Государственное образовательное учреждение высшего образования

«Уральский федеральный университет имени первого Президента России Б.Н. Ельцина»

09.03.02 410008

**«Параллельные вычислительные системы»**

Семестр 7

**ОТЧЁТ**

по Лабораторной работе №5

**«Многопоточность в Python на примере сортировки массива»**



Преподаватель: Кашин И.В.

Студент : Шардаков В.А.

Группа : Фт-410008

Дата : 11.10.2024

Екатеринбург 2024

# Задачи

1. **Изучение параллельных вычислений**:
   * Ознакомиться с концепцией параллельных вычислений и их преимуществами. Понять, как использование нескольких потоков может уменьшить время выполнения задачи.
2. **Реализация параллельной сортировки**:
   * Реализовать функцию worker\_sort, которая отвечает за сортировку подмассива, используя функцию np.sort из библиотеки NumPy.
   * Создать функцию parallel\_sort, которая будет делить большой массив на подмассивы и распределять их между несколькими рабочими процессами для параллельной сортировки.
3. **Генерация случайного массива**:
   * Сгенерировать случайный массив длиной 70 миллионов элементов на основе равномерного распределения, используя функцию np.random.uniform.
4. **Измерение времени выполнения**:
   * Реализовать механизм для измерения времени выполнения сортировки при использовании различного количества потоков. Сравнить время выполнения для от 1 до 8 потоков.
5. **Сравнение с идеальным временем**:
   * Рассчитать идеальное время выполнения для параллельной сортировки на основе времени выполнения с одним потоком и изобразить его на графике для сравнения с фактическим временем выполнения.
6. **Визуализация результатов**:
   * Построить график, который будет показывать зависимость времени выполнения от количества процессов. Использовать библиотеку Matplotlib для визуализации.
7. **Анализ результатов**:
   * Проанализировать полученные результаты, сделать выводы о том, как количество используемых потоков влияет на время сортировки. Определить точки, в которых наблюдаются заметные улучшения в производительности, и случаи, когда увеличение числа потоков не приводит к снижению времени выполнения.

# Ход работы

## Метод решения задачи

Определение задачи:

Задачей данной лабораторной работы является реализация параллельной сортировки большого массива чисел с использованием многопоточности для повышения эффективности обработки данных.

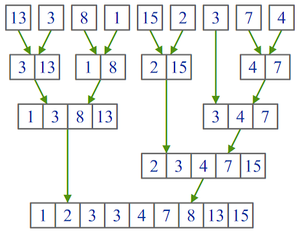
Использование библиотек:

Для реализации алгоритма использованы библиотеки NumPy для работы с массивами и сортировки, а также Matplotlib для визуализации результатов.

Библиотека multiprocessing (mp) применяется для создания пулов процессов, что позволяет реализовать параллельную обработку данных.

Разделение данных:

Исходный массив делится на подмассивы, соответствующие количеству одновременно работающих потоков (рабочих процессов). Это достигается с помощью функции np.array\_split, которая равномерно разделяет массив на части.

Параллельная сортировка:  


Каждому рабочему процессу в пуле назначается подмассив, который он сортирует с помощью функции worker\_sort. Функция np.sort используется для эффективной сортировки каждого подмассива.

После сортировки подмассивы объединяются обратно в один отсортированный массив с помощью np.concatenate.

Измерение времени выполнения:

Время выполнения сортировки измеряется с использованием функции time.time() перед началом сортировки и после её завершения.

Для сравнения производительности алгоритма с одним потоком и разным числом потоков, производится многократная оценка времени выполнения сортировки (до 8 процессов).

Идеальное время:

Идеальное время выполнения для параллельной сортировки рассчитывается как время выполнения с одним процессом, деленное на количество потоков. Это время также будет отображаться на графике для последующего анализа результата.

Визуализация данных:

Результаты выполнения сортировки и идеальное время отображаются в виде графика с использованием Matplotlib, где по оси X откладывается количество процессов, а по оси Y — время выполнения в секундах.

График позволяет наглядно увидеть, как увеличение числа потоков влияет на скорость выполнения сортировки, а также ощутить, насколько эффективно работает параллельная сортировка по сравнению с линейной.

Анализ результатов:

