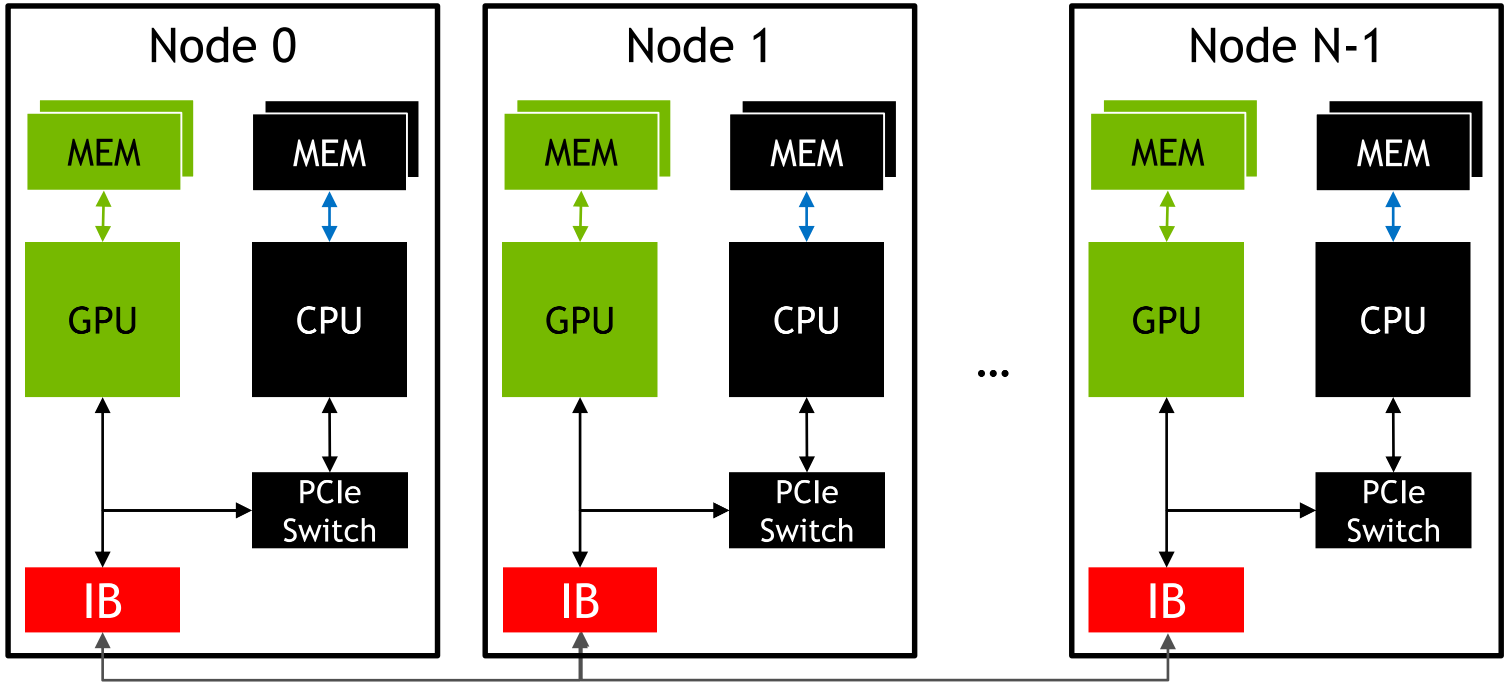
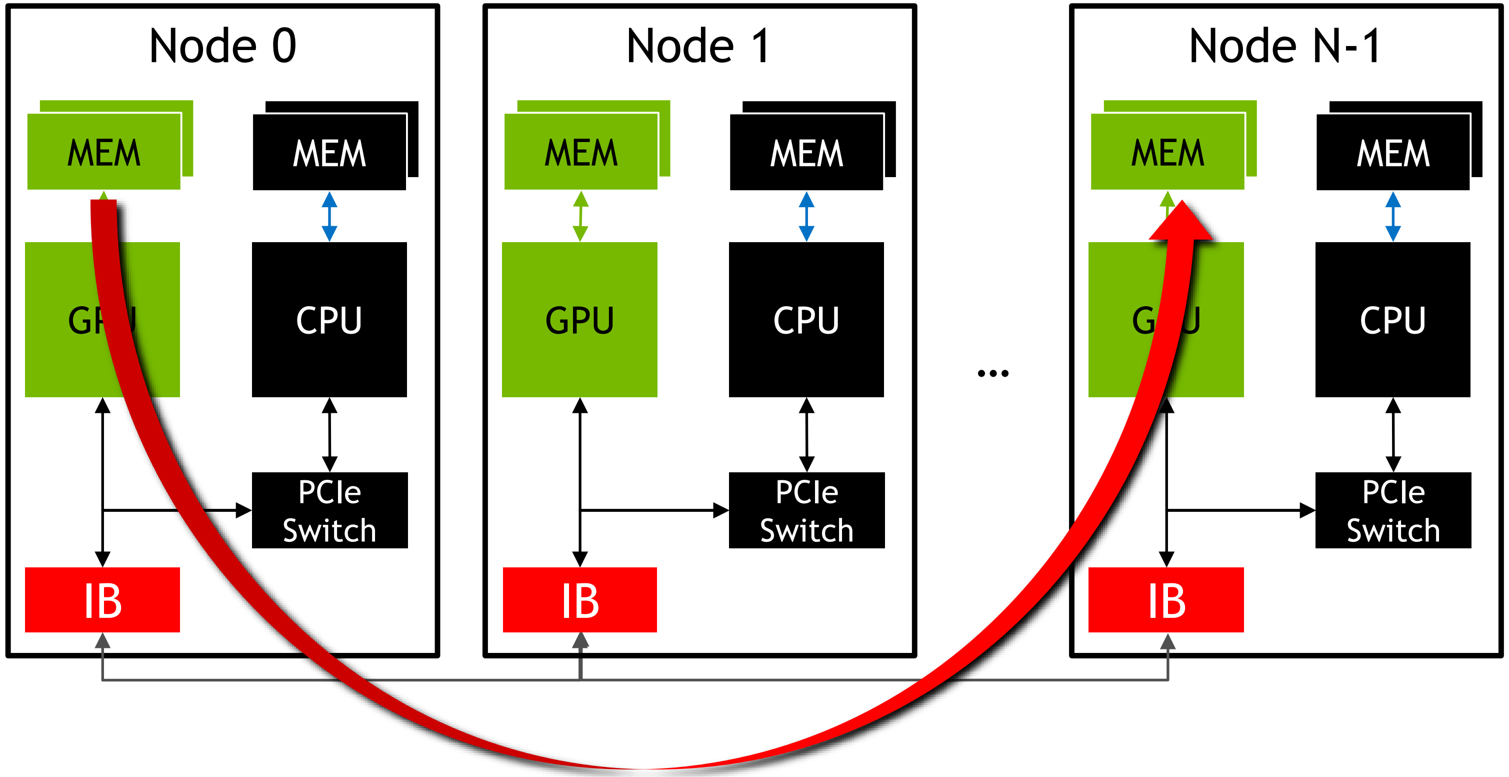
Multi-GPU Programming with MPI

Cheng Yi, Senior Solution Architect, NVIDIA

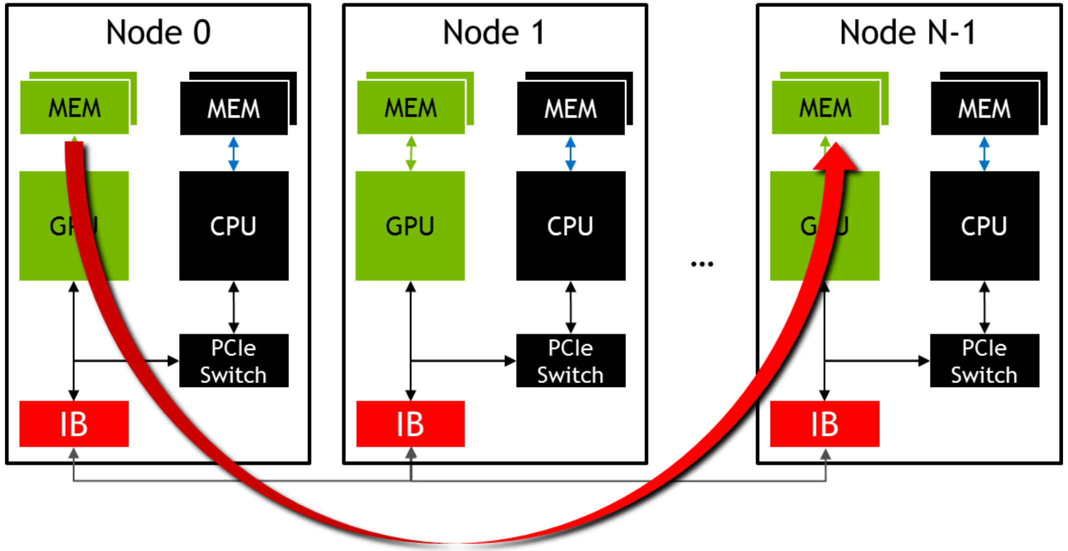
MPI+CUDA


Node 0
MEM
MEM
GPU
CPU
PCIe Switch
IB
Node 1
MEM
MEM
GPU
CPU
PCIe Switch
IB
...
Node N-1
MEM
MEM
GPU
CPU
PCIe Switch
IB

MPI+CUDA


Node 0
MEM
MEM
GPU
CPU
PCIe Switch
IB
Node 1
MEM
MEM
GPU
CPU
PCIe Switch
IB
…
Node N-1
MEM
MEM
GPU
CPU
PCIe Switch
IB

MPI+CUDA



```
//MPI rank 0
MPI_Send(s_buf_d,size,MPI_CHAR,n-1,tag,MPI_COMM_WORLD);

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

YOU WILL LEARN

What MPI is

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

What CUDA-aware MPI is

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

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, …

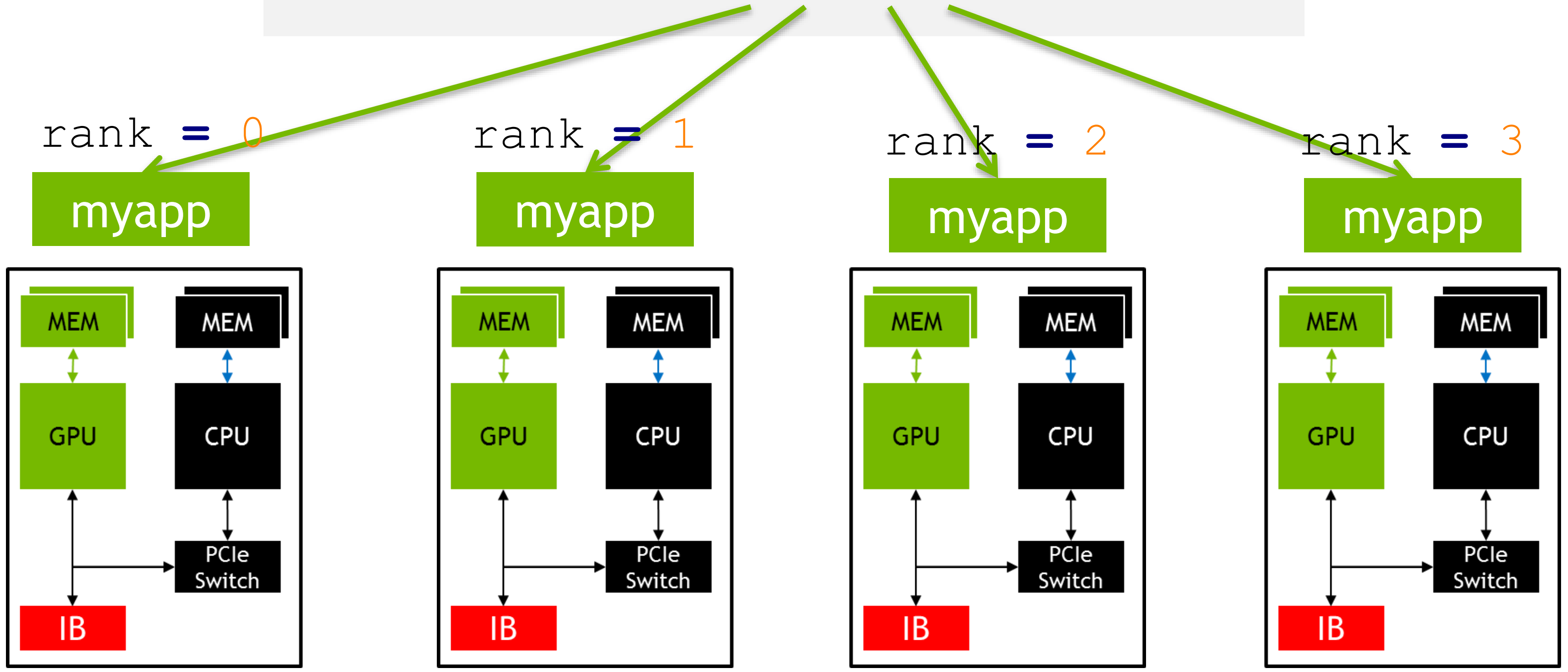
MPI - SKELETON

```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>
```



A SIMPLE EXAMPLE

EXAMPLE: JACOBI SOLVER

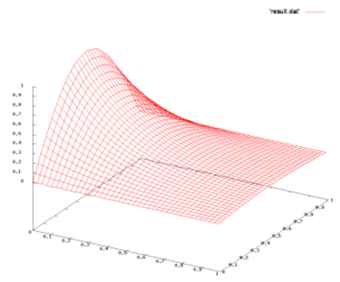
Solves the 2D-Laplace Equation on a rectangle

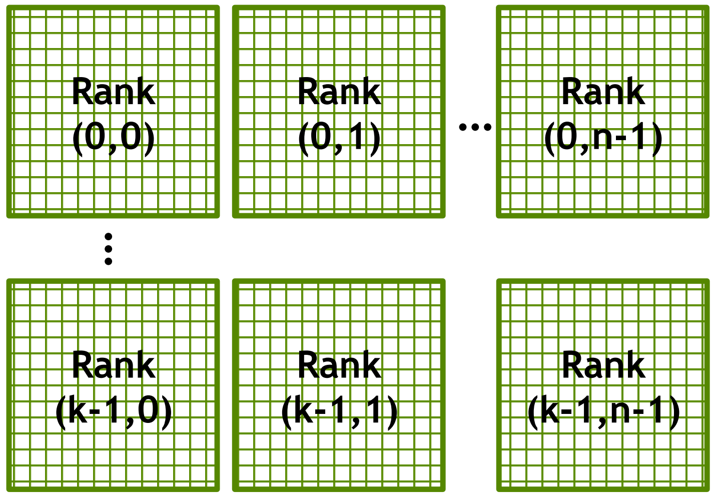
$$\Delta \boldsymbol{u}(\boldsymbol{x},\boldsymbol{y}) = \boldsymbol{0}\ \forall\ (\boldsymbol{x},\boldsymbol{y}) \in \Omega \backslash \boldsymbol{\delta}\Omega$$

Dirichlet boundary conditions (constant values on boundaries)

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

2D domain decomposition with n x k domains





EXAMPLE: JACOBI SOLVER

Single GPU

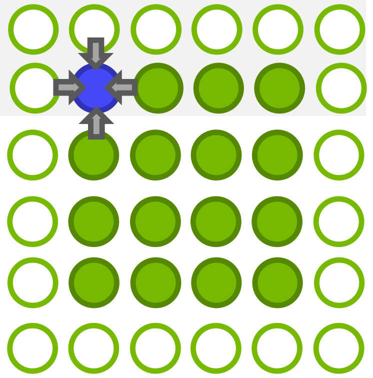
While not converged

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`

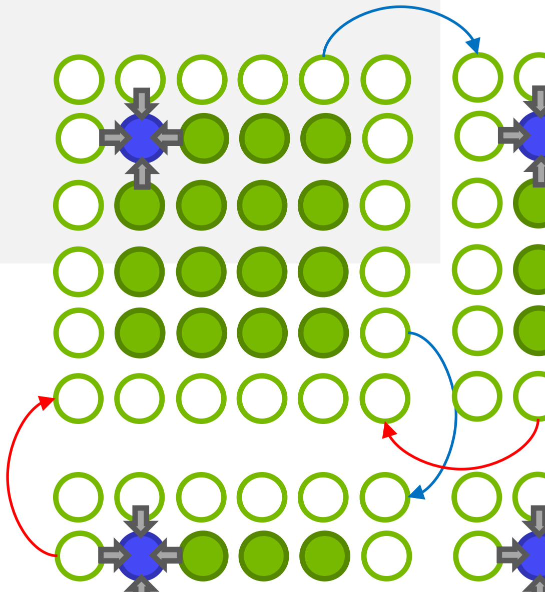
Next iteration



EXAMPLE: JACOBI SOLVER

Multi GPU

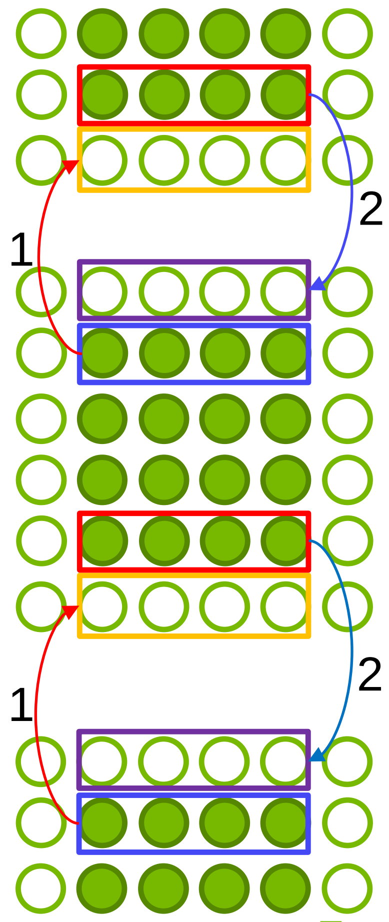
```
While not converged
    Do Jacobi step:
        for (int iy=1; iy < ny-1; ++iy)
        for (int ix=1; ix < nx-1; ++ix)
            u_new[ix][iy] = 0.0f - 0.25f*(  u[ix-1][iy] + u[ix+1][iy]
                                          + u[ix][iy-1] + u[ix][iy+1]);
    Exchange halo with 2  4 neighbors
    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

