GPU TECHNOLOGY CONFERENCE
April 4-7, 2016 | Silicon Valley

# SIMPLIFYING MULTI-GPU COMMUNICATION WITH NVSHMEM

Sreeram Potluri, Nathan Luehr, and Nikolay Sakharnykh

PRESENTED BY
NVIDIA.

# GOAL

Limitations for strong scaling on GPU clusters

Possibly address with GPU Global Address:  NVSHMEM

Case studies using NVSHMEM

Start of a discussion and not a solution

# PROGRAMMING WITH NVSHMEM

# GPU CLUSTER PROGRAMMING

Offload model
- Compute on GPU
- Communication from CPU

Synchronization at boundaries

Overheads on GPU Clusters
- Offload latencies
- Synchronization overheads

Limits strong scaling

More CPU means more power

```
void 2dstencil (u, v, …)
{
  for (timestep = 0; …) {
    interior_compute_kernel <<<…>>> (…)
    pack_kernel <<<…>>> (…)
    cudaStreamSynchronize(…)
    MPI_Irecv(…)
    MPI_Isend(…)
    MPI_Waitall(…)
    unpack_kernel <<<…>>> (…)
    boundary_compute_kernel <<<…>>> (…)
    …
  }
}
```

# GPU-CENTRIC COMMUNICATION

GPU capabilities
- Compute state to hide latencies to global memory
- Implicit coalescing of loads/stores to achieve efficiency

CUDA helps to program to these

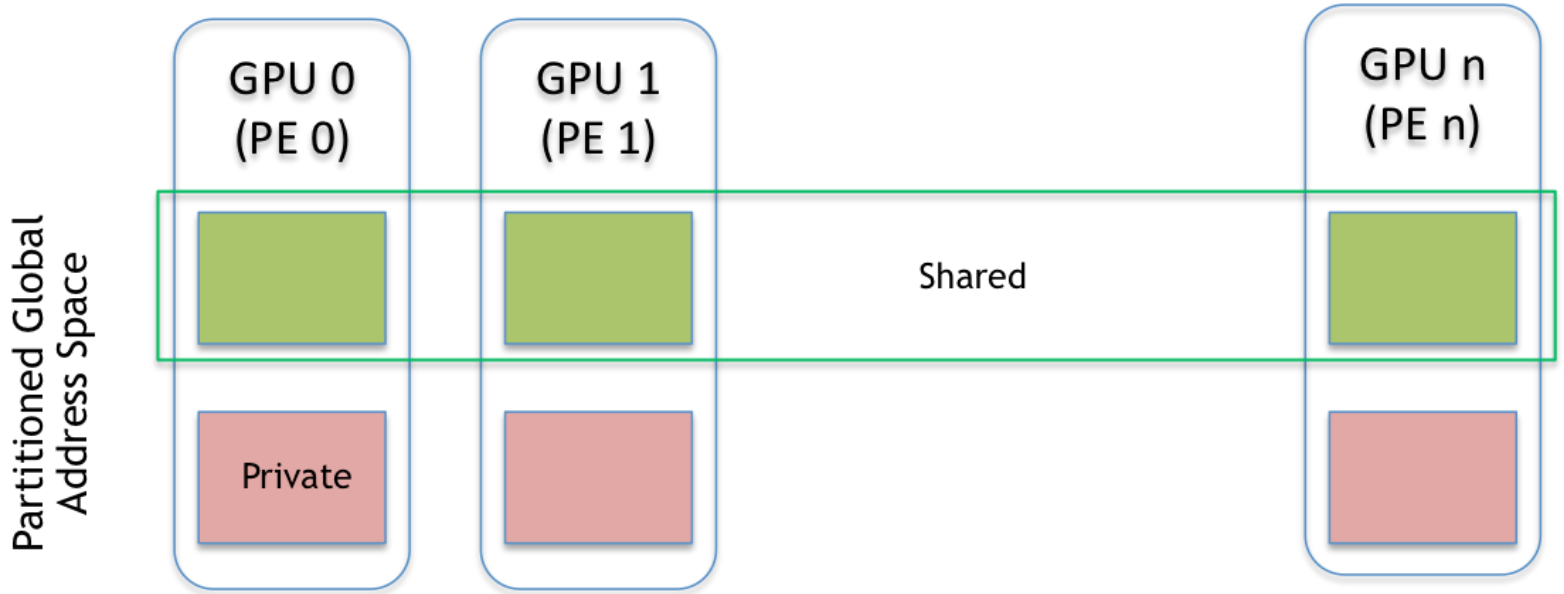
Should also benefit when accessing data over the network

