МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
 РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Московский государственный университет геодезии и картографии»  
(МИИГАиК)  
Факультет геоинформатики и информационной безопасности  
Кафедра геоинформационных систем и технологий

**Лабораторная работа №4**  
**«Знакомство с параллелизацией»**

Проверил: Выполнил:

Лебедев Е.Д. Студент группы: 2024-ФГиИБ-ПИ-1б

Николаев М.А.

Москва 2025

**Цель работы**

Выполнить предоставленные задачи на языке **Python.**

**Задание (вариант №19)**

Реализовать быструю сортировку, сравнить обычную сортировку с многозадачной (многопоточной) сортировкой, рассчитать коэффициент ускорения, сделать выводы относительно ускорения

**Анализ задания**

*Входные данные:*

На вход программа получает N – размер массива

*Промежуточные данные:*

-

*Выходные данные:*

Программа выводит время последовательной сортировки и сортировки по потокам.

**Листинг программы**Ссылка на **GitHub**: <https://github.com/Ep1cMax/Second_Semester/tree/main/Works>

fimport random

import time

import threading

def generate\_array(size):

return [random.randint(1, 1000000) for \_ in range(size)]

def partition(arr, low, high):

pivot = arr[high]

i = low - 1

for j in range(low, high):

if arr[j] < pivot:

i += 1

arr[i], arr[j] = arr[j], arr[i]

arr[i + 1], arr[high] = arr[high], arr[i + 1]

return i + 1

def quicksort(arr, low, high):

if low < high:

pi = partition(arr, low, high)

quicksort(arr, low, pi - 1)

quicksort(arr, pi + 1, high)

def threaded\_quicksort(arr, low, high, max\_depth=0, depth=0):

if low < high:

pi = partition(arr, low, high)

if depth < max\_depth:

left\_thread = threading.Thread(

target=threaded\_quicksort,

args=(arr, low, pi - 1, max\_depth, depth + 1)

)

left\_thread.start()

threaded\_quicksort(arr, pi + 1, high, max\_depth, depth + 1)

left\_thread.join()

else:

quicksort(arr, low, pi - 1)

quicksort(arr, pi + 1, high)

def measure\_execution\_time(func):

start = time.time()

func()

end = time.time()

return (end - start) \* 1000

def main(N):

#N = 1000000 #Объявление N для вызова основной программы не функцией

data = generate\_array(N)

data\_copy = data.copy()

time\_seq = measure\_execution\_time(

lambda: quicksort(data\_copy, 0, len(data\_copy) - 1)

)

print(f"Sequential QuickSort: {time\_seq:.2f} ms")

for threads in [2, 4, 8]:

data\_copy = data.copy()

max\_depth = threads.bit\_length() - 1

time\_threaded = measure\_execution\_time(

lambda: threaded\_quicksort(data\_copy, 0, len(data\_copy) - 1, max\_depth)

)

print(f"Threaded QuickSort ({threads} threads): {time\_threaded:.2f} ms")

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

main(1000000)

**Результаты работы программы:**

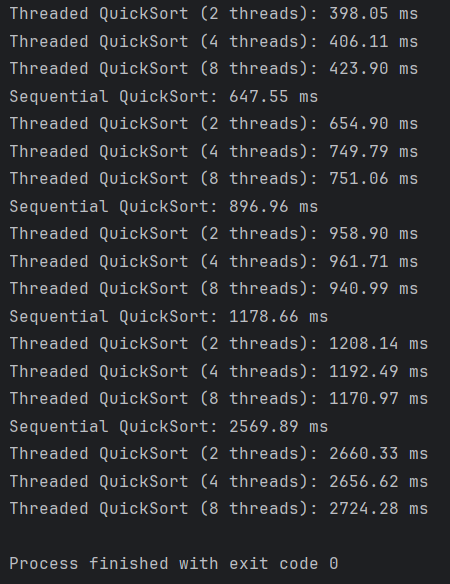


Рис. 1. – Работа программы при разной размерности массива.

