

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ  
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)  
Институт №8 «Компьютерные науки и прикладная математика»  
Кафедра 806 «Вычислительная математика и программирование»

**Лабораторная работа №4**  
**по курсу «Программирование графических процессоров»**  
**Работа с матрицами. Метод Гаусса.**

Выполнил: М.С.Гаврилов  
Группа: 8О-406Б  
Преподаватели: К.Г. Крашенинников,  
А.Ю. Морозов

Москва, 2022

## Условие

**Цель работы.** Использование объединения запросов к глобальной памяти. Реализация метода Гаусса с выбором главного элемента по столбцу. Ознакомление с библиотекой алгоритмов для параллельных расчетов Thrust. Использование *двухмерной сетки потоков*. Исследование производительности программы с помощью утилиты nvprof

Вариант 6.

Нахождение ранга матрицы.

## Программное и аппаратное обеспечение

Характеристики графического процессора	
Compute capability:	7.5
Name:	NVIDIA GeForce GTX 1650
Total Global Memory:	4102029312
Shared memory per block:	49152
Registers per block:	65536
Warp size:	32
Max threads per block:	(1024, 1024, 64)
Max block:	(2147483647, 65535, 65535)
Total constant memory:	65536
Multiprocessors count:	14

Процессор: Intel(R) Core(TM) i5-11260H @ 2.60GHz

Оперативная память: 7875 Мб

Накопитель: kimtigo SSD 256GB

OS: Linux Mint 21

IDE: Visual Studio Code 1.72.0

compiler: Cuda compilation tools, release 11.8, V11.8.89

## Метод решения

Для хранения и обработки матриц используется класс `matrix`. Этот класс хранит как указатель на массив в памяти хоста, содержащий элементы матрицы, так и указатель на массив в глобальной памяти. Для передачи данных между массивами используются функции синхронизации. Матрица хранится по столбцам. Отдельно реализованы методы поиска максимального элемента в столбце (с помощью `thrust`) и перестановки строк. Для перестановки строк нужно отдельное ядро с одномерной сеткой потоков. Поиск ранга выполняется в цикле, на каждом шаге которого происходит отыскание главного элемента в активном столбце, перестановка столбцов и запуск основного ядра. Если в столбце все элементы равны нулю, то он пропускается и активным становится следующий столбец. Основное ядро выполняет шаг алгоритма Гаусса на всей активной части матрицы. Для того

чтобы поле выполнения всех шагов определить, занулился ли последний столбец, используется еще одно ядро, так как копирование данных в память хоста занимает слишком много времени.

## Описание программы

Программа состоит из одного файла, в котором реализован класс `matrix` и ядра. Копирование данных из памяти хоста на мультипроцессор осуществляется единожды, перед началом определения ранга. Метод поиска максимального элемента требует одного обратного копирования для того чтобы понять, не является ли максимальный по модулю элемент нулем. Также обратное копирование требуется при определении того, занулена ли последняя строка, тоже лишь одно: равенство нулю проверяется на мультипроцессоре, а в глобальную память помещается булев флаг.

## Класс `array`

Имеет восемь членов – размеры матрицы, ссылки на глобальную память и память хоста, где хранятся элементы, три переменные, нужные для поиска максимального элемента, чтобы не инициализировать их на каждом шаге, и компаратор, также инициализированный единожды для экономии времени.

Метод	Описание
<code>matrix(int n_, int m_)</code>	Конструктор, что считывает элементы со stdin
<code>void update_host_matrix()</code>	Синхронизатор
<code>void update_device_matrix()</code>	Синхронизатор
<code>void print()</code>	Выводит элементы матрицы на stdout.
<code>void find_max_elm(int array_start, int array_size)</code>	Находит максимальный по модулю элемент. Помещает его индекс в соответствующую переменную – член, либо устанавливает ее значение на -1 если элемент равен 0.
<code>void swap_rows(int lhs, int rhs)</code>	Переставляет строки матрицы.
<code>int rank()</code>	Находит ранг матрицы.

## Ядра

<code>__global__ void kernel_swap_rows(double* elements, int row_1, int row_2, int n, int m)</code>
<code>__global__ void kernel_gaussian_step(double* elements, int n, int m, int start_row_index, int active_colomn)</code>
<code>__global__ void kernel_row_is_zero(double* elements, int n, int m, int row, int start_colomn, bool* res)</code>

Первое ядро осуществляет перестановку строк. Соответствующие элементы из row\_1 меняются местами с элементами из row\_2.

Второе ядро осуществляет шаг метода Гаусса. Предназначено для запуска на двумерной сетке потоков. Коэффициент вычисляется для каждой строки лишь на первой итерации сетки по матрице. Изменения не применяются к начальному столбцу, так как он просто зануляется и больше не будет использован.

Третье ядро осуществляет проверку участка строки матрицы на равенство всех элементов нулю. Каждый поток, столкнувшись с ненулевым элементом изменяет значение флага на true.

## Результаты

### 1. Сравнение времени работы ядра с различными конфигурациями.

Размер матрицы 1000 x 500 элементов.

Размерность ядра	Время работы (мс)
<<<(1,1),(1,1)>>>	25368.1
<<<(4,4),(4,4)>>>	676.537
<<<(8,8),(8,8)>>>	496.168
<<<(16,16),(16,16)>>>	534.477
<<<(32,32),(32,32)>>>	356.082

Размер файла 2000 x 10000 элементов.

