Enabling RDMA

Table of contents

Note

Learn more about RDMA in the technology overview section.

There are two parts to enabling RDMA for Holoscan:

- Enabling RDMA on the ConnectX SmartNIC
- Enabling GPUDirect RDMA

Enabling RDMA on the ConnectX SmartNIC

Skip to the next section if you do not plan to leverage a ConnectX SmartNIC.

The NVIDIA IGX Orin developer kit comes with an embedded ConnectX Ethernet adapter to offer advanced hardware offloads and accelerations. You can also purchase an individual ConnectX adapter and install it on other systems such as x86_64 workstations.

The following steps are required to ensure your ConnectX can be used for RDMA over Converged Ethernet (RoCE):

1. Install MOFED drivers

Ensure the Mellanox OFED drivers version 23.10 or above are installed:

```
cat /sys/module/mlx5_core/version
```

If not installed, or an older version, install the appropriate version from the MLNX_OFED download page, or use the script below:

```
# You can choose different versions/OS or download directly from the # Download
Center in the webpage linked above MOFED_VERSION="23.10-2.1.3.1"
OS="ubuntu22.04"
MOFED_PACKAGE="MLNX_OFED_LINUX-${MOFED_VERSION}-${OS}-$(uname -m)"
```

```
wget --progress=dot:giga
https://www.mellanox.com/downloads/ofed/MLNX_OFED-${MOFED_VERSION}/${MOF
tar xf ${MOFED_PACKAGE}.tgz sudo ./${MOFED_PACKAGE}/mlnxofedinstall # add
the --force flag to force uninstallation if necessary: # sudo
./${MOFED_PACKAGE}/mlnxofedinstall --force rm -r ${MOFED_PACKAGE}*
```

2. Load MOFED drivers

Ensure the drivers are loaded:

```
sudo lsmod | grep ib_core
```

If nothing appears, run the following command:

```
sudo /etc/init.d/openibd restart
```

3. Switch the board Link Layer to Ethernet

The ConnectX SmartNIC can function in two separate modes (called link layer):

- Ethernet (ETH)
- Infiniband (IB)

Holoscan does not support IB at this time (not tested), so the ConnectX will need to use the ETH link layer.

To identify the current mode, run `ibstat` or `ibv_devinfo` and look for the `Link Layer` value. In the example below, the `mlx5_0` interface is in Ethernet mode, while the `mlx5_1` interface is in Infiniband mode. Do not pay attention to the `transport` value which is always `InfiniBand`.

```
$ ibstat CA 'mlx5_0' CA type: MT4129 Number of ports: 1 Firmware version:
28.37.0190 Hardware version: 0 Node GUID: 0x48b02d0300ee7a04 System image
GUID: 0x48b02d0300ee7a04 Port 1: State: Down Physical state: Disabled Rate: 40
Base lid: 0 LMC: 0 SM lid: 0 Capability mask: 0x00010000 Port GUID:
```

```
0x4ab02dfffeee7a04 Link layer: Ethernet CA 'mlx5_1' CA type: MT4129 Number of ports: 1 Firmware version: 28.37.0190 Hardware version: 0 Node GUID: 0x48b02d0300ee7a05 System image GUID: 0x48b02d0300ee7a04 Port 1: State: Active Physical state: LinkUp Rate: 100 Base lid: 0 LMC: 0 SM lid: 0 Capability mask: 0x00010000 Port GUID: 0x4ab02dfffeee7a05 Link layer: InfiniBand
```

If no results appear after `ibstat` and `sudo lsmod | grep ib_core` returns a result like this:

```
ib_core 425984 1 ib_uverbs
```

Consider running the following command or rebooting:

```
sudo /etc/init.d/openibd restart
```

To switch the link layer mode, there are two possible options:

1. On IGX Orin developer kits, you can switch that setting through the BIOS: see IGX Orin documentation.

2. On any system with a ConnectX (including IGX Orin devkits), you can run the command below from a terminal (requires a reboot). `sudo ibdev2netdev -v` is used to identify the PCI address of the ConnectX (any of the two interfaces is fine to use), and `mlxconfig` is used to apply the changes.

   ```
   mlx_pci=$(sudo ibdev2netdev -v | awk '{print $1}' | head -n1) sudo mlxconfig -d $mlx_pci set LINK_TYPE_P1=ETH LINK_TYPE_P2=ETH
   ```

   Note: `LINK_TYPE_P1` and `LINK_TYPE_P2` are for `mlx5_0` and `mlx5_1` respectively. You can choose to only set one of them. You can pass `ETH` or `2` for Ethernet mode, and `IB` or `1` for InfiniBand.