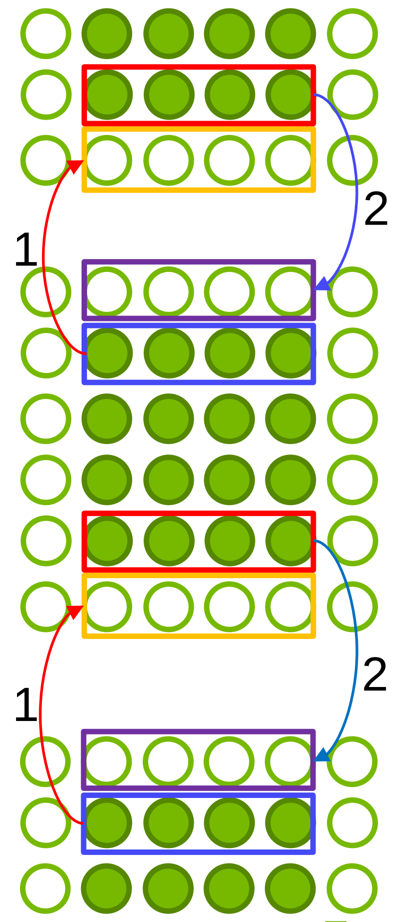
Top/Bottom Halo

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);
}
```

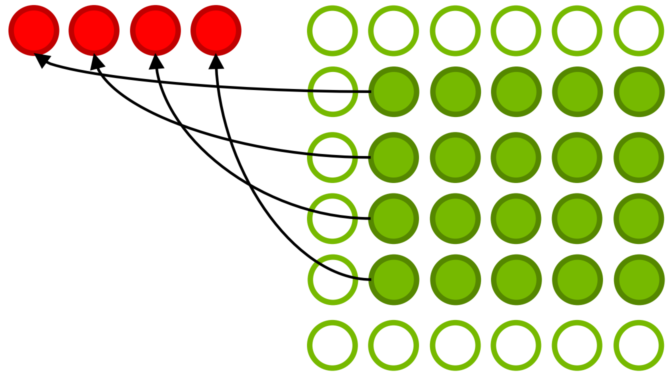
CUDA

```
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);
```



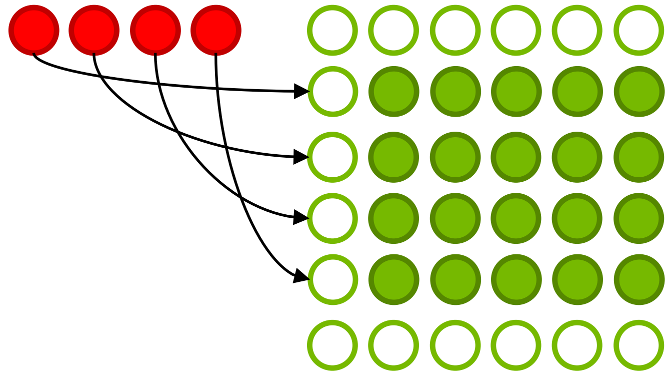
EXAMPLE: JACOBI

Left/Right Halo



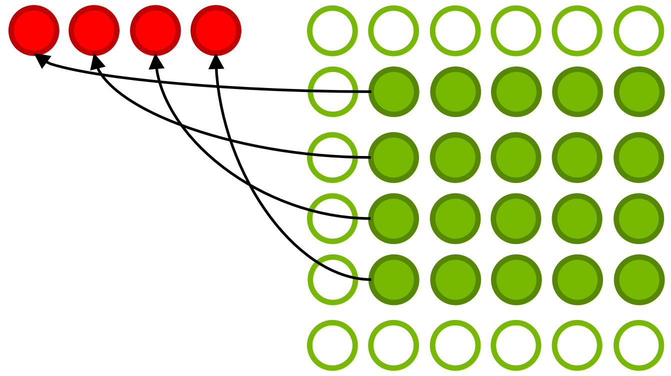
OpenACC

```
//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];
```



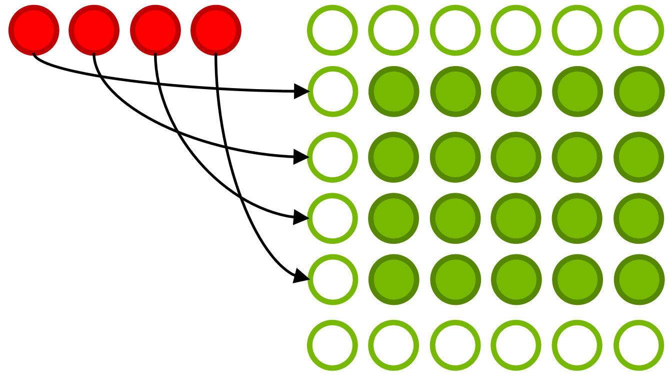
EXAMPLE: JACOBI

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`

```
$ MV2_USE_CUDA=1 mpirun -np ${np} ./myapp <args>
```

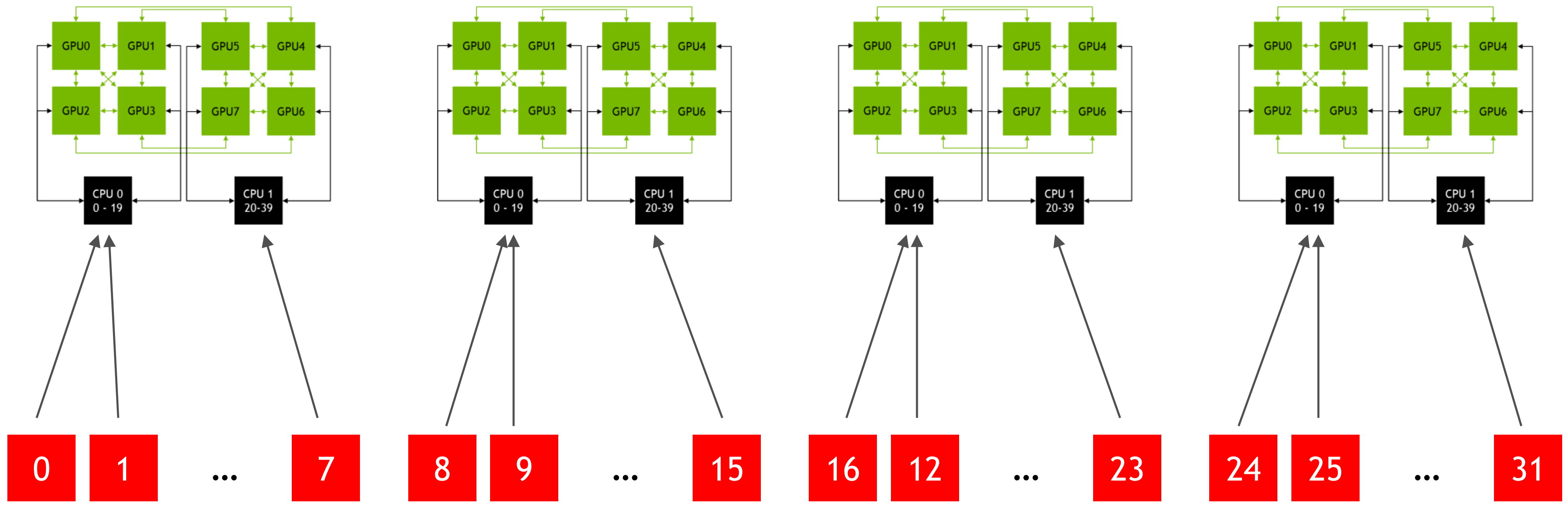
Open MPI: CUDA-aware features are enabled per default

Cray: `MPICH_RDMA_ENABLED_CUDA`

IBM Spectrum MPI:

```
$ mpirun -gpu -np ${np} ./myapp <args>
```

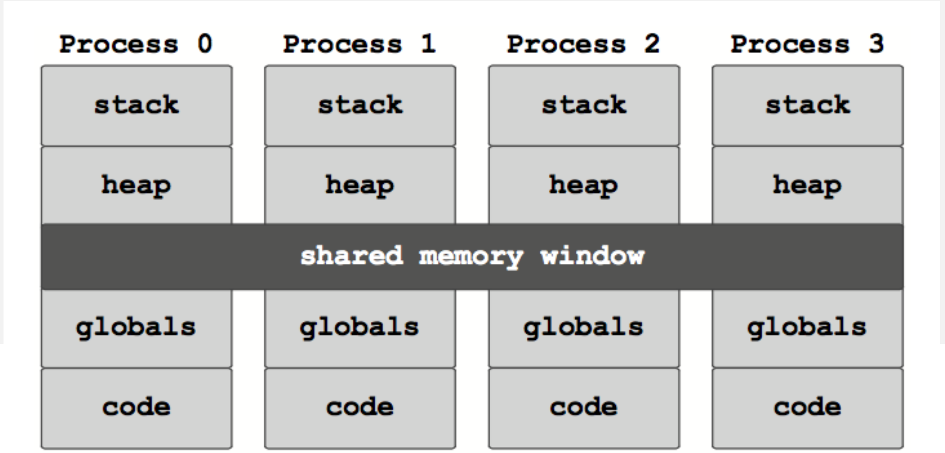
HANDLING MULTIPLE MULTI GPU NODES

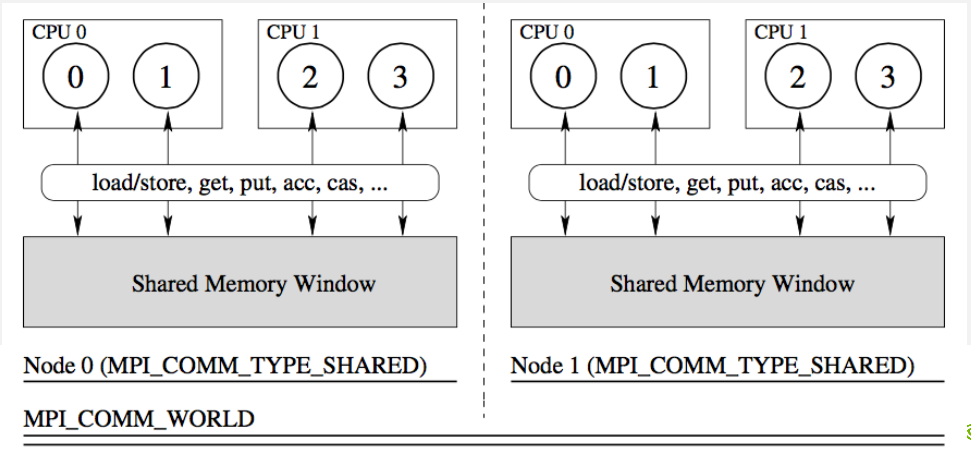


HANDLING MULTIPLE MULTI GPU NODES

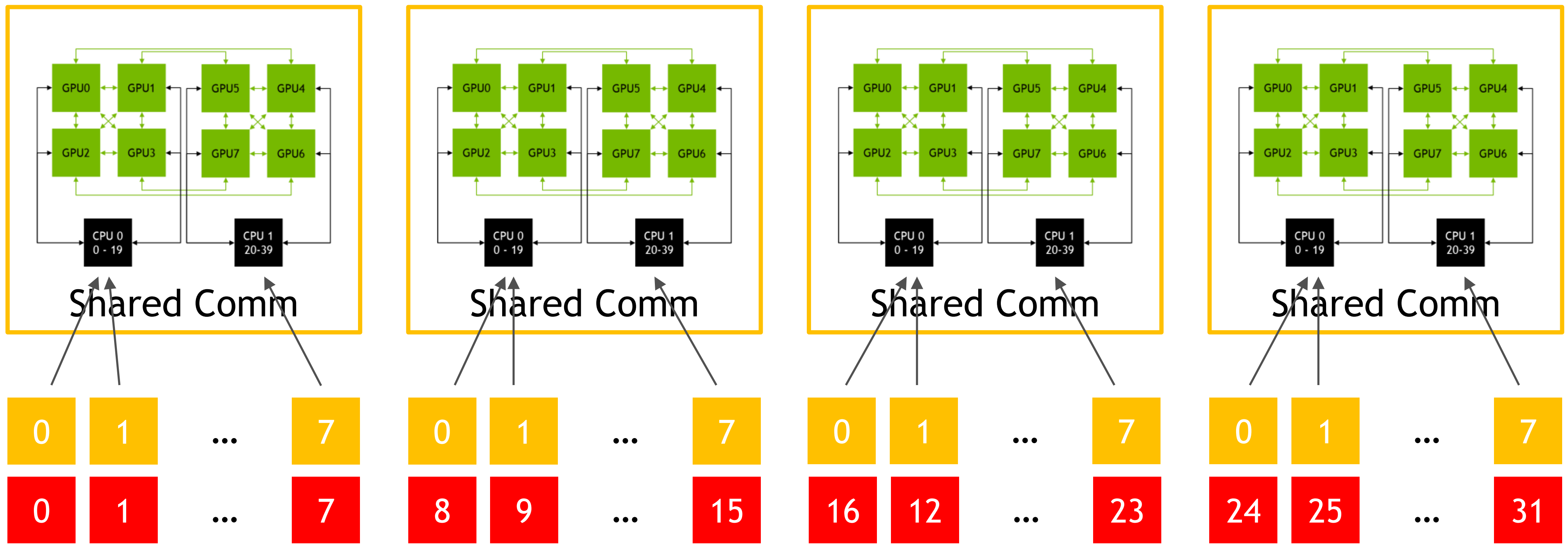
How to determine the local rank? – MPI-3

```
MPI_Comm loc_comm;

MPI_Comm_split_type(MPI_COMM_WORLD, MPI_COMM_TYPE_SHARED, rank, MPI_INFO_NULL, &loc_comm);

int local_rank = -1;

MPI_Comm_rank(loc_comm,&local_rank);

MPI_Comm_free(&loc_comm);
```




CPU 0
CPU 1
0
1
2
3
load/store, get, put, acc, cas, ...
Shared Memory Window
Node 0 (MPI_COMM_TYPE_SHARED)
Node 1 (MPI_COMM_TYPE_SHARED)
MPI_COMM_WORLD

HANDLING MULTIPLE MULTI GPU NODES



HANDLING MULTIPLE MULTI GPU NODES

GPU-affinity

Use local rank:

```
int local_rank = -1;
MPI_Comm_rank(local_comm,&local_rank);
int num_devices = 0;
cudaGetDeviceCount(&num_devices);
cudaSetDevice(local_rank % num_devices);
```

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])
```

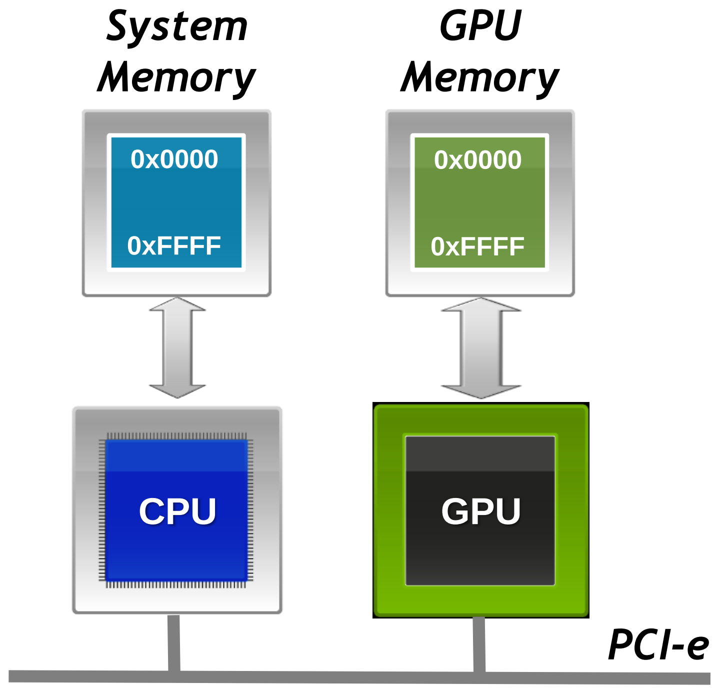
CUDA

```
//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);
```

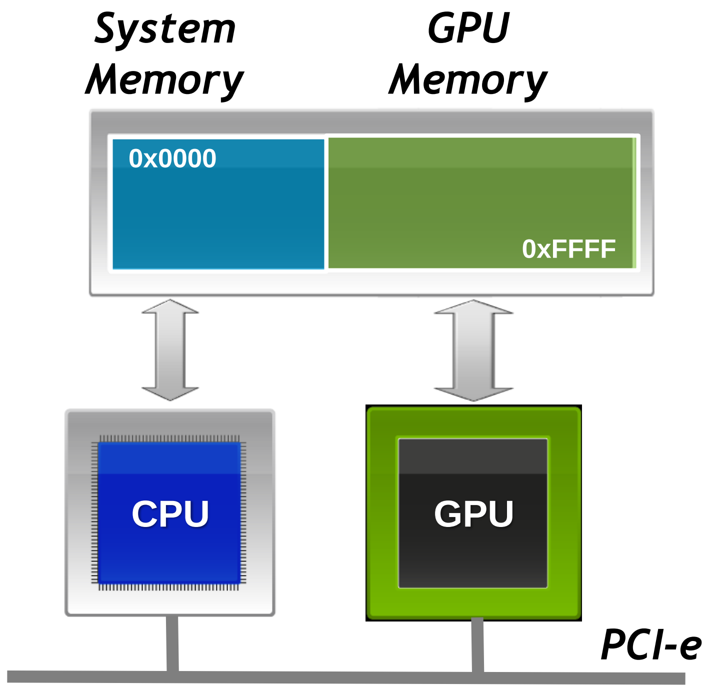
THE DETAILS

UNIFIED VIRTUAL ADDRESSING

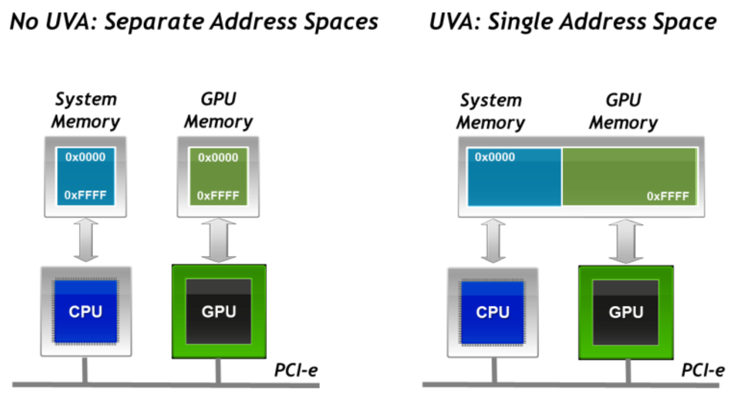
No UVA: Separate Address Spaces



UVA: Single Address Space



UNIFIED VIRTUAL ADDRESSING



One address space for all CPU and GPU memory

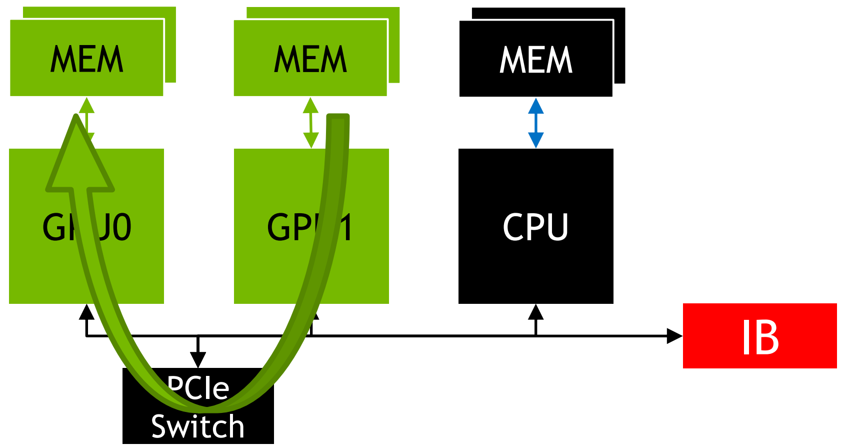
- Determine physical memory location from a pointer value
- Enable libraries to simplify their interfaces (e.g. MPI and cudaMemcpy)

