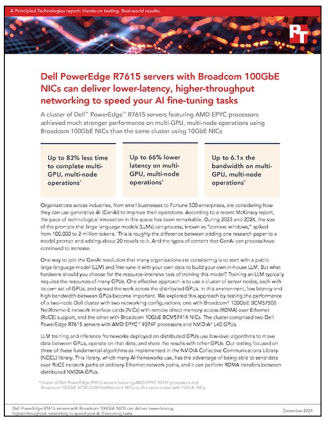
Dell PowerEdge R7615 servers with Broadcom 100GbE NICs can deliver lower-latency, higher-throughput networking to speed your AI fine-tuning tasks

A cluster of Dell™ PowerEdge™ R7615 servers featuring AMD EPYC processors achieved much stronger performance on multi-GPU, multi-node operations using Broadcom 100GbE NICs than the same cluster using 10GbE NICs

Up to 83% less time to complete multi-GPU, multi-node operations*

Up to 66% lower latency on multi-GPU, multi-node operations*

Up to 6.1x the bandwidth on multi-GPU, multi-node operations*

Organizations across industries, from small businesses to Fortune 500 enterprises, are considering how they can use generative AI (GenAI) to improve their operations. According to a recent McKinsey report, the pace of technological innovation in this space has been remarkable. During 2023 and 2024, the size of the prompts that large language models (LLMs) can process, known as "context windows," spiked from 100,000 to 2 million tokens. This is roughly the difference between adding one research paper to a model prompt and adding about 20 novels to it. And the types of content that GenAI can process have continued to increase.

One way to join the GenAI revolution that many organizations are considering is to start with a public large language model (LLM) and fine-tune it with your own data to build your own in-house LLM. But what hardware should you choose for the resource-intensive task of training this model? Training an LLM typically requires the resources of many GPUs. One effective approach is to use a cluster of server nodes, each with its own set of GPUs, and spread the work across the distributed GPUs. In this environment, low latency and high bandwidth between GPUs become important. We explored this approach by testing the performance of a two-node Dell cluster with two networking configurations: one with Broadcom® 100GbE BCM57508 NetXtreme-E network interface cards (NICs) with remote direct memory access (RDMA) over Ethernet (RoCE) support, and the other with Broadcom 10GbE BCM57414 NICs. The cluster comprised two Dell PowerEdge R7615 servers with AMD EPYC™ 9274F processors and NVIDIA® L40 GPUs.

LLM training and inference frameworks deployed on distributed GPUs use low-level algorithms to move data between GPUs, operate on that data, and share the results with other GPUs. Our testing focused on three of these fundamental algorithms as implemented in the NVIDIA Collective Communications Library (NCCL) library. This library, which many AI frameworks use, has the advantage of being able to send data over RoCE network paths or ordinary Ethernet network paths, and it can perform RDMA transfers between distributed NVIDIA GPUs.

*Cluster of Dell PowerEdge R7615 servers featuring AMD EPYC 9274F processors and Broadcom 100GbE NCM57508 NetXtreme-E NICs vs. the same cluster with 10GbE NICs.

Dell PowerEdge R7615 servers with Broadcom 100GbE NICS can deliver lower-latency, higher-throughput networking to speed your AI fine-tuning tasks

December 2024

The science behind the report:

Dell PowerEdge R7615 servers with Broadcom 100GbE NICS can deliver lower-latency, higher-throughput networking to speed your AI fine-tuning tasks

This document describes what we tested, how we tested, and what we found. To learn how these facts translate into real-world benefits, read the report Dell PowerEdge R7615 servers with Broadcom 100GbE NICS can deliver lower-latency, higher-throughput networking to speed your AI fine-tuning tasks.

We concluded our hands-on testing on November 6, 2024. During testing, we determined the appropriate hardware and software configurations and applied updates as they became available. The results in this report reflect configurations that we finalized on October 30, 2024 or earlier. Unavoidably, these configurations may not represent the latest versions available when this report appears.

Our results

To learn more about how we have calculated the wins in this report, go to http://facts.pt/calculating-and-highlighting-wins. Unless we state otherwise, we have followed the rules and principles we outline in that document.

Table 4: Performance of all-reduce multi-GPU, multi-node task in terms of time in microseconds to complete the task on datasets of multiple sizes. Lower is better. Source: Principled Technologies.

