

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	6
4.1. Эксперименты	6
4.2. Анализ	9
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

Все взаимодействие с frontend- и backend-частей происходит через `filters_view`.

3.2 Backend

Backend-часть приложения состоит из четырех основных компонентов. Модули `computing` и `folder_computing`, отвечают за взаимодействие с frontend. Соответственно, в первом реализована функциональность работы с конкретным изображением, а во втором — с папкой. Модули `cpu_computing` и `gpu_computing` нужны для наложения 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 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. Если высота матрицы фильтра равна h , а высота матрицы изображения равна h' , то высота расширенной матрицы: $H = h + h' - 1$. Ширина расширенной матрицы определяется аналогично.

В реализации модуля `cru_computing` все действия происходят последовательно. При применении фильтра, сначала формируется расширенная матрица изображения. Затем фильтр применяется отдельно к каждому пикселю.

В реализации модуля `gru_computing` применение фильтра к изображению происходит параллельно, с использованием нескольких ядер графического процессора. Это реализовано с помощью возможностей библиотеки Сиру и дает существенный рост производительности даже с учетом того, что все данные о фильтрах и изображениях необходимо сначала предать видеокарте, а после обработки, вернуть информацию на центральный процессор.

Работа с директорией, в которой содержатся изображения реализована так: для чтения, обработки и сохранения изображений создан отдельный поток. Это дает возможность не хранить в памяти все изображения одновременно и уменьшает время работы в сравнении с последовательным наложением фильтров к каждому изображению из папки. Общение между этими тремя потоками реализовано с помощью двух очередей. Первая очередь содержит изображения, готовые к обработке, а вторая — готовые к сохранению.

Благодаря тому, что `frontend` и `backend` полученного решения связаны всего двумя методами, 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 потребуется новый способ применения фильтров (напри-

мер 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 последовательно применялось два фильтра: 5×5 и 3×3 . При постановке данного эксперимента вычислялось только время наложения фильтров. В случае расчетов на GPU, учитывалось также время передачи данных между видеокартой и центральным процессором. Опыт проводился 10 раз. Всякий раз фильтры генерировались случайно. В качестве результатов проведения эксперимента выбрано среднее время работы по 10 замерам. Результаты последовательного наложения этих фильтров представлены на рисунке 1.

Во втором эксперименте вычислялось время применения фильтра 101×101 к изображению 646×1000 . Данный эксперимент проводился так же, как и предыдущий: измерялось только время наложения фильтров. Замеры сделаны 10 раз, в качестве результата выбрано среднее время по всем замерам. Результаты представлены на рисунке 2.

Третий эксперимент был таким: ко всем изображениям из выбранной директории, которая содержала 50 изображений размеров 100×100 , последовательно применялось 5 фильтров 5×5 . В данном эксперименте время расчетов состоит не только из времени наложения фильтров, но и из считывания изображений и их сохранения после обработки. Для сравнения результатов, был также проведен замер времени применения тех же

