Online GPUAnalysis using Adaptive DMA Controlled by Softcore for 2D Detectors

R Ponsard, N. Janvier, D Houzet, V. Fristot, W. Mansour

Online GPUAnalysis using Adaptive DMA Controlled by Softcore for 2D Detectors

R. Ponsard*
ESRF
GIPSA-LAB
Grenoble, France
ponsard@esrf.fr

N. Janvier
ESRF
IEEE
Grenoble, France

D. Houzet
GIPSA-LAB
Grenoble, France

V. Fristot
GIPSA-LAB
Grenoble, France

W. Mansour
ESRF
IEEE
Grenoble, France

***Abstract*— New generation X-ray detectors enables cutting-edge experiments that can produce very high throughput data streams that are challenging to manage and store. This paper presents an evaluation of a configurable data placement mechanism from an FPGA device collecting detector raw data to a burst-cache memory and concurrently to a GPU accelerator, bypassing hardware and software extraneous copies and bottlenecks via PCI-Express. It includes a DMA controller dynamically configured in real-time by a Microblaze soft-processor. A low-latency synchronization mechanism using GPUDirect technology is presented as well. Multi-GB, DMA-able memory buffer allocation, leveraging Linux contiguous memory allocator is investigated. As illustrative workloads, real-time raw-data correction as foreseen in Serial Synchrotron X-ray experiments were processed. Obtained results showed that if one could reach a data throughput of 12.7GB/s to CPU memory when using PCIe gen3 x16, a 12-cores OpenMP CPU application processes the raw data only up to 2.7GB/s and is outperformed by a GPU accelerator (NVIDIA RTX 6000).**

FPGA; PCIe; DMA; CMA; GPUDirect; Microblaze

I. Introduction

Many photon science experiments are producing huge amount of data at high repetition rates. Using X ray detectors at full capacity requires not only a high throughput data links, but also challenges traditional workflow (acquisition, transfer, storage, batch processing) putting tremendous pressure on storage and computing infrastructures. Therefore, this enforces online data analysis in real-time pipeline and immediate rejection of non-pertinent data or compression to alleviate the pressure on the storage system.

Serial Synchrotron Crystallography (SSX) is one of the most demanding use case as it produce large amount of data for a long time and will be used in this paper as a realistic challenge as far as data processing is concerned. The foreseen detector has an adaptive gain feature: the processing of a single 4M pixels image requires 8 million 32-bit floating-point operations and will produce nearly one Terabyte of data in one minute [1].

A. *Overview of Bottlenecks in Computer Architecture*

In traditional design, CPUs are the main bottleneck source when data transfer is concerned. This problem can be mitigated using Direct Memory Access engine (DMA).

Other bottlenecks in the operating system itself might affect the throughput of the data transfer:

- during multiple copies of data from the kernel space to user space,

- when the processor goes from user to supervisor state and vice-versa, at each system call or interrupt handling.

B. *Background of Online Data Processing in GPU*

During the limited time slot available between two images (1ms), a GPU accelerator is the best candidate to perform detector raw-data correction, followed by image geometrical reconstruction and then by data compression or rejection.

This data treatment requires transferring the detector data into the GPU accelerator and a low-latency trigger mechanism to synchronize the computation and DMA data flow as well.

C. *Proposed FPGA/GPU system*

Figure 1. shows the proposed system. An X-ray detector Front-end electronics outputs data on Xilinx Aurora serial link. In the backend computer, an FPGA board serves as an Aurora/PCIe bridge.

A DMA engine push data to their final location in main memory for storage and batch processing or straight into GPU memory for fast-feedback. In this paper, we are only discussing the right side of the design.

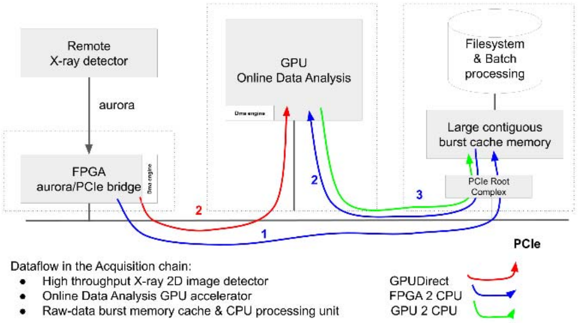

Figure 1. Architectural overview of the data acquisition system.

