**Final Year B. Tech, Sem VII 2022-23**

**PRN – 2020BTECS00211**

**Name – Aashita Narendra Gupta**

**High Performance Computing**

**Lab Batch: B4**

**SLIP No - 42**

**HPC LAB ESE**

**Github Link for Code -** [**https://github.com/Aashita06/HPC\_Practicals**](https://github.com/Aashita06/HPC_Practicals)

**Q.1) Write a program to demonstrate distributed sum of an array using MPI.**

**🡪**

**Code:**

#include "mpi.h"

#include <stdio.h>

#define localSize 1000

// store the subarray data comming from process 0;

int local[1000];

int main(int argc, char \*\*argv)

{

    int rank;

    int num;

    int n = 20;

    int arr[20] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10,11,12,13,14,15,16,17,18,19,20};

    int per\_process, elements\_received;

    MPI\_Init(&argc, &argv);

    MPI\_Comm\_size(MPI\_COMM\_WORLD, &num);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

    MPI\_Status status;

    // process with rank 0 will divide data among all processes and add partial sums to get final sum

    if (rank == 0)

    {

        int index, i;

        per\_process = n / num;

        double start = MPI\_Wtime();

        // if more than 1 processes available

        if (num > 1)

        {

            //divide array data among processes

            for (i = 1; i < num - 1; i++)

            {

                //calculating first index of subarray that need to be send to ith process

                index = i \* per\_process;

                //send no of elements and subarray of that lenght to each process

                MPI\_Send(&per\_process, 1, MPI\_INT, i, 0, MPI\_COMM\_WORLD);

                MPI\_Send(&arr[index], per\_process, MPI\_INT, i, 0, MPI\_COMM\_WORLD);

            }

            // for last process send all remaining elements

            index = i \* per\_process;

            int ele\_left = n - index;

            MPI\_Send(&ele\_left, 1, MPI\_INT, i, 0, MPI\_COMM\_WORLD);

            MPI\_Send(&arr[index], ele\_left, MPI\_INT, i, 0, MPI\_COMM\_WORLD);

        }

        // add numbers on process with rank 0

        int sum = 0;

        for (int i = 0; i < per\_process; i++)

        {

            sum += arr[i];

        }

        // add all partial sums from all processes

        int tmp;

        for (int i = 1; i < num; i++)

        {

            MPI\_Recv(&tmp, 1, MPI\_INT, MPI\_ANY\_SOURCE, 0, MPI\_COMM\_WORLD, &status);

            int sender = status.MPI\_SOURCE;

            sum += tmp;

        }

        printf("\nSum of array = %d\n", sum);

        double end = MPI\_Wtime();

        printf("Time required by %d processors : %f",num,end-start);

    }

    else // if rank of process is not 0, then receive elements and calculate partial sums

    {

        // receive no of elements and elements form process 0 and store them on local array

        MPI\_Recv(&elements\_received, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, &status);

        MPI\_Recv(&local, elements\_received, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, &status);

        // calculate partial local sum

        int partial\_sum = 0;

        for (int i = 0; i < elements\_received; i++)

        {

            partial\_sum += local[i];

           // printf("\nPartial Sum of rank %d is %d ",rank,partial\_sum);

        }

        //send calculated partial sum to process with rank 0

        MPI\_Send(&partial\_sum, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);

