GPU Programming Workshop

Session 3: Advanced topics in OpenACC+MPI

Special Technical Projects
&
Consulting Services Group

UCAR

April 20, 2021

Recap

- Naive Matrix multiplication
- Matrix multiplication with shared memory
- CUDA coding practice

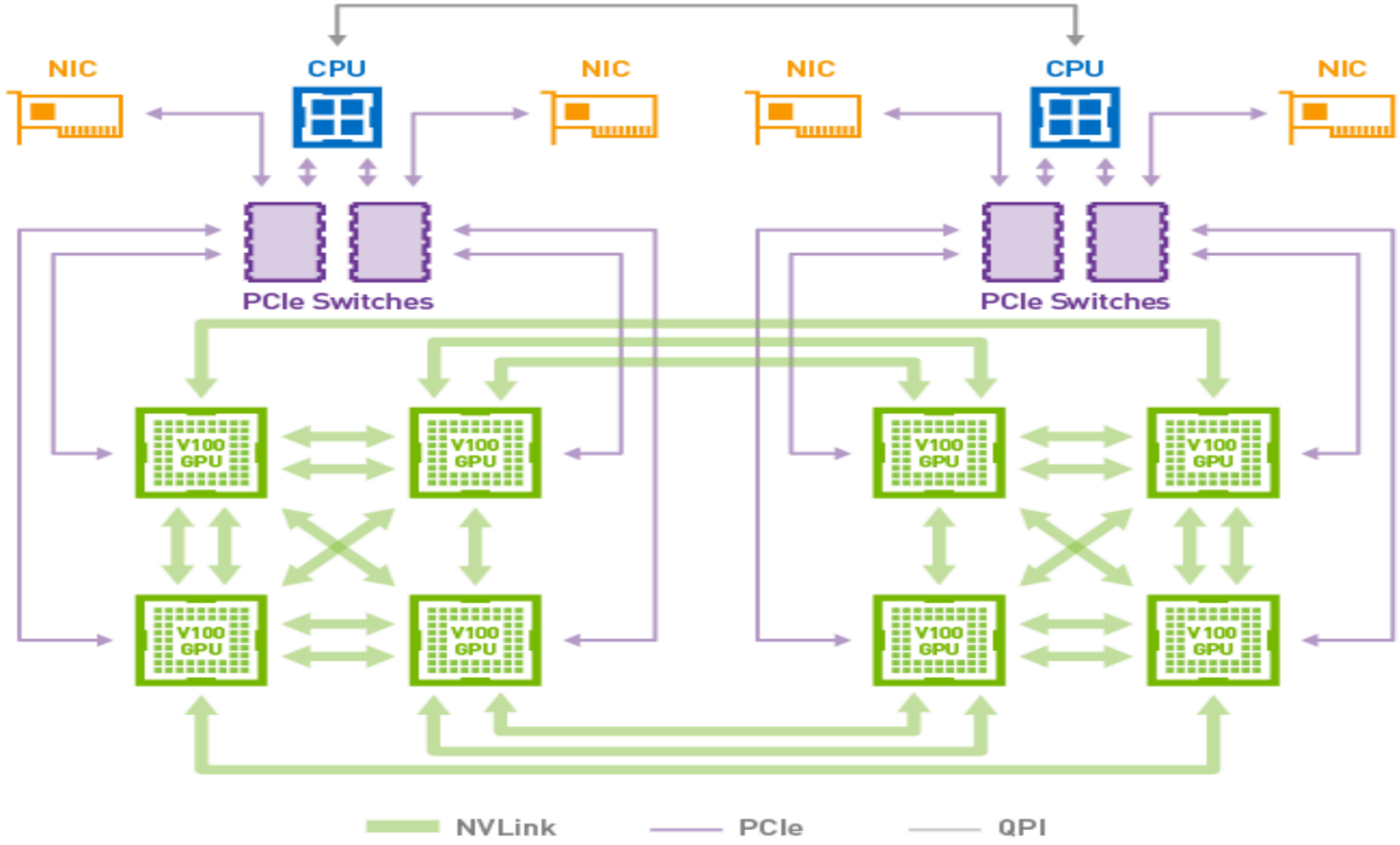
Before going further, clone the workshop code, slides, and reference materials using:
git clone https://github.com/NCAR/GPU_workshop.git

