EUCALL - WP5 – UFDAC

Milestone 5.2: M18 / 31.03.2017

"Evaluation of test case implementations for processing of digitizer data and image analysis"

This milestone deals with two test cases for high throughput data analysis test cases for digitizer data and image analysis. These test cases came out of the previous Milestone 5.1 that defined requirements, interfaces and formats of the various application cases in UFDAC. Following a recent GPU/FPGA workshop at the European XFEL organized by the UFDAC WP that was open for all EUCALL members we loosely defined what applications are suitable for FPGA solutions and which one are suitable for GPU solutions.
Developers should prefer an FPGA-based solution if:

- Low latencies are required, preferably due to direct response to the outcome of the analysis.
- High streaming throughput is required, meaning that buffering a stream is not acceptable, e.g. digitizer or event data that is used for direct feedback. Note that of course buffering of data streams is possible and standard procedure on FPGAs, but is required on GPUs.
- Local memory requirements are not exceptionally high.
- Scalability to massive numbers of independent data pipelines is easy and data exchange between FPGAs is not overly complex.
- Analysis code is limited in complexity and does not need many changes once implemented efficiently.
- Analysis code can stay locally on one FPGA or communication between FPGAs is embarrassingly parallel.

Developers should prefer a GPU-based solution if:

- Latencies can be hidden by using buffering and block-parallel analysis of data, which is often the case for image data.
- High throughput is required but streams can be buffered and worked on in blocks/chunks/packets of data.
- PCIe bandwidth is not limiting in terms of throughput wanted. This limitation can occur for optimized GPU code for which the analysis takes less time than memory transfer to the GPU.
- Local memory requirements are high and can be mitigated by dividing data into blocks/chunks/packets.

This project has received funding from the *European Union's Horizon 2020 research and innovation programme* under grant agreement No 654220

- Parallelization of algorithms over blocks/chunks/packages is possible and the computations performed on one block/chunk/packet are mostly independent from those on other ones.
- Analysis code can become complex and profits from independent steps that can operate on the same data or similar amount of data, see e.g. imaging pipelines.
- Analysis code can require exchange of data between GPUs
- Analysis code requires nonlinear scalability with data rate and cannot be trivially scaled.
- Analysis code is complex, can be subject to frequent changes and might require the use of more than one GPU.

It is clear that applications cannot easily be put in one of these two categories and that the rules mentioned above are not strict but meant as a guideline. As an example, simple signal or image processing tasks can be implemented on FPGAs as well as on GPUs. The decisions for a certain hardware or even a combination of both depends on the application environment, the programming knowledge accessible, price, power, space and maintenance requirements and the question on future needs for scalability.

In general, more complex computations will be easier to implement on GPUs than on FPGAs. However, some FPGA vendors now allow to program their FPGAs using OpenCL or dialects thereof that also can be compiled to run on GPU architectures and CPUs, connection of external DDR3 memory to FPGAs is possible as is FPGA to FPGA communication. Regardless of the choice of platform, it is difficult to write portable code for high data rate analysis applications that allows for changing the hardware during the lifetime of the code. We plan to address some of these problems inside UFDAC.

We thus have restricted ourselves to select two test cases that are typical of the applications supported within UFDAC, namely high data rate digitizer data and image data and show first performance measurements with a given hardware. The insight gained from these test cases will then be fed back into the developments for the other test cases.

This project has received funding from the *European Union's Horizon 2020 research and innovation programme* under grant agreement No 654220

1. Test case Digitizer Data / FPGA

The FPGA test case is based on an measuring the total X-Ray fluorescence yield via Avalanche Photo Diodes (APDs) from a liquid molecule sample excited by a laser system synchronized to the PETRA III storage ring at MHz repetition rate [for details see: A. Britz et. al., J. Synchrotron Rad. (2016). 23, 1409–1423]. This setup is a test bed for future operation at the European XFEL measuring X-ray pulse characteristics at MHZ repetition rate.

Description of the test case including performance requirements

The X-ray fluorescence signal generates APD signal traces that are fed into the DAQ system (a microTCA based solution with high-speed digitizers; SP Devices). Each digitizer consists of four 12-bit analog-to-digital converters (ADCs) accompanied by a Field Programmable Gate Array (FPGA; Xilinx Virtex 6) for online data processing. In its standard configuration, this ADC is able to digitize the traces of our APD detectors at a sampling rate of 2 GS/s.

