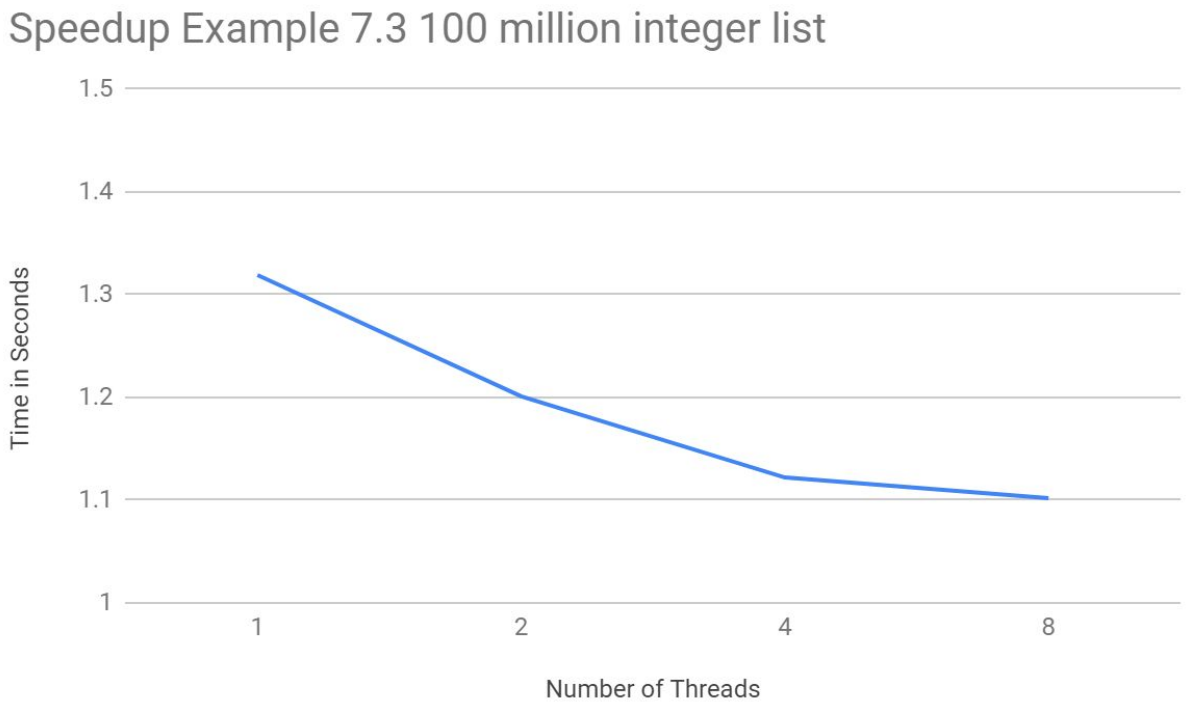
Programming Project #1
Question 1
CS4379: Parallel and Concurrent Programming
CS 5379: Parallel Processing
Spring 2019

Introduction:

Question 1 of the first project required us to experiment with some code from the book. Since this is my first time using pthreads I decided to look up some tutorials first and make sure I had all the libraries necessary to run pthreads. After looking up some example code and running them on my system I felt I had a good enough understanding on how pthreads work.

Example 7.3:

When I first read the description of Question 1 I thought that all that we had to do was copy and paste some code from the book and it would work right away. I realized after looking at the code that we are given the main algorithm, with pthreads incorporated, but we still had to do some of our own coding the get the program to compile and run properly. My first task was to get rid of the current compile errors in the code given. One notable error is the use of MIN_INT on line 8 and 17. After do some thought I realized that this should change to MAX_INT and use the limits.h library. This is because we want to start that variable as high as possible, so that no matter what we get a change in minimum value. If we left it as MIN_INT then that value would be the answer every time. After that I added the mutex locks and wrote the rest of the code. This example helped me better understand how to use pthread_create and pthread_join since this code was not provided. After that I tested my code on disci and go the results listed below.

