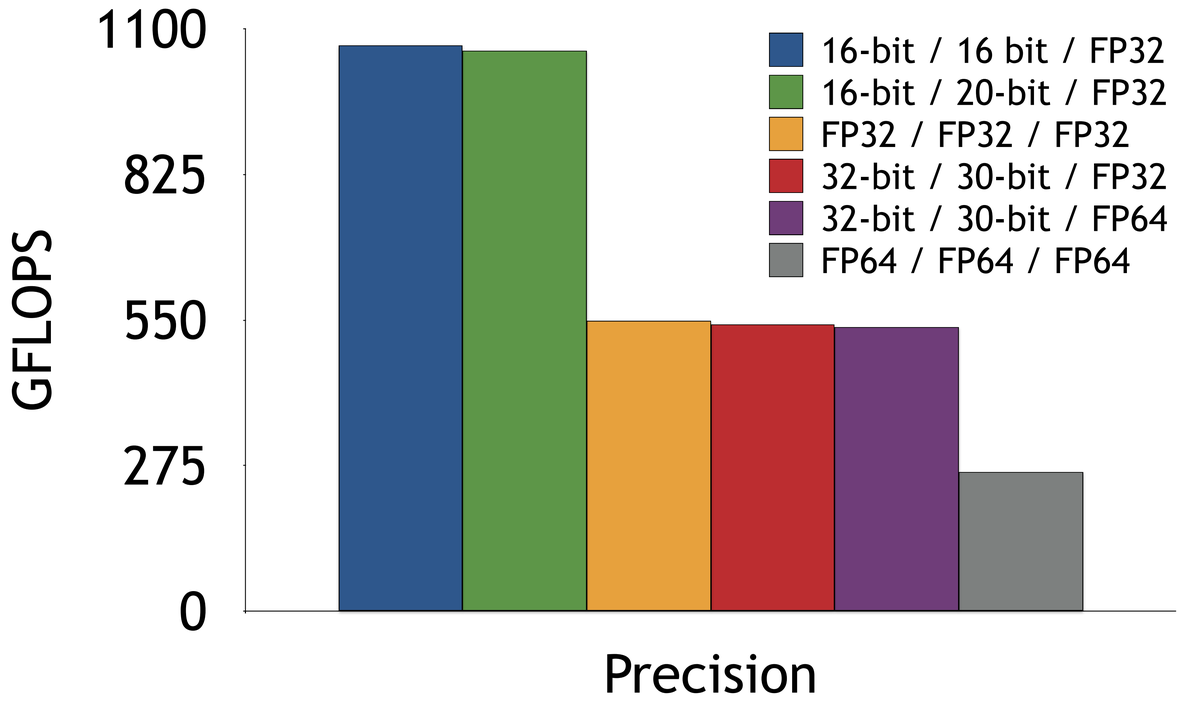
Precision: gauge / fermion / compute

HISQ Dslash element-by-element absolute deviation CDF vs FP64 reference



HISQ Dslash Performance

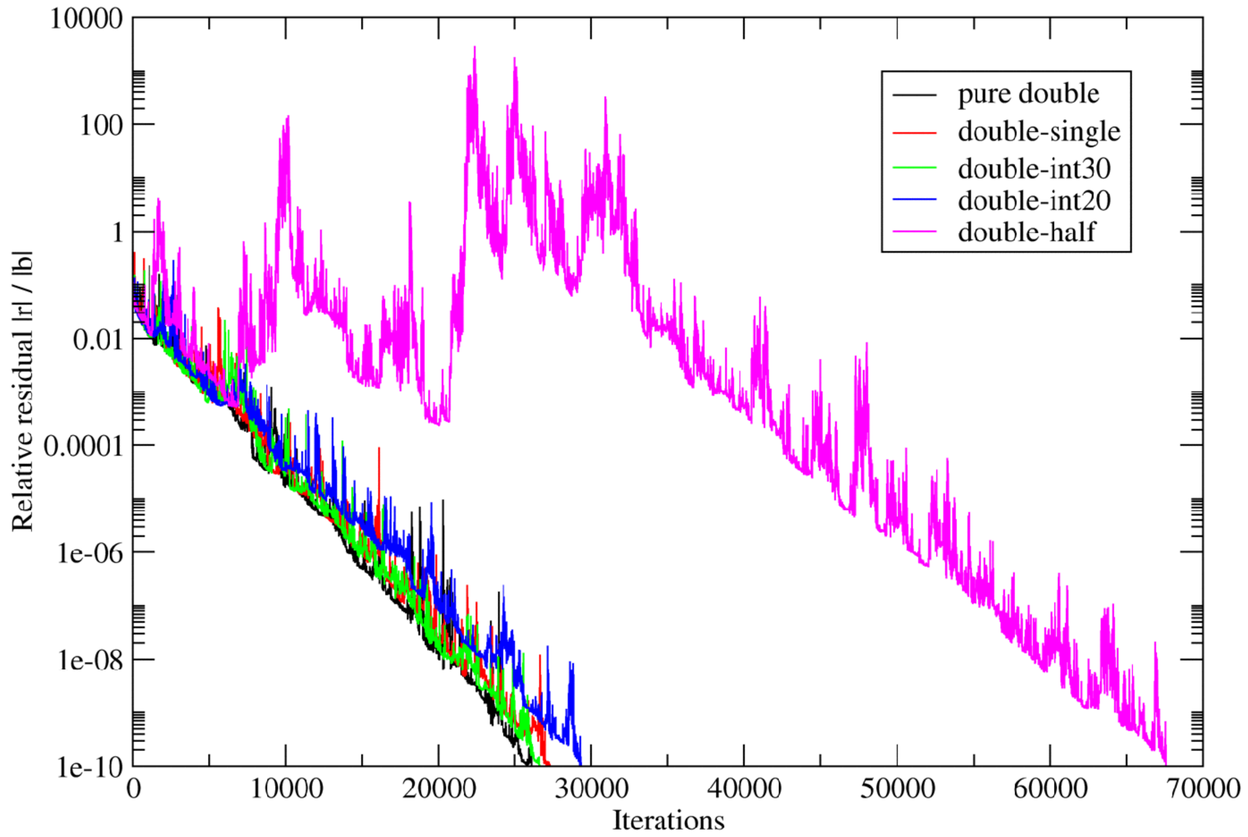
V = 32^4, Quadro GV100



Gain two orders of magnitude in precision with no performance cost

BICGSTAB(4)

HISQ, V = 36^3x72, β = 6.3, m = 0.001



	Iterations	Time (s)
pure double	26064	307
double-single	27308	159
double-int30	26580	150
double-int20	29336	106
double-half	67552	247

MULTI-SHIFT CG SOLVER

Used for RHMC and multi-mass solver propagators

Mixed-precision multi-shift CG

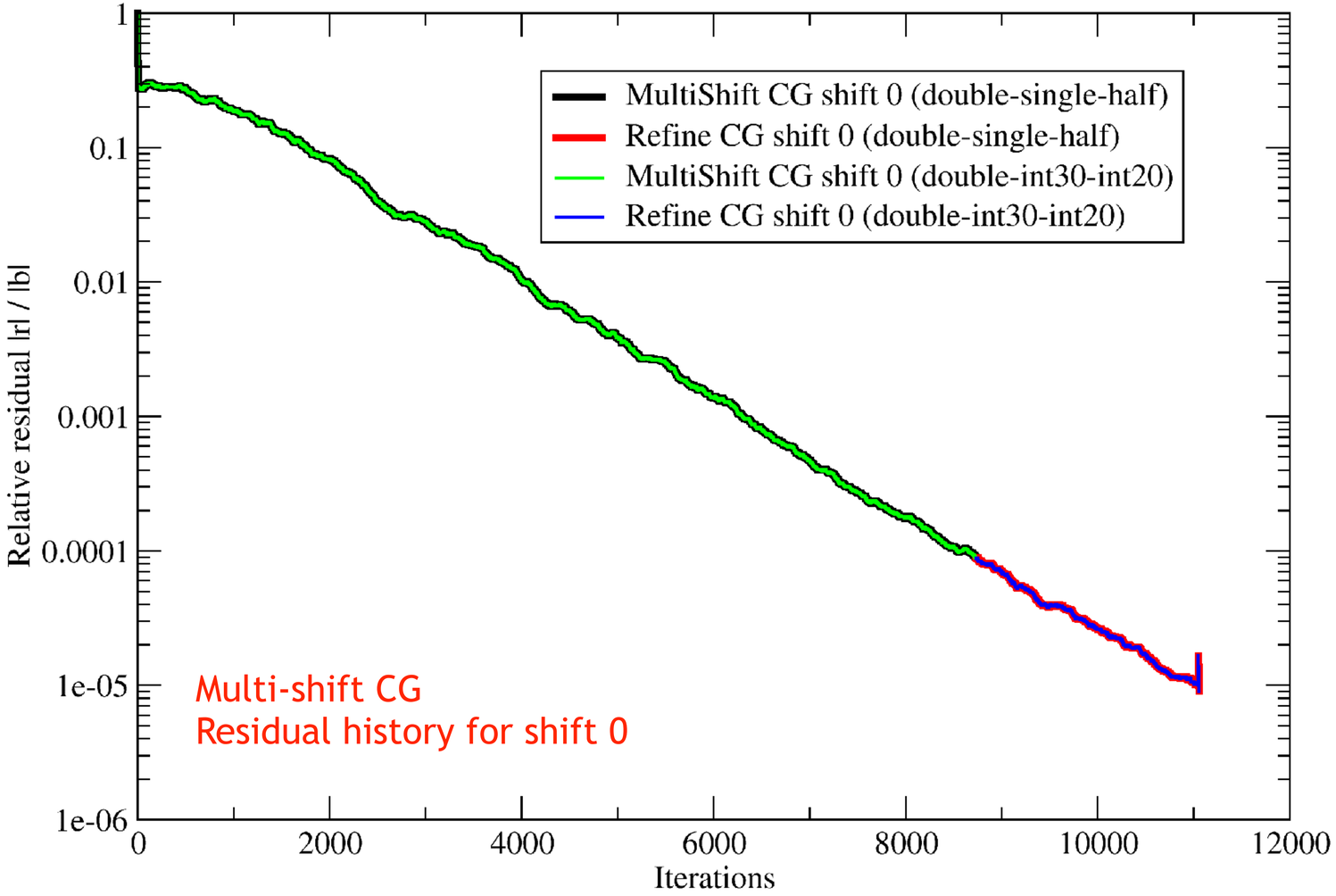
- Essentially mixed-precision CG on shift 0
- Shifted iterated residuals drift away true residual
- Refine each shifted system to correct for lack of residual collinearity
- Many additional iterations can be required

