GPU Programming Workshop

Session 3: Advanced topics in OpenACC+MPI

Special Technical Projects Team

March 3, 2021

UCAR

Recap

- Matrix multiplication with shared memory
- OpenACC and CUDA coding practice
  - PCAST
  - FMA flags
  - Nsys profiling

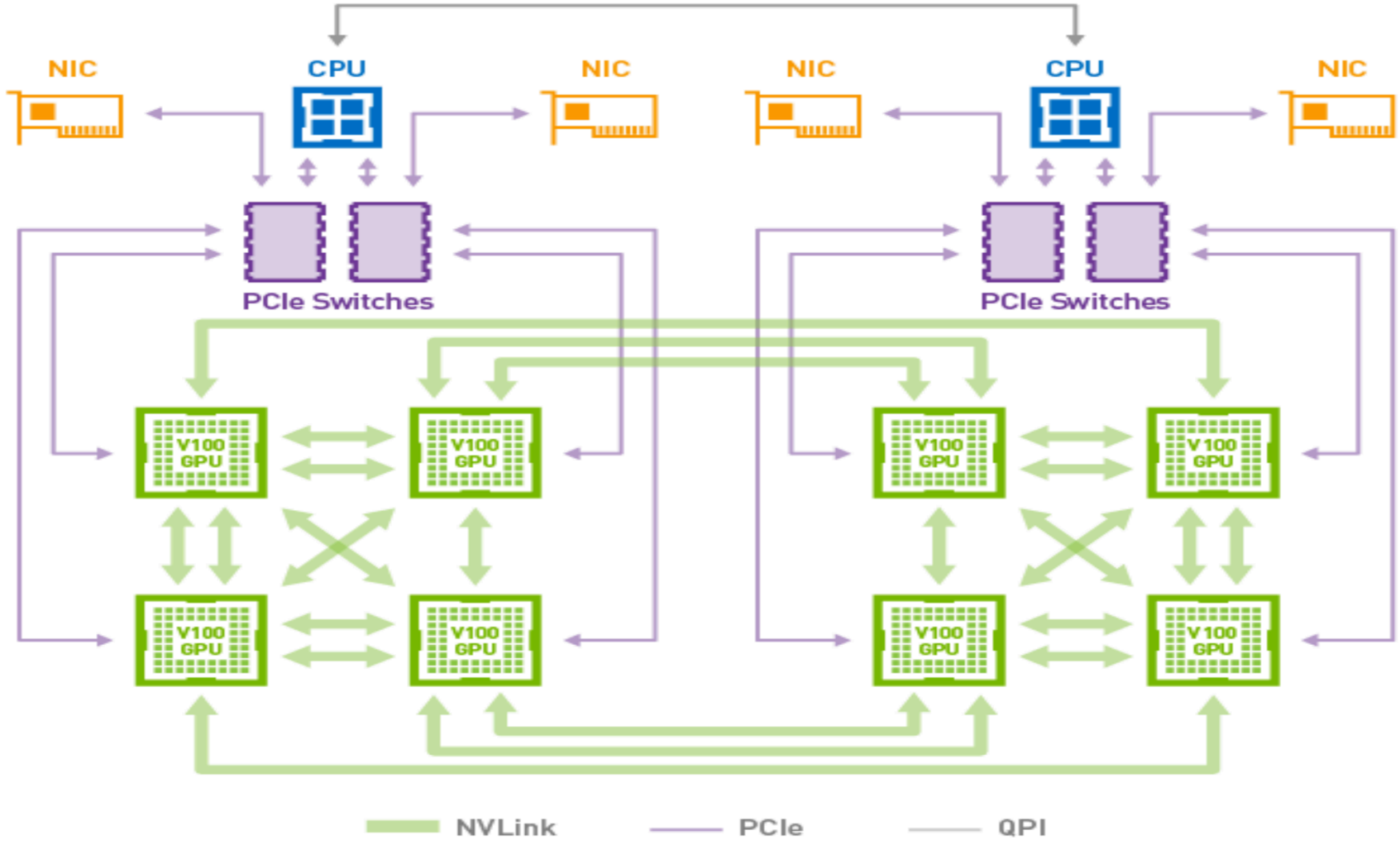
Before going further, clone the workshop code, slides, and reference materials using:
git clone https://github.com/NCAR/GPU_workshop.git

Overview

- OpenACC + MPI
  - Introduction and terminologies
- OpenACC + MPI stencil operation with MPI
  - Profiling MPI applications
  - Coding practice

Before going further, clone the workshop code, slides, and reference materials using:
git clone https://github.com/NCAR/GPU_workshop.git

