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»

Отчет по лабораторной работе №4 по дисциплине "Анализ алгоритмов"

Тема Параллельный поворот фигуры

Студент Петрова А.А.

Группа ИУ7-56Б

Оценка (баллы) _____

Преподаватели Волкова Л.Л.

Москва — 2021 г.

Содержание

Введение

Многопоточность - это специализированная форма многозадачности, и многозадачность - это функция, которая позволяет вашему компьютеру одновременно запускать две или несколько программ. В общем, существует два типа многозадачности: основанные на процессах и потоки [1].

Многозадачность на основе процессов управляет одновременным выполнением программ. Многозадачность на основе потоков связана с одновременным выполнением частей одной и той же программы.

Многопоточная программа содержит две или несколько частей, которые могут запускаться одновременно. Каждая часть такой программы называется потоком, и каждый поток определяет отдельный путь выполнения.

Целью данной лабораторной работы являются изучение и реализация многопоточности на основе алгоритмов компьютерной графики, в частности алгоритма поворота двумерной фигуры, представленной в виде массива точек.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- изучить понятие параллельных вычислений;
- реализовать последовательный и параллельный алгоритмы поворота фигуры;
- сравнить временные характеристики реализованных алгоритмов экспериментально.

1 Аналитическая часть

В этом разделе будет проанализирована предметная область, установлена актуальность задачи, а также будет рассмотрен алгоритм задачи, которая будет подвергнута распараллеливанию.

Задача поворота фигуры, представленной в виде массива точек в двумерном растре, 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 плоскоти, 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 коллосальной прирост производительности.

1.1 Алгоритм поворота точек на плоскости

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

$$\begin{cases} x' = a \cdot x + b \cdot y \\ y' = c \cdot x + d \cdot y \end{cases} \tag{1.1}$$

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

$$\begin{cases} x' = a \\ y' = c \end{cases} \tag{1.2}$$

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

$$\begin{cases} a = cos(\phi) \\ c = sin(\phi) \end{cases} \tag{1.3}$$

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

$$\begin{cases} b = -sin(\phi) \\ d = cos(\phi) \end{cases} \tag{1.4}$$

Тогда всесто системы уравнений 1.1 можно записать:

$$\begin{cases} x' = cos(\phi) \cdot x - sin(\phi) \cdot y \\ y' = sin(\phi) \cdot x + cos(\phi) \cdot y \end{cases} \tag{1.5}$$

Система уравнений 1.5 описывает поворот вокруг точки O - начала системы координат, но часто нужно нужно выполнить поворот относительно заданной точки (x_c, y_c). Тогда система 1.5 примет следующий вид [2]:

$$\begin{cases} x' = x_c + (x - x_c) \cdot cos(\phi) - (y - y_c) \cdot sin(\phi) \\ y' = y_c + (x - x_c) \cdot sin(\phi) + (y - y_c) \cdot cos(\phi) \end{cases} \tag{1.6}$$

Часто такие преобразования удобно представить в виде матричных преобразований:

$$M = \begin{pmatrix} cos(\phi) & -sin(\phi) & 0 \\ sin(\phi) & cos(\phi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{1.7}$$

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.

Входными данными для программы являются:

- плоская фигура, представляющая собой массив точек;
- угол поворота в градусах.

Выходные данные: повёрнутая относительно начала координат фигура в виде преобразованного массива точек.

Ограничения, в рамках которых будет работать программа:

- фигура задаётся на плоскости;
- поворот происходит относительно начала координат.

Функциональные требования к ПО:

- ПО должно содержать 2 раздела: пользовательский (ручной ввод) и экспериментальный (для замеров времени);
- ПО должно выводить координаты точек после их поворота относительно начала координат;
- ПО должно выводить потраченное время;
- корректность данных в пользовательском разделе не проверяется.

2 Конструкторская часть

В этом разделе на основе теоретических данных, полученных в аналитическом разделе, будут построены схемы исследуеммых алгоритмов. А также будут описаны: структуры данных, используемые в алгоритмах, способы тестирования и классы эквивалентности, память, используемая алгоритмом, и структура ПО.

2.1 Схемы алгоритмов

На рисунке 2.1 представлена схема последовательного алгоритма поворота фигуры.

