GPU-Aware MPI with ROCm™

Presenter: Mahdieh Ghazimirsaeed
EuroCC National Competence Centre Sweden
Nov 30th, 2022

AMD
together we advance_

Contributors

- Bill Brantley
- Bob Robey
- Leopold Grinberg
- Justin Chang

Agenda

- Introduction
- Running GPU-Aware MPI examples on Cloud and LUMI
  - Point-to-Point Communication Example
  - Collective Communication Example
- How to check if an OpenMPI build is GPU-Aware?
- MPI Communication Example with Unified Memory
- Measuring GPU-Aware Communication BW and Latency
  - GPU Placement Consideration on LUMI
  - Communication Options
  - Measuring intra-node/inter-node communication bandwidth
  - Measuring collective communication performance
- Conclusion

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

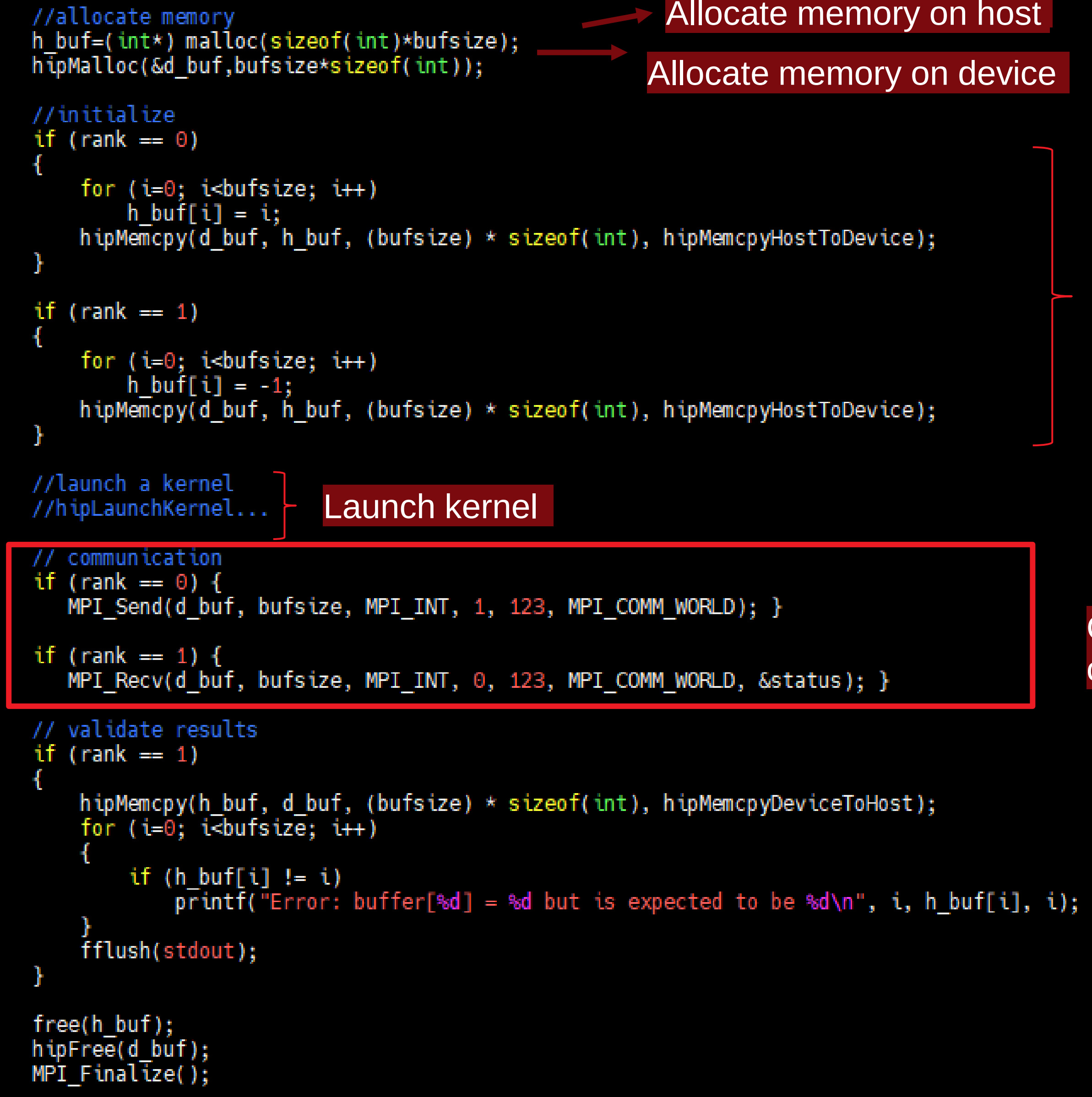
What is GPU-Aware MPI?

- Traditionally, only pointers to 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 and OpenMPI support GPU-Aware Communication

What is GPUDirect RDMA?

- GPUDirect RDMA is a technology that provides the opportunity for network adapters to directly read/write from/to 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));

//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);
}

//launch a kernel
//hipLaunchKernel...