Multi-GPU node architecture

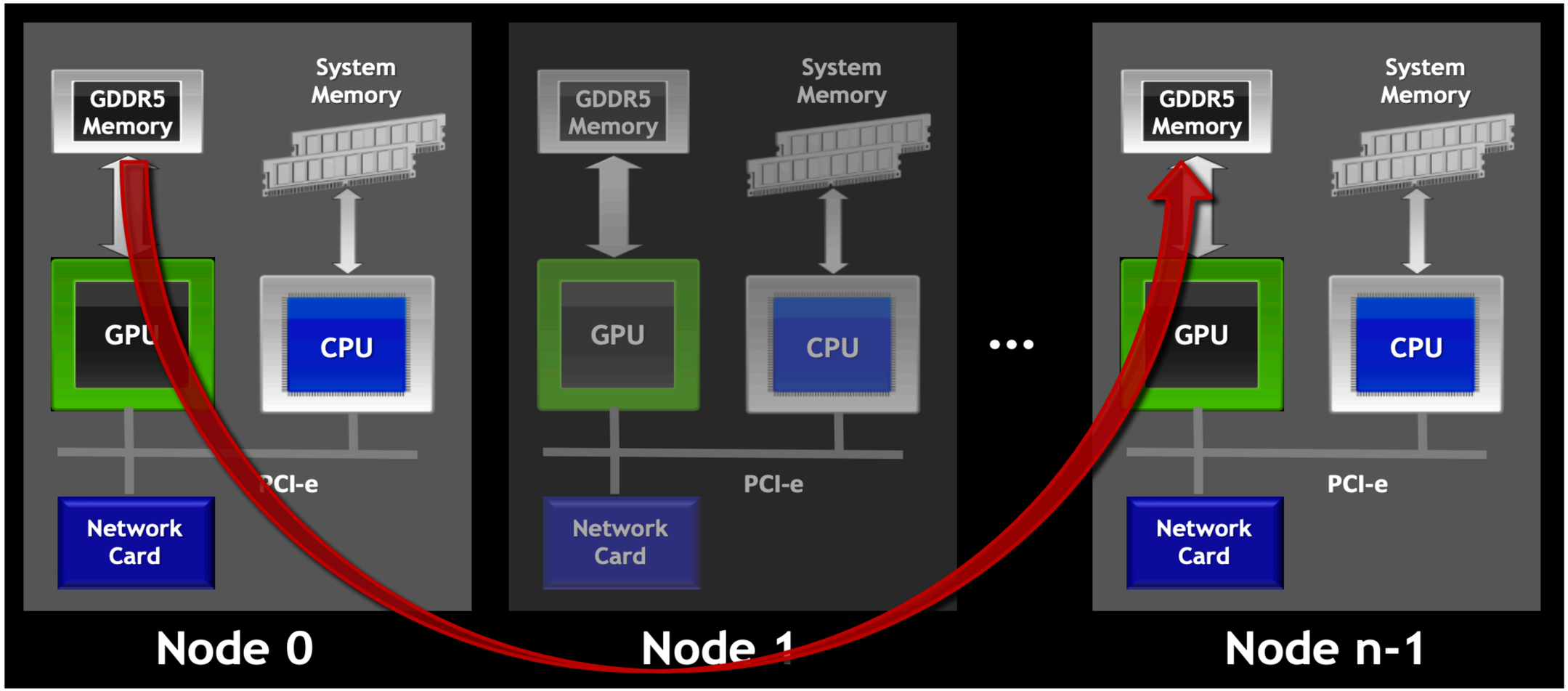


Multi-GPU architecture Command Line

```
bash-4.2$ nvidia-smi topo -m
        GPU0    GPU1    GPU2    GPU3    mlx5_0  mlx5_1  mlx5_2  mlx5_3  mlx5_4  mlx5_5  CPU Affinity    NUMA Affinity
GPU0     X      NV2     NV2     NV2     SYS     SYS     SYS     SYS     PIX     PIX     0-1,36-37       0-1
GPU1    NV2      X      NV2     NV2     SYS     SYS     SYS     SYS     PIX     PIX     0-1,36-37       0-1
GPU2    NV2     NV2      X      NV2     SYS     SYS     SYS     SYS     SYS     SYS     0-1,36-37       0-1
GPU3    NV2     NV2     NV2      X      SYS     SYS     SYS     SYS     SYS     SYS     0-1,36-37       0-1
mlx5_0  SYS     SYS     SYS     SYS      X      PIX     SYS     SYS     SYS     SYS
mlx5_1  SYS     SYS     SYS     SYS     PIX      X      SYS     SYS     SYS     SYS
mlx5_2  SYS     SYS     SYS     SYS     SYS     SYS      X      PIX     SYS     SYS
mlx5_3  SYS     SYS     SYS     SYS     SYS     SYS     PIX      X      SYS     SYS
mlx5_4  PIX     PIX     SYS     SYS     SYS     SYS     SYS     SYS      X      PIX
mlx5_5  PIX     PIX     SYS     SYS     SYS     SYS     SYS     SYS     PIX      X

Legend:

  X    = Self
  SYS  = Connection traversing PCIe as well as the SMP interconnect between NUMA nodes (e.g., QPI/UPI)
  NODE = Connection traversing PCIe as well as the interconnect between PCIe Host Bridges within a NUMA node
  PHB  = Connection traversing PCIe as well as a PCIe Host Bridge (typically the CPU)
  PXB  = Connection traversing multiple PCIe bridges (without traversing the PCIe Host Bridge)
  PIX  = Connection traversing at most a single PCIe bridge
  NV#  = Connection traversing a bonded set of # NVLinks
```

Introduction to CUDA-Aware MPI

Goal: Zero copy



https://on-demand.gputechconf.com/gtc/2013/presentations/S3047-Intro-CUDA-Aware-MPI-NVIDIA-GPUDirect.pdf

https://developer.nvidia.com/blog/introduction-cuda-aware-mpi/

Naive vs CUDA aware MPI support

MPI-Implementation with CUDA-Aware

- MVAPICH2 v(1.8/1.9b+)
- OpenMPI v(1.7+)
- CRAY MPI v(MPT 5.6.2+)
- IBM Spectrum MPI v(8.3+)
- SGI MPI v(1.08+)

Naive GPU MPI

```
//MPI rank 0
cudaMemcpy(s_buf_h,s_buf_d,size,cudaMemcpyDeviceToHost);
MPI_Send(s_buf_h,size,MPI_CHAR,1,100,MPI_COMM_WORLD);

//MPI rank 1
MPI_Recv(r_buf_h,size,MPI_CHAR,0,100,MPI_COMM_WORLD, &status);
cudaMemcpy(r_buf_d,r_buf_h,size,cudaMemcpyHostToDevice);
```

CUDA-Aware with GPU-Direct RDMA

```
//MPI rank 0
MPI_Send(s_buf_d,size,MPI_CHAR,1,100,MPI_COMM_WORLD);

//MPI rank n-1
MPI_Recv(r_buf_d,size,MPI_CHAR,0,100,MPI_COMM_WORLD, &status);
```

Naive vs CUDA aware MPI support OpenACC

Naive GPU MPI

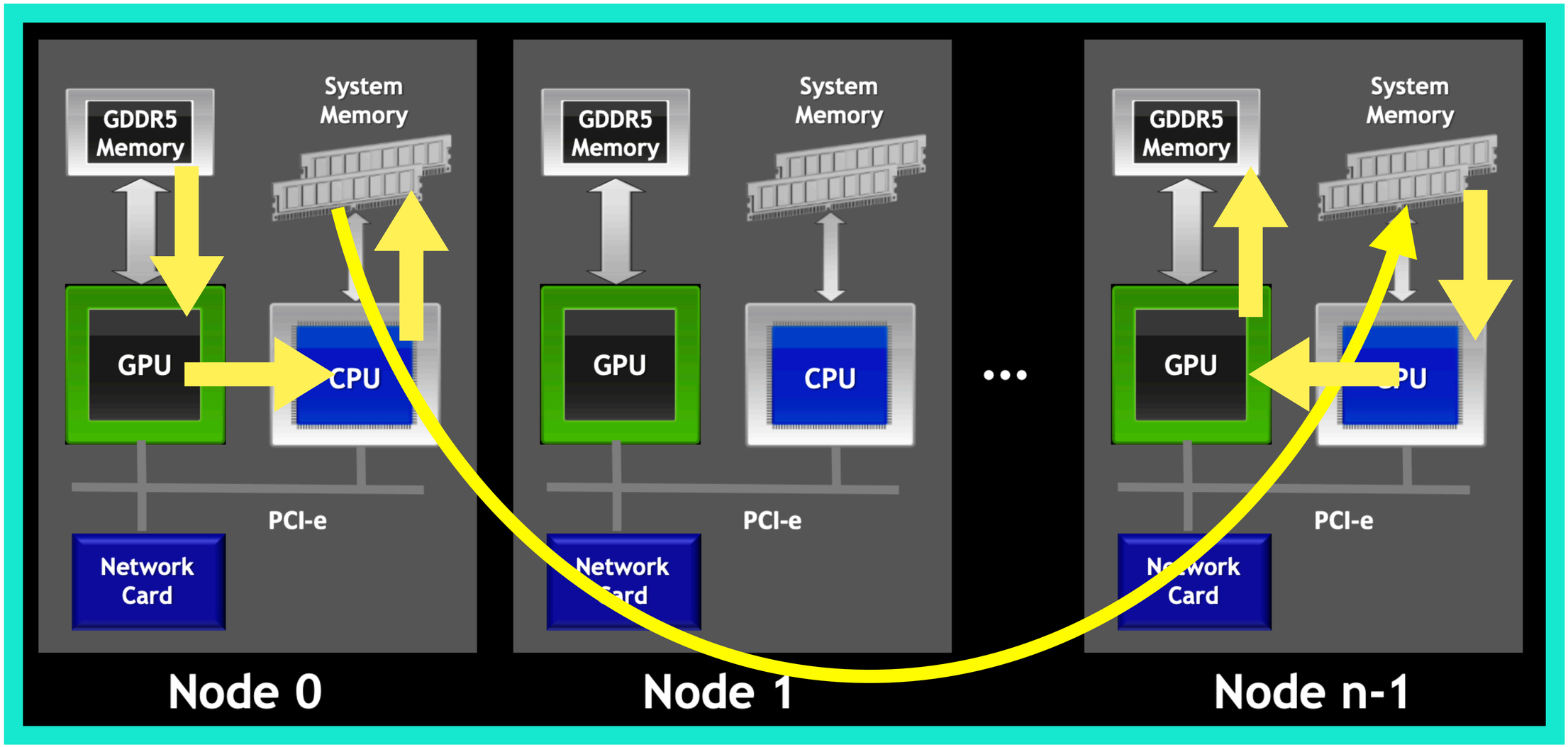
```
#pragma acc update host(s_buf[0:size] )
MPI_Send(s_buf,size,MPI_CHAR,1,tag,MPI_COMM_WORLD);
```

