S9708 – STRONG SCALING HPC APPLICATIONS: BEST PRACTICES WITH A LATTICE QCD CASE STUDY

Kate Clark, Mathias Wagner

AGENDA

Lattice Quantum Chromodynamics

QUDA

Bandwidth Optimization

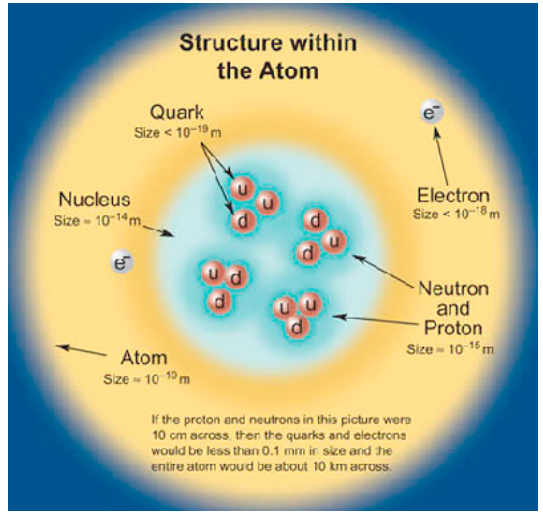
Latency Optimization

Multi-node scaling

NVSHMEM

Summary

QUANTUM CHROMODYNAMICS



The strong force is one of the basic forces of nature
(along with gravity, em and weak)

It's what binds together the quarks and gluons in the proton and the neutron
(as well as hundreds of other particles seen in accelerator experiments)

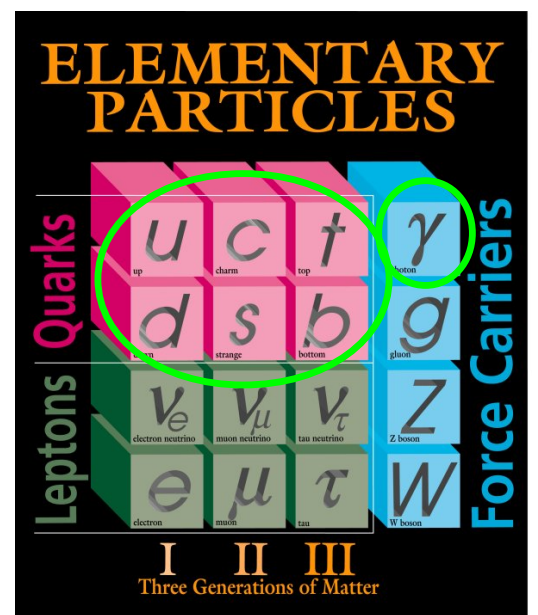
Responsible for the particle zoo seen at sub-nuclear scales (mass, decay rate, etc.)

QCD is the theory of the strong force

It's a beautiful theory…

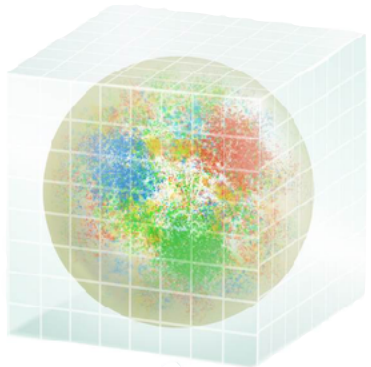
$$\langle \Omega \rangle = \frac{1}{Z} \int [dU] e^{-\int d^4x L(U)} \Omega(U)$$

…but



Fermi National Accelerator Laboratory

LATTICE QUANTUM CHROMODYNAMICS



Theory is highly non-linear ⇒ cannot solve directly

Must resort to numerical methods to make predictions

Lattice QCD

- Discretize spacetime ⇒ 4-d dimensional lattice of size Lx x Ly x Lz x Lt
- Finite spacetime ⇒ periodic boundary conditions
- PDEs ⇒ finite difference equations

Consumer of 10-20% of public supercomputer cycles

- Traditionally highly optimized on every HPC platform for the past 30 years

Andre Walker-Loud S91010: Accelerating our Understanding of Nuclear Physics and the Early Universe

Jiqun Tu S9330: Lattice QCD with Tensor Cores

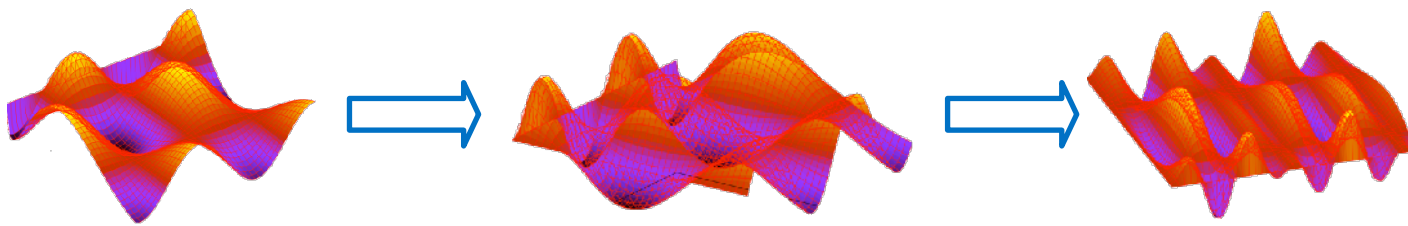
STEPS IN AN LQCD CALCULATION

$$D_{ij}^{\alpha\beta}(x,y;U)\psi_j^{\beta}(y) = \eta_i^{\alpha}(x)$$

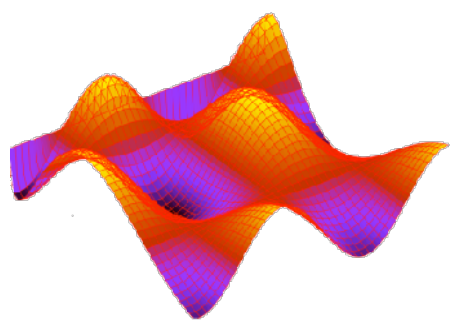
or Ax = b

1. Generate an ensemble of gluon field configurations “gauge generation”
   - Produced in sequence, with hundreds needed per ensemble
   - Strong scaling required with 100-1000 TFLOPS sustained for several months
   - 50-90% of the runtime is in the linear solver
   - O(1) solve per linear system
   - Target 16^4 per GPU

Simulation Cost ~ a^{-6} $V^{5/4}$

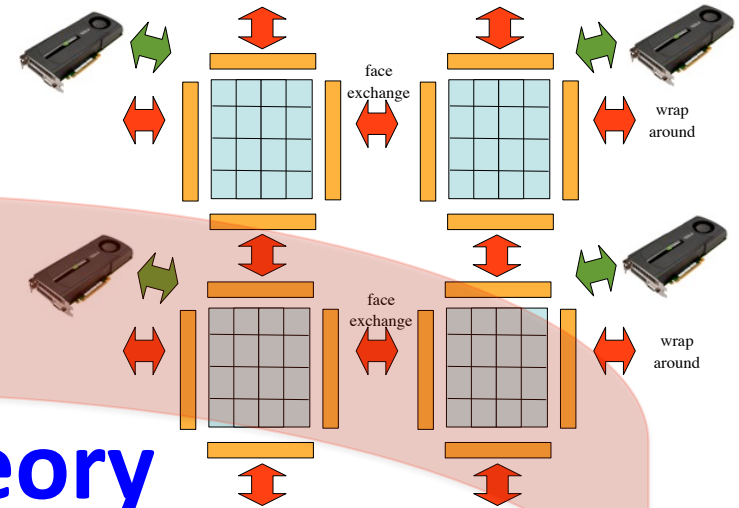


2. “Analyze” the configurations
   - Can be farmed out, assuming ~10 TFLOPS per job
   - Task parallelism means that clusters reign supreme here
   - 80-99% of the runtime is in the linear solver
     Many solves per system, e.g., O(10^6)
   - Target 24^4-32^4 per GPU

