Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Лабораторная работа №2

по курсу «Разработка параллельных и распределенных программ»

«Решение СЛАУ итерационными методами»

Студент группы ИУ9-52Б Волохов А. В.

Преподаватель Царев А. С.

Москва 2024

1 Условие

Пусть есть система из N линейных алгебраических уравнений в виде $Ax = b$, где A — матрица коэффициентов уравнений размером $N \times N$, b — вектор правых частей размером N, x — искомый вектор решений размером N. Решение системы уравнений итерационным методом состоит в выполнении следующих шагов:

1. Задается x_0 — произвольное начальное приближение решения (вектор с произвольными начальными значениями).

2. Приближение многократно улучшается с использованием формулы вида $x_{n+1} = f(x_n)$, где функция f определяется используемым методом.

3. Процесс продолжается, пока не выполнится условие $g(x_n) < \varepsilon$, где функция g определяется используемым методом, а величина ε задает требуемую точность.

В данной задаче был использован метод простой итерации. В методе простой итерации преобразование решения на каждом шаге задается формулой:

$$x_{n+1} = x_n - \tau(Ax_n - b),$$

где τ — константа, параметр метода. В зависимости от значения параметра τ последовательность $\{x_n\}$ может сходиться к решению быстрее или медленнее, или вообще расходиться. В качестве подходящего значения τ можно выбрать $\frac{0.1}{N}$ или $\frac{-0.1}{N}$; знак зависит от задачи. Если с некоторым знаком решение начинает расходиться, то следует сменить его на противоположный.

Критерий завершения счёта задается следующим образом:

$$\frac{\|Ax_n - b\|_2}{\|b\|_2} < \varepsilon,$$

где $\|u\|_2 = \sqrt{\sum_{i=0}^{N-1} u_i^2}$ — евклидова норма вектора u, а ε — требуемая точность.

Подбирать значения параметра ε лучше всего исходя из задачи, начиная со значения 0.00001. Если, начиная с определенной итерации, значение это-

го параметра перестает меняться, значит, можно выбрать в качестве критерия завершения счёта именно это установившееся значение.

2 Практическая реализация

Код для стандартного алгоритма представлен в Листинге 1.

