Direct GPU RDMA Recording using 25 Gb RoCE Ethernet Links

Abstract

Graphic Processing Units (GPUs) are seeing broad usage in military embedded applications. A challenge remains with GPU based systems: how to move large amounts of data efficiently into and out of the GPU based processing system. This paper discusses an approach to move data directly in and out of GPU memory to NVMe storage over 25Gb Ethernet links, using RDMA capable NICs at aggregate rates of over 6 GB/s. Measured data rates from a RDMA demonstration system are also included in this paper.

Direct GPU RDMA Recording using 25Gb RoCE Ethernet Links

Graphic Processing Units (GPUs) are seeing greatly increased usage in military embedded applications. The GPU's ability to process massive amounts of data offers great advantages in sensor and image processing, image feature extraction and recognition, and artificial intelligence applications. A challenge remains with GPU based systems: how to move large amounts of data efficiently into and out of the GPU based processing system, often with multiple GPUs. Ideally, moving the data directly into and out of each GPUs own internal memory, without first buffering the data in host CPU memory, thus relieving the host memory of this burden and reducing system latency.

This paper discusses an approach to move data in and out of GPU memory over 25Gb Ethernet links to/from NVMe storage, leveraging Critical I/O's StoreEngine and StorePak 3U VPX modules as a data recording and playback system, and Critical I/O's RDMA capable XGE 25GbE XMC. The StoreEngine and StorePak combination provides a compact high-performance scalable storage system that can record and playback data directly to/from GPU internal memories at aggregate rates of over 6 GB/s.

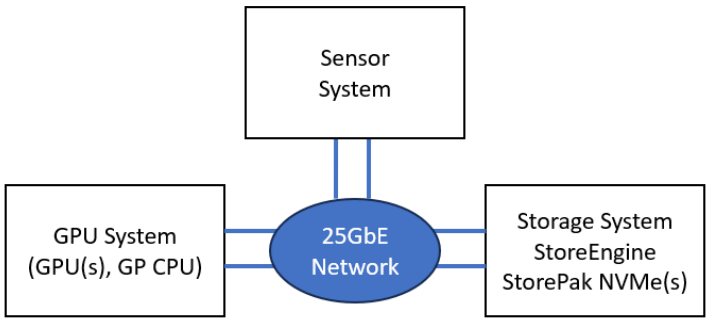

Figure 1. A Notional GPU Based Processing and Data Storage System Architecture

A proof-of-concept system was assembled to demonstrate 25GbE RoCE as a reliable high-performance transport between StoreEngine/ StorePak storage subsystem and a GPU subsystem. This system utilizes a software framework that encapsulates GPU processing with recording/playback capacities, leveraging a mechanism to DMA directly to GPU memory and later for moving back the data directly from GPU memory.

Data Transfer and GPU Software Frameworks: RoCE and CUDA

When RDMA is used over a normal Ethernet network, it is termed RDMA over Commodity Ethernet (RoCE) protocol (aka RDMA over Converged Ethernet). Note that in this paper, the terms RDMA and RoCE are used interchangeably, both to indicate the RDMA protocol running over an Ethernet transport layer. Linux has extensive and mature built-in RoCE support because RoCE is widely used in HPC and large-scale Data Center applications.

Nvidia's CUDA (Compute Unified Device Architecture) is a software platform used to leverage the acceleration potential of CPU/GPU heterogeneous systems. CUDA includes a compiler, kernel mode driver, runtime code, and optimized application specific ML/DL, DNN, FFT, Graph Analysis, Video

processing libraries. CUDA provides robust extensions to standard programming languages such as C, C++, Python, which are commonly used in high performance computing systems.

Critical I/O Storage and Network Technologies

The demonstration system shown in

Figure **2** consists of two subsystems: 1) a GPU processing subsystem with one nVidia GPU which is managed by a general-purpose CPU, and 2) a Storage (recording/playback) subsystem with two Critical I/O StorePak NVMe modules, managed by a StoreEngine module, and using StoreEngine's built-in data recording and playback software. The two chassis are interconnected with two 25Gb Ethernet links, which provide an aggregate data rate of over 6 GB/s. The 25GbE devices are Critical I/O 25GbE XGE RoCE capable network devices providing the RDMA offload transport between the two sub-systems. For this demonstration system, both the PCIe Gen 3 x8 links and the dual 25GbE links are the limiting performance factors. Additional PCIe and 25GbE links may be aggregated to achieve higher performance levels.

