

GPU-Aware MPI: Multi-GPU Programming for HPC Systems



Norwegian research infrastructure services

Hicham Agueny, PhD
Scientific Computing Group
Info.Tech. Department, UiB/NRIS



Σ

Main goal:

**To initiate your interest in combining GPU programming models
with a MPI library**



Motivation

Synchronous OpenACC/OpenMP offloading: **limited to single GPU.**

Asynchronous OpenACC/OpenMP offloading: **multiple GPUs BUT limited to a single node.**

Accelerating existing MPI-based codes.

Fully utilise the capacity of exascale supercomputer such as LUMI.

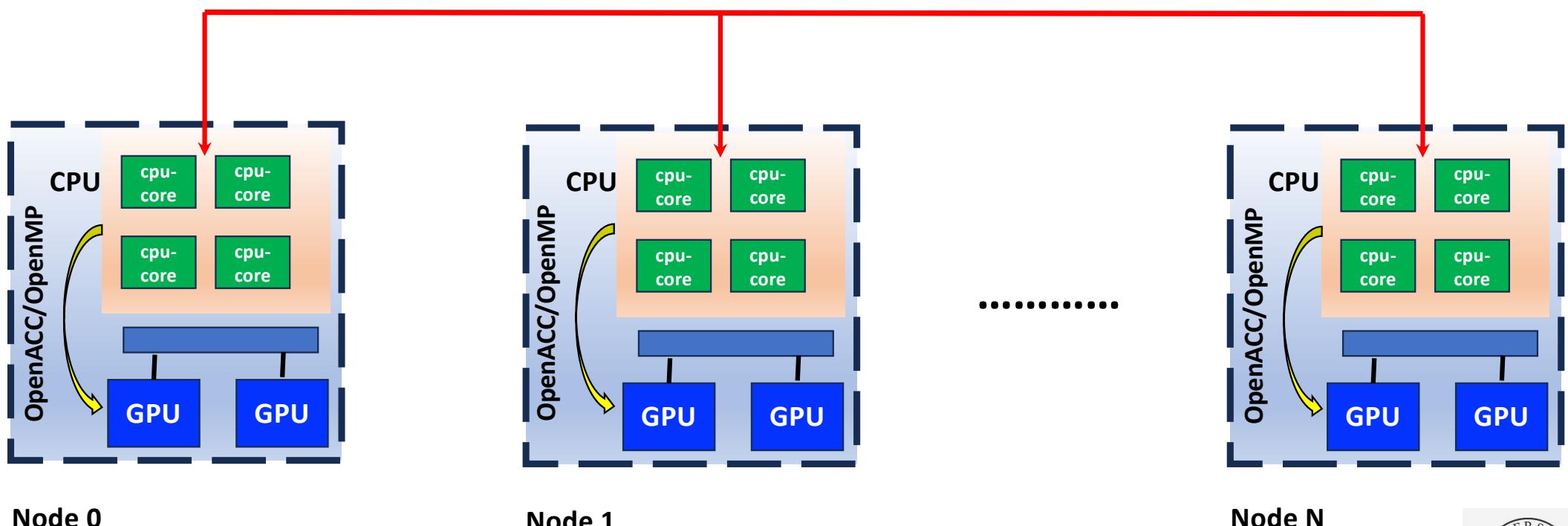
There is a need of multiple GPU programming.



Multiple GPU with MPI communications

Distributed computing

MPI communication library



Outline

□ Concept of:

- Traditional MPI without GPU awareness (with involvement of the CPU memory)
- MPI with GPU awareness (without involvement of the CPU memory)

□ Application: MPI-OpenACC & MPI-OpenMP offloading

- Point-to-point communication (MPI_Send & MPI_Recv)

□ Conclusion & Benchmark

□ Hands-on examples and exercises



Learning Outcomes

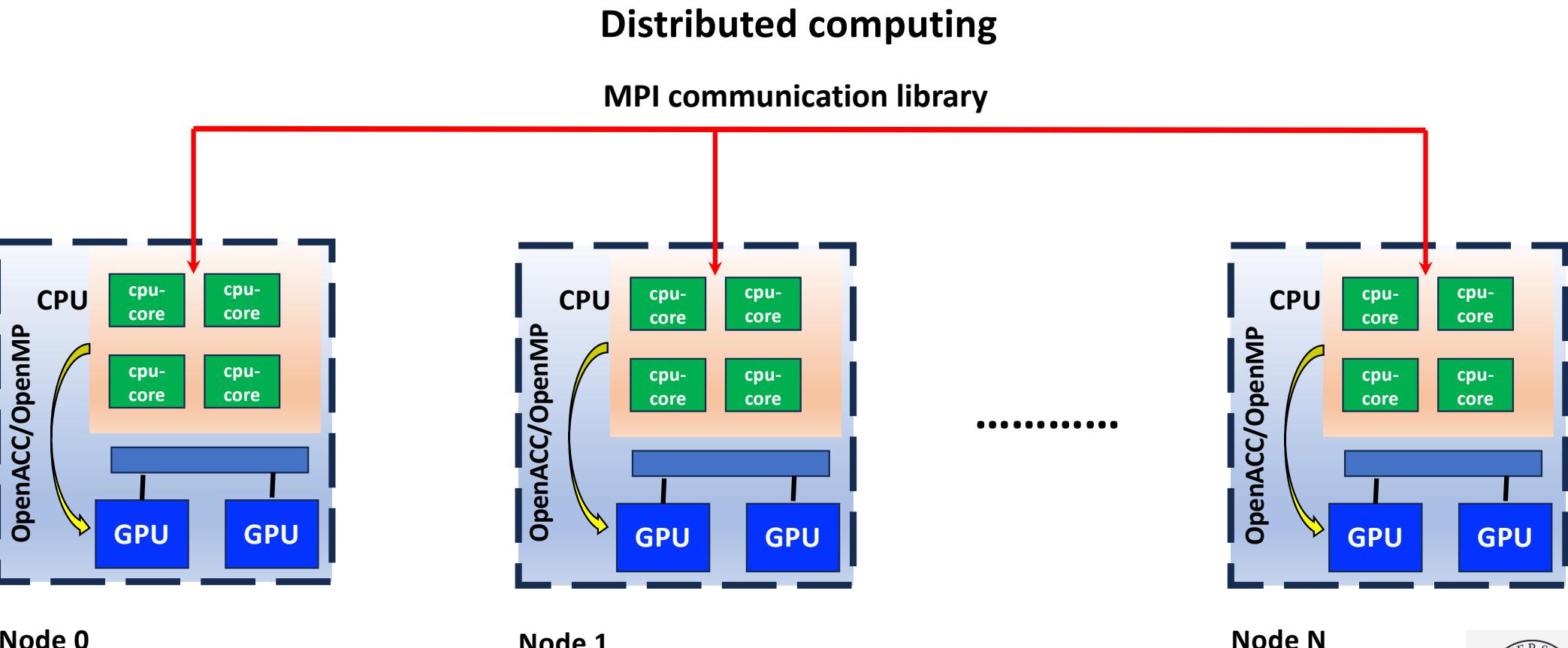
- ❑ How to assign each MPI rank to a GPU device ?
- ❑ How to combine MPI and OpenACC/OpenMP
 - MPI operations : **MPI_Send** and **MPI_Recv**
 - OpenACC directives: **update host(); update device()**
 - OpenMP directives: **update device() from(); update device() to()**
- ❑ How to perform MPI operations with GPU-awareness support
 - MPI operations : **MPI_Send** and **MPI_Recv**
 - OpenACC directive: **host_data device_ptr()**
 - OpenMP directive: **use_device_ptr()**

See our documentation for further details

https://enccs.github.io/gpu-programming/8-multiple_gpu/



Multiple GPU with MPI communications

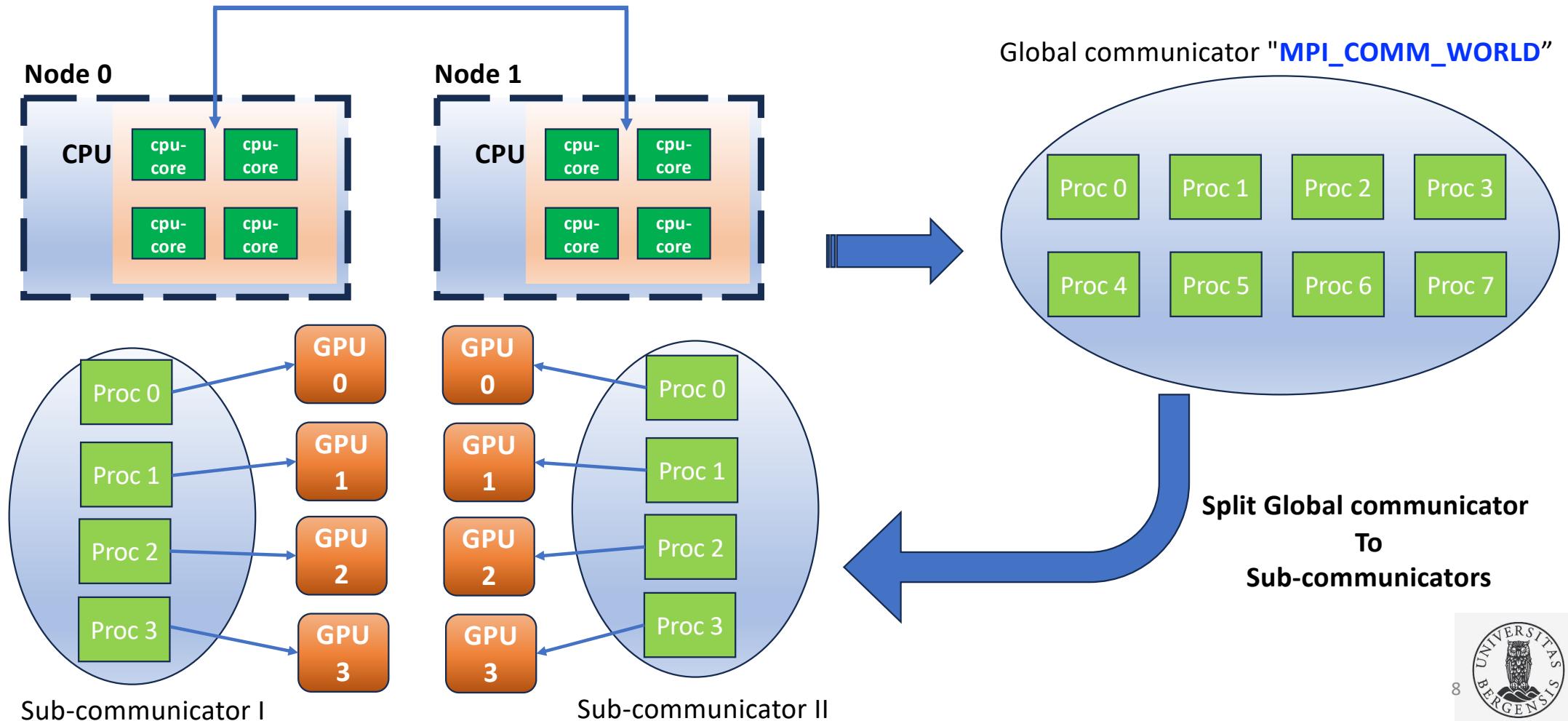


How to assign each MPI rank to a GPU device ?



