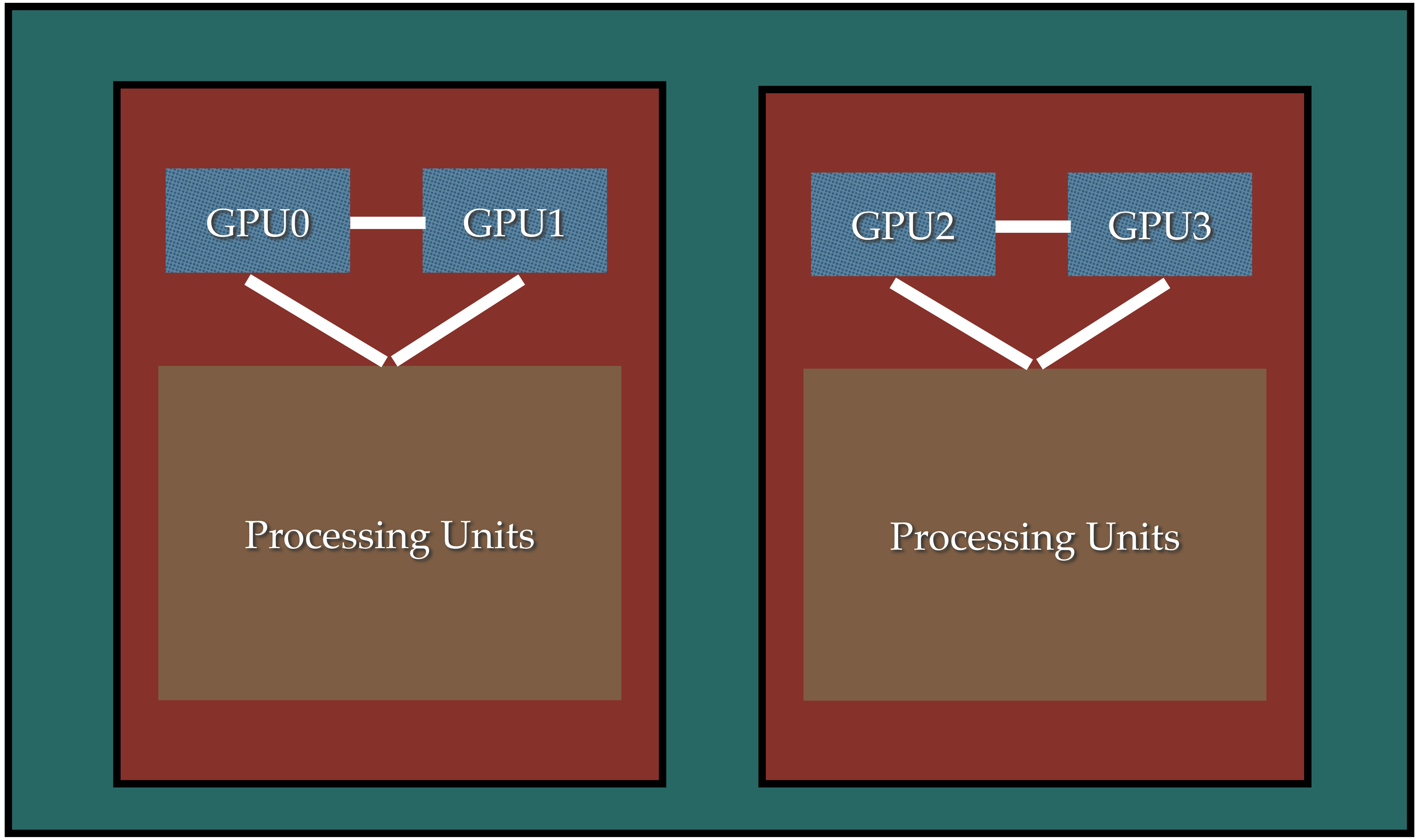
Introduction to Parallel Processing

Lecture 23 : MPI + CUDA

Professor Amanda Bienz

Heterogeneity in Systems



Many more layers of data movement
Different rates of computation

Compiling and running code

- Need to compile both CUDA and MPI code together
- On Xena, the following should work:
  - nvcc -ccbin=mpicxx -arch=compute_35 <filename>
- To run the code, just use standard MPI notation:
  - mpirun -n <num_procs> ./filename
- Make sure to load correct modules:
  - module load cuda
  - **module load openmpi/4.1.4-7gqe (version with CUDA-aware support)**

How to write a heterogeneous program

1. Create MPI ranks : these correspond to CPU processes
2. Have CPU processes offload local computation to a corresponding GPU
3. Communicate data, as needed, between MPI ranks
4. Copy any results from GPU back to CPU ranks

A bit more complicated than this…

- How many MPI ranks should I use, per node?
- Typically, Power9 systems have around 40 available CPU cores per node, but between 4 and 6 GPUs per node
- Possible setups:
  - Use a single CPU core to control all GPUs on the node
  - Use a single CPU core per GPU on the node (4-6 total CPU cores per node)
  - Use all 40 CPU cores, and have 6-10 CPU cores per GPU

Simplest Approach

- Let’s assume we have one CPU core corresponding to each GPU core
- In this case, node_rank (rank of process in shared memory communicator) corresponds to the GPU ID that this process will offload to
- For example: assume we have 4 GPUs per node, and also 4 MPI processes per node (node_ranks = 0,1,2,3 and gpu ids = 0,1,2,3)
- cudaSetDevice(int gpu) : sets device to which I will offload

