An Extensible Parallel Computing Framework for Ultra-Fast X-Ray Imaging

Matthias Vogelgesang

matthias.vogelgesang@kit.edu

Institute for Data Processing and Electronics

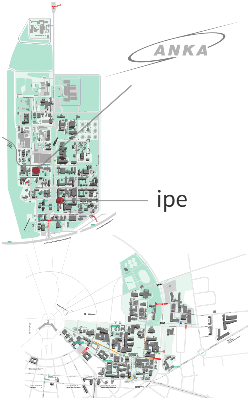
Institute for Data Processing and Electronics

Hardware

- Development (FPGA, ASIC)
- Manufacturing (circuit production, bonding)
- Characterization and long-term tests

Software

- Experiment control and data acquisition
- Analysis of acquired data
- Large scale data storage



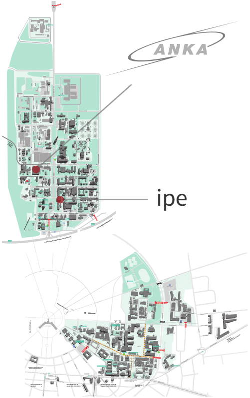
Institute for Data Processing and Electronics

Hardware

- Development (FPGA, ASIC)
- Manufacturing (circuit production, bonding)
- Characterization and long-term tests

Software

- Experiment control and data acquisition
- Analysis of acquired data
- Large scale data storage



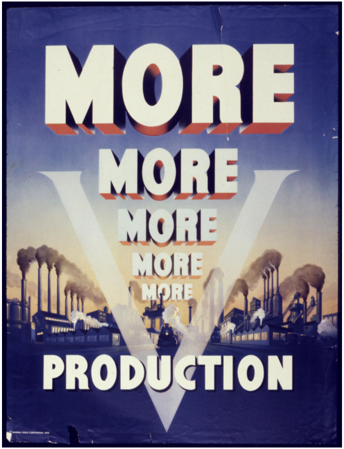
Data analysis for synchrotron μCT

Demanding requirements

- Compute-intensive reconstruction
- Variety of pre- and post-processing steps
- Fast and direct feedback

More data

- Better sensors
- Higher throughput
- Time-resolved scans



Data analysis for synchrotron μCT

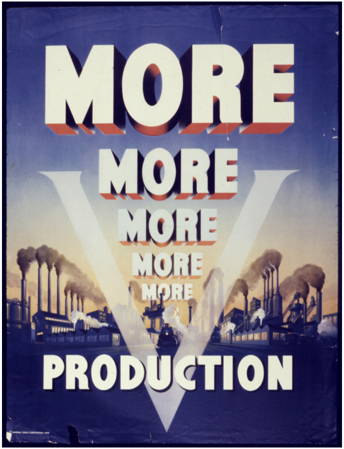
Demanding requirements

- Compute-intensive reconstruction
- Variety of pre- and post-processing steps
- Fast and direct feedback

More data

- Better sensors
- Higher throughput
- Time-resolved scans

Existing tools can hardly satisfy the demands!



Fast image processing on GPUs

ufo framework

- Streamed data processing using heterogeneous compute resources
- Pipelined and parallelized on multiple levels
- Suited for high-volume image processing (e.g. tomography)

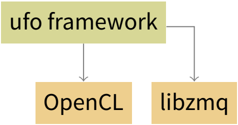
This talk

- Framework does *not* provide any functionality on its own
- Domain-specific tools and applications have to be developed
- This will be a quick tour what is possible and how it is done

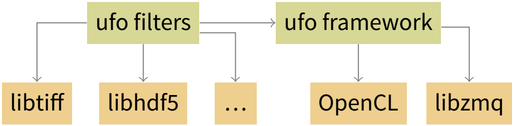
Recap

- Core framework written in C and OpenCL
- Large suite of pre-defined filters for high-throughput image processing
- User specifies workflow, framework takes care of the rest
- Open source (LGPL) and hosted at GitHub `github.com/ufo-kit`