How to assign each MPI rank to a GPU device ?

Split the global communicator "MPI_COMM_WORLD" into sub-communicators



How to assign each MPI rank to a GPU device ?

MPI functions

```
call MPI_COMM_SPLIT_TYPE(MPI_COMM_WORLD,  
                         MPI_COMM_TYPE_SHARED, 0,  
                         MPI_INFO_NULL, host_comm, ierr)  
  
call MPI_COMM_RANK(host_comm, host_rank, ierr)
```

Sub-communicator

New MPI rank

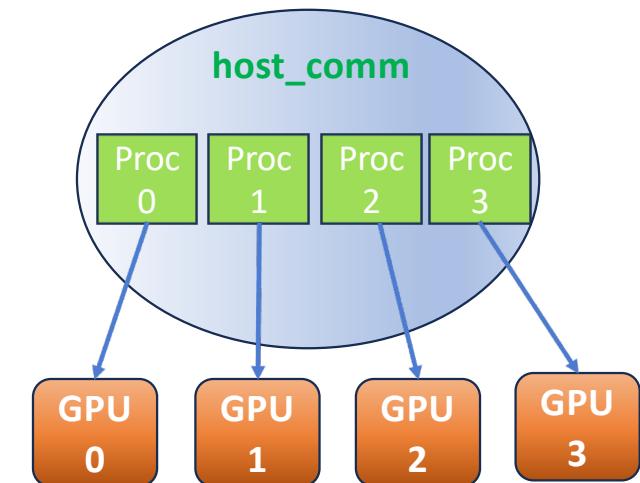
API functions: OpenACC

```
! Sets the device INDEX and the device type to be used  
call acc_set_device_num(host_rank, acc_get_device_type())
```

! Returns the number of devices available on the host (here on each node)

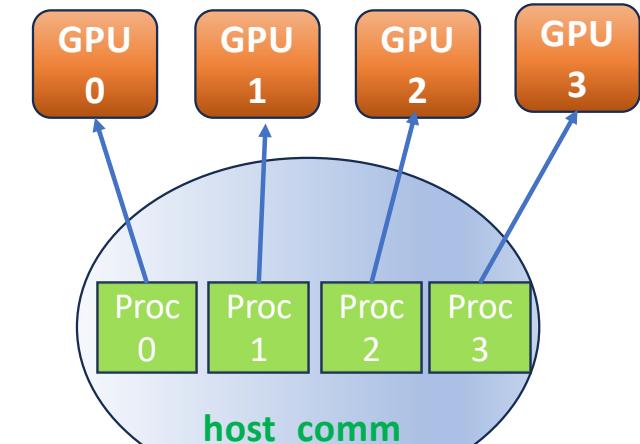
```
numDevice = acc_get_num_devices(acc_get_device_type())
```

Sub-communicator



host_comm

Sub-communicator



9

How to assign each MPI rank to a GPU device ?

```
call MPI_COMM_SPLIT_TYPE(MPI_COMM_WORLD,  
                         MPI_COMM_TYPE_SHARED, 0,  
                         MPI_INFO_NULL, host_comm, ierr)  
  
call MPI_COMM_RANK(host_comm, host_rank, ierr)
```

MPI functions

API functions: OpenACC

! Sets the device INDEX and the device type to be used

```
call acc_set_device_num(host_rank, acc_get_device_type())
```

! Returns the number of devices available on the host (here on each node)

```
numDevice = acc_get_num_devices(acc_get_device_type())
```

API functions: OpenMP

! Sets the device INDEX and the device type to be used

```
call omp_set_default_device(host_rank)
```

! Returns the number of devices available on the host (here on each node)

```
numDevice = omp_get_num_devices()10
```

Hands-on Example 1 (15 min):

How to assign each MPI rank to a GPU device ?

- Download the repo

```
$ git clone https://github.com/HichamAgueny/multigpu\_mpi\_course.git
$ cd multigpu\_mpi\_course
```

Task: Review the codes & Slurm scripts and run the following:

- For MPI-OpenACC example

```
$ cd example_1/setDevice_acc
```

- For MPI-OpenMP example

```
$ cd example_1/setDevice_omp
```

- To compile and execute the code

Load the LUMI software stack

```
$ module load LUMI/24.03 partition/G
$ module load cpeCray
```

Compile: \$./compile.sh

Submit a job: \$ sbatch script.slurm

View the output file: \$ vi setDevice_accxxxxxx.out

Access Slides & Codes

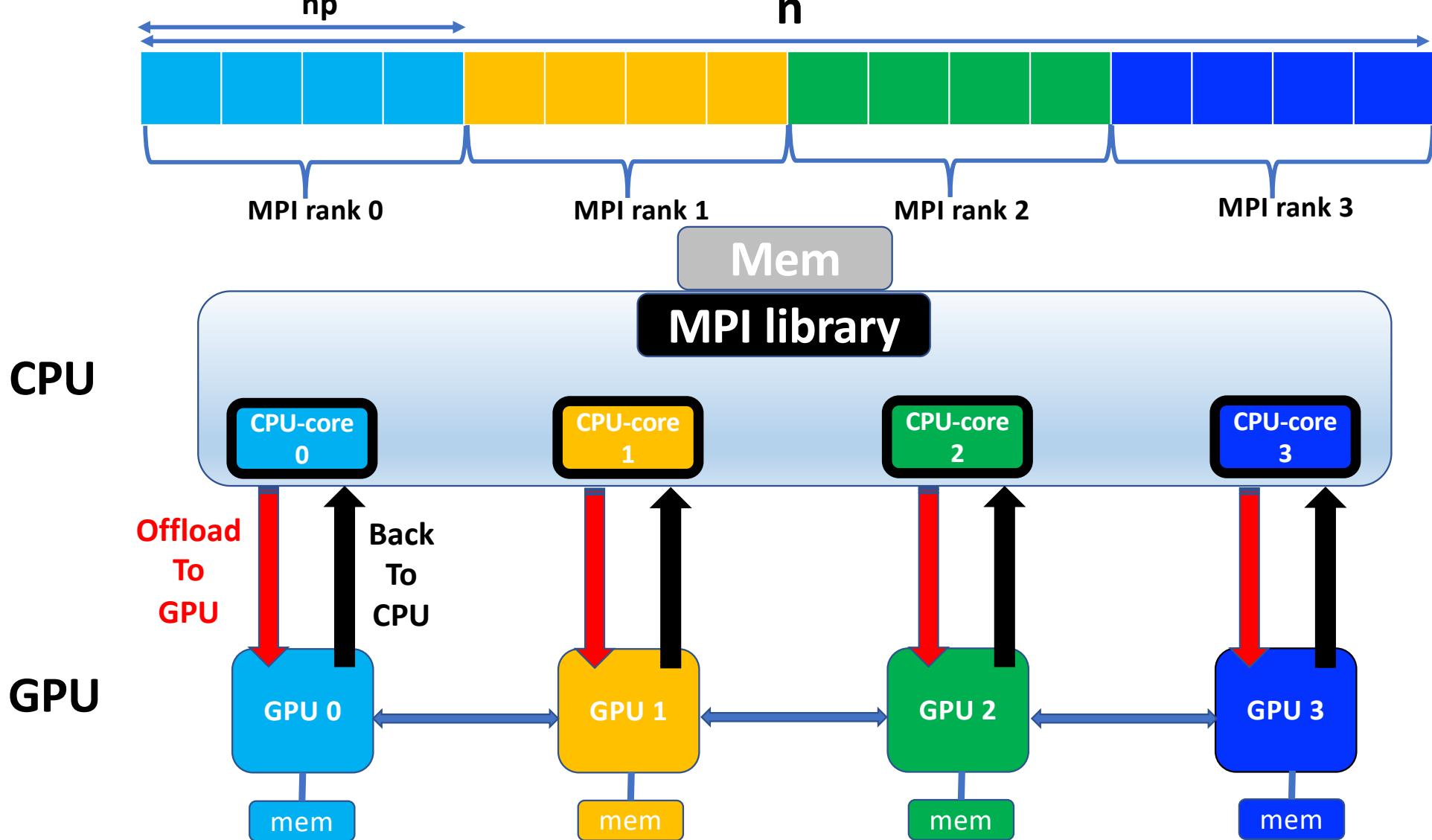


Concept:

