AuE 8930: Computing and Simulation for Autonomy

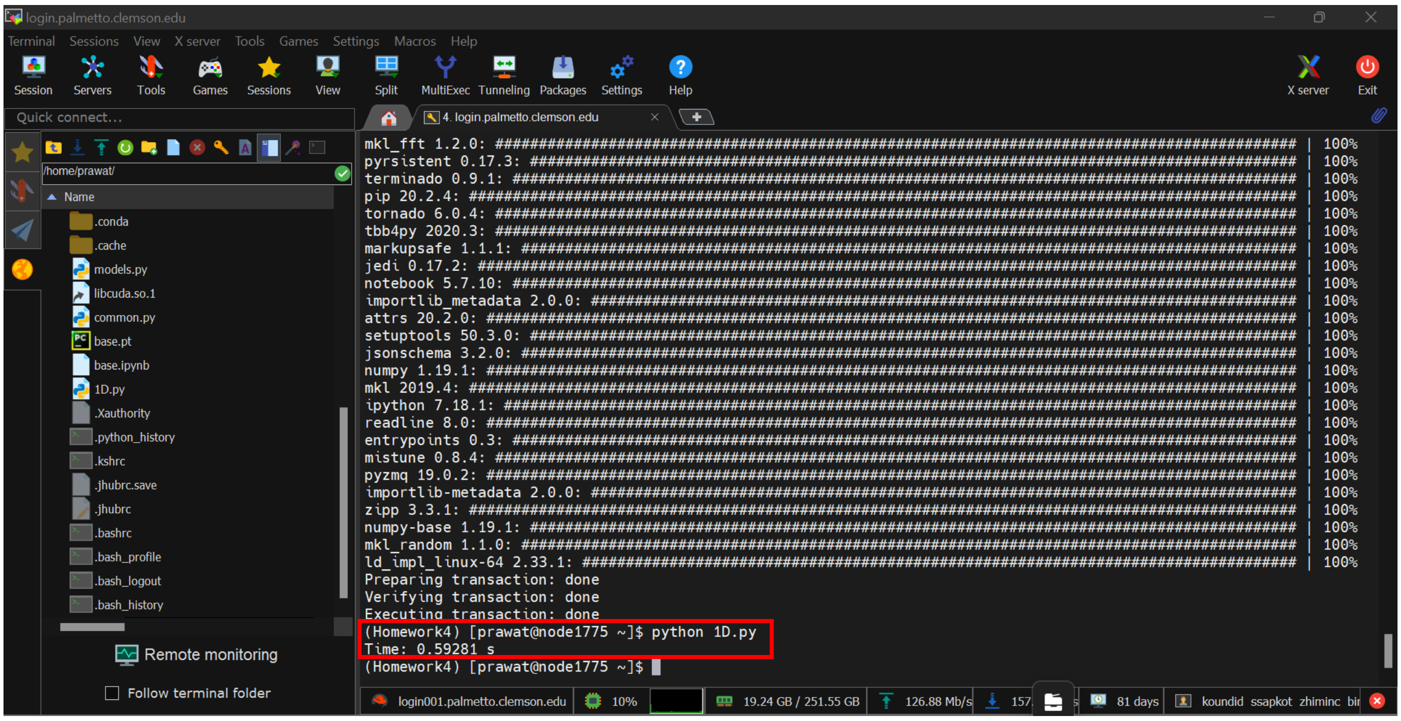
Priyanshu Rawat

<u>GitHub Link:</u> https://github.com/priyanshurawat1509/HPC_Homework_4

<u>Question 1)</u>

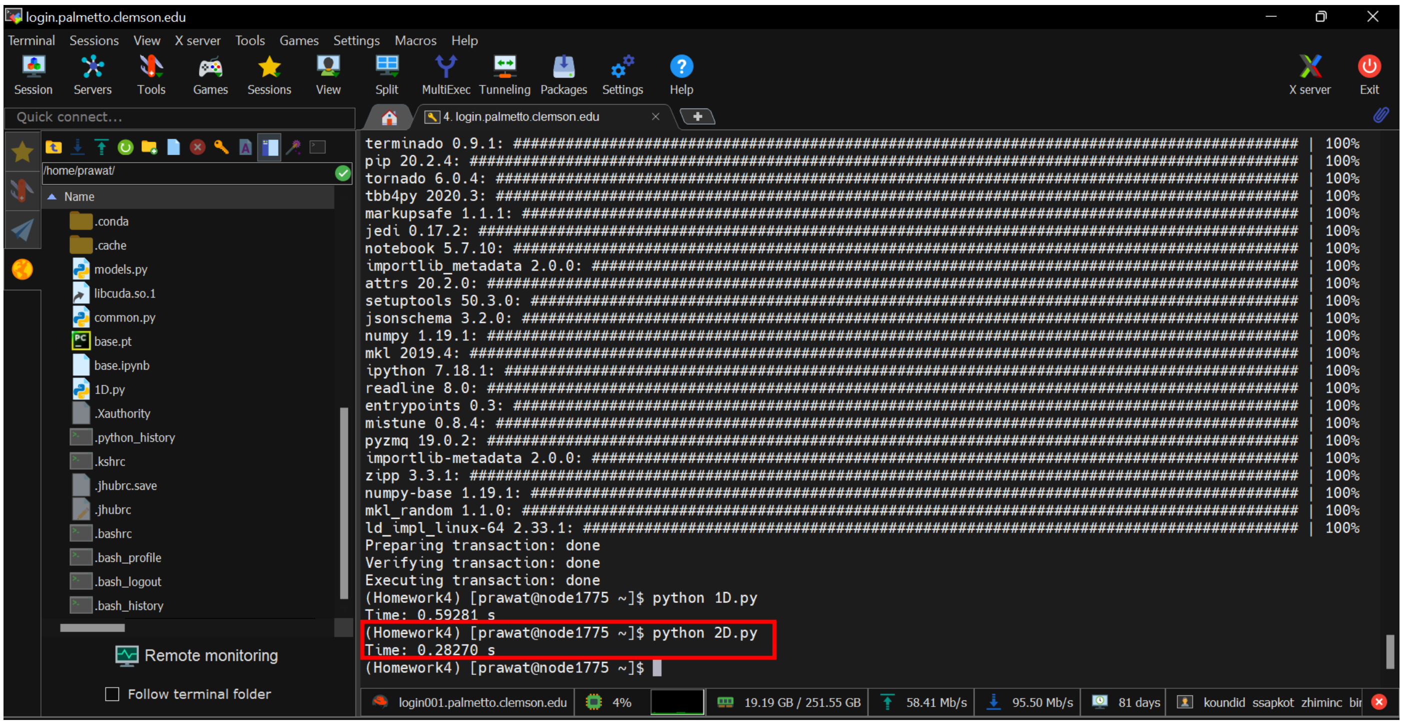
(1) Install Pycuda on Palmetto Cluster and run 1D.py and 2D.py, show screenshots. (5 points)

<u>1D.py</u>


login.palmetto.clemson.edu
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help X server Exit
Quick connect...
4. login.palmetto.clemson.edu
/home/prawat/
Name
.conda
.cache
models.py
libcuda.so.1
common.py
base.pt
base.ipynb
1D.py
.Xauthority
.python_history
.kshrc
.jhubrc.save
.jhubrc
.bashrc
.bash_profile
.bash_logout
.bash_history
Remote monitoring
Follow terminal folder
login001.palmetto.clemson.edu 10% 19.24 GB / 251.55 GB 126.88 Mb/s 157 81 days koundid ssapkot zhiminc

```
mkl_fft 1.2.0: ####################################################################### | 100%
pyrsistent 0.17.3: #################################################################### | 100%
terminado 0.9.1: ###################################################################### | 100%
pip 20.2.4: ########################################################################### | 100%
tornado 6.0.4: ######################################################################## | 100%
tbb4py 2020.3: ######################################################################## | 100%
markupsafe 1.1.1: ##################################################################### | 100%
jedi 0.17.2: ########################################################################## | 100%
notebook 5.7.10: ###################################################################### | 100%
importlib_metadata 2.0.0: ############################################################## | 100%
attrs 20.2.0: ######################################################################### | 100%
setuptools 50.3.0: #################################################################### | 100%
jsonschema 3.2.0: ##################################################################### | 100%
numpy 1.19.1: ######################################################################### | 100%
mkl 2019.4: ########################################################################### | 100%
ipython 7.18.1: ####################################################################### | 100%
readline 8.0: ######################################################################### | 100%
entrypoints 0.3: ###################################################################### | 100%
mistune 0.8.4: ######################################################################## | 100%
pyzmq 19.0.2: ######################################################################### | 100%
importlib-metadata 2.0.0: ############################################################## | 100%
zipp 3.3.1: ########################################################################### | 100%
numpy-base 1.19.1: #################################################################### | 100%
mkl_random 1.1.0: ##################################################################### | 100%
ld_impl_linux-64 2.33.1: ############################################################### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
(Homework4) [prawat@node1775 ~]$ python 1D.py
Time: 0.59281 s
(Homework4) [prawat@node1775 ~]$
```

