Contents

Chapter 1: Introduction

Code Overview: Our code uses semaphores and shared memory to simulate round robin scheduling.
To achieve proper synchronization in between the two threads, we are using turnstile-design-pattern of semaphores.
We maintain a shared memory variable that tells which process' turn it is.
When a process identifies that its turn is over (by checking the shared memory variable), then before switching, it puts all of its threads to waiting state using wait() and only once all of its threads have gone to sleep, it switches to the other process' threads using signal(). Before P1 and P2 begin their execution, we are performing preprocessing on the input matrices. During preprocessing, we are performing two tasks:

1) The transpose of second matrix is taken and saved in a separate file called matrix2-transposed.txt

2) The offset of each line in the input matrix is saved in an array which will be accessed later by the reading process so that the reading processes can work in parallel.

Each thread of P1 performs equal amount of work. The amount of work done by P1 is number of rows divided by the number of threads P1 has.
Each thread of P2 performs equal amount of work as well. The amount of work done by P2 is number of cells in the final matrix divided by the number of threads P2 has.
So each thread in P1 and P2 has a predecided amount of work they have to do which is passed as an argument to those same threads.

Chapter 2: Initial Plots

2.1 Time of P1 process vs Number of threads

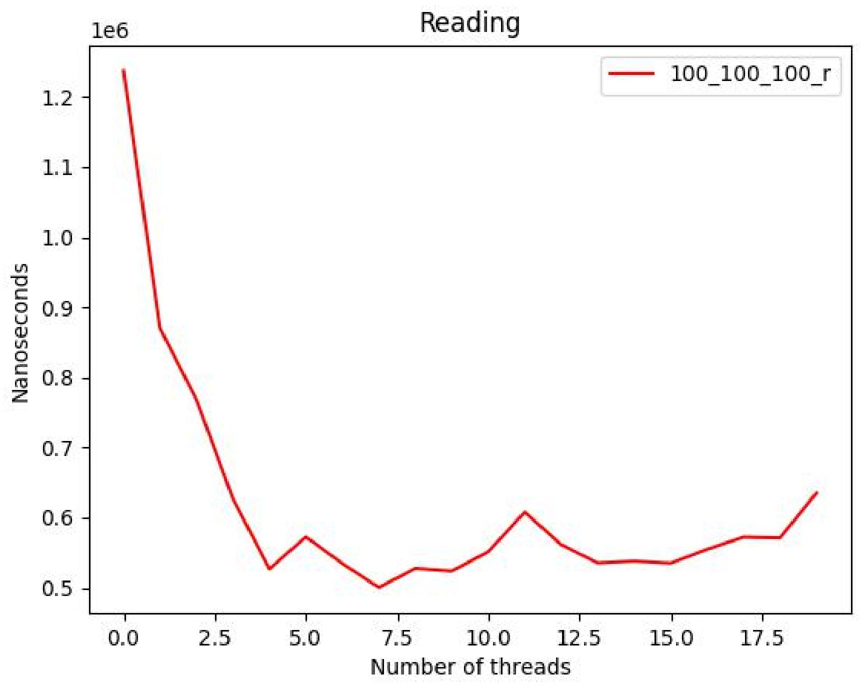

Figure 1: For Matrix Size: 100x100 and 100x100

The above graph if for two input matrix of 100x100 size and we see a sharp drop in obervation time until 10 threads as the graphs were taken on a system with 10 cores. It took on average (across 100 iterations) 1236982.44 nanoseconds for reading the combined reading for both matrixes and 551278.3 nanosecs for 10 threads after which the graph stabilizes.

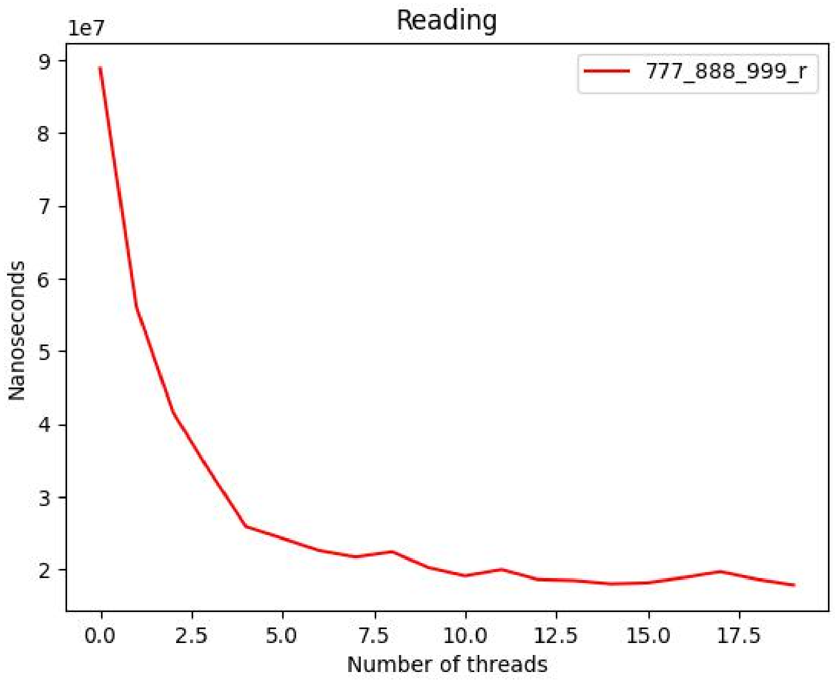

Figure 2: For Matrix Size: 777x888 and 888x999

The above graph tells us that the combined read time for the both of the matrixes was 88866258.3 while single threaded and 20288941.5 with 10 threads.