Traditional MPI without GPU awareness



Traditional MPI without GPU awareness

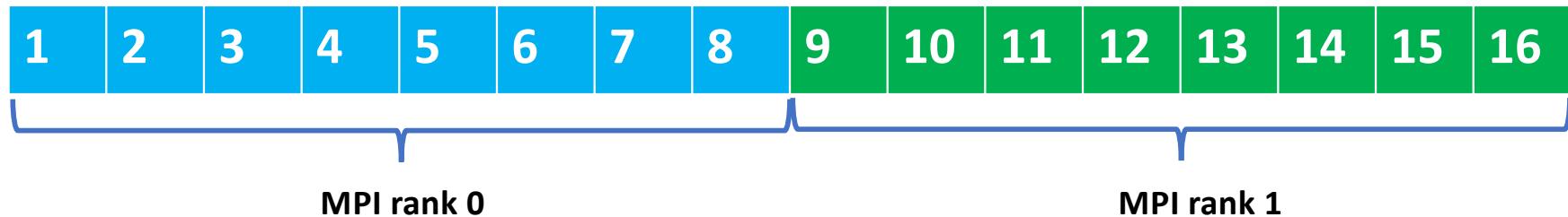


Example: Combining MPI with OpenACC/OpenMP APIs

- MPI operations : **MPI_Send & MPI_Recv**
- OpenACC directives: **update host(); update device()**
- OpenMP directives: **update device() from(); update device() to()**



Scenario with only 2 MPI-processes: MPI_Send & MPI_Recv



```
if(MPIrank.eq.0) then
    call MPI_Send(f_send(:), N, MPI_DOUBLE_PRECISION, MPIrank=1, tag1, ...)
    call MPI_Recv(f_recv(:), N, MPI_DOUBLE_PRECISION, MPIrank=1, tag2, ...)
endif
if(MPIrank.eq.1) then
    call MPI_Recv(f_recv(:), N, MPI_DOUBLE_PRECISION, MPIrank=0, tag1, ...)
    call MPI_Send(f_send(:), N, MPI_DOUBLE_PRECISION, MPIrank=0, tag2, ...)
endif
```

MPI-OpenACC: MPI_Send & MPI_Recv

`!$acc enter data copyin(f_send,f_recv)`

Perform computation on GPU

`!$acc update host(f_send,f_recv)`

Do MPI operations on CPU

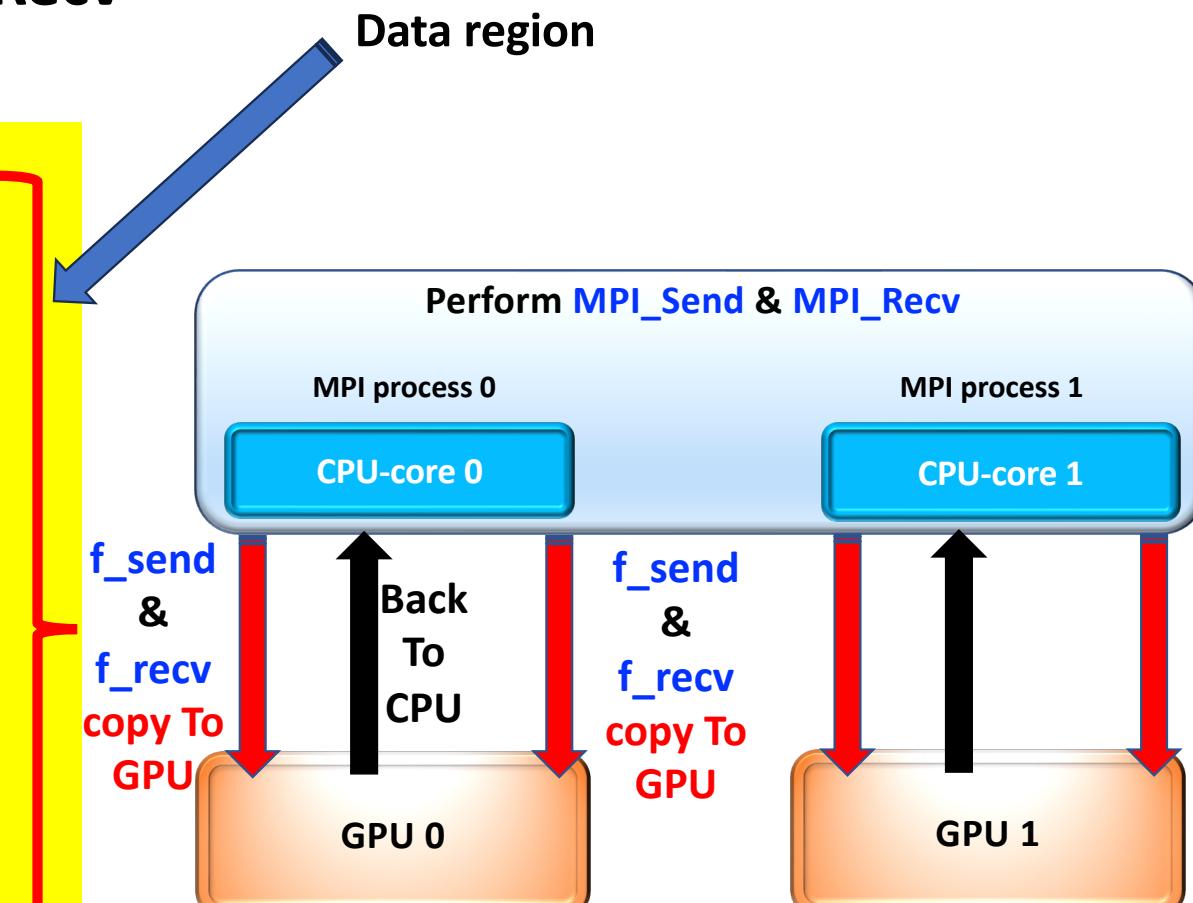
`call MPI_Send(f_send, N ,`

`call MPI_Recv(f_recv, N,`

`!$acc update device(f_send,f_recv)`

Perform computation on GPU

`!$acc exit data copyout(f_recv)`



MPI-OpenACC

```
!$acc enter data copyin(f_send,f_recv)
```

Perform computation on GPU

```
!$acc update host(f_send,f_recv)
```

Do MPI operations on CPU

```
call MPI_Send(f_send, N , .....)
```

```
call MPI_Recv(f_recv, N , .....)
```

```
!$acc update device(f_send,f_recv)
```

Perform computation on GPU

```
!$acc exit data copyout(f_recv)
```

MPI-OpenMP

```
!$omp target enter data map(to: f_send,f_recv)
```

Perform computation on GPU

```
!$omp target update from(f_send,f_recv)
```

Do MPI operations on CPU

```
call MPI_Send(f_send, N , .....)
```

```
call MPI_Recv(f_recv, N , .....)
```

```
!$omp target update to(f_send,f_recv)
```

Perform computation on GPU

```
!$omp target exit data map(from: f_recv)
```

Hands-on Example 2: Traditional MPI with OpenACC/OpenMP

Time: 15 min

Purpose: To measure the time it takes to transfer data between 2 GPUs during MPI communication

Task: Review the codes & Slurm scripts and run the following:

- For MPI-OpenACC

```
$ cd example_2/mpiacc
```

- To compile and execute the code

Load the LUMI software stack

```
$ module load LUMI/24.03 partition/G
```

```
$ module load cpeCray
```

Compile: \$./compile.sh

Submit a job: \$ sbatch script.slurm

- For MPI-OpenMP

```
$ cd example_2/mpiomp
```

View the output file: \$ vi staging_mpiacc-xxxxxx.out

Output from running the code `example_2/mpicc/mpicc_staging.f90`

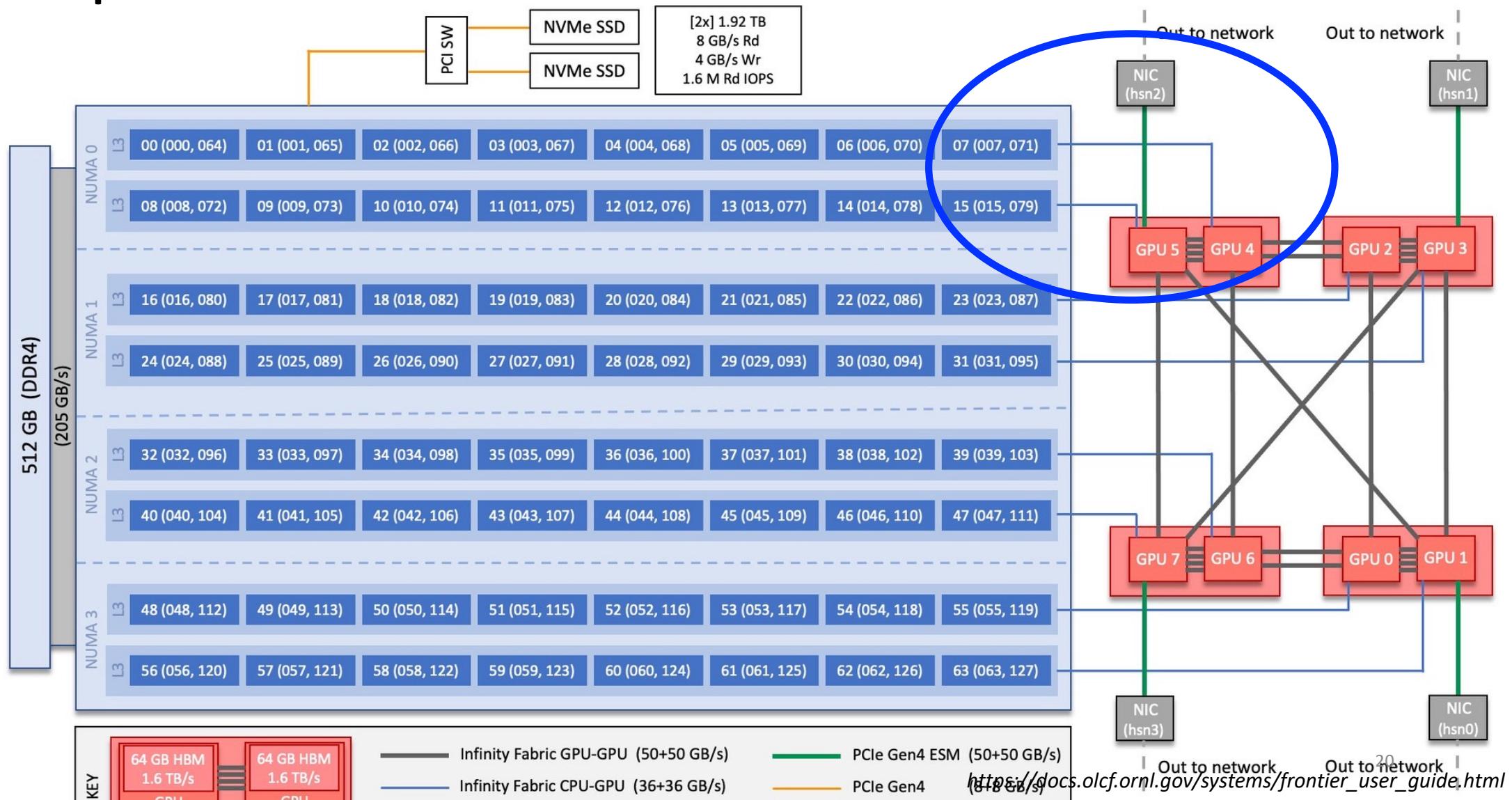
Measuring bandwidth: 2 MPI processes and 2 GPUs, single Node

