Matt Mallett CprE 308 Project 2

This was a fun project. I'm always interested in learning how to make my programs faster and more efficient. I took this as a challenge to make my program as fast as I can. The problem itself wasn't overly complicated which made it easier to focus on multithreading my program. I learned how to implement a multithreaded program from the ground up; in previous labs, most of the code was already written, and most of the problem had been solved. I'm getting a lot more comfortable with pointers and am starting to realize their power. I relied on pointers a lot more in this project. I also decided to try my hand at shell scripting, which made it a lot easier to run all of my tests. All in all, this was a good project and I feel like I learned alot.

My design is pretty straightforward. For N threads, and a grid with S elements, the first N-1 threads each take S/N indeces. The N^{th} thread takes S/N + S%N work. The N^{th} thread is the main process, and does all of the extra work to move the process to the next generation as well. Below is the psuedocode for both types of thread:

```
Global
	mutex thread_complete_mut
	cv thread_complete_cv
	mutex gen_complete_mut
	cv gen_complete_cv
	threads_running
	gen_complete
thread
	for gen in 0 to num_gens
		lock gen_complete_mutex
		while gen > gen_complete
			wait(gen_complete_cv,gen_complete_mutex)
		unlock gen_complete_mutex
		simulate region
		lock thread_complete_mutex
		threads_running--
		broadcast(thread_complete_cv)
		unlock thread_complete_mutex
//main on next page
```

```
main
      for gen in 0 to num_gens
      simulate region
      lock thread_complete_mut
      threads_running--
      broadcast thread_complete_cv
      while(threads_running != 0)
            wait(thread_complete_cv, thread_complete_mut)
      threads_running = num_threads
      unlock thread_complete_mut
      move program to next generation
      lock gen_complete_mut
      gen_complete++
      unlock gen_complete_mut
      broadcast gen_complete_cv
```

The threads run through their generation, then wait until the main thread moves the program to the next generation. The main thread waits until every thread finishes the current generation, then moves the program to the next generation. I think it's necessary to have 2 mutexes in this style. You have to keep track of how many threads are on the current generation, and what the current generation is. Each variable represents a critical region, and needs a cv and mutex.

threads_running forces main to wait until all threads are done before moving the program to the next generation.

gen_complete basically holds the value of the currently executing generation. Any thread that finishes early is blocked from getting too far ahead as long as their generation variable is greater than gen_complete. This prevents any thread from starting on a generation before the main has prepared that generation.

Size	Threads	Real (s)	User (s)	Sys (s)	Speedup
	1	0.019	0	0.003	1.0000
	2	0.013	0.001	0.003	1.4615
	4	0.016	0.003	0.005	1.1875
	6	0.021	0.003	0.007	0.9048
4	8	0.022	0	0.015	0.8636
	1	0.017	0.001	0.003	1.0000
	2	0.014	0.004	0.003	1.2143
	4	0.031	0.003	0.005	0.5484
	6	0.101	0.003	0.01	0.1683
8	8	0.019	0.006	0.01	0.8947
	1	0.068	0.021	0.002	1.0000
	2	0.118	0.026	0.007	0.5763
	4	0.058	0.033	0.003	1.1724
	6	0.031	0.041	0.006	2.1935
32	8	0.033	0.033	0.016	2.0606
	1	0.077	0.06	0.001	1.0000
	2	0.062	0.081	0.009	1.2419
	4	0.06	0.129	0.005	1.2833
	6	0.053	0.135	0.008	1.4528
64	8	0.054	0.147	0.011	1.4259
	1	0.724	0.691	0	1.0000
	2	0.41	0.767	0.005	1.7659
	4	0.222	0.806	0.003	3.2613
	6	0.256	0.068	0.019	2.8281
256	8	0.187	1.112	0.004	3.8717
	1	11.055	10.672	0.016	1.0000
	2	6.369	11.542	0.008	1.7358
	4	3.523	11.87	0.017	3.1380
	6	3.299	14.184	0.032	3.3510
1024	8	2.666	15.803	0.012	4.1467
	1	44.356	42.419	0.024	1.0000
	2	24.831	43.778	0.04	1.7863
	4	13.543	47.415	0.034	3.2752
	6	13.155	55.502	0.058	3.3718
2048	8	12.482	63.067	0.041	3.5536
	1	179.994	169.04	0.113	1.0000
	2	99.751	189.867	0.09	1.8044
	4	54.4	184.645	0.096	3.3087
	6	47.72	221.047	0.103	3.7719
4096	8	36.391	250.067	0.095	4.9461
	1	392.322	372.524	0.228	1.0000
	2	205.07	408.659	0.168	1.9131
	4	117.739	408.659	0.168	3.3321
	6	106.155	475.163	0.205	3.6957
6000	8	80.479	540.729	0.243	4.8748

Tests were run on a machine in Coover 2050. All tests were run through 100 generations. Raw data for these tests are included in the submission in the test_results.txt file. The script used to run the tests is run_tests.sh.

Speedup was calculated as real(1)/real(i) for each input size.