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

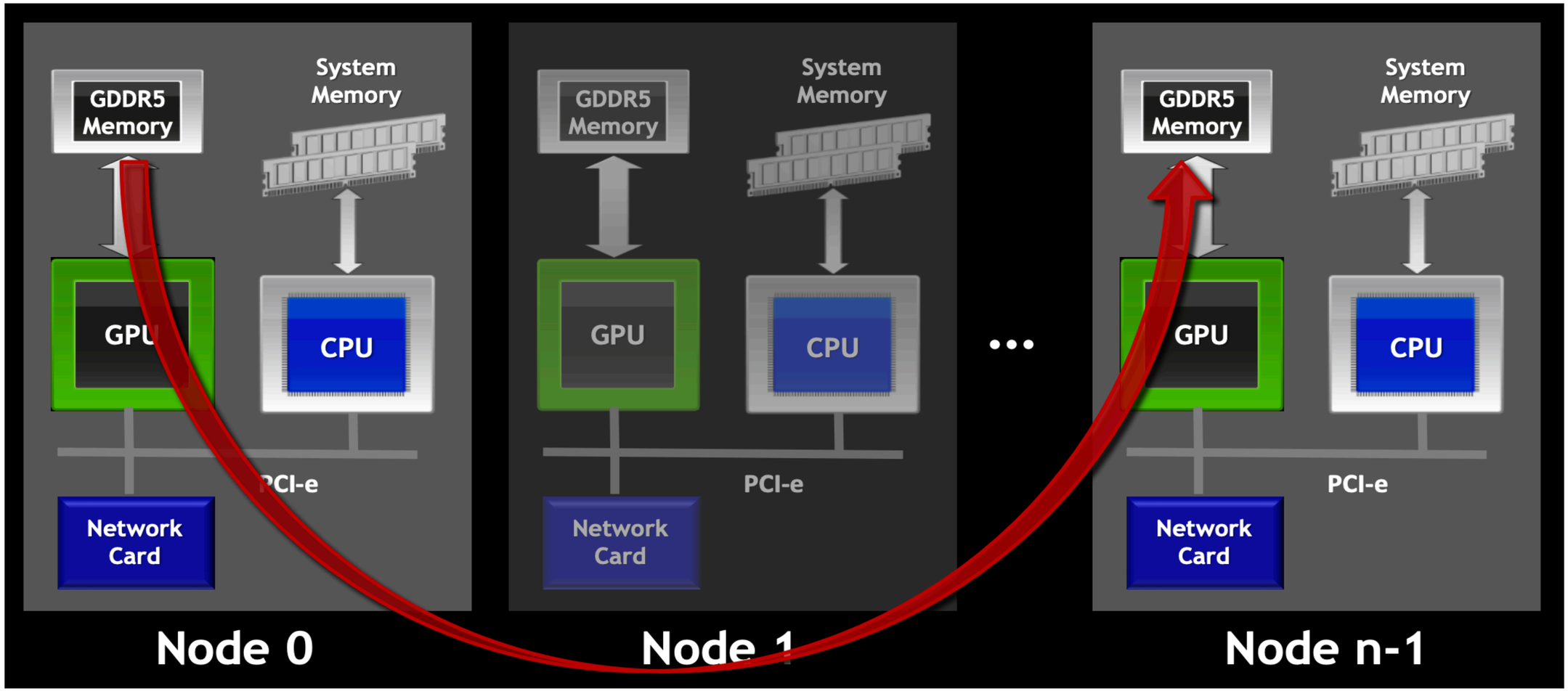
```
bash-4.2$ nvidia-smi topo -m
        GPU0    GPU1    GPU2    GPU3    mlx5_0  mlx5_1  mlx5_2  mlx5_3  mlx5_4  mlx5_5  CPU Affinity    NUMA Affinity
GPU0     X      NV2     NV2     NV2     SYS     SYS     SYS     SYS     PIX     PIX     0-1,36-37       0-1
GPU1    NV2      X      NV2     NV2     SYS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

CUDA-Aware MPI-Implementations

- 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

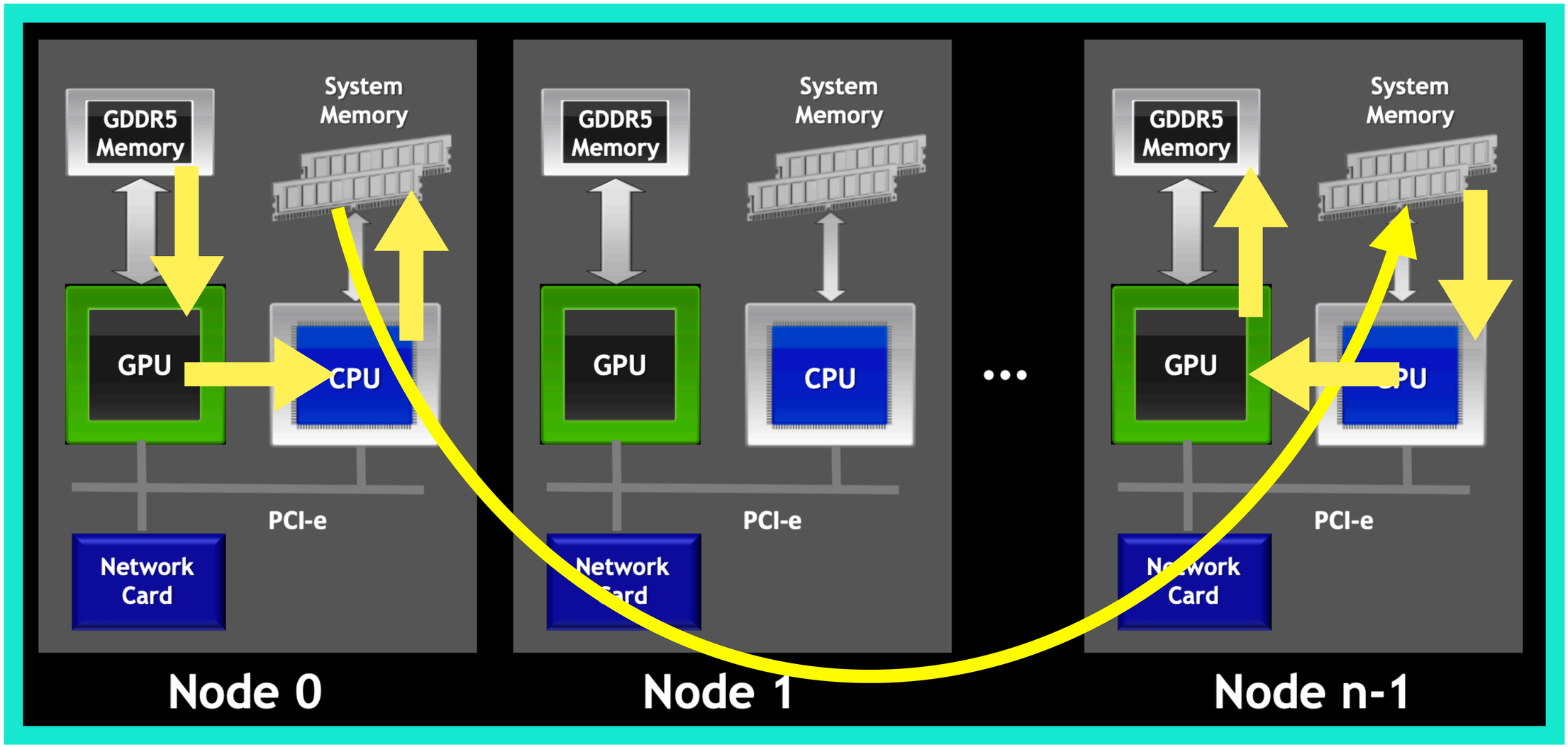
```
#pragma acc update host(s_buf[0:size] )
MPI_Send(s_buf,size,MPI_CHAR,1,tag,MPI_COMM_WORLD);
```

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);
```

