A HETEROGENEOUS FPGA/GPU ARCHITECTURE FOR REAL-TIME DATA ANALYSIS AND FAST FEEDBACK SYSTEMS

M. Vogelgesang*, L.E. Ardila Perez, M. Caselle, S. Chilingaryan, A. Kopmann, L. Rota, M. Weber
Karlsruhe Institute of Technology, Karlsruhe, Germany

Abstract

We propose a versatile and modular approach for a real-time data acquisition and evaluation system for monitoring and feedback control in beam diagnostic and photon science experiments. Our hybrid architecture is based on an FPGA readout card and GPUs for data processing. To increase throughput, lower latencies and reduce overall system strain, the FPGA is able to write data directly into the GPU's memory. After real-time data analysis the GPU writes back results back to the FPGA for feedback systems or to the CPU host system for subsequent processing. The communication and scheduling processing units are handled transparently by our processing framework which users can customize and extend. Although the system is designed for real-time capability purposes, the modular approach also allows standalone usage for high-speed off-line analysis. We evaluated the performance of our solution measuring both processing times of data analysis algorithms used with beam instrumentation detectors as well as transfer times between FPGA and GPU. The latter suggests system throughputs of up to 6 GB/s with latencies down to the microsecond range, thus making it suitable for fast feedback systems.

INTRODUCTION

The repetition rates of modern linear accelerators such as European XFEL and TELBE [1, 2] cause increasing challenges for the development of detectors and beam diagnostics tools. With the recent developments of fast analog to digital readout systems, beam diagnostics has therefore become a big data problem. Although FPGAs emerged as ideal devices to perform on-line data analysis on large amounts of data, the implementation of particular data analysis algorithms still requires specific in-depth knowledge of the hardware and is, compared to software solutions, associated with significantly higher development costs despite efforts of FPGA vendors. At the same time, the data transmission link between the detector and the computational units or external storage is typically the bottleneck that limits the amount of data that can be processed in a given time frame.

Processing the acquired data off-line is a potential solution for most applications, however as soon as on-line monitoring or a feedback control loop is an essential part of the application, stable *real-time* data analysis with guaranteed low latencies is required. In case of on-line monitoring, data must be transferred to the processing machine as fast as possible with low variance in time. Conventional systems either based on a local temporary storage or network interconnects are not suitable due to insufficient transfer times.

* matthias.vogelgesang@kit.edu

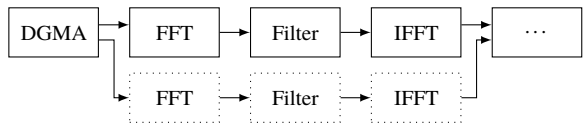

Figure 1: Overview of multi-GPU data stream processing: after inserting the data into a buffer within the DGMA filter, the data processing steps consisting of filtering in the frequency domain can be executed on separate GPUs for higher throughput.

In this paper, we will present a hardware/software architecture that bridges fast data acquisition using FPGAs and fast data processing with GPUs. Using direct memory accesses, we can decrease latency as well as increase throughput by utilizing the entire connection bandwidth. This data acquisition infrastructure is accessible from within our data processing framework which – in this particular use case – is used to analyse digitized spectrometer data in a heterogeneous compute environment. It automatically distributes data among multi-core CPUs and GPUs and uses multi-level parallelism to achieve a higher processing throughput than conventional single-threaded computing. The processing pipeline is flexible and can be re-arranged as well as extended by the user to accommodate for different applications. By adopting the proposed solution, the development time of a particular experimental setup can be reduced significantly.

ARCHITECTURE

FPGA-based Data Acquisition Platform

Our core data acquisition platform is based on our custom "Hi-Flex" FPGA board, that uses a Xilinx Virtex 7 device and is connected to the host computer through a PCI-Express (PCIe) 3.0 8-lane connection. The board has integrated DDR3 memory of 4 GB and an internal maximum throughput of 120 Gbit s^{-1}. Two industry standard FMC connectors (fully populated) are used to interface different mezzanine boards, which host the frontend electronics of different application-specific detectors such as 2D pixel detectors [3] or 1D linear array detectors [4]. The FPGA has an in-house developed Direct Memory Access (DMA) engine for PCIe 2.0 / 3.0 compatible with Xilinx FPGA families 6 and 7 supporting DMA data transfers between main system memory and GPU memory. The DMA engine is described in more detail in [5].

