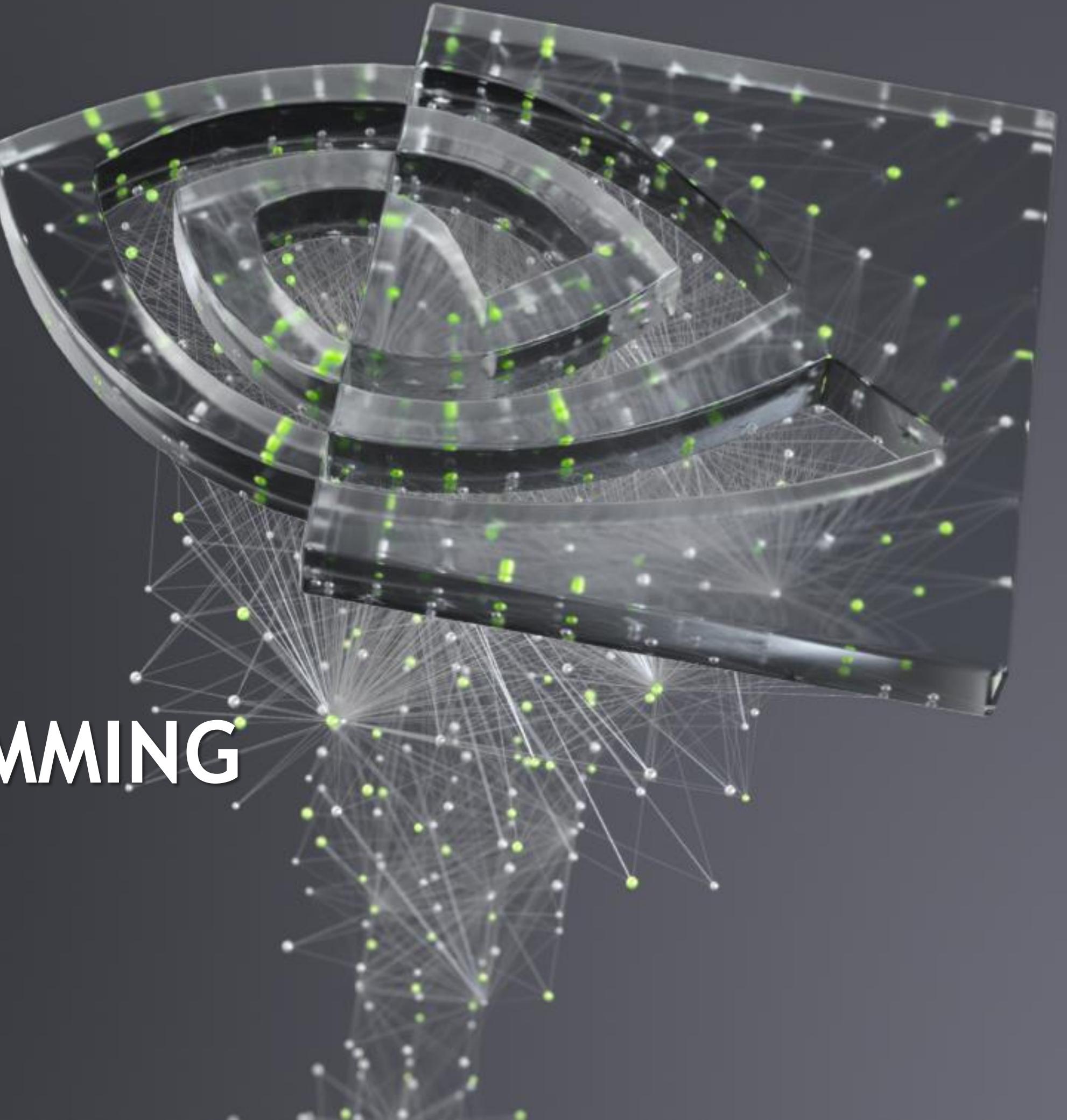




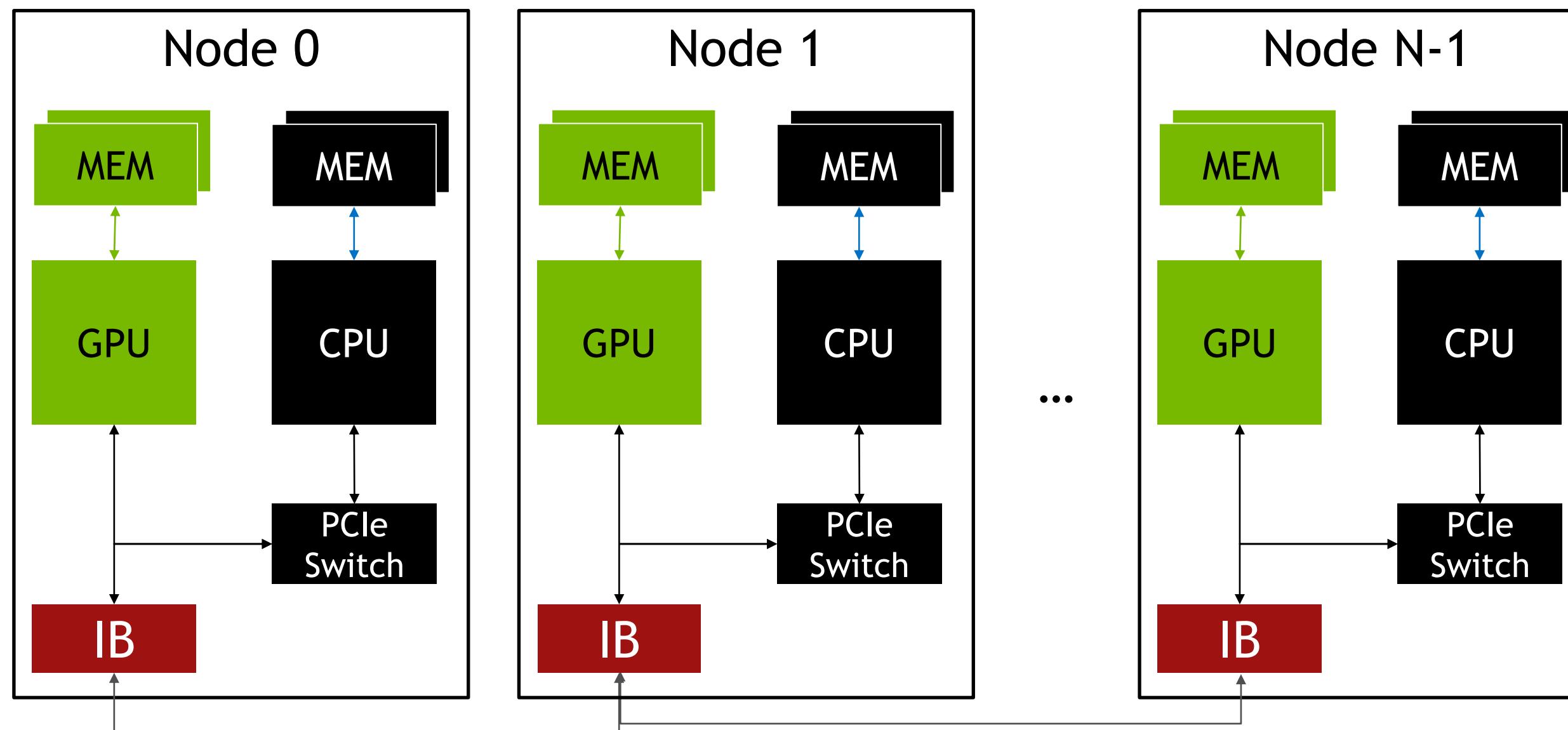
NVIDIA®

# MULTI GPU PROGRAMMING WITH MPI

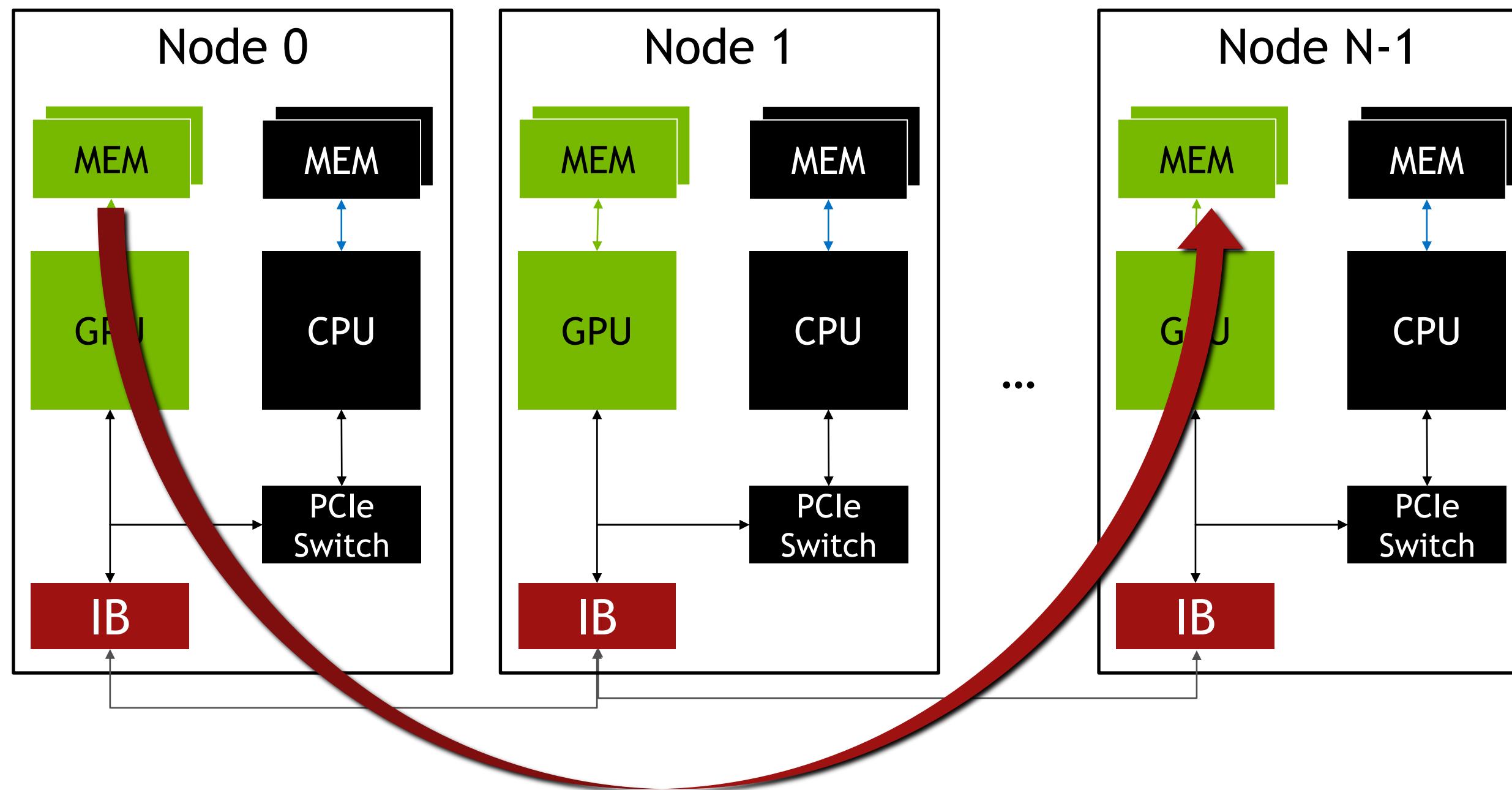
Jiri Kraus, Senior Devtech Compute



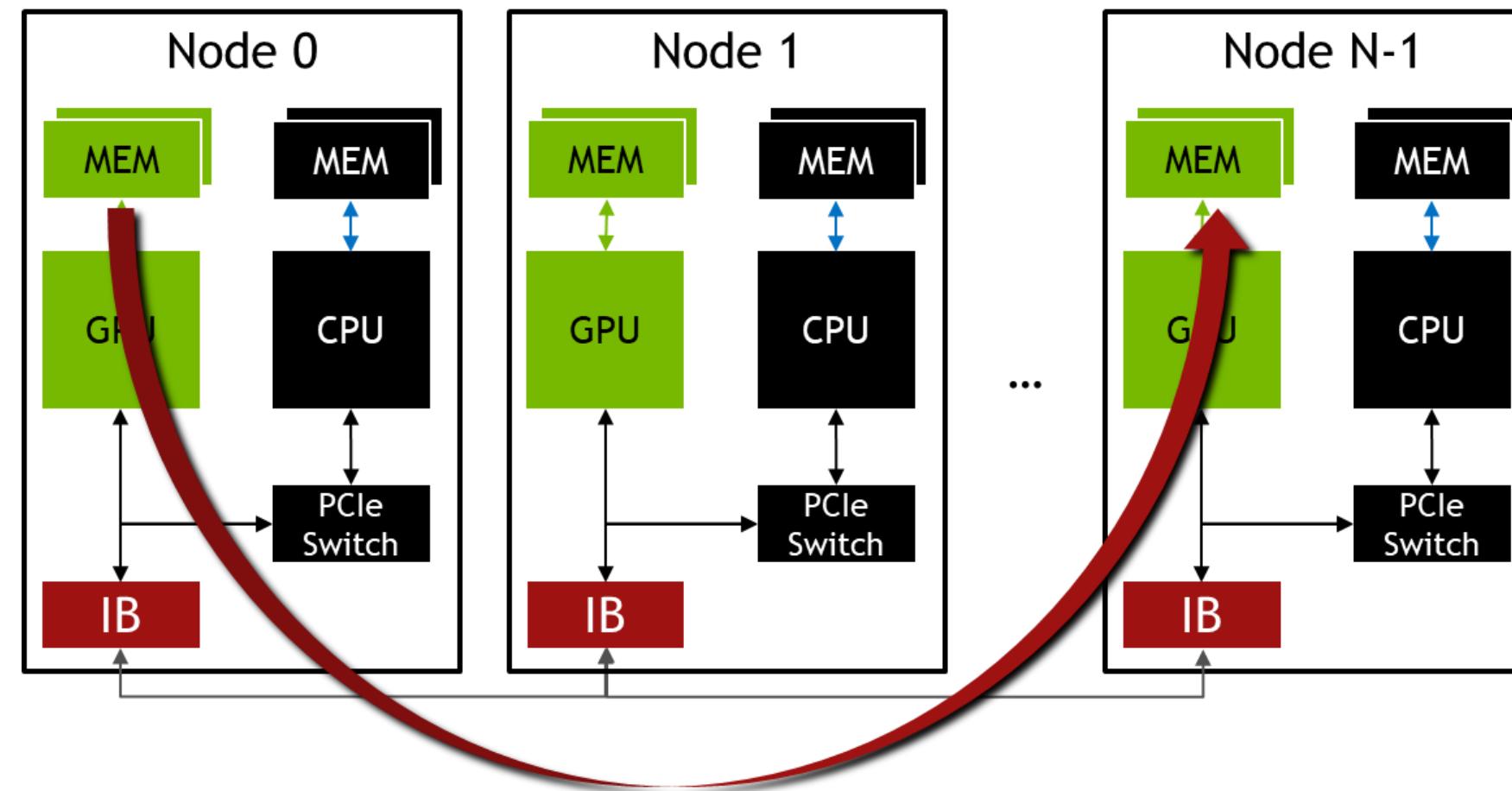
# MPI+CUDA



# MPI+CUDA



# MPI+CUDA



```
//MPI rank 0  
MPI_Send(s_buf_d, size, MPI_BYTE, n-1, tag, MPI_COMM_WORLD);  
  
//MPI rank n-1  
MPI_Recv(r_buf_d, size, MPI_BYTE, 0, tag, MPI_COMM_WORLD, &stat);
```

# YOU WILL LEARN

How to use MPI for inter GPU communication with CUDA and OpenACC

How CUDA-aware MPI works

What Multi Process Service is and how to use it

How to use NVIDIA tools in an MPI environment

How to hide MPI communication times

# A SIMPLE EXAMPLE

# EXAMPLE: JACOBI SOLVER

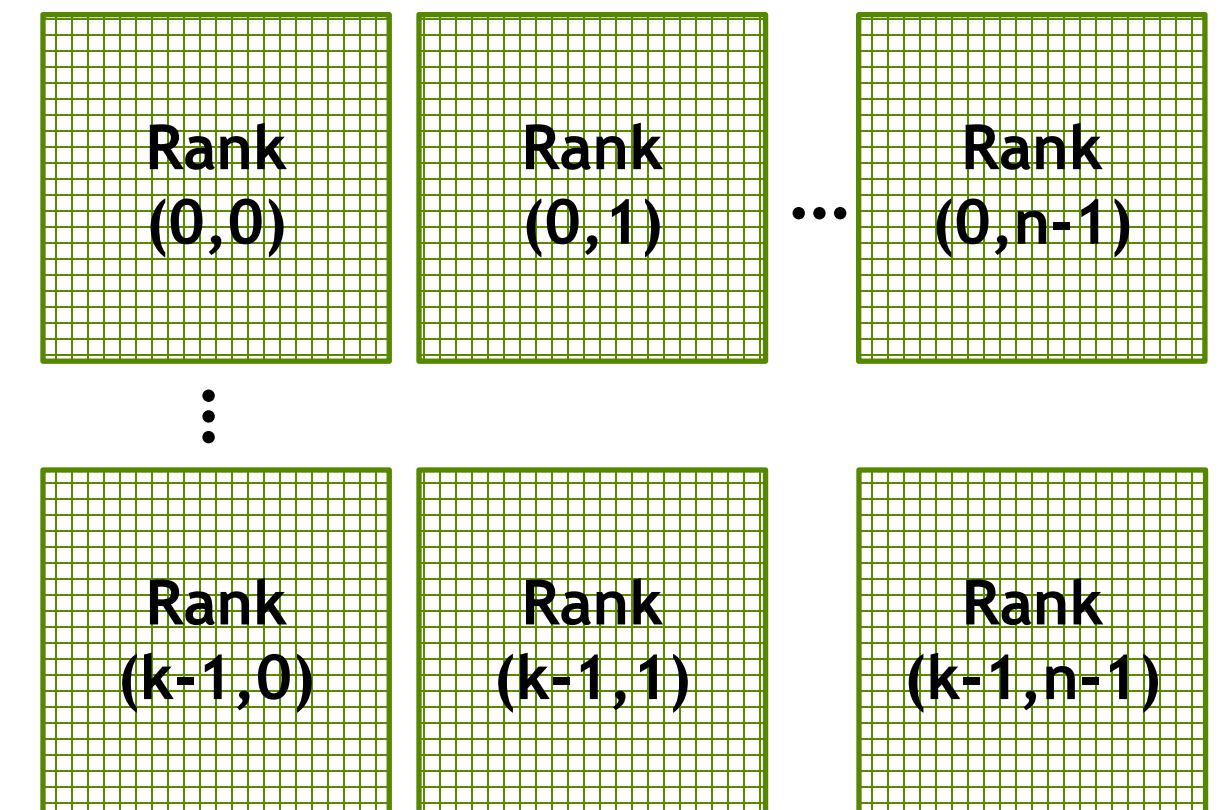
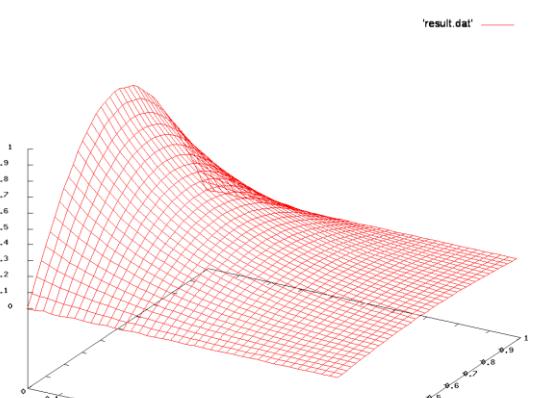
Solves the 2D-Laplace Equation on a rectangle

$$\Delta u(x, y) = 0 \quad \forall (x, y) \in \Omega \setminus \delta\Omega$$

Dirichlet boundary conditions (constant values on boundaries)

$$u(x, y) = f(x, y) \quad \forall (x, y) \in \delta\Omega$$

2D domain decomposition with  $n \times k$  domains



# EXAMPLE: JACOBI SOLVER

## Single GPU

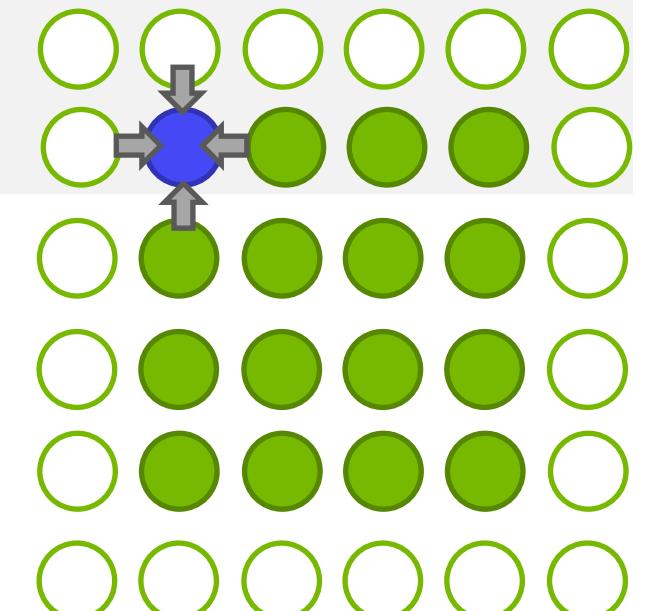
While not converged

Do Jacobi step:

```
for (int iy=1; iy < ny-1; ++iy)  
  
for (int ix=1; ix < nx-1; ++ix)  
  
    u_new[ix][iy] = 0.0f - 0.25f*( u[ix-1][iy] + u[ix+1][iy]  
                                + u[ix][iy-1] + u[ix][iy+1]);
```

Swap `u_new` and `u`

Next iteration



# EXAMPLE: JACOBI SOLVER

## Multi GPU

While not converged

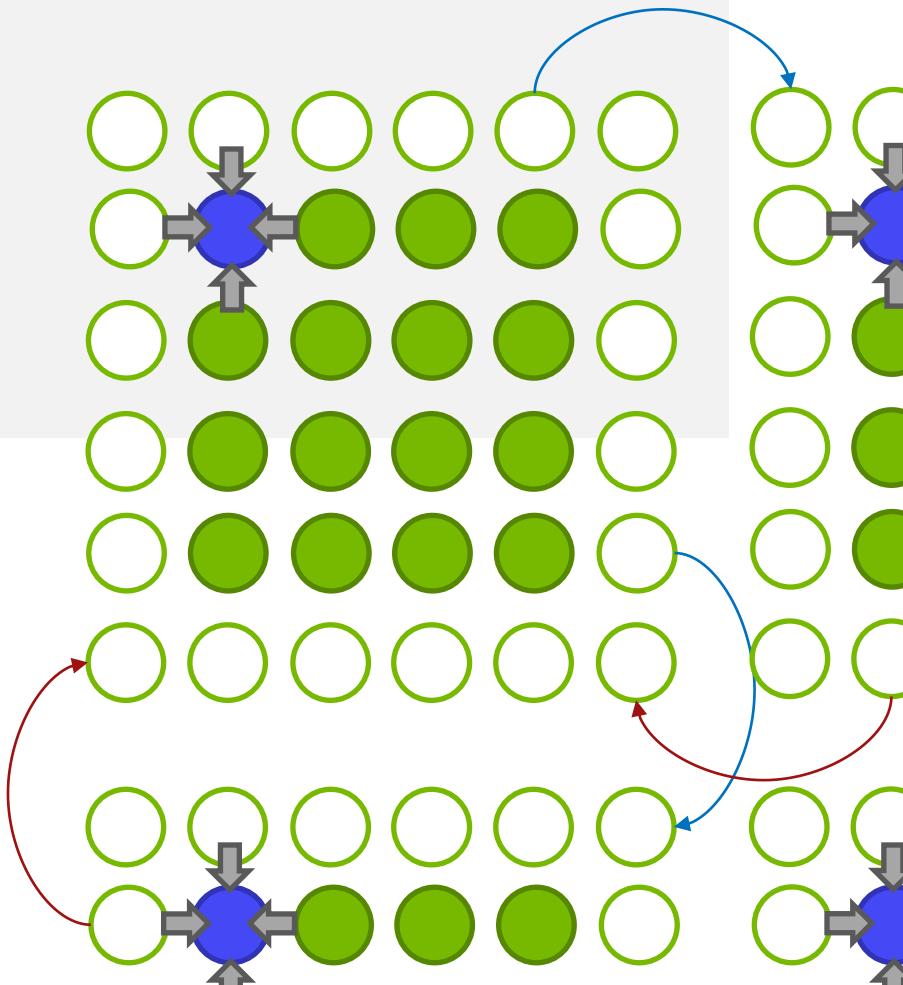
Do Jacobi step:

```
for (int iy=1; iy < ny-1; ++iy)  
  
for (int ix=1; ix < nx-1; ++ix)  
  
    u_new[ix][iy] = 0.0f - 0.25f*( u[ix-1][iy] + u[ix+1][iy]  
        + u[ix][iy-1] + u[ix][iy+1]);
```

