Санкт-Петербургский государственный университет

Порсев Денис Витальевич

Анализ производительности и масштабируемости приложения обработки изображений

26 октября 2021 г.

Санкт-Петербург
2021

Содержание

Введение

Применение фильтров позволяет накладывать различные эффекты на изображения. При этом при увеличении размеров фильтра и изображения возникают вычислительные сложности, исследование которых позволяет оценить производительность различных вычислительных устройств в контексте примитивных операций линейной алгебры.

В качестве домашней работы автором было реализовано приложение с графическим интерфейсом позволяющее применять фильтры к изображениям формата BMP. В нем присутствуют следующие функции: просмотр изображения и информации о нем, задание матричного фильтра, применение фильтра как один раз, так и несколько раз последовательно, отображение изображения до и после применения фильтра, а также возможность сохранить результат и применить фильтр сразу к нескольким изображениям, хранящимся в отдельной папке. Помимо этого, в реализации предусматривается возможность применять фильтры используя как центральный процессор, так и видеокарту.

В данной работе будет проведен анализ производительности и масштабируемости реализованного приложения. А именно, оценена возможность добавления новой функциональности в разработанный продукт, а также оценена скорость применения матричных фильтров с использованием различных вычислительных устройств.

1. Детали реализации

В этой главе будут описаны основные инструменты, использованные при создании приложения, а также выделены основные особенности архитектуры приложения.

1.1 Инструменты

Приложение реализовано на языке Си. Самая известная библиотека для создания пользовательского интерфейса на этом языке — GTK. А именно, была выбрана третья версия этой библиотеки — GTK3[2], предоставляющая множество примитивов для построения пользовательского интерфейса. Это не последняя стабильная версия библиотеки, так как с начала 2021 года стала доступна версия GTK4. Однако основные библиотеки, которые используются вместе с GTK пока не совместимы с новой версией, поэтому работа велась с версией GTK3, которая продолжает поддерживаться.

Помимо пользовательского интерфейса приложение состоит также из вычислительной части, отвечающей за применение фильтров к изображениям. Вычисления производятся как на центральном процессоре, так и на видеокарте.

Для распараллеливания вычислений на процессоре используется OpenMP[4] - стандарт для распараллеливания программ на языке C, C++, Fortran. OpenMP предоставляет реализацию многопоточных параллельных вычислений, доступ к которым происходит через директивы компилятора.

На видеокарте вычисления реализованы с помощью библиотеки OpenCL[1], которая позволяет исполнять программы на графическом процессоре, обеспечивая при этом параллелизм на уровне данных.

1.2 Особенности архитектуры

Архитектуру приложения можно разделить на две части: интерфейс и вычислительную часть приложения.

1.2.1 Интерфейс

Интерфейс приложения представляет собой одно окно, которое наполнено различными компонентами. Их инициализация происходит после того, как пользователь выберет изображение для просмотра. Компоненты помещаются в Gtk grid, который отвечает за их расположение относительно друг друга внутри окна. При этом каждый компонент занимается отдельной функцией приложения.

Так, например, панель с фильтрами (filterspanel) служит пользователю инструментом выбора и задания фильтров, контейнер изображения (imagebox) ответственен за показ изображения и выравнивание его внутри окна с сохранением пропорций изначальной картинки, контейнер для информации об изображении (listbox) хранит в себе

информацию о картинке и представляет размер и другие характеристики изображения в удобном формате.

Таким образом, для добавления новых функциональных частей в интерфейс достаточно создать новый компонент и разместить его в Gtk grid относительно других компонентов.

Интерфейс взаимодействует с вычислительной частью приложения через функции, которые предоставляет ему файл *filters.c*. А именно, пользуется вызовом двух функций *apply_filters_service()* и *apply_and_save_filters_service()*, в которые подаются данные изображения и фильтра, который будет применен. Данные о фильтре, такие как ядро фильтра, размер ядра и веса коррекции фильтра собирает панель с фильтрами и отправляет одним из параметров в выше перечисленные функции. Получается, что общение происходит только между основным компонентом интерфейса — filterspanel и доступными функциями файла *filters.c*. При расширении функций приложения подразумевается, что будут вызываться именно эти функции, а не напрямую реализации алгоритмов применения фильтра.

1.2.2 Вычислительная часть

Вычислительная часть приложения реализует один и тот же алгоритм наложения матричного фильтра[6], используя разные вычислительные устройства.

