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

2019

**Содержание**

[Постановка задачи 3](#_Toc27945404)

[Метод решения 4](#_Toc27945405)

[Последовательный алгоритм 4](#_Toc27945406)

[Схема распараллеливания 6](#_Toc27945407)

[Описание программной реализации 8](#_Toc27945408)

[Руководство пользователя. 8](#_Toc27945409)

[Руководство программиста 8](#_Toc27945410)

[Подтверждение корректности 9](#_Toc27945411)

[Результаты экспериментов 10](#_Toc27945412)

[Заключение 12](#_Toc27945413)

[Приложение. 13](#_Toc27945414)

# 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 3х3, значения коэффициентов которой заполняются по закону нормального распределения (закону Гаусса):

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 3х3 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 3х3. Необходимо:

* Реализовать последовательный алгоритм фильтрации;
* Реализовать параллельный алгоритм фильтрации, используя горизонтальное разбиение;
* Обеспечить подтверждение корректности работы.
* Сравнить время выполнения параллельного и последовательного алгоритма;

# 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 3х3, коэффициенты которой заполняются по нормальному закону (закону Гаусса). Цвет каждого пикселя выходного изображения рассчитывается как линейная комбинация цветов пикселей окрестности размера 3х3 (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 \* kernelSize / 2, height + 2 \* kernelSize / 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.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **1** | **1** | **2** | **3** | **4** | **4** |
| **1** | 1 | 2 | 3 | 4 | **4** |
| **5** | 5 | 6 | 7 | 8 | **8** |
| **9** | 9 | 10 | 11 | 12 | **12** |
| **13** | 13 | 14 | 15 | 16 | **16** |
| **13** | **13** | **14** | **15** | **16** | **16** |

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 |

Создание промежуточного изображения при использовании матрицы свертки 3х3

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

Еще в программу была добавлена возможность загрузки настоящих изображений с помощью библиотеки 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 цикла;
* 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.

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

# Схема распараллеливания

Для дальнейших рассуждений примем следующие обозначения:

– высота изображения;

– ширина изображения;

– число процессов;

– ранг процесса;

В данной лабораторной работе предполагается горизонтальное разделение исходного изображения. Каждый процесс будет обрабатывать свою часть исходного изображения. Беря во внимание введенные выше обозначения, получим, что каждый процесс будет обрабатывать блок состоящий из строк. В случае если число строк не кратно числу процессов, тогда первые процессов (т.е. процессы с рангами от до ) получают блок на одну строку больше.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

N=9

Size=4

Из-за того, что число строк не кратно числу процессов, нулевой процесс будет обрабатывать на одну строчку больше.

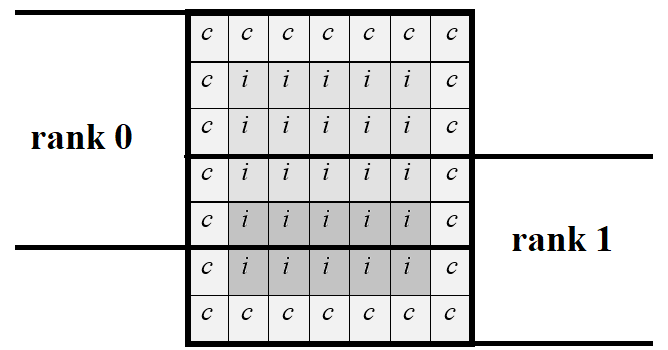
rank 0

rank 1

rank 2

rank 3

Чтобы не возникали проблемы с обработкой границ нужно отсылать на каждый процесс блоки данных не с исходного изображения, а с промежуточного, в котором дублируются границы. Этот блок будет состоять из блока исходного изображения и двух дополнительных столбцов и строчек, которые будут являться границами блока исходного изображения. То есть каждый блок будет состоять изстрок и столбцов. Блоки будут передаваться с некоторым сдвигом назад, чтобы обработать все пиксели исходного изображения. Таким образом получится некое подобие перекрытия:



*Пример работы параллельного алгоритма на изображении 5х5 и 2 процессами*

Первые 3 строчки исходного изображения будут обрабатываться на нулевом процессе, остальные две на первом процессе.

