

%%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);

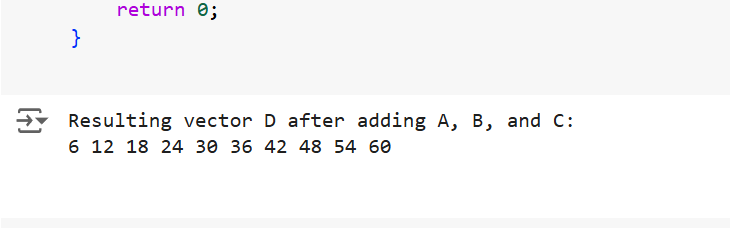
    return 0;

}

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

CUDA ASSIGNMENT :

SUPRATIM NAG (CSE-AIML/22/57)



%%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);

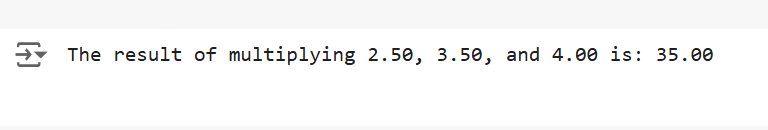
    cudaFree(d\_C);

    cudaFree(d\_D);

    return 0;

}

1. **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>

// 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;

}

**3. Take 3 scalar variables and assign floating point values to them then perform the**

**multiplication and store it in 4th variable.**

**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;

}

