**Отчет**

**По лабораторной работе №2**

Выполнил  
Студент 4 курса

Группы И-1-21  
Селиванов Глеб

Симферополь – 2024.

**Задание:**

**Цель работы**: реализовать параллельные алгоритмы, учитывая особенности и правила многопоточности.

1. Разработать программу, реализующую параллельный алгоритм.

2. Выполнить анализ эффективности параллельной программы:

* построить графики зависимости ускорения и коэффициента эффективности от числа потоков,
* оценить ускорение по закону Амдала (для нерекурсивных алгоритмов),

оценить масштабирование

**Вариант** 2. Алгоритм быстрой сортировки.

*Входными* данными для программы является неотсортированный числовой массив **a**, заполненный случайными значениями.

*Результат* выполнения программы – отсортированный массив **a**. Необходимо реализовать рекурсивный алгоритм быстрой сортировки. При каждом вызове функции быстрой сортировки необходимо порождать поток. Порождение потоков необходимо остановить при достижении количества потоков, равного заданному числу, которое варьируется от 1 до 10 числа процессорных ядер на вычислительном узле. В качестве опорного элемента выбирать первый элемент последовательности.

**Решение:**

import random

import time

import threading

def quick\_sort(arr):

    if len(arr) <= 1:

        return arr

    pivot = arr[0]

    less = [x for x in arr[1:] if x <= pivot]

    greater = [x for x in arr[1:] if x > pivot]

    return quick\_sort(less) + [pivot] + quick\_sort(greater)

def parallel\_quick\_sort(arr, max\_threads):

    if len(arr) <= 1:

        return arr

    # Создаем потоки до максимального количества

    if max\_threads > 1:

        pivot = arr[0]

        less = [x for x in arr[1:] if x <= pivot]

        greater = [x for x in arr[1:] if x > pivot]

        # Разделяем задачу на два потока

        thread1 = threading.Thread(target=parallel\_quick\_sort, args=(less, max\_threads // 2))

        thread2 = threading.Thread(target=parallel\_quick\_sort, args=(greater, max\_threads // 2))

        thread1.start()

        thread2.start()

        thread1.join()

        thread2.join()

        return parallel\_quick\_sort(less, max\_threads // 2) + [pivot] + parallel\_quick\_sort(greater, max\_threads // 2)

    else:

        return quick\_sort(arr)

# Пример использования

if \_\_name\_\_ == "\_\_main\_\_":

    # Генерация случайного массива

    arr = [random.randint(1, 100) for \_ in range(100)]

    print("Исходный массив:", arr)

    # Измерение времени

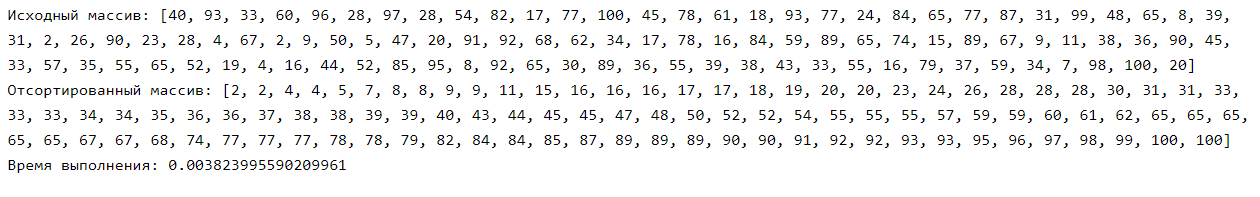
    start\_time = time.time()

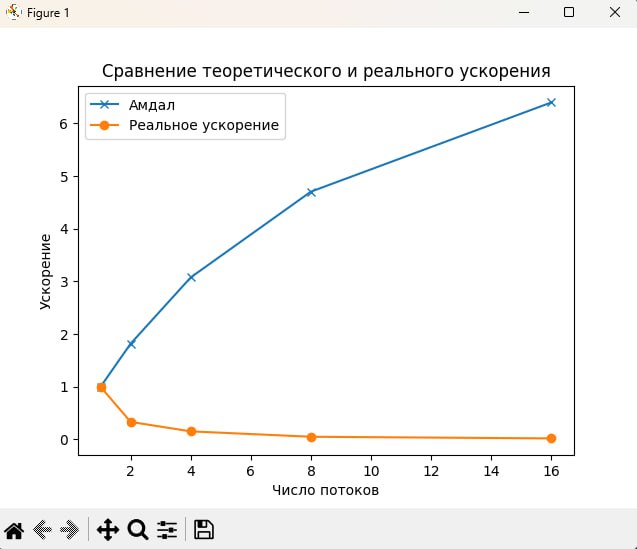
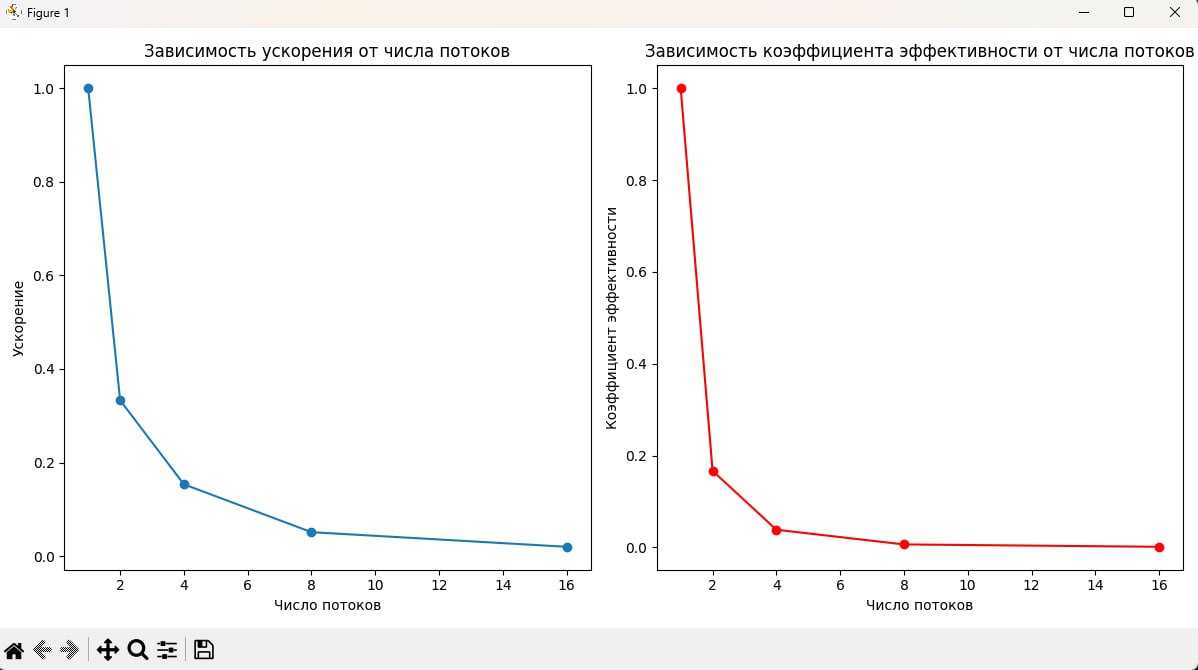
    sorted\_arr = parallel\_quick\_sort(arr, 4)  # Использование 4 потоков

    end\_time = time.time()

    print("Отсортированный массив:", sorted\_arr)

    print("Время выполнения:", end\_time - start\_time)

**Результат:  
**

****