Фильтр задается ядром, которое представляет собой квадратную матрицу нечетной размерности, сумма элементов которой чаще всего равна единице, что позволяет сохранить яркость изначального изображения. Само вычисление фильтра представляет собой итерацию через два вложенных двойных цикла, которые, проходя через каждый пиксель изображения и через каждый элемент ядра матрицы, выбирают квадратную подматрицу с размерностью ядра в изображении, накладывают центр ядра на текущий пиксель и получают новую матрицу поэлементным умножением. Сумма элементов полученной матрицы записывается в пиксель нового изображения. Пиксели находящиеся за пределами изображения оборачиваются вокруг.

Сам алгоритм представлен в виде псевдокода в листинге 1.

Было реализовано три версии алгоритма наложения матричного фильтра: на процессоре (CPU), на процессоре с применением параллельных вычислений (CPU parallel) и на видеокарте (GPU). Реализация на процессоре с применением параллельных вычислений использует директиву компилятора (`# pragma omp parallel for`) для распараллеливания внешнего двойного цикла, что позволяет задействовать все ядра процессора. Реализация на GPU основана на использовании библиотеки OpenCL. Для этого написана программа, работающая на процессоре (хосте), которая компилирует код исполняемый на видеокарте, а также пересылает данные из оперативной памяти в видеопамять и обратно, контролируя процесс исполнения программы на графическом устройстве.

Таким образом, GPU-алгоритм использует большое количество ядер видеокарты для ускорения вычислений, но в обмен возникают затраты на выполнение некоторых подготовительных операции на процессоре.

Все алгоритмы реализованы в отдельном файле и отделены от логики интерфейса приложения. Для того, чтобы расширить вычисления новым алгоритмом, достаточно написать аналогичную реализацию и поместить в файл с другими алгоритмами. После чего новый алгоритм можно будет вызвать из одной из функций, которые отвечают за взаимодействие с интерфейсом.

Algorithm 1 Наложение матричного фильтра (псевдокод)

```
# Input: image - image pixel buffer,
         filter data - kernel (k) and biases (f, b)
# Output: result - result image pixel buffer

# Get image properties such as:
# - channels
# - width
# - height

FOR x TO width
FOR y TO height
# multiply every value of the filter with corresponding image pixel
    FOR filterY TO filterHeight
    FOR filterX TO filterWidth
        imageX = (x - filterWidth / 2 + filterX + width) % width
        imageY = (y - filterHeight / 2 + filterY + height) % height
        FOREACH channel IN channels
            channel = image[y*width+x].channel * k[filterX][filterY]
# truncate values smaller than zero and larger than 255
    FOREACH channel IN channels
        result[y*width+x].channel = MIN(MAX((f*channel+b), 0), 255)
```

Отдельно стоит поговорить о реализации потоковой обработки изображений, благодаря которой ускоряется процесс применения фильтров ко всем изображениям выбранной папки.

Для этого в приложении создается три потока: поток чтения данных изображения, поток применения матричного фильтра и поток сохранения результатов преобразова-

ния в память. К каждому потоку прикрепляется устройство синхронизации — mailbox, которое отвечает за межпроцессное взаимодействие. Mailbox[3] используется множеством потоков в качестве места хранения и получения сообщений (messages), в которых содержатся данные картинок. Сам он реализован с помощью семафоров и по сути является буффером для сообщений, которые посылаются в прикрепленный к нему поток. Каждый поток после этого может передавать обработанные данные в другие потоки, занятые следующей операцией.

Благодаря этому увеличивается производительность расчетов, так как в отличии от однопоточной версии, операции чтения и записи данных происходят независимо от самих вычислений, что позволяет работать параллельно с несколькими картинками.