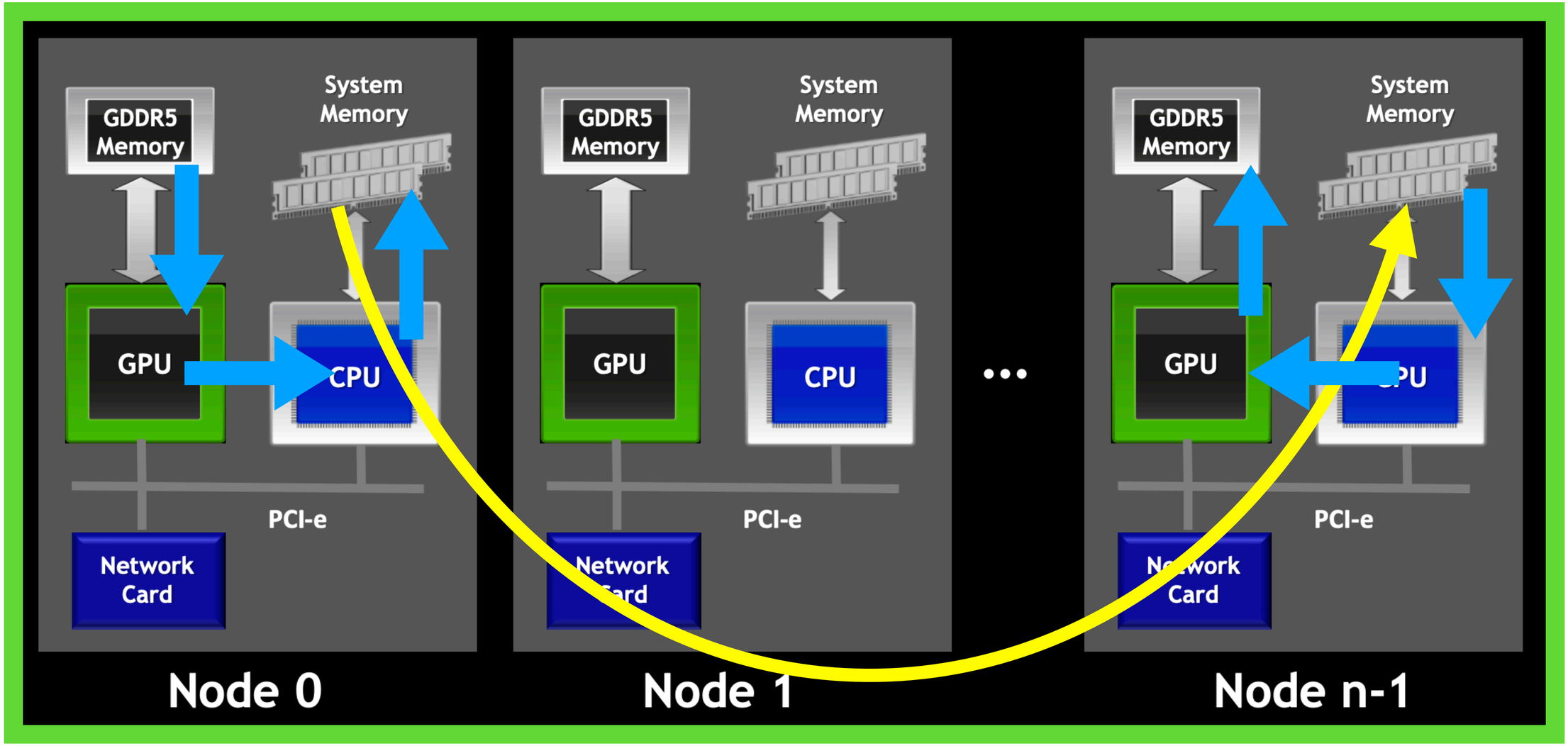
Naive GPU-GPU MPI

-Implements a double copy message passing



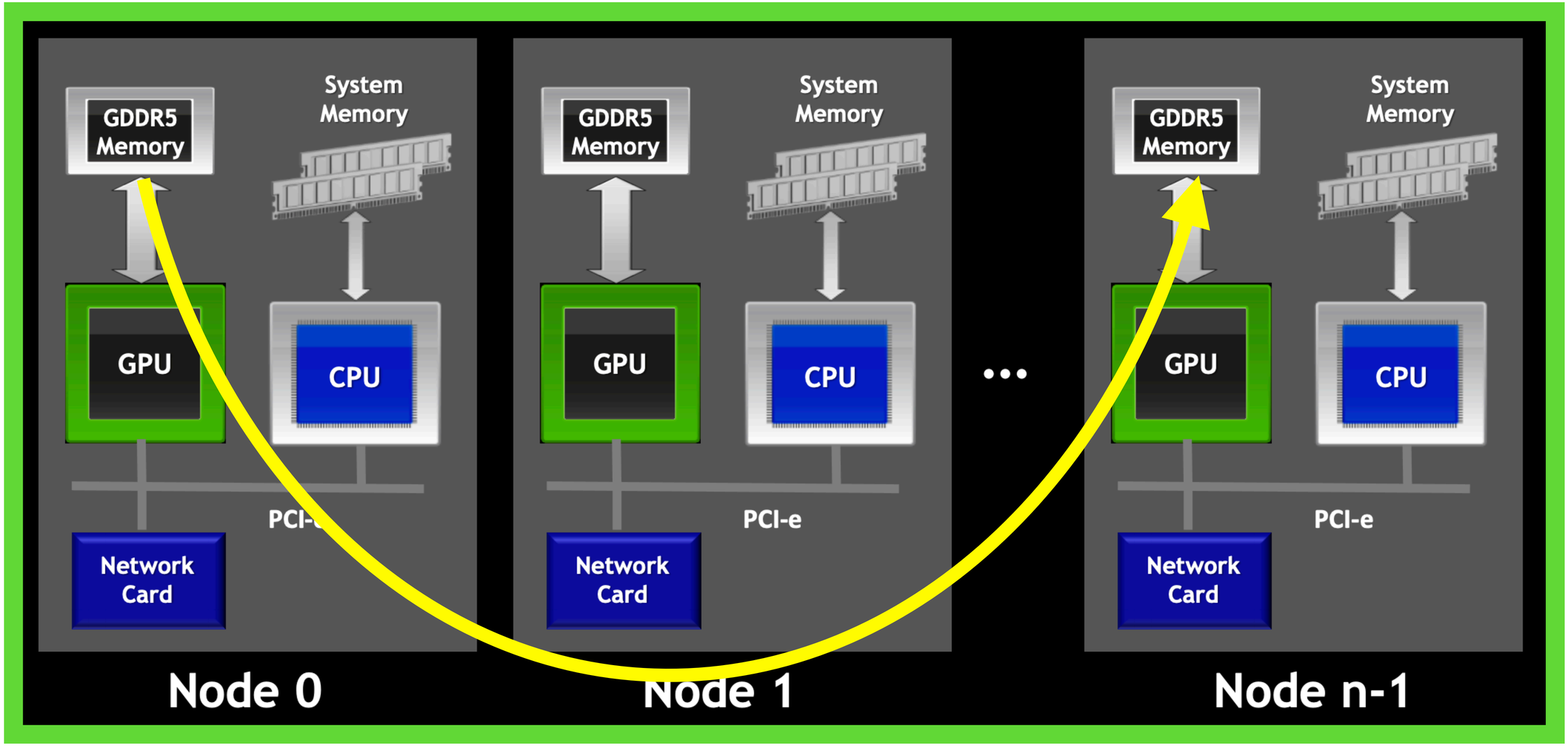
CUDA-aware MPI without GPU-Direct RDMA

- Implements single copy message passing



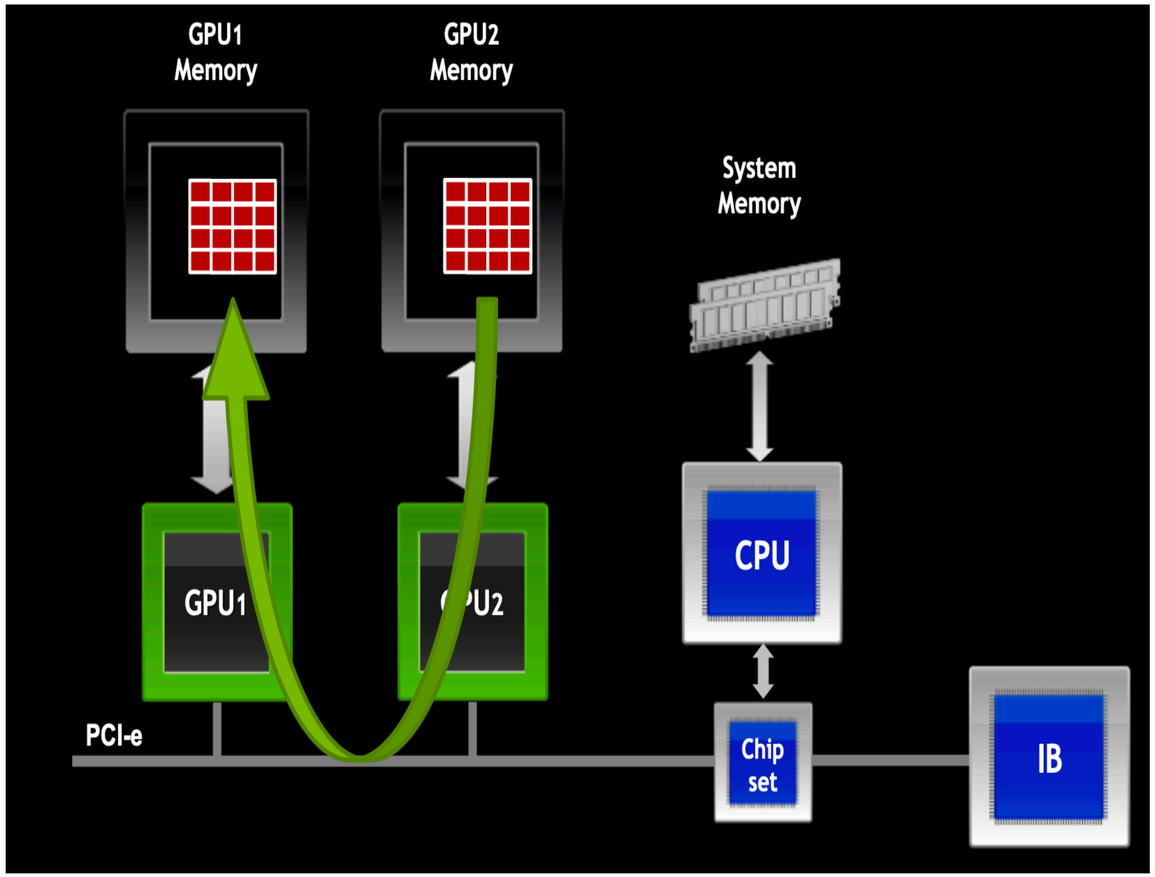
CUDA-aware MPI with GPU Direct RDMA

- Implements “zero copy” message passing!