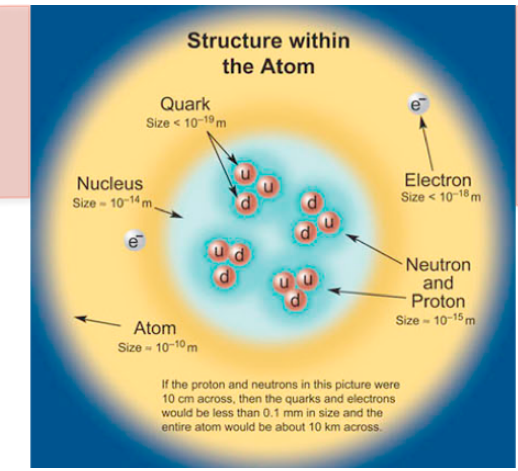


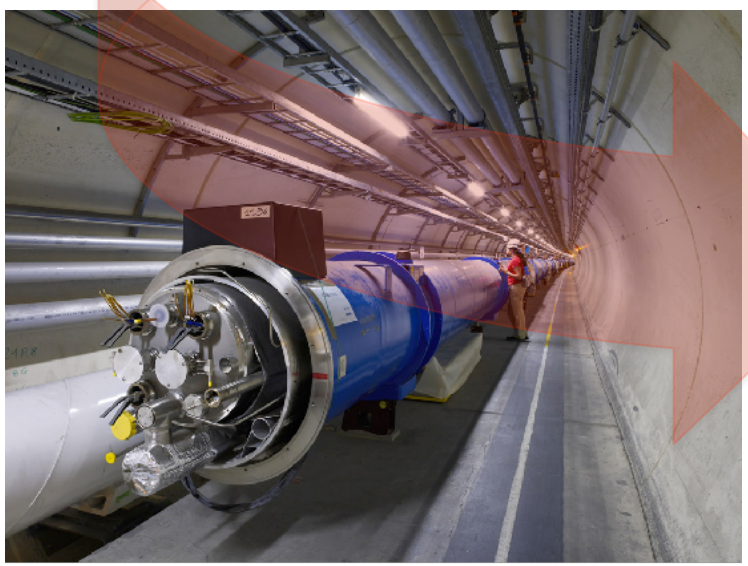
LATTICE QCD IN A NUTSHELL

$$\langle \Omega \rangle = \frac{1}{Z} \int [dU] e^{-\int d^4x L(U)} \Omega(U)$$

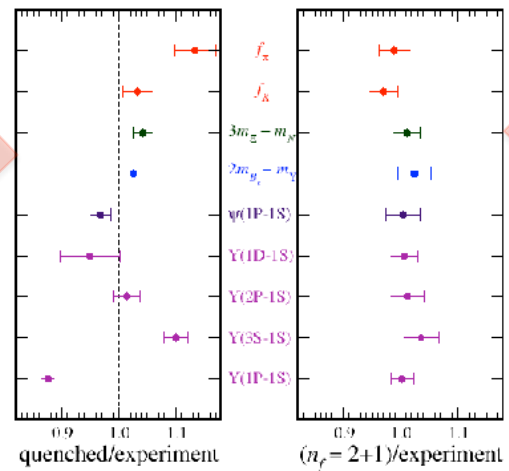


experiment **theory**

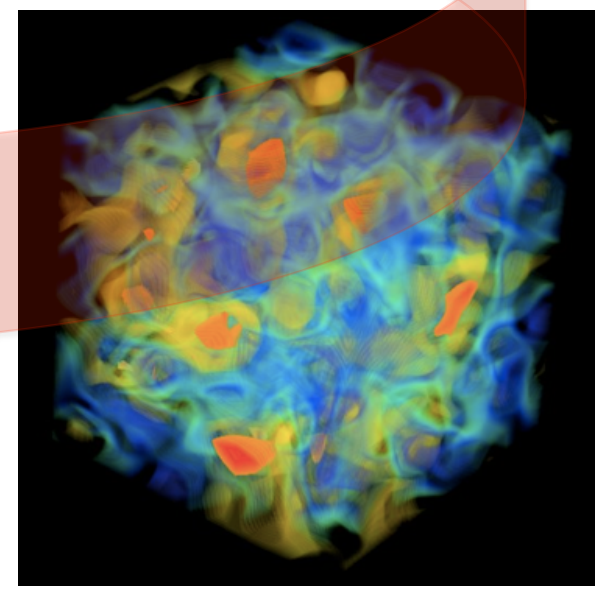




Large Hadron Collider



Davies *et al*



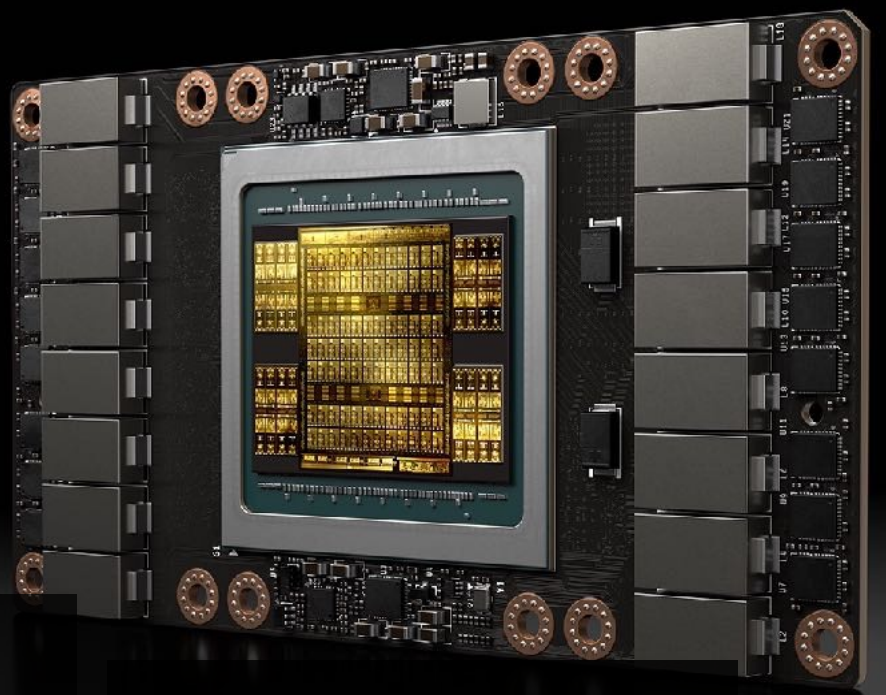
Brookhaven National Laboratory

NVIDIA POWERS WORLD'S FASTEST SUPERCOMPUTER

Summit Becomes First System To Scale The 100 Petaflops Milestone

122 PF HPC

3 EF AI



27,648 Volta Tensor Core GPUs

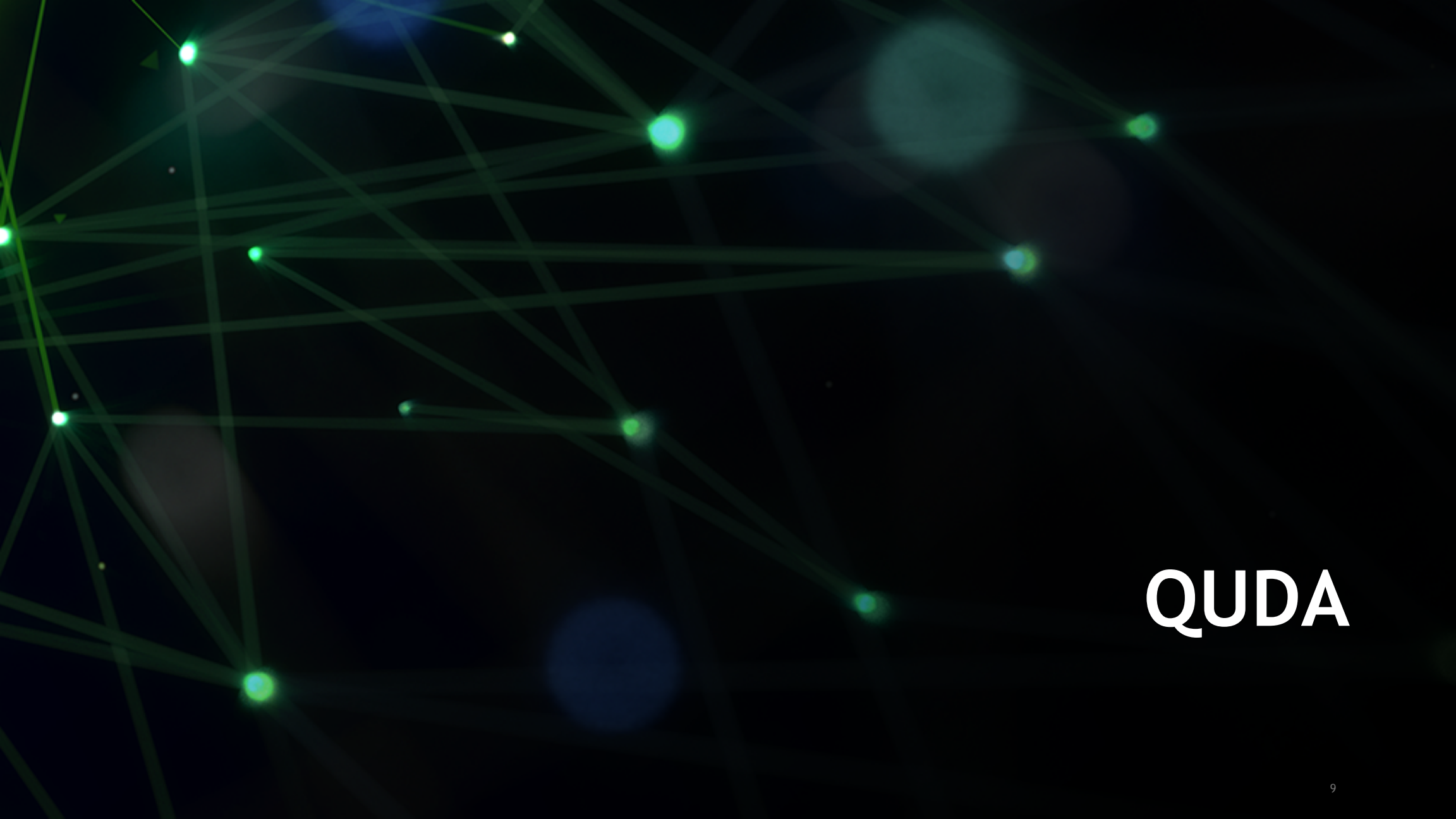
STRONG SCALING

Multiple meanings

- Same problem size, more nodes, more GPUs
- Same problem, next generation GPUs
- Multigrid - strong scaling within the same run (not discussed here)

To tame strong scaling we have to understand the limiters

- Bandwidth limiters
- Latency limiters


QUDA

QUDA

- "QCD on CUDA" – http://lattice.github.com/quda (open source, BSD license)
- 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 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 on all CUDA-capable architectures
  - Domain-decomposed (Schwarz) preconditioners for strong scaling
  - Eigenvector and deflated solvers (Lanczos, EigCG, GMRES-DR)
  - Multi-source solvers
  - Multigrid solvers for optimal convergence
- A research tool for how to reach the exascale

QUDA CONTRIBUTORS

10 years - lots of contributors

Ron Babich (NVIDIA)
Simone Bacchio (Cyprus)
Michael Baldhauf (Regensburg)
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 (William and Mary)
Steve Gottlieb (Indiana University)
Kyriakos Hadjiyiannakou (Cyprus)
Dean Howarth (BU)
Bálint Joó (Jlab)
Hyung-Jin Kim (BNL -> Samsung)
Bartek Kostrzewa (Bonn)
Claudio Rebbi (Boston University)
Hauke Sandmeyer (Bielefeld)
Guochun Shi (NCSA -> Google)
Mario Schröck (INFN)
Alexei Strelchenko (FNAL)
Jiqun Tu (Columbia)
Alejandro Vaquero (Utah University)
Mathias Wagner (NVIDIA)*
Evan Weinberg (NVIDIA)*
Frank Winter (Jlab)

*this work

LINEAR SOLVERS

$$
\begin{aligned}
&\text{while } (|\mathbf{r}_k| > \varepsilon)\ \{\\
&\quad \beta_k = (\mathbf{r}_k, \mathbf{r}_k)/(\mathbf{r}_{k-1}, \mathbf{r}_{k-1})\\
&\quad \mathbf{p}_{k+1} = \mathbf{r}_k - \beta_k \mathbf{p}_k\\
&\quad \mathbf{q}_{k+1} = A\,\mathbf{p}_{k+1}\\
&\quad \alpha = (\mathbf{r}_k, \mathbf{r}_k)/(\mathbf{p}_{k+1}, \mathbf{q}_{k+1})\\
&\quad \mathbf{r}_{k+1} = \mathbf{r}_k - \alpha \mathbf{q}_{k+1}\\
&\quad \mathbf{x}_{k+1} = \mathbf{x}_k + \alpha \mathbf{p}_{k+1}\\
&\quad k = k+1\\
&\}
\end{aligned}
$$

conjugate gradient

QUDA supports a wide range of linear solvers

- CG, BiCGstab, GCR, Multi-shift solvers, etc.

Condition number inversely proportional to mass

- Light (realistic) masses are highly singular
- Naive Krylov solvers suffer from critical slowing down at decreasing mass

Entire solver algorithm must run on GPUs

- Time-critical kernel is the stencil application
- Also require BLAS level-1 type operations

MAPPING THE DIRAC OPERATOR TO CUDA

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 => 3.3×10^6 threads

Looping over direction each thread must

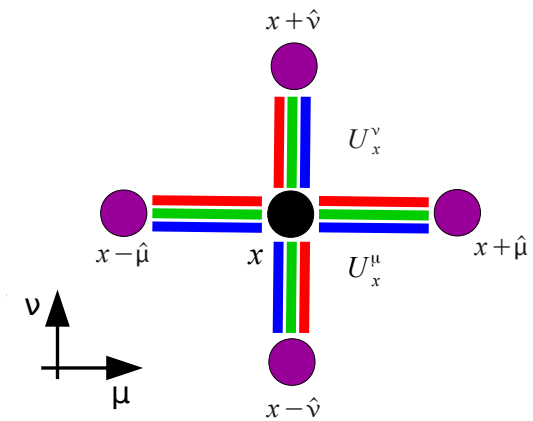
- Load the neighboring spinor (24 numbers x8)
- Load the color matrix connecting the sites (18 numbers x8)
- Do the computation
- Save the result (24 numbers)

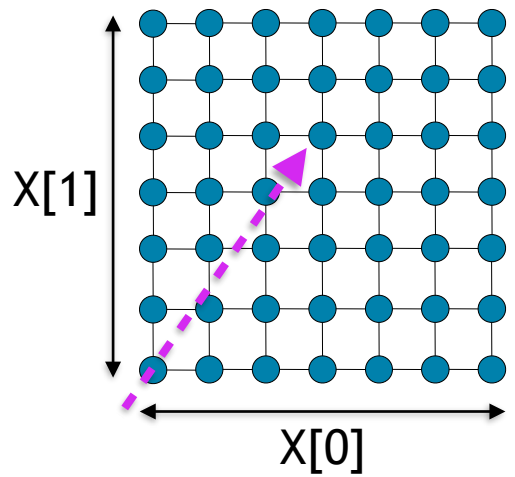
Each thread has (Wilson Dslash) 0.92 naive arithmetic intensity

QUDA reduces memory traffic

- Exact SU(3) matrix compression (18 => 12 or 8 real numbers)
- Use 16-bit fixed-point representation with mixed-precision solver

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





SINGLE GPU PERFORMANCE

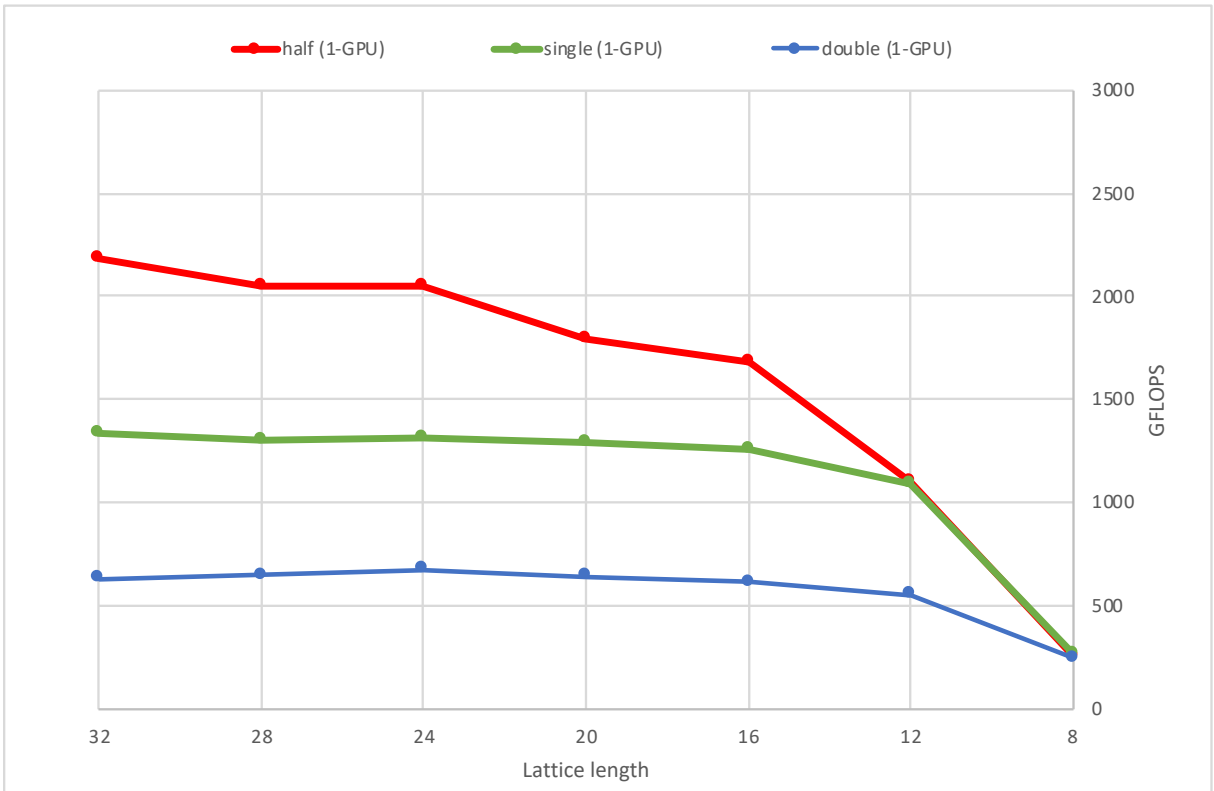
“Wilson Dslash” stencil

1013 GB/s

1119 GB/s

1115 GB/s

cf. STREAM 850 GB/s



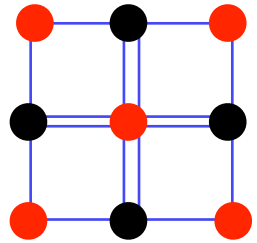
Tesla V100
CUDA 10.1
GCC 7.3

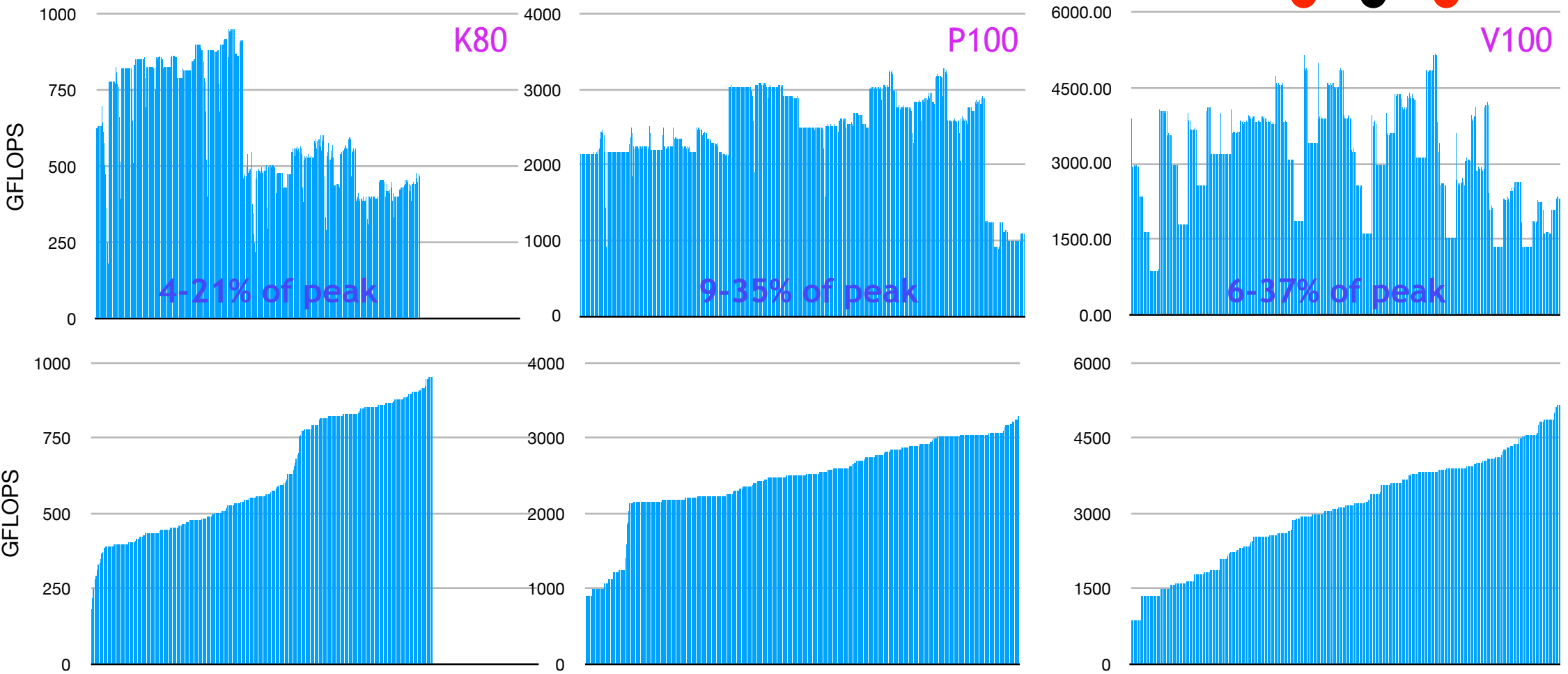
“strong scaling” ⟶

BANDWIDTH OPTIMIZATION

GENERATIONAL COMPARISON

$F_{\mu v}$ kernel - batched 3x3 multiplication





QUDA'S AUTOTUNER

QUDA includes an autotuner for ensuring optimal kernel performance

- virtual C++ class "`Tunable`" that is derived for each kernel you want to autotune

By default `Tunable` classes will autotune 1-d CTA size, shared memory size, grid size

- Derived specializations do 2-d and 3-d CTA tuning

Tuned parameters are stored in a `std::map` and dumped to disk for later reuse

Built in performance metrics and profiling

User just needs to

- State resource requirements: shared memory per thread and/or per CTA, total number of threads
- Specify a tuneKey which gives each kernel a unique entry and break any degeneracy