All-reduce operation			
Data size (B)	100GbE configuration Time to complete task (microseconds)	10GbE configuration Time to complete task (microseconds)	Relative difference
4	40.58	126.55	211.85%
8	52.45	122.90	134.34%
16	40.69	123.30	203.06%
32	63.34	123.65	95.22%
64	52.49	123.55	135.40%
128	64.68	123.90	91.57%
256	64.95	125.35	92.99%
512	42.44	129.00	203.99%
1,024	43.41	135.60	212.37%

All-reduce operation			
Data size (B)	100GbE configuration Time to complete task (microseconds)	10GbE configuration Time to complete task (microseconds)	Relative difference
2,048	55.60	190.05	241.85%
4,096	57.97	199.75	244.57%
8,192	49.21	209.60	325.93%
16,384	56.49	240.55	325.87%
32,768	84.37	524.55	521.76%
65,536	95.92	655.40	583.28%
131,072	131.10	836.25	537.87%
262,144	259.20	1,070.35	312.94%
524,288	680.90	1,275.50	87.33%
1,048,576	435.70	1,995.65	358.03%
2,097,152	775.15	3,083.10	297.74%
4,194,304	1,525.30	5,704.85	274.01%
8,388,608	2,837.25	10,997.50	287.61%
16,777,216	5,650.00	30,570.50	441.07%
33,554,432	11,261.00	60,114.50	433.83%
67,108,864	22,605.00	127,105.00	462.29%
134,217,728	45,005.50	253,713.50	463.74%
268,435,456	89,873.00	513,461.50	471.32%

Table 5: Bandwidth achieved for multi-GPU, multi-node all-reduce task. Higher is better. We did not calculate the relative difference for the first eight data sizes because the tool's precision was too low to make a meaningful comparison. Source: Principled Technologies.

Operational bandwidth for the all-reduce task			
Data size (B)	100GbE configuration Operational bandwidth (Gbps)	10GbE configuration Operational bandwidth (Gbps)	Relative difference
4	0.00	0.00	
8	0.00	0.00	
16	0.00	0.00	
32	0.00	0.00	

Operational bandwidth for the all-reduce task			
Data size (B)	100GbE configuration Operational bandwidth (Gbps)	10GbE configuration Operational bandwidth (Gbps)	Relative difference
64	0.00	0.00	
128	0.00	0.00	
256	0.00	0.00	
512	0.08	0.00	
1,024	0.16	0.08	100.00%
2,048	0.32	0.08	300.00%
4,096	0.60	0.16	275.00%
8,192	1.36	0.32	325.00%
16,384	2.32	0.56	314.29%
32,768	3.08	0.48	541.67%
65,536	5.44	0.80	580.00%
131,072	8.00	1.24	545.16%
262,144	8.08	1.96	312.24%
524,288	6.20	3.32	86.75%
1,048,576	19.24	4.24	353.77%
2,097,152	21.64	5.48	294.89%
4,194,304	22.08	5.88	275.51%
8,388,608	23.64	6.12	286.27%
16,777,216	23.76	4.40	440.00%
33,554,432	23.84	4.48	432.14%
67,108,864	23.72	4.24	459.43%
134,217,728	23.88	4.24	463.21%
268,435,456	23.92	4.16	575.00%

Table 6: Performance of reduce-scatter multi-GPU, multi-node task in terms of time in microseconds to complete the task on datasets of multiple sizes. Lower is better. Source: Principled Technologies.

Reduce-scatter operation			
Data size (B)	100GbE configuration Time to complete task (microseconds)	10GbE configuration Time to complete task (microseconds)	Relative difference
48	28.64	86.85	203.25%
96	28.60	88.31	208.81%

Reduce-scatter operation			
Data size (B)	100GbE configuration Time to complete task (microseconds)	10GbE configuration Time to complete task (microseconds)	Relative difference
240	40.31	84.68	110.09%
480	39.33	88.33	124.60%
1,008	28.55	89.11	212.12%
2,016	27.18	88.78	226.70%
4,080	27.18	97.14	257.46%
8,160	28.89	99.05	242.91%
16,368	46.08	112.45	144.03%
32,736	49.64	234.65	372.70%
65,520	59.97	323.10	438.81%
131,040	76.13	361.20	374.48%
262,128	133.55	635.80	376.08%
524,256	140.05	639.60	356.69%
1,048,560	323.05	1,116.10	245.49%
2,097,120	411.70	1,639.50	298.23%
4,194,288	984.05	2,767.15	181.20%
8,388,576	1,457.65	6,537.90	348.52%
16,777,200	2,898.85	15,876.50	447.68%
33,554,400	5,755.25	30,338.50	427.14%
67,108,848	11,588.50	64,431.00	455.99%
134,217,696	22,937.00	127,747.00	456.95%
268,435,440	45,853.00	259,777.00	466.54%

Table 7: Bandwidth achieved for multi-GPU, multi-node reduce-scatter task. Higher is better. Note that the operational bandwidth at 4MB for the 10G network actually exceeds 10Gbps. For this packet size, the speed and amount of data transferred between GPUs on one node contributed more to the operational bandwidth than that for internode data transfers. We did not calculate the relative difference for the first three data sizes because the tool's precision was too low to make a meaningful comparison Source: Principled Technologies.