// 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); }

// 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);
}

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

Allocate memory on host

Allocate memory on device

Initialize device buffer

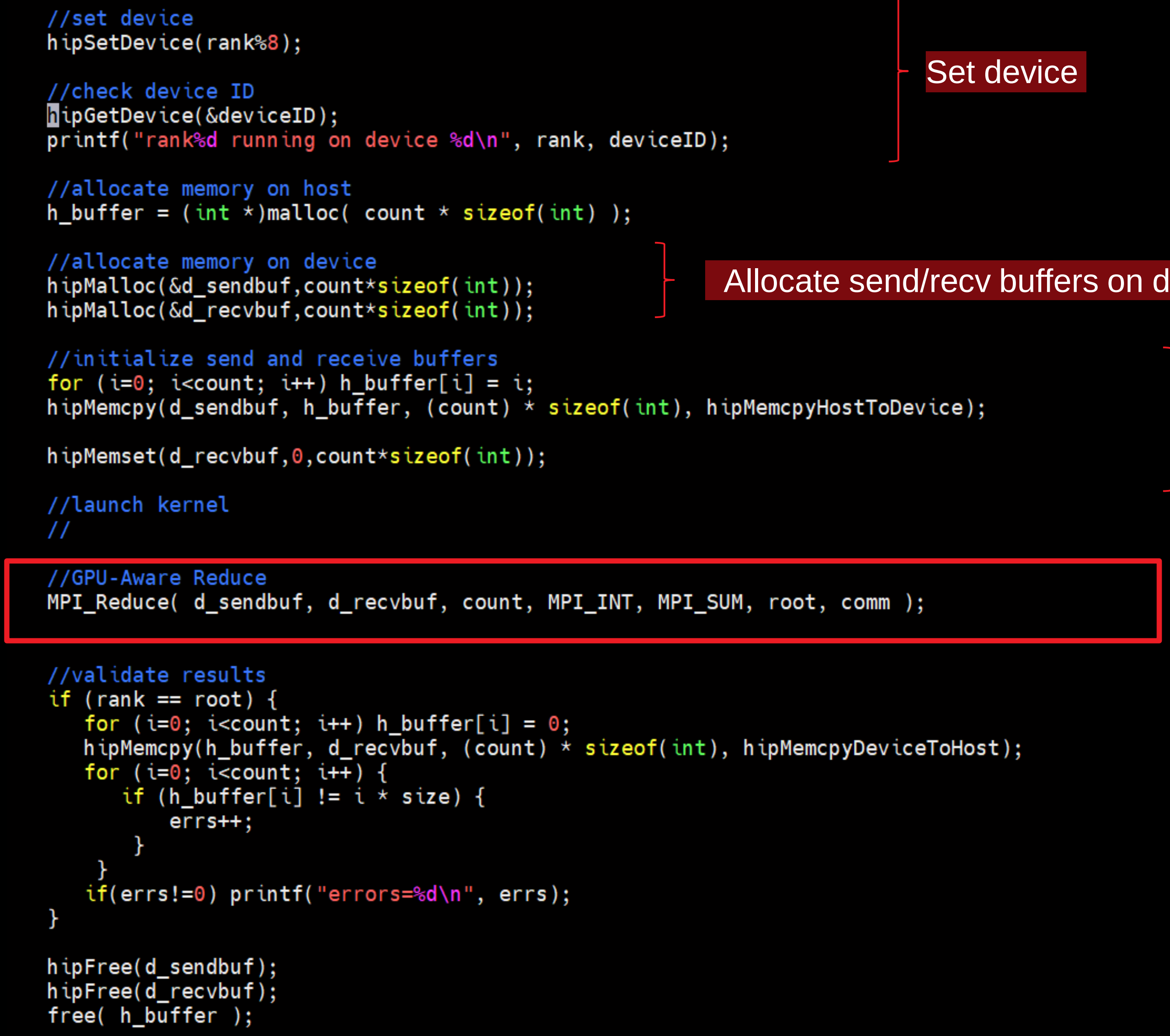
Launch kernel

GPU-Aware P2P communication

Validate results

Free memory

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);

//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));

//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));

