Realtime-Processing of Nanocrystallography Images

Daniel Becker
University of Applied Science
Berlin, Germany
Email: beckerd@htw-berlin.de

Achim Streit
Steinbuch Center for Computing
Karlsruhe Institute of Technology, Germany
Email: achim.streit@kit.edu

***Abstract*—In nanocrystallography, diffraction images are captured to gain insights into the structure of macromolecules. A new generation of experiments is able to take a vast amount of images in a short time. However, most of the images are not suitable for further research. It is not feasible to store and process all images in a reasonable amount of time. In previous work we proposed algorithms able to distinguish useful from useless data in photon science. In this article we propose and discuss a prototype implementation of the algorithms including further optimizations. We also consider its feasibility to cope with the realtime constraints.**

***Keywords*—image processing; photon science; big data; signal processing; X–ray microscopy; nanocrystallography; realtime processing.**

I. Introduction

In nanocrystallography the inner structure of macromolecular samples such as proteins are explored. In the process of analysis, samples are crystallized and then illuminated by an X-ray light source. Due to the high energy of the laser flash, the samples are destroyed within femtoseconds. Since many images with different orientations of the sample are necessary to calculate a three-dimensional model of the sample, a sample transportation system is used. The typical setup of an experiment can be seen in Fig. 1. Samples are transported by a liquid jet which crosses the X-ray laser beam. The diffracted light is then captured by a detector device.

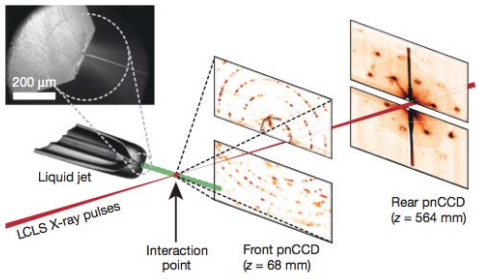

Fig. 1. Typical setup of a nanocrystallography experiment. [1].

One example for such an experiment is the Linac Coherent Light Source (LCLS) in Stanford [2]. Here, data are taken at a rate of 120 images per second. However, due to the inability of synchronizing the light flashes with the stream of probes, not all images taken are suitable for further research. Depending on the experimental run, up to $96.4\,\%$ of the data taken are not suitable for further research [3]. This consumes computational resources as well as storage space. In addition, this practice is not feasible for next generation experiments.

The European XFEL experiment, for example, will be able to generate up to $27{,}000$ images per second [4]. Given this high velocity, it will not be possible to store and process all data in a short amount of time. Therefore, solutions have to be found rejecting as many useless data as early and as close to the detector as possible.

In previously published article, we explored algorithms designed to categorize images (see Sec. I-A2) and identify signals with respect to the specific characteristics of nanocrystallography (see Sec. I-A3). The signals are called Bragg spots and consist of small areas where a comparatively large amount of light is registered.

In this article, we are proposing a prototypical implementation of a combination of both algorithms. The individual steps and their implementation is discussed. We then explore the recognition rates as well as the runtime behavior. Finally, we propose a multi-step solution for handling the high data rate of the European XFEL experiment in realtime.

A. Related Work

1) Cheetah: The Cheetah software [5] has been developed to process data taken in X-ray diffraction nanocrystallography. It is written in C++ as a library to be independent of the data input format. Before the analysis of an image takes place, individual characteristics of the detector used to capture the data can be configured. Pixels can be marked as broken or a custom offset can be set. Additionally, physical effects like a water halo[1] within the image as well as additional noise around the beam hole can be set to be ignored. Cheetah then searches for connected pixels above a preset threshold. The signals identified are stored along with their coordinates and intensity. Cheetah stores images as individual files in the Hierarchical Data Format 5 (HDF5) [6]. An image is stored as a matrix. Each component of the matrix holds a 14-bit integer value representing the intensity registered by a detector pixel (grayscale) [7].

[1]Circle of diffracted light by the transportation liquid of the stream of probes.