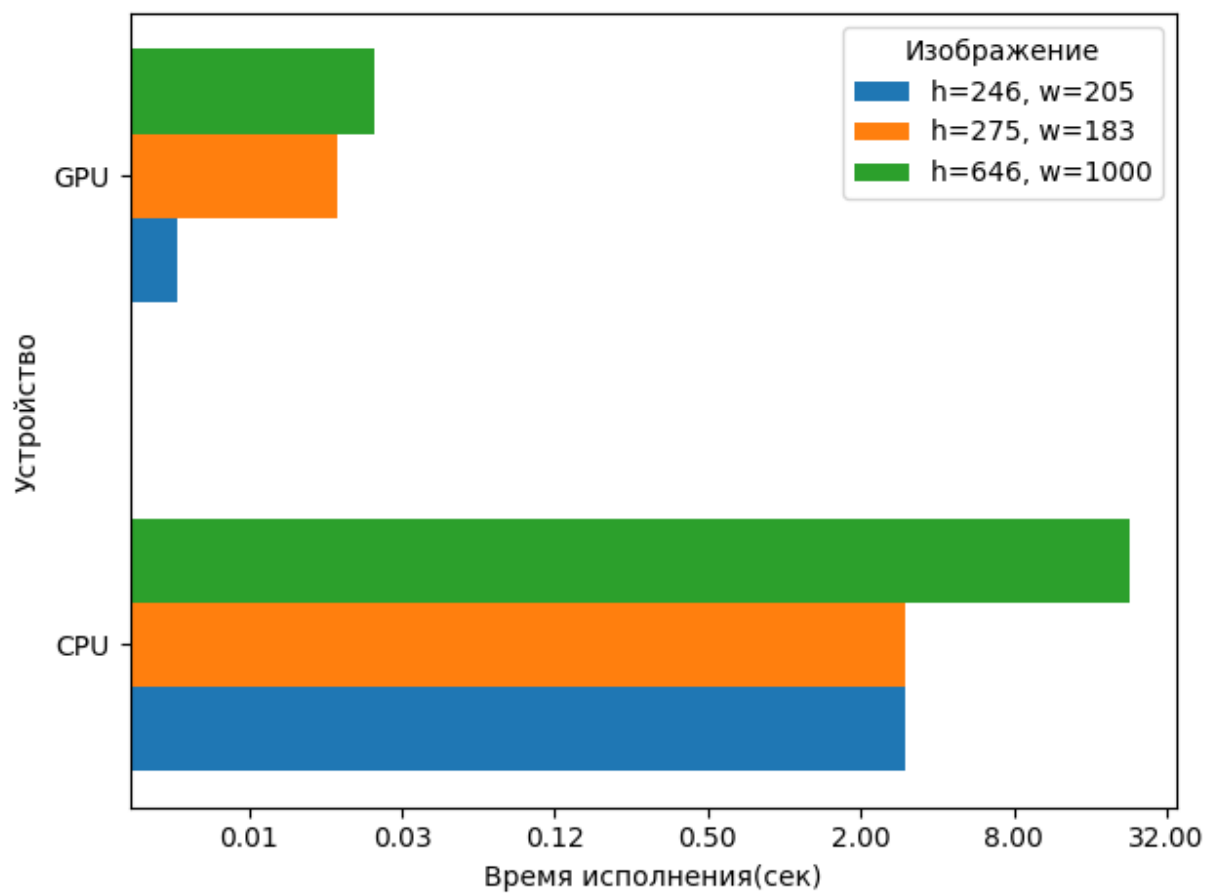


Рис. 1: Применение фильтров 5×5 и 3×3 к изображению.

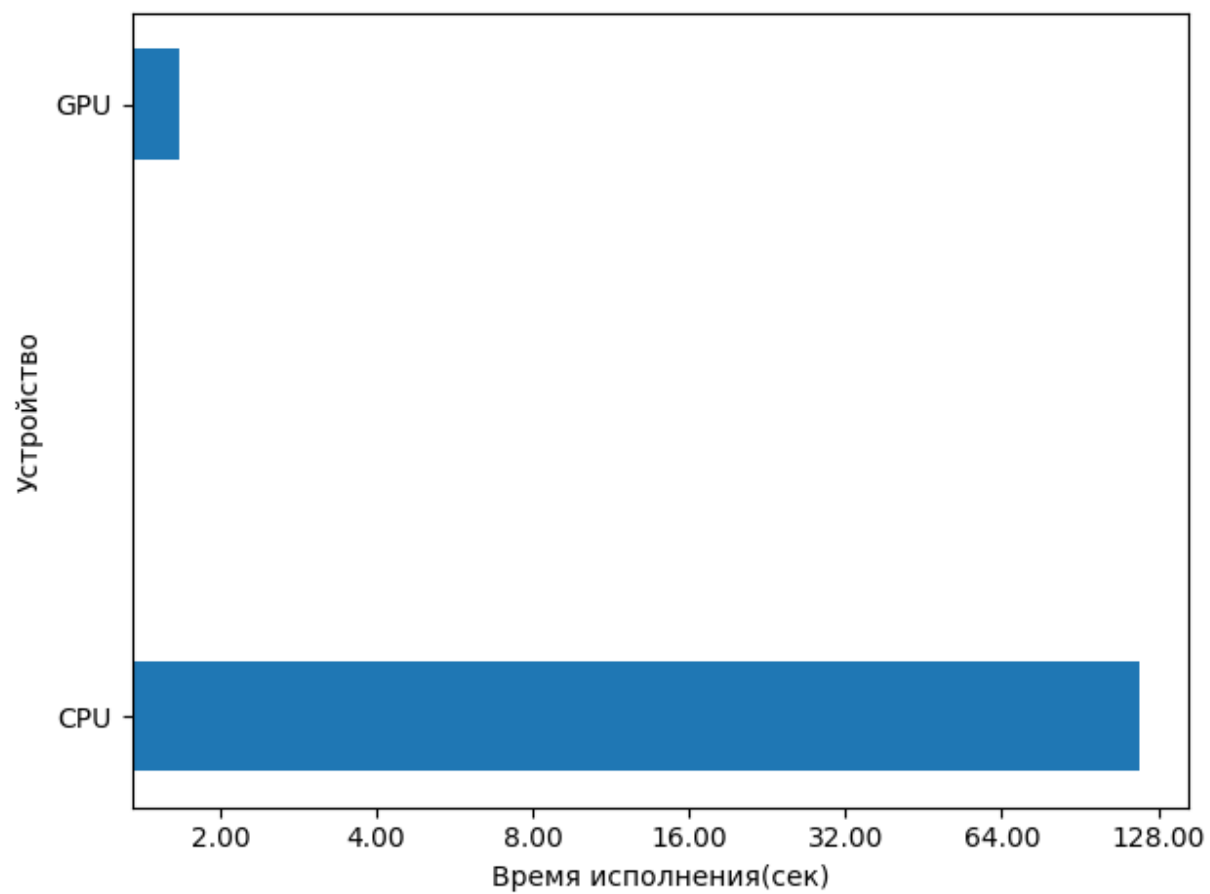


Рис. 2: Применение фильтра 101×101 к изображению 646×1000 .

