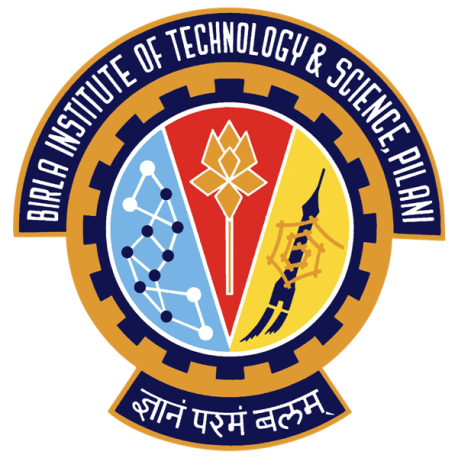
**OPERATING SYSTEM (CS F372)**

SECOND SEMESTER: 2021-22

Assignment 2 Report

Group 25

Creating Multithreaded input and addition procedures scheduled by round robin and pre-emptive priority scheduling



Department of Computer Science and Information Systems BITS Pilani Hyderabad Campus

**Date of Submission:** 15/4/2022

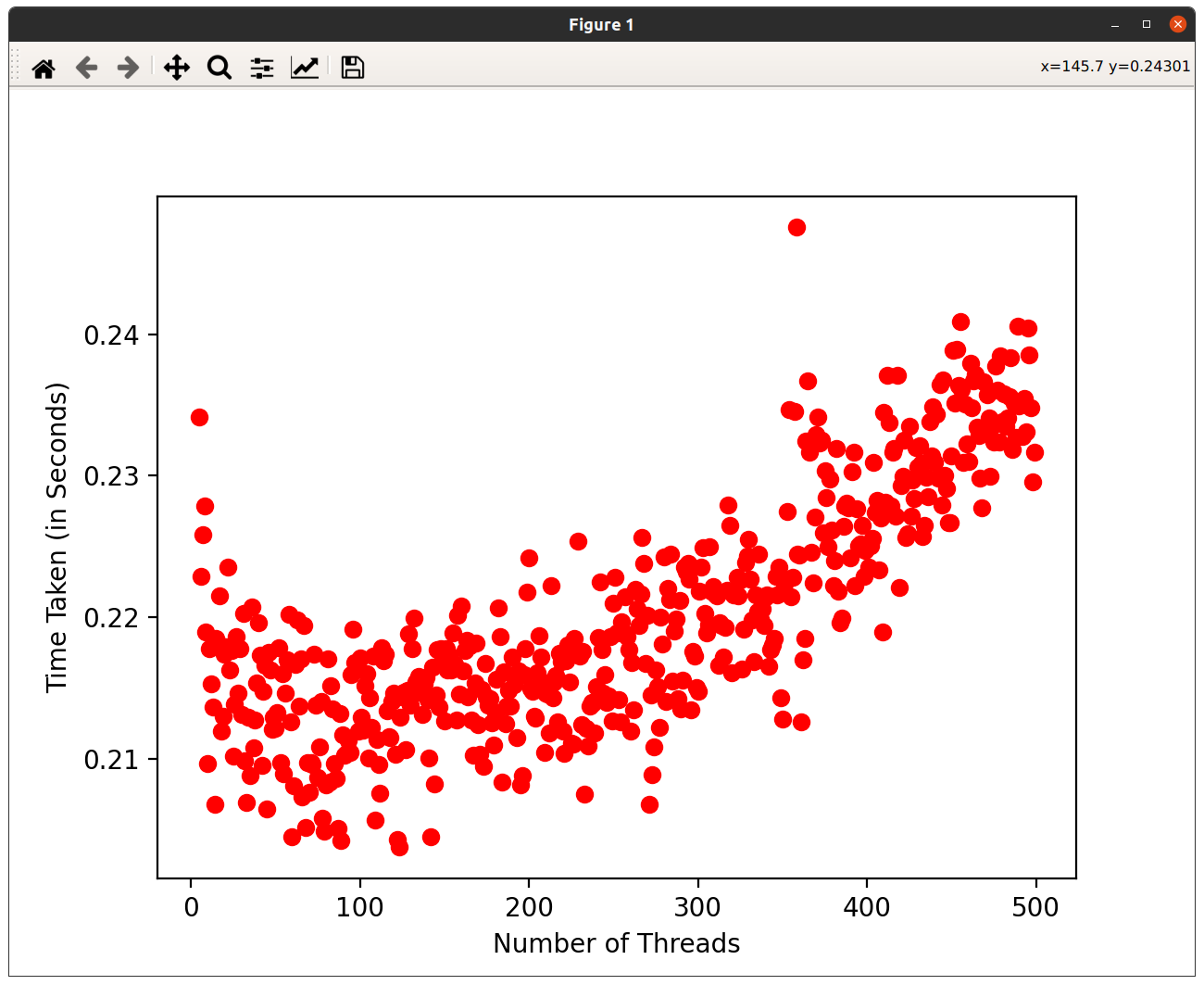
**Problem Statements:**

1. Write a multi-threaded C program (P1) that reads a file containing space separated positive integers in ASCII format. Some example input files are [here](https://drive.google.com/file/d/1hL-yDBNLuClF6ssMlm-r2fKcMga3nQD6/view?usp=sharing). Please generate more input files in order to generate the plots as required below. The number of integers can be arbitrarily large (upto 1 million integers). The name of the file and the number of integers in the file will be supplied in the command line
   1. Different threads should read different parts of the file. Vary the number of threads from 1… to arbitrarily large.
   2. Record the time that it takes to read the entire file into memory with different number of threads (1, 2, … n)
   3. Plot time against the number of threads for different numbers of integers in the input file. Analyze the findings from the plots.
2. Write a C program (P2) which uses IPC mechanisms to receive the numbers read in the program in part (a). This program spawns multiple threads to sum the numbers passed by the program P1 in part (a). The program prints the sum on the terminal
   1. Vary the number of threads from 1… to arbitrarily large. Different threads can calculate the partial sums of the series
   2. Record the time it takes to sum the integers with different number of threads
   3. Plot the time against the number of threads for different numbers of integers. Analyze the findings from the plots.
3. Write a scheduler program S. S spawns 2 children processes which exec to become the processes P1 and P2 in part (a) and part (b). S uses some mechanism (e.g. sleep) to simulate a uniprocessor scheduler. That is, it puts Process 1 to sleep and lets Process 2 execute, and vice versa. Simulate the following scheduling algorithms in S:
   1. Round Robin with time quantum 1 ms
   2. Pre-emptive priority scheduling, where the priority is a function of the amount of processing time that a process has already received (lesser the processing time received, higher the priority). Priority is updated every 1 ms.
   3. Plot the total turnaround time vs workload size and waiting time vs workload size for the different scheduling algorithms. How do the two algorithms compare for the same workload size? Analyze your findings.
   4. What is the switching overhead in the different cases?
4. You might need to take care of race conditions which might arise at different parts of the assignment.
5. For taking input, based on the size of the file, the file is divided into smaller segments and each thread reads a separate segment of file, which is then used to get the sum of all the numbers in the second part of the assignment.

