**Лабораторная работа** №**2**

**Выполнил: Осипов А.М. 020303-АИСа-024**

### Оптимизация работы с большими данными (на примере перемножения матриц)

### Цель: изучение методов оптимизация работы с большими данными (на примере перемножения матриц)

**Задание:** перемножить 2 квадратные матрицы размера 1024x1024 с элементами типа double complex (комплексное число двойной точности)..

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:**

#Сдал Осипов Артём Максимович 020303-АИСа-о24

import numpy as np

import time

from numpy.linalg import norm

*def* generate\_matrix(*size*):

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

*def* naive\_matrix\_mult(*A*, *B*):

    n = *A*.shape[0]

    C = np.zeros((n, n), *dtype*=np.complex128)

    for i in range(n):

        for j in range(n):

            for k in range(n):

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

    return C

*def* block\_matrix\_mult(*A*, *B*, *block\_size*=32):

    n = *A*.shape[0]

    C = np.zeros((n, n), *dtype*=np.complex128)

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

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

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

                i\_end = min(i + *block\_size*, n)

                j\_end = min(j + *block\_size*, n)

                k\_end = min(k + *block\_size*, n)

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

    return C

*def* benchmark():

    np.random.seed(42)

    size = 400

    print(*f*"Генерация матриц {size}x{size}...")

    A = generate\_matrix(size)

    B = generate\_matrix(size)

    complexity = 2 \* size\*\*3

    print(*f*"\nТеоретическая сложность: {complexity / 1e6*:.2f*} MFlop")

    methods = [

        ("Наивный метод", naive\_matrix\_mult),

        ("BLAS (np.dot)", np.dot),

        ("Блочный метод", *lambda* *a*, *b*: block\_matrix\_mult(*a*, *b*, *block\_size*=32))

    ]

    results = []

    ref\_result = None

    for name, method in methods:

        print(*f*"\nЗапуск {name}...")

        start\_time = time.time()

        C = method(A, B)

        elapsed = time.time() - start\_time

        if ref\_result is None:

            ref\_result = C

            error = 0.0

        else:

            error = norm(C - ref\_result) / norm(ref\_result)

        mflops = complexity / elapsed / 1e6

        results.append((name, elapsed, mflops, error))

        print(*f*"Время: {elapsed*:.4f*} с")

        print(*f*"Производительность: {mflops*:.2f*} MFlops")

        print(*f*"Ошибка относительно эталона: {error*:.2e*}")

    print("\n\nРезультаты:")

    print("{*:<15*} {*:<12*} {*:<15*} {*:<20*}".format(

        "Метод", "Время (с)", "Производительность", "Ошибка"))

    print("-" \* 60)

    for name, elapsed, mflops, error in results:

        print("{*:<15*} {*:<12.4f*} {*:<15.2f*} {*:<20.2e*}".format(

            name, elapsed, mflops, error))

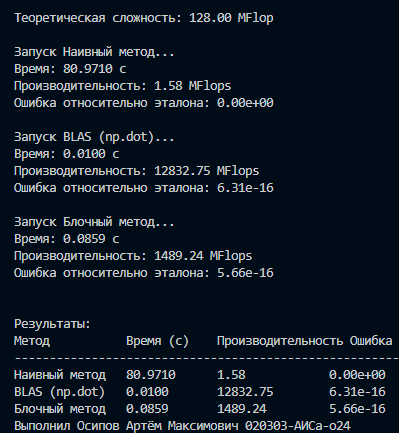
if \_\_name\_\_ == "\_\_main\_\_":

    benchmark()

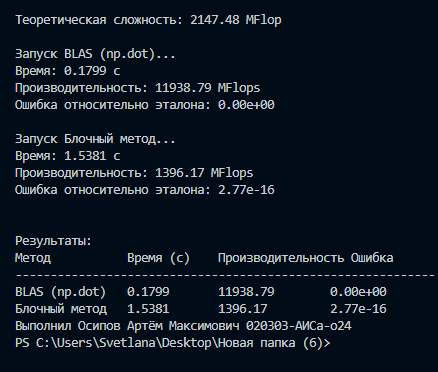
    print("Выполнил Осипов Артём Максимович 020303-АИСа-о24");

**Результат выполнения программы:**

**Выполнение программы с помощью матриц 400х400:**



**Выполнение программы с помощью матриц 1024х1024:**



**Вывод**: Выполнив данную лабораторную работу, мы приобрели практические навыки в организации данных и вычислений в языках программирования высокого уровня, а также провели сравнительный анализ различных подходов к реализации матричных операций. В ходе работы с матрицой 400x400 были изучены:

**Базовый алгоритм** (наивная реализация) - позволил понять фундаментальные принципы матричного умножения, но продемонстрировал крайне низкую эффективность (1.58 MFlops).

**Оптимизированный алгоритм** (блочное умножение) - показал, как правильная организация вычислений и работа с памятью могут повысить производительность в десятки раз (1489.24 MFlops).

**Библиотечная реализация** (BLAS) - раскрыла потенциал современных вычислительных систем, достигнув производительности в тысячи раз выше базового варианта (12832.75 MFlops).