How to communicate data

- On heterogeneous systems, we can assume all data to be communicated starts on a GPU, with a final destination of another GPU

- Straightforward way to communicate data :

  1. Copy original data to a CPU core

  2. MPI communication

  3. Copy received data to GPU

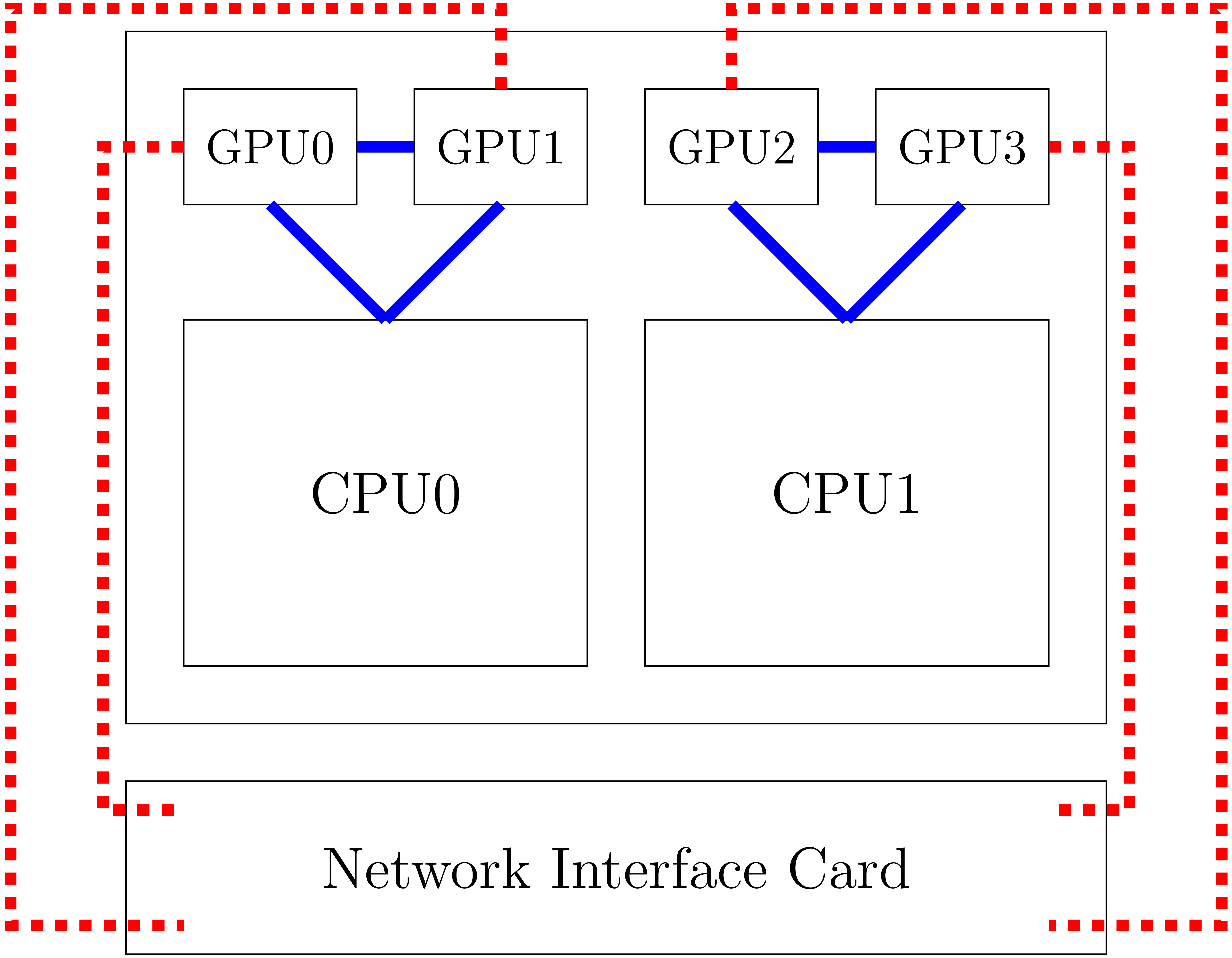
CUDA Aware MPI

- Recent versions of MPI support CUDA Aware MPI
- You can call MPI functions directly from GPU memory
- To do this:
    - Allocate pointer in device memory (cudaMalloc(…))
    - Call MPI method (MPI_Send/MPI_Recv/some collective) with pointer to CUDA memory passed
    - Communication will happen behind the scenes

CUDA Aware MPI

- How CUDA Aware MPI Works :
  - One option : same way that you would by hand, first copying data to CPU memory, communicating between CPU memories, and finally copying back to GPU
  - **GPUDirect** : on newer systems, communication can happen without first copying to the CPU! Transfer data directly from GPU memory to network (ideally faster than first method)

GPU Direct



Is CUDA-Aware MPI Supported?

- This is not supported in the majority of MPI versions on Xena
- To check, you can type the following line into the terminal after loading MPI
- ompi_info —parsable —all | grep mpi_built_with_cuda_support:value
  - **XENA: mca:mpi:base:param:mpi_built_with_cuda_support:value:false**

Unified Memory

- cudaMallocManaged(…) allocates data that can be accessed by either CPU or GPU
  - CUDA manages the transfer between host and device memory for you
  - ```
    int* array;
    cudaMallocManaged((void**)&array, size);
    cpu_method(array);
    gpu_method<<<n_blocks, block_size>>>(array);
    ```
  - Typically, results in poor performance
    ```