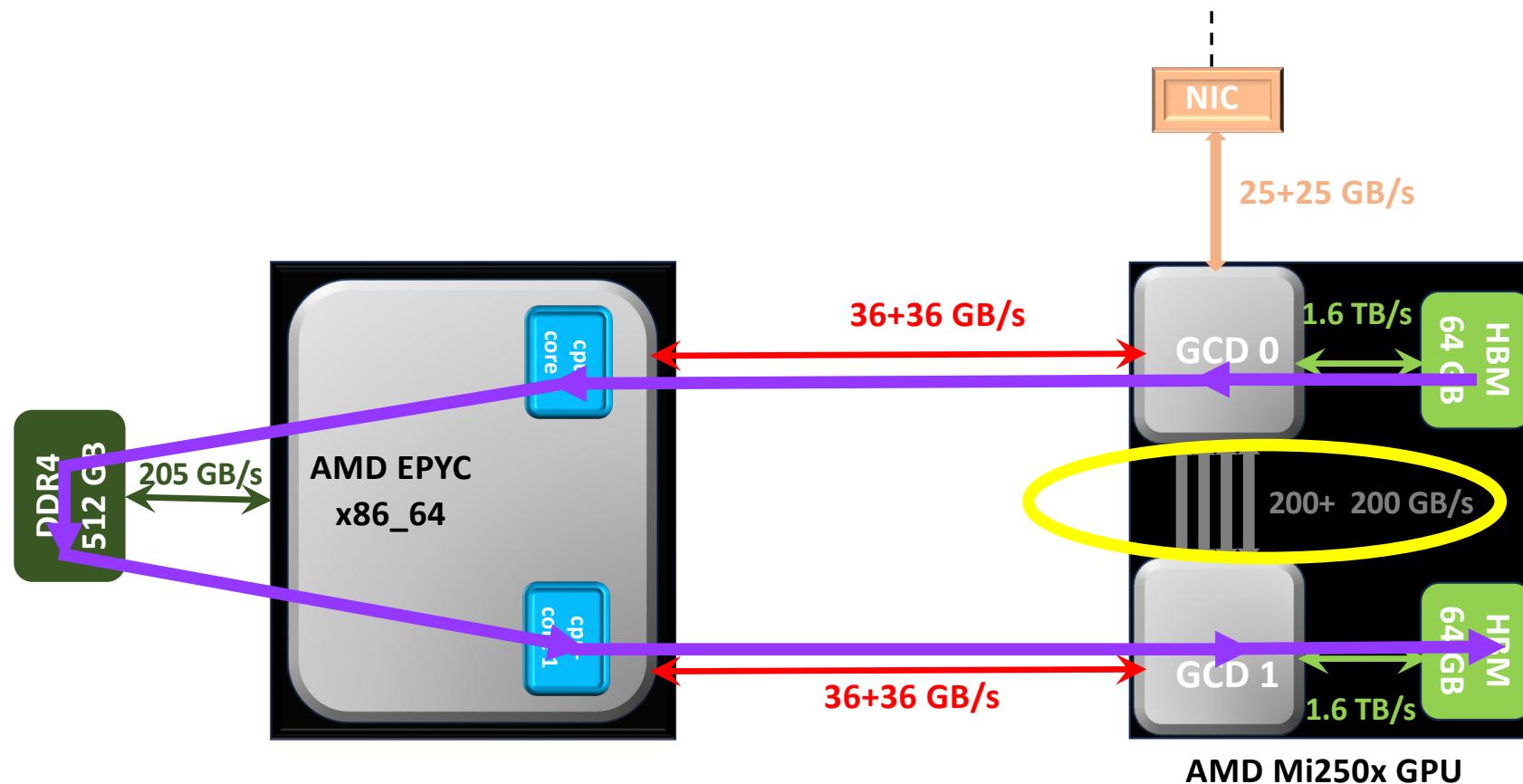
--Time (s)	0.00002	Data size (B)	128 Bandwidth (GBps)	0.01187
--Time (s)	0.00002	Data size (B)	256 Bandwidth (GBps)	0.02398
--Time (s)	0.00003	Data size (B)	512 Bandwidth (GBps)	0.04033
--Time (s)	0.00003	Data size (B)	1024 Bandwidth (GBps)	0.08017
--Time (s)	0.00002	Data size (B)	2048 Bandwidth (GBps)	0.16789
--Time (s)	0.00003	Data size (B)	4096 Bandwidth (GBps)	0.27093
--Time (s)	0.00003	Data size (B)	8192 Bandwidth (GBps)	0.57978
--Time (s)	0.00003	Data size (B)	16384 Bandwidth (GBps)	1.05693
--Time (s)	0.00004	Data size (B)	32768 Bandwidth (GBps)	1.56283
--Time (s)	0.00006	Data size (B)	65536 Bandwidth (GBps)	2.12329
--Time (s)	0.00010	Data size (B)	131072 Bandwidth (GBps)	2.51498
--Time (s)	0.00019	Data size (B)	262144 Bandwidth (GBps)	2.78053
--Time (s)	0.00036	Data size (B)	524288 Bandwidth (GBps)	2.93066
--Time (s)	0.00066	Data size (B)	1048576 Bandwidth (GBps)	3.15746
--Time (s)	0.00112	Data size (B)	2097152 Bandwidth (GBps)	3.73963
--Time (s)	0.00203	Data size (B)	4194304 Bandwidth (GBps)	4.12282
--Time (s)	0.00438	Data size (B)	8388608 Bandwidth (GBps)	3.82907
--Time (s)	0.00733	Data size (B)	16777216 Bandwidth (GBps)	4.58024
--Time (s)	0.01343	Data size (B)	33554432 Bandwidth (GBps)	4.99637
--Time (s)	0.02581	Data size (B)	67108864 Bandwidth (GBps)	5.19988
--Time (s)	0.05028	Data size (B)	134217728 Bandwidth (GBps)	5.33868
--Time (s)	0.09886	Data size (B)	268435456 Bandwidth (GBps)	5.43059
--Time (s)	0.19354	Data size (B)	536870912 Bandwidth (GBps)	5.54795
--Time (s)	0.38534	Data size (B)	1073741824 Bandwidth (GBps)	5.57296

The speed of transferring **1 GB** of data between 2 GPUs is about **6 GB/s**
Too slow!!



Compute Node architecture



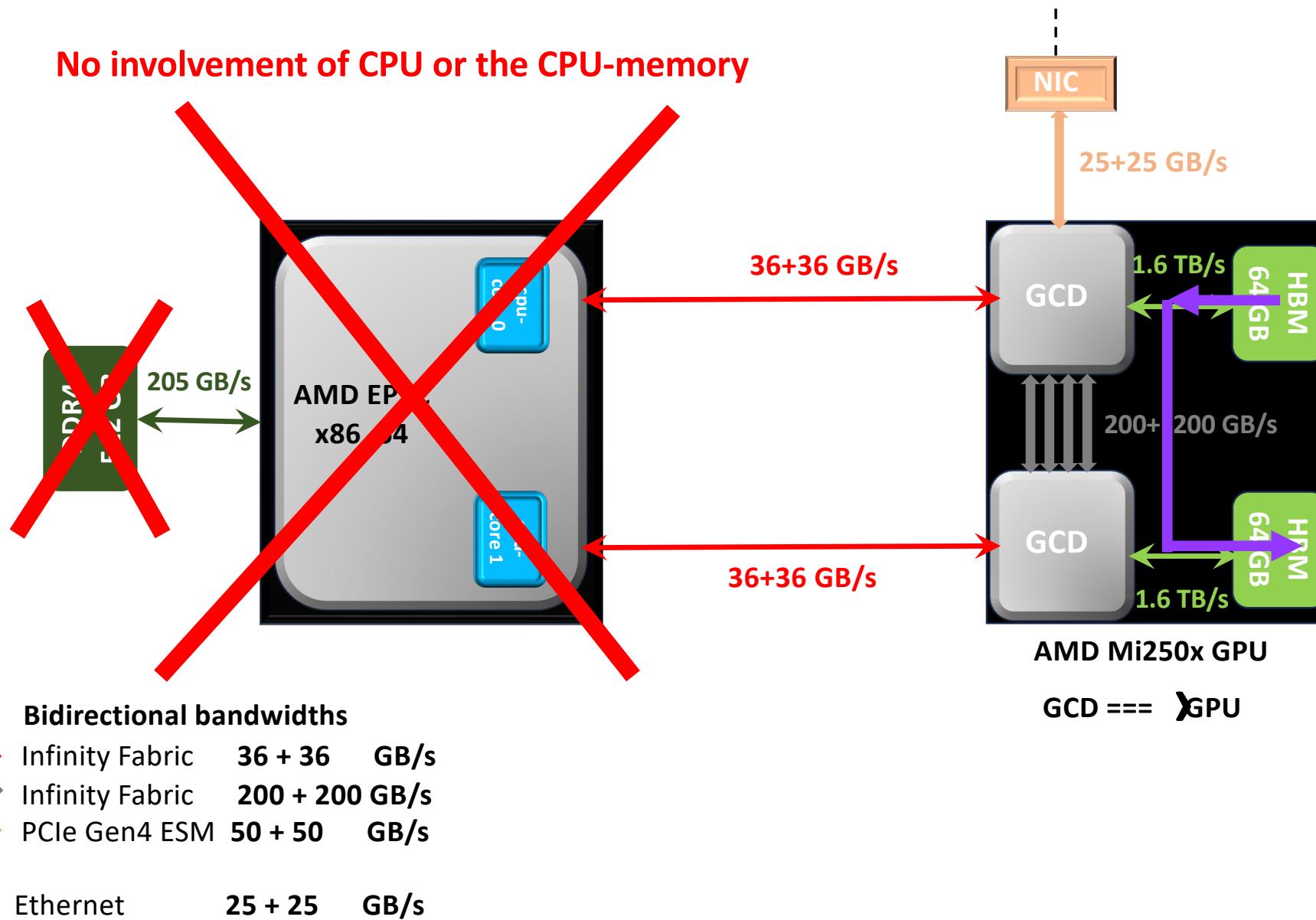


Bidirectional bandwidths

- ↔ **36 + 36 GB/s** Infinity Fabric
- ↔ **200 + 200 GB/s** Infinity Fabric
- ↔ **50 + 50 GB/s** PCIe Gen4 ESM
- ↔ **25 + 25 GB/s** Ethernet

