

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»

**Лабораторная работа №2
по курсу «Основы CUDA»**

Обработка изображений на GPU. Фильтры.

Выполнил: *Ши Хуэй*
Группа: 638
Преподаватели: А.Ю. Морозов,
Е.Е. Заяц

Москва, 2025

Условие

Цель работы

Изучить принципы обработки изображений на GPU и использование текстурной памяти CUDA при реализации оператора Робертса для выделения контуров.

Формат изображений. Изображение является бинарным файлом, со следующей структурой:

| width(w) | height(h) | r | g | b | a | r | g | b | a | r | g | b | a | ... | r | g | b | a | r | g | b | a |
|-----------------|-----------------|--|--|--|-----|---|---|---|---|---|---|---|---|-----|---|---|---|---|---|---|---|---|
| 4 байта, int | 4 байта, int | 4 байта, значение пикселя [1,1] | 4 байта, значение пикселя [2,1] | 4 байта, значение пикселя [3,1] | ... | 4 байта, значение пикселя [w - 1, h] | 4 байта значение пикселя [w,h] | | | | | | | | | | | | | | | |

В первых восьми байтах записывается размер изображения, далее построчно все значения пикселей, где

- r -- красная составляющая цвета пикселя
- g -- зеленая составляющая цвета пикселя
- b -- синяя составляющая цвета пикселя
- a -- значение альфа-канала пикселя

Вариант задания - Выделение контуров. Метод Робертса.

Программное и аппаратное обеспечение

Работа выполнялась на вычислительном кластере «Полюс» Московского государственного университета имени М. В. Ломоносова.

Кластер состоит из 5 вычислительных узлов с пиковой производительностью **55.84 TFlop/s** и производительностью по тесту Linpack — **40.39 TFlop/s**.

На каждом вычислительном узле:

- **Процессоры:** 2 × IBM Power 8
- **Пиковая производительность DP:** $\approx 580 \text{ GFlop/s}$ (290×2)
- **Графические процессоры:** 2 × NVIDIA Tesla P100
- **Число процессорных ядер:** 20 (по 10 на каждом CPU)
- **Число потоков на ядро:** 8
- **Оперативная память:** 256 ГБ (в отдельных узлах до 1024 ГБ)
- **Коммуникационная сеть:** Infiniband / 100 Gb
- **Система хранения данных:** GPFS
- **Операционная система:** Linux Red Hat 7.5

Характеристики GPU — NVIDIA Tesla P100:

- **Compute Capability:** 6.0
- **Общая глобальная память:** 16 ГБ (16384 МиБ)
- **Shared memory per block:** 49152 байт
- **Constant memory:** 65536 байт
- **Registers per block:** 65536
- **Максимальное число нитей в блоке:** 1024 (1024, 1024, 64)

- **Максимальное число блоков:** (2 147 483 647, 65 535, 65 535)
- **Количество мультипроцессоров:** 56

Программное обеспечение:

- **Операционная система:** Linux Red Hat 7.5 (на узлах Polus)
- **Среда разработки:** SSH-доступ к кластеру Polus через Visual Studio Code или nano
- **Компилятор:** nvcc 12.0 (с поддержкой g++ 13.3.0)
- **CUDA Toolkit:** версия 12.0
- **Дополнительные инструменты:** MPI, OpenMP (по необходимости), система очередей LSF для запуска задач
- **Библиотеки:** стандартные CUDA API, без использования внешних (Thrust, cuBLAS и др.)

Метод решения

Для решения задачи использован **оператор Робертса** — простой метод выделения контуров на изображении.

Он основан на вычислении разности яркостей между диагональными соседними пикселями.

Если обозначить яркость пикселя как $I(x, y)$, то градиенты по диагоналям вычисляются так:

$$G_x = I(x + 1, y + 1) - I(x, y), \quad G_y = I(x + 1, y) - I(x, y + 1)$$

После этого общая интенсивность границы определяется как:

$$G = \sqrt{G_x^2 + G_y^2}$$

Значение G ограничивается диапазоном 0–255 и записывается в каждый из трёх цветовых каналов (R, G, B), чтобы получить чёрно-белое изображение.

Программа реализована на CUDA, чтобы выполнять эти вычисления **параллельно** для всех пикселей изображения.

Для чтения исходного изображения используется **текстурная память** — это позволяет удобно обращаться к пикселям в двумерных координатах и автоматически обрабатывать выход за границы (режим clamp).

