



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ИМЕНИ Н.Э. БАУМАНА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)
(МГТУ им. Н.Э. БАУМАНА)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

НАПРАВЛЕНИЕ ПОДГОТОВКИ «09.03.04 Программная инженерия»

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №4

Тема: Распараллеливание алгоритма поворота
массива точек в двумерном растре

Студент: Сироткина П.Ю.

Группа: ИУ7-56Б

Оценка: _____

Преподаватель: Волкова Л.Л.

Москва, 2021 г.

Содержание

Введение	3
1 Аналитический раздел	5
1.1 Алгоритм поворота точек двумерного раstra	5
1.2 Вывод	7
2 Конструкторский раздел	8
2.1 Схемы алгоритмов	8
2.2 Вывод	12
3 Технологический раздел	13
3.1 Требования к ПО	13
3.2 Средства реализации	13
3.3 Листинг кода	13
3.4 Вывод	17
4 Экспериментальный раздел	18
4.1 Пример работы	18
4.2 Тестовые данные	19
4.3 Технические характеристики	19
4.4 Время выполнения алгоритмов	19
4.5 Вывод	22
Заключение	23
Список использованных источников	25

Введение

Многопоточность - способность центрального процессора или одного ядра в многоядерном процессоре одновременно выполнять несколько процессов или потоков, соответствующим образом поддерживаемых операционной системой. [1]

Другими словами, процесс, порождённый в операционной системе, может состоять из нескольких потоков, выполняющихся «параллельно», то есть без предписанного порядка во времени. При выполнении некоторых задач такое разделение может достичь более эффективного использования ресурсов вычислительной машины.

Процесс - это выполняющаяся программа и все её элементы: адресное пространство, глобальные переменные, регистры, стек, открытые файлы и т.д. *Поток* - это наименьшая последовательность запрограммированных команд, которые могут управляться планировщиком независимо. *Задача* - абстрактная концепция работы, которая должна быть выполнена.

Многопоточная парадигма стала более популярной с конца 1990-х годов, поскольку усилия по дальнейшему использованию параллелизма на уровне инструкций застопорились.

Смысл многопоточности - квазимногозадачность на уровне одного исполняемого процесса, то есть все потоки выполняются в адресном пространстве процесса. Кроме этого, все потоки процесса имеют не только общее адресное пространство, но и общие дескрипторы файлов. Выполняющийся процесс имеет как минимум один (главный) поток.

Преимущества многопоточности:

- Отзывчивость: один из потоков может обеспечивать быстрый отклик, пока другой поток заблокирован или работает;
- Разделение ресурсов: поток разделяет код, данные;
- Экономичность: переключение контекста значительно быстрее, чем у процессов;

- Масштабирование: однопоточные процессоры могут выполняются только на одном процессоре, многопоточные в многопроцессорных системах - на нескольких процессорах.

Недостатки многопоточности:

- Необходимо заботиться о корректном распараллеливании задачи, иначе это может привести к блокировкам и ошибкам доступа;
- Необходимо заботиться о разделении задачи на подзадачи сбалансированно, чтобы использовать ресурсы максимально эффективно;
- Нужна синхронизация потоков, если одна подзадача зависит от результата другой;
- Сложнее тестирование и отладка;
- Неправильное разделение данных приводит к ошибкам.

Целью лабораторной работы является изучение и реализация принципов многопоточности на примере алгоритма поворота фигуры, представленной в виде массива точек, в двумерном растре.

Для достижения поставленной цели необходимо выполнить следующие **задачи**:

1. Изучить понятие параллельных вычислений, а также принцип работы алгоритма поворота массива точек в двумерном растре.
2. Привести схему рассматриваемого алгоритма в последовательном и параллельном вариантах.
3. Описать используемые структуры данных.
4. Реализовать последовательный и параллельный алгоритм поворота массива точек в двумерном растре.
5. Протестировать разработанное ПО.
6. Сравнить временные характеристики реализованных версий алгоритма экспериментально.
7. На основании проделанной работы сделать выводы.

1 Аналитический раздел

В данной лабораторной работе многопоточность изучается на примере алгоритма поворота фигуры, представленной в виде массива точек в двумерном растре.

Эта задача является весьма актуальной, т.к. компьютерная графика стала неотъемлемой частью повседневной интернет-жизни человека, и существует потребность в быстром рендеринге изображения, например, при анимации поворота фигуры на двумерном растре.

Как известно, в экранной плоскости изображение представляет из себя набор пикселей (точек). Очевидно, что какая-либо фигура - это тоже набор точек экранной плоскости, и для поворота фигуры требуется над каждой ее точкой произвести преобразование для получения новой позиции.

Если использовать один поток для рендеринга изображения, то при большом количестве и сложности фигур изображение будет генерироваться ощутимо долго, что будет приносить человеку дискомфорт при восприятии, однако если распараллелить этот процесс, т.е. параллельно генерировать части изображения (т.к. эта операция выполняется независимо для каждой точки), то это может дать колоссальной прирост производительности.

Перейдем к рассмотрению алгоритма поворота точек двумерного растра[2].

