Lecture 11: Distributed computing - Part 2

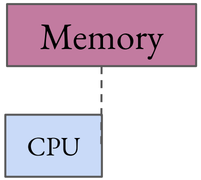
Informatik elective: GPU Computing

Pratik Nayak

In this session

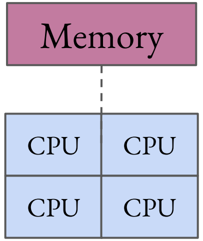
- Distributed multi-GPU computing
    - GPUDirect RDMA
    - NCCL: The collective communications library.
    - NVSHMEM: An OpenSHMEM adaptation for NVIDIA GPUs
    - Demos

Under the hood: Architecture of a node



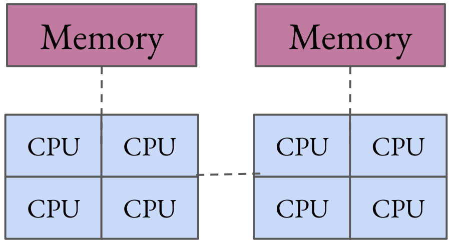
[Credits: Simon Garcia, SNL]

Under the hood: Architecture of a node



[Credits: Simon Garcia, SNL]

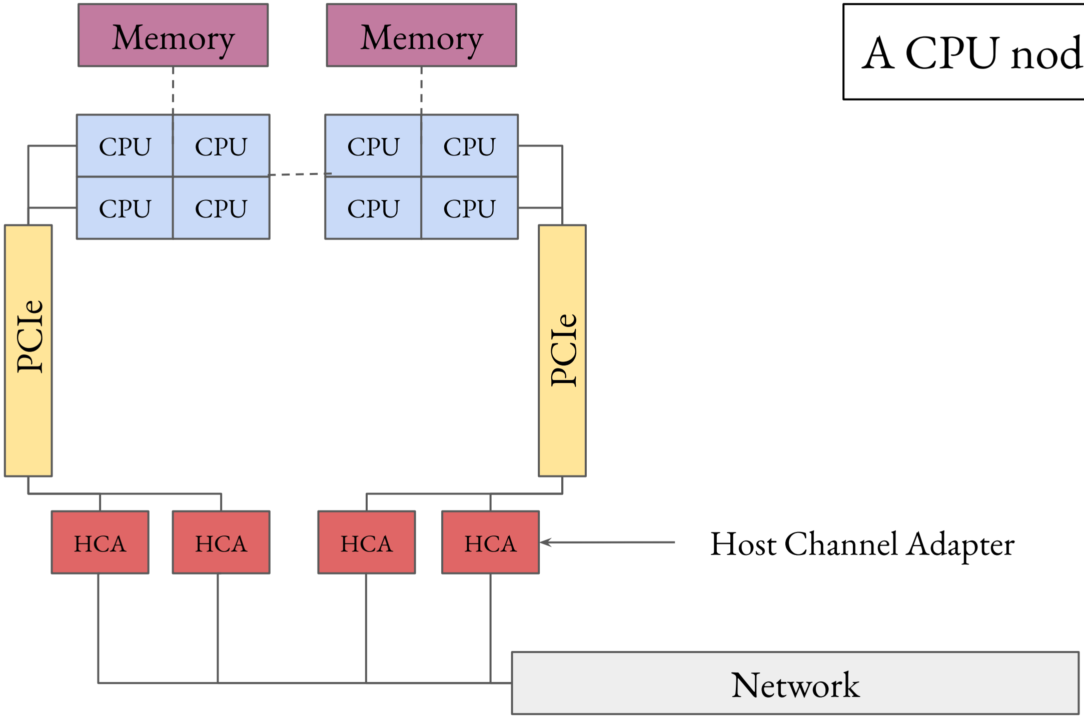
Under the hood: Architecture of a node



NUMA nodes

[Credits: Simon Garcia, SNL]

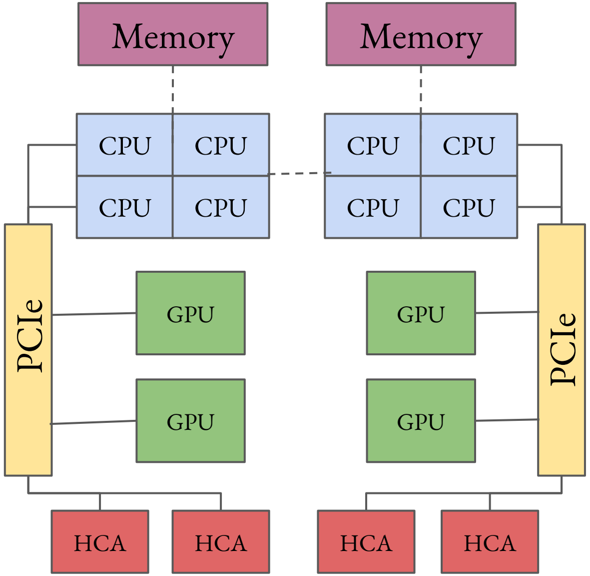
Under the hood: Architecture of a node



A CPU node

[Credits: Simon Garcia, SNL]