Direct accesses to remote memory simplifies programming

Achieving efficiency while making it easier to program

Continuous fine-grained accesses smooths traffic over the network

# GPU GLOBAL ADDRESS SPACE



# NVSHMEM

A subset of OpenSHMEM

Interoperability with MPI/OpenSHMEM, in CUDA kernels/OpenACC regions

Host: initialization and cleanup (host)

- nvstart_pes, nvstop_pes

allocation and deallocation (host)

- nvshmalloc and nvshmcleanup

nvshmem_barrier_all (host)

nvshmem_get_ptr (host/GPU)

put and get routines (GPU)

- nvshmem_(float/int)_(p/g)
- nvshmem_(float/int)_(put/get)

nvshmem_(quiet/fence) (GPU)

nvshmem_wait/wait_until (GPU)

# COMMUNICATION FROM CUDA KERNELS

Long running CUDA kernels

Communication within parallelism

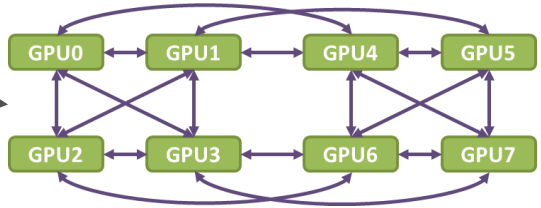
```
__global__ void 2dstencil (u, v, sync, …)
{
    for(timestep = 0; …) {
        if (i+1 > nx) {
            v[i+1] = shmem_float_g (v[1], rightpe);
        }
        if (i-1 < 1) {
            v[i-1] = shmem_float_g (v[nx], leftpe);
        }

        u[i] = (u[i] + (v[i+1] + v[i-1] . . .

        if (i < 2) {
            shmem_int_p (sync + i, 1, peers[i]);
            shmem_quiet();
            shmem_wait_until (sync + i, EQ, 1);
        }
        //intra-kernel sync
        …
    }
}
```

# EXPERIMENTAL PLATFORMS

Single node – GPUs directly connected with NVLink

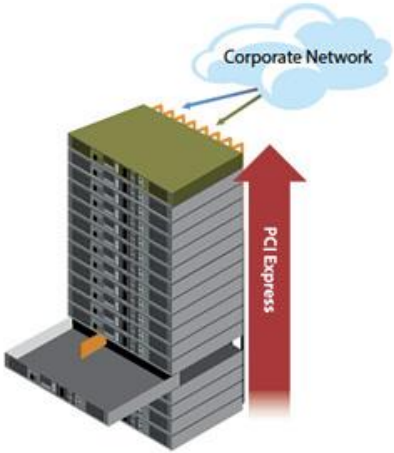



Single node – up to 8 GPUs – 2 per card – 4 cards under same PCIe root complex using raiser cards with PCIe switch

CUDA IPC and P2P

Multi-node platform – Top-of-Rack PCIe Switch – ExpressFabric, proprietary technology from Avago Technologies