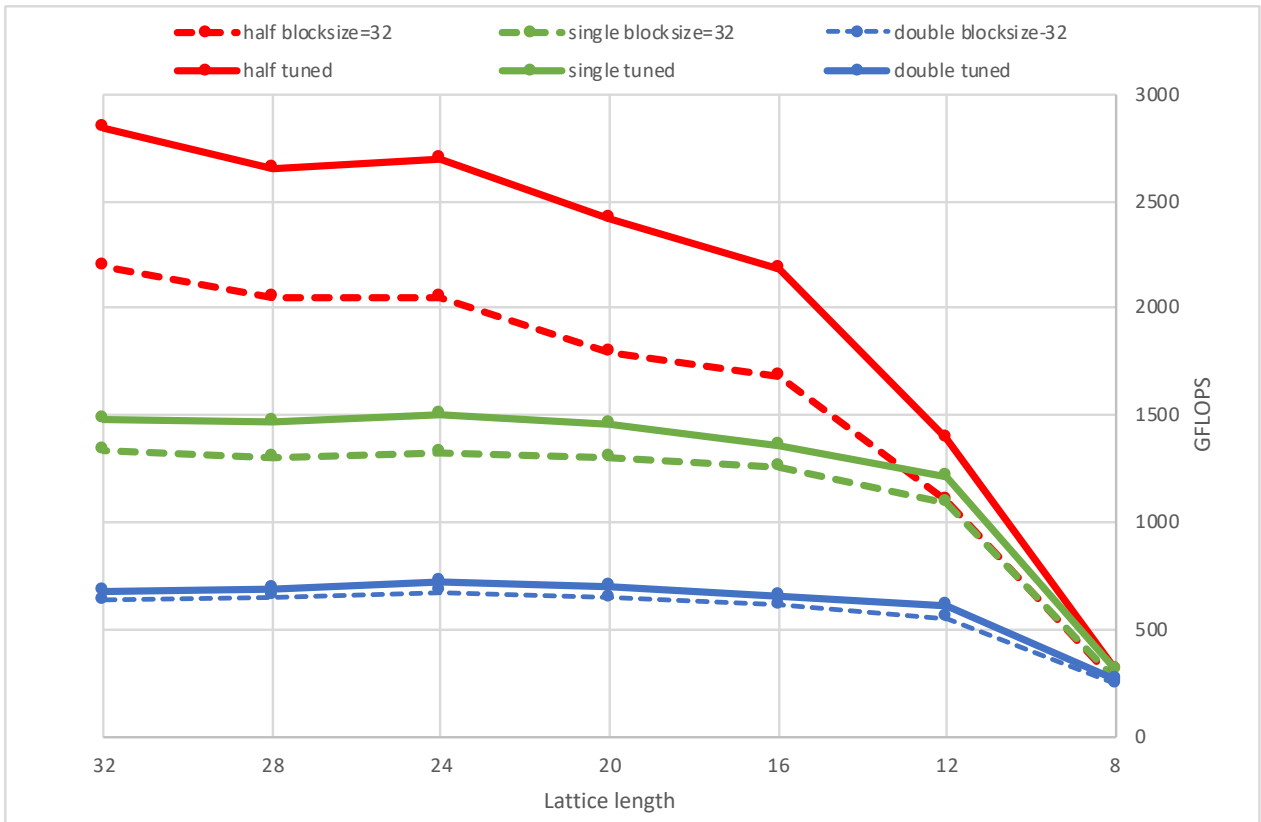
SINGLE GPU PERFORMANCE

“Wilson Dslash” stencil

1312 GB/s

1291 GB/s

1180 GB/s



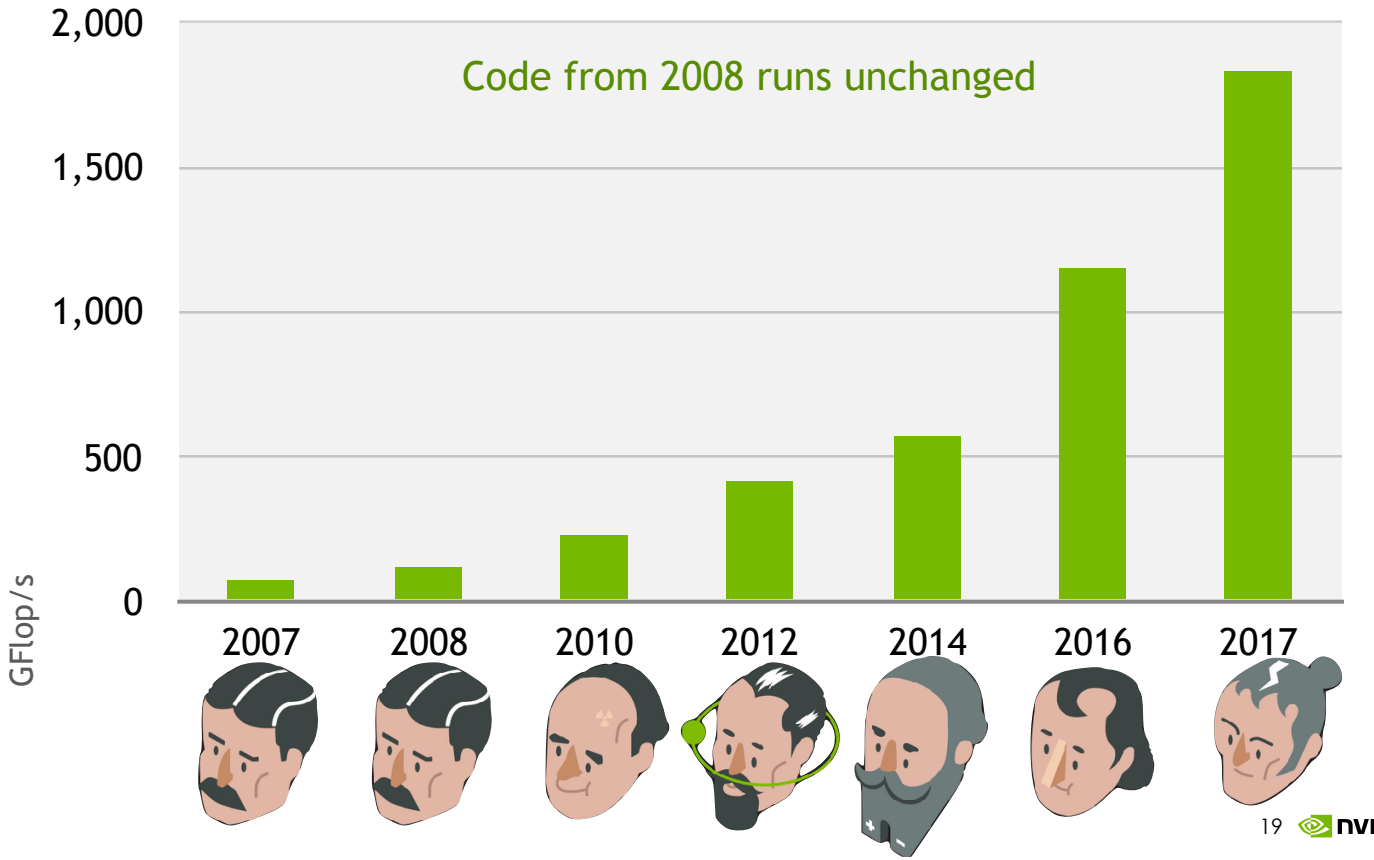
cf Perfect L2 roofline
~ 1700 GB/s

“strong scaling” ⟶

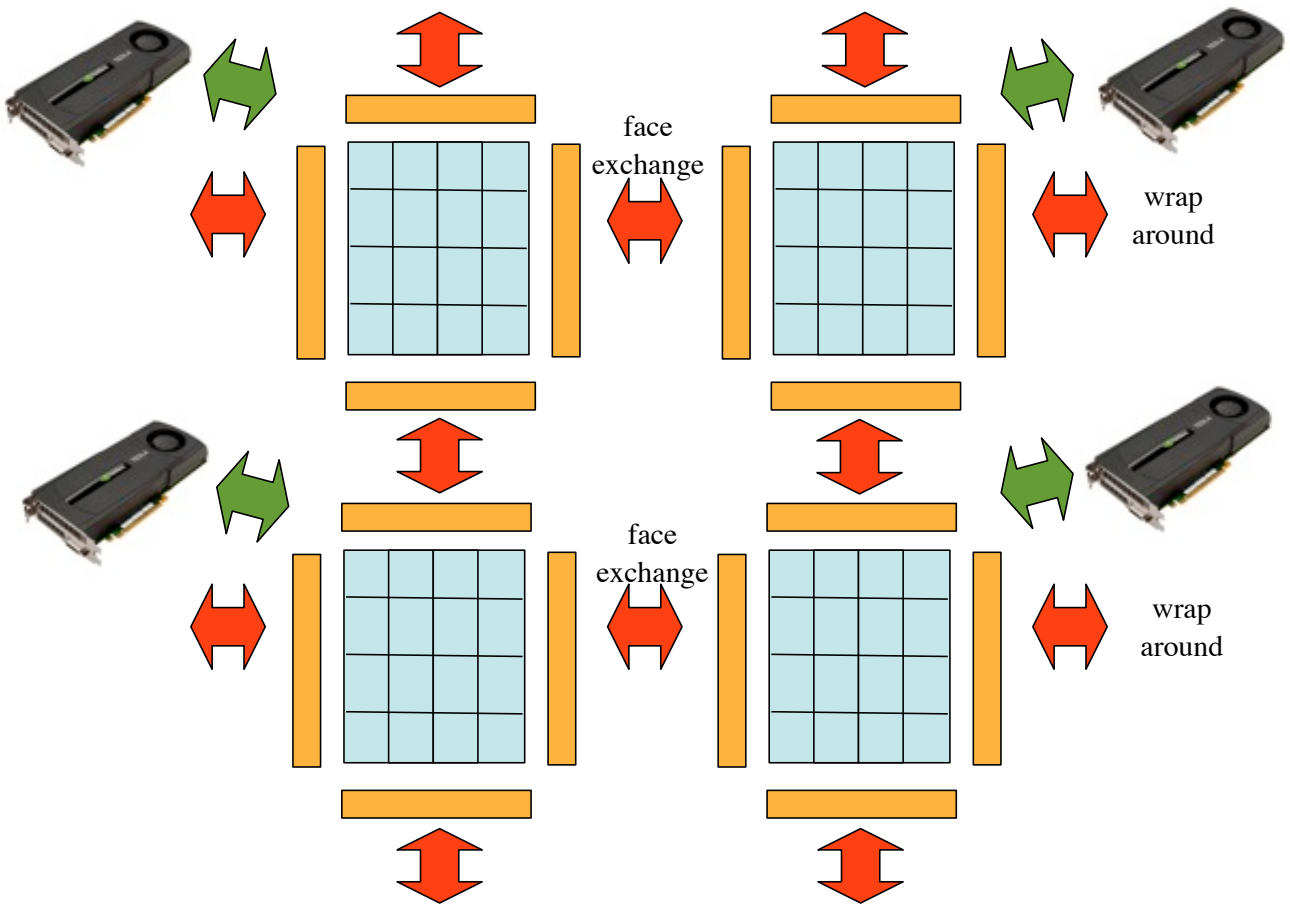
Tesla V100
CUDA 10.1
GCC 7.3

RECOMPILE AND RUN

Autotuning provides performance portability



MULTI GPU BUILDING BLOCKS



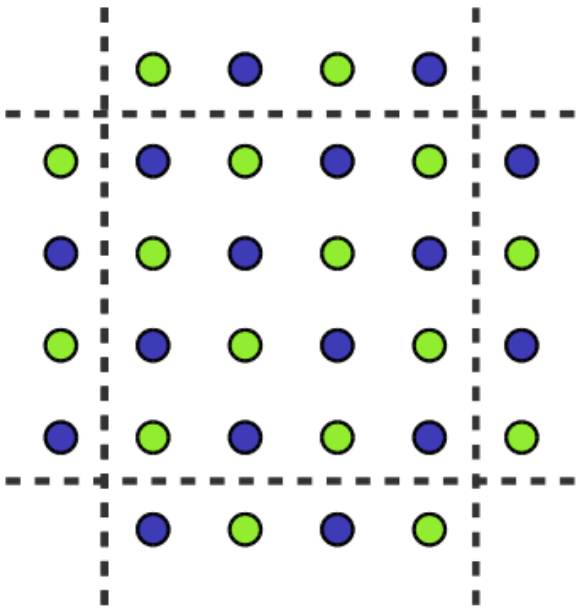
Halo packing Kernel

Interior Kernel

Halo communication

Halo update Kernel

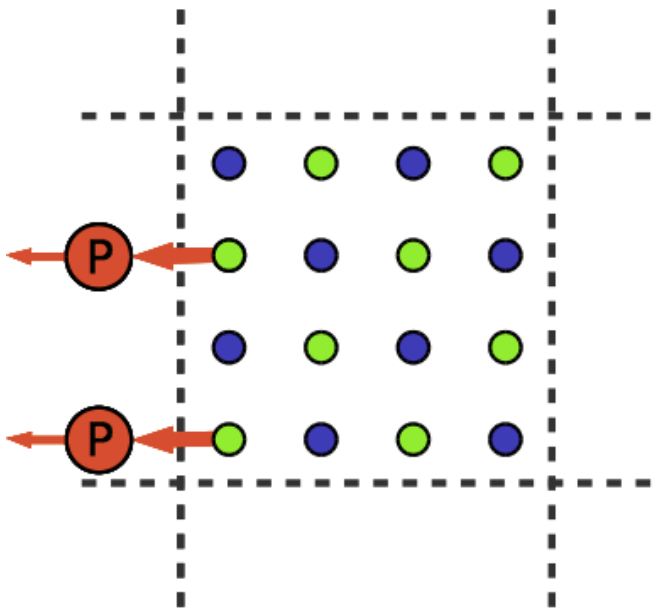
Multi-dimensional Kernel Computation



2-d example

- Checkerboard updating scheme employed, so only half of the sites are updated per application
- Green: source sites
- Purple: sites to be updated
- Orange: site update complete

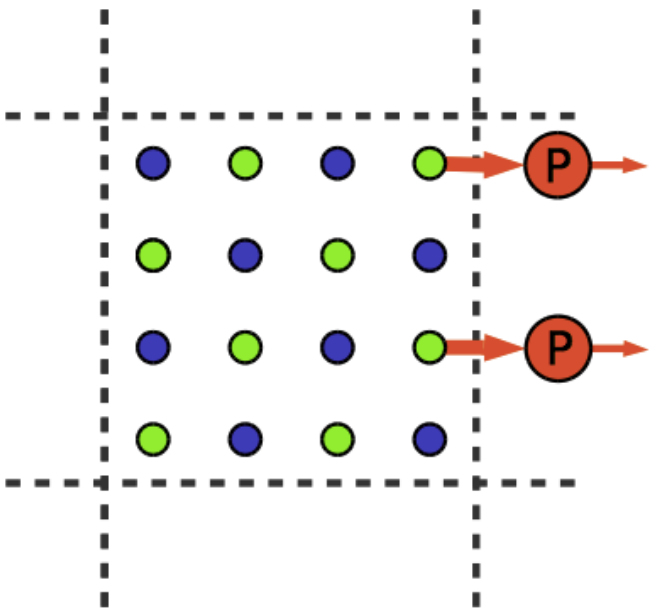
Multi-dimensional Kernel Computation



Step 1

- Gather boundary sites into contiguous buffers to be shipped off to neighboring GPUs, one direction at a time.

