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")
```