Karlsruhe Institute of Technology

PyHST – from CUDA to OpenCL into UFO

Matthias Vogelgesang – *matthias.vogelgesang@ipe.fzk.de*

Institut für Prozessdatenverarbeitung und Elektronik

Outline

- OpenCL in PyHST
- UFO build system

Reasons for OpenCL

- Platform-independent, vendor-neutral and open standard
- Will eventually also cover non-GPU compute devices $\rightarrow$ easier to build heterogenous systems
- Therefore, implementation for the reconstruction software *PyHST* and the future UFO software is needed

Current state of OpenCL in PyHST

- Compiles, links and runs on NVIDIA and AMD/ATI systems
- No support for FAI360
- No OpenCL-CPU support on AMD/ATI
- Grab latest revision at

```
bzr+ssh://user@ufo.kit.edu/vogelgesang/PyHST
```

Performance Preliminaries

There are four dominant phases in the reconstruction process:

- Input/output (largest fraction but neglected here)
- Memory transfer CPU $\leftrightarrow$ GPU
- FFT and filtering
- Back-projection of filtered projections

Tests with three data sets on two NVIDIA GTX580:

Name	# Projections	Width	Height	# Slices
small	250	832	832	400
medium	2000	1311	1331	1700
large	2500	4008	4008	2500

Performance Results

Results are in seconds per slice and device/core. Fastest results highlighted.

System	Name	Data Transfer	FFT & Filtering	Back-Projection
OpenCL	small	**0.00118**	0.000318	0.00509
	medium	**0.00518**	0.004140	0.10649
	large	0.02222	0.010704	1.27534
CUDA	small	0.00130	**0.000266**	**0.00447**
	medium	0.01438	**0.003285**	**0.08169**
	large	**0.00741**	**0.007914**	**0.92508**
CPU i7-950	small		0.017234	0.45104
	medium		0.297115	8.27456
	large		0.881619	88.84089

Conclusion

- Back-projection is the most time-consuming task (memory accesses cannot be easily coalesced)
- Depending on problem size, CUDA's back-projection is 12 to 27 *percent* faster
- However, one OpenCL device is still 90 to 70 *times* faster than a single CPU core

- No pinned memory on OpenCL implemented yet
- CUDA support might still be more mature
- Better optimized FFT
- ~~Texture data type used in CUDA~~

UFO source structure

- Proposed an initial directory structure on the mailing list and developed basic build system

```
.
|-- thirdparty          |-- ufo-core
|   \-- oclfft          |   |-- doc
|       |-- src         |   |-- src
|       \-- tests       |   |   |-- devices
|-- ufo-bindings        |   |   |-- mgmt
|-- ufo-common          |   |   \-- processing
|   \-- cmake           |   \-- tests
...                     \-- ufo-tools
```

