For this assignment, I created a bash script to help test my program with several different combinations of parameters – such as the number of threads used, and the input size of the array being worked on. The bash script runs the program with each input size separately with 1 thread set, then repeats the same process over, but with 2 threads set, and finally with 4 threads set. To run the script, simply enter in the following command: **./script**

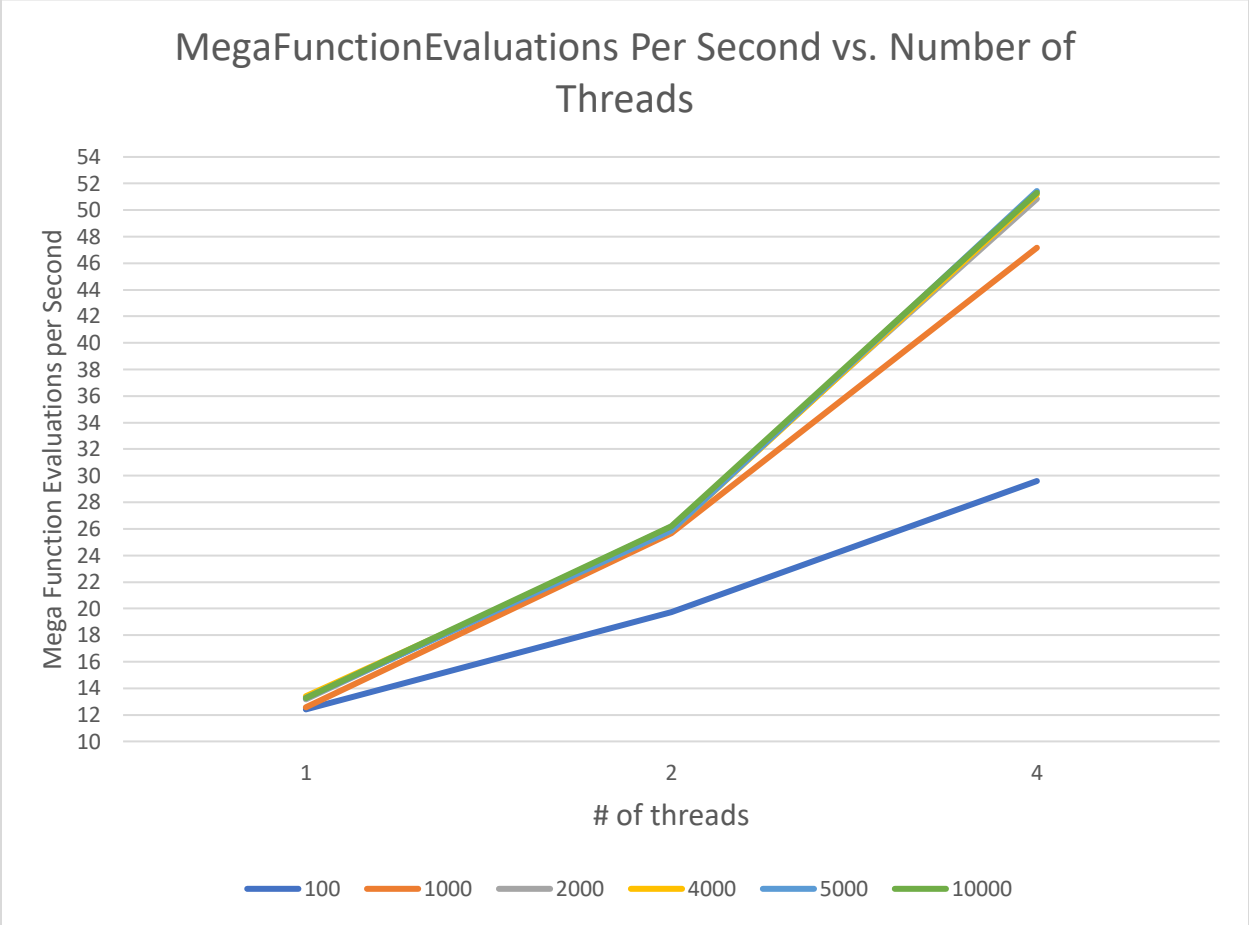
1. Tell what machine you ran this on
   - A Linux system on OSU's FLIP server was used to run Project #0.

