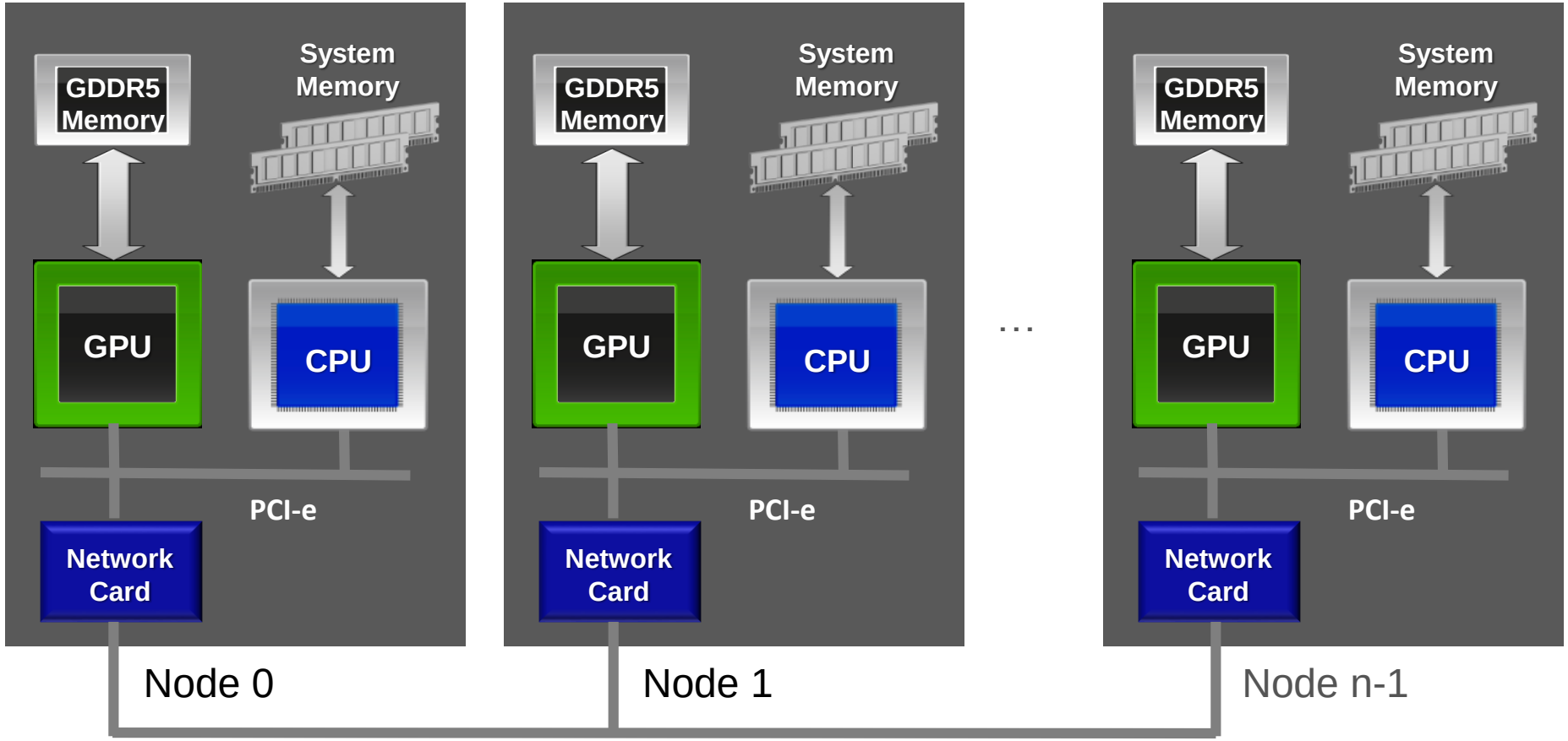
GPU TECHNOLOGY CONFERENCE

NVIDIA

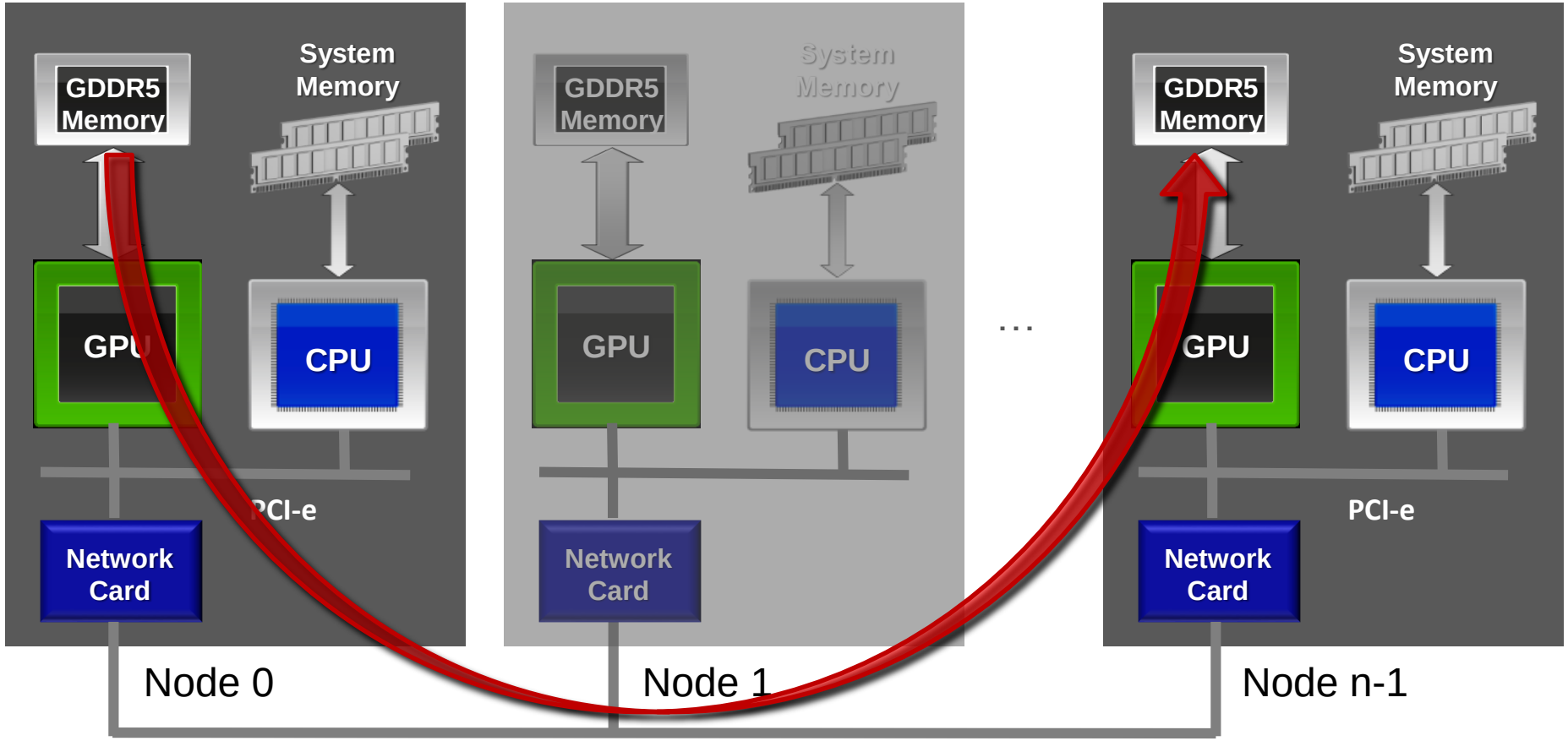
MULTI GPU PROGRAMMING WITH MPI

Jiri Kraus and Peter Messmer, NVIDIA

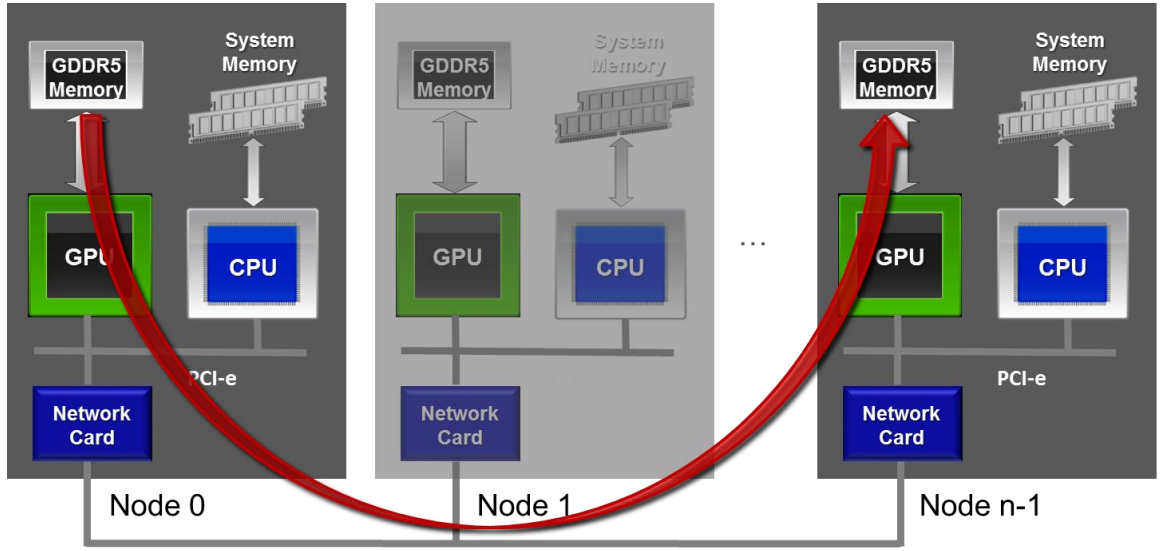
MPI+CUDA



MPI+CUDA


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

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);
```

WHAT 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

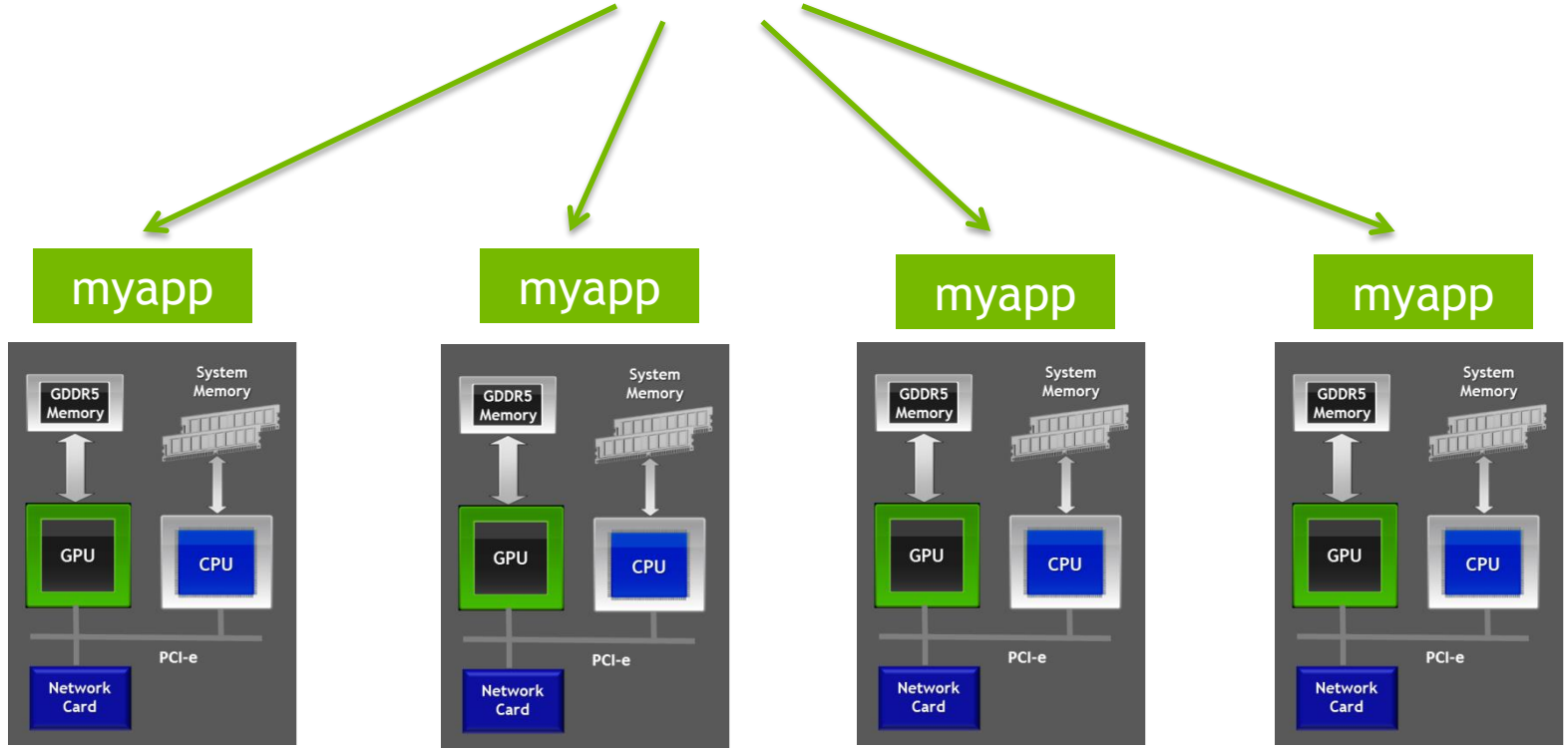
- Standard to exchange data between processes via messages
  - Defines API to exchanges messages
    - Pt. 2 Pt.: e.g. MPI_Send, MPI_Recv
    - Collectives, e.g. MPI_Reduce
- Multiple implementations (open source and commercial)
  - Binding for C/C++, Fortran, Python, …
  - E.g. MPICH, OpenMPI, MVAPICH, IBM Platform MPI, Cray MPT, …

MPI - A MINIMAL PROGRAM

```
#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

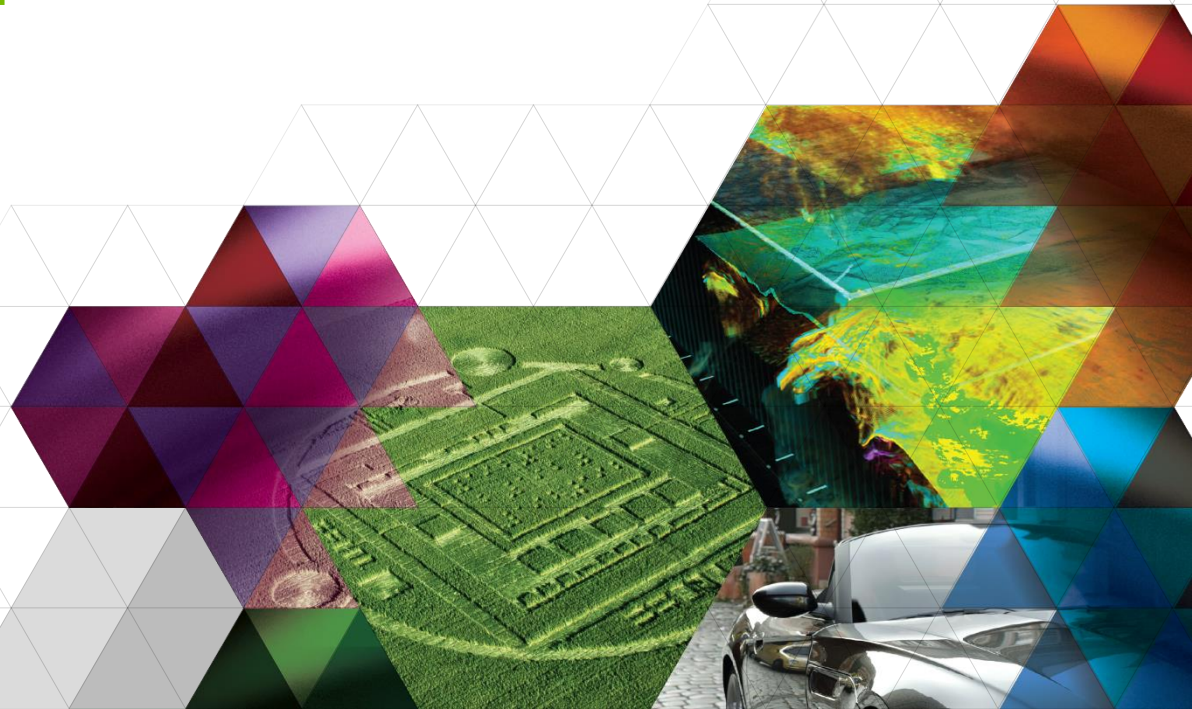
```
$ mpicc -o myapp myapp.c
$ mpirun -np 4 ./myapp <args>
```



GPU TECHNOLOGY CONFERENCE

NVIDIA

A SIMPLE EXAMPLE

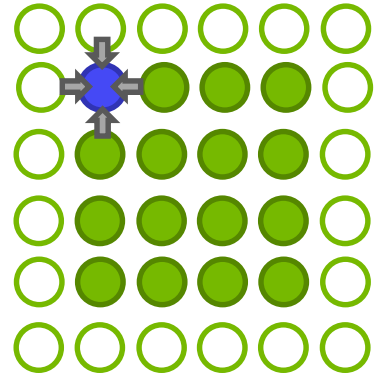


EXAMPLE: JACOBI SOLVER – SINGLE GPU

While not converged

- Do Jacobi step:

```
for (int i=1; i < n-1; i++)
   for (int j=1; j < m-1; j++)
     u_new[i][j] = 0.0f - 0.25f*(u[i-1][j] + u[i+1][j]
                                 + u[i][j-1] + u[i][j+1])
```



- Swap `u_new` and `u`
- Next iteration

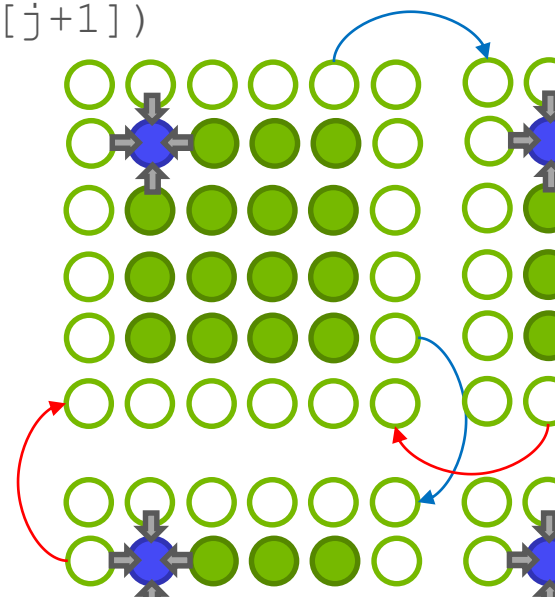
EXAMPLE: JACOBI SOLVER – MULTI GPU

While not converged

- Do Jacobi step:

```
for (int i=1; i < n-1; i++)
  for (int j=1; j < m-1; j++)
    u_new[i][j] = 0.0f - 0.25f*(u[i-1][j] + u[i+1][j]
                                + u[i][j-1] + u[i][j+1])
```

- Exchange halo with 2 4 neighbor
- Swap `u_new` and `u`
- Next iteration



EXAMPLE: JACOBI SOLVER

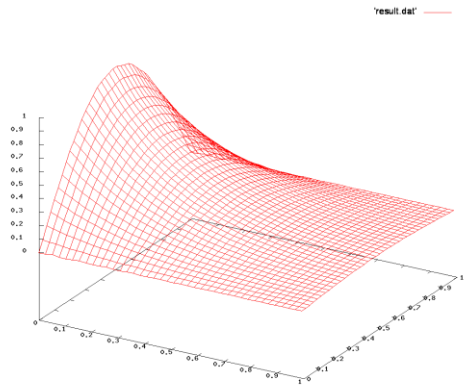
- Solves the 2D-Laplace equation on a rectangle

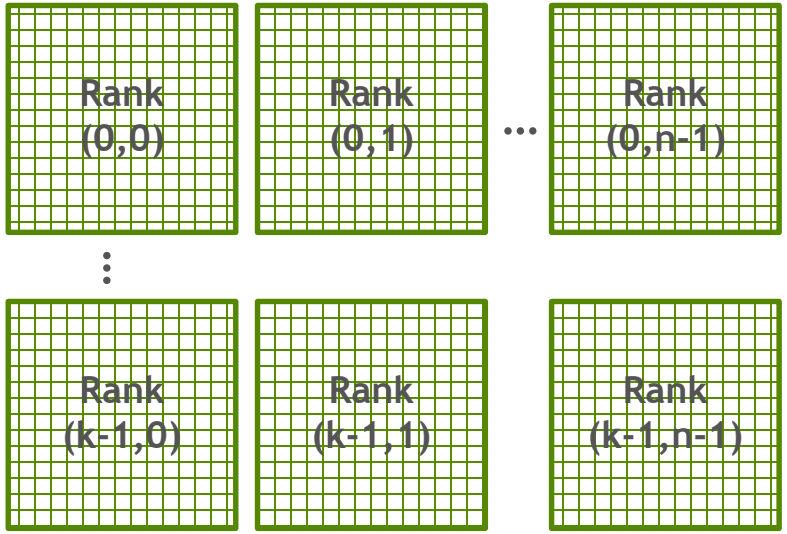
$$\Delta u(x,y) = 0 \; \forall \; (x,y) \in \Omega \backslash \delta\Omega$$

  - Dirichlet boundary conditions (constant values on boundaries)

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

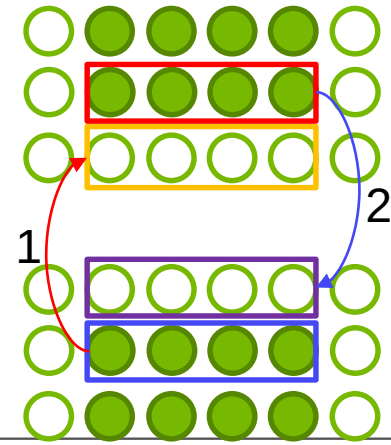
- 2D domain decomposition with n x k domains



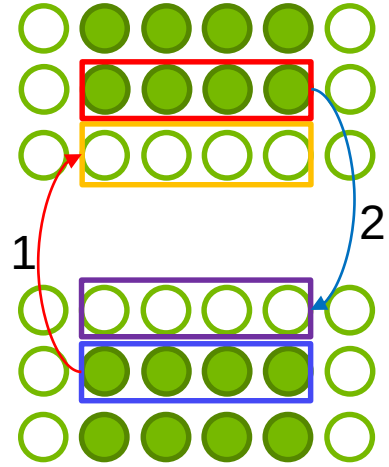


EXAMPLE: JACOBI – TOP/BOTTOM HALO UPDATE

```
MPI_Sendrecv(u_new+offset_first_row, m-2, MPI_DOUBLE, t_nb, 0,
             u_new+offset_bottom_bondary, m-2, MPI_DOUBLE, b_nb, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```



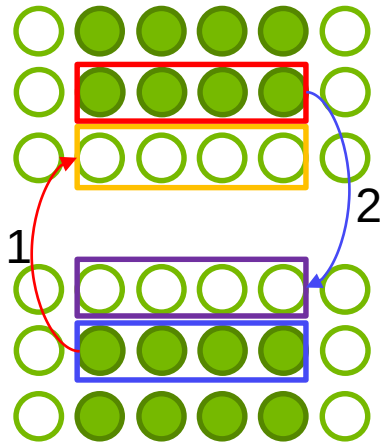

