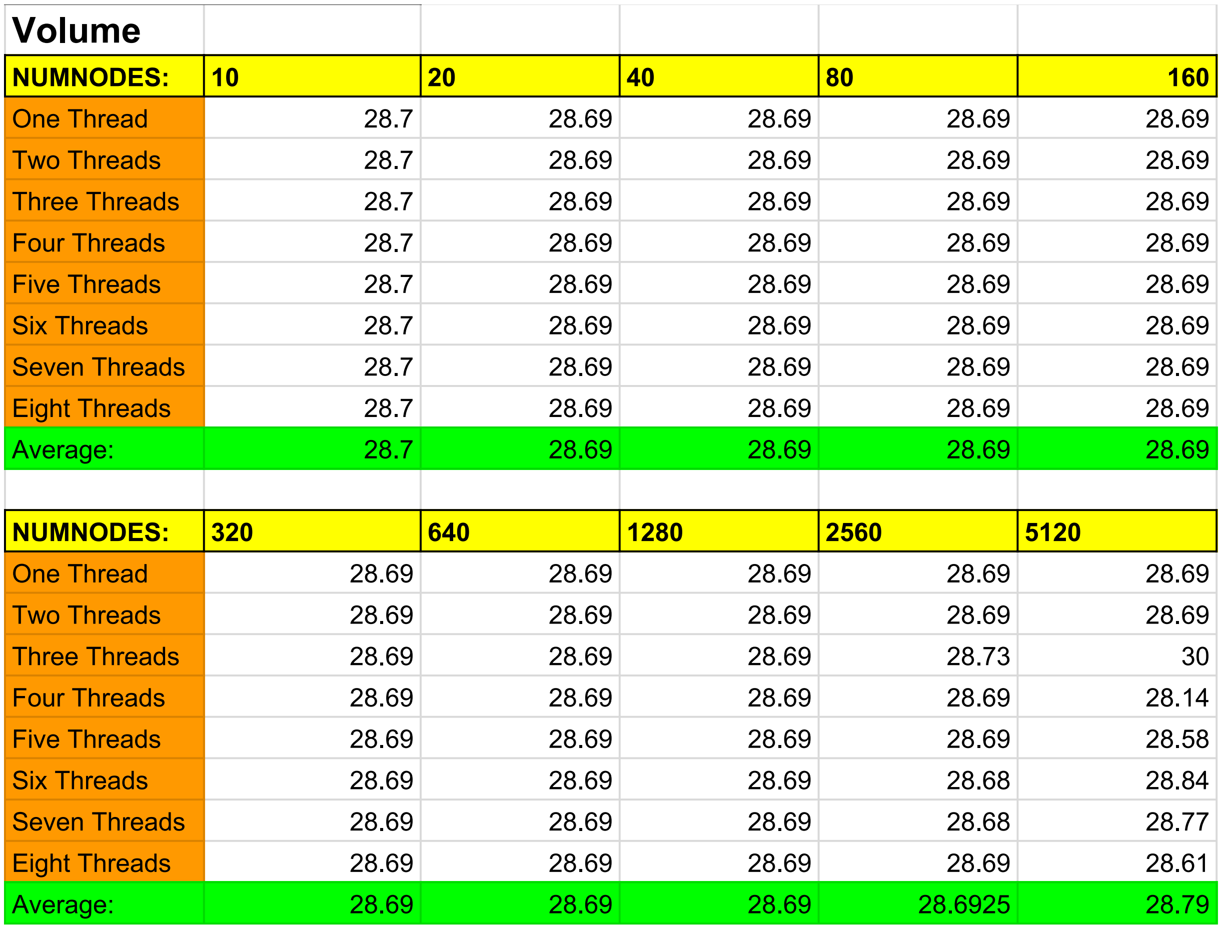
Project 2 Writeup

1. **Estimated Volume = 28.69**



Volume

NUMNODES:	10	20	40	80	160
One Thread	28.7	28.69	28.69	28.69	28.69
Two Threads	28.7	28.69	28.69	28.69	28.69
Three Threads	28.7	28.69	28.69	28.69	28.69
Four Threads	28.7	28.69	28.69	28.69	28.69
Five Threads	28.7	28.69	28.69	28.69	28.69
Six Threads	28.7	28.69	28.69	28.69	28.69
Seven Threads	28.7	28.69	28.69	28.69	28.69
Eight Threads	28.7	28.69	28.69	28.69	28.69
Average:	28.7	28.69	28.69	28.69	28.69

NUMNODES:	320	640	1280	2560	5120
One Thread	28.69	28.69	28.69	28.69	28.69
Two Threads	28.69	28.69	28.69	28.69	28.69
Three Threads	28.69	28.69	28.69	28.73	30
Four Threads	28.69	28.69	28.69	28.69	28.14
Five Threads	28.69	28.69	28.69	28.69	28.58
Six Threads	28.69	28.69	28.69	28.68	28.84
Seven Threads	28.69	28.69	28.69	28.68	28.77
Eight Threads	28.69	28.69	28.69	28.69	28.61
Average:	28.69	28.69	28.69	28.6925	28.79

The higher the NUMNODES, the more accurate the estimate of volume should be in principle since more subdivisions of the floor will better approximate our shape. Interestingly, however, we see a divergence in volume estimates for different threads when NUMNODES = 2560 or 5120.