с - пиксели копии границ, i – пиксели исходного изображения

Стоит отметить, что эти дополнительные строки каждый процесс использует только для вычисления новых цветов пикселей граничных строк своей части изображения. Вычисление цветов пикселей этих дополнительных строк осуществляется теми процессами, частями изображения которых они являются.

Вычисление новых цветов на каждом процессе происходит по описанному в предыдущем разделе алгоритму. После того как каждый процесс обработает свою часть изображения, осуществляется слияние этих частей в единое выходное изображение.

# Описание программной реализации

## Руководство пользователя.

Для запуска используется обычный способ, характерный для всех программ, написанных с использованием технологии MPI. Картинки с которыми будет производится работа следует разместить в папке Picture. Команда для запуска через командную строку имеет вид:

path/to/mpiexec.exe –n <число\_процессов> <название\_программы> <name\_image>

Программа выводит размеры изображения, время работы последовательного и параллельного алгоритма, результат сравнения их работ, а также исходное изображение, изображение с дублированными границами, и изображения, получившиеся после применения алгоритмов.

Пример работы запуска программы

## Руководство программиста

Программа, реализующая поставленную задачу, состоит из следующих функций:

* createKernel – функция, которая создает матрицу свертки для фильтра, основываясь на распределении Гаусса ;
* duplicateBorder – функция, которая возвращает исходную картинку, но с дублированием границ;
* processImage – функция, в которой с помощью созданной матрицы свертки организована обработка изображений;
* checkData – функция, которая сравнивает результат работы двух алгоритмов: последовательного и параллельного;
* main – основная функция, в которой происходит загрузка изображения и его дальнейшая обработка.

# Подтверждение корректности

Для подтверждения корректности в программе реализована функция автоматической проверки *checkData()*, 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. Если подтверждается корректность программы, то выводится сообщение «The algorithm is correct.».

# Результаты экспериментов

Эксперименты проводились на ПК со следующими характеристиками:

* Процессор Intel® Core™ i5 CPU 650 3.20GHz 3.19GHz
* Оперативная память: 8,00ГБ DDR3
* ОС: Windows 10 Pro, сборка 18362.476

Рассмотрим приведённую ниже таблицу для оценки результатов:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 5000x5000 | 10000x10000 | 15000x15000 |
| Последовательный алгоритм | 0.945575 | 3.76993 | 9.0544 |
| Параллельный алгоритм на 2 процессах | | | |
| Время | 0.584624 | 2.33606 | 5.35118 |
| Ускорение | 1,617407 | 1,613798 | 1,692038 |
| Параллельный алгоритм на 4 процессах | | | |
| Время | 0.444597 | 1.76712 | 4.08725 |
| Ускорение | 2,126813 | 2,133375 | 2,215279 |
| Параллельный алгоритм на 6 процессах | | | |
| Время | 0.460525 | 1.786900 | 4.10334 |
| Ускорение | 2,053254 | 2,10976 | 2,206593 |
| Параллельный алгоритм на 8 процессах | | | |
| Время | 0.453678 | 1.79811 | 4.02696 |
| Ускорение | 2,084242 | 2,096607 | 2,248445 |
| Параллельный алгоритм на 10 процессах | | | |
| Время | 0.476257 | 1.79411 | 4.15145 |
| Ускорение | 1,985430 | 2,101281 | 2,241288 |

Результаты экспериментов представлены ниже.

# Заключение

B данной лабораторной работе был разработан параллельный алгоритм фильтрации изображений. 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, и сама фильтрация осуществляется каждым процессом последовательно и занимает довольно много времени.

# Приложение.

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