PyFAI: a Python library for high performance azimuthal integration on GPU

J. Kieffer and J.P. Wright

European Synchrotron Radiation Facility, 6 Rue Jules Horowitz, Grenoble 38000, France

E-mail: jerome.kieffer@esrf.fr, wright@esrf.fr

PyFAI is an open-source Python library for Fast Azimuthal Integration which provides 1D- and 2D-azimuthal regrouping with a clean programming interface and tools for calibration. The library is suitable for interactive use in Python. In optimising the speed of the algorithms there has been no compromise on the accuracy compared to reference software. Fast integrations are obtained by the combination of an algorithm ensuring that each pixel from the detector provides a direct contribution to the final diffraction pattern and an OpenCL implementation that can use graphics cards for acceleration. This contribution describes how the algorithms were modified to work better in parallel.

Key words: Azimuthal integration, OpenCL, GPU

1 Introduction

Online data analysis is needed when using fast detectors in order to follow experiments as they happen. This is especially important for user experiments at synchrotron radiation facilities, where data rates are very high and access to beamtime is usually limited to short visits per measurement. If the data can be processed as soon as they are collected and are still "live" in memory then we can bypass a slow step of re-reading images from disk or over a network.

In powder diffraction and small angle scattering experiments the variables of interest are usually the scattering angles (or momentum transfer) and not the pixel co-ordinates on the detector. Frequently the data should be radially symmetric due to the random orientations inside the sample and integration to a give a 1D profile can be carried out to provide data which is ready for Rietveld refinement. When the radial symmetry is broken then an azimuthal regrouping (or radial transform) is performed as it can be used to measure features in the scattering pattern as a function of direction, the azimuthal angle (χ). This transform is useful in the analysis of crystallographic texture or strain.

Over the years a number of software packages have been developed make a radial averaging of

 0885-7156/2013/28(S2)/S2/1/$18.00

image data (Hammersley *et al.*, 1996; Cervellino *et al.*, 2006; Hinrichsen *et al.*, 2006; Rodriguez-Navarro, 2006; De Nolf and Janssens, 2010). PyFAI is designed to accomplish this same task when used as a library which allows easy integration into other software, including synchrotron beamline control software. This can mean running in a dedicated server (Tango device server, LImA (Homs *et al.*, 2011) where things must be fast and stable over weeks of operation and also suitable for use in interactive graphical user interfaces. PyFAI builds on these recent developments and has been extensively optimised for speed on modern parallel computers while retaining accuracy. The software is already used on two small angle scattering beamlines at ESRF (BM26 and BM29) and under evaluation on several others: ID11, ID13, ID02, ID23 and ID29 at ESRF but also on Cristal at Soleil, Lions at CEA-Saclay and i711 at MAX IV.

2 PyFAI: Python Fast Azimuthal Integration

PyFAI uses the geometry defined in SPD (Bösecke, 2007); it is a 6-parameter geometry considering a flat detector. The orthogonal projection of the sample on the detector plane defines the **P**oint **O**f **N**ormal **I**ncidence (PONI); with 2 coordinates in the detector plane plus the distance to the sample. Finally 3 rotations around the 3 main axes are considered; but one of them (rotation around the incident beam itself) is usually not used due to the symmetry of Debye-Scherrer cones. This geometry does not consider the beam center as origin, making it suitable for detectors mounted on 2θ-arms and other configurations with translation stages reaching large 2θ angles.

PyFAI is a Python library, providing a collection of tools and also a few programs which can be used directly (outside of the Python language). There is a program for determining experimental parameters, **pyFAI-calib**, which makes a calibration based on the known 2θ angles of a standard material (LaB_6, Si, …). Refinement of a calibration with a good initial set of parameters can be done using the **pyFAI-recalib** script. The difference between the two programs is the extraction of the key-points which is manual in the first case and fully automatic in the second. A simple graphical interface based on matplotlib (Hunter, 2007) shows the progress of the process. Refinement of the geometry parameters is performed with the constrained least squares from scipy (scipy.opimize.fmin_slsqp).

Azimuthal regrouping can be performed by the script, **pyFAI-waxs**. There are also a few scripts for the definition of the masked-out regions, etc. but most of the package is actually intended to

https://doi.org/10.1017/S0885715613000924 Published online by Cambridge University Press

be usable for scientists in an interactive Python session like IPython (Pérez and Granger, 2007).