   This is the output of the command above:

```
Device #1: ---------- Device type: ConnectX7 Name: P3740-B0-QSFP_Ax
Description: NVIDIA Prometheus P3740 ConnectX-7 VPI PCIe Switch
Motherboard; 400Gb/s; dual-port QSFP; PCIe switch5.0 X8 SLOT0 ;X16 SLOT2;
secure boot; Device: 0005:03:00.0 Configurations: Next Boot New
LINK_TYPE_P1 ETH(2) ETH(2) LINK_TYPE_P2 IB(1) ETH(2) Apply new
Configuration? (y/n) [n] :
```

`Next Boot` is actually the current value that was expected to be used at the next reboot, while `New` is the value you're about to set to override `Next Boot`.

Apply with `y` and reboot afterwards:

```
Applying... Done! -I- Please reboot machine to load new configurations.
```

4. Configure the IP addresses of the ethernet interfaces

First, identify the logical names of your ConnectX interfaces. Connecting a cable in just one of the interfaces on the ConnectX will help you identify which port is which (in the example below, only `mlx5_1` i.e. `eth3` is connected):

```
$ sudo ibdev2netdev mlx5_0 port 1 ==> eth2 (Down) mlx5_1 port 1 ==> eth3 (Up)
```

Tip

For IGX Orin Developer Kits with no live source to connect to the ConnectX QSFP ports, adding `-v` can show you which logical name is mapped to each specific port:

- `0005:03.00.0` is the QSFP port closer to the PCI slots
- `0005:03.00.1` is the QSFP closer to the RJ45 ethernet ports

```
$ sudo ibdev2netdev -v 0005:03:00.0 mlx5_0 (MT4129 - P3740-
0002 ) NVIDIA IGX, P3740-0002, 2-port QSFP up to 400G,
InfiniBand and Ethernet, PCIe5 fw 28.37.0190 port 1 (DOWN )
==> eth2 (Down) 0005:03:00.1 mlx5_1 (MT4129 - P3740-0002 )
NVIDIA IGX, P3740-0002, 2-port QSFP up to 400G, InfiniBand
and Ethernet, PCIe5 fw 28.37.0190 port 1 (DOWN ) ==> eth2
(Down)
```

If you have a cable connected but it does not show Up/Down in the output of `ibdev2netdev`, you can try to parse the output of `dmesg` instead. The example below shows that `0005:03:00.1` is plugged, and that it is associated with `eth3`:

```
$ sudo dmesg | grep -w mlx5_core ... [ 11.512808] mlx5_core
0005:03:00.0 eth2: Link down [ 11.640670] mlx5_core
0005:03:00.1 eth3: Link down ... [ 3712.267103] mlx5_core
0005:03:00.1: Port module event: module 1, Cable plugged
```

The next step is to set a static IP on the interface you'd like to use so you can refer to it in your Holoscan applications (ex: Emergent cameras, distributed applications...).

First, check if you already have an address setup. We'll use the `eth3` interface in this example for `mlx5_1`:

```
ip -f inet addr show eth3
```

If nothing appears or you'd like to change the address, you can set an IP and MTU (Maximum Transmission Unit) through the Network Manager user interface, CLI (`nmcli`), or other IP configuration tools. In the example below, we use `ip` (`ifconfig` is legacy) to configure the `eth3` interface with an address of `192.168.1.1/24` and a MTU of `9000` (i.e. "jumbo frame") to send Ethernet frames with a payload greater than the standard size of 1500 bytes:

```
sudo ip link set dev eth3 down sudo ip addr add 192.168.1.1/24 dev eth3 sudo ip
```

```
link set dev eth3 mtu 9000 sudo ip link set dev eth3 up
```

Note

If you are connecting the ConnectX to another ConnectX with a LinkX interconnect, do the same on the other system with an IP address on the same network segment.

For example, to communicate with `192.168.1.1/24` above (`/24` -> `255.255.255.0` submask), setup your other system with an IP between `192.168.1.2` and `192.168.1.254`, and the same `/24` submask.

Enabling GPUDirect RDMA

Note

Only supported on NVIDIA's Quadro/workstation GPUs (not GeForce).

Follow the instructions below to enable GPUDirect RDMA:

Ingested Tab Module

Testing with Rivermax

The instructions below describe the steps to test GPUDirect using the Rivermax SDK. The test applications used by these instructions, `generic_sender` and `generic_receiver`, can then be used as samples in order to develop custom applications that use the Rivermax SDK to optimize data transfers.

Note

The Linux default path where Rivermax expects to find the license file is `/opt/mellanox/rivermax/rivermax.lic`, or you can specify the full path and file name for the environment variable `RIVERMAX_LICENSE_PATH`.

Note

If manually installing the Rivermax SDK from the link above, please note there is no need to follow the steps for installing MLNX_OFED/MLNX_EN in the Rivermax documentation.