<u>2D.py</u>


login.palmetto.clemson.edu
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help X server Exit
Quick connect...
4. login.palmetto.clemson.edu
/home/prawat/
Name
.conda
.cache
models.py
libcuda.so.1
common.py
base.pt
base.ipynb
1D.py
.Xauthority
.python_history
.kshrc
.jhubrc.save
.jhubrc
.bashrc
.bash_profile
.bash_logout
.bash_history
Remote monitoring
Follow terminal folder
login001.palmetto.clemson.edu 4% 19.19 GB / 251.55 GB 58.41 Mb/s 95.50 Mb/s 81 days koundid ssapkot zhiminc

```
terminado 0.9.1: ###################################################################### | 100%
pip 20.2.4: ########################################################################### | 100%
tornado 6.0.4: ######################################################################## | 100%
tbb4py 2020.3: ######################################################################## | 100%
markupsafe 1.1.1: ##################################################################### | 100%
jedi 0.17.2: ########################################################################## | 100%
notebook 5.7.10: ###################################################################### | 100%
importlib_metadata 2.0.0: ############################################################## | 100%
attrs 20.2.0: ######################################################################### | 100%
setuptools 50.3.0: #################################################################### | 100%
jsonschema 3.2.0: ##################################################################### | 100%
numpy 1.19.1: ######################################################################### | 100%
mkl 2019.4: ########################################################################### | 100%
ipython 7.18.1: ####################################################################### | 100%
readline 8.0: ######################################################################### | 100%
entrypoints 0.3: ###################################################################### | 100%
mistune 0.8.4: ######################################################################## | 100%
pyzmq 19.0.2: ######################################################################### | 100%
importlib-metadata 2.0.0: ############################################################## | 100%
zipp 3.3.1: ########################################################################### | 100%
numpy-base 1.19.1: #################################################################### | 100%
mkl_random 1.1.0: ##################################################################### | 100%
ld_impl_linux-64 2.33.1: ############################################################### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
(Homework4) [prawat@node1775 ~]$ python 1D.py
Time: 0.59281 s
(Homework4) [prawat@node1775 ~]$ python 2D.py
Time: 0.28270 s
(Homework4) [prawat@node1775 ~]$
```

(2) Try different block size, grid size and matrix dimensions. Analyze the running time. If error occurs with some block/grid size, analyze the reason. (5 points)

1D.py

Block Size	Grid Size	Matrix Dimensions	Time
128	1600*1600/128	1600*1600	0.58552 s
512	1600*1600/512	1600*1600	0.58646 s
128	600*600/128	600*600	1.44461 s
512	3000*3000/512	3000*3000	1.44549 s

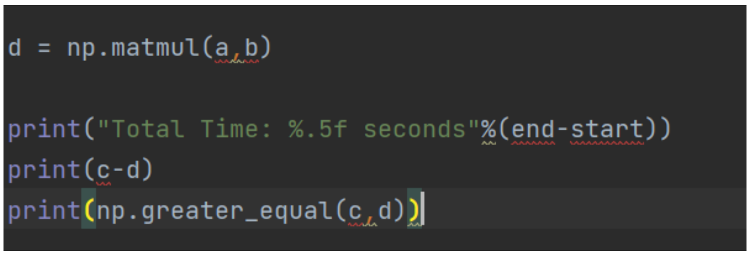
2D.py

Block Size	Grid Size	Matrix Dimensions	Time
32	3000*3000/32	3000*3000	0.28245 s
64	3000*3000/64	3000*3000	1.89878 s
32	800*800/32	800*800	1.89175 s
64	800*800/64	800*800	0.03623 s

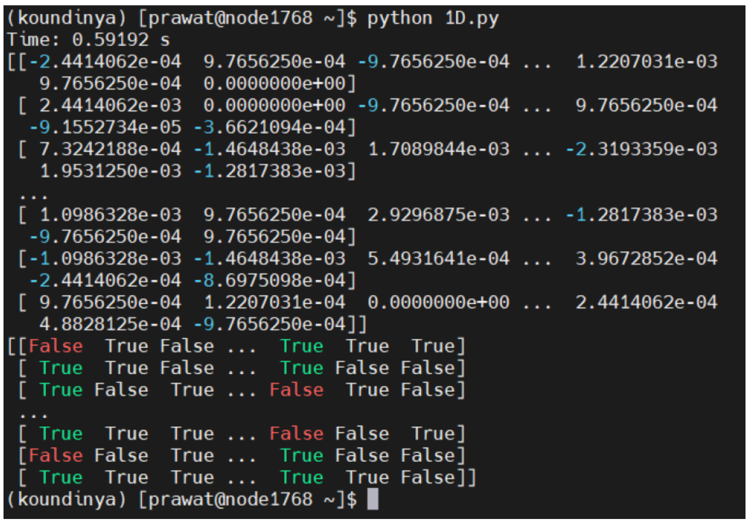
From the result, we can infer that the block size is limited by the selection of hardware. In other words, the total number of running threads is determined by choice of the GPU. For instance, if larger block size is defined, the GPU will reach its maximum capacity and will not be able to process the threads.

