Отчет по лабораторной работе номер 2

По теме: Исследование и сравнение алгоритмов умножения больших квадратных матриц с комплексными числами.

**Цель работы:**

Перемножить 2 квадратные матрицы размера 2048x2048 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

**Код программы:**

# Выполнил Шакула Дмитрий Андреевич 090301-ПОВа-о24

import numpy as np

import time

from scipy.linalg.blas import zgemm

N\_SMALL = 200

N\_LARGE = 2048

# Инициализация матриц с двойной точностью (complex128)

np.random.seed(42)

A\_large = np.random.rand(N\_LARGE, N\_LARGE).astype(np.complex128) + 1j \* np.random.rand(N\_LARGE, N\_LARGE).astype(np.complex128)

B\_large = np.random.rand(N\_LARGE, N\_LARGE).astype(np.complex128) + 1j \* np.random.rand(N\_LARGE, N\_LARGE).astype(np.complex128)

A\_small = np.random.rand(N\_SMALL, N\_SMALL).astype(np.complex128) + 1j \* np.random.rand(N\_SMALL, N\_SMALL).astype(np.complex128)

B\_small = np.random.rand(N\_SMALL, N\_SMALL).astype(np.complex128) + 1j \* np.random.rand(N\_SMALL, N\_SMALL).astype(np.complex128)

# Вычисление сложности

complexity\_small = 2 \* N\_SMALL\*\*3

complexity\_large = 2 \* N\_LARGE\*\*3

# Функция для измерения производительности

def measure\_performance(func, \*args, iterations=3):

elapsed\_times = []

result = None

for \_ in range(iterations):

start\_time = time.time()

result = func(\*args)

elapsed\_times.append(time.time() - start\_time)

elapsed\_time = min(elapsed\_times)

mflops = complexity\_small / (elapsed\_time \* 1e6) if args[0].shape[0] == N\_SMALL else complexity\_large / (elapsed\_time \* 1e6)

return result, elapsed\_time, mflops

# Стандартное перемножение (IJK)

def matrix\_multiply\_formula(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

# BLAS перемножение (cgemm)

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

return zgemm(alpha=1.0, a=A, b=B)

# Оптимизированное перемножение с использованием np.dot

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

return np.dot(A, B)

def main():

print("Работу выполнил: Шакула Дмитрий Андреевич 090301-ПОВа-o24")

print("\n1-Й ВАРИАНТ: УМНОЖЕНИЕ ПО ФОРМУЛЕ ИЗ ЛИНЕЙНОЙ АЛГЕБРЫ")

print(f"Размер матрицы: {N\_SMALL}x{N\_SMALL}")

C\_formula, time\_formula, mflops\_formula = measure\_performance(matrix\_multiply\_formula, A\_small, B\_small)

print(f"Время выполнения: {time\_formula:.2f} секунд")

print(f"Производительность: {mflops\_formula:.2f} MFLOPS")

print("\n\n2-Й ВАРИАНТ: ИСПОЛЬЗОВАНИЕ CBALS\_CGEMM ИЗ БИБЛИОТЕКИ BLAS")

print(f"Размер матрицы: {N\_LARGE}x{N\_LARGE}")

C\_blas, time\_blas, mflops\_blas = measure\_performance(matrix\_multiply\_blas, A\_large, B\_large)

print(f"Время выполнения: {time\_blas:.2f} секунд")

print(f"Производительность: {mflops\_blas:.2f} MFLOPS")

print("\n\n3-Й ВАРИАНТ: ОПТИМИЗИРОВАННЫЙ АЛГОРИТМ (np.dot)")

print(f"Размер матрицы: {N\_LARGE}x{N\_LARGE}")

C\_optimized, time\_optimized, mflops\_optimized = measure\_performance(

matrix\_multiply\_optimized, A\_large, B\_large

)

print(f"Время выполнения: {time\_optimized:.2f} секунд")

print(f"Производительность: {mflops\_optimized:.2f} MFLOPS")

print("\n\nСРАВНЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ:")

print(f"1-й вариант (размер {N\_SMALL}x{N\_SMALL}): {mflops\_formula:.2f} MFLOPS")

print(f"2-й вариант (размер {N\_LARGE}x{N\_LARGE}): {mflops\_blas:.2f} MFLOPS")

print(f"3-й вариант (размер {N\_LARGE}x{N\_LARGE}): {mflops\_optimized:.2f} MFLOPS")

performance\_ratio = mflops\_optimized / mflops\_blas

print(f"Отношение производительности (3-й / 2-й): {performance\_ratio:.2f}")

print(f"Условие выполнено (Оптимизированное >= 30% от BLAS): {'Да' if performance\_ratio >= 0.3 else 'Нет'}")

# Проверка корректности

norm12 = np.sqrt(np.sum(np.abs(C\_formula - matrix\_multiply\_blas(A\_small, B\_small))\*\*2))

norm23 = np.sqrt(np.sum(np.abs(C\_blas - C\_optimized)\*\*2))

print("\nПроверка корректности (нормы разностей):")

print(f"Norm(C1 - C2): {norm12:.2e}")

print(f"Norm(C2 - C3): {norm23:.2e}")

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

main()

**Результаты выполнения:**

