CS3811 - High Performance Computing and Big Data Lab

Lab 10

> Name: M K Lokesh Kumar

> Registration No.: 2201113026

> Class: Cyber Security(Semester 5)

Experiment 1

Objective

Write a CUDA C kernel that performs an array's sum using shared memory.

Code

Written in CPP.

```
#include <iostream>
#include <cuda.h>

#define BLOCK_SIZE 256

__global__ void arraySumSharedMemory(float *input, float *output, int N) {
    __shared__ float sharedData[BLOCK_SIZE];

    int tid = threadIdx.x;
    int index = blockIdx.x * blockDim.x + tid;

    sharedData[tid] = (index < N) ? input[index] : 0.0f;
    __syncthreads();

    for (int stride = blockDim.x / 2; stride > 0; stride /= 2) {
        if (tid < stride) {
            sharedData[tid] += sharedData[tid + stride];
        }
        __syncthreads();
    }

    if (tid == 0) {
        output[blockIdx.x] = sharedData[0];
    }
}
```

```
int main() {
    int N = 1024;
    float *array = new float[N];
    for (int i = 0; i < N; ++i) {
        array[i] = static_cast<float>(i + 1);
    }

    float *d_input, *d_partialSums;
    int numBlocks = (N + BLOCK_SIZE - 1) / BLOCK_SIZE;
    cudaMalloc(&d_input, N * sizeof(float));
    cudaMalloc(&d_partialSums, numBlocks * sizeof(float));

    cudaMemcpy(d_input, array, N * sizeof(float), cudaMemcpyHostToDevice);

    arraySumSharedMemory<<<numBlocks, BLOCK_SIZE>>>(d_input, d_partialSums,
N);
    cudaDeviceSynchronize();

    float *partialSums = new float[numBlocks];
    cudaMemcpy(partialSums, d_partialSums, numBlocks * sizeof(float),
cudaMemcpyDeviceToHost);

    float totalSum = 0.0f;
    for (int i = 0; i < numBlocks; ++i) {
        totalSum += partialSums[i];
    }

    std::cout << "Total Sum: " << totalSum << std::endl;

    delete[] array;
    delete[] partialSums;
    cudaFree(d_input);
    cudaFree(d_partialSums);

    return 0;
}
```

Output

Total Sum: 524800

Experiment 2

Objective

To launch the matrix multiplication kernel using the matrix transpose kernel, using CUDA.

Code

Written in CPP.

```
#include <iostream>
#include <cuda.h>

#define TILE_SIZE 32

__global__ void matrixTransposeSharedMemory(float *input, float *output,
int N) {
    __shared__ float tile[TILE_SIZE][TILE_SIZE + 1];

    int x = blockIdx.x * TILE_SIZE + threadIdx.x;
    int y = blockIdx.y * TILE_SIZE + threadIdx.y;

    if (x < N && y < N) {
        tile[threadIdx.y][threadIdx.x] = input[y * N + x];
    }
    __syncthreads();

    x = blockIdx.y * TILE_SIZE + threadIdx.x;
    y = blockIdx.x * TILE_SIZE + threadIdx.y;

    if (x < N && y < N) {
        output[y * N + x] = tile[threadIdx.x][threadIdx.y];
    }
}

int main() {
    int N = 4;
    float *matrix = new float[N * N];
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            matrix[i * N + j] = static_cast<float>(i * N + j + 1);
        }
    }

    std::cout << "Original Matrix:" << std::endl;
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            std::cout << matrix[i * N + j] << " ";
        }
        std::cout << std::endl;
    }

    float *d_input, *d_output;
    cudaMalloc(&d_input, N * N * sizeof(float));
    cudaMalloc(&d_output, N * N * sizeof(float));

    cudaMemcpy(d_input, matrix, N * N * sizeof(float),
cudaMemcpyHostToDevice);

    dim3 blockSize(TILE_SIZE, TILE_SIZE);
    dim3 gridSize((N + TILE_SIZE - 1) / TILE_SIZE, (N + TILE_SIZE - 1) /
TILE_SIZE);
```

```
    matrixTransposeSharedMemory<<<gridSize, blockSize>>>(d_input, d_output,
N);
    cudaDeviceSynchronize();

    float *transposedMatrix = new float[N * N];
    cudaMemcpy(transposedMatrix, d_output, N * N * sizeof(float),
cudaMemcpyDeviceToHost);

    std::cout << "Transposed Matrix:" << std::endl;
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            std::cout << transposedMatrix[i * N + j] << " ";
        }
        std::cout << std::endl;
    }

    delete[] matrix;
    delete[] transposedMatrix;
    cudaFree(d_input);
    cudaFree(d_output);

    return 0;
}
```

Output

Original Matrix: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 Transposed Matrix: 1 5 9 13 2 6 10 14 3 7 11 15 4 8 12 16