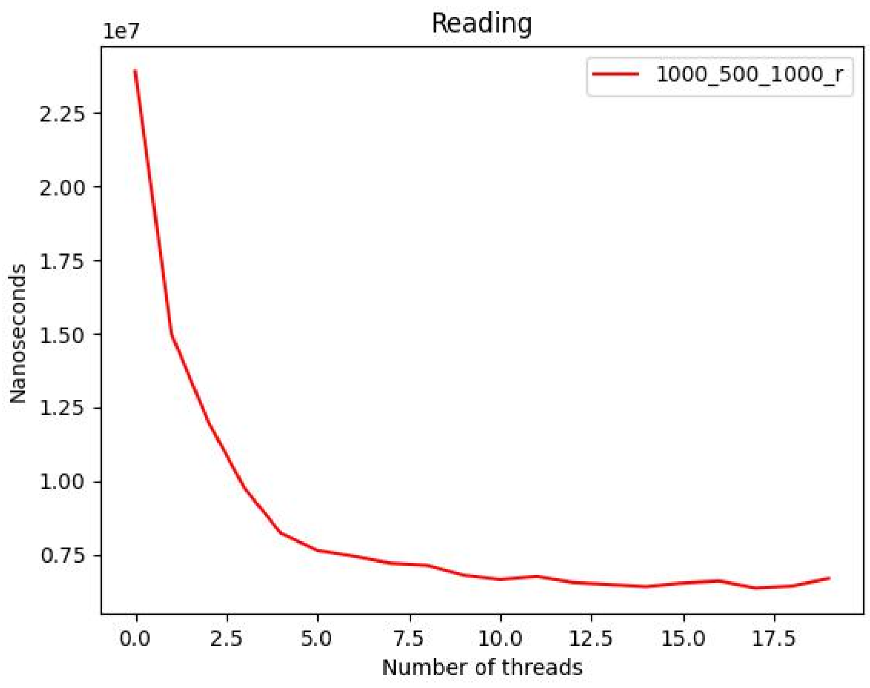

Figure 3: For Matrix Size: 1000x500 and 500x1000

The above graph tells us that the combined read time for the both of the matrixes was 23911629.23 while single threaded and 6798914.13 with 10 threads.

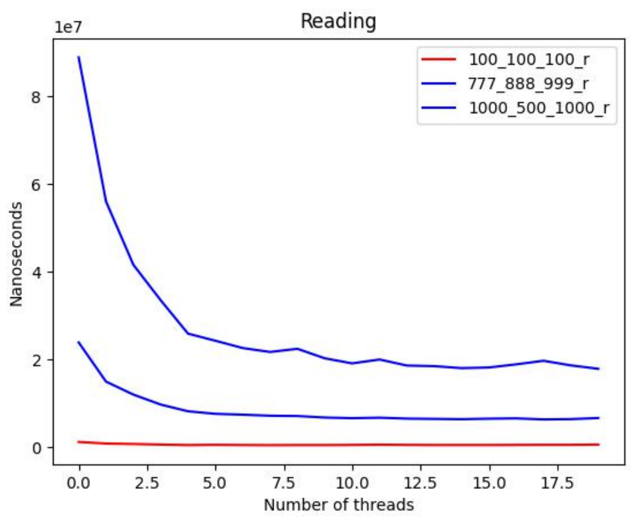

Figure 4: Combined Graph for Comparision

From the combined graph we can see that the matrices with a greater number of elements take more time for reading and there is a much greater time improvement for each increment in the number of threads for the larger matrices than the smaller ones.
We can clearly observe that on varying size of matrices we always notice that the reading time is much lesser than the calculation/writing time.

