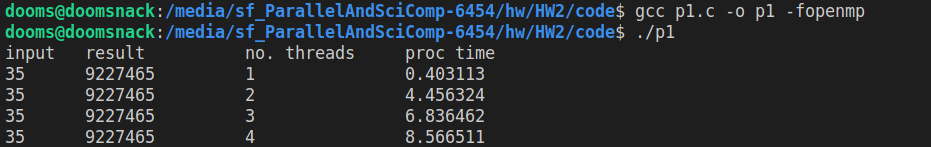
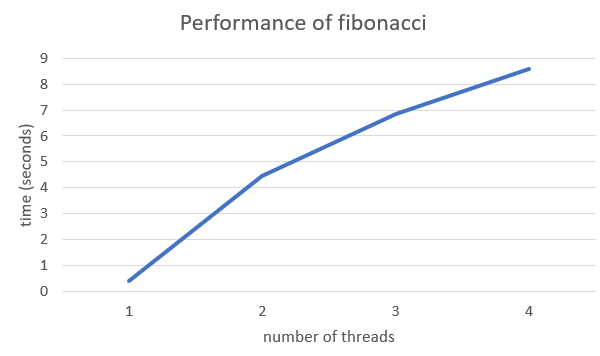
Daniel Murphy  
CSCI 6454 – Fall 2020  
Homework 1 – report

Question 1

Sample input/output:



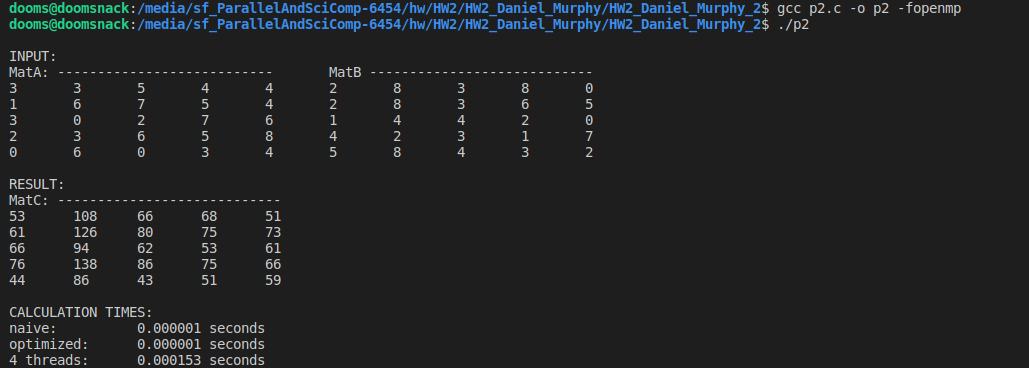


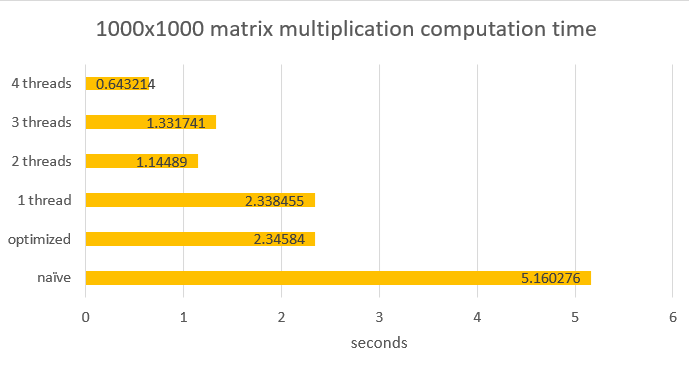
Here we see that increasing the thread count actually increases the time to compute a recursive Fibonacci. This is likely because recursive Fibonacci is a fundamentally linear equation- each iteration requires the output of the previous iteration, so instead of improving performance, parallel processing only adds the overhead of thread management.

One can envision a more efficient algorithm that stores previously calculated results in a hash table- this might eliminate some expensive calculations and may be a better candidate for parallelization.

Question 2

Sample input/output   
(note: the program only prints the input and result matrices for matrix sizes <= 100x100)





I found the naïve implementation to be exceedingly slow. Improving performance required taking into consideration the locality of the data. By transposing the MatA matrix in the optimized implementation, significant improvements could be made.

To realize a parallel implementation, it was necessary to write the results to the MatC matrix in transpose form to ensure that separate threads were not attempting to writing to contiguous memory locations. Fortunately, the extra overhead of transposing MatC back requires relatively insignificant overhead.