PyNX : high-performance computing toolkit for coherent X-ray imaging based on operators

Vincent Favre-Nicolin, Gaétan Girard, Steven Leake, Jerome Carnis, Yuriy Chushkin, Jerome Kieffer, Pierre Paleo, Marie-Ingrid Richard

PyNX: high performance computing toolkit for coherent X-ray imaging based on operators

Favre-Nicolin, Vincent[a,b], Girard, Gaétan[a], Leake, Steven[a], Carnis, Jérôme[c], Chushkin, Yuriy[a], Kieffer, Jérôme[a], Paléo, Pierre[a], and Richard, Marie-Ingrid[b,d]

[a]ESRF, The European Synchrotron, 71 Av. des Martyrs, 38000 Grenoble, France
[b]Univ. Grenoble Alpes, Grenoble, France
[c]Deutsches Elektronen-Synchrotron (DESY), D-22607, Hamburg, Germany
[d]CEA, IRIS-MEM, Nanostructures and Synchrotron Radiation Laboratory, F-38000 Grenoble, France

August 27, 2020

Abstract

The open-source PyNX toolkit [1, 2] has been extended to provide tools for coherent X-ray imaging data analysis and simulation. All calculations can be executed on graphical processing units (GPU) to achieve high performance computing speeds. This can be used for Coherent Diffraction Imaging (CDI), Ptychography and wavefront propagation, in the far or near field regime. Moreover, all imaging operations (propagation, projections, algorithm cycles..) can be used in Python as simple mathematical operators, an approach which can be used to easily combine basic algorithms in a tailored chain. Calculations can also be distributed to multiple GPUs, e.g. for large Ptychography datasets. Command-line scripts are also available for on-line CDI and Ptychography analysis, either from raw beamline datasets or using the Coherent X-ray Imaging data format [3].

1 Introduction

Coherent X-ray Imaging techniques have been intensely developed during the last 20 years thanks to the wide availability of synchrotron sources with high brilliance. This covers a wide range of techniques, beginning with phase contrast imaging [4, 5, 6], coherent diffraction imaging [7, 8, 9], allowing to reconstruct single objects from their diffraction pattern alone, including strain imaging of crystalline nano-objects in the Bragg geometry [10, 11, 12, 13]. More recently X-ray Ptychography [14, 15, 16], which can be used both in the far field and near field regime [17, 18], was developed for imaging extended objects (larger than the incident beam), both in the small angle and in the Bragg geometry [19, 20]; this technique can also be used in the Fourier regime by scanning the transmitted beam [21].

These techniques all provide high-resolution two or three-dimensional imaging, down to 5 to 15 nanometer resolution depending on the experimental setup. The main experimental requirement is a coherent X-ray beam, which is readily available at synchrotron facilities. These experiments will see the main benefit of the current upgrades of synchrotron rings, which promise two orders of magnitude increase in the available coherent X-ray flux thanks to higher brilliance [22, 23, 24], and will enable faster dynamics and imaging experiments, as well as reach higher resolution [25]. The availability of a higher coherent fraction at high energy (>20keV) will enable data collection for thicker samples and allow to mitigate radiation damage with lower absorption.

Data analysis often remains a bottleneck for these experiments, either from the complexity of the algorithms, or simply the computing requirements. A variety of software is readily available for

Ptychography [26, 27, 28, 29, 30, 31, 32], and fewer for CDI [33, 34]. Several limitations remain: (i) software packages are not always publicly distributed, (ii) high-performance computing - generally based on graphical processing units (GPU) is not always available or complicated to setup, and (iii) the software can be difficult to maintain or improve due to the complexity of algorithms or their GPU implementation.

In this article we will present the open-source coherent X-ray imaging modules of the PyNX toolkit. In the previous versions [1, 2], GPU-accelerated computing was only available for scattering calculations (which are unchanged), whereas this new version is a complete rewrite of the Ptychography module, and adds tools for CDI and Wavefront calculations, all GPU-accelerated. We will first present an outline of the toolkit organisation, followed by details of the operator-based approach which is used to simplify the development of custom algorithms, and finally the available command-line scripts.

2 PyNX toolkit organisation

PyNX is written primarily in Python, using the NumPy and SciPy libraries [35] for basic data processing it is organised in several modules for the different tasks:

- `cdi`: Coherent Diffraction Imaging. See section 3.
- `operator`: classes and functions for the *operator*-based approach, which is described in section 3.
- `processing_unit`: functions to automatically select and initialise the processing units, either using OpenCL or CUDA.
- `ptycho`: Ptychography. See section 4.
- `scattering`: legacy GPU-accelerated kinematical scattering calculations, as described in [1].
- `test`: automated test routines and scripts.
- `utils`: array handling and plotting routines.
- `wavefront`: coherent wavefront propagation, mostly for simulation purposes.

All calculations for the new coherent imaging modules (`cdi`, `ptycho` and `wavefront`) are executed on the available GPU, either using the pyCUDA or pyOpenCL libraries [37]. The language is automatically selected as well as the GPU: this is done by favouring CUDA over OpenCL if both are available (see supplementary figures 6 and 7 in the appendix for a comparison), and then selecting the fastest GPU (based on a 2D FFT test) if more than one is available. The user can also opt to select a language and/or a GPU from its name or rank among those available.

One important aspect of the code design is that while most (84% lines) of it is written in Python, all the algorithms are executed *asynchronously* on the GPU, i.e. the commands are sent by the Python process to the GPU for a large number of operations[1]. This ensures that the execution on the GPU is rarely interrupted, i.e. only when it is necessary to get back data from the GPU to the computer main memory.

Illustrated in figure 2, the graphical view of the profiling of a CDI optimisation is demonstrated. It can be seen that after launching the chain of algorithms from Python, the code is executed on the GPU with negligible latency between the different operations (Fourier transforms, support projections, copy of arrays, ...). This remains true until some data has to be retrieved from the GPU memory, but only happens when fetching data from the GPU (e.g. for plotting or displaying figures of merit).

[1]Note that GPU kernels are naturally executed asynchronously, but in order to maintain a high performance, it is important to design all algorithms to fetch values from the GPU as scarcely as possible (every 10s or 100s of cycles), so that the GPU process is not slowed down by a synchronisation process, even if it is only to copy a few floating-point values

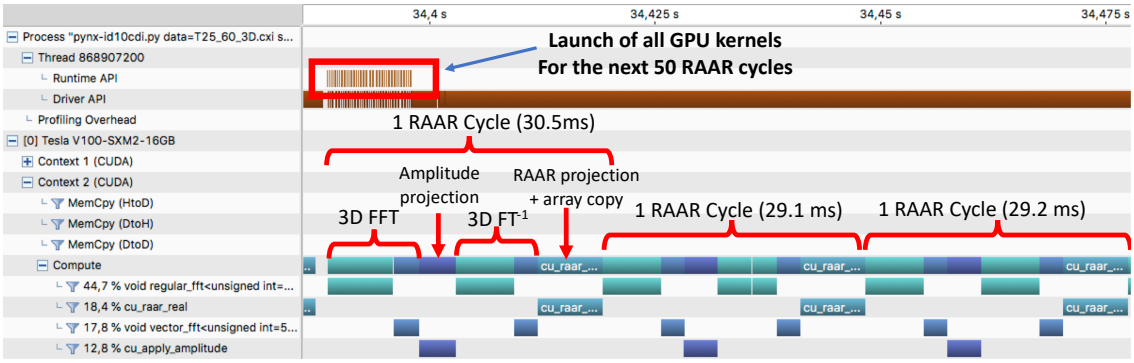

Figure 1: Example profiling during a CDI analysis: the algorithm consists of 50 cycles of Relaxed Averaged Alternating Reflections (RAAR) [36] applied to a 512^3 voxels object. The process includes a Fourier Transform of the complex object $O(n)$, an amplitude projection to apply the observed amplitudes, an inverse Fourier transform, a linear combination with the previous object $O(n)$ depending on whether the object voxel is within the support. As can be seen from the nVidia visual profiler graphical display, the command for *all* the 50 cycles of RAAR are executed ('Runtime API') within 30 ms, and are then executed *asynchronously*: once the commands have been queued, the GPU is constantly working on the different parts of the algorithm (intervals in white or light grey indicate no activity), while the Python code can independently prepare other commands to be queued. This asynchronous behaviour is close to an optimal GPU performance. Such a calculation can be executed in Python simply by writing: `cdi = RAAR()**50 * cdi`.

Finally, the `cdi` and `ptycho` modules also include a `runner` sub-module, which handles automated data processing using command-line scripts, which will be described in section 5.

3 Coherent Diffraction Imaging and operators

The CDI technique consists of reconstructing an object from a far-field diffraction pattern alone [7], a technique which has been expanded to three-dimensional reconstruction by collecting multiple (>100) projections around a rotation axis, either in the small angle [38, 39] or in the Bragg geometries [10, 12] - the latter approach also yields information about strain in the reconstructed object.