Каждый поток GPU обрабатывает один пиксель, вычисляет яркость через линейную комбинацию компонент ($0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$) и применяет оператор Робертса.

Таким образом, вычисление производится в два шага:

1. чтение данных из файла и подготовка текстуры;
2. запуск CUDA-ядра roberts, которое выполняет фильтрацию.

Описание программы

Основные функции:

1.gray_of(uchar4 p)

Это небольшая функция устройства, которая переводит цветной пиксель в оттенок серого.Формула стандартная для RGB-яркости:

$$gray = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

Возвращается значение float, чтобы избежать потерь точности при вычислениях градиента.

2.roberts(...) — основное ядро CUDA

Каждая нить обрабатывает один пиксель изображения.

Используется двумерная сетка потоков (grid, block), поэтому обращение к пикселям идёт по координатам (x, y).

Изображение считывается через **текстурный объект** `cudaTextureObject_t tex`, что упрощает доступ к данным и автоматически обрабатывает выход за границы (режим clamp).

Ядро получает четыре соседних пикселя:

$$\begin{aligned}p11 &= \text{tex2D}(tex, x, y); \\p21 &= \text{tex2D}(tex, x + 1, y); \\p12 &= \text{tex2D}(tex, x, y + 1); \\p22 &= \text{tex2D}(tex, x + 1, y + 1);\end{aligned}$$

После этого вычисляются диагональные разности и итоговая интенсивность:

$$\begin{aligned}gx &= w22 - w11; \\gy &= w21 - w12; \\G &= \sqrt{gx * gx + gy * gy};\end{aligned}$$

Результат записывается в выходной буфер как оттенок серого (одинаковое значение для R, G и B).

3.main()

В главной функции происходит:

- чтение входных данных (`in.data`) и размеров w, h;
- создание и настройка текстурного объекта (`cudaResourceDesc`, `cudaTextureDesc`);
- выделение GPU-памяти для результата (`uchar4* dev_out`);
- запуск ядра:

```
dim3block(n_block, n_block);
dim3grid(n_grid, n_grid );
roberts <<< grid, block >>> (tex, dev_out, w, h);
```

- копирование результата обратно на CPU и запись в файл `out.data`.

После завершения работы освобождаются все ресурсы CUDA — уничтожается текстурный объект, освобождаются массивы и буфера.

Программа не использует сторонние библиотеки, полностью соответствует требованиям лабораторной работы и легко переносится между системами с поддержкой CUDA.

Результаты

В ходе экспериментов были проведены замеры времени работы ядра Робертса на GPU и на CPU при различных конфигурациях запуска и разных размерах изображений.

Для измерения времени использовалась встроенная функция `cudaEventElapsedTime()` для CUDA и `std::chrono` для C++-версии.

Во всех случаях измерялось только время работы ядра (kernel), без учёта операций чтения и записи файлов.

Единица измерения времени — миллисекунды (мс).

Таблица 1 — Сравнение времени выполнения (мс)

| Конфигурация | Картина 1 | | Картина 2 | | Картина 3 | |
|--------------|-----------|-------|-----------|--------|-----------|---------|
| | CUDA | C++ | CUDA | C++ | CUDA | C++ |
| <<<1, 8>>> | 0.0313 | 4.775 | 0.0344 | 42.387 | 0.0371 | 106.765 |
| <<<1, 32>>> | 0.0325 | | 0.0349 | | 0.0348 | |
| <<<8, 8>>> | 0.0309 | | 0.0347 | | 0.0356 | |
| <<<2, 16>>> | 0.0310 | | 0.0348 | | 0.0343 | |
| <<<8,32>>> | 0.0355 | | 0.0380 | | 0.0371 | |
| <<<32,8>>> | 0.0340 | | 0.0373 | | 0.0366 | |

Выходы

Реализованный фильтр Робертса корректно выполняет выделение контуров изображений на GPU.

Применение технологии CUDA позволило достичь многократного ускорения вычислений по сравнению с последовательной реализацией на CPU.

Полученные результаты подтверждают преимущества параллельной обработки изображений и демонстрируют практическую эффективность GPU-ускорения для задач компьютерного

Кроме того, в ходе эксперимента были обработаны три изображения разного размера. На заключительных страницах отчёта приведены результаты фильтрации — изображения после применения оператора Робертса, на которых отчётливо видны выделенные границы объектов.

Картина 1 - Размер изображения: 500 x 500



Картинка 2 - Размер изображения: 1290 x 1720



Картинка 3 - Размер изображения: 2732 x 2048

