TomocuPy: efficient GPU-based tomographic reconstruction with conveyor data processing

VIKTOR NIKITIN *

Advanced Photon Source, Argonne National Laboratory, 60439 Lemont, IL, USA.
E-mail: vnikitin@anl.gov

tomography; reconstruction; GPU; SSD; conveyor processing

Abstract

Real-time 3D data analysis and steering of a dynamic tomographic experiment by changing environmental conditions or acquisition parameters require fast, close to real-time, 3D reconstruction of large data volumes. Here we present a performance-optimized TomocuPy package as a GPU alternative to the commonly-used CPU-based TomoPy package for tomographic reconstruction. TomocuPy utilizes modern hardware capabilities to organize a fast 3D reconstruction conveyor involving parallel read-write operations with storage drives, CPU-GPU data transfers, and GPU computations. In the conveyor, all the operations are timely overlapped to almost fully hide all data management time. Since most cameras work with less than 16-bit digital output, we furthermore optimize the memory usage and processing speed by using 16-bit floating-point arithmetic. As a result, 3D reconstruction with TomocuPy became 20-30 times faster than its multithreaded CPU equivalent. Full reconstruction (including read-write operations and methods initialization) of a $2048^3$ tomographic volume takes less than 7 s on a single Nvidia Tesla A100 and PCIe 4.0 NVMe SSD, and scales almost
linearly increasing the data size. To simplify operation at synchrotron beamlines, TomocuPy provides an easy-to-use command-line interface. Efficacy of the package was demonstrated during a tomographic experiment on gas-hydrate formation in porous samples, where a steering option was implemented as a lens-changing mechanism for zooming to regions of interest.

1. Introduction

Fast dynamic in-situ tomographic experiments at synchrotron facilities are of great interest to various user communities including geology (Butler et al., 2020; Nikitin et al., 2020), material science (Maire et al., 2016; Zhai et al., 2019), and energy research (Finegan et al., 2015; Liu et al., 2019). This is because modern synchrotron light sources of the 3rd and 4th generation provide necessary photon flux to accommodate very fast scanning of large samples with micrometer and nanometer spatial resolution (Willmott, 2019; De Andrade et al., 2021; Nikitin et al., 2022). At the same time, modern detectors allow for continuous tomographic data acquisition at more than 7.7 GB/s rate (Mokso et al., 2017; García-Moreno et al., 2021) generating a series of tomographic datasets representing dynamic sample states at unprecedented high temporal resolution.

One of the most challenging tasks nowadays is the efficient steering of such dynamic experiments. Both manual and AI-based experiment steering can be performed more efficiently utilizing full 3D reconstructed volumes rather than projection images. Reconstructed volumes are more informative and contain all information about the current sample state compared to the projection raw data or a subset of reconstructed slices. Therefore, performance of 3D tomographic reconstruction is critical when processing large amounts of data captured in a short period of time. Fast, close to real-time 3D reconstruction, will allow for AI data analysis and steering experiments, e.g., by
automatic changing of environmental conditions (pressure, temperature, etc.), or by triggering data capturing processes for other measuring devices (higher resolution or ultrafast detectors). One example is a tomographic experiment to study crack formation processes in materials (Singh et al., 2014; Löfl et al., 2019) or geosamples such as a Berea sandstone (benchmark reservoir rock in the oil and gas applications) (Kim et al., 2016). This process is characterized by very different time scales: initial deformation of the sample is slow without noticeable destruction; transition from elastic deformation to crack development is fast and challenging because the crack position and the time of its origination are not predefined. For the experiment, the sample should be deformed in a load cell and scanned continuously with low resolution and a large field of view. An AI-steering function could then immediately analyze reconstructed volumes, and assuming reconstruction is fast, be able to detect the time and position of the crack origination and automatically adjust the instrument to higher magnification (lens change) and selecting the imaging region with the crack (sample stack positions change) (Nikitin et al., 2022). Additionally, fast analysis of reconstructed volumes during the experiment may be used for changing the load-cell pressure to control the crack formation speed.

Another example is the experiment on gas-hydrate formation in porous samples under certain pressure and temperature conditions (Nikitin et al., 2020; Nikitin et al., 2021). Fast 3D reconstruction will allow using advanced machine learning models to automatically detect different phases in the tomographic images: sand, gas, water, or gas hydrate (Fokin et al., 2022). The AI-steering engine will be based on the segmentation of low-resolution scans and selection of the regions containing a larger amount of the hydrate or the ones where the hydrate growth is faster during a specific time period. At each time step, the AI-steer function will sort the regions by priority and automatically configure the instrument for high-resolution scanning. As a result, the
procedure will ensure high-resolution data collecting from the regions with active gas-
hydrate growth which have the most scientific interest. Fast reconstruction and data
analysis are also beneficial for varying pressure-temperature conditions according to
the real-time sample state.

Nowadays there exist many packages for tomography data reconstruction. TomoPy
(Gürsoy et al., 2014) provides a Python interface for pre-processing tomography
data and for applying filtered backprojection to recover 3D sample volumes with
parallel-beam geometry. It provides implementations of different reconstruction meth-
ods, including Gridrec (Dowd et al., 1999; Rivers, 2012) and other (Gürsoy, 2014),
additionally accelerated with CPU multiprocessing, Intel compiler directives, and
Intel Math Kernel Library (MKL). Computational complexity for reconstructing a
3D volume is $O(N^3 \log N)$, assuming that the number of projection angles and vol-
ume size in each dimension are of the order of $N$. The Gridrec implementation on
computer clusters is also available (Marone et al., 2017) and has demonstrated first
steps towards on-the-fly tomography data processing. TomoPy supports Python wrapp-
ers to run reconstruction functions from other packages. One wrapper example is
ASTRA Tomography Toolbox (van Aarle et al., 2015; Van Aarle et al., 2016), which
is also commonly used as an independent package. The ASTRA Toolbox implements
high-performance GPU primitives not only for parallel-beam tomography but also for
cone-beam tomography. Besides the regular filtered backprojection method based on
the summation over lines ($O(N^4)$ computational complexity), the package is optimized
to work with iterative reconstruction methods such as SART (Andersen & Kak, 1984),
SIRT (Gregor & Benson, 2008), and CGLS (Scales, 1987). For an iterative method,
it is possible to keep all the necessary data in the GPU memory, and thereby reduce
the data copy between the storage drive, CPU and GPU memory. In such cases, the
performance of reconstruction is mostly limited by GPU computation speed. Another
package, called UFO (Vogelgesang et al., 2016), provides a multi-threaded, GPU-enabled and distributed data processing framework. Tomographic and laminographic reconstructions are also implemented using the regular filtered backprojection method of $O(N^4)$ complexity.