GCD == XGPU

No involvement of CPU or the CPU-memory



Concept:
MPI with GPU awareness support



What is GPU-aware MPI ?

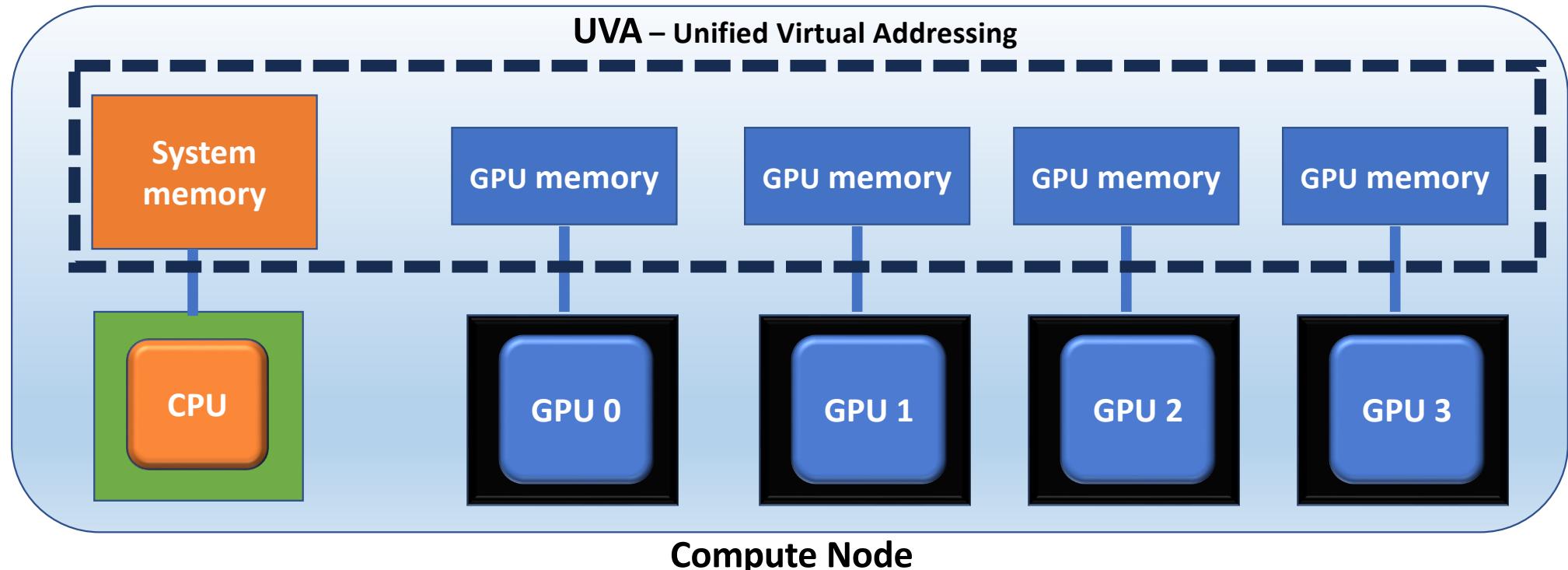
In GPU-aware MPI concept:

MPI library can **directly access the GPU memory without necessarily passing by the CPU memory.**

Device pointers are passed as arguments to an MPI routine



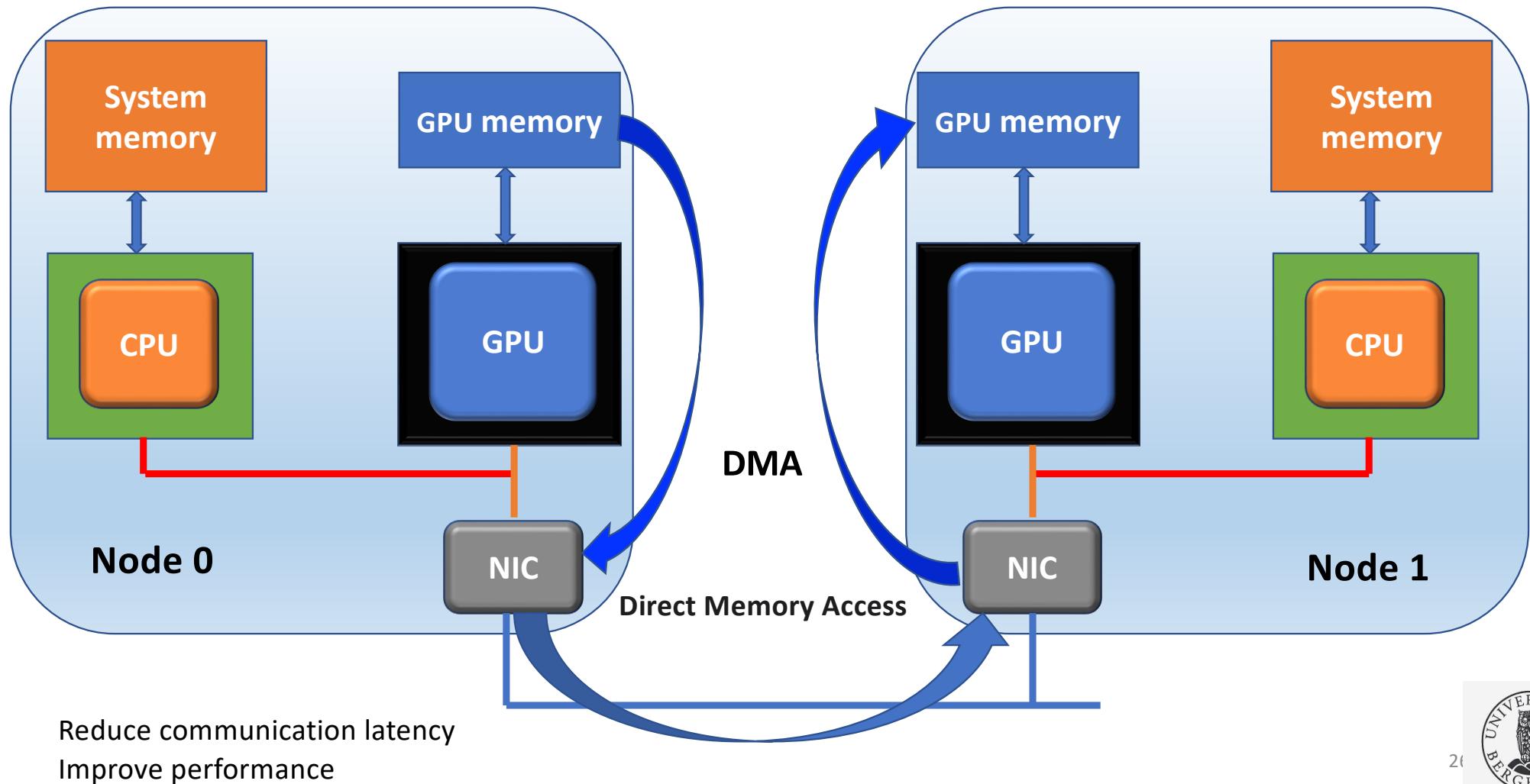
Single node: MPI with GPU awareness



- Device pointers are passed as arguments to an MPI routine
- MPI library will detect that the pointer is a device pointer
- **Unified Virtual Addressing (UVA): single virtual memory system for all memory (GPUs, CPU)**
- Direct GPU-to-GPU communication

Adapted from: <https://developer.nvidia.com/blog/introduction-cuda-aware-mpi/>

Multiple nodes: GPUDirect RDMA (GDR)

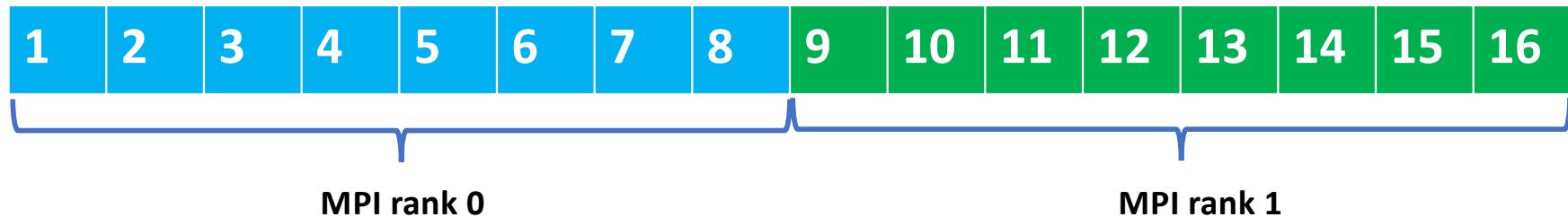


Example: GPU-aware MPI with OpenACC/OpenMP APIs

- MPI operations : `MPI_Send & MPI_Recv`
- OpenACC directive: `host_data device_ptr()`
- OpenMP directive: `use_device_ptr()`



