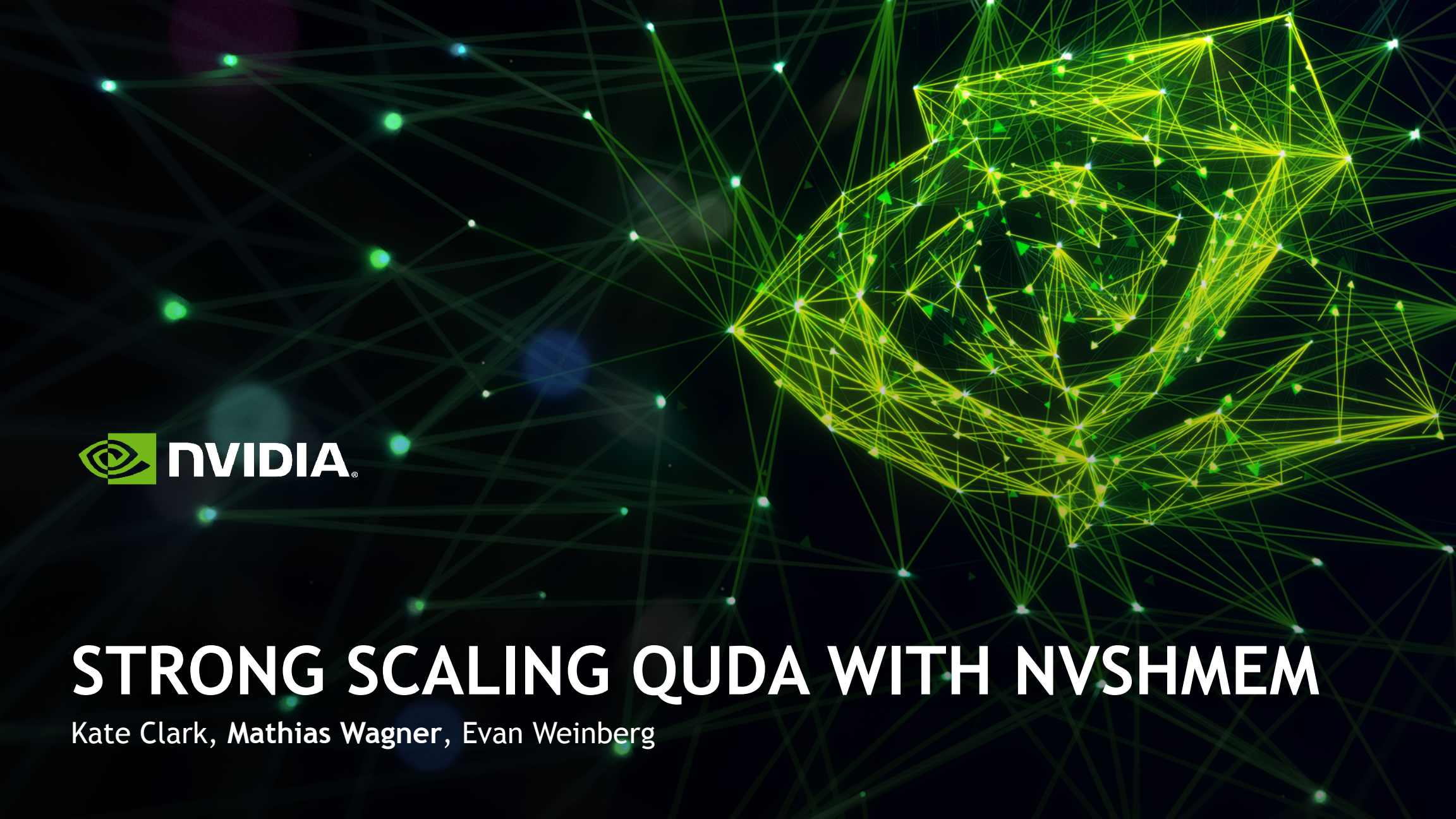


STRONG SCALING QUDA WITH NVSHMEM

Kate Clark, **Mathias Wagner**, Evan Weinberg

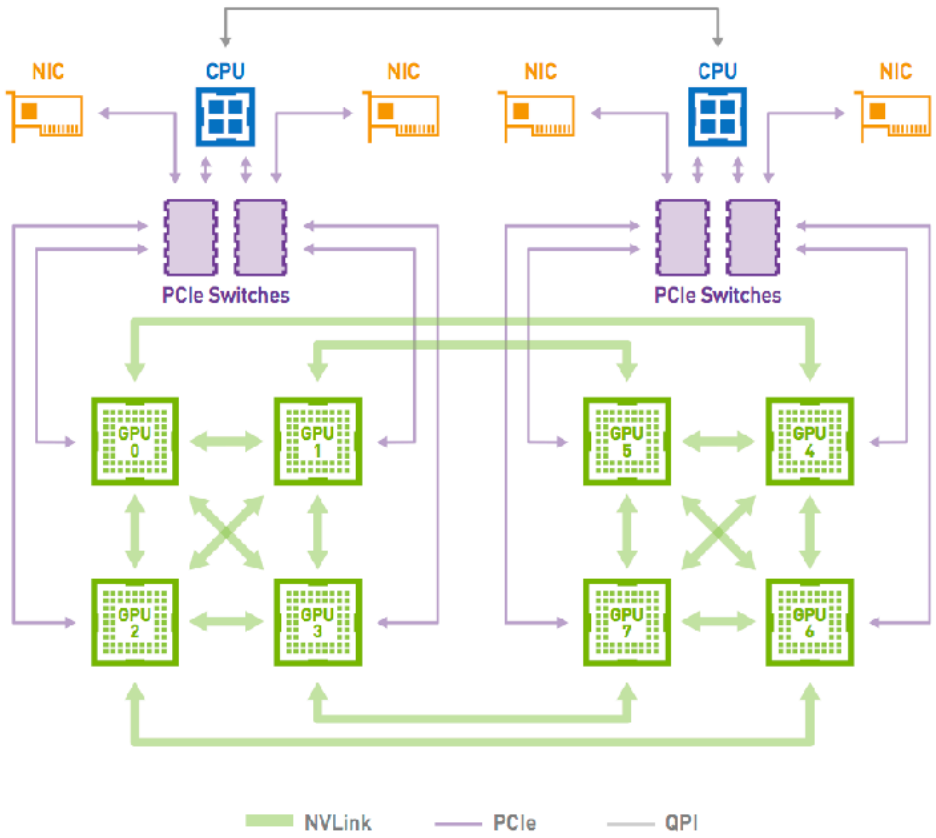
BENCHMARKING TESTBED

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



SCALING OPTIMIZATIONS

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

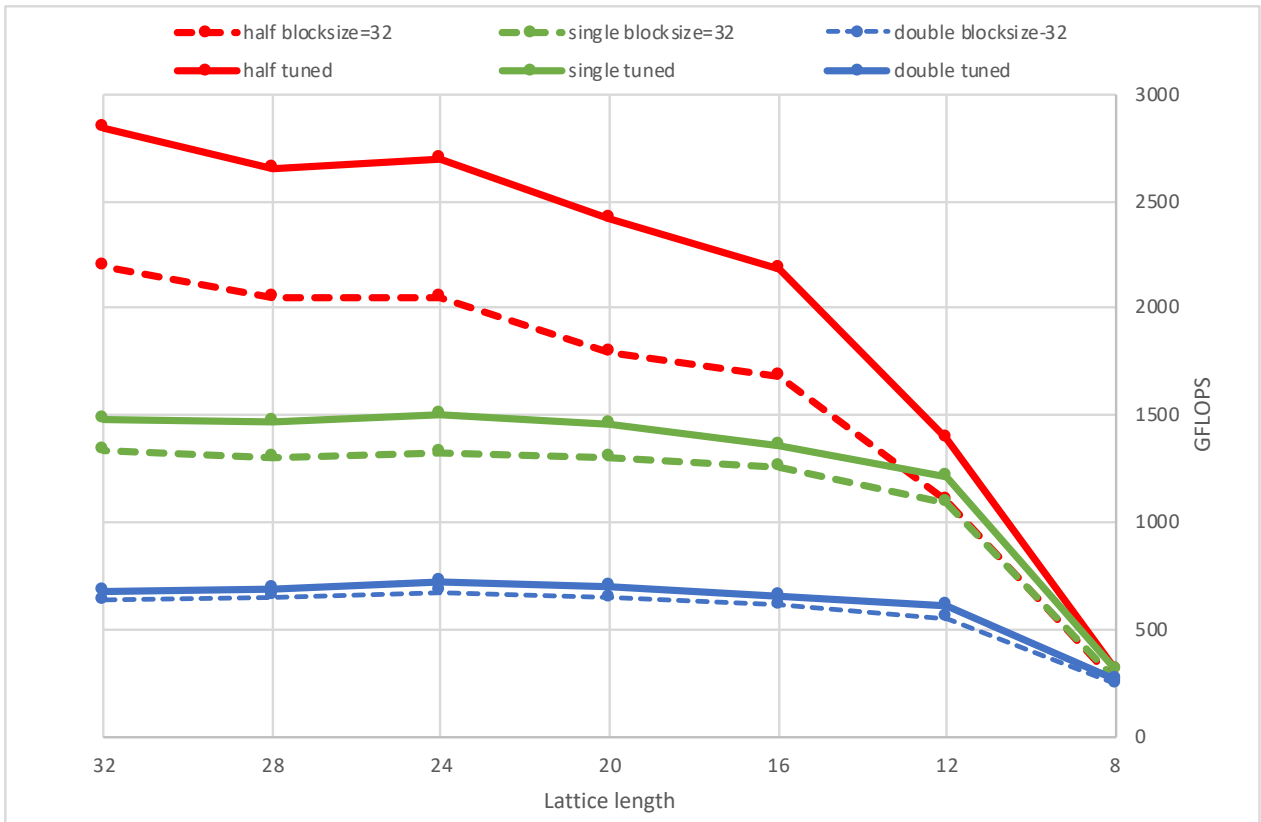
SINGLE GPU PERFORMANCE

“Wilson Dslash” stencil

1312 GB/s

1291 GB/s

1180 GB/s



cf Perfect L2 roofline
~ 1700 GB/s

