

GPU-Aware MPI with ROCm™ and MPI Ghost Cell Exchange Example

Presenters: Giacomo Capodaglio
AMD @ CASTIEL HPC
Oct 28-30, 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
- GPU-Aware MPI Exercises
- MPI Ghost Exchange Example

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 the 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 for 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

```

D D

Device to device communication

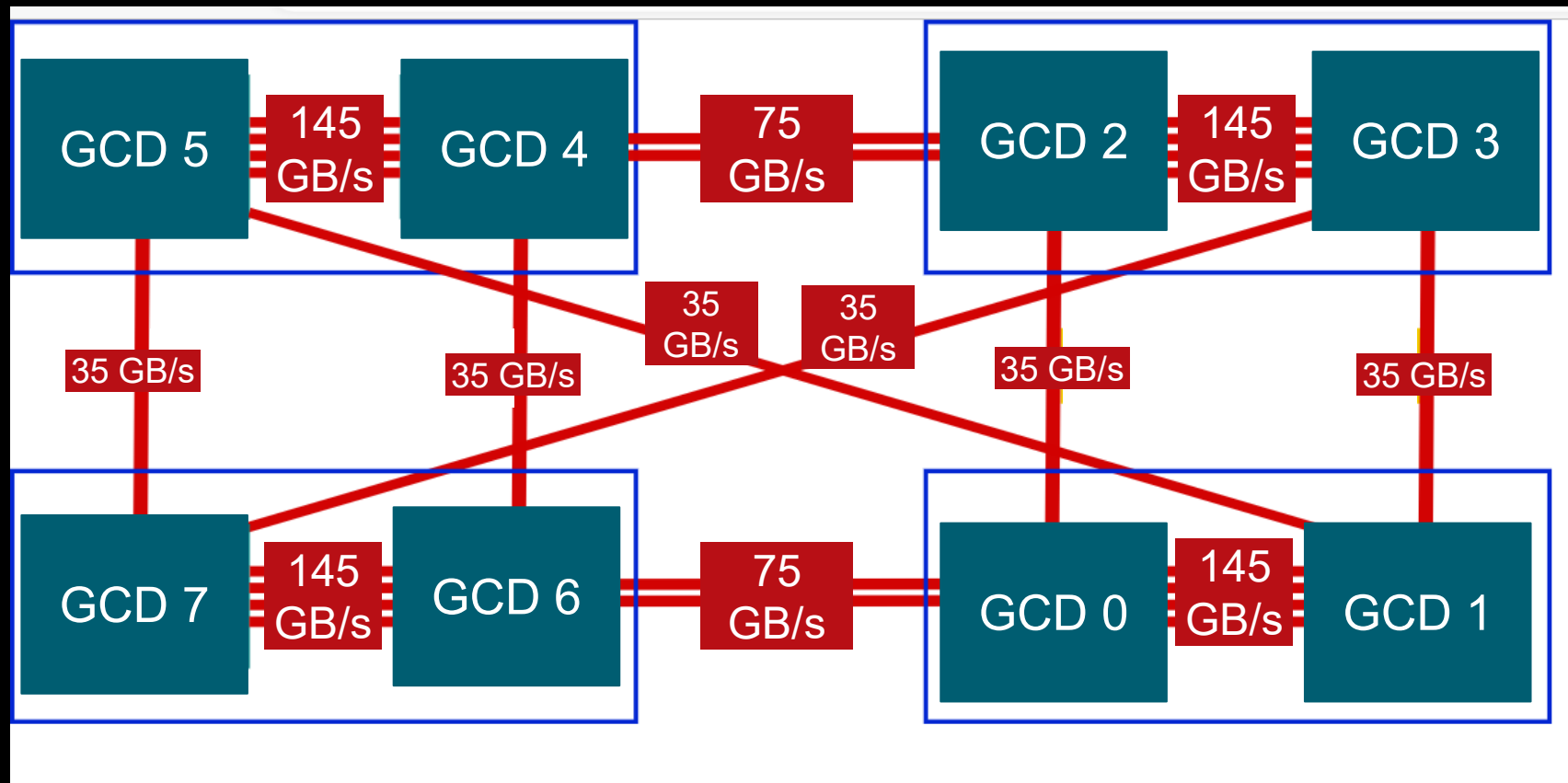
OSU Microbenchmark (OMB): Feature a series of MPI benchmarks that measure the performances of various MPI operations including point-to-point, collective, host-based and device-based communications

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)

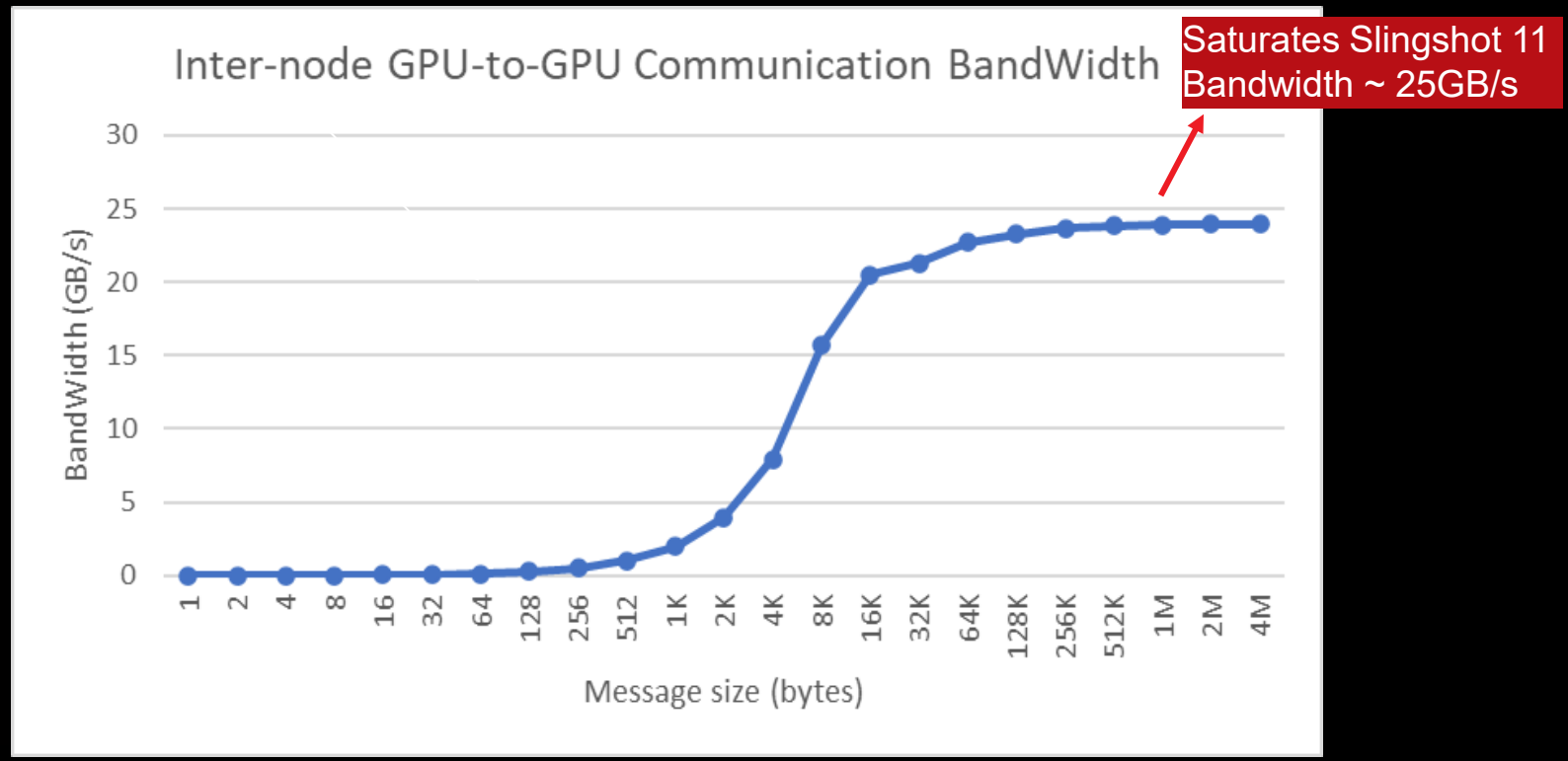
	GCD1	GCD2	GCD3	GCD4	GCD5	GCD6	GCD7
GCD0	142	38	36	36	34	76	68

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
```



