LAB ASSIGNMENT 3

Name: Muhammad Tayyab

Roll No: SP23-BCS-099

Section: C
Course: PDC

Teacher: Sir Akhzar Nazir

You are asked to implement a CUDA program that performs the following tasks step by step:

Answer:

```
#include <stdio.h>

#include <stdlib.h>

#include <cuda_runtime.h>


#define N 1024

#define BLOCK 256


// ---------- Kernels ----------

// Step 2a: C[i] = A[i] + B[i]
```

```
__global__ void kernel1(int *A, int *B, int *C) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < N) C[idx] = A[idx] + B[idx];
}
// Step 2b: D[i] = C[i] * C[i]
__global__ void kernel2(int *C, int *D) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < N) D[idx] = C[idx] * C[idx];
}
// Step 6: Sum reduction using shared memory + atomicAdd
__global__ void sumReduction(int *D, int *sum) {
    __shared__ int sdata[BLOCK];
    int tid = threadIdx.x, i = blockIdx.x * blockDim.x + tid;
    sdata[tid] = (i < N) ? D[i] : 0;
    __syncthreads();
    for (int s = BLOCK / 2; s > 0; s >>= 1) {
        if (tid < s) sdata[tid] += sdata[tid + s];
        __syncthreads();
    }
    if (tid == 0) atomicAdd(sum, sdata[0]);
}

int main() {
```

```
// ---------- Step 1: Memory ----------
int *h_A = (int*)malloc(N * sizeof(int));
int *h_B = (int*)malloc(N * sizeof(int));
int *h_C = (int*)malloc(N * sizeof(int));
int *h_D = (int*)malloc(N * sizeof(int));
int h_sum = 0;

for (int i = 0; i < N; i++) { h_A[i] = i; h_B[i] = 2 * i; }

int *d_A, *d_B, *d_C, *d_D, *d_sum;
cudaMalloc(&d_A, N * sizeof(int));
cudaMalloc(&d_B, N * sizeof(int));
cudaMalloc(&d_C, N * sizeof(int));
cudaMalloc(&d_D, N * sizeof(int));
cudaMalloc(&d_sum, sizeof(int));
cudaMemcpy(d_A, h_A, N * sizeof(int), cudaMemcpyHostToDevice);
cudaMemcpy(d_B, h_B, N * sizeof(int), cudaMemcpyHostToDevice);
cudaMemset(d_sum, 0, sizeof(int));

// ---------- Step 2: Kernels on default stream ----------
kernel1<<<N / BLOCK, BLOCK>>>(d_A, d_B, d_C);
kernel2<<<N / BLOCK, BLOCK>>>(d_C, d_D);
cudaDeviceSynchronize();
```

```
cudaMemcpy(h_C, d_C, N * sizeof(int), cudaMemcpyDeviceToHost);
cudaMemcpy(h_D, d_D, N * sizeof(int), cudaMemcpyDeviceToHost);
printf("Step 2 -> C[10] = %d, D[10] = %d\n", h_C[10], h_D[10]);

// ---------- Step 3: Streams ----------
cudaStream_t s1, s2;
cudaStreamCreate(&s1); cudaStreamCreate(&s2);
kernel1<<<N / BLOCK, BLOCK, 0, s1>>>(d_A, d_B, d_C);
kernel2<<<N / BLOCK, BLOCK, 0, s2>>>(d_C, d_D); // may race!
cudaStreamSynchronize(s1); cudaStreamSynchronize(s2);
printf("Step 3 -> Possible race if kernel2 reads before kernel1 finishes.\n");

// ---------- Step 4: Synchronization ----------
kernel1<<<N / BLOCK, BLOCK>>>(d_A, d_B, d_C);
kernel2<<<N / BLOCK, BLOCK>>>(d_C, d_D);
// cudaDeviceSynchronize(); // uncomment to ensure correctness
cudaMemcpy(h_D, d_D, N * sizeof(int), cudaMemcpyDeviceToHost);
printf("Step 4 -> Without sync, CPU may read before GPU finishes.\n");

// ---------- Step 5: Thread Hierarchy ----------
kernel1<<<1, N>>>(d_A, d_B, d_C); // 1 block, N threads
cudaMemcpy(h_C, d_C, N * sizeof(int), cudaMemcpyDeviceToHost);
printf("Step 5a -> <<<1,N>>>: C[5] = %d\n", h_C[5]);
```

```
    kernel1<<<N / 32, 32>>>(d_A, d_B, d_C); // N/32 blocks, 32 threads
    cudaMemcpy(h_C, d_C, N * sizeof(int), cudaMemcpyDeviceToHost);
    printf("Step 5b -> <<<N/32,32>>>: C[5] = %d\n", h_C[5]);

    // ---------- Step 6: Reduction ----------
    sumReduction<<<N / BLOCK, BLOCK>>>(d_D, d_sum);
    cudaMemcpy(&h_sum, d_sum, sizeof(int), cudaMemcpyDeviceToHost);
    printf("Step 6 -> Sum of D = %d\n", h_sum);

    // ---------- Cleanup ----------
    free(h_A); free(h_B); free(h_C); free(h_D);
    cudaFree(d_A); cudaFree(d_B); cudaFree(d_C); cudaFree(d_D);
cudaFree(d_sum);
    cudaStreamDestroy(s1); cudaStreamDestroy(s2);

    return 0;
}
```