978-1-5090-0888-9/16 $31.00 © 2016 IEEE
DOI 10.1109/UKSim.2016.20

2) Neural Network as a Veto Engine: In our article 'Neural Network as a Veto Engine' [8] we propose a neural network able to reliably categorize data taken in nanocrystallography. To categorize the data, three basic quantities are extracted from the images. Those quantities are then used as an input vector for a neural network. The networks output is the likelihood of the suitability of the image for further analysis. In addition, two image optimizations are introduced. Firstly, background subtraction, which calculates an average noise level of blank images and then subtracts it before processing an image. Secondly, a new quantity has been introduced called the 'transverse intensity'. It represents a factor compensating for the loss of photon intensity towards the outer areas of the detector.

3) Clusterfinder: In the article 'Localization of Signal Peaks in Photon Science Imaging' [9] we introduce a multi-step algorithm able to detect signals within images from nanocrystallography. The signal detection is implemented without any knowledge of the physics. In the first step, random noise spots are removed by applying a suitable convolution. In a second step, edge detection is used to enhance signals even further. Finally, connected pixels above a preset threshold are identified recursively.

II. Data & Methods

A. Samples

To verify our results, diffraction images from three different samples are analyzed. The samples are

- the protein *Cathepsin B* (CatB) [10],
- the *5-Hydroxytryptamine receptor 2B* (5HT-2B) [11],
- the *granulovirus polyhedron* (GV) [12].

The images of the samples are provided as individual files in the HDF5 format. No corrections or optimizations have been applied beforehand.

B. Prototype

The workflow of our prototype is shown in Fig. 2. In the following paragraphs, the individual components are discussed.

1) Normalize Image: Since pixels of the detector can get stuck or break, it is necessary to ignore them during analysis. Therefore, all pixels outside the 14-bit unsigned integer range (0 - 16,384) are set to zero.

2) Remove Single Pixel Noise: Throughout each image, random noise is spread in the form of single bright pixels. These are dampened using convolution. A small image kernel is applied to each pixel. Typically, an image kernel is a 3×3 or 5×5 matrix. The convolution

$$I' = I * K \tag{1}$$

is defined as

$$I'_{x,y} = \sum_{m=1}^{M'} \sum_{n=1}^{N'} I_{x-m+2,y-n+2} K_{m,n} \ , \tag{2}$$

where I is the original, two-dimensional image and K represents the image kernel. In our case, the dimension of the image matrix I is 1552×1480. There are several ways to deal with pixels at the border of images. In our case, the border area of an image is very unlikely to contain any useful information. Therefore, the limits $1 < x < M$ and $1 < y < N$ are applied.

To dampen this kind of noise, the image kernel

$$K = \frac{1}{9} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \tag{3}$$

is used. Since the sum of the entries is 1, the intensity of each pixel is distributed to its neighbors and no intensity is lost.

In view of a possible parallelization of this step, it has been implemented in OpenCL running on a GPU rather than the CPU. For each pixel, a separate work item (thread) is created and processed by the GPU in parallel.

3) Calculate Basic Data: To classify an image a variation of the neural network proposed in [8] is used. Only one output neuron is used to express the likelihood of an image being suitable for further analysis. Three basic quantities are extracted from the image. In this step, the maximum intensity $I_{\max}$ of a single pixel, the average intensity I_{mean}, and standard deviation ΔI of all pixels is calculated in one iteration.

4) Calculate Neural Network Output: Once the three basic data have been calculated, the weights of the trained neural network can be applied to calculate the output of the network for the image

$$\mathrm{Rating} = \mathrm{S}(\mathrm{I_{max}w_1} + \mathrm{I_{mean}w_2} + \Delta \mathrm{Iw_3} + \mathrm{w_4}) \tag{4}$$

where S is the sigmoid function, which is defined as

$$S(x) = \frac{1}{1 + e^{-x}} \ . \tag{5}$$

w_4 is attached to the bias neuron, which is set to 1. It is used to improve the approximation of the neural network. An image is considered as 'useful' if its rating value is above 0.5.