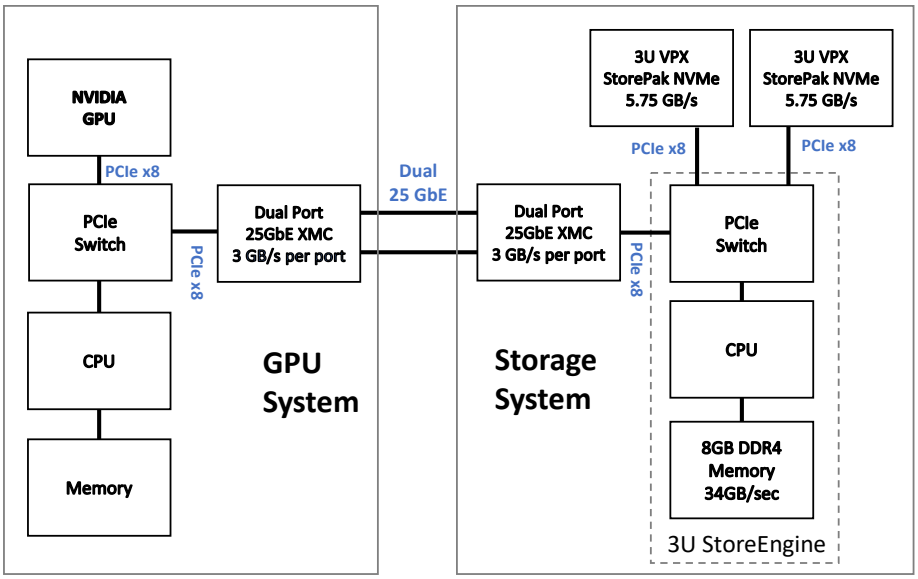

Figure 2. GPU Direct Recording using 25GbE RoCE Demonstration System

Critical I/O 3U VPX StoreEngine

StoreEngine is a scalable storage manager blade designed for high performance embedded systems. StoreEngine can be combined with one or more StorePak XMC SSD modules or 3U StorePak NVME SSD modules. For this demonstration, the StoreEngine XMC interface instead hosts a dual port 25GbE NIC module. StoreEngine has built-in easily configurable data recorder software that allows data streams from multiple sources to be recorded/played back to/from StorePaks (either XMC or 3U VPX blades) storage at high data rates.

Critical I/O 3U VPX StorePak NVMe

For this demonstration, the SSD (solid state drive) storage is provided by two StorePak NVMe 3U VPX modules. A StorePak NVMe is a single slot removable SSD module that can be used either stand-alone as Direct Attached Storage or recording and/or NAS operation in conjunction with Critical I/O's StoreEngine storage manager.

Critical I/O RoCE Capable 25Gb Ethernet XMC

Critical I/O's XGE 25GbE XMC is a RoCE enabled Ethernet NIC with dual 25GbE ports. The XGE hardware fully offloads RoCE operations, with driver support available for both Linux and VxWorks. Critical I/O's XGE products also support and are fully compliant with existing standard Ethernet protocols and infrastructure.

The Ethernet RoCE transport protocol is designed for efficient data transfers directly between application-level memory buffers with user defined permissions. Applications perform RDMA operations such as RDMA WRITE/SEND and RDMA READ/RECV that allow direct access to memory locations in each other's address spaces over the network. This combination of RoCE based DMA using application defined protection domains, low overhead RDMA instructions, and offloaded transport processing of data paths makes RoCE a highly efficient and low-latency reliable communication transport.

The use of RDMA avoids the latency and host CPU burden that would otherwise be introduced when data is first staged in CPU memory prior to copying into GPU internal memory. For improved utilization of both the CPU and GPU, the application's CPU code is decoupled from its GPU code using CUDA's standard Stream API. CUDA's Stream framework allows a CPU to operate multiple GPU kernels concurrently and to asynchronously monitor the status of already started GPU kernels.

GPU Recorder Software and PCIe Framework

The GPU software implemented for this system contains code elements that are strictly performed on the GPU as well as elements that are performed on the general-purpose CPU. The CPU code deals with controlling the flow of the application, GPU device memory allocation and communication with the remote recorder server while also responsible for spawning GPU kernels. The components of the software framework are described in the following sections.