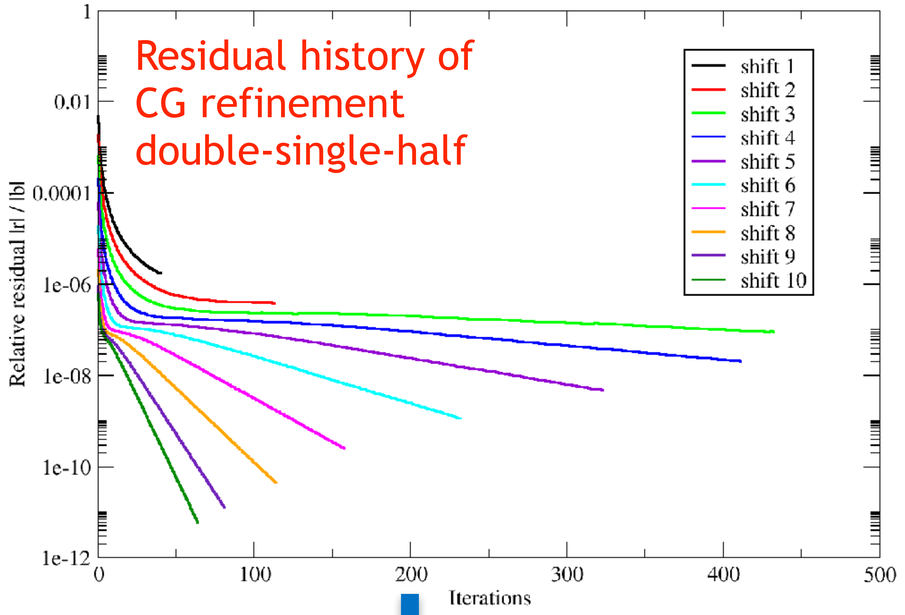
Prior optimal QUDA strategy

- double-single multi-shift-CG
- double-half per shift refinement

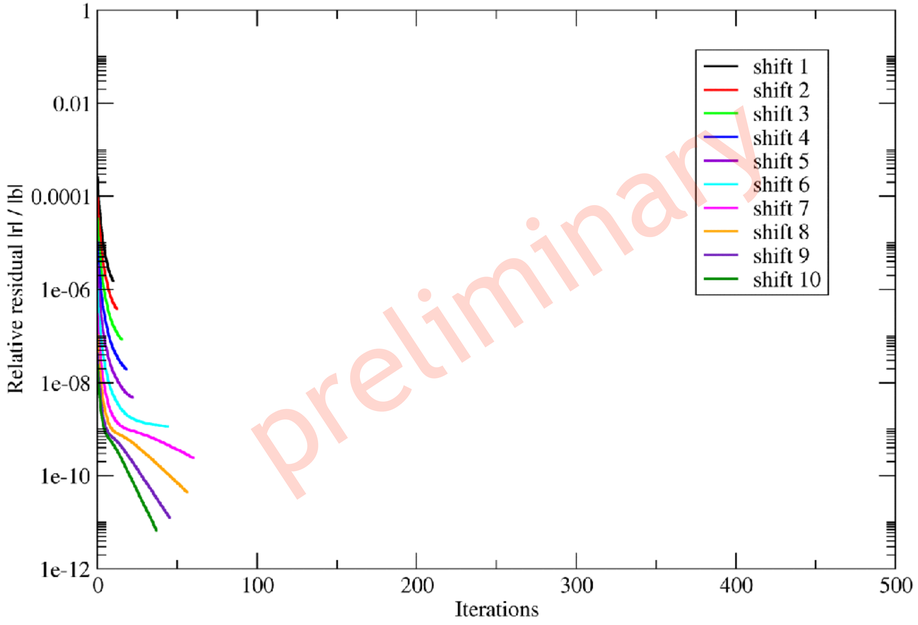
MULTI-SHIFT SOLVER

HISQ RHMC, V = 36^3x72, β = 6.3, m = 0.001, 11 shifts



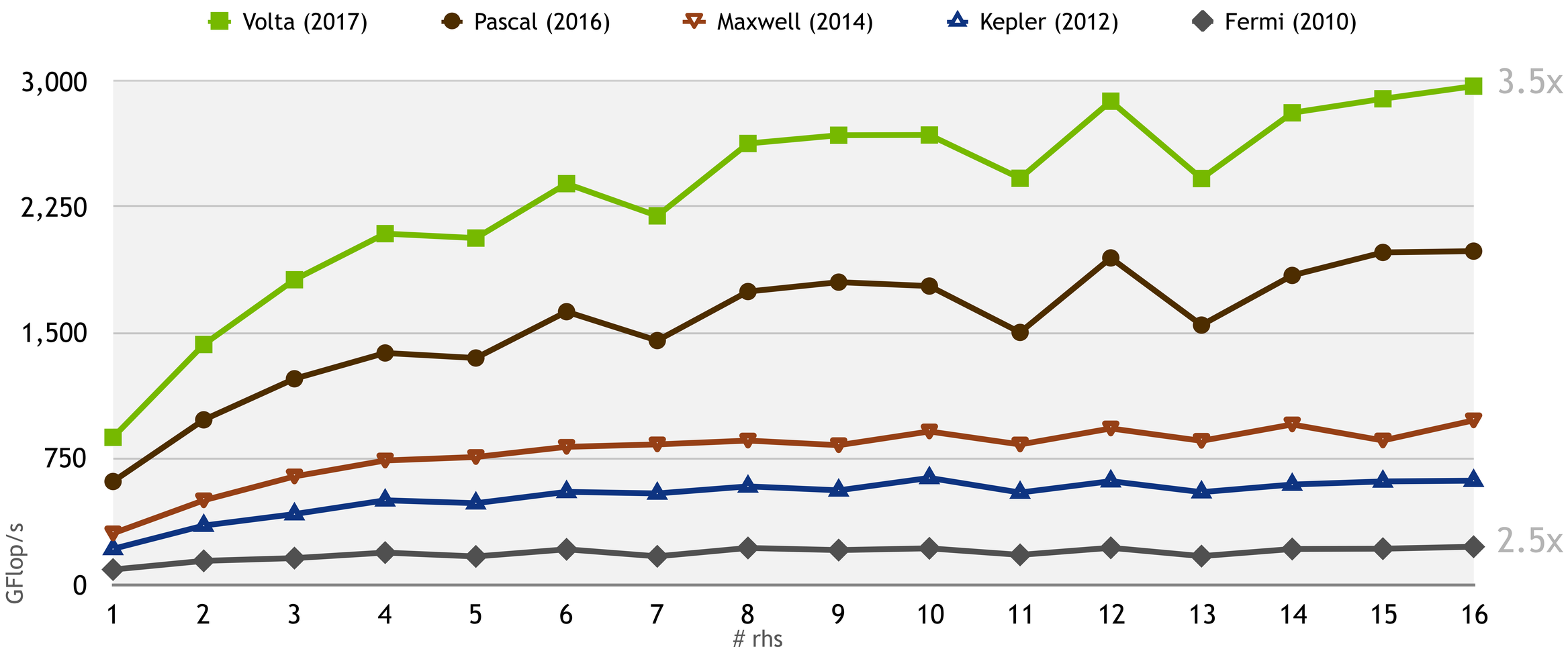


Significant reduction in refinement iterations



DO EVEN MORE WITH YOUR BITS?

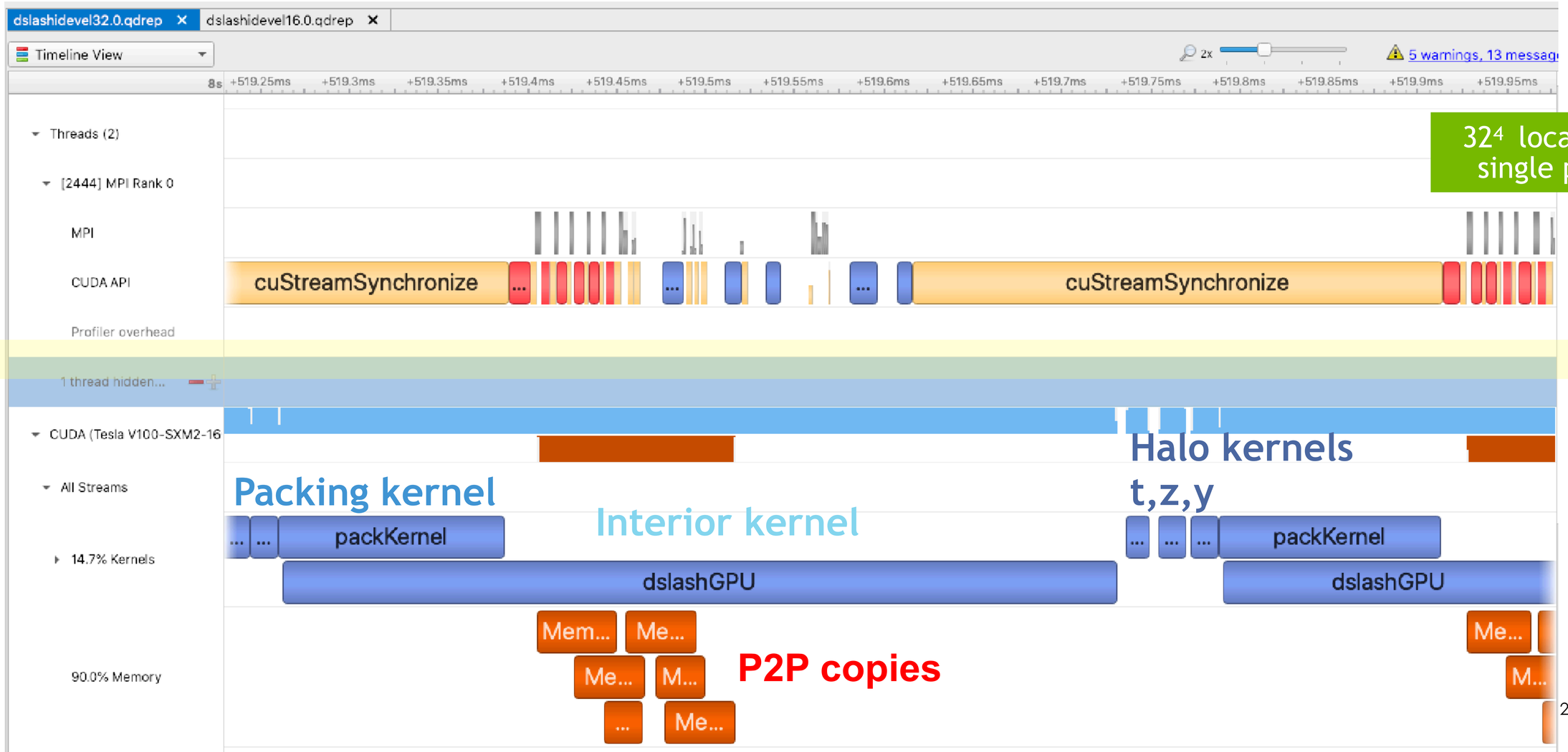
Multiple RHS



GPU-CENTRIC COMMUNICATION

MULTI-GPU PROFILE

overlapping comms and compute



FASTER ON A GPU

How do we scale that?

Faster GPU

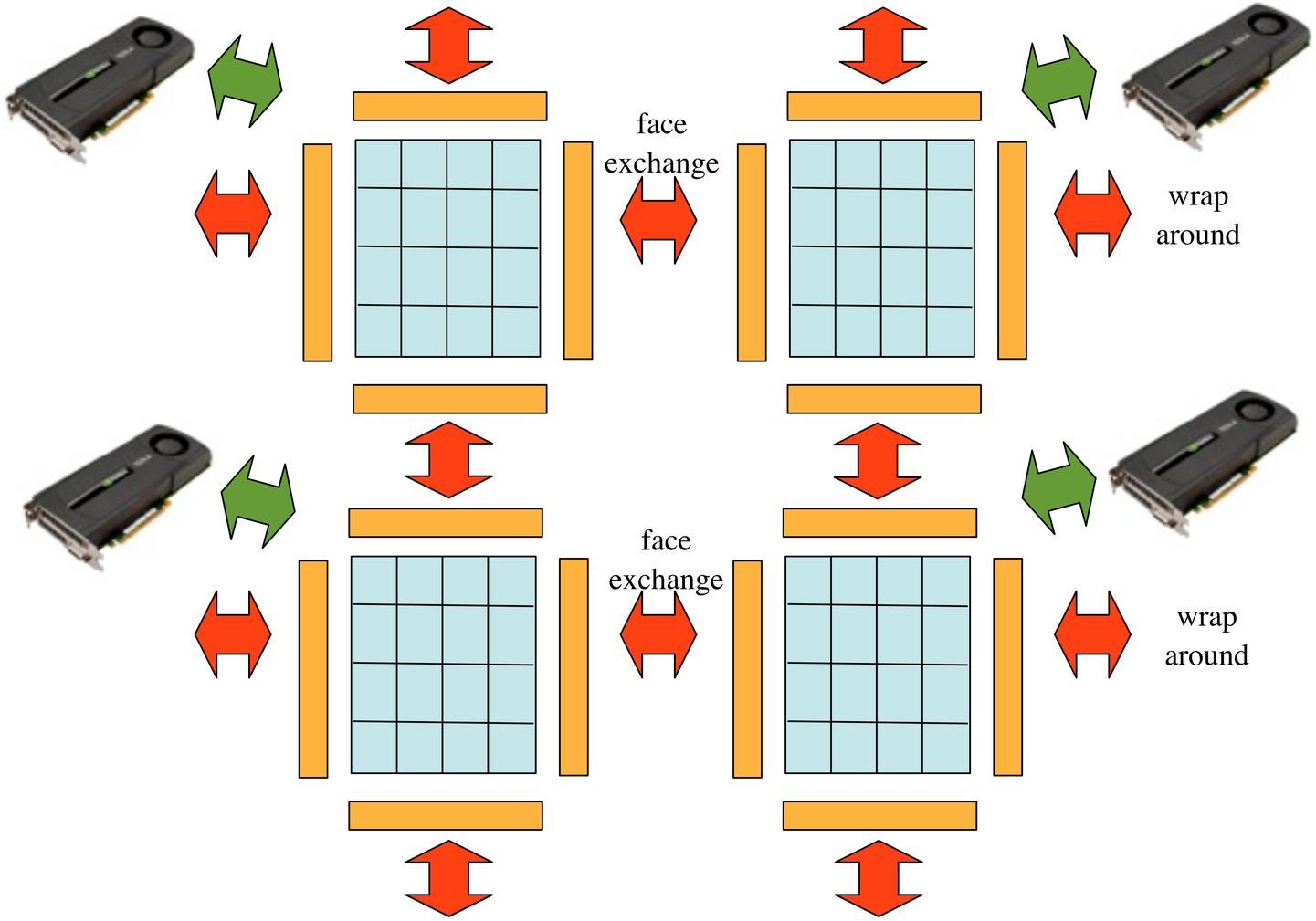
Less Bits

More GPU

-> Shorter Kernels

Less time to communicate

-> Need more network



NVLINK SWITCH SYSTEM

Purpose Built High Performance NVLink Network For Up to 256 GPUs

4th GEN NVLINK

900 GB/s from 18x25GB/sec bi-directional ports
GPU-2-GPU connectivity across nodes

3rd GEN NVSWITCH

All-to-all NVLink switching for 8-256 GPUs
Accelerate collectives - multicast and SHARP

NVLINK SWITCH

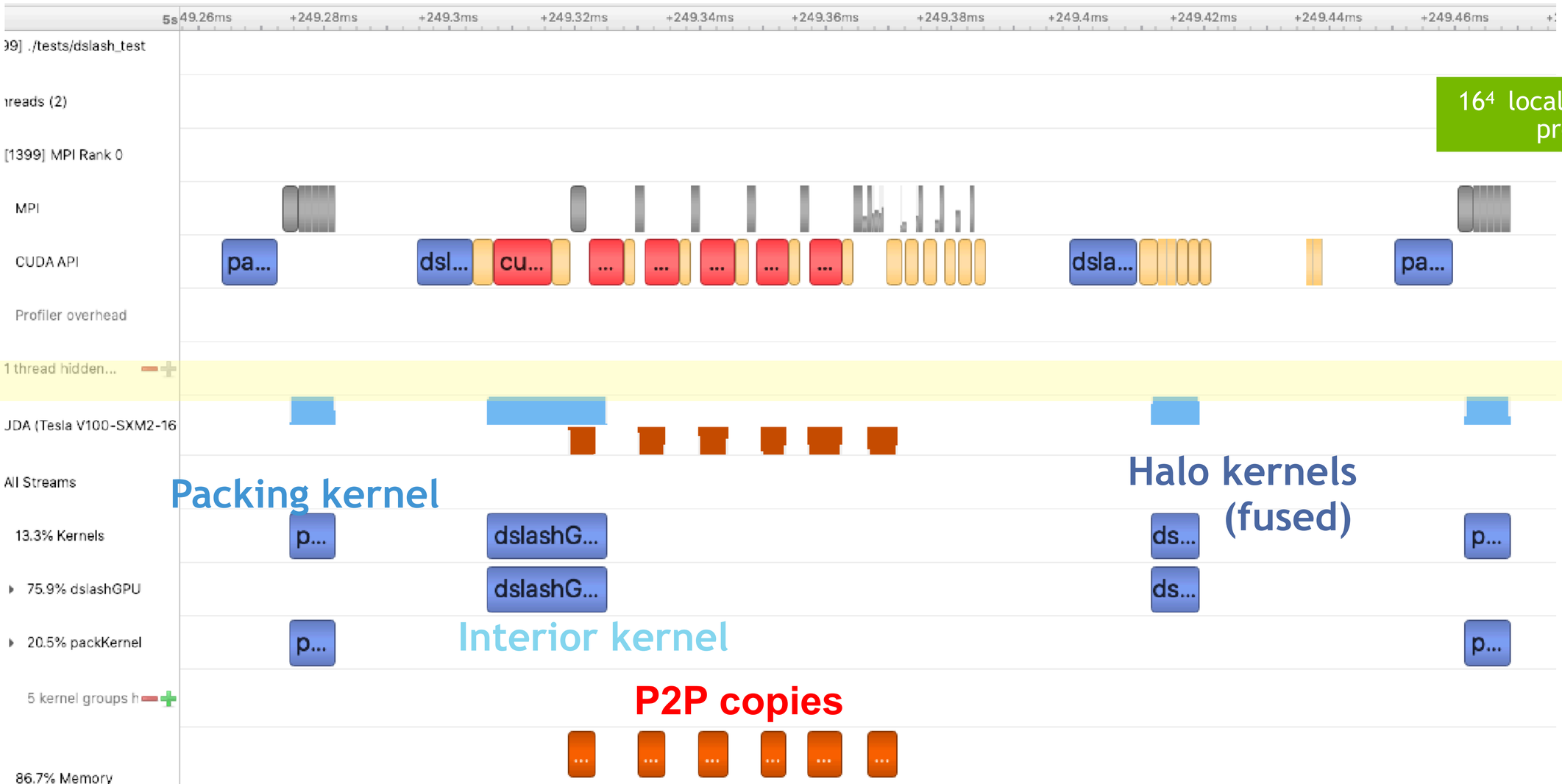
128 port cross-connect based on NVSwitch

H100 CLUSTER (1 SCALABLE UNIT)

57,600 GB/s all-to-all bandwidth
32 servers | 18 NVLink switches | 1,152 NVLink optical cables

STRONG SCALING PROFILE

~~overlapping~~ comms and compute



16^4 local volume, half precision

DGX-1, 1x2x2x2 partitioning

NVIDIA.

