CS 575 Project 1

Jeremy Dao

April 2020

1. The following results were gathered using a machine with a Intel i7-7700k CPU (4 cores, 8 threads) running at 4.20GHz.
2. The closest approximation the the actual volume I achieved was 6.48199.
3. Performance will be measured in MegaVolumes per Second, meaning how many millions of part volumes (calculating height and multiplying it by the appropriate tile area) are calculated a second. The following graph shows the performance versus the number of nodes used for different number of threads:

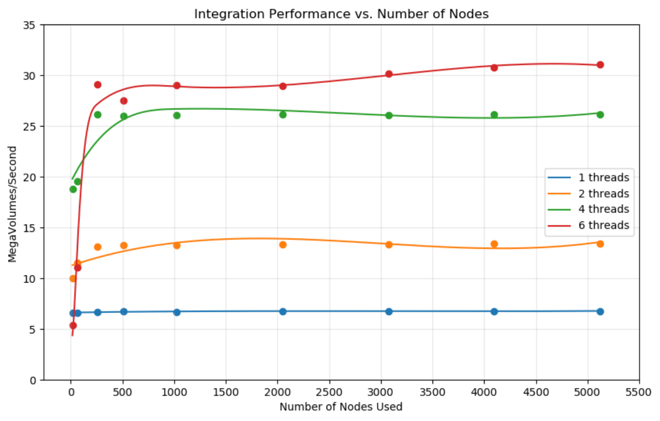

Figure 1: Comparison of node performance for different number threads

The following graph shows the performance versus the number of threads for different number of nodes used.

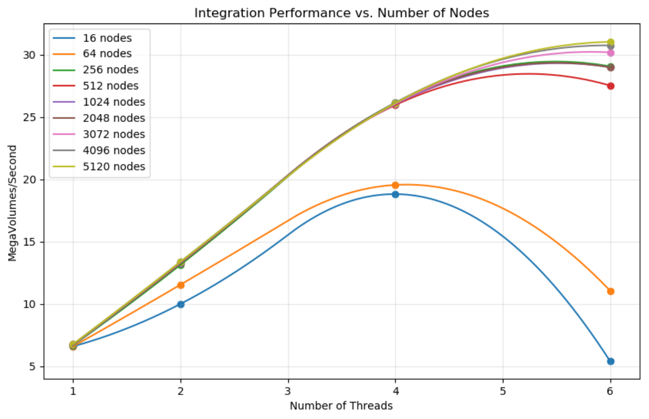

Figure 2: Comparison of thread performance for different number of nodes

The following table displays the same data used in the graphs above. Note that again, all numbers are in MegaVolumes per Second.

threads \ nodes	16	64	256	512	1024	2048	3072	4096	5120
1	6.6	6.59	6.68	6.76	6.7	6.74	6.75	6.76	6.77
2	10.0	11.55	13.14	13.26	13.29	13.32	13.36	13.38	13.38
4	18.81	19.55	26.14	25.96	26.03	26.11	26.07	26.15	26.11
6	5.4	11.06	29.06	27.52	29.0	28.98	30.19	30.74	31.03

4. The most obvious pattern that we can see is that in general using more threads results in greater performance, as can be seen from figure 1. However, this is not the case when there are a small number of nodes (the very left of the graph in figure 1). Note how for threads 1 and 2, the curves are relatively quite flat, where as for threads 4 and 6 the performance starts out low, then increases and plateaus as the number of nodes used increases. Furthermore, note how this effect is more extreme for 6 threads than 4 threads. This same trend is seen in figure 2. We can clearly see that the small number of nodes (16 and 64) do not follow the general performance trend as the other node numbers. They do not have as steep of a performance increase when increasing the number of threads used and when going to 6 threads the performance outright nosedives.
   Another trend we see is that the speed up we get going from 4 threads to 6 threads is not as great as the speed up going from 2 threads to 4 threads even though in both cases we added 2 threads.

5. These trends are to be expected and both are explained by the overhead that comes with parallelism. 4 and 6 threads see slow performance for small number of nodes because there is less data to spread the overhead of using 4 or 6 threads over. In the case 6 threads, the parallelism overhead actually outweighs the benefit of having so many threads, which causes the performance to be slower than using less threads (the nosedive in the graph of figure 2). This is why in figure 1 for 4 and 6 threads, the performance starts out slow then speeds up rather than maintaining a steady value like 1 or 2 threads. The overhead of spawning and managing threads also explains the second trend we see. As more threads are used, we get diminishing returns since there is more overhead, but the same amount of data to spread it over. This is why we get less speed going from 4 to 6 threads than we do going from 2 to 4.

6. The below calculation shows the calculation of the parallel fraction F for going from 1 thread to 6 threads when using 5120 nodes.

$$S_6 = \frac{Perf_6}{Perf_1} = \frac{31.03}{6.77} \approx 4.5835$$
$$F = \frac{6}{6-1}\left(1 - \frac{1}{S_6}\right) = \frac{6}{5}\left(1 - \frac{1}{4.5835}\right) \approx .9382$$

7. Given the parallel fraction calculated above, the maximum possible speed up is:

$$S_{max} = \frac{1}{1-F} = \frac{1}{1-.9382} \approx 16.1783$$