

Санкт–Петербургский государственный университет

Криворучко Денис Игоревич

Отчёт по домашней работе 1

Приложение для работы с изображениями

Санкт-Петербург

2021 г.

Содержание

Глава 1. Введение	3
Глава 2. Используемые инструменты	3
Глава 3. Детали реализации	4
3.1. Frontend	4
3.2. Backend	4
Глава 4. Постановка экспериментов и измерение производи- тельности	7
4.1. Эксперименты	7
4.2. Анализ	10
Глава 5. Заключение	10
Список литературы	11

Глава 1. Введение

В процессе выполнения домашней работы 1, было разработано приложение для обработки изображений. Поддержаны следующие возможности: просмотр информации об изображении (размер, высота, ширина), применение одного или нескольких матричных фильтров, выбор из заранее заданных фильтров, а также конструирование собственного. Приложение позволяет работать с одним изображением или с директорией. В случае работы с одним изображением, предусмотрена также возможность просмотреть результат наложения фильтров и, при желании, сохранить его. Если же происходит обработка списка изображений, которые хранятся в папке, полученные изображения сразу сохраняются. Важной особенностью реализации является возможность применения фильтров с помощью центрального процессора (CPU) или видеокарты (GPU).

Данный отчет содержит обзор разработанного продукта, а также анализ производительности и сравнение скорости обработки на CPU и на GPU.

Глава 2. Используемые инструменты

При разработке приложения использован язык Python 3. Для Frontend-части выбран фреймворк PyQt5 [1], который предоставляет большое количество примитивов для взаимодействия с пользователем. Поскольку большая часть работы в backend-части программы связана с обработкой матрицы изображений, при расчетах на CPU, используется модуль NumPy [2]. Данный модуль подходит для задач, связанных с линейной алгеброй, так как здесь базовые операции над матрицами и векторами отлично оптимизированы. Для вычисления на видеокарте использована библиотека CuPy [3]. CuPy является аналогом NumPy, за исключением того, что расчеты проводятся на GPU. На самом деле, CuPy — это оболочка библиотеки CUDA [4] языка C, разработанной для взаимодействия с видеокартами NVIDIA.

Глава 3. Детали реализации

3.1 Frontend

Приложение содержит 6 окон:

1. `main_view` — для выбора режима работы (обработка одного изображения или директории)
2. `one_picture_view` — для просмотра информации об изображении
3. `filters_view` — для выбора фильтров, которые будут применены к изображению
4. `create_filter_view` — для создания фильтра
5. `two_pictures_view` — для сравнения изображения до и после применения выбранных фильтров
6. `saved_view` — для просмотра папки, в которую сохранен результат наложения фильтров

Общая архитектура frontend-части приложения представлена на рисунке 1. Все взаимодействие с backend-частью происходит через `filters_view`.

3.2 Backend

Backend-часть приложения состоит из трех основных компонентов. Первый из них — `computing`, отвечает за взаимодействие с frontend. Модули `cpu_computing` и `gpu_computing` нужны для наложения фильтров с использованием CPU и GPU соответственно.

Наложение матричных фильтров происходит следующим образом: фильтр задается некоторой матрицей F с нечетными высотой h и шириной w . Далее фильтр применяется к каждому пикселю матрицы изображения следующим образом: в изображении выбирается подматрица M , размерности которой совпадают с размерностями матрицы фильтра. Причем текущий пиксель должен являться центральным в M . После этого формируется новая матрица: она представляет из себя поэлементное умножение