D. *State of the Art*

RDMA transfer is routinely done by using Mellanox Technologies adapters. Our framework address the other use cases, not compliant with Ethernet or Infiniband links. Many authors have proposed frameworks leveraging different GPU accelerators and programming API such as AMD/OpenCL [2] or GPUDirect/CUDA [3].

978-1-7281-9535-3/20/$31.00 ©2020 IEEE
DOI 10.1109/DSD51259.2020.00075

II. Proposed DMA TransferFramework

Our work is part of the RASHPA project [4], a data acquisition framework optimized for 2D X-ray detectors that would be sufficiently generic and scalable to be used in a wide diversity of new high-performance detector developments. We have restricted the scope of this paper to a limited subset of the RASHPA capabilities:

- Zero-copy transfer

The operating system isolates users and kernel memory spaces. Thus, data copies happen between memory regions: this leads to a non-unneglectable overhead for large buffers.

- Direct Memory Access

In modern computers, DMA controllers located in peripheral PCIe devices manage data transfer to and from the main memory without CPU intervention. The CPU only configures the DMA controller with a list of source and destination memory descriptors.

- Geometry reconstruction

Detectors are made of multiple independent modules, as sensor size is limited. To simplify the final data processing, the detector data are moved at an address in destination memory matching their real geometrical arrangement.

- Event mechanism

The CPU is not aware of an on-going DMA transfer by design and therefore, the DMA engine must notify the CPU, or GPU accelerator, about any event such as end of transfer, in order to let them start data processing.

A. Memory Allocation Overview

On the receiver host, the system should allocate and handle local memory buffer suitable for DMA operations: i.e. featuring large memory region, physically contiguous, non-movable and ensuring cache coherency.

Existing solutions on Linux based system include:

- Reserved memory

Reserved Memory is not visible from the Linux Operating System. By definition, this memory region is non-movable and therefore DMA-able.

- Standard memory allocation

Malloc allocates memory regions that are scattered in 4KB pages in physical memory that can be migrated during memory management operations to different physical locations. Therefore, they must be pinned to be suitable for DMA, which takes time and then DMA requires a huge number of different descriptors for multi-GB allocations.

- Huge pages

On high-performance workstation, it is possible to allocate memory backed 1GB pages, but noncontiguous.

- Allocation in a kernel module using **kmalloc**

It is possible to allocate kernel virtual memory, but movable and backed by 4KB physical pages.

- Contiguous memory allocation

A large pool of memory is dedicated to CMA allocation at boot time, which remains available by other non-CMA application. In this work, we took advantage of the CMA methods and developed a custom kernel module as the **dma_alloc_coherent** system-call is not directly accessible from a user application

B. FPGA Soft-Core Design

A proof of concept of a fast data-acquisition system has been developed using an already existing detector equipped of a high-speed serial link (Xilinx Aurora).

Figure 2. shows the transfer of scattered rectangular sub-images, extracted from the produced image (dark and light gray rectangle area).

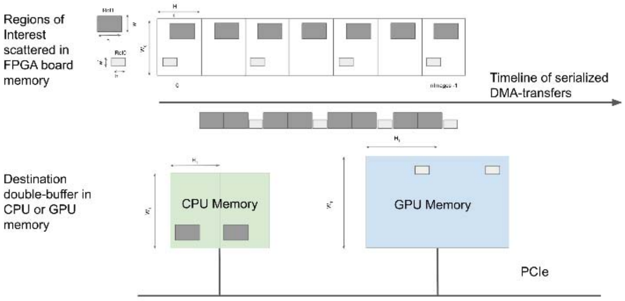
Regions of Interest scattered in FPGA board memory
RoI1
RoI0
H
0
nImages -1
Timeline of serialized DMA-transfers
H1
Destination double-buffer in CPU or GPU memory
CPU Memory
GPU Memory
PCIe

Figure 2. Multiple Region of Interest transfers serialized by DMA engine.