2.2 Time of P2 process vs Number of threads

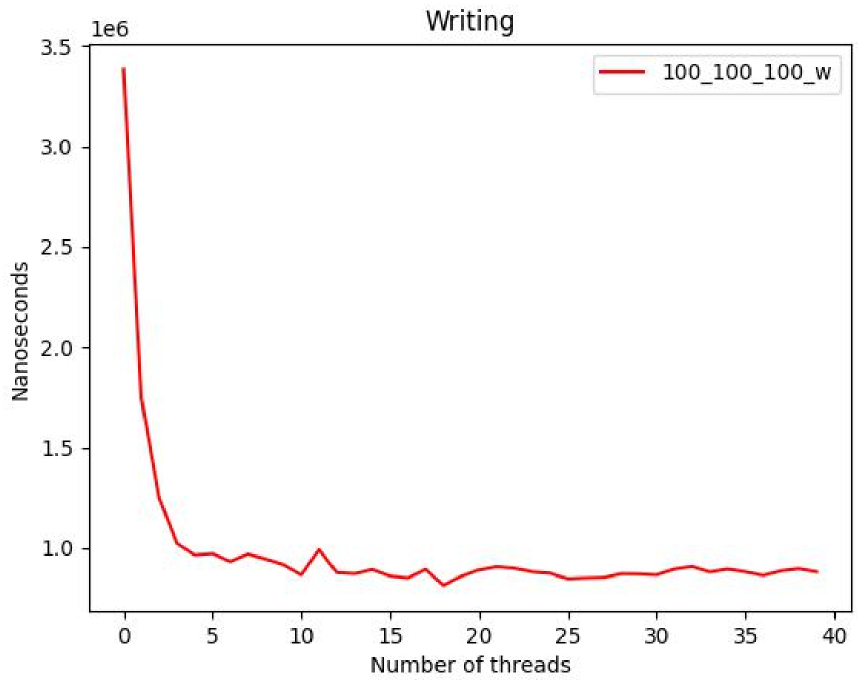

Figure 5: For Matrix Size: 100x100 and 100x100

For writing, we observe 3384848.38 nanosecs on single threaded and 916466.68 nanosecs for at 10 threads after which the graph stabilizes.

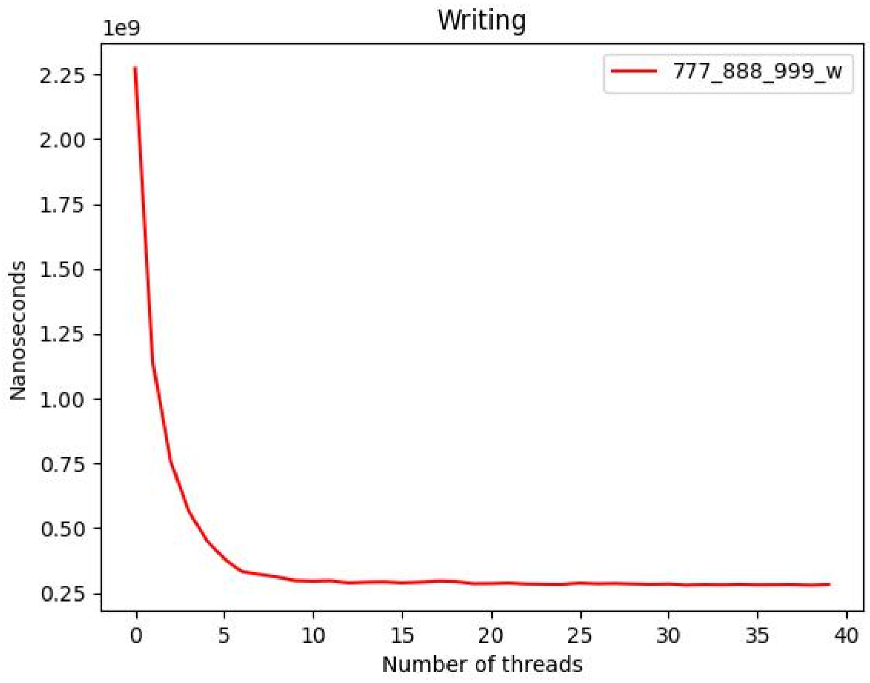

Figure 6: For Matrix Size: 777x888 and 888x999

For writing, we observe 2272799038 nanosecs on single threaded and 298192762.4 nanosecs for at 10 threads after which the graph stabilizes.

