Fast processing of JUNGFRAU data with the Alpaka library

Jonas Schenke[1,2], Florian Warg[1,2], Anna Bergamaschi[3], Martin Brückner[3], Michael Bussmann[1], Carlos Lopez-Cuenca[3], Aldo Mozzanica[3], Sophie Redford[3], Bernd Schmitt[3], Heide Meißner[1]

[1]*Helmholtz-Zentrum Dresden – Rossendorf, Bautzner Landstraße 400, 01328 Dresden, Germany*
[2]*Technische Universität Dresden, 01069 Dresden, Germany*
[3]*Paul Scherrer Institut, 5232 Villigen-PSI, Switzerland*

Abstract
Highly segmented photon detectors play a crucial role at synchrotron and X-ray free electron laser facilities, detecting the photons scattered from targets of interest. These measurements allow the determination of biological and inorganic structures and processes, leading to an advanced understanding of life and material sciences and enabling improvements and discoveries in pharmaceutical and engineering industries. However, as targets become ever-more challenging and photon fluxes intensify, detector frame rates increase and the generated data streams risk becoming overwhelming. Here we demonstrate one solution, exploiting the highly parallelisable nature of data from the JUNGFRAU charge integrating photon detector and processing using the Alpaka library. We find that... To date, no other solution was able to process JUNGFRAU data at this throughput(?). Our results demonstrate that parallel processing, for example in a GPU, is a valuable tool in the treatment of photon detector data. Advantages of this method include online data correction aiding storage efficiency by allowing raw data to be immediately discarded and enabling fast feedback to users for whom time and target resources are at a premium. We see this as an important step in the development of a chain of specialised hardware and software tools necessary to operate charge integrating photon detectors at high frame rates.
Keywords: Photon pixel detector, fast data processing, GPU programming, Alpaka ...

1 Introduction

Data rates and volumes at photon science facilities are increasing due to many technological advancements, including higher beam intensities, robotic or serial sample delivery systems, and photon detectors capable of ever increasing frame rates. Charge integrating detectors compound the issue, as the raw detector output must be processed before the number of photons detected is retrieved. This requires advanced methods for fast, online data processing without which data volumes threaten becoming a limiting factor, quickly filling storage capacity, overwhelming computing processing power and slowing feedback to users.

The JUNGFRAU detector is one such charge integrating photon detector [21], which has been designed for SwissFEL, the free electron laser at the Paul Scherrer Institut in Switzerland [19]. Two 16 Mpixel JUNGFRAU detectors were installed at SwissFEL in 2018 and are now operational [22]. Due to the repetition rate and pulsed illumination of SwissFEL, the detectors are operated continuously with short (10 μs) integration time and at a frame rate of 100 Hz, meaning that each 16 Mpixel detector generates a reasonable data rate of 3.2 GB/s.

However, JUNGFRAU has also proved advantageous under challenging conditions at synchrotrons [16] where the illumination is continuous, requiring a full duty cycle from the detector and hence much higher frame rates. Recent experimental setups have included a one Mpixel detector operated continuously at 1.1 kHz (2.3 GB/s) with online visualisation but slow offline data treatment [16] and a four Mpixel detector operated at 1.1 kHz (9.2 GB/s) for short periods of time ($<$ 200 s) without visualisation but with fast data processing straight after acquisition (Japan, no ref yet). Our goal is to operate a 16 Mpixel detector continuously at 2.2 kHz, successfully receiving the raw data at 74 GB/s, providing an online visualisation, processing it online to convert the raw data into number of photons per pixel, and saving to disk. This necessitates specialised capability for receiving, processing and file writing, motivating this work.

