**Lab Exercise 11.2 – Memory Objects – Global, Shared, and Local Memory in CUDA**

**Objective:**

* Understand how to allocate memory on the host (CPU) and device (GPU).
* Learn how to transfer data between host and device.
* Write a CUDA program that demonstrates these concepts.

**1. Key CUDA Memory Functions**

| **Function** | **Description** |
| --- | --- |
| cudaMalloc() | Allocates memory on the device |
| cudaFree() | Frees memory on the device |
| cudaMemcpy() | Transfers data between host and device |
| cudaMemcpyHostToDevice | Copy data from host to device |
| cudaMemcpyDeviceToHost | Copy data from device to host |

**2. Program: Demonstrating Memory Allocation and Data Transfer**

#include <iostream>

#include <cuda\_runtime.h>

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

int i = threadIdx.x;

if (i < N)

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

}

int main() {

const int N = 5;

int h\_a[N] = {1, 2, 3, 4, 5};

int h\_b[N] = {10, 20, 30, 40, 50};

int h\_c[N];

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

// Allocate device memory

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);

// Launch kernel on device

addKernel<<<1, N>>>(d\_a, d\_b, d\_c, N);

// Copy result from device to host

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

std::cout << "Result from device:\n";

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

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

std::cout << std::endl;

// Free device memory

cudaFree(d\_a);

cudaFree(d\_b);

cudaFree(d\_c);

return 0;

}

**3. Explanation**

* h\_a, h\_b, h\_c: Host arrays.
* d\_a, d\_b, d\_c: Device (GPU) pointers.
* cudaMalloc: Allocates memory on the GPU.
* cudaMemcpy: Transfers data between host and device.
* addKernel: A kernel that adds elements from d\_a and d\_b and stores result in d\_c.
* <<<1, N>>>: Launches the kernel with 1 block of N threads.
* cudaFree: Frees GPU memory after use.

**4. Compile and Run**

nvcc -o mem\_transfer mem\_transfer.cu

./mem\_transfer

Expected output:

Result from device:

11 22 33 44 55

**5. Summary**

* **Memory allocation and data transfer** are foundational to CUDA programming.
* Always allocate device memory before launching kernels.
* Properly manage memory to avoid leaks and segmentation faults.
* Transfers between host and device can impact performance — minimize when possible.