When testing my code, I found no evidence of indeterministic code that might reveal race-condition failures: for any given NUMT and NUMNODES, the output was always the same. My hypothesis for the diverging estimates for the last two columns is this: higher values for NUMNODES and higher threads involves more floating point additions. In addition, these additions are adding smaller and smaller values since they subdivide the floor into smaller units. The floating point unit under the hood is probably rounding these smaller values in such a way as to produce inconsistent results. Given this inconsistency, I choose to use the earlier estimate for volume of 28.69.

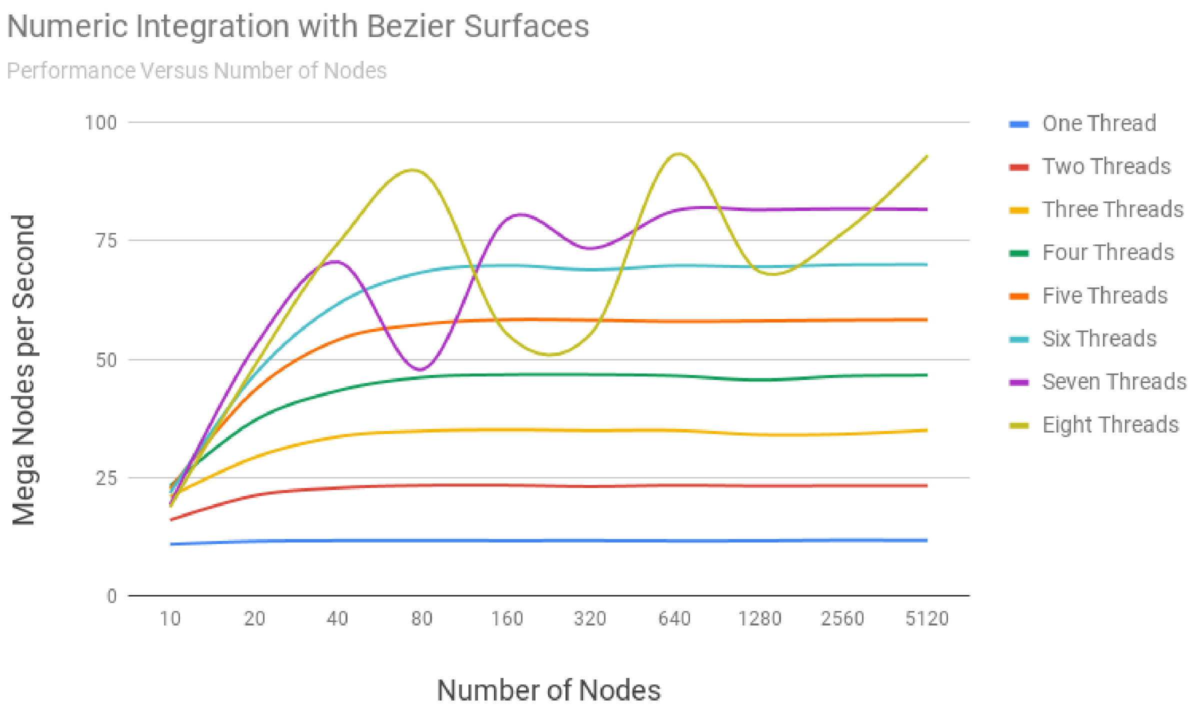
2. **Performance Summary**

I ran my tests on the oregonstate flip3 server. I wrote a shell script to execute my program with $NUMT = \{1...8\}$ and with $NUMNODES = \{10, 20, 40, ...5120\}$.