The increasing data rates during FEL experiments require dedicated detectors as well as advanced methods for fast data processing. At PSI, the Jungfrau detector ([22], [24], [25], [21]) was developed which is capable due to a gain switching scheme to register single photons as well as photon bunches in a charge integration mode. A conversion of the detector data to the energy and the number of registered photons already during data acquisition is beneficial for the analysis of the measurement and furthermore for the reduction of storage space. This online data

conversion algorithm has to correct each pixel value using the gain maps and the continuously updated pedestal maps following a lab-based calibration procedure [24]. The summation of a couple of frames and the cluster identification in the data maps further reduce the amount of data. The different modules of one detector system can be processed fully parallel, however, the data conversion itself is not data parallel. Therefore, and due to the need for an advanced parallel implementation which is independent of the computing hardware, an Alpaka [18] version was written and its performance was tested.

TODO: Related work

common pre-processing steps:

- offset correction & flat field correction [DSSC [20], JF [24], EIGER[26] / [9], MOENCH [6], [MEDIPIX3RX [2]]]
- gain correction [DSSC [20], EIGER [26] / [9], MOENCH [6]]
- dynamic gain switching [AGIPD [1], GOTTHARD [23], JF [24], [MEDIPIX3RX [2]]
- pixel sorting (on FPGA)[AGPID [4]]
- discard uninteresting events (in readeout circuit)[DSSC [10]]
- photon counting [MEDIPIX3RX [2]]
- charge summing / interpolation (to counter charge sharing; on ASIC)[MEDIPIX3RX [2]]
- cluster finder [MOENCH [6]]
- sub pixel interpolation [MOENCH [6]]

detector data processing libraries (with pre-processing stage):

- Karabo [11]
  - developed to support experiments at the European XFEL
  - written in C++ and Python
  - pipelined design
  - supports offline and online operation and an additional rapid-feedback mode
  - can utilize multiple nodes
  - also implements file IO and a GUI
  - extensible through Karabo-Bridge
  - has OnDA interface
  - supports Jupytier notebook
- Hummingbird [8]
  - optimized for real-time usage
  - implemented in Python
  - main purpose: analysis of diffraction data at LCLS
  - achieves rates above 100 Hz
  - contains a GUI and supports multiple file streams at once
  - can utilize multiple computing nodes
  - can read data generated by psana
- OnDA [17]
  - supports online & offline operation
  - can utilize multiple computing nodes (via ZeroMQ and MPI)
  - master/worker architecture
  - one program instance per core
  - designed to give fast online feedback
  - main objective: real-time monitoring of X-ray imaging experiments
  - easily adaptable

  - contains a GUI
  - implemented in Python
  - can utilize a psana back end

- psana [7]
  - = Photon Science ANAlysis
  - similar functionality to pyana (which is written entirely in Python), but implemented mostly in C++ with a Python interface
  - used to analyze data produced by SLAC National Laboratory
  - can utilize multiple compute nodes (via MPI)
  - supports online & offline operation
  - contains detector calibration algorithms (e.g. pedestal calibration, bad-pixel determination, etc.) and GUI front end
  - designed to give fast online feedback

- Cheetah [3]
  - mode for XFEL applications
  - offline & online processing
  - evaluates data quality metrics (e.g. number of Bragg peaks etc.)
  - retain only frames with high likelihood of being usable
  - –¿ main task: data reduction
  - additional pre-processing
  - also has a GUI
  - implemented in C++

- CASS [12]
  - evaluate data acquired using the CFEL ASG Multi Purpose instrument (CAMP), specifically data collected at FLASH and the European XFEL
  - supports multiple detectors in online and offline scenarios
  - implemented in C++ with a Qt front end
  - consists of input ringbuffer and processing modules (which do pre- & post-processing)
  - GUI implemented in two additional programs: joCASSView (plots data) & luCASSView (interactive)
  - GUIs communicate with computation nodes via TCP/IP

- XDS [15]
  - in development since 1988
  - mainly deals with crystallography
  - also does pre-processing steps
  - no real-time/on-line capability

initialized data further used e.g. in crystallography [14] [27] [13]

In section 2.1, we review the design of Jungfrau detectors, followed by the description of the dedicated data conversion algorithm 2.3, its hardware-independent implementation 2.4 and benchmark tests in 2.5. The achieved results and conclusions are presented in sections 3 and 4.