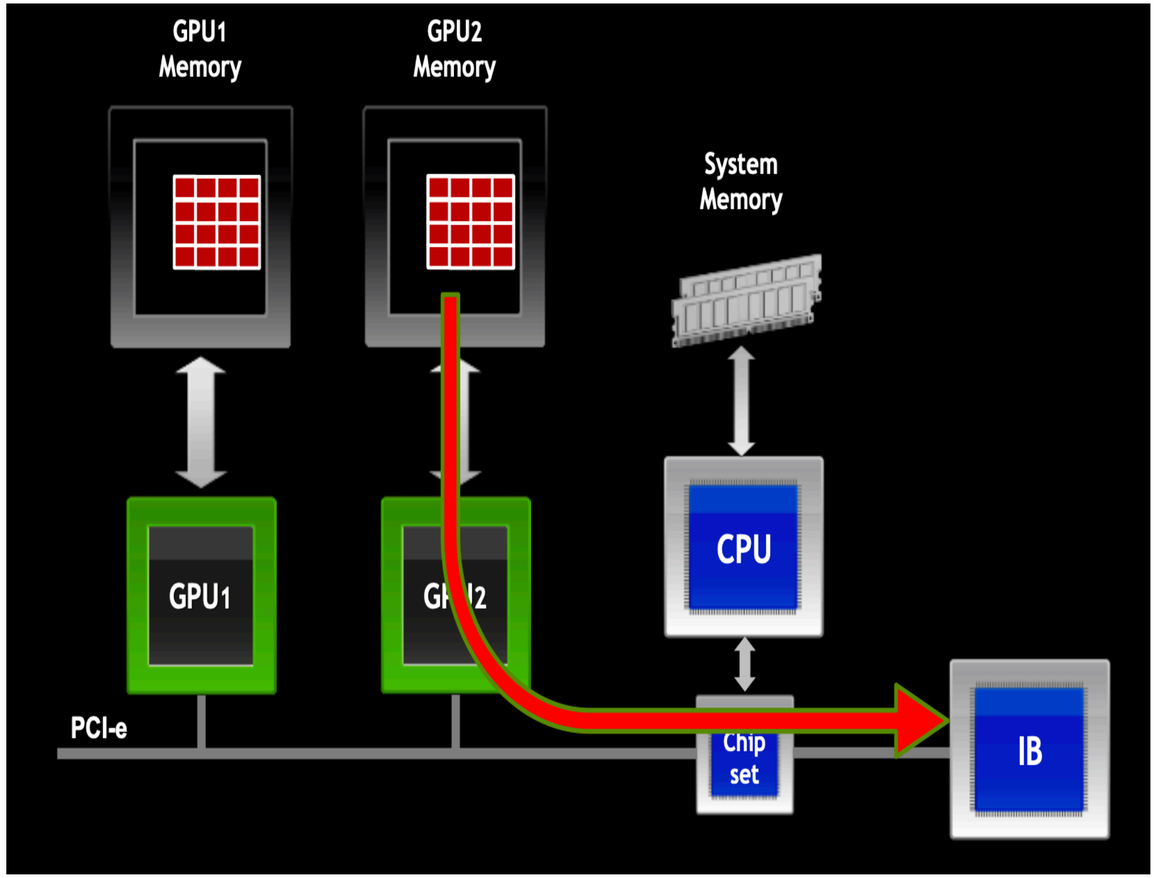


IntraNode vs InterNode Communication

IntraNode transfer - Peer to Peer



InterNode transfer - RDMA support

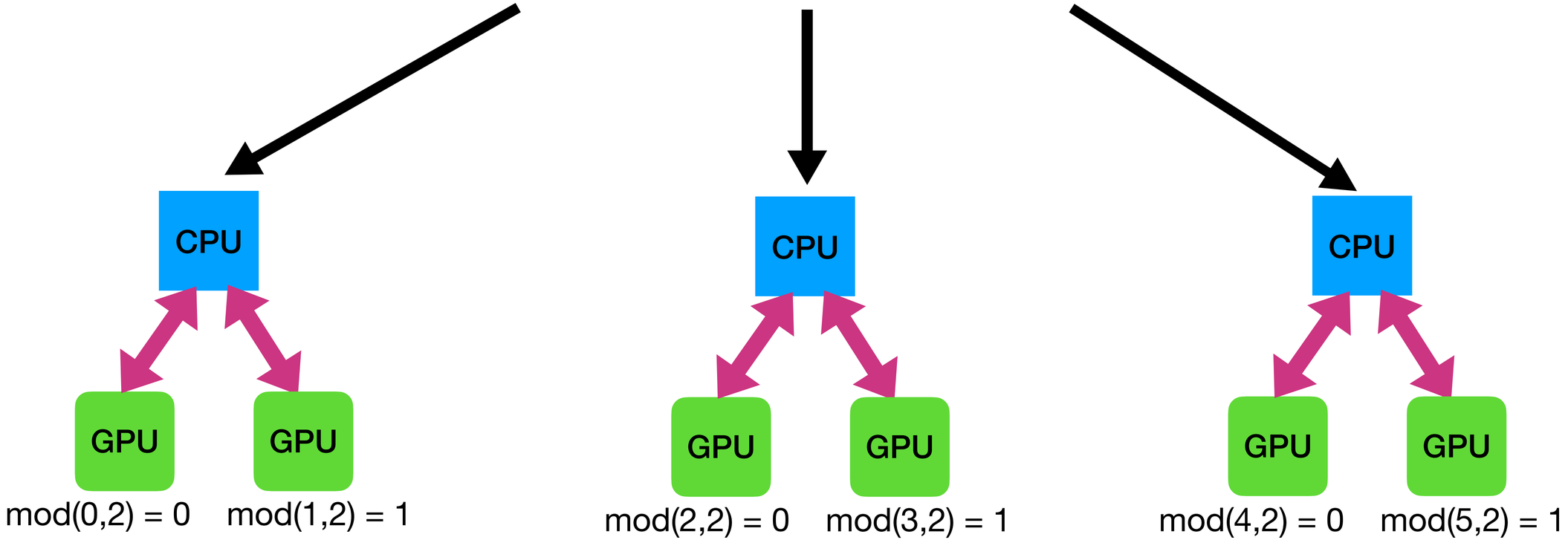


Launching MPI ranks and pinning task to GPU

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



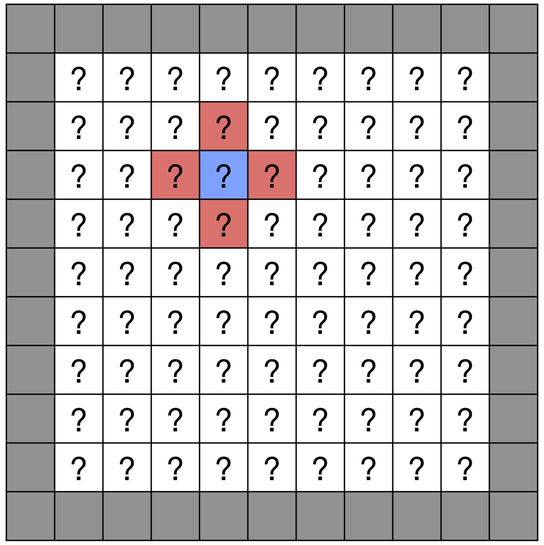
npernode = 2
Ranks = 0, 1
D = 2

npernode = 2
Ranks = 2, 3
D = 2

npernode = 2
Ranks = 4, 5
D = 2

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

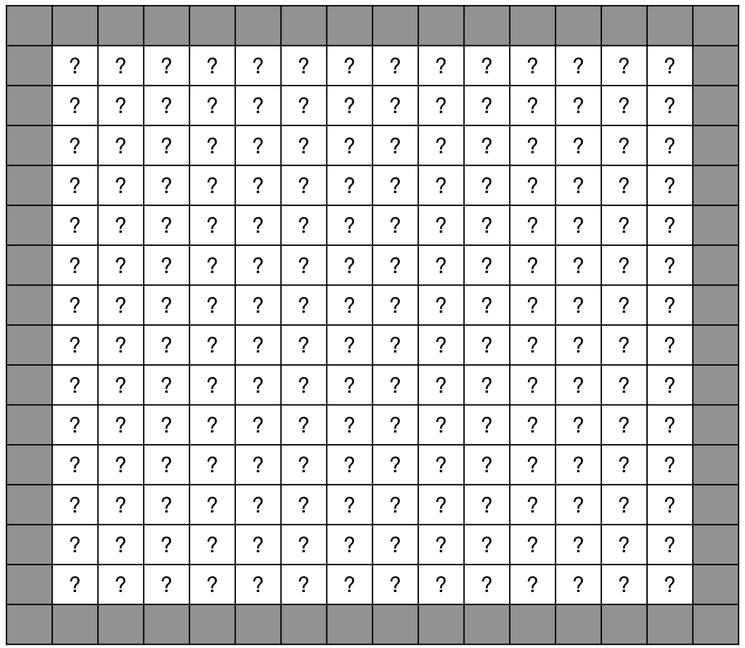
$$M_{new}[i\ ,\ j] = 1/4(M[i+1\ ,\ j]+M[i\ ,\ j+1]+M[i\ ,\ j-1]+M[i-1\ ,\ j])$$

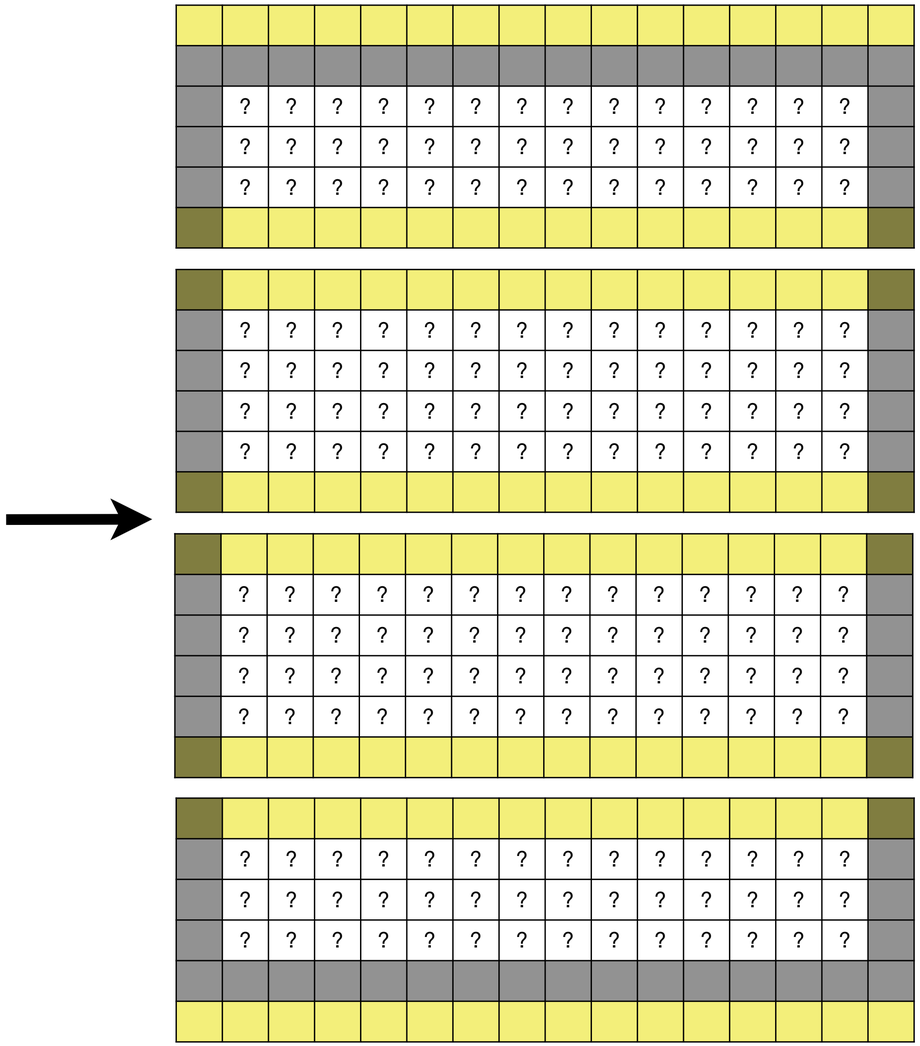
This forms a "stencil" that is applied to all points in the interior

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

MPI-OpenACC: Laplace-Jacobi Distributed Solver Code

Matrix with 16x16 points split among 4 processors:





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.

Unstructured Data Directives

Unstructured Data Directives:

enter data - defines the start of an unstructured data lifetime

clauses : copyin(list), create(list)

exit data - defines the end of an unstructured data lifetime

clauses: copyout(list), delete(list)

```
#pragma acc enter data copyin(a[0:N],b[0:N]) create(c[0:N])

  #pragma acc parallel loop
  for(int i = 0; i < N; i++){
    c[i] = a[i] + b[i];
  }

#pragma acc exit data copyout(c[0:N])
```

Structured v. Unstructured Directives

Structured:

- Regions are defined by **acc data**
  - Use curly braces **{ }** in C++
  - Use **acc end data** in FORTRAN
- Implicit variable lifetime marked
- Most up-to-date data present on GPU with fresh copies

Unstructured:

- Regions defined by **acc enter data** and **acc exit data**
  - Not defined with braces/**end data**
- Data moved onto GPU remains until it is manually cleared
- Data which is not modified on the CPU
  may be reused across parallel loops without additional data movement clauses

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 exit data copyout(A[start:count])	Copies the data from the GPU to the host when exiting the region and then deallocates the data in list on the GPU.
	delete	#pragma acc exit 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.

