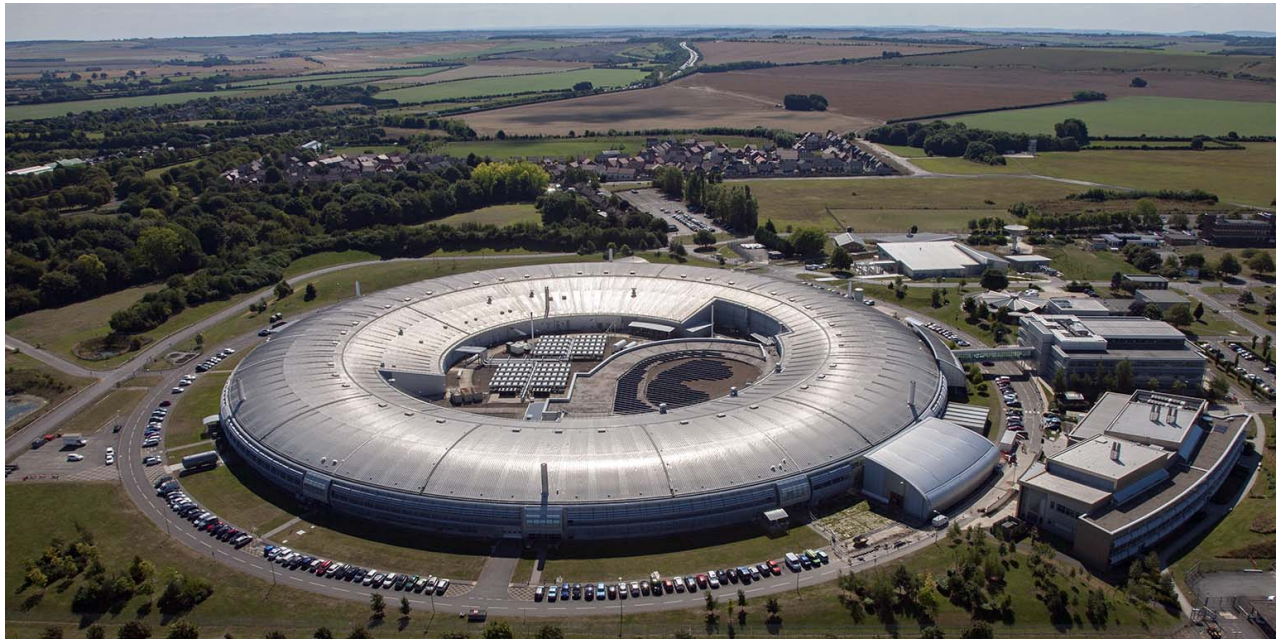
HTTomo, a new high-throughput GPU tool for large tomographic data processing at DLS

Daniil Kazantsev
Senior software scientist and tomography team lead at Diamond Light Source, Harwell, UK

September 25th, 2024

Outline

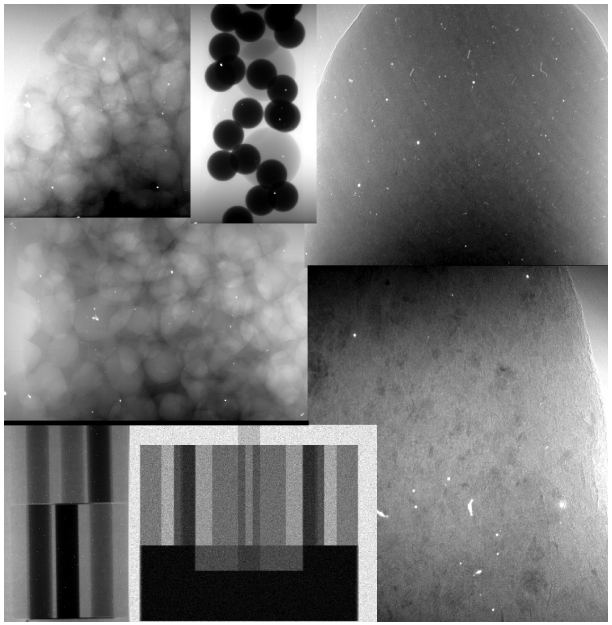
Tomographic imaging at DLS



Diamond Light Source (DLS) synchrotron, Harwell, UK

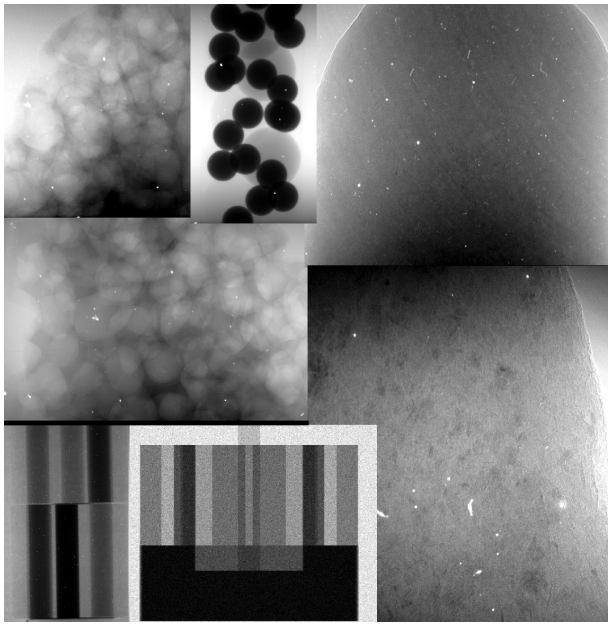
- 32 beamlines in total
- Tomographic data is collected at three dedicated beamlines: DIAD-k11, i13 and i12.

Tomographic Data collection at DLS



Tomographic projections from various beamlines.

Tomographic Data collection at DLS

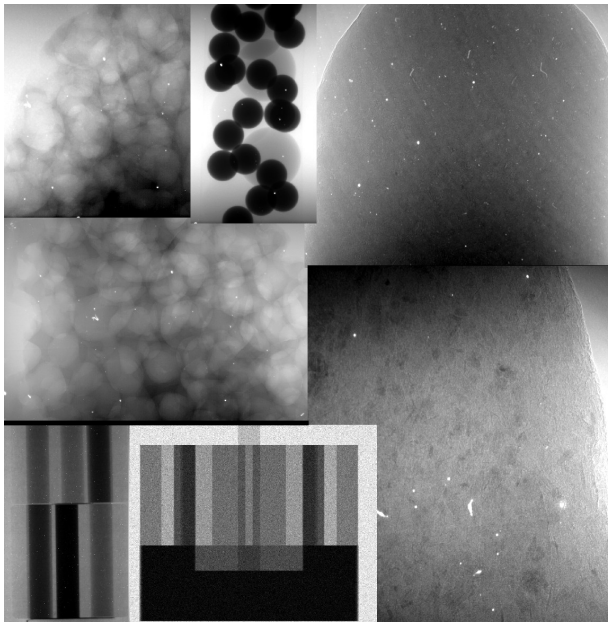


Tomographic projections from various beamlines.