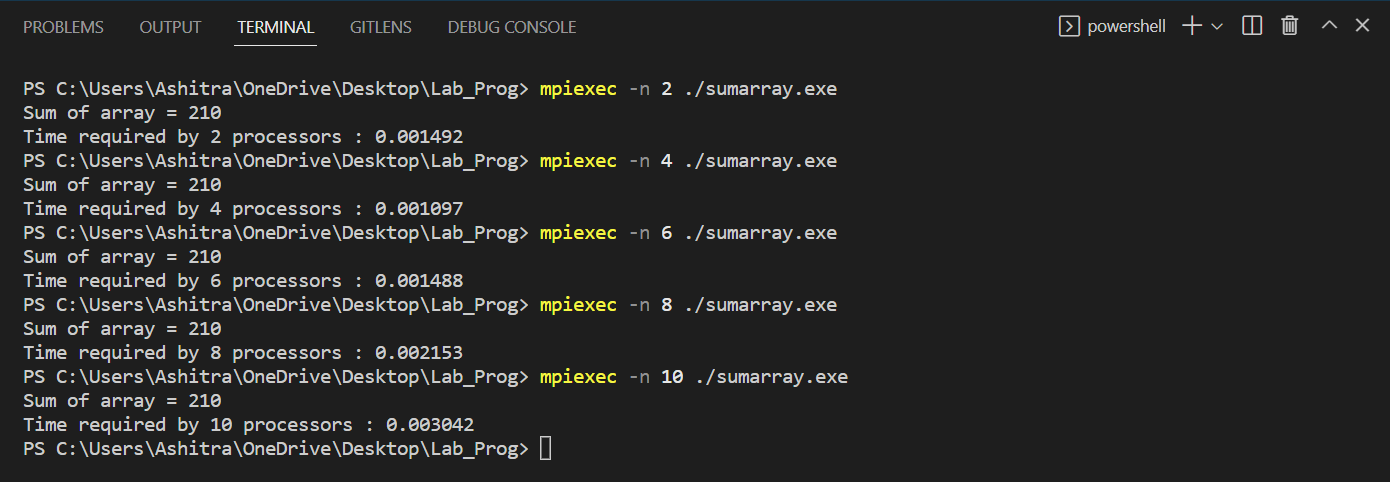
    }

    MPI\_Finalize();

    return 0;

}

**Output:**

****

**We are analysis the execution time taken by different number of processors to calculate the distributed sum of array. Here, due to communication overhead, time is increasing with increasing processors.**

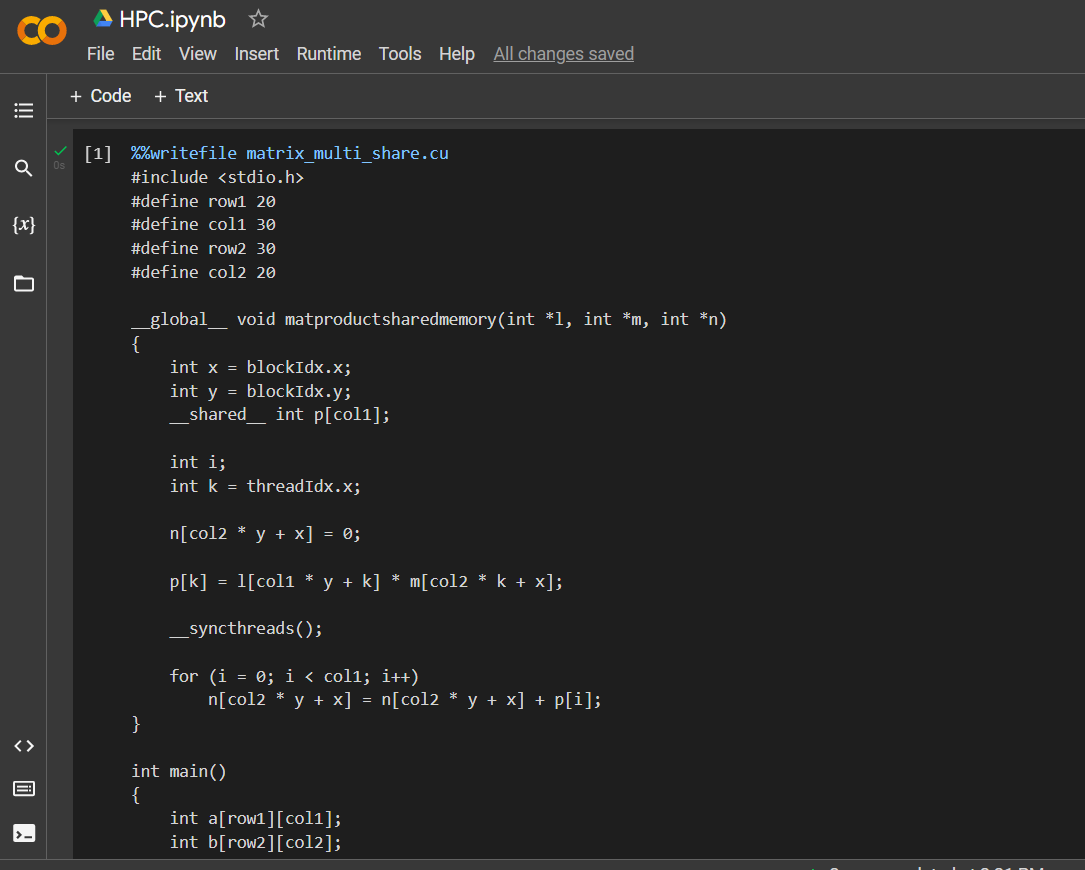
|  |  |
| --- | --- |
| Processors | Time |
| 2 | 0.001492 |
| 4 | 0.001097 |
| 6 | 0.001488 |
| 8 | 0.002153 |
| 10 | 0.003042 |

