МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего образования

**«Национальный исследовательский Нижегородский
государственный университет им. Н.И. Лобачевского»**
(ННГУ)

Институт информационных технологий, математики и механики

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

на тему:

«Повышение контраста полутонового изображения посредством линейной растяжки гистограммы.»

Выполнил:
студент группы 3821Б1ФИ3
_____ Кузнецов А.А.

Преподаватель:
_____ Нестеров А.Ю.

Нижний Новгород
2023 г.

Введение

Один из наиболее распространенных дефектов фотографических, сканерных и телевизионных изображений – слабый контраст. Дефект во многом обусловлен ограниченностью диапазона воспроизводимых яркостей. Под контрастом понимается разность максимального и минимального значений яркости. Контрастность изображения можно повысить за счет изменения яркости каждого элемента изображения и увеличения диапазона яркостей. Существует несколько методов, основанных на вычислении гистограммы. В данной лабораторной работе будет рассматриваться алгоритм **линейного растяжения гистограммы**.

Постановка задачи

Описываемая работа содержала следующие задачи:

1. Изучить алгоритм.
2. Написать последовательную (однопроцессорную) версию алгоритма.
3. Написать параллельную (многопроцессорную) версию алгоритма используя возможности MPI[1].
4. Написать юнит тесты для проверки корректности написанного алгоритма.
5. Сформулировать и обосновать вывод (отчет).

Дополнительные требования к решению задачи:

1. В результате должно быть минимум 4 файла (main.cpp, CMakeLists.txt, .cpp, .h).
2. Все файлы с кодом, должны соответсвовать стилю Google style.

Технические требования:

1. Используемый ЯП - C++.
2. Сборка должна проводится с помощью CMake.
3. Используемый фреймворк для тестирования - Google Test.

[1]Message Passing Interface (MPI): `https://ru.wikipedia.org/wiki/Message_Passing_Interface`

Описание алгоритма

Задача линейного растяжения гистограммы связана с улучшением согласования динамического диапазона изображения и экрана, на котором выполняется визуализация. Если для цифрового представления каждого пикселя изображения отводится 1 байт (8 бит) запоминающего устройства, то входное или выходное изображения могут принимать одно из 256 значений. Чаще всего для отображения используется диапазон от 0 до 255. Пусть минимальная и максимальная яркости исходного изображения равны x_{min} и x_{max} соответственно. Если $x_{min} > 0$ и $x_{max} < 255$, т. е. динамический диапазон узок, изображение выглядит серым, малоконтрастным.

При линейном растяжении гистограммы изображений используется преобразование яркости типа

$$y = ax + b \tag{1}$$

где a и b определяются желаемыми значениями минимальной и максимальной яркости результирующего изображения, обычно 0 и 255. С учетом этого преобразование яркости принимает вид

$$y = \frac{x - x_{min}}{x_{max} - x_{min}}(y_{max} - y_{min}) + y_{min} \tag{2}$$

Функция и пример линейного растяжения гистограммы изображения представлены на рис. 1 и 2 соответственно

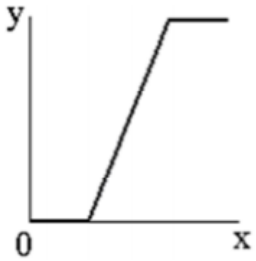

Рис 1: Функция линейного контрастирования изображения

Рис 2: Пример линейного растяжения гистограммы

Описание схемы распараллеливания

На вход подается одномерный массив пикселей размера $m \cdot n$. Благодаря этому массив хранится линейно в оперативной памяти, что дает нам большое преимущество в распределении определенных блоков между процессами при помощи функции **MPI_Scatterv** (рис.3).