Tesla V100
CUDA 10.1
GCC 7.3

“strong scaling” ⟶

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)

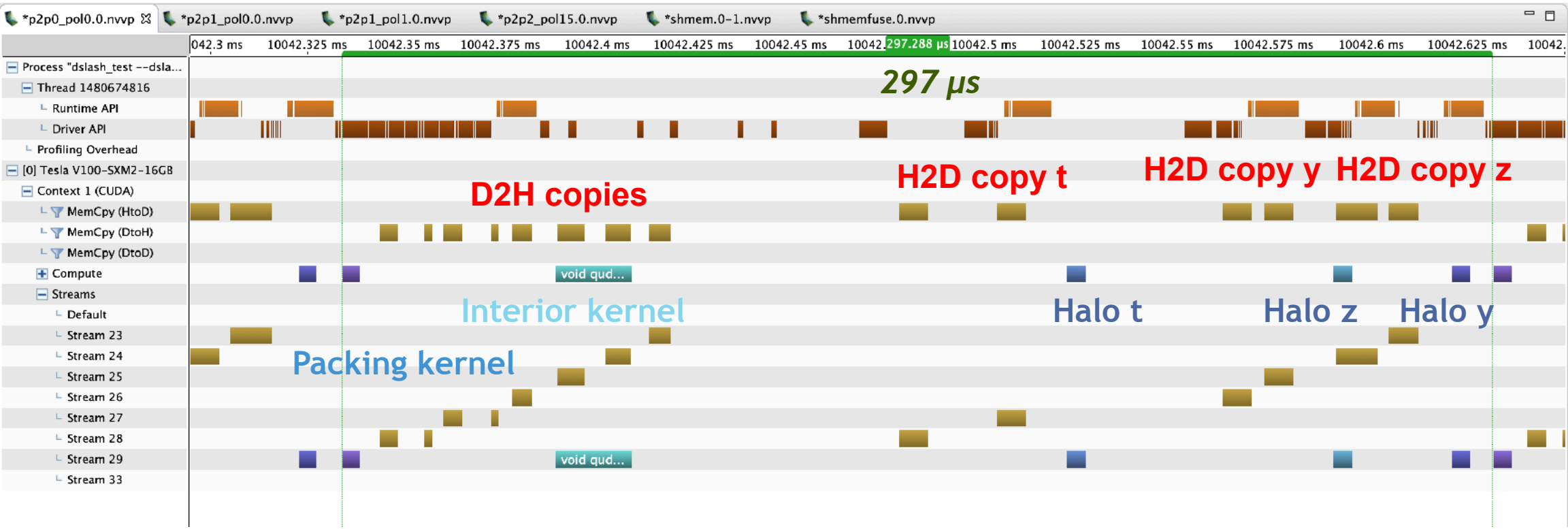
To tame strong scaling we have to understand the limiters

- 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

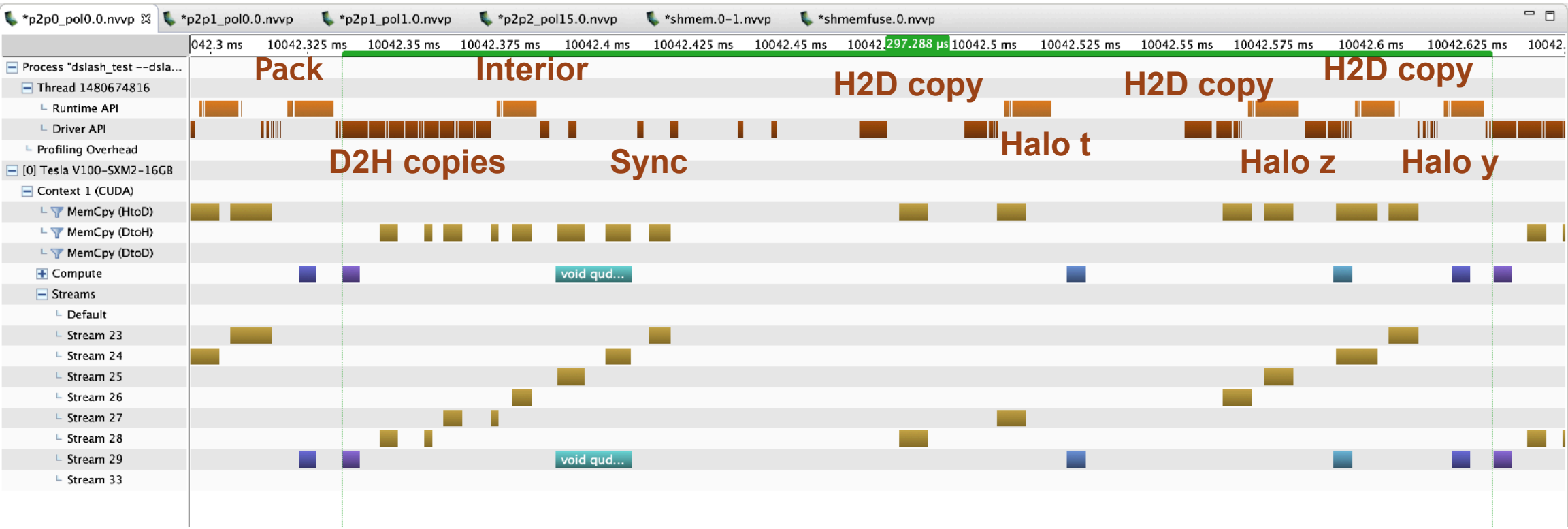
classical host staging



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

API OVERHEADS

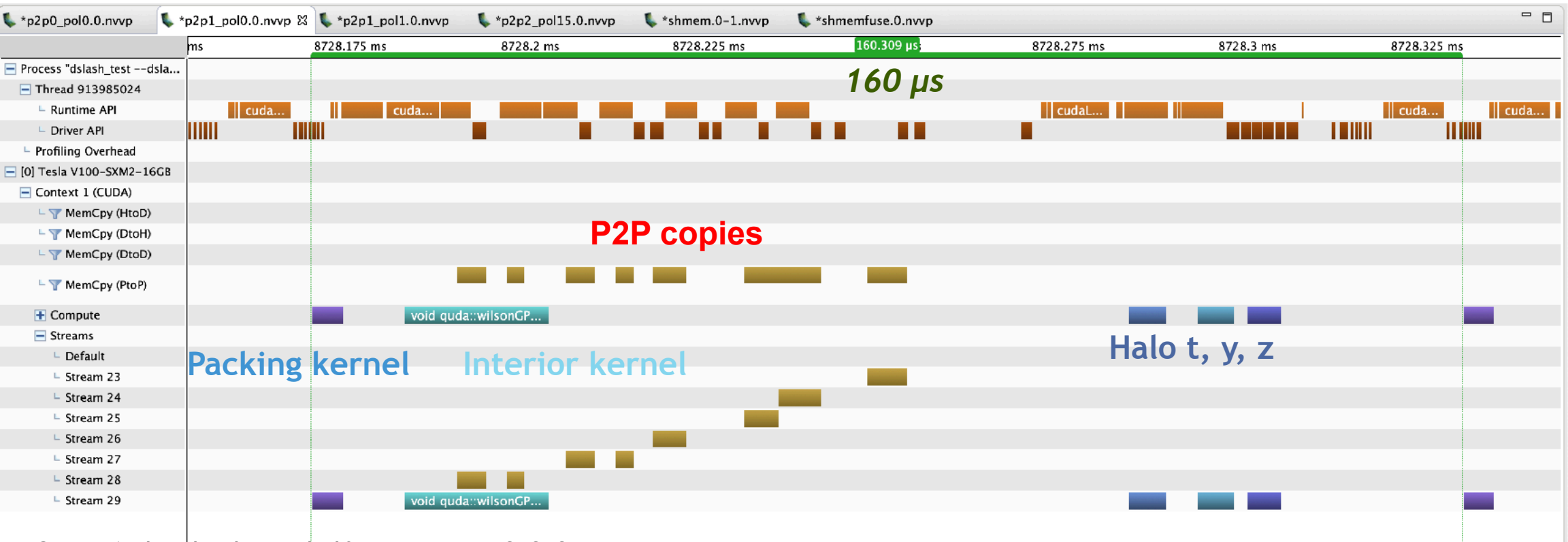
CPU overheads and synchronization are expensive