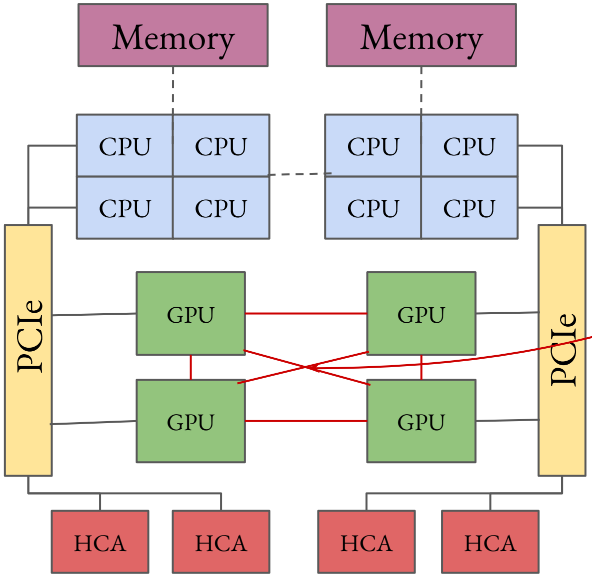
Under the hood: Architecture of a node



A GPU node

[Credits: Simon Garcia, SNL]

Under the hood: Architecture of a node

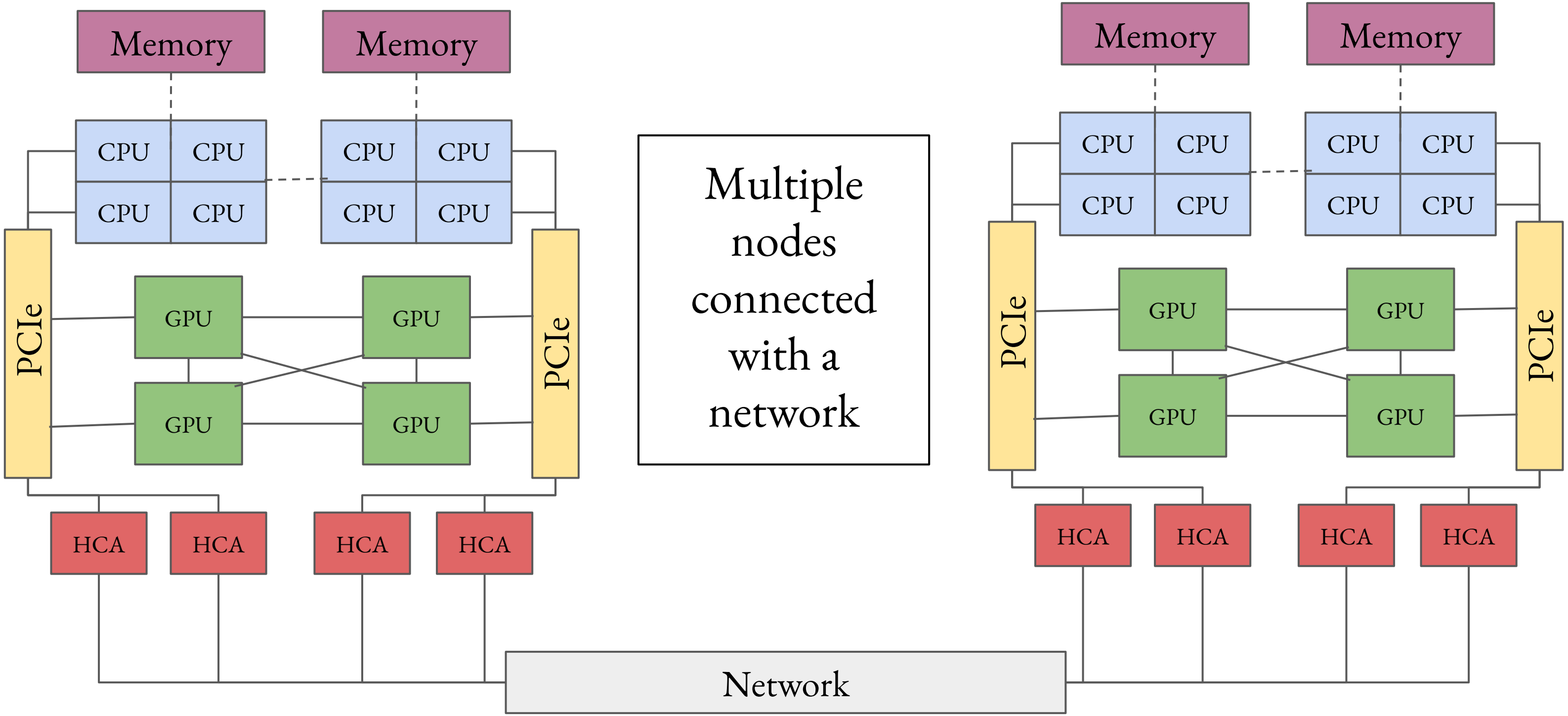


A GPU node

- GPUs on single node can be interconnected (for example for NVIDIA GPUs: NVLINK)

[Credits: Simon Garcia, SNL]

