Avoiding the Jam

Mathias Wagner | Lattice 2022

http://lattice.github.com/quda

SciDAC
Scientific Discovery through Advanced Computing

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 (Kate Clark, Thursday)
  - 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
  - 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 ?*

STRONG SCALING

Reduce time to solution

Same problem size

more nodes (GPUs)

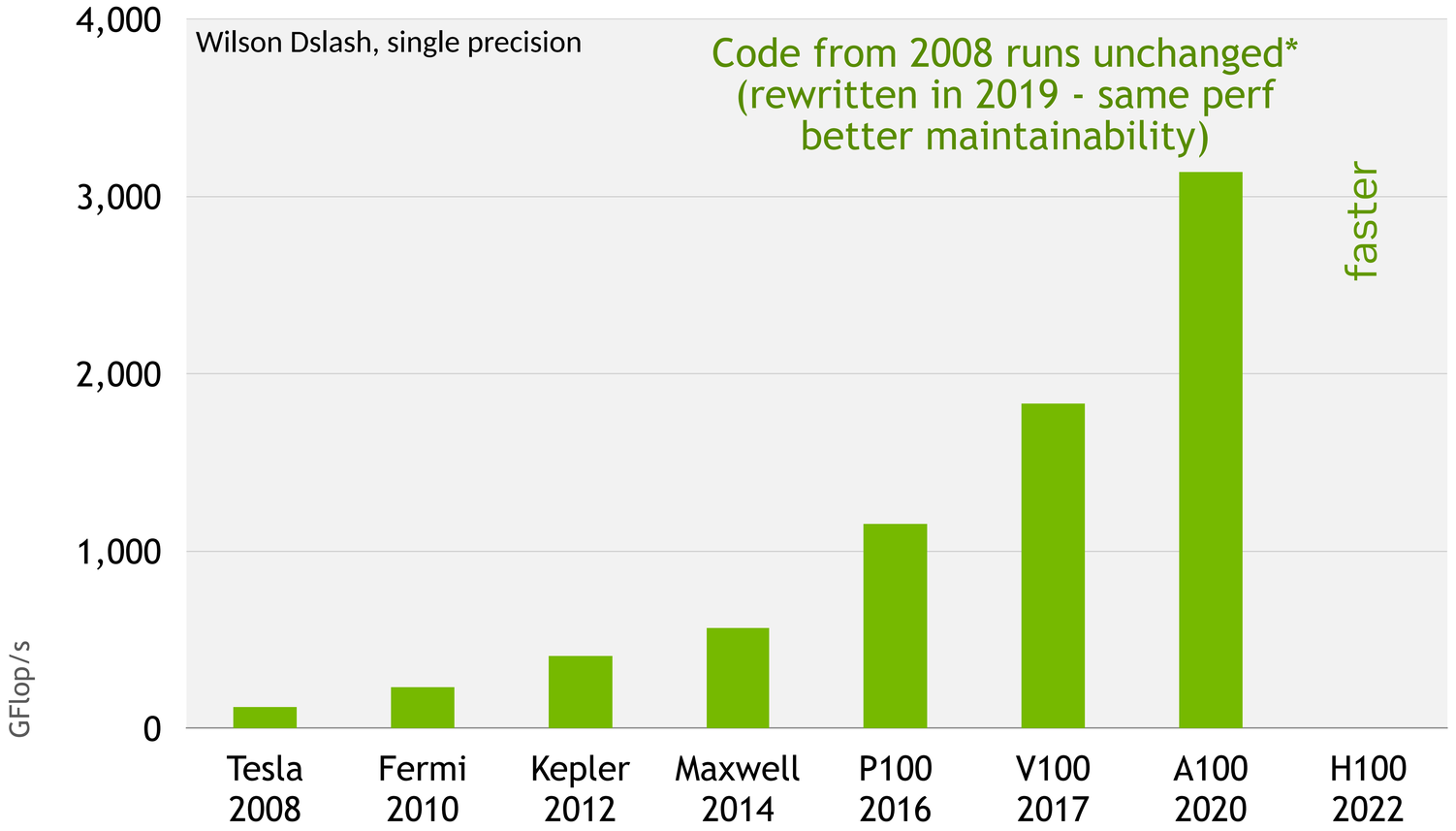
faster GPUs (next generation)

less bits (lower/mixed precision)

