CUDA ASSIGNMENT :

SUPRATIM NAG (CSE-AIML/22/57)

1. Take an array of 10 numbers and perform the summation of these 10 numbers. Use a kernel function.

```
%%cuda
#include <stdio.h>
#include <cuda_runtime.h>

__global__ void sumKernel(int *array, int *result, int n) {
    __shared__ int partialSum[10];  // Shared memory for partial sums

    int tid = threadIdx.x;
    partialSum[tid] = (tid < n) ? array[tid] : 0;  // Load elements into shared memory

    __syncthreads();

    // Perform parallel reduction within the block
    for (int stride = 1; stride < blockDim.x; stride *= 2) {
        if (tid % (2 * stride) == 0) {
            partialSum[tid] += partialSum[tid + stride];
        }
        __syncthreads();
    }

    // First thread in the block writes the result
    if (tid == 0) {
        *result = partialSum[0];
    }
}

int main() {
    const int n = 10;
    int h_array[n] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};  // Example array of 10 numbers
    int h_result = 0;  // Host variable for result

    // Device variables
    int *d_array, *d_result;
    cudaMalloc((void **)&d_array, n * sizeof(int));
    cudaMalloc((void **)&d_result, sizeof(int));

    // Copy array from host to device
    cudaMemcpy(d_array, h_array, n * sizeof(int), cudaMemcpyHostToDevice);

    // Launch kernel with 10 threads in a single block
    sumKernel<<<1, n>>>(d_array, d_result, n);

    // Copy result back to host
    cudaMemcpy(&h_result, d_result, sizeof(int), cudaMemcpyDeviceToHost);

    // Print the result
    printf("Sum of array elements: %d\n", h_result);

    // Free device memory
    cudaFree(d_array);
    cudaFree(d_result);

    return 0;
}
```

```
Sum of array elements: 55
```

2. Take three vectors consisting of 10 elements each and add them and store it in a 4th vector.

```
%%cuda
#include <stdio.h>
#include <cuda_runtime.h>

#define N 10  // Number of elements in each vector

// Kernel function to add vectors
__global__ void vectorAdd(int *A, int *B, int *C, int *D, int n) {
    int tid = threadIdx.x;

    if (tid < n) {
        // Perform element-wise addition and store it in vector D
        D[tid] = A[tid] + B[tid] + C[tid];
    }
}

int main() {
    int h_A[N], h_B[N], h_C[N], h_D[N];  // Host vectors
    int *d_A, *d_B, *d_C, *d_D;           // Device vectors

    // Initialize vectors A, B, and C
    for (int i = 0; i < N; i++) {
        h_A[i] = i + 1;  // Vector A: 1, 2, 3, ..., 10
        h_B[i] = (i + 1) * 2;  // Vector B: 2, 4, 6, ..., 20
        h_C[i] = (i + 1) * 3;  // Vector C: 3, 6, 9, ..., 30
    }

    // Allocate memory on the device
    cudaMalloc((void **)&d_A, N * sizeof(int));
    cudaMalloc((void **)&d_B, N * sizeof(int));
    cudaMalloc((void **)&d_C, N * sizeof(int));
    cudaMalloc((void **)&d_D, N * sizeof(int));

    // Copy data from host to device
    cudaMemcpy(d_A, h_A, N * sizeof(int), cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, h_B, N * sizeof(int), cudaMemcpyHostToDevice);
    cudaMemcpy(d_C, h_C, N * sizeof(int), cudaMemcpyHostToDevice);

    // Launch the kernel with N threads
    vectorAdd<<<1, N>>>(d_A, d_B, d_C, d_D, N);

    // Copy the result from device to host
    cudaMemcpy(h_D, d_D, N * sizeof(int), cudaMemcpyDeviceToHost);

    // Print the result
    printf("Resulting vector D after adding A, B, and C:\n");
    for (int i = 0; i < N; i++) {
        printf("%d ", h_D[i]);
    }
    printf("\n");

    // Free device memory
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);
    cudaFree(d_D);

    return 0;
}
```

```
    return 0;
}
```

```
Resulting vector D after adding A, B, and C:
6 12 18 24 30 36 42 48 54 60
```

3. Take 3 scalar variables and assign floating point values to them then perform the multiplication and store it in 4th variable.

```
%%cuda
#include <stdio.h>
#include <cuda_runtime.h>

// Kernel function to multiply three scalars
__global__ void scalarMultiply(float *a, float *b, float *c, float *result) {
    // Perform the multiplication and store the result
    *result = (*a) * (*b) * (*c);
}

int main() {
    // Declare and initialize the scalar variables
    float h_a = 2.5f, h_b = 3.5f, h_c = 4.0f;  // Host variables
    float h_result = 0.0f;  // Host variable for storing result

    // Device variables
    float *d_a, *d_b, *d_c, *d_result;

    // Allocate memory on the device
    cudaMalloc((void **)&d_a, sizeof(float));
    cudaMalloc((void **)&d_b, sizeof(float));
    cudaMalloc((void **)&d_c, sizeof(float));
    cudaMalloc((void **)&d_result, sizeof(float));

    // Copy data from host to device
    cudaMemcpy(d_a, &h_a, sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, &h_b, sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(d_c, &h_c, sizeof(float), cudaMemcpyHostToDevice);

    // Launch the kernel (1 block, 1 thread)
    scalarMultiply<<<1, 1>>>(d_a, d_b, d_c, d_result);

    // Copy the result from device to host
    cudaMemcpy(&h_result, d_result, sizeof(float), cudaMemcpyDeviceToHost);

    // Print the result
    printf("The result of multiplying %.2f, %.2f, and %.2f is: %.2f\n", h_a, h_b, h_c, h_result);

    // Free device memory
    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);
    cudaFree(d_result);

    return 0;
}
```

```
The result of multiplying 2.50, 3.50, and 4.00 is: 35.00
```

4. Write a kernel function to swap two elements without the use of 3rd variable.

```
%%cuda
#include <stdio.h>
#include <cuda_runtime.h>

// Kernel function to swap two elements without using a third variable
__global__ void swapKernel(int *a, int *b) {
    // Swap the elements using arithmetic operations (addition and subtraction)
    *a = *a + *b;  // a = a + b
    *b = *a - *b;  // b = (a + b) - b = a
    *a = *a - *b;  // a = (a + b) - a = b
}

int main() {
    int h_a = 5, h_b = 10;  // Host variables
    int *d_a, *d_b;  // Device variables

    // Allocate memory on the device
    cudaMalloc((void **)&d_a, sizeof(int));
    cudaMalloc((void **)&d_b, sizeof(int));

    // Copy data from host to device
    cudaMemcpy(d_a, &h_a, sizeof(int), cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, &h_b, sizeof(int), cudaMemcpyHostToDevice);

    // Launch the kernel (1 block, 1 thread)
    swapKernel<<<1, 1>>>(d_a, d_b);

    // Copy the result back from device to host
    cudaMemcpy(&h_a, d_a, sizeof(int), cudaMemcpyDeviceToHost);
    cudaMemcpy(&h_b, d_b, sizeof(int), cudaMemcpyDeviceToHost);

    // Print the swapped values
    printf("After swapping, a = %d and b = %d\n", h_a, h_b);

    // Free device memory
    cudaFree(d_a);
    cudaFree(d_b);

    return 0;
}
```

```
After swapping, a = 10 and b = 5
```