МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

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

Лабораторная работа №5
по курсу «Программирование графических процессоров»

Сортировка чисел на GPU. Свертка, сканирование, гистограмма.

Выполнил: Полей-Добронравова Амелия
Группа: 8О-407Б
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 (*обязательно отразить в отчете*).

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 3. Сортировка подсчетом. Диапазон 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.

Ограничения: $n \leq 537 * 10^6$

Пример:

Входной файл (stdin), hex	Выходной файл (stdout), hex
0A 00 00 00 01 02 03 01 02 03 01 02 03 04	01 01 01 02 02 02 03 03 03 04

Программное и аппаратное обеспечение

Компилятор nvcc версии 7.0(g++ версии 4.8.4) на 64-х битной Ubuntu 14.04 LTS.
Параметры графического процессора:
Compute capability : 6.1
Name : GeForce GTX 1050

Total Global Memory : 2096103424
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 : 5

Метод решения

Распараллеленная сортировка подсчетом состоит из трех этапов: гистограмма, скан, финальное преобразование.
Гистограмма подсчитывает количество встречаемости элементов в массиве. Есть требование использовать разделяемую память: на каждом блоке должна быть своя локальная гистограмма, и потом нужно суммировать их в глобальную.
Скан позволяет вычислять префиксную сумму. Благодаря тому, что диапазон чисел 256 элементов, можно разработать алгоритм скан для одного блока. Его я реализовала по аналогии с алгоритмом, представленным на сайте NVIDIA.
Финальное преобразование формирует выходной отсортированный массив.

Описание программы

Макрос **CSC** - макрос для отслеживания ошибок со стороны GPU, вызывается около функций для cuda и выводит текст ошибки при cudaError_t не равным cudaSuccess.
what_pow(int n) - две идентичные функции для работы на CPU и GPU для нахождения степени двойки для получения аргумента функции.
no_conflict_offset(int n, int banks) - функция GPU для определения сдвига обращения к массиву для неконфликтного обращения к разделяемой памяти.
hist(unsigned char* array, int n, int* out) - функция GPU гистограммы.
prescan(int* d_out, const int* d_in, int blocks) - функция GPU для скана на одном блоке.
kernel(int* pref, unsigned char* out) - функция GPU для финального формирования массива.
main() - ввод-вывод информации, запуск этапов.

Результаты

Конфигурация	Тест размера 135	Тест размера 13500	Тест размера 1350000	Тест размера 135000000
На CPU	1304.9ms	1292.96ms	1371.34ms	5273.9ms
1,32	4745.96ms	4694.1ms	4732.38ms	6422.49ms
32, 32	233.266ms	211.726ms	208.347ms	433.712ms
64,64	119.49ms	110.913ms	105.054ms	301.328ms
256,256	182.261ms	166.191ms	161.451ms	360.342ms
512,512	315.951ms	293.621ms	290.686ms	484.519ms
1024, 1024	576.247ms	587.141ms	568.434ms	765.19ms

Исследование производительности программы с помощью утилиты nvprof

```
user73@server-i72:~/homework$ nvprof ./lab5 < 12.t
==32195== NVPROF is profiling process 32195, command: ./lab5
==32195== Profiling application: ./lab5
==32195== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 82.39%  59.742ms         1  59.742ms  59.742ms  59.742ms  kernel(int*, unsigned char*)
 17.08%  12.381ms         1  12.381ms  12.381ms  12.381ms  hist(unsigned char*, int, int*)
  0.23%  164.94us         3  54.978us  3.4250us  157.90us  [CUDA memcpy DtoH]
  0.21%  154.25us         2  77.123us     672ns  153.58us  [CUDA memcpy HtoD]
  0.07%  47.874us         2  23.937us  3.5520us  44.322us  [CUDA memset]
  0.02%  17.384us         1  17.384us  17.384us  17.384us  prescan(int*, int const *, int)

==32195== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 55.69%  72.727ms         5  14.545ms  31.556us  60.087ms  cudaMemcpy
 43.87%  57.295ms         4  14.324ms  6.1360us  57.040ms  cudaMalloc
  0.28%  363.78us        83  4.3820us     150ns  154.46us  cuDeviceGetAttribute
  0.06%  83.041us         1  83.041us  83.041us  83.041us  cuDeviceTotalMem
  0.04%  48.421us         1  48.421us  48.421us  48.421us  cuDeviceGetName
  0.03%  43.046us         3  14.348us  10.991us  19.371us  cudaLaunch
  0.02%  27.873us         2  13.936us  13.561us  14.312us  cudaMemset
  0.01%  6.9320us         8     866ns     163ns  4.9250us  cudaSetupArgument
  0.00%  2.2800us         3     760ns     478ns  1.3060us  cudaConfigureCall
  0.00%  2.0010us         2  1.0000us     493ns  1.5080us  cuDeviceGetCount
  0.00%     623ns         2     311ns     211ns     412ns  cuDeviceGet
```

82% от всей программы занимает kernel - финальное преобразование массива, расставляющее элементы в выходной массив. Связано это с тем, что у меня расставление одного значения происходит последовательно, а в данном тесте несколько тысяч элементов только двух значений, из-за чего только два потока расставляют их.