(3) Implement a CPU based matrix multiplication using programming language you prefer, compare it with the above GPU sample implementation. Check if the product is the same as GPU implementation, if not, compute and analyze the difference. (15 points)

To perform the matrix multiplication using CPU, I have used NumPy library and made the following changes.

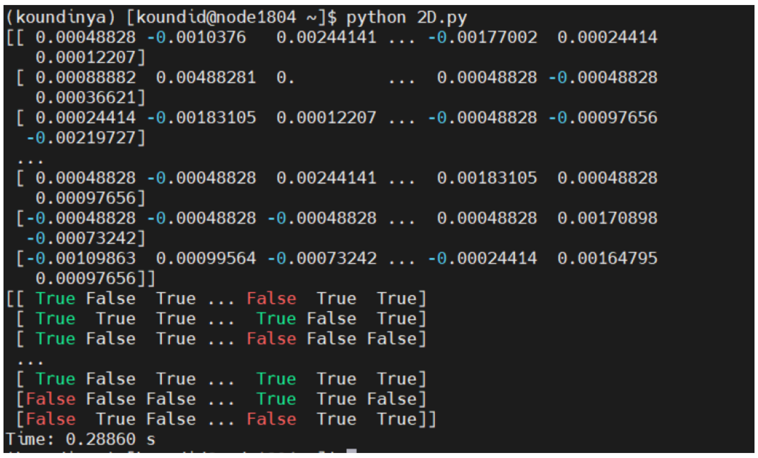


```
d = np.matmul(a,b)

print("Total Time: %.5f seconds"%(end-start))
print(c-d)
print(np.greater_equal(c,d))
```



```
(koundinya) [prawat@node1768 ~]$ python 1D.py
Time: 0.59192 s
[[-2.4414062e-04  9.7656250e-04 -9.7656250e-04 ...  1.2207031e-03
   9.7656250e-04  0.0000000e+00]
 [ 2.4414062e-03  0.0000000e+00 -9.7656250e-04 ...  9.7656250e-04
  -9.1552734e-05 -3.6621094e-04]
 [ 7.3242188e-04 -1.4648438e-03  1.7089844e-03 ... -2.3193359e-03
   1.9531250e-03 -1.2817383e-03]
 ...
 [ 1.0986328e-03  9.7656250e-04  2.9296875e-03 ... -1.2817383e-03
  -9.7656250e-04  9.7656250e-04]
 [-1.0986328e-03 -1.4648438e-03  5.4931641e-04 ...  3.9672852e-04
  -2.4414062e-04 -8.6975098e-04]
 [ 9.7656250e-04  1.2207031e-04  0.0000000e+00 ...  2.4414062e-04
   4.8828125e-04 -9.7656250e-04]]
[[False  True False ...  True  True  True]
 [ True  True False ...  True False False]
 [ True False  True ... False  True False]
 ...
 [ True  True  True ... False False  True]
 [False False  True ...  True False False]
 [ True  True  True ...  True  True False]]
(koundinya) [prawat@node1768 ~]$
```

The above results are for the 1D.py code. The first result in the matrix shows the error between both calculations. While the second result shows which element is greater in value than the other corresponding element.



```
(koundinya) [koundid@node1804 ~]$ python 2D.py
[[ 0.00048828 -0.0010376   0.00244141 ... -0.00177002  0.00024414
   0.00012207]
 [ 0.00088882  0.00488281  0.         ...  0.00048828 -0.00048828
   0.00036621]
 [ 0.00024414 -0.00183105  0.00012207 ... -0.00048828 -0.00097656
  -0.00219727]
 ...
 [ 0.00048828 -0.00048828  0.00244141 ...  0.00183105  0.00048828
   0.00097656]
 [-0.00048828 -0.00048828 -0.00048828 ...  0.00048828  0.00170898
  -0.00073242]
 [-0.00109863  0.00099564 -0.00073242 ... -0.00024414  0.00164795
   0.00097656]]
[[ True False  True ... False  True  True]
 [ True  True  True ...  True False  True]
 [ True False  True ... False False False]
 ...
 [ True False  True ...  True  True  True]
 [False False False ...  True  True False]
 [False  True False ... False  True  True]]
Time: 0.28860 s
```

This difference is mainly because of the approximation in values by CPU. Whereas, the GPU outputs are much more precise, since the GPU does not round off values like the CPU. This result shows that such errors can multiply manifold the CPU is subjected to large calculations (as seen in this case).

(4) Analyze the difference between 1D.py and 2D.py, why 2D.py is quicker than 1D.py? (5 points)

Thread distribution is the primary difference between 1D.py and 2D.py. It is 1-dimensional in the 1D.py code and 2-dimensional in the 2D.py code. This the reason why 2D.py can do a greater number of parallel tasks compared to 1D.py and is thus quicker than latter.

