

GPU-Aware MPI with ROCm™

Presenter: Bob Robey
AMD @ HLRS
May 7th, 2025

Agenda

- Introduction
- Running GPU-Aware MPI examples
 - Point-to-Point Communication Example
 - Collective Communication Example
- Measuring GPU-Aware Communication BW and Latency
 - GPU Placement Consideration on AAC cloud
 - GPU Placement Consideration on LUMI
 - Communication Options
 - Measuring intra-node/inter-node communication bandwidth
 - Measuring collective communication performance
 - Communication performance on MI300A
- ROCm Collective Communication Library (RCCL)
- Summary
- Exercises

What is MPI?

- MPI (Message-Passing Interface) is the de facto standard for communication in High Performance Computing
- Processes in an MPI program have private address space
 - MPI program can be executed on systems with distributed memory space
- MPI standard defines message passing APIs for point-to-point and collective operations

What is GPU-Aware MPI?

- Traditionally, only pointers of the host buffers could be passed to MPI calls
- GPU-Aware MPI provides this opportunity to pass GPU buffers to MPI calls
- Without GPU-Aware MPI, GPU buffers have to be staged through host memory with hipMemcpy
- Many MPI implementations including CRAY-MPICH, MVAPICH and OpenMPI support GPU-Aware Communication

What is GPUDirect RDMA?

- GPUDirect RDMA is a technology that provides the opportunity for network adapters to directly access GPU device memory and completely bypass the host
- Note that GPU-Aware MPI refers to support passing GPU buffers to MPI calls in MPI implementations while GPUDirect RDMA is a technology that enables direct access to GPU memory
- A GPU-Aware MPI may or may not use GPUDirect RDMA for communications between GPUs

GPU-Aware Point-to-Point Communication Example

```
//allocate memory
h_buf=(int*) malloc(sizeof(int)*bufsize);
hipMalloc(&d_buf,bufsize*sizeof(int));
```

Allocate memory on host

Allocate memory on device

```
//initialize
if (rank == 0)
{
  for (i=0; i<bufsize; i++)
    h_buf[i] = i;
  hipMemcpy(d_buf, h_buf, (bufsize) * sizeof(int), hipMemcpyHostToDevice);
}

if (rank == 1)
{
  for (i=0; i<bufsize; i++)
    h_buf[i] = -1;
  hipMemcpy(d_buf, h_buf, (bufsize) * sizeof(int), hipMemcpyHostToDevice);
}
```

Initialize device buffer

```
// communication
if (rank == 0) {
  MPI_Send(d_buf, bufsize, MPI_INT, 1, 123, MPI_COMM_WORLD); }

if (rank == 1) {
  MPI_Recv(d_buf, bufsize, MPI_INT, 0, 123, MPI_COMM_WORLD, &status); }
```

GPU-Aware P2P communication

```
// validate results
if (rank == 1)
{
  hipMemcpy(h_buf, d_buf, (bufsize) * sizeof(int), hipMemcpyDeviceToHost);
  for (i=0; i<bufsize; i++)
  {
    if (h_buf[i] != i)
      printf("Error: buffer[%d] = %d but is expected to be %d\n", i, h_buf[i], i);
  }
  fflush(stdout);
}
```

Validate results

```
free(h_buf);
hipFree(d_buf);
MPI_Finalize();
```

Free memory

What if we don't have GPU-Aware MPI?

- Stage GPU buffers through host memory with hipMemcpy

```
if (rank == 0) {  
    //copy send buffer from device to host  
    hipMemcpy(h_buf, d_buf, (bufsize) * sizeof(int), hipMemcpyDeviceToHost);  
  
    MPI_Send(h_buf, bufsize, MPI_INT, 1, 123, MPI_COMM_WORLD);  
}  
  
if (rank == 1) {  
    MPI_Recv(h_buf, bufsize, MPI_INT, 0, 123, MPI_COMM_WORLD, &status);  
  
    //copy receive buffer from host to device  
    hipMemcpy(d_buf, h_buf, (bufsize) * sizeof(int), hipMemcpyHostToDevice);  
}
```

GPU-Aware Collective Communication Example

```
//set device
hipSetDevice(rank%8);

//check device ID
hipGetDevice(&deviceID);
printf("rank%d running on device %d\n", rank, deviceID);
```

Set device

```
//allocate memory on host
h_buffer = (int *)malloc( count * sizeof(int) );
```

```
//allocate memory on device
hipMalloc(&d_sendbuf, count*sizeof(int));
hipMalloc(&d_recvbuf, count*sizeof(int));
```

Allocate send/rcv buffers on device

```
//initialize send and receive buffers
for (i=0; i<count; i++) h_buffer[i] = i;
hipMemcpy(d_sendbuf, h_buffer, (count) * sizeof(int), hipMemcpyHostToDevice);

hipMemset(d_recvbuf, 0, count*sizeof(int));
```

Initialize send/rcv buffers

```
//GPU-Aware Reduce
MPI_Reduce( d_sendbuf, d_recvbuf, count, MPI_INT, MPI_SUM, root, comm );
```

GPU-Aware Collective Communication

```
//validate results
if (rank == root) {
    for (i=0; i<count; i++) h_buffer[i] = 0;
    hipMemcpy(h_buffer, d_recvbuf, (count) * sizeof(int), hipMemcpyDeviceToHost);
    for (i=0; i<count; i++) {
        if (h_buffer[i] != i * size) {
            errs++;
        }
    }
    if(errs!=0) printf("errors=%d\n", errs);
}
```

Validate results

```
hipFree(d_sendbuf);
hipFree(d_recvbuf);
free( h_buffer );
```

Free memory

GPU connectivity on AAC Cloud (MI210)

rocm-smi --showtopo

```

GPU[0] : (Topology) Numa Node: 0
GPU[0] : (Topology) Numa Affinity: 0
GPU[1] : (Topology) Numa Node: 0
GPU[1] : (Topology) Numa Affinity: 0
GPU[2] : (Topology) Numa Node: 0
GPU[2] : (Topology) Numa Affinity: 0
GPU[3] : (Topology) Numa Node: 0
GPU[3] : (Topology) Numa Affinity: 0
GPU[4] : (Topology) Numa Node: 1
GPU[4] : (Topology) Numa Affinity: 1
GPU[5] : (Topology) Numa Node: 1
GPU[5] : (Topology) Numa Affinity: 1
GPU[6] : (Topology) Numa Node: 1
GPU[6] : (Topology) Numa Affinity: 1
GPU[7] : (Topology) Numa Node: 1
GPU[7] : (Topology) Numa Affinity: 1