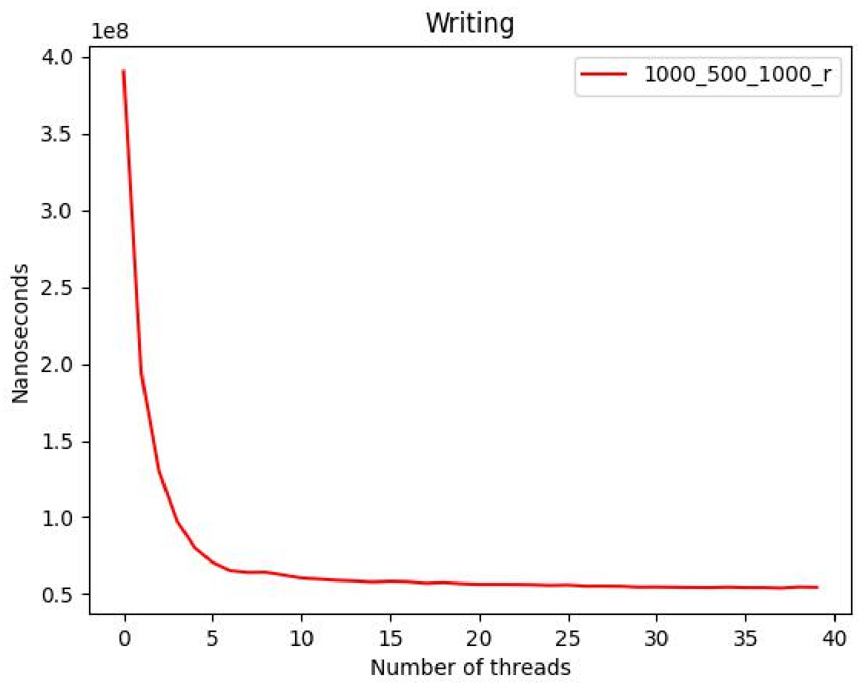

Figure 7: For Matrix Size: 1000x500 and 500x1000

For writing, we observe 390486898.3 nanosecs on single threaded and 62380399.54 nanosecs for at 10 threads after which the graph stabilizes.

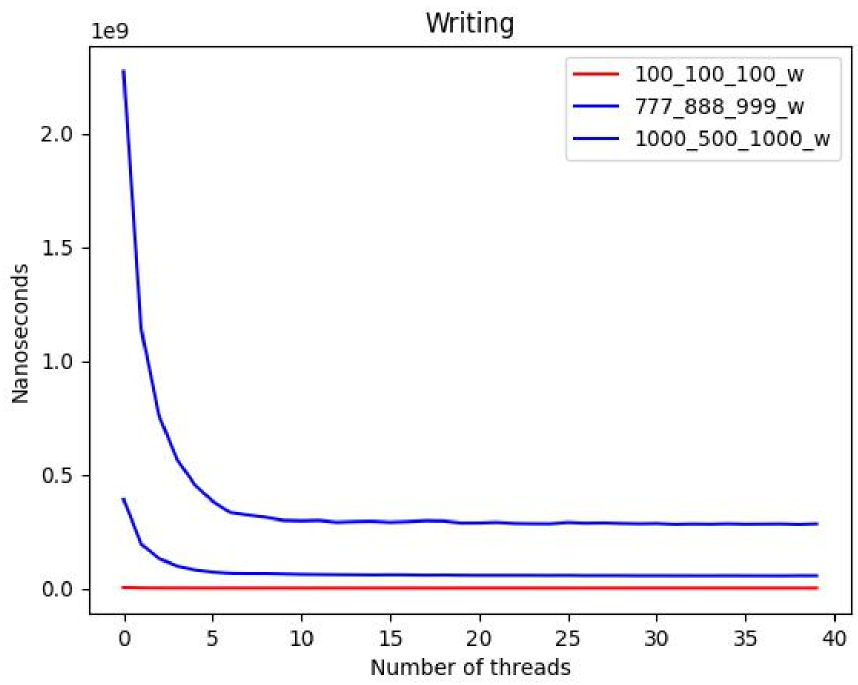

Figure 8: Combined Graph for Comparision

From the combined graph we can see that the matrices with a greater number of elements take more time for writing and there is a much greater time improvement for each increment in the number of threads for the larger matrices than the smaller ones. We see this happen with for the reading process also.

Chapter 3: Round-Robin Plots

3.1 With Time Quantum 2s

3.1.1 Total Turnaround Time vs Workload Size

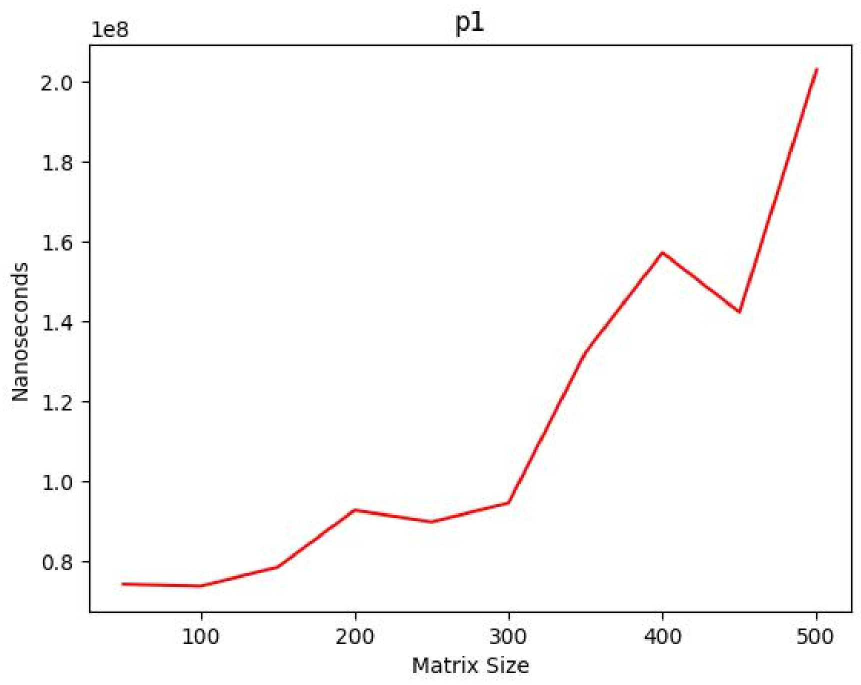

Figure 9: P1 for Time Quantum 2sec

Turnaround time is the amount of time to execute a particular process, interval from submission time to completion time, sum of durations spent waiting to get into memory, waiting in ready queue, executing on CPU, doing input and output