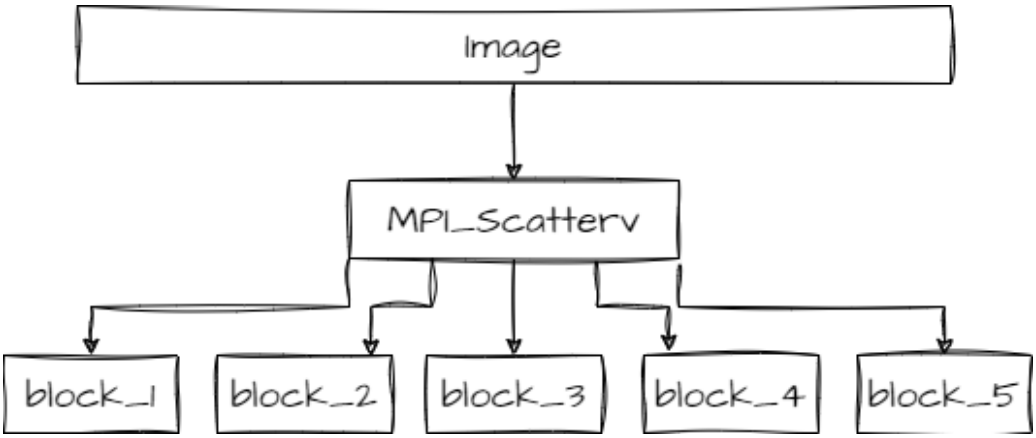

Рис 3: Распределение изображения между 5 процессами.

После того, когда на каждом процессе будет свой участок изображения, нам необходимо найти максимальное и минимальное значение пикселей. Затем при помощи функции **MPI_Allreduce** редуцировать все локальные результаты по операциям **MPI_MAX** и **MPI_MIN** соответсвенно (рис. 4).

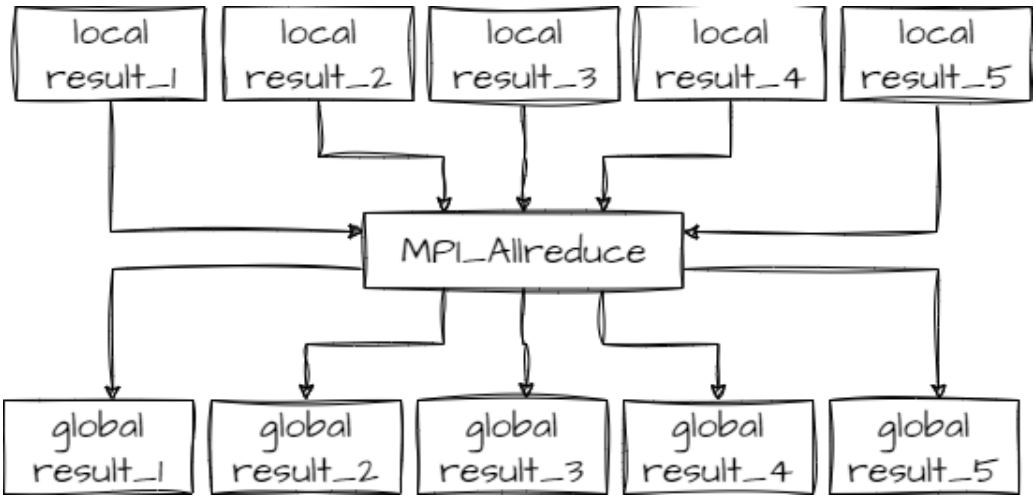

Рис 4: Редуцирование 5 процессов.

Следующим этапом будет вызов последовательной версии алгоритма для каждого из процессов, где каждый пиксель будет обработан по формуле, которая была указана ранее (2).

Самым последним действием будет сбор и объединение полученных локальных изображений при помощи функции **MPI_Gatherv** (рис.5).

