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 №3

**«Многопоточность в Python на примере статистического анализа двудольных графов»**



Преподаватель: Кашин И.В.

Студент : Шардаков В.А.

Группа : Фт-410008

Дата : 27.09.2024

Екатеринбург 2024

# Задачи

Целью данной лабораторной работы является исследование методов параллельных вычислений для оптимизации процессов решения задачи о вычислении значений функции (E), основанной на индивидуальных и парных весах агентов.

Основные задачи, которые необходимо решить в ходе выполнения работы:

1. **Генерация данных**: Создание массива индивидуальных весов агентов и матрицы парных весов, используя случайные значения в заданном диапазоне.
2. **Вычисление функции (E)**: Реализация функции, которая будет вычислять значение (E) для различных конфигураций агентов. Значение функции будет зависеть от индивидуальных и парных весов.
3. **Параллельное выполнение вычислений**: Реализация параллельного расчета значений функции (E) для всех возможных конфигураций агентов с использованием модуля multiprocessing. Это позволит значительно ускорить выполнение программы по сравнению с последовательными вычислениями.
4. **Измерение времени выполнения**: Функция для измерения времени, затраченного на вычисление значений функции (E) при использовании различного количества процессов (от 1 до 8). Сравнение реального времени выполнения с идеальными значениями.
5. **Анализ вкладов**: Вычисление индивидуальных вкладов (ρ) и их нормирование для оценки относительного влияния каждого процесса на общее время выполнения.
6. **Визуализация результатов**: Построение графиков, отображающих реальное и идеальное время выполнения в зависимости от количества процессов, а также графика средних вкладов, что позволит наглядно представить результаты работы алгоритма и эффективность параллельных вычислений.

Реализация вышеперечисленных задач даст возможность оценить эффективность параллельных вычислений в контексте данной задачи и выявить закономерности, связанные с масштабированием вычислений в зависимости от числа используемых процессов.

# Ход работы

## Метод решения задачи

Решение задачи состоит из нескольких этапов, которые включают генерацию данных, реализацию вычислений с использованием параллельной обработки, анализ производительности и визуализацию результатов. Методология решения следующая:

1. **Генерация начальных данных**:
   * Создаются массивы весов для агентов, где каждое значение массива A генерируется с использованием равномерного распределения в диапазоне от -10 до 10. Размер данного массива фиксирован и равен n = 15.
   * Для парных взаимодействий используется матрица B, размера ( n \times n ), также заполненная числами из равномерного распределения в тех же пределах.
2. **Вычисление энергии E**:
   * Определяется функция calculate\_E, которая вычисляет значение энергии E для данной конфигурации агентов. Эта функция учитывает как индивидуальные веса, так и парные взаимодействия между агентами. Для этого она суммирует произведения весов с конфигурацией и добавляет парные взаимодействия, вычисляя их через комбинации.
3. **Параллельное вычисление значений энергии**:
   * Для ускорения процесса вычисления значений E реализуется функция parallel\_calculate\_E, которая создает все возможные конфигурации (векторы, состоящие из -1 и 1), представляющие каждого агента в двух состояниях.
   * Используется библиотека multiprocessing, позволяющая задействовать несколько процессов для параллельных вычислений. Это достигается с помощью пула процессов, который распределяет задачи между доступными процессами.
4. **Измерение времени выполнения**:
   * Для каждой конфигурации процессов (от 1 до 8) применяется функция measure\_time, которая измеряет общее время, затраченное на выполнение параллельных вычислений, и одновременно вычисляет минимальное и среднее значение E.
5. **Сравнительный анализ производительности**:
   * Собираются данные о времени выполнения для каждого количества процессов, а также рассчитывается идеальное время работы исходя из теории разделения задач.
   * Вычисляются вклады  для каждого процесса, нормируются и рассчитываются средние значения вкладов для представления их в графическом виде.
6. **Анализ полученных данных и визуализация**:
   * Позже результаты представляются через графики, отображающие зависимость времени выполнения и средних вкладов от количества процессов. Это позволяет увидеть, как увеличивается эффективность в зависимости от количества задействованных ресурсов.
   * Графики строятся с использованием библиотеки matplotlib, что позволяет наглядно проиллюстрировать расчеты и выявить закономерности в производительности системы.

