Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное
учреждение высшего образования
Национальный исследовательский Нижегородский государственный
университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

Отчет по лабораторной работе

«Линейная фильтрация изображений (горизонтальное разбиение). Ядро Гаусса 3x3»

Выполнил:
студент группы 381706-4
Доброхотов В.Н.

Проверила:
Кустикова В.Д

Нижний Новгород
2020

Содержание

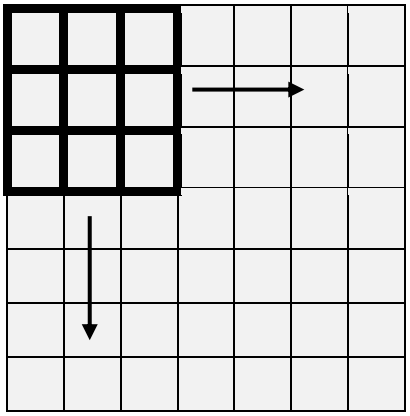
Постановка задачи

Одна из самых важных задач при работе с изображениями связана с их предварительной обработкой - выделением и фильтрацией шума. При этой обработке необходимо обеспечить максимальное сохранение деталей изображения.

Под фильтрацией изображения понимают операцию, имеющую своим результатом изображение того же размера, полученное из исходного по некоторым правилам. В данной лабораторной работе ставится задача реализации линейного фильтра с использованием матрицы свертки Гаусса размерностью 3x3, значения коэффициентов которой заполняются по закону нормального распределения (закону Гаусса):

$$F(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}}$$

Матрица свёртки – это матрица коэффициентов, которая «умножается» на значение пикселей изображения для получения требуемого результата, т. е. свертка – это операция вычисления нового значения выбранного пикселя, учитывающая значения окружающих его пикселей. Для вычисления значения используется матрица, называемая ядром свертки. Обычно ядро свертки является квадратной матрицей n*n, где n — нечетное, однако ничто не мешает сделать матрицу прямоугольной. Во время вычисления нового значения выбранного пикселя ядро свертки как бы «прикладывается» своим центром (именно тут важна нечетность размера матрицы) к данному пикселю. Окружающие пиксели так же накрываются ядром. Далее высчитывается сумма, где слагаемыми являются произведения значений пикселей на значения ячейки ядра, накрывшей данный пиксель. Сумма делится на сумму всех элементов ядра свертки. Полученное значение как раз и является новым значением выбранного пикселя. Если применить свертку Гаусса к каждому пикселю изображения, то в результате получится эффект размытия, т. к. при нормальном распределении значения свертки быстро убывают по мере отдаления от «центра». Таким образом, влияние цветов пикселей окрестности тем больше, чем они ближе расположены к обрабатываемому пикселю, при этом исходный цвет пикселя будет иметь наибольший «вес». Поэтому использование ядра Гаусса позволяет в большей мере сохранять границы и края объектов исходного изображения



Применение фильтра 3x3 к изображению.

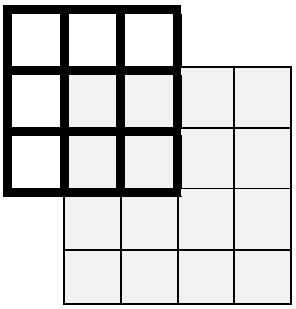
При последовательной обработке каждого пикселя время обработки изображений достаточно велико, что неприемлемо в реальном времени для решения различных прикладных задач. В данной лабораторной работе ставится задача реализации линейного фильтра с ядром Гаусса размерностью 3x3. Необходимо:

- Реализовать последовательный алгоритм линейной фильтрации;
- Реализовать параллельный алгоритм линейной фильтрации (вертикальное разбиение) с использованием технологий OpenMP, TBB, std::threads;
- Обеспечить подтверждение корректности работы.
- Сравнить время выполнения параллельных и последовательного алгоритмов;