In order to reconstruct the object from non-redundant diffraction data it is necessary to recover the lost phases of the measured amplitudes. A variety of algorithms are available, all of which rely on alternating between a real-space estimate of the object, where its extent (the so-called 'support' of the object) is evaluated, and diffraction (Fourier) space, where an amplitude constraint can be applied from the measured intensities. A unified view of these algorithms has been presented in [40], with a demonstration that all operations applied to the object array can be described as mathematical operators. For example, the simplest algorithm -error reduction- can be written in the following way:

$$\rho^{(n+1)} = P_S \mathcal{F}^{-1} P_m \mathcal{F} \rho^{(n)} \tag{1}$$

where $\rho^{(n)}$ is the object's complex density array at iteration (n), $\mathcal{F}$ is the Fourier transform, P_m is a magnitude projection operator which replaces the modulus of the calculated scattering by the observed ones, and P_S is the support projection operator, which multiplies the object array by 0 outside its support.

Since a complete algorithmic chain used to retrieve an object relies on a large number (at least a few hundred) of such mathematical operations, it is convenient to use object-oriented programming to enable writing the sequence of operations exactly as mathematical operations. This is achieved in the following way:

```
import numpy as np
# This imports all necessary operators & classes
# The GPU and language (OpenCL/CUDA) are auto-selected
from pynx.cdi import *
# Load data, support and mask of bad pixels
iobs = np.load("iobs.npz")['iobs']
mask = np.load("mask.npz")['mask']
support = np.load("support.npz")['support']
# Create main CDI object
cdi = CDI(iobs, support=support, mask=mask)
# Initial scaling of object with respect to Iobs
cdi = ScaleObj(method='F') * cdi
# Do 40 cycles of HIO with a positivity constraint
cdi = HIO(positivity=True)** 40 * cdi
# Support update operator
sup = SupportUpdate(threshold_relative=0.17)
# Do 20*(40 cycles of HIO, 5 of ER, support update)
cdi = ShowCDI()* (sup*ER()** 5*HIO()** 40)**20 * cdi
```

Figure 2: Example CDI reconstruction code using operators. All the reconstruction operations (`HIO`, `ER`, `SupportUpdate`) are transparently (no explicit initialisation is required) and asynchronously executed on the GPU. Only when the `ShowCDI` operator is used, the resulting object and support are fetched from the GPU for display, which automatically waits for all operations queued on the GPU to be finished.

1. a CDI class is defined, including as data the object array (either in real or Fourier space) and the support array (0 outside the support, 1 inside), with a few input/output functions.
2. a family of CDI operators is created, each operator allowing to alter or analyse a CDI object by left-multiplication. These operators also take care of preparing the data and execution kernels in GPU space.

For example if *cdi* is a CDI object, applying an Error Reduction algorithm one simply writes:

```
cdi = ER() * cdi
```

The main property of operators is that they can be multiplied by another operator to be chained, allowing arbitrarily long operations, or raised to a given integer N to execute the operator N consecutive times. For example:

```
cdi = ER()**20 * HIO()**100 * cdi
```

will apply 100 cycles of Hybrid Input-Output followed by 20 cycles of Error Reduction.

This operator-based approach presents several advantages: (i) it is very easy to combine and alter the algorithmic chain which is used for the data analysis, allowing greater flexibility for a wide range of datasets (choices may vary depending on signal/noise quality, hard or soft condensed matter, low or high energy, etc..) and (ii) operators are independent pieces of code which can very easily be expanded or replaced as needed, avoiding the risk of turning the program into a cathedral-like construction which cannot evolve. The only limit to that approach is that the way the arrays (object, observed intensity, masks) are stored in the `CDI` object must be common to all algorithms.

A list of the most important CDI operators is given in appendix 7, and an example reconstruction code is given in figure 2. Such an operator-based approach combined with the asynchronous execution can achieve close to optimal speed, as illustrated in Figure 2, even when combining algorithms.

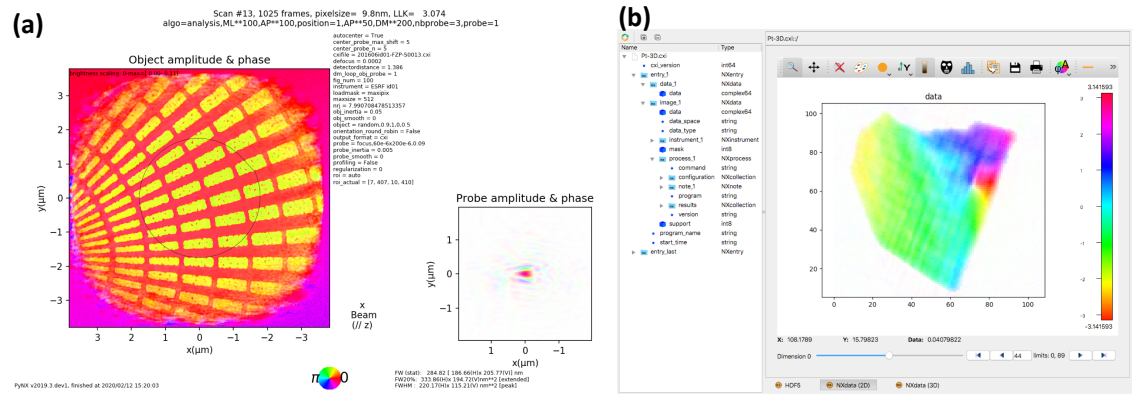

Figure 3: Example results from a command-line analysis using PyNX: (a) plot from the Ptychography analysis of a Siemens star, with the object and the probe. This plot is automatically produced at the end of the script by the 'analysis' step, and includes the algorithm chain used for the analysis as well as all parameters, which are also saved in the CXI output file. (b) View of the result of a CDI analysis on a 3D Pt nano-particle with a dislocation, using the silx toolkit viewer [41, 42]. Note on the left the details of the file contents with CXI/HDF5/NeXuS formatting. In all graphics, a hue-saturation color scheme is used to represent both the amplitude as saturation, and the phase as color, as indicated by the colour wheel at the bottom of (a). Both figures are available in a larger version as supplementary figures 8 and 9 in the appendix

In order to analyse the performance in detail, we can compare the time for a single RAAR cycle for a dataset of 512^3 voxels. As on a GPU, the speed is generally limited by the memory transfers (and not the time to perform floating-point operations), the relevant figure is the bandwidth of the process. One RAAR cycle shown in Figure 2 takes ≈29.5 ms, and requires 9.625 read and 9 writes [2] of the 3D complex array in 32-bit precision. This corresponds to a memory throughput of 677 GB/s. This can be compared to the theoretical throughput of 900 GB/s, and the observed actual throughput for a simple on-GPU copy of 790 GB/s, which shows that the asynchronous execution delivers a very good performance.

Examples of speed achieved for various CDI configurations are given in table 1, and includes the average timing for cycles, the time for updating the support and reporting the log-likelihood every 50 cycles.

The most important features of the CDI module are:

- main reconstruction algorithms include: Error Reduction [43], Hybrid Input-Output [44], Relaxed Averaged Alternating Reflections [36], Charge Flipping [45], Maximum-Likelihood conjugate gradient [46], General Proximal Smoothing [47]
- support update function based on a threshold level (relative to the maximum or the support average or root-mean-square), optionally followed by multiple steps of shrinking and expanding by a few pixels [9]
- taking into account partial coherence using a point-spread function convolution kernel [48]
- initial support determination using either a fixed geometrical form or auto-correlation [9]
- detwinning algorithms [49]

[2] 1 read + 1 write for each of the dimensions of the forward and backward FFT, 1 read and 1 write to apply the amplitude constraints to the complex array in Fourier space, 0.5 read of the intensity array (32-bit float instead of 32-bit complex), and 2.125 read + 2 writes for the RAAR projections operation (the .125 comes from reading the 8-bit support array)

Method	Configuration	Size of dataset	Algorithm chain	<time/cycle> (ms)	Δt_{total}(s)
CDI	support update every 50 cycles	512×512×512	ER()**200 * RAAR()**600	27(ER) 30(RAAR)	24
Ptychography	far field, 1 probe	1000×(256×256) object: 1176×1188	ML()**100*DM()**100	17 (DM) 34 (ML)	6.6
Ptychography	far field, 3 probe	1000×(256×256) object: 1176×1188	ML()**100*DM()**100	44 (DM) 92 (ML)	15
Ptychography	near field, 1 probe	17×(2048×2048)	ML()**100*DM()**400	39 (DM) 84 (ML)	25
MPI-Ptychography	far field, 1 probe	70.10^3×(512×512) object: 16940×16300	ML()**200*DM()**400	480 (DM) 890 (ML)	409
MPI-Ptychography	far field, 1 probe	250.10^3×(256×256) object: 15653×15179	ML()**200*DM()**400	375 (DM) 760 (ML)	329

Table 1: Example speed achieved using a single nVidia V100 GPU (or 12 GPU using MPI for the last two lines), for coherent diffraction imaging and Ptychography data analysis, using the CUDA language. The total time reported only include algorithm time, not loading or saving results, but includes some overhead (typically 5-20%) compared to individual cycles, due to fetching data from the GPU (log-likelihood reporting), initialising GPU kernels, or extra operations (scaling, check for drifts,..). The CDI algorithm includes updating the support and computing the log-likelihood every 50 cycles. Ptychography algorithms update both object and probe, and the time per cycle scales linearly with the number of frames. Note that while powers-of-two sizes are reported here, the FFT libraries used allow sizes with prime number decomposition factors up to 7 (CUDA) or 13 (OpenCL). The CDI speeds are close to optimal, e.g. achieving an average memory throughput of 677 GB/s during a RAAR cycle, but some improvements can still be added for Ptychography algorithms which are more complex, notably for some parts (e.g. position updates, not shown here) which do not achieve high memory throughput. Note finally that for relatively small datasets (less than a minute data processing), the initialisation of scripts (kernel compilations, random number generation on large arrays, ...) and input/output time can require relatively large amounts of time (>10 seconds) which affect the overall performance - this can however be mitigated by chaining the analysis of multiple datasets with the same configuration, thus avoiding unnecessary initialisation.

