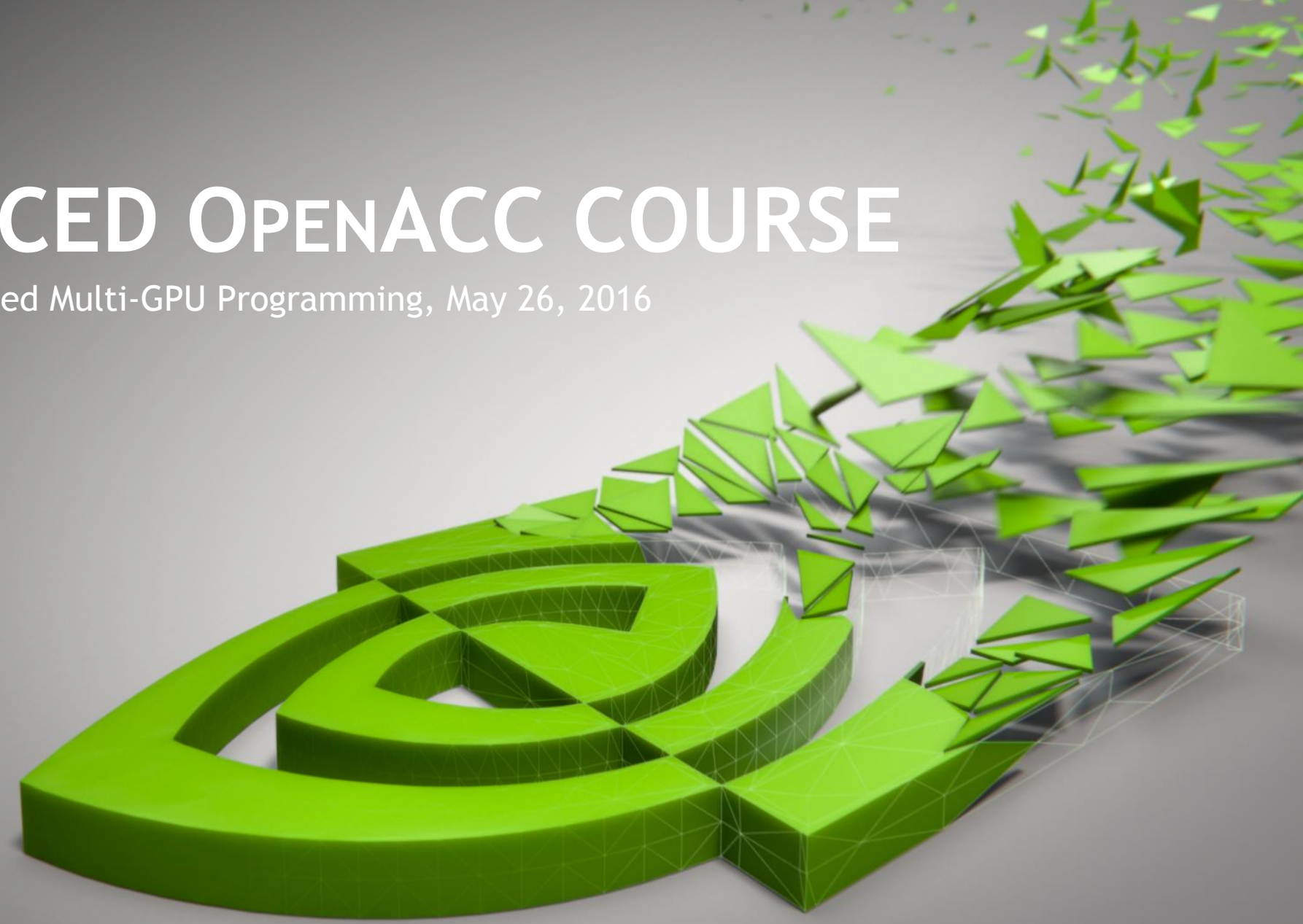
ADVANCED OPENACC COURSE

Lecture 2: Advanced Multi-GPU Programming, May 26, 2016



Course Objective:

Enable *you* to scale *your* applications on multiple GPUs and optimize with profiler tools

Course Syllabus

May 19: Advanced Profiling of OpenACC Code

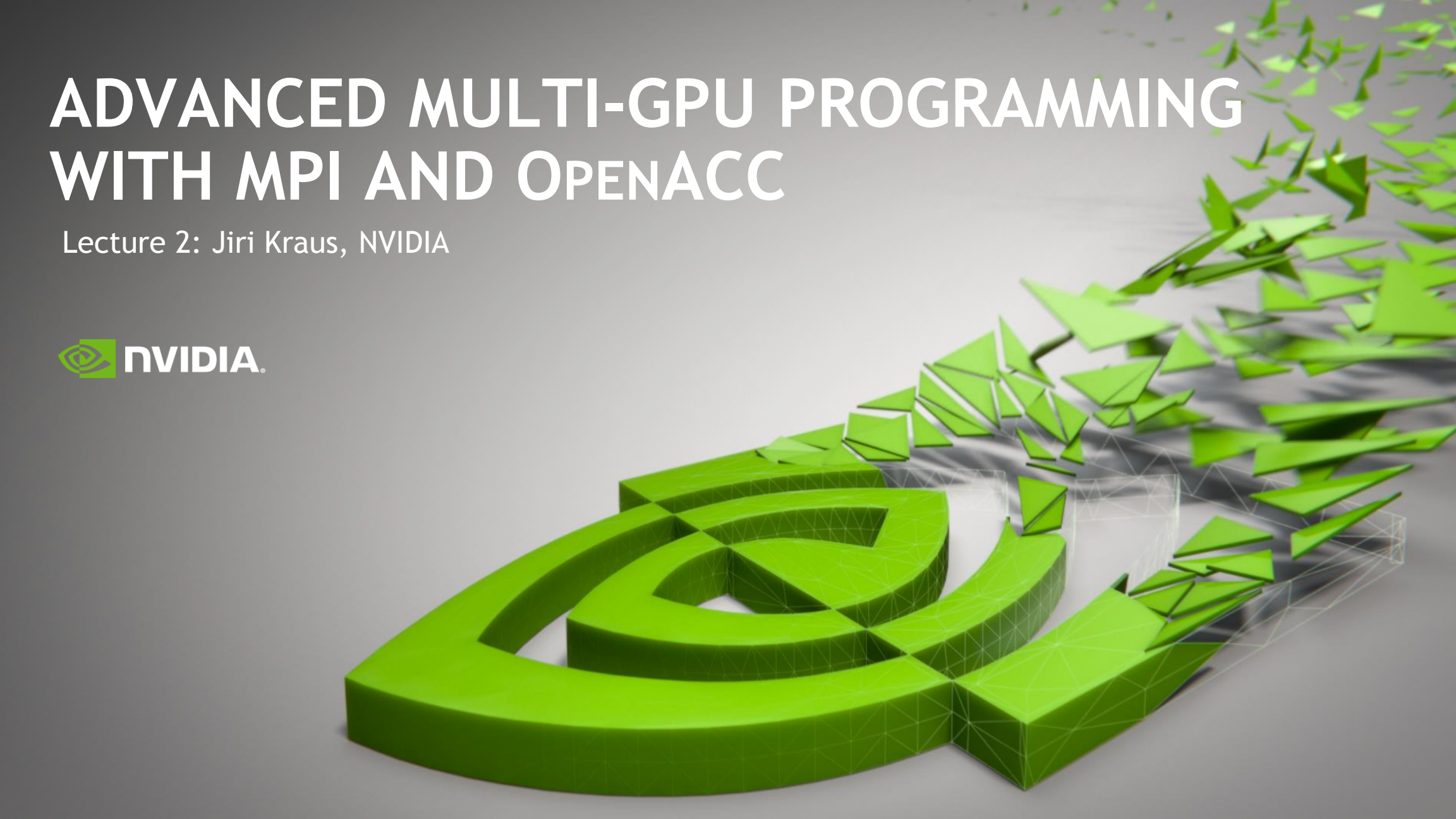
May 26: Office Hours

June 2: Advanced multi-GPU Programming with MPI and OpenACC

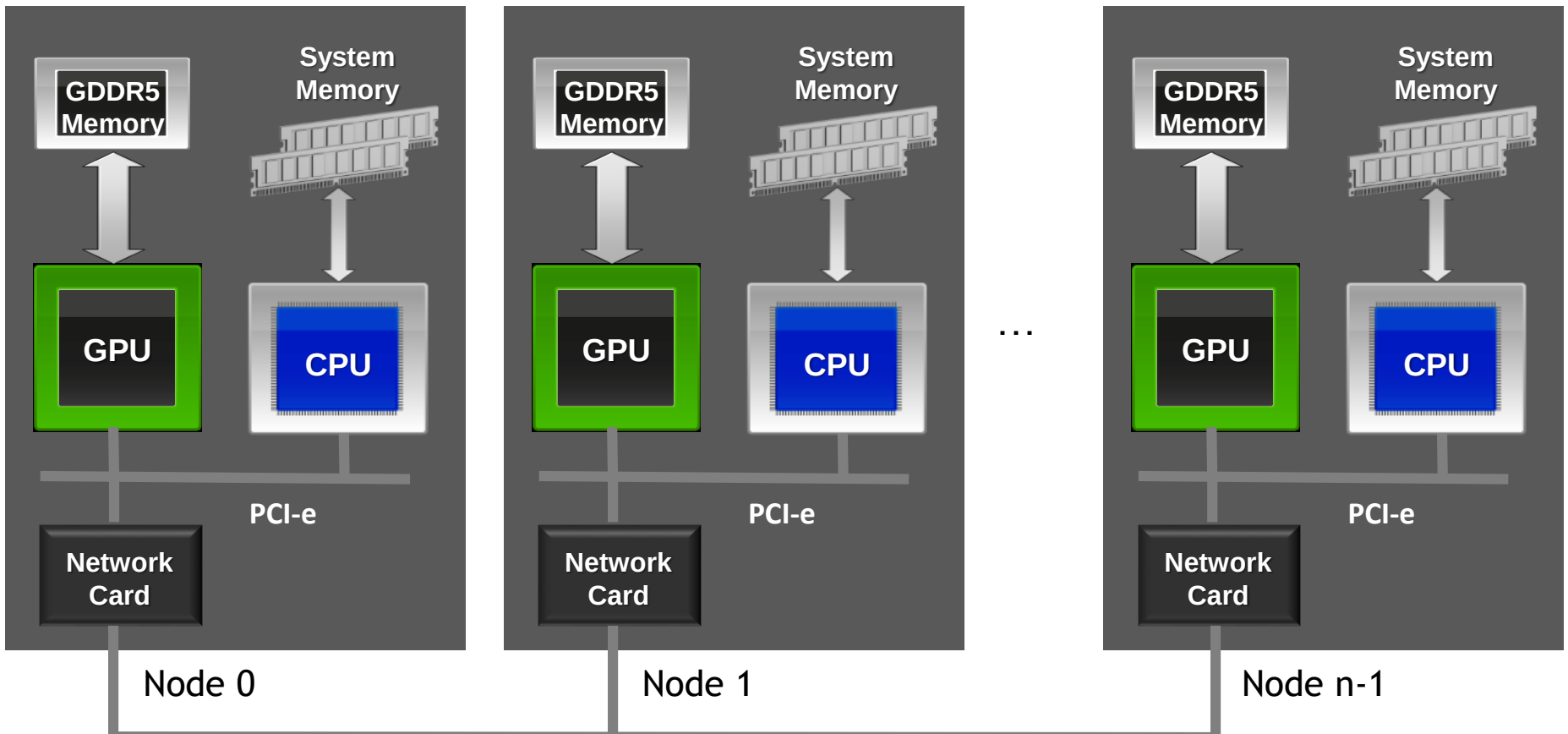
Recordings:
https://developer.nvidia.com/openacc-advanced-course

ADVANCED MULTI-GPU PROGRAMMING WITH MPI AND OPENACC

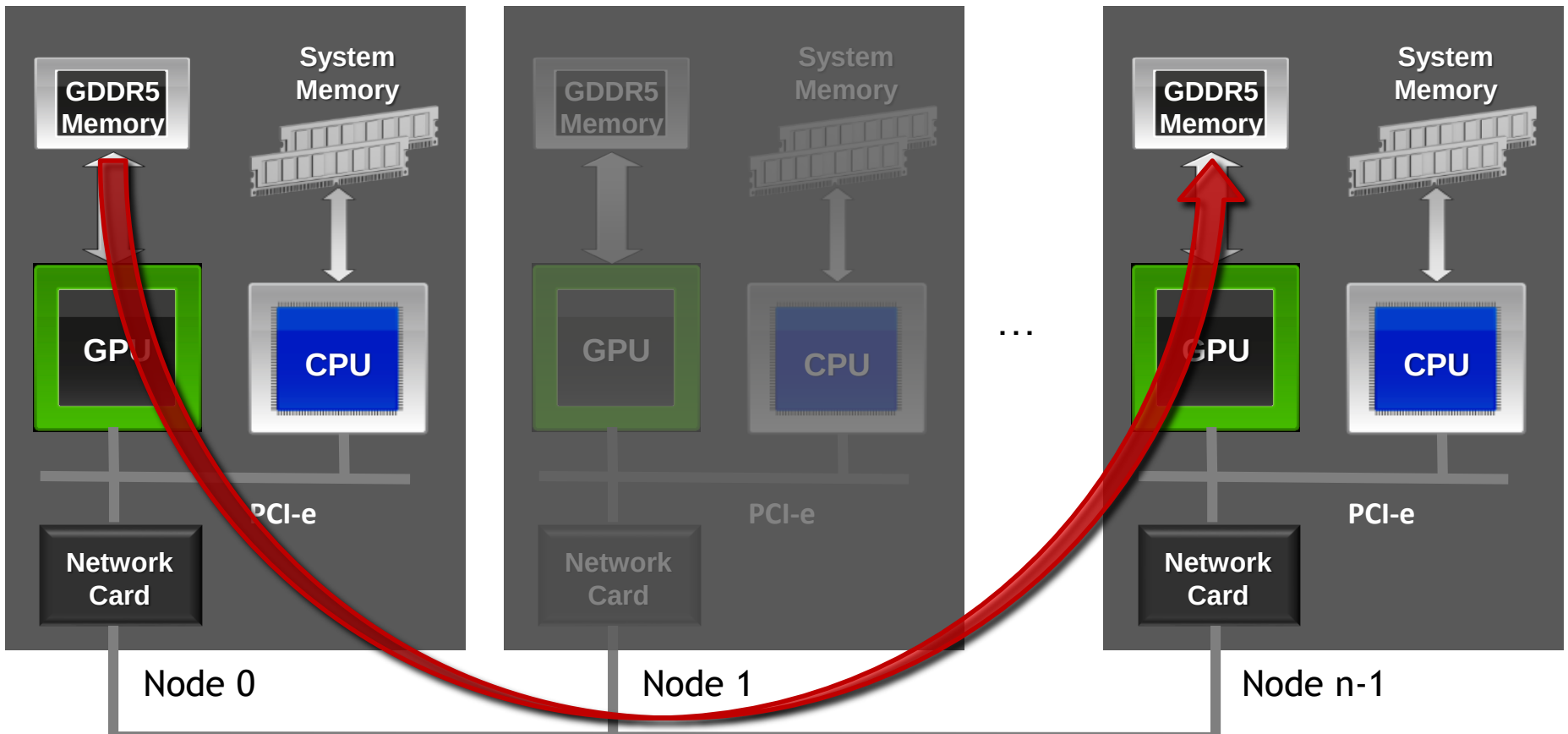
Lecture 2: Jiri Kraus, NVIDIA



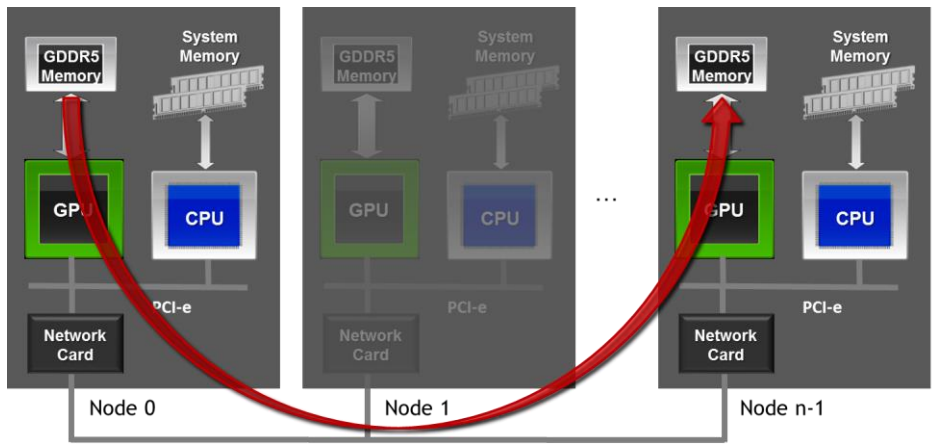
MPI+OPENACC


GDDR5 Memory
System Memory
GPU
CPU
PCI-e
Network Card
Node 0
GDDR5 Memory
System Memory
GPU
CPU
PCI-e
Network Card
Node 1
...
GDDR5 Memory
System Memory
GPU
CPU
PCI-e
Network Card
Node n-1

MPI+OPENACC


GDDR5 Memory
System Memory
GPU
CPU
PCI-e
Network Card
Node 0
GDDR5 Memory
System Memory
GPU
CPU
PCI-e
Network Card
Node 1
...
GDDR5 Memory
System Memory
GPU
CPU
PCI-e
Network Card
Node n-1

MPI+OPENACC




```
//MPI rank 0
#pragma acc host_data use_device( sbuf )
MPI_Send(sbuf, size, MPI_DOUBLE, n-1, tag, MPI_COMM_WORLD);

//MPI rank n-1
#pragma acc host_data use_device( rbuf )
MPI_Recv(rbuf, size, MPI_DOUBLE, 0, tag, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

Agenda

Using MPI for inter GPU communication

Debugging and Profiling of MPI+OpenACC apps

Multi Process Service (MPS)

Decreasing parallel overhead

Using MPI for inter GPU communication

MESSAGE PASSING INTERFACE - MPI

Standard to exchange data between processes via messages

- Defines API to exchanges messages
  - Point to Point: e.g. `MPI_Send`, `MPI_Recv`
  - Collectives: e.g. `MPI_Reduce`

Multiple implementations (open source and commercial)

- Bindings for C/C++, Fortran, Python, …
- E.g. MPICH, OpenMPI, MVAPICH, IBM Platform MPI, Cray MPT, …

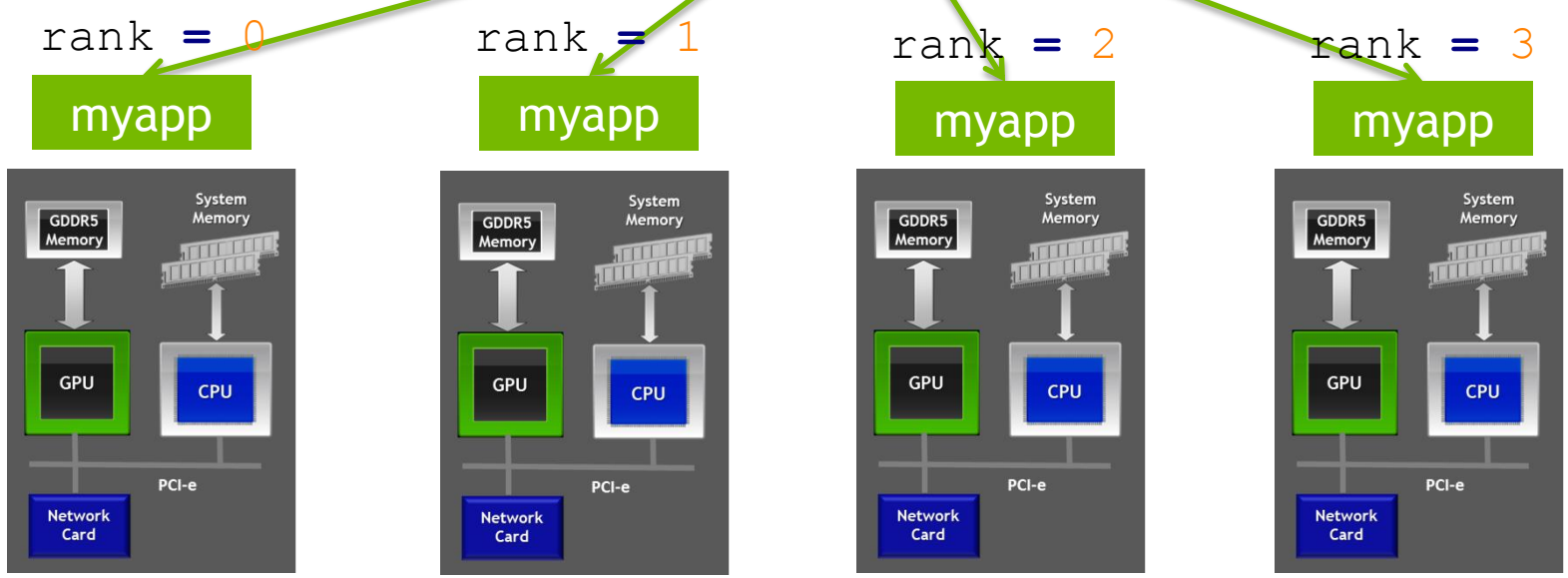
MPI - SKELETON

```c
#include <mpi.h>
int main(int argc, char *argv[]) {
    int rank,size;
    /* Initialize the MPI library */
    MPI_Init(&argc,&argv);
    /* Determine the calling process rank and total number of ranks */
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    MPI_Comm_size(MPI_COMM_WORLD,&size);
    /* Call MPI routines like MPI_Send, MPI_Recv, ... */
    ...
    /* Shutdown MPI library */
    MPI_Finalize();
    return 0;
}
```

MPI

Compiling and Launching