In the above graph, a number n on the X axis refers to both the matrixes having a size of nxn. The general trend is that the turnaround time increases with the increase with the size of the matrix. It should also be noted that for getting more realistic and consistent values we have taken the number of

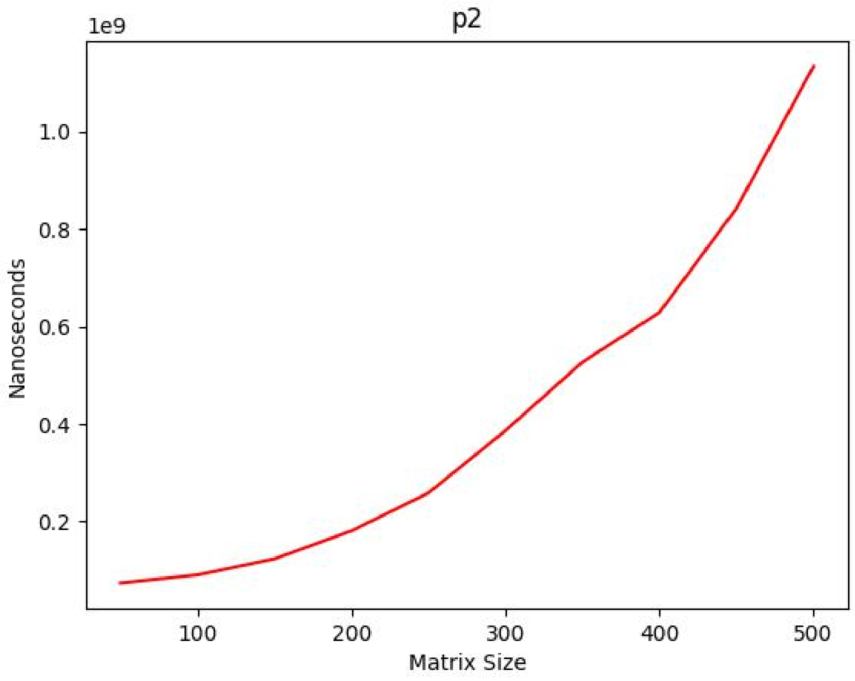

Figure 10: P2 for Time Quantum 2sec

threads as a value greater than the number of logical processors present on the system(4 for this graph)

3.1.2 Waiting Time vs Workload Size

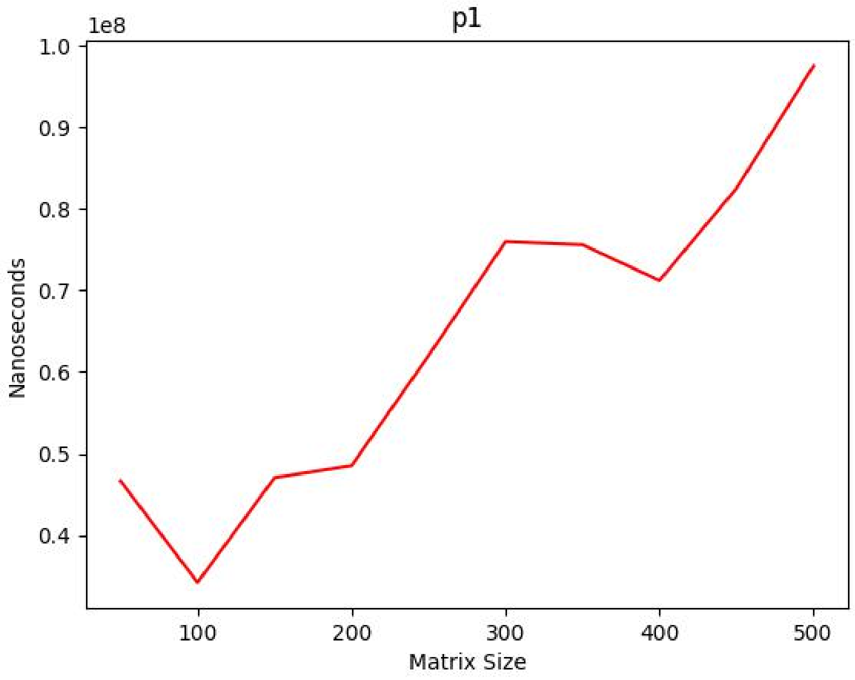

Figure 11: P1 for Time Quantum 2sec

Waiting time is the amount of time a process has been waiting in the ready queue

As the size increases, it stands to reason that the number of time quantums that happen increase, thus the total waiting time on both the processes generally increase when the time quantum is 2ms. This is the same pattern observed with time quantum as 2ms also.

