Aerial Robotics Kharagpur Task 2.1

Raghav Aggarwal

I. INTRODUCTION

The problem was to optimise an already written code as best as we can and to minimise the time taken by the code to run. I tried several approaches to the problem to minimise the runtime.

II. PROBLEM STATEMENT

In the first task you have to use optimisation techniques to reduce the running time of the code provided to you. First download the zip file here. Before proceeding further, read the Instructions.md file which provides a lengthier explanation of the problem statement along with hints. Markdown(md) files can be viewed on Typora or on your code editor. A bash script to time your code has also been provided to you along with instructions on how to run it. Optimisation techniques can range from basic time complexity reduction to more advanced optimisations using parallel programming, locality of references and the like. Initially the code provided to you will not work for larger values of n, we will slide n from 4 to 1000 to test your code. Code working on large values of n in less time will fetch maximum points. Start with basic time complexity reduction and ease into more advanced methods to see how your running time reduces and have fun trying to achieve the minimum running time possible!

III. INITIAL ATTEMPTS

As mentioned in the instructions first part was to be done by saving the whole 2d array but the problem was to how two calculate each cell value more efficiently. In the second part my initial attempt comprised of going through the internet trying to learn different algorithms for matrix multiplication. For the third part I decided to convert as many arrays I could convert into vectors.

IV. FINAL APPROACH

Part 1 :

Build 2d arrays which stored all the minimum and maximum values so the program did not have to calculate the whole path every time. This was achieved by calculating each cell's value by comparing the values of the cells one column ahead and one row ahead (with the exception of extreme cells). This resulted in a very efficient calculating of the whole path's minimum and maximum cost which was then stored in the cell to be accessed whenever called.

Part 2 :

Since the matrix multiplication algorithms were not giving very efficient results I decided to calculate an Algorithm specific to this program by myself and ended up making the runtime go from 6 seconds to 70 milliseconds for n=1000. It included filter arrays direct multiplication with the columns of matrix B reducing it from a nxn matrix to nx4 matrix drastically reducing runtime.

Part 3 :

Most of this part was already handled by the part 2 since it was now already spitting out values multiplied by the filter array into final answer array. Still I decided to make as many arrays I could into vectors.

V. RESULTS AND OBSERVATION

After handling the part 1 of the task the time taken reduced from 2.8 seconds to 54 microseconds for n=12. Then i took the value n=1000 and started reducing the time complexity from the matrix multiplication. I did the part 3 first by converting many arrays into vectors and the time taken initially reduced to 6 seconds for n=1000. After doing the part 2 and applying the finalmatrix producing algorithm i got the run time from 6 seconds to 72 milliseconds.

CONCLUSION

The problems main aim was to increase efficiency of the code. This process is helpful in writing real life codes which are dictated any time taken by the program to run. These small blocks of codes make up an entire project and to make the project more useful and widely applicable it is important that it doesn't take a lot of time to perform the desired task. Hence optimizing these small blocks of codes of code is important as they return the efficiency in user's ease of using the product