Data Processing Framework

The basis for processing the input data in real-time is our heterogeneous data processing framework, initially devel-

ISBN 978-3-95450-177-9

oped for X-ray image processing tasks [6]. As shown in Figure 1 the core concept is a graph defined by the user who specifies the flow of data through a set of nodes that process input data to produce some result. The tasks within these nodes is scheduled by the run-time system and executed on processing units such as CPU threads and GPU kernels. By duplicating chains of tasks intelligently as denoted by the dotted nodes, the system can make use of multiple levels of parallelism including pipelining, multi-threading, fine-grained data parallelism within a GPU as well use of multiple GPUs per machine. To process data on the GPUs, specific GPU kernel code is written and executed using the vendor-independent OpenCL standard [7].

The framework has a variety of options to access it in order to accommodate for different user requirements. The most straightforward way consists of chaining and parameterizing the plugins on the command line. For example, to read data from the FPGA, compute a one-dimensional FFT on the multi-dimensional data and write the result into an HDF5 file, the user merely has to run

```
ufo-launch
  direct-gma width=256 height=4096 ! \
  fft dimensions=1 ! \
  write filename=output.hdf5:/dataset
```

on the command line. As one can see, the user can entirely concentrate on devising the correct processing chain for his end result and does not have to worry about execution on the target platform. Besides using the framework in this immediate mode, it can also be linked to from any C or C++ program as well as being called from most third-party scripting languages such as Python via a meta-bindings. Since each task is a plugin following a simple interface, users can extend the framework by implementing new functionality as new plugins.

Low Latency Data Transfers

In order to obtain the maximum throughput of the FPGA's and GPU's PCIe connection as well as to process data on the GPUs in real-time, we have to avoid any intermediate data copies between system and GPU memory. Using AMD's DirectGMA extension for OpenCL we can tightly integrate DMA-based data transfers and transmit data directly into the GPU without storing the acquired data temporarily in system memory.

In order to initiate a data transfer on the FPGA, the physical bus addresses of the GPU memory regions to be written have to be set in special FPGA registers. For CPU writes these addresses are provided by our FPGA kernel driver. With GPUs these memory addresses are retrieved by passing the allocated OpenCL buffer to the AMD-specific `clEnqueueMakeResidentAMD()` call. Unlike general GPU buffers, FPGA-accessible GPU buffers are limited in size, for example on an AMD W9100 the maximum buffer size is about 90 MB. Therefore, we use a double buffering approach that swaps multiple temporary buffers in order to keep the DMA engine running. While filling one half of the restricted buffer, the content of the other half is transferred using `clEnqueueCopyBuffer()` to a non-DMA buffer which has the size of the final buffer. When both DMA transfer and internal copy finish, the roles of the buffers reverse, i.e. the first buffer is copied and the second is filled. Once the large buffer is filled completely, it is swapped for processing by the GPU. Using this strategy, we can overlap both DMA transfers as well as data processing. This mechanism is working because data is copied much faster within a GPU (about 320 GB/s on a W9100) than between FPGA and GPU (8 GB theoretical throughput for PCIe 3.0 x8). Besides transfers from FPGA to GPU, the GPU can also write back to FPGA registers and memory regions that were made accessible to compute kernels. This requires passing known physical FPGA bus addresses to the `clEnqueueMakeResidentAMD()` call. This creates a virtual proxy buffer that can be used by the GPU kernel to write seemingly *into* the FPGA address space. This facility is necessary for trigger applications that have to avoid passing the trigger information first to the CPU.

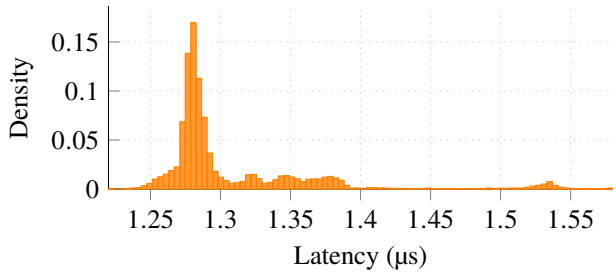

Figure 2: Round-trip latency distribution of data going from FPGA to GPU and back again.

RESULTS AND USE CASES

In this section we will investigate the behaviour of our proposed system both on a lower level comparing data throughput and latency as well as on a higher level measuring data processing times.

Performance

Figure 2 shows the latency distribution measured for data transfered from FPGA to GPU and back to the FPGA again. An FPGA-side timer is started as soon as a write happens. Meanwhile, a GPU kernel is launched that polls for that data to arrive. Once the kernel thread registers a change, it triggers a response causing write back to the FPGA. As we can see, the mean latency of data transfers of about 1.31 µs with a jitter of 0.06 µs suggests applicability of our system for most monitoring and feedback applications.