More bandwidth, but what about latency

SINGLE GPU PERFORMANCE

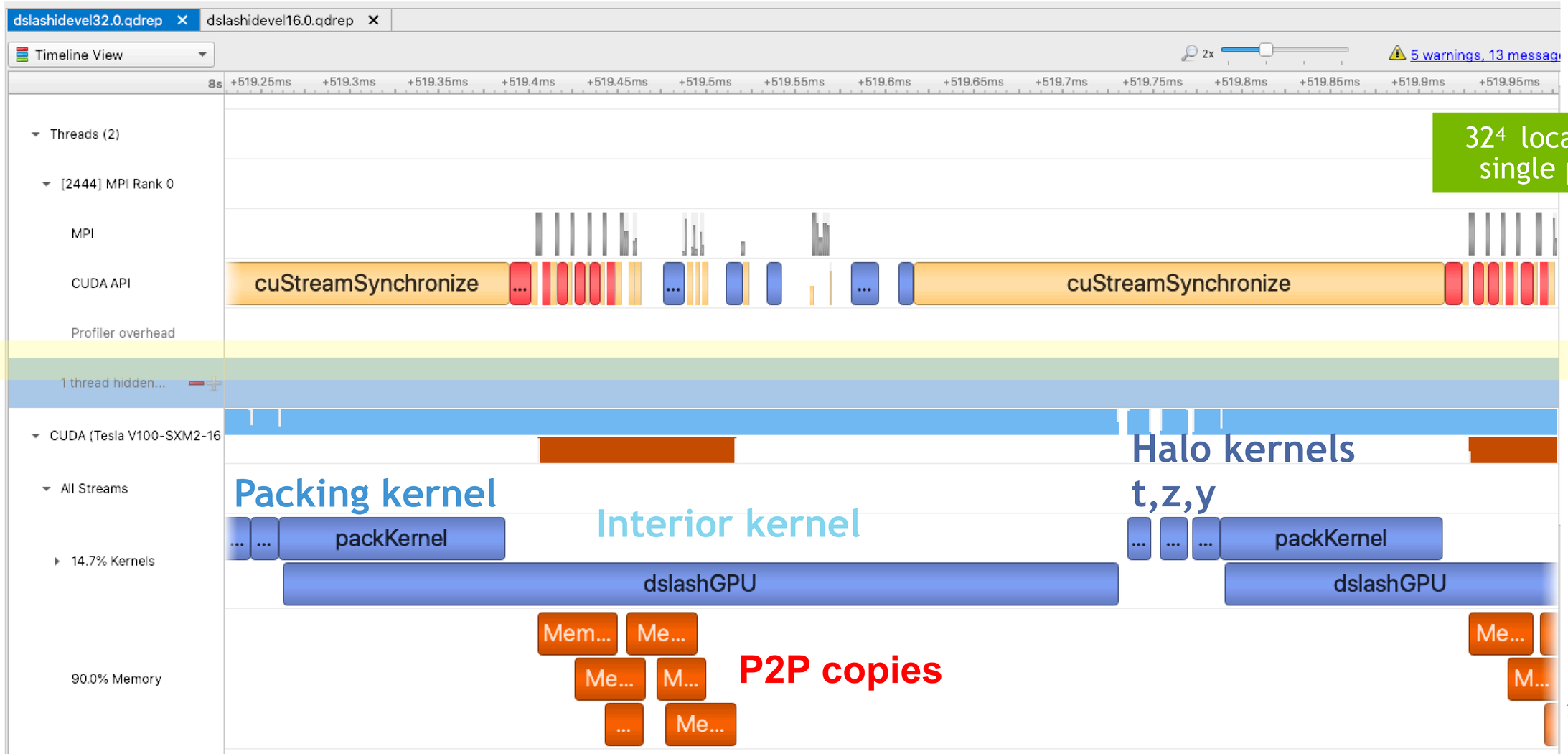
Wilson Dslash Kernel



GPU-CENTRIC COMMUNICATION

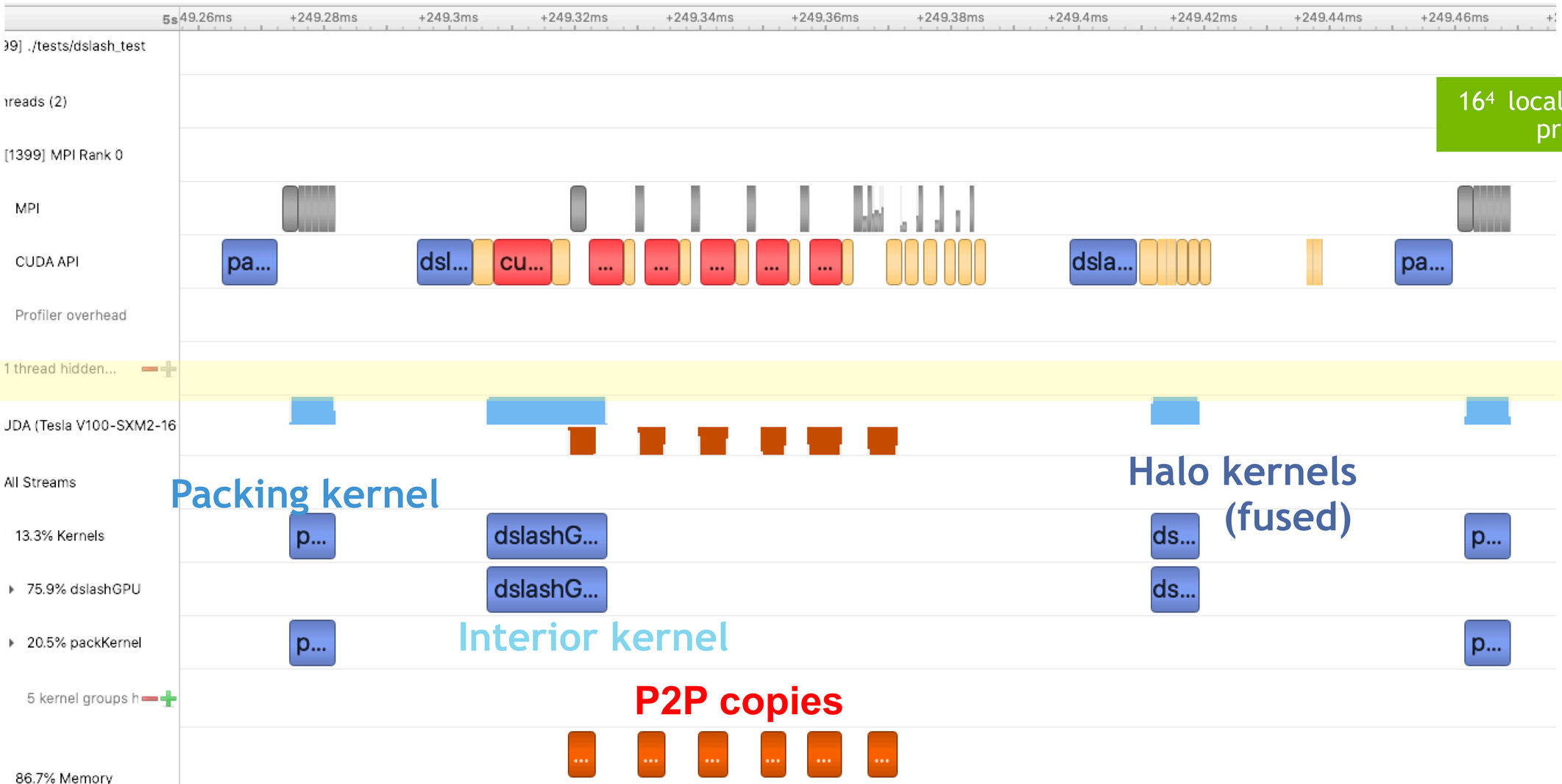
MULTI-GPU PROFILE

overlapping comms and compute



STRONG SCALING PROFILE

~~overlapping~~ comms and compute



16^4 local volume, half precision

DGX-1, 1x2x2x2 partitioning

NVIDIA

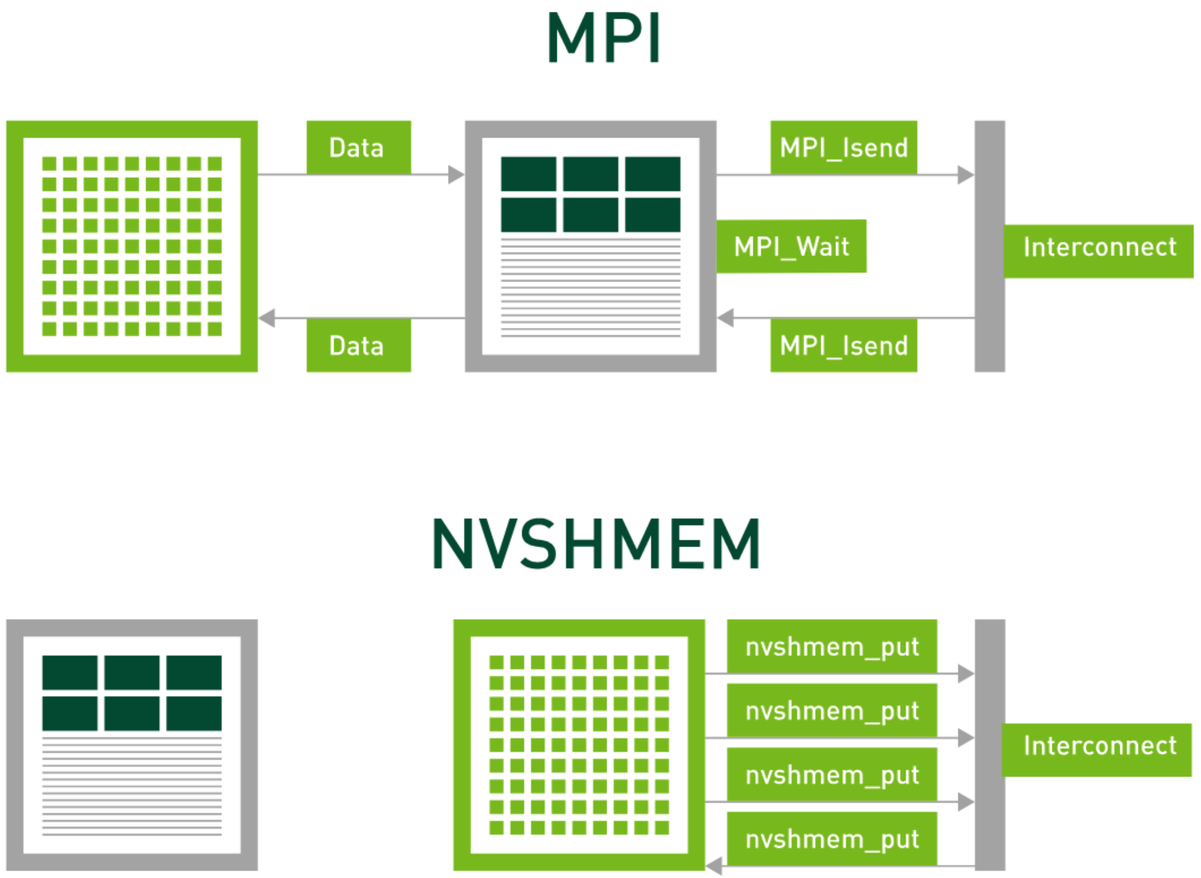
NVSHMEM: OpenSHMEM FOR CLUSTERS OF NVIDIA GPUS