```
MPI_Sendrecv(u_new+offset_last_row, m-2, MPI_DOUBLE, b_nb, 1,
             u_new+offset_top_bondary, m-2, MPI_DOUBLE, t_nb, 1,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```



EXAMPLE: JACOBI – TOP/BOTTOM HALO UPDATE

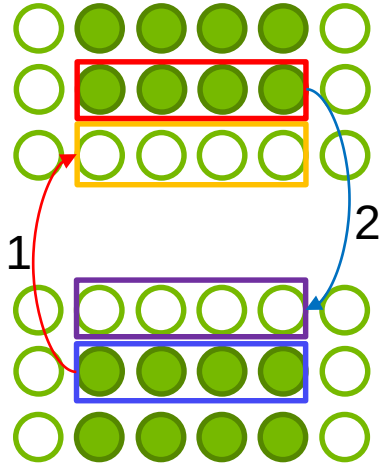
OpenACC

```
#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_bondary, 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_bondary, m-2, MPI_DOUBLE, t_nb, 1,
               MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}
```



CUDA

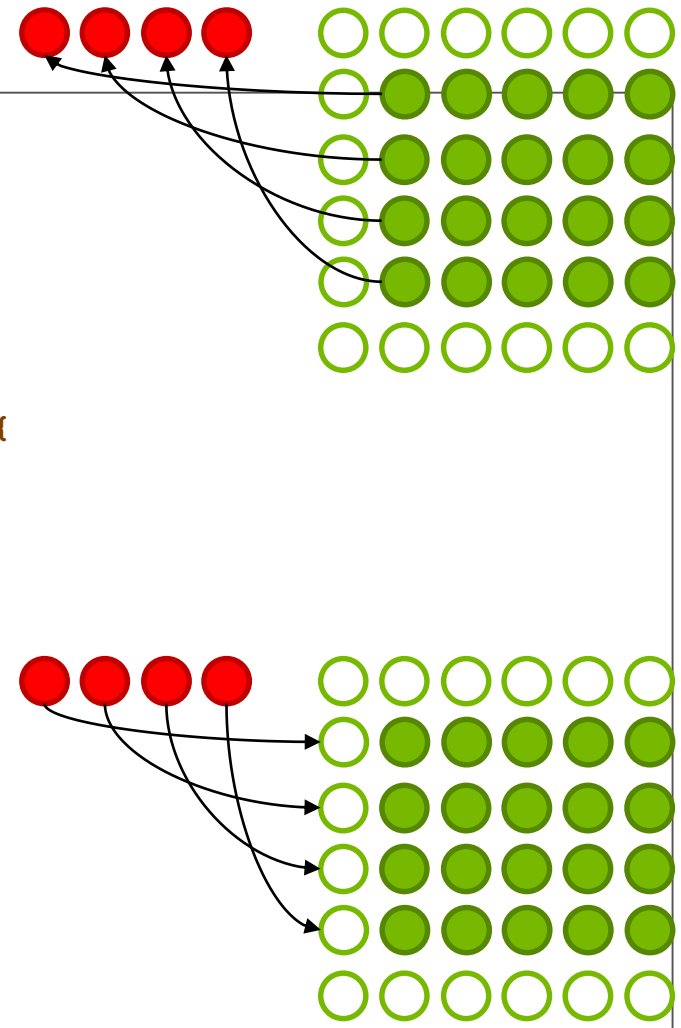
```
MPI_Sendrecv(u_new_d+offset_first_row, m-2, MPI_DOUBLE, t_nb, 0,
             u_new_d+offset_bottom_bondary, m-2, MPI_DOUBLE, b_nb, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
MPI_Sendrecv(u_new_d+offset_last_row, m-2, MPI_DOUBLE, b_nb, 1,
             u_new_d+offset_top_bondary, m-2, MPI_DOUBLE, t_nb, 1,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```



EXAMPLE: JACOBI – LEFT/RIGHT HALO UPDATE

OpenACC

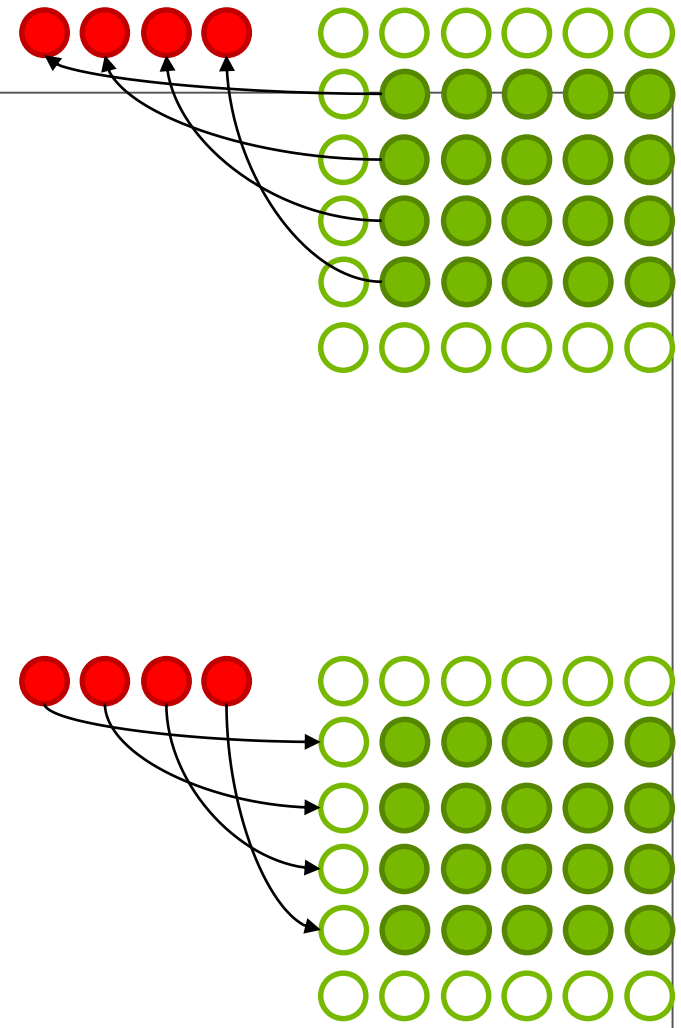
```
//right neighbor omitted
#pragma acc parallel loop present ( u_new, to_left )
for ( int i=0; i<n-2; ++i )
     to_left[i] = u_new[(i+1)*m+1];

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

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



EXAMPLE: JACOBI – LEFT/RIGHT HALO UPDATE

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_left_d, n-2, MPI_DOUBLE, l_nb, 0,
              MPI_COMM_WORLD, MPI_STATUS_IGNORE );




unpack<<<gs,bs,0,s>>>(u_new_d, from_left_d, n, m);
```



STARTING MPI+CUDA/OPENACC PROGRAMS

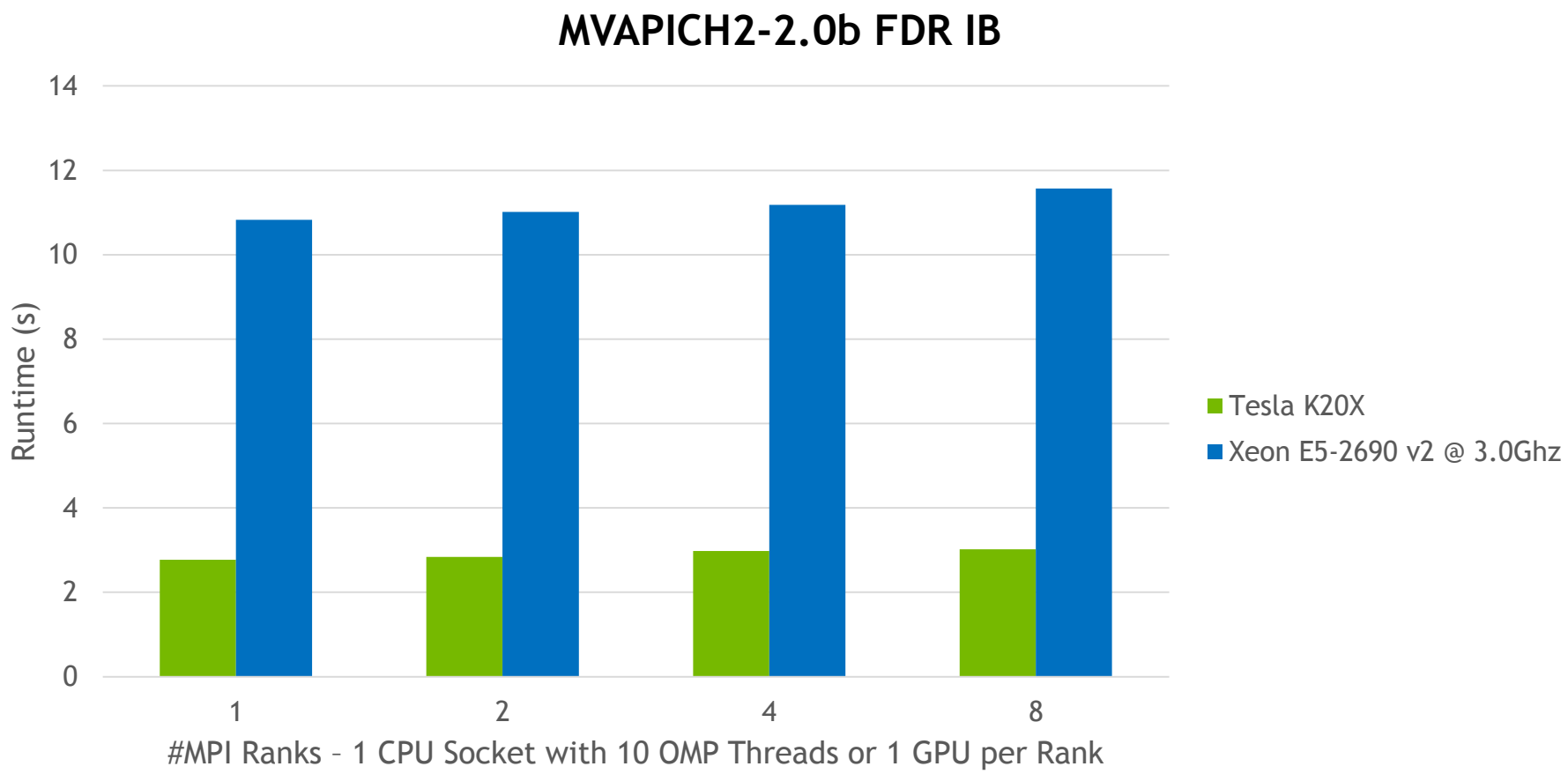
- Launch one process per GPU
  - MVAPICH: `MV2_USE_CUDA`

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

  - Open MPI: CUDA-aware features are enabled per default
  - Cray: `MPICH_RDMA_ENABLED_CUDA`
  - IBM Platform MPI: `PMPI_GPU_AWARE`

JACOBI RESULTS (1000 STEPS)

WEAK SCALING 4K X 4K PER PROCESS



EXAMPLE: JACOBI – TOP/BOTTOM HALO UPDATE – WITHOUT CUDA-AWARE MPI

OpenACC

```
#pragma acc update host( u_new[1:m-2], u_new[(n-2)*m+1:m-2] )
MPI_Sendrecv(u_new+offset_first_row, m-2, MPI_DOUBLE, t_nb, 0,
             u_new+offset_bottom_bondary, 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_bondary, m-2, MPI_DOUBLE, t_nb, 1,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
#pragma acc update device( u_new[0:m-2], u_new[(n-2)*m:m-2] )
```

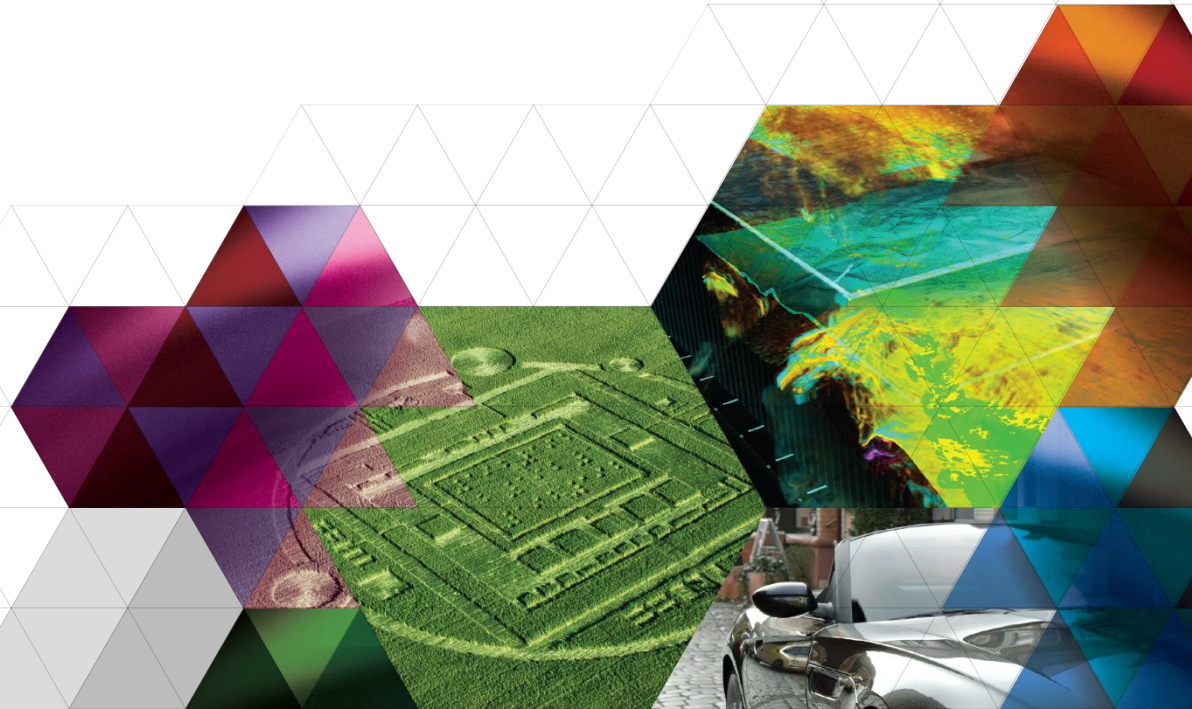
CUDA

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

cudaMemcpy(u_new+1, u_new_d+1, (m-2)*sizeof(double), cudaMemcpyDeviceToHost);
MPI_Sendrecv(u_new+offset_first_row, m-2, MPI_DOUBLE, t_nb, 0,
             u_new+offset_bottom_bondary, m-2, MPI_DOUBLE, b_nb, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
cudaMemcpy(u_new_d, u_new, (m-2)*sizeof(double), cudaMemcpyDeviceToHost);
```

GPU TECHNOLOGY CONFERENCE

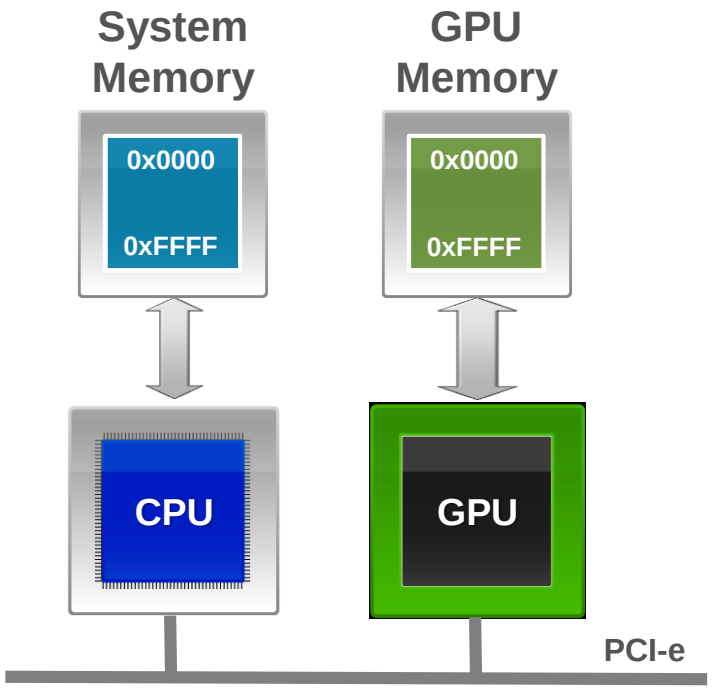
NVIDIA

THE DETAILS

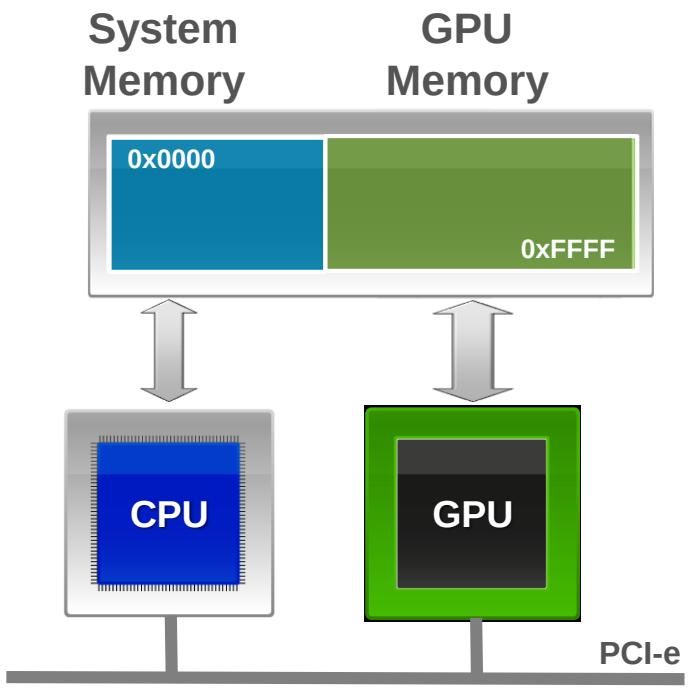


UNIFIED VIRTUAL ADDRESSING

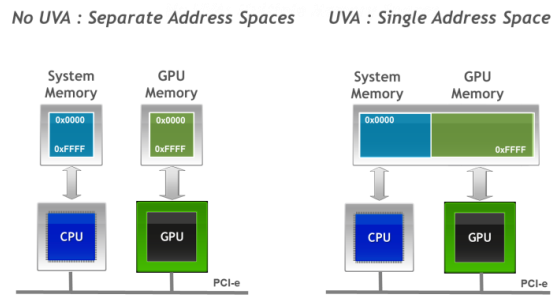
No UVA : Separate Address Spaces



UVA : Single Address Space

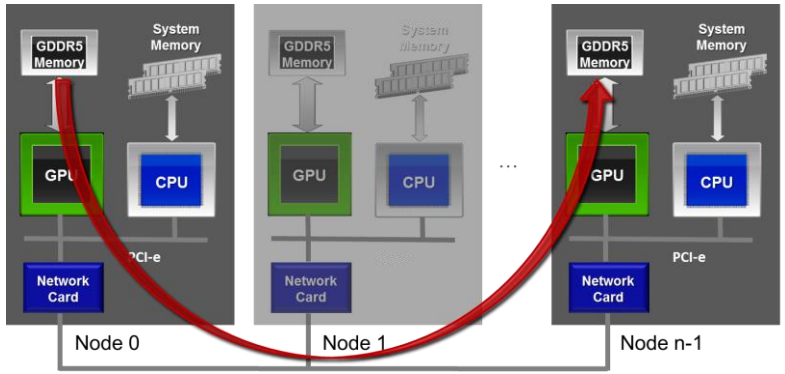


UNIFIED VIRTUAL ADDRESSING



- 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 on Windows also TCC mode

MPI+CUDA



With UVA and CUDA-aware MPI

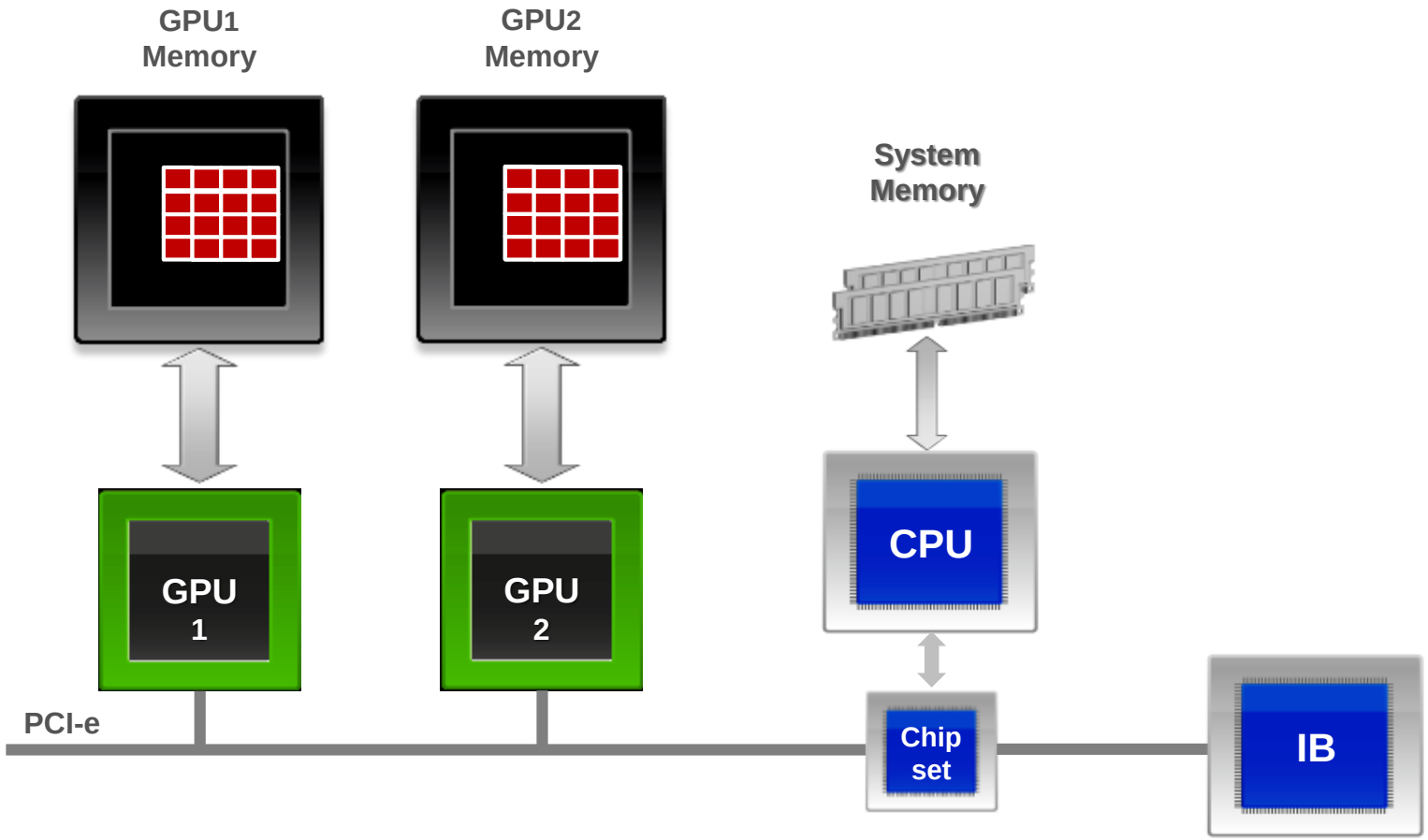
```
//MPI rank 0
MPI_Send(s_buf_d,size,…);

//MPI rank n-1
MPI_Recv(r_buf_d,size,…);
```

