Описание результатов

После запуска программы вы получите два времени выполнения — одно для последовательного доступа и другое для случайного. Обычно последовательный доступ будет значительно быстрее, чем случайный, из-за более эффективного использования кэш-памяти.

Заключение

В примере видно, что последовательный доступ к массиву использует кэш- память гораздо более эффективно, чем случайный доступ. Это связано с тем, что при последовательном доступе данные загружаются в кэш блоками, что уменьшает количество обращений к медленной оперативной памяти.

Пример работы программы:

Sequential Sum: 4999999.999999999, Time: 0.123456 seconds

Random Sum: 4999999.999999999, Time: 1.234567 seconds

Пример кода на Python:

import numpy as np import time

# Размер массива N = 10\*\*7

# Создание массива

array = np.random.rand(N)

def sum\_sequential(arr): total = 0

for i in range(len(arr)): total += arr[i]

return total

def sum\_random(arr): total = 0

indices = np.random.permutation(len(arr)) for i in indices:

total += arr[i] return total

# Замер времени для последовательного доступа start\_time = time.time()

sequential\_sum = sum\_sequential(array) sequential\_time = time.time() - start\_time

print(f"Sequential Sum: {sequential\_sum}, Time: {sequential\_time:.6f} seconds")

# Замер времени для случайного доступа start\_time = time.time()

random\_sum = sum\_random(array) random\_time = time.time() - start\_time

print(f"Random Sum: {random\_sum}, Time: {random\_time:.6f} seconds")