Data Transfer to Remote GPUs Over High-Performance Networks

Ali Marafi[1] **- Andrea Bocci**[2]

[1] Kuwait University, gotocollegenow@gmail.com

[2] CERN, Andrea.Bocci@cern.ch

Abstract: Rate of transferring data is one of the most important aspects in science especially when it comes to high computation performance. In most cases, the prerequisite of computations is data while data occasionally not stored locally, it needs to be transferred remotely. Thus, finding the best way to transfer data into the computation location (GPU) plays a critical role in performance of computations. Different methods applied in transferring data implementing Open MPI with CUDA libraries and Mellanox hardware are chosen in this paper for demonstration. The results, remote data can be achieved as fast as local data or faster when using Remote Direct Memory Access (RDMA) or RDMA Over Converged Ethernet (RoCE).

1. Introduction

The main purpose of this experiment is to find the most efficient method of transferring data over a network connection from a client machine to a remote GPU in a server machine, performing a computation on the remote GPU, and transferring the results back to the client. In order to evaluate the impact of using a remote GPU, we need a reference that is not affected by any network overhead. Therefore, we use as a reference the performance of a machine that does the same computation on a local GPU. Naively, the local GPU should be faster compared to any method that uses a remote GPU.

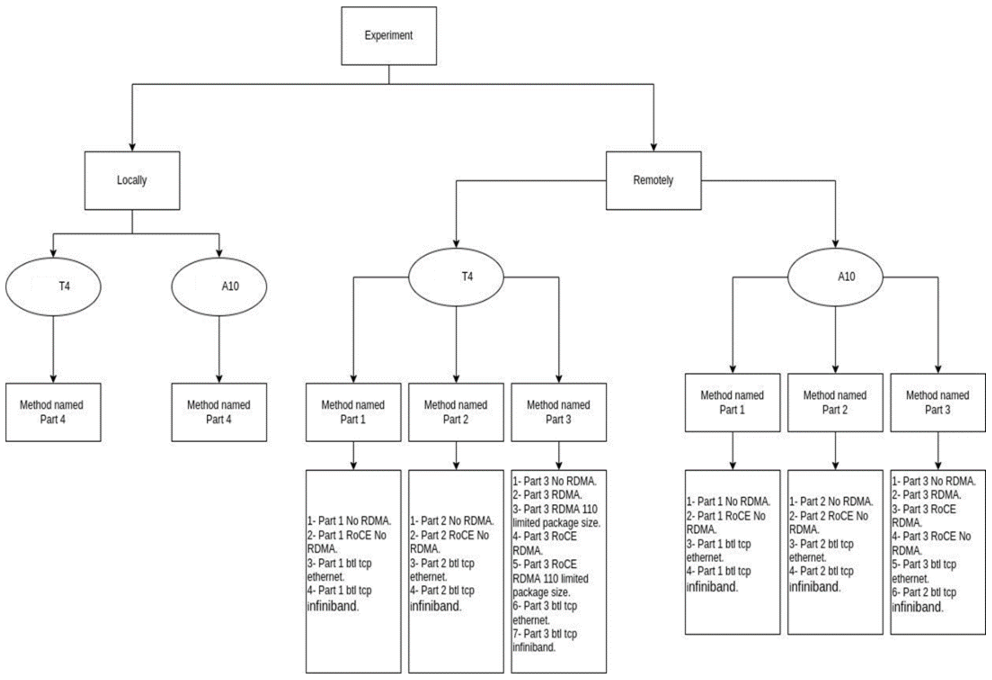
Experiment
Locally
Remotely
T4
A10
T4
A10
Method named Part 4
Method named Part 4
Method named Part 1
Method named Part 2
Method named Part 3
Method named Part 1
Method named Part 2
Method named Part 3
1- Part 1 No RDMA. 2- Part 1 RoCE No RDMA. 3- Part 1 btl tcp ethernet. 4- Part 1 btl tcp infiniband.
1- Part 2 No RDMA. 2- Part 2 RoCE No RDMA. 3- Part 2 btl tcp ethernet. 4- Part 2 btl tcp infiniband.
1- Part 3 No RDMA. 2- Part 3 RDMA. 3- Part 3 RDMA 110 limited package size. 4- Part 3 RoCE RDMA. 5- Part 3 RoCE RDMA 110 limited package size. 6- Part 3 btl tcp ethernet. 7- Part 3 btl tcp infiniband.
1- Part 1 No RDMA. 2- Part 1 RoCE No RDMA. 3- Part 1 btl tcp ethernet. 4- Part 1 btl tcp infiniband.
1- Part 2 No RDMA. 2- Part 2 RoCE No RDMA. 3- Part 2 btl tcp ethernet. 4- Part 2 btl tcp infiniband.
1- Part 3 No RDMA. 2- Part 3 RDMA. 3- Part 3 RoCE RDMA. 4- Part 3 RoCE No RDMA. 5- Part 3 btl tcp ethernet. 6- Part 2 btl tcp infiniband.

