3МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Факультет прикладной математики и физики
Кафедра вычислительной математики и программирования

Лабораторная работа №5
по курсу «Параллельная Обработка Данных»

Сортировка чисел на GPU. Свертка, сканирование, гистограмма.

Выполнил: Иларионов Д.А.
Группа: М8О-408Б-17
Преподаватели: Крашенинников К.Г.,
Морозов А.Ю.

Москва, 2020

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 4. Сортировка чет-нечет.

Требуется реализовать блочную сортировку чет-нечет для чисел типа int.
Должны быть реализованы:

● Алгоритм чет-нечет сортировки для предварительной сортировки блоков.

● Алгоритм битонического слияния, с использованием разделяемой памяти.

Ограничения: $n \leq 16 * 10^6$

Программное и аппаратное обеспечение

GPU:

- Name: GeForce GTX 1060
- Compute capability: 6.1
- Частота видеопроцессора: 1404 – 1670 (Boost) МГц
- Частота памяти: 8000 МГц
- Графическая память: 6144 МБ
- Разделяемая память: 2048 Б
- Количество регистров на блок: 65536
- Максимальное количество блоков: (2147483647, 65535, 65535)
- Максимальное количество нитей: (1024, 1024, 64)
- Количество мультипроцессоров: 10

Сведения о системе:

- Процессор: Intel Core i7-8750H 2.20GHz x 6
- Оперативная память: 16 ГБ
- SSD: 128 ГБ
- HDD: 1000 ГБ

Программное обеспечение:

- OS: Windows 10
- IDE: Visual Studio 2019
- Компилятор: nvcc

Метод решения

По сути данная лабораторная должна быть легче предыдущей. Однако, с ней я возился еще дольше, и она мне показалась сложнее, не смотря на несложный вариант. Много раз не проходила по времени. Сначала, я неправильно понял условие, и вместо битонического слияния каждый раз сортировал чет-нечет. Потом, я сделал рекурсивное битоническое слияние, преподаватель сказал, что это плохо, поэтому снова не проходила по времени на 14 тесте. Затем, у меня был неправильный ответ. Как оказалось – гонка потоков. А дело в том, что синхронизация работала некорректно,

потому что была в конце 2 ветвей, и в итоге синхронизировались не все потоки. Да, я переделал рекурсию в цикл. А когда я перенес синхронизацию из ветвей в конец цикла, то все заработало. А сама программа работает так. Во первых у нас все данные в битовом формате (как в 2-3 ЛР), в итоге пришлось помучиться с тем, как их считывать и записывать, но нашел в интернете и разобрался. Далее, мы применяем сортировки и используем разделяемую память, доступ к которой примерно в 100 раз быстрее, чем к глобальной. Это позволяет выполнять итерации быстрее, поэтому и программа прошла в итоге по времени. Мы сначала сортируем чет-нечет весь массив. В итоге у нас блоки с возрастающими элементами. Радует то, что разделяемая память – общая для всех потоков одного блока, в итоге я мог запускать по 1024 блока потоков одновременно, что тоже очень оптимизировало программу. Далее мы используем битоническое слияние n+1 раз, в итоге у нас получается отсортированный массив, который мы так же записываем в выходной бинарный файл.