//launch kernel
//

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

//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);
}

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

Set device

Allocate send/recv buffers on device

Initialize send/recv buffers

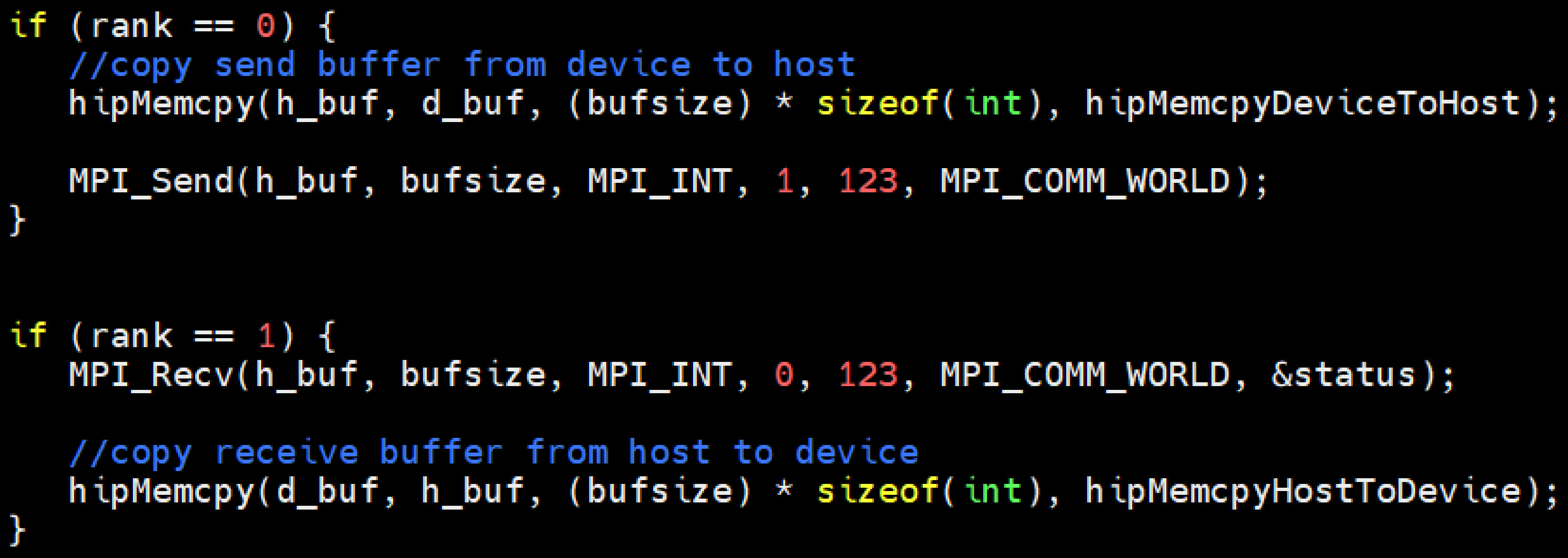
GPU-Aware Collective Communication

Validate results

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);
}
```

Instructions to Build/Run GPU-Aware MPI Examples on Cloud and LUMI

- MPI implementation available of cloud is OpenMPI
  - module load rocm/5.3.0 openmpi/4.1.4-gcc
  - export OMPI_CXX=hipcc
  - mpicxx -o ./sndrcv ./sndrcv.cpp
  - mpirun -n 2 ./sndrcv
- MPI implementation available on LUMI is Cray-MPICH
  - module load rocm cray-mpich/8.1.18
  - Two options for compiling
    - Compile with Cray compiler wrappers (cc/CC) and link rocm

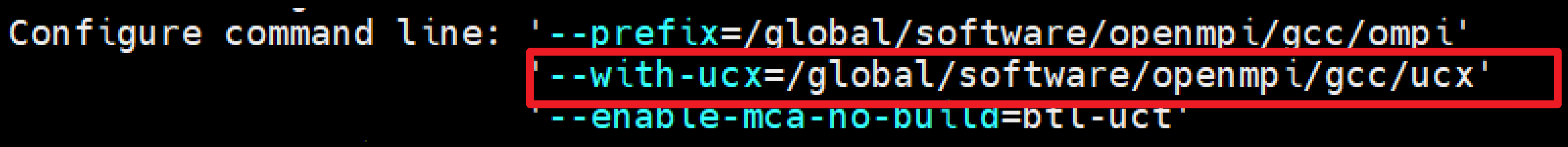
      cc -o /sndrcv ./sndrcv.cpp -I/opt/rocm/include/ -L/opt/rocm/lib -lamdhip64 -lhsa-runtime64
    - Compile with hipcc and link cray-mpich

      hipcc -o ./sndrcv ./sndrcv.cpp -I/opt/cray/pe/mpich/8.1.18/ofi/cray/10.0/include/ -L/opt/cray/pe/mpich/8.1.18/ofi/cray/10.0/lib -L/opt/cray/pe/mpich/8.1.18/gtl/lib/ -lmpi_gtl_hsa -lmpi\
  - export MPICH_GPU_SUPPORT_ENABLED=1
  - srun -n 2 ./sndrcv

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

- Is OpenMPI built with UCX?

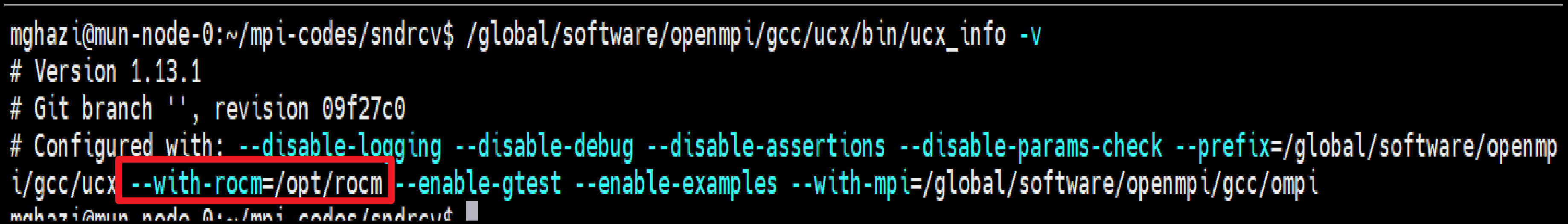
$ompi_info



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

- Is UCX built with ROCM^{TM}?

$ /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/ompi
mghazi@mun-node-0:~/mpi-codes/sndrcv$
```