Inter Host Communication with TWC – Tunneled Window Connection




Source: http://www.avagotech.com/applications/datacenters/expressfabric/

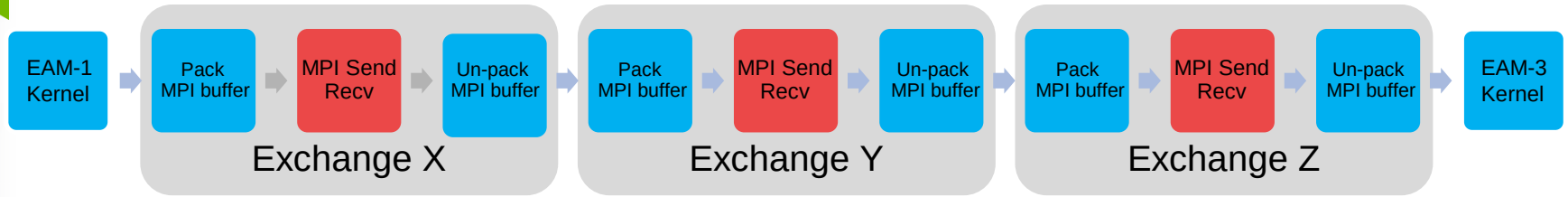
# CURRENT ARCHITECTURES

| | Operations | Kepler + P2P over PCIe (Single Node) | Kepler + PCIe Express Fabric (Multi Node) | Pascal + NVLink |
|---|---|---|---|---|
| Communication | Write | ☑ | ☑ | ☑ |
| | Read | ☑ | ☒ | ☑ |
| | Atomics | ☒ | ☒ | ☑ |
| Execution | Inter-thread Synchronization | (1) Avoid intra-WARP synchronization<br>(2) Ensure synchronizing blocks are scheduled | | |

# PERFORMANCE STUDIES

# CoMD MOLECULAR DYNAMICS

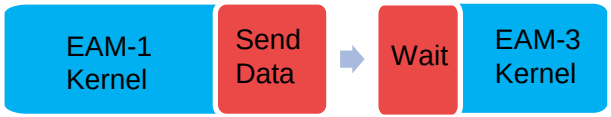
## MPI vs. NVSHMEM for Halo Exchange in EAM Force Evaluation




GPU-driven communication

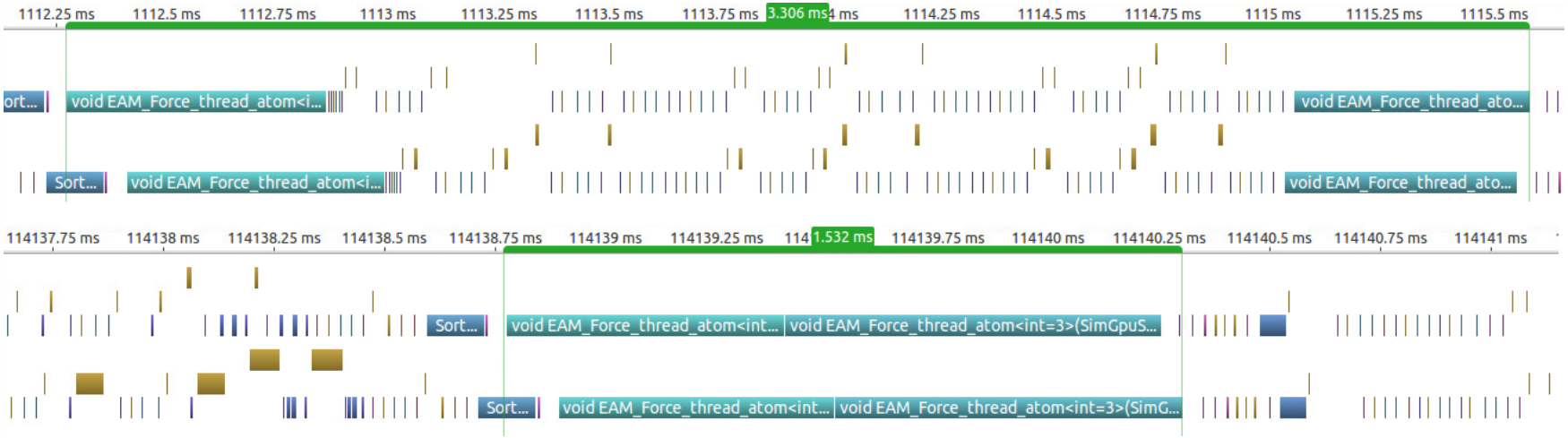
Fine-grained communication at the thread level