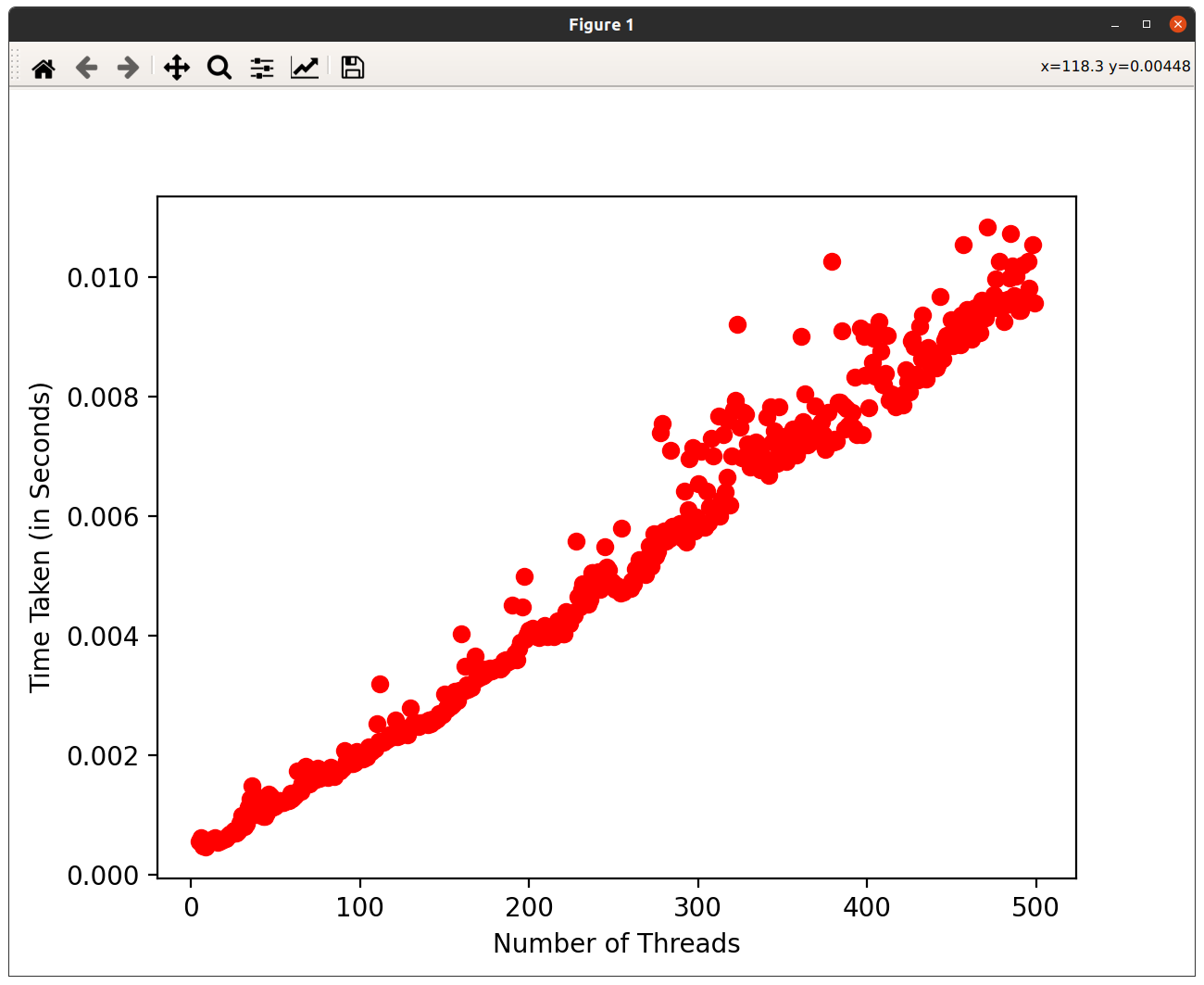
On varying numbers of threads on different sizes of input file different graphs were observed.

For the file containing million integers, the time taken to read whole file first decreases as achieving sub-result , then increases as there is a limit of CPU to perform a limited number of tasks in parallel, moreover creating and allocating memory for a lot of threads also increases the time.