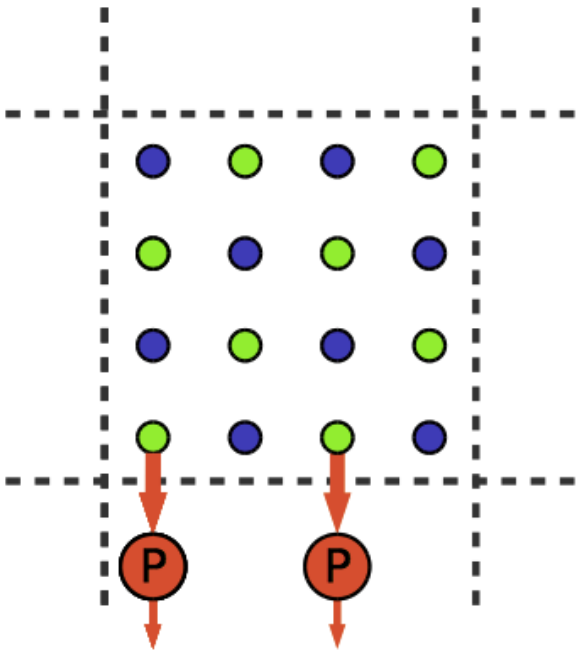
Multi-dimensional Kernel Computation



Step 1

- Gather boundary sites into contiguous buffers to be shipped off to neighboring GPUs, one direction at a time.

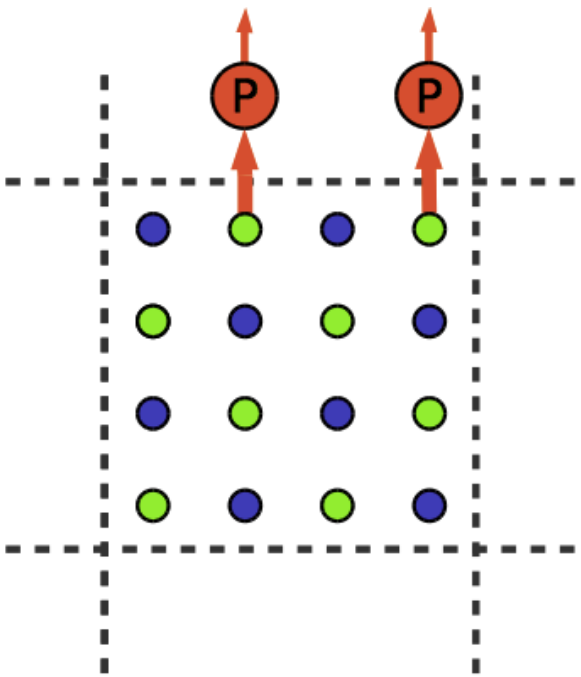
Multi-dimensional Kernel Computation



Step 1

- Gather boundary sites into contiguous buffers to be shipped off to neighboring GPUs, one direction at a time.

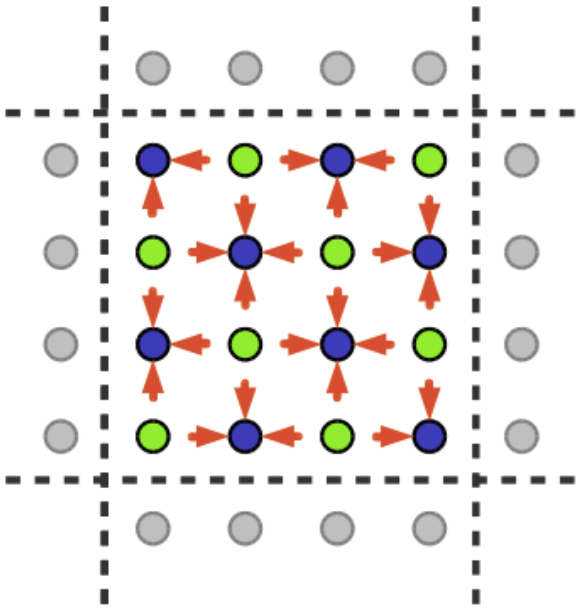
Multi-dimensional Kernel Computation



Step 1

- Gather boundary sites into contiguous buffers to be shipped off to neighboring GPUs, one direction at a time.

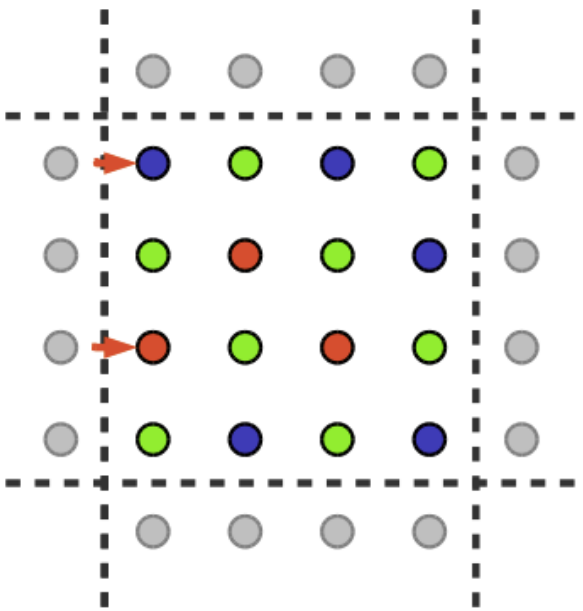
Multi-dimensional Kernel Computation



Step 2

- An “interior kernel” updates all local sites to the extent possible. Sites along the boundary receive contributions from local neighbors.

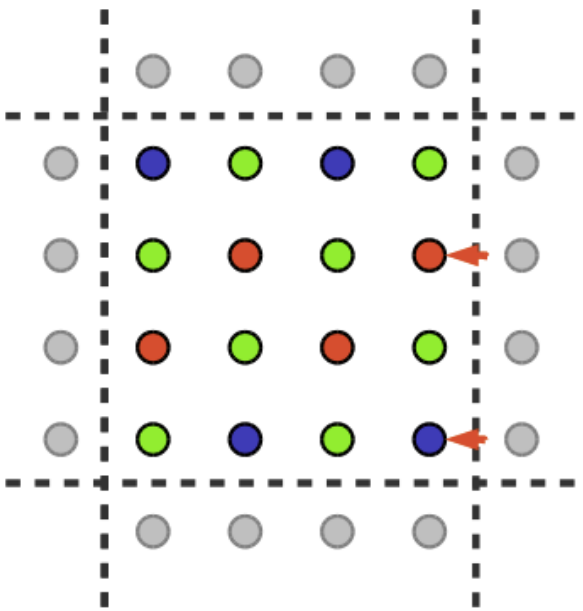
Multi-dimensional Kernel Computation



Step 3

- Boundary sites are updated by a series of kernels - one per direction.
- A given boundary kernel must wait for its ghost zone to arrive
- Note in higher dimensions corner sites have a race condition - serialization of kernels required

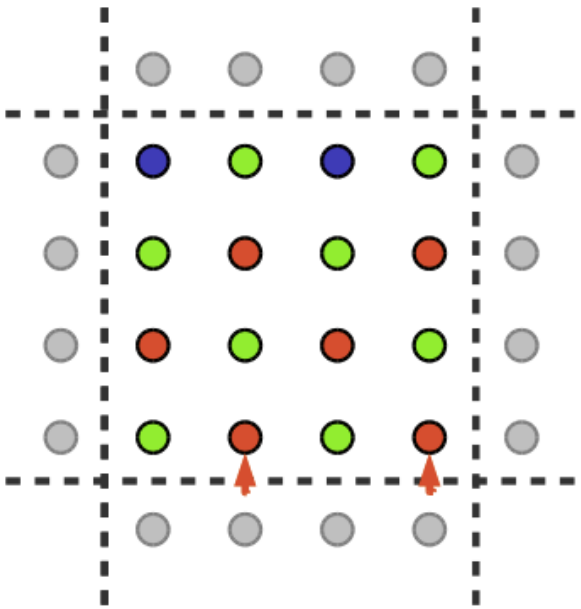
Multi-dimensional Kernel Computation



Step 3

- Boundary sites are updated by a series of kernels - one per direction.
- A given boundary kernel must wait for its ghost zone to arrive
- Note in higher dimensions corner sites have a race condition - serialization of kernels required

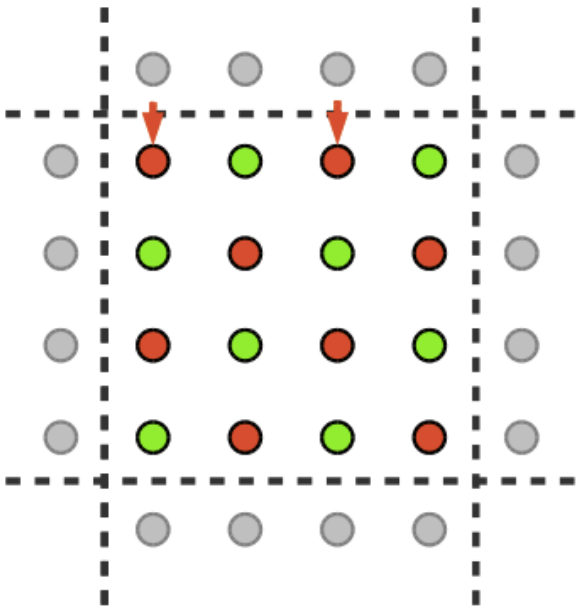
Multi-dimensional Kernel Computation



Step 3

- Boundary sites are updated by a series of kernels - one per direction.
- A given boundary kernel must wait for its ghost zone to arrive
- Note in higher dimensions corner sites have a race condition - serialization of kernels required

Multi-dimensional Kernel Computation



Step 3

- Boundary sites are updated by a series of kernels - one per direction.
- A given boundary kernel must wait for its ghost zone to arrive
- Note in higher dimensions corner sites have a race condition - serialization of kernels required

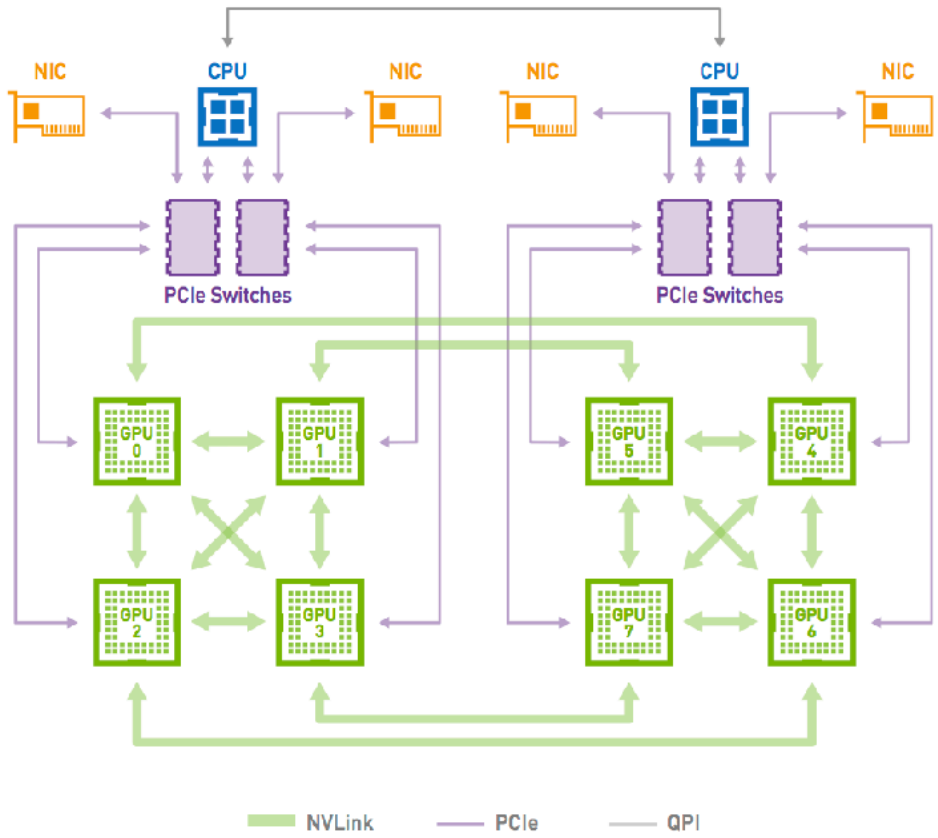
BENCHMARKING TESTBED

NVIDIA Prometheus Cluster

DGX-1V

- 8x V100 GPUs
- Hypercube-Mesh NVLink
- 4x EDR for inter-node communication
- Optimal placement of GPUs and NIC for GDR

CUDA 10.1, GCC 7.3, OpenMPI 3.1



METHODOLOGY

Gain insight from multi-GPU single node performance

Simulate strong scaling, with 1x2x2x2 topology on 8 GPUs

Use “Wilson Dslash” stencil

Then move to multi-node…

```
nvidia-smi topo -m
```

```
      GPU0    GPU1    GPU2    GPU3    GPU4    GPU5    GPU6    GPU7    mlx5_0  mlx5_2  mlx5_1  mlx5_3  CPU Affinity
GPU0     X      NV1     NV1     NV2     NV2     SYS     SYS     SYS     SYS     PIX     SYS     PHB     SYS 0-19,40-59
GPU1    NV1      X      NV2     NV1     SYS     NV2     SYS     SYS     SYS     PIX     SYS     PHB     SYS 0-19,40-59
GPU2    NV1     NV2      X      NV2     SYS     SYS     NV1     SYS     PHB     SYS     PIX     SYS     SYS 0-19,40-59
GPU3    NV2     NV1     NV2      X      SYS     SYS     SYS     NV1     PHB     SYS     PIX     SYS     SYS 0-19,40-59
GPU4    NV2     SYS     SYS     SYS      X      NV1     NV1     NV2     SYS     PIX     SYS     PHB     PHB 20-39,60-79
GPU5    SYS     NV2     SYS     SYS     NV1      X      NV2     NV1     SYS     PIX     SYS     PHB     PHB 20-39,60-79
GPU6    SYS     SYS     NV1     SYS     NV1     NV2      X      NV2     SYS     PHB     SYS     PIX     PIX 20-39,60-79
GPU7    SYS     SYS     SYS     NV1     NV2     NV1     NV2      X      SYS     PHB     SYS     PIX     PIX 20-39,60-79
mlx5_0  PIX     PIX     PHB     PHB     SYS     SYS     SYS     SYS      X      SYS     PHB     SYS
mlx5_2  SYS     SYS     SYS     SYS     PIX     PIX     PHB     PHB     SYS      X      SYS     PHB
mlx5_1  PHB     PHB     PIX     PIX     SYS     SYS     SYS     SYS     PHB     SYS      X      SYS
mlx5_3  SYS     SYS     SYS     SYS     PHB     PHB     PIX     PIX     SYS     PHB     SYS      X

Legend:

  X    = Self
  SYS  = Connection traversing PCIe as well as the SMP interconnect between NUMA nodes (e.g., QPI/UPI)
  NODE = Connection traversing PCIe as well as the interconnect between PCIe Host Bridges within a NUMA node
  PHB  = Connection traversing PCIe as well as a PCIe Host Bridge (typically the CPU)
  PXB  = Connection traversing multiple PCIe switches (without traversing the PCIe Host Bridge)
  PIX  = Connection traversing a single PCIe switch
  NV#  = Connection traversing a bonded set of # NVLinks
```

Binding script with explicit NUMA binding and NIC assignment

https://github.com/lattice/quda/wiki/Multi-GPU-Support#maximizing-gdr-performance

LEGACY IMPLEMENTATION (2011)

