Aerial Robotics Kharagpur Documentation Task 1

Agnibha Sinha

***Abstract*— This is a documentation for the "Optimize Me" task of ARK where the given code has been optimized in 3 steps using different optimization techniques like parallel computation.**

I. PROBLEM STATEMENT

The task is to optimize the given code. The code can be mainly divided into 3 parts. First, 2 cost matrices, the minimum and maximum are calculated using recursive functions. Secondly these 2 matrices are multiplied. Lastly, a dummy filter is applied on the product matrix. Each of these parts have been optimized separately.

II. INITIAL ATTEMPTS

The final approach was mostly the same as the initial approach. For the 2nd step of matrix multiplication, Strassen's Matrix Multiplication method was used initially to reduce the complexity from n^3 to $n^{2.8}$ but the process was very long and complex and not worth the reduction in complexity. The multiplication by taking transpose was easier to understand and also efficient as it provided sequential access to elements stored in the memory.

III. FINAL APPROACH

The final approach has been divided into 3 steps.

Step 1: This is the step which majorly optimizes the given code and makes it work for large values of sizen greater than 100. 2 arrays have been used to keep record of the already calculated minimum and maximum costs. This reduces the number of recursive calls for recalculating the same values significantly. The arrays are initialized with -1 and before recursively calling the function we check if the value at that index is -1 or not and return the value directly from the array in case it is not -1. Also the values calculated for the first time are updated in the array.

Step 2: This step optimizes the matrix multiplication used by providing sequential element access. The values of the cost matrices are stored in the matrices first instead of directly calling functions during multiplication. The transpose of the 2^{nd} matrix is taken and used for multiplication row wise as it provides more sequential access according to the way data is stored in memory.

Step 3: This step optimizes the filter used. In the original program, one element of the final array is calculated at a time. In my approach, the values of the elements in the final array get calculated parallely and this saves computation time. We move to the ith row in the product matrix and perform dot product of that row with the corresponding filter row (i%4) and add it to the i/4 th row of the final matrix.

IV. RESULTS AND OBSERVATION

Without any optimization, the code was not working for sizen greater than 20.

sizen	Step 1	Step 2	Step 3
100	0.0579	0.0555	0.0469
200	0.1967	0.1693	0.1091
300	0.4982	0.2754	0.1973
400	1.7136	0.6903	0.4622
500	2.4382	0.9285	0.7733

CONCLUSION

The overall program was highly optimized. Still there may be scope for further optimization. I learnt many new things while performing this task.

REFERENCES

[1] https://www.geeksforgeeks.org/strassens-matrix-multiplication/
[2] https://stackoverflow.com/questions/1907557/optimized-matrix-multiplication-in-c