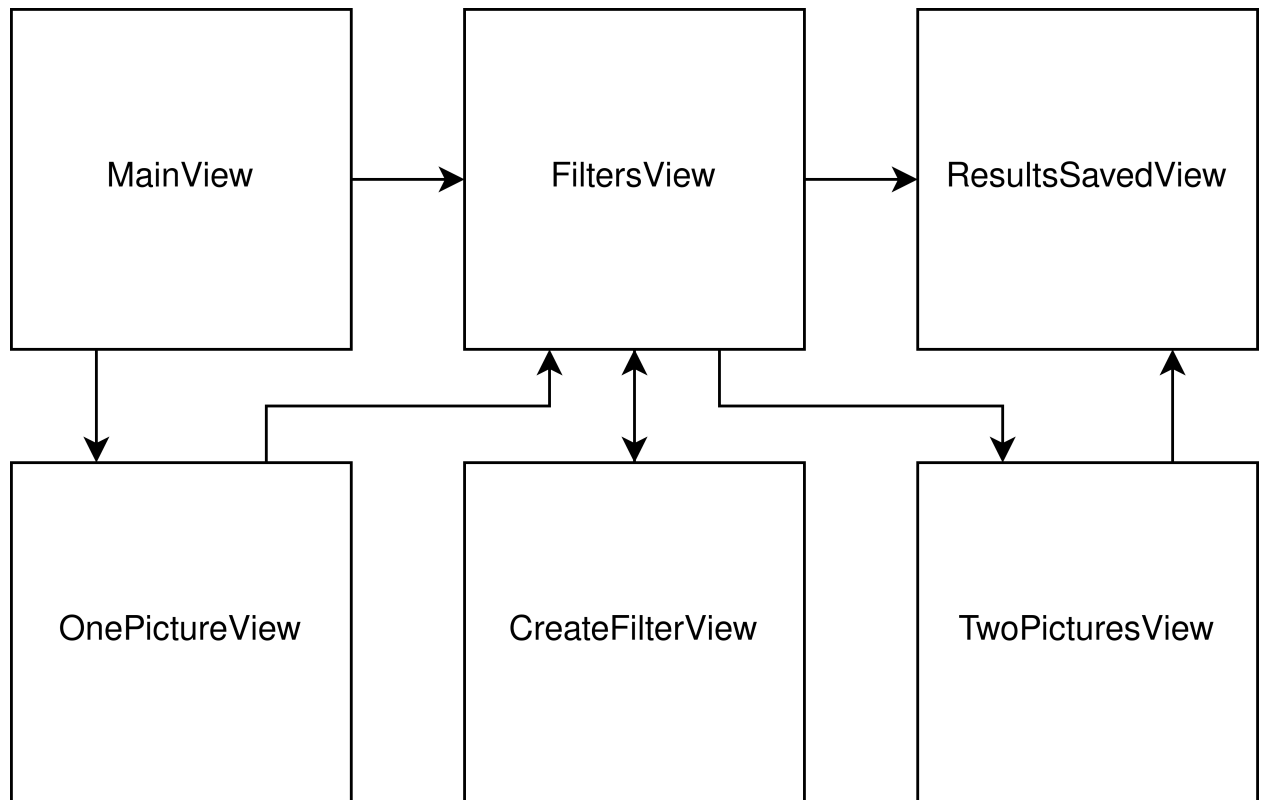


Рис. 1: Диаграмма Frontend-части приложения.

матрицы F на матрицу M . Значение пикселя в изображении, которое является результатом применения фильтра, — это сумма элементов матрицы M . Для того, чтобы работать с пикселями, которые находятся на границе изображения, формируется расширенная матрица изображения. Ее высота определяется формулой 1, а ширина — формулой 2. Алгоритм заполнения матрицы представлен на листинге 1.

Листинг 1: Заполнение расширенной матрицы изображения

```

# Input: p_m — pixmap matrix
#         p_h — pixmap matrix height
#         p_w — pixmap matrix width
#         f_h — filter matrix height
#         f_w — filter matrix width

```

Пусть высота матрицы изображения — H_1 , высота матрицы фильтра — H_2 . Тогда высота расширенной матрицы $H = H_1 + H_2 - 1$

Формула 1: Высота расширенной матрицы

Пусть ширина матрицы изображения — W_1 , ширина матрицы фильтра — W_2 . Тогда ширина расширенной матрицы $W = W_1 + W_2 - 1$

Формула 2: Ширина расширенной матрицы

Output: m

m = [[0, ..., 0], ..., [0, ..., 0]]

for i = 0 to (p_h + f_h - 1) - 1:

h = 0

if i <= f_h / 2: h = 0

else if f_h / 2 < i < f_h / 2 + p_h: h = i - f_h / 2

else: h = p_h - 1

for j = 1 to (p_w + f_w - 1) - 1:

w = 0

if j <= f_w / 2: w = 0

else if f_w / 2 < j < f_w / 2 + p_w: w = j - f_w / 2

else: w = p_w - 1

m[i][j] = p_m[h][w]

В реализации модуля `cru_computing` все действия происходят последовательно. При применении фильтра, сначала формируется расширенная матрица изображения. Затем фильтр применяется отдельно к каждому пикселю. Если фильтров и изображений несколько, то каждый фильтр применяется поочередно к каждому изображению.

В реализации модуля `gru_computing` применение фильтра к изображению происходит параллельно, с использованием нескольких ядер гра-

фического процессора. Это реализовано с помощью возможностей библиотеки Cpu и дает существенный рост производительности даже с учетом того, что все данные о фильтрах и изображениях необходимо сначала передать видеокарте, а после обработки, вернуть информацию на центральный процессор. Случай применения нескольких фильтров к нескольким изображениям реализован так же, как и в `cpu_computing`: каждый фильтр применяется поочередно к каждому изображению.