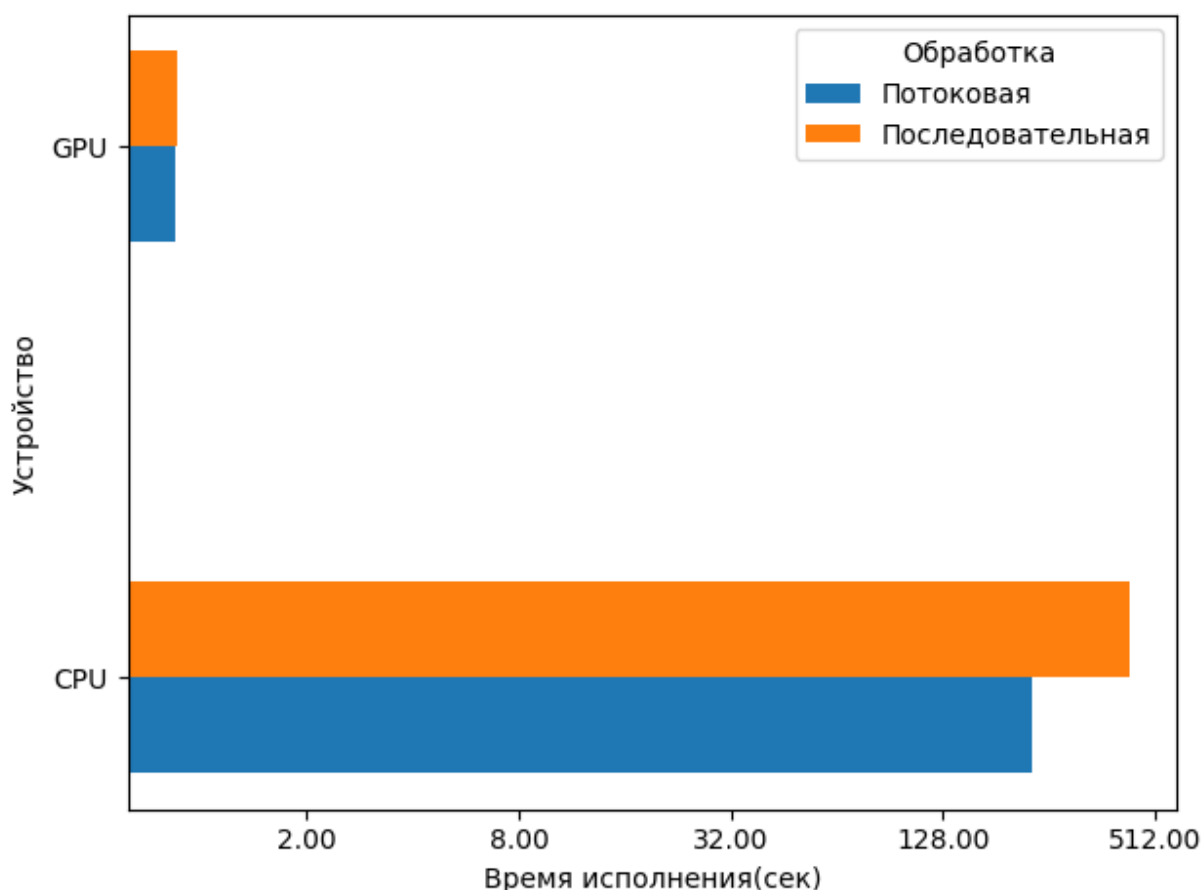


Рис. 3: Применение 5 фильтров 5×5 к папке из 50 изображений.

фильтров к той же директории, но без использования потоковой обработки, вычисления для каждого изображения происходили по очереди. Результаты проведения эксперимента представлены на рисунке 3.

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.

При увеличении матрицы фильтра до размеров 101×101 , разница во времени обработки одного изображения на CPU и GPU становится не такой значительной. Это связано с транспортировкой данных между графическим и центральным процессорами. Однако, расчеты на видеокарте все еще требуют почти в 100 раз меньше времени.

При работе с директорией потоковая обработка дает преимущество в скорости для обеих реализаций. Особенно заметна разница в реализации на CPU — почти в 2 раза. Что же касается разницы во времени работы между графическим и центральным процессорами, то тут она увеличивается по сравнению с предыдущим экспериментом. Стоит заметить, что, во-первых, часть измеряемого времени занимает чтение и запись изображений, а, во-вторых, обмен данными между видеокартой и процессором требует еще больше времени, чем в прошлых экспериментах. Однако, реализация на GPU требует почти в 270 раз меньше времени, чем на CPU.

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/>