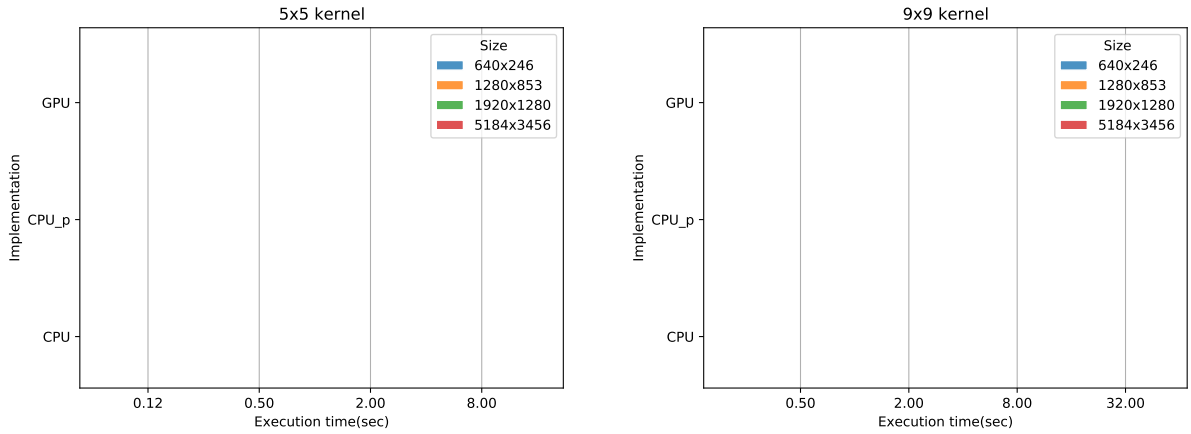
2. Проведение эксперимента

Измерения производились на компьютере со следующими характеристиками: видеокарта AMD Radeon 8650G, процессор AMD A10-5757M 2.5 GHz, 8 Гб оперативной памяти DDR3 под управлением операционной системы Ubuntu 20.04.3 LTS.

В качестве исходных данных были использованы изображения BMP разных размеров, взятых с сайта FileSamples[5]. А именно, четыре изображения с размерами 640×426, 1280×853, 1920×1280 и 5184×3456 пикселей соответственно, с размерами файлов — 798 Кб, 3.12 Мб, 7.03 Мб и 51.26 Мб. При этом все эти изображения являются одной и той же картинкой разных размеров.

Эксперимент был поставлен следующим образом. Пользователь запускал графический интерфейс приложения, выбирал картинку соответствующего размера, после чего в поле с количеством итераций фильтра выставлял значение 10 и применял фильтры с размерами матриц 5×5 и 9×9 соответственно. Перед этим в код программы в функции применения фильтра были внесены изменения, проводящие замеры на однопоточном процессоре, процессоре, использующем многопоточность, и видеокарте. Для измерения времени исполнения использовался заголовочный файл time.h стандартной библиотеки языка Си, измерялось реальное время, а не процессорное время вычисления, так как в параллельных программах значение процессорного времени не отражает того, как долго исполнялась программа. Замеры записывались в секундах.

Таким образом, применялось 10 итераций наложения фильтров с размерами 5×5 и 9×9 для каждого изображения исходных данных. Замерялось время работы алгоритма без учета времени загрузки изображения в память. Полученные временные значения записывались в файл. На рисунке 1 представлены результаты проведенных измерений.



(a) Время исполнения на ядре 5x5

(b) Время исполнения на ядре 9x9

Рис. 1: Результаты измерений первого эксперимента.
Среднее 10 замеров.

Рисунок 1 иллюстрирует среднее время наложения фильтра в различных реализа-

циях в виде гистограммы. Такое представление удобно использовать, в случае когда данные можно сгруппировать по категориям. В данном случае категориями являются различные версии алгоритма. На временной оси используется логарифмический масштаб. Это связано с тем, что на однопоточном процессоре алгоритм исполняется сильно дольше, чем в остальных реализациях.

Далее было решено оценить влияние размера фильтра на производительность алгоритма с одинаковыми исходными данными изображения. Для этого была выбрана одна картинка с размером 1280×853, добавлены новые фильтры с размерами 3×3 и 15×15, и тем же образом запущен эксперимент на 10 итерации с каждым фильтром для данной картинки. Результаты представлены на рисунке 2.

