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 чисел на GPU. Свертка, сканирование, гистограмма.

Выполнил: Симонов С.Я.
Группа: 8О-406Б-18
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2021

Условие:
Требуется реализовать поразрядную сортировку для чисел типа uint.
Должны быть реализованы:
● Алгоритм сортировки через префиксные суммы для одного битового разряда.
● Алгоритм сканирования для любого размера, с рекурсией и бесконфликтным использованием разделяемой памяти.

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.

Вариант 8. Поразрядная сортировка.

Программное и аппаратное обеспечение
GPU:

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

Сведения о системе:

Операционная система: Ubuntu 14.04 LTS 64-x битная
Рабочая среда: nano
Компилятор: nvcc

Метод решения

Считываем данные и создаем три массива:

1) dev_data - сортируемые данные
2) deb_b - префиксная сумма
3) dev_s - копия сортируемого массива

Затем выполняется побитовая сортировка: выделяем текущий бит в соответствии с итерацией, выполняем алгоритм сканирования, с его помощью находим префиксную сумму и по вычисленной префиксной сумме сортируем массив.

Описание программы

Для реализации алгоритма сортировки через префиксные суммы для одного битового разряда были реализованы четыре ядра:

1) digit_of_a_number - побитовое смещение на i всего нашего массива, где i это текущая итерация.
2) par_scan - реализация алгоритма сканирования.
3) back - вычисление префиксной суммы.
4) radix_sort - сортировка массива по текущему биту.

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream>
#include <vector>

#define BLO 512
#define uint unsigned int

#define CSC(call)
                                \
do {
                                        \
        cudaError_t res = call;
                \
        if (res != cudaSuccess) {
                \
                fprintf(stderr, "ERROR in %s:%d. Message: %s\n",                        \
                                __FILE__, __LINE__, cudaGetErrorString(res));                \
                exit(0);
                        \
        }
                                                \
} while(0)

__global__ void digit_of_a_number(uint* dev_data, int size, int i, uint* dev_b) {
        int idx = blockIdx.x * blockDim.x + threadIdx.x;
        int offsetx = blockDim.x * gridDim.x;
        for (int j = idx; j < size; j += offsetx) {
                dev_b[j] = (dev_data[j] >> i) & 1;
        }
}

#define _index(i) ((i) + ((i) >> 8))

__global__ void par_scan(uint* dev_b, int size, uint* displace) {
        __shared__ int sdata[BLO + 120];
        int idx = threadIdx.x;
        int offsetx = (gridDim.x * blockIdx.y + blockIdx.x) * BLO;
```

```
	uint tmp;
	int index, of, s;
	if (offsetx + idx < size) {
		sdata[_index(idx)] = dev_b[offsetx + idx];
		for (s = 1; s <= BLO / 2; s <<= 1) {
			__syncthreads();
			of = s - 1;
			index = 2 * s * idx;
			if (index < BLO)
				sdata[_index(of + index + s)] += sdata[_index(of + index)];
		}
		if (idx == 0) {
			displace[gridDim.x * blockIdx.y + blockIdx.x] = sdata[_index(BLO - 1)];
			sdata[_index(BLO - 1)] = 0;
		}
		for (s = BLO / 2; s > 0; s >>= 1) {
			__syncthreads();
			of = s - 1;
			index = 2 * s * idx;
			if (index < BLO) {
				tmp = sdata[_index(of + index + s)];
				sdata[_index(of + index + s)] += sdata[_index(of + index)];
				sdata[_index(of + index)] = tmp;
			}
		}
		__syncthreads();
		dev_b[offsetx + idx] = sdata[_index(idx)];
	}
}

__global__ void back(uint* dev_b, int size, int blocks, uint* displace) {
	int idx = blockDim.x * blockIdx.x + threadIdx.x;
	int offset = gridDim.x * blockDim.x;
	for (int i = idx; i < size; i += offset) {
		dev_b[i] += displace[i / BLO];
	}
}


void scan(uint* dev_b, int size) {
	int blocks = (size - 1) / BLO + 1;
	uint* displace;
	cudaMalloc(&displace, blocks * sizeof(uint));
	if (blocks < BLO) {
		par_scan << <dim3(BLO, BLO), dim3(BLO) >> > (dev_b, size, displace);
		std::cerr << blocks << "(1) ";
		CSC(cudaGetLastError());
	}
```

```
        else {
                par_scan << <dim3(BLO, BLO), dim3(BLO) >> > (dev_b, size, displace);
                std::cerr << blocks << "(2) ";
                CSC(cudaGetLastError());
        }
        if (blocks == 1) {
                CSC(cudaFree(displace));
                return;
        }
        scan(displace, blocks);
        back << <BLO, BLO >> > (dev_b, size, blocks, displace);
        std::cerr << blocks << "(3) ";
        CSC(cudaGetLastError());
        CSC(cudaFree(displace));
}