1.1 Алгоритм поворота точек двумерного растра

Пусть необходимо повернуть точку $P(x, y)$ вокруг начала координат O на угол ϕ . Изображение новой точки обозначим через $P'(x', y')$. Всегда существуют четыре числа a, b, c, d , такие, что новые координаты могут быть вычислены по значениям старых координат x и y из следующей системы уравнений:

$$\begin{cases} x' = a \cdot x + b \cdot y; \\ y' = c \cdot x + d \cdot y, \end{cases} \quad (1.1)$$

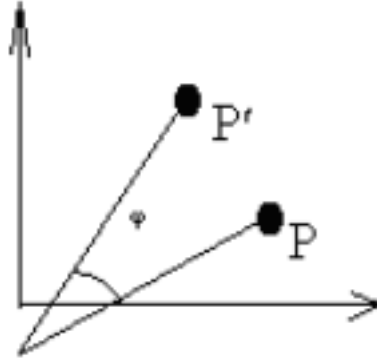


Рисунок 1.1 — Пример расположения точек P и P'

Для получения значений a , b , c , d рассмотрим вначале точку $P(x, y) = (1, 0)$. Полагая $x = 1$ и $y = 0$ в уравнении (1.1), получим:

$$\begin{cases} x = a; \\ y = c, \end{cases}$$

Но в этом простом случае, как это видно из рис. 1.1, значения x' и y' равны соответственно $\cos(\phi)$ и $\sin(\phi)$. Тогда имеем:

$$\begin{cases} a = \cos(\phi); \\ c = \sin(\phi), \end{cases}$$

Аналогичным образом рассматривая точку $P(x, y) = (0, 1)$, получим:

$$\begin{cases} b = -\sin(\phi); \\ d = \cos(\phi), \end{cases}$$

Тогда вместо системы уравнений (2.1) можем записать

$$\begin{cases} x_{new} = x \cdot \cos(\phi) - y \cdot \sin(\phi); \\ y_{new} = x \cdot \sin(\phi) + y \cdot \cos(\phi), \end{cases} \quad (1.2)$$

Система уравнений (1.2) описывает поворот вокруг точки O - начала системы координат $(0, 0)$, но часто это не то, что нужно. Если

требуется выполнить поворот относительно заданной точки (x_c, y_c) , то в этих уравнениях можно заменить: x - на $(x - x_c)$, y - на $(y - y_c)$, x' - на $(x' - x_c)$ и y' - на $(y' - y_c)$.

Система уравнений, которая описывает поворот точки вокруг некоторой точки с координатами (x_c, y_c) :

$$\begin{cases} x_{new} = x_c + (x - x_c) \cdot \cos(\phi) + (y - y_c) \cdot \sin(\phi); \\ y_{new} = y_c - (x - x_c) \cdot \sin(\phi) + (y - y_c) \cdot \cos(\phi), \end{cases}$$

Часто удобно представлять подобные преобразования в виде матрицы преобразований:

$$M = \begin{pmatrix} \cos(\phi) & -\sin(\phi) & 0 \\ \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Таким образом, распараллеливание будет заключаться в том, что массив точек будет разбиваться на подмассивы, для каждого из которых независимо от других будет решаться задача преобразования.

1.2 Вывод

В данном разделе была проанализирована предметная область, установлена актуальность задачи, также был рассмотрен алгоритм задачи, которая будет подвергнута распараллеливанию.

2 Конструкторский раздел

В данном разделе представлены последовательная и параллельная схемы алгоритма, описанного в аналитической части.

2.1 Схемы алгоритмов

На рисунке 2.1 представлена схема последовательного алгоритма поворота точек двумерного растра.

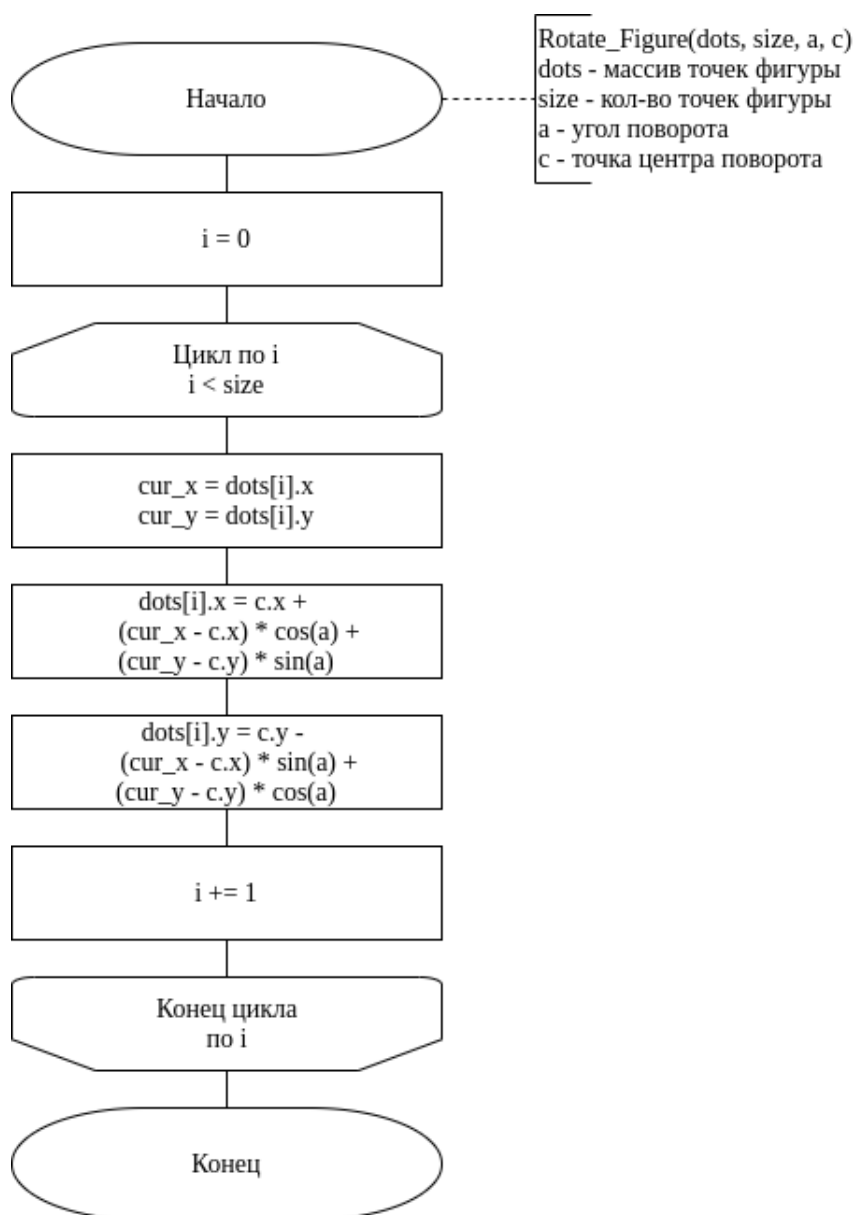


Рисунок 2.1 — Схема последовательного алгоритма поворота точек двумерного растра

На рисунке 2.2 представлена схема распараллеливания алгоритма поворота точек двумерного растра.

