

# 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

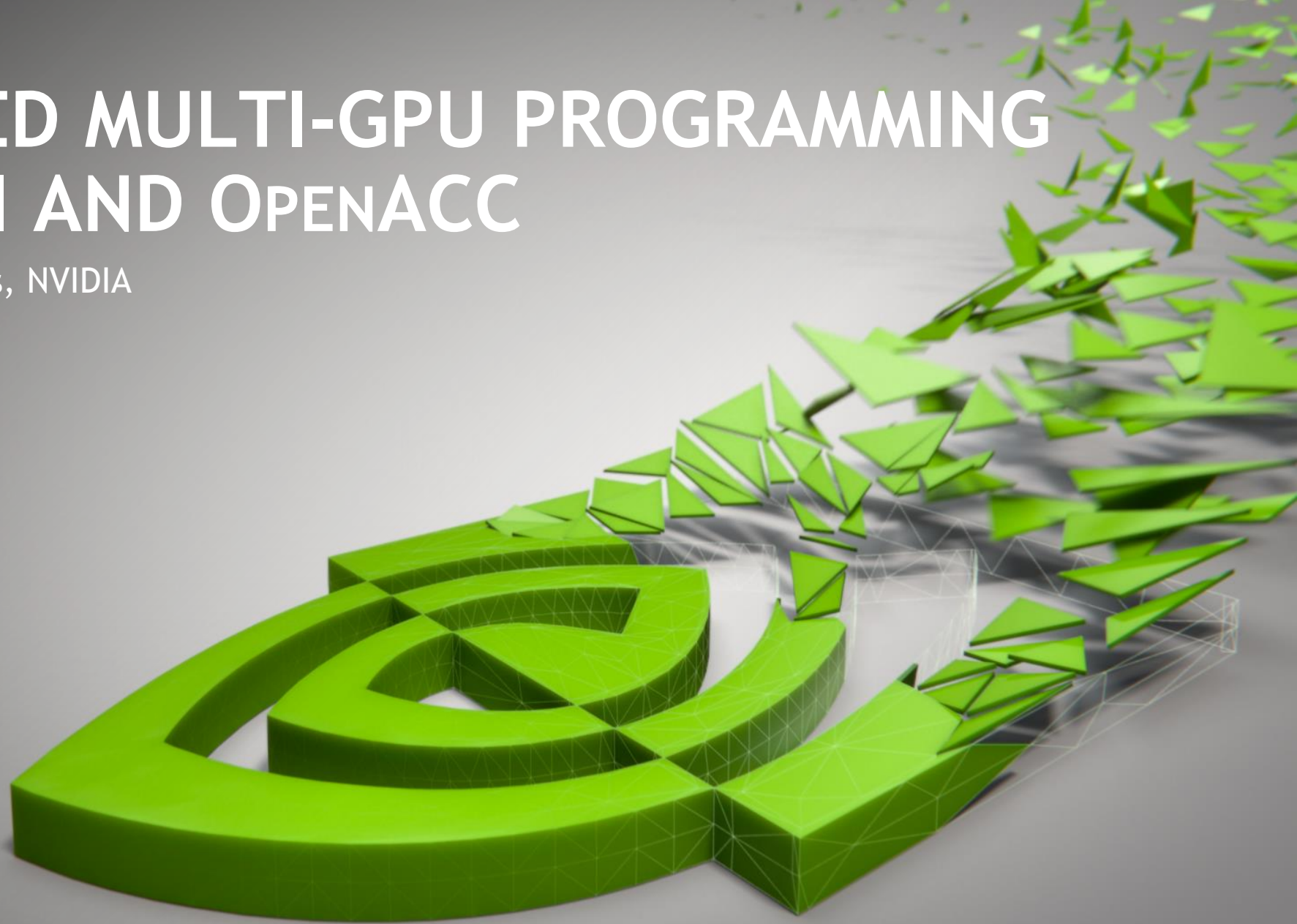
June 2: Advanced multi-GPU Programming with  
MPI and OpenACC

## Recordings:

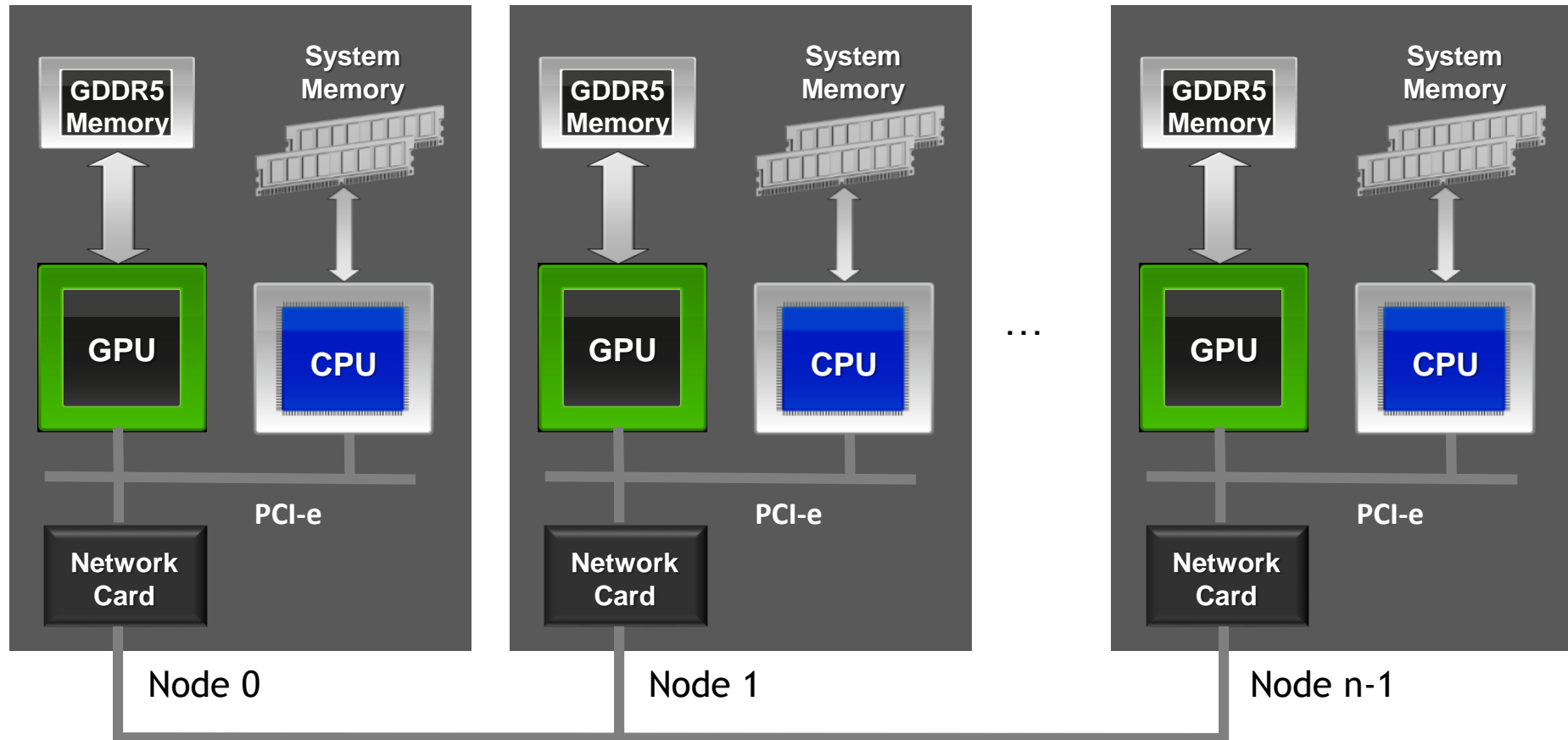
<https://developer.nvidia.com/openacc-advanced-course>

# ADVANCED MULTI-GPU PROGRAMMING WITH MPI AND OPENACC

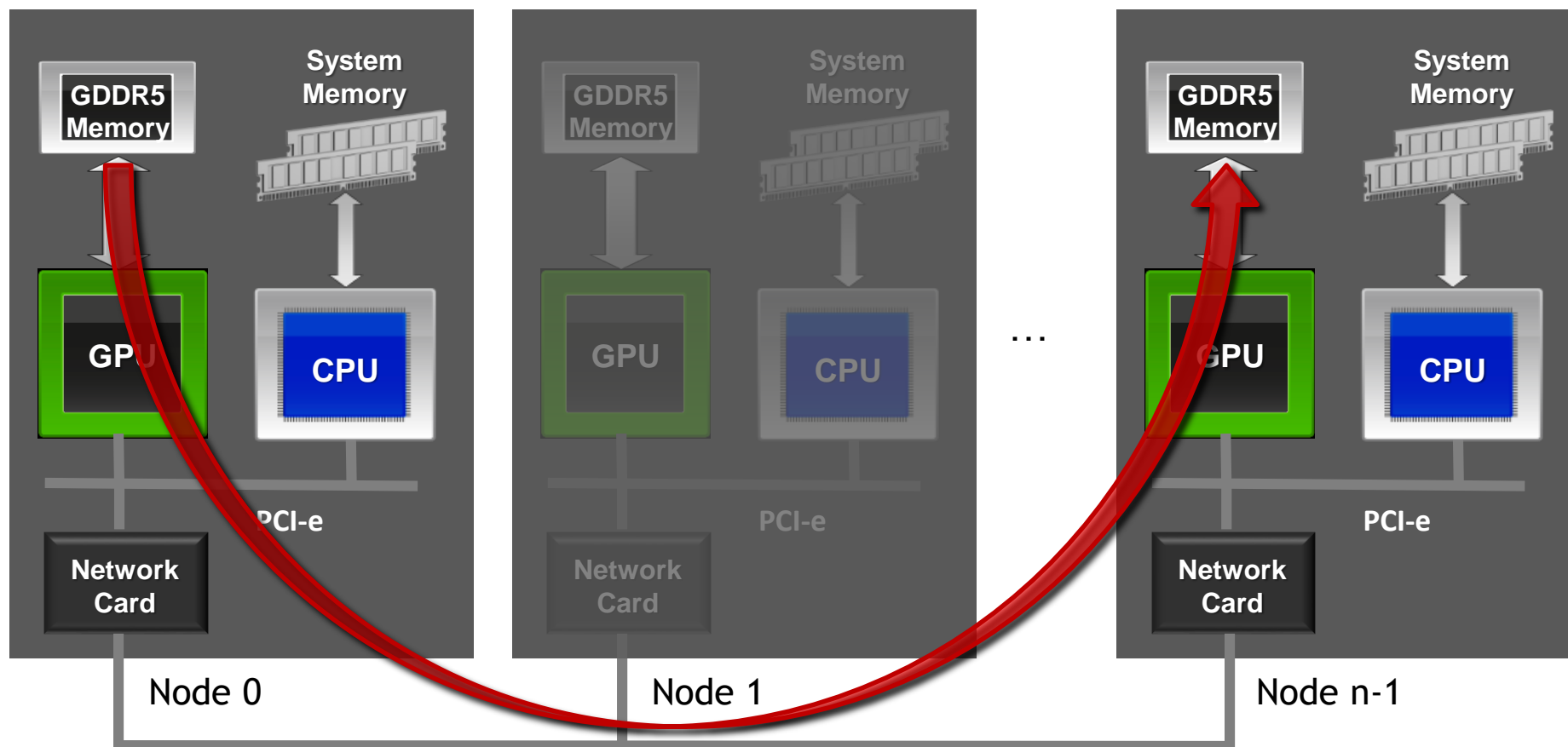
Lecture 2: Jiri Kraus, NVIDIA



# MPI+OPENACC

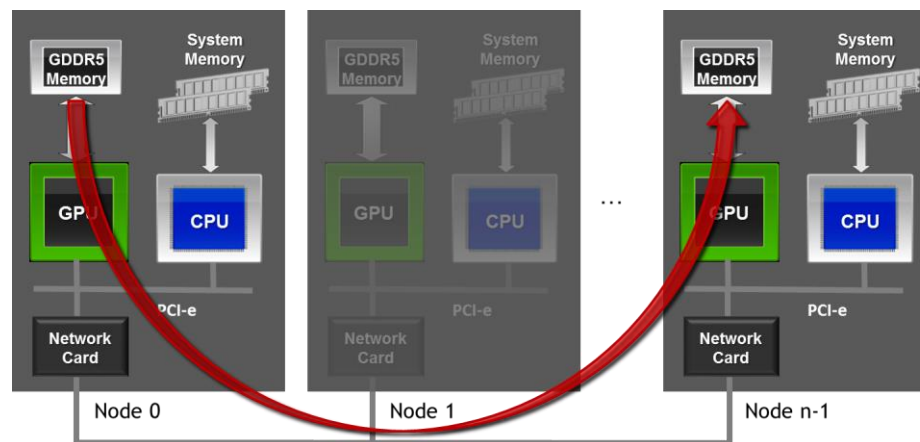


# MPI+OPENACC





# 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



The background is a solid green color with a subtle, abstract pattern of overlapping geometric shapes, primarily triangles and polygons, in a lighter shade of green, creating a textured, low-poly effect.

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

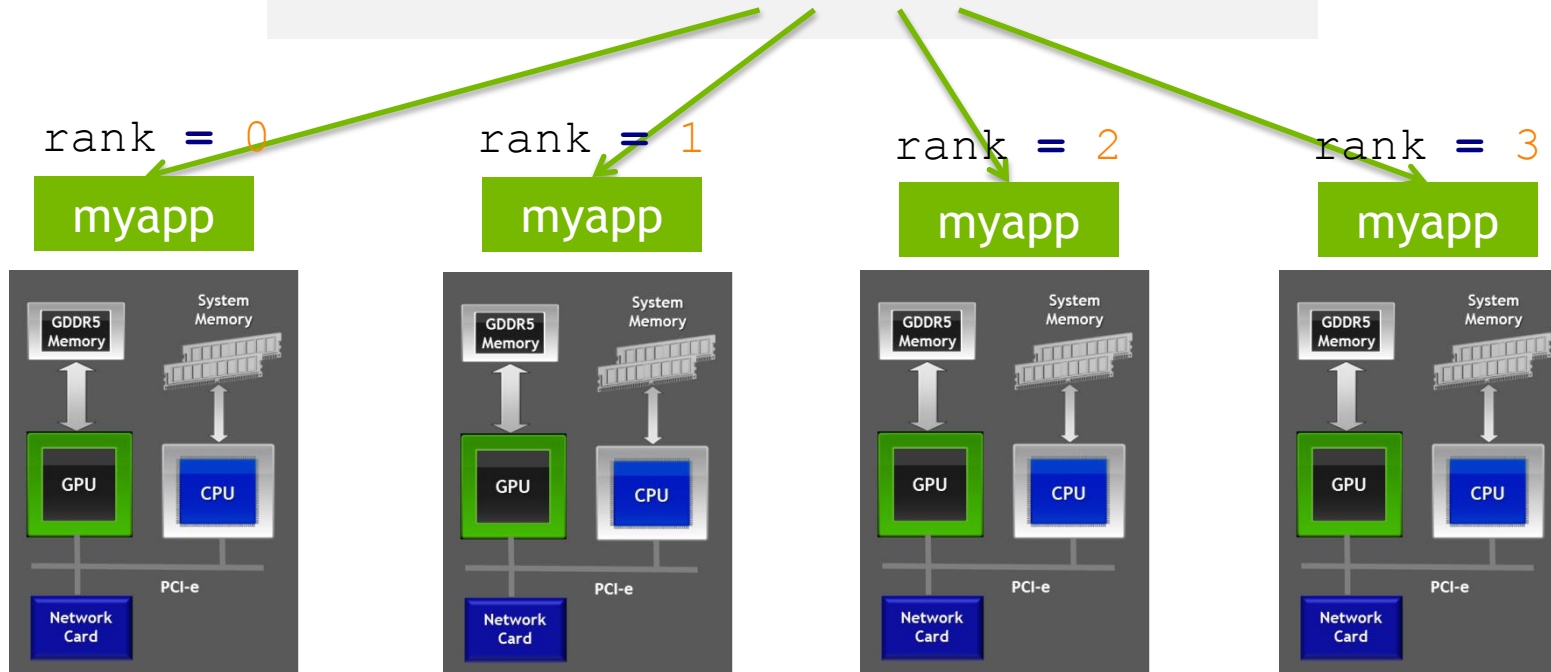
```
#include <mpi.h>

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>
```



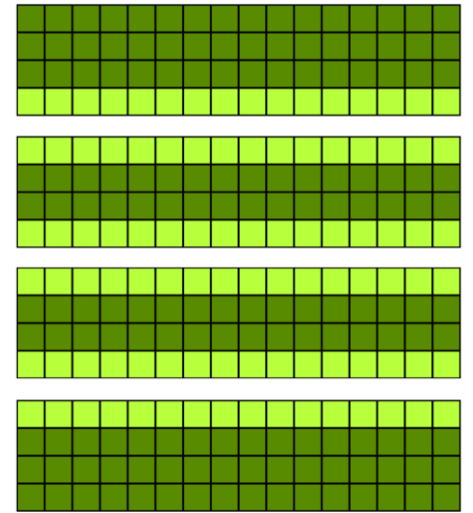
# EXAMPLE: JACOBI SOLVER

Solves the 2D-Poisson Equation on a rectangle

$$\Delta u(x, y) = e^{-10*(x^2+y^2)} \quad \forall (x, y) \in \Omega \setminus \delta\Omega$$

Periodic boundary conditions

Domain decomposition with stripes



Horizontal Stripes

# EXAMPLE: JACOBI SOLVER

## Single GPU

While not converged

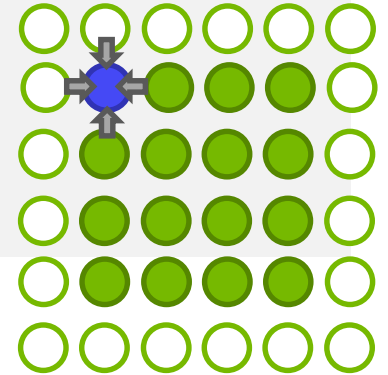
Do Jacobi step:

```
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

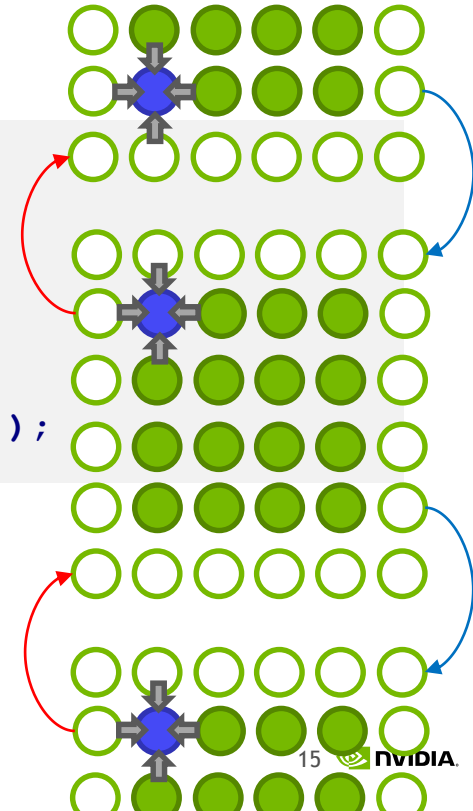
Do Jacobi step:

