Xiaoying Li
lixiaoyi@oregonstate.edu
Project #2 Numeric Integration with OpenMP Reduction

- **Machine:**
  I ran the program on OSU Engr Server flip3.

- **The actual volume:**

# of subdivisions / # of threads	16	32	64	128	256	512	1024	2048
1	6.19	6.39	6.45	6.47	6.48	6.48	6.48	6.48
2	6.19	6.39	6.45	6.47	6.48	6.48	6.48	6.48
4	6.19	6.39	6.45	6.47	6.48	6.48	6.48	6.48
8	6.19	6.39	6.45	6.47	6.48	6.48	6.48	6.48

The table above shows the results of total volume my program computed under different number of threads and subdivisions. Therefore, the actual volume≈6.48.

- **Table showing performance (MegaNodes/Seconds) versus subdivisions and threads:**

# of subdivisions / # of threads	16	32	64	128	256	512	1024	2048
1	3.77	3.67	3.61	3.59	3.57	3.56	3.55	3.55
2	7.33	7.29	7.22	7.17	7.12	7.12	7.09	7.09
4	13.22	13.94	14.04	14.02	13.99	13.93	13.91	13.91
8	17.87	20.15	20.94	20.96	21.18	21.51	22.48	23.01

- **Parallel Fraction for this application, using the Inverse Amdahl equation:**
  Use the performance (MegaNodes/Seconds) results shown in last part of the 4 runs of 1, 2, 4, and 8 threads with the maximum number of subdivisions (2048) to calculate Speedup and Fp.

$$Speedup_2 = \frac{7.09}{3.55} \approx 1.997$$

$$Speedup_4 = \frac{13.91}{3.55} \approx 3.918$$

$$Speedup_8 = \frac{23.01}{3.55} \approx 6.482$$

According to the Inverse Amdahl equation $Fp = \frac{n}{n-1} \cdot (1 - \frac{1}{Speedup})$,

$$Fp_2 = \frac{2}{2-1} \cdot (1 - \frac{1}{1.997}) \approx 0.998$$

$$Fp_4 = \frac{4}{4-1} \cdot (1 - \frac{1}{3.918}) \approx 0.993$$

$$Fp_8 = \frac{8}{8-1} \cdot (1 - \frac{1}{6.482}) \approx 0.967$$

And according to formula $\overline{Fp} = \frac{\sum_{i=2}^{N} Fp_i}{N-1}$, $Fp = \frac{0.998+0.993+0.967}{3} \approx 0.99$

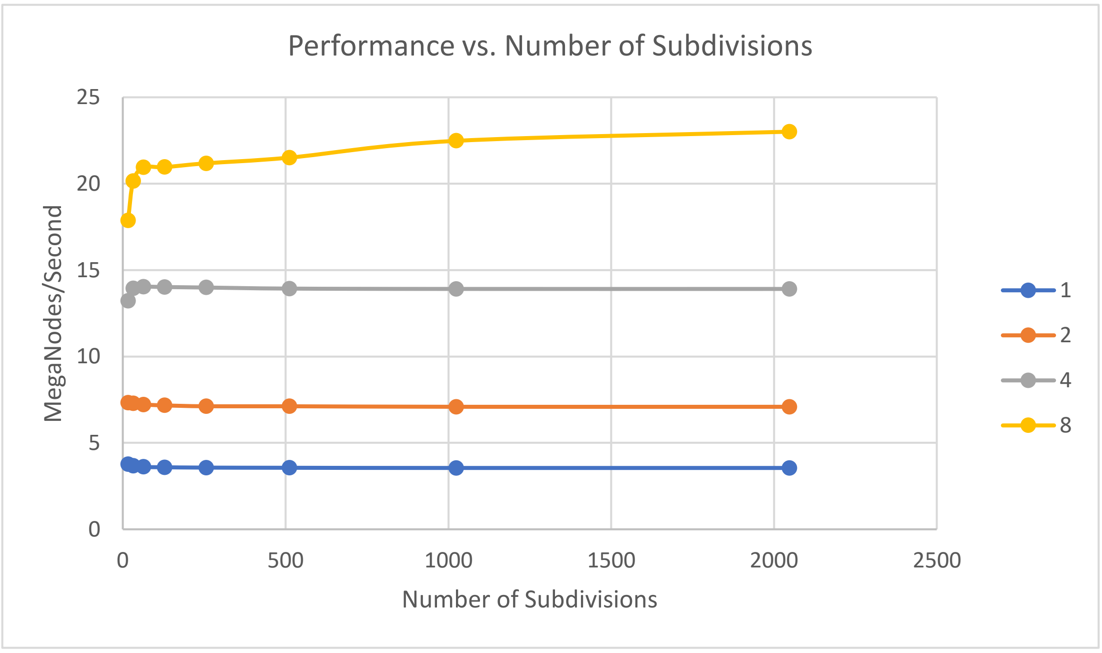
Therefore, this application's Parallel Fraction is 0.99.