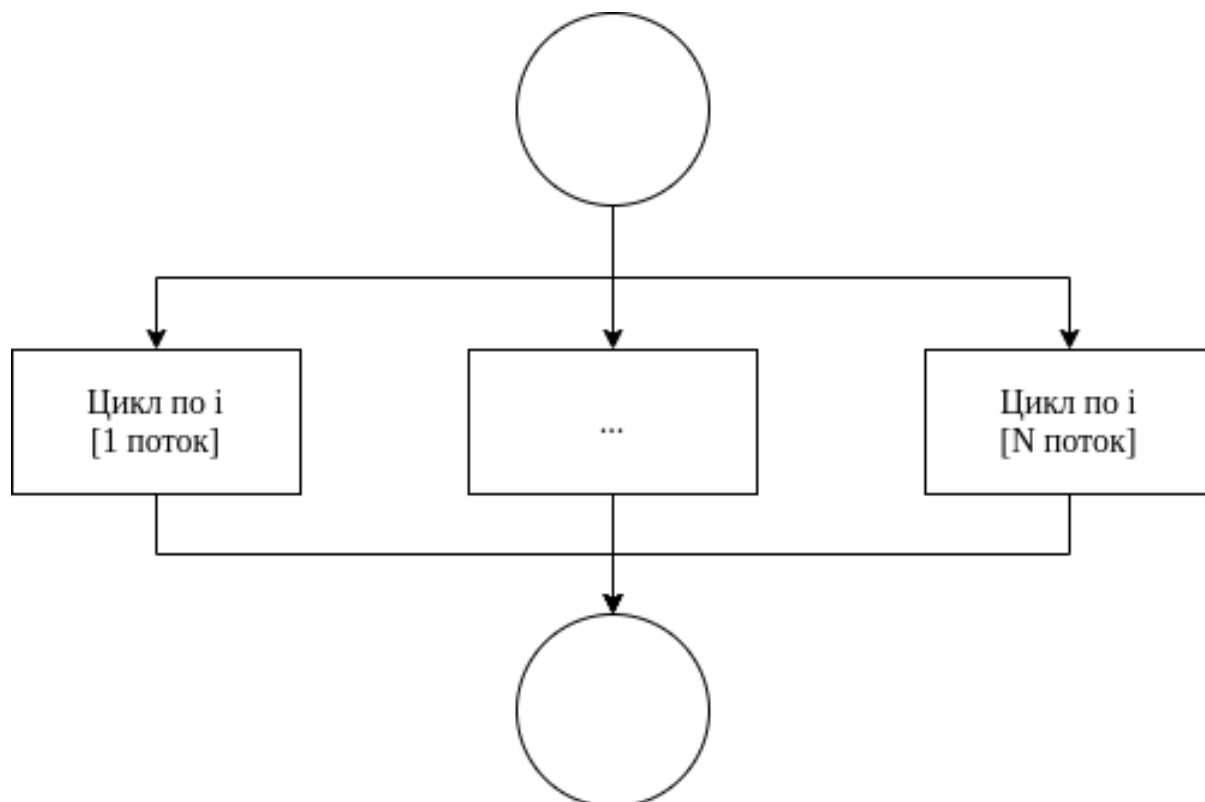


Рисунок 2.2 — Схема распараллеливания алгоритма поворота точек двумерного растра

На рисунке 2.3 представлена схема распараллеленного алгоритма поворота точек двумерного растра.