Figure 1.the entire experiment devices and parts.

2. Hardware

The measurements were performed on two pairs of machines.
The first pair is equipped with dual Intel Xeon Gold 6130[1] "Skylake" CPUs (with PCIe 3.0), Mellanox ConnectX-5 100Gb/s network cards[2] that can operate in InfiniBand and RDMA over Converged Ethernet (RoCE) mode, and Intel 10Gb/s Ethernet network cards; one of the two machines is equipped with an NVIDIA Tesla T4 GPU[3]; the machines are connected to each other via both Intel and Mellanox cards.
The second pair is equipped with a single AMD EPYC 7502P[4] "Rome" CPU (with PCIe 4.0), Mellanox ConnectX-5 Ex 100Gb/s network cards that can operate in InfiniBand and RoCE mode, and Broadcom NetXtreme 1Gb/s Ethernet network cards; one of the two machines is equipped with an NVIDIA A10 GPU[5]; the machines are connected to each other via both Broadcom and Mellanox cards.
The Broadcom and Intel cards support only the standard IP-over-Ethernet protocol. The Mellanox cards have been tested using the InfiniBand, RoCE, and IP-over-InfiniBand protocols.

3. Software

To perform the measurements using both local and remote GPUs we wrote two applications: a first one for programming a local GPU, **`cudaTimeMeasurment`**, and a second one for programming a remote GPU, **`mpiCudaGeneric`**. Figure 2 shows the different parts of the experiment. Both programs generate two arrays of single precision floating point numbers with a size variable between 10 and 400 million elements, then transfer them to the GPU to perform a computation, and copy the resulting array back to the host memory. The GPU simply computes the pairwise sum of the two arrays; to emulate the impact of a more complex computation, the sum can be repeated an arbitrary number of times. Finally, the result is compared with the result of the same operation performed on the CPU; if the results are correct, the program prints the amount of time spent in the various parts of the task: data transfers and GPU computations. Both programs use CUDA 11.5[6] to program the GPUs.
The mpiCudaGeneric application uses the Open MPI 4.1[7] framework and optionally the Unified Communication X (UCX)[8] library for the inter-process communications. The data copies from the client machine to the GPU in the server machines have been implemented in different ways: in "part 1", the data is copied to a buffer on the server pageable host memory, and from there to the GPU memory; in "part 2", the data is copied directly to a buffer in pinned host memory, which supports a faster copy to the GPU memory. When using the Mellanox cards in InfiniBand or RoCE mode with the UCX library, the communication with the remote GPU can take advantage of the NVIDIA GPUDirect Remote Direct Memory Access (RDMA) to transfer the data directly to or from the remote GPU memory, bypassing the machine's host memory. This is done in "part 3", where the data is copied directly from the client host memory to the server's GPU memory.
The **`cmsenv_mpirun`** script sets up the environment for running the local and remote applications, then calls mpirun and passes it the user application and its parameters.
An example can be seen in Figure 2.

```
cmsenv_mpirun\
    -H <client> -np 1 --mca pml ucx -x UCX_NET_DEVICES=mlx5_0:1 -- \
    ./mpiCudaGeneric -t1 -a10 -p111333333 -s1000 -f : \
    -H <server> -np 1 --mca pml ucx -x UCX_NET_DEVICES=mlx5_0:1 -x UCX_IB_GPU_DIRECT_RDMA=yes -- \
    <path>/mpiCudaGeneric -t1 -a10 -p111333333 -s1000
```

Figure 2. example to run the mpiCudaGeneric application on the client and server machines. The --mca pml ucx instructs Open MPI to use the UCX transport library. The -x options pass additional configuration to UCX, for example, "`-x UCX_IB_GPU_DIRECT_RDMA=yes`" to explicitly enable RDMA . The -t1 -a10 -p111333333 -s1000 parameters instruct the mpiCudaGeneric to run the "part 3" measurement six times, each repeated ten times, with a buffer of 1000 elements.

