Blair Todd

CS4380

Project 3

Collatz

Question 3.1a) What compute times do you get (in seconds)? List them in a table where the first column shows the number of threads used (increasing from top to bottom) and the second column lists the compute time. Use two digits after the decimal point for the compute times.

# of threads	Compute Time
1	18.63
4	4.94
8	2.65
12	1.84
16	1.47
20	1.23
24	1.08
28	0.98
32	0.99
36	0.75
40	0.70
44	0.70
48	0.78
52	0.71
56	0.71
60	0.70
64	0.67

*upper bound is constant at: 50000000

Question 3.1b) What is the highest observed speedup (relative to the pthread code running with one thread)?

- 64 threads : 27.80

Question 3.1c) The highest speedup is significantly higher than the number of cores in a compute node of Lonestar 5. What makes such a high speedup possible?

- Pthreads use threads instead of full-fledged processes meaning more instructions run simultaneously.

Question 3.1d) Sometimes, the highest speedup is not achieved with a thread count that is an integer multiple of the core count. Provide a good plausible reason for why that is

- The speed up could be affected by the load distribution being dispersed among the cores. Having the thread count being a multiple could affect the timing of the threads(more than one threads waiting on a mutex)

Fractals

Question 3.2a) What compute times do you get (in seconds)? List them in a table where the first column shows the number of threads used (increasing from top to bottom), the second column shows the compute time for the smaller input, and the third column shows the compute time of the larger input. Use two digits after the decimal point for the compute times.

Threads	Small Input Times (512)	Large Input Times (1028)
1	9.39	37.61
8	1.86	7.37
16	0.98	3.954
24	0.61	2.33
32	0.53	2.01
40	0.45	1.62
48	0.25	0.81
56	0.32	1.11
64	0.38	1.28

Question 3.2b) What is the highest observed speedup for the larger input, the corresponding efficiency (in percent), and at what thread count is the lowest compute time achieved?

- 46.43 :highest , 96%
- 8

Question 3.2c) The cyclic partitioning potentially results in false sharing in the pic array. Looking at the code (rather than the compute times), is it likely that significant false sharing will occur? Explain why or why not.

- Threads accessing the pic array's information could cause false sharing by other threads caching data between calls causing errors between thread calls (not thread-safe).

Raytrace

Question 3.3a) What compute times do you get (in seconds)? List them in a table where the first column shows the number of threads used (increasing from top to bottom), the second column shows the compute time for the smaller input, and the third column shows the compute time of the larger input. Use two digits after the decimal point for the compute times.

Threads	Small Input	Large Input
1	0.81	3.22
8	0.15	0.34
16	0.06	0.22
24	0.08	0.23
32	0.06	0.18
40	0.06	0.19
48	0.07	0.16
56	0.05	0.21
64	0.06	0.15

Question 3.3b) What is the highest observed speedup for the larger input and why is it so much lower than the corresponding speedup obtained on the fractal code?

- Highest speedup: 21.46
- More computations and frame/width size are increased. Also more code was serialized than fractals.

Question 3.3c) Explain why running the prepare function serially is not a significant problem (in terms of Amdahl's law) for the given inputs.

- The prepare function when ran serially only adds a small amount of serial overhead and most of the computation is made only once.

Question 3.3d) Which aspect of the ball_y computation makes it hard to find a closed-form solution so the values do not have to be precomputed (like in fractal)?

- The aspect that ball_y uses itself when checking if vel needs to be multiplied by a -1.