The final location in memory can be changed to match by example the gaps between detector modules.

The system is scalable to multiple detectors and can aggregate multiple Region of Interest from different detectors into a single computing unit. Fanning out data to multiple accelerator is also possible as well.

At CPU or GPU memory destination, a double buffer provides a lockless ping-pong mechanism between transfer and data processing.

C. On-the-fly DMA Configuration using Microblaze

The DMA descriptors are prepared during the previous on-going transfer. Two memory-descriptor configurations (addresses, size, both at source and destination) must be computed in real-time: this could not be fully pre-processed in advance as we are handling very large datasets and storing all descriptors would require a huge amount of memory.

We have developed an FPGA design for Xilinx Virtex Ultrascale+ (Alveo U200) and a firmware running on the Xilinx Microblaze soft-core to perform the on-the-fly configuration of the DMA engine.

In addition to the soft-core, the proposed design includes the following intellectual property blocks (IP):

- DMA engine (Xilinx CDMA IP)

This is the DMA engine operating in Scatter/Gather mode: it can transfer a list buffers, predefined by a list of buffer descriptors (BD). The idea was to evaluate for which configuration the calculation of the next list of BD could be hidden during the transfer of the previous list.

- PCIe Bridge (Xilinx DMA Bridge subsystem IP)

This IP handles data transfer to/from FPGA from/to CPU performing address translation between soft-core addresses (AXI bus) and host CPU physical addresses.

Allocating all the available memory in a single NUMA set was possible.

III. GPU Online Data Analysis

GPU accelerators efficiently handle massively parallel operations but are not fully autonomous. We developed a CPU application to orchestrate data transfers from the host CPU to the GPU device and back and to the scheduling of the sequence of operation.

A. *Proposed ODA Pipeline*

This application pre-launches the GPU kernels so that their execution could begin as soon as the data are available with the smallest overhead.

In the GPU device, we have implemented an execution loop with the three sequences of parallel operations:

- Data transfer from FPGA to GPU via PCIe,
- Processing of the data correction,
- Transfer of results to a large CPU memory buffer.

The sequences are CUDA Streams can possibly execute in parallel. This transforms the previous loop into a three-stage pipeline in which transfers overlap with computations.

End of DMA transfer must trigger the GPU processing: i) the FPGA soft-core application increments a host memory location by a PCIe write operation, ii) a CUDA stream memory operation puts the GPU stream on hold, iii) polling this memory location until data availability.

Transfers from CPU to GPU memory and way back and computations can overlap if the CPU memory is pinned so that GPU DMA engine could be activated.

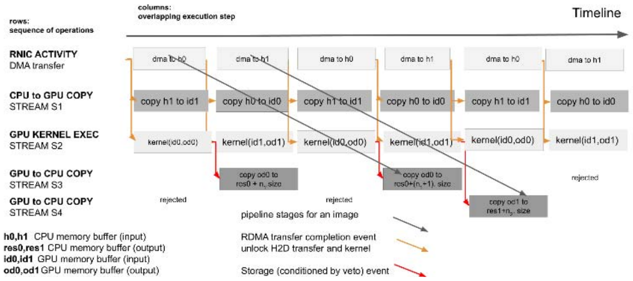

Figure 3. Synchronization mechanism between FPGA and GPU.

B. *GPU Algorithm*

The data pre-processing to convert detector raw-data into float value is performed by the formula:

$$Pixel_{i,j}\,[keV\,] = \frac{\left(Pixel_{i,j}\,[ADU] - Pede_{i,j}\,[ADU]\right)}{Gain_k \left[\frac{ADU}{keV}\right]}$$

Where: Gain_k is the gain factor at level k as set by two most significant bits of the data, Pixel the raw data expressed in Arbitrary Detector Units (ADU), Pede is the dark image.

This computation is inherently parallel. The gain factor is a large constant data set kept in GPU memory. The pedestal is a dark X-ray beam interleaved between images.

