

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 массива считываются 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 оборачиваются 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 $k = 2, 4, 8, \dots, n2$ 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-нечёт 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

1. 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

2. 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 memcpy 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 μ s	2.2160 μ s	513.45 μ s
cudaFree	0.02	271.80 μ s	1	271.80 μ s	271.80 μ s	271.80 μ s
cuDeviceGetAttribute	0.01	113.13 μ s	114	0.992 μ s	0.065 μ s	40.023 μ s
cudaGetLastError	0.00	35.752 μ s	301	0.118 μ s	0.054 μ s	11.112 μ s
cudaEventRecord	0.00	31.436 μ s	2	15.718 μ s	9.792 μ s	21.644 μ s
cuDeviceGetName	0.00	23.206 μ s	1	23.206 μ s	23.206 μ s	23.206 μ s
cudaEventCreate	0.00	18.740 μ s	2	9.370 μ s	0.420 μ s	18.320 μ s
cudaEventElapsedTime	0.00	9.112 μ s	1	9.112 μ s	9.112 μ s	9.112 μ s
cuDeviceGet	0.00	4.464 μ s	2	2.232 μ s	0.177 μ s	4.287 μ s
cudaEventSynchronize	0.00	3.177 μ s	1	3.177 μ s	3.177 μ s	3.177 μ s
cuDeviceGetCount	0.00	1.585 μ s	3	0.528 μ s	0.078 μ s	1.335 μ s
cuDeviceGetPCIBusId	0.00	1.442 μ s	1	1.442 μ s	1.442 μ s	1.442 μ s
cuDeviceTotalMem	0.00	0.411 μ s	1	0.411 μ s	0.411 μ s	0.411 μ s

API вызов	Доля времени, %	Общее время	Кол-во вызовов	Среднее время	Минимум	Максимум
cuModuleGetLoadingMode	0.00	0.234 μ s	1	0.234 μ s	0.234 μ s	0.234 μ s
cuDeviceGetUuid	0.00	0.132 μ s	1	0.132 μ s	0.132 μ s	0.132 μ 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.