GPU Chassis PCIe Device Memory Access

The GPU(s) and 25GbE NICs share the same PCIe root and are thus peer devices, making it possible for the 25Gb XGE to DMA data directly into or out of GPU device memory. The GPUs internally use 64KB pages and the physical address of a GPU memory page cannot be obtained using standard CPU page mapping techniques. To obtain a mapping for a GPU physical memory address the demonstration uses a nVidia Peer Memory driver (nvidia-peermem.ko).

GPU Processing with CUDA Runtime API and CUDA Libraries

The demonstration software framework is shown in **Figure** 3. User GPU applications rely on the CUDA framework to support a heterogenous combination of CPU and GPU operations at runtime. The CUDA framework API on the CPU is used to carry out GPU interactions and to start and control the compute

kernels on the GPU. This demonstration application uses standard CUDA Runtime API and standard CUDA processing libraries (e.g., cuFFT). For this system, the CUDA runtime version used is version 8.0, while the CUDA driver version was 12.0. CUDA was installed through the CUDA repository for Linux Fedora 23.

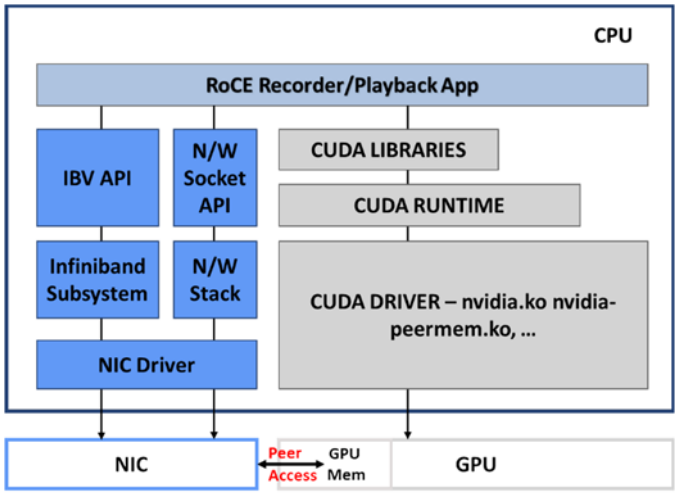

Figure 3. GPU RoCE Recorder Software Framework

The demonstration user level application includes the CUDA API and the standard InfiniBand Verbs API (IBV). When the application allocates GPU memory using CUDA API (`cudaMalloc`) it obtains an address which is accessible only on the GPU. However, an RDMA transaction to/from a GPU memory location using a third-party network device such as the XGE requires a physical GPU PCIe address to be made accessible to the XGE over the PCIe link. IBV's standard one-time memory region registration API (`ibv_reg_mr`) provides the necessary framework and utilizes the underlying peer memory driver interface to map a GPU address for XGE's RDMA access.

Network Socket API for TCP Control Connection

GPU Recorder application maintains a TCP socket connection with a RoCE recorder server operating on the remote StoreEngine for sharing control information prior to RoCE connection setup. This control link is also used during connection tear down when a recording or playback has completed.

StoreEngine RoCE Recording/Playback Implementation

A RoCE stream for recording and playback functionality is set up using the StoreEngine web-based recorder management interface. The basic configuration steps include defining the PCIe topology, creating recorder LUNs (a LUN is the basic storage container), adding recorder streams of the type RoCE and then configuring the streams.

StoreEngine Recording Filesystem

For all stream types, StoreEngine manages the recording of streams using user-defined constant size data blocks and using a recording file system. The file system has a simple hierarchy, with files that are organized into groups by recording LUN and channel.

Recorder and Playback Server

StoreEngine recording software performs partial setup of the RoCE stream if such a stream has been configured for a LUN. A stream is then completely set up when the RoCE recorder client (e.g., GPU Recorder) on the GPU host establishes a network connection and each end shares information.

GPU Compute Operations in Demonstration

The demonstration system integrates GPU computational operations with the recording and playback data pipelines to show an end-to-end application of the GPU recording architecture. In this demonstration we made an arbitrary choice to use CUDA's standard cuFFT library to perform complex-complex Discrete Fourier Transforms on complex valued single precision data points. Carrying out FFTs on the signal data lets the system take into account overall pipeline performance which now also includes the start and control of GPU computation kernels as would be observed in real world applications.

