NCCL Testing Over RoCE Fabric - Complete Tutorial

Overview

This guide covers setting up and running NCCL (NVIDIA Collective Communications Library) tests between bare metal GPU nodes using RoCE (RDMA over Converged Ethernet) fabric for high-performance inter-node communication.

Prerequisites

- Multiple bare metal servers with NVIDIA GPUs

Installation

```
None
apt install -y git
rm -rf nccl-tests
git clone https://github.com/NVIDIA/nccl-tests.git
cd nccl-tests
make -j16 MPI=1 MPI_HOME=/usr/lib/x86_64-linux-gnu/openmpi/
```

1. Hardware and Network Setup

Network Infrastructure Requirements

- **Switch Configuration**: Enable PFC (Priority Flow Control) and ECN (Explicit Congestion Notification)
- **Cabling**: Use appropriate cables (typically 25GbE, 50GbE, or 100GbE)
- **Network Topology**: Ensure low-latency, high-bandwidth connectivity between nodes

Verify GPU and Network Hardware

```
Shell
# Check GPUs
nvidia-smi

# Check network interfaces
ip link show
ibv_devices  # Should show RDMA devices

# Check RoCE capability
show_gids | grep -i roce

# Method 1: Use rdma command (more modern approach)
rdma link show

# Method 2: Use ibv_devinfo to check port capabilities
ibv_devinfo -v | grep -i roce
```

2. Software Installation

Install Required Packages

```
Shell
# Update system
sudo apt update && sudo apt upgrade -y

# Install essential packages (optional: already in the system)
sudo apt install -y build-essential cmake git
sudo apt install -y libibverbs-dev librdmacm-dev
sudo apt install -y rdma-core perftest

# Install NVIDIA drivers (if not already installed)
wget
https://developer.download.nvidia.com/compute/cuda/repos/ubuntu20
04/x86_64/cuda-keyring_1.1-1_all.deb
sudo dpkg -i cuda-keyring_1.1-1_all.deb
sudo apt update
sudo apt install -y cuda-drivers cuda-toolkit-12-4
```

Install NCCL

```shell
# Download and install NCCL
wget
https://developer.download.nvidia.com/compute/redist/nccl/v2.19.3
/nccl_2.19.3-1+cuda12.0_x86_64.txz
tar -xvf nccl_2.19.3-1+cuda12.0_x86_64.txz
sudo cp -r nccl_2.19.3-1+cuda12.0_x86_64/* /usr/local/
sudo ldconfig

# Verify installation
pkg-config --modversion nccl
```

Install NCCL Tests (if not installed in above step)

```shell
# Clone NCCL tests repository
git clone https://github.com/NVIDIA/nccl-tests.git
cd nccl-tests

# Build NCCL tests
make MPI=1 MPI_HOME=/usr/lib/x86_64-linux-gnu/openmpi
CUDA_HOME=/usr/local/cuda NCCL_HOME=/usr/local
```

Quick Setup

1. Install NCCL Tests

```
Shell
# Install prerequisites
apt install -y git openmpi-bin openmpi-common libopenmpi-dev

# Build NCCL tests
cd ~
rm -rf nccl-tests
git clone https://github.com/NVIDIA/nccl-tests.git
cd nccl-tests
make -j16 MPI=1 MPI_HOME=/usr/lib/x86_64-linux-gnu/openmpi/
```

2. Find Your RoCE Interfaces

```
Shell
# Find interfaces with RDMA devices (these are your RoCE
interfaces)
for iface in $(ls /sys/class/net/); do
    rdma_dev=$(ls /sys/class/net/$iface/device/infiniband/
2>/dev/null)
    if [ -n "$rdma_dev" ]; then
        echo "RoCE Interface: $iface → RDMA device: $rdma_dev"
    fi
done

# Alternative: Use ibdev2netdev to see mappings
ibdev2netdev
```

Sample Output:

```None
RoCE Interface: enp156s0np0 → RDMA device: mlx5_6
```

3. Check GPU-NIC Topology

```Shell
# See which NICs are closest to which GPUs
nvidia-smi topo -m
```

Single Node Test

Create Test Script

```Shell
cat > ~/test_single_node.sh << 'EOF'
#!/bin/bash

echo "=== Single Node NCCL Test ==="

# Get GPU count
GPU_COUNT=$(nvidia-smi -L | wc -l)
echo "Testing with $GPU_COUNT GPUs"

cd ~/nccl-tests

# Run the test (target: >400 GB/s bus bandwidth)
./build/all_reduce_perf -b 1G -f 2 -e 1G -g $GPU_COUNT
EOF

chmod +x ~/test_single_node.sh
```

Run Single Node Test

Shell

```
~/test_single_node.sh
```

Expected Output: Look for busbw > 400 GB/s in the results.