2. What do you think the actual volume is?

I think that the actual volume is around 25.31 since the volume results I got from using different combinations of nodes and threads was only different with the digits past the hundredths decimal point.

3. Show the performances you achieved in tables and graphs as a function of NUMNODES and NUMT.
   - **Performance** (Mega Function Evaluations Per Second)

Performance	**1-Thread**	**2-Threads**	**4-Threads**
Input size: 100	12.409356	19.728255	29.601519
Input size: 1000	12.56757	25.692681	47.16299
Input size: 2000	13.164303	26.163943	50.846645
Input size: 4000	13.39726	25.87759	51.185389
Input size: 5000	13.258619	25.87461	51.434914
Input size: 10000	13.263885	26.202353	51.302217



- **Elapsed Time** (seconds)

Elapsed Time (seconds)	**1-Thread**	**2-Threads**	**4-Threads**
Input size: 100	0.00080584	0.00050689	0.00033782
Input size: 1000	0.07756988	0.03892159	0.02120307
Input size: 2000	0.30385201	0.15288215	0.07866792
Input size: 4000	1.19427406	0.6182956	0.3125892
Input size: 5000	1.88556586	0.96619813	0.48605116
Input size: 10000	7.53926927	3.81645112	1.9492335

4. What patterns are you seeing in the speeds?

Speedup	**2-To-1 Thread Speedup**	**4-To-1 Thread Speedup**
Input size: 100	1.589772929	2.38541235
Input size: 1000	1.99297819	3.658426822
Input size: 2000	1.987491738	3.862464013
Input size: 4000	1.9315584	3.820586444
Input size: 5000	1.951531266	3.879356774
Input size: 10000	1.975465959	3.867812281

The patterns I'm seeing is that the speedup is greater for input sizes greater than 100, and as you use threads for the same input size, the speedup also increases.

5. Why do you think it is behaving this way?

As you use increase the number of threads used in your program, the overall performance of your program also increases for data sets of the same size. This is because the equation for calculating performance is measured by the amount of work executed in your program divided by the amount of time it took to do that work. So, as you add more threads to tackle a problem with the same input size, the execution time also goes down which causes the overall performance to increase – which can be seen with all input sizes tested.

In the graph, you'll also see small subdivisions (small input sizes) at the bottom – which indicates low performance, and large subdivision (large input sizes) at the top – which indicates high performance. This shows that the speedup is higher for input sizes greater than 100, since small input sizes are not suitable for multithreading.

6. What is the Parallel Fraction for this application, using the Inverse Amdahl equation?

Parallel Fraction	**1-To-2 Threads Parallel Fraction**	**1-To-4 Threads Parallel Fraction**
Input size: 100	0.741958701	0.774380357
Input size: 1000	0.996476725	0.968877955
Input size: 2000	0.993706509	0.988130549
Input size: 4000	0.964566642	0.98434677
Input size: 5000	0.975163742	0.989633743
Input size: 10000	0.987580631	0.988607882

7. Given that Parallel Fraction, what is the maximum speed-up you could ever get?

The maximum speed-up you could ever get is listed below for all input sizes tested.

Maximum Speedup	**1-To-2 Threads**	**1-To-4 Threads**
Input size: 100	3.875348658	4.432238193
Input size: 1000	283.8268569	32.13156412
Input size: 2000	158.8943152	84.24989209
Input size: 4000	28.22199374	63.88457804
Input size: 5000	40.2637146	96.46683611
Input size: 10000	80.51938617	87.7799915