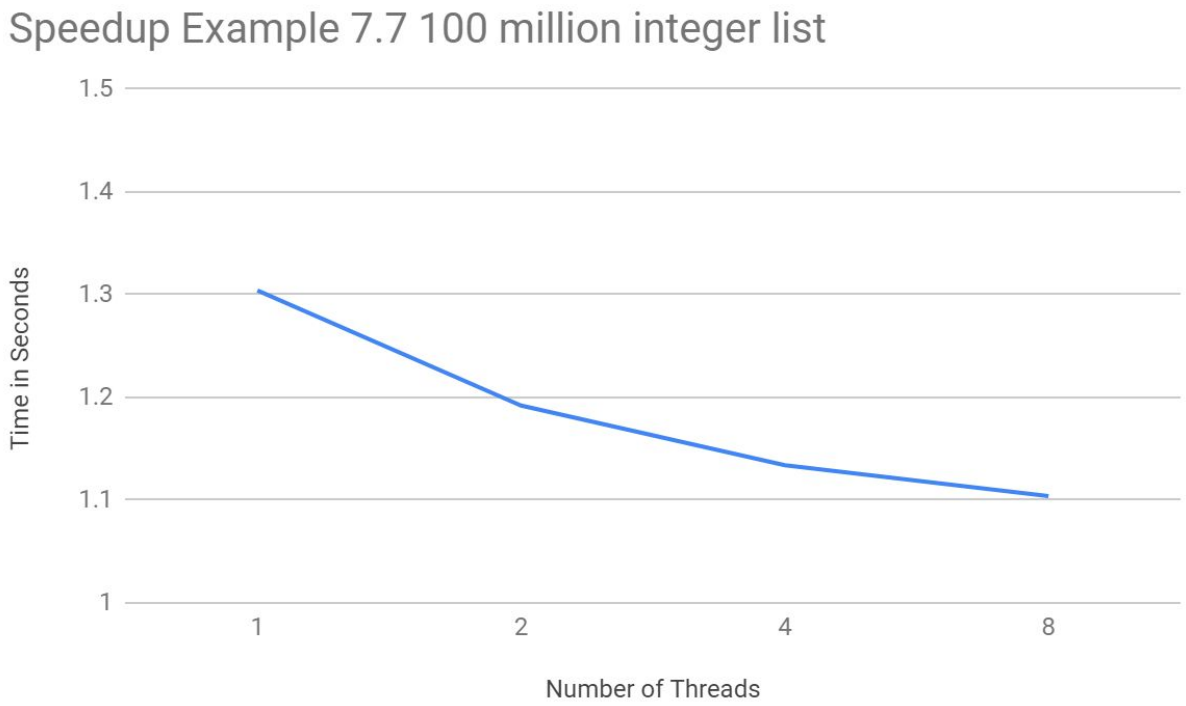


Example 7.3 Results:

The results were calculated and timed from the start to end of the program, which means that time includes the created of the array. These results were what I thought should occur. The biggest jumps between 1 and 2 threads and slowly gets less efficient between threads.

Example 7.7:

This example is very similar to 7.3 but uses read-write locks instead of mutex locks. The read-write locks were harder to understand at first but after dissecting and running the code I understand the process and how it can have performance improvements over mutex locks. Again, we had the framework of the code but we needed to provide the supporting code. This was extremely easy after working on example 7.3 and I got the code running in a few minutes. In theory, read-write locks should improve on performance over mutex locks in this situation but the code was so small I did not see much change. I wanted to test with arrays over 100 million to see a real increase in performance but I was limited because of c array limit. My performance results for example 7.7 are posted below.



Example 7.7 Results:

Example 7.7 had nearly identical results to example 7.3. Add threads did increase performance but performance increase lowered as we kept adding threads. My theory is with a larger array of integers, example 7.7 would outperform 7.3 because it would not need to call write as often.

What I Learned:

I learned a lot about pthreads from these two examples. The hardest part about multithreading is providing locks to account for multithreading and with these examples, I got to see two different ways to work with locks and a situation that if we did not have locks, we could have a race condition. This also gave me some confidence on tackling the gaussian elimination problem, and also made me think about all the race conditions that could occur in the gaussian elimination problem. Overall I learned a lot from these two examples.