PyFAI, a versatile library for azimuthal regrouping

2013 J. Phys.: Conf. Ser. 425 202012

(http://iopscience.iop.org/1742-6596/425/20/202012)

Journal of Physics: Conference Series **425** (2013) 202012 doi:10.1088/1742-6596/425/20/202012

PyFAI, a versatile library for azimuthal regrouping

Jérôme Kieffer, Dimitrios Karkoulis

European Synchrotron Radiation Facility; 6 rue Jules Horowitz; 38043 Grenoble; France

E-mail: jerome.kieffer@esrf.fr

Abstract. $2D$ area detectors like CCD or pixel detectors have become popular in the last 15 years for diffraction experiments (e.g. for WAXS, SAXS, single crystal and powder diffraction (XRPD)). These detectors have a large sensitive area of millions of pixels with high spatial resolution. The software package pyFAI has been designed to reduce SAXS, WAXS and XRPD images taken with those detectors into $1D$ curves (azimuthal integration) usable by other software for in-depth analysis such as Rietveld refinement, or $2D$ images (a radial transformation named *caking*). As a library, the aim of pyFAI is to be integrated into other tools like PyMca or EDNA with a clean pythonic interface. However pyFAI features also command line tools for batch processing, converting data into *q-space* (q being the momentum transfer) or 2θ-space (θ being the Bragg angle) and a calibration graphical interface for optimizing the geometry of the experiment using the Debye-Scherrer rings of a reference sample. PyFAI shares the geometry definition of SPD but can directly import geometries determined by the software FIT2D. PyFAI has been designed to work with any kind of detector and geometry (transmission or reflection) and relies on FabIO, a library able to read more than 20 image formats produced by detectors from 12 different manufacturers. During the transformation from cartesian space (x, y) to polar space $(2\theta, \chi)$, both local and total intensities are conserved in order to obtain accurate quantitative results. Technical details on how this integration is implemented and how it has been ported to native code and parallelized on graphic cards are discussed in this paper.

1. Introduction

With the advent of hyperspectral experiments like diffraction tomography in the world of synchrotron radiation, existing software tools for azimuthal integration like FIT2D[1] and SPD[2] reached their performance limits owing to the fast data rate needed by such experiments. Even when integrated into massively parallel frameworks like EDNA[3], such stand-alone programs, due to their monolithic nature, cannot keep the pace with the data flow of new detectors. Therefore we decided to implemente from scratch a novel azimuthal integration tool which is designed to take advantage of modern parallel hardware features.

2. Python Fast Azimuthal Integration

PyFAI is implemented in Python programming language, which is open source and already very popular for scientific data analysis (PyMca[4], PyNX[5], ...).

2.1. Geometry and calibration

PyFAI and SPD[2] share the same 6-parameter geometry definition: distance, point of normal incidence (2 coordinates) and 3 rotations around the main axis; these parameters are saved in text files usually with the *.poni* extension. The program *pyFAI-calib* helps calibrating the

experimental setup using a constrained least squares optimization on the Debye-Scherrer rings of a reference sample (LaB_6, silver behenate, ...). Alternatively, geometries calibrated using FIT2D[1] can be imported into pyFAI, including geometric distortions (i.e. optical-fiber tapers distortion) described as *spline-files*.

2.2. PyFAI executables

PyFAI was designed to be used by scientists needing a simple and effective tool for azimuthal integration. Two command line programs *pyFAI-waxs* and *pyFAI-saxs* are provided with pyFAI for performing the integration of one or more images. The WAXS version outputs result in $2\theta/I$, whereas the SAXS version outputs result in $q/I(/\sigma)$. Options for these programs are parameter file (*poni-file*) describing the geometry and the mask file. They can also do some pre-processing like dark-noise subtraction and flat-field correction (solid-angle correction is done by default).

2.3. Python library

PyFAI is first and foremost a library: a tool of the scientific toolbox built around IPython[6] and NumPy[7] to perform data analysis either interactively or via scripts. Figure 1 shows an interactive session where an integrator is created, and an image loaded and integrated before being plotted.