Alternatively, it is also possible to configure the digitizer in the so-called two-channel interleaved mode to increase the sampling rate to 4 GS/(s channel) (which reduces the number of available channels to two per ADC).The repetition rate of X-Ray pulses was 7.8 MHz and pulses were collected for processing in packets of either 40 or 60 pulses using a 130 kHz gating.

Decision on FPGA-based solution

The detector outputs have been connected to ADCs and then FPGAs to process data to extract different information like time of flight, energy of pulses or pulse peak information then forwarding this output to software for more processing.

At European XFEL digitizers from “Sweden SP-Devices Company” have been used. These digitizers exploit four 2 GSPS ADCs and one Xilinx Virtex 6 FPGA with PCIe connection to transfer data to PC. It is also possible to configure it in 2 ADCs at four GSPS.

This technology provides remote access to digitizers for programming, upgrading and also monitoring which is suitable for synchrotron applications. Several different algorithms have been implemented in these FPGAs for on-line processing.

With this configuration continuous extraction of the most important pulse characteristics at MHZ repetition rate could be obtained.

This project has received funding from the *European Union's Horizon 2020 research and innovation programme* under grant agreement No 654220

Algorithmic details

Zero suppression

The idea of zero suppression is to detect all pulses in stream of received data from ADCs and send pulse samples and also time of flight to the PC instead of all raw data from ADC. To find a pulse, a threshold should be defined to compare samples with this threshold. When a few numbers of samples are above this threshold a rising edge of a pulse will be reported then stamping the time and grabbing data will be started. This algorithm is composed of two parts. First part is threshold finder and is called Adaptive Threshold Filter and second part is pulse finder and reporter. Adaptive Threshold tracks noise level and defines Threshold according to noise level. Pulse finder uses output of Adaptive Threshold and compares stream of data with threshold. When data is above threshold that means a pulse is detected.

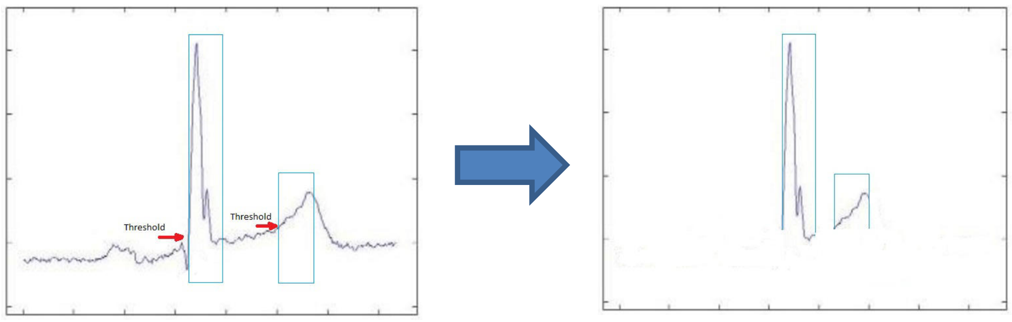


Figure 1: Zero suppression reduces the sample points by selecting those points identified as signal data.

Precise Peak Detection

This algorithm is similar to zero suppression but instead of reporting time of flight and all samples of pulse, it reports the area which peak can happen. It acts like zero suppression to detect rising edge of pulses using one threshold value, and then looks for peak value in detected pulse. The precision of the peak value is limited by the ADC's resolution, therefore an area is reported for peak value which is several samples as well as time of maximum value among these samples. These information are forwarded to subsequent processing steps, for example interpolation to get a more precise peak value.

For detecting peak values eight samples are processed along with four other samples, two samples on the right side and two other samples on the left side (the reason for eight samples is that available digitizers provide eight parallel samples per clock cycle). These ancillary samples are old and new samples in time order. Differences between every other sample are calculated then these differences are compared to zero to find zero crossing. This result is valid only when the samples are above threshold.

This project has received funding from the *European Union's Horizon 2020 research and innovation programme* under grant agreement No 654220

For each eight samples only two peak values are detectable. Therefore for each detected peak four samples are reported and peak value will be the second sample from the right out of these four samples along with index of the peak respect to the trigger signal.

