A comprehensive introduction to UFO

Matthias Vogelgesang et al.

matthias.vogelgesang@kit.edu

Institute for Data Processing and Electronics

In a nut shell

Karlsruhe Institute of Technology

- **Library** and set of **programs** for **general purpose** data processing
- Modularity through composition of individual blocks
- **GPU-accelerated** and **real-time** processing of data streams

- Access through C and Python APIs and tools built on top
- Runs on Linux and MacOS
- Is free and open source

github.com/ufo-kit

Manual

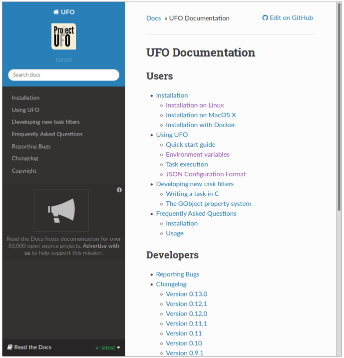
End-user and developer **documentation** is available from

ufo-core.readthedocs.org

Plugin **reference** is available from

ufo-filters.readthedocs.org

If you find errors or missing important information, do not hesitate and contact me.



INSTALLATION

Requirements

Karlsruhe Institute of Technology

- Any Linux[1] box with a working **OpenCL** run-time
- NVIDIA OpenCL supports NVIDIA GPUs
- AMD OpenCL supports AMD GPUs and x86_64 CPUs
- Intel OpenCL supports x86_64 CPUs and Intel Xeon Phi accelerators

OpenCL

[1] Mac works too but I cannot test that myself

Stable packages

Karlsruhe Institute of Technology

CentOS 7 and RHEL 7

```
wget http://download.opensuse.org/.../RHEL_7/home:ufo-kit.repo \
  -O /etc/yum.repos.d/ufo-kit.repo
yum update && yum install ufo-core ufo-filters
```

openSUSE 13.1 – Leap 42.2

```
zypper add repo \
  http://download.opensuse.org/.../openSUSE_Leap_42.2/home:ufo-kit.repo
zypper update && zypper install ufo-core ufo-filters
```

Debian 9 (Stretch), Ubuntu 17.04

```
apt install ufo-core ufo-filters
```

From source

Install dependencies

gcc or clang, CMake, GLib-2.0, json-glib, OpenCL, …[2]

Get sources

```
git clone https://github.com/ufo-kit/ufo-{core, filters}  # or ...
wget https://github.com/ufo-kit/ufo-{core, filters}/archive/v0.13.0.tar.gz
```

Build and install

```
cd ufo-{core-filters} && mkdir build && cd build
cmake .. && make && make test && make install
```

[2] …as well as any library required for actual work, e.g. libtiff or libhdf5.

USAGE

General idea

Dataflow model

- User models data transformation as **graph** of tasks
- **Tasks** generate or process data and pass the result on
- Properties allow run-time parameterization of tasks

Execution model

- Tasks use either GPUs or CPUs for computation
- Run-time scheduler maps tasks to processing units
- Scheduler allocates resources and minimizes data transfer overheads

Framework

Structure

- The dataflow and execution model is implemented as a **framework**
- Classes implement resource management, plugins processing behaviour
- C or Python API, JSON format or **command line interface**

Run-time options

- `G_MESSAGES_DEBUG=all` prints out debug messages
- `UFO_DEVICES=0,1,3` sets the OpenCL devices to use
- `UFO_DEVICE_TYPE=cpu` restricts OpenCL device types

Data processing on the command line

Idea

- Describe graph as a simple string of linear, nested tasks
- `ufo-launch` reads the description, instantiates plugins and runs the graph
- Further glue code can be written in any shell language

Syntax

- Tasks are parameterized with simple `key=value` pairs
- Tasks are separated with an exclamation mark (`!`)
- Multiple sources are grouped with square brackets and commas (`[t1, …, tn]`)

Input and output

Karlsruhe Institute of Technology

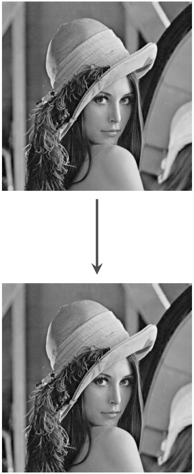
- Data flow must start with a **source** and end in a **sink** task
- Typical sources are file readers, sinks file writers
- `read` and `write` support Multi-TIFF, HDF5, JPG and raw files

```
ufo-launch read path=<file-or-path> start=0 number=1 !
           write filename=<file-or-spec>
```

- Input can be a file, a directory or a glob (`*`)
- Output can be a file, a format string (`out-%05i.tif`) or nothing for `stdout`

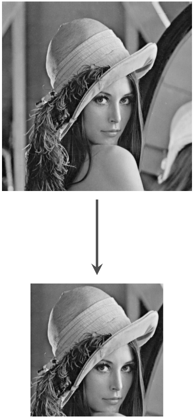
Basic operations

```
ufo-launch read path=lena.tif !
  write filename=out.h5:/dataset bits=16
```