CUDA-Aware with GPU-Direct RDMA

```
#pragma acc host_data use_device(s_buf)
MPI_Send(s_buf,size,MPI_CHAR,1,tag,MPI_COMM_WORLD);
```

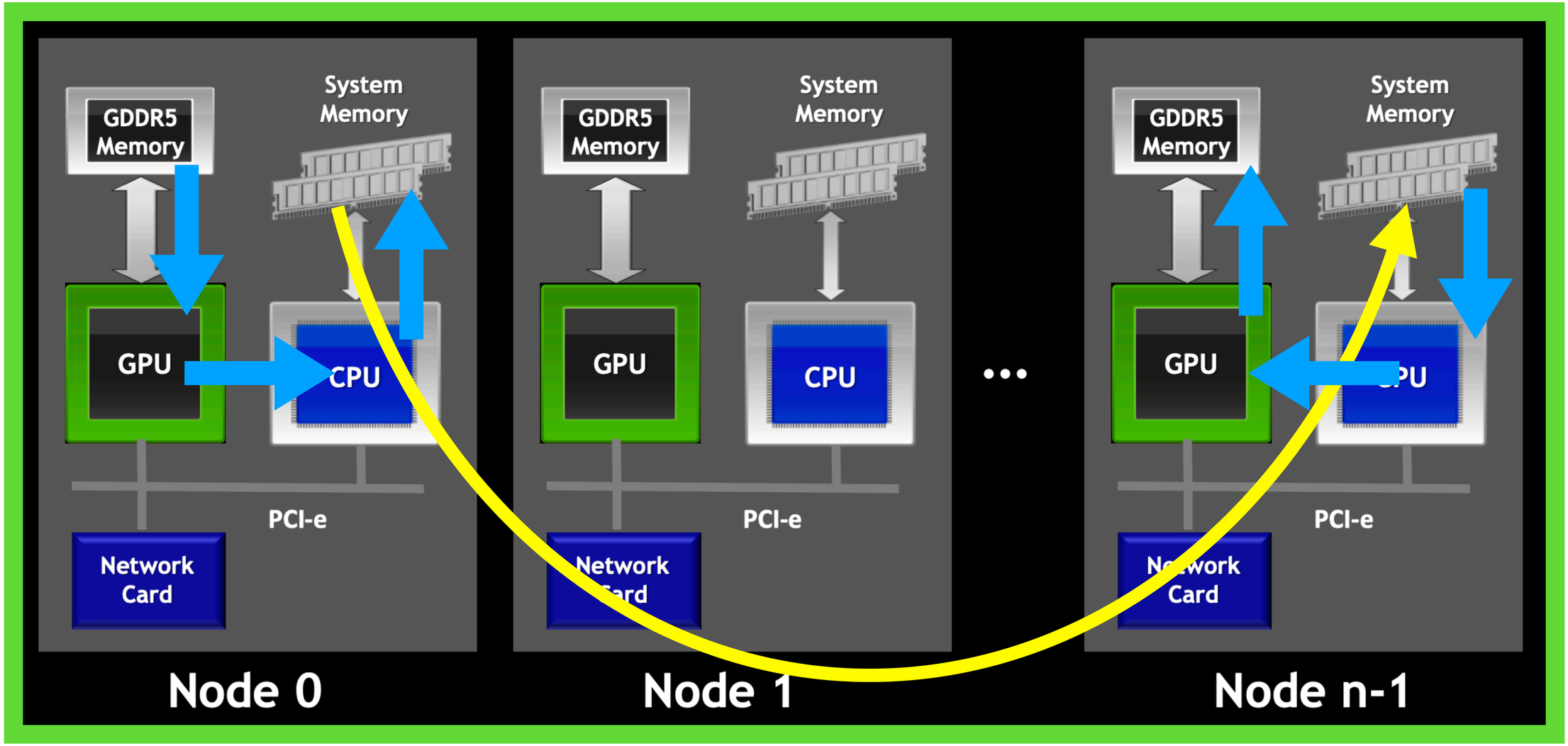
Naive GPU-GPU MPI

-Implements a double copy message passing