- Grab latest revision at

```
bzr+ssh://user@ufo.kit.edu/vogelgesang/ufo
```

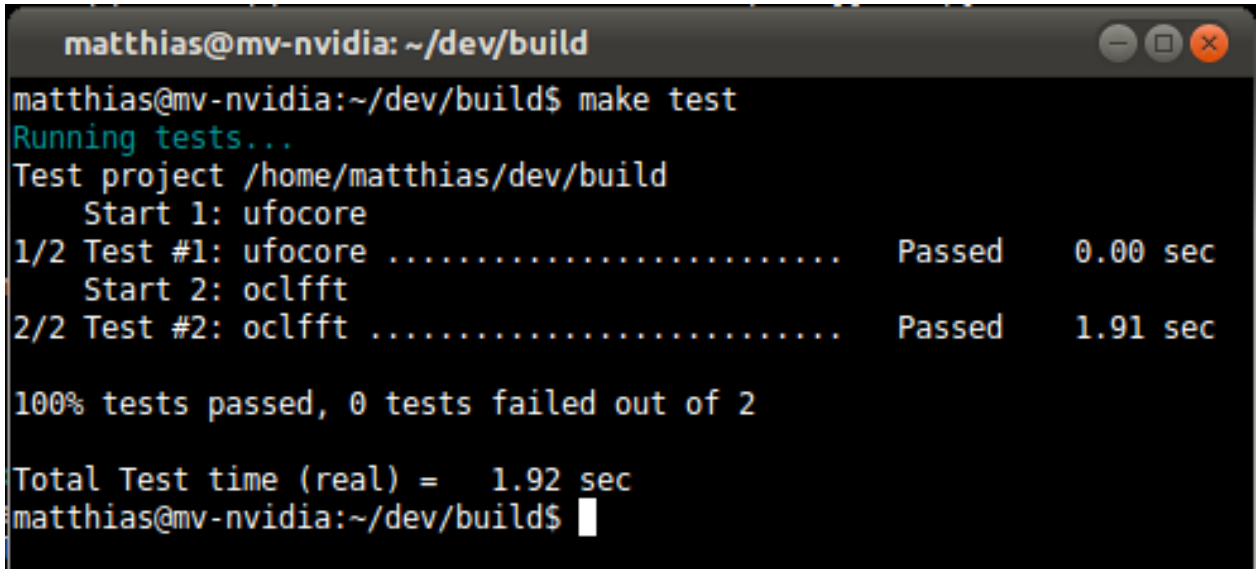
Out-of-source Builds

cmake && make

```
matthias@mv-nvidia: ~/dev/build
matthias@mv-nvidia:~/dev/build$ ls
matthias@mv-nvidia:~/dev/build$ cmake ../ufo/local/ && make
-- The C compiler identification is GNU
-- The CXX compiler identification is GNU
-- Check for working C compiler: /usr/bin/gcc
-- Check for working C compiler: /usr/bin/gcc -- works
-- Detecting C compiler ABI info
                                        ... more checking
/fft_kernelstring.cpp.o
Linking CXX static library liboclfft.a
[ 50%] Built target oclfft
Scanning dependencies of target ufocore
[ 66%] Building C object ufo-core/src/CMakeFiles/ufocore.dir/ufo.c.o
Linking CXX static library libufocore.a
[ 66%] Built target ufocore
Scanning dependencies of target ufotest
[ 83%] Building C object ufo-core/tests/CMakeFiles/ufotest.dir/ufo-tes
ts.c.o
Linking CXX executable ufotest
[ 83%] Built target ufotest
Scanning dependencies of target oclfft-test
[100%] Building C object thirdparty/oclfft/tests/CMakeFiles/oclfft-tes
t.dir/oclfft-tests.c.o
Linking CXX executable oclfft-test
[100%] Built target oclfft-test
matthias@mv-nvidia:~/dev/build$
```