5) Apply Binary Edge Detection: Edge detection is applied to images categorized as 'useful' by the previous step. The shape of the signals in an image may vary. What they have in common however is a rapid increase of intensity within the range of a few pixels. This characteristic can be exploited by applying edge detection. To detect edges in an image, convolution is used as described in Sec. II-B2. Two asymmetrical kernels are used to identify a sudden increase or decrease in intensity between adjacent pixels. The kernels are

$$S_h = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \tag{6}$$

and

$$S_v = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \ . \tag{7}$$

They are known as the Sobel operator [13] and are applied by

$$\Delta I_{x,y} = \sqrt{\left(I^{(\mathrm{noise\_reduced})} * S_h\right)^2_{\mathrm{x,y}} + \left(I^{(\mathrm{noise\_reduced})} * S_v\right)^2_{\mathrm{x,y}}} \tag{8}$$

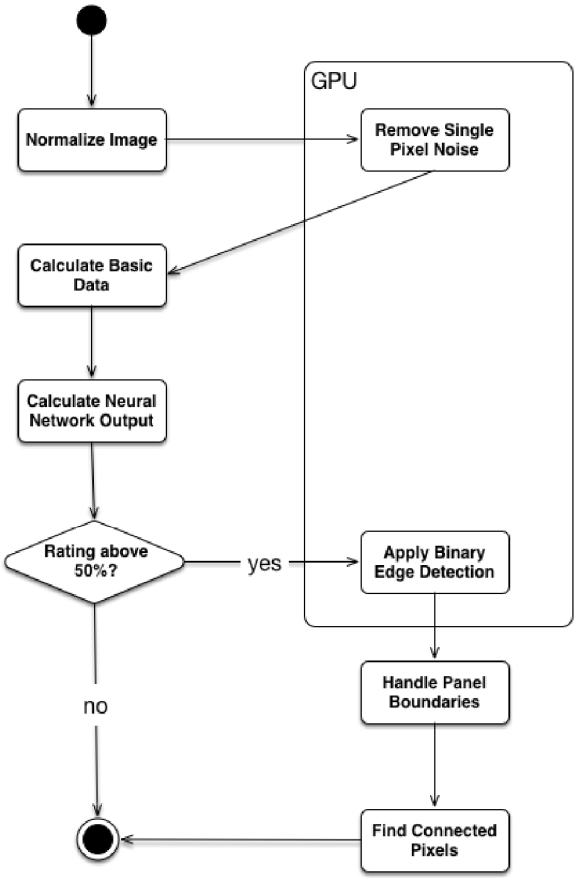

Fig. 2. Workflow of the proposed prototype.

where $\Delta I_{x,y}$ is the gradient of the intensity of the pixel at the coordinate x, y of the image I.

To separate signals from noise, a threshold has to be applied. To conserve as many iterations in the following steps as possible, thresholding is applied directly after edge detection. Here, the step function

$$\theta_{\text{threshold}}(I_i) = \begin{cases} 0, I_i < \text{threshold} \\ 1, I_i \geq \text{threshold} \end{cases} \tag{9}$$

is applied to each pixel, where I_i denotes the intensity of the i-th pixel.

6) Remove Empty Grid: The detector is composed of many individual panels [14]. Around each panel, a row and column of zeros is recorded. This results in a rapid change in intensity, which might trigger a false positive. To avoid this, the intensities at the panel borders are set to zero. In addition, all remaining pixels with the value 1 but without any adjacent pixel are set to 0. This is done because once edge detection has been applied, a single pixel can never represent a valid signal.

7) Find Connected Pixels: In the last step, connected pixels are detected using a recursive algorithm. Each pixel that has been inspected is set to 0 to mark it as treated and prevent the algorithm from considering it again.

III. Results

A. Recognition Rates

To verify our prototype, 10 manually preselected images of each sample have been analyzed. In a first step, we categorized the images as 'indexable'[2] and 'not-indexable'[3] based on the calculated output of the neural network. All images for which we received an output value of > 0.5 were considered as containing data. To these images, the modified version of the clusterfinder algorithm is applied in order to identify all Bragg Spots along with their coordinates and intensities. The spots found were then compared to the ones found using the Cheetah software (see I-A1). The results can be found in Tab. I and II. The rather low amount of signals detected by our proposed prototype is due to a conservative intensity threshold, ensuring as few false positives as possible.