```
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



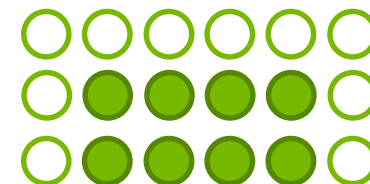
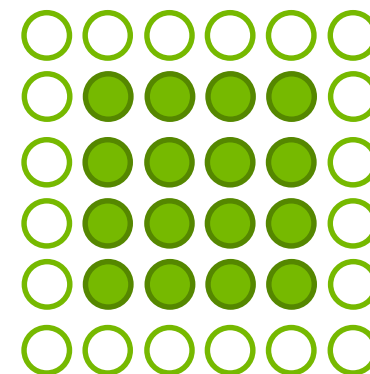
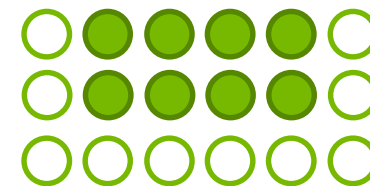


# EXAMPLE JACOBI

## Top/Bottom Halo

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

```
}
```



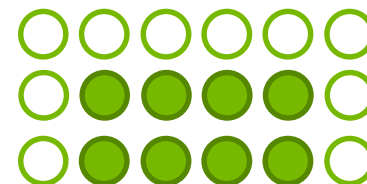
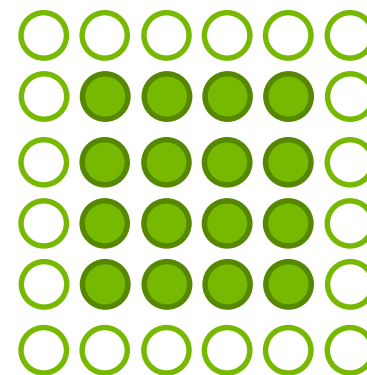
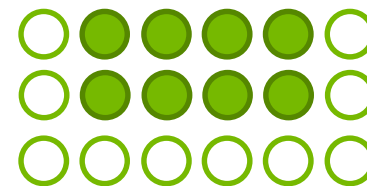
# EXAMPLE JACOBI

## 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);

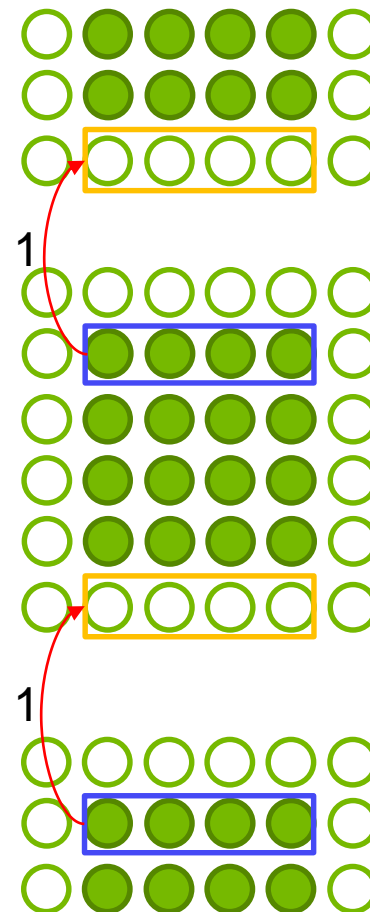
}
```



# 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);  
  
}
```

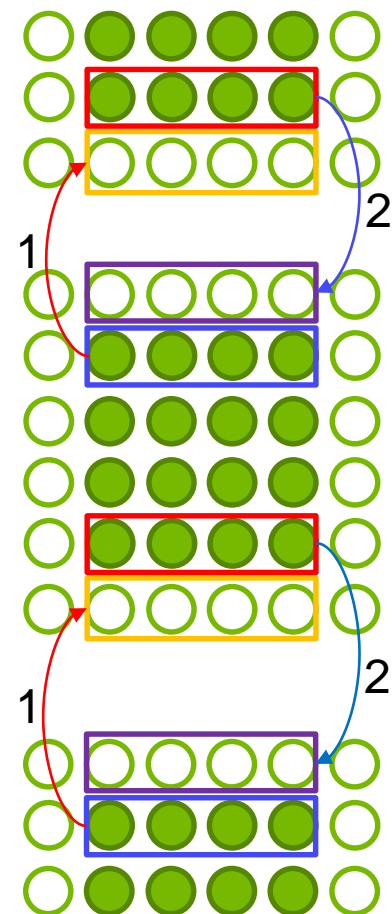


# EXAMPLE JACOBI

## 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

```
jkraus@ivb114:~/workspace/qwiklabs/Multi-GPU-MPI/task3
[jkraus@ivb114 task3]$ mpirun -np 2 cuda-memcheck --log-file laplace2d.%q{OMPI COMM WORLD RANK}.log --save laplace2d.%q{OMPI COMM WORLD RANK}.memcheck ./laplace2d
Jacobi relaxation Calculation: 2048 x 2048 mesh
Calculate reference solution and time serial execution.
call to cuMemcpyDtoHAsync returned error 719: Launch failed (often invalid pointer dereference)
call to cuMemcpyDtoHAsync returned error 719: Launch failed (often invalid pointer dereference)
-----
Primary job terminated normally, but 1 process returned
a non-zero exit code.. Per user-direction, the job has been aborted.
-----
mpirun detected that one or more processes exited with non-zero status, thus causing
the job to be terminated. The first process to do so was:

    Process name: [[42894,1],0]
    Exit code:      1
-----
[jkraus@ivb114 task3]$ ls laplace2d.*.log laplace2d.*.memcheck
laplace2d.0.log laplace2d.0.memcheck laplace2d.1.log laplace2d.1.memcheck
[jkraus@ivb114 task3]$
```

# MEMORY CHECKING WITH CUDA-MEMCHECK

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

```
jkraus@ivb114:~/workspace/qwiklabs/Multi-GPU-MPI/task3
=====
Saved host backtrace up to driver entry point at kernel launch time
=====
Host Frame:/usr/lib64/libcuda.so.1 (cuLaunchKernel + 0x2cd) [0x150bbd]
=====
Host Frame:/shared/apps/pgi/centos-6.2/linux86-64/15.1/lib/libaccn.so (__pgi_uacc_
cuda_launch + 0x1796) [0x10896]
=====
Host Frame:/shared/apps/pgi/centos-6.2/linux86-64/15.1/lib/libaccg.so (__pgi_uacc_
launch + 0x1a5) [0x10ed5]
=====
Host Frame:./laplace2d [0x26fd]
=====
Invalid __global__ write of size 4
=====
at 0x00000778 in /home-2/jkraus/workspace/qwiklabs/Multi-GPU-MPI/task3/./laplace2d
_serial.h:35:laplace2d_serial_32_gpu
=====
by thread (33,0,0) in block (8,6,0)
=====
Address 0x24edd2f088 is out of bounds
=====
Saved host backtrace up to driver entry point at kernel launch time
=====
Host Frame:/usr/lib64/libcuda.so.1 (cuLaunchKernel + 0x2cd) [0x150bbd]
=====
Host Frame:/shared/apps/pgi/centos-6.2/linux86-64/15.1/lib/libaccn.so (__pgi_uacc_
cuda_launch + 0x1796) [0x10896]
=====
Host Frame:/shared/apps/pgi/centos-6.2/linux86-64/15.1/lib/libaccg.so (__pgi_uacc_
launch + 0x1a5) [0x10ed5]
=====
Host Frame:./laplace2d [0x26fd]
=====
Invalid __global__ write of size 4
=====
at 0x00000778 in /home-2/jkraus/workspace/qwiklabs/Multi-GPU-MPI/task3/./laplace2d
_serial.h:35:laplace2d_serial_32_gpu
```

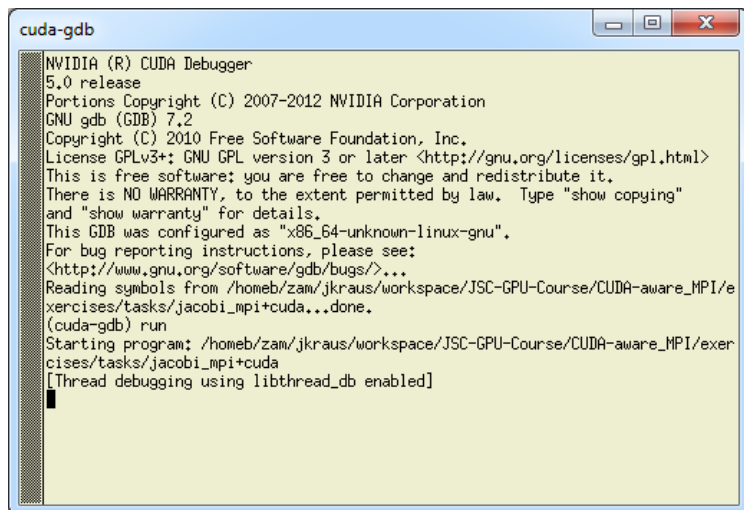
# DEBUGGING MPI+OPENACC APPLICATIONS

Using `cuda-gdb` with MPI Applications

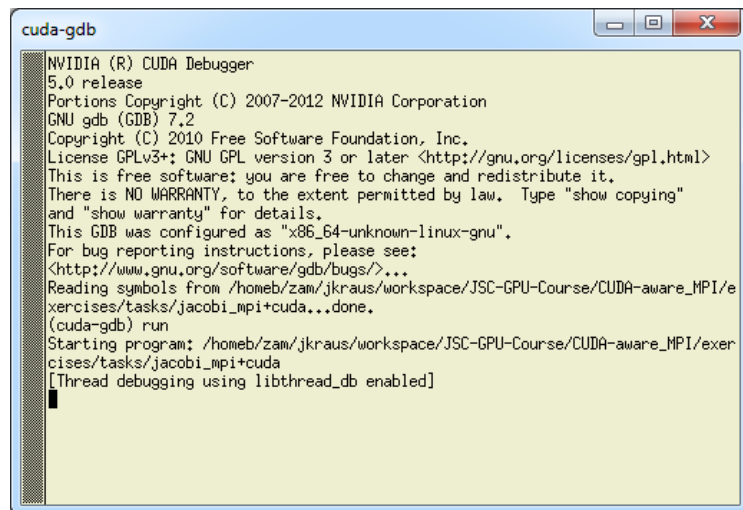
Use `cuda-gdb` just like `gdb`

For smaller applications, just launch `xterms` and `cuda-gdb`

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



```
cuda-gdb
NVIDIA (R) CUDA Debugger
5.0 release
Portions Copyright (C) 2007-2012 NVIDIA Corporation
GNU gdb (GDB) 7.2
Copyright (C) 2010 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/>...
Reading symbols from /homeb/zam/jkraus/workspace/JSC-GPU-Course/CUDA-aware_MPI/e
xercises/tasks/jacobi_mpi+cuda...done.
(cuda-gdb) run
Starting program: /homeb/zam/jkraus/workspace/JSC-GPU-Course/CUDA-aware_MPI/exe
rcises/tasks/jacobi_mpi+cuda
[Thread debugging using libthread_db enabled]
█
```



```
cuda-gdb
NVIDIA (R) CUDA Debugger
5.0 release
Portions Copyright (C) 2007-2012 NVIDIA Corporation
GNU gdb (GDB) 7.2
Copyright (C) 2010 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/>...
Reading symbols from /homeb/zam/jkraus/workspace/JSC-GPU-Course/CUDA-aware_MPI/e
xercises/tasks/jacobi_mpi+cuda...done.
(cuda-gdb) run
Starting program: /homeb/zam/jkraus/workspace/JSC-GPU-Course/CUDA-aware_MPI/exe
rcises/tasks/jacobi_mpi+cuda
[Thread debugging using libthread_db enabled]
█
```

# 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

```
jkraus@sb077:~/workspace/Jacobi/main/bin
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_MPI=1
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 can now attach a debugger to the application (PID 28250) for sb077: The application encountered a device error and CUDA_DEBUGGER can now attach a debugger to the application (PID 28252) for sb077: The application encountered a device error and CUDA_DEBUGGER can now attach a debugger to the application (PID 28251) for sb077: The application encountered a device error and CUDA_DEBUGGER can now attach a debugger to the application (PID 28249) for

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]
0x00000000018e1ce8 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 0x00000000018e1ce8 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+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

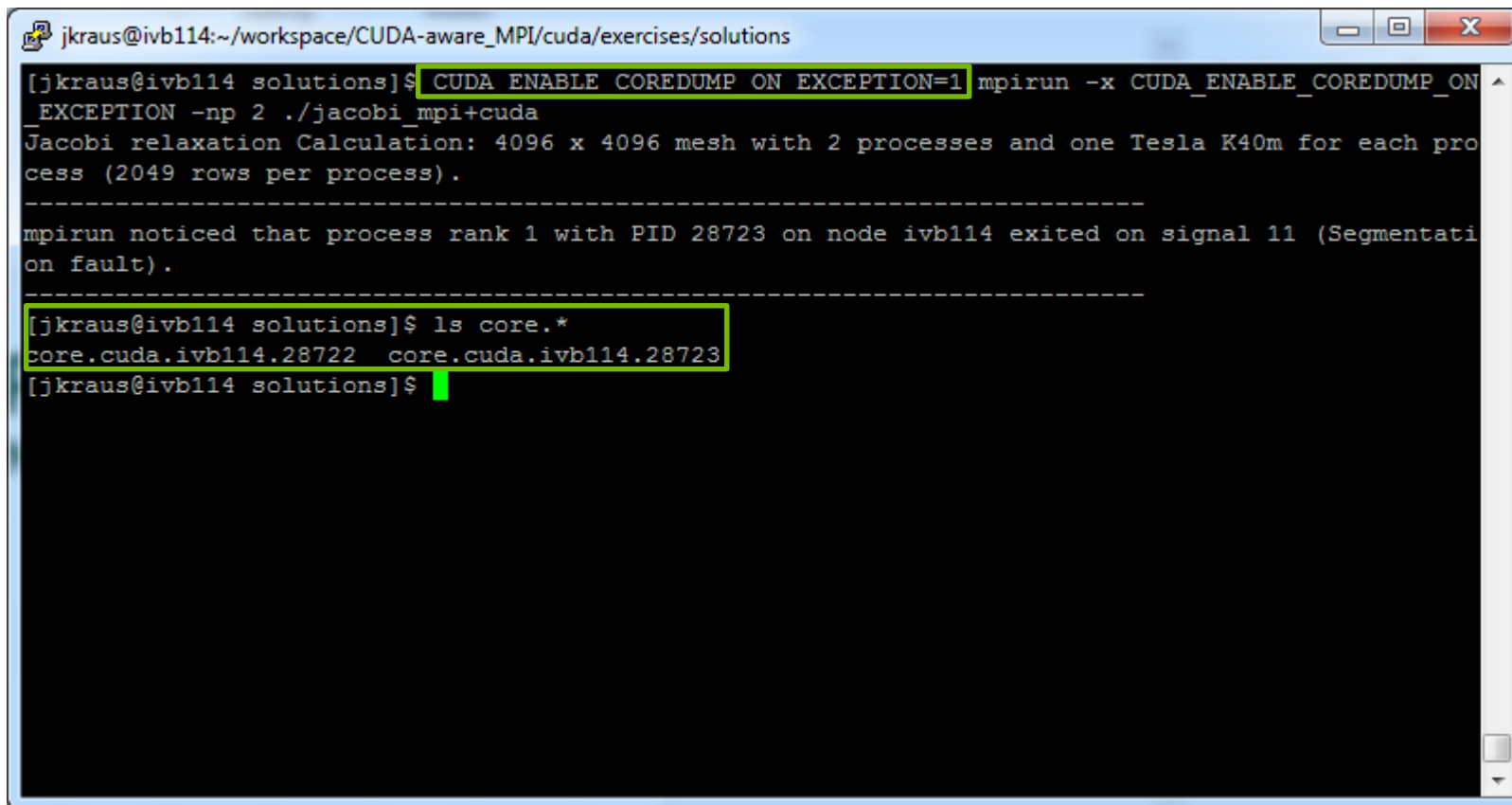
Open GPU: `(cuda-gdb) target cudacore core.cuda`

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



# DEBUGGING MPI+OPENACC APPLICATIONS

CUDA\_ENABLE\_COREDUMP\_ON\_EXCEPTION



```
jkraus@ivb114:~/workspace/CUDA-aware_MPI/cuda/exercises/solutions
[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]$
```

# DEBUGGING MPI+OPENACC APPLICATIONS

CUDA\_ENABLE\_COREDUMP\_ON\_EXCEPTION

```
jkraus@ivb114:~/workspace/CUDA-aware_MPI/cuda/exercises/solutions
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) █
```

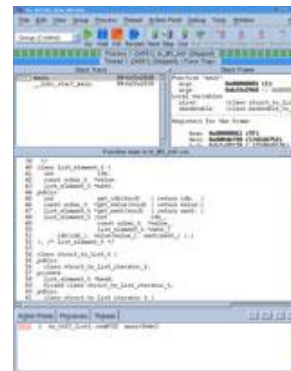
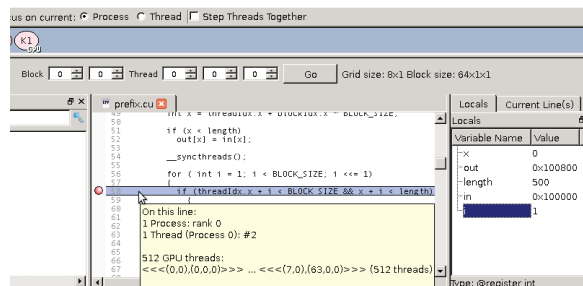
# DEBUGGING MPI+OPENACC APPLICATIONS

## Third Party Tools

Allinea DDT debugger

Rogue Wave TotalView

Stacks		
Threads	CUDA Threads	Function
1	0	main (prefix.cu:193)
1	0	cudasummer (prefix.cu:143)
1	0	prefixsum (prefix.cu:105)
1	512	zarro (prefix.cu:89)
1	480	zarro (prefix.cu:90)



# 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} \
```

New with  
CUDA 7.5

```
--process-name "rank %q{OMPI_COMM_WORLD_RANK}" \
```

```
--context-name "rank %q{OMPI_COMM_WORLD_RANK}"
```