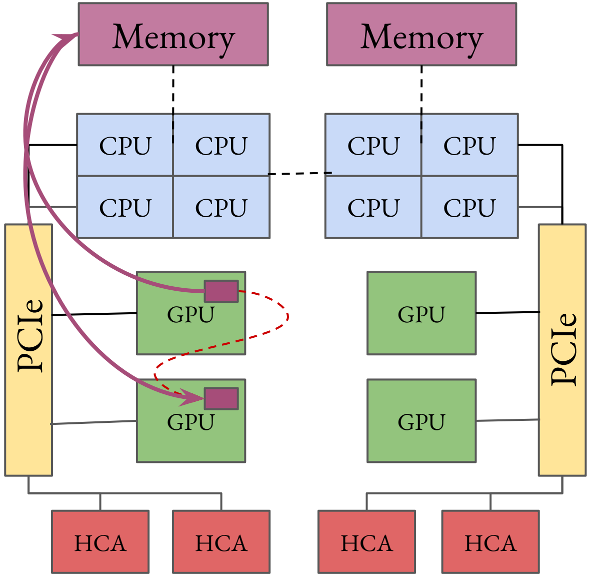
Under the hood: Architecture of a node



[Credits: Simon Garcia, SNL]

Lower latencies with GPUDirect

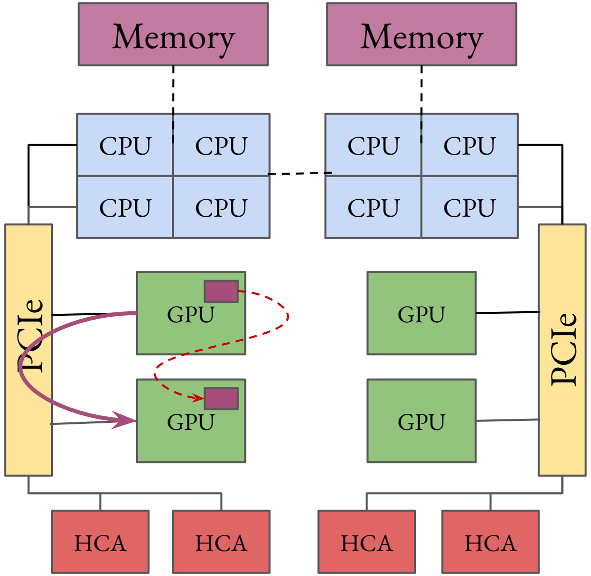
- GPU to GPU on a single node will communicate through the main memory
- Higher latency: GPU→ PCIe →Memory → Memory → PCIe → GPU



[Credits: Simon Garcia, SNL]

