МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

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

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

Выполнил: *Чистяков К.С.*
Группа: *8О-406Б-21*
Преподаватель: А.Ю. Морозов

Москва, 2025

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.
2. **Вариант задания.** Вариант 5. Поэлементное нахождение максимума 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 нахождения максимума исходных векторов.

Программное и аппаратное обеспечение

Графический процессор (Google Colab)

Compute capability	7.5
Name	Tesla T4
Total Global Memory	15828320256
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	6553
Multiprocessors count	40

Процессор AMD Ryzen 5 4500U (2.38 GHz)

Technology	7 nm
Cores	6
Threads	6
Core Speed	1390 MHz
Cache (L1 Data)	6 x 32 KBytes (8-way)

Cache (L1 Inst.)	6 x 32 KBytes (8-way)
Cache (Level 2)	6 x 512 KBytes (8-way)
Cache (Level 3)	2 x 4 MBytes (16-way)

Оперативная память

Number of modules	2
Module Manuf.	Samsung
Module Size	8 GBytes (total - 16 GBytes)
Generation	DDR4

Жесткий диск - 512 ГБ SSD
OS - Windows 10 Домашняя + Linux
IDE - VS Code + Google Colab
Compiler - nvcc, gcc

Метод решения

Общее описание алгоритма: динамически выделяем память для двух масивов, куда положим исходные данные, и ещё одного массива для результата. Потом попарно сравниваем элементы первых двух массивов, находя из пары максимум, и записываем его в результирующий массив. Далее выводим результат в стандарный поток вывода.

Описание программы

Один файл, включающий три функции: CSC - для отлова ошибок; kernel - для работы ядра (тут непосредственно и происходят поэлементное нахождение максимума веторов) и main - тут мы инициализируем переменные и массивы, считываем данные, заполняем массивы, передаем их в GPU, вызываем функцию ядра, возвращаем результирующий массив, выводим данные, очищаем память.

Ядро:

```
__global__ void kernel(double *vec1, double *vec2, double *result, int n) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    int offset = gridDim.x * blockDim.x;
    while (idx < n) {
        result[idx] = vec1[idx] > vec2[idx] ? vec1[idx] : vec2[idx];
        idx += offset;
    }
}
```

Передаём в него два массива, которые содержат исходные данные, результирующий массив и размер массивов. Потом происходит попарное нахождение максимума двух исходных массивов и заполняется результирующий массив.

Результаты

Замеры времени работы ядер с различными конфигурациями и размерами массивов n (n = 100, n = 100000, n = 10000000). А также приведён замер работы аналогичной программы без использования технологии CUDA - на CPU.

1. **Конфигурация <<< 1, 32 >>>**

n	Время (в мс)
100	0.103680
100000	1.828256
10000000	226.164795

2. **Конфигурация <<< 128, 64 >>>**

n	Время (в мс)
100	0.108832
100000	0.114560
10000000	1.452000

3. **Конфигурация <<< 512, 512 >>>**

n	Время (в мс)
100	0.112512
100000	0.068000
10000000	1.147040

4. **Конфигурация <<< 1024, 1024 >>>**

n	Время (в мс)
100	0.110816

100000	0.079360
10000000	1.188576

5. **CPU**

n	Время (в мс)
100	0
100000	0,739
10000000	86,93

Выводы

В ходе выполнения лабораторной работы я ознакомился с программно-аппаратной архитектурой параллельных вычислений (CUDA), а также реализовал один из примитивных операций над векторами - поэлементное нахождение максимума векторов.

Этот алгоритм может применяться в области компьютерного зрения и обработки изображений (слияние изображений, фильтрация изображений), в области машинного обучения и анализа данных (агрегация данных, обучение нейросетей), в области финансового анализа (оценка рисков, оптимизация портфеля) и т.д.

Программировать было несложно, просто непривычно, так как до этого я с применением CUDA не сталкивался. Основными сложностями были: 1) Ошибка несоответствия версий куда. Решил это компилируя код на старой версии. 2) Ошибка Time limit exceeded at test 15.t. Решил поменяв тип данных размера массивов с int на long int.

Сравнивая время работы ядер с различными конфигурациями и размерами массивов, а также аналогичной программы без использования технологии CUDA - на CPU, можно заметить, что лучше всего с применением технологии CUDA справляются конфигурации <<< 512, 512 >>> и <<< 1024, 1024 >>>, конфигурация <<< 1, 32 >>> справляется также хорошо на маленьком тесте, но заметно уступает на больших. Выполнение тестов на CPU показало, что алгоритм работает так намного быстрее на малом и среднем тестах, но сильно уступает конфигурациям <<< 512, 512 >>> и <<< 1024, 1024 >>> на большом тесте (но всё же быстрее конфигурации <<< 1, 32 >>>). Из этого сравнения можно сделать вывод, что при правильном выборе конфигурации можно значительно увеличить производительность алгоритма на больших данных.