```

```

===== Link Type between two GPUs =====

```

	GPU0	GPU1	GPU2	GPU3	GPU4	GPU5	GPU6	GPU7
GPU0	0	XGMI	XGMI	XGMI	PCIE	PCIE	PCIE	PCIE
GPU1	XGMI	0	XGMI	XGMI	PCIE	PCIE	PCIE	PCIE
GPU2	XGMI	XGMI	0	XGMI	PCIE	PCIE	PCIE	PCIE
GPU3	XGMI	XGMI	XGMI	0	PCIE	PCIE	PCIE	PCIE
GPU4	PCIE	PCIE	PCIE	PCIE	0	XGMI	XGMI	XGMI
GPU5	PCIE	PCIE	PCIE	PCIE	XGMI	0	XGMI	XGMI
GPU6	PCIE	PCIE	PCIE	PCIE	XGMI	XGMI	0	XGMI
GPU7	PCIE	PCIE	PCIE	PCIE	XGMI	XGMI	XGMI	0

```

===== Hops between two GPUs =====

```

	GPU0	GPU1	GPU2	GPU3	GPU4	GPU5	GPU6	GPU7
GPU0	0	1	1	1	3	3	3	3
GPU1	1	0	1	1	3	3	3	3
GPU2	1	1	0	1	3	3	3	3
GPU3	1	1	1	0	3	3	3	3
GPU4	3	3	3	3	0	1	1	1
GPU5	3	3	3	3	1	0	1	1
GPU6	3	3	3	3	1	1	0	1
GPU7	3	3	3	3	1	1	1	0

Demo: Intra-node GPU-to-GPU Communication Bandwidth on AAC Cloud (MI210)

Use UCX for communication

```

$export HIP_VISIBLE_DEVICES=0,1
$mpirun -n 2 -mca pml ucx ./build/libexec/osu-micro-benchmarks/mpi/pt2pt/osu_bw -m $((16*1024*1024)):$((16*1024*1024)) D D
# OSU MPI-ROCM Bandwidth Test v7.2
# Send Buffer on DEVICE (D) and Receive Buffer on DEVICE (D)
# Size    Bandwidth (MB/s)
# Datatype: MPI_CHAR.
16777216    41454.31

```

GPU 0 & 1 → 41 GB/s

```

$export HIP_VISIBLE_DEVICES=0,4
$mpirun -n 2 -mca pml ucx ./build/libexec/osu-micro-benchmarks/mpi/pt2pt/osu_bw -m $((16*1024*1024)):$((16*1024*1024)) D D
# OSU MPI-ROCM Bandwidth Test v7.2
# Send Buffer on DEVICE (D) and Receive Buffer on DEVICE (D)
# Size    Bandwidth (MB/s)
# Datatype: MPI_CHAR.
16777216    25172.16

```

GPU 0 & 4 → 25 GB/s

Device to device communication

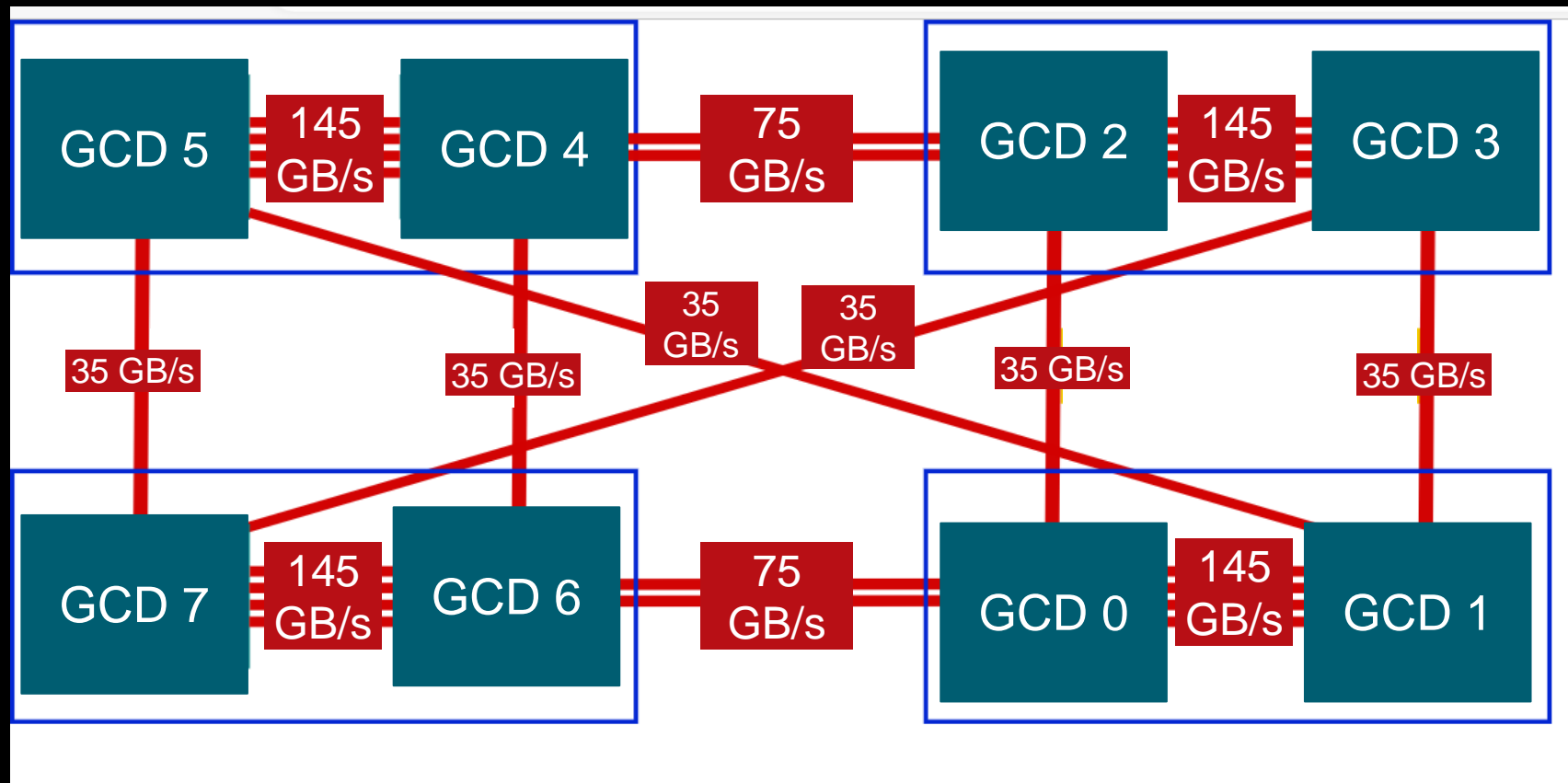
OSU Microbenchmark (OMB): Feature a series of MPI benchmarks that measure the performances of various MPI operations including point-to-point, collective, host-based and device-based communications

GPU-to-GPU Communication Options

- There are two options for GPU-to-GPU communication
 - SDMA engine
 - Provides the opportunity to overlap communication with computation
 - Each SDMA engine can provide maximum communication BW of 49 GB/s between GCDs
 - blit kernels
 - Launch kernel to handle communication
 - Pros: higher communication bandwidth
 - Cons: cannot overlap communication with computation

