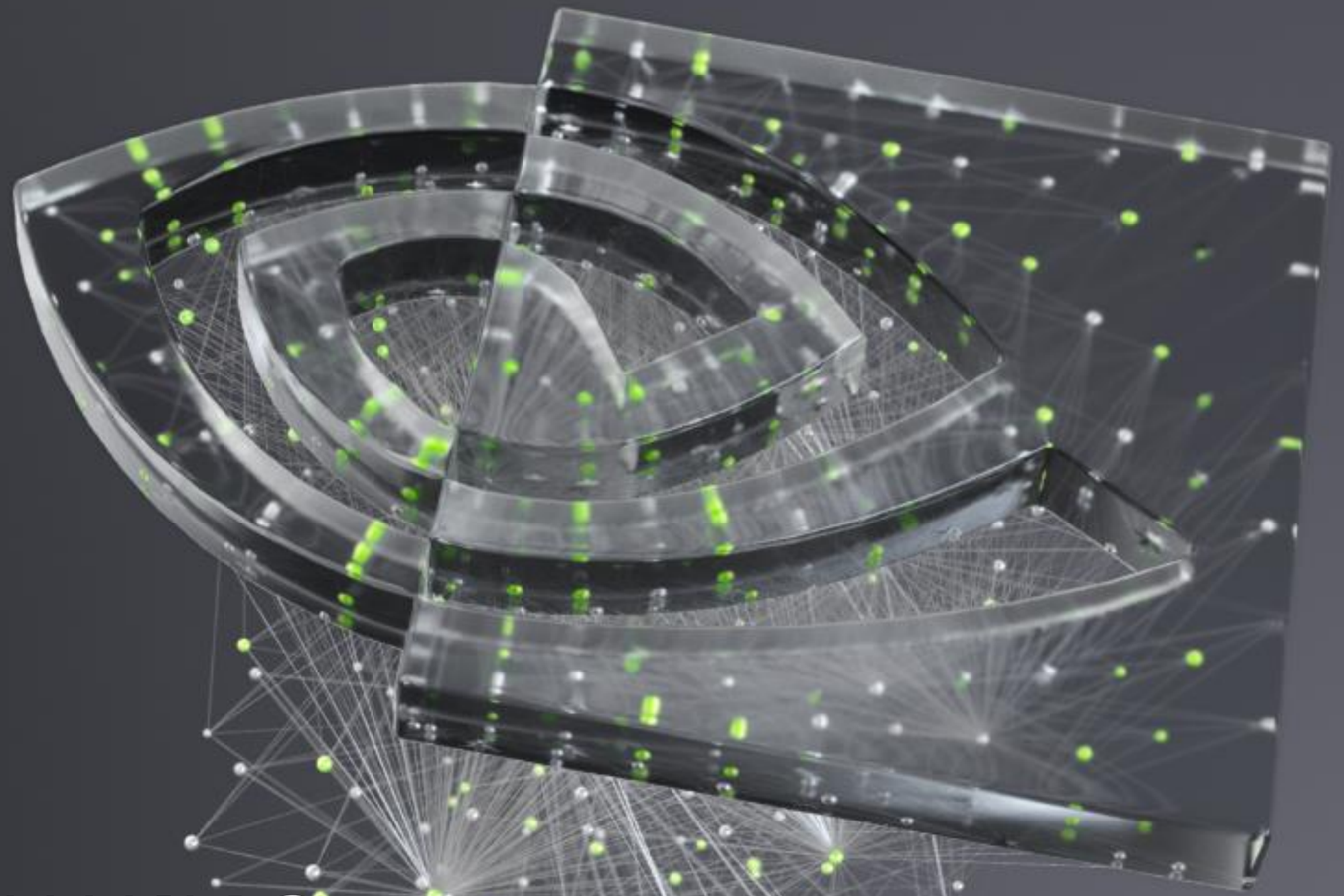
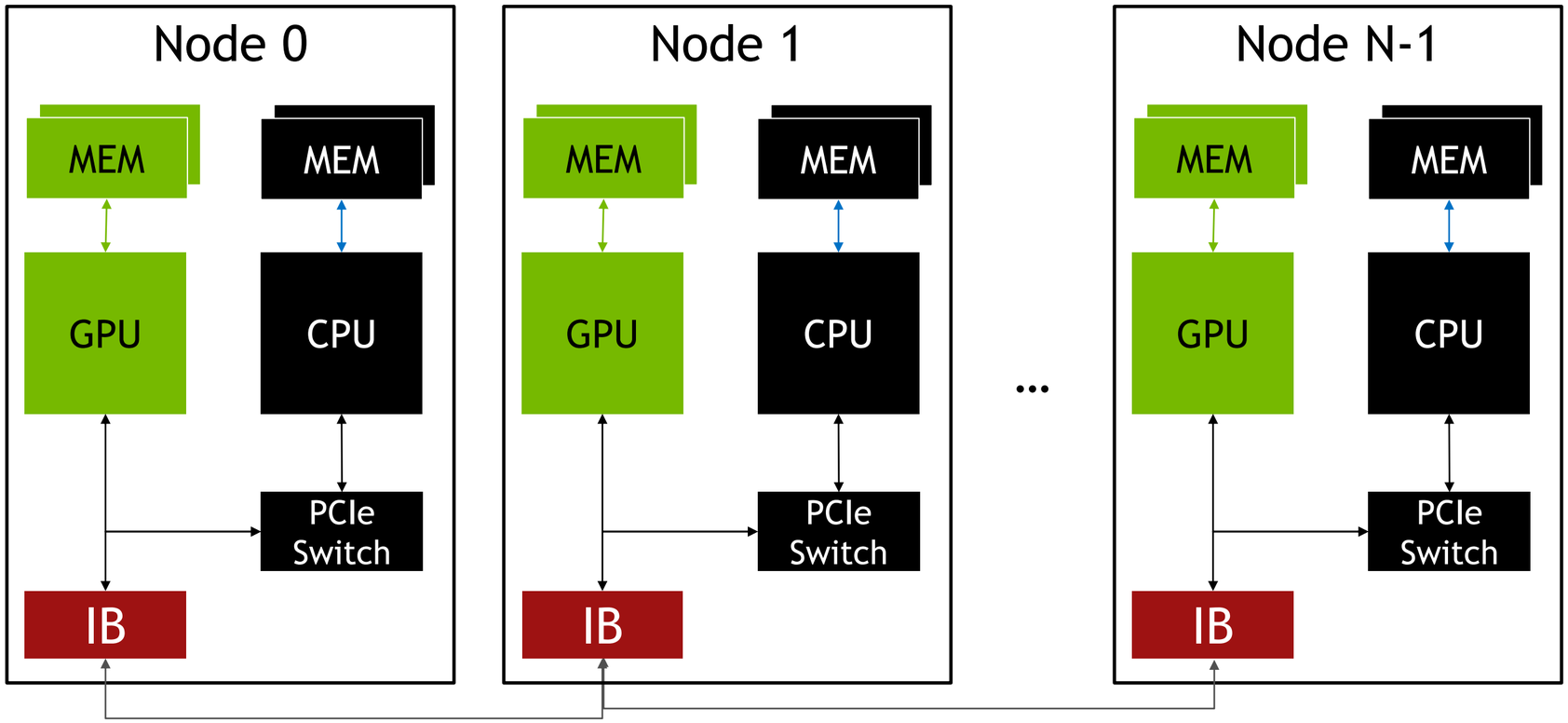


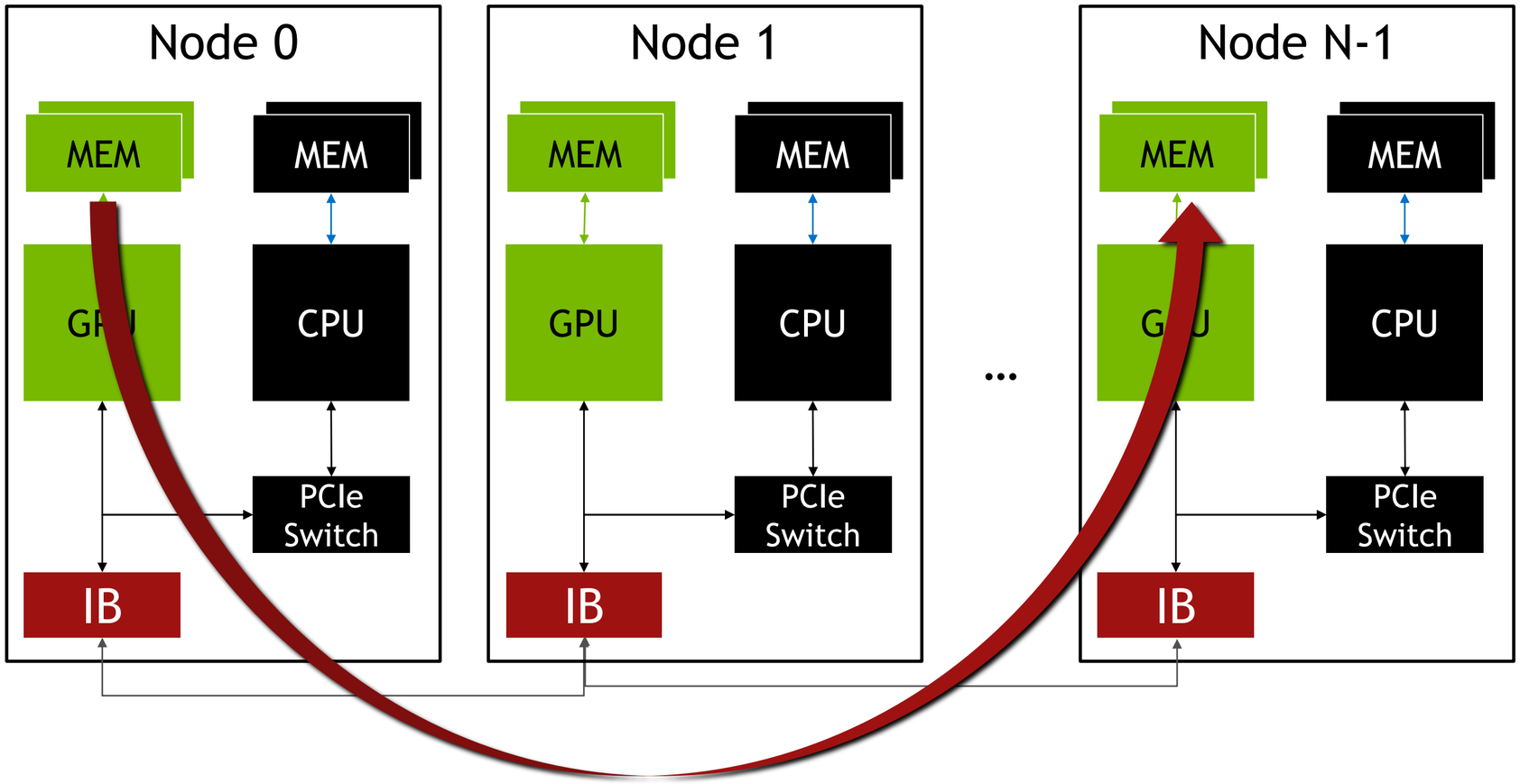
MULTI GPU PROGRAMMING WITH MPI

Jiri Kraus, Senior Devtech Compute

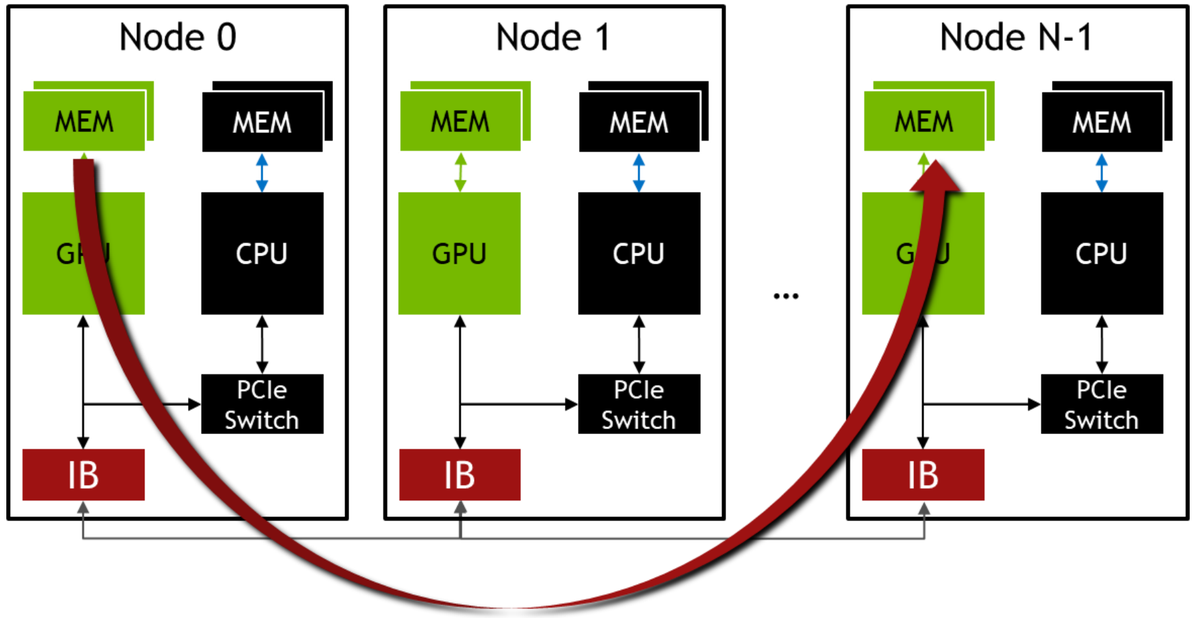
MPI+CUDA



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

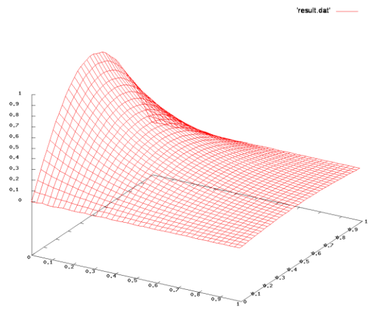
Solves the 2D-Laplace Equation on a rectangle

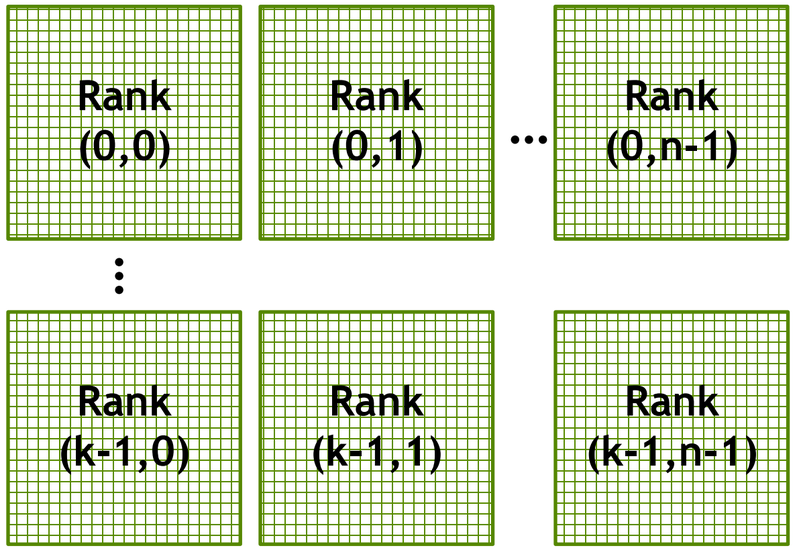
$$\Delta \boldsymbol{u}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{0} \; \forall \; (\boldsymbol{x}, \boldsymbol{y}) \in \Omega \backslash \boldsymbol{\delta}\Omega$$

Dirichlet boundary conditions (constant values on boundaries)

$$\boldsymbol{u}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}) \; \forall \; (\boldsymbol{x}, \boldsymbol{y}) \in \boldsymbol{\delta}\Omega$$