__global__ void radix_sort(uint* dev_s, int size, int i, uint* dev_data, uint* dev_b) {
        uint ter = 0;
        if (((dev_s[size - 1] >> i) & 1) == 1) {
                ter = 1;
        }
        int idx = blockIdx.x * blockDim.x + threadIdx.x;
        int offsetx = blockDim.x * gridDim.x;
        for (int j = idx; j < size; j += offsetx) {
                if (((dev_s[j] >> i) & 1) == 0) {
                        dev_data[j - dev_b[j]] = dev_s[j];
                }
                else {
                        dev_data[dev_b[j] + (size - (dev_b[size - 1] + ter))] = dev_s[j];
                }
        }
}

int main() {
        std::ios_base::sync_with_stdio(0);
        std::cin.tie(0);
        int size;
        fread(&size, sizeof(int), 1, stdin);
        if (size == 0) {
                return 0;
        }
        uint* data = new uint[size];
        fread(data, sizeof(uint), size, stdin);
        uint* dev_data, * dev_s, * dev_b, * tmp;
        CSC(cudaMalloc(&dev_data, size * sizeof(uint)));
        CSC(cudaMalloc(&dev_s, size * sizeof(uint)));
        CSC(cudaMalloc(&dev_b, size * sizeof(uint)));
        CSC(cudaMemcpy(dev_data, data, size * sizeof(uint), cudaMemcpyHostToDevice));
```

```
	CSC(cudaMemcpy(dev_s, data, size * sizeof(uint), cudaMemcpyHostToDevice));
	for (int j = 0; j < 32; j++) {
		digit_of_a_number << <BLO, BLO >> > (dev_data, size, j, dev_b);
		CSC(cudaGetLastError());
		scan(dev_b, size);
		tmp = dev_data;
		dev_data = dev_s;
		dev_s = tmp;
		radix_sort << <BLO, BLO >> > (dev_s, size, j, dev_data, dev_b);
		CSC(cudaGetLastError());
	}
	CSC(cudaMemcpy(data, dev_data, size * sizeof(uint), cudaMemcpyDeviceToHost));
	fwrite(data, sizeof(uint), size, stdout);
	cudaFree(dev_data);
	cudaFree(dev_b);
	cudaFree(dev_s);
	free(data);
	return 0;
}
```

Результаты

Размер теста	**Время работы (секунды)**
1000	2.79993
10000	2.80032
100000	2.80427
1000000	4.24263
10000000	5.48043
100000000	8.47761

Ниже приведены результаты профилировщика nvprof. Конфликты банков памяти отсутствуют, значита алгоритм сканирования реализован с бесконфликтным использованием разделяемой памяти.

```
Invocations                              Event Name         Min         Max         Avg
Device "GeForce GT 545 (0)"
       Kernel: digit_of_a_number(unsigned int*, int, int, unsigned int*)
         32                        divergent_branch           1           1           1
         32                   global_store_transaction          96          96          96
         32                         l1_local_load_hit           0           0           0
         32                    l1_shared_bank_conflict           0           0           0
       Kernel: radix_sort(unsigned int*, int, int, unsigned int*, unsigned int*)
         32                        divergent_branch           1           1           1
         32                   global_store_transaction          96         288         152
         32                         l1_local_load_hit           0           0           0
         32                    l1_shared_bank_conflict           0           0           0
       Kernel: back(unsigned int*, int, int, unsigned int*)
         32                        divergent_branch           1           1           1
         32                   global_store_transaction          96          96          96
         32                         l1_local_load_hit           0           0           0
         32                    l1_shared_bank_conflict           0           0           0
       Kernel: par_scan(unsigned int*, int, unsigned int*)
         64                        divergent_branch           1           1           1
         64                   global_store_transaction           6         102          54
         64                         l1_local_load_hit           0           0           0
         64                    l1_shared_bank_conflict           0           0           0

==8604== Metric result:
Invocations                              Metric Name                        Metric Description         Min         Max         Avg
Device "GeForce GT 545 (0)"
       Kernel: digit_of_a_number(unsigned int*, int, int, unsigned int*)
         32                             sm_efficiency                   Multiprocessor Activity      61.73%      78.50%      75.68%
       Kernel: radix_sort(unsigned int*, int, int, unsigned int*, unsigned int*)
         32                             sm_efficiency                   Multiprocessor Activity      73.49%      78.22%      75.63%
       Kernel: back(unsigned int*, int, int, unsigned int*)
         32                             sm_efficiency                   Multiprocessor Activity      71.88%      77.41%      75.23%
       Kernel: par_scan(unsigned int*, int, unsigned int*)
         64                             sm_efficiency                   Multiprocessor Activity      92.37%      92.62%      92.50%
```

Конфликты банков памяти исчезли, а это значит программа работает корректно, дополнительное время не тратиться.

Выводы

Выполнив данную лабораторную работу, я реализовал распараллеленный алгоритм поразрядной сортировки при помощи алгоритма сканирования. Сразу хочется отметить важность данного алгоритма, поскольку он является однима из основых в параллельной обработке данных.

В ходе выполнения возникло множество проблем с правильной реализацией программного кода сканирования (возникала неоднократная потребность в помощи, за ней обращался к Александру Юрьевичу). Было весьма трудно вычлинять ошибки, так как основной алгоритм выполнялся в 32 итерации а ошибка могла появиться на любой из них. Очень важно отметить, что не стоит вставлять костыли в ЧЕТКО НАПИСАННЫЙ алгоритм, такие вещи приводят только к исправлению найденной ошибки, но при этом создаются еще и новые неполадки.