МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

**Лабораторная работа №4**

**по курсу «Параллельная обработка данных»**

**Сортировка чисел на GPU. Свертка, сканирование, гистограмма.**

Выполнил: Г.Н. Хренов

Группа: 8О-407Б

Преподаватели: К.Г. Крашенинников,

А.Ю. Морозов

Москва, 2021

**Условие**

1. Цель работы: Ознакомление с фундаментальными алгоритмами GPU: свертка

(reduce), сканирование (blelloch scan) и гистограмма (histogram). Реализация одной из сортировок на CUDA. Использование разделяемой и других видов памяти. Исследование производительности программы с помощью утилиты nvprof.

Вариант 6. Карманная сортировка с битонической сортировкой в каждом кармане.

**Программное и аппаратное обеспечение**

GPU name: NVIDIA GeForce RTX 2060

compute capability 7:5

totalGlobalMem: 6442450944

sharedMemPerBlock: 49152

totalConstMem: 65536

regsPerBlock: 65536

maxThreadsDim: 1024 1024 64

maxGridSize: 2147483647 65535 65535

multiProcessorCount: 30

CPU name: AMD Ryzen 7 3750H with Radeon Vega Mobile Gfx

MaxClockSpeed: 2300

NumberOfCourse: 4

RAM: 8

SSD: 256, HDD: 1024

OS: Windows10

Compiler: nvcc

**Метод решения**

Карманная сортировка содержит все фундаментальные алгоритмы, поэтому необходимо реализовать свертку max min, гистограмму и скан, а также битоническую сортировку для кармана. После этого находим min max с помощью редукции, затем разбиваем элементы массива на малые карманы, присваивая каждому номер кармана. Далее по этим номерам выполняем сортировку подсчетом(гистограмма-скан-подсчет). Когда массив упорядочен по малым карманам, объединяем малые карманы в большие и запускаем для каждого битоническую сортировку. Если малые карманы оказались слишком велики, для них рекурсивно выполняется карманная сортировка с самого начала.

**Описание программы**

lab5.cu:

\_\_global\_\_ void kernel\_reduce\_max(float\* data, int n, float\* out) – свертка, нахождение максимального элемента

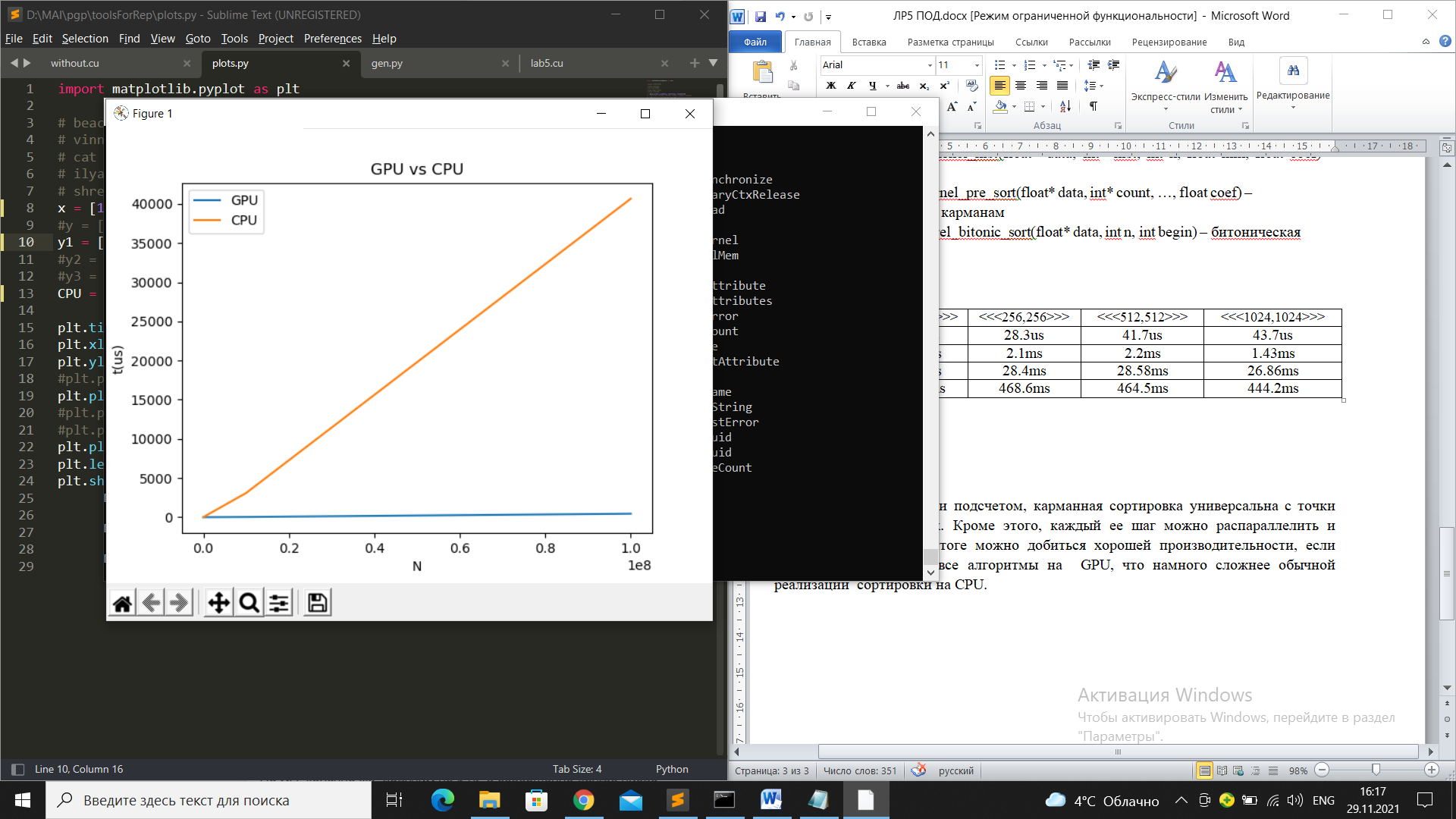
\_\_global\_\_ void kernel\_hist(float\* data, int\* hist, int n, float min, float coef) – гистограмма