In Figure 3, the system throughput is plotted for a increasingly larger datasets. Each transfer is measured with three different data block sizes that were transmitted at once. The throughput includes the data transfer as well as all as any overheads induced by startups and the run-time system. With this in mind, we can see a throughput that is near full utiliza-

ISBN 978-3-95450-177-9

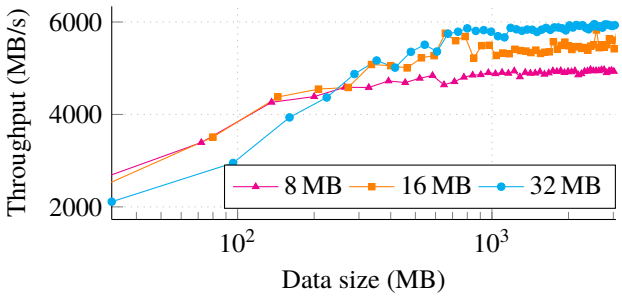

Figure 3: System throughput from FPGA to GPU for different data block sizes and including all overheads.

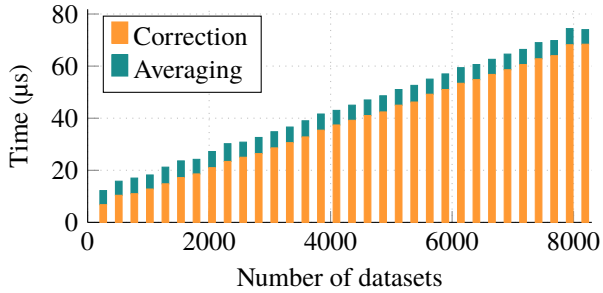

Figure 4: Kernel execution times for processing averaging and background correction on an AMD S1970.

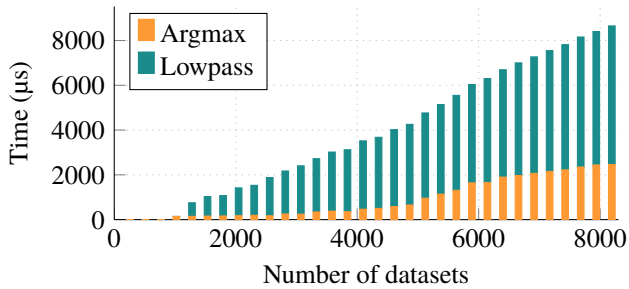

Figure 5: Kernel execution times for specific analysis.

tion of the PCIe bus for larger data sizes and is sufficient for even the highest demanding photon science experiments.

KALYPSO Data Analysis

As a typical application example for a beam diagnostic solution, we have chosen KALYPSO, a linear array detector developed to allow real-time monitoring of longitudinal bunch profiles measured with Electro Optical Spectral Decoding setups. The detector setup and characteristics is described in detail in [8]. The main idea of this detector setup consists of the reconstruction of the bunch profile from the measured spectrum of a laser pulse. In order to measure the bunch profile, three different measurements have to be acquired: the background signal, the unmodulated signal with the laser pulse used as a reference and the modulated signal where the laser pulse contains the information of the bunch profile. The discretized spectrum of the laser pulse is acquired and digitized on a dedicated card that is connected to the "Hi-Flex" FPGA board.

The most fundamental work required for subsequent data analysis is the interpretation of the raw ADC channel data transferred from the FPGA board and *correction* for background noise. The first step consists of *averaging* the stream of background and unmodulated data sets d_b and d_u over time, i.e. $\hat{d}_{\{b,u\}}[i] = \frac{1}{n}\sum d_{\{b,u\}}[i][j]$ for the i-th channel of the j-th out of n pre-recorded datasets. This data is used to remove static background and dark noise from the currently recorded datasets d_m, i.e.

$$\hat{d}_m[i][j] = \frac{d_m[i][j] - \hat{d}_b[i \bmod 256]}{\hat{d}_u[i \bmod 256]}.$$

Each time raw data is accessed, the transmitted value has to be masked with `0x3fff` in order to retrieve the 14 bit ADC values from a 16 bit data packet. Because of data dependencies, we map one GPU thread to one ADC channel to parallelize the averaging which results in 256 threads running in parallel. The data correction itself is free of any data dependencies, thus one GPU thread computes the corrected value for one input.

Figure 4 shows the execution time for averaging and correction. Averaging is constant, with a mean execution time of 5.975 µs, because the same background and unmodulated data is used for each run, the time for correction scales linearly and is approximately $t(n) = (0.00777n + 5.105)$ µs.

