**Code 1**

import cupy as cp

# Define vectors

a = cp.arange(10, dtype=cp.float32)

b = cp.arange(10, dtype=cp.float32)

print("vector a = ",a)

print("vector b = ",b)

# Elementwise addition (runs on GPU)

c = a + b

print("Result:", c)

# Custom CUDA kernel (Hello World style)

kernel\_code = r'''

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

void add\_arrays(const float\* a, const float\* b, float\* c, int n) {

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

printf("blockIdx.x=%d, threadIdx.x=%d, idx=%d\n", blockIdx.x, threadIdx.x, idx);

if (idx < n) {

c[idx] = a[idx] + b[idx];

}

}

'''

add\_arrays = cp.RawKernel(kernel\_code, 'add\_arrays')

n = 10

c = cp.zeros(n, dtype=cp.float32)

threads = 16

blocks = 1

print("Number of blocks = ",blocks)

add\_arrays((blocks,), (threads,), (a, b, c, n))

print("Kernel Result:", c)

**Code 2**

import cupy as cp

import time

# Define a simple element-wise kernel (adding 1 to each element)

kernel\_code = r'''

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

void add\_one(float\* x, int N) {

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

if (idx < N) {

x[idx] += 1.0f;

}

}

'''

# Compile kernel

module = cp.RawModule(code=kernel\_code)

add\_one\_kernel = module.get\_function("add\_one")

# Set N

N = 10\_000\_000

x = cp.random.rand(N, dtype=cp.float32)

# Function to run kernel and measure time

def run\_kernel(threads\_per\_block):

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

start = time.time()

add\_one\_kernel((blocks,), (threads\_per\_block,), (x, N))

cp.cuda.Device().synchronize()

end = time.time()

print(f"Threads per block: {threads\_per\_block}, Blocks: {blocks}, Time: {end-start:.5f} s")

# Run with inefficient 2 threads per block

run\_kernel(2)

# Run with inefficient 4 threads per block

run\_kernel(4)

# Run with inefficient 8 threads per block

run\_kernel(8)

# Run with inefficient 16 threads per block

run\_kernel(16)

# Run with efficient 32 threads per block

run\_kernel(32)

# Run with more optimized 128 threads per block (optional)

run\_kernel(128)

# Run with more optimized 256 threads per block (optional)

run\_kernel(256)

# Run with more optimized 512 threads per block (optional)

run\_kernel(512)

# Run with more optimized 1024 threads per block (optional)

run\_kernel(1024)

# Run with more optimized 2048 threads per block (optional)

#run\_kernel(2048)

**Code 3**

import cupy as cp

kernel\_code = r'''

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

void hello\_threads() {

printf("blockIdx=(%d,%d), threadIdx=(%d,%d), blockDim=(%d,%d), gridDim=(%d,%d)\n",

blockIdx.x, blockIdx.y,

threadIdx.x, threadIdx.y,

blockDim.x, blockDim.y,

gridDim.x, gridDim.y);

}

'''

hello\_threads = cp.RawKernel(kernel\_code, 'hello\_threads')

# --- Configure launch ---

threads\_per\_block = (2, 2) # 2x2 threads = 4 threads per block

blocks\_per\_grid = (3, 2) # 3x2 blocks = 6 blocks in grid

# Launch

hello\_threads(blocks\_per\_grid, threads\_per\_block)