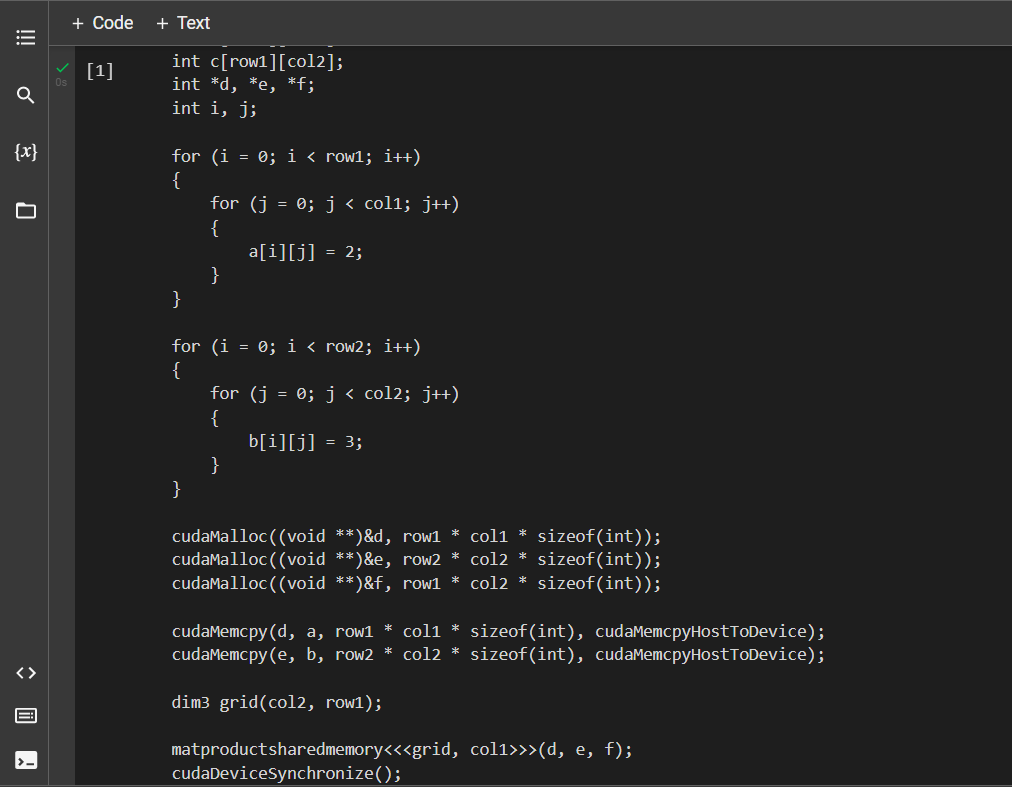
**Q.2) Implement matrix matrix multiplication using CUDA (shared memory).**

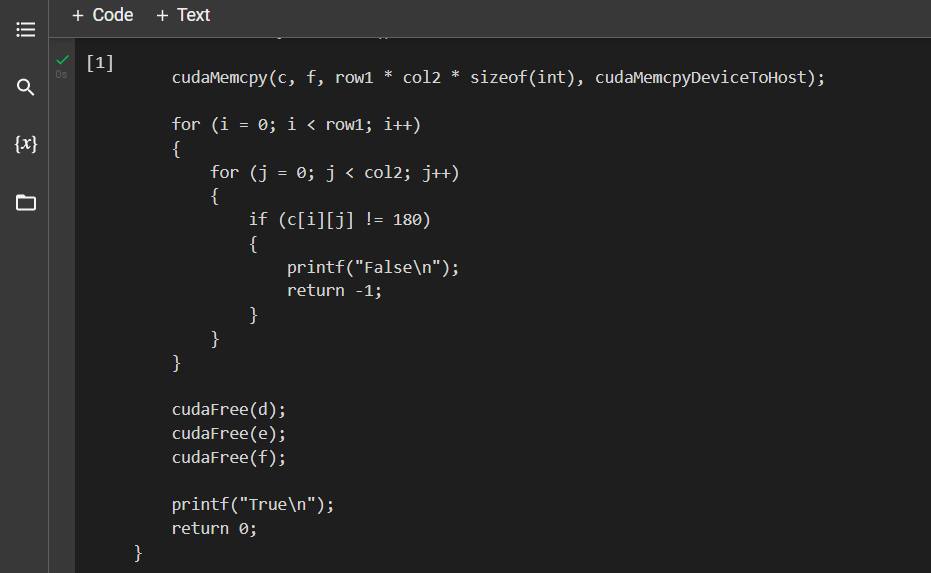
**(Conduct performance and speedup analysis of all programs)**