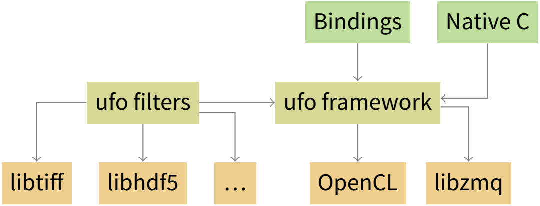
Components



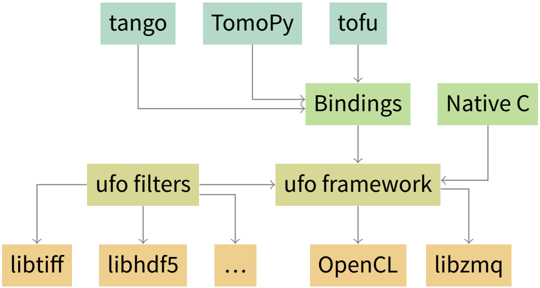
Components



Components

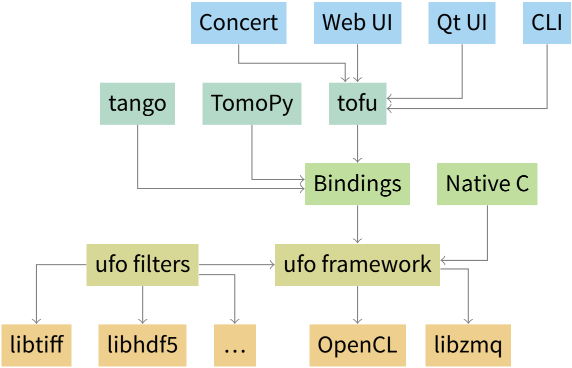

Bindings
Native C
ufo filters
ufo framework
libtiff
libhdf5
...
OpenCL
libzmq

Components

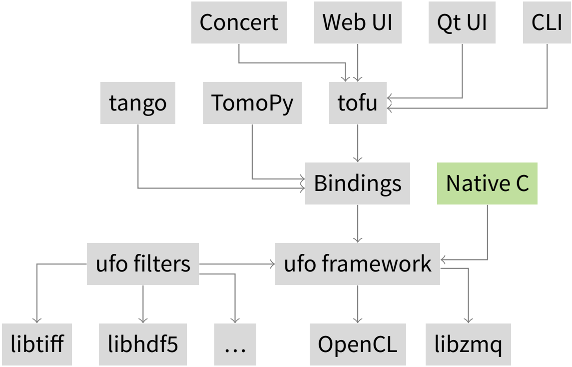


tango
TomoPy
tofu
Bindings
Native C
ufo filters
ufo framework
libtiff
libhdf5
…
OpenCL
libzmq

Components



Concert
Web UI
Qt UI
CLI
tango
TomoPy
tofu
Bindings
Native C
ufo filters
ufo framework
libtiff
libhdf5
...
OpenCL
libzmq


Concert
Web UI
Qt UI
CLI
tango
TomoPy
tofu
Bindings
Native C
ufo filters
ufo framework
libtiff
libhdf5
...
OpenCL
libzmq

Builtin tools

Written in native C

- Tools written directly in C
- General purpose: `ufo-launch` and `ufo-runjson`
- (Domain-specific: laminographic reconstructor)

Builtin tools

Written in native C

- Tools written directly in C
- General purpose: `ufo-launch` and `ufo-runjson`
- (Domain-specific: laminographic reconstructor)

Launching linear pipelines

- Used for basic one-off jobs and specified on the command line
- Tasks separated by exclamation marks
- Parameterized with key-value property assignments

Launch examples

Read and write data

```
ufo-launch read path=folder/sino*.tif !
    write filename=multi.tif
```

Launch examples

Read and write data

```
ufo-launch read path=multi.tif !
    write filename=folder/single-%05i.tif
```

Launch examples

Read and write data

```
ufo-launch read path=folder/sino*.tif !
    write filename=output.h5:/raw
```

Launch examples

Downscale input data

```
ufo-launch read path=folder/sino*.tif !
    rescale factor=0.5 !
    write filename=output.h5:/raw
```

Launch examples

Apply OpenCL expressions

```
ufo-launch read path=folder/sino*.tif !
    rescale factor=0.5 !
    calculate expression="log(v)" !
    write filename=output.h5:/raw
```

Launch examples

Remove vertical stripes

```
ufo-launch read path=folder/sino*.tif !
    rescale factor=0.5 !
    calculate expression="log(v)" !
    fft dimensions=2 ! filter-stripes ! ifft dimensions=2 !
    write filename=output.h5:/raw
```

Launch examples

Compute filtered backprojection

```
ufo-launch read path=folder/sino*.tif !
    rescale factor=0.5 !
    calculate expression="log(v)" !
    fft dimensions=2 ! filter-stripes ! ifft dimensions=2 !
    fft ! filter ! ifft ! backproject !
    write filename=output.h5:/entry/data/data
```

