Name: Parshwa Herwade

PRN No: 22510064

**High Performance Computing Lab**

**Practical No. 10**

**Title of practical: Understanding concepts of CUDA Programming**

**Problem Statement 1:**

**Execute the following program and check the properties of your GPGPU.**

**#include <stdio.h>**

**#include <stdlib.h>**

**int main()**

**{**

**int deviceCount;**

**cudaGetDeviceCount(&deviceCount);**

**if (deviceCount == 0)**

**{**

**printf("There is no device supporting CUDA\n");**

**}**

**int dev;**

**for (dev = 0; dev < deviceCount; ++dev)**

**{**

**cudaDeviceProp deviceProp;**

**cudaGetDeviceProperties(&deviceProp, dev);**

**if (dev == 0)**

**{**

**if (deviceProp.major < 1)**

**{**

**printf("There is no device supporting CUDA.\n");**

**}**

**else if (deviceCount == 1)**

**{**

**printf("There is 1 device supporting CUDA\n");**

**}**

**else**

**{**

**printf("There are %d devices supporting CUDA\n", deviceCount);**

**}**

**}**

**printf("\nDevice %d: \"%s\"\n", dev, deviceProp.name);**

**printf(" Major revision number: %d\n", deviceProp.major);**

**printf(" Minor revision number: %d\n", deviceProp.minor);**

**printf(" Total amount of global memory: %d bytes\n", deviceProp.totalGlobalMem);**

**printf(" Total amount of constant memory: %d bytes\n", deviceProp.totalConstMem);**

**printf(" Total amount of shared memory per block: %d bytes\n", deviceProp.sharedMemPerBlock);**

**printf(" Total number of registers available per block: %d\n", deviceProp.regsPerBlock);**

**printf(" Warp size: %d\n", deviceProp.warpSize);**

**printf(" Multiprocessor count: %d\n",deviceProp.multiProcessorCount );**

**printf(" Maximum number of threads per block: %d\n", deviceProp.maxThreadsPerBlock);**

**printf(" Maximum sizes of each dimension of a block: %d x %d x %d\n", deviceProp.maxThreadsDim[0],deviceProp.maxThreadsDim[1], deviceProp.maxThreadsDim[2]);**

**printf(" Maximum sizes of each dimension of a grid: %d x %d x %d\n", deviceProp.maxGridSize[0], deviceProp.maxGridSize[1], deviceProp.maxGridSize[2]);**

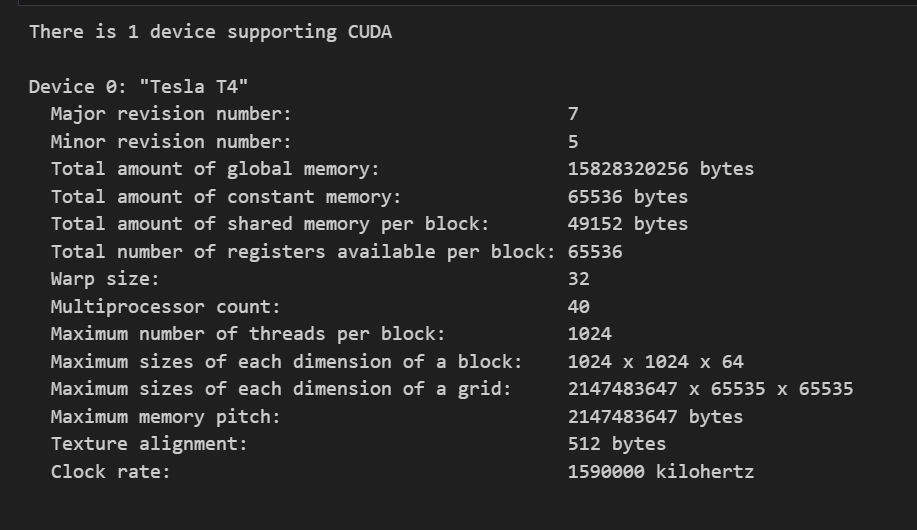
**printf(" Maximum memory pitch: %d bytes\n", deviceProp.memPitch);**