Exchange halo with 1 to 4 neighbors

Swap  $u_{\text{new}}$  and  $u$

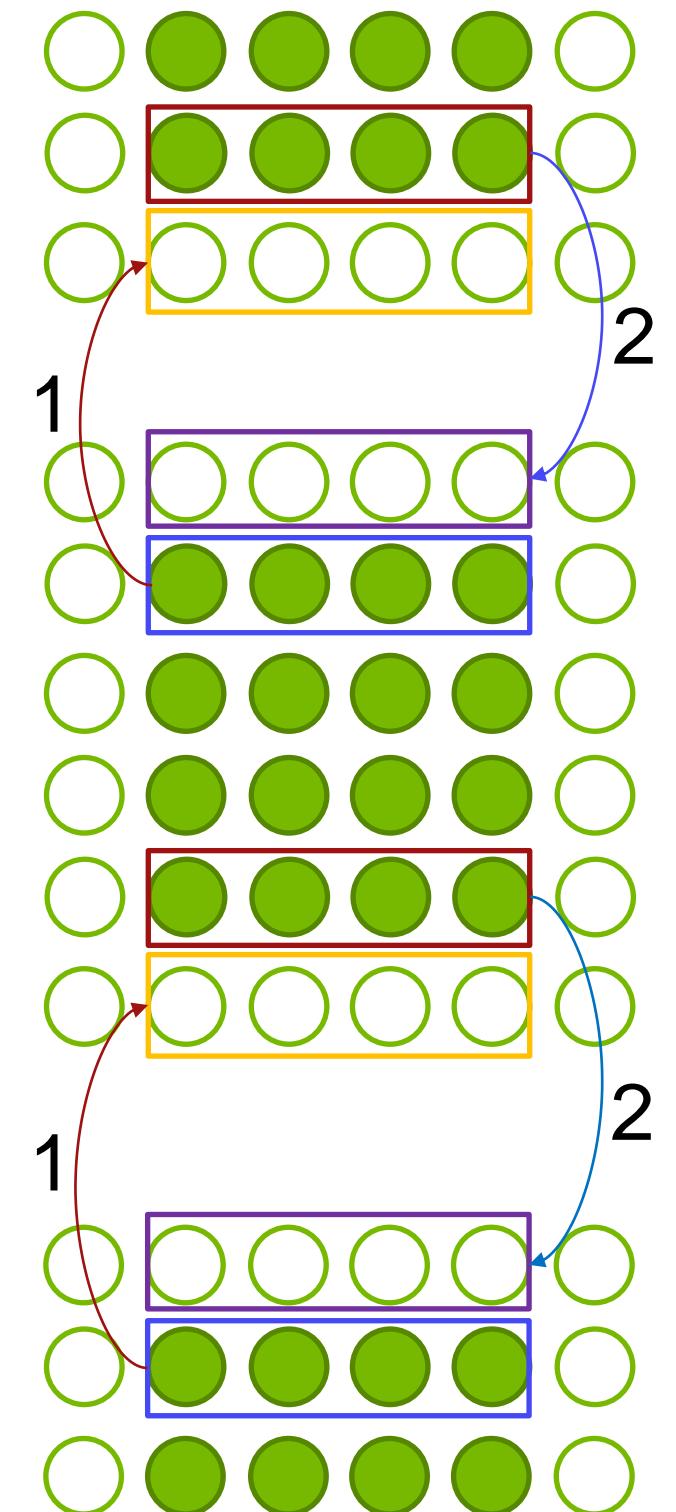
Next iteration



# EXAMPLE JACOBI

## Top/Bottom Halo

```
MPI_Sendrecv(u_new+offset_first_row, m-2, MPI_DOUBLE, t_nb, 0,  
             u_new+offset_bottom_boundary, m-2, MPI_DOUBLE, b_nb, 0,  
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);  
  
MPI_Sendrecv(u_new+offset_last_row, m-2, MPI_DOUBLE, b_nb, 1,  
             u_new+offset_top_boundary, m-2, MPI_DOUBLE, t_nb, 1,  
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```



# EXAMPLE JACOBI

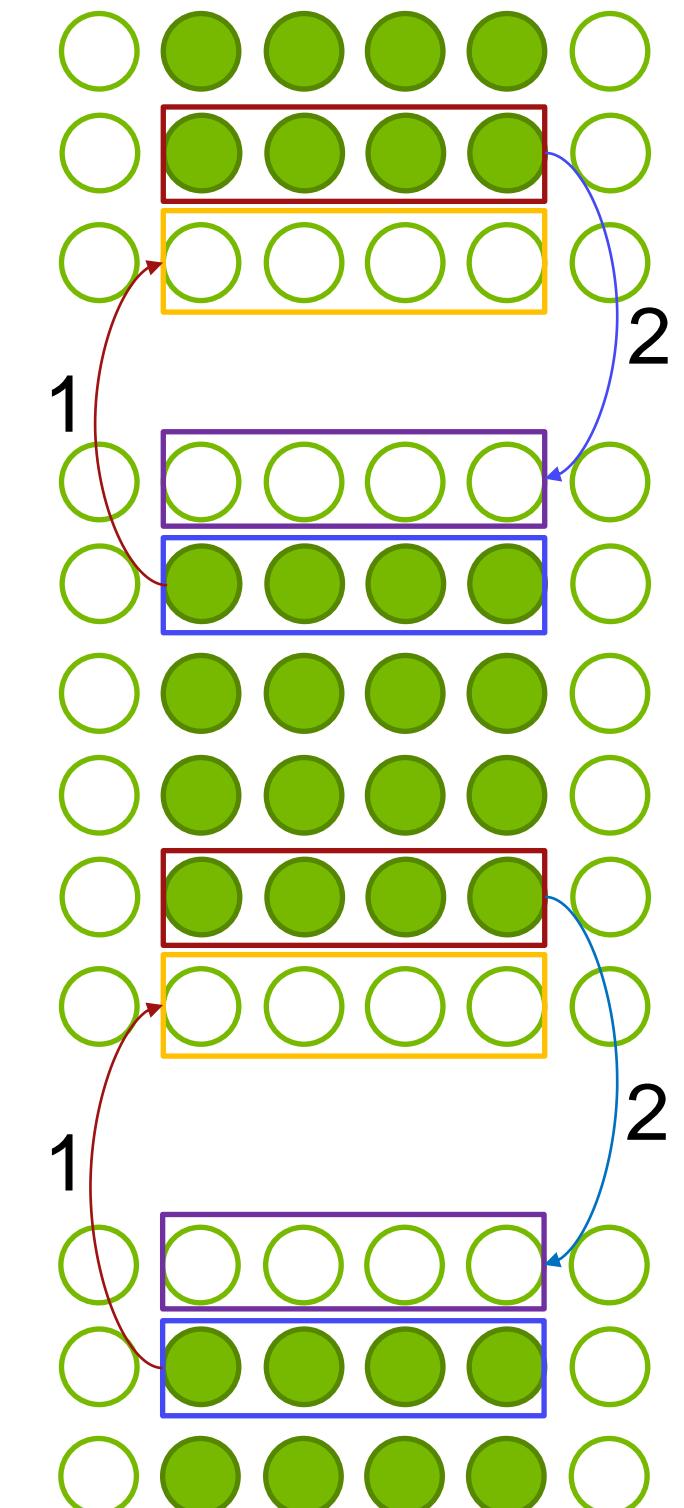
## Top/Bottom Halo

OpenACC

```
#pragma acc host_data use_device ( u_new ) {  
  
MPI_Sendrecv(u_new+offset_first_row, m-2, MPI_DOUBLE, t_nb, 0,  
             u_new+offset_bottom_boundary, m-2, MPI_DOUBLE, b_nb, 0,  
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);  
  
MPI_Sendrecv(u_new+offset_last_row, m-2, MPI_DOUBLE, b_nb, 1,  
             u_new+offset_top_boundary, m-2, MPI_DOUBLE, t_nb, 1,  
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);  
}
```

CUDA

```
MPI_Sendrecv(u_new_d+offset first row, m-2, MPI_DOUBLE, t_nb, 0,  
             u_new_d+offset bottom boundary, m-2, MPI_DOUBLE, b_nb, 0,  
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);  
  
MPI_Sendrecv(u_new_d+offset last row, m-2, MPI_DOUBLE, b_nb, 1,  
             u_new_d+offset_top_boundary, m-2, MPI_DOUBLE, t_nb, 1,  
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```



# EXAMPLE JACOBI

## Top/Bottom Halo

without  
CUDA-aware  
MPI

OpenACC

```
#pragma acc update host(u_new[offset_first_row:m-2],u_new[offset_last_row:m-2])
MPI_Sendrecv(u_new+offset_first_row, m-2, MPI_DOUBLE, t_nb, 0,
             u_new+offset_bottom_boundary, m-2, MPI_DOUBLE, b_nb, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
MPI_Sendrecv(u_new+offset_last_row, m-2, MPI_DOUBLE, b_nb, 1,
             u_new+offset_top_boundary, m-2, MPI_DOUBLE, t_nb, 1,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
#pragma acc update device(u_new[offset_top_boundary:m-2],u_new[offset_bottom_boundary:m-2])
```

CUDA

```
//send to bottom and receive from top top bottom omitted

cudaMemcpy( u_new+offset_first_row,
            u_new_d+offset_first_row, (m-2)*sizeof(double), cudaMemcpyDeviceToHost);
MPI_Sendrecv(u_new+offset_first_row, m-2, MPI_DOUBLE, t_nb, 0,
             u_new+offset_bottom_boundary, m-2, MPI_DOUBLE, b_nb, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
cudaMemcpy( u_new_d+offset_bottom_boundary,
            u_new+offset_bottom_boundary, (m-2)*sizeof(double), cudaMemcpyDeviceToHost);
```

# EXAMPLE: JACOBI

## Left/Right Halo

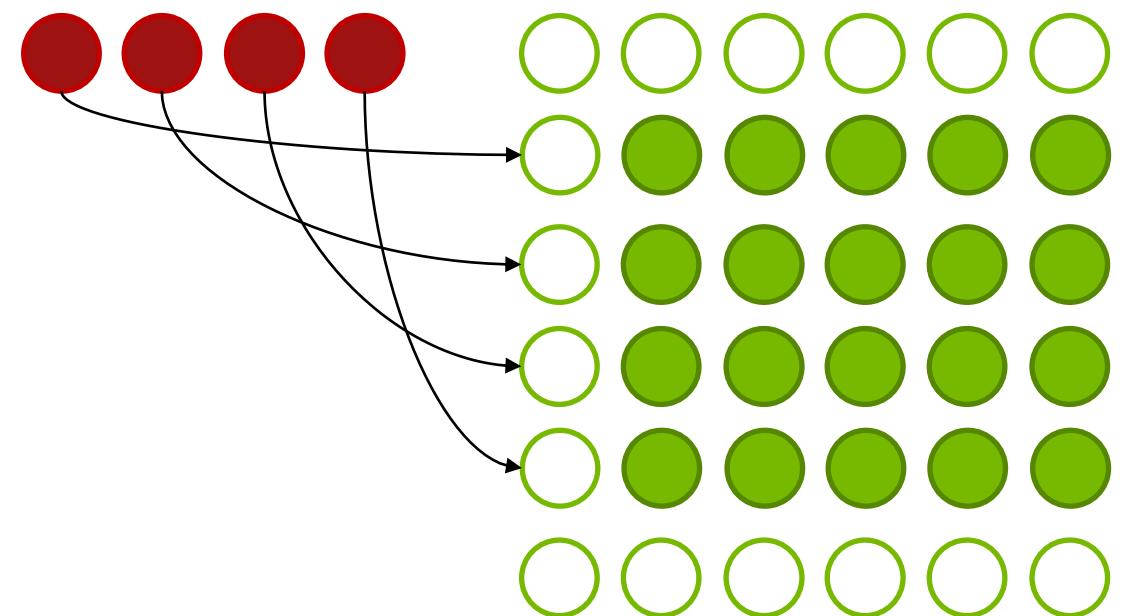
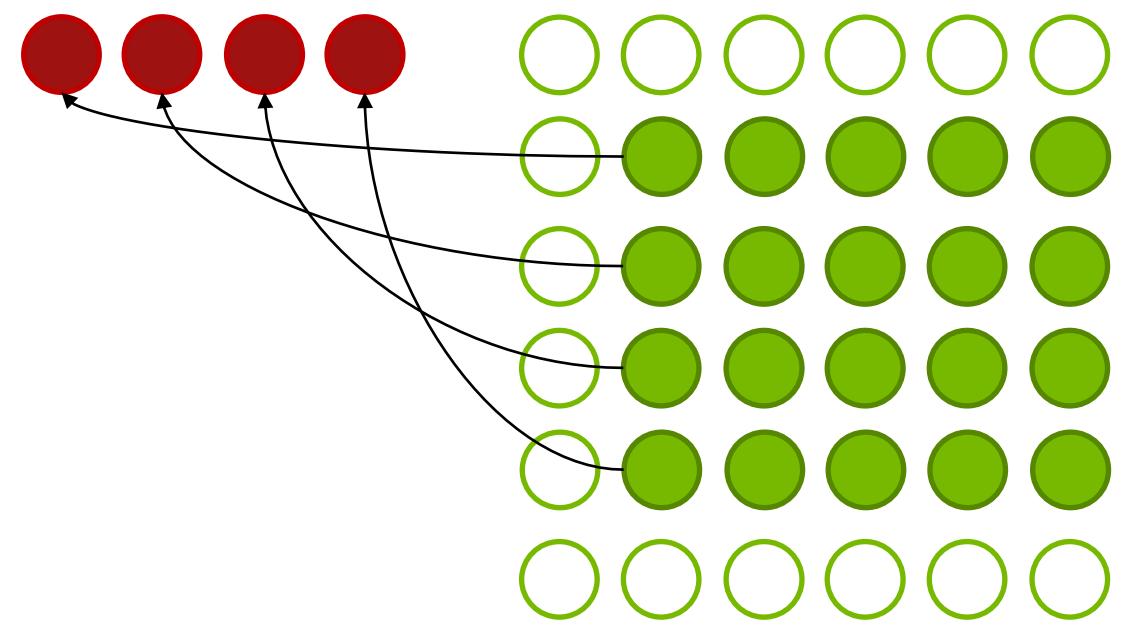
OpenACC

```
//right neighbor omitted

#pragma acc parallel loop present ( u_new, to_left )
for ( int i=0; i<n-2; ++i )
    to_left[i] = u_new[(i+1)*m+1];

#pragma acc host_data use_device ( from_right, to_left ) {
    MPI_Sendrecv( to_left, n-2, MPI_DOUBLE, l_nb, 0,
                  from_right, n-2, MPI_DOUBLE, r_nb, 0,
                  MPI_COMM_WORLD, MPI_STATUS_IGNORE );
}

#pragma acc parallel loop present ( u_new, from_right )
for ( int i=0; i<n-2; ++i )
    u_new[(m-1)+(i+1)*m] = from_right[i];
```

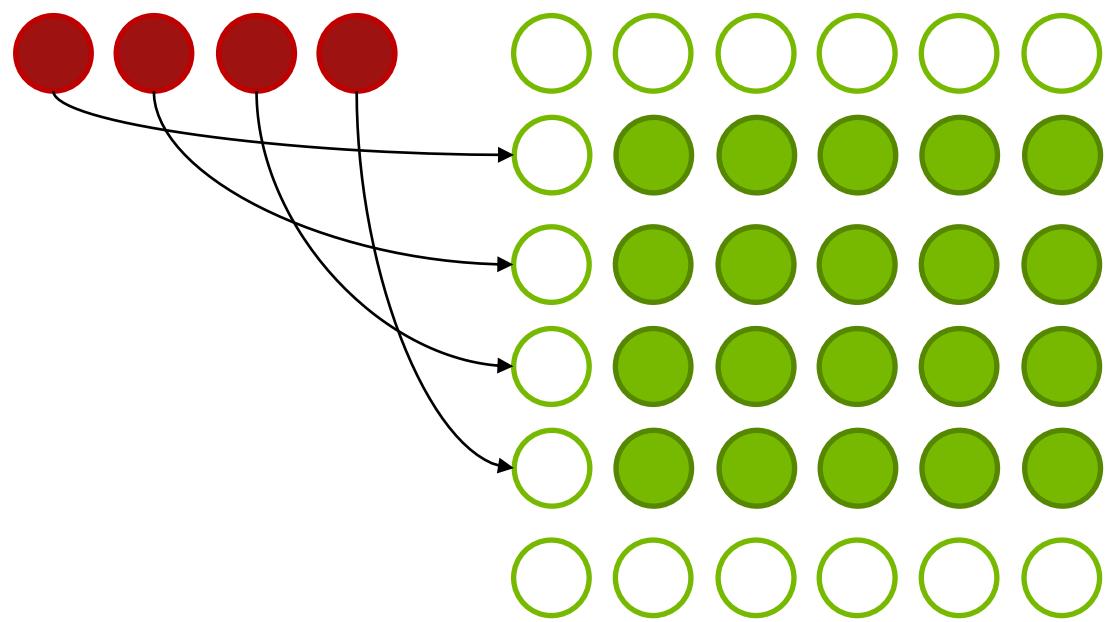
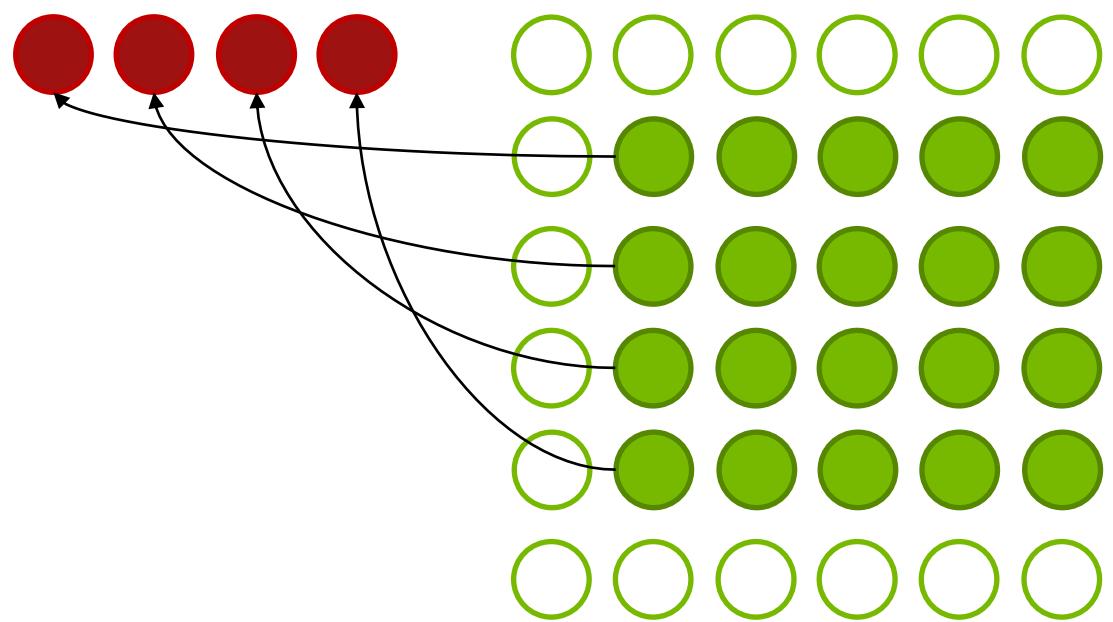


# EXAMPLE: JACOBI

## Left/Right Halo

CUDA

```
//right neighbor omitted  
pack<<<gs,bs,0,s>>>(to_left_d, u_new_d, n, m);  
cudaStreamSynchronize(s);  
  
MPI_Sendrecv( to_left_d, n-2, MPI_DOUBLE, l_nb, 0,  
              from_right_d, n-2, MPI_DOUBLE, r_nb, 0,  
              MPI_COMM_WORLD, MPI_STATUS_IGNORE );  
  
unpack<<<gs,bs,0,s>>>(u_new_d, from_right_d, n, m);
```



# LAUNCH MPI+CUDA/OPENACC PROGRAMS

Launch one process per GPU

MVAPICH: \$ **MV2\_USE\_CUDA=1** mpirun -np \${np} ./myapp <args>

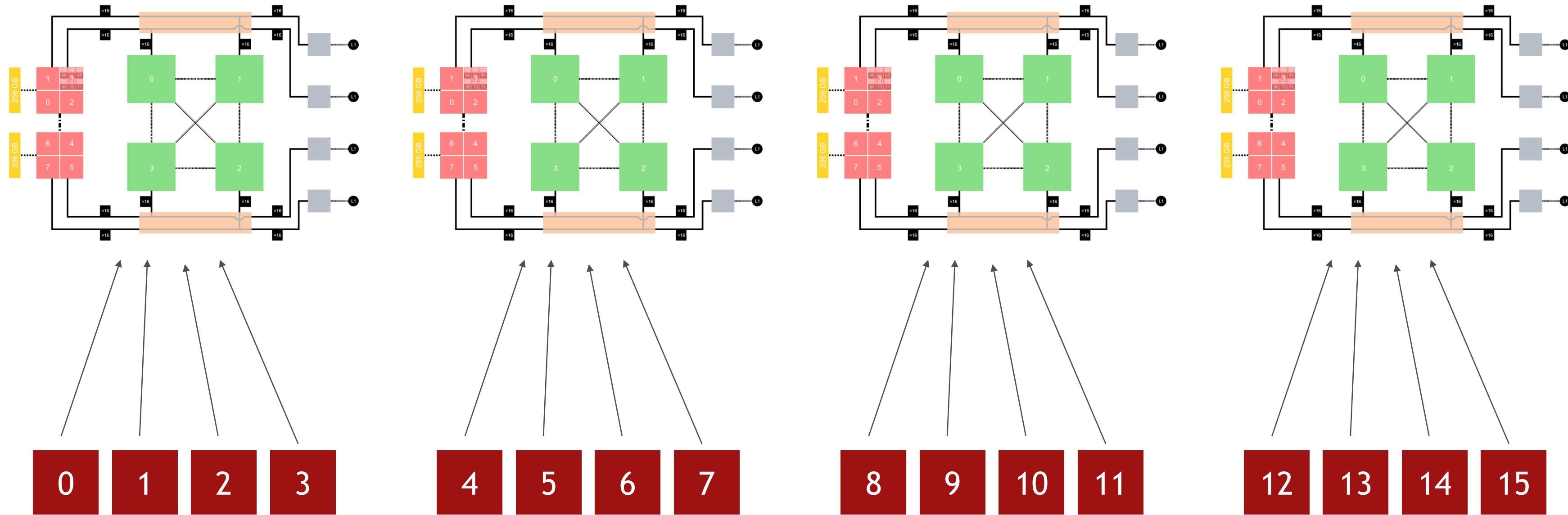
Open MPI: CUDA-aware features are enabled per default

Cray: MPICH\_RDMA\_ENABLED\_CUDA

IBM Spectrum MPI: \$ mpirun -gpu -np \${np} ./myapp <args>

ParaStation MPI: \$ **PSP\_CUDA=1** mpirun -np \${np} ./myapp <args>

# HANDLING MULTI GPU NODES

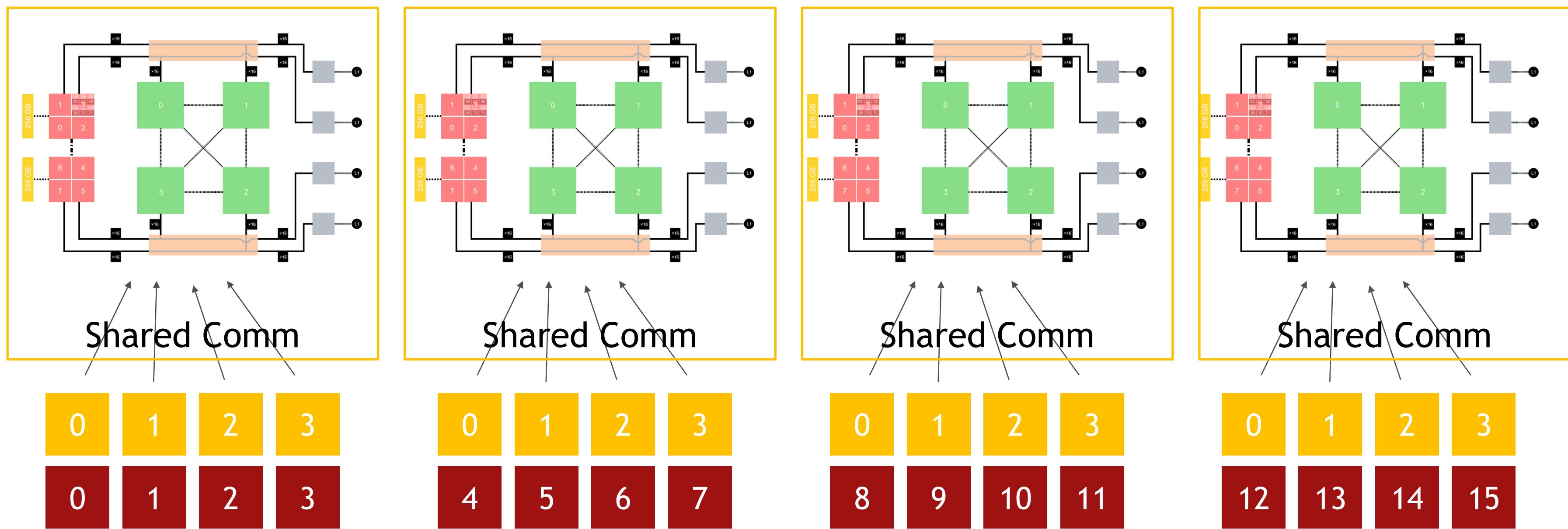


# HANDLING MULTI GPU NODES

## How to determine the local rank? - MPI-3

```
MPI_Comm local_comm;  
  
MPI_Info info;  
  
MPI_Info_create(&info);  
  
MPI_Comm_split_type(MPI_COMM_WORLD, MPI_COMM_TYPE_SHARED, rank, info, &local_comm);  
  
int local_rank = -1;  
  
MPI_Comm_rank(local_comm, &local_rank);  
  
MPI_Comm_free(&local_comm);  
  
MPI_Info_free(&info);
```

# HANDLING MULTI GPU NODES



# HANDLING MULTI GPU NODES

## GPU-affinity

Use local rank:

```
int local_rank = //determine local rank  
int num_devs = 0;  
cudaGetDeviceCount(&num_devs);  
cudaSetDevice(local_rank%num_devs);
```

Needed if  
resource manager  
handles GPU  
affinity

# UCX TIPS AND TRICKS

## Example binding script

<pre> case \${SLURM_LOCALID} in   0)     export CUDA_VISIBLE_DEVICES=0     export UCX_NET_DEVICES=mlx5_1:1     CPU_BIND=18-23     ;;   1)     export CUDA_VISIBLE_DEVICES=1     export UCX_NET_DEVICES=mlx5_0:1     CPU_BIND=6-11     ;;   2)     export CUDA_VISIBLE_DEVICES=2     export UCX_NET_DEVICES=mlx5_3:1     CPU_BIND=42-47     ;;   3)     export CUDA_VISIBLE_DEVICES=3     export UCX_NET_DEVICES=mlx5_2:1     CPU_BIND=30-35     ;; esac numactl --physcpubind=\${CPU_BIND} \$* </pre>	<p>[kraus1@jwb0007]\$ nvidia-smi topo -m</p> <table border="1"> <thead> <tr> <th></th> <th>GPU0</th> <th>GPU1</th> <th>GPU2</th> <th>GPU3</th> <th>mlx5_0</th> <th>mlx5_1</th> <th>mlx5_2</th> <th>mlx5_3</th> <th>CPU Affinity</th> <th>NUMA Affinity</th> </tr> </thead> <tbody> <tr> <td><b>GPU0</b></td> <td>X</td> <td>NV4</td> <td>NV4</td> <td>NV4</td> <td>SYS</td> <td>PIX</td> <td>SYS</td> <td>SYS</td> <td><b>18-23,66-71</b></td> <td>3</td> </tr> <tr> <td><b>GPU1</b></td> <td>NV4</td> <td>X</td> <td>NV4</td> <td>NV4</td> <td>PIX</td> <td>SYS</td> <td>SYS</td> <td>SYS</td> <td><b>6-11,54-59</b></td> <td>1</td> </tr> <tr> <td><b>GPU2</b></td> <td>NV4</td> <td>NV4</td> <td>X</td> <td>NV4</td> <td>SYS</td> <td>SYS</td> <td>SYS</td> <td>PIX</td> <td><b>42-47,90-95</b></td> <td>7</td> </tr> <tr> <td><b>GPU3</b></td> <td>NV4</td> <td>NV4</td> <td>NV4</td> <td>X</td> <td>SYS</td> <td>SYS</td> <td>PIX</td> <td>SYS</td> <td><b>30-35,78-83</b></td> <td>5</td> </tr> <tr> <td>mlx5_0</td> <td>SYS</td> <td>PIX</td> <td>SYS</td> <td>SYS</td> <td>X</td> <td>SYS</td> <td>SYS</td> <td>SYS</td> <td></td> <td></td> </tr> <tr> <td>mlx5_1</td> <td>PIX</td> <td>SYS</td> <td>SYS</td> <td>SYS</td> <td>SYS</td> <td>X</td> <td>SYS</td> <td>SYS</td> <td></td> <td></td> </tr> <tr> <td>mlx5_2</td> <td>SYS</td> <td>SYS</td> <td>SYS</td> <td>PIX</td> <td>SYS</td> <td>SYS</td> <td>X</td> <td>SYS</td> <td></td> <td></td> </tr> <tr> <td>mlx5_3</td> <td>SYS</td> <td>SYS</td> <td>PIX</td> <td>SYS</td> <td>SYS</td> <td>SYS</td> <td>SYS</td> <td>X</td> <td></td> <td></td> </tr> </tbody> </table> <p>Legend:</p> <ul style="list-style-type: none"> <li>X = Self</li> <li>SYS = Connection traversing PCIe as well as the SMP interconnect</li> <li>NODE = Connection traversing PCIe as well as the interconnection node</li> <li>PHB = Connection traversing PCIe as well as a PCIe Host Bridge</li> <li>PXB = Connection traversing multiple PCIe bridges (without</li> <li>PIX = Connection traversing at most a single PCIe bridge</li> <li>NV# = Connection traversing a bonded set of # NVLinks</li> </ul>		GPU0	GPU1	GPU2	GPU3	mlx5_0	mlx5_1	mlx5_2	mlx5_3	CPU Affinity	NUMA Affinity	<b>GPU0</b>	X	NV4	NV4	NV4	SYS	PIX	SYS	SYS	<b>18-23,66-71</b>	3	<b>GPU1</b>	NV4	X	NV4	NV4	PIX	SYS	SYS	SYS	<b>6-11,54-59</b>	1	<b>GPU2</b>	NV4	NV4	X	NV4	SYS	SYS	SYS	PIX	<b>42-47,90-95</b>	7	<b>GPU3</b>	NV4	NV4	NV4	X	SYS	SYS	PIX	SYS	<b>30-35,78-83</b>	5	mlx5_0	SYS	PIX	SYS	SYS	X	SYS	SYS	SYS			mlx5_1	PIX	SYS	SYS	SYS	SYS	X	SYS	SYS			mlx5_2	SYS	SYS	SYS	PIX	SYS	SYS	X	SYS			mlx5_3	SYS	SYS	PIX	SYS	SYS	SYS	SYS	X		
	GPU0	GPU1	GPU2	GPU3	mlx5_0	mlx5_1	mlx5_2	mlx5_3	CPU Affinity	NUMA Affinity																																																																																										
<b>GPU0</b>	X	NV4	NV4	NV4	SYS	PIX	SYS	SYS	<b>18-23,66-71</b>	3																																																																																										
<b>GPU1</b>	NV4	X	NV4	NV4	PIX	SYS	SYS	SYS	<b>6-11,54-59</b>	1																																																																																										
<b>GPU2</b>	NV4	NV4	X	NV4	SYS	SYS	SYS	PIX	<b>42-47,90-95</b>	7																																																																																										
<b>GPU3</b>	NV4	NV4	NV4	X	SYS	SYS	PIX	SYS	<b>30-35,78-83</b>	5																																																																																										
mlx5_0	SYS	PIX	SYS	SYS	X	SYS	SYS	SYS																																																																																												
mlx5_1	PIX	SYS	SYS	SYS	SYS	X	SYS	SYS																																																																																												
mlx5_2	SYS	SYS	SYS	PIX	SYS	SYS	X	SYS																																																																																												
mlx5_3	SYS	SYS	PIX	SYS	SYS	SYS	SYS	X																																																																																												