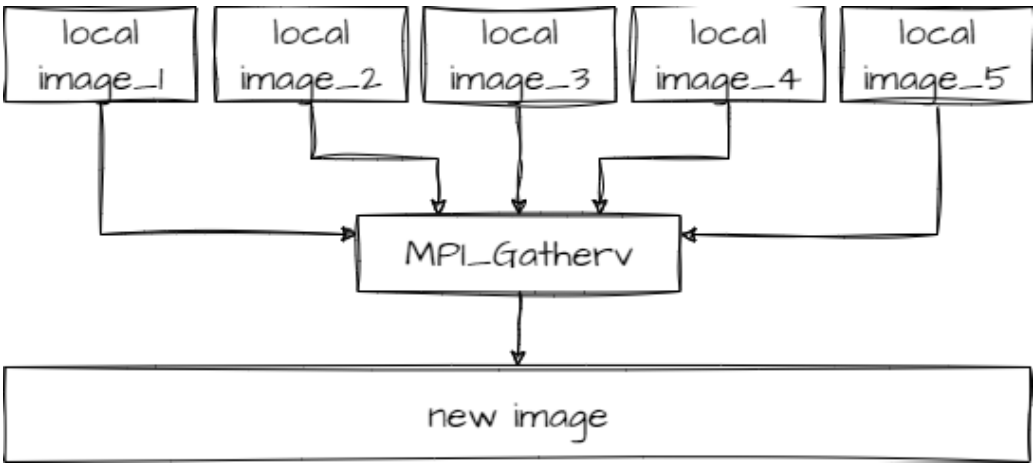

Рис 5: Объединение 5 локальных изображений в результирующее.

Заключение

В данной лабораторной работе мною были расмотрены и реализованы две версии алгоритма линейного растяжения гистограммы.

За время выполнения работы я получил большой опыт в проектировании, а также последующей разработке и отладке многопроцессорных программ.

Таким образом в заключение можно отметить, что многопроцессорные программы гораздо эффективнее обычных (однопроцессорных), однако их проектирование, разработка и сопровождение на несколько порядков сложнее.

Список литературы

[1] Получение и обработка изображений на ЭВМ : учебно-методическое пособие / В.В. Старовойтов, Ю.И. Голуб. – Минск : БНТУ, 2018. – 204 с. ISBN 978-985-550-770-4.

[2] Чабан Л.Н. Автоматизированная обработка аэрокосмической информации при картографировании геопространственных данных. Учебное пособие. – М.: МИИГАиК, 2013г., - 96 с.

[3] Wikipedia - `https://en.wikipedia.org/wiki/Main_Page`

Приложение

contrast_enhancement.h

```
1 // Copyright 2023 Kuznetsov Artem
2 #ifndef
      TASKS_TASK_3_KUZNETSOV_A_CONTRAST_ENHANCEMENT_CONTRAST_ENHANCEMENT_H_
3 #define
      TASKS_TASK_3_KUZNETSOV_A_CONTRAST_ENHANCEMENT_CONTRAST_ENHANCEMENT_H_
4
5 #include <mpi.h>
6
7 #include <algorithm>
8 #include <random>
9 #include <vector>
10
11 std::vector<uint8_t> create_random_image(size_t m, size_t n, uint8_t min,
12                                          uint8_t max);
13 void seq_increase_contrast(std::vector<uint8_t>* image, uint8_t old_min,
14                            uint8_t old_max, uint8_t new_min, uint8_t new_max
      );
15 void par_increase_contrast(std::vector<uint8_t>* image, size_t m, size_t n,
16                            uint8_t new_min, uint8_t new_max, MPI_Comm comm);
17
18 #endif  //
      TASKS_TASK_3_KUZNETSOV_A_CONTRAST_ENHANCEMENT_CONTRAST_ENHANCEMENT_H_
```