**🡪**

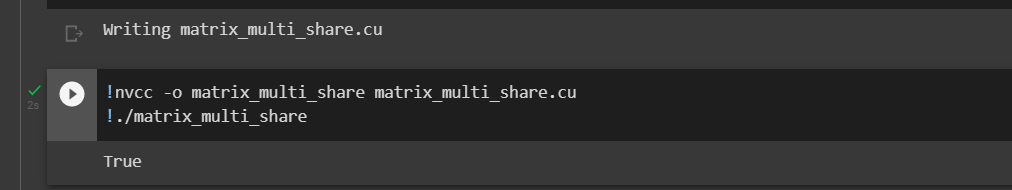
**Code:**

****

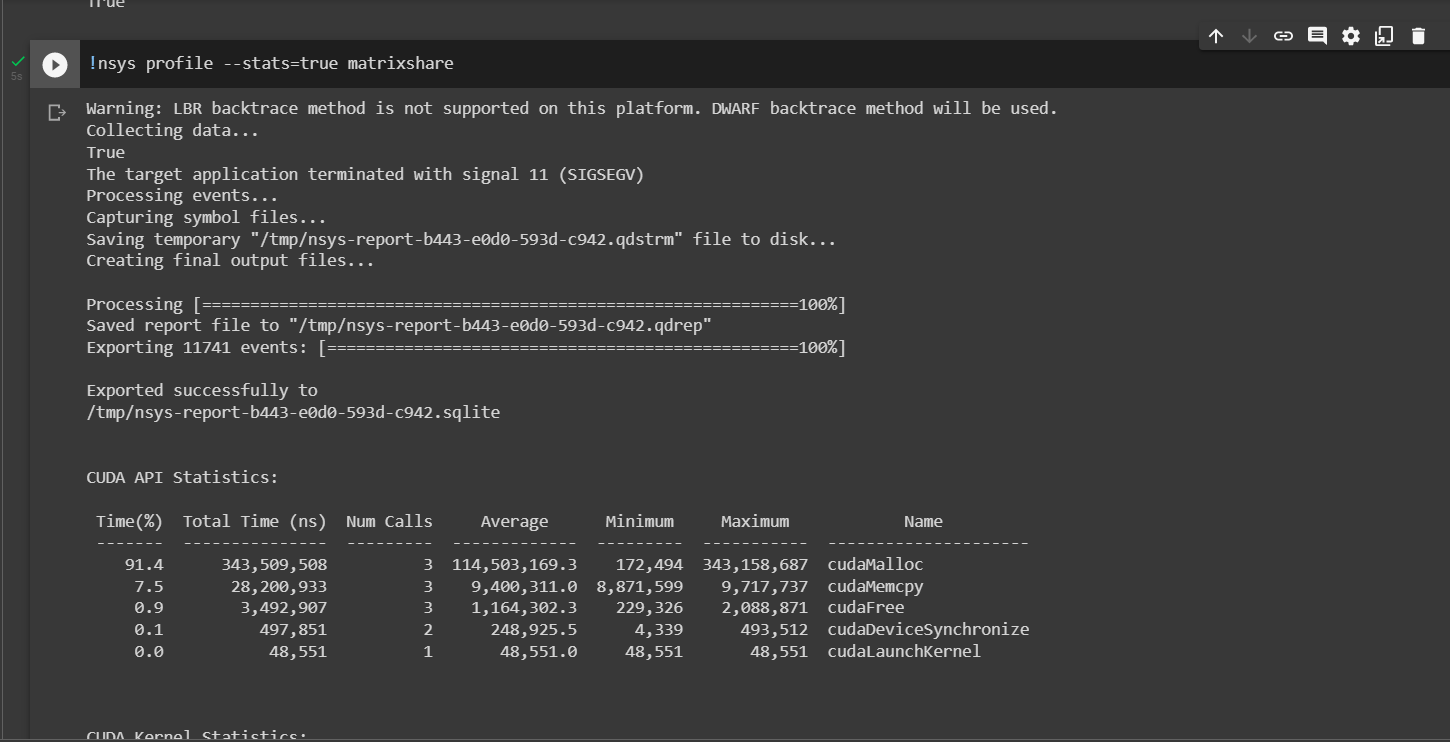
****