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

P2P TRANSFERS

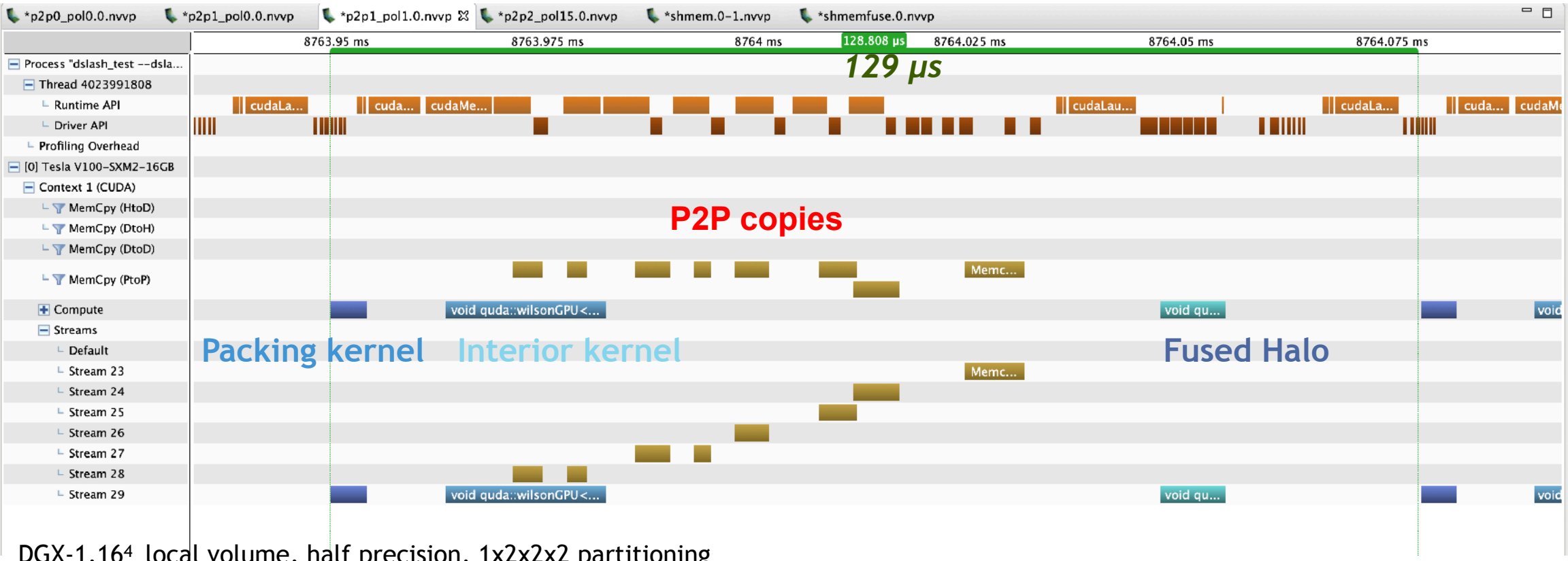
use NVLink, only 1 copy instead of D2H + H2D pair, higher bandwidth



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

FUSING KERNELS

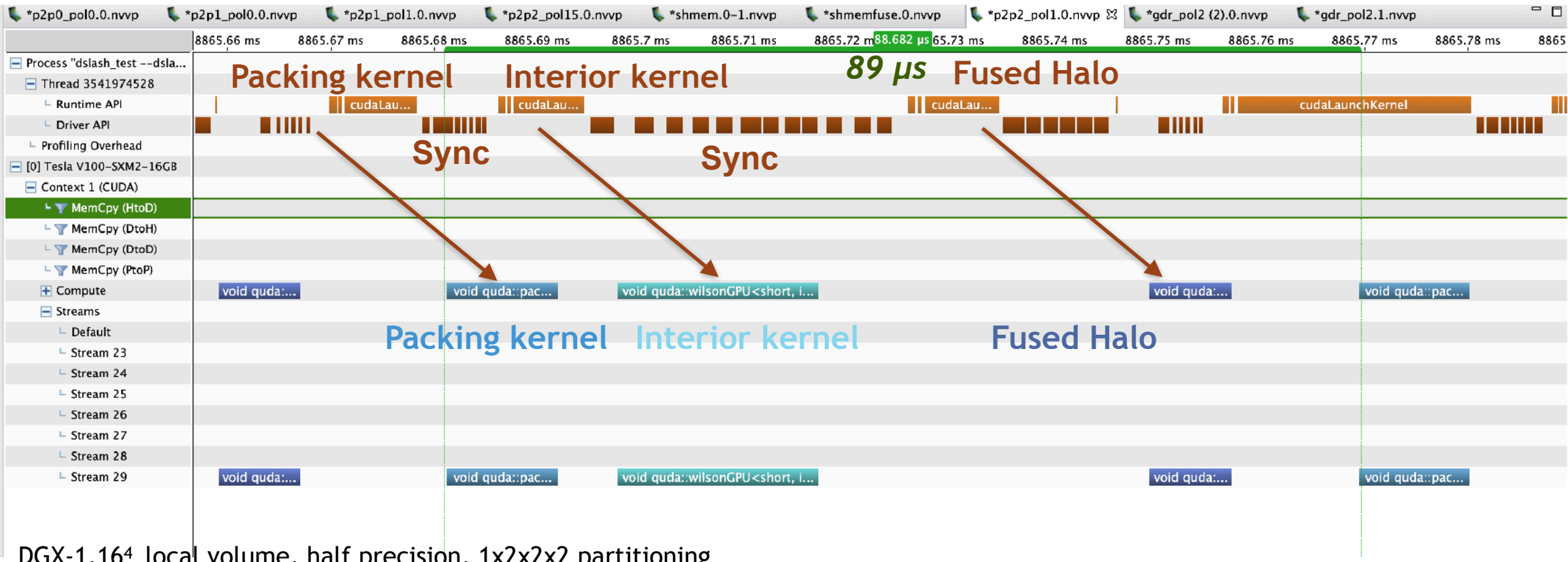
halo kernels do not saturate GPU



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

REMOTE WRITE

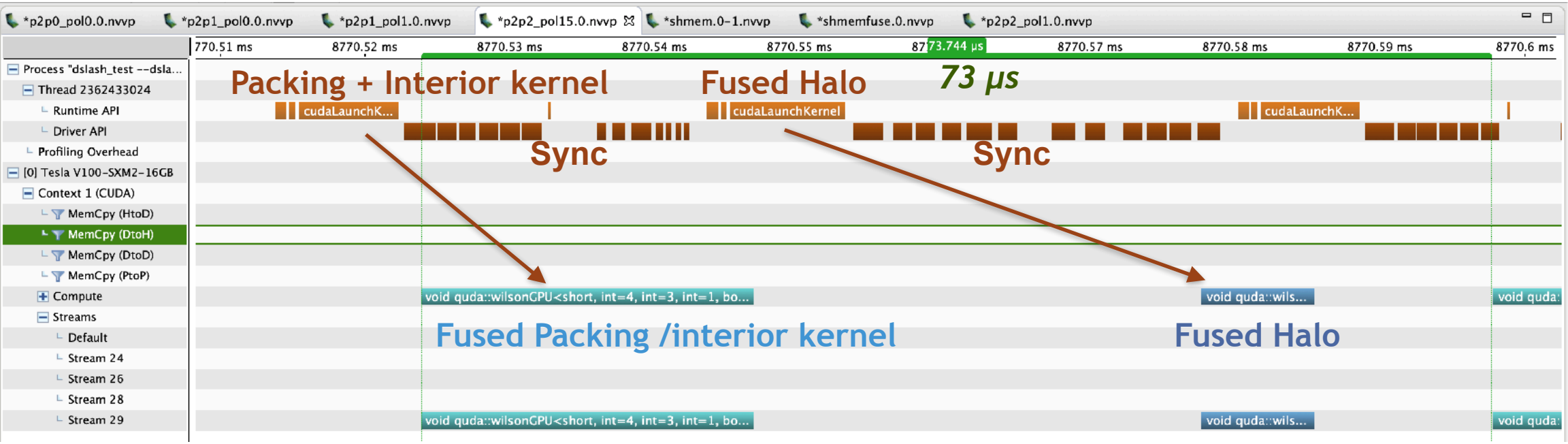
Packing kernel writes to remote GPU using CUDA IPC