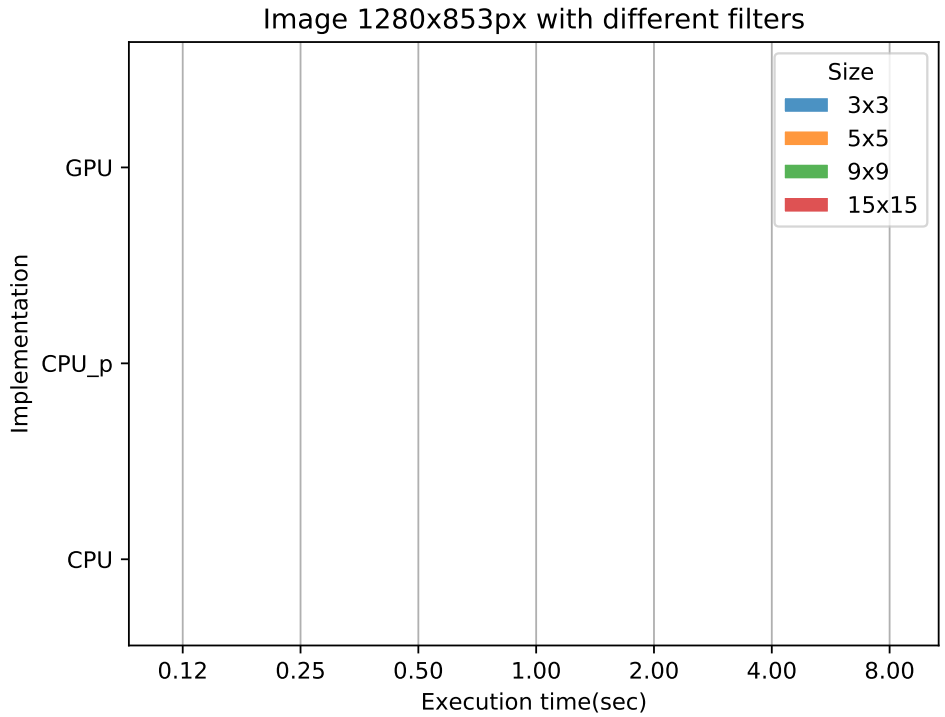


Рис. 2: Первый эксперимент с разными фильтрами.
Среднее 10 замеров.

После чего был проведен эксперимент, связанный с применением потоковой обработки к папке с изображениями, следующим образом. В папку было загружено 10 изображений размера 5184×3456 пикселей и 51.26 Мб каждое. Также в код программы было внесено изменение в части вызова сервиса, который отвечал за применение фильтров к папке изображением. Замерялось время чтения, обработки и записи всех изображений. Применялся фильтр размера 5×5, его параллельная версия на CPU. Всего было выполнено 5 итераций на многопоточном и последовательном вариантах исполнения. Результаты эксперимента приведены на рисунке 3.

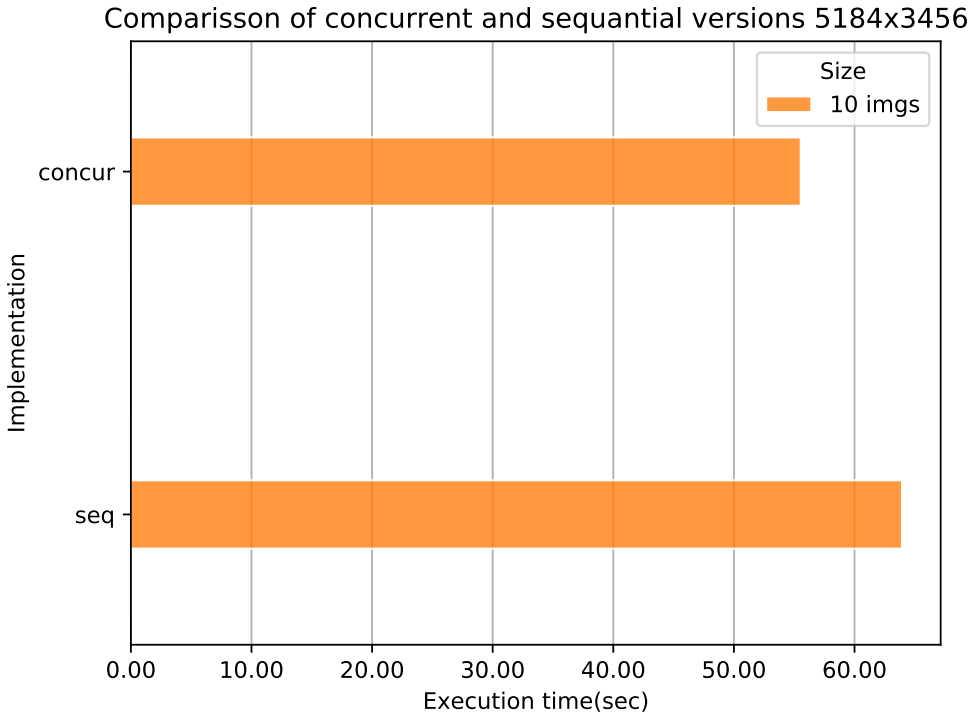


Рис. 3: Результаты измерений второго эксперимента.
Среднее 5 замеров.

На этом рисунке concur обозначает многопоточный вариант, seq — последовательный.

В заключении было проведено сравнение многопоточного и последовательного варианта обработки изображений папки на большем количестве изображений, но меньшего размера. Так, было взято 100 изображений 640×426, размера 798 Кб и также замерено время исполнения для 5 итераций. Было замечено, что с увеличением числа картинок многопоточная версия начинает существеннее опережать последовательный вариант. Поэтому то же самое измерение было также произведено для 10 и 1000 изображений, чтобы опровергнуть или подтвердить эту гипотезу. Результаты представлены на рисунке 4.

