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: *А.Ю. Гришин*
Группа: *8О-408Б*
Преподаватель: А.Ю. Морозов

Москва, 2024

Условие

Кратко описывается задача:

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 8:** реверс вектора.

Программное и аппаратное обеспечение

- Графический процессор
  - Compute capability: 7.5
  - Объем графической памяти: 15,83 ГБ
  - Объем постоянной памяти: 65536 байт
  - Разделяемая память на блок: 49152 байт
  - Количество регистров на блок: 65536
  - Максимальное количество потоков на блок: 1024
  - Количество мультипроцессоров: 40
- Процессор
  - Количество физических ядер: 2
  - Количество логических ядер: 4
  - Частота: 2000 МГц
- Оперативная память
  - Тип оперативной памяти: DDR4
  - Объем: 32 ГБ
- Жесткий диск
  - Объем: 1 ТБ
- Программное обеспечение
  - ОС: Ubuntu 22.04.4 LTS
  - IDE: Visual Studio Code
  - Компилятор: nvcc

Метод решения

Реверс вектора представляет собой смену местами между двумя парными элементами. Иными словами, первый элемент меняется местами с последним, второй – с предпоследним, и т. д.

Обозначим операцию смены местами в виде функции $f(i)$, где i – индекс левого элемента пары. Тогда реверс вектора можно описать как вызова функции $f(i)$ для $i \in [1; \left\lfloor \frac{n}{2} \right\rfloor]$, где n – размер исходного вектора.

Отметим, что подзадачи $f(i)$ не зависят друг от друга и не используют одну и ту же область памяти для работы (каждая подзадача работает со своей парой элементов). А, следовательно, есть основание рассмотреть параллельную версию этого алгоритма.

Описание программы

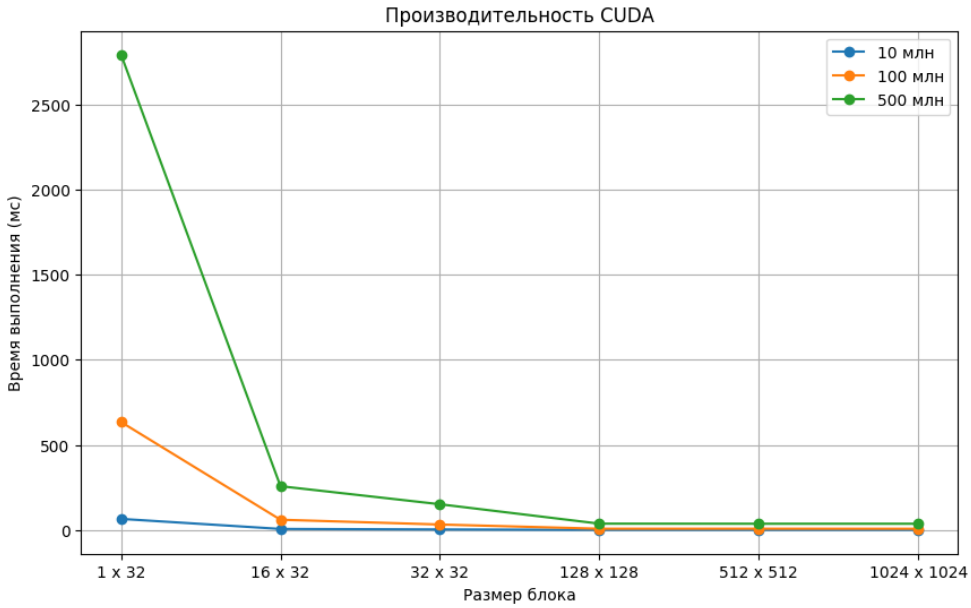
Программа состоит из одного файла, содержащего код, написанный на языке CUDA. Код программы состоит из:

1. Макроса `EXIT_WITH_ERROR`, который отвечает за вывод сообщения об ошибке в `stderr` и экстренный выход из программы с кодом возврата 0;
2. Функции `kernel`, которая содержит в себе всю логику, которая будет выполняться на графическом процессоре. В контексте данной лабораторной работы, функция `kernel` содержит алгоритм по реверсу части массива, за которую отвечает отдельный поток;
3. Функции `main`, которая отвечает за инициализацию ресурсов, с которыми в дальнейшем будет работать функция `kernel`. А также за их освобождение и вывод результата на экран;

