CS3811 - High Performance Computing and Big Data Lab

Lab 9

> Name: M K Lokesh Kumar

> Registration No.: 2201113026

> Class: Cyber Security(Semester 5)

Experiment 1

Objective

To launch the image brightening kernel using CUDA.

Code

Written in CPP.

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

using namespace std;

__global__ void brightKernel(float *d_Pin, float *d_Pout, int n, int m) {
    int Row = blockIdx.y * blockDim.y + threadIdx.y;
    int Col = blockIdx.x * blockDim.x + threadIdx.x;

    if ((Row < m) && (Col < n)) {
        d_Pout[Row * n + Col] = 2 * d_Pin[Row * n + Col];
    }
}

int main() {
    int n = 1024;
    int m = 1024;

    // Allocate host memory
    size_t size = n * m * sizeof(float);
    float *h_Pin = (float *)malloc(size);
    float *h_Pout = (float *)malloc(size);

    for (int i = 0; i < n * m; ++i) {
        h_Pin[i] = static_cast<float>(1);
    }
```

```
    float *d_Pin, *d_Pout;
    cudaMalloc(&d_Pin, size);
    cudaMalloc(&d_Pout, size);

    cudaMemcpy(d_Pin, h_Pin, size, cudaMemcpyHostToDevice);

    dim3 blockDim(8, 4);
    dim3 gridDim((n + blockDim.x - 1) / blockDim.x, (m + blockDim.y - 1) /
blockDim.y);

    brightKernel<<<gridDim, blockDim>>>(d_Pin, d_Pout, n, m);

    cudaMemcpy(h_Pout, d_Pout, size, cudaMemcpyDeviceToHost);

    cout << "First 10 elements of the output array:\n";
    for (int i = 0; i < 10; ++i) {
        cout << h_Pout[i] << " ";
    }
    cout << "\n";

    bool success = true;
    for (int i = 0; i < n * m; ++i) {
        if (h_Pout[i] != 2 * h_Pin[i]) {
            success = false;
            cout << "Mismatch at index " << i << "\n";
            break;
        }
    }

    if (!success) {
        std::cout << "Kernel executed failed.\n";
    }

    cudaFree(d_Pin);
    cudaFree(d_Pout);

    free(h_Pin);
    free(h_Pout);

    return 0;
}
```

Output

First 10 elements of the output array: 2 2 2 2 2 2 2 2 2 2

Experiment 1

Objective

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

Code

Written in CPP.

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

#define TILE_DIM 8
#define BLOCK_ROWS 4

using namespace std;

__global__ void MatrixMulKernel(float *d_M, float *d_N, float *d_P, int
Width) {
    int Row = blockIdx.y * blockDim.y + threadIdx.y;
    int Col = blockIdx.x * blockDim.x + threadIdx.x;

    if ((Row < Width) && (Col < Width)) {
        float Pvalue = 0;
        for (int k = 0; k < Width; ++k) {
            Pvalue += d_M[Row * Width + k] * d_N[k * Width + Col];
        }
        d_P[Row * Width + Col] = Pvalue;
    }
}

__global__ void transposeMatrix(float *At, const float *A, int Width) {
    int x = blockIdx.x * TILE_DIM + threadIdx.x;
    int y = blockIdx.y * TILE_DIM + threadIdx.y;
    int width = gridDim.x * TILE_DIM;

    for (int j = 0; j < TILE_DIM; j += BLOCK_ROWS) {
        At[x * width + (y + j)] = A[(y + j) * width + x];
    }
}

int main() {
    int Width = 1024;

    size_t size = Width * Width * sizeof(float);
    float *h_M = (float *)malloc(size);
    float *h_N = (float *)malloc(size);
    float *h_P = (float *)malloc(size);
    float *h_At = (float *)malloc(size);

    for (int i = 0; i < Width * Width; ++i) {
        h_M[i] = 1.0f;
        h_N[i] = 2.0f;
    }
```

```
    float *d_M, *d_N, *d_P, *d_At;
    cudaMalloc(&d_M, size);
    cudaMalloc(&d_N, size);
    cudaMalloc(&d_P, size);
    cudaMalloc(&d_At, size);

    cudaMemcpy(d_M, h_M, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_N, h_N, size, cudaMemcpyHostToDevice);

    dim3 blockDim(8, 4);  // 8 * 4 = 32 threads per block
    dim3 gridDim((Width + blockDim.x - 1) / blockDim.x,
                 (Width + blockDim.y - 1) / blockDim.y);

    MatrixMulKernel<<<gridDim, blockDim>>>(d_M, d_N, d_P, Width);

    dim3 transposeBlockDim(TILE_DIM, BLOCK_ROWS);  // 8 * 4 = 32 threads
    dim3 transposeGridDim((Width + TILE_DIM - 1) / TILE_DIM,
                          (Width + TILE_DIM - 1) / TILE_DIM);

    transposeMatrix<<<transposeGridDim, transposeBlockDim>>>(d_At, d_M,
Width);

    cudaMemcpy(h_P, d_P, size, cudaMemcpyDeviceToHost);
    cudaMemcpy(h_At, d_At, size, cudaMemcpyDeviceToHost);

    cout << "First 10 elements of the result matrix (multiplication):\n";
    for (int i = 0; i < 10; ++i) {
        cout << h_P[i] << " ";
    }
    cout << "\n";

    cout << "First 10 elements of the transposed matrix:\n";
    for (int i = 0; i < 10; ++i) {
        cout << h_At[i] << " ";
    }
    cout << "\n";

    cudaFree(d_M);
    cudaFree(d_N);
    cudaFree(d_P);
    cudaFree(d_At);

    free(h_M);
    free(h_N);
    free(h_P);
    free(h_At);

    return 0;
}
```

Output

First 10 elements of the result matrix (Multiplication): 2048 2048 2048 2048 2048 2048 2048 2048 2048 2048 First 10 elements of the transposed matrix: 1 1 1 1 1 1 1 1 1 1