Computational complexity plays an important role when reconstructing data from large detectors, or from data obtained by stitching several projection datasets (Vescovi et al., 2018; Tile, 2022). For example, for $N = 2048$ the complexity $O(N^3 \log N)$ becomes approximately 186 times lower than $O(N^4)$. With increasing data sizes, the potential acceleration becomes higher (341 complexity lowering factor for $N = 4096$, 630 - for $N = 8192$, and so on), therefore it is always beneficial to operate with algorithms of lower computational complexity with introduction of new detectors having large sensors (e.g. 13392×9528 sensor shr661 camera from SVS-VISTEK) they indeed become critical for any future tomography applications. Examples of methods with $O(N^3 \log N)$ complexity include the Fourier-based gridding method (Beylkin, 1998) and log-polar-based method (Andersson et al., 2016a).

Modern hardware resources are powerful enough to perform efficient reconstruction, often faster than the tomographic raw data acquisition time. First, the acceleration of computations can be reached using GPUs, and aforementioned Astra/UFO packages are just a couple of examples showing this high efficiency. The performance table in (Andersson et al., 2016a) reports 0.045 s log-polar-based reconstruction of 1 slice 2048×2048 on Nvidia GeForce GTX 770 (release date: May 30th 2013), which corresponds to 92 s for reconstructing the full volume. The reported time does not include initialization and data transfer costs. Modern GPUs are several tens of times faster than GTX 770 and may reduce reconstruction times to a few seconds. Another feature of modern GPUs is 16-bit (half-precision) arithmetic. Most detectors used for tomography have less than 16-bit digital output. It is therefore potentially possible to
decrease processing data sizes and accelerate computations even more.

Moreover, many modern data storage devices allow for efficient parallel functionality for accelerating read and write operations. For example, IBM General Parallel File System (GPFS) is a clustered file system where data are broken into blocks and striped across multiple disks in an array. Parallel read/write operations from blocks may result in several tens of GB/s speed. Modern storage based on Non Volatile Memory Express (NVMe) solid state disks (SSDs) also delivers unprecedented performance provided by parallelization of the read/write operations, which results in $8 \times$ acceleration compared to regular SATA SSDs (Xu et al., 2015). Besides computer clusters, current NVMe SSDs connected via PCIe v3 or PCIe v4 are also used in common workstations and demonstrate 3.5-7 GB/s speed for parallel operations with the disk. In this case, writing one tomographic volume of size $2048^3$ in 32-bit precision may potentially take less than 5 s.

Finally, modern network solutions allow for fast and efficient transfer between the data storage and processing nodes. Nvidia Mellanox switches deliver low latency and 100 Gb/s of bandwidth, which together with the GPUDirect technology (Rossetti & Team, 2015) enable new GPUs to communicate faster by eliminating the need for CPU involvement in the communication, resulting in increased overall system performance and efficiency by reducing the GPU-to-GPU communication time by 30% (Shainer et al., 2011; Rossetti & Team, 2015).

In this paper, we present a new package called TomocuPy where we combined efficient reconstruction methods and modern hardware capabilities to accelerate the whole tomographic reconstruction process including data read/write operations, CPU-GPU data transfers, and computations on GPU. In Section 2 we will describe the implementation details and an easy-to-use command-line interface for processing experimental data. Section 3 provides performance analysis on synthetic data and accuracy anal-
ysis on experimental data from a micro-CT synchrotron beamline and a comparison between different methods. An example of a dynamic tomography experiment at synchrotron where automatic steering was possible due to fast reconstruction provided by TomocuPy is presented in Section 4. Conclusions and outlook are given in Section 5.

2. Fast GPU-based reconstruction with TomocuPy

TomocuPy (Nikitin, 2022) is a Python package that provides support for a fast and efficient data conveyor, data processing and tomographic reconstruction on Nvidia graphics processing units (GPUs) with 16-bit or 32-bit computational precision. It implements GPU-based pre-processing steps and filtered backprojection operators, as well as optimized data transfer mechanisms among storage drives, CPU RAM memory and GPU memory. In what follows we will describe the main package features leading to fast, close to real-time, tomographic reconstruction.

2.1. 16-bit precision arithmetic

Area detectors used for tomographic imaging incorporate an analog to digital converter (ADC) to digitize the images with 8, 10, 12 or 16-bit output. The conventional tomography reconstruction is typically performed with 32-bit floating-point operation which might be inefficient in terms of computational speed. In this work we considered 16-bit floating-point (FP16) arithmetic as an alternative to the conventional 32-bit floating-point (FP32) arithmetic. FP16 is used in many computer graphics environments to store pixels, including Nvidia CUDA, OpenGL and Direct3D. Currently, it is also gaining popularity in deep learning applications with Nvidia GPUs. Nvidia’s recent Pascal architecture was the first GPU that offered FP16 support. FP16 arithmetic was significantly optimized for following Nvidia architectures including Volta and Ampere, and became beneficial for code optimization in terms of performance.
and memory usage.

To adapt tomographic reconstruction in TomocuPy for FP16 computations we followed the guidance from (Ho & Wong, 2017) that shows different issues and opportunities with code migration to FP16. We also reviewed all mathematical operations in the code and made sure that the accuracy and correctness of computations are not lost. The accuracy can be lost when a mathematical operation is performed between large and small numbers, e.g. $1000 - 0.1 = 1000$ (FP16), incorrect results are obtained when multiplying two large numbers: $1000 \times 1000 = \text{inf}$ (FP16) since the maximum representable value in FP16 precision is 65504. To address these issues, we reorganized arithmetic operations where it was possible. In places where the reorganization was not possible, we perform the operation with arguments converted to FP32 and cut the precision of the result back to FP16. As a result, we were able to decrease the total amount of memory (CPU RAM, GPU, and storage disk space) by 2 times and accelerate computations on GPU.

2.2. Pre-processing steps and backprojection

Pre-processing steps in tomographic reconstruction include dark-flat field correction, taking negative logarithm of the data, one-dimensional filtering with the Shepp-Logan, Parzen or other filter. Additionally, pre-processing may include ring-removal filtering with using wavelets (Münch et al., 2009) or with an analytical formula by V. Titarenko (Titarenko et al., 2010; Titarenko, 2016), zinger artifacts reduction (Rivers, 1998), or propagation-based phase-retrieval procedure with the Paganin filter (Paganin et al., 2002). To accelerate computation of all these steps we used CuPy Python library (Okuta et al., 2017), which is a GPU-accelerated analog of NumPy Python library. All regular linear algebra operations such as multiplication, summation, logarithm, exponent, are easily ported to CuPy library calls in 16-bit and 32-bit
floating-point precision. At the time of writing this manuscript, CuPy does not support computing FFTs in 16-bit precision, moreover, 16-bit FFTs are supported in CUDA C only for the sizes that are powers of 2. Therefore, we prepared CUDA C codes for allocating 16-bit and 32-bit CUDA FFT plans at the beginning of reconstruction and executing the plans on a set of tomography slices during data reconstruction by chunks. 16-bit data are additionally padded/unpadded to the power of 2 sizes, 32-bit data are padded to the sizes represented as $2^a \times 3^b \times 5^c \times 7^d$ ($a, b, c, d$ are positive integer numbers) for optimal evaluation of Bluestein’s algorithm (Bluestein, 1970) in CUDA cuFFT library.

