МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Факультет «Информационные технологии и прикладная математика»
Кафедра «Вычислительная математика и программирование»

# Лабораторная работа №1
# по курсу «Программирование графических процессоров»

# Освоение программного обеспечения для работы с технологией CUDA. Примитивные операции над векторами.

Студент: Лысенко Д.А.
Группа: 8О-408Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2019

## Условие

1. *Цель работы:* ознакомление и установка программного обеспечения для работы с программно-аппаратной архитектурой параллельных вычислений(CUDA). Реализация одной из примитивных операций над векторами. В качестве вещественного типа данных необходимо использовать тип данных double. Все результаты выводить с относительной точностью 10-10. Ограничение: n < 225.

*Вариант 3. Поэлементное умножение векторов.*

*Входные данные. На первой строке задано число n -- размер векторов. В*

*следующих 2-х строках, записано по n вещественных чисел -- элементы векторов.*

*Выходные данные. Необходимо вывести n чисел -- результат поэлементного*

*умножения исходных векторов.*

## Программное и аппаратное обеспечение

Compute capability: 5.0
Name: GeForce GTX 960M
Total Global Memory: -2147483648
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: 5
Процессор (CPU) – Intel Core i5-6300HQ 2.30GHz; 8 ГБ RAM;
OS Microsoft Windows 10 (x64)
CUDA V10.1

## Метод решения

Каждый элемент результирующего вектора получается в результате перемножения соответствующих элементов исходных векторов. При запуске ядра каждая нить в соответствии со своим глобальным индексом находит результат перемножения исходных векторов и записывает в результат.

## Описание программы

Ядро программы, где происходят вычисления

```
__global__ void multiply(double* dev_A, double* dev_B, size_t arrLen) {
   size_t index = threadIdx.x + blockIdx.x * blockDim.x;
   while (index < arrLen) {
      dev_A[index] = dev_A[index] * dev_B[index];
      index += blockDim.x * gridDim.x;
   }
}
```

## Оценка производительности

Размеры тестов: 10000, 100000, 1000000, 10000000

| Test | CUDA <1,32> TIME | CUDA <32,32> TIME | CUDA <256,256> TIME | CUDA <1024,1024> TIME | CPU TIME |
|---|---|---|---|---|---|
| 1 | 0.095 | 0.012 | 0.014 | 0.204 | 0.000 |
| 2 | 1.816 | 0.149 | 0.122 | 0.303 | 0.000 |
| 3 | 18.101 | 1.415 | 1.233 | 1.2 | 0.000 |
| 4 | 180.940 | 14.153 | 12.247 | 12.122 | 46.875 |

## Вывод

В данной лабораторной работе я ознакомился с основами работы с программно-аппаратной архитектурой параллельных вычислений CUDA. Основной время при выполнении лабораторной я потратил на изучение основ работы с памятью устройства и принципов исполнения кода ядра. Сама задача поэлементного умножения является тривиальной. В процессе тестирования выяснилось, что запуск ядер на больших конфигурациях не всегда дает прирост производительности, это связано с тем, что большинство нитей ядра на небольших данных просто простаивают.