Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Нижегородский государственный университет им. Н.И. Лобачевского»

**Институт информационных технологий, математики и механики**

**Отчет**

по лабораторной работе №3

**«**Повышение контраста полутонового изображения посредством линейной растяжки гистограммы**»**

**Выполнил:**

студент группы 381708-2

Сметанин Д.А.

Нижний Новгород

2019

**Содержание**

[Постановка задачи 2](#_Toc531367741)

[Метод решения 3](#_Toc531367742)

[Схема распараллеливания 4](#_Toc531367743)

[Описание программной реализации 5](#_Toc531367744)

[Подтверждение корректности 6](#_Toc531367745)

[Результаты экспериментов 7](#_Toc531367746)

[Заключение 8](#_Toc531367747)

[Приложение 9](#_Toc531367748)

# Постановка задачи

Цель – получение навыков распараллеливания задач компьютерной графики средствами библиотеки MPI.

Ограничения:

* по срокам выполнения работы

Критерии эффективности разработки:

* результаты тестирования корректности работы алгоритма (сравнение результатов работы линейной и параллельной версии программы)
* производительность параллельной версии программы выше, чем линейной

Требуется:

* К исходному изображению применяется линейное растяжение гистограммы
* Должна использоваться библиотека MPI
* Схема распараллеливания не зависит от количества предоставленных ресурсов

# Метод решения

Концепция: Задача была решена методом горизонтального разбиения исходного изображения.

1. Находится оптимальное количество блоков, на которое разбивается исходное изображение;
2. Изображение разбивается на горизонтальные блоки и отсылается от 0 процесса всем остальным;
3. На всех процессах одновременно находится максимальное и минимальное значение яркости в блоке, а затем находится минимальное и максимальное значение яркости в изображении;
4. К блокам применяется линейное растяжение гистограммы;
5. Обработанные блоки отсылаются на 0 процесс;
6. 0 процесс производит компоновку блоков для получения результирующего изображения.

# Схема распараллеливания

Нулевой процесс отсылает горизонтальные блоки исходного изображения всем остальным процесса.

Обработка изображения происходит на всех процессах.

Затем обработанные блоки отсылаются обратно на нулевой процесс и он же осуществляет финальную компоновку результирующего изображения.

# Описание программной реализации

**Руководство пользователя**

Программа реализует линейную и параллельную версии алгоритма линейного растяжения гистограммы.

Для начала работы с программой необходимо установить библиотеки MPI и OpenCV 3.4.0

Запуск программы возможен только из интерфейса командной строки, посредством введения в неё команды: mpiexec [–n <нужное кол-во процессов>] Project1.exe <путь к файлу изображения> .

**Руководство программиста**

Код программы можно просмотреть в разделе «[Приложение](#_Приложение)».

# Подтверждение корректности

Для подтверждения корректности в программе предусмотрена автоматическая проверка полного соответствия результата работы линейной версии алгоритма результату работы параллельной версии алгоритма.

Также предусмотрена возможность визуально сравнить исходной изображение, изображение полученное линейной версией алгоритма программы и изображение полученное в результате работы параллельной версии алгоритма.

# Результаты экспериментов

По данным экспериментов видно, что меняя количество процессов на которых исполняется программа – меняется время работы алгоритма. При определённом количестве процессов накладные расходы на пересылку данных нивелируются сложностью вычислений и виден прирост производительности параллельной версии программы по сравнению с линейной. Также можно заметить, что от изменения числа процессов не меняется результат работы программы.

# Заключение

Выполнив данную лабораторную работу мы получили навыки работы с библиотекой MPI: подключение библиотеки к проекту, использование средств библиотеки, запуск приложений использующих данную библиотеку. Был получен опыт оптимального разбиения двухмерных данных на горизонтальные блоки.

Также нам удалось убедиться в эффективности работы с библиотекой MPI в том случае когда пересылка большого объема данных нивелируется сложностью вычислений.

# Приложение

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <mpi.h>

#include <stdlib.h>

#include <iostream>

using namespace cv;

unsigned char clamp(float val) {

if (val < 0)

val = 0;

else if(val > UCHAR\_MAX)

val = UCHAR\_MAX;

return (unsigned char)val;

}

int main(int argc, char\* argv[])

{

if (argc != 2) {

std::cout << " Usage: display\_image ImageToLoadAndDisplay" << std::endl;

return -1;

}

Mat \*image = nullptr;

Mat \*lin\_image = nullptr;

Mat \*res\_image = nullptr;

int img\_width = 0;

int img\_height = 0;

int img\_size = 0;

unsigned char \*buf = nullptr;

int size = 0;

double start\_time = 0;

double end\_time = 0;

MPI\_Init(&argc, &argv);

int proc\_num = 0;

int proc\_rank = 0;