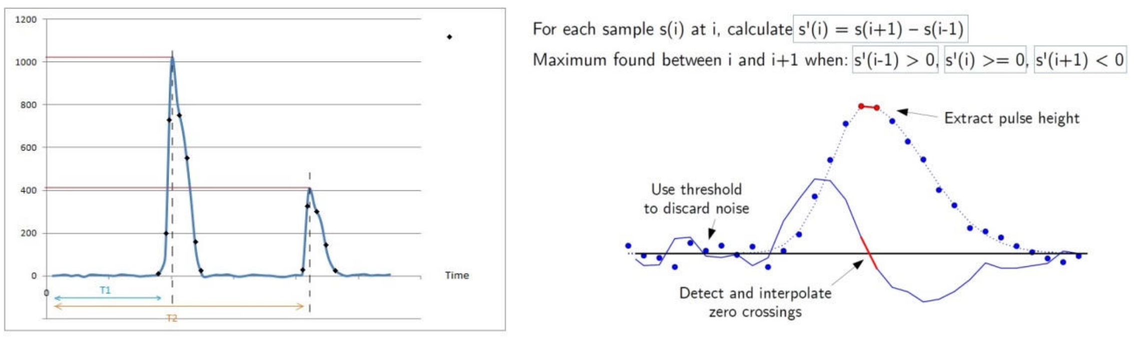


Figure 2: Peak finding algorithm using a sampling area around the maximum value for later interpolation.

Energy calculation

The energy calculator is designed to perform an integration of pulse area. It depends on input signal and type of digitizer or even ADC's base line, e.g the noise level is not necessarily zero. Therefore the value of the pulse integration should be corrected by removing the DC value of the base line. To do integration for both noise level and pulse area, certain areas are defined for both, then integration is done for these areas and finally an overall calibration is applied. This area is defined by first analyzing raw pulses and extracting pulse position, then applying this information to the FPGA algorithm.

In other words the user has to define some parameters before starting energy calculation. That means raw data should be captured first and after analyzing it and extracting required parameters, then these parameters should be applied to the FPGA energy calculator. These parameters are initial delay, noise area which is start and stop points, pulse region which is given by the pulse start and stop and an upper limit. The difference between upper limit and initial delay will define the pulse repetition frequency.

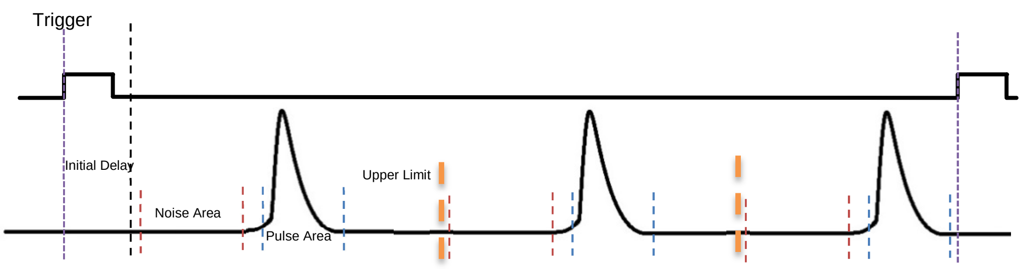


Figure 3: Pulse energy calculation by window selection, base line removal and pulse integration.

This project has received funding from the *European Union's Horizon 2020 research and innovation programme* under grant agreement No 654220

Measurement results

Measurements at PETRA III showed that indeed the aforementioned setup was able to perform all operations at the design repetition rate of 7.8 MHz [for details see: A. Britz et. al., J. Synchrotron Rad. (2016). 23, 1409–1423].

Dissemination & Transfer within UFDAC / EUCALL

As with every FPGA solution transfer to a different FPGA hardware can be difficult. However, many components of the setup such as the microTCA connection are already based on standards used by several partners. As such, transfer of algorithmic ideas and central parts of the DAQ will be discussed within UFDAC. It is foreseen to also implement the same algorithms on GPUs using a parallelized code from HZDR.

This project has received funding from the *European Union's Horizon 2020 research and innovation programme* under grant agreement No 654220

2. Test case Image Analysis / GPU

The GPU test case deals with the conversion from raw detector data coming from the PSI JUNGFRAU detector to photon counts based on calibration curves obtained with a calibrated source. The conversion can be performed on each image individually and requires simple matrix operations.

Description of the test case including performance requirements

Each JUNGFRAU chip comprises 256x256 pixels of 75x75 μm^2 each. These chips are arranged in arrays of 2x4 chips to form modules of 0.5 MPixels and about 4x8 cm^2. Each JUNGFRAU module connects to a JUNGFRAU Module Control Board (JMCB) that controls the JUNGFRAU chips during the readout phase, see Figure 4.