No UVA and regular MPI

```
//MPI rank 0
cudaMemcpy(s_buf_h,s_buf_d,size,…);
MPI_Send(s_buf_h,size,…);

//MPI rank n-1
MPI_Recv(r_buf_h,size,…);
cudaMemcpy(r_buf_d,r_buf_h,size,…);
```

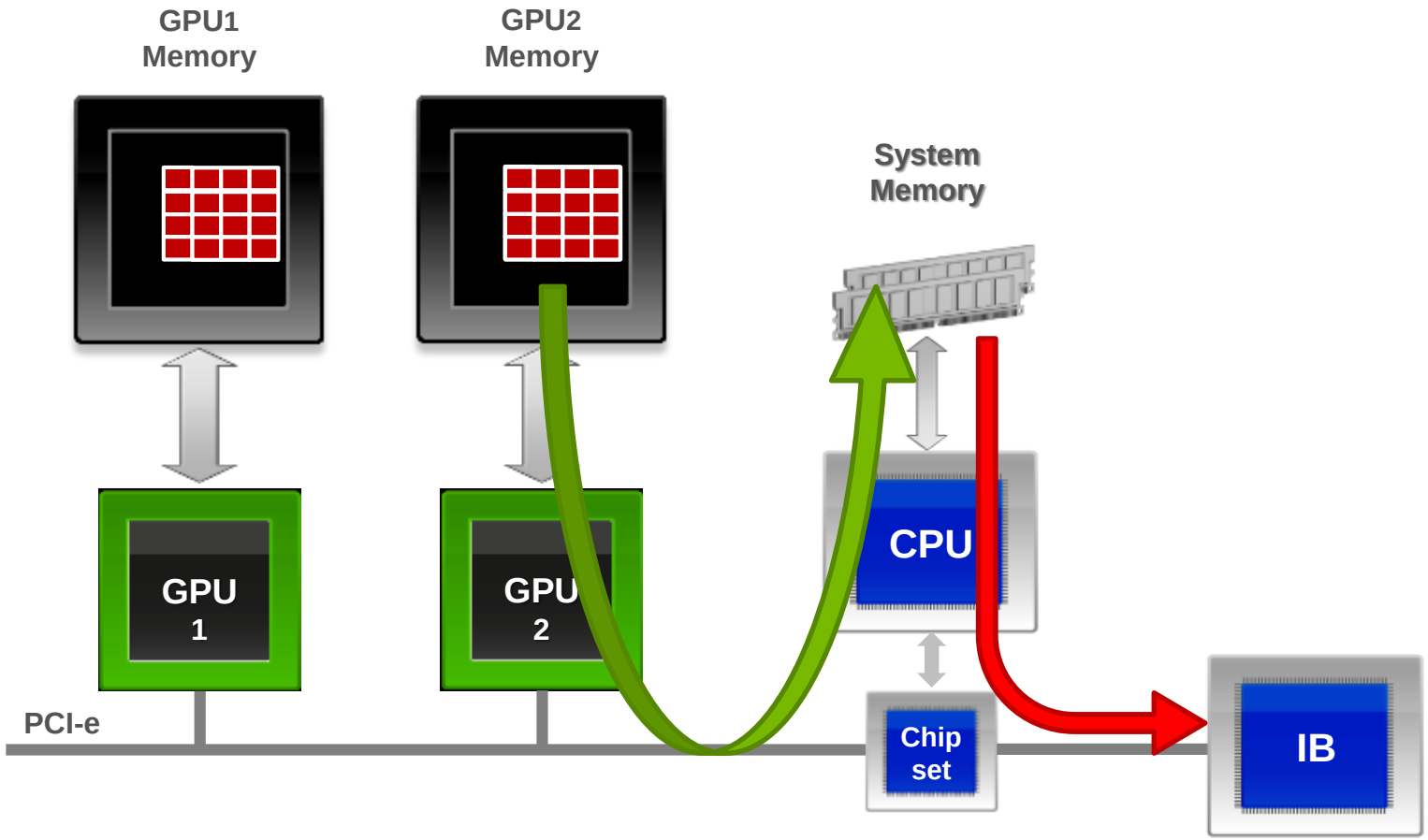
NVIDIA GPUDIRECT™

ACCELERATED COMMUNICATION WITH NETWORK & STORAGE DEVICES



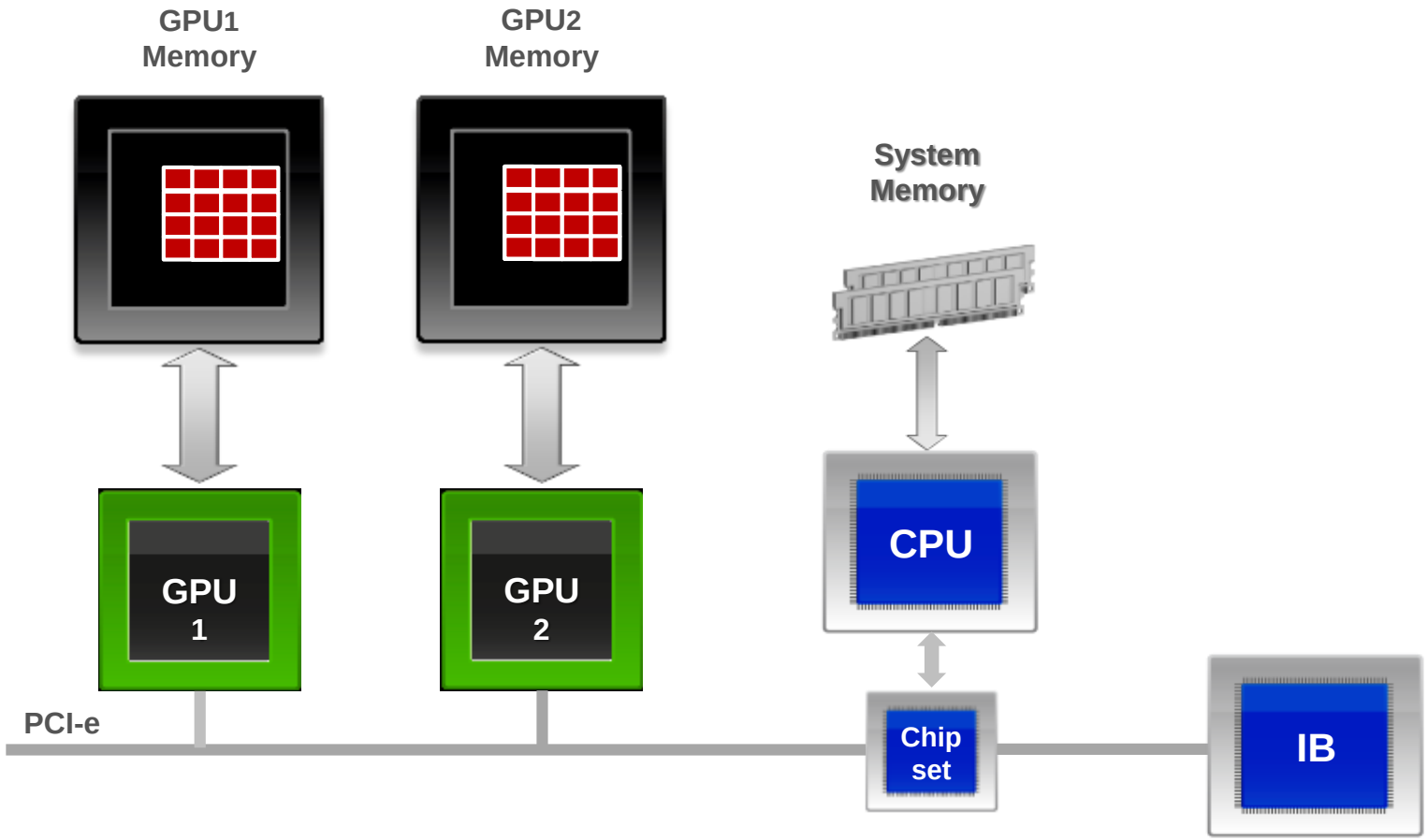
NVIDIA GPUDIRECT™

ACCELERATED COMMUNICATION WITH NETWORK & STORAGE DEVICES



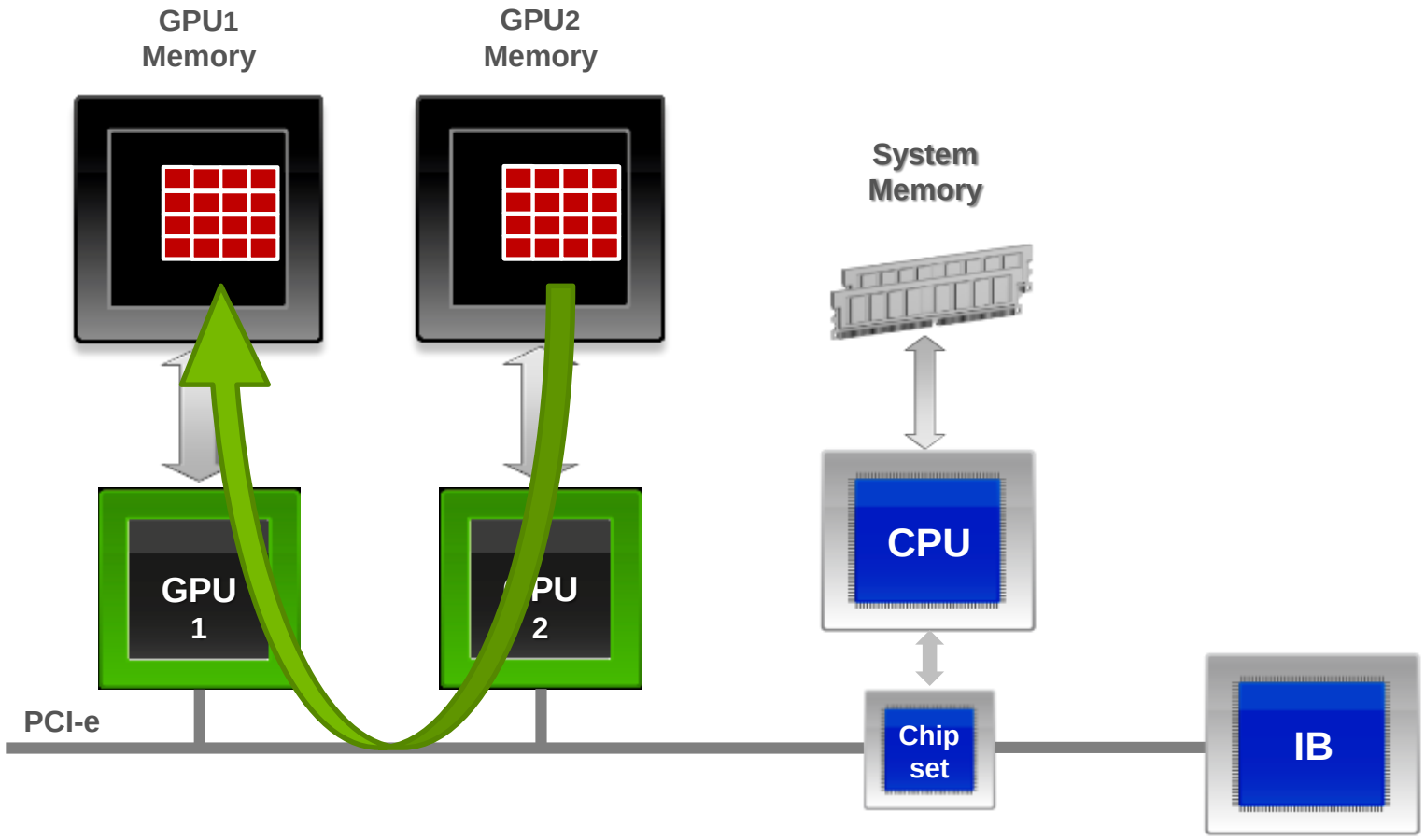
NVIDIA GPUDIRECT™

PEER TO PEER TRANSFERS



NVIDIA GPUDIRECT™

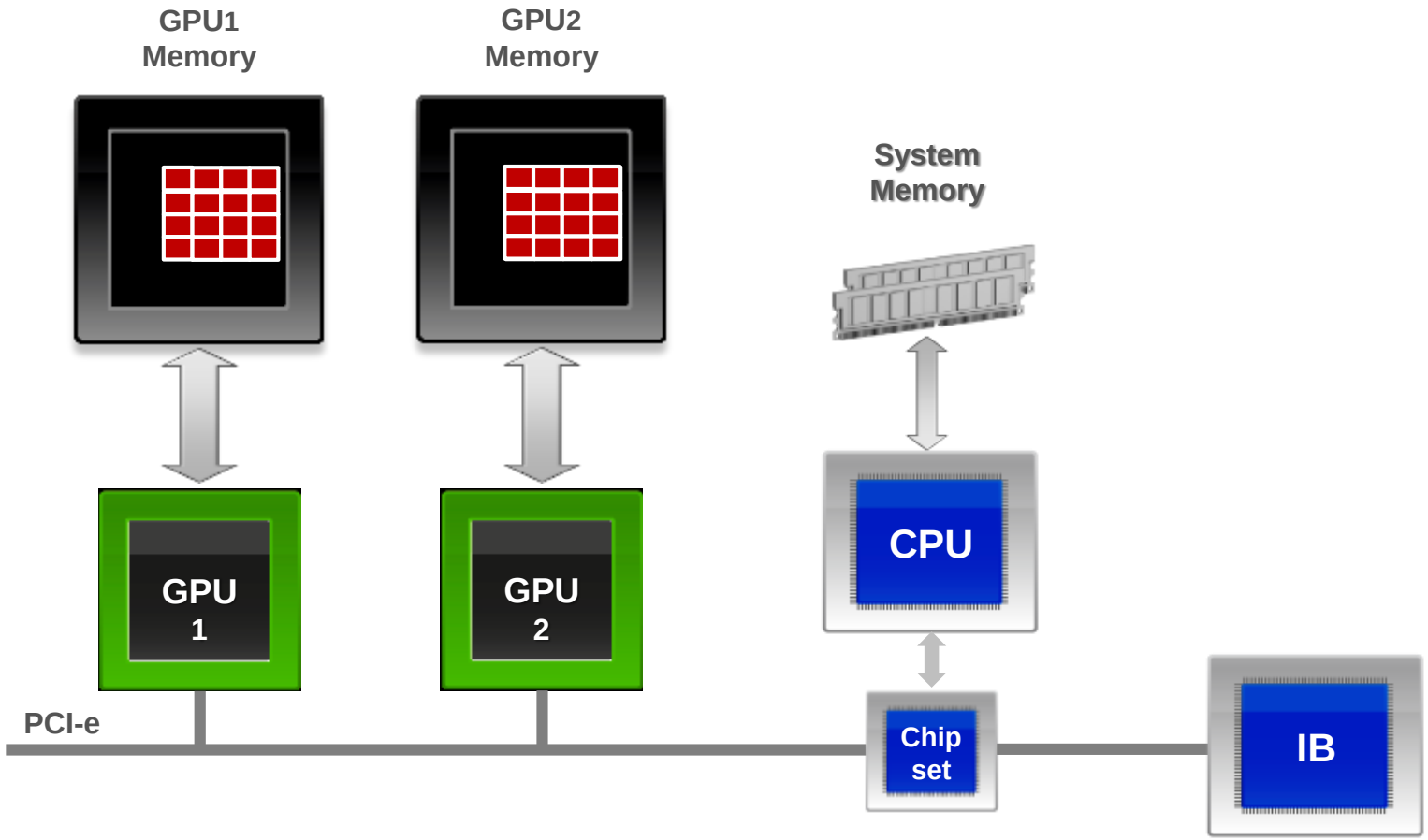
PEER TO PEER TRANSFERS



GPU TECHNOLOGY CONFERENCE

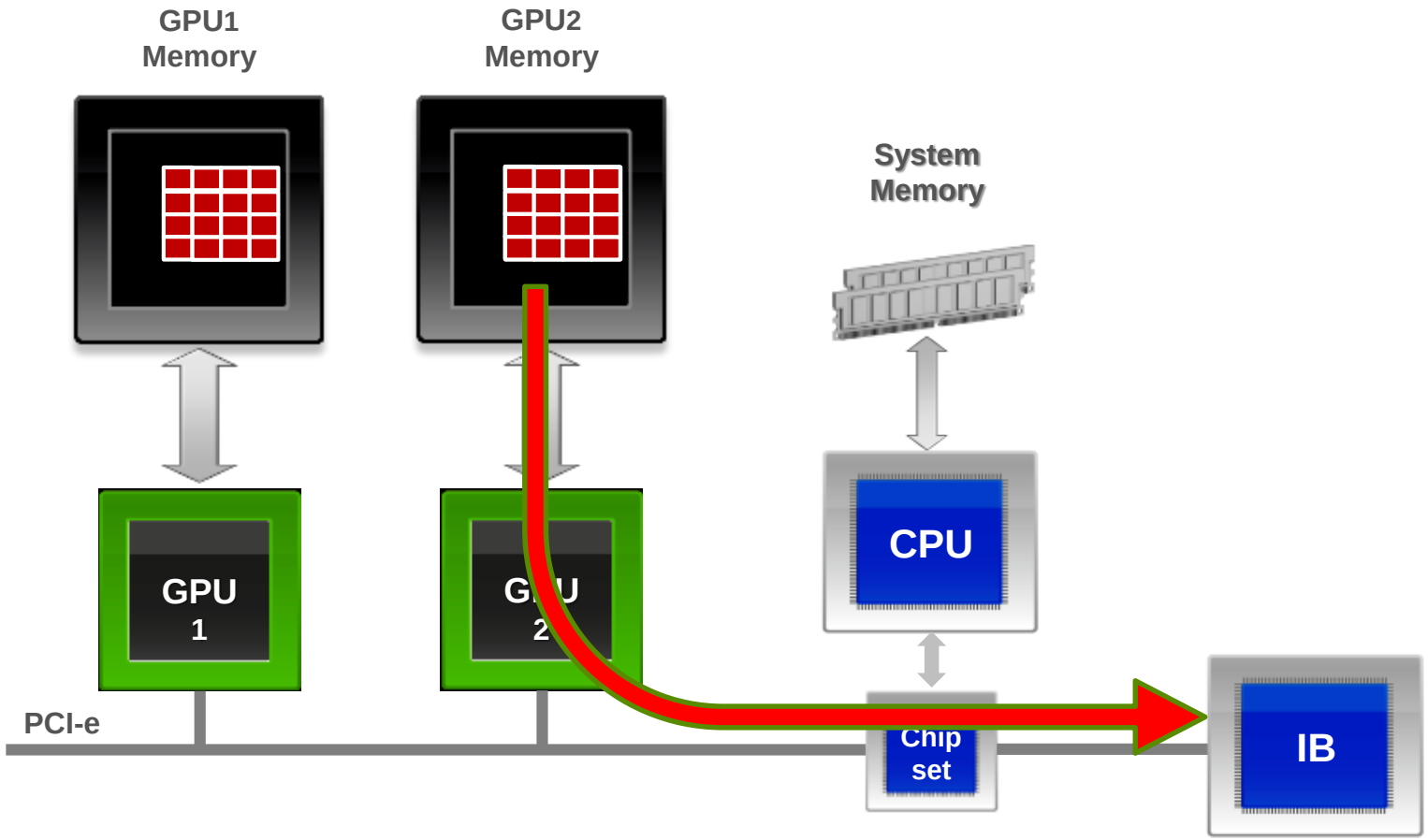
NVIDIA GPUDIRECT™

SUPPORT FOR RDMA



NVIDIA GPUDIRECT™

SUPPORT FOR RDMA



CUDA-AWARE MPI

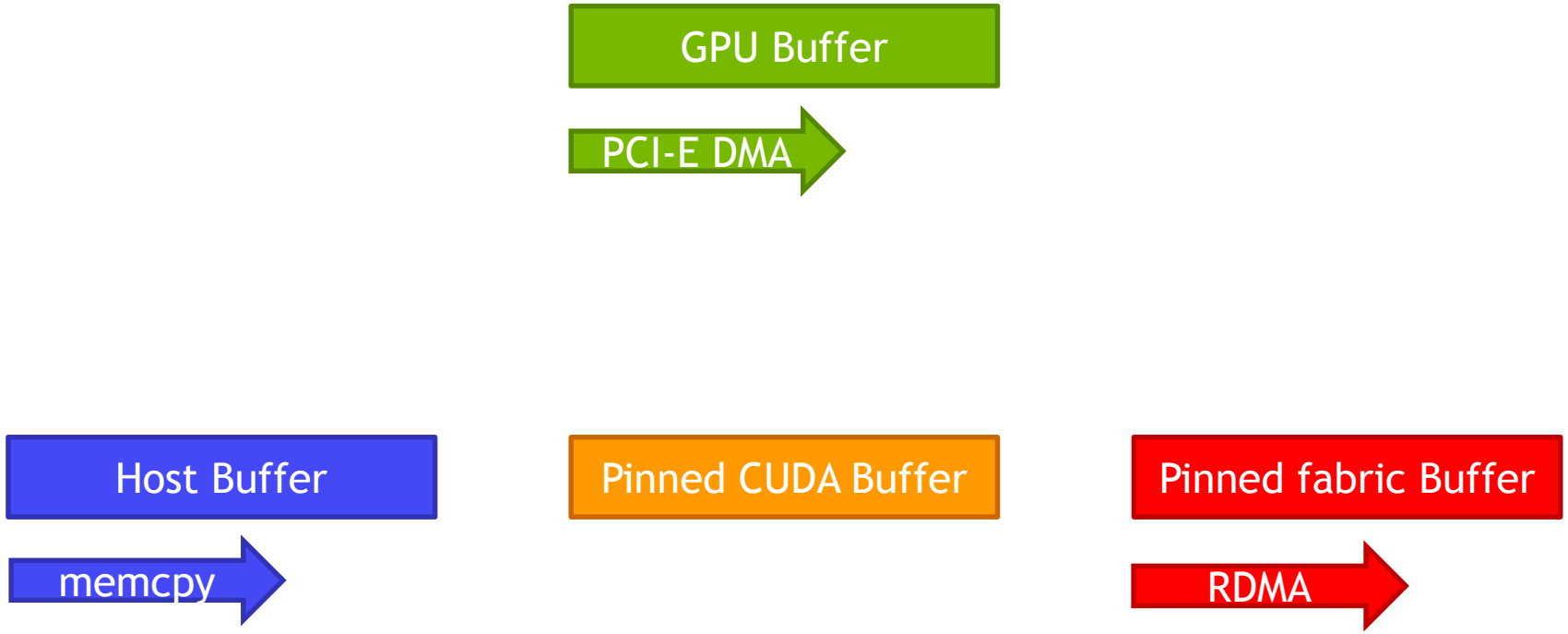
Example:

MPI Rank 0 MPI_Send from GPU Buffer

MPI Rank 1 MPI_Recv to GPU Buffer

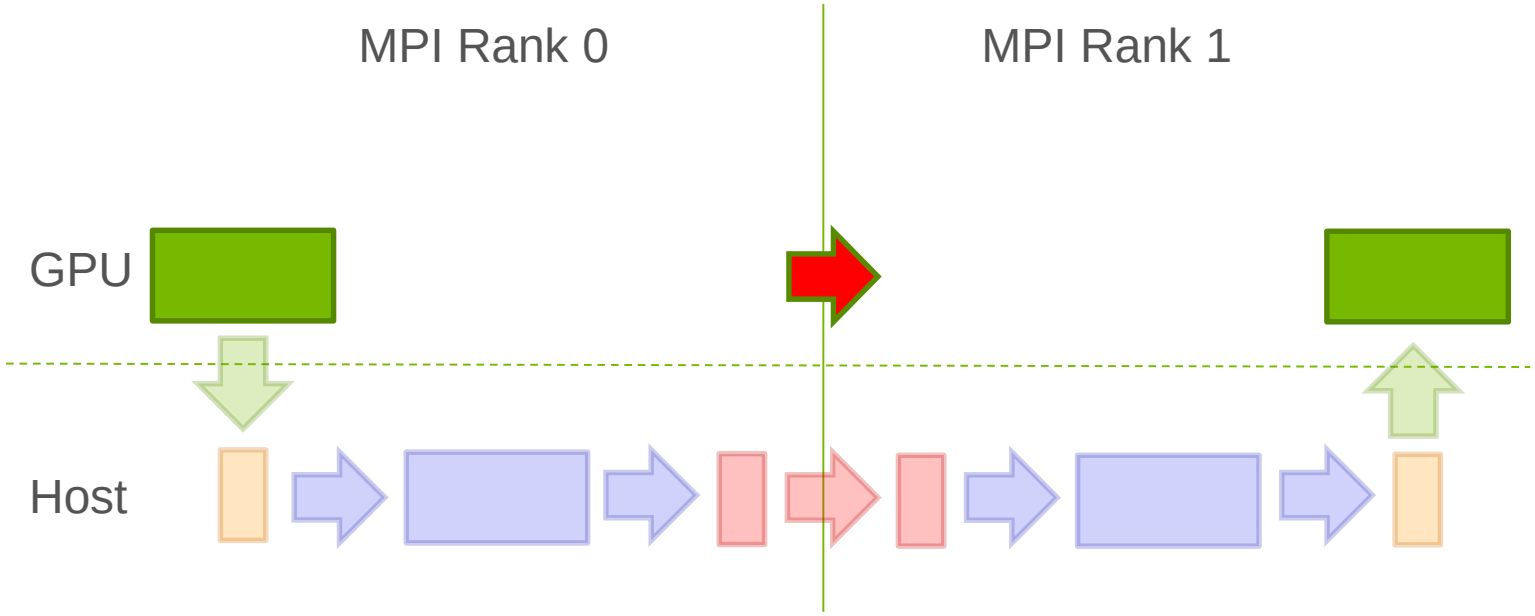
- Show how CUDA+MPI works in principle
  - Depending on the MPI implementation, message size, system setup, … situation might be different
- Two GPUs in two nodes

CUDA-AWARE MPI



MPI GPU TO REMOTE GPU

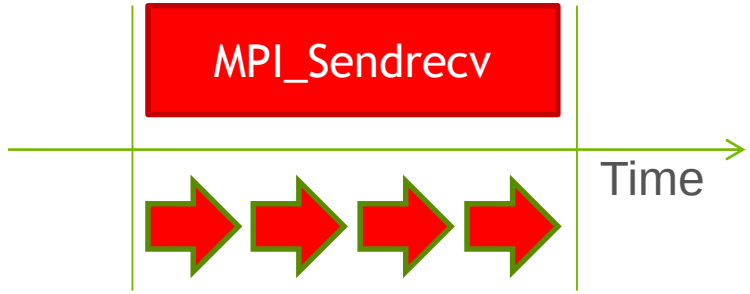
GPUDIRECT SUPPORT FOR RDMA



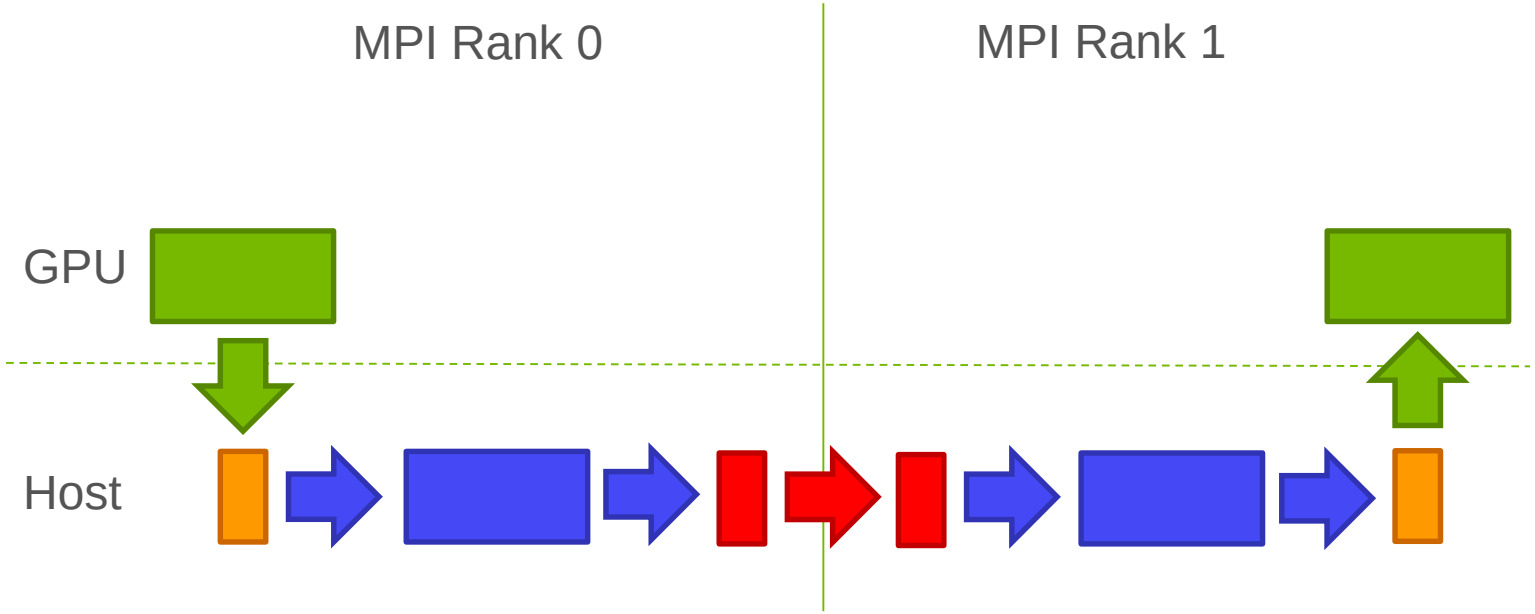

```
MPI_Send(s_buf_d,size,MPI_CHAR,1,tag,MPI_COMM_WORLD);

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

MPI GPU TO REMOTE GPU

GPUDIRECT SUPPORT FOR RDMA

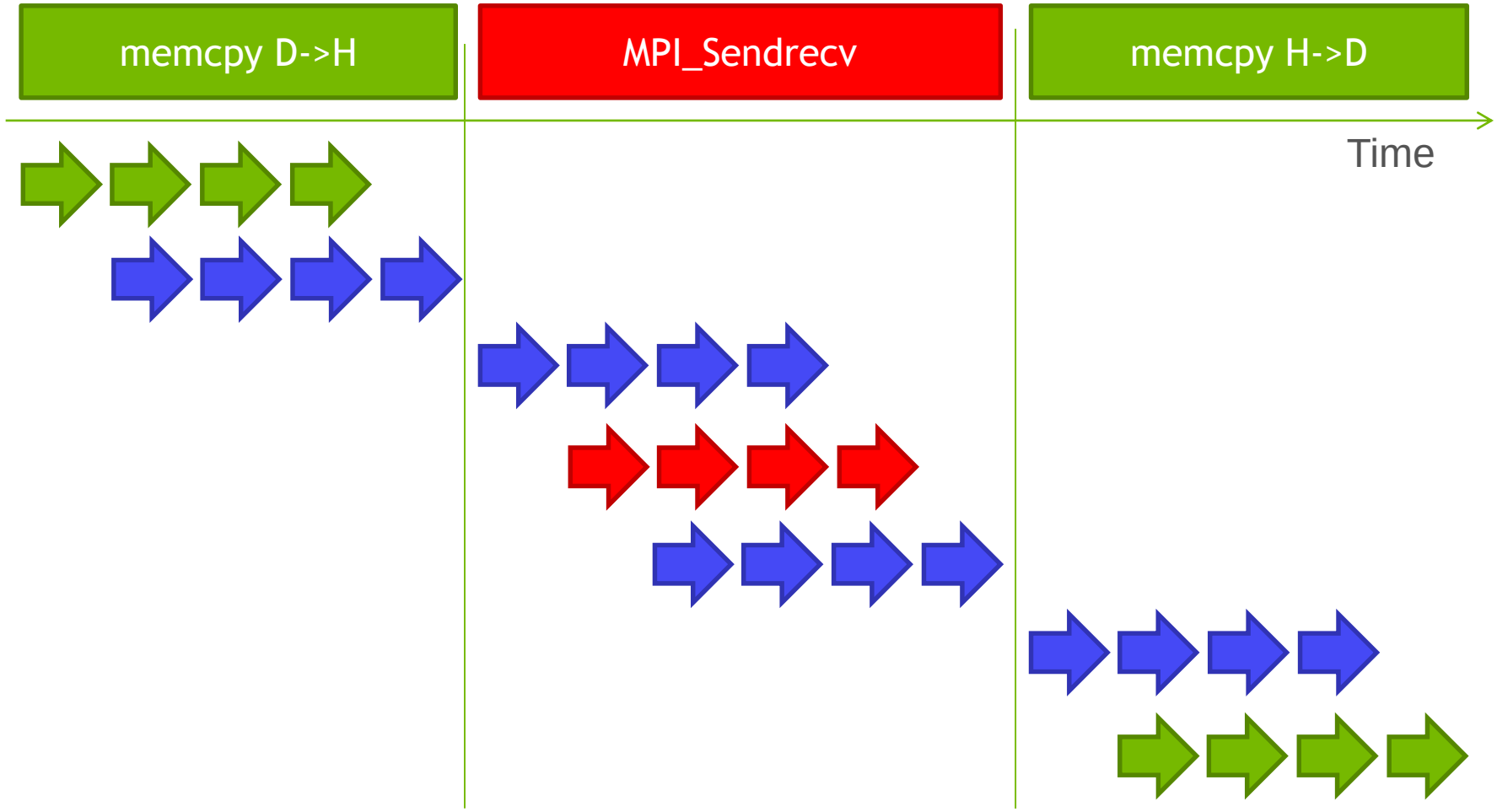


REGULAR MPI GPU TO REMOTE GPU



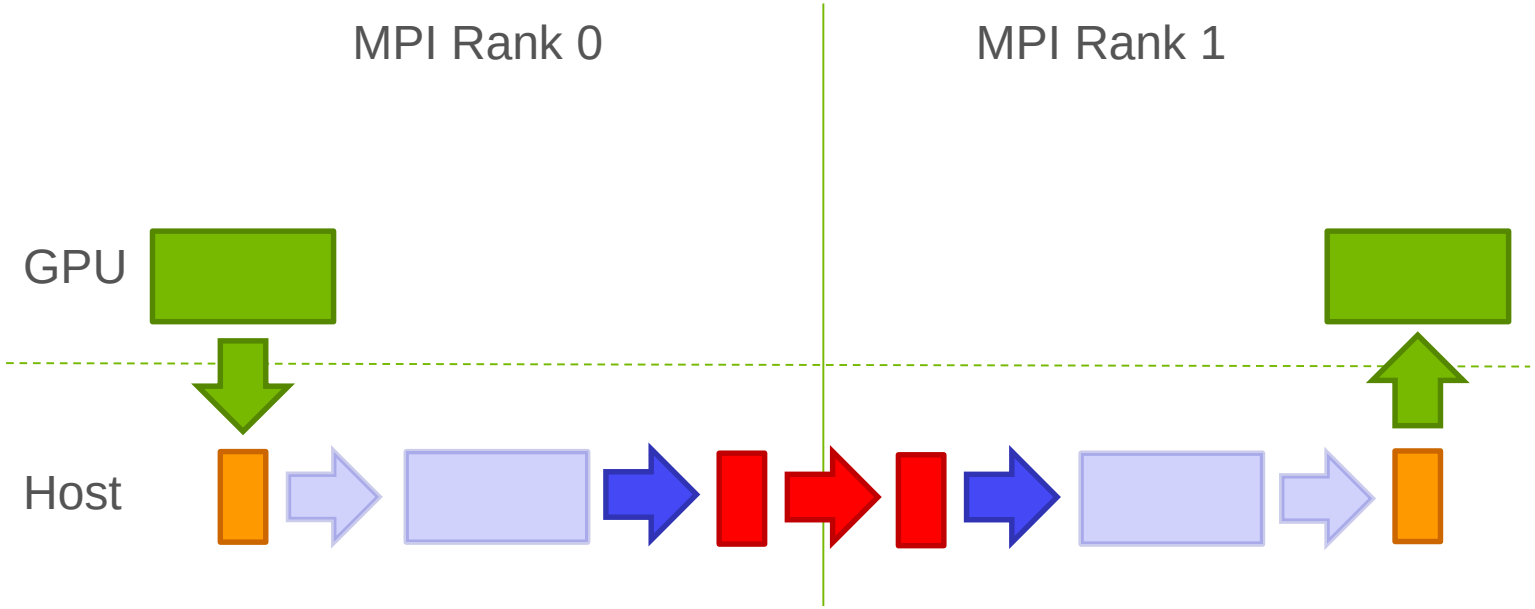

```
cudaMemcpy(s_buf_h,s_buf_d,size,cudaMemcpyDeviceToHost);
MPI_Send(s_buf_h,size,MPI_CHAR,1,tag,MPI_COMM_WORLD);