To smoothen the high variance of the input data a low pass filter based on moving averages has been implemented. Similar to the regular averaging filter, the kernel for *moving averages* maps one GPU thread to one particular ADC channel, computing the moving averages with a default order of 3 for all datasets. In certain applications such as measurement of the arrival time, it is necessary to find out which of the channels a has the largest value at the moment, i.e. one wants to determine $a_i = \text{argmax}_j d[i][j]$. This is a simple search across all datasets which also requires mapping of one GPU thread to one channel. Figure 5 compares the execution time for these two kernels: unlike the previous pre-processing tasks these steps take several milliseconds to compute thus are only useful in a monitoring environment.

Besides these fundamental tasks, we can also make use of the existing *transpose* filter to re-interpret the dataset columns as rows and do subsequent computations in time domain. Moreover, we can use the existing *fft* and a newly written *powerspectrum* filter to compute the frequency components of the signal. Whereas the power spectrum computation can be parallelized in the same way as the background correction kernel with similar performance characteristic, the FFT is a more complex filter requiring appropriate padding and more time.

CONCLUSION

In this paper, we introduced a comprehensive data acquisition and processing system with low latencies and high throughput properties as its main characteristics. The core

ISBN 978-3-95450-177-9

components of the system are tight coupling between data acquisition and processing that is driven by the acquiring FPGA rather than the host CPU. Unlike conventional data paths, writing from FPGA to the GPU directly allows lower latencies and higher total throughput and enables real-time monitoring, control and feedback systems. To realize real-time monitoring we also off-load critical pre-processing to the GPU with average execution times in the µs range.

The entire setup was used to measure and analyse data with the KALYPSO detector setup and has proven to be a viable alternative to conventional off-line data analysis given insight into the data while acquiring it. In the future, we will investigate how the analysis can be visualized in a user-friendly way and how the result can be incorporated in the entire feedback loop.

ACKNOWLEDGEMENT

This work was partially funded by the German Federal Ministry of Education and Research (BMBF) as UFO-2 under the grant 05K10VKE. The authors would also like to thank Patrik Schönfeldt and Miriam Brosi for the data and helpful discussions.

REFERENCES

[1] B. Green, S. Kovalev, J. Hauser, M. Kuntzsch, H. Schneider, S. Winnerl, W. Seidel, S. Zvyagin, U. Lehnert, M. Helm *et al.*, "Telbe-the super-radiant thz facility at elbe," *Verhandlungen der Deutschen Physikalischen Gesellschaft*, 2013.

[2] M. Altarelli, R. Brinkmann, M. Chergui, W. Decking, B. Dobson, S. Düsterer, G. Grübel, W. Graeff, H. Graafsma, J. Hajdu *et al.*, "The european x-ray free-electron laser," *Technical Design Report, DESY*, vol. 97, pp. 1–26, 2006.

[3] M. Caselle, S. Chilingaryan, A. Herth, A. Kopmann, U. Stevanovic, M. Vogelgesang, M. Balzer, and M. Weber, "Ultrafast streaming camera platform for scientific applications," *IEEE Transactions on Nuclear Science*, vol. 60, no. 5, pp. 3669–3677, 10 2013.

[4] L. Rota, M. Vogelgesang, L. A. Perez, M. Caselle, S. Chilingaryan, T. Dritschler, N. Zilio, A. Kopmann, M. Balzer, and M. Weber, "A high-throughput readout architecture based on pci-express gen3 and directgma technology," *Journal of Instrumentation*, vol. 11, no. 02, 2016.

[5] L. Rota, M. Caselle, S. Chilingaryan, A. Kopmann, and M. Weber, "A high-throughput pcie dma architecture for gigabyte data transmission," *IEEE Transactions on Nuclear Science*, vol. 62, no. 3, pp. 972–976, 6 2015.

[6] M. Vogelgesang, S. Chilingaryan, T. dos Santos Rolo, and A. Kopmann, "Ufo: A scalable gpu-based image processing framework for on-line monitoring," in *High Performance Computing and Communication 2012 IEEE 9th Int. Conf. on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th Int. Conf. on*, 6 2012, pp. 824–829.

[7] A. Munshi, B. Gaster, T. Mattson, J. Fung, and D. Ginsburg, *OpenCL programming guide*. Addison-Wesley Professional, 2011.

[8] L. Rota, M. Caselle, N. Hiller, A. Müller, and M. Weber, "An ultrafast linear array detector for single-shot electro-optical bunch profile measurements," in *Proceedings of the 3rd Int. Beam Instrumentation Conf.*, 9 2014.

ISBN 978-3-95450-177-9