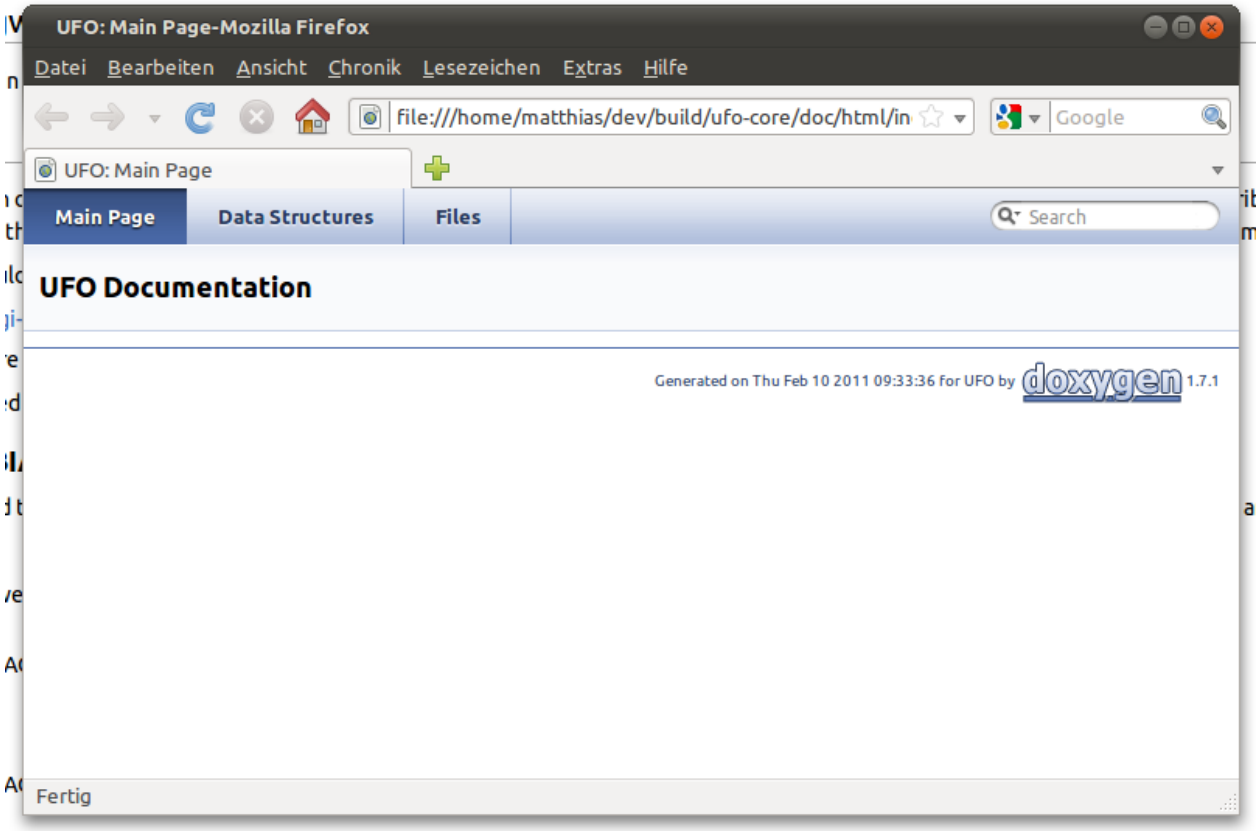
Integrated unit tests

make test



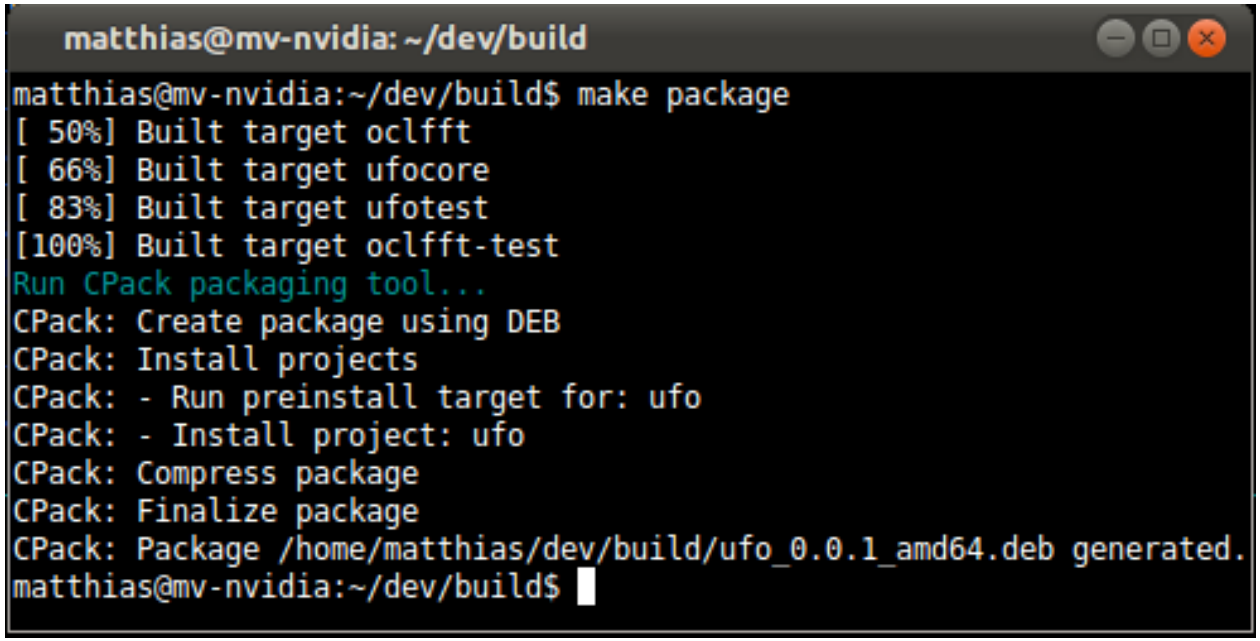
Documentation

make doc



Packaging

make package



Any Questions?

My entry for the logo contest ;-)