Testing GPUDirect RDMA on DGX1 Systems
Ahmed Shamsul Arefin, PhD
Scientific Computing, IM&T, CSIRO Canberra, Australia

ABSTRACT
In this work, we present a test deployment of a peer-to-peer remote direct memory access (RDMA) technology called GPUDirect introduced by NVIDIA. We deployed it on two DGX1 systems which demonstrates performance improvement on GPGPU accelerated HPC-based Machine Learning applications, such as Tensorflow.

INTRODUCTION
GPUDirect is a family of NVIDIA technologies that enables direct data exchange between multiple GPUs, third party network adapters, solid-state drives and other devices using standard features of PCIe [1]. Among these features, the two most related to HPC-ML are peer-to-peer (P2P) transfers between GPUs and remote direct memory access (RDMA).

GPUDirect RDMA is a multi-host version that enables a Host Channel Adapter (HCA) to directly write and read GPU memory data buffers and then transfer that data through a remote HCA to a GPU on a second host, again without the need to copy data to host memory or involve the host CPU (see Figure 1).

DEPLOYMENT
We deployed the GPUDirect technology on two test nodes: d1 and d2, with fresh installation of the Ubuntu 18.04. The setup was tested using OSU Benchmarks (P2P and Latency) and Tensorflow with resnet50 model with GPU and MPI enabled. The testing has been deployed as per the instructions in the MLNX GPUDirect User Manual [3].

SOFTWARE INSTALLATION
The following five software packages were used and configured.
1. MLNX OFED 4.6.1
2. NVIDIA CUDA 10.1
3. NVIDIA GDRCopy [4]
5. MVAPICH with GDR 2.3.1 [6]

We setup the DGX1 systems standalone systems which can communicate each other and Keyless ssh accesses were setup from one to another.

MLNX OFED INSTALLATION
wget http://content.mellanox.com/ofed/MLNX_OFED-4.6-1.0.1.1/MLNX_OFED_LINUX-4.6-1.0.1.1-ubuntu18.04-x86_64.tgz
tar xvf MLNX_OFED_LINUX-4.6-1.0.1.1-ubuntu18.04-x86_64.tgz
./mlnxofedinstall
/etc/init.d/openibd restart
CUDA SETUP

sudo dpkg -i cuda-repo-ubuntu1804_10.1.168-1_amd64.deb
sudo apt-key adv --fetch-keys https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64/7fa2af80.pub
sudo apt-get update
sudo apt-get install cuda

vim ~/.bashrc
export PATH=/usr/local/cuda/bin${PATH:+:${PATH}}
export LD_LIBRARY_PATH=/usr/local/cuda/lib64${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}

NV_PEER_MEM

tar xvf nvidia-peer-memory_1.0-8.tar.gz
cd nvidia-peer-memory-1.0/
./build_module.sh

MVAPICH-GDR

cd /tmp/
wget http://mvapich.cse.ohio-state.edu/download/mvapich/gdr/2.3.1/mofed4.4/mvapich2-gdr-mcast.cuda10.1.mofed4.4.gcc4.8.5-2.3.1-1.el7.x86_64.rpm
alien -i mvapich2-gdr-mcast.cuda10.1.mofed4.4.gcc4.8.5-2.3.1-1.el7.x86_64.rpm

ADDITONAL PACKAGES
apt install alien
sudo apt-get install libibmad5
sudo apt-get install libcr-dev
OUTCOMES

We created a directory called “cloud” that is NFS shared between the both d1 and d2 DGX1 systems, where downloaded the OSU P2P, Latency benchmarks and the Tensor flow tests.

```
user@d1:/cloud/pt2pt$ cat hosts
d1
d2
```

P2P TEST

We started with a P2P test between d1 and d2, which demonstrated that as the buffer size increases, data transfer MB/S stabilizes more and more. Please see the test outcomes below.

```
user@d1:/cloud/pt2pt$ mpirun -np 2 -f hosts ./osu_bw -d cuda D D
# OSU MPI-CUDA Bandwidth Test v5.6.1
# Send Buffer on DEVICE (D) and Receive Buffer on DEVICE (D)
# Size   Bandwidth (MB/s)
1       0.80
2       0.98
4       1.93
8       3.87
16      10.58
32      62.76
64      125.19
128     245.20
256     484.49
512     933.43
1024    1674.86
2048    2989.75
4096    4783.61
8192    6366.60
16384   2982.03
32768   933.43
65536   1674.86
131072  2989.75
262144  4783.61
524288  6366.60
1048576 8101.52
2097152 933.43
4194304 1674.86
```