The main component of the JMCB is an Altera Cyclone V Field Programmable Gate Array (FPGA), that acquires the digital output of the chip (employed gain bits) and the digital output of the 32 ADC channels that carry out the conversion of the analog output from the pixels (deposited charge).

The whole stream of data (digital and analog) is properly descrambled to obtain a contiguous image that is later streamed out to the user server via 2x10 GBps links. The readout capability of the JMCB is designed to sustain readout rates of 2000 frames/s.

By arranging several of these modules it is possible to build detector systems of several Mpixels. More specifically, multi-module systems of up to 16 Mpixels consisting of 32 JUNGFRAU modules are planned for various experimental setups, with a net output data-stream of 80 GBps.

The data streamed out for every pixel consists of the digitized value of the deposited charge plus the coding of the employed gain stage together in a 16-bit word.

However, data scientists need to process this output stream of data to obtain meaningful information. In fact, what scientists need is a photon image to do their research, that is, a photo with the number of photons recorded per pixel.

The conversion to number-of-photons is simple but has to be carried out for every single pixel and with parameters that vary from pixel to pixel. These parameters are the actual gain for each gain stage and the offset due to noise and leakage which also varies from gain stage to gain stage.

This project has received funding from the *European Union's Horizon 2020 research and innovation programme* under grant agreement No 654220

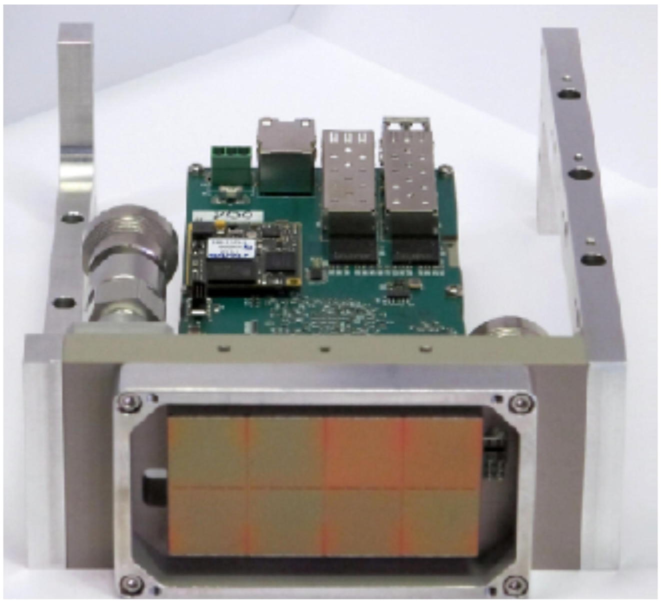


Figure 4: Image of one JUNGFRAU detector module with its JMCB.

Decision on GPU-based solution

Currently, this conversion from charge to number-of-photons is carried out offline, but an on-line processing would have several benefits apart from the reduced CPU usage, for instance:

- Perform further on-line signal processing.
- Carry out on-line data compression to reduce both throughput and storage requirements,

since number-of-photon data can be better compressed.

All these sets of parameters per pixel can be grouped in 6 maps per JUNGFRAU module, namely 3 gain maps plus 3 offset (pedestal) maps. When coding this parameters in 16-bit words, all this information adds a total of 6MB of data that needs to be accessed at a rate of 2 kHz, which sets the focus not only on the processing time, but also on the memory bandwidth.

Given the high capabilities in parallel computation, high memory bandwidth and scalability to work with bigger detector systems, GPUs seem to be the best alternative to carry out this task. Moreover, adding future capabilities such as online compression of the pixel data could be implemented easily.