Листинг 1 - mpi parallel.py

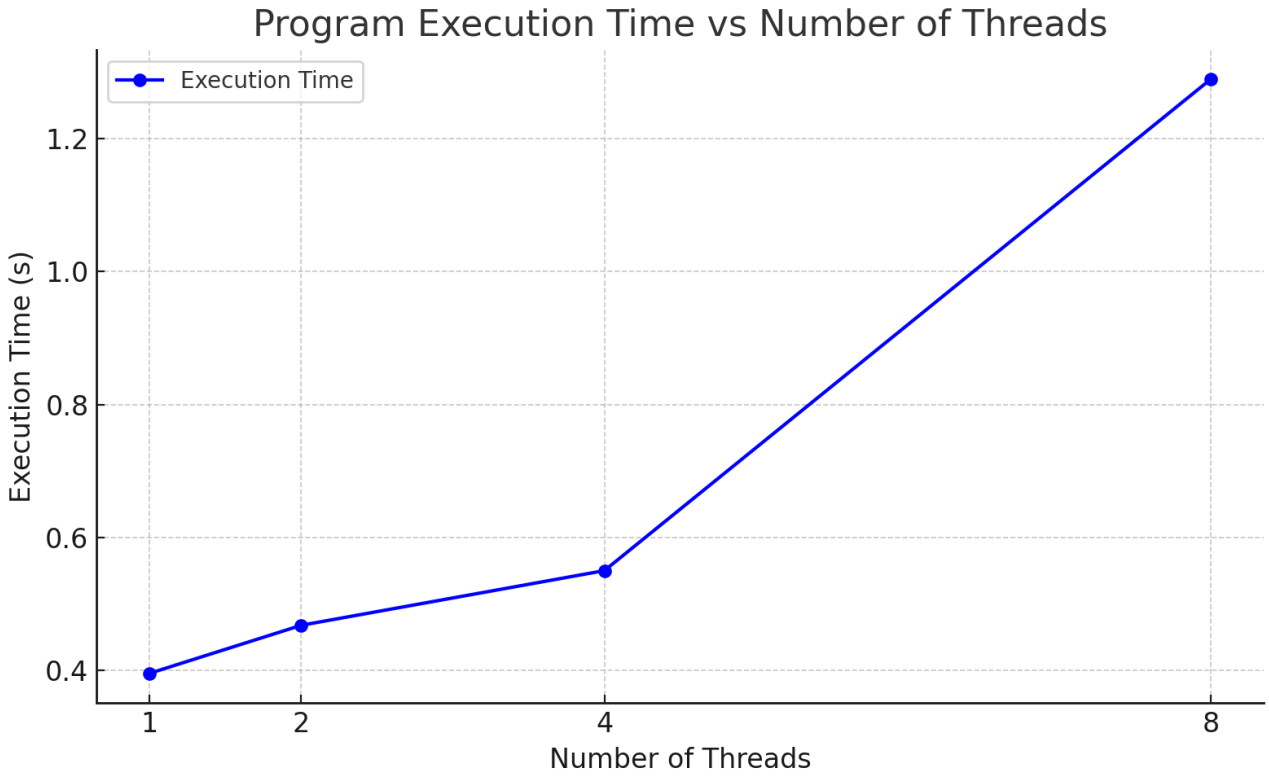
```
from mpi4py import MPI
import numpy as np
import time


def parallel_simple_iteration(A, b, tau, epsilon, max_iter=10000):
    comm = MPI.COMM_WORLD
    rank = comm.Get_rank()  #
    size = comm.Get_size()  #

    N = len(b)
    #

    local_N = N // size + (1 if rank < N % size else 0)
    local_start = sum(N // size + (1 if i < N % size else 0) for i in range(
        rank))
    local_end = local_start + local_N

    #

    local_A = A[local_start:local_end]
    local_b = b[local_start:local_end]
    local_x = np.zeros(local_N)

    #                                                    x

    global_x = np.zeros(N)

    for n in range(max_iter):
        #                                    global_x

        comm.Bcast(global_x, root=0)

        #                                          Ax
        local_Ax = np.dot(local_A, global_x)
```

```
        #
                              x
        local_x_new = local_x - tau * (local_Ax - local_b)

        #                                          x

        comm.Allgather(local_x_new, global_x)

        #
        norm_diff = np.linalg.norm(np.dot(A, global_x) - b) / np.linalg.norm(
            b)
        #

        norm_diff = comm.bcast(norm_diff, root=0)

        #
        if norm_diff < epsilon:
            if rank == 0:
                print(f"                                        {n + 1}
                                    ")
            break

        #                                                  x
        local_x = local_x_new

    return global_x if rank == 0 else None


def test_known_solution(N, tau, epsilon):
    A = np.full((N, N), 1.0)
    np.fill_diagonal(A, 2.0)
    b = np.full(N, N + 1)

    start_time = time.time()
    x_solution = parallel_simple_iteration(A, b, tau, epsilon)
    end_time = time.time()

    if MPI.COMM_WORLD.Get_rank() == 0:
        print("
                                :")
        print("                  :", x_solution)
        print(f"                                              : {end_time -
            start_time:.4f}                ")
```

```
def test_arbitrary_solution(N, tau, epsilon):
    A = np.full((N, N), 1.0)
    np.fill_diagonal(A, 2.0)
    u = np.sin(2 * np.pi * np.arange(N) / N)
    b = A @ u  #                    b

    start_time = time.time()
    x_solution = parallel_simple_iteration(A, b, tau, epsilon)
    end_time = time.time()

    if MPI.COMM_WORLD.Get_rank() == 0:
        print("
                          :")
        print("              :", x_solution)
        print(f"                                          : {end_time -
            start_time:.4f}             ")


if __name__ == "__main__":
    total_start_time = time.time()

    N = 256
    epsilon = 0.0001
    tau = 0.1 / N

    #
    test_known_solution(N, tau, epsilon)
    test_arbitrary_solution(N, tau, epsilon)

    total_end_time = time.time()
    total_elapsed_time = total_end_time - total_start_time

    if MPI.COMM_WORLD.Get_rank() == 0:
        print("
                        : {:.4f}             ".format(total_elapsed_time))
```

Количество потоков	Скорость работы программы (s)
1	0.3959
2	0.4681
4	0.5503
8	1.2892

Таблица 1: Зависимость скорости работы программы от количества потоков



3 Заключение

Эксперименты показали, что итерационный метод успешно справляется с решением систем уравнений, соответствующих тестовым задачам. Однако, с увеличением количества потоков время выполнения возрастает, что может быть связано с особенностями реализации параллельного алгоритма именно на языке Python и распределения данных между процессами.