Avoids synchronization and artificial serialization

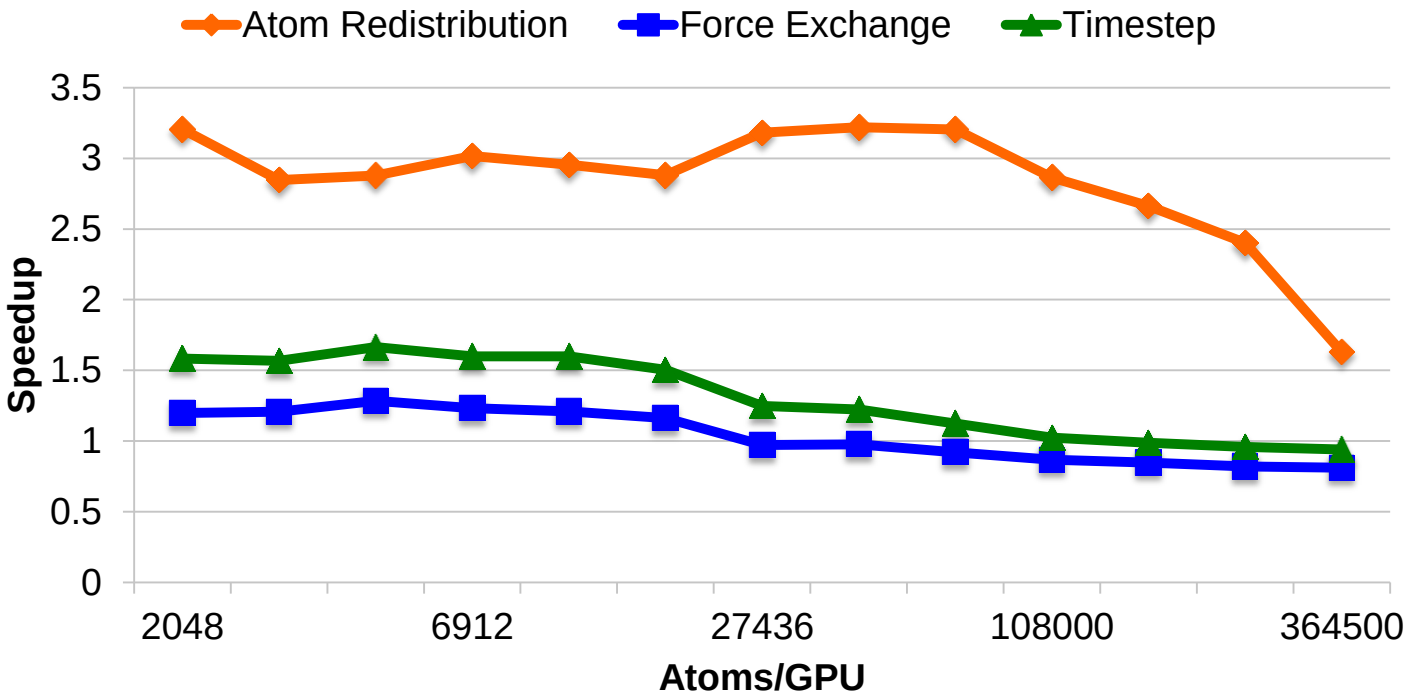


# CoMD FORCE EVALUATION

## NVProf timeline for EAM forces using Link Cells



# CoMD PERFORMANCE




*4 K80s (8 GPUs) connected over PCIe*

# MULTI-GPU TRANSPOSE

GPU 0

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |

GPU 1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |
| 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 |
| 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |

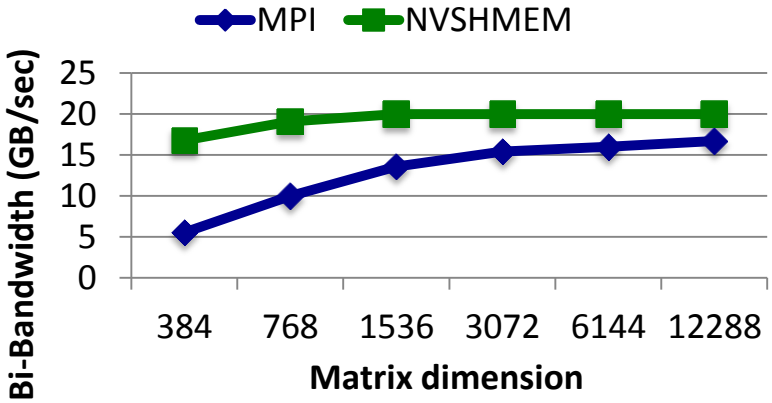
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 8 | 16 | 24 | 32 | 40 | 48 | 56 |
| 1 | 9 | 17 | 25 | 33 | 41 | 49 | 57 |
| 2 | 10 | 18 | 26 | 34 | 42 | 50 | 58 |
| 3 | 11 | 19 | 27 | 35 | 43 | 51 | 59 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 4 | 12 | 20 | 28 | 36 | 44 | 52 | 60 |
| 5 | 13 | 21 | 29 | 37 | 45 | 53 | 61 |
| 6 | 14 | 22 | 30 | 38 | 46 | 54 | 62 |
| 7 | 15 | 23 | 31 | 39 | 47 | 55 | 63 |