GPU Recording Demonstration Data Flow and Results

In the recording capability demonstration, the GPU is the source of the data stream, while the StoreEngine (ultimately the StorePak NVMe SSDs) are the destinations of the RDMA SEND operations.

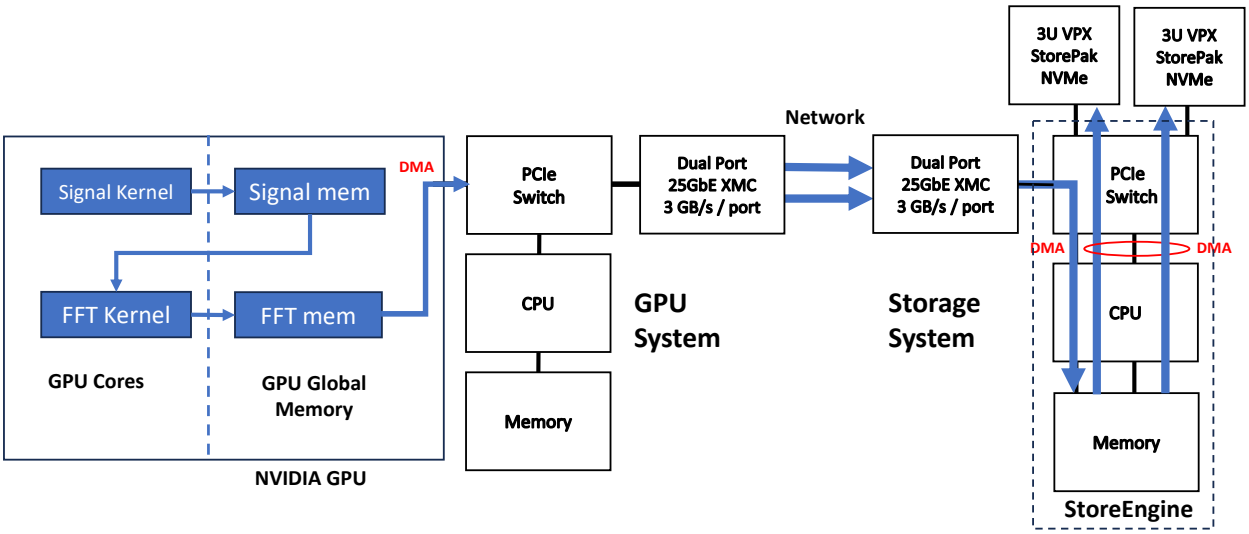

Figure 4. Dual Port RoCE GPU Dataflow with DMA from GPU Memory

For demonstration purposes the application pre-generates its own signal information at the outset. It emulates sensor data as an array of complex numbers by using CUDA cuFFT library's complex number generation method to create a 1MB buffer in GPU memory that contains 128 such arrays with each containing 1024 generated complex numbers. The application starts a CUDA kernel which is shown in Figure 4 as Signal Kernel that creates the complex number data in the desired format and loads it into the GPU Signal Mem buffer. The application also allocates 1MB transmit buffers (FFT mem in Figure 4**)** that are also GPU buffers. Each of these buffers are associated with an asynchronous stream and a FFT plan. Through the life of the application, it starts a CUDA kernel (FFT Kernel) on the transmit buffer's stream to perform an FFT of the data available in signal mem. When the FFT is completed, the application begins a RDMA SEND operation on the FFT mem transmit buffer wherein the XGE DMAs the

data directly out of the GPU memory and sends it to a remote StorePak. When the SEND completes the transmit buffer is free for another transform.

The test performs GPU RDMA recording over each 25GbE port in single and concurrent/dual port modes of operation. For dual port mode the test operates two independent instances of the RoCE recorder application. Measured results for the Direct GPU RDMA recording tests are shown in Table 1.

Table 1. Measured Single and Dual Port GPU RDMA Recording Rate.

RECORDING (from GPU)	Port 1 (MB/sec)	Port 2 (MB/sec)	Aggregate (MB/sec)
SINGLE PORT	3051		3051
		3028	3028
DUAL PORT	3029	3028	6057

When operating in playback mode the GPU memory is the destination of the RoCE operation, while the respective StorePaks are the sources. The recorder application initiates RDMA READ operations, such that the data is read from respective StorePaks and brought directly into GPU memory.