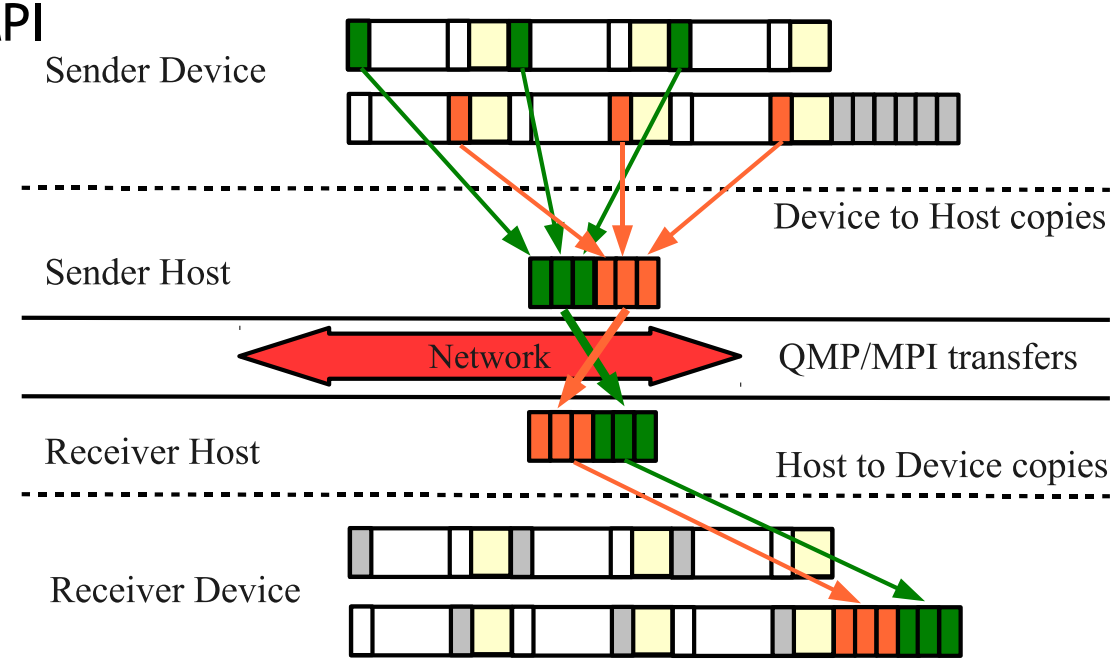
Early CUDA had no interoperability with MPI

Stage MPI buffers in CPU memory

Early large-scale machines ~1 GPU / node

GPUs relatively slower

Host staging reasonable approach

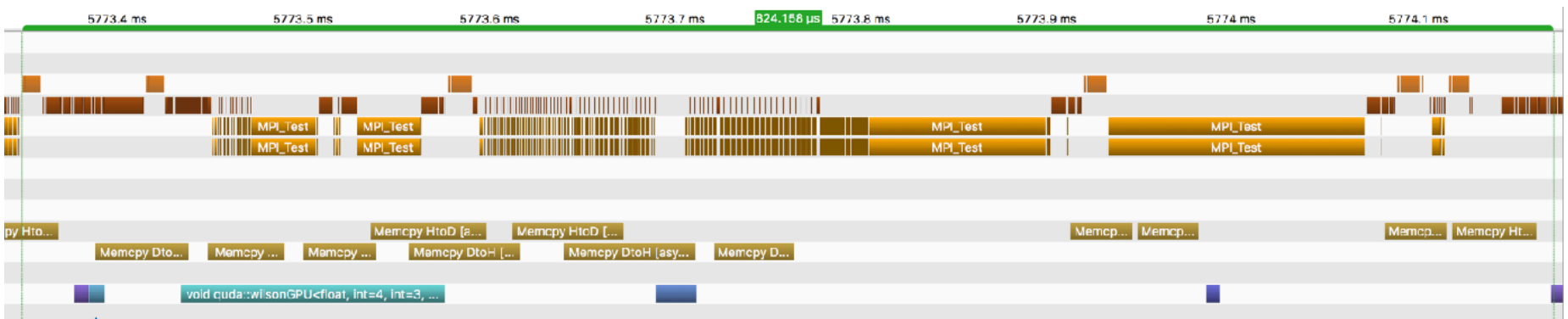


SINGLE NODE PERFORMANCE

Host Staging

Host-Device transfers and MPI exchange dominate the execution

24^4 half precision timeline



Interior kernel

Halo t

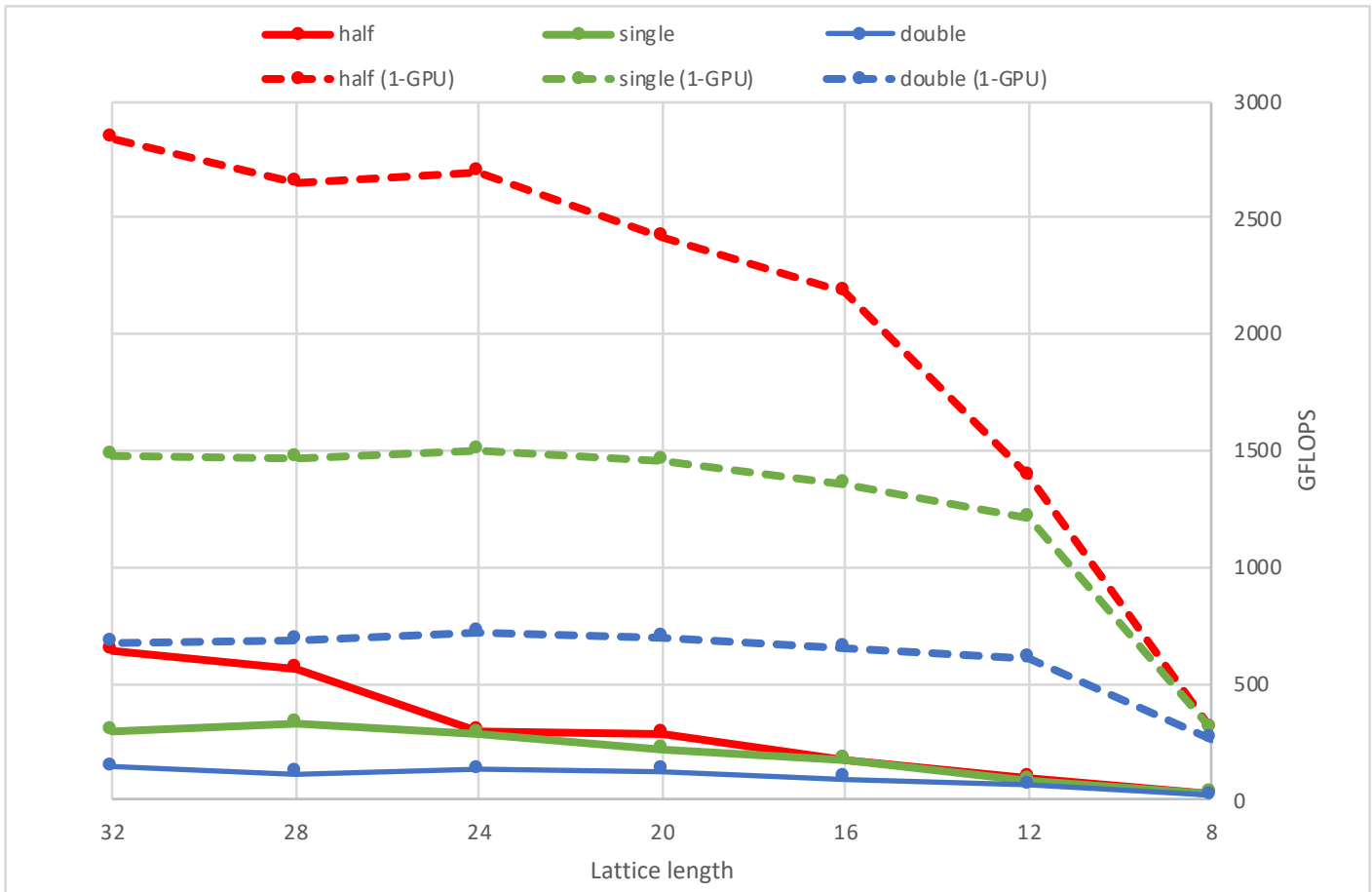
Halo z

Halo y

Packing kernel

SINGLE NODE PERFORMANCE

Host Staging



DGX-1V, 1x2x2x2 partitioning

ENABLING NVLINK COMMUNICATION

Three possible ways to utilize NVLink inter-GPU connections

1.) Use CUDA-aware MPI

Easiest

Can just work out of the box

2.) Copy Engines

Bandwidth

3.) Direct reading / writing from kernels

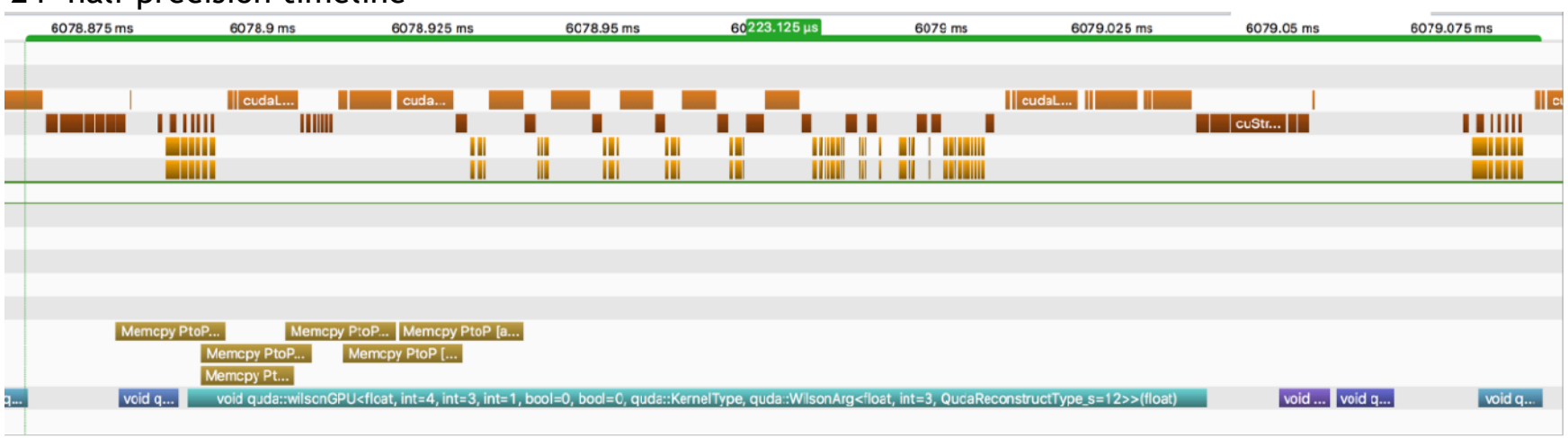
Least latency since a single kernel can write to multiple GPUs

SINGLE NODE PERFORMANCE

Copy Engines

Communication is completely hidden when using NVLink at larger volumes

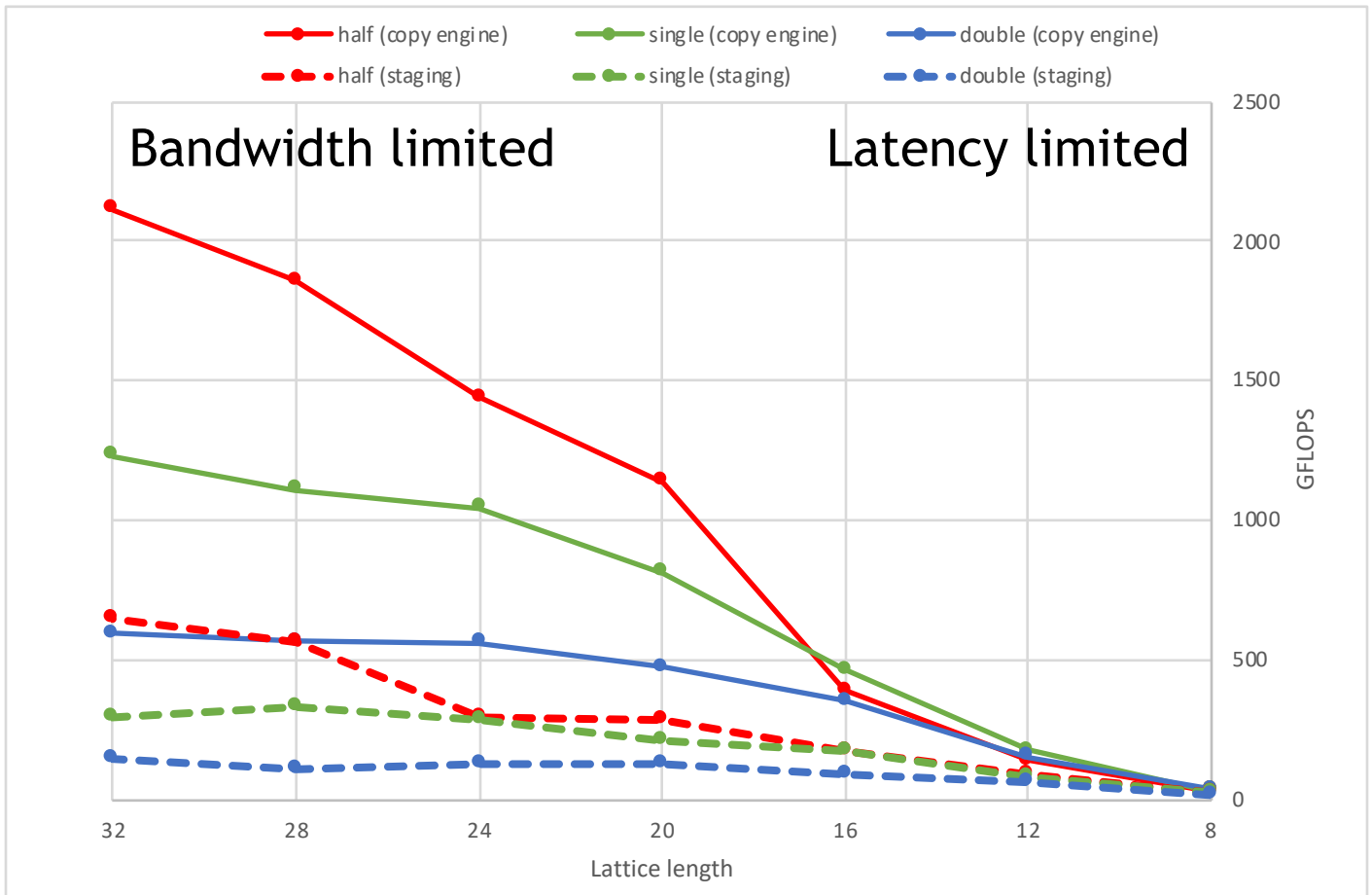
24^4 half precision timeline



Packing kernel Interior kernel Halo t Halo z Halo y

SINGLE NODE PERFORMANCE

Copy Engines



DGX-1V, 1x2x2x2 partitioning

LATENCY OPTIMIZATION

STRONG SCALING

Multiple meanings

- Same problem size, more nodes, more GPUs
- Same problem, next generation GPUs
- Multigrid - strong scaling within the same run (not discussed here)