JSON representation

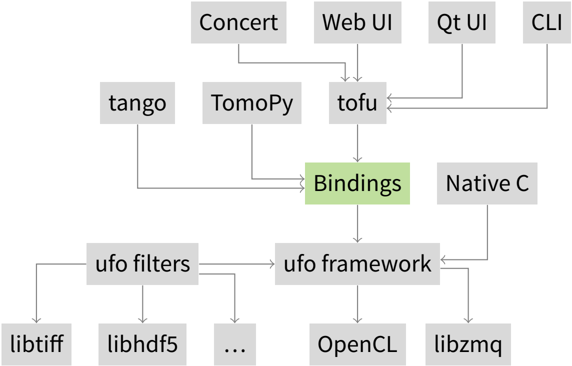
- `ufo-launch` can only execute linear pipelines
- Complex relationships must be expressed programmatically or with a data structure
- We use a simple JSON format to serialize the data structure
- This structure can then be run via

```
$ ufo-runjson dataflow.json
```

JSON example

Karlsruhe Institute of Technology


```
{
  "nodes": [
    {"plugin": "read", "name": "read",
     "properties": {"path": "folder/sino*.tif"}},
    {"plugin": "rescale", "name": "rescale",
     "properties": {"factor": 0.5}},
    {"plugin": "write", "name": "write",
     "properties": {"filename": "output.h5:/raw"}}
  ],
  "edges": [
    {"from": "read", "to": "rescale", "input": 0},
    {"from": "rescale", "to": "write", "input": 0}
  ]
}
```


Concert
Web UI
Qt UI
CLI
tango
TomoPy
tofu
Bindings
Native C
ufo filters
ufo framework
libtiff
libhdf5
...
OpenCL
libzmq

Language bindings

- JSON is a good format to freeze a data flow
- Further customization requires writing C code or bind to a scripting language
- Introspection mechanism allows for third-party language support
- Including JavaScript, Python, Ruby, Lua, Go, Haskell …

Language bindings

- JSON is a good format to freeze a data flow
- Further customization requires writing C code or bind to a scripting language
- Introspection mechanism allows for third-party language support
- Including JavaScript, Python, Ruby, Lua, Go, Haskell …
- However, our primary target for now is Python

Binding interface

```python
# "ufo-runjson" in five lines

import sys
from gi.repository import Ufo

pm = Ufo.PluginManager()
g = Ufo.TaskGraph.read_from_file(pm, sys.argv[1])

sched = Ufo.Scheduler()
sched.run(g)
```

Binding interface

```python
from gi.repository import Ufo

pm = Ufo.PluginManager()
read = pm.get_task('read')
rescale = pm.get_task('rescale')
write = pm.get_task('write')

read.set_properties(path='folder/sino*.tif')
rescale.set_properties(factor=0.5)
write.set_properties(filename='output.h5:/raw')

g = Ufo.TaskGraph()
g.connect_nodes(read, rescale)
g.connect_nodes(rescale, write)

sched = Ufo.Scheduler()
sched.run(g)
```

Improved Python integration

Global Interpreter Lock

- GIL would block Python interpreter during computation
- GIL is released during execution and insertion of data

Interfacing with NumPy

- C module converts between ufo and NumPy
- Alternatively data pointers can be re-used

High-level abstractions

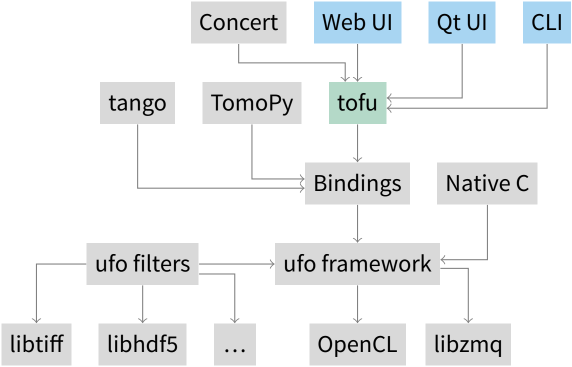
- `ufo` module wraps filters during import
- More magic but cleaner instantation and setup

High-level Python

```python
from ufo import Read, Write, Rescale

read = Read(path='folder/sino*.tif')
rescale = Rescale(factor=0.5)
write = Write(filename='output.h5:/raw')

# wait for execution to finish
write(rescale(read())).run().wait()
```