Scenario with only 2 MPI-processes: MPI_Send & MPI_Recv



```
if(MPIrank.eq.0) then
    call MPI_Send(f_send(:, N, MPI_DOUBLE_PRECISION, MPIrank=1, tag1, ...)
    call MPI_Recv(f_recv(:, N, MPI_DOUBLE_PRECISION, MPIrank=1, tag2, ...)
endif
if(MPIrank.eq.1) then
    call MPI_Recv(f_recv(:, N, MPI_DOUBLE_PRECISION, MPIrank=0, tag1, ...)
    call MPI_Send(f_send(:, N, MPI_DOUBLE_PRECISION, MPIrank=0, tag2, ...)
endif
```

GPU-aware MPI with OpenACC: MPI_Send & MPI_Recv

```
!$acc enter data copyin(f_send,f_recv)  
Perform computation on GPU
```

Device pointers f_send & f_recv are passed to MPI_send & MPI_recv

```
!$acc host_data use_device(f_send,f_recv)
```

```
call MPI_Send(f_send, N , .....)
```

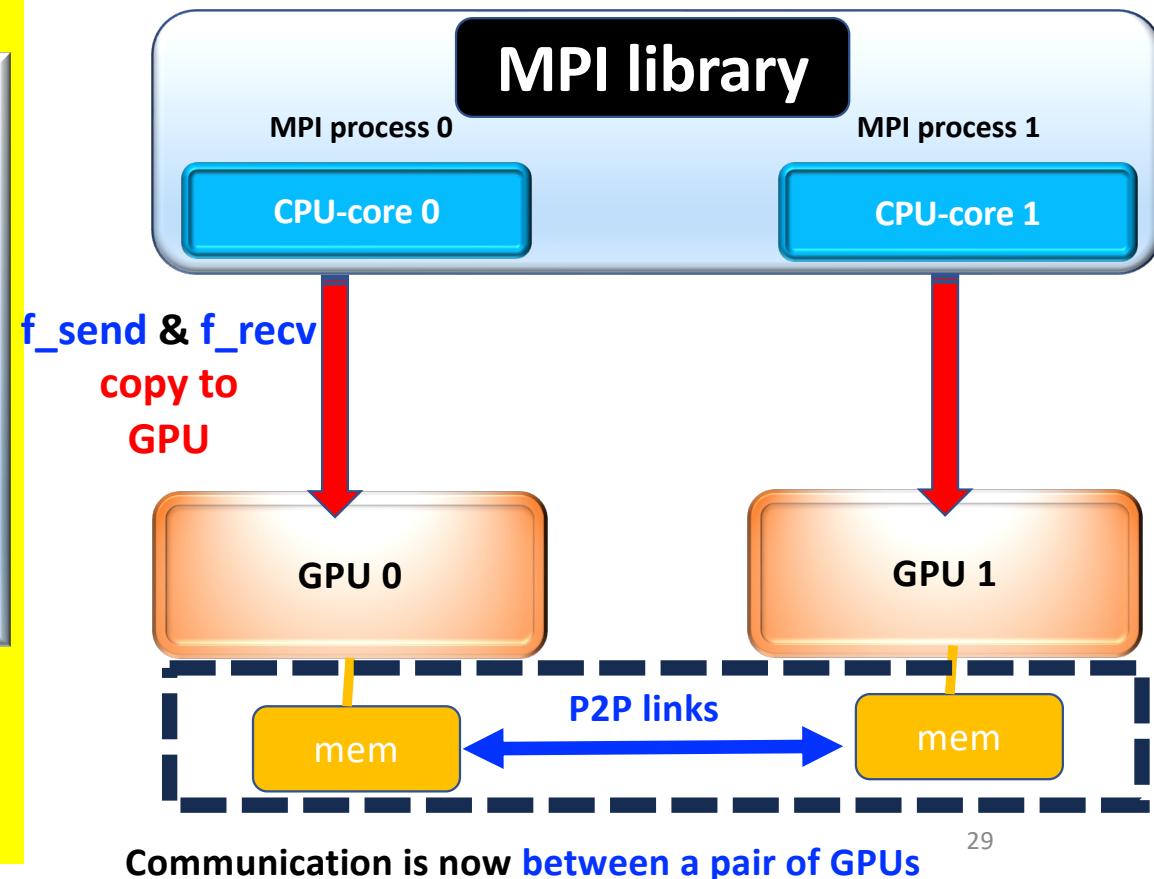
```
call MPI_Recv(f_recv, N , .....)
```

```
!$acc end host_data
```

Perform computation on GPU

```
!$acc exit data copyout(f_recv)
```

To enable GPU-aware support in MPICH lib
\$ export MPICH_GPU_SUPPORT_ENABLED=1



GPU-aware MPI with OpenACC

```
!$acc enter data copyin(f_send,f_recv)
```

Perform computation on GPU

Device pointers f_send & f_recv are passed to MPI_send & MPI_recv

```
!$acc host_data use_device(f_send,f_recv)
```

```
call MPI_Send(f_send, N , .....)
```

```
call MPI_Recv(f_recv, N, ....)
```

```
!$acc end host_data
```

Perform computation on GPU

```
!$acc exit data copyout(f_recv)
```

GPU-aware MPI with OpenMP

```
!$omp target enter data map(to: f_send,f_recv)
```

Perform computation on GPU

Device pointers f_send & f_recv are passed to MPI_send & MPI_recv

```
!$omp target data use_device_ptr(f_send,f_recv)
```

```
call MPI_Send(f_send, N , .....)
```

```
call MPI_Recv(f_recv, N, ....)
```

```
!$omp end target data
```

Perform computation on GPU

```
!$omp target exit data map(from: f_recv)
```

GPU-aware MPI with OpenACC

```
!$acc enter data copyin(f_send,f_recv)
```

Perform computation on GPU

Device pointers f_send & f_recv are passed to MPI_send & MPI_recv

```
!$acc host_data use_device(f_send,f_recv)
```

```
call MPI_Send(f_send, N , .....)
```

```
call MPI_Recv(f_recv, N, .......)
```

```
!$acc end host_data
```

Perform computation on GPU

```
!$acc exit data copyout(f_recv)
```

Traditional MPI-OpenACC

```
!$acc enter data copyin(f_send,f_recv)
```

Perform computation on GPU

```
!$acc update host(f_send,f_recv)
```

Do MPI operations on CPU

```
call MPI_Send(f_send, N , .....)
```

```
call MPI_Recv(f_recv, N , .......)
```

```
!$acc update device(f_send,f_recv)
```

Perform computation on GPU

```
!$acc exit data copyout(f_recv)
```

Hands-on Example 3: GPU-aware MPI with OpenACC/OpenMP ?

Time: 15 min

Purpose: To measure the time it takes to transfer data between 2 GPUs during MPI communication

Task: 1. Review the codes & Slurm scripts and run the following:

2. Compare CPU-based (example 2) vs. GPU-aware approaches (example 3)

- For MPI-OpenACC

```
$ cd example_3/gpuaware_mpiacc
```

- To compile and execute the code

Load the LUMI software stack

```
$ module load LUMI/24.03 partition/G
```

```
$ module load cpeCray
```

Compile: \$./compile.sh

Submit a job: \$ sbatch script.slurm

- For MPI-OpenMP

```
$ cd example_3/gpuaware_mpiomp
```

View the output file: \$ vi gpuaware_mpiacc-xxxxxx.out

Measruing bandwidth: 2 MPI processes and 2 GPUs, single Node

Traditional MPI combined with OpenACC

--Time (s)	0.00002	Data size (B)	128 Bandwidth (GBps)	0.01187
--Time (s)	0.00002	Data size (B)	256 Bandwidth (GBps)	0.02398
--Time (s)	0.00003	Data size (B)	512 Bandwidth (GBps)	0.04033
--Time (s)	0.00003	Data size (B)	1024 Bandwidth (GBps)	0.08017
--Time (s)	0.00002	Data size (B)	2048 Bandwidth (GBps)	0.16789
--Time (s)	0.00003	Data size (B)	4096 Bandwidth (GBps)	0.27093
--Time (s)	0.00003	Data size (B)	8192 Bandwidth (GBps)	0.57978
--Time (s)	0.00003	Data size (B)	16384 Bandwidth (GBps)	1.05693
--Time (s)	0.00004	Data size (B)	32768 Bandwidth (GBps)	1.56283
--Time (s)	0.00006	Data size (B)	65536 Bandwidth (GBps)	2.12329
--Time (s)	0.00010	Data size (B)	131072 Bandwidth (GBps)	2.51498
--Time (s)	0.00019	Data size (B)	262144 Bandwidth (GBps)	2.78053
--Time (s)	0.00036	Data size (B)	524288 Bandwidth (GBps)	2.93066
--Time (s)	0.00066	Data size (B)	1048576 Bandwidth (GBps)	3.15746
--Time (s)	0.00112	Data size (B)	2097152 Bandwidth (GBps)	3.73963
--Time (s)	0.00203	Data size (B)	4194304 Bandwidth (GBps)	4.12282
--Time (s)	0.00438	Data size (B)	8388608 Bandwidth (GBps)	3.82907
--Time (s)	0.00733	Data size (B)	16777216 Bandwidth (GBps)	4.58024
--Time (s)	0.01343	Data size (B)	33554432 Bandwidth (GBps)	4.99637
--Time (s)	0.02581	Data size (B)	67108864 Bandwidth (GBps)	5.19988
--Time (s)	0.05028	Data size (B)	134217728 Bandwidth (GBps)	5.33868
--Time (s)	0.09886	Data size (B)	268435456 Bandwidth (GBps)	5.43059
--Time (s)	0.19354	Data size (B)	536870912 Bandwidth (GBps)	5.54795
--Time (s)	0.38534	Data size (B)	1073741824 Bandwidth (GBps)	5.57296