contrast_enhancement.cpp

```
1 // Copyright 2023 Kuznetsov Artem
2 #include "task_3/kuznetsov_a_contrast_enhancement/contrast_enhancement.h"
3
4 std::vector<uint8_t> create_random_image(size_t m, size_t n, uint8_t min,
5                                          uint8_t max) {
6   if (m * n == 0) return std::vector<uint8_t>();
7
8   std::random_device rd;
9   std::mt19937 gen(rd());
10  std::uniform_int_distribution<int> dist(min, max);
11
12  size_t size = m * n;
13  std::vector<uint8_t> arr(size);
14  for (auto& elem : arr) elem = dist(gen);
15
16  return arr;
17 }
18
19 void seq_increase_contrast(std::vector<uint8_t>* image, uint8_t old_min,
20                            uint8_t old_max, uint8_t new_min, uint8_t new_max
      ) {
21   if (old_min == old_max || image->size() == 0) return;
22   for (auto& pix : *image) {
23     pix = (pix - old_min) * (new_max - new_min) / (old_max - old_min) +
      new_min;
24   }
```

```
25 }
26
27 void par_increase_contrast(std::vector<uint8_t>* image, size_t m, size_t n,
28                            uint8_t new_min, uint8_t new_max, MPI_Comm comm)
     {
29   int rank = 0;
30   int size_world = 0;
31   size_t count_pix = image->size();
32
33   if (count_pix < m * n || count_pix == 0) return;
34
35   MPI_Comm_rank(comm, &rank);
36   MPI_Comm_size(comm, &size_world);
37
38   const int chunk = static_cast<int>(count_pix / size_world);
39   const int tail = static_cast<int>(count_pix % size_world);
40
41   std::vector<int> send_counts(size_world, chunk);
42   std::vector<int> displs(size_world, 0);
43
44   // Uniform distribution of elements between processes
45   for (int i = 0; i < size_world; ++i) {
46     if (i < tail) ++send_counts[i];
47     displs[i] = i == 0 ? 0 : displs[i - 1] + send_counts[i - 1];
48   }
49
50   uint8_t global_min = 255;
51   uint8_t global_max = 0;
52
53   std::vector<uint8_t> local_img(send_counts[rank]);
54
55   // Distribution of elements to processors
56   MPI_Scatterv(image->data(), send_counts.data(), displs.data(), MPI_UINT8_T
       ,
57                local_img.data(), send_counts[rank], MPI_UINT8_T, 0, comm);
58
59   uint8_t local_min = 255;
60   uint8_t local_max = 0;
61
62   // Finding the minimum and maximum values in an image
63   if (rank < count_pix) {
64     local_min = *std::min_element(local_img.begin(), local_img.end());
65     local_max = *std::max_element(local_img.begin(), local_img.end());
66   }
67
68   // Sends the found maximum and minimum to all processors
69   MPI_Allreduce(&local_min, &global_min, 1, MPI_UINT8_T, MPI_MIN, comm);
70   MPI_Allreduce(&local_max, &global_max, 1, MPI_UINT8_T, MPI_MAX, comm);
71
72   // Running a sequential version
73   seq_increase_contrast(&local_img, global_min, global_max, new_min, new_max
       );
74
75   // Gather results
76   MPI_Gatherv(local_img.data(), send_counts[rank], MPI_UINT8_T, image->data
       (),
77               send_counts.data(), displs.data(), MPI_UINT8_T, 0, comm);
78 }
```