(5) Survey and describe some methods to further speedup the 2D.py (5 points)

To further speedup 2D.py, we can use libraries like tensor flow. The matrix multiplication of tensor flow can improve the speed. On top of that, defining global variables instead of local variables can affect the speed positively. Last but not the least, threading can be optimized further to improve the efficiency of the code.

<u>Question 2</u>

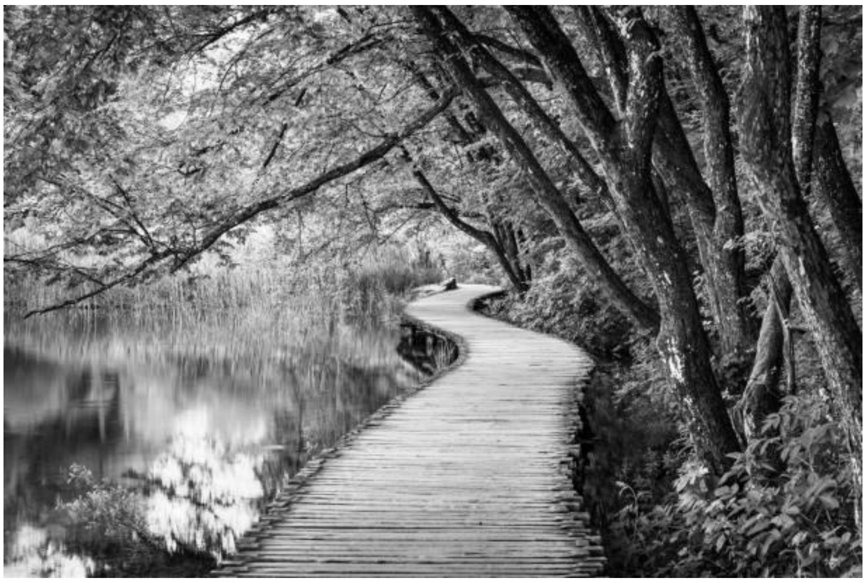
(1) Refer to the sample code in Q3 folder, complete it by importing libraries and implementing your main function. Run and analyze the behavior of the code. (15 points) You can refer to the readme.md.

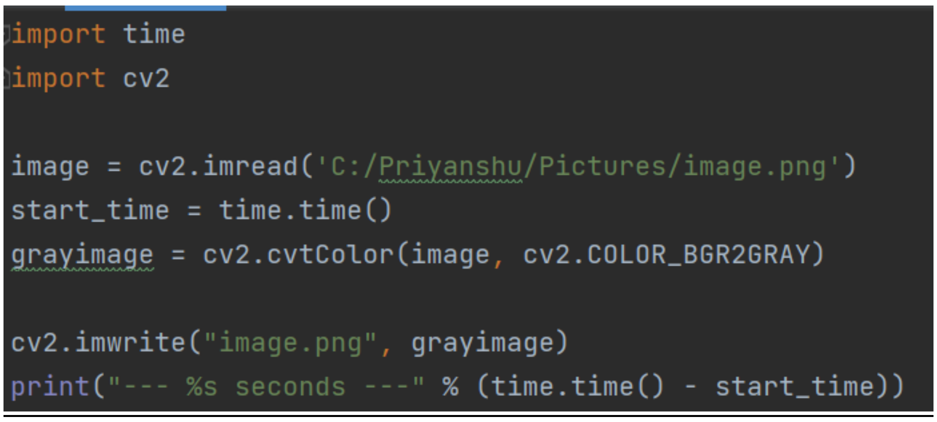
<u>Initial Image</u> -

<u>Final Image</u> **-**

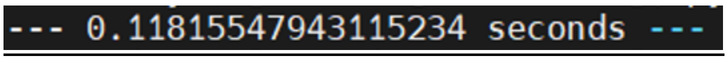
(2) Write a Python code using CPU to mimic the behavior of the above GPU code, compare and analyze the result and speed. You can use third party library such as the OpenCV. (15 points)

CPU Implementation (using OpenCV library)-



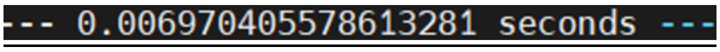


```
import time
import cv2

image = cv2.imread('C:/Priyanshu/Pictures/image.png')
start_time = time.time()
grayimage = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

cv2.imwrite("image.png", grayimage)
print("--- %s seconds ---" % (time.time() - start_time))
```



```
--- 0.11815547943115234 seconds ---
```