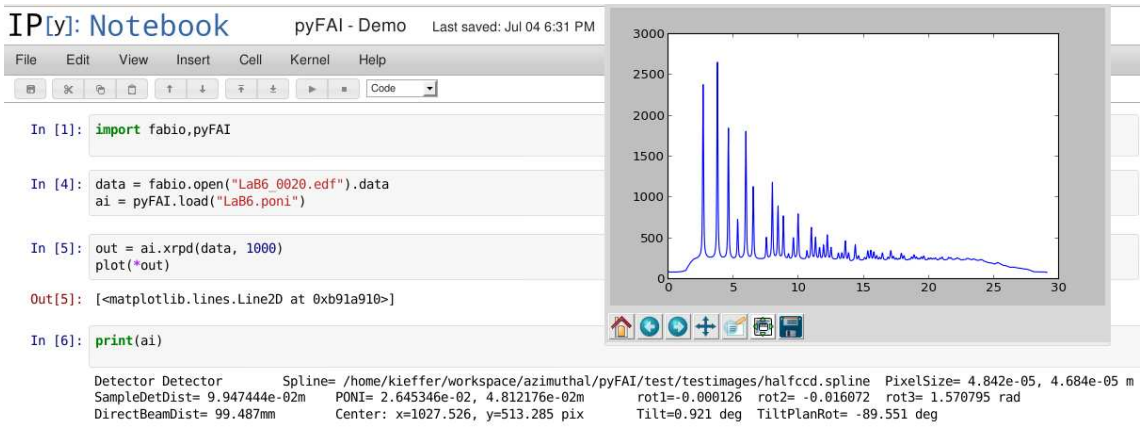

Figure 1. Example of interactive use of FabIO and pyFAI in the notebook edition of IPython.

3. Regrouping mechanism

In pyFAI, regrouping is performed using a histogram-like algorithm. Each pixel of the image is associated to its polar coordinates $(2\theta, \chi)$ or (q, χ), then a pair of histograms versus 2θ (or q) are built, one non weighted for measuring the number of pixels falling in each bin and another weighted by pixel intensities (after dark-current subtraction, and corrections for flat-field, solid-angle and polarization). The division of the weighted histogram by the number of pixels per bin gives the diffraction pattern. $2D$ regrouping (called *caking* in FIT2D) is obtained in the same way using two-dimensional histograms over radial (2θ or q) and azimuthal angles (χ).

3.1. Pixel splitting algorithm

Powder diffraction patterns obtained by histogramming have a major weakness where pixel statistics are low. A manifestation of this weakness becomes apparent in the $2D$-regrouping where most of the bins close to the beam-stop are not populated by any pixel. In Figure 2,

many pixels are missing in the low 2θ region, due to the arbitrary discretization of the space in pixels as intensities were assigned to each pixel center which does not reflect the physical reality of the scattering experiment. PyFAI solves this problem by pixel splitting (Figure 3): in addition to the pixel position, its spatial extension is calculated and each pixel is then split and distributed over the corresponding bins, the intensity being considered as homogeneous within a pixel and spread accordingly.

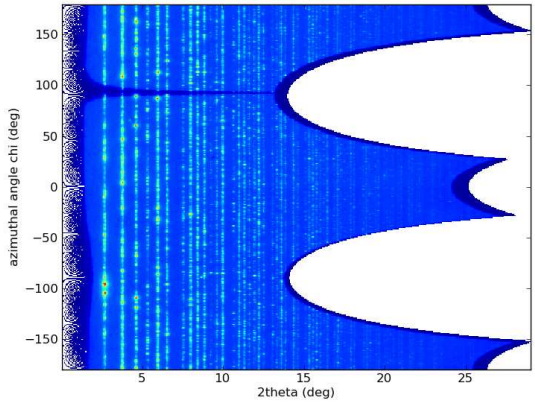

Figure 2. 2*D*-regrouped image without pixel splitting. Note the missing pixels near the beam stop and the high-frequency noise patterns.

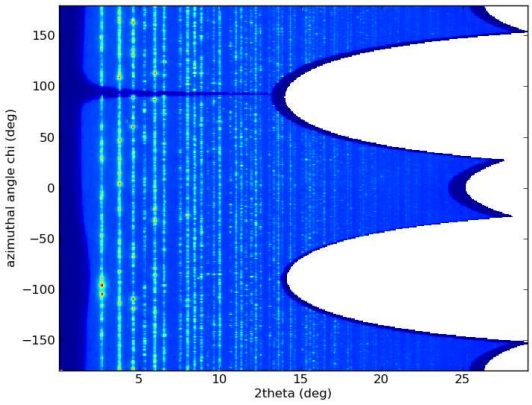

Figure 3. 2*D*-regrouped image with pixel splitting. The transformation of a smooth image remains smooth.

3.2. Performances and migration to native code

Originally, regrouping was implemented using the histogram provided by NumPy[7], then re-implemented in Cython[8] with pixel splitting to achieve a four-fold speed-up. The computation time scales like O(N) with the size of the input image. The number of output bins shows only little influence; overall the single threaded Cython implementation has been stated at 30 Mpix/s (on a 3.4 GHz Intel core i7-2600).

3.3. Graphic card implementation

Graphics Processing Units (GPUs) are composed of hundreds of arithmetic logic units; they are optimized for highly parallel algorithms with speed-up factors reaching up to 3 orders of magnitude over sequential codes running on Central Processing Units (CPU). While histograms do not fall into this category, they can nevertheless be ported to a GPU architecture efficiently. In order to benefit from GPU acceleration, the Open Computing Language[9] (OpenCL) was used. OpenCL can make use of multiple different devices such as CPUs and GPUs with very different features and capabilities. OpenCL allows the code to work on multiple CPU cores, which is useful for validation purposes. As azimuthal integration is a reduction of millions of pixels into hundreds of bins, double-precision arithmetic is preferred, which however is not available on all OpenCL devices. Table 1 summarizes the execution times for images recorded on various detectors on a dual-processor computer, either using 1) single threaded implementation in Cython, 2) OpenCL on 12 CPU-cores, 3) OpenCL on a nVidia Tesla C2075 (448 GPU-cores), 4) OpenCL on a nVidia GTX580 (512 GPU-cores) and 5) OpenCL on an AMD FirePro v7800 (1440 GPU-cores but only in single precision).