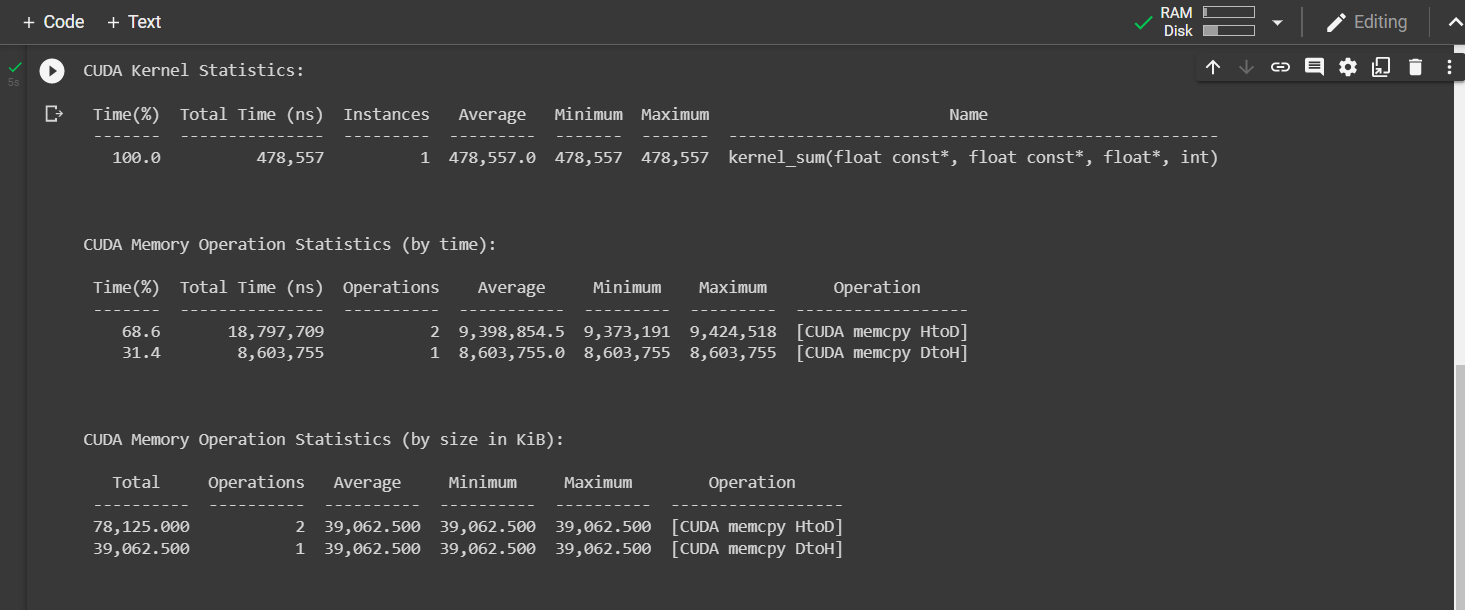
****

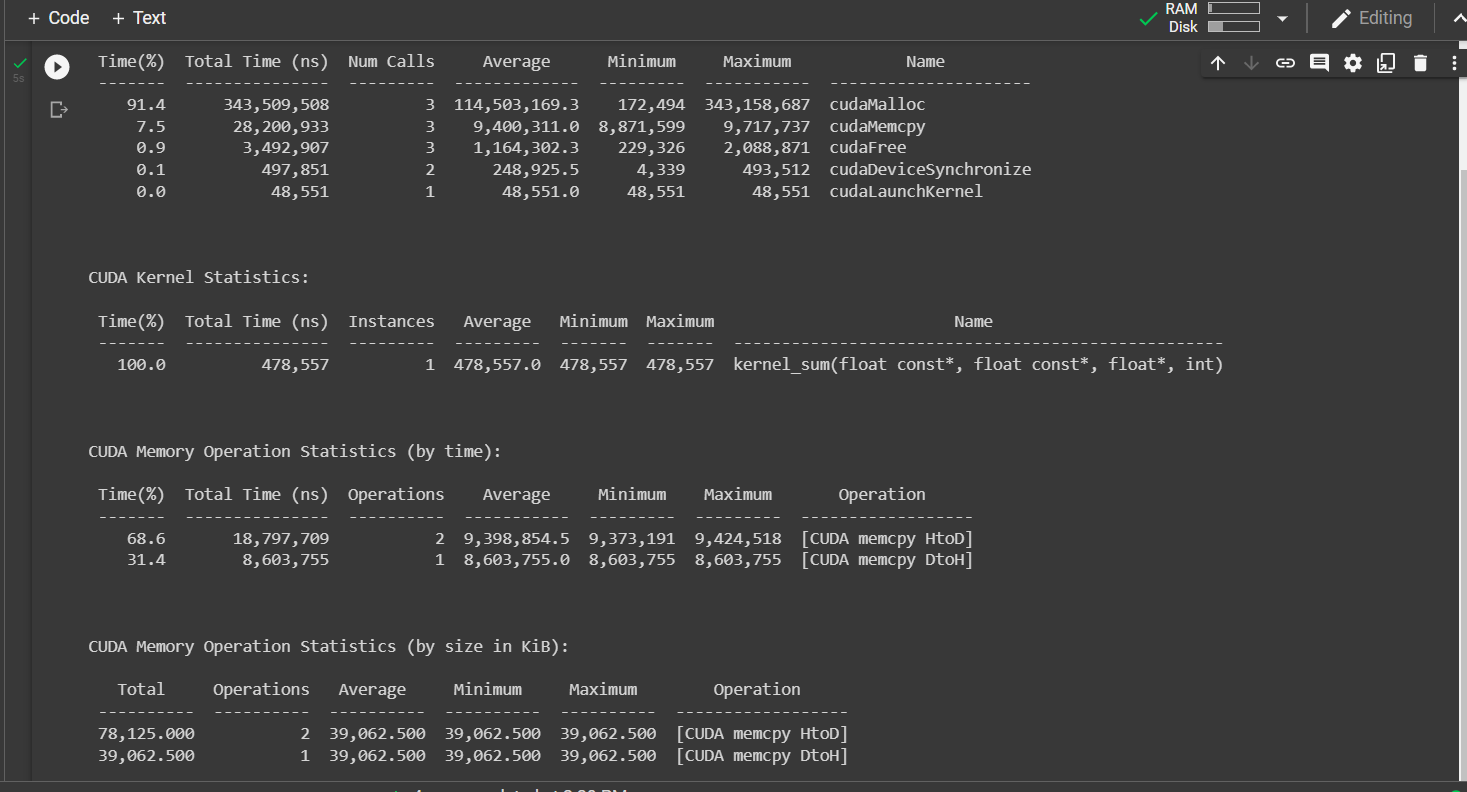
**Output:**

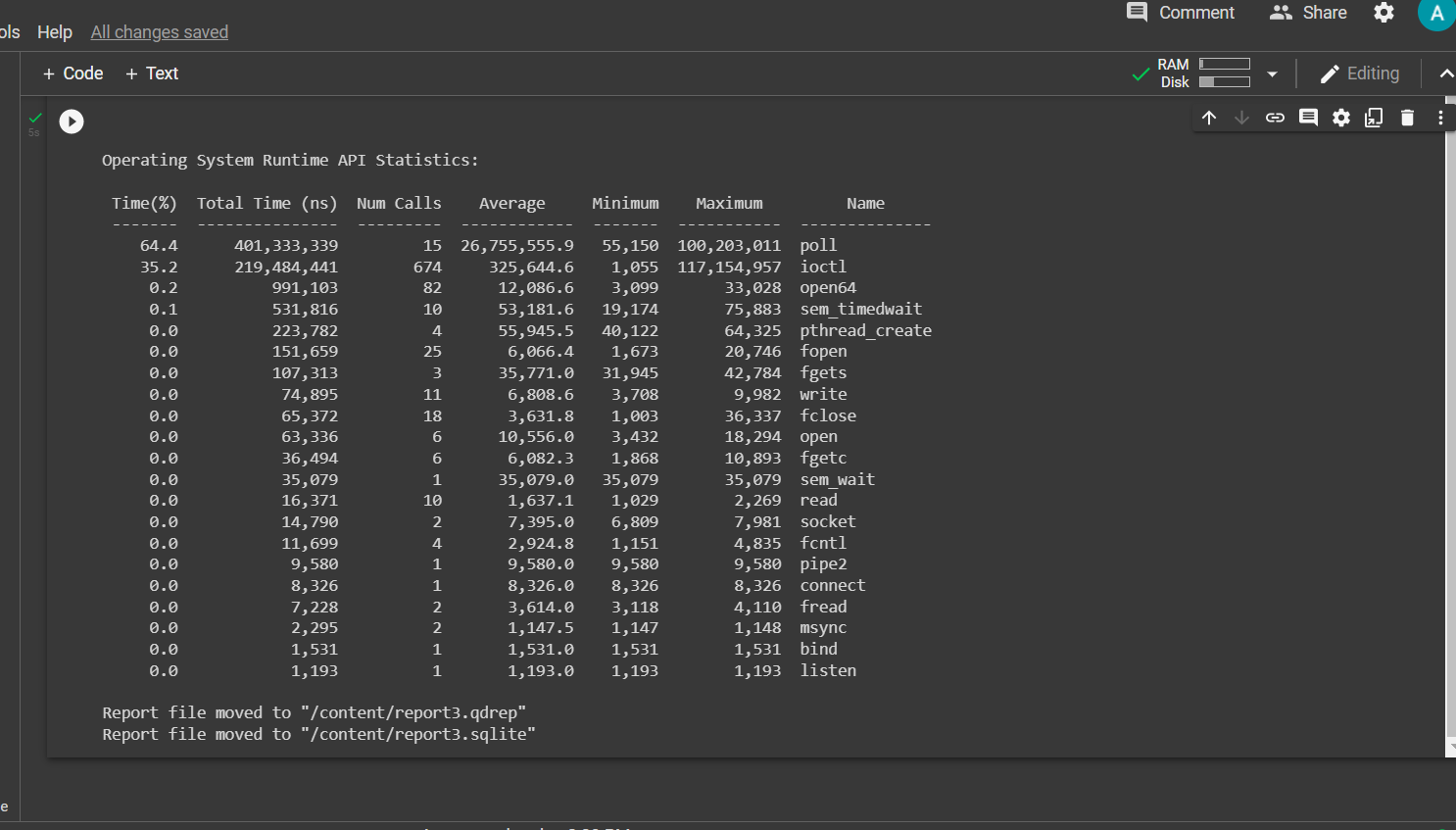