OSU Micro-Benchmarks (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

- Building OMB with CRAY-MPICH (LUMI)
  - CC and CXX should refer to cray compiler path

```
./configure --prefix=~/OMB/build/ CC=/opt/cray/pe/craype/2.7.17/bin/cc CXX=/opt/cray/pe/craype/2.7.17/bin/CC --enable-rocm --with-rocm=/opt/rocm
LDFLAGS=“-L/opt/cray/pe/mpich/8.1.18/gtl/lib/ /opt/cray/pe/mpich/8.1.18/gtl/lib/libmpi_gtl_hsa.so.0”
```

Enable rocm extension

  - make –j12
  - make install
  - If you get the error “error: duplicate symbol: omb_papi_output_filename”, comment the line “char omb_papi_output_filename[OMB_PAPI_FILE_PATH_MAX_LENGTH];” in “./c/util/osu_util_papi.h”

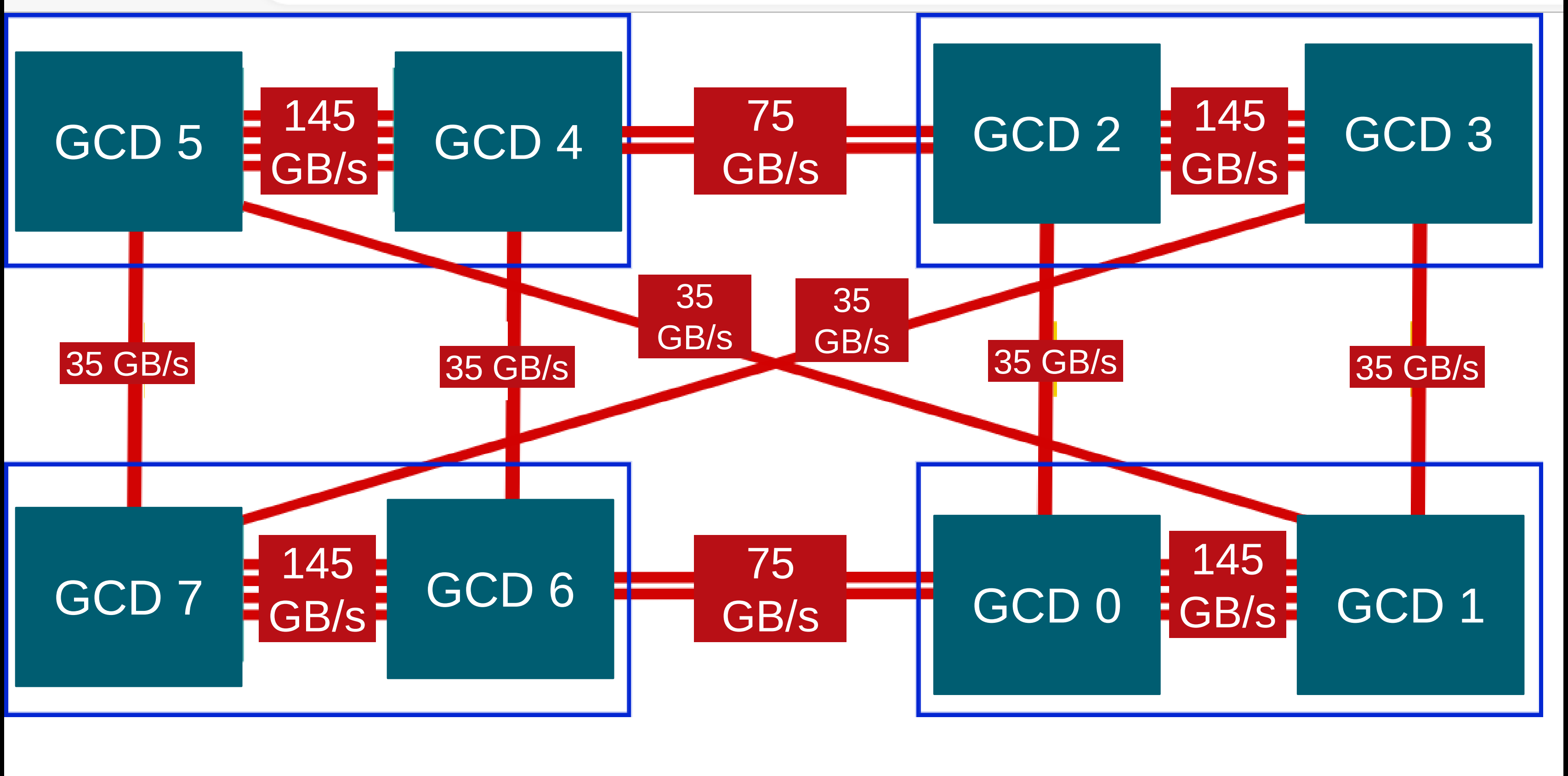
GPU-to-GPU Communication Options

- 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
- SDMA is the default in current ROCm^{TM} version available on LUMI (ROCM5.0.2)

