Государственное образовательное учреждение высшего образования

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

09.03.02 310008

**«Параллельные вычислительные системы»**

Семестр 7

**ОТЧЁТ**

по Лабораторной работе №2

**«Многопоточность в Python на примере расчёта объёма сложной фигуры в сферической системе координат»**



Преподаватель: Кашин И.В.

Студент : Шардаков В.А.

Группа : Фт-410008

Дата : 27.09.2024

Екатеринбург 2024

# Задачи

В рамках данной лабораторной работы необходимо выполнить следующие задачи:

1. Построение фигуры в сферической системе координат:

Используя математические оценки, разработать и визуализировать трехмерную фигуру в сферических координатах. Для задания формы фигуры можно использовать функцию, определяющую радиус в зависимости от углов .

1. Вычисление объема фигуры методом Монте-Карло:

Реализовать алгоритм Монте-Карло для оценки объема построенной фигуры. Генерировать случайные точки в ограничивающем параллелепипеде и проверять, попадают ли они в заданную фигуру, затем оценить объем на основе соотношения точек внутри фигуры к общему количеству сгенерированных точек.

1. Использование библиотеки multiprocessing для многопоточной генерации точек:

Внедрить многопоточность для ускорения процесса генерации случайных точек, используя возможности библиотеки multiprocessing. Сравнить время выполнения алгоритма при различном количестве потоков (до 8 потоков) на четырехъядерном ноутбуке. Провести анализ производительности и визуализировать результаты, чтобы определить, насколько многопоточность влияет на эффективность вычислений.

# Ход работы

## Метод решения задачи

В данной лабораторной работе используется метод Монте-Карло для оценки объема фигуры, заданной в сферических координатах функцией . Процесс состоит из нескольких ключевых этапов, описанных ниже:

**1. Построение фигуры в сферической системе координат**

Сначала создается трехмерная фигура, описываемая радиусом, зависящим от угла . Для визуализации фигуры в сферических координатах используется библиотека matplotlib. Углы  и варьируются от 0 до  и от 0 до 2\ соответственно, и на их основе вычисляются декартовые координаты точек (x, y, z) на поверхности фигуры:

* 
* 
* 

Здесь (r) определяется как , что дает геометрическую интерпретацию фигуры в 3D-пространстве.

**2. Оценка объема методом Монте-Карло**

Для вычисления объема фигуры применяется метод Монте-Карло, который включает следующие шаги:

* Генерация случайных точек в кубе с заданными границами: ([-1, 1]) по всем координатам (x, y, z).
* Для каждой точки вычисляется расстояние (r) от начала координат до точки, а также угол  относительно оси z.
* Затем рассчитывается значение функции радиуса surf\_point\_r = .
* Путем сравнения сгенерированного расстояния (r) с surf\_point\_r определяется, попадает ли точка внутрь фигуры. Если точка считается находящейся внутри фигуры, и счетчик count\_inside увеличивается на единицу.
* По завершению генерации (n) точек вычисляется объем фигуры по формуле:

volume = (count\_inside / n\_points) \* 8

где (8) — это объем куба, в котором мы генерировали точки.

**3. Параллельная обработка с использованием библиотеки multiprocessing**

Для ускорения процесса генерации случайных точек и оценки объема применяется многопоточность с использованием модуля **multiprocessing**. Процесс включает следующие шаги:

* Определяется количество доступных процессоров, и общее количество точек делится на количество процессов (ядер), чтобы обеспечить равномерное распределение задач.
* Каждый процесс запускает свой экземпляр функции **monte\_carlo\_volume**, получая подмножество точек для обработки.
* Результаты вычислений агрегируются, и вычисляется среднее значение объема, что позволяет улучшить точность результатов.
* Для оценки производительности проводятся эксперименты с различным числом потоков (от 1 до 8) с целью измерения времени выполнения и анализа ускорения.

**4. Визуализация результатов**

Результаты работы, включая график зависимости времени выполнения от количества процессоров, также выводятся с использованием библиотеки **matplotlib**. Это позволяет наглядно представить, как увеличение числа потоков влияет на скорость выполнения задачи, а также проверить достижения в использовании многопоточности.