2 Methods

2.1 JUNGFRAU detector description (PSI)

JUNGFRAU is a hybrid photon detector consisting of a 320 μm thick silicon sensor in which photons are absorbed and converted into an electrical signal, and an Application Specific Integrated Circuit (ASIC) chip in which the signal generated by the photons in the silicon is processed. Both sensor and ASIC layers are pixelated with 75 μm pitch, to provide two dimensional information of the position of the incident photons.

The JUNGFRAU ASIC is realised with UMC 110 nm CMOS technology and operates in the 'charge integrating' mode, meaning that the signal received by the ASIC within a well-defined, configurable time window (an exposure) is integrated and amplified [21]. Charge integrating detectors typically suffer from either a low dynamic range or poor resolution in the single photon regime. JUNGFRAU achieves both high dynamic range and single photon resolution by employing a gain switching technique, whereby the amplification factor of the initial photon-induced signal is dependent on the size of the signal itself. The gain switching happens automatically and independently per pixel and per exposure. The gain factor employed by each pixel (high, medium or low) in each exposure is encoded in a 2-bit output.

Each sensor has the dimensions of 1030×514 pixels, resulting in an active area of 7.7×3.8 cm^2. Eight 256×256 pixel ASICs are bump-bonded to each sensor in 4×2 arrangement to achieve an electrical connection in every pixel. Between the ASICs are two rows or columns of double-sized pixels, ensuring no gaps or loss of sensitivity. The eight ASICs are glued to a high density interconnect (HDI) and electrically connected with wire-bonds along the two long sides. Additional wire-bonds to the sensor provide the high voltage to bias the sensor, crucial for detecting the charge generated there by the absorbed photons. A readout board connects to the back of the HDI to collect the signals processed by the eight ASICs. Four 14-bit ADCs convert the incoming analogue signal to digital samples. An FPGA arranges this 14-bit signal information from each pixel together with the 2-bit gain information of each pixel into a 1 MB image (a frame) of each exposure. This is output as UDP packets through two 10 Gb optical fibres.

This single detector module can either operate alone, or can be tiled together with other detector modules to form part of a larger detector system, with typical sizes ranging from 1 Mpixel (two modules) to 16 Mpixels (32 modules). Figure 1 shows the front and back of a 4 Mpixel (8 module) detector. Each module retains its individual control (Ethernet), data (10 Gb optical fibres) and power connections. The only inter-module connection is a flat-ribbon cable used for triggering purposes. Data rates thus scale linearly with the number of modules.

While sending the UDP packets in parallel with 2×10 Gb/s per module is less challenging, receiving them in one or a few spots for further processing is more complex. Implementing this on a standard hardware with a non-optimised Linux system easily results in packet loss due to CPU and memory bandwidth limitations. Several avenues to resolve this issue are under development. Offloading the CPU can be achieved on a software level by bypassing the network stack, which avoids several memory copies. Another option is to offload the full receiving process into an FPGA, which receives the UDP packets with it's 100 Gb/s interface, reassembles full images and sends them over PCIe directly into host via DMA or GPU memory over GPUDirect. This goes beyond the scope of the current work, which focuses on processing the successfully received JUNGFRAU data.

