

Министерство цифрового развития, связи
и массовых коммуникаций Российской Федерации

Сибирский государственный университет
телекоммуникаций и информатики

Кафедра прикладной математики и кибернетики

ЛАБОРАТОРНАЯ РАБОТА №5

По дисциплине: «Программирование графических процессоров»

Выполнили:

Студенты 3 курса группы ИП-211

Оганесян А.С.

Лацук А.Ю.

Проверил:

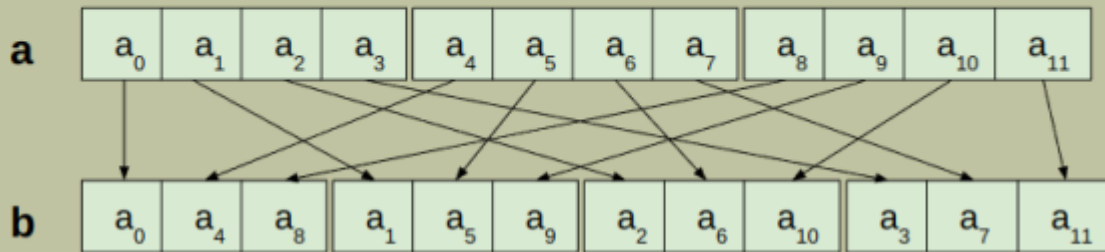
Профессор кафедры ПМиК

Малков Е.А.

Новосибирск, 2025

Задание:

Провести копирование массива a , включающего N векторов длины K по образцу, приведенному на диаграмме:



Цель: априорное понимание совместного доступа к глобальной памяти.

Выполнение работы:

Была написана программа для транспонирования матрицы по примеру из лекций. Матрицу используем заданной $N \times K$, где $N = 4 \times 2^{12}$ и $K = 8 \times 2^{12}$, в вызове функции `gTransposition` мы берем размерность сетки, и `threads_per_block = 128` нити на поток.

Функция `gTransposition` выполняет копирование матрицы с массива a в массив b с использованием глобальной памяти.

```
#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 num = 1 << 12;
    int N = 4 * num, K = 8 * num, threads_per_block = 128;
    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;
        }
    }

    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);

    cudaDeviceSynchronize();

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

    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 main.cu

Команда компиляции и результат работы программы:

```
D:\Projects\CUDA_CMake\cmake-build-debug\LR04_GPU.exe  
CUDA Event time:  
    0.5776  
  
Process finished with exit code 0
```

Вывод:

В ходе выполнения лабораторной работы, была исследована и применена работа с глобальной памятью графического процессора (GPU) с использованием технологии CUDA. В ходе работы мы ознакомились с работой с глобальной памятью и её синхронизацией.