МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

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

Лабораторная работа №3
по курсу «Параллельная обработка данных»

Сортировка чисел на GPU. Свертка, сканирование, гистограмма

Выполнил: И.А. Мариничев
Группа: 8О-408Б
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.

Вариант 5: сортировка чет-нечет с предварительной битонической сортировкой.

Программное и аппаратное обеспечение

Compute capability	:	2.1
Name	:	GeForce GT 545
Total Global Memory	:	3150381056
Shared memory per block	:	49152
Registers per block	:	32768
Warp size	:	32
Max threads per block	:	(1024, 1024, 64)
Max block	:	(65535, 65535, 65535)
Total constant memory	:	65536
Multiprocessors count	:	3
Processor	:	Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
RAM	:	16 GB
Drive	:	349G
OS	:	Ubuntu 16.04.6 LTS
IDE	:	Visual Studio Code
Compiler	:	NVIDIA (R) Cuda compiler driver V7.5.17

Метод решения

Считываем бинарные входные данные. В случае, если размер массива не кратен размеру блока, то дополняем наш массив значениями INT_MAX. Затем проводим битоническую сортировку для каждого блока. Если наш массив был меньше размера блока, то на этом этапе сортировка завершается. В остальных случаях проводим битоническое слияние четных пар блоков, а затем нечетных пар блоков (второй блок пары мы берем в обратном порядке следования элементов, для образования битонической последовательности). Данный этап проводим число блоков раз. В конце выводим полученный ответ, обрезая хвост добавленный в начале, и освобождаем память.

Описание программы

```
// функция для заполнения массива значениями INT_MAX до размера кратного BLOCK_SIZE
__global__ void fill_array(int *array, int n, int value)

// функция реализующая полуочиститель B(n)
__device__ void half_cleaner(int *buff, int M, int B, unsigned int idx1)

// функция реализующая полуочиститель B(n) для каждого блока
__global__ void bitonic_sort(int *array, int n, int M, int B)

// функция реализующая битоническое слияние M(BLOCK_SIZE) для четных-нечетных пар блоков
__global__ void even_odd_merge(int *array, int n, int start, int end)

// Блочная сортировка чет-нечет:

// Этап 1: Битоническая сортировка блоков
for (int M = 2; M <= BLOCK_SIZE; M <<= 1) { // цикл по битоническим слияниям M(n)
	for (int B = M; B >= 2; B >>= 1) { // цикл по полуочистителям B(n)
		bitonic_sort<<<BLOCKS, BLOCK_SIZE>>>(dev_array, new_n, M, (B >> 1));
		CSC(cudaGetLastError());
	}
}

// в случае, если размер массива равен BLOCK_SIZE, то на этом сортировка завершается
int blocks_count = new_n / BLOCK_SIZE;
if (blocks_count > 1) {
	// Этап 2: Битонические  слияния
	for (int i = 0; i < blocks_count; i++) {
		// сливаем четные пары блоков
		even_odd_merge<<<BLOCKS, BLOCK_SIZE>>>(dev_array, new_n, 0, new_n);
		CSC(cudaGetLastError());

		// сливаем нечетные пары блоков
		even_odd_merge<<<BLOCKS, BLOCK_SIZE>>>(dev_array, new_n, (BLOCK_SIZE / 2), new_n - BLOCK_SIZE);
		CSC(cudaGetLastError());
	}
}
```

Результаты

Конфигурация	Размер теста (время указано в ms)				
	O(10^2)	O(10^3)	O(10^4)	O(10^5)	O(10^6)
<<< 1, 1024 >>>	0.227328	0.326400	4.751424	363.439392	34763.664062
<<< 32, 1024 >>>	0.283968	0.365440	2.223488	132.212128	12058.425781
<<< 256, 1024 >>>	1.609664	1.782176	4.248000	145.081055	12037.632812
<<< 512, 1024 >>>	3.120320	3.406464	6.635360	155.215591	12303.497070
<<< 1024, 1024 >>>	6.156832	6.663968	11.437728	175.595581	12967.238281
CPU	26.0	2440.0	167746.0	1.0475e+07	1.05043e+09

Результаты профилирования

Ниже представлены результаты профилирования программы при n=16000 (конфигурация <<< 32, 1024 >>>):

```
==25964== Event result:
Invocations                                Event Name         Min         Max         Avg
Device "GeForce GT 545 (0)"
    Kernel: fill_array(int*, int, int)
          1                            divergent_branch           0           0           0
          1                  global_store_transaction          36          36          36
          1                   l1_shared_bank_conflict           0           0           0
          1                         l1_local_load_hit           0           0           0
    Kernel: even_odd_merge(int*, int, int, int)
         32                            divergent_branch        2400        2560        2480
         32                  global_store_transaction         480         576         528
         32                   l1_shared_bank_conflict           0           0           0
         32                         l1_local_load_hit           0           0           0
    Kernel: bitonic_sort(int*, int, int, int)
         55                            divergent_branch           0        1524         538
         55                  global_store_transaction         576         576         576
         55                   l1_shared_bank_conflict           0           0           0
         55                         l1_local_load_hit           0           0           0

==25964== Metric result:
Invocations                                Metric Name                        Metric Description         Min         Max         Avg
Device "GeForce GT 545 (0)"
    Kernel: fill_array(int*, int, int)
          1                             sm_efficiency                   Multiprocessor Activity      37.47%      37.47%      37.47%
    Kernel: even_odd_merge(int*, int, int, int)
         32                             sm_efficiency                   Multiprocessor Activity      83.48%      88.92%      86.18%
    Kernel: bitonic_sort(int*, int, int, int)
         55                             sm_efficiency                   Multiprocessor Activity      71.55%      76.72%      73.63%
```

Выводы

Сложность первого этапа (битонической сортировки): n * log(n) * log(n), где log(n) - битонических слияний, log(n) - битонических полуочистителей, n - один полуочиститель (распараллеливается). Сложность второго этапа (сортировки чет-нечет): 2*n (распараллеливается) * n. Как видно из результатов сравнения, GPU дает значительный прирост по времени, особенно при больших n. Результаты профилирования показали, что конфликты банков памяти и спиллинг регистров отсутствуют и имеется небольшая дивергенция потоков.