Fast Azimuthal Integration

PyFAI tries to offer a unified and Pythonic interface for azimuthal integration and at the same time very high performance. Memory is traded for speed, so PyFAI caches large arrays of pixel values for 2θ, χ, solid angles, polarization, etc. Where appropriate, calculations are only performed when needed and the results are stored in a cache when they can be used again. This kind of "lazy evaluation" approach means that the time for processing the first image processed is significantly longer than for subsequent images. The expectation is that a series of similar images will be processed. For various trigonometrical calculations each pixel in the in the input image can be treated independently and so the calculations can be done in parallel without changing the code. Cython-OpenMP was used to speed up processing of this kind of trivially parallelizable problem on computers which have multiple CPU cores.

When processing, the raw image must be corrected for dark current, flat-field, solid angle and polarization effects; then azimuthal integration is performed by histogramming in 1D (or 2D) the 2θ- positions (or the 2θ, χ -positions) weighted by the intensity of the image. Pixels are split and contribute to adjacent bins depending on their spatial extent. This method works well on a single processor but runs into problems requiring so called "atomic operations" when run in parallel. Processing pixels in the input data order causes write access conflicts which become less efficient with the increase of number of computing units. This is the main limit of the method exposed in (Kieffer and Karkoulis, 2013); especially on GPU where hundreds of threads are executed simultaneously.

To overcome this limitation; instead of looking at where input pixels *GO TO* in the output image, we instead look at where the output pixels *COME FROM* in the input image. The correspondence between pixels and output bins can be stored in a look-up table (LUT) together with the pixel weight which make the integration look like a simple (if large and sparse) matrix vector product. This look-up table size depends on whether pixels are split over multiple bins and to exploit the sparse structure, both index and weight of the pixel have to be stored. We measured that 500 Mb are needed to store the LUT to integrate a 16 megapixel image, which fits onto a reasonable quality graphics card nowadays. By making this change we switched from a “linear read / random write” forward algorithm to a “random read / linear write” backward

https://doi.org/10.1017/S0885715613000924 Published online by Cambridge University Press

algorithm which is more suitable for parallelization. This algorithm was implemented in Cython-OpenMP (Behnel *et al.*, 2011) and OpenCL (Stone et al., 2010). When using OpenCL for the GPU we used a compensated, or Kahan summation (Kahan, 1965) to reduce the error accumulation in the histogram summation (at the cost of more operations to be done). This allows accurate results to be obtained on cheap hardware that performs calculations in single precision floating-point arithmetic (32 bits) which are available on consumer grade graphic cards. Double precision operations are currently limited to high price and performance computing dedicated GPUs. The additional cost of Kahan summation, 4x more arithmetic operations, is hidden by smaller data types, the higher number of single precision units and that the GPU is usually limited by the memory bandwidth anyway.

3 Examples of scientific usage:

Separation of powder from single crystal diffraction

An IPython notebook (Pérez and Granger, 2007) offers a flexible user interface for testing ideas and algorithms and using the library (Figure 1). In the different boxes the following steps are being carried out:

1) Load libraries FabIO (Knudsen *et al.*, 2013) and pyFAI (Kieffer and Karkoulis, 2013) into current Python interpreter

2) Load images and correct for dark-current and flat-field

3-4) Create an "Integrator" object from Fit2D (Hammersley *et al.*, 1996) parameters and integrate an image

5) Generate a computed image from the powder pattern, subtract it from original image to highlight Bragg peaks

6) Following a 2D radial transform, apply a median filter on the vertical dimension to separate amorphous scattering from crystalline scattering

https://doi.org/10.1017/S0885715613000924 Published online by Cambridge University Press

