CS3510: Operating Systems - 1
Programming Assignment - 1
Report

Peram Sree Keerthan Reddy
EE20BTECH11040

We need to use multi-processing to find a list of tetrahedral numbers.
As given in the assignment, we have the values of N i.e, the numbers until which the program needs to run and K, which is the number of processes. We create separate files for each process (Eg: "output1.txt has the outputs of the numbers processed in the 1st process). Finally, the file OutMain.txt has all the identified tetrahedral numbers along with the process number that identified it.

The program is written as follows:
We open the input file (throw an error in case it doesn't open).
We read the values of N and K from the input file and validate it. If it is invalid, we throw an error.
We then create a matrix dividing the numbers among the K processes equally. We have a N*N matrix all initialized with 1.
However, the matrix we need has K rows and N/K columns, if K perfectly divides N or N/K + 1 columns if it doesn't.
If the number is not within the range, we assign the space to be -1.
The load is distributed such that each process gets almost equal numbers.
Eg, if we divide 27 numbers among 5 processes, we have:

P1	1	6	11	16	21	26
P2	2	7	12	17	22	27
P3	3	8	13	18	23	-1
P4	4	9	14	19	24	-1
P5	5	10	15	20	25	-1

We then set up shared memory for each of the child processes. We have created one shared memory ("SharedMemory1"), to store the number and the identifying process. We have another shared memory (SharedMemory2) to store the offset.

We then fork and create child processes. For each child process, We create a separate file for each process to add the numbers and a statement corresponding to whether it is a tetrahedral number or not. We open the shared memory and then do a check on whether it is a tetrahedral number. If it is, we add it to the shared memory, and increment the pointer in the second shared memory by 2. We use the pointer in the second shared memory to find the offset in the first shared memory to update the values. The number is then added to the corresponding process file along with the statement that “it is a tetrahedral number.”
If it’s not a tetrahedral number, we add the number to the file along with the statement that “it is not a tetrahedral number.”
The child process has to exit to avoid looping.

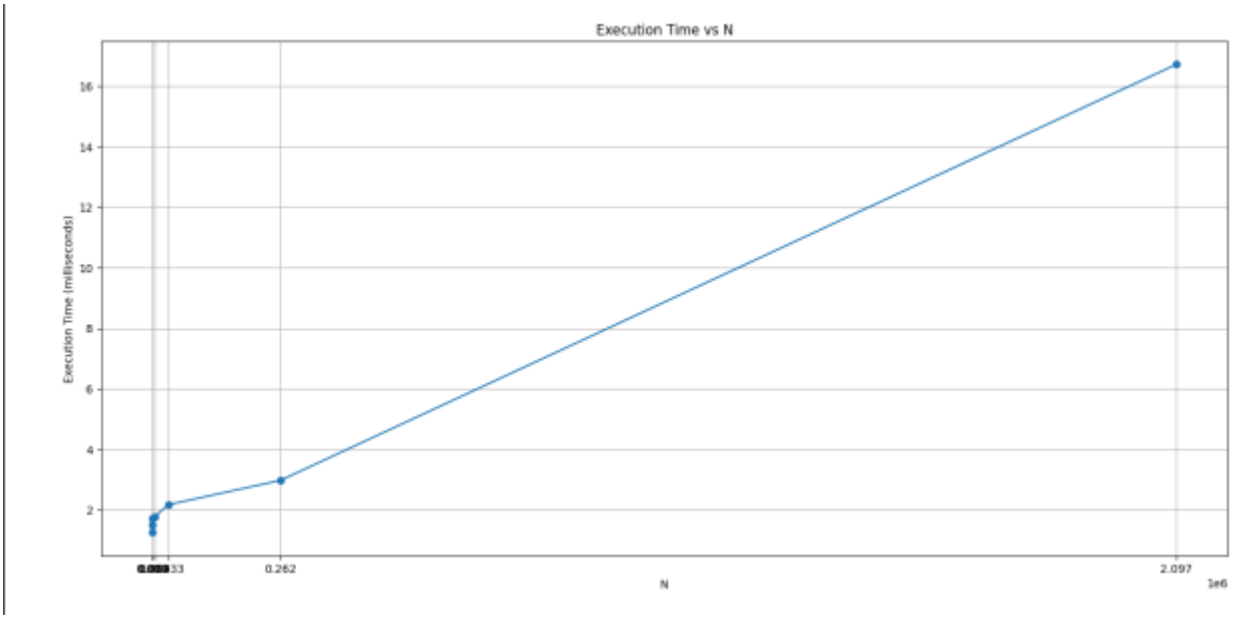
The parent process waits until all the child processes complete execution.