High-level Python

```python
from ufo import Read, Rescale

read = Read(path='folder/sino*.tif')
rescale = Rescale(factor=0.5)

# use result immediately
for image in rescale(read()):
    print(np.mean(image))
```


Concert
Web UI
Qt UI
CLI
tango
TomoPy
tofu
Bindings
Native C
ufo filters
ufo framework
libtiff
libhdf5
…
OpenCL
libzmq

TOFU reconstruction toolkit

Idea

- Move reconstruction-related code to single Python module
- Simplifies setup and execution of reconstruction pipelines using ufo
- Visualization widgets based on PyQtGraph

Focus

- Tomographic reconstruction with FBP, DFI and SART
- Laminographic reconstruction with FBP
- Manual and automatic axis alignment

Command line interface

Karlsruhe Institute of Technology

- Offline reconstruction for power users
- Parameters are stored in a configuration

```
$ ufo-reconstruct init
$ vi reco.conf
$ ufo-reconstruct tomo
```

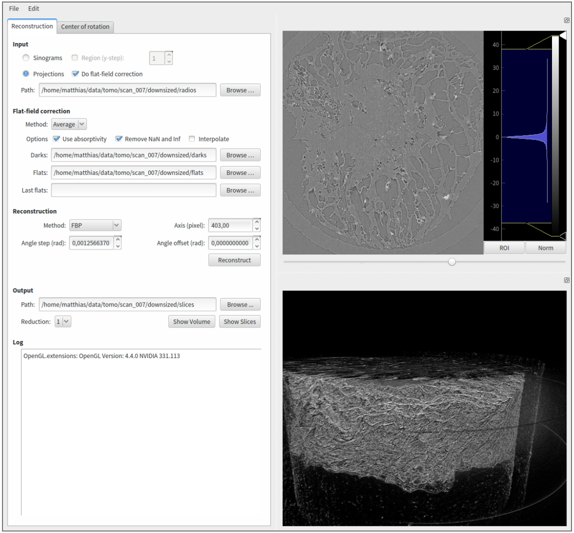
- …which can be overriden with command line arguments

```
$ ufo-reconstruct run --axis=234.5
```

Qt GUI

- Offline reconstruction for regular users
- Shares configuration with command line version
- Uses PyQt and PyQtGraph widgets for visualization

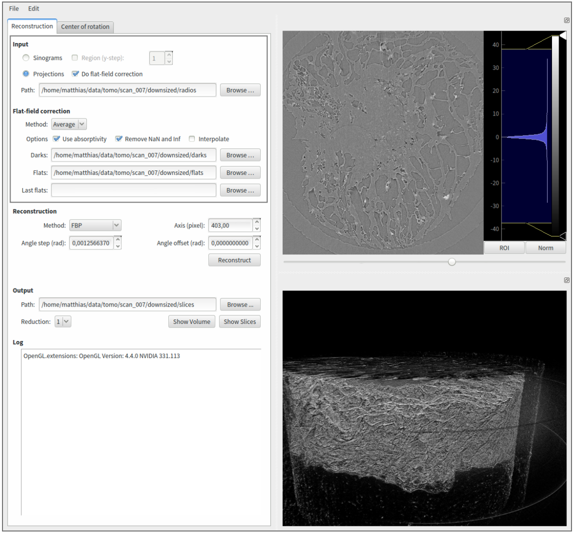
Qt GUI


File
Edit
Reconstruction
Center of rotation
Input
Sinograms
Region (y-step):
Projections
Do flat-field correction
Path: /home/matthias/data/tomo/scan_007/downsized/radios
Browse ...
Flat-field correction
Method: Average
Options
Use absorptivity
Remove NaN and Inf
Interpolate
Darks: /home/matthias/data/tomo/scan_007/downsized/darks
Flats: /home/matthias/data/tomo/scan_007/downsized/flats
Last flats:
Reconstruction
Method: FBP
Axis (pixel): 403,00
Angle step (rad): 0,0012566370
Angle offset (rad): 0,0000000000
Reconstruct
Output
Path: /home/matthias/data/tomo/scan_007/downsized/slices
Reduction: 1
Show Volume
Show Slices
Log
OpenGL.extensions: OpenGL Version: 4.4.0 NVIDIA 331.113
ROI
Norm