```
$ mpicc -o myapp myapp.c
$ mpirun -np 4 ./myapp <args>
```



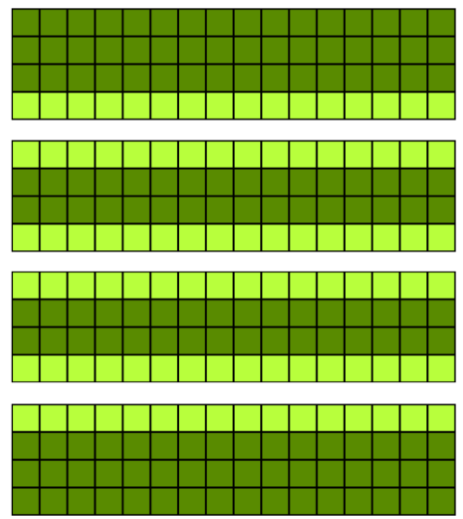
EXAMPLE: JACOBI SOLVER

Solves the 2D-Poission Equation on a rectangle

$$\Delta \boldsymbol{u}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{e}^{-\mathbf{10}*(\boldsymbol{x}^2+\boldsymbol{y}^2)} \; \forall \; (\boldsymbol{x}, \boldsymbol{y}) \in \Omega \backslash \boldsymbol{\delta}\Omega$$

Periodic boundary conditions

Domain decomposition with stripes



Horizontal Stripes

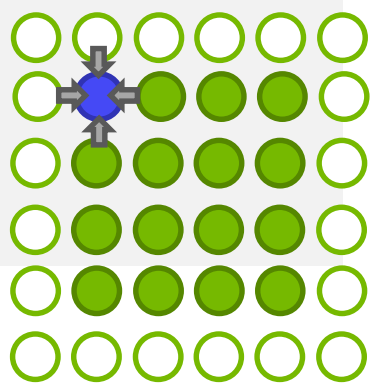
EXAMPLE: JACOBI SOLVER

Single GPU

While not converged

Do Jacobi step:

```
for (int iy = 1; iy < NY-1; ++iy)

for (int ix = 1; ix < NX-1; ++ix)

    Anew[iy][ix] = - 0.25f*(rhs[iy][ix] - ( A[iy][ix-1] + A[iy][ix+1]

                                          + A[iy-1][ix] + A[iy+1][ix]) );
```



Copy `Anew` to `A`

Apply periodic boundary conditions

Next iteration

EXAMPLE: JACOBI SOLVER

Multi GPU

While not converged

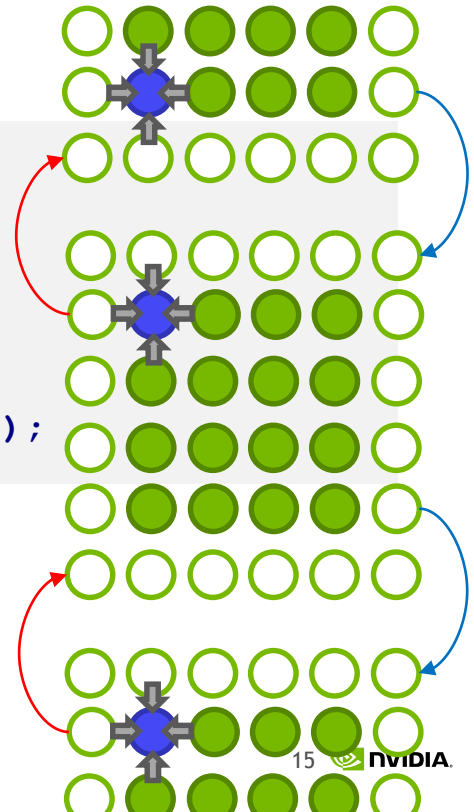
Do Jacobi step:

```
for (int iy = iy_start; iy < iy_end; ++iy)

for (int ix = 1; ix < NX-1; ++ix)

    Anew[iy][ix] = - 0.25f*(rhs[iy][ix] - ( A[iy][ix-1] + A[iy][ix+1]

                                          + A[iy-1][ix] + A[iy+1][ix]) );
```

Copy `Anew` to `A`

Apply periodic boundary conditions and exchange halo with 2 neighbors

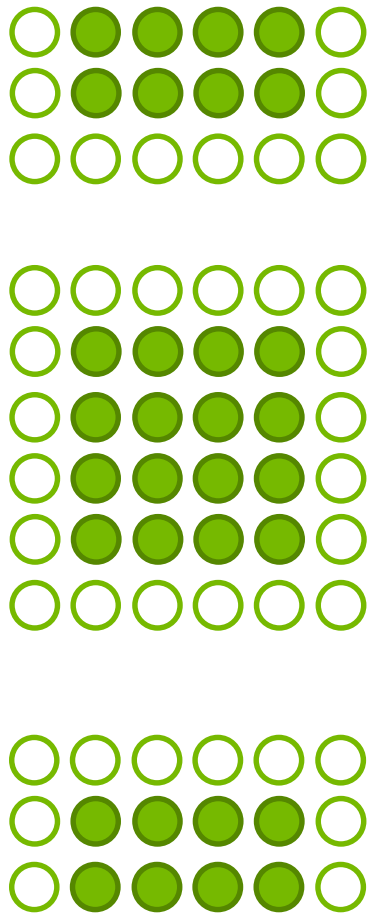
Next iteration



EXAMPLE JACOBI

Top/Bottom Halo

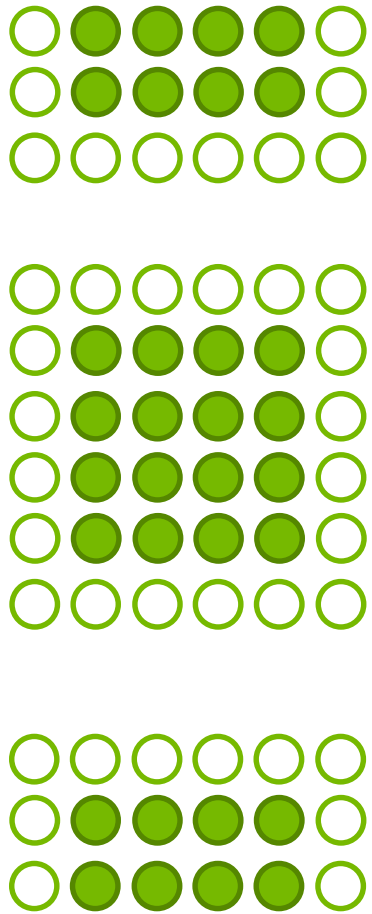
```
#pragma acc host_data use_device ( A )
{




}
```



EXAMPLE JACOBI

Top/Bottom Halo

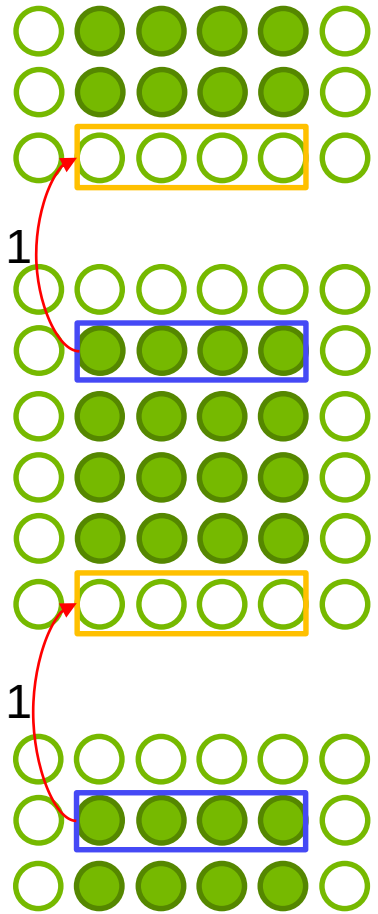
```
#pragma acc host_data use_device ( A )
{

MPI_Sendrecv(&A[iy_start][1], NX-2, MPI_DOUBLE, top, 0,
             &A[iy_end][1], NX-2, MPI_DOUBLE, bottom, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);









}
```



EXAMPLE JACOBI

Top/Bottom Halo

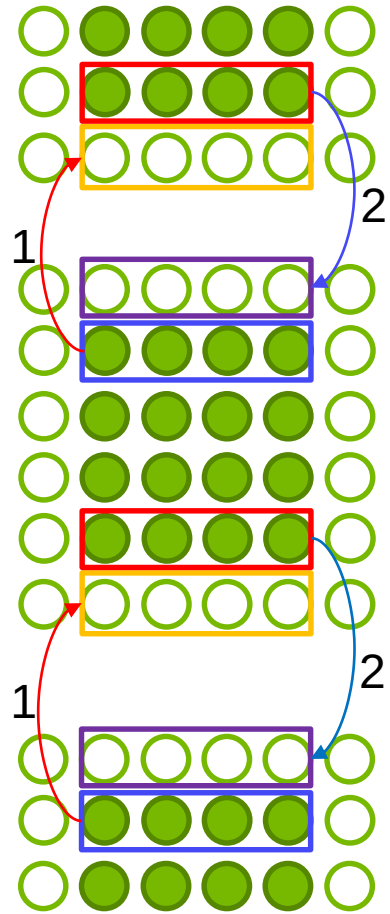
```
#pragma acc host_data use_device ( A )
{

MPI_Sendrecv(&A[iy_start][1], NX-2, MPI_DOUBLE, top, 0,
             &A[iy_end][1], NX-2, MPI_DOUBLE, bottom, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);









}
```



EXAMPLE JACOBI

Top/Bottom Halo

```
#pragma acc host_data use_device ( A )
{

MPI_Sendrecv(&A[iy_start][1], NX-2, MPI_DOUBLE, top, 0,
             &A[iy_end][1], NX-2, MPI_DOUBLE, bottom, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);




MPI_Sendrecv(&A[(iy_end-1)][1], NX-2, MPI_DOUBLE, bottom, 1,
             &A[(iy_start-1)][1], NX-2, MPI_DOUBLE, top, 1,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);


}
```



HANDLING MULTI GPU NODES

GPU-affinity

```
#if _OPENACC

    acc_device_t device_type = acc_get_device_type();

    if ( acc_device_nvidia == device_type ) {

        int ngpus=acc_get_num_devices(acc_device_nvidia);

        int devicenum=rank%ngpus;

        acc_set_device_num(devicenum,acc_device_nvidia);

    }

    acc_init(device_type);

#endif /*_OPENACC*/
```

Alternative (OpenMPI):

```
int devicenum = atoi(getenv("OMPI_COMM_WORLD_LOCAL_RANK"));
```

Alternative (MVAPICH2):

```
int devicenum = atoi(getenv("MV2_COMM_WORLD_LOCAL_RANK"));
```

Debugging and Profiling of MPI+OpenACC apps

TOOLS FOR MPI+OPENACC APPLICATIONS

Memory checking: `cuda-memcheck`

Debugging: `cuda-gdb`

Profiling: `nvprof` and the NVIDIA Visual Profiler (`nvvp`)

MEMORY CHECKING WITH CUDA-MEMCHECK

`cuda-memcheck` is a tool similar to Valgrind's memcheck

Can be used in a MPI environment

```
mpiexec -np 2 cuda-memcheck ./myapp <args>
```

Problem: Output of different processes is interleaved

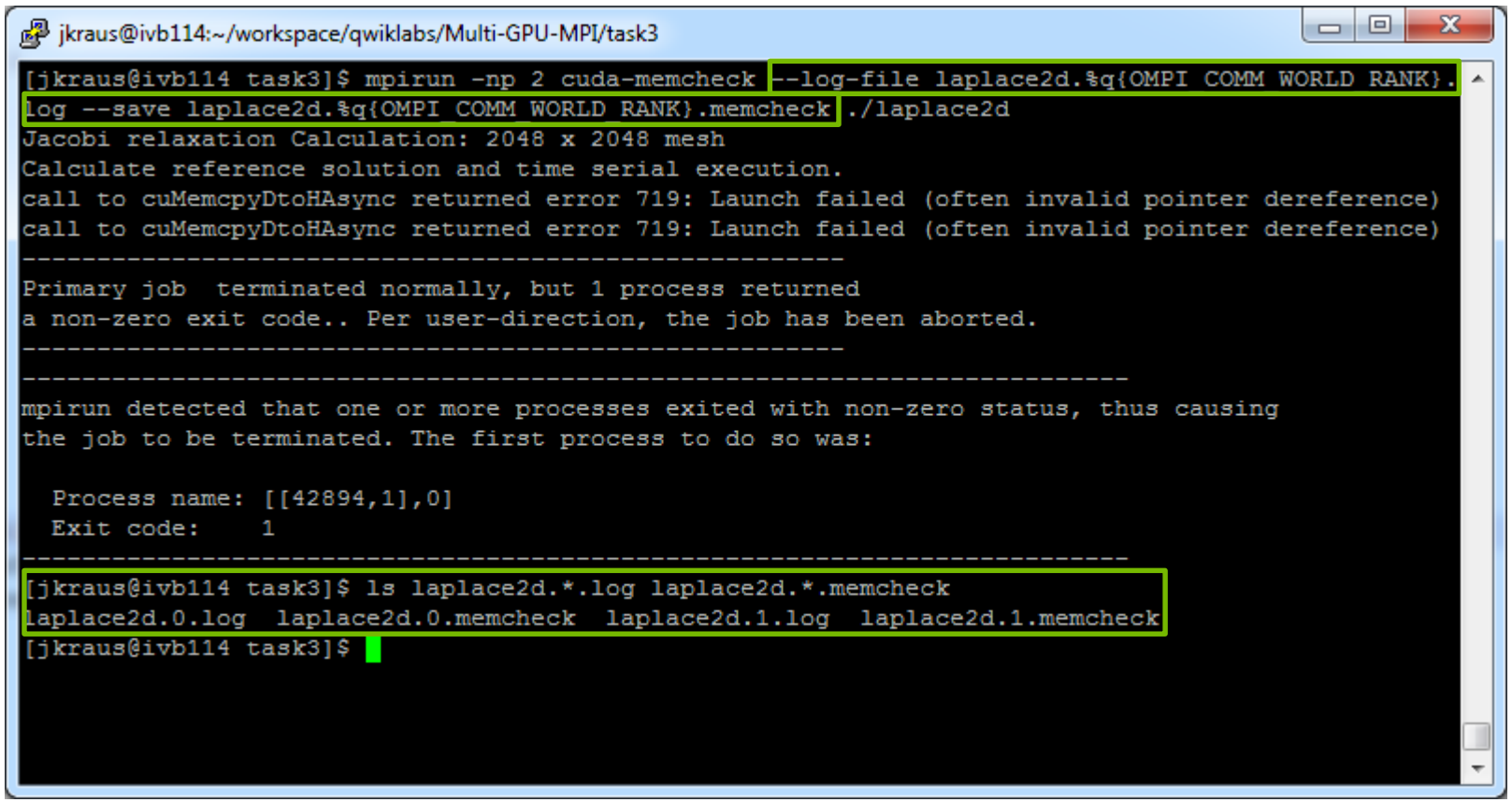
Solution: Use save or log-file command line options

OpenMPI: `OMPI_COMM_WORLD_RANK`

MVAPICH2: `MV2_COMM_WORLD_RANK`

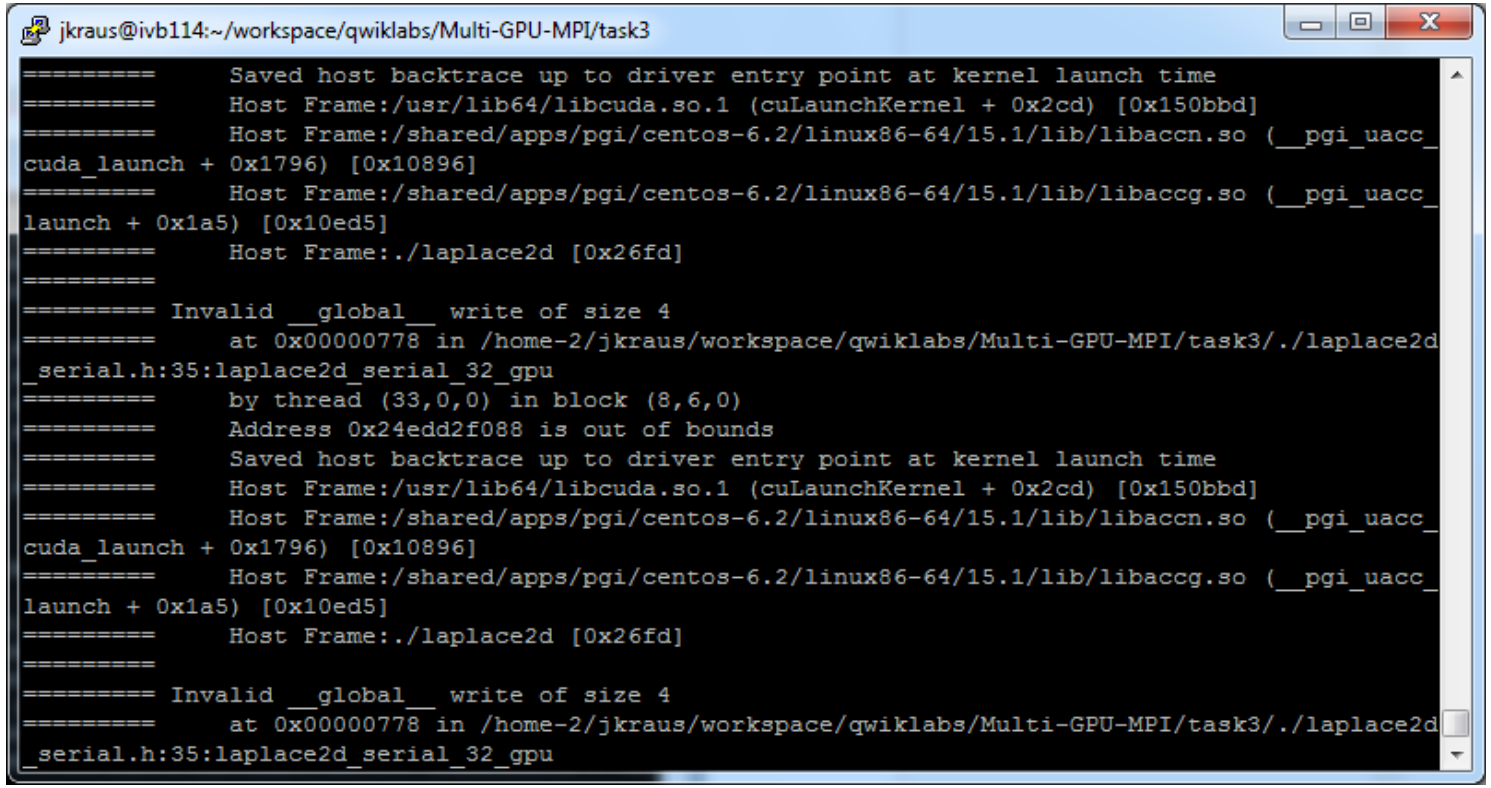
```
mpirun -np 2 cuda-memcheck                                  \
 --log-file name.%q{OMPI_COMM_WORLD_RANK}.log               \
 --save name.%q{OMPI_COMM_WORLD_RANK}.memcheck              \
./myapp <args>
```

MEMORY CHECKING WITH CUDA-MEMCHECK



MEMORY CHECKING WITH CUDA-MEMCHECK

Read Output Files with `cuda-memcheck --read`



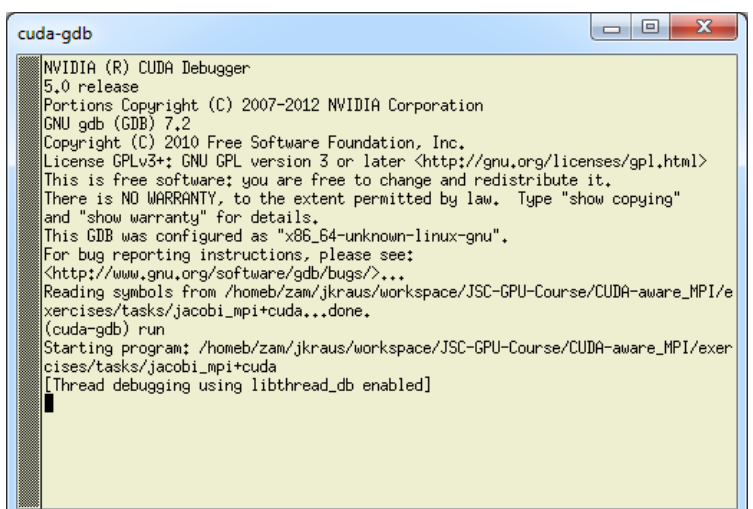
DEBUGGING MPI+OPENACC APPLICATIONS

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+OPENACC APPLICATIONS

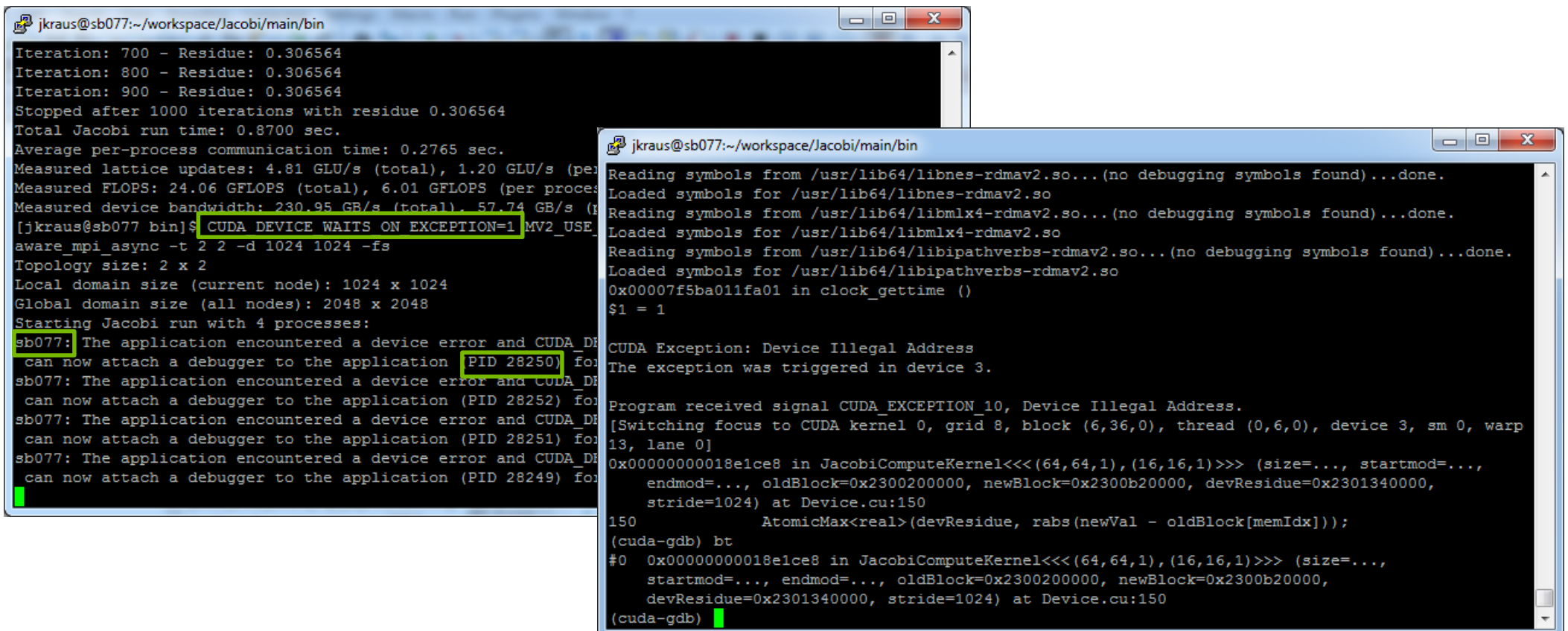
cuda-gdb Attach