- **Given that Parallel Fraction, the maximum speed-up could ever get:**

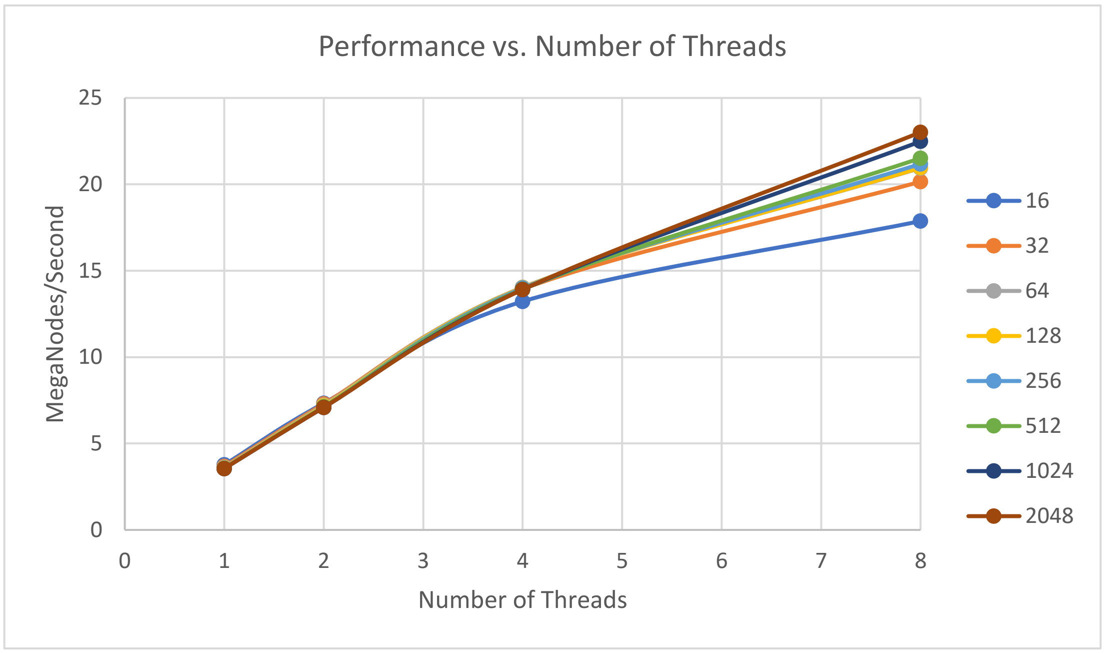
$$max\ Speedup = \frac{1}{1-Fp} = \frac{1}{1-0.99} = 100$$

Therefore, when $Fp = 0.99$, the maximum speed-up is 100.

- **Graph of performance vs. number of subdivisions:**



- **Graph of performance vs. number of threads**



- **Patterns in the speeds and why:**

For the same number of subdivisions, as the number of threads gets larger, the speeds (MegaNodes/Seconds) also gets larger. Because as the number of threads gets larger, the for-loop was distributed to more threads, so more work can be done in the same amount of time and the speeds increase.

For the same number of threads, as the number of subdivisions changes, the corresponding speeds (MegaNodes/Seconds) do not change much. Because there are always some overheads in the threaded code, like thread startup and operation reduction in this program. So, when the number of subdivisions increases, the time spent on these overhead factors also increased, especially for the operation reduction, which lowers the program's speed, leading to similar speed under small data size and large data size.