# CUDA HANDS-ON: HANDLING GPU AFFINITY

/p/scratch/share/jwb-porting-2021/Multi-GPU-Programming-with-MPI/  
CUDA/exercises/task1

Run with `CUDA_VISIBLE_DEVICES=0,1,2,3`

Handle GPU affinity with `MPI_COMM_TYPE_SHARED`

Run and report the performance

Look for TODOs

```
900, 0.122885
Num GPUs: 4.
8192x8192: 1 GPU: 25.1564 s, 4 GPUs: 24.4987 s, speedup: 1.03
```

Make Targets:

run:	run jacobi with \$NP procs.
jacobi:	build jacobi bin (default)
sanitize:	run with compute-sanitizer
profile:	profile with Nsight Systems
Solution is in ../../solution1	

<https://www.open-mpi.org/doc/current/>

# CUDA HANDS-ON: APPLY DOM. DEC.

/p/scratch/share/jwb-porting-2021/Multi-GPU-Programming-with-MPI/  
CUDA/exercises/task2

Calculate first (`iy_start`) and last (`iy_end`) row to be processed by each rank

Use `MPI_Sendrecv` to handle halo updates and periodic boundary conditions

Use `MPI_Allreduce` to calculate global L2 norm

Look for TODOs

```
900, 0.122885
Num GPUs: 4.
8192x8192: 1 GPU:    5.4283 s, 4 GPUs:    5.3205 s, speedup:    1.02
```

Make Targets:

run:	run jacobi with \$NP procs.
jacobi:	build jacobi bin (default)
sanitize:	run with compute-sanitizer
profile:	profile with Nsight Systems
Solution is in ../../solution2	

<https://www.open-mpi.org/doc/current/>

# OPENACC HANDS-ON: APPLY DOMAIN DECOMPOSITION

/p/scratch/share/jwb-porting-2021/Multi-GPU-Programming-with-MPI/OpenACC/exercises/**C|FORTRAN**/task1

- Handle GPU affinity
- Run with `CUDA_VISIBLE_DEVICES=0,1,2,3`
- Halo Exchange

```
$ make  
mpicc -c -DUSE_DOUBLE -Minfo=accel -fast -acc=gpu -gpu=cc70 poisson2d.c [...]  
srun ./poisson2d  
Jacobi relaxation Calculation: 4096 x 4096 mesh  
[...]  
Num GPUs: 2.  
4096x4096: 1 GPU: 1.3159 s, 2 GPUs: 1.3446 s, speedup:  
MPI time: 0.0000 s, inter GPU BW: 2484.22 GiB/s
```

Look for TODOs

**Make Targets:**  
run: run poisson2d (default)  
poisson2d: build poisson2d binary  
profile: profile with Nsight Systems  
\*.solution: same as above with solution  
(poisson2d.solution.\*)

# THE DETAILS

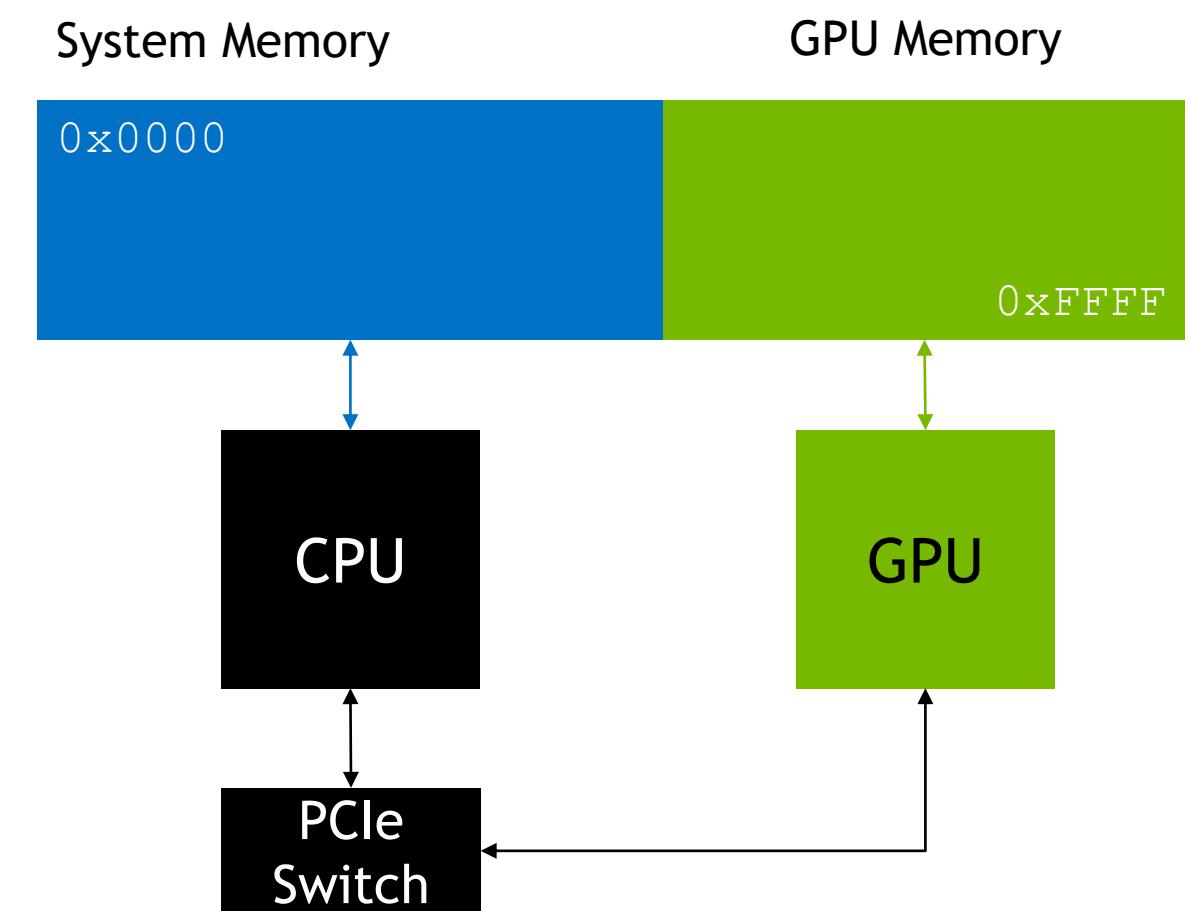
# UNIFIED VIRTUAL ADDRESSING

One address space for all CPU and GPU memory

Determine physical memory location from a pointer value

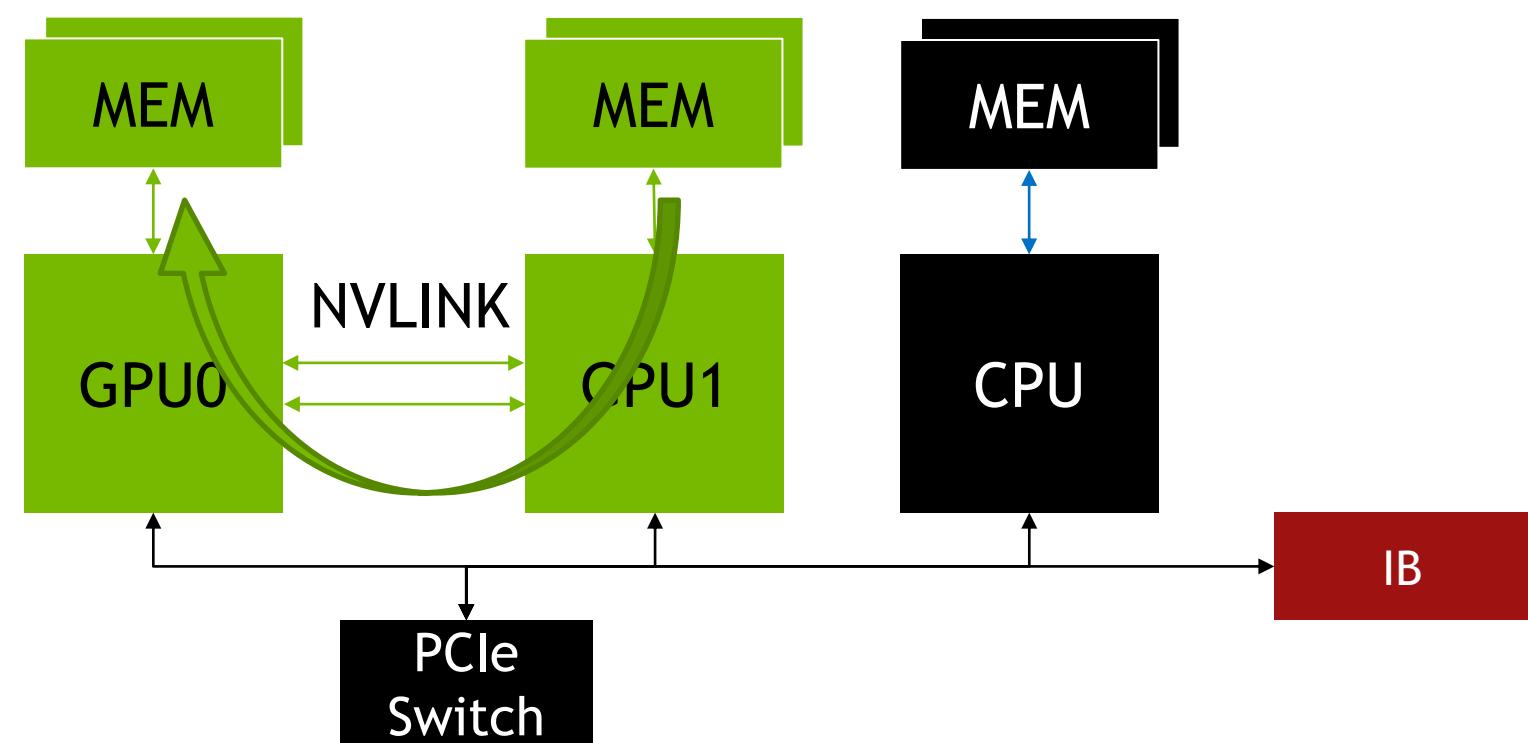
Enable libraries to simplify their interfaces (e.g. MPI and cudaMemcpy)

Supported on devices with compute capability 2.0+ for  
64-bit applications on Linux and Windows (+TCC)



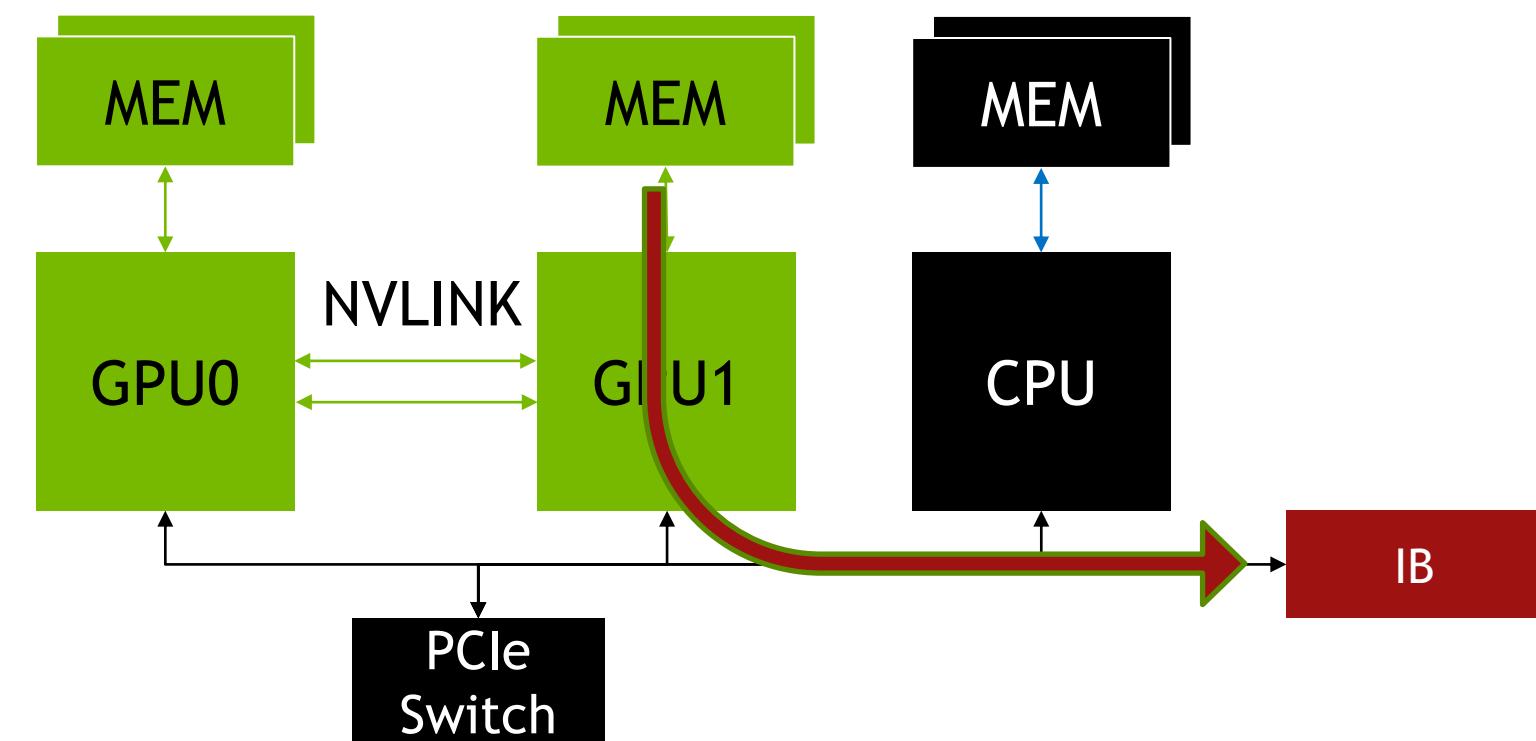
# NVIDIA GPUDIRECT

## Peer to Peer Transfers



# NVIDIA GPUDIRECT

## Support for RDMA



# CUDA-AWARE MPI

Example:

MPI Rank 0 MPI\_Send from GPU Buffer

MPI Rank 1 MPI\_Recv to GPU Buffer

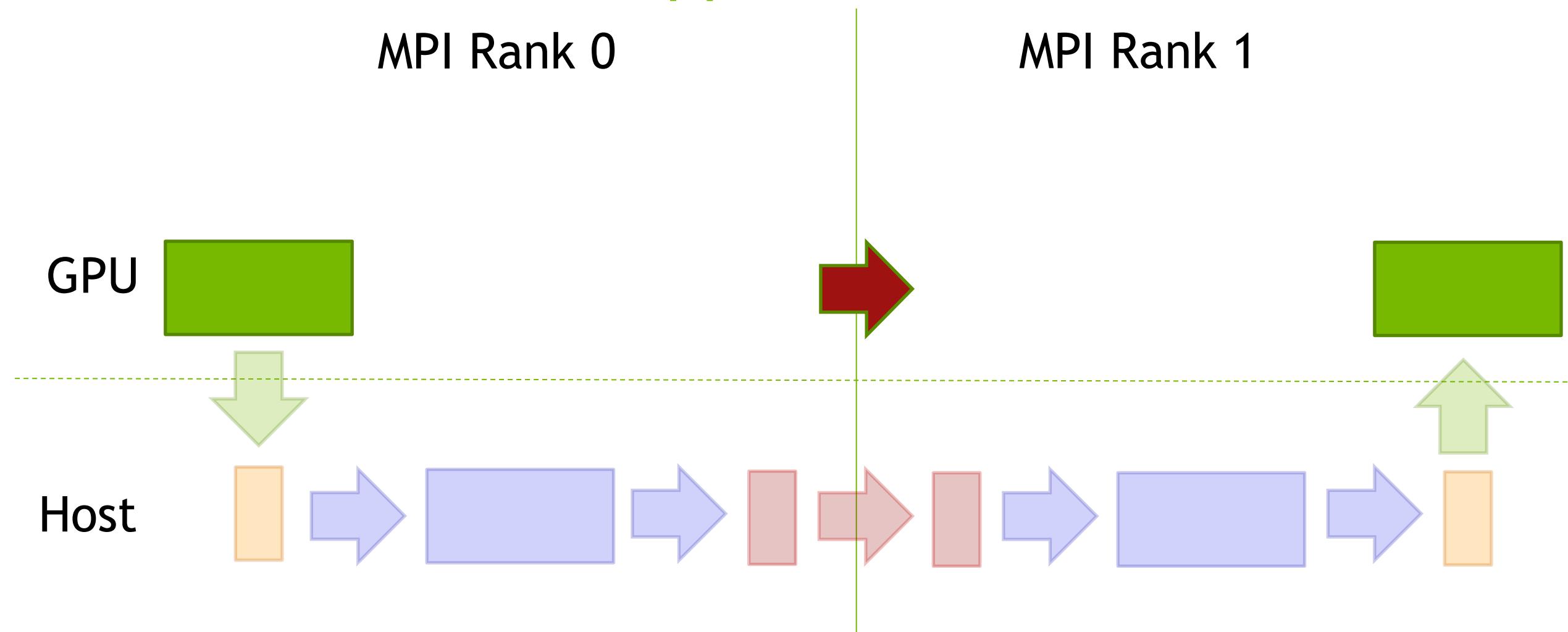
Show how CUDA+MPI works in principle

Depending on the MPI implementation, message size, system setup, ... situation might be different

Two GPUs in two nodes

# MPI GPU TO REMOTE GPU

## Support for RDMA

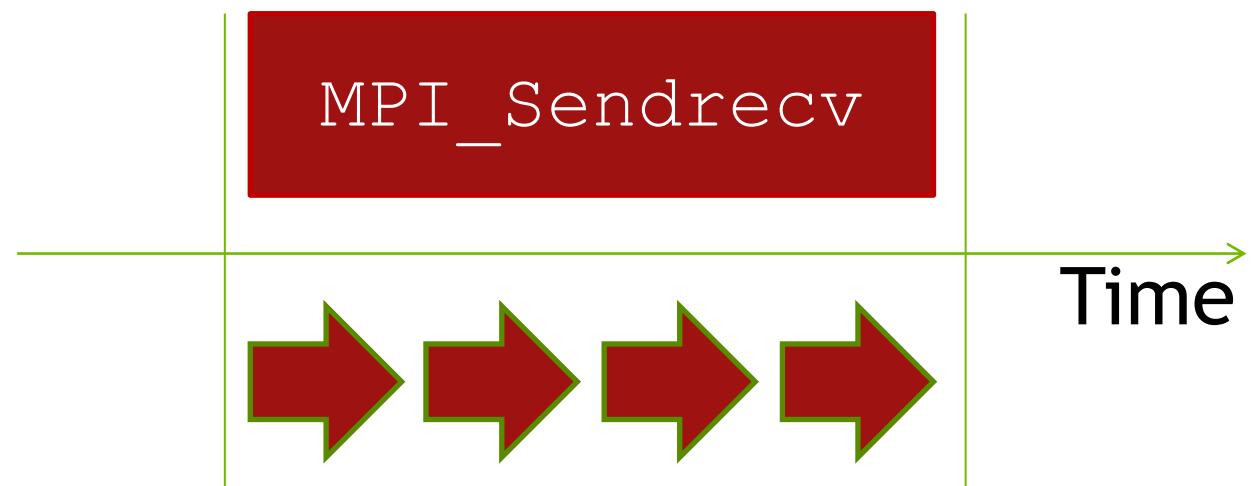


```
MPI_Send(s_buf_d, size, MPI_BYTE, 1, tag, MPI_COMM_WORLD);
```

```
MPI_Recv(r_buf_d, size, MPI_BYTE, 0, tag, MPI_COMM_WORLD, &stat);
```

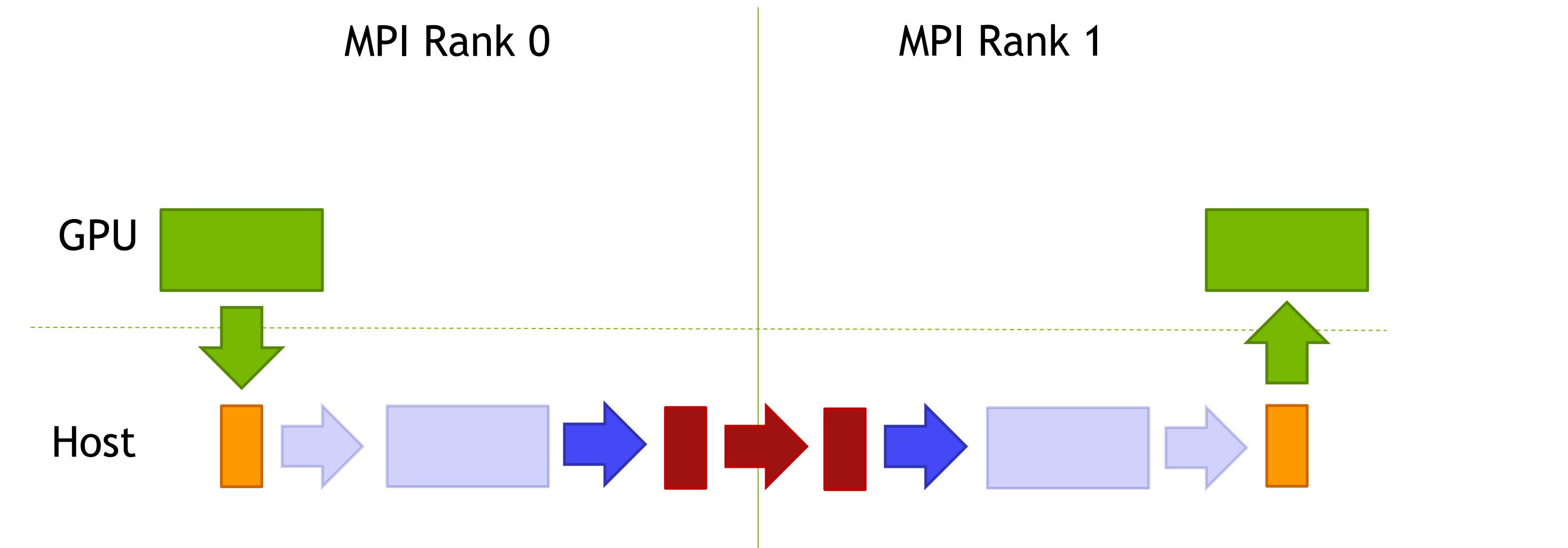
# MPI GPU TO REMOTE GPU

## Support for RDMA



# MPI GPU TO REMOTE GPU

## without GPUDirect

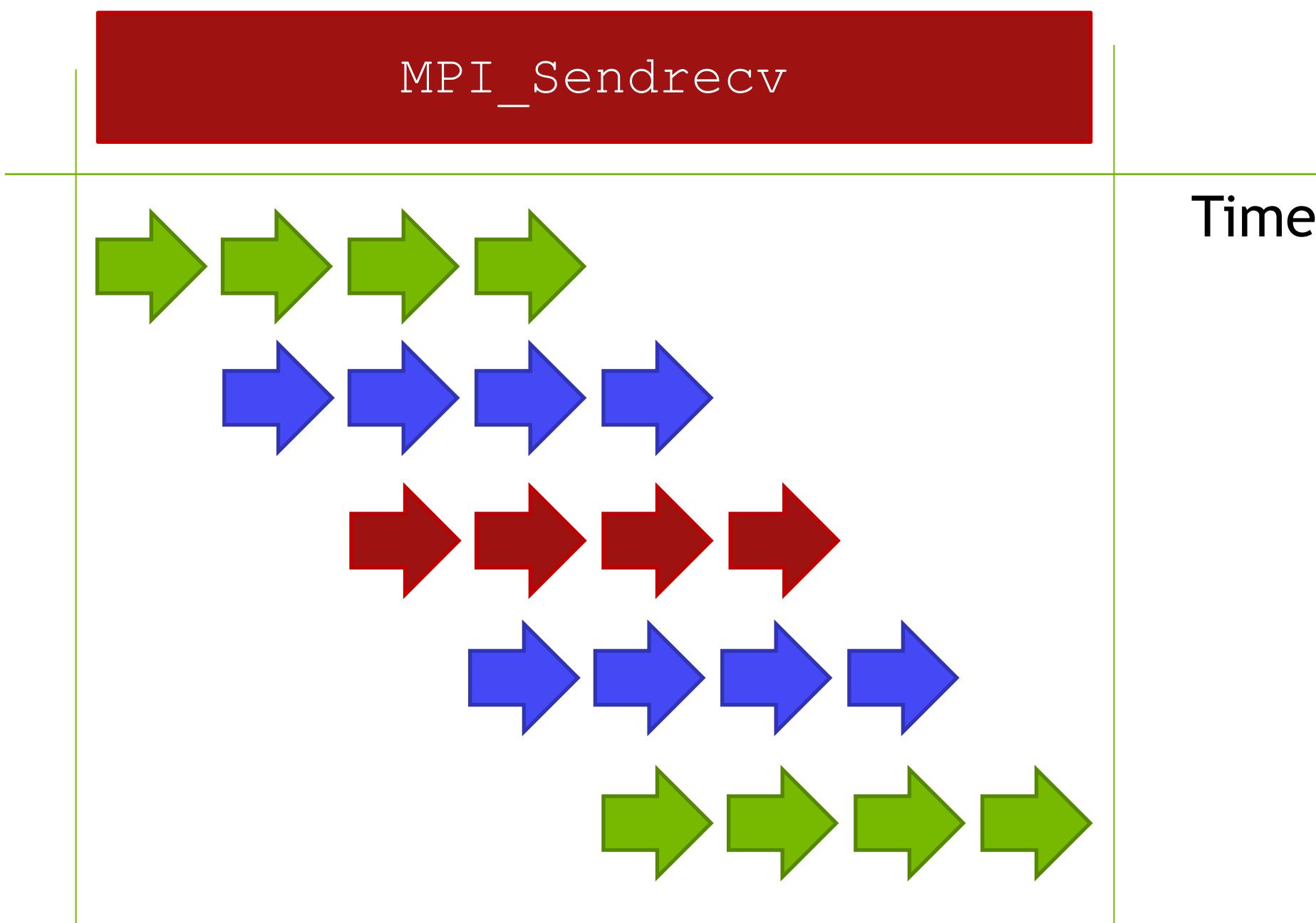


```
MPI_Send(s_buf_h, size, MPI_BYTE, 1, tag, MPI_COMM_WORLD);
```