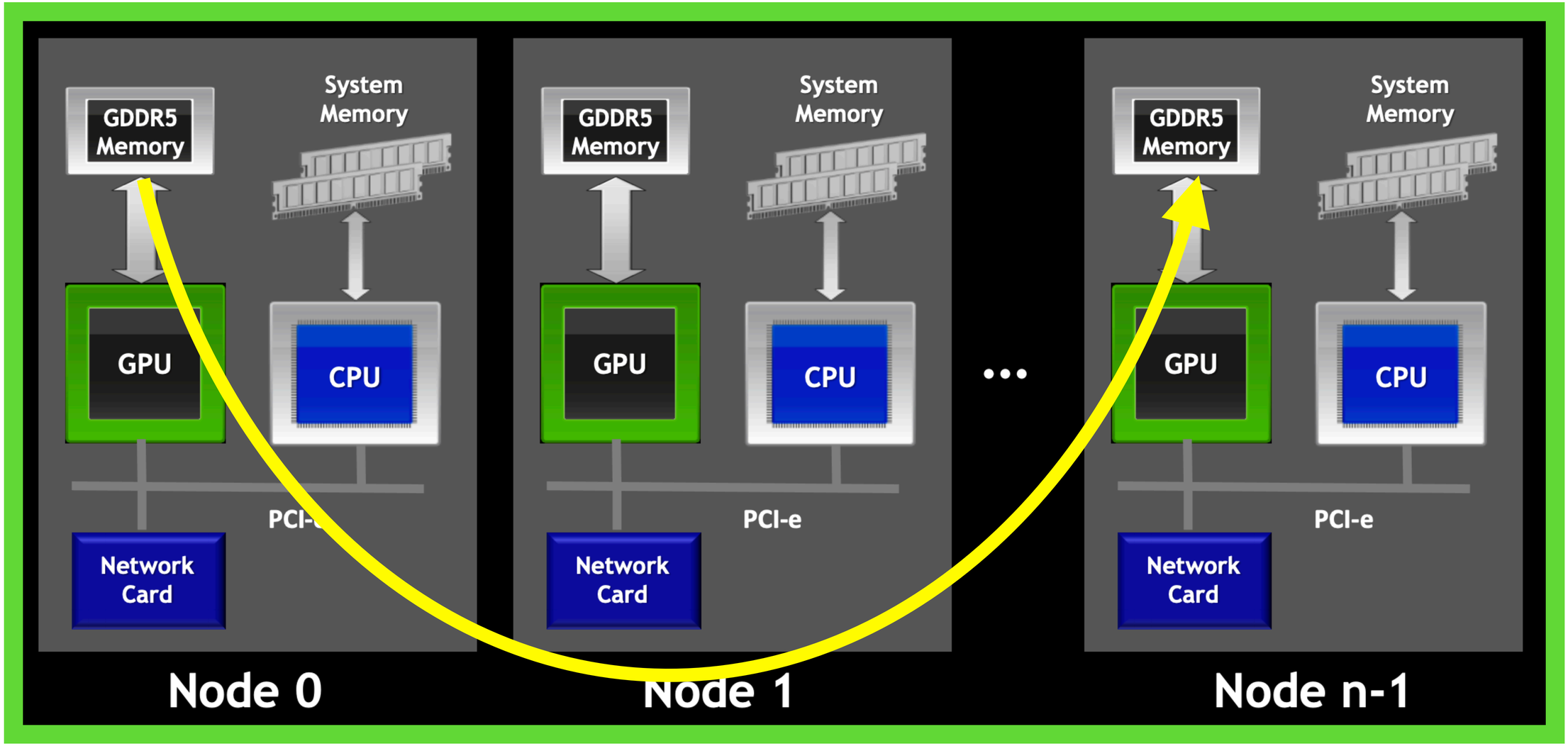
CUDA-aware MPI without GPU-Direct RDMA

- Implements single copy message passing



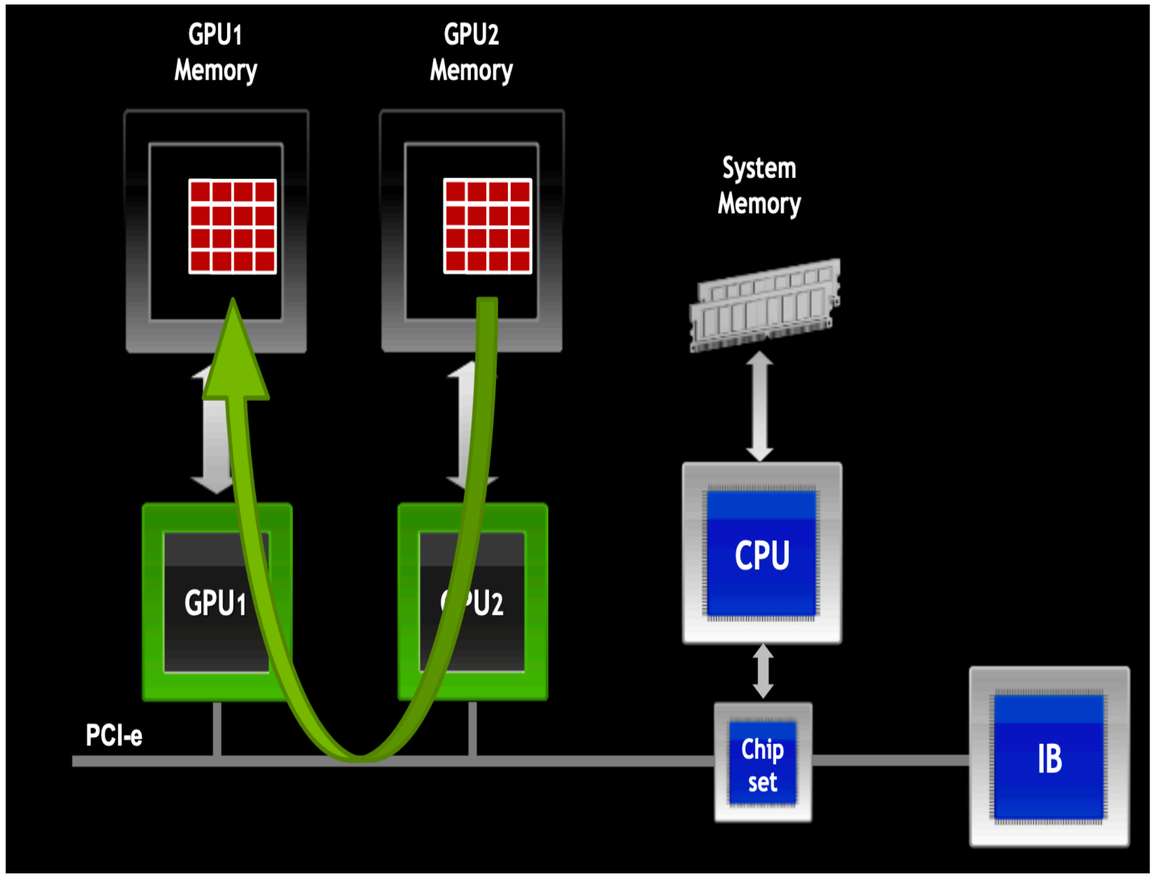
CUDA-aware MPI with GPU Direct RDMA

- Implements “zero copy” message passing!

