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 Кутепов А.О.
Рьянов А.Е.
Смирнов А.В.

Принял преподаватель

Гуличева А.А.

Работа выполнена «__»_____20___г

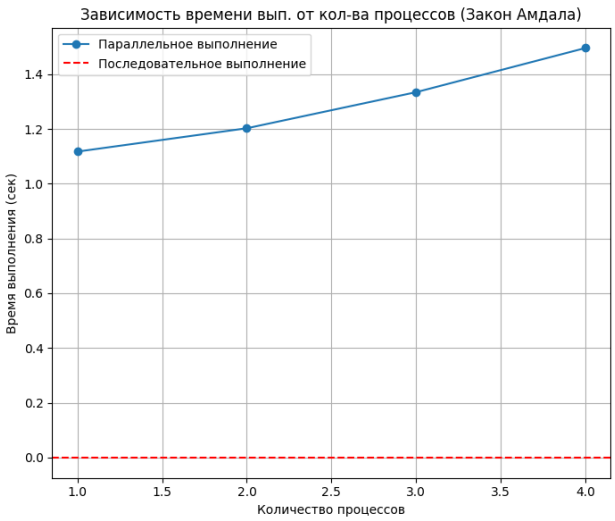
«Зачтено»
«___»_____20___г

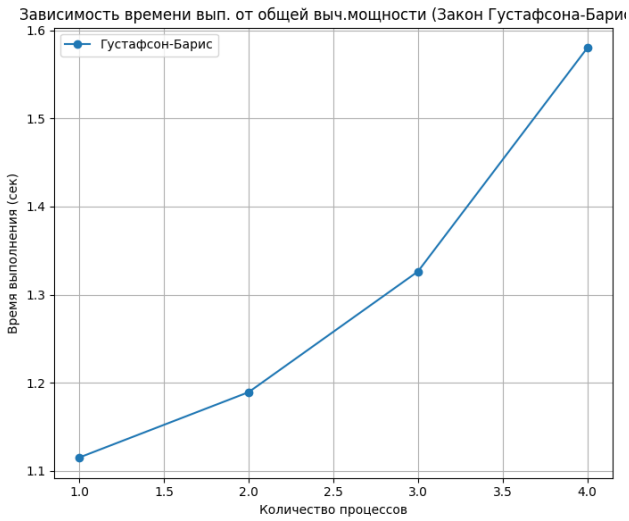
Москва 2024

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()
```





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()
```