Andrew Yaros
CS 361 Assignment 1 Part 2
Report

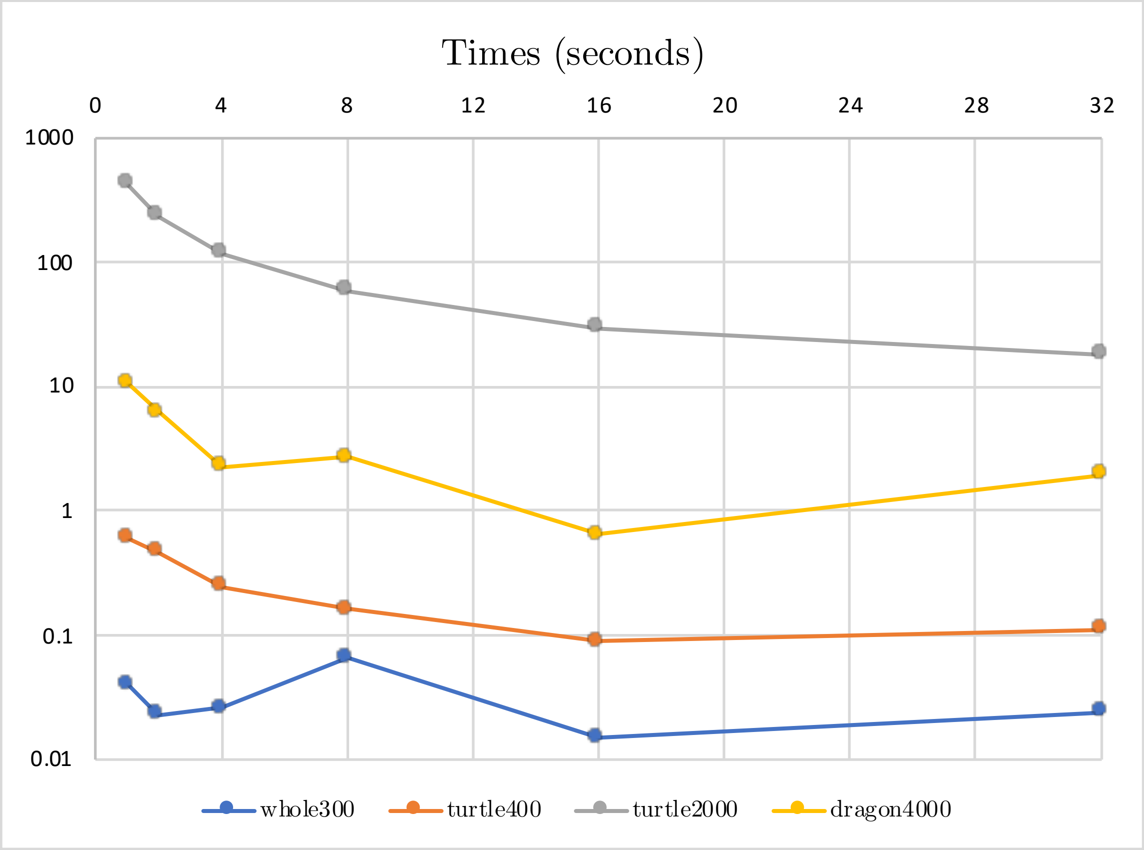
The makefile runs each test for 1, 2, 4, 8, 16, and 32 threads. The order of arguments is irrelevant; the program will work as long as each textual argument is followed by a number. There must be exactly 14 arguments (7 name/value pairs) or the program will exit.

