МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №5
по курсу «Параллельная обработка данных»

Сортировка чисел на GPU. Свертка, сканирование, гистограмма.

Студент: А. А. Садаков
Группа: 8О-406Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2022

Условие

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*).

Вариант 2: Сортировка подсчетом. Диапазон от 0 до $2^{24} - 1$.

Программное и аппаратное обеспечение

ГРАФИЧЕСКИЙ ПРОЦЕССОР
Имя устройства: NVIDIA GeForce GTX 1650
Compute capability: 7.5
Размер графической памяти: 4242604032
Размер разделяемой памяти: 49152
Размер константной памяти: 65536
Максимальное количество регистров на блок: 65536
Максимальное количество потоков на блок: 1024
Количество мультипроцессоров: 14

ПРОЦЕССОР
Имя устройства: Intel Core I5-10300H
Архитектура: Comet Lake-H
Количество ядер (потоков): 4(8)
Базовая (максимальная) частота: 2,5(4,5)ГГц
Кеш 1-го уровня: 64Кб (на ядро)
Кеш 2-го уровня: 256Кб (на ядро)
Кеш 3-го уровня: 6Мб (всего)

ОПЕРАТИВНАЯ ПАМЯТЬ
Объём: 8Гб
Тип: DDR4
Частота:2933МГц

ЖЁСТКИЙ ДИСК
Имя устройства: Intel SSDPEKNW512G8L
Тип диска: SSD
Объём памяти: 512Гб

Скорость чтения/записи: 1500/1000 Мб/с

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ
OS: Windows 10,Подсистема Ubuntu 20.04
IDE: Visual Studio Code
Компилятор: nvcc

Метод решения

Алгоритм сортировки подсчётом реализуется следующей последовательностью алгоритмов: гистограмма, сканирование, запись в результирующий массив.

Описание программы

ОСновные функции:

- $histogram$,
- $addKernel$,
- $scanKernel$,
- $CountingSortWrite$

При помощи функции $histogram$ происходит подсчёт одинаковых элементов с использованием атомарной операции $atomicAdd$.

Функция $addKernel$ нужна для добавления массива сумм к результирующему масииву после работы $scanKernel$.

Функция $scanKernel$ рекурсивно считывает блоки данных по 1024 элемента и строит префиксную сумму.

Функция $CountingSortWrite$ записывает отсортированный массив на основе префиксной суммы.

Алгоритм сортировки состоит в последовательном применении гистограммы, сканирования и записи.

Код функций:

```
1 __global__ void histogram (uint32_t *count, DATA_TYPE *data, uint32_t n)
    {
```

```
2       uint32_t idx = blockIdx.x * blockDim.x + threadIdx.x;
3       uint32_t offsetX = gridDim.x * blockDim.x;
4
5       for (uint32_t i = idx; i < n; i += offsetX) {
6           atomicAdd(count + data[i], 1);
7       }
8   }
9
10  __global__ void addKernel(uint32_t* data, uint32_t* sums, uint32_t
        sums_size) {
11      uint32_t blockId = blockIdx.x + 1;
12      while (blockId < sums_size) {
13          uint32_t idx = blockId * blockDim.x + threadIdx.x;
14          if (blockId != 0) {
15              data[idx] += sums[blockId];
16          }
17          blockId += gridDim.x;
18      }
19  }
20
21  __global__ void scanKernel(uint32_t* data, uint32_t* sums, uint32_t
        sums_size) {
22      uint32_t blockId = blockIdx.x;
23      //uint32_t idx = blockId * blockDim.x + threadIdx.x;
24      while (blockId < sums_size) {
25          __shared__ uint32_t tmp[1024];
26          uint32_t step = 1, a, b, tmp_el;
27          tmp[index(threadIdx.x)] = data[blockId * blockDim.x + threadIdx.x];
28          __syncthreads();
29          while (step < BLOCK_SIZE) {
30              if ((threadIdx.x + 1) % (step << 1) == 0) {
31                  a = index(threadIdx.x);
32                  b = index(threadIdx.x − step);
33                  tmp[a] += tmp[b];
34              }
35              step <<= 1;
36              __syncthreads();
37          }
```

```
38            if (threadIdx.x == 0) {
39                sums[blockId] = tmp[index(BLOCK_SIZE − 1)];
40                tmp[index(BLOCK_SIZE − 1)] = 0;
41            }
42            __syncthreads();
43            step = 1 << 10 − 1;
44            while (step >= 1) {
45                if ((threadIdx.x + 1) % (step << 1) == 0) {
46                    a = index(threadIdx.x);
47                    b = index(threadIdx.x − step);
48                    tmp_el = tmp[a];
49                    tmp[a] += tmp[b];
50                    tmp[b] = tmp_el;
51                }
52                step >>= 1;
53                __syncthreads();
54            }
55            data[blockId ∗ blockDim.x + threadIdx.x] = tmp[index(threadIdx.x)];
56            blockId += gridDim.x;
57        }
58    }
59
60    __global__ void CountingSortWrite (DATA_TYPE ∗data, uint32_t ∗count,
          uint32_t size) {
61        uint32_t idx = blockIdx.x ∗ blockDim.x + threadIdx.x;
62        uint32_t offsetX = gridDim.x ∗ blockDim.x;
63        uint32_t idy = blockIdx.y ∗ blockDim.y + threadIdx.y;
64        uint32_t offsetY = gridDim.y ∗ blockDim.y;
65
66        for (uint32_t i = idx; i < DATA_MAX_EL; i += offsetX) {
67            uint32_t k = count[i];
68            for (uint32_t j = idy + k; j < count[i + 1]; j += offsetY) {
69                data[j] = i;
70            }
71        }
72        for (uint32_t i = idx + count[DATA_MAX_EL]; i < size; i += offsetX) {
73            data[i] = DATA_MAX_EL;
74        }
```

Результаты

1. Замеры времени работы ядер с различными конфигурациями (время указано в микросекундах).

Конфигурации ядра \ Размер данных	100	1.000	100.000	1.000.000	10.000.000
1, 32	210 847	253 763	270 424	320 984	373 038
32, 32	7 508	7 766	8 576	8 993	9 630
32, 256	7 603	7 831	8 398	8 790	9 080
256, 256	7 512	7 766	8 332	8 571	8 908
1024, 1024	7 616	7 861	8 436	8 679	9 188

2. Сравнение с CPU

Размер входных данных	100	1.000	100.000	1.000.000	10.000.000
GPU(256, 256)	7 512	7 766	8 332	8 571	8 908
CPU	360	3932	323 424	3 320 984	32 373 038

3. nv-nsight-cu-cli

   Конфликт банков памяти был в функции сканирования. Для исправления был изменён алгоритм.

Выводы

Применение сортировки подсчётом целесообразно лишь тогда, когда сортируемые числа имеют (или их можно отобразить в) диапазон возможных значений, который достаточно мал по сравнению с сортируемым множеством, например, миллион натуральных чисел меньших 1000. Эффективность алгоритма падает при попадании нескольких различных элементов в одну ячейку, так как приходится использовать атомарные операции, замедляющие работу программы.