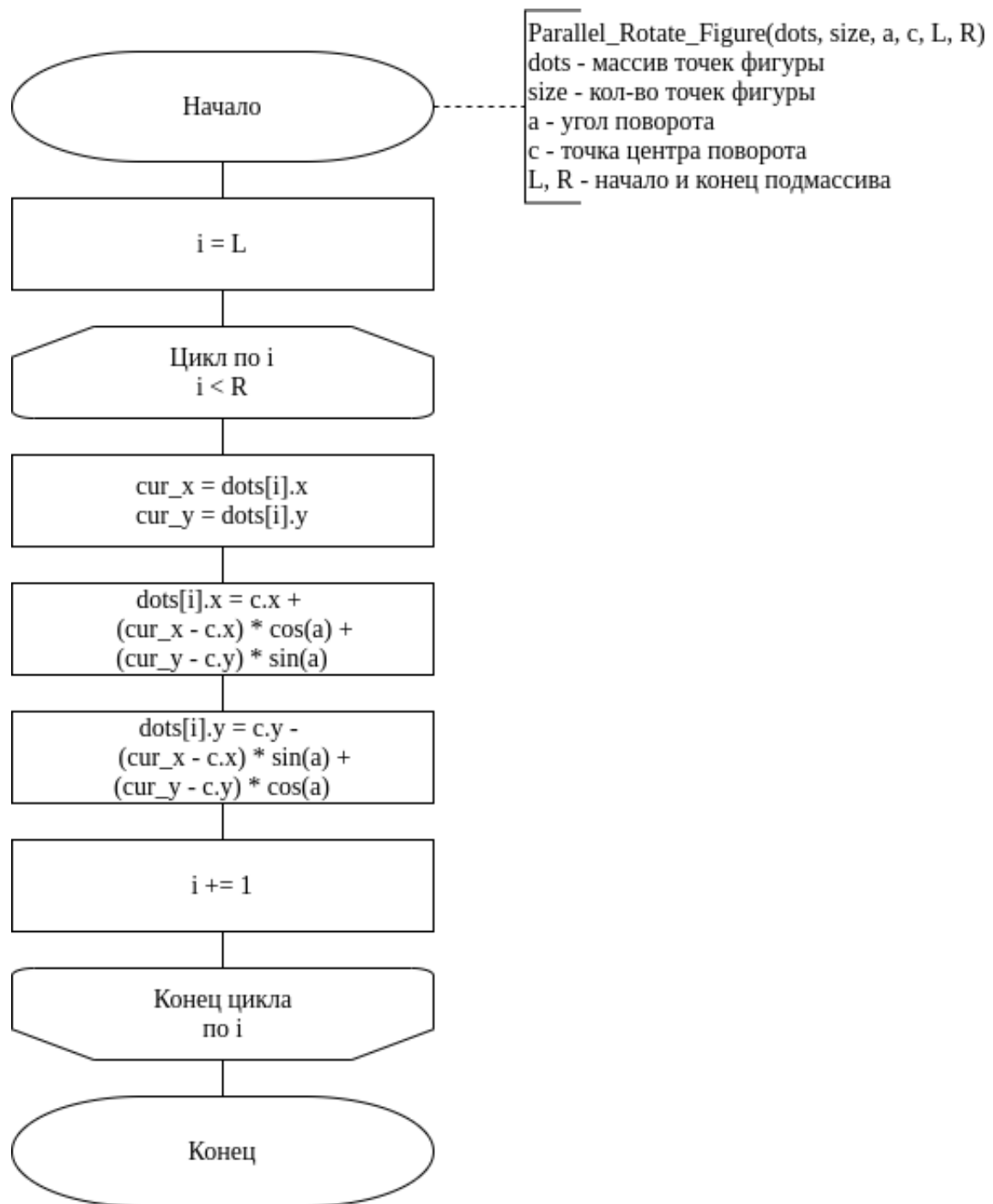


Рисунок 2.3 — Схема распараллеленного алгоритма поворота точек двумерного растра

На рисунке 2.4 представлена схема функции создания потоков и запуска параллельных реализаций вычислений.

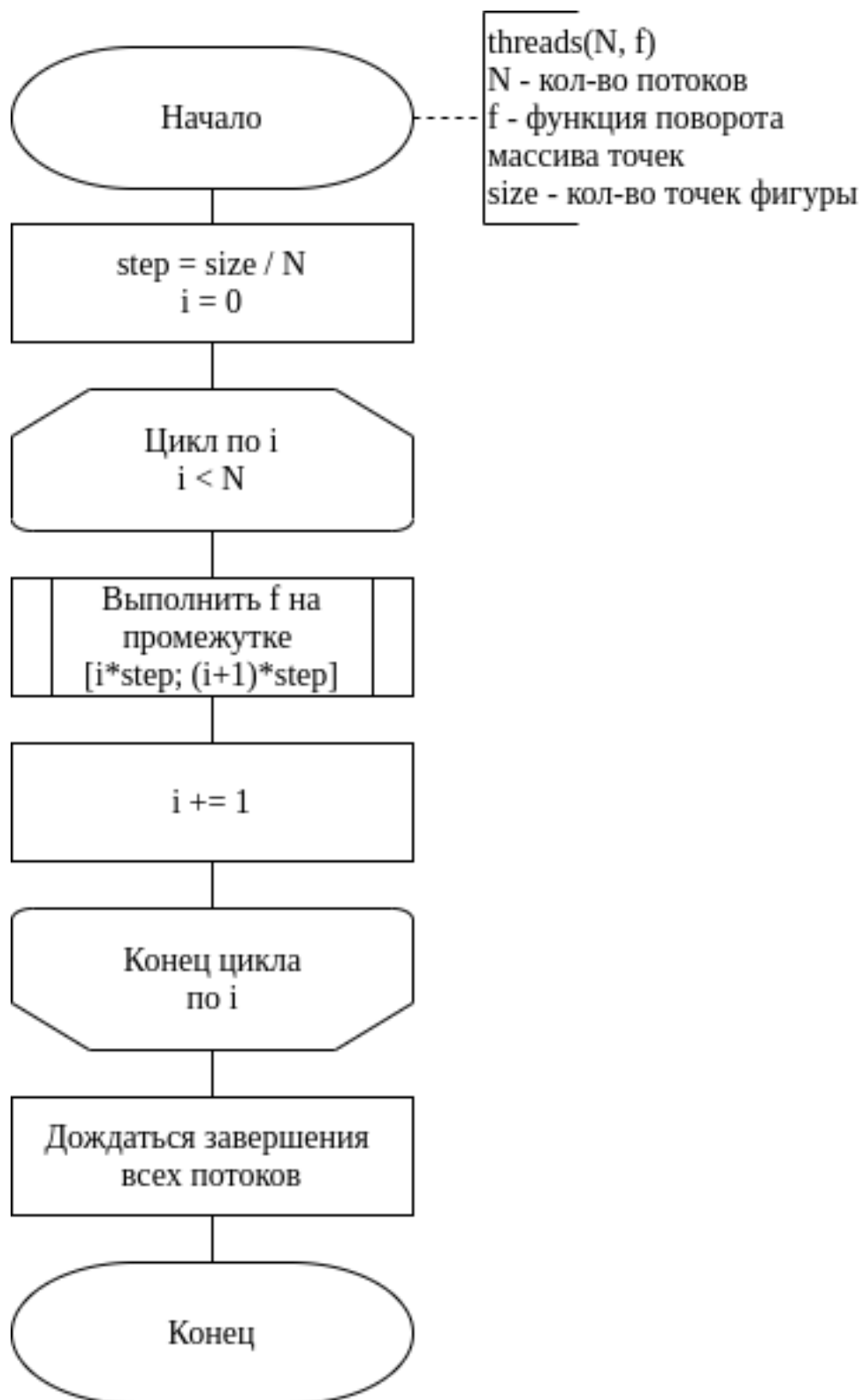


Рисунок 2.4 — Схема функции создания потоков и запуска параллельных реализаций вычислений

2.2 Вывод

На основе теоретических данных, полученных из аналитического раздела, были разработаны схемы требуемых алгоритмов.