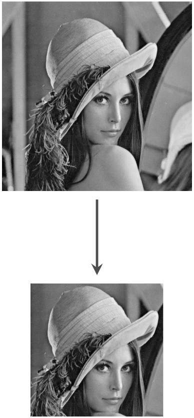
Basic operations

```
ufo-launch read path=lena.tif !
  crop x=71 y=38 width=360 height=360 !
  write filename=cropped.tif
```



Basic operations


```
ufo-launch read path=lena.tif !
  crop x=71 y=38 width=360 height=360 !
  calculate expression="v/256" !
  write filename=cropped.tif
```



Basic operations

```
ufo-launch read path=lena.tif !
  crop x=71 y=38 width=360 height=360 !
  calculate expression="v/256" !
  clip min=0.2 max=0.8 !
  write filename=cropped.tif
```

Basic operations

```
ufo-launch read path=lena.tif !
  crop x=71 y=38 width=360 height=360 !
  calculate expression="v/256" !
  clip min=0.2 max=0.8 !
  binarize threshold=0.5 !
  write filename=cropped.tif
```

Basic operations

```
ufo-launch read path=lena.tif !
  crop x=71 y=38 width=360 height=360 !
  calculate expression="v/256" !
  clip min=0.2 max=0.8 !
  binarize threshold=0.5 !
  flip !
  write filename=cropped.tif
```

Arbitrary OpenCL

Kernel

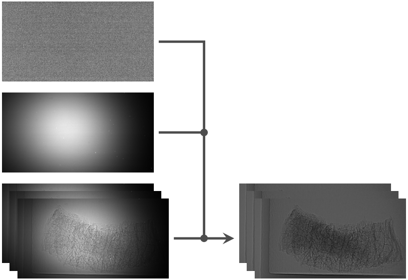
```
kernel void diff (global float *a, global float *b, global float *c)
{
  size_t idx = get_global_id(1) * get_global_size(0) + get_global_id(1);
  c[idx] = a[idx] - b[idx];
}
```

Integration

```
ufo-launch [ read path=a/ , read path=b/ ] !
  opencl filename=diff.cl kernel=diff !
  write filename=difference.tif
```

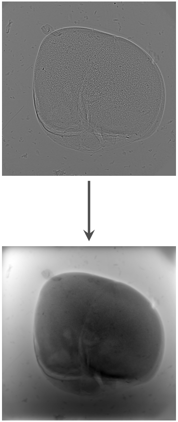
Flat field correction

```
ufo-launch [read path=radios/,
  read path=darks ! average,
  read path=flats ! average] !
  flat-field-correct
  write filename=out/out-%05i.tif
```



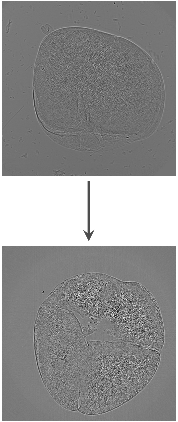
Phase retrieval

```
ufo-launch read path=input/ !
  fft dimensions=2 !
  retrieve-phase energy=20
                 distance=0.945
                 pixel-size=0.75e-6 !
  ifft dimensions=2 !
  write filename=correct/correct-%05i.tif
```



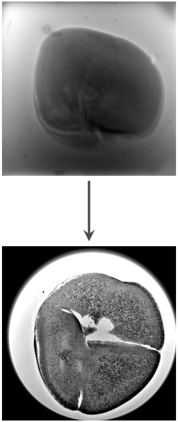
Reconstruction

```
ufo-launch read path=input/ !
  transpose-projections number=1605 !
  fft ! filter ! ifft !
  backproject axis-pos=1066.5
              angle-step=0.00393 !
  write filename=slices/slice-%05i.tif
```



Reconstruction

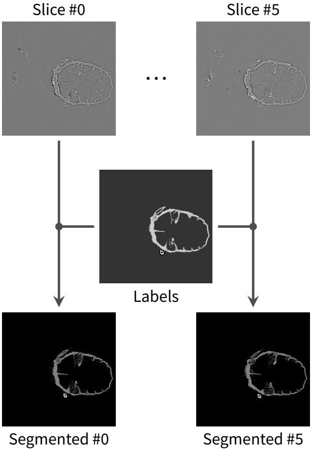
```
ufo-launch read path=correct/ !
  transpose-projections number=1605 !
  fft ! filter ! ifft !
  backproject axis-pos=1066.5
              angle-step=0.00393 !
  write filename=slices/slice-%05i.tif
```



Semi-automatic segmentation

Karlsruhe Institute of Technology

```
ufo-launch [read path=slices/ !
             stack number=20,
           read path=labels.tif] !
  segment !
  write filename=segmented.tif
```



Language bindings

- `ufo-launch` is written on top of the C API
- Constructing the pipeline directly using the C API is possible but cumbersome

