****

**Comsats University Islamabad, Lahore campus**

**ASSIGNMENT#1(Lab)**

**RAMSHA KHAN**

**(SP23-BCS-112)**

**Section: C**

**Course: PDC**

**Instructor’s name:Akhzar Nazir**

**Due date: 25-09-25**

**Part 1: Hello GPU with CUDA**

** Write a simple CUDA kernel that prints:**

**Hello from thread X**

** Understand how GPU threads, blocks, and grids work by experimenting with different**

**launch configurations.**

%%writefile hello.cu

#include <stdio.h>

\_\_global\_\_ void helloFromGPU() {

    int threadId = threadIdx.x;

    int blockId  = blockIdx.x;

    int globalId = blockIdx.x \* blockDim.x + threadIdx.x;

    printf("Hello from block %d, thread %d (global %d)\n", blockId, threadId, globalId);

}

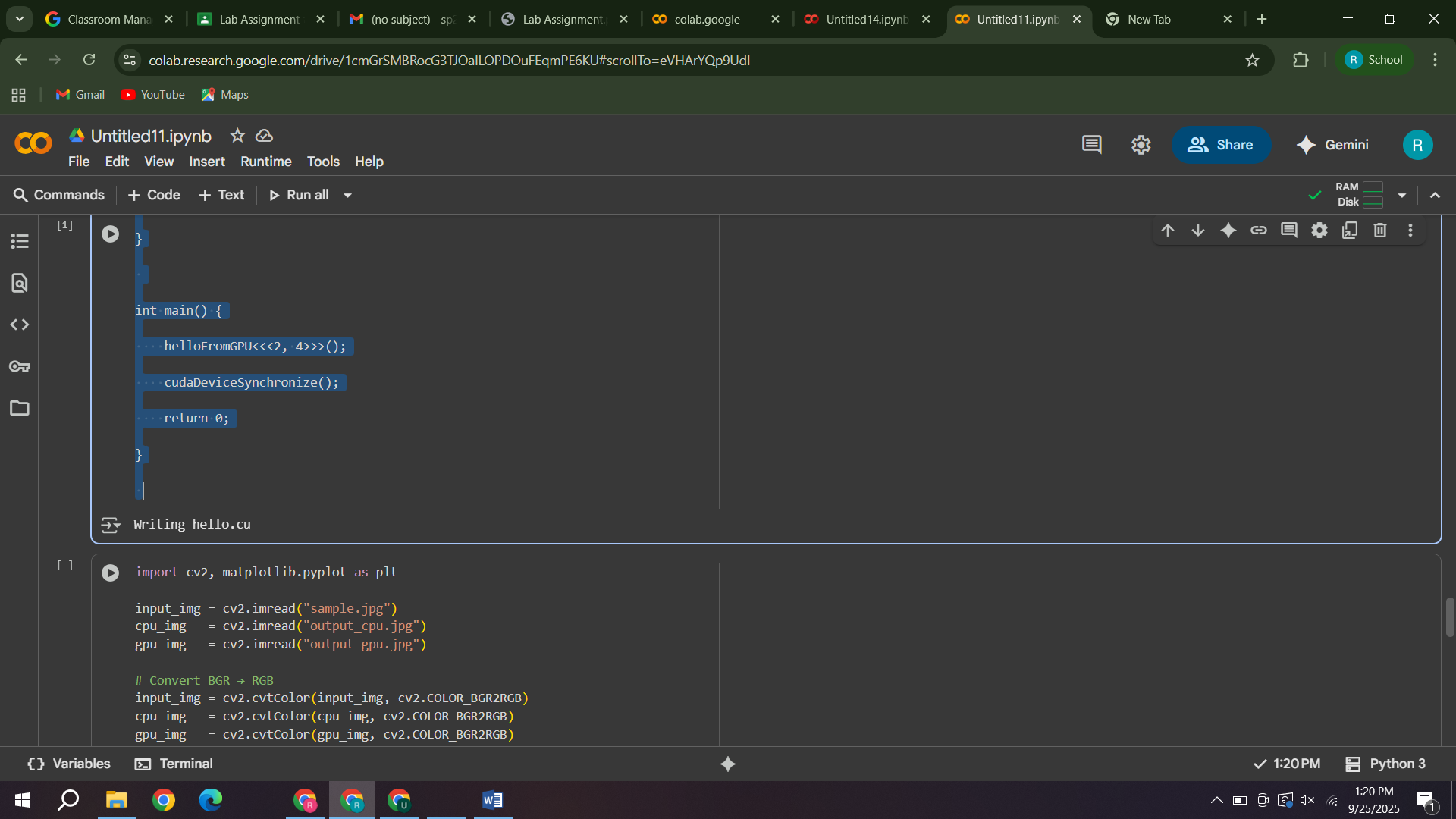
int main() {

    helloFromGPU<<<2, 4>>>();

    cudaDeviceSynchronize();

    return 0;

}



**Part 2: Vector Addition (CPU vs GPU)**

** Implement vector addition of two large arrays (e.g., 10 million elements):**

**o First on CPU (normal C++ loop).**

**o Then on GPU (CUDA kernel).**

** Measure the execution time of both.**

** Calculate the speedup ratio:**

**Speedup=CPU TimeGPU Time\text{Speedup} = \frac{\text{CPU Time}}{\text{GPU**

**Time}}Speedup=GPU TimeCPU Time​**

%%writefile vector\_add.cu

#include <stdio.h>

#include <cuda.h>

#include <chrono>

#include <iostream>

using namespace std;

\_\_global\_\_ void vectorAdd(float \*A, float \*B, float \*C, int n) {

    int i = blockIdx.x \* blockDim.x + threadIdx.x;

    if (i < n) {

        C[i] = A[i] + B[i];

    }

}

int main() {

