Programming Project #1
Gaussian Elimination
CS4379: Parallel and Concurrent Programming
CS 5379: Parallel Processing
Spring 2019
Jordan Coe R#11380285

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. After doing some internet searching and some trial and error, I was able to come up with an algorithm that solve gaussian elimination and allowed me to parallelize the inner loop. This algorithm requires more steps than others provided, but if I was able to parallelize it correctly, it should increase performance over the serial version.

Dividing Into Chunks

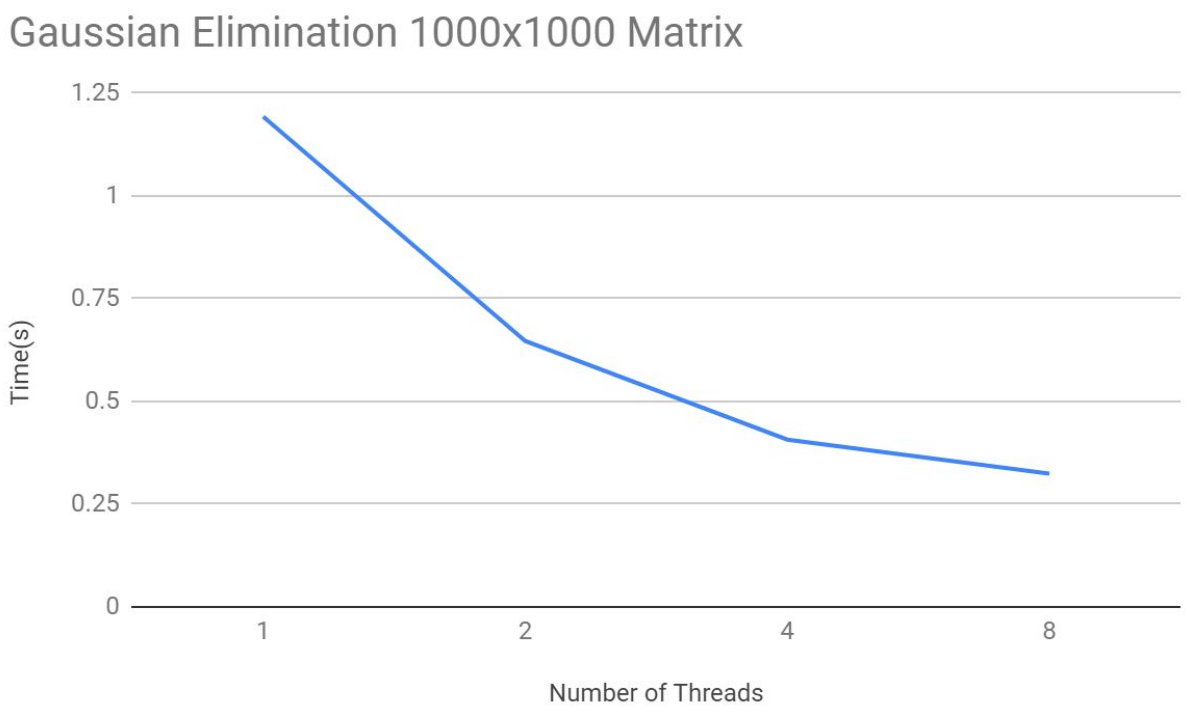
Like the first Question 1, I wanted to divide the matrix into chunks and allow each thread to work on a certain part of the matrix simultaneously. I sat down and worked on how I could come up with a good way to divide the inner loop so that each thread can work on it. This became more complicated than Question 1 because the matrix is more dynamic and would rarely divide evenly among the threads. After doing some calculations and testing, I found a good way to divide the matrix. First, I would need to calculate how many times the inner loop would run. After doing some testing I realized that the inner loop would always run from k to the size of the matrix, so that equation is simply n-k. After that I needed to divide n-k between the threads. This equation turns out to be (n-k)/num_threads. Now the hard part was trying to figure out what to do if that equation does not divide evenly. For example, if (n-k) was 7 and i tried to divide that up

