

## 3.1.2. Множество Мандельброта

Алгоритм выполняет вычисление множества Мандельброта с использованием многопоточности для повышения производительности. Он разделен на несколько этапов:

### 1. Инициализация данных:

- Программа принимает два аргумента: количество потоков ( `nthreads` ) и количество точек ( `npoints` ) по каждой оси комплексной плоскости.
- Также задаются границы области комплексной плоскости, в пределах которой будет вычисляться множество Мандельброта: от `min_real` до `max_real` по оси действительных чисел и от `min_imag` до `max_imag` по оси мнимых чисел.

### 2. Подготовка к многопоточному выполнению:

- Массив `result` используется для хранения результатов проверки принадлежности каждой точки множеству Мандельброта. Для каждой точки хранится логическое значение (принадлежит или нет).
- Программа делит все точки на участки, которые будут обрабатываться параллельно в каждом потоке. Каждому потоку назначается определенный диапазон точек для обработки.
- Каждый поток получает объект структуры `ThreadData`, содержащий параметры для вычислений: границы диапазона точек, параметры плоскости, максимальное количество итераций для проверки принадлежности множеству и указатель на общий массив результатов.

### 3. Вычисления в потоках:

- Каждый поток вычисляет, принадлежат ли назначенные ему точки множеству Мандельброта. Для каждой точки координаты преобразуются в соответствующие действительное и мнимое число.

- Функция `is_in_mandelbrot` проводит итеративное вычисление для каждой точки на основе рекуррентного соотношения.
- Если после определенного количества итераций (максимум — 1000) модуль комплексного числа не превышает 2, точка считается принадлежащей множеству.

#### 4. Запись результатов:

- После завершения вычислений все потоки завершаются, и программа объединяет результаты.
- Результаты (координаты точек, принадлежащих множеству) записываются в CSV-файл.
- Также время выполнения алгоритма и ключевые параметры (количество потоков, количество точек) записываются в текстовый файл с форматированием: каждая запись включает три значения (количество потоков, количество точек и время выполнения), записанных через запятую.

#### 5. Измерение времени:

- Время выполнения программы измеряется с использованием встроенного таймера ( `GET_TIME` ), чтобы оценить эффективность параллельного вычисления и производительность системы при изменении числа потоков и количества точек.

## Оценка времени и эффективности работы

После неоднократного запуска программы с различными значениями получились следующие результаты:

#### Кол-во потоков, кол-во точек, время выполнения программы

```
10,100,0.019438
100,100,0.043376
1000,100,0.234723
10000,100,2.295463
10,500,0.275231
10,1000,0.978109
10,2000,3.560509
```

10,3000,8.997245  
10,5000,33.814126

```
import matplotlib.pyplot as plt

# Данные для графика 1: Влияние количества потоков на время выполнения
num_threads = [10, 100, 1000, 10000]
execution_times_threads = [0.019438, 0.043376, 0.234723, 2.295401]

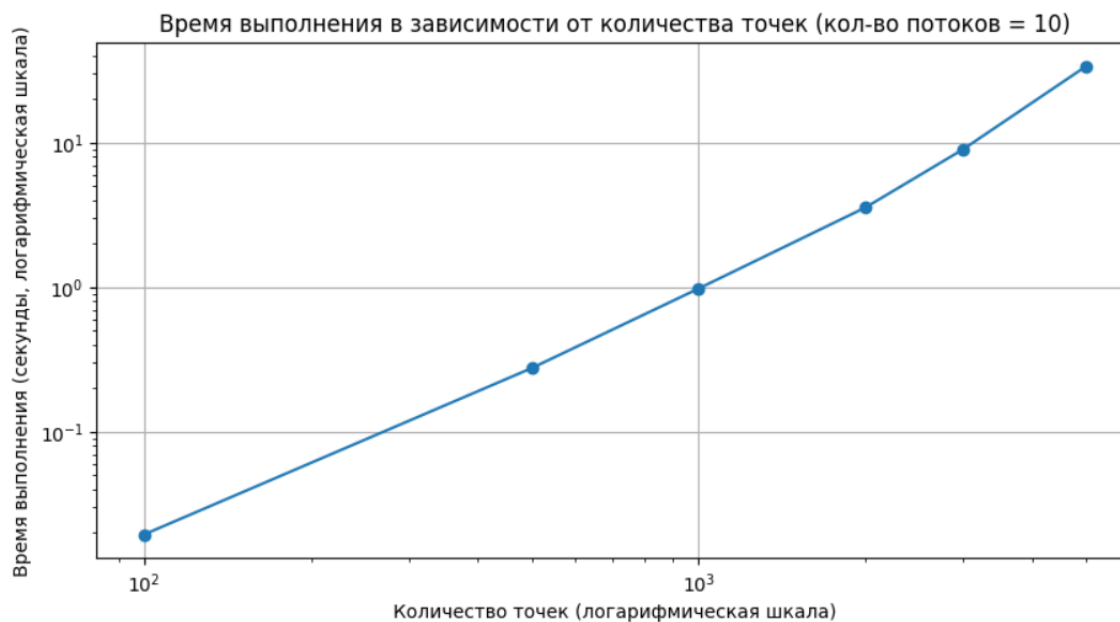
plt.figure(figsize=(10, 5))
plt.plot(num_threads, execution_times_threads, marker='o')
plt.xscale('log')
plt.yscale('log')
plt.title('Время выполнения в зависимости от количества потоков')
plt.xlabel('Количество потоков (логарифмическая шкала)')
plt.ylabel('Время выполнения (секунды, логарифмическая шкала)')
plt.grid(True)
plt.show()

# Данные для графика 2: Влияние количества точек на время выполнения
num_points = [100, 500, 1000, 2000, 3000, 5000]
execution_times_points = [0.019438, 0.275231, 0.978109, 3.560501, 10.019438, 33.814126]

plt.figure(figsize=(10, 5))
plt.plot(num_points, execution_times_points, marker='o')
plt.xscale('log')
plt.yscale('log')
plt.title('Время выполнения в зависимости от количества точек (логарифмическая шкала)')
plt.xlabel('Количество точек (логарифмическая шкала)')
plt.ylabel('Время выполнения (секунды, логарифмическая шкала)')
plt.grid(True)
plt.show()
```



Как видно из графика, время выполнения растет с увеличением числа потоков. Это вполне логично и связано с накладными расходами на создание потоков, синхронизацией между ними и эффективностью использования ресурсов.



Из графика видно, что при увеличении количества точек время выполнения также растет, так как каждый поток обрабатывает больше данных, а вычисления становятся более ресурсоемкими.

## **Выводы**

При увеличении количества потоков от 10 до 10,000 время выполнения программы возрастает не линейно. Наименьшее время выполнения наблюдается при 10 потоках, после чего время начинает расти с увеличением числа потоков. Это связано с накладными расходами на управление потоками и синхронизацию, что приводит к неэффективному использованию ресурсов процессора при высоком числе потоков.

При фиксированном количестве потоков (10) время выполнения значительно возрастает с увеличением количества точек. Эта зависимость также нелинейна: время выполнения увеличивается значительно быстрее при больших значениях точек. Это указывает на то, что алгоритм становится более затратным по времени, когда требуется обработать большее количество точек для вычисления множества Мандельброта.