3 Технологический раздел

В данном разделе приведены требования к программному обеспечению, средства реализации и листинги кода.

3.1 Требования к ПО

К программе предъявляются следующие требования:

- На вход подается массив точек фигуры двумерного раstra в виде пар чисел (x, y) , угол ϕ , на который необходимо повернуть фигуру, а также точка центра поворота (x_c, y_c) ;
- На выходе - преобразованный массив точек фигуры двумерного раstra, определяющий положение фигуры на экранной плоскости, повернутой на угол ϕ относительно точки центра поворота (x_c, y_c) .
- Программа должна корректно реагировать на ошибки пользователя и невалидные данные о фигуре или угле поворота.

3.2 Средства реализации

В качестве языка программирования для реализации лабораторной работы был выбран язык C. Выбор этого языка обусловлен его быстродействием и эффективностью, это легко читаемый, лаконичный и гибкий язык. Также выбор обусловлен моим личным желанием получить больше практики написания программ на этом языке. Этот язык также поддерживает функцию нативных потоков, что дает возможность реализации поставленной задачи.

3.3 Листинг кода

В листинге 3.1 представлены используемые структуры данных.

Листинг 3.1 — Используемые структуры данных

```
1 typedef struct
2 {
3     int x;
4     int y;
5 } point_t;
6
```

```

7 typedef struct
8 {
9     point_t *dots;
10    int size;
11 } figure_t;
12
13 typedef struct {
14     figure_t *figure;
15     int angle;
16     point_t center;
17 } rotate_args_t;
18
19 typedef struct {
20     rotate_args_t *rotate_args;
21     int t_id;
22     int size;
23     int cnt_threads;
24 } pthread_args_t;

```

В листинге 3.2 представлена последовательная реализация алгоритма поворота точек фигуры в двумерном растре.

Листинг 3.2 — Последовательная реализация алгоритма поворота

```

25 void rotate(rotate_args_t *args)
26 {
27     int cur_x, cur_y;
28     double a = (double) args->angle / 180 * M_PI;
29     for (int i = 0; i < args->figure->size; i++)
30     {
31         cur_x = args->figure->dots[i].x;
32         cur_y = args->figure->dots[i].y;
33
34         args->figure->dots[i].x = args->center.x +
35             (cur_x - args->center.x) * cos(a) +
36             (cur_y - args->center.y) * sin(a);
37
38         args->figure->dots[i].y = args->center.y -
39             (cur_x - args->center.x) * sin(a) +
40             (cur_y - args->center.y) * cos(a);
41     }
42 }

```

В листинге 3.3 представлена реализация алгоритма распределения потоков.

Листинг 3.3 — Реализация алгоритма распределения потоков

```
43 int start_threading(rotate_args_t *args, const int cnt_threads)
44 {
45     pthread_t *threads = malloc(cnt_threads * sizeof(pthread_t));
46
47     if (!threads)
48         return ALLOCATE_ERROR;
49
50     pthread_args_t *p_args = malloc(sizeof(pthread_args_t) * cnt_threads);
51
52     if (!p_args)
53     {
54         free(threads);
55         return ALLOCATE_ERROR;
56     }
57
58     for (int i = 0; i < cnt_threads; i++)
59     {
60         p_args[i].rotate_args = args;
61         p_args[i].t_id = i;
62         p_args[i].size = args->figure->size;
63         p_args[i].cnt_threads = cnt_threads;
64
65         pthread_create(&threads[i], NULL, parallel_rotate, &p_args[i]);
66     }
67
68     for (int i = 0; i < cnt_threads; i++)
69         pthread_join(threads[i], NULL);
70
71     free(p_args);
72     free(threads);
73
74     return OK;
75 }
```

В листинге 3.4 представлена распараллеленная реализация алгоритма поворота точек фигура в двумерном растре.

Листинг 3.4 — Распараллеленная реализация алгоритма поворота

```

76 void *parallel_rotate(void *args)
77 {
78     pthread_args_t *argsp = args;
79
80     int i_start = argsp->t_id * (argsp->size / argsp->cnt_threads);
81     int i_end = (argsp->t_id + 1) * (argsp->size / argsp->cnt_threads);
82
83     int cur_x, cur_y;
84     double a = (double) argsp->rotate_args->angle / 180 * M_PI;
85
86     for (int i = i_start; i < i_end; i++)
87     {
88         cur_x = argsp->rotate_args->figure->dots[i].x;
89         cur_y = argsp->rotate_args->figure->dots[i].y;
90
91         argsp->rotate_args->figure->dots[i].x =
92             argsp->rotate_args->center.x +
93             (cur_x - argsp->rotate_args->center.x) * cos(a) +
94             (cur_y - argsp->rotate_args->center.y) * sin(a);
95
96         argsp->rotate_args->figure->dots[i].y =
97             argsp->rotate_args->center.y -
98             (cur_x - argsp->rotate_args->center.x) * sin(a) +
99             (cur_y - argsp->rotate_args->center.y) * cos(a);
100     }
101     return NULL;

```

В листинге 3.5 представлена реализация ассемблерной вставки, которая ведет посчет тиков процессора[3]:

Листинг 3.5 — "Ассемблерная вставка для замера тиков процессора"

```

1 uint64_t tick(void)
2 {
3     uint32_t high, low;
4     __asm__ __volatile__(
5         "rdtsc\n"
6         "movl %%edx, %0\n"
7         "movl %%eax, %1\n"
8         : "=r"(high), "=r"(low) : "%rax", "%rbx", "%rcx", "%rdx");
9

```



```
10     uint64_t ticks = ((uint64_t)high << 32) | low;
11
12     return ticks;
13 }
```