The backprojection operator is the most computationally intensive step of the reconstruction procedure. Its direct evaluation by discretizing line integrals has computational complexity of $O(N^3 N_\theta)$ assuming that the sample size in each dimension and the total number of projection angles are of orders $N$ and $N_\theta$, respectively. There exist methods for fast evaluation of the backprojection operator. The most common one is based on using the Fourier-slice theorem and using Fourier transforms on unequally-spaced grids (Beylkin, 1998; Dowd et al., 1999). It has complexity $O(N^3 \log N)$ for reconstructing 3D volumes. The same complexity demonstrates the log-polar-based method (Andersson et al., 2016b). However, in comparison to the Fourier-based method where interpolation-like procedures are conducted in the frequency domain, the log-polar-based method assumes interpolation in the image domain where data are substantially less oscillatory. Therefore, the log-polar-based method demonstrates accurate reconstruction results with using interpolation schemes of moderate order (linear or cubic splines), whereas the Fourier-based method has to operate with exponential, or other complex-type functions that can be approximated with only high-order polynomials. The log-polar-based method outperform the Fourier-based method (Andersson et al., 2016a) because of the interpolation
type, however, its current implementation assumes that projection data are given for equally spaced angles. In very rare cases, e.g. during an interlaced scanning (Mohan et al., 2015), tomographic data are collected for non-equally spaced angles and the log-polar-based method is not applicable.

TomocuPy provides 3 implementations of the backprojection operator: (1) direct discretization of the backprojection line integral, (2) Fourier-based method with exponential functions for interpolation in the frequency domain, and (3) the log-polar-based method with cubic splines interpolation in the image domain. Although the direct line discretization method is not optimal, we keep it as an option since the method can be used for computing backprojection in a laminographic geometry (Helfen et al., 2007) where the rotary stage is tilted with respect to the beam direction, yielding more efficient scanning of flat samples. Since the backprojection is the most computationally demanding part of reconstruction, we fully implemented it with CUDA C by writing optimized codes for FFTs, CUDA raw kernels, and data handling. Users can easily switch between different backprojection methods depending on application.

2.3. Conveyor data processing

Besides data processing on GPU, tomographic reconstruction requires data transfer operations between a storage drive, CPU RAM memory and GPU memory. Non-optimal organization of data transfers among these components, especially in GPU computing, can significantly slow down the whole reconstruction pipeline, causing GPU being idle while waiting for new data chunk transfers to complete. Because of non-optimal organization of data transfers, significant GPU acceleration is typically visible for iterative tomography reconstructions, where data is loaded to GPU memory and tens or hundreds of iterations are performed while keeping the whole dataset in GPU memory. One-step filtered backprojection implemented on GPU with sequential
data transfers does not yield such performance gain compared to the CPU version because of relatively slow memory transfer operations. Here we organize and optimize a processing conveyor where all data transfers are overlapped with GPU computations. This way, time for data transfers is effectively hidden from the total computational time required by the reconstruction step.

TomocuPy implements this optimal conveyor in two levels. First, independent Python threads are started for (1) reading data chunks from the storage drive into a Python data queue object and for (2) writing reconstructed chunks from another Python queue object to the storage drive disk. Both queue objects are stored in CPU RAM memory. The size and the number of threads for each queue are defined based on the system characteristics. To maximize performance of parallel read/write operations we work with Intel® SSD D7-P5510 Series PCIe 4.0 NVMe drives. These drives work on high-end parallel data paths for faster operations than regular SAS/SATA HDDs or SSDs, protocols of which are based on CPU cycles and are not designed to handle severe data loads.

Second, independently on read/write operations with storage drives, we overlap CPU-GPU data transfers with GPU computations by using CUDA Streams. CuPy interface allows organizing the concurrent execution of streams directly within the Python code, without writing CUDA C code. It also allows for direct allocation of pinned GPU memory, which is necessary to run data transfers and GPU computations concurrently. To implement the overlap, the pinned memory on CPU and device memory on GPU should be both allocated for 2 data chunks and 2 reconstruction chunks. Three CUDA streams are running simultaneously by switching between chunks, the first stream performs a data copy to the first chunk of the pinned memory, followed by the transfer to the first chunk of GPU memory. The second stream performs GPU computations on the second data chunk in GPU memory (whenever it is available)
and places the result to the second reconstruction chunk in GPU memory. The third stream executes a data transfer from the first reconstruction chunk in GPU memory to the first pinned memory chunk for reconstruction. The chunk is then copied to the queue for further writing to the storage drive. After processing each chunk, all streams synchronize and switch the chunk id (0 or 1) they operate with.

Figure 1 shows the timeline view report from the Nvidia Nsight System performance analysis tool that demonstrates a comparison between the conveyor and sequential execution types. The test was performed for reconstructing a $2048 \times 2048 \times 2048$ dataset with the log-polar-based method and FP32 arithmetic. The timeline view for the conveyor execution is shown for 40 ms. During this time, we observe continuous data read/write operations with NVMe SSD, so as continuous CUDA kernel execution and GPU-GPU memory transfers. We also observe 2 GPU-CPU and 2 GPU-CPU memory transfers for data chunks. Since all operations are overlapped in time by using Python CPU threads and CUDA Streams, the total reconstruction time in this case can be approximately estimated only by GPU computations. In turn, the timeline view for the sequential execution with 1 CPU thread and 1 CUDA Stream shows 160 ms running time without any overlap. During this period, about 60% of the time is spent for read/write operations with an NVMe SSD in 1 thread, the remaining 40% is used for GPU computations and memory transfers between CPU and GPU. The left panels of both reports also show that the total GPU utilization consists of about 70% CUDA kernel execution and 30% memory transfers. Total time for reconstructing a $2048 \times 2048 \times 2048$ dataset with the conveyor execution, as measured by the Nvidia Nsight System performance analysis tool, is approximately 2.5 times lower than for the sequential execution (8 s vs 21 s).
Conveyor execution (40 ms timeline)

Sequential execution (160 ms timeline)

Fig. 1. Timeline view report from Nvidia Nsight System tool for the conveyor and sequential execution of reconstruction with TomocuPy. Reconstruction was performed for a $2048 \times 2048 \times 2048$ dataset with the log-polar-based method and FP32 arithmetic.