GPU connectivity

- Achievable GPU-to-GPU Communication Bandwidth using blit kernel
- Different number of Infinity Fabric™ links between GCDs
 - GCDs of the same GPU are connected with 4 Infinity Fabric™ links
- Different number of hops between GCDs



```

mghazimi@uan02:~/OMB/osu_benchmark> export HIP_VISIBLE_DEVICES=0,1
mghazimi@uan02:~/OMB/osu_benchmark> srun -N 1 -n 2 ./build/libexec/osu-micro-benchmarks/mpi/pt2pt/osu_bw -m $((16*1024*1024)):$((16*1024*1024)) D D
# OSU MPI-ROCM Bandwidth Test v7.0
# Send Buffer on DEVICE (D) and Receive Buffer on DEVICE (D)
# Size      Bandwidth (MB/s)
16777216    142341.39
mghazimi@uan02:~/OMB/osu_benchmark> export HIP_VISIBLE_DEVICES=0,2
mghazimi@uan02:~/OMB/osu_benchmark> srun -N 1 -n 2 ./build/libexec/osu-micro-benchmarks/mpi/pt2pt/osu_bw -m $((16*1024*1024)):$((16*1024*1024)) D D
# OSU MPI-ROCM Bandwidth Test v7.0
# Send Buffer on DEVICE (D) and Receive Buffer on DEVICE (D)
# Size      Bandwidth (MB/s)
16777216    38963.39
mghazimi@uan02:~/OMB/osu_benchmark> export HIP_VISIBLE_DEVICES=0,3
mghazimi@uan02:~/OMB/osu_benchmark> srun -N 1 -n 2 ./build/libexec/osu-micro-benchmarks/mpi/pt2pt/osu_bw -m $((16*1024*1024)):$((16*1024*1024)) D D
# OSU MPI-ROCM Bandwidth Test v7.0
# Send Buffer on DEVICE (D) and Receive Buffer on DEVICE (D)
# Size      Bandwidth (MB/s)
16777216    36903.69
mghazimi@uan02:~/OMB/osu_benchmark> export HIP_VISIBLE_DEVICES=0,4
mghazimi@uan02:~/OMB/osu_benchmark> srun -N 1 -n 2 ./build/libexec/osu-micro-benchmarks/mpi/pt2pt/osu_bw -m $((16*1024*1024)):$((16*1024*1024)) D D
# OSU MPI-ROCM Bandwidth Test v7.0
# Send Buffer on DEVICE (D) and Receive Buffer on DEVICE (D)
# Size      Bandwidth (MB/s)
16777216    36908.40
mghazimi@uan02:~/OMB/osu_benchmark> export HIP_VISIBLE_DEVICES=0,5
mghazimi@uan02:~/OMB/osu_benchmark> srun -N 1 -n 2 ./build/libexec/osu-micro-benchmarks/mpi/pt2pt/osu_bw -m $((16*1024*1024)):$((16*1024*1024)) D D
# OSU MPI-ROCM Bandwidth Test v7.0
# Send Buffer on DEVICE (D) and Receive Buffer on DEVICE (D)
# Size      Bandwidth (MB/s)
16777216    34986.18
mghazimi@uan02:~/OMB/osu_benchmark> export HIP_VISIBLE_DEVICES=0,6
mghazimi@uan02:~/OMB/osu_benchmark> srun -N 1 -n 2 ./build/libexec/osu-micro-benchmarks/mpi/pt2pt/osu_bw -m $((16*1024*1024)):$((16*1024*1024)) D D
# OSU MPI-ROCM Bandwidth Test v7.0
# Send Buffer on DEVICE (D) and Receive Buffer on DEVICE (D)
# Size      Bandwidth (MB/s)
16777216    76276.50
mghazimi@uan02:~/OMB/osu_benchmark> export HIP_VISIBLE_DEVICES=0,7
mghazimi@uan02:~/OMB/osu_benchmark> srun -N 1 -n 2 ./build/libexec/osu-micro-benchmarks/mpi/pt2pt/osu_bw -m $((16*1024*1024)):$((16*1024*1024)) D D
# OSU MPI-ROCM Bandwidth Test v7.0
# Send Buffer on DEVICE (D) and Receive Buffer on DEVICE (D)
# Size      Bandwidth (MB/s)
16777216    68778.59

```

GCD 0 & 1 → 142 GB/s

Device to device communication

GCD 0 & 2 → 38 GB/s

GCD 0 & 3 → 36 GB/s

GCD 0 & 4 → 36 GB/s

GCD 0 & 5 → 34 GB/s

GCD 0 & 6 → 76 GB/s

GCD 0 & 7 → 68 GB/s

Demo: Intra-node GPU-to-GPU Communication Bandwidth on LUMI Using blit Kernels

```

$module load rocm
$module load cray-mpich/8.1.18
$export MPICH_GPU_SUPPORT_ENABLED=1
$export HSA_ENABLE_SDMA=0

