Jesse Thoren

April 10, 2017

CS 475 – Parallel Programming

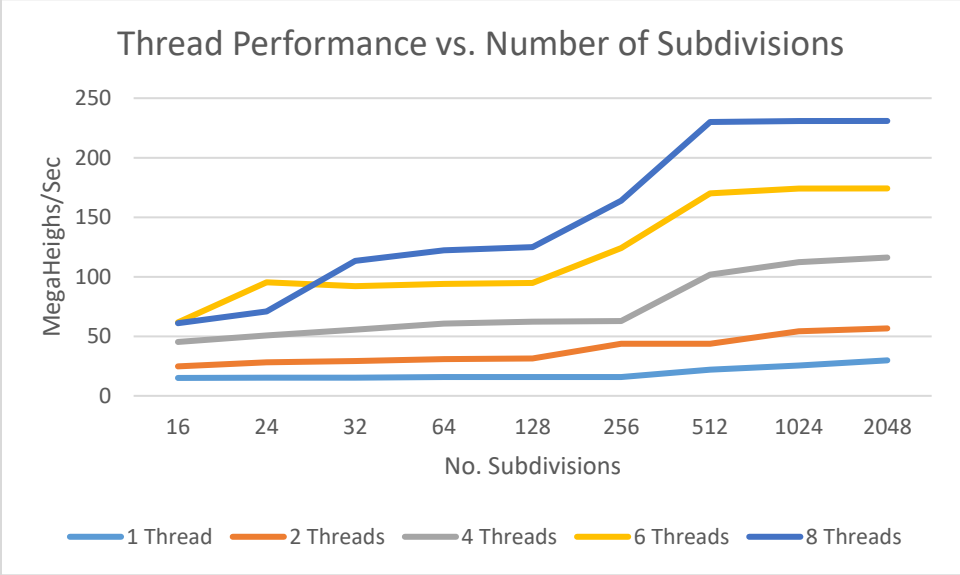
Project 1 – Written Commentary

1. My program was run on the OSU Flip server at 6:47 PM Pacific on April 10, 2017.
2. My program suggests the actual volume of the surface is around 25.312 units^3. This is based on the results of computing the volume using 2048 Subdivisions for both X and Y.
3. The average megaheights/sec over 100 runs is listed in the table below.

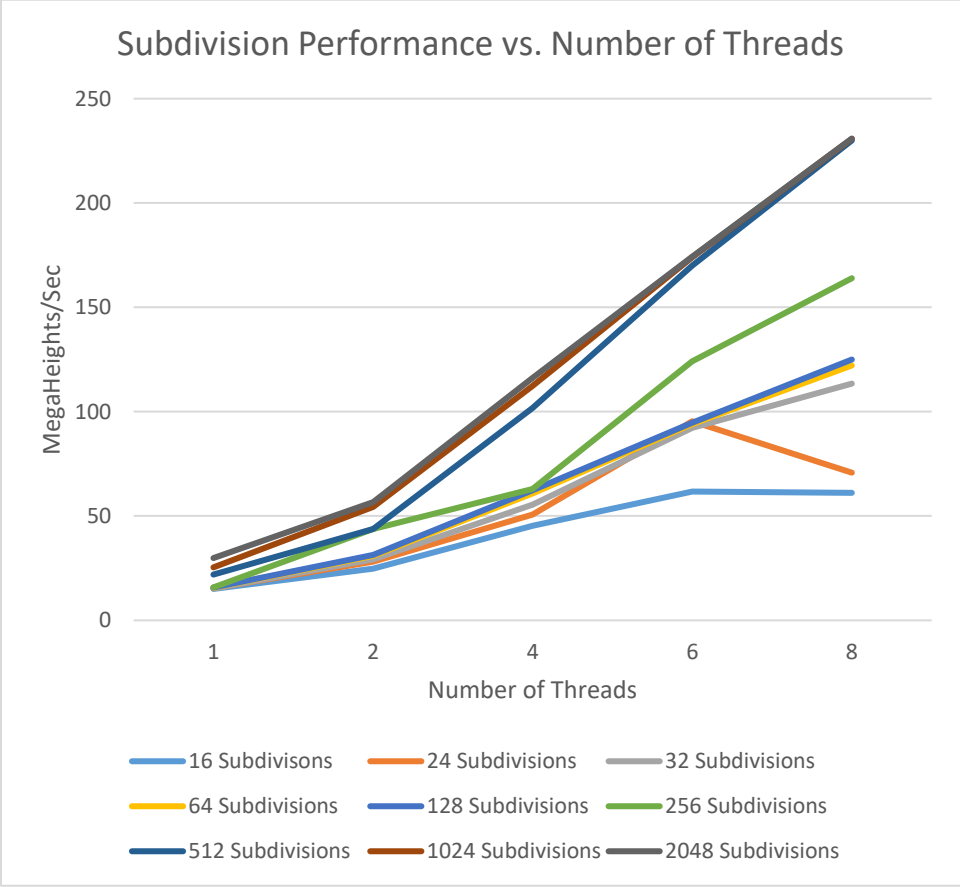
	16 Subdivisions	24 Subdivisions	32 Subdivisions	64 Subdivisions	128 Subdivisions	256 Subdivisions	512 Subdivisions	1024 Subdivisions	2048 Subdivisions
1 Thread	15.059	15.357	15.236	15.697	15.769	15.79	21.938	25.393	29.845
2 Threads	24.776	28.133	29.19	30.883	31.367	43.79	43.789	54.378	56.665
4 Threads	45.284	50.678	55.469	60.797	62.264	62.959	101.702	112.253	116.171
6 Threads	61.716	95.394	92.242	94.148	94.736	124.096	169.993	174.005	174.226
8 Threads	61.044	70.818	113.449	122.141	124.994	163.942	229.979	230.777	230.881