Метод решения

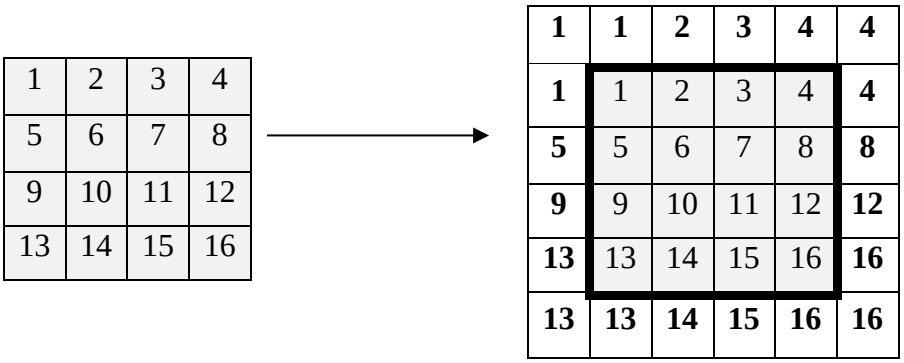
Предполагается, что изображение задается в оттенках серого, т.е. может быть представлено в программе как массив байт. Ядро фильтра представляет собой матрицу размерности 3x3, коэффициенты которой заполняются по нормальному закону (закону Гаусса). Цвет каждого пикселя выходного изображения рассчитывается как линейная комбинация цветов пикселей окрестности размера 3x3 (цвет каждого пикселя умножается на соответствующий коэффициент матрицы свертки).

Стоит упомянуть о случае, который касается обработки границ изображения. У пикселей верхней строки нет соседей сверху, у пикселей нижней строки нет соседей снизу. Аналогичные ограничения имеются для крайних столбцов и «угловых» пикселей.



Проблемы, возникающие при обработке границ изображения.

Существует несколько способов решения данной проблемы. В работе используется метод создания промежуточного изображения. Идея состоит в том, создается временное изображение с размерами (width + 2, height+ 2). В центр изображения копируется входная картинка, а края заполняются крайними пикселями изображения. Размытие применяется к промежуточному буферу, а потом из него извлекается результат. Данный метод не имеет недостатков в качестве, но необходимо производить лишние вычисления.



Создание промежуточного изображения при использовании матрицы свертки 3x3

Для применения фильтра ко всему изображению необходимо передвигать матрицу по всем пикселям. Выходное изображение так же представляет собой массив байт.

Для простоты загрузки и обработки изображений к программе будет подключена библиотека OpenCV.

Последовательный алгоритм

Последовательный алгоритм состоит из четырех простых этапов:

1. Загрузить картинку, если она цветная, то перевести ее сначала в оттенки серого.
2. Создать промежуточное изображение, в котором границы будут дублироваться.
3. Последовательно применить фильтр к каждому пикселю промежуточного изображения.
4. Из промежуточного изображения извлечь результат.

Последовательное применение фильтра к каждому пикселю состоит в следующем:

- Осуществить проход по всему изображению с помощью циклов;
- Для каждого пикселя выполнить:
    - Получить значение цвета пикселя из окрестности;
    - Умножить полученное значение на соответствующий коэффициент матрицы свёртки;
    - Прибавить полученную величину к получаемому значению нового цвета.
    - Проверить не вышел ли пиксель за границы цвета, если вышел за максимум, то присвоить значение 255, если за минимум - 0;
    - Вернуть значение нового пикселя.

Вышеперечисленные шаги для каждого пикселя осуществляются до тех пор, пока не будут пройдены все пиксели окрестности.

Схема распараллеливания

Для дальнейших рассуждений примем следующие обозначения:

N – высота изображения;

M – ширина изображения

num_threads – количество потоков

id – номер работающего потока

Схема распараллеливания линейной фильтрации Гаусса вертикальным разбиением с использованием различных библиотек будет схожа. Поскольку пиксели изображения хранятся в виде массива предлагается разбить этот массив по столбцам на *M/num_thread* частей, и на каждом потоке запустить последовательный алгоритм фильтрации Гаусса.