Qt GUI

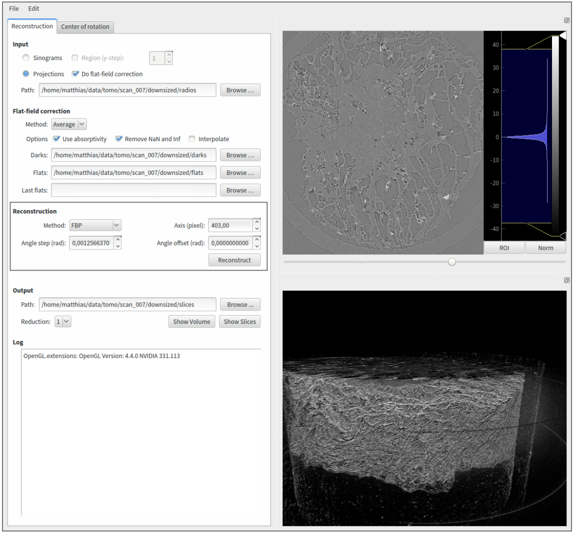
Karlsruhe Institute of Technology

Input



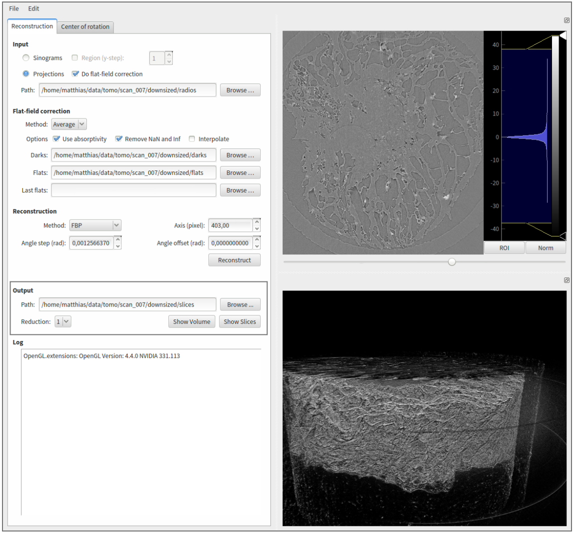
Qt GUI

Parameters



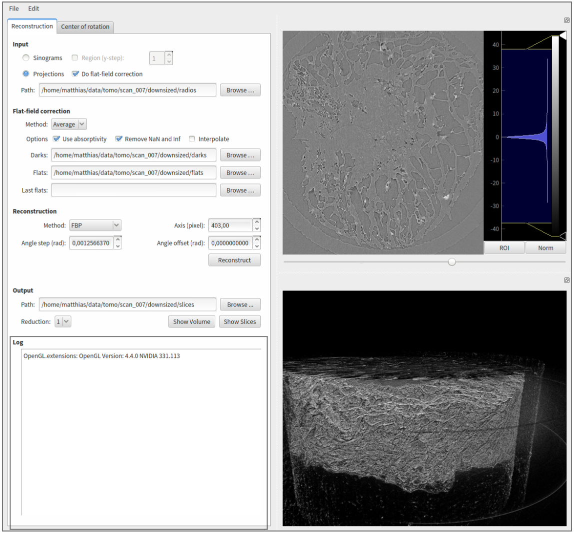
Qt GUI

Output

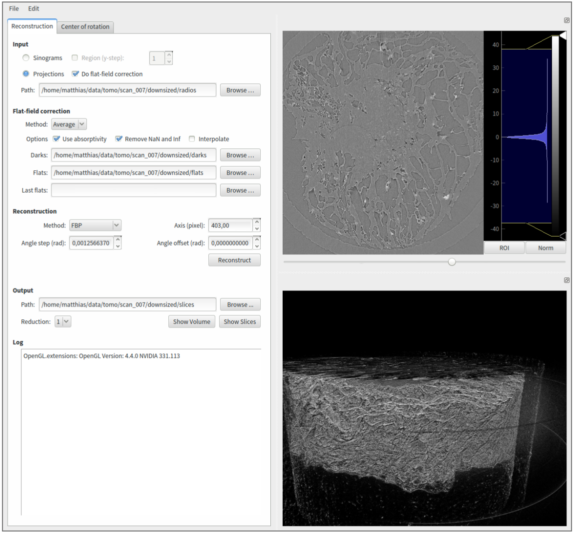


Qt GUI

Log

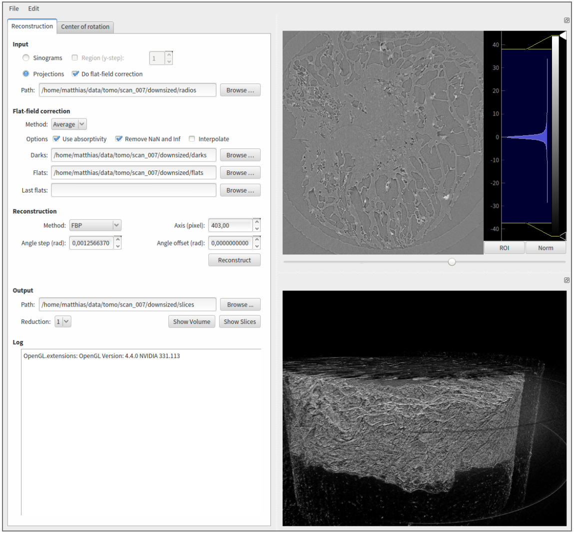

File Edit
Reconstruction
Center of rotation
Input
Sinograms
Region (y-step):
Projections
Do flat-field correction
Path: /home/matthias/data/tomo/scan_007/downsized/radios
Browse ...
Flat-field correction
Method: Average
Options Use absorptivity Remove NaN and Inf Interpolate
Darks: /home/matthias/data/tomo/scan_007/downsized/darks
Flats: /home/matthias/data/tomo/scan_007/downsized/flats
Last flats:
Reconstruction
Method: FBP