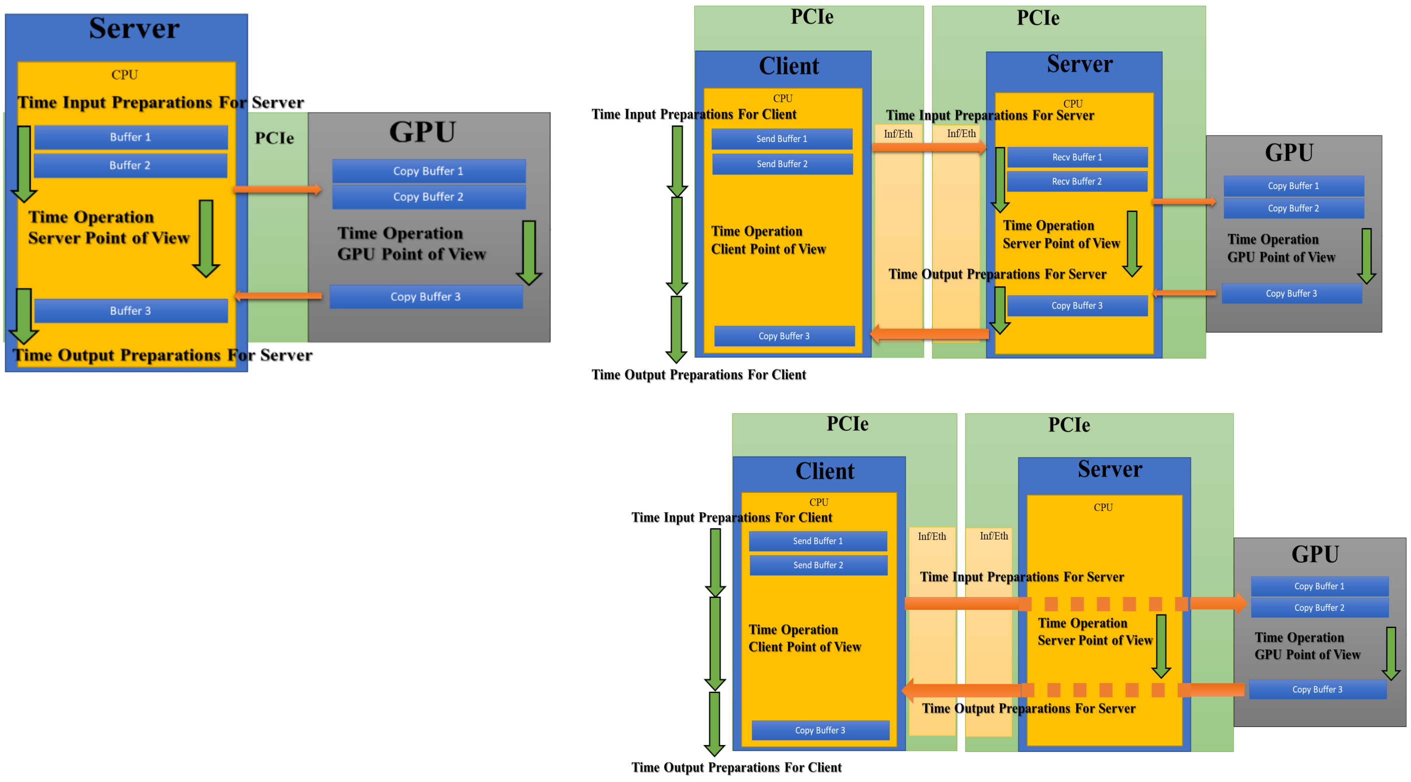
Server
CPU
Time Input Preparations For Server
Buffer 1
Buffer 2
Time Operation Server Point of View
Buffer 3
Time Output Preparations For Server
PCIe
GPU
Copy Buffer 1
Copy Buffer 2
Time Operation GPU Point of View
Copy Buffer 3
PCIe
Client
CPU
Time Input Preparations For Client
Send Buffer 1
Send Buffer 2
Time Operation Client Point of View
Copy Buffer 3
Time Output Preparations For Client
Inf/Eth
Inf/Eth
Server
Time Input Preparations For Server
Recv Buffer 1
Recv Buffer 2
Time Operation Server Point of View
Time Output Preparations For Server
Copy Buffer 3
GPU
Copy Buffer 1
Copy Buffer 2
Time Operation GPU Point of View
Copy Buffer 3

Figure 3, Architecture of time measurements in the local experiment (left). Architecture of time measurements in part 1 and 2 (right). Architecture of time measurements in part 3 (bottom).

3.1. Local GPUs

Working with a local GPU, the time measurements are taken in four points as shown in Figure 3: the **Time Input Preparations** measures the time to copy the data from pageable host memory to pinned host memory, then copy it from pinned host memory to GPU memory; the **Time Operations on CPU Point of View** is the time spent performing the computation, as measured on the CPU; the **Time Operations on Device Point of View** is the same time measured on the GPU; finally, the **Time Output Preparations** measures the time spent copying the results from GPU memory to pinned host memory, and then to pageable memory.

