

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ  
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»

**Лабораторная работа №5**  
**по курсу «Программирование графических процессоров»**  
***Освоение программного обеспечения для работы с технологией CUDA.***

***Сортировка чисел на GPU. Свертка, сканирование, гистограмма.***

Выполнил: Бачурин Павел

Дмитриевич

Группа: М8О-403Б-22

Преподаватели: А.Ю. Морозов,  
Е.Е. Заяц

Москва, 2025

## **Условие**

Цель работы:

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 (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**

Графический процессор (GPU):

Модель: NVIDIA GeForce MX330

Архитектура CUDA: Pascal

Compute Capability: 6.1

Видеопамять (VRAM): 2 ГБ

Пропускная способность памяти: ~64 ГБ/с

Количество CUDA-ядер: 384

Разделяемая память на блок: 48 КБ

Константная память: 64 КБ

Количество регистров на блок: 65 536 32-битных регистров

Максимальное количество блоков (на одном мультипроцессоре): 1024

Максимальное количество потоков (на один блок): 1024

Количество мультипроцессоров: 3

Центральный процессор (CPU):

Модель: Intel i5-1035G1

Количество ядер: 4

Количество потоков на ядро: 2

Тактовая частота: 3.6 ГГц

Количество оперативной памяти: 12 ГБ

Объем жесткого диска: 200 ГБ

Операционная система:

Операционная система: Manjaro Linux Gnome x86\_64

Ядро: Linux 6.x

Программная среда: Lenovo 81WD IdeaPad 3 14IIL05

CUDA Toolkit: 12.9

Драйвер NVIDIA: 580.82.09

Компилятор gcc: 11.4.0

## Метод решения

### 1) Подготовка данных на CPU

- 1.** Из стандартного входного потока (stdin) считывается целое число  $n$  — количество элементов массива.
- 2.** На CPU выделяется память под массив  $h\_data$  размером  $n$  элементов типа int.
- 3.** Все  $n$  элементов массивачитываются из бинарного входного потока.
- 4.** Для корректной работы битонической сортировки определяется ближайшая степень двойки  $n2 \geq n$ .
- 5.** Формируется расширенный массив  $h\_padded$  длины  $n2$ :
  - первые  $n$  элементов копируются из входного массива;
  - оставшиеся элементы заполняются значением INT\_MAX, что гарантирует корректность сортировки по возрастанию.
- 6.** CPU выделяет память на GPU:
  - $d\_data$  — массив входных данных в глобальной памяти GPU.
- 7.** Подготовленный массив  $h\_padded$  копируется в  $d\_data$  с помощью cudaMemcpy.

### 2) Подготовка GPU и конфигурации вычислений

- 1.** Используется фиксированная конфигурация:
  - размер блока BLOCK\_SIZE = 512,
  - количество блоков GRID\_SIZE = 512.
- 2.** Общее количество потоков (TOTAL\_THREADS) не зависит от размера входных данных, что соответствует требованиям лабораторной работы.
- 3.** Все вызовы CUDA API обрабатываются макросом CSC(...) для проверки ошибок выполнения.

### 3) Параллельные вычисления на GPU

Алгоритм сортировки состоит из двух основных этапов.

#### 3.1 — Блочная чёт-нечет сортировка

Запускается ядро:

```
odd_even_sort_block<<<GRID_SIZE, BLOCK_SIZE>>>(d_data, n2);
```

Каждый блок GPU:

- загружает BLOCK\_SIZE последовательных элементов массива из глобальной памяти в разделяемую память (shared memory);
- выполняет чёт-нечет сортировку внутри блока.

Алгоритм чёт-нечет сортировки состоит из последовательности фаз:

- на чётных фазах сравниваются пары (0,1), (2,3), ...;
- на нечётных фазах сравниваются пары (1,2), (3,4), ...).

Каждая операция сравнения и обмена выполняется параллельно потоками одного блока.

Для синхронизации потоков используется `__syncthreads()`.

После завершения сортировки внутри блока результат записывается обратно в глобальную память GPU.

#### 3.2 — Битоническое слияние

После локальной сортировки блоков выполняется глобальное битоническое слияние всего массива.

Для каждого значения  $k = 2, 4, 8, \dots, n_2$  выполняются фазы:

```
for (int j = k / 2; j > 0; j /= 2)
```

Запускается ядро:

```
bitonic_merge_global<<<GRID_SIZE, BLOCK_SIZE>>>(d_data, n2, j, k);
```

Каждый поток:

- обрабатывает один элемент массива;
- находит индекс партнёра  $ixj = idx \wedge j$ ;
- определяет направление сортировки (по возрастанию или убыванию) на основе текущей фазы;
- выполняет сравнение и, при необходимости, обмен элементов.

Для предотвращения гонок данных обмен выполняется только тем потоком, для которого  $ixj > idx$ .

Между фазами осуществляется синхронизация на уровне CPU с помощью:

```
cudaDeviceSynchronize();
```

Это гарантирует корректность выполнения битонического алгоритма.

5) Завершение работы и возврат результата

- 1.** Отсортированный массив копируется из `d_data` в `h_data`.
- 2.** В стандартный выходной поток (`stdout`) выводятся только первые `n` элементов (без паддинга).
- 3.** Освобождается память GPU (`cudaFree`) и память CPU (`free`).