Lower latencies with GPUDirect

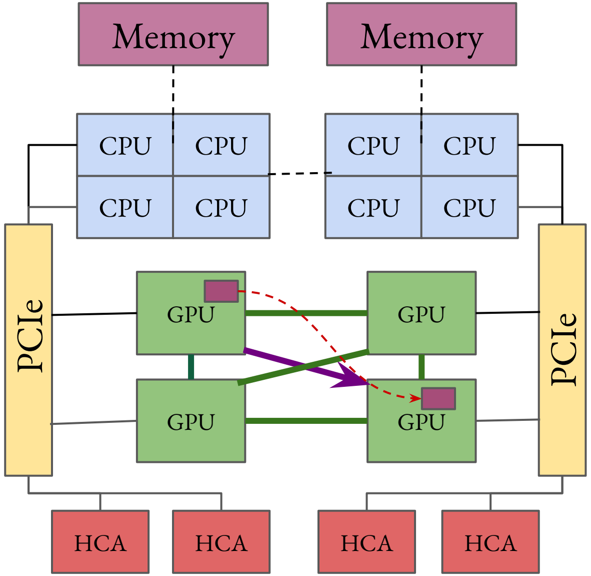
- GPU to GPU on a single node will communicate through the main memory
- Higher latency: GPU→ PCIe →Memory → Memory → PCIe → GPU
- With GPUDirect, the GPU driver chooses the shortest path: Through the PCIe express
- On a single node, this is called GPUDirect P2P



[Credits: Simon Garcia, SNL]

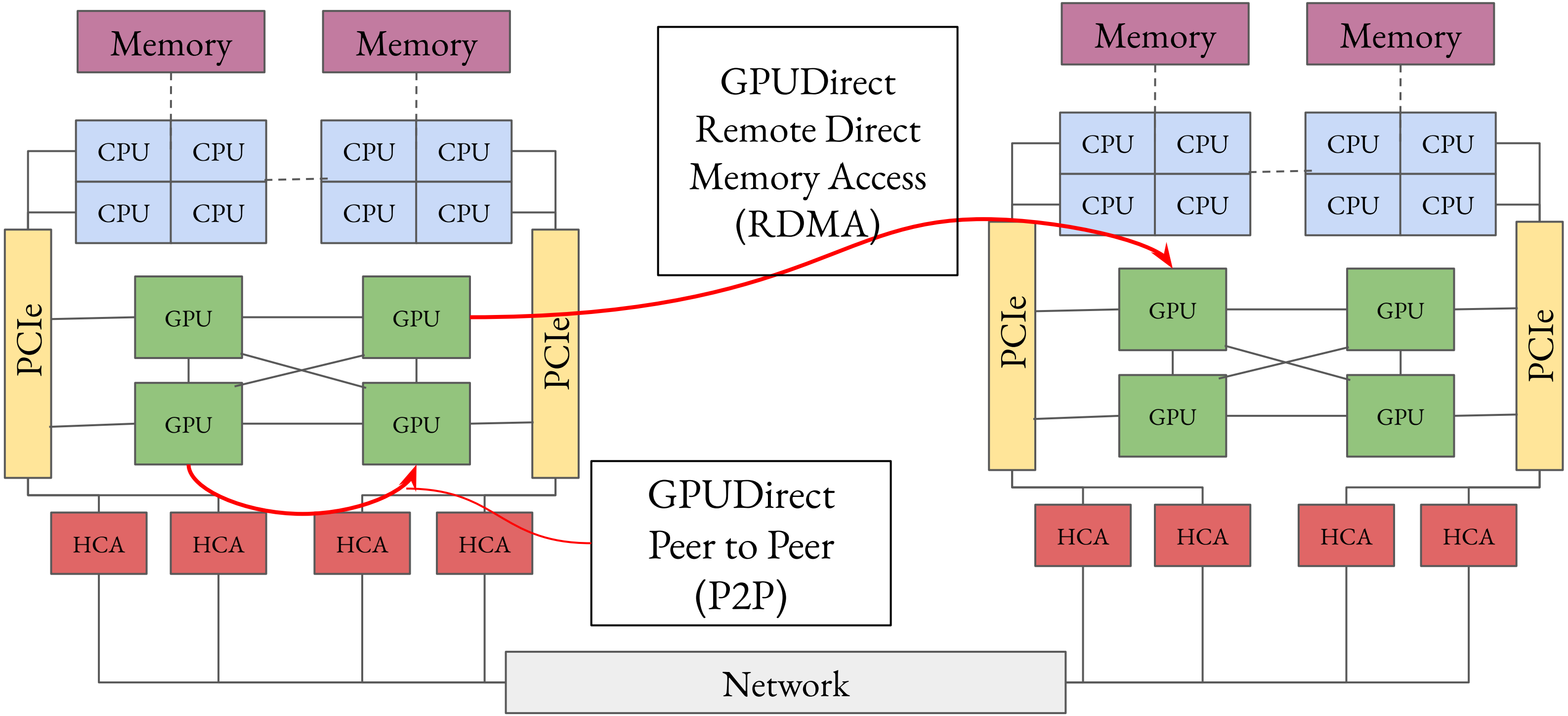
Lower latencies with GPUDirect

- GPUs can also have direct connections: NVLINK
- GPUDirect automatically chooses these when available.
- Off-node data transfers can also be staged through NVLINK – PCIe
- Called GPUDirect RDMA (Remote Direct Memory Access).