Thus, the total time spent performing all the copies can be extracted from the total time spent on the server, minus the time spent by the GPU performing the computations:

Latency = Time input preparations + Time Operations on Server + Time output preparations - Time Operations on Device.

3.2. Remote GPUs

While the measurements working with remote GPUs are more complex, as shown in the right side of Figure 3, they follow a similar scheme. When performing copies using an intermediate buffer on the server host (top), the overall measurement must include the time spent transferring the data first to the server host memory, and then from there to the GPU memory. When using an RDMA transfer like InfiniBand or RoCE (bottom) there is no intermediate step.

In any case, the time spent performing all the copies from the client to the remote GPU can be extracted from the total time as measured by the client, minus the time spent by the GPU performing the computations:

Latency = Time input preparation for client + Time Operations client point of view + Time output preparations for client - Time operations GPU point of view

4. Results

The measurement process involves first warming up the GPU by running "part 1" of the program three times as shown in Figure 2. Next, the actual measurement is performed six times, with the average and standard deviation extracted for each run. This entire process is repeated ten times for each step, and the measurement with the smallest standard deviation is used as the representative value.

The size of the data buffers is gradually increased from 10 to 400 million elements, as shown in Figure 4 (right) for the A10 GPU. For the T4 GPU (Figure 4, left), the buffer size is limited to 28 million elements due to PCIe BAR size limitations of 128 MB.

Figure 5 shows a comparison of the data transfer performance for local GPUs and remote GPUs that rely on Remote Direct Memory Access (RDMA) for several key buffer sizes: 10k, 1M, and 28M elements for the T4 (left), the same buffer sizes plus 100M elements for the A10 (right).

In Figure 6, a similar comparison is shown for different interconnects and transfer methods that do not rely on RDMA. It is interesting to note that these methods can reach the maximum buffer size of 400M elements for both the T4 (left) and A10 (right) GPUs because they don't rely on RDMA.

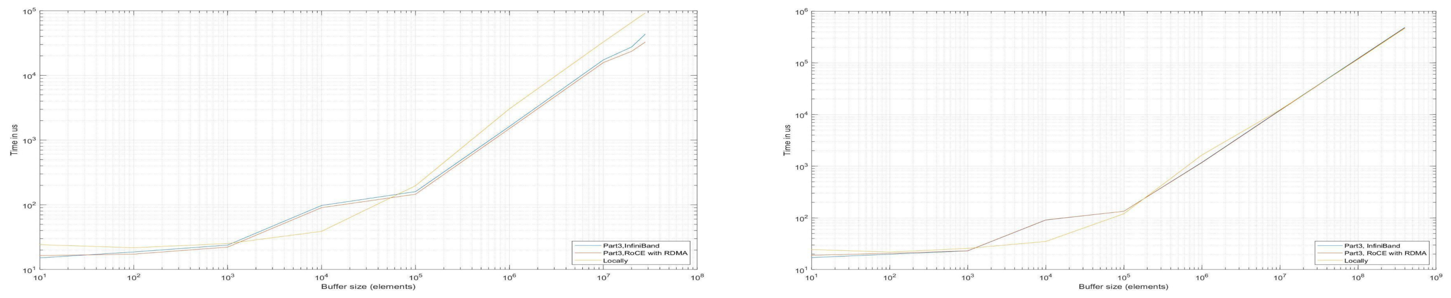

Figure 4. Transfer time in milliseconds vs. buffer size in elements for local, remote over InfiniBand (with RDMA), and and RoCE (with RDMA) on a T4 GPU (left) and an A10 GPU (right).