C. *GPUDirect Peer Device DMA from FPGA to GPU*

It is possible with specific high-end GPUs to decrease the transfer latency by directly transferring data from the FPGA card to the accelerator memory using GPUDirect technology.

cuMemAlloc allocates memory in GPU device by CUDA driver. In order to make this GPU memory DMA-able by the FPGA, we have developed a kernel module to pin this GPU memory region in host address space and to get its physical address forwarded to the DMA controller. The code is based on **nv_p2p_get_pages** [5].

IV. EXPERIMENTAL RESULTS

A. *Methodology*

We have performed bandwidth and latency benchmarks:

- FPGA pushing the detector data to CPU memory for storage or processing by an OpenMP application.
- Data forwarded from CPU memory to GPU accelerator and processed by CUDA application.
- Data sent directly from the FPGA board to the GPU.
- CPU and GPU simultaneously processing data.

B. *Test Setup*

Due to the Covid19 lock-down, we could not upgrade our test bench for proper performance evaluations. An ALVEO U200 board is installed in PCIE gen3 x16 slot but RTX6000 GPU is in a PCIE x4 slot, unfortunately in different PCIe interconnect. Consequently, throughput is capped below 2.8GB/s from CPU to GPU. Workstation is a Supermicro with two XEON, 12 cores, 64GB and Linux kernel 4.19.75.

C. *DMA Transfer to CPU Memory Throughput*

The Figure 4. shows FPGA TO host throughput: best results are achieved when it is possible to hide BD processing during ongoing data transfer.

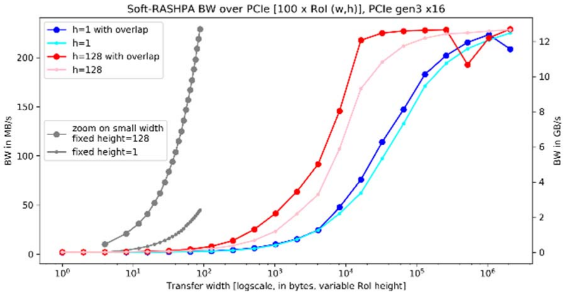

Figure 4. Observed DMA throughput from/to host memory.

DMA throughput does not suffer limitation within the FPGA design. In our setup with PCIe gen3 x16, the theoretical maximum is 16GT/s. As expected when taking into account the diverse overheads, the observed maximum is around 12.8GB/s.

During DMA transfer handled by CDMA engine, the Microblaze soft-core remains free for other tasks. This free time is used to process the configuration of the next list of

BD. DMA transfer take less time than BD pre-processing except, as expected, for small size transfer.

D. CPU OpenMP Processing Results

A CPU processing application was developed using OpenMP compiler to automatically dispatch computations on the available cores using directive **#pragma omp for**.

Observed throughput on 12 cores was around 2.8GB/s.

E. GPU Processing Results

FPGA and CPU to GPU data transfer results are shown in Figure 5. Poor BW result using GPUDirect technology result might be explained: i) FPGA and GPU not in the same NUMA set, ii) GPU is in PCIe a gen x4 slot, iii) old motherboard PCIe root complex (RC) is badly handling PCIe peer to peer requests. On another test bench, we performed DMA transfer with a Mellanox Technologies network adapter instead of an FPGA, and achieved 12GB/s to CPU and 10GB/s to GPU.

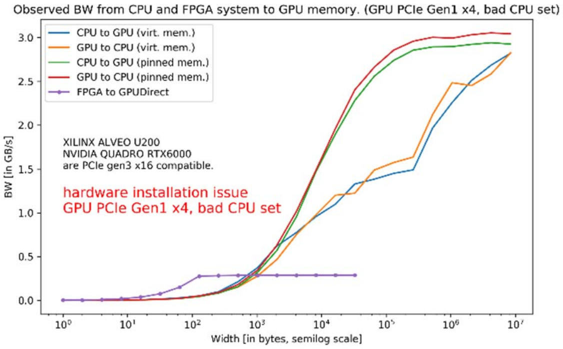

Figure 5. Observed throughput from FPGA and CPU to GPU.

