# Speeding up Praxes code with Numba and CUDA Python

## Summary

In this work we created GPU versions of the low-level functions for **gvecToDetectorXY**, **DectorXYToGVec**, and **oscillAnglesofHKLs**, using CUDA Python in Numba. In all GPU code, there is a constant overhead of about 11ms to 16ms that requires a problem of sufficient size to be run on the GPU in order to see improvement over equivalent CPU code. If a problem of sufficient size is run, we were able to see improvement of 2x to 68x using the GPU versions of the code.

Assuming your data-sizes will be large-enough to justify it, we recommend continuing with this project to build a version of the library that uses the GPU with and end-to-end workflow. We estimate 2.5-person months of effort to complete the new version of the software including testing and documentation. We also recommend a period of pre-paid support of at least 2-weeks to allow continued maintenance and updates over one-year of the software.

All of our code is checked in to a GitHub repository ContinuumIO/hexrd which is a fork of praxes/hexrd. Included in the repository is a directory called “recipe” for building a hexrd conda package.

## Procedure

We translated the C transform functions gvecToDetectorXY\_cfunc, detectorXYToGvec\_cfunc, oscillAnglesOfHKLs\_cfunc (found in hexrd / transforms / transforms\_CFUNC.c) into CUDA Python functions. The python functions were created in hexrd/xrd/pycfunc\_transforms.py. The Python code is a copy of the C code with the majority of the modifications being confined to 1) translating the C-style for loops to Python for i in range statements, 2) adjusting the indexing of matrices that were flattened in C to use row, column indexing in Python, and 3) passing in arrays as parameters for NUMBA to work in nopython mode rather than array creation inside the function. The main for loop for each function was then placed in its own function so Numba CUDA could be applied to the main for loop.

The three provided benchmark test scripts in hexrd/test/:

test\_gVecToDetectorXY.py

test\_DetectorXYToGVec.py

test\_oscillAnglesofHKLs.py

were copied to new files and prefixed with the name “pycfuncs\_” and code added to 1) benchmark the Numba test code and 2) asserts were added to make sure data produced by the Numba code matched that of the provided Python and C code.

The benchmark code was then run on a Linux machine with the following specs:

Ubuntu 12.04

Cuda 6.0

4 - Intel(R) Core(TM) i7-4820K CPU @ 3.70GHz

32G RAM

GPU Tesla K20c

**Benchmarks:**

All timing data is given in seconds.

1. $ python pycfuncs\_test\_gVecToDetectorXY.py

Timing Data for gVecToDetectorXY with increasing number of n for pvec, where pvec = 204.8 \* np.linspace(-1, 1, n). A size of 16384 gave a memory error.

|  |  |  |  |
| --- | --- | --- | --- |
| N | LLNL Python | LLNL C Code | CUDA Python Code |
| 512 | 4.478197 | 0.009959 | 0.014121 |
| 2048 | 72.365316 | 0.170658 | 0.093493 |
| 8192 | Not Ran | 2.726921 | 1.330982 |

2. $ python pcfuncs\_DetectorXYToGVec.py

Timing Data for DetectorXYToGVec with increasing number of n for pvec, where

pvec = 204.8 \* np.linspace(-1, 1, n). A size of 16384 gave a memory error.

|  |  |  |  |
| --- | --- | --- | --- |
| n | LLNL Python | LLNL C Code | CUDA Python Code |
| 512 | 0.138302 | 0.041341 | 0.016000 |
| 2048 | 3.022490 | 0.695421 | 0.133712 |
| 8192 | 48.659840 | 11.084787 | 1.924594 |

3. $ python pycfuncs\_test\_oscillAnglesofHKLs.py

Timing Data for oscillAnglesofHKLs function with increasing input file size.

To increase the file size, the original data (ruby\_4537-8\_log.txt) sans header row was just copied and pasted.

|  |  |  |  |
| --- | --- | --- | --- |
| File Size | LLNL Python | LLNL C Code | CUDA Python Code |
| 50K | 0.345038 | 0.002502 | 0.011282 |
| 150K | 1.105788 | 0.007042 | 0.011044 |
| 600K | 5.968502 | 0.027526 | 0.011293 |
| 1.2M | 14.858672 | 0.054515 | 0.011107 |
| 4.7M | not ran | 0.223333 | 0.013588 |
| 9.4M | not ran | 0.447933 | 0.016930 |
| 19M | not ran | 0.897627 | 0.043904 |
| 38M | not ran | 1.788874 | 0.037020 \* |
| 75M | not ran | 3.575378 | 0.060746 |
| 150M | not ran | 7.193876 | 0.111216 |
| 300M | not ran | 14.317272 | 0.209414 |

\* - Tests for filesize of 19M and 38M was run three times. Each time the code ran faster with a file size of 38M than 19M.