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

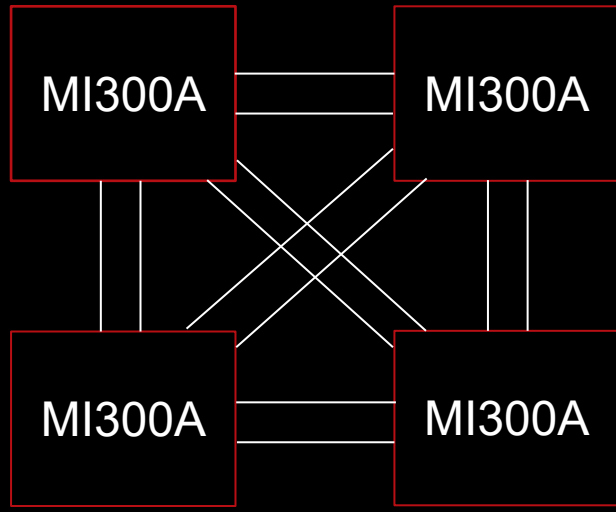
```
srun -N 1 -n 8 --ntasks-per-node=8 ./build/libexec/osu-micro-benchmarks/mpi/collective/osu_allreduce -m 128 -d rocm
# OSU MPI-ROCM Allreduce Latency Test v7.0
# Size      Avg Latency(us)
4           1.27
8           1.24
16          1.27
32          1.27
64          1.32
128         1.39
```

8 ranks on node 0

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`



MPI Example: Ghost Exchange

AMD 
together we advance_

Oct 28-30, 2025

AMD @ CASTIEL HPC

What is covered in the MPI Example: Ghost Exchange

- What is the Ghost Exchange example and how it can help scientific application developers familiarize with porting and running on GPUs
- How to compile and run the example
- Different implementations of the Ghost Exchange examples using OpenMP® or HIP
- Several programming improvements to better the overall performance of the example
- Examples of how to use affinity to further improve the performance of the Ghost Exchange example

MPI Example: Ghost Exchange – what is it?

- A simplified instance of what an actual scientific application code using MPI might look like
- The problem is discretized on a structured Cartesian grid where the solution is defined on a cell-wise fashion
- MPI is used to execute the computations in parallel: multiple processes handle partitions of the initial computational domain
- **Ghost cells** are cells assigned to a given process that are instead own by a different one: this means that a process can read the solution defined at the ghost cells but should not modify it
- The ghost cells exist because mathematical operators in discretized form need information on the neighboring cells to compute the values of physical fields on a given cell
- Ghost cells surround the cells owned by a process therefore forming a **halo** around the subdomain owned by the process. Note that boundary cells are also included in the halo in this example
- Values of the solution at the ghost cells need to be transferred from the process that owns the ghost cells to the process that is only reading their information: this is done through a **halo exchange** using MPI
- Boundary conditions are of **outflow** type, meaning that the value of the solution at the boundary cells is set equal to the neighboring interior cell.
- Boundary conditions are enforced prior to the halo exchanges

Ghost Cell Exchange in two-phase scheme

Example of the 2-step halo exchange, considering 9 processes, each owning a 4x4 subset of the mesh