main.cpp

```
1  // Copyright 2023 Kuznetsov Artem
2  #include <gtest/gtest.h>
3  #include <mpi.h>
4
5  #include <iostream>
6
7  #include "./contrast_enhancement.h"
8
9  TEST(MPI_TESTS, Test_small) {
10   int rank = 0;
11   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
12
13   const size_t m = 4;
14   const size_t n = 4;
15   const uint8_t new_min = 0;
16   const uint8_t new_max = 255;
17
18   std::vector<uint8_t> image{10,  50, 100, 150, 200, 16,  0, 54,
19                              122, 89, 100, 0,   1,   255, 4, 5};
20   std::vector copyImage(image);
21
22   par_increase_contrast(&image, m, n, new_min, new_max, MPI_COMM_WORLD);
23
24   if (rank == 0) {
25     uint8_t old_min = *std::min_element(copyImage.begin(), copyImage.end());
26     uint8_t old_max = *std::max_element(copyImage.begin(), copyImage.end());
27
28     seq_increase_contrast(&copyImage, old_min, old_max, new_min, new_max);
29
30     ASSERT_EQ(image, copyImage);
31   }
32 }
33
34 TEST(MPI_TESTS, Test_all_zero) {
35   int rank = 0;
36   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
37
38   const size_t m = 4;
39   const size_t n = 4;
40   const uint8_t new_min = 0;
41   const uint8_t new_max = 255;
42
43   std::vector<uint8_t> image(m * n, 0);
44   std::vector copyImage(image);
45
46   par_increase_contrast(&image, m, n, new_min, new_max, MPI_COMM_WORLD);
47
48   if (rank == 0) {
49     uint8_t old_min = *std::min_element(copyImage.begin(), copyImage.end());
50     uint8_t old_max = *std::max_element(copyImage.begin(), copyImage.end());
51
52     seq_increase_contrast(&copyImage, old_min, old_max, new_min, new_max);
53
54     ASSERT_EQ(image, copyImage);
55   }
56 }
57
58 TEST(MPI_TESTS, Test_all_255) {
```

```
59   int rank = 0;
60   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
61
62   const size_t m = 4;
63   const size_t n = 4;
64   const uint8_t new_min = 0;
65   const uint8_t new_max = 255;
66
67   std::vector<uint8_t> image(m * n, 255);
68   std::vector copyImage(image);
69
70   par_increase_contrast(&image, m, n, new_min, new_max, MPI_COMM_WORLD);
71
72   if (rank == 0) {
73     uint8_t old_min = *std::min_element(copyImage.begin(), copyImage.end());
74     uint8_t old_max = *std::max_element(copyImage.begin(), copyImage.end());
75
76     seq_increase_contrast(&copyImage, old_min, old_max, new_min, new_max);
77
78     ASSERT_EQ(image, copyImage);
79   }
80 }
81
82 TEST(MPI_TESTS, Test_empty) {
83   int rank = 0;
84   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
85
86   const size_t m = 0;
87   const size_t n = 0;
88   const uint8_t new_min = 0;
89   const uint8_t new_max = 255;
90
91   std::vector<uint8_t> image{};
92   std::vector copyImage(image);
93
94   par_increase_contrast(&image, m, n, new_min, new_max, MPI_COMM_WORLD);
95
96   if (rank == 0) {
97     ASSERT_EQ(image, copyImage);
98   }
99 }
100
101 TEST(MPI_TESTS, Test_random) {
102   int rank = 0;
103   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
104
105   const size_t m = 1000;
106   const size_t n = 1000;
107   const uint8_t new_min = 100;
108   const uint8_t new_max = 200;
109
110   std::vector<uint8_t> image = create_random_image(m, n, 0, 255);
111   std::vector copyImage(image);
112
113   par_increase_contrast(&image, m, n, new_min, new_max, MPI_COMM_WORLD);
114
115   if (rank == 0) {
116     uint8_t old_min = *std::min_element(copyImage.begin(), copyImage.end());
117     uint8_t old_max = *std::max_element(copyImage.begin(), copyImage.end());
118
```

```
119     seq_increase_contrast(&copyImage, old_min, old_max, new_min, new_max);
120
121     ASSERT_EQ(image, copyImage);
122   }
123 }
124
125 int main(int argc, char** argv) {
126   int result_code = 0;
127   int rank = 0;
128
129   ::testing::InitGoogleTest(&argc, argv);
130   ::testing::TestEventListeners& listeners =
131       ::testing::UnitTest::GetInstance()->listeners();
132
133   if (MPI_Init(&argc, &argv) != MPI_SUCCESS) MPI_Abort(MPI_COMM_WORLD, -1);
134   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
135
136   if (rank != 0) {
137     delete listeners.Release(listeners.default_result_printer());
138   }
139
140   result_code = RUN_ALL_TESTS();
141   MPI_Finalize();
142
143   return result_code;
144 }
```