[2]Suitable for further research.
[3]Not suitable for further research.

TABLE I
SPOTS FOUND IN INDEXABLE IMAGES

	Total Spots Found	Spots in Common	Spots Cheetah Only	Spots Prototype Only
CatB	442	294	118	30
5HT-2B	465	190	207	68
GV	871	265	419	187

TABLE II
SPOTS FOUND IN INDEXABLE IMAGES

	% Spots in common	% Spots Cheetah only	% Spots Prototype only
CatB	67%	27%	7%
5HT-2B	41%	45%	15%
GV	30%	48%	21%

TABLE III
SPOTS FOUND IN INDEXABLE IMAGES

	Total Spots Found	Spots in Common	Spots Cheetah Only (valid)	Spots Prototype Only (valid)
CatB	36	20	35 (3)	0
5HT-2B	52	44	22(15)	11 (6)
GV	95	21	51(33)	23 (8)

Besides the signals both algorithms found, each algorithm detected signals the other one did not. To explore this behavior, we took two images from each sample and verified the signals manually. The results for these two images per sample can be found in Tab. III. It can be seen that both algorithms find additional valid spots as well as false positives. We found that the false positives in case of the prototype are a result of defective pixels within a panel and the high amount of noise around the beam hole. Cheetah on the other hand identified false positives mostly at panel edges and within the water halo. In addition, we found that the valid spots identified by either algorithm were almost exclusively very weak. Therefore, their identification differs with respect to optimization handling.

B. Runtime

To benchmark the prototype, it has been compiled using Apple LLVM version 7.0.2 (clang 70.1.81) on Mac OS 10.11.3. No compiler optimizations are used and the runtime is measured using the Apple 'Instruments' software. The software is part of the Apple XCode IDE [15]. In Instruments, we used the 'time profiler' instrument to determine the execution time for each step. Each run of the application is performed individually and the specified amount of panels of a single image is processed in each run.

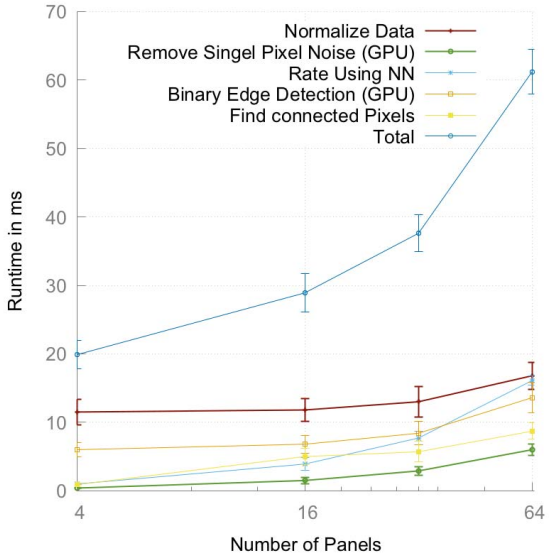

Fig. 3. Runtime of the prototype for different panel counts. The error bars represent the standard deviation.

The execution time is measured once an image is loaded into memory. The results of each step are written into a new matrix. In steps involving the GPU for processing, the data transfer between the host and GPU is recorded as well.

The average recorded runtimes can be found in Tab. IV along with the standard deviation for each step. The runtime is plotted in Fig. 3. The error bars represent the standard deviation. The runtime, in general, is decreasing for fewer panels. However, is is also visible, that the steps running on the GPU are tending to saturate. This is most likely due to the overhead of copying data between host and GPU. The runtime of the other steps is decreasing almost linearly with the inverse number of panels. The rather high runtime of the 'Rate using Neural Network' step might be explained by the need for iterating over each pixel of the image and calculating two sums in order to determine the average intensity and its standard deviation. This is compatible with the observed $\mathcal{O}(n^2)$ behavior of the runtime. The technical specs of the system the benchmark has been run on can be found in Tab. V. The influence of the 'Turbo Boost' technology is not explored in this article.