**printf(" Texture alignment: %d bytes\n", deviceProp.textureAlignment);**

**printf(" Clock rate: %d kilohertz\n", deviceProp.clockRate);**

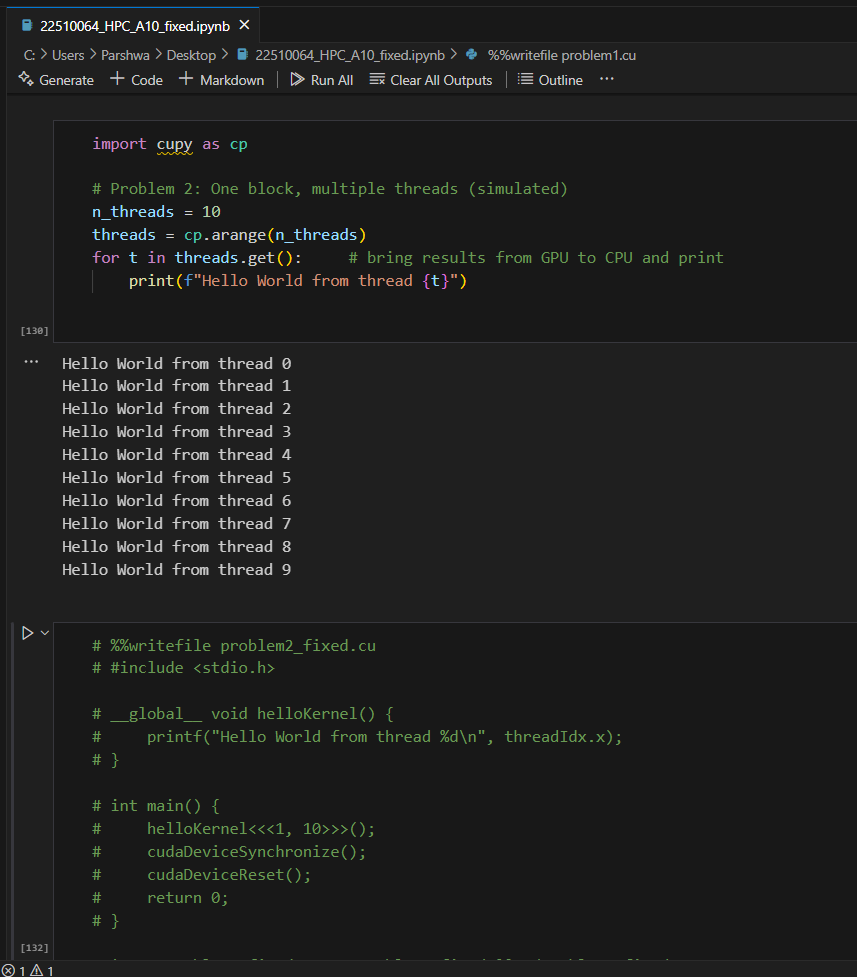
**}**

**}**

****

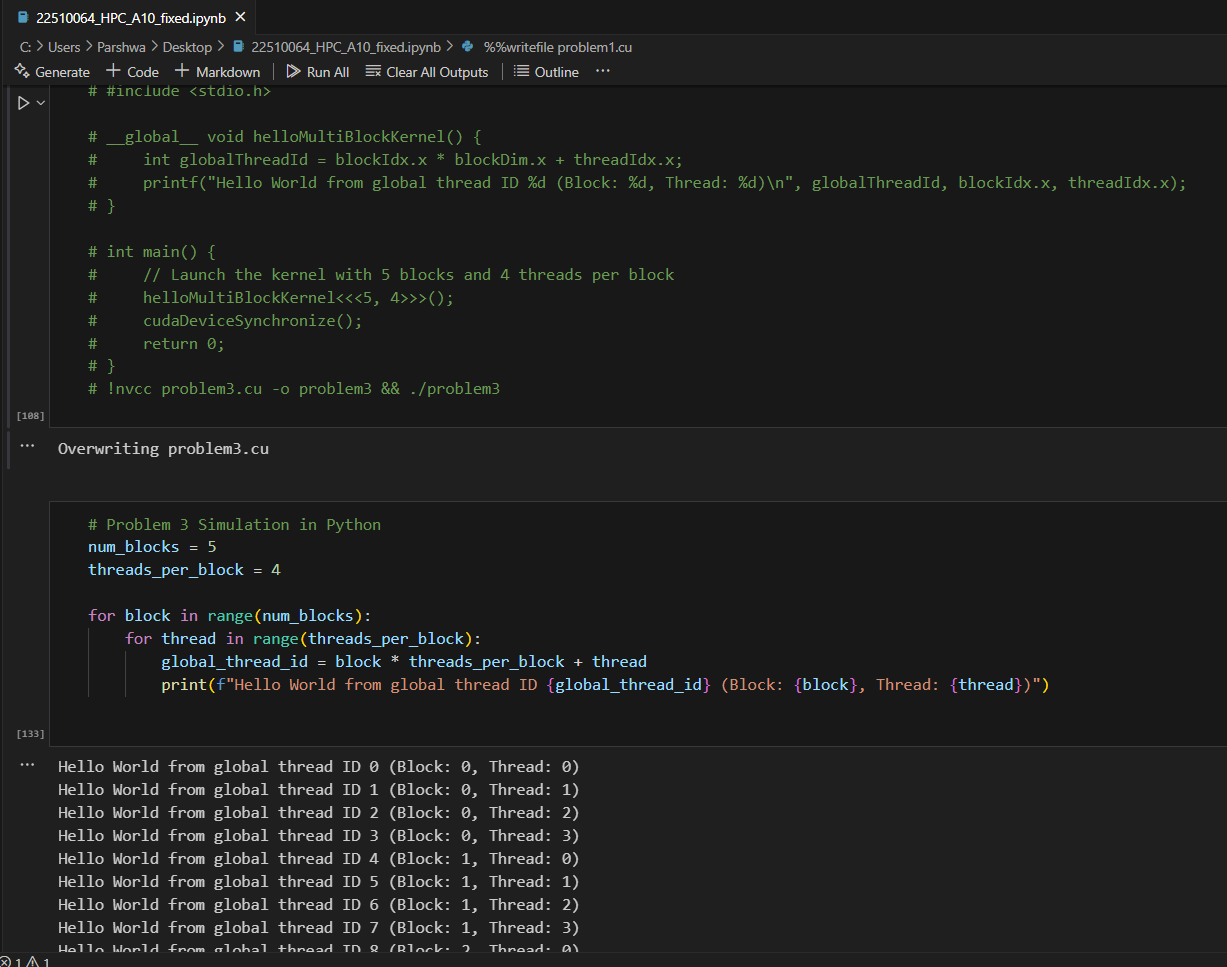
**Problem Statement 2:**

**Write a program to where each thread prints its thread ID along with hello world. Lauch the kernel with one block and multiple threads.**

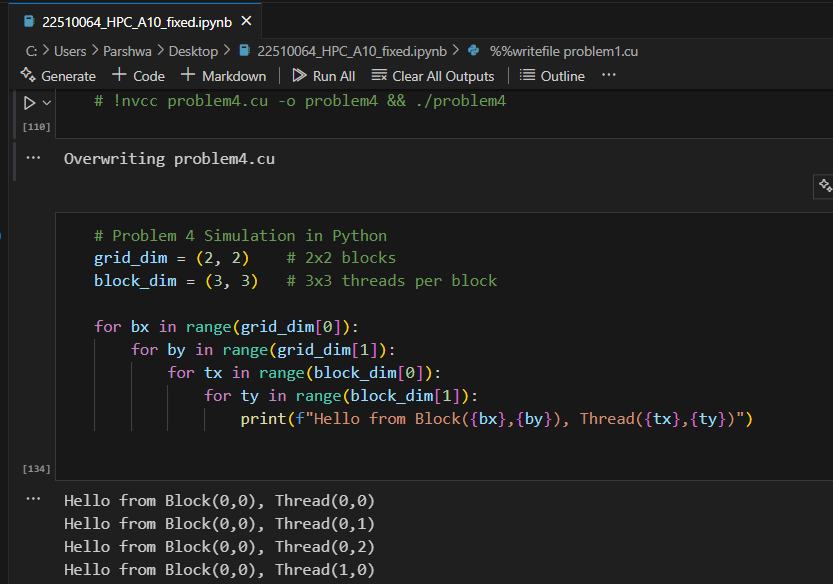
****

**Problem Statement 3:**

**Write a program to where each thread prints its thread ID along with hello world. Lauch the kernel with multiple blocks and multiple threads.**

**Problem Statement 4:**

**Write a program to where each thread prints its thread ID along with hello world. Lauch the kernel with 2D blocks and 2D threads.**

****

**Problem statement 5: Vector Addition using CUDA**

**Problem Statement: Write a CUDA C program that performs element-wise addition of two vectors A and B of size N. The result of the addition should be stored in vector C.**