Types of experiments include:

- Full-field tomography in 180 degrees

Tomographic Data collection at DLS

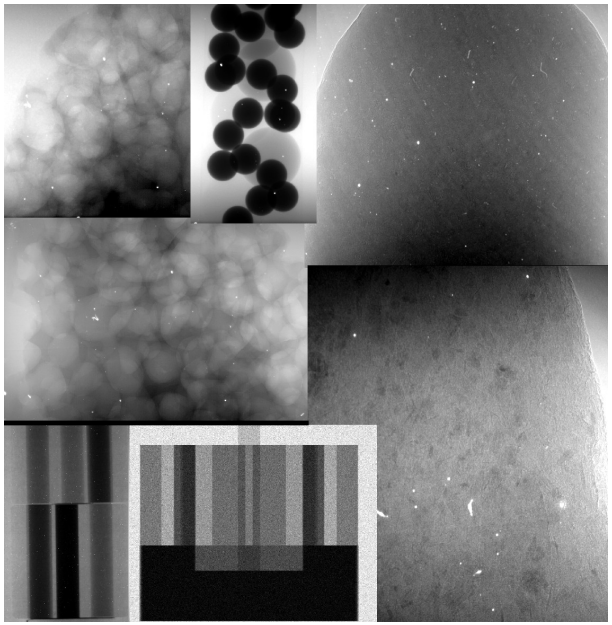


Tomographic projections from various beamlines.

Types of experiments include:

- Full-field tomography in 180 degrees
- Double FoV or 360 degrees scans

Tomographic Data collection at DLS

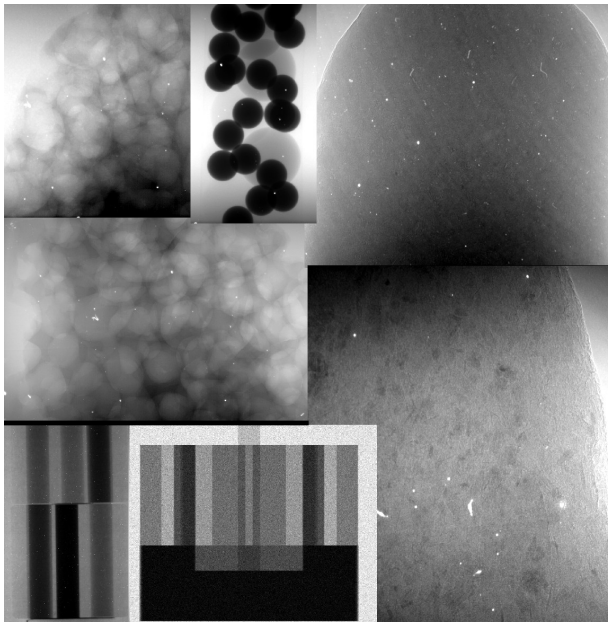


Tomographic projections from various beamlines.

Types of experiments include:

- Full-field tomography in 180 degrees
- Double FoV or 360 degrees scans
- 4D or time-lapse tomography (*In-situ* experiments under controlled conditions)

Tomographic Data collection at DLS

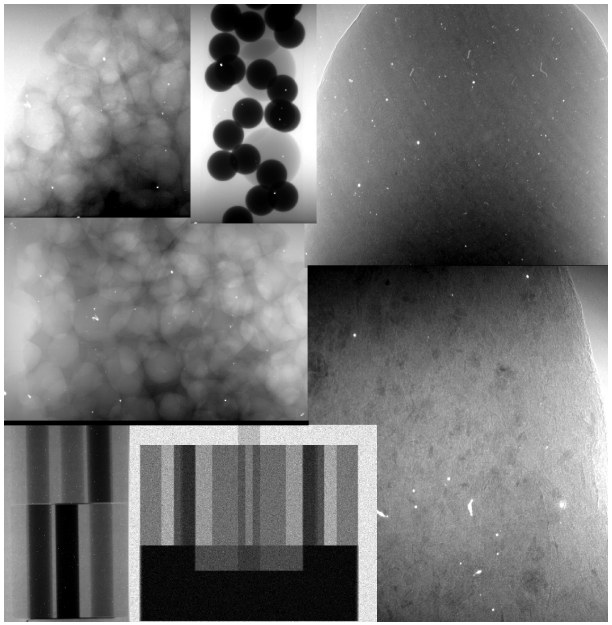


Tomographic projections from various beamlines.

Types of experiments include:

- Full-field tomography in 180 degrees
- Double FoV or 360 degrees scans
- 4D or time-lapse tomography (*In-situ* experiments under controlled conditions)
- Helical scans

Tomographic Data collection at DLS

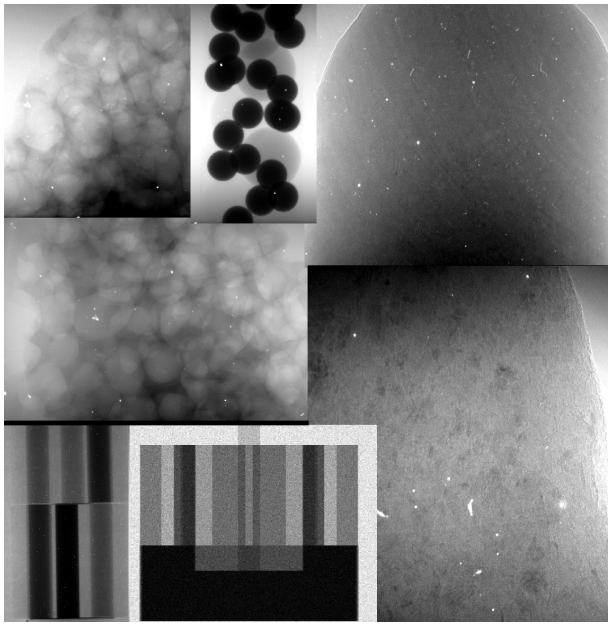


Tomographic projections from various beamlines.

Types of experiments include:

- Full-field tomography in 180 degrees
- Double FoV or 360 degrees scans
- 4D or time-lapse tomography (*In-situ* experiments under controlled conditions)
- Helical scans
- Limited angle/missing wedge, laminography

Tomographic Data collection at DLS

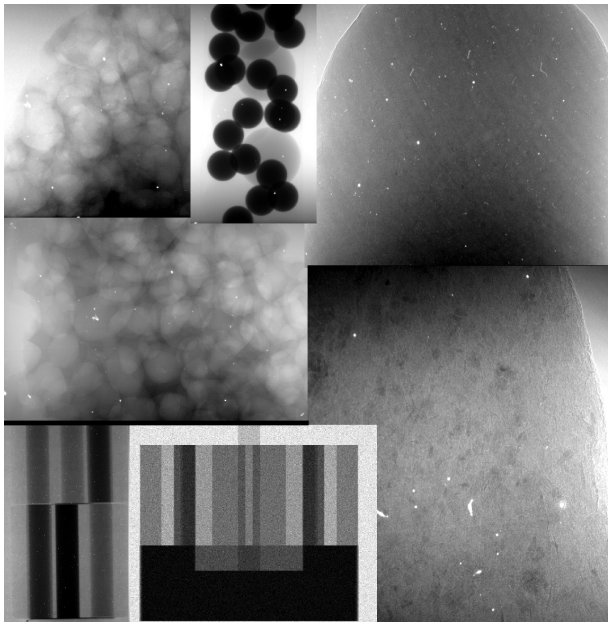


Tomographic projections from various beamlines.