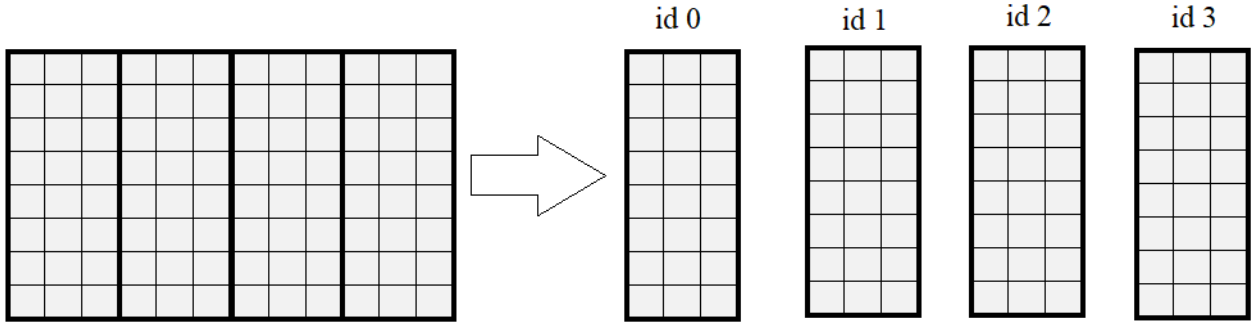
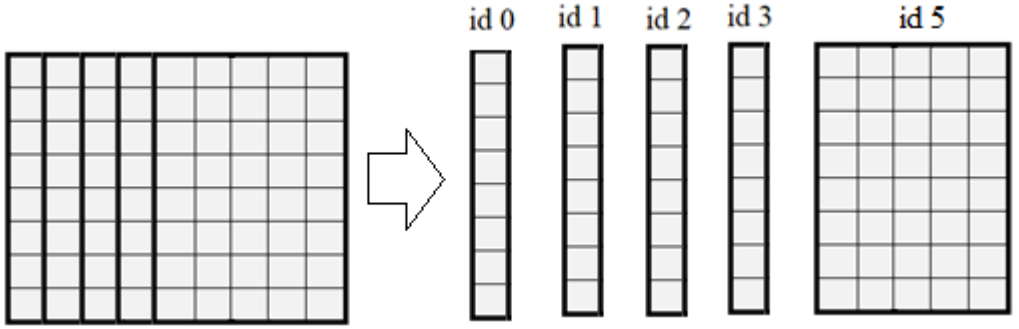


Схема распараллеливания

Так как в OpenMP и TBB реализовано распараллеливание итераций, то можно не беспокоится, что какие-то части изображения не будут обработаны. В std::threads нету распараллеливания цикла, поэтому надо заранее определить какие части изображения каждый поток должен обработать. Можно на каждый поток отправлять *col/num_thread*, а оставшуюся часть отправить в последний поток. Но при таком подходе существует ситуация, когда ему может достаться слишком много вычислений, а, следовательно, программа может сильно замедлится.



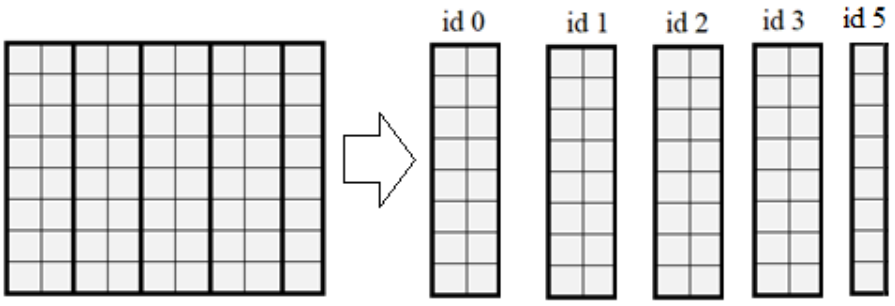
9 столбцов 5 потоков

На каждый процесс отправляется 9 div 5 = 1 столбец

Остальные 4 столбца будут обрабатываться в последнем потоке и из-за чего он будет нагружен, и программа замедлится

