|  |
| --- |
| Изображение выглядит как зарисовка, корона, рисунок, символ  Автоматически созданное описание |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА - Российский технологический университет»**  **РТУ МИРЭА** |

Институт Информационных Технологий

Кафедра Вычислительной Техники (ВТ)

**Отчёт**

по дисциплине

«Архитектура устройств и систем вычислительной техники»

|  |  |
| --- | --- |
|  | |
| Выполнили студенты группы ИВМО-02-24  Принял преподаватель | Кутепов А.О.  Рьянов А.Е.  Смирнов А.В.    Гуличева А.А. |
|  |  |

Работа выполнена «\_\_\_»\_\_\_\_\_\_\_\_20\_\_\_г

«Зачтено»

«\_\_\_»\_\_\_\_\_\_\_\_20\_\_\_г

Москва 2024

Реализация закона Амдала и закона Густафсона-Барсиса на Python

import time  
import matplotlib.pyplot as plt  
from multiprocessing import Pool  
  
  
def factorial\_part(start, end):  
 *"""Вычисляет произведение чисел от start до end (включительно)."""* result = 1  
 for i in range(int(start), int(end) + 1):  
 result \*= i  
 return result  
  
  
def parallel\_factorial(n, num\_processes):  
 *"""Вычисляет факториал числа n с использованием параллельных процессов."""* chunk\_size = n // num\_processes  
 ranges = [(i \* chunk\_size + 1, (i + 1) \* chunk\_size) for i in range(num\_processes)]  
  
 # Обрабатываем последний диапазон  
 ranges[-1] = (ranges[-1][0], n)  
  
 with Pool(processes=num\_processes) as pool:  
 results = pool.starmap(factorial\_part, ranges)  
  
 final\_result = 1  
 for result in results:  
 final\_result \*= result  
  
 return final\_result  
  
  
def measure\_execution\_time(n, num\_processes):  
 *"""Измеряет время выполнения параллельного вычисления факториала."""* start\_time = time.time()  
 fact = parallel\_factorial(n, num\_processes)  
 end\_time = time.time()  
 return end\_time - start\_time  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Параметры  
 n = 100  
 max\_processes = 4 # Максимальное количество процессов  
 sequential\_time = []  
  
 # Измеряем время выполнения для последовательного алгоритма  
 start\_time = time.time()  
 fact\_sequential = factorial\_part(1, n)  
 end\_time = time.time()  
 sequential\_time.append(end\_time - start\_time)  
  
 # Измеряем время выполнения для параллельного алгоритма  
 parallel\_times = []  
 for num\_processes in range(1, max\_processes + 1):  
 exec\_time = measure\_execution\_time(n, num\_processes)  
 parallel\_times.append(exec\_time)  
  
 # График 1: Закон Амдала  
 plt.figure(figsize=(14, 6))  
  
 # График зависимости времени выполнения от количества процессов  
 plt.subplot(1, 2, 1)  
 plt.plot(range(1, max\_processes + 1), parallel\_times, label='Параллельное выполнение', marker='o')  
 plt.axhline(y=sequential\_time[0], color='r', linestyle='--', label='Последовательное выполнение')  
 plt.title('Зависимость времени выполнения от количества процессов (Закон Амдала)')  
 plt.xlabel('Количество процессов')  
 plt.ylabel('Время выполнения (сек)')  
 plt.legend()  
 plt.grid()  
  
 # График 2: Закон Густафсона-Бариса  
 # Для этого графика мы будем использовать те же данные, но с учетом увеличения задачи  
 scaling\_factors = [n \* (1 + (i / 10)) for i in range(max\_processes)] # Увеличиваем задачу  
 gustafson\_times = [measure\_execution\_time(scaling\_factors[i], i + 1) for i in range(max\_processes)]  
  
 plt.subplot(1, 2, 2)  
 plt.plot(range(1, max\_processes + 1), gustafson\_times, label='Густафсон-Барис', marker='o')  
 plt.title('Зависимость времени выполнения от общей вычислительной мощности (Закон Густафсона-Бариса)')  
 plt.xlabel('Количество процессов')  
 plt.ylabel('Время выполнения (сек)')  
 plt.legend()  
 plt.grid()  
  
 plt.tight\_layout()  
 plt.show()

Изображение выглядит как текст, линия, График, диаграмма

Автоматически созданное описание

**Объяснение кода:**

1. **Функция measure\_execution\_time(n, num\_processes)**: измеряет время выполнения параллельного вычисления факториала для заданного числа процессов.
2. **Измерение времени выполнения**: сначала мы вычисляем время выполнения последовательного алгоритма, а затем для параллельного с различным количеством процессов.
3. **Графики**:
   * **График 1 (Закон Амдала)**: показывает зависимость времени выполнения от количества процессов. Красная пунктирная линия показывает время выполнения последовательного алгоритма.
   * **График 2 (Закон Густафсона-Бариса)**: показывает, как время выполнения изменяется с увеличением задачи при параллельном выполнении.