Благодаря тому, что `frontend` и `backend` полученного решения связаны единственным методом, приложение является расширяемым. Например, для того, чтобы добавить новый стандартный фильтр, необходимо создать одну кнопку в `filters_view`, и указать матрицу фильтра в конфигурационном файле. Если же потребуются новый способ применения фильтров (например на CPU, но параллельно), `backend` таких вычислений нужно реализовать аналогично с двумя другими, уже существующими модулями, а в окне `filters_view` добавить одну кнопку.

Глава 4. Постановка экспериментов и измерение производительности

4.1 Эксперименты

Измерения производились на компьютере с процессором INTEL Core i5 8th Gen (2.30GHz), видеокартой NVIDIA GeForce GTX 1050ti, 16 гб оперативной памяти и операционной системой Ubuntu 20.04.

Первый эксперимент заключался в следующем: производительность решений с использованием CPU и GPU сравнивалась отдельно на трех изображениях: 246×205 , 275×183 , 646×1000 . К каждому из них применялось два фильтра: 5×5 и 3×3 . Результаты экспериментов представлены на рисунке 2.

Второй эксперимент был таким: к 5 изображениям из выбранной директории применялось 5 фильтров 5×5 . Результат этого эксперимента представлен на рисунке 3.

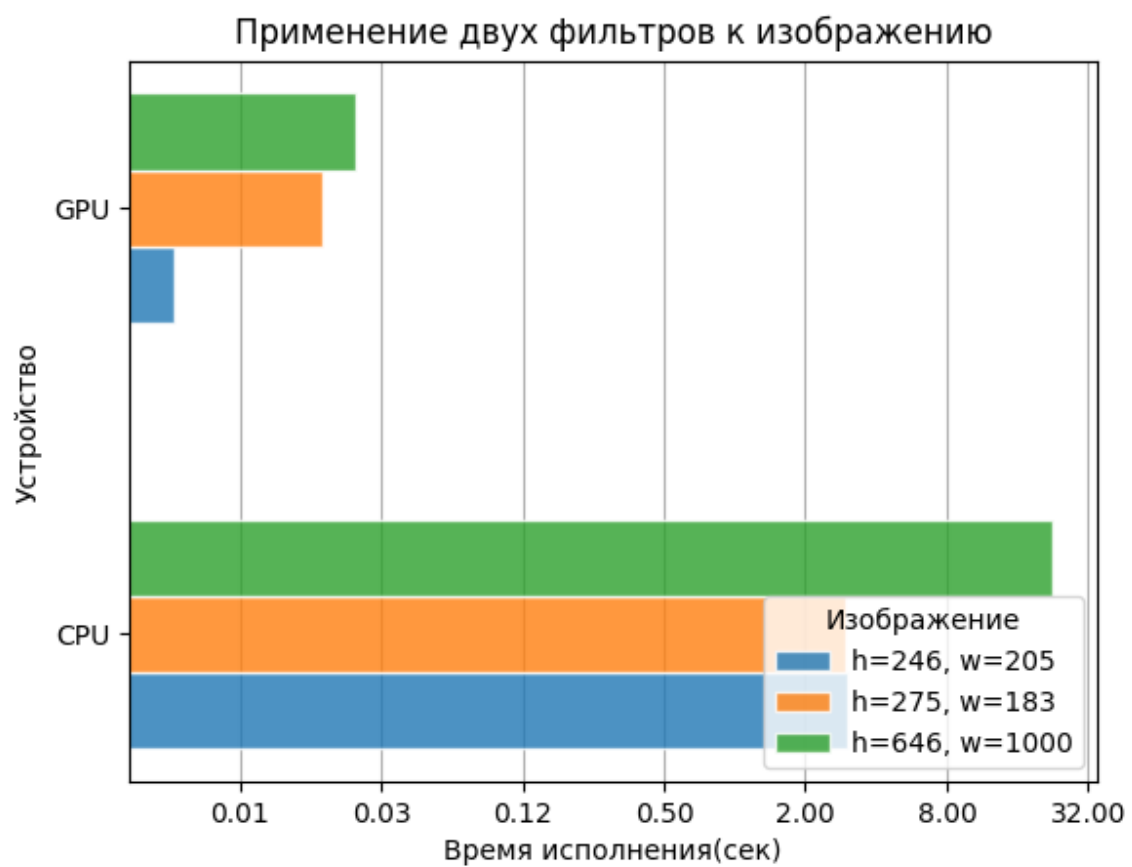


Рис. 2: Примнение фильтров к изображению.