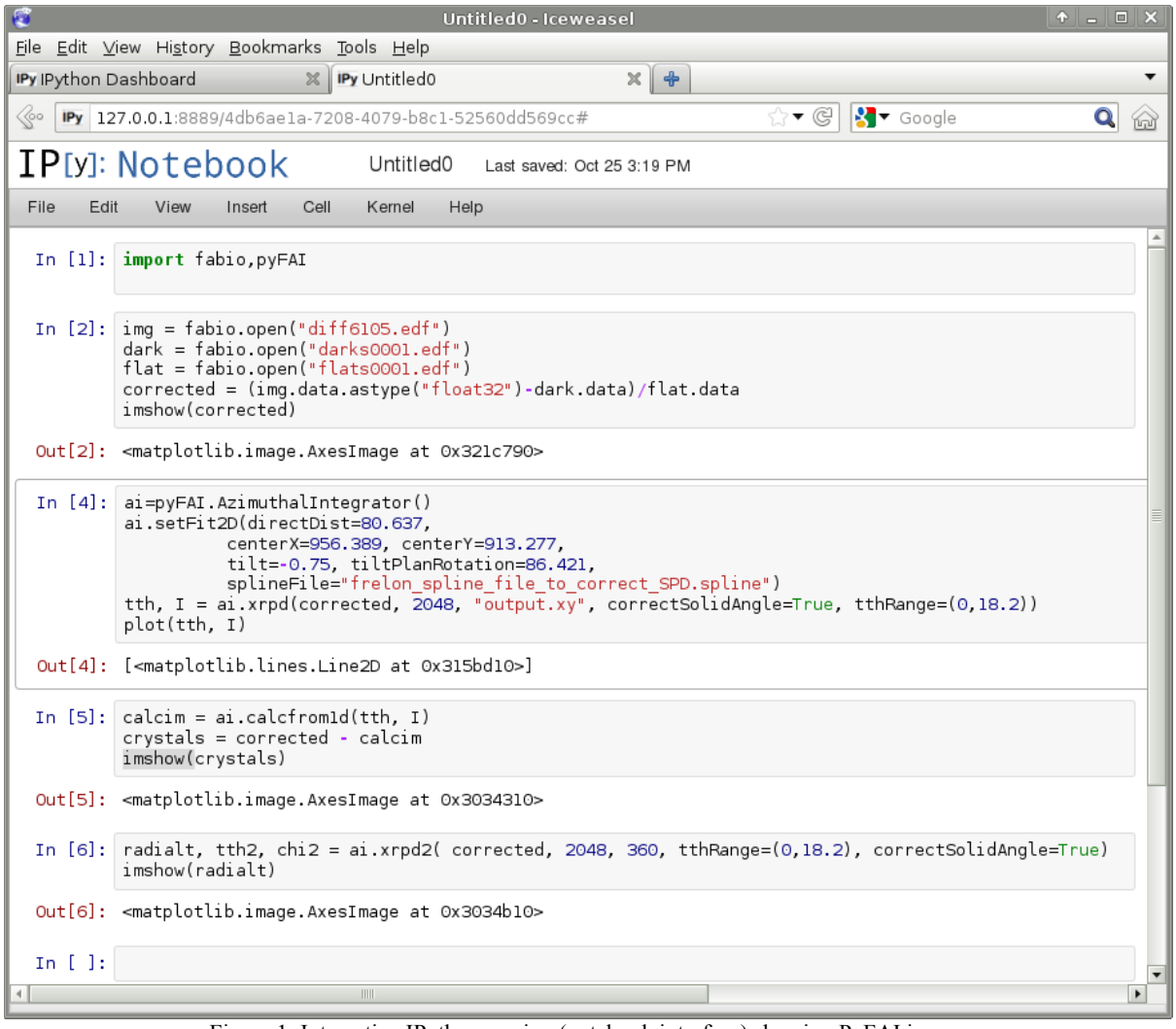

Figure 1: Interactive IPython session (notebook interface) showing PyFAI in use.

In this example we have exploited the calcfrom1D method of the PyFAI integrator object to produce a computed image from the 1D integrated data. This is the image we would have recorded in the absence of any errors, given the 1D integrated data. We believe this computed image is extremely useful for diagnostics as is allows a direct comparison of the 2D recorded data with the integrated radially symmetric part.