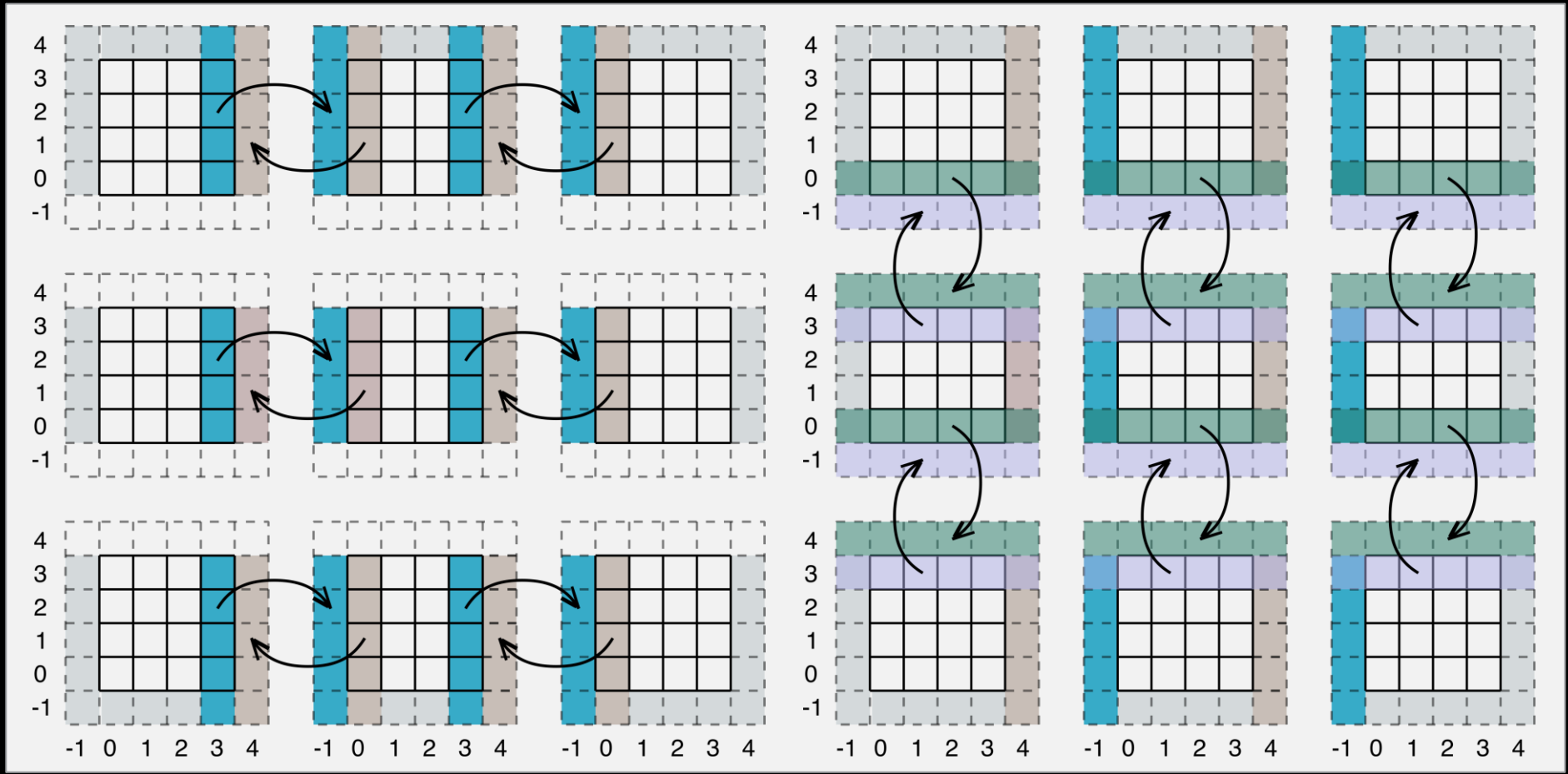


image from: *Parallel and High Performance Computing* by Robey and Zamora
Oct 28-30, 2025