Supported on devices with compute capability 2.0+ for

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

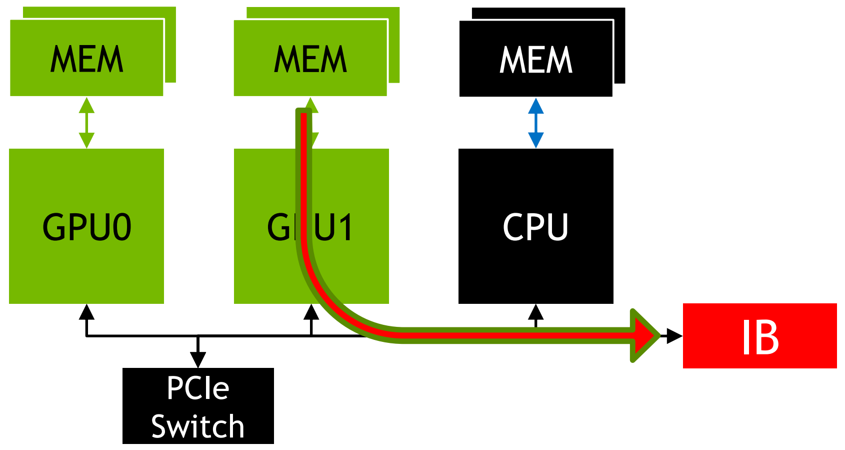
NVIDIA GPUDIRECT™

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

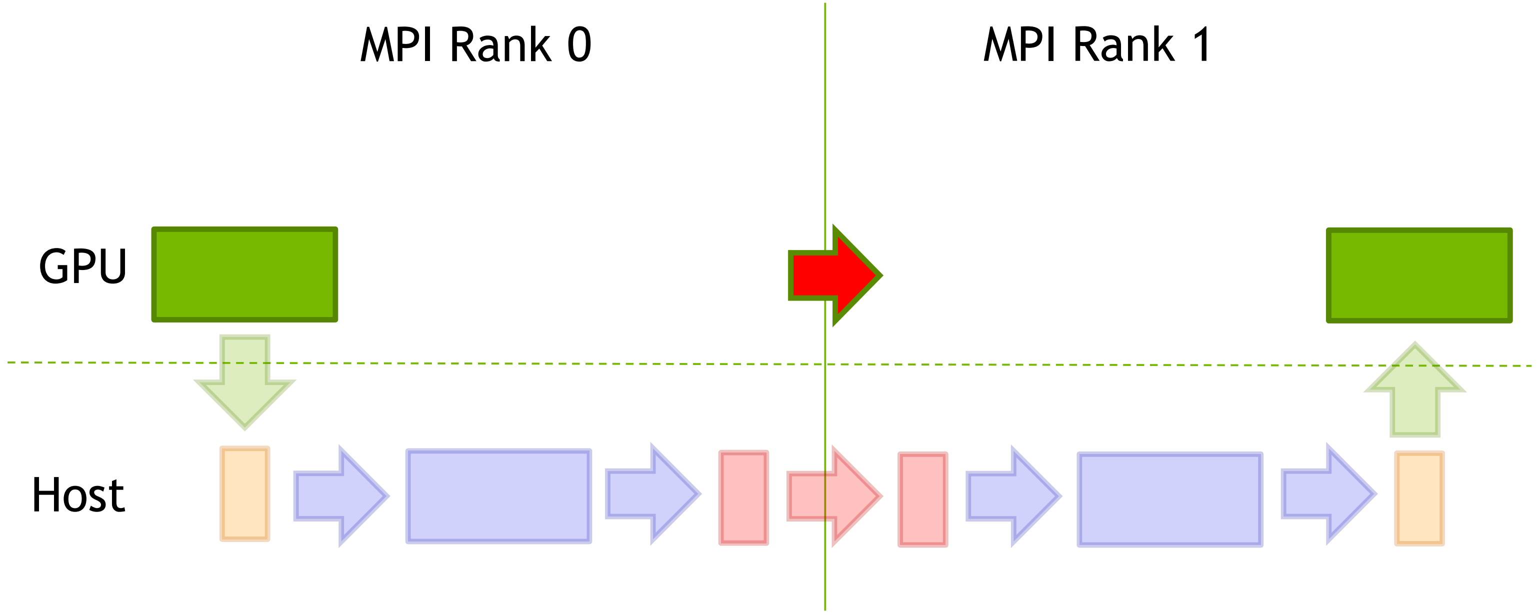
Show how CUDA+MPI works in principle

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

Two GPUs in two nodes

MPI GPU TO REMOTE GPU

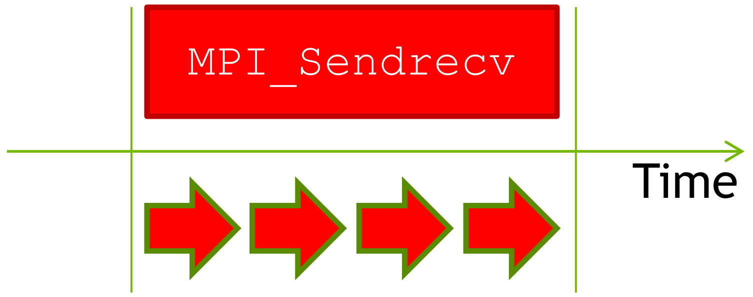
Support for RDMA



```
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);
```

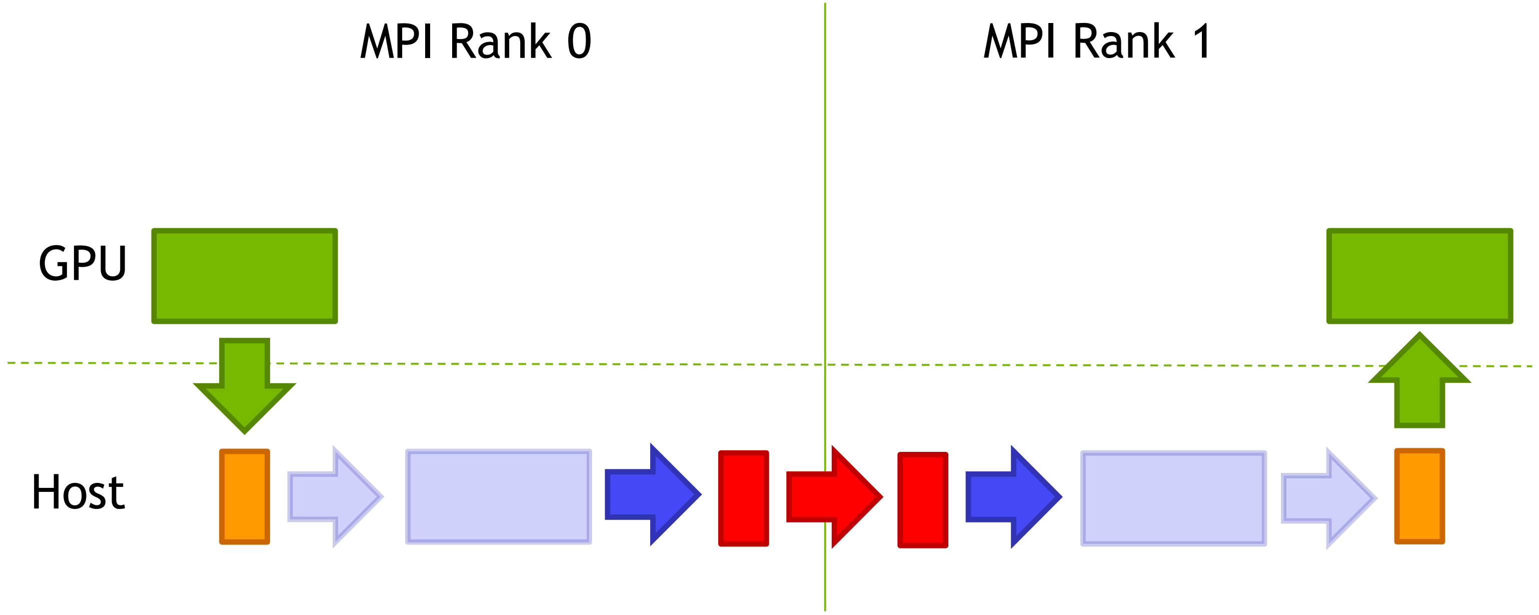
MPI GPU TO REMOTE GPU

Support for RDMA



MPI GPU TO REMOTE GPU

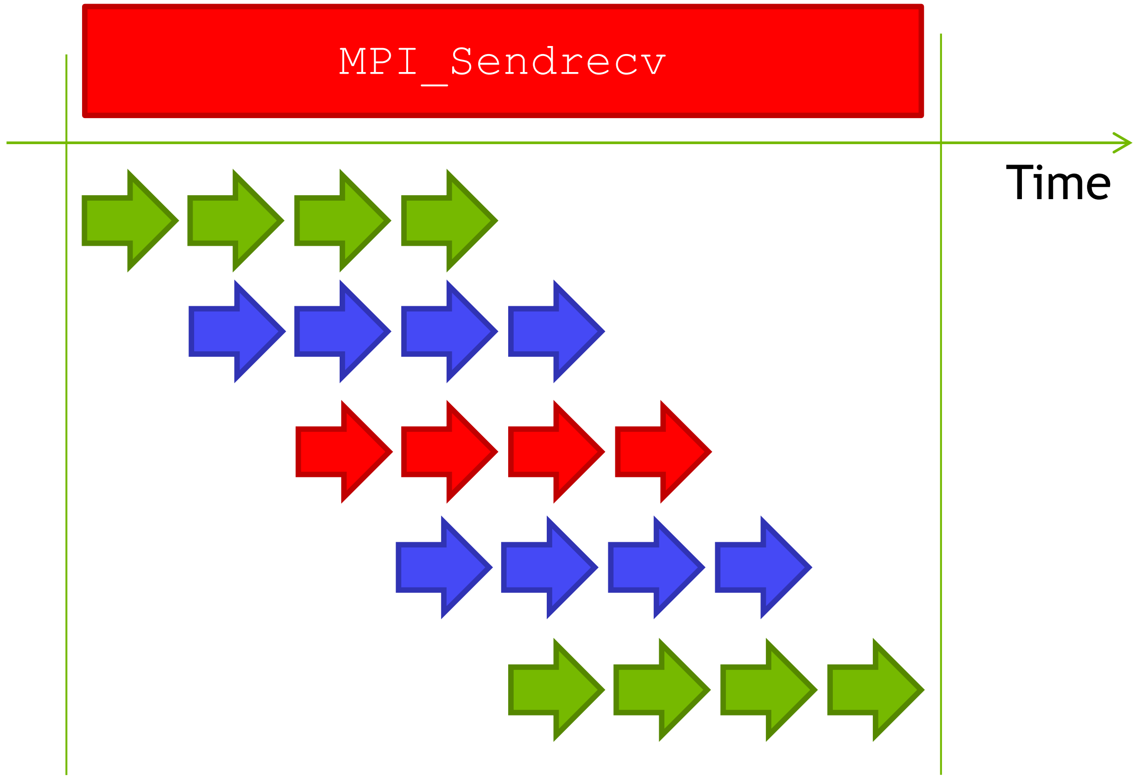
without GPUDirect



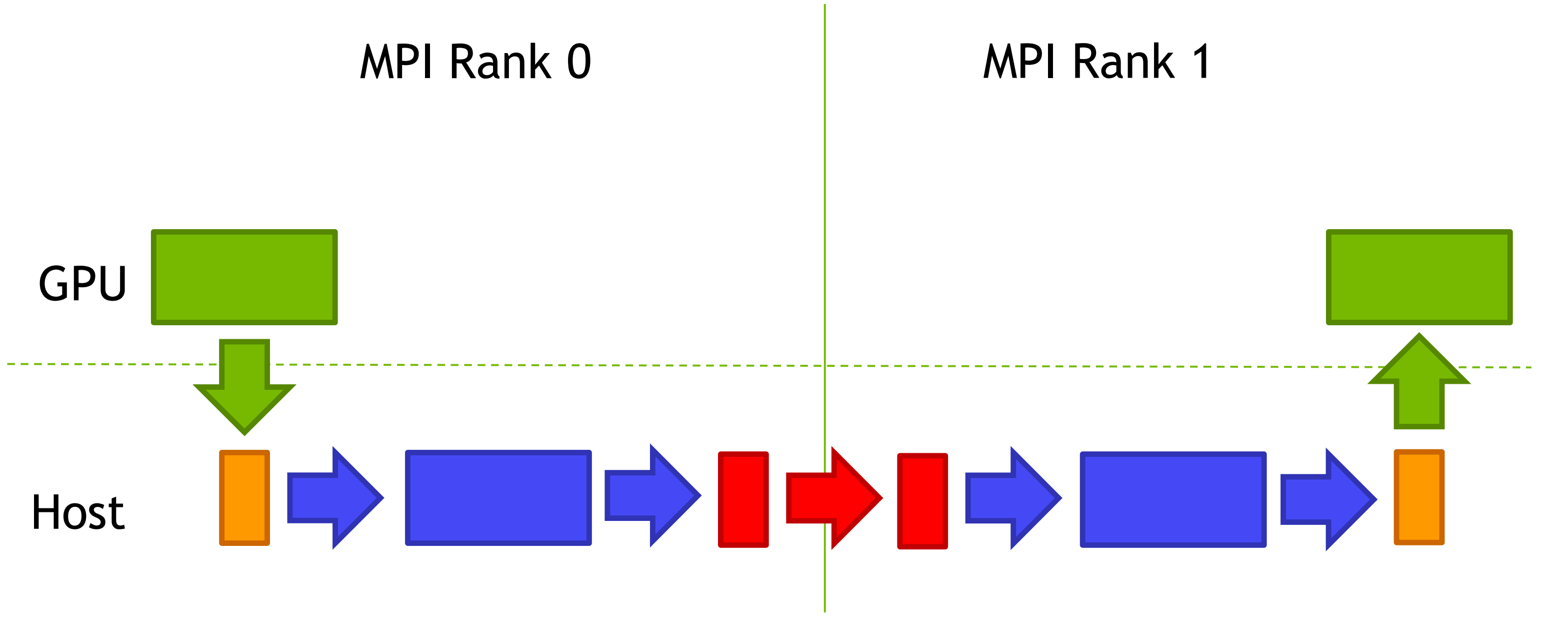
```
MPI_Send(s_buf_d,size,MPI_CHAR,1,tag,MPI_COMM_WORLD);

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

MPI GPU TO REMOTE GPU

without GPUDirect

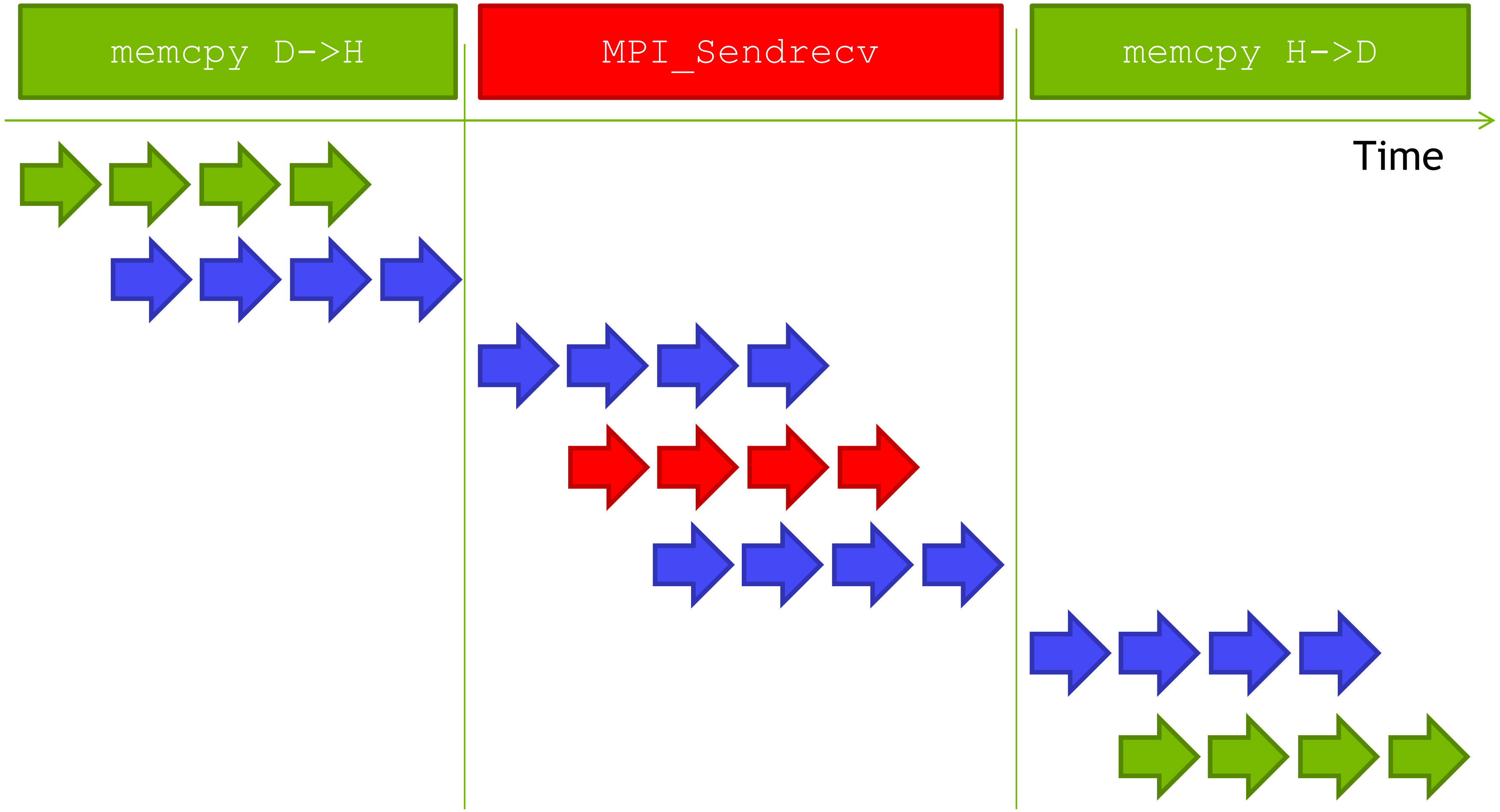


REGULAR MPI GPU TO REMOTE GPU



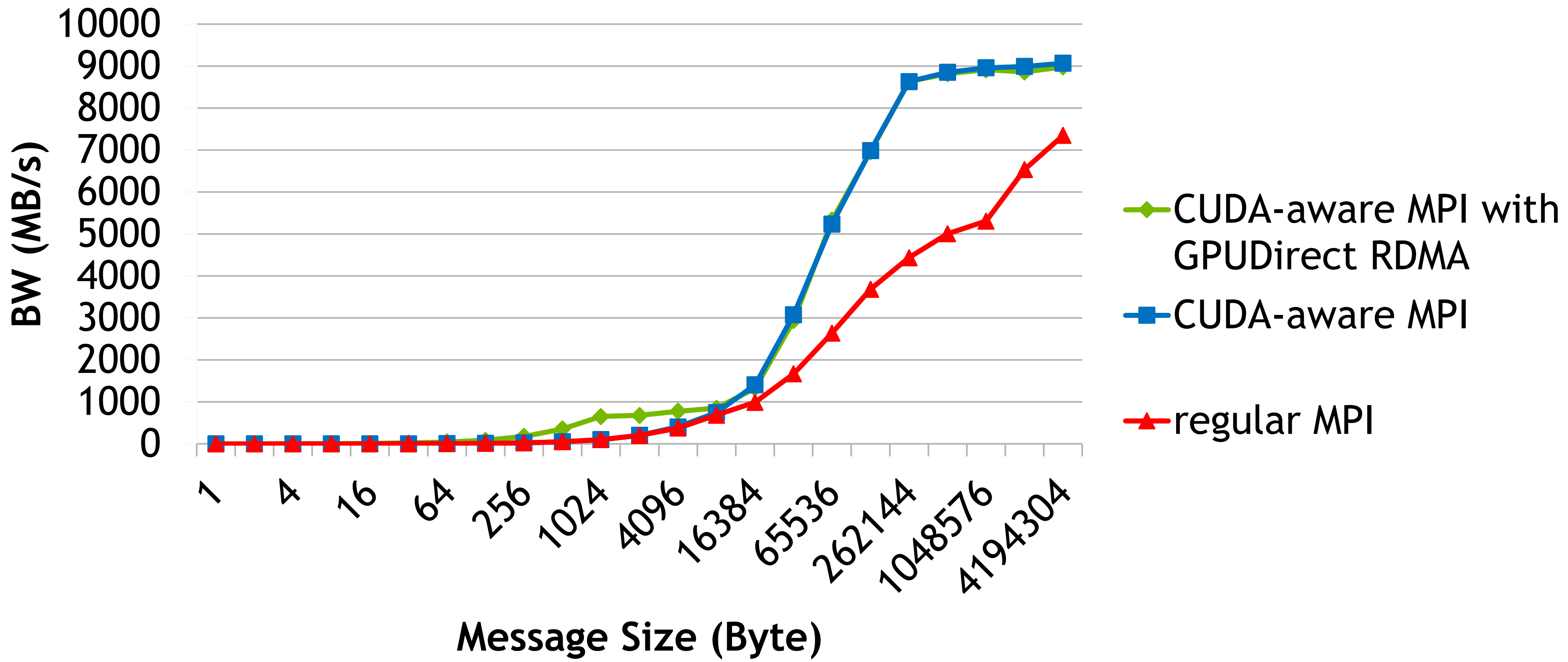
```
cudaMemcpy(s_buf_h,s_buf_d,size,cudaMemcpyDeviceToHost);
MPI_Send(s_buf_h,size,MPI_CHAR,1,tag,MPI_COMM_WORLD);