The OpenCL implementation of pyFAI is very fast on GPU providing an extra five-fold speed-up over the CPU implementation. The profiling of the code revealed new bottlenecks which will

Table 1. Execution time in milliseconds measured on a Dell T7500 with two Intel Xeon X5690 @3.47GHz and various GPUs.

Detector type	Image size	CPU X5690		OpenCL 1D regrouping			
	in Mpix	1D	2D	X5690	C2075	GTX580	FirePro
Pilatus-1M	1	34.4	63.1	13.9	7.2	6.3	13.8
Half-Frelon	2	76.6	132.4	23.4	14.4	12.2	18.8
Frelon	4	165.0	269.4	52.6	34.1	28.2	40.0
Pilatus-6M	6	232.0	350.7	74.4	49.8	40.7	48.1
Fairchild	16	613.9	849.7	158.9	99.0	96.4	95.6

be addressed in future optimizations. The OpenCL implementation of the 2D regrouping will also be finalized in a future release.

4. Conclusion

The library pyFAI was developed with two main goals:

- Performing azimuthal integration with a clean programming interface.
- No compromise on the quality of the results is accepted: a careful management of the geometry and precise pixel splitting ensures total and local intensity preservation.

PyFAI is the first implementation of an azimuthal integration algorithm on a GPU as far as we are aware of, and the stated twenty-fold speed up opens the door to a new kind of analysis, not even considered before: a ten-line Python script is sufficient to reduce the data of a whole diffraction-tomography experiment. Such analysis takes a few minutes using pyFAI on a 60 x 200 frames dataset whereas it used to take days with existing tools. We believe PyFAI is able to sustain the data streams from the next generation high-speed detectors.

Acknowledgments

The authors would like to express their most sincere appreciation to their colleagues and especially M. Sánchez del Río for suggesting the usage of weighted histograms; P. Bösecke for the provision of the experiment geometry setup; V. A. Solé for his expertise on developing native code under Windows and J. Wright for precise specifications and validations. Porting pyFAI to GPU would have not been possible without the financial support of LinkSCEEM-2 (RI-261600).

Appendix

PyFAI is open-source software released under the GPL licence. As of July 2012, pyFAI version 0.6, which includes OpenCL acceleration, is available on the EPN campus forge[10]. PyFAI depends on Python v2.6 or v2.7, NumPy[7] and OpenCL[9]. In order to be able to read images from various detectors, pyFAI relies on the FabIO[11] library available from SourceForge. In addition, the graphical user interface for calibration of diffraction setups uses matplotlib[12], SciPy[13], and FFTw3[14]. C, C++ compilers and Cython[8] are needed to build pyFAI from sources. PyFAI is packaged and available in common Linux distributions like Debian 7.0 and Ubuntu 12.04. Installer packages for Windows are also available on the EPN campus forge.

References

[1] Hammersley A P, Svensson S O, Hanfland M, Fitch A N and Hausermann D 1996 *High Press. Res.* **14** 235–248
[2] Bösecke P 2007 *J. Appl. Cryst.* **40** s423–s427

[3] Incardona M F, Bourenkov G P, Levik K, Pieritz R A, Popov A N and Svensson O 2009 *J. Synchrotron Rad.* **16** 872–879
[4] Solé V, Papillon E, Cotte M, Walter P and Susini J 2007 *Spectrochim. Acta Part B* **62** 63 – 68
[5] Favre-Nicolin V, Coraux J, Richard M I and Renevier H 2011 *J. Appl. Cryst.* **44** 635–640
[6] Pérez F and Granger B E 2007 *Comput. Sci. Eng.* **9** 21–29 URL `http://ipython.org`
[7] Oliphant T E 2007 *Comput. Sci. Eng.* **9** 10–20
[8] Behnel S, Bradshaw R, Citro C, Dalcin L, Seljebotn D and Smith K 2011 *Comput. Sci. Eng.* **13** 31 –39
[9] Khronos OpenCL Working Group 2010 *The OpenCL Specification, version 1.1* URL `http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf`
[10] Kieffer J *et al.* 2010– Azimuthal integration URL `https://forge.epn-campus.eu/projects/azimuthal`
[11] Sorensen H O, Knudsen E, Wright J, Kieffer J *et al.* 2007– FabIO: I/O library for images produced by 2D X-ray detectors URL `http://fable.sf.net/`
[12] Hunter J D 2007 *Comput. Sci. Eng.* **9** 90–95 ISSN 1521-9615
[13] Jones E, Oliphant T, Peterson P *et al.* 2001– SciPy: Open source scientific tools for Python URL `http://www.scipy.org/`
[14] Frigo M and Johnson S G 2005 *Proceedings of the IEEE* **93** 216–231