**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.ПО6.33203-03 12 00

## Листов 5

|  |  |
| --- | --- |
| Руководитель | Ю.В. Савицкий |
| Выполнил | Д. М. Мартынович |
| Консультант |  |
| по ЕСПД | Ю.В. Савицкий |
|  |  |

2022

**СОДЕРЖАНИЕ**

Consistent.cpp – файл с реализацией последовательной обработки.

#include <iostream>

#include <string>

#include <stdio.h>

#include <opencv2/opencv.hpp>

#include <opencv2/core.hpp>

#include <opencv2/imgcodecs.hpp>

#include <opencv2/highgui.hpp>

#include <time.h>

#include <Windows.h>

using namespace std;

using namespace cv;

Mat ConvolutionOperationRobert(Mat& src);

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

{

system("color f0");

string inputImageName;

cout << "Enter the name of your input image: ";

cin >> inputImageName;

string outputImageName;

cout << "Enter the name of your output image: ";

cin >> outputImageName;

unsigned int end\_time = clock();

Mat img = imread(inputImageName, IMREAD\_GRAYSCALE);

Mat output = ConvolutionOperationRobert(img);

if (img.empty()) {

cout << "Could not to read image: " << inputImageName << endl;

}

imwrite(outputImageName, output);

cout << "Operation success! Time = " << end\_time / 1000.0 << endl;

cout << "File saved in project catalog" << endl;

cout << "Whant to see the file?(Y/N)";

char variant;

cin >> variant;

switch (variant) {

case 'y':

cout << "Open..." << endl;

namedWindow(outputImageName, WINDOW\_NORMAL);

imshow(outputImageName, output);

waitKey(0);

break;

case 'n':

break;

default:

cout << "Invalid input" << endl;

break;

}

return 0;

}

Mat ConvolutionOperationRobert(Mat& src)

{

Mat dst(src.rows, src.cols, src.type(), Scalar(0));

if (src.channels() == 1)

for (int i = 0; i < src.rows - 1; i++)

for (int j = 0; j < src.cols - 1; j++)

dst.at<uchar>(i, j) = abs(src.at<uchar>(i, j) - src.at<uchar>(i + 1, j + 1)) + abs(src.at<uchar>(i + 1, j) - src.at<uchar>(i, j + 1));

return dst;

}

Parallel.cpp – файл с параллельной обработкой изображения.

#include <iostream>

#include <string>

#include "mpi.h"

#include "opencv2/opencv.hpp"

#include <time.h>

using namespace std;

using namespace cv;

void ConvolutionOperationRobert(uchar\*& partialBuffer, size\_t imagePartialSize, int channels);

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

{

unsigned int end\_time = clock();

// the input image

cv::Mat image;

// the total size of the image matrix (rows \* columns \* channels):

size\_t imageTotalSize;

// partial size (how many bytes will be sent to each process):

size\_t imagePartialSize;

// how many channels are there in the image?

int channels;

// partial buffer, to contain the image.

// 'uchar' means 'unsigned char', i.e. an 8-bit value, because each pixel in an image is a byte (0..255)

uchar\* partialBuffer;

// also create the output image, where we will save the results:

cv::Mat outImage;

// ------------------------------------

// start the MPI part

MPI\_Init(&argc, &argv);

// get the world size and current rank:

int size;

int rank;

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

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

// read the image and its properties in the ROOT process:

if (rank == 0)

{

// read the image

image = cv::imread("1.bmp", IMREAD\_GRAYSCALE);

resize(image, image, Size(6000, 6000));

// check if it's empty:

if (image.empty())

{

std::cerr << "Image is empty, terminating!" << std::endl;

return -1;

}

// get the number of channels in the image

channels = image.channels();

// get the total size of the image matrix (rows \* columns \* channels)

// the explanation can be found here: https://stackoverflow.com/a/26441073/4003714

imageTotalSize = image.step[0] \* image.rows;