In Figure 2 we compare the integration obtained in 1D and 2D for a specific example where the sample contains a few large crystals and some diffraction spots. Using the PyFAI library it is straightforward to separate the different contributions to the diffraction pattern.

https://doi.org/10.1017/S0885715613000924 Published online by Cambridge University Press

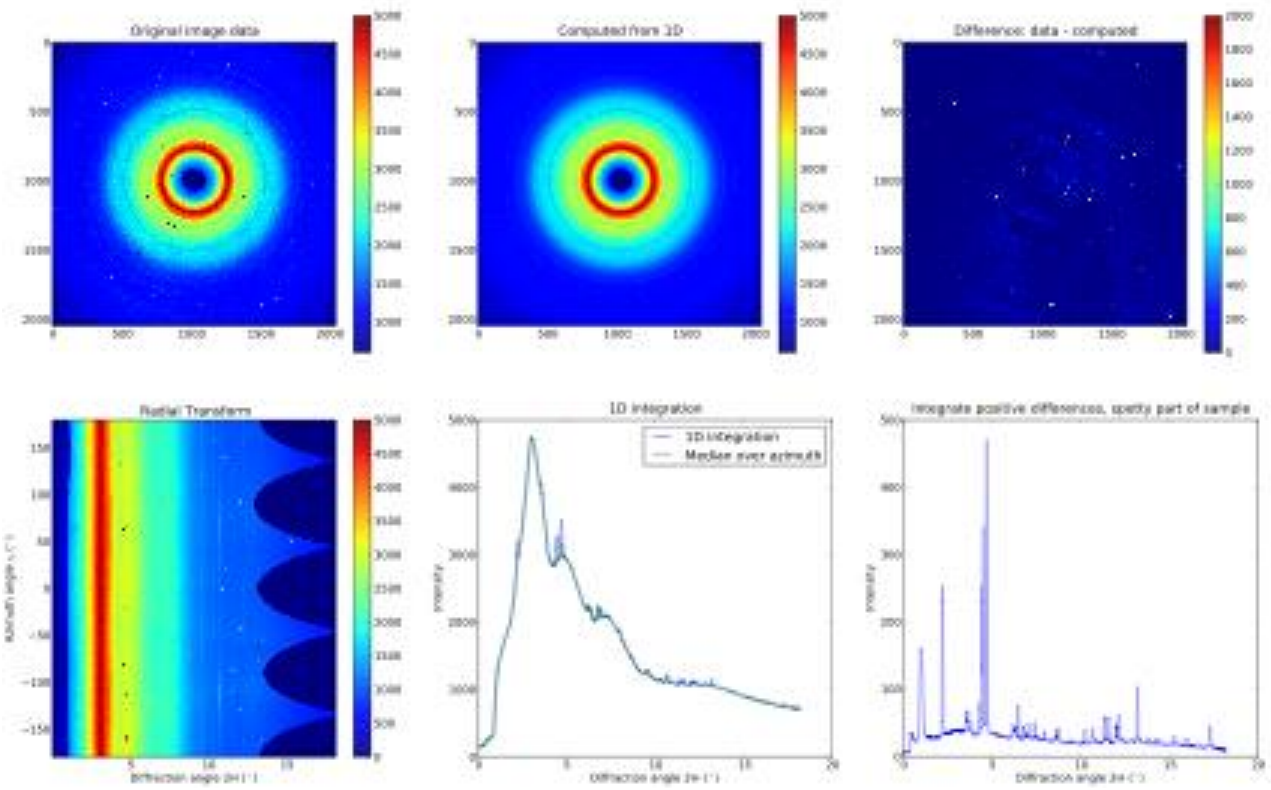

Figure 2: Results obtained using PyFAI on a spotty diffraction image. Top left: Original data. Bottom left: 2D radial transform. Top middle: Computed 2D image from conventional 1D integration. Bottom middle: Conventional 1D integration compared to a median on the 2D radial transform. Top right: Residual image after the 1D computed part is removed. Bottom right: The spotty part of the diffraction pattern can be extracted from the background.

Large diffraction angles

PyFAI is especially suited to geometry with large 2θ values or detectors mounted on moving stages. In the later case; a few diffraction images of a reference sample on various detector positions have to be calibrated, then their parameters can be extrapolated from the linear regression of those few images. In Figure 3; seven diffraction images of LaB_6 have been taken at various angles of the 2θ-arm, from 0 to 120°, on the i711 beamline at synchrotron Max IV. The three first images were calibrated using **pyFAI-calib**; then a linear regression allowed to guess parameters for the other images; those guess-parameters were used to re-calibrate all seven images using the **pyFAI-recalib** tool. During this procedure up to 25 diffraction rings per image were extracted and fitted fully automatically in a few seconds. PyFAI has no problem calibrating experiment setups with tilt > 90°, where diffraction rings are no more circles or ellipses but branches of hyperbola (pyFAI uses the general conic equation for fitting).

