

Lab 1: Saxpy in “CUDA Python”

Objective

- Implement saxpy in “CUDA Python”

Quick Lesson to CUDA Python

- Similar to CUDA-C
- `threadIdx, blockIdx, blockDim` -> `cuda.threadIdx, cuda.blockIdx, cuda.blockDim`
 - `i = cuda.threadIdx.x + cuda.blockIdx.x * cuda.blockDim.x`
- Host->Device
 - `d_ary = cuda.to_device(ary)`
- Host->Device (allocate only, no copy)
 - `d_ary = cuda.to_device(ary, copy=False)`
- Device->Host
 - `d_ary.to_host()`
- Decorate kernel
 - `cuda.autojit, cuda.jit`
- Kernel launch
 - `a_kernel[griddim, blockddim](arg0, arg1)`
 - similar to C: `a_kernel<<<griddim, blockdim>>>(arg0, arg1)`
 - `griddim`: tuple of 1-2 ints
 - `blockdim`: tuple of 1-3 ints

Exercise (10 mins)

```
from numba import cuda
from numba import *
import numpy as np
import math
from timeit import default_timer as time
```

```
@cuda.autojit
def saxpy(Out, X, Y, Z):
    "Compute Out = X * Y + Z"
    # ----- Exercise -----
    # Complete this kernel
    # threadIdx ---> cuda.threadIdx
    # blockIdx ---> cuda.blockIdx
    # blockDim ---> cuda.blockDim
```

```

def main():
    # Prepare data
    thread_per_block = 512
    block_per_grid = 10
    n = thread_per_block * block_per_grid
    X = np.random.random(n)
    Y = np.random.random(n)
    Z = np.random.random(n)
    Out = np.empty_like(X)

    # ----- Exercise -----
    # Host->Device
    # Complete the transfer for Y, Z, and Out

    # Kernel launch
    blockdim = thread_per_block, 1, 1
    griddim = block_per_grid, 1

    # ----- Exercise -----
    # Kernel launch
    # Complete the kernel launch for saxpy

    # ----- Exercise -----
    # Device->Host
    # Complete the transfer for dOut

    print('-- Result --')
    print(Out)
    # Verify
    print("verify: %s" % np.allclose(X * Y + Z, Out))

if __name__ == '__main__':
    main()

```