- 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



FINE-GRAINED SYNCHRONIZATION

libcu++ gives us std::atomic in CUDA

Need replacement for kernel boundaries

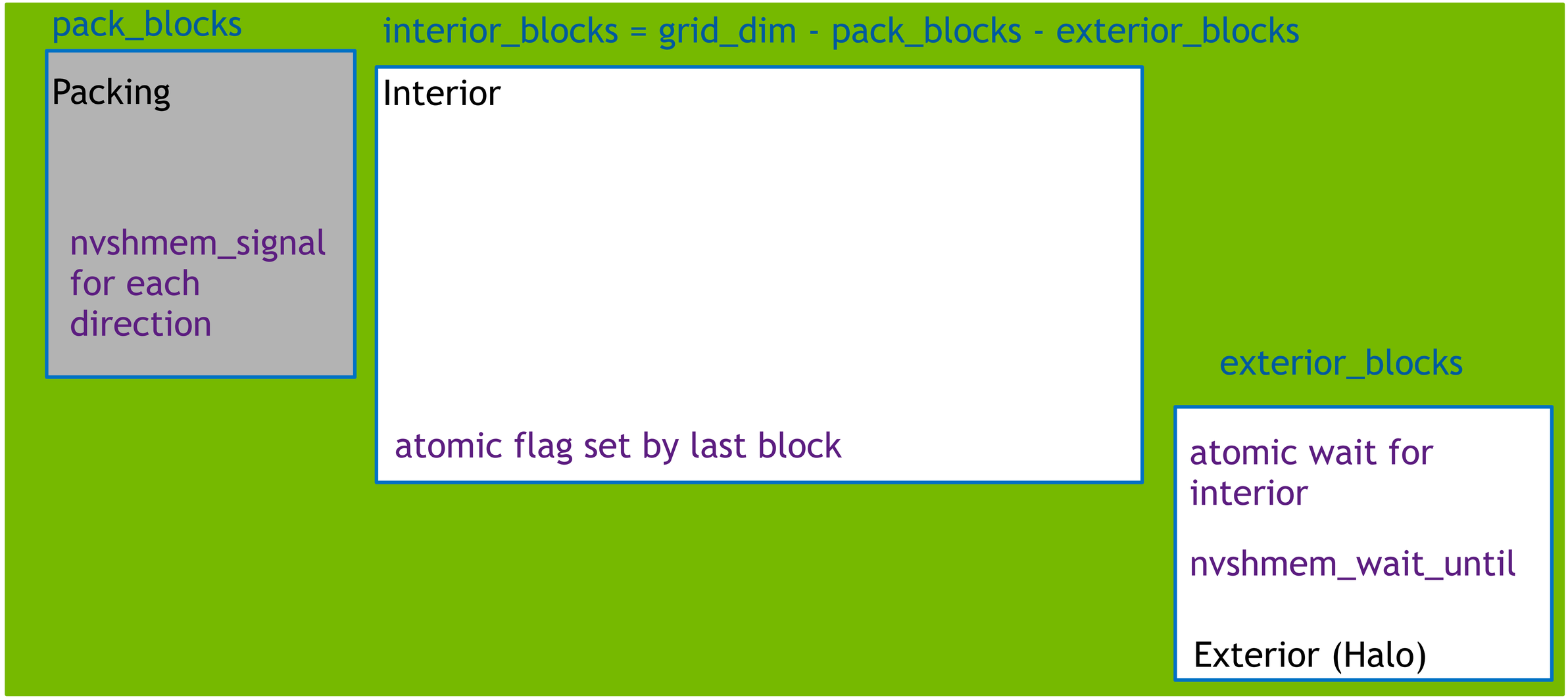
Packing is independent of interior

interior and exterior update on boundary
→ possible race condition

use cuda::atomic from libcu++

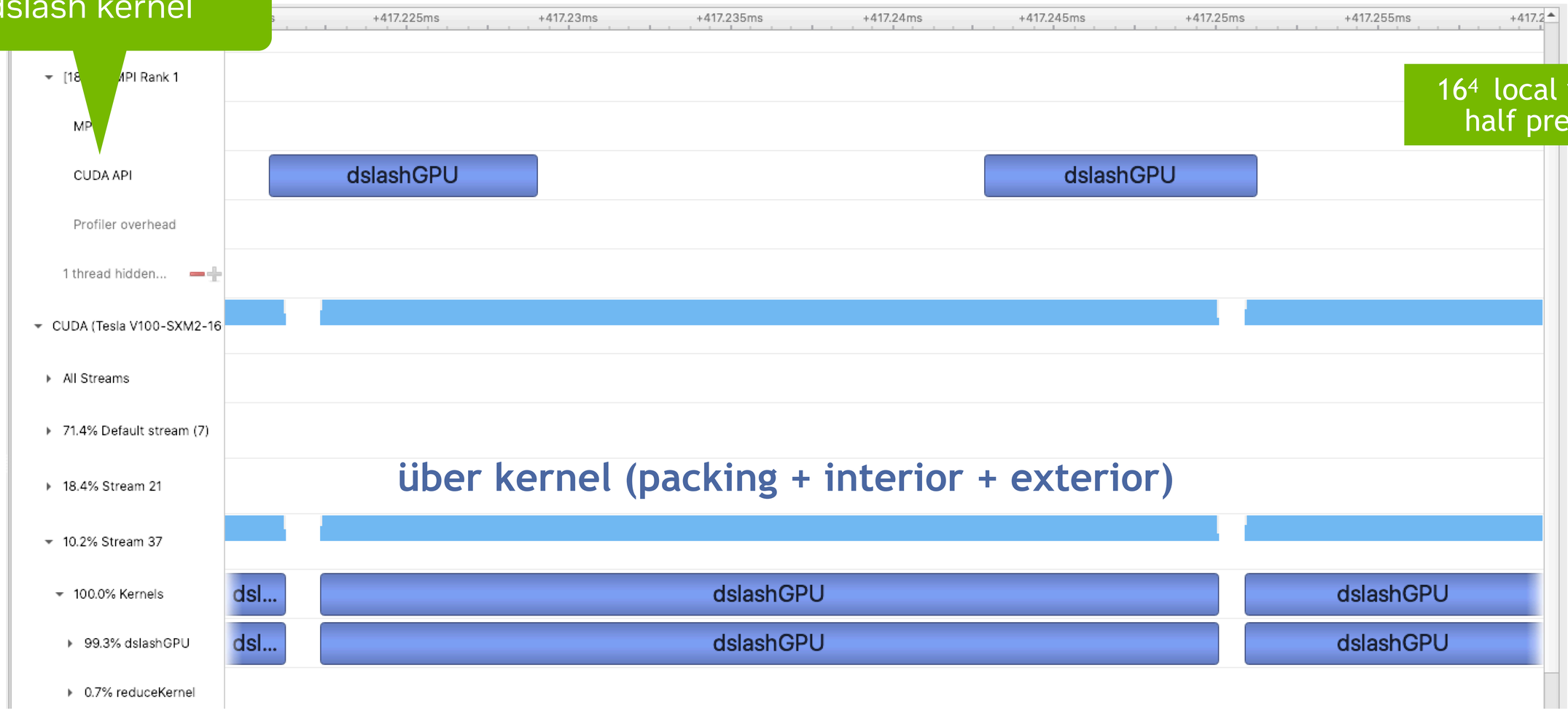
```
#include <atomic>
std::atomic<int> x;

#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

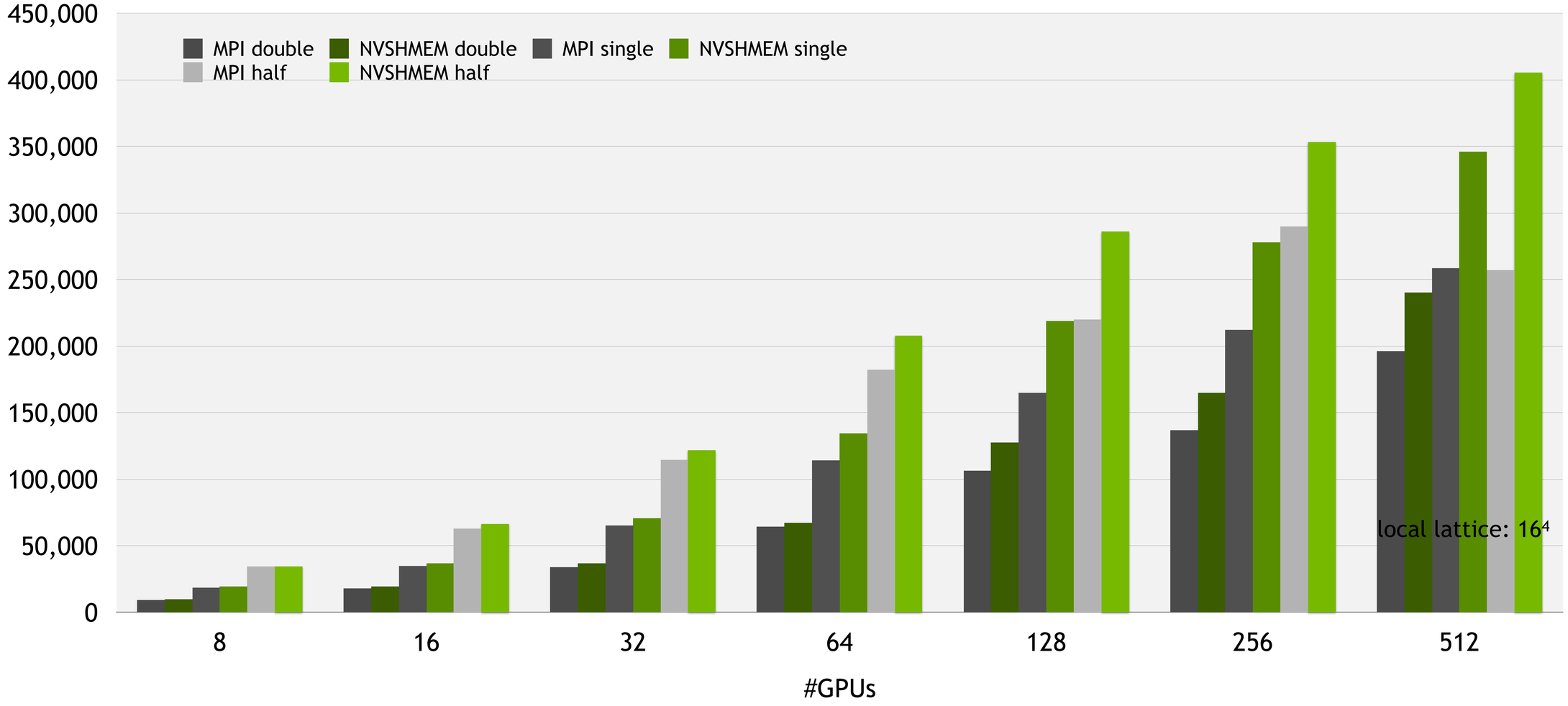


16^4 local volume, half precision

DGX-1, 1x2x2x2 partitioning