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

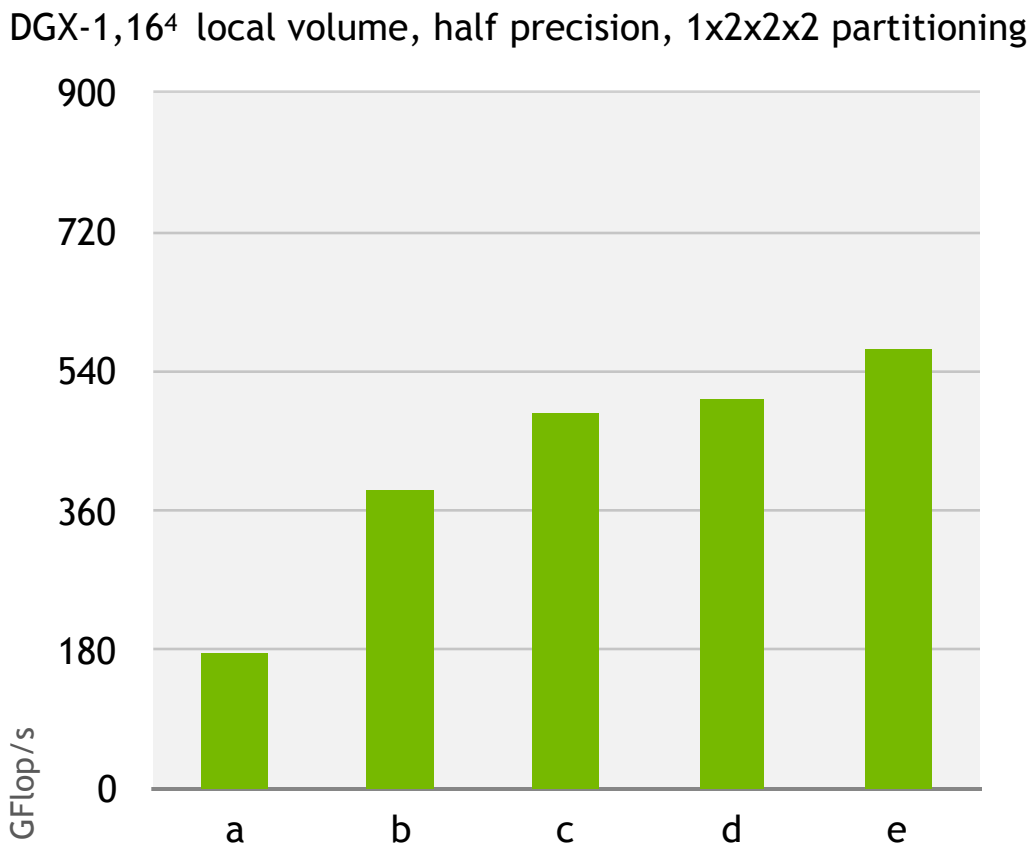
LATENCY OPTIMIZATIONS

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

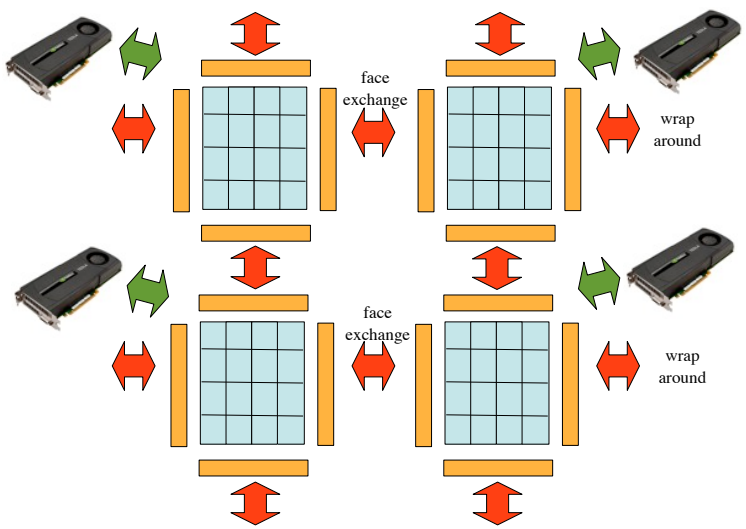


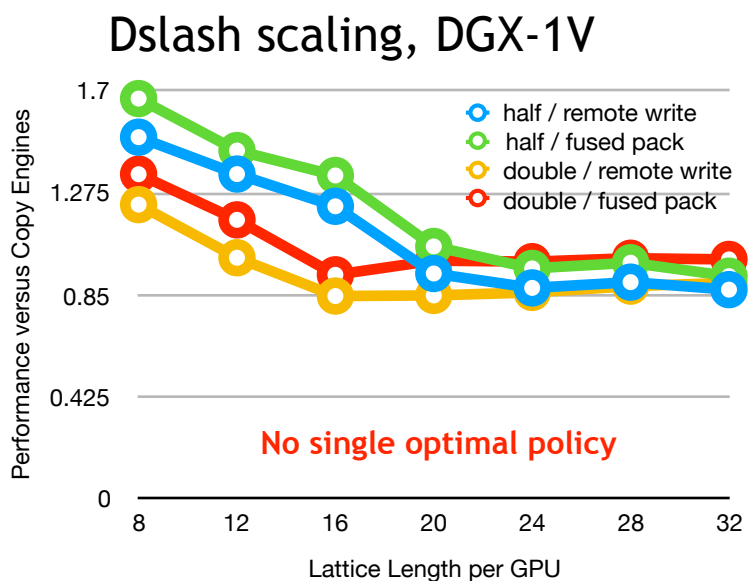
POLICY AUTOTUNING

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

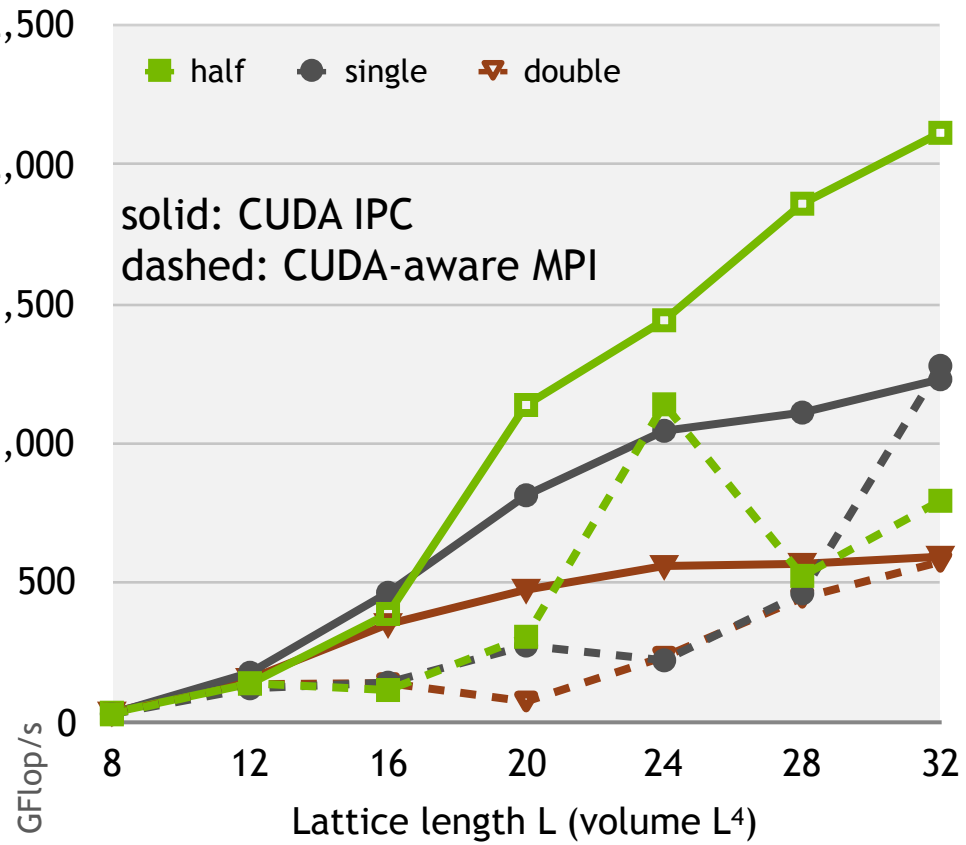
heuristics for transfer protocol

performance is implementation dependent

Great for inter-node