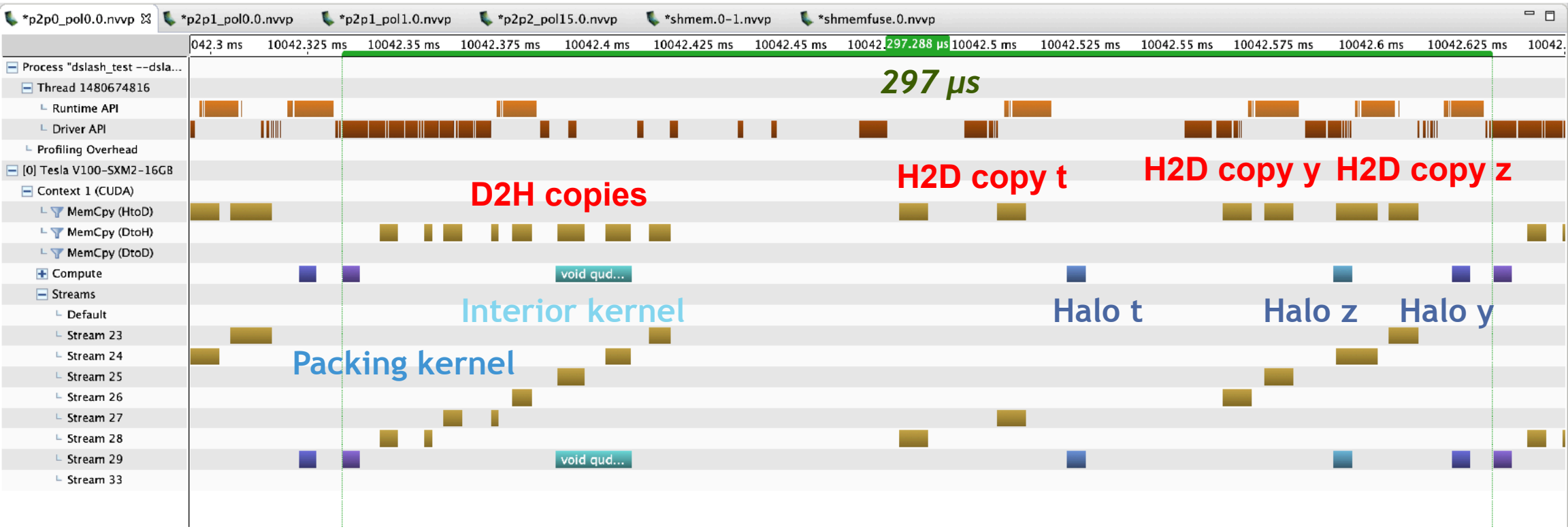
To tame strong scaling we have to understand the limiters

- Bandwidth limiters
- Latency limiters

Look at scaling of a half precision Dslash with 16^4 local volume on one DGX-1

WHAT IS LIMITING STRONG SCALING

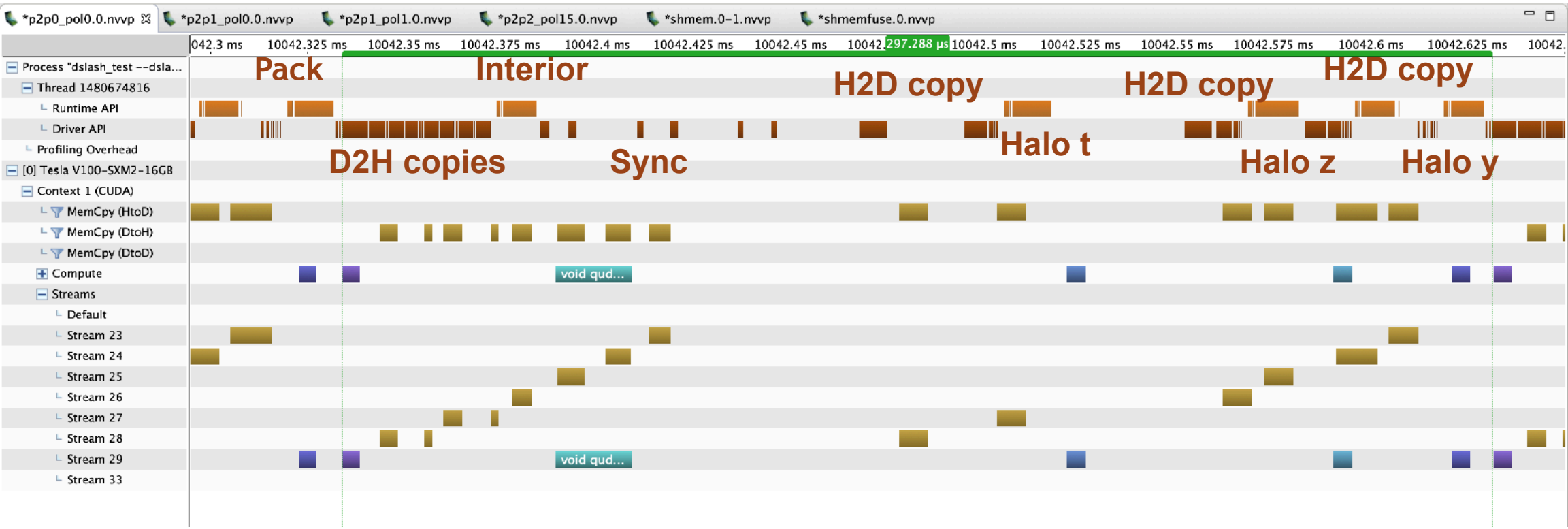
classical host staging



DGX-1,16^4 local volume, half precision, 1x2x2x2 partitioning

API OVERHEADS

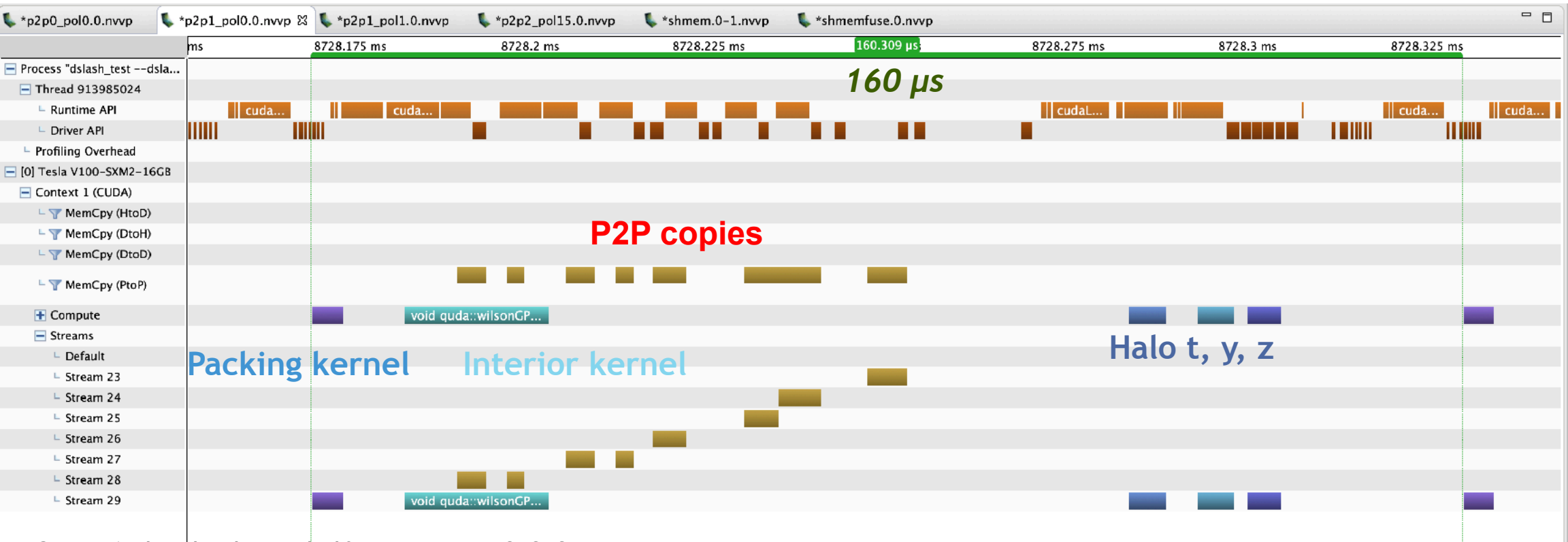
CPU overheads and synchronization are expensive



DGX-1,16^4 local volume, half precision, 1x2x2x2 partitioning

P2P TRANSFERS

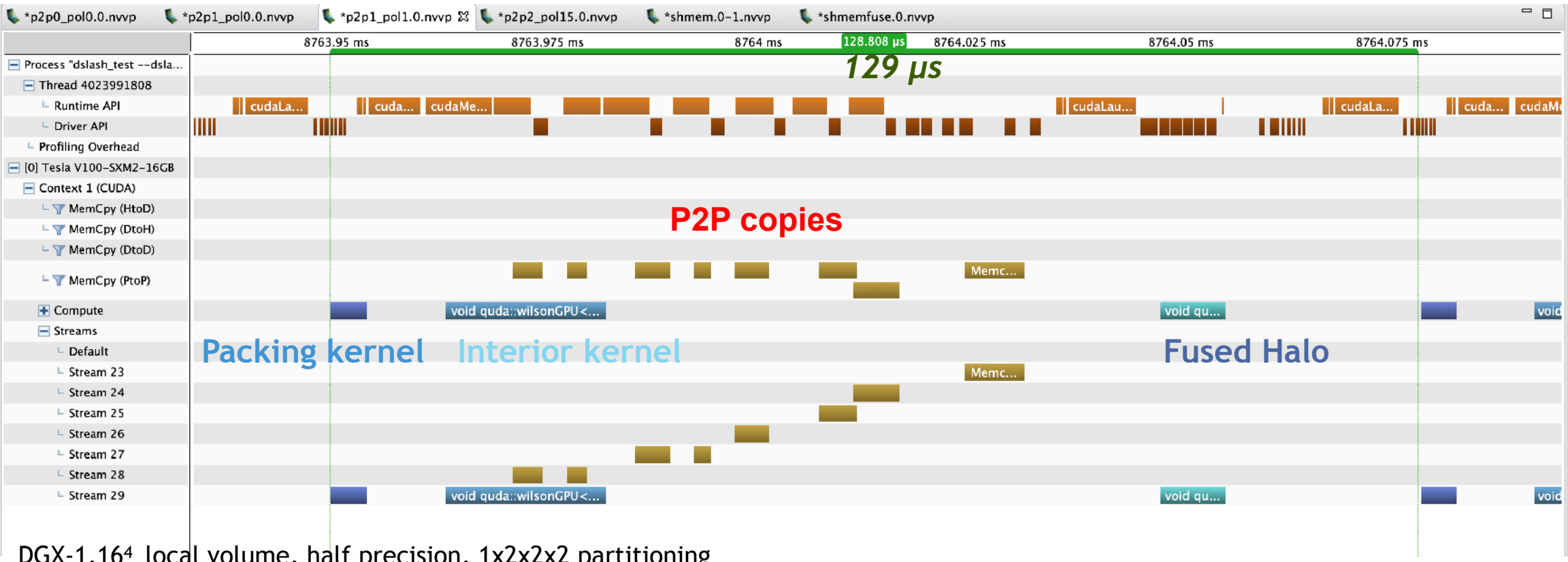
use NVLink, only 1 copy instead of D2H + H2D pair, higher bandwidth



DGX-1, 16^4 local volume, half precision, 1x2x2x2 partitioning

FUSING KERNELS

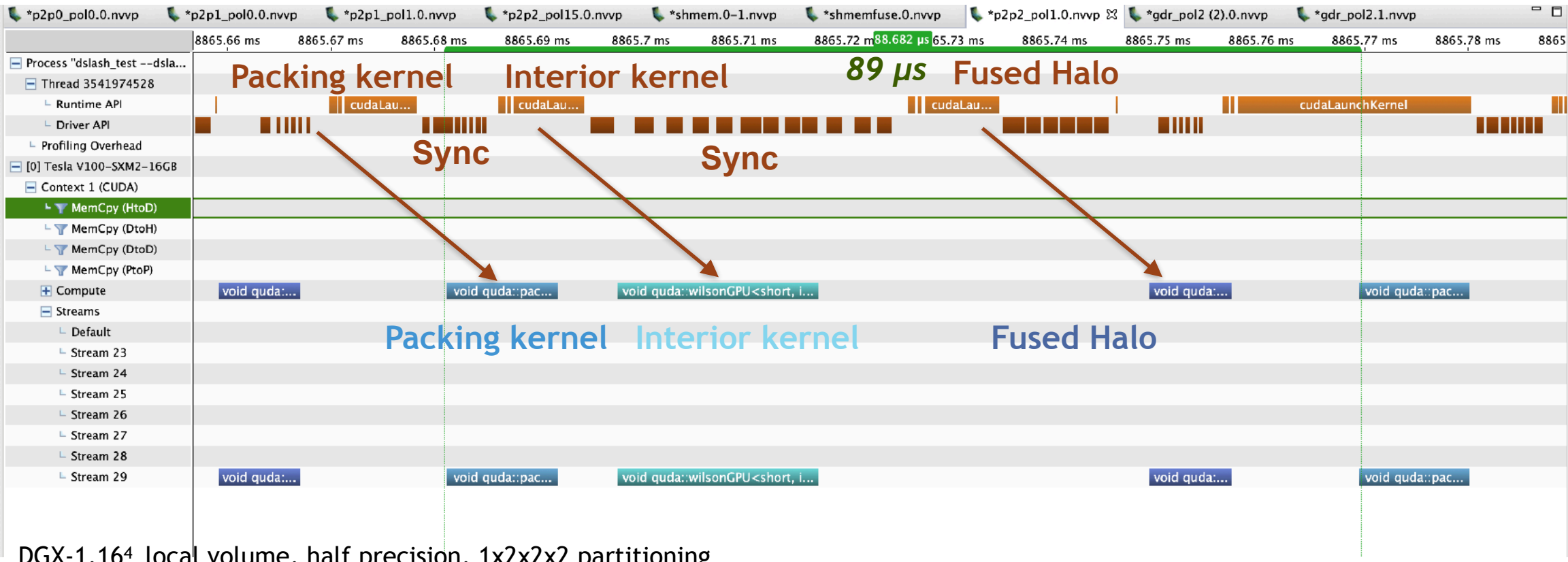
halo kernels do not saturate GPU



DGX-1, 16^4 local volume, half precision, 1x2x2x2 partitioning

REMOTE WRITE

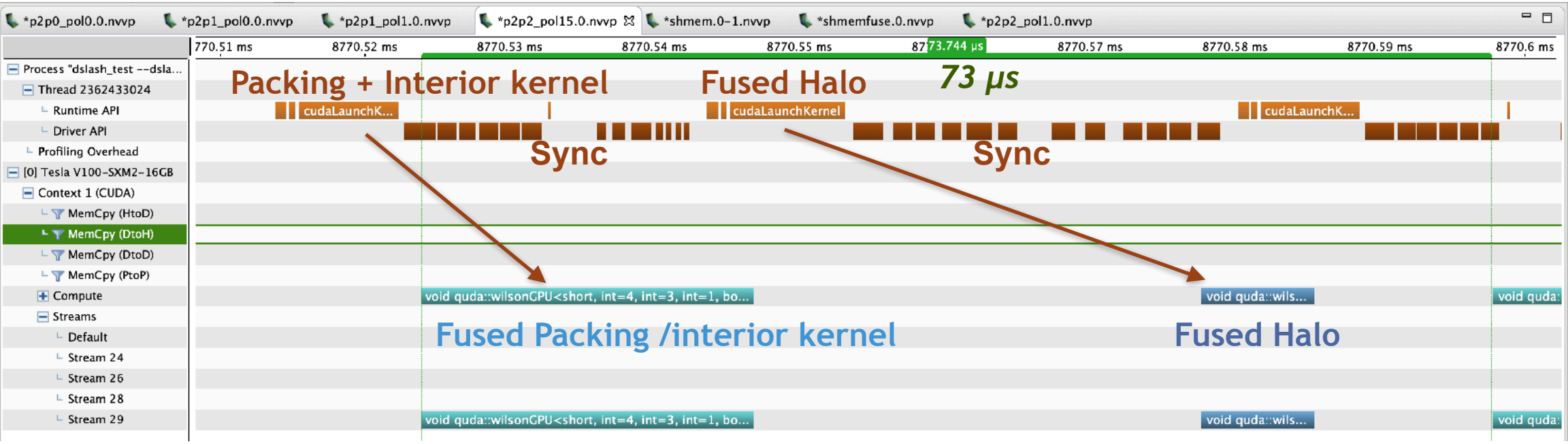
Packing kernel writes to remote GPU using CUDA IPC