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



PERFORMANCE RESULTS GPUDIRECT RDMA

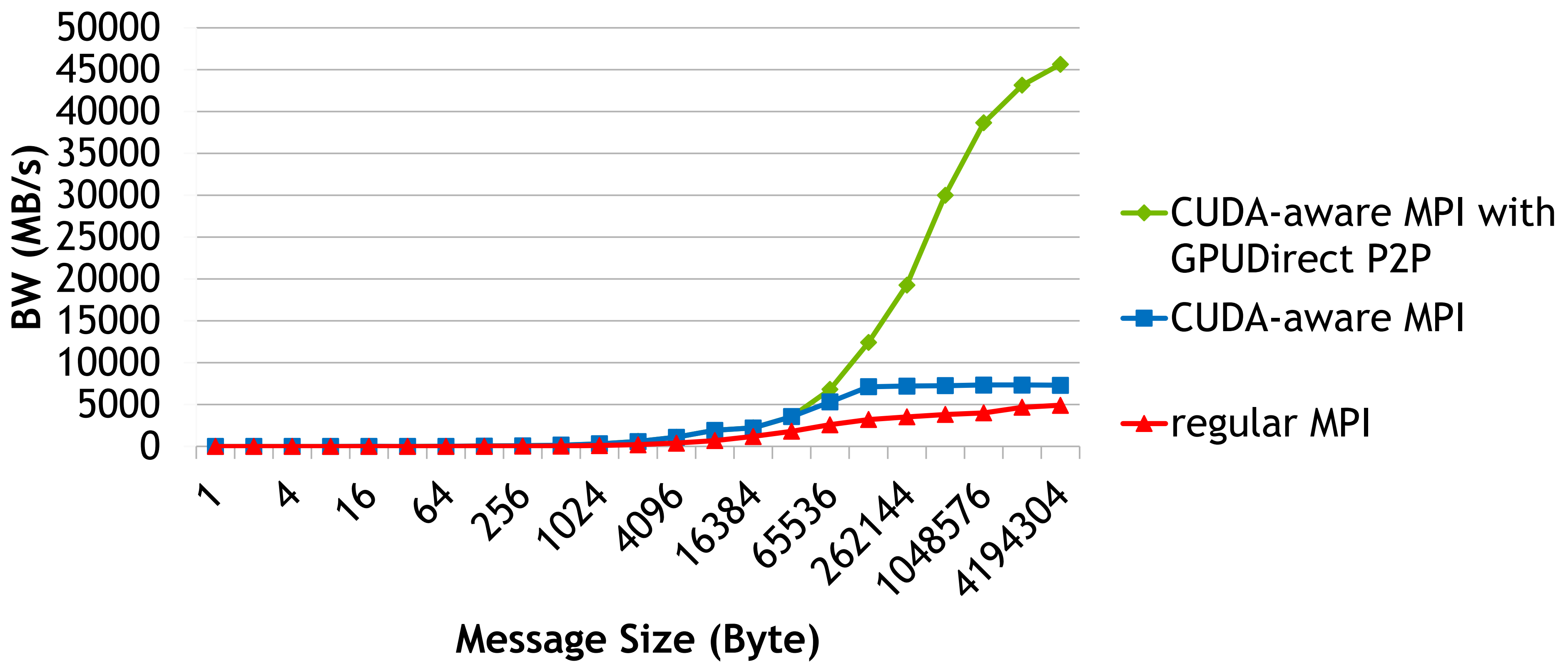
MVAPICH2-GDR 2.3a DGX-1V Tesla V100



Latency (1 Byte) 16.75 us 18.68 us 3.25 us

PERFORMANCE RESULTS GPUDIRECT P2P

MVAPICH2-GDR 2.3a DGX-1V Tesla V100



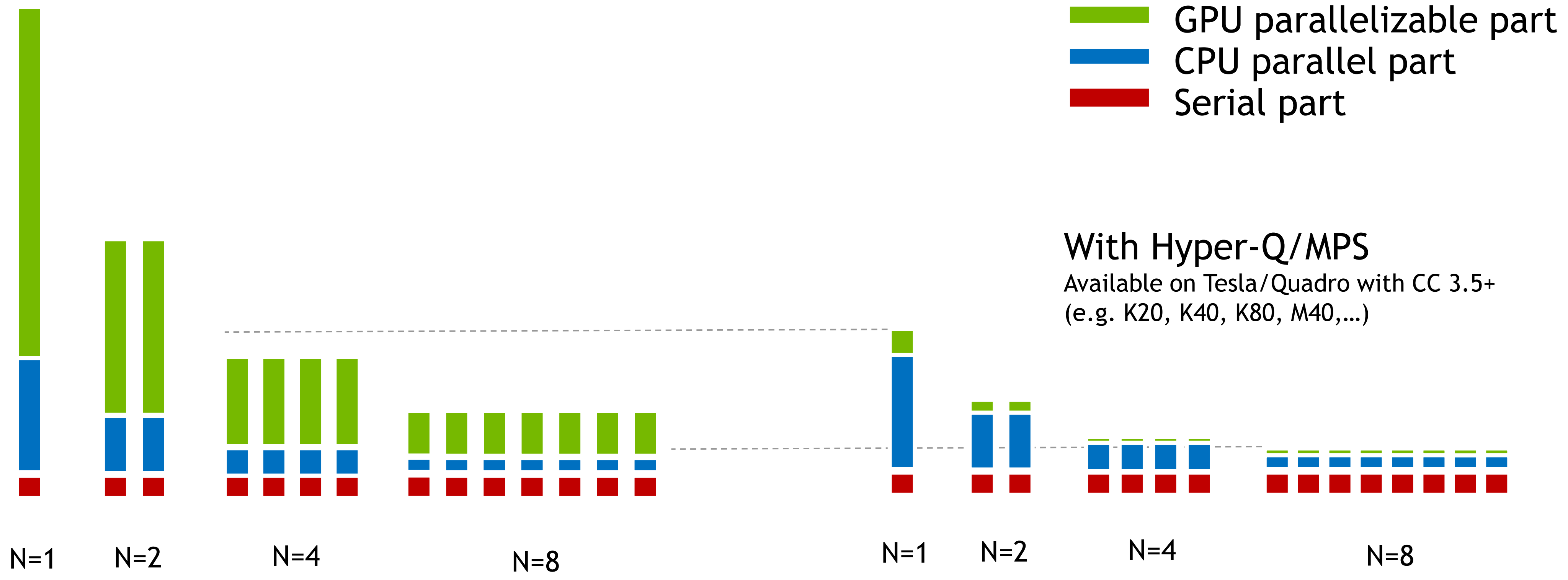
MULTI PROCESS SERVICE (MPS) FOR MPI APPLICATIONS

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



With Hyper-Q/MPS

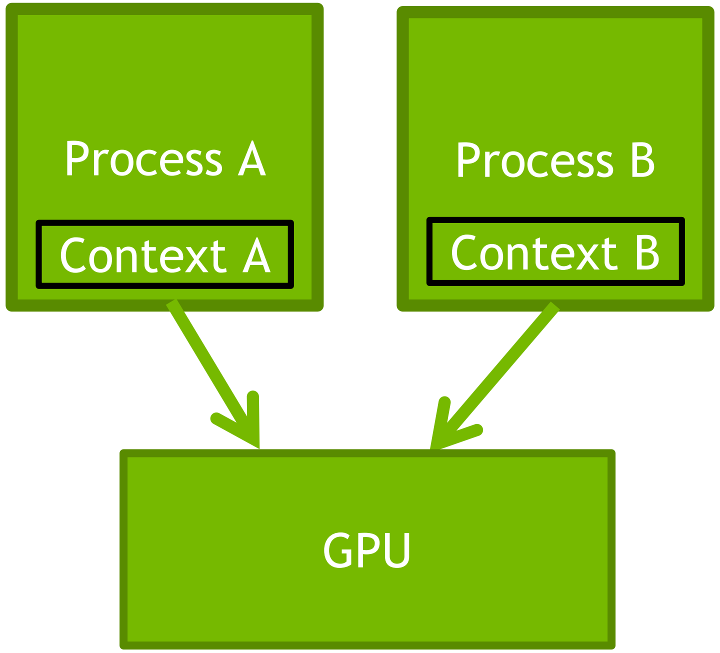
Available on Tesla/Quadro with CC 3.5+ (e.g. K20, K40, K80, M40,…)