- read and write data or reconstructed object using the Coherent X-ray Imaging (CXI) format [3] which is based on hdf5, also using NeXus formatting [50]. This in turn allows the automatic display of relevant data when opening the files with the silx toolkit [41, 42], as shown in Fig. 3b)
- functions to simulate data, both for testing and educational purposes.
- using the 'free-log-likelihood' figure of merit which provides an unbiased metric for CDI [51] and allows the automatic selection of the best object estimate without *a priori* knowledge of its support [3]
- automated testing functions to check the code (including consistency between OpenCL and CUDA calculations)

4 Ptychography

Ptychography was first developed for electron microscopy [53, 54, 55] and then exploited for coherent X-ray microscopy [56, 14]. The technique relies on the coherent scattering of an extended object with a shifting illumination. Exploitation of the redundancy of the overlapped illuminated areas yields robust reconstructions with a variety of algorithms [15, 16, 46, 57, 58, 30], for which several software packages are available [26, 28, 27, 31].

The implementation of the `ptycho` module in PyNX follows the same principle as for the CDI one, with a separation between data and the mathematical operators, with three types of objects:

1. a `PtychoData` object includes all experimental parameters: observed intensity, probe translation positions, detector distance, X-ray wavelength, bad pixel mask, near or far field flag
2. a `Ptycho` object includes a `PtychoData` object, the current object and probe estimates (with optionally several modes [59]), an incoherent background, and an array $\Psi_j(\mathbf{r})$ which is the view of the multiplication of the object and probe at different positions, and can be propagated from sample to detector space.
3. a family of Ptychography *operators*, each one allowing to alter a `Ptycho` object using the same left-multiplication approach. An operator may alter either object, probe, Ψ arrays, or perform another task (display, export..)

Note that in practice the Ψ array is set to a fixed number of N frames, typically a stack between 16 to 128 (it does not need to be a power of two) but larger values can be used, which are simultaneously computed to achieve higher performance through parallelism, while avoiding excessive memory usage. Operators will then loop over the stack of frames to take into account the entire dataset.

The operators can be used in exactly the same way as for CDI, e.g. doing 100 cycles of difference map and then maximum likelihood on a `Ptycho` object `p` would be written:

```
p =  ML(update_obj=True,update_probe=True)**100 * \
     DM(update_obj=True,update_probe=True)**100 * p
```

Most of the high-level operators include a number of options which tailor the calculations accordingly, e.g. for the `ML` operator, the full list with default values is:

```
ML(update_object=True, update_probe=False,
   update_background=False, floating_intensity=False,
   reg_fac_obj=0, reg_fac_probe=0,
   calc_llk=False, show_obj_probe=False, fig_num=-1,
   update_pos=False, update_pos_mult=1,
   update_pos_max_shift=2, update_pos_history=False)
```

[3] the complete analysis code used for the figures of that article is available (along with datasets) from [52], and can be used as examples of CDI analysis with PyNX.

These options control not only what is optimised but also log-likelihood printing and graphical display.

The most important features of the `ptycho` module are:

- main reconstruction algorithms include: Difference Map (DM) [60], Maximum Likelihood (ML) conjugate gradient (based on Poisson noise) [46], Alternating Projections (AP) [61]
- near and far-field geometries
- position optimisation (during AP and ML) following the method presented by Odstrčil *et al* [30]. Note that as this method relies on comparing the shift of the back-propagated exit wave to the object gradient, a filter has been added to avoid shifting the positions when the norm of the object gradient is too small. See an example application if Figure 4
- incoherent background optimisation (AP) [61]
- floating intensities (currently only in OpenCL for AP and ML algorithms) [27]
- multiple incoherent modes for the probe [59]
- smoothing parameters for object and probe update (via regularisation for ML, and inertia with a convolution kernel for AP and DM) [27]
- functions to simulate data, both for testing and educational purposes
- reporting of log-likelihood based on Poisson, Gaussian and Euclidian noise models [46]
- read and write data or reconstructed object and probe using the hdf5-based Coherent X-ray Imaging format [3]
- automated testing functions to check the code (including consistency between OpenCL and CUDA calculations)

Up to early 2020, ptychography datasets collected at ESRF did not require large amounts of GPU memory, and could be analysed with only 16 or 32 GB. However that is quickly changing with new synchrotron sources which can produce a much higher coherent flux [22, 23, 24]. It is therefore useful to exploit multiple GPU or computing nodes [26, 27, 31] to handle larger ptychography datasets.

PyNX now includes the ability to analyse datasets with multiple GPU and/or multiple computing nodes, using the Message Passing Interface through the *mpi4py* python module [62, 63]. For this, a new `PtychoSplit` class has been derived from the `Ptycho` one, which manages the coordination between all processes. This follows the *asynchronous* approach presented by Nashed *et al*[26], minimising latency due to the synchronisation (as object and probe arrays need to be copied from the GPU to the host memory, and then between compute nodes).

The process follows the different steps:

1. the script (see section 5) loads the scan positions, and distributes them among the different process. This is done by using the k-means algorithm from the `scikit-learn` python module [64], after which the distribution of scanning positions is adjusted between neighbour sets to reach an homogeneous number of positions per process, and finally every set has to share at least 20 scanning positions common to neighbouring sets, for later synchronisation.
2. each process loads the images
3. object and probe are initialised in the master process and each part is distributed among the different process
4. the analysis algorithm is performed independently among the different process, as for a normal ptychography analysis

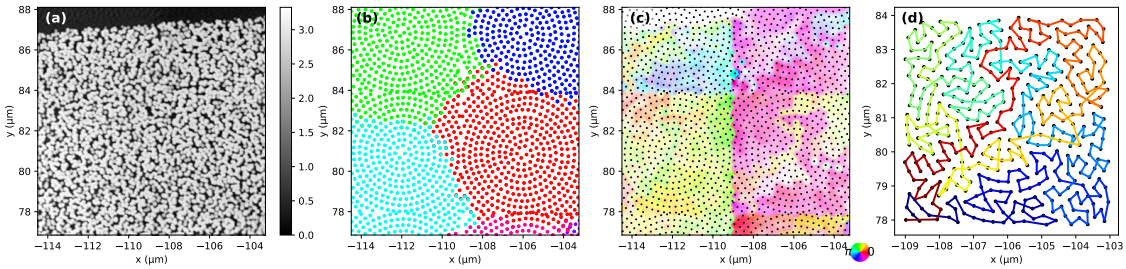

Figure 4: Example Ptychography distributed analysis MPI. The object array is about 3000×3000 pixels, $30 \times 30 \mu m^2$ with a pixel size of 10 nm, and was reconstructed from 6560 scan positions, composed of 16 scans of 410 points (only 6 are partially visible in the shown area). (a) phase of a $11 \times 11 \mu m^2$ area of the object. (b) distribution of the scanning positions between the 12 GPU - note that some points are shared between neighbouring sets of points, for synchronisation. (c) heat map of the position optimisation, where the colour indicates the direction, and the saturation indicates the amplitude of the displacement relative to the maximum (153 nm). (d) the order with which the scanning positions are visited to minimise motor displacements, following the line from red->yellow->green->blue. This scanning order is visible in (c), where the line corresponding to the red part of the scan in (d) is lighter than the surrounding points, which is particularly visible in the top right and center right scans. Note that after stitching, the borders between the different distributed parts in (b) are not visible in either (a) or (d), which indicates that both phases and positions have been correctly synchronised. Also note at the top left of in (a) that there is no contrast in the object, which is why the position updates (c) have been inhibited in this area due to the lack of contrast. A wider view of (a), (b) and (c) is available as supplementary figures 10 and 11

5. the different parts of the object are stitched together (this can also be done during the algorithm e.g. if the user requests a graphical update of the object).

The stitching step has to take into account the different ways the object and probe can differ between independent processes, as only their multiplication can be quantified, unless an image without the object has been included in the dataset. Thus object and probe can have different relative scale factors, a different phase shift, and also a different phase ramp (linearly varying over the 2D array dimensions, in opposite ways for the object and probe). Finally, if the scan positions are updated, the average shifts can differ between the parallel processes.

The relative phase ramp of the object and probe is first removed in each process independently using the `ZeroPhaseRamp` operator, which computes the center of mass of the square norm of the Fourier transform of the probe, and then corrects both probe and object for the phase ramp corresponding to the sub-pixel shift relative to the center of the array in Fourier space.[4,5]

The stitching is then done in the following way:

1. the probes are aligned (up to a pixel precision)

2. the scan positions (if they have been updated) are merged and the relative shifts minimised (using the shared positions between neighbouring sets of points)

3. the phase differences between different parts are minimised, by using small areas of the object around the shared scanning positions.