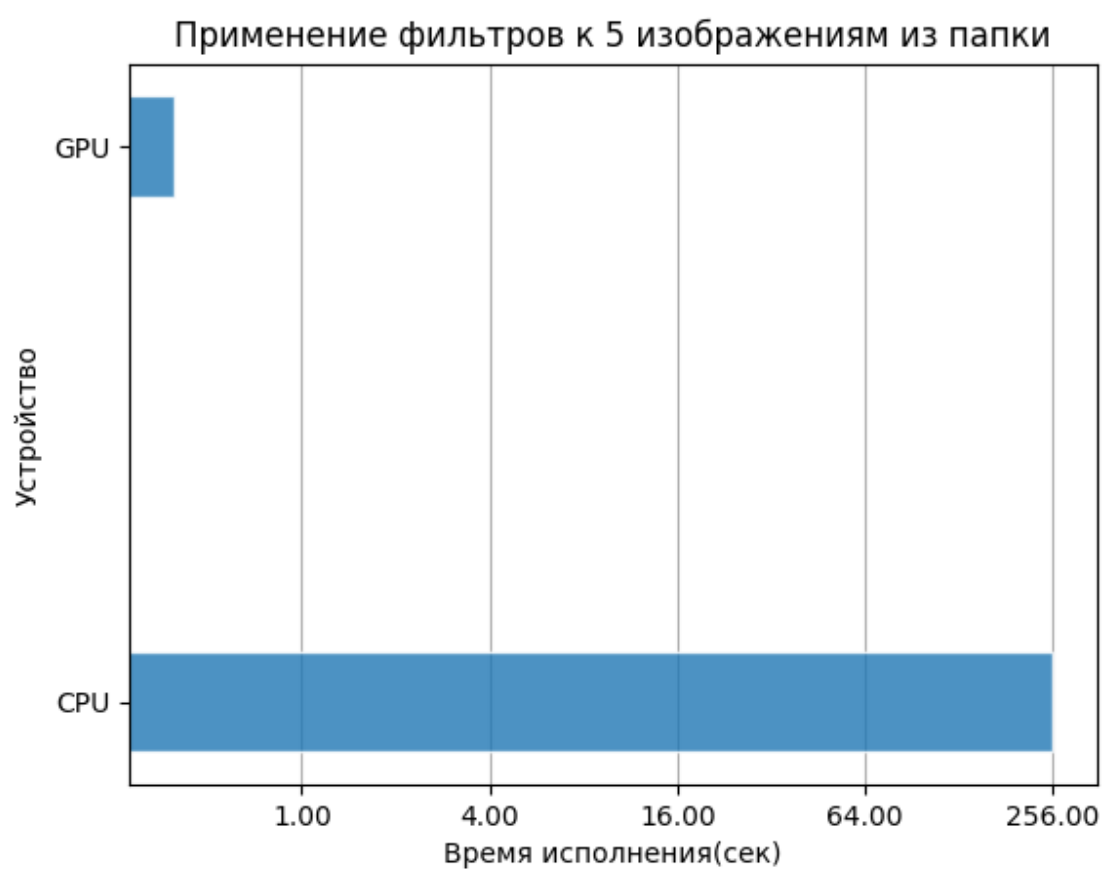


Рис. 3: Применение фильтров к папке.

4.2 Анализ

Несмотря на то, что при использовании видеокарты в расчетах, время уходит не только на применение фильтров, но и на обмен данными между графическим и центральным процессорами, реализация на GPU показывает значительно лучшие результаты. В среднем, при работе с CPU, время, затрачиваемое на обработку одного изображения, более, чем в 300 раз превосходит время, которое требуется при наложении фильтров на GPU.

Существенно, что при увеличении входных данных, время расчетов на графическом процессоре увеличивается незначительно — менее, чем в 2 раза, при увеличении изображения примерно в 12 раз, в то время, как время обработки на CPU при таком же увеличении данных, растет более, чем в 7 раз.

Что касается работы с несколькими изображениями, на GPU время растет незначительно и при применении 5 фильтров к 5 изображениям остается меньшим секунды. Однако, те же расчеты на CPU занимают более 4 минут.

Глава 5. Заключение

На основе проведенных экспериментов можно сделать вывод, что для решения задач линейной алгебры, в которых требуется со всеми элементами матриц или векторов производить однообразные независимые операции, например, при наложении матричных фильтров на изображение, использование GPU является более оптимальным, чем использование CPU.

Список литературы

- [1] PyQt documentation. URL: <https://doc.qt.io/qtforpython/>
- [2] NumPy documentation. URL: <https://numpy.org/>
- [3] CuPy documentation. URL: <https://docs.cupy.dev/en/stable/>
- [4] CUDA documentation. URL: <https://docs.nvidia.com/cuda/>