```
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+OPENACC APPLICATIONS

CUDA_DEVICE_WAITS_ON_EXCEPTION



DEBUGGING MPI+OPENACC 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, e.g. too many nodes needed to reproduce

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

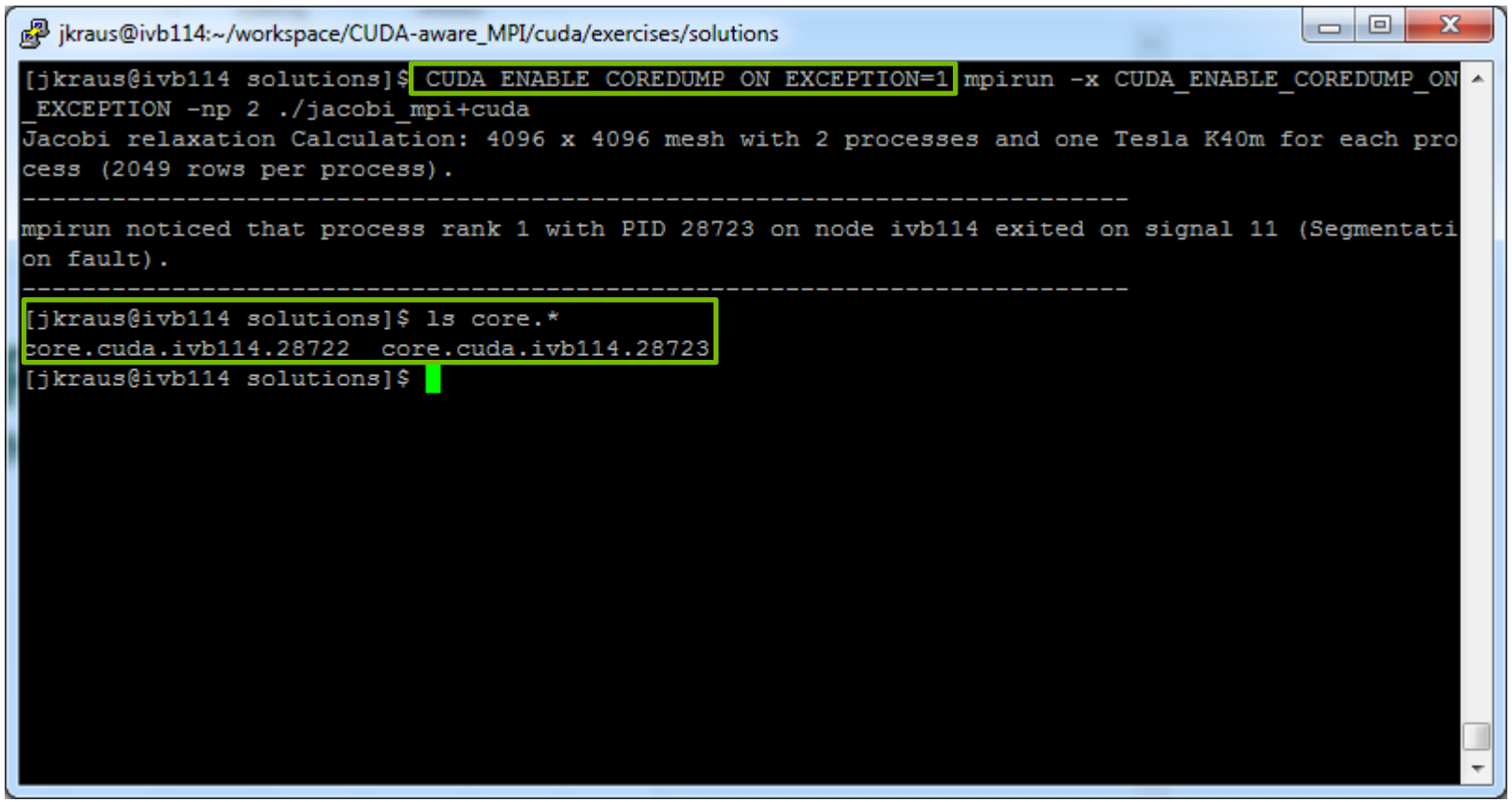
`CUDA_COREDUMP_FILE`: Specify name of core dump file

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

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

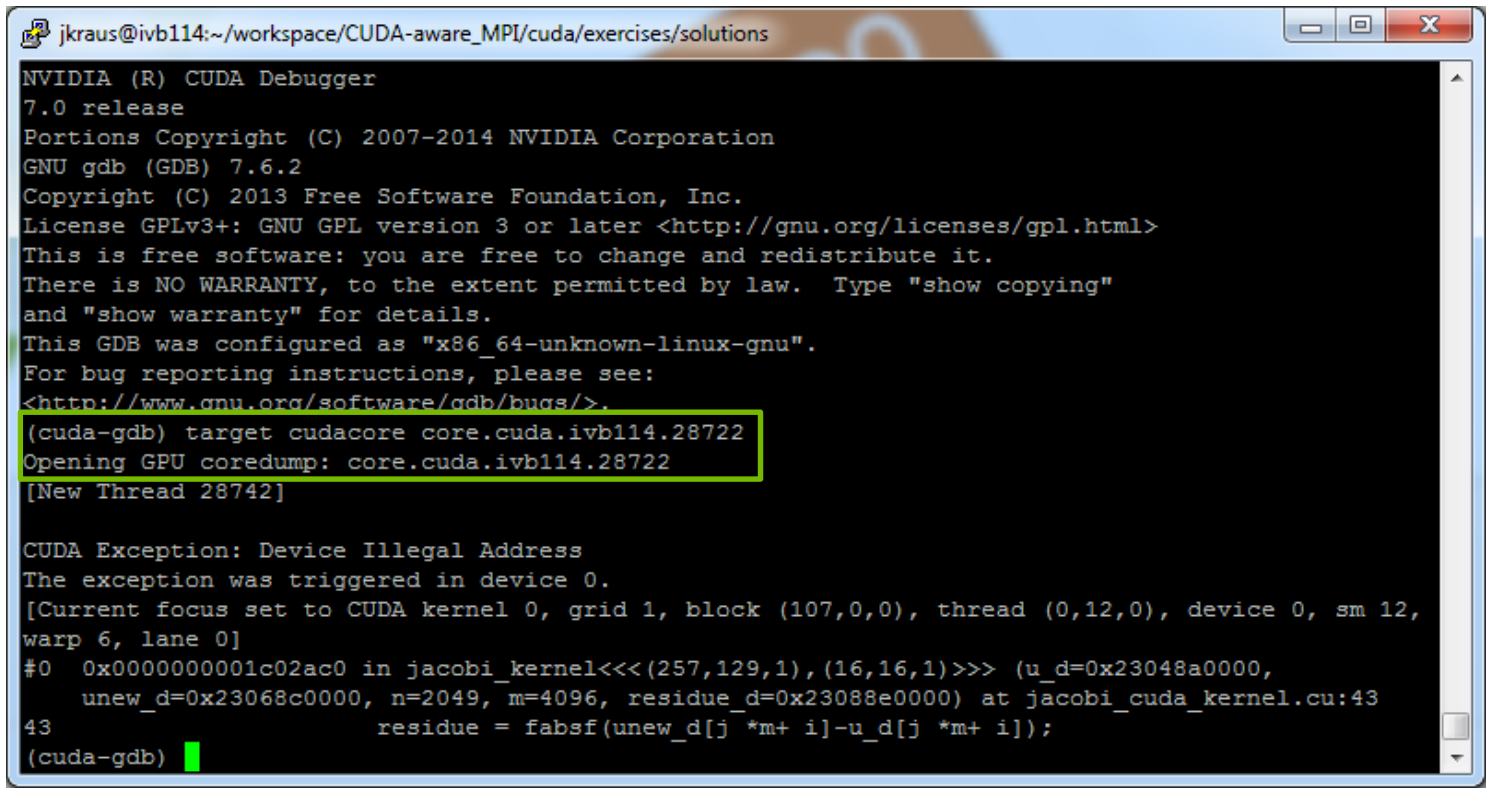
DEBUGGING MPI+OPENACC APPLICATIONS

CUDA_ENABLE_COREDUMP_ON_EXCEPTION



DEBUGGING MPI+OPENACC APPLICATIONS

CUDA_ENABLE_COREDUMP_ON_EXCEPTION

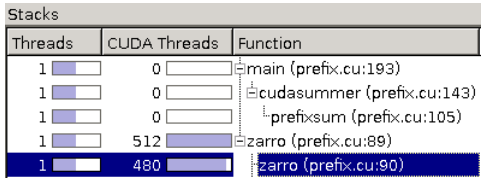


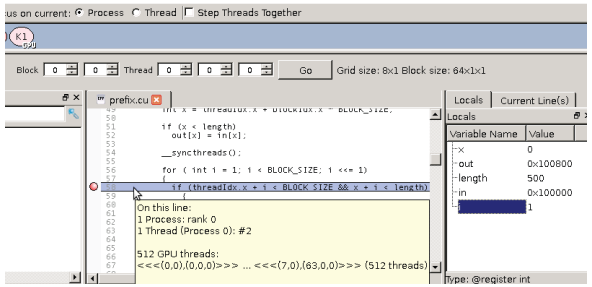
DEBUGGING MPI+OPENACC APPLICATIONS

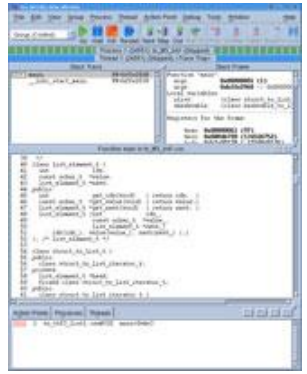
Third Party Tools

Allinea DDT debugger

Rogue Wave TotalView







PROFILING MPI+OPENACC APPLICATIONS

Using nvprof+NVVP

Embed MPI rank in output filename, process name, and context name

```
mpirun -np $np nvprof --output-profile profile.%q{OMPI_COMM_WORLD_RANK}     \
                      --process-name "rank %q{OMPI_COMM_WORLD_RANK}"        \
                      --context-name "rank %q{OMPI_COMM_WORLD_RANK}"
```

New with CUDA 7.5

OpenMPI: `OMPI_COMM_WORLD_RANK`

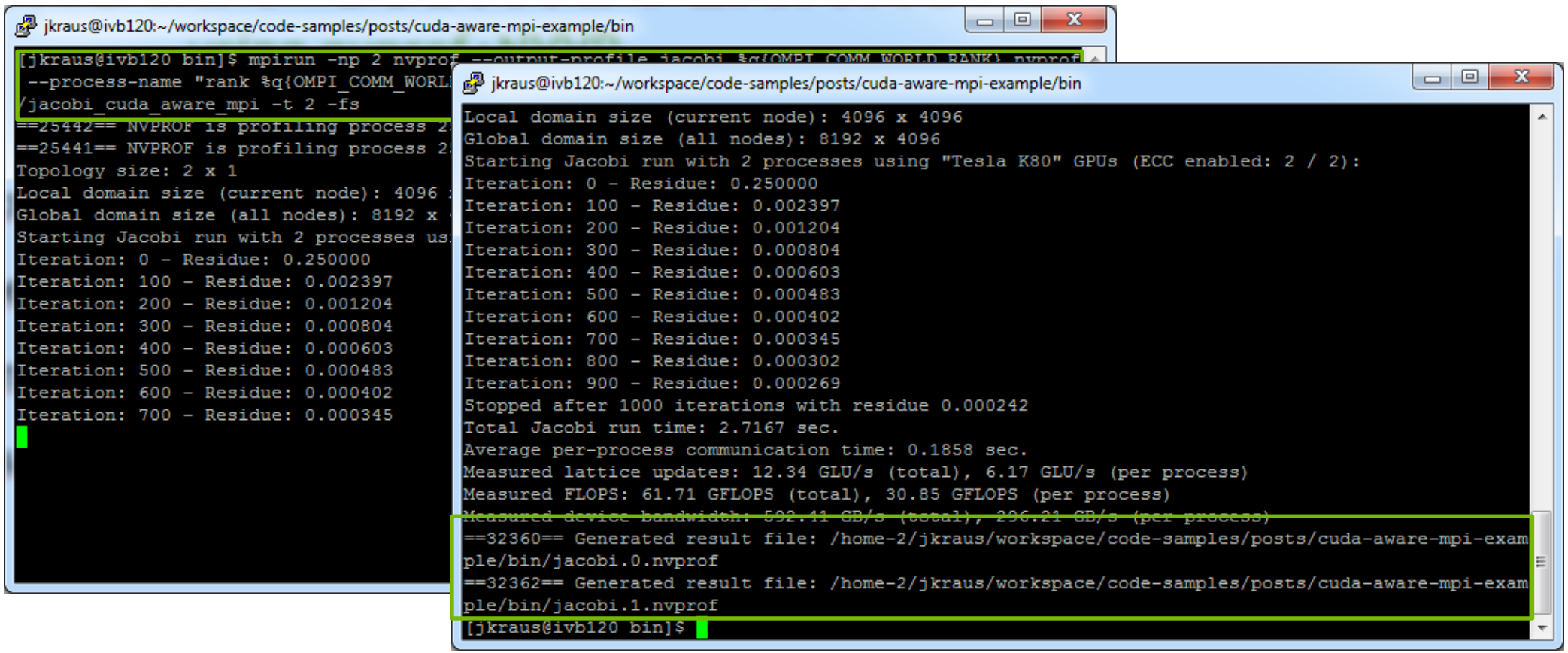
MVAPICH2: `MV2_COMM_WORLD_RANK`

Alternatives:

- Only save the textual output (`--log-file`)
- Collect data from all processes that run on a node (`--profile-all-processes`)

PROFILING MPI+OPENACC APPLICATIONS

Using `nvprof`+NVVP

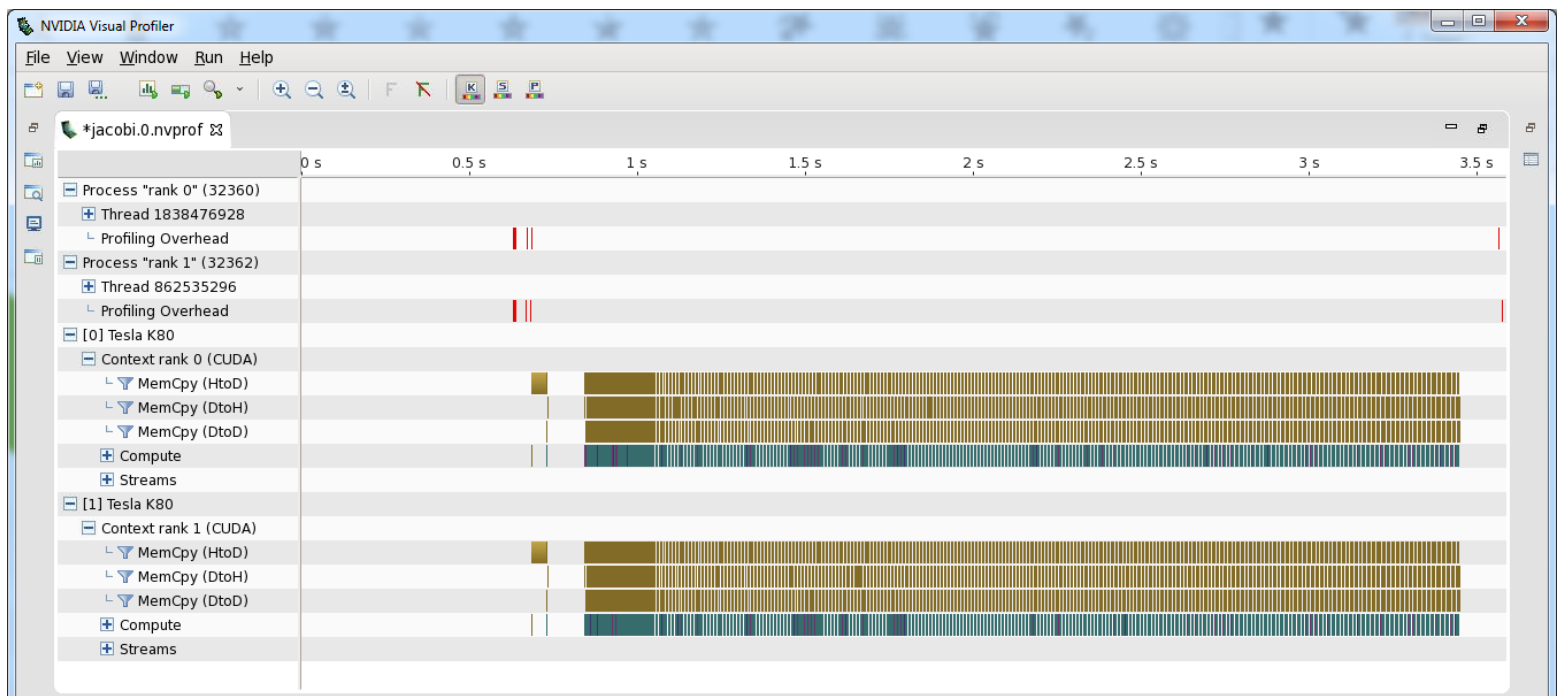


PROFILING MPI+OPENACC APPLICATIONS

Using nvprof+NVVP

```
nvvp jacobi.*.nvprof
```

Or use the import Wizard



PROFILING MPI+OPENACC APPLICATIONS

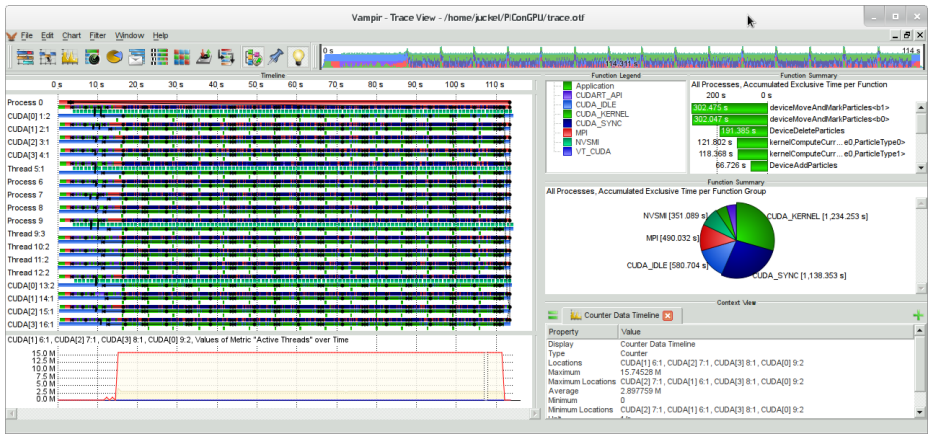
Third Party Tools

Multiple parallel profiling tools are OpenACC-aware

Score-P

Vampir

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



Multi Process Service (MPS)

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

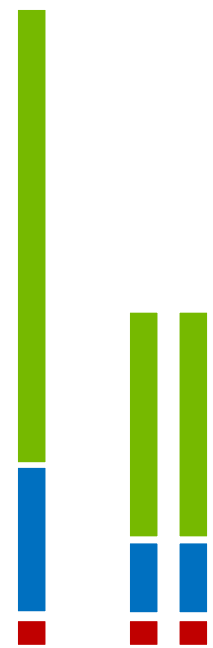
GPU parallelizable part
CPU parallel part
Serial part

N=1

Multicore CPU only

MULTI PROCESS SERVICE (MPS)

For Legacy MPI Applications



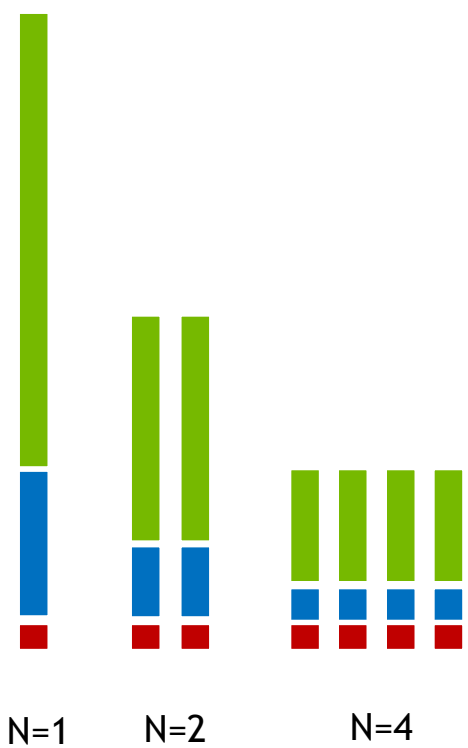
N=1 N=2

GPU parallelizable part
CPU parallel part
Serial part

Multicore CPU only

MULTI PROCESS SERVICE (MPS)

