**Assignment: Introduction to CUDA Programming**

In this assignment, you will explore **basic CUDA concepts** and learn how GPU computing provides massive speedup compared to CPU execution. Complete the following tasks step by step:

**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.

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

* Implement **vector addition** of two large arrays (e.g., 10 million elements):
  + First on **CPU** (normal C++ loop).
  + 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​

**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.

**Part 1:**

!pip install pycuda

import numpy as np

import pycuda.autoinit

import pycuda.gpuarray as gpuarray

from pycuda.compiler import SourceModule

# CUDA kernel

kernel\_code = """

\_\_global\_\_ void hello(int \*out)

{

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

    out[tid] = tid;  // write thread ID

}

"""

# Compile

mod = SourceModule(kernel\_code)

hello = mod.get\_function("hello")

# Prepare GPU array (8 threads total)

n\_threads = 8

out\_gpu = gpuarray.zeros(n\_threads, dtype=np.int32)

# Launch <<<2,4>>>

hello(out\_gpu, block=(4,1,1), grid=(2,1))

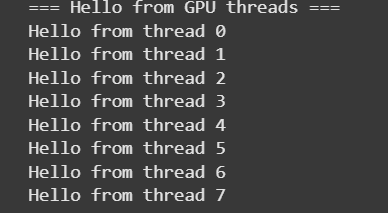
# Copy back and print

print("=== Hello from GPU threads ===")

for tid in out\_gpu.get():   # .get() copies back to host

    print(f"Hello from thread {tid}")

OUTPUT PART 1



**Part 2:**

!pip install pycuda

import numpy as np

import pycuda.autoinit

import pycuda.driver as cuda

from pycuda.compiler import SourceModule

import time

kernel\_code = """

\_\_global\_\_ void vector\_add(float \*a, float \*b, float \*c, int n)

{

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

    if (idx < n) {

        c[idx] = a[idx] + b[idx];

    }

}

"""

mod = SourceModule(kernel\_code)

vector\_add = mod.get\_function("vector\_add")

N = 10\_000\_000

a = np.random.rand(N).astype(np.float32)

b = np.random.rand(N).astype(np.float32)

c = np.zeros\_like(a)

a\_gpu = cuda.mem\_alloc(a.nbytes)

b\_gpu = cuda.mem\_alloc(b.nbytes)

c\_gpu = cuda.mem\_alloc(c.nbytes)

cuda.memcpy\_htod(a\_gpu, a)

cuda.memcpy\_htod(b\_gpu, b)

start = time.time()

c\_cpu = a + b

cpu\_time = time.time() - start

print(f"CPU Time: {cpu\_time:.4f} sec")

threads\_per\_block = 256

blocks\_per\_grid = (N + threads\_per\_block - 1) // threads\_per\_block

start = time.time()

vector\_add(a\_gpu, b\_gpu, c\_gpu, np.int32(N),

           block=(threads\_per\_block,1,1), grid=(blocks\_per\_grid,1))

cuda.Context.synchronize()

gpu\_time = time.time() - start

print(f"GPU Time: {gpu\_time:.4f} sec")

cuda.memcpy\_dtoh(c, c\_gpu)

print("Results match:", np.allclose(c, c\_cpu))

speedup = cpu\_time / gpu\_time

print(f"Speedup = {speedup:.2f}x")

**OUTPUT PART 2:**

A computer screen with white text

AI-generated content may be incorrect.

**Part 3:**

import numpy as np

import cupy as cp

import cv2

import time

H, W = 2048, 2048   # Adjust size if you want

gray = np.tile(np.arange(W, dtype=np.uint8), (H, 1))

cv2.imwrite("test\_input.png", gray)

print(f"Generated test image: {H}x{W}")

cpu\_start = time.time()

cpu\_inverted = np.empty\_like(gray)

for i in range(H):

    for j in range(W):

        cpu\_inverted[i, j] = 255 - gray[i, j]

cpu\_time = time.time() - cpu\_start

print(f"CPU loop inversion time: {cpu\_time:.4f} sec")

gpu\_start = time.time()

d\_img = cp.array(gray)

invert\_kernel = cp.RawKernel(r'''

extern "C" \_\_global\_\_

void invert(unsigned char\* img, unsigned char\* out, int H, int W) {

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

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

    if (x < W && y < H) {

        int idx = y \* W + x;

        out[idx] = 255 - img[idx];

    }

}

''', "invert")

d\_out = cp.empty\_like(d\_img)

threads = (16, 16)

blocks = ((W + threads[0] - 1) // threads[0],

          (H + threads[1] - 1) // threads[1])

invert\_kernel(blocks, threads, (d\_img, d\_out, H, W))

gpu\_inverted = d\_out.get()

gpu\_time = time.time() - gpu\_start

print(f"GPU inversion time: {gpu\_time:.4f} sec")

diff = np.abs(cpu\_inverted.astype(int) - gpu\_inverted.astype(int)).sum()

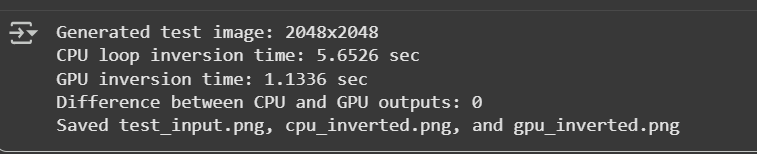
print("Difference between CPU and GPU outputs:", diff)

cv2.imwrite("cpu\_inverted.png", cpu\_inverted)

cv2.imwrite("gpu\_inverted.png", gpu\_inverted)

print("Saved test\_input.png, cpu\_inverted.png, and gpu\_inverted.png")

**OUTPUT PART 3**

****