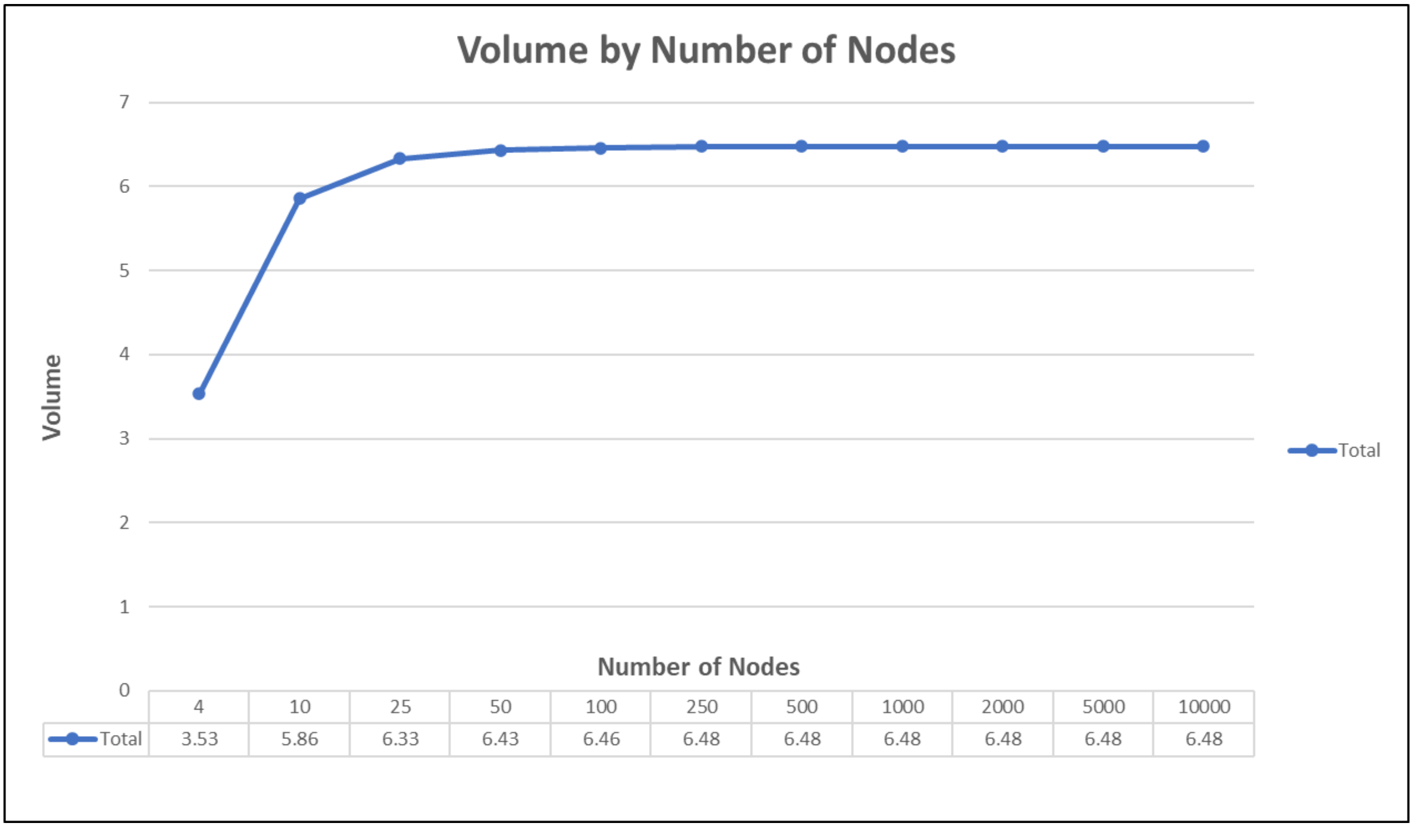
Clinton Hawkes
hawkesc@oregonstate.edu
CS475-400
Project #2