For Legacy MPI Applications

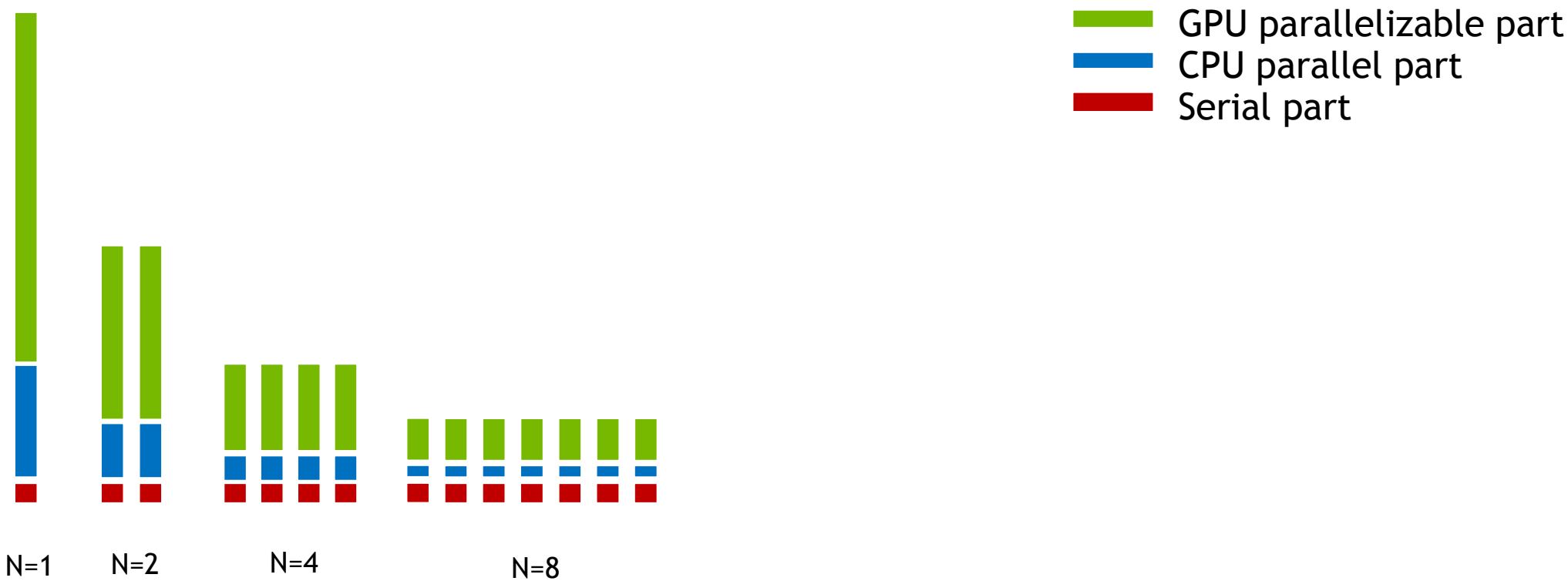


GPU parallelizable part
CPU parallel part
Serial part

Multicore CPU only

MULTI PROCESS SERVICE (MPS)

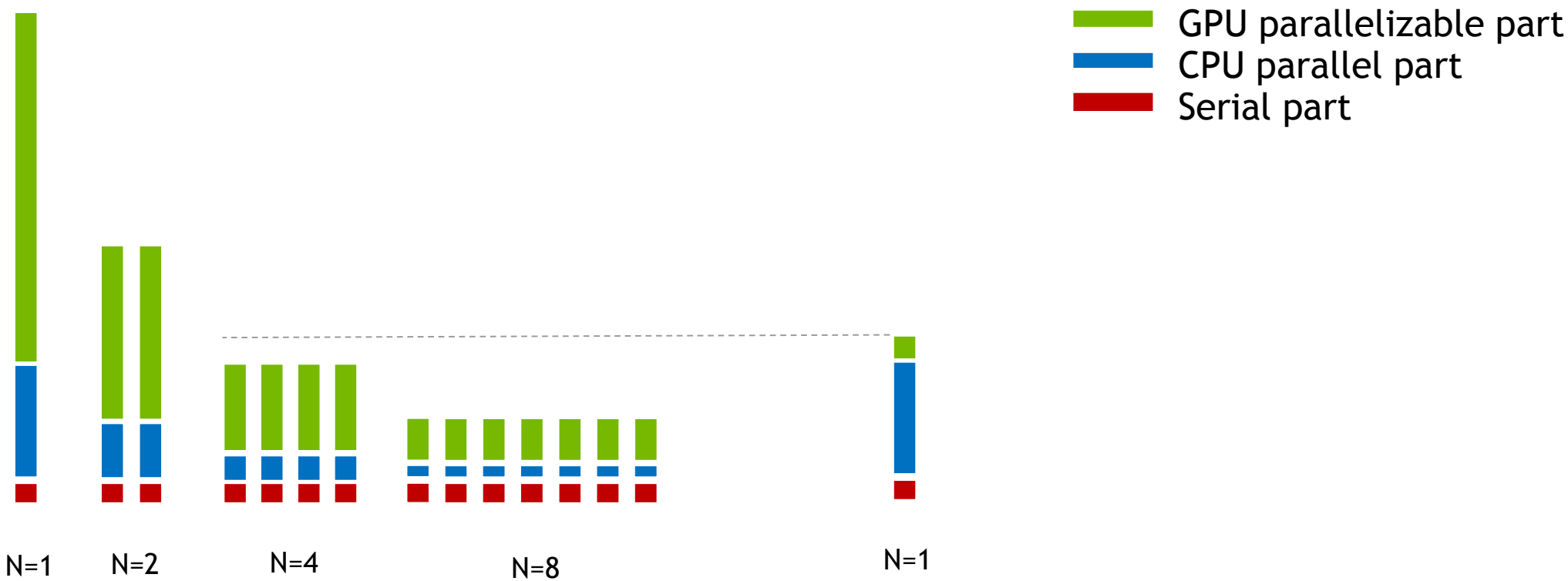
For Legacy MPI Applications



Multicore CPU only

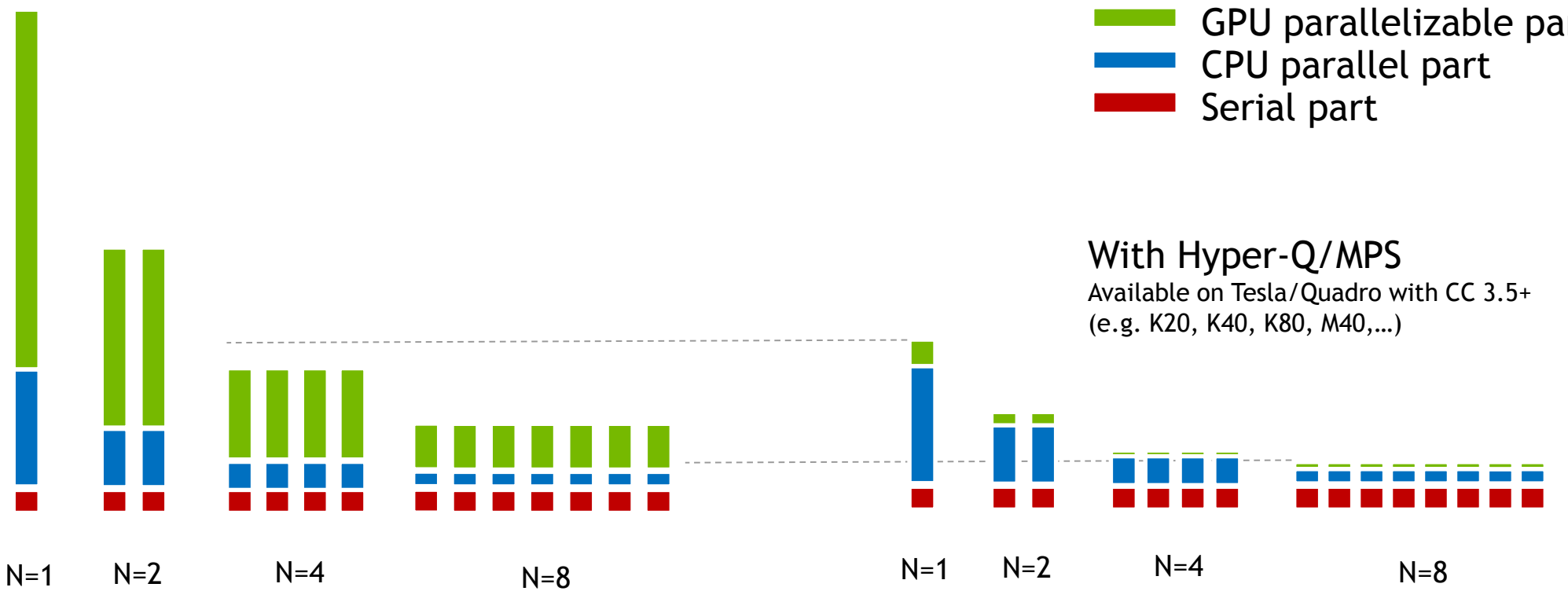
MULTI PROCESS SERVICE (MPS)

For Legacy MPI Applications



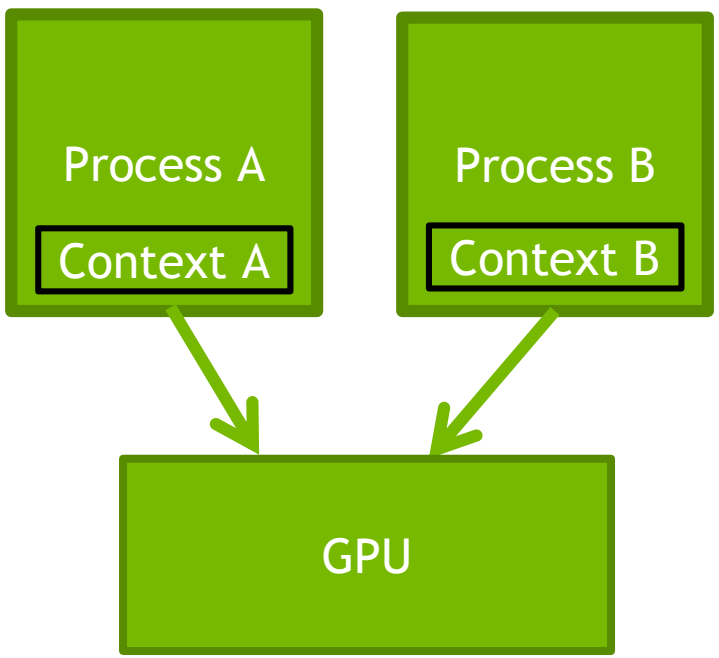
MULTI PROCESS SERVICE (MPS)