3 cont.

Graphed below is the Thread Performance plotted against number of subdivisions.



Graphed below is the Subdivision Performance plotted against number of Threads.



4. From the graphs above, we can see that Performance generally increases by using additional threads, at least when comparing 1, 2, 4, 6, and 8 threads.
   Furthermore, we see that performance generally increases by creating additional subdivisions, but this levels off after a sufficiently large amount of subdivisions have been created.
5. I think the performance gain that is found by increasing the amount of threads is resulting from the additional threads running in parallel on their own cores. This seems to be the case because the performance from using 1, 2, 4, 6, and 8 threads seems to be multiples of 1, 2, 4, 6, and 8 respectively. When computing average (mean) S values for the different runs, I found that the S value for 2 threads is 0.504653, for 4 threads was 0.263481, for 6 threads was 0.164097, and for 8 threads is 0.142613. Inverting these suggest that 2 threads is 1.981559 times faster than 1 thread, 4 threads is 3.795347 times faster than 1 thread, 6 threads is 6.093975 times faster than 1 thread, and 8 threads is 7.011975 times faster than 1 thread.
   The value that doesn't make sense here is 6.093975 for 6 threads, which shouldn't theoretically exceed 6. This may be occurring due to variations in CPU load as the testing script was completing its runs, causing extremely good average values for 6 threads, or extremely bad average values for 1 thread.
6. Since we know the average speedup values (as calculated in number 5) we can use the inverse Amdahl's law formula to help compute Fp. I will be computing Fp from each of the S values, and taking the mean of these as an approximation for Fp.
   Fp(n) = (n/(n-1))*(1-1/S(n)), where S(n) is the speedup for n threads.
   Fp(2) = (2/1)* (1-.504653) ≈ .9906
   Fp(4) = (4/3)* (1-.263481) ≈ .9820
   Fp(6) = (6/5)* (1-.164097) ≈ 1.0031
   Fp(8) = (8/7)* (1-.142613) ≈ .9799
   FpAverage ≈ .9889. This suggests that around 98.89% of the code in this test is parallelizable.
7. If we use the parallel fraction .9889, we can compute the maximum speed-up we could ever get by using a limit (theoretically infinite amount of processor).
   Speedup(n) = 1/((Fp/n) + (1-Fp)) as n goes to infinity simplifies to
   Speedup(Max) = 1/(1-Fp) ≈ 1/(1-.9889) ≈ 90.09 times theoretical max speedup.