```

Enable blit kernel

Demo: Intra-node GPU-to-GPU Communication Bandwidth on LUMI using SDMA

```
[Public]
mghazimi@uan02:~/OMB/osu_benchmark> export HIP_VISIBLE_DEVICES=0,1
mghazimi@uan02:~/OMB/osu_benchmark> srun --jobid=2057636 -N 1 -n 2 ./build/libexec/osu-micro-benchmarks/mpi/pt2pt/osu_bw -m $((16*1024*1024)):$((16*1024*1024)) D D
# OSU MPI-ROCM Bandwidth Test v7.0
# Send Buffer on DEVICE (D) and Receive Buffer on DEVICE (D)
# Size      Bandwidth (MB/s)
16777216    49955.50
mghazimi@uan02:~/OMB/osu_benchmark> export HIP_VISIBLE_DEVICES=0,2
mghazimi@uan02:~/OMB/osu_benchmark> srun --jobid=2057636 -N 1 -n 2 ./build/libexec/osu-micro-benchmarks/mpi/pt2pt/osu_bw -m $((16*1024*1024)):$((16*1024*1024)) D D
# OSU MPI-ROCM Bandwidth Test v7.0
# Send Buffer on DEVICE (D) and Receive Buffer on DEVICE (D)
# Size      Bandwidth (MB/s)
16777216    36377.30
mghazimi@uan02:~/OMB/osu_benchmark> export HIP_VISIBLE_DEVICES=0,3
mghazimi@uan02:~/OMB/osu_benchmark> srun --jobid=2057636 -N 1 -n 2 ./build/libexec/osu-micro-benchmarks/mpi/pt2pt/osu_bw -m $((16*1024*1024)):$((16*1024*1024)) D D
# OSU MPI-ROCM Bandwidth Test v7.0
# Send Buffer on DEVICE (D) and Receive Buffer on DEVICE (D)
# Size      Bandwidth (MB/s)
16777216    36940.74
mghazimi@uan02:~/OMB/osu_benchmark> export HIP_VISIBLE_DEVICES=0,4
mghazimi@uan02:~/OMB/osu_benchmark> srun --jobid=2057636 -N 1 -n 2 ./build/libexec/osu-micro-benchmarks/mpi/pt2pt/osu_bw -m $((16*1024*1024)):$((16*1024*1024)) D D
# OSU MPI-ROCM Bandwidth Test v7.0
# Send Buffer on DEVICE (D) and Receive Buffer on DEVICE (D)
# Size      Bandwidth (MB/s)
16777216    36955.43
mghazimi@uan02:~/OMB/osu_benchmark> export HIP_VISIBLE_DEVICES=0,5
mghazimi@uan02:~/OMB/osu_benchmark> srun --jobid=2057636 -N 1 -n 2 ./build/libexec/osu-micro-benchmarks/mpi/pt2pt/osu_bw -m $((16*1024*1024)):$((16*1024*1024)) D D
# OSU MPI-ROCM Bandwidth Test v7.0
# Send Buffer on DEVICE (D) and Receive Buffer on DEVICE (D)
# Size      Bandwidth (MB/s)
16777216    36359.46
mghazimi@uan02:~/OMB/osu_benchmark> export HIP_VISIBLE_DEVICES=0,6
mghazimi@uan02:~/OMB/osu_benchmark> srun --jobid=2057636 -N 1 -n 2 ./build/libexec/osu-micro-benchmarks/mpi/pt2pt/osu_bw -m $((16*1024*1024)):$((16*1024*1024)) D D
# OSU MPI-ROCM Bandwidth Test v7.0
# Send Buffer on DEVICE (D) and Receive Buffer on DEVICE (D)
# Size      Bandwidth (MB/s)
16777216    49971.79
mghazimi@uan02:~/OMB/osu_benchmark> export HIP_VISIBLE_DEVICES=0,7
mghazimi@uan02:~/OMB/osu_benchmark> srun --jobid=2057636 -N 1 -n 2 ./build/libexec/osu-micro-benchmarks/mpi/pt2pt/osu_bw -m $((16*1024*1024)):$((16*1024*1024)) D D
# OSU MPI-ROCM Bandwidth Test v7.0
# Send Buffer on DEVICE (D) and Receive Buffer on DEVICE (D)
# Size      Bandwidth (MB/s)
16777216    49945.63
```

GCD 0 & 1 → 49 GB/s

GCD 0 & 2 → 36 GB/s

GCD 0 & 3 → 36 GB/s

GCD 0 & 4 → 36 GB/s

GCD 0 & 5 → 36 GB/s

GCD 0 & 6 → 49 GB/s

GCD 0 & 7 → 49 GB/s

```
$module load rocm
$module load cray-mpich/8.1.18
$export
MPICH_GPU_SUPPORT_ENABLED=1
$export HSA_ENABLE_SDMA=1
```

Enable SDMA

```
** Recent update **
$export HSA_ENABLE_SDMA_GANG=1
```

Utilize all links to make ↑ SDMA transfers....but *will* impact bidirectional traffic

Summary of the Achievable Bandwidth with blit kernel vs SDMA

- Achieve up to 49 GB/s using SDMA
- Achieve up to 142 GB/s using blit kernel
- The communication bandwidth between GCDs depends on
 - SDMA vs blit kernel
 - Number of Infinity Fabric™ links between GCDs
 - Number of hops between GCDs

Achieved Bandwidth on LUMI with blit kernel (GB/s)

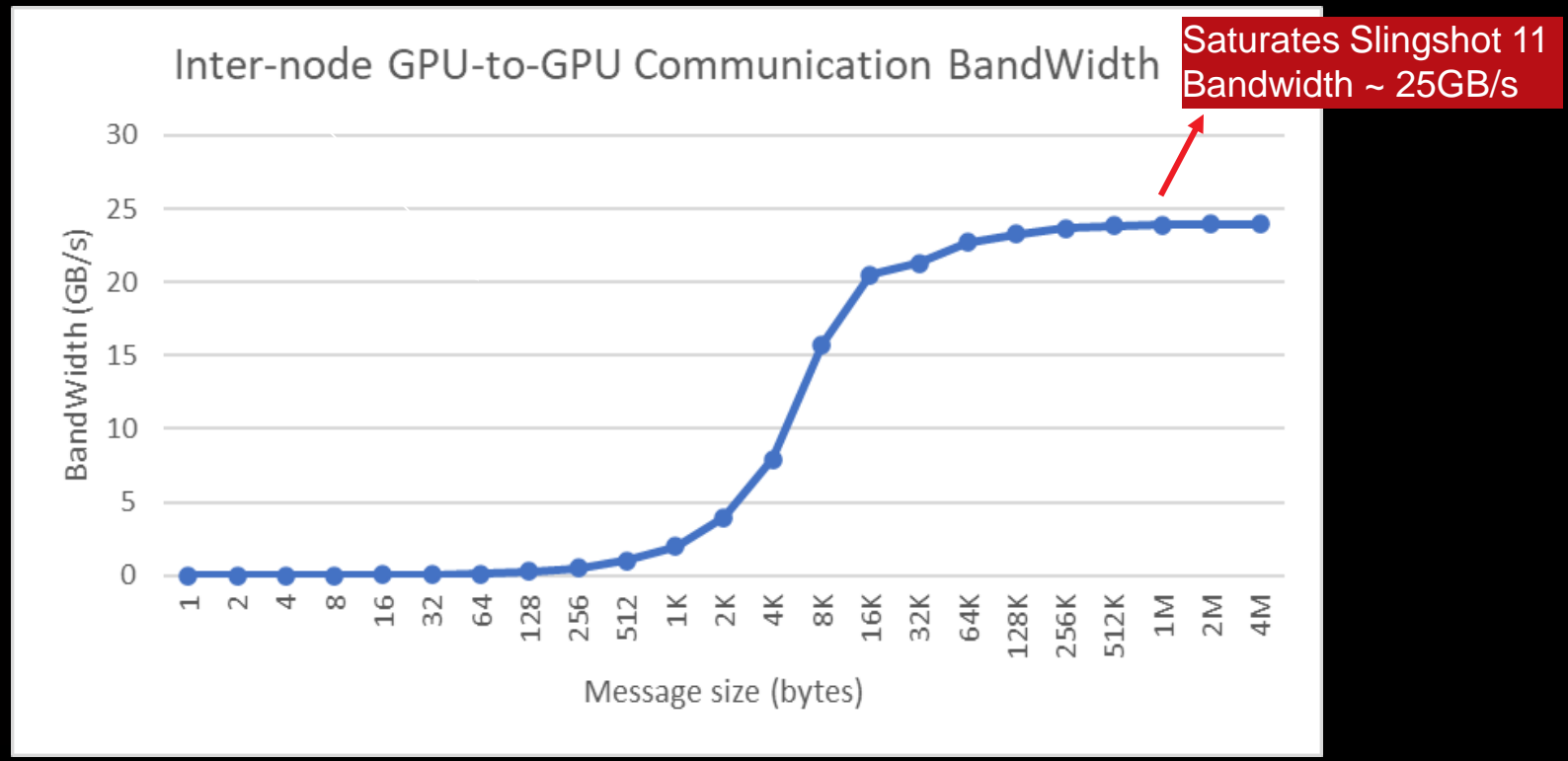
	GCD1	GCD2	GCD3	GCD4	GCD5	GCD6	GCD7
GCD0	142	38	36	36	34	76	68