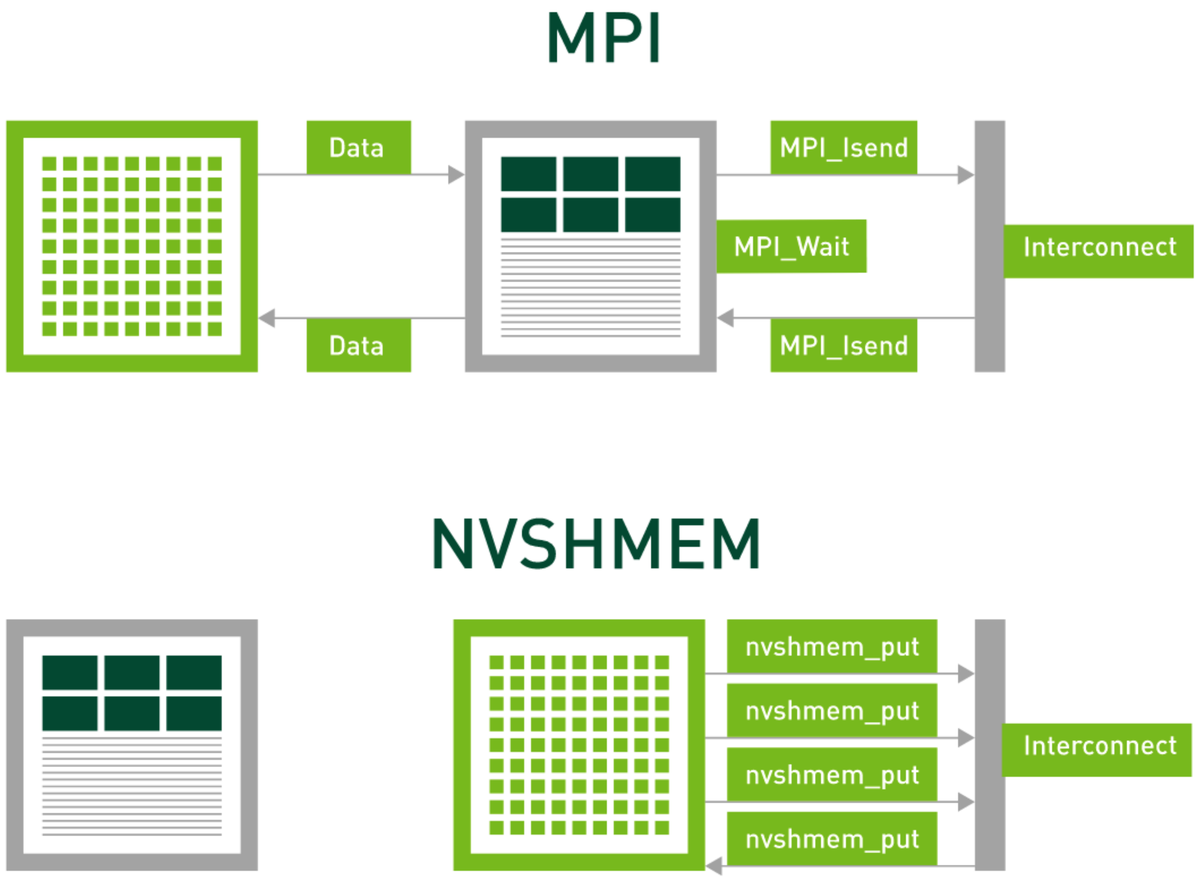
NVSHMEM: OpenSHMEM FOR CLUSTERS OF NVIDIA GPUS

- Compute on GPU
- Communication from GPU

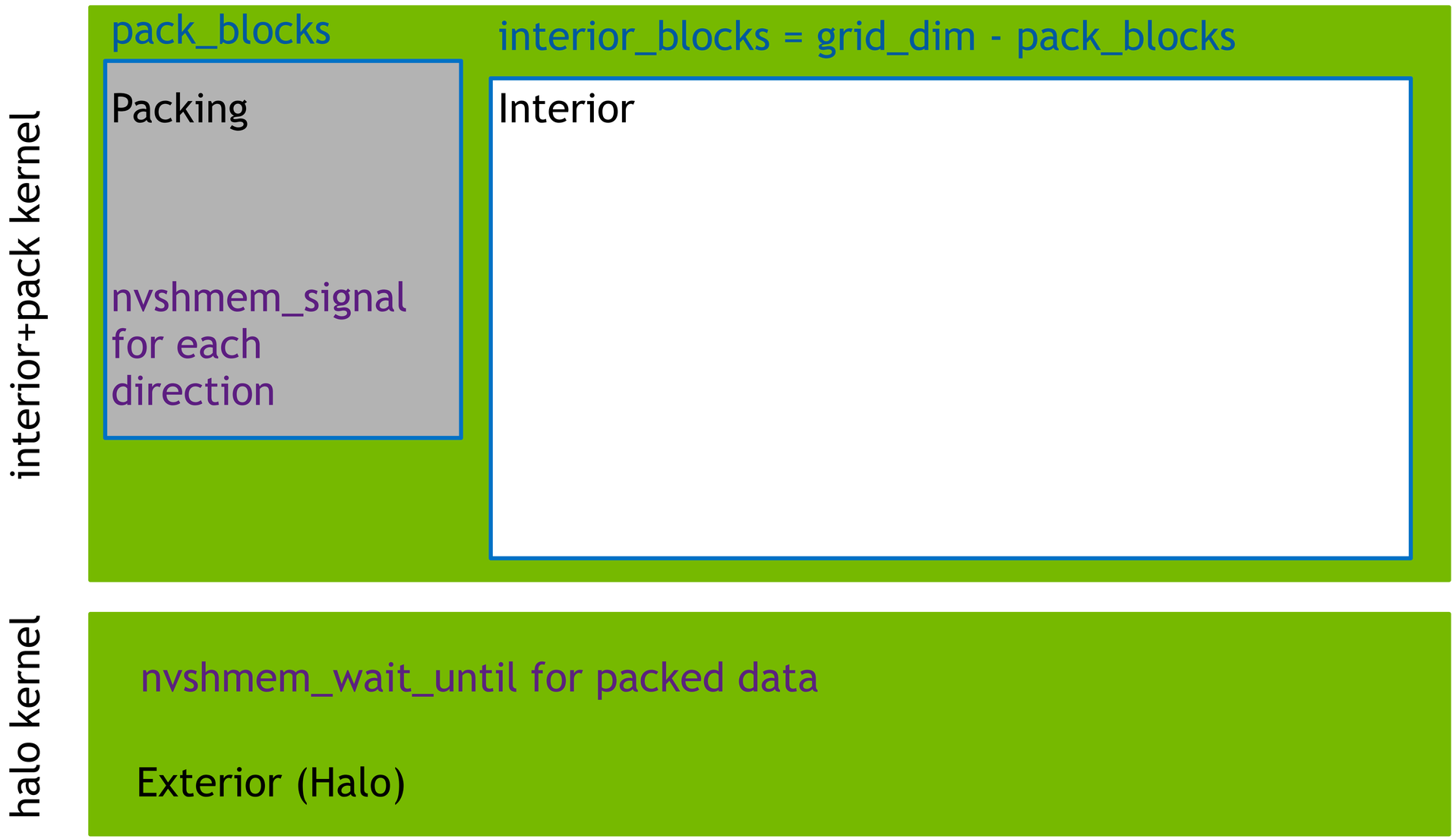
Benefits:

- Eliminates offload latencies
- Improves overlap of computation and communication
- Hides latencies using multithreading
- Easier to express scalable algorithms with inline communication

NVSHMEM's Partitioned Global Address Space (PGAS) model improves performance while making it easier to program

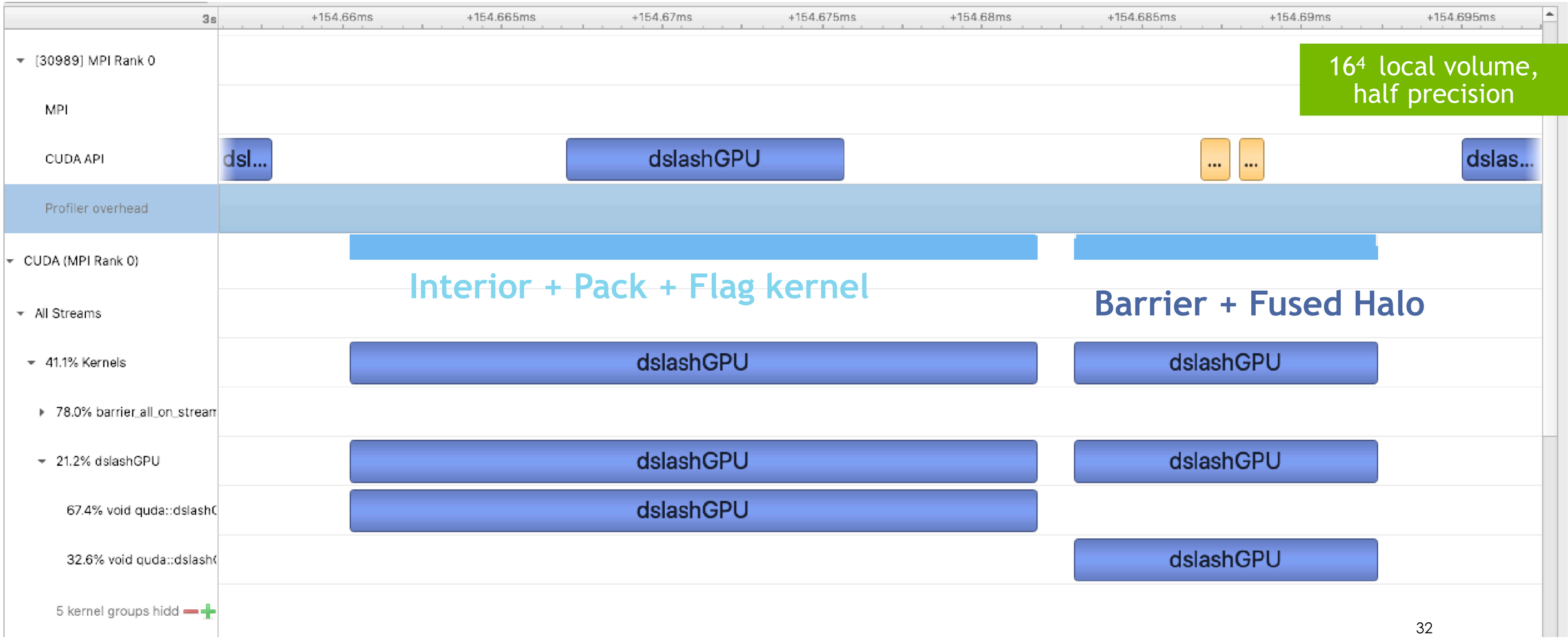


FUSED DSLASH + PACKING KERNEL



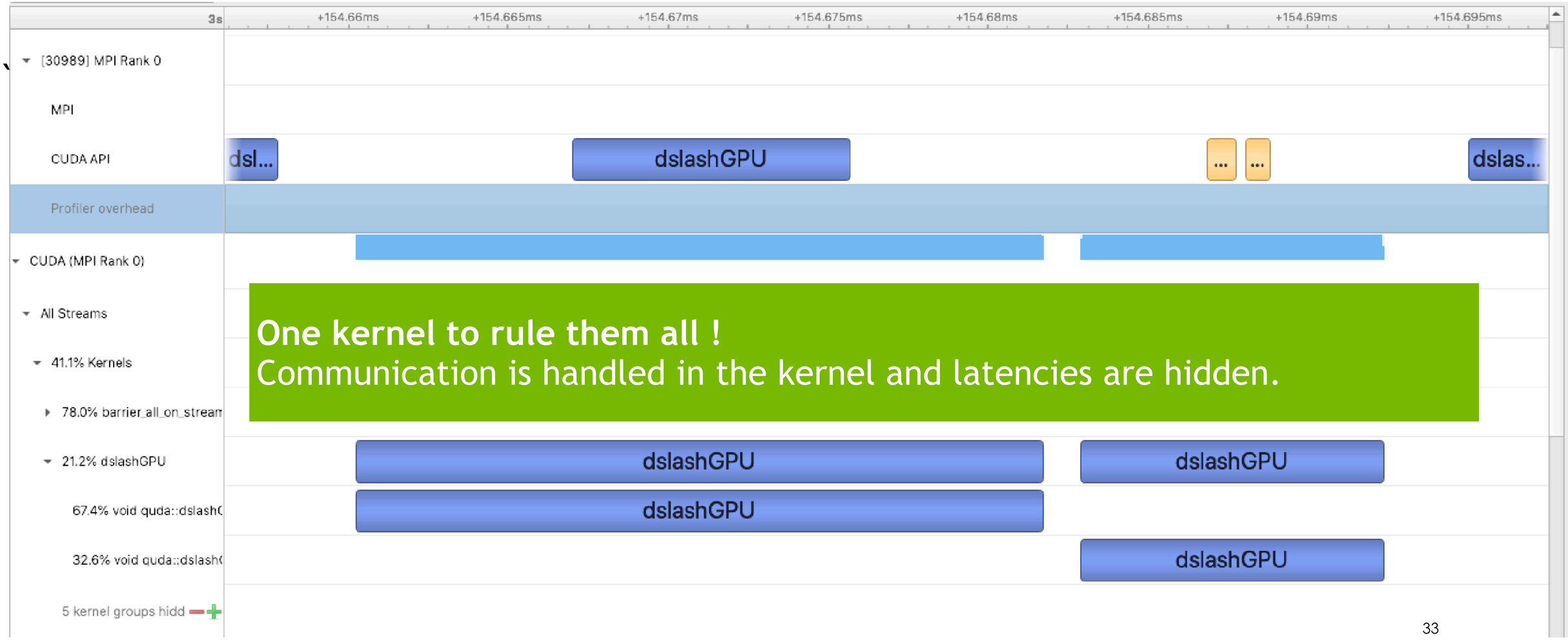
NVSHMEM + FUSING KERNELS

no extra packing and barrier kernels needed



DGX-1, 1x2x2x2 partitioning

CAN WE GO FURTHER ?



