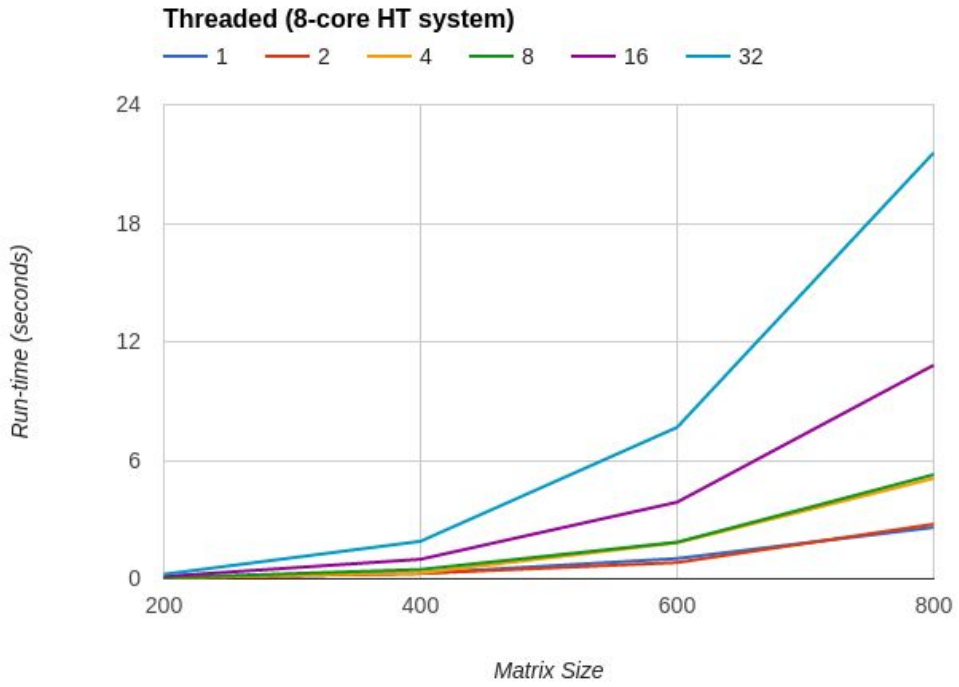
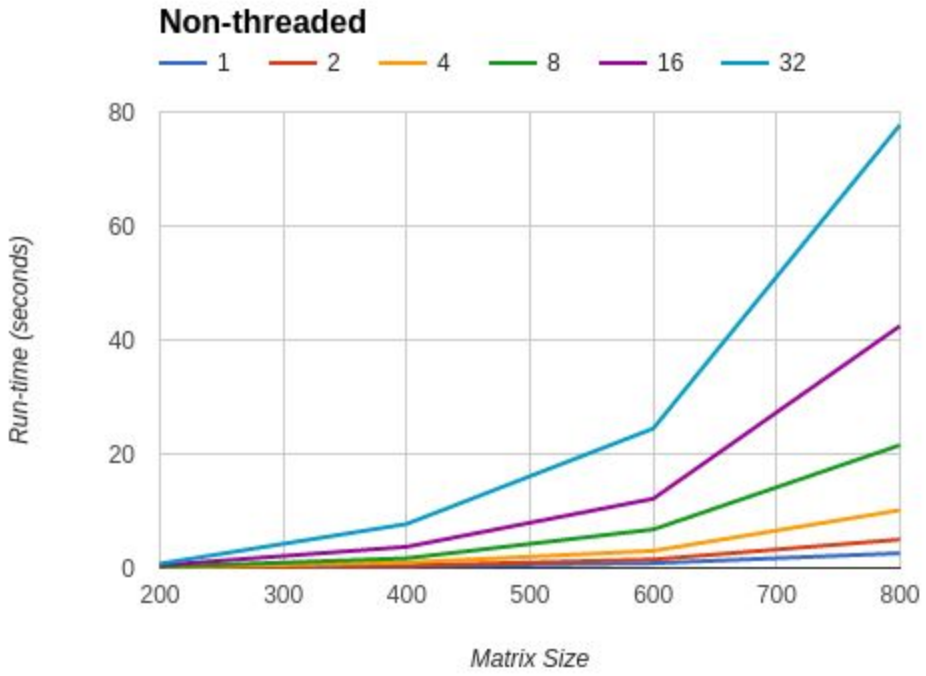
Shawn Jones
CSCI 322, HW2
Matrix multiplication
Performance comparison and code questions

Threaded	1	2	4	8	16	32
200	0.0496931	0.050591	0.0507219	0.0557611	0.122183	0.230155
400	0.276725	0.271976	0.295257	0.471816	0.989623	1.89888
600	1.0283	0.82365	1.82829	1.85417	3.87914	7.66805
800	2.60448	2.76613	5.08493	5.2749	10.8154	21.5703



Non-threaded	1	2	4	8	16	32
200	0.0509269	0.0922449	0.136677	0.194187	0.477206	0.751923
400	0.28855	0.438091	0.959512	1.70241	3.71488	7.71784
600	0.838922	1.49113	3.0122	6.76155	12.1489	24.4693
800	2.58905	4.9915	10.1347	21.5483	42.4622	77.7313



1. Threading did improve the performance of my program. Some cases where I would not expect to see performance gains from threading include:

    - there isn't a large data set involved, and a thread is created to perform a highly trivial operation (a small for loop for example)
    - there is a high amount of dependency among the compute operations, such as operation 1's output is operation 2's input, and operation 2's output is operation 3's input, etc.
    - the program has a high cache-reuse ratio in non-threaded use, the architecture uses a cache that the cores share, and the programmer divides up the data set and uses threads to operate on each subset of the data, resulting in a much lower cache-reuse ratio

2. Once the number of threads exceeded the number of processors, the benefit of adding more threads disappeared. For example, the system executing my code had 8 processors, and doubling the number of threads did not double the execution time until jumping from 8 to 16 threads. When moving from 8 to 16 threads the execution time doubled, and it doubled yet again when moving from 16 to 32 threads.