|  |
| --- |
| OMAR ISMAIL – SP23-BCS-110 |
| ASSIGNMENT 1 |
| PARALLEL AND DISTRIBUTED COMPUTING |

1. **HELLO THREADS PROGRAM:**

**CODE:**

1. import pycuda.autoinit
2. import pycuda.driver as drv
3. from pycuda.compiler import SourceModule
4. import numpy as np
5. mod = SourceModule("""
6. \_\_global\_\_ void hello\_thread(int \*out)
7. {
8. int tid = threadIdx.x + blockIdx.x \* blockDim.x;
9. out[tid] = tid;
10. }
11. """)
12. hello\_thread = mod.get\_function("hello\_thread")
13. nthreads = 8
14. out\_gpu = drv.mem\_alloc(nthreads \* np.int32().nbytes)
15. hello\_thread(out\_gpu, block=(4,1,1), grid=(2,1,1))
16. out\_host = np.empty(nthreads, dtype=np.int32)
17. drv.memcpy\_dtoh(out\_host, out\_gpu)
18. for tid in out\_host:
19. print(f"Hello from thread {tid}")

**NOTE:**

Used pyCuda for cuda implementation, if the memory is not copied from device to host, output is not showed.

1. **VECTOR ADDITION OF 2 LISTS:**

**CPU:**

Used simple vector addition which took around 1.8 – 2 seconds to finish.

**CODE:**

import time

n = 10\_000\_000

a = list(range(n))

b = list(range(n))

start = time.time()

for i in range(n):

  c = [a[i] + b[i]]

end = time.time()

print("Time taken:", end - start, "seconds")

**GPU:**

Used CUDA to perform vector addition of two lists. The average execution time was between 0.0006 – 0.0009 seconds

**CODE:**

import cupy as cp

import time

n = 10\_000\_000

a = cp.arange(n, dtype=cp.float32)

b = cp.arange(n, dtype=cp.float32)

start = time.time()

c = a + b

cp.cuda.runtime.deviceSynchronize()

end = time.time()

print("Time taken:", end - start, "seconds")

**SPEEDUP RATIO:**

**FOR GPU:**

Time taken: 0.0006968975067138672 seconds

**FOR CPU:**

Time taken: 1.906116008758545 seconds

**RATIO:**

Speedup ratio = 2735.12

1. **IMAGE INVERSION:**

**GPU:**

**CODE:**

import cupy as cp

import numpy as np

import time

image\_matrix = np.random.randint(0, 256, (4096, 4096), dtype=np.uint8)

d\_image = cp.array(image\_matrix)

start = time.time()

d\_image = 255 - d\_image

cp.cuda.runtime.deviceSynchronize()

end = time.time()

inverted = cp.asnumpy(d\_image)

print("Original:\n", image\_matrix)

print("Inverted:\n", inverted)

print("Execution time:", end - start, "seconds")

Execution time: 0.0022580623626708984 seconds

**CPU:**

**CODE:**

import numpy as np

import time

# Create random image

image\_matrix = np.random.randint(0, 256, (4096, 4096), dtype=np.uint8)

# Measure time

start = time.time()

# Invert image (elementwise)

inverted = 255 - image\_matrix

end = time.time()

print("Original:\n", image\_matrix)

print("Inverted:\n", inverted)

print("Execution time:", end - start, "seconds")

Execution time: 0.004038572311401367 seconds

**NOTE:** For very large images, cuda processing is faster that cpu numpy otherwise for small images numpy cpu is way faster due to less time wasted in handing over requests from cpu to gpu.