Types of experiments include:

- Full-field tomography in 180 degrees
- Double FoV or 360 degrees scans
- 4D or time-lapse tomography (*In-situ* experiments under controlled conditions)
- Helical scans
- Limited angle/missing wedge, laminography
- Phase-constrast imaging, Ptycho-tomography

Tomographic Data collection at DLS

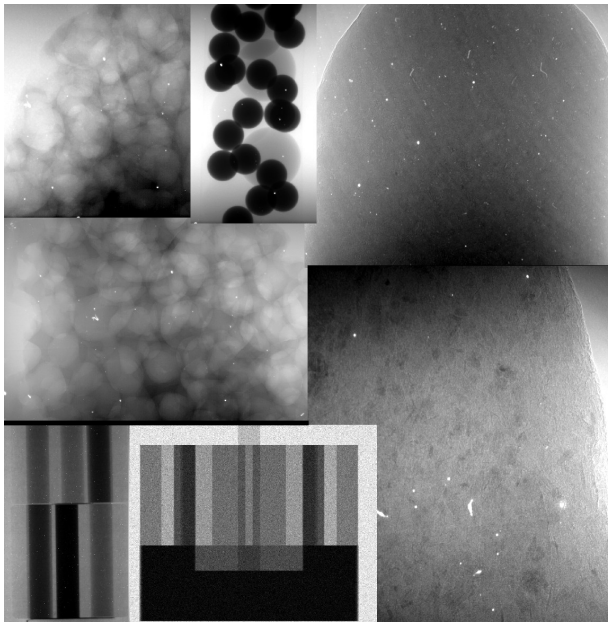


Tomographic projections from various beamlines.

Types of experiments include:

- Full-field tomography in 180 degrees
- Double FoV or 360 degrees scans
- 4D or time-lapse tomography (*In-situ* experiments under controlled conditions)
- Helical scans
- Limited angle/missing wedge, laminography
- Phase-constrast imaging, Ptycho-tomography
- Raster scans (XRF, X-ray absorption)

Tomographic Data collection at DLS



Tomographic projections from various beamlines.

Types of experiments include:

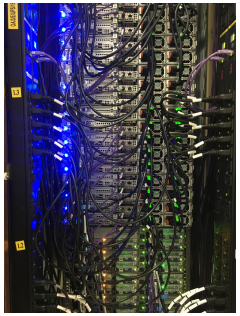
- Full-field tomography in 180 degrees
- Double FoV or 360 degrees scans
- 4D or time-lapse tomography (*In-situ* experiments under controlled conditions)
- Helical scans
- Limited angle/missing wedge, laminography
- Phase-constrast imaging, Ptycho-tomography
- Raster scans (XRF, X-ray absorption)

Data processing at DLS

Diamond's 'Wilson' computing cluster

- A typical tomographic dataset is approximately 40Gb in size (16bit data of $3600 \times 2560 \times 2180$ voxels).

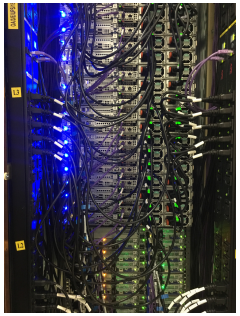
Data processing at DLS



Diamond's 'Wilson' computing cluster

- A typical tomographic dataset is approximately 40Gb in size (16bit data of 3600 $\times$ 2560 $\times$ 2180 voxels).
- 3D+time or *in-situ* imaging can require more than a hundred of scans per sample, resulting in terabytes of data.

Data processing at DLS



Diamond's 'Wilson' computing cluster

- A typical tomographic dataset is approximately 40Gb in size (16bit data of 3600 $\times$ 2560 $\times$ 2180 voxels).
- 3D+time or *in-situ* imaging can require more than a hundred of scans per sample, resulting in terabytes of data.
- Collected data is stored at General Parallel File System (GPFS).

Data processing at DLS

Diamond's 'Wilson' computing cluster

- A typical tomographic dataset is approximately 40Gb in size (16bit data of 3600 $\times$ 2560 $\times$ 2180 voxels).
- 3D+time or *in-situ* imaging can require more than a hundred of scans per sample, resulting in terabytes of data.
- Collected data is stored at General Parallel File System (GPFS).
- Data processing software can access GPFS and perform processing in parallel using multiple CPU/GPU cluster nodes.

Outline

Outline

Tomographic open-source software landscape

savu

HTTomo

python

Developed at DLS

TomoPy
Tomographic reconstruction in Python

CCPi Tomographic Imaging
C I L

ASTRA-toolbox

TIGRE

TomoSIPO

NABU

pyHST2

ToMoBAR

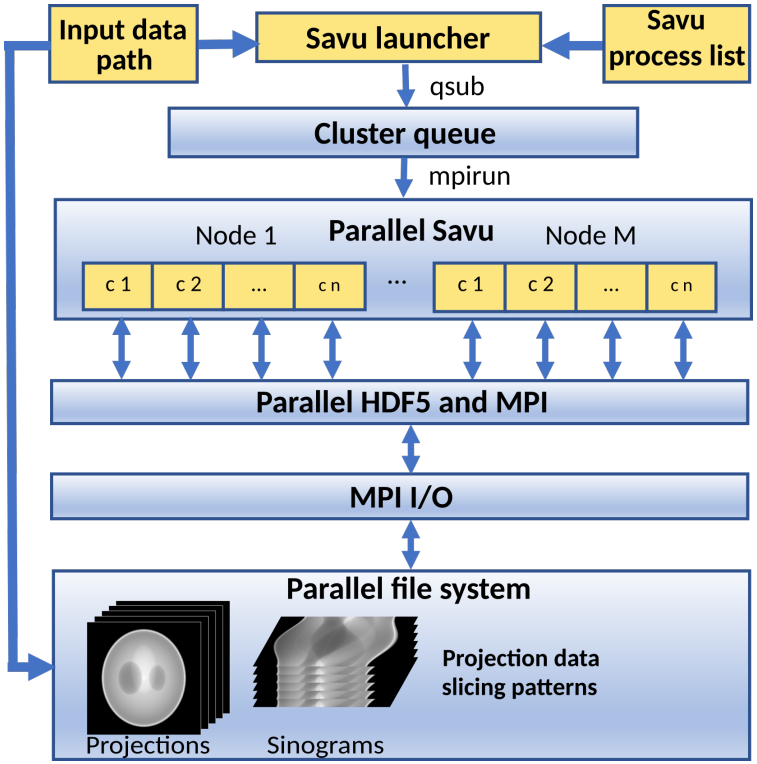
Project UFO

Tofu

https://tomopedia.github.io/

Savu - Tomographic software in production at DLS

Savu is an OOP Python package to process and reconstruct multidimensional data in serial or in parallel across a computing cluster. The project started in 2014 by Dr. Nicola Wadeson and Dr. Mark Basham.



- Implemented using MPI and parallel HDF5
- Heavy bounded I/O: all processing is in-disk. The result of each data processing step is written into an HDF5 file.
- Mostly contains integrated into the framework CPU methods.