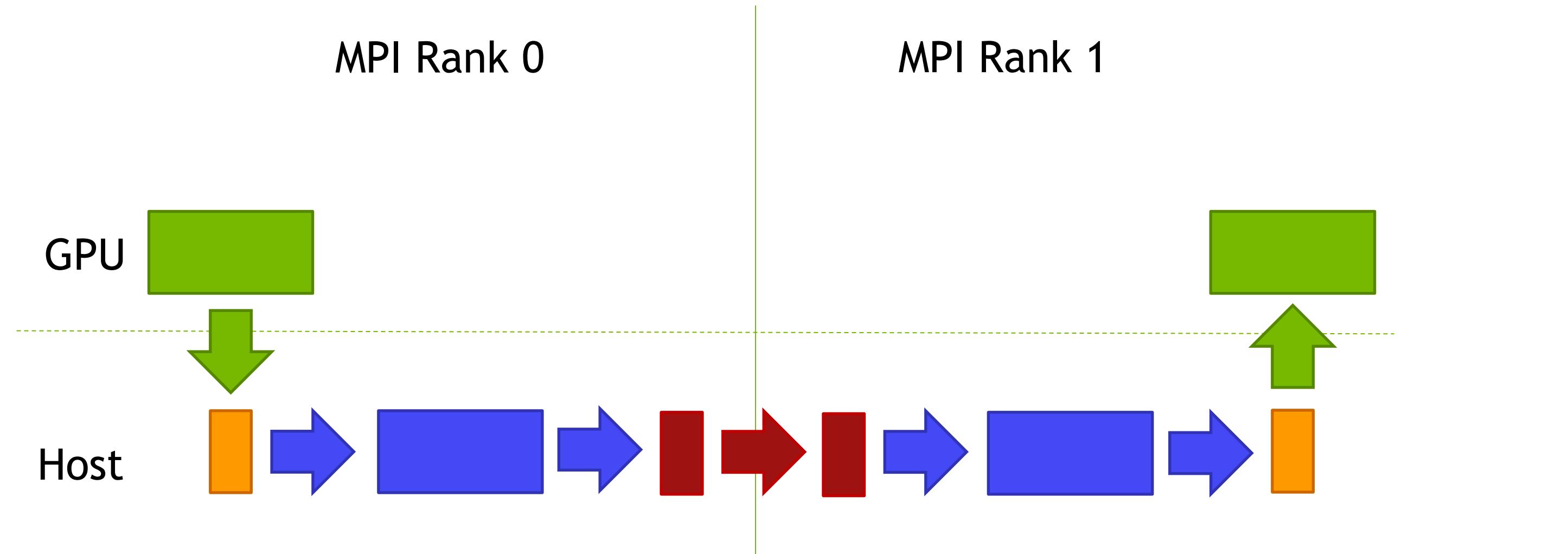
```
MPI_Recv(r_buf_h, size, MPI_BYTE, 0, tag, MPI_COMM_WORLD, &stat);
```

# MPI GPU TO REMOTE GPU

## without GPUDirect



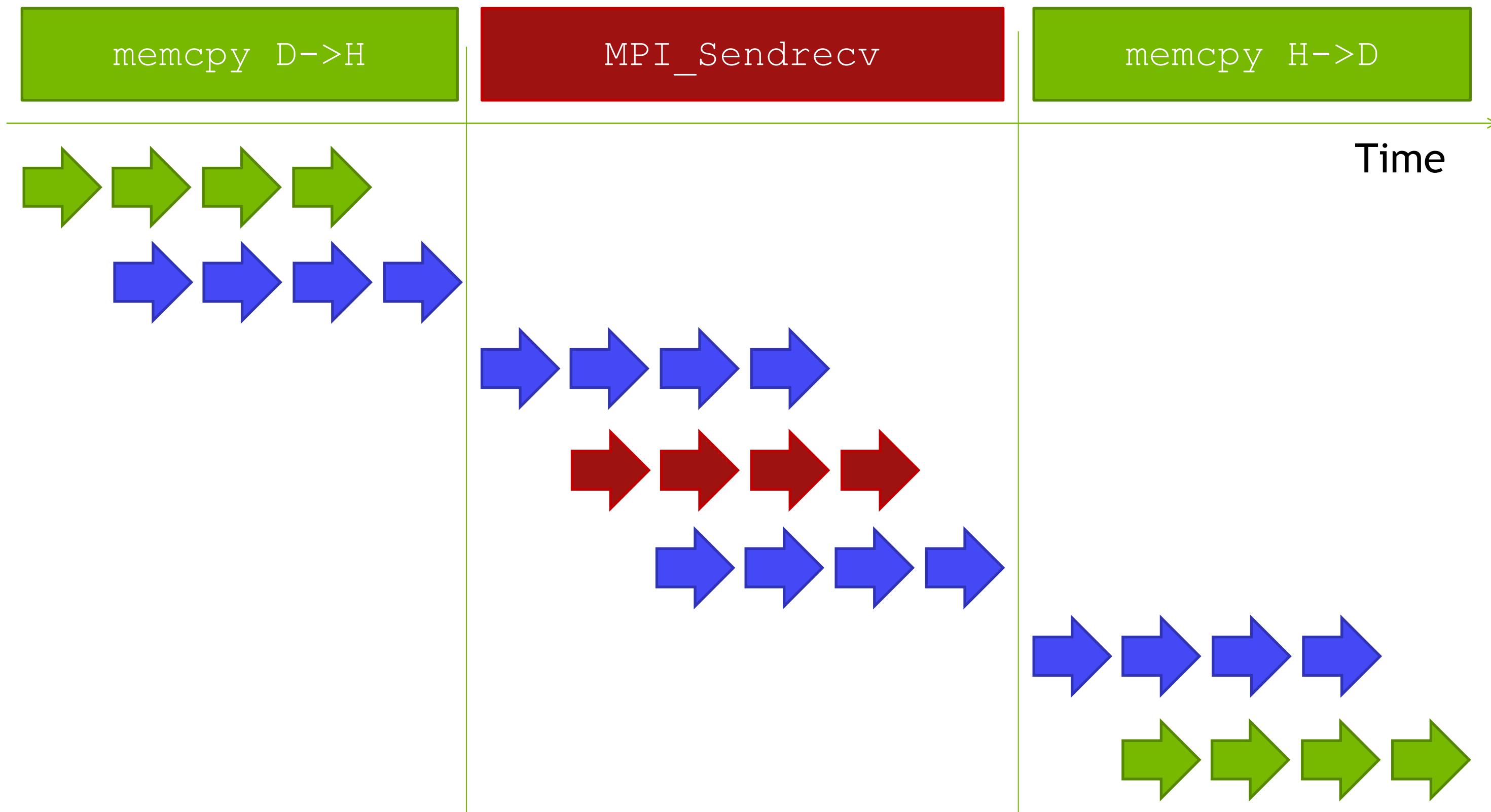
# REGULAR MPI GPU TO REMOTE GPU



```
cudaMemcpy(s_buf_h,s_buf_d,size,cudaMemcpyDeviceToHost);
MPI_Send(s_buf_h,size,MPI_BYTE,1,tag,MPI_COMM_WORLD);

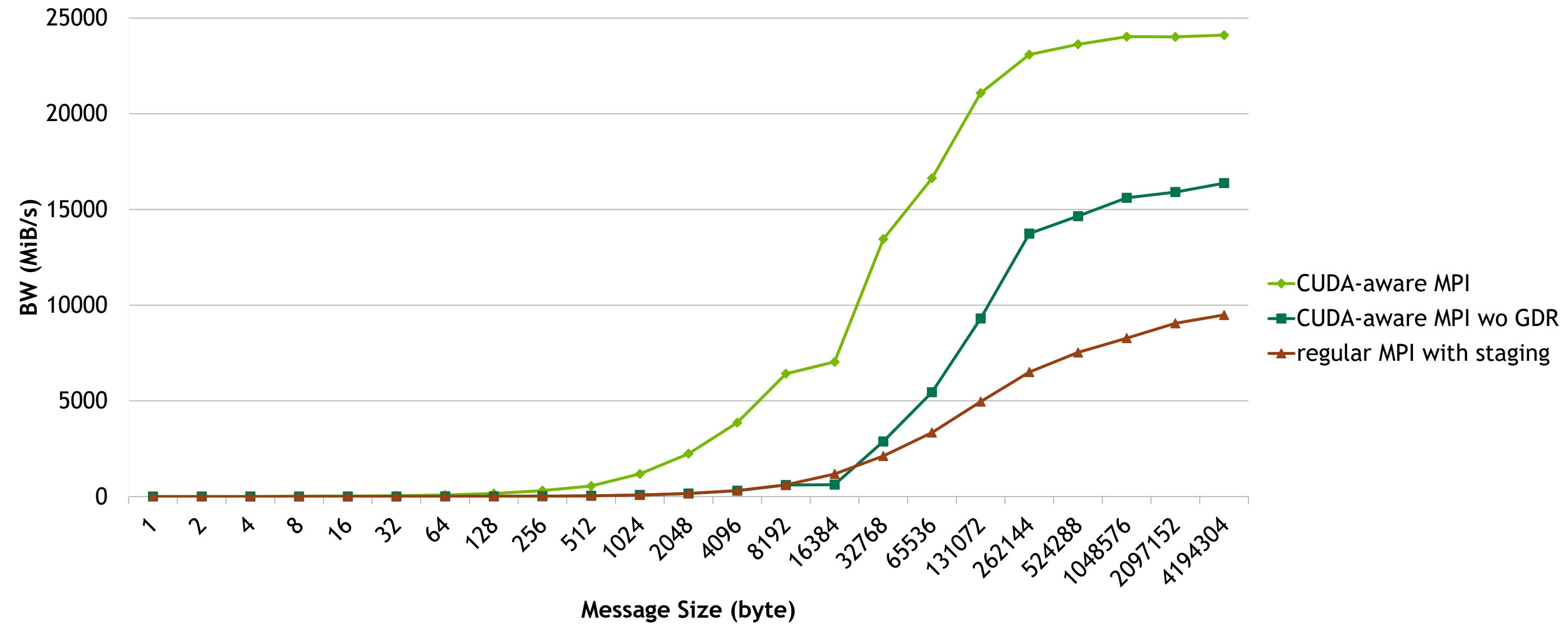
MPI_Recv(r_buf_h,size,MPI_BYTE,0,tag,MPI_COMM_WORLD,&stat);
cudaMemcpy(r_buf_d,r_buf_h,size,cudaMemcpyHostToDevice);
```

# REGULAR MPI GPU TO REMOTE GPU



# PERFORMANCE RESULTS GPUDIRECT RDMA

## OpenMPI 4.1.0RC1 + UCX 1.9.0 on JUWELS-Booster

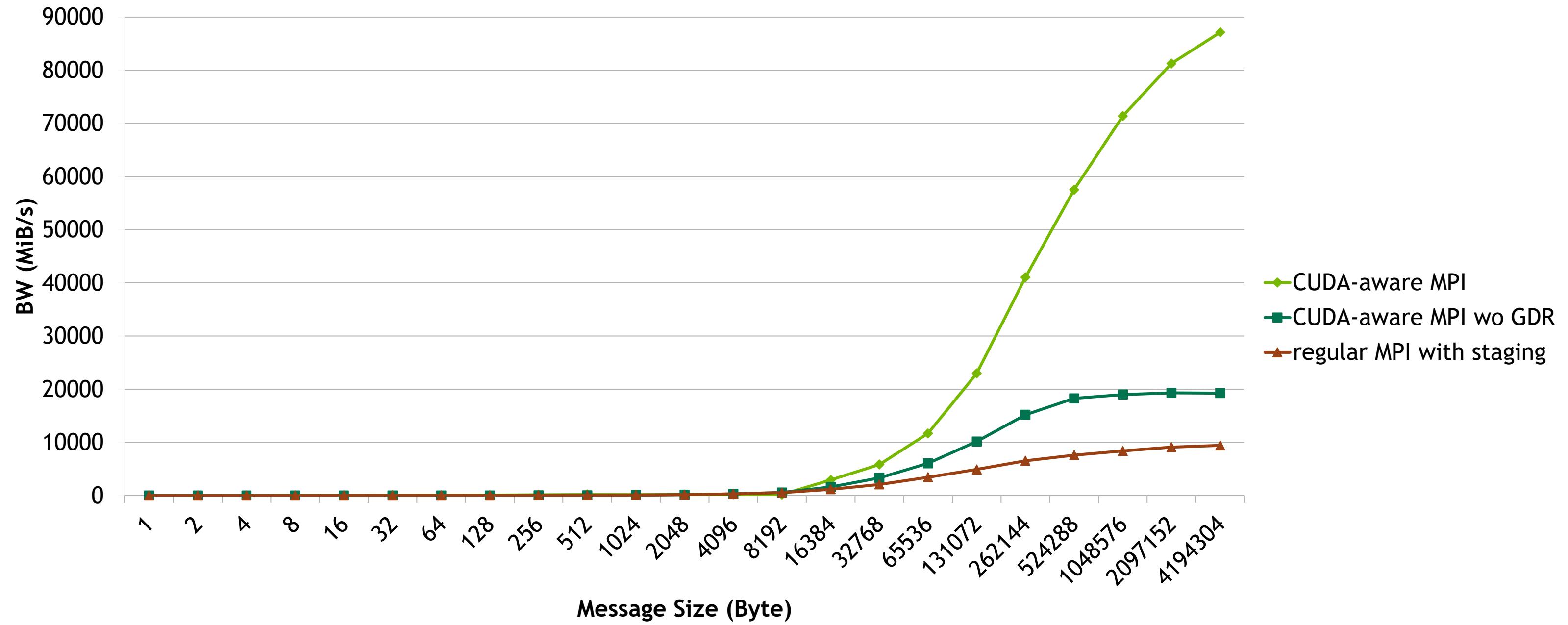


Latency (1 byte)    4.27 us    24.56 us    25.64 us

JUWELS: <http://fz-juelich.de/ias/jsc/juwels>

# PERFORMANCE RESULTS GPUDIRECT P2P

## OpenMPI 4.1.0RC1 + UCX 1.9.0 on JUWELS-Booster



Latency (1 byte)    2.45 us    22.01 us    23.50 us

JUWELS: <http://fz-juelich.de/ias/jsc/juwels>

# UCX TIPS AND TRICKS

Check setting and knobs with `ucx_info`

```
$ ucx_info -caf | grep -B9 UCX_RNDV_SCHEME
#
# Communication scheme in RNDV protocol.
# get_zcopy - use get_zcopy scheme in RNDV protocol.
# put_zcopy - use put_zcopy scheme in RNDV protocol.
# auto      - runtime automatically chooses optimal scheme to use.
#
#
# syntax: [get_zcopy|put_zcopy|auto]
#
UCX_RNDV_SCHEME=auto
```

# UCX TIPS AND TRICKS

Enable logging to see what is going on

`UCX_LOG_LEVEL=data UCX_LOG_FILE=log-%h-%p` helpful to check for used protocols and selected HCAs:

```
[1605706306.970537] [jwb1238:7263 :0]    ucp_worker.c:1627 UCX  INFO  ep_cfg[0]: tag(cuda_copy/cuda); rma(gdr_copy/cuda);  
[1605706306.972721] [jwb1238:7263 :0]    ucp_worker.c:1627 UCX  INFO  ep_cfg[1]: tag(self/memory rc_mlx5/mlx5_1:1 cma/memory cuda_copy/cuda);  
[1605706306.997849] [jwb1238:7263 :1]    ucp_worker.c:1627 UCX  INFO  ep_cfg[2]: tag(rc_mlx5/mlx5_1:1);
```

# UCX TIPS AND TRICKS

<https://github.com/openucx/ucx/wiki/UCX-environment-parameters>

**UCX\_NET\_DEVICES**: To select HCA for optimal GPU-HCA affinity, should not be necessary with UCX 1.9 or newer

**UCX\_MAX\_RNDV\_RAILS**: Defaults to 2, negatively impacting large message perf if UCX\_NET\_DEVICES is not used to mask out HCAs.

**UCX\_TLS**: Select transports to use, default: all

cuda is an alias for: cuda\_copy, cuda\_ipc, gdr\_copy

To run without any GPUDirect flavor set UCX\_TLS to only include cuda\_copy, e.g. UCX\_TLS=rc,sm,cuda\_copy and UCX\_IB\_GPU\_DIRECT\_RDMA=no (rc transport uses GPUDirect RDMA otherwise).

Parastation MPI also has PSP\_CUDA\_ENFORCE\_STAGING=1.

Up to UCX 1.8 using sm (alias for mm, cma, knem, xpmem) effectively disabled cuda\_ipc. This is fixed with UCX 1.9 so it is recommended to include sm with UCX 1.9 or later.

**UCX\_MEMTYPE\_CACHE**: Set to n to disable mem type cache. Often necessary if the CUDA runtime is linked statically!

**UCX\_IB\_PCI\_RELAXED\_ORDERING**: On UCX 1.9.0 set to on

see <https://hpcadvisorycouncil.atlassian.net/wiki/spaces/HPCWORKS/pages/1280442391/AMD+2nd+Gen+EPYC+CPU+Tuning+Guide+for+InfiniB+and+HPC#Relaxed-Ordering>

# HANDS-ON: OSU MICROBENCHMARKS

Download build and run the OSU Microbenchmarks:

<http://mvapich.cse.ohio-state.edu/benchmarks/>

Configure:

```
../configure CC=`which mpicc` CXX=`which mpicxx` --prefix=<prefix>  
--enable-cuda --with-cuda=$CUDA_HOME
```

Benchmarks used in this presentation:

```
srun mpi/pt2pt/osu_bw D D
```

```
srun mpi/pt2pt/osu_latency D D
```

# COLLECTIVES

# HIERARCHICAL COMMUNICATION ALGORITHMS LIBRARY

## HCOLL

Library for software

Hierarchical COmmunication aLgorithms (HCOL) - CPU and GPU data

NCCL - GPU data\*

and hardware

Scalable Hierarchical Aggregation and Reduction Protocol (SHARP - in Network/Switch)

Collectives Offload Resource Engine (CORE-Direct - HCA offloading)

accelerated collectives.

To be evolved into the UCF project Unified Collective Communications (UCC): <https://github.com/openucx/ucc>

Used by HPC-X, other MPI implementations can link `libhcoll.so`, e.g. OpenMPI, Parastation MPI (in internal QA)

HCOLL predecessor was Fabric Collective Accelerator (FCA)

\*With MPI depending on the used thread mode and if blocking or non-blocking collectives are used limitations apply.

# NCCL ACCELERATED MPI COLLECTIVES

## Limitations

NCCL accelerated collective do not work with MPS

Thread Mode	Blocking Collectives	Non-blocking Collectives
thread-single	Any communicator	COMM_WORLD only
thread-funneled	Any communicator	COMM_WORLD only
thread-serialized	Any communicator	COMM_WORLD only
thread-multiple	COMM_WORLD only	COMM_WORLD only

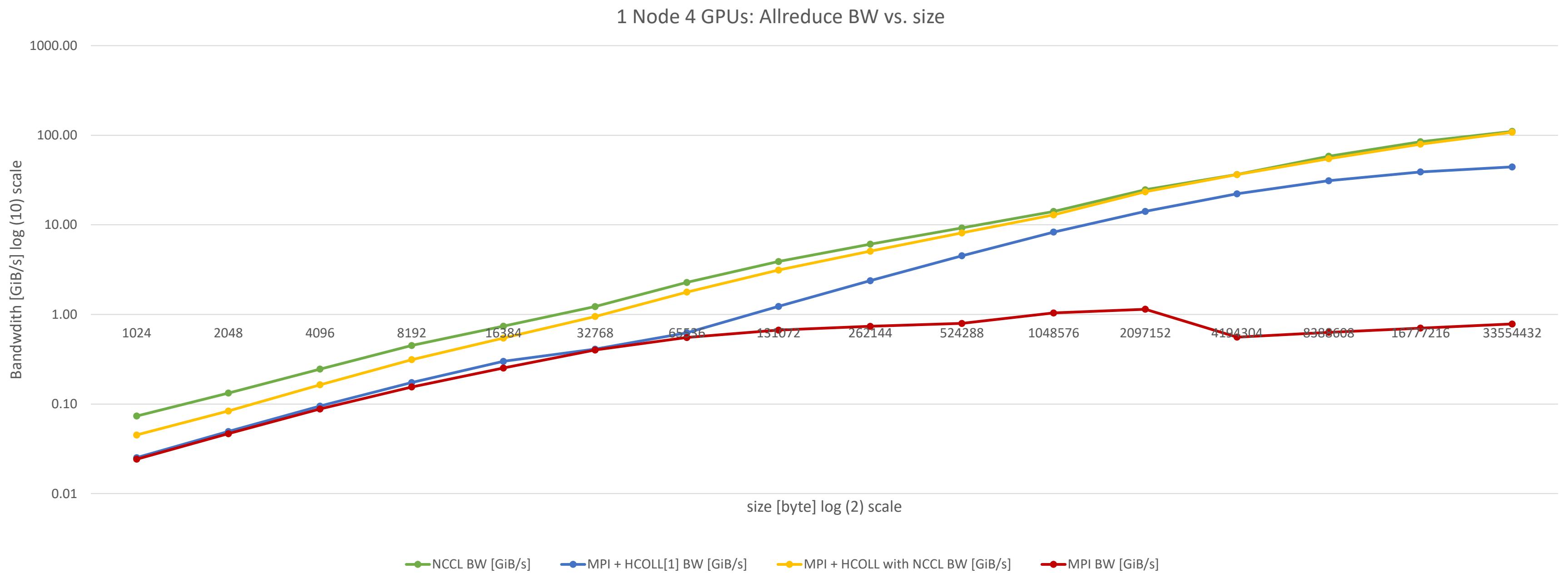
Limitations apply to HCOLL and similar limitations will apply to UCC!

HCOLL NCCL backend requires opt-in by setting: `HCOLL_CUDA_BCOL=nccl`

**CAVEAT:** HCOLL does not enforce these restrictions so apps may run into deadlocks if they opt into the NCCL backend and use an unsupported combination!

# ACCELERATED MPI COLLECTIVES

HPC-X 2.7 + UCX 1.9.0 + NCCL 2.7.8 on JUWELS-Booster



# MULTI PROCESS SERVICE (MPS) FOR MPI APPLICATIONS

# GPU ACCELERATION OF LEGACY MPI APPS

Typical legacy application

- MPI parallel

- Single or few threads per MPI rank (e.g. OpenMP)

Running with multiple MPI ranks per node

GPU acceleration in phases

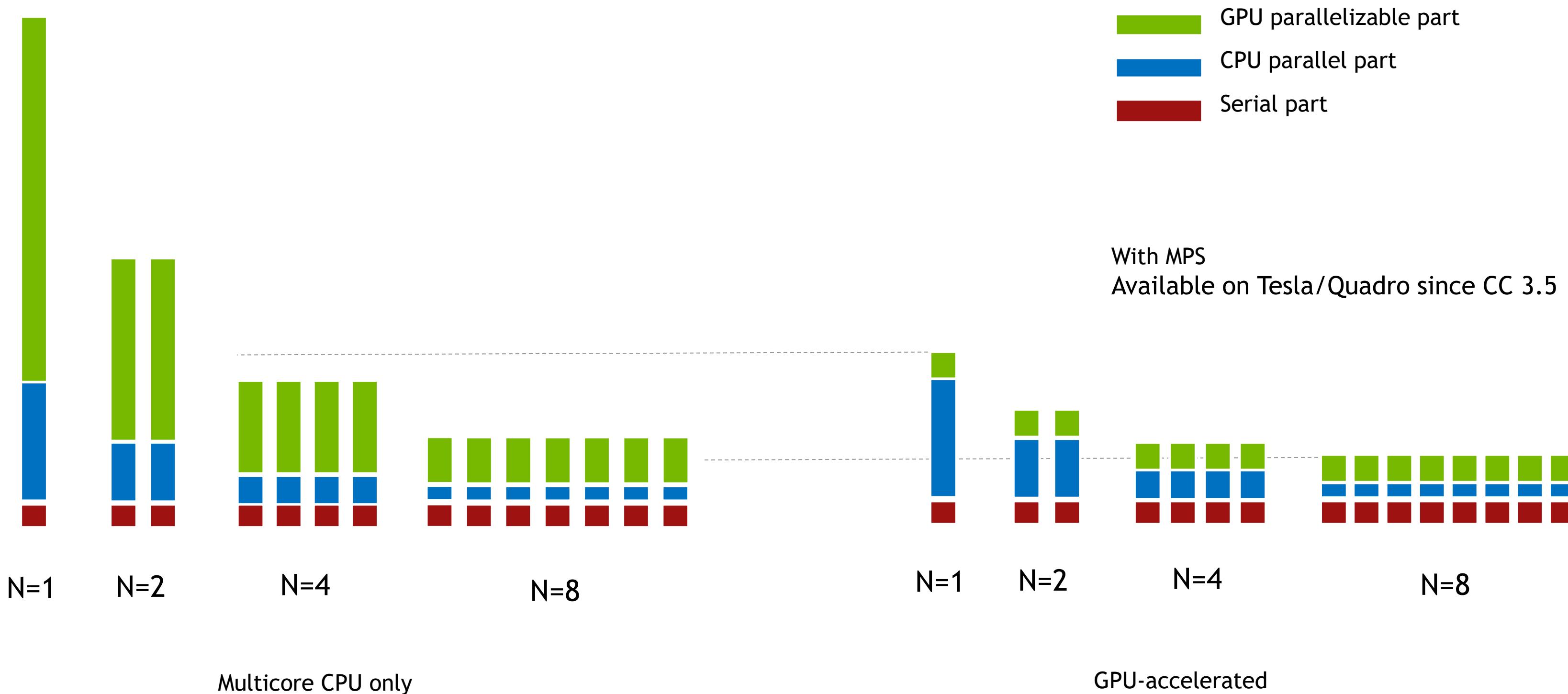
- Proof of concept prototype, ...