    int n = 10000000;  // 10 million

    size\_t size = n \* sizeof(float);

    float \*h\_A = new float[n];

    float \*h\_B = new float[n];

    float \*h\_C = new float[n];

    for (int i = 0; i < n; i++) {

        h\_A[i] = 1.0f;

        h\_B[i] = 2.0f;

    }

    auto start\_cpu = chrono::high\_resolution\_clock::now();

    for (int i = 0; i < n; i++) {

        h\_C[i] = h\_A[i] + h\_B[i];

    }

    auto end\_cpu = chrono::high\_resolution\_clock::now();

    double cpu\_time = chrono::duration<double>(end\_cpu - start\_cpu).count();

    float \*d\_A, \*d\_B, \*d\_C;

    cudaMalloc((void\*\*)&d\_A, size);

    cudaMalloc((void\*\*)&d\_B, size);

    cudaMalloc((void\*\*)&d\_C, size);

    cudaMemcpy(d\_A, h\_A, size, cudaMemcpyHostToDevice);

    cudaMemcpy(d\_B, h\_B, size, cudaMemcpyHostToDevice);

    int threadsPerBlock = 256;

    int blocksPerGrid = (n + threadsPerBlock - 1) / threadsPerBlock;

    auto start\_gpu = chrono::high\_resolution\_clock::now();

    vectorAdd<<<blocksPerGrid, threadsPerBlock>>>(d\_A, d\_B, d\_C, n);

    cudaDeviceSynchronize();

    auto end\_gpu = chrono::high\_resolution\_clock::now();

    double gpu\_time = chrono::duration<double>(end\_gpu - start\_gpu).count();

    cudaMemcpy(h\_C, d\_C, size, cudaMemcpyDeviceToHost);

    printf("CPU time = %f sec\n", cpu\_time);

    printf("GPU time = %f sec\n", gpu\_time);

    printf("Speedup = %f\n", cpu\_time / gpu\_time);

    cudaFree(d\_A); cudaFree(d\_B); cudaFree(d\_C);

    delete[] h\_A; delete[] h\_B; delete[] h\_C;

    return 0;

