**Lab Exercise 6.5 – Host and Device Interaction in CUDA**

**Objective:**

* Understand the interaction between the host (CPU) and the device (GPU) in CUDA programming.
* Learn how to allocate memory on the device, transfer data between host and device, and launch kernel functions to process data on the GPU.

**1. Introduction to Host and Device Interaction:**

* **Host**: The CPU, where the program runs.
* **Device**: The GPU, where the parallel computations take place.
* **Memory management**: Data needs to be transferred between the host and the device. CUDA provides functions like cudaMalloc, cudaMemcpy, and cudaFree for memory allocation and data transfer.

**2. Basic Steps for Host-Device Interaction:**

* **Step 1**: Allocate memory on the host.
* **Step 2**: Allocate memory on the device using cudaMalloc.
* **Step 3**: Copy data from the host to the device using cudaMemcpy.
* **Step 4**: Launch a kernel to process the data on the device.
* **Step 5**: Copy the results back from the device to the host using cudaMemcpy.
* **Step 6**: Free memory on both the host and the device.

**3. Sample CUDA Program: Host-Device Interaction**

This simple program demonstrates how to add two arrays on the GPU, using the host to control the memory and data transfer.

#include <iostream>

#include <cuda\_runtime.h>

#define N 10 // Size of the arrays

// Kernel function to add elements of two arrays

\_\_global\_\_ void addArrays(int \*a, int \*b, int \*c) {

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

if (idx < N) {

c[idx] = a[idx] + b[idx]; // Perform element-wise addition

}

}

int main() {

// Host arrays

int h\_a[N], h\_b[N], h\_c[N];

// Initialize host arrays

for (int i = 0; i < N; i++) {

h\_a[i] = i;

h\_b[i] = 2 \* i;

}

// Device arrays

int \*d\_a, \*d\_b, \*d\_c;

// 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));

// Copy data from host to device

cudaMemcpy(d\_a, h\_a, N \* sizeof(int), cudaMemcpyHostToDevice);

cudaMemcpy(d\_b, h\_b, N \* sizeof(int), cudaMemcpyHostToDevice);

// Define block and grid size

int blockSize = 256;

int numBlocks = (N + blockSize - 1) / blockSize;

// Launch kernel

addArrays<<<numBlocks, blockSize>>>(d\_a, d\_b, d\_c);

// Synchronize to ensure the kernel has finished

cudaDeviceSynchronize();

// Copy results from device to host

cudaMemcpy(h\_c, d\_c, N \* sizeof(int), cudaMemcpyDeviceToHost);

// Print the results

std::cout << "Array A: ";

for (int i = 0; i < N; i++) {

std::cout << h\_a[i] << " ";

}

std::cout << std::endl;

std::cout << "Array B: ";

for (int i = 0; i < N; i++) {

std::cout << h\_b[i] << " ";

}

std::cout << std::endl;

std::cout << "Array C (Sum): ";

for (int i = 0; i < N; i++) {

std::cout << h\_c[i] << " ";

}

std::cout << std::endl;

// Free memory on the device

cudaFree(d\_a);

cudaFree(d\_b);

cudaFree(d\_c);

return 0;

}

**4. Explanation of the Code:**

**Step 1: Memory Allocation on Host:**

int h\_a[N], h\_b[N], h\_c[N];

* We define three arrays (h\_a, h\_b, h\_c) on the host to store the input data and the result of the addition.

**Step 2: Initialize Host Arrays:**

for (int i = 0; i < N; i++) {

h\_a[i] = i;

h\_b[i] = 2 \* i;

}

* We initialize the host arrays h\_a and h\_b with some values. For example:
  + h\_a = {0, 1, 2, 3, ..., 9}
  + h\_b = {0, 2, 4, 6, ..., 18}

**Step 3: Memory Allocation on Device:**

cudaMalloc((void\*\*)&d\_a, N \* sizeof(int));

cudaMalloc((void\*\*)&d\_b, N \* sizeof(int));

cudaMalloc((void\*\*)&d\_c, N \* sizeof(int));

* We use cudaMalloc to allocate memory for the device arrays (d\_a, d\_b, d\_c). These arrays will be used to store the input and output data on the GPU.

**Step 4: Copy Data from Host to Device:**

cudaMemcpy(d\_a, h\_a, N \* sizeof(int), cudaMemcpyHostToDevice);

cudaMemcpy(d\_b, h\_b, N \* sizeof(int), cudaMemcpyHostToDevice);

* We use cudaMemcpy to copy the data from the host arrays h\_a and h\_b to the device arrays d\_a and d\_b.

**Step 5: Launch Kernel:**

addArrays<<<numBlocks, blockSize>>>(d\_a, d\_b, d\_c);

* The kernel addArrays is launched with numBlocks blocks and blockSize threads per block.
* The kernel will execute on the device, where each thread adds the corresponding elements from d\_a and d\_b and stores the result in d\_c.

**Step 6: Synchronize the Device:**

cudaDeviceSynchronize();

* We use cudaDeviceSynchronize to make sure the kernel finishes executing before proceeding with the next steps. This is important to ensure that the data has been processed on the device.

**Step 7: Copy Results from Device to Host:**

cudaMemcpy(h\_c, d\_c, N \* sizeof(int), cudaMemcpyDeviceToHost);

* We copy the results from the device array d\_c back to the host array h\_c.

**Step 8: Display Results:**

std::cout << "Array A: ";

for (int i = 0; i < N; i++) {

std::cout << h\_a[i] << " ";

}

* We print the values of the input arrays h\_a, h\_b, and the result array h\_c.

**Step 9: Free Device Memory:**

cudaFree(d\_a);

cudaFree(d\_b);

cudaFree(d\_c);

* After the computations are done, we free the memory allocated on the device using cudaFree.

**5. Expected Output:**

After running the program, the output should display:

Array A: 0 1 2 3 4 5 6 7 8 9

Array B: 0 2 4 6 8 10 12 14 16 18

Array C (Sum): 0 3 6 9 12 15 18 21 24 27

* Array A and Array B are the input arrays.
* Array C (Sum) is the result of element-wise addition of arrays A and B.

**6. Conclusion:**

* This lab exercise demonstrates how to interact with both the host (CPU) and the device (GPU) in CUDA programming.
* It shows how to allocate memory, transfer data between the host and device, and launch a kernel on the GPU.
* Host-device interaction is a fundamental concept for efficient GPU computing, as it allows us to utilize the GPU for parallel computation while managing data from the host.