2D domain decomposition with n x k domains





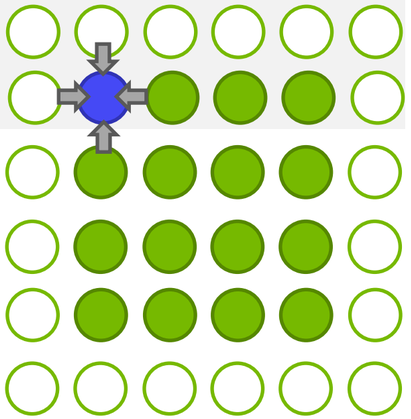
EXAMPLE: JACOBI SOLVER

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

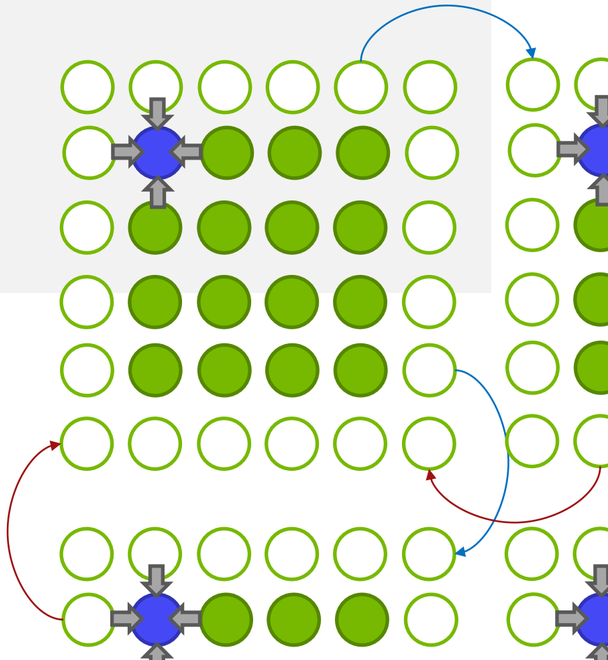
Do Jacobi step:

```
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_new` and `u`

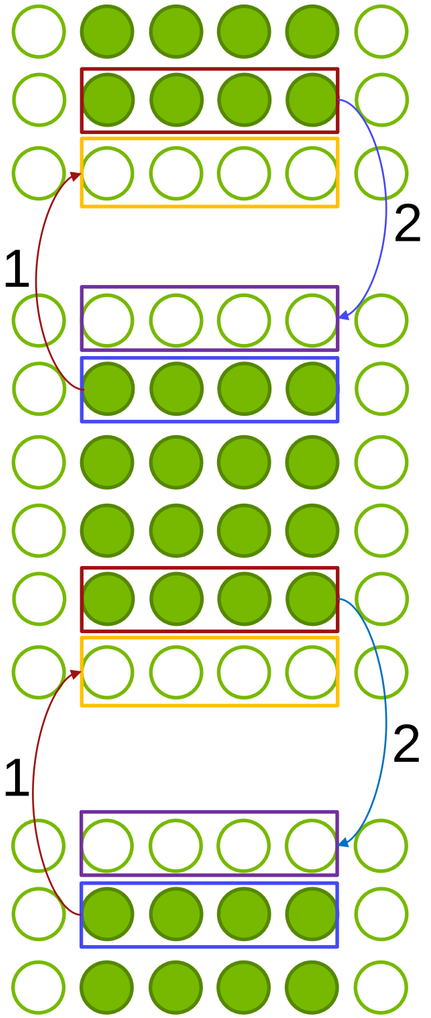
Next iteration



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

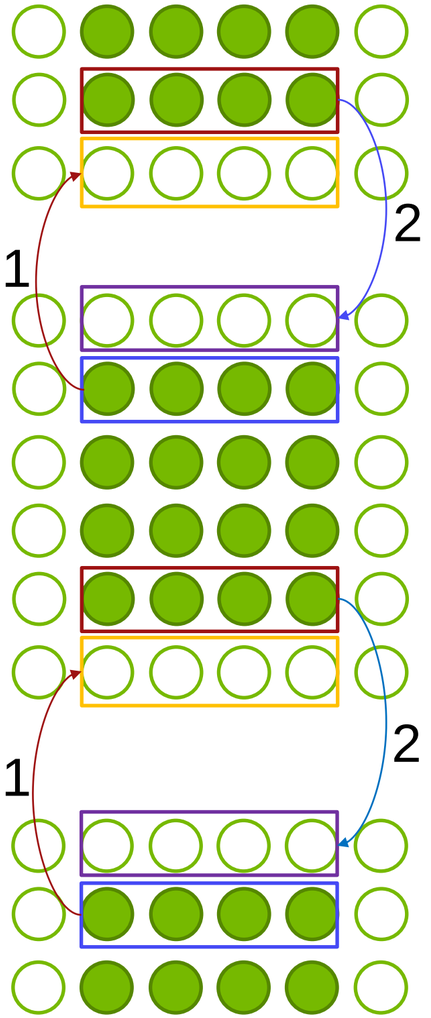
Top/Bottom Halo

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

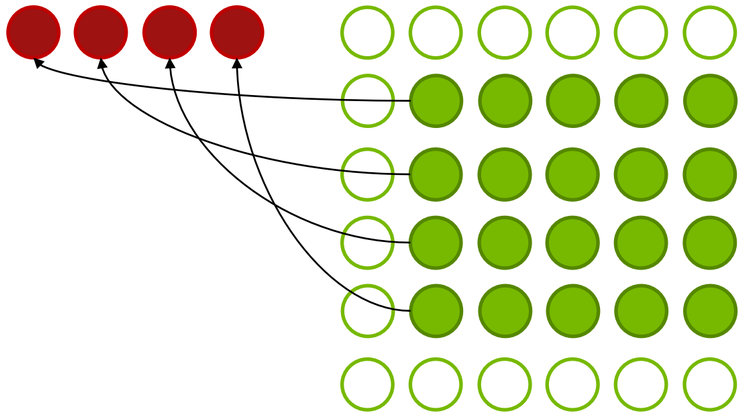
```
#pragma acc update host(u_new[offset_first_row:m-2],u_new[offset_last_row:m-2])
MPI_Sendrecv(u_new+offset_first_row, m-2, MPI_DOUBLE, t_nb, 0,
             u_new+offset_bottom_boundary, m-2, MPI_DOUBLE, b_nb, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
MPI_Sendrecv(u_new+offset_last_row,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

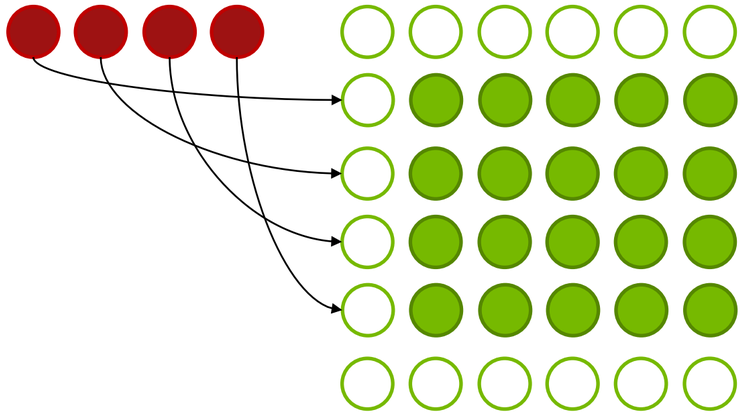
CUDA

```
//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



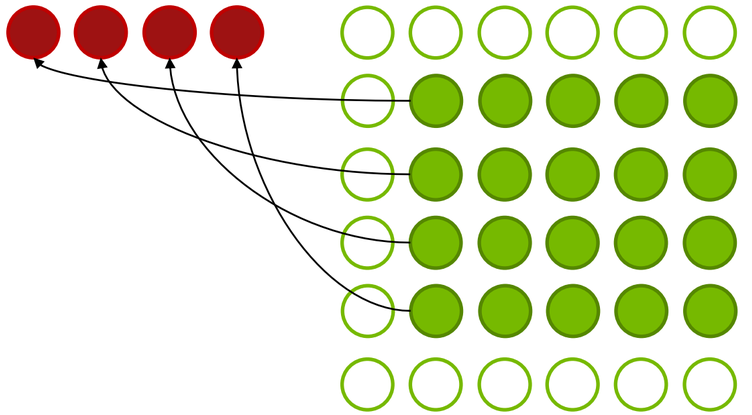


OpenACC

```
//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];
```

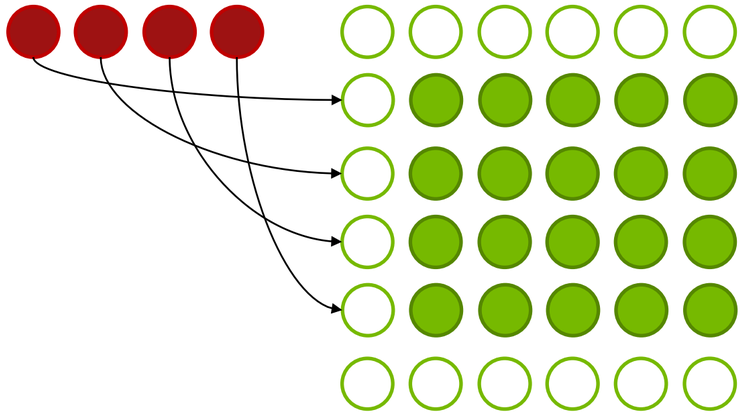
EXAMPLE: JACOBI

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>`

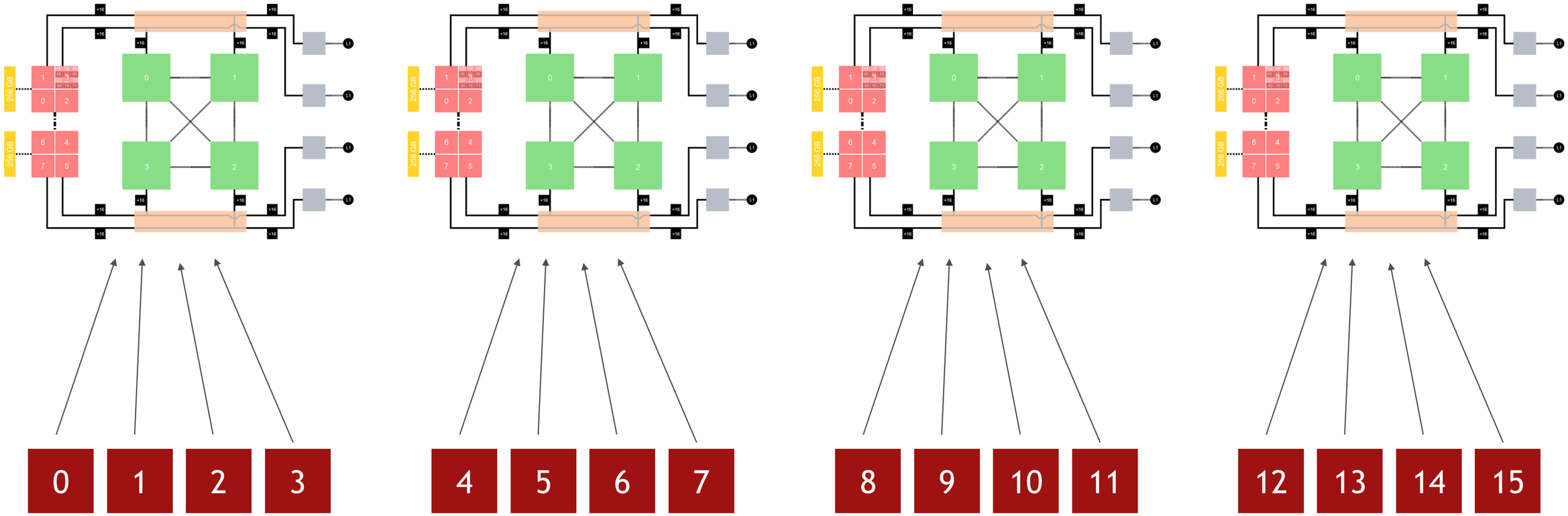
Open MPI: CUDA-aware features are enabled per default

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

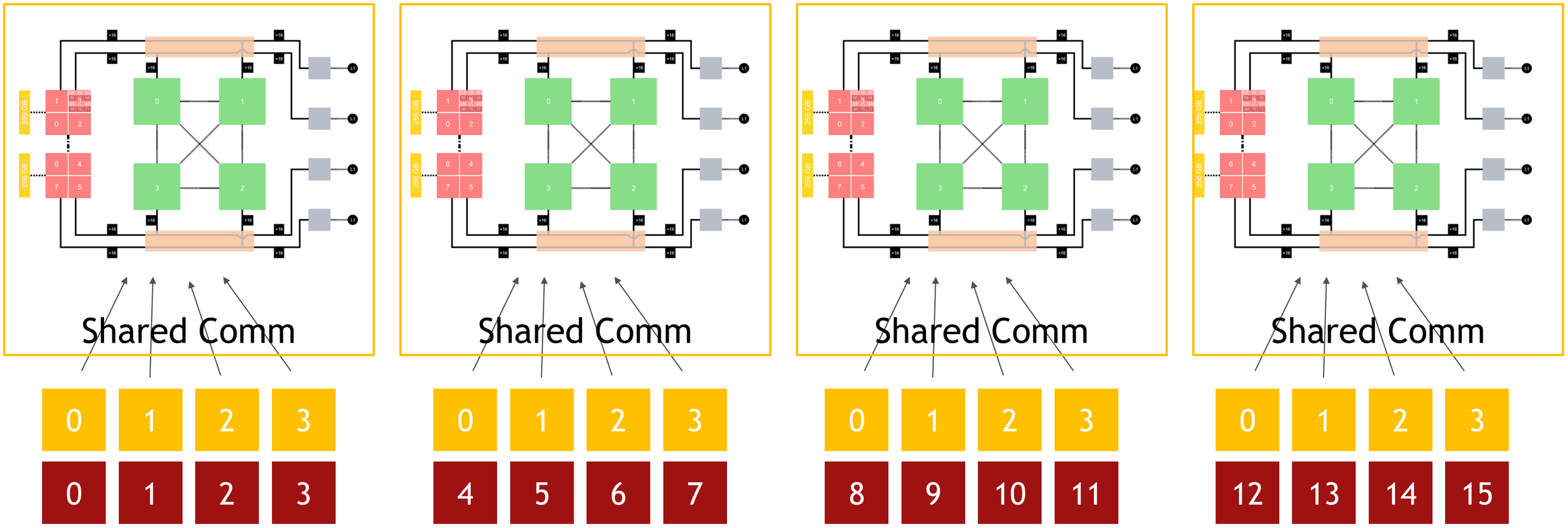


HANDLING MULTI GPU NODES

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);
```

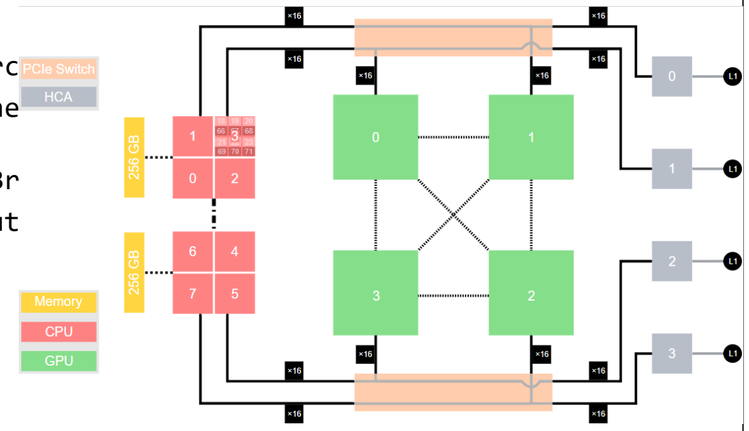
Needed if resource manager handles GPU affinity