Savu: a Python-based, MPI framework for simultaneous processing of multiple, N-dimensional, large tomography datasets by N. Wadeson, M. Basham, 2016

Outline

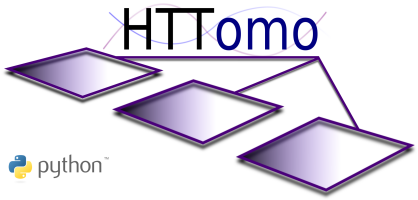
New tomographic software requirements

- High throughput and better scalability with larger data sizes.
- In-memory compute on the GPUs. Reduction of host-device data transfers.
- Methods as *plug-and-play* modules, re-use of available tomographic libraries, such as, TomoPy.
- A simple UI as an independent and replaceable layer.
- Implementation using new Python features and libraries.

New tomographic software requirements

- High throughput and better scalability with larger data sizes.
- In-memory compute on the GPUs. Reduction of host-device data transfers.
- Methods as *plug-and-play* modules, re-use of available tomographic libraries, such as, TomoPy.
- A simple UI as an independent and replaceable layer.
- Implementation using new Python features and libraries.

High-Throughput Tomography software was initiated in 2022.

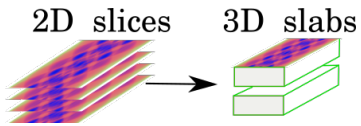


Outline

Outline

Main HTTomo concepts

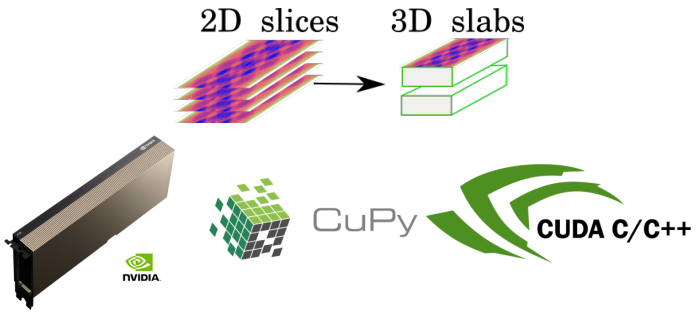
- Make I/O more efficient by working with larger 3D data chunks.



https://diamondlightsource.github.io/httomo/

Main HTTomo concepts

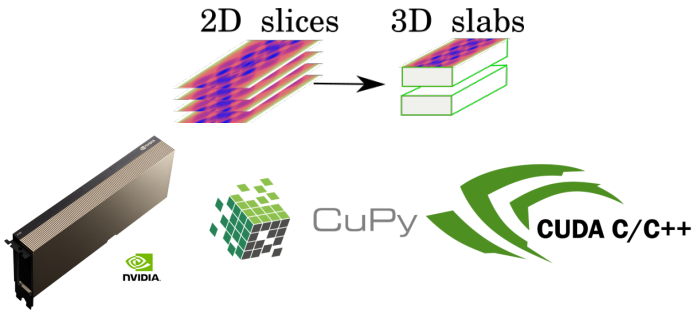
- Make I/O more efficient by working with larger 3D data chunks.
- Perform computations on the GPUs in a memory-aware fashion, i.e., avoiding CUDA OOM error.

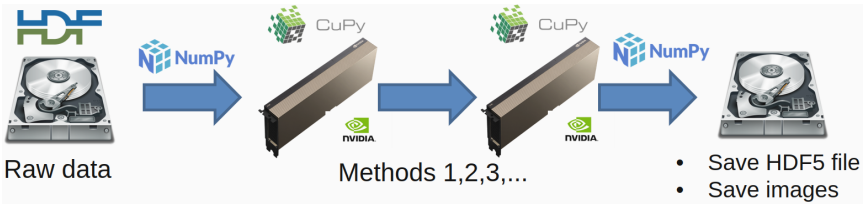


https://diamondlightsource.github.io/httomo/

Main HTTomo concepts

- Make I/O more efficient by working with larger 3D data chunks.
- Perform computations on the GPUs in a memory-aware fashion, i.e., avoiding CUDA OOM error.
- Chain methods together so that the data **remains** on the GPU device for longer.

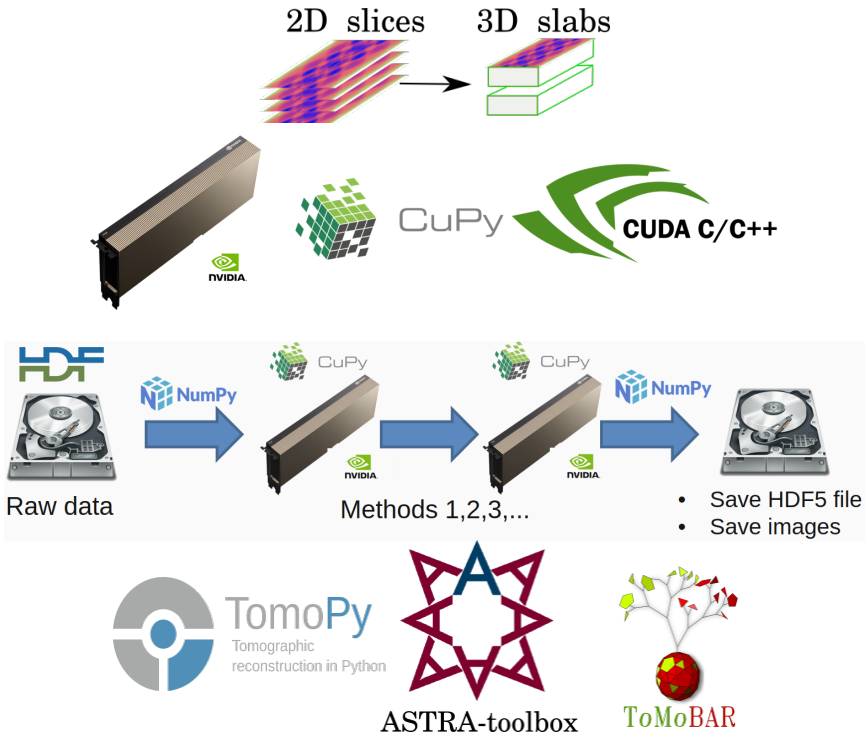




https://diamondlightsource.github.io/httomo/

Main HTTomo concepts

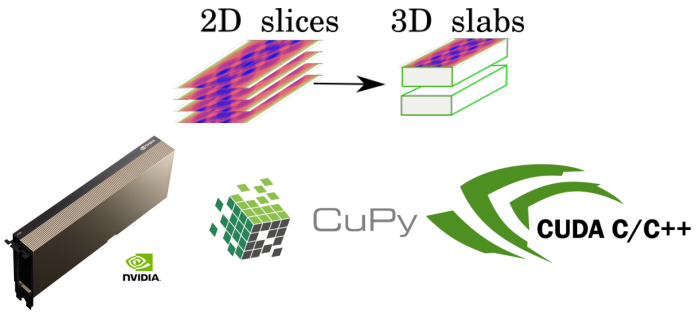
- Make I/O more efficient by working with larger 3D data chunks.
- Perform computations on the GPUs in a memory-aware fashion, i.e., avoiding CUDA OOM error.
- Chain methods together so that the data **remains** on the GPU device for longer.
- Employ other data processing libraries as **backends**.