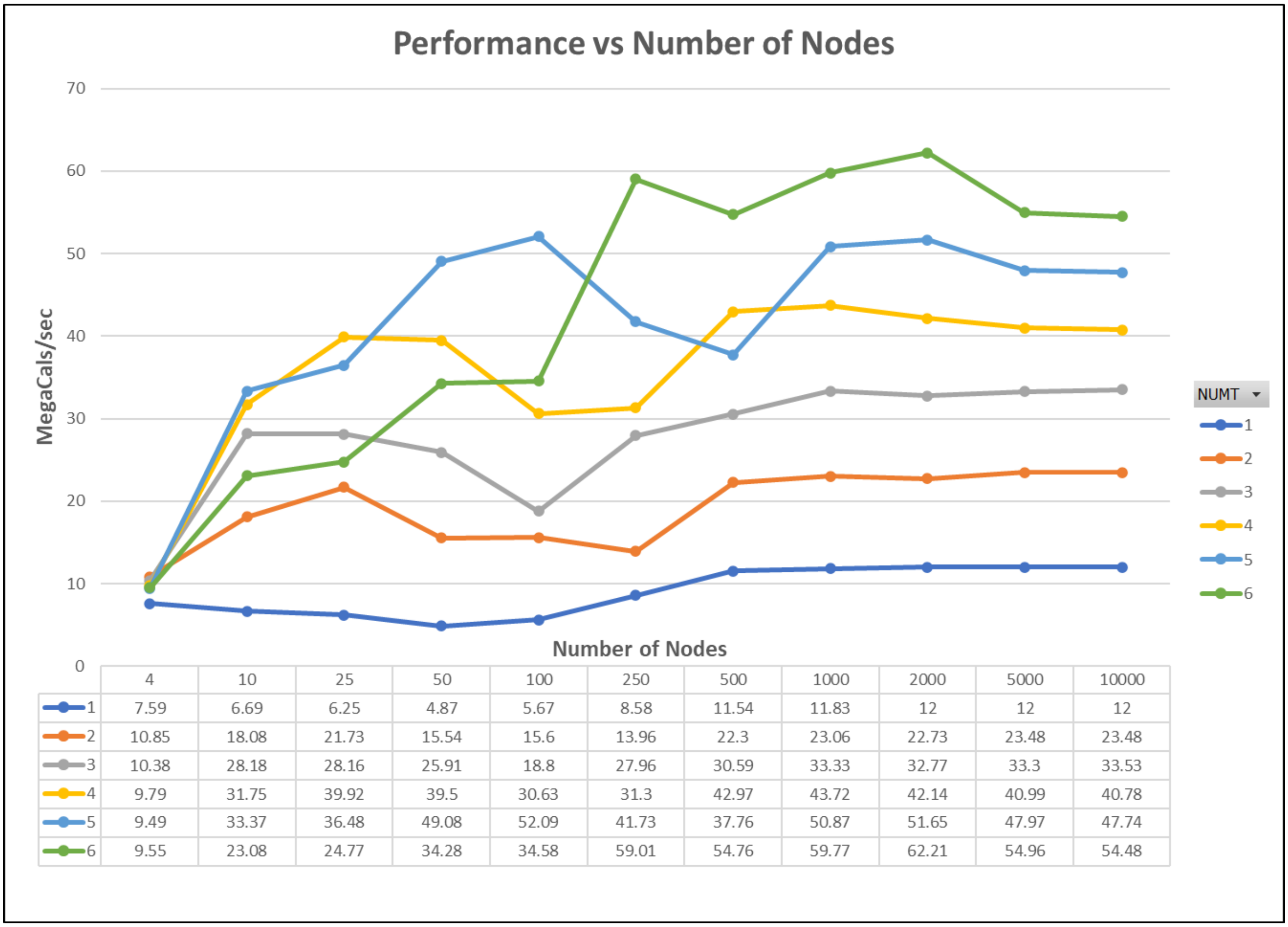
1. I ran this program on my laptop:
   Lenovo X1 Extreme
   Intel Core i7 9750H (6 cores/12 threads)
   32GB Ram
   Nvidia GTX 1650 max q
   Linux kernel 5.5.9

2. I found the volume of the superquadratic to be 6.48 units^3. The volume grew in size with each increase in the NUMNODES value from 4 to 250. After NUMNODES = 250, the volume measurement remained constant for the remaining increases to 10,000. Not surprisingly, this was the case for every NUMT value.