Axis (pixel): 403,00
Angle step (rad): 0,0012566370
Angle offset (rad): 0,0000000000
Reconstruct
Output
Path: /home/matthias/data/tomo/scan_007/downsized/slices
Reduction: 1
Show Volume
Show Slices
Log
OpenGL.extensions: OpenGL Version: 4.4.0 NVIDIA 331.113
ROI
Norm

Qt GUI



Slices

Qt GUI

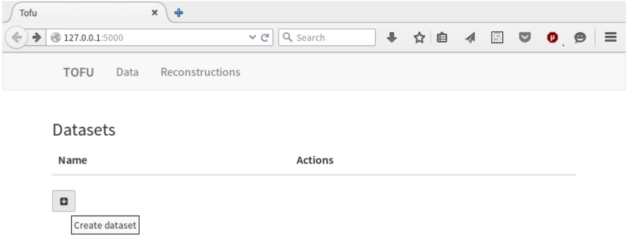


Volume

Web UI

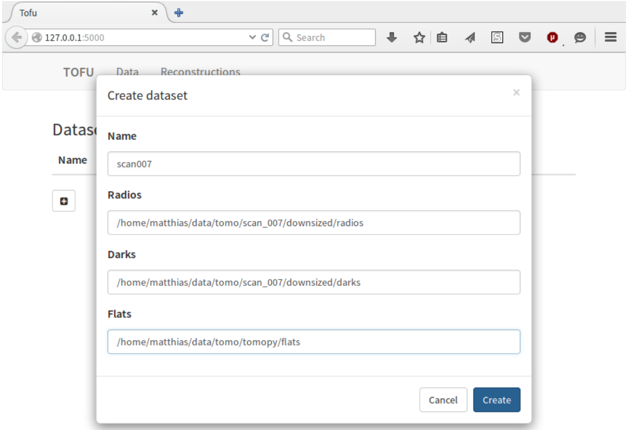
- Offline reconstruction for regular users
- Simplifies deployment and maintenance
- Uses Flask backend, Bootstrap frontend and WebGL for basic visualization

Web UI prototype



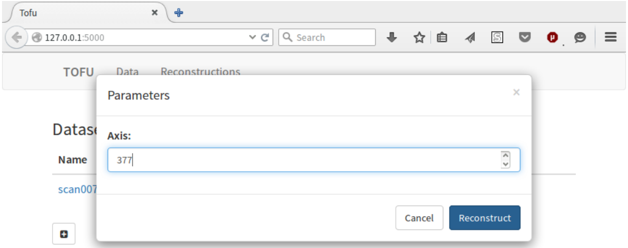
Create dataset from experiment data

Web UI prototype



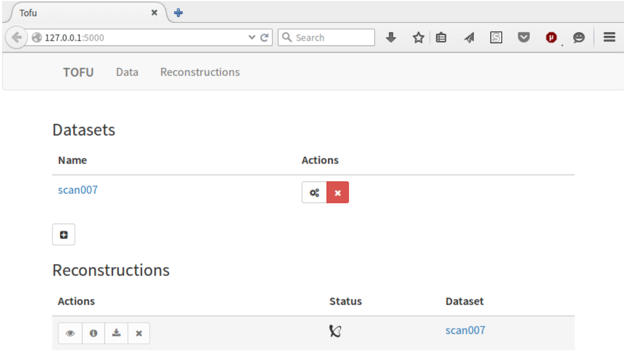
…by specifying paths for now.