DGX-1, 16^4 local volume, half precision, 1x2x2x2 partitioning

MERGING KERNELS

Packing and interior merged with remote write (ok for intranode)



DGX-1,16^4 local volume, half precision, 1x2x2x2 partitioning

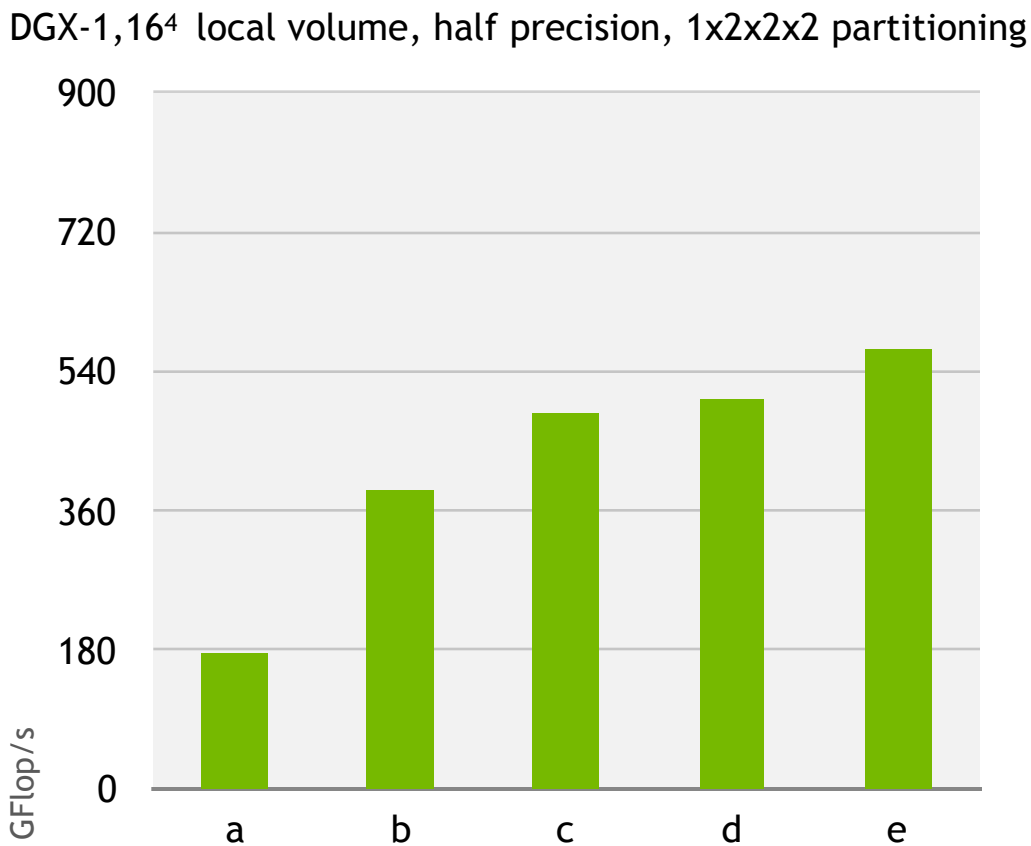
LATENCY OPTIMIZATIONS

Different strategies implemented

a) baseline
b) use P2P copies
c) fuse halo kernels
d) use remote write to neighbor GPU
e) fuse packing and interior

reduces overhead through
- fewer API calls
- fewer kernel launches

still CPU synchronization and API overheads

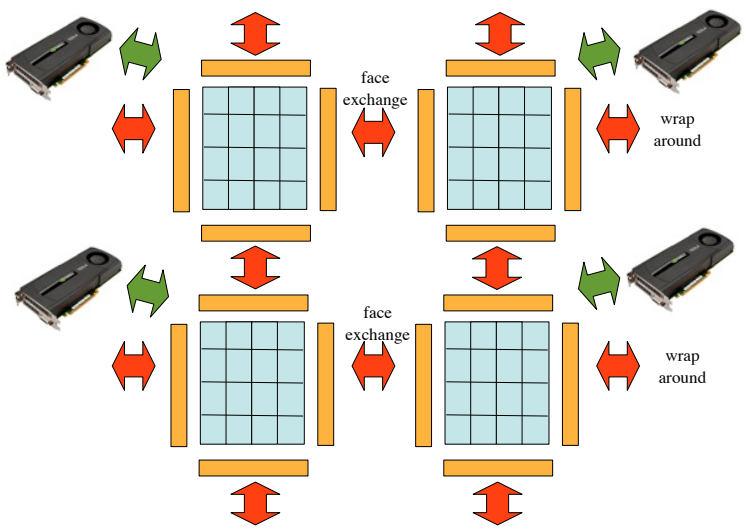


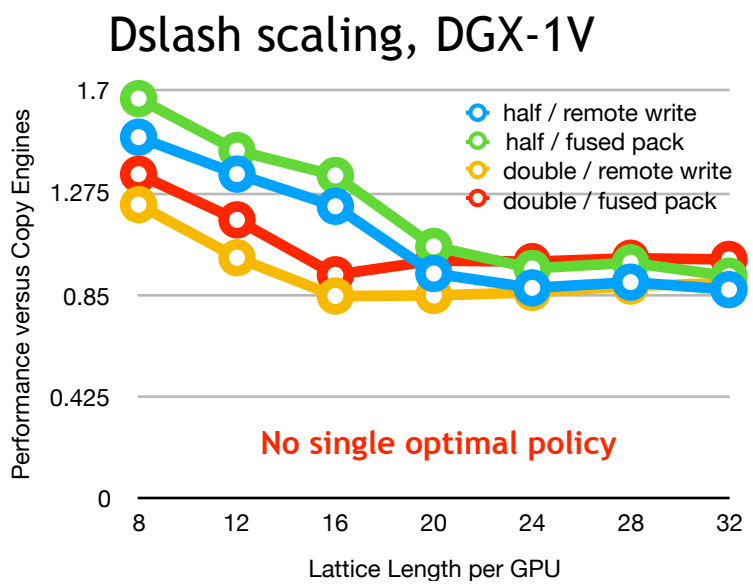
POLICY AUTOTUNING

extended the autotuner to go beyond kernel tuning

What policy to use?