UCX TIPS AND TRICKS

Example binding script

```
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} $*
```

```
[kraus1@jwb0007]$ nvidia-smi topo -m
        GPU0    GPU1    GPU2    GPU3    mlx5_0  mlx5_1  mlx5_2  mlx5_3  CPU Affinity    NUMA Affinity
GPU0     X      NV4     NV4     NV4     SYS     PIX     SYS     SYS     18-23,66-71     3
GPU1    NV4      X      NV4     NV4     PIX     SYS     SYS     SYS     6-11,54-59      1
GPU2    NV4     NV4      X      NV4     SYS     SYS     SYS     PIX     42-47,90-95     7
GPU3    NV4     NV4     NV4      X      SYS     SYS     PIX     SYS     30-35,78-83     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

Legend:

  X    = Self
  SYS  = Connection traversing PCIe as well as the SMP interc
  NODE = Connection traversing PCIe as well as the interconne
node
  PHB  = Connection traversing PCIe as well as a PCIe Host Br
  PXB  = Connection traversing multiple PCIe bridges (without
  PIX  = Connection traversing at most a single PCIe bridge
  NV#  = Connection traversing a bonded set of # NVLinks
```



https://apps.fz-juelich.de/jsc/hps/juwels/booster-overview.html

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

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:   1.3159 s, 2 GPUs:   1.3446 s, speedup:
MPI time:   0.0000 s, inter GPU BW:  2484.22 GiB/s
```

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

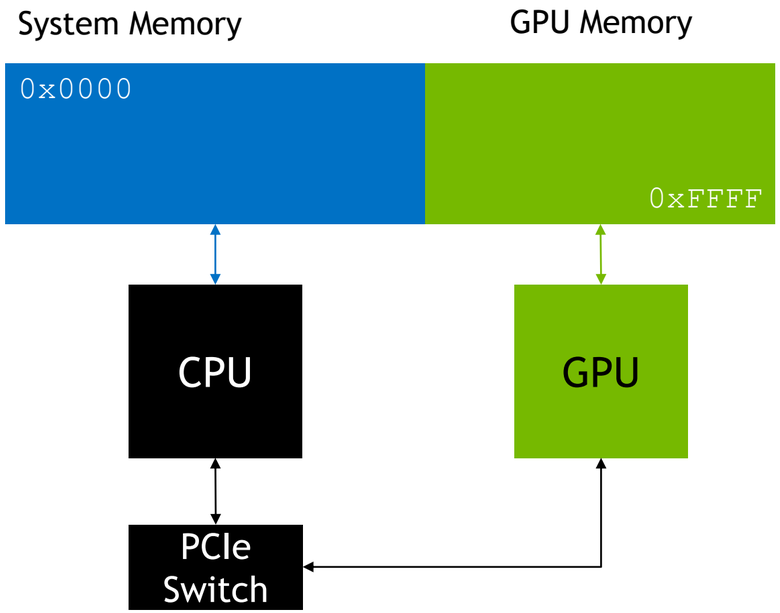
UNIFIED VIRTUAL ADDRESSING

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)

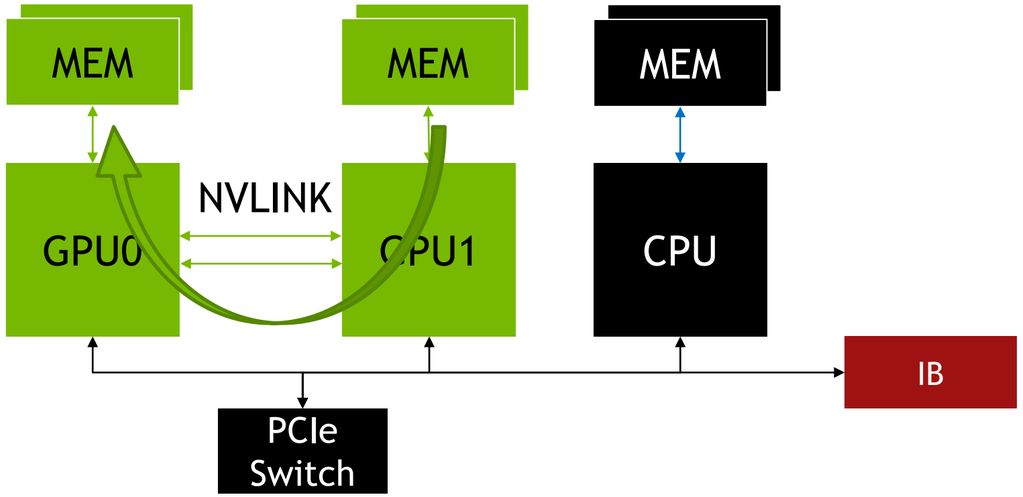
Supported on devices with compute capability 2.0+ for

- 64-bit applications on Linux and Windows (+TCC)



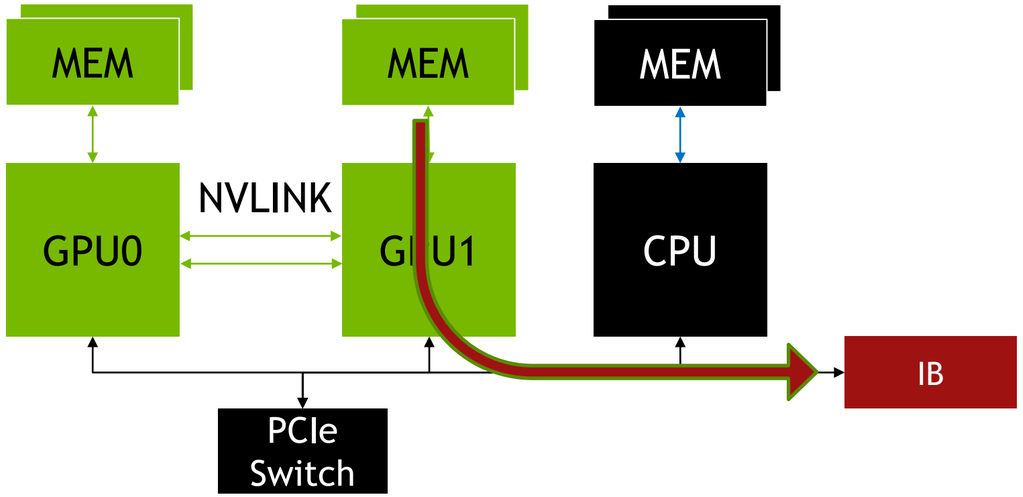
NVIDIA GPUDIRECT

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

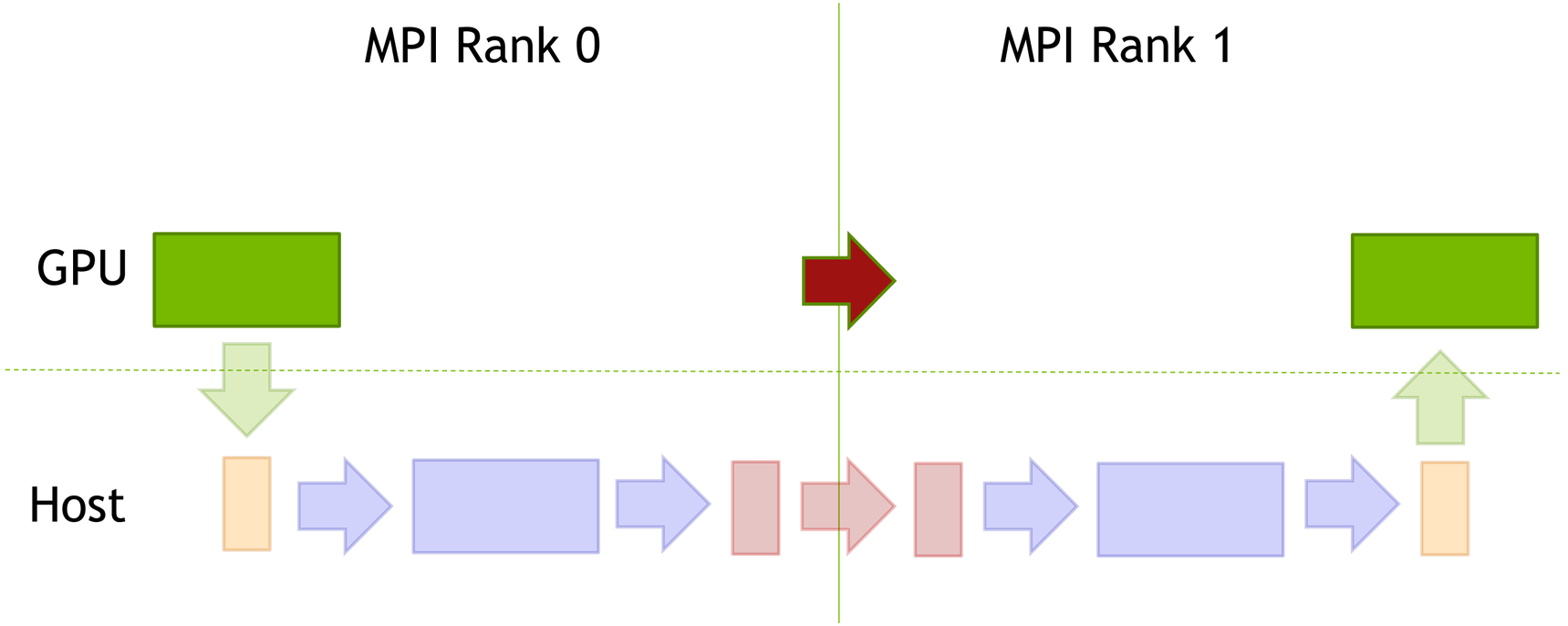
Show how CUDA+MPI works in principle

- 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



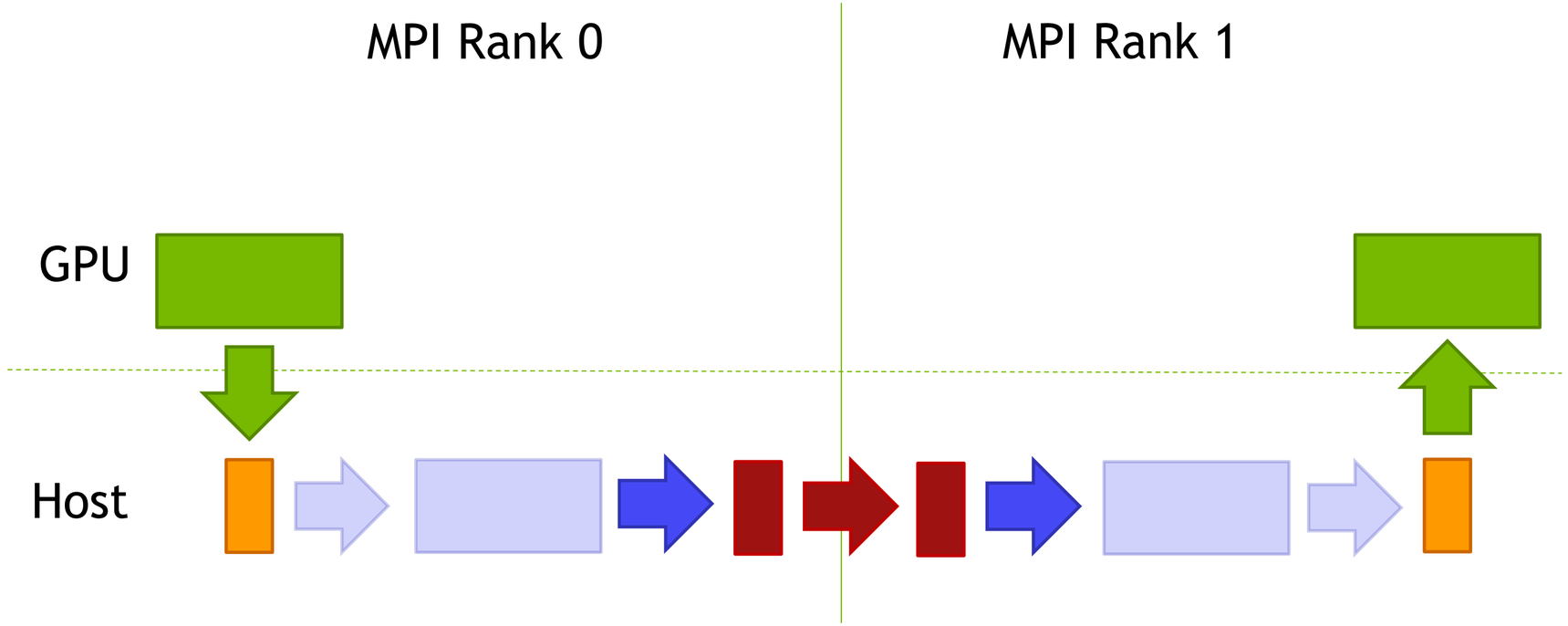

```
MPI_Send(s_buf_d,size,MPI_BYTE,1,tag,MPI_COMM_WORLD);

