CS 475 Parallel Computing
Dr. Matthew
Student: Khuong Luu
Project 0 – Simple OpenMP Experiment

1. **Tell what machine you ran this on**
The computer I ran this experiment on has:
- Operating System: Ubuntu 16.04
- Kernel: 4.13.0-37-generic
- Processor: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz, 4 cores
- Memory: 16GB, 2400Mhz
- gcc version 5.4.0 20160609, OpenMP 3.1

2. What do you think the actual volume is?

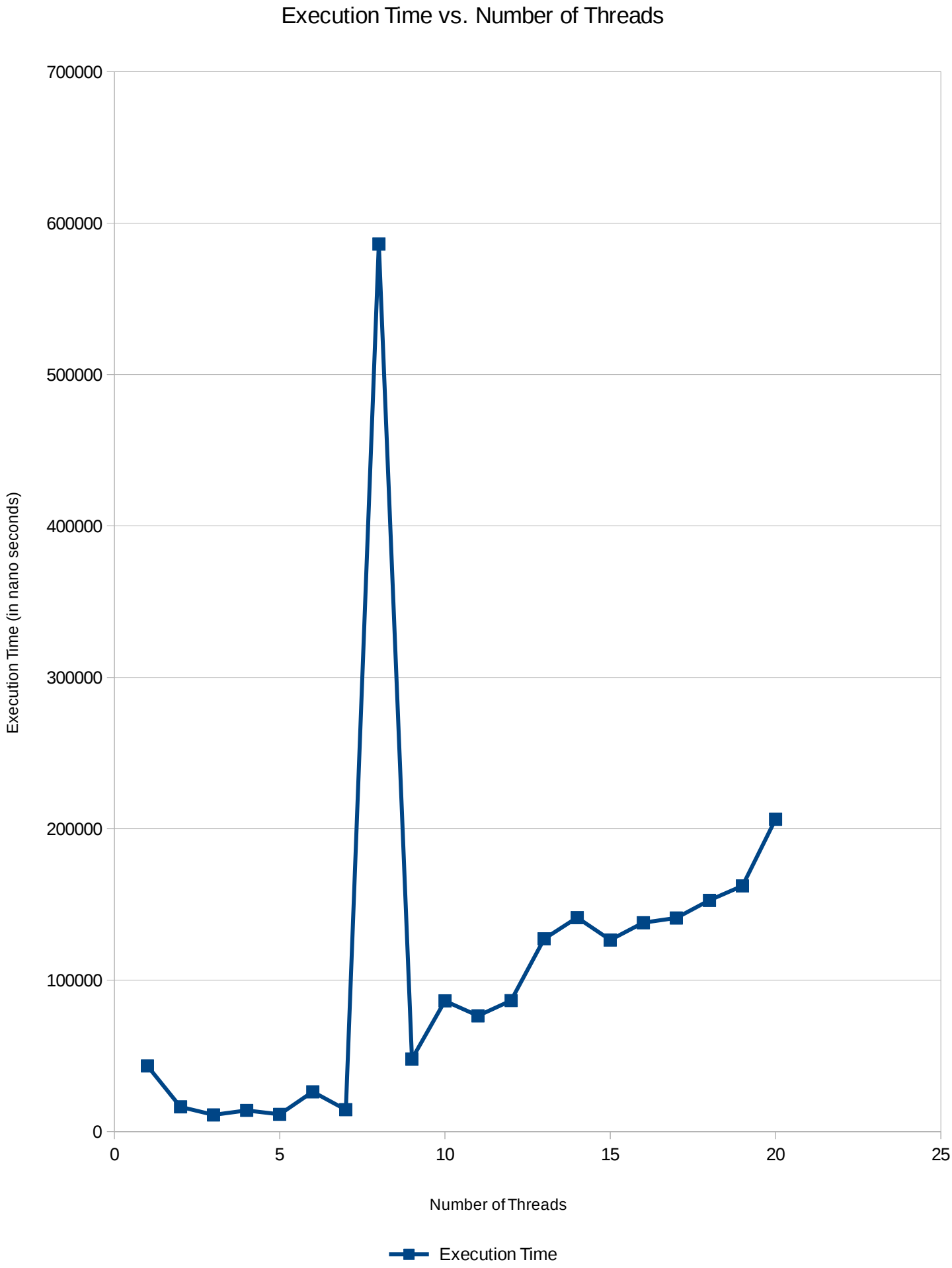
As I increase the number of subdivisions from 2, to 4, to 10, to 100, to 1000, and to 10000. The value of computed volume converges to 25.31
My computing result is in the table below:

Number of subdivisions	Volume
2	27.00
4	26.11
10	25.41
100	25.31
1000	25.31
10000	25.31

3. Show the performances you achieved in tables and graphs as a function of NUMNODES and NUMT

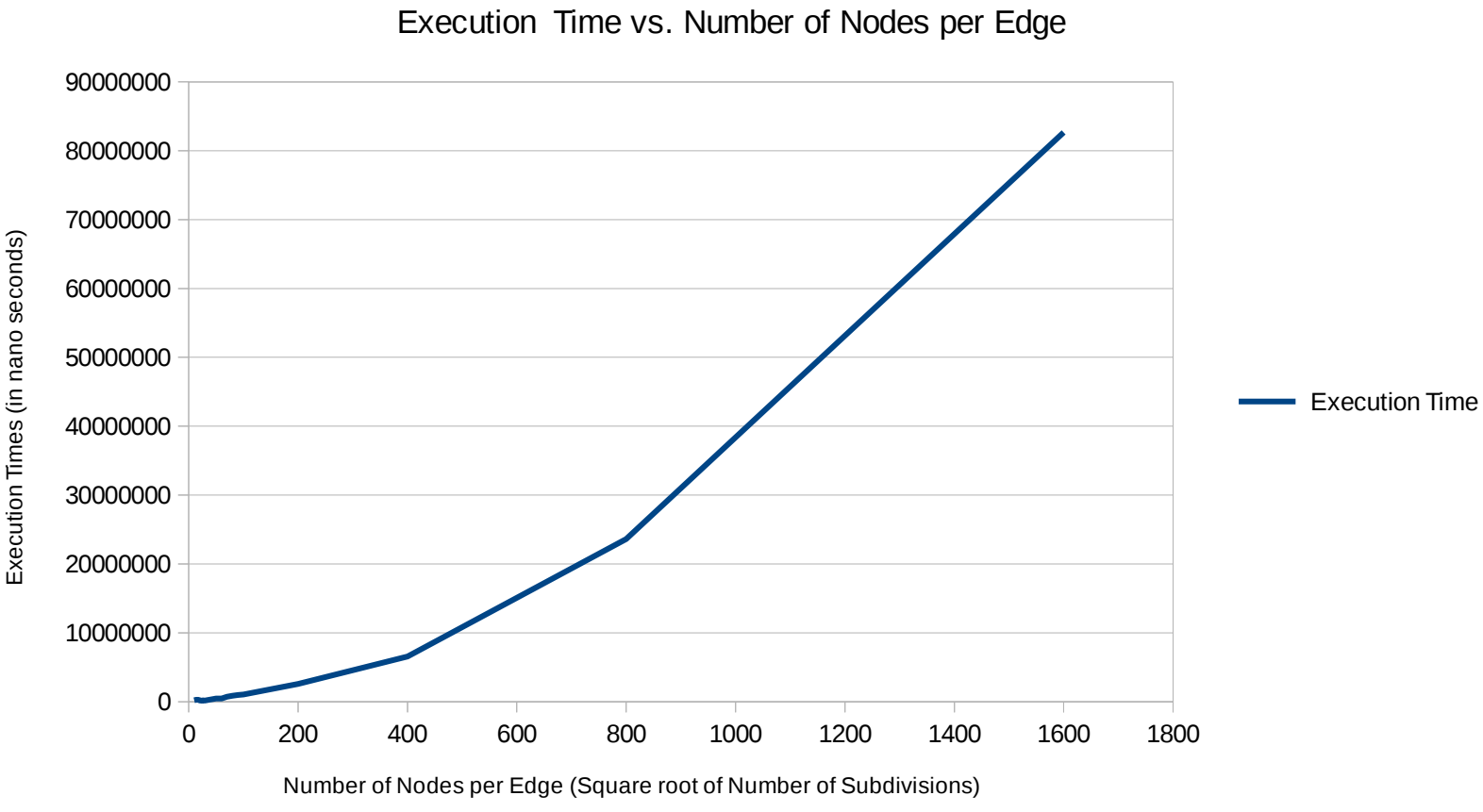
3.1 Execution Time (in nano seconds) vs. NUMT (with NUMNOES = 10)

