МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

**Лабораторная работа №4**

**по курсу «Программирование графических процессоров»**

**Работа с матрицами. Метод Гаусса.**

Выполнил: А. О. Тояков

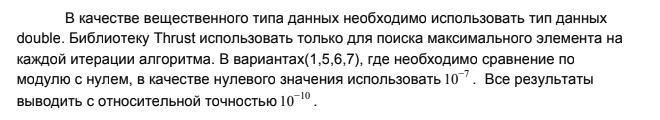
Группа: М8О-407Б-18

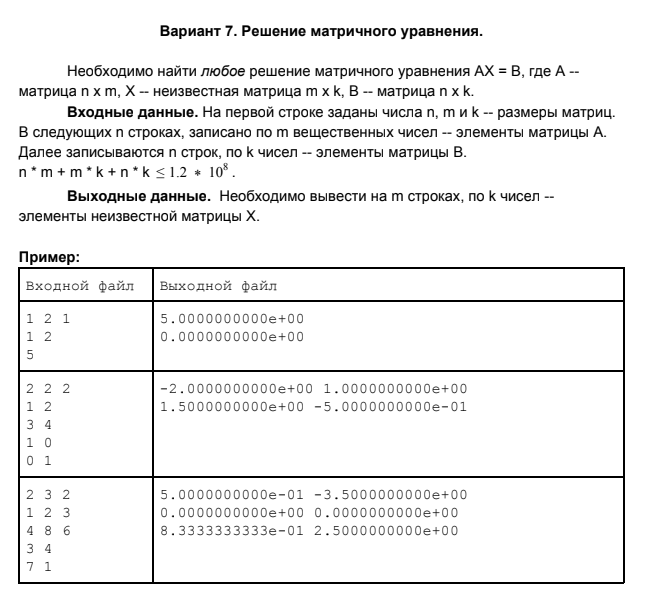
Преподаватели: К. Г. Крашенинников,

А. Ю. Морозов

# 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.





# Программное и аппаратное обеспечение

Device: GeForce MX250

Размер глобальной памяти: 3150381056

Размер константной памяти : 65536

Размер разделяемой памяти: 49152

Регистров на блок: 32768

Максимум потоков на блок: 1024

Количество мультипроцессоров : 3

OS: Linux Ubuntu 18.04

Редактор: VSCode

Компилятор: nvcc версии 11.4 (g++ версии 7.5.0)

# Метод решения

Для реализации метода Гаусса решения матричного уравнения необходимо найти максимальный элемент в столбце (это реализовано с помощью функции библиотеки thrust и написанного компаратора). Затем нужно строку с этим элементом переместить наверх.

После чего мы делаем параллельный проход методом Гаусса вниз, обнуляя соответствующие элементы под главным, и запоминаем индекс ступеньки. После приведения матрицы к треугольному виду необходимо совершить проход вверх, чтобы матрица А стала единичной, а в матрице B столбцы станут решением. Затем копируем данные с GPU на CPU и выводим результат таким образом, что на месте ступенек в искомой матрице X появятся столбцы матрицы B, а строки, индексы которых не соответствуют индексам ступенек занулятся.

# Описание программы

Макрос **CSC**отвечает за отслеживание ошибок в функциях cuda, поэтому все cuda-вызовы оборачиваются в него и при cudaError\_t != cudaSuccess выводится сообщение об ошибке.

struct comparator – перегрузка оператора () для вычисления максимального элемента в столбце с помощью библиотеки thrust.

fill\_AB – отвечает за заполнение матрицы AB элементами по столбцам.

swap – функция на GPU, в которой строки меняются местами.

down\_pass – функция на GPU, которая совершает один проход методом Гаусса вниз.

up\_pass – функция на GPU, которая совершает один проход методом Гаусса вверх.

int main() ­– отвечает за ввод, и перенос данных на GPU и вывод.

# тесты производительности

В обоих программах числа для массивов генерировались рандомно в промежутке [-1000, 1000].

Работа на GPU:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Тест:  (размеры матрицы) | Результат: | | | | | |
| down\_pass<<<dim3(8,16), dim3(8,16)>>>  up\_pass<<<dim3(8,16), dim3(8,16)>>> | down\_pass<<<dim3(8,32), dim3(8,32)>>>  up\_pass<<<dim3(8,32), dim3(8,32)>>> | down\_pass<<<dim3(16,32), dim3(16,32)>>>  up\_pass<<<dim3(16,32), dim3(16,32)>>> | | down\_pass<<<dim3(32,32), dim3(32,32)>>>  up\_pass<<<dim3(32,32), dim3(32,32)>>> | |
| 10 \* 10 \* 10 | time = 0.542720 | time = 0.648192 | | time = 0.916480 | | time = 2.220032 |
| 100 \* 100 \* 100 | time = 5.253792 | time = 5.637184 | | time = 9.588672 | | time = 22.995457 |
| 1000 \* 1000 \* 1000 | time = 851.362183 | time = 859.228455 | | time = 950.938354 | | time = 1378.070435 |

Работа на CPU:

|  |  |
| --- | --- |
| Тест: | Результат: |
| 10 \* 10 \* 10 | time = 9.234577 |
| 100 \* 100 \* 100 | time = 215.25452 |
| 1000 \* 1000 \* 1000 | time = 12235.547828 |

# Выводы

После выполнения лабораторной работы № 4 я узнал для себя несколько новых аспектов параллельного программирования.

Во-первых, метод Гаусса имеет сложность o(n^3), и как известно, чем сложнее метод, тем проще его распараллелить, поэтому я считаю это преимуществом этого алгоритма.

Во-вторых, я познакомился с библиотекой thrust, которая позволяет экономить время и не писать фундаментальные алгоритмы (например поиск максимального элемента или scan).

В-третьих, я познакомился с одним из способов оптимизации при работе с глобальной памятью, а именно с объединением запросов к глобальной памяти. Это значит, что GPU умеет объединять ряд запросов к глобальной памяти в транзакцию одного сегмента, причем длина сегмента должна быть 32/64/128 байт и сегмент должен быть выровнен по своему размеру. Таким образом мы получим увеличение скорости работы с памятью на порядок.