HPDC 2017
Washington DC
June 26-30 2017

GPU I/O persistent kernel for latency bound systems

Michele Martinelli
INFN, Sezione di Roma "Sapienza", Italy

On behalf of NaNet collaboration team of Istituto Nazionale di Fisica Nucleare, Roma, Italy (R. Ammendola, A. Biagioni, P. Cretaro, O. Frezza, G. Lamanna, F. Lo Cicero, A. Lonardo, P. S. Paolucci, E. Pastorelli, R. Piandani, L. Pontisso, D. Rossetti, F. Simula, M. Sozzi, P. Valente, P. Vicini)

Abstract

In "hybrid" High Performance Computing (HPC) systems the defining feature is the chance of integrating different kind of processing units (CPUs, GPUs, FPGAs, etc.) on the same computational node.

In this kind of systems, a feature called "GPUDirect" allows a device on the PCIe bus to autonomously read/write data from/to the internal memory of an NVidia GPU device. This enables a PCIe Network Interface Card (NIC) to send data over the network without the intervention of the host CPU, directly to a remote GPU memory and vice-versa.

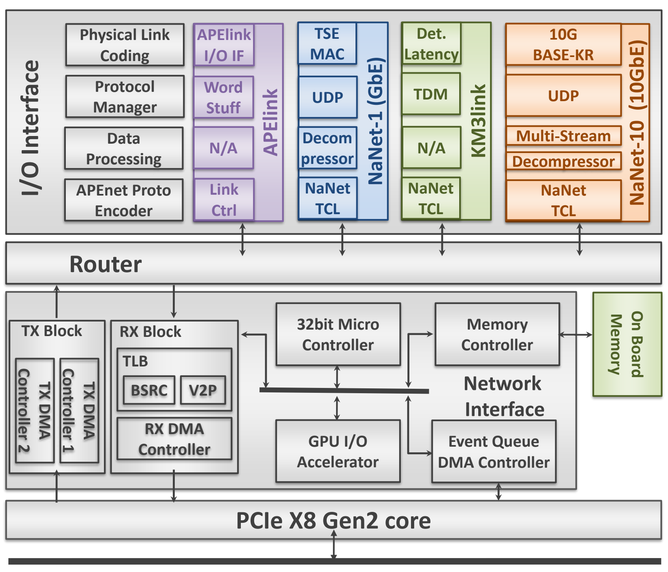
In the standard network software stack the host CPU initiates the communication by triggering the NIC, typically using a kernel-space device driver, even when the data to be transferred resides on the GPU memory.

In this situation, a network stream undergoing a GPU processing stage (i.e. reading data from the network, processing them on the GPU and sending the results back over the network) requires iterating over CPU network receive - GPU kernel launch - CPU network send; the incurring overhead of the CUDA kernel launch latency is glaring.

In our approach we envision the GPU driving the NIC by a "persistent" CUDA kernel performing both the network and processing tasks, and thus reducing the overall latency.

This same idea was also tested on the real-case scenario of the NA62 high energy physics experiment at CERN, where our custom NIC board NaNet is used.

NaNet Design

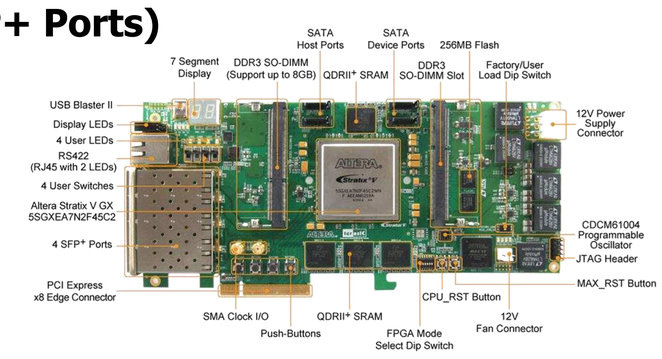


Design and implementation of a **family of** FPGA-based PCIe Network Interface Cards :

- Bridging the front-end electronics and the software trigger computing nodes.
- Supporting multiple link technologies and network protocols.
- Enabling a low and stable communication latency.
- Having a high bandwidth:
  - PCI Gen2:
    - CPU: 2.8 GB/s Read, 2.5 GB/s Write
    - GPU: 2.5 GB/s Read & Write
  - PCI Gen3:
    - CPU: 4.8 GB/s Read, 4.2 GB/s Write
    - GPU: 3 GB/s Write
- Processing data streams from the network channels on the fly (data compression/decompression, re-formatting ...).
- Optimizing data transfers with GPU accelerators.

