Name: M. Musadiq

Reg No. SP23-BCS-095

Section: C

Lab Assignment: 1

Task1

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.

```
!pip install pycuda

import pycuda.driver as cuda

import pycuda.autoinit

from pycuda.compiler import SourceModule


mod = SourceModule("""

#include <stdio.h>


__global__ void hello_kernel() {

    int tid = threadIdx.x;

    printf("Hello from thread %d\\n", tid);

}

""")
```

```
hello_kernel = mod.get_function("hello_kernel")
hello_kernel(block=(5,1,1), grid=(1,1,1))
```

Output1:

```
=== Hello from GPU threads ===
Hello from thread 0
Hello from thread 1
Hello from thread 2
Hello from thread 3
Hello from thread 4
Hello from thread 5
Hello from thread 6
Hello from thread 7
```

Task2:

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:

```
import numpy as np

import time

import pycuda.autoinit

import pycuda.driver as cuda

from pycuda.compiler import SourceModule


N = 10_000_000

A = np.ones(N, dtype=np.float32)

B = np.full(N, 2.0, dtype=np.float32)

C_cpu = np.zeros_like(A)

C_gpu = np.zeros_like(A)


start_cpu = time.time()

C_cpu = A + B

end_cpu = time.time()

cpu_time = (end_cpu - start_cpu) * 1000


kernel_code = """

__global__ void vectorAdd(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("vectorAdd")

A_gpu = cuda.mem_alloc(A.nbytes)
B_gpu = cuda.mem_alloc(B.nbytes)
C_gpu_mem = cuda.mem_alloc(C_gpu.nbytes)

cuda.memcpy_htod(A_gpu, A)
cuda.memcpy_htod(B_gpu, B)

block_size = 256
grid_size = (N + block_size - 1) // block_size

start_gpu = cuda.Event()
end_gpu = cuda.Event()

start_gpu.record()
vector_add(A_gpu, B_gpu, C_gpu_mem, np.int32(N),
           block=(block_size, 1, 1), grid=(grid_size, 1))
```

```
end_gpu.record()
end_gpu.synchronize()

gpu_time = start_gpu.time_till(end_gpu)

cuda.memcpy_dtoh(C_gpu, C_gpu_mem)

correct = np.allclose(C_cpu, C_gpu)

print("=== Vector Addition (10 million elements) ===")
print(f"CPU time: {cpu_time:.2f} ms")
print(f"GPU time: {gpu_time:.2f} ms")
print(f"Speedup:  {cpu_time / gpu_time:.2f}x")
print(f"Correctness: {'PASS' if correct else 'FAIL'}")
```

Output:

```
=== Vector Addition (10 million elements) ===
CPU time: 15.90 ms
GPU time: 0.60 ms
Speedup:  26.31x
Correctness: PASS
```

Task3:

Image Inversion (CPU vs GPU)

- **Load an image (e.g., PNG or JPG).**
- **Implement pixel inversion:**

```
import cv2
import numpy as np
import time
import pycuda.autoinit
import pycuda.driver as cuda
from pycuda.compiler import SourceModule
from google.colab import files
from google.colab.patches import cv2_imshow

uploaded = files.upload()
filename = list(uploaded.keys())[0]

image = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
if image is None:
    raise ValueError("Could not load image!")

h, w = image.shape
N = h * w
print(f"Image size: {w}x{h} ({N} pixels)")

start_cpu = time.time()
cpu_inverted = 255 - image
end_cpu = time.time()
cpu_time = (end_cpu - start_cpu) * 1000
```

```
kernel_code = """
__global__ void invert_image(unsigned char *input, unsigned char *output, int n)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < n) {
        output[idx] = 255 - input[idx];
    }
}
"""

mod = SourceModule(kernel_code)
invert_image = mod.get_function("invert_image")

input_gpu = cuda.mem_alloc(image.nbytes)
output_gpu = cuda.mem_alloc(image.nbytes)

cuda.memcpy_htod(input_gpu, image)

block_size = 256
grid_size = (N + block_size - 1) // block_size

start_gpu = cuda.Event()
end_gpu = cuda.Event()

start_gpu.record()
invert_image(input_gpu, output_gpu, np.int32(N),
             block=(block_size, 1, 1), grid=(grid_size, 1))
```

```
end_gpu.record()
end_gpu.synchronize()

gpu_time = start_gpu.time_till(end_gpu)

gpu_inverted = np.empty_like(image)
cuda.memcpy_dtoh(gpu_inverted, output_gpu)

correct = np.array_equal(cpu_inverted, gpu_inverted)

cv2.imwrite("cpu_inverted.png", cpu_inverted)
cv2.imwrite("gpu_inverted.png", gpu_inverted)

print("=== Image Inversion ===")
print(f"CPU time: {cpu_time:.2f} ms")
print(f"GPU time: {gpu_time:.2f} ms")
print(f"Speedup: {cpu_time / gpu_time:.2f}x")
print(f"Correctness: {'PASS' if correct else 'FAIL'}")

print("\nShowing results (CPU vs GPU inversion):")
cv2_imshow(cpu_inverted)
cv2_imshow(gpu_inverted)
```

Output:

Image Inversion

CPU time: 3.04 ms

GPU time: 0.27 ms

Speedup: 11.23x

Correctness: PASS