[4]This is also applied for non-distributed ptychography, so that independent optimisations of the same dataset will yield the same phase ramp for the object.

[5]Note that this approach can also be used to remove the object phase ramp for far field ptychography: by computing the sum of all calculated diffraction frames and measuring the sub-pixel shift of the center of mass of this intensity, the average phase ramp in the object can be determined. However, as the object can have a varying thickness or composition, this is not an absolute method to derive the correct phase, which can only be obtained if a reference direct beam is used, as shown by Diaz *et a*[65].

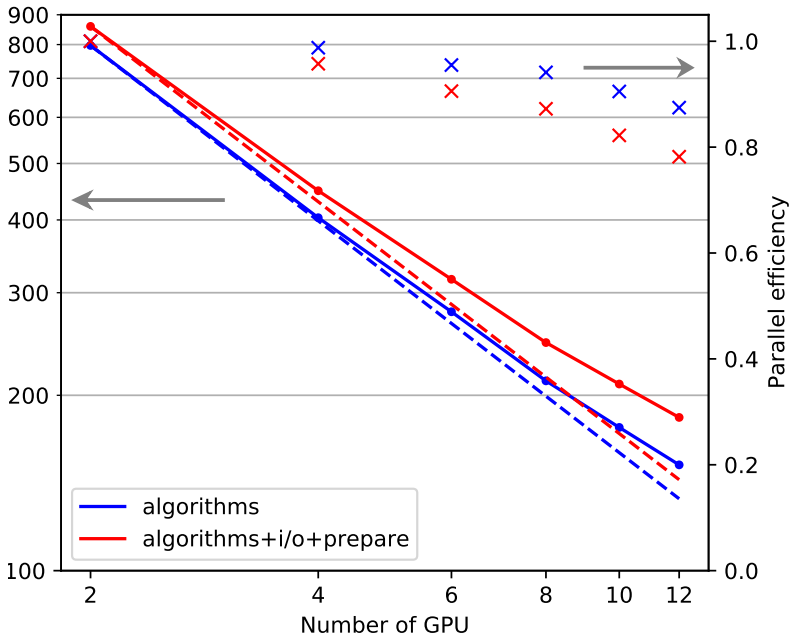

Figure 5: Performance of the MPI-distributed ptychography reconstruction presented in Fig. 4 with a dataset comprising 6560 frames with 400×400 pixels, using 1 to 6 compute nodes with two Nvidia v100 GPU each. The blue and red lines indicate the compute time in seconds, either only for the algorithmic parts (including the final object stitching), or also including the input/output and preparation. The relative efficiency of the parallelism is indicated by the crosses, and remains above 87% (78% with i/o) for 12 GPUs, even though each GPU is only handling a relatively small dataset with about 570 positions.

4. the object is finally stitched by a linear combination of the different object parts, weighted by the object illumination for each part.

Note that in principle, this *asynchronous* approach with stitching at the end of the process also allows for a *sequential* analysis of the different parts of the object, which could only be stitched together after all parts have been processed. This would be useful when an insufficiently large enough GPU cluster is available, and would allow to handle very large datasets, as long as the final object does not exceed available memory.

An example result of distributed optimisation is shown in Figure 4, for a modulator object reconstructed from 6560 scan positions, with diffraction images cropped to 400 × 400 pixels. The analysis was done using 3 probe modes, first with 200 DM cycles, then 100 AP cycles, then 1000 AP cycles with positions updates, then 100 ML cycles.

This analysis was performed on the ESRF GPU cluster using from 1 to 6 compute nodes, each with two Nvidia V100 GPUs with 32 GB of memory. The performance of the parallel calculation is shown in Figure 4, with the overall time (including i/o and object/probe initialisation) going from 860 to 183 seconds using two or twelve GPU. Note that when using 6 nodes, the number of images per GPU is only about 570, which is fairly low and contributes to decreasing the overall parallel efficiency at 78%.

Other large-scale tests are reported in Table 1, with 70k 512 × 512 frames and 250k 256 × 256 frames, spread over 12 GPU.

Technique	algorithm chain	details
CDI	`ER**50,(Sup*ER**5*HIO**50)**10`	50 cycles of HIO followed by 5 cycles or ER and support update, repeated 10 times, then 50 cycles of ER
CDI	`ER**50,(Sup*PSF*HIO**50)**4, positivity=0,(Sup*HIO**50)**8`	50 cycles of HIO followed by support update, repeated 8 times, then deactivation of a positivity constraint,then 50 cycles of HIO with the Point-Spread-Function partial coherence kernel update and support update, repeated 4 times, then 50 cycles of ER
Ptycho	`ML**40,DM**100,probe=1`	Activate probe optimisation, then 100 DM and 40 ML cycles, followed by an analysis of the probe (determination of focal point, widths, and mode decomposition)
Ptycho	`ML**100,DM**200,nbprobe=3,ML**40, DM**100,probe=1,DM**20`	First 20 DM cycles with object update only, then 100 DM and 40ML whilst also updating the probe, then use 3 probe modes and do 200 DM followed by 100 ML cycles
Ptycho	`(ML**10*AP**20)**3,position=1, AP**50*DM**50,nbprobe=2,probe=1`	First 50 DM cycles with object and probe update and 2 probe modes, followed by 50 AP and 50 ML, then activate probe position optimisation, then perform 20 AP and 10 ML cycles 3 times

Table 2: Example of algorithm chains which can be used for CDI or Ptychography analysis using the command-line scripts. Each chain will be executed from right to left, as when applying an operator to a mathematical array in an equation. Each step can either be a modification of a default parameter (e.g. the `positivity` for CDI, the number of probe modes (`nbprobe`) for ptychography, or a more specific task (`analysis`,)... Alternatively a chain of algorithm operators can be given (`ER`, `DM`, etc..). After each step which is comma-separated, the result can be saved to a CXI file. This approach allows for a great flexibility of the algorithm, without any compromise on the performance since all optimisation steps are queued asynchronously on the GPU.

5 Command-line scripts

In addition to the Python programming interface, both CDI and Ptychography datasets can be analysed using command-line scripts, either at the beamline during an experiment, or afterwards. These scripts accept parameters which give access to a range of options, as well as specifying the algorithm in a simple mathematical-like string, which can be easily interpreted using the operator approach.

5.1 CDI

For example, the analysis of a CDI dataset stored in a CXI file, can be simply written:

```
pynx-id01cdi data=pt.cxi support_threshold=0.2
```

which would perform a simple analysis based on default parameters (600 cycles of HIO followed by 200 cycles of ER, including a support update and printing the log-likelihood every 50 cycles).

Alternatively, the exact algorithmic chain can be specified using the `algorithm` keyword:

```
pynx-id01cdi data=pt.cxi algorithm=ER**200*HIO**600
             support_threshold=0.2 liveplot
```

This approach allows the full customisation of the sequence of algorithms used, mixing all standard algorithms (ER, HIO, RAAR,...) along with other algorithms (support update, partial

coherence) or parameters (positivity constraint..). More examples of how the `algorithm=` keyword can be used to customise the actual algorithm chain is given in table 2.

As indicated by the name `-pynx-id01cdi-` this script is tuned for the ESRF id01 beamline [66], with the default parameters optimised for Bragg CDI. Another script `-pynx-id10cdi-` is also available, with different parameters, notably disabling the use of the auto-correlation to determine the initial support as on the ESRF id10 beamline, a central stop is used, making that method ineffective. Other customised scripts (e.g. to handle different types of input files) can easily be added.

These scripts can exploit multiple GPU on one or multiple nodes to either distribute the analysis of multiple scans, or when multiple analysis runs are performed on the same dataset (e.g. to select the best solution based on the free log-likelihood analysis [51]) by performing the calculations on any number of parallel processes.

Many other options are available for the command-line scripts, as listed in the online documentation [6].

5.2 Ptychography

The scripts for ptychography analysis follow the same principles, for example when reading a CXI data file (which includes all observed frames, probe positions, detector distance, mask, wavelength), one can use:

```
pynx-cxipty.py data=data.cxi
 probe=focus,60e-6x200e-6,0.09
 algorithm=analysis,ML**100,DM**200,nbprobe=3,probe=1
 saveplot liveplot
```

This will trigger the data analysis, starting from simulating the initial probe as a rectangular aperture of $60(h) \times 200(v)$ μm^2 focused by a lens with a focal length of 9 cm, then optimising the object and the probe (3 modes) with 200 cycles of DM and 100 cycles of ML, followed by an analysis of the resulting probe. The `saveplot` option allows to save images, including a view of both object and probe as depicted in Fig. 3a), or of the probe analysis results (width, propagation to the focal point, modes), and a map of the scan position shifts like in Fig. 4c) if these were optimised.

Other scripts are available to handle directly data from different beamlines (id01, id13 and id16A at ESRF, NanoMAX at MaxIV, Nanoscopium and Cristal at Soleil), or from different software such as PtyPy [27]. These scripts all use the same base code, the main change being the functions to load data from the various input files and formats, so that it can easily be extended to other input formats.

6 Conclusion

To conclude, the PyNX toolkit provides a wide range of modules for the simulation and analysis of coherent imaging data, which transparently exploits accelerated computing on GPUs using either the OpenCL or CUDA language. The programming approach, which mimics mathematical operators, affords a great flexibility in the choice of algorithm chains to be used for data analysis.