For the file containing, hundred integers, a sudden increase in time can be observed on increasing number of threads as increasing number of threads decreases the size of each segment and each segment becomes significantly small that increasing the number of threads becomes redundant.



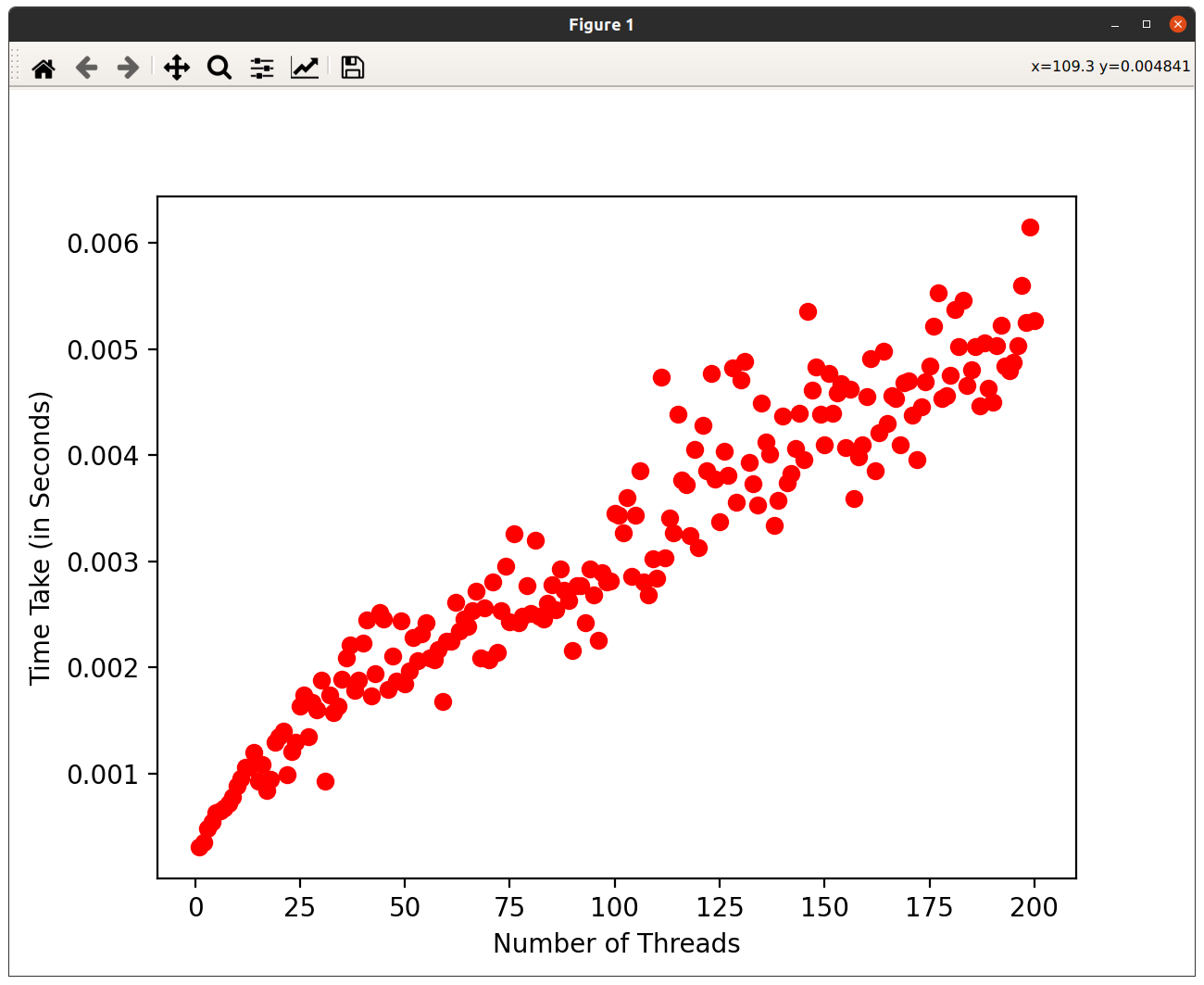
Time VS Number of threads for file containing million integers



