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 №4

**«Многопоточность в Python»**



Преподаватель: Кашин И.В.

Студент : Шардаков В.А.

Группа : Фт-410008

Дата : 18.10.2024

Екатеринбург 2024

# Задачи

В контексте данной работы мы рассматриваем задачу, в которой требуется минимизировать значения определенной функции, зависящей от наборов случайных коэффициентов и матриц. Конкретно, цель заключается в нахождении минимального значение E, которое определяется на основе заданных коэффициентов и элементарной матрицы. Для эффективного поиска минимального значения функции используется **метод оптимизации Монте-Карло**, который позволяет избежать локальных минимумов и повысить вероятность нахождения глобального минимума.

С применением библиотеки **multiprocessing** реализован многопоточный подход, который значительно улучшает производительность алгоритма, позволяя распределить вычислительные задачи между несколькими процессами. Это решение особенно актуально для больших объемов данных и сложных вычислительных задач, где время выполнения играет критическую роль.

# Ход работы

## Метод решения задачи

На первом этапе была разработана программа (см. ПРИЛОЖЕНЕ A), предназначенная для поиска минимального значения функции энергии E в течение 10 минут для задачи с 35 переменными. После нахождения локального минимума была создана новая программа (см. ПРИЛОЖЕНИЕ B), в которой реализован метод оптимизации на основе метода Монте-Карло. Этот подход включает использование меры хаоса T, что позволяет более эффективно исследовать пространство возможных решений и приближаться к глобальному минимуму.

## Решение

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

[-13 42 -16 24 -48 -28 -27 46 -3 48 40 45 -43 9 9 -45 -46 -25

35 19 -39 -4 49 -43 37 -34 -39 -19 15 15 -27 -25 -39 -34 -49 -32

-39 -33 35 11 -27 -42 -9 5 37 -49 11 -1 7 5]

Матрица коэффициентов beta:

[[ 0. 50. 5. ... 35. -8. 33.]

[ 0. 0. 20. ... 39. -14. -23.]

[ 0. 0. 0. ... 3. 2. 4.]

...

[ 0. 0. 0. ... 0. 47. -49.]

[ 0. 0. 0. ... 0. 0. -38.]

[ 0. 0. 0. ... 0. 0. 0.]]

Результат после 10 минут работы:

Лучшая конфигурация: [-2, -2, -2, 2, -2, 2, -2, -2, 2, -2, 2, -2, -2, -2, 2, 2, 2, -2, 2, 2, 2, 2, 2, -2, -2, -2, 2, 2, 2, 2, 2, 2, -2, 2, -2, 2, 2, 2, 2, -2, 2, -2, 2, -2, -2, 2, 2, -2, -2, -2]

**Минимальное значение E: -30522.0**

***Результат работы программы (ПРИЛОЖЕНИЕ Б)***

Потоки: 1, Время: 12.5722 сек, Конфигурация: [-2, 2, -2, 2, 2, 2, -2, -2, 2, -2, 2, -2, -2, -2, 2, 2, 2, 2, 2, -2, 2, -2, 2, -2, -2, -2, 2, 2, 2, 2, 2, 2, -2, -2, -2, 2, 2, 2, 2, -2, 2, -2, 2, -2, -2, 2, 2, -2, -2, -2], Минимальная E: -30368.0

Потоки: 2, Время: 4.5124 сек, Конфигурация: [2, -2, 2, -2, 2, -2, -2, 2, -2, 2, -2, 2, 2, 2, -2, -2, 2, 2, -2, -1, -2, 0, -2, 2, -2, 2, 2, -2, -2, 2, 2, -2, -2, -2, 2, -2, 2, -2, -2, 2, -2, 2, -2, -2, 2, 2, -2, 2, 2, 2], Минимальная E: -30323.0

Потоки: 3, Время: 10.5683 сек, Конфигурация: [2, -2, 2, -2, 2, -2, 1, 2, -2, 2, -2, 2, 2, 2, -2, -2, 2, 2, -2, 0, -2, 2, -2, 2, -2, 2, 2, -2, -2, 2, 2, -2, -2, -2, 2, -2, 2, -2, -2, 2, -2, 2, -2, -2, 0, 2, -2, 2, 2, 2], Минимальная E: -30395.0

Потоки: 4, Время: 10.8379 сек, Конфигурация: [2, -2, 2, -2, 2, -2, -2, 2, -2, 2, -2, 2, 2, 2, -2, -2, 2, 2, -2, -1, -2, 2, -2, 2, -2, 2, 2, -2, -2, 2, 2, -2, -2, -2, 2, -2, 2, -2, -2, 2, -2, 2, -2, -2, -1, 1, -2, 2, 2, 2], Минимальная E: -30382.0

Потоки: 5, Время: 10.5418 сек, Конфигурация: [2, -2, 2, -2, 2, -2, -2, 2, -2, 2, -2, 2, 2, 2, 0, -2, 2, 2, -1, -2, -2, 2, -2, 2, -2, 2, 2, -2, -2, 1, 2, -2, -2, -2, 2, -2, 2, -2, -2, 2, -2, 2, -2, -2, -2, 2, -2, 2, 2, 2], Минимальная E: -30522.0