Achieved Bandwidth on LUMI with SDMA (GB/s)

	GCD1	GCD2	GCD3	GCD4	GCD5	GCD6	GCD7
GCD0	49	36	36	36	34	49	49

Demo: Inter-node GPU-to-GPU Communication Bandwidth on LUMI

```
mghazimi@uan02:~/OMB/osu_benchmark> srun -N 2 -n 2 ./build/libexec/osu-micro-benchmarks/mpi/pt2pt/osu_bw D D
# OSU MPI-ROCM Bandwidth Test v7.0
# Send Buffer on DEVICE (D) and Receive Buffer on DEVICE (D)
# Size      Bandwidth (MB/s)
1           2.07
2           4.13
4           8.28
8           16.60
16          33.19
32          66.45
64          132.14
128         264.68
256         498.90
512         996.77
1024        1987.55
2048        3975.71
4096        7921.45
8192        15705.86
16384       20549.96
32768       21298.89
65536       22707.28
131072      23268.52
262144      23647.31
524288      23827.88
1048576     23903.00
2097152     23947.73
4194304     23968.83
```



May 5-8th, 2025

AMD @ HLRS



Demo: GPU-Aware Collective Communication

```
$srun -N 2 -n 8 --ntasks-per-node=4 ./build/libexec/osu-micro-benchmarks/mpi/collective/osu_allreduce -m 128 -d rocm
# OSU MPI-ROCM Allreduce Latency Test v7.0
# Size      Avg Latency(us)
4           5.23
8           5.22
16          5.23
32          5.22
64          5.26
128         5.57
```

4 ranks on node 0
4 ranks on node 1

```
srun -N 1 -n 8 --ntasks-per-node=8 ./build/libexec/osu-micro-benchmarks/mpi/collective/osu_allreduce -m 128 -d rocm
# OSU MPI-ROCM Allreduce Latency Test v7.0
# Size      Avg Latency(us)
4           1.27
8           1.24
16          1.27
32          1.27
64          1.32
128         1.39
```

8 ranks on node 0

MPI Communication Example with Unified Memory on MI250

- Unified Memory provides the opportunity to define CPU and GPU memory space as a single coherent memory
- The system manages data access between CPU and GPU without explicit memory copy functions.

```

// Allocate Unified Memory -- accessible from CPU or GPU
hipMallocManaged(&sendbuf, bufsize*sizeof(int));
hipMallocManaged(&recvbuf, bufsize*sizeof(int));

for(i=0;i<bufsize;i++) {
    sendbuf[i]=i;
    recvbuf[i]=0;
}

if(rank==0) {
    MPI_Send(sendbuf, bufsize, MPI_INT, 1, 123, MPI_COMM_WORLD);
}

if(rank==1) {
    MPI_Recv(recvbuf, bufsize, MPI_INT, 0, 123, MPI_COMM_WORLD, &status);
}

if(rank==1) {
    for(i=0;i<bufsize;i++) {
        if(recvbuf[i] != i) {
            printf("Error: buffer[%d]=%d was expected to be %d\n", i, recvbuf[i], i);
        }
    }
    fflush(stdout);
}

hipFree(sendbuf);
hipFree(recvbuf);

```

Allocate Unified Memory

Initialize send/recv buffers

Sending/Receiving Unified Memory Buffers

Validate results

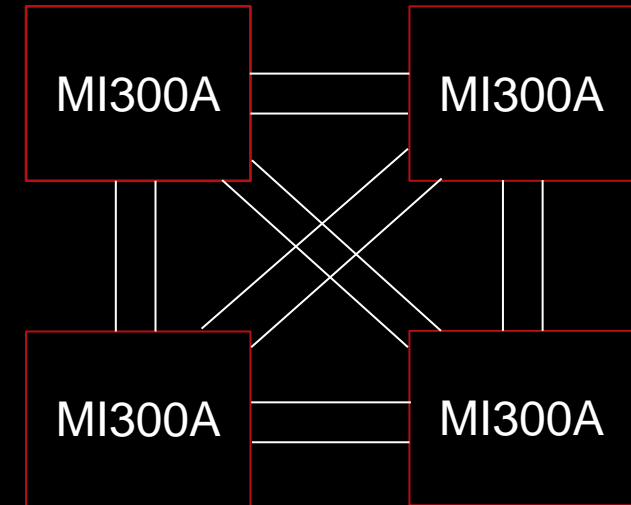
Free memory

Network Topology on AAC Nodes with MI300A

```
rocm-smi --showtopo
```

```
===== Link Type between two GPUs =====
  GPU0      GPU1      GPU2      GPU3
GPU0  0        XGMI      XGMI      XGMI
GPU1  XGMI     0        XGMI      XGMI
GPU2  XGMI     XGMI     0        XGMI
GPU3  XGMI     XGMI     XGMI     0

===== Numa Nodes =====
GPU[0]      : (Topology) Numa Node: 0
GPU[0]      : (Topology) Numa Affinity: 0
GPU[1]      : (Topology) Numa Node: 1
GPU[1]      : (Topology) Numa Affinity: 1
GPU[2]      : (Topology) Numa Node: 2
GPU[2]      : (Topology) Numa Affinity: 2
GPU[3]      : (Topology) Numa Node: 3
GPU[3]      : (Topology) Numa Affinity: 3
```



Demo: Communication Example on MI300A System

```
$mpirun -n 2 --map-by package --mca pml ucx -x HIP_VISIBLE_DEVICES=1,2 ./install/libexec/osu-micro-benchmarks/mpi/pt2pt/osu_bw -m
$((4194304*16)):$((4194304*16)) D D
```

```
# OSU MPI-ROCM Bandwidth Test v7.2
```

```
# Send Buffer on DEVICE (D) and Receive Buffer on DEVICE (D)
```

```
# Size    Bandwidth (MB/s)
```

```
# Datatype: MPI_CHAR.
```

```
67108864    90628.39
```

hipMalloc'ed buffer

```
$mpirun -n 2 --map-by package --mca pml ucx -x HIP_VISIBLE_DEVICES=1,2 ./install/libexec/osu-micro-benchmarks/mpi/pt2pt/osu_bw -m
4194304:4194304
```

```
# OSU MPI Bandwidth Test v7.2
```

```
# Size    Bandwidth (MB/s)
```

```
# Datatype: MPI_CHAR.
```

```
4194304    22986.99
```