Bandwidth limited

MPI version carefully pipelines local transposes and inter-process data movement

NVSHMEM significantly reduces code complexity




*2 K40s connected over PCIe*

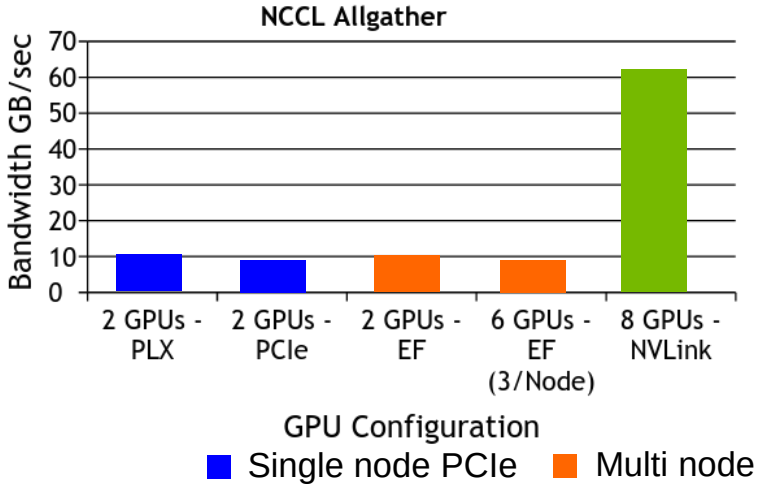
# COLLECTIVE COMMUNICATION

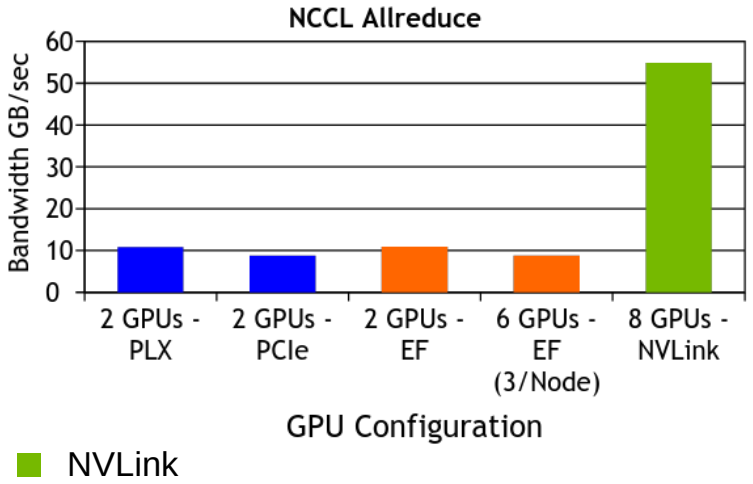
NCCL collectives communication library

Uses fine-grained load/stores between GPUs – No DMAs used!