Command-line scripts are also available to handle CDI and ptychography datasets without any programming knowledge, and new ones can easily be added.

PyNX is open-source (CeCILL-B license[7] similar to the BSD one) and freely available, distributed by ESRF from http://ftp.esrf.fr/pub/scisoft/PyNX/. This includes installation scripts (available for Linux and macOS) to create python virtual environments with all necessary dependencies. The online documentation includes a number of examples as jupyter notebooks. The git repository is also accessible on the ESRF gitlab server (https://gitlab.esrf.fr) on demand.

[6] http://ftp.esrf.fr/pub/scisoft/PyNX/doc

[7] https://cecill.info/

7 CDI operators and example code

The list of the main CDI operators is given below. All operations (with the exception of graphical display) are executed transparently on the GPU, and are implemented both for the OpenCL and CUDA languages.

Name	CDI Operator
`AutoCorrelationSupport`	Initialise the object support from the intensity auto-correlation [9]
`FT`	Fourier Transform
`IFT`	Inverse Fourier Transform
`ApplyAmplitude`	In Fourier space, replace the modulus by the observed amplitude
`FourierApplyAmplitude`	`IFT() * ApplyAmplitude() * FT()`
`ERproj`	Set the object to zero outside the support (support projection)
`ER`	`ERproj() * FourierApplyAmplitude()`
`EstimatePSF`	Update the point-spread-function kernel to take into account partial coherence [48]
`HIO`	Hybrid Input-Output (also uses `FourierApplyAmplitude`)
`LLK`	Compute the Poisson log-likelihood from the calculated and observed intensities
`PRTF`	Compute and plot the Phase Retrieval Transfer Function [38, 39]
`RAAR`	Relaxed Averaged Alternating Reflections (also uses `FourierApplyAmplitude`)
`ShowCDI`	Plot the current estimate of the object (amplitude, phase) with the observed and calculated amplitude
`SupportUpdate`	Update the support based on a threshold relative to the maximum amplitude in the object optionally expanding or shrinking the support afterwards [9]

8 Ptychography operators

The list of the main Ptychography operators is given below. All operations are executed transparently on the GPU, and are implemented both for the OpenCL and CUDA languages. Some operators actually apply to a stack of N frames (N=16 to 128), whereas others apply to all frames by looping over all stacks of frames.

Name	Ptychography Operator
`AnalyseProbe`	Analyse the probe (modes, determination of focus and width)
`ApplyAmplitude`	In Fourier space, replace the modulus by the observed amplitude
`AP`	Alternating projection algorithm
`DM`	Difference Map algorithm
`FT`	Fourier Transform
`IFT`	Inverse Fourier Transform
`LLK`	Compute the Poisson log-likelihood from the calculated and observed intensities
`ML`	Maximum Likelihood (conjugate gradient, Poisson noise) algorithm
`ObjProbe2Psi`	Multiply object and probe, $\Psi_j(\mathbf{r}) = O(\mathbf{r})P(\mathbf{r}-\mathbf{r_j})$
`PropagateApplyAmplitude`	`IFT() * ApplyAmplitude() * FT()` or `PropagateNearField() * ApplyAmplitude() * PropagateNearField()`
`PropagateNearField`	Near field propagation (forward or backward)
`Psi2ObjProbe`	Update the object and/or probe from the back-propagated Ψ_j
`Psi2PosShift`	Update the shift of illumination positions [30]
`ShowObjProbe`	Plot the current estimate of the object (amplitude, phase) and probe

The authors acknowledge the help of Manfred Burghammer, Julio Cesar da Silva, Virginie Chamard, Peter Cloetens, Joël Eymery, Tilman Gruenewald, Ross Harder, Stéphane Labat, Kadda Medjoubi, Linus Pithan and Tobias Schülli for useful discussions and/or test datasets. A number

of features included in PyNX are inspired from the open-source package PtyPy [27], as well as from the CDI matlab code originally written by Jesse Clark.

References

[1] V. Favre-Nicolin, J. Coraux, M.-I. Richard, and H. Renevier, "Fast computation of scattering maps of nanostructures using graphical processing units," *J. Appl. Cryst.*, vol. 44, pp. 635–640, Apr. 2011.

[2] O. Mandula, M. Elzo Aizarna, J. Eymery, M. Burghammer, and V. Favre-Nicolin, "PyNX.Ptycho- a computing library for X-ray coherent diffraction imaging of nanostructures," *J. Appl. Cryst.*, vol. 49, pp. 1842–1848, Oct. 2016.

[3] F. R. N. C. Maia, "The Coherent X-ray Imaging Data Bank," *Nature Methods*, vol. 9, pp. 854–855, Sept. 2012.

[4] P. Cloetens, R. Barrett, J. Baruchel, J.-P. Guigay, and M. Schlenker, "Phase objects in synchrotron radiation hard x-ray imaging," *J. Phys. D: Appl. Phys.*, vol. 29, no. 1, p. 133, 1996.

[5] K. A. Nugent, T. E. Gureyev, D. F. Cookson, D. Paganin, and Z. Barnea, "Quantitative Phase Imaging Using Hard X Rays," *Physical Review Letters*, vol. 77, pp. 2961–2964, Sept. 1996.

[6] P. Cloetens, W. Ludwig, J. Baruchel, D. Van Dyck, J. Van Landuyt, J. P. Guigay, and M. Schlenker, "Holotomography- Quantitative phase tomography with micrometer resolution using hard synchrotron radiation x rays," *Applied Physics Letters*, vol. 75, no. 19, p. 2912, 1999.

[7] J. Miao, P. Charalambous, J. Kirz, and D. Sayre, "Extending the methodology of X-ray crystallography to allow imaging of micrometre-sized non-crystalline specimens," *Nature*, vol. 400, pp. 342–344, July 1999.

[8] J. Miao, K. O. Hodgson, and D. Sayre, "An approach to three-dimensional structures of biomolecules by using single-molecule diffraction images," *PNAS*, vol. 98, pp. 6641–6645, June 2001.

[9] S. Marchesini, H. He, H. N. Chapman, S. P. Hau-Riege, A. Noy, M. R. Howells, U. Weierstall, and J. C. H. Spence, "X-ray image reconstruction from a diffraction pattern alone," *Phys. Rev. B*, vol. 68, p. 140101, Oct. 2003.

[10] G. J. Williams, M. A. Pfeifer, I. A. Vartanyants, and I. K. Robinson, "Three-Dimensional Imaging of Microstructure in Au Nanocrystals," *Phys. Rev. Lett.*, vol. 90, p. 175501, Apr. 2003.

[11] M. A. Pfeifer, G. J. Williams, I. A. Vartanyants, R. Harder, and I. K. Robinson, "Three-dimensional mapping of a deformation field inside a nanocrystal," *Nature*, vol. 442, pp. 63–66, July 2006.

[12] I. Robinson and R. Harder, "Coherent X-ray diffraction imaging of strain at the nanoscale," *Nat Mater*, vol. 8, pp. 291–298, Apr. 2009.

[13] V. Favre-Nicolin, F. Mastropietro, J. Eymery, D. Camacho, Y. M. Niquet, B. M. Borg, M. E. Messing, L.-E. Wernersson, R. E. Algra, E. P. A. M. Bakkers, T. H. Metzger, R. Harder, and I. K. Robinson, "Analysis of strain and stacking faults in single nanowires using Bragg coherent diffraction imaging," *New J. Phys.*, vol. 12, p. 035013, Mar. 2010.

[14] P. Thibault, M. Dierolf, A. Menzel, O. Bunk, C. David, and F. Pfeiffer, "High-Resolution Scanning X-ray Diffraction Microscopy," *Science*, vol. 321, pp. 379–382, July 2008.

[15] P. Thibault, M. Dierolf, O. Bunk, A. Menzel, and F. Pfeiffer, "Probe retrieval in ptychographic coherent diffractive imaging," *Ultramicroscopy*, vol. 109, pp. 338–343, Mar. 2009.

[16] A. M. Maiden and J. M. Rodenburg, "An improved ptychographical phase retrieval algorithm for diffractive imaging," *Ultramicroscopy*, vol. 109, pp. 1256–1262, Sept. 2009.

[17] M. Stockmar, P. Cloetens, I. Zanette, B. Enders, M. Dierolf, F. Pfeiffer, and P. Thibault, "Near-field ptychography: phase retrieval for inline holography using a structured illumination," *Sci. Rep.*, vol. 3, May 2013.

[18] M. Stockmar, M. Hubert, M. Dierolf, B. Enders, R. Clare, S. Allner, A. Fehringer, I. Zanette, J. Villanova, J. Laurencin, P. Cloetens, F. Pfeiffer, and P. Thibault, "X-ray nanotomography using near-field ptychography," *Opt. Express*, vol. 23, p. 12720, May 2015.

[19] V. Chamard, M. Allain, P. Godard, A. Talneau, G. Patriarche, and M. Burghammer, "Strain in a silicon-on-insulator nanostructure revealed by 3D x-ray Bragg ptychography," *Sci. Rep.*, vol. 5, p. 9827, May 2015.