- Great speedup at kernel level

Application performance misses expectations

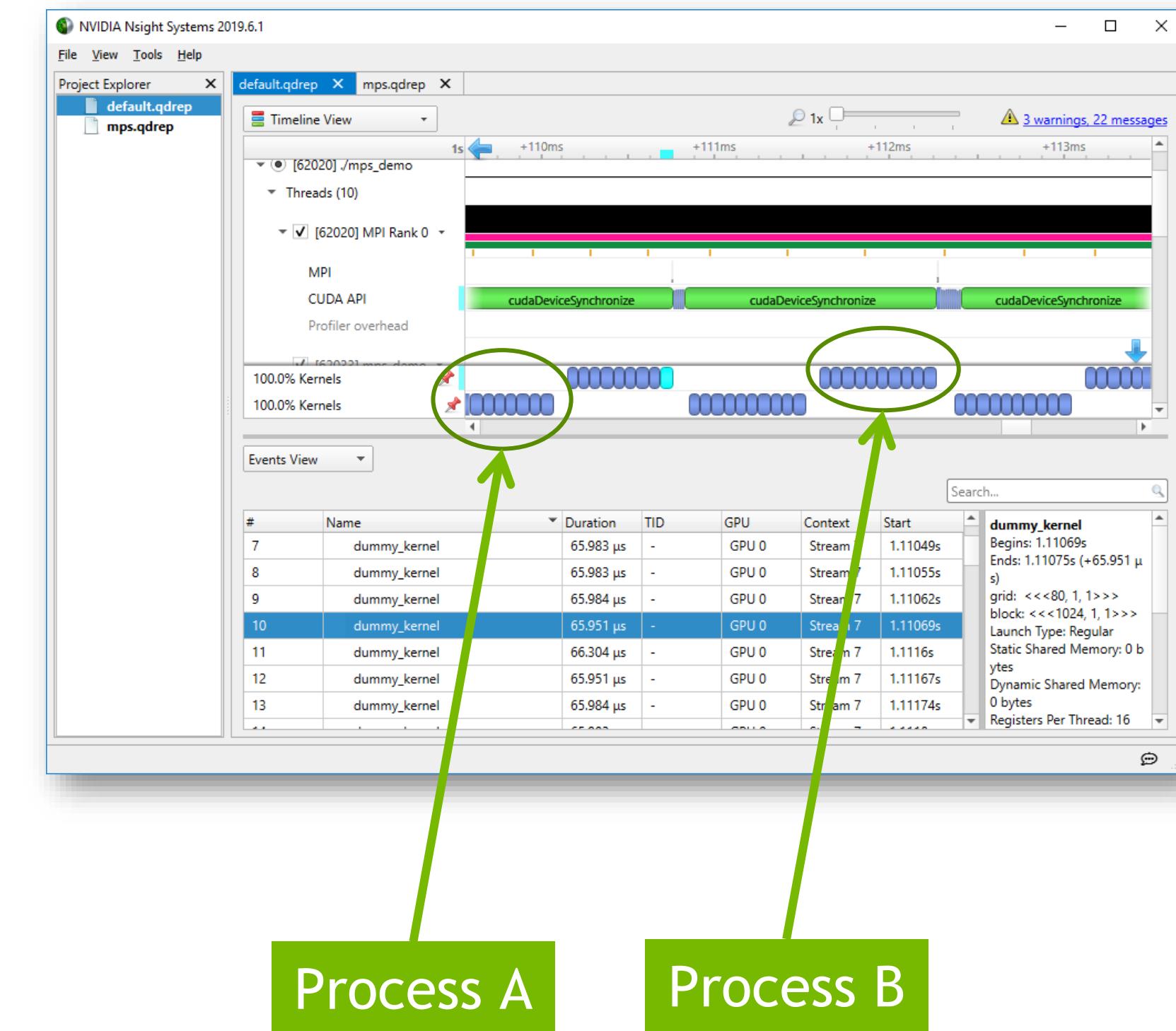
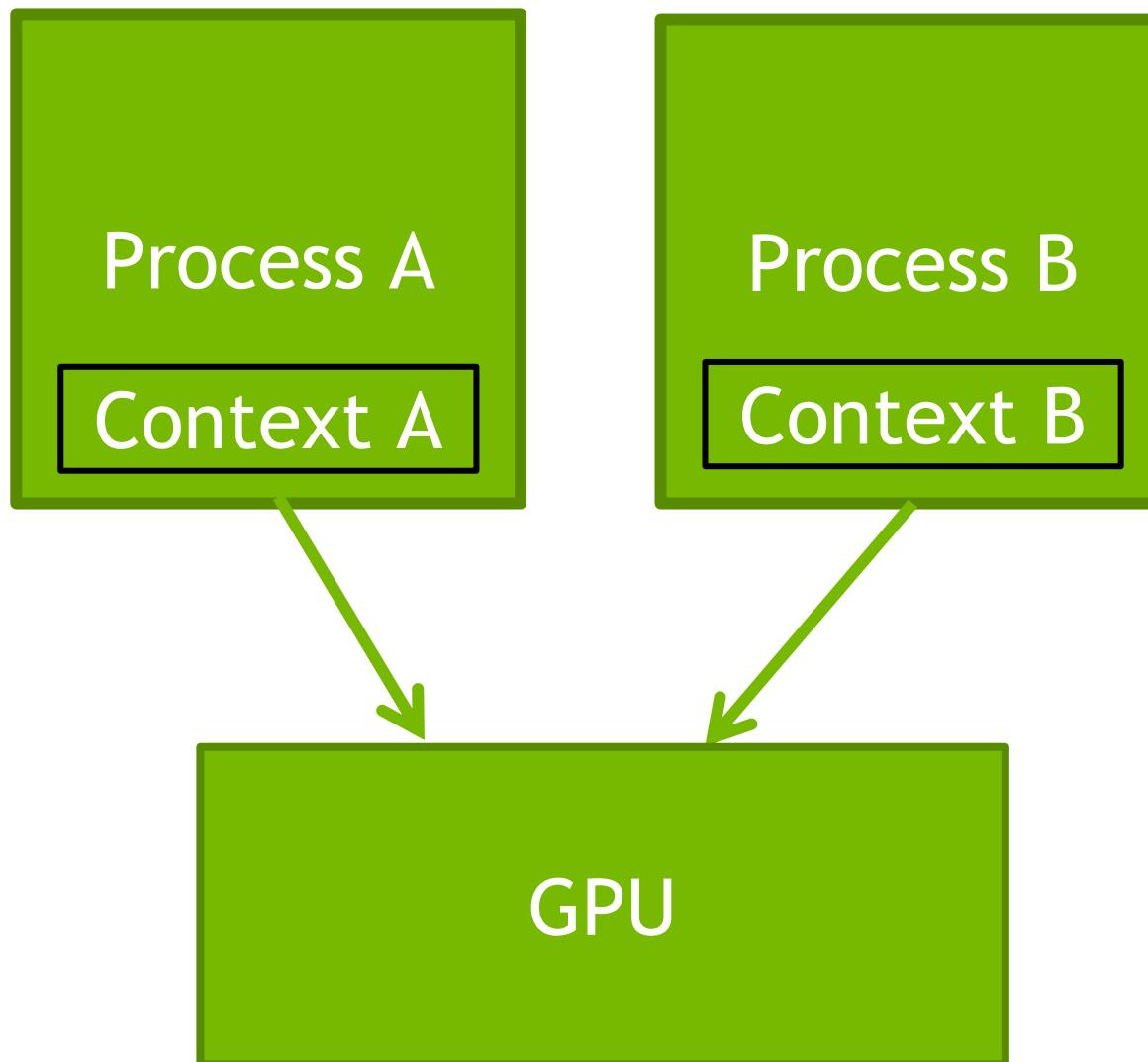
# MULTI PROCESS SERVICE (MPS)

## For Legacy MPI Applications



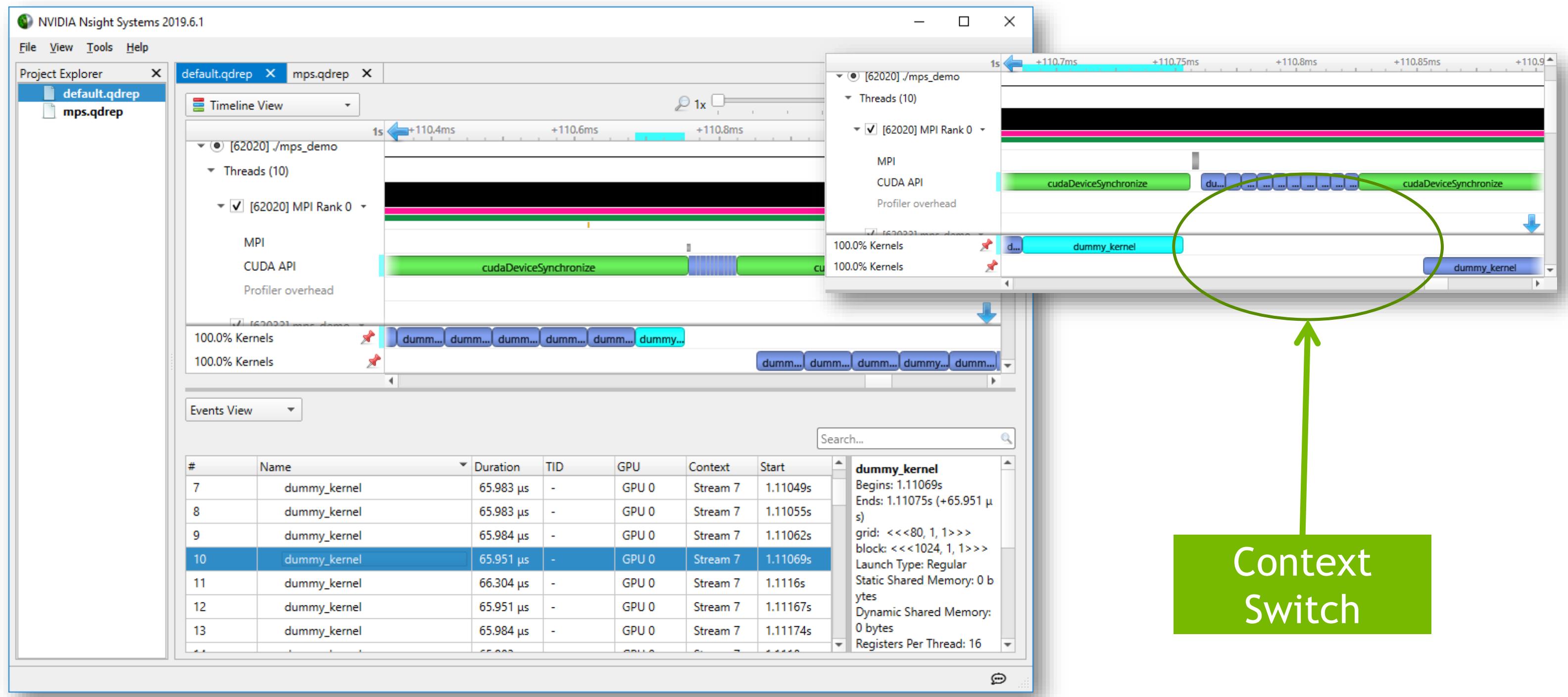
# PROCESSES SHARING GPU WITHOUT MPS

## No Overlap



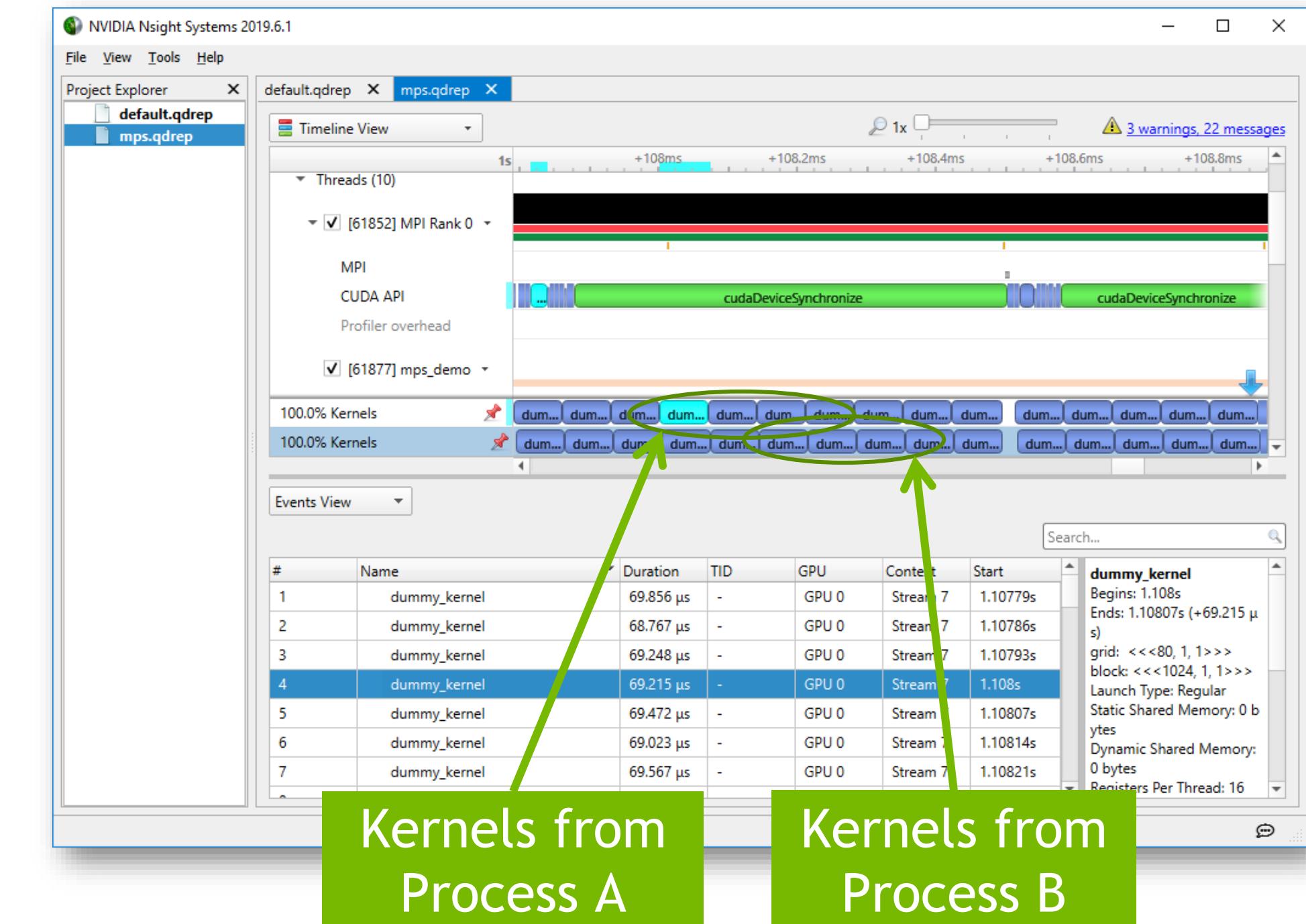
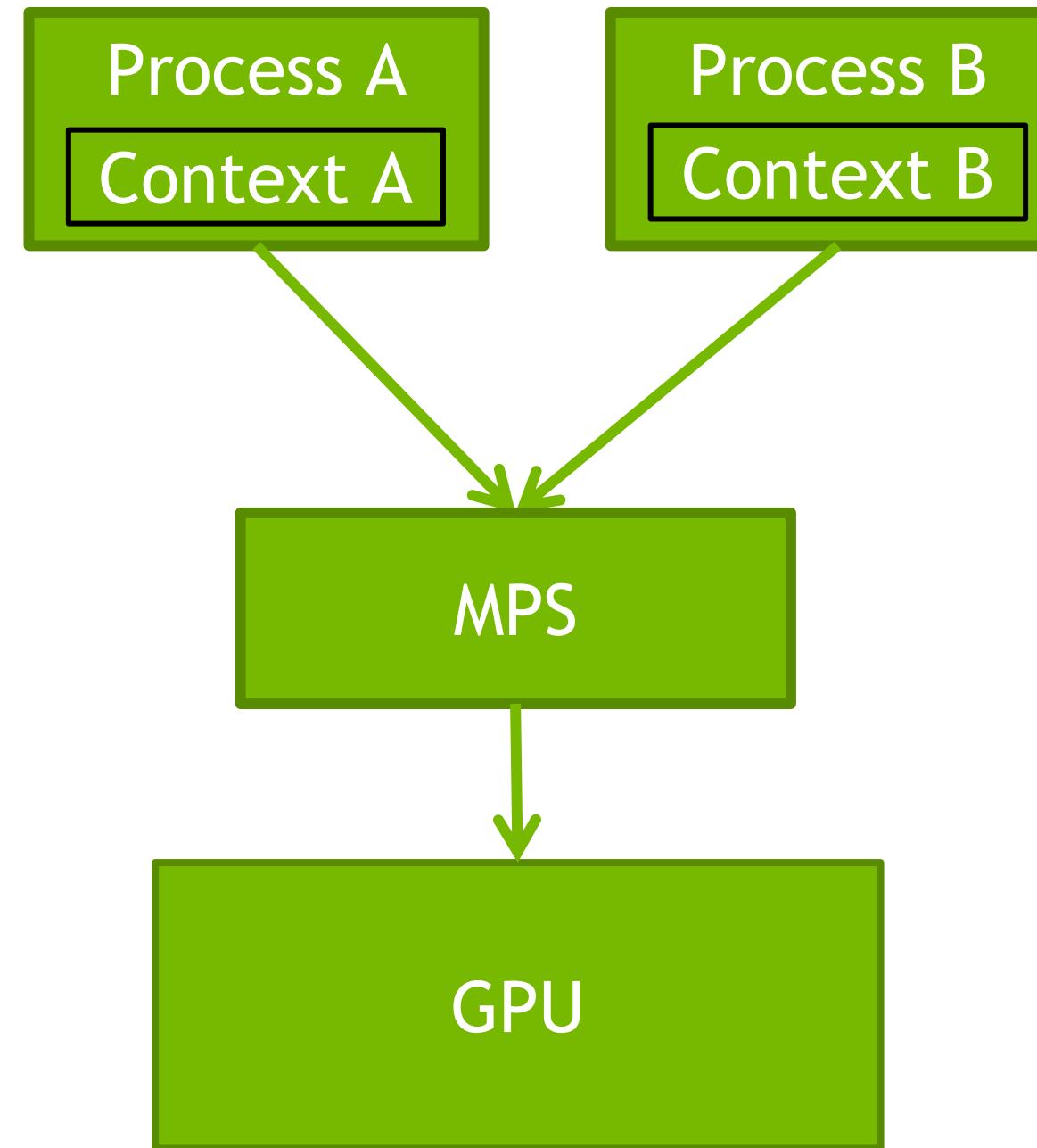
# PROCESSES SHARING GPU WITHOUT MPS

## Context Switch Overhead



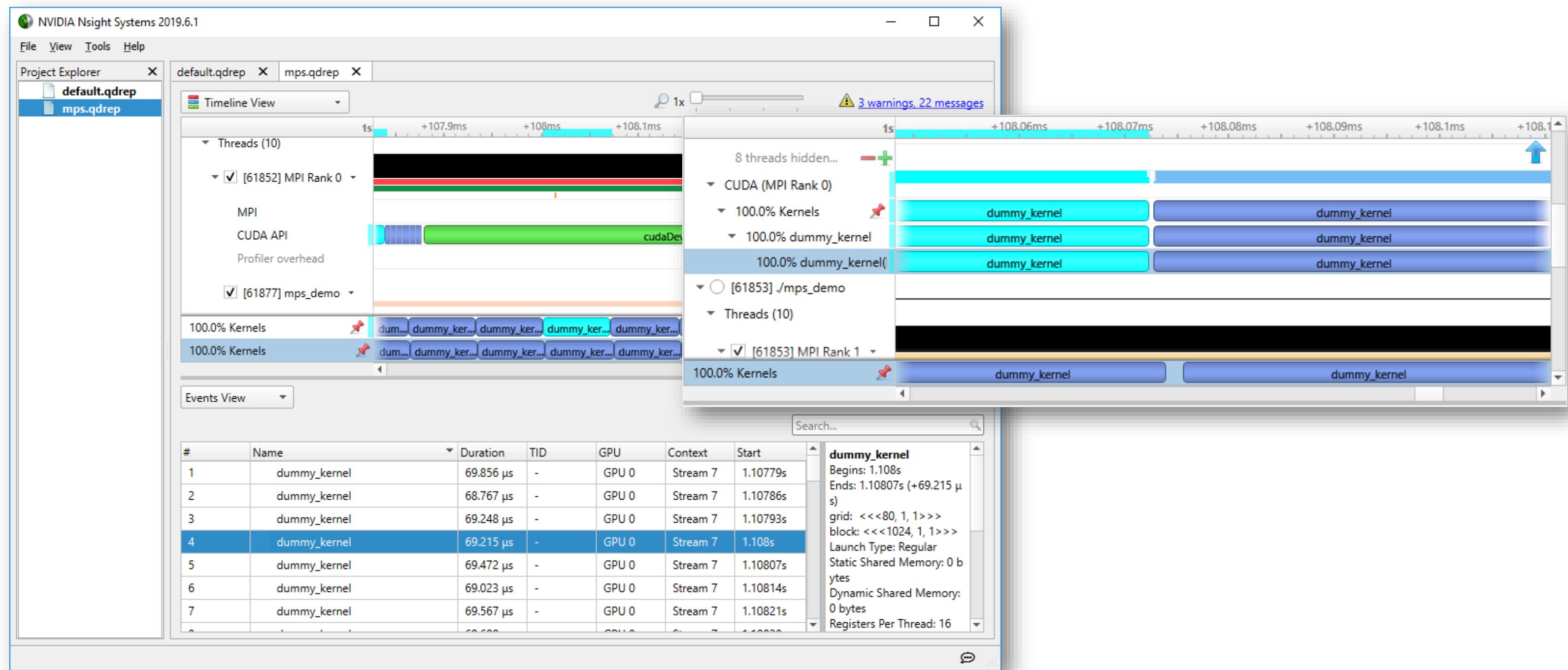
# PROCESSES SHARING GPU WITH MPS

## Maximum Overlap

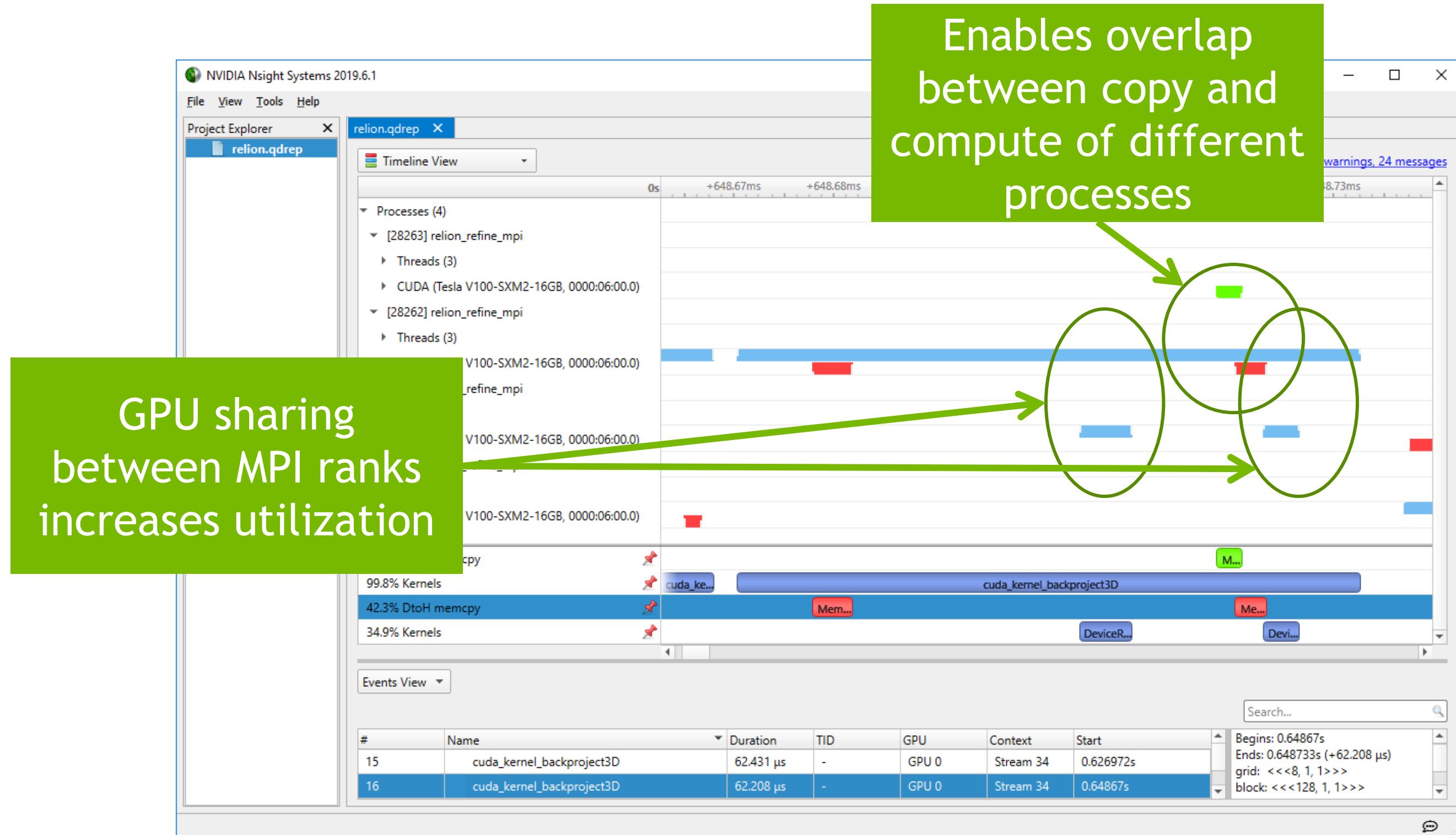


# PROCESSES SHARING GPU WITH MPS

## No Context Switch Overhead



# MPS CASE STUDY: RELION



# USING MPS

No application modifications necessary

Not limited to MPI applications

MPS control daemon

Spawn MPS server upon CUDA application startup

#Manually

```
nvidia-smi -c EXCLUSIVE_PROCESS
```

```
nvidia-cuda-mps-control -d
```

#JUWELS-Booster

```
sbatch --cuda-mps
```

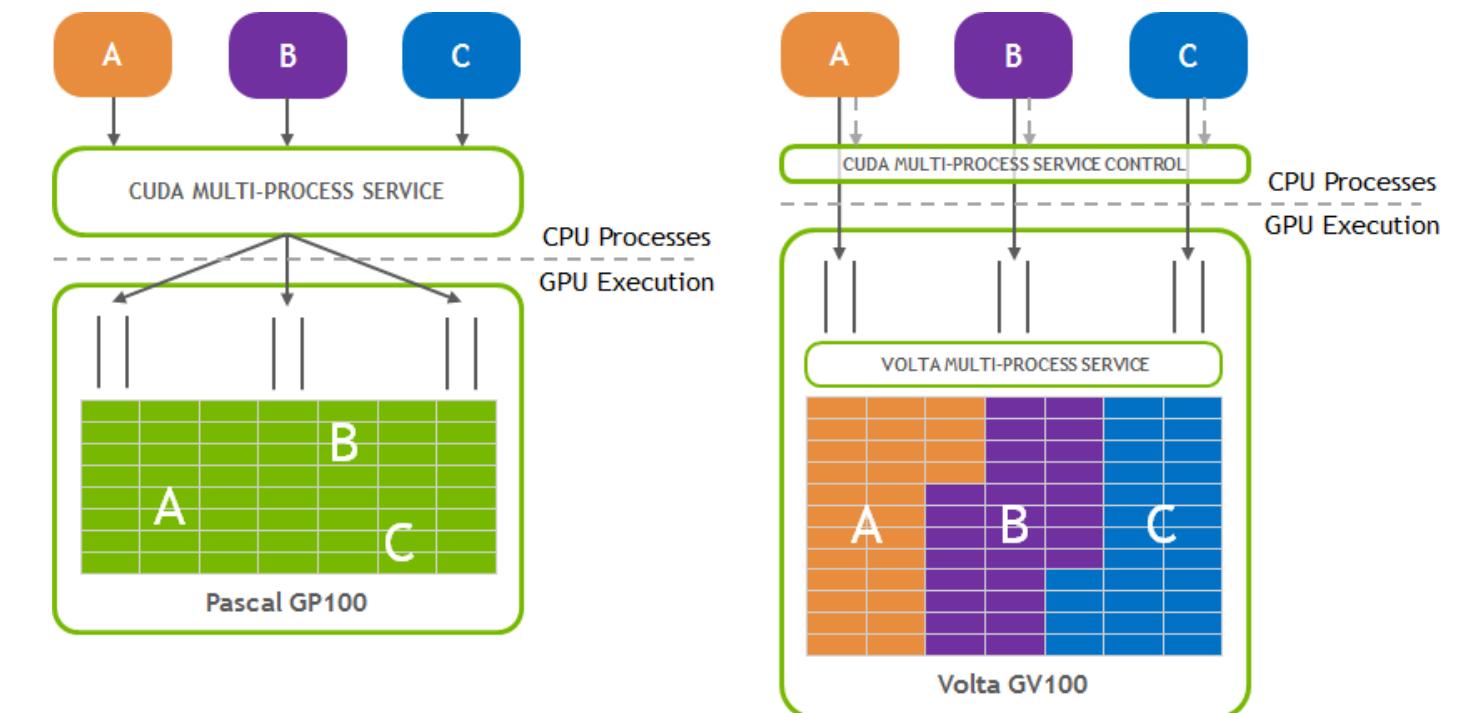
# MPS: IMPROVEMENTS WITH VOLTA

**More MPS clients per GPU:** 48 instead of 16

**Less overhead:** Volta MPS clients submit work directly to the GPU without passing through the MPS server.

**More security:** Each Volta MPS client owns its own GPU address space instead of sharing GPU address space with all other MPS clients.