2.4. Command-line interface

To simplify the execution of tomographic reconstructions with TomocuPy, we have developed a command-line interface wrapping Python classes with processing functions. The command can be executed in a Unix terminal and accepts a list of parameters to customize the reconstruction procedure. The executable file is installed as a part of the whole TomocuPy package by using pip or conda install. An example of a command line for running in an Anaconda environment and reconstructing one full tomographic dataset stored as an HDF5 file is as follows:

```
$ tomocupy recon --file-name /data/testing_131.h5 --rotation-axis 1224 --reconstruction-type full
```
where the reconstruction parameters are submitted with the syntax `--<parameter> <value>`.

The list of all available parameters can be obtained by running `tomocupy recon -h`. The description of some parameters (approx. 20% of the full list) looks as follows:

```
--binning {0,1,2,3}  Reconstruction binning factor (default: 0)
--dtype {float32,float16}
    Data type used for reconstruction. Note float16 works
    with power of 2 sizes. (default: float32)
--fbp-filter {shepp,parzen}
    Filter for FBP reconstruction (default: parzen)
--max-read-threads MAX_READ_THREADS
    Max number of threads for reading by chunks (default: 1)
--max-write-threads MAX_WRITE_THREADS
    Max number of threads for writing by chunks (default: 4)
--nsino NSINO
    Location of the sinogram used to find rotation axis
    (0 top, 1 bottom) (default: 0.5)
--nsino-per-chunk NSINO_PER_CHUNK
    Number of sinograms per chunk. Use larger numbers with GPUs
    with larger memory. (default: 8)
--rotation-axis ROTATION_AXIS
    Location of rotation axis (default: -1.0)
--rotation-axis-auto {manual,auto}
    How to get rotation axis: auto calculate ("auto"),
    or manually ("manual") (default: manual)
--reconstruction-algorithm {fourierrec,lprec,linesummation}
    Reconstruction algorithm (default: fourierrec)
--reconstruction-type {full,try}
    Reconstruct full data set. (default: try)
--start-row START_ROW
    Start slice (default: 0)
--end-row END_ROW
    End slice (default: -1)
```

A general reconstruction procedure typically consists of two steps: 1) reconstruction of 1 slice for different rotation centers and saving reconstructed tiff files with names corresponding to these centers (parameter `--reconstruction-type try`). Users then open all files and select the rotation center by scrolling through different tiff files. The selected rotation center is then entered (`--rotation-axis 1224`) and used for the full reconstruction (`--reconstruction-type full`). TomocuPy also provides an automatic center search option (`--rotation-axis-auto auto`) by using SIFT algorithm (Lowe, 1999) to find shifts between 0 and (flipped) 180 degrees projections.

Command-line interface for TomocuPy was developed to assure compatibility with
the command-line interface TomoPy-cli (https://tomopycli.readthedocs.io) for CPU-based reconstruction. TomoPy-cli uses TomoPy package (Gürsoy et al., 2014) as a backend and implements an efficient workflow for processing tomographic data files (tiff, HDF5) from storage drives. Both packages, TomocuPy and TomoPy-cli, has the same syntax for passing parameters. They also provide the same names for most of parameters, except method-specific parameters, such as --dtype, --max-read-threads, etc. Likewise, the input/output format, file names are identical. It is therefore not complicated to switch between two packages and compare performance and quality of reconstruction results.

It is important to note that a multi-GPU version of tomographic reconstruction is straightforward to implement because in the parallel beam geometry reconstruction is done independently for different slices through the volume. TomocuPy provides parameters --start-row and --end-row for specifying the range of slices for reconstruction, therefore multi-GPU reconstruction can be performed, for instance, by setting environment variable CUDA_VISIBLE_DEVICES associated with the GPU number and running daemon processes in bash for each subset of slices,

```
CUDA_VISIBLE_DEVICES=0 tomocupy recon --file-name test2048.h5 \
  --reconstruction-type full --start-row 0 --end-row 512 &
CUDA_VISIBLE_DEVICES=1 tomocupy recon --file-name test2048.h5 \ 
  --reconstruction-type full --start-row 512 --end-row 1024 &
CUDA_VISIBLE_DEVICES=2 tomocupy recon --file-name test2048.h5 \ 
  --reconstruction-type full --start-row 1024 --end-row 1536 &
CUDA_VISIBLE_DEVICES=3 tomocupy recon --file-name test2048.h5 \ 
  --reconstruction-type full --start-row 1536 --end-row 2048 &
wait
```

Since the processes are independent the total performance will be limited only by the storage and system bus speed for data transfers.

3. Performance and accuracy analysis

To check reconstruction quality that TomocuPy package demonstrates when processing experimental datasets, we collected tomographic projections for a sample consist-
ing of 20-40 µm glass beads packed in a kapton tube with 4 mm diameter. The measurements were performed at the bending magnet micro-CT beamline 2-BM (Nikitin et al., 2022) of the Advanced Photon Source. The beamline was adjusted for using the pink beam (polychromatic X-ray beam reflected from a grazing mirror) cutting energies higher than 30 keV, and with additional 6 mm glass filtering of low energies. Projections were acquired by a CMOS detector Oryx 5.0 MP Mono 10GigE, 2448×2048 chip size, 3.45 µm pixel size, made by Teledyne FLIR LLC. The detector used a 2x magnification infinity corrected objective by a Mitutoyo resulting in 1.725 µm pixel size. The lens was focused to a 50 µm Crytur LuAG:Ce scintillator converting X-rays to visible light.

Tomographic projections were acquired in fly scanning mode, while the sample was continuously rotated for a 180 degrees interval. In total 2048 projections of the size 2048×2048 (cropped field of view for the detector) were collected with 0.05 s exposure time per projection, yielding 1.7 min total acquisition time. The reconstruction procedure was performed by using 3 reconstruction algorithms implemented in TomocuPy, and by using Gridrec method from TomoPy (with TomoPy-cli interface for data pre-processing and transfers):

- FourierRec - Fourier-based method with exponential-function interpolation in the frequency domain (Beylkin, 1998), computational complexity $O(N^3 \log N)$
- LpRec - Log-polar-based method with cubic interpolation in the space domain (Andersson et al., 2016a), computational complexity $O(N^3 \log N)$
- LineRec - Direct discretization of the line integral with linear interpolation for computing backprojection, computational complexity $O(N^4)$
- Gridrec - TomoPy implementation of the Fourier-based method (Dowd et al., 1999), computational complexity $O(N^3 \log N)$
where the computational complexity is calculated assuming the number of projection angles, and the object size in each dimension are of the order of $N$. All methods employed the commonly-used Parzen filter for implementing filtered backprojection.