3.4 Вывод

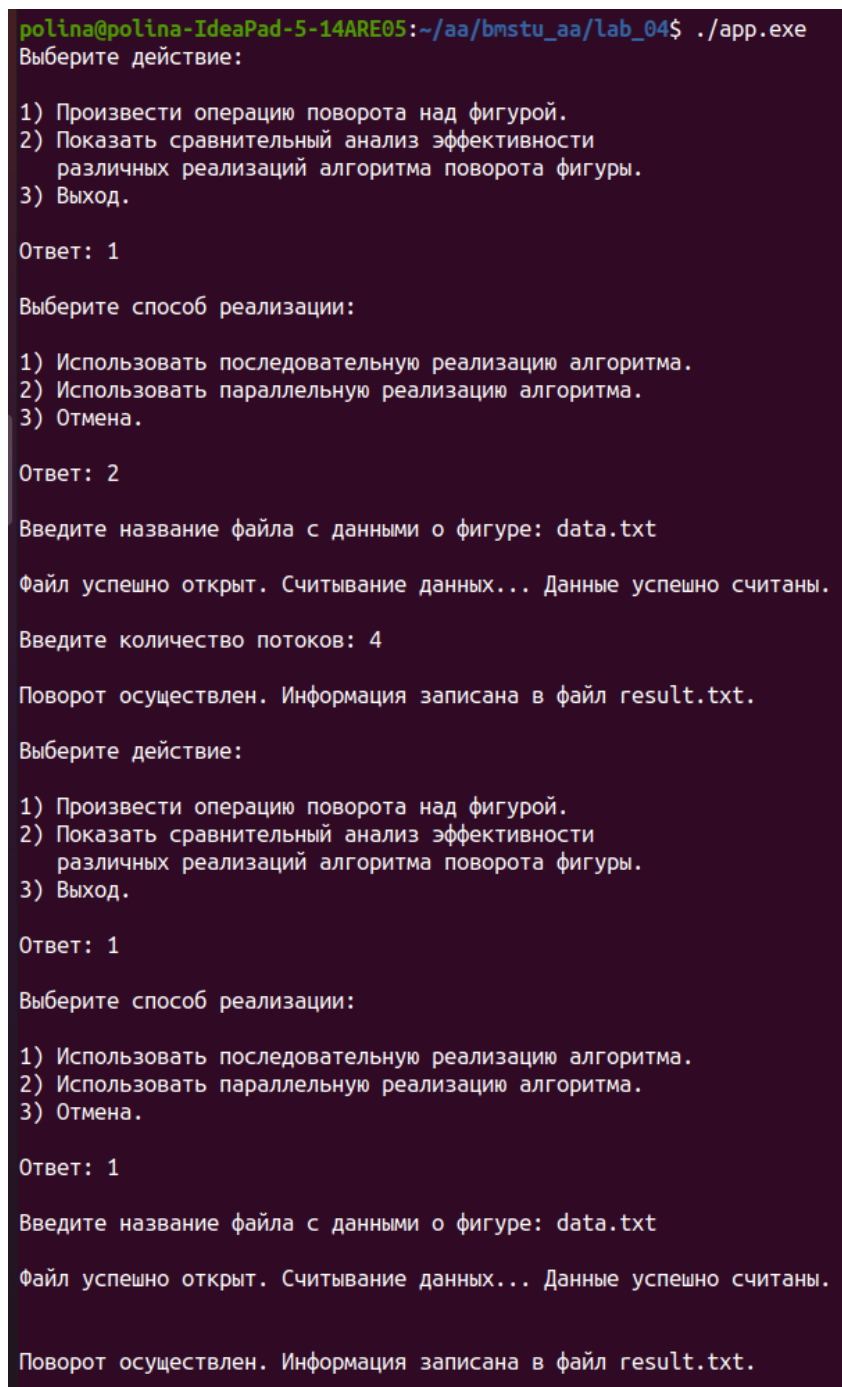
В данном разделе были реализованы последовательная и распараллеленная версии алгоритма поворота точек фигуры в двумерном растре.

4 Экспериментальный раздел

В данном разделе представлен пользовательский интерфейс, а также проведена оценка эффективности различных реализаций алгоритма.

4.1 Пример работы

На рисунке 4.1 приведен пример работы программы.



```
polina@polina-IdeaPad-5-14ARE05:~/aa/bmstu_aa/lab_04$ ./app.exe
Выберите действие:

1) Произвести операцию поворота над фигурой.
2) Показать сравнительный анализ эффективности различных реализаций алгоритма поворота фигуры.
3) Выход.

Ответ: 1

Выберите способ реализации:

1) Использовать последовательную реализацию алгоритма.
2) Использовать параллельную реализацию алгоритма.
3) Отмена.

Ответ: 2

Введите название файла с данными о фигуре: data.txt

Файл успешно открыт. Считывание данных... Данные успешно считаны.

Введите количество потоков: 4

Поворот осуществлен. Информация записана в файл result.txt.

Выберите действие:

1) Произвести операцию поворота над фигурой.
2) Показать сравнительный анализ эффективности различных реализаций алгоритма поворота фигуры.
3) Выход.

Ответ: 1

Выберите способ реализации:

1) Использовать последовательную реализацию алгоритма.
2) Использовать параллельную реализацию алгоритма.
3) Отмена.

Ответ: 1

Введите название файла с данными о фигуре: data.txt

Файл успешно открыт. Считывание данных... Данные успешно считаны.

Поворот осуществлен. Информация записана в файл result.txt.
```

Рисунок 4.1 — Пример работы программы

4.2 Тестовые данные

В таблице 4.1 приведены тестовые данные, на которых было протестировано разработанное ПО. Все тесты были успешно пройдены.