**More control:** Volta MPS supports limited execution resource provisioning for Quality of Service (QoS). ->  
CUDA\_MPS\_ACTIVE\_THREAD\_PERCENTAGE

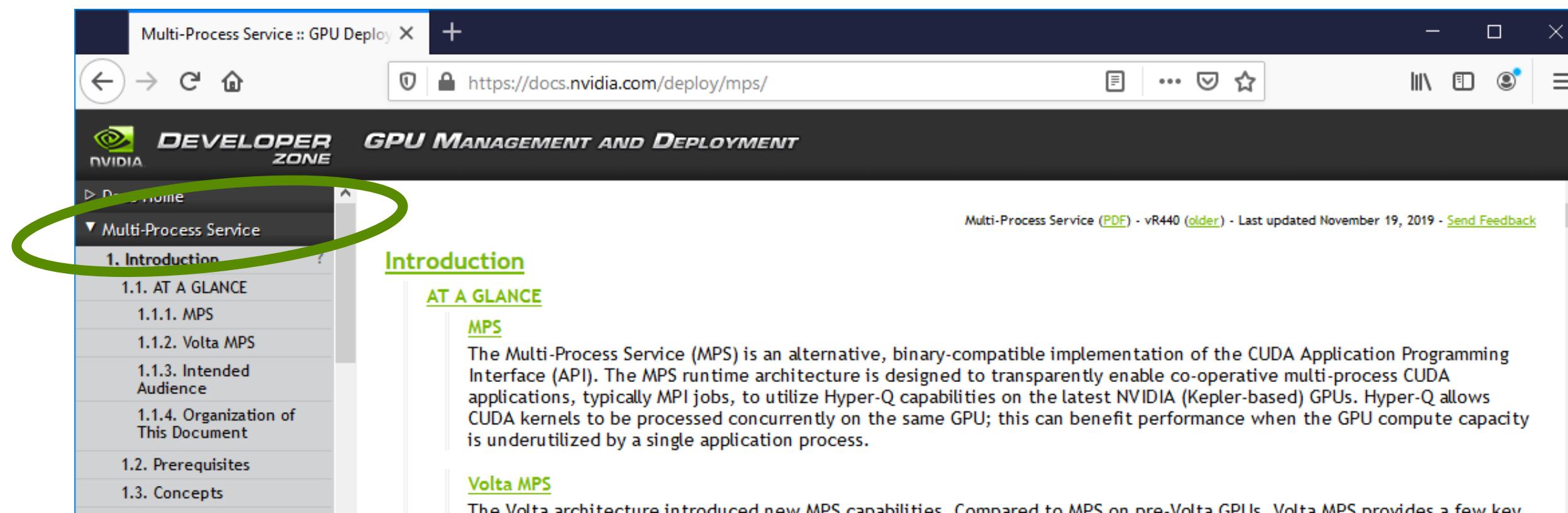


# MPS SUMMARY

Easy path to get GPU acceleration for legacy applications

Enables overlapping of memory copies and compute between different MPI ranks

Remark: MPS adds some overhead!



# DEBUGGING AND PROFILING

# TOOLS FOR MPI+CUDA APPLICATIONS

Memory checking: compute-sanitizer

Debugging: cuda-gdb

Profiling: NVIDIA Nsight Systems

# ERROR CHECKING WITH COMPUTE SANITIZER

compute-sanitizer is a functional correctness checking suite comparable to Valgrind

Can be used in a MPI environment

```
mpiexec -np 2 compute-sanitizer ./myapp <args>
```

Problem: Output of different processes is interleaved

Solution: Use save or log-file command line options

```
mpirun -np 2 compute-sanitizer  
      --log-file name.%q{OMPI_COMM_WORLD_RANK}.log  
      --save name.%q{OMPI_COMM_WORLD_RANK}.compute-sanitizer  
      ./myapp <args>
```

**OpenMPI:** OMPI\_COMM\_WORLD\_RANK

**MVAPICH2:** MV2\_COMM\_WORLD\_RANK

**Slurm with PMI-2:** PMI\_RANK

**Slurm with PMIX:** PMIX\_RANK

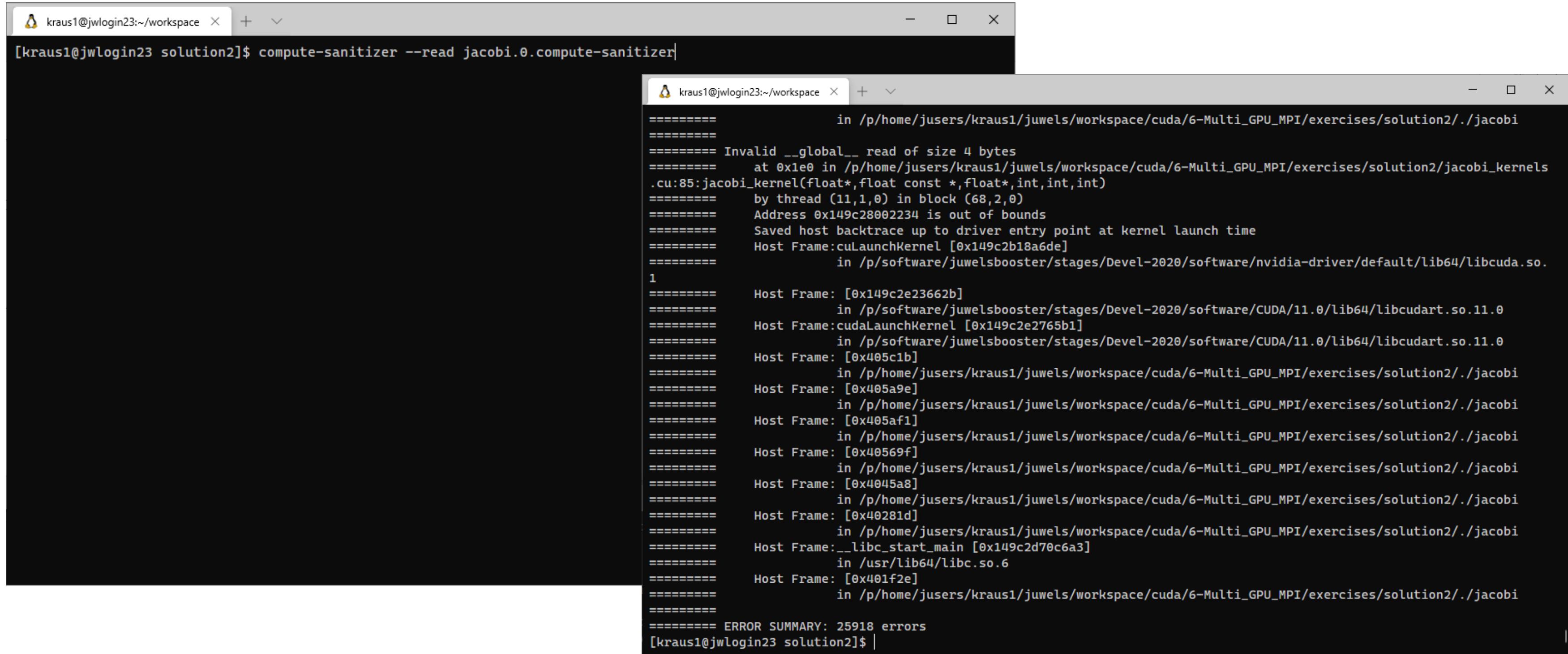
# ERROR CHECKING WITH COMPUTE SANITIZER

```
kraus1@jwlogin23:~/workspace $ srun -n 2 compute-sanitizer --log-file jacobi.%q{PMIX_RANK}.compute-sanitizer.log --save jacobi.%q{PMIX_RANK}.compute-sanitizer ./jacobi
ERROR: CUDA RT call "cudaStreamSynchronize( compute_stream )" in line 353 of file jacobi.cpp failed with unspecified launch failure (719).
ERROR: CUDA RT call "cudaMemcpy( a_ref_h, a, nx*ny*sizeof(real), cudaMemcpy )" in line 360 of file jacobi.cpp failed with unspecified launch failure (719).
ERROR: CUDA RT call "cudaEventDestroy( push_bottom_done )" in line 366 of file jacobi.cpp failed with unspecified launch failure (719).
ERROR: CUDA RT call "cudaEventDestroy( push_top_done )" in line 367 of file jacobi.cpp failed with unspecified launch failure (719).
ERROR: CUDA RT call "cudaEventDestroy( compute_done )" in line 368 of file jacobi.cpp failed with unspecified launch failure (719).
ERROR: CUDA RT call "cudaStreamDestroy( push_bottom_stream )" in line 369 of file jacobi.cpp failed with unspecified launch failure (719).
ERROR: CUDA RT call "cudaStreamDestroy( push_top_stream )" in line 370 of file jacobi.cpp failed with unspecified launch failure (719).
ERROR: CUDA RT call "cudaStreamDestroy( compute_stream )" in line 371 of file jacobi.cpp failed with unspecified launch failure (719).
ERROR: CUDA RT call "cudaFreeHost( l2_norm_h )" in line 373 of file jacobi.cpp failed with unspecified launch failure (719).
ERROR: CUDA RT call "cudaFree( l2_norm_d )" in line 374 of file jacobi.cpp failed with unspecified launch failure (719).
ERROR: CUDA RT call "cudaFree( a_new )" in line 376 of file jacobi.cpp failed with unspecified launch failure (719).
ERROR: CUDA RT call "cudaFree( a )" in line 377 of file jacobi.cpp failed with unspecified launch failure (719).
ERROR: CUDA RT call "cudaMalloc( &a, nx*ny*sizeof(real) )" in line 173 of file jacobi.cpp failed with unspecified launch failure (719).
ERROR: CUDA RT call "cudaMalloc( &a_new, nx*ny*sizeof(real) )" in line 174 of file jacobi.cpp failed with unspecified launch failure (719).
ERROR: CUDA RT call "cudaMemset( a, 0, nx*ny*sizeof(real) )" in line 175 of file jacobi.cpp failed with unspecified launch failure (719).
ERROR: CUDA RT call "cudaMemset( a_new, 0, nx*ny*sizeof(real) )" in line 176 of file jacobi.cpp failed with unspecified launch failure (719).

[jwb0001:31218] Failing at address: 0x241c000079f2
[jwb0001:31218] [ 0] /usr/lib64/libpthread.so.0(+0x12dd0)[0x148ffffcdbdd0]
[jwb0001:31216] [ 0] /usr/lib64/libpthread.so.0(+0x12dd0)[0x149c2d4dbdd0]
[jwb0001:31216] [ 1] /p/software/juwelsbooster/stages/Devel-2020/software/nvidia-driver/default/lib64/libcuda.so.1(+0x22f960)[0x149c2b0d5960]
[jwb0001:31216] [ 2] /p/software/juwelsbooster/stages/Devel-2020/software/nvidia-driver/default/lib64/libcuda.so.1(+0x3ffcc29)[0x149c2b2a2c29]
[jwb0001:31216] [ 3] /p/software/juwelsbooster/stages/Devel-2020/software/CUDA/11.0/lib64/libcudart.so.11.0(cudaMemsetAsync+0xbe)[0x149c2e2751ae]
[jwb0001:31216] [ 4] /p/home/jusers/kraus1/juwels/workspace/cuda/6-Multi_GPU_MPI/exercises/solution2./jacobi[0x402e8f]
[jwb0001:31216] [ 5] /usr/lib64/libc.so.6(__libc_start_main+0xf3)[0x149c2d70c6a3]
[jwb0001:31216] [ 6] /p/home/jusers/kraus1/juwels/workspace/cuda/6-Multi_GPU_MPI/exercises/solution2./jacobi[0x401f2e]
[jwb0001:31218] [ 1] /p/software/juwelsbooster/stages/Devel-2020/software/nvidia-driver/default/lib64/libcuda.so.1(+0x22f960)[0x148ffd8d5960]
[jwb0001:31218] [ 2] /p/software/juwelsbooster/stages/Devel-2020/software/nvidia-driver/default/lib64/libcuda.so.1(+0x3ffcc29)[0x148ffdaa2c29]
[jwb0001:31218] [ 3] /p/software/juwelsbooster/stages/Devel-2020/software/CUDA/11.0/lib64/libcudart.so.11.0(cudaMemsetAsync+0xbe)[0x149000a751ae]
[jwb0001:31218] [ 4] /p/home/jusers/kraus1/juwels/workspace/cuda/6-Multi_GPU_MPI/exercises/solution2./jacobi[0x402e8f]
[jwb0001:31218] [ 5] /usr/lib64/libc.so.6(__libc_start_main+0xf3)[0x148ffff0c6a3]
[jwb0001:31218] [ 6] /p/home/jusers/kraus1/juwels/workspace/cuda/6-Multi_GPU_MPI/exercises/solution2./jacobi[0x401f2e]
[jwb0001:31218] *** End of error message ***
[jwb0001:31216] *** End of error message ***
srun: error: jwb0001: task 0: Exited with exit code 11
srun: error: jwb0001: task 1: Terminated
srun: Force Terminated job step 3089145.8
[kraus1@jwlogin23 solution2]$ ls
Makefile          jacobi
core.jwb0001.juwels.31216 jacobi.0.compute-sanitizer
core.jwb0001.juwels.31218 jacobi.0.compute-sanitizer.log
[jwb0001:31218] jacobi.1.compute-sanitizer
[jwb0001:31218] jacobi.1.compute-sanitizer.log
[jwb0001:31218] jacobi.cpp
[jwb0001:31218] jacobi_kernels.cu
[jwb0001:31218] jacobi_kernels.o
```

# ERROR CHECKING WITH COMPUTE SANITIZER

## Read Output Files with `compute-sanitizer --read`



The image shows two terminal windows side-by-side. The left terminal window has a title bar "kraus1@jwlogin23:~/workspace" and contains the command "[kraus1@jwlogin23 solution2]\$ compute-sanitizer --read jacobi.0.compute-sanitizer". The right terminal window also has a title bar "kraus1@jwlogin23:~/workspace" and displays the output of the command. The output is a long list of error messages from the Compute Sanitizer. It starts with a header "===== in /p/home/jusers/kraus1/juwels/workspace/cuda/6-Multi\_GPU\_MPI/exercises/solution2./jacobi". Following this, there are multiple entries, each starting with "===== Invalid \_\_global\_\_ read of size 4 bytes" and providing details about the memory access error, including the file name (.cu), line number (85), kernel name (jacobi\_kernel), thread ID (11,1,0), block ID (68,2,0), and address (0x149c28002234). The output continues with "Address 0x149c28002234 is out of bounds" and "Saved host backtrace up to driver entry point at kernel launch time". The host backtrace includes frames such as "Host Frame:cuLaunchKernel [0x149c2b18a6de]" and "Host Frame:[0x149c2e23662b]". The final part of the output is "===== ERROR SUMMARY: 25918 errors".

```
[kraus1@jwlogin23 solution2]$ compute-sanitizer --read jacobi.0.compute-sanitizer
=====
===== in /p/home/jusers/kraus1/juwels/workspace/cuda/6-Multi_GPU_MPI/exercises/solution2./jacobi
=====
===== Invalid __global__ read of size 4 bytes
=====   at 0x1e0 in /p/home/jusers/kraus1/juwels/workspace/cuda/6-Multi_GPU_MPI/exercises/solution2/jacobi_kernels
.cu:85:jacobi_kernel(float*,float const *,float*,int,int,int)
=====   by thread (11,1,0) in block (68,2,0)
=====   Address 0x149c28002234 is out of bounds
=====   Saved host backtrace up to driver entry point at kernel launch time
=====   Host Frame:cuLaunchKernel [0x149c2b18a6de]
=====           in /p/software/juwelsbooster/stages/Devel-2020/software/nvidia-driver/default/lib64/libcuda.so.
1
=====
=====   Host Frame: [0x149c2e23662b]
=====           in /p/software/juwelsbooster/stages/Devel-2020/software/CUDA/11.0/lib64/libcudart.so.11.0
=====   Host Frame:cudaLaunchKernel [0x149c2e2765b1]
=====           in /p/software/juwelsbooster/stages/Devel-2020/software/CUDA/11.0/lib64/libcudart.so.11.0
=====   Host Frame: [0x405c1b]
=====           in /p/home/jusers/kraus1/juwels/workspace/cuda/6-Multi_GPU_MPI/exercises/solution2./jacobi
=====
=====   Host Frame: [0x405a9e]
=====           in /p/home/jusers/kraus1/juwels/workspace/cuda/6-Multi_GPU_MPI/exercises/solution2./jacobi
=====
=====   Host Frame: [0x405af1]
=====           in /p/home/jusers/kraus1/juwels/workspace/cuda/6-Multi_GPU_MPI/exercises/solution2./jacobi
=====
=====   Host Frame: [0x40569f]
=====           in /p/home/jusers/kraus1/juwels/workspace/cuda/6-Multi_GPU_MPI/exercises/solution2./jacobi
=====
=====   Host Frame: [0x4045a8]
=====           in /p/home/jusers/kraus1/juwels/workspace/cuda/6-Multi_GPU_MPI/exercises/solution2./jacobi
=====
=====   Host Frame: [0x40281d]
=====           in /p/home/jusers/kraus1/juwels/workspace/cuda/6-Multi_GPU_MPI/exercises/solution2./jacobi
=====
=====   Host Frame:__libc_start_main [0x149c2d70c6a3]
=====           in /usr/lib64/libc.so.6
=====
=====   Host Frame:[0x401f2e]
=====           in /p/home/jusers/kraus1/juwels/workspace/cuda/6-Multi_GPU_MPI/exercises/solution2./jacobi
=====
===== ERROR SUMMARY: 25918 errors
[kraus1@jwlogin23 solution2]$ |
```

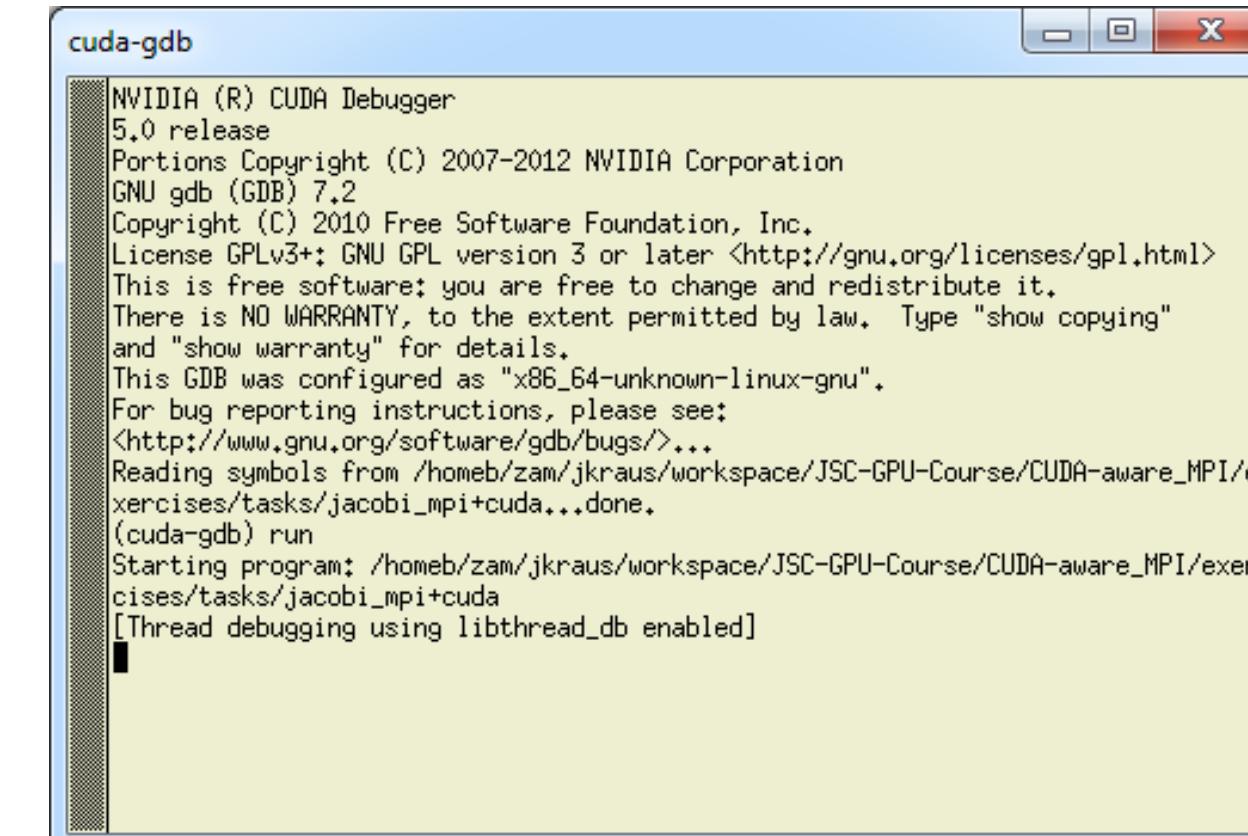
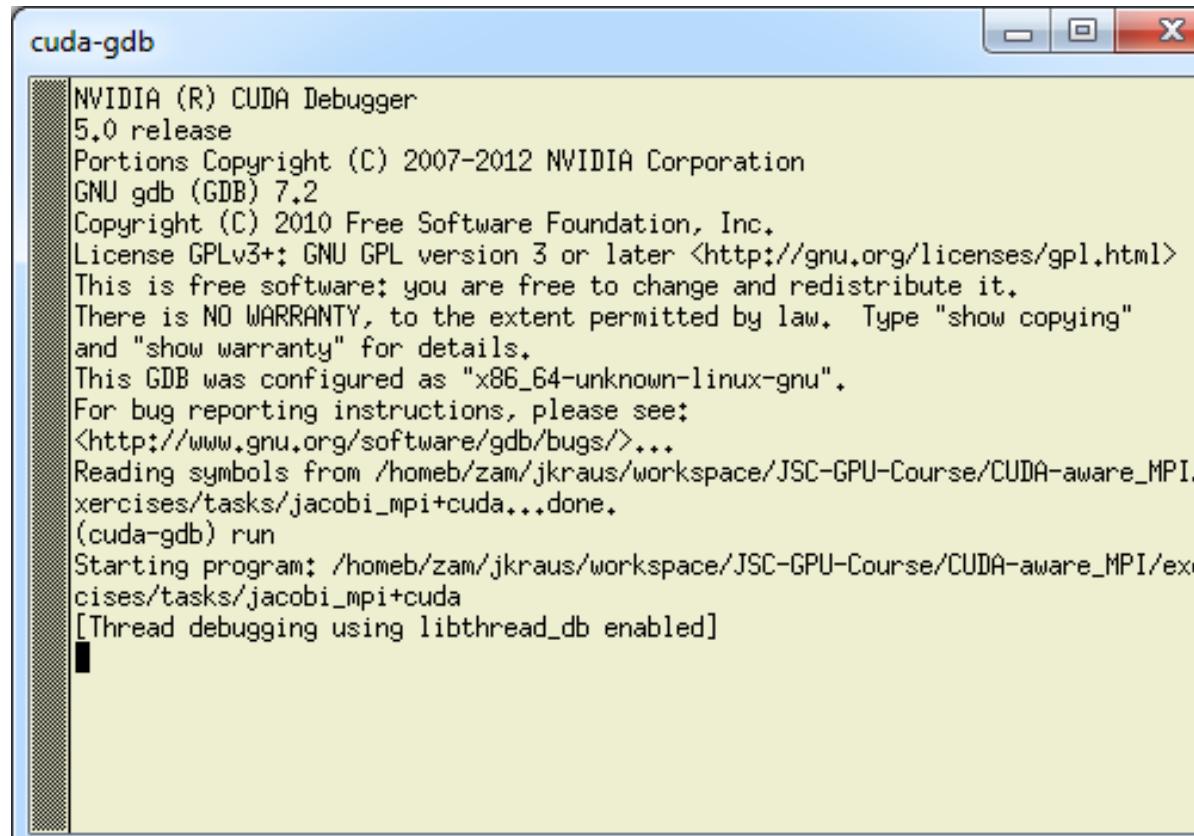
# DEBUGGING MPI+CUDA APPLICATIONS

## Using cuda-gdb with MPI Applications

Use cuda-gdb just like gdb

For smaller applications, just launch xterms and cuda-gdb

```
mpiexec -x -np 2 xterm -e cuda-gdb ./myapp <args>
```



# DEBUGGING MPI+CUDA APPLICATIONS

## cuda-gdb Attach

```
if ( rank == 0 ) {
    int i=0;
    printf("rank %d: pid %d on %s ready for attach\n.", rank, getpid(), name);
    while (0 == i) { sleep(5); }
}

> mpiexec -np 2 ./jacobi_mpi+cuda
Jacobi relaxation Calculation: 4096 x 4096 mesh with 2 processes and one Tesla M2070 for
each process (2049 rows per process).
rank 0: pid 30034 on judge107 ready for attach
> ssh judge107
jkraus@judge107:~> cuda-gdb --pid 30034
```

# DEBUGGING MPI+CUDA APPLICATIONS

## CUDA\_DEVICE\_WAITS\_ON\_EXCEPTION

The image shows two terminal windows side-by-side. The left window displays the application's execution log, including iteration statistics and error messages. The right window shows the debugger's response to the application's exception.