**Details:**

**• Initialize the vectors A and B with random numbers.**

**• The output vector C[i] = A[i] + B[i], where i ranges from 0 to N-1.**

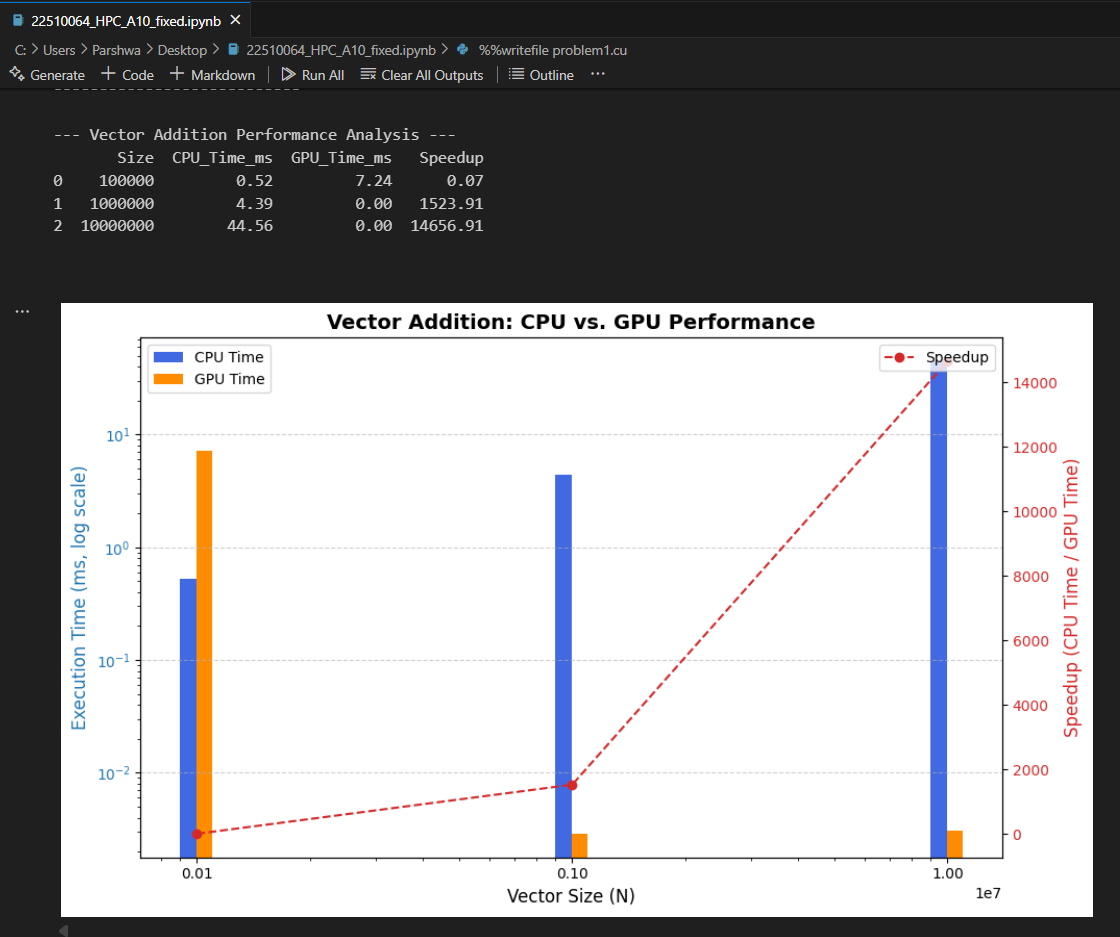
**• Use CUDA kernels to perform the computation in parallel.**

**• Write the code for both serial (CPU-based) and parallel (CUDA-based) implementations.**

**• Measure the execution time of both the serial and CUDA implementations for different values of N (e.g., N = 10^5, 10^6, 10^7).**

**Task:**

**• Calculate and report the speedup (i.e., the ratio of CPU execution time to GPU execution time).**

****

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Problem statement 6: Matrix Addition using CUDA**

**Problem Statement: Write a CUDA C program to perform element-wise addition of two matrices A and B of size M x N. The result of the addition should be stored in matrix C.**

