Nick Pepperling
4/21/2014
CS475

HW1: Numeric Integration with OpenMP - Writeup

- Using OpenMP I calculated the volume for each step of subdivisions used. I took the average of the highest granularity used (4096 subdivisions) for each runthrough with different thread counts. The final calculated total was: **14.011**

1.) Machine Configuration

I ran this on the machine intel gave us for our senior design project. It contains:

- Intel Xeon E5-2690
- 8-Cores @ 2.90GhZ
- Hyperthreaded (16 total cores)
- 20Mb CPU cache
- 64 GB RAM
- SSD

2.) What do you think the actual volume is?

If I had to guess I would say 14 exactly. This is because a few of the iterations returned volume values slightly below 14 and a few returned slightly above 14. If I would have redefined my loop to check thread counts (1-16) rather than (1, 2, 4, 8, 16), I could have had more confidence in my guess but 14 seems reasonable.

3.) Show the performances you achieved in tables and graphs?

See Next Pages

Table Data for Multiple Run Throughs

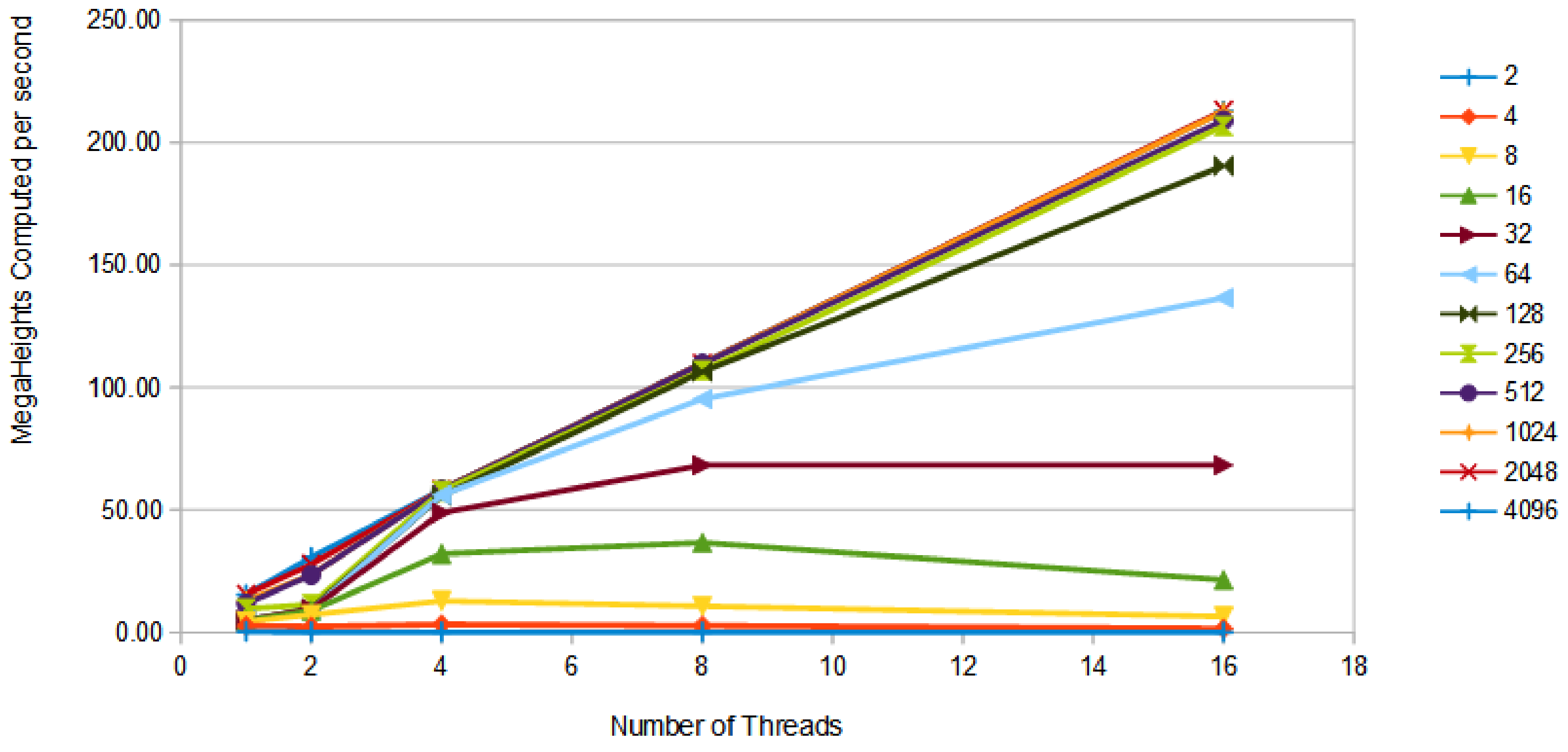
	1	**2**	**4**	**8**	**16**
2	0.27	0.04	0.05	0.04	0.02
4	2.67	2.29	3.20	2.67	1.45
8	4.27	7.11	12.80	10.67	6.40
16	4.74	8.83	32.00	36.57	21.33
32	4.85	9.39	48.76	68.27	68.27
64	4.81	8.87	56.11	95.26	136.53
128	4.87	9.65	55.73	106.39	190.51
256	9.57	11.34	57.59	106.91	206.74
512	11.63	23.38	58.14	109.68	209.05
1024	12.87	23.49	58.18	109.80	212.65
2048	15.37	27.72	58.12	109.78	213.10
4096	15.37	30.59	58.16	109.86	212.83