Running the Rivermax sample applications requires two systems, a sender and a receiver, connected via ConnectX network adapters. If two Developer Kits are used then the onboard ConnectX can be used on each system, but if only one Developer Kit is available then it is expected that another system with an add-in ConnectX network adapter will need to be used. Rivermax supports a wide array of platforms, including both Linux and Windows, but these instructions assume that another Linux based platform will be used as the sender device while the Developer Kit is used as the receiver.

Note

The `$rivermax_sdk` variable referenced below corresponds to the path where the Rivermax SDK package was installed. If the Rivermax SDK was installed via SDK Manager, this path will be:

```
rivermax_sdk=$HOME/Documents/Rivermax/1.31.10
```

If the Rivermax SDK was installed via a manual download, make sure to export your path to the SDK:

```
rivermax_sdk=$DOWNLOAD_PATH/1.31.10
```

Install path might differ in future versions of Rivermax.

1. Determine the logical name for the ConnectX devices that are used by each system. This can be done by using the `lshw -class network` command, finding the `product:` entry for the ConnectX device, and making note of the `logical name:` that corresponds to that device. For example, this output on a Developer Kit shows the onboard ConnectX device using the `enp9s0f01` logical name (`lshw` output shortened for demonstration purposes).

   ```
   $ sudo lshw -class network *-network:0 description: Ethernet interface product: MT28908 Family [ConnectX-6] vendor: Mellanox Technologies physical id: 0 bus info: pci@0000:09:00.0 <b>logical name: enp9s0f0</b> version: 00 serial: 48:b0:2d:13:9b:6b capacity: 10Gbit/s width: 64 bits clock: 33MHz capabilities: pciexpress vpd msix pm bus_master cap_list ethernet physical 1000bt-fd 10000bt-fd autonegotiation configuration: autonegotiation=on broadcast=yes driver=mlx5_core driverversion=5.4-1.0.3 duplex=full firmware=20.27.4006 (NVD0000000001) ip=10.0.0.2 latency=0 link=yes multicast=yes resources: iomemory:180-17f irq:33 memory:1818000000-1819ffffff
   ```

   The instructions that follow will use the `enp9s0f0` logical name for `ifconfig` commands, but these names should be replaced with the corresponding logical names as determined by this step.

2. Run the `generic_sender` application on the sending system.

   a. Bring up the network:

   ```
   $ sudo ifconfig enp9s0f0 up 10.0.0.1
   ```

   b. Build the sample apps:

```
$ cd ${rivermax_sdk}/apps $ make
```

e. Launch the `generic_sender` application:

```
$ sudo ./generic_sender -l 10.0.0.1 -d 10.0.0.2 -p 5001 -y 1462 -k 8192 -z 500 -v
... +############################################# | Sender
index: 0 | Thread ID: 0x7fa1ffb1c0 | CPU core affinity: -1 | Number of streams
in this thread: 1 | Memory address: 0x7f986e3010 | Memory length:
59883520[B] | Memory key: 40308
+############################################# | Stream
index: 0 | Source IP: 10.0.0.1 | Destination IP: 10.0.0.2 | Destination port: 5001
| Number of flows: 1 | Rate limit bps: 0 | Rate limit max burst in packets: 0 |
Memory address: 0x7f986e3010 | Memory length: 59883520[B] | Memory key:
40308 | Number of user requested chunks: 1 | Number of application chunks:
5 | Number of packets in chunk: 8192 | Packet's payload size: 1462
+**********************************************
```

3. Run the `generic_receiver` application on the receiving system.

a. Bring up the network:

```
$ sudo ifconfig enp9s0f0 up 10.0.0.2
```

b. Build the `generic_receiver` app with GPUDirect support from the Rivermax GitHub Repo. Before following the instructions to build with CUDA-Toolkit support, apply the changes to file `generic_receiver/generic_receiver.cpp` in this PR, this was tested on the IGX Orin Developer Kit with Rivermax 1.31.10.

c. Launch the `generic_receiver` application from the `build` directory:

```
$ sudo ./generic_receiver -i 10.0.0.2 -m 10.0.0.2 -s 10.0.0.1 -p 5001 -g 0 ...
Attached flow 1 to stream. Running main receive loop... Got 5877704 GPU
packets | 68.75 Gbps during 1.00 sec Got 5878240 GPU packets | 68.75 Gbps
during 1.00 sec Got 5878240 GPU packets | 68.75 Gbps during 1.00 sec Got
```

```
5877704 GPU packets | 68.75 Gbps during 1.00 sec Got 5878240 GPU packets | 68.75 Gbps during 1.00 sec ...
```

With both the `generic_sender` and `generic_receiver` processes active, the receiver will continue to print out received packet statistics every second. Both processes can then be terminated with `&lt;ctrl-c&gt;`.