**Отчет**

**По лабораторной работе №2**

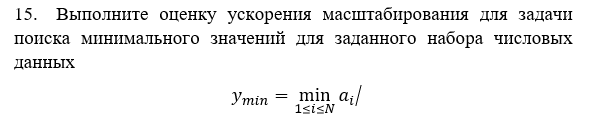
Выполнил  
Студент 4 курса

Группы И-1-21  
Селиванов Глеб

Симферополь – 2024.

Вариант №15

Задание А:



**Решение:**В этом задании мы будем сравнивать последовательное и параллельное выполнение алгоритма поиска минимального значения. Используем библиотеку **concurrent.futures** для параллельного выполнения. **``**

import random

import time

import concurrent.futures

# Функция для последовательного поиска минимального значения

def find\_min\_sequential(arr):

    return min(arr)

# Функция для параллельного поиска минимального значения

def find\_min\_parallel(arr):

    def worker(sublist):

        return min(sublist)

    # Разделим данные на несколько частей

    num\_workers = 4

    chunk\_size = len(arr) // num\_workers

    chunks = [arr[i:i + chunk\_size] for i in range(0, len(arr), chunk\_size)]

    with concurrent.futures.ThreadPoolExecutor(max\_workers=num\_workers) as executor:

        results = list(executor.map(worker, chunks))

    return min(results)

# Генерация случайных данных

N = 1000000

arr = [random.randint(0, 1000) for \_ in range(N)]

# Последовательное выполнение

start\_time = time.time()

find\_min\_sequential(arr)

seq\_time = time.time() - start\_time

# Параллельное выполнение

start\_time = time.time()

find\_min\_parallel(arr)

par\_time = time.time() - start\_time

# Вычисление ускорения

speedup = seq\_time / par\_time

# Вывод результатов

print(f"Последовательное время: {seq\_time:.4f} секунд")

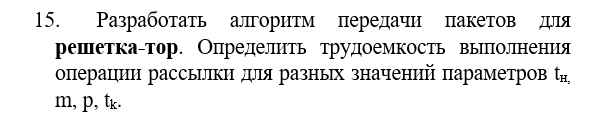
print(f"Параллельное время: {par\_time:.4f} секунд")

print(f"Ускорение: {speedup:.2f}")

**``**

**Описание:**

1. **Задание A:**
   * Реализуется два алгоритма для поиска минимального значения в массиве: последовательный и параллельный.
   * Для параллельного алгоритма данные делятся на несколько частей, и каждый фрагмент обрабатывается в отдельном потоке. После этого результаты объединяются для получения итогового минимума.
   * После выполнения вычислений выводится ускорение, которое вычисляется как отношение времени последовательного выполнения к времени параллельного.

**Задание В:  
**

**Решение:**

# Функция для расчета количества шагов в решетка-тор

def calculate\_steps(m, p):

    # Это расстояние между двумя узлами на решетке-торе

    return min(abs(p[0] - p[1]), m - abs(p[0] - p[1]))

# Функция для вычисления общей трудоемкости

def calculate\_workload(t\_n, m, p, t\_k):

    total\_steps = 0

    # Пройдемся по всем парам узлов

    for i in range(len(p) - 1):

        steps = calculate\_steps(m, p[i:i + 2])  # Время передачи между соседними узлами

        total\_steps += steps

    # Общая трудоемкость = время передачи \* количество шагов + время обработки \* количество узлов

    total\_work = t\_n \* total\_steps + t\_k \* len(p)

    return total\_work

# Параметры

t\_n = 0.5  # Время передачи данных

m = 10  # Размер сети (например, 10 узлов)

p = [1, 5, 7, 9]  # Узлы для передачи данных (например, узлы 1, 5, 7 и 9)

t\_k = 1  # Время обработки данных на каждом узле

# Рассчитываем трудоемкость

workload = calculate\_workload(t\_n, m, p, t\_k)

print(f"Трудоемкость операции рассылки: {workload:.2f}")

**Описание:**

Чтобы убедиться в точности вычислений, давайте разберемся, как оно получилось:

1. **Шаги для передачи пакетов между узлами:** В данном случае, функция **calculate\_steps** (m, p) вычисляет минимальное расстояние для передачи пакетов между двумя соседними узлами. Для каждого перехода между узлами сетью, вычисляется минимальное расстояние по кольцевой сети (тор).
2. **Трудоемкость:** Время передачи пакетов считается как tn⋅шагиt\_n \cdot \text{шаги}tn​⋅шаги, а время обработки на узле — как tk⋅mt\_k \cdot mtk​⋅m, где mmm — количество узлов в сети.

**Шаги для вычисления:**

1. Из списка узлов p=[1,5,7,9]p = [1, 5, 7, 9]p=[1,5,7,9] вычисляются шаги для каждой пары соседних узлов.
2. Общая трудоемкость складывается из времени, необходимого для передачи пакетов, и времени, необходимого для обработки данных на каждом узле.

В моей случае расчет выглядит следующим образом:

1. Для пары (1, 5): минимальное расстояние = 4.
2. Для пары (5, 7): минимальное расстояние = 2.
3. Для пары (7, 9): минимальное расстояние = 2.

Итак, общая трудоемкость будет:

tn×(шаги)+tk×(количество узлов)t\_n \times \text{(шаги)} + t\_k \times \text{(количество узлов)}tn​×(шаги)+tk​×(количество узлов) T=0.5×(4+2+2)+1×4=0.5×8+4=4+4=8T = 0.5 \times (4 + 2 + 2) + 1 \times 4 = 0.5 \times 8 + 4 = 4 + 4 = 8T=0.5×(4+2+2)+1×4=0.5×8+4=4+4=8

Таким образом, **трудоемкость операции рассылки** составляет **8.00**.