GPU Implementation-

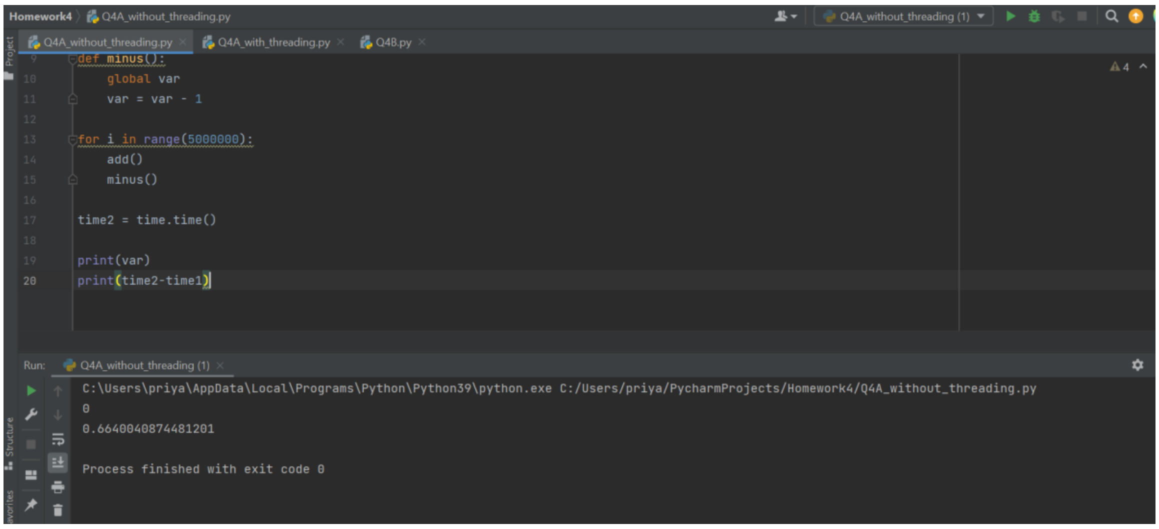


```
--- 0.006970405578613281 seconds ---
```

Question 3.1)

(1) Use "import threading" library to speedup Q4A.py as quick as you can. (10 points) (e.g., you may use threading.Thread() … to create your new thread to do the computing)

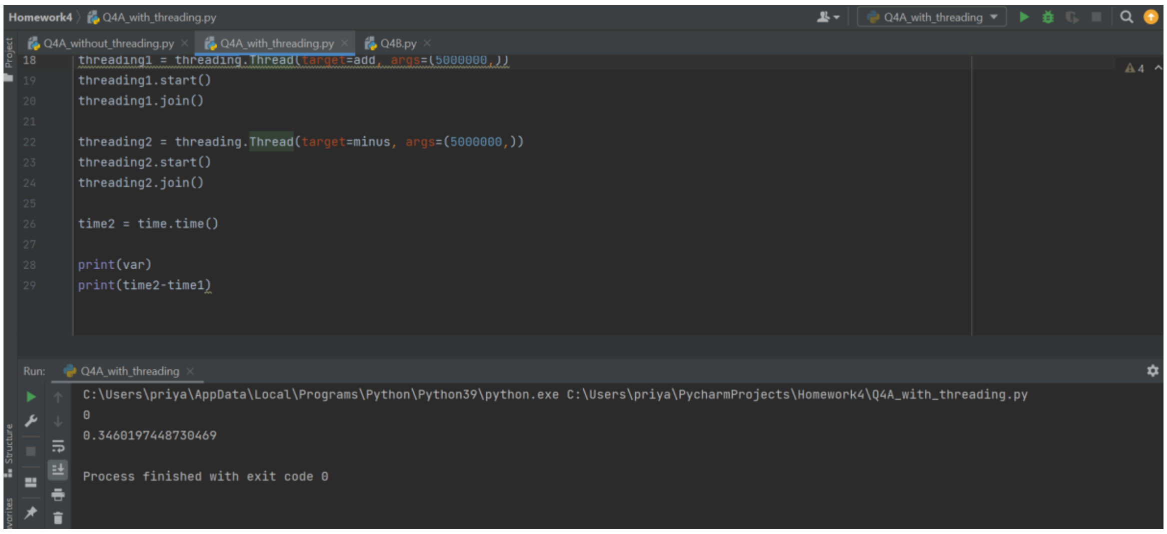
Without Threading



```
def minus():
    global var
    var = var - 1

for i in range(5000000):
    add()
    minus()

time2 = time.time()

print(var)
print(time2-time1)
```

```
C:\Users\priya\AppData\Local\Programs\Python\Python39\python.exe C:/Users/priya/PycharmProjects/Homework4/Q4A_without_threading.py
0
0.6640040874481201

Process finished with exit code 0
```