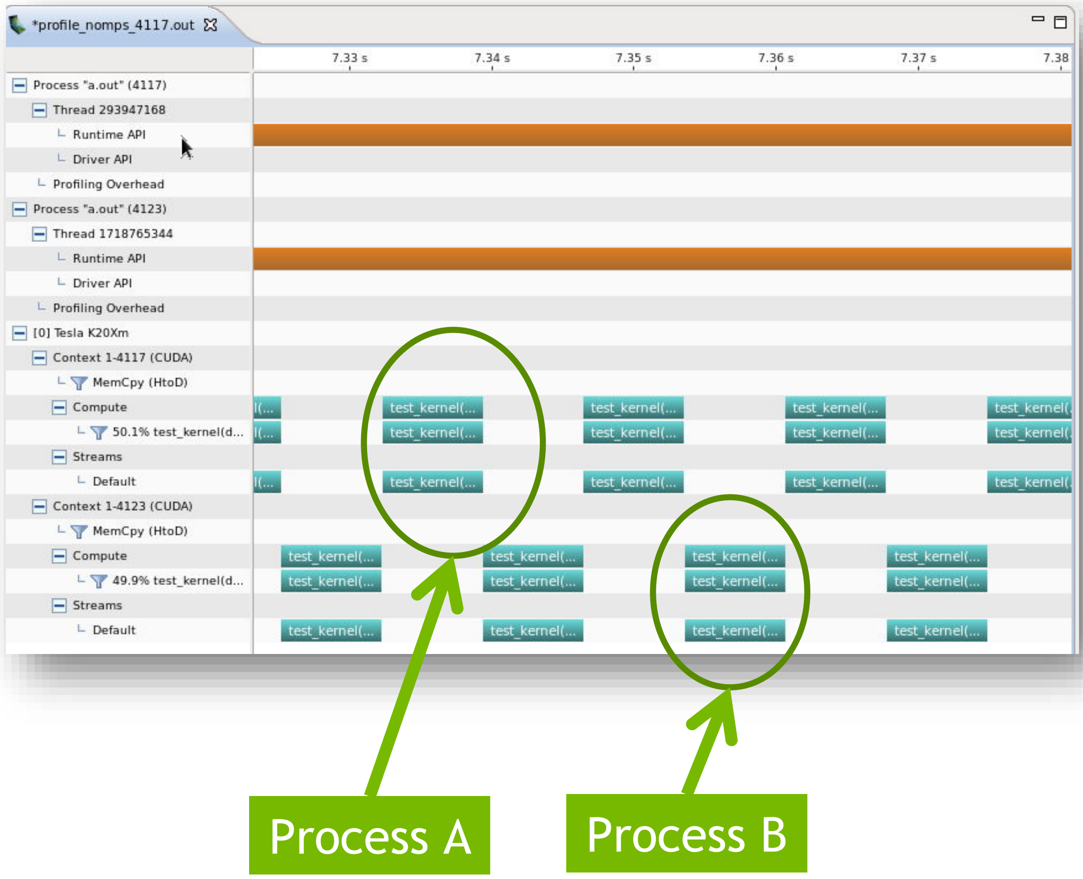
Multicore CPU only

GPU-accelerated

PROCESSES SHARING GPU WITHOUT MPS

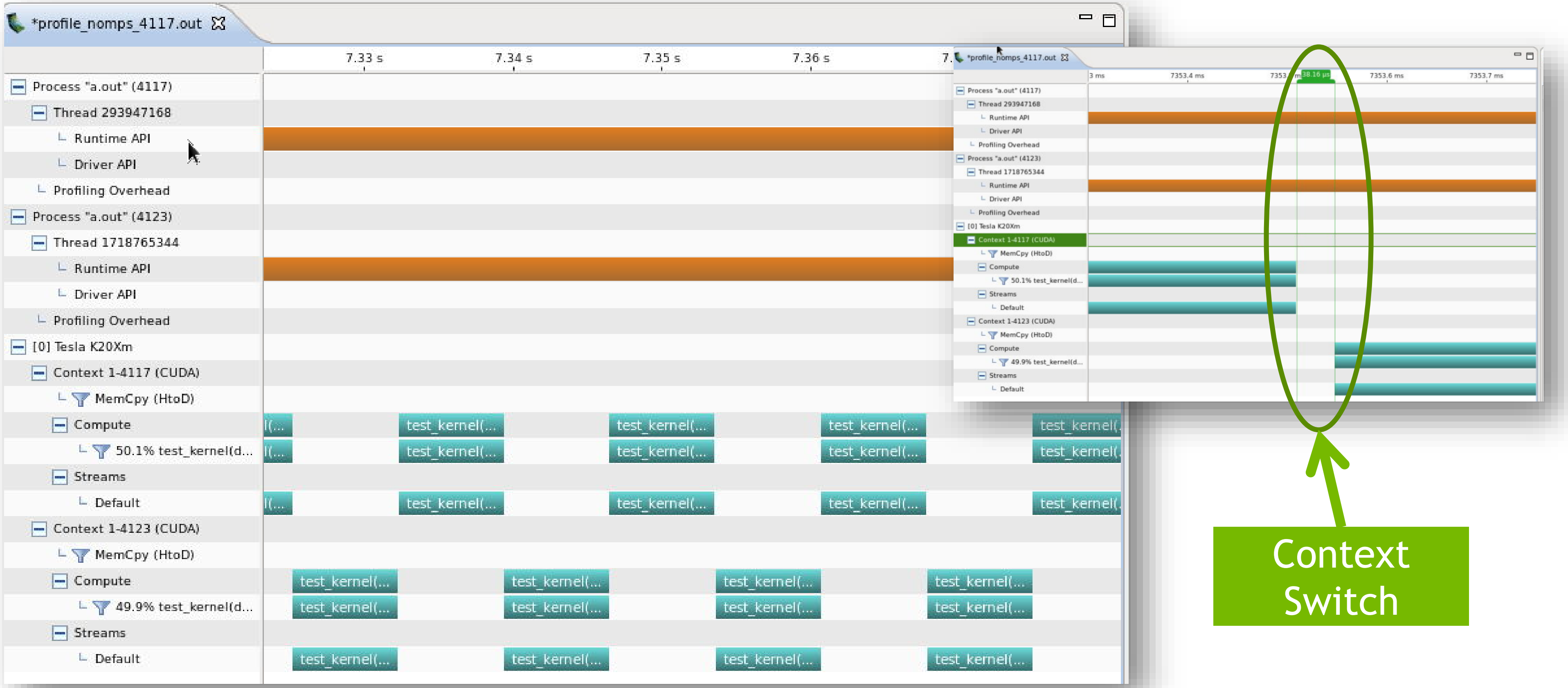
No Overlap





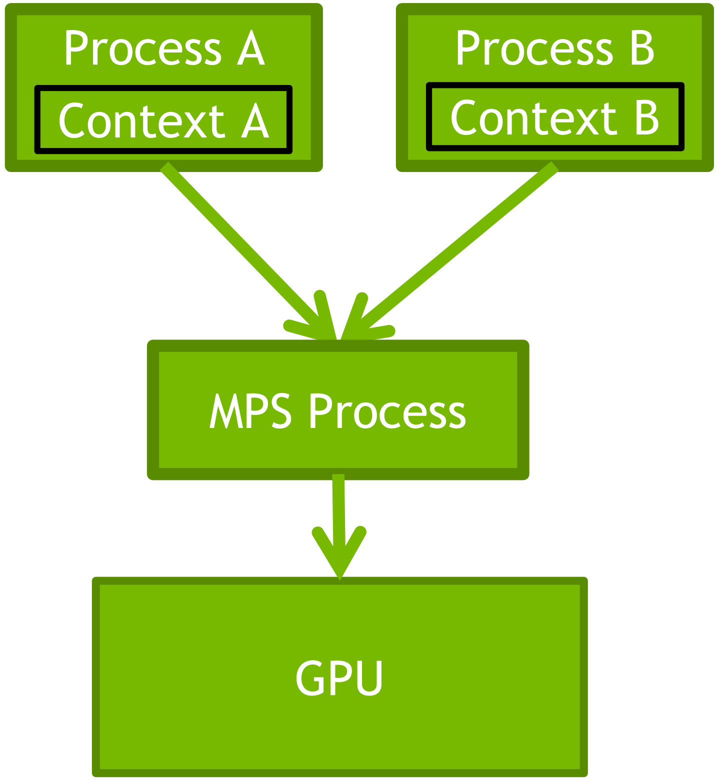
PROCESSES SHARING GPU WITHOUT MPS

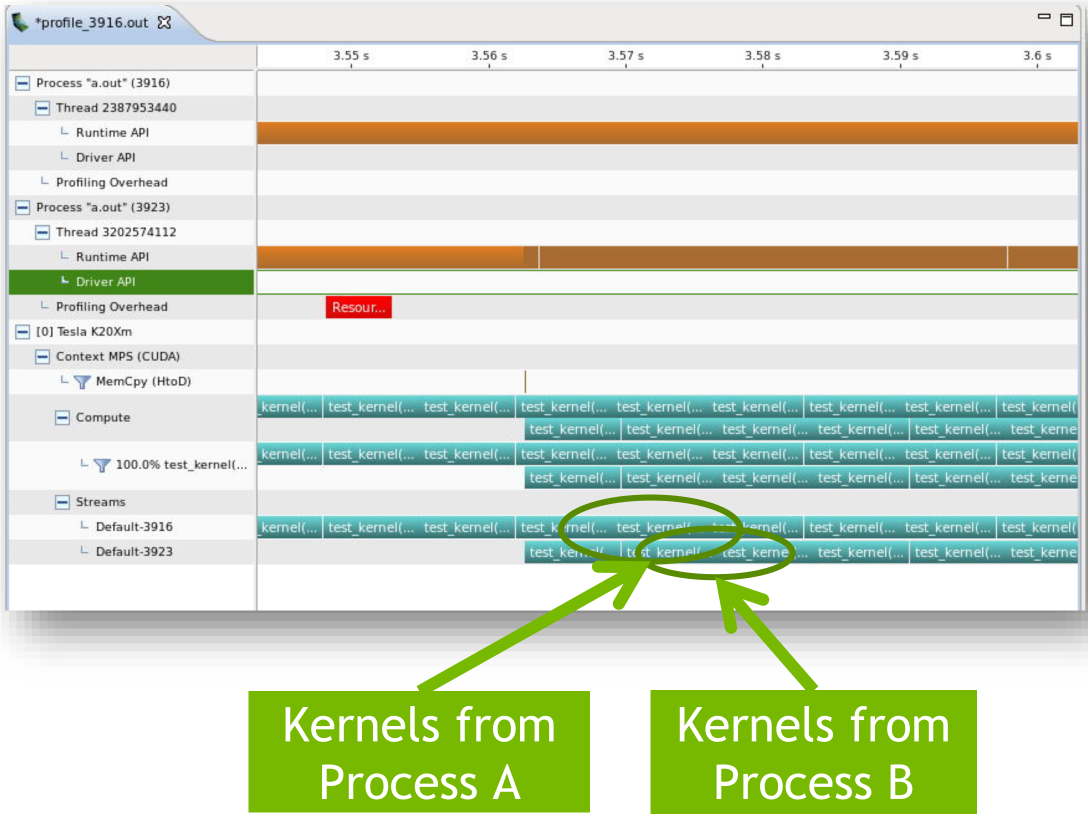
Context Switch Overhead



PROCESSES SHARING GPU WITH MPS

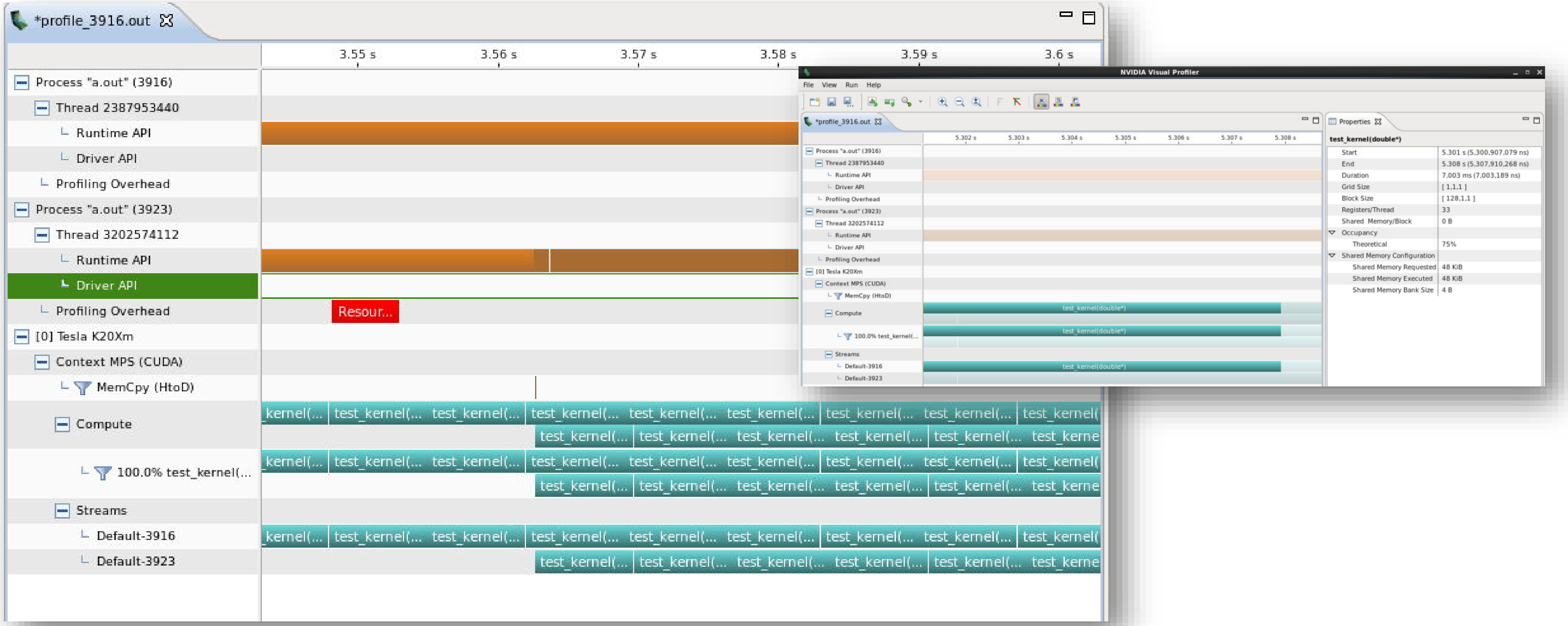
Maximum Overlap





PROCESSES SHARING GPU WITH MPS

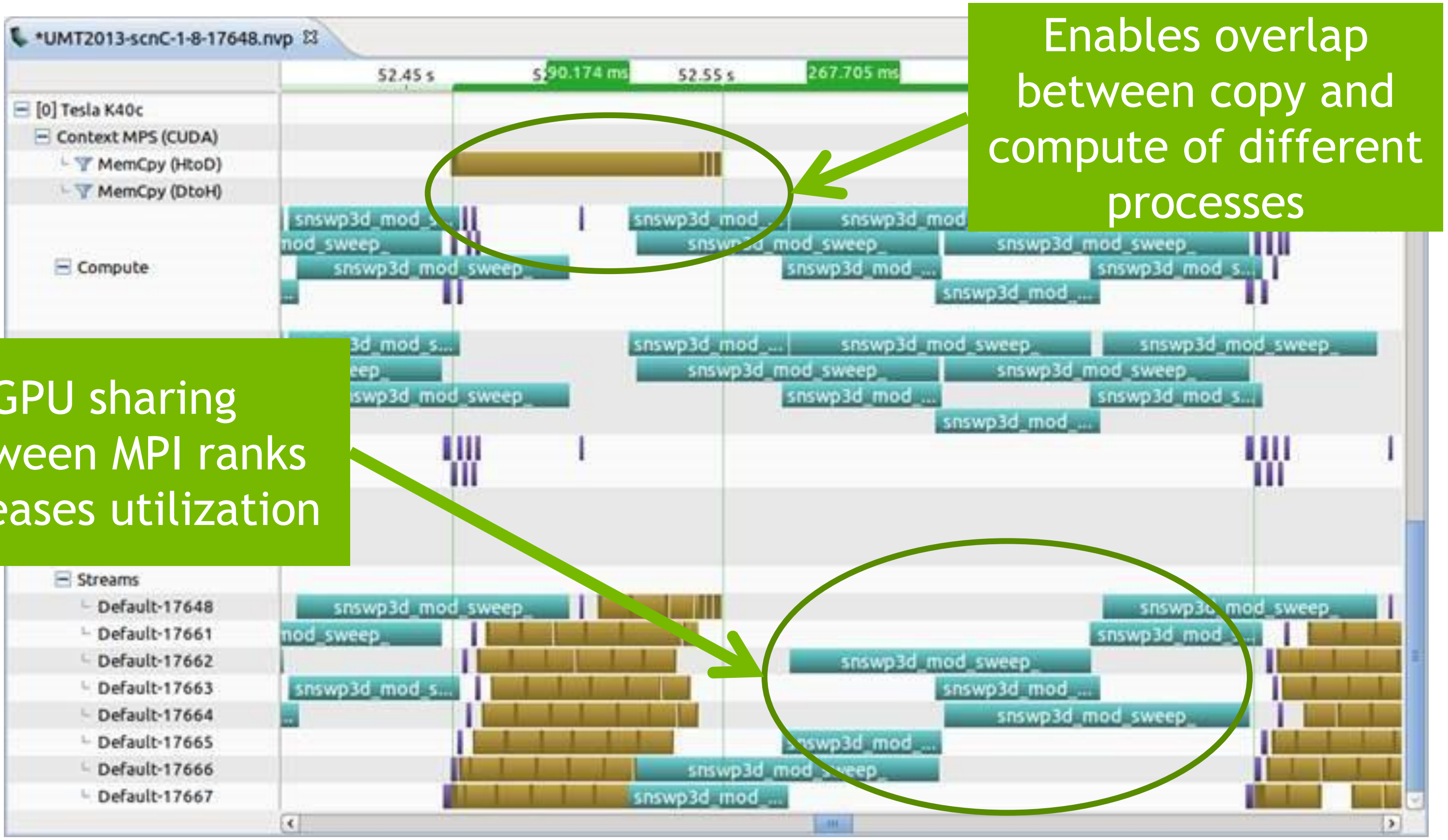
No Context Switch Overhead



test_kernel(double*)

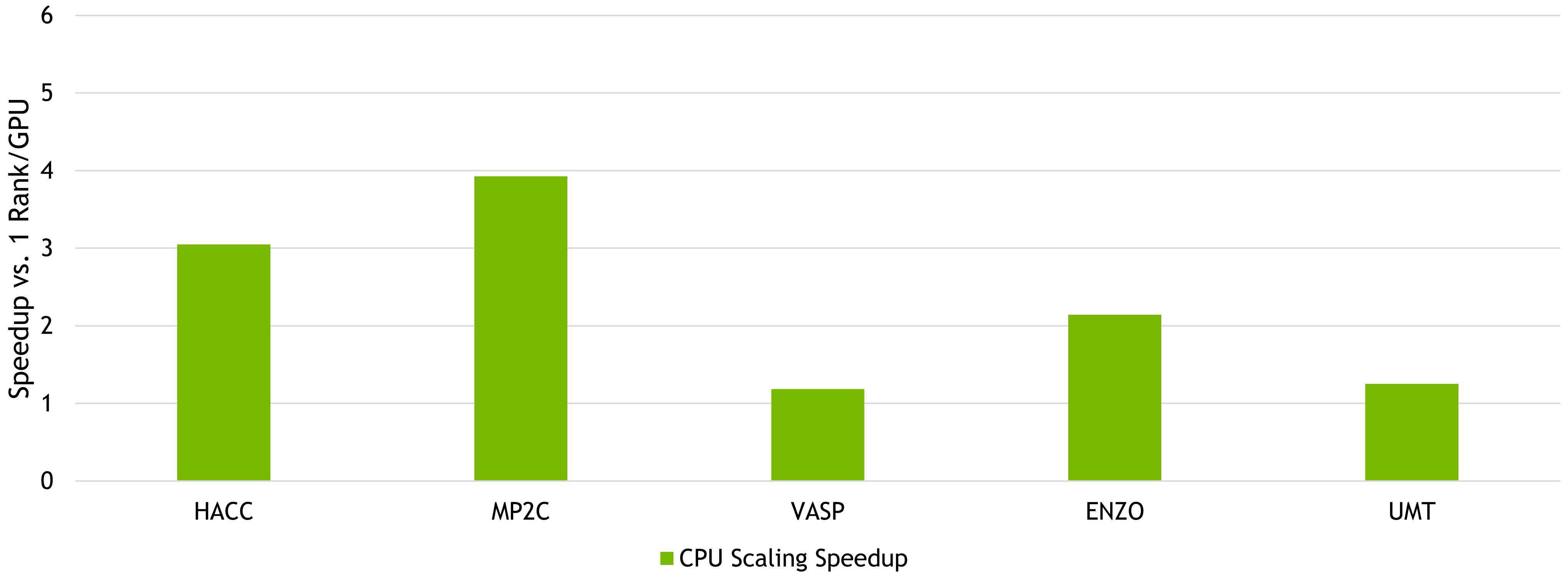
Start	5.301 s (5,300,907,079 ns)
End	5.308 s (5,307,910,268 ns)
Duration	7.003 ms (7,003,189 ns)
Grid Size	[1,1,1]
Block Size	[128,1,1]
Registers/Thread	33
Shared Memory/Block	0 B
Occupancy	
Theoretical	75%
Shared Memory Configuration	
Shared Memory Requested	48 KiB
Shared Memory Executed	48 KiB
Shared Memory Bank Size	4 B