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



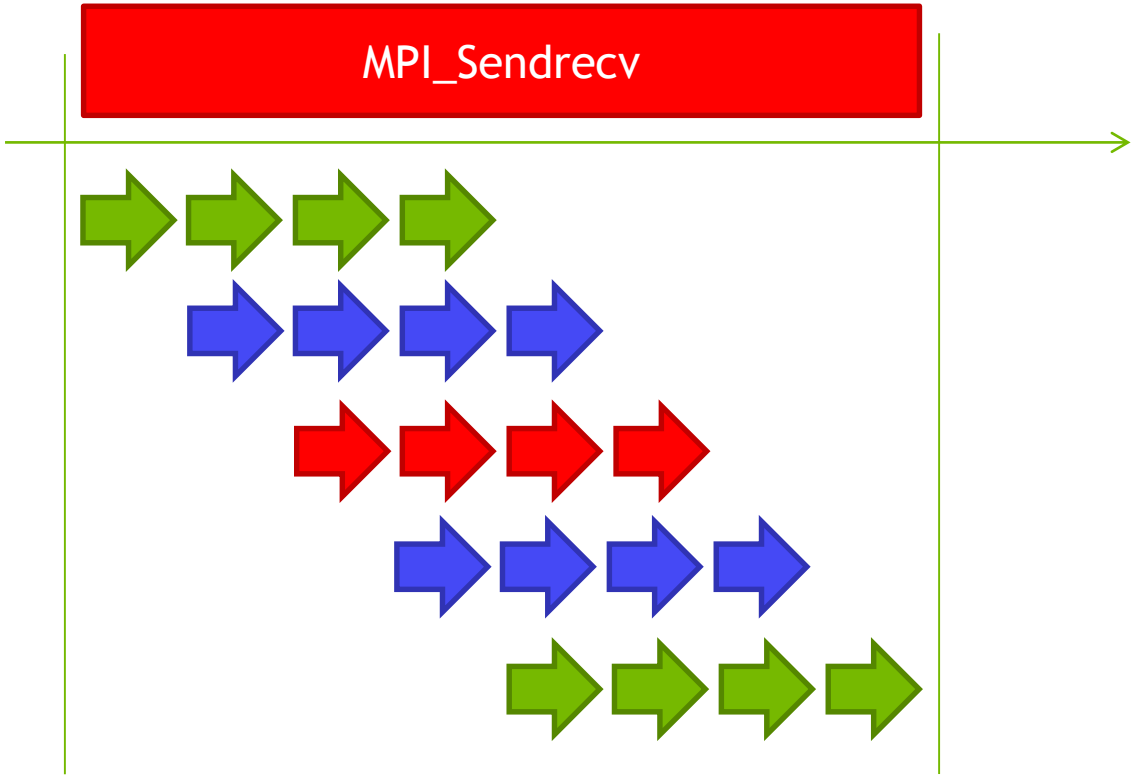
MPI GPU TO REMOTE GPU

WITHOUT GPUDIRECT

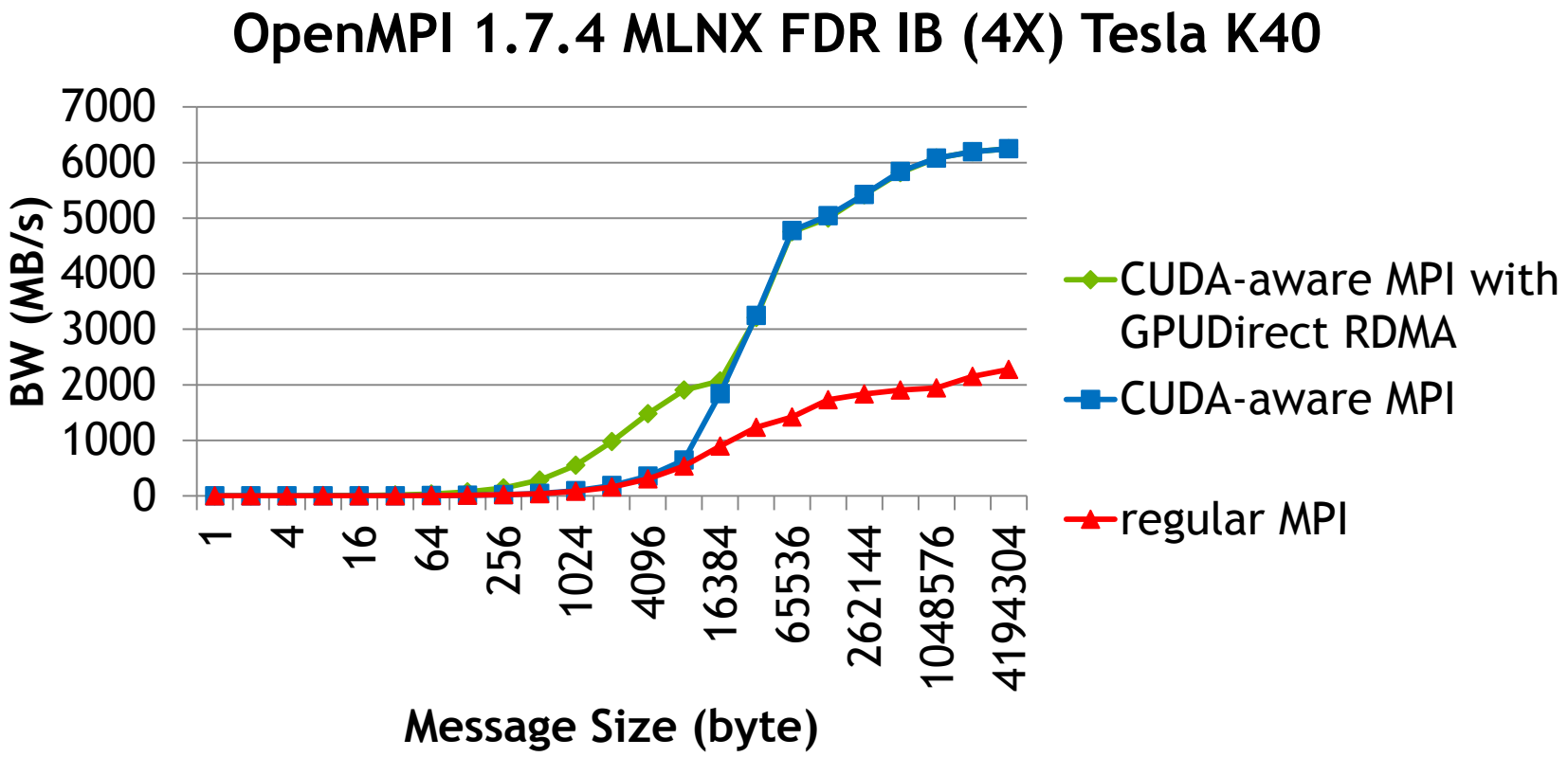



```
MPI_Send(s_buf_h,size,MPI_CHAR,1,tag,MPI_COMM_WORLD);

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

MPI GPU TO REMOTE GPU
WITHOUT GPUDIRECT



PERFORMANCE RESULTS TWO NODES

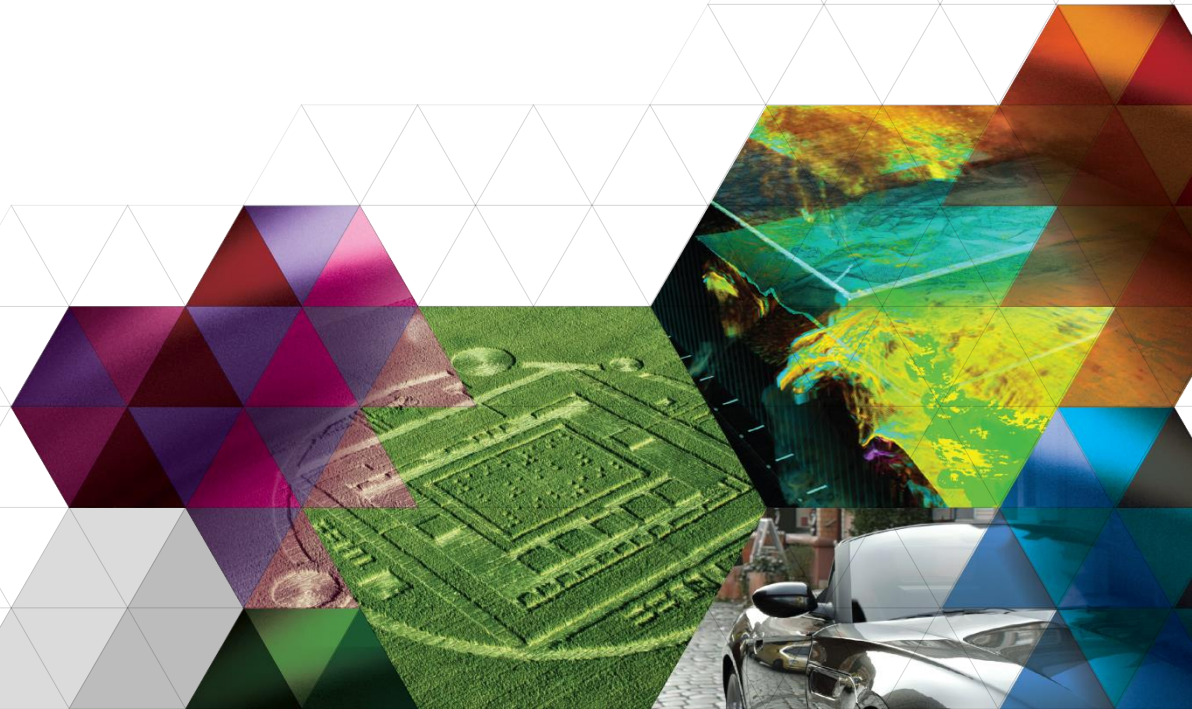


Latency (1 byte)	19.04 us	16.91 us	5.52 us

GPU TECHNOLOGY CONFERENCE

NVIDIA

MULTI PROCESS SERVICE (MPS) FOR MPI APPLICATIONS



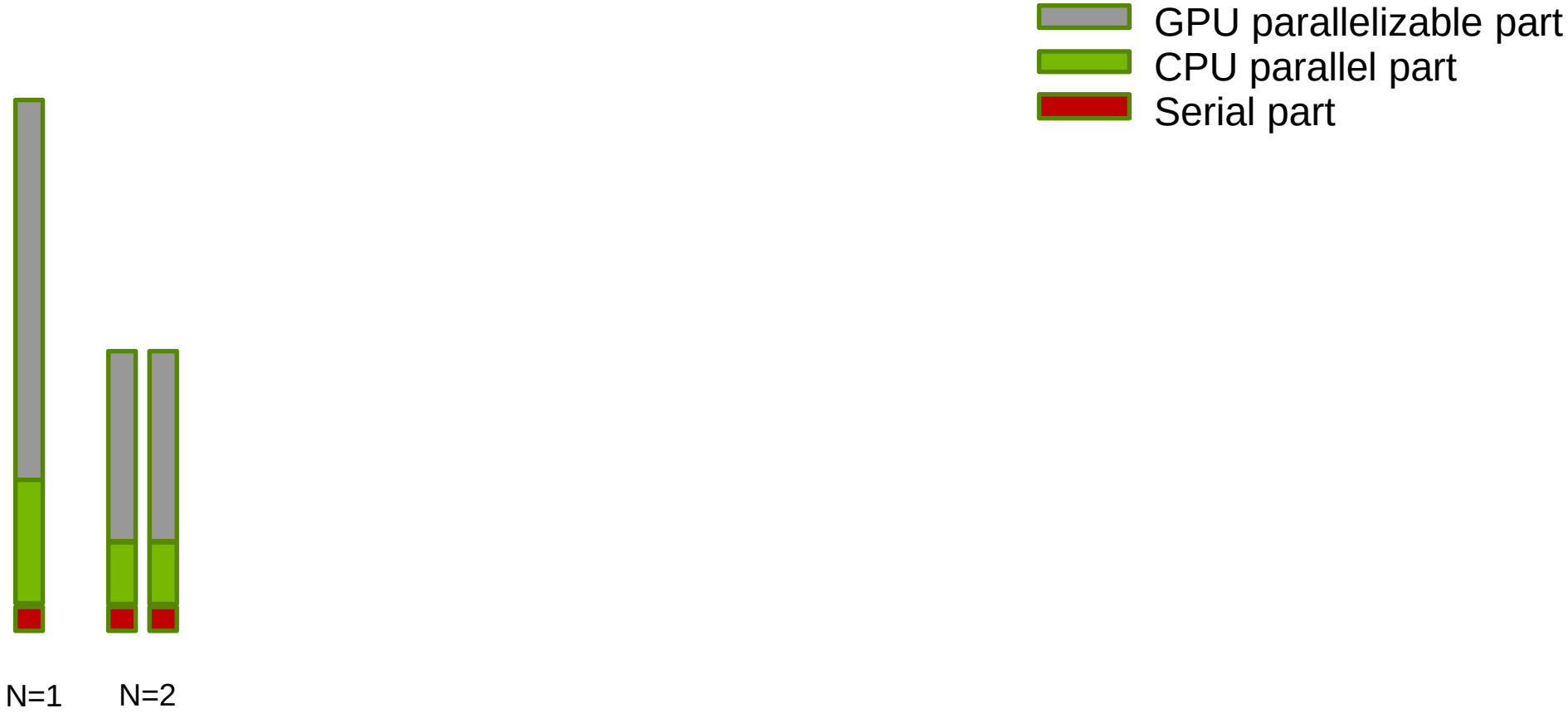
GPU ACCELERATION OF LEGACY MPI APPLICATION

- 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

N=1

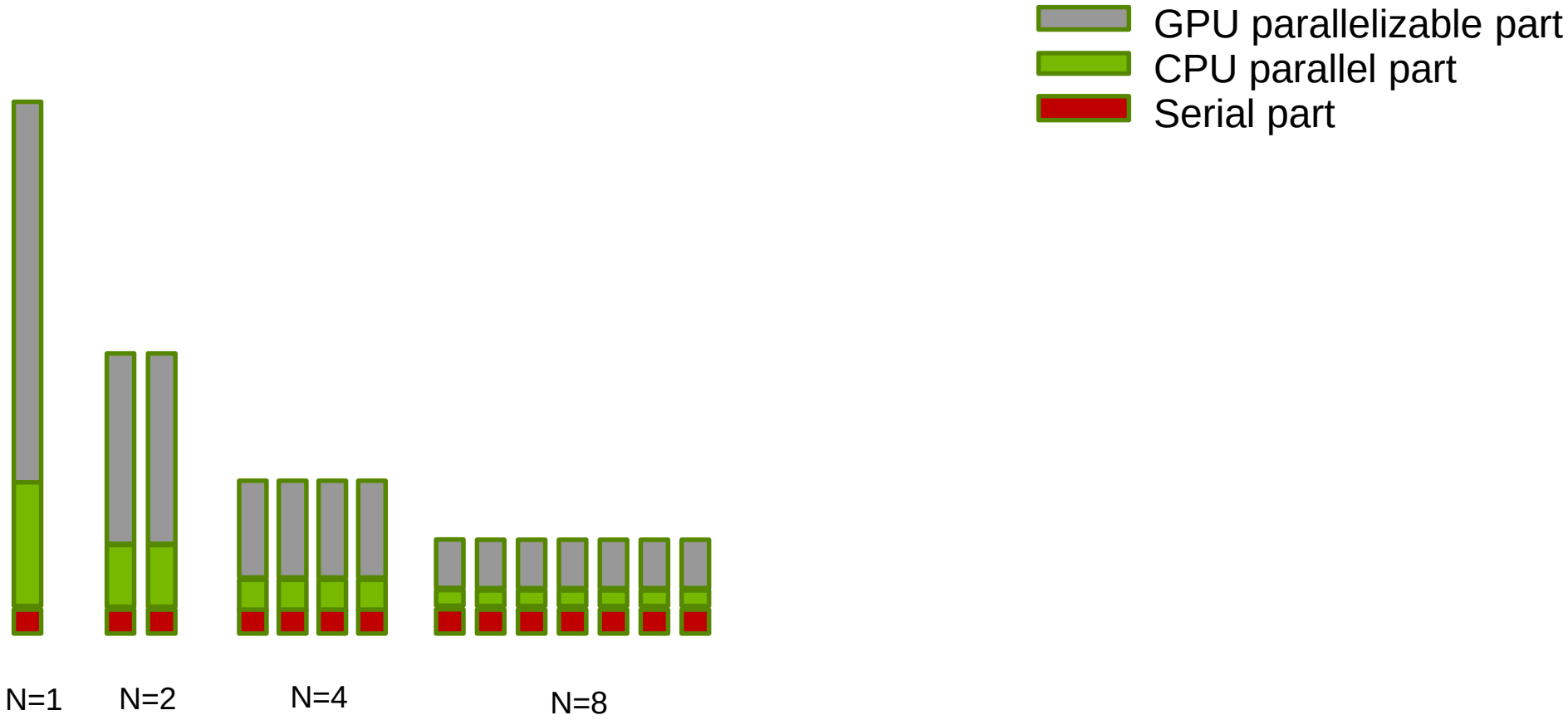
GPU parallelizable part
CPU parallel part
Serial part

Multicore CPU only

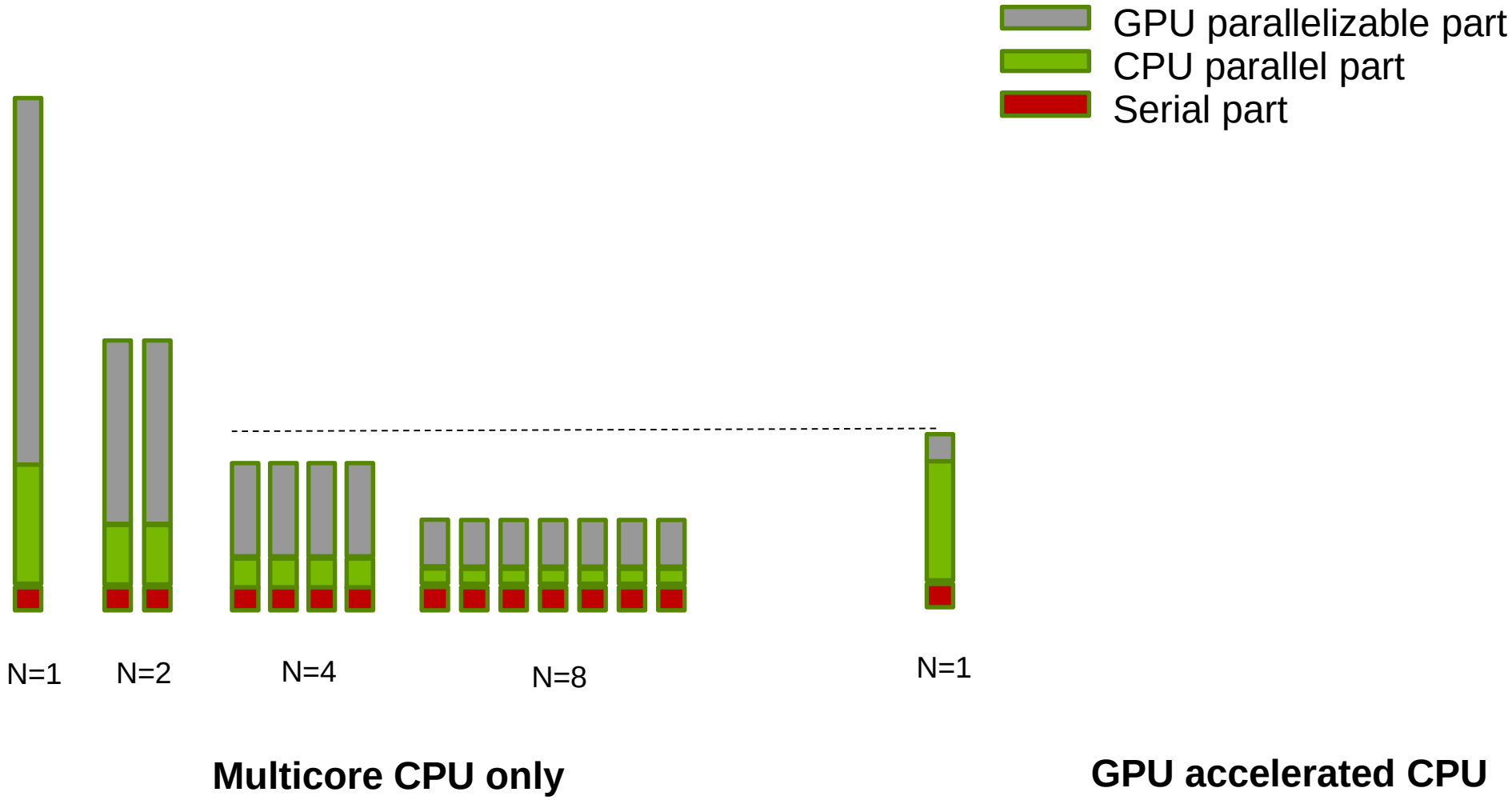


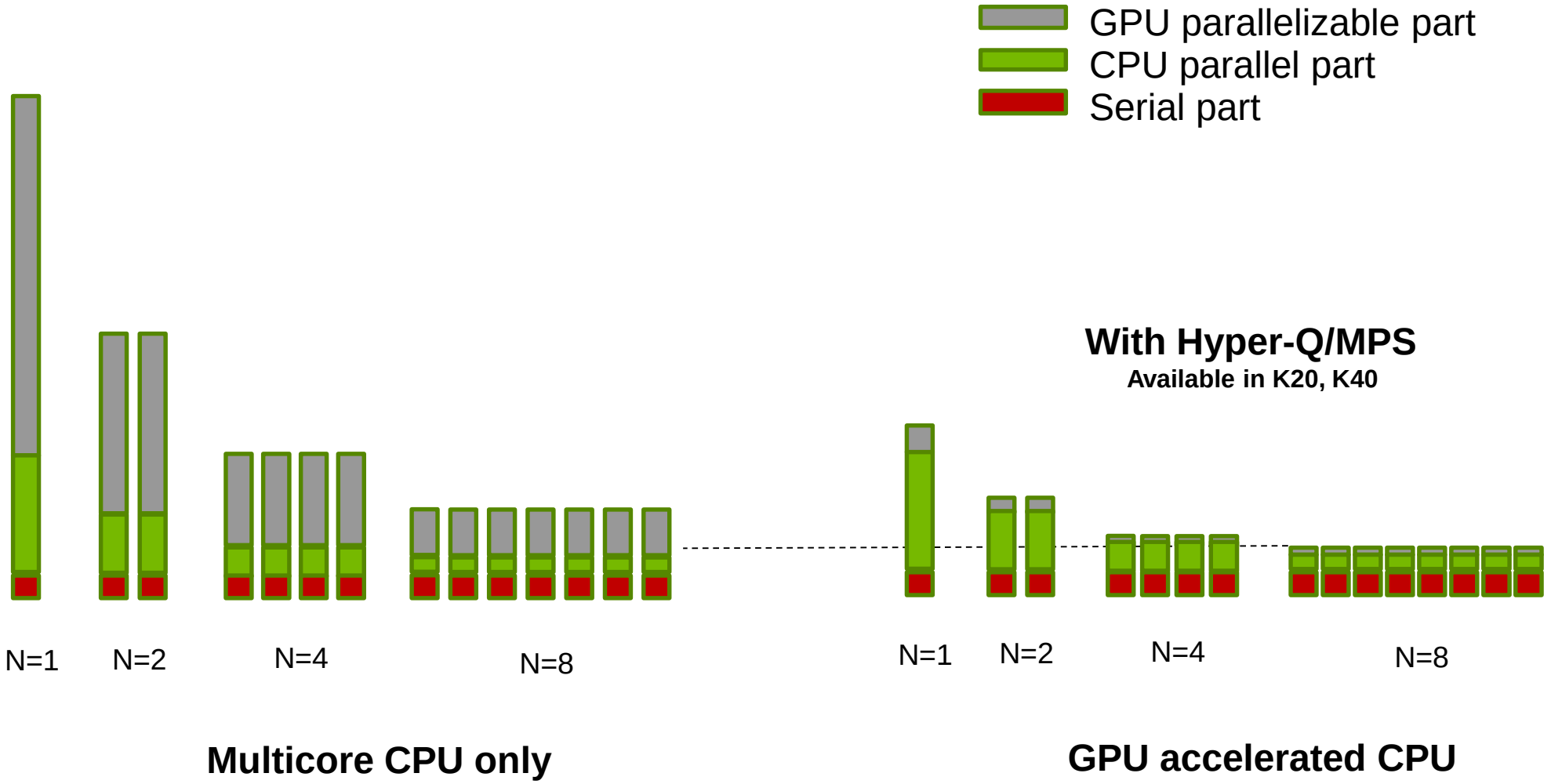
Multicore CPU only

Multicore CPU only

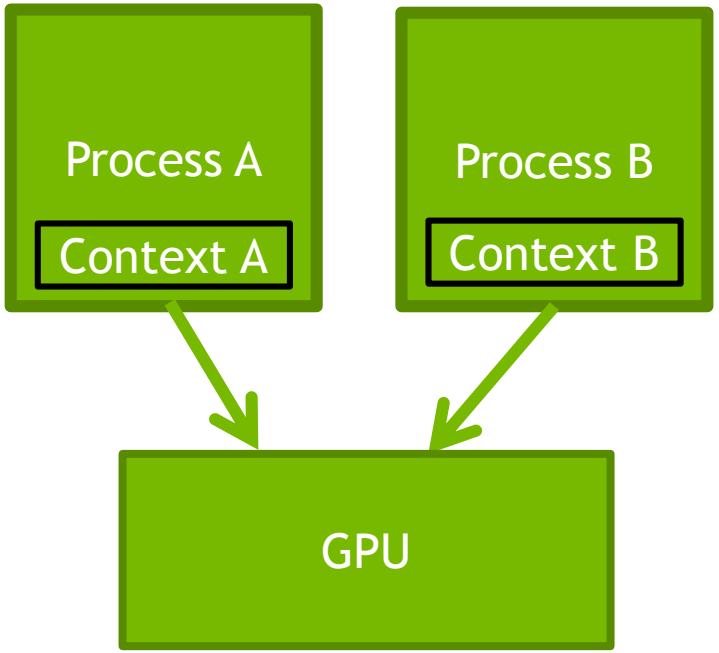


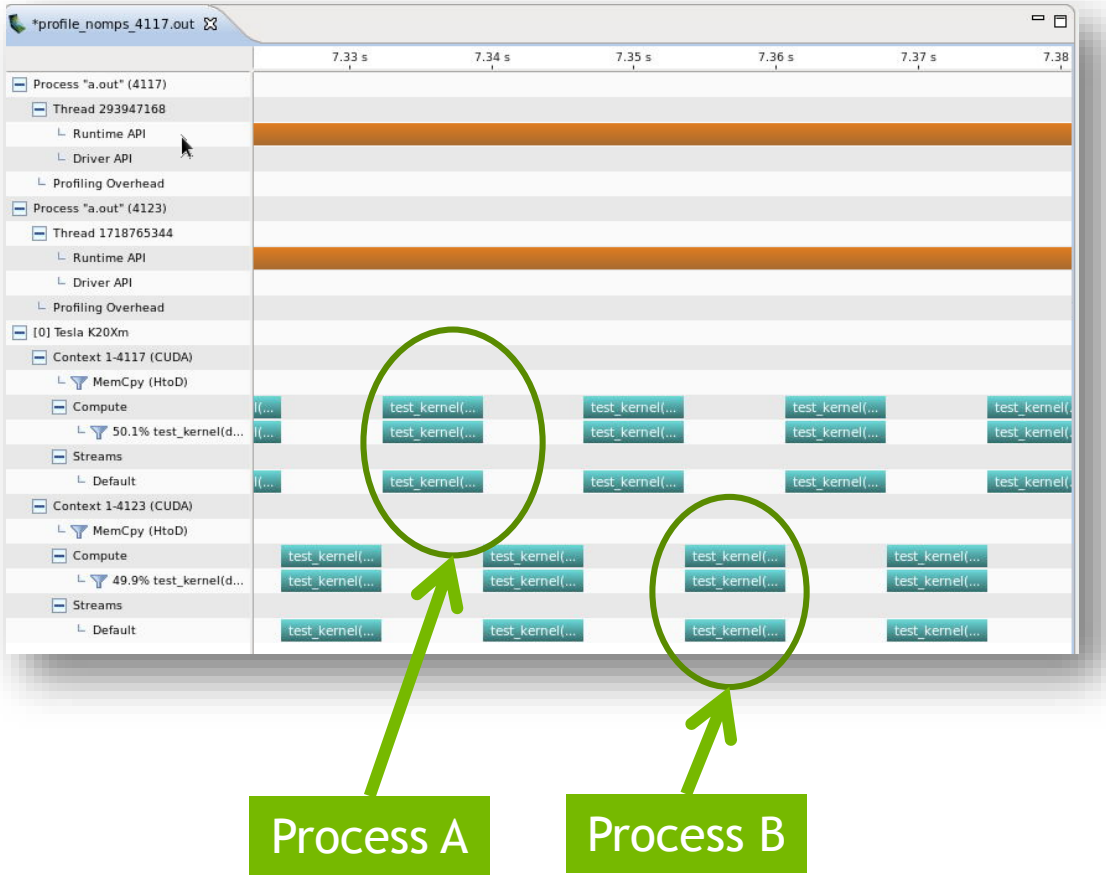
Multicore CPU only



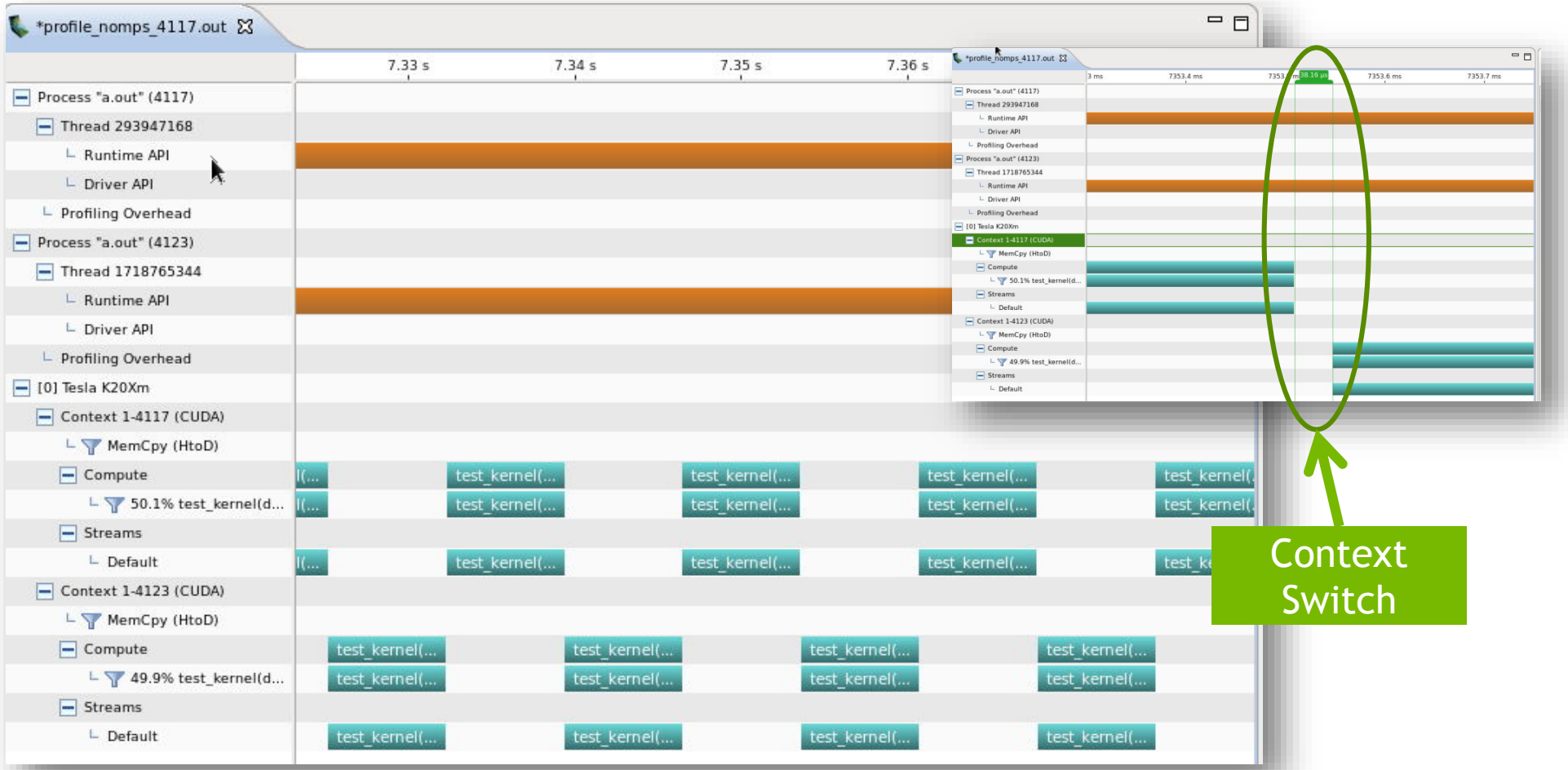


PROCESSES SHARING GPU WITHOUT MPS: NO OVERLAP

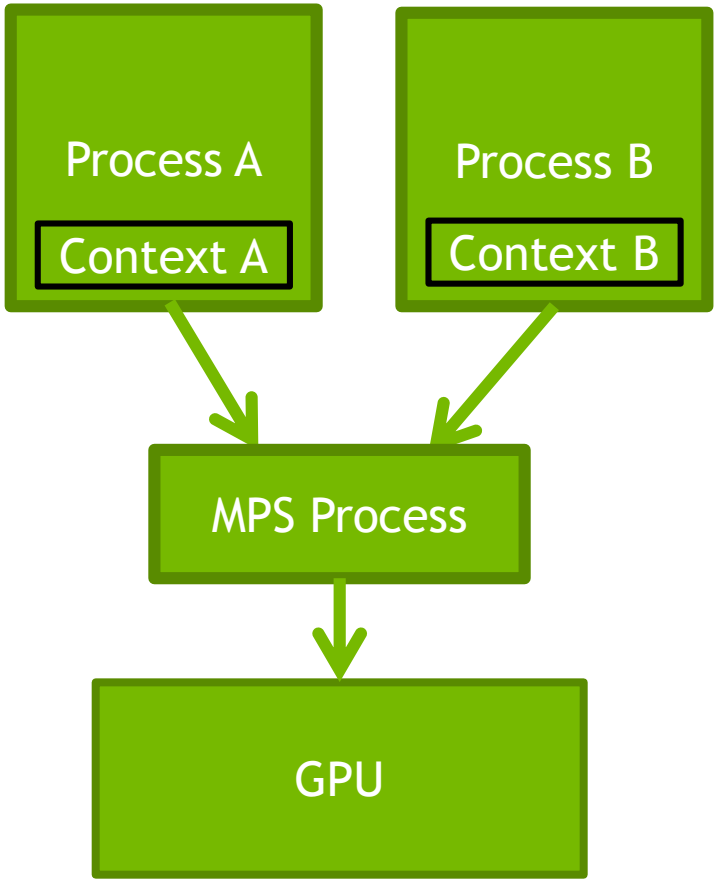


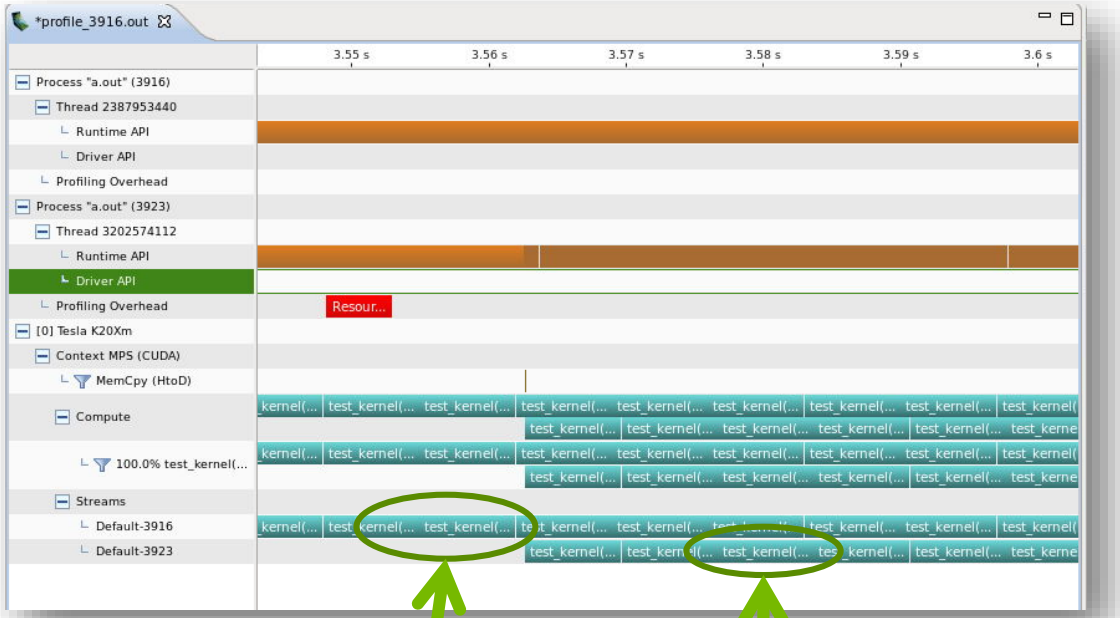


GPU SHARING WITHOUT MPS



PROCESSES SHARING GPU WITH MPS: MAXIMUM OVERLAP

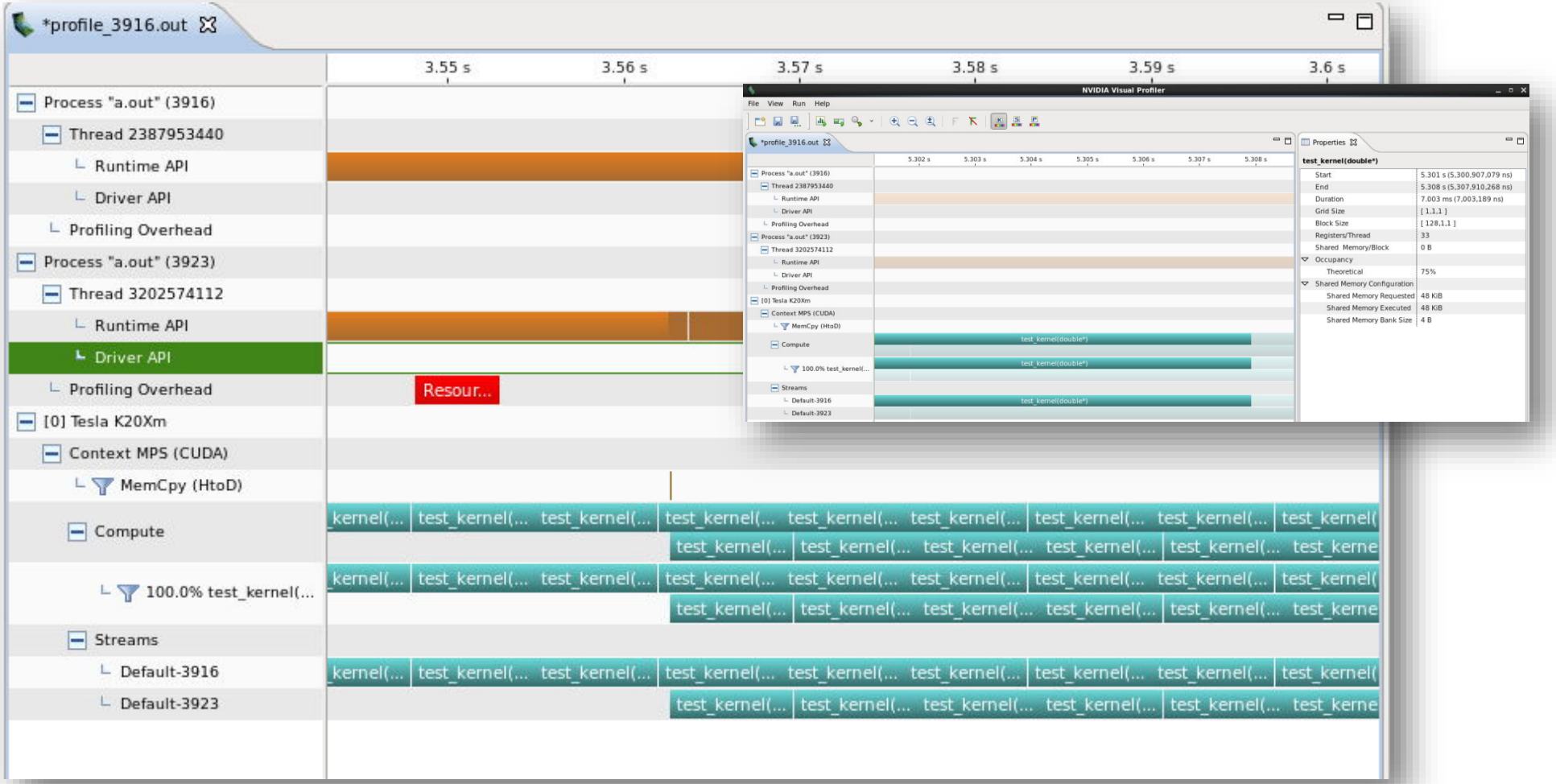




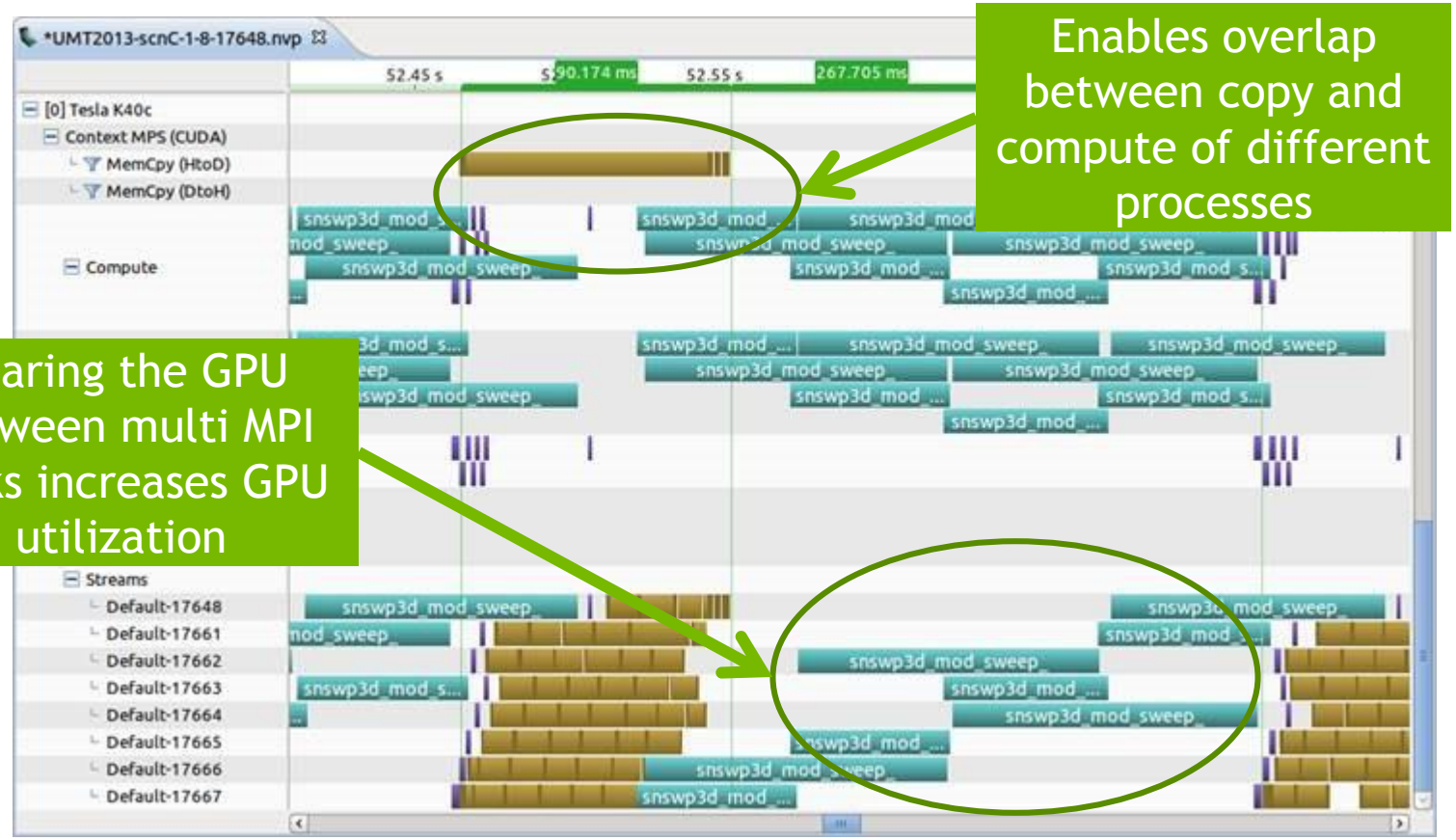
Kernels from Process A

Kernels from Process B

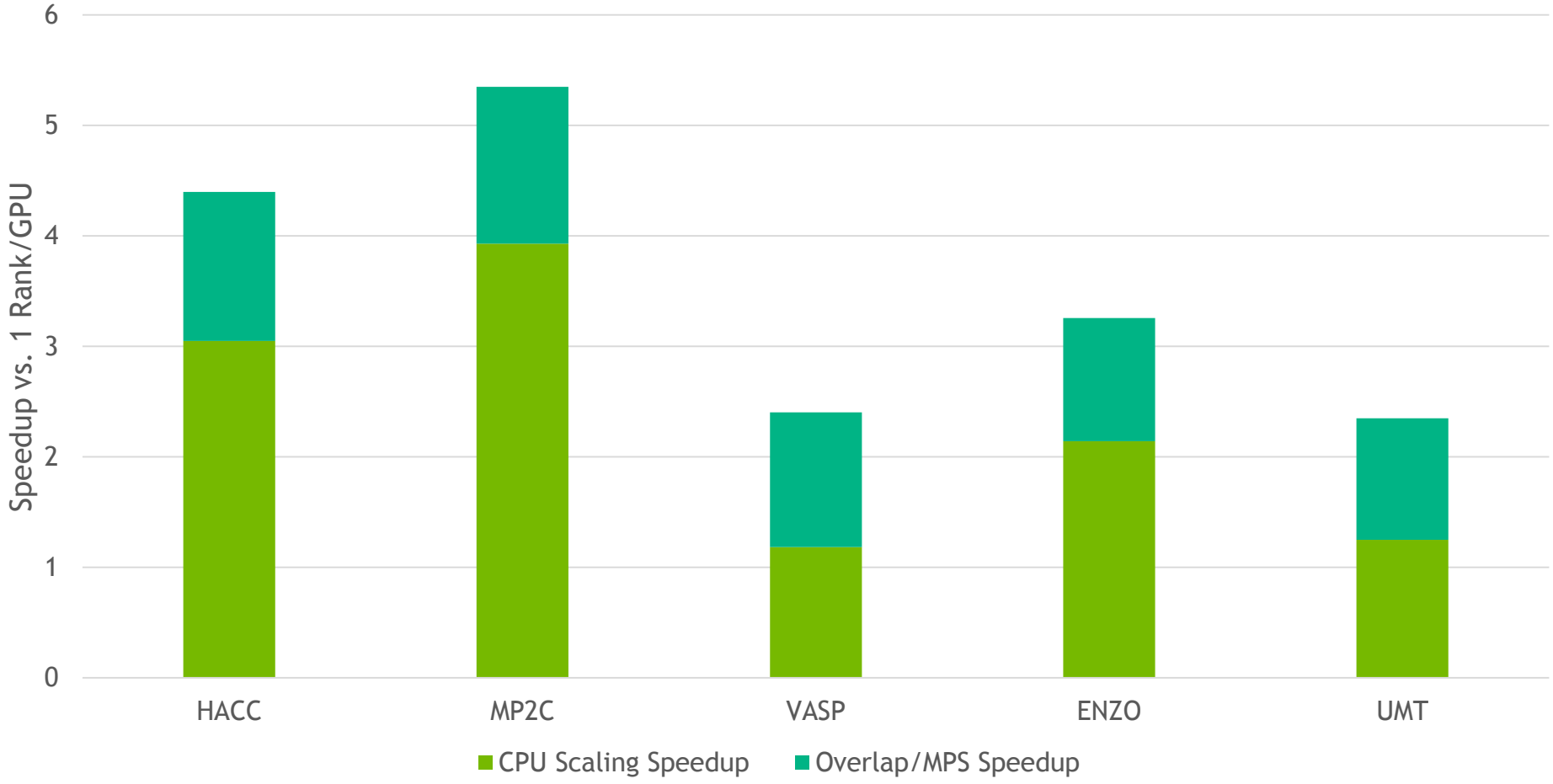
GPU SHARING WITH MPS



CASE STUDY: HYPER-Q/MPS FOR UMT



HYPER-Q/MPS CASE STUDIES



USING MPS

- No application modifications necessary
- Not limited to MPI applications
- MPS control daemon
  - Spawn MPS server upon CUDA application startup

- Typical setup