Результаты

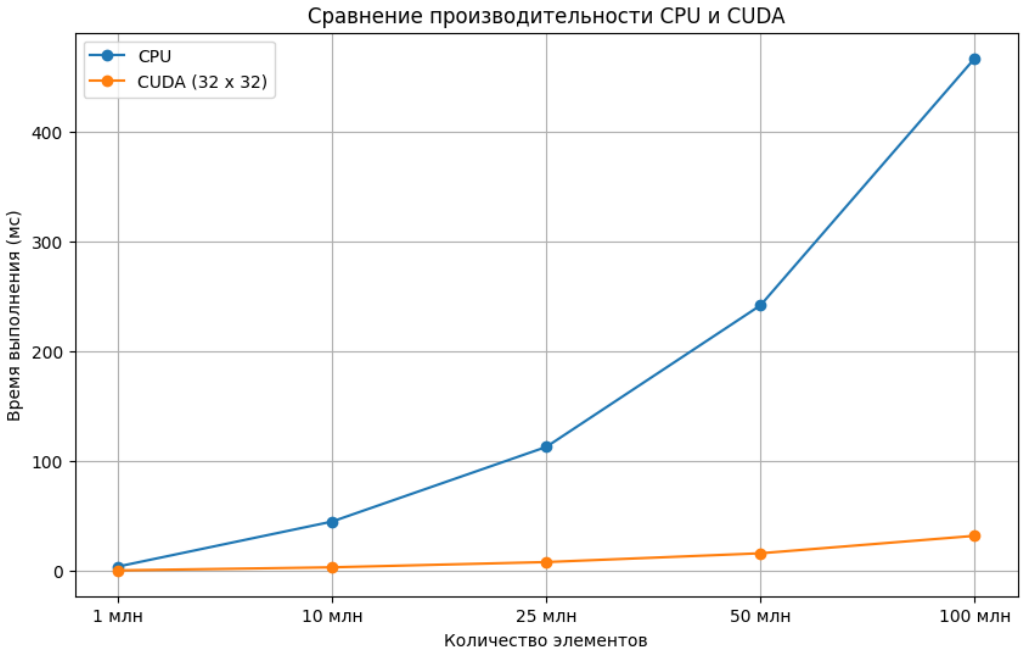
1. Зависимость времени выполнения программы от конфигурации и размера данных. В качестве единиц измерения времени были выбраны миллисекунды (мс).

Конфигурация / Размер вектора	10 млн	100 млн	500 млн
1×32	65	634	2792
16×32	6	60	257
32×32	3	32	151
128×128	0,7	7,33	37
512×512	0,8	7,298	36,5
1024×1024	0,8	7,1	36,5



2. Сравнение программы на CUDA с 32×32 потоками и программы на CPU с одним потоком. В качестве единиц измерения времени были выбраны миллисекунды (мс).

Размер вектора	Время на CUDA (мс)	Время на CPU (мс)
1 млн	0,516	4
10 млн	3,35	45
25 млн	8,13	113
50 млн	16,13	242
100 млн	32,05	467



Выводы

Проделав лабораторную работу, я использовал базовые операции для работы с CUDA и реализовал алгоритм реверса вектора. Данный алгоритм может быть применен на практике при решении задач, связанных с рендерингом изображений, или при реализации алгоритмов, связанных с машинным обучением (в частности, нейронными сетями).

При решении задачи я столкнулся с проблемой, заключающейся в асинхронности выполнения процессов на CPU и GPU, что приводило к некорректным результатам. Данная проблема была решена с помощью функции `cudaDeviceSynchronize()`, которая обеспечивает синхронизацию процессов.

При анализе я заметил, что на GPU программа работает значительно эффективнее, однако временная сложность также оставалась линейной, поскольку на каждом ядре выполнялся линейный алгоритм. Тем не менее, параллельное выполнение на нескольких ядрах позволило значительно сократить общее время выполнения задачи.