HYPER-Q/MPS CASE STUDY: UMT



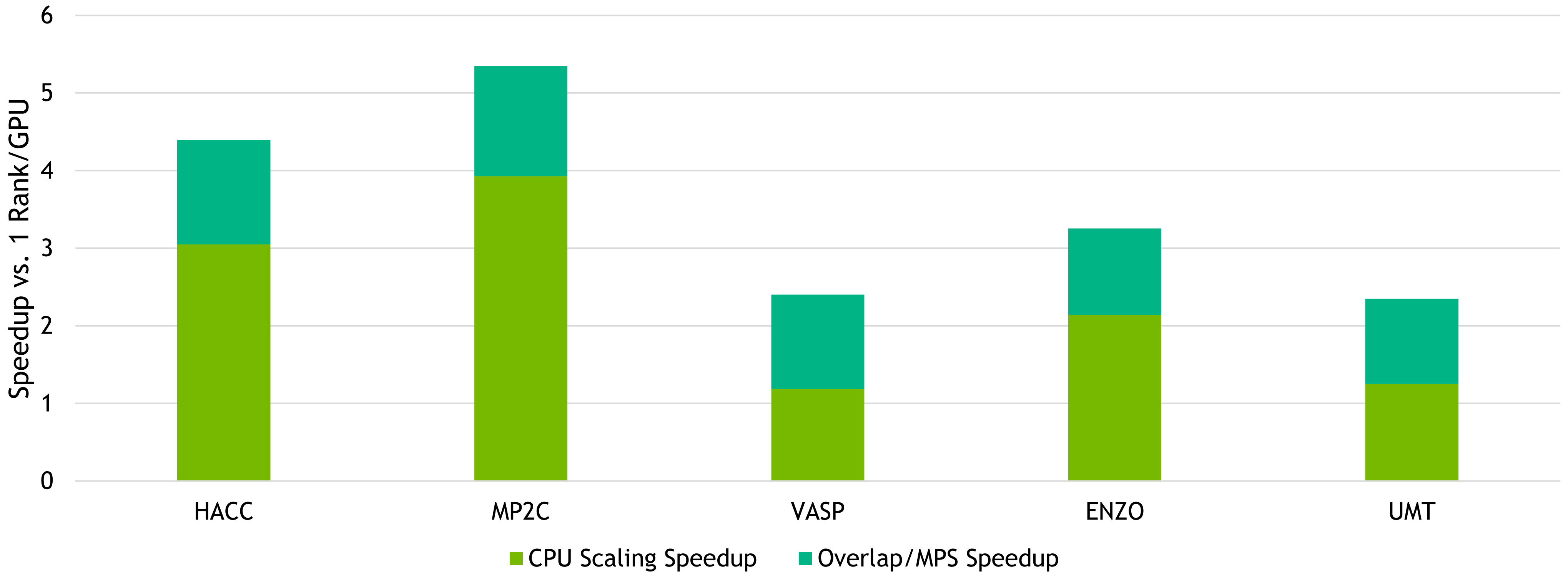
HYPER-Q/MPS CASE STUDIES

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
```

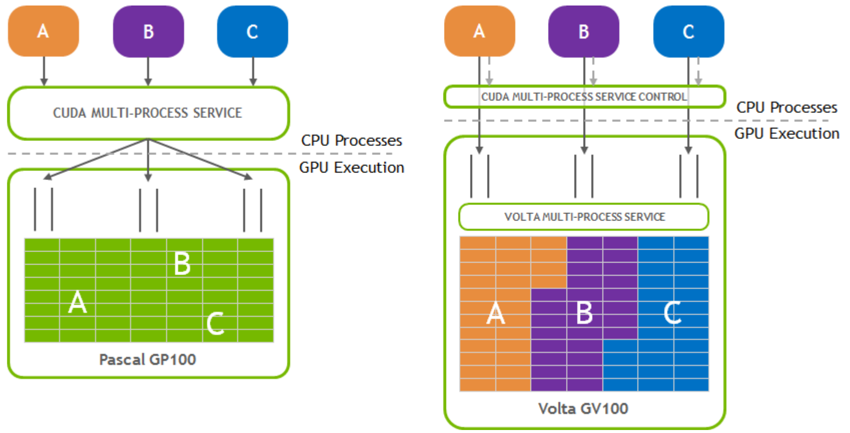
MPS: IMPROVEMENTS WITH VOLTA

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`

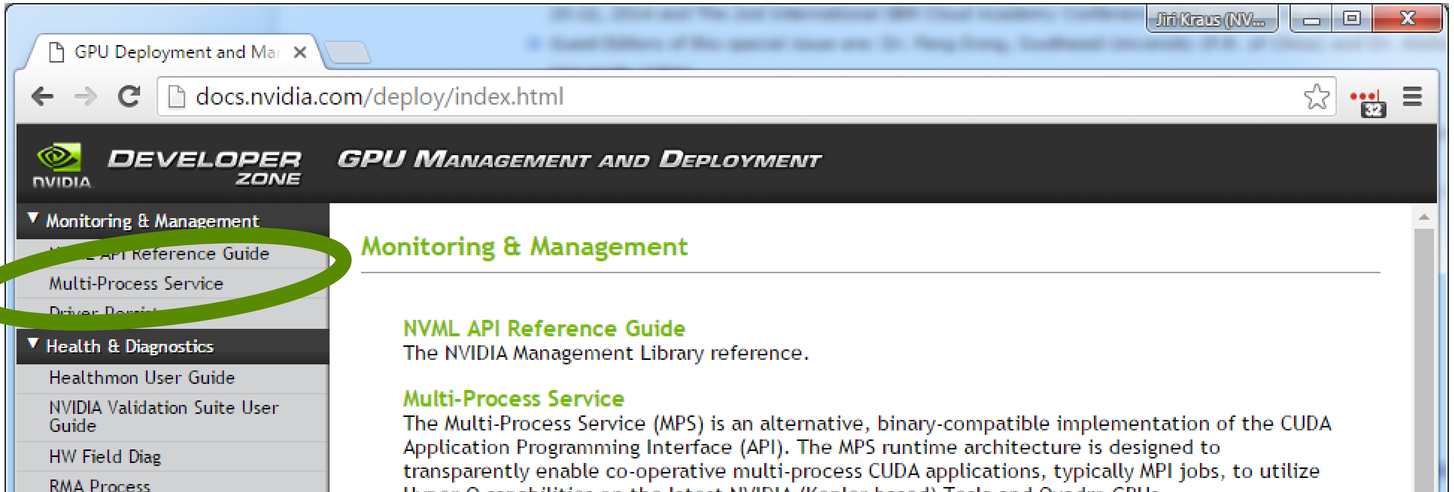

A
B
C
CUDA MULTI-PROCESS SERVICE
CPU Processes
GPU Execution
A
B
C
Pascal GP100
A
B
C
CUDA MULTI-PROCESS SERVICE CONTROL
CPU Processes
GPU Execution
VOLTA MULTI-PROCESS SERVICE
A
B
C
Volta GV100

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!



Monitoring & Management

NVML API Reference Guide
The NVIDIA Management Library reference.

Multi-Process Service
The Multi-Process Service (MPS) is an alternative, binary-compatible implementation of the CUDA Application Programming Interface (API). The MPS runtime architecture is designed to transparently enable co-operative multi-process CUDA applications, typically MPI jobs, to utilize

DEBUGGING AND PROFILING

TOOLS FOR MPI+CUDA 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

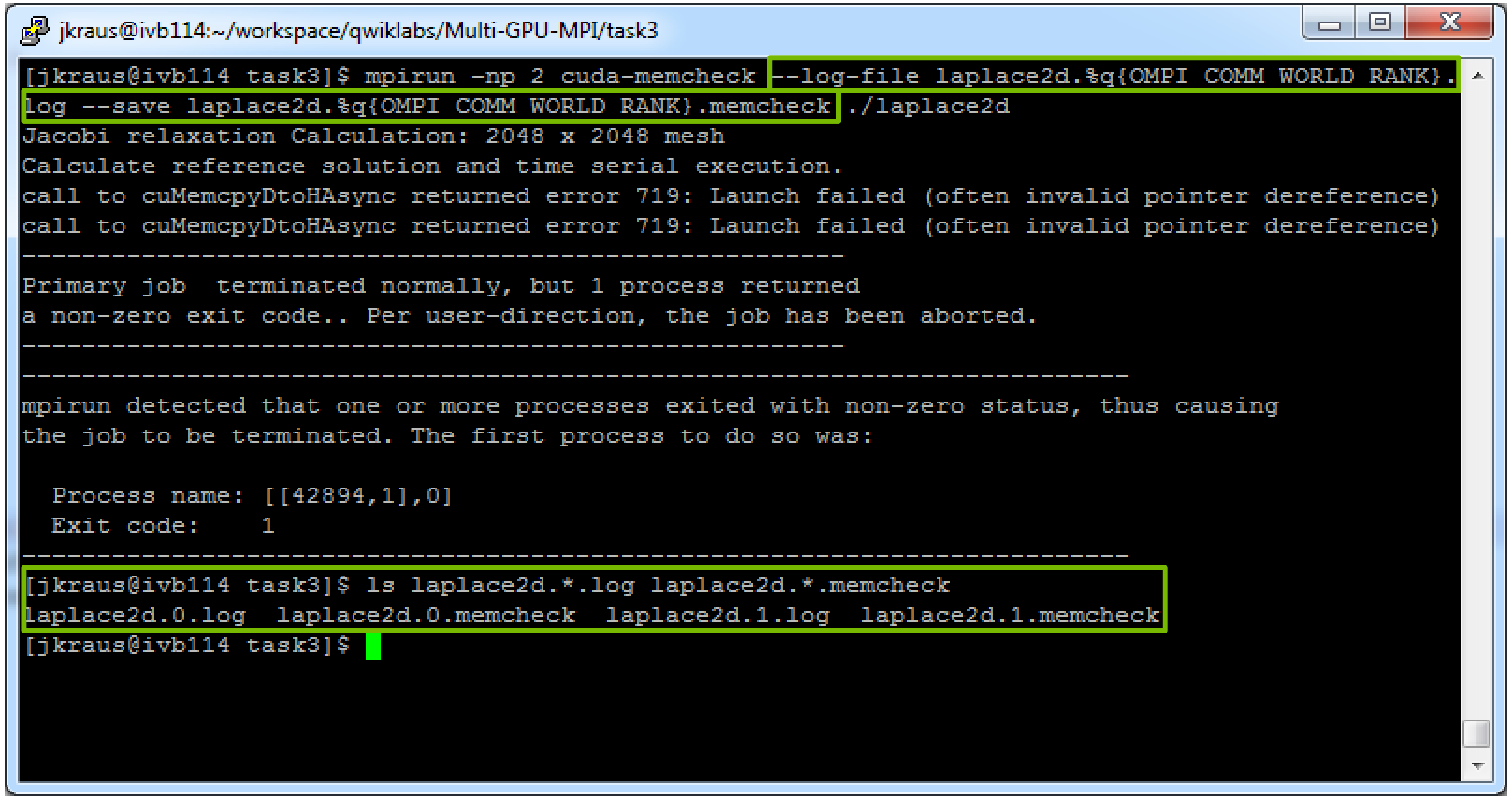
Solution: Use save or log-file command line options

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

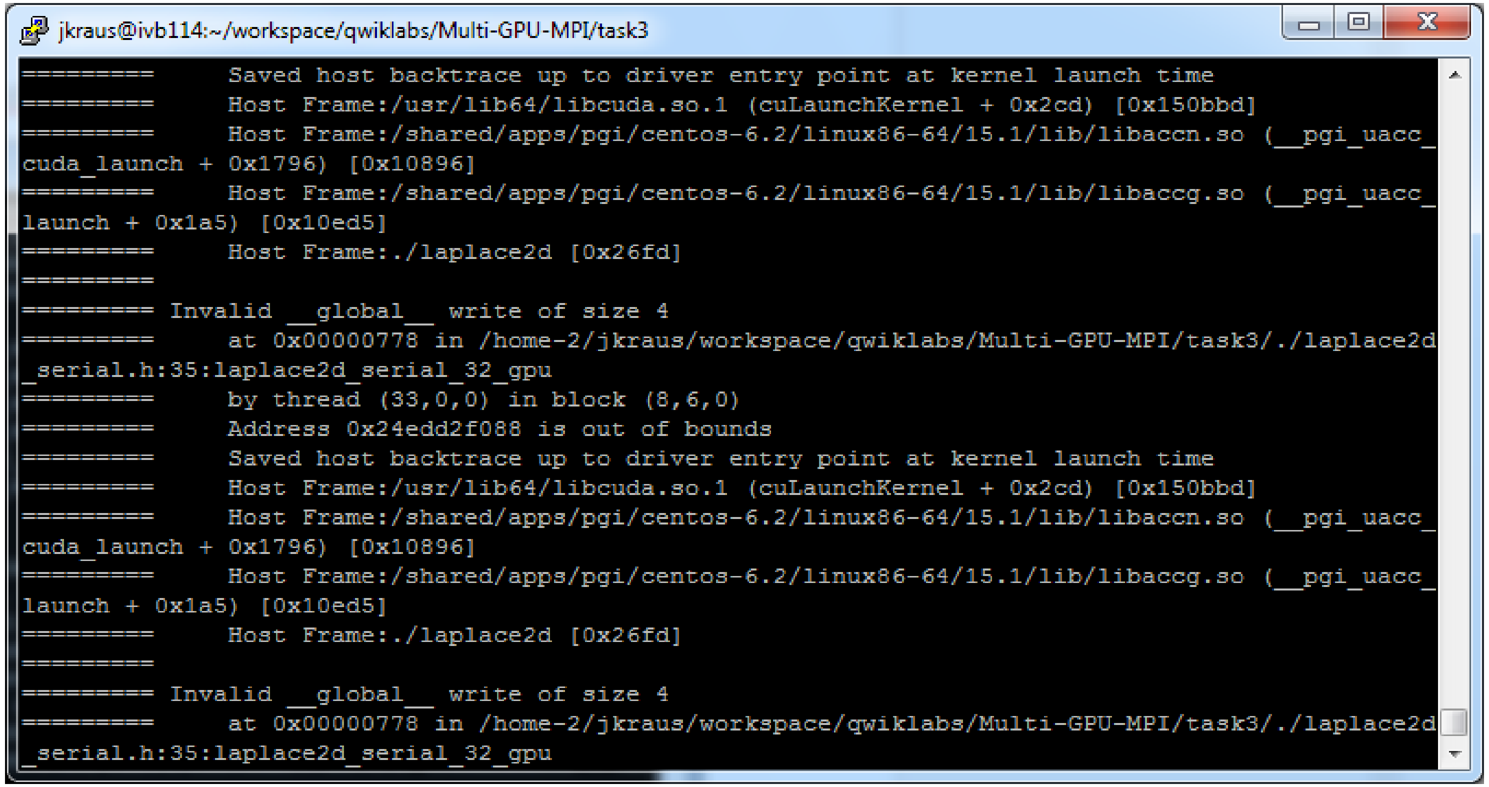


jkraus@ivb114:~/workspace/qwiklabs/Multi-GPU-MPI/task3

```
[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`



```
=========     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
```

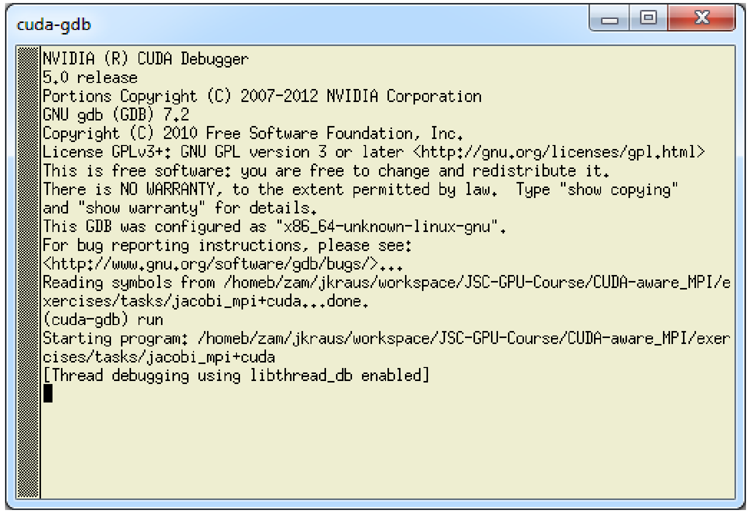
DEBUGGING MPI+CUDA APPLICATIONS

Using `cuda-gdb` with MPI Applications

Use `cuda-gdb` just like `gdb`

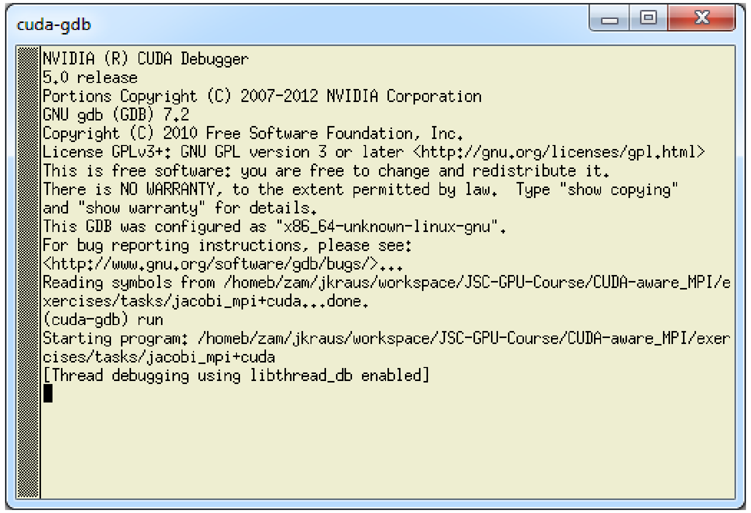
For smaller applications, just launch xterms and `cuda-gdb`

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



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/exer
cises/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/exer
cises/tasks/jacobi_mpi+cuda
[Thread debugging using libthread_db enabled]
```

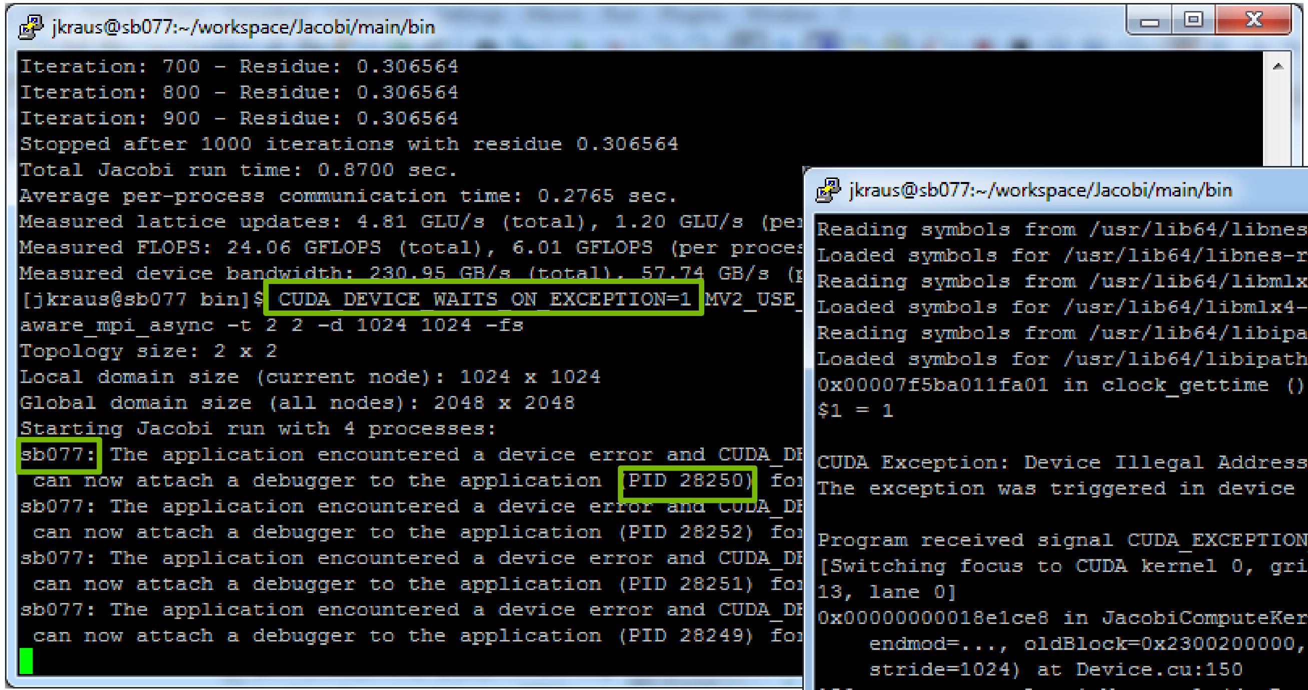
DEBUGGING MPI+CUDA APPLICATIONS

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
```

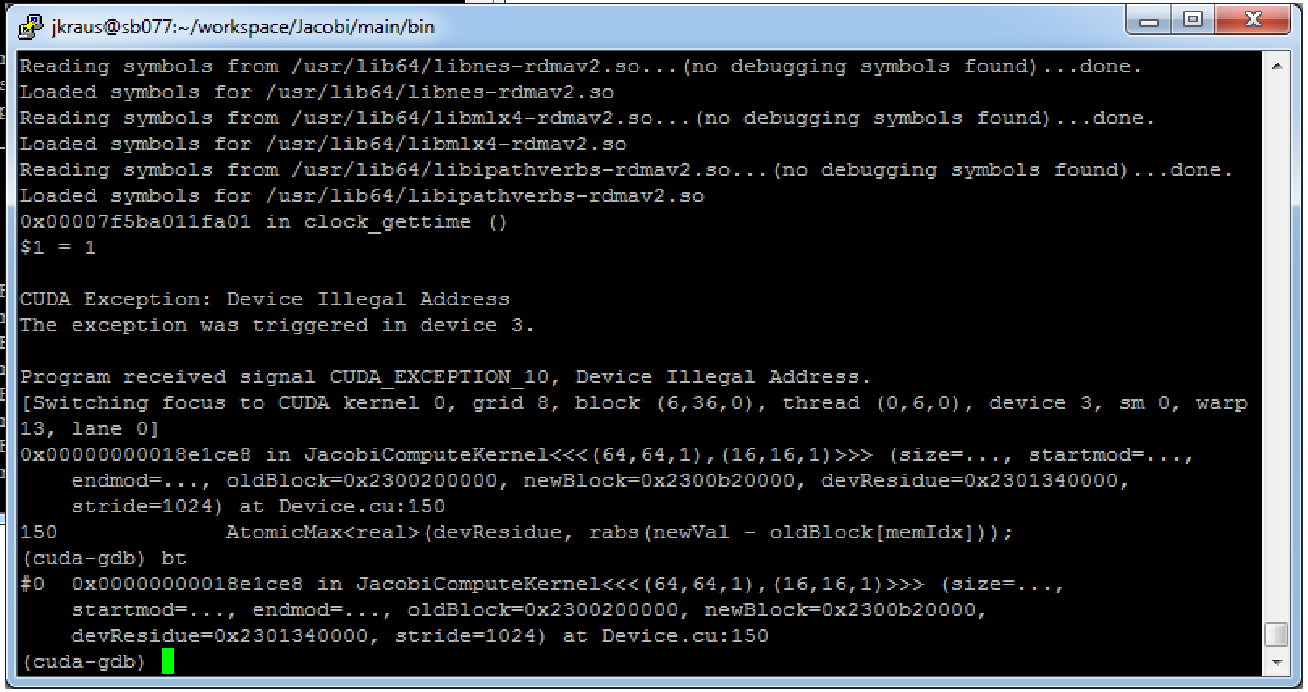
DEBUGGING MPI+CUDA APPLICATIONS

CUDA_DEVICE_WAITS_ON_EXCEPTION



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 (pe
Measured FLOPS: 24.06 GFLOPS (total), 6.01 GFLOPS (per proce
Measured device bandwidth: 230.95 GB/s (total), 57.74 GB/s (
[jkraus@sb077 bin]$ CUDA_DEVICE_WAITS_ON_EXCEPTION=1 MV2_USE_
aware_mpi_async -t 2 2 -d 1024 1024 -fs
Topology size: 2 x 2
Local domain size (current node): 1024 x 1024
Global domain size (all nodes): 2048 x 2048
Starting Jacobi run with 4 processes:
sb077: The application encountered a device error and CUDA_DE
 can now attach a debugger to the application (PID 28250) fo
sb077: The application encountered a device error and CUDA_DE
 can now attach a debugger to the application (PID 28252) fo
sb077: The application encountered a device error and CUDA_DE
 can now attach a debugger to the application (PID 28251) fo
sb077: The application encountered a device error and CUDA_DE
 can now attach a debugger to the application (PID 28249) fo
```



jkraus@sb077:~/workspace/Jacobi/main/bin