**Блок-схема**

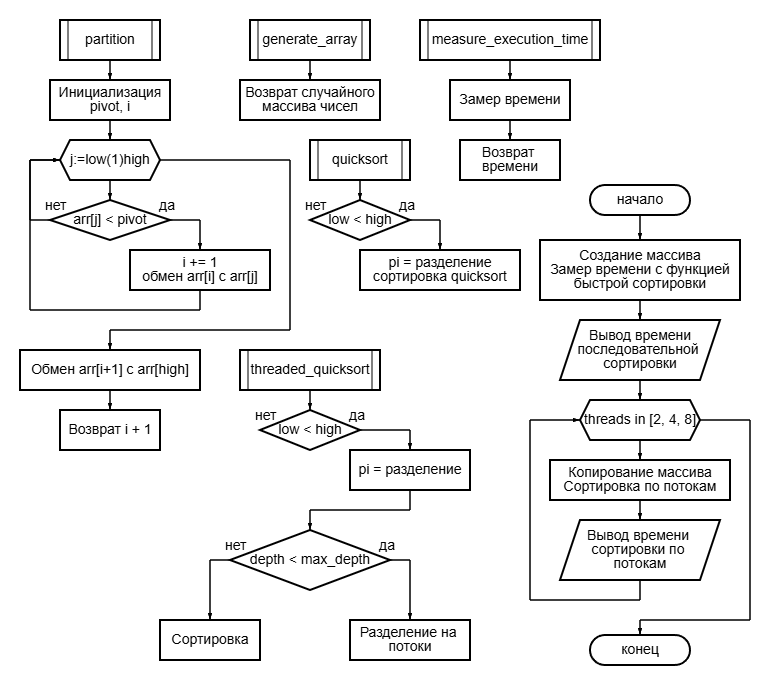


Рис. 2. Блок-схемы.

Быстрая сортировка соответствует предыдущим замерам, до большего ускорения эффективности при больших размерах массива

Граф 1. – Зависимость времени сортировки от размерности (в мс)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размерность массива | Послед. сортировка | 2 потока | 4 потока | 8 потоков |
| 10 | 0,00 | 1,03 | 1,03 | 1,29 |
| 100 | 0,00 | 0,00 | 1,06 | 1,04 |
| 1000 | 1,01 | 1,00 | 3,01 | 2,00 |
| 10000 | 13,01 | 14,56 | 16,03 | 18,32 |
| 100000 | 181,27 | 177,93 | 189,16 | 185,15 |
| 200000 | 384,13 | 398,05 | 406,11 | 423,90 |
| 300000 | 647,55 | 654,90 | 749,79 | 751,06 |
| 400000 | 896,96 | 958,90 | 961,71 | 940,99 |
| 500000 | 1178,66 | 1208,14 | 1192,49 | 1170,97 |
| 1000000 | 2569,89 | 2660,33 | 2656,62 | 2724,28 |

Табл. 1. – Время сортировки разных массивов от кол-ва потоков (мс)

Коэффициент ускорения рассчитывается по формуле: время сортировки с 1 потоком / время сорт. с n потоками.

|  |  |  |  |
| --- | --- | --- | --- |
| Размерность массива | Ускорение при 2 потоках | Ускорение при 4 потоках | Ускорение при 8 потоках |
| 10 | 0,00 | 0,00 | 0,00 |
| 100 | 0,00 | 0,00 | 0,00 |
| 1000 | 1,01 | 0,34 | 0,51 |
| 10000 | 0,89 | 0,81 | 0,71 |
| 100000 | 1,02 | 0,96 | 0,98 |
| 200000 | 0,97 | 0,95 | 0,91 |
| 300000 | 0,99 | 0,86 | 0,86 |
| 400000 | 0,94 | 0,93 | 0,95 |
| 500000 | 0,98 | 0,99 | 1,01 |
| 1000000 | 0,97 | 0,97 | 0,94 |

Табл. 2. – Ускорение при разных размерах массива

**Вывод**

Ускорение с использованием потоков начинает быть заметным лишь при больших размерах массивов, а при маленьких потоки вызывают лишнюю нагрузку памяти, почти не дающую ускорения.