Operational bandwidth for the reduce-scatter task			
Data size (B)	100GbE configuration Operational bandwidth (Gbps)	10GbE configuration Operational bandwidth (Gbps)	Relative difference
48	0.00	0.00	
96	0.00	0.00	

Operational bandwidth for the reduce-scatter task			
Data size (B)	100GbE configuration Operational bandwidth (Gbps)	10GbE configuration Operational bandwidth (Gbps)	Relative difference
240	0.04	0.00	
480	0.12	0.08	50.00%
1,008	0.28	0.08	250.00%
2,016	0.60	0.16	275.00%
4,080	1.20	0.32	275.00%
8,160	2.24	0.64	250.00%
16,368	3.12	1.16	168.97%
32,736	5.28	1.12	371.43%
65,520	8.76	1.60	447.50%
131,040	13.80	2.92	372.60%
262,128	15.68	3.28	378.05%
524,256	29.92	6.68	347.90%
1,048,560	28.36	7.52	277.13%
2,097,120	40.76	10.24	298.05%
4,194,288	34.08	12.16	180.26%
8,388,576	46.04	10.28	347.86%
16,777,200	46.28	8.44	448.34%
33,554,400	46.64	8.84	427.60%
67,108,848	46.32	8.36	454.07%
134,217,696	46.80	8.40	457.14%
268,435,440	46.84	8.24	468.45%

Table 8: Performance of send-receive multi-GPU, multi-node task in terms of time in microseconds to complete the task on datasets of multiple sizes. Lower is better. Source: Principled Technologies.

Send-receive operation			
Data size (B)	100GbE configuration Time to complete task (microseconds)	10GbE configuration Time to complete task (microseconds)	Relative difference
4	41.05	57.75	40.68%
8	52.80	55.78	5.65%
16	41.05	56.66	38.03%
32	41.04	58.13	41.66%

Send-receive operation			
Data size (B)	100GbE configuration Time to complete task (microseconds)	10GbE configuration Time to complete task (microseconds)	Relative difference
64	60.44	55.05	-8.92%
128	40.82	57.87	41.77%
256	41.61	55.98	34.55%
512	42.05	54.83	30.41%
1,024	49.83	57.25	14.89%
2,048	41.96	60.71	44.67%
4,096	41.24	61.35	48.75%
8,192	52.56	66.13	25.81%
16,384	42.14	75.06	78.12%
32,768	45.33	96.38	112.64%
65,536	55.68	181.45	225.88%
131,072	70.68	335.55	374.75%
262,144	106.95	620.60	480.27%
524,288	181.70	958.65	427.60%
1,048,576	432.05	1,495.80	246.21%
2,097,152	691.45	2,772.70	301.00%
4,194,304	1,120.10	5,521.05	392.91%
8,388,608	2,167.80	11,569.50	433.70%
16,777,216	4,347.95	23,473.00	439.86%
33,554,432	8,557.95	49,222.50	475.17%
67,108,864	16,834.00	102,962.00	511.63%
134,217,728	33,593.00	214,140.50	537.46%
268,435,456	67,113.00	433,726.50	546.26%

Table 9: Bandwidth available for multi-GPU, multinode send-receive task. Higher is better. We did not calculate the relative difference for the first seven data sizes because the tool's precision was too low to make a meaningful comparison. Source: Principled Technologies.

Operational bandwidth for the send-receive task			
Data size (B)	100GbE configuration Operational bandwidth (Gbps)	10GbE configuration Operational bandwidth (Gbps)	Relative difference
4	0.00	0.00	
8	0.00	0.00	
16	0.00	0.00	

Operational bandwidth for the send-receive task			
Data size (B)	100GbE configuration Operational bandwidth (Gbps)	10GbE configuration Operational bandwidth (Gbps)	Relative difference
32	0.00	0.00	
64	0.00	0.00	
128	0.00	0.00	
256	0.08	0.00	
512	0.08	0.08	0.00%
1,024	0.16	0.16	0.00%
2,048	0.40	0.24	66.67%
4,096	0.80	0.52	53.85%
8,192	1.28	1.00	28.00%
16,384	3.08	1.72	79.07%
32,768	5.80	2.72	113.24%
65,536	9.40	2.88	226.39%
131,072	14.84	3.16	369.62%
262,144	19.64	3.44	470.93%
524,288	23.08	4.44	419.82%
1,048,576	21.04	5.60	275.71%
2,097,152	24.88	6.04	311.92%
4,194,304	29.96	6.08	392.76%
8,388,608	30.96	5.80	433.79%
16,777,216	30.88	5.72	439.86%
33,554,432	31.36	5.44	476.47%
67,108,864	31.92	5.20	513.85%
134,217,728	32.00	5.00	540.00%
268,435,456	32.00	4.96	545.16%

System configuration information

Table 10: Detailed information on the systems we tested.

