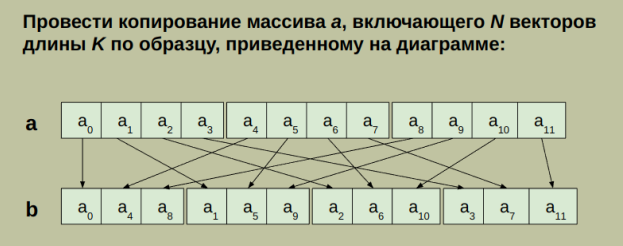


**Задание:**



**Цель:** априорное понимание совместного доступа к глобальной

памяти.

**Выполнение работы:**

Была написана программа для транспонирования матрицы по примеру из лекций. Матрицу используем заданной N\*K, где N = 4\*2^12 и K = 8\*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 использование 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.