Web UI prototype



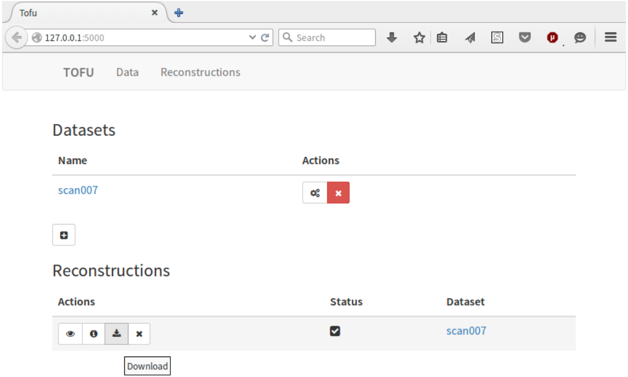
Start a reconstruction

Web UI prototype



and wait for reconstruction to finish.

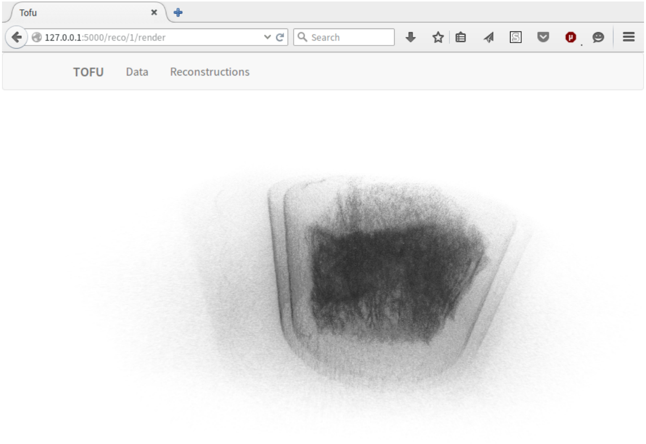
Web UI prototype



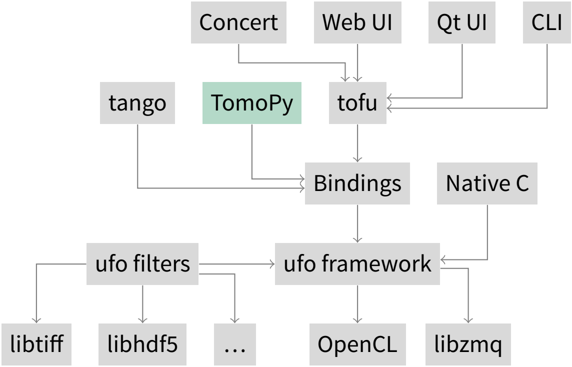
Download result

Web UI prototype

Karlsruhe Institute of Technology



...or visualize it.


Concert
Web UI
Qt UI
CLI
tango
TomoPy
tofu
Bindings
Native C
ufo filters
ufo framework
libtiff
libhdf5
...
OpenCL
libzmq

TomoPy

Versatile reconstruction platform

- TomoPy is APS' Python reconstruction toolkit
- A custom `ufo` Python module hooks into TomoPy

Benefits

- We can re-use existing I/O and pre-processing code
- TomoPy's reconstruction speed can be improved

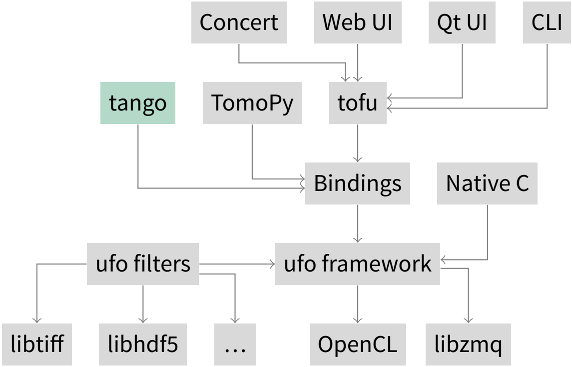
TomoPy Standard Demo

Karlsruhe Institute of Technology

```python
import tomopy

data, white, dark, theta = tomopy.xtomo_reader('demo/data.h5')

d = tomopy.xtomo_dataset()
d.dataset(data, white, dark, theta)
d.normalize()
d.correct_drift()
d.phase_retrieval()
d.correct_drift()
d.center = 661.5
d.gridrec()

tomopy.xtomo_writer(d.data_recon, 'tmp/test_', axis=0)
```