FINE-GRAINED SYNCHRONIZATION

libcu++ gives us std::atomic in CUDA

Need replacement for kernel boundaries

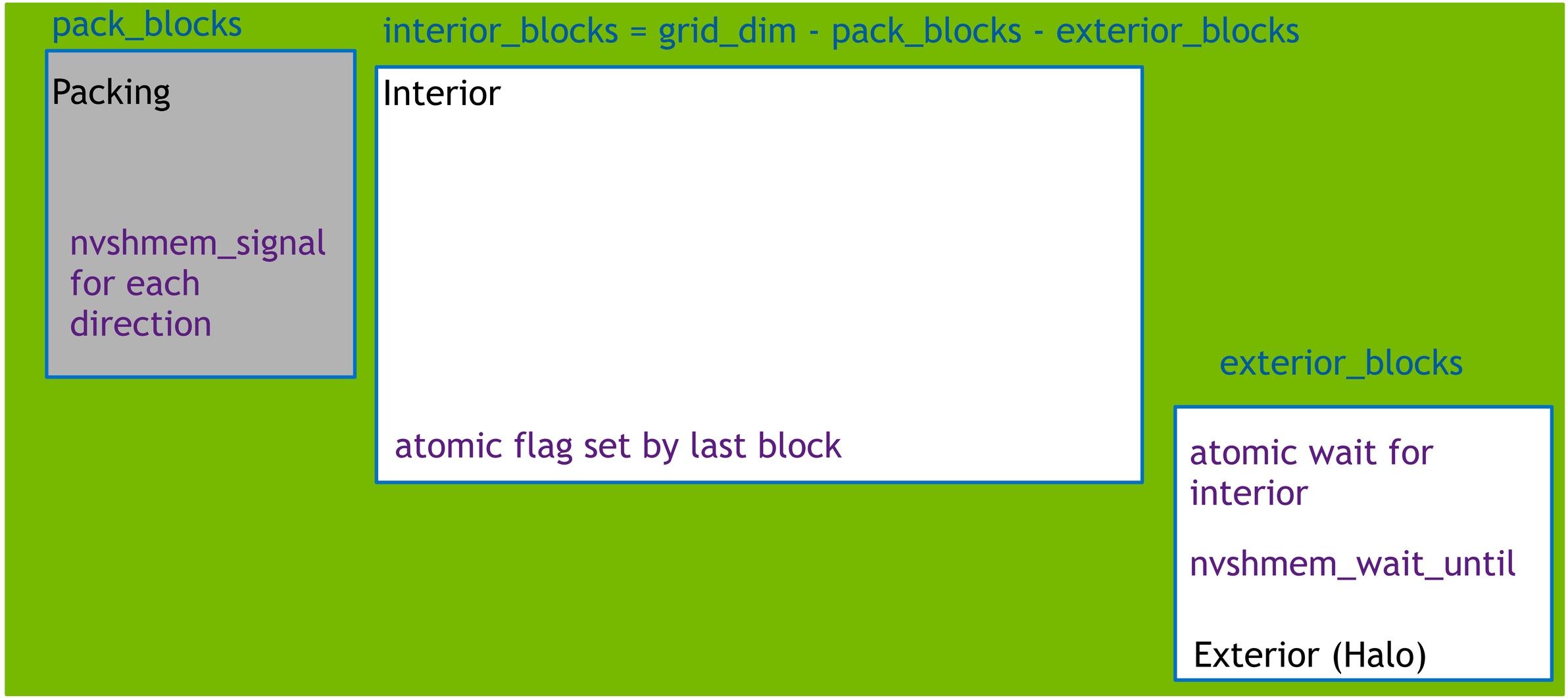
Packing is independent of interior

interior and exterior update on boundary
→ possible race condition

use cuda::atomic from libcu++

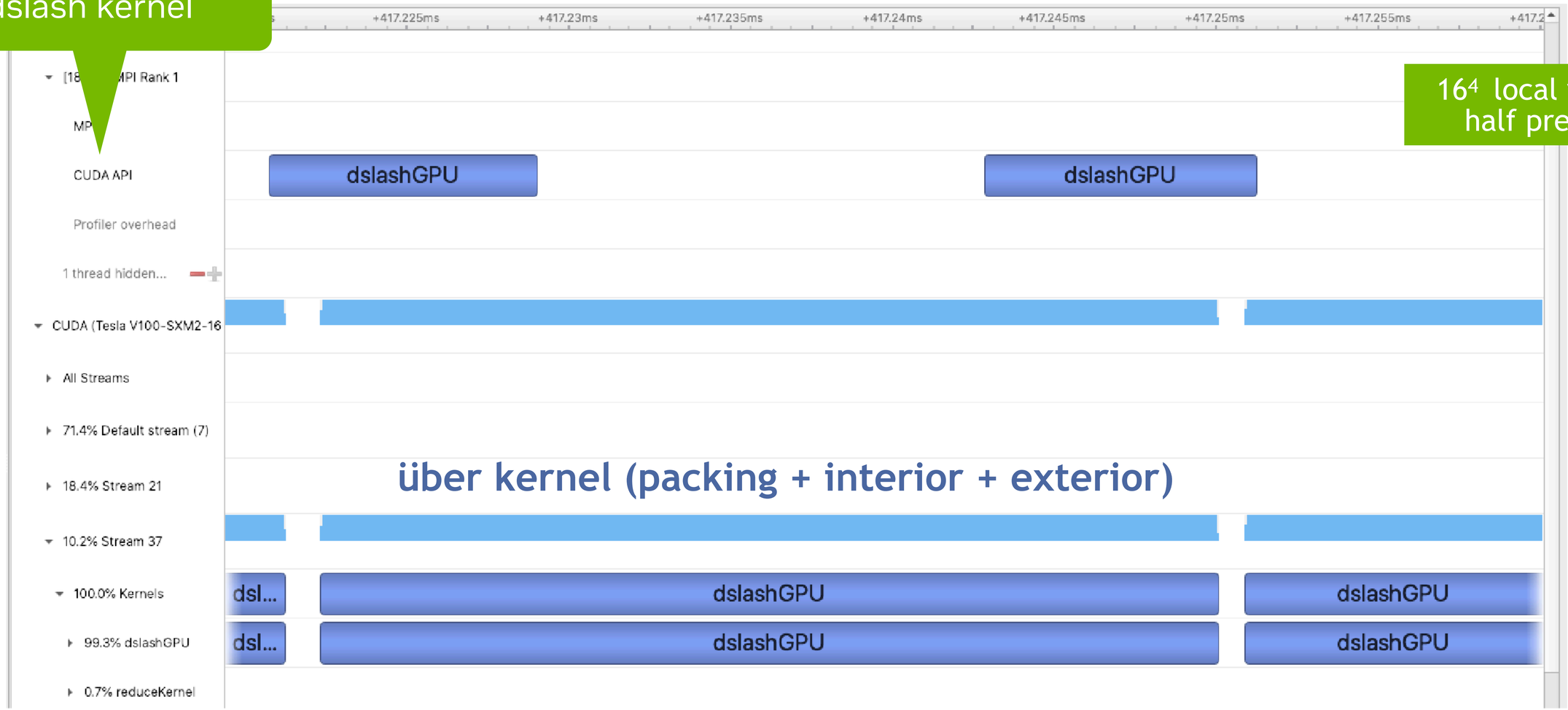
```
#include <atomic>
std::atomic<int> x;

#include <cuda/std/atomic>
cuda::std::atomic<int> x;

#include <cuda/atomic>
cuda::atomic<int, cuda::thread_scope_block> x;
```