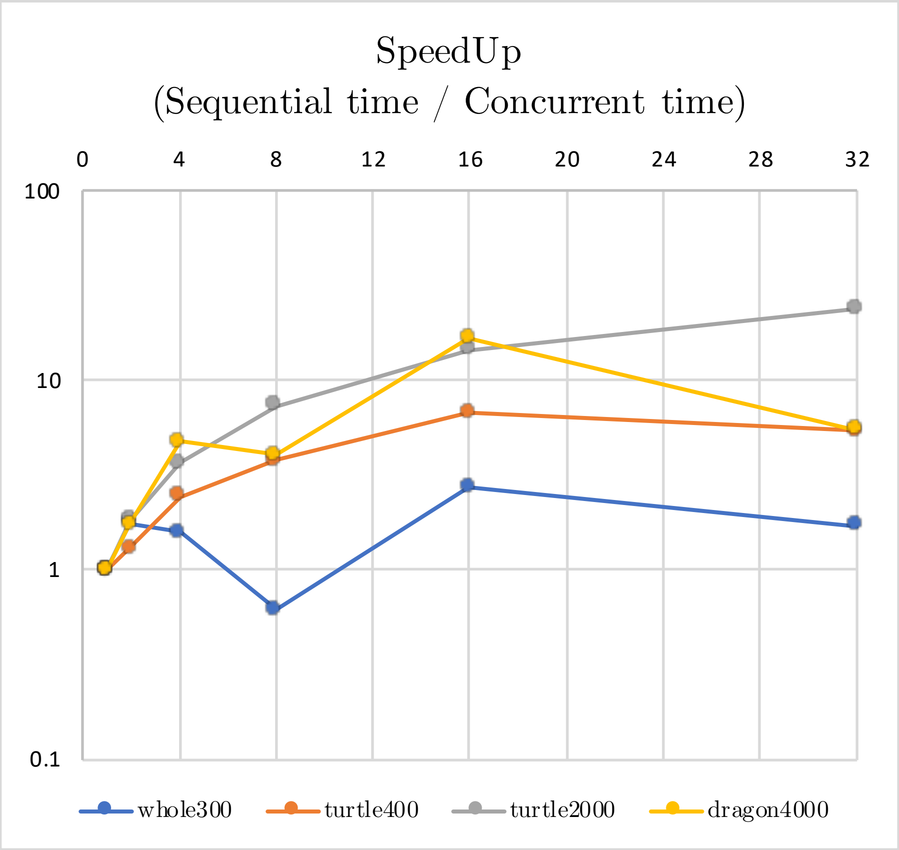
I put a static volatile 4-dimensional array of doubles into the main class to hold one 3D array per thread (as before, a 3D array is generated: width x height x color). I used the approach where I created a thread class that extends the Thread object; the necessary arguments are passed in via the constructor. Each thread uses it's own local 3D array to hold its part of the data. When a thread's computations are finished, it puts it's data into the static variable in the main class. Once the execution is finished, the 4D array of results is converted into a single 3D array containing all the data, after which an image is saved.

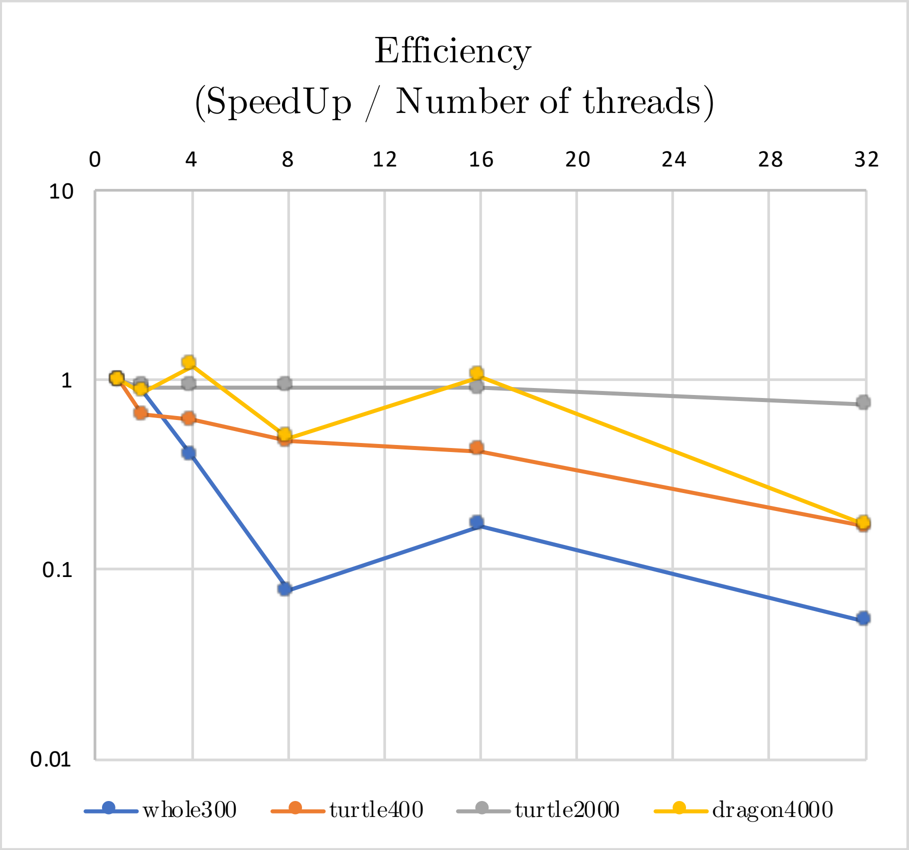
The program divides up the computation so a portion of the height of each image is calculated; the YLO and YHI values for each thread are adjusted to allocate only 1/NUMTHREAD of the image. Also, the program is written so the last thread may be given slightly more data if there is a remainder; when looping through the threads to start or join them we only go to NUMTHREADS-1; the NUMTHREADSth thread is done on it's own.

I optimized the code by running the calculations 5 times in a row before actually timing them to benefit from the "JVM warmup" effect. This significantly improved the speed of the timed calculations. Note; there is an issue where running the first benchmark (whole300) with 8 threads results in a much slower time than I would have expected; I'm still not exactly sure why this is the case. This issue isn't present on the larger benchmarks, which show a consistent speed up as the number of threads is increased.