NaNet-10 (four 10GbE SFP+ Ports)

- ALTERA Stratix V Terasic DE5-NET dev board.
- 4 SFP+ ports (Link speed up to 10 Gb/s).
- Implemented on Terasic DE5-NET board.
- GPUDirect P2P/RDMA capability.
- UDP offload supports.
- **40 GbE** development.

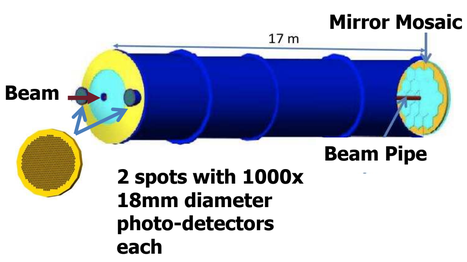


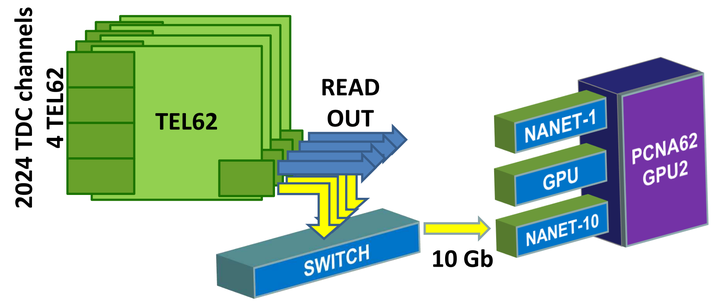
Case Study: CERN NA62 experiment

The experiment aims at measuring the branching ratio of the ultra-rare decay of the charged kaon into a pion and a neutrino anti-neutrino pair.

Ring-Imaging Cherenkov (RICH) detector is used:
the particles generate a circular footprint radiation beam onto the light-sensitive tubes of two photo-multipliers arrays. In the standard implementation, the FPGAs on the “readout boards” compute simple “trigger primitives” on the fly, such as hit multiplicities and rough hit patterns, which are then sent to a central processor for matching and trigger decision, with a time budget of 800 μs.

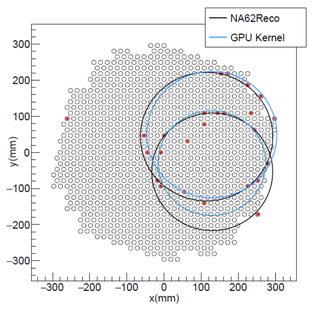
We then added a GPU-based processing stage between the RICH detector readout and the L0 trigger processor (L0TP) with the task of generating, in real-time, physics-related primitives (i.e. centers and radii of Cerenkov ring patterns on the photomultipliers arrays), in order to boost the L0 trigger discrimination power.



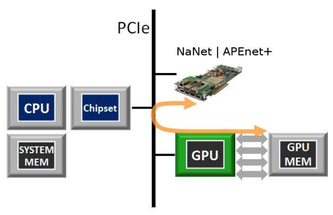


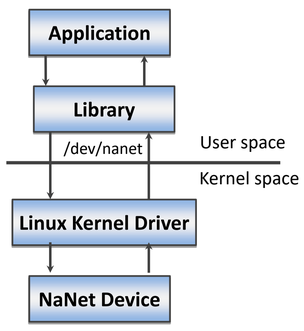
Rings pattern recognition and fit **performed on GPU:**

- ❑ Specific algorithm developed for trackless, fast, and high resolution ring fitting.
- ❑ Detection of particle speed (radius) and direction (center).
- ❑ **250 ns per event** (on NVIDIA P100).



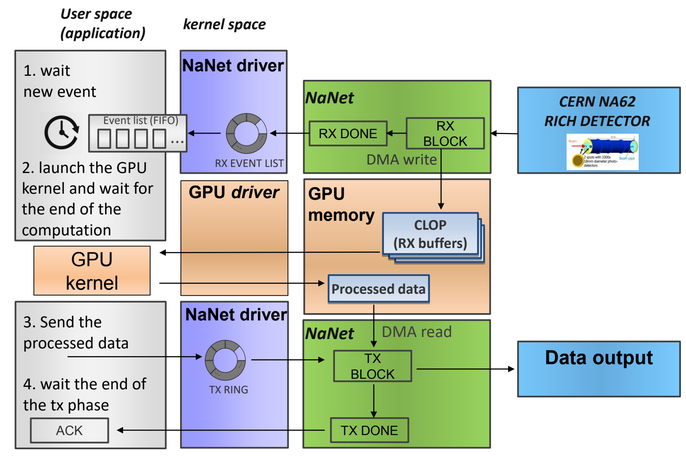
NaNet Software stack – classical approach (HOST + GPUdirect)





NaNet NIC DMA-writes data from the readout boards of the detector directly into the GPU memory (through PCIe bus) using GPUdirect RDMA.