**Details:**

**• Initialize the matrices A and B with random values.**

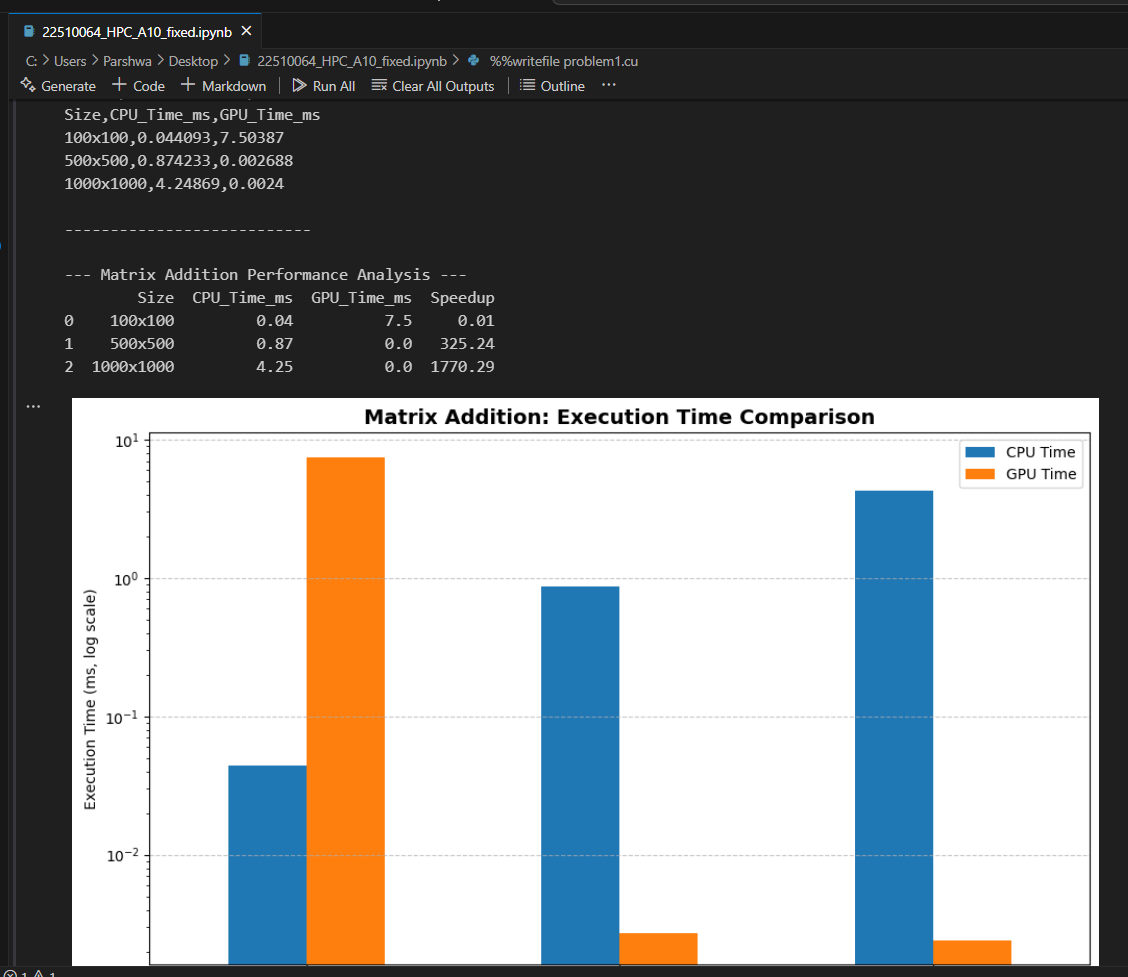
**• The output matrix C[i][j] = A[i][j] + B[i][j] where i ranges from 0 to M-1 and j ranges from 0 to N-1.**