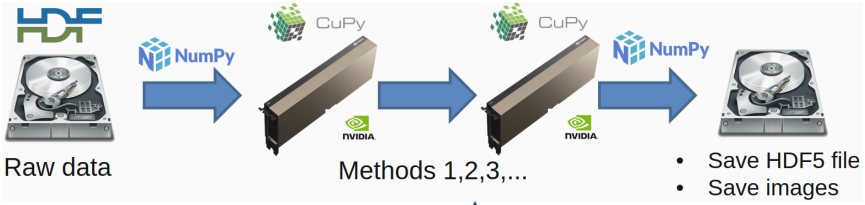


https://diamondlightsource.github.io/httomo/

Main HTTomo concepts

- Make I/O more efficient by working with larger 3D data chunks.
- Perform computations on the GPUs in a memory-aware fashion, i.e., avoiding CUDA OOM error.
- Chain methods together so that the data **remains** on the GPU device for longer.
- Employ other data processing libraries as **backends**.
- Provide human-readable UI: we use YAML

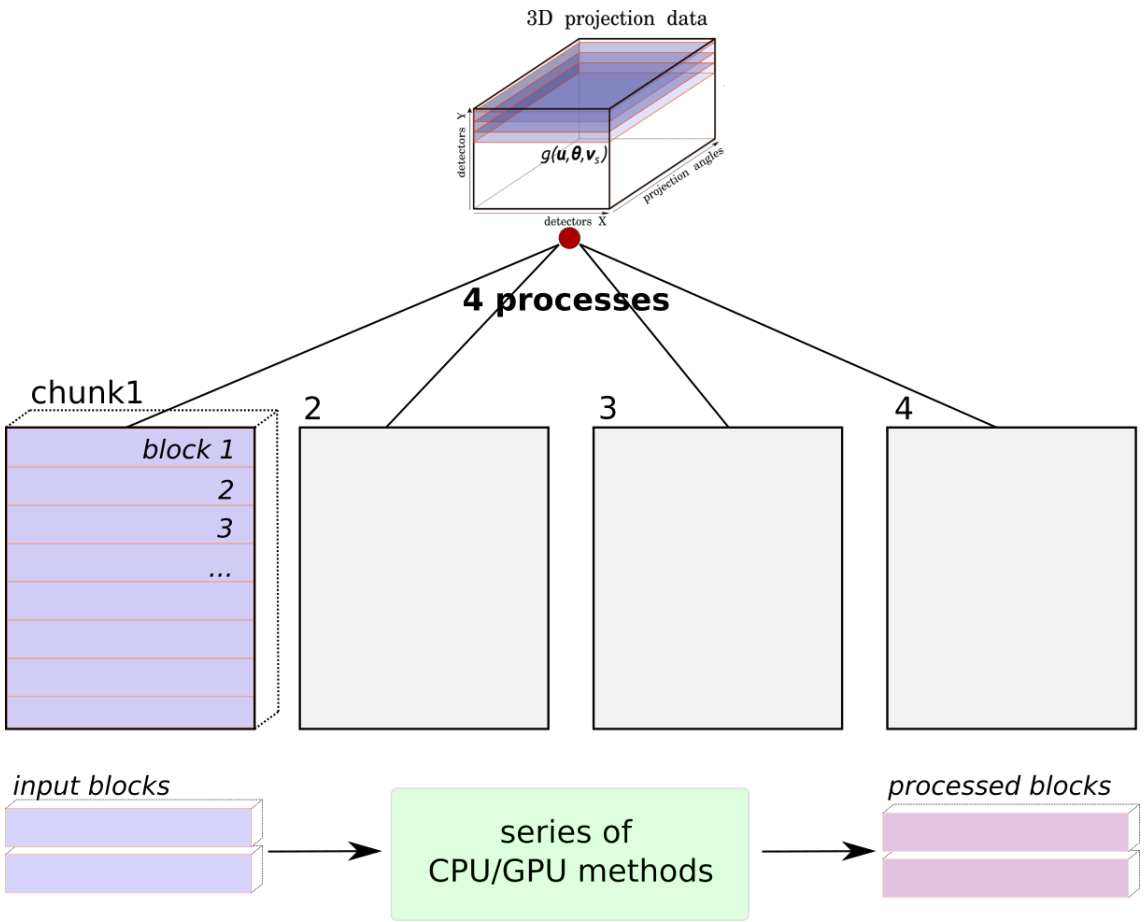





```yaml
- method: find_center_vo
  module_path: httomolibgpu.recon.rotation
  id: centering      # for identification purposes
  parameters:        # all method parameters
    ind: mid
```

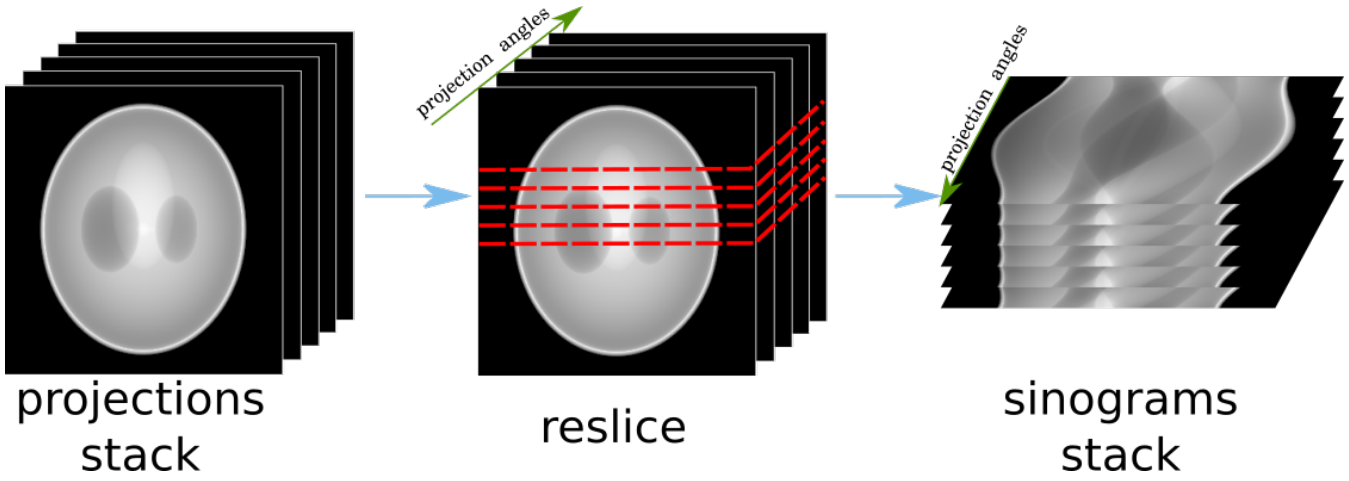
https://diamondlightsource.github.io/httomo/