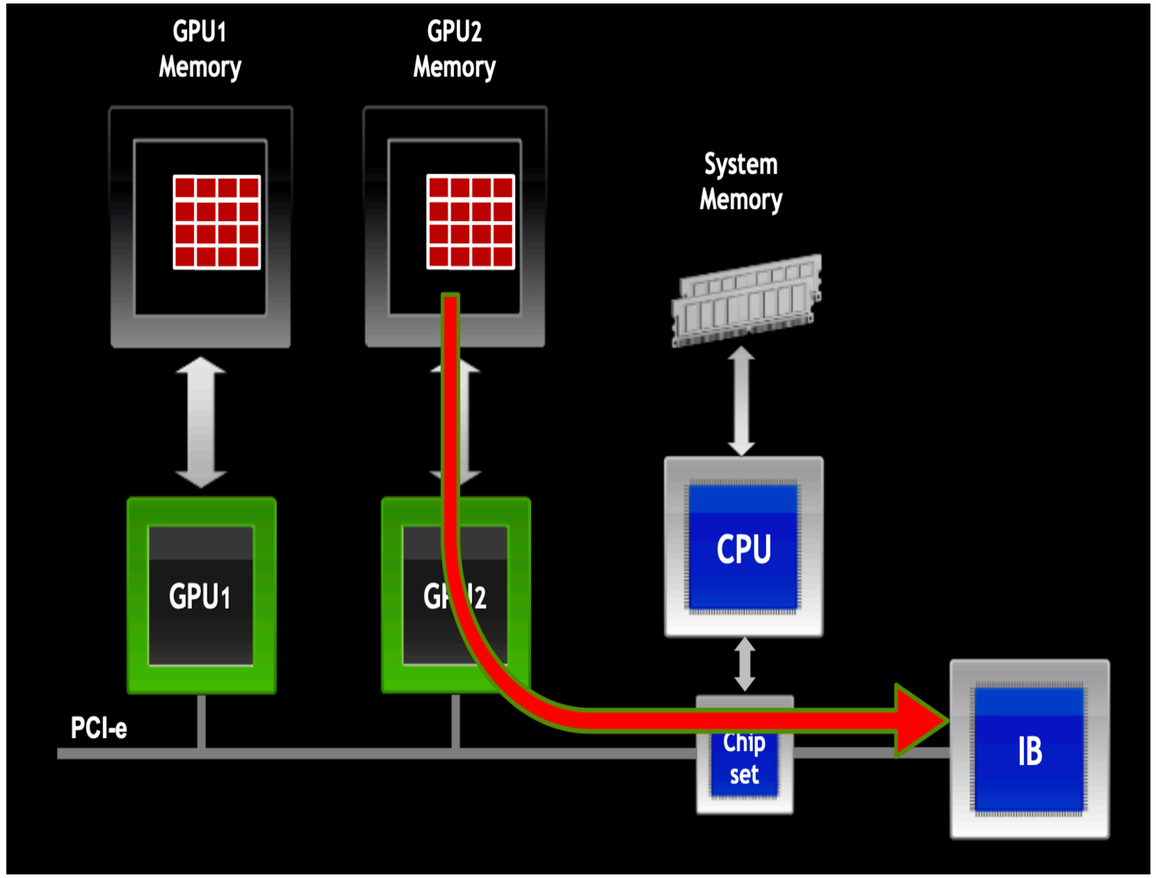


IntraNode vs InterNode Communication

IntraNode transfer - Peer to Peer



InterNode transfer - RDMA support

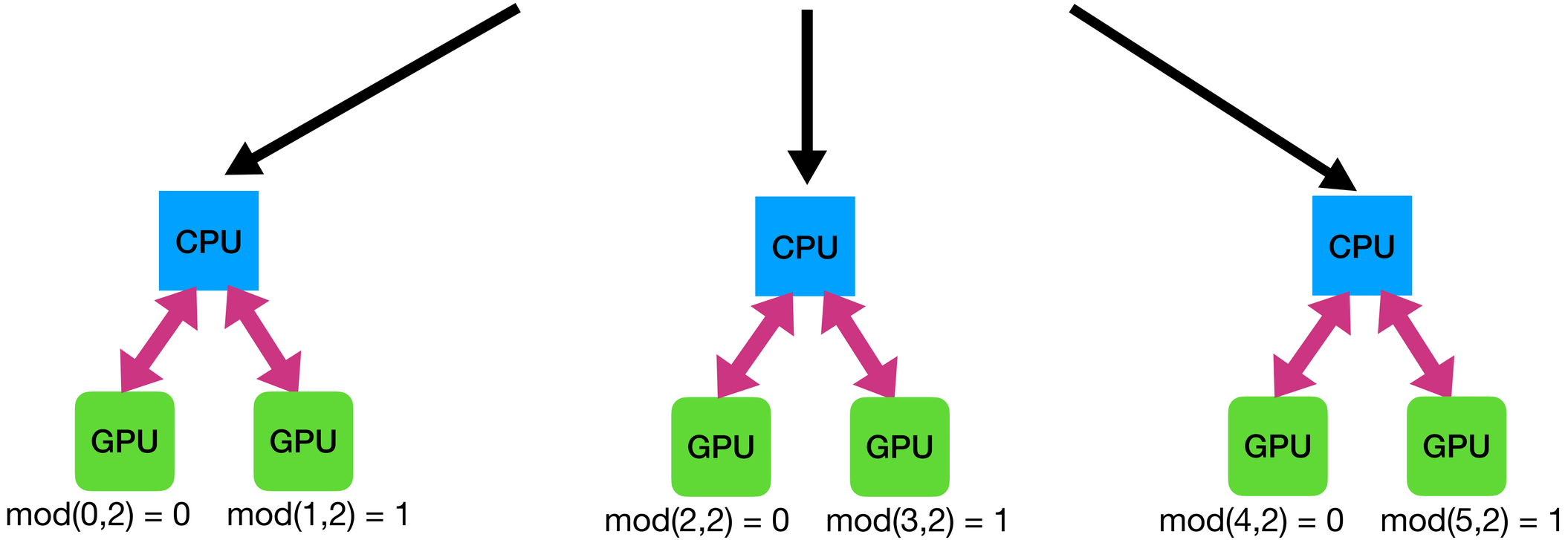


Launching MPI ranks and pinning task to GPU

acc_set_device_num(gpu_r , acc_device_nvidia)

mod(rank,D)=gpu_r
D = number of gpus/node

mpirun -n 6 -npernode 2 <args> ./mpi_test



npernode = 2
Ranks = 0, 1
D = 2

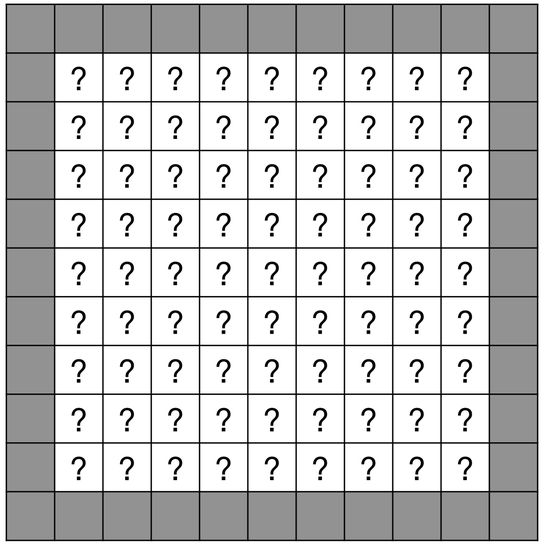
npernode = 2
Ranks = 2, 3
D = 2

npernode = 2
Ranks = 4, 5
D = 2

Code Explanation

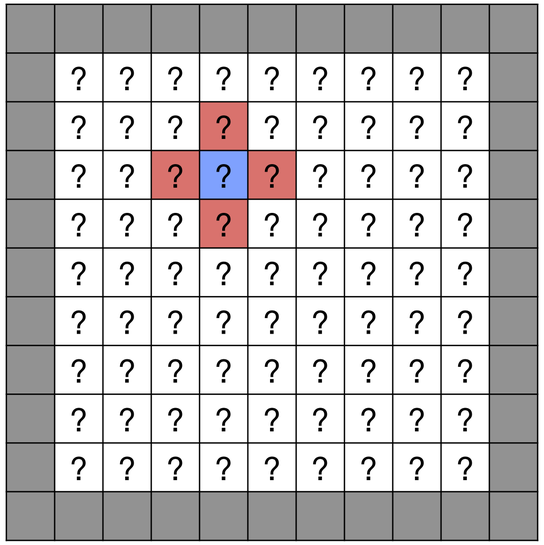
MPI-OpenACC: Problem Conception

Given a 2D grid of points with a border of known values, find the steady-state of the interior points



MPI-OpenACC: Problem Conception

Given a 2D grid of points with a border of known values, find the steady-state of the interior points



The value of the interior points will be found by iteratively taking the average of the 4 points surrounding it

$$M_{new}[i\ ,\ j] = 1/4(M[i+1\ ,\ j]+M[i\ ,\ j+1]+M[i\ ,\ j-1]+M[i-1\ ,\ j])$$

This forms a "stencil" that is applied to all points in the interior

Iteration will cease once the difference between M_new[i , j] and M[i , j] is below a certain threshold

Aside: Finding the steady-state is an application of the Laplace equation. Rearranging the equation above and creating a flattened version of the interior points puts the problem in a form that fits for the Jacobi method. So the code is an iterative Laplace Jacobi solver