This project has received funding from the *European Union's Horizon 2020 research and innovation programme* under grant agreement No 654220

Algorithmic details

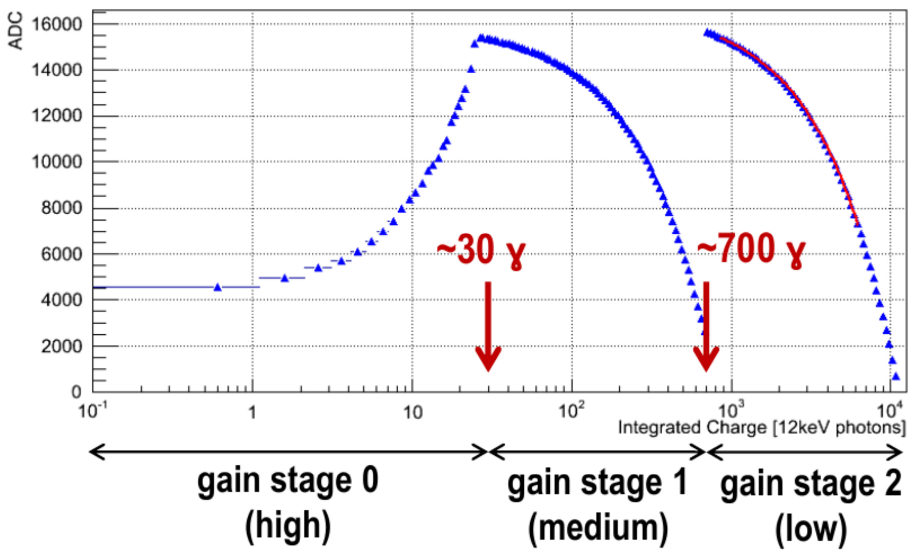


Figure 5: Gain stages (ADC count vs. integrated charge)

The detector gain is divided in three stages G0, G1 and G2, see Figure 5. For computing the photon number, the pedestal maps correction CCorr are computed from the ADC values DADC and the pedestal maps P0, P1 and P2:

G0: Ccorr[x,y] = DADC[x,y] – P0[x,y]

G1: Ccorr[x,y] = P1[x,y] - DADC[x,y]

G2: Ccorr[x,y] = P2[x,y] - DADC[x,y]

and then the gain maps Ecal are computed from the gain curves Gain0, Gain1 and Gain2 and the pedestal maps corrections

G0: Ecal[x,y] = Ccorr[x,y] * Gain0[x,y]

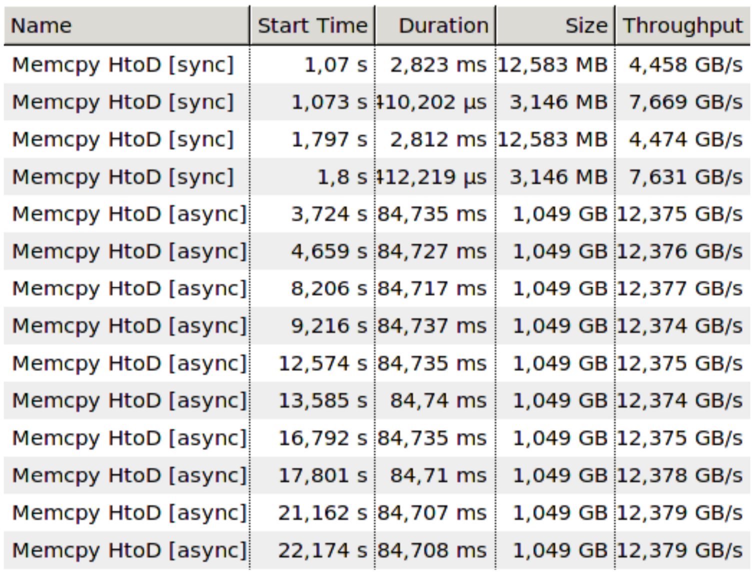
G1: Ecal[x,y] = Ccorr[x,y] * Gain1[x,y]

G2: Ecal[x,y] = Ccorr[x,y] * Gain2[x,y]

For the GPU-based solution a concurrently accessible ring buffer was implemented on the host side (CPU) which can be accessed for writing and reading. The detector data is copied to the GPU and these copies are interleaved with computations on the GPU, effectively hiding the latency of the copying process. For this, asynchronous kernel calls and asynchronous memory copies with CUDA streams and pinned memory are used.