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

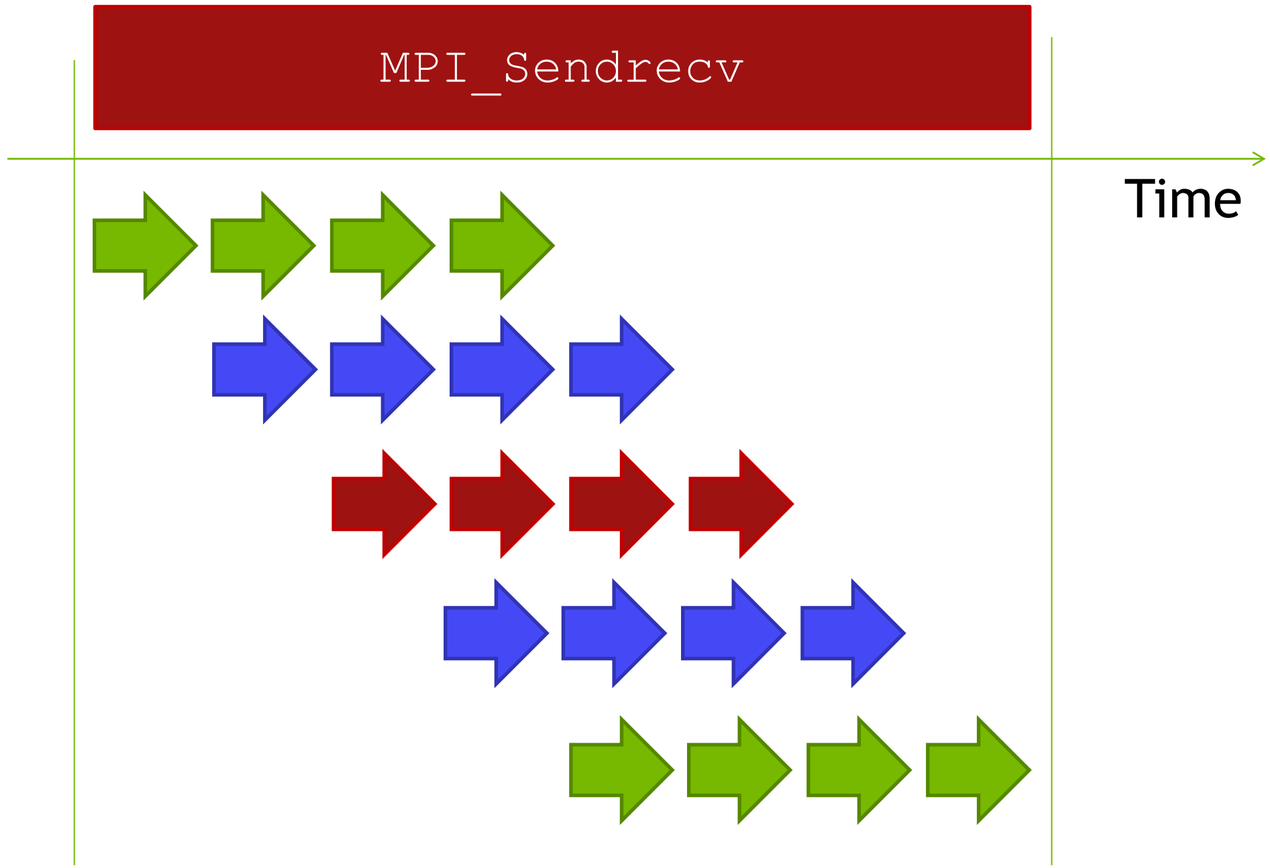
without GPUDirect



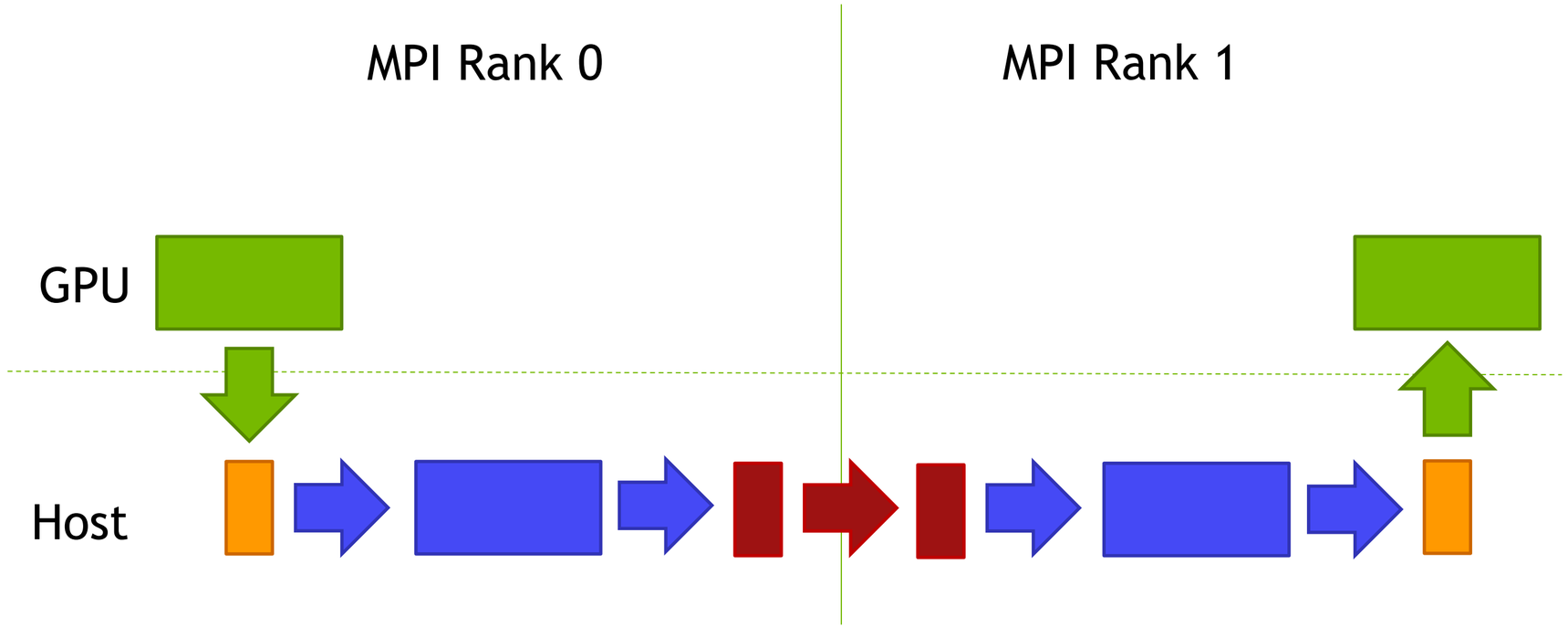

```
MPI_Send(s_buf_h,size,MPI_BYTE,1,tag,MPI_COMM_WORLD);

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

MPI GPU TO REMOTE GPU

without GPUDirect

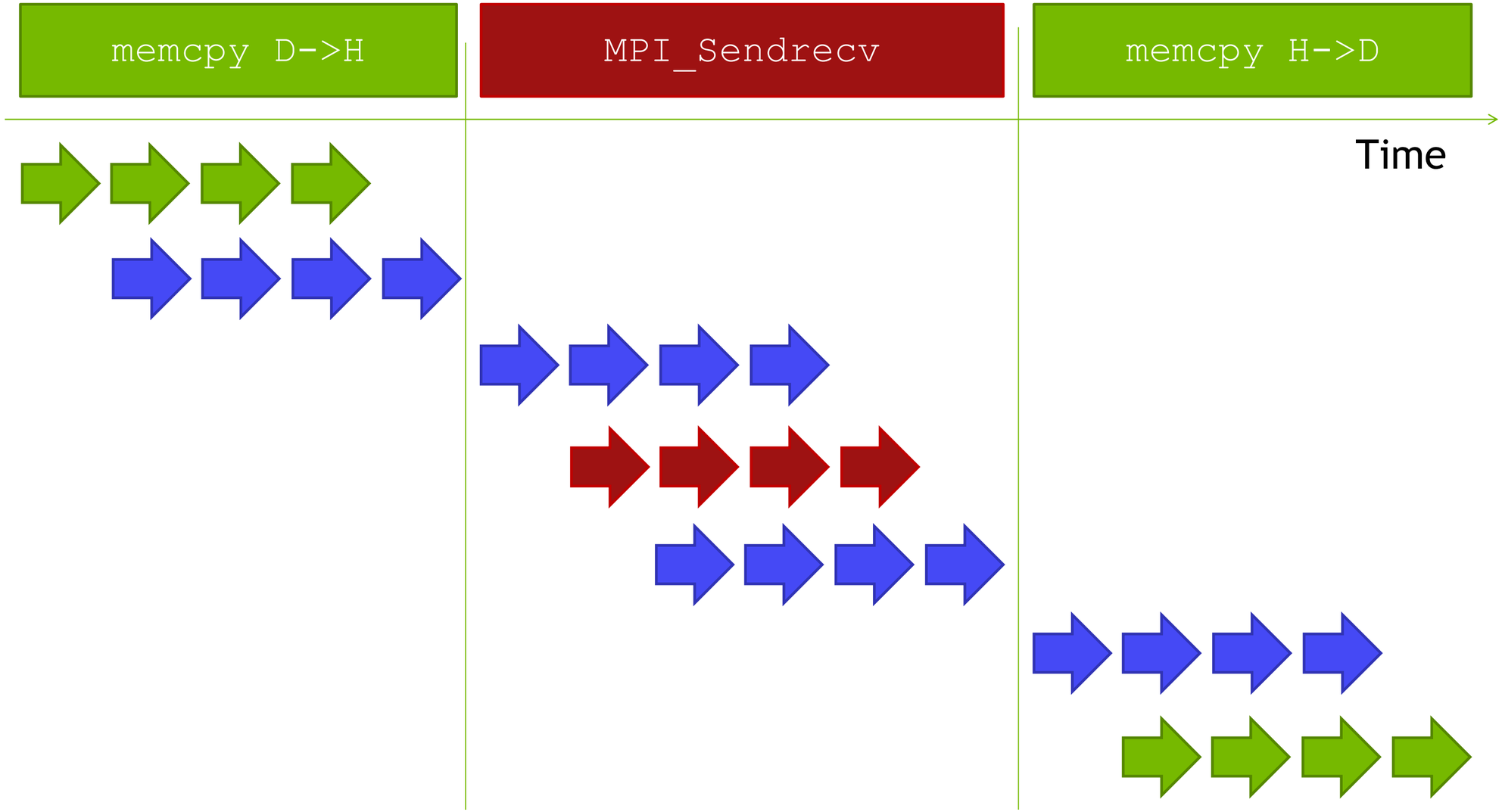


REGULAR MPI GPU TO REMOTE GPU



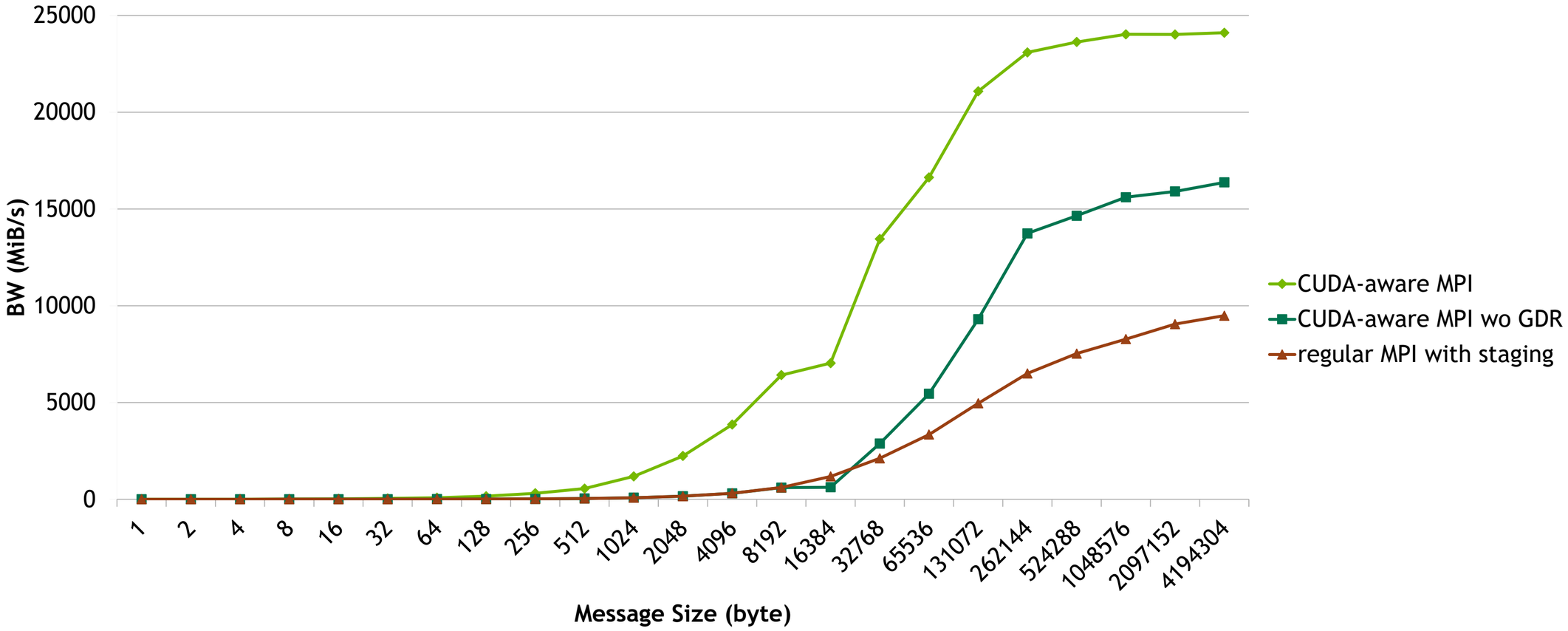

```
cudaMemcpy(s_buf_h,s_buf_d,size,cudaMemcpyDeviceToHost);
MPI_Send(s_buf_h,size,MPI_BYTE,1,tag,MPI_COMM_WORLD);

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

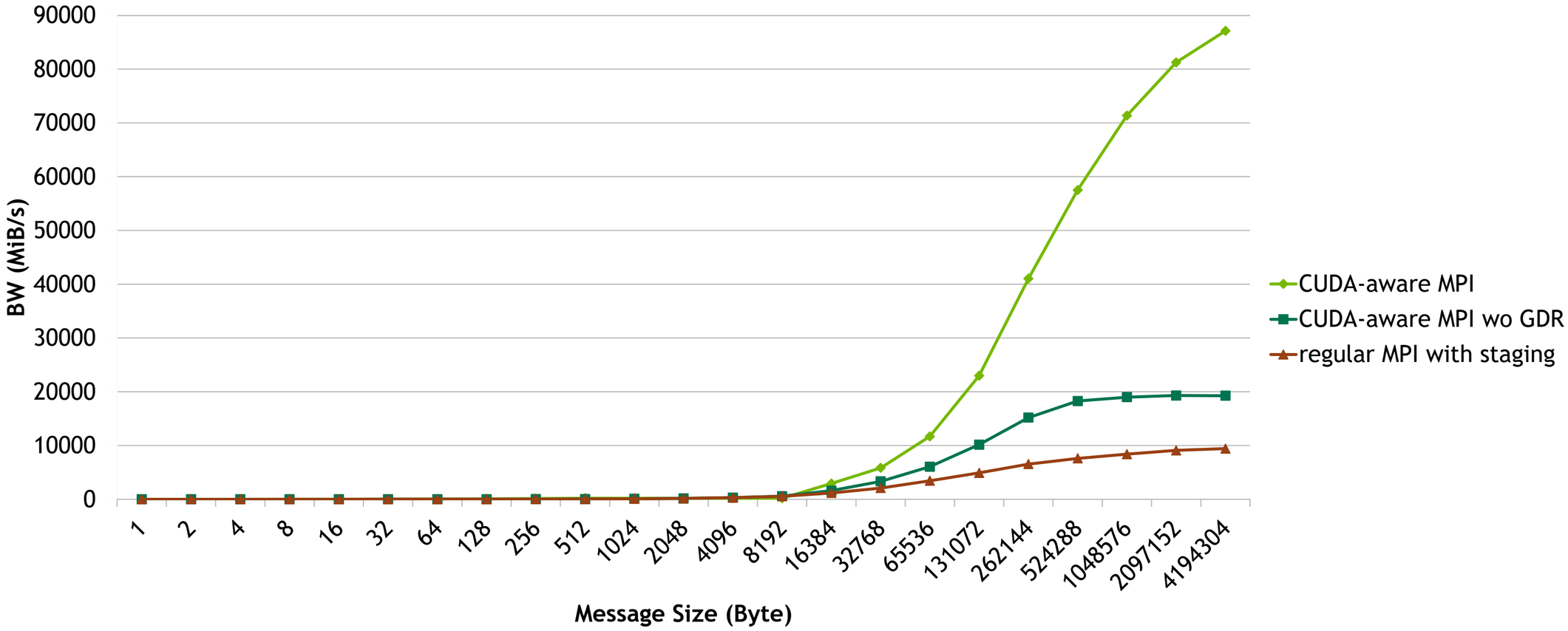


Latency (1 byte) 4.27 us 24.56 us 25.64 us

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+InfiniBand+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 **CO**mmunication a**L**gorithms (HCOL) – CPU and GPU data

NCCL – GPU data*

and hardware

Scalable **H**ierarchical **A**ggregation and **R**eduction **P**rotocol (SHARP – in Network/Switch)

Collectives **O**ffload **R**esource **E**ngine (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 **F**abric **C**ollective **A**ccelerator (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_WOLRD only
thread-funneled	Any communicator	COMM_WORLD only
thread-serialized	Any communicator	COMM_WORLD only
thread-multiple	COMM_WORLD only	COMM_WORLD only

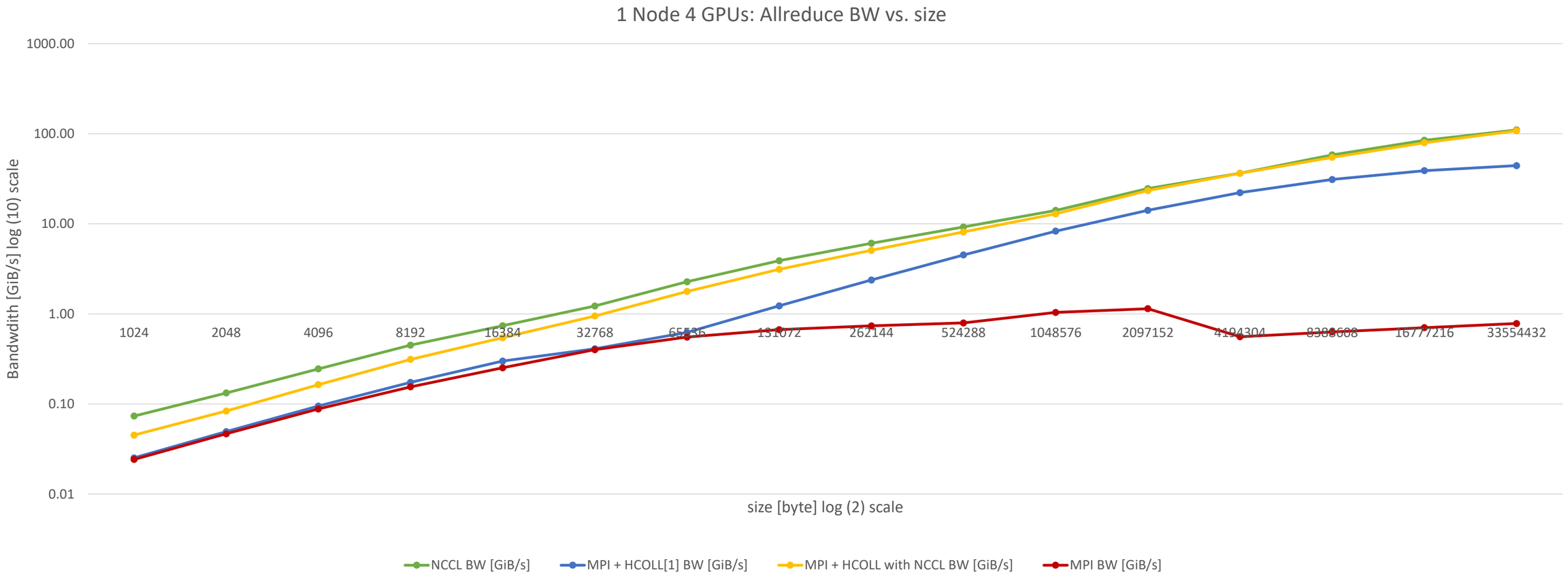
Limitations apply to HCOLL and similar limitations will apply to UCC!

