Anubha Bhargava
Homework 3

I created two scripts for Homework 3 to demonstrate the difference between using Python and using a GPU. In this homework, I used PyOpenCL.

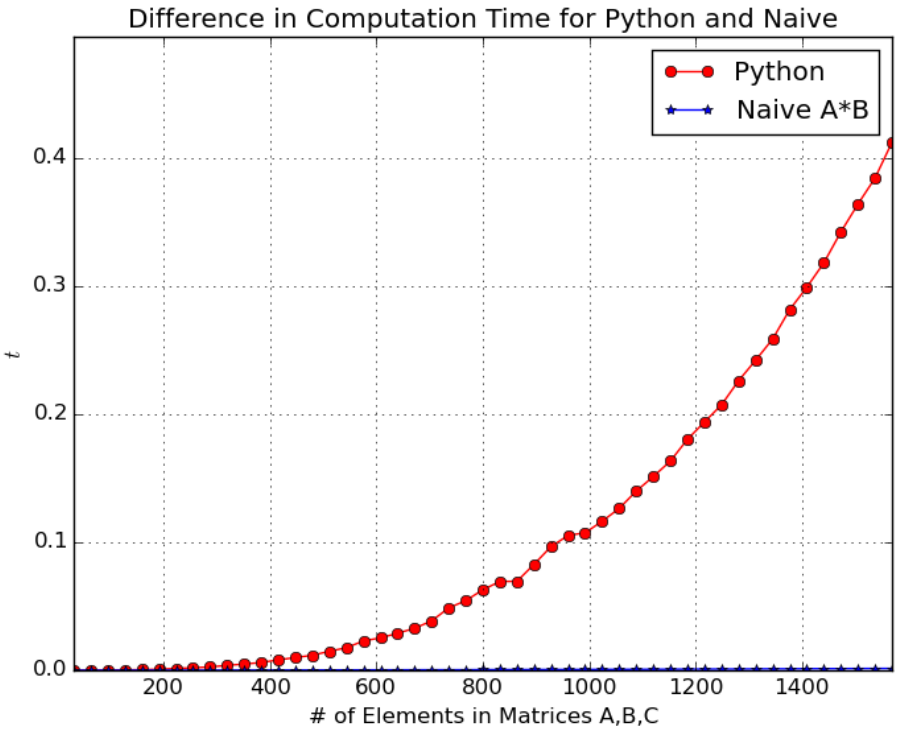
The first script does simple Naïve matrix multiplication and outputs the dimensions of the matrix, the time it takes Python to compute the result and the time it takes for naïve implementation. The following is the output of the script:

```
Multiplication of y=A*B

Dimensions      Python Time     Naive Implementation Time
( 32 , 32 )     1.55568122864e-05       5.29885292053e-05
( 64 , 64 )     0.000629007816315       5.37037849426e-05
( 96 , 96 )     0.000118255615234       5.82933425903e-05
( 128 , 128 )   0.00035697221756        6.46710395813e-05
( 160 , 160 )   0.000530481338501       6.77704811096e-05
( 192 , 192 )   0.000795245170593       7.48038291931e-05
( 224 , 224 )   0.00143790245056        8.27312469482e-05
( 256 , 256 )   0.0019662976265         9.13143157959e-05
( 288 , 288 )   0.00262588262558        0.00010347366333
( 320 , 320 )   0.00388437509537        0.000148832798004
( 352 , 352 )   0.00492936372757        0.000150799751282
( 384 , 384 )   0.00607448816299        0.000185966491699
( 416 , 416 )   0.00818824768066        0.000196516513824
( 448 , 448 )   0.00989258289337        0.000217974185944
( 480 , 480 )   0.0117400884628         0.000235259532928
( 512 , 512 )   0.0148465037346         0.000326991081238
( 544 , 544 )   0.0172622799873         0.000323951244354
( 576 , 576 )   0.0207367539406         0.000343680381775
( 608 , 608 )   0.0250620245934         0.000365555286407
( 640 , 640 )   0.0289934873581         0.000401556491852
( 672 , 672 )   0.0326902270317         0.000411748886108
( 704 , 704 )   0.0385386943817         0.000409245491028
( 736 , 736 )   0.0436209440231         0.000451385974884
( 768 , 768 )   0.0486841797829         0.000483512878418
( 800 , 800 )   0.0630772709846         0.000519216060638
( 832 , 832 )   0.0695405006409         0.000540494918823
( 864 , 864 )   0.0777850151062         0.000556290149689
( 896 , 896 )   0.0901777148247         0.000619232654572
( 928 , 928 )   0.0953832864761         0.00062495470047
( 960 , 960 )   0.0953977704048         0.00066351890564
( 992 , 992 )   0.118944406509  0.000709295272827
( 1024 , 1024 ) 0.11710357666   0.000756561756134
( 1056 , 1056 ) 0.125670313835  0.00078547000885
( 1088 , 1088 ) 0.139838218689  0.0008185505867
( 1120 , 1120 ) 0.153462707996  0.000866234302521
( 1152 , 1152 ) 0.177244484425  0.000918030738831
( 1184 , 1184 ) 0.179315984249  0.000923216342926
( 1216 , 1216 ) 0.193480014801  0.000964283943176
( 1248 , 1248 ) 0.215669929981  0.00105929374695
( 1280 , 1280 ) 0.24375975132   0.00109779834747
( 1312 , 1312 ) 0.244932949543  0.00109422206879
( 1344 , 1344 ) 0.26440101862   0.00118952989578
( 1376 , 1376 ) 0.286651492119  0.0012184381485
( 1408 , 1408 ) 0.299344539642  0.00132375955582
( 1440 , 1440 ) 0.342114031315  0.00134575366974
( 1472 , 1472 ) 0.346270978451  0.00140124559402
( 1504 , 1504 ) 0.374393045902  0.00147473812103
( 1536 , 1536 ) 0.411134004593  0.00152122974396
( 1568 , 1568 ) 0.424044489861  0.00153976678848
```

The above shows that naïve implementation works much faster than python. To see the efficiency of the algorithm, we plot naïve matrix multiplication of matrix A*B against the time it takes Python to compute. It takes a much longer time as the number of elements increase for Python to get to the result.

Anubha Bhargava
Homework 3



In my second script, I chose to compare four different algorithms. I compared the naïve implementation, naïve with scalars as the middle values of the matrix, naïve using local memory to compute and transposing the matrix. First, I outputted the time it took for Python and OpenCL to compute the transpose algorithm. As displayed, the time for Python to compute this transpose matrix algorithm was much faster than the time it took for PyOpenCL. In order to confirm the algorithms were equal as well, the output displays a check as shown below:

```
OUTPUT:

Python time for Transpose Algorithm:   1.90734863281e-06
OpenCL time for Transpose Algorithm:   0.0016770362854

Are Python and PyOpenCL Algorithms equal?
Naive OpenCL = Python:  True
Naive OpenCL with Scalars as Middle Values = Python:  True
OpenCL using Local Memory = Python:  True
Transpose:  True
```

To compare the different algorithms, I chose to plot 3 of the algorithms. As shown, using scalars as middle values is the fastest implementation, while using local memory was the slowest. Also, when

Anubha Bhargava
Homework 3

changing the numbers of elements in the array from 32x64 to 64x128, it didn't make a difference.