Programming Project #1
Gaussian Elimination
CS4379: Parallel and Concurrent Programming
CS 5379: Parallel Processing
Spring 2019

Solving Gaussian Elimination In Serial:

The first thing i set out to do for project 1 and to solve the Gaussian Elimination problem in serial. I used many sources to solve the problem, some including the textbook and Chen's github page. After I solved Gaussian Elimination in serial I went back and solved for back substitution. After this I was set out to put this problem into parallel.

Parallelism and Problems:

I set out to parallelize the gaussian elimination problem. I looked at the various loops and decided I would parallelize the inner loop since it does most of the work. I soon realized that parallelizing the serial version I had created would cause race conditions within my code. After testing out my first version, my theory was correct. I was getting different outputs every time I ran it because a thread would change the value of a certain index in the matrix and another thread would use this same value for calculation and caused race conditions if the threads ran in different orders. After this I went back to revise my serial version so that threads did not need to access the same data.

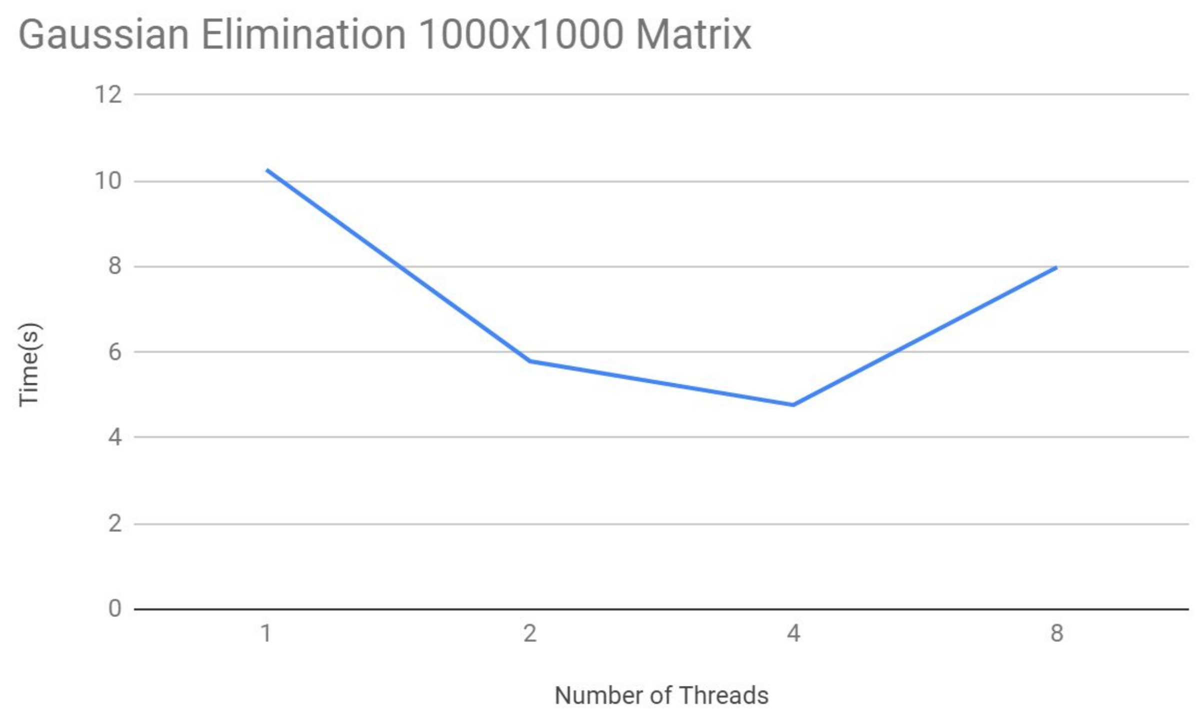
Serial Version Part 2:

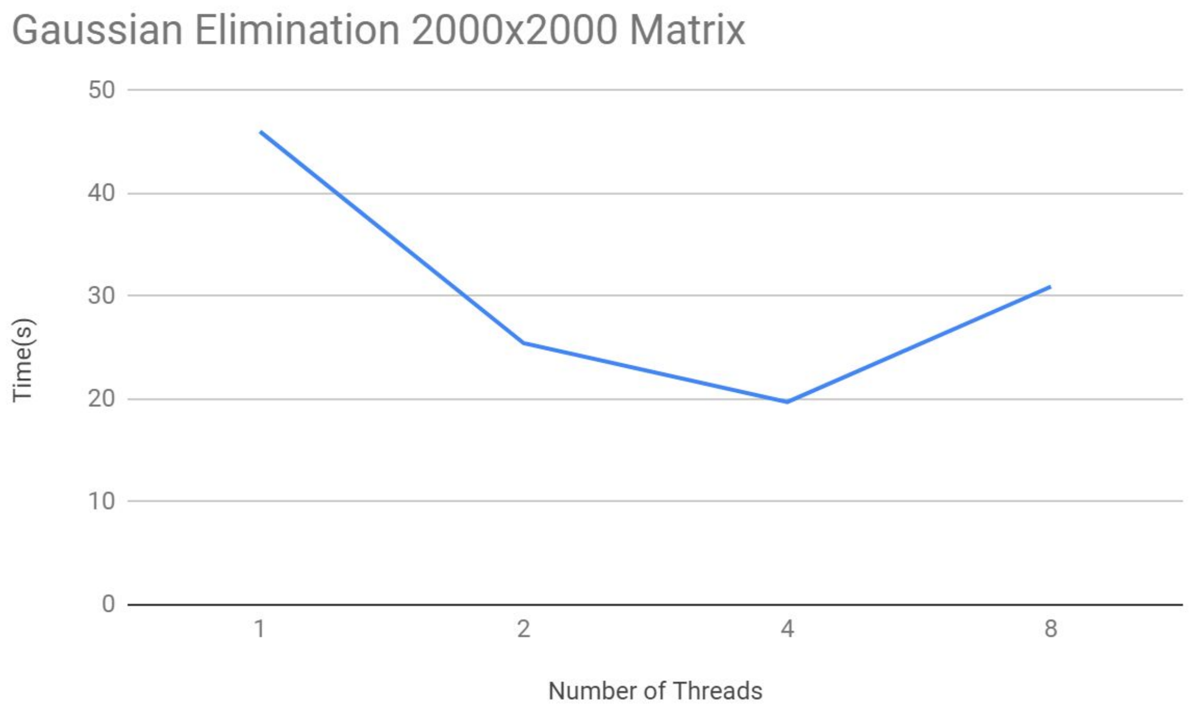
I went back to searching for different methods of doing Gaussian Elimination and decided to do a method that did calculation row by row instead. This method allowed me to parallelize this row calculation because the loop would only go through 1 row at a time, and would never rely on other rows. After testing it in serial, I noticed that this way of doing it requires more steps, but it was easier to parallelize and would run faster in parallel.

Parallel Finish:

After I revised my serial version, I went back to parallelize it. I noticed that I did not have any race conditions but my next problem was how to limit the concurrent threads, since this loop could run over 100 times. To solve this issue I made a variable that keep track of how many threads were created. Once this variable reached the limit for concurrent threads I would stop the loop and wait for all threads to finish, then would continue the code from there. I realized that this would cause slow down though, since all threads are waiting on each other to finish, but I could not figure out another way to solve this issue. I tried using conditional variables to limit the creation of threads, then assign finished threads jobs when they were not working. I could not get this working correctly, but this method could improve performance.

The Results:





Reflection on Results:

After testing 1000x1000 and 2000x2000 matrix on the disci I got some results that I did not expect. I expected that going from 1 to 2 and from 2 to 4 threads would increase performance, but going from 4 to 8 threads decreased performance in both cases. My theory is that this decrease in performance is caused by 1 of the 8 threads getting held up, and because of the way the I coded the project, this could potentially take longer than 4 threads. I did get some cases where 8 threads had good performance but those were rare.

What I learned:

I learned a lot about pthreads and multithreading in general from this project. I also learned how complicated multithreading can get. I think a big thing I learned is how race conditions can be a real problem with multithreading. Multiple scenarios I had to rewrite or rethink how my code works because I was running into these race conditions. Overall, I believe I have a better understanding of multithreading from working on this project.