F. Evaluation of GPU Low-latency Synchronization

For proper operation, the duration of the selected GPU kernel must take less time that data transfer to enable pipelined data processing.

In our setup, we have implemented overlapping data transfer in both directions and successfully process raw data correction as needed by a PSI Jungfrau detector.

Three heterogeneous systems are involved in our pipeline: an FPGA, a CPU and a GPU. However, to date, a FPGA system cannot directly trigger a GPU event and we have designed a specific cascaded synchronization procedure, described below:

- By design, the receiver CPU is not informed of the on-going DMA traffic. However, it is possible to notify an event to the CPU: when the FPGA has sent a full image, it writes a 32 bit event value in a predefined memory location.
- Upon notification, the CPU control application in turn triggers the GPU stream execution. A memory lock mechanism relying on CUDA stream memory operation (**cuStreamWaitValue32**) starts with a minimal latency the actions previously put on hold in their streams. This mechanism removes the launch overhead detrimental to real-time data processing as shown Figure 6. Hence, we measured a total kernel launch time of 4us instead of 40us.

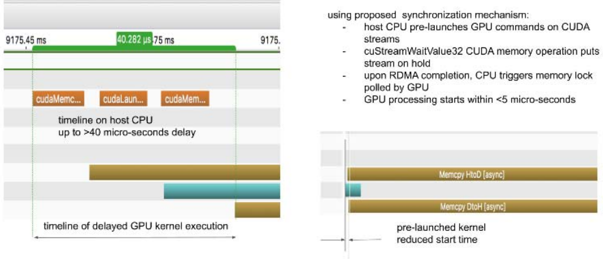
using proposed synchronization mechanism:
- host CPU pre-launches GPU commands on CUDA streams
- cuStreamWaitValue32 CUDA memory operation puts stream on hold
- upon RDMA completion, CPU triggers memory lock polled by GPU
- GPU processing starts within <5 micro-seconds

Figure 6. Low-latency GPU commands launch time.

For real-time data processing, a low execution jitter is desirable. This jitter was measured during the transfer of one buffer of 4 bytes. Round-trip was evaluated by a read operation from the FPGA (single sided communication).

V. CONCLUSION

We have developed the core components of a versatile data acquisition framework performing DMA transfer at full throughput over PCIe from FPGA to CPU memory or GPU accelerator, configurable by a soft-core firmware and featuring a low-latency synchronization mechanism with CUDA kernels.

We proposed two kernel modules for large memory allocation on host CPU and GPU device, suitable for DMA operations. New data transfer patterns can easily be tested, bypassing long workflow such as HDL synthesis.

REFERENCES

[1] F. Leonarski *et al.*, "JUNGFRAU detector for brighter x-ray sources: Solutions for IT and data science challenges in macromolecular crystallography," *Structural Dynamics*, vol. 7, no. 1, p. 014305, Jan. 2020, doi: 10.1063/1.5143480.

[2] R. Kobayashi, N. Fujita, Y. Yamaguchi, A. Nakamichi, and T. Boku, "GPU-FPGA Heterogeneous Computing with OpenCL-Enabled Direct Memory Access," in *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2019, pp. 489–498, doi: 10.1109/IPDPSW.2019.00090.

[3] P. Huber and M. Rosenthal, "Direct data exchange between FPGAs and GPUs using GPUDirect," presented at the Embedded World Conference 2020, Nürnberg, 25.-27. Februar 2020, Feb. 2020, Accessed: May 08, 2020. [Online]. Available: https://digitalcollection.zhaw.ch/handle/11475/19732.

[4] F. L. Mentec, P. Fajardo, T. L. Caer, C. Herve, A. Homs, and R. J. Horowitz, "RASHPA: A DATA ACQUISITION FRAMEWORK FOR 2D X-RAY DETECTORS," p. 4, 2014.

[5] D. Rossetti, *NVIDIA/gdrcopy*. NVIDIA Corporation, 2020.