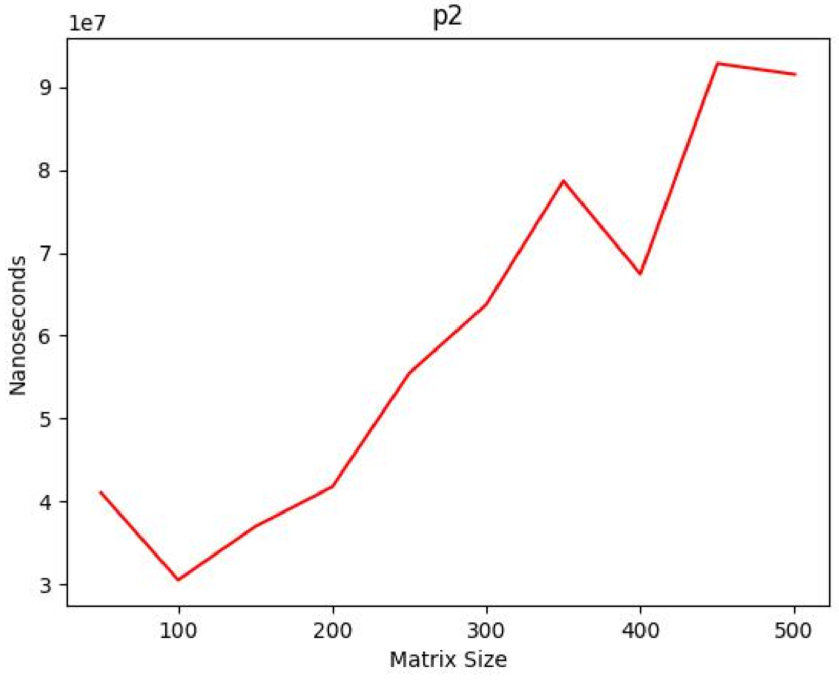

Figure 12: P2 for Time Quantum 2sec

3.1.3 Switching Overhead vs Workload Size

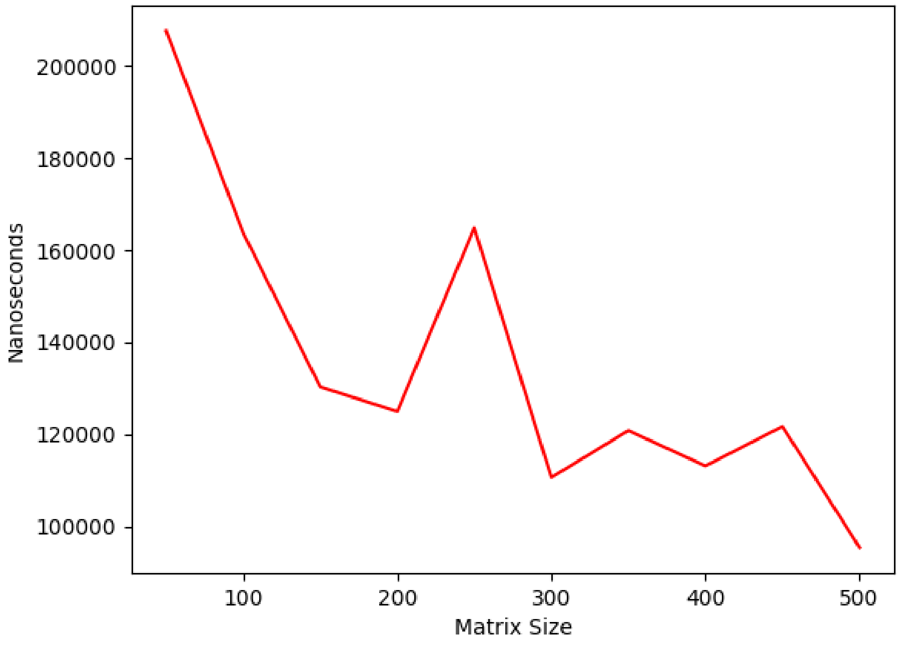

Figure 13: P1 for Time Quantum 2sec

A context switching is a process that involves switching of the CPU from one process or task to another.

As the size of the matrices increases, the calculation time is significantly higher than the reading time, therefore the reading process exits first and there is almost no overhead in context switches after that as it is just the calculation/writing process executing. Thus the average context switching time drops as the size of the matrix decreases.