- NaNet NIC DMA-writes a “receiving done” event in a memory region called "event queue"
  - trapped by a kernel-space device driver
  - notified to the user application which launches a CUDA kernel to process the data using the GPU (fast rings-reconstruction)
- Results of the processing – i.e. number and kind (electron, pion, kaon) of rings – is eventually sent via NaNet board to the trigger system that collects data from all detectors:
  - data are DMA-read directly from GPU memory
  - the kernel device driver (invoked by the user application on HOST) instructs the NIC by filling a “descriptor” into a dedicated, DMA-accessible memory region called “TX ring”
  - the presence of new descriptors is notified to NaNet by writing on a doorbell register over PCIe
  - NaNet NIC issues a “tx done” completion event in the “event queue”

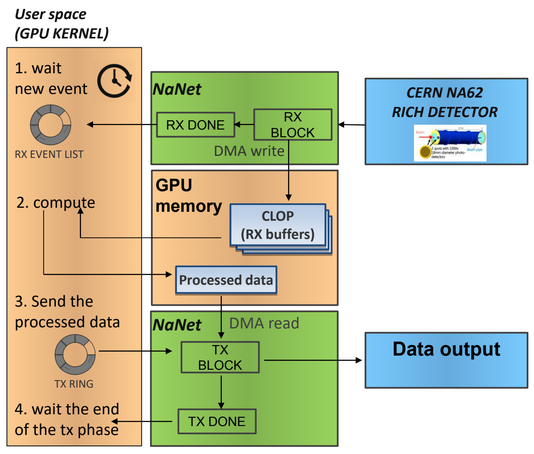


The idea: a persistent CUDA kernel directly drive the NIC

Eliminate the latency due to the user ⇔ kernel space switch by accessing the board directly from a persistent CUDA kernel

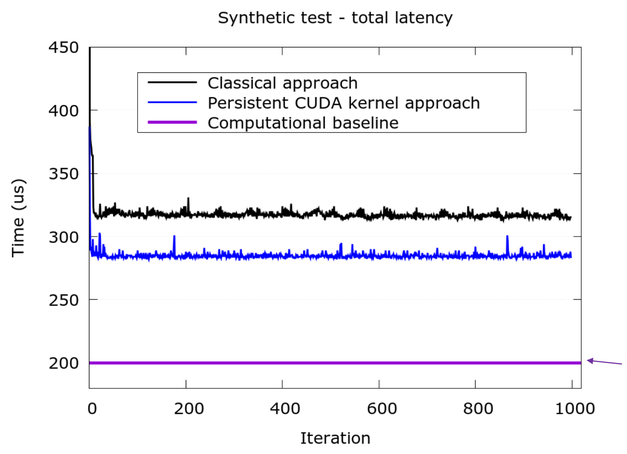
Save the overhead of launching the CUDA kernel every time a new bunch of events arrives since the kernel is already running on the GPU

1. Remap in GPU memory both the *TX ring* (for the sending phase) and the *event queue* (for both the receiving and sending phase, in order to catch the “completions”)
   - *p2p_get_pages()* GPUDirect kernel API is used to translate a virtual address into the correspondent physical one.
2. The PCIe transactions must be as few as possible
   - use a kernel-space device driver for the initialization and remapping phase, we then mainly need to write just the doorbell register in the sending phase.



Preliminary results

1) We simulate the arrival of new events by sending packets from the Ethernet interface of the PC to the NaNet NIC, which writes the data into the memory of the GPU.
2) A dummy kernel is launched to simulate the reconstruction (~200 μs).
3) Data are sent from NaNet to another Ethernet board to mimic progressing further towards the NA62 trigger system.



In the plot, the classical approach to control our NaNet NIC, based on a kernel device driver is compared with a persistent CUDA kernel in a synthetic test that reproduce the NA62 CERN experiment (gather data, process it and then send over an Ethernet connection)

baseline of computation time

Contacts:
The NaNet project: http://apegate.roma1.infn.it/nanet
Presenter Contact: michele.martinelli@roma1.infn.it
NaNet project coordinator: alessandro.lonardo@roma1.infn.it

NaNet is an INFN Scientific Committee 5 funded experiment.
This work was carried out within the ExaNeSt project, under grant agreement EU H2020 FP No 671553.