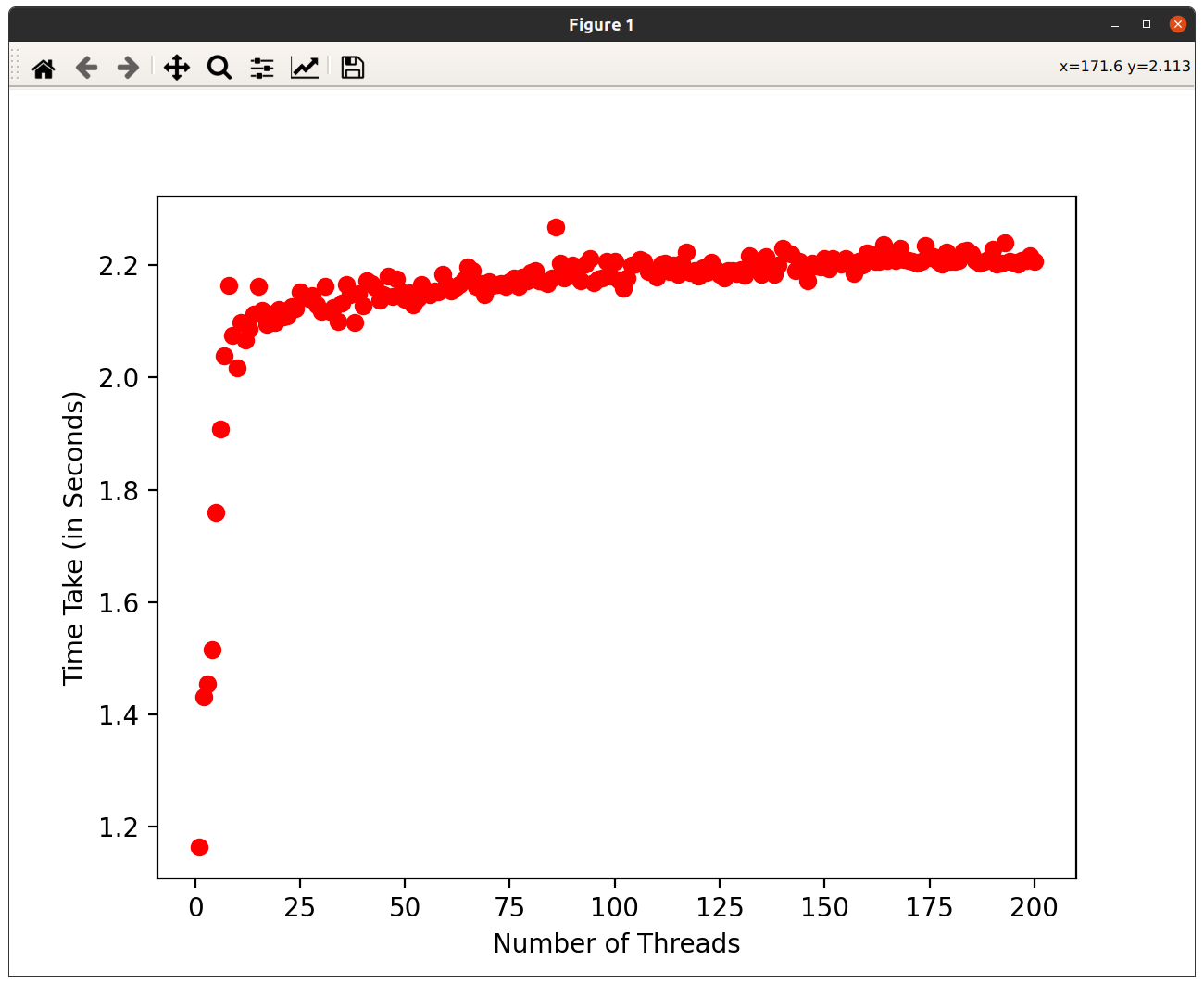
Input Time VS Number of threads for file containing 100 integers