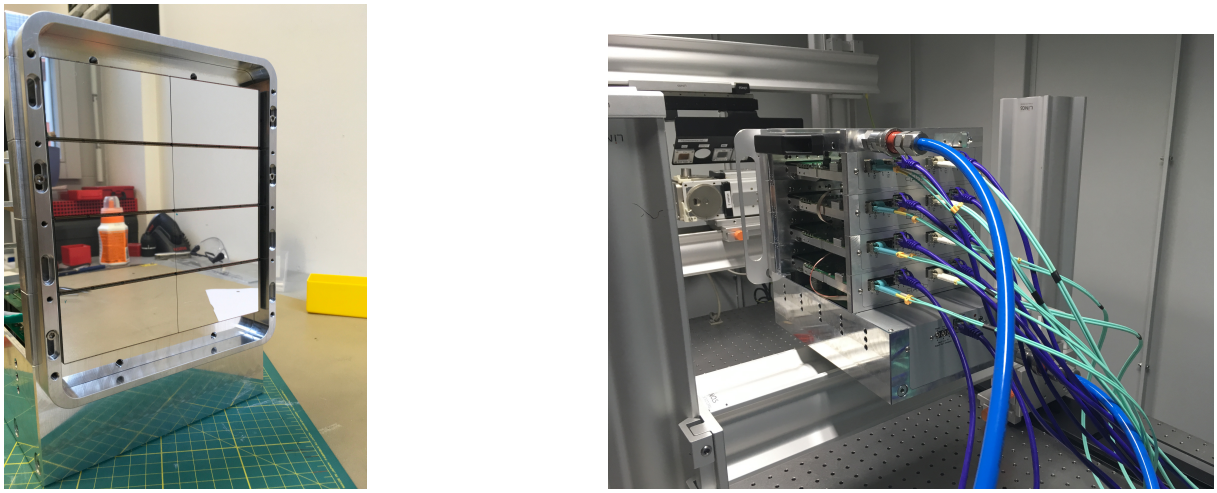

Figure 1: A 4 Mpixel JUNGFRAU detector shown from the front (left) and back (right). Photons impinge on the eight mirrored sensor surfaces, which are tiled to ensure minimal dead area. The readout boards plug into the back of the modules. Each has its own Ethernet (purple cables) and optical fibre (green cables). Water cooling (blue pipes) and power (black cable) are provided in common. A flat ribbon cable, visible on the left, connects to all readout boards for triggering purposes.

2.2 Raw data treatment (PSI)

To retrieve the information of how much energy, and in monochromatic situations, how many photons were detected in each pixel in each frame, a series of corrections must be applied to the raw 16-bit output of each pixel.

2.2.1 Pedestal correction and tracking

As any charge integrating detector, the JUNGFRAU detects some current in every pixel during every exposure, even if no photons were absorbed. This dark current or 'pedestal' is individual to each pixel, and is strongly dependent on the configuration of the detector, for instance the exposure time, and on environmental factors such as temperature. It is therefore measured by acquiring a series of dark frames before each measurement with photons. To measure the pedestals in medium and low gain, the detector is forced to switch gain during the dark frames. The ADC values recorded in the dark frames in high, medium and low gain are averaged to give a per-pixel, per-gain pedestal. A rolling average is preferred [ref cook], both to avoid committing long arrays of data to memory, and to place higher importance on more recent exposures.

In the case of steady operation at XFELs, with a short integration time and the detector at room temperature, recording the pedestals every few hours is sufficient. However in the case of operation at synchrotrons, with a long integration time and the detector chilled to $\sim -10^{\circ}C$, the pedestals are much more sensitive [25]. They must be recorded before every measurement with photons, and moreover, must be tracked through the measurement. This is achieved by filtering the recorded exposures, and updating the pedestal measurement of a particular pixel if the frame recorded a value within $\pm 2\sigma$ of the existing pedestal. This avoids exposures containing photon signals (typically $> 10\sigma$ from the noise) from biasing the pedestal.

2.2.2 Gain correction

The pedestal-corrected ADC samples are translated into energies, so taking into account the amplification of the original signal by the ASIC. The amplification factors are calculated per pixel and per gain in a dedicated laboratory based procedure during the commissioning of the detector [24]. Once the energy detected per pixel has been retrieved, in monochromatic situations this can be converted to a number of photons by normalising by the energy of the incident radiation, as shown in the following equation:

$$N_{\gamma} = \frac{\text{ADC output [ADU]} - \text{pedestal [ADU]}}{\text{gain [ADU/keV]} \times E_{\gamma}\text{[keV]}} \tag{1}$$

where N_{γ} is the number of photons detected, E_{γ} is the photon energy, and the pedestal and gain corrections are pixel dependent.