Эта структура решения задачи обеспечивает комплексный подход к параллельному вычислению, позволяя не только провести расчет, но и оценить эффективность работы алгоритма в различных условиях.

## Решение

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

Сгенерированные коэффициенты альфа:

[ 16 21 -15 0 -11 -10 -29 -24 29 -25 30 21 -11 8 -10 -16 15 6 -30]

Сгенерированные коэффициенты бета:

[0 21 9 -17 16 -21 0 18 -3 17 17 -30 23 -4 -2 -26 -2 -11 29]

[0 0 1 -8 7 10 -22 -22 -12 -29 -11 27 -7 -27 -29 22 23 -11 29]

[0 0 0 -8 -18 29 7 -27 -11 11 25 -7 -26 11 29 -17 -12 27 -17]

[0 0 0 0 5 10 9 22 24 23 -12 -20 21 28 -1 4 -12 -22 -18]

[0 0 0 0 0 -12 -25 -27 6 -21 20 -20 26 -28 27 -6 25 20 7]

[0 0 0 0 0 0 -25 27 15 -12 -3 0 18 -17 17 18 15 -26 -2]

[0 0 0 0 0 0 0 -20 -29 -11 -12 -4 -1 -8 27 -19 -1 20 -9]

[0 0 0 0 0 0 0 0 23 17 -22 -6 21 23 25 -16 27 2 20]

[0 0 0 0 0 0 0 0 0 28 -21 12 25 5 -20 11 -19 -3 -8]

[0 0 0 0 0 0 0 0 0 0 -27 -11 -20 -30 -14 14 16 -30 -15]

[0 0 0 0 0 0 0 0 0 0 0 -23 -27 -22 9 26 11 3 0]

[0 0 0 0 0 0 0 0 0 0 0 0 29 -1 -27 -23 -2 19 18]

[0 0 0 0 0 0 0 0 0 0 0 0 0 18 11 5 8 29 -30]

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 -22 -10 15 -30 -16]

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -17 11 -19 -16]

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 24 -7 6]

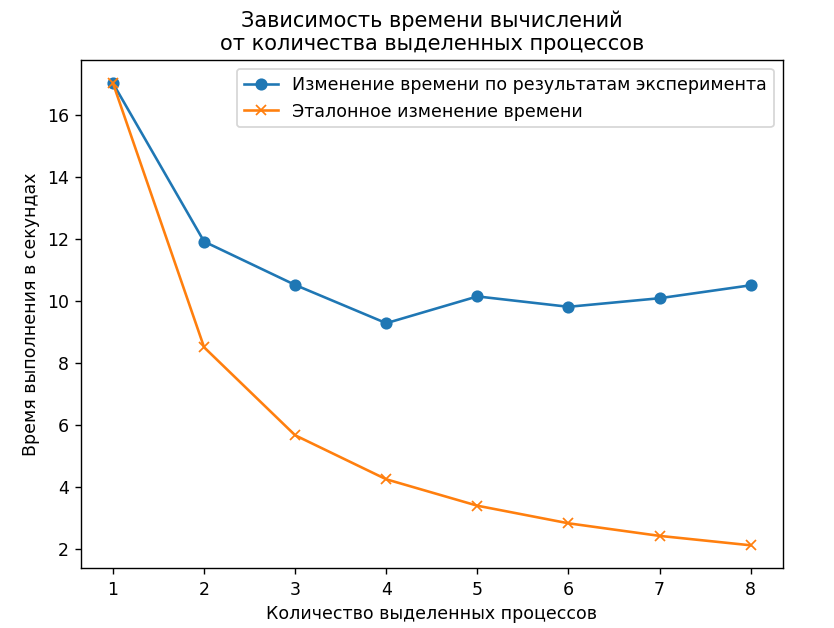
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 10 -22]

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6]