Sample Output:

Shell

```
  size         count      type   redop    root     time   algbw   busbw #wrong
time   algbw   busbw #wrong
#        (B)    (elements)                              (us)  (GB/s)  (GB/s)
(us)  (GB/s)  (GB/s)
  1073741824     268435456     float     sum      -1   5110.3  210.11  367.70      0
5112.6  210.02  367.53      0
# Out of bounds values : 0 OK
# Avg bus bandwidth    : 367.616
#
# Collective test concluded: all_reduce_perf
```

Multi-Node Test

Create Multi-Node Test Script

Shell

```
cat > ~/test_multi_node.sh << 'EOF'
#!/bin/bash

# Usage: ./test_multi_node.sh "ip1,ip2"
HOSTS=$1

if [ -z "$HOSTS" ]; then
    echo "Usage: $0 \"ip1,ip2,ip3\""
    echo "Example: $0 \"192.168.1.10,192.168.1.11\""
    exit 1
```

```
fi

# Parse hosts
IFS=',' read -r -a ip_array <<< "$HOSTS"
NP=$(echo "${ip_array[@]}" | tr ' ' '\n' | sort -u | wc -l)

# Assuming 8 GPUs per node (adjust as needed)
NEW_NP=$(( NP * 8 ))
NEW_HOSTS=$(echo $HOSTS | sed 's/\([0-9\.]*\)/\1:8/g')

echo "Testing $NP nodes, $NEW_NP total GPUs"

cd ~/nccl-tests

# Run multi-node test
mpirun --bind-to none --mca pml ucx --allow-run-as-root \
        -x NCCL_CROSS_NIC=0 \
        -np $NEW_NP -H ${NEW_HOSTS} \
        ./build/all_reduce_perf -b 1G -f 2 -e 1G
EOF

chmod +x ~/test_multi_node.sh
```

Run Multi-Node Test

Shell
```
# Replace with your actual node IPs
~/test_multi_node.sh "10.45.169.116,10.45.169.109"
```

Expected Output: Look for `busbw` > 400 GB/s across nodes.

Troubleshooting

Abrupt test termination

An abrupt test termination is usually associated with hardware problems, like a misconfiguration or a bad cable, so it is worth checking the following:

- all GPU NICs are up
- they are all on the same VLAN
- they are plugged in the right leaf switches

These are example outputs associated to good nodes:

Interfaces are up =>

```
None
# Currently back-end network is addressed via IPv6, this
may be subject to change to enable PFC.
# ip -br a
lo                UNKNOWN        127.0.0.1/8 ::1/128
eno8303           DOWN
eno8403           DOWN
enp26s0np0        UP
fe80::a288:c2ff:fee4:446a/64
enp27s0f0np0      UP
enp27s0f1np1      UP
enp60s0np0        UP
fe80::a288:c2ff:fee3:638a/64
enp77s0np0        UP
fe80::a288:c2ff:feda:31ea/64
enp94s0np0        UP
fe80::a288:c2ff:feda:d712/64
enp156s0np0       UP
fe80::a288:c2ff:feda:d6c2/64
enp157s0f0np0     UP
enp157s0f1np1     UP
```

```
enp188s0np0      UP
fe80::5aa2:e1ff:fe11:5ffc/64
enp204s0np0      UP
fe80::a288:c2ff:fee4:4322/64
enp220s0np0      UP
fe80::a288:c2ff:feda:d6e2/64
bond0            UP             10.45.169.109/24
fe80::9e63:c0ff:fef6:1eb2/64
```

Interfaces are on the same VLAN =>

```
None
# lldpctl | grep -i VLAN
  VLAN:         108, pvid: yes vlan-108
  VLAN:         108, pvid: yes vlan-108
  VLAN:         108, pvid: yes vlan-108
  VLAN:         108, pvid: yes vlan-108
  VLAN:         108, pvid: yes vlan-108
  VLAN:         108, pvid: yes vlan-108
  VLAN:         108, pvid: yes vlan-108
  VLAN:         108, pvid: yes vlan-108
  VLAN:         108, pvid: yes vlan-108
  VLAN:         108, pvid: yes vlan-108
  VLAN:         108, pvid: yes vlan-108
  VLAN:         108, pvid: yes vlan-108
```