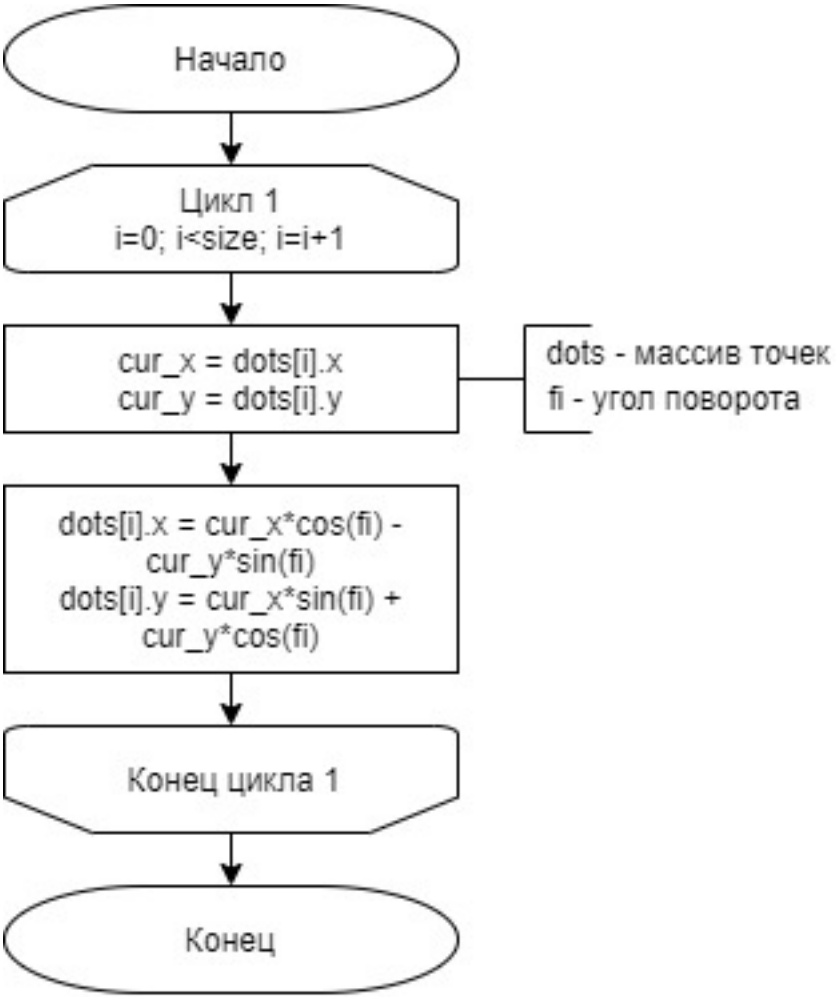

Рис. 2.1: Схема последовательного алгоритма поворота фигуры

На рисунках 2.2, 2.3 представлена схема распараллеливания алгоритма поворота фигуры.