**Left Terminal Output:**

```
Iteration: 700 - Residue: 0.306564
Iteration: 800 - Residue: 0.306564
Iteration: 900 - Residue: 0.306564
Stopped after 1000 iterations with residue 0.306564
Total Jacobi run time: 0.8700 sec.
Average per-process communication time: 0.2765 sec.
Measured lattice updates: 4.81 GLU/s (total), 1.20 GLU/s (per process)
Measured FLOPS: 24.06 GFLOPS (total), 6.01 GFLOPS (per process)
Measured device bandwidth: 230.95 GB/s (total), 57.74 GB/s (per process)
[jkraus@sb077 bin]$ CUDA_DEVICE_WAITS_ON_EXCEPTION=1 MV2_USE_AWARE_MPI_ASYNC -t 2 2 -d 1024 1024 -fs
Topology size: 2 x 2
Local domain size (current node): 1024 x 1024
Global domain size (all nodes): 2048 x 2048
Starting Jacobi run with 4 processes:
sb077: The application encountered a device error and CUDA_DEBUGGER_ATTACHED can now attach a debugger to the application (PID 28250) for device 3.
sb077: The application encountered a device error and CUDA_DEBUGGER_ATTACHED can now attach a debugger to the application (PID 28252) for device 3.
sb077: The application encountered a device error and CUDA_DEBUGGER_ATTACHED can now attach a debugger to the application (PID 28251) for device 3.
sb077: The application encountered a device error and CUDA_DEBUGGER_ATTACHED can now attach a debugger to the application (PID 28249) for device 3.
```

**Right Terminal Output:**

```
Reading symbols from /usr/lib64/libnes-rdmav2.so... (no debugging symbols found)...done.
Loaded symbols for /usr/lib64/libnes-rdmav2.so
Reading symbols from /usr/lib64/libmlx4-rdmav2.so... (no debugging symbols found)...done.
Loaded symbols for /usr/lib64/libmlx4-rdmav2.so
Reading symbols from /usr/lib64/libipathverbs-rdmav2.so... (no debugging symbols found)...done.
Loaded symbols for /usr/lib64/libipathverbs-rdmav2.so
0x00007f5ba011fa01 in clock_gettime ()
$1 = 1

CUDA Exception: Device Illegal Address
The exception was triggered in device 3.

Program received signal CUDA_EXCEPTION_10, Device Illegal Address.
[Switching focus to CUDA kernel 0, grid 8, block (6,36,0), thread (0,6,0), device 3, sm 0, warp 13, lane 0]
0x000000000018e1ce8 in JacobiComputeKernel<<<(64,64,1),(16,16,1)>>> (size=..., startmod=..., endmod=..., oldBlock=0x2300200000, newBlock=0x2300b20000, devResidue=0x2301340000, stride=1024) at Device.cu:150
150          AtomicMax<real>(devResidue, rabs(newVal - oldBlock[memIdx]));
(cuda-gdb) bt
#0 0x000000000018e1ce8 in JacobiComputeKernel<<<(64,64,1),(16,16,1)>>> (size=..., startmod=..., endmod=..., oldBlock=0x2300200000, newBlock=0x2300b20000, devResidue=0x2301340000, stride=1024) at Device.cu:150
(cuda-gdb)
```

# DEBUGGING MPI+CUDA APPLICATIONS

With `CUDA_ENABLE_COREDUMP_ON_EXCEPTION=1` core dumps are generated in case of an exception:

- Can be used for offline debugging

- Helpful if live debugging is not possible

`CUDA_ENABLE_CPU_COREDUMP_ON_EXCEPTION`: Enable/Disable CPU part of core dump (enabled by default)

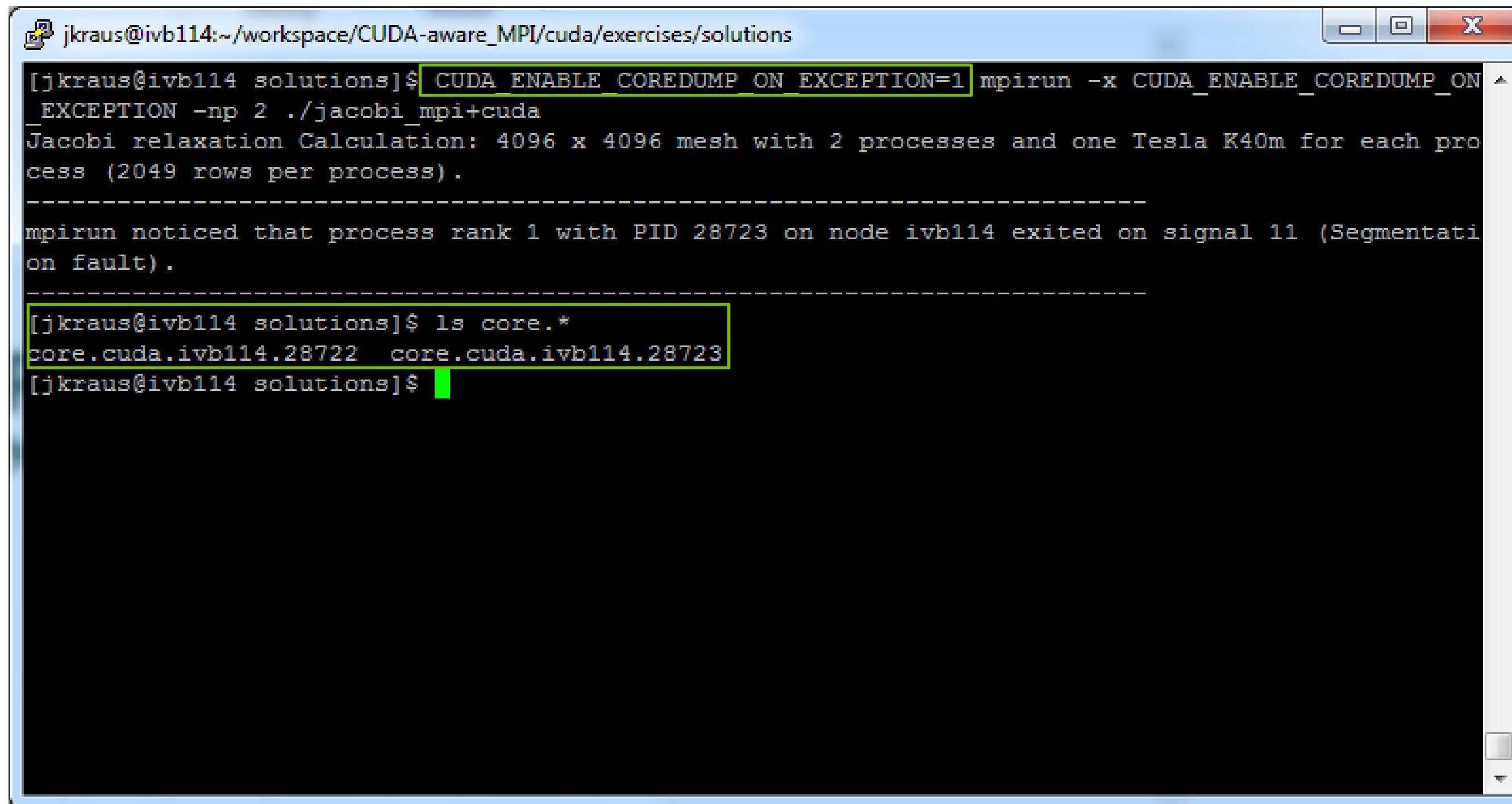
`CUDA_COREDUMP_FILE`: Specify name of core dump file

Open GPU: (cuda-gdb) target cudacore core.cuda

Open CPU+GPU: (cuda-gdb) target core core.cpu core.cuda

# DEBUGGING MPI+CUDA APPLICATIONS

## CUDA\_ENABLE\_COREDUMP\_ON\_EXCEPTION



The screenshot shows a terminal window with the following session:

```
[jkraus@ivb114 solutions]$ CUDA_ENABLE_COREDUMP_ON_EXCEPTION=1 mpirun -x CUDA_ENABLE_COREDUMP_ON_EXCEPTION -np 2 ./jacobi_mpi+cuda
Jacobi relaxation Calculation: 4096 x 4096 mesh with 2 processes and one Tesla K40m for each process (2049 rows per process).

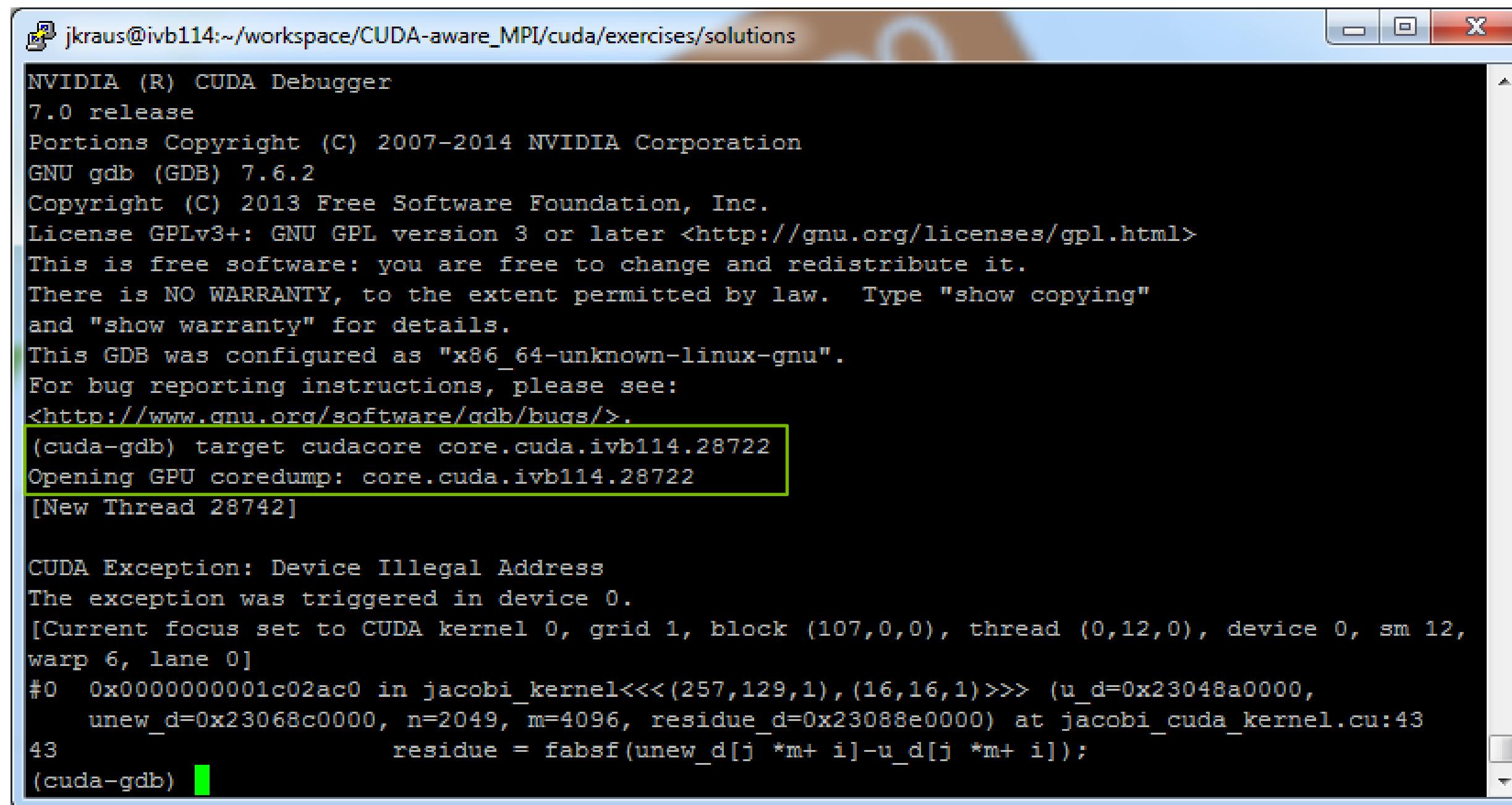
mpirun noticed that process rank 1 with PID 28723 on node ivb114 exited on signal 11 (Segmentation fault).

[jkraus@ivb114 solutions]$ ls core.*
core.cuda.ivb114.28722  core.cuda.ivb114.28723
[jkraus@ivb114 solutions]$
```

The command `CUDA_ENABLE_COREDUMP_ON_EXCEPTION=1` is highlighted in yellow, and the output of `ls core.*` is also highlighted in yellow.

# DEBUGGING MPI+CUDA APPLICATIONS

## CUDA\_ENABLE\_COREDUMP\_ON\_EXCEPTION



The screenshot shows the NVIDIA (R) CUDA Debugger interface. The title bar reads "jkraus@ivb114:~/workspace/CUDA-aware\_MPI/cuda/exercises/solutions". The main window displays the following text:

```
NVIDIA (R) CUDA Debugger
7.0 release
Portions Copyright (C) 2007-2014 NVIDIA Corporation
GNU gdb (GDB) 7.6.2
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-unknown-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
(cuda-gdb) target cudacore core.cuda.ivb114.28722
Opening GPU coredump: core.cuda.ivb114.28722
[New Thread 28742]

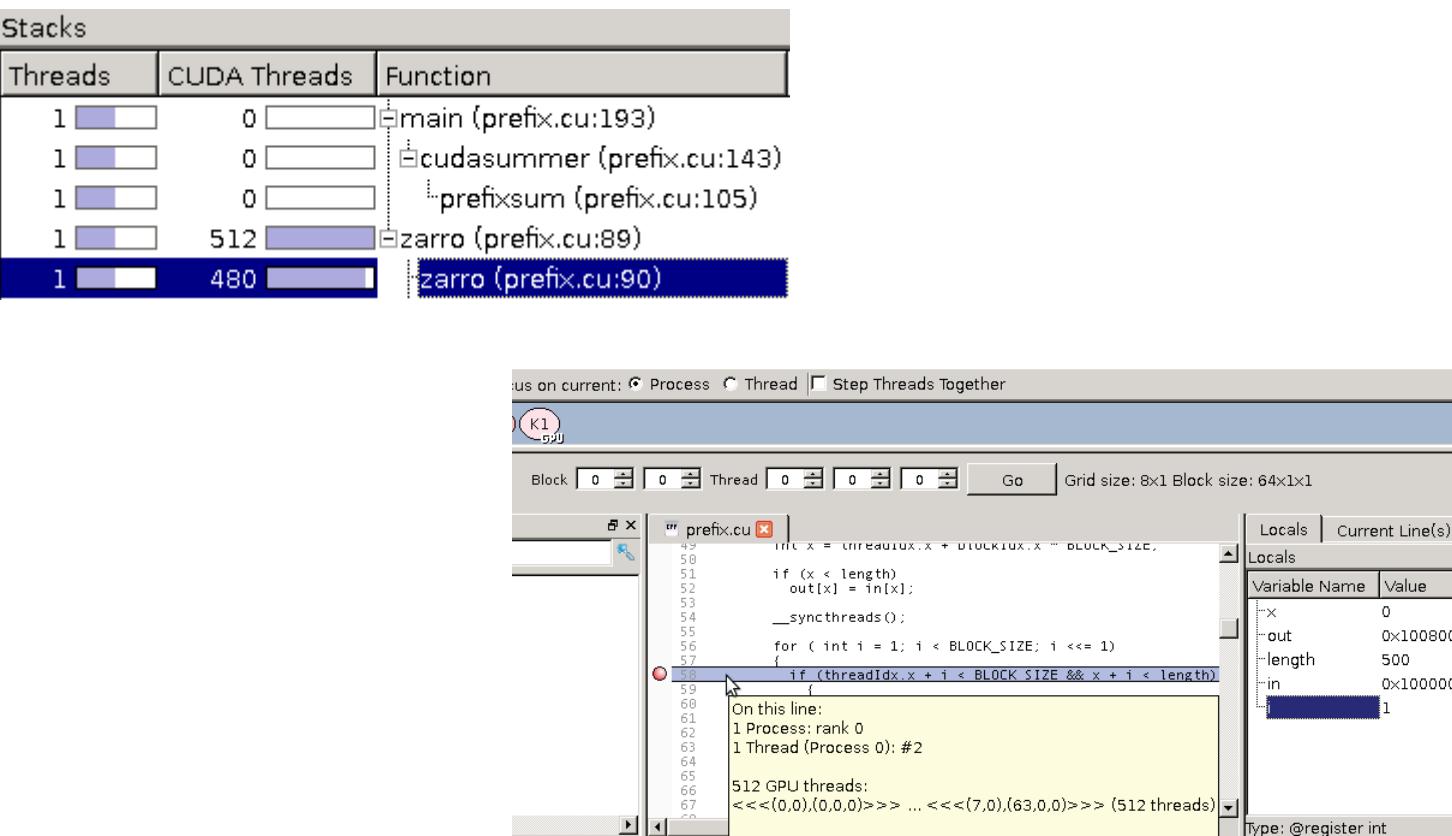
CUDA Exception: Device Illegal Address
The exception was triggered in device 0.
[Current focus set to CUDA kernel 0, grid 1, block (107,0,0), thread (0,12,0), device 0, sm 12,
warp 6, lane 0]
#0 0x0000000001c02ac0 in jacobi_kernel<<<(257,129,1),(16,16,1)>>> (u_d=0x23048a0000,
    unew_d=0x23068c0000, n=2049, m=4096, residue_d=0x23088e0000) at jacobi_cuda_kernel.cu:43
43             residue = fabsf(unew_d[j *m+ i]-u_d[j *m+ i]);
(cuda-gdb)
```

# DEBUGGING MPI+CUDA APPLICATIONS

## Third Party Tools

Arm DDT debugger

Rogue Wave TotalView



# PROFILING MPI+CUDA APPLICATIONS

## Using Nsight Systems

Trace MPI and embed MPI rank in output filename (OpenMPI)

```
mpirun -np $np nsys profile -o profile.%q{OMPI_COMM_WORLD_RANK} \
--trace=mpi,cuda --mpi-impl openmpi
```

**MVAPICH2:** MV2\_COMM\_WORLD\_RANK  
--mpi-impl mpich

# PROFILING MPI+CUDA APPLICATIONS

## Using Nsight Systems

The image shows two terminal windows side-by-side, both titled "prom.nvidia.com - PuTTY".

The left terminal window displays the configuration of an MPI profile:

```
jkraus@prm-dgx-01:~/workspace/multi-gpu-pro
pi, cuda --mpi-impl openmpi -o jacobi.profile
WARNING: OpenMPI or MPICH tracing relies on
        ftrace events.
**** collection configuration ****
  output_filename = /home/jkraus/work
  force-overwrite = false
  stop-on-exit = true
  export_sqlite = false
  stats = false
  capture-range = none
  stop-on-range-end = false
Beta: ftrace events:
  ftrace-keep-user-config = false
  trace-GPU-context-switch = false
  delay = 0 seconds
  duration = 0 seconds
  kill = signal number 15
  inherit-environment = true
  show-output = true
  trace-fork-before-exec = false
  sample_cpu = true
  backtrace_method = LBR
  wait = all
  trace_cublas = false
```

The right terminal window shows the import and saving of the profile data:

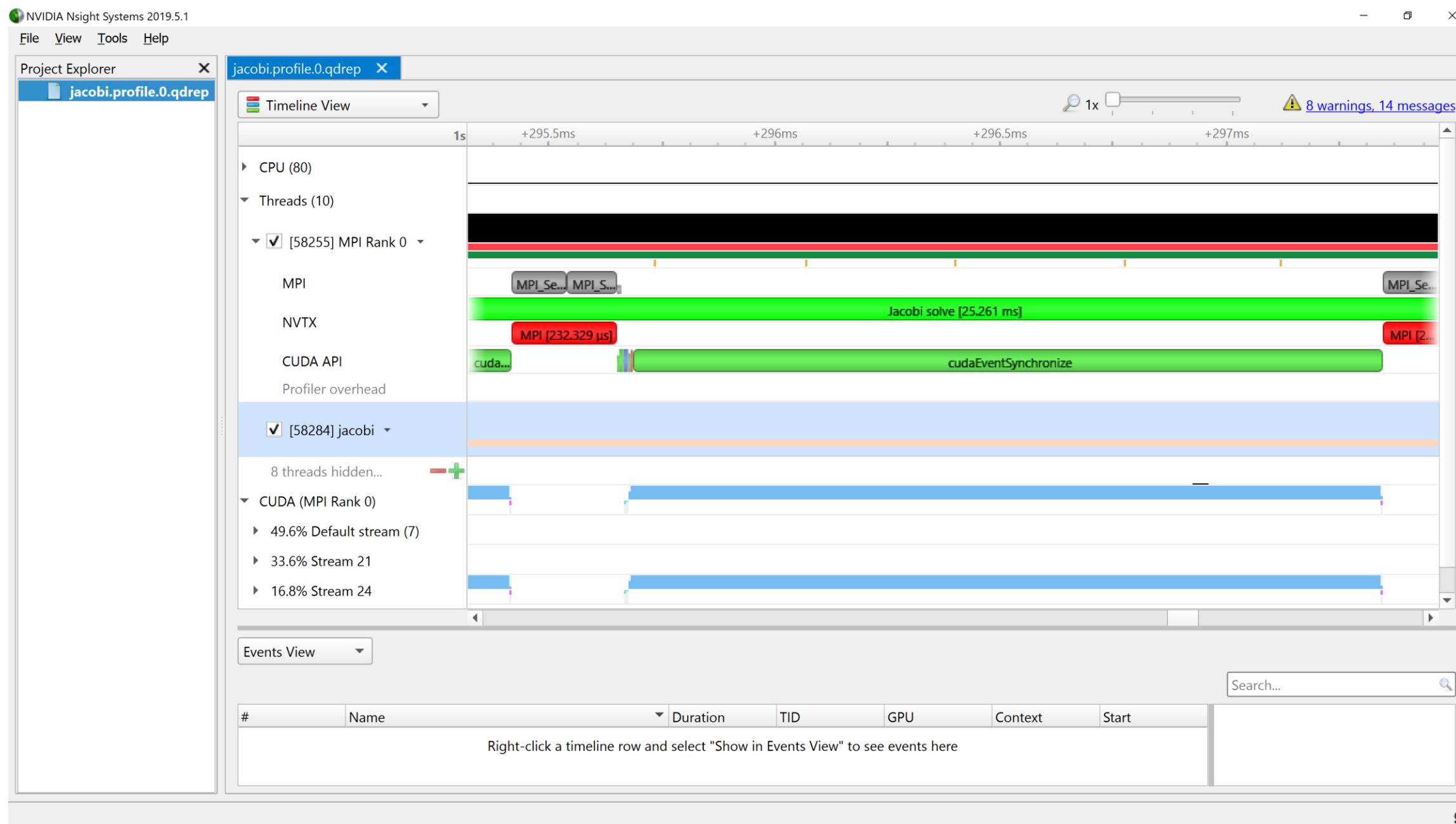
```
Importing...
Importing [=====58%]
Importing [=====100%]
Saving report to file "/home/jkraus/workspace/multi-gpu-programming-models/mpi/jacobi.profile.1.qd
rep"
Importing [=====100%]
Saving report to file "/home/jkraus/workspace/multi-gpu-programming-models/mpi/jacobi.profile.0.qd
rep"
Report file saved.
Please discard the qdstrm file and use the qdrep file instead.

Removed /home/jkraus/workspace/multi-gpu-programming-models/mpi/jacobi.profile.1.qdstrm as it was
successfully imported.
Please use the qdrep file instead.
Report file saved.
Please discard the qdstrm file and use the qdrep file instead.

Removed /home/jkraus/workspace/multi-gpu-programming-models/mpi/jacobi.profile.0.qdstrm as it was
successfully imported.
Please use the qdrep file instead.
jkraus@prm-dgx-01:~/workspace/multi-gpu-programming-models/mpi$ ls *.qdrep
jacobi.profile.0.qdrep  jacobi.profile.1.qdrep
jkraus@prm-dgx-01:~/workspace/multi-gpu-programming-models/mpi$
```

# PROFILING MPI+CUDA APPLICATIONS

## Using Nsight Systems



# PROFILING MPI+CUDA APPLICATIONS

## Third Party Tools

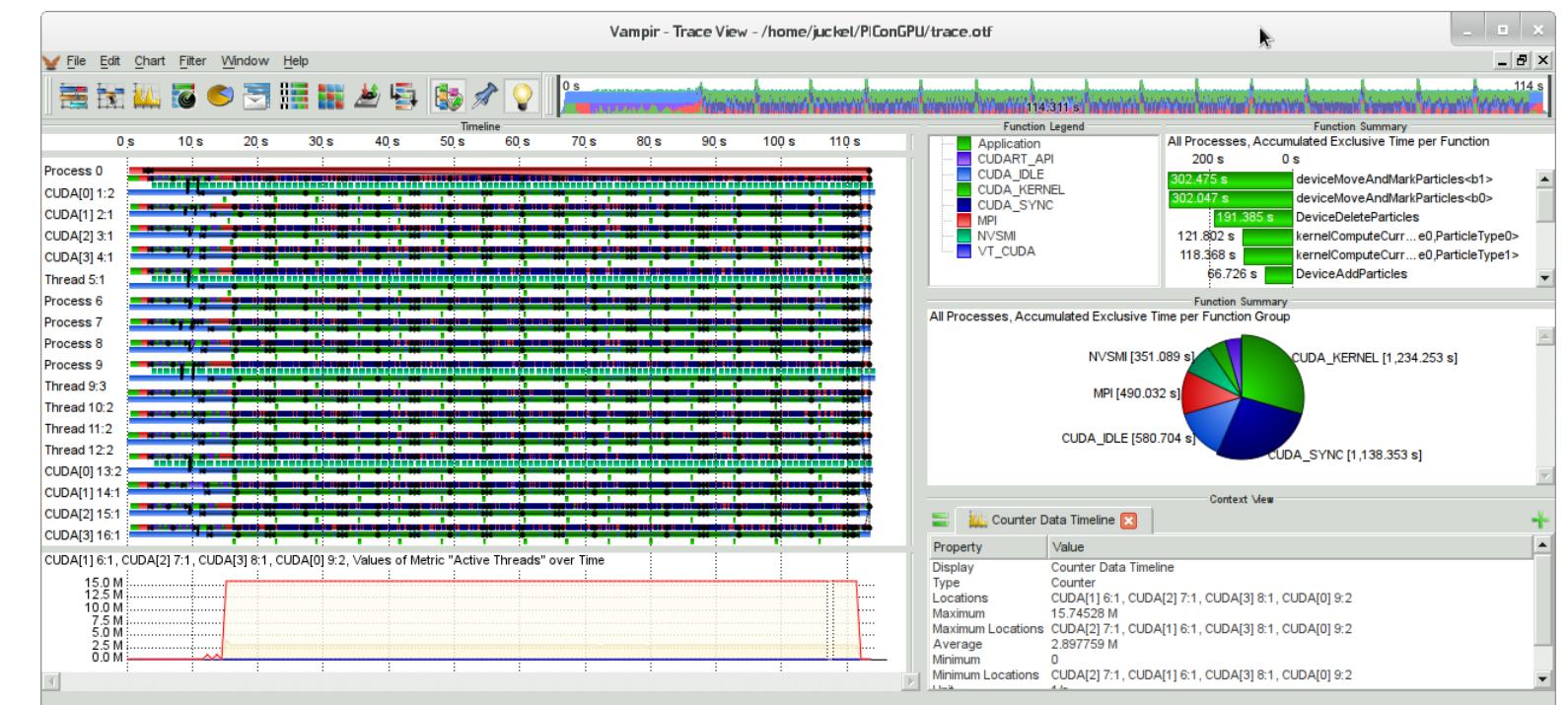
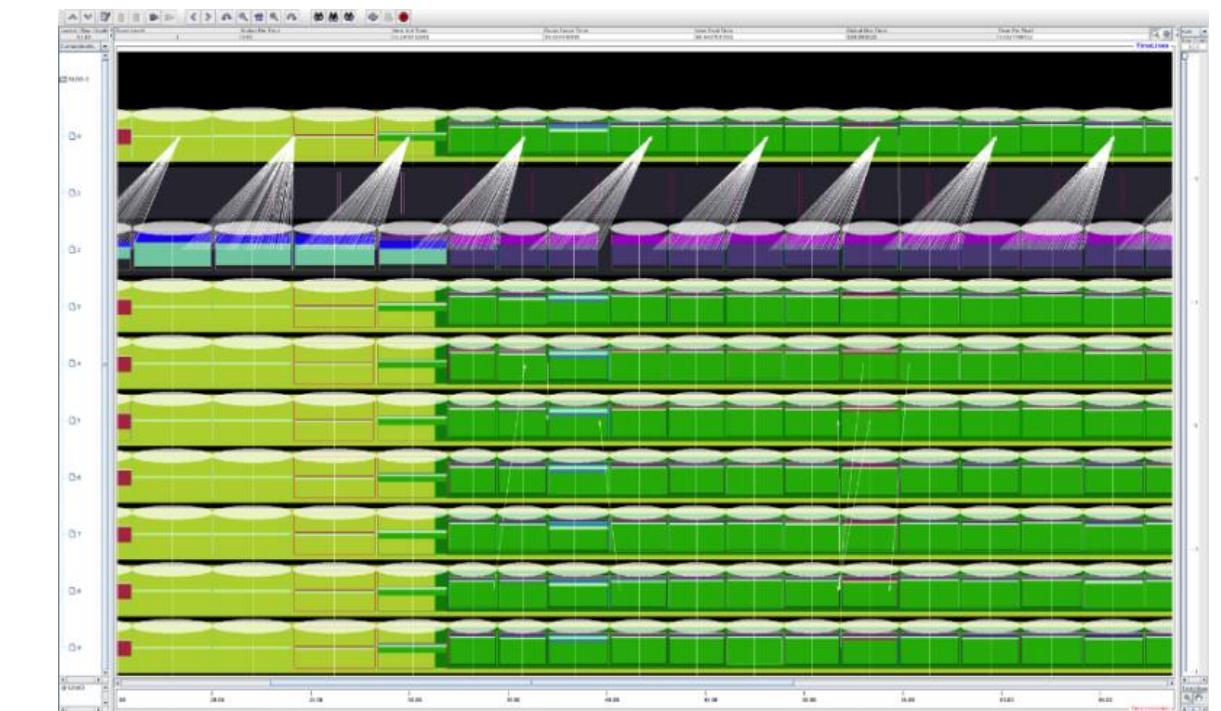
Multiple parallel profiling tools are CUDA-aware

Score-P

Vampir

Tau

These tools are good for discovering MPI issues as well as basic CUDA performance inhibitors.