Figure 2 presents a comparison for reconstructions using TomocuPy (FourierRec, LpRec, LineRec) with 32 and 16-bit floating-point arithmetic. Each image shows one reconstructed slice using different methods, together with insets showing 10x zoom-in to the region marked with the black rectangle. Visually, reconstructions for FP32 and FP16 look the same. The right part of the figure shows the difference between them, so as calculated Structural Similarity Index (SSIM) (Wang et al., 2004) quantifying image quality degradation. SSIM is higher than 0.93 for all methods, which confirms high quality of FP16 results. Note that the data were collected in 12-bit precision, i.e. in maximum precision for most of tomographic detectors used in fast imaging. We therefore can conclude that 16-bit arithmetic is enough for processing tomographic data, and all reconstructed volumes can be stored with using twice lower amount of memory.
Fig. 2. Comparison of micro glass beads reconstructions for 16 and 32-bit floating-point precision arithmetic. Inset plots show 10x zooming to the region marked by the black square. Colorbar range for the difference plot is 10 times smaller than for reconstructions.

As a second quality test, we compared TomocuPy reconstructions with the ones produced by the Gridrec method implemented in TomoPy. Gridrec is a Fourier-based method, i.e. is using the Fourier-slice theorem and fast evaluation of Fourier transforms on unequally spaced grids. The difference with its TomocuPy equivalent, called FourierRec, is in the interpolation kernels used for data re-gridding in the frequency domain, and oversampling factors for frequencies. TomoPy implementation of Gridrec does not include oversampling, therefore regular reconstruction contains phase wrapping artifacts. In order to minimize these artifacts, additional padding of sinograms
is typically performed before the filtered backprojection operation (Marone & Stampanoni, 2012). FourierRec includes oversampling by a factor of 2 and accuracy controls in computing the backprojection integral. A detailed accuracy analysis for evaluating backprojection with the Fourier and Log-polar-based methods for the Shepp-Logan phantom sample can be found in (Andersson et al., 2016). In the paper, filtered versions of the Shepp-Logan phantom, as well as corresponding projection data, are computed analytically and therefore directly used for evaluating the backprojection error for different methods. Based on the fact that the FourierRec method is the method with the highest accuracy in computing backprojection (based on the accuracy tests from (Andersson et al., 2016)), we will present results for other methods in comparison to FourierRec. Additionally, all reconstruction methods implemented in TomocuPy involve data padding for the filtering operation, which allows suppressing artifacts when processing samples not fitting into the detector field of view.

In Figure 3a we show the difference between reconstructions between Gridrec from TomoPy and FourierRec from TomocuPy. One can see that regular TomoPy Gridrec reconstruction (top row) has errors in low-frequency components, visible as amplitude changes in the regions close to the borders. SSIM is relatively low (0.731). In turn, reconstruction with additional sinogram padding (bottom row) does not have visually observed amplitude changes, however, the difference to TomocuPy FourierRec still highlights errors in low frequencies. Despite the errors in low frequencies, TomoPy is still commonly used for reconstructing tomographic data because information given by high frequencies (small features) is more important in several applications, and it is accurately recovered with SSIM=0.915. Note again that the accuracy of the methods implemented in TomocuPy was checked using analytical expressions for the Shepp-Logan phantom and its projection data (Andersson et al., 2016).

In Figure 3b we provide the difference images between LpRec and FourierRec (top),
and between LineRec and FourierRec (bottom). One can observe very high accuracy of the LpRec method where cubic interpolations to and from log-polar coordinates are done in the image domain. The LineRec method is implemented with linear interpolation in the image domain, thus errors in high-frequency components are clearly visible. Structural similarity index for these two methods are 0.998 and 0.812, respectively.

![Fig. 3. Comparison reconstruction results in 32-bit floating-point precision: a) for TomocuPy FourierRec and TomoPy Gridrec (with and without padding of sinograms) methods, and b) between TomocuPy FourierRec, LpRec and LineRec. Inset plots show 10x zooming to the region marked by the black square. Colorbar range for the difference plots is 10 times smaller than for reconstruction.](image)

For the performance analysis, TomocuPy package was tested using synthetic HDF5 datasets of different sizes. Synthetic datasets were generated for $N$ 16-bit tomographic projections with $N \times N$ detector sizes, where $N$ ranges from 512 to 16384. Reconstructed volumes ($N \times N \times N$) were obtained as sets of tiff files in 16-bit and 32-bit precision. Note that the selected projection data sizes do not satisfy the Crowther sampling criterion stating that the number of angles should be $\frac{\pi}{2}N \approx \frac{3}{2}N$ (Crowther et al., 1970). In tomographic experiments this criterion is typically relaxed, and reconstruction results with acceptable quality are demonstrated for a significantly lower
number of angles, e.g. equal to $N$ or $3/4N$.

For completeness, we also analyzed the performance of TomoPy-cli package, where all pre-processing steps and Gridrec reconstruction were accelerated using multi-threaded CPU functions and Intel® Math Kernel Library. Recall that both TomoPy-cli and TomocuPy command-line interfaces have almost the same set of parameters and in most cases can be easily interchanged.

All performance tests were performed on a machine with Intel Xeon® Gold 6326 CPU @ 2.90GHz, 1 TB DDR4 3200 memory, 1 Nvidia Tesla A100 with 40 GB memory, and Intel SSD D7-P5510 Series PCIe 4.0 NVMe disks of total capacity 84 TB. Installed software included Python 3.9, CuPy 10.4.0, Nvidia CUDA toolkit 11.6, and Intel Math Kernel Library Version 2022.1 (only for fast CPU-based computations in TomoPy).

Table 1 shows the dataset dimensions used to test the performance of tomocuPy methods (FourierRec, LpRec, LineRec) and the CPU-based TomoPy Gridrec.

Table 2 shows the total time to reconstruct the test datasets listed in Table 1 using TomocuPy’s FourierRec and LpRec methods with FP16 and FP32 precision, TomocuPy’s LineRec with FP32 precision, as well results for TomoPy-cli package where all pre-processing steps and Gridrec method for reconstruction are executed in FP32 precision.

| Test dataset | 1   | 2   | 3   | 4   | 5   | 6   |
|--------------|-----|-----|-----|-----|-----|-----|
| Size in each dimension, $N$ | 512 | 1024| 2048| 4096| 8192| 16384|
| Raw data size on SSD, (8-bit) | 128MB | 1GB | 8GB | 64GB | 512GB | 4TB |
| Reconstruction size on SSD, 16(32)-bit | 256(512)MB | 2(4)GB | 16(32)GB | 128(256)GB | 1(2)TB | 8(16)TB |

Table 1. Dataset dimensions used to test the performance of tomocuPy methods (FourierRec, LpRec, LineRec) and TomoPy Gridrec.
Table 2. Total time in seconds to reconstruct the datasets listed in Table 1 using TomocuPy’s FourierRec and LpRec methods with FP16 and FP32 precision, TomocuPy’s LineRec with FP32 precision, and TomoPy-cli package where all pre-processing steps and Gridrec method for reconstruction are executed in FP32 precision. Raw data (16-bit) and reconstructed volumes (16 or 32-bit) both have sizes \(N \times N \times N\), assuming the number of projection angles equals to \(N\). Reconstruction time includes all parts of the processing pipeline (reading HDF5 data chunks from NVMe SSD, writing reconstructed tiff files to NVMe SSD, CPU-GPU transfers, and all CPU/GPU computations).