## Основные функции и CUDA-ядра

### 1. `CSC(call)`

Проверяет корректность вызовов CUDA API и завершает программу при возникновении ошибки.

### 2. `odd_even_sort_block`

Выполняет чёт-нечет сортировку элементов внутри одного блока с использованием разделяемой памяти.

### 3. `bitonic_merge_global`

Реализует одну фазу битонического слияния всего массива в глобальной памяти GPU.

## Результаты

- Сравнение времени выполнения программы на CUDA в зависимости от количества потоков для массива из 10.000.000 чисел

Количество потоков	Время выполнения (ms)
16 x 16	5180.379
64 x 64	772.814
128 x 128	765.687
512 x 512	518.422

- Сравнение программы на CUDA 512x512 и программы на CPU с одним потоком:

Размер массива	Время на CUDA (в мс)	Время на CPU (в мс)
100.000	10.907	216.852
1.000.000	53.638	2039.754
10.000.000	518.422	34535.420

## Исследование производительности

```
nvprof ./gpu.out < in.bin > out.bin
==25425== NVPROF is profiling process 25425, command: ./gpu.out
==25425== Profiling application: ./gpu.out
==25425== Profiling result:
```

GPU activities

Тип операции	Доля времени, %	Общее время	Кол-во вызовов	Среднее время	Минимум	Максимум	Имя
GPU kernel	61.51	644.09 ms	300	2.1470 ms	1.3518 ms	2.8299 ms	bitonic_merge_global(int*, int, int, int)
GPU kernel	35.22	368.78 ms	1	368.78 ms	368.78 ms	368.78 ms	odd_even_sort_block(int*, int)
Memory copy	2.10	21.976 ms	1	21.976 ms	21.976 ms	21.976 ms	CUDA memcpy HtoD

Тип операции	Доля времени, %	Общее время	Кол-во вызовов	Среднее время	Минимум	Максимум	Имя
Memory copy	1.18	12.339 ms	1	12.339 ms	12.339 ms	12.339 ms	CUDA memcpuy DtoH

CUDA API calls

API вызов	Доля времени, %	Общее время	Кол-во вызовов	Среднее время	Минимум	Максимум
cudaDeviceSynchronize	91.22	1.01442 s	301	3.3702 ms	1.3563 ms	368.79 ms
cudaMalloc	5.49	60.998 ms	1	60.998 ms	60.998 ms	60.998 ms
cudaMemcpy	3.06	34.082 ms	2	17.041 ms	12.509 ms	21.573 ms
cudaLaunchKernel	0.18	1.9975 ms	301	6.6360 µs	2.2160 µs	513.45 µs
cudaFree	0.02	271.80 µs	1	271.80 µs	271.80 µs	271.80 µs
cuDeviceGetAttribute	0.01	113.13 µs	114	0.992 µs	0.065 µs	40.023 µs
cudaGetLastError	0.00	35.752 µs	301	0.118 µs	0.054 µs	11.112 µs
cudaEventRecord	0.00	31.436 µs	2	15.718 µs	9.792 µs	21.644 µs
cuDeviceGetName	0.00	23.206 µs	1	23.206 µs	23.206 µs	23.206 µs
cudaEventCreate	0.00	18.740 µs	2	9.370 µs	0.420 µs	18.320 µs
cudaEventElapsedTime	0.00	9.112 µs	1	9.112 µs	9.112 µs	9.112 µs
cuDeviceGet	0.00	4.464 µs	2	2.232 µs	0.177 µs	4.287 µs
cudaEventSynchronize	0.00	3.177 µs	1	3.177 µs	3.177 µs	3.177 µs
cuDeviceGetCount	0.00	1.585 µs	3	0.528 µs	0.078 µs	1.335 µs
cuDeviceGetPCIBusId	0.00	1.442 µs	1	1.442 µs	1.442 µs	1.442 µs
cuDeviceTotalMem	0.00	0.411 µs	1	0.411 µs	0.411 µs	0.411 µs

API вызов	Доля времени, %	Общее время	Кол-во вызовов	Среднее время	Минимум	Максимум
cuModuleGetLoadingMode	0.00	0.234 $\mu\text{s}$	1	0.234 $\mu\text{s}$	0.234 $\mu\text{s}$	0.234 $\mu\text{s}$
cuDeviceGetUuid	0.00	0.132 $\mu\text{s}$	1	0.132 $\mu\text{s}$	0.132 $\mu\text{s}$	0.132 $\mu\text{s}$

## Выводы

В ходе лабораторной работы была реализована параллельная сортировка массива целых чисел с использованием технологии CUDA. Цель работы — изучение принципов программирования GPU и организации параллельных вычислений — была достигнута.

Был реализован гибридный алгоритм сортировки, сочетающий чётно-нечётную сортировку внутри блоков и глобальное битоническое слияние. Также была разработана CPU-версия алгоритма для сравнения. Корректность работы программы подтверждена тестированием.

Профилирование показало, что основное время выполнения на GPU затрачивается на этап битонического слияния, а значительная часть накладных расходов связана с синхронизацией потоков. В результате работы были получены практические навыки разработки, отладки и анализа производительности CUDA-программ.