# ASSIGNMENT - II

```
#include <stdio.h>
#include <stdlib.h>
#include <cuda.h>
#include <time.h>
#include <math.h>

// CUDA kernel for matrix multiplication
__global__ void matrixMulKernel(float *A, float *B, float *C, int N) {
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    float sum = 0.0f;

    if (row < N && col < N) {
        for (int k = 0; k < N; k++) {
            sum += A[row * N + k] * B[k * N + col];
        }
        C[row * N + col] = sum;
    }
}
// CUDA kernel for matrix transpose
__global__ void transposeKernel(float *A, float *B, int N) {
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    if (row < N && col < N) {
        B[col * N + row] = A[row * N + col];
    }
}
// CPU function for matrix multiplication
void matrixMulCPU(float *A, float *B, float *C, int N) {
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++) {
            float sum = 0.0f;
```

```
            for (int k = 0; k < N; k++)
                sum += A[i * N + k] * B[k * N + j];
            C[i * N + j] = sum;
        }
}

// CPU function for matrix transpose
void transposeCPU(float *A, float *B, int N) {
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            B[j * N + i] = A[i * N + j];
}

int main() {
    int N;
    printf("Enter matrix size N (e.g., 3 for 3x3): ");
    scanf("%d", &N);

    size_t size = N * N * sizeof(float);
    float *A = (float *)malloc(size);
    float *B = (float *)malloc(size);
    float *C_cpu = (float *)malloc(size);
    float *C_gpu = (float *)malloc(size);
    float *T_cpu = (float *)malloc(size);
    float *T_gpu = (float *)malloc(size);

    srand(time(NULL));
    for (int i = 0; i < N * N; i++) {
        A[i] = rand() % 10;
        B[i] = rand() % 10;
    }

    // CPU computation
    clock_t start_cpu = clock();
```

```
matrixMulCPU(A, B, C_cpu, N);
transposeCPU(A, T_cpu, N);
clock_t end_cpu = clock();
double cpu_time = ((double)(end_cpu - start_cpu)) / CLOCKS_PER_SEC;

// GPU computation
float *d_A, *d_B, *d_C, *d_T;
cudaMalloc((void **)&d_A, size);
cudaMalloc((void **)&d_B, size);
cudaMalloc((void **)&d_C, size);
cudaMalloc((void **)&d_T, size);

cudaMemcpy(d_A, A, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_B, B, size, cudaMemcpyHostToDevice);

dim3 threads(16, 16);
dim3 blocks((N + 15) / 16, (N + 15) / 16);

cudaEvent_t start_gpu, stop_gpu;
cudaEventCreate(&start_gpu);
cudaEventCreate(&stop_gpu);

cudaEventRecord(start_gpu);
matrixMulKernel<<<blocks, threads>>>(d_A, d_B, d_C, N);
transposeKernel<<<blocks, threads>>>(d_A, d_T, N);
cudaEventRecord(stop_gpu);
cudaEventSynchronize(stop_gpu);

float gpu_time = 0;
cudaEventElapsedTime(&gpu_time, start_gpu, stop_gpu);
gpu_time /= 1000.0; // Convert ms to seconds

cudaMemcpy(C_gpu, d_C, size, cudaMemcpyDeviceToHost);
cudaMemcpy(T_gpu, d_T, size, cudaMemcpyDeviceToHost);
```

```
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);
    cudaFree(d_T);

    // Verify results
    int correct = 1;
    for (int i = 0; i < N * N; i++) {
        if (fabs(C_cpu[i] - C_gpu[i]) > 1e-3 || fabs(T_cpu[i] - T_gpu[i]) > 1e-3) {
            correct = 0;
            break;
        }
    }

    printf("\nCPU Execution Time: %.6f sec\n", cpu_time);
    printf("GPU Execution Time: %.6f sec\n", gpu_time);

    if (correct)
        printf("\nResults match between CPU and GPU.\n");
    else
        printf("\nMismatch detected between CPU and GPU results.\n");

    free(A);
    free(B);
    free(C_cpu);
    free(C_gpu);
    free(T_cpu);
    free(T_gpu);

    return 0;
}
```

## OUTPUT

```
(base) PS C:\Users\Karunya\Documents\Sem 7 - LAs\GPA\Assignments> .\matrix_ops.exe
Enter matrix size N (e.g., 3 for 3x3): 3

CPU Execution Time: 0.000000 sec
GPU Execution Time: 0.049499 sec

Results match between CPU and GPU.
(base) PS C:\Users\Karunya\Documents\Sem 7 - LAs\GPA\Assignments> .\matrix_ops.exe
Enter matrix size N (e.g., 3 for 3x3): 512

CPU Execution Time: 0.432000 sec
GPU Execution Time: 0.001305 sec

Results match between CPU and GPU.
(base) PS C:\Users\Karunya\Documents\Sem 7 - LAs\GPA\Assignments> .\matrix_ops.exe
Enter matrix size N (e.g., 3 for 3x3): 1024

CPU Execution Time: 4.051000 sec
GPU Execution Time: 0.006287 sec

Results match between CPU and GPU.
(base) PS C:\Users\Karunya\Documents\Sem 7 - LAs\GPA\Assignments>
```

*Figure 1: Transposition and Multiplication of Matrices of Different Dimensions.*