3.2 With Time Quantum 1s

3.2.1 Total Turnaround Time vs Workload Size

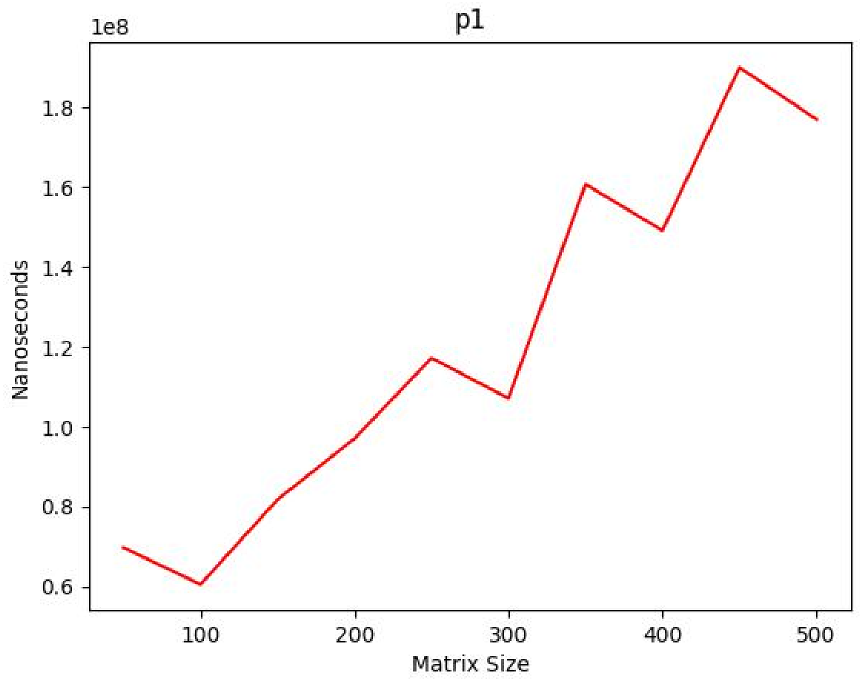

Figure 14: P1 for Time Quantum 1sec

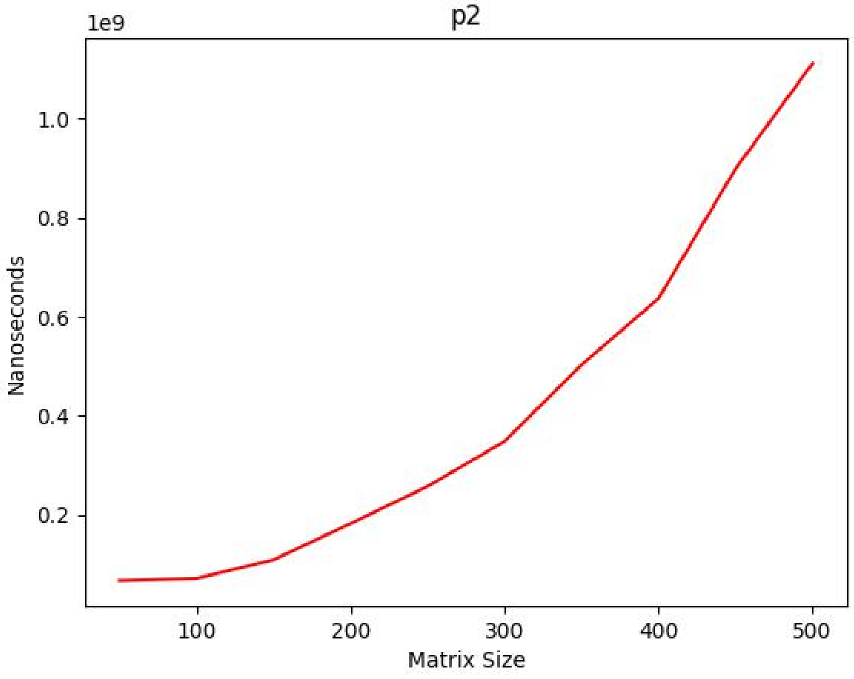

Figure 15: P2 for Time Quantum 1sec

3.2.2 Waiting Time vs Workload Size

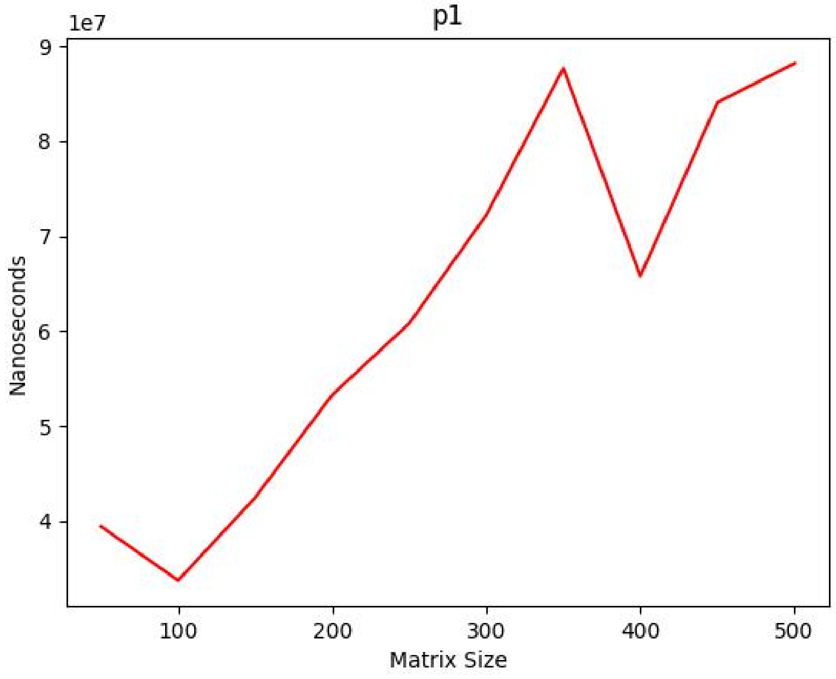

Figure 16: P1 for Time Quantum 1sec

As the size increases, it stands to reason that the number of time quantums that happen increase, thus the total waiting time on both the processes generally increase when the time quantum is 1ms.