Код программы для CPU:

```
#include
<iostream>
                #include <cstring>


                const int LWORD = 65;
                const int LKEY = 7;


                struct TElement {
                   char index[LKEY] = {};
                   unsigned long long key = 0;
                   char word[LWORD] = {};
                };


                class TVector {
                   size_t vSize;
                   size_t capacity;
                   TElement* data;
                public:
                   TVector() {
                       vSize = 0;
                       capacity = 0;
                       data = new TElement[2]();
                   }
                   TVector(size_t s) {
                       vSize = 0;
                       capacity = s;
                       data = new TElement[s]();
                   }
                   TVector(size_t s, TElement* d) {
                       vSize = s;
                       capacity = s;
                       data = new TElement[s]();
                       if (s != 0) {
                           for (size_t i = 0; i < s; i++) {
                               data[i] = d[i];
                           }
                       }
                   }
```

```
    TElement& operator[] (size_t i) {
        return data[i];
    }
    size_t Size() {
        return vSize;
    }

    size_t Capacity() {
        return capacity;
    }
    void Resize() {
        if (capacity == 0) {
            capacity = 1;
        }
        capacity *= 2;
        TElement* temp = new TElement[capacity]();
         if(temp == 0x0 ) {
             exit(-1);
         }
        for (size_t i = 0; i < vSize; i++) {
            temp[i] = data[i];
        }
        delete [] data;
        data = temp;
    }
    void PushBack(TElement elem) {
        if (vSize == capacity) {
            Resize();
        }
        data[vSize++] = elem;
    }
    ~TVector() {
        delete [] data;
    }
};

void CountingSort(TVector &v) {
    TVector temp(v.Size());
```

```
    unsigned long long max = v[0].key;
    int i;
    for (size_t j = 1; j < v.Size(); j++) {
        if (v[j].key > max) {
            max = v[j].key;
        }
    }
    max++;
    TVector c((size_t)max);
    for (size_t j = 0; j < c.Capacity(); j++) {
        c[j].key = 0;
    }
    for (size_t j = 0; j < v.Size(); j++) {
        ++c[v[j].key].key;
    }
    for (size_t j = 1; j < c.Capacity(); j++) {
        c[j].key = c[j].key + c[j - 1].key;
    }
    for (int j = v.Size() - 1; j >= 0; j--) {
        c[v[j].key].key--;
        temp[c[v[j].key].key] = v[j];
    }
    for (size_t m = 0; m < v.Size(); m++) {
        v[m] = temp[m];
    }
}


int main(void) {
    std::ios::sync_with_stdio(false);

    TVector dataVector;
    TElement element;
    size_t i = 0;
    char str[72] = {};
    while (true) {
        if (std::cin.eof()) {
            break;
        }
        str[0] = '\0';
```

```
            std::cin.getline(str, 72);
             if (str[0] != '\0') {
                element.key = 0;
                for (i = 0; i < 6; i++) {
                    element.index[i] = str[i];
                    element.key = element.key * 10 + str[i] - '0';
                }
                i++;
                for (; i < 72; i++) {
                    if (str[i] == '\0') {
                        break;
                    }
                    element.word[i - 7] = str[i];
                }
                element.word[64] = '\0';
                element.index[6] = '\0';
                dataVector.PushBack(element);
                for (i = 0; i < 72; i++) {
                    str[i] = 0;
                }
                 for (i = 0; i < 64; i++) {
                     element.word[i] = 0;
                 }
                 for (i = 0; i < 6; i++) {
                     element.index[i] = 0;
                 }
            }

       }
       if (dataVector.Size() != 0) {
            CountingSort(dataVector);
            for (size_t j = 0; j < dataVector.Size(); j++) {
                        std::cout << dataVector[j].index << "\t"<<
dataVector[j].word << std::endl;
            }
       }
       return 0;
}
```

Выводы

1. Ключевая особенность сортировок на GPU - многопроходность.
2. Нужно анализировать входные данные массива и его ориентировочный размер при написании алгоритма сортировки: для разных случаев он свой.
3. Атомарные операции заставляют потоки синхронизировать доступ к ячейкам и при увеличении количества элементов эффективность данного алгоритма будет падать.
4. Один из способов, позволяющий избежать использования атомарных операций при построении гистограммы, заключается в использовании отдельного набора ячеек для каждого потока и последующей свертки этих локальных гистограмм. Недостаток — при большом количестве потоков нам может элементарно не хватить памяти для хранения всех локальных гистограмм.