OpenMPI: OMPI\_COMM\_WORLD\_RANK

MPICH2: 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

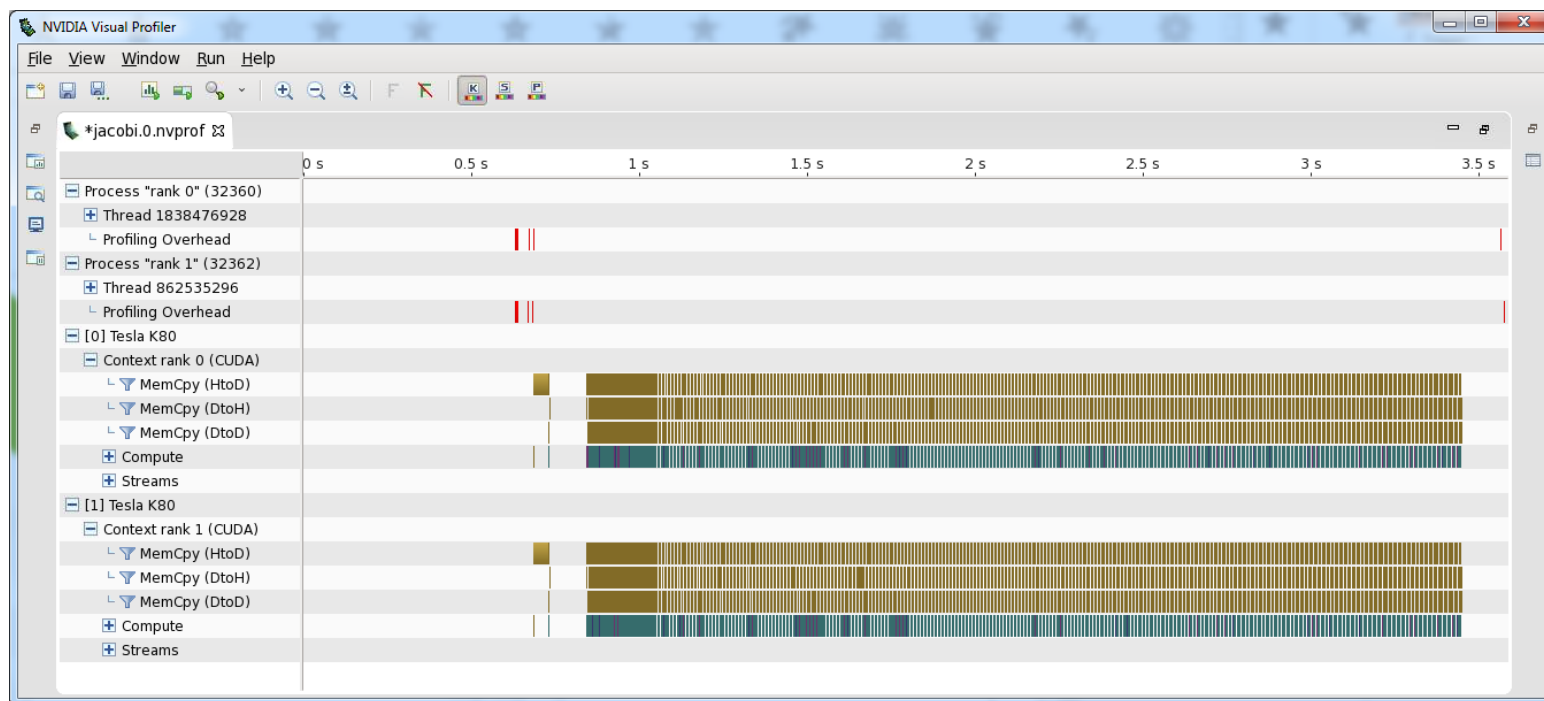
```
jkraus@ivb120:~/workspace/code-samples/posts/cuda-aware-mpi-example/bin
[jkraus@ivb120 bin]$ mpirun -np 2 nvprof --output-profile jacobi ${OMPI_COMM_WORLD_RANK} nvprof
--process-name "rank ${OMPI_COMM_WORLD_RANK}" /jacobi_cuda_aware_mpi -t 2 -fs
==25442== NVPROF is profiling process 2
==25441== NVPROF is profiling process 2
Topology size: 2 x 1
Local domain size (current node): 4096 x 4096
Global domain size (all nodes): 8192 x 4096
Starting Jacobi run with 2 processes using "Tesla K80" GPUs (ECC enabled: 2 / 2):
Iteration: 0 - Residue: 0.250000
Iteration: 100 - Residue: 0.002397
Iteration: 200 - Residue: 0.001204
Iteration: 300 - Residue: 0.000804
Iteration: 400 - Residue: 0.000603
Iteration: 500 - Residue: 0.000483
Iteration: 600 - Residue: 0.000402
Iteration: 700 - Residue: 0.000345
Stopped after 1000 iterations with residue 0.000242
Total Jacobi run time: 2.7167 sec.
Average per-process communication time: 0.1858 sec.
Measured lattice updates: 12.34 GLU/s (total), 6.17 GLU/s (per process)
Measured FLOPS: 61.71 GFLOPS (total), 30.85 GFLOPS (per process)
Measured device bandwidth: 592.11 GB/s (total), 296.21 GB/s (per process)
==32360== Generated result file: /home-2/jkraus/workspace/code-samples/posts/cuda-aware-mpi-example/bin/jacobi.0.nvprof
==32362== Generated result file: /home-2/jkraus/workspace/code-samples/posts/cuda-aware-mpi-example/bin/jacobi.1.nvprof
[jkraus@ivb120 bin]$
```

# PROFILING MPI+OPENACC APPLICATIONS

Using `nvprof`+NVVP

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

Or use the import Wizard



# PROFILING MPI+OPENACC APPLICATIONS

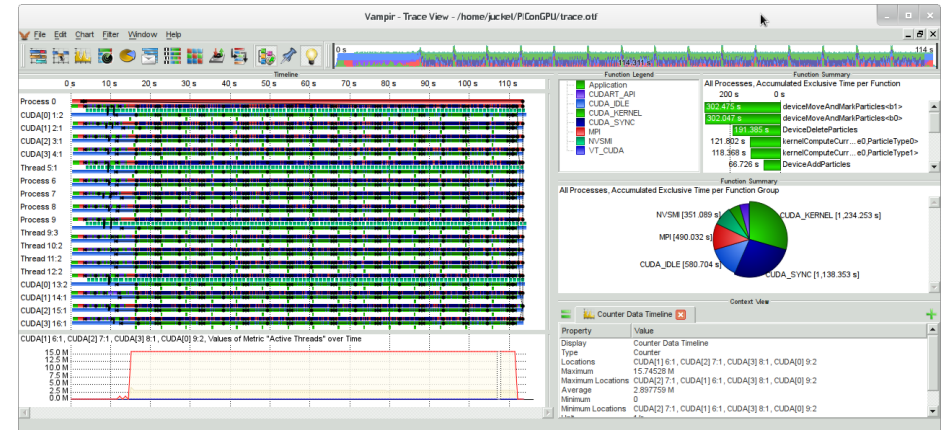
## Third Party Tools

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



N=1

- 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)

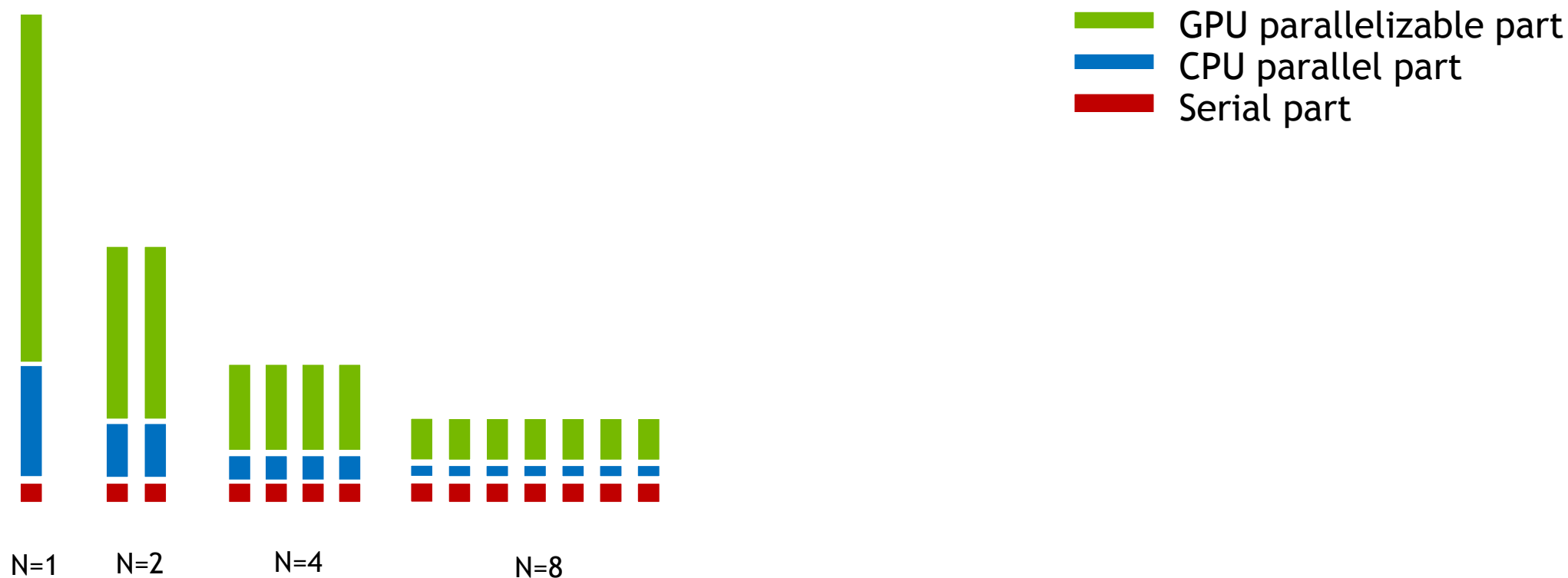
For Legacy MPI Applications



Multicore CPU only

# MULTI PROCESS SERVICE (MPS)

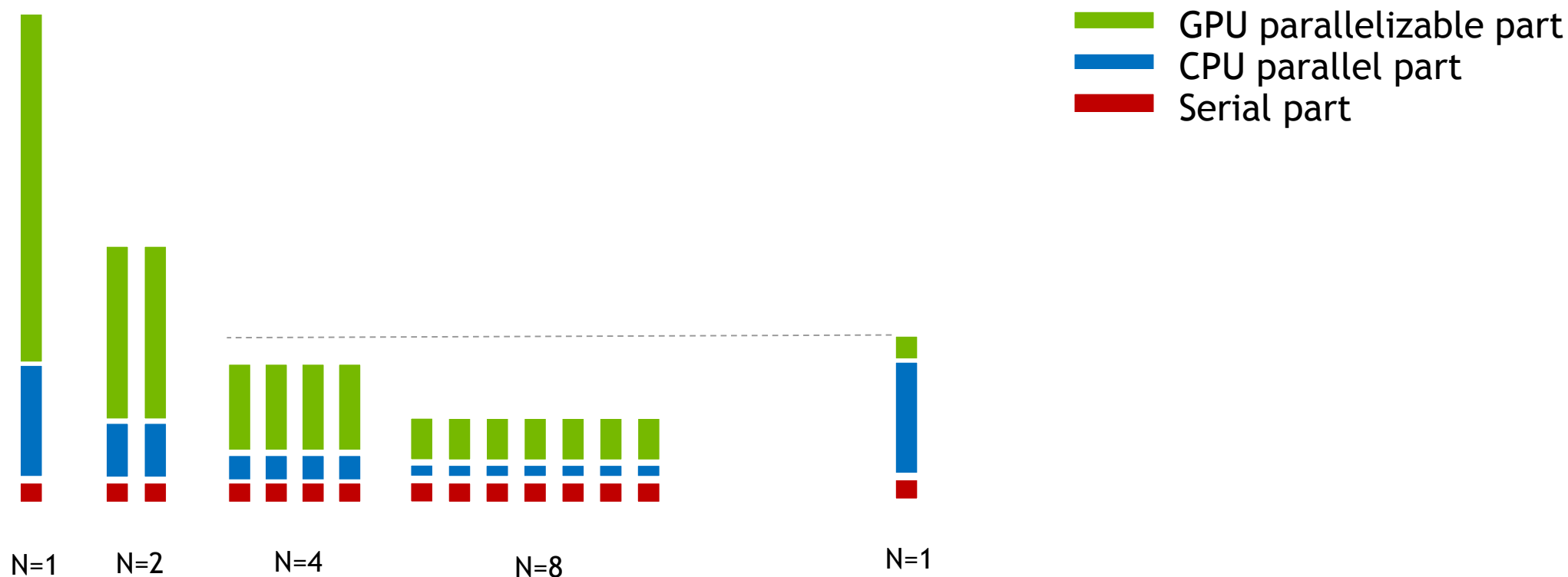
For Legacy MPI Applications



Multicore CPU only

# MULTI PROCESS SERVICE (MPS)

For Legacy MPI Applications

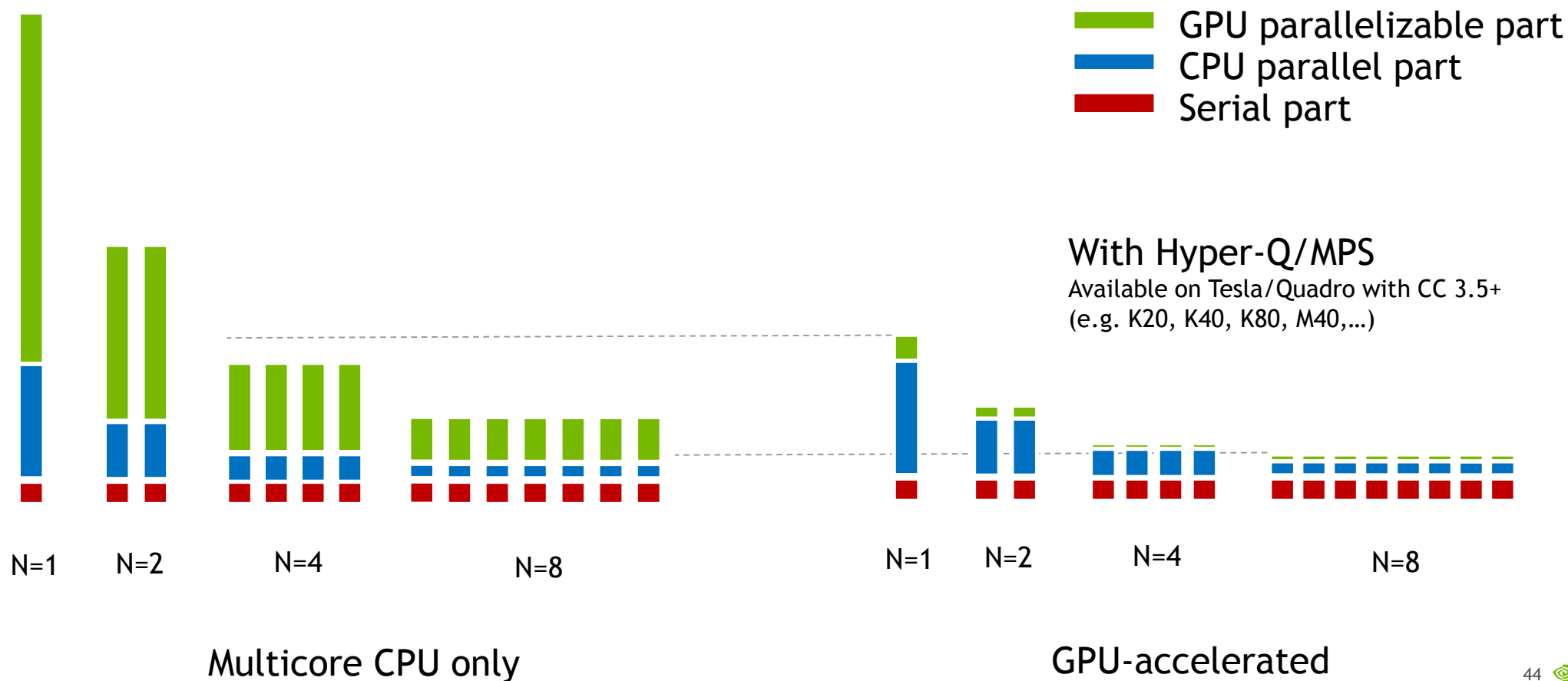


Multicore CPU only

GPU-accelerated

# MULTI PROCESS SERVICE (MPS)

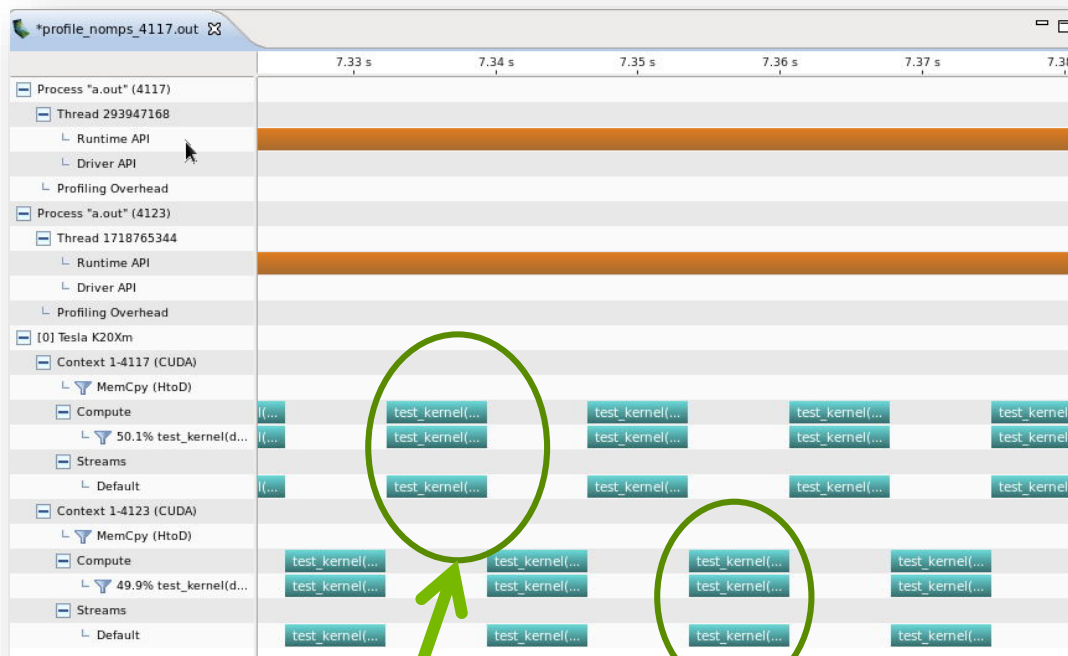
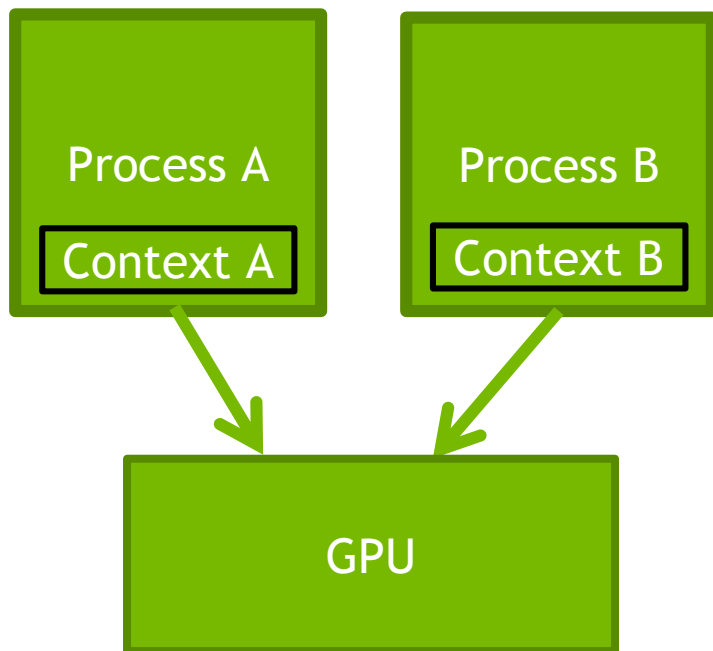
For Legacy MPI Applications