2.2.3 Pixel mask

A pixel mask can be defined using information from the calibration procedure and the pedestal calculation. Pixels can be non-responsive, or respond badly, for a number of reasons, including a chip level defect, bump bonding defect, wire-bond column defect or high leakage current. A pixel which could not be calibrated for any of these reasons, or could not have an accurate pedestal calculated, is masked in the later processing.

2.2.4 Frame summation

Depending on the application, further processing may be necessary. For macro-molecular crystallography, groups of subsequent frames (in our experience from 2 to 100 frames) can be summed together before saving [16]. Depending on the rotation speed of the sample, this can improve later analysis to determine crystal structure. It also acts as an effective compression of the data.

2.2.5 Pixel clustering

In the case of soft X-rays at low incoming flux, pixels may be clustered to draw additional positioning information from the data. Description needed [5].

2.3 Data processing algorithm (PSI)

The following enumerates the typical data acquisition and processing steps for a single measurement with photons in a synchrotron environment, which is the most challenging in terms of data rate and which requires pedestal tracking.

1. acquisition of 1000 dark frames in forced switching to medium gain. The medium gain pedestal is calculated and pixels not in medium gain are added to the pixel mask.

2. acquisition of 1000 dark frames in forced switching to low gain. The low gain pedestal is calculated and pixels not in low gain are added to the pixel mask.

3. acquisition of at least 1000 dark frames in gain switching mode. Using the first 1000 frames, the high gain pedestal is calculated and pixels not in high gain are added to the pixel mask. At 1000 frames, pixels with larger than normal noise (pedestal RMS) are also added to the pixel mask and the noise level per pixel in high gain is taken as the pedestal tracking threshold. After 1000 frames, the high gain pedestal is tracked and the detector and software is 'ready and waiting for photons'.

4. without stopping and restarting the detector, the beamline shutter opens and frames with photons are acquired in gain switching mode. Frames must be converted to number of photons per pixel. High gain pedestals are tracked. Frames can be optionally summed or clustered before being saved.

Whether online or offline, whether implemented in CPU, GPU, FPGA or other hardware, this processing pipeline must be realised in order to convert the raw data output from the JUNGFRAU into information in terms of number of photons per pixel, for use in later analyses.

2.4 Alpaka implementation of fast data processing (Jonas)

requirements:

- real-time data processing (at least 1 detector at 2.5 kHz)
- portable to other architectures (CPU OpenMP, TBB, ...) with minimal code changes
- library for easy integration into existing projects

optional requirements:

- easily adaptable to similar detectors (e.g. MÖNCH)

expected bottlenecks:

- communication speed between host and device
- RAM speed

features:

- automatic pixel masking (wrong gainstage in calibration; stddev threshold; user supplied mask)
- max value extraction
- pedestal updates → runtime fallback to initial pedestalmaps also implemented

design overview:
upload strategies:

- split frames and upload them → very inefficient
- upload blocks sequentially → inefficient because application is memory bound
- pipeline copy and kernel operations
- use alpaka to write platform independent code –> explain alpaka shortly (?)
- written in modern C++ for performance
- pipelined stream processing of data: upload data to GPU -> process -> download again
- multiple detectors and multiple GPUs usable in parallel

<insert data flow diagram here>
architecture specific optimizations:

- contiguous memory vs. grid striding loop
- only upload data if needed

software architecture:
hardware accelerator concept

- host
- device
- memory not necessarily shared
- data might need to be copied onto device
- device starts (possibly asynchronous) execution
- result is copied back to host
- common host: CPU
- common devices: GPUs, FPGAs, Multicore CPUs (in general multi- and manycore systems)

<insert hardware sketch here>
the dispenser possible work distribution

- thread pool → unnecessarily complex
- source-sink relationship → every source has a dedicated sink in which data is written → enqueueing of source sink pairs
- handles devices (initialization, memory management, etc.) using a ringbuffer
- usage of double buffering (input data in user provided buffers which is then uploaded to the device, calculated into a new device buffer and downloaded into a user provided output buffer) → minimal number of buffers; user is in charge of buffer handling
- enqueues data to next free devices (asynchronously)
- copies data back from devices (asynchronously)

<insert dispenser structure here>
the kernel structure

- written in empty struct as static function
- all needed values passed to function, no real access to member variables
- most data passed by pointer, only single numbers passed directly
- kernel has access to device properties (block size, grid size, device specific functions, etc.)