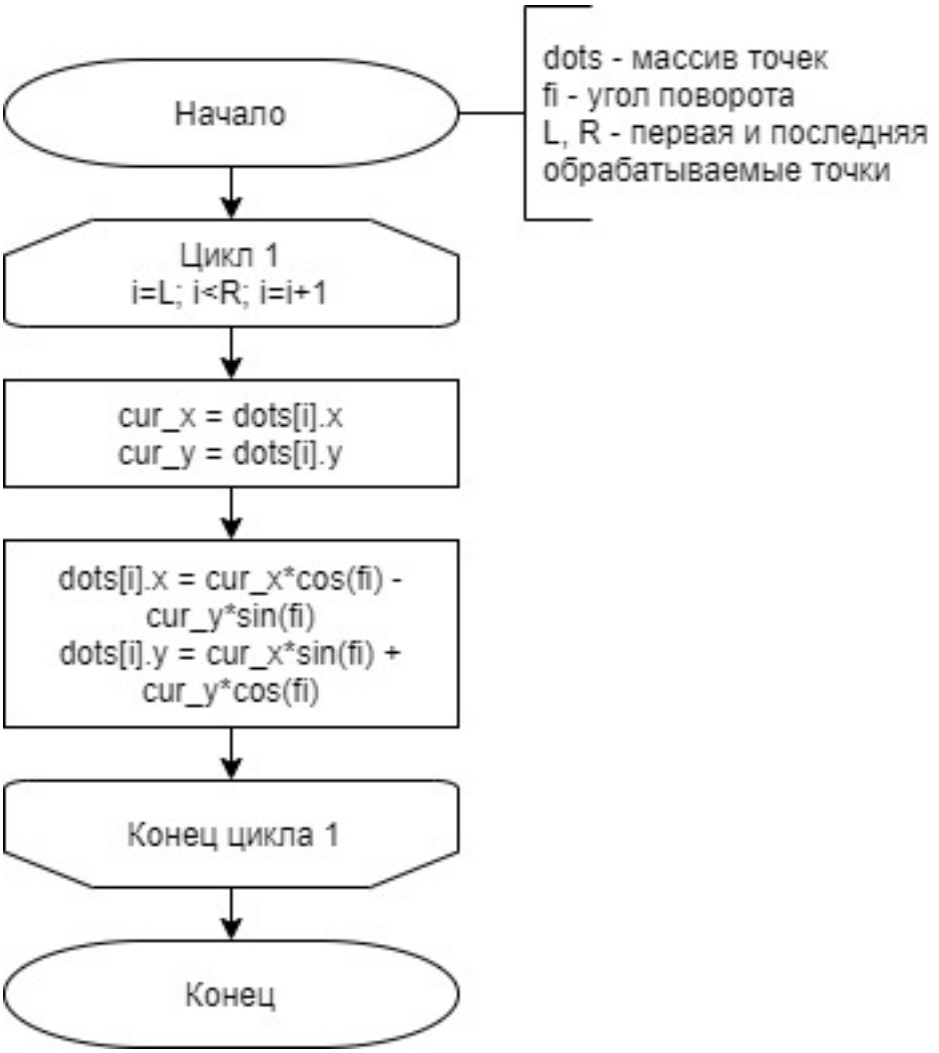

Рис. 2.2: Схема многопоточного алгоритма поворота фигуры

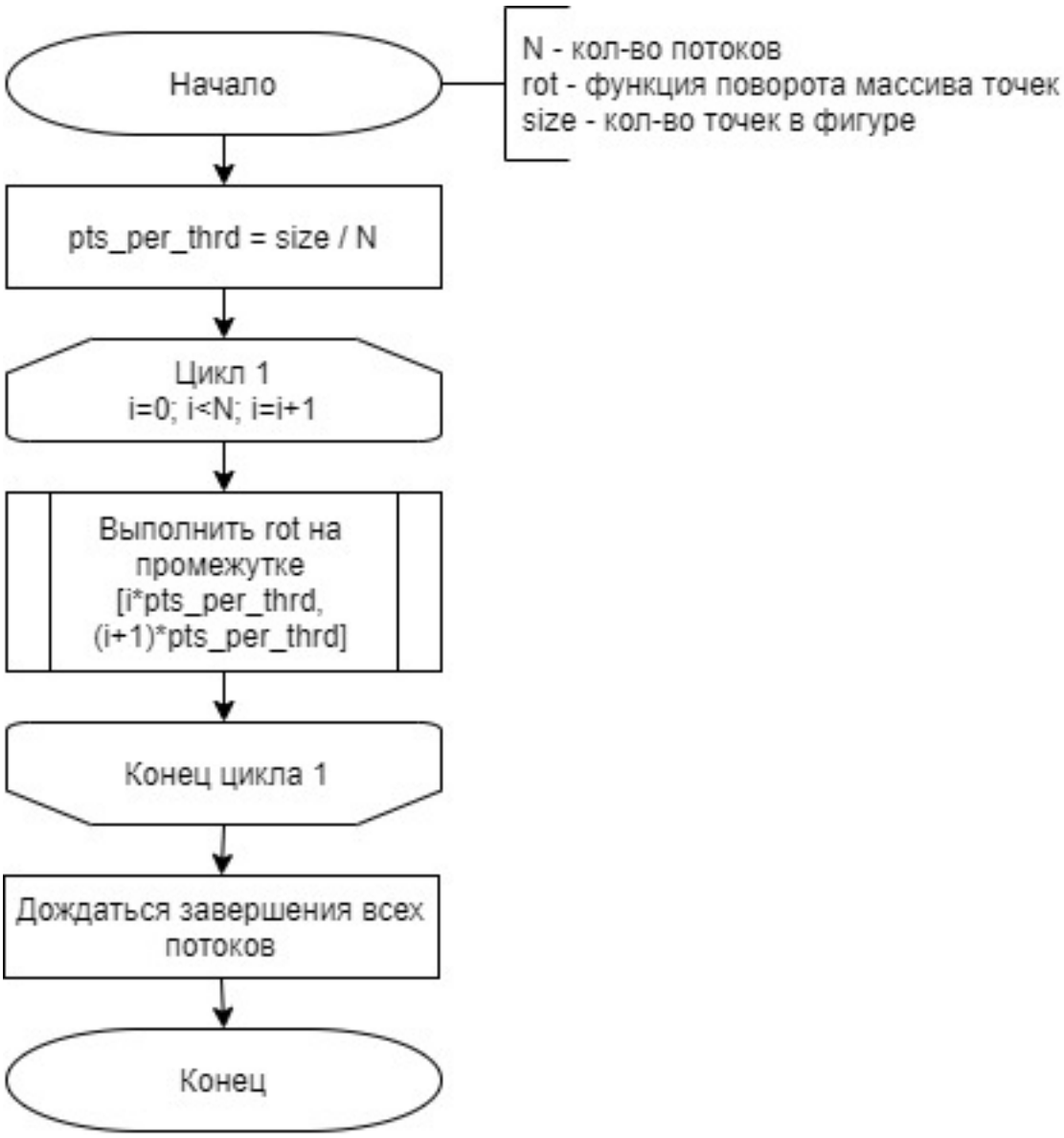

Рис. 2.3: Схема создания и запуска потоков

2.2 Структуры данных

Для хранения фигуры будет использоваться структура, полями которой являются массив точек и количество точек (размер массива).

Элементами массива точек в свою очередь будут также являться структуры, полями которых являются координаты этих точек по x и y.

Все создаваемые потоки также будут храниться в массиве.

2.3 Тестирование и классы эквивалентности

Для проверки работоспособности ПО будет применяться функциональное тестирование.

Классы эквивалентности:

- положительный угол поворота;
- отрицательный угол поворота;
- угол поворота - нулевой;
- угол поворота равен 360 градусам.

2.4 Используемая память

Все вышеописанные массивы будут динамическими. В связи с этим, нужно будет контролировать проблемы с памятью, в частности при выделении памяти под массивы и её освобождении из-под них.

Из всего вышесказанного следует, что количество памяти, необходимой алгоритмам, прямо пропорционально количеству точек фигуры и количеству потоков, используемых для распараллеливания.

2.5 Структура ПО

ПО будет состоять из набора функций:

- основная функция (main), работающая с меню;
- ввод исходных данных с клавиатуры;
- последовательный алгоритм поворота фигуры;
- параллельный алгоритм поворота фигуры;
- замеры процессорного времени.

2.6 Вывод

На основе теоретических данных, полученные в аналитическом разделе были построены схемы исследуеммых алгоритмов. Также были описаны: структуры данных, используемые в алгоритмах, способы тестирования и классы эквивалентности, память, используемая алгоритмом, и структура ПО.

3 Технологическая часть

В этом разделе будут разработаны исходные коды последовательного и параллельного алгоритмов поворота фигуры.

3.1 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран многопоточный язык C++. Данный выбор обусловлен моим желанием расширить свои знания в области применения данного языка.

3.2 Реализация алгоритмов

В листингах 3.1 - 3.3 приведена реализация последовательного и параллельного алгоритмов поворота фигуры [3].

Листинг 3.1: Последовательный алгоритм поворота

```
1  int rot(figure_t &fig, double fi, int fl)
2  {
3    double angle_pi = degrees_to_rad(fi);
4    for (int i = 0; i < fig.n; ++i)
5      point_rotate(fig.pts[i], angle_pi);
6    return SUCCESS;
7  }
```

