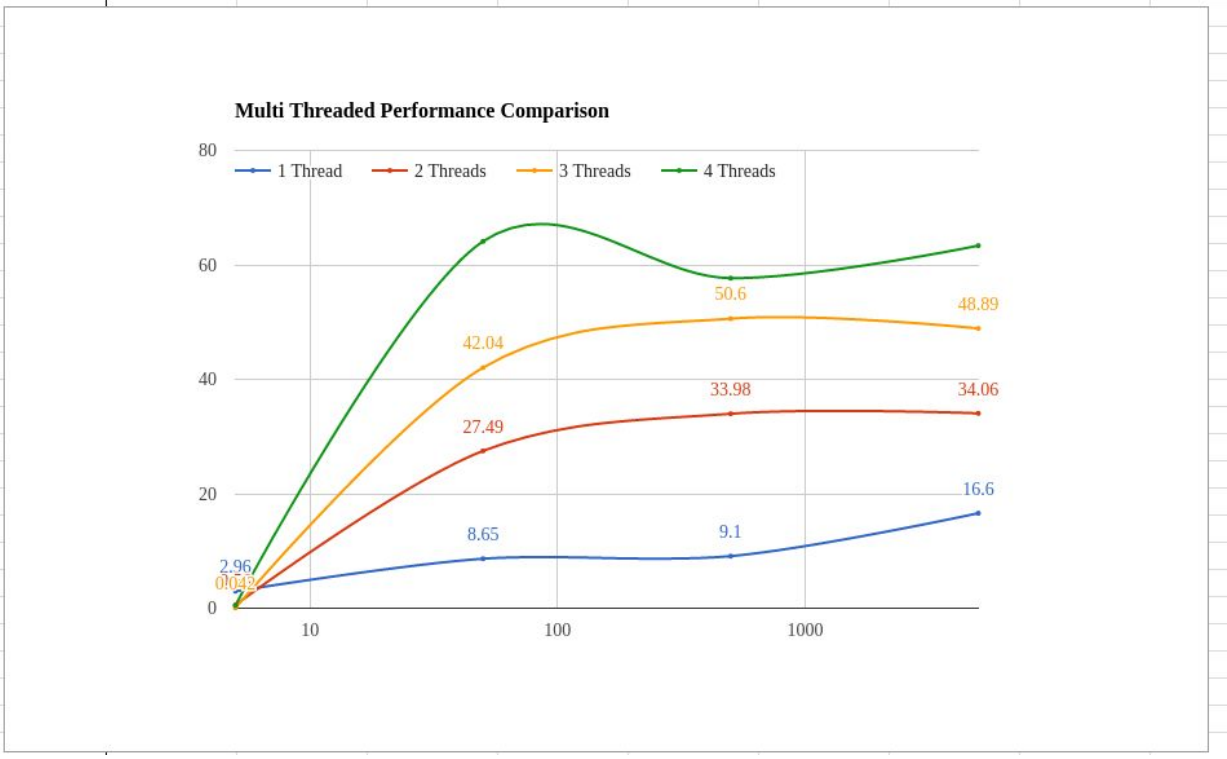
Cody Malick
Project 1
CS 475

1. I ran this on my home desktop, which is running an Intel Sandy Bridge processor with four cores, no hyper threading. It has 8 Gigabytes of RAM.
2. After several runs, and with an accuracy of up to 5000 sections, I believe the actual volume of the shape to be ~14 -> ~14.3
3. Here is a screen capture of the performance of the different runs. It is a log scaled graph, each line is a set of runs with x amount of cores. The X axis is number of sections, 5→ 5000, and the Y axis is MegaMults.

	1 Thread	2 Threads	3 Threads	4 Threads
5	2.96	0.56	0.042	0.49
50	8.65	27.49	42.04	64.11
500	9.1	33.98	50.6	57.68
5000	16.6	34.06	48.89	63.36



4. Each set of threads does next to nothing with the initial run size (5), but as the problem set grows, it really starts to show the advantage of having multiple threads running through the program. I've also noticed that you reach the peak increased performance fairly quickly (~100 sections), and maintains that average performance.
5. I believe it behaves this way because the cores get sufficient advantage taken of them once the problem size is large enough.
6. I would say the program is roughly 80% F(parallel) and 20% F(Sequential), which yields the following fraction:

$$Speedup = \frac{1}{\frac{.8}{4}+.2} = 2.5$$

7. Given this fraction, maximum speedup we could ever have would be:
   $$max\ Speedup = \frac{1}{F(sequential)} = \frac{1}{.8} = 5$$
   So the maximum speed up we could have if we gave it infinite cores would be about five times increased performance.

   For fun, I computed the F(parallel) given the times I had, and they came out pretty close to the predicted speedup I got from the above fraction!
   $$F_{parallel} = \frac{n}{(n-1)} * \frac{T_1 - T_n}{T_1} = \frac{4}{4-1} * \frac{1.49-.386}{1.49} = 2.222$$