Таблица 4.1 — Таблица тестовых данных

Массив точек	Угол	Центр	Ожидание	Результат
[(1;1), (2;2), (3;3)]	0°	(0;0)	[(1;1), (2;2), (3;3)]	[(1;1), (2;2), (3;3)]
[(1;1), (2;2), (3;3)]	360°	(0;0)	[(1;1), (2;2), (3;3)]	[(1;1), (2;2), (3;3)]
[(1;1), (2;2), (3;3)]	50°	(0;0)	[(1;0), (2;0), (4;0)]	[(1;0), (2;0), (4;0)]
[(1;1), (2;2), (3;3)]	50°	(5;5)	[(0;5), (0;5), (2;5)]	[(0;5), (0;5), (2;5)]
[(1;1), (2;2), (3;3)]	-50°	(0;0)	[(0;1), (0;2), (0;4)]	[(0;1), (0;2), (0;4)]
[(1;1), (2;2), (3;3)]	-50°	(5;5)	[(5;0), (5;0), (5;2)]	[(5;0), (5;0), (5;2)]
[(11;11)]	90°	(0;0)	[(-11;11)]	[(-11;11)]
[(11;11)]	90°	(0;0)	[(-21;11)]	[(-21;11)]

4.3 Технические характеристики

Технические характеристики машины, на которой выполнялось тестирование:

- Операционная система: Ubuntu[4] Linux[5] 20.04 64-bit.
- Оперативная память: 16 Gb.
- Процессор: AMD(R) Ryzen(TM)[6] 5 4500U CPU @ 2.3 CHz.
- Количество физических ядер: 6.
- Технология HTT: отключена.

4.4 Время выполнения алгоритмов

В таблице 4.1 представлены замеры процессорного времени в тиках для массива точек размерностью 1920000 (стандартное разрешение 1600 x 1200).

Таблица 4.2 — Таблица замеров процессорного времени (в тиках) для массива размерностью 1920000 точек

Кол-во потоков	Последовательный алгоритм	Распараллеленный алгоритм
1	160 463 146	172 594 611
2	155 682 533	95 950 866
4	156 871 101	63 696 538
6	157 084 139	50 082 076
12	165 735 160	47 568 720
18	163 151 738	46 918 113
24	167 200 297	46 370 355

На рисунке 4.2 приведен график зависимости процессорного времени от количества потоков при размерности массива точек равным 1920000.

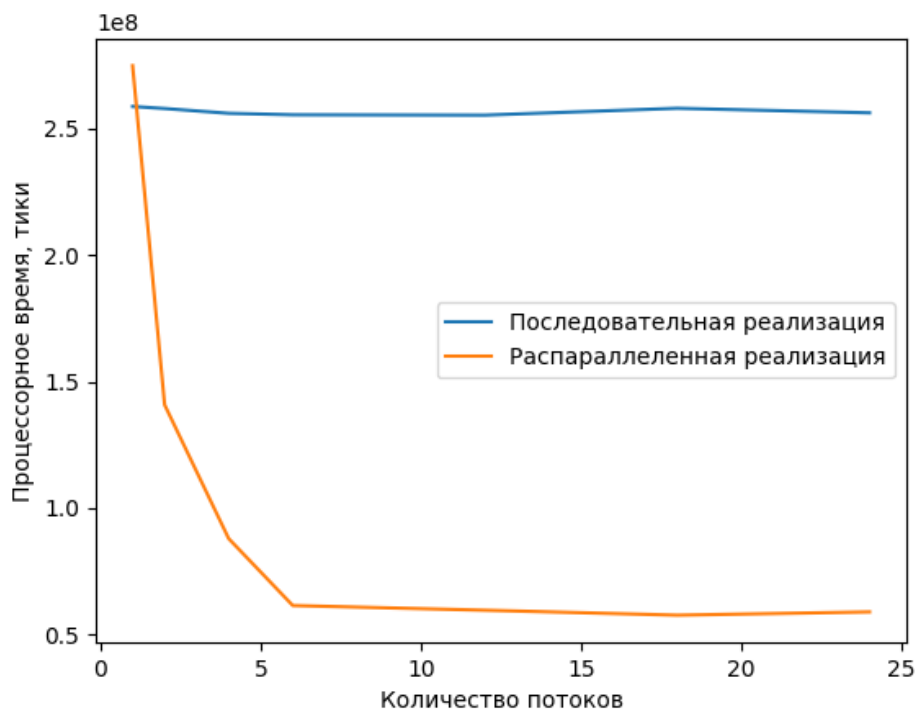


Рисунок 4.2 — График зависимости процессорного времени от количества потоков при размерности массива точек равным 1920000

В таблице 4.2 приведены замеры процессорного времени для количества потоков, равного количеству физических ядер процессора.

Таблица 4.3 — Таблица замеров процессорного времени для кол-ва потоков, равного кол-ву физических ядер процессора

Длина массива	Последовательный алгоритм	Распараллеленный алгоритм
1000000	134 060 378	41 350 496
1200000	163 374 421	42 446 985
1400000	190 399 071	47 994 463
1600000	216 826 883	53 237 655
1800000	240 351 709	56 796 711
1920000	256 190 597	59 028 938

На рисунке 4.3 приведен график зависимости процессорного времени от размерности массива точек фигуры для последовательной и распараллеленной реализаций (количество потоков везде равно 6).