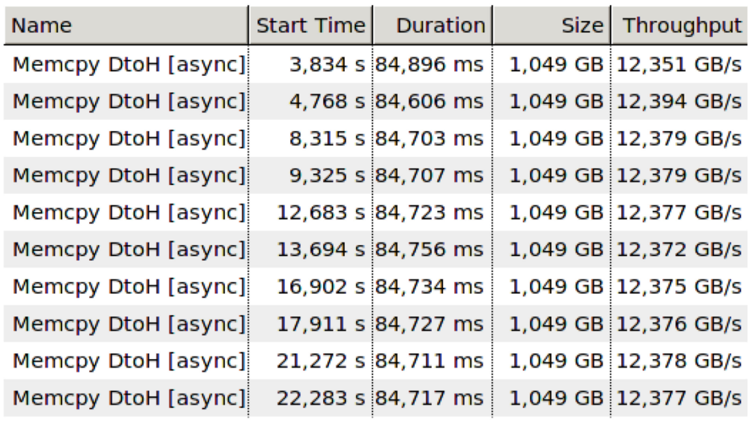
This project has received funding from the *European Union's Horizon 2020 research and innovation programme* under grant agreement No 654220

Measurement results



Name	Start Time	Duration	Size	Throughput
Memcpy HtoD [sync]	1,07 s	2,823 ms	12,583 MB	4,458 GB/s
Memcpy HtoD [sync]	1,073 s	410,202 µs	3,146 MB	7,669 GB/s
Memcpy HtoD [sync]	1,797 s	2,812 ms	12,583 MB	4,474 GB/s
Memcpy HtoD [sync]	1,8 s	412,219 µs	3,146 MB	7,631 GB/s
Memcpy HtoD [async]	3,724 s	84,735 ms	1,049 GB	12,375 GB/s
Memcpy HtoD [async]	4,659 s	84,727 ms	1,049 GB	12,376 GB/s
Memcpy HtoD [async]	8,206 s	84,717 ms	1,049 GB	12,377 GB/s
Memcpy HtoD [async]	9,216 s	84,737 ms	1,049 GB	12,374 GB/s
Memcpy HtoD [async]	12,574 s	84,735 ms	1,049 GB	12,375 GB/s
Memcpy HtoD [async]	13,585 s	84,74 ms	1,049 GB	12,374 GB/s
Memcpy HtoD [async]	16,792 s	84,735 ms	1,049 GB	12,375 GB/s
Memcpy HtoD [async]	17,801 s	84,71 ms	1,049 GB	12,378 GB/s
Memcpy HtoD [async]	21,162 s	84,707 ms	1,049 GB	12,379 GB/s
Memcpy HtoD [async]	22,174 s	84,708 ms	1,049 GB	12,379 GB/s

Table 1: Memory transfer rate (throughput) from host (CPU) to device (GPU). Except for the very first few images the throughput is always close to the maximum value achievable with PCIe.



Name	Start Time	Duration	Size	Throughput
Memcpy DtoH [async]	3,834 s	84,896 ms	1,049 GB	12,351 GB/s
Memcpy DtoH [async]	4,768 s	84,606 ms	1,049 GB	12,394 GB/s
Memcpy DtoH [async]	8,315 s	84,703 ms	1,049 GB	12,379 GB/s
Memcpy DtoH [async]	9,325 s	84,707 ms	1,049 GB	12,379 GB/s
Memcpy DtoH [async]	12,683 s	84,723 ms	1,049 GB	12,377 GB/s
Memcpy DtoH [async]	13,694 s	84,756 ms	1,049 GB	12,372 GB/s
Memcpy DtoH [async]	16,902 s	84,734 ms	1,049 GB	12,375 GB/s
Memcpy DtoH [async]	17,911 s	84,727 ms	1,049 GB	12,376 GB/s
Memcpy DtoH [async]	21,272 s	84,711 ms	1,049 GB	12,378 GB/s
Memcpy DtoH [async]	22,283 s	84,717 ms	1,049 GB	12,377 GB/s

Table 2: Throughput for memory transfer of processed data from GPU to CPU. Again, the theoretical PCIe bandwidth can be reached.

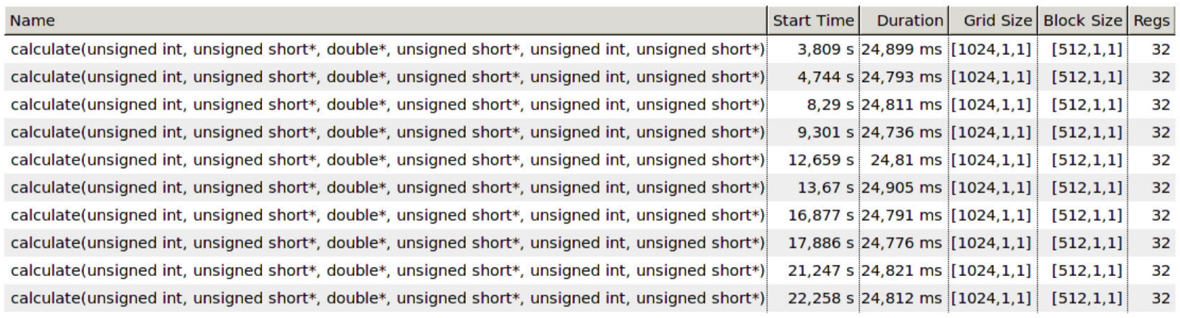
All performance measurements were performed on a single NVIDIA K80 GPU connected to host memory via PCIe Gen 3.