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 128
```

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

Device	Context	Stream	Kernel	achieved_occupancy	gld_transactions	gst_transactions	flop_count_sp
Tesla V100-SXM2	1	17	LaplaceJacobi_MPIACC	0.061920	2289	636	16128

- 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
```

Exercise: Distributed LaplaceJacobi Solver

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

- 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 **./casper_build.sh**

To submit use **qsub casper_submit.sh**

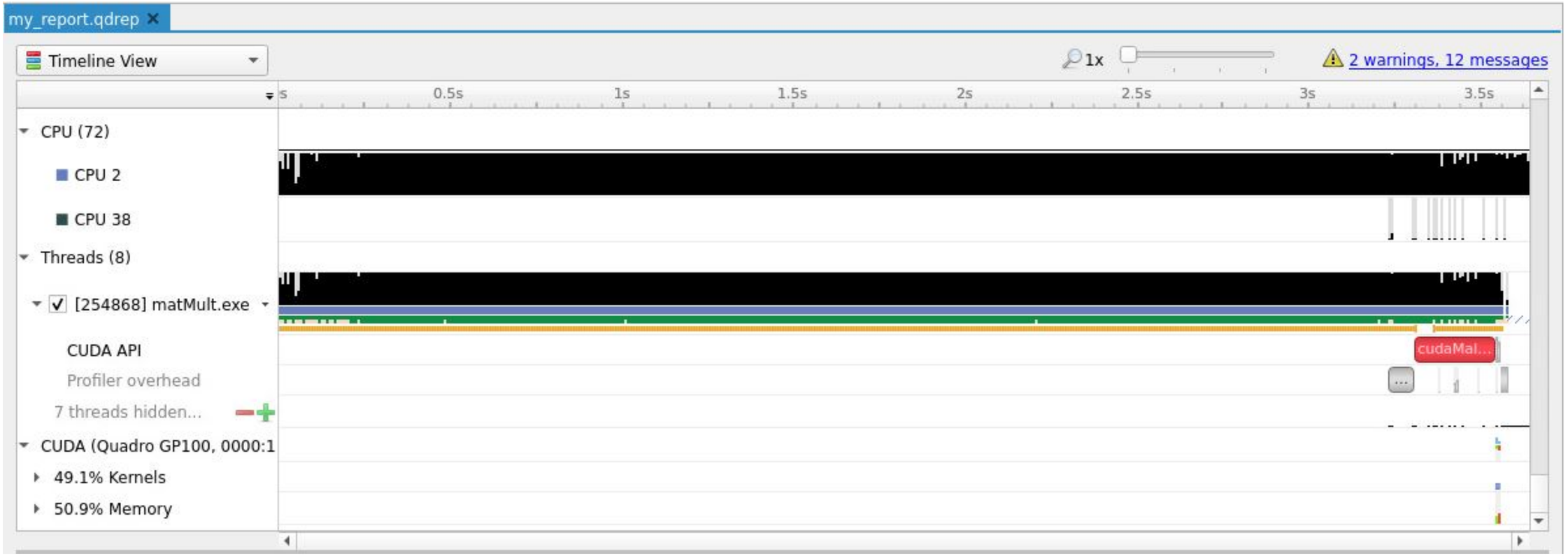
Breakout Room

MPI Stencil with OpenACC

Nsight Profiler

In the breakout room, we looked at NV_ACC_TIME and nvprof CLI as ways to profile your code.

Now, 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 Profiler on Casper (Using FastX via web browser)

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: Establish an ssh tunnel from your desktop to fastx.ucar.edu

ssh -L 3300:fastx.ucar.edu:3300 username@fastx.ucar.edu

Step 2: Open your browser with URL **https://localhost:3300/**

Step 3: You may find warning as in **Fig: Step3**

Step 4: No worries! simply click **Advanced** circled in red

Step 5: You should see additional messages as in **Fig: Step 4** simply click **Proceed to localhost (unsafe)** circled in red

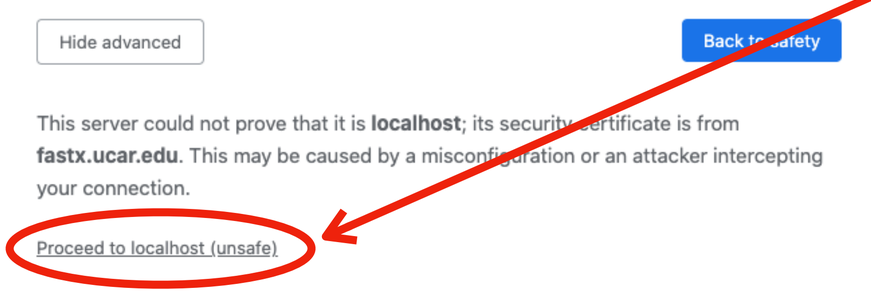


Fig: Step 4

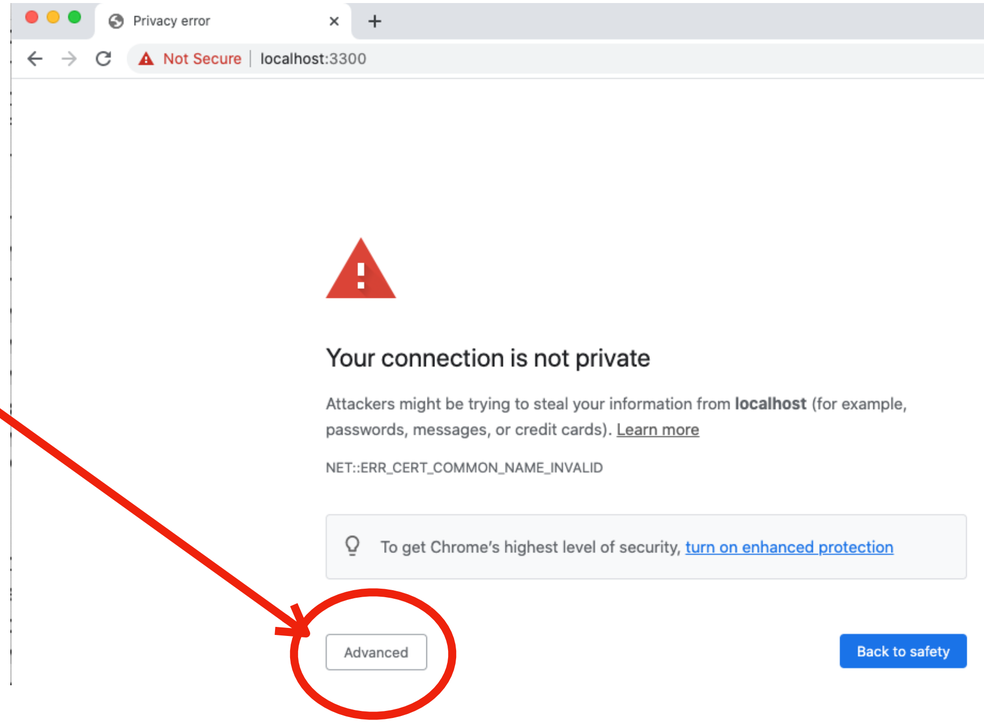


Fig: Step 3

Using Nsight Profiler on Casper (Using FastX via web browser) contd

Step 6: You should see screen prompting your username as in **Fig: Step 6**, **insert your username** and click Login