SELENE STRONG SCALING

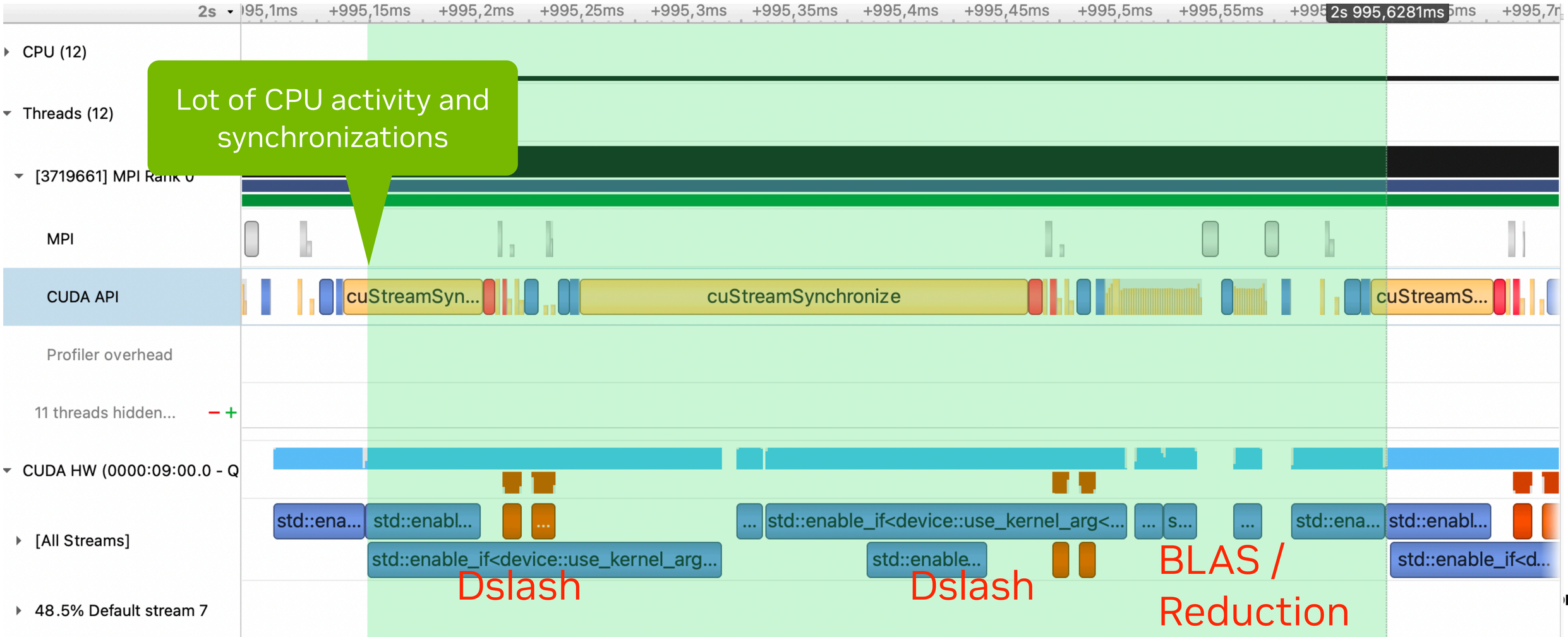
Global Volume $64^3 x128$, Wilson-Dslash



INVERTER SCALING

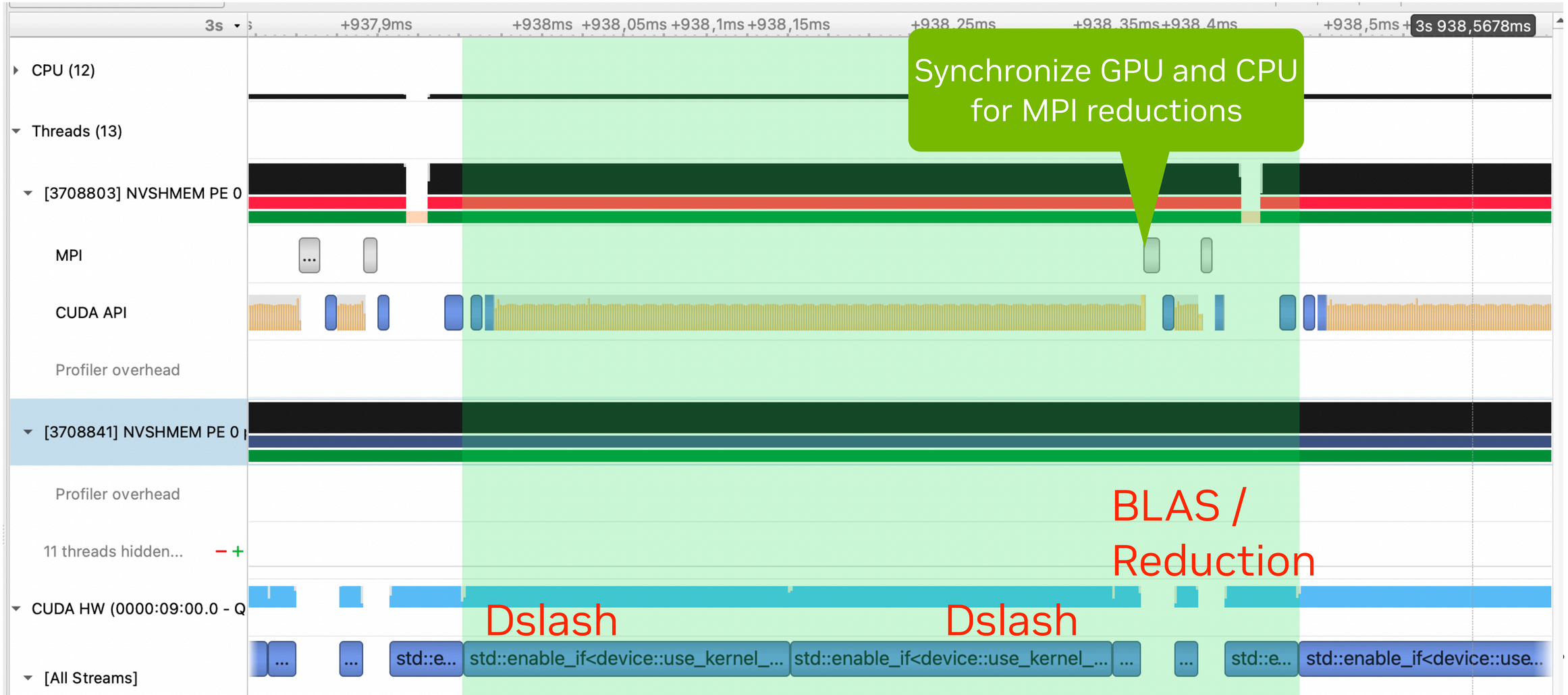
CONJUGATE GRADIENT

MPI + IPC



CONJUGATE GRADIENT

NVSHMEM



REDUCTIONS

Host-Device Synchronization

Need to synchronize the device and host when doing a reduction

Traditional QUDA method

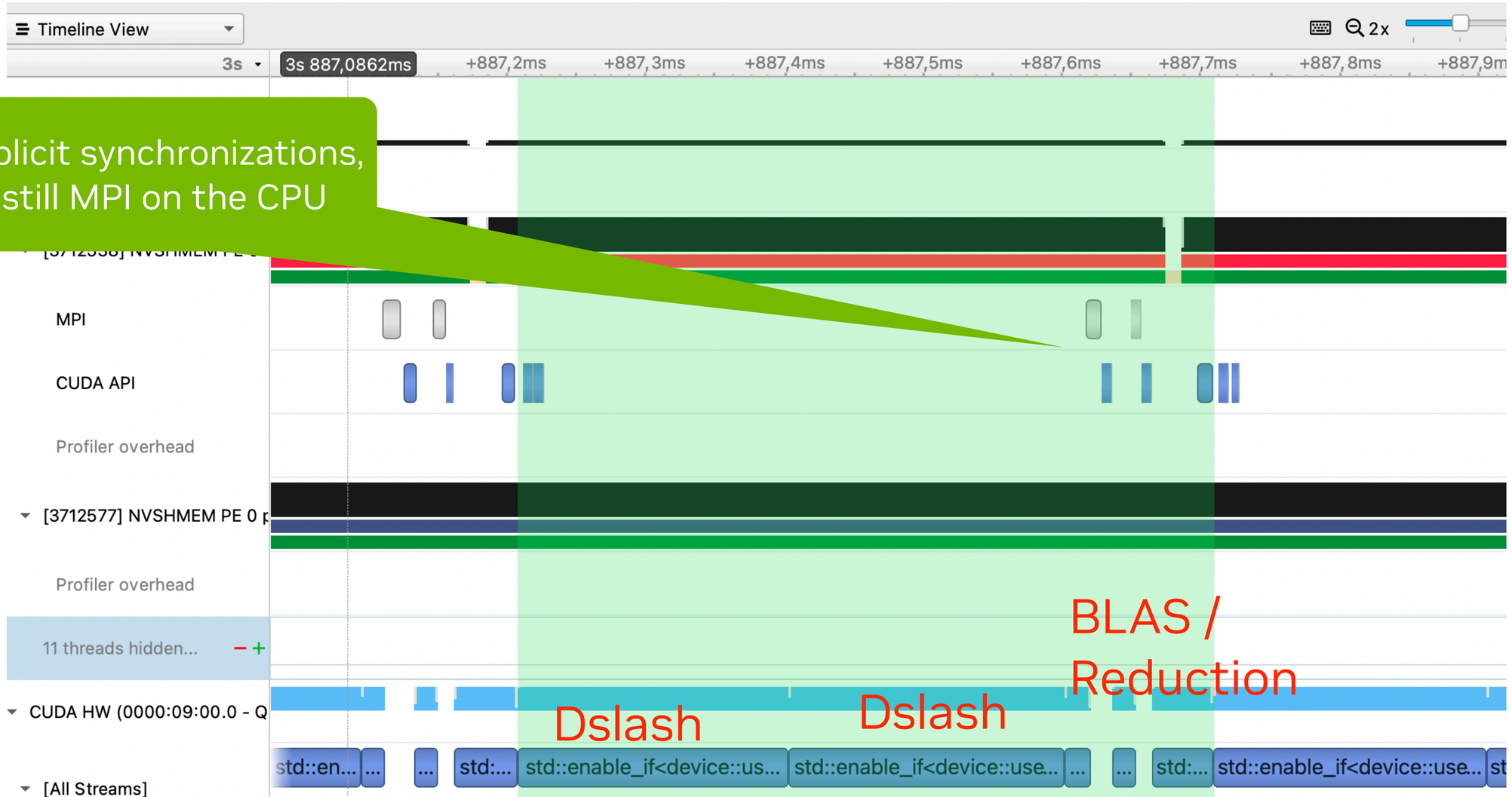
- Kernel does per-device reduction writing result to sysmem
- Synchronize host and device

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



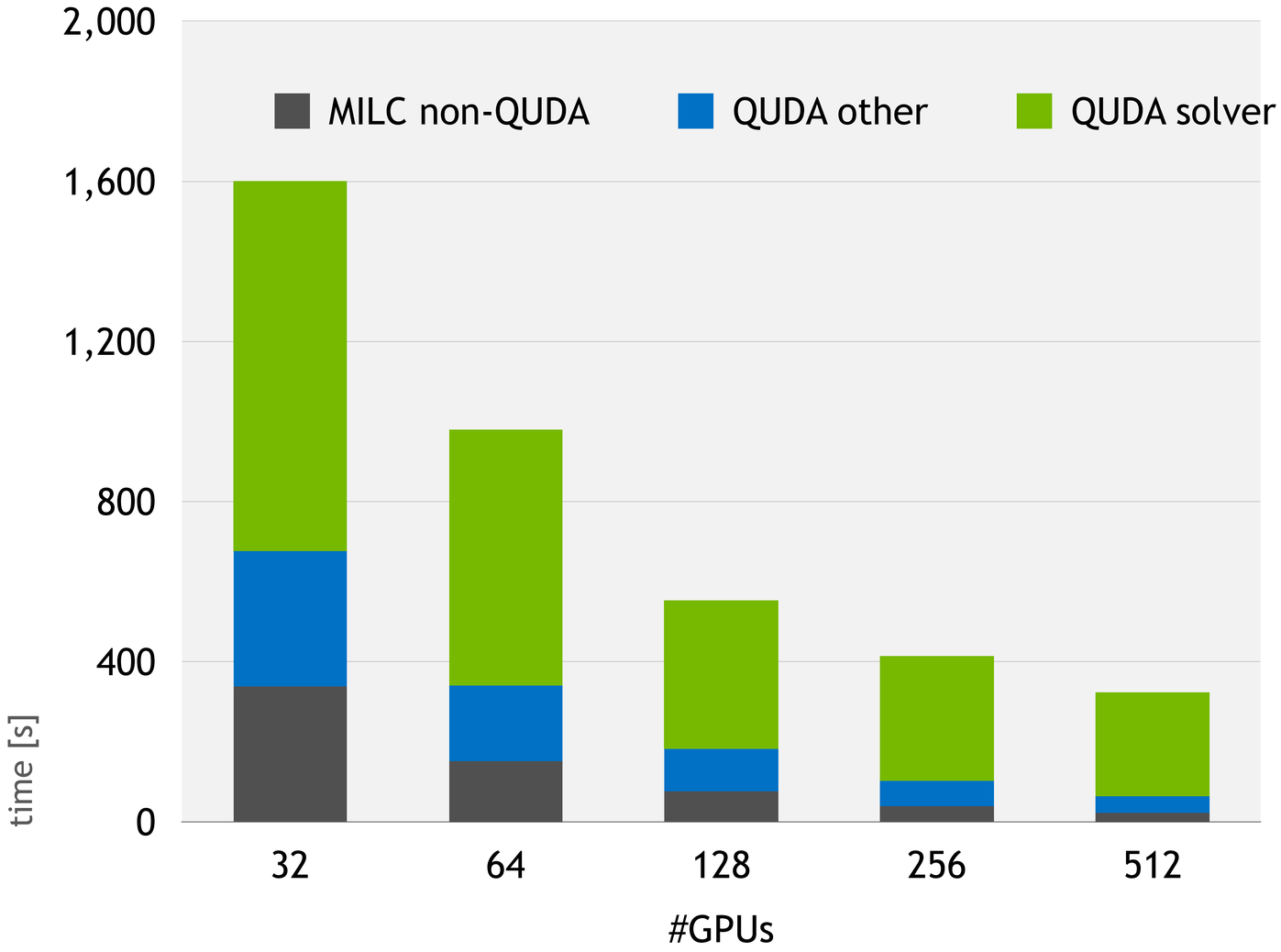
RHMC SCALING

