Stephanie Galvan
Parallel Computing
MW 4:30PM – 5:50PM

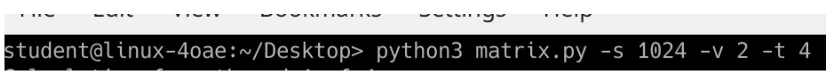
Lab 2: Parallel Matrix Multiplication

Description:

With the help of pyMP library available in python, the following lab aims to run multiplication of two square matrices in parallel. The performance of the program will be tested with 1, 2, 4, and 8 threads. Similar to lab 1, all matrices are assumed to be square and some utility functions were provided to help in the finalization of the program.

Approach:

The given python file contained valid arguments to pass on the terminal (such as size and value for the matrix), so an additional argument was created to represent the num of threads used for parallel. Below is a sample entry for the terminal to run the program:



```
student@linux-4oae:~/Desktop> python3 matrix.py -s 1024 -v 2 -t 4
```

Similar to the previous lab, three nested for loops were used to multiply two given matrices.

Problems/Bugs:

During the first run with pymp, the program took around 9 minutes, but this was later fixed by changing *range* to *p.range*. The first attempt at parallel programming failed due to implementing the wrong loop.

Time Completion:

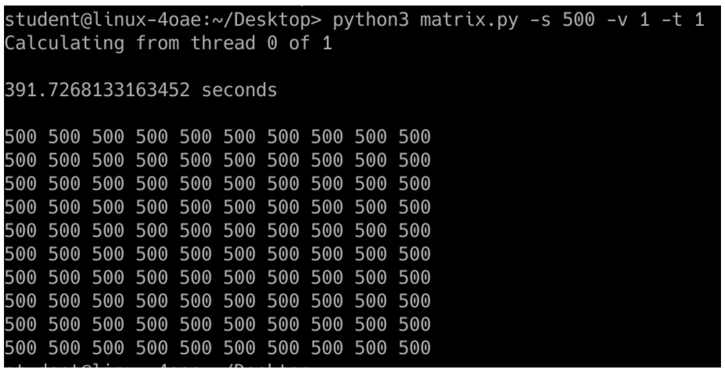
Disregarding the time to read documentation and understand the pymp library, it took around 4 hours to complete this lab. This time includes implementation and testing.

Performance Measurements:

The following tests will use matrices of size 500 and value 1. To compare performance, the tests will use 1, 2, 4, and 8 threads. The results are shown below.

<u>One (1) Thread:</u>

A single thread resulted in a runtime of 391.72 seconds or around 7 minutes.



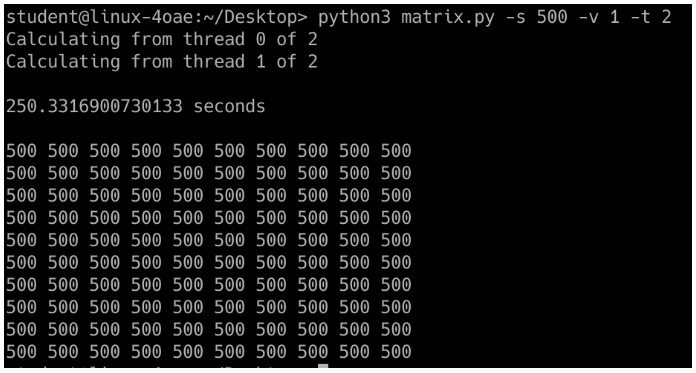
```
student@linux-4oae:~/Desktop> python3 matrix.py -s 500 -v 1 -t 1
Calculating from thread 0 of 1

391.7268133163452 seconds

500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
```

Two (2) Threads:

Two threads resulted in a runtime of 250.3 seconds or around 4 minutes.



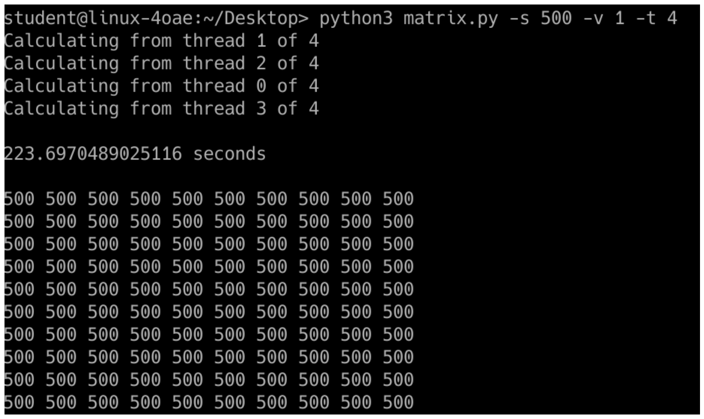
```
student@linux-4oae:~/Desktop> python3 matrix.py -s 500 -v 1 -t 2
Calculating from thread 0 of 2
Calculating from thread 1 of 2

250.3316900730133 seconds

500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
```

Four (4) Threads:

Four threads resulted in a runtime of 223.7 seconds or around 3 minutes.



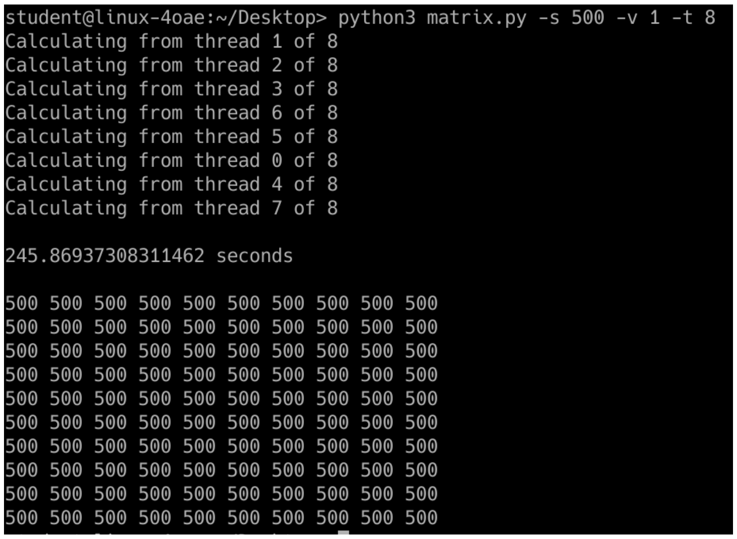
```
student@linux-4oae:~/Desktop> python3 matrix.py -s 500 -v 1 -t 4
Calculating from thread 1 of 4
Calculating from thread 2 of 4
Calculating from thread 0 of 4
Calculating from thread 3 of 4

223.6970489025116 seconds

500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
```

Eight (8) Threads:

Eight threads resulted in a runtime of 245 seconds or around 4 minutes.



```
student@linux-4oae:~/Desktop> python3 matrix.py -s 500 -v 1 -t 8
Calculating from thread 1 of 8
Calculating from thread 2 of 8
Calculating from thread 3 of 8
Calculating from thread 6 of 8
Calculating from thread 5 of 8
Calculating from thread 0 of 8
Calculating from thread 4 of 8
Calculating from thread 7 of 8

245.86937308311462 seconds

500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
500 500 500 500 500 500 500 500 500 500
```

Analysis:

As demonstrated on the test results, an increasing number of threads resulted in a decreasing runtime. This happened because multithreading allows the program to run faster because processes are divided between threads. However, there are some cases in which regardless of the number of threads, the runtime will remain the same (will not be faster) because processes are completed but there are still some threads running.