Step 7: You should see screen prompting Token_Response as in **Fig: Step 7**, respond to Token_Response as usual and click Submit

Step 8: On successful authentication, you should see screen like **Fig: Step 8**, click +

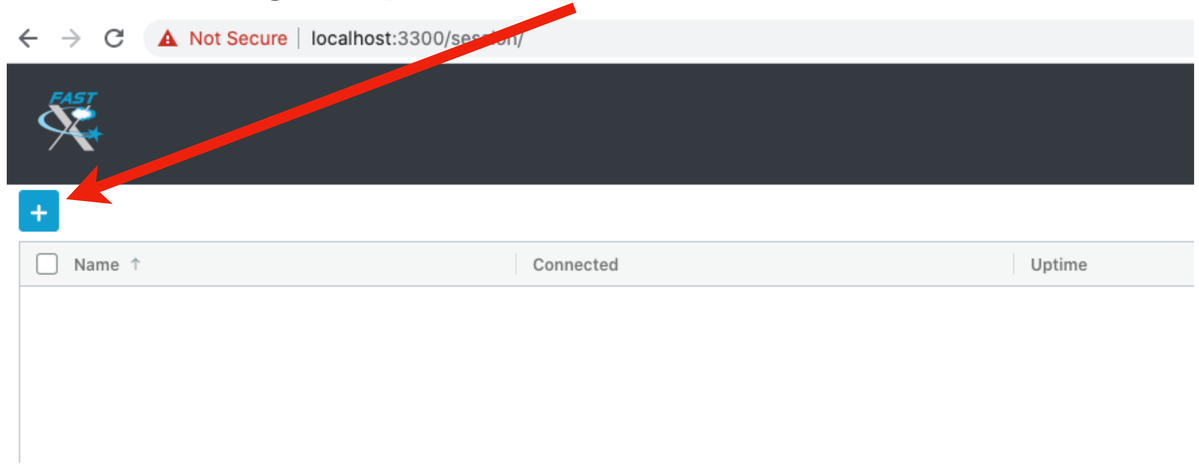


Fig: Step 8

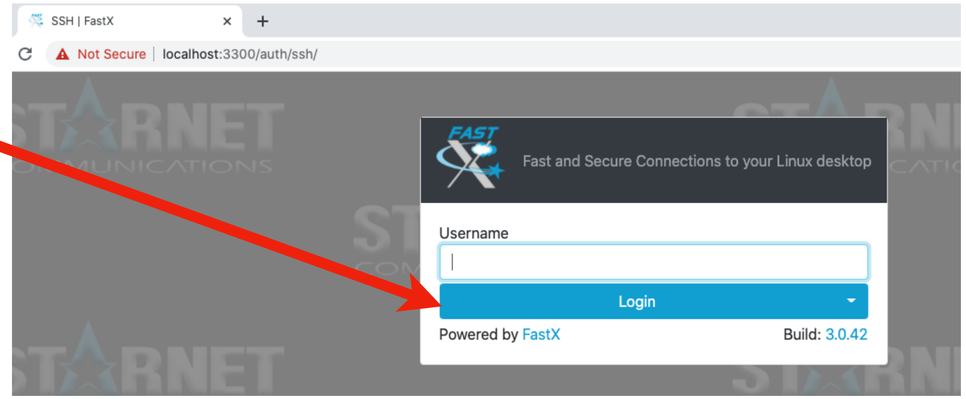


Fig: Step 6

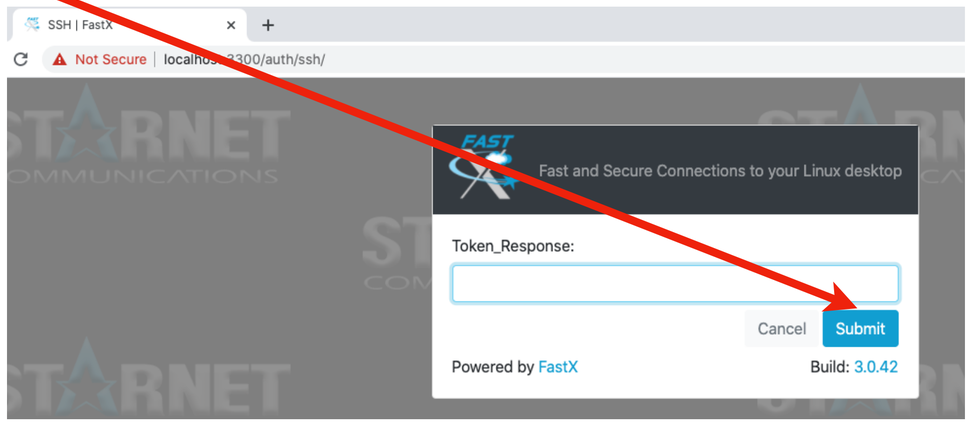


Fig: Step 7

Using Nsight Profiler on Casper (Using FastX via web browser) contd

Step 9: You should see a pop-up as in **Fig: Step 9**, click KDE circled in red and click Launch button

Step 10: You should see KDE desktop within your browser window as in **Fig: Step 10**, right-click (2 simultaneous button in your Mac touchpad) to get Menu, choose **Konsole** to get a terminal window