The amount of images processable per second is given by the fraction of an image processed by one process multiplied with the total execution time and the amount of analysis necessary to process a whole image. The system we used for benchmarking is equipped with two cores and can process two half images in parallel. According to Tab. IV the total runtime per process for 32 panels, which corresponds to $50\,\%$ of the image, is $37.6\,\mathrm{ms}$. This results in

$$\frac{\frac{1}{2}\mathrm{image}}{37.6\,\mathrm{ms}} \times 2\ \mathrm{Cores} = 26.6\ \frac{\mathrm{images}}{\mathrm{s}}\,. \tag{10}$$

The hardware we used for benchmarking does not repre-

TABLE IV
AVERAGE RUNTIME (AVG) FOR 10 RUNS OF EACH STEP INCLUDING THE STANDARD DEVIATION (SD).

Step	64 Panels		32 Panels		16 Panels		4 Panels	
	AVG	SD	AVG	SD	AVG	SD	AVG	SD
Normalize Image	16.8 ms	2.0 ms	13.0 ms	2.2 ms	11.8 ms	1.7 ms	11.5 ms	1.9 ms
Remove Single Pixel Noise	6.0 ms	0.8 ms	2.9 ms	0.6 ms	1.5 ms	0.5 ms	0.4 ms	0.0 ms
Rate using NN	16.1 ms	0.74 ms	7.7 ms	0.99 ms	3.9 ms	0.85 ms	1.0 ms	0.09 ms
Binary Edge Detection	13.6 ms	2.2 ms	8.4 ms	1.7 ms	6.8 ms	1.3 ms	6.0 ms	1.0 ms
Find connected Pixels	8.7 ms	1.2 ms	5.7 ms	1.5 ms	5.0 ms	1.2 ms	0.9 ms	0.2 ms
Total	61.2 ms	3.29 ms	37.6 ms	2.67 ms	28.9 ms	2.8 ms	19.9 ms	2.1 ms

TABLE V
TECHNICAL SPECS OF THE SYSTEM USED FOR BENCHMARKING

CPU	Intel I7-4650U 1.7 GHz (TurboBoost 3.3 GHz), 2 Cores
GPU	Intel HD Graphics 5000, 1.5 GB VRAM, 40 execution units
RAM	8 GB DDR 1600 MHz

TABLE VI
TECHNICAL SPECS OF THE SYSTEM USED FOR ANALYZING DIFFRACTION IMAGES AT CFEL HAMBURG.

CPU	24x Intel(R) Xeon(R) CPU X7542 2.67 GHz 6 Cores
RAM	768 GB

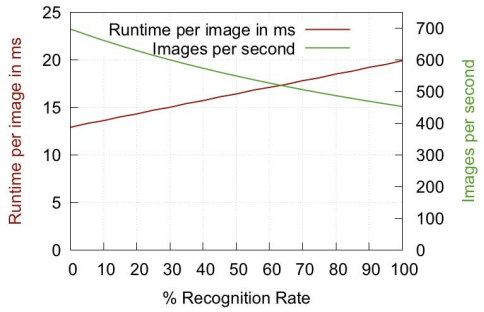

Fig. 4. Runtime of the prototype for different average efficiencies.

sent real world conditions. Much more powerful systems are currently used to process images in nanocrystallography. The Center for Free-Electron Laser Science (CFEL) in Hamburg, for example, uses a SGI system with 24 CPUs and 144 physical cores, see Tab. VI. Assuming the same processing speed for each step of the prototype, we can extrapolate our results. Using all 144 cores of the system and splitting each image into 16 parts (4 panels each), it is possible to process 16 images in parallel. According to Tab. IV, 4 panels can be processed in 19.9 ms. This leads to a processing speed of

$$\frac{\frac{1}{16}\text{image}}{19.9\,\text{ms}} \times 144\ \text{Cores} = 452.3\ \frac{\text{images}}{\text{s}}\ . \qquad (11)$$

This calculation still assumes an efficiency of 100 %, i.e. each image is assumed to be 'useful'. As mentioned in Sec. I, the hit rates may even be smaller than 5 %. For experimental runs producing images with high contrast and comparatively low noise, the efficiency might be as high as 50 %.