LATENCY TEST

The OSU latency test demonstrates that an increase in the buffer size is proportionate to the latency.

```
user@d1:/cloud/pt2pt$ mpirun -np 2 -f hosts ./osu_latency -d cuda D D
# OSU MPI-CUDA Latency Test v5.6.1
# Send Buffer on DEVICE (D) and Receive Buffer on DEVICE (D)
# Size   Latency (us)
0       1.64
1       2.70
2       3.56
4       3.56
8       3.56
16      2.75
32      3.24
64      3.17
128     3.22
256     3.37
512     3.35
1024    3.52
2048    3.84
```
The tensor flow was tested with the model called resnet50 [7], which is used for Deep Residual Learning for Image Recognition. First, we run the test on only one node d1.

```
python tf_cnn_benchmarks.py --model=resnet50
```

```
Step  Img/sec  total_loss
1    images/sec: 246.1 +/- 0.0 (jitter = 0.0) 8.220
10   images/sec: 246.9 +/- 0.1 (jitter = 0.3) 7.880
20   images/sec: 247.0 +/- 0.1 (jitter = 0.2) 7.910
30   images/sec: 247.0 +/- 0.1 (jitter = 0.2) 7.820
40   images/sec: 247.0 +/- 0.1 (jitter = 0.3) 8.004
50   images/sec: 247.0 +/- 0.1 (jitter = 0.3) 7.769
60   images/sec: 247.0 +/- 0.1 (jitter = 0.3) 8.115
70   images/sec: 247.0 +/- 0.0 (jitter = 0.3) 7.815
80   images/sec: 247.0 +/- 0.0 (jitter = 0.3) 7.979
90   images/sec: 247.0 +/- 0.0 (jitter = 0.3) 8.097
100  images/sec: 247.0 +/- 0.0 (jitter = 0.2) 8.035
```

```
total images/sec: 246.93
```

Next, we enabled the Horovod to support the Tensorflow with MPI. We also exported the following environment variables to setup the MPI with CUDA and GDR support.

```
export MV2_USE_CUDA=1
export MV2_SMP_USE_CMA=0
export MV2_USE_GDRCOPY=1
export MV2_GUDIRECT_GDRCOPY_LIB=/usr/local/gdrcopy/lib64/libgdrcopy.so
export MV2_IBA_HCA=mlx5_0
export MV2_SUPPORT_TENSOR_FLOW=1
```

```
mpirun -np 2 -f hosts python tf_cnn_benchmarks.py --model=resnet50 --variable_update=horovod
```

```
Step  Img/sec  total_loss
1    images/sec: 236.3 +/- 0.0 (jitter = 0.0) 8.217
10   images/sec: 236.2 +/- 0.4 (jitter = 1.5) 7.877
20   images/sec: 234.3 +/- 1.0 (jitter = 1.8) 7.901
30   images/sec: 235.2 +/- 0.7 (jitter = 1.7) 7.815
40   images/sec: 235.2 +/- 0.5 (jitter = 1.6) 7.986
50   images/sec: 235.6 +/- 0.5 (jitter = 1.4) 7.750
60   images/sec: 235.5 +/- 0.4 (jitter = 1.4) 8.086
70   images/sec: 235.7 +/- 0.3 (jitter = 1.3) 7.797
80   images/sec: 235.8 +/- 0.3 (jitter = 1.1) 7.953
90   images/sec: 235.6 +/- 0.3 (jitter = 1.1) 8.054
100  images/sec: 235.4 +/- 0.3 (jitter = 1.1) 7.989
```

```
total images/sec: 470.77
```

TENSORFLOW

The tensorflow was tested with the model called resnet50 [7], which is used for Deep Residual Learning for Image Recognition.
the Tensorflow with MPI and GDR was much faster to execute, and the total number of images processed was doubled per second.

CONCLUSION AND FUTURE WORKS

In this work, we demonstrated the installation procedure of the latest version of GPUDirect technology. It takes times to create a test environment and the relevant user manuals are often outdated. This work should ease the sysadmins who wish to deploy the latest GPUDirect on a GPU-enabled cluster system. We tested it on two DGX1 systems as a proof concept, but aim to deploy with in our production cluster shortly and update the work.

REFERENCES

2. Potheri, M. Scaling HPC and ML with GPUDirect RDMA on vSphere 6.7, VMWare Blogs, June 19, 2018
4. NVIDIA GDRCopy https://github.com/NVIDIA/gdrcopy
5. Mellanox Technologies GPUDirect nv_peer_mem
6. MVAPICH with GDR http://mvapich.cse.ohio-state.edu/userguide/gdr/