МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»

Лабораторная работа № 3
по курсу «Программирование графических процессоров»

Сортировка чисел на GPU. Свертка, сканирование, гистограмма

Выполнил: И.В. Сектименко
Группа: М8О-410Б-22
Преподаватели: А.Ю. Морозов,
Е.Е. Заяц

Москва, 2025

Условие

Кратко описывается задача:

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*.
2. Вариант 3. Сортировка подсчетом. Диапазон чисел от 0 до 255.
   Требуется реализовать сортировку подсчетом для чисел типа uchar. Должны быть реализованы:
   — алгоритм гистограммы, с использованием атомарных операций и разделяемой памяти;
   — алгоритм сканирования, с бесконфликтным использованием разделяемой памяти.
   Входные данные. На первой строке задано число n ($n \leq 537\ 10^6$) – количество чисел, которые необходимо отсортировать. Далее сами числа.
   Выходные данные. Необходимо вывести n чисел в порядке возрастания.

Программное и аппаратное обеспечение

Характеристики:

— графического процессора:

compute capability:	7.5
графическая память:	15828320256
разделяемая память на блок:	49152
константная память:	65536
количество регистров на блок:	65536
максимальное количество блоков:	(2147483647, 65535, 65535)
максимальное количество нитей на блок:	(1024, 1024, 64)
количество мультипроцессоров:	40

— процессора: Intel(R) Xeon(R) CPU @ 2.00GHz;

— оперативной памяти: 12Гб общей памяти;

— жесткого диска: 256 Гб общей памяти.

Во время выполнения работы использовалась IDE Google Colab. В нее встроена ОС Ubuntu 22.04.4 LTS с видеокартой NVIDIA-SMI 550.54.15.

Метод решения

Для решения задачи необходимо реализовать алгоритм гистограммы: посчитать сколько чисел каждого типа в исходном массиве. Затем реализовать алгоритм сканирования и применить его к массиву, полученному после применения алгоритма гистограммы: посчитать префиксные суммы. Теперь известны позиция каждого числа в результирующем, отсортированном массиве, осталось только расставить числа по своим позициям: первым индексом проходимся по массиву префиксных сумм, вторым - по результирующему массиву, если значение второго индекса меньше значения массива префиксных сумм по первому индексу, то в результирующем массиве по второму

индексу ставим значение первого индекса, иначе увеличиваем первый индекс на единицу.

Описание программы

Программа состоит из 1 файла.

Сначала считывается вся необходимая информация из стандартного потока ввода в бинарном формате.

Массив чисел перекопируется в глобальную память GPU и запускается ядро *hist*, которое реализует алгоритм гистограммы для массива чисел. Создается массив в разделяемой памяти длинной 255 с нулевыми значениями: именно столько различных значений могут принимать числа массива. Далее необходимо циклом пройтись по всем элементам входного массива и с помощью атомарных операций увеличить на единицу значение в массиве с гистограммой по индексу, который равен значению текущего элемента входного массива. Наконец, с помощью атомарных операций необходимо перекопировать в новый массив в глобальной памяти значения из разделяемой памяти.

После этого запускается ядро *scan*, которое реализует алгоритм сканирования: каждый блок нитей обрабатывает свой блок массива гистограммы (на тот случай, если нитей в блоке меньше, чем длина массива). Сначала перекопируем значения в разделяемую память. Реализуем алгоритм blelloch scan. В массиве sums сохраняются последние значения массива сканирования, полученные в каждом блоке, чтобы в конце приплюсовать разницу к этементам в тех блоках, где она будет нужна. Так же важно избавиться от банковских конфликтов. Это достигается путем хранения следующих элементов не по порядку, а с некоторым сдвигом, чтобы обращение происходило к разным банкам разделяемой памяти. Это, в свою очередь, позволит ускорить процесс выполнения алгоритма.

Теперь следует последняя фаза алгоритма, которая реализована в ядре *kernel*: на основе массива с префиксными суммами расставить значения исходного массива по нужным местам, чтобы в итоге получился отсортированный массив. Алгоритм описан в методе решения.