System allocated buffer e.g., malloc

ROCm Collective Communication Library (RCCL)

- Library of standard communication routines for GPUs
- Implementing collectives e.g., all-reduce, all-gather, reduce, broadcast, reduce-scatter, gather, scatter, and all-to-all
- Initial support for direct GPU-to-GPU send and receive operations
- It is used as backend for collective communication for AI applications e.g., in PyTorch
- RCCL-test is a benchmark suite to evaluate the performance and correctness of RCCL operations

Demo: RCCL test on MI300A

git clone <https://github.com/ROCm/rccl-tests.git>

cd rccl-tests/

make MPI=1 MPI_HOME=/opt/rocmplus-6.1.0/openmpi/ HIP_HOME=/opt/rocm/

#After successful build, you should be able to see the executables in ./build directory. You can run the collectives with:

./build/all_reduce_perf -b 4M -e 128M -f 2 -g 4 → Run with 4 GPUs

Run for 4M to 128M messages

multiplication factor between sizes

```
sghazimi@d9dbb2d52d84:~/rccl/rccl-tests$ ./build/all_reduce_perf -b 4M -e 128M -f 2 -g 4
# nThread 1 nGpus 4 minBytes 4194304 maxBytes 134217728 step: 2(factor) warmup iters: 5 iters: 20 agg iters: 1 validation: 1 graph: 0
#
rccl-tests: Version develop:990f88c
# Using devices
# Rank 0 Pid 1004519 on d9dbb2d52d84 device 0 [0000:01:00.0] AMD Instinct MI300A
# Rank 1 Pid 1004519 on d9dbb2d52d84 device 1 [0001:01:00.0] AMD Instinct MI300A
# Rank 2 Pid 1004519 on d9dbb2d52d84 device 2 [0002:01:00.0] AMD Instinct MI300A
# Rank 3 Pid 1004519 on d9dbb2d52d84 device 3 [0003:01:00.0] AMD Instinct MI300A
#
#
#          size          count          type  redop  root      time      out-of-place      in-place
#          (B)      (elements)          float  sum    -1      (us)      (GB/s)  (GB/s)  #wrong      (us)  (GB/s)  (GB/s)  #wrong
#  4194304          1048576          float  sum    -1      91.43      45.88   68.81      0      85.67   48.96   73.44      0
#  8388608          2097152          float  sum    -1     128.0      65.53   98.30      0     135.4   61.97   92.95      0
# 16777216          4194304          float  sum    -1     226.2      74.16  111.25      0     246.0   68.19  102.29      0
# 33554432          8388608          float  sum    -1     409.7      81.91  122.86      0     441.4   76.01  114.01      0
# 67108864          16777216          float  sum    -1     789.2      85.04  127.56      0     819.6   81.88  122.83      0
#134217728          33554432          float  sum    -1    1567.0      85.65  128.48      0    1612.5   83.24  124.85      0
```

Summary

- Many MPI implementations including Cray-MPICH and OpenMPI support GPU-Aware communication with ROCm™
- Using OSU microbenchmark to measure communication bandwidth and latency between GPUs
- Measured communication bandwidth on LUMI (MI250) and AAC (MI210)
 - The communication bandwidth between GPUs depend on
 - Infinity Fabric™ connected vs PCIe® connected
 - Using SDMA vs blit kernel
 - Number of Infinity Fabric™ links
 - Number of hops
- Measured communication bandwidth on systems with MI300A
- Measured collective communication performance With RCCL

Exercises

Instructions to Run GPU-Aware MPI Examples on AAC

- Two MPI implementations are available in the AAC Training System – OpenMPI and MVAPICH. We'll work with OpenMPI, but MVAPICH is similar
 - Get example code
 - `git clone https://github.com/AMD/HPCTrainingExamples`
 - `cd ~/HPCTrainingExamples/MPI-examples`
 - Setup the environment
 - `module load openmpi rocm`
 - Build code – first we override the underlying compiler for the OpenMPI compiler wrapper and then compile with mpicxx
 - `export OMPI_CXX=hipcc`
 - `mpicxx -o ./pt2pt ./pt2pt.cpp`
 - `mpirun -n 2 ./pt2pt`

OSU Micro-Benchmarks (OMB)

- Retrieving OSU Benchmark code

- `mkdir OMB`
- `cd OMB`
- `wget https://mvapich.cse.ohio-state.edu/download/mvapich/osu-micro-benchmarks-7.3.tar.gz`
- `tar -xvf osu-micro-benchmarks-7.3.tar.gz`
- `cd osu-micro-benchmarks-7.3`
- `module load rocm openmpi`

- Building OMB with OpenMPI (AAC Cloud)

- `./configure --prefix=`pwd`/../build/ CC=`which mpicc` CXX=`which mpicxx` \`
- `CPPFLAGS=-D__HIP_PLATFORM_AMD__=1 --enable-rocm --with-rocm=${ROCM_PATH}`
- `make -j12`
- `make install`

Enable rocm extension

- If you get the error "cannot include hip/hip_runtime_api.h", search for `__HIP_PLATFORM_HCC__` and replace it with `__HIP_PLATFORM_AMD__` in `configure.ac` and `configure` files.