AMD @ CASHEL HPC

Ghost Exchange: OpenMP[®] based versions

- OpenMP based implementations can be found at:
https://github.com/amd/HPCTrainingExamples/tree/main/MPI-examples/GhostExchange/GhostExchange_ArrayAssign
- Multiple versions available:
 - Orig: CPU only implementation
 - Ver1: offload to GPU using OpenMP and unified shared memory, needs `export HSA_XNACK=1` (variation of Orig)
 - Ver2: added roctx markers for profiling (variation of Ver1)
 - Ver3: the communication buffers are allocated on the GPU using the OpenMP API (variation of Ver2)
 - Ver4: the communication buffers are dynamically allocated on the CPU with malloc only once at the beginning of the run instead of every time step the ghost exchange call is made (variation of Ver2)
 - Ver5: the solution arrays is unrolled from a 2D array into a 1D array (variation of Ver4)
 - Ver6: uses explicit memory management with OpenMP, can do `unset HSA_XNACK` (variation of Ver5)
 - Orig8: the computation of the averaging kernel is done in an asynchronous way, all on the CPU. The cells that do not need information from the cells in the ghost halos are advanced first. Then, the MPI communication is performed, which updates the value of the cells at the ghost cells, and then the solution is advanced on those cells that need information from cells on the ghost halos (variation of Orig)

Ghost Exchange: HIP based versions

- HIP based implementations can be found at:
[HPCTrainingExamples/MPI-examples/GhostExchange/ArrayAssign_HIP at main · amd/HPCTrainingExamples](https://github.com/AMD-EPIC/HPCTrainingExamples/tree/main/AMD/HPCTrainingExamples/MPI-examples/GhostExchange/ArrayAssign_HIP)
- Multiple versions available:
 - Ver1: offload to GPU using HIP and unified shared memory, needs `export HSA_XNACK=1`` (variation of Orig from the OpenMP® dir)
 - Ver1Cuda: same as Ver1 but in CUDA instead of HIP. This version is present to test hipify tools
 - Ver1WithBug: same as Ver1 but with a bug introduced in the thread grid launch parameters to have users debug it on their own
 - Ver2: added roctx markers for profiling (variation of Ver1)
 - Ver3: the communication buffers are allocated on the GPU using `hipMalloc`: GPU aware MPI is leveraged (variation of Ver2)
 - Ver4: the communication buffers are dynamically allocated on the CPU with `malloc` only once at the beginning of the run instead of every time step the ghost exchange call is made (variation of Ver2)
 - Ver5: the solution arrays is unrolled from a 2D array into a 1D array (variation of Ver4)
 - Ver6: the solution arrays and communication buffers are allocated on the GPU, can do `unset HSA_XNACK`` (variation of Ver5)
 - Ver8: the computation advancing the solution happens on the GPU and overlaps with the MPI exchanges happening on the CPU. This feature is particularly valuable for the MI300A architecture since no copy and transfer of data has to be performed (variation of Orig8 from the OpenMP dir)

Run the Ghost Exchange CPU only version

- `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 -j`
- `mpirun -n 4 ./GhostExchange -x 2 -y 2 -i 20000 -j 20000 -h 1 -c -I 100`
 - This will run 4 MPI ranks in a 2x2 processor grid (-x 2 -y 2)
 - The ghost cell halo will be 1 cells (-h 1) and the corners (-c)
 - Each process will have a 20,000 by 20,000 cell domain (-i 20000 -j 20000)
 - Problem will run for 100 timesteps (-I 100)