HTTomo concepts: chunks and blocks



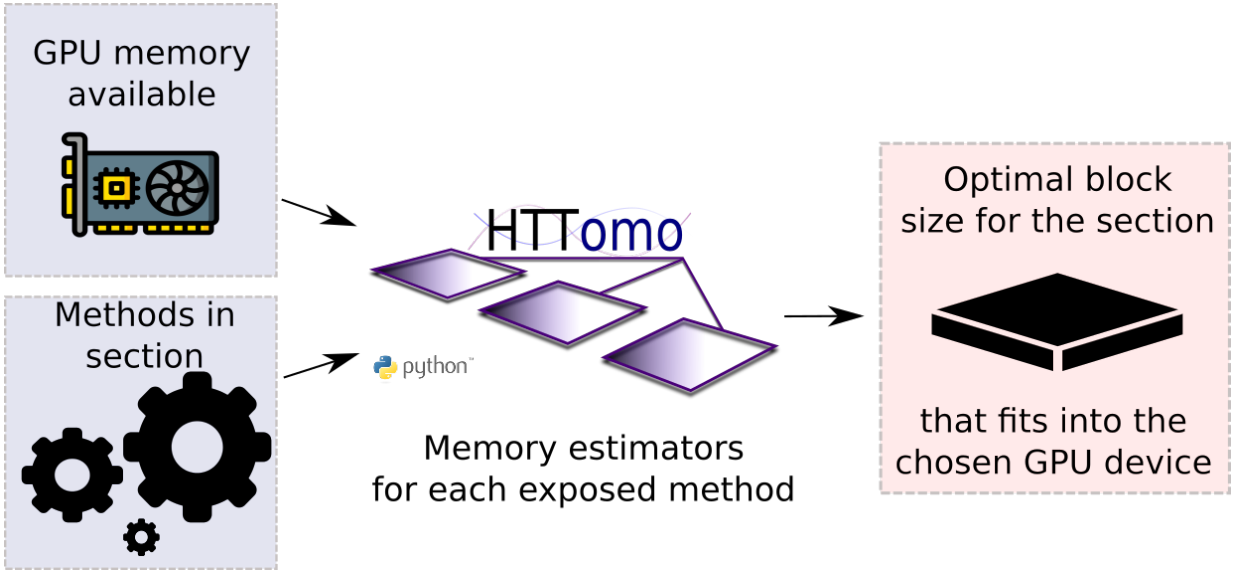
- **Chunk**: a piece of data associated with the current MPI process.
- **Block**: a smaller piece of data out of a chunk that fits into the GPU memory.

HTTomo concepts: reslice and pattern



The re-slicing of data happens when we need to access a slice which is orthogonal to the current one. It is also called a change of the **pattern**. In HTTomo we have 3 patterns in total: *projection*, *sinogram* and *all*.

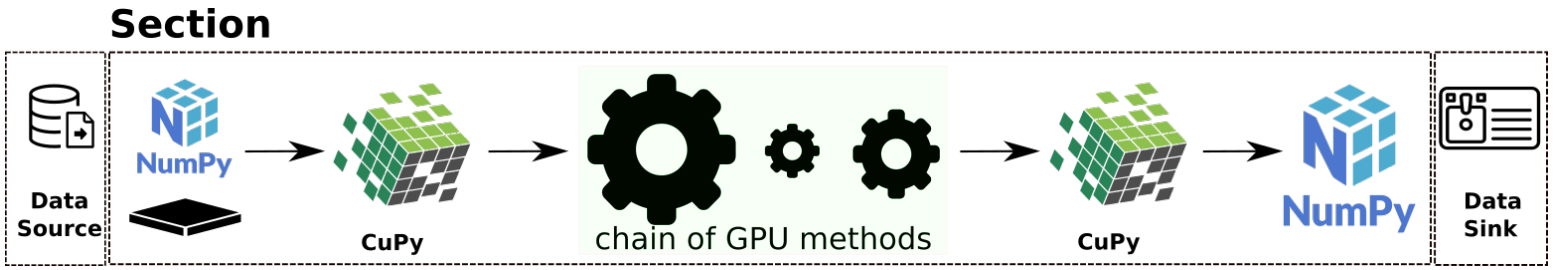
HTTomo concepts: GPU memory estimators



Memory estimators will take into account how much of the GPU memory available and how much is needed for a specific method. **They will suggest the size of the block that would fit the GPU for a specific method.**

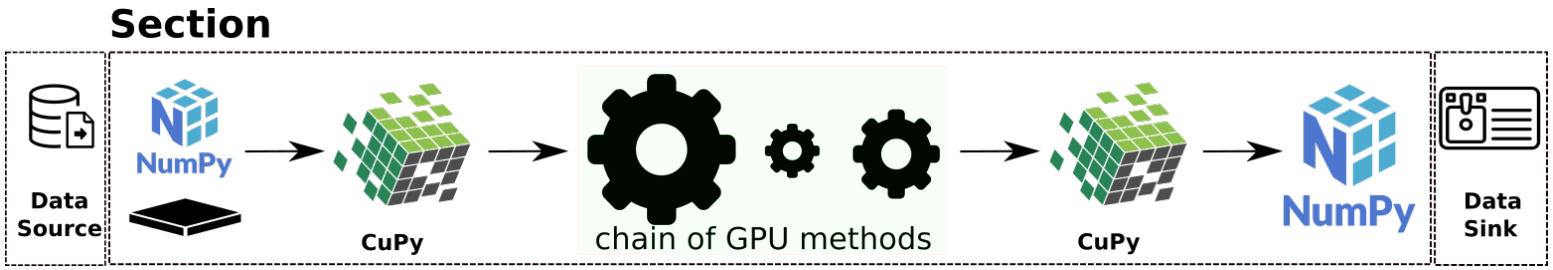
HTTomo concepts: Sections

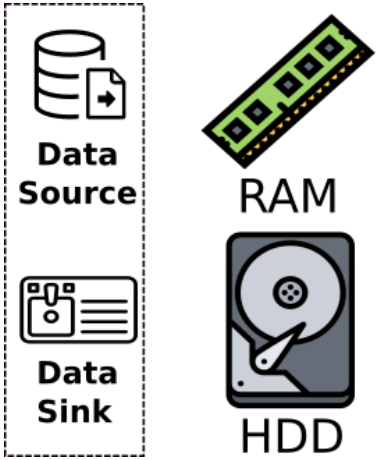
Sections will structure the pipeline execution order and ensure that the methods in that pipeline are 'chained' together, where possible.



HTTomo concepts: Sections

Sections will structure the pipeline execution order and ensure that the methods in that pipeline are 'chained' together, where possible.





Data Source and Data Sink are backed up by either CPU RAM or HDD storage. This makes HTTomo runnable even on laptops with limited RAM.