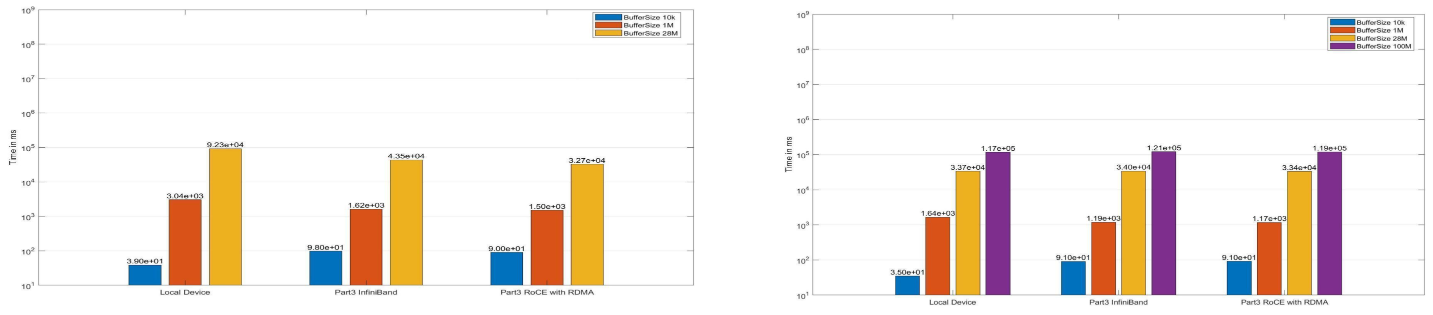

Figure 5. Transfer time in milliseconds for selected buffers sizes of 10k, 1M, 28M and 100M elements for local, remote over InfiniBand (with RDMA), and RoCE (with RDMA) on a T4 GPU (left) and an A10 GPU (right).
RDMA transfers to a T4 GPU are limited to about 30M elements.

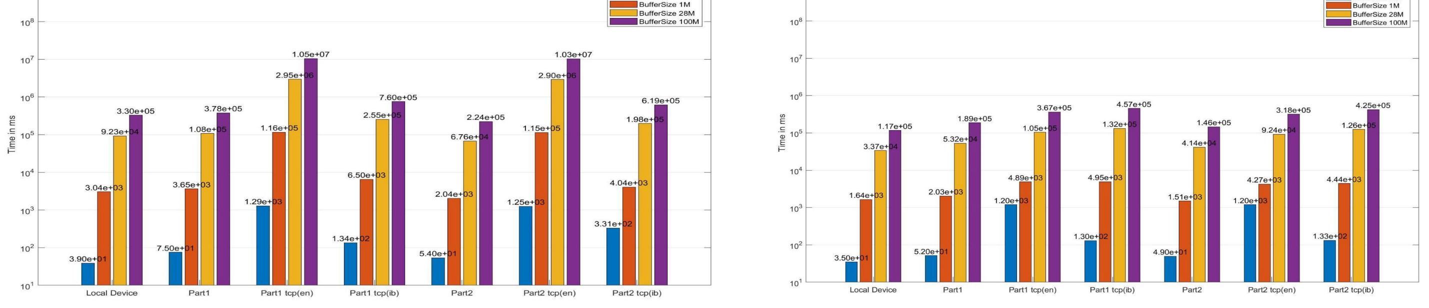

Figure 6. Transfer time in milliseconds for selected buffer sizes of 10k, 1M, 28M and 100M elements for local transfers and different remote protocols (without RDMA) on a T4 GPU (left) and an A10 GPU (right).

5. Conclusions

As expected, the Mellanox cards are significantly faster than the Ethernet cards we tested. Interestingly, in the Mellanox card experiment there was not a significant difference in performance between the InfiniBand and RoCE protocols. On the other hand, the InfiniBand protocol has better performance compared with TCP/IP protocol: it is 2 to 3 times faster than the IP-over-InfiniBand protocol. With respect to RDMA technology, the RDMA capabilities of data transfer to remote GPUs is as fast (or even faster) than using a local GPU. This is a very encouraging result for extending the use of GPUs beyond those available on a local machine. Finally, in the absence of RDMA technology, the measurements illustrate that using an intermediate buffer in pinned host memory plays a vital role in speeding up the data transfers with a remote GPU.

References

1) Intel® Xeon® Gold 6130 Processor, https://ark.intel.com/content/www/us/en/ark/products/120492/intel-xeon-gold-6130-processor-_22m-cache-2-10-ghz.html
2) Mellanox ConnectX®-5 VPI EDR/100GbE Adapters, https://support.mellanox.com/s/productdetails/a2v5000000052eDAAQ/connectx5-card
3) NVIDIA® T4 70W, https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-t4/t4-tensor-core-product-brief.pdf
4) Advanced Micro Devices AMD EPYC™ 7502P, https://www.amd.com/en/products/cpu/amd-epyc-7502p
5) NVIDIA® A10 GPU Accelerator, https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a10/pdf/A10-Product-Brief.pdf
6) NVIDIA® CUDA® Toolkit, https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html
7) Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. Edgar Gabriel, GrahamE. Fagg, George Bosilca, Thara Angskun, Jack Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. In Proceedings, 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, September 2004.
8) UCX: An Open-Source Framework for HPC Network APIs and Beyond Pavel Shamis, Manjunath Gorentla Venkata, M. Graham Lopez, Matthew B. Baker, Oscar Hernandez2015 IEEE 23rd Annual Symposium on High-Performance Interconnects