Приложение А

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
КАФЕДРА ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

ПРОЕКТИРОВАНИЕ СИСТЕМ ПАРАЛЛЕЛЬНОЙ ОБРАБОТКИ С ИСПОЛЬЗОВАНИЕМ MPI.

ТЕКСТ ПРОГРАММЫ

КП.ПО5.170154 - 01 12 00

Листов 5

Руководитель	Савицкий Ю. В.
Выполнил	Корнасевич И. Д.
Консультант по ЕСПД	Савицкий Ю. В.

Брест 2021

Листинг 1: main.cpp

```
1  #include <iostream>
2  #include <string>
3  #include <omp.h>
4  #include "bitmap_image.hpp"
5  #include "functions.h"
6
7  #define THREAD_COUNT 24
8  #define IMAGE_PIECE_SIZE 125
9
10
11 void nonParallel() {
12     deNoise(0, IMAGE_SIZE);
13     upgrade(0, IMAGE_SIZE);
14 }
15
16 void parallel() {
17 #pragma omp parallel for default(none)
18     for (int i = 0; i < THREAD_COUNT; ++i) {
19         deNoise(i * IMAGE_PIECE_SIZE, (i + 1) * IMAGE_PIECE_SIZE);
20     }
21 #pragma omp parallel for default(none)
22     for (int i = 0; i < THREAD_COUNT; ++i) {
23         upgrade(i * IMAGE_PIECE_SIZE, (i + 1) * IMAGE_PIECE_SIZE);
24     }
25 }
26
27 int main(int argc, char *argv[]) {
28     omp_set_num_threads(THREAD_COUNT);
29     const string fileName = "../imageNoised.bmp";
30     bitmap_image image(fileName);
31     if (!image) {
32         cout << "Unable to open file" << endl;
33         return 1;
34     }
35     loadPixels(image, 0, IMAGE_SIZE);
36     double start = omp_get_wtime();
37     if (argc > 1){
38         cout << "parallel\n";
39         parallel();
40     }
41     else{
42         cout << "nonParallel\n";
```

```
43          nonParallel();
44      }
45      double end = omp_get_wtime();
46      cout << "time:" << (end - start) << endl;
47      unloadPixels(image, 0, IMAGE_SIZE);
48      image.save_image("../output.bmp");
49      return 0;
50  }
```

Листинг 2: functions.h

```
1  #ifndef COURSEWORK_FUNCTIONS_H
2  #define COURSEWORK_FUNCTIONS_H
3  #define IMAGE_SIZE 3000
4
5  using namespace std;
6  void loadPixels(const bitmap_image &image, size_t start, size_t end)
       ;
7
8  void deNoise(size_t start, size_t end);
9
10 void upgrade(size_t start, size_t end);
11
12 void unloadPixels(bitmap_image &image, size_t start, size_t end);
13
14 #endif //COURSEWORK_FUNCTIONS_H
```

Листинг 3: functions.cpp

```
1  #include "bitmap_image.hpp"
2  #include "functions.h"
3  static u_char pixels[IMAGE_SIZE][IMAGE_SIZE];
4  static u_char bufPixels[IMAGE_SIZE][IMAGE_SIZE];
5
6  typedef bool (*func)(u_char pointValue);
7
8  void loadPixels(const bitmap_image &image, size_t start, size_t end)
       {
9      for (size_t i = start; i < end; i++) {
10         for (size_t j = 0; j < IMAGE_SIZE; j++) {
11             pixels[i][j] = image.get_pixel(i, j).green;
12         }
13     }
14 }
15
```

```
16 void deNoise(size_t start, size_t end) {
17     if (start == 0) {
18         start++;
19     }
20     if (end == IMAGE_SIZE) {
21         end--;
22     }
23     for (size_t i = start; i < end; i++) {
24         for (size_t j = 1; j < IMAGE_SIZE - 1; j++) {
25             uint accumulator = 0;
26             accumulator += pixels[i - 1][j - 1];
27             accumulator += pixels[i][j - 1];
28             accumulator += pixels[i + 1][j - 1];
29             accumulator += pixels[i - 1][j];
30             accumulator += pixels[i + 1][j];
31             accumulator += pixels[i - 1][j + 1];
32             accumulator += pixels[i][j + 1];
33             accumulator += pixels[i + 1][j + 1];
34             bufPixels[i][j] = accumulator >> 3;
35         }
36     }
37     for (size_t i = start; i < end; i++) {
38         memcpy(pixels[i], bufPixels[i], IMAGE_SIZE);
39     }
40 }
41
42 bool hasAround(size_t i, size_t j, func f) {
43     return f(pixels[i - 1][j - 1]) ||
44            f(pixels[i][j - 1]) ||
45            f(pixels[i + 1][j - 1]) ||
46            f(pixels[i - 1][j]) ||
47            f(pixels[i + 1][j]) ||
48            f(pixels[i - 1][j + 1]) ||
49            f(pixels[i][j + 1]) ||
50            f(pixels[i + 1][j + 1]);
51 }
52
53 void increase(size_t start, size_t end, func f, u_char value) {
54     for (size_t i = start; i < end; i++) {
55         for (size_t j = 0; j < IMAGE_SIZE; j++) {
56             if (j != 0 && j != IMAGE_SIZE - 1 && hasAround(i, j, f))
    {
57                 bufPixels[i][j] = value;
```

```
58              } else {
59                  bufPixels[i][j] = pixels[i][j];
60              }
61          }
62      }
63      for (size_t i = start; i < end; i++) {
64          memcpy(pixels[i], bufPixels[i], IMAGE_SIZE);
65      }
66  }
67
68  void upgrade(size_t start, size_t end) {
69      func increaseFunc = [](u_char value) { return value > 245; };
70      func reduceFunc = [](u_char value) { return value < 15; };
71      increase(start, end, reduceFunc, 0);
72      increase(start, end, reduceFunc, 0);
73      increase(start, end, reduceFunc, 0);
74      increase(start, end, increaseFunc, 255);
75  }
76
77  void unloadPixels(bitmap_image &image, size_t start, size_t end) {
78      for (size_t i = start; i < end; i++) {
79          for (size_t j = 0; j < IMAGE_SIZE; j++) {
80              u_char pixel = pixels[i][j];
81              if (i == 0 || j == 0 || i == IMAGE_SIZE - 1 || j ==
    IMAGE_SIZE - 1)
82                  pixel = 0;
83              image.set_pixel(i, j, pixel, pixel, pixel);
84          }
85      }
86  }
```