There are several observations from Table 2. First, all methods allow working with very large data sizes (up to several TB on SSD), which is useful for processing data from the detectors with large sensors or from mosaic tomographic scans. In mosaic scans, large samples are scanned at different vertical and horizontal positions to obtain a set of datasets that are then stitched together to generate one large dataset.

Second, we observe that FP16 computations not only reduce data sizes but also accelerate the reconstruction step by approximately 10% and 30% for LpRec and FourierRec, respectively. Double memory size reduction allowed processing data for \(N = 16384\), which was not possible for 32-bit precision due to the GPU memory limit. We think that the difference between acceleration factors for the two methods (10% and 30%) is caused by the implementation of cubic B-spline interpolation procedures in the log-polar-based method, see (Andersson et al., 2016a) for details. Hard-wired linear interpolation is implemented in GPU texture memory and works with lower than 32-bit precision due to the texture reading access organization. As a result, the performance of read/write texture access with interpolation might not be significantly different for 16 and 32-bit precision.

Third, the table shows that for large data sizes the fastest TomocuPy method

| Test dataset         | 1     | 2     | 3     | 4     | 5     | 6     |
|----------------------|-------|-------|-------|-------|-------|-------|
| FourierRec, 16-bit   | \(4.2 \times 10^{-1}\)s | \(1.7 \times 10^{0}\)s | \(1.1 \times 10^{1}\)s | \(8.2 \times 10^{1}\)s | \(6.8 \times 10^{2}\)s | \(6.7 \times 10^{3}\)s |
| FourierRec, 32-bit   | \(6.3 \times 10^{-1}\)s | \(2.4 \times 10^{0}\)s | \(1.5 \times 10^{1}\)s | \(1.2 \times 10^{2}\)s | \(9.8 \times 10^{2}\)s | -     |
| LpRec, 16-bit        | \(3.1 \times 10^{-1}\)s | \(1.1 \times 10^{0}\)s | \(6.4 \times 10^{1}\)s | \(5.2 \times 10^{2}\)s | \(4.9 \times 10^{3}\)s | \(5.2 \times 10^{4}\)s |
| LpRec, 32-bit        | \(4.3 \times 10^{-1}\)s | \(1.4 \times 10^{0}\)s | \(7.3 \times 10^{1}\)s | \(5.9 \times 10^{2}\)s | \(5.5 \times 10^{3}\)s | -     |
| LineRec              | \(6.5 \times 10^{-1}\)s | \(5.5 \times 10^{0}\)s | \(8.0 \times 10^{1}\)s | \(1.3 \times 10^{2}\)s | \(2.1 \times 10^{3}\)s | -     |
| TomoPy Gridrec, 32-bit | \(3.7 \times 10^{0}\)s | \(1.5 \times 10^{1}\)s | \(1.4 \times 10^{2}\)s | \(1.7 \times 10^{3}\)s | \(1.6 \times 10^{4}\)s | \(1.7 \times 10^{5}\)s* |

* estimated by using a lower number of chunks.
(LpRec) outperforms CPU-based TomoPy implementation of Gridrec by a factor of 33 and 29 for 16 and 32-bit precision, respectively. It has been already demonstrated that GPU is more efficient than CPU for tomographic data reconstruction, see Table 1 in (Andersson et al., 2016a). However, additional time for CPU-GPU data transfers, so as read/write operations with storage drives, concealed this efficiency. With the conveyor execution proposed in this work, the benefits of using GPU became clearly visible.

Finally, it is important to note that computational complexity of algorithms is crucial in accelerating reconstruction algorithms. Computational complexity of all algorithms presented in Table 2 is $O(N^3 \log N)$, except for LineRec that has complexity $O(N^4)$. Although lineRec is also optimized and works via the conveyour execution, its reconstruction time for large data sizes is higher than the one for the CPU-based TomoPy reconstruction. We expect to see similar performance behaviour when working with other GPU-based implementations, such as ASTRA toolbox wrapper inside TomoPy (Pelt et al., 2016), where the backprojection method has computational complexity of $O(N^4)$. With such complexity, total reconstruction time for large data volumes will be mostly estimated by the GPU processing time, since time for all data transfers increases linearly with data sizes.

4. Dynamic tomography experiment with steering

To demonstrate efficacy of the TomocuPy package for processing data from a dynamic experiment with steering we considered in-situ study of gas-hydrate formation inside porous media (Nikitin et al., 2020; Nikitin et al., 2021). The gas hydrate study with X-ray is important because it allows for developing models that connect physical properties of sedimentary rocks at macro-scale to morphology and formation history of gas hydrates in the pore space at micro-scale. Further these models can be used
for developing efficient geophysical methods for natural gas-hydrate exploration. In particular, it is important to follow fast dynamic processes, such as water redistribution or gas hydrate dissociation in porous samples, since they directly affect the hydrate growth which results in changes of acoustic and electric properties of the sample.

The setup of the experiment at sector 2-BM of the Advanced Photon Source is presented in Figure 4a. The formation process was carried out in an environmental cell (Fusseis et al., 2014) packed with sand and 10% NaBr water solution. Methane gas was served under high-pressure via PEEK tubes connected from the top and bottom sides of the cell. An Oxford 800 cryostream system was used for cooling the sample by a flow of nitrogen gas of low temperature. For multi-resolution scanning we used the Optique Peter system with automatic lens and scintillator changing mechanisms. In the current setup we operated with the following pairs: 1.1x magnification – 100 µm scintillator, and 5x magnification – 50 µm scintillator. All lenses are Mitutoyo Infinity Corrected Long Working Distance Objectives, scintillator type is Crytur LuAG:Ce. The detector is Oryx 5.0 MP Mono 10GigE, 2448×2048 chip size, 3.45 µm pixel size by Teledyne FLIR LLC.

The whole sample was represented as a cylinder with height 2 cm and diameter 0.5 cm. To measure a full sample volume with low resolution (1.1x lens) we scanned it at 3 vertical regions with further data stitching procedures. 1200 projections were acquired for each scan with 0.04 s exposure time per projection, yielding 48 s per one scan. High-resolution scans with 5x lens were acquired with 1800 angles per scan and 0.15 s exposure time. The reconstruction procedure involved dark-flat field correction, ring removal, taking negative logarithm, and filtered backprojection by the log-polar-based method. Additional phase-retrieval filtering (Paganin et al., 2002) was applied for processing high-resolution data to enhance gas hydrate contrast in local tomography imaging.
For the steering demonstration, we were monitoring the gas-hydrate formation process in low spatial resolution (1.1x lens), detected regions with fast water flows happening spontaneously, and automatically zoomed-in to these regions for higher resolution (5x lens) scanning. Such automatic steering allowed us to capture the initiation and evolution of the gas-hydrate growth process inside the pore initially filled with water.