Achievable GPU-to-GPU Communication Bandwidth Using blit

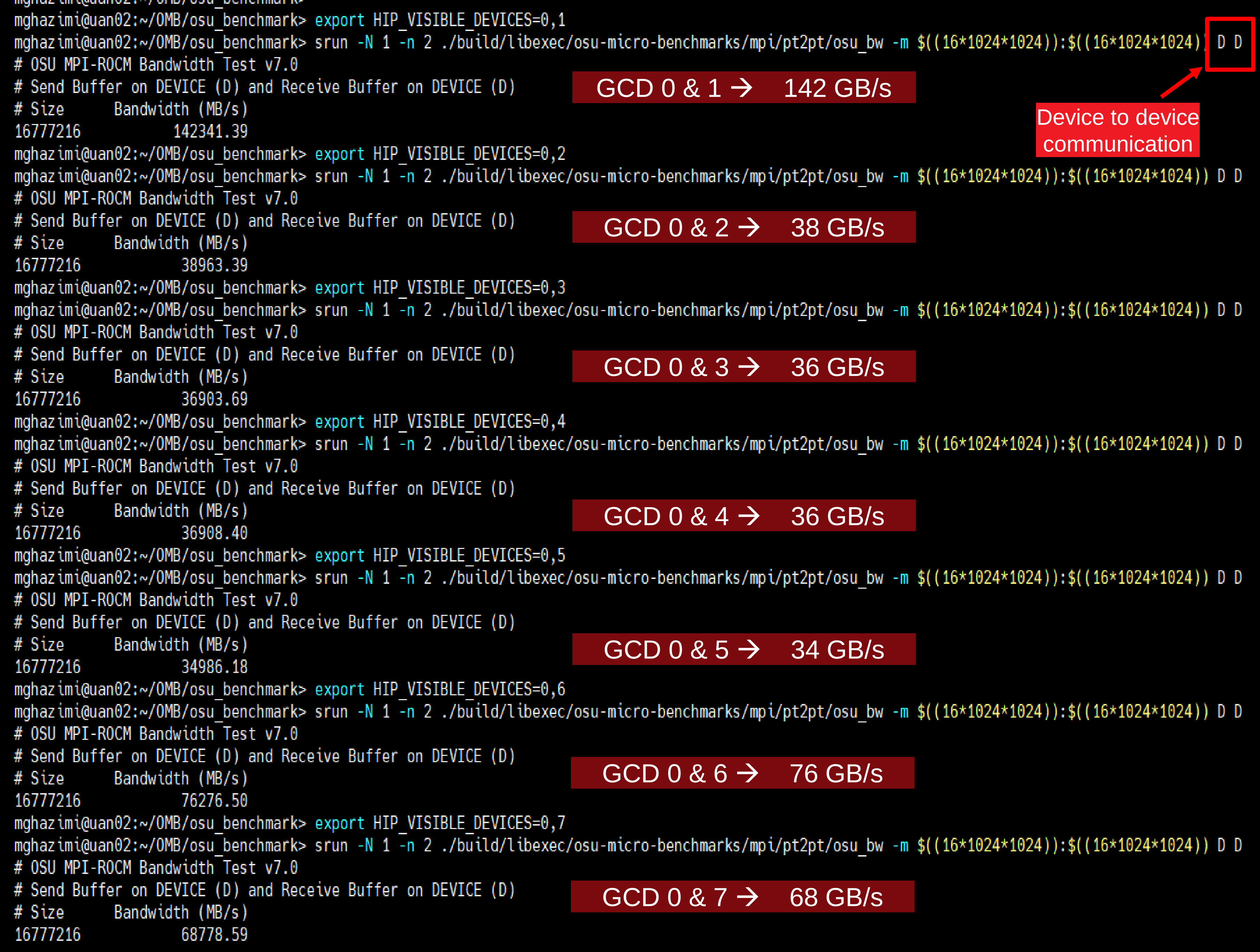
- Different number of Infinity Fabric™ links between GCDs
  - GCDs of the same GPU are connected with 4 Infinity Fabric™ links

- Different number of hops between GCDs



[14] The numbers in this figure is calculated with (achievable bw ~ Theoretical bw * 0.7). Theoretical bw is obtained from https://docs.olcf.ornl.gov/systems/frontier_user_guide.html#frontier-compute-nodes

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



```
mghazimi@uan02:~/OMB/osu_benchmark>
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
```