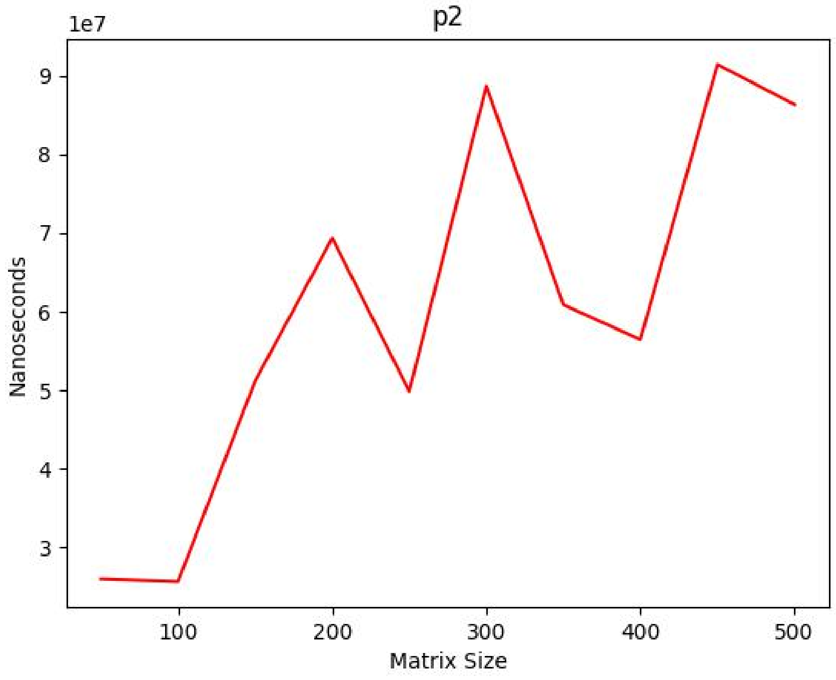

Figure 17: P2 for Time Quantum 1sec

3.2.3 Switching Overhead vs Workload Size

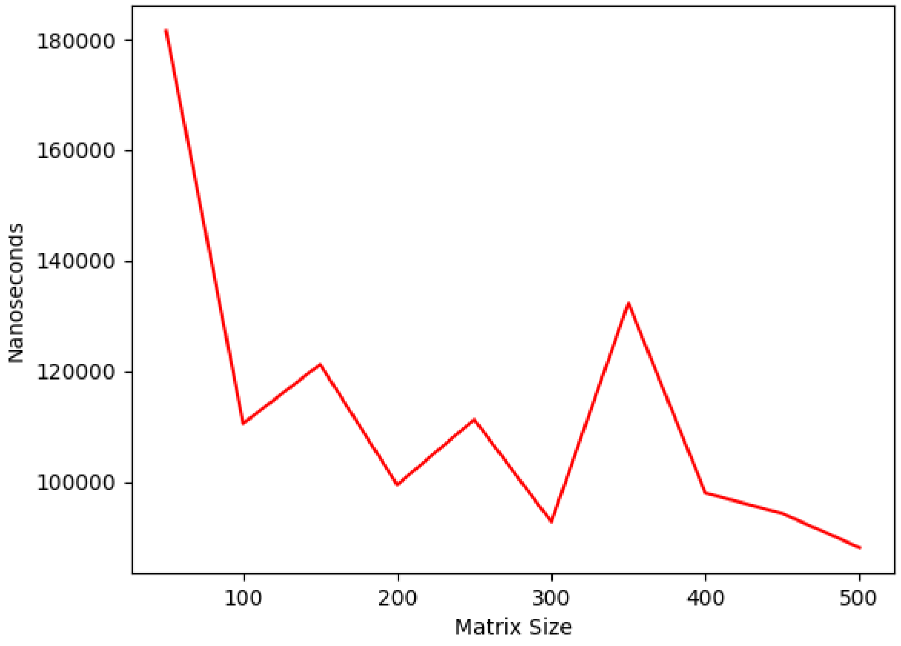

Figure 18: P1 for Time Quantum 1sec

Chapter 4: Optimizations

To check which all rows P1 has read, so that P2 can calculate accordingly, we are not maintaining any extra array variable in the shared memory. We are reusing the file-position array for that purpose.
We optimized the normal turnstile-mechanism of semaphores, that was being used to create a barrier, so that the design pattern could be reused without creating any extra for-loops to reset the semaphore value.
In signalling the context switch time is greater than using semaphores because the signal requires the entire state of the thread to be stored or loaded. In semaphores, all the details are still saved on the stack thus there is lesser time wasted in copying to and from the memory.