Ghost Exchange: CPU vs GPU Timings Comparison

Orig (CPU)

```
Solution Advancement: 24.927985
Boundary Condition Enforcement: 0.078748
Ghost Cell Update: 0.479542
Total: 25.786622
```

```
for (int j = 0; j < jsize; j++){
  for (int i = 0; i < isize; i++){
    xnew[j][i] = ( x[j][i] + x[j][i-1] + x[j][i+1] + x[j-1][i] + x[j+1][i] )/5.0;
  }
}
```

Ver1 OpenMP® (GPU target offload)

```
Solution Advancement: 3.122815
Boundary Condition Enforcement: 0.585140
Ghost Cell Update: 0.462915
Total: 5.616494
```

```
#pragma omp target teams distribute parallel for collapse(2)
for (int j = 0; j < jsize; j++){
  for (int i = 0; i < isize; i++){
    xnew[j][i] = ( x[j][i] + x[j][i-1] + x[j][i+1] + x[j-1][i] + x[j+1][i] )/5.0;
  }
}
```

← export HSA_XNACK=1 →

Ver1 HIP

```
Solution Advancement: 8.047540
Boundary Condition Enforcement: 0.571013
Ghost Cell Update: 1.992076
Total: 11.819252
```

```
__global__ void blur (double **x, double **xnew, int jsize, int isize)
{
  int tidx = threadIdx.x + blockIdx.x * blockDim.x;
  int tidy = threadIdx.y + blockIdx.y * blockDim.y;
  if (tidy < jsize && tidx < isize) {
    xnew[tidy][tidx] = ( x[tidy][tidx] + x[tidy][tidx-1] + x[tidy][tidx+1]
                      + x[tidy-1][tidx] + x[tidy+1][tidx] )/5.0;
  }
}
```

Run on MI210 with ROCm 6.4.2 and OpenMPI 5.0.7

Ghost Exchange: Ver1 vs Ver4 Timings Comparison

Ver1 OpenMP®

export HSA_XNACK=1

Ver1 HIP

Solution Advancement: 3.122815
Boundary Condition Enforcement: 0.585140
Ghost Cell Update: 0.462915
Total: 5.616494

Solution Advancement: 8.047540
Boundary Condition Enforcement: 0.571013
Ghost Cell Update: 1.992076
Total: 11.819252

Ver4 OpenMP®

Ver4 HIP

Solution Advancement: 3.104172
Boundary Condition Enforcement: 0.445526
Ghost Cell Update: 0.489475
Total: 5.314528

Solution Advancement: 7.120858
Boundary Condition Enforcement: 0.208986
Ghost Cell Update: 2.710922
Total: 11.053205

```
roctxRangePush("BufAlloc");
xbuf_left_send = (double *)malloc(bufcount*sizeof(double));
xbuf_right_send = (double *)malloc(bufcount*sizeof(double));
xbuf_right_recv = (double *)malloc(bufcount*sizeof(double));
xbuf_left_recv = (double *)malloc(bufcount*sizeof(double));
roctxRangePop(); //BufAlloc
```

Run on MI210 with ROCm 6.4.2 and OpenMPI 5.0.7

Oct 28-30, 2025

AMD @ CASTIEL HPC

Ghost Exchange: Ver4 vs Ver6 Timings Comparison

Ver4 OpenMP®

```
Solution Advancement: 3.104172
Boundary Condition Enforcement: 0.445526
Ghost Cell Update: 0.489475
Total: 5.314528
```

Ver4 HIP

```
Solution Advancement: 7.120858
Boundary Condition Enforcement: 0.208986
Ghost Cell Update: 2.710922
Total: 11.053205
```

export HSA_XNACK=1

Ver6 OpenMP®

```
Solution Advancement: 1.210582
Boundary Condition Enforcement: 0.613608
Ghost Cell Update: 0.384211
Total: 2.227684
```