```
export CUDA_VISIBLE_DEVICES=0
nvidia-smi -i 0 -c EXCLUSIVE_PROCESS
nvidia-cuda-mps-control -d
```

- On Cray XK/XC systems

```
export CRAY_CUDA_MPS=1
```

USING MPS ON MULTI-GPU SYSTEMS

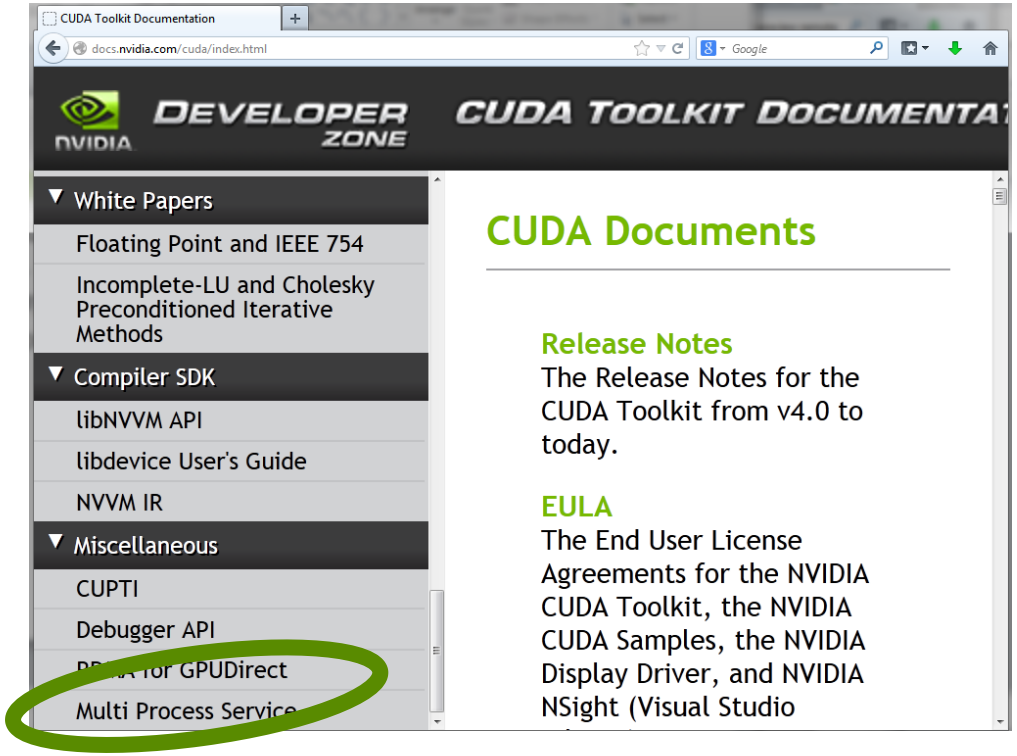
- MPS server only supports a single GPU
  - Use one MPS server per GPU
- Target specific GPU by setting CUDA_VISIBLE_DEVICES
- Adjust pipe/log directory

```
export DEVICE=0
export CUDA_VISIBLE_DEVICES=${DEVICE}
export CUDA_MPS_PIPE_DIRECTORY=${HOME}/mps${DEVICE}/pipe
export CUDA_MPS_LOG_DIRECTORY=${HOME}/mps${DEVICE}/log
cuda_mps_server_control -d
export DEVICE=1    …
```

- More at http://cudamusing.blogspot.de/2013/07/enabling-cuda-multi-process-service-mps.html

MPS SUMMARY

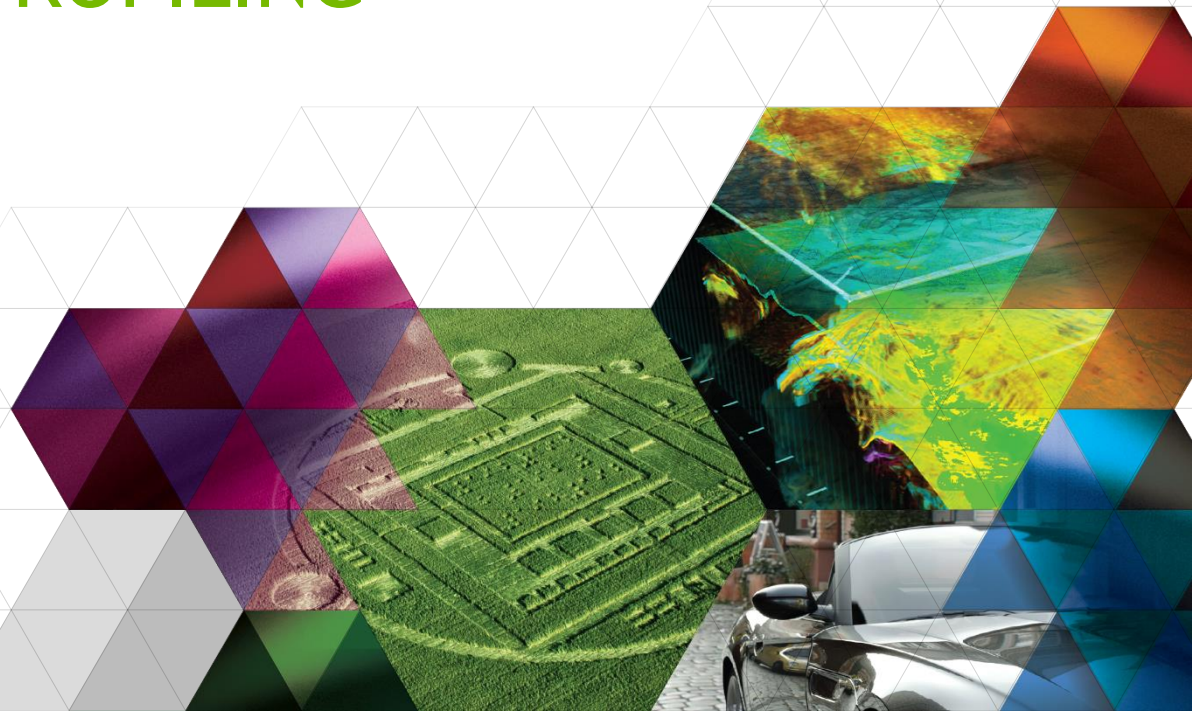
- Easy path to get GPU acceleration for legacy applications
- Enables overlapping of memory copies and compute between different MPI ranks



GPU TECHNOLOGY CONFERENCE

NVIDIA

DEBUGGING AND PROFILING



TOOLS FOR MPI+CUDA APPLICATIONS

- Memory Checking `cuda-memcheck`
- Debugging `cuda-gdb`
- Profiling `nvprof` and NVIDIA Visual Profiler

MEMORY CHECKING WITH CUDA-MEMCHECK

- Cuda-memcheck is a functional correctness checking suite similar to the valgrind memcheck tool
- Can be used in a MPI environment

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

- Problem: output of different processes is interleaved
  - Use save, log-file command line options and launcher script

```
#!/bin/bash
LOG=$1.$OMPI_COMM_WORLD_RANK
#LOG=$1.$MV2_COMM_WORLD_RANK
cuda-memcheck --log-file $LOG.log --save $LOG.memcheck $*