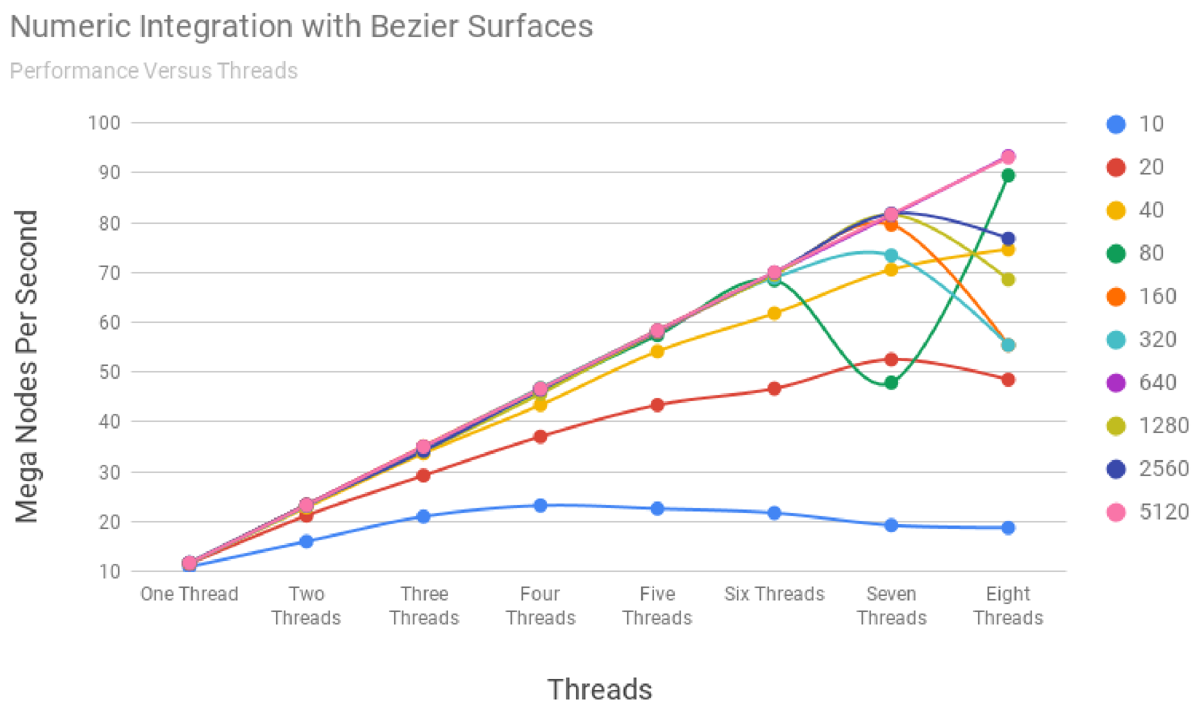
The performance results are shown in the following two tables:

NUMNODES:	10	20	40	80	160
One Thread	10.93	11.56	11.7	11.72	11.69
Two Threads	16.03	21.21	22.84	23.37	23.4
Three Threads	21.02	29.25	33.67	34.85	35.13
Four Threads	23.24	37.05	43.41	46.2	46.77
Five Threads	22.61	43.38	54.15	57.4	58.37
Six Threads	21.74	46.7	61.77	68.42	69.82
Seven Threads	19.28	52.53	70.56	47.88	79.57
Eight Threads	18.77	48.48	74.62	89.45	55.4

NUMNODES:	320	640	1280	2560	5120
One Thread	11.7	11.65	11.67	11.79	11.74
Two Threads	23.14	23.39	23.25	23.29	23.28
Three Threads	34.96	34.98	34.08	34.2	35.03
Four Threads	46.8	46.53	45.63	46.47	46.64
Five Threads	58.28	58.01	58.12	58.28	58.37
Six Threads	68.93	69.79	69.54	69.98	70.04
Seven Threads	73.41	81.42	81.6	81.8	81.68
Eight Threads	55.42	93.31	68.55	76.76	93.06



The only surprising result from this graph is that there is high inconsistency in performance for execution with seven and eight threads. My only current working hypothesis is that since the number of nodes is small, the higher thread executions create higher overhead in delegating fewer and fewer tasks. However, I find this hypothesis unlikely since the actual work to delegate to each thread is equal to $NUMNODES^2/NUMT$, which means that there are still many iterations to delegate to eight threads. Moreover, this hypothesis would explain only a diminishing return and not, per se, the instability we find as the number of nodes increases for seven and eight threads.



A noticable anomoly is the the execution with seven threads when NUMNODES=80. This anomoly is probably due to a small fluctuation in server load during execution of my shell script, since further executions showed no particular drop in performance during that execution. Another interesting finding is that performance tends to more quickly plateau for smaller values of NUMNODES, which makes sense since the overhead for running a multithreaded program will tend against the minor performance gains in delegating a small number of tasks.

3. **Parallel Fraction**

My estimate for the parallel fraction is **0.997**. I used the results from executing the code with eight threads and with 5120 number of nodes to estimate the parallel fraction.

The Inverse Amdahl's Law (i.e. solving Amdahl's for P_f) is the following:

$$F_p = \frac{n}{n-1}(1 - \frac{1}{S})$$

where S is the speedup and n is the number of cores. Let S_1 and S_8 be the speeds of the program executed with one thread and eight threads respectively. From my results, $S_1 = 11.74$ and $S_8 = 93.06$, so the speedup is $\frac{93.06}{11.74} = \mathbf{7.92}$.

Subsituting into the Inverse Amdahl's Law:

$$F_p = \frac{8}{8-1}(1 - \frac{1}{7.92})$$
$$= \mathbf{0.997}$$

4. **Maximum Possible Speedup**

The maximum possible speedup is:

$$F_p = \frac{1}{S_f}$$

where S_f is the sequential portion of the code. The sequential portion is the complement of the parallel fraction:

$$\begin{aligned} F_p &= \frac{1}{1 - 0.997} \\ &= \mathbf{333.33} \end{aligned}$$