Device to device communication

GCD 0 & 1 → 142 GB/s

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

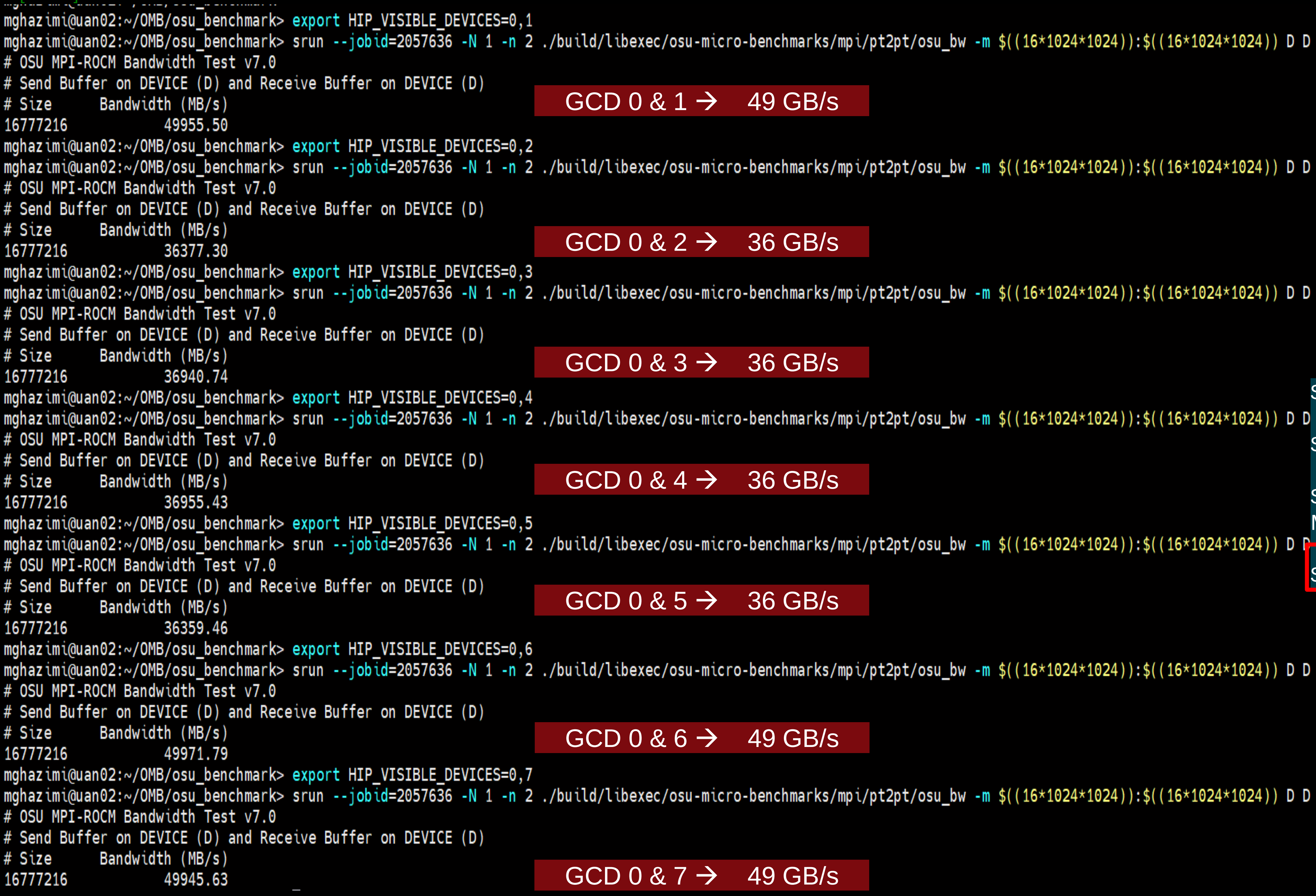
GCD 0 & 7 → 68 GB/s

```
$module load rocm

$module load cray-mpich/8.1.18

$export MPICH_GPU_SUPPORT_ENABLED=1

$export HSA_ENABLE_SDMA=0
```

Enable blit kernel

These tests have been executed as a demo (not performance claim) on LUMI pilot partition with MI250x GPUs, rocm 5.0.2 and cray-mpich/8.1.18 on Nov/23/2022.

