APPS620
Codensity Quadra™ P2P with NVIDIA GPU (CUDA) Application Note

V1.0

Table of Contents

1 Table of Abbreviations

Abbreviation:	Full Form:
AI	Artificial Intelligence
API	Application Programming Interface
AUD	Access Unit Delimiters
AV	Audio Video
AV1	Alliance Open Media Video 1 Codec
CABAC	Context Adaptive Binary Arithmetic Coding
CBR	Constant Bit Rate
CPU	Central Processing Unit
CRF	Constant Rate Factor
CUDA	Compute Unified Device Architecture
DSP	Digital Signal Processing
EOF	End Of File
FIXQB	Fixed Quantization Parameter
FF	Form Factor
FPS	Frames Per Second
GOP	Group of pictures
HDR	High Dynamic Range
HHHL	Half Height, Half Length
HHFL	Half Height, Full Length
HLG	Hybrid Log Gamma
HRD	Hypothetical Reference Decoder
IE	Inference Engine
Mbps	Mega Bit per second
MBps	Mega Byte per second
NLP	Natural Language Processing
NPU	Neural Processing Unit
NVMe	Non-Volatile Memory express
P2P	Peer-to-Peer
PCIe	Peripheral Component Interconnect express
PPS	Picture Parameter Set
QP	Quoted Printable
RGB	Red Green Blue
RGBA	Red Green Blue Alpha
RDO	Rate-Distortion Optimization
ROI	Region of interest
SDK	Software Development Kit
SEI	Supplemental Enhancement Information
SRIOV	Single Root I/O Virtualization
SPS	Sequence Parameter Set
VBR	Variable Bit Rat
VCL	Video Coding Layer
VPS	Video Parameter Set
VUI	Video Usability Information

2 References

Technical Notes:

APPS565-Codensity Quadra Scaler P2P Demo Application Note v1.0.pdf

https://docs.nvidia.com/datacenter/tesla/pdf/Driver_Installation_Guide.pdf

Buffer Sharing and Synchronization (dma-buf) — The Linux Kernel documentation

3 Background

This Application Note explains how to set up and run the peer-to-peer (P2P) demo from an NVIDIA GPU to a NETINT Quadra VPU. This note also covers installing the following dependencies

- NVIDIA open kernel drivers
- NVIDIA CUDA Toolkit
- NETINT libxcoder

Instructions on how to run the P2P read demo (**xcoderp2p_read_cuda**) will also be covered.

3.1 Intended Audience

This document is intended for application developers who want to use the P2P transfer feature from Nvidia GPU to Quadra VPU.

3.2 Compatibility

Software Compatibility:

This guide is intended to be used with NETINT Codensity Quadra Video Transcoder software Release 5.5 or newer.

Hardware Compatibility:

Release 5.5 or newer supports NETINT Codensity Quadra Video Transcoder hardware.

Operating System

Ubuntu 24.04 LTS or above.

This demo is designed to run on Linux systems only and is not supported on other operating systems such as Windows, Android, or macOS.

4 NVIDIA GPU to Quadra VPU P2P demo

4.1 Overview

PCIe Peer-to-Peer (P2P) allows two PCIe devices to exchange data directly over the PCIe fabric without involving system memory or the CPU. This enables lower latency, reduced CPU load, and true zero-copy data movement between accelerators.

The NVIDIA GPU to Quadra P2P demo code transfers GPU frames directly from an NVIDIA GPU to a NETINT Quadra VPU using dma-buffers and the NETINT kernel driver module. The demo allocates frame memory on the GPU using the CUDA SDK and then exports the memory as dma-buf handles. Quadra can then read the frames directly using a P2P read and encode them, providing a low-latency workflow.

Note: Driver requirements: NVIDIA's dma-buf support requires the open kernel module variant, and only this method can be used. The proprietary (closed) module can not be used as this blocks the dma-buf interop, which causes the CUDA import/export to fail.

4.2 Prerequisites

Host Operating System: Ubuntu 24.04+ recommended; kernels ≥ 6.18 preferred for smooth module builds.

Hardware: At least one NVIDIA GPU (Pascal GTX10xx or above) with dma-buf support and a NETINT Quadra card. Supported Quadra cards are the T1A, T2A, T1M or T1U. This application has been tested using an Nvidia L4 GPU (Ada Lovelace) with a NETINT Quadra T1U VPU. Please refer to the following link for the complete list of all supported GPUs (GitHub - NVIDIA/open-gpu-kernel-modules: NVIDIA Linux open GPU kernel module source).

