Amy Stockinger (stockina@oregonstate.edu)
CS 475 Project 2 – Numeric Integration with OpenMP
Due: April 28, 2019

In this program, we are going to compute the volume between two Bezier surfaces. I ran my program on flip3 with uptimes around 4.00. I did not use -O3 when compiling. I also changed every 'float' in my program to be a double.

The program was run once for each of the following combinations:

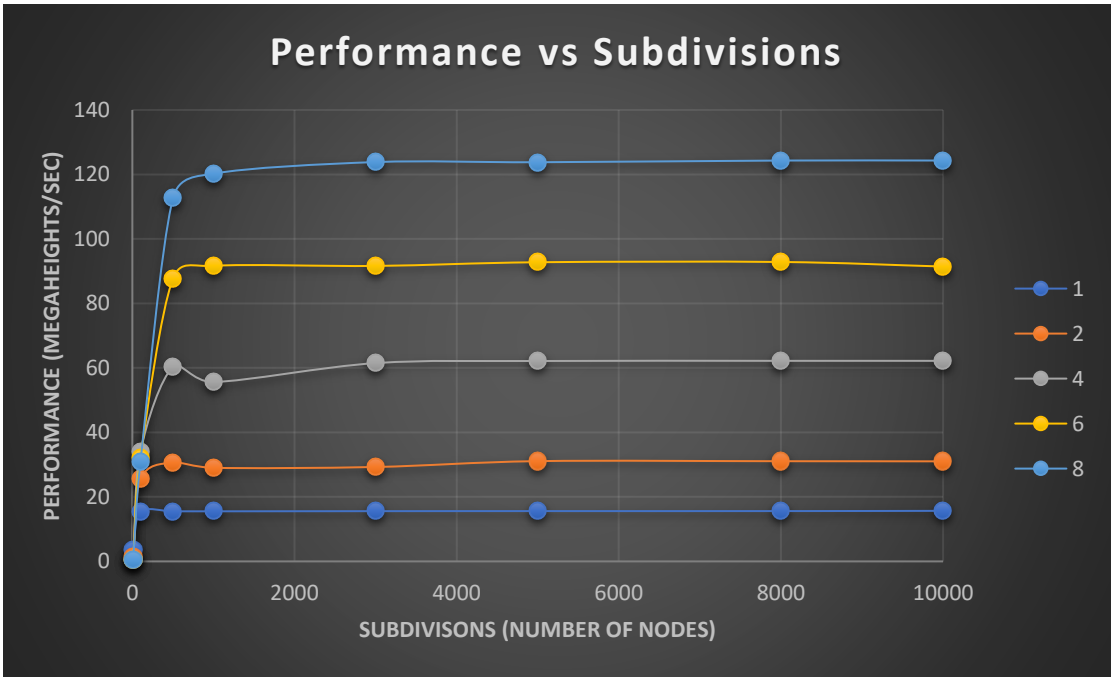
Threads: 1, 2, 4, 6, 8

Subdivisions: 10, 100, 500, 1000, 3000, 5000, 8000, 10000

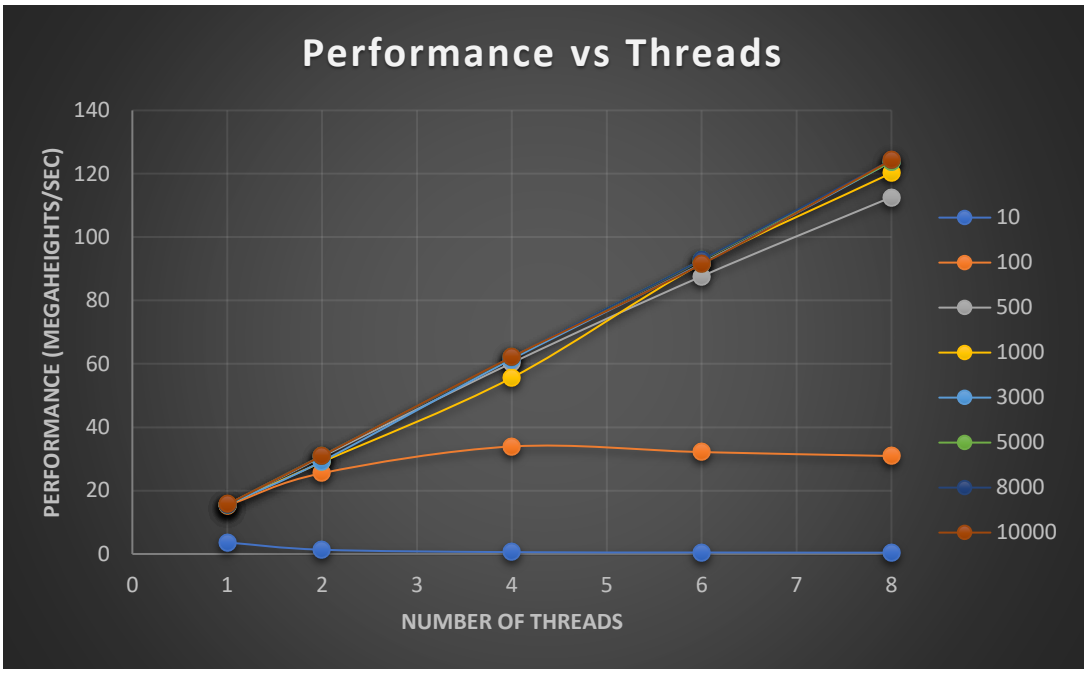
Below are the results:

Calculated Performance for each run

Threads \ Number of Nodes	10	100	500	1000	3000	5000	8000	10000
1	3.63	15.29	15.52	15.53	15.58	15.6	15.6	15.68
2	1.34	25.51	30.7	29.03	29.28	31.09	31.07	31.03
4	0.58	33.93	60.3	55.64	61.49	62.15	62.2	62.2
6	0.47	32.16	87.64	91.65	91.63	92.77	92.85	91.44
8	0.41	30.92	112.6	120.24	123.84	123.79	124.31	124.31



The above graph shows performance vs NUMNODES. The performance stabilizes for almost all numbers of threads around 500-1000 nodes, though some are sooner.



This graph shows performance vs number of threads. There is a clear advantage to having more threads as the number of subdivisions increases. Just like in the performance vs subdivision graph, there is a notable stabilization beginning at 500 nodes.

What do you think the actual volume is?

It is about 28.69.

Parallel Fraction

I used the 10,000 subdivision run performance for these calculations:

2 Threads: $S = P_2/P_1 = (31.03)/(15.68) = 1.98$
$F_P = (2 / 1) * (1 – (1 / 1.98)) = 0.99$
4 Threads: $S = P_4/P_1 = (62.20)/(15.68) = 3.97$
$F_P = (4 / 3) * (1 – (1 / 3.97)) = 1.00$
6 Threads: $S = P_6/P_1 = (91.44)/(15.68) = 5.83$
$F_P = (6 / 5) * (1 – (1 / 5.83)) = 0.99$
8 Threads: $S = P_8/P_1 = (124.31)/(15.68) = 7.93$
$F_P = (8 / 7) * (1 – (1 / 7.93)) = 1.00$

F_p (avg) = (0.99 + 1.00 + 0.99 + 1.00)/4 = 1.00

It seems numeric integration is extremely parallelizable.

Given this parallel fraction, what is the maximum speed-up we could *ever* get?

Since 100% of the program is parallelizable, 0% is sequential.
maxSpeedup = $1/(1 – F_P) = 1/(1 - 1) = 1/0$ = the limit of 1/n as n approaches 0 is infinity.
In theory, we could get higher and higher speedup with more threads/processors.