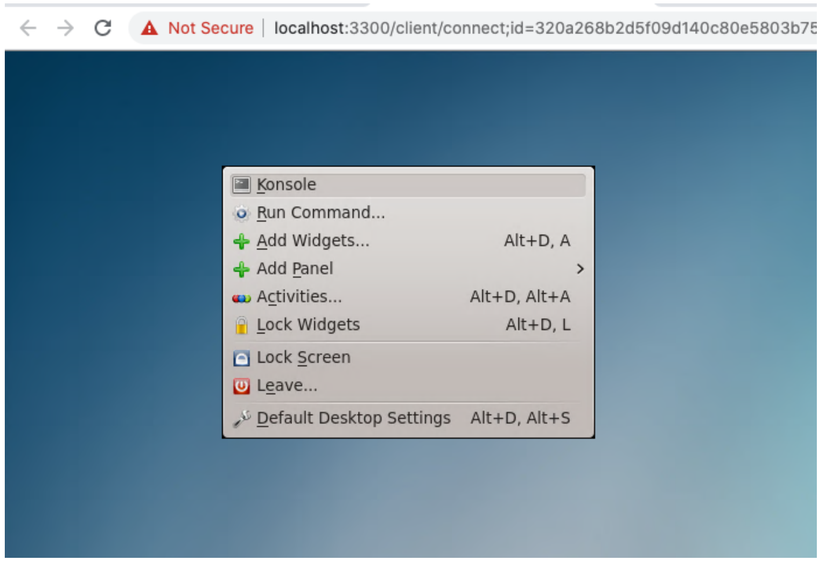


Fig: Step 10

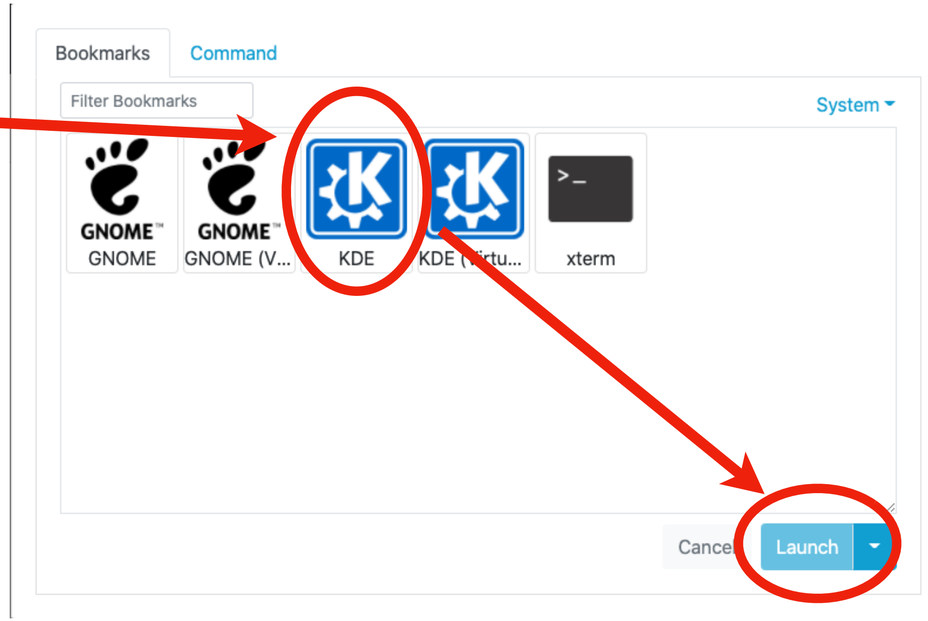


Fig: Step 9

For more detail: https://www2.cisl.ucar.edu/resources/computational-systems/casper/using-fastx-remote-desktops

Using Nsight on Casper

Step 11: 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 12: Run the build script (if necessary)

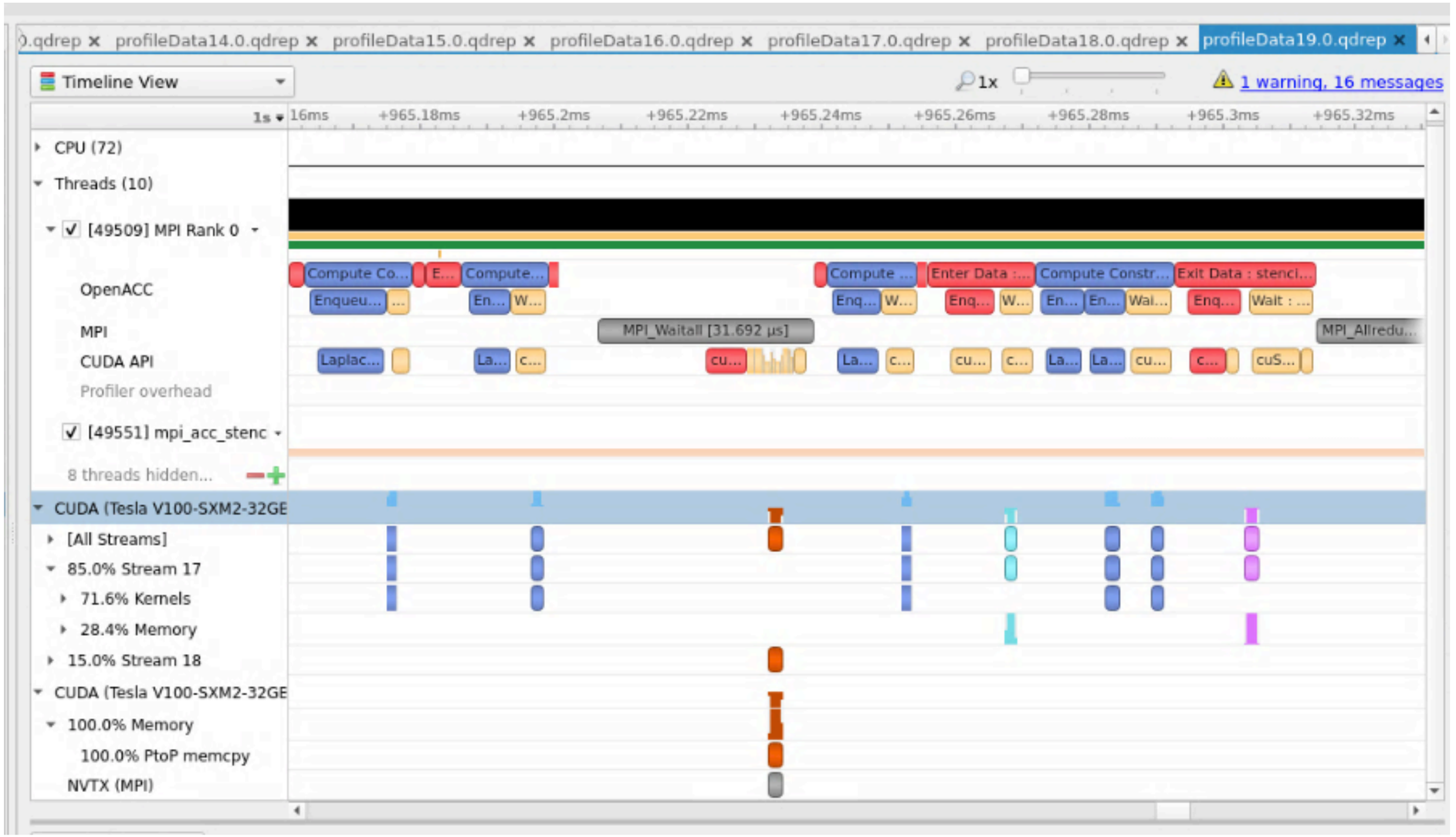
Now we profile!

Step 13: Uncomment the following command in the submit script to generate profiles:

```
mpirun -n 4 nsys profile -t ,openacc,mpi,nvtx --mpi-impl openmpi -o
profileData."%q{OMPI_COMM_WORLD_RANK}" ./mpi_acc_stencil.exe 64
```

Step 14: Finally, run the command "`nsight-sys profileData.qdrep`" to view your profile on the GUI

Profiling MPI with nvprof GUI



Summary Session 1

- GPUs and CPUs have different strengths:
  - CPUs have latency-oriented cores suitable for serial workloads
  - GPUs have throughput-oriented cores suitable for parallel workloads
- OpenACC is a simple directive-based language to port codes on to GPUs
- nvprof and PCAST are available to profile and validate the code running on GPUs

Summary Session 2

- CUDA gives more control to program GPUs
- Programmer can utilize most of the hardware present on the GPU using CUDA
- There are many levels of memory in NVIDIA GPUs, selecting the correct kind of memory is important for optimized performance.
- OpenACC is much easier to use compared to CUDA

Summary Session 3

- Software & hardware improvements allows transfer of GPU data without involving the CPU. This decreases execution time and makes programming easier.
- For MPI-OpenACC programs, we can divide workloads across GPUs by pinning a rank to each device
- Using CUDA-aware MPI allows us to port MPI applications to GPU
- Nsight GUI is another way to profile GPU codes