1. Inputs from the first part are then supplied to this section, then multithreading is used to calculate the overall sum of all the numbers. On varying the number of threads a similar pattern is observed in this section as well.

For the file containing, hundred integers, a sudden increase in time can be observed on an increasing number of threads as increasing number of threads decreases the size of each segment and each segment becomes significantly small that increasing the number of threads becomes redundant. For the file containing, million integers the time remains almost constant on increasing the number of threads.

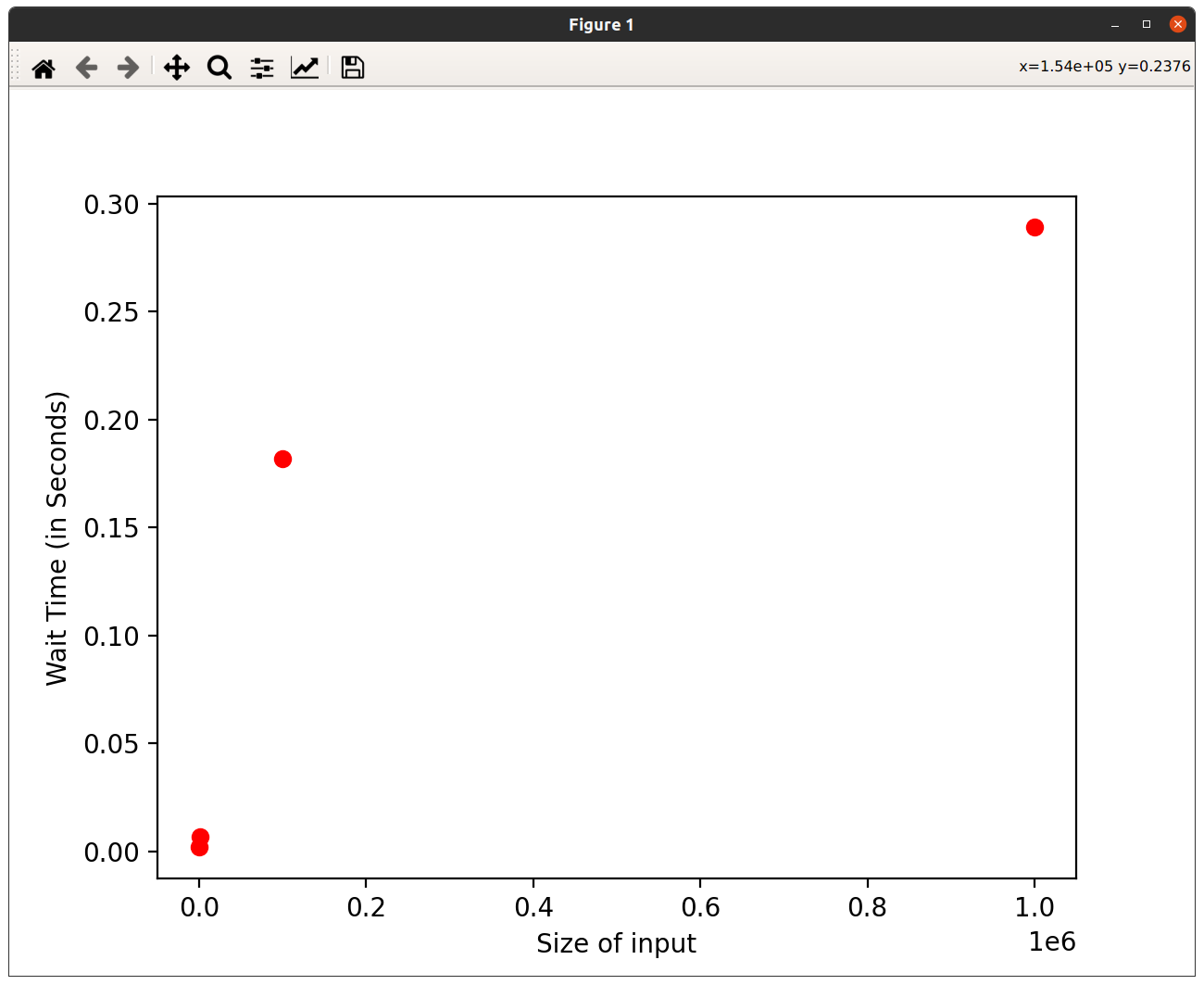
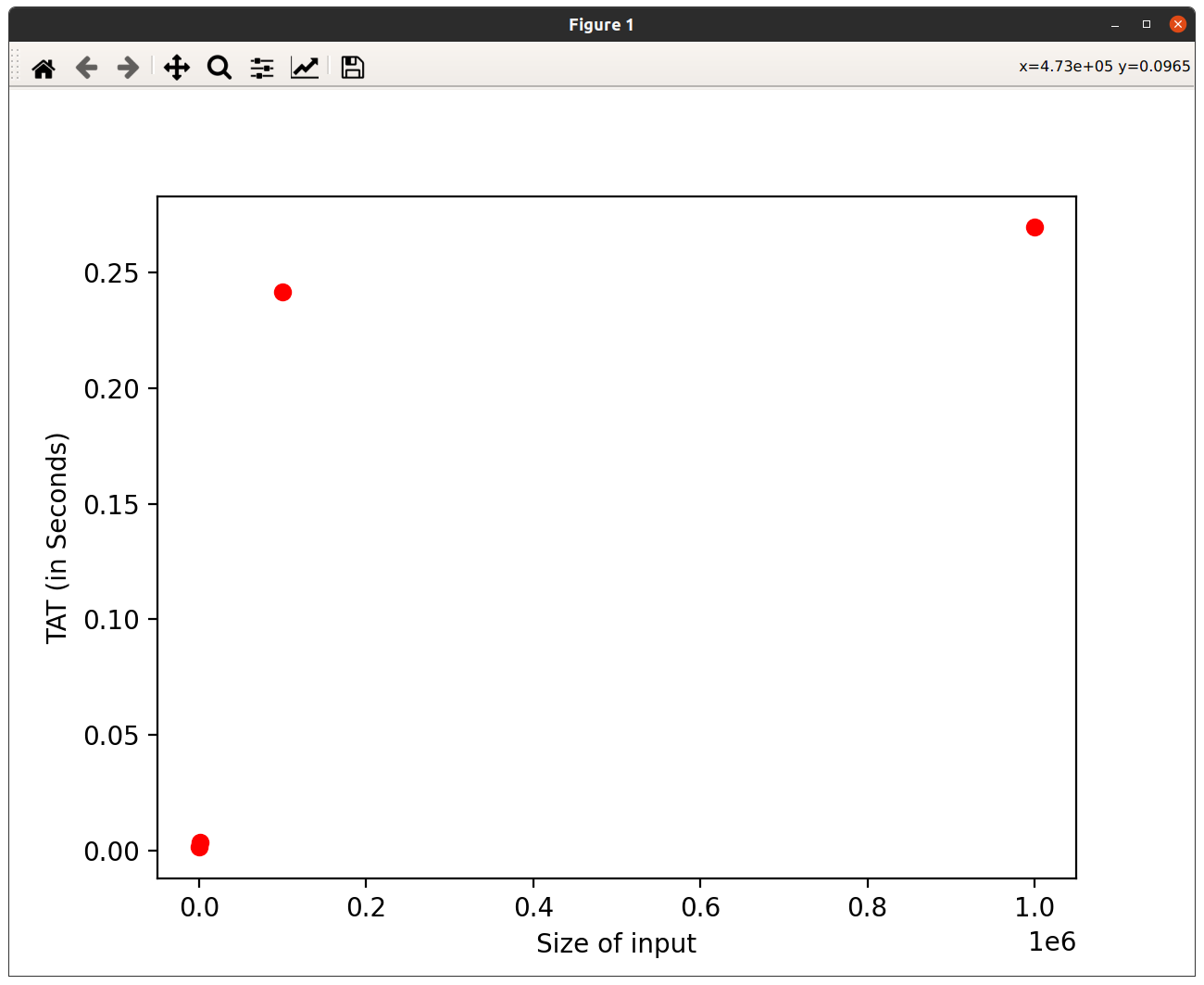