GPU I/O persistent kernel for latency bound systems

Michele Martinelli
INFN, Sezione di Roma "Sapienza", Italy. On behalf of NaNet collaboration team. email: michele.martinelli@roma1.infn.it.
PhD student.

ACM Reference format:
Michele Martinelli. 2017. GPU I/O persistent kernel for latency bound systems. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 2 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In "hybrid" High Performance Computing (HPC) systems, the defining feature is the chance of integrating different kind of processing units (CPUs, GPUs, etc.) on the same footing. In this kind of systems, a feature called "GPUDirect" allows a device on the PCIe bus to autonomously read/write data from/to an NVidia GPU memory. If this PCIe device is a Network Interface Card (NIC) the data can be sent over the network directly to a remote GPU memory and vice-versa.

One of the bottlenecks in such systems is the need of the host CPU to initiate the communication by triggering the NIC, commonly using a kernel-space device driver, even when the data to be transferred resides on the GPU memory.

One possible solution envisions the GPU itself driving the NIC, relieving the host CPUs from this task and speeding up the process of receiving, processing and sending data.

Disappointingly, this approach has been demonstrated to yield no advantage in communication bandwidth and latency on a setup equipped with an InfiniBand NIC (see [2]).

In the following, we perform similar investigation on the advantages of a GPU-driven NIC within a different environment, peculiar because latency-bound: Data Acquisition Systems (DAQs). This same idea was also tested on the real-case scenario of a physics experiment with encouraging results.

NaNet is an INFN Scientific Committee 5 funded experiment. This work was carried out within the ExaNeSt project, under grant agreement EU H2020 FP No 671553.

Conference'17, Washington, DC, USA

DOI: 10.1145/nnnnnnn.nnnnnnn

2 NANET PROJECT

NaNet [1] is a custom modular design for a FPGA-based family of PCIe NICs, with RDMA-style paradigm and supporting GPUDirect, specialized for real-time, low-latency operation.

The I/O interface is a highly flexible design that has been tailored to implement different kind (and number) of channels to meet the diverse requirements of the High Energy Physics experiments adopting NaNet for the data transport in their DAQs and Trigger systems.

NaNet-10 (the multi-port 10 GbE version of NaNet) is currently employed in the CERN NA62 experiment, which aims at measuring the branching ratio of the ultra-rare decay of the charged kaon into a pion and a neutrino anti-neutrino pair. Without delving too deep into the details of the experiment, in NA62 a stream of physics events at a 10 MHz rate has to be decimated to 1 MHz before it is passed to subsequent stages. In order to do this, a cascade of detectors is used; we have focused our work on the Ring-Imaging Cherenkov (RICH) detector. In this particular detector, the particles generate a radiation beam that impinges with a circular footprint onto the light-sensitive tubes of a photo-multipliers array. Using the information of this "rings" (radius, number of rings, etc.) we can infer what kind of particles have passed through the detector.

So, the problem translates to identifying ring patterns in a cloud of points, that is well suited for a GPU.

In the setup of the NA62 experiment, incoming data are DMA-copied into the GPU memory through the PCIe bus by the NaNet NIC which is directly connected to the readout boards of the detector. When data land in GPU memory, the NIC DMA-write a "Receiving done" completion event in a memory region called "event queue", where is trapped by a kernel-space device driver that is in charge of notifying it to the user application, which in turn launches a CUDA kernel to process the data using the GPU.

This CUDA kernel implements an algorithm performing fast rings-reconstruction. Results of this processing, i.e. number and kind (electron, pion, kaon) of rings, is eventually sent via NaNet board to a FPGA performing the first stage trigger, collecting data from all detectors. The whole process, from data receive in NaNet to results delivered to the FPGA trigger system, has to be completed strictly within 1 ms to avoid data loss.

In the transmission phase, data is to be pulled directly from GPU memory, with the kernel device driver (invoked by the user application) instructing the NIC by filling a "descriptor" with all the relevant information for the transfer (destination IP address, source data memory address, etc.), then dropping it into a dedicated, DMA-accessible memory region called "TX ring". The presence of new descriptors is notified to NaNet by writing on a *doorbell register* over PCIe bus.

