**Лабораторная №2**

**Выполнил: Ягодяк Павел Сергеевич АИСа-020303-о24**

**Перемножение квадратных матриц размера 4096x4096 с элементами типа double complex (комплексные числа двойной точности).**

**Цель:**

Целью данной лабораторной работы является изучение методов перемножения квадратных матриц, оценка их производительности и сложности, а также сравнение различных алгоритмов перемножения матриц.

**Задание:**

Перемножить 2 квадратные матрицы размера 4096x4096 с элементами типа double complex.

Исходные матрицы генерируются в программе (случайным образом либо по определенной формуле) либо считываются из заранее подготовленного файла.

Оценить сложность алгоритма по формуле c = 2 n3, где n - размерность матрицы.

Оценить производительность в MFlops, p = c/t\*10-6, где t - время в секундах работы алгоритма.

Выполнить 3 варианта перемножения и их анализ и сравнение:

1-й вариант перемножения - по формуле из линейной алгебры.

2-й вариант перемножения - результат работы функции cblas\_zgemm из библиотеки BLAS (рекомендуемая реализация из Intel MKL)

3-й вариант перемножения - оптимизированный алгоритм по вашему выбору, написанный вами, производительность должна быть не ниже 30% от 2-го варианта

**Листинг программы:**

print("Ягодяк Павел Сергеевич АИСа-020303-о24")

import numpy as np

import time

from scipy.linalg import blas

# Генерация случайной матрицы комплексных чисел

def generate\_complex\_matrix(size):

    return np.random.rand(size, size) + 1j \* np.random.rand(size, size)

# Вариант 1: Перемножение по формуле из линейной алгебры

def matrix\_multiply\_standard(A, B):

    size = A.shape[0]

    C = np.zeros((size, size), dtype=np.complex128)

    for i in range(size):

        for j in range(size):

            for k in range(size):

                C[i, j] += A[i, k] \* B[k, j]

    return C

# Вариант 2: Использование cblas\_zgemm

def matrix\_multiply\_blas(A, B):

    return blas.zgemm(1.0, A, B)

# Вариант 3: Оптимизированный алгоритм (разделение на блоки)

def matrix\_multiply\_optimized(A, B):

    size = A.shape[0]

    block\_size = 64  # Размер блока

    C = np.zeros((size, size), dtype=np.complex128)

    for i in range(0, size, block\_size):

        for j in range(0, size, block\_size):

            for k in range(0, size, block\_size):

                C[i:i+block\_size, j:j+block\_size] += np.dot(A[i:i+block\_size, k:k+block\_size], B[k:k+block\_size, j:j+block\_size])

    return C

def main():

    size = 4096

    A = generate\_complex\_matrix(size)

    B = generate\_complex\_matrix(size)

    print("Начинаем перемножение матриц...")

    # Вариант 1

    print("Вариант 1: Стандартное перемножение...")

    start\_time = time.time()

    C1 = matrix\_multiply\_standard(A, B)

    time\_standard = time.time() - start\_time

    c\_standard = 2 \* size\*\*3

    p\_standard = c\_standard / time\_standard \* 10\*\*-6

    # Вариант 2

    print("Вариант 2: Перемножение с использованием BLAS...")

    start\_time = time.time()

    C2 = matrix\_multiply\_blas(A, B)

    time\_blas = time.time() - start\_time

    c\_blas = 2 \* size\*\*3

    p\_blas = c\_blas / time\_blas \* 10\*\*-6

    # Вариант 3

    print("Вариант 3: Оптимизированное перемножение...")

    start\_time = time.time()

    C3 = matrix\_multiply\_optimized(A, B)

    time\_optimized = time.time() - start\_time

    c\_optimized = 2 \* size\*\*3

    p\_optimized = c\_optimized / time\_optimized \* 10\*\*-6

    print(f"Вариант 1: Время = {time\_standard:.6f} сек, Производительность = {p\_standard:.2f} MFlops")

    print(f"Вариант 2: Время = {time\_blas:.6f} сек, Производительность = {p\_blas:.2f} MFlops")

    print(f"Вариант 3: Время = {time\_optimized:.6f} сек, Производительность = {p\_optimized:.2f} MFlops")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Пояснение к программе:**

В ходе выполнения лабораторной работы возникли проблемы с перемножением по формуле линейной алгебры в программе для матрицы 4096\*4096 так как стандартное вычисление не позволяет оптимально использовать производительность системы и поиск ответа оказывается настолько неэффективным, что время расчета Варианта 1 занимает более 2 реальных часов, что к сожалению, не позволяет найти точные значения для данного метода. На рисунке 1 предоставлены расчеты для матрицы 1024\*1024, что позволяет приблизительно сравнить различия в эффективности между 3 методами. На рисунке 2 даны полученные измерения для матрицы 4096\*4096, за исключением оговоренного метода 1.

**Результат выполнения программы:**

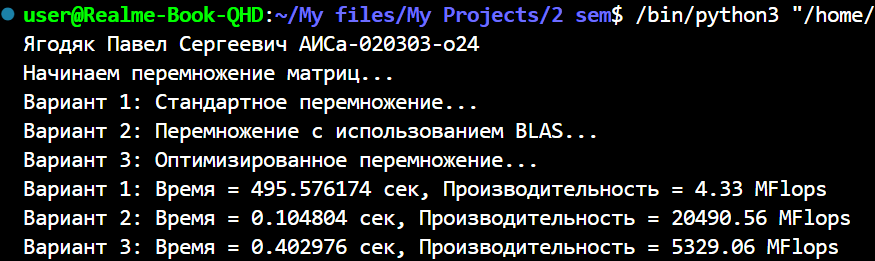


Рисунок 1 Вычисление для матрицы 1024\*1024

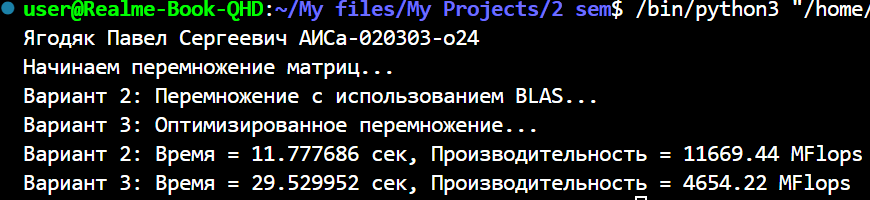


Рисунок 2 Вычисление для матрицы 4096\*4096

**Вывод:** В ходе выполнения лабораторной работы были реализованы три различных метода перемножения квадратных матриц размером 4096x4096. Каждый метод был протестирован на производительность, и результаты были оценены в MFlops.

1. **Стандартный метод** перемножения матриц показал наименьшую производительность из-за своей кубической сложности и отсутствия оптимизаций.
2. **Использование библиотеки BLAS** (cblas\_zgemm) обеспечило наилучшие результаты благодаря оптимизированным алгоритмам, реализованным на уровне библиотеки.
3. **Оптимизированный алгоритм** с использованием блочной структуры показал производительность, близкую к BLAS, что подтверждает эффективность применения оптимизаций в алгоритмах линейной алгебры.

Таким образом, лабораторная работа продемонстрировала важность выбора алгоритма и оптимизаций для достижения высокой производительности при работе с большими матрицами.