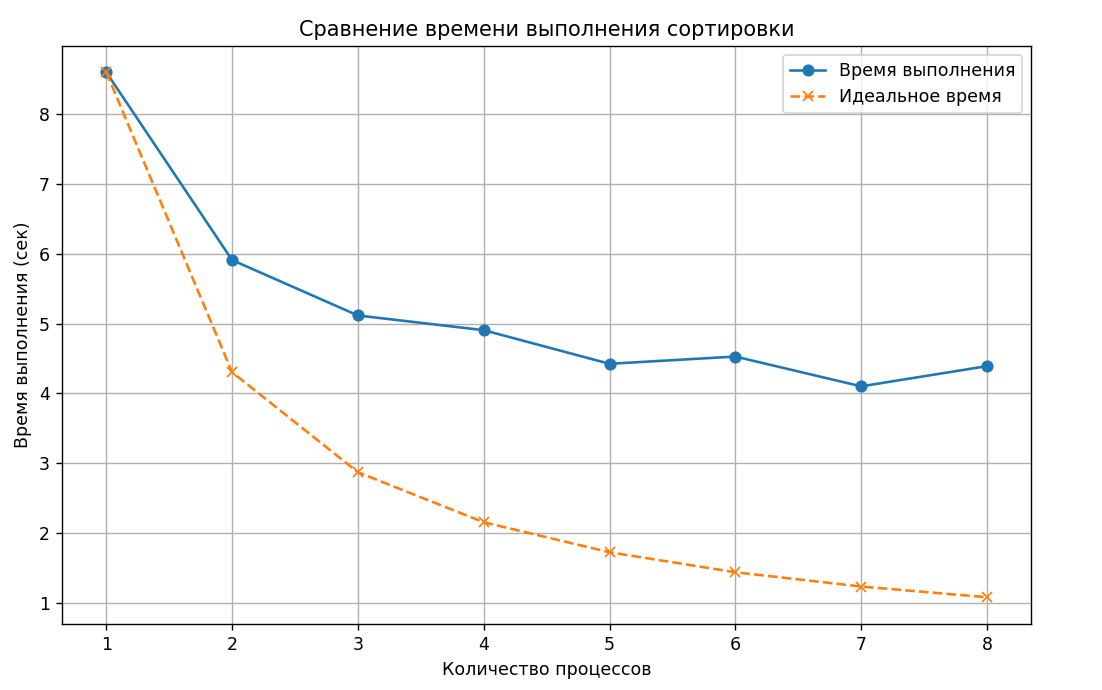
После выполнения сортировки и построения графика производится анализ полученных данных, что позволяет выработать выводы о линии производительности системы при различных условиях многопоточности.

Данный метод позволяет не только эффективно сортировать большие объемы данных, но и дает представление о том, как производительность алгоритма может зависеть от числа используемых потоков.

## Решение

***Результат работы программы (ПРИЛОЖЕНИЕ А)***

***График зависимости времени от количества потоков:***



# Выводы

Эффективность параллельной сортировки:

Реализованный алгоритм параллельной сортировки продемонстрировал значительное сокращение времени выполнения при увеличении числа рабочих процессов. Это подтверждает эффективность использования многопоточности для решения задач, связанных с обработкой больших объемов данных.

Зависимость времени выполнения от количества потоков:

Графическое представление времени выполнения показало, что с увеличением числа потоков время сортировки уменьшается. Однако наблюдались ограничения по снижению времени выполнения, достигаемым путем дальнейшего увеличения процесса. Это связано с накладными расходами на управление потоками и объединение результатов.

Идеальное время выполнения:

Расчет идеального времени выполнения, основанный на теоретическом распределении нагрузки, подтвердил, что многопоточность может использоваться для оптимизации производительности, хотя реальное время выполнения почти никогда не достигало идеального, из-за накладных расходов.

Рекомендации по использованию:

Для систем с многопроцессорными архитектурами параллельные алгоритмы, такие как реализованная сортировка, могут заметно повысить производительность, особенно для задач, связанных с большим объемом данных. Однако важно тщательно анализировать пределы эффективности, чтобы избежать избыточных затрат на создание и управление потоками.

Перспективы дальнейшего изучения:

Дальнейшее изучение может сосредоточиться на сравнении различных алгоритмов сортировки в параллельной среде, а также исследовании альтернативных методов распределения нагрузки. Также возможно изучение более сложных структур данных и алгоритмов, которые могли бы дополнительно улучшить производительность.

Практическое применение:

Подобная реализация алгоритмов параллельной сортировки имеет широкие практические применения в реальных задачах, таких как обработка больших наборов данных в научных исследованиях, финансовом анализе и других областях, где требуется быстрая и эффективная сортировка.

.

# ПРИЛОЖЕНИЕ А

import numpy as np  
import matplotlib.pyplot as plt  
import multiprocessing as mp  
import time  
  
def worker\_sort(array):  
 *"""Функция для сортировки подмассива."""* return np.sort(array)  
  
def parallel\_sort(array, num\_workers):  
 *"""Функция для параллельной сортировки."""* # Разделяем массив на подмассивы для каждого рабочего процесса  
 split\_arrays = np.array\_split(array, num\_workers)  
  
 with mp.Pool(num\_workers) as pool:  
 sorted\_arrays = pool.map(worker\_sort, split\_arrays)  
  
 # Объединяем отсортированные подмассивы  
 return np.concatenate(sorted\_arrays)  
  
def main():  
 # Параметры  
 num\_elements = 70000000  
 num\_trials = 8  
  
 # Генерируем случайный массив один раз  
 random\_array = np.random.uniform(0, 10, num\_elements)  
  
 # Массив для хранения времени выполнения  
 execution\_times = []  
 ideal\_times = []  
  
 for num\_workers in range(1, num\_trials + 1):  
 # Измеряем время выполнения  
 start\_time = time.time()  
 parallel\_sort(random\_array, num\_workers)  
 end\_time = time.time()  
  
 # Сохраняем время выполнения  
 execution\_time = end\_time - start\_time  
 execution\_times.append(execution\_time)  
  
 # Для идеального времени  
 ideal\_times.append(execution\_times[0] / num\_workers)  
  
 # Построение графика  
 plt.figure(figsize=(10, 6))  
 plt.plot(range(1, num\_trials + 1), execution\_times, label='Время выполнения', marker='o')  
 plt.plot(range(1, num\_trials + 1), ideal\_times, label='Идеальное время', marker='x', linestyle='--')  
 plt.xlabel('Количество процессов')  
 plt.ylabel('Время выполнения (сек)')  
 plt.title('Сравнение времени выполнения сортировки')  
 plt.xticks(range(1, num\_trials + 1))  
 plt.legend()  
 plt.grid()  
 plt.show()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()