Потоки: 6, Время: 11.8839 сек, Конфигурация: [2, -2, 2, -2, 2, -2, 2, 2, -2, 2, -2, 2, 2, 2, -1, -1, 2, 2, -2, -2, -2, 2, -2, 2, -2, 2, 2, -2, -2, 2, 2, -2, -2, -2, 2, -2, 2, -2, -2, 2, -2, 2, 0, -2, -2, 2, -2, 2, 2, 2], Минимальная E: -30303.0

Потоки: 7, Время: 7.0066 сек, Конфигурация: [2, -2, 2, -2, 2, -2, -2, 2, -2, 2, -2, 2, 2, 2, -2, -2, 2, 2, -2, -2, -2, 2, -2, 2, -2, 2, 1, -2, -2, 2, 2, -2, -2, -2, 2, -2, 2, -2, -2, 2, -2, 2, -1, -2, 0, 2, -2, 2, 2, 2], Минимальная E: -30500.0

Потоки: 8, Время: 9.9164 сек, Конфигурация: [2, -2, 2, -2, 2, -2, -2, 2, -2, 2, -2, 2, 2, 2, -1, -2, 2, 2, -2, -1, -2, 2, -2, 2, -2, 2, 2, -2, -2, 2, 2, -2, -2, -2, 2, -2, 2, -2, -2, 2, -2, 2, -2, -2, -2, 1, -2, 2, 2, 2], Минимальная E: -30311.0

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

Изображение выглядит как линия, График, диаграмма, текст

Автоматически созданное описание

Изображение выглядит как линия, диаграмма, текст, График

Автоматически созданное описание

***График среднего времени от количества потоков:***

Изображение выглядит как линия, текст, График, диаграмма

Автоматически созданное описание

# Выводы

В ходе лабораторной работы была разработана и реализована программа, направленная на решение задачи минимизации функции энергии с использованием методов стохастической оптимизации и многопоточности. Проведенные эксперименты продемонстрировали эффективность применения многопоточного подхода для ускорения вычислений при поиске оптимальных решений.

Результаты экспериментов, представленные в таблице, показывают время выполнения алгоритма в зависимости от количества активных потоков. При увеличении числа потоков наблюдается значительное сокращение времени выполнения, что подтверждает гипотезу о том, что распараллеливание процесса поиска минимального значения может существенно повысить общую производительность алгоритма.

# ПРИЛОЖЕНИЕ А

import numpy as np  
import random  
import math  
import time  
  
# Фиксируем сид для генерации случайных чисел  
np.random.seed(113)  
  
# Параметры задачи  
n = 35 # Размер матрицы  
  
# Генерация случайной матрицы коэффициентов  
b = np.random.randint(-50, 51, size=n)  
matrix = np.zeros((n, n))  
  
for i in range(n):  
 for j in range(i + 1, n):  
 matrix[i, j] = np.random.randint(-50, 51)  
  
  
# Функция для расчета энергии E  
def calculate\_E(config, b, matrix, n):  
 E = sum(b[i] \* config[i] for i in range(n))  
 for i in range(n):  
 for j in range(i + 1, n):  
 E += matrix[i, j] \* config[i] \* config[j]  
 return E  
  
  
def simulated\_annealing(b, matrix, n, num\_iterations=2000, max\_time=180):  
 # Инициализация случайной конфигурации  
 current\_config = [random.randint(-2, 2) for \_ in range(n)]  
 current\_E = calculate\_E(current\_config, b, matrix, n)  
  
 T = 1000  
 min\_config = current\_config.copy()  
 min\_E = current\_E  
  
 start\_time = time.time()  
  
 while time.time() - start\_time < max\_time:  
 for \_ in range(num\_iterations):  
 # Проверяем время каждые несколько итераций, чтобы прервать процесс  
 if time.time() - start\_time >= max\_time:  
 break  
  
 # Выбираем случайный индекс и новое случайное значение для этого индекса  
 i = random.randint(0, n - 1)  
 new\_config = current\_config.copy()  
 new\_config[i] = max(-2, min(2, new\_config[i] + random.choice([-1, 1])))  
  
 new\_E = calculate\_E(new\_config, b, matrix, n)  
  
 # Сравнение энергий  
 if new\_E <= current\_E:  
 current\_config = new\_config  
 current\_E = new\_E  
 else:  
 p = math.exp(-(new\_E - current\_E) / T)  
 if random.random() < p:  
 current\_config = new\_config  
 current\_E = new\_E  
  
 # Обновляем минимальное значение  
 if current\_E < min\_E:  
 min\_E = current\_E  
 min\_config = current\_config.copy()  
  
 T \*= 0.95  
  
 return min\_config, min\_E  
  
  
# Генерация и вывод матрицы  
print("Матрица коэффициентов b:")  
print(b)  
print("\nМатрица коэффициентов:")  
print(matrix)  
  