Pipelines data movement and overlaps it with computation (virtue of WARP scheduling)

Implemented over NVSHMEM




Single node PCIe  Multi node PCIe  NVLink

# HPGMG-FV

## Intra-level communication

Proxy for geometric multi-grid linear solvers

Boundary exchange is symmetric

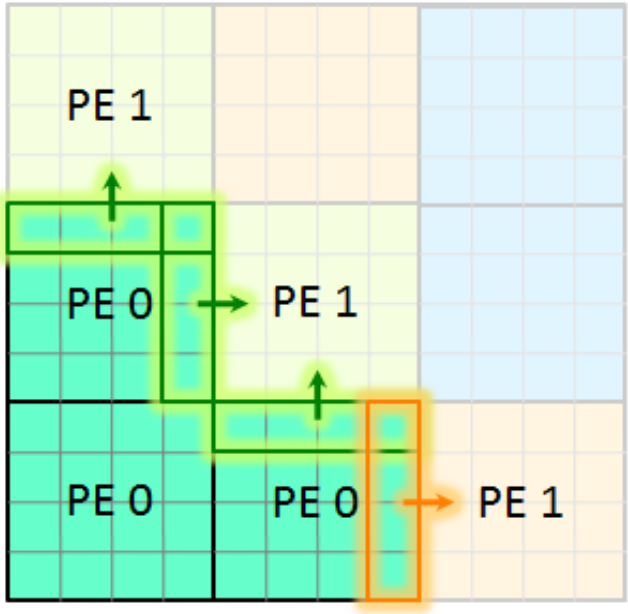
Point-to-point between neighbors

MPI uses 3 Steps:

1 – send data (boundary->MPI buffer)

2 – local exchange (internal->internal)

3 – receive data (MPI buffer->boundary)



# HPGMG-FV – BOUNDARY EXCHANGE

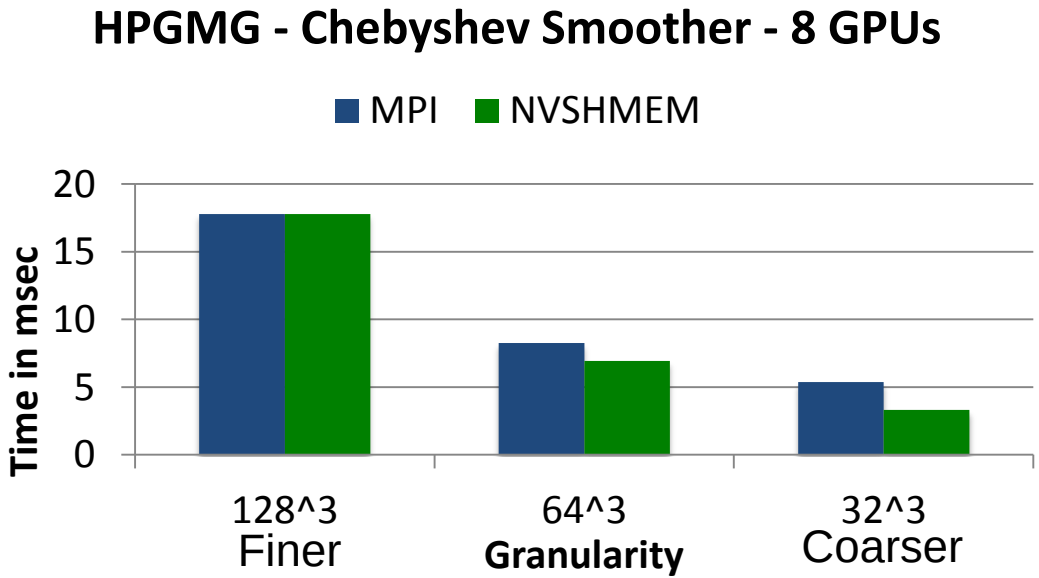
## Implementation complexity

| MPI | NVSHMEM |
|---|---|
| CopyKernel(BOUNDARY-TO-BUFFER)<br>cudaDeviceSync<br>MPI_Irecv + MPI_Isend<br>CopyKernel(INTERNAL-TO-INTERNAL)<br>MPI_Waitall<br>CopyKernel(BUFFER-TO-BOUNDARY) | CopyKernel(ALL-TO-ALL)<br>Nvshmem_barrier_all_offload |

# HPGMG CHEBYSHEV SMOOTHER

Limited by latencies – more so at coarser levels

Use fine-grained put/get with NVSHMEM




*4 K80s (8 GPUs) connected over PCIe*

# SUMMARY

Strong scaling important on GPU clusters

Overheads from CPU orchestrated communication

NVSHMEM is a prototype library for GPU-initiated Communication

Better performance and better programmability

Promising results with NVIDIA Collectives library and Mini-Apps

# THANK YOU!