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



```
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

These tests have been executed as a demo (not performance claim) on LUMI pilot partition with MI250x GPUs, rocm 5.0.2 and cray-mpich/8.1.18 on Nov/23/2022

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
- Note that these numbers are with rocm5.0.2 which is currently available on LUMI

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

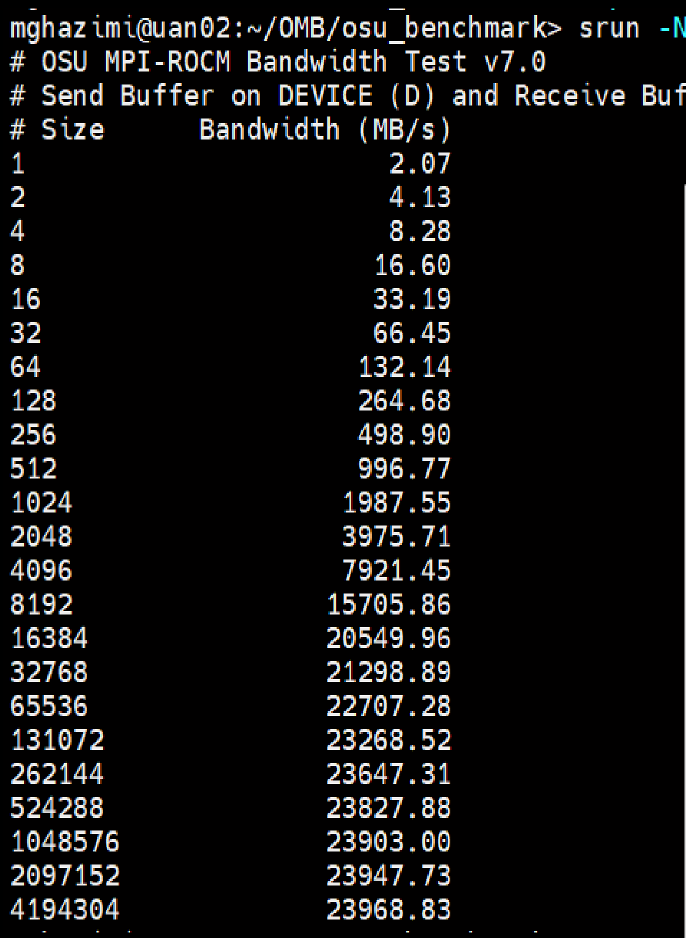
	GCD1	GCD2	GCD3	GCD4	GCD5	GCD6	GCD7
GCD0	142	38	36	36	34	76	68

Achieved Bandwidth on LUMI with SDMA (GB/s)

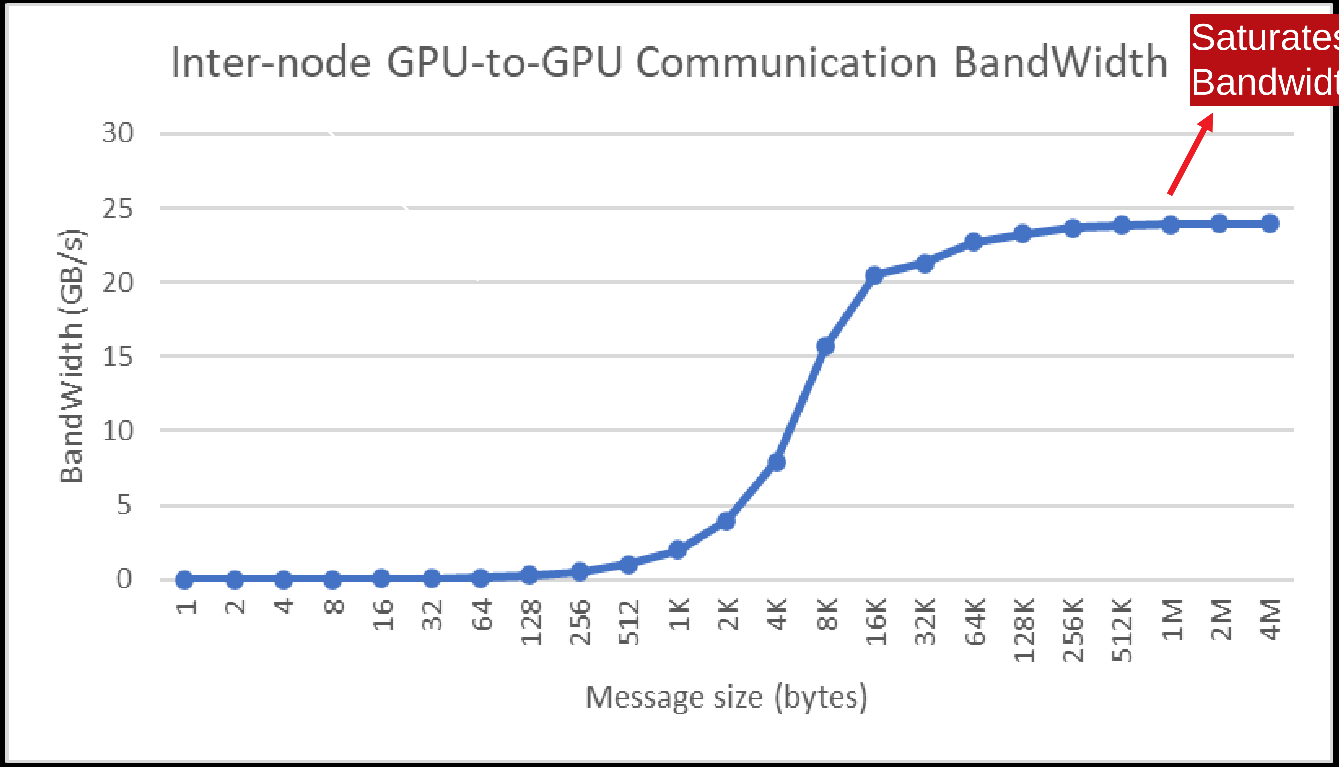
	GCD1	GCD2	GCD3	GCD4	GCD5	GCD6	GCD7
GCD0	49	36	36	36	34	49	49

These numbers have been taken as a demo on (not performance claim)LUMI pilot partition with MI250x GPUs , rocm 5.0.2 and cray-mpich/8.1.18 on Nov/23/2022

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
```



