ПРИЛОЖЕНИЕ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

КАФЕДРА ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

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

КОД ПРОГРАММЫ

КП.ПО-9.1-40 01 01

Листов

Руководитель Ю. В. Савицкий

Выполнил З. С. Харитонович

Консультант по ЕСПД Ю. В. Савицкий

.

Брест 2023

Последовательный алгоритм.

Source.cpp

```
#include <iostream>
#include <string>
#include <opencv2/opencv.hpp>

cv::Mat imageProcessing(cv::Mat image);

int main(int argc, char** argv) {
   cv::Mat image = cv::imread("d:/input.bmp");
   if (image.empty()) return -1;
   cv::Mat result;
   std::clock_t timer = std::clock();
   result = imageProcessing(image);
   timer = std::clock() - timer;
   std::cout << timer << " ms" << std::endl;
   cv::imwrite("d:/outputSeq.bmp", result);
   return 0;
}
cv::Mat imageProcessing(cv::Mat image) {
   cv::Mat result;
   cv::medianBlur(image, result, 3); result);

   cv::Mat diffX;
   cv::Sobel(result, diffX, CV_32F, 1, 0);
   cv::Mat diffY;
   cv::Sobel(result, diffY, CV_32F, 0, 1);

   cv::magnitude(diffX, diffY, result);

   return result;
}
```

Параллельный алгоритм

Source.cpp

```
#include <iostream>
#include <string>
#include <mpi.h>
#include <opencv2/opencv.hpp>

cv::Mat imageProcessing(cv::Mat image);

int main(int argc, char** argv) {
   MPI_Init(&argc, &argv);
   int rank, size;
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   MPI_Comm_size(MPI_COMM_WORLD, &size);
   cv::Mat image = cv::imread("d:/input.bmp"), result;
   if (image.empty()) return -1;
   std::clock_t timer = std::clock();
   int width = image.cols, height = image.rows;
   cv::Mat imagePart = image.clone()(cv::Rect(rank * (width /
```

```
size), 0, width / size, height));

   imagePart = imageProcessing(imagePart);
   if (rank == 0) {
      result = imagePart.clone();
      for (int i = 1; i < size; i++) {
         MPI_Recv(imagePart.data, width * height * 3,
MPI_UNSIGNED_CHAR,
            i, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
         cv::hconcat(result, imagePart, result);
      }
      timer = std::clock() - timer;
      std::cout << timer << " ms" << std::endl;
      cv::imwrite("d:/outputMPI.bmp", result);
   }
   else {
      MPI_Send(imagePart.data, width * height * 3,
MPI_UNSIGNED_CHAR, 0, 0, MPI_COMM_WORLD);
   }
   MPI_Finalize();
   return 0;
}

cv::Mat imageProcessing(cv::Mat image) {
   cv::Mat result;
   cv::medianBlur(image, result, 3); result);

   cv::Mat diffX;
   cv::Sobel(result, diffX, CV_32F, 1, 0);
   cv::Mat diffY;
   cv::Sobel(result, diffY, CV_32F, 0, 1);

   cv::magnitude(diffX, diffY, result);

   return result;
}
```