4.3 Install NVIDIA Open Kernel Driver (dma-buf enabled)

1. Install the open kernel driver package and reboot the system. You will require 'sudo' privileges.

```
sudo apt update
sudo apt install nvidia-open # Major version 590 - at the time of writing
sudo reboot
```

2. Verify the module license string to confirm that the open driver is active (should indicate Dual MIT/GPL).

```
modinfo nvidia | grep license
Dual MIT/GPL
```

If the license is listed as **NVIDIA** then the installed driver is the proprietary version, which does not support dma-buf for P2P transfer, and therefore this demo will **not** work.

Please refer to the NVIDIA Driver Installation Guide for additional information on the NVIDIA driver installation.

4.4 Install CUDA Toolkit and Toolchain

Follow NVIDIA's package manager installer flow to install CUDA in Ubuntu.

a. Install CUDA via the repository method

Follow the NVIDIA documentation for your Ubuntu version

Overview (actual commands vary by CUDA version):

- Add NVIDIA CUDA repository
- apt update
- apt install cuda-toolkit-

The following is an example for the installation of Nvidia toolkit version 13 on Ubuntu 24.04 (this has reliable dma-buf support, CUDA 12.x+ is recommended)

```
wget
https://developer.download.nvidia.com/compute/cuda/repos/ub
untu2404/x86_64/cuda-keyring_1.1-1_all.deb
sudo dpkg -i cuda-keyring_1.1-1_all.deb
sudo apt-get update
sudo apt-get install -y cuda-toolkit-13-0
```

b. Now add the environment variables (adjust the variable values according to the version):

```
echo 'export PATH=/usr/local/cuda-13.0/bin:$PATH' >>
~/.bashrc
echo 'export LD_LIBRARY_PATH=/usr/local/cuda-
13.0/lib64:$LD_LIBRARY_PATH' >> ~/.bashrc
source ~/.bashrc
```

c. Confirm the installation is working :

```
# check nvcc version
nvcc --version
```

d. Run the Nvidia system management interface utility to check the driver and CUDA versions.

Sample output with driver and CUDA versions.

```
# check driver and cuda version
nvidia-smi
```

```
nvme@nvme-cli527:~$ nvidia-smi
Mon Dec 15 16:53:50 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 590.44.01              Driver Version: 590.44.01      CUDA Version: 13.1     |
+-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA L4                      On  |   00000000:02:00.0 Off |                    0 |
| N/A   26C    P8             12W /   72W |       0MiB /  23034MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
|   1  NVIDIA L4                      On  |   00000000:41:00.0 Off |                    0 |
| N/A   28C    P8             12W /   72W |       0MiB /  23034MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
|   2  NVIDIA L4                      On  |   00000000:C5:00.0 Off |                    0 |
| N/A   25C    P8             12W /   72W |       0MiB /  23034MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI              PID   Type   Process name                      GPU Memory |
|        ID   ID                                                             Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
nvme@nvme-cli527:~$
```

The output confirms the following driver and CUDA versions

- Driver Version: 590.44.01 (minimum supported driver: 525.xx or above)
- CUDA Version: 13.1 (minimum supported CUDA Toolkit: CUDA 12.x or above)

4.5 Build and Install the NETINT libxcoder

The NETINT libxcoder package contains the Quadra API and some sample applications (including the P2P utilities). Build the library from the NETINT release package on the host. This process will also handle the installation.

```
cd libxcoder/
./build.sh
```

4.6 Build and Load the NETINT dma-buf kernel module

The NETINT dma-buf kernel module exposes a character device; this is used by the P2P demo to set up the shared buffers with Quadra.

a. Install the Linux kernel headers that match your running kernel:

```
sudo apt install linux-headers-$(uname -r)
```

b. Build the module.

```
cd dma-buf/
make
```

c. Load the module and confirm the device node appears:

```
sudo insmod netint.ko
ls -l /dev/netint
```

d. Change the permission of character device

```
sudo chmod a+rw /dev/netint
```

e. If you have not done so already, initialize the resource monitor. Note that we run this as *root.*

```
cd libxcoder/build
sudo ./init_rsrc
```

4.7 Build and Run the xcoderp2p_read_cuda demo

The xcoderp2p_read_cuda demo performs the following:

i. Creates and allocates frames on the NVIDIA GPU
ii. Exports the dma-buf handles
iii. Instructs the Quadra card to read the frames using P2P read
iv. Instructs the Quadra card to encode the frames

The executable is typically named **xcoderp2p_read_cuda** within the nvidia_p2p_read/ folder. Follow these instructions to build and run the demo.

a. Build the CUDA P2P demo target (folder names may vary):

```
cd nvidia-p2p-read/
make
```

b. Generate test ABGR data using FFmpeg.