# PROCESSES SHARING GPU WITHOUT MPS

No Overlap

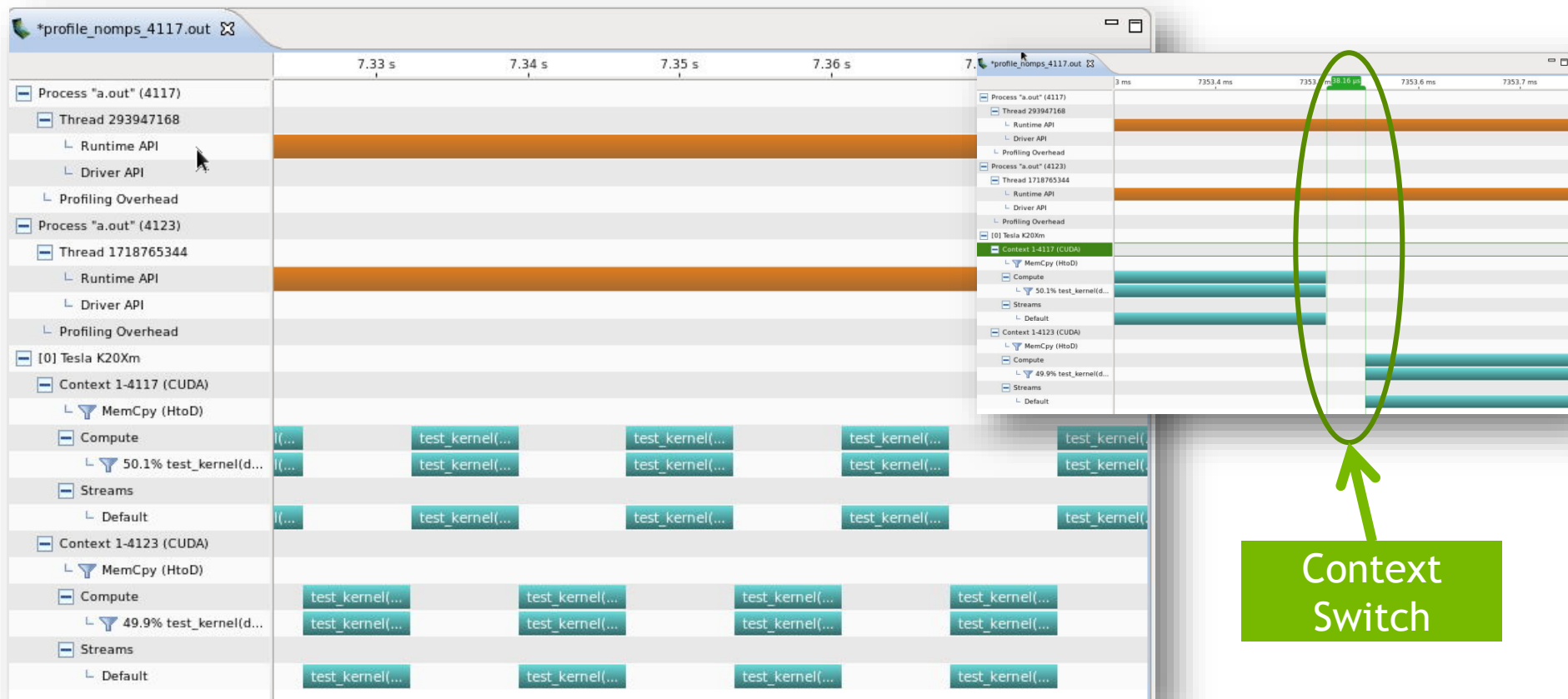


Process A

Process B

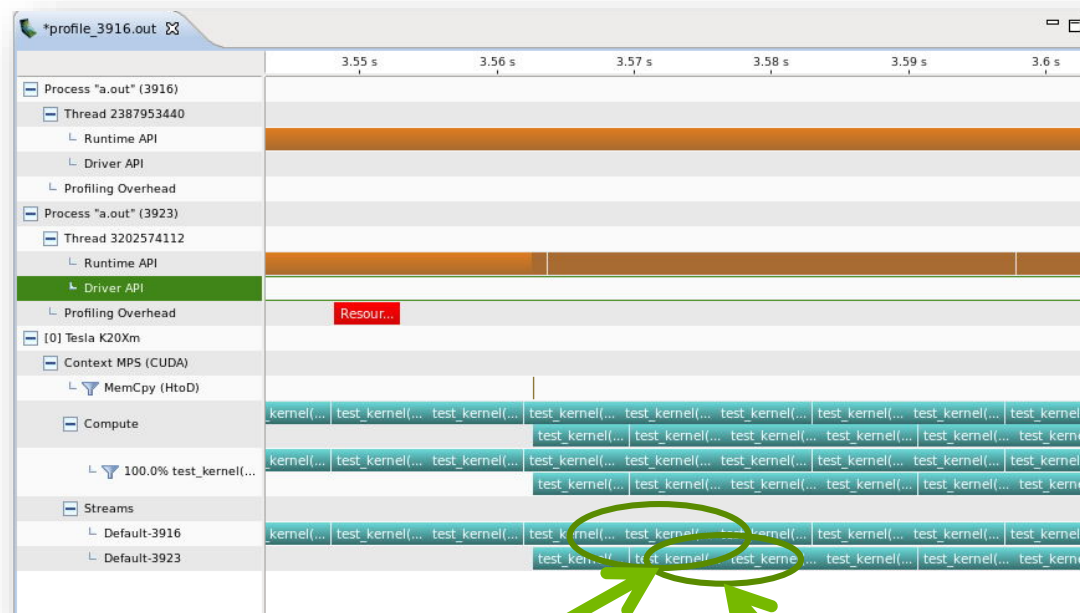
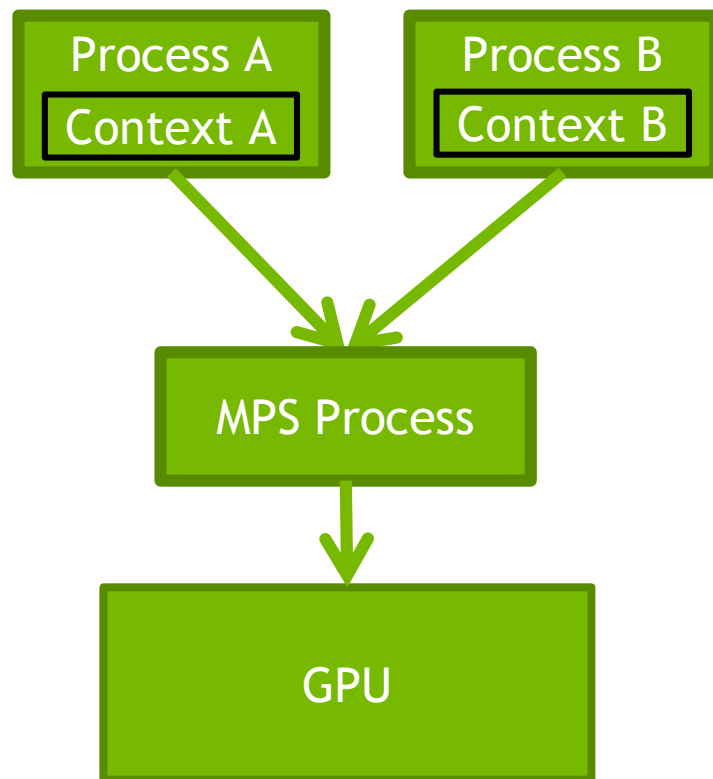
# PROCESSES SHARING GPU WITHOUT MPS