Такое подробное описание метода позволяет читателю понять структуру решения задачи, используемые алгоритмы и библиотеки, а также смысл и значение каждой части кода.

## Решение

График функции, построенный на Geogebra:

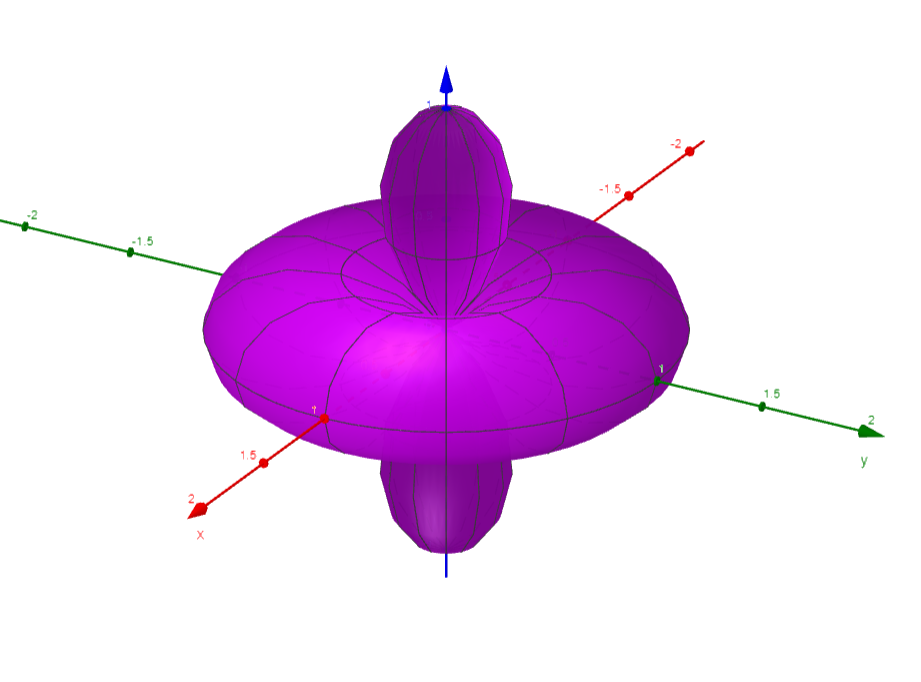
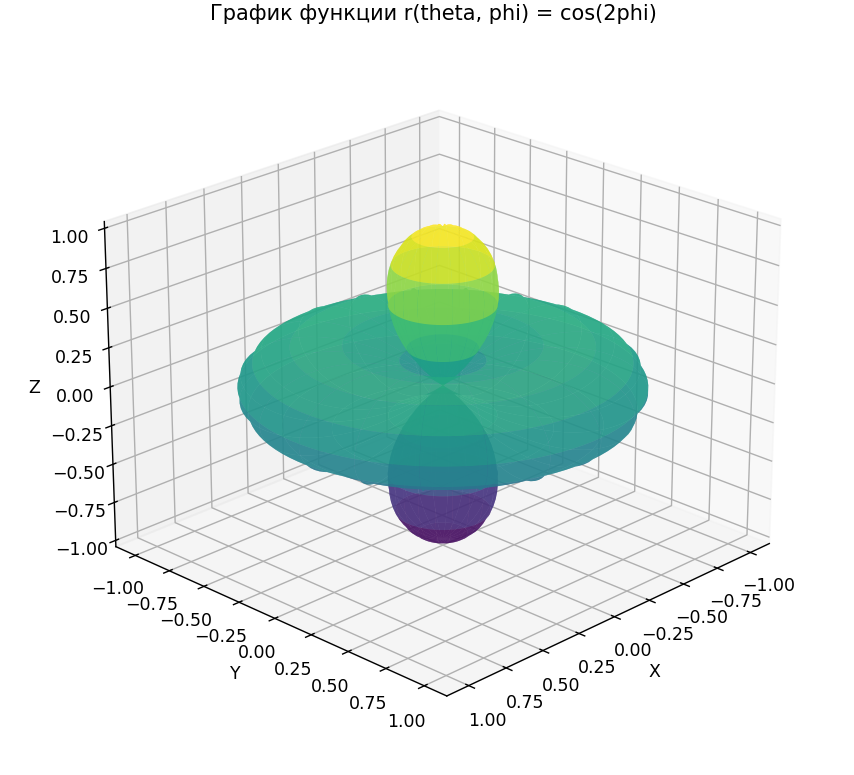