// check if we can evenly divide the image bytes by the number of processes

// the image.total() method returns the number of elements, i.e. (rows \* cols)

if (image.total() % size)

{

std::cerr << "Cannot evenly divide the image between the processes. Choose a different number of processes!" << std::endl;

return -2;

}

// get partial size (how many bytes are sent to each process):

imagePartialSize = imageTotalSize / size;

std::cout << "The image will be divided into blocks of " << imagePartialSize << " bytes each" << std::endl;

}

// send the "partial size" from #0 to other processes:

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

// send the number of channels in the image from #0 to other processes:

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

// synchronize the processes here, to make sure that the sizes are initialized:

MPI\_Barrier(MPI\_COMM\_WORLD);

// allocate the partial buffer:

partialBuffer = new uchar[imagePartialSize];

// synchronize the processe here, to make sure each process has allocated the buffer:

MPI\_Barrier(MPI\_COMM\_WORLD);

// scatter the image between the processes:

MPI\_Scatter(image.data, imagePartialSize, MPI\_UNSIGNED\_CHAR, partialBuffer, imagePartialSize, MPI\_UNSIGNED\_CHAR, 0, MPI\_COMM\_WORLD);

// synchronize the image processing:

MPI\_Barrier(MPI\_COMM\_WORLD);

// -----------------------------------------------------------------------------------------------------------------------------------------------

// AND NOW HERE EACH PROCESS HAS ITS OWN PART OF THE IMAGE!

// THE PARTS ARE DIVIDED ROW-WISE, I.E. THE FIRST PROCESS WILL HAVE THE FIRST COUPLE OF ROWS, THE SECOND PROCESS HAS THE NEXT COUPLE OF ROWS, ETC.

// IMPORTANT: the matrix `partialBuffer` is 1-dimensional, and contains the image data in the following format:

// b0 g0 r0 b1 g1 r1 b2 g2 r2 b3 g3 r3 ....... bN-1 gN-1 rN-1

// so, just a sequence of (B, G, R) values, from beginning to end

// you can convert it into a 3D matrix here, if you want, or into a 2D-matrix... or just leave it like that.

// but anyway, you can now process the image, FOR EXAMPLE:

// iterate through the image

ConvolutionOperationRobert(partialBuffer, imagePartialSize, channels);

// -----------------------------------------------------------------------------------------------------------------------------------------------

// synchronize the image processing:

MPI\_Barrier(MPI\_COMM\_WORLD);

// initialize the output image (only need to do it in the ROOT process)

if (rank == 0)

{

outImage = cv::Mat(image.size(), image.type());

}

// and now we finally send the partial buffers back to the ROOT, gathering the complete image:

MPI\_Gather(partialBuffer, imagePartialSize, MPI\_UNSIGNED\_CHAR, outImage.data, imagePartialSize, MPI\_UNSIGNED\_CHAR, 0, MPI\_COMM\_WORLD);

// Save and display image, onle in the ROOT process

if (rank == 0)

{

// save the image:

resize(outImage, outImage, Size(4000, 4000));

cv::imwrite("out.bmp", outImage);

cout << "Operation success! Time = " << end\_time / 1000.0 << endl;

}

delete[]partialBuffer;

MPI\_Finalize();

}

void ConvolutionOperationRobert(uchar\*& partialBuffer, size\_t imagePartialSize, int channels) {

uchar\* out = new uchar[imagePartialSize];

uchar A, B, C, D;

for (size\_t i = 0; i < 1000 - 1; i += channels)

{

for (int j = 0; j < 6000 - 1; j++) {

A = partialBuffer[i \* 6000 + j];

B = partialBuffer[(i + 1) \* 6000 + j];

C = partialBuffer[i \* 6000 + j + 1];

D = partialBuffer[(i + 1) \* 6000 + j + 1];

out[i \* 6000 + j] = abs(A - D) + abs(B - C);

out[i \* 6000 + j] = saturate\_cast<uchar>(out[i \* 6000 + j] \* 2);

}

}

swap(out, partialBuffer);

delete[]out;

}