Описание программы

Файл kernel.cu

```
#include <string>
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
//for __syncthreads()
#ifndef __CUDACC_RTC__
#define __CUDACC_RTC__
#endif // !(__CUDACC_RTC__)

#include <device_functions.h>

#define SHARED_MEMORY 512
#define MAXIMAL_INTEGER 2147483647 // 2^31 - 1

using namespace std;


#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)

using namespace std;


class ClassSort {
public:
	int* intArray;
	int size = 0;
```

```
	ClassSort()
	{
		freopen(NULL, "rb", stdin);
		fread(&size, sizeof(int), 1, stdin);
		cerr << "Len: " << size << "\n";

		intArray = (int*)malloc(sizeof(*intArray) * size);

		fread(intArray, sizeof(int), size, stdin);

		//for (int i = 0; i < size; i++) {
		//	cerr << (int)intArray[i] << " ";
		//}

		fclose(stdin);
	};

	ClassSort(string inpFile) {
		FILE* file;
		if ((file = fopen(inpFile.c_str(), "rb")) == NULL) {
			cerr << "Can't load file" << "\n";
			exit(1);
		}

		fread(&size, sizeof(size), 1, file);
		cerr << "Len: " << size << "\n";
		intArray = (int*)malloc(sizeof(*intArray) * size);

		fread(intArray, sizeof(int), size, file);

		fclose(file);
	}

	void PrintRSS() {
		for (int i = 0; i < size - 1; i++) {
			if ((int)intArray[i + 1] < (int)intArray[i]) {
				cerr << (int)intArray[i] << " : " << (int)intArray[i+1] <<
"| ";
			}
		}
		cerr << "\n";
		freopen(NULL, "wb", stdout);
		fwrite(intArray, sizeof(*intArray), size, stdout);
		fclose(stdout);
	}

	void PrintRSS(string outFile) {
		for (int i = 0; i < size; i++) {
			cerr << (int)intArray[i] << " ";
		}
		cerr << "\n";

		FILE* file = fopen(outFile.c_str(), "wb");
		fwrite(intArray, sizeof(*intArray), size, file);
		fclose(file);
	}

	~ClassSort() {
		fclose(stdin);
		fclose(stdout);
		free(intArray);
	};
};


__device__ void oddEvenOne(const int tid, int* tmp, int shift, int len) {
	if ((tid + shift) % 2 == 0) {
		int a = tid;
```

```
		int b = tid + 1;
		if (b < len && tmp[a] > tmp[b]) {
			int temp = tmp[a];
			tmp[a] = tmp[b];
			tmp[b] = temp;
		}
	}
}


__global__ void OddEvenBlocks(int* theArray, int size, int shift2) {

	int tx = gridDim.x * blockDim.x;
	int tt = blockDim.x;
	int tid = threadIdx.x;
	int bid = blockDim.x * blockIdx.x;

	int tid_full = bid + threadIdx.x;

	__shared__ int tmp[SHARED_MEMORY];
	__shared__ int shift;


	for (int begin = bid + shift2; begin < size; begin += tx) {

		tid_full = begin + threadIdx.x;

		int end = (size > begin + tt ? begin + tt : size);
		int len = end - begin;

		if (tid < len) {
			tmp[tid] = theArray[tid_full];
		}
		else {
			tmp[tid] = MAXIMAL_INTEGER;
		}

		__syncthreads();

		for (int j = 0; j < len; ++j) {
			if (j % 2 == 0) {
				shift = 0;
			}
			else {
				shift = 1;
			}
			oddEvenOne(tid, tmp, shift, len);
			__syncthreads();
		}

		if (tid < len) {
			theArray[tid_full] = tmp[tid];
		}

		__syncthreads();
	}
}



__global__ void BitonicMerge(int* theArray, int size, int shift2) {


	int tx = gridDim.x * blockDim.x;
	int tt = blockDim.x; // 2 * SHARED_MEMORY
	int tid = threadIdx.x;
	int bid = blockDim.x * blockIdx.x;

	int shiftt = shift2;
```

```
	int tid_full = bid + threadIdx.x;
	__shared__ int tmp[2 * SHARED_MEMORY];



	for (int begin = bid + shiftt; begin < size; begin += tx) {
		tid_full = begin + threadIdx.x;

		int end = (size > begin + tt ? begin + tt : size);
		int len = end - begin;

		if ((tid < tt/2) && (tid < len)) {
			tmp[tid] = theArray[tid_full];
		}
		else if (tid < len) {
			tmp[tt + tt/2 - tid - 1] = theArray[tid_full];
		}
		else if ((tid < tt) && (tid >= tt / 2)) {
			tmp[tt + tt / 2 - tid - 1] = MAXIMAL_INTEGER;
		}
		else {
			tmp[tid] = MAXIMAL_INTEGER;
		}

		__syncthreads();

		int base = tt / 2;
		int shift = 0;
		int n_tid = tid;

		while (base >= 1) {
			if (n_tid >= base) {
				int opTid = n_tid - base;
				if (tmp[opTid + shift] > tmp[n_tid + shift]) {
					int temp = tmp[opTid + shift];
					tmp[opTid + shift] = tmp[n_tid + shift];
					tmp[n_tid + shift] = temp;
				}
				if (base >= 1) {
					base = base / 2;
					n_tid = n_tid - base;
					shift = shift + base;
				}
			}
			else {
				if (base >= 1) {
					base = base / 2;
				}
			}
			__syncthreads();
		}

		__syncthreads();


		if (tid < len) {
			theArray[tid_full] = tmp[tid];
		}

		__syncthreads();

	}
}


void Sorting(ClassSort* sort) {
	int* D_theArray;
```

```
	CSC(cudaMalloc((void**)& D_theArray, sizeof(*D_theArray) * sort->size));

	CSC(cudaMemcpy(D_theArray, sort->intArray, sizeof(*sort->intArray) * sort-
>size, cudaMemcpyHostToDevice));

	//pre-entry odd-even sort

	OddEvenBlocks << <1024, SHARED_MEMORY >> > (D_theArray, sort->size, 0);

	CSC(cudaGetLastError());


	//cycle bitonic-merge / odd-even sort
	int iters = (sort->size + 1) / SHARED_MEMORY + 2;
	for (int i = 0; i < iters; ++i) {
		int evOdd = i % 2;


		BitonicMerge << <1024, 2 * SHARED_MEMORY >> > (D_theArray, sort->size,
evOdd * SHARED_MEMORY);
		CSC(cudaGetLastError());


		//cerr << "Iter: " << i << "\n";
	}

	CSC(cudaMemcpy(sort->intArray, D_theArray, sizeof(int) * sort->size,
cudaMemcpyDeviceToHost));
	CSC(cudaFree(D_theArray));
}




int main(void) {

	//ClassSort theSort = ClassSort("test.data");
	//Sorting(&theSort);
	//theSort.PrintRSS("test_out.data");

	ClassSort theSort = ClassSort();
	Sorting(&theSort);
	theSort.PrintRSS();

	return 0;
}
```