After the sending phase, where data to be transferred is actually DMA-read by the NaNet NIC, this latter issues a "tx done" completion event, which is pushed into the "event queue", where the kernel-space driver acknowledges it.

As can be seen, the software stack is continuously switching between user-space and kernel-space in either the receive and the sending phase.

3 THE SOFTWARE: GPU-CONTROLLED NIC

Our idea is to have a "persistent" CUDA kernel to handle ingress of data, their processing and then egressing the results away without intervention of the host CPU.

To this end, we need to remap in GPU memory both the "TX ring" (for the sending phase) and the "event queue" (for both the receiving and sending phase, in order to catch the completions) to access them directly from the GPU process.

Notice that standard behaviour for a PCIe device driver should be allocating I/O memory via the *pci_alloc_coherent()* function which returns physically contiguous memory; memory of this kind should be remapped by the *cudaHostRegister()* CUDA API with the *cudaHostRegisterIoMemory* flag, but the disadvantage of this approach is that it increases the total latency due to the need of accessing the host memory through the PCIe bus.

So we proceed the other way round, by allocating sufficient GPU memory via *cudaMalloc()* and then translating its virtual memory address into the corresponding physical one via *p2p_get_pages()* GPUDirect kernel API; this physical address can be used directly by the NaNet DMA engines. With this approach, we are able to map both the "TX ring" and the "event queue" in GPU memory to access them directly from a CUDA kernel.

Another important aspect is represented by the PCIe transactions: as a PCIe device, NaNet is mainly driven through accessing memory-mapped registers. Even taking advantage of the aforementioned *cudaHostRegister* CUDA API to handle I/O memory from inside a CUDA Kernel, the PCIe memory read/write operations are still time consuming. Therefore, we strove to keep the PCIe transactions to as few as possible. We decided to use a kernel-space device driver for the first initialization and remapping phase, we then mainly need to write just the *doorbell register* in the sending phase to inform the NIC about the presence of fresh data to move.

4 PRELIMINARY RESULTS

The test-bed is composed by a SuperMicro server equipped with Intel Xeon E5-2630 CPUs, a NaNet NIC and an NVidia K20x GPU. We simulate the arrival of new events by sending packets from the Ethernet interface directly to the NaNet NIC, which writes the data into the memory of the GPU, then a dummy kernel is launched to simulate the rings search (~200μs) and, finally, data are sent from NaNet to another Ethernet board to mimic progressing further towards the NA62 FPGA trigger.

Using the idea presented in this paper, *i.e.* a persistent CUDA kernel to directly drive the NIC (at the moment no further improvements are made, just a single thread is used), we reap benefits on two different sides:

- we eliminate the latency due to the user ⇔ kernel space switch by accessing the board directly from the persistent CUDA kernel;
- we save the overhead of launching the CUDA kernel every time a new bunch of events arrives since the kernel is already running on the GPU.

The preliminary result in terms of total latency (adding up receive, compute and sending phases) are shown in the plot 1, where the old approach is compared with the new "kernel-persistent" one.

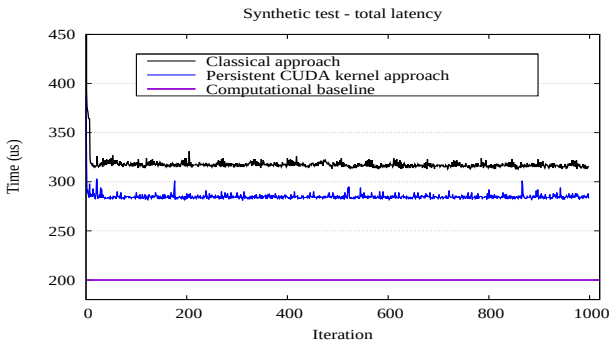


Figure 1: Preliminary performance results in terms of total latency: the classical approach to control our NaNet NIC, based on a kernel device driver, is compared with a persistent CUDA kernel in a synthetic test that reproduce the NA62 CERN experiment flow (gather data, process it and then send over an ethernet connection). Notice that the plot starts at 200μs to highlight the baseline of computation time.

REFERENCES

[1] R. Ammendola et al. 2016. NaNet-10: a 10GbE network interface card for the GPU-based low-level trigger of the NA62 RICH detector. *Journal of Instrumentation* 11, 03 (2016), C03030.

[2] L. Oden, H. Froning, and F.-J. Pfreundt. 2014. Infiniband-Verbs on GPU: A Case Study of Controlling an Infiniband Network Device from the GPU. In *IPDPS Workshop, 2014 IEEE International*. 976–983.