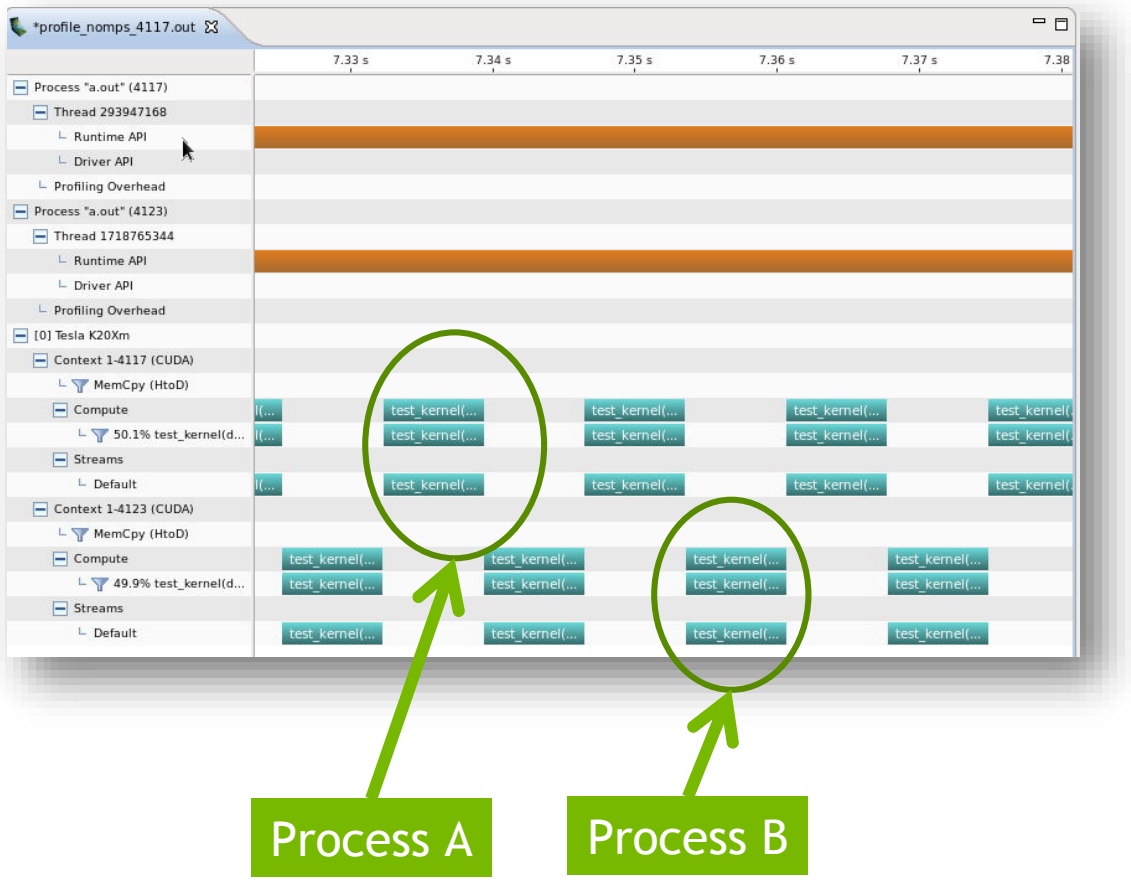
For Legacy MPI Applications



PROCESSES SHARING GPU WITHOUT MPS

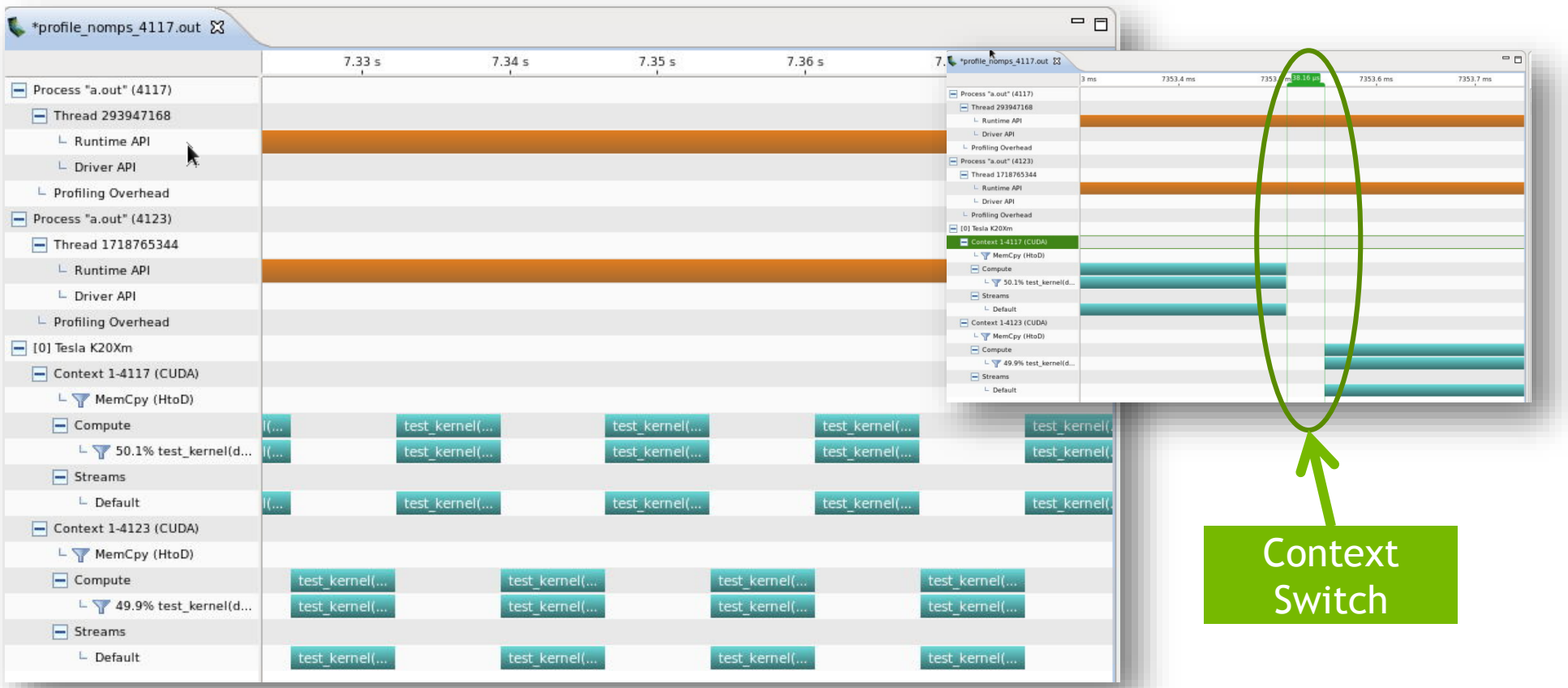
No Overlap





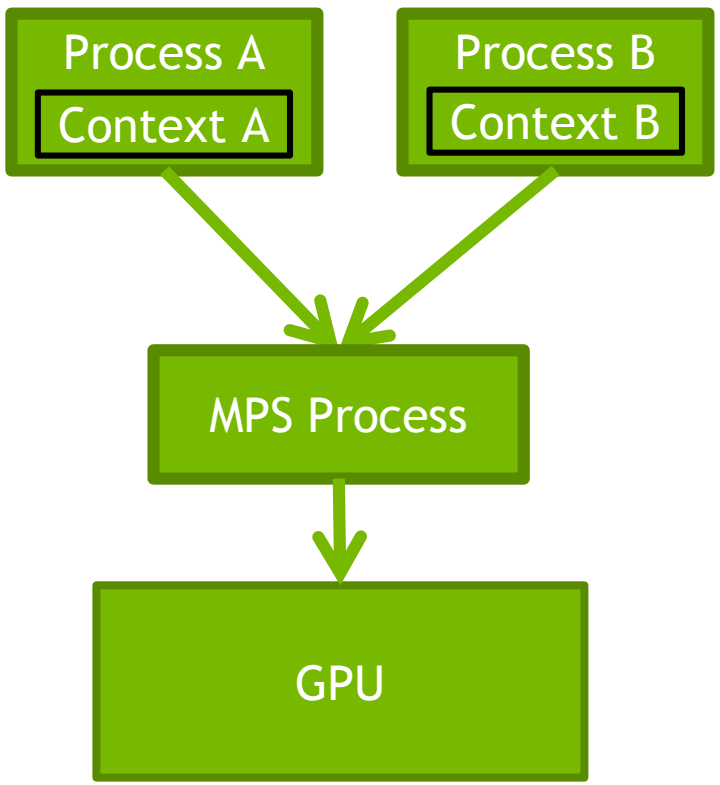
PROCESSES SHARING GPU WITHOUT MPS

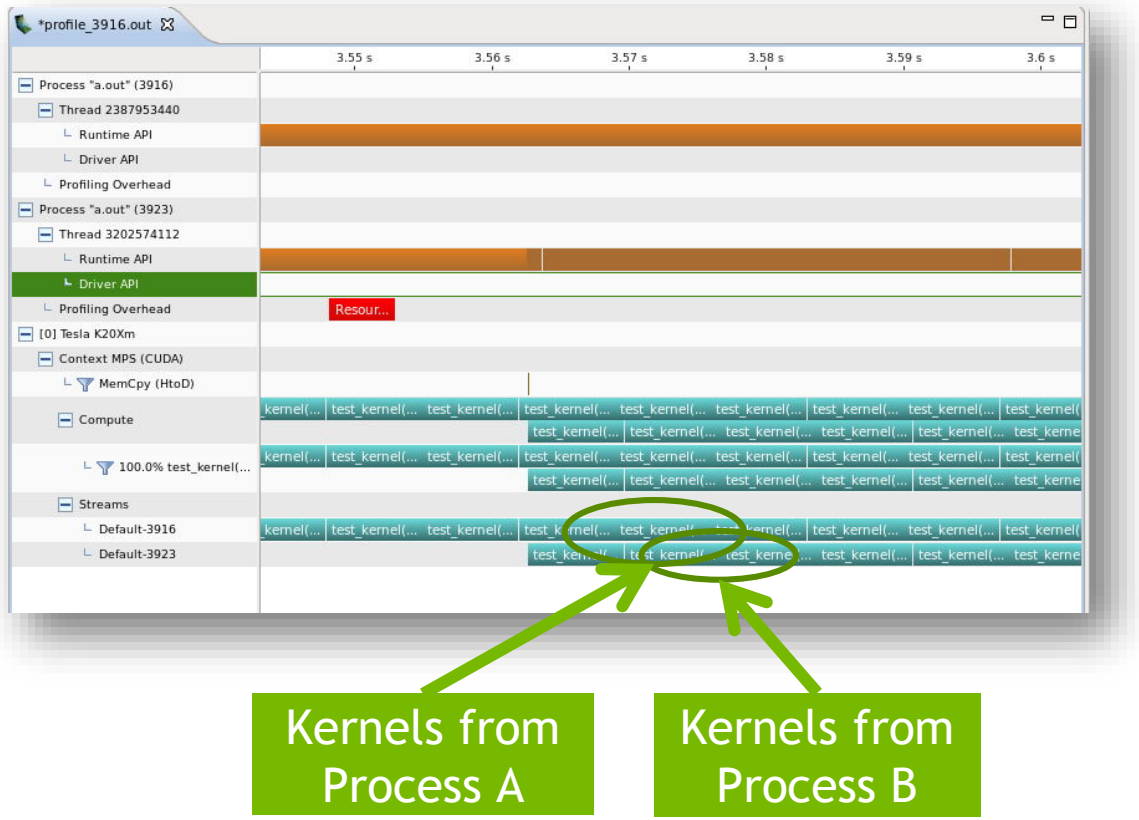
Context Switch Overhead



PROCESSES SHARING GPU WITH MPS

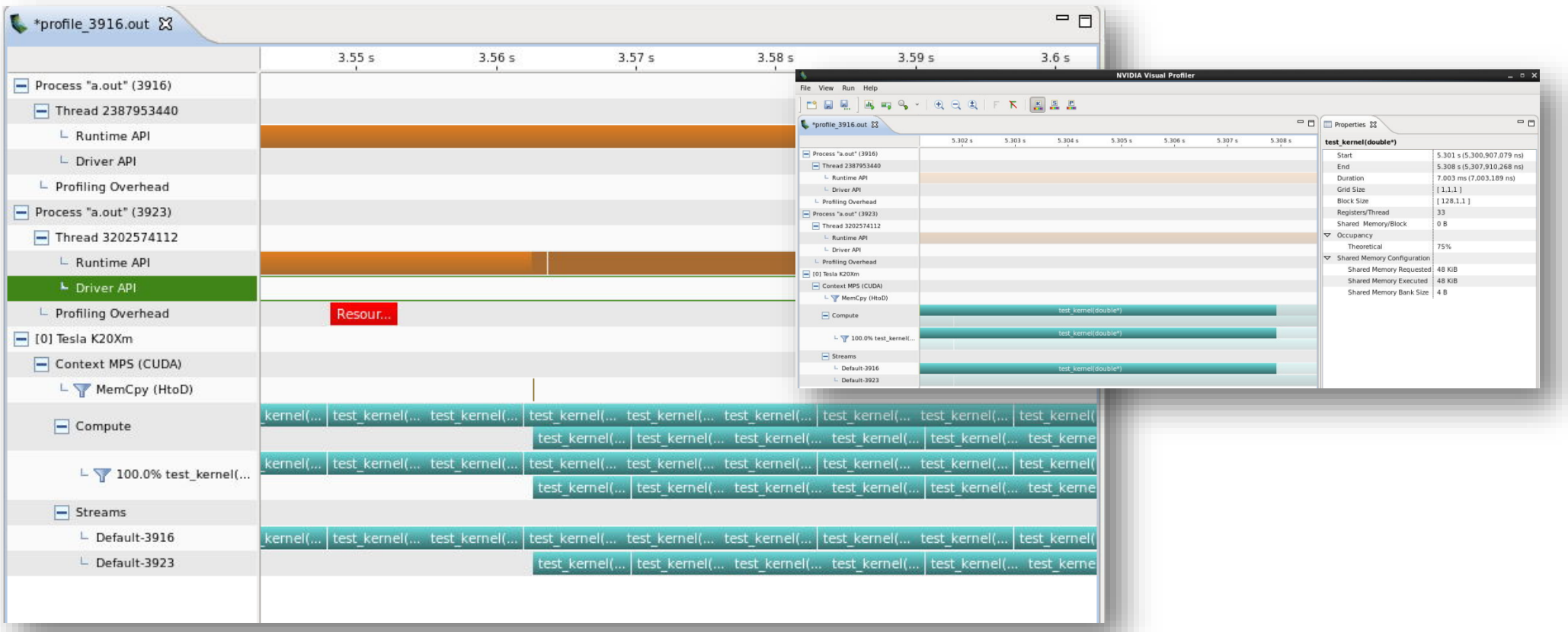
Maximum Overlap



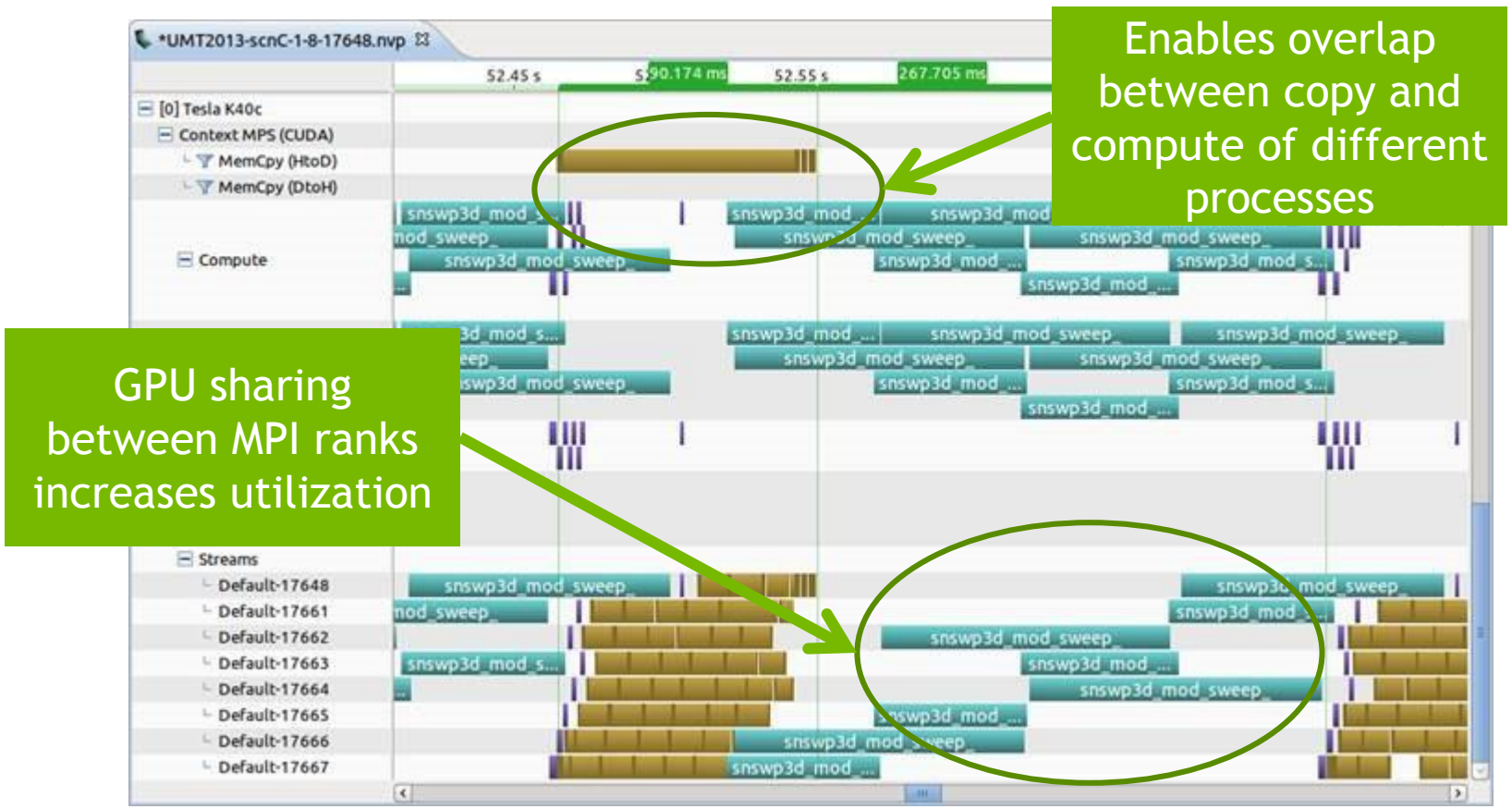


PROCESSES SHARING GPU WITH MPS

No Context Switch Overhead

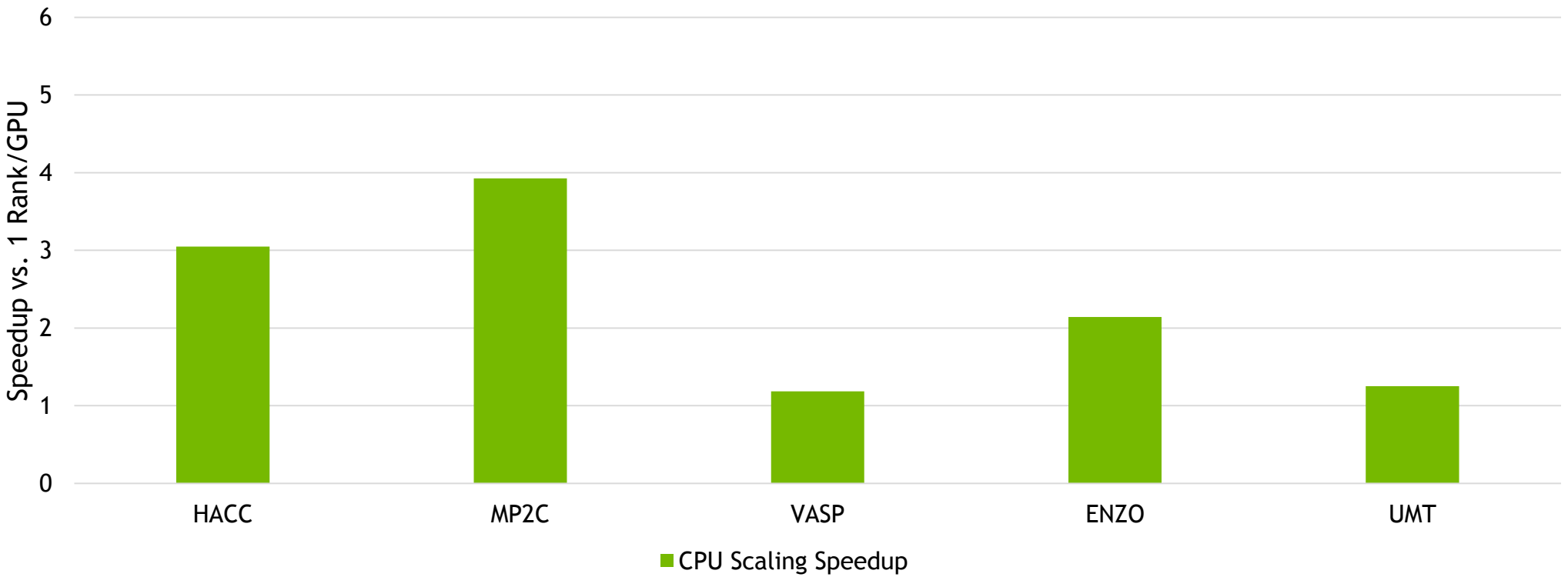


HYPER-Q/MPS CASE STUDY: UMT



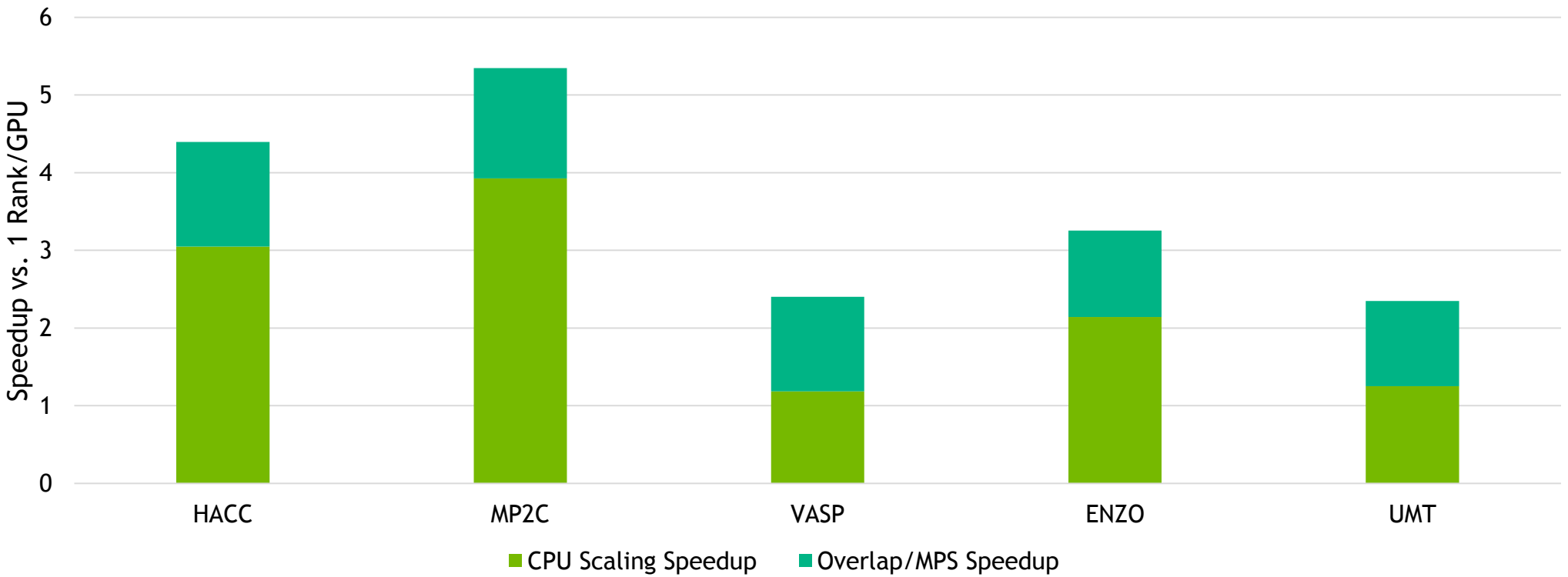
HYPER-Q/MPS CASE STUDIES

CPU Scaling Speedup



HYPER-Q/MPS CASE STUDIES

Additional Speedup with MPS



USING MPS

No application modifications necessary

Not limited to MPI applications

MPS control daemon

Spawn MPS server upon CUDA application startup

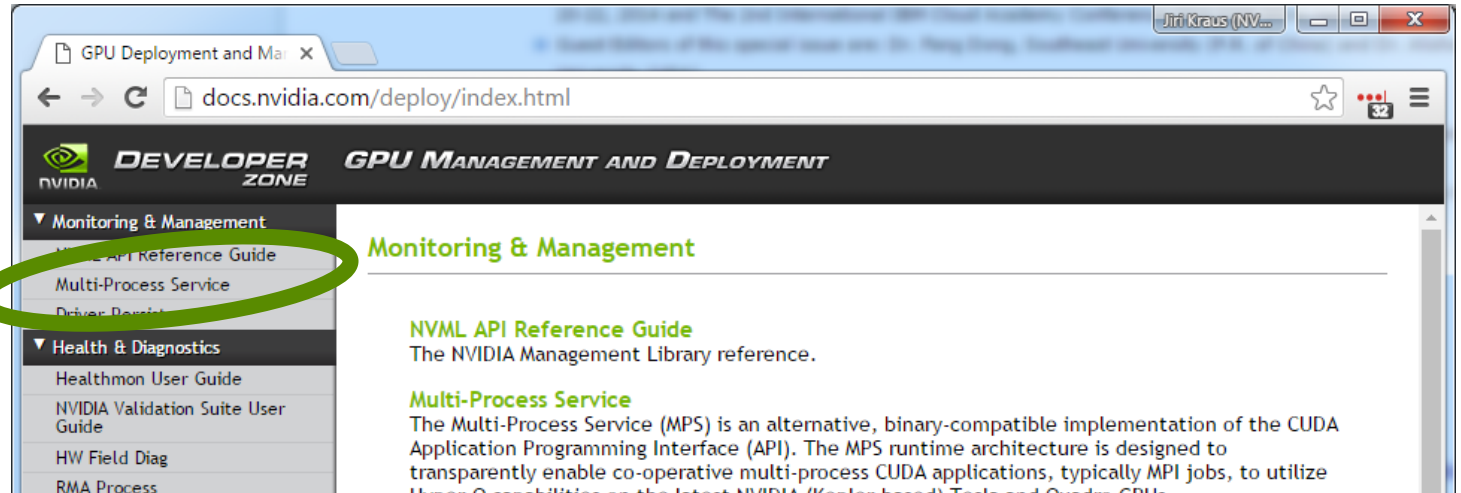
```
#Typical Setup

nvidia-smi -c EXCLUSIVE_PROCESS

nvidia-cuda-mps-control -d


#On Cray XK/XC systems

export CRAY_CUDA_MPS=1
```

MPS SUMMARY

Easy path to get GPU acceleration for legacy applications

Enables overlapping of memory copies and compute between different MPI ranks

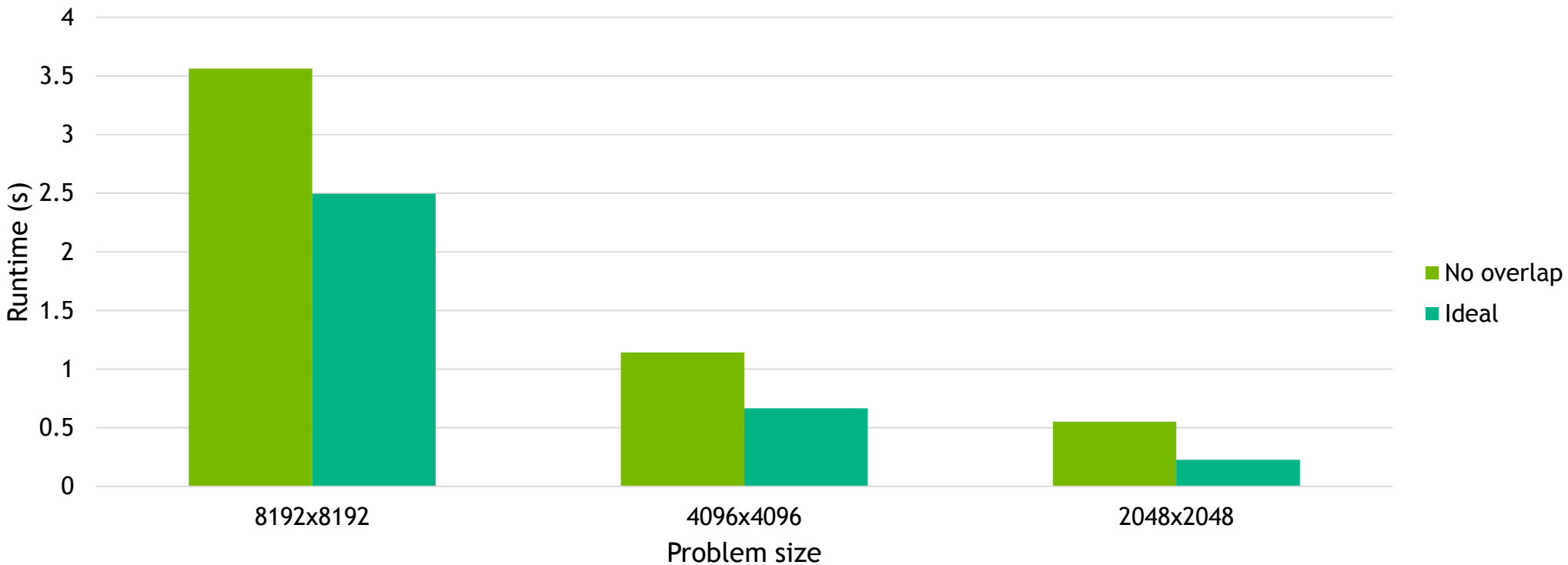
Remark: MPS adds some overhead!



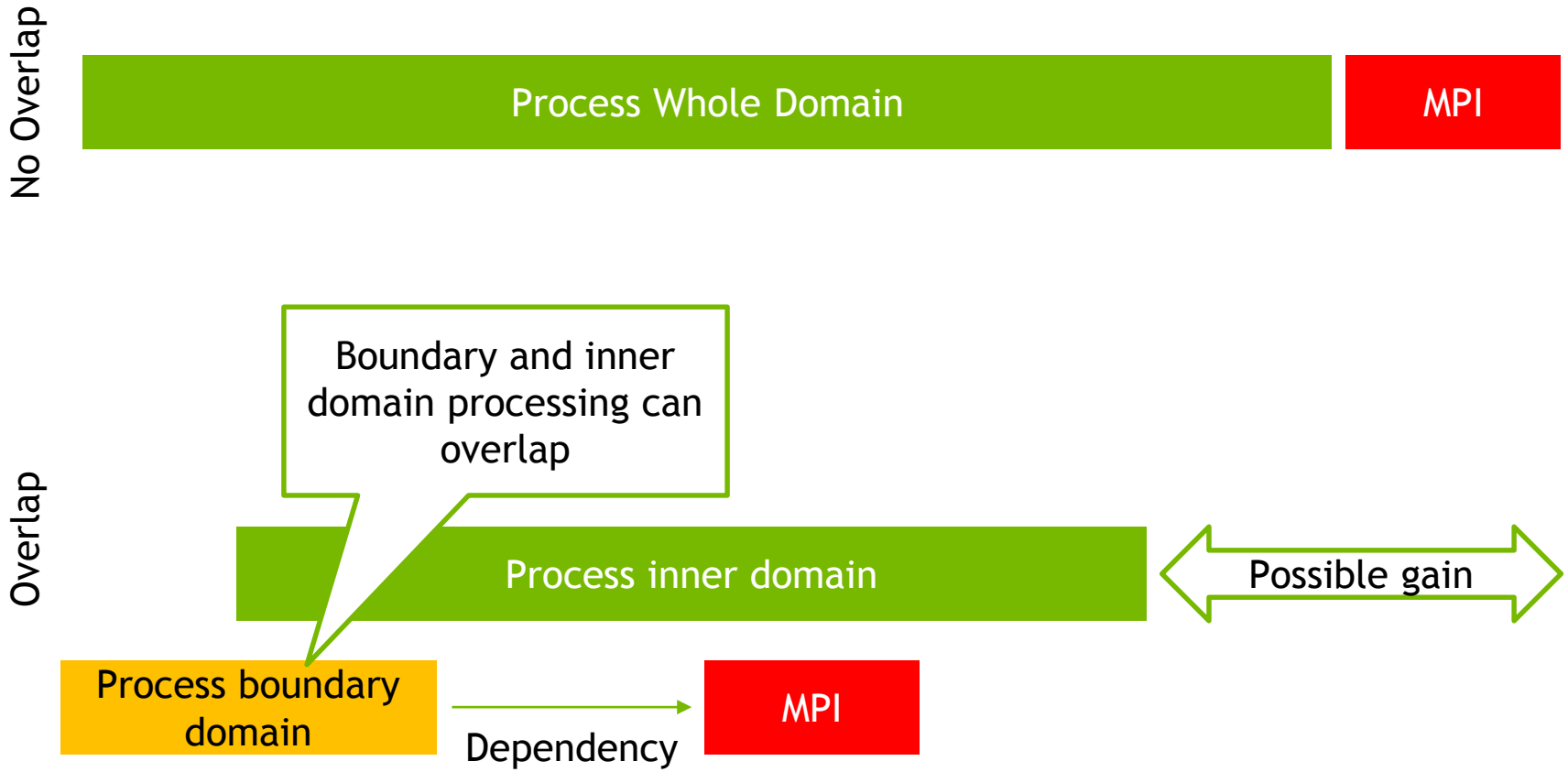
Decreasing parallel overhead

COMMUNICATION + COMPUTATION OVERLAP

OpenMPI 1.10.2 - 4 Tesla K80



COMMUNICATION + COMPUTATION OVERLAP



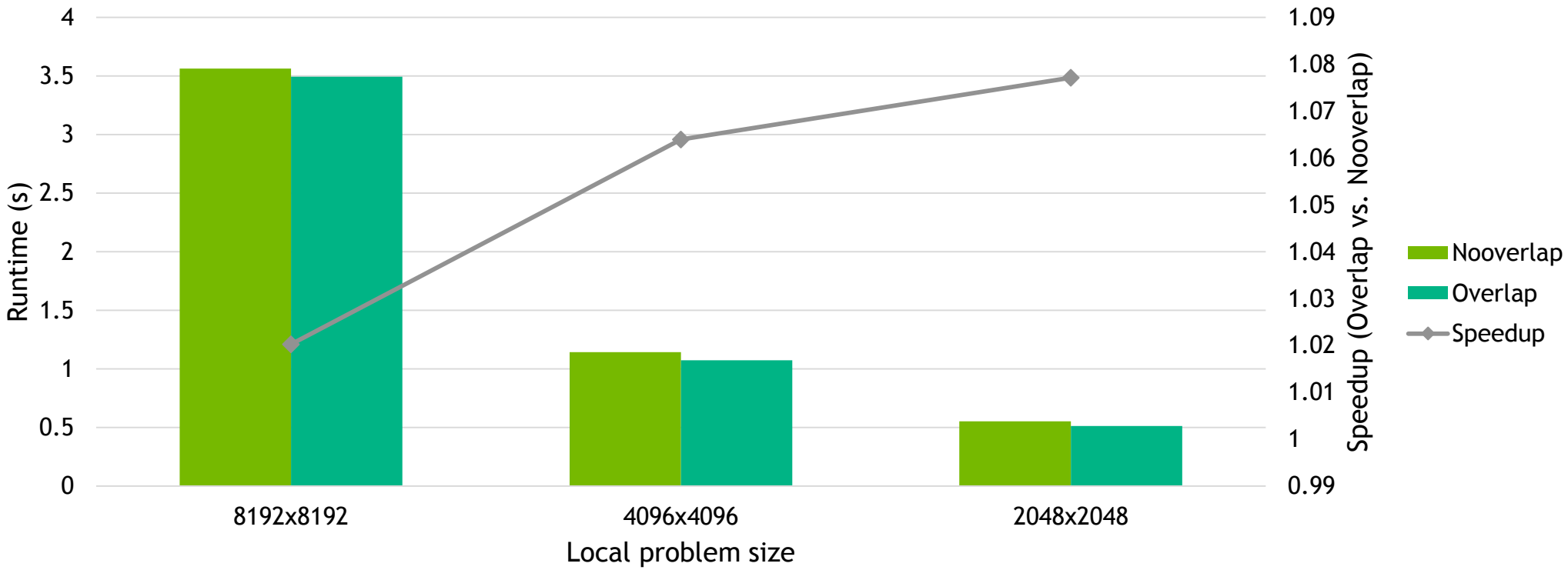
COMMUNICATION + COMPUTATION OVERLAP

OpenACC with Async Queues

```
#pragma acc kernels present ( A, Anew )
for ( ... )
    //Process boundary
#pragma acc kernels present ( A, Anew ) async
for ( ... )
    //Process inner domain
#pragma acc host_data use_device ( A ) {
    MPI_Sendrecv( &A[iy_start][ix_start], (ix_end-ix_start), MPI_DOUBLE, top , 0,
                  &A[iy_end][ix_start], (ix_end-ix_start), MPI_DOUBLE, bottom, 0,
                  MPI_COMM_WORLD, MPI_STATUS_IGNORE );
    MPI_Sendrecv( &A[(iy_end-1)][ix_start], (ix_end-ix_start), MPI_DOUBLE, bottom, 0,
                  &A[(iy_start-1)][ix_start], (ix_end-ix_start), MPI_DOUBLE, top , 0,
                  MPI_COMM_WORLD, MPI_STATUS_IGNORE ); }
#pragma acc wait                    //wait for iteration to finish
```

