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: Семин А. В.
Группа: 8О-406Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2023

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: $n < 2^{25}$.
2. **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)

Название	Tesla T4
Вычислительные мощности	7.5
Видеопамять	15835 MB
Общая память на блок	49152 bytes
Регистр на блок	65536 bytes
Постоянная память	65536 bytes
Макс. количество потоков на мультипроцессор	2048
Макс. количество потоков на блок	1024
Мултипроцессоры	40

Процессор AMD Ryzen 7 4800H

Ядра	*8*
Потоки	16
Частота	2.9 GHz
Макс. Частота	4.2 GHz
Кэш 1-го уровня	64KB (на ядро)
Кэш 2-го уровня	512KB (на ядро)
Кэш 3-го уровня	8 MB (общий)
Техпроцесс	7 nm
Критическая температура	105 градусов

16 Гб оперативной памяти и 512 ГБ SSD.
OS – Windows 11Домашняя, IDE – VS Code, Compiler - nvcc, gpp.

Метод решения

Алгоритм решения очень примитивный: динамически выделяем память для двух массивов, в которые положим исходные данные, а затем в рамках вычислений ядра выполняем попарное сравнение соответствующих элементов в данных массивах и запоминаем из каждой пары максимальный элемент. Выводим результат в стандартный поток вывода. Какой-либо сложной и нестандартной архитектуры программа не имеет.

Описание программы

Лабораторная работа состоит из одного файла, включающего функцию работы ядра и main. В главной функции выполняются процессы инициализации массивов, считывания и вывода данных, передачи этих данных между процессором и ядром, вызова метода решения, очистки памяти.

Сигнатура функции ядра:

__global__ void findElByElMaximums(double* vec1, double* vec2, double* res, int n).

Ядро параллельно выполняется на нитях графического процессора. На вход, как видно из сигнатуры, подаются два массива, содержащих входные данные, и массив, в которые будут помещены результаты, а также n – размер исходных массивов. Далее ядро выполняет попарное сравнение элементов и запись результата.

Результаты

Время выведено в ***миллисекундах***.

	10^2	10^4	10^6	10^7
CPU	0,0054100	0,01474960	13,7348650	85.3458220
<1, 32>	0.033408	0.220288	23.714657	230.247238
<64, 64>	0.029728	0.030528	0.255680	2.229952
<256, 256>	0.036256	0.026112	0.115840	1.082304
<512, 512>	0.035904	0.044992	0.108128	1.029664
<1024, 1024>	0.038560	0.040800	0.115168	0.997888

Выводы

Данный алгоритм решения задачи может применяться во многих областях, среди которых есть такие, как: обработка изображений и компьютерное зрение, научные вычисления, анализ данных и статистика, машинное обучение.

В ходе выполнения лабораторной работы главной сложностью было разобраться в окружении языка CUDA, его компиляции и запуске, потому что это методы его запуска немного непривычны.

Даже на примере простой задачи, сравнив время работы CPU и GPU с различным количеством потоков, мы выяснили, что GPU лучше справляется с большим объемом данных, так как, очевидно, он обрабатывает данные параллельно. CPU, я считаю, с меньшими объемами данных справился лучше, и выбор между CPU и GPU зависит лишь от выполняемой задачи.