FULLY FUSED DSLASH KERNEL



FULLY FUSED KERNEL

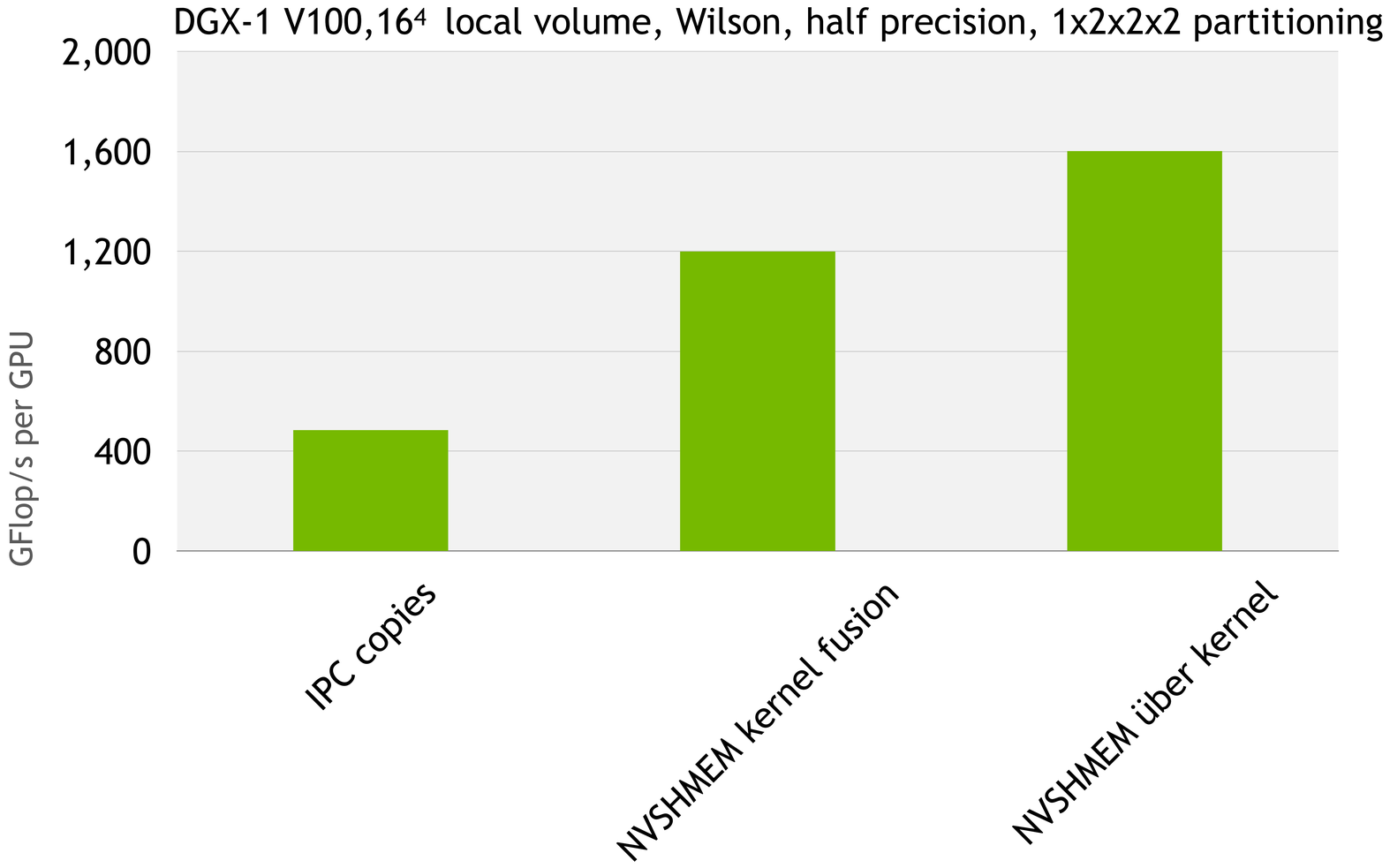
CPU now only launches the dslash kernel



16^4 local volume, half precision

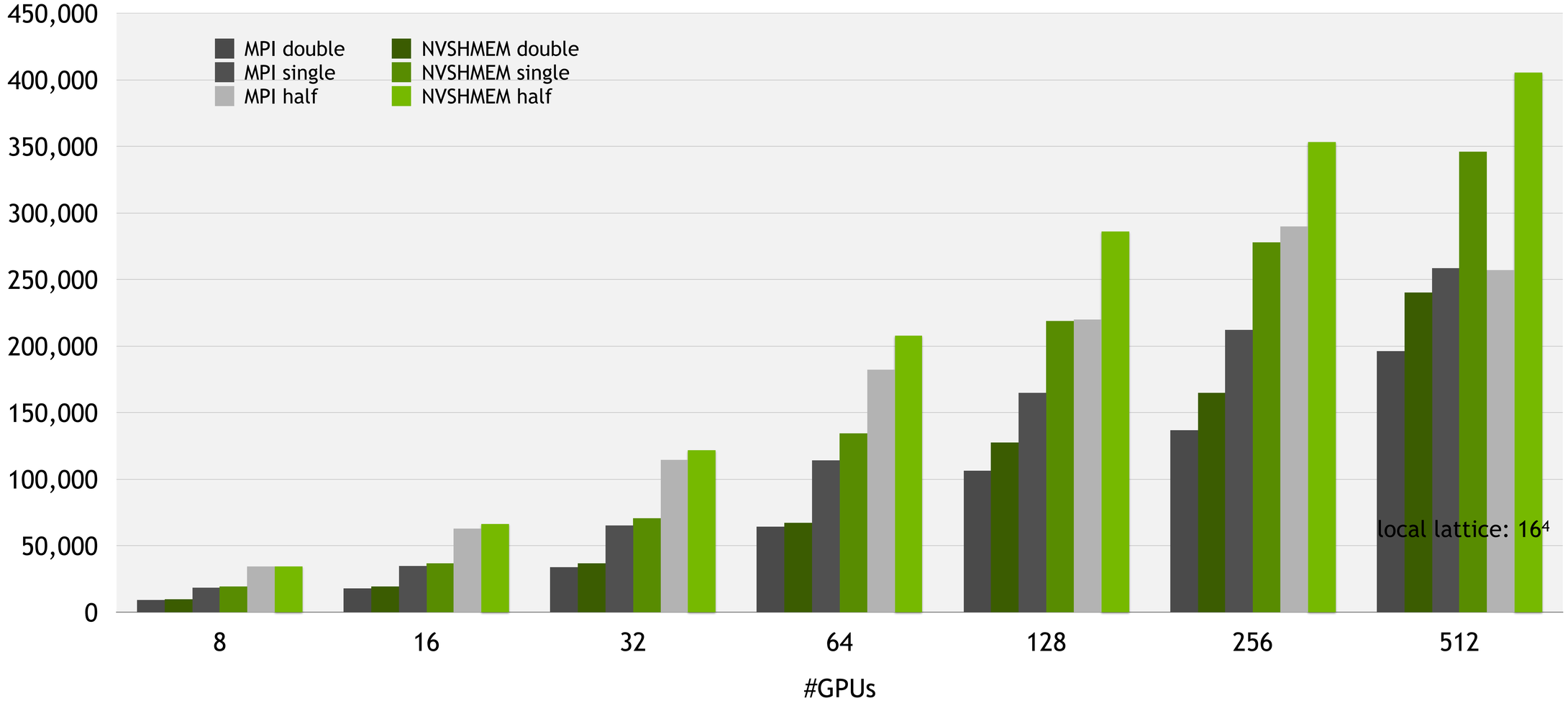
DGX-1, 1x2x2x2 partitioning

LATENCY REDUCTIONS



SELENE (DGX A100-80) STRONG SCALING

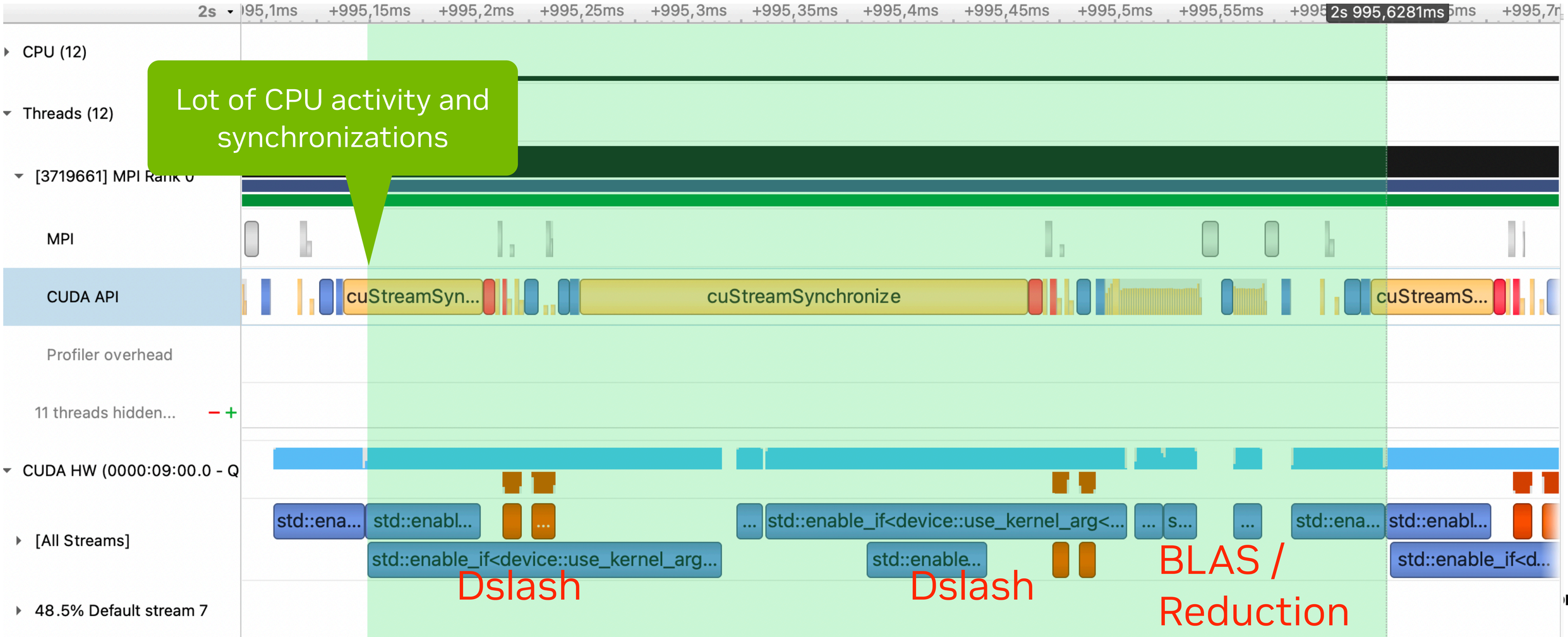
Global Volume 64^3x128, Wilson-Dslash



INVERTER SCALING

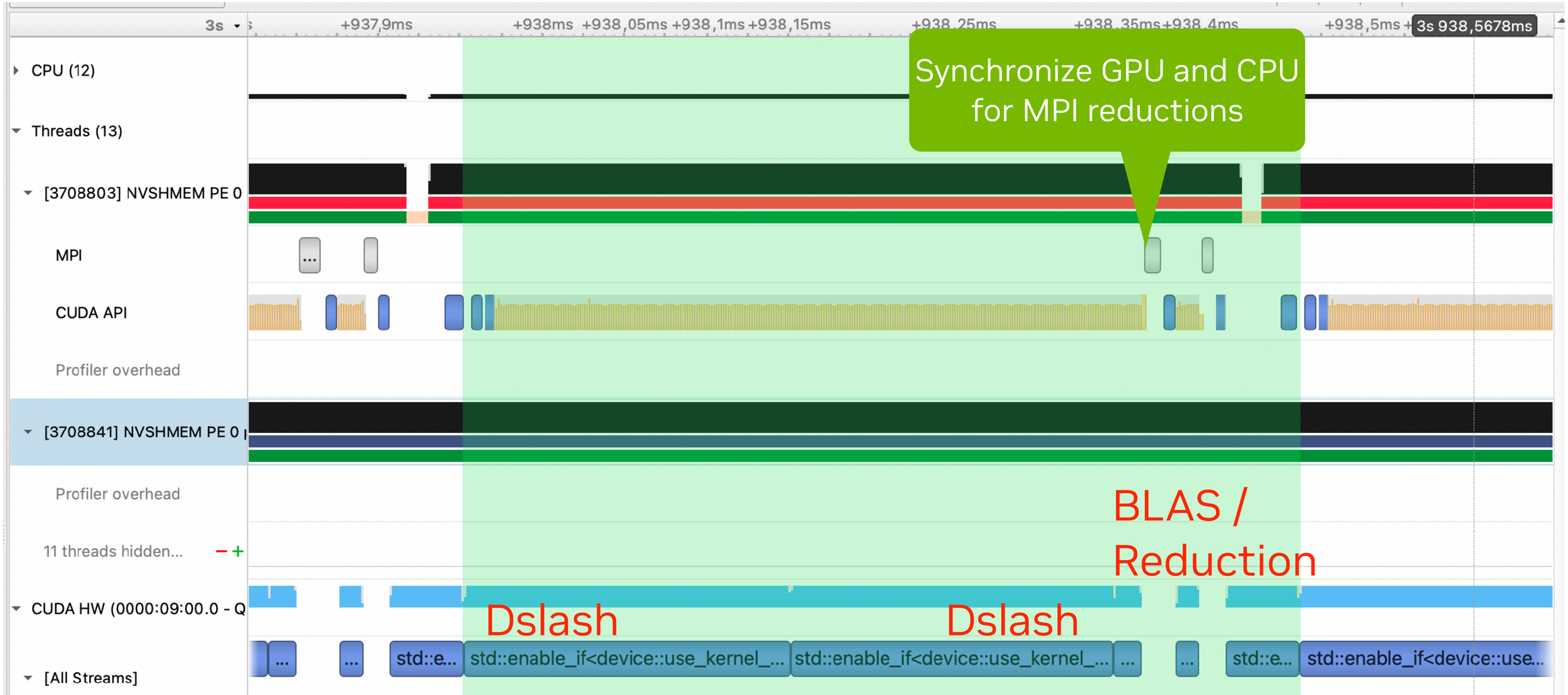
CONJUGATE GRADIENT

MPI + IPC



CONJUGATE GRADIENT

NVSHMEM



REDUCTIONS

Host-Device Synchronization

Need to synchronize the device and host when doing a reduction

Traditional QUDA method

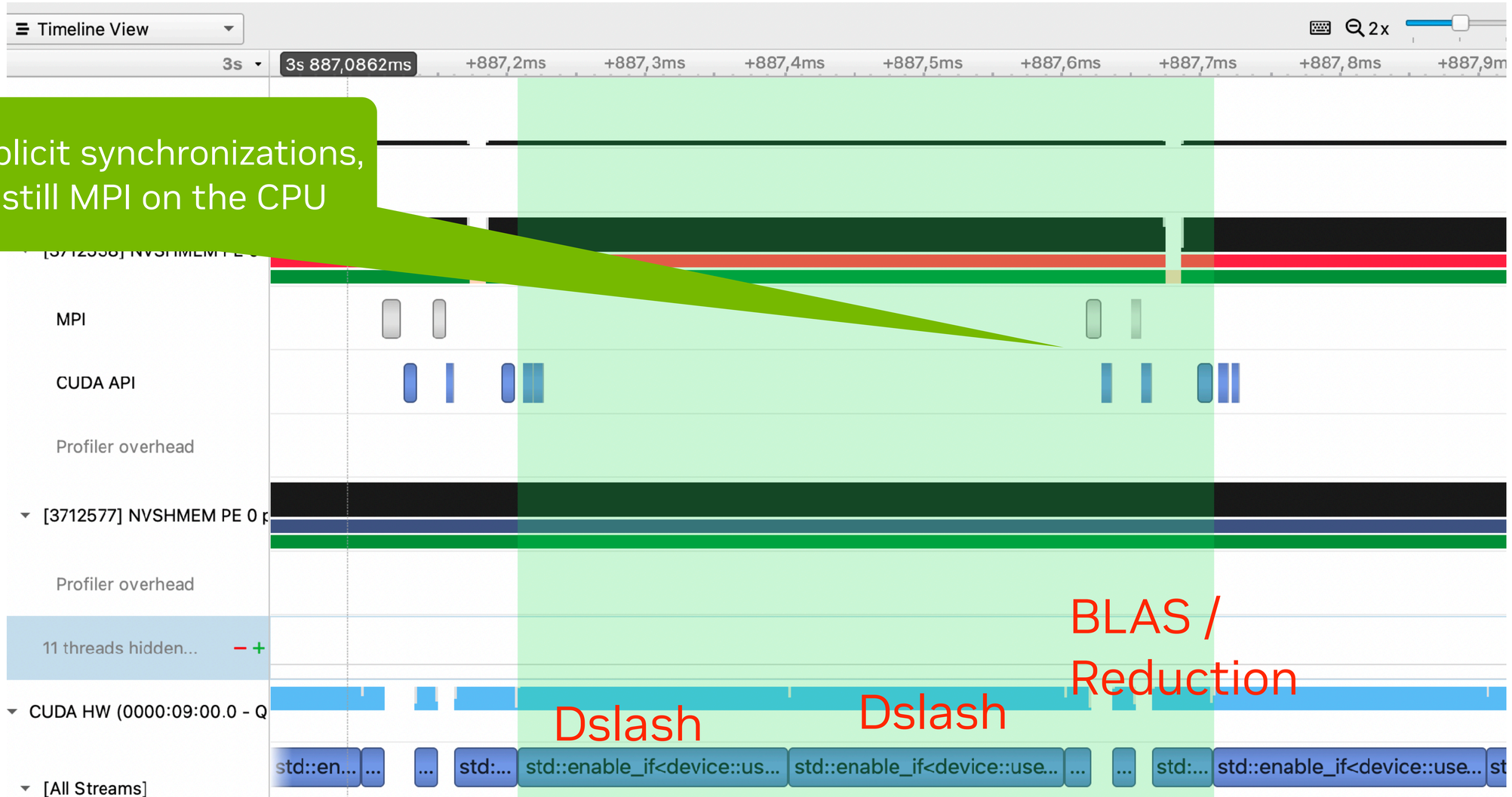
- Kernel does per-device reduction writing result to sysmem
- Synchronize host and device

Idea: use the reduced value(s) themselves as the host-device synchronization medium

- Use libcu++'s *heterogeneous atomics*
- Initialize atomic to some initial value on host
  1. Launch reduction kernel
  2. Reduced values are written as heterogeneous atomics to sysmem
  3. Host polls on heterogeneous atomic values for completion