Interfaces are ordered correctly on leaf switches (compare output on both nodes - order must match) =>

```
None
# lshw -c network -businfo | grep X-7 | awk '{print $2}' |
xargs lldpctl | grep SysName
    SysName:      am4-t0-g1-leaf07
    SysName:      am4-t0-g1-leaf04
    SysName:      am4-t0-g1-leaf03
    SysName:      am4-t0-g1-leaf08
    SysName:      am4-t0-g1-leaf06
    SysName:      am4-t0-g1-leaf05
    SysName:      am4-t0-g1-leaf01
    SysName:      am4-t0-g1-leaf02
```

If Test Hangs

Add debug info:

```Shell
# Add to your multi-node script
mpirun --bind-to none --mca pml ucx --allow-run-as-root \
        -x NCCL_CROSS_NIC=0 \
        -x NCCL_DEBUG=INFO \
        -np $NEW_NP -H ${NEW_HOSTS} \
        ./build/all_reduce_perf -b 1G -f 2 -e 1G
```

Look for messages like:

- **Good**: `via NET/IB/1/GDRDMA` (RDMA working)
- **Bad**: `via NET/IB/1` (falling back to system memory)

Expected Output:

```None
atl1g3r4u9gpu:146600:146782 [5] transport/net_ib.cc:2244 NCCL WARN NET/IB: Got
completion from peer 10.46.70.21<50854> with status=5 opcode=31245 len=0 vendor
```

```
err 249 (Recv) localGid fe80::9e63:c0ff:feab:b8da
remoteGidsfe80::9e63:c0ff:feab:b9ca hca mlx5_9
atl1g3r4u9gpu:146600:146782 [5] NCCL INFO transport/net.cc:1362 -> 6
atl1g3r4u9gpu:146600:146782 [5] NCCL INFO proxy.cc:728 -> 6
atl1g3r4u9gpu:146600:146782 [5] NCCL INFO proxy.cc:912 -> 6 [Progress Thread]

atl1g3r1u16gpu:496000:496186 [6] transport/net_ib.cc:2244 NCCL WARN NET/IB: Got
completion from peer 10.46.70.22<39226> with status=12 opcode=32224 len=0
vendor err 129 (Recv) localGid fe80::9e63:c0ff:feab:938a
remoteGidsfe80::9e63:c0ff:feab:b6b2 hca mlx5_10
atl1g3r1u16gpu:496000:496186 [6] NCCL INFO transport/net.cc:1362 -> 6
atl1g3r1u16gpu:496000:496186 [6] NCCL INFO proxy.cc:728 -> 6
atl1g3r1u16gpu:496000:496186 [6] NCCL INFO proxy.cc:912 -> 6 [Progress Thread]
```

They indicate that pairs of nodes(Node-A and Node-B), in this case `10.46.70.21` and `10.46.70.22`, are having communication issue over the GPU fabric. This can be verified with using ping6 from Node-A:

```
None
 ping6 -c 2 -I enp156s0np0 fe80::9e63:c0ff:feab:b9ca
```

Where `enp156s0np0` is one of the 8 400G Nic, and `fe80::9e63:c0ff:feab:b9ca` the IPv6 address of the same NIC on Node-B.

Slow bus bandwidth

In this case, you need to isolate the culprit. The most obvious thing to rule out is software misconfiguration. What happens most often than not is that the GPU ends up talking to the NIC off system memory, so RDMA is not place. In other words you would need to see RDMA in-place and nccl tells you so in this form when you pass `-x NCCL_DEBUG=info` to its arguments:

```
None
      [...]
```

```
    am4g1r34bm1:23858:23875 [0] NCCL INFO Channel 00/0 :
    0[6] -> 1[6] [send] via NET/IB/1/GDRDMA

    am4g1r34bm1:23858:23875 [0] NCCL INFO Channel 01/0 :
    0[6] -> 1[6] [send] via NET/IB/1/GDRDMA

    am4g1r34bm1:23858:23875 [0] NCCL INFO Channel 02/0 :
    0[6] -> 1[6] [send] via NET/IB/1/GDRDMA

    am4g1r34bm1:23858:23875 [0] NCCL INFO Channel 03/0 :
    0[6] -> 1[6] [send] via NET/IB/1/GDRDMA
```