After termination of all the processes, the main memory opens its OutputMain.txt file where it adds all the tetrahedral numbers and the process that identifies it that is stored in SharedMemory1. This is done using the offset that is stored in SharedMemory2, which is used as a looping condition.

Analysis:

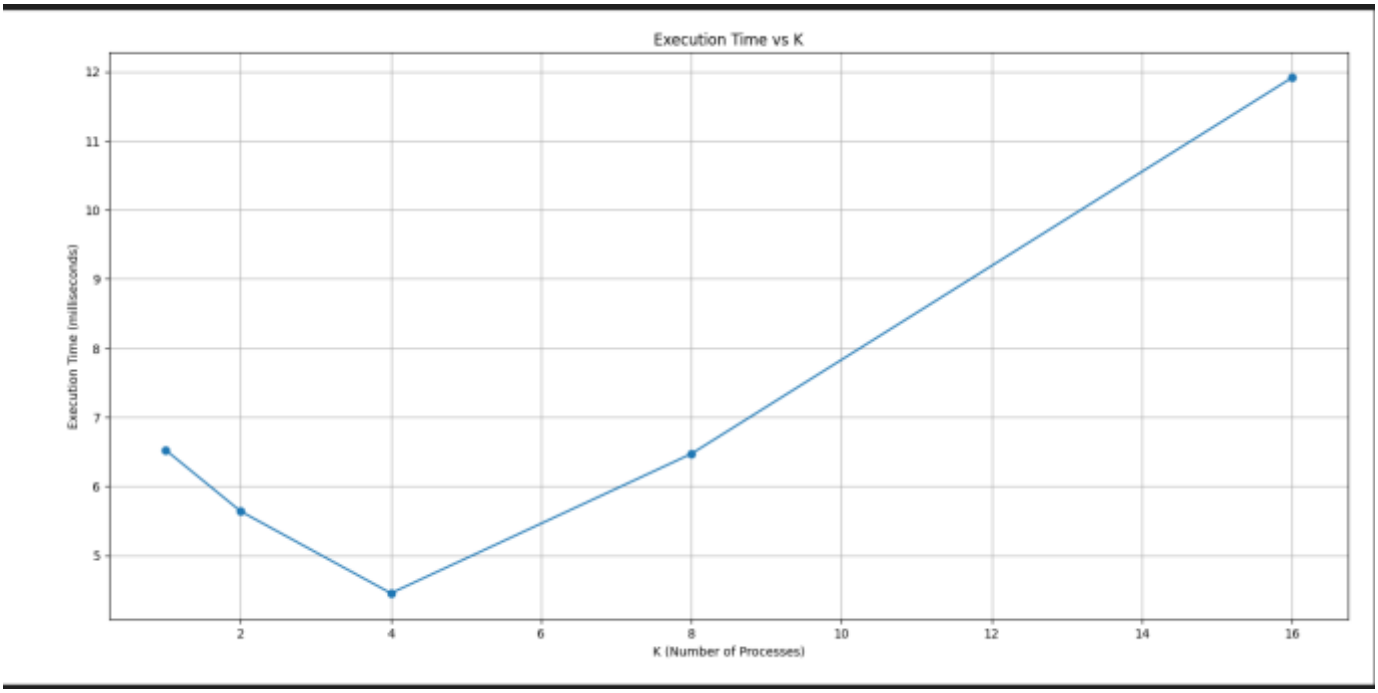
Plot for execution time v/s N

Here N = 2^(3n) where n ranges from 1 to 7.



We see that as the number increases the execution time increases linearly for the same number of processes increases. It is obvious that we have a linear relation here.

Plot for execution time vs K



Here K varies as 2,4,6,8,10.. Etc. Initially, we see that the execution time decreases. This is because for the same N, more processes do the computation parallely. However with further increase in the number of processes, they all contest for the same shared memory, increasing K may lead to contention and the need for synchronization mechanisms. Thus, as a result, the time taken increases in a practical scenario.