Результаты

Приведу таблицу и результаты nvprof. Программа в visual studio у меня почему-то сразу закрывалась и не давала полных логов. Поэтому, я успел сделать лишь пару скриншотов (для 10 и для 1 млн. элементов). В таблице отобразил время работы для одинакового количества блоков (1024), но с разной разделяемой памятью и числом нитей. А также для CPU я перестал тестировать примерно после 50 тыс. элементов, тк это уже занимало очень много времени, поэтому я примерно прикинул время, но оно не точное.

	Количество блоков = 1024						
El / Core [ms]	SM = 4	SM = 16	SM = 64	SM = 128	SM = 256	SM = 512	CPU
10	0,037	0,037	0,034	0,035	0,035	0,034	0,11
100	0,077	0,043	0,035	0,035	0,034	0,048	0,21
1 000	0,405	0,349	0,062	0,042	0,037	0,041	16,36
10 000	3,774	1,080	0,259	0,155	0,122	0,067	3 914
20 000	58,53	6,829	0,512	0,279	0,160	0,098	15 883
50 000	3 916	8,12	1,221	0,904	0,326	0,184	107 725
100 000	19 653	855	2,465	1,229	0,617	0,325	500 000
200 000	90 448	6 196	4,798	2,420	1,243	0,616	2 500 000
Avg. Geom	33,972	4,526	0,330	0,222	0,147	0,106	1154,178

C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.1\bin\nvprof.exe

```
Len: 101
==24560== NVPROF is profiling process 24560, command: Lab5.exe >prof.txt
==24560== Warning: System profiling could not be enabled on the underlying platform.
-49 -48 -47 -46 -45 -44 -43 -42 -41 -40 -39 -38 -37 -36 -35 -34 -33 -32 -31 -30 -29 -28 -27 -26 -25 -24 -23 -22 -21 -20
-19 -18 -17 -16 -15 -14 -13 -12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51
==24560== Profiling application: Lab5.exe >prof.txt
==24560== Profiling result:
   Start  Duration           Grid Size      Block Size     Regs*    SSMem*    DSMem*      Size  Throughput  SrcMemType
 DstMemType          Device   Context    Stream  Name
373.02ms  1.0880us                    -               -         -         -         -      404B  354.12MB/s    Pageable
     Device  GeForce GTX 106         1         7  [CUDA memcpy HtoD]
373.03ms  589.32us          (1024 1 1)       (512 1 1)        28  2.0039KB        0B         -           -           -
         -  GeForce GTX 106         1         7  OddEvenBlocks(int*, int, int) [113]
373.62ms  186.27us          (1024 1 1)      (1024 1 1)        26  4.0000KB        0B         -           -           -
         -  GeForce GTX 106         1         7  BitonicMerge(int*, int, int) [115]
373.80ms  179.78us          (1024 1 1)      (1024 1 1)        26  4.0000KB        0B         -           -           -
         -  GeForce GTX 106         1         7  BitonicMerge(int*, int, int) [117]
373.98ms     640ns                    -               -         -         -         -      404B  602.01MB/s      Device
   Pageable  GeForce GTX 106         1         7  [CUDA memcpy DtoH]

Regs: Number of registers used per CUDA thread. This number includes registers used internally by the CUDA driver and/or
 tools and can be more than what the compiler shows.
SSMem: Static shared memory allocated per CUDA block.
DSMem: Dynamic shared memory allocated per CUDA block.
SrcMemType: The type of source memory accessed by memory operation/copy
DstMemType: The type of destination memory accessed by memory operation/copy
```