System configuration information	Dell PowerEdge R7615 server 1	Dell PowerEdge R7615 server 1
BIOS name and version	Dell 1.8.3	Dell 1.8.3
Non-default BIOS settings	Logical processor disabled	Logical processor disabled
Operating system name and version/build number	Ubuntu 22.04.5 LTS	Ubuntu 22.04.5 LTS
Date of last OS updates/patches applied	10/30/24	10/30/24
Power management policy	Performance	Performance
Processor		
Number of processors	1	1
Vendor and model	AMD EPYC™ 9374F	AMD EPYC 9374F
Core count (per processor)	32	32
Core frequency (GHz)	3.85	3.85
Stepping	1	1
Memory module(s)		
Total memory in system (GB)	192	192
Number of memory modules	12	12
Vendor and model	Hynix® HMCG78MEBRA107N	Hynix HMCG78MEBRA107N
Size (GB)	16	16
Type	PC5-38400	PC5-38400
Speed (MHz)	4,800	4,800
Speed running in the server (MHz)	4,800	4,800
Storage controller		
Vendor and model	Dell PERC H755	BOSS-N1 Monolithic
Cache size (GB)	8	N/A
Firmware version	52.26.0-5179	2.1.13.2025
Driver version	N/A	N/A
Local storage (type A)		
Number of drives	6	2
Drive vendor and model	Kioxia® KPM6XRUG 960 GB	Dell EC NVMe ISE 7400 RI M.2 960 GB
Drive size (GB)	960	960
Drive information (speed, interface, type)	12 Gbps, SAS, SSD	M.2, PCIe, SSD
Purpose	OS, application	OS

System configuration information	Dell PowerEdge R7615 server 1	Dell PowerEdge R7615 server 1
Local storage (type B)		
Number of drives	N/A	4
Drive vendor and model	N/A	Dell Ent NVMe® CM6 MU 6.4TB
Drive size (GB)	N/A	6,400
Drive information (speed, interface, type)	N/A	16GT/s, NVMe, SSD
Purpose	N/A	Applications
Network adapter (A)		
Vendor and model	Broadcom® BCM57508 NetXtreme-E 10Gb/25Gb/40Gb/50Gb/100Gb/200Gb	Broadcom BCM57508 NetXtreme-E 10Gb/25Gb/40Gb/50Gb/100Gb/200Gb
Number and type of ports	2 x 100GbE	2 x 100GbE
Driver version	bnxt_en 1.10.3-231.0.162.0 bnxt_re 231.0.162.0	bnxt_en 1.10.3-231.0.162.0 bnxt_re 231.0.162.0
Network adapter (B)		
Vendor and model	Broadcom BCM57414 NetXtreme-E 10Gb/25Gb	Broadcom BCM57414 NetXtreme-E 10Gb/25Gb
Number and type of ports	2 x 10GbE	2 x 10GbE
Driver version	bnxt_en 1.10.3-231.0.162.0	bnxt_en 1.10.3-231.0.162.0
Network adapter (C)		
Vendor and model	Broadcom BCM5720 Gigabit Ethernet	Broadcom BCM5720 Gigabit Ethernet
Number and type of ports	2 x 1GbE	2 x 1GbE
Driver version	tg3 6.8.0-45-generic	tg3 6.8.0-45-generic
Cooling fans		
Vendor and model	Dell Gold	Dell Gold
Number of cooling fans	12	12
Power supplies		
Vendor and model	Dell 06C11WA028	Dell 008PMKA00
Number of power supplies	2	2
Wattage of each (W)	1,400	2,800

How we tested

Unless otherwise noted, perform the following configuration and installation steps on both servers.

Configuring BIOS settings and creating storage volumes for the systems' OS

1. Log into the iDRAC.
2. Select Configuration, and Configure BIOS. From the pull-down menu:
    a. Reset all settings to their default.
    b. Set the power profile to Performance.
    c. Disable Simultaneous Multi-Threading (SMT).
3. Skip the following steps for the server with NVME SSDs: Select Configuration, Storage Configuration, and Select the system's controller.
4. Select Virtual Disk Configuration, and Create Virtual Disk.
5. Select the physical disks, and choose RAID 60.
6. Click Apply Now.

