Daniela Garcia Hernandez

**Serial Matrix Multiply:**

When running the serial version of the matrix multiply python file, it produced the following results when running it with inputs of the following sizes:

On a pixelbook(2 cores):

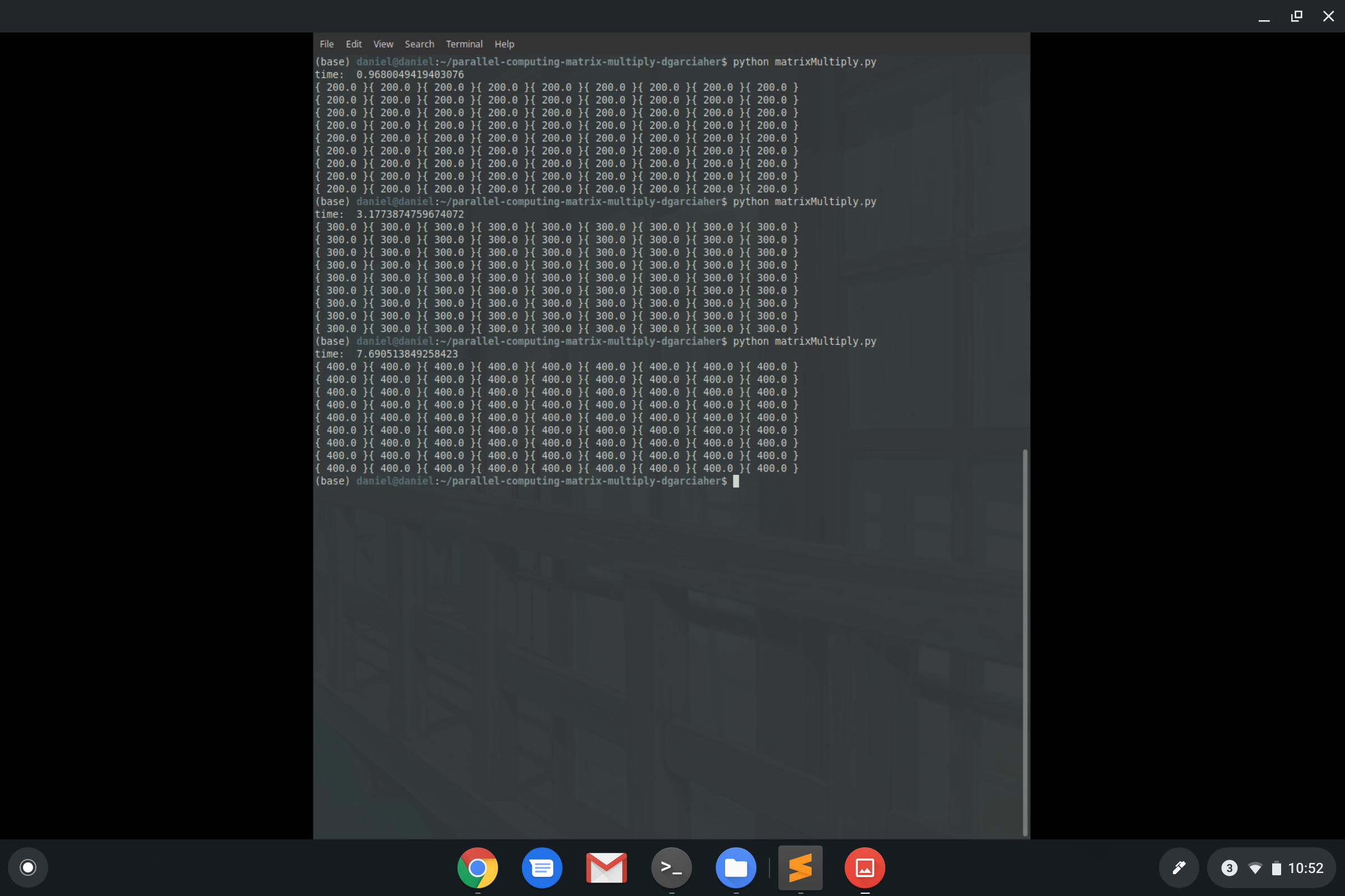
Size 100: 1.35 seconds

Size 150: 4.22 seconds

Size 200: 10.86 seconds

On a desktop(8 cores):