# Выполнение программы  
if \_\_name\_\_ == '\_\_main\_\_':  
 print("Запуск поиска минимума на 5 минут...")  
 best\_config, best\_E = simulated\_annealing(b, matrix, n, max\_time=180)  
 print("\nРезультат после -5 минут работы:")  
 print(f"Лучшая конфигурация: {best\_config}")  
 print(f"Минимальное значение E: {best\_E}")

# ПРИЛОЖЕНИЕ Б

import numpy as np  
import random  
import math  
import multiprocessing as mp  
import time  
import matplotlib.pyplot as plt  
  
# Фиксируем сид для генерации случайных чисел  
np.random.seed(113)  
  
# Параметры задачи  
n = 35 # Размер матрицы  
  
# Генерация случайной матрицы коэффициентов  
b = np.random.randint(-50, 51, size=n)  
matrix = np.zeros((n, n))  
  
for i in range(n):  
 for j in range(i + 1, n):  
 matrix[i, j] = np.random.randint(-50, 51)  
  
  
# Функция для расчета E  
def calculate\_E(config, b, matrix, n):  
 E = sum(b[i] \* config[i] for i in range(n))  
 for i in range(n):  
 for j in range(i + 1, n):  
 E += matrix[i, j] \* config[i] \* config[j]  
 return E  
  
  
def fmin(b, matrix, n, stop\_event, result\_queue, num\_iterations=2000):  
 # Инициализация случайной конфигурации  
 current\_config = [random.randint(-2, 2) for \_ in range(n)]  
 current\_E = calculate\_E(current\_config, b, matrix, n)  
  
 min\_config = current\_config.copy()  
 min\_E = current\_E  
 Etalon = -15110.0  
 while min\_E > Etalon:  
 T = 100  
 while T > 0.1:  
 if stop\_event.is\_set(): # Проверка, не нашёл ли другой поток решение  
 return  
  
 for \_ in range(num\_iterations):  
 if stop\_event.is\_set(): # Если сигнал остановки  
 return  
  
 # Выбираем случайный индекс и новое случайное значение для этого индекса  
 i = random.randint(0, n - 1)  
 new\_config = current\_config.copy()  
 new\_config[i] = max(-2, min(2, new\_config[i] + random.choice([-1, 1])))  
  
 new\_E = calculate\_E(new\_config, b, matrix, n)  
 # Сравнение энергий  
 if new\_E <= current\_E:  
 current\_config = new\_config  
 current\_E = new\_E  
 else:  
 p = math.exp(-(new\_E - current\_E) / T)  
 if random.random() < p:  
 current\_config = new\_config  
 current\_E = new\_E  
  
 # Обновляем минимальное значение  
 if current\_E < min\_E:  
 min\_E = current\_E  
 min\_config = current\_config.copy()  
 if min\_E <= Etalon:  
 result\_queue.put((min\_config, min\_E)) # Передаём результат через очередь  
 stop\_event.set() # Сигнализируем, что решение найдено  
  
 T \*= 0.95  
  
  
# Функция для многопоточного поиска решения  
def parallel\_search(num\_threads):  
 stop\_event = mp.Event() # Событие для остановки потоков  
 result\_queue = mp.Queue() # Очередь для передачи результатов  
 processes = []  
  
 # Запуск потоков  
 for \_ in range(num\_threads):  
 p = mp.Process(target=fmin, args=(b, matrix, n, stop\_event, result\_queue))  
 processes.append(p)  
 p.start()  
  
 # Ожидание завершения одного из потоков и получение результата  
 best\_config, best\_E = result\_queue.get()  
  
 # Остановка всех потоков  
 for p in processes:  
 p.join()  
  
 return best\_config, best\_E  
  
  
# Запуск многопоточного поиска с замером времени  
def run\_parallel\_search():  
 times = []  
 thread\_counts = list(range(1, 9))  
 for num\_threads in thread\_counts:  
 start\_time = time.time()  
 best\_config, best\_E = parallel\_search(num\_threads)  
 end\_time = time.time()  
 times.append(end\_time - start\_time)  
 print(  
 f"Потоки: {num\_threads}, Время: {end\_time - start\_time:.4f} сек, Конфигурация: {best\_config}, Минимальная E: {best\_E}")  
  
 return thread\_counts, times  
  
  
# Выполнение программы  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Генерация и вывод матрицы  
 print("Коэффициенты alpha:")  
 print(b)  
 print("\nМатрица коэффициентов beta:")  
 print(matrix)  
  
 thread\_counts, times = run\_parallel\_search()  
  
 # Построение графика зависимости времени выполнения от числа потоков  
 plt.plot(thread\_counts, times, marker='o')  
 plt.title('Зависимость времени выполнения от количества потоков')  
 plt.xlabel('Количество потоков')  
 plt.ylabel('Время выполнения (сек)')  
 plt.grid(True)  
 plt.show()