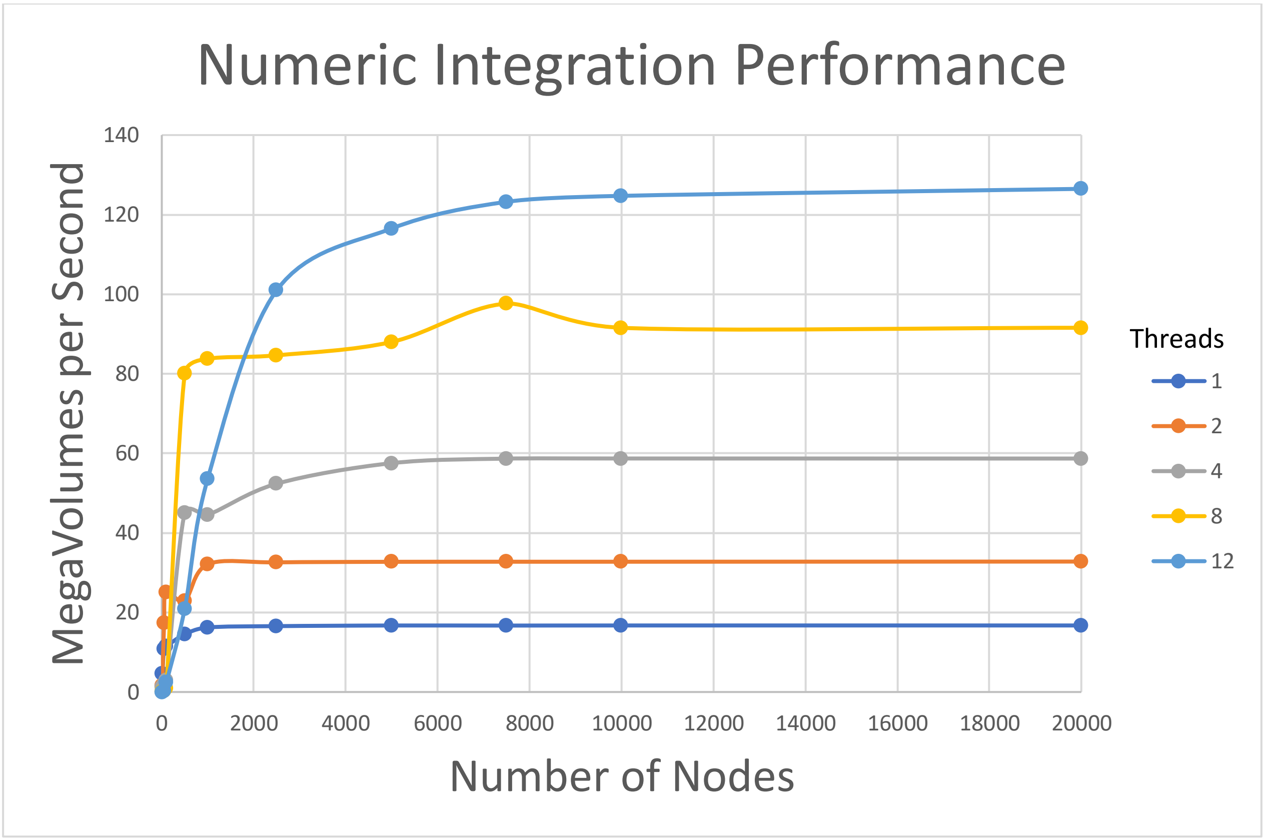
Project 2: Numeric Integration

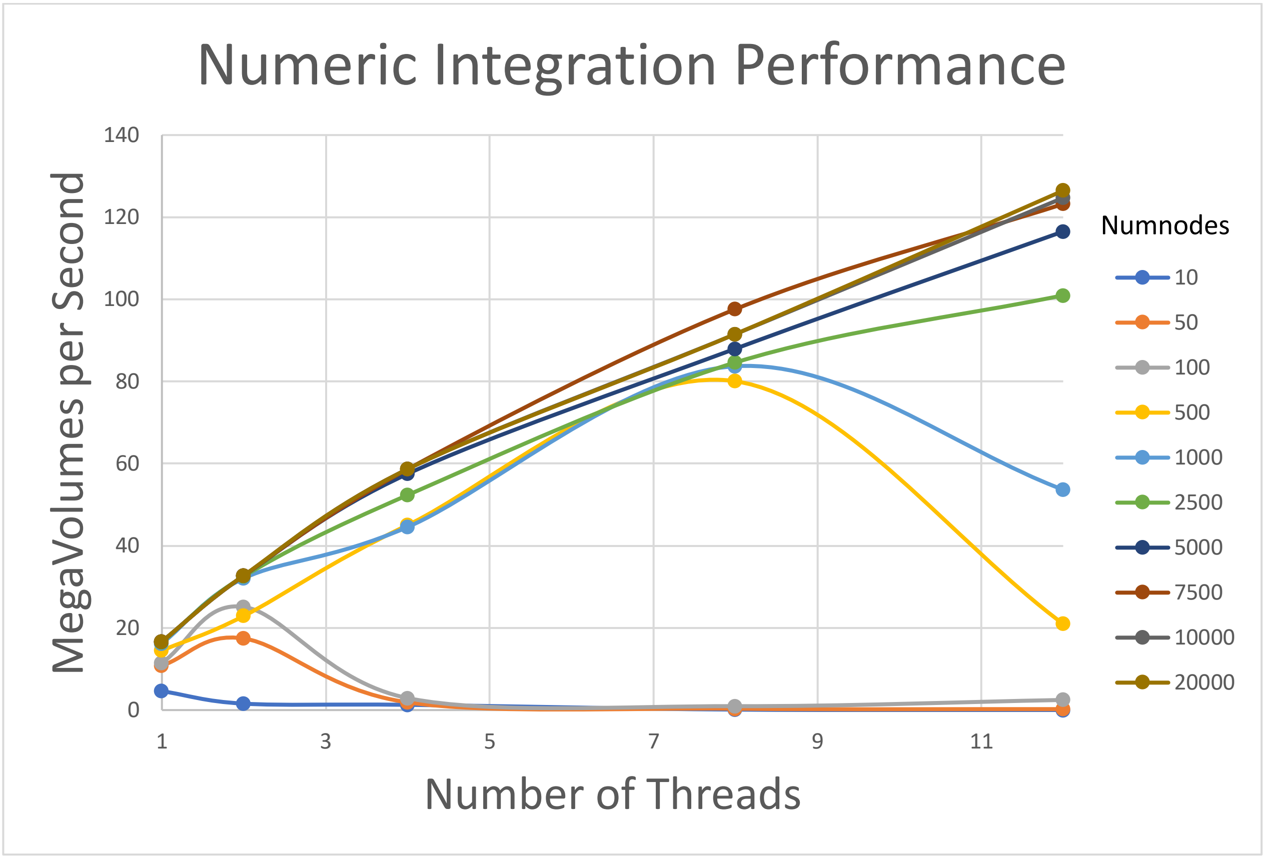
Casey Dinsmore
dinsmorc@oregonstate.edu

For this simulation, I discovered that once a large enough number of numnodes were selected, the estimated volume of the shape stabilized around 0.435.

I executed my code on my Dell workstation with a 6-core Xeon E-2236 @ 3.40GHz CPU, with hyperthreading enabled. I ran between 1 and 20k nodes using 1, 2, 4, 8, and 12 threads.

	10	50	100	500	1000	2500	5000	7500	10000	20000
1	4.61	10.84	11.51	14.51	16.22	16.56	16.72	16.7	16.73	16.73
2	1.57	17.43	25.12	22.93	32.1	32.59	32.73	32.75	32.75	32.79
4	1.28	1.88	2.92	45.05	44.58	52.36	57.53	58.67	58.7	58.69
8	0.12	0.34	0.96	80.06	83.73	84.64	87.99	97.62	91.55	91.56
12	0.01	0.24	2.47	20.98	53.65	100.98	116.52	123.22	124.72	126.48





Like many of the problems that we have seen, when the problem space is small, executing on multiple threads does not provide a speed up, in fact it causes the code to execute slower. This is presumably due to the overhead of OpenMP building up and tearing down the additional threads. An interesting point in this graph is at the 8 thread mark, between 100, 500, and 1000 numnodes. Preformance is abysmal at 100 and jumps up significantly for 500 and 1000 numnodes. The performance is nearly the same at 8 threads for 500 and 1000 numnodes, but when the threads are increased to 12, the performance once again tapers off, as there is not enough work for the additional threads to justify the overhead. As numnodes increases after 1000, the 12 thread count continues to provide performance benefits.

Using 12 cores provided the best megavolumes per second, 126.48 when executing 20,000 trials, while a single thread performed at 16.73 megavolumes per second.

Speedup = 126.48 / 16.73 = 7.56

Parallel Fraction = (12/11) * (1 – (1/7.56)) = 0.947