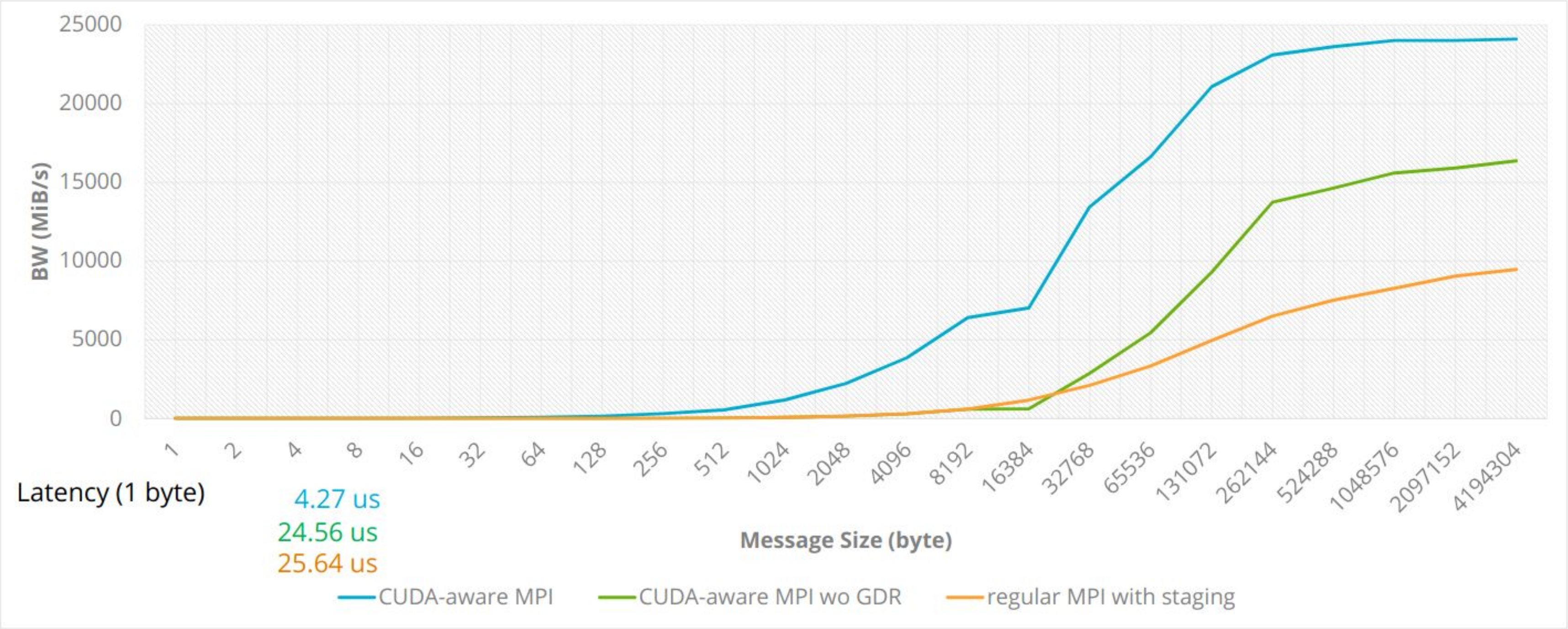
[Credits: Simon Garcia, SNL]

Lower latencies with GPUDirect



[Credits: Simon Garcia, SNL]

Performance: Using GPUDirect RDMA (off-node)



OpenMPI 4.1 + UCX on Juwels Booster (JSC)

[Credits: Simon Garcia, SNL]

Collective communication with NCCL

- Collective communication is important in many applications (See Lecture 10):
  - Broadcast, Allreduce, AllGather, ReduceScatter etc.
- Using NCCL (NVIDIA Collective Communications library) API, you can get portable performance for inter-GPU collective communication, and also point to point communication.
- The API closely resembles MPI:
  - Get unique_id on one rank and broadcast to other ranks.
  - Create a NCCL communicator,
  - Using the NCCL communicator to call the collective functions.
  - Can assign a stream to the collective call.