[20] S. O. Hruszkewycz, M. Allain, M. V. Holt, C. E. Murray, J. R. Holt, P. H. Fuoss, and V. Chamard, "High-resolution three-dimensional structural microscopy by single-angle Bragg ptychography," *Nat Mater*, vol. 16, pp. 244–251, Nov. 2016.

[21] K. Wakonig, A. Diaz, A. Bonnin, M. Stampanoni, A. Bergamaschi, J. Ihli, M. Guizar-Sicairos, and A. Menzel, "X-ray Fourier ptychography," *Sci. Adv.*, vol. 5, p. eaav0282, Feb. 2019.

[22] U. Johansson, T. Johansson Falk, S. Leemann, U. Mueller, M. Sjöström, and M. Thunnissen, "MAX IV is Ready to Make the Invisible Visible," *Synchrotron Radiation News*, vol. 29, pp. 16–25, Nov. 2016.

[23] P. Raimondi, "ESRF-EBS: The Extremely Brilliant Source Project," *Synchrotron Radiation News*, vol. 29, pp. 8–15, Nov. 2016.

[24] C. G. Schroer, C. Baumbach, R. Döhrmann, S. Klare, R. Hoppe, M. Kahnt, J. Patommel, J. Reinhardt, S. Ritter, D. Samberg, M. Scholz, A. Schropp, F. Seiboth, M. Seyrich, F. Wittwer, and G. Falkenberg, "Hard x-ray nanoprobe of beamline P06 at PETRA III," *AIP Conference Proceedings - 12th International Conference on Synchrotron Radiation Instrumentation (SRI2015)*, vol. 1741, p. 030007, 2016.

[25] V. Favre-Nicolin, Y. Chushkin, P. Cloetens, J. C. da Silva, S. Leake, B. Ruta, and F. Zontone, "Dynamics and Imaging Using Coherent X-rays at the European Synchrotron," *Synchrotron Radiation News*, vol. 30, pp. 13–18, Sept. 2017.

[26] Y. S. G. Nashed, D. J. Vine, T. Peterka, J. Deng, R. Ross, and C. Jacobsen, "Parallel ptychographic reconstruction," *Opt. Express*, vol. 22, p. 32082, Dec. 2014.

[27] B. Enders and P. Thibault, "A computational framework for ptychographic reconstructions," *Proc Math Phys Eng Sci*, vol. 472, Dec. 2016.

[28] S. Marchesini, H. Krishnan, B. J. Daurer, D. A. Shapiro, T. Perciano, J. A. Sethian, and F. R. N. C. Maia, "SHARP- a distributed GPU-based ptychographic solver," *J Appl Crystallogr*, vol. 49, pp. 1245–1252, Aug. 2016.

[29] Y. S. Nashed, T. Peterka, J. Deng, and C. Jacobsen, "Distributed Automatic Differentiation for Ptychography," *Procedia Computer Science*, vol. 108, pp. 404–414, 2017.

[30] M. Odstrčil, A. Menzel, and M. Guizar-Sicairos, "Iterative least-squares solver for generalized maximum-likelihood ptychography," *Optics Express*, vol. 26, p. 3108, Feb. 2018.

[31] Z. Dong, Y.-L. L. Fang, X. Huang, H. Yan, S. Ha, W. Xu, Y. S. Chu, S. I. Campbell, and M. Lin, "High-Performance Multi-Mode Ptychography Reconstruction on Distributed GPUs," *2018 New York Scientific Data Summit (NYSDS)*, pp. 1–5, Aug. 2018. arXiv: 1808.10375.

[32] K. Wakonig, H.-C. Stadler, M. Odstrčil, E. H. R. Tsai, A. Diaz, M. Holler, I. Usov, J. Raabe, A. Menzel, and M. Guizar-Sicairos, "PtychoShelves, a versatile high-level framework for high-performance analysis of ptychographic data," *J Appl Crystallogr*, vol. 53, pp. 574–586, Apr. 2020.

[33] F. R. N. C. Maia, T. Ekeberg, D. van der Spoel, and J. Hajdu, "Hawk- the image reconstruction package for coherent X-ray diffractive imaging," *J Appl Crystallogr*, vol. 43, Oct. 2010.

[34] M. C. Newton, Y. Nishino, and I. K. Robinson, "Bonsu- the interactive phase retrieval suite," *J. Appl. Cryst.*, vol. 45, pp. 840–843, July 2012.

[35] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. . . Contributors, "SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python," *arXiv:1907.10121 [physics]*, July 2019. arXiv: 1907.10121.

[36] D. R. Luke, "Relaxed averaged alternating reflections for diffraction imaging," *Inverse Problems*, vol. 21, p. 37, Feb. 2005.

[37] A. Klöckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, and A. Fasih, "PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation," *0911.3456*, Nov. 2009.

[38] H. N. Chapman, A. Barty, S. Marchesini, A. Noy, S. P. Hau-Riege, C. Cui, M. R. Howells, R. Rosen, H. He, J. C. H. Spence, U. Weierstall, T. Beetz, C. Jacobsen, and D. Shapiro, "High-resolution ab initio three-dimensional x-ray diffraction microscopy," *J. Opt. Soc. Am. A*, vol. 23, pp. 1179–1200, May 2006.

[39] Y. Chushkin, F. Zontone, E. Lima, L. De Caro, P. Guardia, L. Manna, and C. Giannini, "Three-dimensional coherent diffractive imaging on non-periodic specimens at the ESRF beamline ID10," *Journal of Synchrotron Radiation*, vol. 21, pp. 594–599, May 2014.

[40] S. Marchesini, "A unified evaluation of iterative projection algorithms for phase retrieval," *Review of Scientific Instruments*, vol. 78, no. 1, p. 011301, 2007.

[41] SILX, "SILX- Scientific Library for eXperimentalists," 2020. http://www.silx.org.

[42] T. Vincent, V. Valls, H. Payno, J. Kieffer, V. Armando Solé, P. Paleo, D. Naudet, P. Knobel, J. Garriga, M. Retegan, M. Rovezzi, H. Fangohr, P. Kenter, W. De Nolf, UUSim, V. Favre-Nicolin, C. Nemoz, F. Picca, T. A. Caswell, A. Campbell, C. J. Wright, G. Communie, J. Kotanski, T. Coutinho, N0B0dY, Schooft, and L. Pithan, "SILX- Scientific Library for eXperimentalists," June 2020.

[43] R. Gerchberg and W. Saxton, "A practical algorithm for the determination of phase from image and diffraction plane pictures," *Optik*, vol. 35, no. 2, pp. 237–246, 1972.

[44] J. R. Fienup, "Phase retrieval algorithms: a comparison," *Appl. Opt.*, vol. 21, no. 15, pp. 2758–2769, 1982.

[45] J. S. Wu and J. C. H. Spence, "Reconstruction of complex single-particle images using charge-flipping algorithm," *Acta Cryst A*, vol. 61, pp. 194–200, Feb. 2005.

[46] P. Thibault and M. Guizar-Sicairos, "Maximum-likelihood refinement for coherent diffractive imaging," *New J. Phys.*, vol. 14, p. 063004, June 2012.

[47] M. Pham, P. Yin, A. Rana, S. Osher, and J. Miao, "Generalized proximal smoothing (GPS) for phase retrieval," *Opt. Express*, vol. 27, p. 2792, Feb. 2019.

[48] J. Clark, X. Huang, R. Harder, and I. Robinson, "High-resolution three-dimensional partially coherent diffraction imaging," *Nat Commun*, vol. 3, p. 993, Aug. 2012.

[49] M. Guizar-Sicairos and J. R. Fienup, "Understanding the twin-image problem in phase retrieval," *Journal of the Optical Society of America A*, vol. 29, p. 2367, Nov. 2012.

[50] P. Klosowski, M. Koennecke, J. Z. Tischler, and R. Osborn, "NeXus: A common format for the exchange of neutron and synchroton data," *Physica B: cond matt*, vol. 241-243, pp. 151–153, Dec. 1997.

[51] V. Favre-Nicolin, S. J. Leake, and Y. Chushkin, "Free log-likelihood as an unbiased metric for coherent diffraction imaging," *Scientific Reports*, vol. 10, Feb. 2020.

[52] V. Favre-Nicolin, "Free log-likelihood as an unbiased metric for coherent diffraction imaging: figures and data," Sept. 2019.

[53] W. Hoppe, "Principles of electron structure research at atomic resolution using conventional electron microscopes for the measurement of amplitudes and phases," *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, vol. 26, no. 4, pp. 414–426, 1970.

[54] W. Hoppe, "Trace structure analysis, ptychography, phase tomography," *Ultramicroscopy*, vol. 10, no. 3, pp. 187–198, 1982.

[55] J. M. Rodenburg and H. M. L. Faulkner, "A phase retrieval algorithm for shifting illumination," *Appl. Phys. Lett.*, vol. 85, no. 20, p. 4795, 2004.

[56] J. M. Rodenburg, A. C. Hurst, A. G. Cullis, B. R. Dobson, F. Pfeiffer, O. Bunk, C. David, K. Jefimovs, and I. Johnson, "Hard-X-Ray Lensless Imaging of Extended Objects," *Phys. Rev. Lett.*, vol. 98, pp. 034801–4, Jan. 2007.

