МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Информационные технологии и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа № 1
по курсу «Программирование графических процессоров»

Сортировка чисел на GPU. Свертка, сканирование, гистограмма.

Выполнил: А.В. Скворцов
Группа: 8О-408Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2020

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.

Вариант 6: Карманная сортировка с битонической сортировкой в каждом кармане. Требуется реализовать карманную сортировку для чисел типа float. Должны быть реализованы:

- Алгоритм гистограммы, с использованием атомарных операций.
- Алгоритм свертки для любого размера, с использованием разделяемой памяти.
- Алгоритм сканирования для любого размера, с использованием разделяемой памяти. (Можно воспользоваться библиотекой Thrust)
- Алгоритм битонической сортировки для карманов.

Программное и аппаратное обеспечение

- GPU: Geforce 940MX
  - Compute capability : 5.0
  - Total Global Memory : 2147483648
  - Shared memory per block : 49152
  - Registers per block : 65536
  - Max threads per block : (1024, 1024, 64)
  - Max block : (2147483647, 65535, 65535)
  - Total constant memory : 65536
  - Multiprocessors count : 3
- CPU: Intel Core i5-6200U 2.30GHz
- RAM: 4GB
- Software: Windows 10, Visual Studio Code, nvcc

Метод решения

Карманная сортировка — сортировка за линейное время, основанная на предположении о равномерном распределении данных. Ее идея достаточно проста, мы создаем конечное число отдельных блоков, количество которых пропорционально размеру исходного массива, так, чтобы все элементы в каждом следующем по порядку блоке были всегда больше, чем в предыдущем. Каждый блок затем сортируется отдельно. Разбить это можно на следующие шаги:

1. Поиск максимального и минимального элементов массива (алгоритм свертки) для последующего нормирования элементов и вычисления их номеров карманов.
2. Подсчет количества элементов в каждом кармане (тут нам пригодится алгоритм гистограммы)

3. Вычисление индекса начала каждого блока в результирующем массиве (применяем алгоритм сканирования к гистограмме распределения элементов)
4. Сортировка каждого из карманов. Если размер кармана больше некоторого заданного числа, повторяем алгоритм рекурсивно, иначе применяем сортировку эффективную на малых размерах массивов (в данном варианте битоническая сортировка)

Битоническая сортировка основана на сортировке битонных последовательностей (которая сначала монотонно не убывает, а затем монотонно не возрастает, либо приводится к такому виду в результате циклического сдвига). Такие битонные последовательности можно легко и быстро сделать или просто возрастающими, или наоборот просто убывающим. Тогда отсортировав две битонные последовательности, мы получаем новую объединенную битонную последовательность, которую опять можем отсортировать.

Описание программы

Программа состоит из одного файла и состоит из следующих функций:

1. void bucket_sort(float *arr, uint32_t len) – собственно сама сортировка, управляет памятью для гпу и вызывает более низкоуровневую recursive_bucket_sort.
2. void recursive_bucket_sort – собственно сама сортировка, выполняет ее непосредственные шаги, т. е. создает сами бакеты и сортирует их.
3. __global__ void bucket_bitonic_sort — битоническая сортировка одного кармана. Если размер кармана больше заданной константы, сортировка не производится и делается запись о том, что данный карман нужно сортировать отдельно рекурсивно.
4. __global__ void histogram_kernel — считает гистограму распределения элементов по карманам.
5. __global__ void prepare_buckets — распределяет элементы по карманам в соответствии с гистограммой
6. float reduce(float *arr, uint32_t len, int fun_type, float *buff = NULL) — алгоритм свертки, обертка над кудой.
7. __global__ void reduce_kernel — ядро свертки.
8. __device__ float atIndex — возвращает элемент массива по умолчанию, если индекс массива находится за его пределами.

Результаты

В измерение времени работы на гпу также учитываются все аллокации и копирования.

Кол-во элементов	10^4	10^5	10^6	10^7	10^8
Bucket sort gpu	141 604	146 078	150 857	228 193	2 558 138
Thrust::sort	1179	1635	6470	45 443	1 577 686
Bucket sort cpu	2147	26 616	325 456	3 903 623	42 039 108
Std::sort	2003	24 149	298 312	3 442 403	39 076 645

Видно, что гпу сортировки на порядок быстрее аналогичных на цпу.

Зависимость скорости работы от количества кластеров:

Кол-во кластеров	1	4	8	16	32
Время	2 558 138	4 966 152	5 370 598	3 860 703	3 193 345

Выводы

Карманная сортировка — один из алгоритмов сортировки за линейное время, который требует наличия апостериорных знаний о природе распределения элементов. Так при большом количестве мало отличимых элементов, распределенным по нескольким кластерам, скорость алгоритма заметно деградирует и использовать сортировку становится непрактично. Из достоинств можно отметить возможность эффективного распараллеливания, что увеличивает ее производительность на порядок. Для сортировки карманов можно использовать, например, специальную параллельную битоническую сортировку, которую, зная особенности куды, можно эффективно применять для массивов помещающихся в разделяемой памяти.