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 $72^3 \times 144$



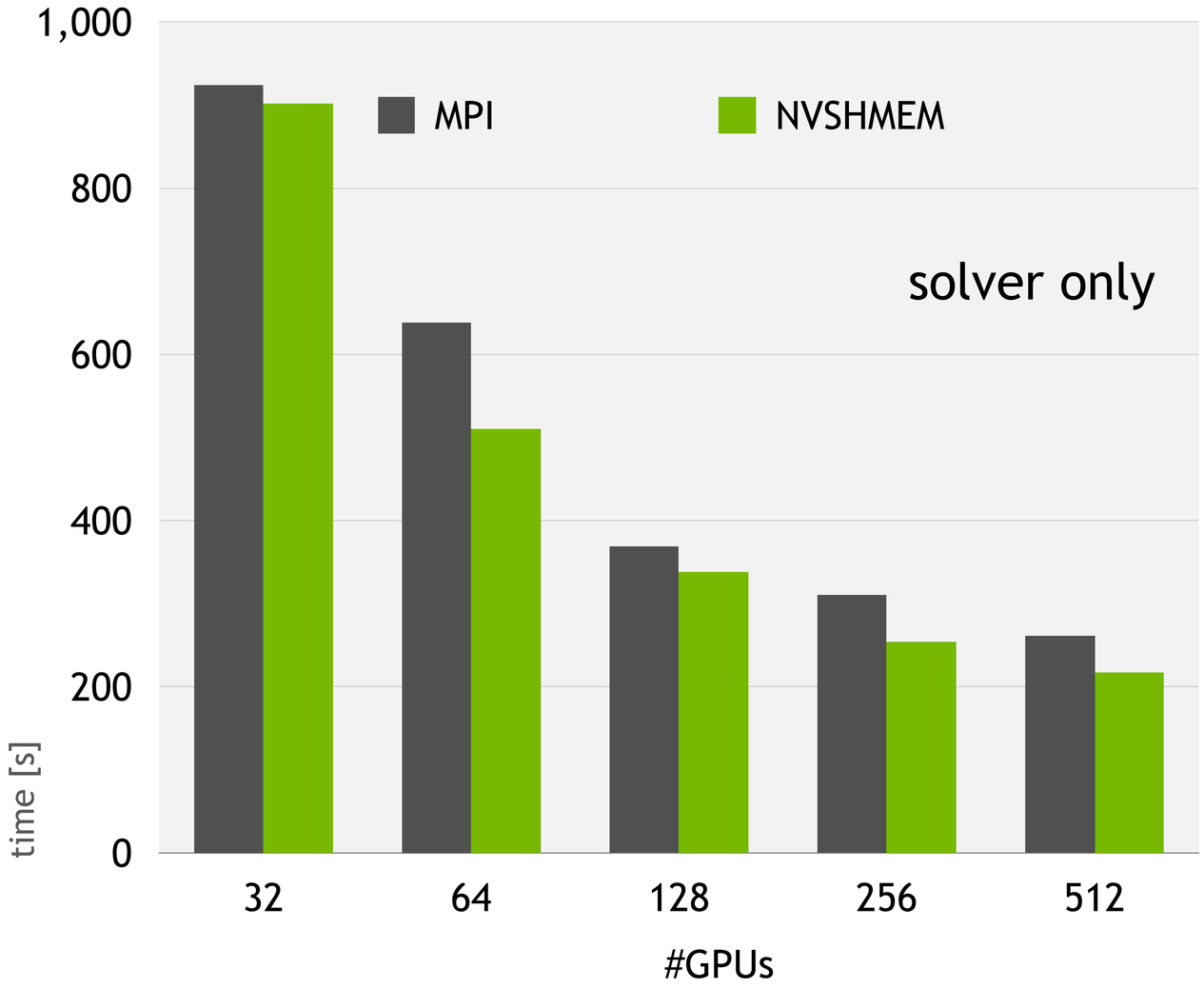
Running with MPI

other part scales reasonably
(not limited by communication)

solver part needs improvements

MILC SOLVER SCALING ON SELENE

NERSC LARGE BENCHMARK $72^3 \times 144$



mixed precision methods:

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

QUDA solver in MILC

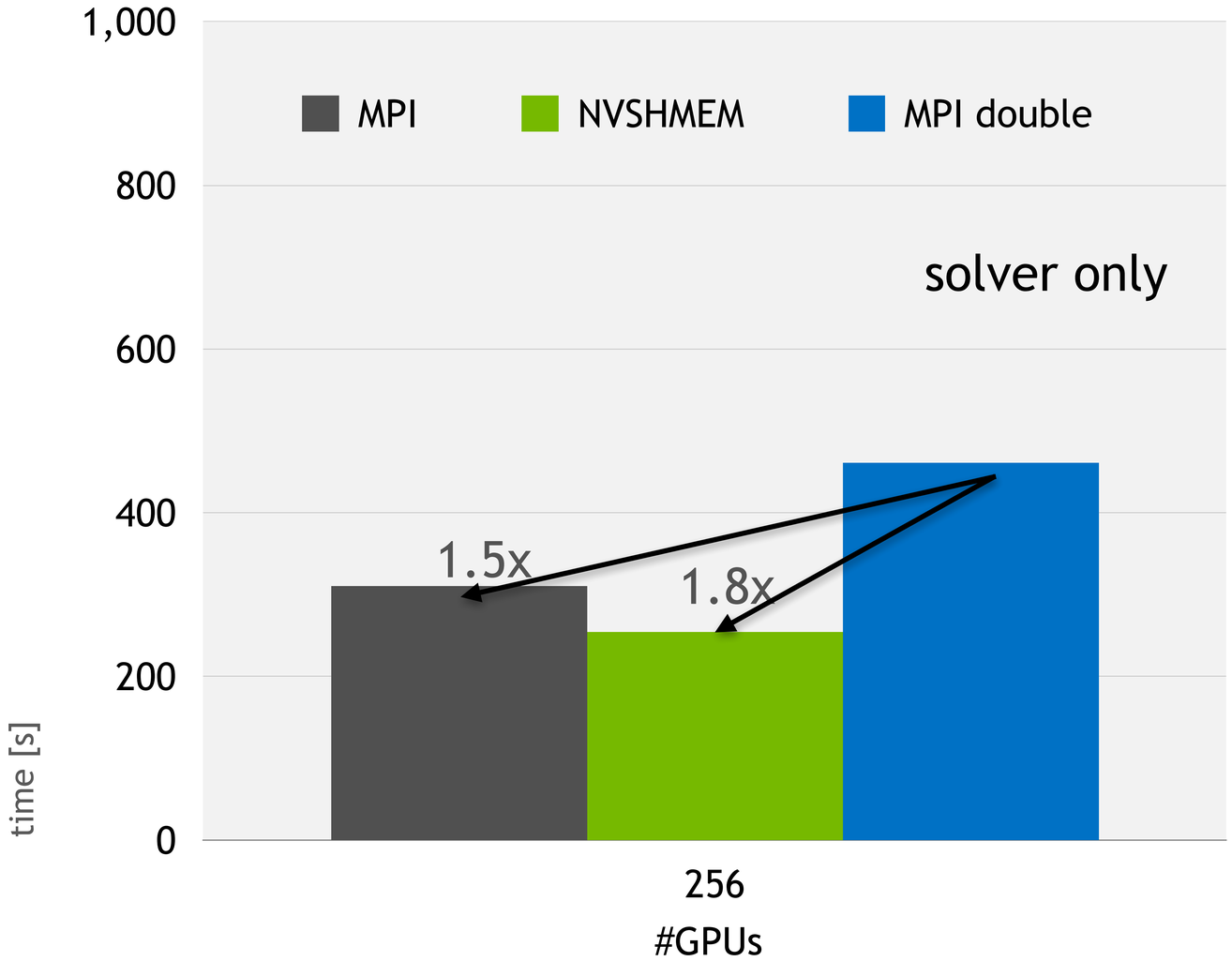
- mixed precision multishift: double-single
- mixed precision refinement: double-half

sweet spot at 256 GPUs:

- ~20% less time in solver