```
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+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)

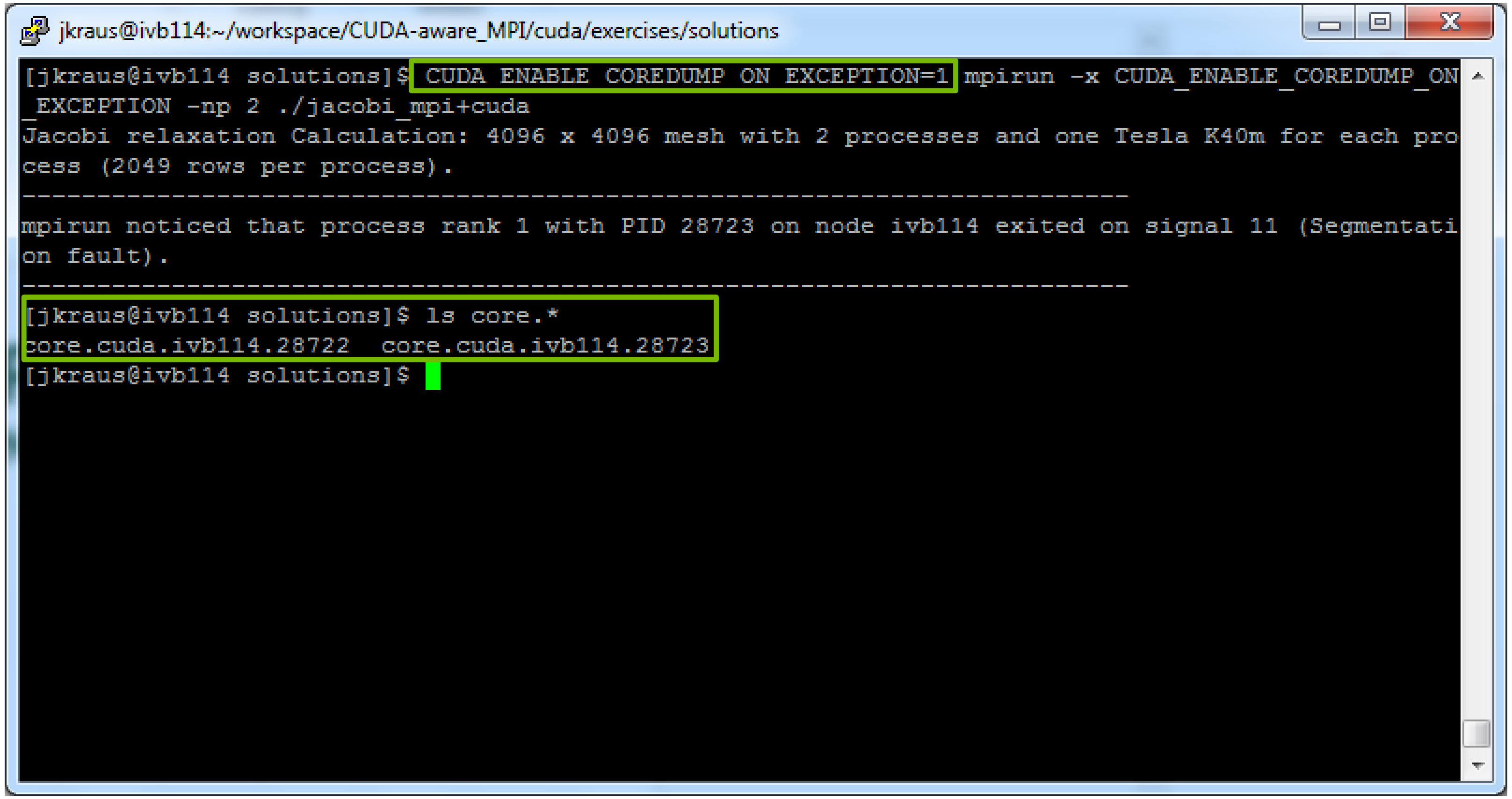
`CUDA_COREDUMP_FILE`: Specify name of core dump file

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

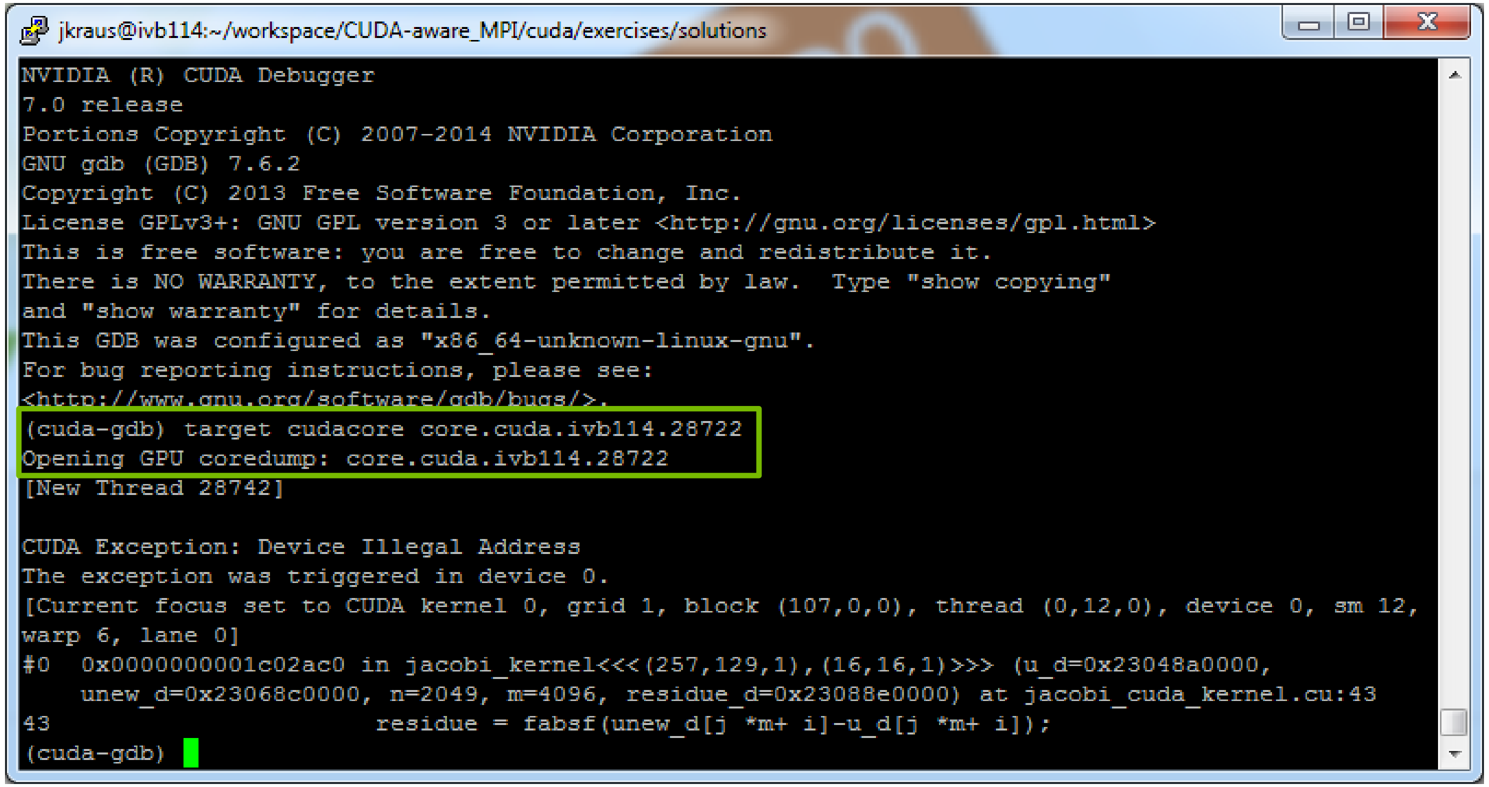


jkraus@ivb114:~/workspace/CUDA-aware_MPI/cuda/exercises/solutions

```
[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 pro
cess (2049 rows per process).
--------------------------------------------------------------------------
mpirun noticed that process rank 1 with PID 28723 on node ivb114 exited on signal 11 (Segmentati
on fault).
--------------------------------------------------------------------------
[jkraus@ivb114 solutions]$ ls core.*
core.cuda.ivb114.28722  core.cuda.ivb114.28723
[jkraus@ivb114 solutions]$
```

DEBUGGING MPI+CUDA APPLICATIONS

CUDA_ENABLE_COREDUMP_ON_EXCEPTION



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

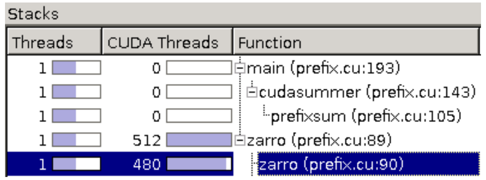
```
NVIDIA (R) CUDA Debugger
7.0 release
Portions Copyright (C) 2007-2014 NVIDIA Corporation
GNU gdb (GDB) 7.6.2
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute 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+CUDA APPLICATIONS

Third Party Tools

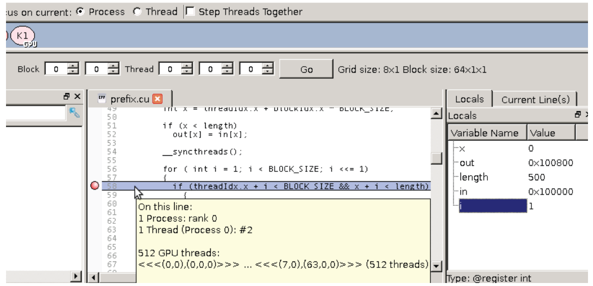
Allinea DDT debugger

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)



us on current: [✓ Process] Thread Step Threads Together

Block 0 0 Thread 0 0 0 Go Grid size: 8x1 Block size: 64x1x1

prefix.cu

```
        int x = threadIdx.x + blockIdx.x * BLOCK_SIZE;

        if (x < length)
          out[x] = in[x];

        __syncthreads();

        for ( int i = 1; i < BLOCK_SIZE; i <<= 1)
        {
          if (threadIdx.x + i < BLOCK_SIZE && x + i < length)
            {
```

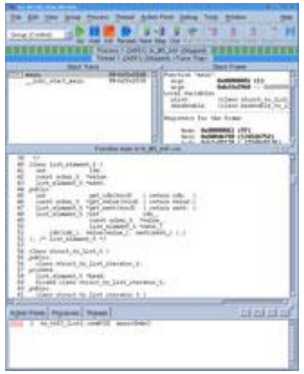
On this line:
1 Process: rank 0
1 Thread (Process 0): #2

512 GPU threads:
<<<(0,0),(0,0,0)>>> ... <<<(7,0),(63,0,0)>>> (512 threads)

Locals

Variable Name	Value
x	0
out	0x100800
length	500
in	0x100000
i	1

Type: @register int



PROFILING MPI+CUDA APPLICATIONS

Using nvprof+NVVP

New since CUDA 9

Embed MPI rank in output filename, process name, and context name (OpenMPI)

```
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}"       \
                      --annotate-mpi openmpi
```

Alternatives:

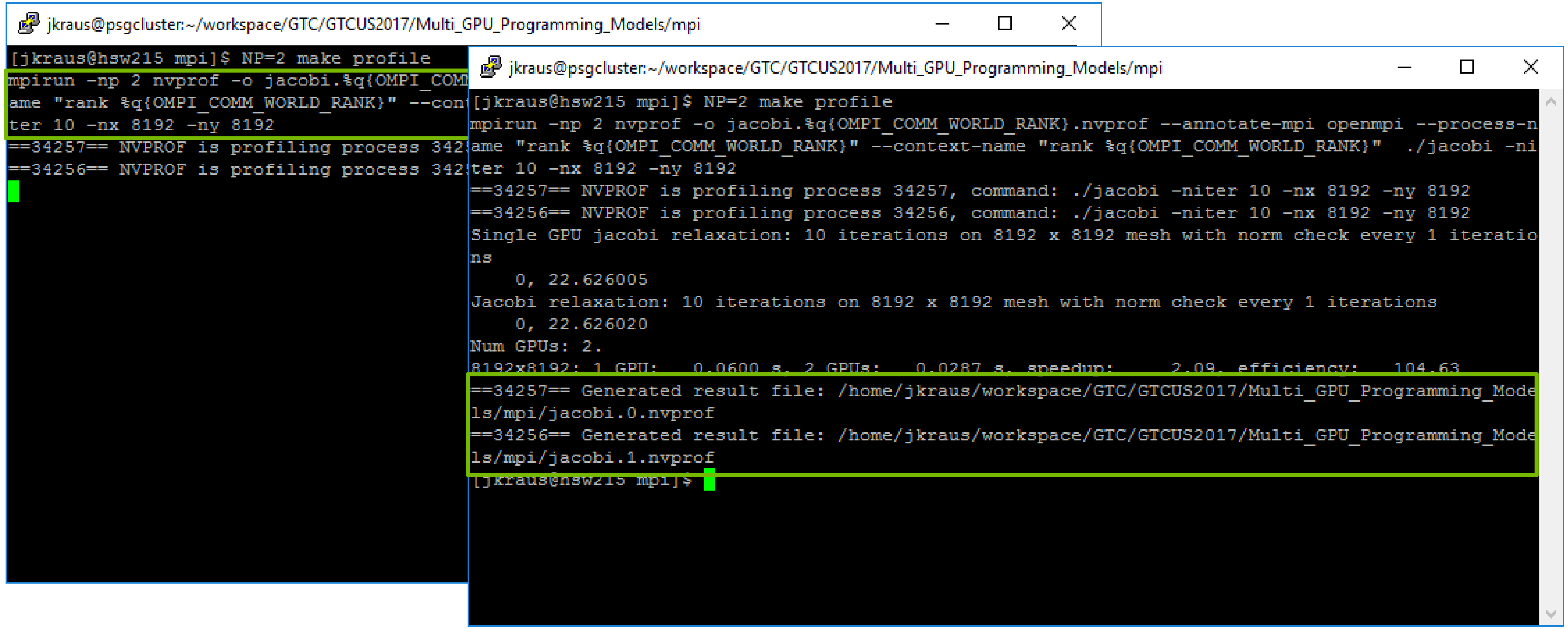
Only save the textual output (`--log-file`)

Collect data from all processes that run on a node (`--profile-all-processes`)

MVAPICH2: `MV2_COMM_WORLD_RANK`
`--annotate-mpi mpich`

PROFILING MPI+CUDA APPLICATIONS

Using `nvprof`+NVVP



jkraus@psgcluster:~/workspace/GTC/GTCUS2017/Multi_GPU_Programming_Models/mpi

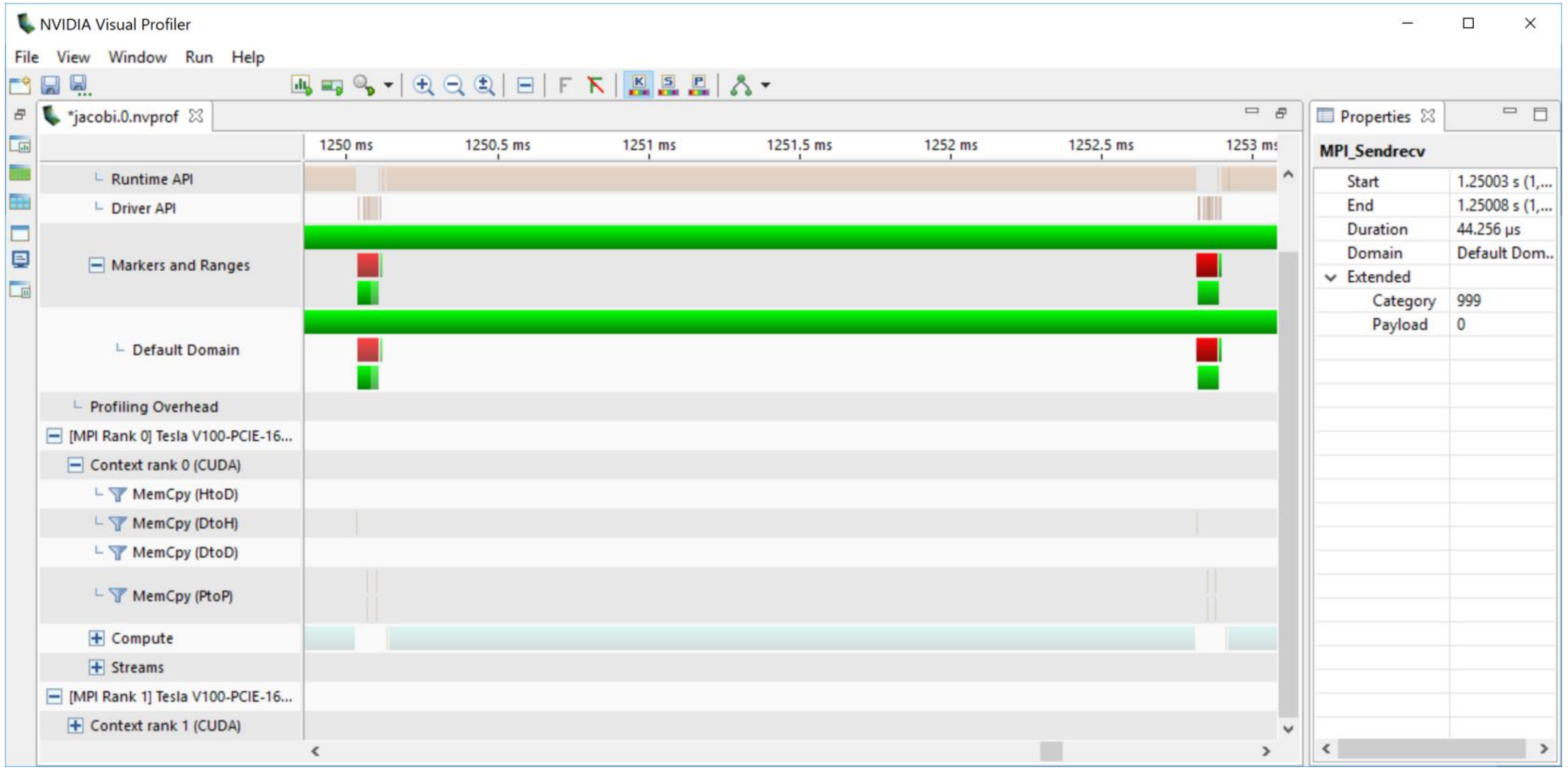
```
[jkraus@hsw215 mpi]$ NP=2 make profile
mpirun -np 2 nvprof -o jacobi.%q{OMPI_COM
ame "rank %q{OMPI_COMM_WORLD_RANK}" --con
ter 10 -nx 8192 -ny 8192
==34257== NVPROF is profiling process 342
==34256== NVPROF is profiling process 342
```

jkraus@psgcluster:~/workspace/GTC/GTCUS2017/Multi_GPU_Programming_Models/mpi