График функции, построенный при работе приложения:



Результат работы программы при 1 000 000 точек:

Процессоров: 1, Оценка объёма: 1.6299, Время выполнения: 9.0283 секунд

Процессоров: 2, Оценка объёма: 1.6390, Время выполнения: 5.1132 секунд

Процессоров: 3, Оценка объёма: 1.6325, Время выполнения: 3.9189 секунд

Процессоров: 4, Оценка объёма: 1.6340, Время выполнения: 3.6229 секунд

Процессоров: 5, Оценка объёма: 1.6273, Время выполнения: 3.5008 секунд

Процессоров: 6, Оценка объёма: 1.6334, Время выполнения: 3.4916 секунд

Процессоров: 7, Оценка объёма: 1.6339, Время выполнения: 4.4414 секунд

Процессоров: 8, Оценка объёма: 1.6333, Время выполнения: 4.6139 секунд

График зависимости времени от количества потоков:

Изображение выглядит как текст, линия, График, диаграмма

Автоматически созданное описание

Результат работы программы при 2 000 000 точек:

Процессоров: 1, Оценка объёма: 1.6288, Время выполнения: 16.3741 секунд

Процессоров: 2, Оценка объёма: 1.6298, Время выполнения: 8.9806 секунд

Процессоров: 3, Оценка объёма: 1.6320, Время выполнения: 6.7093 секунд

Процессоров: 4, Оценка объёма: 1.6335, Время выполнения: 6.1714 секунд

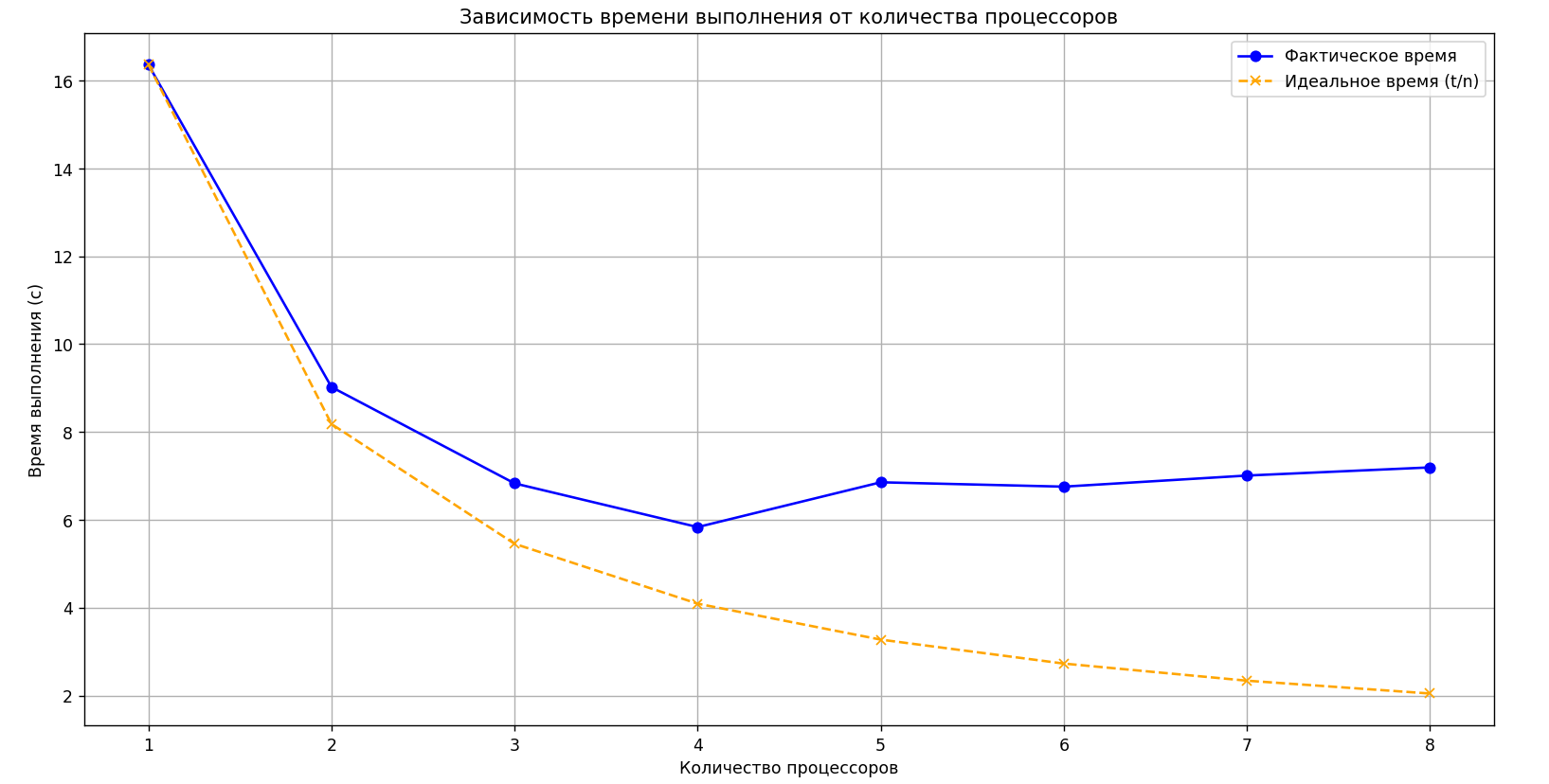
Процессоров: 5, Оценка объёма: 1.6317, Время выполнения: 6.8384 секунд