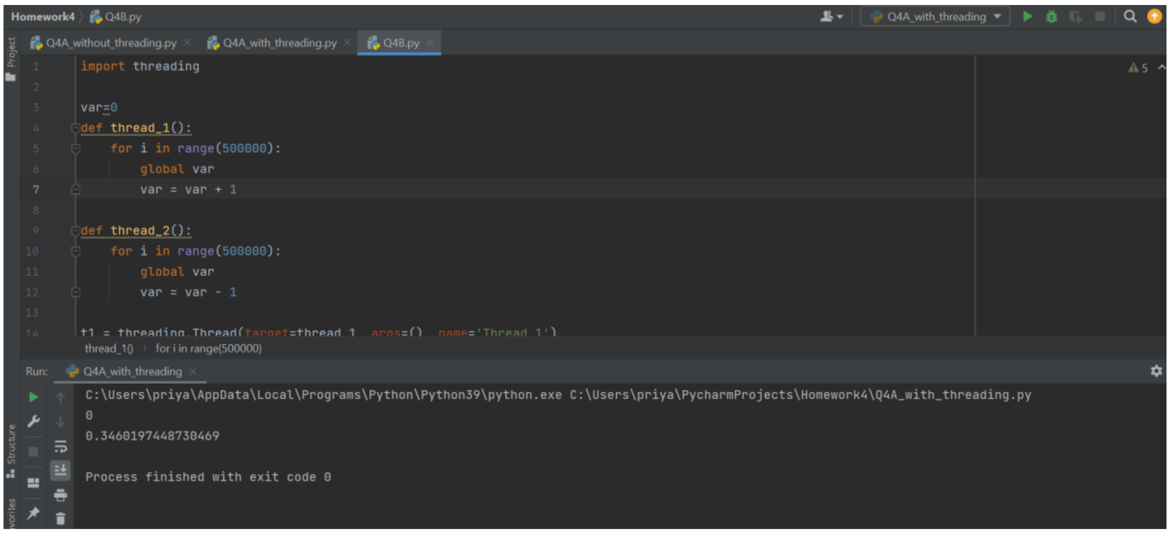
With Threading –



```
threading1 = threading.Thread(target=add, args=(5000000,))
threading1.start()
threading1.join()

threading2 = threading.Thread(target=minus, args=(5000000,))
threading2.start()
threading2.join()

time2 = time.time()

print(var)
print(time2-time1)
```

```
C:\Users\priya\AppData\Local\Programs\Python\Python39\python.exe C:\Users\priya\PycharmProjects\Homework4\Q4A_with_threading.py
0
0.3460197448730469

Process finished with exit code 0
```

(2) Run Q4B.py several times, analyze what’s wrong with it, try to fix it (you may use Lock). (10 points)