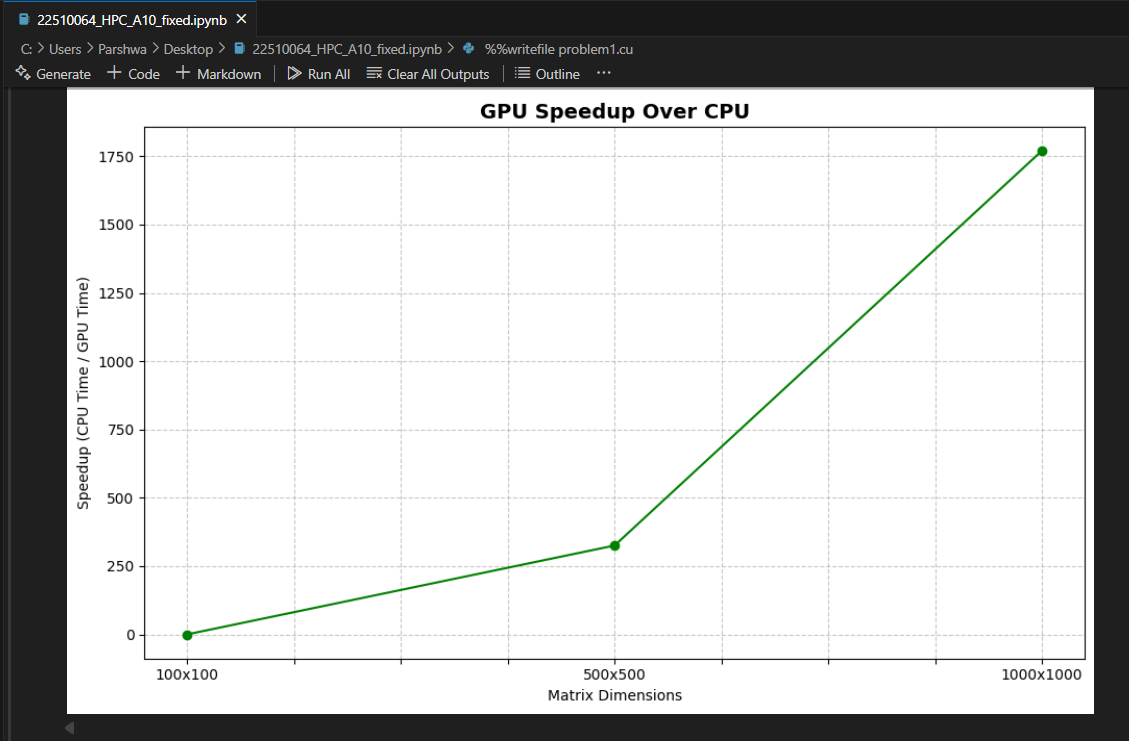
**• Write code for both serial (CPU-based) and parallel (CUDA-based) implementations.**

**• Measure the execution time of both implementations for various matrix sizes (e.g., 100x100, 500x500, 1000x1000).**

**Task:**

**• Calculate the speedup using the execution times of the CPU and GPU implementations.**

****

****

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Problem statement 7: Dot Product of Two Vectors using CUDA**

**Problem Statement: Write a CUDA C program to compute the dot product of two vectors A and B of size N. The dot product is defined as:**