}

**Part 3: Image Inversion (CPU vs GPU)**

** Load an image (e.g., PNG or JPG).**

** Implement pixel inversion:**

**new\_pixel=255−old\_pixel\text{new\\_pixel} = 255 -**

**\text{old\\_pixel}new\_pixel=255−old\_pixel**

** Do it once using a CPU loop, and again using a CUDA kernel.**

** Compare performance and verify that the output images are identical.**

%%writefile image\_invert.cu

#include <opencv2/opencv.hpp>

#include <iostream>

#include <cuda\_runtime.h>

using namespace cv;

using namespace std;

\_\_global\_\_ void invertImage(unsigned char\* input, unsigned char\* output, int width, int height, int channels) {

    int x = blockIdx.x \* blockDim.x + threadIdx.x;

    int y = blockIdx.y \* blockDim.y + threadIdx.y;

    if (x < width && y < height) {

        int idx = (y \* width + x) \* channels;

        for (int c = 0; c < channels; c++) {

            output[idx + c] = 255 - input[idx + c];

        }

    }

}

int main() {

    Mat input = imread("sample.jpg");

    if (input.empty()) {

        cout << "Error: Could not open image!" << endl;

        return -1;

    }

    int width = input.cols;

    int height = input.rows;

    int channels = input.channels();

    int img\_size = width \* height \* channels;

    unsigned char\* h\_input = input.data;

    unsigned char\* h\_output = new unsigned char[img\_size];

    // CPU inversion

    for (int i = 0; i < img\_size; i++) {

        h\_output[i] = 255 - h\_input[i];

    }

    Mat cpu\_img(height, width, CV\_8UC3, h\_output);

    imwrite("output\_cpu.jpg", cpu\_img.clone());

    // GPU inversion

    unsigned char \*d\_input, \*d\_output;

    cudaMalloc((void\*\*)&d\_input, img\_size);

    cudaMalloc((void\*\*)&d\_output, img\_size);

    cudaMemcpy(d\_input, h\_input, img\_size, cudaMemcpyHostToDevice);

    dim3 block(16, 16);

    dim3 grid((width + block.x - 1) / block.x, (height + block.y - 1) / block.y);

    invertImage<<<grid, block>>>(d\_input, d\_output, width, height, channels);

    cudaDeviceSynchronize();

    cudaMemcpy(h\_output, d\_output, img\_size, cudaMemcpyDeviceToHost);

    Mat gpu\_img(height, width, CV\_8UC3, h\_output);

    imwrite("output\_gpu.jpg", gpu\_img.clone());

    cudaFree(d\_input);

    cudaFree(d\_output);

    delete[] h\_output;

    cout << "Done: output\_cpu.jpg and output\_gpu.jpg generated" << endl;

    return 0;

}

import cv2

import matplotlib.pyplot as plt

# Load images

original\_img = cv2.imread("sample.jpg")

cpu\_img = cv2.imread("output\_cpu.jpg")

gpu\_img = cv2.imread("output\_gpu.jpg")

# Convert BGR to RGB for proper color display

original\_rgb = cv2.cvtColor(original\_img, cv2.COLOR\_BGR2RGB)

cpu\_rgb = cv2.cvtColor(cpu\_img, cv2.COLOR\_BGR2RGB)

gpu\_rgb = cv2.cvtColor(gpu\_img, cv2.COLOR\_BGR2RGB)

# Display side-by-side

plt.figure(figsize=(18,6))

plt.subplot(1, 3, 1)

plt.imshow(original\_rgb)

plt.title("Original Image")

plt.axis("off")

plt.subplot(1, 3, 2)

plt.imshow(cpu\_rgb)

plt.title("CPU Inverted Image")

plt.axis("off")

plt.subplot(1, 3, 3)

plt.imshow(gpu\_rgb)

plt.title("GPU Inverted Image")

plt.axis("off")

plt.show()