https://doi.org/10.1017/S0885715613000924 Published online by Cambridge University Press

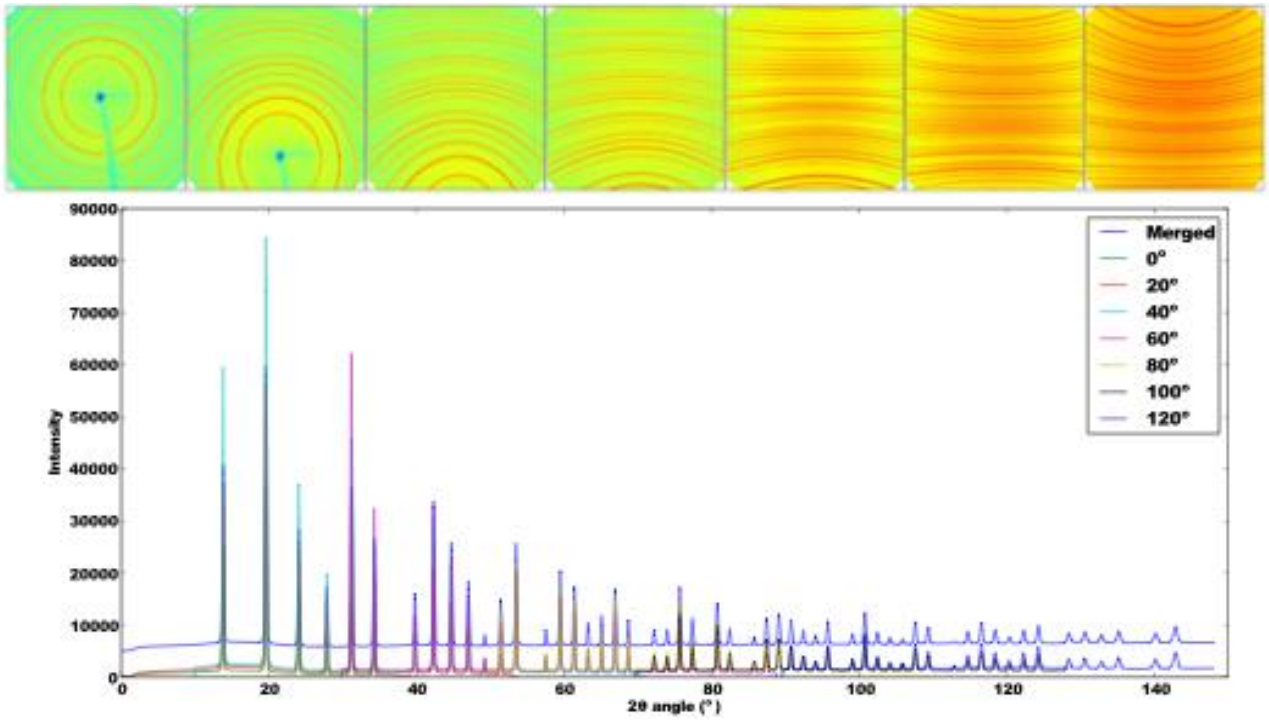

Figure 3: On the top, seven diffraction images of LaB_6 powder taken with the detector on a 2θ-arm moving from 0 to 120°. The graph shows the overlay of the seven individual powder patterns (lower curve) and the full reconstructed powder pattern obtained by merging, on one side weighted and on the other all un-weighted histograms, from the various images (upper curve).

As pyFAI is open source and Python is a dynamic language; one can use some of the internals of the library, here the two raw histograms that are calculated during integration. All seven images can be integrated on a large 2θ range from 0 to 150°; forcing the position of the 2θ-bins to be the same on all powder patterns. Then intensity-weighted histograms in 2θ and un-weighted ones can be summed together, weighted on one side, un-weighted on the other. Finally the full powder pattern is obtained by dividing the weighted histogram by the un-weighted one (Figure 3, blue curve), merging the data into a single pattern. If such experiments become more frequent, a specific class could be added to pyFAI for performing such type of analysis in a more routine way.