- Running benchmarks

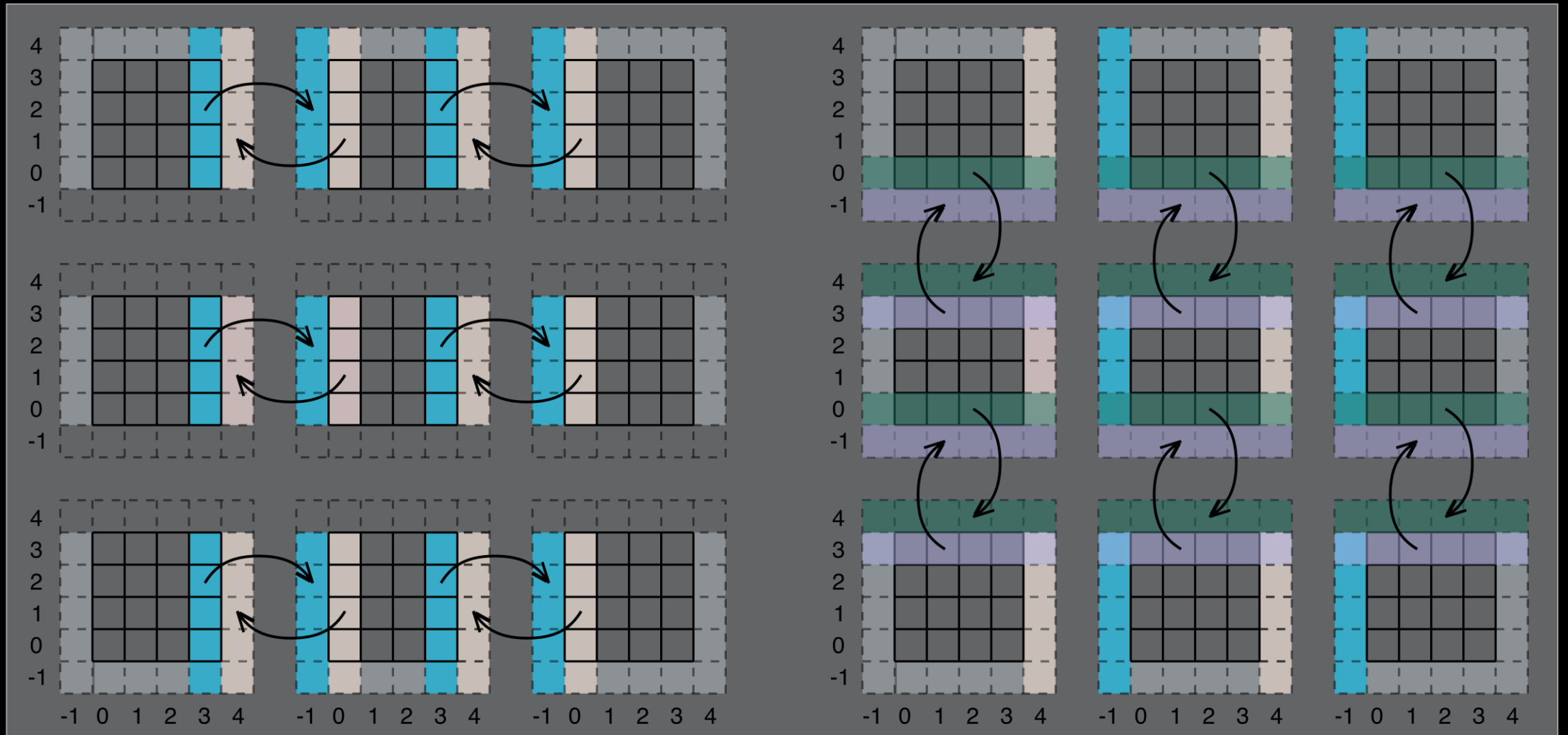
- `ls -l ../build/libexec/osu-micro-benchmarks/mpi`
- `export HIP_VISIBLE_DEVICES=0,1`
- `mpirun -N 2 -n 2 -mca pml ucx ../build/libexec/osu-micro-benchmarks/mpi/pt2pt/osu_bw -m 10240000`

Ghost cell exchange example

MPI Ghost Cell Exchange

- Many applications need to exchange and fill their ghost cells with data from adjacent processes.
- This exercise will look at the exchange for a regular cartesian grid.
- We start with the examples that accompany Chapter 8 of Parallel and High Performance Computing, Manning Publications
 - `git clone https://EssentialsOfParallelComputing/Chapter8`
 - `cd GhostExchange`
- These examples include various versions ranging from simple methods to those using MPI Datatypes and MPI Cartesian topology capabilities.
 - The size of the ghost cell halo can be specified with `-h`
 - Corner cells can be included with the `-c` flag and a synchronization is required between left/right and up/down exchanges
 - `-l` specifies the maximum iterations
- For GPU-Aware MPI, the versions using MPI Datatypes have not been optimized in most MPI implementations.
- We will use the simpler methods. We will pack the column data into to a buffer and send the buffer. For the row data, we can send it directly. If corner data is needed, synchronization is required.

Ghost Cell Exchange in two-phase scheme



Test platform

- These examples have been tested on both MI210 and MI300A systems
- ROCm version is 6.1.0
- Clang compiler is from the ROCm install
- Managed memory enabled with HSA_XNACK=1 environment variable

Original MPI Ghost Cell Exchange

- `module load rocm amdclang openmpi`
- `git clone https://github.com/AMD/HPCTrainingExamples`
- `cd HPCTrainingExamples/MPI-examples/GhostExchange/GhostExchange_ArrayAssign`
- `cd Orig`
- `mkdir build && cd build`
- `cmake ..`
- `make`
 - `mpirun -n 8 --mca pml ucx ./GhostExchange -x 4 -y 2 -i 20000 -j 20000 -h 2 -t -c -l 1000`
 - This will run 8 MPI ranks in a 4x2 processor grid (-x 4 -y 2)
 - The ghost cell halo will be 2 cells (-h 2) and the corners (-c)
 - Each processor will have a 20,000 by 20,000 cell domain
 - Problem will run for 1000 timesteps (-l 1000)

Note that the `MPI_Barrier` is commented out so there is some bleed-over from one timed region to another

- Timing is stencil 168.679181 boundary condition 0.226925 ghost cell 0.784193 total 169.843542

Adding affinity

- Though we are still on the CPU, we haven't gotten the most out of the memory performance on the node
- Let's use some process placement directives to improve performance
- The most important thing is to spread out the processes across all the NUMA regions
- `mpirun -n 8 --mca pml ucx --bind-to core --map-by ppr:4:numa --report-bindings ./GhostExchange -x 4 -y 2 -i 20000 -j 20000 -h 2 -t -c -l 1000`
- Timing is stencil 82.281613 boundary condition 0.103020 ghost cell 0.427124 total 82.895074
- Overall run time is about 51% faster. Most of the difference is in the stencil calculation.
 - `--map-by ppr:4:numa` – ppr is processes per resource. So this says to place 4 processes per numa region
 - Is this correct for your architecture? Use `lscpu` to see how many NUMA regions are present
 - Review the report of placement and bindings from the `--report-bindings` option
 - `--bind-to-core` – bind process to the core that it is placed on

Moving the calculation over to the GPU

- We'll take advantage of the MI250X Managed Memory to make the process easier
 - Similar for the MI300A Unified Address space
- CMakeLists.txt changes
- Make the minimum cmake version 3.21 and the CXX standard C++11

```
cmake_minimum_required(VERSION 3.21 FATAL_ERROR)
set (CMAKE_CXX_STANDARD 11)
```
- Add the Clang compiler flags for offloading

```
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -fopenmp --offload-arch=gfx90a -fstrict-aliasing")
set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -fopenmp --offload-arch=gfx90a -fstrict-aliasing")
```
- Add the environment variable to enable Managed Memory

```
export HSA_XNACK=1
```
- Add the required headers

```
#include <omp.h>
#pragma omp requires unified_shared_memory
```
- Now add before every computational loop in GhostExchange.cc

```
#pragma omp target teams distribute parallel for collapse(2)
```

Running the GPU program (Version 1)

```
cd Ver1
mkdir build && cd build
cmake ..
make
export HSA_XNACK=1
mpirun -n 8 --mca pml ucx --bind-to core --map-by ppr:4: numa -x
HIP_VISIBLE_DEVICES=0,1,2,3,4,5,6,7 ./GhostExchange -x 4 -y 2 -i 20000 -j 20000 -h 2 -t -c -I
1000
```