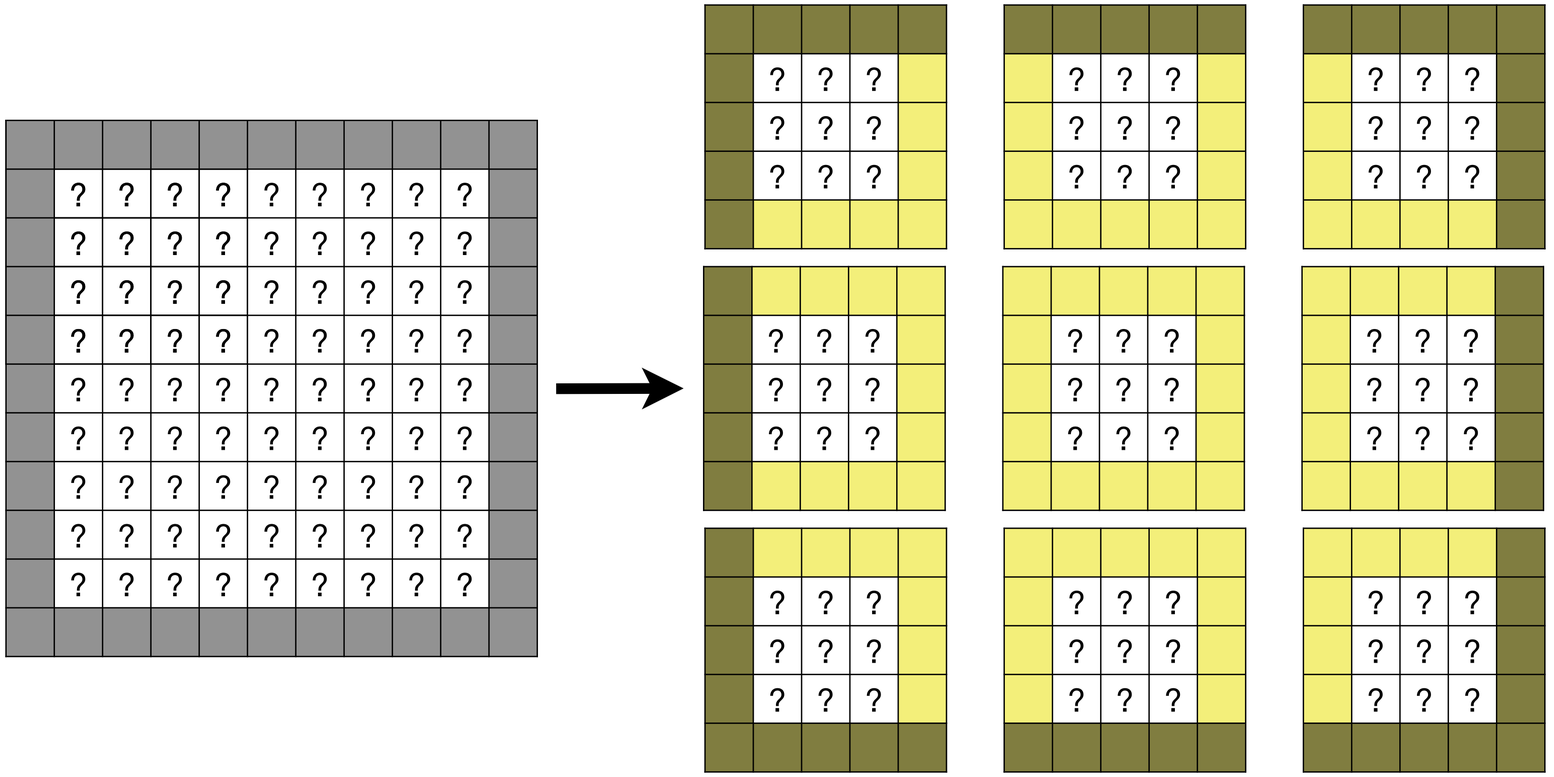
MPI-OpenACC: Laplace-Jacobi Base Solver Code

```
// Allocate the second version of the M matrix used for the computation
M_new = (float*)malloc(ny*nx*sizeof(float));

do {
        maxdiff = 0.0f;
        itr++;
        // Update M_new with M
        for(int i=1; i<ny-1; i++){
                for(int j=1; j<nx-1; j++){
                        M_new[i*nx+j] = 0.25f *(M[(i-1)*nx+j]+M[i*nx+j+1]+ \
                                                M[(i+1)*nx+j]+M[i*nx+j-1]);
                }
        }

        // Check for convergence while copying values into M
        for(int i=1; i<ny-1; i++){
                for(int j=1; j<nx-1; j++){
                        maxdiff = MAX(fabs(M_new[i*nx+j] - M[i*nx+j]), maxdiff);
                        M[i*nx+j] = M_new[i*nx+j];
                }
        }
} while(maxdiff > JACOBI_TOLERANCE);

// Free malloc'd memory
free(M_new);
```

MPI-OpenACC: Laplace-Jacobi Distributed Solver Code

A large matrix can take a long time to update, but the code is embarrassingly parallel. So, we can divide the matrix among separate ranks with a "ghost area" around each to enable exchanging of information.

Consider the same matrix with 9x9 interior points split among 9 processors the problem then looks like:



Each MPI rank will:

1. Update its interior points
2. Perform a "halo exchange" with its neighbors by sharing part of its updated values using MPI_Irecv and MPI_Isend.
3. Check for convergence of its own values.
4. Share its convergence error with all processes to determine if the algorithm is finished.

Profiling Kernels with nvprof CLI

```
mpirun  -n 16 nvprof --print-gpu-trace --profile-api-trace none --kernels "::LaplaceJacobi_MPIACC:2"  --metrics
"achieved_occupancy,gld_transactions,gst_transactions,flop_count_sp"  ./mpi_acc_stencil.exe 64 4
```

- Profiles 2nd invocation of all kernels launched with “LaplaceJacobi_MPIACC” in name
- Gathers specified metrics (Use “nvprof -- query-metrics” for full list of options)

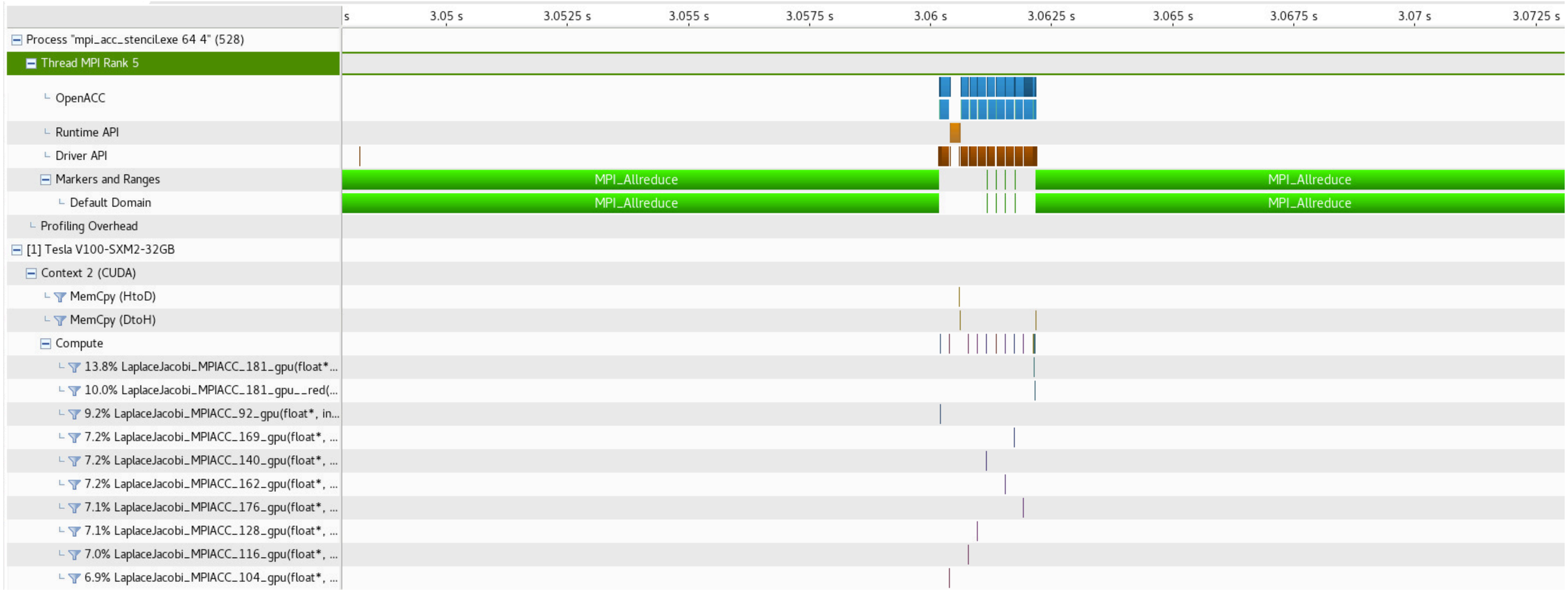
```
==74878== Profiling result:
         Device   Context    Stream                Kernel  achieved_occupancy  gld_transactions  gst_transactions  flop_count_sp

Tesla V100-SXM2         2        29  LaplaceJacobi_MPIACC            0.062388               587               160           4096
```

- Monitor the achieved occupancy as the data size increases
  - Ex: for a Matrix size 16,384 divided over 16 Ranks

```
==76531== Profiling result:
         Device   Context    Stream                Kernel  achieved_occupancy  gld_transactions  gst_transactions  flop_count_sp

Tesla V100-SXM2         2        29  LaplaceJacobi_MPIACC            0.936929           9630915           2621440       67108864
```