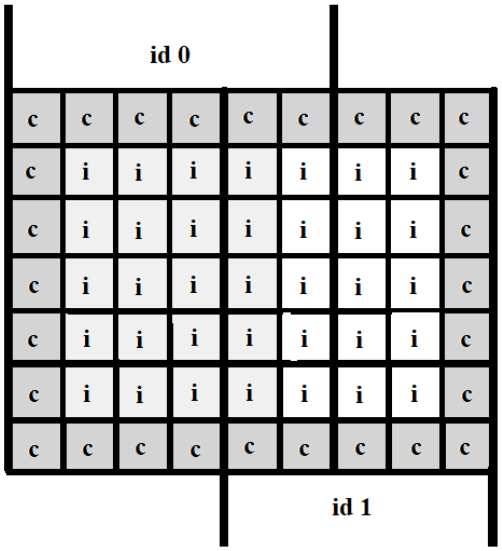
Пример неравномерного распределение вычислений

Чтобы избежать этого предлагается не вместившуюся часть равномерно распределить между потоками. После этого нагрузка на каждый поток будет равномерно распределена.



Пример равномерного распределение вычислений

Для того чтобы бы обработались все пиксели обработка будет проводиться с использованием промежуточного изображения, в котором дублируются границы исходной картинки. Каждый поток будет захватывать не просто свою часть, а часть исходных пикселей с дублирующего изображения, две дополнительные строчки и два дополнительных столбца с некоторым сдвигом назад. Таким образом, получится некое подобие перекрытия и обработаются все пиксели:



Первые четыре столбца исходного изображения будут обрабатываться на первом потоке, остальные три на втором.

Пример работы параллельного алгоритма с использованием std::threads на изображении 7x5 и 2 потоками

Стоит отметить, что эти дополнительные столбцы каждый поток использует только для вычисления новых цветов пикселей граничных столбцов своей части изображения. Вычисление цветов пикселей этих дополнительных столбцов осуществляется теми потоками, частями изображения которых они являются.

Описание программной реализации

Руководство пользователя.

Для запуска используется обычный способ, характерный для всех программ, написанных с использованием библиотек. В папке с программой также должны находиться 2 dll файла: opencv420.dll и tbb.dll. Команда для запуска через командную строку имеет вид:

```
<название_программы> <путь_до_картинки> <число_потоков>
```

Если число не будет указано, то программа запустится с максимально возможным количеством потоков.

Программа выводит количество потоков, результаты проверки каждого из алгоритмов, время работы каждого алгоритма, а также исходное изображение и изображения, получившиеся после применения алгоритмов.

```
C:\Users\chele\Downloads>main.exe sm.jpg
image to grey color
Threads: 6
Check algorithms openMP: 1
Check algorithms TBB: 1
Check algorithms std::thread: 1
Time seq:  0.0051856
Time openMP:  0.0056781
Time TBB:  0.0030369
Time std::thread:  0.0023052
```

Руководство программиста

Программа, реализующая поставленную задачу, состоит из следующих функций:

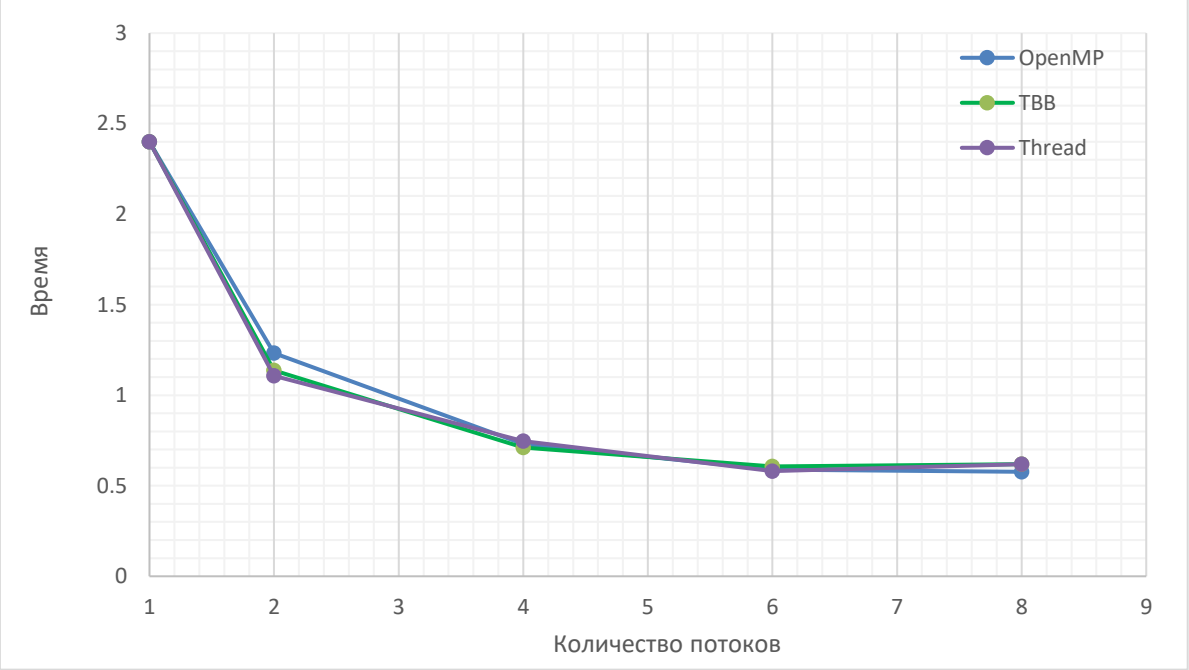
- **createKernel –** функция, генерирующая ядро Гаусса для фильтрации изображения;
- **printKernel** – печатает ядро на экран;
- **Clamp –** функция, проверяющая не вышел ли пиксель за цветовую гамму;
- **duplicateBorder –** функция, создающая изображение с дублирующими границами;
- **gaussSeq –** функция, последовательной фильтрации изображения;
- **gaussOpenMP –** функция, параллельной фильтрации изображения с использованием OpenMP;
- **gaussTBB –** функция, параллельной фильтрации изображения с использованием TBB;
- **funcForTBB –** функция нужная для правильной работы gaussTBB.Она будет выполнять фильтрацию изображения части изображения на каждом потоке;
- **gaussThread –** функция, параллельной фильтрации изображения с использованием std::thread;
- **gaussForThread –** функция нужная для правильной работы gaussThreads. Она будет выполнять фильтрацию изображения части изображения на каждом потоке;
- **checkImages –** функция проверки корректности работы алгоритмов;
- **main –** главная функция, в которой считывается изображение и осуществляется последовательной вызов всех функций фильтрации изображения и замер работы каждого из алгоритмов.

Подтверждение корректности

Подтвержение корректности можно осуществить путем простого сравнения двух картинок на глаз, но из-за маленького ядра на больших картинках будет сложно заметить разницу. Поэтому для подтверждения корректности в программе реализована функция автоматической проверки *checkImages()*, которая принимает на вход результат работы двух алгоритмов: последовательного и параллельного. Функция возвращает 0, если расходятся значения каких-то элементов. Функция возвращает 1, если результаты работы двух алгоритмов совпадают.