Листинг 3.2: Параллельный алгоритм поворота

```
void rot_part(figure_t& fig, double fi, int start_pt, int end_pt)
{
  for (int i = start_pt; i < end_pt; i++)
    point_rotate(fig.pts[i], fi);
}

int rot_paral(figure_t& fig, double fi, int t_count)
{
  auto* threads = new std::thread[t_count];

  double angle_pi = degrees_to_rad(fi);

  int pts_per_t = fig.n / t_count;
  int start_pt = 0, end_pt;
  for (int i = 0; i < t_count - 1; i++)
  {
    start_pt = i * pts_per_t;
    end_pt = (i + 1) * pts_per_t;
    threads[i] = std::thread(rot_part, std::ref(fig), angle_pi, start_pt,
       end_pt);
  }
  threads[t_count - 1] = std::thread(rot_part, std::ref(fig), angle_pi, (
     t_count - 1) * pts_per_t, fig.n);

  for (int i = 0; i < t_count; i++)
    threads[i].join();

  return SUCCESS;
}
```

Листинг 3.3: Функция поворота точки

```
void point_rotate(point_t& point, double angle)
{
  double tmp_x = point.x;
  point.x = point.x * cos(angle) - point.y * sin(angle);
  point.y = tmp_x * sin(angle) + point.y * cos(angle);
}
```

В листинге 3.4 представлена реализация замера процессорного времени [4].

Листинг 3.4: Функция замера процессорного времени

```
#include <Windows.h>

double getCPUTime()
{
  FILETIME createTime;
  FILETIME exitTime;
  FILETIME kernelTime;
  FILETIME userTime;
  if (GetProcessTimes(GetCurrentProcess(),
  &createTime, &exitTime, &kernelTime, &userTime) != -1)
  {
    SYSTEMTIME userSystemTime;
    if (FileTimeToSystemTime(&userTime, &userSystemTime) != -1)
    return (double)userSystemTime.wHour * 3600.0 +
    (double)userSystemTime.wMinute * 60.0 +
    (double)userSystemTime.wSecond +
    (double)userSystemTime.wMilliseconds / 1000.0;
  }
}
```

3.3 Тестовые данные

В таблицах 3.1 - 3.2 приведены тестовые данные, на которых было протестированно разработанное ПО. Как видно из этой таблицы, все тесты были успешно пройдены, что означает, что программа работает правильно.

Таблица 3.1: Ожидаемый результат тестов

Точки	**Угол поворота**	**Ожидаемый результат**
[[1, 2], [2, 3], [3, 4], [4, 5]]	90	[[-2, 1], [-3, 2], [-4, 3], [-5, 4]]
[[1, 2], [2, 3], [3, 4], [4, 5]]	-90	[[2, -1], [3, -2], [4, -3], [5, -4]]
[[1, 2], [2, 3], [3, 4], [4, 5]]	360	[[1, 2], [2, 3], [3, 4], [4, 5]]
[[1, 2], [2, 3], [3, 4], [4, 5]]	0	[[1, 2], [2, 3], [3, 4], [4, 5]]

Таблица 3.2: Полученный результат тестов

Точки	**Угол поворота**	**Полученный результат**
[[1, 2], [2, 3], [3, 4], [4, 5]]	90	[[-2, 1], [-3, 2], [-4, 3], [-5, 4]]
[[1, 2], [2, 3], [3, 4], [4, 5]]	-90	[[2, -1], [3, -2], [4, -3], [5, -4]]
[[1, 2], [2, 3], [3, 4], [4, 5]]	360	[[1, 2], [2, 3], [3, 4], [4, 5]]
[[1, 2], [2, 3], [3, 4], [4, 5]]	0	[[1, 2], [2, 3], [3, 4], [4, 5]]

3.4 Вывод

В данном разделе были разработаны исходные коды последовательного и параллельного алгоритмов поворота фигуры.

4 Исследовательская часть

В этом разделе будет проведён эксперимент для определения эффективности по времени работы каждого из разработанных алгоритмов.

4.1 Пример работы

Демонстрация работы программы приведена на рисунке 4.1.

```
Menu:
1 - rotate
2 - measure time
0 - exit
Choice: 1
Input number of points: 4

Input points (in format: x y):
1 2
2 3
3 4
4 5

Input angle of rotation (in degrees): 90

Result of rotation with linear rotation algorithm :
-2 1
-3 2
-4 3
-5 4

Result of rotation with parallel rotation algorithm :
-2 1
-3 2
-4 3
-5 4
```

Рис. 4.1: Работа алгоритмов поворота фигуры.