Saturates Slingshot 11 Bandwidth ~ 25GB/s

This test has been executed as a demo (not performance claim) on LUMI pilot partition with MI250x GPUs, rocm 5.0.2 and cray-mpich/8.1.18 on Nov/23/2022

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

These tests have been executed as a demo (not performance claim) on LUMI pilot partition with MI250x GPUs , rocm 5.0.2 and cray-mpich/8.1.18 on Nov/23/2022

Conclusion

- GPU-Aware MPI provides the opportunity to pass GPU buffers to MPI calls
- Many MPI implementations including OpenMPI and Cray-MPICH support GPU-Aware communication
- Using OSU microbenchmark to measure communication bandwidth and latency between GPUs
- Measured intra-node/inter-node communication bandwidth
- Measured collective communication performance
- The communication bandwidth between GCDs depend on
  - Using SDMA vs blit kernel
  - Number of Infinity Fabric™ links between GCDs
  - Number of hops between GCDs

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.

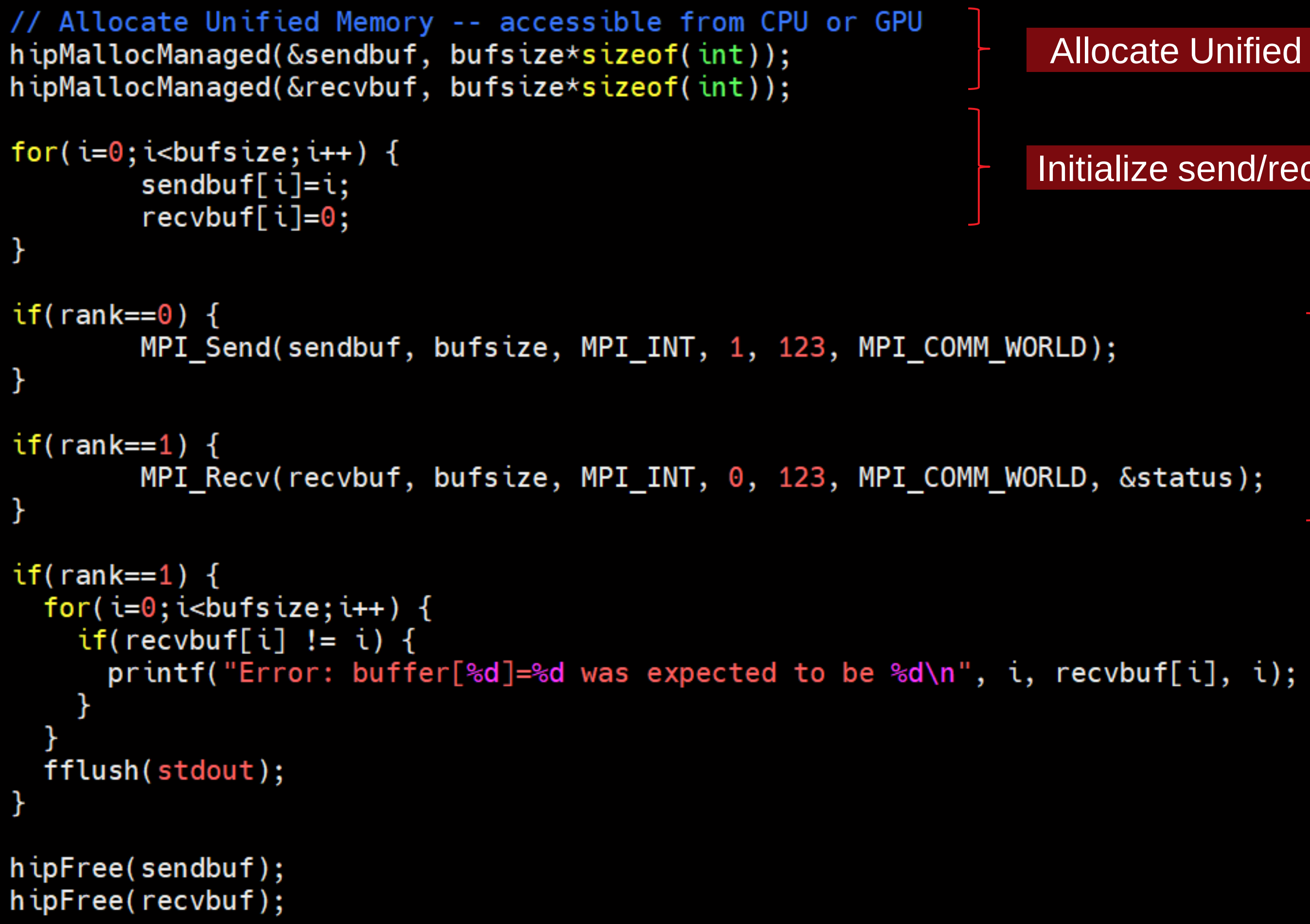
THIS INFORMATION IS PROVIDED ‘AS IS.” AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, 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.

Backup slide(s)

MPI Communication Example with Unified Memory

- Unified Memory is a technology that 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