When it does not, the output is more like this one:

```
None
    [...]

    am4g1r35bm1:1226446:1226462 [0] NCCL INFO Channel 00/0
    : 0[6] -> 1[6] [receive] via NET/IB/1

    am4g1r35bm1:1226446:1226462 [0] NCCL INFO Channel 01/0
    : 0[6] -> 1[6] [receive] via NET/IB/1

    am4g1r35bm1:1226446:1226462 [0] NCCL INFO Channel 00/0
    : 1[6] -> 0[6] [send] via NET/IB/1

    am4g1r35bm1:1226446:1226462 [0] NCCL INFO Channel 01/0
    : 1[6] -> 0[6] [send] via NET/IB/1
```

And this is bad. In this case check that all the nvidia drivers are loaded correctly:

```
None
    # lsmod | grep nvid

    nvidia_uvm           4956160  0
```

```
nvidia_peermem         16384  0

ib_uverbs             200704  3
nvidia_peermem,rdma_ucm,mlx5_ib

nvidia_drm            122880  0

nvidia_modeset       1355776  1 nvidia_drm

nvidia              54292480  63
nvidia_uvm,nvidia_peermem,nvidia_modeset

video                  73728  2
dell_wmi,nvidia_modeset
```

`nvidia_peermem` is the module in question. If that is not loaded, than you probably need to rebuild it and install it again. Check what you have:

```None
# dkms status | grep nv

nvidia/550.90.07, 6.8.0-52-generic, x86_64: installed
```

Then do:

```None
# Remove nvidia driver

dkms remove nvidia/550.90.07



# Install nvidia driver

dkms install nvidia/550.90.07
```

Once that is done, `modprobe` module `nvidia_peermem` in and verify that is loaded correctly. You can attempt now to perform a new test, if it is still not in the clear, then performance slowness could be the result of the following:

- bad optics
- bad fiber
- bad NIC (if a card is replaced, remember to reset ETH_MODE and ATS attributes, check the appendix for more details)
- bad GPU

Check NVIDIA Drivers

```shell
# Verify nvidia_peermem is loaded (required for RDMA)
lsmod | grep nvidia_peermem

# If missing, reload drivers
sudo dkms remove nvidia/$(nvidia-smi --query-gpu=driver_version
--format=csv,noheader,nounits | head -1)
sudo dkms install nvidia/$(nvidia-smi --query-gpu=driver_version
--format=csv,noheader,nounits | head -1)
sudo modprobe nvidia_peermem
```

Test Individual GPU-NIC Pairs

If you get low bandwidth, test each GPU-NIC combination:

```shell
cat > ~/test_gpu_nic_pair.sh << 'EOF'
#!/bin/bash

# Usage: ./test_gpu_nic_pair.sh "ip1,ip2" gpu_id nic_id
HOSTS=$1
GPU_ID=$2
NIC_ID=$3
```

```
NEW_HOSTS=$(echo $HOSTS | sed 's/\([0-9\.]*\)/\1:1/g')

echo "Testing GPU$GPU_ID with NIC$NIC_ID"

cd ~/nccl-tests

mpirun --bind-to none --mca pml ucx --allow-run-as-root \
        -x NCCL_CROSS_NIC=0 \
        -x NCCL_DEBUG=info \
        -x CUDA_VISIBLE_DEVICES=$GPU_ID \
        -x NCCL_IB_HCA=mlx5_$NIC_ID \
        -np 2 -H ${NEW_HOSTS} \
        ./build/all_reduce_perf -b 1G -f 2 -e 1G
EOF

chmod +x ~/test_gpu_nic_pair.sh

# Test specific GPU-NIC pairs (based on nvidia-smi topo -m)
~/test_gpu_nic_pair.sh "ip1,ip2" 0 0
~/test_gpu_nic_pair.sh "ip1,ip2" 1 3
# etc.
```

Expected single GPU-NIC bandwidth: ~48 GB/s

Network Verification

Check Interface Status

Shell
```
# Check interfaces are up
ip -br a

# Check VLAN (all interfaces should be on same VLAN)
lldpctl | grep -i VLAN
```

```
# Check switch connectivity
lshw -c network -businfo | grep ConnectX | awk '{print $2}' |
xargs lldpctl | grep SysName
```

Test Network Connectivity

Shell

```
# Ping between nodes using RoCE interfaces
ping6 -c 2 -I enp156s0np0 fe80::target_ipv6_address
```