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

CAVEAT: HCOLL does not enforce these restrictions so apps my 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



[1] With HCOLL_GPU_STAGING_MAX_THRESHOLD=65536

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

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

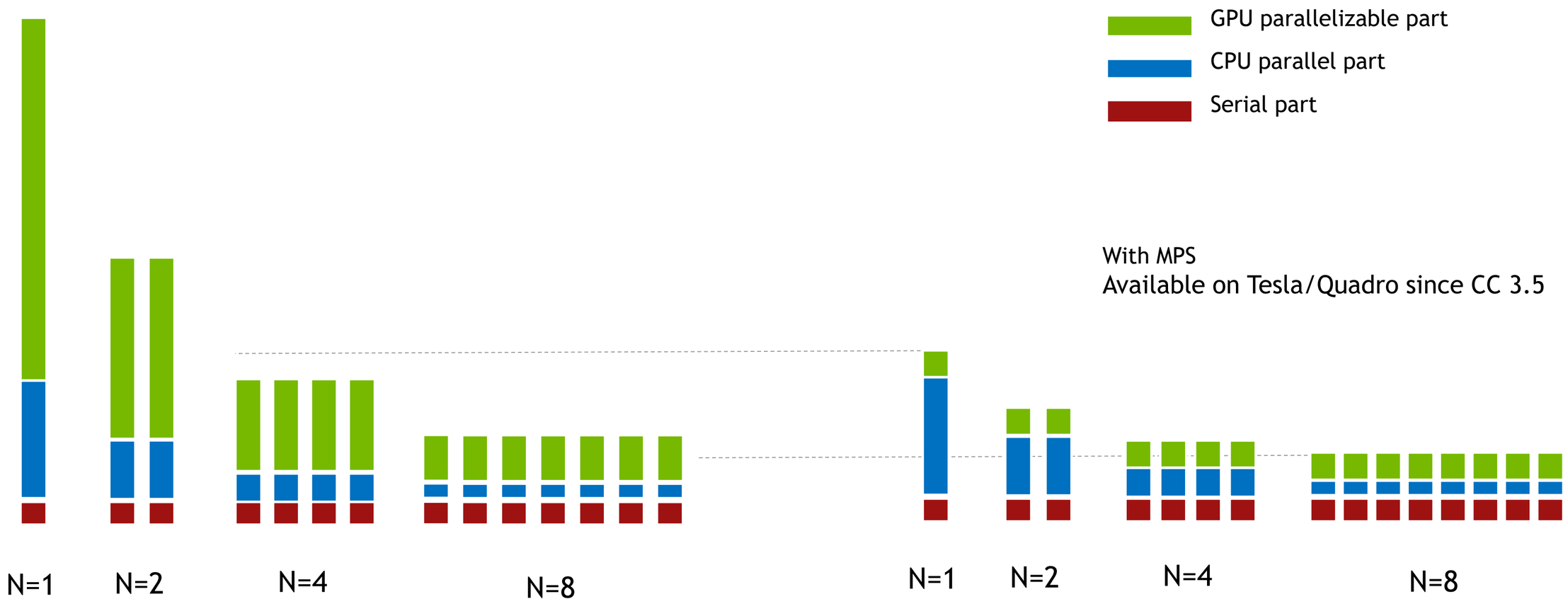
GPU acceleration in phases

- Proof of concept prototype, …
- Great speedup at kernel level

Application performance misses expectations

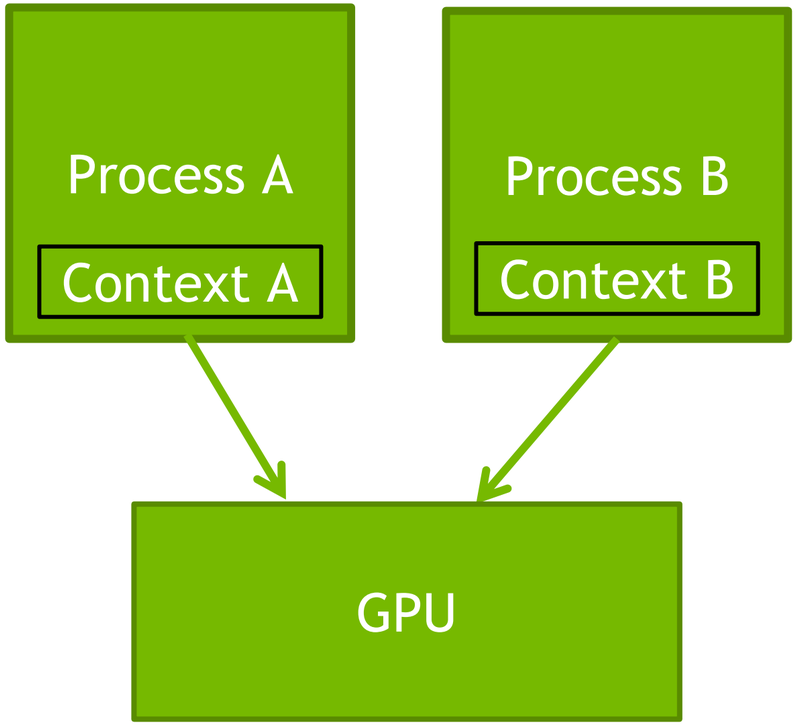
MULTI PROCESS SERVICE (MPS)