## Context Switch Overhead



# PROCESSES SHARING GPU WITH MPS

## Maximum Overlap

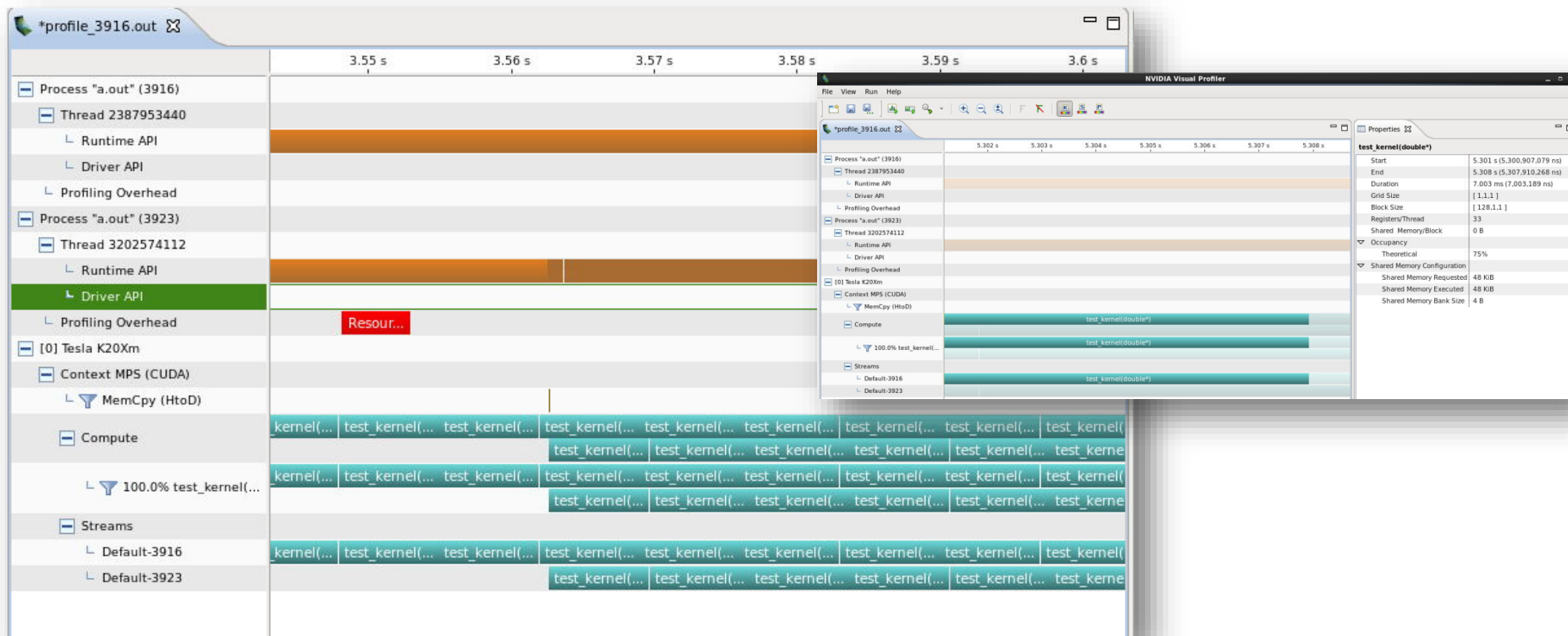


Kernels from  
Process A

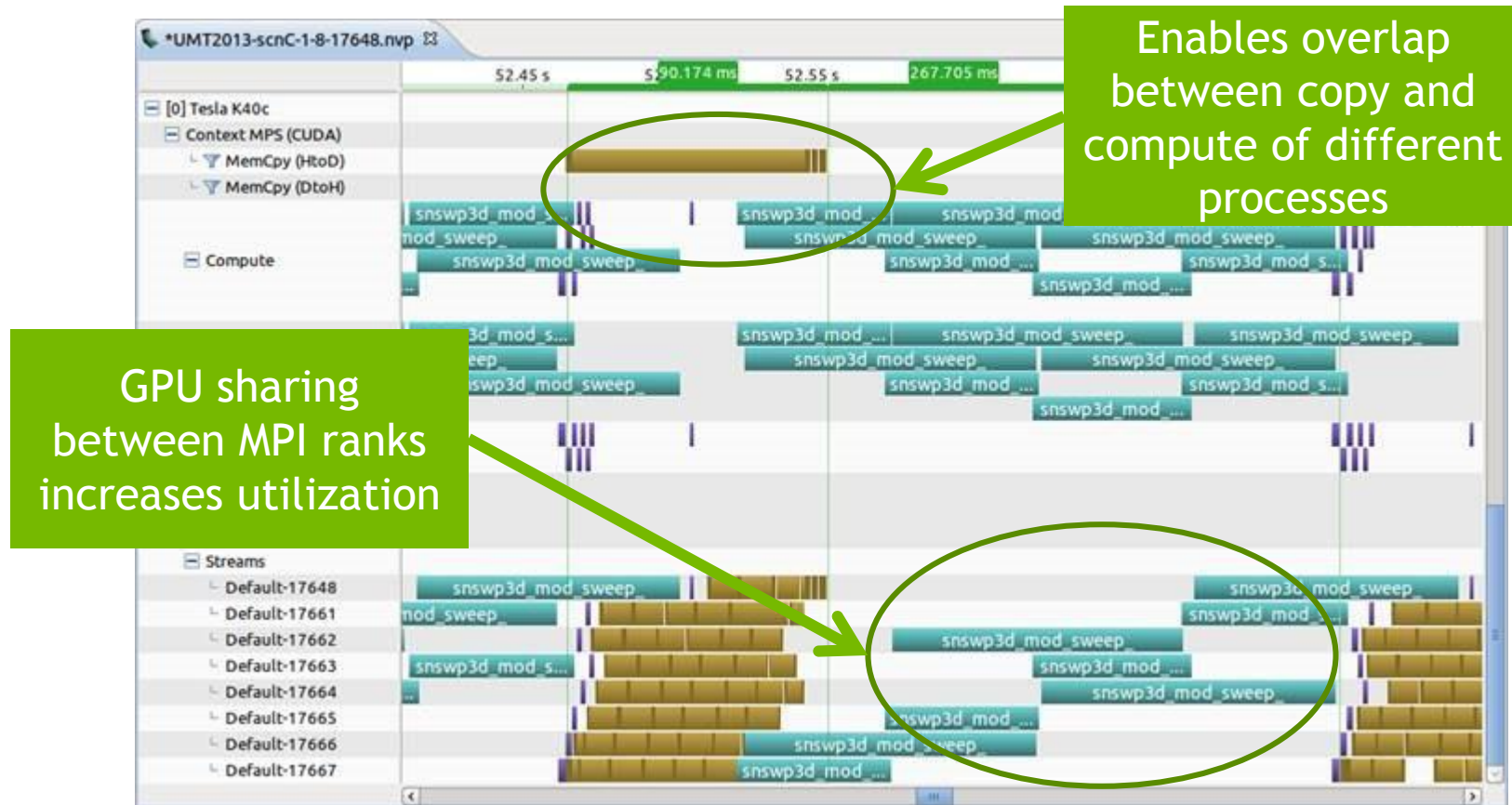
Kernels from  
Process B

# PROCESSES SHARING GPU WITH MPS

## No Context Switch Overhead

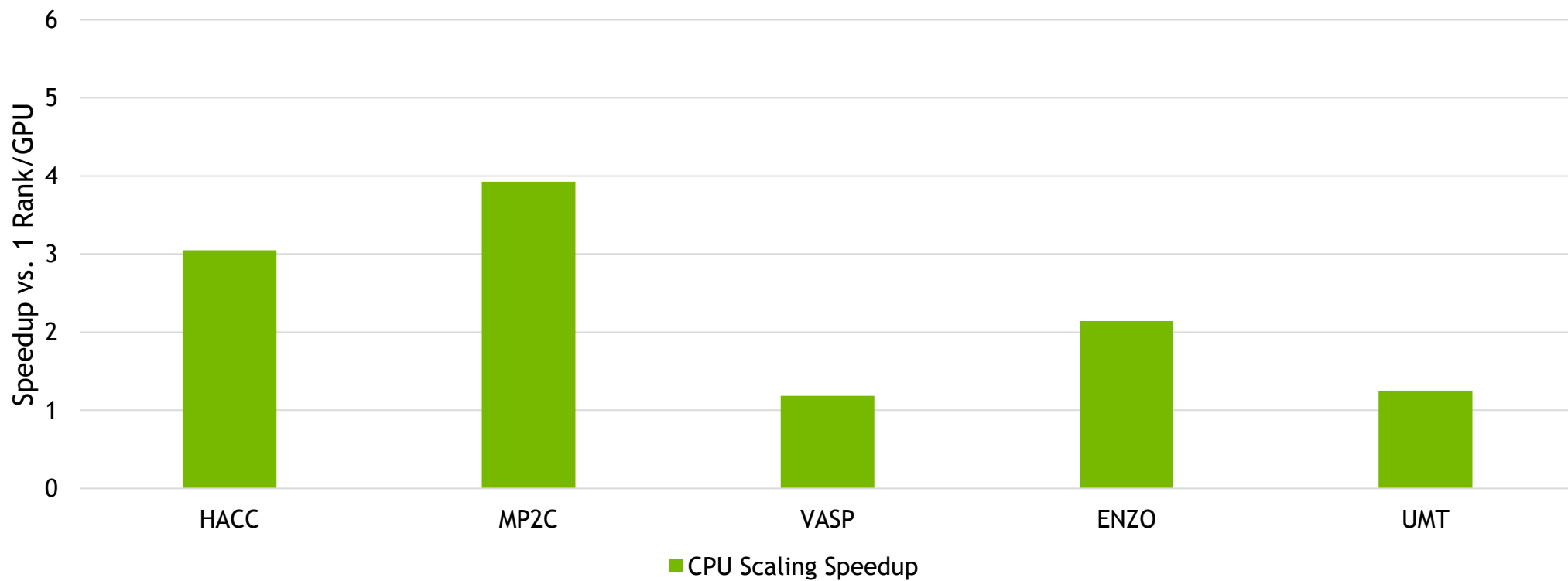


# HYPER-Q/MPS CASE STUDY: UMT



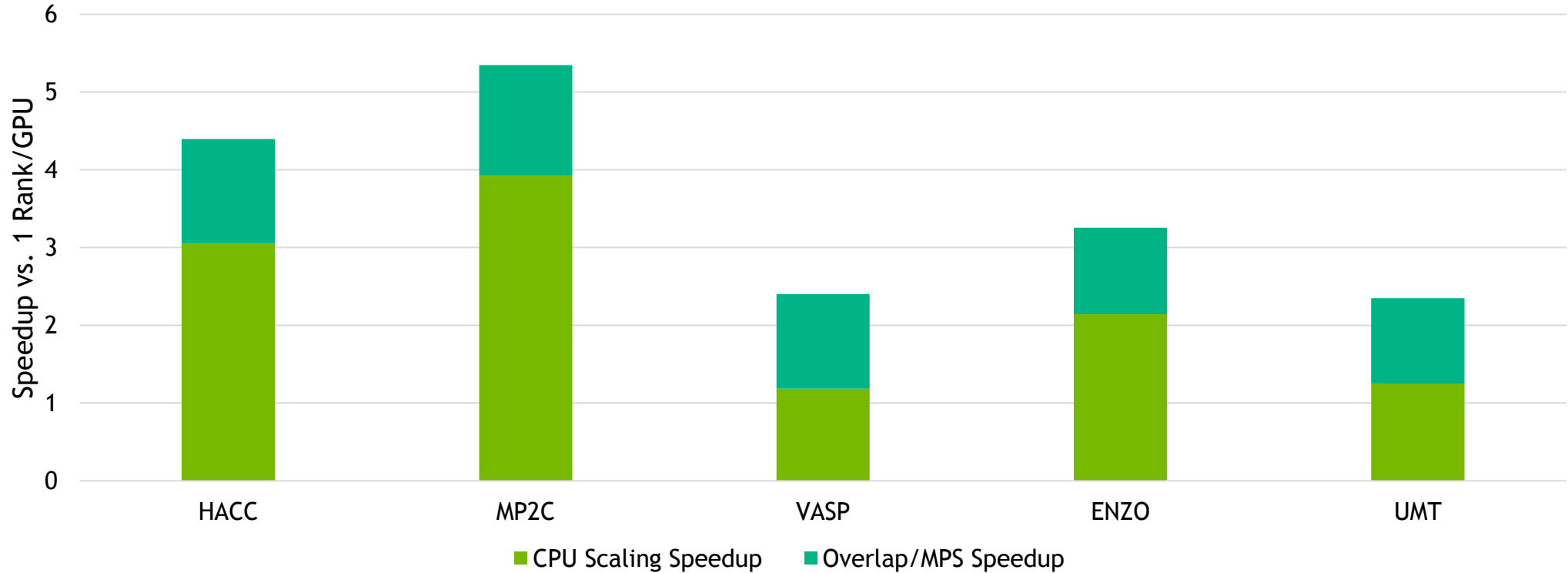
# HYPER-Q/MPS CASE STUDIES

## 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
```

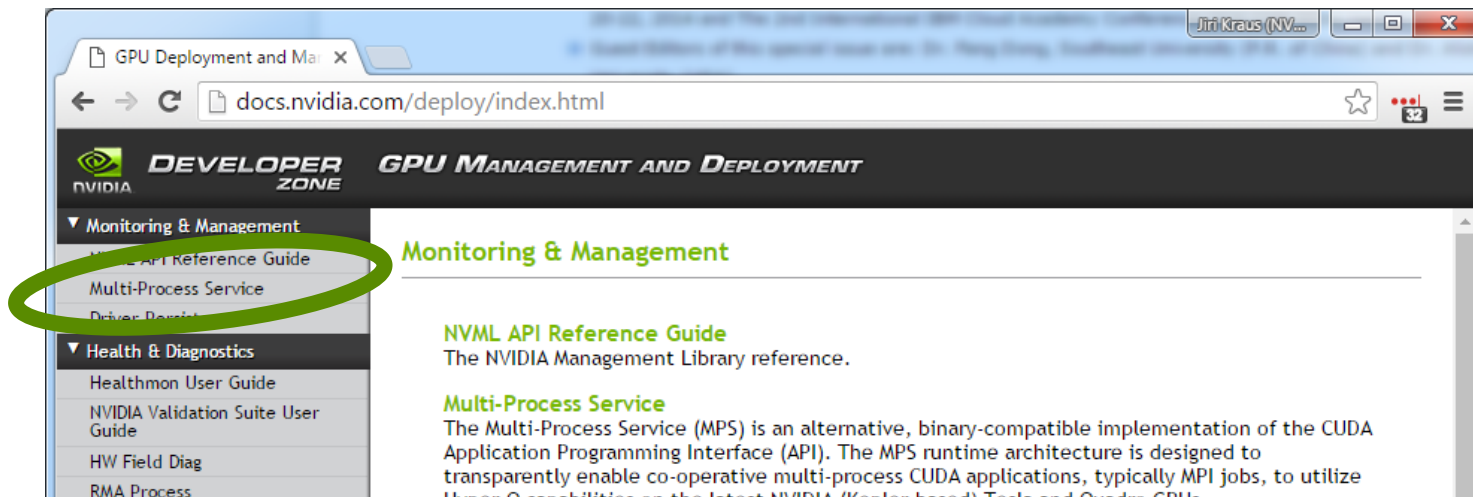


# MPS SUMMARY

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!

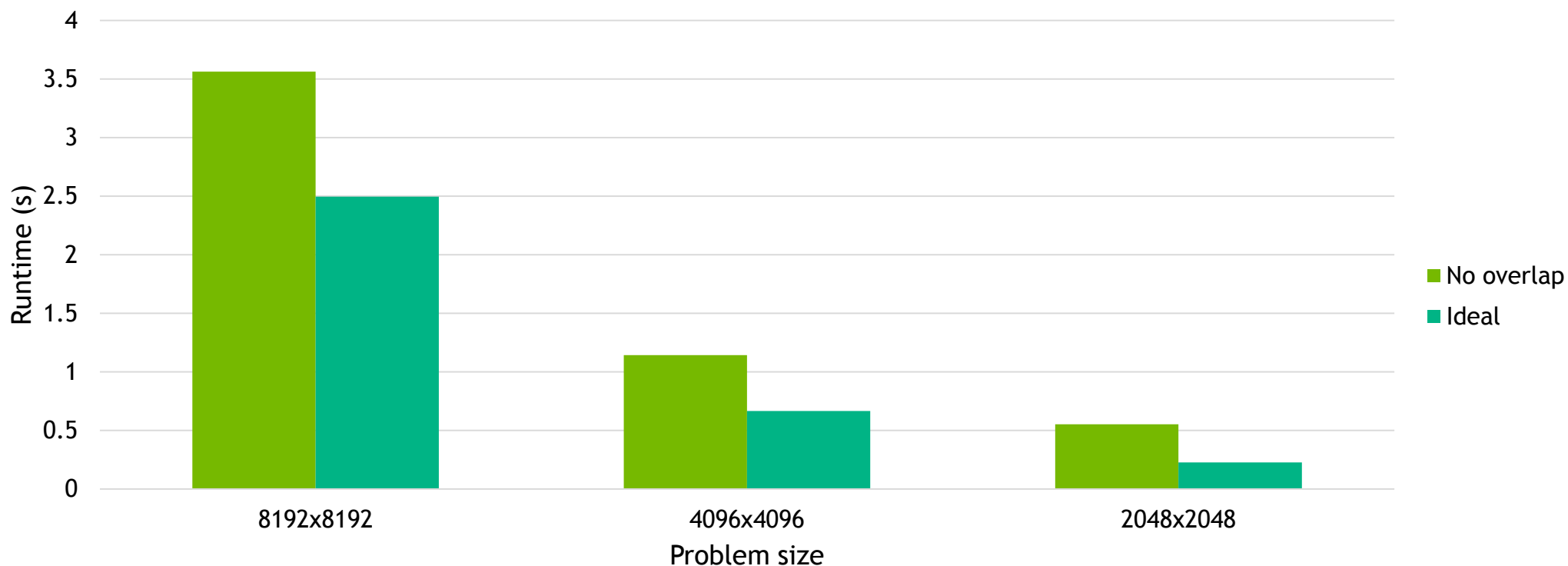




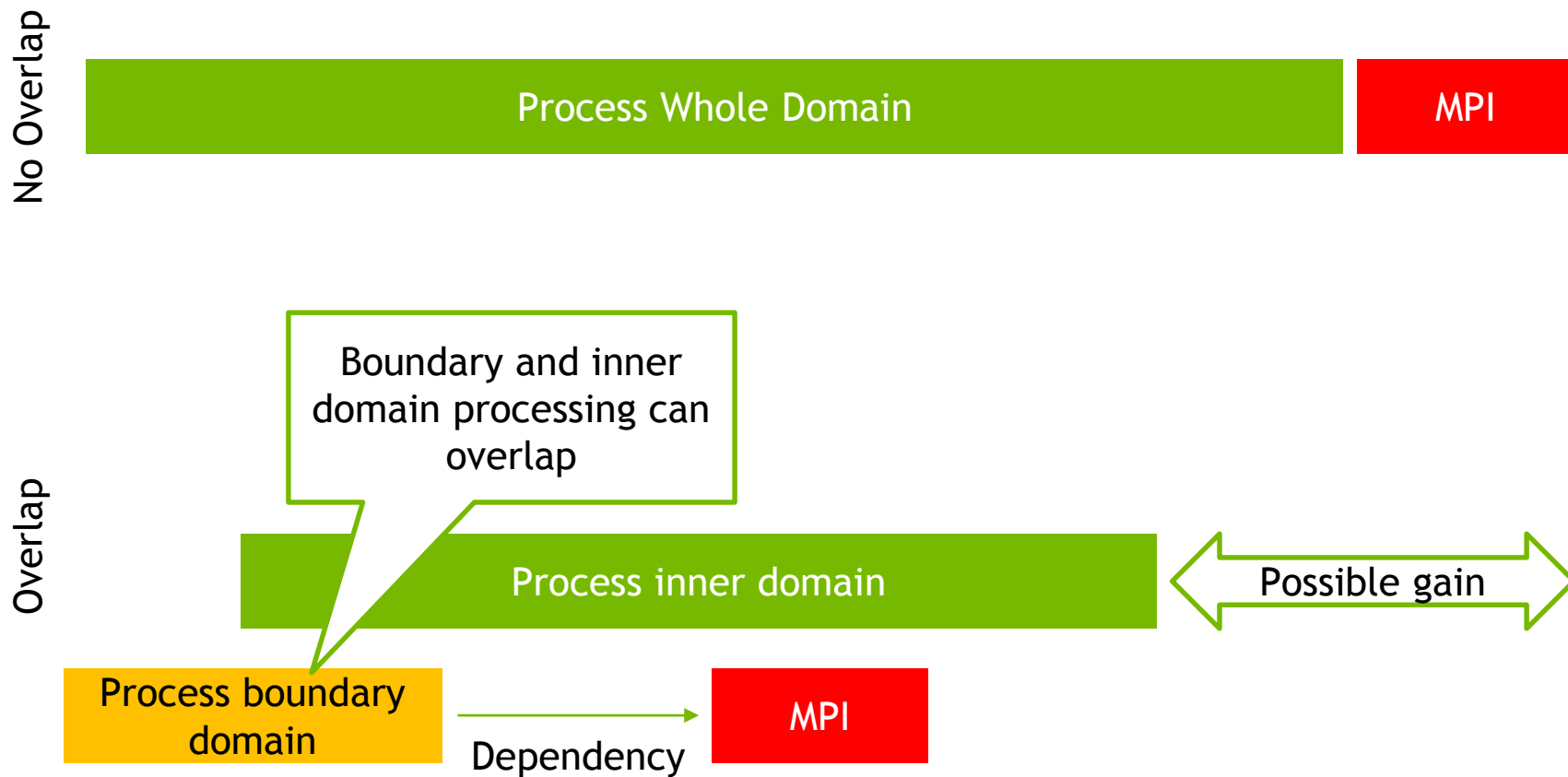
Decreasing parallel overhead

# COMMUNICATION + COMPUTATION OVERLAP

OpenMPI 1.10.2 - 4 Tesla K80



# COMMUNICATION + COMPUTATION OVERLAP



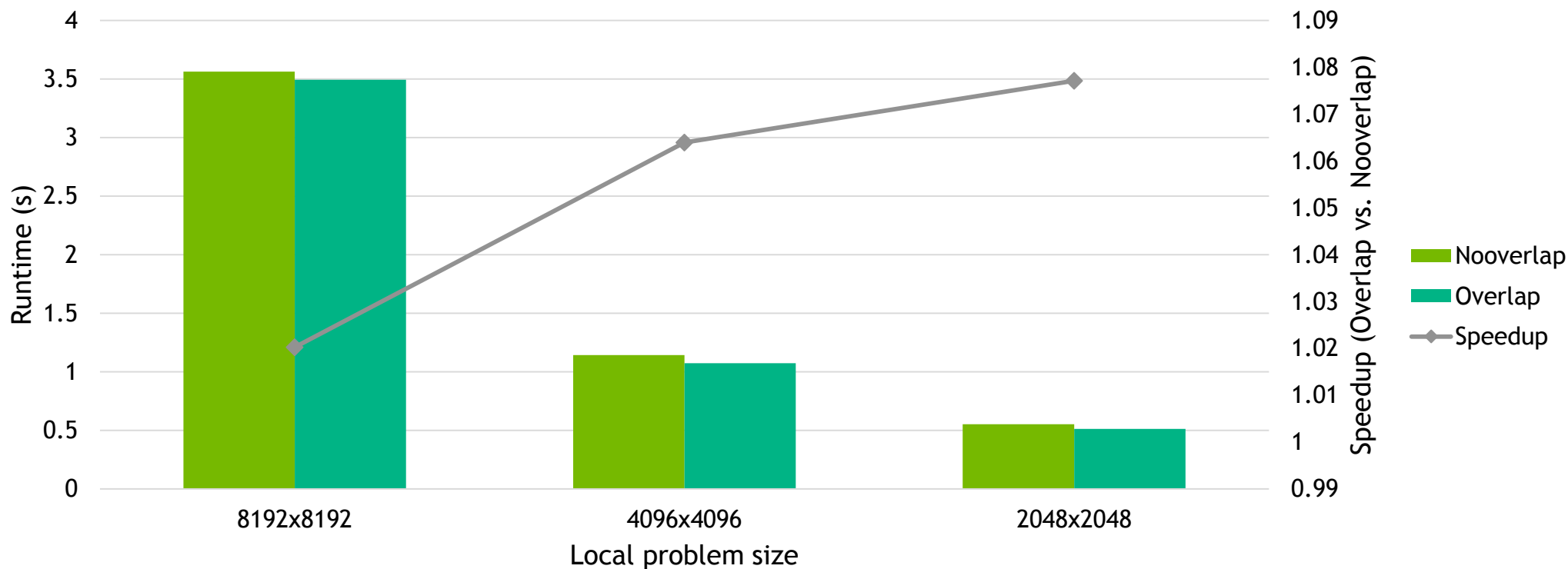
# COMMUNICATION + COMPUTATION OVERLAP

## 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



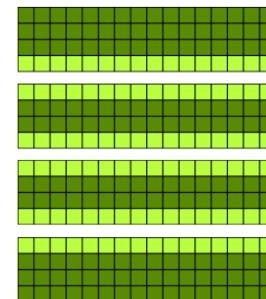
# DOMAIN DECOMPOSITION STRATEGIES

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

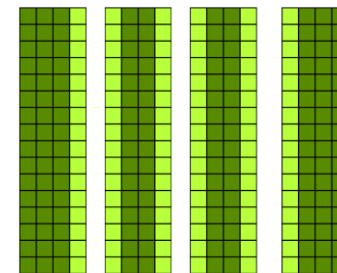
- Minimizes the number of neighbors

- Avoids noncontiguous halo exchange

- Good for latency bound problems



Horizontal Stripes



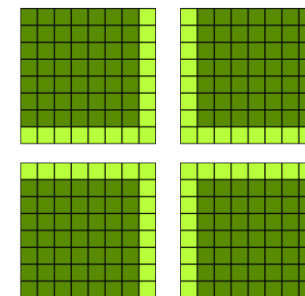
Vertical Stripes

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

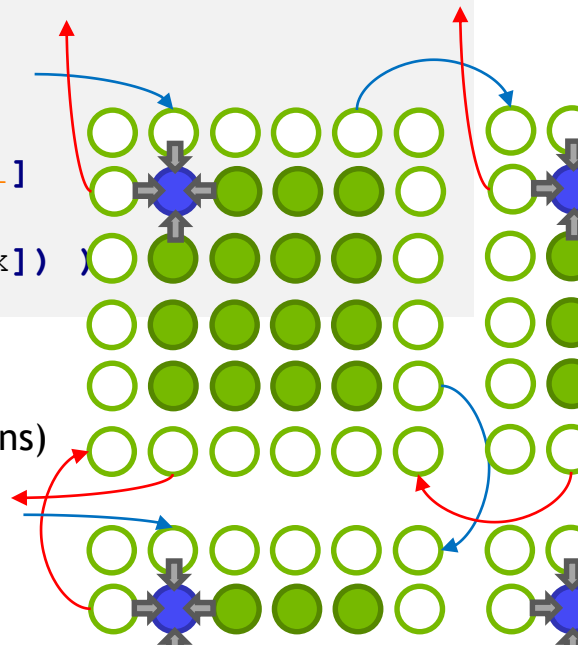
Do Jacobi step:

```
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)

