Lachlan Sinclair

sinclala@oregonstate.edu

4/22/2020

CS 475 Spring

Project #2: Numeric Integration with OpenMP Reduction

System: I used my personal computer for this project which uses a Windows OS. The program was developed in Visual Studios and the script used to collect the data was written in Python using PyCharm.

Results:

From my runs, the volume of the superquadric (N=4) was 6.48. After 320 subdivisions, the volume calculation converged on approximately 6.48.

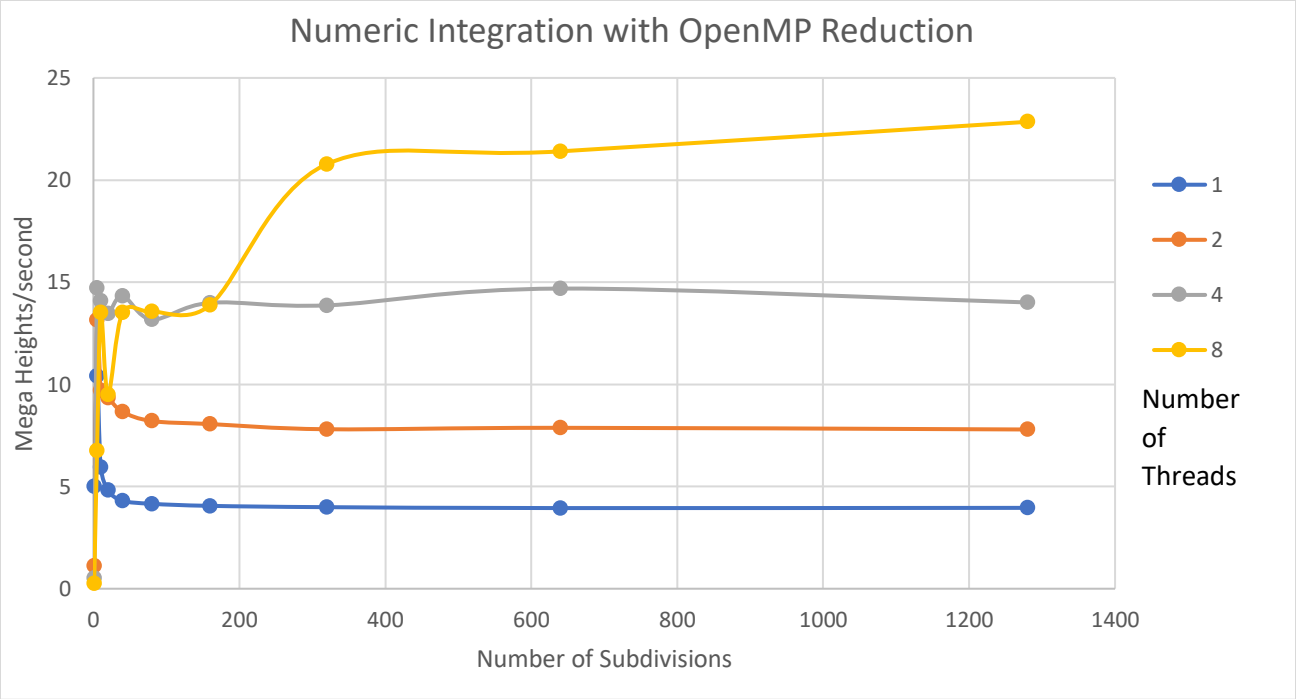
Data gathered from the results.txt file.

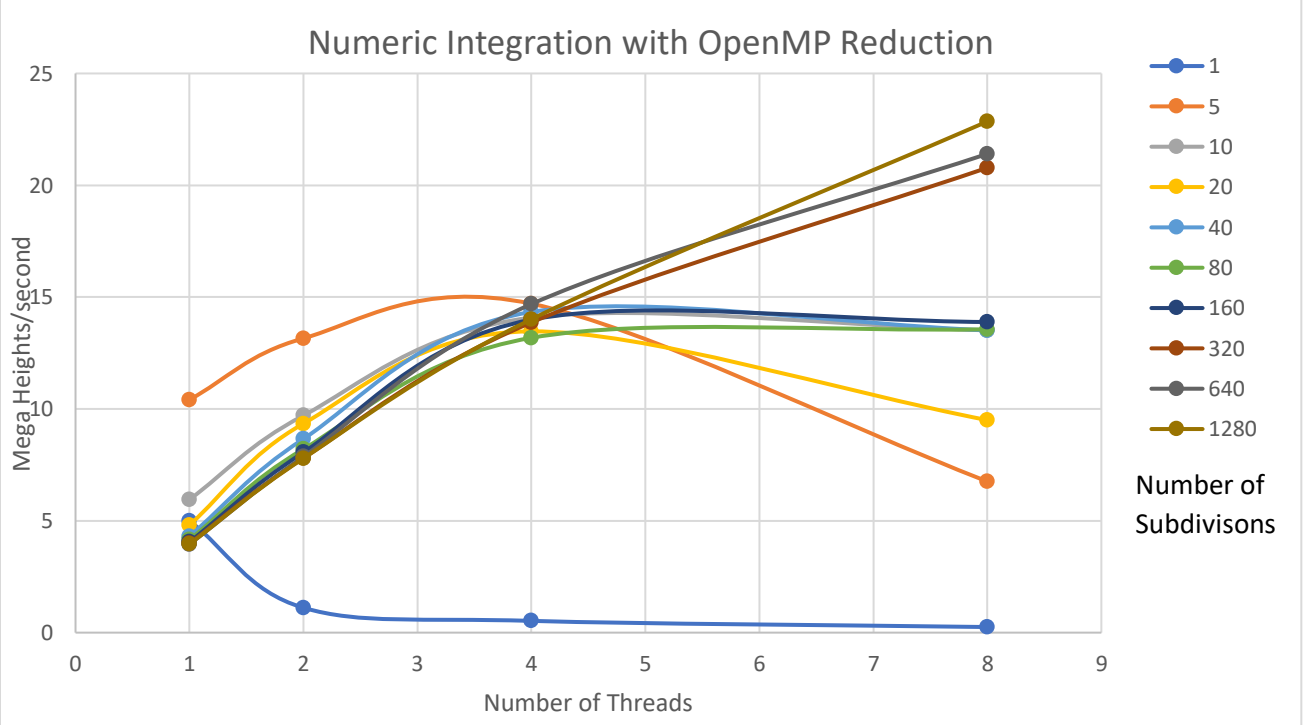
Threads	Number of subdivisions	Mega Heights per second	Volume
1	1	5.000144	0
1	5	10.416682	4.402326
1	10	5.952382	5.856353
1	20	4.813478	6.246917
1	40	4.310345	6.412089
1	80	4.156384	6.456376
1	160	4.055317	6.472585
1	320	3.992	6.479541
1	640	3.947453	6.481246
1	1280	3.958153	6.472962
2	1	1.111112	0
2	5	13.157895	4.402326
2	10	9.708739	5.85635
2	20	9.345795	6.246922
2	40	8.662697	6.412087
2	80	8.207232	6.456361
2	160	8.058931	6.472583
2	320	7.805235	6.479324
2	640	7.875741	6.48079
2	1280	7.793225	6.478932
4	1	0.526316	0
4	5	14.705896	4.402326
4	10	14.084501	5.85635
4	20	13.468013	6.246922
4	40	14.336917	6.412089