The above table shows the Number of subdivisions in the leftmost column (NUMS) and the number of threads in the topmost row (NUMT). The data incident to each row column pair is the calculated number of MegaHeight Calculations per Second made calculated via:

$$\frac{NUMS * NUMS}{EndTime - StartTime}$$

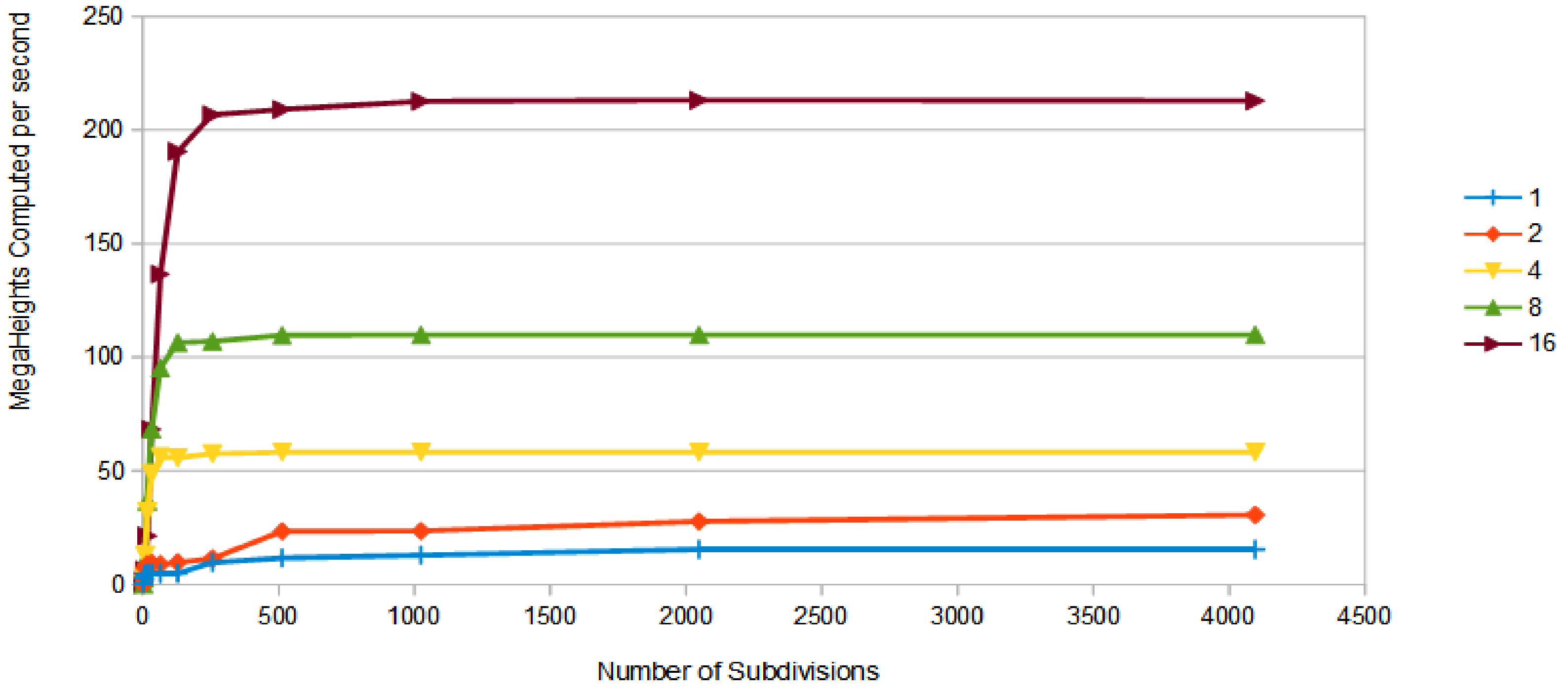
OpenMP: Numeric Integration

Figure 1.1 - Data Series as Number of Subdivisions



OpenMP: Numeric Integration

Figure 1.2 - Data Series as Number of Threads



4.) What patterns are you seeing in speed?

The most obvious pattern in figure 1.1 is the speed increase as threads are added. Apart from when the number of subdivisions is less than or equal to 16, each time the number of threads increases there is a significant performance boost. Figure 1.2 is interesting. It is clear that thread count makes a big difference in performance as the number of subdivisions increases, and the performance jumps between each data series is large and relatively proportional. This was to be expected as the processor in the computer I used was designed for high performance computing and scalability.

5.) Why do you think it is behaving that way?

As mentioned before, the results were highly expected of the hardware I used to perform the experiment. Furthermore, looking at figure 1.1, we see that when run with subdivisions less than or equal to 32 there is a speed decrease past the 8 core count mark. This could be due to worker "starvation" as there is not enough work to be spread amongst all cores evenly, thus the overhead of managing the threads becomes a bottleneck and decreases performance. Looking at figure 1.2, we see what seems to be an asymptotic plateau in performance as the number of subdivisions increase. This is likely because the processor has maximized its ability to run parallely based on the code in the program.

6.) What is the parallel fraction of this application?

To calculate this I will use the number of subdivisions with the finest granularity: 4096

-S (single thread) time spent @ 4096 subdivisions: 1.0917 seconds
-P (16 threads) time spent @ 4069 subdivsions: 0.0788 seconds

$$F_p = \frac{1.0917}{1.0917 + .0788} = \underline{.933}$$

7) What is the maximum speedup you could *ever* get?

$$\text{MaxSpeedup} = \frac{1}{1 - .933} = \underline{14.93}$$