**Оптимизации в коде:**

1. **Использование numpy**: Вместо обычного цикла для вычисления произведения, мы используем **numpy.prod** и **numpy.arange**, что значительно ускоряет процесс.
2. **Упрощение перемножения результатов**: Мы используем **numpy.prod** для перемножения результатов, что также может быть более эффективно, чем использование обычного цикла.

import time  
import numpy as np  
import matplotlib.pyplot as plt  
from multiprocessing import Pool  
  
  
def factorial\_part(start, end):  
 *"""Вычисляет произведение чисел от start до end (включительно) с использованием numpy."""* return np.prod(np.arange(start, end + 1))  
  
  
def parallel\_factorial(n, num\_processes):  
 *"""Вычисляет факториал числа n с использованием параллельных процессов."""* chunk\_size = n // num\_processes  
 ranges = [(i \* chunk\_size + 1, (i + 1) \* chunk\_size) for i in range(num\_processes)]  
  
 # Обрабатываем последний диапазон  
 ranges[-1] = (ranges[-1][0], n)  
  
 with Pool(processes=num\_processes) as pool:  
 results = pool.starmap(factorial\_part, ranges)  
  
 final\_result = np.prod(results) # Используем numpy для перемножения  
 return final\_result  
  
  
def measure\_execution\_time(n, num\_processes):  
 *"""Измеряет время выполнения параллельного вычисления факториала."""* start\_time = time.time()  
 fact = parallel\_factorial(n, num\_processes)  
 end\_time = time.time()  
 return end\_time - start\_time  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Параметры  
 n = 100  
 max\_processes = 4 # Максимальное количество процессов  
 sequential\_time = []  
  
 # Измеряем время выполнения для последовательного алгоритма  
 start\_time = time.time()  
 fact\_sequential = factorial\_part(1, n)  
 end\_time = time.time()  
 sequential\_time.append(end\_time - start\_time)  
  
 # Измеряем время выполнения для параллельного алгоритма  
 parallel\_times = []  
 for num\_processes in range(1, max\_processes + 1):  
 exec\_time = measure\_execution\_time(n, num\_processes)  
 parallel\_times.append(exec\_time)  
  
 # График 1: Закон Амдала  
 plt.figure(figsize=(14, 6))  
  
 # График зависимости времени выполнения от количества процессов  
 plt.subplot(1, 2, 1)  
 plt.plot(range(1, max\_processes + 1), parallel\_times, label='Параллельное выполнение', marker='o')  
 plt.axhline(y=sequential\_time[0], color='r', linestyle='--', label='Последовательное выполнение')  
 plt.title('Зависимость времени вып. от кол-ва процессов (Закон Амдала)')  
 plt.xlabel('Количество процессов')  
 plt.ylabel('Время выполнения (сек)')  
 plt.legend()  
 plt.grid()  
  
 # График 2: Закон Густафсона-Бариса  
 # Для этого графика мы будем использовать те же данные, но с учетом увеличения задачи  
 scaling\_factors = [n \* (1 + (i / 10)) for i in range(max\_processes)] # Увеличиваем задачу  
 gustafson\_times = [measure\_execution\_time(scaling\_factors[i], i + 1) for i in range(max\_processes)]  
  
 plt.subplot(1, 2, 2)  
 plt.plot(range(1, max\_processes + 1), gustafson\_times, label='Густафсон-Барис', marker='o')  
 plt.title('Зависимость времени вып. от общей выч.мощности (Закон Густафсона-Бариса)')  
 plt.xlabel('Количество процессов')  
 plt.ylabel('Время выполнения (сек)')  
 plt.legend()  
 plt.grid()  
  
 plt.tight\_layout()  
 plt.show()

Изображение выглядит как текст, линия, График, диаграмма

Автоматически созданное описание