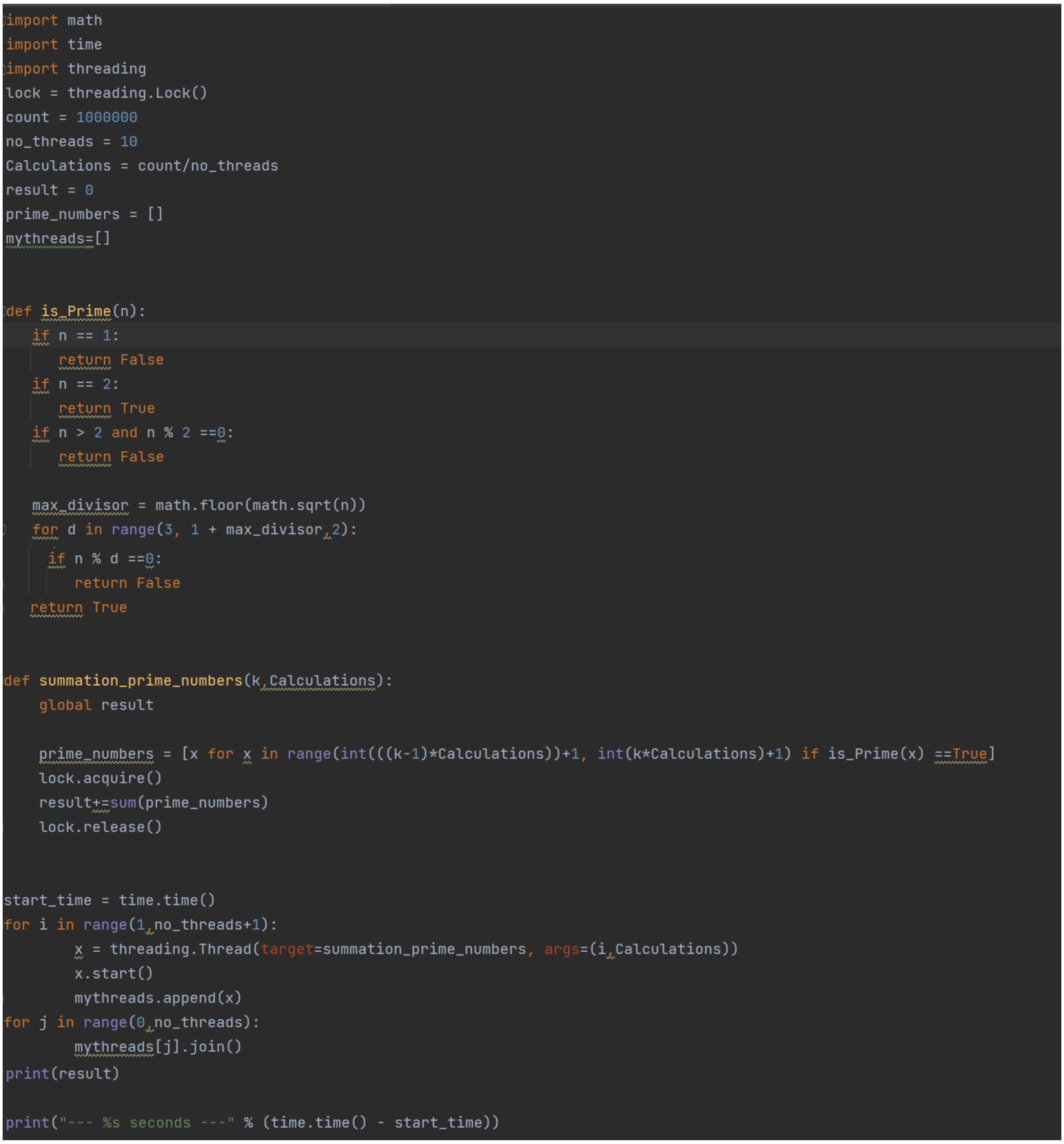


```
import threading

var=0
def thread_1():
    for i in range(500000):
        global var
        var = var + 1

def thread_2():
    for i in range(500000):
        global var
        var = var - 1

t1 = threading.Thread(target=thread_1, args=(), name='Thread_1')
```

```
C:\Users\priya\AppData\Local\Programs\Python\Python39\python.exe C:\Users\priya\PycharmProjects\Homework4\Q4A_with_threading.py
0
0.3460197448730469

Process finished with exit code 0
```

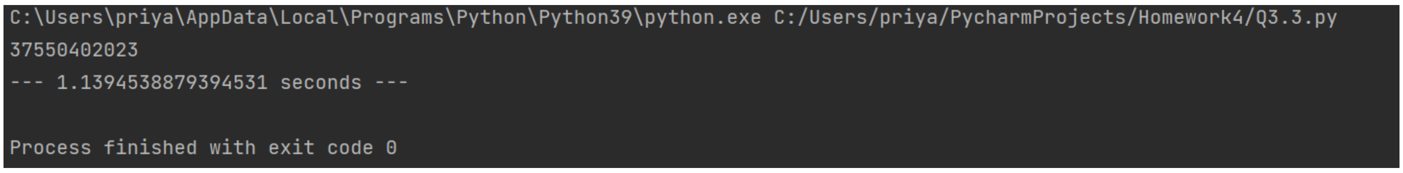
The output of the code should be ‘0’, but this was not the case. This discrepancy in the output occurred as two threads were running simultaneously leading to a situation called ‘racing between threads.’ This problem can be solved by using threading.join() command to join the two threads so both can run synchronously. After making the necessary correction, the result of the code came as expected.

Question 3.3)



```python
import math
import time
import threading
lock = threading.Lock()
count = 1000000
no_threads = 10
Calculations = count/no_threads
result = 0
prime_numbers = []
mythreads=[]


def is_Prime(n):
    if n == 1:
        return False
    if n == 2:
        return True
    if n > 2 and n % 2 ==0:
        return False

    max_divisor = math.floor(math.sqrt(n))
    for d in range(3, 1 + max_divisor,2):
        if n % d ==0:
            return False
    return True


def summation_prime_numbers(k,Calculations):
    global result

    prime_numbers = [x for x in range(int(((k-1)*Calculations))+1, int(k*Calculations)+1) if is_Prime(x) ==True]
    lock.acquire()
    result+=sum(prime_numbers)
    lock.release()


start_time = time.time()
for i in range(1,no_threads+1):
        x = threading.Thread(target=summation_prime_numbers, args=(i,Calculations))
        x.start()
        mythreads.append(x)
for j in range(0,no_threads):
        mythreads[j].join()
print(result)

print("--- %s seconds ---" % (time.time() - start_time))
```

Result -



```
C:\Users\priya\AppData\Local\Programs\Python\Python39\python.exe C:/Users/priya/PycharmProjects/Homework4/Q3.3.py
37550402023
--- 1.1394538879394531 seconds ---

Process finished with exit code 0
```