COMMUNICATION + COMPUTATION OVERLAP

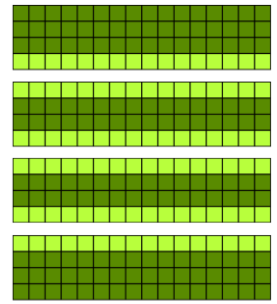
OpenMPI 1.10.2 - 4 Tesla K80



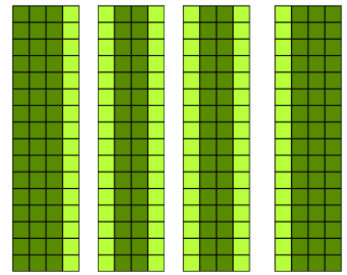
DOMAIN DECOMPOSITION STRATEGIES

Using stripes (2D)/planes(3D)

- Minimizes the number of neighbors
- Avoids noncontiguous halo exchange
- Good for latency bound problems



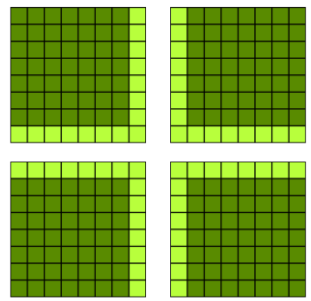
Horizontal Stripes



Vertical Stripes

Using tiles (2D)/using boxes (3D)

- Minimizes surface to volume ratio
- Requires noncontiguous halo exchange
- Good for bandwidth bound problems



Tiles

EXAMPLE: JACOBI SOLVER

Multi GPU with tiled domain decomposition

While not converged

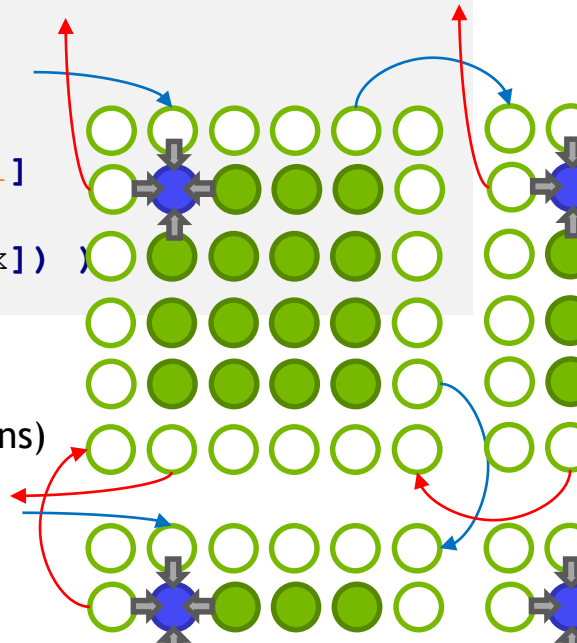
Do Jacobi step:

```
for (int iy = iy_start; iy < iy_end; ++iy)

for (int ix = ix_start; ix < ix_end; ++ix)

    Anew[iy][ix] = - 0.25f*(rhs[iy][ix] - ( A[iy][ix-1] + A[iy][ix+1]

                                          + A[iy-1][ix] + A[iy+1][ix]) )
```

Copy `Anew` to `A`

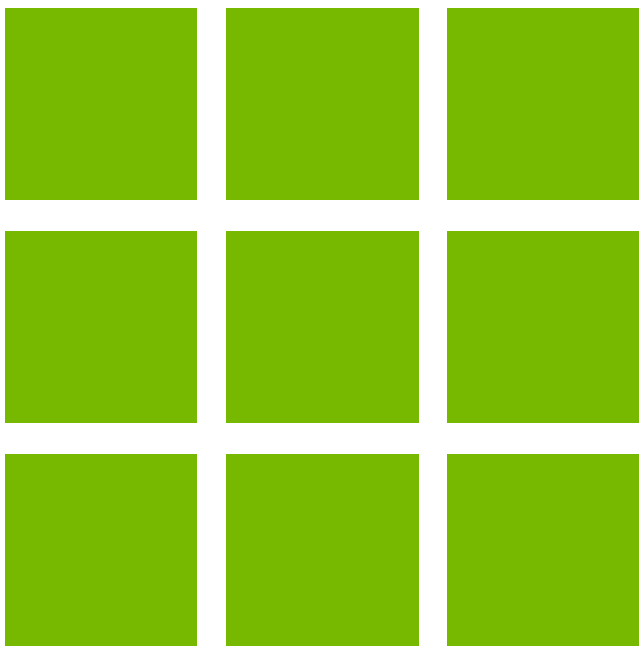
Exchange halo with 4 neighbors (ring exchange implicitly handles periodic boundary conditions)

Next iteration



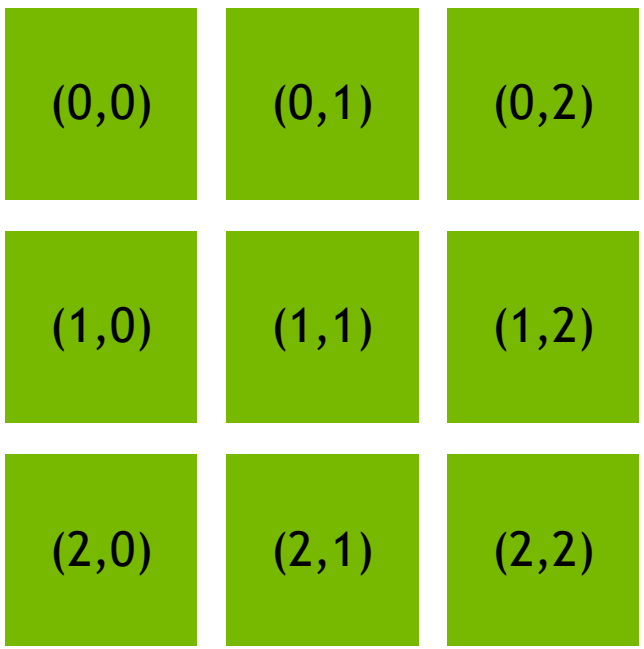
DOMAIN DECOMPOSITION WITH TILES

Handling Tile <-> GPU/MPI Rank affinity



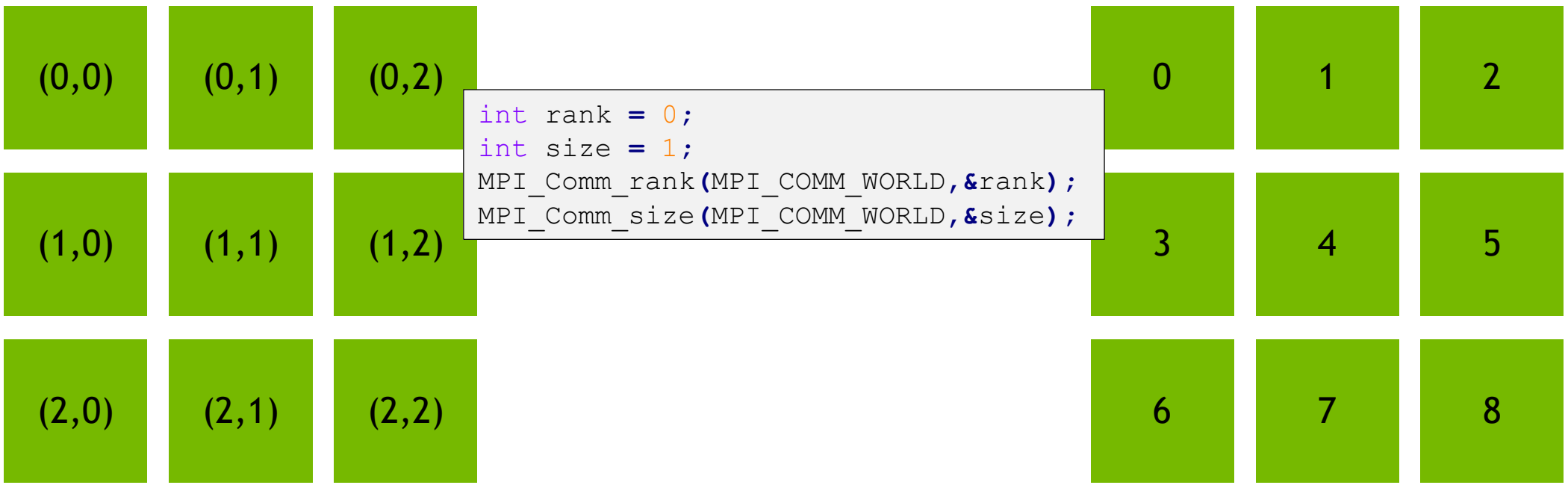
DOMAIN DECOMPOSITION WITH TILES

Handling Tile <-> GPU/MPI Rank affinity



DOMAIN DECOMPOSITION WITH TILES

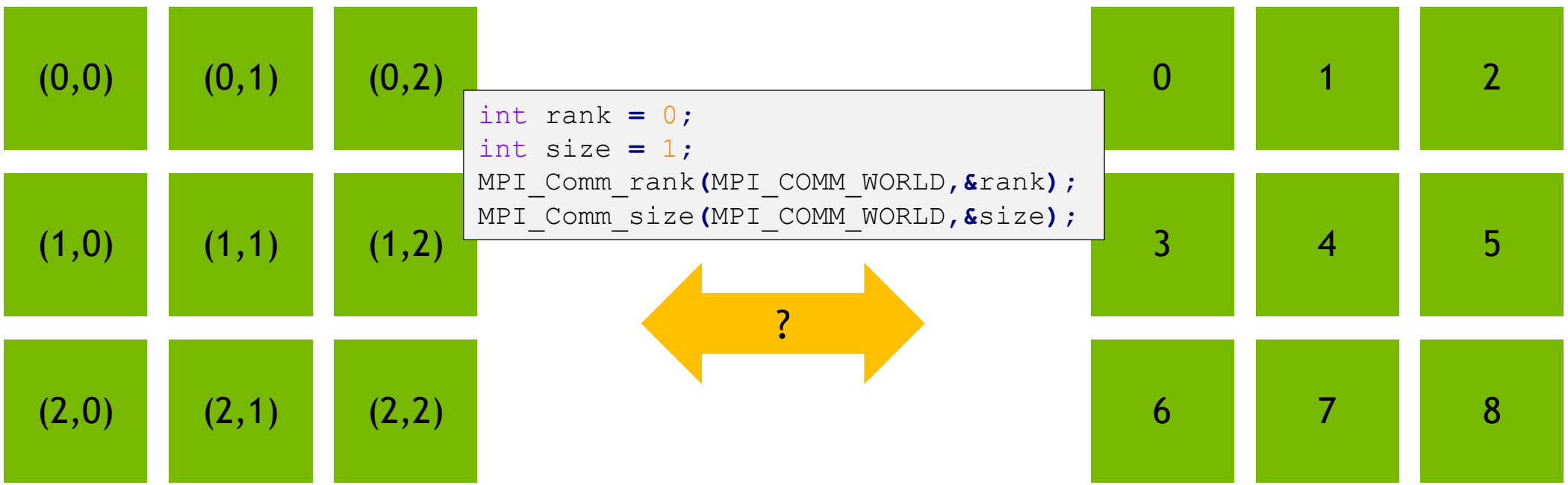
Handling Tile <-> GPU/MPI Rank affinity




```
int rank = 0;
int size = 1;
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
MPI_Comm_size(MPI_COMM_WORLD,&size);
```

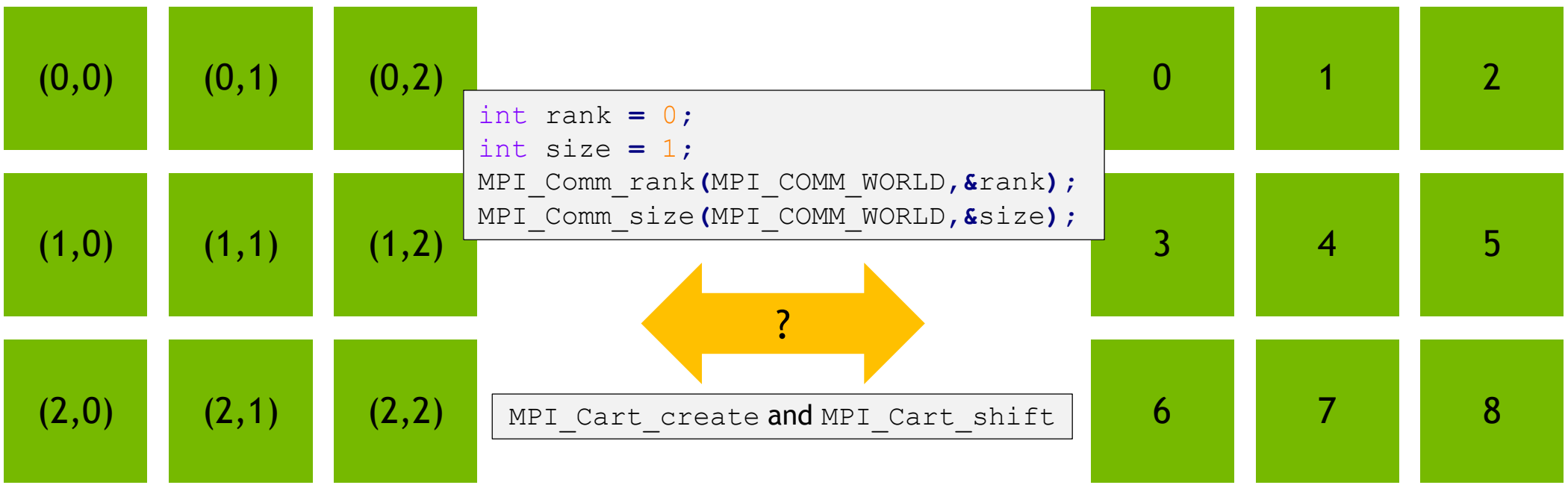
DOMAIN DECOMPOSITION WITH TILES

Handling Tile <-> GPU/MPI Rank affinity



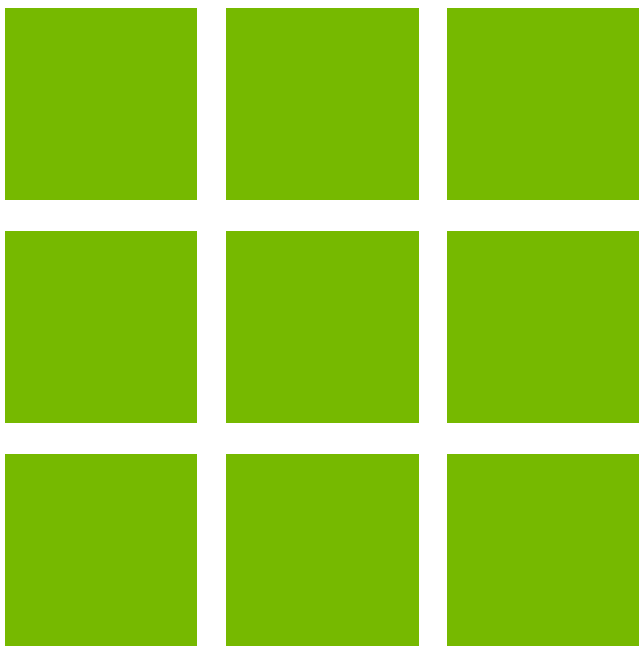
DOMAIN DECOMPOSITION WITH TILES

Handling Tile <-> GPU/MPI Rank affinity



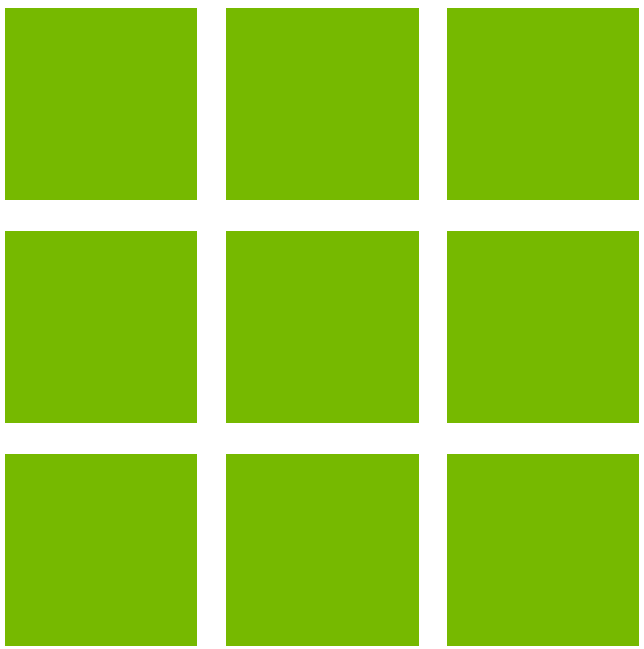
DOMAIN DECOMPOSITION WITH TILES

Handling Tile <-> GPU/MPI Rank affinity



DOMAIN DECOMPOSITION WITH TILES

Handling Tile <-> GPU/MPI Rank affinity

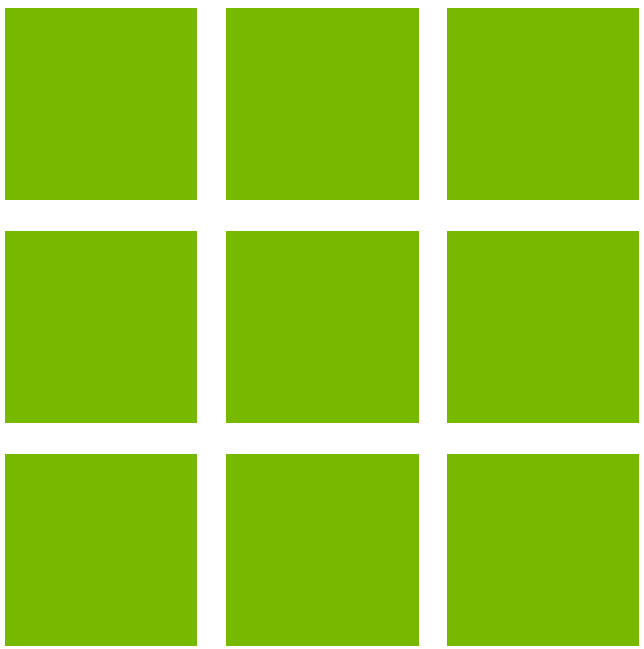


1. Determine Ranks in x direction and y direction:

```
int size = 1;
MPI_Comm_size(MPI_COMM_WORLD,&size);
int sizex = 3;
int sizey = size/sizex;
assert(sizex*sizey == size);
```

DOMAIN DECOMPOSITION WITH TILES

Handling Tile <-> GPU/MPI Rank affinity



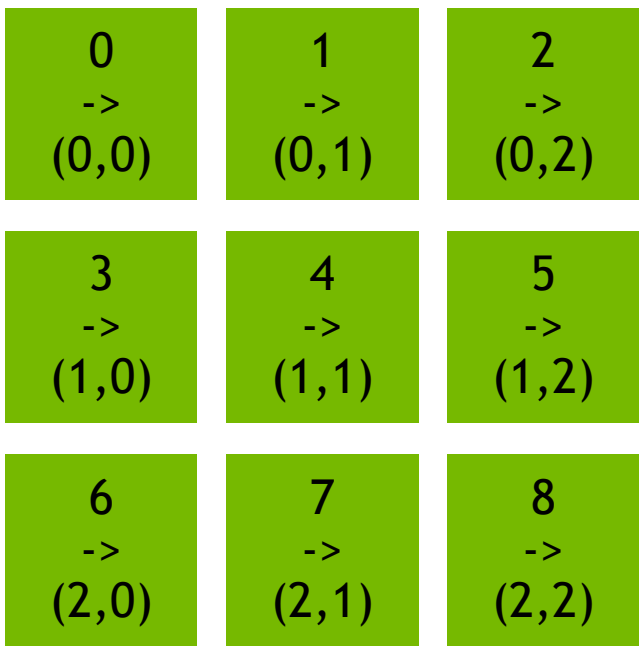
1. Determine Ranks in x direction and y direction:

```
int size = 1;
MPI_Comm_size(MPI_COMM_WORLD,&size);
int sizex = 3;
int sizey = size/sizex;
assert(sizex*sizey == size);
```