# ADVANCED MPI ON GPUS

# BEST PRACTICE: USE NON-BLOCKING MPI

BLOCKING

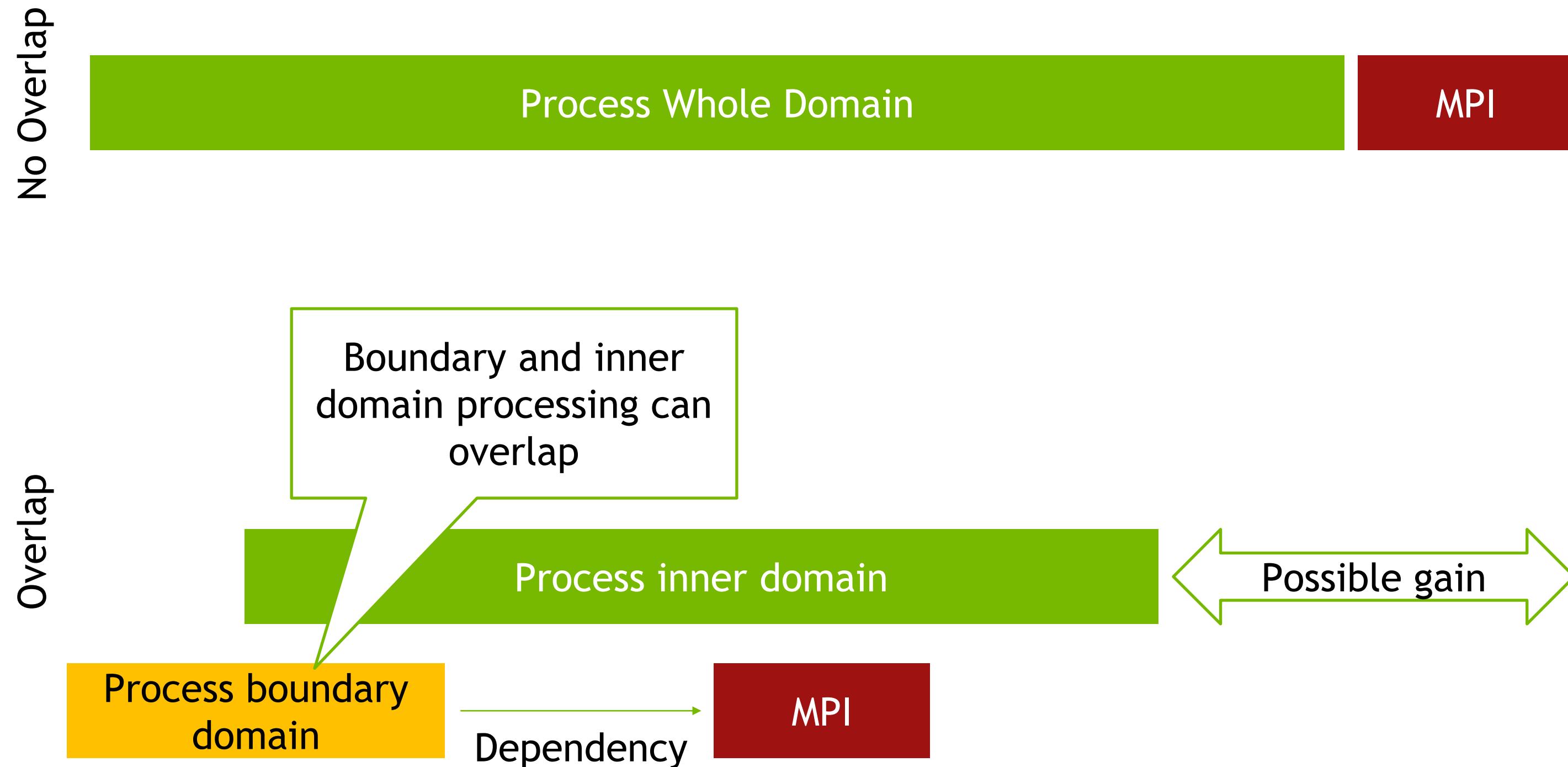
```
#pragma acc host_data use_device ( u_new ) {  
    MPI_Sendrecv(u_new+offset_first_row, m-2, MPI_DOUBLE, t_nb, 0,  
                 u_new+offset_bottom_boundary, m-2, MPI_DOUBLE, b_nb, 0,  
                 MPI_COMM_WORLD, MPI_STATUS_IGNORE);  
    MPI_Sendrecv(u_new+offset_last_row, m-2, MPI_DOUBLE, b_nb, 1,  
                 u_new+offset_top_boundary, m-2, MPI_DOUBLE, t_nb, 1,  
                 MPI_COMM_WORLD, MPI_STATUS_IGNORE);  
}
```

NON-BLOCKING

```
MPI_Request t_b_req[4];  
  
#pragma acc host_data use_device ( u_new ) {  
    MPI_Irecv(u_new+offset_top_boundary,m-2,MPI_DOUBLE,t_nb,t_b_req+0);  
    MPI_Irecv(u_new+offset_bottom_boundary,m-2,MPI_DOUBLE,b_nb,t_b_req+1);  
    MPI_Isend(u_new+offset_last_row,m-2,MPI_DOUBLE,b_nb,t_b_req+2);  
    MPI_Isend(u_new+offset_first_row,m-2,MPI_DOUBLE,t_nb,t_b_req+3);  
}  
  
MPI_Waitall(4, t_b_req, MPI_STATUSES_IGNORE);
```

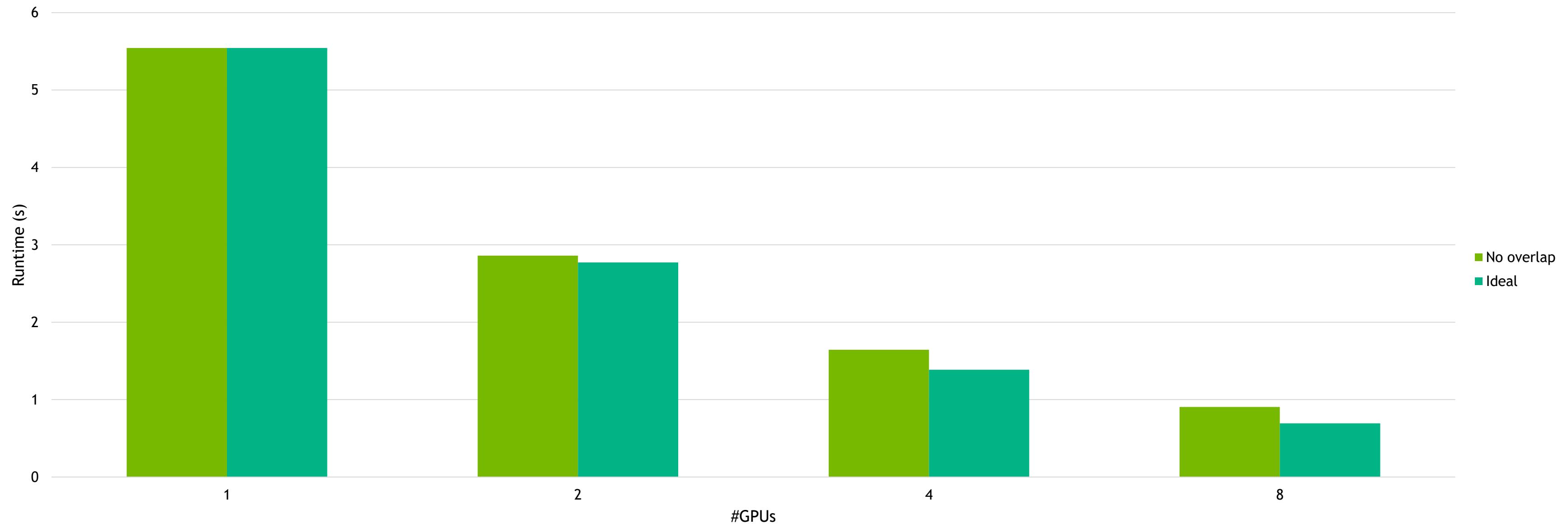
Gives MPI more  
opportunities to build  
efficient pipelines

# COMMUNICATION + COMPUTATION OVERLAP



# COMMUNICATION + COMPUTATION OVERLAP

ParaStation MPI 5.4.2-1 JUWELS - Tesla V100 - Jacobi on 18432x18432



Source: <https://github.com/NVIDIA/multi-gpu-programming-models/>

JUWELS: <http://fz-juelich.de/ias/jsc/juwels>

# COMMUNICATION + COMPUTATION OVERLAP

## CUDA with Streams

```
process_boundary_and_pack<<<gs_b,bs_b,0,s1>>>(u_new_d,u_d,to_left_d,to_right_d,n,m);  
  
process_inner_domain<<<gs_id,bs_id,0,s2>>>(u_new_d, u_d,to_left_d,to_right_d,n,m);  
  
cudaStreamSynchronize(s1);      //wait for boundary  
MPI_Request req[8];  
  
//Exchange halo with left, right, top and bottom neighbor  
  
MPI_Waitall(8, req, MPI_STATUSES_IGNORE);  
unpack<<<gs_s,bs_s,0,s2>>>(u_new_d, from_left_d, from_right_d, n, m);  
  
cudaDeviceSynchronize();        //wait for iteration to finish
```

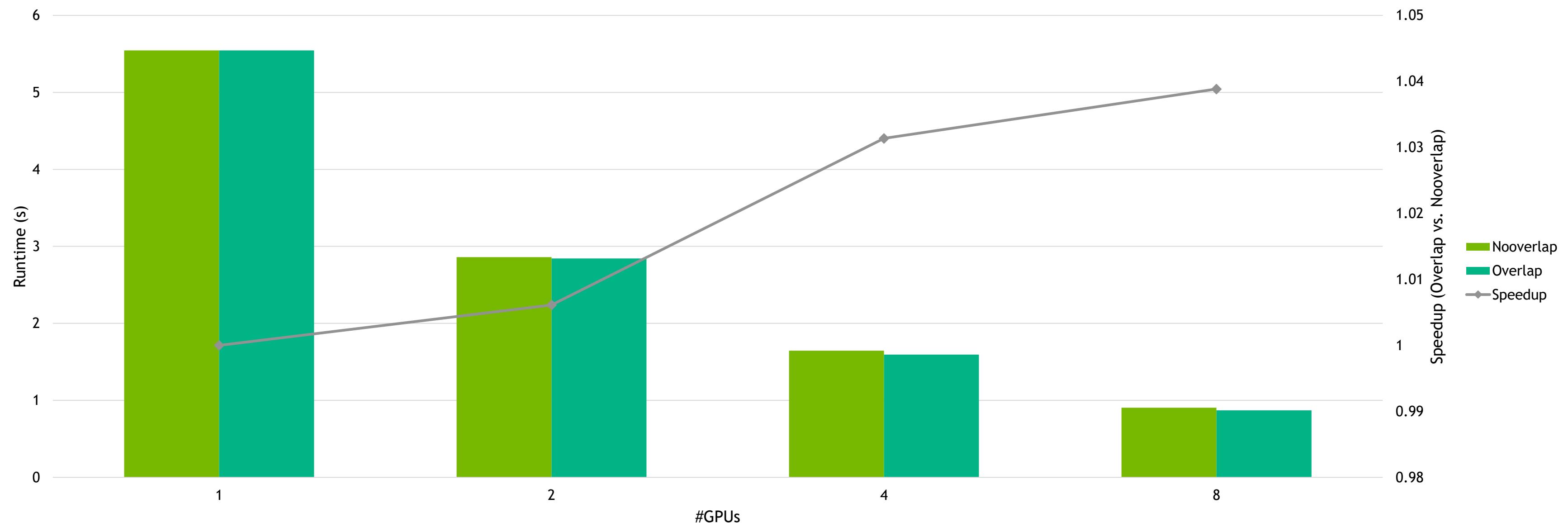
# COMMUNICATION + COMPUTATION OVERLAP

## OpenACC with Async Queues

```
#pragma acc parallel loop present ( u_new, u, to_left, to_right ) async(1)
for ( ... )
    //Process boundary and pack to_left and to_right
#pragma acc parallel loop present ( u_new, u ) async(2)
for ( ... )
    //Process inner domain
#pragma acc wait(1)          //wait for boundary
MPI_Request req[8];
#pragma acc host_data use_device ( from_left, to_left, form_right, to_right, u_new ) {
    //Exchange halo with left, right, top and bottom neighbor
}
MPI_Waitall(8, req, MPI_STATUSES_IGNORE);
#pragma acc parallel loop present ( u_new, from_left, from_right ) async(2)
for ( ... )
    //unpack from_left and from_right
#pragma acc wait           //wait for iteration to finish
```

# COMMUNICATION + COMPUTATION OVERLAP

ParaStation MPI 5.4.2-1 JUWELS - Tesla V100 - Jacobi on 18432x18432



Source: <https://github.com/NVIDIA/multi-gpu-programming-models/>

JUWELS: <http://fz-juelich.de/ias/jsc/juwels>

# CUDA HANDS-ON: MPI/COMPUTE OVERLAP

/p/scratch/share/jwb-porting-2021/Multi-GPU-Programming-with-MPI/  
CUDA/exercises/task3

Use `cudaStreamCreate` to create halo processing stream

Split jacobi step in top boundary, bottom boundary and bulk part

Launch top and bottom boundary part in halo processing stream

Look for TODOs

```
900, 0.122891
Num GPUs: 4.
8192x8192: 1 GPU:    5.4247 s, 4 GPUs:    1.4978 s, speedup:      3.62
```

Make Targets:

run:	run jacobi with \$NP procs.
jacobi:	build jacobi bin (default)
sanitize:	run with compute-sanitizer
profile:	profile with Nsight Systems
Solution is in ../../solution3	

<https://www.open-mpi.org/doc/current/>

# OPENACC HANDS-ON: HIDE MPI COMMUNICATION TIME

/p/scratch/share/jwb-porting-2021/Multi-GPU-Programming-with-MPI/OpenACC/exercises/**C | FORTRAN**/task2

- Start copy loop asynchronously
- Wait for async copy loop after MPI comm. is done

Look for TODOs

```
$ make  
mpicc -c -DUSE_DOUBLE -Minfo=accel -fast -acc=gpu -gpu=cc70 poisson2d.c [...]  
srun ./poisson2d  
Jacobi relaxation Calculation: 4096 x 4096 mesh  
[...]  
Num GPUs: 2.  
4096x4096: 1 GPU: 1.3163 s, 2 GPUs: 0.7757 s, speedup:  
MPI time: 0.0878 s, inter GPU BW: 1.39 GiB/s
```

## Make Targets:

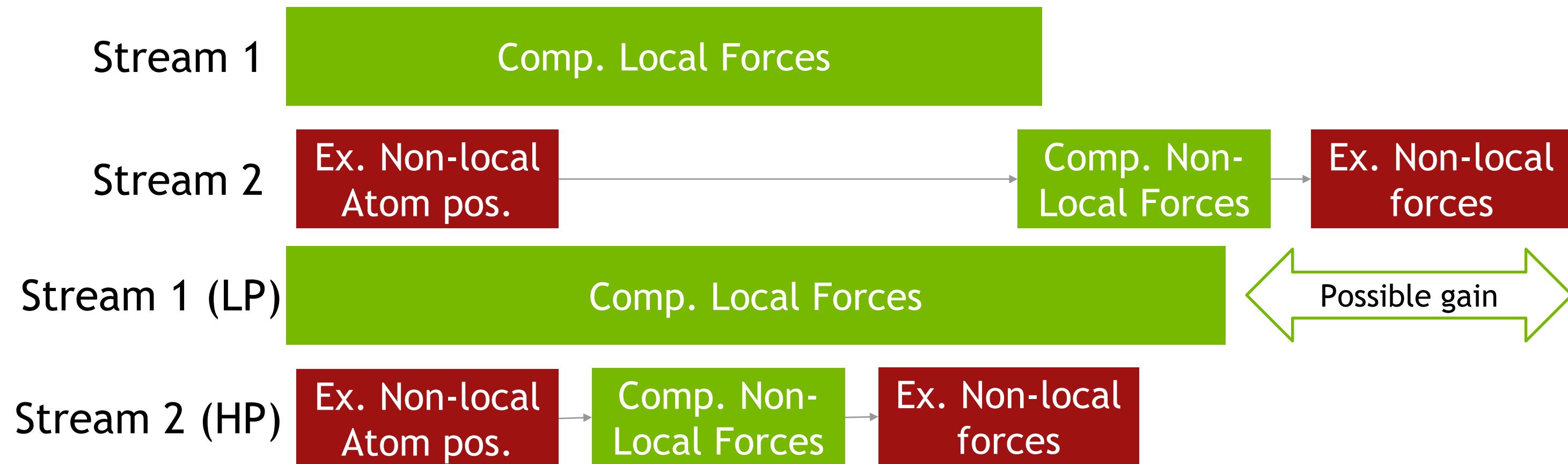
run:	run poisson2d ( <b>default</b> )
poisson2d:	build poisson2d <b>binary</b>
profile:	<b>profile with pgprof</b>
*.solution:	<b>same as above with solution</b> (poisson2d.solution.*)

# HIGH PRIORITY STREAMS

Improve scalability with high priority streams (available on CC 3.5+)

`cudaStreamCreateWithPriority`

Use-case: MD Simulations



# MPI AND UNIFIED MEMORY

## CAVEAT

Using Unified Memory with a non Unified Memory-aware MPI might fail with errors or even worse silently produce wrong results, e.g. when registering Unified Memory for RDMA.



**Use a Unified Memory-aware MPI,  
e.g. UCX based MPI (ParaStation, OpenMPI,...) or MVAPICH2-GDR  
since 2.2b**

# MPI AND UNIFIED MEMORY

## Performance Implications

Unified Memory can be used by any processor in the system

Memory pages of a Unified Memory allocation may migrate between processors' memories to ensure coherence and maximize performance

Different data paths are optimal for performance depending on where the data is:  
e.g. NVLink between peer GPUs



The MPI implementation needs to know where the data is,  
but it can't!

# MPI AND UNIFIED MEMORY

## Performance Implications - Simple Example

```
cudaMallocManaged( &array, n*sizeof(double), cudaMemAttachGlobal );  
  
while( . . . ) {  
  
    foo(array,n);  
  
    MPI_Send(array,...);  
  
    foo(array,n);  
  
}
```

# MPI AND UNIFIED MEMORY

## Performance Implications - Simple Example

- ▶ If foo is a CPU function pages of array might migrate to System Memory
- ▶ If foo is a GPU function pages of array might migrate to GPU Memory
- ▶ The MPI implementation is not aware of the application and thus doesn't know where array is and what's optimal

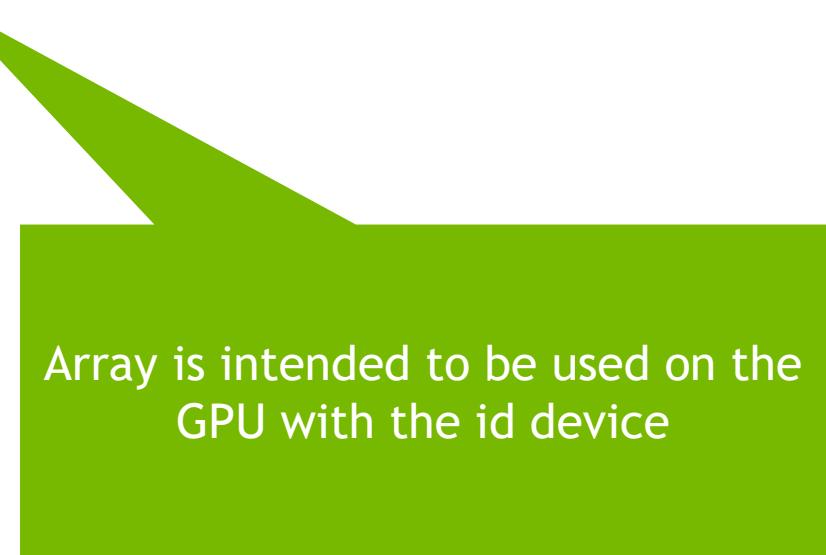
```
while( ... ) {  
    foo(array, n);  
    MPI_Send(array, ...);  
    foo(array, n);  
}
```

# MPI AND UNIFIED MEMORY

## The Future with Data Usage Hints

Tell where the application intends to use the data

```
cudaMallocManaged( &array, n*sizeof(double), cudaMemAttachGlobal ) ;  
  
cudaMemAdvise(array,n*sizeof(double),cudaMemAdviseSetPreferredLocation,device) ;  
  
while( ... ) {  
  
    foo(array,n) ;  
  
    MPI_Send(array,...) ;  
  
    foo(array,n) ;  
  
}
```



Array is intended to be used on the GPU with the id device

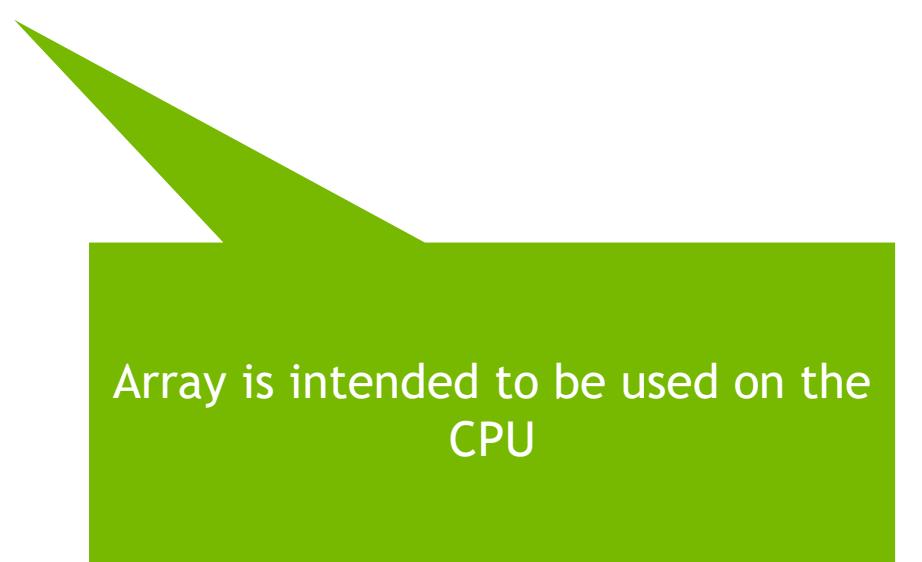
Remark: Data Usage Hints are available since CUDA 8, but currently not evaluated by any Unified Memory-aware MPI implementation.

# MPI AND UNIFIED MEMORY

## The Future with Data Usage Hints

Tell where the application intends to use the data

```
cudaMallocManaged( &array, n*sizeof(double), cudaMemAttachGlobal );  
  
cudaMemAdvise(array,n*sizeof(double),cudaMemAdviseSetPreferredLocation,cudaCpuDeviceId);  
  
while( ... ) {  
  
    foo(array,n);  
  
    MPI_Send(array,...);  
  
    foo(array,n);  
  
}
```



Array is intended to be used on the CPU

Remark: Data Usage Hints are available since CUDA 8, but currently not evaluated by any Unified Memory-aware MPI implementation.

# MPI AND UNIFIED MEMORY

## The Future with Data Usage Hints - Summary

Data usage hints can be queried by the MPI Implementation and allow it to take the optimal data path

If the application lies about the data usage hints it will run correctly but performance will be affected

Performance tools help to identify missing or wrong data usage hints

Data usage hints are general useful for the Unified Memory system and can improve application performance.

Remark: Data Usage Hints are only hints to guide the data usage policies of the Unified Memory system. The Unified Memory system might ignore them, e.g. to ensure coherence or in oversubscription scenarios.

# MPI AND UNIFIED MEMORY

## Current Status

Available Unified Memory-aware MPI implementations

- UCX-based MPIS, e.g. OpenMPI and ParaStation MPI
- MVAPICH2-GDR (since 2.2b)

Currently both don't evaluate Data Usage Hints, i.e. all Unified Memory is treated as Device Memory



Potential performance issues if not all buffers used in MPI are touched mainly on the GPU.

# DETECTING CUDA-AWARENESS

ParaStation MPI and OpenMPI (since 2.0.0) via `mpi-ext.h`

Macro:

```
MPIX_CUDA_AWARE_SUPPORT
```

Function for runtime decisions

```
MPIX_Query_cuda_support()
```

See <http://www.open-mpi.org/faq/?category=runcuda#mpi-cuda-aware-support>

ParaStation MPI: `MPI_INFO_ENV`

```
MPI_Info_get(MPI_INFO_ENV, "cuda_aware",
             sizeof(is_cuda_aware)-1, is_cuda_aware,
             &api_available);
```

# THANK YOU FOR YOUR ATTENTION

## HANDS-ON

- Work on your own codes
- Tasks from CUDA|OpenACC courses in /p/scratch/share/jwb-porting-2021/Multi-GPU-Programming-with-MPI
  - OpenACC task are available in C and FORTRAN
  - Instructions are in CUDA|OpenACC/exercises/Instructions.ipynb
  - PDFs with solutions are in CUDA|OpenACC/slides
  - Makefiles pick up: `export JSC_SUBMIT_CMD="srun --reservation=booster-porting-day3 -p booster -N 1 --gres=gpu:4 --pty"`
- Play with OSU Microbenchmarks: <http://mvapich.cse.ohio-state.edu/benchmarks/>
- Simple Jacobi Solver: <https://github.com/NVIDIA/multi-gpu-programming-models>

# BACKUP

# CUDA-AWARE MPI IMPLEMENTATIONS

HPC-X: OpenMPI/UCX based

OpenMPI: UCX based

MPICH: UCX based, next stable release 3.4 adds GPU support

MVAPICH2-GDR: MPICH based, no UCX, CUDA-aware MPI pioneer

Cray MPICH: MPICH based

IBM Spectrum MPI: OpenMPI/PAMI based

Parastation MPI: MPICH/UCX based, new kid on the block

Disclaimer: List might be incomplete!