Number of Threads	Execution Time
1	43395.2
2	16360.1
3	11081.3
4	14005.3
5	11446.8
6	26307.7
7	14496.1
8	586175.1
9	47973
10	86339
11	76527.2
12	86536.8
13	127303.5
14	141334
15	126537
16	137959.7
17	141084.3
18	152769.6
19	162300.4
20	206258.2


Execution Time vs. Number of Threads
700000
600000
500000
400000
300000
200000
100000
0
Execution Time (in nano seconds)
0
5
10
15
20
25
Number of Threads
Execution Time

3.2. Execution Time (in nano seconds) vs. NUMNODES (with NUMT = 10)

Number of Subdivisions	Execution Time
10	189252.4
12	241961.5
14	281486.7
16	296190.5
18	292345.9
20	158501
22	152335
24	126152.8
26	206554.9
28	151913.1
30	145714.1
40	311317.3
50	457529.1
60	456020.4
70	721136.3
80	867229.9
90	975219.1
100	1059217.1
200	2585622.7
400	6575153.8
800	23618236.2
1600	82671239.3



4. What patterns are you seeing in the speeds?

4.1. Performance vs. NUMT

With a fixed number of subdivisions, as the number of threads increases until a certain value num_threads = X, the performance increases gradually (it increase dramatically from 0 threads to 1 threads). Then once the number of threads is greater than X, the performance generally decreases gradually. *There is an extreme outlier at number of threads = 8 where the execution time suddenly spike to a relatively very huge value (15 times bigger than the values around)*

4.2. Performance vs. NUMNODES

With a fixed number of threads, as the number of subdivisions (number of nodes per edge) increases, the performance decreases a bit fairly quickly.

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

5.1. Performance vs. NUMT

The increase in performance from NUMT = 0 to NUMT = 5 is due to the benefits from parallelization. The partials volumes are divided for threads to compute and in the end those partial volume aggregated into the final volume value (thus the keyword reduce(+:volume) in my openmp directive).

The decrease in performance starting from NUMT = 6 is due to the overhead which cause by the problem size for each thread is so small that the cost of overhead from parallelization of that many thread has outweighted the speed up benefit from that of many threads.

However I do not have an explanation for the huge spike at NUMT = 8. This is very interesting. I would love to hear feedback from TA and Instructor about this behavior. I have inspected my code and the way to calculate the data but have not found anything suspicious. Some may think this is just an outlier, but I don't think so because I got the same pattern after over 6 times of samplings. For each sampling, I let the program run 10 times and took the average execution times as the final value.

5.2. Performance vs. NUMNODES

The decrease in performance (or increase in the execution time) as we increase the number of subdivisions (or the number of nodes per edge) is naturally reasonable as it would make more work for each thread to compute as thus the execution time of each thread as well as the final execution time. In exchange as we have better accuracy in volume value.

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

Without losing generality, I take the execution time at NUMT=5 as a reference to calculate speedup and parallel fraction.

So the speedup, S, when moving from 1 threads to 5 threads is:
S = (Execution time with 1 threads) / (Execution time with 5 threads)
  = 43395.2 / 11446.8
  = 3.8

Amdahl equation:
S = 1 / (Fp / n + 1 – Fp)
3.8 = 1 / (Fp / 5 + 1 – Fp)
=> Fp = 0.92

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

$\max(S) = \lim(n \rightarrow +\text{infinity})(S) = 1/F_{\text{sequential}} = 1 / (1 - F_{\text{parallel}}) = 1 / (1 – 0.92) = 12.5$

So the maximum speed-up I could ever get from parallelizing this program is 12.5 times