<insert example here>
easy hardware abstraction

- useful settings defined for every possible device class
- device is selected by using statement (similar to a typedef)
- no additional changes needed

<insert example here>

2.5 Benchmark tests (Jonas)

Design, objectives, and evaluation of tests

- machine specks
- used libraries
- which tests where chosen?
- why were they chosen?
- what are the expectations?

3 Results

3.1 Achieved improvements (Jonas)

Results of tests of software parts on various computing hardware using suitable Alpaka backends

Where are the bottlenecks

expected:

- RAM and PCIe bandwidth @ ˜12 GB/s, after that: multiple nodes necessary
- unbalanced up- and downloads (mainly on GPU)

Available capacity on GPUs

expected: up to 1000 Frames/s; 100 to 1000 Frames per stream
comparison of devices (expected):

- after 10 Frames per stream the frame count has diminishing effects on performance (on GPU)
- on CPU: number of Frames per streams doesn't effect performance much
- photon calculation a lot faster (˜4x) than energy; clustering in between (this effect will probably be less on CPU)
- number and type of kernels mostly insignificant for GPU; some effects on CPU
- frame summing will speed up almost linearly on less frames on GPU, on CPU not so big differences
- almost linear scaling with number of GPUs
- almost linear scaling with number of cores on CPU
- pedestal updates and masking doesn't effect performance mush
- exception: clustering (mainly on GPU): scales linearly with clusters to download calculate
- multiple modules show weak scaling
- effects of cluster size???

Best system
expected: 4x V100 System w/ 100 or 1000 frames per stream @ > 10000 Frames/s

3.2 Experiences from practical application of improved code (PSI)

Application results

4 Conclusions and Outlook

Is presented method applicable / useful for other detectors, e.g. AGIPD?

applicable to data from similar sensors (rectangular sensor, consecutive in memory)
calculation on FPGAs in the future
more backends to come (SYCL, ...)

5 Acknowledges

The authors would like to thank the Alpaka developers (names?)for their support.

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

References

[1] A Allahgholi et al. “AGIPD, a high dynamic range fast detector for the European XFEL”. In: *Journal of Instrumentation* 10.01 (2015), p. C01023.

[2] Rafael Ballabriga et al. “The Medipix3RX: a high resolution, zero dead-time pixel detector readout chip allowing spectroscopic imaging”. In: *Journal of Instrumentation* 8.02 (2013), p. C02016.

[3] Anton Barty et al. “Cheetah: software for high-throughput reduction and analysis of serial femtosecond X-ray diffraction data”. In: *Journal of applied crystallography* 47.3 (2014), pp. 1118–1131.

[4] J Becker et al. “High speed cameras for X-rays: AGIPD and others”. In: *Journal of Instrumentation* 8.01 (2013), p. C01042.

[5] S. Cartier et al. “Study of the signal response of the MÖNCH 25μm pitch hybrid pixel detector at different photon absorption depths”. In: *Journal of Instrumentation* 10.03 (2015), pp. C03022–C03022. DOI: 10.1088/1748-0221/10/03/c03022. URL: https://doi.org/10.1088%2F1748-0221%2F10%2F03%2Fc03022.

[6] Sebastian Cartier et al. “Micrometer-resolution imaging using MÖNCH: towards G2-less grating interferometry”. In: *Journal of synchrotron radiation* 23.6 (2016), pp. 1462–1473.

[7] D Damiani et al. “Linac Coherent Light Source data analysis using psana”. In: *Journal of Applied Crystallography* 49.2 (2016), pp. 672–679.

[8] Benedikt J Daurer et al. “Hummingbird: monitoring and analyzing flash X-ray imaging experiments in real time”. In: *Journal of applied crystallography* 49.3 (2016), pp. 1042–1047.