Both memory transfer of input data from the CPU host memory to the GPU memory and processed data from the GPU back to the CPU reach the maximum achievable PCIe transfer rate of 12 GB/s due to hiding the latency of the memory copy by using CUDA streams to overlap computation and memory transfer, see Tables 1+2.

In comparison, the duration of the algorithm for computing the gain maps Ecall from the raw ADC data takes on average less than 25 µs per image, see Table 3. This means that the

This project has received funding from the *European Union's Horizon 2020 research and innovation programme* under grant agreement No 654220

current single GPU solution is limited by the PCIe bandwidth available. Already today new interconnects such as NVLink are available that can reach more than ten times PCIe Gen. 3 throughput this limitation is likely to vanish. However, this example makes clear that there is potential for more complex operations on the data before reaching the minimum throughput acceptable.
In the current example the initial requirement of reaching a throughput of 6MB at 2 kHz rate comes close to the throughput limit given by the PCIe interconnect. The algorithm itself can be trivially scaled to accommodate for higher rates by adding more GPUs, as the computational demand scales linearly with the data rate. This makes it easy to adapt to larger detector setups comprising of multiple JUNGFRAU detector modules.



Name	Start Time	Duration	Grid Size	Block Size	Regs
calculate(unsigned int, unsigned short*, double*, unsigned short*, unsigned int, unsigned short*)	3,809 s	24,899 ms	[1024,1,1]	[512,1,1]	32
calculate(unsigned int, unsigned short*, double*, unsigned short*, unsigned int, unsigned short*)	4,744 s	24,793 ms	[1024,1,1]	[512,1,1]	32
calculate(unsigned int, unsigned short*, double*, unsigned short*, unsigned int, unsigned short*)	8,29 s	24,811 ms	[1024,1,1]	[512,1,1]	32
calculate(unsigned int, unsigned short*, double*, unsigned short*, unsigned int, unsigned short*)	9,301 s	24,736 ms	[1024,1,1]	[512,1,1]	32
calculate(unsigned int, unsigned short*, double*, unsigned short*, unsigned int, unsigned short*)	12,659 s	24,81 ms	[1024,1,1]	[512,1,1]	32
calculate(unsigned int, unsigned short*, double*, unsigned short*, unsigned int, unsigned short*)	13,67 s	24,905 ms	[1024,1,1]	[512,1,1]	32
calculate(unsigned int, unsigned short*, double*, unsigned short*, unsigned int, unsigned short*)	16,877 s	24,791 ms	[1024,1,1]	[512,1,1]	32
calculate(unsigned int, unsigned short*, double*, unsigned short*, unsigned int, unsigned short*)	17,886 s	24,776 ms	[1024,1,1]	[512,1,1]	32
calculate(unsigned int, unsigned short*, double*, unsigned short*, unsigned int, unsigned short*)	21,247 s	24,821 ms	[1024,1,1]	[512,1,1]	32
calculate(unsigned int, unsigned short*, double*, unsigned short*, unsigned int, unsigned short*)	22,258 s	24,812 ms	[1024,1,1]	[512,1,1]	32

Table 3: Duration of the algorithm computing Ecall for 1000 JUNGFRAU module images of 512x1024 pixels size.

Dissemination & Transfer within UFDAC / EUCALL

The current implementation of the algorithm in CUDA C was developed as a joint EUCALL project between PSI and HZDR. The code is open source based on the GPL vers. 3. ELI-ALPS has expressed interest in using the code base for their own application.

In order to foster transfer to other EUCALL partners we plan to scale the code to multiple GPUs and multiple nodes using the CRACEN library developed at HZDR and port the CUDA version to other architectures using the ALPAKA library developed at HZDR.

This project has received funding from the *European Union's Horizon 2020 research and innovation programme* under grant agreement No 654220