Detecting regions with fast water flows was done after reconstructing full data volumes with TomocuPy and comparing them with the ones from the previous sample state by taking element-wise difference. Reconstruction and the region of interest detection took approximately 12 s, which is much less than the total scan time (48 s). Therefore the steering engine had sufficient time to select the appropriate region for high-resolution scanning with 5x lens. Figure 4b,c shows slices through reconstructed volumes in low resolution for the sample state before and after water redistribution. In this figure, bright color corresponds to sand grains and water solution, and dark gray/black - to methane gas. The region with water outflow is marked by white arrows. Immediately after the low-resolution scan, this region was detected and scanned with high resolution (Figure 4d), where the hydrate structure formed on the water-gas interface can be observed in light gray color. The region was further continuously scanned until the final state (the end of experiment time) shown in Figure 4e.
Fig. 4. Gas-hydrate formation tomography experiment with automatic steering (zooming to a region of interest with water outflow): a) experiment setup at sector 2-BM of the Advanced Photon Source; b,c) sample states in low spatial resolution before (State 0) and after (State 1) water redistribution, respectively; d) the region of interest in high spatial resolution after water redistribution (State 1), e) the region of interest with formed gas hydrate (State 2). Bright color corresponds to sand grains and water solution, dark gray/black - to methane gas, light gray in high resolution images - to gas hydrate.

5. Conclusions and Outlook

By developing TomocuPy package we have shown that full tomography reconstruction from a standard detector (2k×2k sensor size), including all read/write operations with storage drives and initialization functions, can be done in less than 7 s on a single Nvidia Tesla A100 and NVME PCIe v4 SSD, which should result in close to real-time performance by adding multiple GPUs and SSD disks. The conveyor data processing almost completely hides the CPU-GPU data transfers time and read/write operations with storage drives are optimized for parallel operations. Additionally, switching to 16-bit floating point arithmetic decreased memory usage and processing times without significant reduction in reconstruction quality. The package is publicly available at
Performance tests showed almost linear time scaling with increasing data sizes up to $16384^3$. The linear scaling is due to efficient TomocuPy algorithms with low computational complexity ($N^3 \log N$), which becomes beneficial when working with modern detectors having large sensors. Full processing time to reconstruct a $16384^3$ volume on 1 GPU is approximately 1.5 h, and can be decreased linearly with adding GPUs because tomography slices are processed independently. For comparison, a CPU-based reconstruction with an Intel® Xeon Gold takes approximately 47 h, i.e. requires at least 33 computing nodes and a fast GPFS storage to demonstrate TomocuPy performance.

Fast 3D tomographic reconstruction with TomocuPy opens new possibilities for automatic steering in-situ experiments. As a first application, we considered a geological experiment for gas hydrate formation in porous media, where the initiation of the formation process after water redistribution was captured in high resolution inside a large sample. As the next step, we plan to conduct gas-hydrate experiments with varying the Cryostream temperature based on the sample state. It has been shown with acoustic measurements that temperature cycles affect the hydrate growth speed (Dugarov et al., 2019) and new tomography measurements may provide more details about this process. The steering mechanism could have a wide range of applications not only in geosciences but also in materials science, environmental science, and medical research. We plan to study the crack formation process inside different materials. The crack will be initiated with using a load cell while low-resolution projection data are continuously captured and reconstructed. The deformed regions of interest will be measured with high resolution to monitor the crack initiation in details. Additionally, we plan to vary pressure according to the sample state obtained from reconstructions.
TomocuPy package can be extended by adding new processing and reconstruction methods. New methods implemented with Python NumPy library are directly adapted for GPU computations by switching to Python CuPy library. Currently TomocuPy provides GPU implementations only of the one-step filtered backprojection, which is explained by the aim of having reconstructions as fast as possible. Iterative reconstruction schemes are significantly slower but they still can be added to process data more efficiently. Iterative schemes, especially the ones where slices are not reconstructed independently (e.g., 3D Total Variation regularization), can be implemented more efficiently with the conveyor GPU reconstruction and CPU-GPU data transfers. If data volumes are too large then the conveyor should also include read/write operations with the storage drive.

TomocuPy package is in routine use at the micro-CT 2-BM and nano-CT 32-ID beamlines. Because of its easy-to-use command-line interface that has almost the same commands as the one used by TomoPy, the package has quickly become friendly for beamline users. Data reconstruction for most experiments is currently done during the experiment beamtime.

Acknowledgements

The author thanks Francesco De Carlo and Anton Duchkov for their insightful comments and help with improving the paper structure, Alex Deriy and Pavel Shevchenko for help in conducting the gas-hydrate formation experiment. This research used resources of the Advanced Photon Source, a U.S. Department of Energy (DOE) Office of Science User Facility operated for the DOE Office of Science by Argonne National Laboratory under Contract No. DE-AC02-06CH11357.

References

van Aarle, W., Palenstijn, W. J., De Beenhouwer, J., Altantzis, T., Bals, S., Batenburg, K. J. & Sijbers, J. (2015). Ultramicroscopy, 157, 35–47. [URL: https://doi.org/10.1016/j.ultramic.2015.05.002]
Andersen, A. H. & Kak, A. C. (1984). Ultrasonic imaging, 6(1), 81–94.
URL: https://www.sciencedirect.com/science/article/pii/0161734684900087

Andersson, F., Carlsson, M. & Nikitin, V. V. (2016a). SIAM Journal on Imaging Sciences, 9(2), 637–664.
URL: https://doi.org/10.1137/15M1023762

Andersson, F., Carlsson, M. & Nikitin, V. V. (2016b). SIAM Journal on Imaging Sciences, 9(2), 637–664.
URL: https://doi.org/10.1137/15M1023762

Beylkin, G. (1998). Mathematical Geophysics Summer School (Stanford Univ., Stanford, 1998).
URL: https://amath.colorado.edu/faculty/beylkin/papers/applusfft.pdf

Bluestein, L. I. (1970). IEEE Transactions on Audio and Electroacoustics, 18, 451–455.

Butler, I., Fusseis, F., Cartwright-Taylor, A. & Flynn, M. (2020). Journal of synchrotron radiation, 27(6).
URL: https://doi.org/10.1107/S160057752001173X

Crowther, R. A., DeRosier, D. & Klug, A. (1970). Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences, 317(1530), 319–340.

De Andrade, V., Nikitin, V., Wojcik, M., Deriy, A., Bean, S., Shu, D., Mooney, T., Peterson, K., Kc, P., Li, K., Ali, S., Fezzaa, K., Gursoy, D., Arico, C., Ouendi, S., Trodlec, D., Simon, P., De Carlo, F. & Lethien, C. (2021). Advanced Materials, 33(21), 2008653.
URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/adma.202008653

Dowd, B., GH, C., RB, M., VV, N., SV, T., L, A. & DP, S. (1999). In Proc. SPIE, vol. 3772, pp. 224–236.