Since for each image which is categorized as 'not-indexable', the steps 'Edge Detection' and 'Find connected Pixels' do not have to be applied, the total execution time is shorter. This reduces the processing time for this kind of image to 13 ms, see Tab. IV. In Fig. 4, the average runtime for an efficiency between 100 % and 0 % is shown. It can be seen that the amount of images per second increases to up to 692 images per second at 0 % efficiency (where no image contains a signal). This rate is still not sufficient to meet the realtime demands at the European XFEL experiment.

The processing speed could be improved further by utilizing additional hardware close to the detector. The article [16] shows that an 1600×1200 pixel image can be processed by FPGAs in 0.684 ms. One panel of the detector consists of 194×185 pixel. Therefore, the processing time can be reduced to

$$\frac{194 \times 185\ \text{pixel}}{1600 \times 1200\ \text{pixel}}\ 0.684\,\text{ms} = 0.0128\,\text{ms}\ . \qquad (12)$$

This translates to a processing speed of 77,810 images per second using a desktop-class GPU, which is almost three times the image rate of the European XFEL. This, in turn, means that all convolution operations could be carried out by dedicated hardware close to the detector using GPUs. This would save 6.4 ms processing time. In addition, FPGAs connected to the

detector panels could be used to normalize the data directly after they are taken. This would save additional $11.5\,\mathrm{ms}$ of processing time.

The remaining steps to be carried out by more complex hardware would then be 'Rate using Neural Network' ($1\,\mathrm{ms}$) and 'Find connected Pixels' ($0.9\,\mathrm{ms}$). Using Eq. 11, this translates to

$$\frac{1}{16}\,\frac{1}{1.9/1000}\,144\,\frac{\text{images}}{\text{s}} = 4736.8\,\frac{\text{images}}{\text{s}}\,. \qquad (13)$$

This corresponds to $17.5\,\%$ of the image rate of the European XFEL. Here, we assumed that every diffraction image contains 'useful' data. As already mentioned several times, most of the images are 'blank' and contain only noise as the samples are not hit by a laser flash. This means that given an efficiency below $17.5\,\%$, it is possible to process all images taken by the European XFEL in realtime.

IV. Conclusion

In this article we proposed an implementation of two algorithms presented in earlier work. The prototype has been explored in terms of the recognition rate of signals as well as runtime behavior. The recognition rate is compared to the software currently in use (Cheetah) and to signals identified manually. Signals that are well separated from the background noise are reliably identified by our prototype as well as Cheetah. For weaker signals and signals at the border of the detector, we found that the Cheetah software returned false positives whereas our prototype on the other hand only returned false positives in case of defective areas of the detector, which can be dismissed if the coordinates of these areas are known.

We also explored the runtime behavior of our prototype. Here, we were able to show that it is possible to run all steps on parts of the image in parallel. We demonstrated an increase in processing speed. For steps implemented on the GPU we observed a saturation of the speedup, most likely due to memory operations. The runtime of the other steps increased nearly linearly with the inverse size of the image. Using the measured runtime on our system used for benchmarking we extrapolated the results to hardware currently in use to process images in nanocrystallography. We proposed a multilevel step setup for processing data in nanocrystallography. The image rate of the European XFEL experiment should be processable by our proposed solution.

In our experiments it was necessary to supply a threshold for signals manually. Ideally, it should be possible to determine a reasonable threshold dynamically. This eliminates the need for an expert reviewing the data. A basis for such a dynamic threshold might be extractable from the data provided by our neural network. It should be explored to which extend the average intensity in combination with the standard deviation could be used to calculate a dynamic threshold. In addition, it would be desirable to get a better understanding of the origin of noise within the images. By a close cooperation with researchers working directly at an experiment it might be possible to derive a model or simulation for the different sources of noise. This could then be used to remove most of the noise, and thus increase the recognition rates and processing speed.

Acknowledgment

This work is supported by the Portfolio Extension of the German Helmholtz Association "Large Scale Data Management and Analysis"[17].