\_\_global\_\_ void kernel\_pre\_sort(float\* data, int\* count, …, float coef) – упорядочение массива по карманам

\_\_global\_\_ void kerel\_bitonic\_sort(float\* data, int n, int begin) – битоническая сортировка

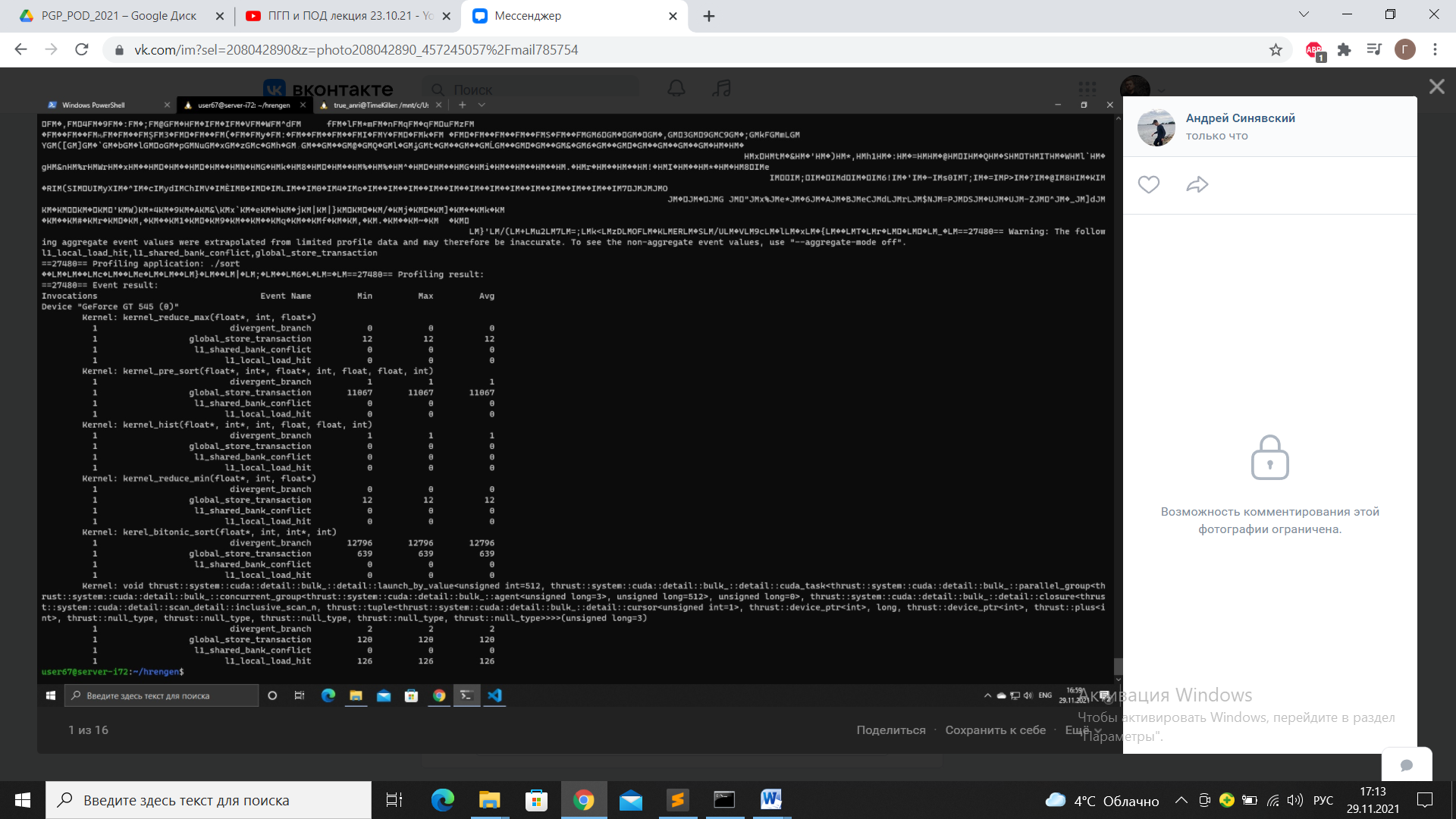
**Результаты**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | <<<64,64>>> | <<<256,256>>> | <<<512,512>>> | <<<1024,1024>>> |
| 100 | 27.1us | 28.3us | 41.7us | 43.7us |
| 1000000 | 3.93ms | 2.1ms | 2.2ms | 1.43ms |
| 10000000 | 42.2ms | 28.4ms | 28.58ms | 26.86ms |
| 100000000 | 544.2ms | 468.6ms | 464.5ms | 444.2ms |



Профилировка

На всех этапах алгоритма реализация не содержит конфликтов банков памяти. В реализации битонической сортировки есть дивергенция потоков, однако это вынужденая мера, так как на длину сортируемого массива накладывются ограничения. Реализация без дивергенции возможна, но потребует дополнительных расходов по памяти и скорее всего нецесообразна.



**Выводы**

В отличие от сортировки подсчетом, карманная сортировка универсальна с точки зрения входных данных. Кроме этого, каждый ее шаг можно распараллелить и запускать на GPU. В итоге можно добиться хорошей производительности, если корректно реализовать все алгоритмы на GPU, что намного сложнее обычной реализации сортировки на CPU.