**Details:**

**• Initialize the vectors A and B with random values.**

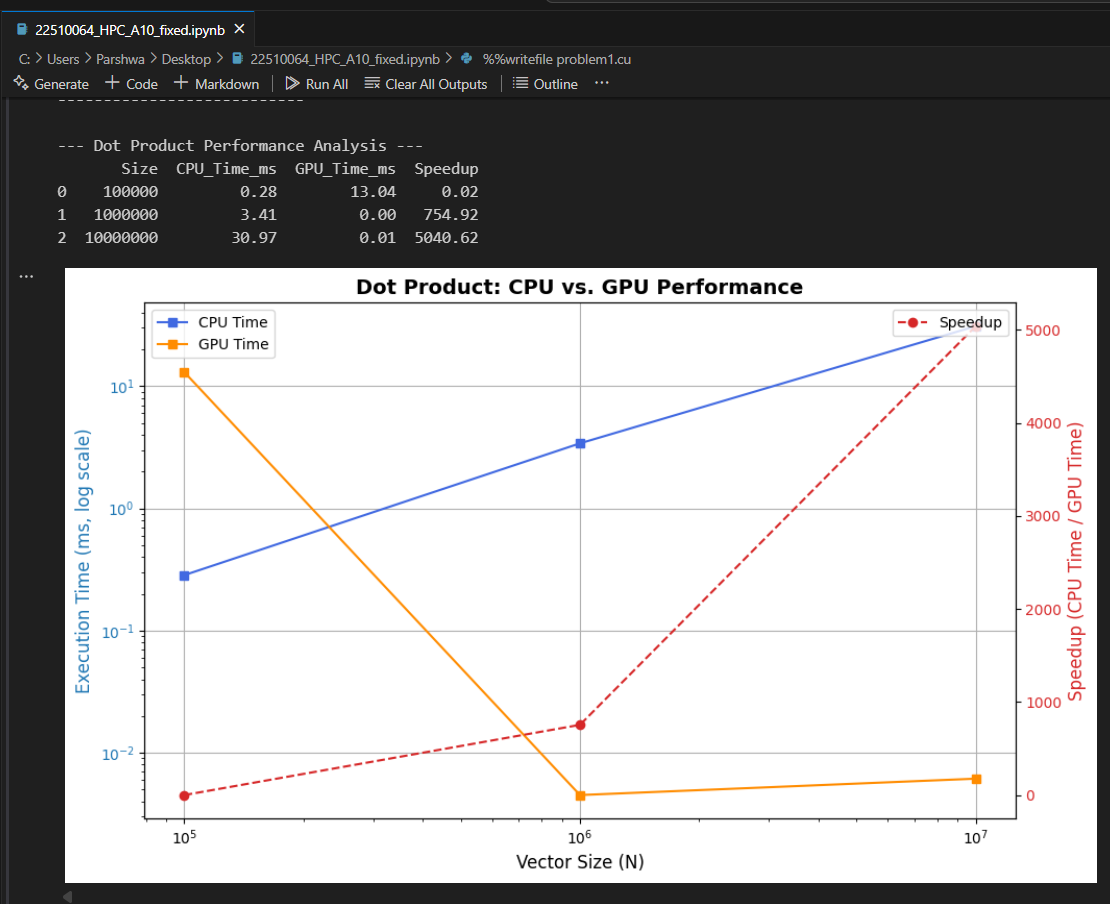
**• Implement the dot product calculation using both serial (CPU) and parallel (CUDA) approaches.**

**• Measure the execution time for both implementations with different vector sizes (e.g., N = 10^5, 10^6, 10^7).**

**• Use atomic operations or shared memory reduction in the CUDA kernel to compute the final sum.**

**Task:**

**• Calculate and report the speedup for different vector sizes.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** **\_\_**

**Problem statement 8: Matrix Multiplication using CUDA**

**Problem Statement: Write a CUDA C program to perform matrix multiplication. Given two matrices A (MxN) and B (NxP), compute the resulting matrix C (MxP) where:**

**Details:**

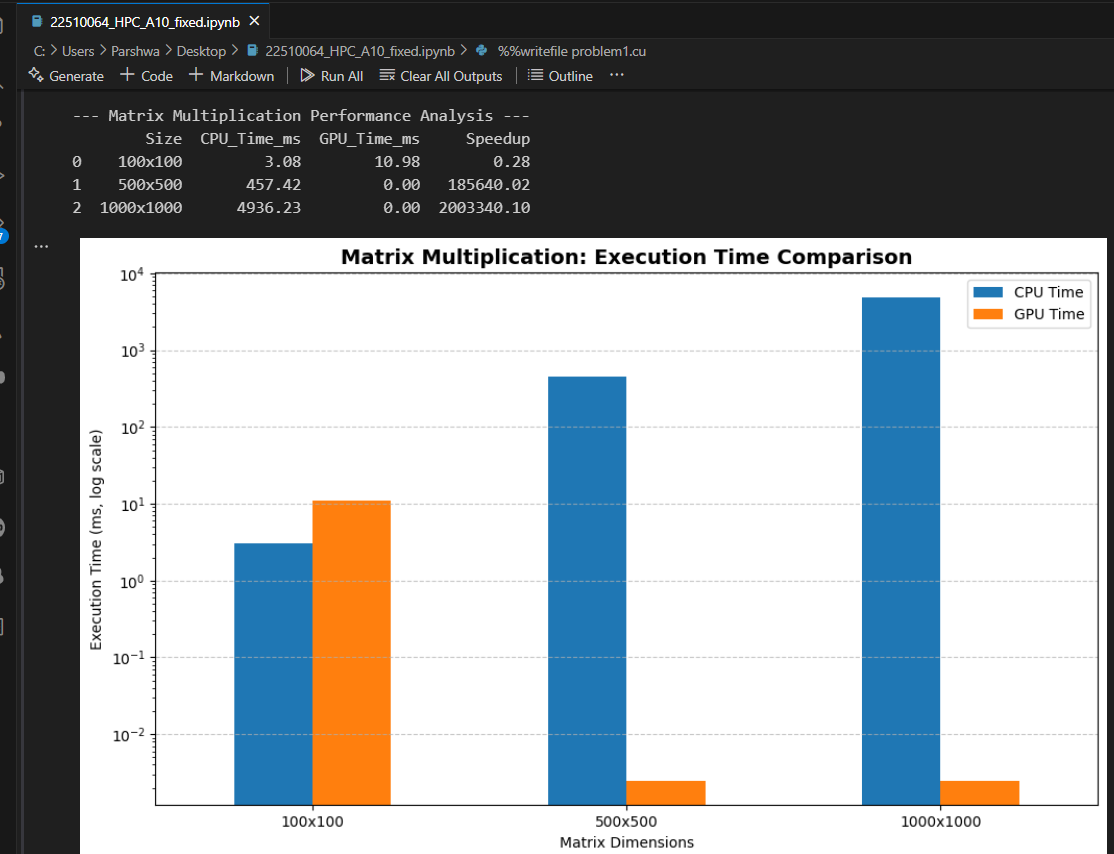
**• Initialize the matrices A and B with random values.**