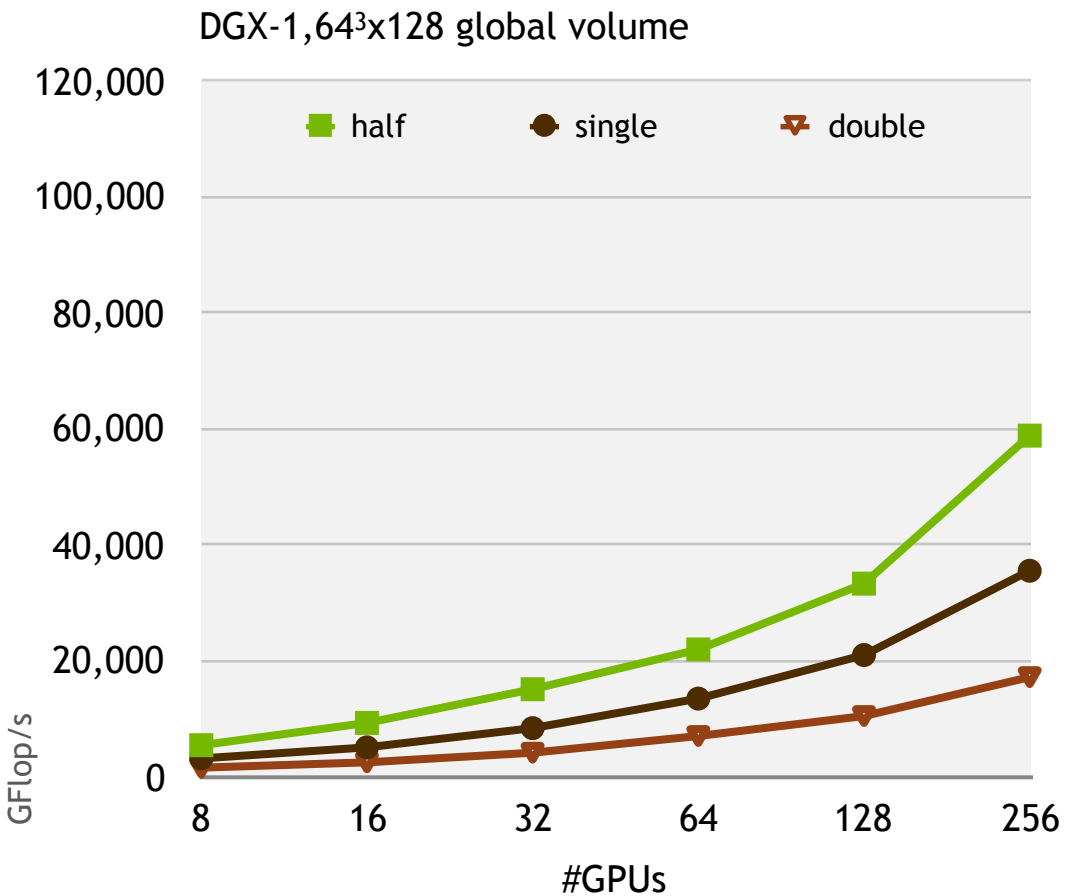
GPUDirect RDMA

data from GPU directly transferred to NIC



MULTI-NODE SCALING

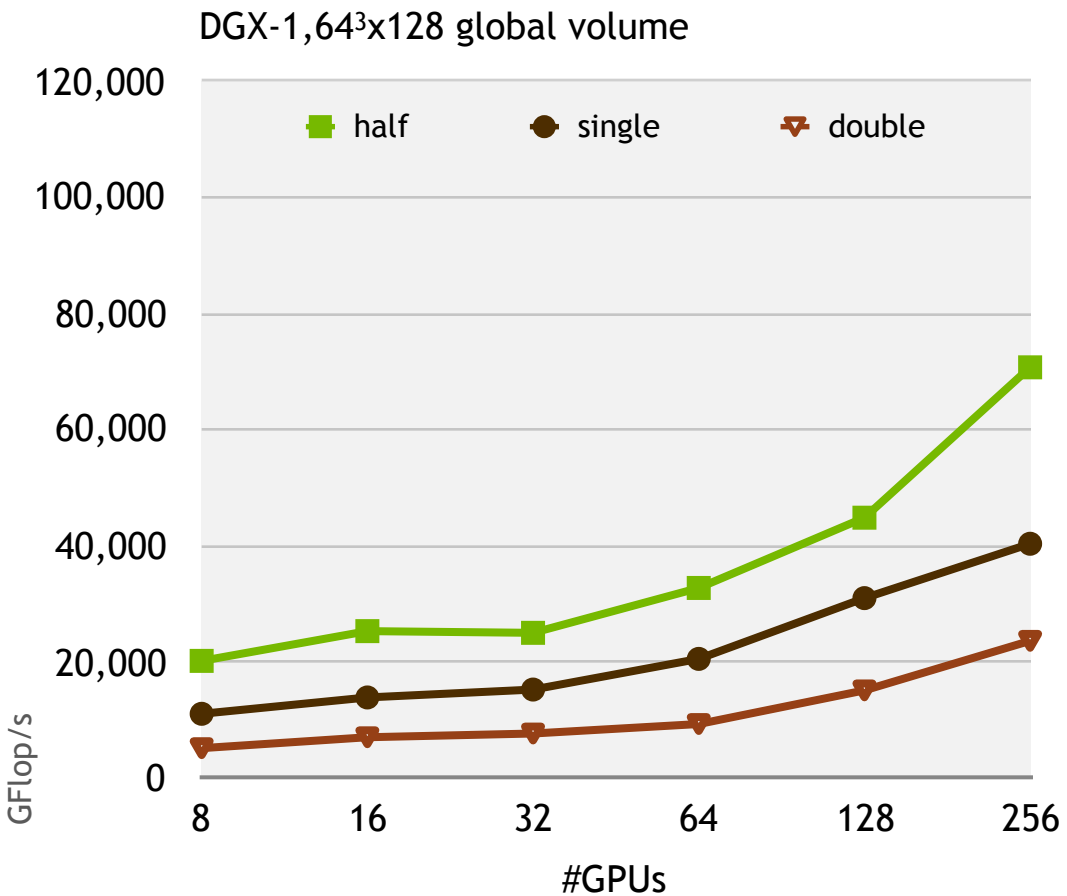
autotuner will pick detailed policy



Host staging

MULTI-NODE SCALING

autotuner will pick detailed policy

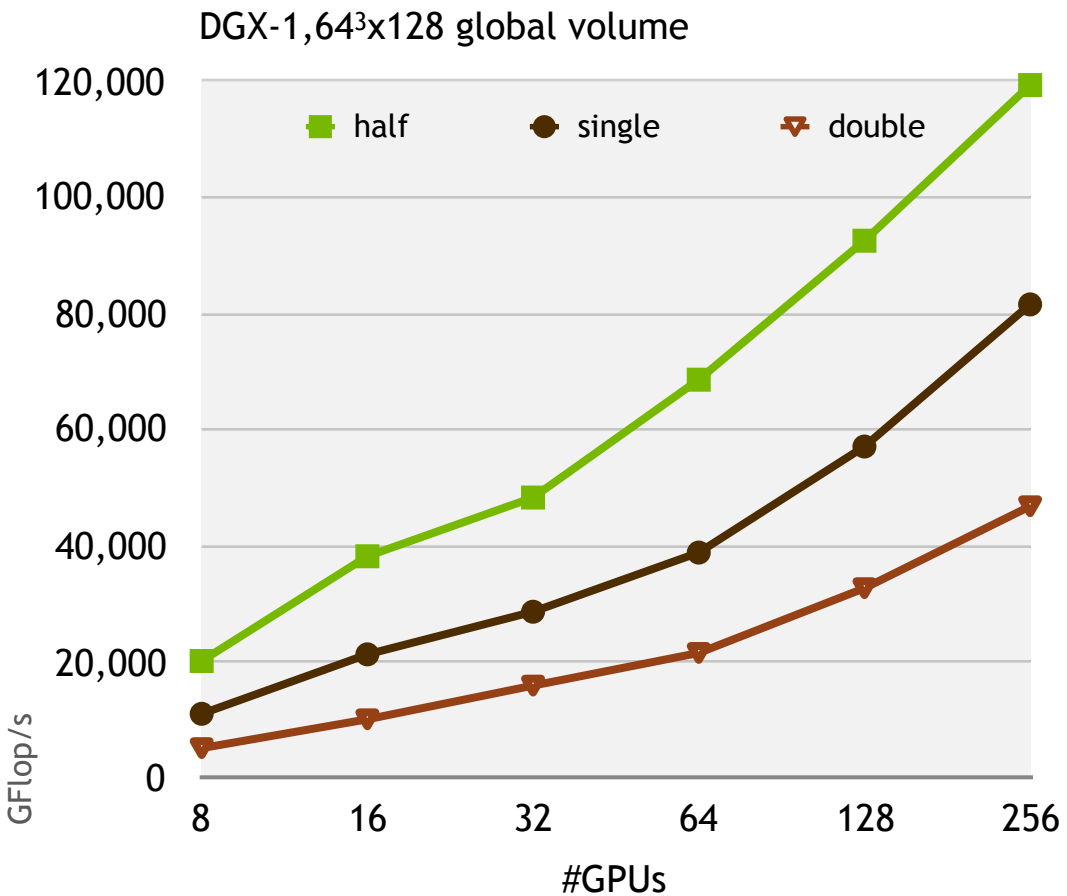


Host staging

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

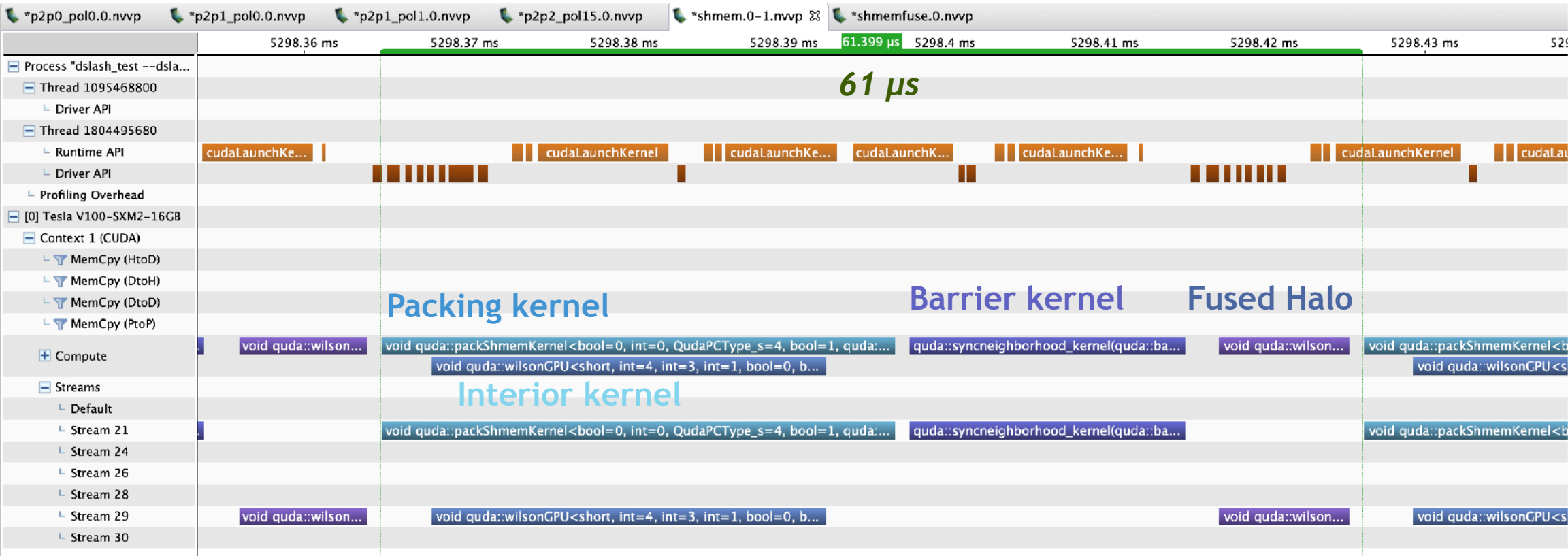
Use `nvshmem_putmem_nbi` to write to remote GPU over IB (1 RDMA transfer)