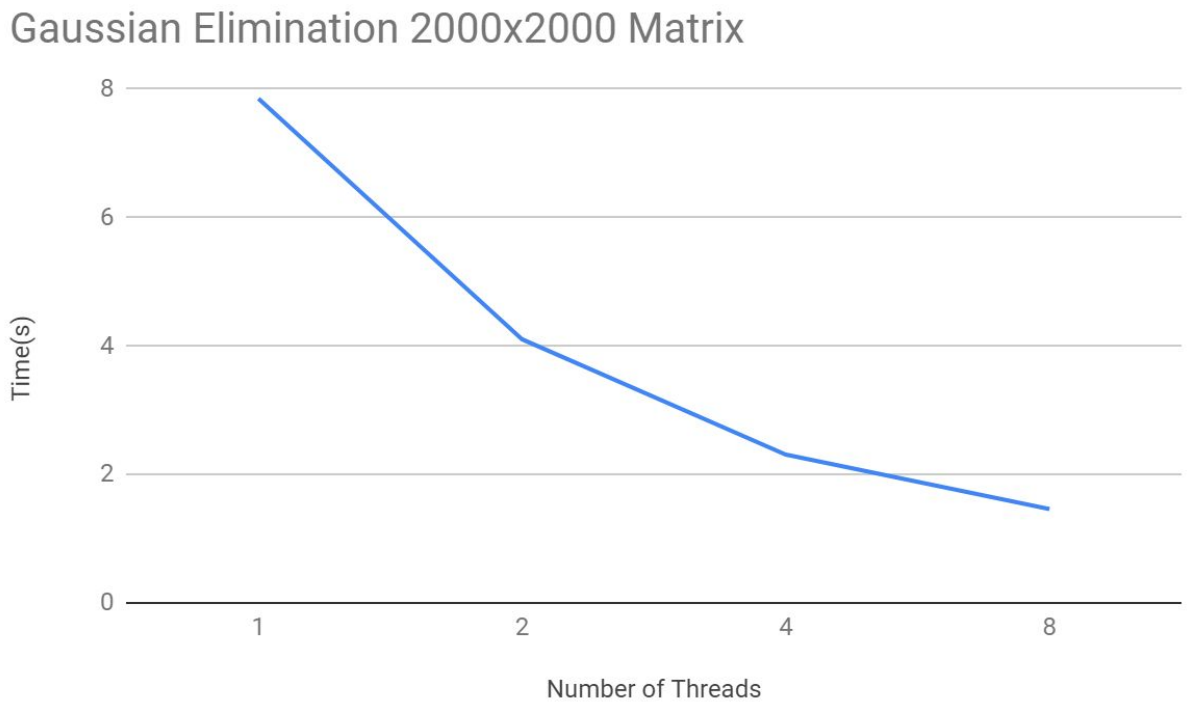
into 4 threads, the equation would be 7/4 = 1 in c. Doing it like this would leave me missing 3 rows in the matrix. After thinking about it, I decided to have a remainder variable that calculates how many threads would need 1 extra iteration. With the equation (k-n)%num_threads I was able to calculate how many threads would need extra iterations. So for the example of a 8x8 matrix using 4 threads my code would:

Number of Iterations = (n-k) = 8-1 = 7
Iteration per Thread = 7/4 = 1
Remainder = 7%4 = 3

So now that I know that 3 threads will need 1 extra iteration. So 2 + 2 + 2 + 1 = 7. The way I utilized this remainder number is I checked if there was a remainder. If there was a remainder than I add +1 to that threads iteration in the for loop and subtract the remainder by 1. Eventually the remainder will be 0 and the rest of the threads will simply use the Iteration per Thread Value.

The Results:





All my results were timed from start to end of the program. This means that the program loss some time because of matrix creation, but i decided to omit asking the user for a matrix when I tested performance, since that would eat up some time. As you can see from the results, performance correlated with number of threads, so adding threads increased the speed. However, the amount of speed increase does lose efficiency when the thread count gets higher. This is because going from 1 thread 2 to thread splits the work in half, and from there gets halved, so eventually increasing thread count will have little improvement on performance.

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.