Error handling

Error evaluation and handling is especially important for small angle scattering. According to the "Poisson's law", the error of a single pixel can be estimated from the square root of the raw value (prior to any corrections); hence its variance is the raw value itself. PyFAI can propagate this error during the integration: the error of each bin being the square root of the variance-weighted histogram, divided by the number of pixels falling into this bin. This implies having access to

https://doi.org/10.1017/S0885715613000924 Published online by Cambridge University Press

the raw data, so that dark-current and flat-field corrections are applied by pyFAI itself. A specific keywork (error_model=”Poisson”) was added to provide this feature when calling the saxs integration method. While the Poisson model works well for photon counting detectors, the alternative for other detector types is currently to let the user provide the variance array.

4 Performance

The performance of the code is monitored on a variety of computing hardware and for different image sizes. A number of different kernels for azimuthal integration are benchmarked and compared so that a user may select the right optimised version for a specific computer setup. Figure 4 shows the number of frames processed per second versus image size (larger numbers are better). The maximum data throughput is of the order 800 megabytes per second (200 Mpixel/second of float data), and therefore depends on the computer having a sustained supply of data to be able to continue working at this speed. This out performs the speed of most current hard disks. For a typical image of 2048x2048 pixels, after loading and setting up, pyFAI takes about 20 milliseconds per new frame to compute the 1D integrated profile but sustained rate on integrating large datasets (thousands of images) is only of 100 milliseconds per image due to reading bottleneck.

https://doi.org/10.1017/S0885715613000924 Published online by Cambridge University Press