P2P demo supports following RAW video formats,

- ARGB → A, R, G, B (8 bits each → 32-bit)
- BGRA → B, G, R, A (8 bits each → 32-bit)
- YUV420 → Y, U, V (8 bits each)

Example ABGR stream generation with FFmpeg:

```
# the ffmpeg version doesn't mater here, we are generating
a 720p ARGB test stream for the P2P testing.
ffmpeg -lavfi testsrc=1280x720:duration=10,format=abgr
test_720_250_abgr.rgb
```

Note:

I. RGB or BGR formats without alpha are not supported.
II. Horizontal stride for luma and chroma needs to be 128-byte aligned. If not, then the input needs to be padded until it meets the requirement.

Please refer to the *Integration & Programming Guide (IPG)* document for the complete list of supported color formats and stride requirements.

c. Run the demo as root user to allow access to /dev/netint, then perform the P2P operations:

```
sudo ./xcoderp2p_read_cuda -l trace -c 0 -g 0 -i
test_720_250_abgr.rgb -s 1280x720 -m r2h -o
test_720_250_p2p_output.h265 > p2p.log 2>&1
```

d. The xcoderp2p_read_cuda command line parameters can be seen with “-h” (help).

```
options:
 -h | --help       Show help.
 -v | --version    Print version info.
 -l | --loglevel   Set loglevel of libxcoder API.
                   [none, fatal, error, info, debug, trace]
                   Default: info
 -c | --card       Set card index to use.
                   See `ni_rsrc_mon` for cards on system.
                   (Default: 0)
 -g | --gpucard    Set gpu card index to use.
                   See `ni_rsrc_mon` for cards on system.
 -i | --input      Input file path.
 -r | --repeat     (Positive integer) to Repeat input X times for performance
                   test. (Default: 1)
 -s | --size       Resolution of input file in format WIDTHxHEIGHT.
                   (eg. '1280x720')
 -m | --mode       Input to output codec processing mode in format:
                   INTYPE2OUTTYPE. [p2a, p2h, r2a, r2h]
                   Type notation: p=P2P, a=AVC, h=HEVC, r=ABGR
 -o | --output     Output file path.
 -w | --swapchain  Set size of swapchain.
                   (Default: 1) Valid values are 1, 2, or 3
```

4.8 Verification

The expected result of the demo application

1. Logs the successful dma-buf export/import paths
2. Quadra performs the P2P reads from the GPU buffer
3. An output file will be produced. This output file is an elementary stream file (e.g., H.265).
4. The h265 output stream can be validated for successful data transfer.
5. Performance metrics are also printed in the log.

4.9 Troubleshooting

If an older or proprietary version of the driver is installed then follow these steps to first remove these NVIDIA drivers and reboot the system. A reboot is required only if an upgrade was not successful.

```
sudo apt remove --purge '^nvidia-.' '^libnvidia-.'
sudo apt autoremove --purge
sudo reboot
```

Next, perform the *Install NVIDIA Open Kernel Driver* step again.

Verify the dma-buf capability by checking the name from the nvidia-smi command, and check if it is listed in the supported GPU list (https://github.com/NVIDIA/open-gpu-kernel-modules)

Check the kernel module is loaded.

```
lsmod | grep netint
```

Check the character device has the correct permissions.

```
ll /dev/netint
crw-rw-rw- 1 root root 507, 0 Dec 18 12:01 /dev/netint

If the permissions are not correct change the permissions
sudo chmod a+rw /dev/netint
```

Inspect the kernel dma-buf usage and lifetimes, to detect any leaks, or failed exports.

```
sudo cat /sys/kernel/debug/dma_buf/bufinfo
```

5 Revision History

Version	Change	Date	Author Name
0.1	Draft Document Created	17-12-2025	Shanmugam Thangavel
1.0	First Release to customer	19-12-2025	Marcus O'Brien

6 Legal Notice

Information in this document is provided in connection with NETINT products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights1 is granted by this document. Except as provided in NETINT's terms and conditions of sale for such products, NETINT assumes no liability whatsoever and NETINT disclaims any express or implied warranty, relating to sale and/or use of NETINT products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right.

A "Mission Critical Application" is any application in which failure of the NETINT Product could result, directly or indirectly, in personal injury or death. Should you purchase or use NETINT's products for any such mission critical application, you shall indemnify and hold NETINT and its subsidiaries, subcontractors and affiliates, and the directors, officers, and employees of each, harmless against all claims costs, damages, and expenses and reasonable attorney's fees arising out of, directly or indirectly, any claim of product liability, personal injury, or death arising in any way out of such mission critical application, whether or not NETINT or its subcontractor was negligent in the design, manufacture, or warning of the NETINT product or any of its parts.

NETINT may make changes to specifications, technical documentation, and product descriptions at any time, without notice. The information here is subject to change without notice. Do not finalize a design with this information. The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications.

NETINT, Codensity, and NETINT Logo are trademarks of NETINT Technologies Inc. All other trademarks or registered trademarks are the property of their respective owners.