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

MEMORY CHECKING WITH CUDA-MEMCHECK

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

```
[jkraus@sb077 bin]$ MV2_USE_CUDA=1 mpiexec -np 4 ./cuda-memcheck-script.sh ./jacobi_cuda_aware_m
pi_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:
Error: CUDA result "unspecified launch failure" for call "cudaDeviceSynchronize()" in file "Host
.c" at line 453. Terminating...

=====================================================================================
=   BAD TERMINATION OF ONE OF YOUR APPLICATION PROCESSES
=   EXIT CODE: 255
=   CLEANING UP REMAINING PROCESSES
=   YOU CAN IGNORE THE BELOW CLEANUP MESSAGES
=====================================================================================
[jkraus@sb077 bin]$ ls *.memcheck
jacobi_cuda_aware_mpi_async.0.memcheck  jacobi_cuda_aware_mpi_async.2.memcheck
jacobi_cuda_aware_mpi_async.1.memcheck  jacobi_cuda_aware_mpi_async.3.memcheck
[jkraus@sb077 bin]$ cuda-memcheck --read jacobi_cuda_aware_mpi_async.0.memcheck
```

MEMORY CHECKING WITH CUDA-MEMCHECK

Read outputfiles with `cuda-memcheck --read`

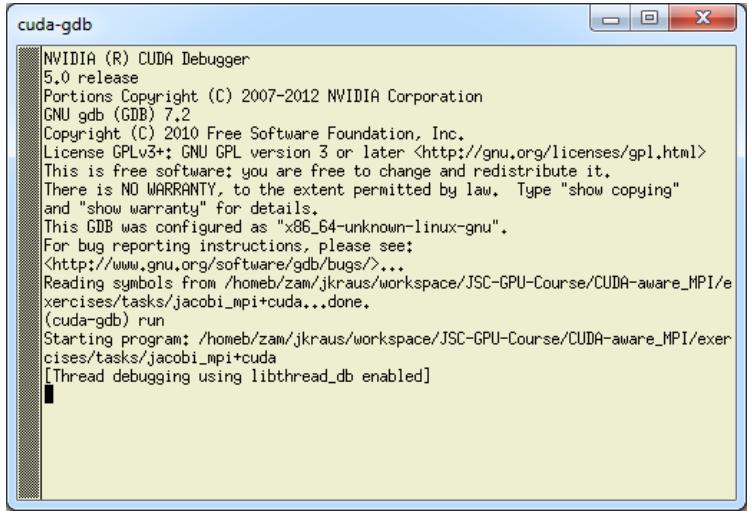
```
jkraus@sb077:~/workspace/Jacobi/main/bin
=========     Saved host backtrace up to driver entry point at kernel launch time
=========     Host Frame:/usr/lib64/libcuda.so.1 (cuLaunchKernel + 0x331) [0x138251]
=========     Host Frame:/shared/apps/cuda/CUDA-v6.0.26/lib64/libcudart.so.6.0 [0xfa98]
=========     Host Frame:/shared/apps/cuda/CUDA-v6.0.26/lib64/libcudart.so.6.0 (cudaLaunch + 0x1
43) [0x2e6a3]
=========     Host Frame:./jacobi_cuda_aware_mpi_async [0x34bd]
=========     Host Frame:./jacobi_cuda_aware_mpi_async [0x4067]
=========     Host Frame:./jacobi_cuda_aware_mpi_async [0x3073]
=========     Host Frame:./jacobi_cuda_aware_mpi_async [0x1aff]
=========     Host Frame:/lib64/libc.so.6 (__libc_start_main + 0xfd) [0x1ecdd]
=========     Host Frame:./jacobi_cuda_aware_mpi_async [0x16a5]
=========
========= Invalid __global__ read of size 8
=========     at 0x00000888 in /home-2/jkraus/workspace/Jacobi/main/src/Async/Device.cu:145:Jaco
biComputeKernel(double const *, double*, double*, int2, int, int2, int2)
=========     by thread (1,12,0) in block (4,63,0)
=========     Address 0x2300a04210 is out of bounds
=========     Saved host backtrace up to driver entry point at kernel launch time
=========     Host Frame:/usr/lib64/libcuda.so.1 (cuLaunchKernel + 0x331) [0x138251]
=========     Host Frame:/shared/apps/cuda/CUDA-v6.0.26/lib64/libcudart.so.6.0 [0xfa98]
=========     Host Frame:/shared/apps/cuda/CUDA-v6.0.26/lib64/libcudart.so.6.0 (cudaLaunch + 0x1
43) [0x2e6a3]
=========     Host Frame:./jacobi_cuda_aware_mpi_async [0x34bd]
=========     Host Frame:./jacobi_cuda_aware_mpi_async [0x4067]
```

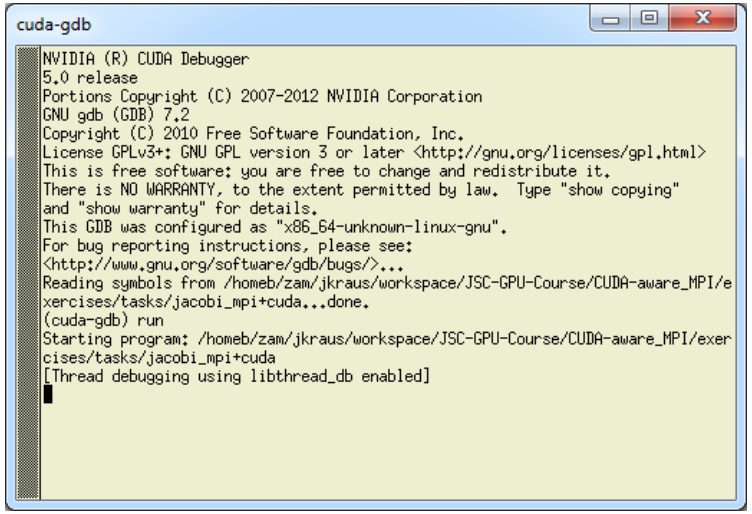
DEBUGGING MPI+CUDA APPLICATIONS

USING CUDA-GDB WITH MPI APPLICATIONS

- You can use cuda-gdb just like gdb with the same tricks
- For smaller applications, just launch xterms and cuda-gdb

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





DEBUGGING MPI+CUDA APPLICATIONS

CUDA-GDB ATTACH

- CUDA 5.0 and forward have the ability to attach to a running process

```
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 (per process)
Measured FLOPS: 24.06 GFLOPS (total), 6.01 GFLOPS (per process)
Measured device bandwidth: 230.95
[jkraus@sb077 bin]$ CUDA_DEVICE_WA
aware_mpi_async -t 2 2 -d 1024 102
Topology size: 2 x 2
Local domain size (current node):
Global domain size (all nodes): 20
Starting Jacobi run with 4 process
sb077: The application encountered
 can now attach a debugger to the
sb077: The application encountered
 can now attach a debugger to the
sb077: The application encountered
 can now attach a debugger to the
sb077: The application encountered
 can now attach a debugger to the
```

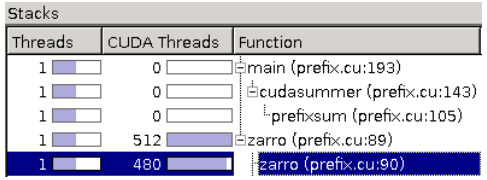
jkraus@sb077:~/workspace/Jacobi/main/bin

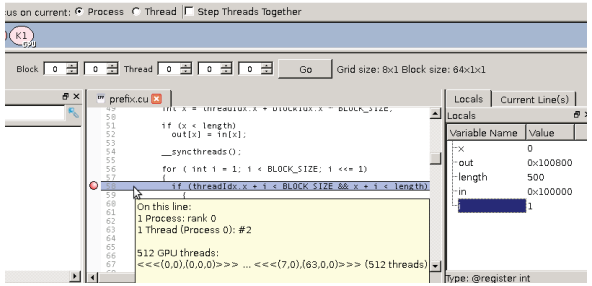
```
Reading symbols from /usr/lib64/libnes-rdmav2.so...(no debugging symbols found)...done.
Loaded symbols for /usr/lib64/libnes-rdmav2.so
Reading symbols from /usr/lib64/libmlx4-rdmav2.so...(no debugging symbols found)...done.
Loaded symbols for /usr/lib64/libmlx4-rdmav2.so
Reading symbols from /usr/lib64/libipathverbs-rdmav2.so...(no 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)
```

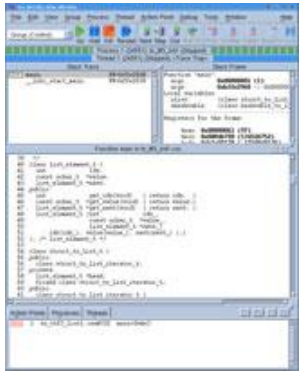
DEBUGGING MPI+CUDA APPLICATIONS

THIRD PARTY TOOLS

- Allinea DDT debugger
- Totalview
- S4284 - Debugging PGI CUDA Fortran and OpenACC on GPUs with Allinea DDT - Tuesday 4pm LL20D







DDT: THREAD LEVEL DEBUGGING

Focus

Breakpoint inside kernel

Launch configuration

Per thread variabless

Allinea DDT 4.2-34071 (on nid00015)

File View Control Search Tools Window Help

Focus on current: Process Thread Step Threads Together Step CUDA threads by: Warp (default)

Threads (1 hidden): 1 K1 GPU

CUDA Threads (gpuTranspose_kernel) Block 0 0 0 Thread 5 2 0 Go Grid size: 32x32x1 Block size: 32x32x1

main.cu

```
42
43    // Stage Matrix tile to Shared memory
44    tile[threadIdx.y][threadIdx.x] = in[j*cols+i];
45
46    __syncthreads();
47
48    i = blockIdx.y * blockDim.y + threadIdx.x;
49    j = blockIdx.x * blockDim.x + threadIdx.y;
51    // Write matrix tile back out, transposed
52    out[j*rows+i] = tile[threadIdx.x][threadIdx.y];
53  }
54  void gpuTranspose(int rows, int cols, float * in, float * out) {
55    gpuTranspose_kernel<<<dim3(rows/32,cols/32),dim3(32,32)>>>(rows,cols,in,out);
56    cudaCheckError();
57  }
58
59
60  int main(int argc, char** argv)
61  {
```

Locals | Current Line(s) | Current Stack | GPU Devices

Current Line(s)

Variable Name	Value
i	5
j	2
out	0xb0060
rows	1024
threadIdx	{x = 5, y = 2, z = 0}
threadIdx.x	5
threadIdx.y	2
tile	

Type: none selected

Inp... Bre... Wat... Stacks Ker... Tra...

Input/Output

```
Process 0: OMP threads: 1
Process 0: CPU+OMP: Kernel bandwidth: 1.106102 gb/sec
Process 0: err = 0
```

Type here ('Enter' to send): More

Evaluate

Expression	Value

Ready

PROFILING MPI+CUDA APPLICATIONS

USING NVPROF+NVVP

3 Usage modes:

- Embed pid in output filename

```
mpirun -np 2 nvprof --output-profile profile.out.%p
```

- Only save the textual output

```
mpirun -np 2 nvprof --log-file profile.out.%p
```

- Collect profile data on all processes that run on a node

```
nvprof --profile-all-processes -o profile.out.%p
```

PROFILING MPI+CUDA APPLICATIONS

USING NVPROF+NVVP

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

```
[jkraus@sb077 bin]$ MV2_USE_CUDA=1 mpiexec -np 4 nvprof -o jacobi_cuda_aware_mpi_async.%p.nvvp
./jacobi_cuda_aware_mpi_async -t 2 2 -d 1024 1024 -fs
==29287== NVPROF is profiling process 29287, command: ./jacobi_cuda_aware_mpi_async -t 2 2 -d 10
24 1024 -fs
==29282== NVPROF is profiling process 29282, command: ./jacobi_cuda_aware_mpi_async -t 2 2 -d 10
24 1024 -fs
==29288== NVPROF is profiling process 29288, command: ./jacobi_cuda_aware_mpi_async -t 2 2 -d 10
24 1024 -fs
==29284== NVPROF is profiling process 29284, command: ./jacobi_cuda_aware_mpi_async -t 2 2 -d 10
24 1024 -fs
```

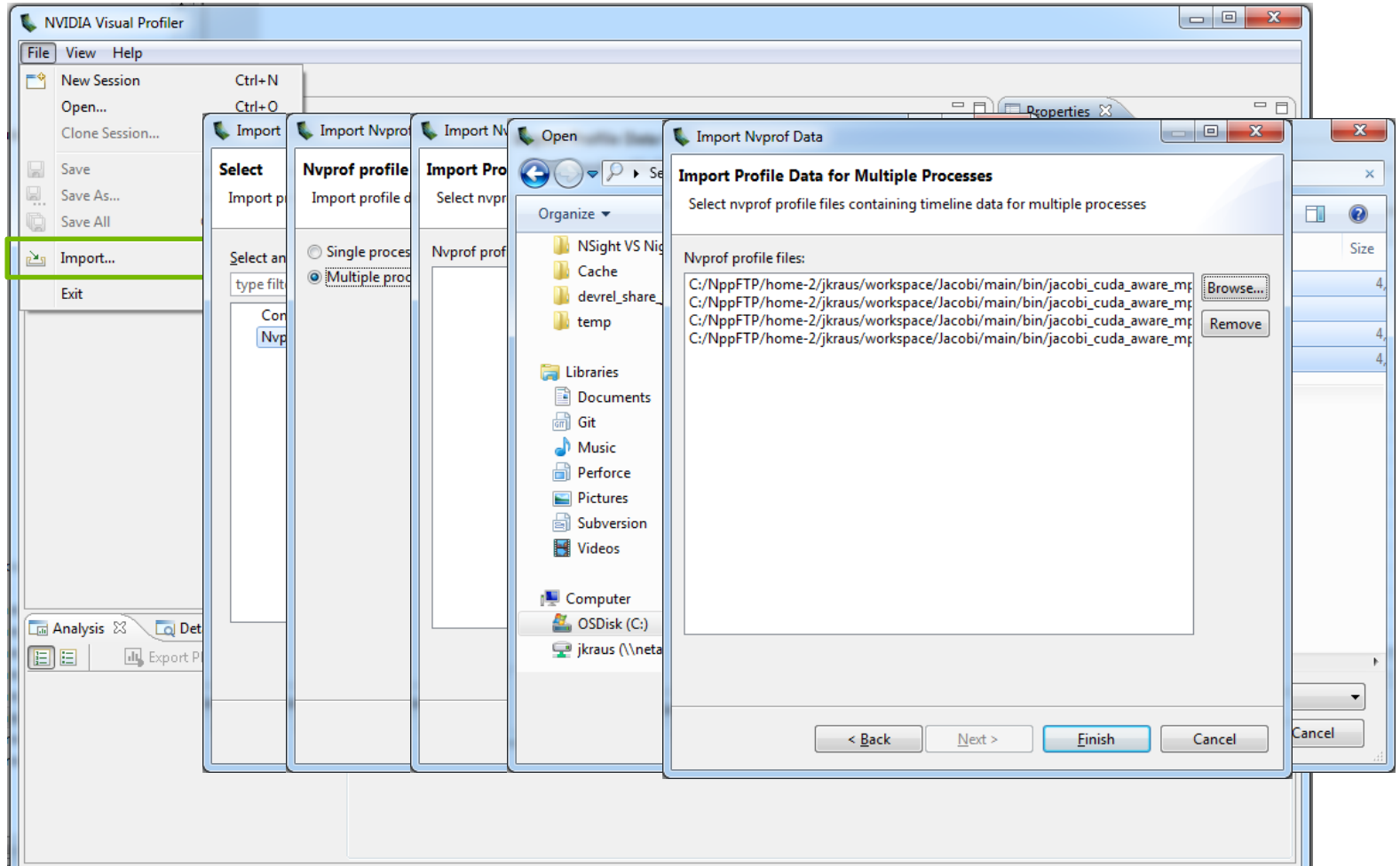
jkraus@sb077:~/workspace/Jacobi/main/bin

```
Iteration: 100 - Residue: 0.002394
Iteration: 200 - Residue: 0.001202
Iteration: 300 - Residue: 0.000801
Iteration: 400 - Residue: 0.000601
Iteration: 500 - Residue: 0.000481
Iteration: 600 - Residue: 0.000400
Iteration: 700 - Residue: 0.000343
Iteration: 800 - Residue: 0.000300
Iteration: 900 - Residue: 0.000266
Stopped after 1000 iterations with residue 0.000240
Total Jacobi run time: 0.4353 sec.
Average per-process communication time: 0.2639 sec.
Measured lattice updates: 9.62 GLU/s (total), 2.40 GLU/s (per process)
Measured FLOPS: 48.08 GFLOPS (total), 12.02 GFLOPS (per process)
Measured device bandwidth: 461.59 GB/s (total), 115.40 GB/s (per process)
==29284== Generated result file: /home-2/jkraus/workspace/Jacobi/main/bin/jacobi_cuda_aware_mpi_
async.29284.nvvp
==29287== Generated result file: /home-2/jkraus/workspace/Jacobi/main/bin/jacobi_cuda_aware_mpi_
async.29287.nvvp
==29288== Generated result file: /home-2/jkraus/workspace/Jacobi/main/bin/jacobi_cuda_aware_mpi_
async.29288.nvvp
==29282== Generated result file: /home-2/jkraus/workspace/Jacobi/main/bin/jacobi_cuda_aware_mpi_
async.29282.nvvp
[jkraus@sb077 bin]$
```