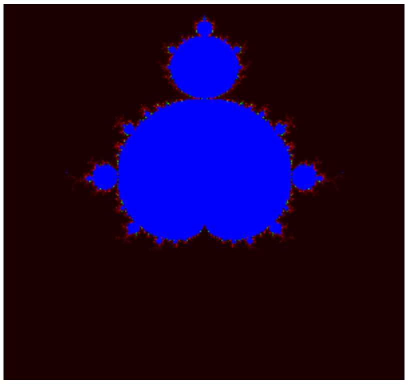
# of threads	whole300	turtle400	turtle2000	dragon4000	
1	0.041	0.608	430.034	10.872	Times (seconds)
2	0.023	0.464	234.016	6.283	
4	0.026	0.248	117.087	2.298	
8	0.066	0.161	58.707	2.714	
16	0.015	0.090	29.546	0.653	
32	0.024	0.112	18.069	1.969	
1	1.00	1.00	1.00	1.00	SpeedUp
2	1.78	1.31	1.84	1.73	(sequential/
4	1.58	2.45	3.67	4.73	concurrent)
8	0.62	3.78	7.33	4.01	
16	2.73	6.76	14.55	16.65	
32	1.71	5.43	23.80	5.52	
1	1.00	1.00	1.00	1.00	Efficiency
2	0.89	0.66	0.92	0.87	(speedup/
4	0.39	0.61	0.92	1.18	# of threads)
8	0.08	0.47	0.92	0.50	
16	0.17	0.42	0.91	1.04	
32	0.05	0.17	0.74	0.17	



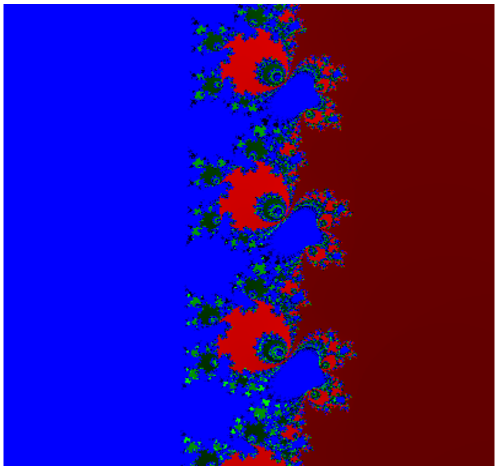

SpeedUp
(Sequential time / Concurrent time)
0
4
8
12
16
20
24
28
32
100
10
1
0.1
whole300
turtle400
turtle2000
dragon4000


Efficiency
(SpeedUp / Number of threads)
0
4
8
12
16
20
24
28
32
10
1
0.1
0.01
whole300
turtle400
turtle2000
dragon4000

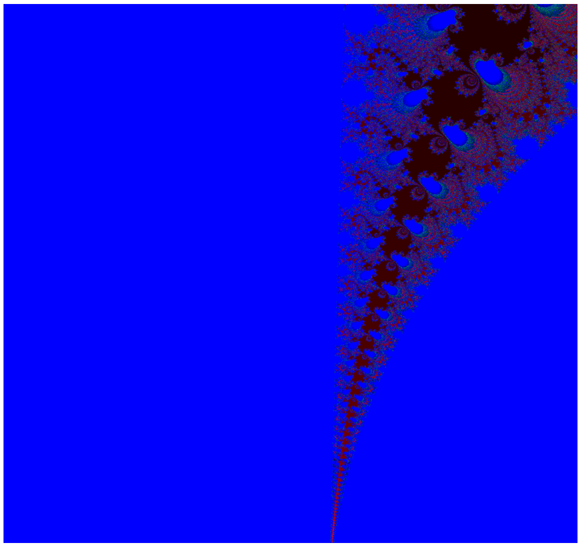
These images were generated on the 32 thread iterations.
(not that it makes much difference)



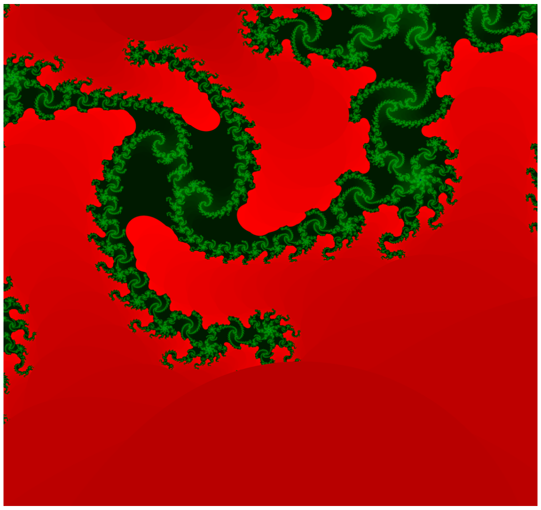
whole300



turtle400



turtle2000-5000



dragon-4000