Installing Ubuntu Server 22.04 LTS

1. Log into iDRAC.
2. Assign the Ubuntu Server 22.04 installation ISO to the virtual media.
3. Boot the system from the virtual ISO.
4. At the Grub screen, select Try or Install Ubuntu Server, and press enter.
5. Select English for the installation, and press enter.
6. Choose English (US) for the keyboard layout, click Done.
7. Select Ubuntu Server for the base installation, click Done.
8. On the network connections screen, select the system's management NIC, and enter its IP address. Click Done.
9. On the Configure Proxy screen, click Done.
10. On the Configure Ubuntu archive mirror screen, click Done.
11. On the Guide storage configuration screen, select Use an entire disk, select the system's OS volume, select Set up the disk as an LVM group, and click Done.
12. On the Storage configuration screen, assign all free space to the default volume group, and click Done.
13. On the Confirm destructive action pop-up screen, click Continue.
14. On the profile setup screen, enter `ubuntu` for Your name, `host-01` or `host-02` for Your server's name, `ubuntu` for Pick a username, and enter a password. Click Done.
15. On the SSH Setup screen, select Install OpenSSH server, and click Done.
16. On the Featured Server Snaps screen, click Done to start the installation.
17. When the installation completes, click Reboot Now.
18. After the system reboots, login as user ubuntu.
19. Update the system packages:

```
sudo apt update
sudo apt upgrade
```

20. Configure the 10GbE and 100GbE NICs by editing the file /etc/netplsn/00-installer-config.yaml. Set the IP Address, netmask, and set the MTU to 9,000.
21. Activate this network configuration:

```
sudo netplan apply
```

22. Disable iommu for better Broadcom NIC performance:
    a. Edit the file /etc/default/grub, and add `iommu=off` to the end of the kernel line.
    b. Regenerate the grub menu:

```
sudo update_grub
```

23. Install additional packages:

```
sudo apt install net-tools build-essential linux-headers-$(uname -r) \
 linux-tools-common linux-tools-6.5.0-45-generic ibverbs-utils \
 libibverbs-dev librdmacm-dev libibumad-dev rdmacm-utils t libpci-dev \
 python3 pythin3-pip openmpi-bin gfortran git tmux
```

24. Set-up passwordless SSH between the servers. Perform the following steps on only one server:

```
# set the variable ipadd to the IP address of the second server
ipadd=<IP ADDRESS of SECOND SERVER>
ssh-keygen -t ecdsa -b 521
ssh-copy-id -i .ssh/id_ecdsa.pub ${ipadd}
scp .ssh/id_ecdsa $(ipadd}:.ssh/
```

25. Reboot the server:

```
sudo shutdown -r now
```

Installing NVIDIA CUDA libraries

1. Download the local installer:

```
cd
wget "https://developer.download.nvidia.com/compute/cuda/12.6.0/local_installers/
cuda_12.6.0_560.28.03_linux.run"
```

2. Start the installer with the option to select the "open flavor" of the kernel modules:

```
sudo sh ./cuda_12.6.0_560.28.03_linux.run -m=kernel-open
```

3. Type `accept` to accept the EULA.
4. On the CUDA Installer page, make sure the CUDA Toolkit12.6 option is selected.
5. Move the cursor to Install, and press enter.

Updating the Broadcom modules and configuring the BCM5708 NIC

1. Download the Broadcom driver and software package bcm_231.1.162.1b.
2. Extract files from the archive:

```
cd ~
tar -xf bcm_231.1.162.1b.tar.gz
```

3. Update the driver for port 1 of the BCM57508 NIC:

```
cd bcm_231.1.162.1b/linux_installer
sudo bash install.sh -i enp129s0f0np0 -w -g
```

4. Apply the Broadcom suggested tunings to all Broadcom NICs:

```
cd ../utils/nictune
sudo ./nictune -t -i
```

5. Enable RoCE on port 1 of the BCM57508 NIC, reboot the server, and log back in:

```
sudo niccli -i 1 nvm -setoption support_rdma -scope 0 -value 1
sudo niccli -i 1 reset
sudo shutdown -r now
```

Installing the Linux drivers supporting GPUDirect RDMA for Broadcom NICs

These instructions follow closely the steps in document bcm_231.1.162.1b/drivers_linux/readme_peer_mem.txt.

1. Check out the NVIDIA code for the module:

```
cd ~
git clone https://github.com/NVIDIA/open-gpu-kernel-modules
cd open-gpu-kernel-modules
git checkout 560
```

2. Patch this code so that it works with Broadcom NICs. We modified the original Broadcom patch to work with this kernel. The modified Broadcom patch is in file `modded_gpudirectbuild.patch` in the Scripts and files we used in our testing section.

```
patch -b -p1 < ~/modded_gpudirectbuild.patch
```

3. Compile and install the module:

```
export BNXT_PEER_MEM_INC="~/bcm_231.1.162.1b/drivers_linux/peer_mem/netxtreme-
peermem-231.0.162.0/peer_mem"
make modules -j$(nproc)
sudo make modules_install -j$(nproc)
```