- Introspection mechanism enables third-party language integration …
- … for example JavaScript, Python, Ruby, Lua, Go and Haskell
- Our primary target for now is Python

Python bindings resemble C API

```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)
```

Improving Python support

Unlocking the 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

Karlsruhe Institute of Technology

```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

Karlsruhe Institute of Technology

```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))
```

High-level Python

Karlsruhe Institute of Technology

```python
from ufo import Rescale

data = [np.ones((1024, 1024)) * i for i in range(10)]
rescale = Rescale(factor=0.5)

# insert NumPy arrays
for image in rescale(data):
    print(np.mean(image))
```

Tofu reconstruction

Idea

- Move reconstruction-related code to single Python module
- Simplify setup and execution of reconstruction pipelines using UFO
- Provide visualization widgets based on PyQtGraph

Focus

- Dark current and flat field correction
- Tomographic reconstruction with FBP, DFI and IR method
- Laminographic reconstruction with FBP
- Manual and automatic axis alignment

Command line interface

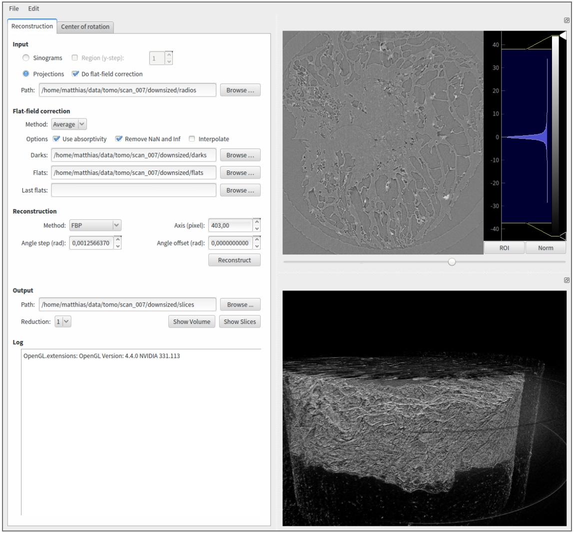
- Offline reconstruction for power users
- Parameters are stored in a configuration file

```
$ tofu init
$ vi reco.conf
$ tofu tomo
```

- Command line arguments can override parameters

```
$ tofu tomo --axis=234.5
```

Qt GUI


File
Edit
Reconstruction
Center of rotation
Input
Sinograms
Region (y-step):
1
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
Browse ...
Flats:
/home/matthias/data/tomo/scan_007/downsized/flats
Browse ...
Last flats:
Browse ...
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
Browse ...
Reduction:
1
Show Volume
Show Slices
Log
OpenGL.extensions: OpenGL Version: 4.4.0 NVIDIA 331.113
40
30
20
10
0
-10
-20
-30
-40
ROI
Norm

TomoPy + Gridrec

Karlsruhe Institute of Technology

```python
import tomopy, dxchange

proj, flat, dark, theta = dxchange.read_aps_32id('tooth.h5', sino=(0, 2))
proj = tomopy.normalize(proj, flat, dark)
proj = tomopy.minus_log(proj)
center = tomopy.find_center(proj, theta, init=290, ind=0, tol=0.5)
recon = tomopy.recon(proj, theta, center=center,
                     algorithm='gridrec')
```

TomoPy + UFO

```python
import tomopy, dxchange, ufo.tomopy

proj, flat, dark, theta = dxchange.read_aps_32id('tooth.h5', sino=(0, 2))
proj = tomopy.normalize(proj, flat, dark)
proj = tomopy.minus_log(proj)
center = tomopy.find_center(proj, theta, init=290, ind=0, tol=0.5)
recon = tomopy.recon(proj, theta, center=center,
                     algorithm=ufo.tomopy.fbp, ncore=1)
```

OUTLOOK

Parameterized pipelines

Problem

- Application-specific tools can only cover a limited set of problems
- Parameterization requires additional programming or shell wrappers

Solution

- Separate problem description from parameterization
- Templates contain general problem
- Parameters are expanded at run-time

Proof of concept

- Combine existing JSON description language with Jinja templates
- Pass parameter values and value ranges at run-time
- Each parameter permutation causes a single **run**

Example

Karlsruhe Institute of Technology

Template

```
{
  "plugin": "binarize", "properties": { "threshold": {{ threshold }} }
},
{
  "plugin": "write",
  "properties": { "filename": t-{{ threshold|round(1) }}.tif }
},
```

Execution

```
$ python runner.py run threshold.json threshold=0:128:32
```

produces `t-0.tif`, `t-4.tif` etc. with threshold set accordingly

CONCLUSION

Summary

Karlsruhe Institute of Technology

- UFO is a framework for high-throughput, general purpose image processing
- Automatic scheduling on heterogeneous compute systems
- High-level tools on top and integration with other systems

QUESTIONS?

github.com/ufo-kit

matthias.vogelgesang@kit.edu