МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Факультет прикладной математики и физики
Кафедра вычислительной математики и программирования

Лабораторная работа №5
по курсу «Программирование графических процессоров»

Сортировка чисел на GPU.
Свертка, сканирование, гистограмма.

Выполнил: Н.И. Забарин
Группа: 8О-408Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2017

Условие

1. **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.
2. **Вариант задания:**
   Вариант 4. Сортировка чет-нечет.

Программное и аппаратное обеспечение

Программное и аппаратное обеспечение

Спецификации GPU

Name:	GeForce GT 620M
Compute capability:	2.1
Warp size:	32
Max threads per block:	1024
Clock rate:	1250000
Multiprocessor count:	2
Max threads dim:	1024 1024 64
Max grid size:	65535 65535 65535

Спецификации видеопамяти

Total global memory:	1024 MB
Shared memory per block:	48 KB
Registers per block:	32 KB
Total constant memory:	64 KB

Спецификации CPU

Процессор	Intel Core i5-3317U
Ядер	4
Базовая частота	1.7 GHz

Спецификация оперативной памяти

Объем памяти	10 Гб
Частота	1600 МГц

Спецификация жесткого диска

Тип	SSD
Интерфейс	M.2
Объём	240Gb

Спецификация программного обеспечения

CUDA Toolkit	7.5
OS	Ubuntu 16.10
IDE	Vim
Compiler	nvcc V7.5.17

Метод решения

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: $n \leq 16 * 10^6$

- Алгоритм сортировки "Чет - Нечет"

Алгоритм представляет вариацию алгоритма пузырьковой сортировки. Интересен он тем, что допускает естественное распараллеливание. Как и в алгоритме пузырька, внешний цикл задает n проходов по сортируемому массиву. В каждом проходе, происходит сравнение и обмен двух соседних элементов. Но есть два важных отличия:

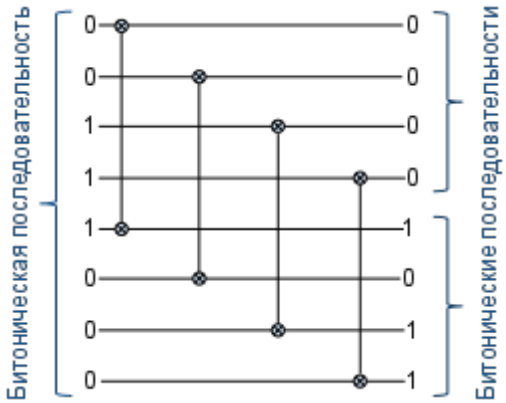
1. В каждом проходе производится n/2 независимых сравнений соседних пар, так что никакой элемент пары не участвует в дальнейших сравнениях при данном проходе;
2. Проходы делятся на четные и нечетные. На четных проходах обмен начинается с пары (a_0, a_1). На нечетном проходе производится сдвиг и начальной парой является пара (a_1, a_2). (Предполагается, что нумерация элементов массива начинается с нуля).

- Битоническое слияние

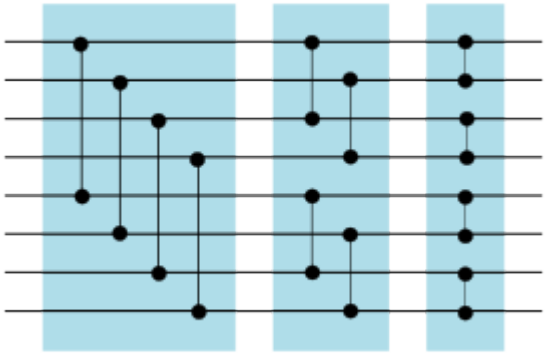
Битонической последовательностью (англ. bitonic sequence) называется конечный упорядоченный набор (кортеж) из вещественных чисел, в котором они сначала монотонно возрастают, а затем монотонно убывают, или набор, который приводится к такому виду путем циклического сдвига.

Битонический сортировщик представляет собой каскад так называемых **полуфильтров** (англ. *half-cleaner*). Каждый полуфильтр — сеть компараторов единичной глубины, в которой i-й входной провод сравнивается со входным проводом с номером $\frac{n}{2} + i$, где $i = 1, 2, \ldots, \frac{n}{2}$(число входов n— чётное).