Dugarov, G. A., Duchkov, A. A., Duchkov, A. D. & Drobcik, A. N. (2019). Journal of Natural Gas Science and Engineering, 63, 38–46.

Finegan, D. P., Scheel, M., Robinson, J. B., Tjaden, B., Hunt, I., Mason, T. J., Millichamp, J., Di Michiel, M., Offer, G. J., Hinds, G. et al. (2015). Nature communications, 6(1), 1–10.
URL: https://doi.org/10.1038/ncomms7924

Fokin, M., Nikitin, V. & Duchkov, A. (2022). Journal of Computational Geosciences.
URL: https://doi.org/10.1002/adma.202014659

García-Moreno, F., Kamm, P. H., Neu, T. R., Bülk, F., Noack, M. A., Wegener, M., von der Eltz, N., Schlepütz, C. M., Stampanoni, M. & Banhart, J. (2021). Advanced Materials, 33(45), 2104659.
URL: https://doi.org/10.1002/adma.202014659

Gregor, J. & Benson, T. (2008). IEEE transactions on medical imaging, 27(7), 918–924.

Gürsoy, D., (2014). Tomopy documentation.
URL: https://tomopy.readthedocs.io/en/stable/api/tomopy.recon.algorithm.html

Gürsoy, D., De Carlo, F., Xiao, X. & Jacobsen, C. (2014). Journal of Synchrotron Radiation, 21(5), 1188–1193.
URL: https://doi.org/10.1107/S1600577514013929

Helfen, L., Myagotin, A., Rack, A., Pernot, P., Mikulik, P., Di Michiel, M. & Baumbach, T. (2007), physica status solidi (a), 204, 2760 – 2765.

Ho, N.-M. & Wong, W.-F. (2017). In 2017 IEEE High Performance Extreme Computing Conference (HPEC), pp. 1–7. IEEE.

Kim, K. Y., Zhuang, L., Yang, H., Kim, H. & Min, K.-B. (2016). Rock Mechanics and Rock Engineering, 49(4), 1201–1210.

Liu, D., Shadike, Z., Lin, R., Qian, K., Li, H., Li, K., Wang, S., Yu, Q., Liu, M., Ganapathy, S. et al. (2019). Advanced Materials, 31(28), 1806620.
URL: https://doi.org/10.1002/adma.201806620
Löfll, C., Saage, H. & Göken, M. (2019). International Journal of Fatigue, 124, 138–148.

Lowe, D. G. (1999). In Proceedings of the seventh IEEE international conference on computer vision, vol. 2, pp. 1150–1157. IEEE.

Maire, E., Le Bourlot, C., Adrien, J., Mortensen, A. & Mokso, R. (2016). International Journal of Fracture, 200(1-2), 3–12. 
URL: https://doi.org/10.1016/j.ijfract.2016.0077-y

Marone, F. & Stampanoni, M. (2012). Journal of synchrotron radiation, 19(6), 1029–1037.

Marone, F., Studer, A., Billich, H., Sala, L. & Stampanoni, M. (2017). Advanced Structural and Chemical Imaging, 3(1), 1–11.

Mohan, Aditya, K., Venkatakrishnan, S. V., Gibbs, J. W., Gulsoy, E. B., Xiao, X., De Graef, M., Voorhees, P. W. & Bouman, C. A. (2015). IEEE Transactions on Computational Imaging, 1(2), 96–111.

Mokso, R., Schleputz, C. & Stampanoni, M. (2017). J. Synchrotron radiation, 1.
URL: https://doi.org/10.1107/S1600577517013522

Münch, B., Trtik, P., Marone, F. & Stampanoni, M. (2009). Opt. Express, 17(10), 8567–8591.
URL: http://opg.optica.org/oe/abstract.cfm?URI=oe-17-10-8567

Nikitin, V. (2022). Tomocupy. 
URL: https://tomocupy.readthedocs.io/en/latest/

Nikitin, V., Tekawade, A., Duchkov, A., Shevchenko, P. & De Carlo, F. (2022). Journal of Synchrotron Radiation, 29(3).

Nikitin, V. V., Dugarov, G. A., Duchkov, A. A., Fokin, M. I., Drochik, A. N., Shevchenko, P. D., De Carlo, F. & Mokso, R. (2020). Marine and Petroleum Geology, 115, 104234.
URL: https://doi.org/10.1016/j.marpetgeo.2020.104234

Nikitin, V. V., Fokin, M. I., Dugarov, G. A., Drochik, A. N., De Andrade, V., Shevchenko, P. D., Manakov, A. Y. & Duchkov, A. A. (2021). Fuel, 298, 120699.
URL: https://doi.org/10.1016/j.fuel.2021.120699

Okuta, R., Unno, Y., Nishino, D., Hido, S. & Loomis, C. (2017). 31st Conference on Neural Information Processing Systems, 151.
URL: https://cupy.dev

Paganin, D., Mayo, S., Gureyev, T. E., Miller, P. R. & Wilkins, S. W. (2002). Journal of microscopy, 206(1), 33–40.
URL: https://doi.org/10.1046/j.1365-2818.2002.01010.x

Pelt, D. M., Gürsoy, D., Palenstijn, W. J., Sijbers, J., De Carlo, F. & Batenburg, K. J. (2016). Journal of synchrotron radiation, 23(3), 842–849.

Rivers, M., (1998). Zinger removal. https://www.mcs.anl.gov/research/projects/X-ray-cmt/rivers/tutorial.html, Last accessed on 2020-07-21.

Rivers, M. L. (2012). In Developments in X-Ray Tomography VIII, edited by S. R. Stock, vol. 8506, p. 85060U. International Society for Optics and Photonics, SPIE.
URL: https://doi.org/10.1117/12.930022

Rossetti, D. & Team, S. (2015). In GPU Technology Conference.

Scales, J. A. (1987). Geophysics, 52(2), 179–185.

Shainer, G., Ayoub, A., Lui, P., Liu, T., Kagan, M., Trott, C. R., Scantlen, G. & Crozier, P. S. (2011). Computer Science-Research and Development, 26(3), 267–273.

Singh, S. S., Williams, J. J., Hruby, P., Xiao, X., De Carlo, F. & Chawla, N. (2014). Integrating Materials and Manufacturing Innovation, 3(1), 109–122.

Tile, (2022). Tile documentation. 
URL: https://tile.readthedocs.io/en/latest/

Titarenko, S., Withers, P. J. & Yagola, A. (2010). Applied Mathematics Letters, 23(12), 1489–1495.
Synopsis

We present TomocuPy, a Python software package for fast 3D reconstruction on GPU and modern storage drives supporting parallel read-write operations. The package demonstrates significant performance gain compared to analogs and can be efficiently used when processing data during experiments requiring steering of environment conditions and acquisition schemes, or triggering data capturing processes for other measuring devices.