2. Map 1D MPI rank to 2D MPI ranks

DOMAIN DECOMPOSITION WITH TILES

Handling Tile <-> GPU/MPI Rank affinity



1. Determine Ranks in x direction and y direction:

```
int size = 1;
MPI_Comm_size(MPI_COMM_WORLD,&size);
int sizex = 3;
int sizey = size/sizex;
assert(sizex*sizey == size);
```

2. Map 1D MPI rank to 2D MPI ranks

```
int rank = 0;
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
int rankx = rank%sizex;
int ranky = rank/sizex;
```

DOMAIN DECOMPOSITION WITH TILES

Handling Tile <-> GPU/MPI Rank affinity

(0,0) -> 0	(0,1) -> 1	(0,2) -> 2
(1,0) -> 3	(1,1) -> 4	(1,2) -> 5
(2,0) -> 6	(2,1) -> 7	(2,2) -> 8

DOMAIN DECOMPOSITION WITH TILES

Handling Tile <-> GPU/MPI Rank affinity

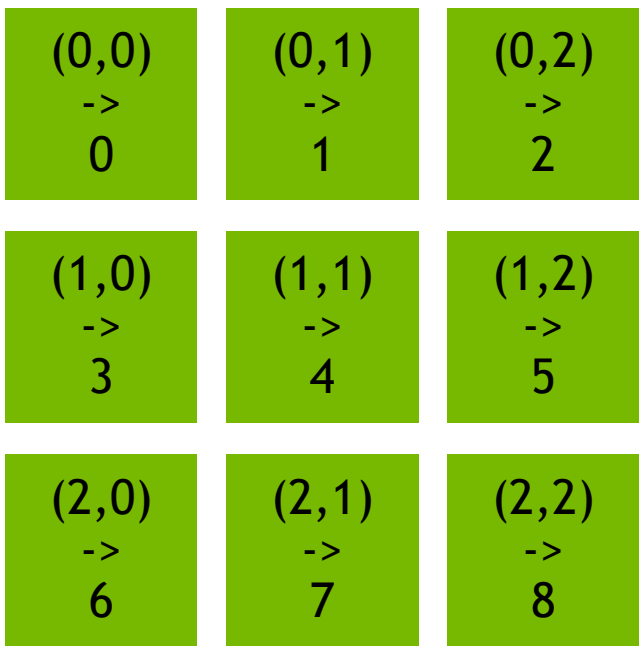
(0,0) -> 0	(0,1) -> 1	(0,2) -> 2
(1,0) -> 3	(1,1) -> 4	(1,2) -> 5
(2,0) -> 6	(2,1) -> 7	(2,2) -> 8

1. Determine neighbors in y direction:

```
int lefty  = (ranky == 0) ? (sizey-1) : ranky-1;
int righty = (ranky == (sizey-1)) ? 0 : ranky+1;
```

DOMAIN DECOMPOSITION WITH TILES

Handling Tile <-> GPU/MPI Rank affinity



1. Determine neighbors in y direction:

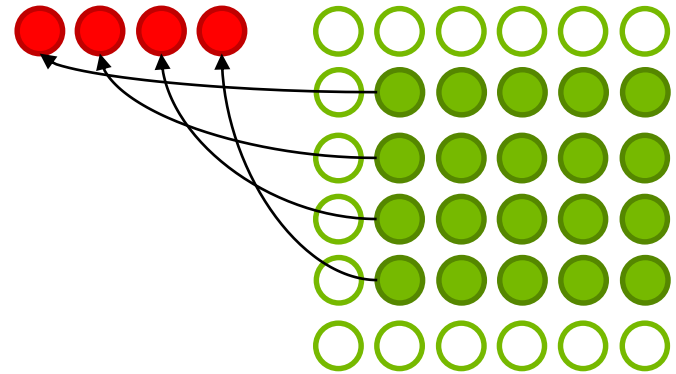
```
int lefty  = (ranky == 0) ? (sizey-1) : ranky-1;
int righty = (ranky == (sizey-1)) ? 0 : ranky+1;
```

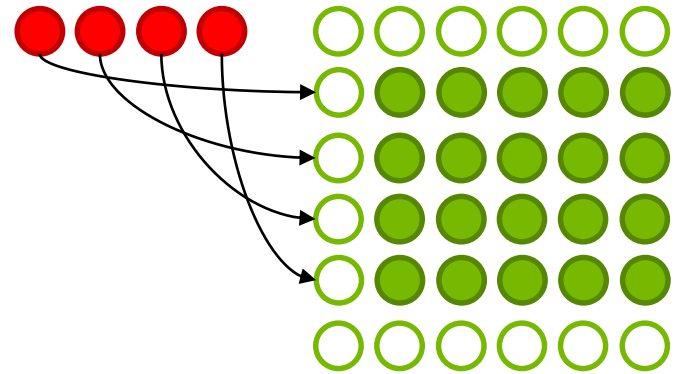
2. Map 2D MPI rank to 1D MPI ranks

```
int left  = lefty  * sizex + rankx;
int right = righty * sizex + rankx;
```

EXAMPLE: JACOBI

Left/Right Halo





```
//right neighbor omitted
#pragma acc kernels present ( A, to_left )
for (int iy = iy_start; iy < iy_end; iy++)
        to_left[iy] = A[iy][ix_start];

#pragma acc host_data use_device ( from_right, to_left ) {
  MPI_Sendrecv( to_left, NY-2, MPI_DOUBLE, left, 0,
                from_right, NY-2, MPI_DOUBLE, right, 0,
                MPI_COMM_WORLD, MPI_STATUS_IGNORE );
}

#pragma acc kernels present ( A, from_right )
for (int iy = iy_start; iy < iy_end; iy++)
    A[iy][ix_end] = from_right[iy];
```

Homework

ACCESS TO HOMEWORK

Qwiklabs: Getting access

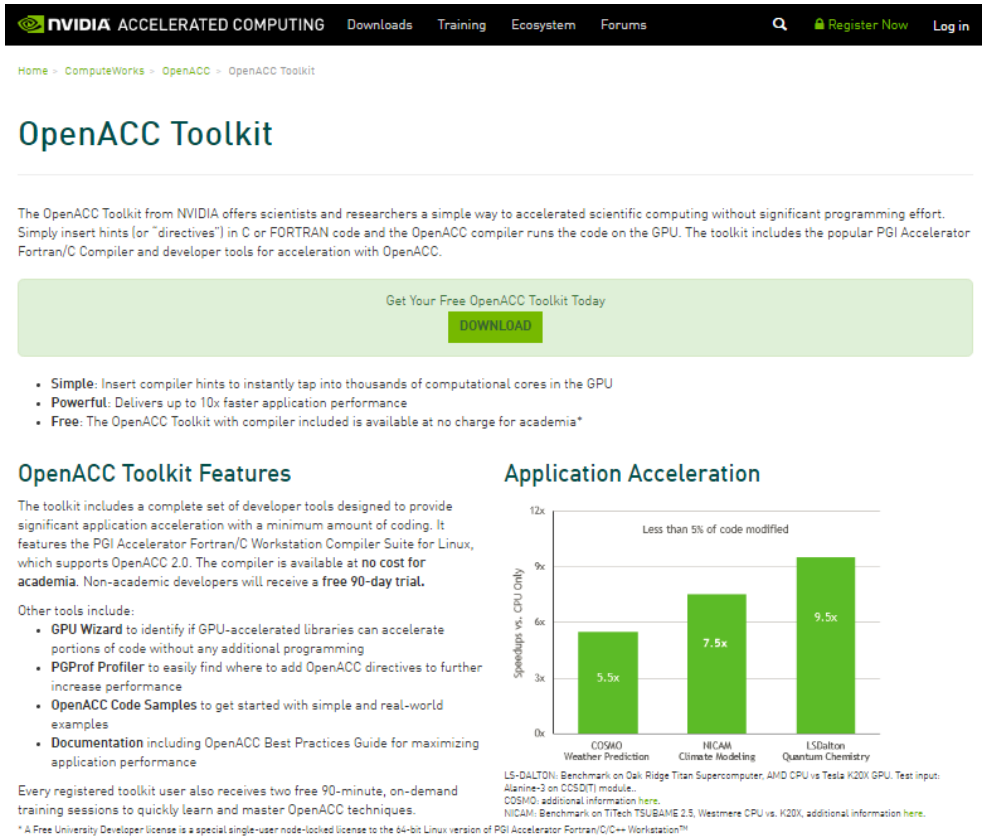
1. Go to https://developer.nvidia.com/qwiklabs-signup
2. Use OPENACC promo code to register and get free access
3. Receive a confirmation email with access instructions
4. Take ‘Advanced Multi GPU Programming with MPI and OpenACC’ lab: http://bit.ly/oaccnvlab7

Questions?

Email to openacc@nvidia.com

INSTALL THE OPENACC TOOLKIT (OPTIONAL)

- Go to developer.nvidia.com/openacc-toolkit
- Register for the OpenACC Toolkit
- Install on your personal machine (Linux Only)
- Free workstation license for academia/90 day free trial for the rest



Course Syllabus

May 19: Advanced Profiling of OpenACC Code

May 26: Office Hours

June 2: Advanced multi-GPU Programming with MPI and OpenACC

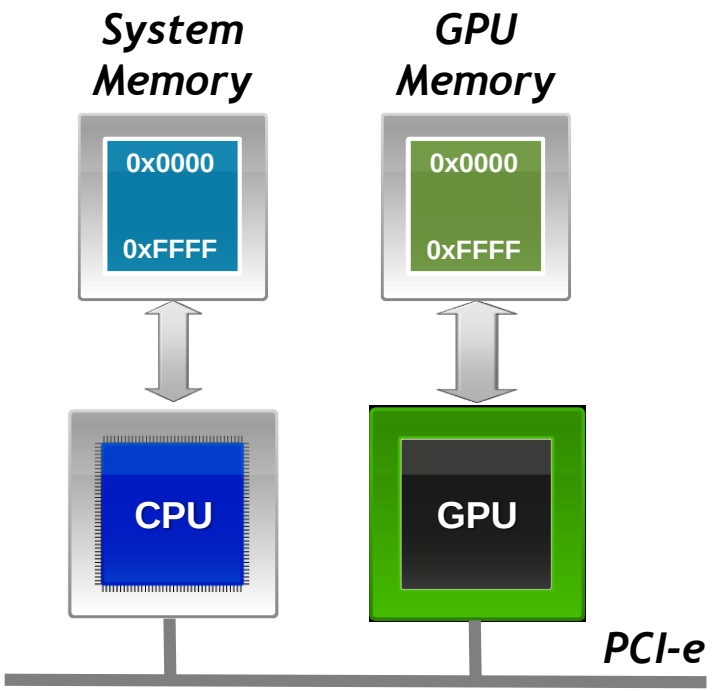
Recordings:
https://developer.nvidia.com/openacc-advanced-course

Questions? Email openacc@nvidia.com

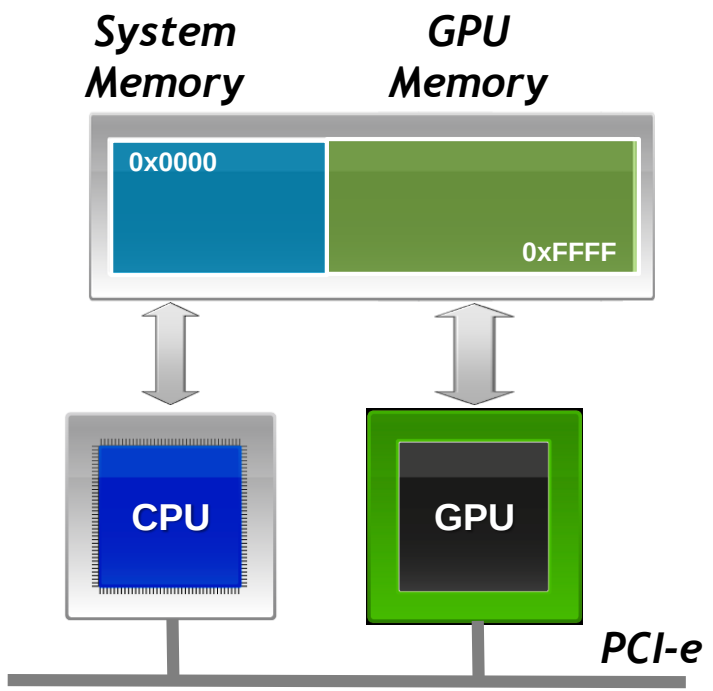
CUDA-aware MPI implementation details

UNIFIED VIRTUAL ADDRESSING

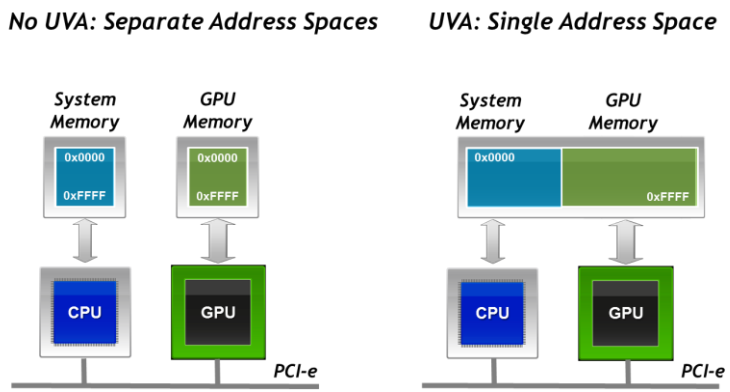
No UVA: Separate Address Spaces



UVA: Single Address Space



UNIFIED VIRTUAL ADDRESSING



One address space for all CPU and GPU memory

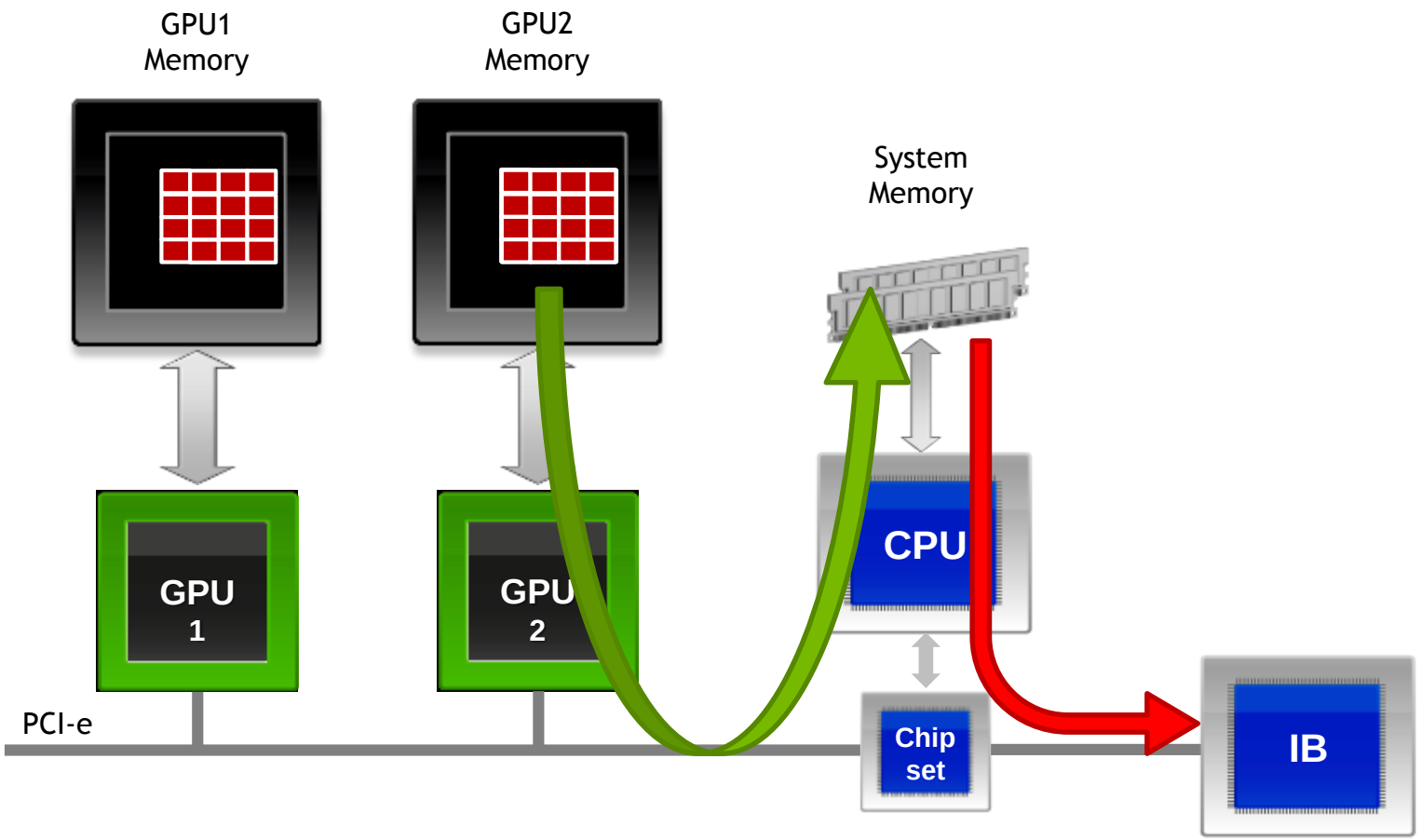
- Determine physical memory location from a pointer value
- Enable libraries to simplify their interfaces (e.g. MPI and cudaMemcpy)