GPU-aware MPI with OpenACC

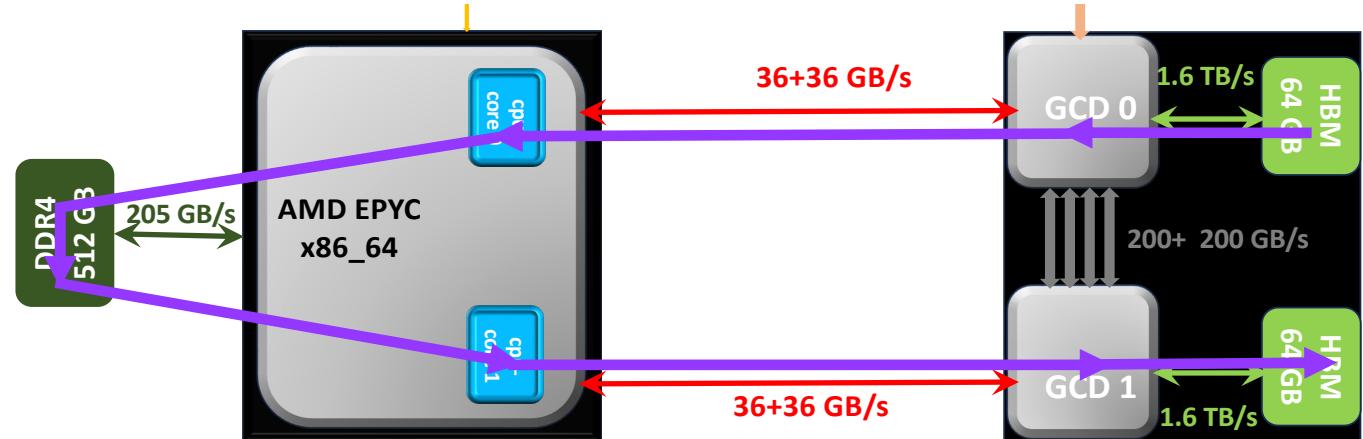
0.00001	Data size (B)	128 Bandwidth (GBps)	0.03195
0.00054	Data size (B)	256 Bandwidth (GBps)	0.00095
0.00001	Data size (B)	512 Bandwidth (GBps)	0.16521
0.00003	Data size (B)	1024 Bandwidth (GBps)	0.07636
0.00003	Data size (B)	2048 Bandwidth (GBps)	0.12525
0.00003	Data size (B)	4096 Bandwidth (GBps)	0.24760
0.00053	Data size (B)	8192 Bandwidth (GBps)	0.03119
0.00005	Data size (B)	16384 Bandwidth (GBps)	0.60414
0.00005	Data size (B)	32768 Bandwidth (GBps)	1.22875
0.00005	Data size (B)	65536 Bandwidth (GBps)	2.44771
0.00006	Data size (B)	131072 Bandwidth (GBps)	4.66266
0.00006	Data size (B)	262144 Bandwidth (GBps)	8.59824
0.00007	Data size (B)	524288 Bandwidth (GBps)	14.75897
0.00009	Data size (B)	1048576 Bandwidth (GBps)	23.20806
0.00013	Data size (B)	2097152 Bandwidth (GBps)	32.36915
0.00019	Data size (B)	4194304 Bandwidth (GBps)	45.31348
0.00030	Data size (B)	8388608 Bandwidth (GBps)	55.42490
0.00055	Data size (B)	16777216 Bandwidth (GBps)	61.15308
0.00098	Data size (B)	33554432 Bandwidth (GBps)	68.44087
0.00188	Data size (B)	67108864 Bandwidth (GBps)	71.25060
0.00383	Data size (B)	134217728 Bandwidth (GBps)	70.11301
0.00714	Data size (B)	268435456 Bandwidth (GBps)	75.14631
0.01384	Data size (B)	536870912 Bandwidth (GBps)	77.59176
0.02676	Data size (B)	1073741824 Bandwidth (GBps)	80.25726

The speed of transferring **1 GB** of data between 2 GPUs is about:
 Traditional MPI with GPU **6 GB/s** vs **80 GB/s** GPU-aware MPI

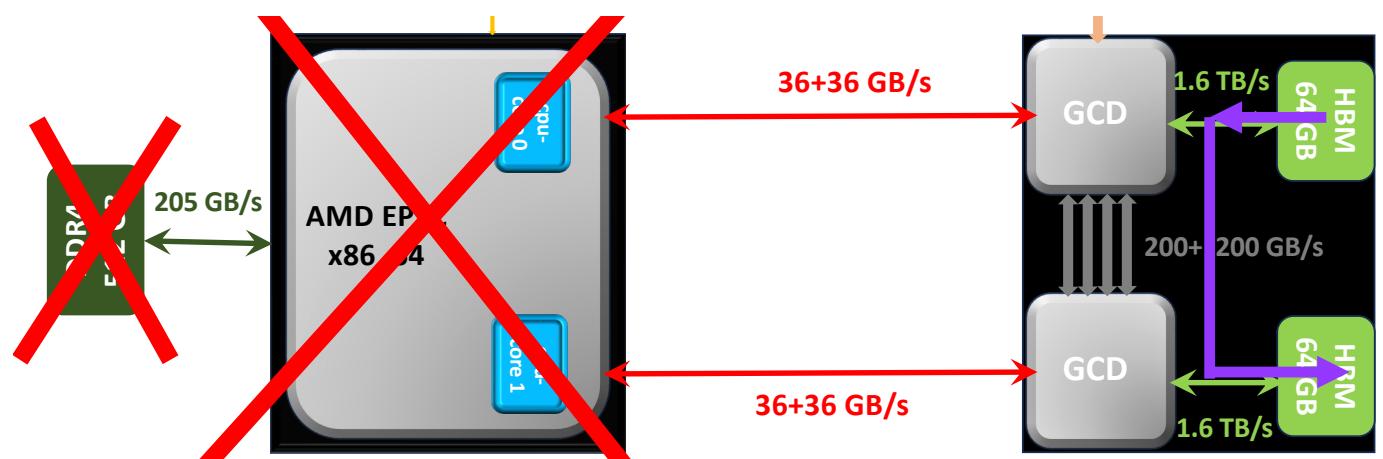


This what happens during MPI communication

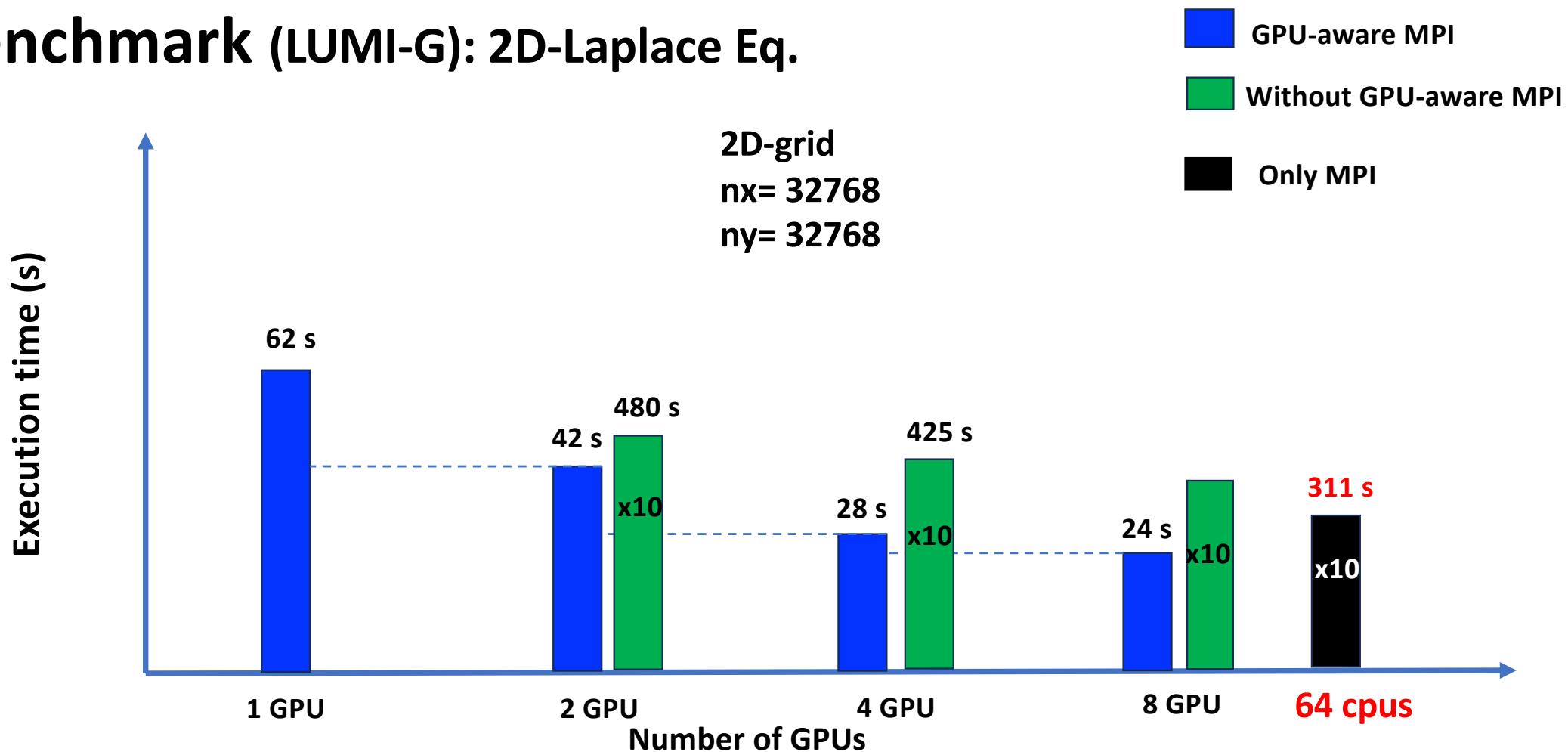
Traditional MPI
With No GPU-awareness



GPU-aware MPI



Benchmark (LUMI-G): 2D-Laplace Eq.