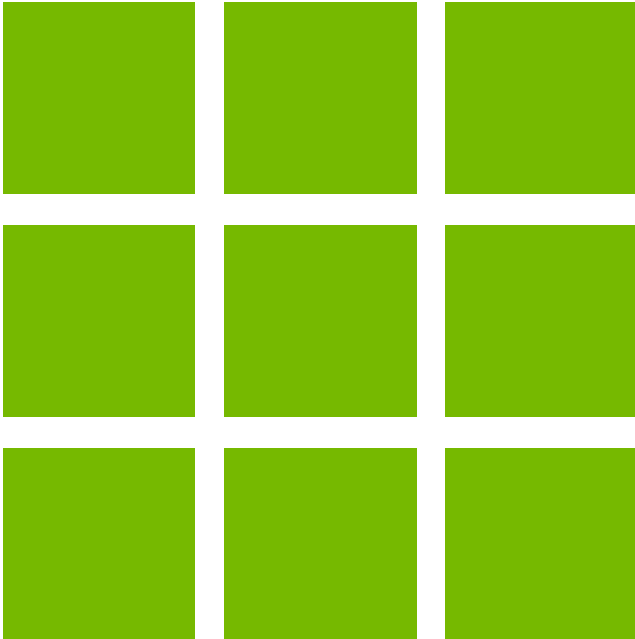
Next iteration





# DOMAIN DECOMPOSITION WITH TILES

Handling Tile  $\leftrightarrow$  GPU/MPI Rank affinity



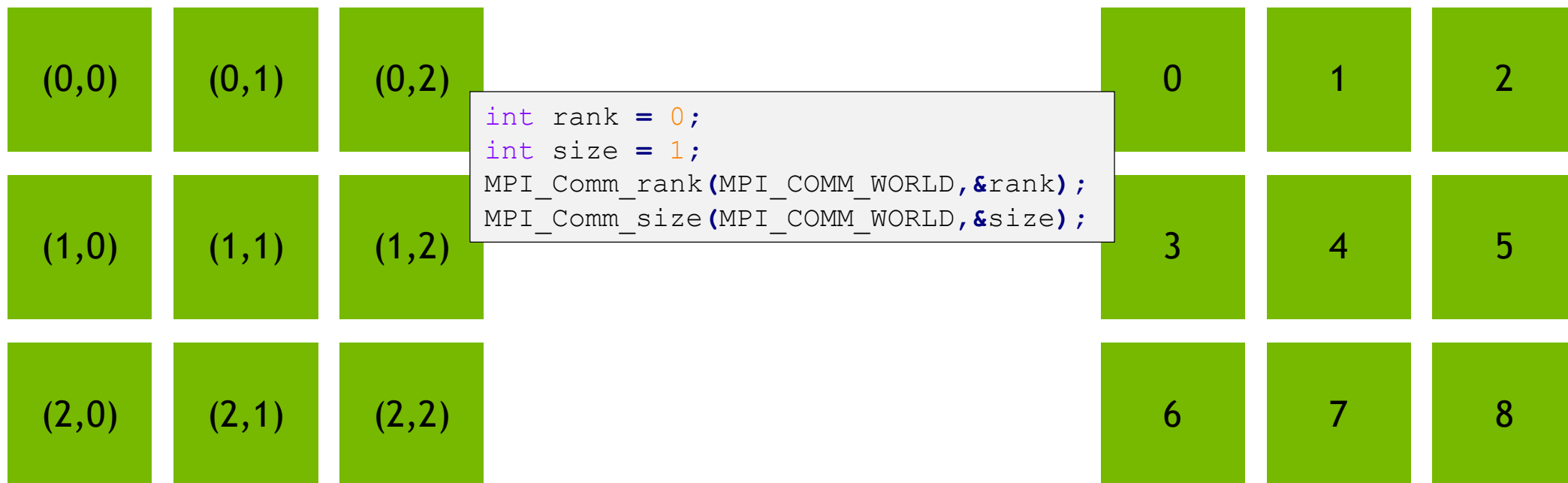
# DOMAIN DECOMPOSITION WITH TILES

Handling Tile  $\leftrightarrow$  GPU/MPI Rank affinity



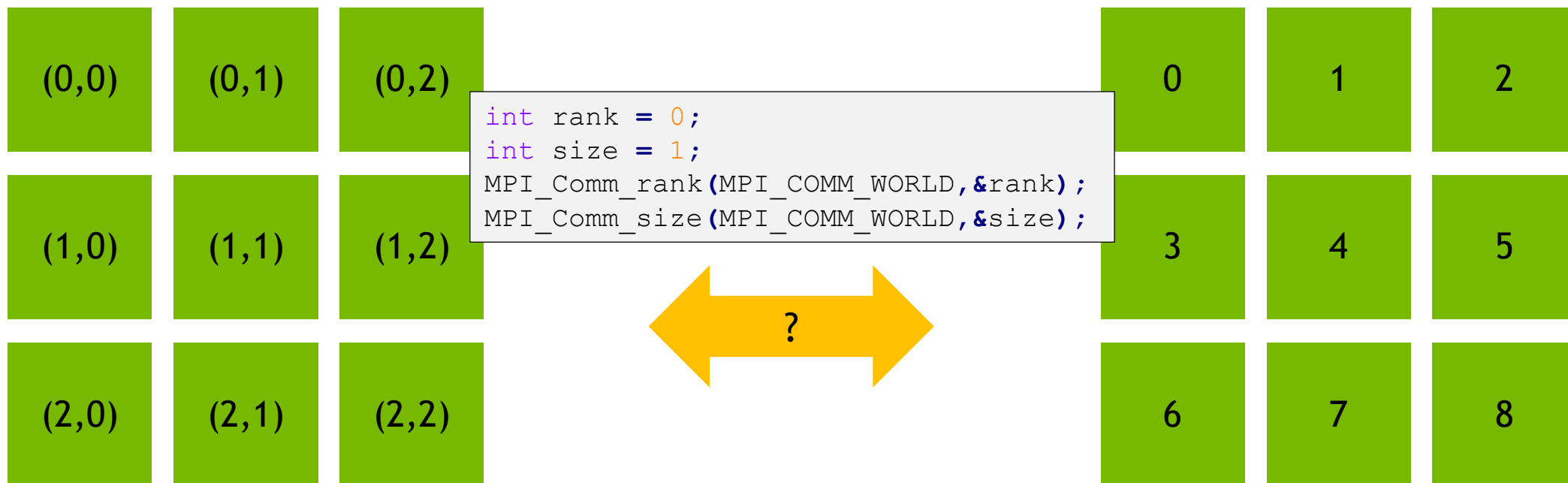
# DOMAIN DECOMPOSITION WITH TILES

Handling Tile  $\leftrightarrow$  GPU/MPI Rank affinity



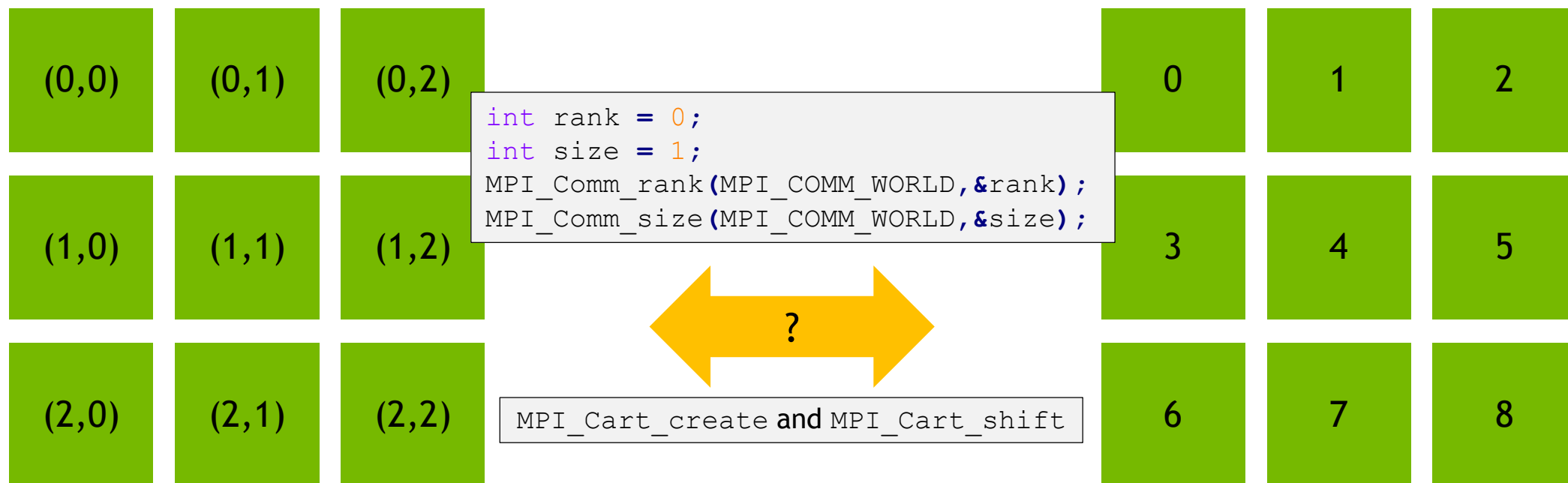
# DOMAIN DECOMPOSITION WITH TILES

Handling Tile  $\leftrightarrow$  GPU/MPI Rank affinity



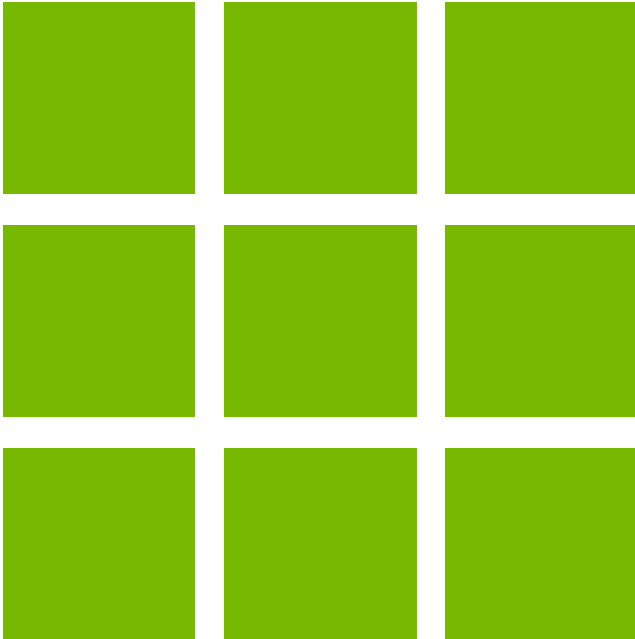
# DOMAIN DECOMPOSITION WITH TILES

Handling Tile  $\leftrightarrow$  GPU/MPI Rank affinity



# DOMAIN DECOMPOSITION WITH TILES

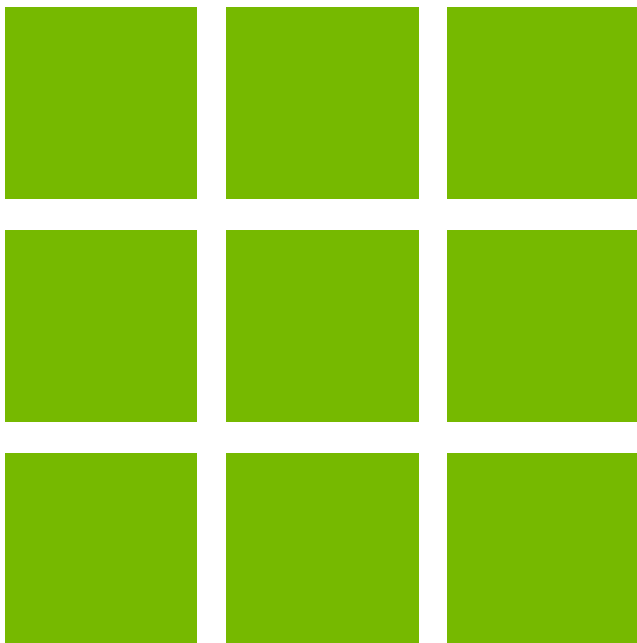
Handling Tile  $\leftrightarrow$  GPU/MPI Rank affinity



# DOMAIN DECOMPOSITION WITH TILES

Handling Tile  $\leftrightarrow$  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);
```

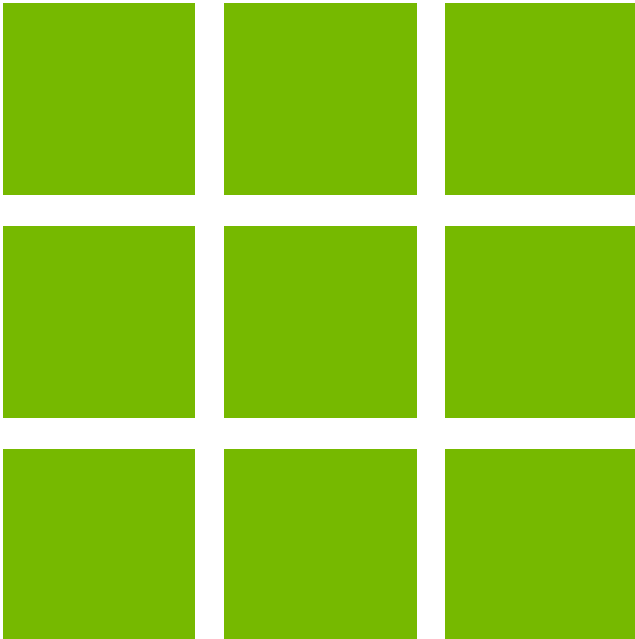
# DOMAIN DECOMPOSITION WITH TILES

Handling Tile  $\leftrightarrow$  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  $\leftrightarrow$  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;
```

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

# DOMAIN DECOMPOSITION WITH TILES

Handling Tile  $\leftrightarrow$  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  $\leftrightarrow$  GPU/MPI Rank affinity

1. Determine neighbors in y direction:

(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

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

# DOMAIN DECOMPOSITION WITH TILES

Handling Tile  $\leftrightarrow$  GPU/MPI Rank affinity

1. Determine neighbors in y direction:

(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

```
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;
```

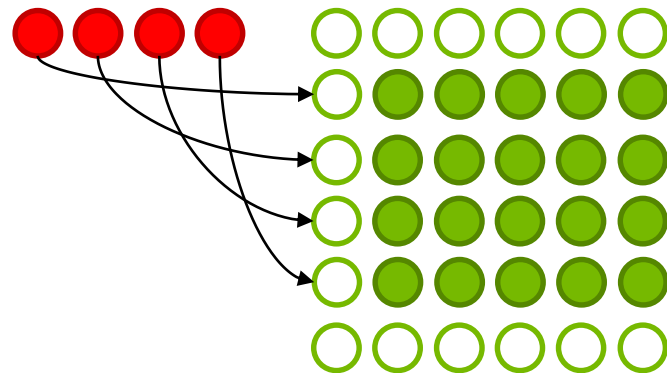
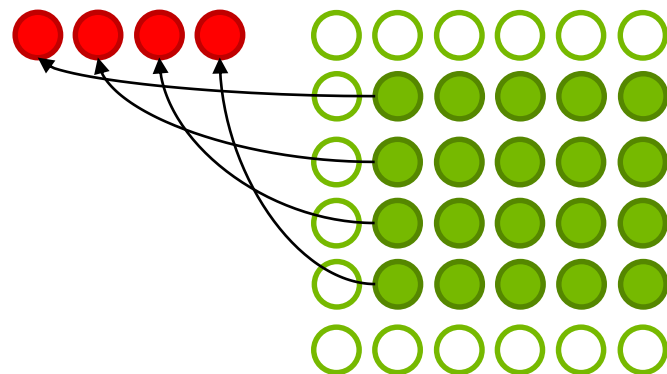
# EXAMPLE: JACOBI

## 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

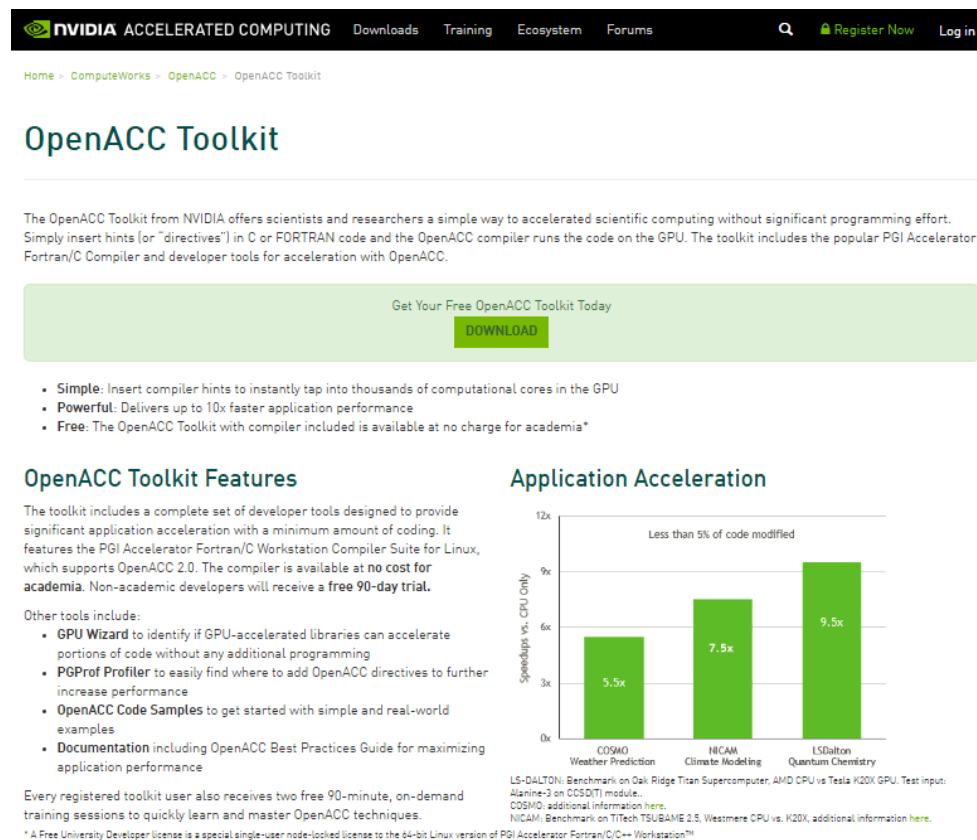
1. Go to <https://developer.nvidia.com/qwiklabs-signup>
2. Use OPENACC promo code to register and get free access
3. Receive a confirmation email with access instructions
4. Take 'Advanced Multi GPU Programming with MPI and OpenACC' lab:  
<http://bit.ly/oaccnvlab7>

## Questions?

Email to [openacc@nvidia.com](mailto:openacc@nvidia.com)

# INSTALL THE OPENACC TOOLKIT (OPTIONAL)

- ▶ Go to [developer.nvidia.com/openacc-toolkit](https://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



The screenshot shows the NVIDIA Accelerated Computing website for the OpenACC Toolkit. The header includes the NVIDIA logo and navigation links: Downloads, Training, Ecosystem, Forums, a search icon, and links to Register Now and Log in. The breadcrumb trail is Home > ComputeWorks > OpenACC > OpenACC Toolkit. The main heading is "OpenACC Toolkit". Below it, a paragraph describes the toolkit as a simple way to accelerate scientific computing without significant programming effort, mentioning the PGI Accelerator Fortran/C Compiler and developer tools. A green button labeled "DOWNLOAD" is prominent. Below the button, three bullet points highlight the toolkit's features: Simple (inserting hints for GPU acceleration), Powerful (up to 10x faster performance), and Free (available at no charge for academia). The "OpenACC Toolkit Features" section describes the complete set of developer tools, including the PGI Accelerator Fortran/C Workstation Compiler Suite for Linux, which supports OpenACC 2.0. It also lists other tools: GPU Wizard, PGProf Profiler, OpenACC Code Samples, and documentation. To the right, the "Application Acceleration" section features a bar chart showing speedups for three applications: COSMO Weather Prediction (5.5x), NICAM Climate Modeling (7.5x), and LSDalton Quantum Chemistry (9.5x). A note above the chart states "Less than 5% of code modified". Below the chart, detailed benchmark information is provided for each application, including the hardware used (Oak Ridge Titan Supercomputer, AMD CPU vs. Tesla K20X GPU) and the test input.