Процессоров: 6, Оценка объёма: 1.6320, Время выполнения: 7.0099 секунд

Процессоров: 7, Оценка объёма: 1.6281, Время выполнения: 7.0664 секунд

Процессоров: 8, Оценка объёма: 1.6317, Время выполнения: 7.4743 секунд

График зависимости времени от количества потоков:



# Выводы

В ходе выполнения лабораторной работы была применена методика Монте-Карло для оценки объема фигуры, заданной радиусом в сферических координатах функцией . Результаты экспериментов показали, что применение многопоточности существенно ускоряет процесс вычислений и позволяет более эффективно использовать ресурсы ноутбука с четырьмя физическими ядрами и восемью потоками.

Результаты экспериментов

При запуске программы с 1 000 000 случайными точками мы наблюдали следующие результаты:

* При использовании 1 процессора объем был оценен в (1.6299) с временем выполнения (9.0283) секунд.
* С увеличением числа процессоров до 8 объем оставался в пределах (1.6333) с временем выполнения, сократившимся до (4.6139) секунд.

Аналогичные результаты были получены при использовании 2 000 000 точек, где объём оценивался в (1.6288) с временем выполнения (16.3741) секунд на одном процессоре, и варьировался в диапазоне от (1.6281) до (1.6335) при использовании от 2 до 8 процессоров. Это подтверждает, что увеличение числа процессов приводит к снижению времени вычисления, что подчеркивает эффективность многопоточной реализации.

Точность метода

Метод Монте-Карло характеризуется вероятностной природой, что в свою очередь приводит к небольшому разбросу результатов. Объемы, рассчитанные при различных запусках программы, показывают некоторую степень изменчивости, которая может быть вызвана случайностью в генерации точек. Погрешности в оценке объема находились в пределах нескольких сотых, что не критично для большинства практических приложений. Это указывает на достаточную точность метода для данной задачи, однако стоит учитывать, что с увеличением числа точек точность оценки, как правило, возрастает.

Ограничения аналитического решения

К сожалению, в данной работе не удалось получить аналитическое выражение для объема фигуры, заданной функцией . Поэтому метод Монте-Карло стал основным инструментом для численного вычисления, что подчеркивает его актуальность и универсальность в задачах, где аналитическое решение невозможно или затруднительно.

Таким образом, проведенные эксперименты подтвердили эффективность метода Монте-Карло и его потенциальные возможности в вычислительных задачах с многопоточностью, а также продемонстрировали его применимость для оценки объемов сложных фигур в сферических координатах.