Отсортированный массив перекопируется в память на CPU и выводится в стандартный поток вывода в бинарном формате.

Результаты

С помощью утилиты *nvprof* профилируем каждое ядро по отдельности на данных разного объема.

Ниже приведена сравнительная таблица для ядра *hist*. В строках – разные конфигурации ядра, в столбцах – разные объемы данных, то есть размер массива.

	1 000	10 000	100 000
<<<1, 32>>>	20,512 мкс	155,36 мкс	1,504 мс
<<<64, 64>>>	3,776 мкс	4,736 мкс	15,552 мкс
<<<256, 256>>	4,832 мкс	4,768 мкс	6,112 мкс
<<<1024, 1024>>>	40,384 мкс	40,416 мкс	41,696 мкс

Ниже приведена сравнительная таблица для ядра *scan*. В строках – разные конфигурации ядра, в столбцах – разные объемы данных, то есть размер массива.

	1 000	10 000	100 000
<<<1, 256>>>	7,296 мкс	7,392 мкс	7,328 мкс
<<<16, 16>>>	7,136 мкс	6,816 мкс	6,815 мкс
<<<4, 256>>>	8,544 мкс	8,223 мкс	8,159 мкс
<<<64, 64>>>	14,016 мкс	13,344 мкс	13,343 мкс
<<<256, 256>>	85,695 мкс	85,952 мкс	84,511 мкс

Ниже приведена сравнительная таблица для ядра *kernel*. В строках – разные конфигурации ядра, в столбцах – разные объемы данных, то есть размер массива.

	1 000	10 000	100 000
<<<1, 32>>>	10,176 мкс	129,31 мкс	1,2352 мс
<<<64, 64>>>	4,192 мкс	123,87 мкс	1,2257 мс
<<<256, 256>>	3,744 мкс	123,1 мкс	1,2259 мс
<<<1024, 1024>>>	26,048 мкс	123,26 мкс	1,226 мс

Сравним наилучший результат работы на GPU с результатом на CPU.

	1 000	10 000	100 000
GPU	15 мкс	134 мкс	461 мкс
CPU	15 мкс	86 мкс	864 мкс

Профилируем программу с помощью утилиты *ncu* на данных размером n = 100 000 с конфигурациями ядер, которые дали лучший результат по времени.

	hist	scan	kernel
Shared Memory Configuration Size	32,77 Кб	32,77 Кб	32,77 Кб
Driver Shared Memory Per Block	0 б	0 б	0 б
Dynamic Shared Memory Per Block	0 б	68 б	0 б
Static Shared Memory Per Block	1,02 Кб	0 б	0 б

Выводы

Сортировка – важный алгоритм, который может быть частью чего-то большего. Например, если известно, что нужная информация находится по некоторому числовому ключу, то этот числовой ключ можно достаточно быстро найти в отсортированном массиве с помощью бинарного поиска за логарифм. В то же время в неотсортированном массиве поиск осуществится за линию.

Сортировка подсчетом – алгоритм, который позволяет отсортировать данные за линию, и это считается достаточно быстро.

Было достаточно тяжело понять, как распараллеливаются алгоритмы гистограммы и сканирования, зато после понимания, было достаточно легко их запрограммировать.

Так как данное задание была на работу с разделяемой памятью, с помощью утилиты ncu я решила посмотреть, сколько же разделяемой памяти было занято. Ядро *kernel* вообще

не работает с разделяемой памятью, поэтому там очевидно все по нулям. В ядре *scan* выделяемся память путем передачи аргумента в конфигурации и зависит от выделяемых потоков: логически важно, чтобы каждый поток обрабатывал одну ячейку разделяемой памяти. В ядре *hist* стабильно выделяется на этапе компиляции 1,02 Кб. Кроме того, утилита ncu имеет и другие метрики, которые позволяют оптимизировать и отлаживать код.