**OpenACC Toolkit Features**

The toolkit includes a complete set of developer tools designed to provide significant application acceleration with a minimum amount of coding. It features the PGI Accelerator Fortran/C Workstation Compiler Suite for Linux, which supports OpenACC 2.0. The compiler is available at **no cost for academia**. Non-academic developers will receive a **free 90-day trial**.

Other tools include:

- **GPU Wizard** to identify if GPU-accelerated libraries can accelerate portions of code without any additional programming
- **PGProf Profiler** to easily find where to add OpenACC directives to further increase performance
- **OpenACC Code Samples** to get started with simple and real-world examples
- **Documentation** including OpenACC Best Practices Guide for maximizing application performance

Every registered toolkit user also receives two free 90-minute, on-demand training sessions to quickly learn and master OpenACC techniques.

**Application Acceleration**

Less than 5% of code modified

Application	Speedups vs. CPU Only
COSMO Weather Prediction	5.5x
NICAM Climate Modeling	7.5x
LSDalton Quantum Chemistry	9.5x

LS-DALTON: Benchmark on Oak Ridge Titan Supercomputer, AMD CPU vs. Tesla K20X GPU. Test input: Algalene-3 on CCSD(T) module.  
COSMO: additional information [here](#).  
NICAM: Benchmark on TITech TSUBAME 2.5, Westmere CPU vs. K20X, additional information [here](#).

\* A Free University Developer license is a special single-user node-locked license to the 64-bit Linux version of PGI Accelerator Fortran/C/C++ Workstation™.



# 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>

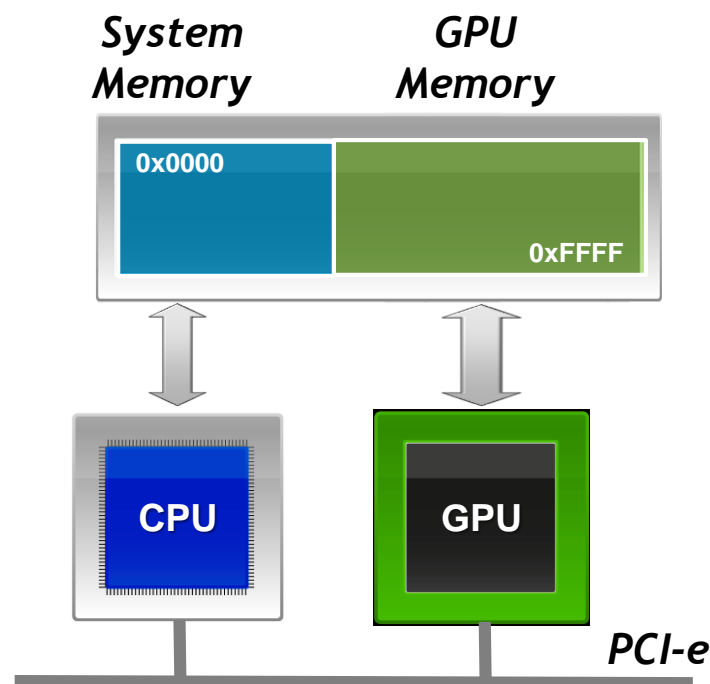
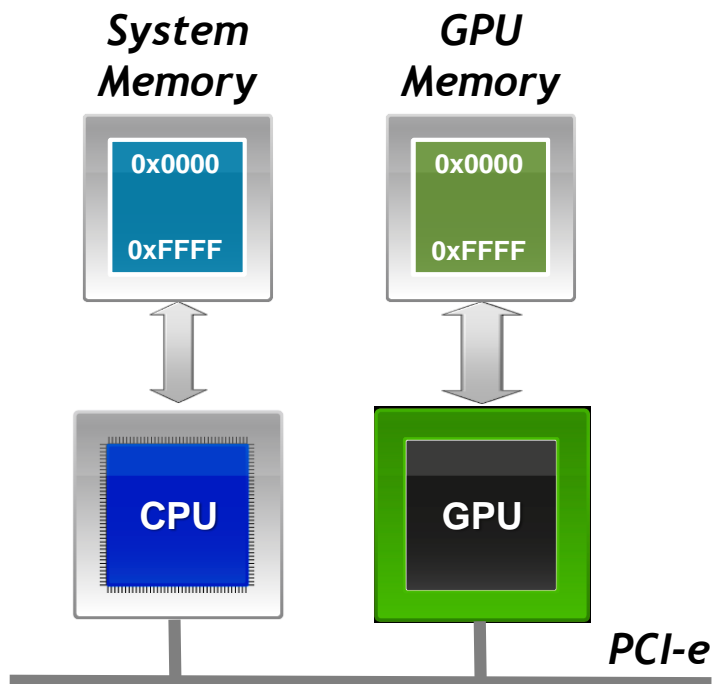
Questions? Email [openacc@nvidia.com](mailto:openacc@nvidia.com)

# CUDA-aware MPI implementation details

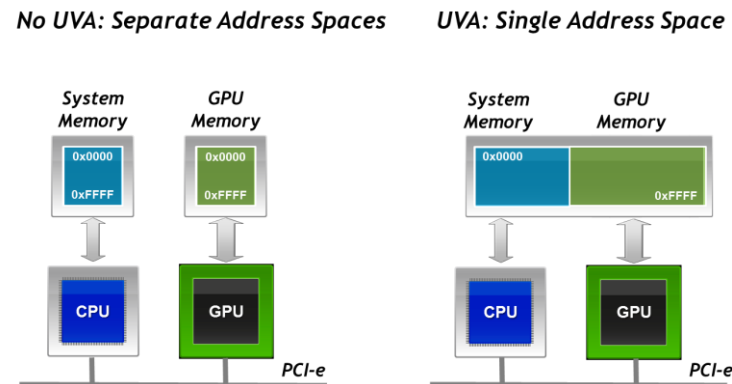
# UNIFIED VIRTUAL ADDRESSING

*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

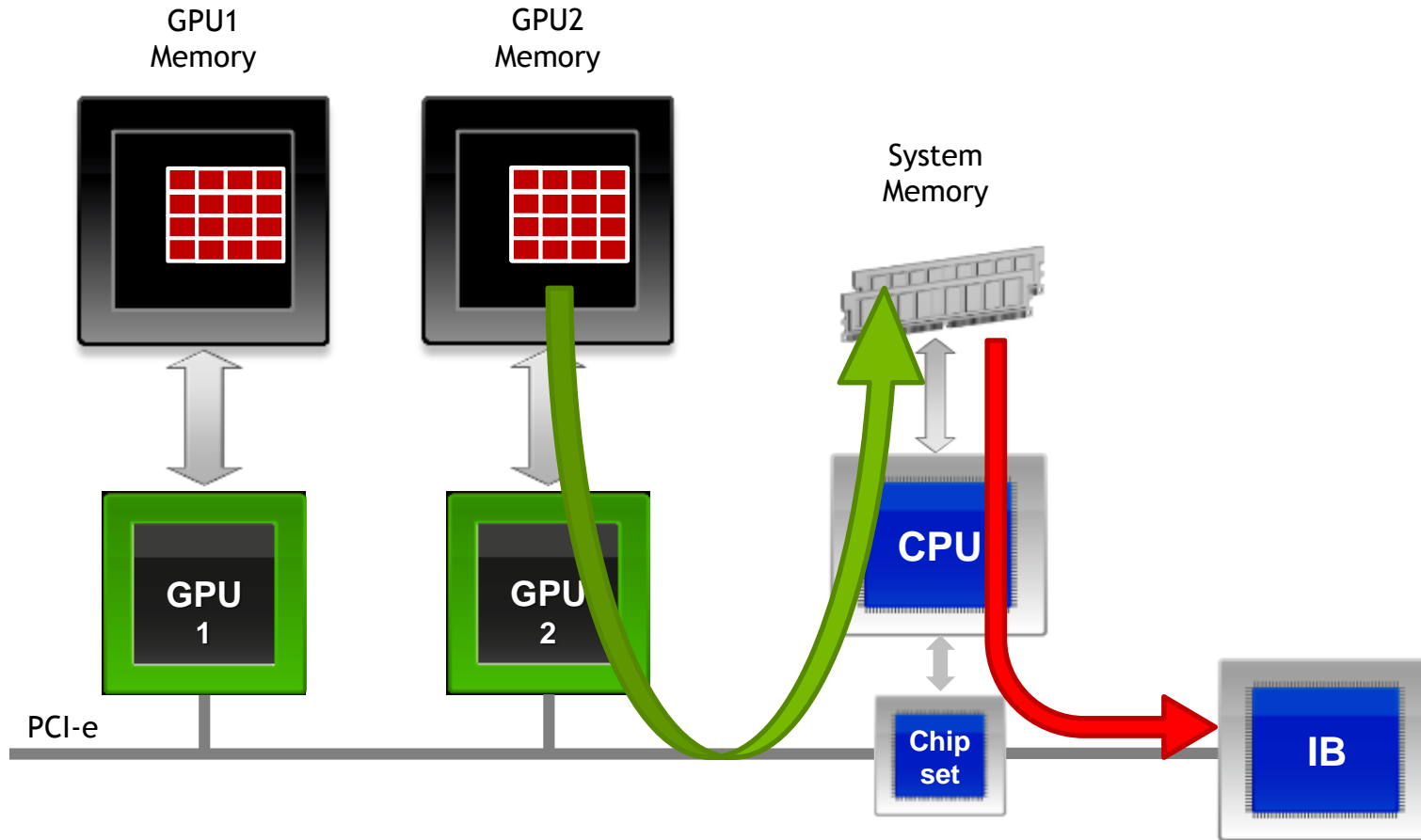
Enable libraries to simplify their interfaces (e.g. MPI and cudaMemcpy)

Supported on devices with compute capability 2.0+ for

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

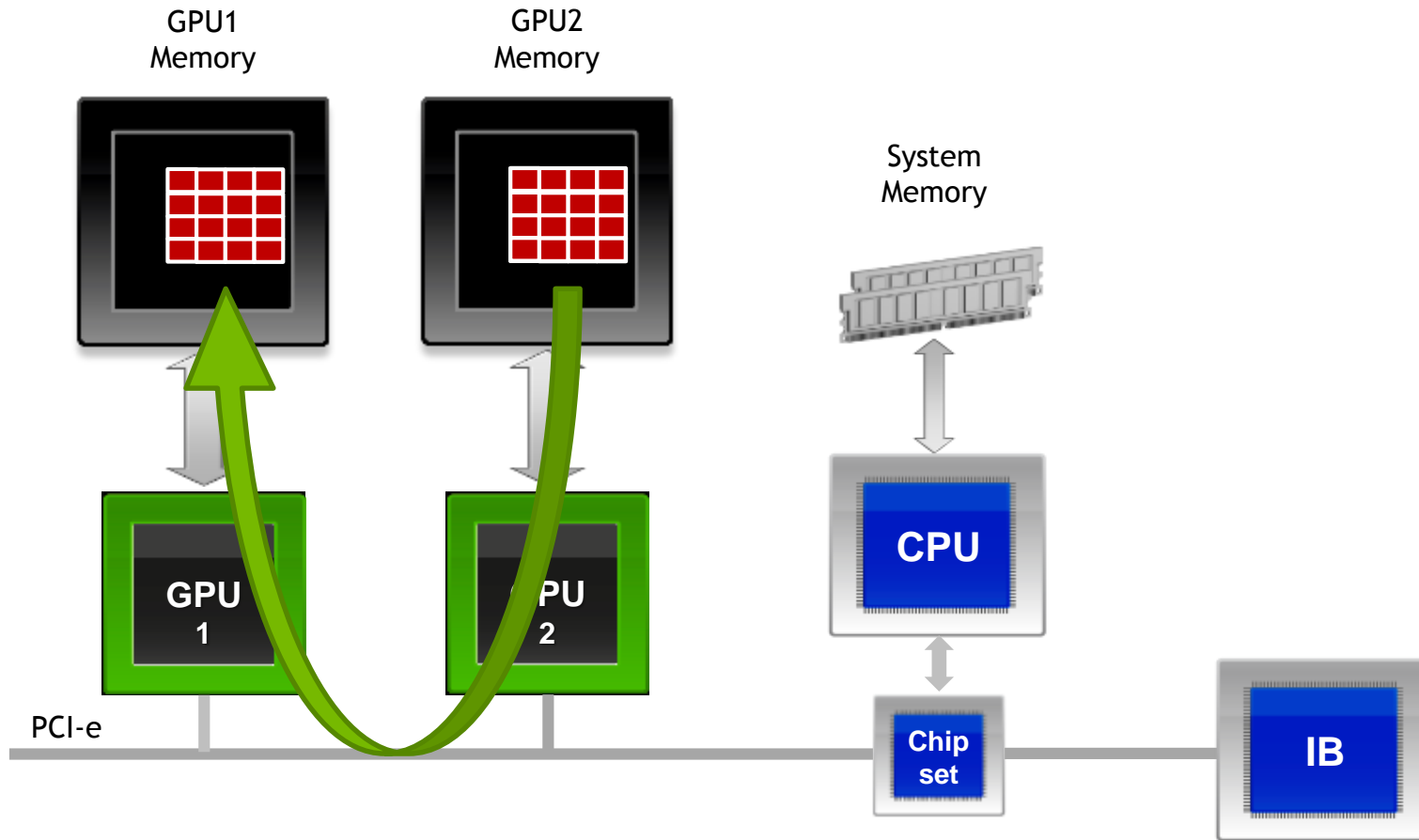
# NVIDIA GPUDIRECT™

Accelerated Communication with Network & Storage Devices



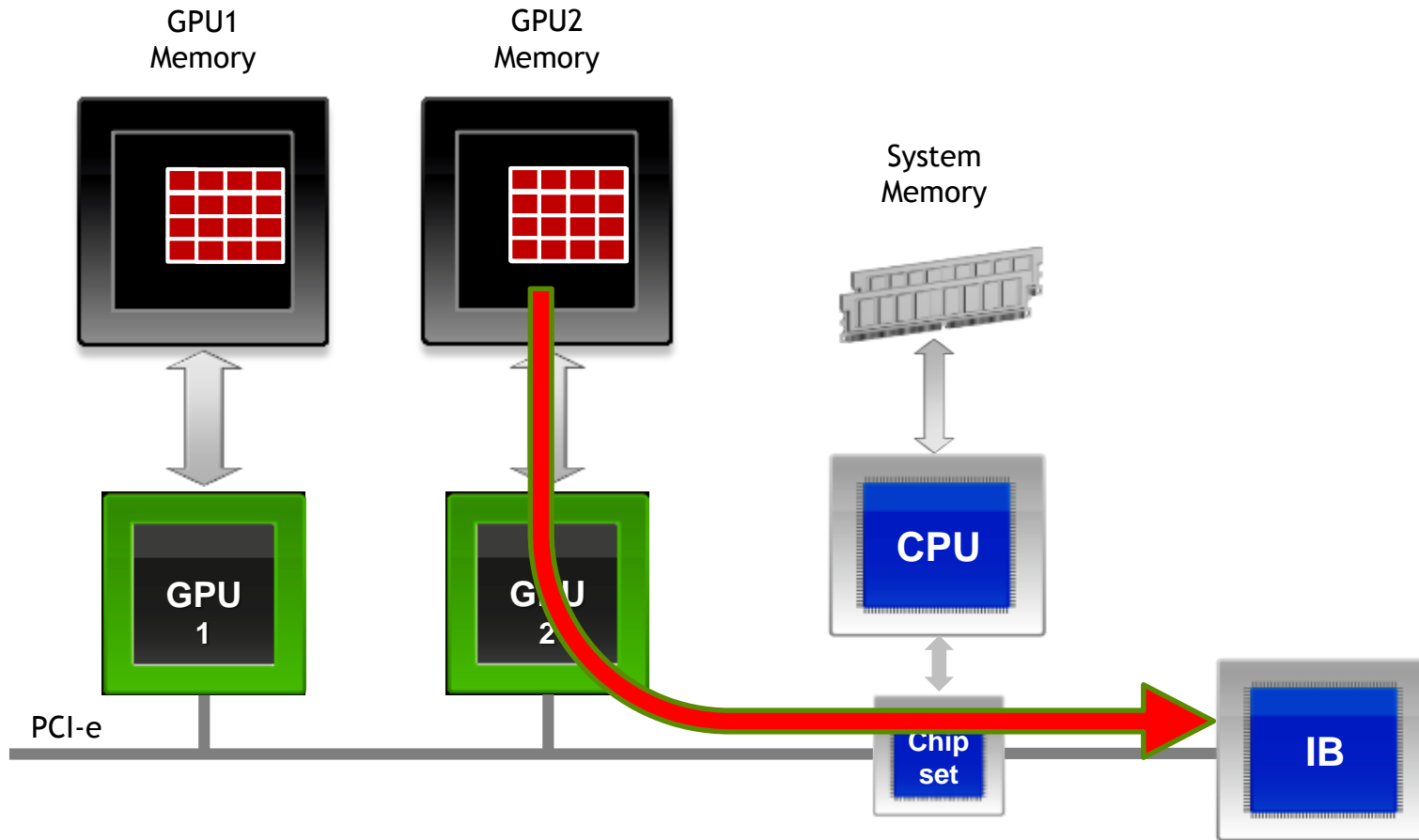
# NVIDIA GPUDIRECT™

## 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

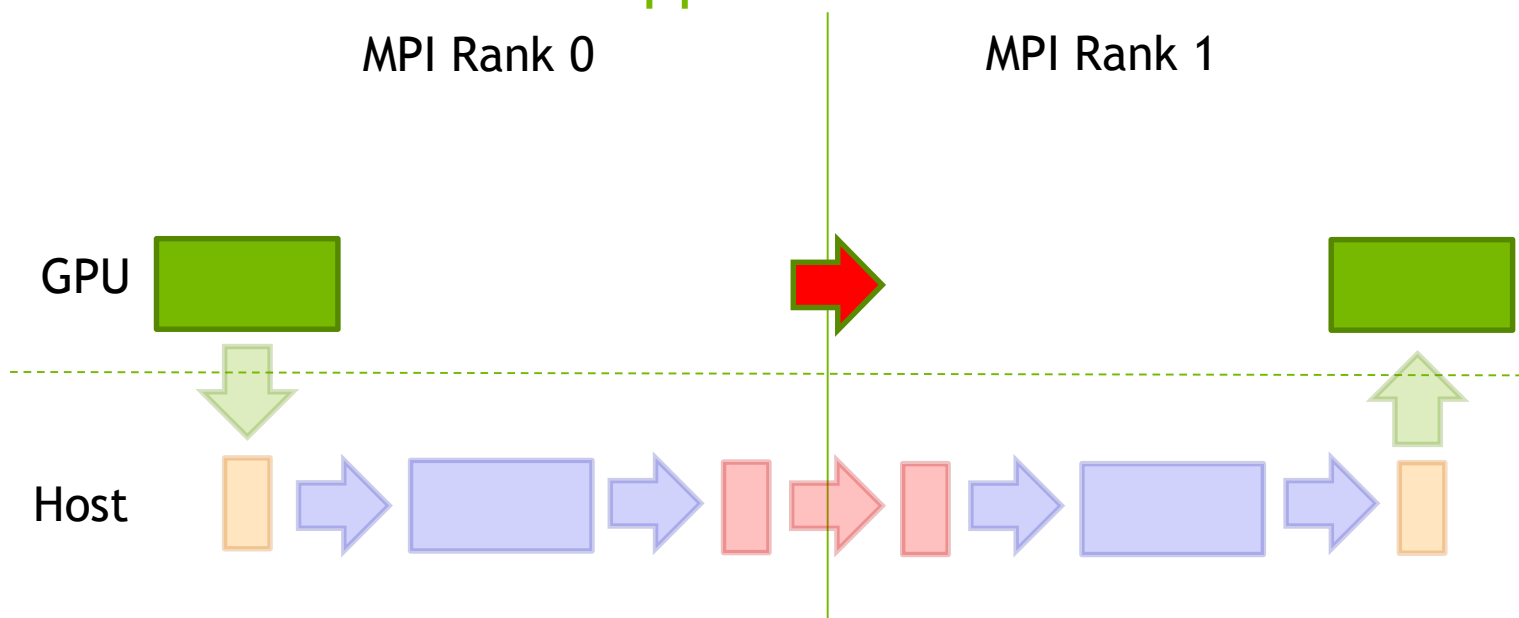
Depending on the MPI implementation, message size, system setup, ...  
situation might be different