CONJUGATE GRADIENT

NVSHMEM + Heterogenous Atomics



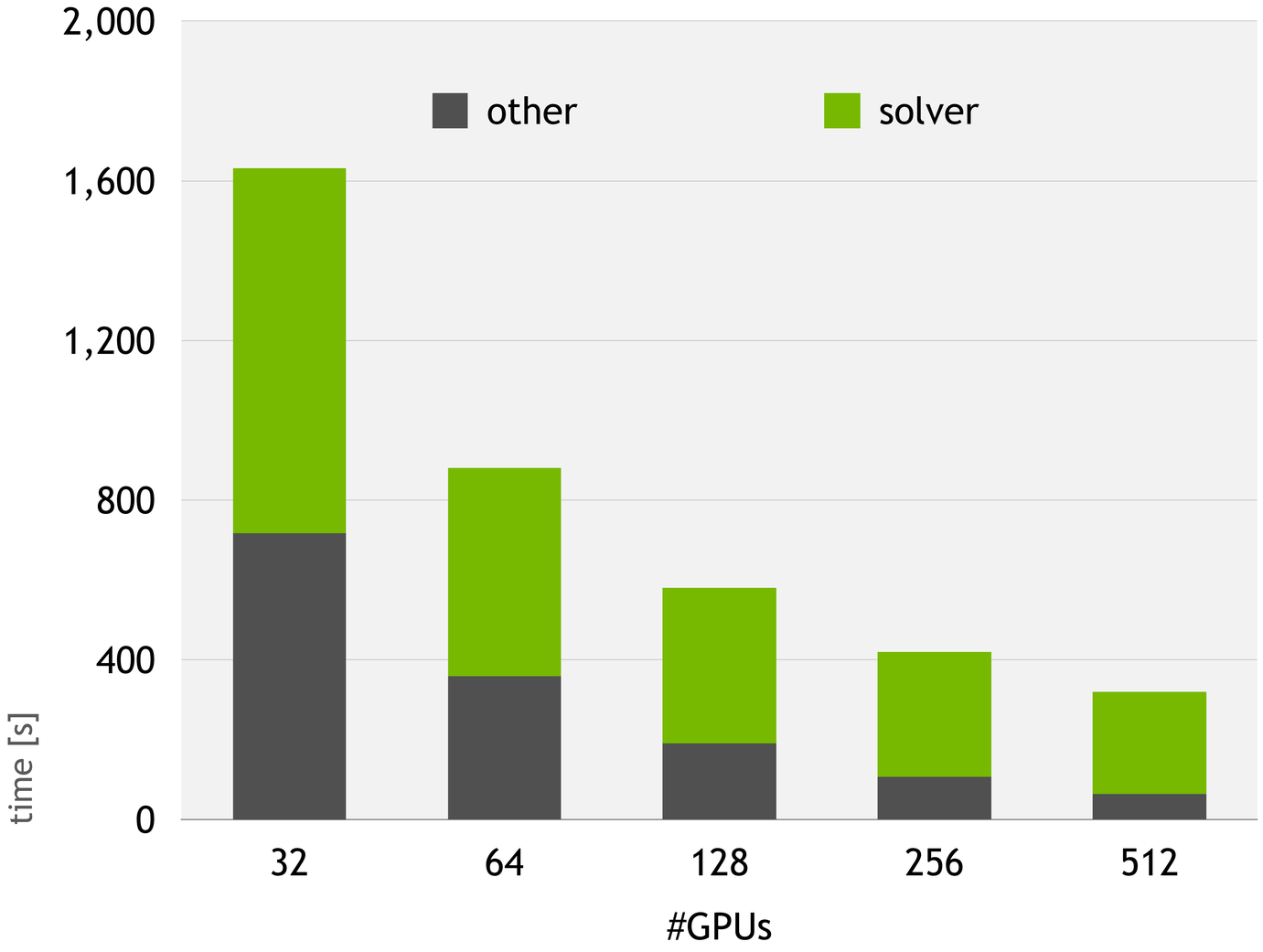
RHMC SCALING

MILC NERSC BENCHMARK OVERVIEW

- MILC NERSC Benchmark comes in 4 lattice sizes
  - small 18^3x36, medium 36^3x72, large 72^3x144, x-large 144^3x288
- Benchmark runs the RHMC algorithm
  - Dominated by the multi-shift CG sparse linear solver (stencil operator)
  - Also have auxiliary “Force” and “Link” computations
- Since 2012 MILC has built-in QUDA support
  - Enabled through a Makefile option
  - All time-critical functions off loaded to QUDA

MILC HMC SCALING ON SELENE

NERSC LARGE BENCHMARK $72^3 \times 144$



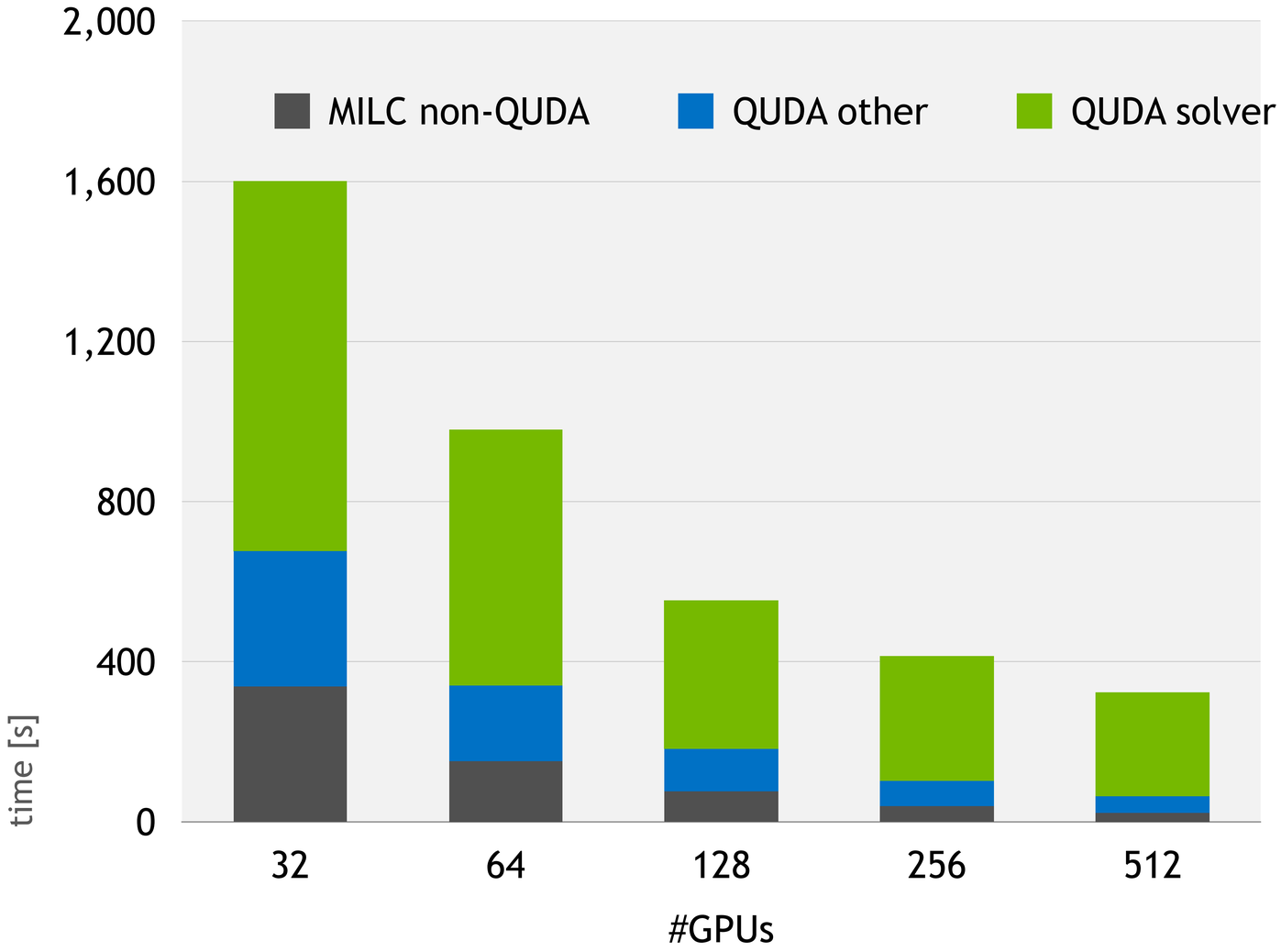
Running with MPI

other part scales reasonably
(not limited by communication)

solver part needs improvements

MILC HMC SCALING ON SELENE

NERSC LARGE BENCHMARK $72^3 \times 144$



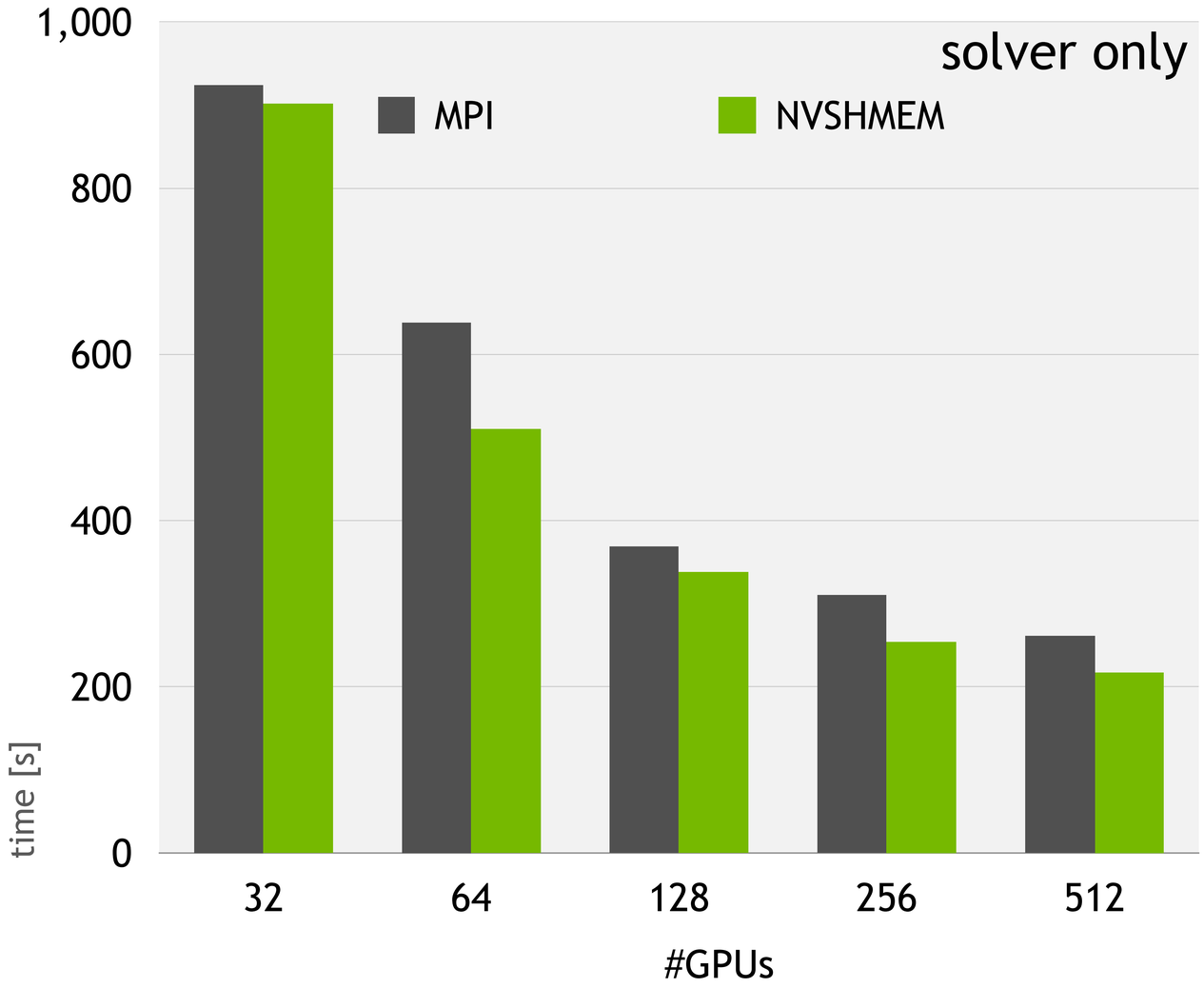
Running with MPI

other part scales reasonably
(not limited by communication)

solver part needs improvements

MILC SOLVER SCALING ON SELENE

NERSC LARGE BENCHMARK $72^3 x144$



mixed precision methods:

- lower precisions harder to scale
- NVSHMEM crucial for mixed precision

QUDA solver in MILC

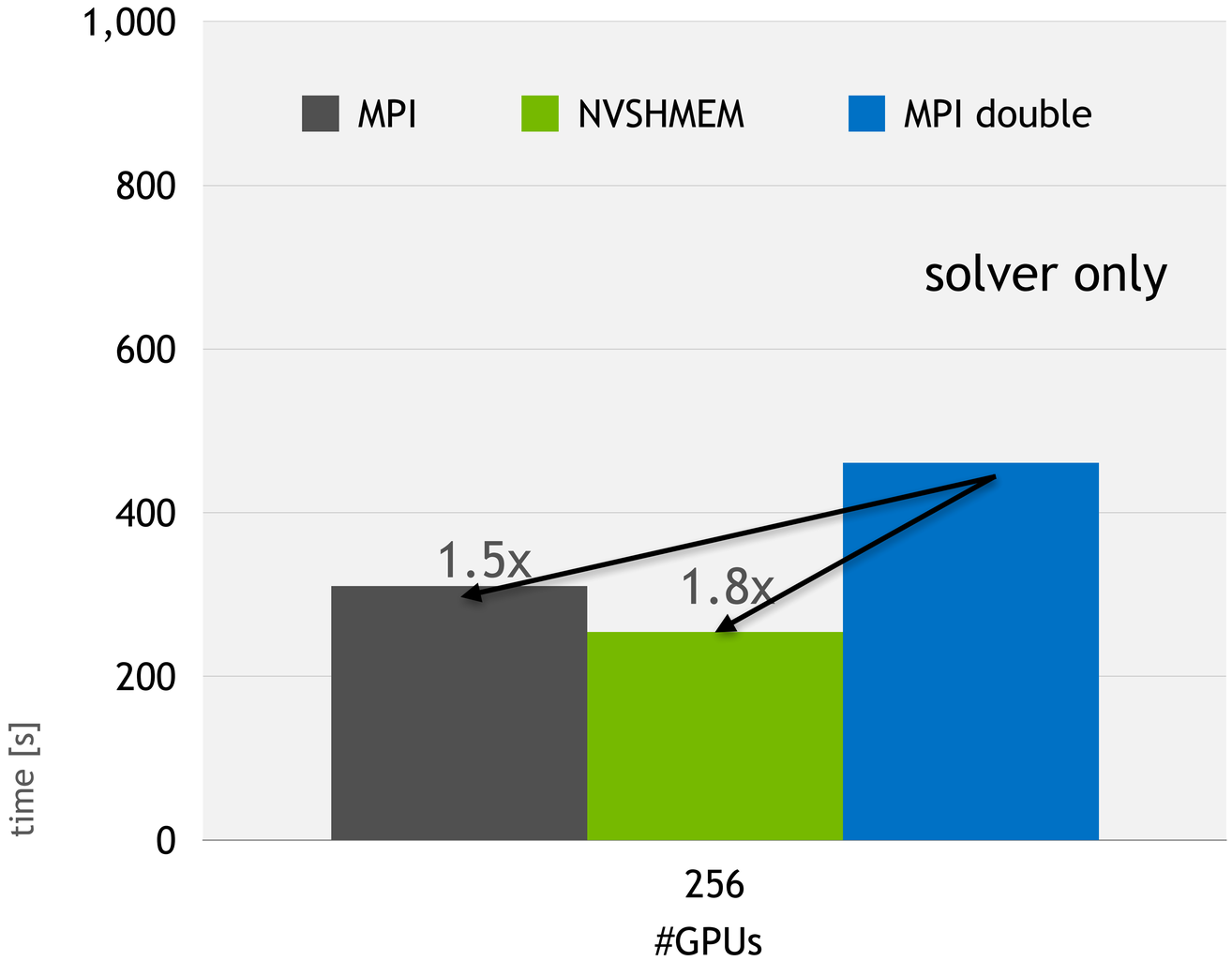
- mixed precision multishift: double-single
- mixed precision refinement: double-half

sweet spot at 256 GPUs:

- ~20% less time in solver

MILC SOLVER SCALING ON SELENE

NERSC LARGE BENCHMARK $72^3 \times 144$



mixed precision methods:

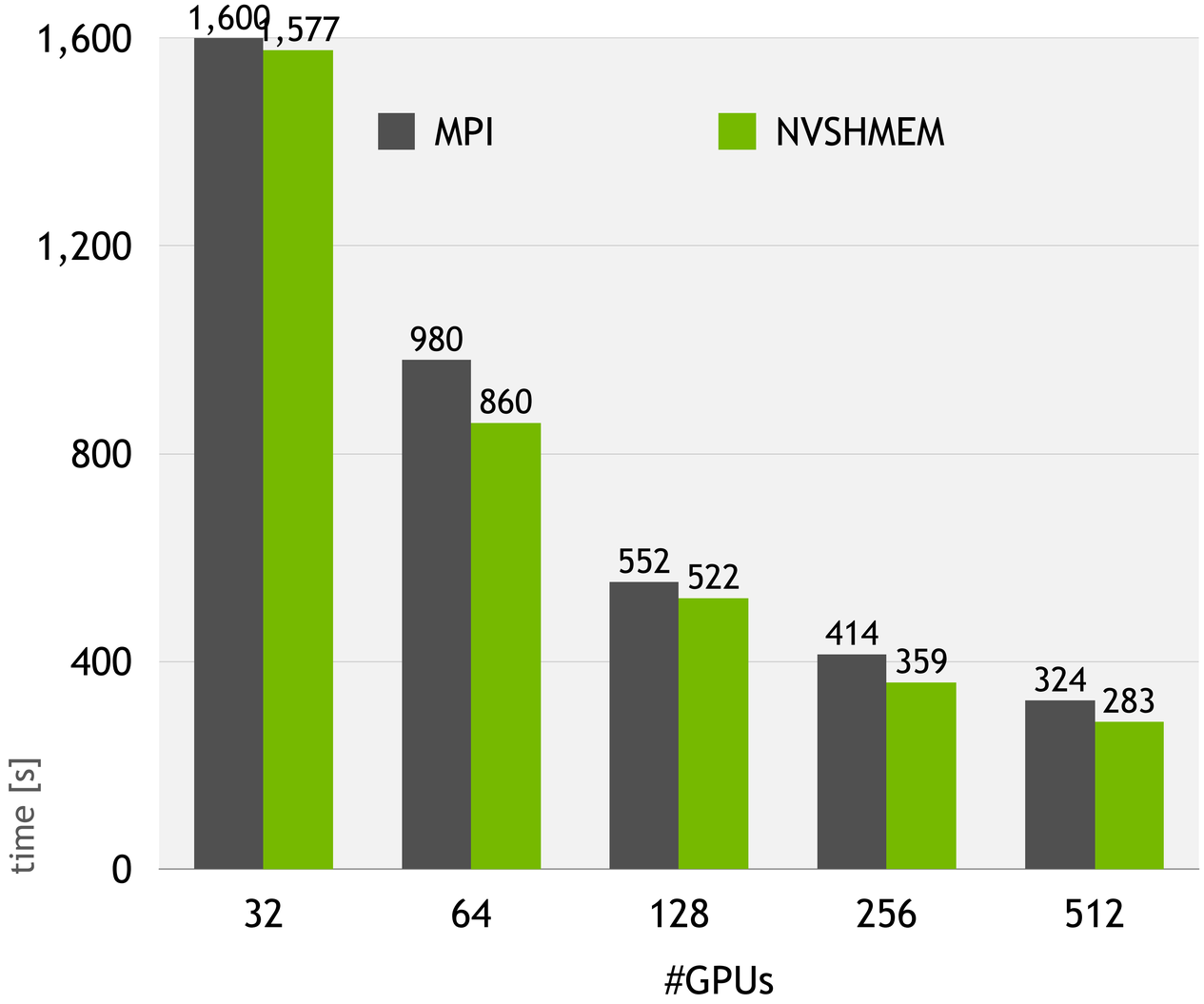
- lower precisions harder to scale
- NVSHMEM crucial for mixed precision

at 256 GPUs:

- NVSHMEM recovers expect almost 2x benefit of mixed precision

MILC SOLVER SCALING ON SELENE

NERSC LARGE BENCHMARK 72^3x144

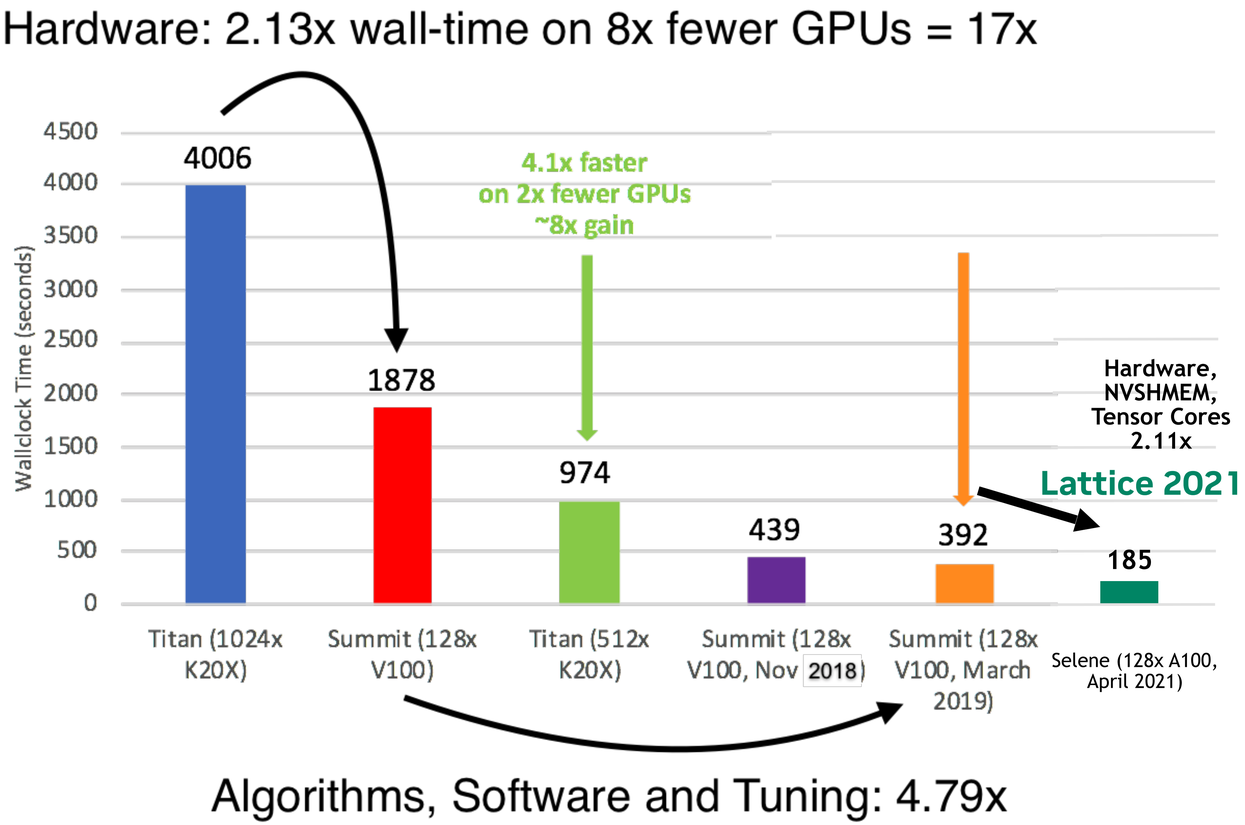


Full benchmark

>10% gains for runtime

CHROMA WILSON-CLOVER HMC

- Dominated by QUDA Multigrid
- Few solves per gauge configuration, can be bound by “heavy” (well-conditioned) solves
  - Evolve and refresh coarse space as the gauge field evolves
- Mixed precision an important piece of the puzzle
  - Double – outer solve precision
  - Single – GCR preconditioner
  - Half – Coarse operator precision
  - Int32 – deterministic parallel coarsening
- Wilson-clover MG implemented in **QUDA**, hooked into **Chroma** ; support in **Grid**
- **Latest and greatest:** Wilson-clover NVSHMEM plus tensor-core-accelerated setup

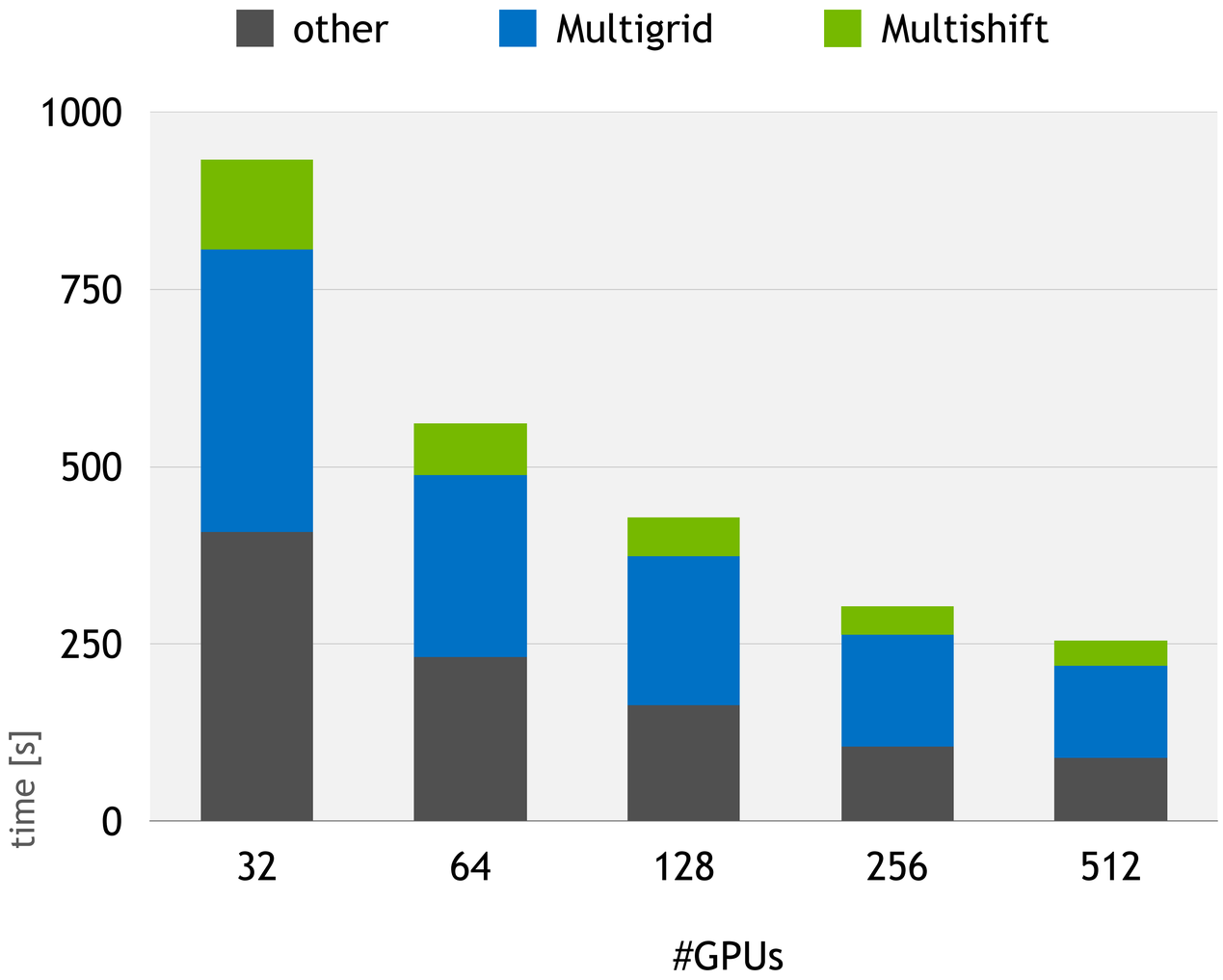


Chroma w/ QDP-JIT and QUDA, ECP FOM data, $V=64^3x128$ sites, m_π ~172 MeV, (QDP-JIT by F. Winter, Jefferson Lab)

Original figure credit Balint Joo

CHROMA WILSON-CLOVER HMC SCALING

MPI timing breakdown on Selene (Lattice 2021)



2 trajectories

other (non QUDA)

QUDA multigrid (MPI/QMP)