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

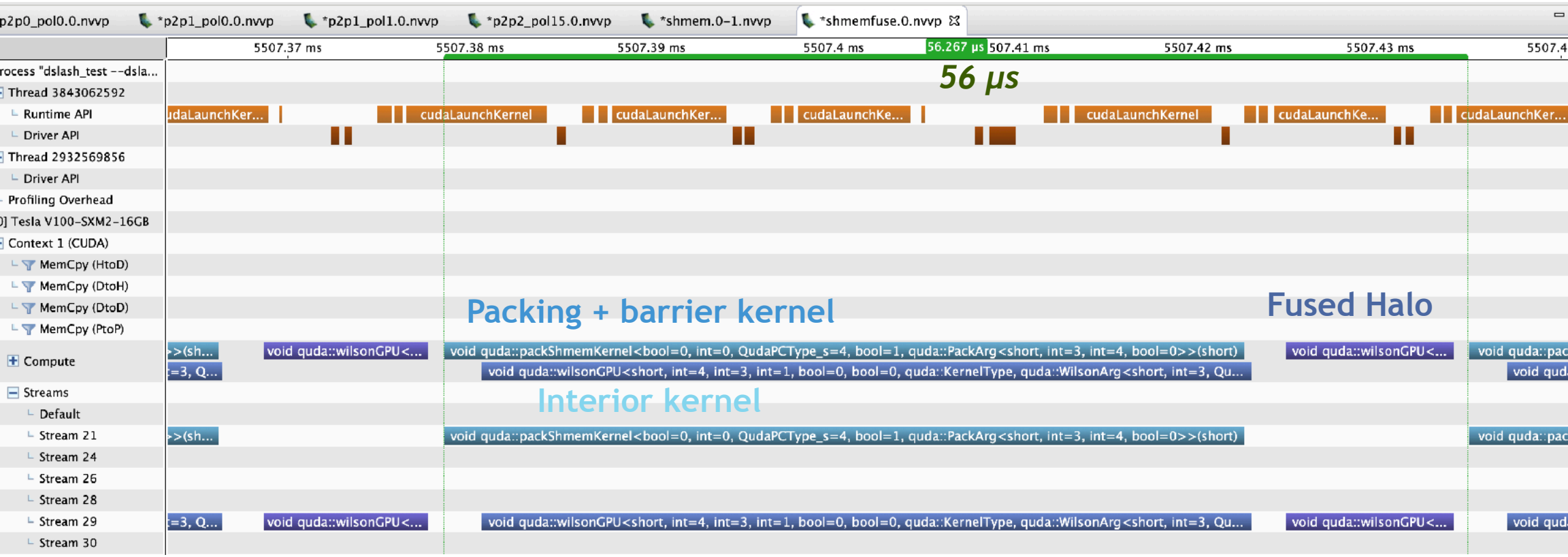
NVSHMEM DSLASH



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

NVSHMEM + FUSING KERNELS

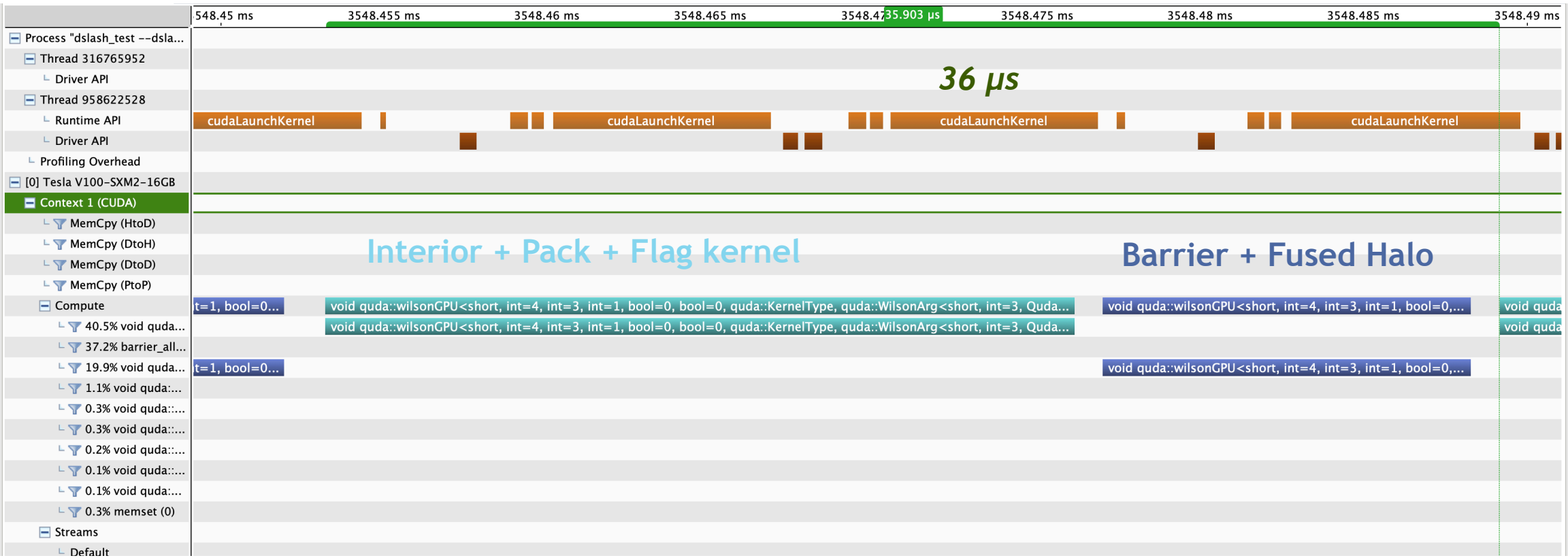
less Kernel launches



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

NVSHMEM + FUSING KERNELS II