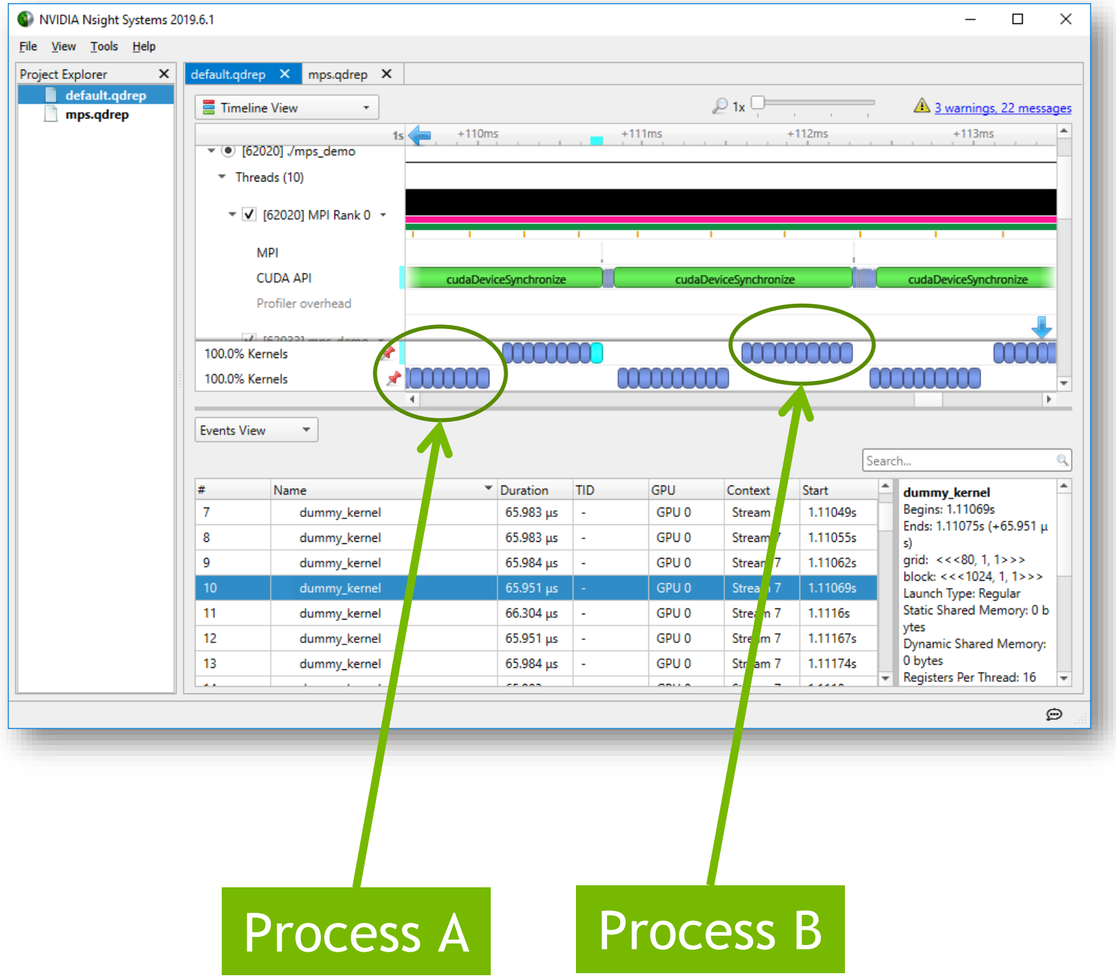
For Legacy MPI Applications



PROCESSES SHARING GPU WITHOUT MPS

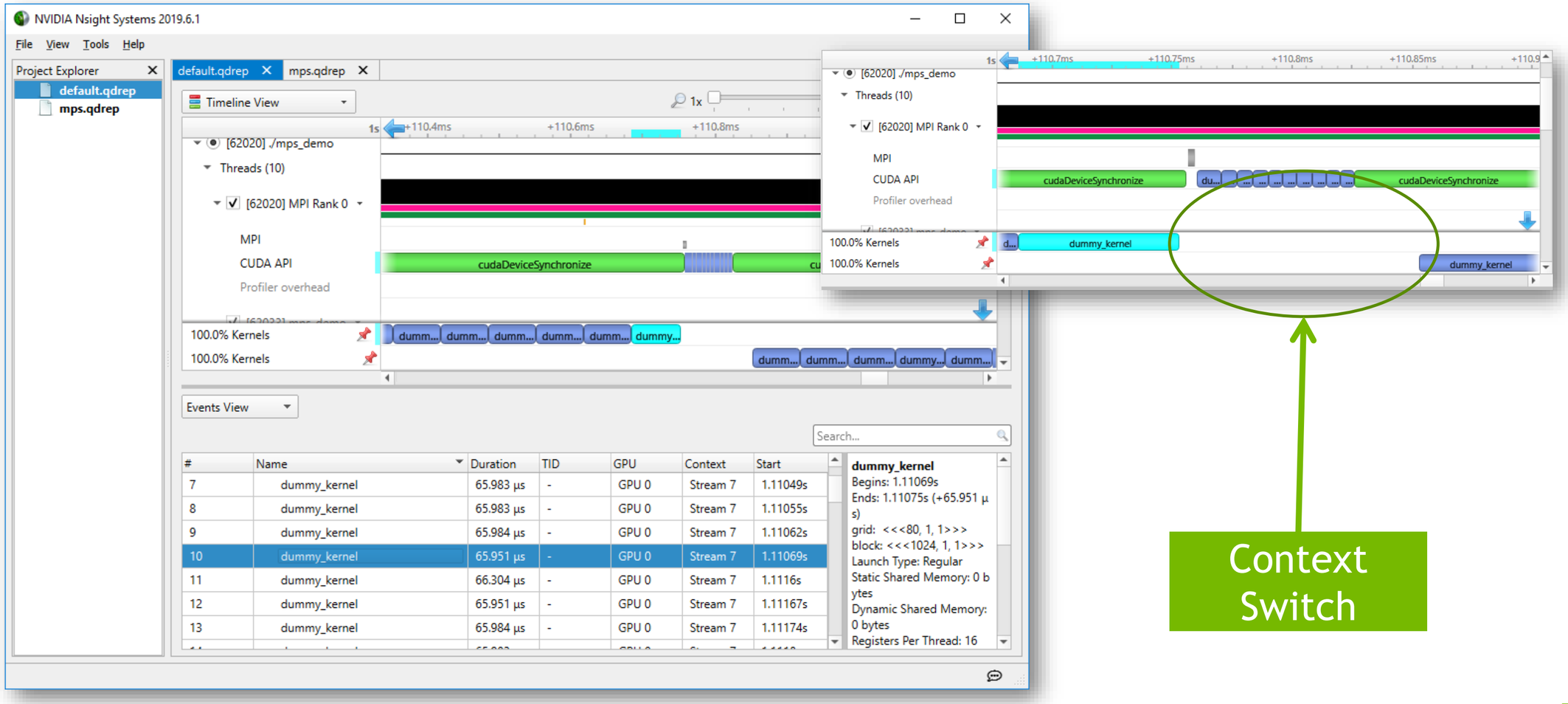
No Overlap





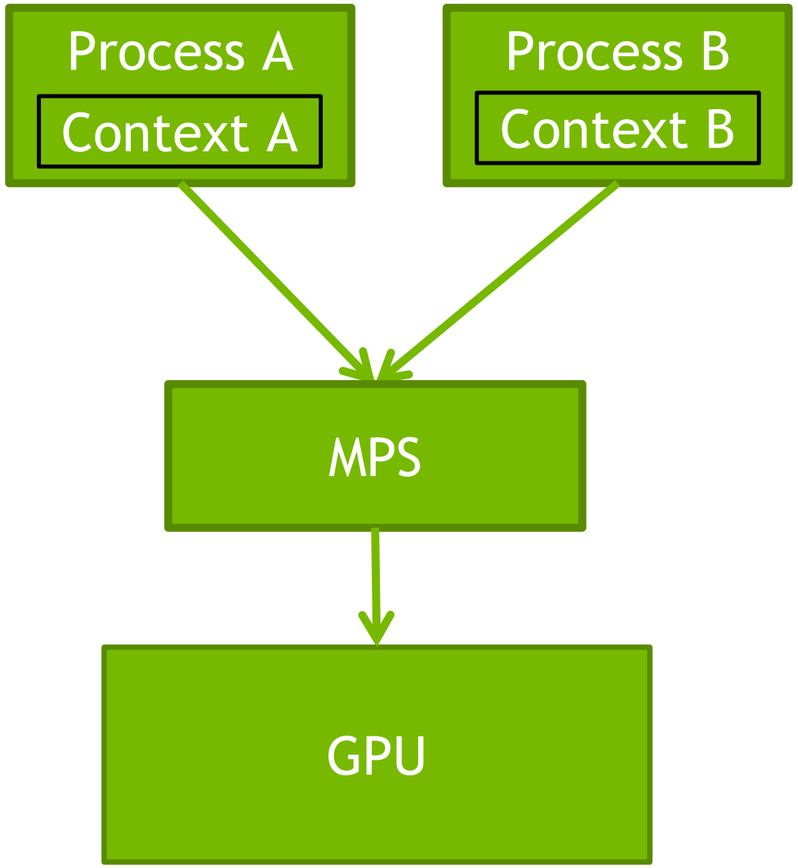
PROCESSES SHARING GPU WITHOUT MPS

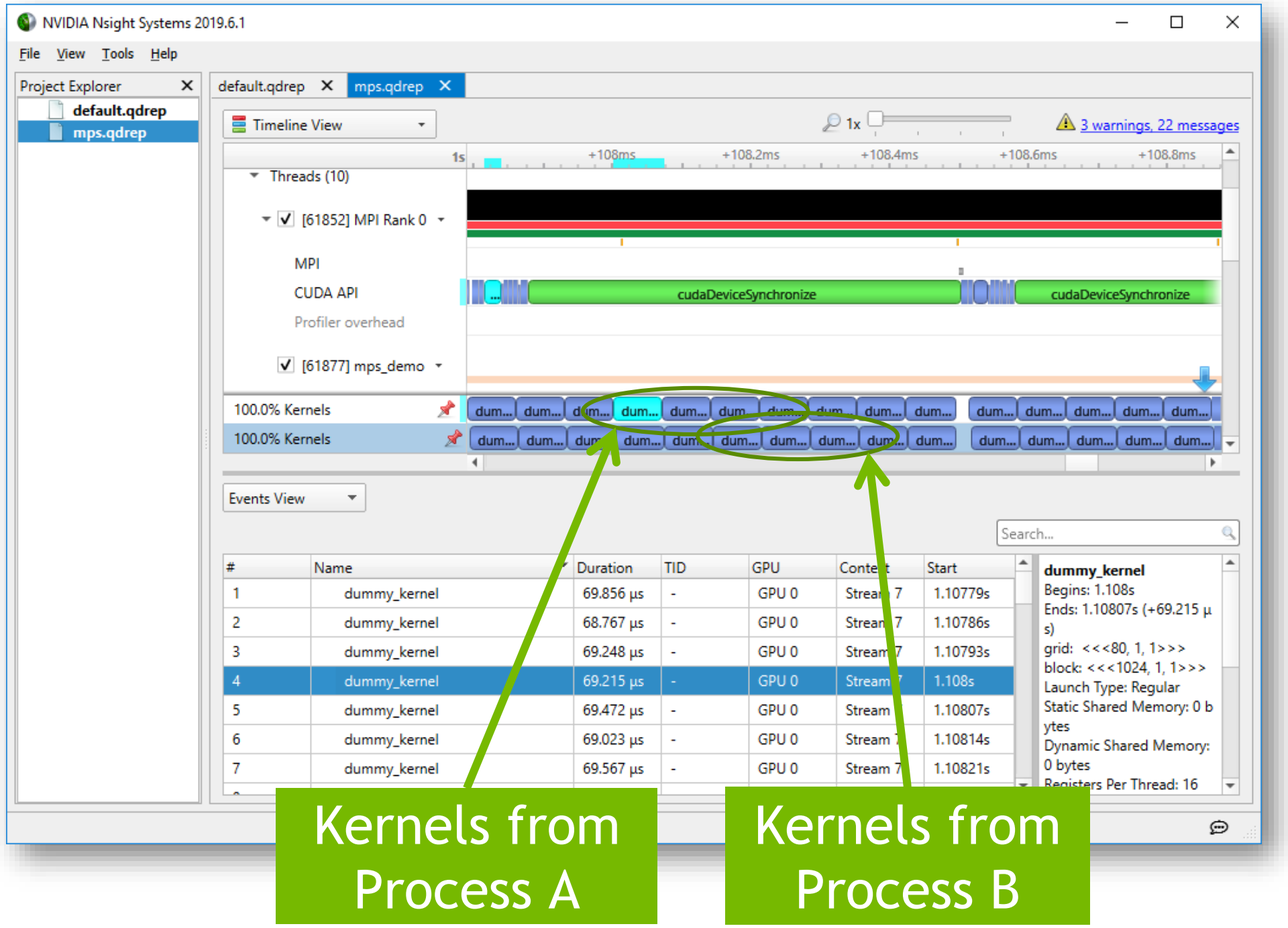
Context Switch Overhead



PROCESSES SHARING GPU WITH MPS

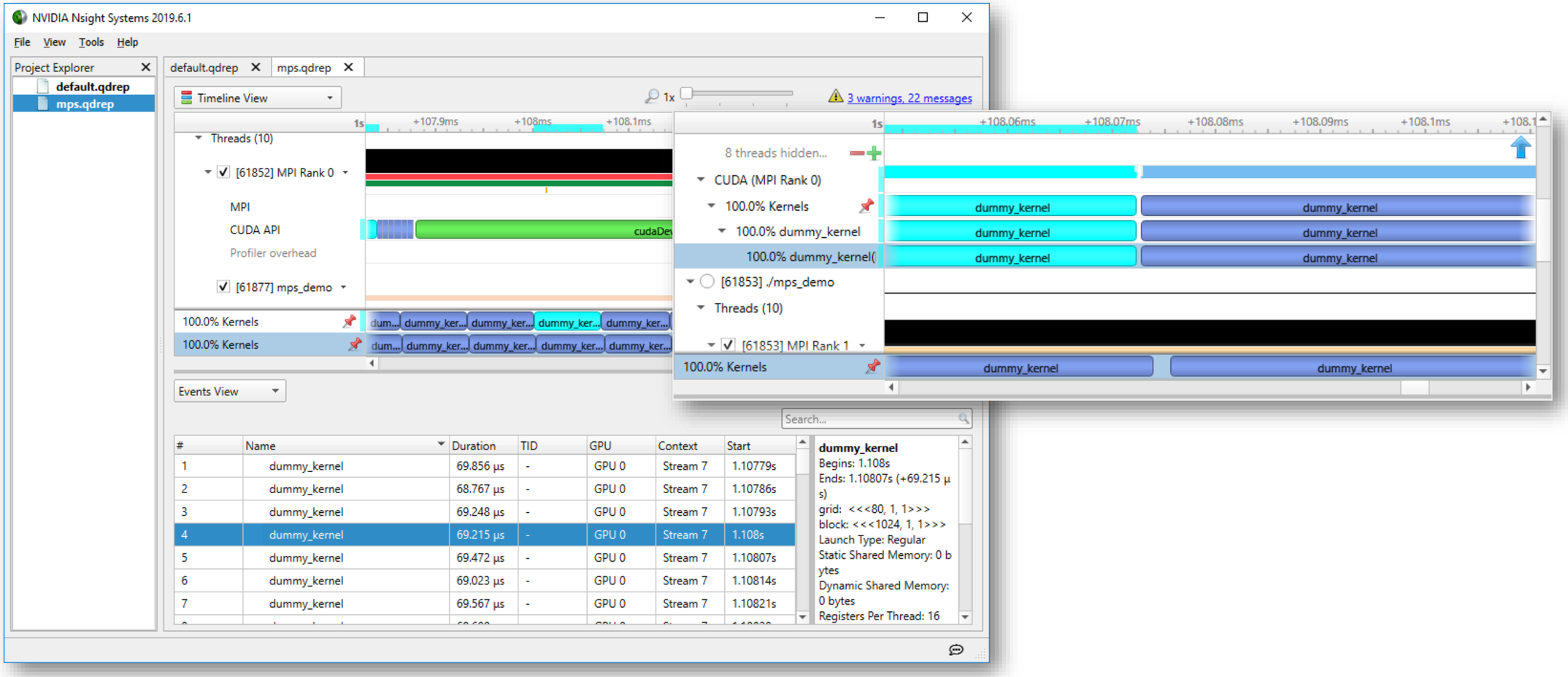
Maximum Overlap



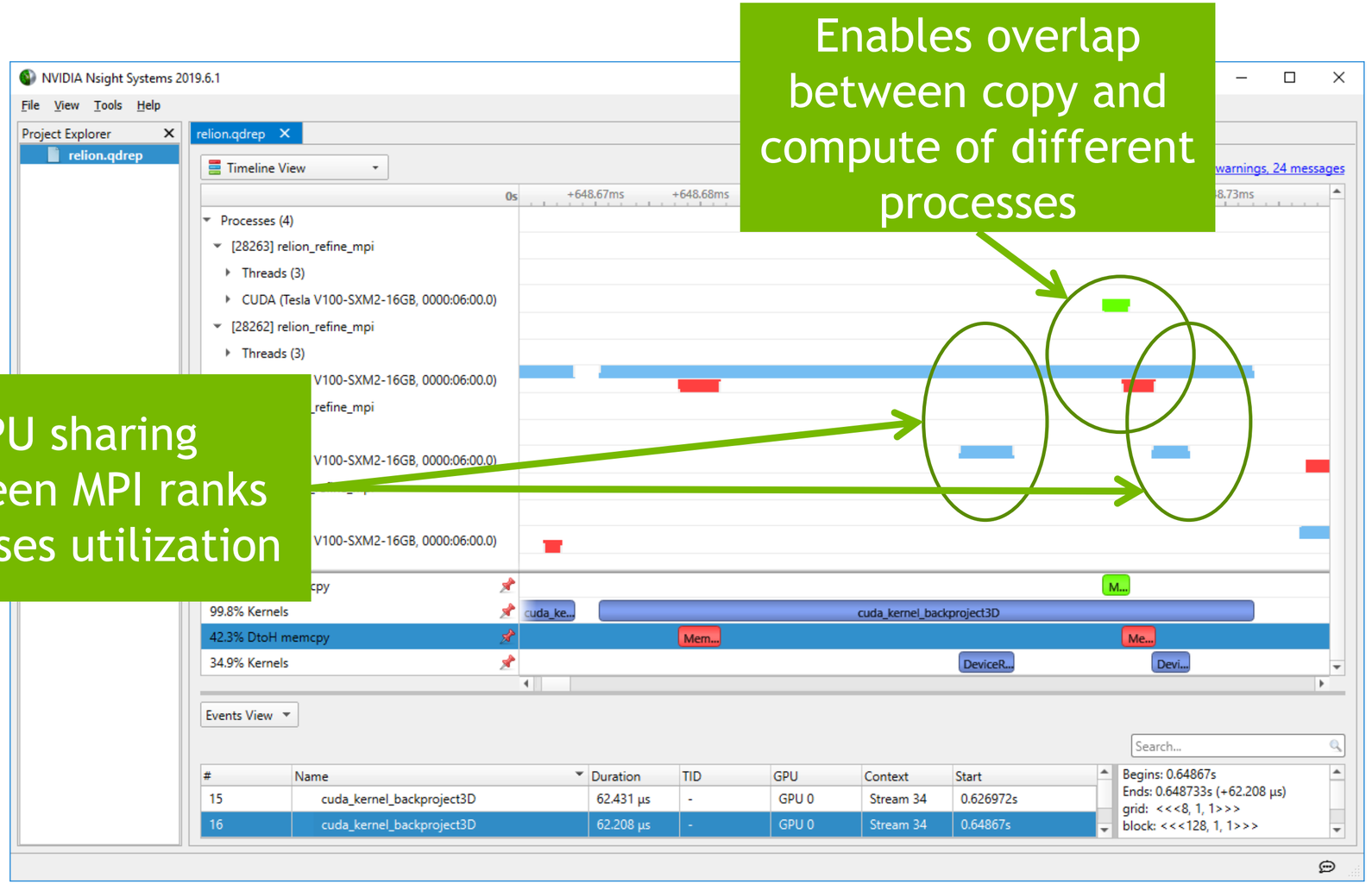


PROCESSES SHARING GPU WITH MPS

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
```

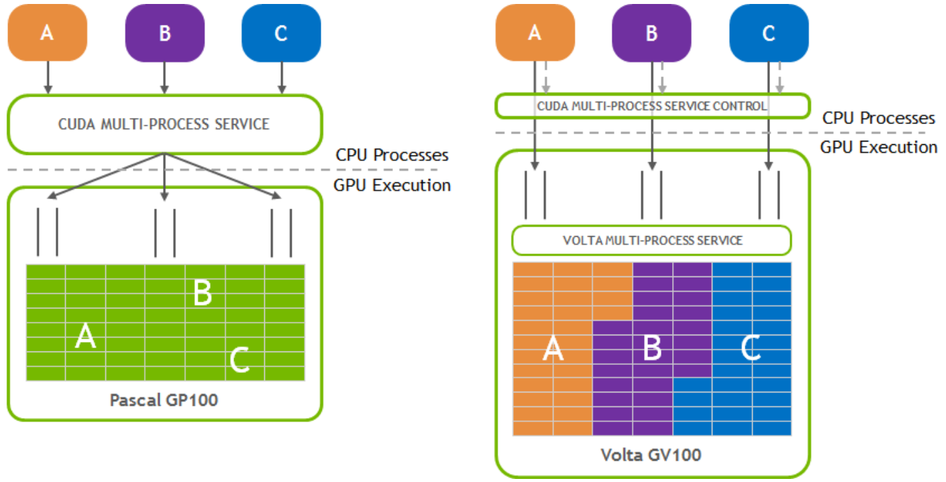
MPS: IMPROVEMENTS WITH VOLTA

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>
```

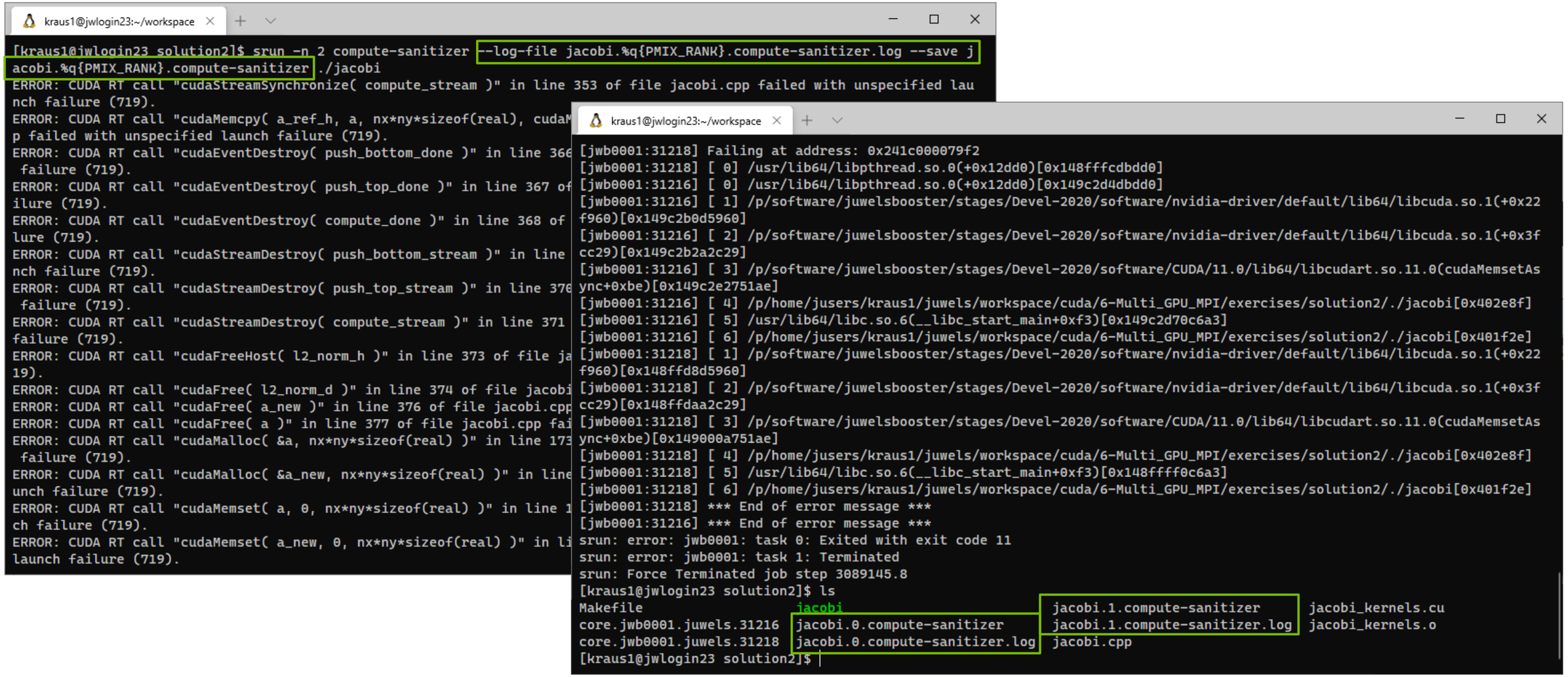
OpenMPI: `OMPI_COMM_WORLD_RANK`

MVAPICH2: `MV2_COMM_WORLD_RANK`

Slurm with PMI-2: `PMI_RANK`

Slurm with PMIX: `PMIX_RANK`

ERROR CHECKING WITH COMPUTE SANITIZER



ERROR CHECKING WITH COMPUTE SANITIZER

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

kraus1@jwlogin23:~/workspace

```
[kraus1@jwlogin23 solution2]$ compute-sanitizer --read jacobi.0.compute-sanitizer
```

kraus1@jwlogin23:~/workspace

```
=========                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]$
```