Profiling MPI with nvprof GUI

```
mpirun -n 16 nvprof --annotate-mpi openmpi -o Jacobi.%q{OMPI_COMM_WORLD_RANK}.nvprof ./mpi_acc_stencil.exe 64 4
```



MPI-OpenACC: Constructs and Clauses Used

Construct	Clause	C code	Description
Parallel		#pragma acc parallel [clauses]	Launches a number of gangs in parallel each with a number of workers, each with vector or SIMD operations.
Loop	collapse	#pragma acc loop [clauses]	Applies to the immediately following loop or nested loops and describes the type of parallelism to execute these loops on the GPU.
	reduction	#pragma acc loop reduction(+:temp)	The reduction clause specifies a reduction operator and one or more vars (e.g. +, -, max, min).
	present	#pragma acc loop present(A[start:count])	Compiler hint that the variable is already in device memory. If not a runtime error occurs.
Enter Data	copyin	#pragma acc enter data copyin(A[start:count])	Allocates the data in list on the GPU and copies the data from the host to the GPU when entering the region.
	create	#pragma acc enter data create(A[start:count])	Allocates the data in list on the GPU, but does not copy data between the host and device.
Exit Data	copyout	#pragma acc enter data copyin(A[start:count])	Allocates the data in list on the GPU and copies the data from the GPU to the host when exiting the region.
	delete	#pragma acc enter data delete(A[start:count])	Free the GPU memory used by the variable(s).
Host data	use_device	#pragma acc host_data use_device(A)	Directs the compiler to use the device address of any entry in list.

Most text copied from https://www.openacc.org/sites/default/files/inline-files/OpenACC_2.5_ref_guide_1.pdf or https://www.nvidia.com/docs/IO/116711/OpenACC-API.pdf

Exercise: Distributed LaplaceJacobi Solver

Look for left-aligned comments starting with "//" for tips on what to do in the code

Steps:

- In main.cpp

- Add openacc.h header
- After acc_init call the function provided to map GPUs to MPI ranks
- Fill in the call to LaplaceJacobi_MPIACC with the correct arguments

- In stencil.cc

- Add openacc.h header file
- Add a set of unstructured data pragmas to copyin and create all the variables needed
- Add a set of unstructured data pragmas to copyout and delete variables
- Add pragmas to parallelize the update and convergence loops
- Add pragmas to ensure the MPI communication can see the send/receive buffers.
- Parallelize any other loops inside the LaplaceJacobi_MPIACC function

To build use **./build.sh**

To submit use **sbatch submit.sh**

Exercise Check: Distributed LaplaceJacobi Solver

What questions did you have?

Exercise Check: Distributed LaplaceJacobi Solver

Some things to think about:

- What happens when we don't map GPUs and MPI ranks?

- Do we need to do a halo exchange every iteration? What might change if we don't?

- The time for CPU code with the input `mpirun -n 16 ./mpi_acc_stencil.exe 64 4` is about 0.103006 seconds. If we run the same inputs on OpenACC code, is this a fair comparison?

- If we were to run this code on only 1 V100 GPU, how big of a matrix could we process? (recall global memory size).

Exercise Check: Distributed LaplaceJacobi Solver

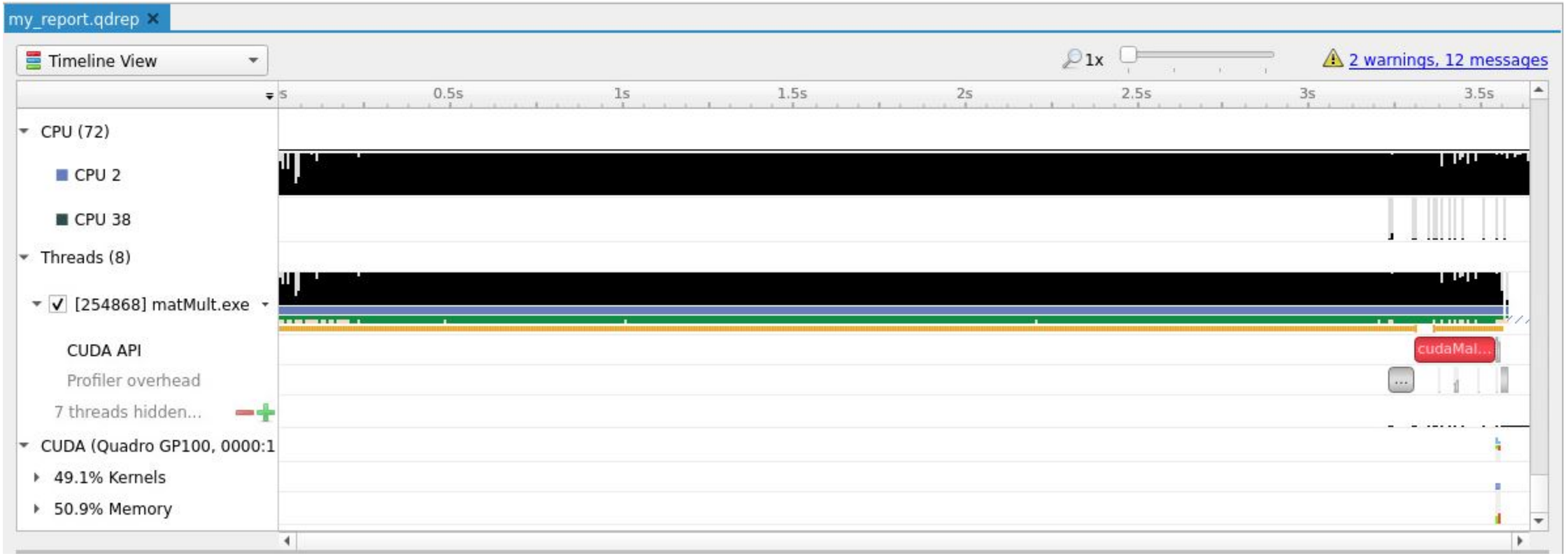
Some things to think about:

- What happens when we don't map GPUs and MPI ranks?
  - All code will execute on one GPU
- Do we need to do a halo exchange every iteration? What might change if we don't?
  - Consider a very large matrix, the most interior points will take a long time to converge regardless of what occurs at the edges. We might be able to save execution time without message passing
- The time for CPU code with the input `mpirun -n 16 ./mpi_acc_stencil.exe 64 4` is about 0.103006 seconds. If we run the same inputs on OpenACC code, is this a fair comparison?
  - No. Not only do we need optimized CPU code, the timers used in main.cpp aren't how we would get GPU speedup. We would look at output from NV_ACC_TIME=1 or profiling and average by the number of calls.
- If we were to run this code on only 1 V100 GPU, how big of a matrix could we process? (recall global memory size).
  - Theoretically about 46k by 46k matrices. This is half of what we expect because of M_new.

Nsight Profiler

Last session, we looked at the nvprof which is a command line profiler that prints out information about your program execution.

Today, we will be looking at Nsight which is also a profiler but is more user friendly as it displays information about your program execution on an interactive GUI as seen below.



Using Nsight Profiling on Casper

Because we will be using the Nsight GUI, this exercise will require more steps than was required to use the nvprof command line profiler.

Step 1: Install TigerVnc viewer on your host computer. Information about installation can be found at: https://tigervnc.org/

Step 2: Login to Casper and run the commands below. After running these commands, your screen should look like the image below.

