Assignment 1 – Matrix Multiplication

CO2206 – Operating Systems

NAME : CHATHURANGA HKAG

INDEX NUMBER : 17/ENG/016

REGISTRATION NO : EN 86135

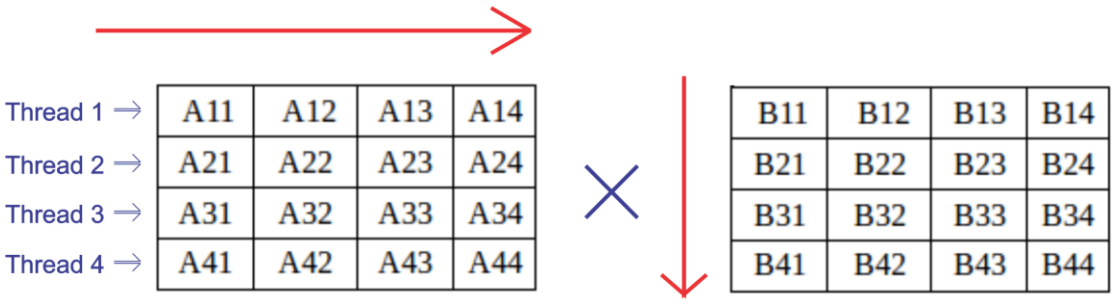
DATE OF SUBMISSION : 12/11/2019

Introduction

There is a project that I created for multiply 2 matrices. When we run the code firstly we can see program asked the dimension of the first matrix and second matrix. After user enter the dimentions , programis checked the whether the two matrixes are multipliable. If two matrixes are multipliable, program is continued. Else, program is shown a message and shown please re enter the dimentions.

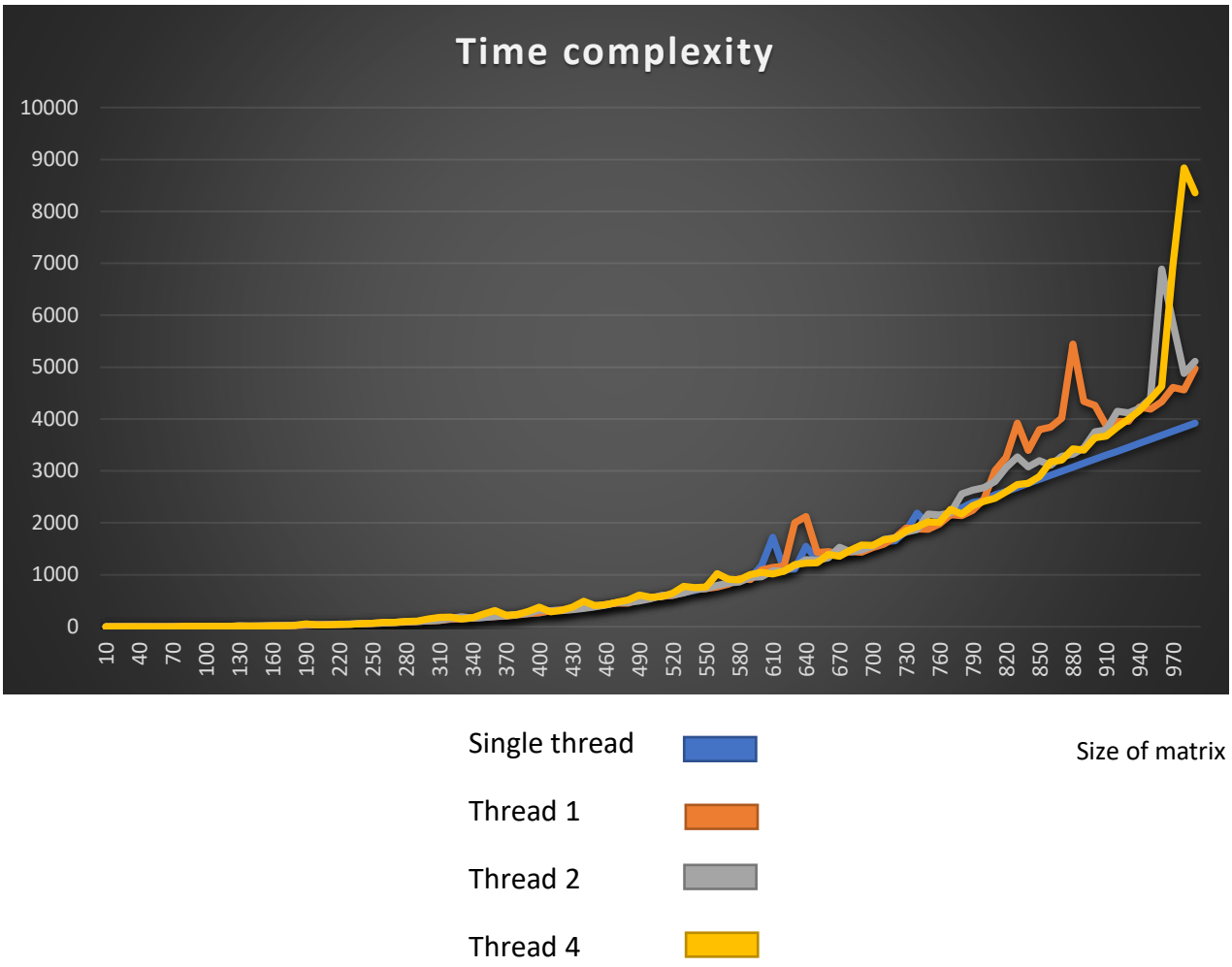
After enter dimentions correctly, program is asked 'Multiply the two matrices using a single thread' or 'Multiply the two matrices using multiple threads' from the user. So user can select first or second method. If user enter first method he should create 'file_1.txt' and 'file_2.txt'. File_1.txt should have first matrix and File_2.txt should have first matrix. If user select second method, user not need to create any text files, because program auto create a matrix and fill it using random numbers. According user entered data, finally program output the Resultant matrix.

When we consider about second method, We create different threads, each thread evaluating some part of matrix multiplication. Depending upon the number of cores your processor has, you can create the number of threads required. Although you can create as many threads as you need, a better way is to create each thread for one core. we create a seperate thread for each element in resultant matrix. Using pthread_exit() we return computed value from each thread which is collected by pthread_join(). This approach does not make use of any global variables. In this main.c file we create only 4 threads for row by row.It is shown below.



Since we are working with quite big matrices (1000x1000), it is better to just use the heap instead of the stack. For this reason, we use two dimensional array of pointers. Because it's size can't be known at compile time it will be initialized in the heap, not in the stack. We are going to graph computation time for the two options. In that case we want all time according to the matrix size. I implemented the main2.c file for that. I used for loop by changing matrix size 10 by 10 to 1000. I changed number of Threads and graph the all of data. Graph is shown below.

Calculation Time



At the beginning of the chart, the single execution in the main thread of more efficient because the workload is very small. The multi-thread execution is very slow because, even that they work in parallel, the workload is so small that it doesn't compensate for the overhead of creating, initializing and joining the threads.

Nevertheless, as the workload increases (the matrices get bigger) the multi-threading options gets better and better. This is, obviously, because each time, more and more work can be performed in parallel and the overhead is very small in comparision to the calculation time.