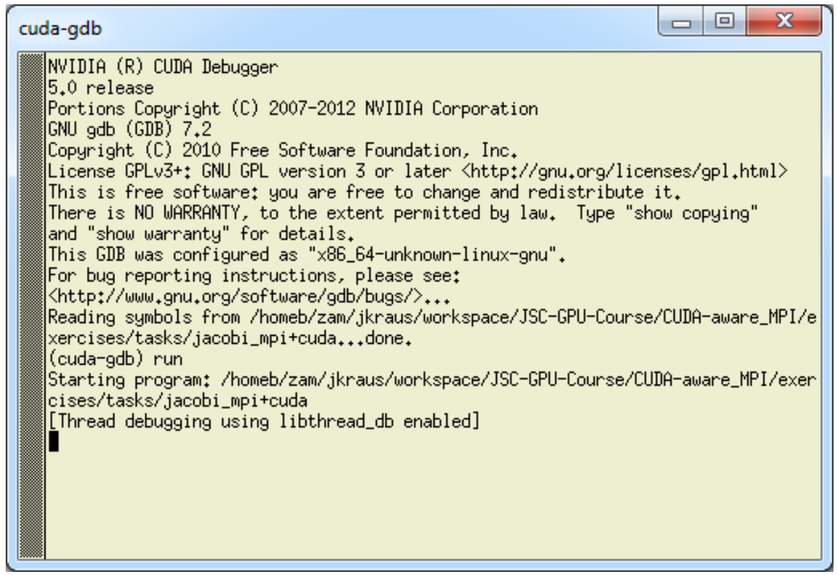
DEBUGGING MPI+CUDA APPLICATIONS

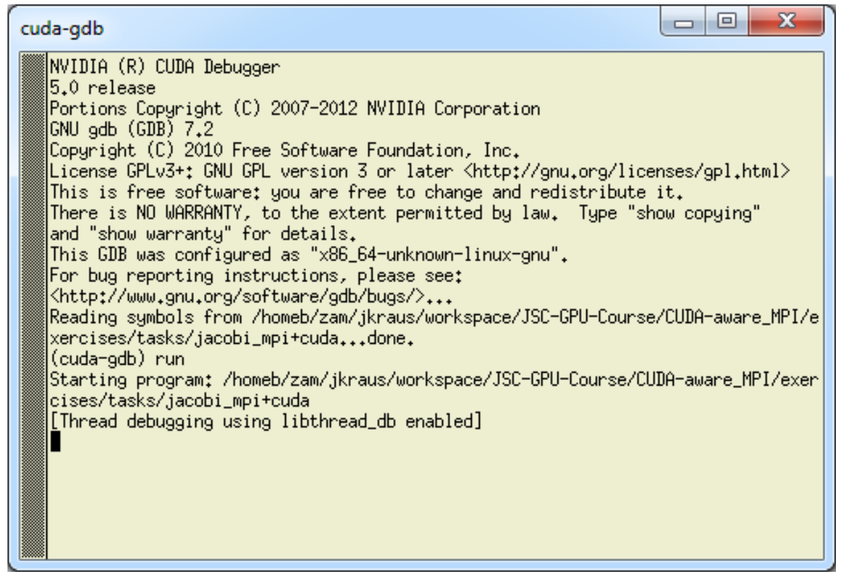
Using cuda-gdb with MPI Applications

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>
```





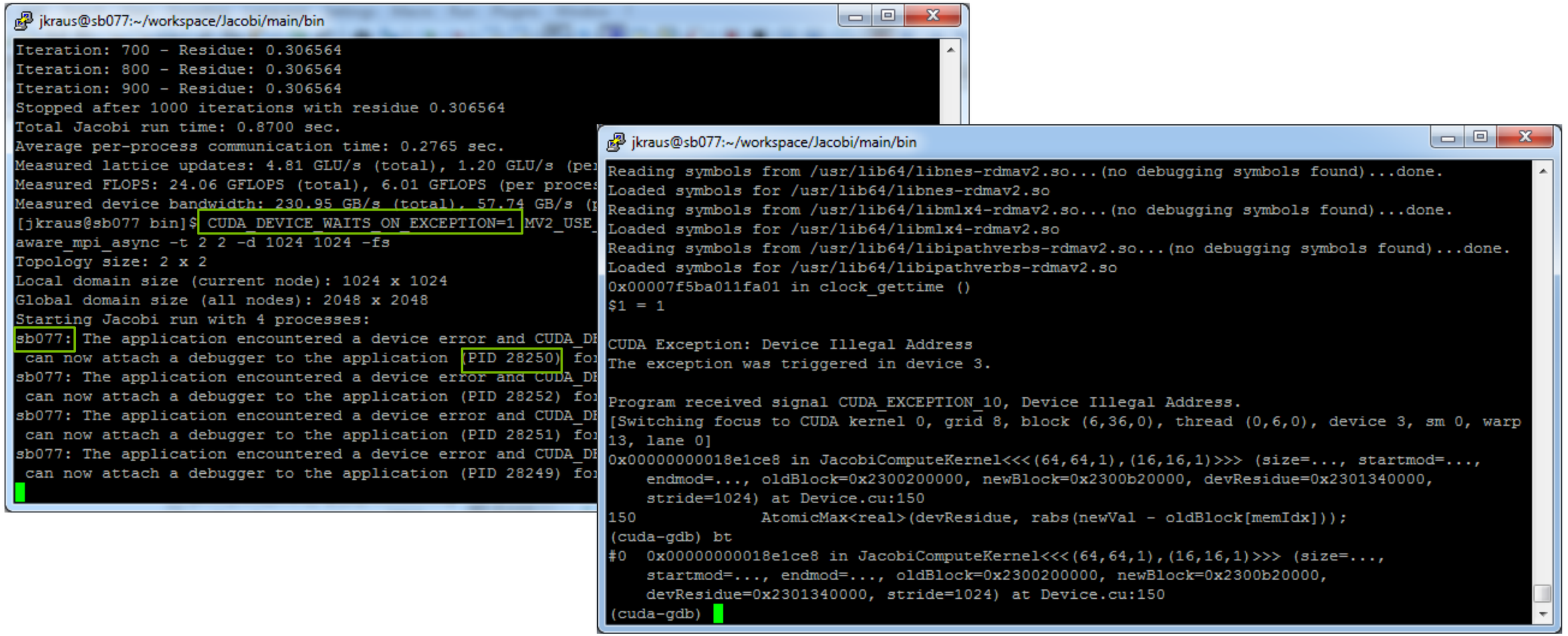
DEBUGGING MPI+CUDA APPLICATIONS

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



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)

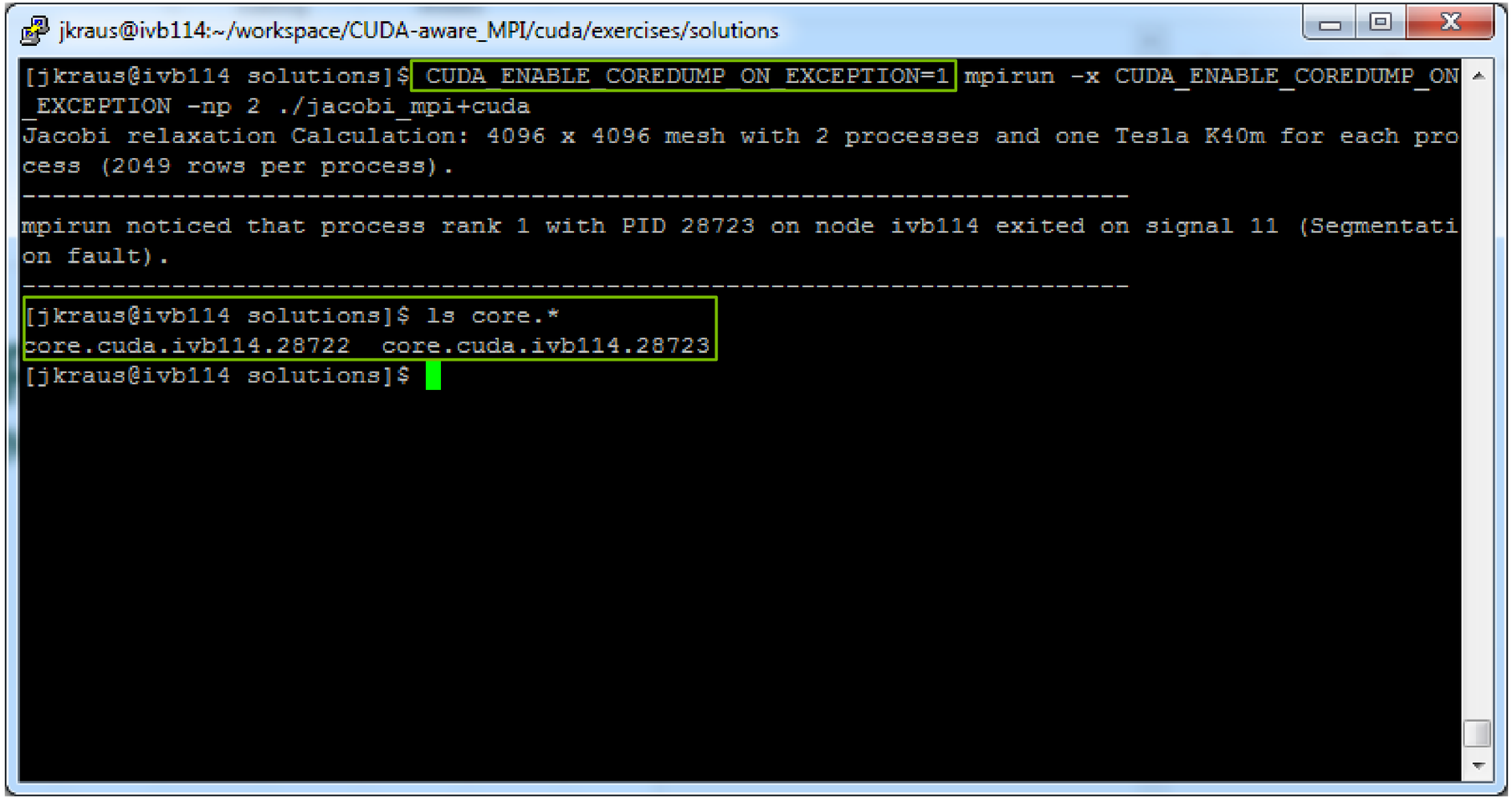
`CUDA_COREDUMP_FILE:` Specify name of core dump file

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

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

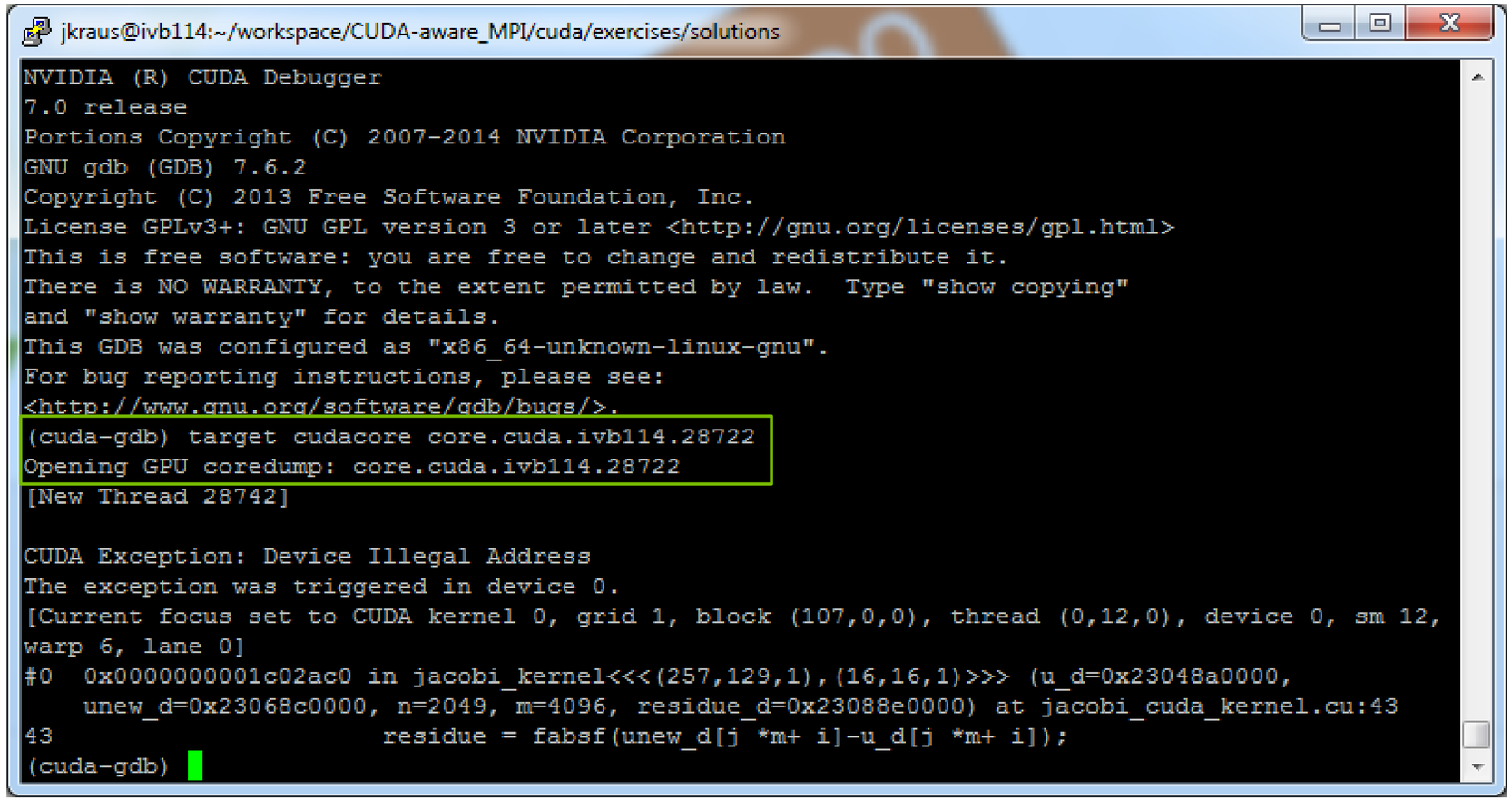
DEBUGGING MPI+CUDA APPLICATIONS

CUDA_ENABLE_COREDUMP_ON_EXCEPTION



DEBUGGING MPI+CUDA APPLICATIONS

CUDA_ENABLE_COREDUMP_ON_EXCEPTION

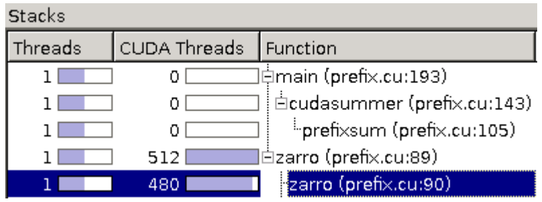


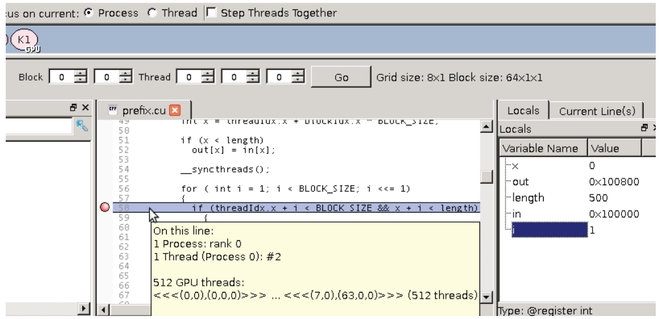
DEBUGGING MPI+CUDA APPLICATIONS

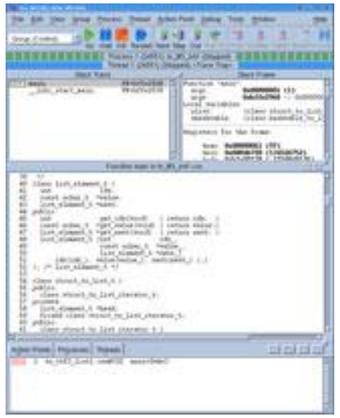
Third Party Tools

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

prom.nvidia.com - PuTTY

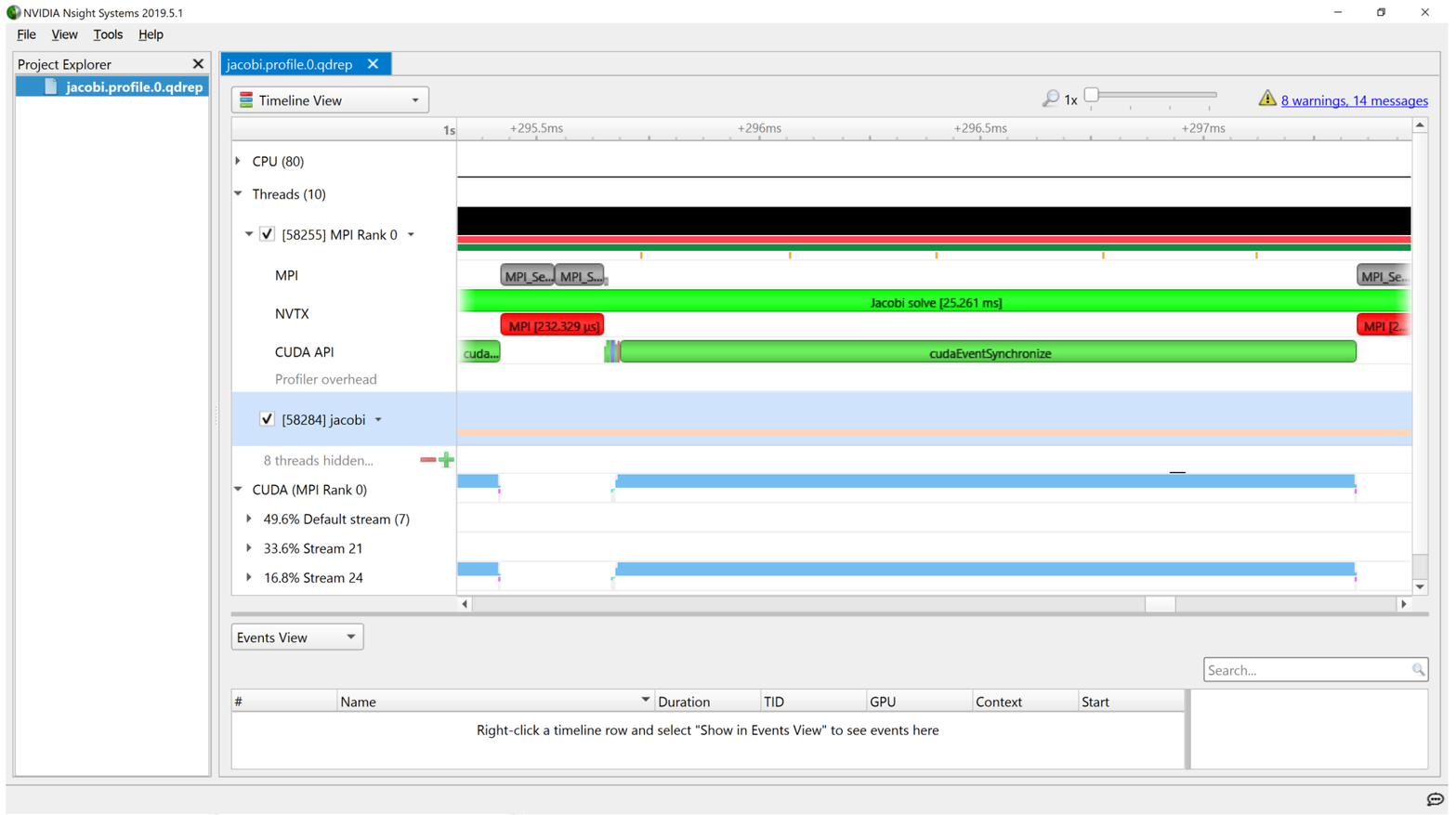
```
jkraus@prm-dgx-01:~/workspace/multi-gpu-pro
pi,cuda --mpi-impl openmpi -o jacobi.profil
WARNING: OpenMPI or MPICH tracing relies on