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);
```

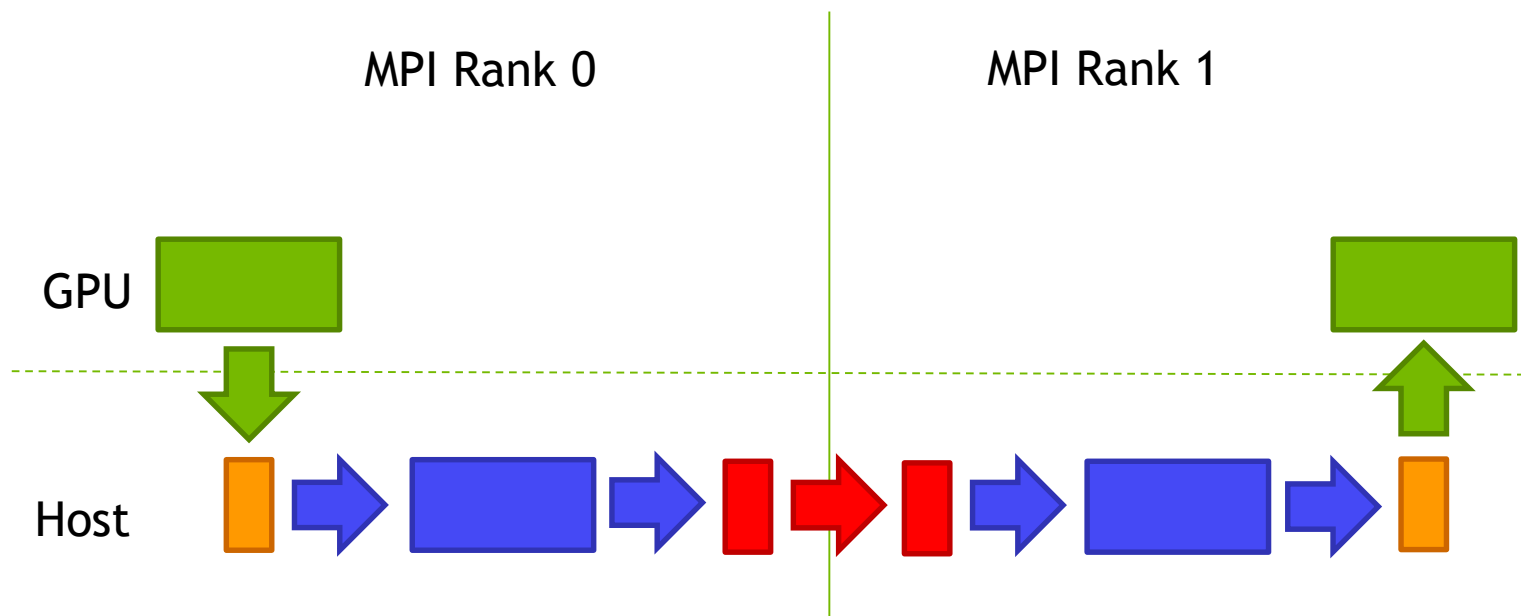
```
MPI_Recv(r_buf_d,size,MPI_CHAR,0,tag,MPI_COMM_WORLD,&stat);
```

# MPI GPU TO REMOTE GPU

Support for RDMA



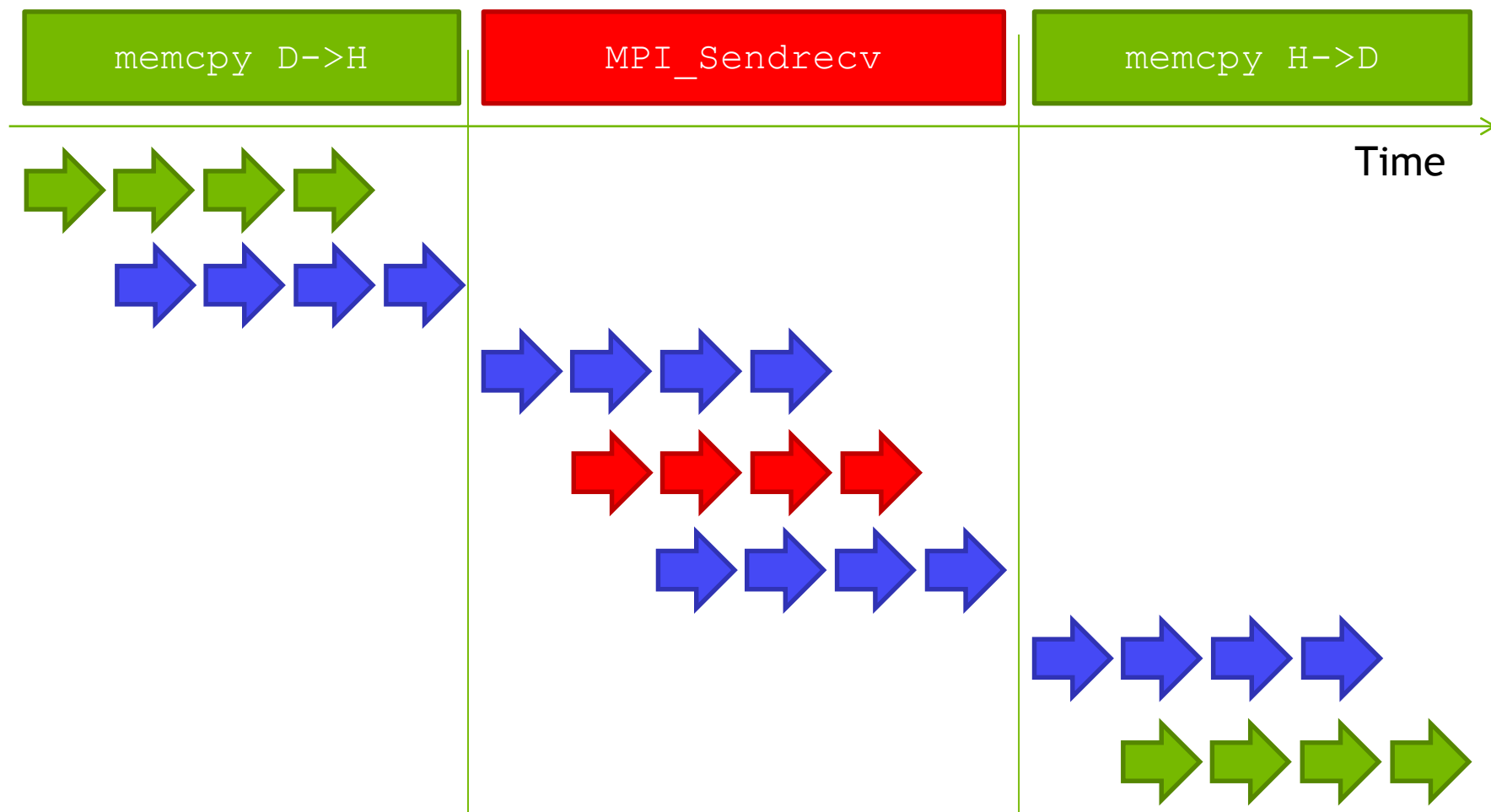
# REGULAR MPI GPU TO REMOTE GPU



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

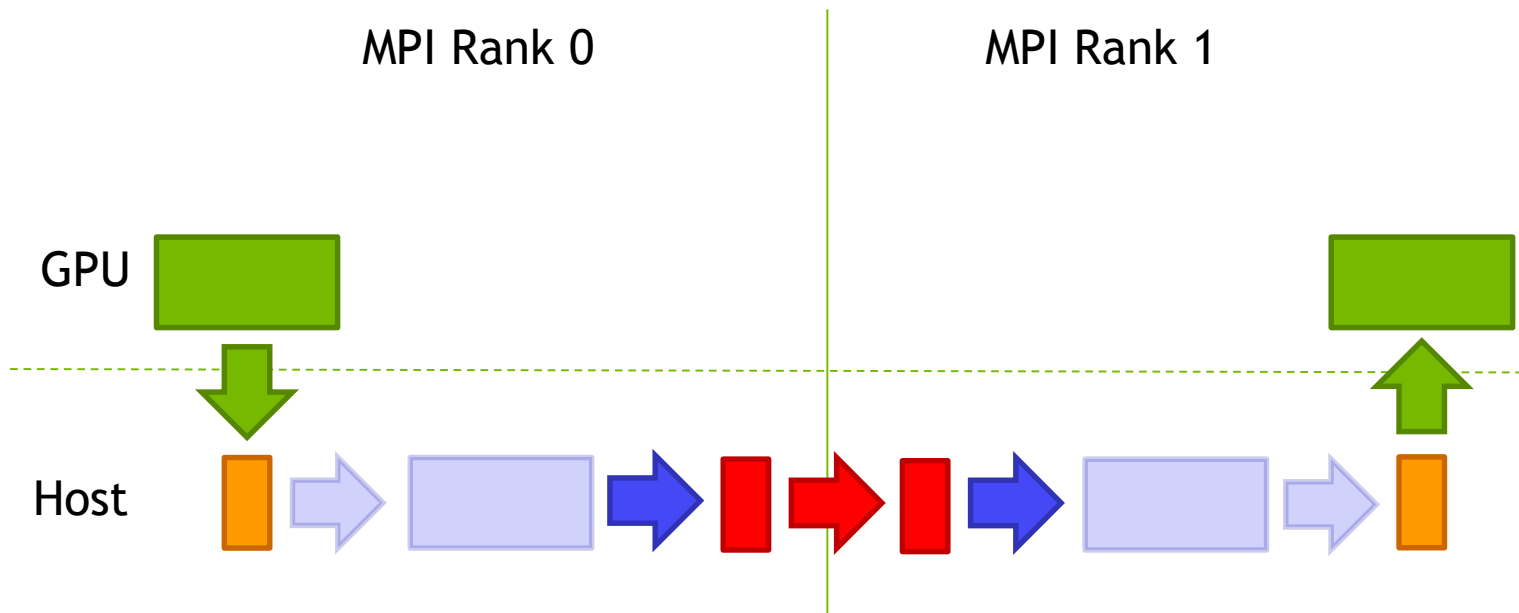
```
MPI_Recv(r_buf_h,size,MPI_CHAR,0,tag,MPI_COMM_WORLD,&stat);  
cudaMemcpy(r_buf_d,r_buf_h,size,cudaMemcpyHostToDevice);
```

# REGULAR MPI GPU TO REMOTE GPU



# MPI GPU TO REMOTE GPU

without GPUDirect

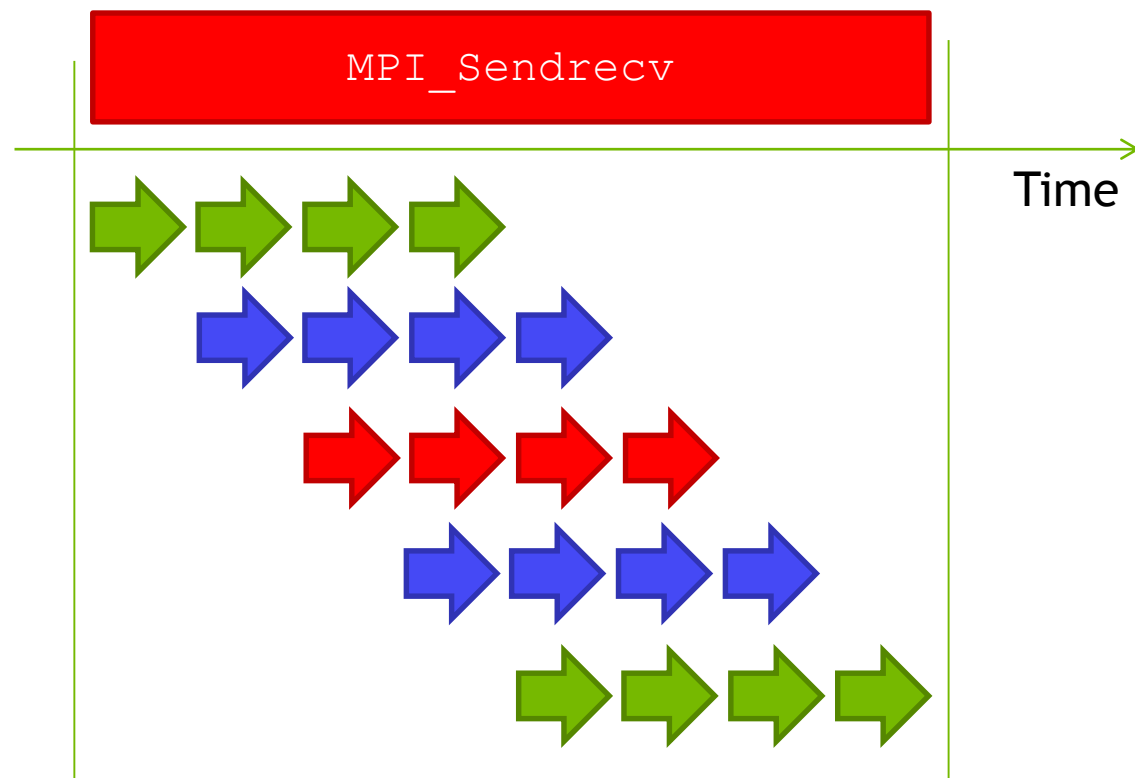


```
MPI_Send(s_buf_d,size,MPI_CHAR,1,tag,MPI_COMM_WORLD);
```

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

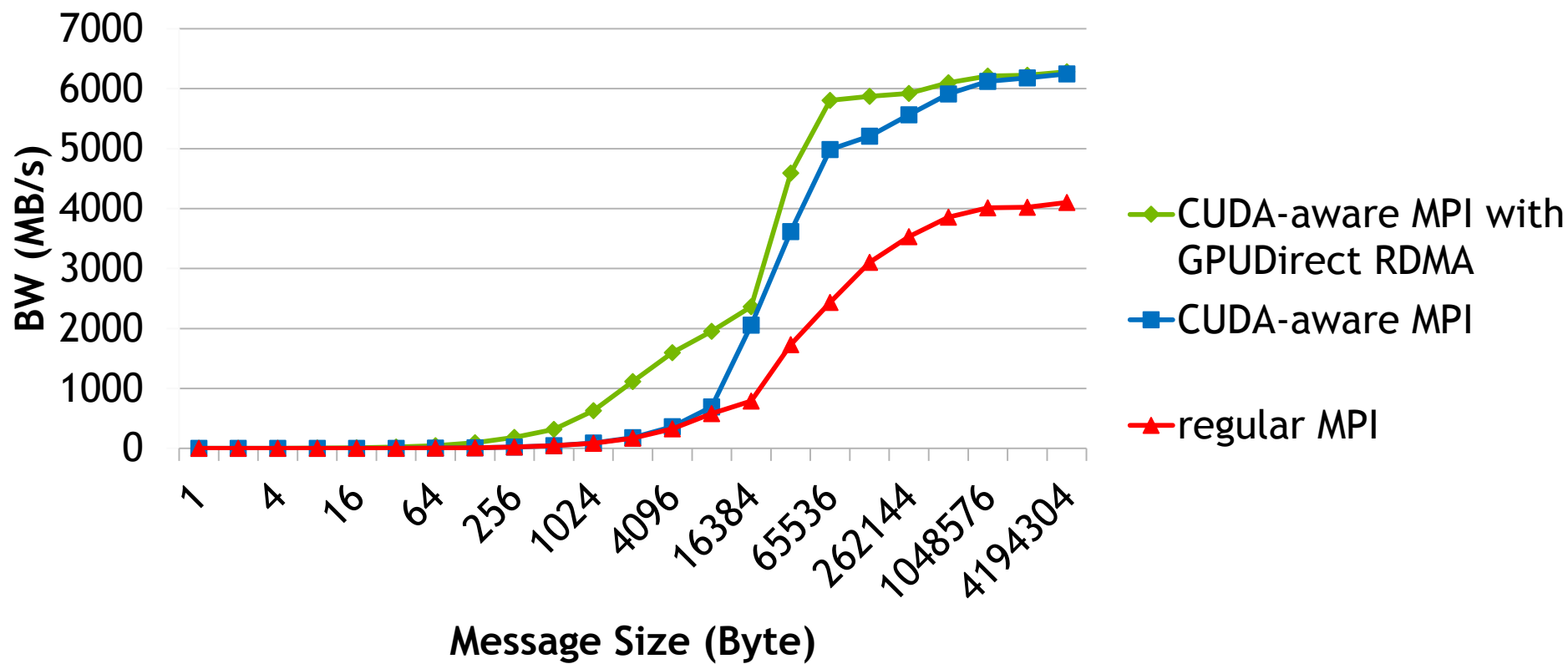
# MPI GPU TO REMOTE GPU

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