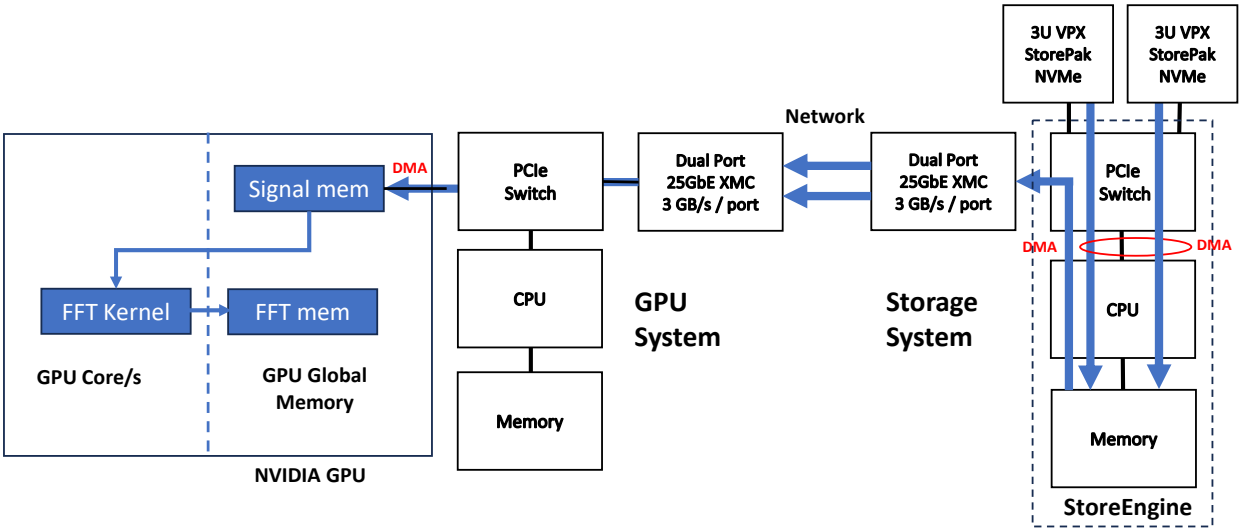

Figure 5. Dual Port RoCE GPU Playback Dataflow with DMA Direct to GPU Memory

As shown in Figure 5 above, the GPU Recorder allocates 1MB receive buffers (Signal mem) and 1MB buffers for FFT output (FFT mem) both out of GPU device memory. It associates a FFT plan with the FFT memory buffers. Each Signal mem region which acts as a receive buffer memory region is registered with the XGE using IBV API. When performing a RDMA READ the XGE does a DMA of the data into the registered receive buffer, which is a memory region within GPU memory.

The received data has a format of 128 arrays with each array containing 1024 complex numbers. When a READ operation completes it implies that RoCE has completed a DMA to the GPU memory, and therefore the application starts a CUDA kernel (FFT Kernel as shown Figure 5) on the receive buffer to perform the transform. The result of the transformation is available in the FFT mem, GPU memory region.

The GPU recorder operating in playback mode moves data directly over the network into GPU memory without having to stage it in a CPU memory buffer. Data rates were measured in both single and dual/concurrent port mode. Measured results for the Direct GPU RDMA playback tests are shown in Table 2.

Table 2. Measured Single and Dual Port GPU RDMA Playback Rate

PLAYBACK (to GPU)	Port 1 (MB/sec)	Port 2 (MB/sec)	Aggregate (MB/sec)
SINGLE PORT	3055		3055
		3054	3054
DUAL PORT	3003	3002	6005

Summary

The assembled demonstration system implements recording and playback directly to and from GPU internal memory, using a dual port 25GbE RoCE transport. This implementation uses CUDA API for GPU operations, and standard IBV API for RDMA transactions to a remote StoreEngine RoCE recorder to record and playback directly from GPU internal memory over the dual port 25GbE RoCE links without using a CPU staging buffer. It also shows that the RoCE can be used as a reliable transport in GPU applications, both for playback of a large data stream into a GPU processing pipeline as well as recording a similarly large data stream of GPU processed results into the storage system. The combination of StoreEngine and StorePak(s), along with a 25 GbE RoCE capable interconnect, provides a flexible and configurable high performance GPU recording/playback architecture with rates demonstrated to be over 3 GB/sec per XGE port, with an aggregate (dual port) recording/playback rate of over 6 GB/sec.