TomoPy + UFO

```
import tomopy
import ufo.tomopy   # <<<< new

data, white, dark, theta = tomopy.xtomo_reader('demo/data.h5')

d = tomopy.xtomo_dataset()
d.dataset(data, white, dark, theta)
d.normalize()
d.correct_drift()
d.phase_retrieval()
d.correct_drift()
d.center = 661.5
d.ufo_fbp()          # <<<< changed

tomopy.xtomo_writer(d.data_recon, 'tmp/test_', axis=0)
```


Concert
Web UI
Qt UI
CLI
tango
TomoPy
tofu
Bindings
Native C
ufo filters
ufo framework
libtiff
libhdf5
...
OpenCL
libzmq

TANGO integration

Karlsruhe Institute of Technology

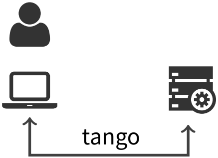
- tango is the European synchrotron *de facto* control system
- It provides "device" servers to access hardware devices or software services
- Using tango's Python interface we can provide remote computing within the existing control system infrastructure

TANGO
Connecting things together

Approach

Protocol

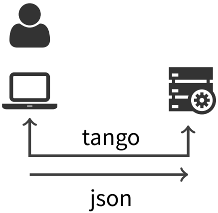
- Server listens for compute requests



Approach

Protocol

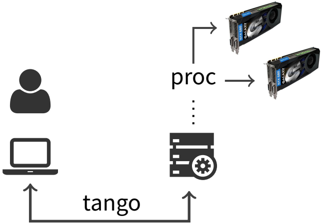
- Server listens for compute requests
- Client sets the `json` attribute and calls the `Run` or `RunContinuous` command



Approach

Protocol

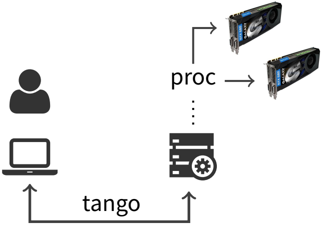
- Server listens for compute requests
- Client sets the `json` attribute and calls the `Run` or `RunContinuous` command
- The server spawns a new compute process identified by a process id



Approach

Protocol

- Server listens for compute requests
- Client sets the `json` attribute and calls the `Run` or `RunContinuous` command
- The server spawns a new compute process identified by a process id



Execution models

1. Single-run processes ("fire and forget")
2. Continuous processes (update description and re-run)

Single-run processes

Interface

```python
process = PyTango.DeviceProxy('hzgctkit/process/1')
process.json = "{ ... }"

pid = process.Run()
print(process.Running(pid))   # still running?
print(process.jobs)           # list of active jobs, e.g. [7041]

process.Wait(pid)
print(process.ExitCode(pid))  # return code of job
```

Single-run processes

Karlsruhe Institute of Technology

Interface

```
process = PyTango.DeviceProxy('hzgctkit/process/1')
process.json = "{ ... }"

pid = process.Run()
print(process.Running(pid))   # still running?
print(process.jobs)           # list of active jobs, e.g. [7041]

process.Wait(pid)
print(process.ExitCode(pid))  # return code of job
```

Remarks

- Simple to use and understand
- No extended use of resources

Continuous processes

Interface

```
pid = process.RunContinuous()
process.Continue(pid)       # trigger execution
process.json = "{ ... }"    # update description
process.Continue(pid)
process.Stop(pid)           # terminate process
```

Continuous processes

Karlsruhe Institute of Technology

Interface

```
pid = process.RunContinuous()
process.Continue(pid)       # trigger execution
process.json = "{ ... }"    # update description
process.Continue(pid)
process.Stop(pid)           # terminate process
```

Remarks

- Enables continuous exploration
- Resources are allocated as long as process is running
- Forgetting to call `Stop` leaks resources

Outlook

Karlsruhe Institute of Technology

Status

- The ufo framework provides various integration points
- All presented tools are open sourced and free for anyone to use

Ongoing

- Use tofu for the TomoPy integration
- Batch reconstruction within astor
- Finish web GUI and merge with the astor data portal