(CE vs remote write) ⊗ (Zero copy vs GDR vs staging) ⊗ kernel fusion





CUDA AWARE MPI

Hit or miss for strong scaling

preferred over manual host staging

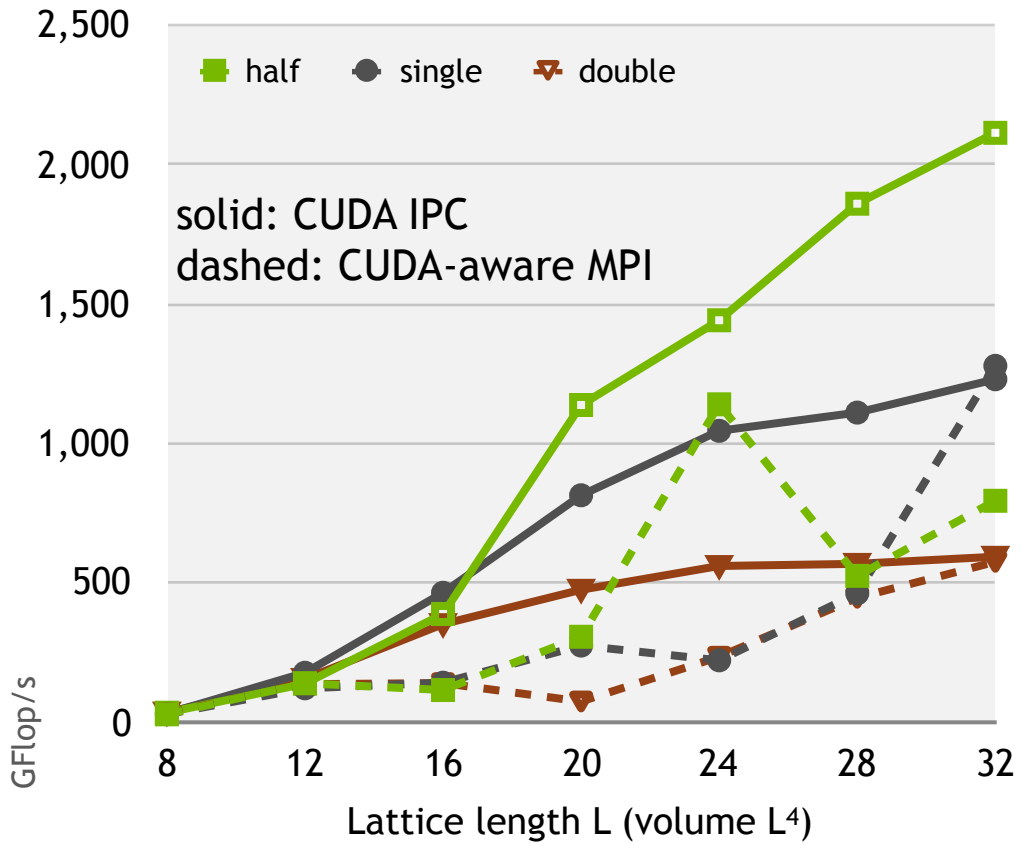
can use CUDA IPC for intra-node

heuristics for transfer protocol

performance is implementation dependent

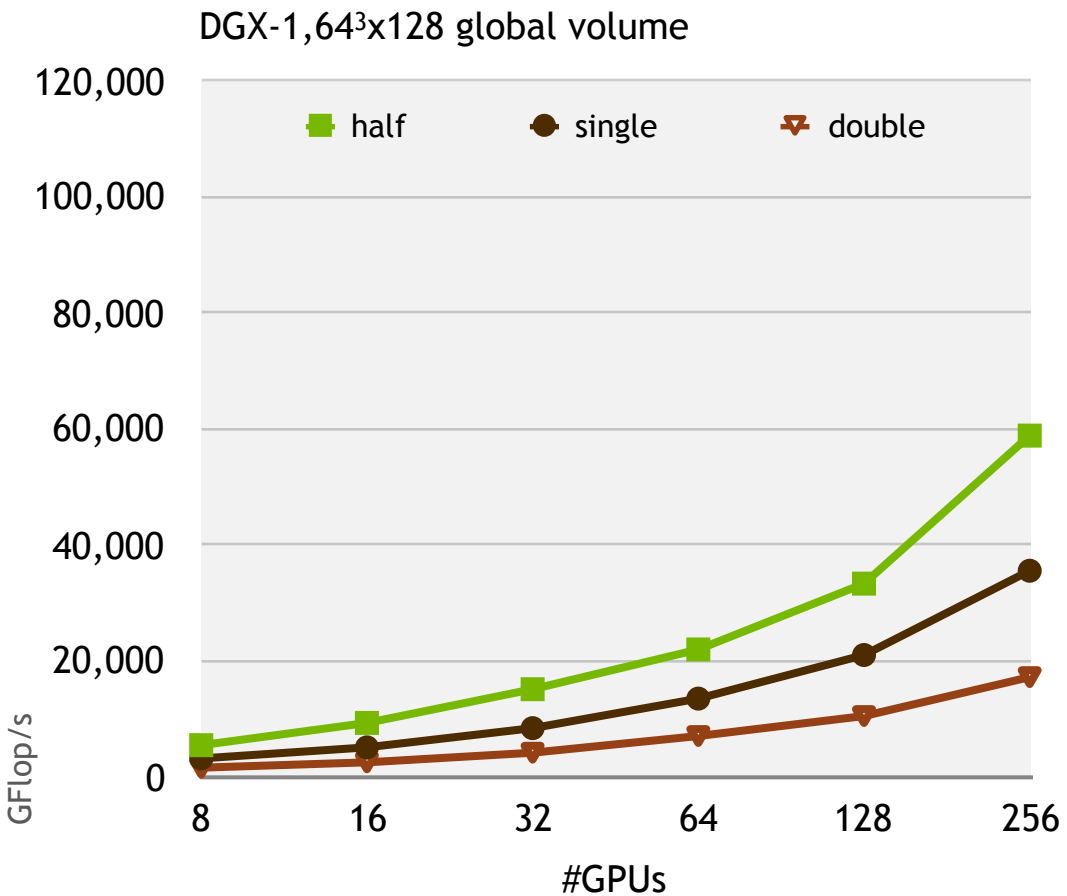
Great for inter-node

- use GPUDirect RDMA
- used in QUDA



MULTI-NODE SCALING

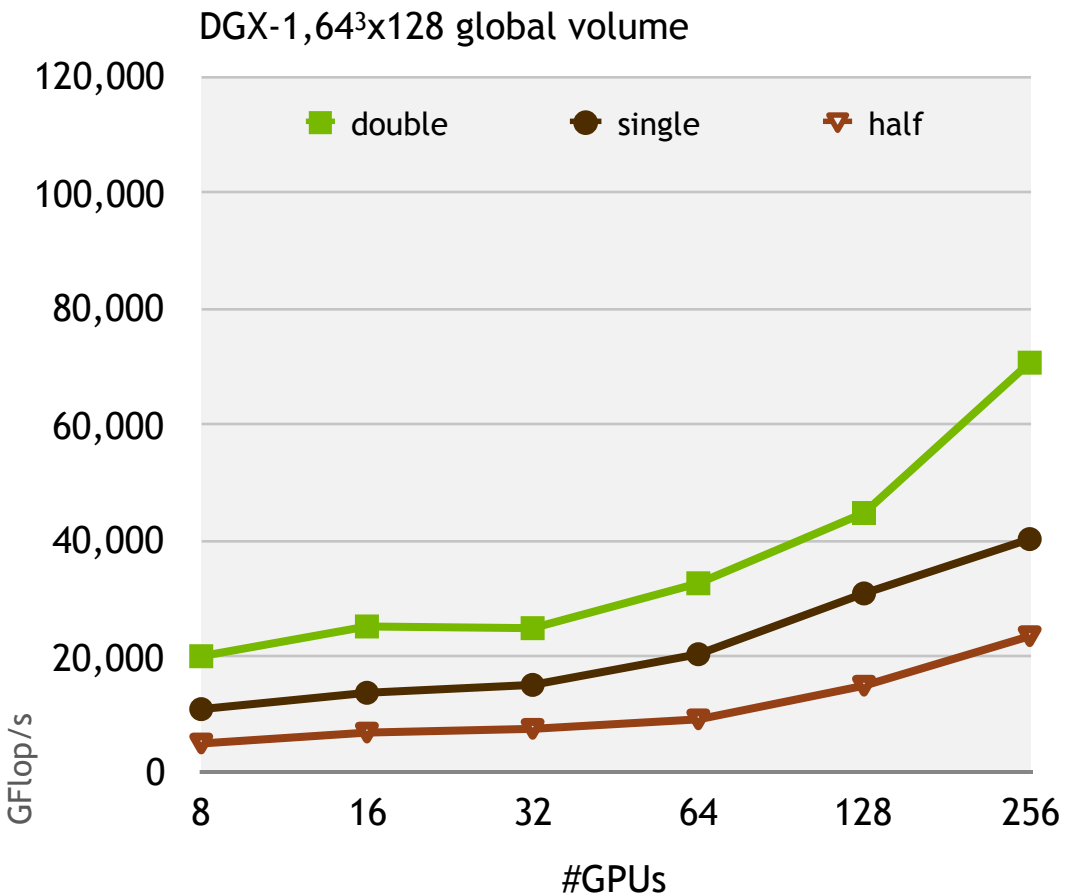
autotuner will pick detailed policy



Host staging

MULTI-NODE SCALING

autotuner will pick detailed policy

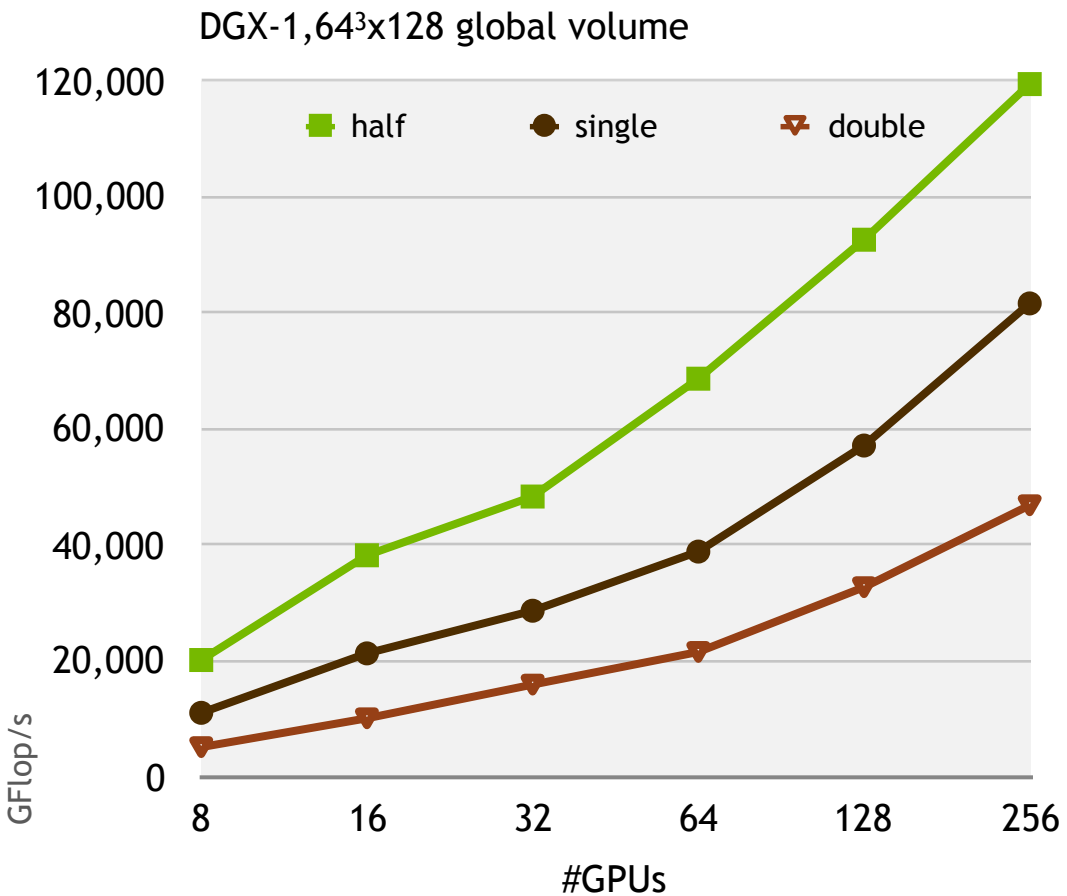


Host staging

Intranode with CUDA IPC

MULTI-NODE SCALING

autotuner will pick detailed policy



Host staging

Intranode with CUDA IPC

CUDA IPC + GPU Direct RDMA


NVSHMEM

NVSHMEM

GPU centric communication

Implementation of OpenSHMEM, a Partitioned Global Address Space (PGAS) library

- Defines API to (symmetrically) allocate memory that is remotely accessible
- Defines API to access remote data
  - One-sided: e.g. `shmem_putmem`, `shmem_getmem`
  - Collectives: e.g. `shmem_broadcast`

NVSHMEM features

- Symmetric memory allocations in device memory
- Communication API calls on CPU (standard and stream-ordered)
- Allows kernel-side communication (API and LD/ST) between GPUs
- Interoperable with MPI

NVSHMEM STATUS

Research vehicle for designing and evaluating GPU-centric workloads

Early access (EA2) available – please reach out to nvshmem@nvidia.com

Main Features

- NVLink and PCIe support
- InfiniBand support (new)
- X86 and Power9 (new) support
- Interoperability with MPI and OpenSHMEM (new) libraries

Limitation: current version requires device linking (see also S9677)

DSLASH NVSHMEM IMPLEMENTATION

First exploration

Keep general structure of packing, interior and exterior Dslash

Use `nvshmem_ptr` for intra-node remote writes (fine-grained)

Packing buffer is located on remote device

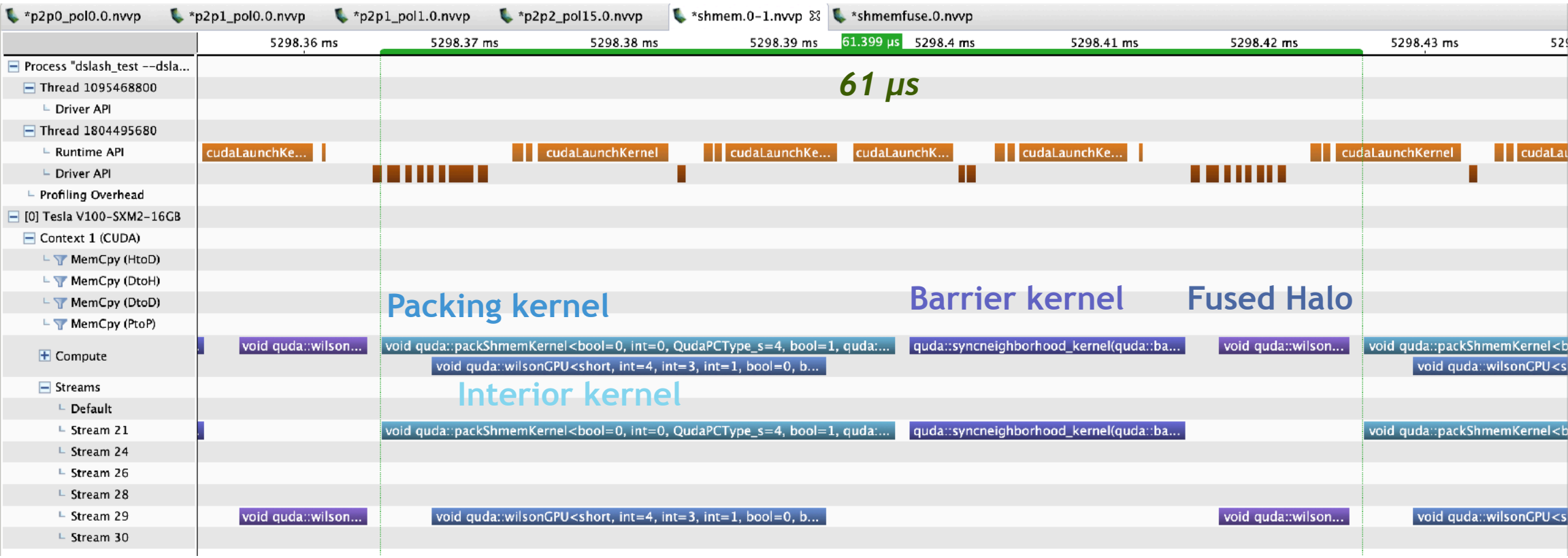
Use `nvshmem_putmem_nbi` to write to remote GPU over IB (1 RDMA transfer)

Need to make sure writes are visible: `nvshmem_barrier_all_on_stream`

or barrier kernel that only waits for writes from neighbors

Disclaimer:
Results from an first implementation in QUDA with a pre-release version of NVSHMEM

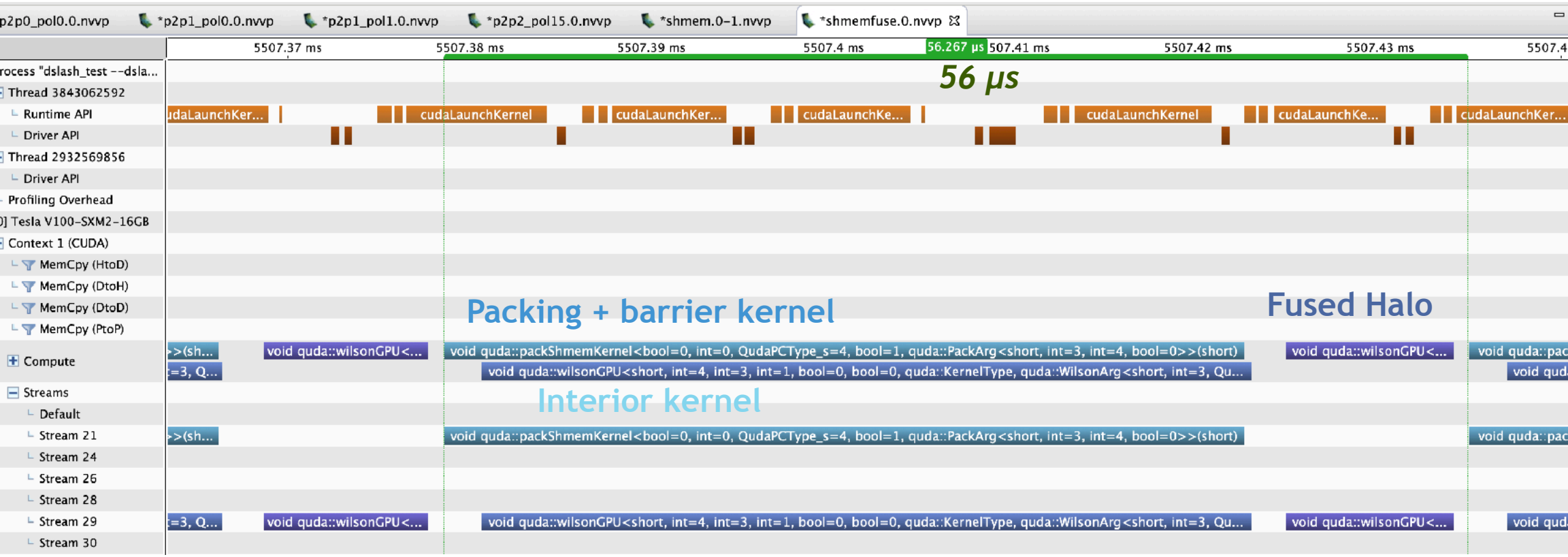
NVSHMEM DSLASH



DGX-1, 16^4 local volume, half precision, 1x2x2x2 partioning

FUSING KERNELS

less Kernel launches

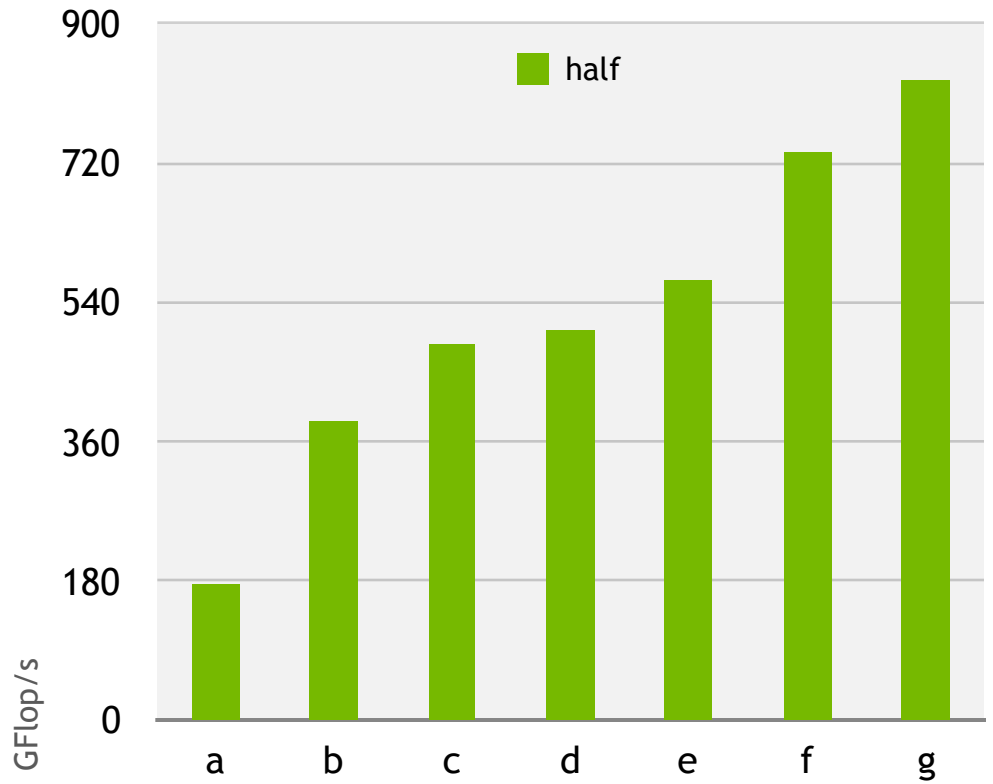


DGX-1, 16^4 local volume, half precision, 1x2x2x2 partioning

LATENCY OPTIMIZATIONS

Different strategies implemented

a) baseline
b) use P2P copies
c) fuse halo kernels
d) use remote write to neighbor GPU
e) fuse packing and interior
f) Shmem
g) Shmem fused packing+barrier



NVSHMEM OUTLOOK

intra-kernel synchronization and communication

SUMMARY

STRONG SCALING LATTICE QCD

Optimize for latency and bandwidth

Autotuning ensures optimal kernel performance and policy selection

Overlap communication and compute for optimal bandwidth

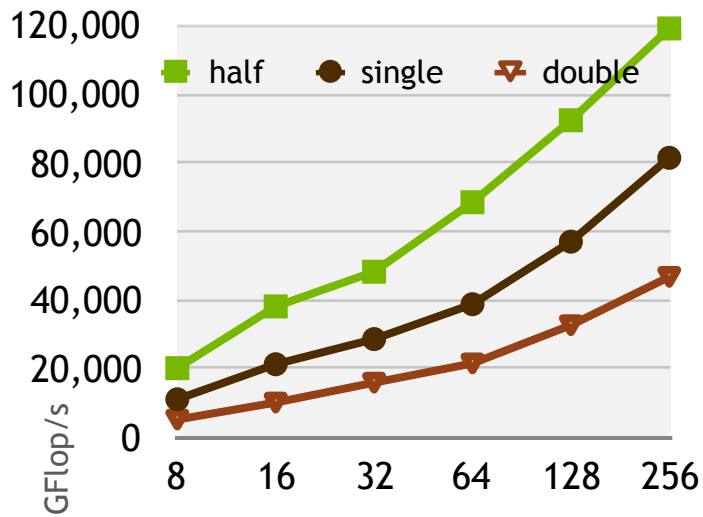
API overheads and CPU/GPU synchronization are costly

prevent overlapping communication and compute

reduce kernel launches and API synchronization calls (fused kernels)

GPU centric communication with NVSHMEM takes CPU limitations out

GPU Direct RDMA techniques for writing data directly to the network




NVIDIA