Результаты экспериментов

Эксперимент проводился на изображении размером 10453x7466. Результаты экспериментов представлены на графике ниже:



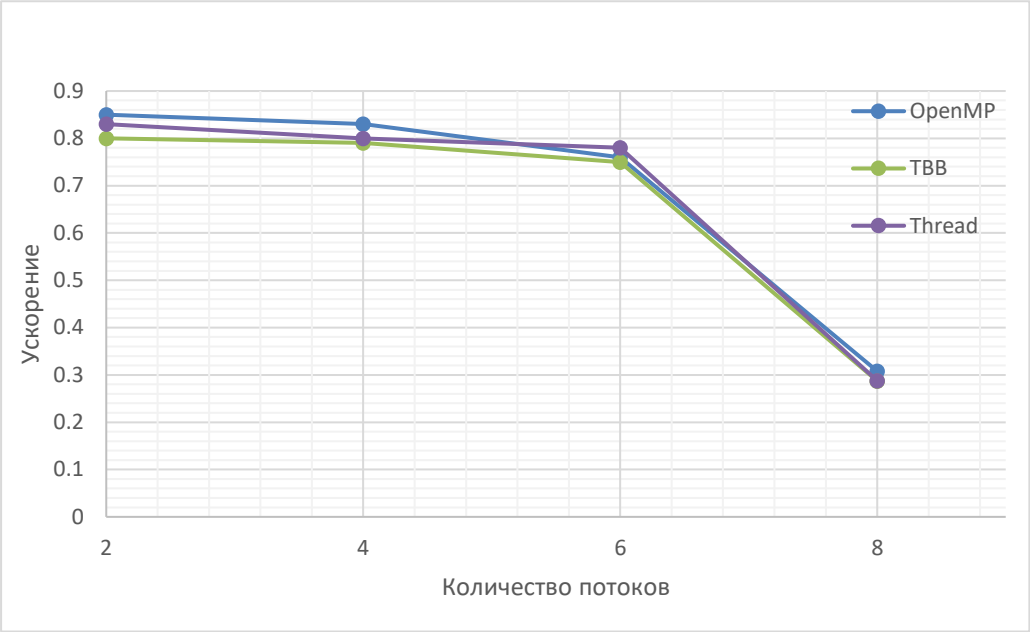
По графику явно видно, что при увеличении числа потоков время работы программы уменьшается, но после 6 потоков особого ускорения не наблюдается. Это связано с техническими характеристиками машины, на которой запускается тесты

Также можно заметить, что обе технологии дают примерно одинаковое ускорение. Для большей наглядности можно использовать в качестве метрики оценивания эффективность параллельного алгоритма. Она рассчитывается по следующей формуле:

$$E = \frac{S}{p}$$

где S - ускорение работы программы на P процессорах.

Идеальной эффективностью считается единица, поэтому разумно стремиться к данному значению. Данная программа близка к данному значению при числе процессов равному или меньшему шести. После, эффективность уменьшается и в результате достигнет нулевое значение.



Идеальной эффективностью считается единица, поэтому разумно стремиться к данному значению. Данная программа близка к данному значению при числе процессов равному или меньшему шести. После, эффективность уменьшается и в итоге достигнет нулевого значения.

Эксперименты проводились на ПК со следующими характеристиками:

- Процессор AMD FX-6300 3.5GHz 6 ядер
- Оперативная память: 8,00ГБ DDR3
- ОС: Windows 10 Pro, сборка 18362.476

Заключение

В данной лабораторной работе были реализованы несколько параллельных алгоритмов фильтрации изображений с использованием различных технологий. По представленным выше результатам экспериментов видно, что параллельная программа работает быстрее последовательной. Однако ускорение не является таким уж и большим. Это можно объяснить тем, что дублирование границ происходит лишь на одном процессе, сама фильтрация осуществляется каждым процессом последовательно и занимает довольно много времени, а тестирование происходило не на самом мощном железе.

Приложение.

Код находится в репозитории https://github.com/Dybbor/parallel-programming-1