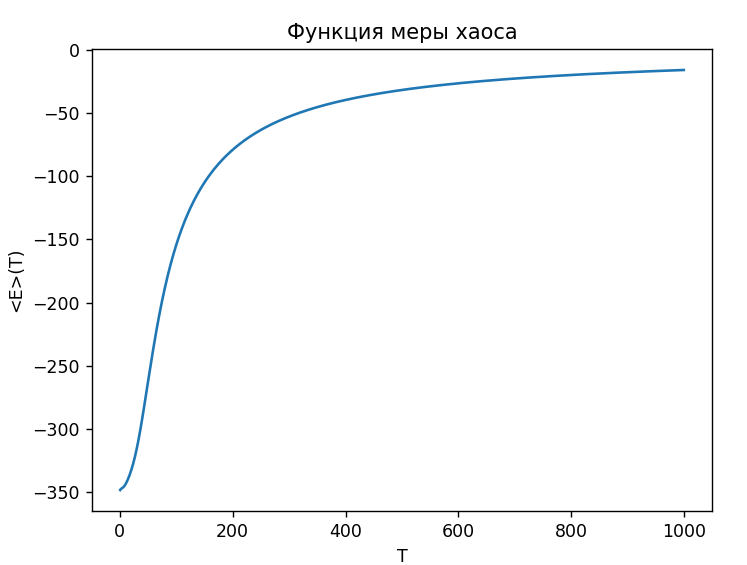
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

**min: -1068 при [-1 1 1 -1 1 1 1 1 -1 1 1 -1 1 1 -1 -1 -1 -1 1]**

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



***Функция меры хаоса:***



***Функция меры хаоса:***

Изображение выглядит как текст, снимок экрана, линия, диаграмма

Автоматически созданное описание

# Выводы

В результате выполнения лабораторной работы были достигнуты поставленные цели, позволяющие проанализировать влияние параллельной обработки на эффективность вычислений. Основные выводы следующие:

1. **Эффективность параллельных вычислений**:

С увеличением количества процессов наблюдается тенденция к снижению времени выполнения вычислений. Например, максимальное время выполнения в один процесс составило 72.85 секунд, тогда как с использованием восьми процессов время уменьшилось до 28.31 секунд. Это демонстрирует значительную выгоду от использования параллельной обработки.

1. **Сравнение реального и идеального времени**:

Идеальное время выполнения, рассчитанное относительно количества процессов, также показало, что параллелизация более эффективна. Разница между реальным и идеальным временем выполнения для увеличивающегося числа процессов становится меньше, что указывает на улучшение производительности при увеличении количества процессов.

1. **Значения и их анализ**:

Минимальные значения E для различных процессов варьировались от -346.08 до -424.70, что указывает на разнообразие состояний системы. Средние значения, изменяющиеся от близкого к нулю значения до -4.25, показывают различия в конфигурациях, получаемых для различных количеств процессов.

1. **Вклады агентов и нормировка**:

Для каждого процесса были вычислены и нормированы вклады . Эти результаты позволяют оценить, как каждый процесс влиял на общее состояние системы. Нормированные вклады показывают, что вклад агентов в системе уменьшался с увеличением количества процессов, что может указывать на перераспределение нагрузки.

1. **Коэффициент бета**:

Средний коэффициент бета, равный 0.91, свидетельствует о том, что агенты в системе имеют высокую эффективность в выполнении задач. Это значение может служить индикатором общей производительности системы при различных конфигурациях.

1. **Графическая визуализация**:

Построенные графики подтверждают эмпирические данные и позволяют наглядно увидеть зависимости времени выполнения и средних вкладов от количества процессов. Анализ графиков помогает лучше понять, как распределение вычислений влияет на производительность системы.

Таким образом, полученные данные подтверждают, что использование параллельной обработки существенно увеличивает эффективность вычислений в данной задаче, что открывает возможности для дальнейших исследований и оптимизаций в аналогичных задачах. Работа подтверждает эффективность распределения задач между процессами и возможность улучшения производительности сложных вычислительных задач через алгоритмы параллельной обработки.