Теперь используем полуфильтры для сортировки битонических последовательностей. Один полуфильтр разделяет битоническую последовательность на две равные части, одна из которых однородна, а другая сама по себе является битонической последовательностью, причем части расположены в правильном порядке. Тогда мы можем каждую часть снова отправить в полуфильтр вдвое меньшего размера, чем предыдущий. Затем, если нужно, четыре получившихся части снова отправить в полуфильтры и так далее, пока количество проводов в одной части больше 1.

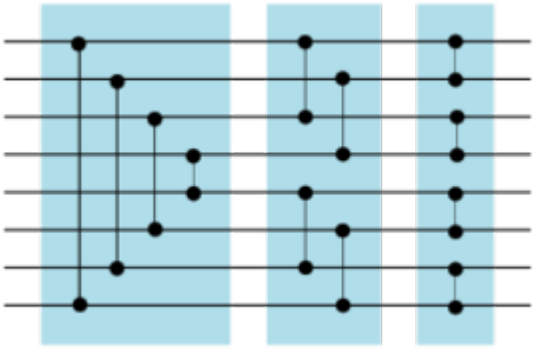


Полуфильтр для 8 проводов.



Битонический сортировщик на восемь входов с выделенными полуфильтрами.

Объединяющая сеть (англ. *merger*) — сеть компараторов, объединяющая две отсортированные входные последовательности в одну отсортированную выходную последовательность.



Сеть, объединяющая две отсортированные последовательности из четырёх чисел в одну отсортированную последовательность из восьми чисел.

Описание программы

В программе объявляются две константы
const int BLOCK_SIZE = 1024; - размер блоков, сортируемых четно-нечетной сортировкой.
const int GRID_SIZE = 16384; - количество блоков.

То есть, мксимальный размер мссива, который может быть отсортирован данной программой – 16777216 элементов.

BLOCK_SIZE должен быть степенью двойки, так как на этапе битонического слияния, сортируются блоки размера BLOCK_SIZE * 2, а битоническая объединяющая сеть работает с блоками, имеющими размер, равный степени 2.

Если длина массива не кратна BLOCK_SIZE, массив дополняется значениями INF, заведомо большими, чем сортируемые элементы. Так как в лаборторной работе требуется сортировать числа типа int, то значание INF берется как:
const int64_t INF = 2147483647;

Пусть длина массива равна n. Тогда в начале n / 2 раз вызывается ядро
__global__ void k_blocksort(int64_t *arr, int len), которое выполняет два шага сравнения и обмена (нечетный проход и четный).

После этого этапа, массив представляет собой последовательность n / BLOCK_SIZE отсортированных блоков. Для их объединения используем битоническое слияние, которое похоже на обобщение четно-нечетной сортировки. Блоки сливаются попарно и независимо от других пар. На четных шагах слияние начинается с блоков (b_0, b_1), на нечетных – с (b_1, b_2).

За битоническое слияние отвечает ядро
__global__ void k_merge(int64_t *arr, int len, bool odd)
Ядро вызывается n / BLOCK_SIZE. После этого массив будет отсортирован.

В каждом ядре каждый блок в сетке работате только со своим блоком из массива, независимо от остальных блоков. Поэтому данные для каждого блока потоков в начале работы алгоритма заносятся в разделяемую память

Результаты

Сортируются массивы длины, равной степени 2.

n	Время
256 (2^8)	0.431776
512 (2^8)	0.433344
1024 (2^10)	3.743104
16384 (2^14)	53.183777
131072 (2^17)	451.714752
1048576 (2^20)	5318.974609
4194304 (2^22)	45081.019531

Выводы

Разделяемая память позволяет ускорить выполнение программы за счет уменьшения времени доступа к памяти. Так же она удобна тем, что является общей для всех потоков одного блока. Она может использоваться в таких фундаментальных параллельных алгоритмах как: histogram, reduce, scan.

Однако стоит не забывать, что производительность может уменьшаться при образовании конфликтов доступа к банкам памяти.