Supported on devices with compute capability 2.0+ for

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

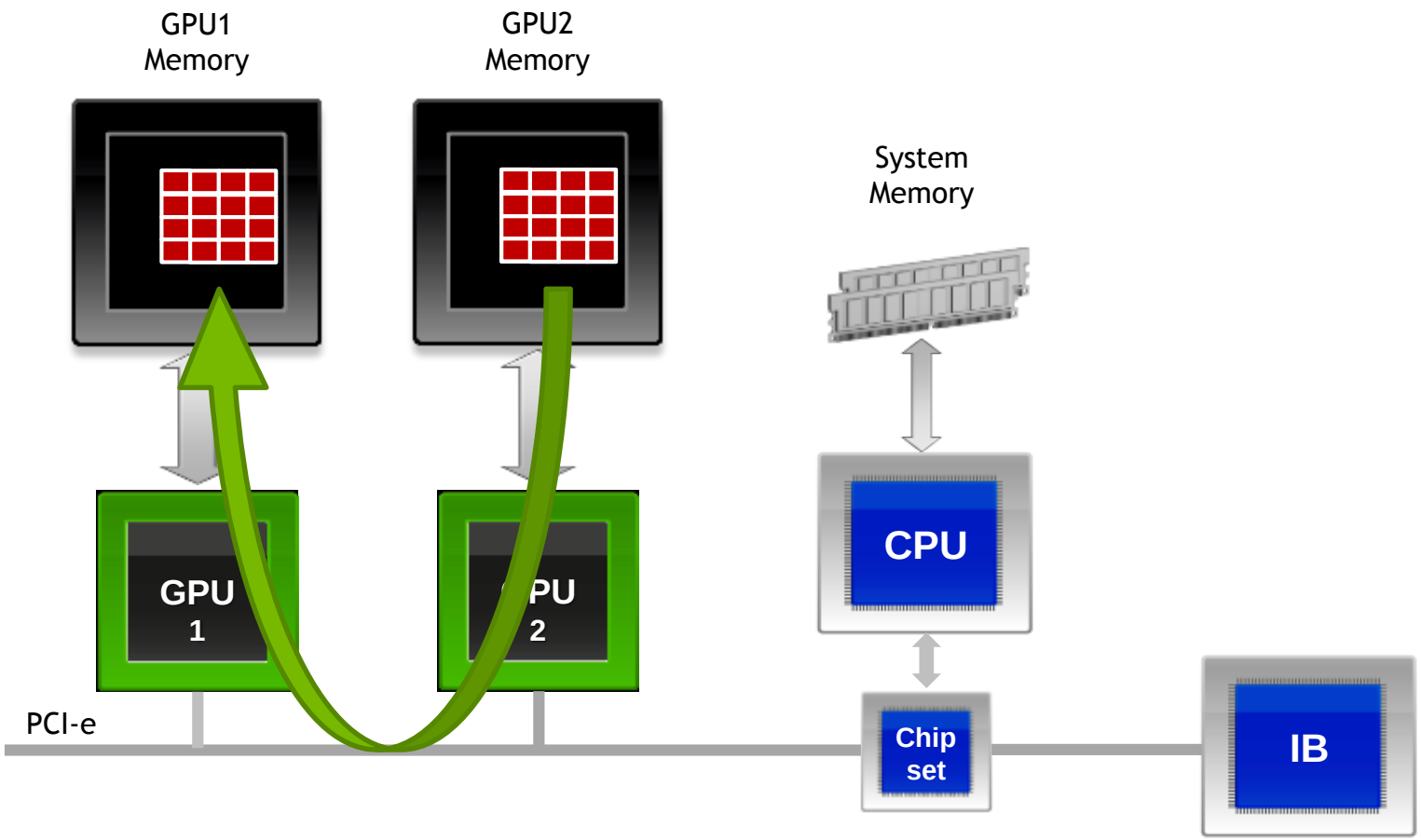
NVIDIA GPUDIRECT™

Accelerated Communication with Network & Storage Devices



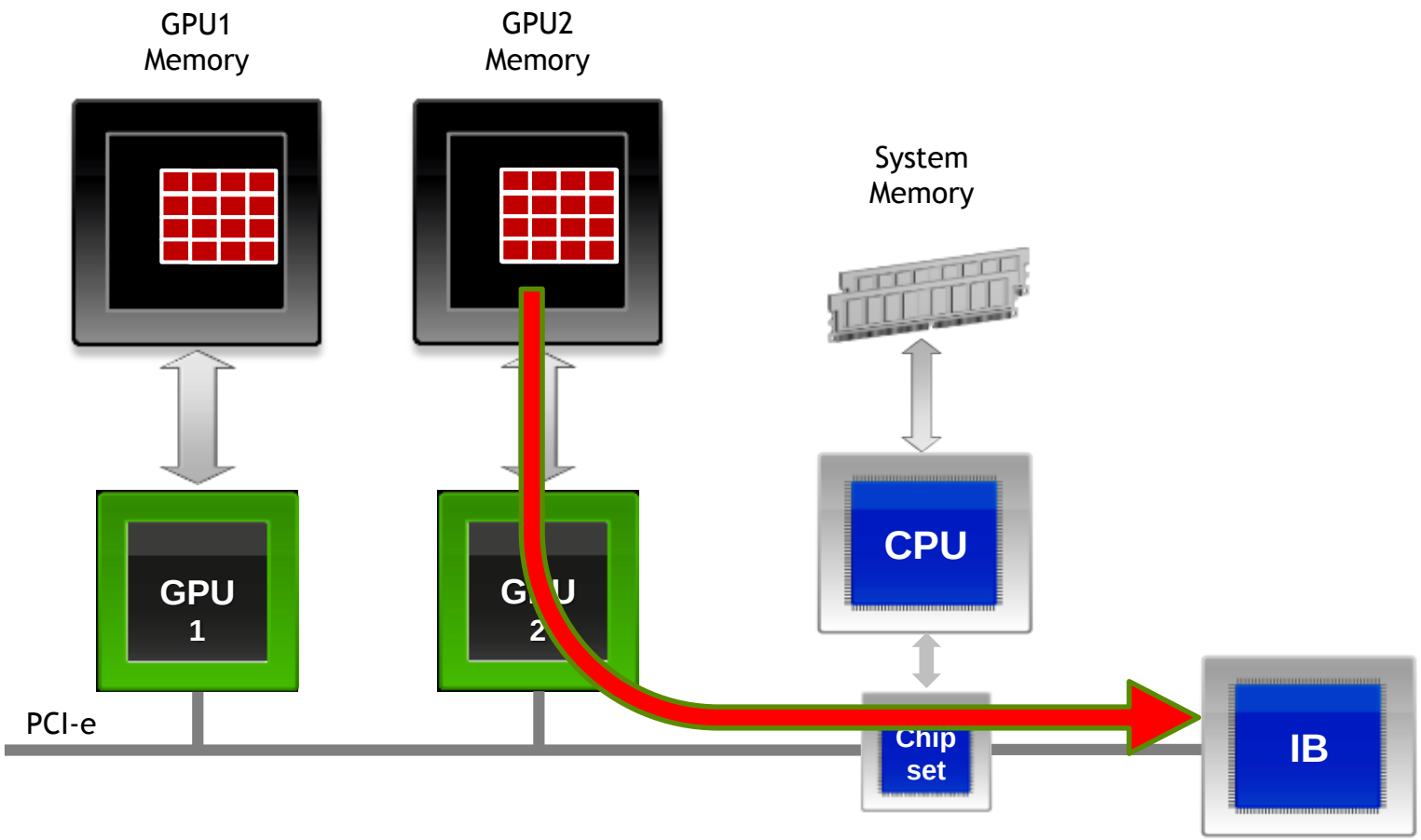
NVIDIA GPUDIRECT™

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

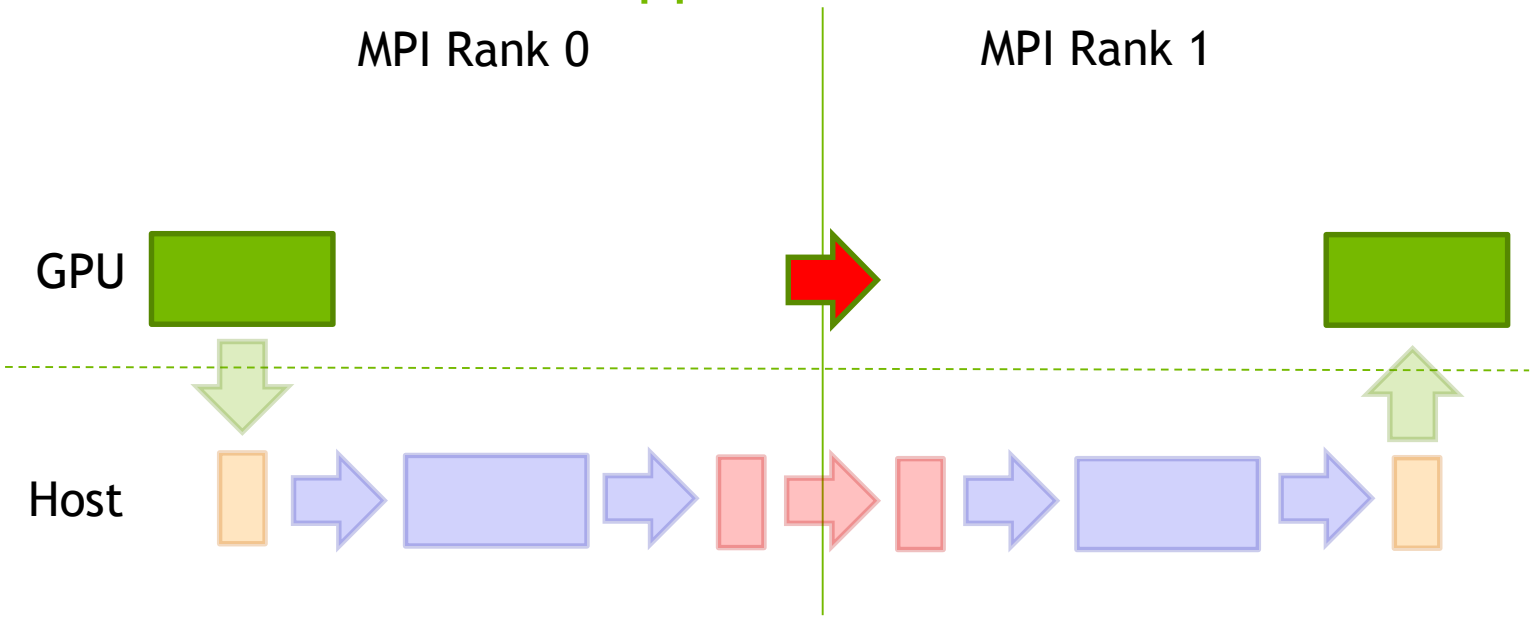
Show how CUDA+MPI works in principle

> 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

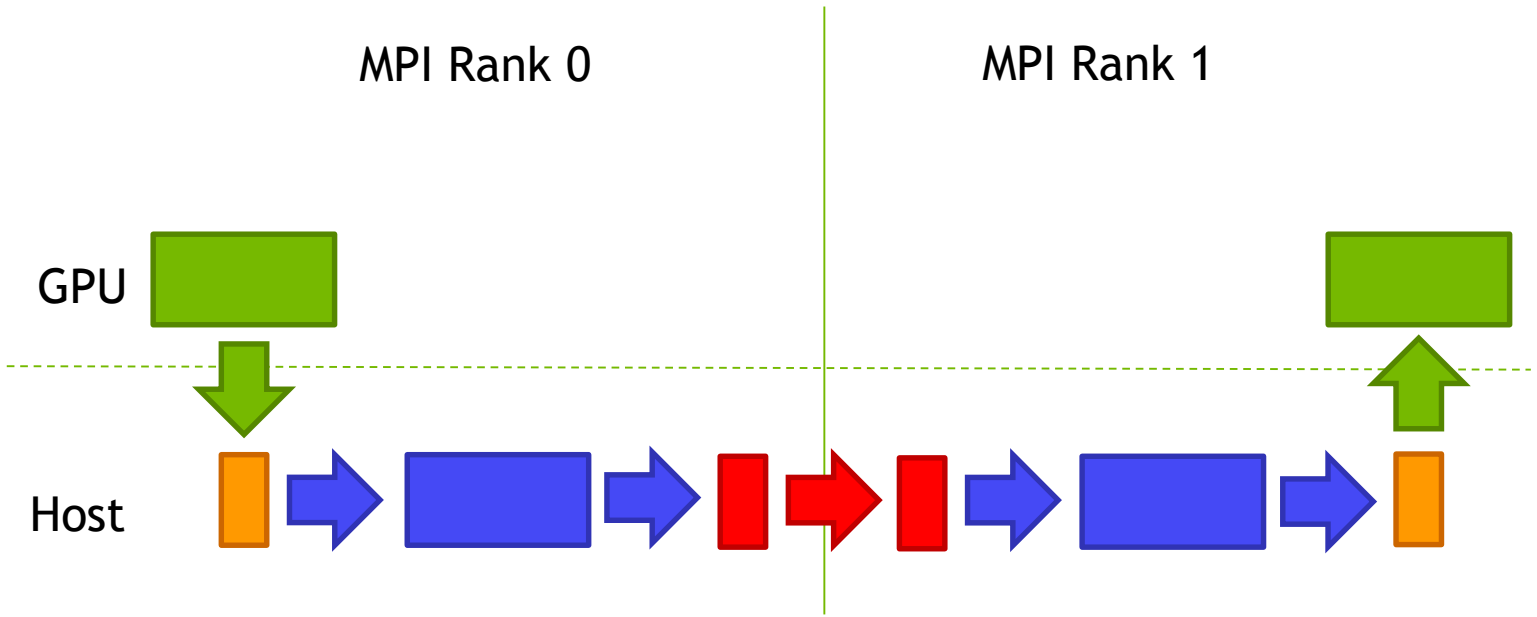



```
MPI_Send(s_buf_d,size,MPI_CHAR,1,tag,MPI_COMM_WORLD);

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

MPI GPU TO REMOTE GPU

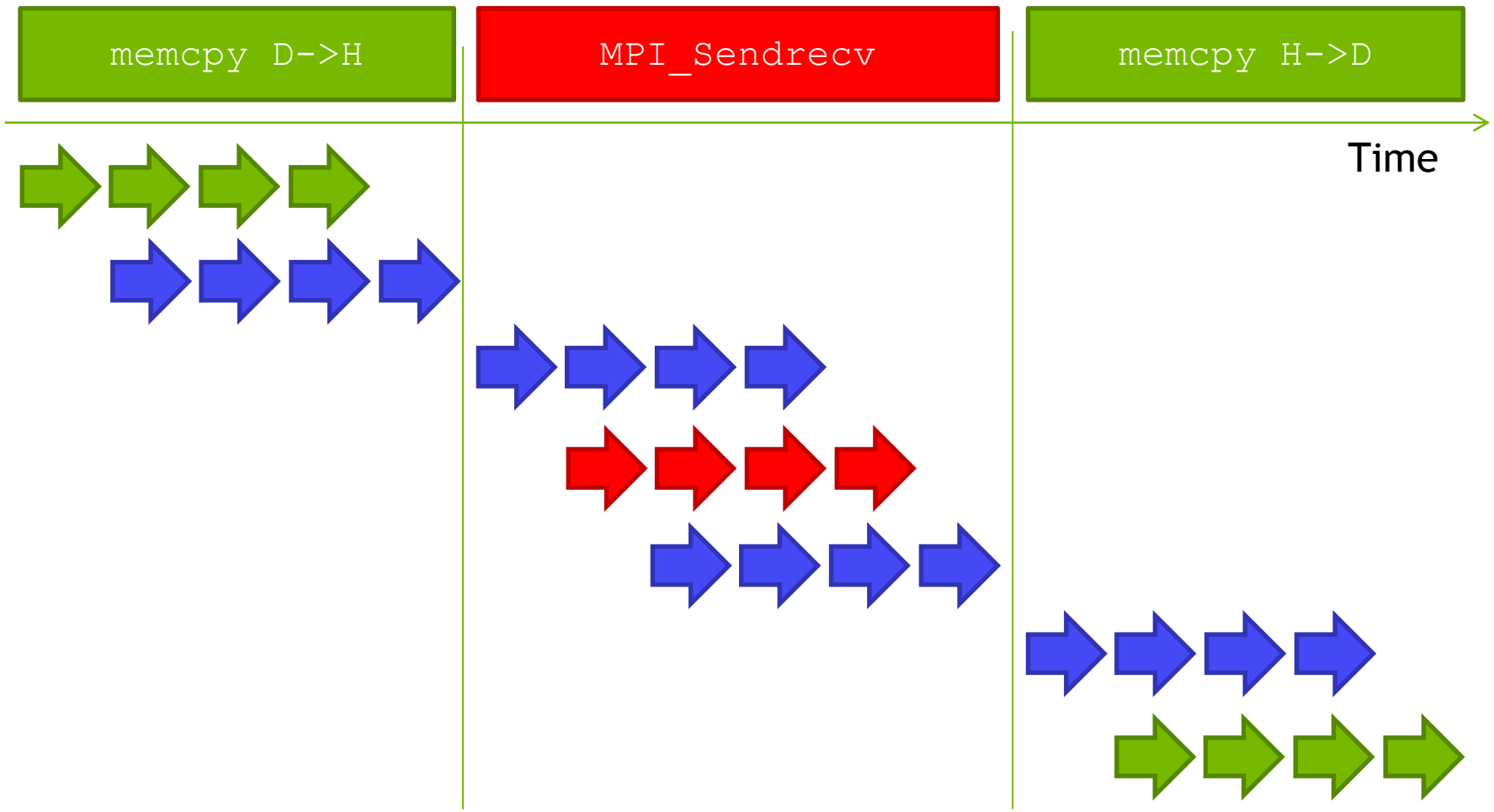
Support for RDMA

REGULAR MPI GPU TO REMOTE GPU



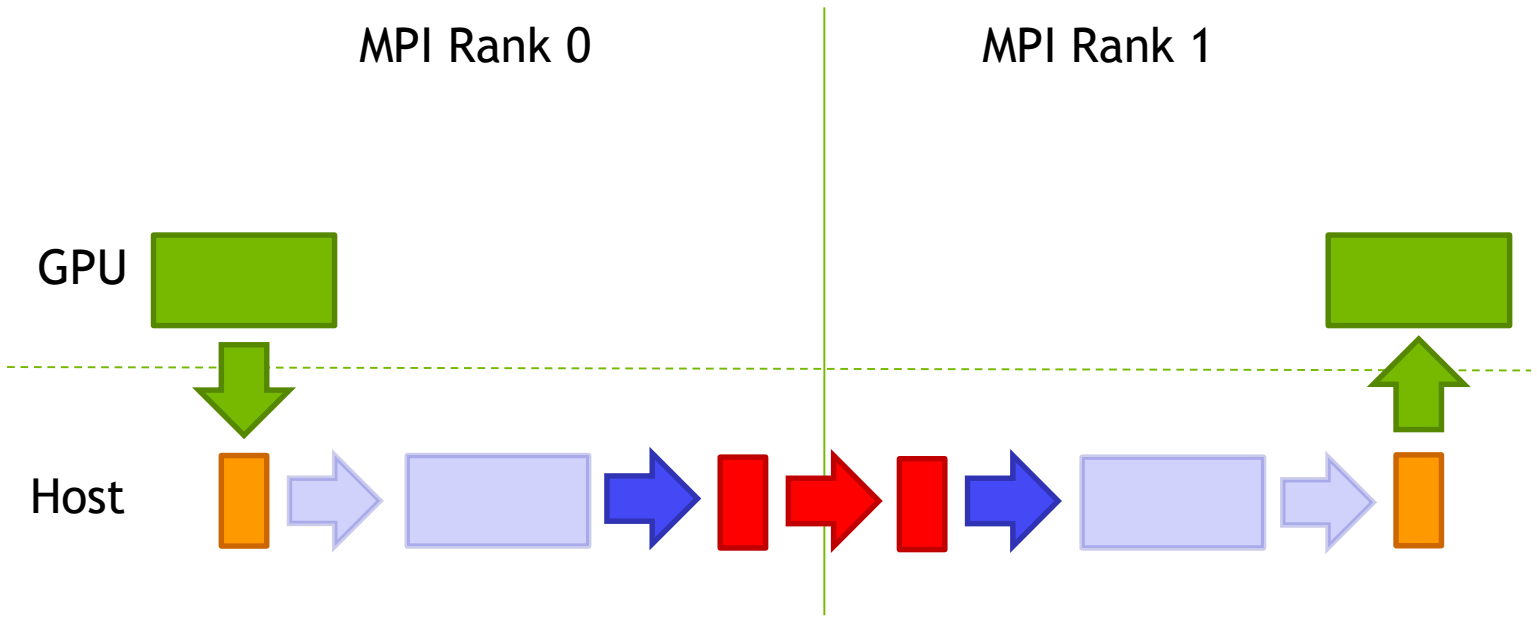

```
cudaMemcpy(s_buf_h,s_buf_d,size,cudaMemcpyDeviceToHost);
MPI_Send(s_buf_h,size,MPI_CHAR,1,tag,MPI_COMM_WORLD);

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

REGULAR MPI GPU TO REMOTE GPU



MPI GPU TO REMOTE GPU

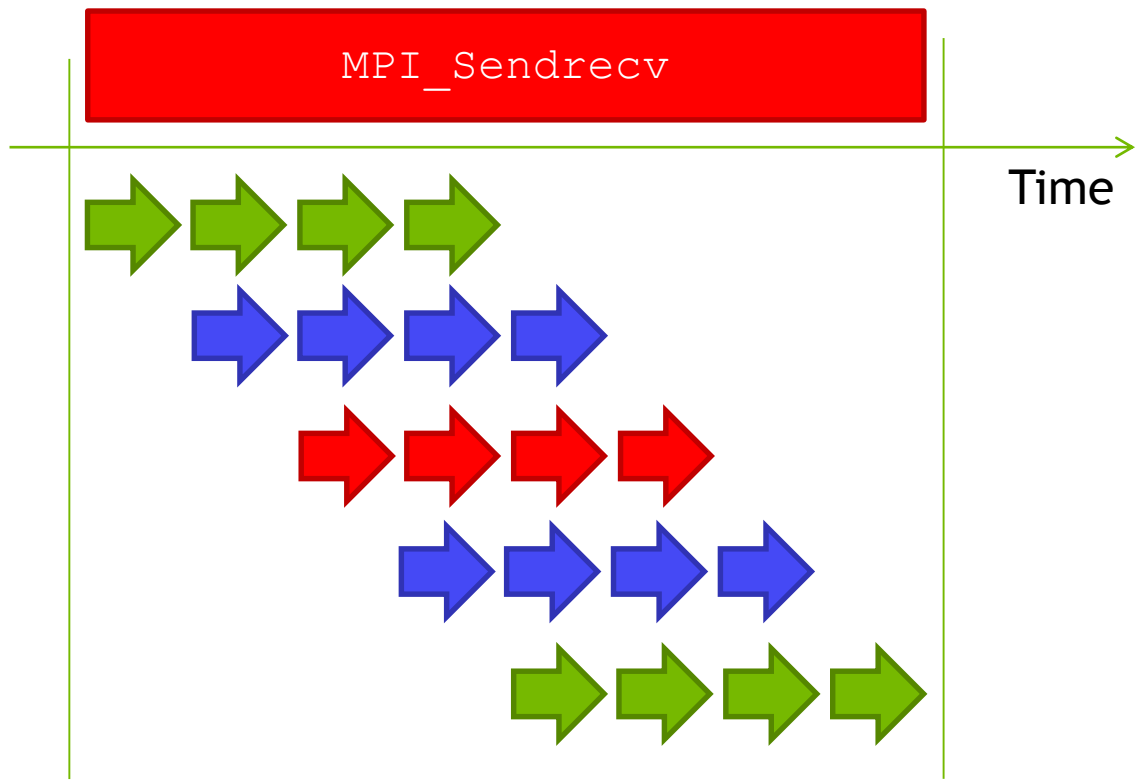
without GPUDirect




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

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

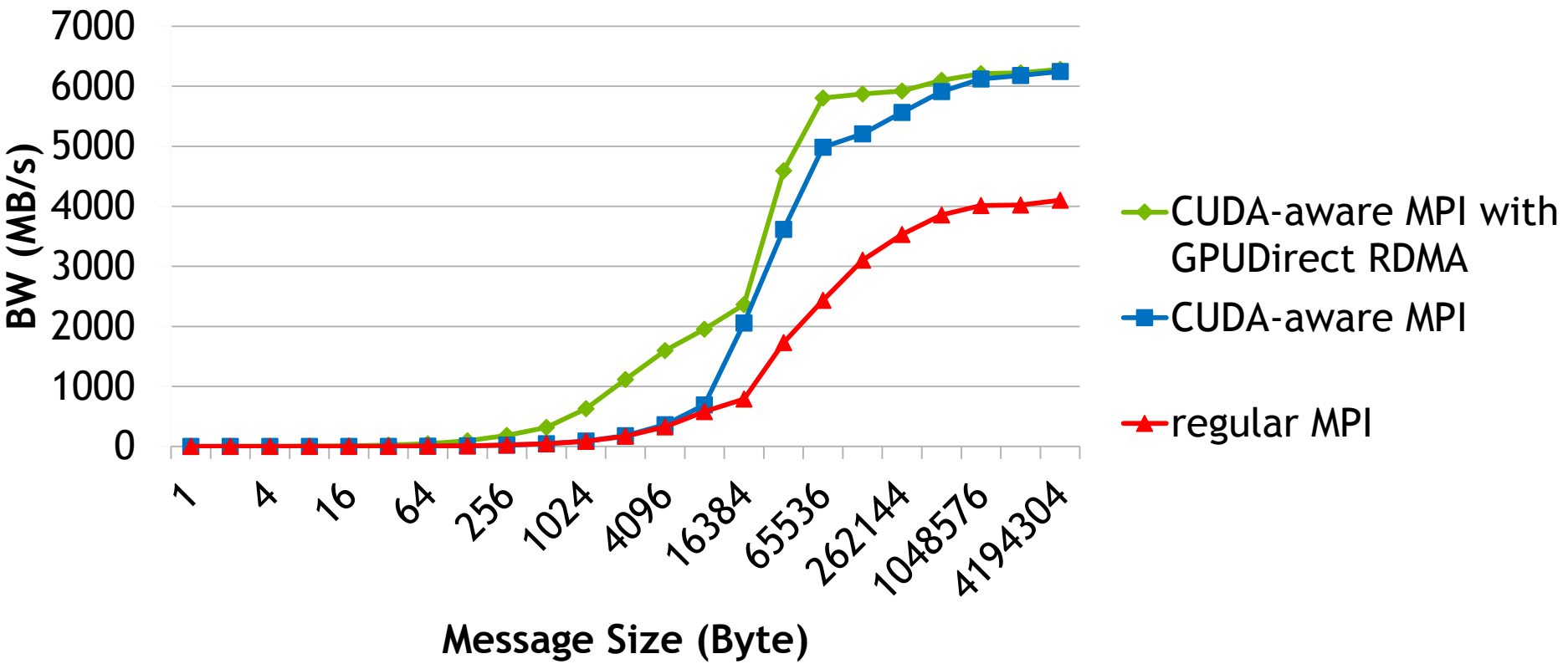
MPI GPU TO REMOTE GPU

without GPUDirect



PERFORMANCE RESULTS TWO NODES

OpenMPI 1.10.2 MLNX FDR IB (4X) Tesla K40@875



Latency (1 Byte)	24.99 us	21.72 us	5.65 us