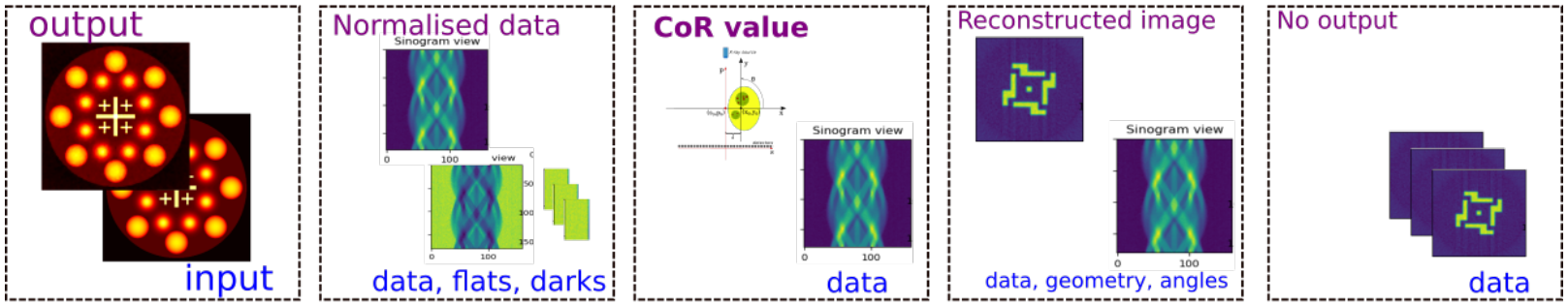
HTTomo concepts: Method wrappers

Libraries with exposed methods

TomoPy
Tomographic reconstruction in Python

HTTomoLib
HTTomoLibGPU

HTTomo wrappers



Generic | Normalisation | Centering | Reconstruction | Image Saver

HTTomo concepts: YAML UI

```yaml
- method: standard_tomo
  module_path: httomo.data.hdf.loaders
  parameters:
    name: tomo
    data_path: entry1/tomo_entry/data/data
    image_key_path: entry1/tomo_entry/instrument/detector/image_key
    rotation_angles:
      data_path: /entry1/tomo_entry/data/rotation_angle
- method: find_center_pc
  module_path: tomopy.recon.rotation
  parameters:
    proj1: auto
    proj2: auto
    tol: 0.5
    rotc_guess: null
  id: centering
  side_outputs:
    cor: centre_of_rotation
- method: normalize
  module_path: tomopy.prep.normalize
  parameters:
    cutoff: null
    averaging: mean
- method: minus_log
  module_path: tomopy.prep.normalize
  parameters: {}
- method: recon
  module_path: tomopy.recon.algorithm
  parameters:
    center: ${{centering.side_outputs.centre_of_rotation}}
    sinogram_order: false
    algorithm: 'gridrec'
    init_recon: null
```

- Backend methods are exposed through YAML templates.

HTTomo concepts: YAML UI

```yaml
- method: standard_tomo
  module_path: httomo.data.hdf.loaders
  parameters:
    name: tomo
    data_path: entry1/tomo_entry/data/data
    image_key_path: entry1/tomo_entry/instrument/detector/image_key
    rotation_angles:
      data_path: /entry1/tomo_entry/data/rotation_angle
- method: find_center_pc
  module_path: tomopy.recon.rotation
  parameters:
    proj1: auto
    proj2: auto
    tol: 0.5
    rotc_guess: null
  id: centering
  side_outputs:
    cor: centre_of_rotation
- method: normalize
  module_path: tomopy.prep.normalize
  parameters:
    cutoff: null
    averaging: mean
- method: minus_log
  module_path: tomopy.prep.normalize
  parameters: {}
- method: recon
  module_path: tomopy.recon.algorithm
  parameters:
    center: ${{centering.side_outputs.centre_of_rotation}}
    sinogram_order: false
    algorithm: 'gridrec'
    init_recon: null
```

- Backend methods are exposed through YAML templates.
- The YAML templates are automatically generated and uploaded to documentation page.

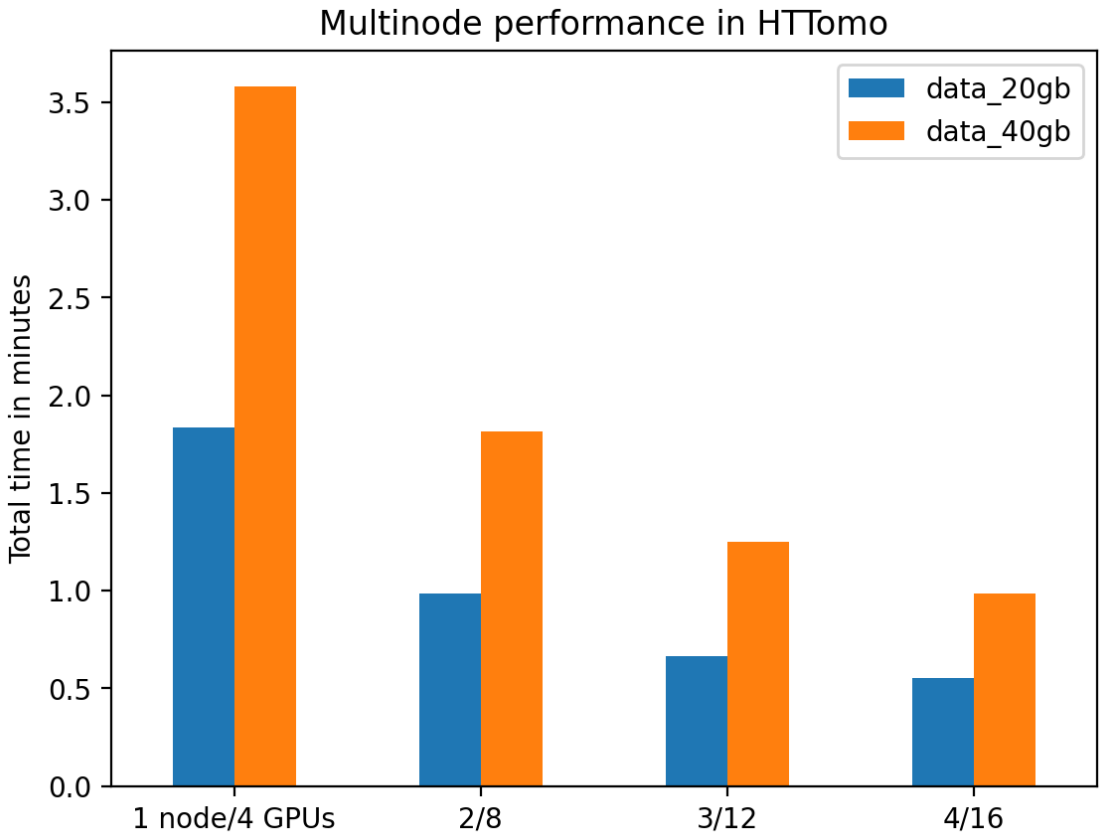
HTTomo concepts: YAML UI

```yaml
- method: standard_tomo
  module_path: httomo.data.hdf.loaders
  parameters:
    name: tomo
    data_path: entry1/tomo_entry/data/data
    image_key_path: entry1/tomo_entry/instrument/detector/image_key
    rotation_angles:
      data_path: /entry1/tomo_entry/data/rotation_angle
- method: find_center_pc
  module_path: tomopy.recon.rotation
  parameters:
    proj1: auto
    proj2: auto
    tol: 0.5
    rotc_guess: null
  id: centering
  side_outputs:
    cor: centre_of_rotation
- method: normalize
  module_path: tomopy.prep.normalize
  parameters:
    cutoff: null
    averaging: mean
- method: minus_log
  module_path: tomopy.prep.normalize
  parameters: {}
- method: recon
  module_path: tomopy.recon.algorithm
  parameters:
    center: ${{centering.side_outputs.centre_of_rotation}}
    sinogram_order: false
    algorithm: 'gridrec'
    init_recon: null
```

- Backend methods are exposed through YAML templates.
- The YAML templates are automatically generated and uploaded to documentation page.
- The YAML checker would check the validity of the YAML pipeline. It automatically executed before the run.

