Experiment 7

**Aim: To implement programs based on device global memory and data transfer.**

**Assignment 1: Implement a program in CUDA for addition of two numbers.**

**Steps for environment configuration :**

### Step 1 : Install extension that allows users to run CUDA C/C++ code directly within Jupyter notebooks.

pip install nvcc4jupyter

### Step 2 : Load nvcc4jupyter extension.

%load\_ext nvcc4jupyter

### Step 3 : CUDA code for addition of two numbers.

%%cuda

#include <stdio.h>

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

   \*c = \*a + \*b;

}

int main() {

   int a, b, c;

   int \*dev\_a, \*dev\_b, \*dev\_c;

   int size = sizeof(int);

   cudaMalloc((void\*\*)&dev\_a, size);

   cudaMalloc((void\*\*)&dev\_b, size);

   cudaMalloc((void\*\*)&dev\_c, size);

   a = 30;

   b = 6;

   cudaMemcpy(dev\_a, &a, size, cudaMemcpyHostToDevice);

   cudaMemcpy(dev\_b, &b, size, cudaMemcpyHostToDevice);

   Addition<<<1, 1>>>(dev\_a, dev\_b, dev\_c);

   cudaMemcpy(&c, dev\_c, size, cudaMemcpyDeviceToHost);

   cudaFree(dev\_a);

   cudaFree(dev\_b);

   cudaFree(dev\_c);

   printf("Result of addition is:%d\n", c);

   return 0;

}

**Assignment 2: Implement a program in CUDA that calculates the sum of two arrays on the GPU and then transfers the result back to the host for printing.**

%%cuda

#include <stdio.h>

// Kernel function to add two arrays element-wise

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

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

if (idx < size) {

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

}

}

int main() {

const int size = 5;

int a[size] = {1, 2, 3, 4, 5};

int b[size] = {10, 20, 30, 40, 50};

int c[size]; // Result array

int \*d\_a, \*d\_b, \*d\_c; // Device arrays

// Allocate memory on the device

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

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

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

// Transfer data from host to device

cudaMemcpy(d\_a, a, size \* sizeof(int), cudaMemcpyHostToDevice);

cudaMemcpy(d\_b, b, size \* sizeof(int), cudaMemcpyHostToDevice);

// Define grid and block dimensions

int threadsPerBlock = 256;

int blocksPerGrid = (size + threadsPerBlock - 1) / threadsPerBlock;

// Launch kernel

addArrays<<<blocksPerGrid, threadsPerBlock>>>(d\_a, d\_b, d\_c, size);

// Wait for kernel to finish

cudaDeviceSynchronize();

// Transfer result from device to host

cudaMemcpy(c, d\_c, size \* sizeof(int), cudaMemcpyDeviceToHost);

// Print the result

printf("Result of adding two arrays is :\n");

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

printf("%d + %d = %d\n", a[i], b[i], c[i]);

}

// Free device memory

cudaFree(d\_a);

cudaFree(d\_b);

cudaFree(d\_c);

return 0;

}