100, 150, and 200 threads



Size 100: 0.97 seconds

Size 150: 3.17 seconds

Size 200: 7.7 seconds

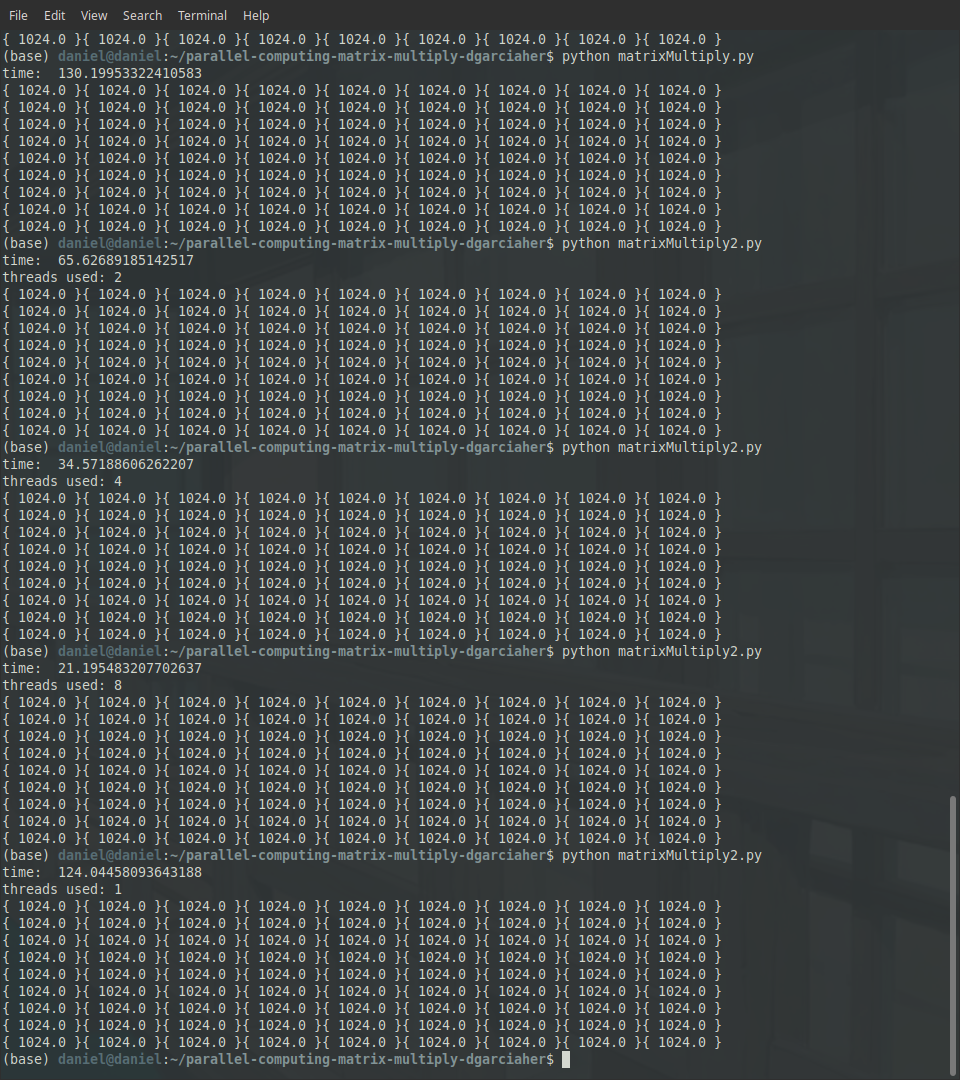
1024: 131 seconds

**Parallel Matrix Multiply:**

When running the matrix version of the matrix multiply python file, it produced the following results when running it with inputs of the following sizes:

1024:

* 1 thread: 124.04 seconds
* 2 threads: 65.2 seconds
* 4 threads: 34.6 seconds
* 8 threads: 21.2 seconds



Some of the problems I encountered when writing this program were that   
I had not programmed in python in over a year, so it took a bit of time to get used to it again, something else I struggled with was figuring out how to split the work to the different threads without making it slower rather than faster.

It took me a while to complete the assignment, especially since I wasn’t getting the results I thought I should be getting on my computer(it only has 2 cores) so I had to ask a friend to help me run it on his computer.

The parallel program performed at almost the same as the serial when run with only 1 thread, however, as you added more threads the speed decreased exponentially. The program behaves like this because the work is essentially being split into smaller chunks and all being worked on at the same time and put together at the end, which makes the computation a lot faster.