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 числе на GPU. Свертка, сканирование, гистограмма.

Выполнил: А. О. Тояков
Группа: М8О-407Б-18
Преподаватели: К. Г. Крашенинников,
А. Ю. Морозов

Москва, 2022

УСЛОВИЕ

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.

Все входные-выходные данные являются бинарными и считываются из **stdin** и выводятся в **stdout**.

Входные данные. В первых четырех байтах записывается целое число n -- длина массива чисел, далее следуют n чисел типа заданного вариантом.

Выходные данные. В бинарном виде записывают n отсортированных по возрастанию чисел.

Пример входных-выходных данных. Десять чисел типа int, от 0 до 9.

```
Входной файл (stdin), hex:
0A000000 00000000 09000000 08000000 07000000 06000000 05000000
04000000 03000000 02000000 01000000

Выходной файл (stdout), hex:
00000000 01000000 02000000 03000000 04000000 05000000 06000000
07000000 08000000 09000000
```

Вариант на “два”. Сортировка подсчетом.

Вариант №2, с использованием алгоритма сканирования из библиотеки Thrust.

ПРОГРАММНОЕ И АППАРАТНОЕ ОБЕСПЕЧЕНИЕ

Device: GeForce MX250

Размер глобальной памяти: 3150381056

Размер константной памяти : 65536

Размер разделяемой памяти: 49152

Регистров на блок: 32768

Максимум потоков на блок: 1024

Количество мультипроцессоров : 3

OS: Linux Ubuntu 18.04

Редактор: VSCode

Компилятор: nvcc версии 11.4 (g++ версии 7.5.0)

МЕТОД РЕШЕНИЯ

Сортировка подсчётом содержит 2 фундаментальных алгоритма – гистограмму и сканирование. Последний, согласно варианту, можно было использовать из библиотеки thrust, поэтому я реализовал только гистограмму для больших чисел с использованием глобальной памяти. Вначале мы находим максимальный элемент в исходном массиве, затем создаём массив гистограммы размера [0, max] и заполняем его нулями. После применяем алгоритм гистограммы и на полученном массиве используем включающее сканирование. Затем остаётся лишь записать данные в результирующий массив по формуле, используя атомарную операцию сложения.

ОПИСАНИЕ ПРОГРАММЫ

Макрос **CSC** отвечает за отслеживание ошибок в функциях cuda, поэтому все cuda-вызовы оборачиваются в него и при cudaError_t != cudaSuccess выводится сообщение об ошибке.

__global__ void histogram() – алгоритм гистограммы с использованием глобальной памяти.

__global__ void out() – записать результата на GPU в результирующий массив.

void counting_sort() – сортировка подсчётом.

int maximum() – функция нахождения максимума в массиве.

int main() – отвечает за ввод, и перенос данных на GPU и вывод.

ТЕСТЫ ПРОИЗВОДИТЕЛЬНОСТИ

В обоих программах числа для массивов генерировались рандомно в промежутке [0, 1000].

Работа на GPU:

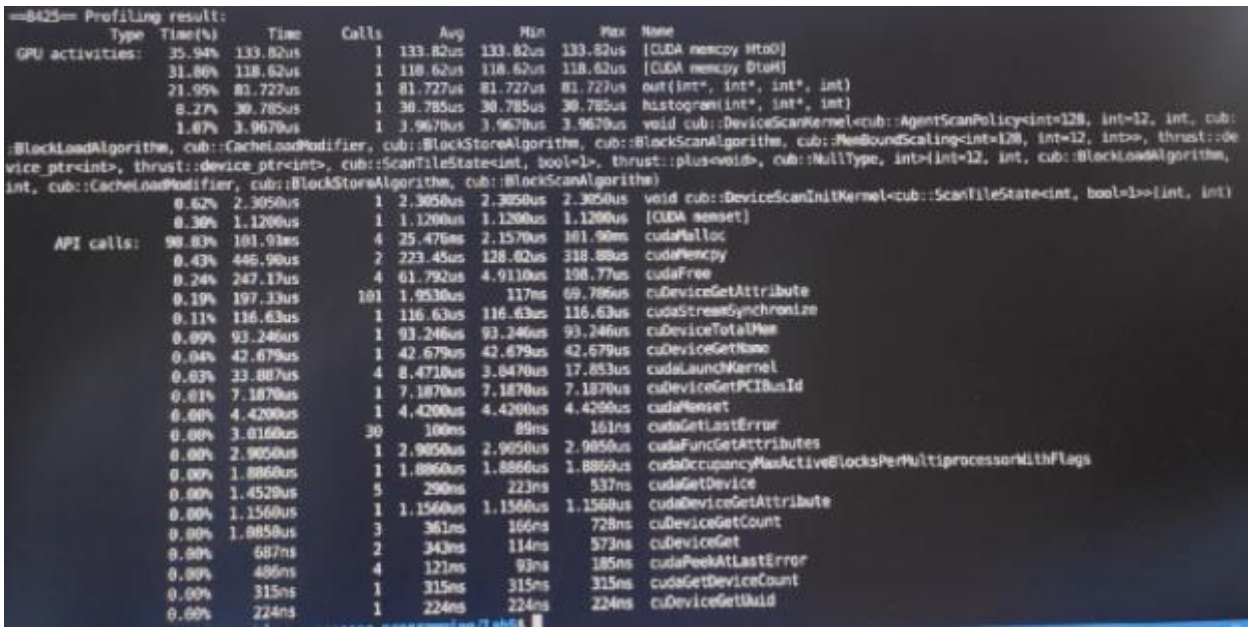
Тест: (размер массива)	Результат:			
	<<<64, 64>>>	<<<256, 256>>>	<<<512, 512>>>	<<<1024, 1024>>>
1000	time = 0.042272	time = 0.045408	time = 0.052128	time = 0.108704
10000	time = 0.048704	time = 0.058880	time = 0.055680	time = 0.119840
100000	time = 0.125824	time =0.120960	time = 0.135936	time = 0.177216

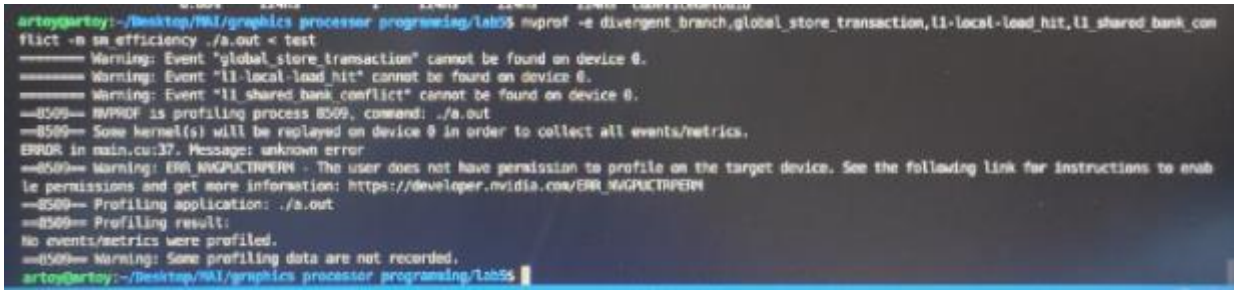
Работа на CPU:

Тест:	Результат:
1000	time = 0.065
10000	time = 0.391
100000	time = 1.174

ПРОФИЛИРОВКА

По запросу nvprof –query-metrics нужные метрики не были найдены на моём устройстве. Соответственно профилировщик сообщил об этом при попытке посчитать требуемые метрики.





Несмотря на то, что на девайсе не было метрик, я думаю, что ошибок в моей программе быть не должно, т. к. на всех этапах алгоритма используются либо атомарные операции, либо функции из библиотеки thrust, а разделяемая память вообще не применялась.

ВЫВОДЫ

Сортировку подсчётом легко распараллелить, так как все этапы алгоритма можно выполнить на GPU, используя фундаментальные алгоритмы и средства технологии CUDA. По результатам тестов производительности видно, что программа на девайсе работает в разы быстрее программы на хосте. Однако, данная сортировка не самая универсальная. Требуется, чтобы входные данные были целочисленными, а также необходимо находить максимальный элемент, что занимает дополнительное время работы, но в благоприятных условиях программа отрабатывает за O(n), что хорошо.