[57] J. Qian, C. Yang, A. Schirotzek, F. Maia, and S. Marchesini, "Efficient Algorithms for Ptychographic Phase Retrieval," in *Contemporary Mathematics* (P. Stefanov, A. Vasy, and M. Zworski, eds.), vol. 615, pp. 261–279, Providence, Rhode Island: American Mathematical Society, 2014.

[58] S. Marchesini, Y.-C. Tu, and H.-T. Wu, "Alternating projection, ptychographic imaging and phase synchronization," *Applied and Computational Harmonic Analysis*, vol. 41, pp. 815–851, Nov. 2016.

[59] P. Thibault and A. Menzel, "Reconstructing state mixtures from diffraction measurements," *Nature*, vol. 494, pp. 68–71, Feb. 2013.

[60] V. Elser, I. Rankenburg, and P. Thibault, "Searching with iterated maps," *Proceedings of the National Academy of Sciences*, vol. 104, no. 2, pp. 418–423, 2007.

[61] S. Marchesini, A. Schirotzek, C. Yang, H.-t. Wu, and F. Maia, "Augmented projections for ptychographic imaging," *Inverse Problems*, vol. 29, p. 115009, Nov. 2013.

[62] L. Dalcín, R. Paz, and M. Storti, "MPI for Python," *Journal of Parallel and Distributed Computing*, vol. 65, pp. 1108–1115, Sept. 2005.

[63] L. D. Dalcin, R. R. Paz, P. A. Kler, and A. Cosimo, "Parallel distributed computing using Python," *Advances in Water Resources*, vol. 34, pp. 1124–1139, Sept. 2011.

[64] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011.

[65] A. Diaz, P. Trtik, M. Guizar-Sicairos, A. Menzel, P. Thibault, and O. Bunk, "Quantitative x-ray phase nanotomography," *Phys. Rev. B*, vol. 85, Jan. 2012.

[66] S. J. Leake, G. A. Chahine, H. Djazouli, T. Zhou, C. Richter, J. Hilhorst, L. Petit, M.-I. Richard, C. Morawe, R. Barrett, L. Zhang, R. A. Homs-Regojo, V. Favre-Nicolin, P. Boesecke, and T. U. Schülli, "The Nanodiffraction beamline ID01/ESRF: a microscope for imaging strain and structure," *Journal of Synchrotron Radiation*, vol. 26, pp. 571–584, Mar. 2019.

A Appendix

A.1 OpenCL vs CUDA FFT performance

Both OpenCL and CUDA languages rely on the same hardware. Generally speaking, the performance is almost identical for floating point operations, as can be seen when evaluating the scattering calculations (Mandula et al, 2011). However the FFT performance depends on low-level tuning of the underlying libraries, namely the cuFFT and clFFT libraries, which are respectfully optimised for Nvidia and AMD devices.

The performance of both libraries has been evaluated for an Nvidia V100 GPU, for 2D and 3D FFT of all sizes for which the largest prime factor decomposition is at most 7 (note that clFFT allows up to 13, and cuFFT allows values larger than 7 but with a degraded performance). The configuration used for the comparison was: Nvidia driver 435.21, CUDA version 10.1, clFFT v2.12.2, pyopencl 2019.1.2, pycuda 2019.1.2, gpyfft git commit 2c07fa8e7674757.

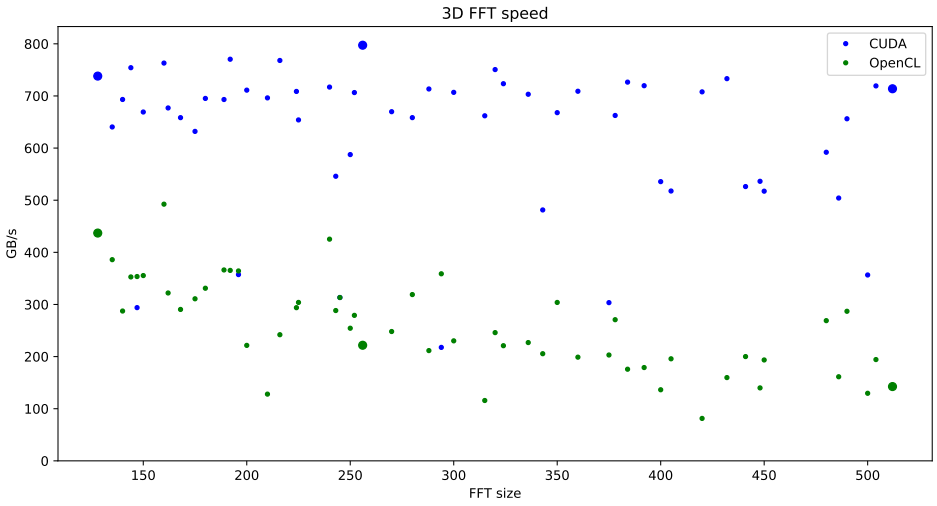

Figure 6: 3D FFT performance, measured on a Nvidia V100 GPU, using CUDA and OpenCL, as a function of the FFT size. The obtained speed can be compared to the theoretical memory bandwidth of 900 GB/s. Larger dots are shown for power-of-twos transforms

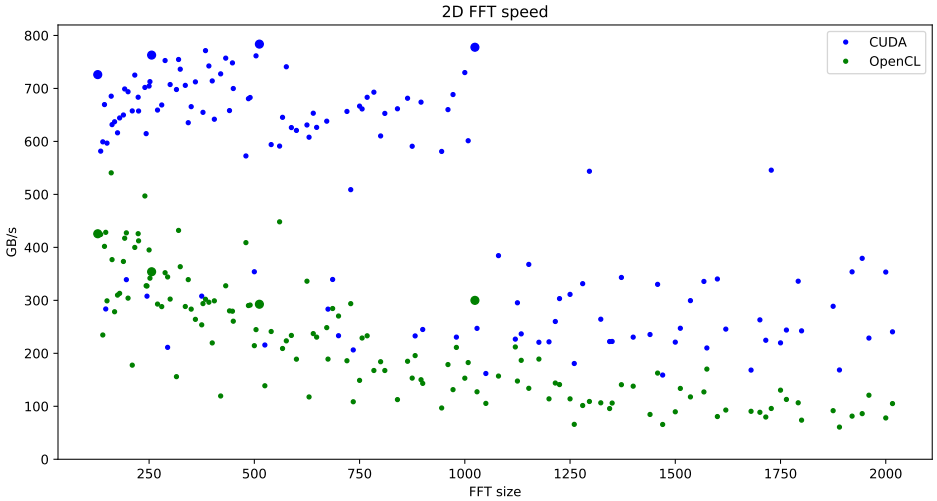

Figure 7: 2D FFT performance, measured on a Nvidia V100 GPU, using CUDA and OpenCL, as a function of the FFT size up to N=2000. The obtained speed can be compared to the theoretical memory bandwidth of 900 GB/s. Larger dots are shown for power-of-twos transforms

As performance on a GPU is limited by the memory throughput rather than the floating-point operations, we report in Fig. 6 and 7 the average processing speed in GB/s, taking into account the N read and write operations for the N-dimensional FFT. Each test is done by performing two pairs of backward and forward FT in single precision (32-bit floating point), and the test is repeated four times, the best time being kept for reporting. In the case of clFFT, each possible order for the axes transforms (a N-dimensional FT is a succession of N 1-dimensional FT) for the FT is tested and the best time is used.

Note that these tests do not imply that cuFFT is superior to clFFT *in general*, but rather that it is at least the case *on Nvidia hardware*. This is expected as clFFT is optimised for AMD GPU. One notable difference is the warp-size which is 32 for Nvidia GPU, whereas for AMD the wavefront is 64 both numbers correspond to the number of low-level compute threads which are executed in parallel on a compute unit a difference which can explain that clFFT tuning is not optimal on Nvidia hardware. Also note that in-place cuFFT transforms require 2x the amount of memory for the transform, in order to optimise memory transfers (it is faster to copy values from adjacent memory, so the transforms along the different axis of the FFT are better optimised by re-arranging the memory). clFFT does not require this (which may be a reason for a lower performance), and can thus handle larger transforms, which can be useful for large 3D CDI FFT. All the tests can be reproduced using the function: `pynx.test.speed.plot_fft_speed()`