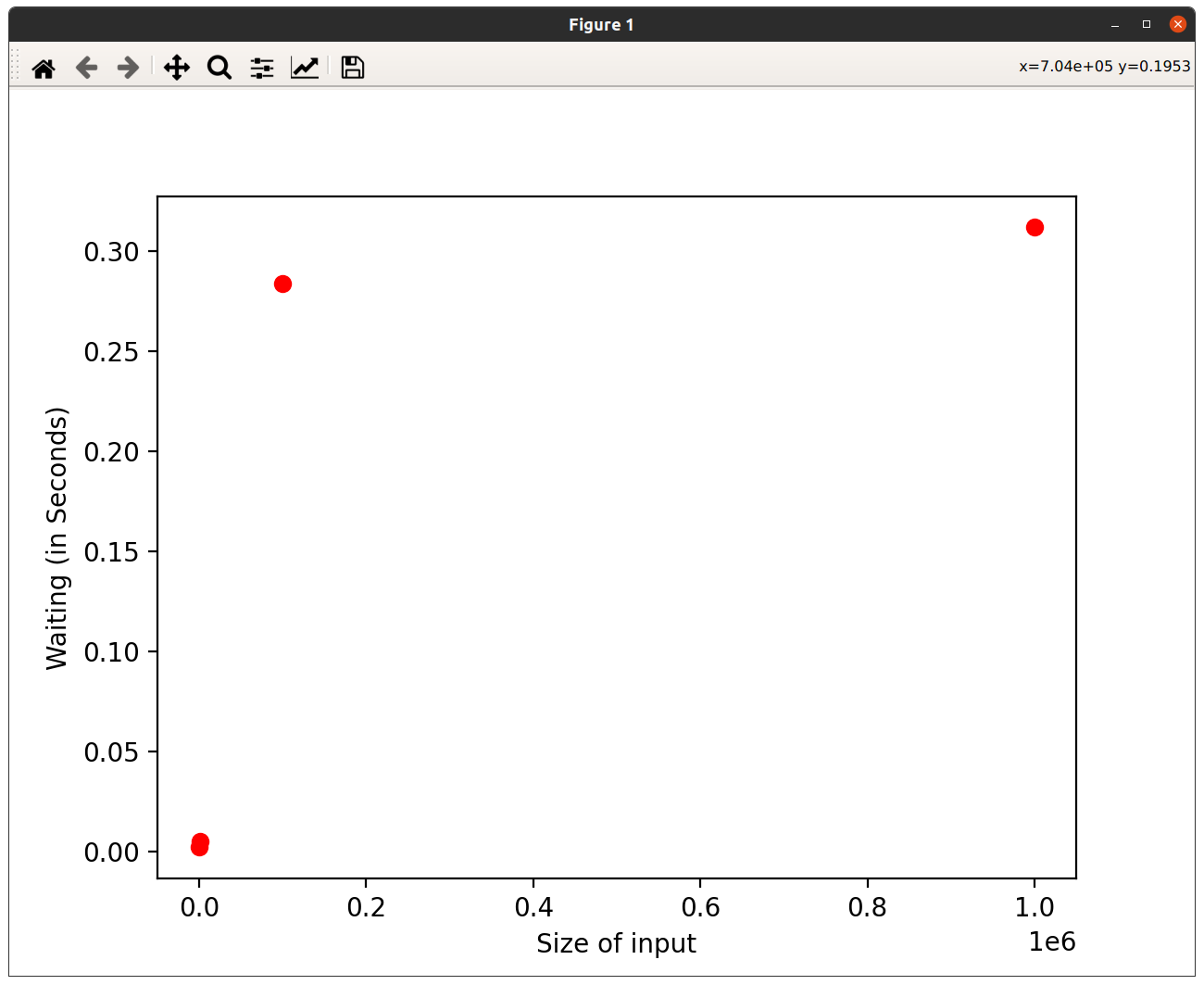
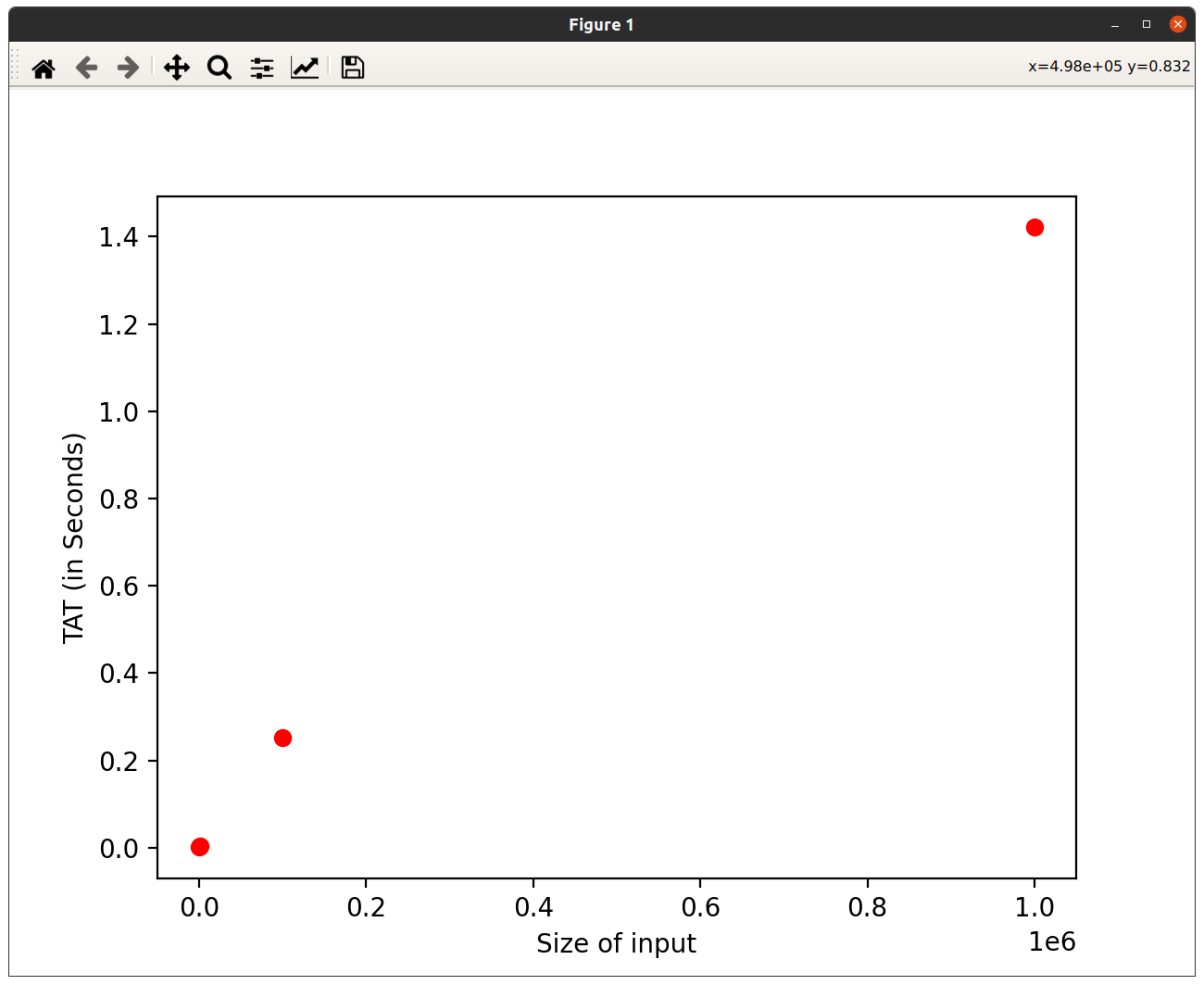
Sum Time VS Number of threads for file containing 100 integers



Sum Time VS Number of threads for file containing million integers

3. In the third part, a Round Robin was created which starts with taking i/p for 1ms then switches to get the sum of those inputs.





Observations:

For Input:

Size: TAT Waiting Time

100: 0.001312 0.002

1000: 0.003281 0.007

100000: 0.241651 0.274

1000000: 0.269604 0.329

For Sum:

Size: TAT Waiting Time

100: 0.001003 0.002

1000: 0.003924 0.005

100000: 0.253183 0.284

1000000: 1.421524 0.312

4. The mutual exclusion had been taken race condition had been taken care of completely both during input and getting sum as during input the threads add the inputs in different locations thus removing the case of overwriting, and in the case of getting sum as each threads store get subresult in different with is then merged in the main function to get the result thus again removing the case for race-condition.