Министерство цифрового развития, связи
и массовых коммуникаций Российской Федерации

Сибирский государственный университет
телекоммуникаций и информатики

Кафедра прикладной математики и кибернетики

ЛАБОРАТОРНАЯ РАБОТА №4

По дисциплине: «Программирование графических процессоров»

Выполнили:

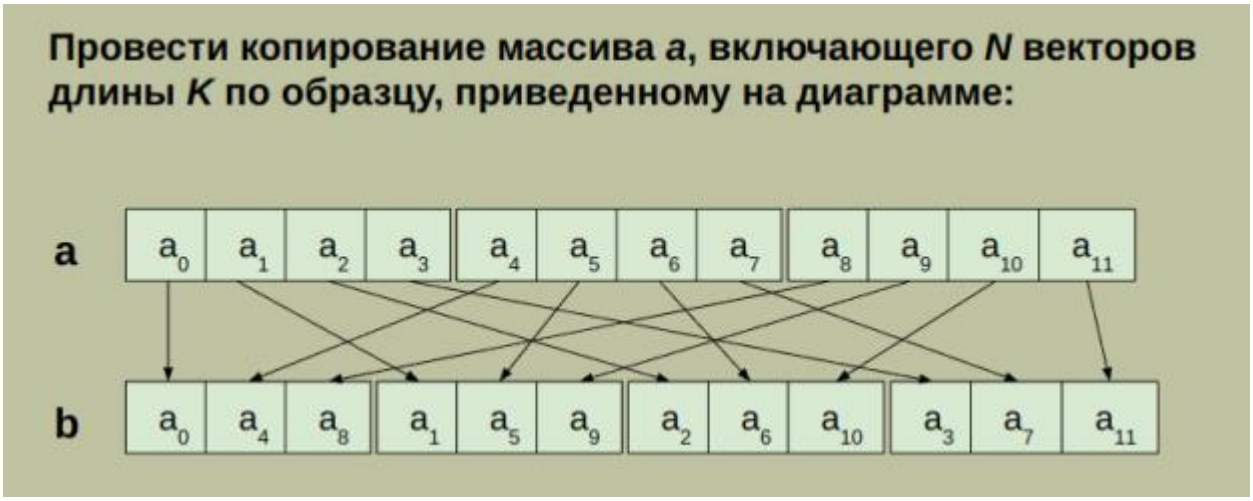
Студенты 3 курса группы ИП-111
Корнилов А.А.,
Попов М.И.,
Толкач А.А.

Проверил:

Профессор кафедры ПМиК
Малков Е.А.

Новосибирск, 2024

Задание:



Цель: априорное понимание совместного доступа к глобальной памяти.

Выполнение работы:

Для первого задания была написана для транспонирования матрицы по потому как было данной в лекции, матрицу используем заданной N*K (4*8), в вызове функции gTransposition мы берем размерность сетки [2,1,1], размерность самого блока [4,4,1] соответствуя количеству нитей на поток.

```
#include <cuda.h>
#include <iostream>
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

using namespace std;

__global__ void gTransposition(int *a, int *b, int N, int K) {
    unsigned int k = threadIdx.x + blockIdx.x * blockDim.x;
    unsigned int n = threadIdx.y + blockIdx.y * blockDim.y;
    b[n + k * N] = a[k + n * K];
}

int main() {
    const int N = 4, K = 8, threads_per_block = 4;
    float elapsedTime;
    int *GPU_pre_matrix, *local_pre_matrix, *GPU_after_matrix, *local_after_matrix;
    cudaEvent_t start, stop;
    cudaEventCreate(&start);
    cudaEventCreate(&stop);

    cudaMalloc((void **) &GPU_pre_matrix, N * K * sizeof(int));
    cudaMalloc((void **) &GPU_after_matrix, N * K * sizeof(int));
    local_pre_matrix = (int *) calloc(N * K, sizeof(int));
    local_after_matrix = (int *) calloc(N * K, sizeof(int));
```

```
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < K; ++j) {
            local_pre_matrix[j + i * K] = j + i * K + 1;
            cout << local_pre_matrix[j + i * K] << " ";
        }
        cout<< endl;
    }

    cudaMemcpy(GPU_pre_matrix, local_pre_matrix, K * N * sizeof(int),
cudaMemcpyHostToDevice);
    cudaEventRecord(start, nullptr);

    gTransposition <<< dim3((K + threads_per_block - 1) / threads_per_block,
                            (N + threads_per_block - 1) / threads_per_block),
                       dim3(threads_per_block,
threads_per_block)>>>(GPU_pre_matrix, GPU_after_matrix, N, K);

    cudaEventRecord(stop, nullptr);
    cudaEventSynchronize(stop);
    cudaDeviceSynchronize();
    cudaMemcpy(local_after_matrix, GPU_after_matrix, K * N * sizeof(float),
cudaMemcpyDeviceToHost);

    cout << endl;
    for (long long i = 0; i < K; ++i) {
        for (long long j = 0; j < N; ++j) {
            cout << local_after_matrix[j + i * N] << " ";
        }
        cout << endl;
    }

    cudaEventElapsedTime(&elapsedTime, start, stop);

    cout << "CUDA Event time:\n\t"
         << elapsedTime
         << endl;

    cudaFree(GPU_pre_matrix);
    cudaFree(GPU_after_matrix);
    free(local_pre_matrix);
    free(local_after_matrix);

    return 0;
}
```

Листинг 1 – программа LR03_1.cu

Команда компиляции и результат работы программы:

```
D:\Projects\CUDA_CMake\cmake-build-debug\LR04_GPU.exe
1 2 3 4 5 6 7 8
```

```
9 10 11 12 13 14 15 16
17 18 19 20 21 22 23 24
25 26 27 28 29 30 31 32

1 9 17 25
2 10 18 26
3 11 19 27
4 12 20 28
5 13 21 29
6 14 22 30
7 15 23 31
8 16 24 32
CUDA Event time:
        0.725152

Process finished with exit code 0
```