4.2 Технические характеристики

Ниже приведеные технические характеристики устройства, на котором было проведенно тестирование ПО:

- Операционная система: Windows 10 64-bit Home [5].
- Оперативная память: 8 GB.
- Процессор: 11th Gen Intel(R) Core(TM) i3-1115G4 @ 3.00GHz [6].

4.3 Время выполнения алгоритмов

В таблицах 4.1 - 4.2 представлены замеры времени работы для каждого из алгоритмов.

Таблица 4.1: Время работы при различном кол-ве потоков (кол-во точек = 8000)

Кол-во потоков	Время (мс)
1	1094
2	594
4	328
8	279
16	218
32	207

Таблица 4.2: Время работы при разном количестве точек

Кол-во точек	1 поток	4 потока
1000	109	107
2000	203	168
3000	344	209
4000	438	228
5000	546	265
6000	641	327
7000	766	406
8000	891	484

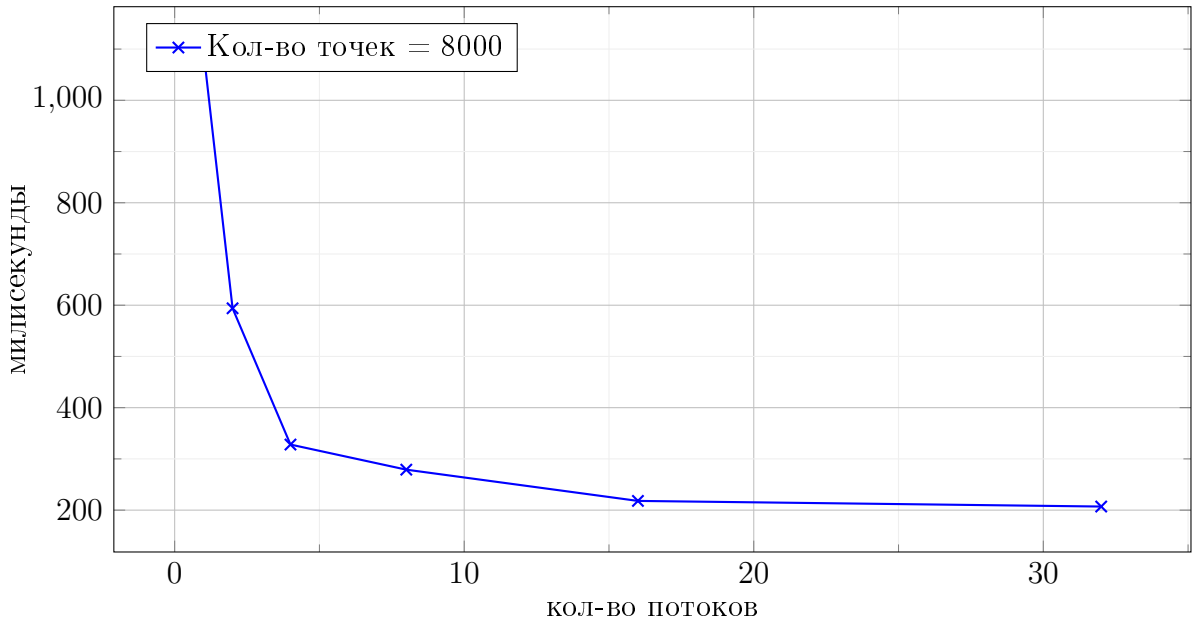

Рис. 4.2: Параллельный алгоритм при различном кол-ве потоков

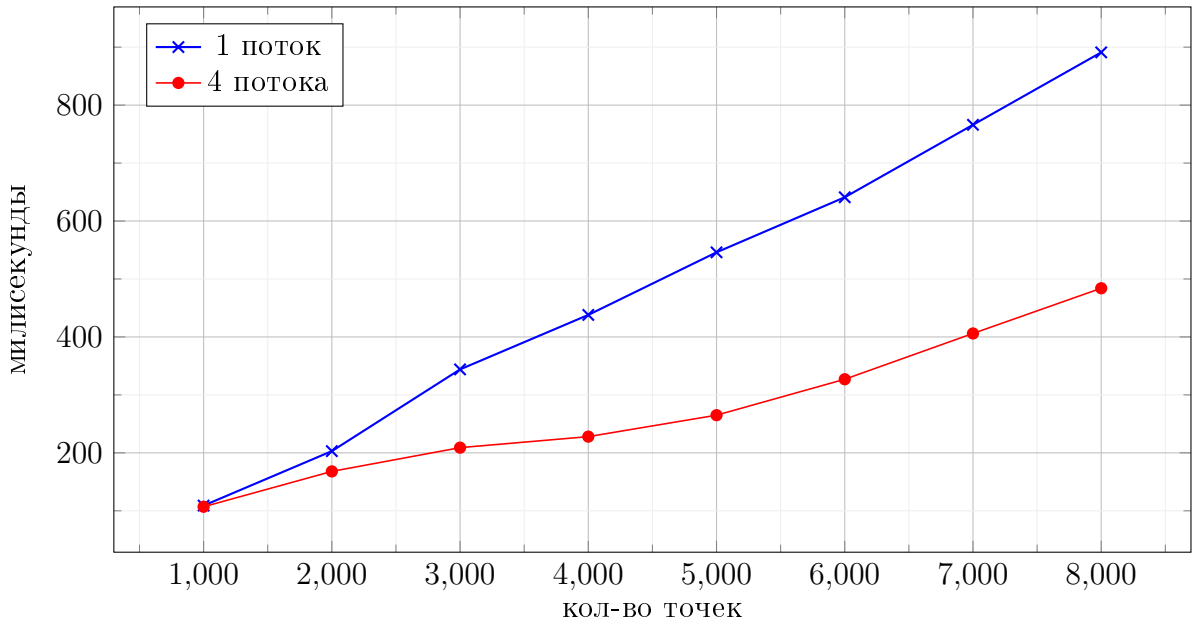

Рис. 4.3: Сравнение времени работы алгоритмов при разном кол-ве точек

4.4 Вывод

В результате проведенного эксперимента можно сделать вывод, что использование многопоточности однозначно может дать существенный выигрыш по времени работы алгоритма. Так, алгоритм с 4 потоками на 8000 точек эффективнее однопоточного примерно на 45,7%.

При этом на небольшом количестве точек использование многопоточности неэффективно, т. к. в таком случае накладные расходы на создание потоков превышают выигрыш от

использования распараллеливания. Например, на 1000 точек, как видно из эксперимента, однопоточный алгоритм практически не отличается по времени от многопоточного.

Заключение

В ходе проделанной работы были изучены и реализованы однопоточный и многопоточный алгоритмы поворота фигуры.

