Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»

(МГТУ им. Н.Э. Баумана)

Факультет: Информатика и системы управления

Кафедра: Теоретическая информатика и компьютерные технологии

**Лабораторная работа №3**

**«Параллельная реализация решения системы линейных алгебраических**

**уравнений с помощью MPI»**

**по курсу: «Разработка параллельных и распределенных программ»**

Выполнил:

Студент группы ИУ9-52Б

Джабаров Р.А.

Проверил:

Царев А.С.

**1. Цель работы**

1. Написать программу, которая реализует итерационный алгоритм решения системы линейных алгебраических уравнений вида Ax=b в соответствии с выбранным вариантом. Здесь A – матрица размером N×N, x и b – векторы длины N. Тип элементов – double.
2. Программу распараллелить с помощью MPI с разрезанием матрицы A по строкам на близкие по размеру, возможно не одинаковые, части.

Соседние строки матрицы должны располагаться в одном или в

соседних MPI-процессах. Реализовать два варианта программы:

1: векторы x и b дублируются в каждом MPI-процессе,

2: векторы x и b разрезаются между MPI-процессами аналогично

матрице A. (только для сдающих после срока)

Уделить внимание тому, чтобы при запуске программы на различном

числе MPI-процессов решалась одна и та же задача (исходные данные

заполнялись одинаковым образом).

1. Замерить время работы двух вариантов программы при использовании различного числа процессорных ядер: 1,2, 4, 8, 16. Построить графики зависимости времени работы программы, ускорения и эффективности распараллеливания от числа используемых ядер. Исходные данные, параметры N и ε подобрать таким образом, чтобы решение задачи на одном ядре занимало не менее 30 секунд. Также параметр N разрешено подобрать таким образом, чтобы он нацело делился на на 1,2,4,8 и 16.
2. На основании полученных результатов сделать вывод о целесообразности использования одного или второго варианта программы.

**2. Реализация**

Код программы:

import numpy as np

from mpi4py import MPI

import pickle

import time

# mpiexec -n 4 /Library/Frameworks/Python.framework/Versions/3.11/bin/python3 lab2.py

comm = MPI.COMM\_WORLD

rank = comm.Get\_rank()

size = comm.Get\_size()

st = time.time()

comm.send(st, dest=size-1)

# Функция дл создания матриц

def generate\_matrix(a, b, n):

for i in range(n):

b.append(n + 1)

a\_elem = []

for j in range(n):

if (i == j):

a\_elem.append(2.0)

else:

a\_elem.append(1.0)

a.append(a\_elem)

def split\_array(original\_array, subarray\_size):

subarrays = []

for i in range(0, len(original\_array), subarray\_size):

subarray = original\_array[i:i+subarray\_size]

subarrays.append(subarray)

# Combine the subarrays into a flat array

flat\_array = [item for sublist in subarrays for item in sublist]

return flat\_array

def sopr(A, b, x, max\_iterations=1000, tolerance=1e-3):

r = b - np.dot(A, x)

p = r.copy()

for i in range(max\_iterations):

alpha = np.dot(r, r) / np.dot(p, np.dot(A, p))

x = x + alpha \* p

r\_next = r - alpha \* np.dot(A, p)

beta = np.dot(r\_next, r\_next) / np.dot(r, r)

p = r\_next + beta \* p

r = r\_next

if (np.linalg.norm(r) / np.linalg.norm(b)) < tolerance:

break

#print(x)

return x

n = 640

a = []

b = []

x = [0.01] \* n

generate\_matrix(a, b, n)

a = split\_array(a, n // size)

#print(a)

res = []

k = 0

for i in range(n // size \* rank, n // size \* (1+rank)):

k += 1

res.append(sopr(a[i], b, x))

#print(sopr(a[i], b, x), i)

#print(a[i])

x\_all = comm.gather(res, root=0)

#if rank == 0:

#print(x\_all, rank, size)

if rank == size - 1:

et = time.time()

st = comm.recv(source=0)

print(round((et - st), 3), "s")

# mpiexec -n 8 /Library/Frameworks/Python.framework/Versions/3.11/bin/python3 lab2.py

**4. Время работы:**

Время вычисления по 1 потокe: 37.714 s

Время вычисления по 2 потокам: 19.266 s

Время вычисления по 3 потокам: 13.304 s

Время вычисления по 4 потокам: 10.263 s

Время вычисления по 5 потокам: 9.314 s

Время вычисления по 6 потокам: 8.593 s

Время вычисления по 7 потокам: 7.705 s

Время вычисления по 8 потокам: 7.086 s