- Timing is stencil 11.504447 boundary condition 0.950199 ghost cell 23.170661 total 36.026371
- Overall run time is 56% faster
 - This speedup is due to the stencil operation running faster on the GPU than the CPU

Setting GPU placement

- While we set CPU affinity in the earlier CPU run, we also need to set the GPU that each MPI rank uses.
- This will minimize the cost of data transferred from the CPU to the GPU (and back)
- Two methods for setting the GPUs
 - Script that sets `HIP_VISIBLE_DEVICES` for each rank
 - Set within the application using `omp_set_default_device`
- Code for both methods are shown
 - We used the script method in our runs. It is more flexible and uses `rocm-smi -showtopo` to get actual placement

Setting GPU for each rank in program

- Get the GPU order with `rocm-smi --showtopo`

```
int
=nprocs/8;
int DevNum;
switch(rank/numPerNUMA)
{
  case 0:
    DevNum = 3;
    break;
  case 1:
    DevNum = 2;
    break;
  case 2:
    DevNum = 1;
    break;
  case 3:
    DevNum = 0;
    break;
  case 4:
    DevNum = 7;
    break;
  case 5:
    DevNum = 6;
    break;
  case 6:
    DevNum = 5;
    break;
  case 7:
    DevNum = 4;
    break;
}
omp_set_default_device(DevNum);
```

Set GPU for each MPI rank

Using a script to set GPU for each rank

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

```
export GPU_NUMBER_STRIDE=`expr ${OMPI_COMM_WORLD_SIZE} / ${SLURM_GPUS}`  
export GPU_NUMBER_INDEX=`expr ${OMPI_COMM_WORLD_LOCAL_RANK} / ${GPU_NUMBER_STRIDE}`  
GPU_ORDER=`rocm-smi --showtopo |grep Node | sort -k 4 |cut -f 1 | \  
    sed -e '1,1d' -e '1,$s/]//\' -e '1,$s/GPU\[//\' | tr '\n' ' ' |sed -e 's/,,$//\'`  
gpu_list=(${GPU_ORDER})  
export my_gpu=${gpu_list[${GPU_NUMBER_INDEX}]}  
#echo "MPI Rank ${OMPI_COMM_WORLD_LOCAL_RANK} will run on index ${GPU_NUMBER_INDEX} GPU ${my_gpu}"  
  
export ROCR_VISIBLE_DEVICES=$my_gpu  
  
"$@"
```

Reducing memory transfers from CPU to GPU

- The buffers for the ghost cell communication are still being transferred to the GPU
- The buffers are only being used on the GPU, so we can declare on the GPU and skip the transfer

```
double *xbuf_left_send = (double *)omp_target_alloc(bufcount*sizeof(double), omp_get_default_device());
double *xbuf_rght_send = (double *)omp_target_alloc(bufcount*sizeof(double), omp_get_default_device());
double *xbuf_rght_recv = (double *)omp_target_alloc(bufcount*sizeof(double), omp_get_default_device());
double *xbuf_left_recv = (double *)omp_target_alloc(bufcount*sizeof(double), omp_get_default_device());
    and
omp_target_free(xbuf_left_send, omp_get_default_device());
omp_target_free(xbuf_rght_send, omp_get_default_device());
omp_target_free(xbuf_rght_recv, omp_get_default_device());
omp_target_free(xbuf_left_recv, omp_get_default_device());
```

- Then run the new code

```
export HSA_XNACK=1
cd Ver3
mkdir build && cd build
cmake ..
Make
mpirun -n 8 --mca pml ucx --bind-to core --map-by ppr:4:numa -x HIP_VISIBLE_DEVICES=0,1,2,3,4,5,6,7 ./GhostExchange -x 4 -y 2 -i 20000 -j 20000 -h 2 -t -c -l 1000
```

- Timing is stencil 10.004451 boundary condition 2.117502 ghost cell 24.129182 total 36.648684
- Ghost cell time have not improved much. Memory allocation seems to be the bottleneck

Allocating MPI buffers only once

- We can allocate the buffers for the MPI communication just once instead of every cycle
- This version is shown in the Ver4 directory
- Timing is stencil 10.004451 boundary condition 2.117502 ghost cell 24.129182 total 30.49277

Results on MI300A

- orig-no affinity: Timing is stencil 95.946122 boundary condition 0.230346 ghost cell 0.340710 total 96.611253
- orig-with affinity: Timing is stencil 33.941722 boundary condition 0.138084 ghost cell 0.650967 total 34.791984
- ver1: Timing is stencil 3.070043 boundary condition 1.522021 ghost cell 4.165035 total 9.521089
- ver3: Timing is stencil 2.655060 boundary condition 0.896726 ghost cell 4.306507 total 8.628740
- ver4: Timing is stencil 2.526294 boundary condition 1.417029 ghost cell 3.715349 total 8.454250

Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED 'AS IS.' AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Third-party content is licensed to you directly by the third party that owns the content and is not licensed to you by AMD. ALL LINKED THIRD-PARTY CONTENT IS PROVIDED "AS IS" WITHOUT A WARRANTY OF ANY KIND. USE OF SUCH THIRD-PARTY CONTENT IS DONE AT YOUR SOLE DISCRETION AND UNDER NO CIRCUMSTANCES WILL AMD BE LIABLE TO YOU FOR ANY THIRD-PARTY CONTENT. YOU ASSUME ALL RISK AND ARE SOLELY RESPONSIBLE FOR ANY DAMAGES THAT MAY ARISE FROM YOUR USE OF THIRD-PARTY CONTENT.

© 2025 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, ROCm, Infinity Fabric, and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names are for informational purposes only and may be trademarks of their respective owners.

Git and the Git logo are either registered trademarks or trademarks of Software Freedom Conservancy, Inc., corporate home of the Git Project, in the United States and/or other countries

PCIe® is a registered trademark of PCI-SIG Corporation.

Backup slide(s)

How to check if an OpenMPI build is GPU-Aware?

- Is OpenMPI built with UCX?

```
$ompi_info
```

```
Configure command line: '--prefix=/global/software/openmpi/gcc/mpi'
                        '--with-ucx=/global/software/openmpi/gcc/ucx'
                        '--enable-mca-no-build=ptl-uct'
```

- Is UCX built with ROCM™?

```
$/global/software/openmpi/gcc/ucx/bin/ucx_info -v
```

```
mghazi@mun-node-0:~/mpi-codes/sndrcv$ /global/software/openmpi/gcc/ucx/bin/ucx_info -v
# Version 1.13.1
# Git branch '', revision 09f27c0
# Configured with: --disable-logging --disable-debug --disable-assertions --disable-params-check --prefix=/global/software/openmp
i/gcc/ucx --with-rocm=/opt/rocm --enable-gtest --enable-examples --with-mpi=/global/software/openmpi/gcc/mpi
mghazi@mun-node-0:~/mpi-codes/sndrcv$
```