MPI\_Comm\_size(MPI\_COMM\_WORLD, &proc\_num);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &proc\_rank);

if (proc\_rank == 0) {

image = new Mat();

\*image = imread(argv[1], IMREAD\_GRAYSCALE);

if (!image->data) {

std::cout << "Could not open or find the image" << std::endl;

return -1;

}

img\_width = image->cols;

img\_height = image->rows;

img\_size = img\_height \* img\_width;

lin\_image = new Mat(img\_height, img\_width, CV\_8UC1);

res\_image = new Mat(img\_height, img\_width, CV\_8UC1);

unsigned char max = 0;

unsigned char min = UCHAR\_MAX;

start\_time = MPI\_Wtime();

for (int i = 0; i < img\_size; i++) {

if (image->data[i] < min)

min = image->data[i];

if (image->data[i] > max)

max = image->data[i];

}

for (int i = 0; i < img\_size; i++) {

lin\_image->data[i] = clamp(((float)(image->data[i] - min) \* ((float)UCHAR\_MAX / (max - min))));

}

end\_time = MPI\_Wtime();

std::cout << "Line time: " << end\_time - start\_time << std::endl;

int first\_end = img\_height / proc\_num;

size = first\_end \* img\_width;

start\_time = MPI\_Wtime();

for (int i = 1; i < proc\_num; i++) {

int begin = i \* img\_height / proc\_num;

int end = (i + 1) \* img\_height / proc\_num;

MPI\_Send(&image->data[begin \* img\_width], (end - begin) \* img\_width, MPI\_CHAR, i, 0, MPI\_COMM\_WORLD);

}

}

int gen\_min = 0;

int gen\_max = 0;

int min = UCHAR\_MAX;

int max = 0;

if (proc\_rank > 0){

MPI\_Status stat;

MPI\_Probe(0, 0, MPI\_COMM\_WORLD, &stat);

MPI\_Get\_count(&stat, MPI\_CHAR, &size);

buf = new unsigned char[size];

MPI\_Recv(buf, size, MPI\_CHAR, 0, 0, MPI\_COMM\_WORLD, &stat);

for (int i = 0; i < size; i++) {

if (buf[i] < min)

min = buf[i];

if (buf[i] > max)

max = buf[i];

}

}

else {

for (int i = 0; i < size; i++) {

if (image->data[i] < min)

min = image->data[i];

if (image->data[i] > max)

max = image->data[i];

}

}

MPI\_Allreduce(&max, &gen\_max, 1, MPI\_INT, MPI\_MAX, MPI\_COMM\_WORLD);

MPI\_Allreduce(&min, &gen\_min, 1, MPI\_INT, MPI\_MIN, MPI\_COMM\_WORLD);

if (proc\_rank > 0) {

for (int i = 0; i < size; i++) {

buf[i] = clamp(((float)(buf[i] - gen\_min) \* ((float)UCHAR\_MAX / (gen\_max - gen\_min))));

}

MPI\_Send(buf, size, MPI\_CHAR, 0, 1, MPI\_COMM\_WORLD);

}

else {

for (int i = 0; i < size; i++) {

res\_image->data[i] = clamp(((float)(image->data[i] - gen\_min) \* ((float)UCHAR\_MAX / (gen\_max - gen\_min))));

}

}

if (proc\_rank == 0) {

for (int i = 1; i < proc\_num; i++) {

MPI\_Status stat;

int recv\_size = 0;

MPI\_Probe(MPI\_ANY\_SOURCE, 1, MPI\_COMM\_WORLD, &stat);

MPI\_Get\_count(&stat, MPI\_CHAR, &recv\_size);

int begin = stat.MPI\_SOURCE \* img\_height / proc\_num;

MPI\_Recv(&res\_image->data[begin \* img\_width], recv\_size, MPI\_CHAR, stat.MPI\_SOURCE, 1, MPI\_COMM\_WORLD, &stat);

}

end\_time = MPI\_Wtime();

std::cout << "Parallel time: " << end\_time - start\_time << std::endl;

bool ok = true;

for (int i = 0; i < img\_size; i++) {

if (res\_image->data[i] != lin\_image->data[i]) {

ok = false;

}

}

if (ok) {

std::cout << "TEST PASSED!" << std::endl;

}

else {

std::cout << "TEST FAILED!" << std::endl;

}

namedWindow("RAW IMAGE", WINDOW\_KEEPRATIO);

imshow("RAW IMAGE", \*image);

namedWindow("LINE CALC", WINDOW\_KEEPRATIO);

imshow("LINE CALC", \*lin\_image);

namedWindow("PARALLEL CALC", WINDOW\_KEEPRATIO);

imshow("PARALLEL CALC", \*res\_image);

waitKey(0);

}

MPI\_Finalize();

return 0;

}