Collective communication with NCCL

```
int main(int argc, char* argv[])
{
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myRank)
...
if (myRank == 0) ncclGetUniqueId(&id);
MPI_Bcast((void *)&id, sizeof(id), MPI_BYTE, 0, MPI_COMM_WORLD);
ncclCommInitRank(&comm, nRanks, id, myRank);
...
ncclAllReduce((const void*)sendbuff, (void*)recvbuff, size, ncclFloat, ncclSum, comm, stream);
...
ncclCommDestroy(comm);
...
}
```

Graph captures

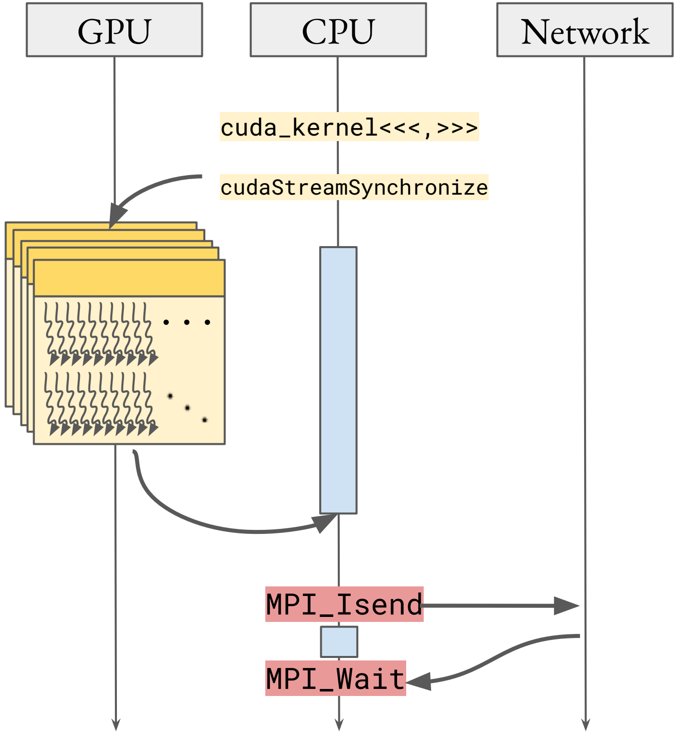
- As NCCL operations take stream parameters, they can be captured in a CUDA graph
- Follows the same structure as a regular CUDA graph:
    - Using stream capture, capture the operation
    - Instantiate the graph.
    - And then launch the graph instance repeatedly.
- Reduces the kernel launch overhead and overall lower latencies.

```
cudaGraph_t graph;
cudaStreamBeginCapture(stream);
kernel_A<<< ..., stream >>>(...);
kernel_B<<< ..., stream >>>(...);
ncclAllreduce(..., stream);
kernel_C<<< ..., stream >>>(...);
cudaStreamEndCapture(stream, &graph);

cudaGraphExec_t instance;
cudaGraphInstantiate(&instance, graph, NULL,
NULL, 0);
cudaGraphLaunch(instance, stream);
cudaStreamSynchronize(stream);
```

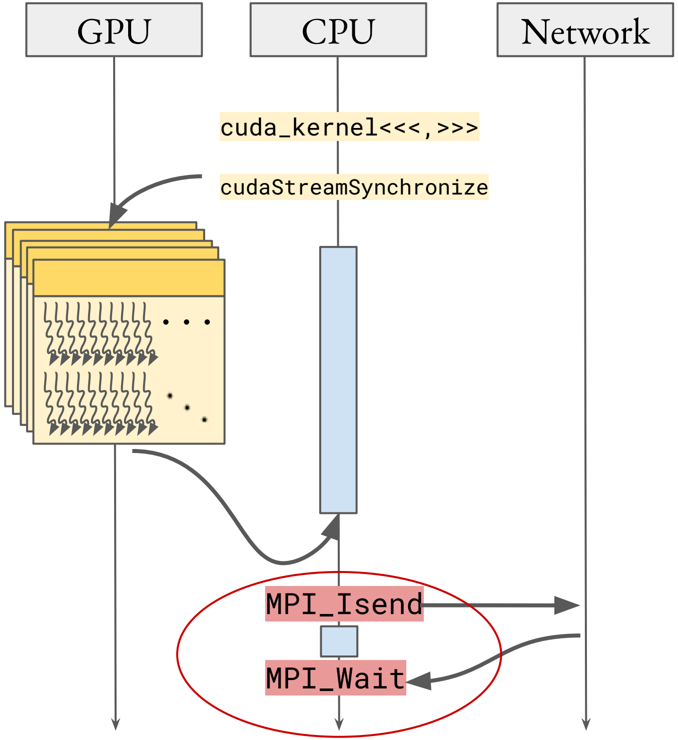
Distributed multi-GPU computing with MPI

- Call a GPU kernel, and compute on the GPU
- Concurrently compute on the CPU
- After kernel completion, communicate with other nodes.
  - GPU-Aware ? transfer GPU buffer directly
  - Otherwise, transfer via host.
- Use MPI functions to communicate.