# ПРИЛОЖЕНИЕ А

import numpy as np  
import matplotlib.pyplot as plt  
from multiprocessing import Pool  
import time  
  
  
def monte\_carlo\_volume(n\_points):  
 count\_inside = 0  
  
 # Параметры параллелепипеда  
 x\_bound = [-1, 1]  
 y\_bound = [-1, 1]  
 z\_bound = [-1, 1]  
  
 for \_ in range(n\_points):  
 # Генерация случайных точек  
 x = np.random.uniform(\*x\_bound)  
 y = np.random.uniform(\*y\_bound)  
 z = np.random.uniform(\*z\_bound)  
  
 # Вычисляем полярные координаты  
 r = np.sqrt(x \*\* 2 + y \*\* 2 + z \*\* 2)  
 if r == 0:  
 continue  
  
 phi = np.arccos(z / r) # Угол от положительной оси Z  
 # Параметрическая функция радиуса  
 surf\_point\_r = abs(np.cos(2 \* phi))  
  
 if r <= surf\_point\_r:  
 count\_inside += 1  
  
 volume = (count\_inside / n\_points) \* 8  
 return volume  
  
  
def calculate\_volumes(processors, n\_points):  
 # Вычисление объема методом Монте-Карло с использованием нескольких процессов  
 with Pool(processors) as p:  
 # Деля текущее количество точек между процессами  
 points\_per\_process = n\_points // processors  
 volumes = p.map(monte\_carlo\_volume, [points\_per\_process] \* processors)  
  
 return np.mean(volumes), volumes # Возвращаем также индивидуальные объемы  
  
  
def time\_function(processors, n\_points):  
 start\_time = time.time()  
 volume, individual\_volumes = calculate\_volumes(processors, n\_points)  
 elapsed\_time = time.time() - start\_time  
 print(f'Процессоров: {processors}, Оценка объёма: {volume:.4f}, Время выполнения: {elapsed\_time:.4f} секунд')  
 return elapsed\_time, volume # Возвращаем время и объем  
  
  
def plot\_performance():  
 n\_points = 1000000 # Общее количество точек для Монте-Карло  
 processor\_counts = range(1, 9) # От 1 до 8 процессоров  
 times = []  
 ideal\_times = []  
  
 for pc in processor\_counts:  
 elapsed\_time, volume = time\_function(pc, n\_points)  
 times.append(elapsed\_time)  
 ideal\_times.append(times[0] / pc) # Идеальное время выполнения  
  
 # Построим график зависимости времени от количества процессоров  
 plt.figure()  
 plt.plot(processor\_counts, times, marker='o', label='Фактическое время', color='blue')  
 plt.plot(processor\_counts, ideal\_times, marker='x', linestyle='dashed', label='Идеальное время (t/n)', color='orange')  
 plt.xlabel('Количество процессоров')  
 plt.ylabel('Время выполнения (с)')  
 plt.title('Зависимость времени выполнения от количества процессоров')  
 plt.legend()  
 plt.grid()  
 plt.show()  
  
  
def plot\_spherical\_function():  
 # Определяем углы  
 phi = np.linspace(0, np.pi, 100) # угол от 0 до π  
 theta = np.linspace(0, 2 \* np.pi, 100) # угол от 0 до 2π  
 phi, theta = np.meshgrid(phi, theta) # создаем сетку углов  
  
 # Значение функции в сферических координатах  
 r = np.cos(2 \* phi) # p(theta, phi) = cos(2phi)  
  
 # Преобразуем сферические координаты в декартовые  
 x = r \* np.sin(phi) \* np.cos(theta)  
 y = r \* np.sin(phi) \* np.sin(theta)  
 z = r \* np.cos(phi)  
  
 # Построение графика  
 fig = plt.figure()  
 ax = fig.add\_subplot(111, projection='3d')  
 ax.plot\_surface(x, y, z, cmap='viridis', alpha=0.7, rstride=5, cstride=5)  
  
 ax.set\_xlabel('X')  
 ax.set\_ylabel('Y')  
 ax.set\_zlabel('Z')  
 ax.set\_title('График функции r(theta, phi) = cos(2phi)')  
 plt.show()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Построение графика функции  
 plot\_spherical\_function()  
  
 # Построение графика зависимости времени выполнения от количества процессоров  
 plot\_performance()