4. Reboot the server:

```
shutdown -r now
```

Installing system resource and power-measurement tools

NVIDIA and system resource capturing script

1. Install tools to measure NVIDIA GPU resource usage:

```
mkdir -p ~/venv/measure
python3 -m venv --upgrade-deps ~/venv/measure
. ~/venv/measure/bin/activate
pip install nvitop pandas
```

2. Create the script nvistats.py (in the Scripts and files we used in our testing section) to report system and GPU resource usage into file `results.csv`.
3. Install impi tools for server power measurement

```
sudo apt install ipmitool
sudo systemctl restart openipmi
```

4. Create the script `power.sh` (in the Scripts and files we used in our testing section) to measure server power with output in CSV format.

Installing NCCL and MPI libraries and tools

```
sudo apt install gfortran openmpi-bin libopenmpi-dev
wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/sbsa/cuda-keyring_1.1-1_all.deb
sudo dpkg -i cuda-keyring_1.1-1_all.deb
sudo apy update
sudo apt install libnccl2=2.23.4-1+cuda12.6 libnccl-dev=2.23.4-1+cuda12.6
```

Configuring the switch for testing

1. Log onto the PowerSwitch as admin and set the global configuration:

```
conf
dcbx enable
wred wred_ecn
 random-detect color green minimum-threshold 1000 maximum-threshold 2000 drop-probability 100
 random-detect color yellow minimum-threshold 500 maximum-threshold 1000 drop-probability 100
 random-detect color red minimum-threshold 100 maximum-threshold 500 drop-probability 100
 random-detect ecn
 exit
class-map type network-qos pfcdot1p3
 match qos-group 3
 exit
class-map type queuing Q0
 match queue 0
 exit
class-map type queuing Q3
 match queue 3
 exit
class-map type queuing Q7
 match queue 7
 exit
trust dscp-map rDSCP
 qos-group 7 dscp 48
 qos-group 3 dscp 26
 exit
qos-map traffic-class 2Q
 queue 0 qos-group 0-2,4-6
 queue 3 qos-group 3
 queue 7 qos-group 7
 exit
policy-map type network-qos policy_pfcdot1p3
 class pfcdot1p3
  pause
  pfc-cos 3
 exit
policy-map type queuing policy_2Q
 class Q0
  bandwidth percent 30
 class Q3
  bandwidth percent 70
  random-detect wred_ecn
  exit
system qos
  buffer-statistics-tracking
  exit
```

2. Configure 10GbE network as admin on the PowerSwitch:

   a. Create VLAN 10:

```
conf
int vlan10
no shutdown
mtu 9216
exit
```

   b. Configure interfaces:

```
int ethernet1/1/13:2
 no shutdown
 switchport access vlan 20
 flowcontrol receive off
 exit
```

```
int ethernet1/1/13:4
 no shutdown
 switchport access vlan 20
 flowcontrol receive off
 exit
```

3. Configure 100GbE network as admin on the PowerSwitch:

   a. Create VLAN 10:

```
conf
int vlan10
 no shutdown
 mtu 9216
 exit
```

   b. Configure interfaces:

```
int ethernet1/1/9
 no shutdown
 switchport access vlan 10
 mtu 9216
 flowcontrol receive off
 flowcontrol transmit off
 priority-flow-control mode on
 service-policy input type network-qos policy_pfcdot1p3
 service-policy output type queuing policy_2Q
 ets mode on
 qos-map traffic-class 2Q
 trust-map dscp rDSCP
 exit
int ethernet1/1/10
 no shutdown
 switchport access vlan 10
 mtu 9216
 flowcontrol receive off
 flowcontrol transmit off
 priority-flow-control mode on
 service-policy input type network-qos policy_pfcdot1p3
 service-policy output type queuing policy_2Q
 ets mode on
 qos-map traffic-class 2Q
 trust-map dscp rDSCP
 exit
```

Testing

On each server, the file `/etc/hosts` contains the IP addresses for each server's management link, 10GbE link, and 100GbEl ink. Here, the names associated with these IP addresses are host-01, host-01; host-01t, host-02t; host-01h, and host-02h, respectively.

The names of the RoCE ports associated with the 100GbE links are the same on each server: rocep129s0f0.