****

**Profiling of matrix matrix multiplication program in CUDA (Shared Memory).**

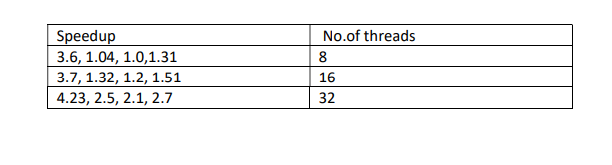
****

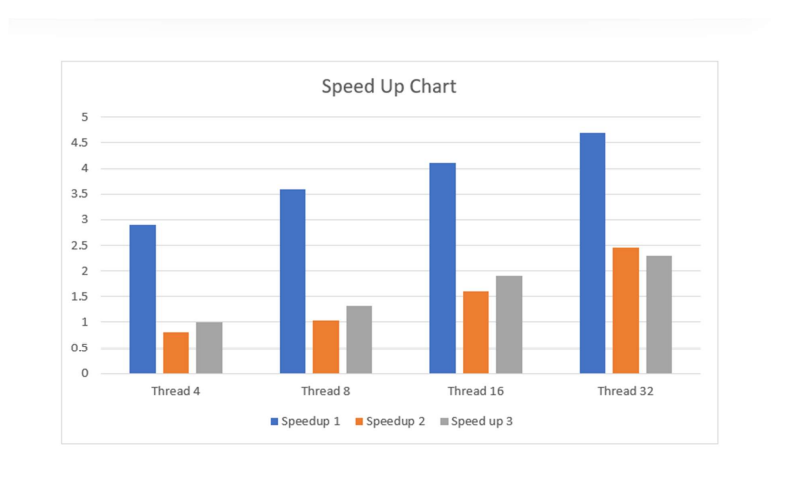
****

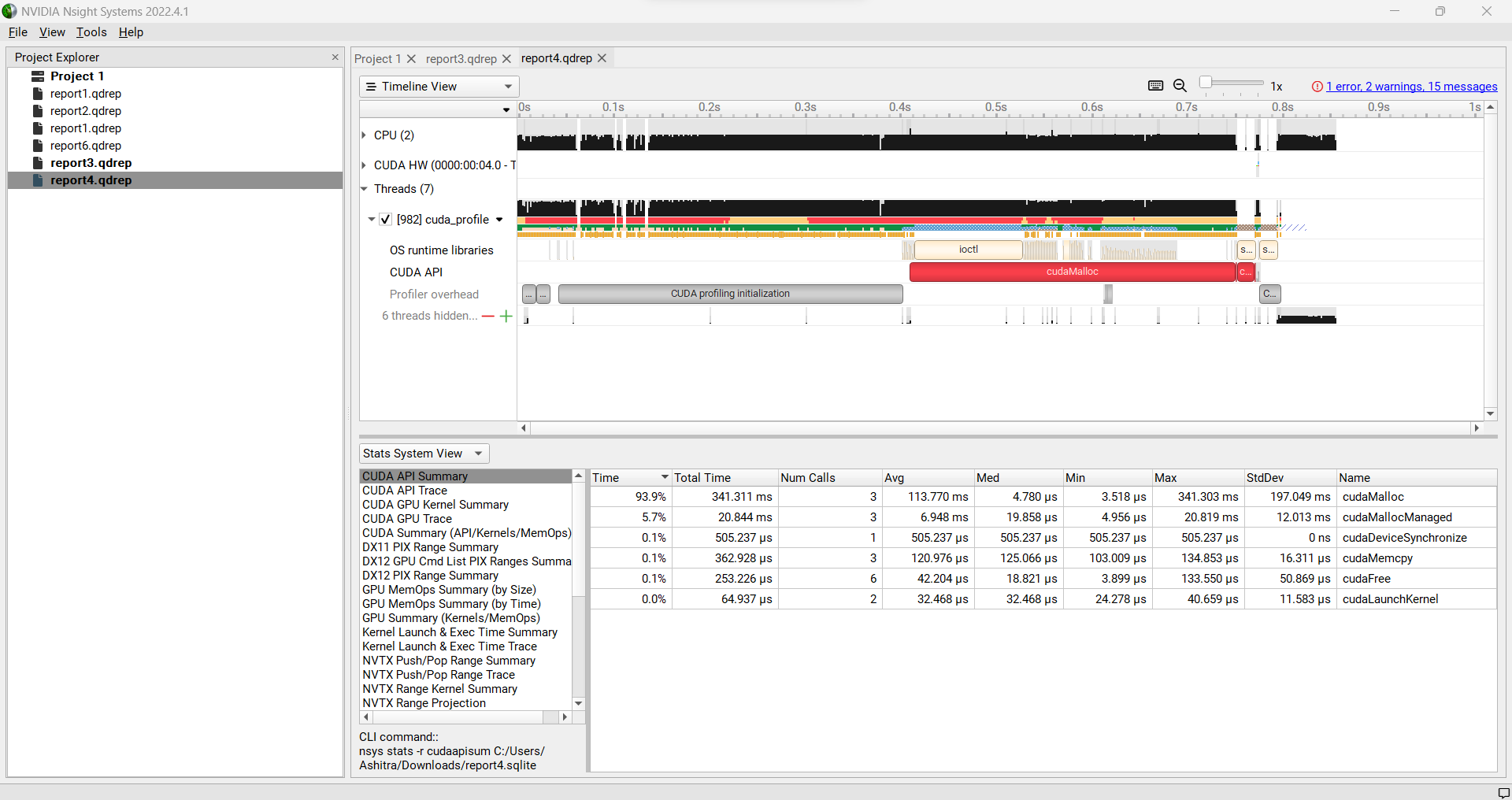
****

****

**Keeping No. of blocks constants and increasing no. of threads, we are calculating speedup of the following pmatrix multiplication program.**

****

****

****