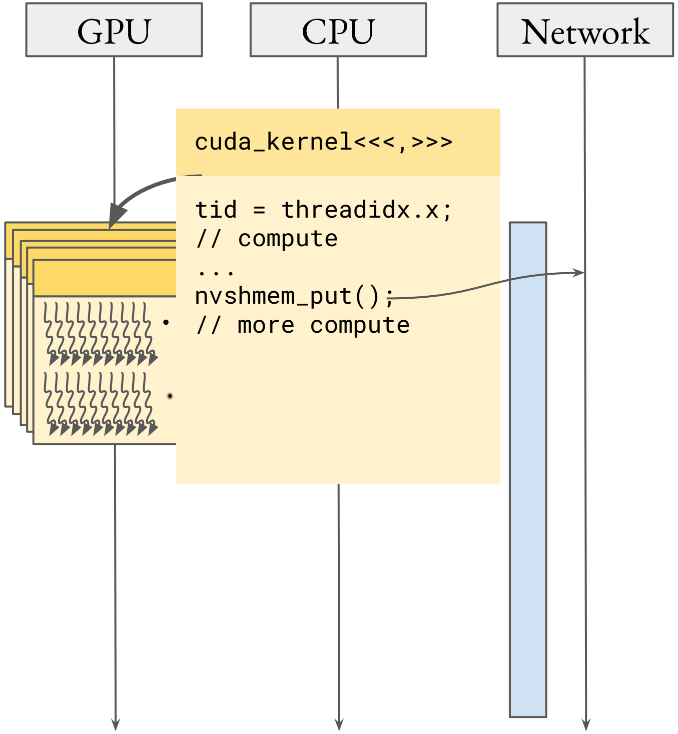
Caveats

- CPU initiated communication
- Communication outside the GPU kernels → More kernels necessary.
- Increased latencies, even when GPU Aware MPI is available.
- GPU and CPU have enforced dependencies and hence cannot be fully concurrent.



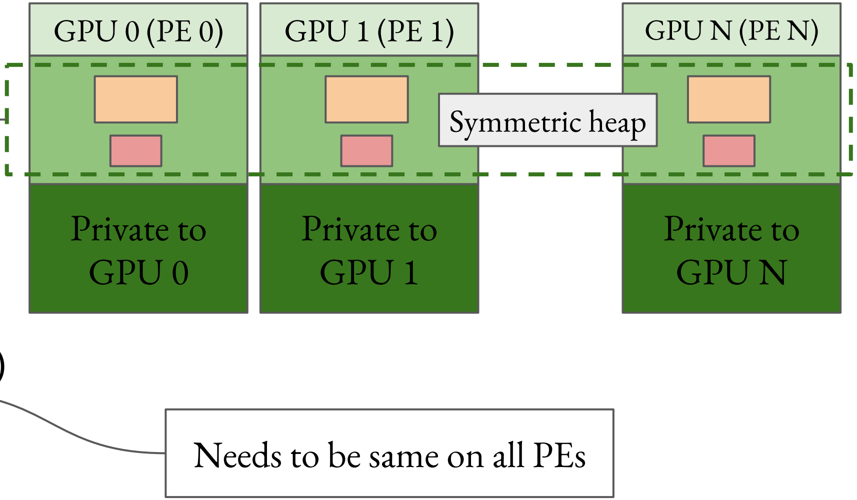
NVSHMEM

- With NVSHMEM, we can initiate communication from device kernels
- Better communication/computation overlap.
- Lower latencies and overheads.
- NVSHMEM: An extension of OpenSHMEM model (https://openshmem.org/site/)