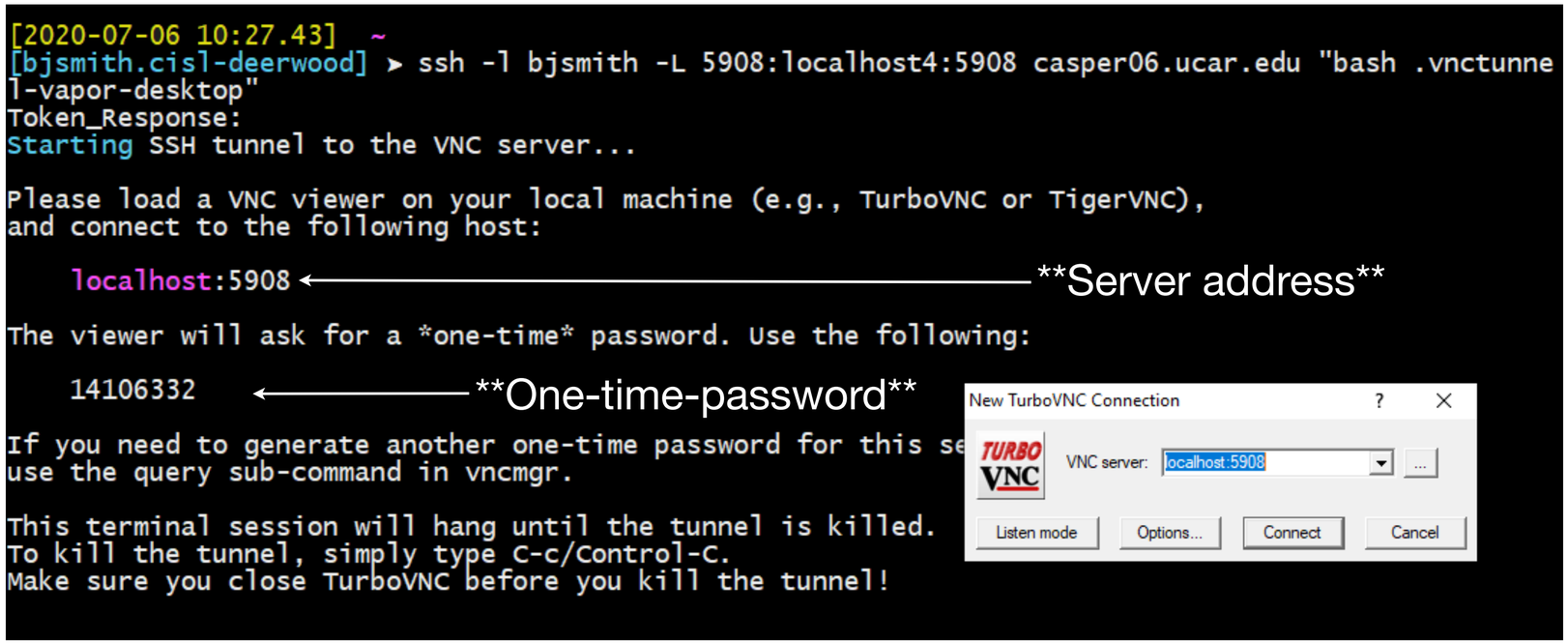
- module load nvhpc.
- vncmgr list
- vncmgr create [session name] --account [Project]

```
To activate the VNC session on your client machine (external network):

  1) Create an SSH tunnel to stream desktop data to your local computer:

       ssh -l bjsmith -L 5908:localhost4:5908 casper06.ucar.edu "bash .vnctunnel-vapor-desktop"

  2) Follow instructions given in local terminal. The local terminal session
     will hang after creating the tunnel. This behavior is normal.

If you disconnect from the VNC server and wish to reconnect,
you will need to restart the tunnel to generate a new password.
This VNC server will timeout and shut down automatically after 7:59:59.

To end the Slurm job and shut down the VNC server:

  1) Use scancel to end the VNC server job:

       scancel 5530022

Note: you can avoid the tunneling step if using the NCAR internal network
      or the NCAR VPN (https://www2.cisl.ucar.edu/user-support/vpn-access)
```

Using Nsight Profiler on Casper

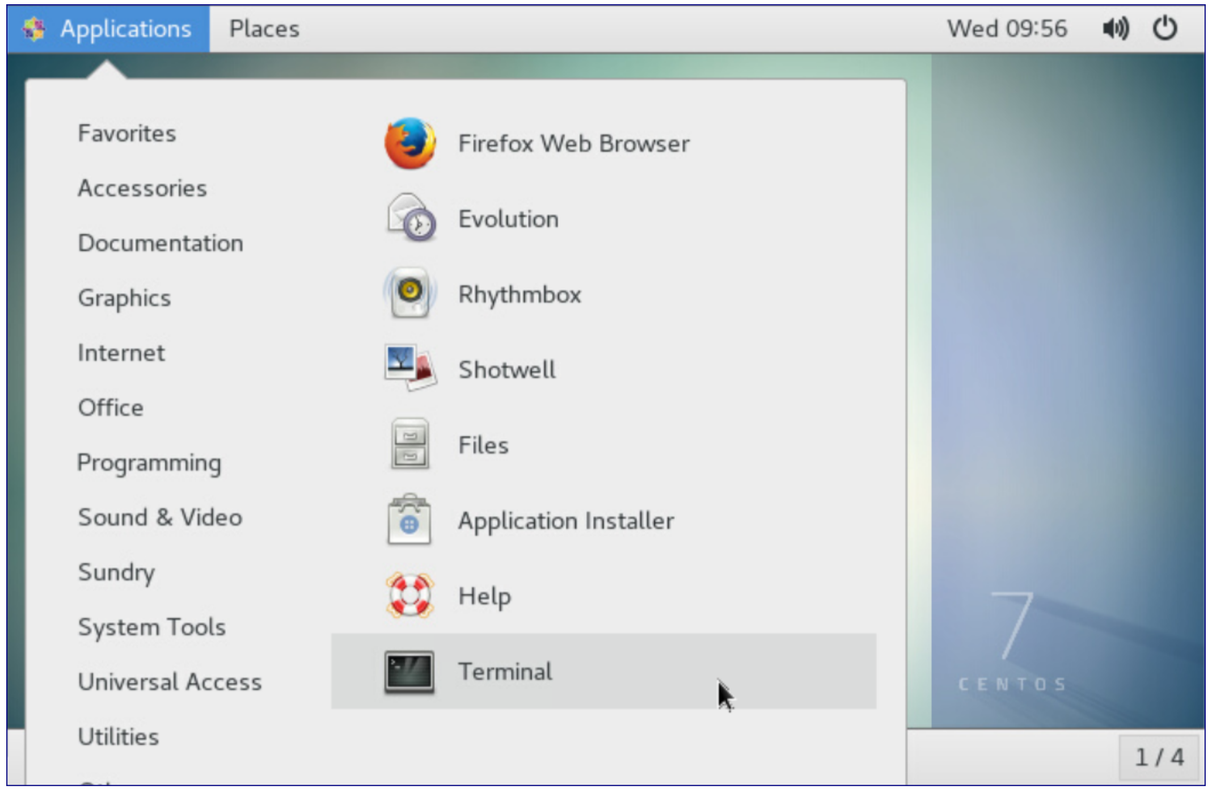
Step 3: On your host computer, run the ssh command gotten from the output of the previous step.
Note: Your host computer must be connected to the NCAR Internal network or the NCAR VPN.

Step 4: Open Tigervnc viewer and enter the Server address and the "one-time" password given to you after running the ssh command.



Using Nsight Profiler on Casper

Step 5: Once you are logged in, open the terminal window.



Using Nsight on Casper

Step 6: Once you are on the terminal, load the cuda module using "module load cuda". Without loading this, the nsys commands will not be loaded into our environment.

Step 7: Run the build script and submit the job.

Now we profile!

Step 8: Run the following command to profile your code:

"nsys profile -o profileData ./exec_file.exe"

Step 9: Finally, run the command "nsight-sys profileData.qdrep" to view your profile on the GUI