down to two kernels

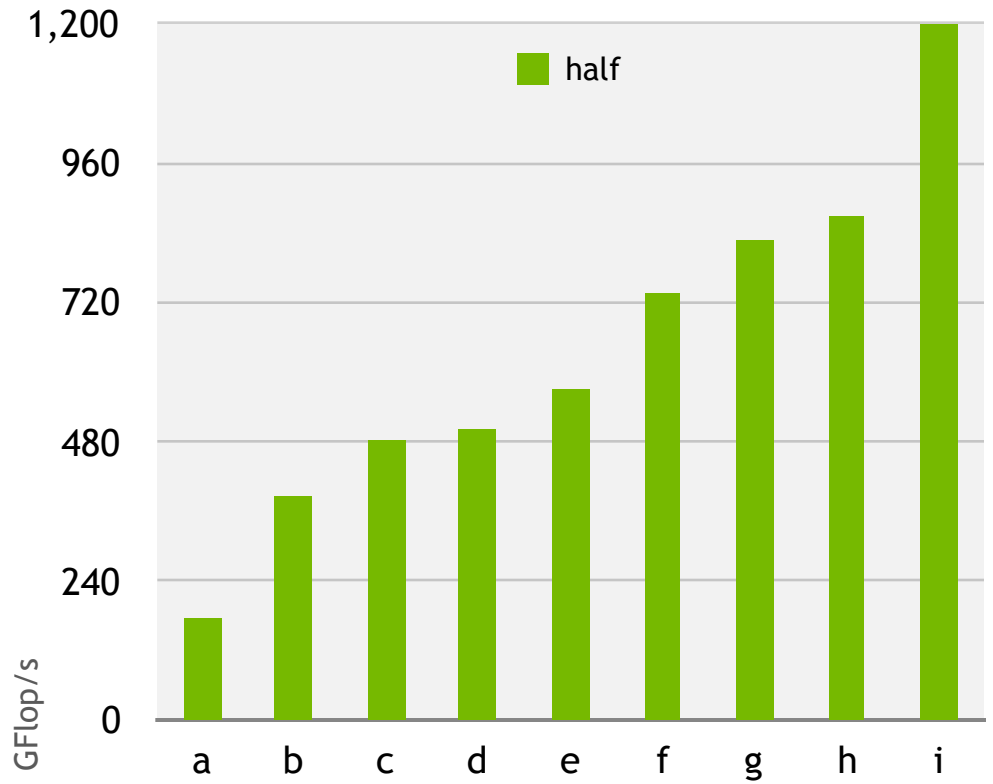


DGX-1, 16^4 local volume, half precision, 1x2x2x2 partioning

LATENCY OPTIMIZATIONS

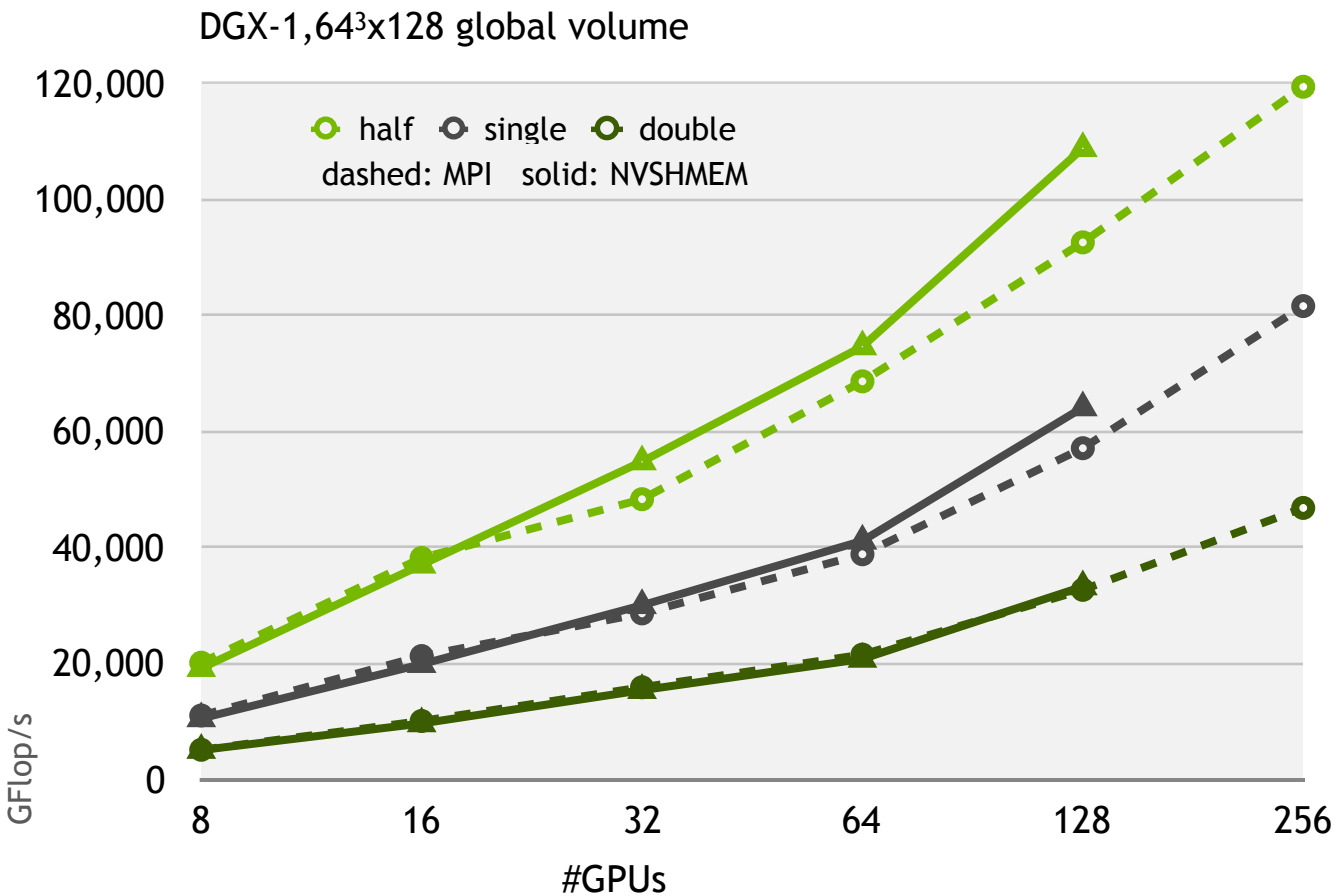
Different strategies implemented

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
h) shmem fuse packing and interior
i) shmem fuse packing+int, split barrier+ext