1. Compile NCCL test tools:

```
cd /home/ubuntu
git clone https://github.com/NVIDIA/nccl-tests.git
cd nccl-tests
make MPI_HOME=/usr/lib/x86_64-linux-gnu/openmpi MPI=1 -j 5
```

2. Reboot the server before each run.
3. Start tmux:

```
tmux
```

4. Start the system and GPU resource collector:

```
. ~/venv/measure/bin/activate
python nvistats.py
```

5. Press ctrl-a ctrl-n to create a new windows, and start the power collector:

```
sudo bash power.sh
```

6. Press ctrl-a ctrl-n to create a new window to run the test script.
7. Run the test script `runTest.sh` (see the Scripts and files we used in our testing section) with options to use the 10GbE (10) or 100GbE with RoCE (100) network, and to use the all_reduce_perf, sendrecv_perf, or reduce_scatter_perf test. For example, to test the 100GbE network with the all_reduce_perf test:

```
bash ./power.sh 100 all-reduce
```

Scripts and files we used in our testing

Contents of file modded_gpudirectbuild.patch

```
diff --git a/kernel-open/conftest.sh b/kernel-open/conftest.sh
index 1226cea2..d16dcd4c 100755
--- a/kernel-open/conftest.sh
+++ b/kernel-open/conftest.sh
@@ -5327,7 +5327,8 @@ compile_test() {
                check_for_ib_peer_memory_symbols "$MLNX_OFED_KERNEL_DIR/$ARCH/$KERNELRELEASE" || \
                check_for_ib_peer_memory_symbols "$MLNX_OFED_KERNEL_DIR/$KERNELRELEASE" || \
                check_for_ib_peer_memory_symbols "$MLNX_OFED_KERNEL_DIR/default" || \
-               check_for_ib_peer_memory_symbols "$VAR_DKMS_SOURCES_DIR"; then
+               check_for_ib_peer_memory_symbols "$VAR_DKMS_SOURCES_DIR" || \
+               check_for_ib_peer_memory_symbols "$BNXT_PEER_MEM_INC" ; then
                 echo "#define NV_MLNX_IB_PEER_MEM_SYMBOLS_PRESENT" | append_conftest "symbols"
             else
                 echo "#undef NV_MLNX_IB_PEER_MEM_SYMBOLS_PRESENT" | append_conftest "symbols"
diff --git a/kernel-open/nvidia-peermem/nvidia-peermem.Kbuild b/kernel-open/nvidia-peermem/
nvidia-peermem.Kbuild
index d2bcaf8d..3bd0fdb1 100644
--- a/kernel-open/nvidia-peermem/nvidia-peermem.Kbuild
+++ b/kernel-open/nvidia-peermem/nvidia-peermem.Kbuild
@@ -42,18 +42,20 @@ else
 endif
 OFA_DIR := /usr/src/ofa_kernel
 OFA_CANDIDATES = $(OFA_DIR)/$(OFA_ARCH)/$(KERNELRELEASE) $(OFA_DIR)/$(KERNELRELEASE) $(OFA_DIR)/default
/var/lib/dkms/mlnx-ofed-kernel
-MLNX_OFED_KERNEL := $(shell for d in $(OFA_CANDIDATES); do \
-                             if [ -d "$$d" ]; then \
-                               echo "$$d"; \
-                               exit 0; \
-                             fi; \
-                           done; \
-                           echo $(OFA_DIR) \
-                     )
+#MLNX_OFED_KERNEL := $(shell for d in $(OFA_CANDIDATES); do \
+#                              if [ -d "$$d" ]; then \
+#                                echo "$$d"; \
+#                                exit 0; \
+#                              fi; \
```

```
+#                            done; \
+#                            echo $(OFA_DIR) \
+#                       )
+
+MLNX_OFED_KERNEL := $(shell echo /lib/modules/`uname -r`/build)

 ifneq ($(shell test -d $(MLNX_OFED_KERNEL) && echo "true" || echo "" ),)
     NVIDIA_PEERMEM_CFLAGS += -I$(MLNX_OFED_KERNEL)/include -I$(MLNX_OFED_KERNEL)/include/rdma
-    KBUILD_EXTRA_SYMBOLS := $(MLNX_OFED_KERNEL)/Module.symvers
+    KBUILD_EXTRA_SYMBOLS := $(BNXT_PEER_MEM_INC)/Module.symvers
 endif

 $(call ASSIGN_PER_OBJ_CFLAGS, $(NVIDIA_PEERMEM_OBJECTS), $(NVIDIA_PEERMEM_CFLAGS))
```

Contents of file nvistats.py

```
#!env python3
import datetime
import time
from nvitop import ResourceMetricCollector
import pandas as pd

csvfile = "results.csv"
# log all devices and all GPU processes
collector = ResourceMetricCollector(root_pids={1}, interval=1.0)
df = pd.DataFrame()

print("Looping forver. Press ^c to stop")
with collector(tag="resources"):
    try:
        while True:
            metrics = collector.collect()
            df_metrics = pd.DataFrame.from_records(metrics, index=[len(df)])
            df = pd.concat([df, df_metrics], ignore_index=True)
            time.sleep(1)
    except KeyboardInterrupt:
        print("Done. Writing results to file", csvfile)
        df.insert(0, "time", df["resources/timestamp"].map(datetime.datetime.fromtimestamp))
        df.to_csv(csvfile, index=False)
```

