09_ex_Numba

March 14, 2020

0.1 Exercise 1

Create a vectorized version of log and exp math function for 1D array A = [2, 5, 10, 3, 8]

Results should be: + [0.6931472 1.609438 2.3025851 1.0986123 2.0794415] + [7.3890562e+00 1.4841316e+02 2.2026465e+04 2.0085537e+01 2.9809580e+03]

```
In [1]: import os
        import numpy as np
        import math
        from numba import jit, njit, vectorize, cuda, int64, float32, int32, int16
```

```
In [2]: @vectorize([float32(float32)], target='parallel', fastmath=True)
        def exp_vect(arr):
            y= math.exp(arr)
            return y
        @vectorize([float32(float32)], target='parallel', fastmath=True)
        def log_vect(arr):
            y=math.log(arr)
            return y

        A=np.array([2,5,10,3,8], dtype='float32')

        print(log_vect(A))
        print(exp_vect(A))
```

```
[0.6931472 1.609438  2.3025851 1.0986123 2.0794415]
[7.3890562e+00 1.4841316e+02 2.2026465e+04 2.0085537e+01 2.9809580e+03]
```

0.2 Exerice 2

Compute the value of a Gaussian probability density function at x with $mean = 1$, $\sigma = 1$, lower and upper bound in $(-3, 3)$ and $size = 100000$

```
In [3]: import math
        import numpy as np

        mean=1
        sigma=1
```

```
@vectorize([int32(float32,float32)], target="parallel", fastmath=True)
def hit_or_miss(x,y):
    global mean
    global sigma
    p=math.exp(-(((x-mean)/sigma)**2)/2)
    if p<y:
        c=1
    else:
        c=0
    return c
N=1000000


x=np.random.uniform(-3,3,N).astype(np.float32)
y=np.random.uniform(-3,3,N).astype(np.float32)

value=np.sum([hit_or_miss(x,y)])/N
print(value)
```

```
0.43355
```

0.3 Exercise 3

Create a "zero suppression" function. A common operation when working with waveforms is to force all samples values below a certain absolute magnitude to be zero, as a way to eliminate low amplitude noise. Plot the data before and after the application of the zero_suppress function.

$thresold = 15$

```
In [4]: %matplotlib inline
        from matplotlib import pyplot as plt

        n = 100000
        noise = np.random.normal(size=n) * 3
        pulses = np.maximum(np.sin(np.arange(n) / (n / 23)) - 0.3, 0.0)
        data = ((pulses * 300) + noise).astype(np.int16)



        @vectorize([int16(int16)], target="parallel", fastmath=True)
        def noise_suppress (x):
            if x<15:
                x=0
            return x

        cleaned_data=noise_suppress(data)
```