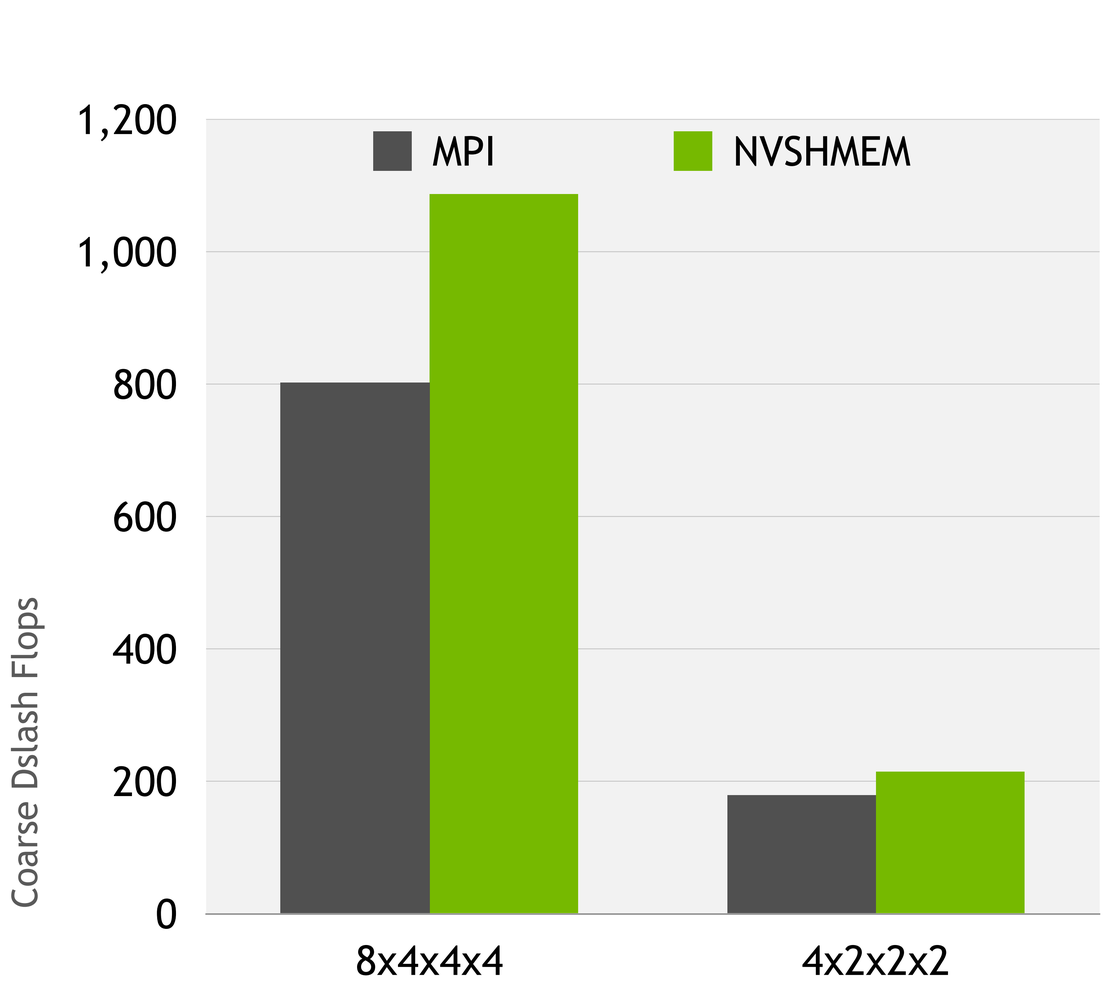
QUDA multishift (MPI/QMP)

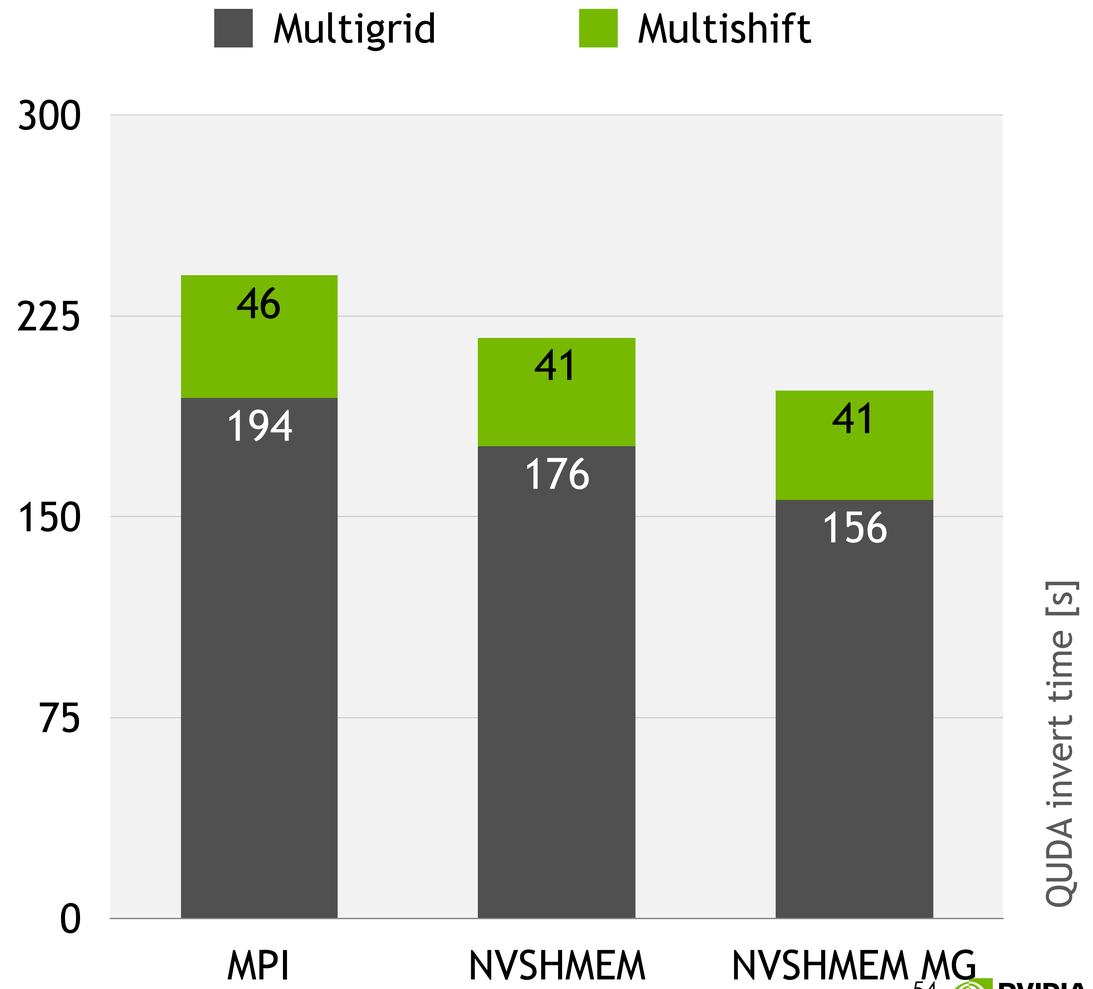
Chroma dominated by Multigrid solves

NVSHMEM FOR MULTIGRID (COARSE DSLASH)

CHROMA WILSON-CLOVER HMC SCALING

Improvements 2021 (MPI) to 2022 (NVSHMEM)

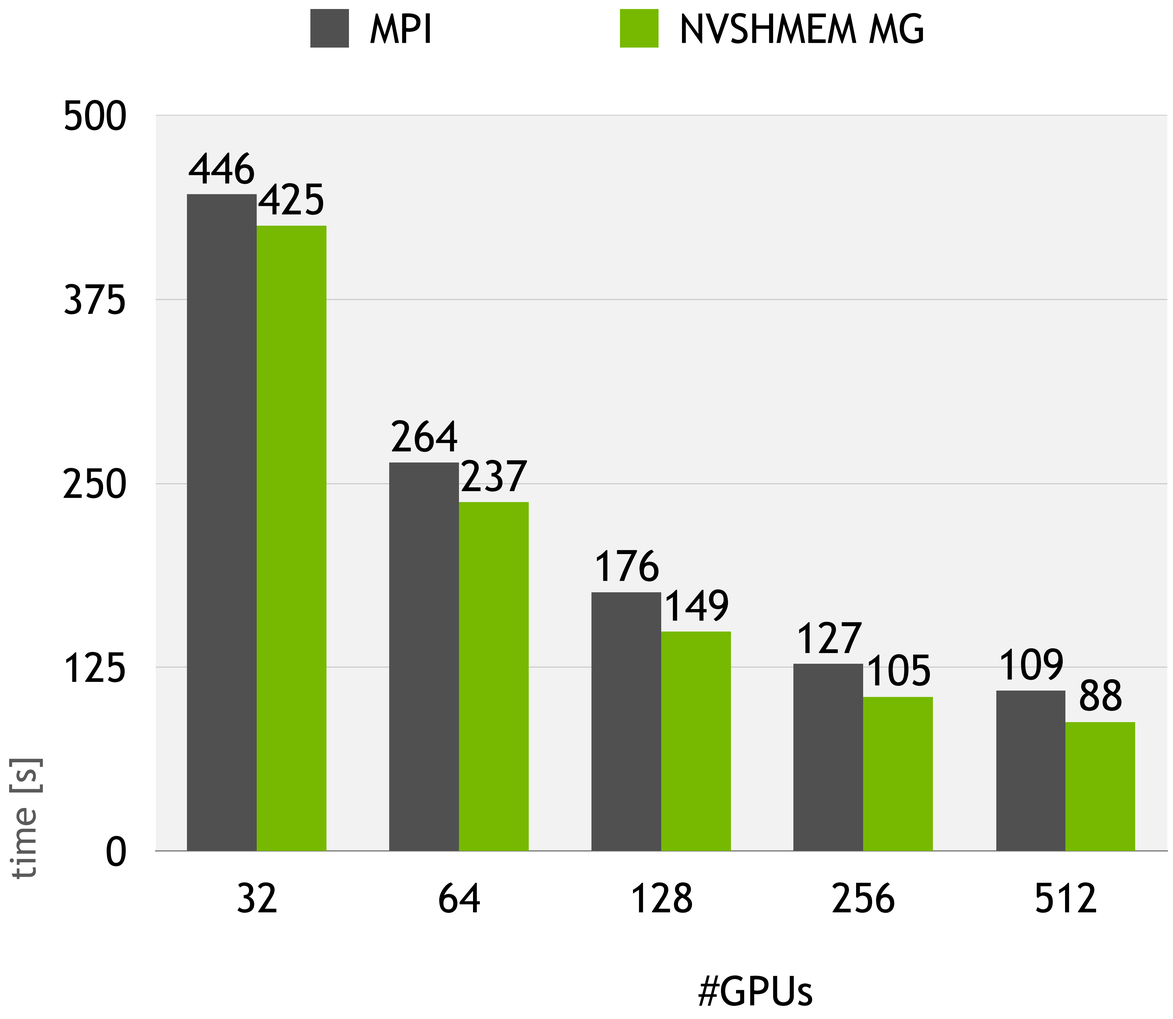




128 x NVIDIA A100 (Selene)

CHROMA WILSON-CLOVER HMC SCALING

Improvements 2021 (MPI) to 2022 (NVSHMEM)



Time for full HMC trajectory

NVSHMEM for Multigrid enabled

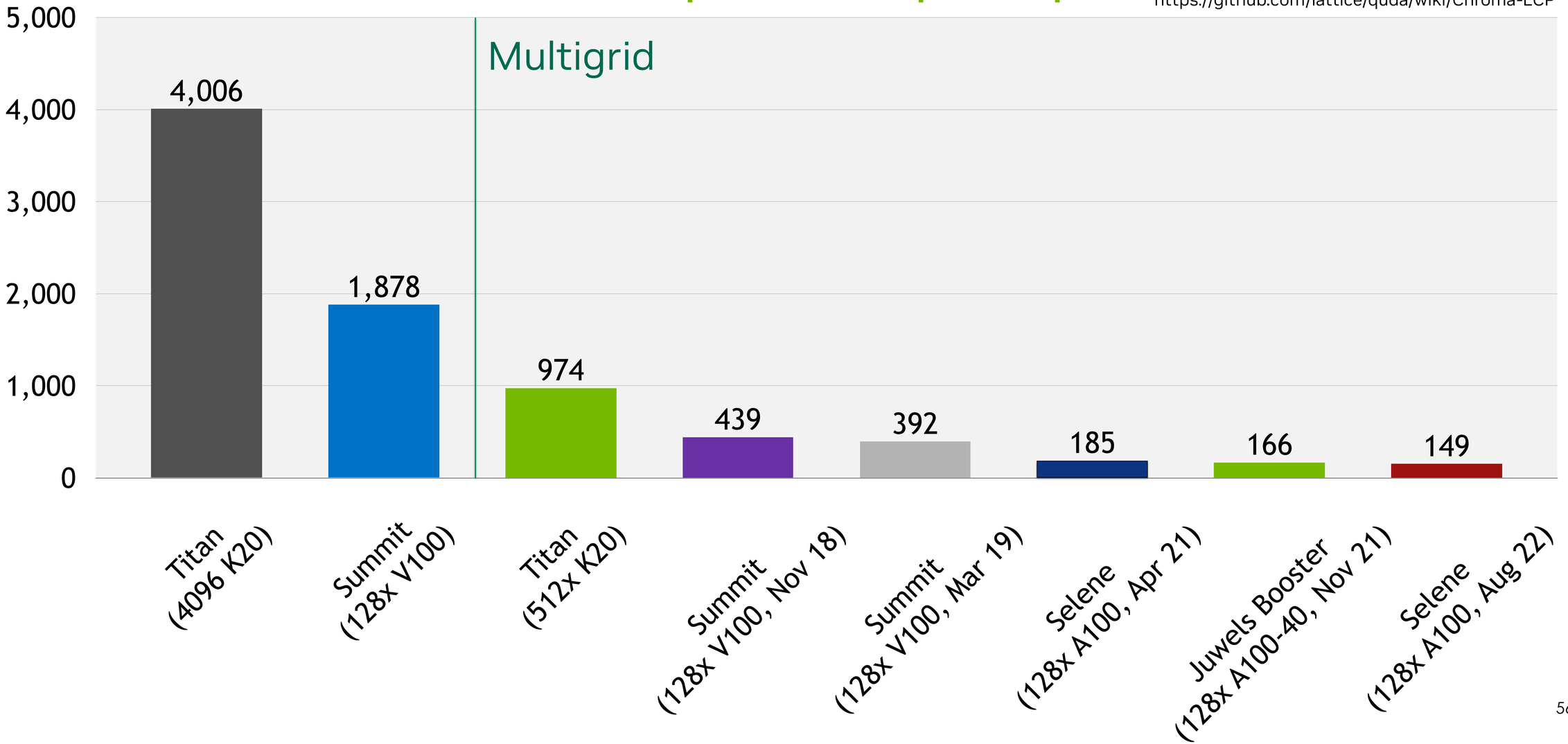
Same hardware - Same algorithm

Up to ~20% speedup

CHROMA ECP BENCHMARK

Multiplicative Speedup

https://github.com/lattice/quda/wiki/Chroma-ECP



MORE QCD FROM YOUR GPU

Multiplicative speedup from hardware and software

QUDA and GPUs are here to stay: 10+ years on NVIDIA GPUs

Mixed and custom precisions to optimally match needs

Algorithmic improvements

NVSHMEM delivers RHMC scaling benefits

  up to 20% reduction in time to solution

Watch out for more to come …

… and join the flywheel