	4	10	25	50	100	250	500	1000	2000	5000	10000
Total	3.53	5.86	6.33	6.43	6.46	6.48	6.48	6.48	6.48	6.48	6.48

3. Performance related to the number of nodes for each thread count:

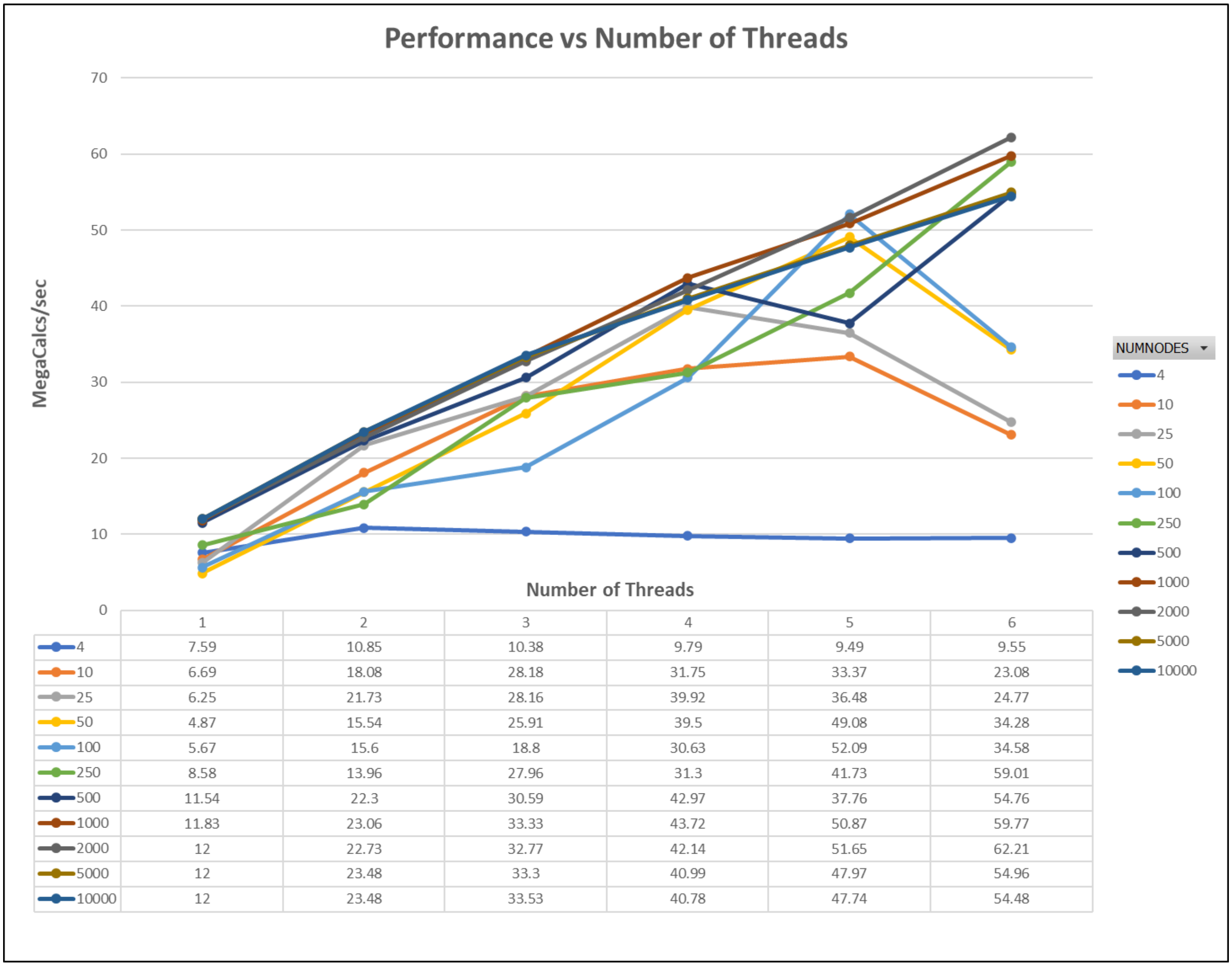


	4	10	25	50	100	250	500	1000	2000	5000	10000
1	7.59	6.69	6.25	4.87	5.67	8.58	11.54	11.83	12	12	12
2	10.85	18.08	21.73	15.54	15.6	13.96	22.3	23.06	22.73	23.48	23.48
3	10.38	28.18	28.16	25.91	18.8	27.96	30.59	33.33	32.77	33.3	33.53
4	9.79	31.75	39.92	39.5	30.63	31.3	42.97	43.72	42.14	40.99	40.78
5	9.49	33.37	36.48	49.08	52.09	41.73	37.76	50.87	51.65	47.97	47.74
6	9.55	23.08	24.77	34.28	34.58	59.01	54.76	59.77	62.21	54.96	54.48

Overall, this graph showed an increase in the performance as the number of threads increased. This is was the behavior I expected. I also observed that lower values of NUMNODES produced some unexpected behavior. The performance increased as NUMNODES value increase to 25. Performance then decreased as NUMNODES value increased to 250. Performance then increased again when NUMNODES value increased from 250 to 2,000. This is where performance peaked for most of the thread counts. Performance then slowly decreased after NUMNODES = 2,000. We learned that as the number of calculations increases, we may see a slow down due to the calculations being faster than the time it takes to fetch the values from memory. This may be what is happening here.

In general, we can conclude from the graph above that performance will increase as the number of threads increase, and max performance is reached when NUMNODES is 1,000-2,000. This seems to hold true for all thread counts. This is a generalization and may not hold true for every test run. This is what I observed on my personal laptop, which was not busy with other processes while performing the test.