A.2 Larger version of Figures 5a and b

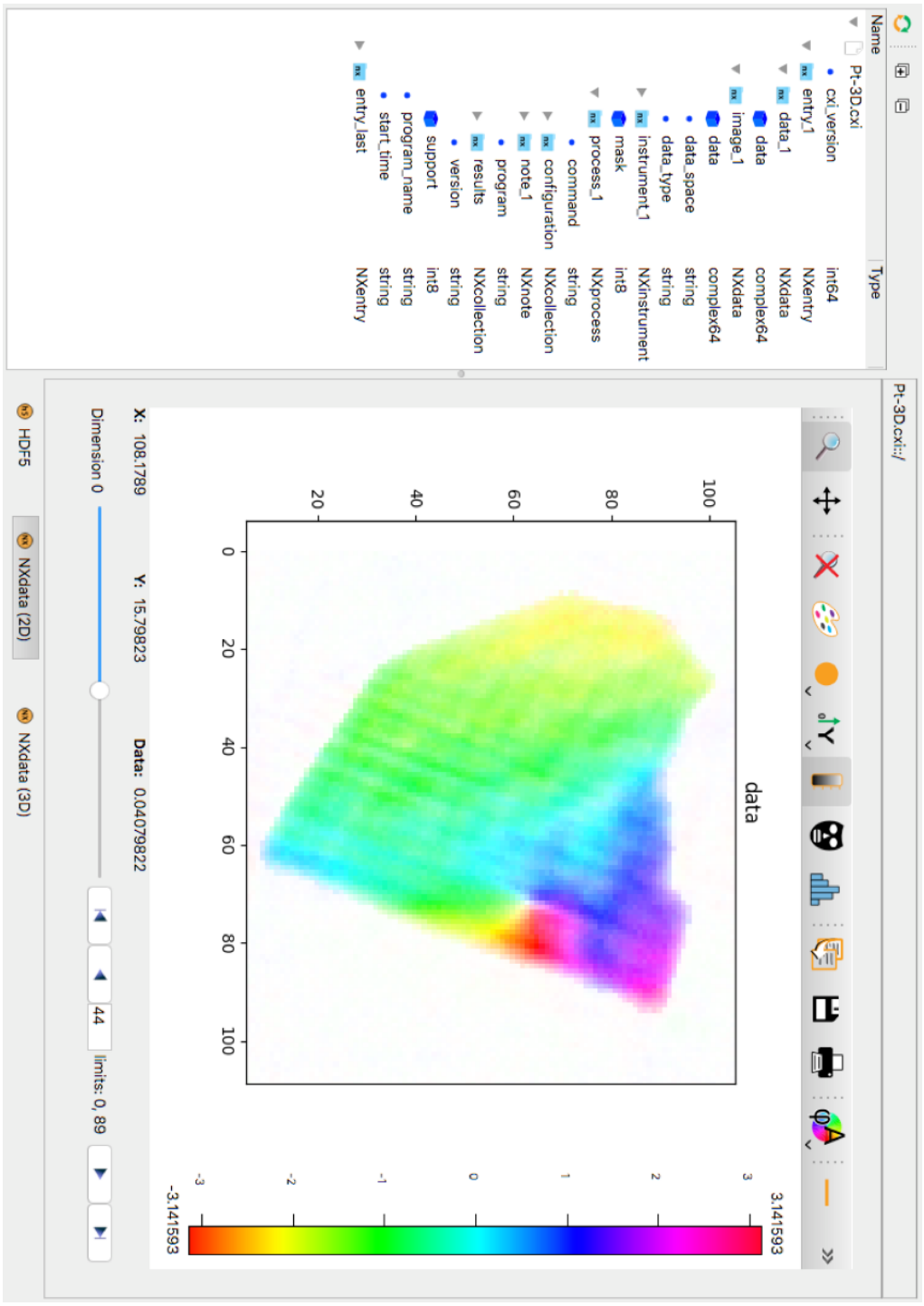

Figure 8: larger version of article Figure 3b. Display of the CXI file, using the silx toolkit viewer, obtained as a result from a CDI analysis. As the CXI file is NeXus-formatted, the view automatically opens the relevant object. The HSV colour map gives information both about the amplitude and the phase of the object. This example is the result of Bragg CDI on a Pt nano-crystal with a dislocation. As can be seen in the left of the image, different fields in the CXI/hdf5 file include information about the object as well as the process and parameters used for the analysis.

Figure 9: larger version of Fig.3a - example output plot of a ptychography experiment, showing both the object and the probe in RGBA, as well as all parameters used for the analysis

A.3 Larger view of the MPI-ptychography reconstruction of the modulator

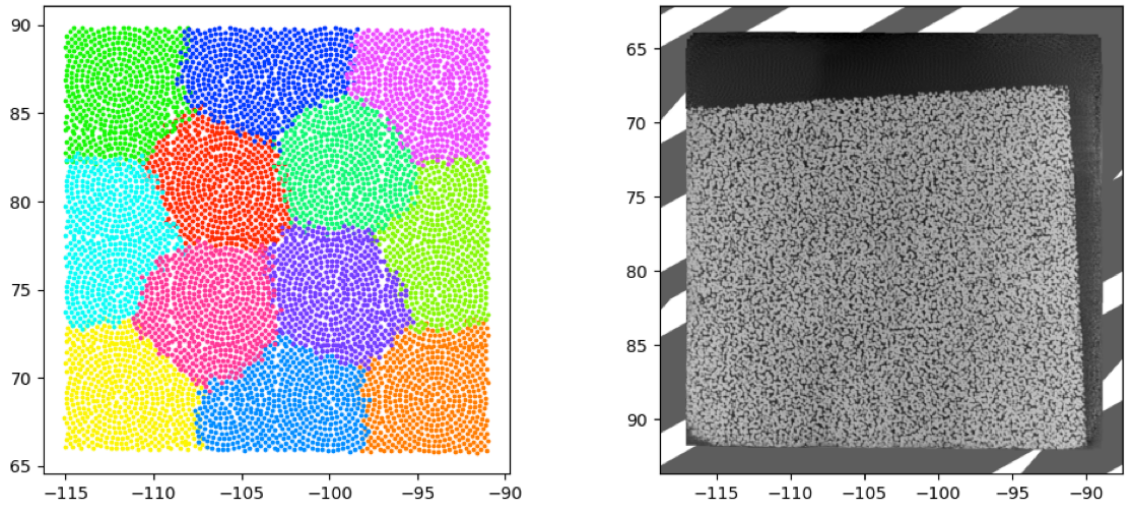

Figure 10: overview of the reconstructed modulator, shown in the article Fig. 4. Left: Division of the scanning positions into 12 domains with 575 points each, including 30 shared with the neighbours. Right: complete view of the reconstructed object phase

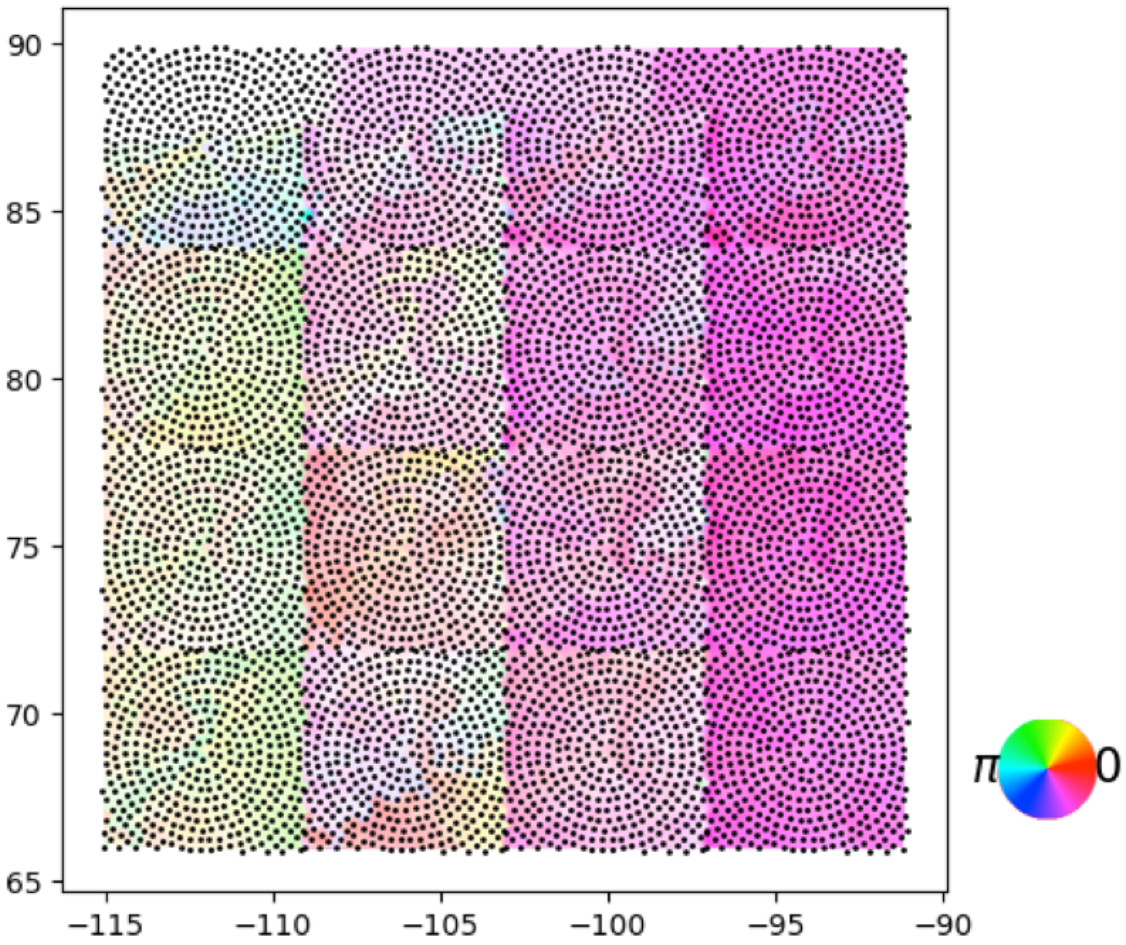

Figure 11: complete heat map of the optimised position shifts (see article Fig. 4c for explanations), which shows that the overall trend is a drift vs time for the 16 successive scans which compose the entire dataset. In this representation, the maximum displacement is 313nm, and the reference region at the top left.