Contents of file power.sh

```
#!/bin/bash
# must run w/ sudo
if [ -z "$1" ]; then
  echo "usage: $0 counts"
  exit
fi
sample_count=$1
echo "Starting time: $(date)"
echo "Time, instantaneous power, one-second average"
for (( count=0 ; count < sample_count; count++ )); do
  stats="$(ipmitool dcmi power reading | \
    awk -F: '/power reading/ {printf "%s;",$2}; \
      /timestamp/ {printf "%s:%s:%s\n",$2,$3,$4}')"
  IFS=';' read -ra fields <<<"$stats"
```

```
  ins=${fields[0]}; ins=${ins#"${ins%%[![:space:]]*}"}
  ave=${fields[1]}; ave=${ave#"${ave%%[![:space:]]*}"}
  tim=${fields[2]}; tim=${tim#"${tim%%[![:space:]]*}"}
  printf '"%s", "%s", "%s"\n' "$tim" "$ins" "$ave"
  sleep 1
done
echo "Ending time: $(date)"
```

Contents of file runTest.sh

```
#!/bin/bash
# usage: speed testName
# tests: all_reduce_perf or sendrecv_perf or reduce_scatter_perf
# defaults: 100 all_reduce-perf

speed="${1:-100}"
testName="${2:-all_reduce-perf}"

case "$speed" in
  10)
    hosts="host-01t,host-02t"
    ifname="=enp130s0f1np1,eno12399np0"
    ibname=""
    roce="no"
    ;;
  100)
    hosts="host-01h,host-02h"
    ifname="enp129s0f0np0"
    ibname="rocep129s0f0"
    roce="yes"
    ;;
  *)
    echo "Unexpected speed '$speed'"
    echo "Options are 10 or 100"
    exit 1
esac

case "$testName" in
  all_reduce_perf)
    byteStart=4; byteStop=256M ;;
  sendrecv_perf)
    byteStart=4; byteStop=256M ;;
  reduce_scatter_perf)
    byteStart=24; byteStop=192M ;;
  *)
    echo "Unexpected test '$testName'"
    echo "Options are all_reduce_perf, sendrecv_perf, or reduce_scatter_perf"
    exit 2
esac

echo "Killing previous jobs, if any."
pkill -ef "$testName"
ssh host-02m pkill -ef "$testName"

debug="WARN"; N=100; NN=10
swDir="/home/ubuntu/nccl-tests/build"
echo "Run start @ $(date)"
echo "  from $byteStart to $byteStop with $N / $NN"
if [[ $roce != "yes" ]] ; then
 echo "$testName w/o RoCE on ${speed}G network"
 mpirun --allow-run-as-root -np 2 --host $hosts \
  -x LD_LIBRARY_PATH=/usr/local/cuda/lib64 \
  -x NCCL_IB_CUDA_SUPPORT=1 -x NCCL_DEBUG="$debug" \
  -x NCCL_IB_DISABLE=1 -x NCCL_SOCKET_IFNAME="$ifname" \
  -x NCCL_NET="Socket" -x NCCL_OOB_NETIFNAME="eno8303" \
  -x NCCL_NET_GDR_LEVEL=LOC -x NCCL_NET_GDR_READ=1 \
  -x NCCL_P2P_LEVEL=0 -x NCCL_SHM_DISABLE=1 \
  "$swDir/$testName" -b $byteStart -e $byteStop -f 2 -g 3 -n "$N" -w "$NN"
```

```
else
 echo "$testName w/ RoCE on ${speed}G network"
 mpirun --allow-run-as-root -np 2 --host $hosts \
  -x LD_LIBRARY_PATH=/usr/local/cuda/lib64 \
  -x NCCL_IB_CUDA_SUPPORT=1 -x NCCL_DEBUG="$debug" \
  -x NCCL_IB_DISABLE=0 -x NCCL_IB_HCA="$ibname" \
  -x NCCL_IB_GID_INDEX=3
  -x NCCL_NET="IB" -x NCCL_OOB_NETIFNAME="eno8303"
  -x NCCL_NET_GDR_LEVEL=SYS -x NCCL_NET_GDR_READ=1 \
  -x NCCL_P2P_LEVEL=0 -x NCCL_SHM_DISABLE=1 \
  "$swDir/$testName" -b $byteStart -e $byteStop -f 2 -g 3 -n "$N" -w "$NN"
fi
echo "Run end @ $(date)"
```

Read the report at https://facts.pt/QAauY1Y ▶

This project was commissioned by Dell Technologies.

Principled Technologies®

Facts matter.®