4. Performance related to the number of threads for each value of NUMNODES:



	1	2	3	4	5	6
4	7.59	10.85	10.38	9.79	9.49	9.55
10	6.69	18.08	28.18	31.75	33.37	23.08
25	6.25	21.73	28.16	39.92	36.48	24.77
50	4.87	15.54	25.91	39.5	49.08	34.28
100	5.67	15.6	18.8	30.63	52.09	34.58
250	8.58	13.96	27.96	31.3	41.73	59.01
500	11.54	22.3	30.59	42.97	37.76	54.76
1000	11.83	23.06	33.33	43.72	50.87	59.77
2000	12	22.73	32.77	42.14	51.65	62.21
5000	12	23.48	33.3	40.99	47.97	54.96
10000	12	23.48	33.53	40.78	47.74	54.48

It is a little difficult to see which lines are on top, so I will tell you that they are the lines representing NUMNODES equal to 1,000 and 2,000. This matches my observation of the Performance vs Num Nodes graph above in section 3. Performance increased until it reached the max performance at NUMNODES = 2,000, and then performance slowly decreases thereafter.

The graph above shows somewhat sporadic performance figures for the lines that represent NUMNODES in the 4 – 100 range. This is most likely due to the system spending a greater percentage of time for thread setup when NUMNODES is small, while the percentage of time spent for thread setup is less when the NUMNODE figures are large (there is more time spent on calculations when there are more nodes). This is probably the reason the lines representing the higher NUMNODE values appear “smoother”.

5. Since I observed the greatest performance when NUMNODES was 2,000, I used the performance figures I gathered with NUMT = 6 and NUMNODES = 2,000:

   Speedup = 62.21 / 12 = 5.18

   F_p = (6 / 5) * (1 - (1 / 5.18)) = .97

6. The maximum speedup possible for the program is 33.33.

   $\text{Speedup}_{\text{max}}$ = 1 / (1 - .97) = 33.33