C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.1\bin\nvprof.exe

```
         -  GeForce GTX 106         1         7  BitonicMerge(int*, int, int) [851]
1.64957s  847.69us          (1024 1 1)      (1024 1 1)        26  4.0000KB        0B         -           -           -
         -  GeForce GTX 106         1         7  BitonicMerge(int*, int, int) [853]
1.65042s  851.40us          (1024 1 1)      (1024 1 1)        26  4.0000KB        0B         -           -           -
         -  GeForce GTX 106         1         7  BitonicMerge(int*, int, int) [855]
1.65127s  846.85us          (1024 1 1)      (1024 1 1)        26  4.0000KB        0B         -           -           -
         -  GeForce GTX 106         1         7  BitonicMerge(int*, int, int) [857]
1.65212s  854.21us          (1024 1 1)      (1024 1 1)        26  4.0000KB        0B         -           -           -
         -  GeForce GTX 106         1         7  BitonicMerge(int*, int, int) [859]
1.65297s  851.21us          (1024 1 1)      (1024 1 1)        26  4.0000KB        0B         -           -           -
         -  GeForce GTX 106         1         7  BitonicMerge(int*, int, int) [861]
1.65383s  849.89us          (1024 1 1)      (1024 1 1)        26  4.0000KB        0B         -           -           -
         -  GeForce GTX 106         1         7  BitonicMerge(int*, int, int) [863]
1.65468s  853.67us          (1024 1 1)      (1024 1 1)        26  4.0000KB        0B         -           -           -
         -  GeForce GTX 106         1         7  BitonicMerge(int*, int, int) [865]
1.65553s  853.80us          (1024 1 1)      (1024 1 1)        26  4.0000KB        0B         -           -           -
         -  GeForce GTX 106         1         7  BitonicMerge(int*, int, int) [867]
1.65638s  844.58us          (1024 1 1)      (1024 1 1)        26  4.0000KB        0B         -           -           -
         -  GeForce GTX 106         1         7  BitonicMerge(int*, int, int) [869]
1.65723s  850.57us          (1024 1 1)      (1024 1 1)        26  4.0000KB        0B         -           -           -
         -  GeForce GTX 106         1         7  BitonicMerge(int*, int, int) [871]
1.65809s  849.29us          (1024 1 1)      (1024 1 1)        26  4.0000KB        0B         -           -           -
         -  GeForce GTX 106         1         7  BitonicMerge(int*, int, int) [873]
1.65894s  855.65us          (1024 1 1)      (1024 1 1)        26  4.0000KB        0B         -           -           -
         -  GeForce GTX 106         1         7  BitonicMerge(int*, int, int) [875]
1.65979s  852.29us          (1024 1 1)      (1024 1 1)        26  4.0000KB        0B         -           -           -
         -  GeForce GTX 106         1         7  BitonicMerge(int*, int, int) [877]
1.66064s  854.31us          (1024 1 1)      (1024 1 1)        26  4.0000KB        0B         -           -           -
         -  GeForce GTX 106         1         7  BitonicMerge(int*, int, int) [879]
```

Выводы

В данной лабораторной мне пришлось работать с бинарным вводом и выводом. А также изучить некоторые сортировки. Из-за проблем со скоростью программы потратил много времени и нервов, а также потревожил преподавателя. Но в итоге, я доделал лабу! И наконец, свобода) Хотя, я думаю, я еще сделаю необязательную лабу по OpenGL, но это уже после праздников. А так я закрепил знания о сортировках, ну и понял, как работать с разделяемой памятью и как правильно синхронизировать потоки, чтобы не было их гонки. Также в выводе напишу еще кое-что. Поздравляю вас с наступающим 2021 годом, желаю, чтобы он принес вам много хорошего, чтобы этот год стал одним из лучших в вашей жизни! Счастья, удачи, любви, крепкого здоровья и стальных нервов, желаю успеха на работе, и исполнения всех желаний. Ведь в Новый Год случаются чудеса! Нужно лишь верить в это. Все будет хорошо:)