4	80	13.182286	6.456368
4	160	13.994424	6.472581
4	320	13.869513	6.47911
4	640	14.690218	6.480544
4	1280	14.010723	6.483705
8	1	0.25	0
8	5	6.756759	4.402326
8	10	13.513517	5.856351
8	20	9.501187	6.246921
8	40	13.513514	6.412086
8	80	13.562196	6.456368
8	160	13.887382	6.472594
8	320	20.783016	6.479224
8	640	21.405466	6.480999
8	1280	22.852457	6.481893

Below is the able used to generate the graphs. The rows are the number of threads and the columns are the number of subdivisions.

	1	5	10	20	40	80	160	320	640	1280
1	5.000144	10.416682	5.952382	4.813478	4.310345	4.156384	4.055317	3.992	3.947453	3.958153
2	1.111112	13.157895	9.708739	9.345795	8.662697	8.207232	8.058931	7.805235	7.875741	7.793225
4	0.526316	14.705896	14.084501	13.468013	14.336917	13.182286	13.994424	13.869513	14.690218	14.010723
8	0.25	6.756759	13.513517	9.501187	13.513514	13.562196	13.887382	20.783016	21.405466	22.852457





The data collected follows the expected trends. When the amount of subdivisions is small, the runs that had a smaller thread counts have better performance. This is because the amount of time required to create the threads dominates the time required to run the actual calculations. This is highlighted very well in the second graph, for example, the blue line that represents a single subdivision has the worst performance for every thread count other than the single thread. Once the number of nodes being

processed got very large the performance between the different thread counts behaved as expected, the higher the thread count the higher the performance (up to a thread count of 8). This is because the amount of time required to perform the calculations is significantly larger than the thread set up time.

Every number of threads produced roughly the same volume when using a higher number of subdivisions, this means that the reduction method for summing the volume worked correctly. The remaining difference seen in the volumes is “down in the noise” as mentioned by the professor in the Piazza, the number of threads affects the reduction, which changes the rounding that occurs slightly. The small swaying in the performance for the thread counts 4 and 8 after they plateau was likely because of my computers other processes, or something along those lines. I do not think this program was being affected by temporal coherence.

Speedup = 22.852457 / 3.958153 = 5.77

Then when considering the 1 thread to 8 thread speedup the parallel fraction is:

Fp = (8 / 7) * (1-(1/5.77)) = 0.95

The maximum speed up is defined by 1/(1-Fp):

Max speedup = 1 / (1-.95) = 20

Explanation:

This program works by splitting up the 2D central plane of the superquadric into a number of subdivisions based off a command line input. The program then uses OpenMP to distribute the calculation of each nodes (square in the grid of subdivisions) height between the various threads in the pool. The pragma call uses reduction for summing the volume. This is required since each thread will multiply the area of the node by the calculated height after and then add that result to the overall volume. Using reduction allows for each thread to create it own private temporary volume variable.

The program accepts two command line arguments, the first being the number of threads and the second being the number of subdivisions to use. This volume calculation is performed 10 times when the program is run and the maximum performance is returned, in each of these 10 runs the volume calculation will always be the same. The volume and other information such as subdivisions, performance, number of threads used, and number of subdivisions is printed out in a excel exportable manner. The performance is tracked in mega heights per second.

One major thing to note about the logic of my program is how I handle corner and edge cases. After stepping through the code and watching the variables when calling the height method on an edge or corner nodes I noticed that the height will always be zero. After looking through the equations I noticed that this will hold true for any value of N. Therefor I refactored my code to check to see if the node being calculated is a corner or an edge case, if it is the code moves onto the next node without changing the volume. This increases the efficiency of program since it instantly handles these cases rather than checking through nested if statements and calling the height function only to multiply the reduced area of the node by 0.