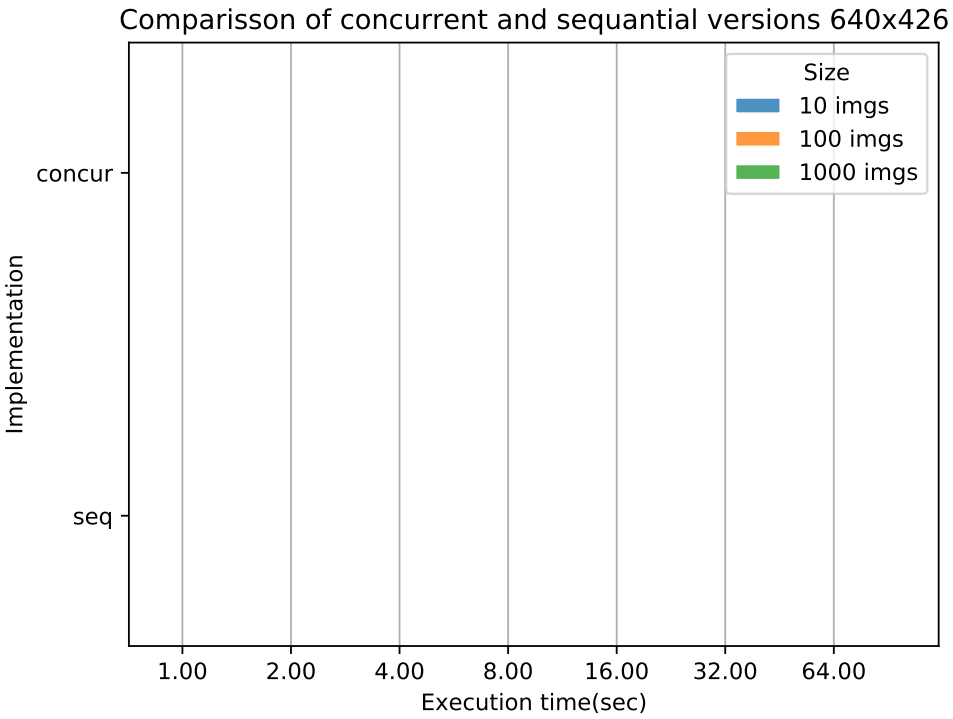


Рис. 4: Результаты измерений второго эксперимента.
Среднее 5 замеров.

Заключение

В результате проведенных экспериментов было получено:

1. Реализация алгоритма на видеокарте оказалось самой эффективной для большинства входных данных в проведенных экспериментах. Выяснилось, что при небольших размерах фильтра и самого изображения реализация на процессоре с многопоточностью является не менее быстрой. По всей видимости, это происходит из-за траснспортировки данных между памятью процессора и видеокарты. Временные затраты на перемещение данных в этом случае нивелируют эффект от более быстрых вычислений на ядрах видеокарты.

2. Важно отметить, что алгоритм на видеокарте показывает меньшую зависимость от размеров входных данных. Так, на первом рисунке видно, что при увеличении размеров изображения, время исполнения растет не так сильно, как в реализациях на процессоре. На втором рисунке можно отметить, что при увеличении размеров фильтра и сохранении размеров изображения эта тенденция также сохраняется.

3. Алгоритм, распаралеленный на процессоре при помощи OpenMP показал производительность в четыре раза выше однопоточного варианта, что кратно числу ядер процессора. Это говорит об эффективной реализации многопоточности в этом стандарте. Однако, не смотря на это, при больших входных данных эта реализация алгоритма существенно медленнее варианта на GPU.

4. Потоковая обработка изображений показала более эффективные результаты, чем последовательная, во всех экспериментах. Более того, была изучена зависимость времени исполнения от числа изображений: при росте числа изображений время исполнения однопоточной версии алгоритма росло практически линейно, так же как и в многопоточной версии.

5. Тем не менее, прирост в производительности многопоточной версии составляет примерно 15%, что является не таким существенным показателем. Это можно объяснить временными затратами на синхронизацию потоков.

Таким образом, был проведен экспериментальный анализ, на основе которого можно сделать вывод о том, что для независимых вычислений матричных операций видеокарта более производительна, чем центральный процессор.

Список литературы

[1] K. O. W. Group. The opencl specification.

[2] GTK. The gtk toolkit v3.0 documentation.

[3] H. C. Lauer. Cs 3013 cs 502 operating systems. wpi, summer 2006.

[4] OpenMP Architecture Review Board. OpenMP application program interface version 3.0, May 2008.

[5] F. Samples. Get file samples to use for testing purposes.

[6] L. Vandevenne. Lode's computer graphics tutorial. image filtering.