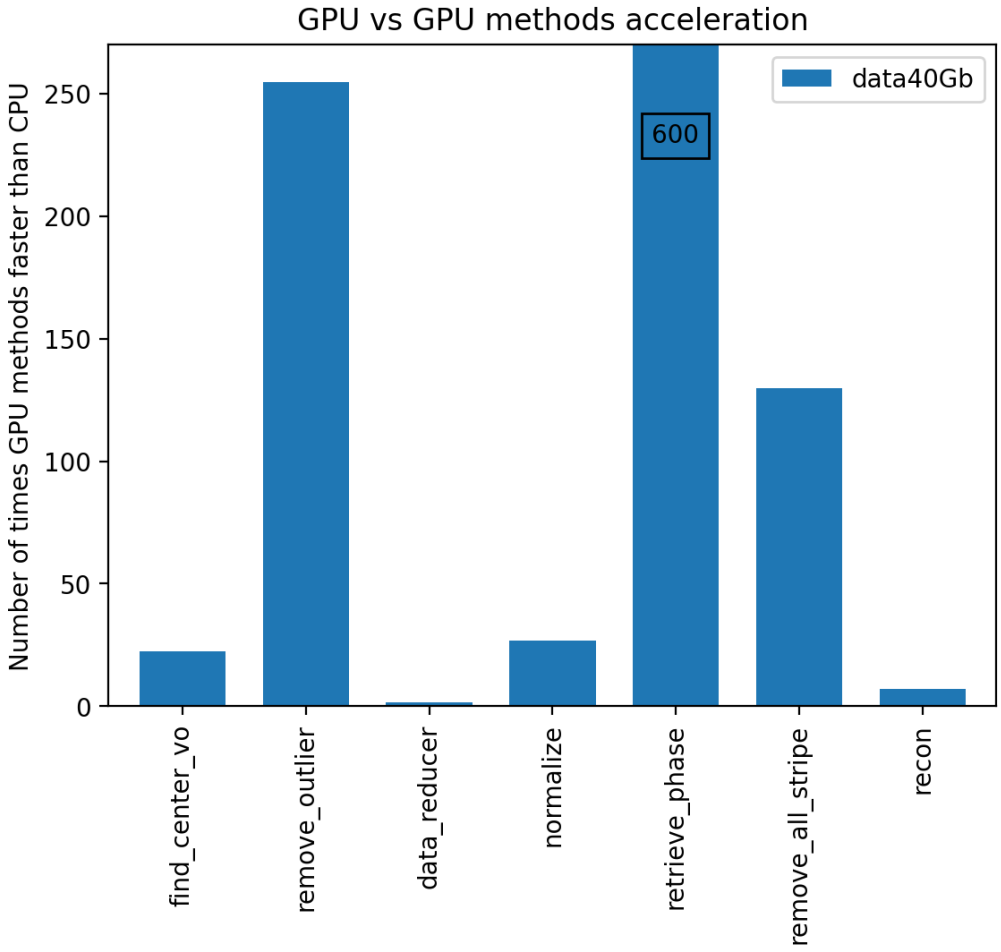
Outline

Multinode performance for HTTomo



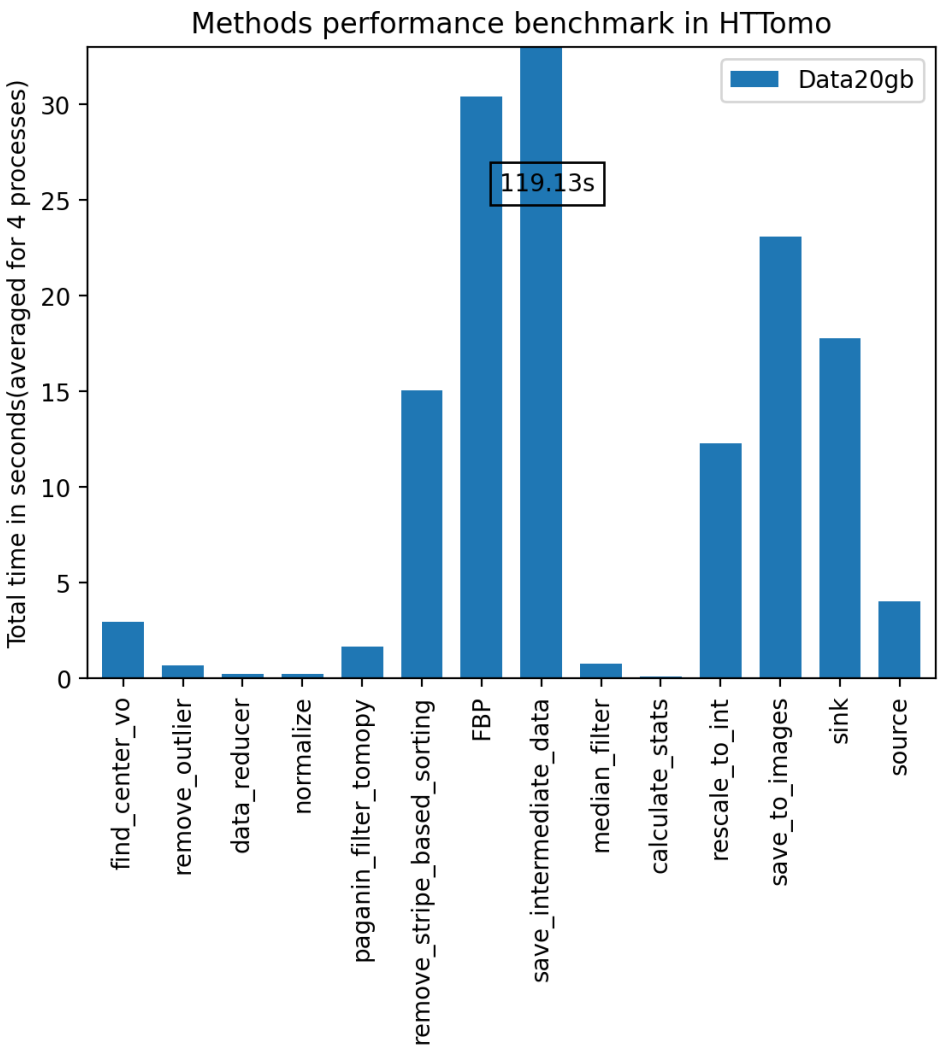
The pipeline includes the following methods: **a loader, d/f normaliser, CoR estimator, Paganin filter, Stripe Removal, Reconstruction (FBP) and 3D median filter**.

CPU vs GPU methods performance in HTTomo



Some method were taken from the TomoPy library, re-written using CuPy and CUDA and integrated into HTTomoLibGPU library

Time contribution for different methods



The most time consuming methods: saving the result of the reconstruction (56Gb) HDF5-chunked, FBP and saving images as tiffs (2100 of them).

Outline

Future directions for HTTomo

- Become a production software for tomography processing at DLS by the end of this year.

Future directions for HTTomo

- Become a production software for tomography processing at DLS by the end of this year.
- Continue with integration of new features and algorithms as requested by beamlines.

Future directions for HTTomo

- Become a production software for tomography processing at DLS by the end of this year.
- Continue with integration of new features and algorithms as requested by beamlines.
- Further optimisation of performance, faster HDF5/zarr data saving.

Future directions for HTTomo

- Become a production software for tomography processing at DLS by the end of this year.
- Continue with integration of new features and algorithms as requested by beamlines.
- Further optimisation of performance, faster HDF5/zarr data saving.
- Consider processing in 16 bit to reduce CPU/GPU memory footprint and processing time.

Big thanks to all HTTomo contributors:

Yousef Moazzam, Jessica Verschoyle, Naman Gera, Jacob Williamson, Garry O'Donnell, and Jorg Lotze (Xcelerit).

Thank you for your attention

Questions