MULTI-NODE SCALING

with NVSHMEM



bandwidth-limited at small node count

QUDA's use of NVSHMEM not yet fully optimized

potential shows at 128 GPUs

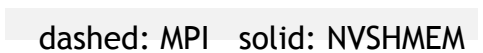
more data by Lattice conference

non-optimal topology

MULTI-NODE SCALING

with NVSHMEM

DGX-1,64^3x128 global volume



bandwidth-limited at small node count

QUDA's use of NVSHMEM not yet fully optimized

potential shows at 128 GPUs

more data by Lattice conference

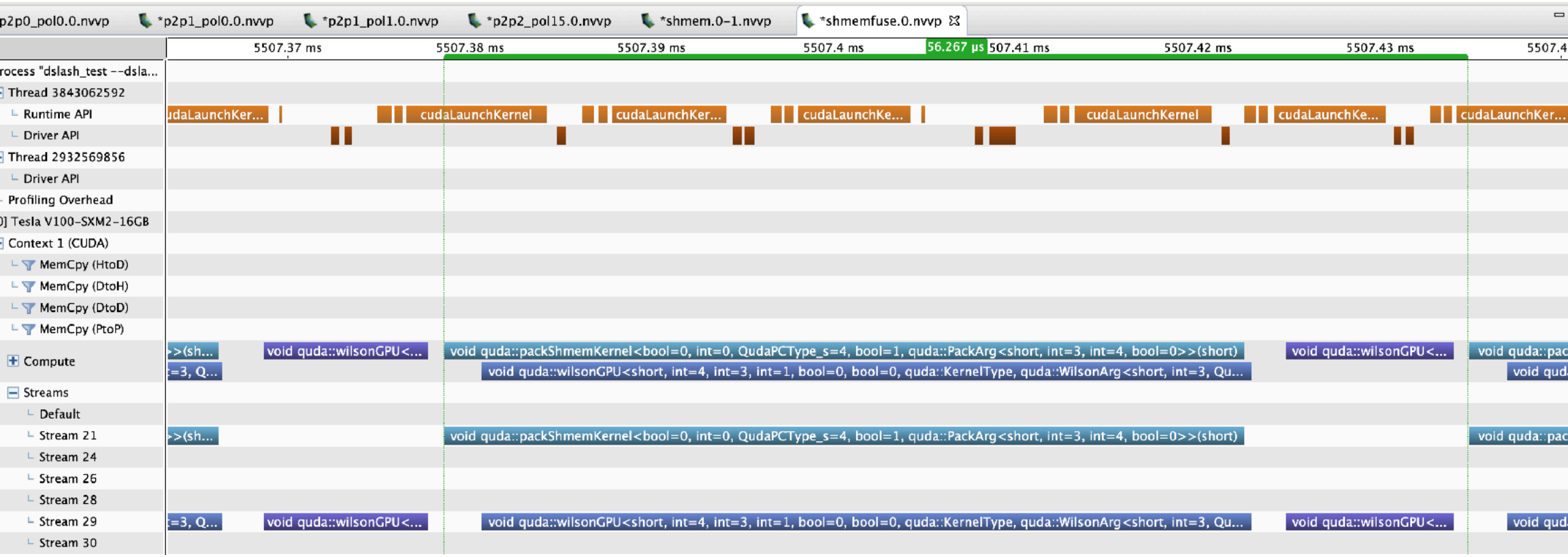
non-optimal topology

GFlop/s

#GPUs

NVSHMEM OUTLOOK

intra-kernel synchronization and communication



NVSHMEM OUTLOOK

intra-kernel synchronization and communication

SUMMARY

NVSHMEM FOR STRONG SCALING LATTICE QCD

API overheads and CPU/GPU synchronization are costly

prevent overlapping communication and compute

reduce kernel launches and API synchronization calls (fused kernels)

enabled by rewrite of QUDA kernels

GPU centric communication with NVSHMEM takes CPU limitations out

Prototype implementation already shows nice speed up for Wilson

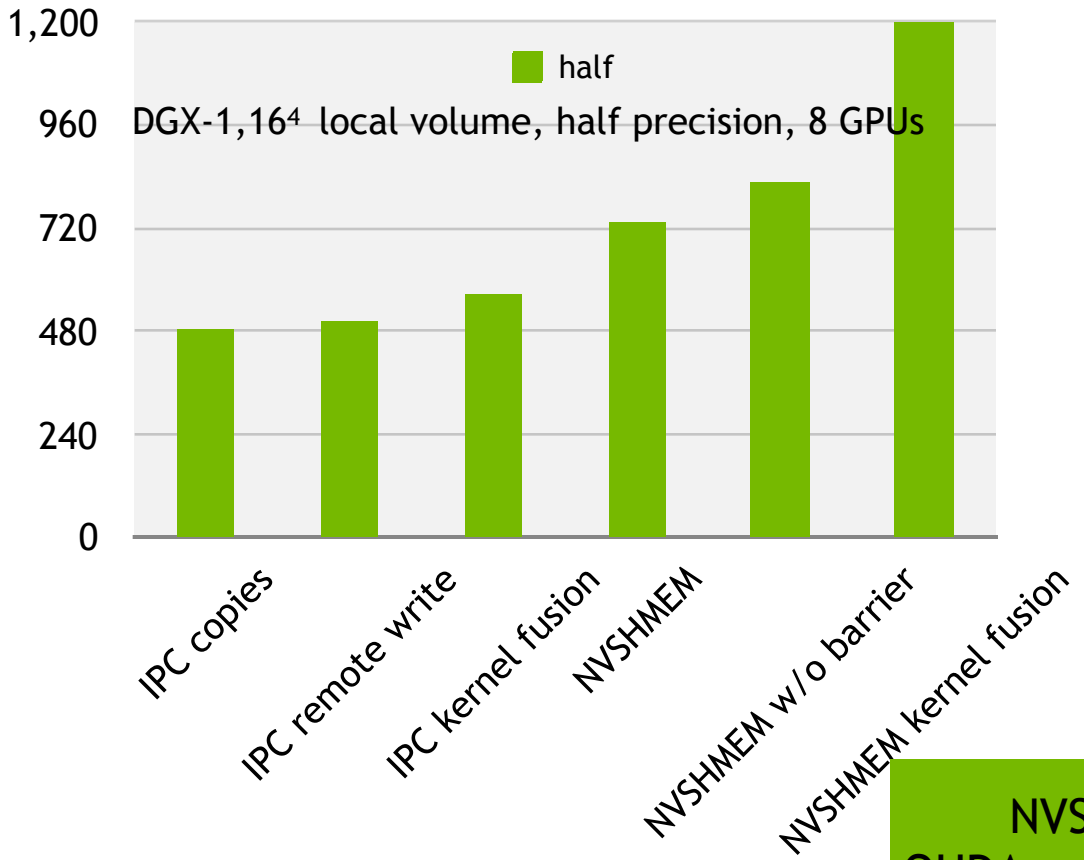
Will be extended to all Dslashes when experimentation phase cools down

will be awesome for latency limited Multigrid

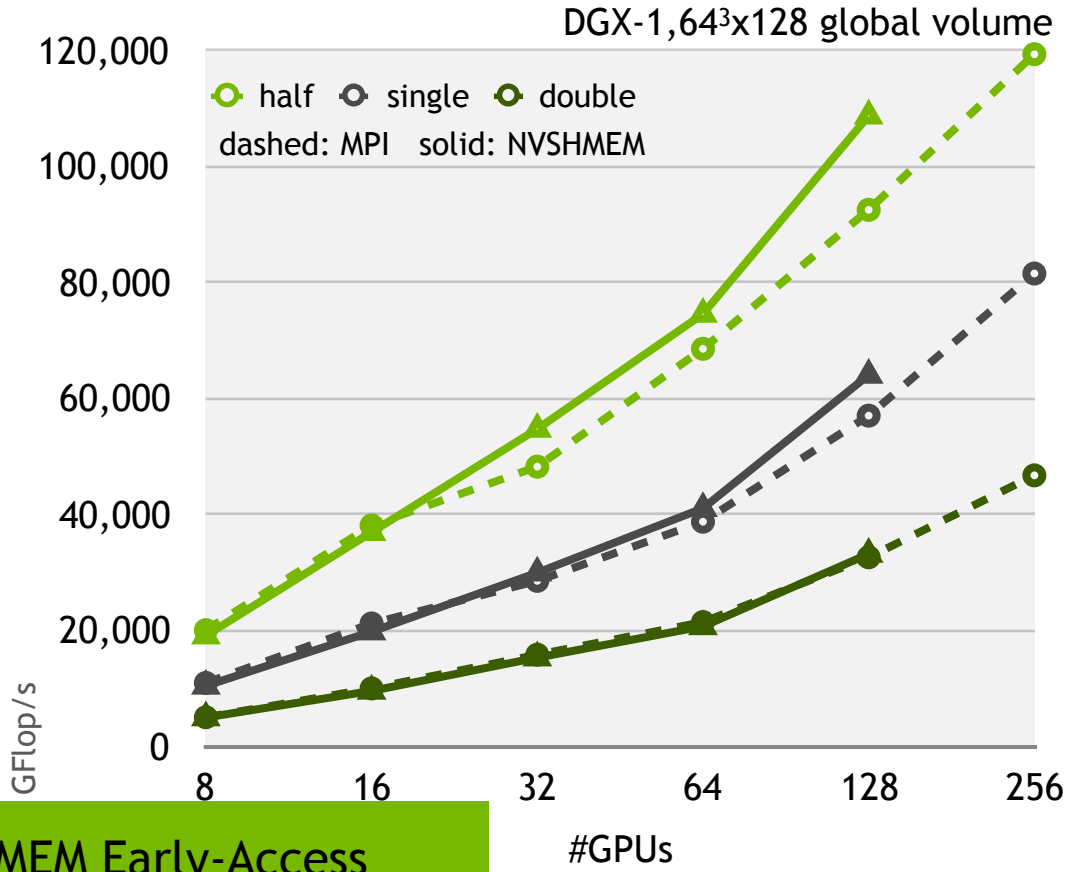
Works on x86 and Power

NVSHMEM FOR STRONG SCALING LATTICE QCD

single node



multi-node



NVSHMEM Early-Access
QUDA prototype implementation


NVIDIA