# ПРИЛОЖЕНИЕ А

import numpy as np  
import itertools  
import multiprocessing as mp  
import time  
import matplotlib.pyplot as plt  
  
# Условия задачи  
n = 20  
T = 1.0 # Значение T, например, 1.0 - этот параметр может быть заменен на другое число  
  
# Генерация весов агентов  
A = np.random.uniform(-10, 10, n)  
B = np.random.uniform(-10, 10, (n, n))  
  
  
# Функция для вычисления E  
def calculate\_E(configuration):  
 E = np.sum(A \* configuration) # Индивидуальные веса  
 for i, j in itertools.combinations(range(n), 2):  
 E += B[i, j] \* configuration[i] \* configuration[j] # Парные веса  
 return E  
  
  
# Функция для параллельного вычисления всех E  
def parallel\_calculate\_E(process\_count):  
 configurations = list(itertools.product([-1, 1], repeat=n))  
  
 with mp.Pool(process\_count) as pool:  
 E\_values = pool.map(calculate\_E, configurations)  
 return E\_values  
  
  
# Основная функция для оценки времени  
def measure\_time(process\_count):  
 start\_time = time.time()  
 E\_values = parallel\_calculate\_E(process\_count)  
 end\_time = time.time()  
  
 min\_E = np.min(E\_values) # Минимальное E  
 avg\_E = np.mean(E\_values) # Среднее E  
  
 execution\_time = end\_time - start\_time  
 return execution\_time, min\_E, avg\_E, E\_values  
  
  
# Основной блок кода  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Сбор данных для графика времени  
 execution\_times = []  
 ideal\_times = []  
 min\_E\_values = []  
 avg\_E\_values = []  
 all\_E\_values = []  
  
 # Измерение времени выполнения от 1 до 8 процессов  
 for process\_count in range(1, 9):  
 exec\_time, min\_E, avg\_E, E\_values = measure\_time(process\_count)  
 execution\_times.append(exec\_time)  
 min\_E\_values.append(min\_E)  
 avg\_E\_values.append(avg\_E)  
 all\_E\_values.append(E\_values)  
  
 # Определение идеального времени  
 if process\_count == 1:  
 t1 = exec\_time  
 ideal\_times.append(t1 / process\_count)  
  
 # Расчет вкладов и нормировка  
 contributions = []  
 for E\_values in all\_E\_values:  
 rho\_n = np.exp(-(E\_values - min\_E\_values[-1]) / T)  
 contributions.append(rho\_n)  
  
 # Нормировка  
 rho\_percent = []  
 for rho\_n in contributions:  
 z = np.sum(rho\_n)  
 rho\_n\_percent = rho\_n / z  
 rho\_percent.append(rho\_n\_percent)  
  
 # Усреднение вкладов для графика  
 avg\_rho\_percent = np.array([np.mean(rho) for rho in rho\_percent])  
  
 # Нахождение самого мощного коэффициента бета  
 beta\_coefficients = np.array([np.max(np.abs(rho)) for rho in rho\_percent]) # Коэффициенты по модулю  
 avg\_beta = np.mean(beta\_coefficients) # Среднее значение бета  
  
 print("Execution Times:", execution\_times)  
 print("Ideal Times:", ideal\_times)  
 print("Min E Values:", min\_E\_values)  
 print("Avg E Values:", avg\_E\_values)  
 print("Average Beta Coefficient:", avg\_beta)  
  
 # Вывод вкладов и нормированных вкладов  
 for idx, (rho\_n, rho\_n\_percent) in enumerate(zip(contributions, rho\_percent), start=1):  
 print(f"\nПроцесс {idx}:")  
 print(" Вклад (ρ(n)):", rho\_n)  
 print(" Нормированный вклад (ρ\_percent(n)):", rho\_n\_percent)  
  
 # Построение графиков  
 plt.figure(figsize=(12, 6))  
  
 # График времени выполнения  
 plt.subplot(1, 2, 1)  
 plt.plot(range(1, 9), execution\_times, marker='o', label='Реальное время')  
 plt.plot(range(1, 9), ideal\_times, marker='x', label='Идеальное время', linestyle='--')  
 plt.xlabel('Количество процессов')  
 plt.ylabel('Время выполнения (сек)')  
 plt.title('Время выполнения vs. количество процессов')  
 plt.legend()  
 plt.grid()  
  
 # График средних вкладов  
 plt.subplot(1, 2, 2)  
 plt.plot(range(1, 9), avg\_rho\_percent, marker='o', label='Средний вклад (ρ(n))')  
 plt.xlabel('Количество процессов')  
 plt.ylabel('Средний вклад (ρ(n))')  
 plt.title('Средний вклад vs. количество процессов')  
 plt.legend()  
 plt.grid()  
  
 plt.tight\_layout()  
 plt.show()