Ver6 HIP

```
Solution Advancement: 5.505561
Boundary Condition Enforcement: 0.019133
Ghost Cell Update: 0.791291
Total: 6.637961
```

unset HSA_XNACK

```
#pragma omp target enter data map(alloc: xbuf_left_send[0:bufcount], xbuf_rght_send[0:bufcount])
#pragma omp target enter data map(alloc: xbuf_rght_rcv[0:bufcount], xbuf_left_rcv[0:bufcount])
```

```
#pragma omp target enter data map(alloc: x[0:totcells], xnew[0:totcells])
```

```
roctxRangePush("BufAlloc");
HIP_CHECK(hipMalloc(&xbuf_left_send, bufcount*sizeof(double)));
HIP_CHECK(hipMalloc(&xbuf_rght_send, bufcount*sizeof(double)));
HIP_CHECK(hipMalloc(&xbuf_rght_rcv, bufcount*sizeof(double)));
HIP_CHECK(hipMalloc(&xbuf_left_rcv, bufcount*sizeof(double)));
roctxRangePop(); //BufAlloc
```

```
HIP_CHECK(hipMalloc(&x, totcells * sizeof(double)));
HIP_CHECK(hipMalloc(&xnew, totcells * sizeof(double)));
```

Run on MI210 with ROCm 6.4.2 and OpenMPI 5.0.7

Oct 28-30, 2025

AMD @ CASTIEL HPC



Ghost Exchange Ver1 HIP (no affinity settings)

```
$ mpirun -n 4 ./GhostExchange -x 2 -y 2 -i 20000 -j 20000 -h 1 -c -I 100
```

```
Solution Advancement: 8.112173
Boundary Condition Enforcement: 0.493496
Ghost Cell Update: 1.985867
Total: 11.952207
```

MPI 003	-	HWT 131	-	RT_GPU_ID 0,1,2,3,4,5,6,7	-	GPU_ID N/A
MPI 002	-	HWT 130	-	RT_GPU_ID 0,1,2,3,4,5,6,7	-	GPU_ID N/A
MPI 000	-	HWT 128	-	RT_GPU_ID 0,1,2,3,4,5,6,7	-	GPU_ID N/A
MPI 001	-	HWT 129	-	RT_GPU_ID 0,1,2,3,4,5,6,7	-	GPU_ID N/A

MPI rank

Hardware thread process ran on

GPUs available to process

GPU bound to process

Ghost Exchange Ver1 HIP (GPU affinity)

```
$ mpirun -n 4 ../../set_gpu_device.sh ./GhostExchange -x 2 -y 2 -i 20000 -j 20000 -h 1 -c -I 100
```

```
Solution Advancement: 2.004534  
Boundary Condition Enforcement: 0.020589  
Ghost Cell Update: 0.661813  
Total: 3.264538 3.66x speed-up
```

MPI 000	-	HWT 128	-	RT_GPU_ID 0	-	GPU_ID 3
MPI 003	-	HWT 131	-	RT_GPU_ID 0	-	GPU_ID 0
MPI 002	-	HWT 002	-	RT_GPU_ID 0	-	GPU_ID 1
MPI 001	-	HWT 129	-	RT_GPU_ID 0	-	GPU_ID 2

MPI rank

Hardware thread process ran on

GPUs available to process

GPU bound to process

Ghost Exchange example summary

- Porting simple MPI communication examples to the GPU can be straightforward
 - Use of complex MPI Datatypes can make it more difficult
- Some of the kernels for MPI communication and boundary cell conditions do not have a lot of work
 - On the MI300A, it might be better to do this on the CPU – see Ver8 HIP
- Additionally shown:
 - Taking advantage of Managed Memory (MI 200 series) and Unified Address (MI300A) for porting
 - Affinity and process placement
 - Avoiding memory allocations and transfers by allocating memory once on the GPU in the main routine

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.

© 2024 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.

HPE is a registered trademark of Hewlett Packard Enterprise Company and/or its affiliates.

The OpenMP name and the OpenMP logo are registered trademarks of the OpenMP Architecture Review Board