MILC SOLVER SCALING ON SELENE

NERSC LARGE BENCHMARK 72^3x144



mixed precision methods:

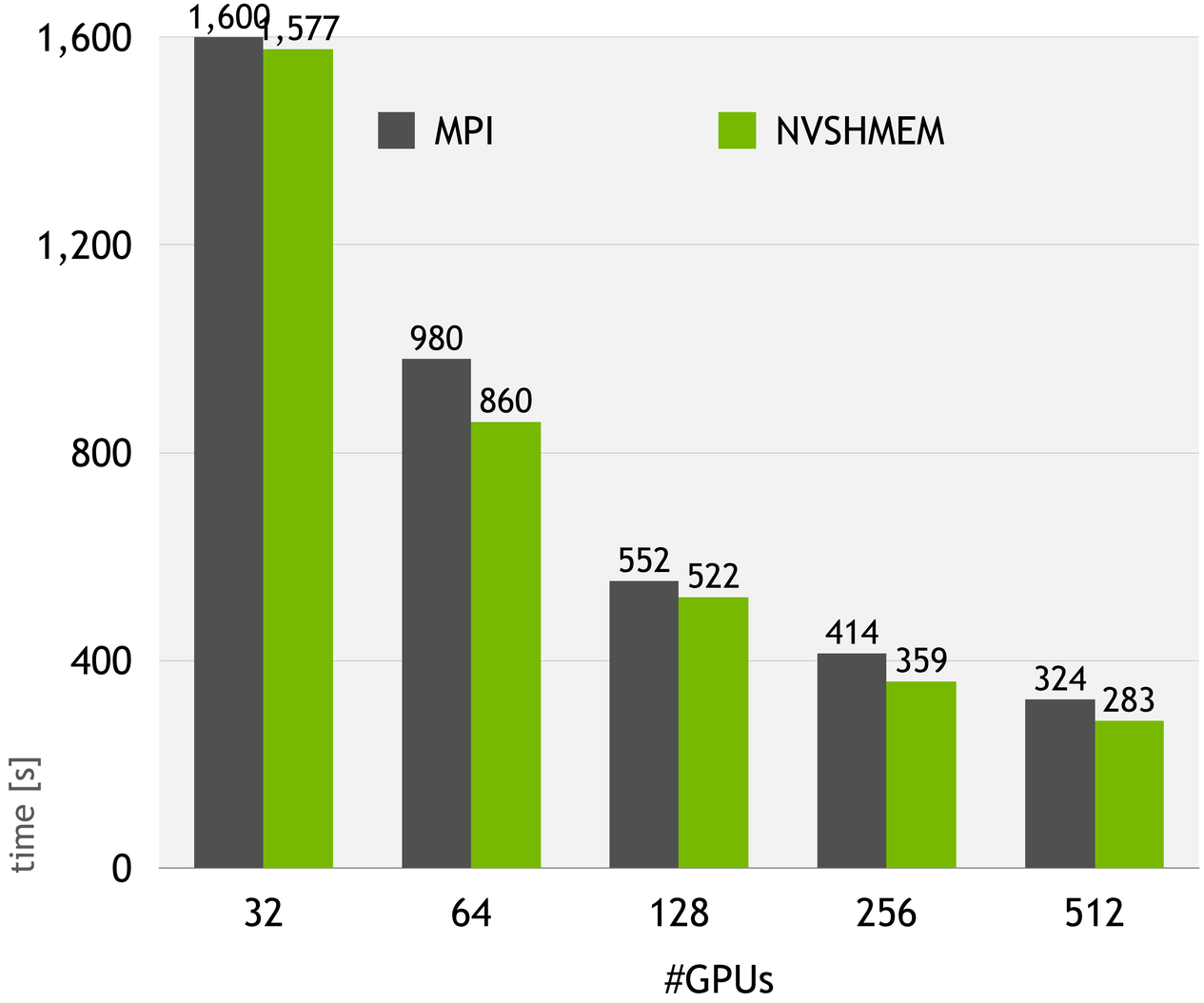
- lower precisions harder to scale
- NVSHMEM crucial for mixed precision

at 256 GPUs:

- NVSHMEM recovers expect almost 2x benefit of mixed precision

MILC SOLVER SCALING ON SELENE

NERSC LARGE BENCHMARK 72^3x144

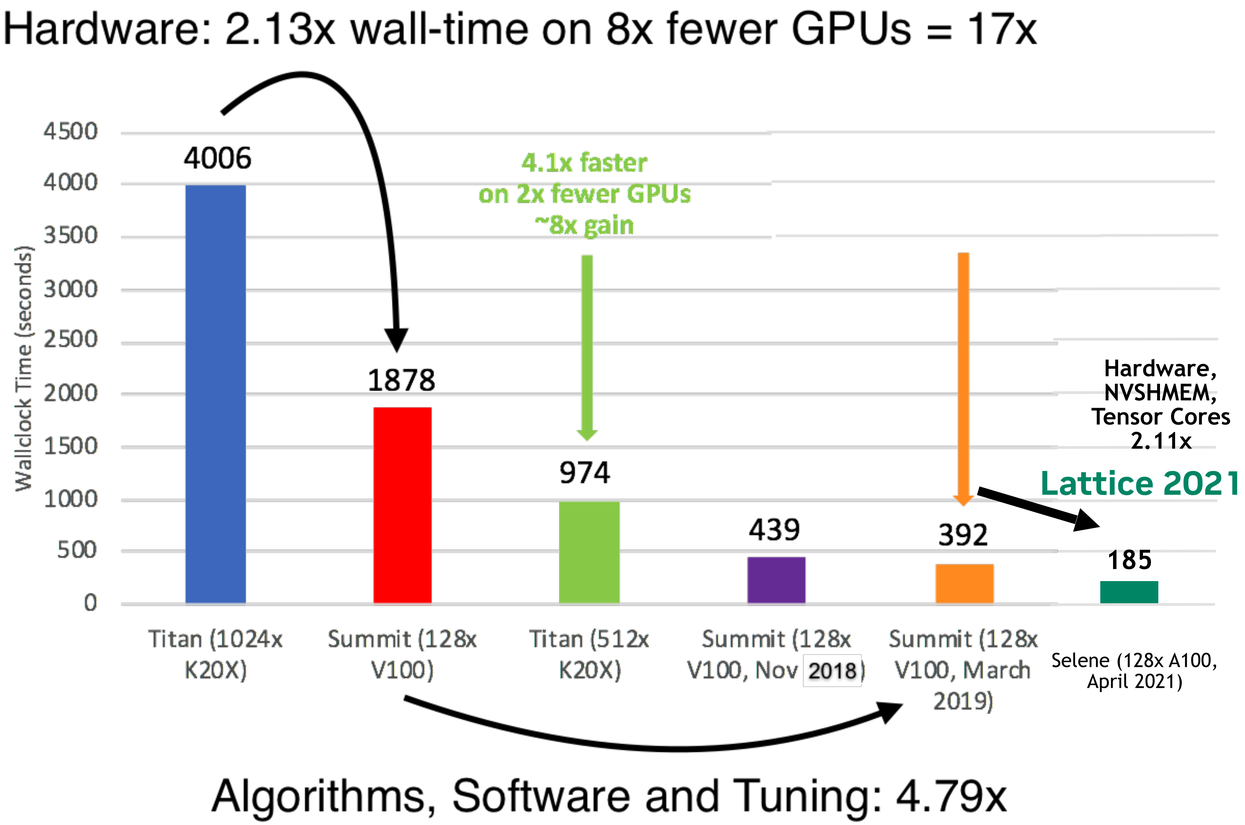


Full benchmark

>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

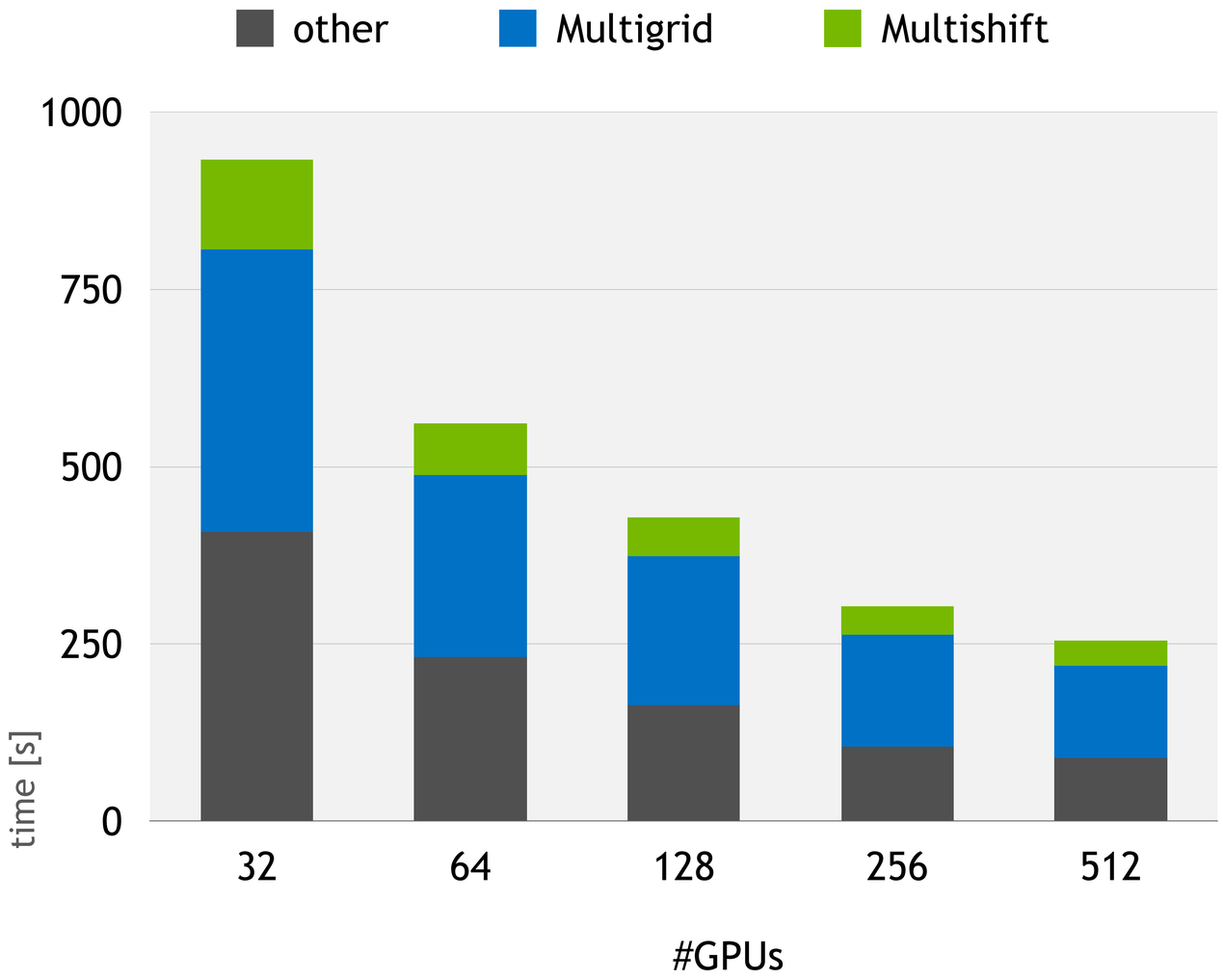


Chroma w/ QDP-JIT and QUDA, ECP FOM data, $V=64^3x128$ sites, m_π ~172 MeV, (QDP-JIT by F. Winter, Jefferson Lab)