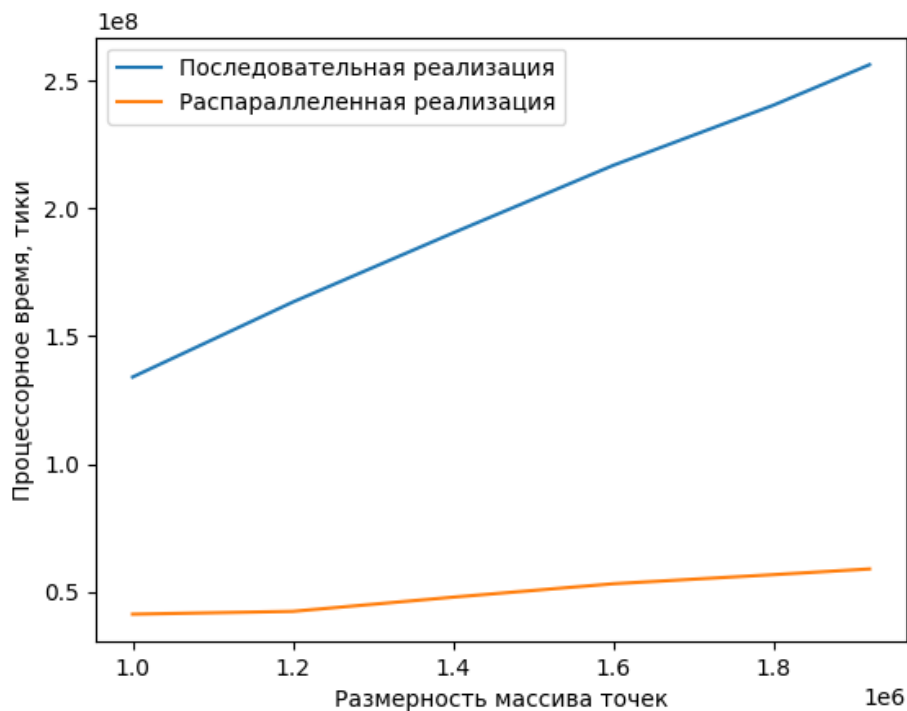


Рисунок 4.3 — График зависимости процессорного времени от размерности массив (количество потоков равно 6)

4.5 Вывод

Из полученных выше результатов замеров процессорного времени можно сделать вывод, что использование многопоточности однозначно может дать существенный прирост эффективности. В данной лабораторной работе в качестве примера рассматривался алгоритм поворота массива точек фигуры в двумерном растре, и при использовании многопоточности удалось добиться как минимум 4-кратного увеличения производительности.

Максимальное количество потоков, при которых выполнялись замеры - 24 (по 4 потока на 6 физических ядер), и при этом количестве производительность выросла в 4 раза, однако нельзя утверждать, что так будет всегда - если использовать слишком много потоков и часть данных для обработки в результате будет малой, то больше времени будет тратиться на создание и завершение потока, чем на обработку части информации, поэтому не стоит злоупотреблять количеством потоков.

Также можно сделать вывод, что использовать 1 поток неэффективно, т.к. по сути это эквивалентно обыкновенному последовательному алгоритму, однако сверх этого тратятся ресурсы системы на создание и закрытие потока, и в результате программа работает медленнее.

Заключение

В ходе выполнения лабораторной работы были выполнены поставленные задачи, а именно:

1. Было изучено понятие параллельных вычислений, а также принцип работы алгоритма поворота массива точек в двумерном растре.
2. Были построены схемы рассматриваемого алгоритма в последовательном и параллельном вариантах.
3. Были описаны используемые структуры данных.
4. Реализованы последовательный и параллельный алгоритм поворота массива точек в двумерном растре.
5. Проведено тестирование разработанного ПО.
6. Проведен сравнительный анализ временных характеристик реализованных версий алгоритма экспериментально.

Экспериментально были подтверждены различия между последовательной и распараллеленной реализацией алгоритма:

Из полученных выше результатов замеров процессорного времени можно сделать вывод, что использование многопоточности однозначно может дать существенной прирост эффективности. В данной лабораторной работе в качестве примера рассматривался алгоритм поворота массива точек фигуры в двумерном растре, и при использовании многопоточности удалось добиться как минимум 4-кратного увеличения производительности.

Максимальное количество потоков, при которых выполнялись замеры - 24 (по 4 потока на 6 физических ядер), и при этом количестве производительность выросла в 4 раза, однако нельзя утверждать, что так будет всегда - если использовать слишком много потоков и часть данных для обработки в результате будет малой, то больше времени будет тратиться на создание и завершение потока, чем на обработку части информации, поэтому не стоит злоупотреблять количеством потоков.

Также можно сделать вывод, что использовать 1 поток неэффективно, т.к. по сути это эквивалентно обыкновенному последовательному

алгоритму, однако сверх этого тратятся ресурсы системы на создание и закрытие потока, и в результате программа работает медленнее.

Таким образом, можно сделать вывод, что количество потоков желательно выбирать взвешенно - надо избегать напрасной траты ресурсов системы, например, не создавать один поток или не создавать слишком много потоков. Нужно отследить, при каком минимальном количестве потоков достигается минимально приемлемый выигрыш во времени, т.е. выбрать пик производительности при возможном минимуме потоков, и в дальнейшем использовать это число.

Также следует учитывать, что наиболее эффективно распараллеливать анализ больших данных, т.к. маленькие задачи и так выполняются достаточно быстро с учетом высокой производительности современных компьютеров.

Список использованных источников

1. Многопоточность. — [Электронный ресурс]. Режим доступа: <https://ru.bmstu.wiki/Многопоточность.>, 12.10.2021.
2. Двумерный алгоритм преобразование в новые координаты. — [Электронный ресурс]. Режим доступа: <http://www.codenet.ru/progr/cg/lec12.php.>, 12.10.2021.
3. C/C++: как измерять процессорное время. — [Электронный ресурс]. Режим доступа: [https://habr.com/ru/post/282301/.](https://habr.com/ru/post/282301/), 16.09.2021.
4. Ubuntu. — [Электронный ресурс]. Режим доступа: <https://ru.wikipedia.org/wiki/Ubuntu.>, 16.09.2021.
5. Linux. — [Электронный ресурс]. Режим доступа: <https://ru.wikipedia.org/wiki/Linux.>, 16.09.2021.
6. Процессор AMD Ryzen(TM) 5. — [Электронный ресурс]. Режим доступа: [https://shop.lenovo.ru/product/81YM007FRU/.](https://shop.lenovo.ru/product/81YM007FRU/), 21.09.2021.