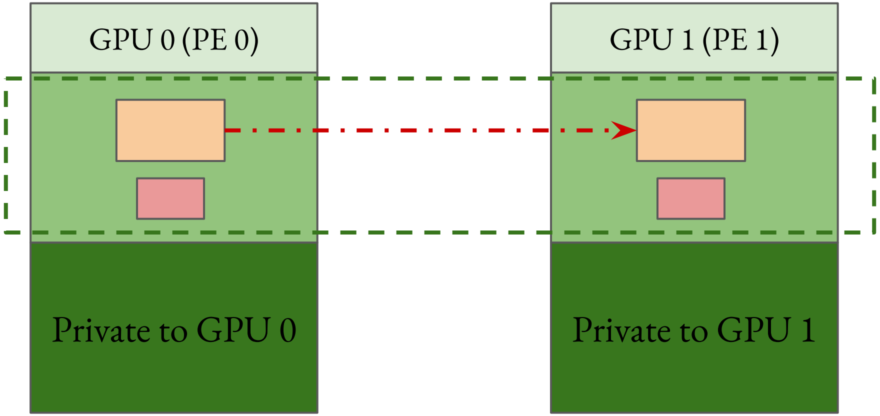
NVSHMEM basics

- A distributed shared memory programming model.
  - Partitioned Global Address Space (PGAS)
- Symmetric objects allocated with NVSHMEM:
  - Shared between all Processing elements
  - Allocated with `nvshmem_alloc(shared_size)`
- Private memory allocated as usual:
  - Using `cudaMalloc(...)`
- Communicate with one-sided functions (put and get)



[Credits: Lena Oden, FUH]

NVSHMEM Host API put

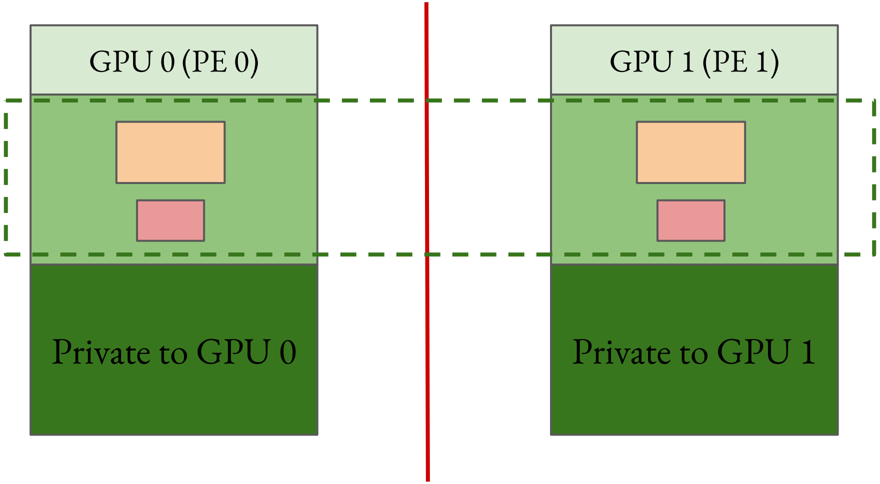


Copies `nelems` of type `T` from symmetric objects `source` to `dest`.

```
void nvshmem_<T>_put(T*dest, const T*source, size_t nelems, int pe);
void nvshmemx_<T>_put_on_stream(T*dest, const T*src, size_t nelems, int pe, cudaStream_t stream);
```

[Credits: Lena Oden, FUH]

NVSHMEM Host API barrier



Synchronize all PEs to ensure that communication has finished.

```
void nvshmem_barrier_all(void);
void nvshmemx_barrier_all_on_stream(cudaStream_t stream)
```

[Credits: Lena Oden, FUH]

MPI + NVSHMEM

- Supports interoperation with MPI.
- Create N processes, initialize attributes with MPI_COMM.
- Allocate symmetric objects with `nvshmem_malloc()`
- Kernel can then directly call nvshmem_put, barrier calls etc.
- Free objects and need to finalize to cleanup

```
MPI_Init(&argc, &argv);
MPI_Comm mpi_comm = MPI_COMM_WORLD;
nvshmemx_init_attr_t attr;
attr.mpi_comm = &mpi_comm;
nvshmemx_init_attr(NVSHMEMX_INIT_WITH_MPI_COMM,
&attr);
assert( size == nvshmem_n_pes() );
assert( rank == nvshmem_my_pe() );
...
call_kernel<<<>>>(); // contains nvshmem_put calls
...
nvshmem_finalize();
MPI_Finalize();
```

Demos

- Examples of different multi-gpu programming models:

  https://github.com/NVIDIA/multi-gpu-programming-models/

Summary

- Distributed multi-gpu computing:
    - Hardware and programming models
    - GPUDirect and GPU-Awareness
- NCCL: Collective communication library
- NVSHMEM: PGAS model for communication within kernels.

Next lecture (Last lecture)

- Programming models for other GPUs: AMD, Intel
  - HIP, ROCM, SYCL
- Portable GPU programming.
  - Kokkos, RAJA, OCCA
- LLVM Intermediate representation (IR)