**ПРИЛОЖЕНИЕ А**

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

# КАФЕДРА ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

РЕАЛИЗАЦИЯ ПАРАЛЛЕЛЬНОЙ ОБРАБОТКИ ДЛЯ КЛАСТЕРНЫХ / МУЛЬТИПРОЦЕССОРНЫХ СИСТЕМ НА БАЗЕ ПРОТОКОЛА MPI

**ТЕКСТ ПРОГРАММЫ**

КП.ПО7.200149-03 12 00

## Листов 5

|  |  |
| --- | --- |
| Руководитель | Ю.В. Савицкий |
| Выполнил | А.Е. Комиссаров |
| Консультант |  |
| по ЕСПД | Ю.В. Савицкий |
|  |  |

2022

**СОДЕРЖАНИЕ**

**Normal.cpp –** основной файл программы, реализующий последовательный алгоритм обработки изображения.

**Parallel.cpp –** основной файл программы, реализующий параллельный алгоритм обработки изображения.

**Normal.cpp**

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <opencv2/imgproc.hpp>

#include <chrono>

#include <iostream>

#include <iomanip>

#include <cmath>

using namespace cv; //OpenCV2 - Computer Vision

using namespace std; //std::

Mat BordersRobertsDefault(Mat IMAGE) { //выделение границ оператором Робертса без MIP

auto start = chrono::steady\_clock::now();

int first, second, result, p1, p2, p3, p4;

int x, y;

for (x = 0; x < IMAGE.rows - 1; x++) { //Все циклы длиной = размер - 1 пикс.

for (y = 0; y < IMAGE.cols - 1; y++) {

p1 = IMAGE.at<uchar>(x, y); // A = x , y

p2 = IMAGE.at<uchar>(x + 1, y + 1); // B = x + 1 , y + 1

p3 = IMAGE.at<uchar>(x + 1, y); // C = x + 1 , y

p4 = IMAGE.at<uchar>(x, y + 1); // D = x , y + 1

first = p1 - p2; // I = A - B

second = p3 - p4; // II = C - D

result = sqrt(pow(first, 2) + pow(second, 2)); // Результирующий пиксель: R = (I^2 + II^2)^1/2;

IMAGE.at<uchar>(x, y) = result; // Запись в пиксель

}

}

auto end = chrono::steady\_clock::now();

cout << fixed;

cout << setprecision(4);

cout << "Elapsed time in seconds: "<< float(chrono::duration\_cast<chrono::milliseconds>(end - start).count()) / 1000<< " sec"<<endl;

return IMAGE; //возврат матрицы

}

float getScale() { //функция ввода масштаба картинки

float scale;

cout << "Enter scale: " << endl;

cin >> scale;

return scale;

}

void Show(Mat image) { //функция показа картинки

imshow("Image", image);

waitKey(0);

}

int main() {

float scale;

string image\_path = "raw.bmp"; //путь к картинке

Mat image = imread(image\_path, IMREAD\_GRAYSCALE); //чтение в Ч/Б режиме

image = BordersRobertsDefault(image); //выделение границ

scale = getScale(); //получаем масштаб с клавиатуры

resize(image, image, cv::Size(0, 0), scale, scale, INTER\_LINEAR); //меняем размеры картинки по масштабу

Show(image); //показать изображение

return 0;

}

**Parallel.cpp**

#include <iostream>

#include "mpi.h"

#include "opencv2/opencv.hpp"

#include <cmath>

#include <iostream>

using namespace cv;

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

{

/////////////////////////////////////

int size, rank, first = 0, second = 0, result = 0, channels = 0;

Mat image, new\_image;

uchar\* sector\_data;

size\_t full\_size = 0, sector\_size = 0;

float scale;

/////////////////////////////////////

MPI\_Init(&argc, &argv);

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

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

/////////////////////////////////////

if (rank == 0)

{

image = imread("img/raw.bmp", IMREAD\_GRAYSCALE);

if (image.empty()) //get empty image error

{

std::cout << "(err) Image has not been loaded!" << std::endl;

return -1;

}

channels = image.channels(); //get channels number

full\_size = image.step[0] \* image.rows; //get full matrix size

if (image.total() % size) //get sector size error

{ //image.total() == image.rows \* image.cols

std::cout << "(err) Cannot evenly divide the image." << std::endl;

return -2;

}

sector\_size = full\_size / size; //get sector size

std::cout << "Dividing image into " << sector\_size << " byte sectors" << std::endl;

}

// broadcast sector size and channels number from #0 to other processes

MPI\_Bcast(&sector\_size, 1, MPI\_UNSIGNED\_LONG\_LONG, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(&channels, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

MPI\_Barrier(MPI\_COMM\_WORLD); //wait for all processes to get sizes

double start = MPI\_Wtime();

sector\_data = new uchar[sector\_size]; //init the sector data buffer

MPI\_Barrier(MPI\_COMM\_WORLD); //wait for all processes to initialize their buffer

// scatter image between processes

MPI\_Scatter(image.data, sector\_size, MPI\_UNSIGNED\_CHAR, sector\_data, sector\_size, MPI\_UNSIGNED\_CHAR, 0, MPI\_COMM\_WORLD);

MPI\_Barrier(MPI\_COMM\_WORLD); // wait for all processes to get their own image sector

//after all of thi is done, each process will have their own image sector, now we can work on image processing

//image is divided by rows (thats how it's stored in the memory)

//sector\_data buffer contains a sequence of (B, G, R) values

for (size\_t i = 0; i < sector\_size - image.cols - 1; i += channels)

{

int A = sector\_data[i]; // A = x , y

int B = sector\_data[i + 1 + image.cols]; // B = x + 1 , y + 1

int C = sector\_data[i + 1]; // C = x + 1 , y

int D = sector\_data[i + image.cols]; // D = x , y + 1

first = A - B; // I = A - B

second = C - D; // II = C - D

result = sqrt(pow(first, 2) + pow(second, 2)); // Result: R = (I^2 + II^2)^1/2;

sector\_data[i] = result; // Copying the result into a pixel

}

MPI\_Barrier(MPI\_COMM\_WORLD); // wait for all the processes to be done with image processing

double end = MPI\_Wtime();

if (rank == 0)

{

new\_image = cv::Mat(image.size(), image.type()); //init the output image

}

//get the scattered info back, but this time it's been processed

MPI\_Gather(sector\_data, sector\_size, MPI\_UNSIGNED\_CHAR, new\_image.data, sector\_size, MPI\_UNSIGNED\_CHAR, 0, MPI\_COMM\_WORLD);

if (rank == 0)

{

// save the image:

std::cout << "Enter image scale: ";

std::cin >> scale;

resize(new\_image, new\_image, cv::Size(0, 0), scale, scale, INTER\_LINEAR);

imwrite("img/computed.bmp", new\_image);

std::cout << "Elapsed time in seconds: " << end - start << " sec" << std::endl;

}

delete[] sector\_data;

MPI\_Finalize();

}