References

[1] H. N. Chapman, P. Fromme, A. Barty, T. A. White, R. A. Kirian, A. Aquila, M. S. Hunter, J. Schulz, D. P. DePonte, U. Weierstall *et al.*, "Femtosecond x-ray protein nanocrystallography," *Nature*, vol. 470, no. 7332, pp. 73–77, 2011.

[2] J. N. Galayda, "The linac coherent light source project," SLAC-PUB-10115, Aug, Tech. Rep., 2002.

[3] W. Liu, D. Wacker, C. Gati, G. W. Han, D. James, D. Wang, G. Nelson, U. Weierstall, V. Katritch, A. Barty *et al.*, "Serial femtosecond crystallography of g protein–coupled receptors," *Science*, vol. 342, no. 6165, pp. 1521–1524, 2013.

[4] R. Klanner, J. Becker, E. Fretwurst, I. Pintilie, T. Pöhlsen, J. Schwandt, and J. Zhang, "Challenges for silicon pixel sensors at the european xfel," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 730, pp. 2–7, 2013.

[5] A. Barty, R. A. Kirian, F. R. Maia, M. Hantke, C. H. Yoon, T. A. White, and H. Chapman, "Cheetah: software for high-throughput reduction and analysis of serial femtosecond x-ray diffraction data," *Journal of Applied Crystallography*, vol. 47, no. 3, pp. 1118–1131, 2014.

[6] M. Folk, A. Cheng, and K. Yates, "Hdf5: A file format and i/o library for high performance computing applications," in *Proceedings of supercomputing*, vol. 99, 1999, pp. 5–33.

[7] S. Herrmann, S. Boutet, B. Duda, D. Fritz, G. Haller, P. Hart, R. Herbst, C. Kenney, H. Lemke, M. Messerschmidt *et al.*, "Cspad-140k: A versatile detector for lcls experiments," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 718, pp. 550–553, 2013.

[8] D. Becker and A. Streit, "A neural network based pre-selection of big data in photon science," in *Big Data and Cloud Computing (BdCloud), 2014 IEEE Fourth International Conference on*. IEEE, 2014, pp. 71–76.

[9] ——, "Localization of signal peaks in photon science imaging,," *IEEE UKSim*, vol. 25, p. 27, 2015.

[10] L. Redecke, K. Nass, D. P. DePonte, T. A. White, D. Rehders, A. Barty, F. Stellato, M. Liang, T. R. Barends, S. Boutet *et al.*, "Natively inhibited trypanosoma brucei cathepsin b structure determined by using an x-ray laser," *Science*, vol. 339, no. 6116, pp. 227–230, 2013.

[11] J. M. Perkel, "Decoding protein structure, one femtosecond at a time," 2014.

[12] E. Chiu, F. Coulibaly, and P. Metcalf, "Insect virus polyhedra, infectious protein crystals that contain virus particles," *Current opinion in structural biology*, vol. 22, no. 2, pp. 234–240, 2012.

[13] I. Sobel and G. Feldman, "A 3x3 isotropic gradient operator for image processing," *a talk at the Stanford Artificial Project in*, pp. 271–272, 1968.

[14] P. Hart, S. Boutet, G. CarmI, A. Dragone, B. Duda, D. Freytag, G. Haller, R. Herbst, S. Herrmann, C. Kenney, J. Morse, M. Nordby, J. Pines, N. van Bakel, M. Weaver, and G. J. Williams, "The Cornell-SLAC pixel array detector at LCLS," pp. 538–541, 2012.

[15] "Apple Instruments." https://developer.apple.com/library/mac/documentation/DeveloperTools/Conceptual/InstrumentsUserGuide/index.html#//apple_ref/doc/uid/TP40004652, [Online; accessed 2015-02-16].

[16] L. M. Russo, E. C. Pedrino, E. Kato, and V. O. Roda, *Image convolution processing: A GPU versus FPGA comparison*. IEEE, 2012.

[17] J. van Wezel, A. Streit, C. Jung, R. Stotzka, S. Halstenberg, F. Rigoll, A. Garcia, A. Heiss, K. Schwarz, M. Gasthuber *et al.*, "Data life cycle labs, a new concept to support data-intensive science," *arXiv preprint arXiv:1212.5596*, 2012.