**** 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
```

prom.nvidia.com - PuTTY

```
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$
```

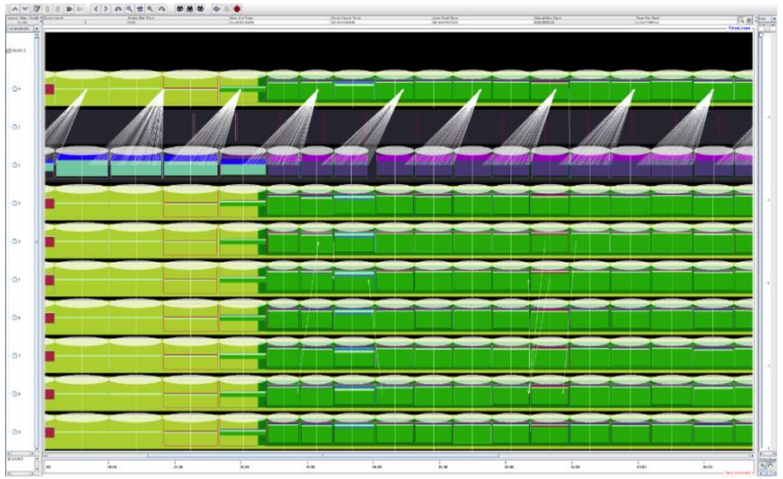
PROFILING MPI+CUDA APPLICATIONS

Using Nsight Systems



PROFILING MPI+CUDA APPLICATIONS

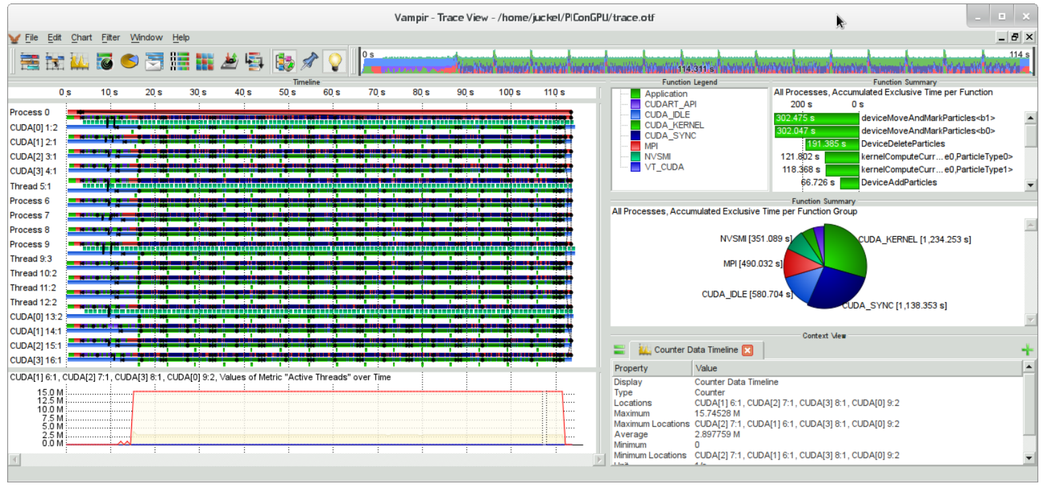
Third Party Tools



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

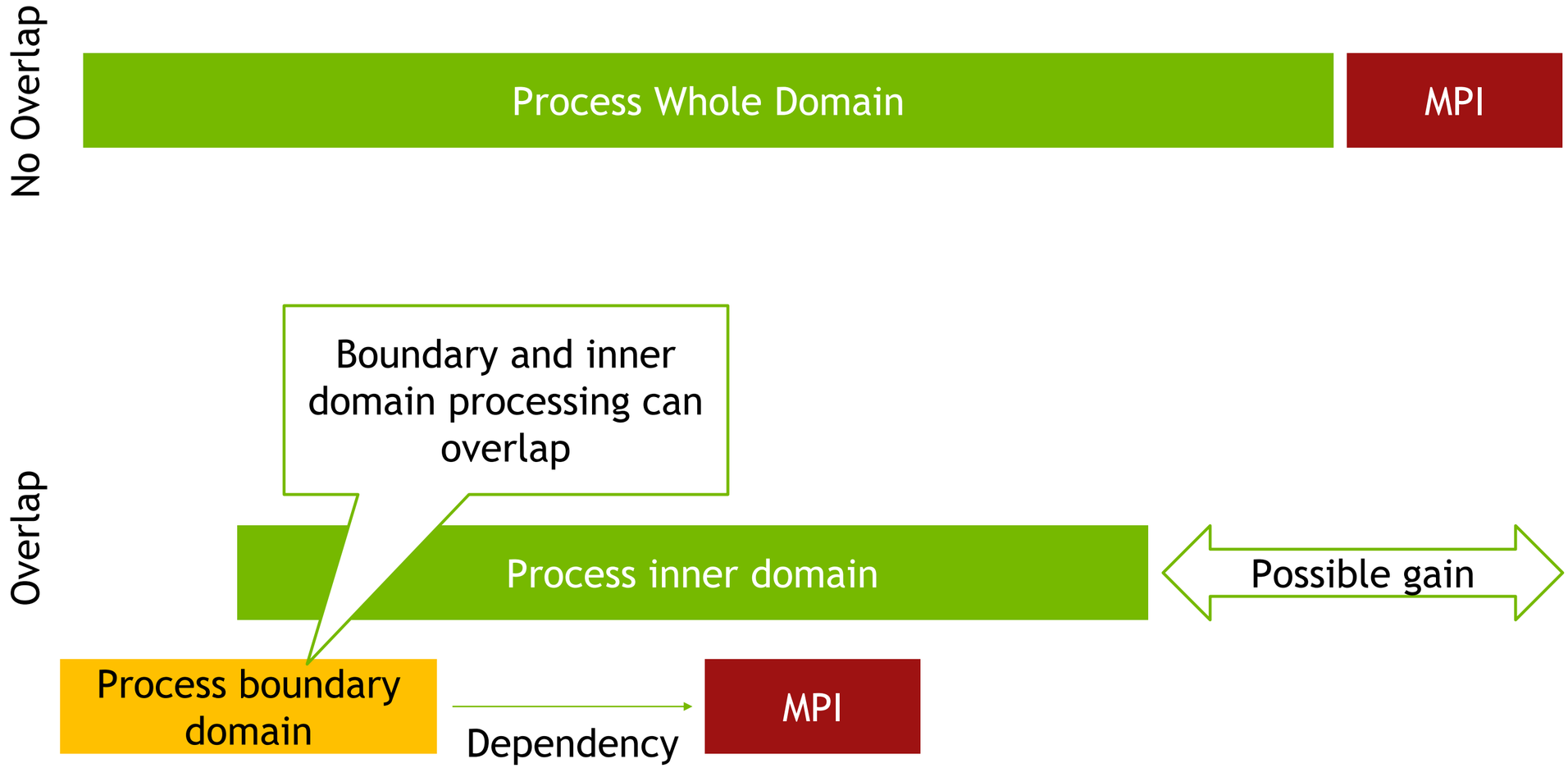
```
#pragma acc host_data use_device ( u_new ) {
MPI_Sendrecv(u_new+offset_first_row, m-2, MPI_DOUBLE, t_nb, 0,
             u_new+offset_bottom_boundary, m-2, MPI_DOUBLE, b_nb, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
MPI_Sendrecv(u_new+offset_last_row, m-2, MPI_DOUBLE, b_nb, 1,
             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_DOUBL
  MPI_Irecv(u_new+offset_bottom_boundary,m-2,MPI
  MPI_Isend(u_new+offset_last_row,m-2,MPI_DOUBLE
  MPI_Isend(u_new+offset_first_row,m-2,MPI_DOUBLE
}
MPI_Waitall(4, t_b_req, MPI_STATUSES_IGNORE);
```

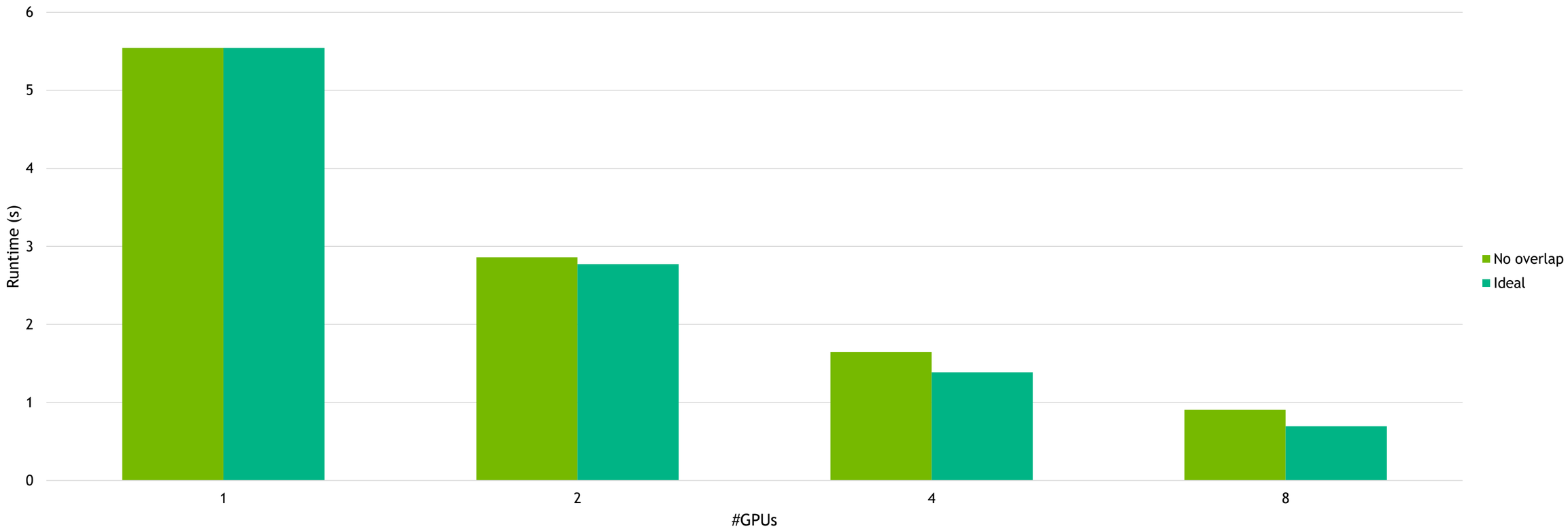
Gives MPI more opportunities to build efficient piplines

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
```

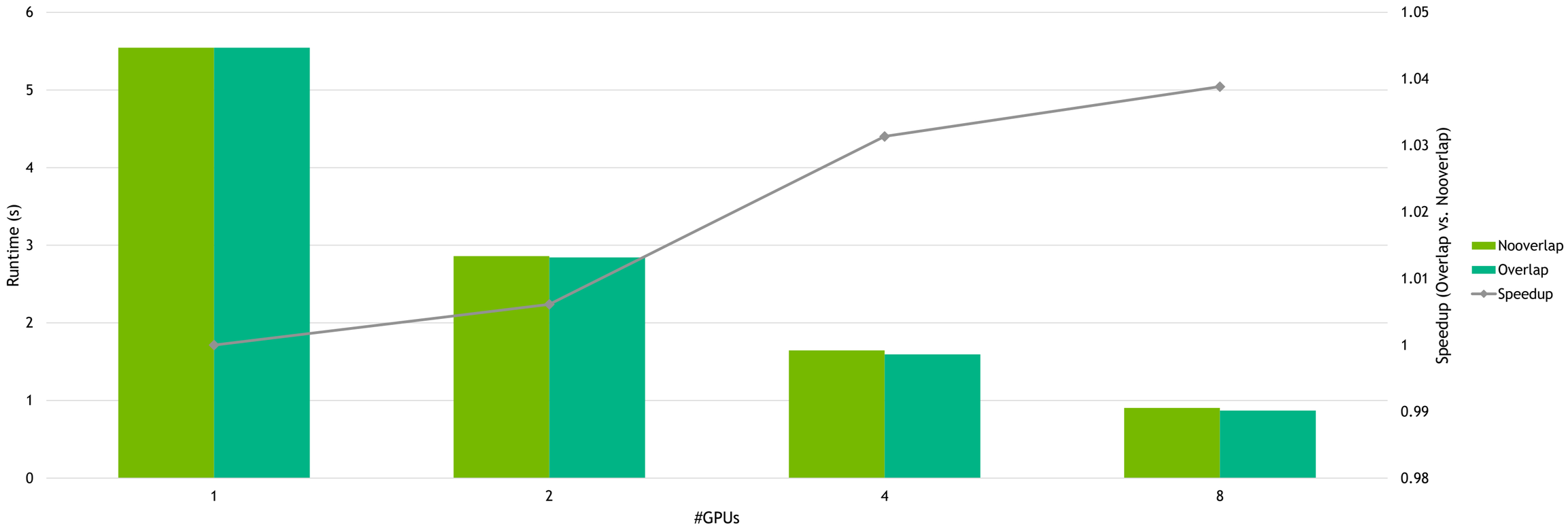
COMMUNICATION + COMPUTATION OVERLAP

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`

Look for TODOs

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

```
$ 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:

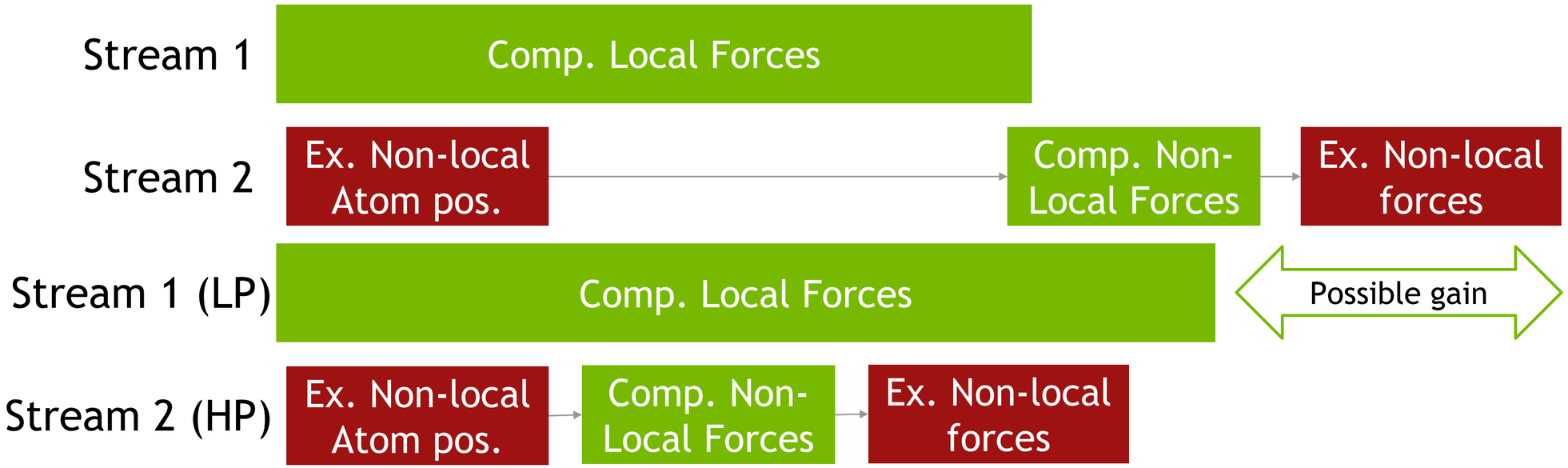
`run`:	run `poisson2d` (default)
`poisson2d`:	build `poisson2d` binary
`profile`:	profile with `pgprof`
`*.solution`:	same as above with solution (`poisson2d.solution.*`)

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

Unified Memory-aware: CUDA-aware MPI with support for Unified Memory

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

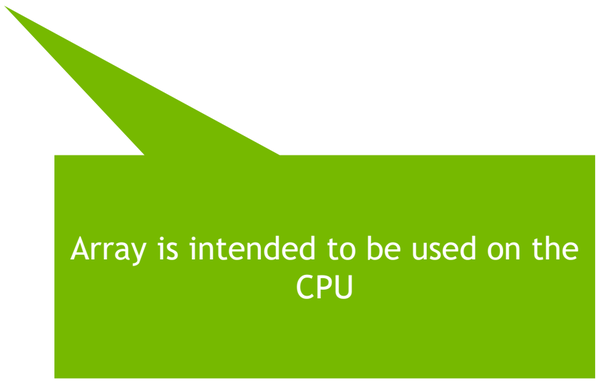
Remark: Data Usage Hints are available since CUDA 8, but currently not evaluated by any Unified Memory-aware MPI implementation.

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);

}
```



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!