[9] Roberto Dinapoli et al. “EIGER characterization results”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 731 (2013), pp. 68–73.

[10] Florian Erdinger et al. “The DSSC pixel readout ASIC with amplitude digitization and local storage for DEPFET sensor matrices at the European XFEL”. In: *2012 IEEE Nuclear Science Symposium and Medical Imaging Conference Record (NSS/MIC)*. IEEE. 2012, pp. 591–596.

[11] Hans Fangohr et al. “Data analysis support in Karabo at European XFEL”. In: (2018).

[12] Lutz Foucar et al. “Cass—cfel-asg software suite”. In: *Computer Physics Communications* 183.10 (2012), pp. 2207–2213.

[13] Ralf W Grosse-Kunstleve et al. “The Computational Crystallography Toolbox: crystallographic algorithms in a reusable software framework”. In: *Journal of Applied Crystallography* 35.1 (2002), pp. 126–136.

[14] Wolfgang Kabsch. “Integration, scaling, space-group assignment and post-refinement”. In: *Acta Crystallographica Section D: Biological Crystallography* 66.2 (2010), pp. 133–144.

[15] Wolfgang Kabsch. “Xds”. In: *Acta Crystallographica Section D: Biological Crystallography* 66.2 (2010), pp. 125–132.

[16] Filip Leonarski et al. “Fast and accurate data collection for macromolecular crystallography using the JUNGFRAU detector”. In: *Nature Methods* (2018). DOI: 10.1038/s41592-018-0143-7. URL: https://doi.org/10.1038/s41592-018-0143-7.

[17] Valerio Mariani et al. “OnDA: online data analysis and feedback for serial X-ray imaging”. In: *Journal of applied crystallography* 49.3 (2016), pp. 1073–1080.

[18] A. Matthes et al. “Tuning and Optimization for a Variety of Many-Core Architectures Without Changing a Single Line of Implementation Code Using the Alpaka Library”. In: *High Performance Computing*. Ed. by Julian M. Kunkel et al. Cham: Springer International Publishing, 2017, pp. 496–514. ISBN: 978-3-319-67630-2.

[19] Christopher J. Milne et al. “SwissFEL: The Swiss X-ray Free Electron Laser”. In: *Applied Sciences* 7.7 (2017). ISSN: 2076-3417. DOI: 10.3390/app7070720. URL: http://www.mdpi.com/2076-3417/7/7/720.

[20] David Moch et al. “Calibration of the Non-Linear System Characteristic of a Prototype of the DSSC Detector for the European XFEL”. In: *2014 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC)*. IEEE. 2014, pp. 1–4.

[21] A. Mozzanica et al. “Characterization results of the JUNGFRAU full scale readout ASIC”. In: *Journal of Instrumentation* 11.02 (2016), pp. C02047–C02047. DOI: 10.1088/1748-0221/11/02/c02047. URL: https://doi.org/10.1088%2F1748-0221%2F11%2F02%2Fc02047.

[22] A. Mozzanica et al. “The JUNGFRAU Detector for Applications at Synchrotron Light Sources and XFELs”. In: *Synchrotron Radiation News* 31.6 (2018), pp. 16–20.

[23] Aldoa Mozzanica et al. “The GOTTHARD charge integrating readout detector: design and characterization”. In: *Journal of Instrumentation* 7.01 (2012), p. C01019.

[24] S. Redford et al. “First full dynamic range calibration of the JUNGFRAU photon detector”. In: *Journal of Instrumentation* 13.01 (2018), pp. C01027–C01027.

[25] S. Redford et al. “Operation and performance of the JUNGFRAU photon detector during first FEL and synchrotron experiments”. In: *Journal of Instrumentation* 13.11 (2018), pp. C11006–C11006.

[26] Gemma Tinti et al. “Electron crystallography with the EIGER detector”. In: *IUCrJ* 5.2 (2018), pp. 190–199.

[27] Thomas A White et al. “CrystFEL: a software suite for snapshot serial crystallography”. In: *Journal of applied crystallography* 45.2 (2012), pp. 335–341.