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)

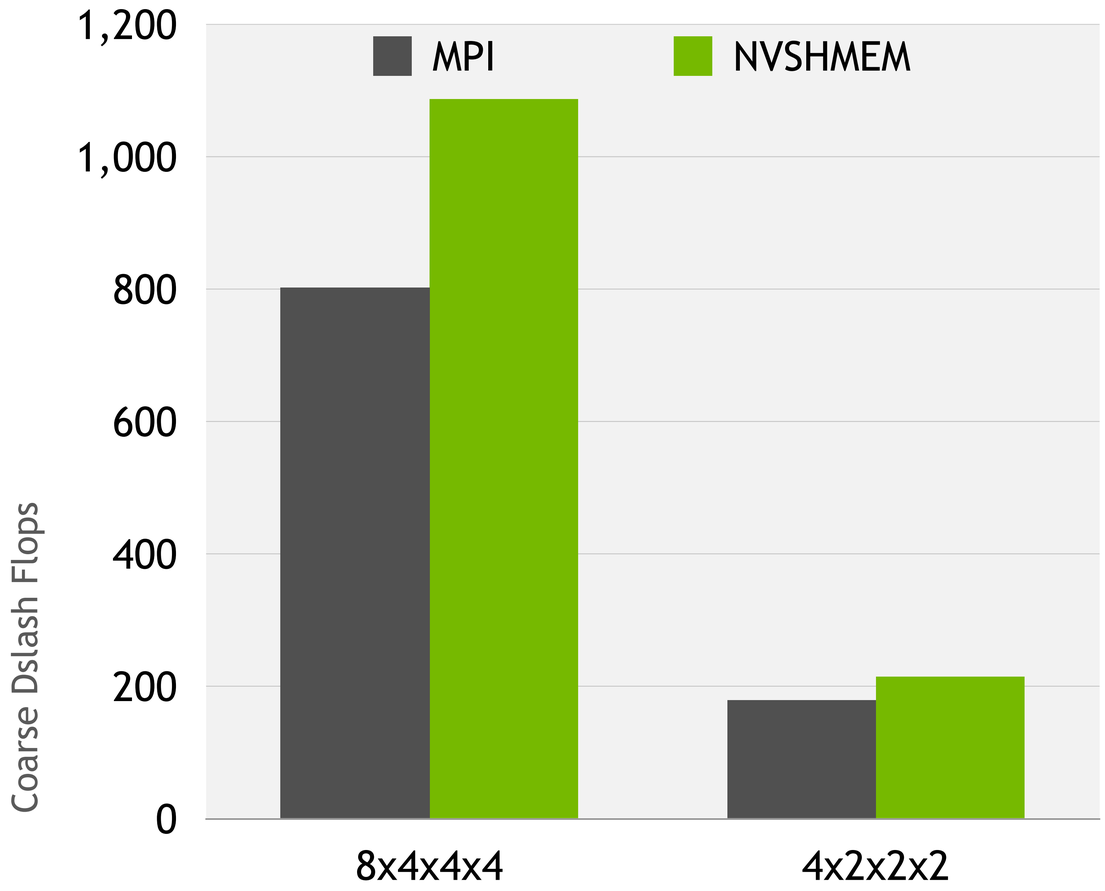
QUDA multishift (MPI/QMP)

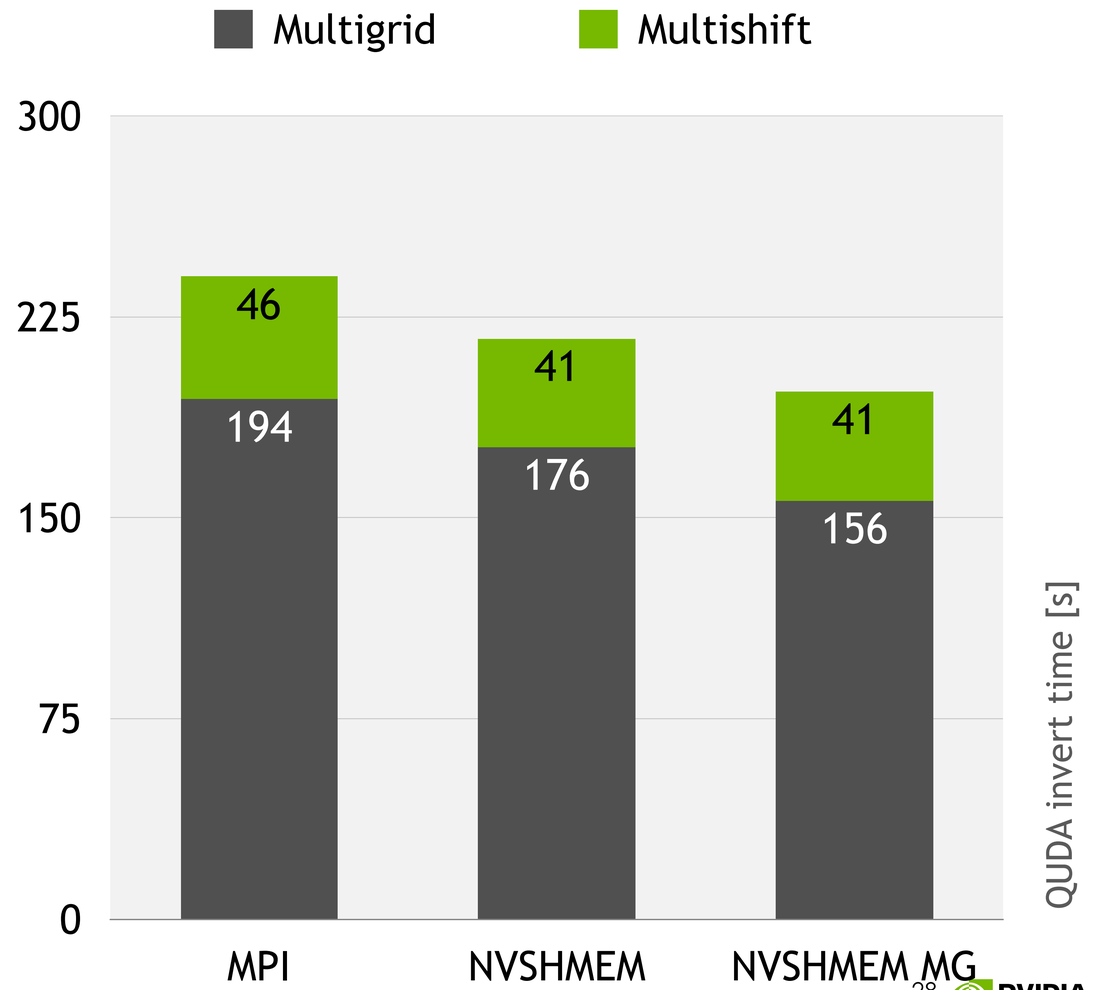
Chroma dominated by Multigrid solves

NVSHMEM FOR MULTIGRID (COARSE DSLASH)

CHROMA WILSON-CLOVER HMC SCALING

Improvements 2021 (MPI) to 2022 (NVSHMEM)

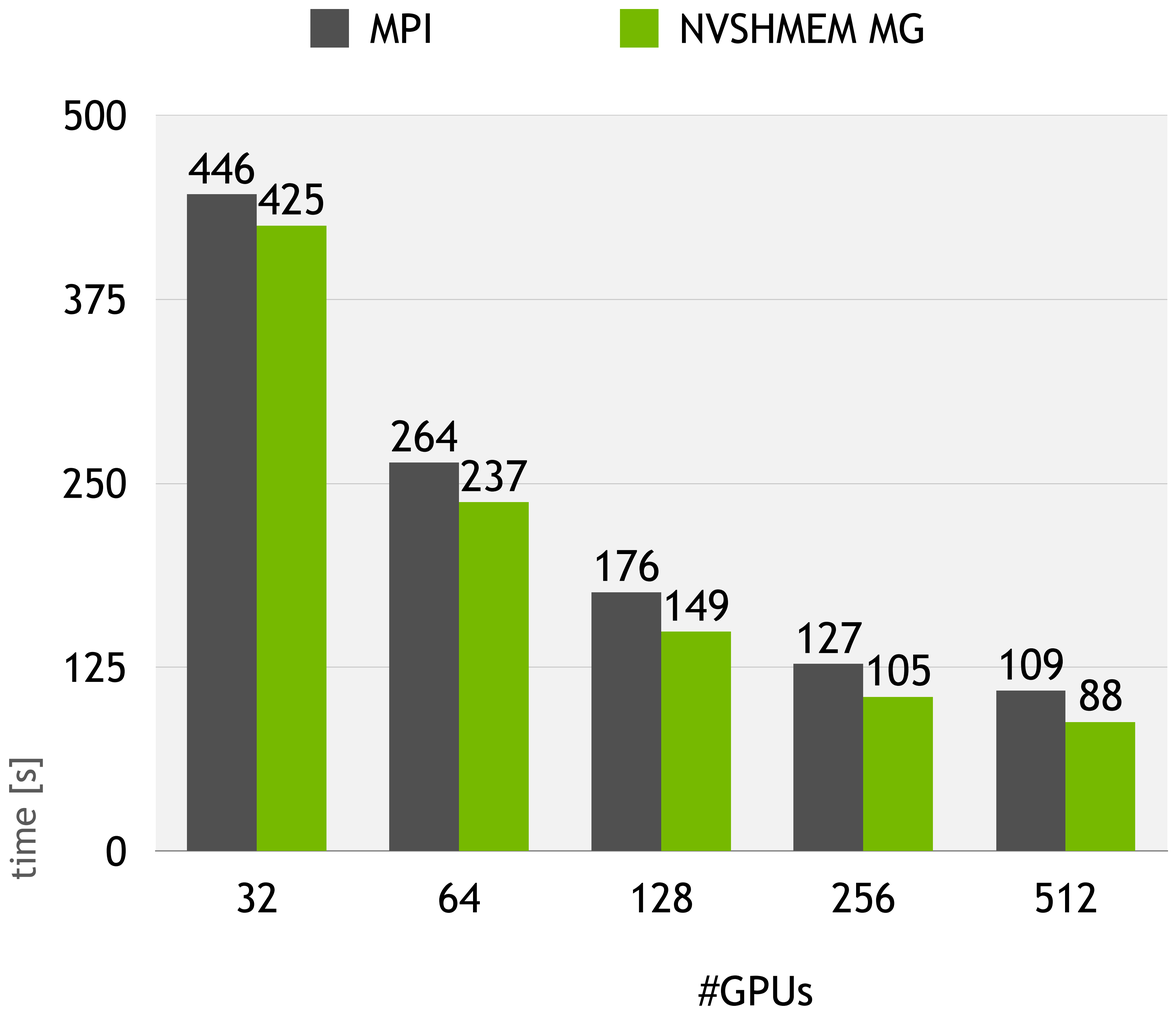




128 x NVIDIA A100 (Selene)

CHROMA WILSON-CLOVER HMC SCALING

Improvements 2021 (MPI) to 2022 (NVSHMEM)



Time for full HMC trajectory

NVSHMEM for Multigrid enabled

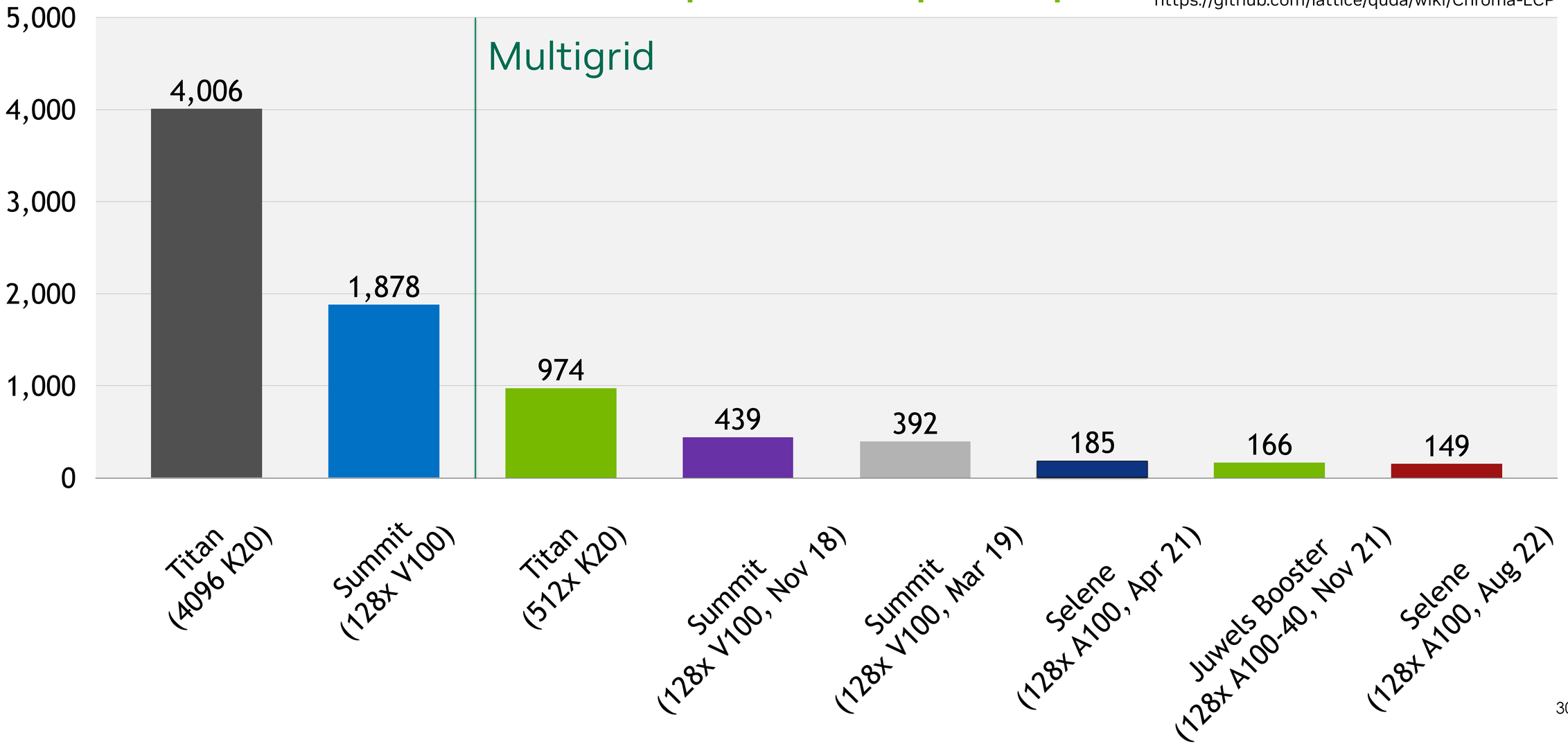
Same hardware - Same algorithm

Up to ~20% speedup

CHROMA ECP BENCHMARK

Multiplicative Speedup

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



OUTLOOK NVSHMEM WITH QUDA

RHMC scaling benefits in MILC and Chroma, up to 20% reduction in time to solution

- MILC dominated by Multishift: Full NVSHMEM benefit
- Chroma dominated by Multigrid, now also improved with NVSHMEM

Further improve performance over IB using GPU Initiated Communications (GIC)

NVSHMEM everywhere:

- Fully fused coarse Dslash kernel
- Further kernel fusion (multiple Dslash applications, solvers)
- Remove MPI / GPU synchronizations