Экспериментально было подтверждено различие по временной эффективности этих алгоритмов на материале замеров процессорного времени выполнения реализации на варьирующемся количестве точек. Так, многопоточный алгоритм на существенных размерах данных работает значительно быстрее однопоточного (например, алгоритм с 4 потоками на 8000 точек эффективнее по времени однопоточного примерно на 45,7%).

При этом на малом количестве точек эта разница незаметна или даже, наоборот, многопоточный алгоритм начинает терять свою эффективность, так как накладные расходы на создание потоков в таком случае превышают выигрыш распараллеливания.

Литература

[1] A. Williams. C++ Concurrency in Action: Practical Multithreading. Manning Publications Co. 2019.

[2] Лурье А. И. Аналитическая механика. — М.:Физматлит. 1961. с. 824.

[3] Класс thread. Режим доступа: https://docs.microsoft.com/ru-ru/cpp/standard-library/thread-class?view=msvc-160. Дата обращения: 26.10.2021.

[4] GetProcessTimes function (processthreadsapi.h). Режим доступа: https://docs.microsoft.com/ru-ru/windows/win32/api/processthreadsapi/nf-processthreadsapi-getprocesstimes?redirectedfrom=MSDN. Дата обращения: 27.10.2021.

[5] Windows 10 Pro и Windows 10 Домашняя. Режим доступа: https://www.microsoft.com/ru-ru/windows/compare-windows-10-home-vs-pro. Дата обращения: 18.10.2021.

[6] Процессор Intel® Core™ i3-1115G4 (6 МБ кэш-памяти, до 4,10 ГГц). Режим доступа: https://www.intel.ru/content/www/ru/ru/products/sku/208652/intel-core-i31115g4-processor-6m-cache-up-to-4-10-ghz/specifications.html. Дата обращения: 14.10.2021.