```
[jkraus@hsw215 mpi]$ NP=2 make profile
mpirun -np 2 nvprof -o jacobi.%q{OMPI_COMM_WORLD_RANK}.nvprof --annotate-mpi openmpi --process-n
ame "rank %q{OMPI_COMM_WORLD_RANK}" --context-name "rank %q{OMPI_COMM_WORLD_RANK}"  ./jacobi -ni
ter 10 -nx 8192 -ny 8192
==34257== NVPROF is profiling process 34257, command: ./jacobi -niter 10 -nx 8192 -ny 8192
==34256== NVPROF is profiling process 34256, command: ./jacobi -niter 10 -nx 8192 -ny 8192
Single GPU jacobi relaxation: 10 iterations on 8192 x 8192 mesh with norm check every 1 iteratio
ns
    0, 22.626005
Jacobi relaxation: 10 iterations on 8192 x 8192 mesh with norm check every 1 iterations
    0, 22.626020
Num GPUs: 2.
8192x8192: 1 GPU:   0.0600 s, 2 GPUs:   0.0287 s, speedup:     2.09, efficiency:   104.63
==34257== Generated result file: /home/jkraus/workspace/GTC/GTCUS2017/Multi_GPU_Programming_Mode
ls/mpi/jacobi.0.nvprof
==34256== Generated result file: /home/jkraus/workspace/GTC/GTCUS2017/Multi_GPU_Programming_Mode
ls/mpi/jacobi.1.nvprof
[jkraus@hsw215 mpi]$
```

PROFILING MPI+CUDA APPLICATIONS

Using `nvprof`+NVVP



MPI_Sendrecv

Start	1.25003 s (1,...
End	1.25008 s (1,...
Duration	44.256 μs
Domain	Default Dom...
Extended	
Category	999
Payload	0

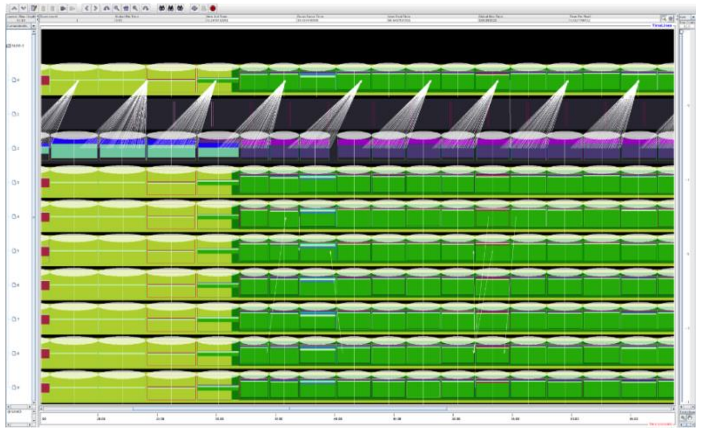
PROFILING MPI+CUDA APPLICATIONS

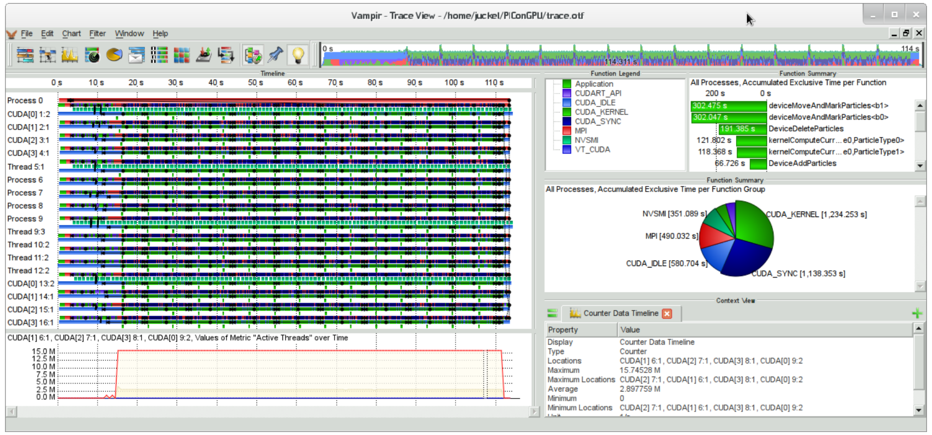
Third Party Tools

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.




Vampir - Trace View - /home/juckel/PIConGPU/trace.otf
File Edit Chart Filter Window Help
Timeline
Function Legend
Application
CUDART_API
CUDA_IDLE
CUDA_KERNEL
CUDA_SYNC
MPI
NVSMI
VT_CUDA
Function Summary
All Processes, Accumulated Exclusive Time per Function
deviceMoveAndMarkParticles<b1>
deviceMoveAndMarkParticles<b0>
DeviceDeleteParticles
DeviceAddParticles
All Processes, Accumulated Exclusive Time per Function Group
NVSMI [351.089 s]
MPI [490.032 s]
CUDA_IDLE [580.704 s]
CUDA_KERNEL [1,234.253 s]
CUDA_SYNC [1,138.353 s]
Context View
Counter Data Timeline
Property
Value

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);
}
```

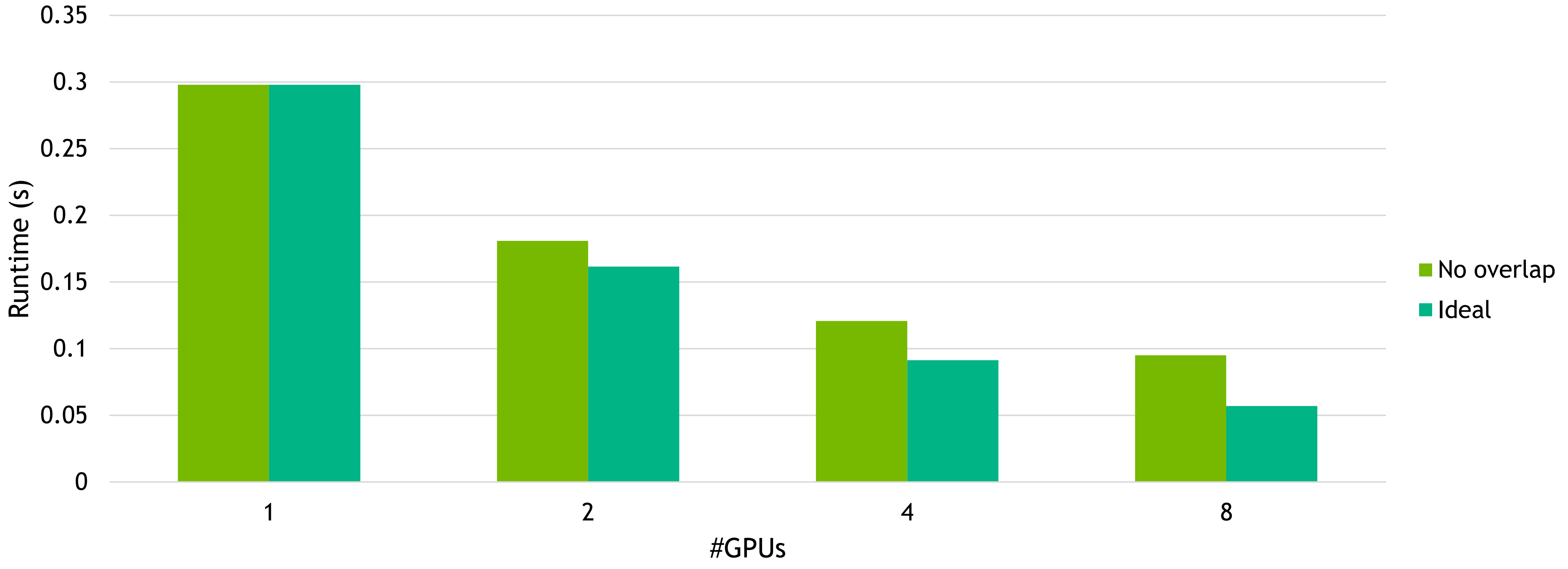
NON-BLOCKING

```
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,MP                        _req+1);
  MPI_Isend(u_new+offset_last_row,m-2,MPI_DOUBLE                     b_req+2);
  MPI_Isend(u_new+offset_first_row,m-2,MPI_DOUBLE             MM_WORLD,t_b_req+3);
}
MPI_Waitall(4, t_b_req, MPI_STATUSES_IGNORE);
```

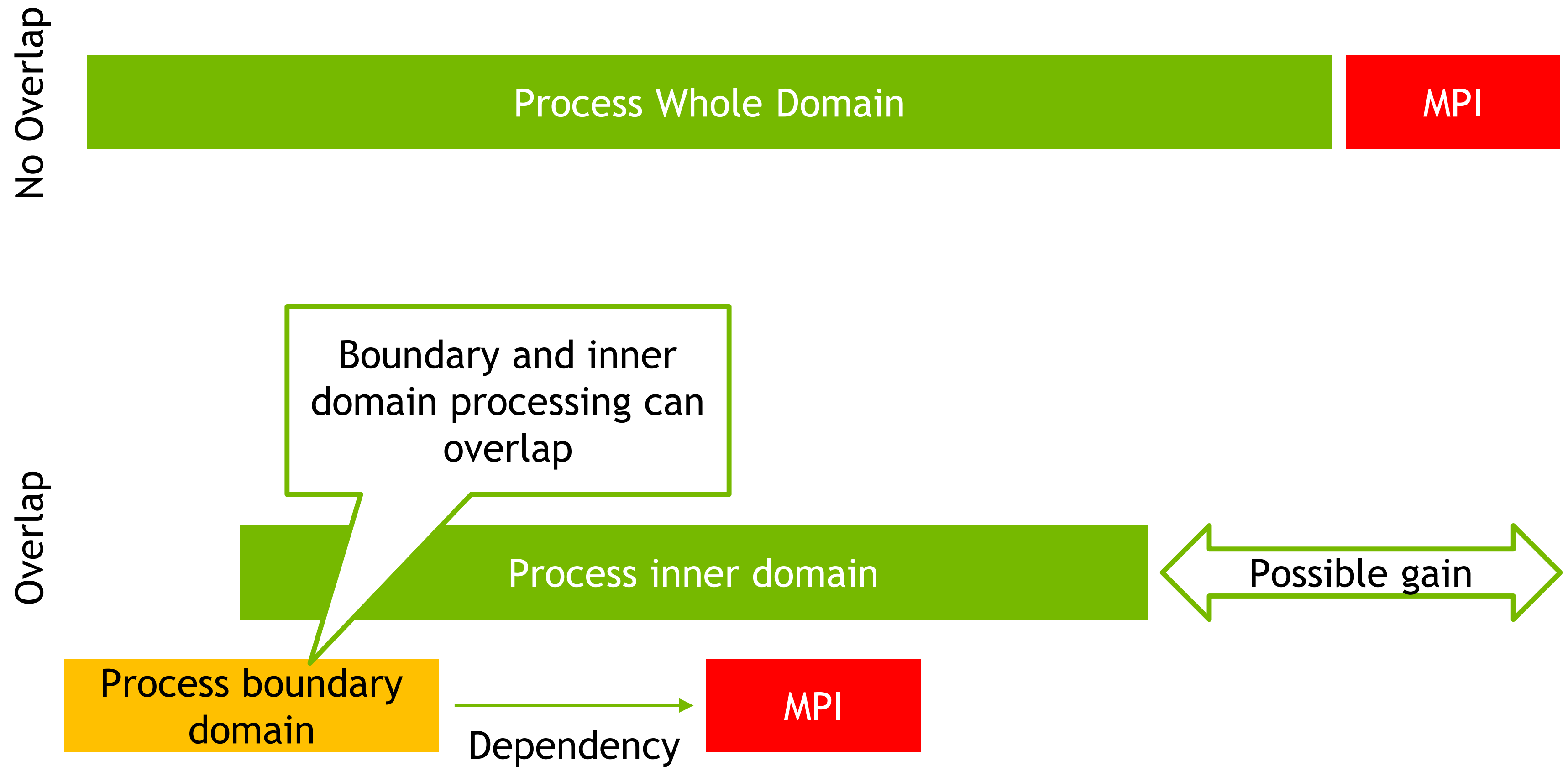
Gives MPI more opportunities to build efficient piplines

COMMUNICATION + COMPUTATION OVERLAP

OpenMPI 3.0.1RC1 – DGX-1V – 2048x2048



COMMUNICATION + COMPUTATION OVERLAP



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
```

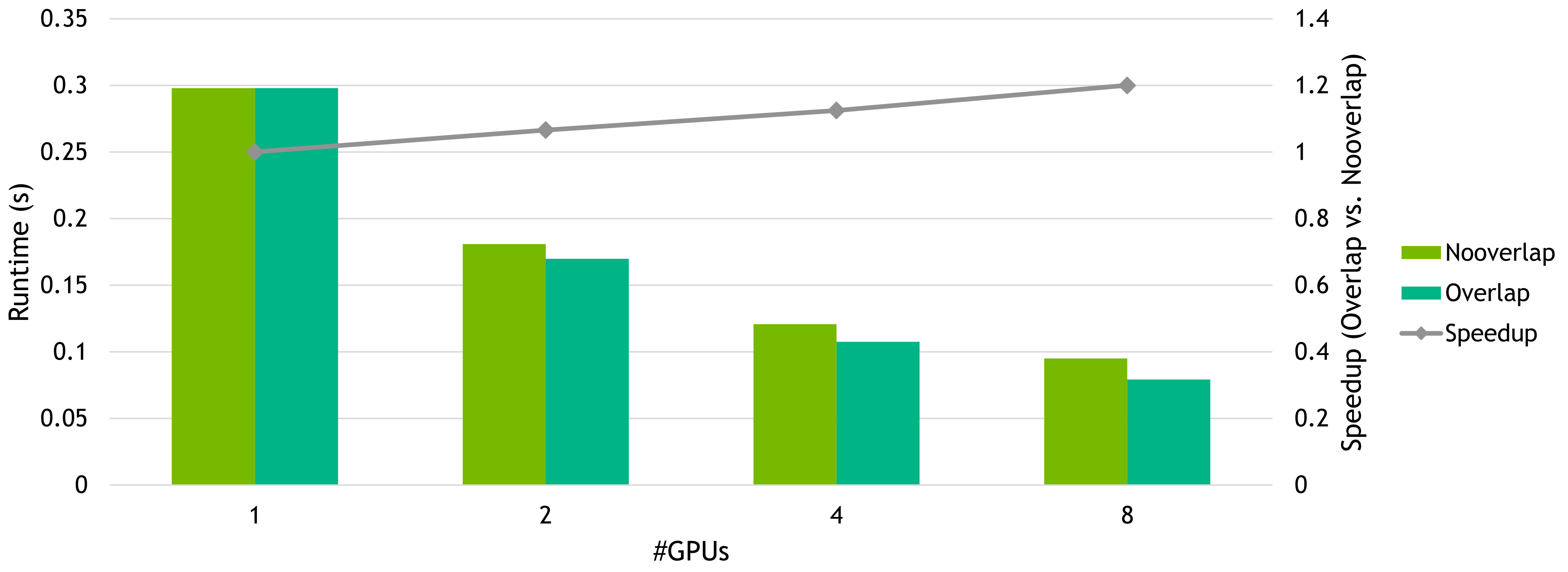
COMMUNICATION + COMPUTATION OVERLAP

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

OpenMPI 3.0.1RC1 – DGX-1V – 2048x2048

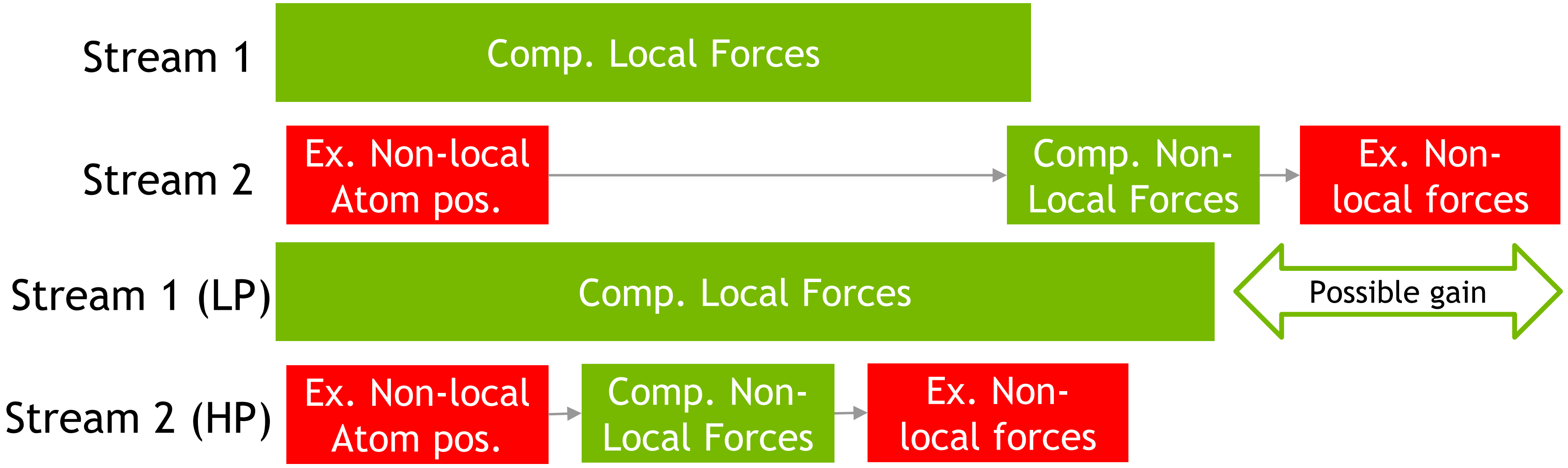


HIGH PRIORITY STREAMS

Improve scalability with high priority streams

> __host__ cudaError_t cudaStreamCreateWithPriority (cudaStream_t* pStream, unsigned int flags, int priority)

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. OpenMPI since 1.8.5 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);

}
```

Array is intended to be used on the CPU

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

MPI AND UNIFIED MEMORY

The Future with Data Usage Hints - Summary

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

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

Performance tools help to identify missing or wrong data usage hints

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

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

MPI AND UNIFIED MEMORY

Current Status

Available Unified Memory-aware MPI implementations

- OpenMPI (since 1.8.5)
- MVAPICH2-GDR (since 2.2b)
  - Performance improvements with 2.2RC1 for Intranode GPU to GPU communication

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

→

Good performance if all buffers used in MPI are touched mainly on the GPU.

MPI AND UNIFIED MEMORY

Without Unified Memory-aware MPI

Only use non Unified Memory Buffers for MPI: cudaMalloc, cudaMallocHost or malloc

Application managed non Unified Memory Buffers also allow to work around current missing cases in Unified Memory-aware MPI Implementations.

DETECTING CUDA-AWARENESS

OpenMPI (since 2.0.0):

Macro:

```
MPIX_CUDA_AWARE_SUPPORT
```

Function for runtime decisions

```
MPIX_Query_cuda_support()
```

Include `mpi-ext.h` for both.

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