Project #1
OpenMP: Numeric Integration with OpenMP

Haoxiang Wang; Student ID: 932359049

1. Machine Running the Test

I used school's server "flip" to run the test. At the time my test is running, the "uptime" is less than 1.

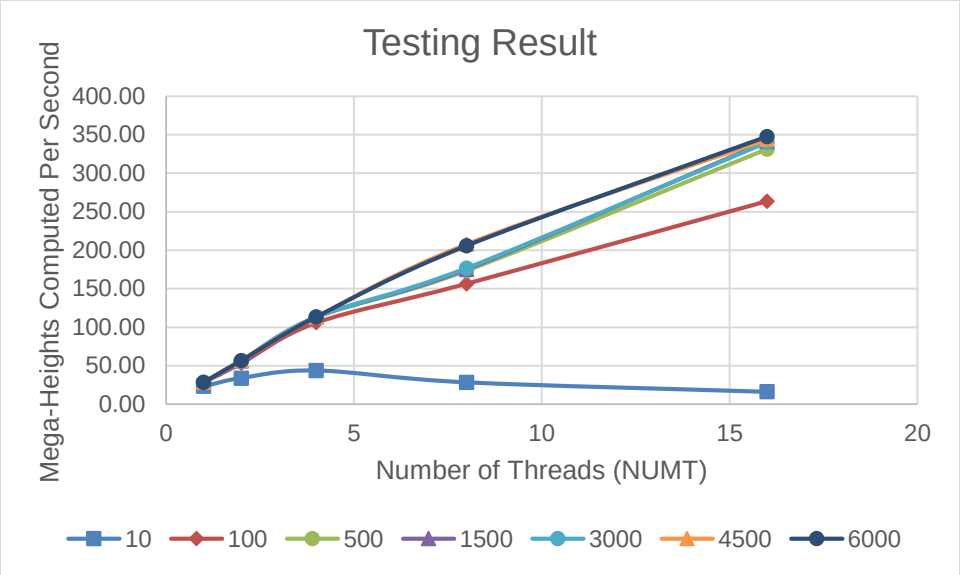
2. Performance Results

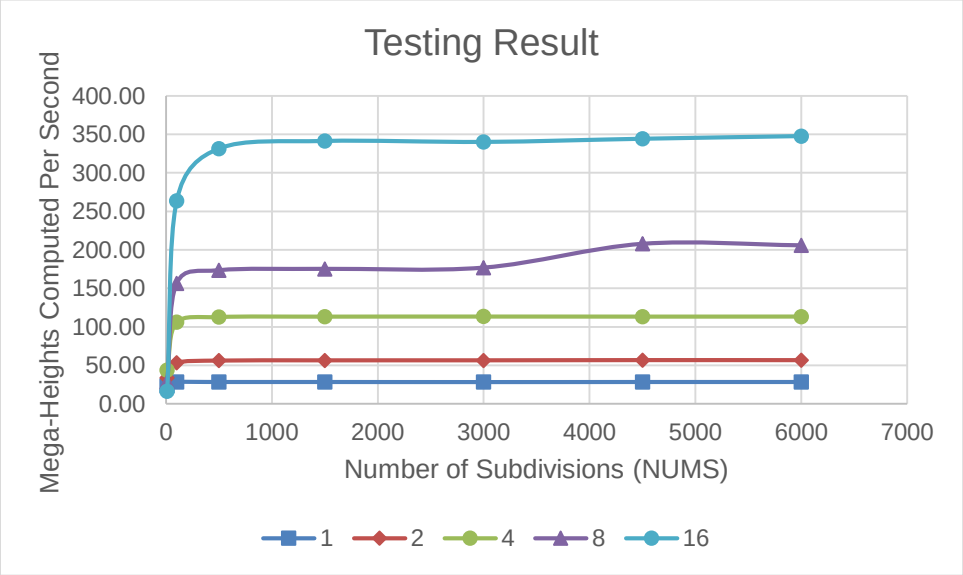
I used Benchmarks to let the program running with different number of threads, and different size of the array. The number of thread I used for the test are 1, 2, 4, 8, and 16. The size of the subdivision I used in the test are 10, 100, 500, 1500, 3000, 4500 and 6000. Since the subdivision is for the edge of the surface, the total size of the whole problem is the square of the subdivisions. The performance is evaluated by calculating mega-heights computed per second. The following form is the result I got from the test.

	10	100	500	1500	3000	4500	6000
1	23.13	28.24	28.34	28.34	28.25	28.34	28.34
2	33.90	53.09	55.94	56.25	56.29	56.56	56.49
4	43.61	105.86	112.68	113.19	113.32	113.25	113.30
8	28.34	156.50	173.44	175.18	176.79	207.73	205.75
16	16.15	263.71	331.48	341.32	340.09	344.19	347.66

In the form, the horizontal direction stands for the number of subdivisions, the vertex direction stands for the number of the threads. In the middle is the original testing results I got from the test. Two graphs are drawn based on these results. In the first one, the x axis will be number of threads, and the y axis will be mega-heights computed per second, and in the second one, the x axis will be the number of subdivisions and the y axis will be mega-heights computed per second. These two graphs are shown below.

Also, depends on the volume results I got from each test, I believe the actual volume is 14.06. It is the most common result I got from all of the tests.





3. Behavior Explanation

As it shows in the graph 1, each line stands for one particular number of subdivisions, and each node on the line stands for the performance based on particular number of threads. The bigger value mega-heights computed per second has, the higher performance it shows.

So based on the graph 1, when the number of subdivision is fixed, the more threads the test uses, the higher performance it has,

except the case that only has 10 subdivisions. This really makes sense since the more threads work at the same time the shorter time they will use to complete the task. When the problem size is really small, the time creating parallel tasks might longer than the time finishing the tasks, so the performance goes down while number of threads goes up.

As it shows in the graph 2, each line stands for one particular number of threads, and each node on the line stands for the performance based on particular number of subdivisions. The bigger value mega-heights computed per second has, the higher performance it shows.

So based on the graph 2, when the thread number is fixed, the larger the number of subdivisions is, the higher “performance” it produces in the beginning. I think this is because the threads have more work to do and haven’t reach their highest limits. Then in the upcoming cases with more subdivisions, the performance stays similar till the end. I think this is because threads have reached their highest limits.

The way I calculate the Parallel Fraction is calculating each F parallel based on fixed number of threads and each subdivisions using Inverse Amdahl's law. Since the case of 10 subdivisions is so wired that could not be considered as a common case, I used cases with number of subdivisions from 100 to 6000. Then I calculate the average along different subdivisions in order to have the Parallel Fraction of one fixed number of threads. The results are:

2 threads: 0.979130003	4 threads: 0.992168107
8 threads: 0.956277381	16 threads: 0.962157526

Based on these Parallel Fractions, the maximum speed-up could be calculated, and the results are:

2 threads: 47.91567526	4 threads: 127.6830529
8 threads: 22.87145703	16 threads: 26.4253334

So the maximum speed-up I could ever get is 127.6830529