Размерность ядра (основного)	Время работы (мс)
<<<(8,8),(8,8)>>>	12501.7
<<<(16,16),(16,16)>>>	6646.13
<<<(32,32),(32,32)>>>	6214.9

### 2. Сравнение времени работы CPU и ядра (конфигурация <<64 64>>)

Размер теста	Время на CPU (мс)	Время на GPU (мс)
1 000 x 1 000 pix	1274	356.082

### 3. Примеры работы программы

Входные данные	3 4 0 1 2 2 0 3 6 4 0 5 10 1
----------------	---------------------------------------



```

thrust::null_type, thrust::null_type, thrust::null_type, thrust::null_type, thrust::null_type, thrust::null_type>,
thrust::cuda_cub::tag, thrust::use_default, thrust::use_default>>, __int64>(__int64, thrust::null_type)
0.36% 1.0955ms 500 2.1900us 2.1120us 5.4090us void
thrust::cuda_cub::core::kernel_agent<thrust::cuda_cub::__parallel_for::ParallelForAgent<thrust::cuda_cub::f
or_each_f<thrust::pointer<thrust::tuple<double, __int64, thrust::null_type, thrust::null_type, thrust::null_type,
thrust::null_type, thrust::null_type, thrust::null_type, thrust::null_type, thrust::null_type>,
thrust::cuda_cub::tag, thrust::use_default, thrust::use_default>,
thrust::detail::wrapped_function<thrust::detail::allocator_traits_detail::gozer, void>>, __int64>,
thrust::cuda_cub::for_each_f<thrust::pointer<thrust::tuple<double, __int64, thrust::null_type,
thrust::null_type, thrust::null_type, thrust::null_type, thrust::null_type, thrust::null_type, thrust::null_type>,
thrust::cuda_cub::tag, thrust::use_default, thrust::use_default>, thrust::use_default>,
thrust::detail::wrapped_function<thrust::detail::allocator_traits_detail::gozer, void>>, __int64>(__int64,
thrust::null_type)

```

```

API calls: 37.95% 744.17ms 1001 743.43us 10.755us 604.32ms cudaMalloc
21.33% 418.41ms 1500 278.94us 50.845us 1.3185ms cudaEventSynchronize
10.18% 199.55ms 2000 99.774us 26.889us 631.16us cudaStreamSynchronize
6.67% 130.87ms 1001 130.74us 9.2880us 1.7380ms cudaFree
4.48% 87.822ms 2500 35.128us 20.533us 549.02us cudaLaunchKernel
4.16% 81.661ms 18003 4.5350us 2.4440us 2.0964ms cudaGetLastError
3.77% 73.901ms 1500 49.267us 33.733us 1.0135ms cudaEventElapsedTime
3.29% 64.573ms 501 128.89us 82.623us 1.2648ms cudaMemcpy
2.20% 43.219ms 500 86.438us 61.600us 375.96us cudaMemcpyAsync
1.79% 35.131ms 1 35.131ms 35.131ms 35.131ms cuDevicePrimaryCtxRelease
1.49% 29.190ms 3000 9.7300us 5.8660us 736.76us cudaEventRecord
1.24% 24.246ms 4501 5.3860us 2.9330us 562.22us cudaGetDevice
0.70% 13.782ms 3000 4.5940us 2.4440us 184.80us cudaPeekAtLastError
0.70% 13.753ms 2500 5.5010us 2.9330us 519.20us cudaDeviceGetAttribute
0.02% 358.84us 101 3.5520us 2.4440us 8.8000us cuDeviceGetAttribute
0.01% 289.42us 1 289.42us 289.42us 289.42us cuModuleUnload
0.01% 122.22us 2 61.111us 29.822us 92.400us cudaEventCreate
0.00% 23.466us 1 23.466us 23.466us 23.466us cuDeviceGetUuid
0.00% 18.577us 3 6.1920us 2.9330us 11.733us cuDeviceGetCount
0.00% 17.111us 1 17.111us 17.111us 17.111us cudaFuncGetAttributes
0.00% 9.2880us 2 4.6440us 3.4220us 5.8660us cuDeviceGet
0.00% 6.8450us 1 6.8450us 6.8450us 6.8450us cuDeviceGetName
0.00% 5.8660us 1 5.8660us 5.8660us 5.8660us cuModuleGetLoadingMode
0.00% 3.9110us 1 3.9110us 3.9110us 3.9110us cuDeviceTotalMem
0.00% 3.9110us 1 3.9110us 3.9110us 3.9110us cudaGetDeviceCount
0.00% 2.9330us 1 2.9330us 2.9330us 2.9330us cuDeviceGetLuid

```

## Вывод

В ходе выполнения этой лабораторной работы я ознакомился с методами распараллеливания метод Гаусса и реализовал алгоритм отыскания ранга матрицы с использованием GPU. Я узнал о возможности объединения запросов к глобальной памяти и на практике увидел, насколько это улучшает производительность. Также я получил опыт работы с функциями библиотеки thrust.

Основной трудностью при выполнении работы была необходимость добиться очень большой скорости работы программы, что потребовало больших усилий в оптимизации.

Версия на мультипроцессоре, оказалась быстрее, чем на CPU, уменьшение затраченного времени при увеличении размерности сетки наблюдалось.