https://github.com/HichamAgueny/multigpu_mpi_course.git

 HichamAgueny	Update README.md	2191a55 · 3 days ago	43 Commits
 example_1	Update project Nbr	3 days ago	
 example_2	Update project Nbr	3 days ago	
 example_3	Update project Nbr	3 days ago	
 exercise	Update project Nbr	3 days ago	
 mpi	Update project Nbr	3 days ago	
 solution	Update project Nbr	3 days ago	
 LICENSE	Create LICENSE	last year	
 README.md	Update README.md	3 days ago	
 gpu_architecture_Hicham.pdf	add slides: gpu architecture	last week	
 presentation_mutipleGPUwithMPI.pdf	update the slides	last year	

Hands-on Exercise

Apply GPU-aware MPI to a 2D-Laplace equation

Binding option: CPU-GPU affinity

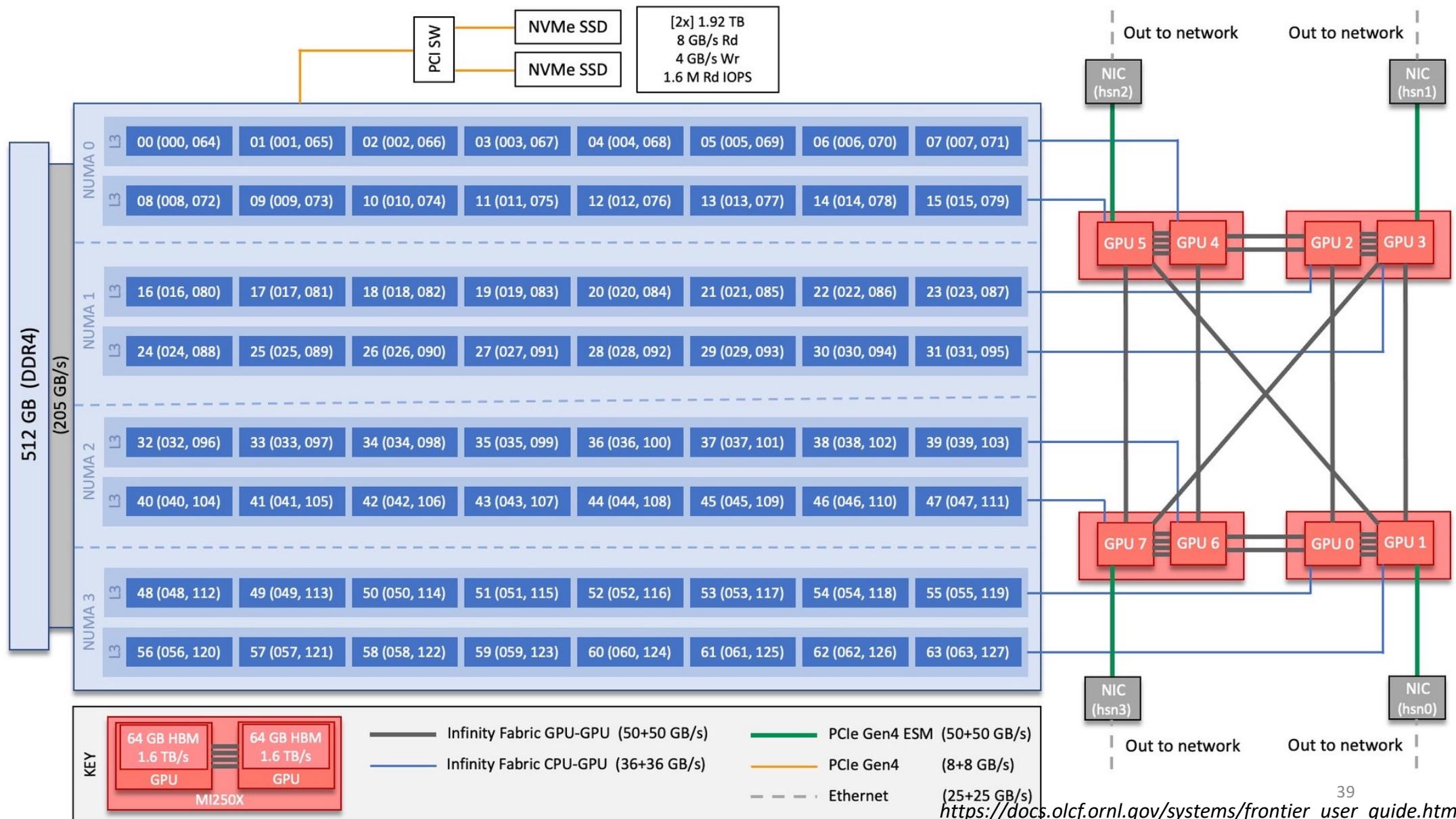
```
|hiagueny@uan01:~> salloc -A project_465000485 -t 00:05:00 -p standard-g -N 1 --gpus 8
salloc: Pending job allocation 3636016
salloc: job 3636016 queued and waiting for resources
salloc: job 3636016 has been allocated resources
salloc: Granted job allocation 3636016
|hiagueny@uan01:~> srun rocm-smi --showtoponuma
=====
===== ROCm System Management Interface =====
===== Numa Nodes =====
GPU[0]      : (Topology) Numa Node: 3
GPU[0]      : (Topology) Numa Affinity: 3
GPU[1]      : (Topology) Numa Node: 3
GPU[1]      : (Topology) Numa Affinity: 3
GPU[2]      : (Topology) Numa Node: 1
GPU[2]      : (Topology) Numa Affinity: 1
GPU[3]      : (Topology) Numa Node: 1
GPU[3]      : (Topology) Numa Affinity: 1
GPU[4]      : (Topology) Numa Node: 0
GPU[4]      : (Topology) Numa Affinity: 0
GPU[5]      : (Topology) Numa Node: 0
GPU[5]      : (Topology) Numa Affinity: 0
GPU[6]      : (Topology) Numa Node: 2
GPU[6]      : (Topology) Numa Affinity: 2
GPU[7]      : (Topology) Numa Node: 2
GPU[7]      : (Topology) Numa Affinity: 2
=====
===== End of ROCm SMI Log =====
|hiagueny@uan01:~> srun lscpu | grep NUMA
NUMA node(s):                      4
NUMA node0 CPU(s):                  0-15,64-79
NUMA node1 CPU(s):                  16-31,80-95
NUMA node2 CPU(s):                  32-47,96-111
NUMA node3 CPU(s):                  48-63,112-127
```

NUMA node 3

NUMA node 1

NUMA node 0

NUMA node 2



Binding option: CPU-GPU affinity

```
|hiagueny@uan01:~> salloc -A project_465000485 -t 00:05:00 -p standard-g -N 1 --gpus 8
salloc: Pending job allocation 3636016
salloc: job 3636016 queued and waiting for resources
salloc: job 3636016 has been allocated resources
salloc: Granted job allocation 3636016
|hiagueny@uan01:~> srun rocm-smi --showtoponuma
```

```
=====
===== ROCm System Management Interface =====
===== Numa Nodes =====
GPU[0]      : (Topology) Numa Node: 3
GPU[0]      : (Topology) Numa Affinity: 3
GPU[1]      : (Topology) Numa Node: 3
GPU[1]      : (Topology) Numa Affinity: 3
GPU[2]      : (Topology) Numa Node: 1
GPU[2]      : (Topology) Numa Affinity: 1
GPU[3]      : (Topology) Numa Node: 1
GPU[3]      : (Topology) Numa Affinity: 1
GPU[4]      : (Topology) Numa Node: 0
GPU[4]      : (Topology) Numa Affinity: 0
GPU[5]      : (Topology) Numa Node: 0
GPU[5]      : (Topology) Numa Affinity: 0
GPU[6]      : (Topology) Numa Node: 2
GPU[6]      : (Topology) Numa Affinity: 2
GPU[7]      : (Topology) Numa Node: 2
GPU[7]      : (Topology) Numa Affinity: 2
=====
===== End of ROCm SMI Log =====
```

```
|hiagueny@uan01:~> srun lscpu | grep NUMA
NUMA node(s):                      4
NUMA node0 CPU(s):                  0-15,64-79
NUMA node1 CPU(s):                  16-31,80-95
NUMA node2 CPU(s):                  32-47,96-111
NUMA node3 CPU(s):                  48-63,112-127
```

For NVIDIA: *srun nvidia-smi topo -m*

NUMA node 3

NUMA node 1

NUMA node 0

NUMA node 2

```
#!/bin/bash
```

....

....

...

```
srun --cpu-bind=map_cpu: 7,15, 23,31, 39,47,55,63
./application
```

Hands-on Exercise: Apply GPU-aware MPI in a 2D-Laplace equation

- For MPI-OpenACC

```
$ cd exercise/mpiaacc_gpuaware
```

- For MPI-OpenMP

```
$ cd exercise/mpioomp_gpuaware
```

- Follow the instructions described

laplace_gpuaware_mpiaacc.f90 for MPI-OpenACC

laplace_gpuaware_mpioomp.f90 for MPI-OpenMP

- To compile and execute the code

Load the LUMI software stack

```
$ module load LUMI/24.03 partition/G
```

```
$ module load cpeCray
```

Compile: \$./compile.sh

Submit a job: \$ sbatch script.slurm

Conclusion

Traditional MPI without GPU awareness:

- Explicit data transfer between CPU and GPU (with the involvement of CPU memory).
- GPU memory is not directly accessible from/by MPI library.



Additional overhead caused by data movement.

MPI with GPU awareness:

- MPI processes can directly access GPU memory (No involvement of CPU memory).
- Elimination of CPU-GPU data transfer.

Performance Benefits:

- Reduce data movement overhead.
- Easier to integrate directives (GPU-aware support) into existing MPI codes.



Significant improvement of performance

I stop here 😊

Get in touch:
My LinkedIn profile

Access Slides & Codes



https://github.com/HichamAgueny/multigpu_mpi_course



<https://www.linkedin.com/in/hicham-agueny-956a1368/>