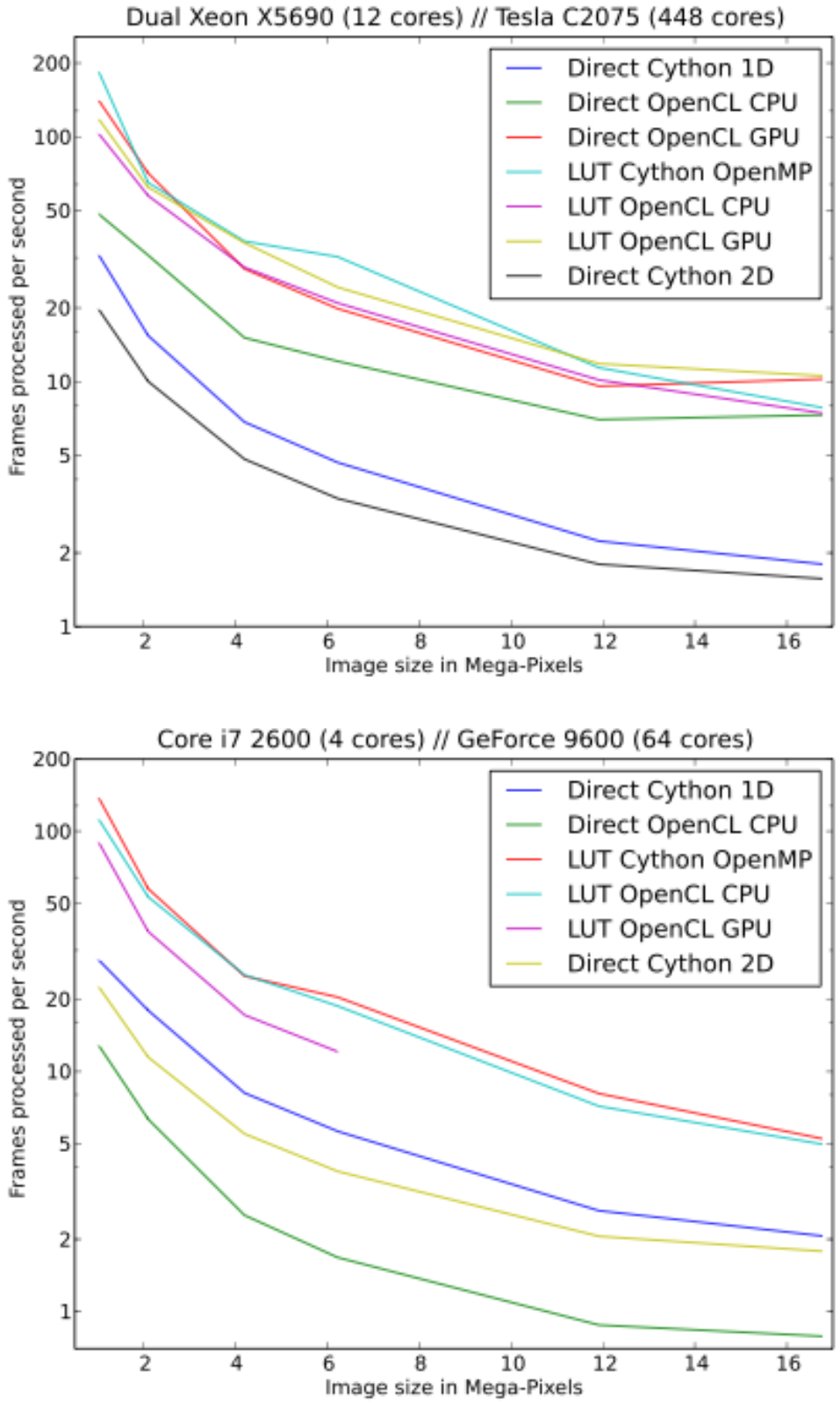

Figure 4: Performances measured on a high-end workstation (above) and on a consumer desktop computer (below). The LUT_OpenCL_GPU line in magneta terminates when the consumer card ran out of memory.

https://doi.org/10.1017/S0885715613000924 Published online by Cambridge University Press

5 Project status

The PyFAI code is fully open source under the GPL license and available from http://github.com/kif/pyFAI. PyFAI is packaged and available in common Linux distributions like Debian 7.0 and Ubuntu 12.04. Installer packages for Windows and MacOSX are available on https://forge.epn-campus.eu/projects/azimuthal/files. The software depends on: Python 2.6+, NumPy, SciPy, matplotlib and FabIO. Thanks to image input code from the FabIO (Knudsen *et al.*, 2013) library, the code is already compatible with at least 20 detectors from 12 different manufacturers. For best performances: FFTw3 with pyFFTw, PyOpenCL (Kloeckner, 2012) (and a GPU) are recommended.

The version 0.8 described in this article has been released in November 2012 and includes:

- 1D-regrouping parallel algorithm using look-up tables (LUT) on both CPU and GPU
- Polarization effect correction in addition to dark-current, flat-field and solid-angle corrections
- Fast checksums to avoid unnecessary data transfers to GPU
- A unified interface to OpenCL code via PyOpenCL (Kloeckner, 2012)
- Comprehensive test suites for most algorithms

In the future we plan to make PyFAI available for integration into LImA (Library for Image Acquisition (Homs et al., 2011) so that reduced data will be available inside the detector device servers, going around the bottleneck of the disk access. Also the code will be provided as a plugin into PyMca (Solé *et al.*, 2007) to provide azimuthal integration of stacks of images within a nice graphical user interface and in EDNA (Incardona *et al.*, 2009) for online data analysis. Version 1.0 should include in addition:

- 2D-regrouping algorithm using look-up tables (LUT)
- Nicer and more intuitive graphical interface for calibration
- Plugins for LimA (Homs *et al.*, 2011), PyMca (Solé *et al.*, 2007) and EDNA (Incardona *et al.*, 2009)
- Extensive API documentation and program manuals

In the spirit of an open source project, any contributions of improvements, test cases or adaptations for specialised instrumentation are most welcome.

6 Conclusions

PyFAI is a novel library for azimuthal integration which already provides geometric equivalence with SPD (Bösecke, 2007) and Fit2D (Hammersley *et al.*, 1996). A clean and modern programming interface has been developed which is suitable for interactive use as well as integration into beamline control systems.

By re-ordering the numerical operations to avoid write access conflicts, we have improved the performance of parallel implementations of a radial transform running on multi-core systems and graphic cards. Numerical precision has been improved via the use of compensated summations.

https://doi.org/10.1017/S0885715613000924 Published online by Cambridge University Press

We show how a radially symmetric 2D image can be back-computed from a 1D powder pattern and we suggest this computed image can be a powerful tool for processing 2D image data. We also show how multiple images taken from a moving detector can be merged into a single powder pattern.

Acknowledgements:
The authors would like to express their most sincere appreciation to their colleagues and especially to Dimitris Karkoulis who ported the original algorithm to GPU, 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 compiling native code under Windows and MacOSX and Carsten Gundlach from beamlines i711 and i811 at MAX IV for providing an interesting test case with diffraction images of a detector mounted on a 2θ–arm.

Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D. S. and Smith, K. (**2011**). "Cython: The Best of Both Worlds," Comput. Sci. Eng. **13**, 31-39. [DOI: 10.1109/MCSE.2010.118].

Bösecke, P. (**2007**). "Reduction of two-dimensional small- and wide-angle X-ray scattering data," J. Appl. Crystallogr. **40**, 423-427. [DOI: 10.1107/S0021889807001100].

Cervellino, A., Giannini, C., Guagliardi, A. and Ladisa, M. "Folding a two-dimensional powder diffraction image into a one-dimensional scan: a new procedure," J. Appl. Crystallogr. (**2006**). **39**, 745-748. [DOI: 10.1107/S0021889806026690].

De Nolf, W. and Janssens, K. (**2010**). "Micro X-ray diffraction and fluorescence tomography for the study of multilayered automotive paints," Surf. Interface Anal. **42**, 411-418. [DOI: 10.1002/sia.3125].

Hammersley, A. P., Svensson, S. O., Hanfland, M., Fitch, A. N. and Hausermann, D. (**1996**). "Two-dimensional detector software: From real detector to idealised image or two-theta scan," High Press. Res. **14**, 235-248. [DOI: 10.1080/08957959608201408].

Hinrichsen, B., Dinnebier, R. E., Rajiv, P., Hanfland, M., Grzechnik, A. and Jansen, M. (**2006**). "Advances in data reduction of high-pressure x-ray powder diffraction data from two-dimensional detectors: a case study of Schafarzikite (FeSb2O4)," J. Phys.: Condens. Matter **18**, S1021-S1037 [DOI: 10.1088/0953-8984/18/25/S09].

Homs , A., Claustre, L., Kirov, A., Papillon, E. and Petitdemange, S. (**2011**). "LImA: a

Generic library for high throughput image acquisition," ECALEPCS proceedings 676-679.

Hunter, J. D. (**2007**). "Matplotlib: A 2D graphics environment," Comput. Sci. Eng. **9**, 90-95. [DOI: 10.1109/MCSE.2007.55].

Incardona, M-F., Bourenkov, G. P., Levik, K., Pieritz, R. A., Popov, A. N. and Svensson, O. (**2009**). "EDNA: a framework for plugin-based applications applied to X-ray experiment online data analysis," J. Synchrotron Radiat. **16**, 872-879. [DOI: 10.1107/S0909049509036681].

Kahan, W. (**1965**). "Pracniques: Further remarks on reducing truncation errors," Commun. ACM **8**, 40. [DOI:10.1145/363707.363723].

Kieffer, J. and Karkoulis, D. (**2013**). "PyFAI, a versatile library for azimuthal regrouping," J. Phys.: Conf. Ser. **425**, 202012. [DOI: 10.1088/1742-6596/425/20/202012].

Kloeckner, A., Pinto, N., Lee, Y. Catanzaro, B., Ivanov, P. and Fasih, A. (**2012**). "PyCUDA and PyOpenCL: A scripting-based approach to GPU run-time code generation," Parallel Comput. **38**, 157-174. [DOI: 10.1016/j.parco.2011.09.001].

Knudsen, E. B., Sorensen, H. O., Wright, J. P., Goret, G. and Kieffer, J. (**2013**). "FabIO: easy access to 2D X-ray detector images in Python," J. Appl. Crystallogr. **46**, 537-539. [DOI: 10.1107/S0021889813000150].

Pérez, F. and Granger, B. E. (**2007**). "IPython: a system for interactive scientific computing," Comput. Sci. Eng. **9**, 21-29. [DOI: 10.1109/MCSE.2007.53].

Rodriguez-Navarro, A. B. (**2006**). "XRD2DScan: new software for polycrystalline materials characterization using two-dimensional X-ray diffraction," J. Appl. Crystallogr. **39**, 905-909. [DOI: 10.1107/S0021889806042485].

Solé, V. A., Papillon, E., Cotte, M., Walter, Ph. and Susini, J. (**2007**). "A multiplatform code for the analysis of energy-dispersive X-ray fluorescence spectra," Spectrochim. Acta, Part B **62**, 63- 68. [DOI: 10.1016/j.sab.2006.12.002].

Stone, J. E., Gohara, D. and Shi, G., C. (**2010**). "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems," Comput. Sci. Eng. **12**, 66-73. [DOI: 10.1109/MCSE.2010.69].