GPU TECHNOLOGY CONFERENCE

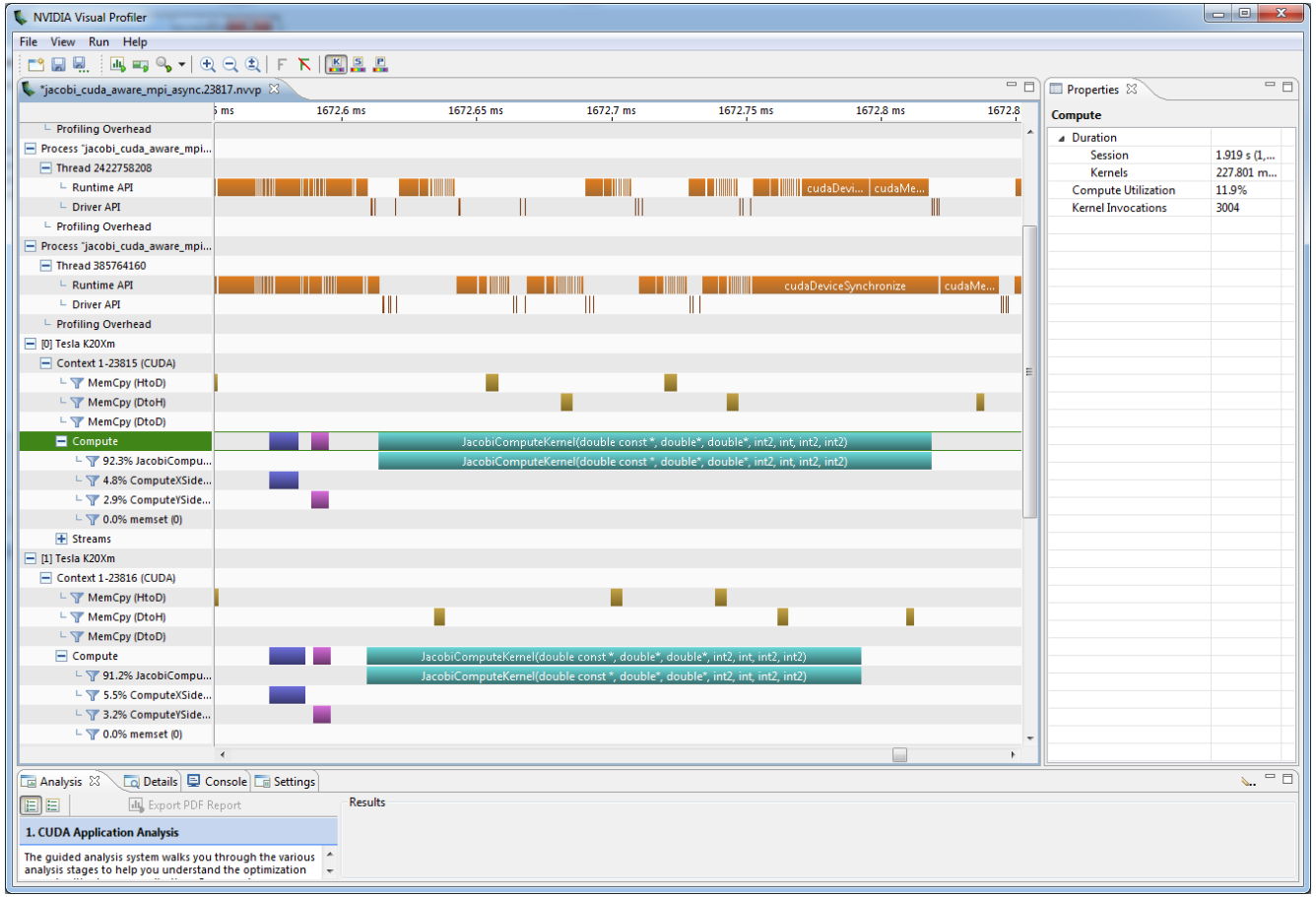
PROFILING MPI+CUDA APPLICATIONS

USING NVPROF+NVVP



PROFILING MPI+CUDA APPLICATIONS

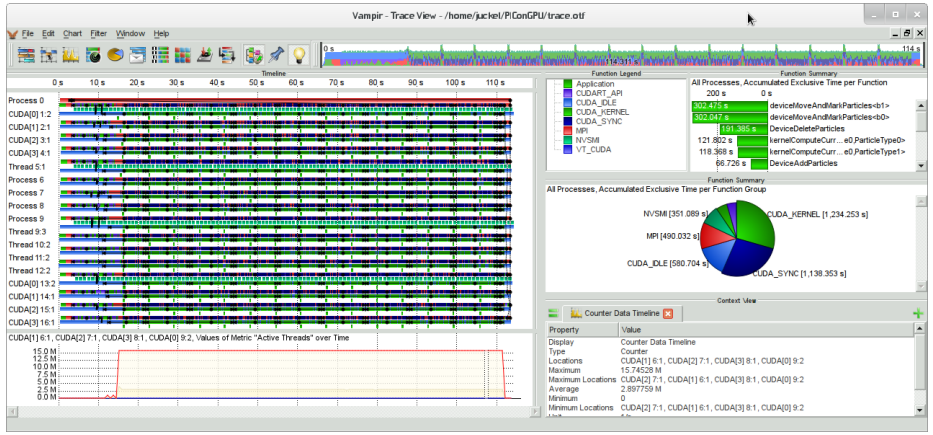
USING NVPROF+NVVP

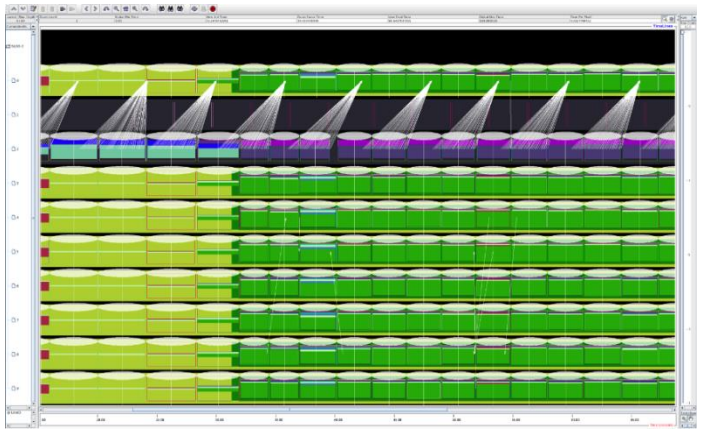


PROFILING MPI+CUDA APPLICATIONS

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

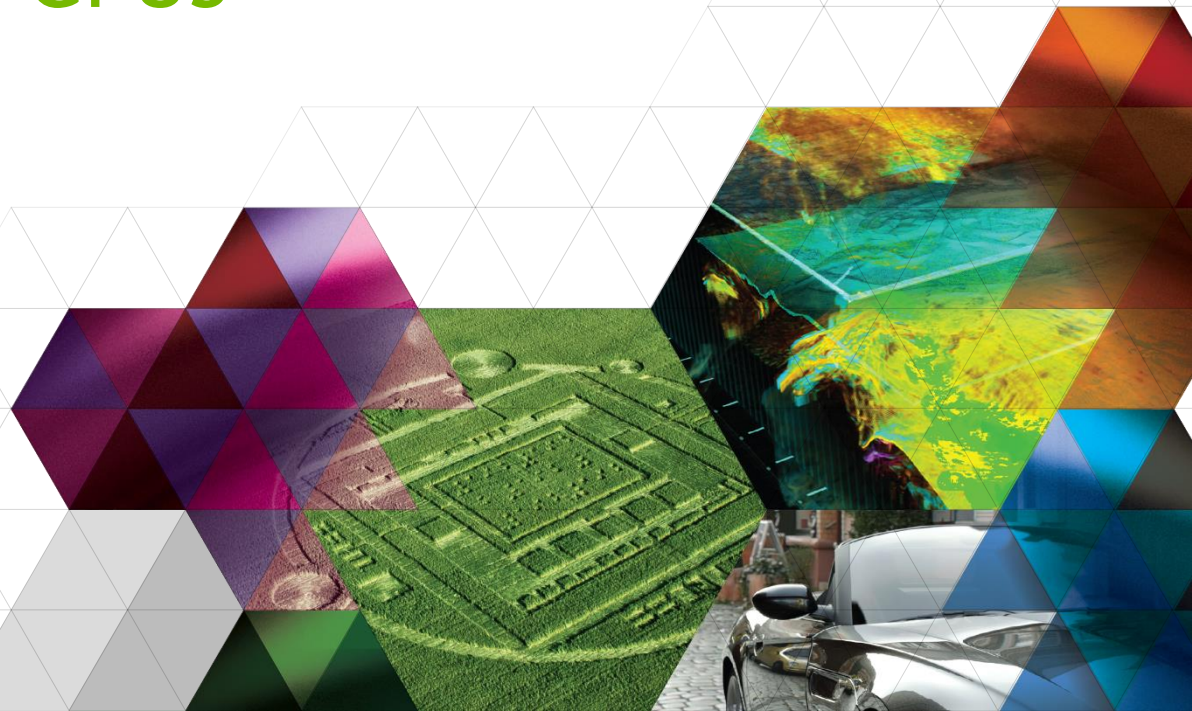




GPU TECHNOLOGY CONFERENCE

NVIDIA

ADVANCED MPI ON GPUS



BEST PRACTICE: USE NONE-BLOCKING MPI

BLOCKING

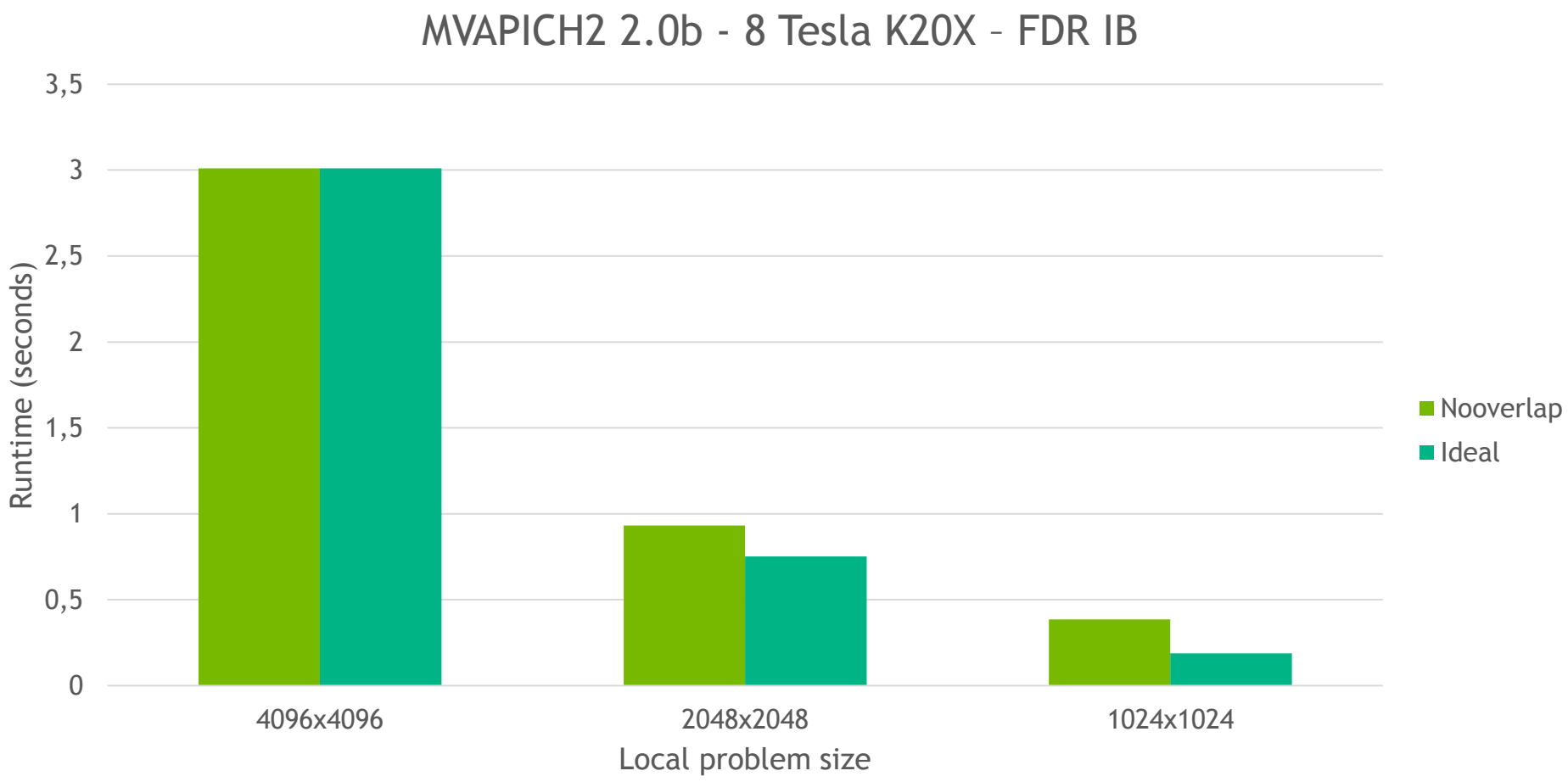
```
#pragma acc host_data use_device ( u_new ) {
MPI_Sendrecv(u_new+offset_first_row, m-2, MPI_DOUBLE, t_nb, 0,
             u_new+offset_bottom_bondary, m-2, MPI_DOUBLE, b_nb, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
MPI_Sendrecv(u_new+offset_last_row, m-2, MPI_DOUBLE, b_nb, 1,
         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

NONE-BLOCKING

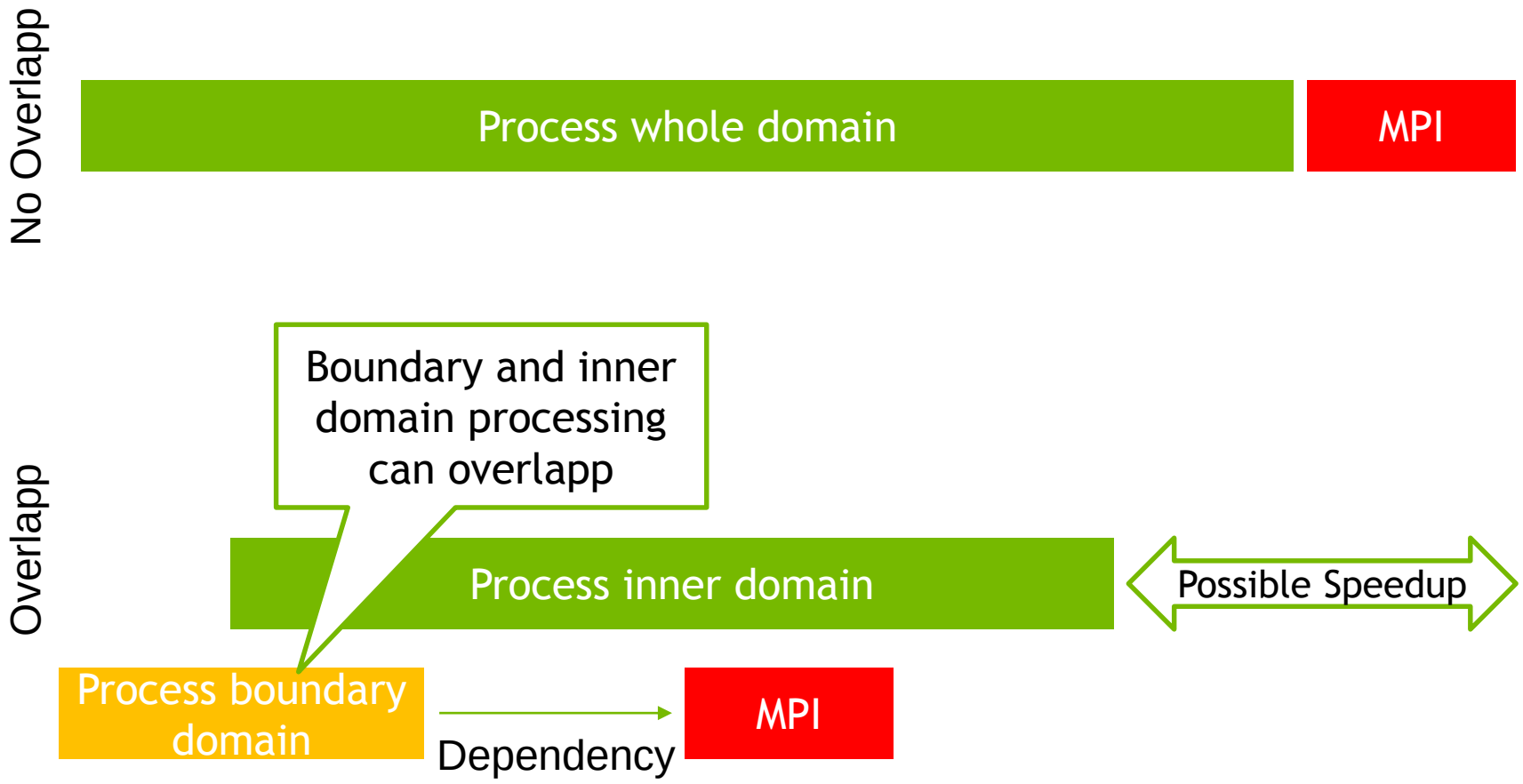
```
MPI_Request t_b_req[4];
#pragma acc host_data use_device ( u_new ) {
  MPI_Irecv(u_new+offset_top_bondary,m-2                                  req);
  MPI_Irecv(u_new+offset_bottom_bondar                                  WORLD,t_b_req+1);
  MPI_Isend(u_new+offset_last_row,m-2,M                                  COMM_WORLD,t_b_req+2);
  MPI_Isend(u_new+offset_first_row,m-2,M                                  nb,1,MPI_COMM_WORLD,t_b_req+3);
}
MPI_Waitall(4, t_b_req, MPI_STATUSES_IGNORE);
```

Gives MPI more opportunities to build efficient piplines

OVELAPPING COMMUNICAITON AND COMPUTATION



OVERLAPPING COMMUNICATION AND COMPUTATION



OVERLAPPING COMMUNICATION AND COMPUTATION

OpenACC

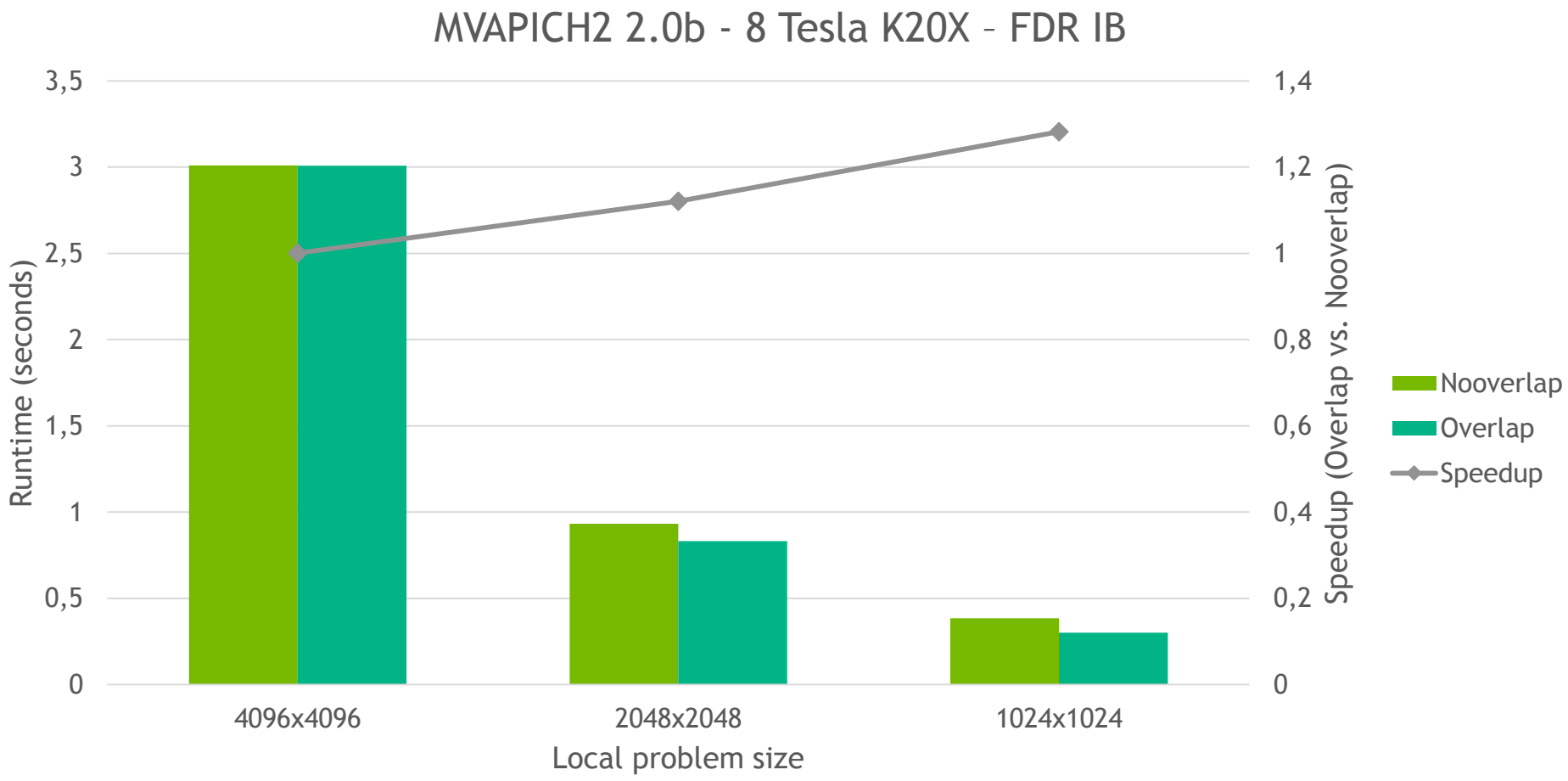
```
#pragma acc parallel loop present ( u_new, u, to_left, to_right ) async(1)
for ( ... )
      //Process boundary and pack to_left and to_right
#pragma acc parallel loop present ( u_new, u ) async(2)
for ( ... )
      //Process inner domain
#pragma acc wait(1)                //wait for boundary
MPI_Request 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 )
for ( ... )
      //unpack from_left and from_right
#pragma acc wait                   //wait for iteration to finish
```

OVERLAPPING COMMUNICATION AND COMPUTATION

CUDA

```
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>>>(u_new_d, from_left_d, from_right_d, n, m);



cudaDeviceSynchronize();          //wait for iteration to finish
```

OVELAPPING COMMUNICAITON AND COMPUTATION



MPI AND UNIFIED MEMORY

- Unified Memory support for CUDA-aware MPI needs changes to the MPI implementations
  - Check with your MPI implementation of choice for their plans
  - It might work in some situations but it is not supported
- Unified Memory and regular MPI
  - Require unmanaged staging buffers
    - Regular MPI has no knowledge of managed memory
    - CUDA 6 managed memory does not play well with RDMA protocols

HANDLING MULTI GPU NODES

- Multi GPU nodes and GPU-affinity:
  - Use local rank:

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

  - Use exclusive process mode + `cudaSetDevice(0)`

HANDLING MULTI GPU NODES

- How to determine local rank:
  - Rely on process placement (with one rank per GPU)

```
int rank = 0;
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
int num_devices = 0;
cudaGetDeviceCount(&num_devices); // num_devices == ranks per node
int local_rank = rank % num_devices;
```

  - Use environment variables provided by MPI launcher
    - e.g for OpenMPI

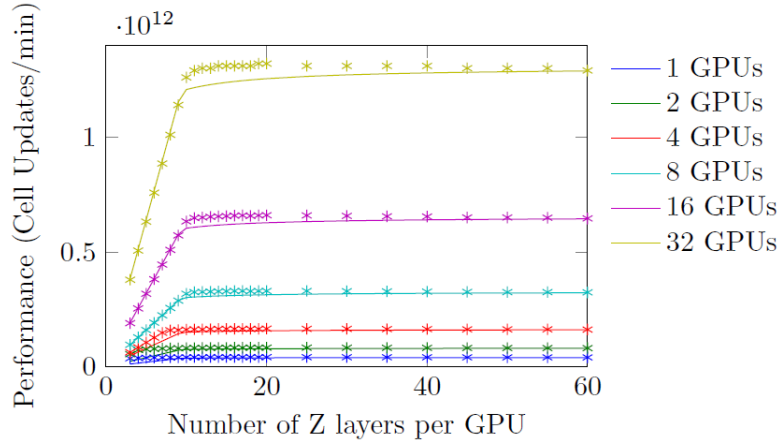
```
int local_rank = atoi(getenv("OMPI_COMM_WORLD_LOCAL_RANK"));
```

    - e.g. For MVPAICH2

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

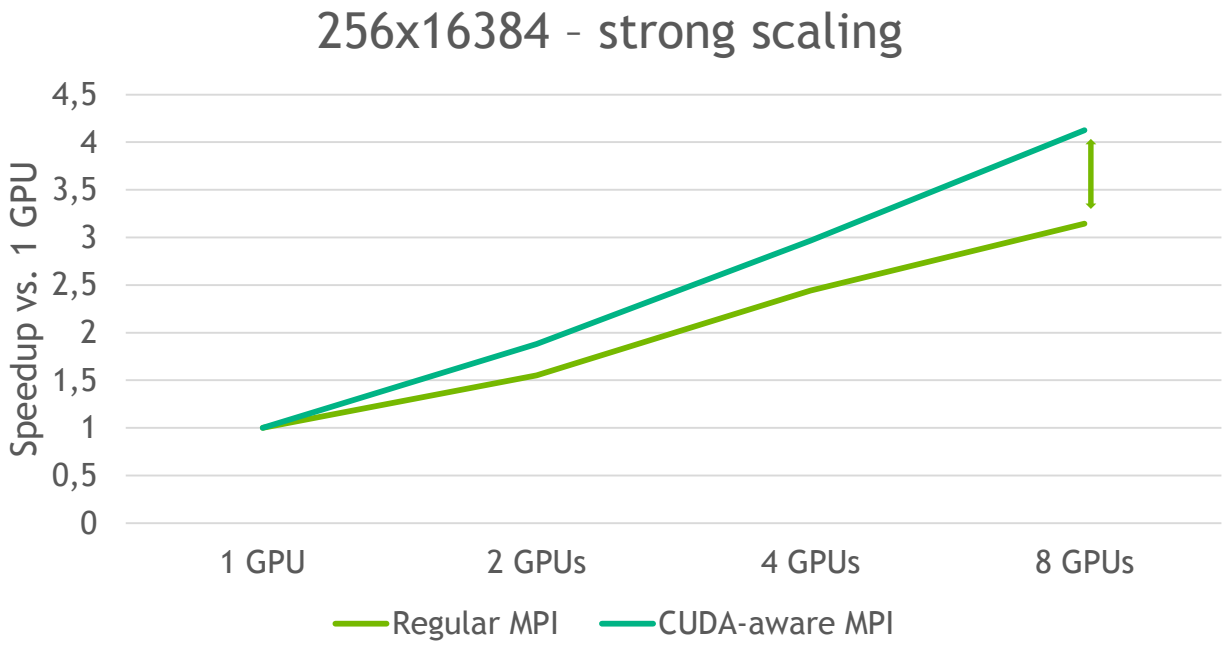
CASE STUDY: B-CALM

- CUDA-aware MPI enabled a easy transition from single node multi GPU to multi node multi GPU
- Multi node multi GPU version allows to solve problems that could not be tackled before
- More info at: S4190 - Finite Difference Simulations on GPU Clusters: How Far Can You Push 1D Domain Decomposition? (Wed. 03/26)



CASE STUDY: LBM D2Q37

- CUDA-aware MPI improves strong scalability and simplifies programming



- More info at: S4186 - Optimizing a LBM code for Compute Clusters with Kepler GPUs (Wed. 03/26)

CONCLUSIONS

- Using MPI as abstraction layer for Multi GPU programming allows multi GPU programs to scale beyond a single node
  - CUDA-aware MPI delivers ease of use, reduced network latency and increased bandwidth
- All NVIDIA tools are usable and third party tools are available
- Multipe CUDA-aware MPI implementations available
  - OpenMPI, MVAPICH2, Cray, IBM Platform MPI
- Other intressting sessions:
  - S4517 - Latest Advances in MVAPICH2 MPI Library for NVIDIA GPU Clusters with InfiniBand – Tuesday 3pm LL21A
  - S4589 - OpenMPI with RDMA Support and CUDA – Thursday 2pm 211B

OVERLAPPING COMMUNICATION AND COMPUTATION – TIPS AND TRICKS

- CUDA-aware MPI might use the default stream
  - Allocate stream with the non-blocking flag (cudaStreamNonBlocking)
  - More info: S4158 - CUDA Streams: Best Practices and Common Pitfalls Tuesday 03/27)
- In case of multiple kernels for boundary handling the kernel processing the inner domain might sneak in
  - Use single stream or events for inter stream dependencies via cudaStreamWaitEvent (#pragma acc wait async) – disables overlapping of boundary and inner domain kernels
  - Use high priority streams for boundary handling kernels – allows overlapping of boundary and inner domain kernels
- As of CUDA 6.0 GPUDirect P2P in multi process can overlap disable it for older releases

HIGH PRIORITY STREAMS

- Improve sclability with high priority streams (cudaStreamCreateWithPriority)
  - S4158 - CUDA Streams: Best Practices and Common Pitfalls (Thu. 03/27)
- Use-case MD Simulations:
  - S4465 - Optimizing CoMD: A Molecular Dynamics Proxy Application Study (Wed. 03/26)