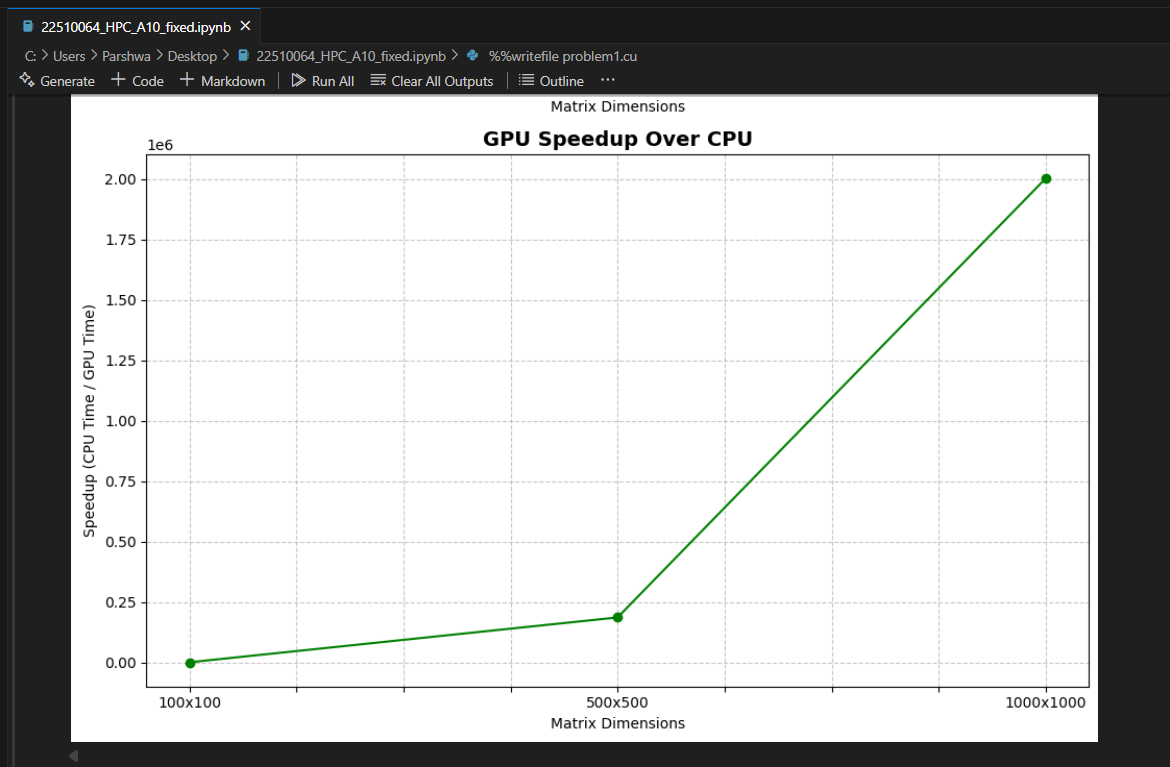
**• Write code for both serial (CPU-based) and parallel (CUDA-based) implementations.**

**• Measure the execution time of both implementations for various matrix sizes (e.g., 100x100, 500x500, 1000x1000).**

**Task:**

**• Calculate the speedup by comparing the CPU and GPU execution times.**

****

****