|  | **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 Информатики и систем управления

КАФЕДРА Теоретической информатики и компьютерных технологий

**ЛАБОРАТОРНАЯ РАБОТА № 2**

**ПО КУРСУ:**

***«Разработка параллельных и распределенных программ»***

Студент *Виленский С.Д.*

Преподаватель *Царёв А.С.*

*Москва, 2023 г.*

**ОГЛАВЛЕНИЕ**

[1. Постановка задачи 3](#_heading=h.30j0zll)

[2. Практическая реализация 4](#_heading=h.1fob9te)

# 1. Постановка задачи

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 один вариант программы: векторы x и b дублируются в каждом MPI-процессе. Уделить внимание тому, чтобы при запуске программы на различном числе MPI-процессов решалась одна и та же задача (исходные данные заполнялись одинаковым образом).
3. Замерить время работы программы при использовании различного числа процессорных ядер: 1, 2, 4, 8, 16. Построить график 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. Практическая реализация

import numpy as np

from mpi4py import MPI

class SLESolver:

@classmethod

def \_\_next\_iter(

cls,

matrix\_a: np.matrix[float, float],

vector\_x: np.ndarray[None, float],

vector\_b: np.ndarray[None, float],

tau: float

) -> np.ndarray[None, float]:

return vector\_x - (mpi\_matrix\_dot(matrix\_a, vector\_x) - vector\_b) \* tau

@classmethod

def \_\_end\_measure(

cls,

matrix\_a: np.matrix[float, float],

vector\_x: np.ndarray[None, float],

vector\_b: np.ndarray[None, float],

) -> float:

return (np.linalg.norm(mpi\_matrix\_dot(matrix\_a, vector\_x) - vector\_b) /

np.linalg.norm(vector\_b))

@classmethod

def simple\_iteration(

cls,

matrix\_a: np.matrix[float, float],

vector\_b: np.ndarray[None, float],

epsilon: float = 1e-10,

) -> (np.ndarray[float, float] | None):

N = vector\_b.shape[0]

assert matrix\_a.shape == (N, N)

assert vector\_b.shape == (N, 1)

tau = .0001 / N

last\_iter\_measure = None

vector\_x = np.full((N, 1), 0)

for \_ in iter(int, 1):

actual\_iter\_measure = cls.\_\_end\_measure(

matrix\_a, vector\_x, vector\_b)

if actual\_iter\_measure < epsilon:

break

if (last\_iter\_measure is not None and

actual\_iter\_measure > last\_iter\_measure):

if tau < 0:

return None

tau \*= -1

vector\_x = cls.\_\_next\_iter(matrix\_a, vector\_x, vector\_b, tau)

last\_iter\_measure = actual\_iter\_measure

return vector\_x

def matrix\_dot(

matrix\_a: np.matrix[float, float],

vector\_x: np.ndarray[None, float],

) -> np.ndarray[None, float]:

return np.dot(matrix\_a, vector\_x)

def mpi\_matrix\_dot(

matrix\_a: np.matrix[float, float],

vector\_x: np.ndarray[None, float],

) -> (np.ndarray[None, float] | None):

N = vector\_x.shape[0]

assert matrix\_a.shape == (N, N)

assert vector\_x.shape == (N, 1)

if size <= 2:

return matrix\_dot(matrix\_a, vector\_x)

vector\_b = np.zeros((N, 1))

step = max(N // (size - 1), 1) + 1

reqs, reqr = [], []

for i in range(1, size):

index = (i - 1) \* step

reqs.append(comm.isend(

(matrix\_a[index: index + step], vector\_x), dest=i))

reqr.append(comm.irecv(source=i))

for i in range(size - 1):

reqs[i].wait()

index = i \* step

vector\_b[index: index + step] = reqr[i].wait()

return vector\_b

comm = MPI.COMM\_WORLD

worker = comm.Get\_rank()

size = comm.Get\_size()

if worker != 0:

while True:

\_matrix\_a, \_vector\_x = comm.recv(source=0)

if \_matrix\_a is None:

exit()

comm.send(matrix\_dot(\_matrix\_a, \_vector\_x), dest=0)

\_matrix\_a = np.matrix([

[9., 1., 3., -7., 9., -0., -9., 7.],

[-8., 10., -3., -0., -4., 1., 1., -3.],

[-8., 7., 7., -0., -0., 10., -1., -0.],

[7., -5., 5., 9., 1., 1., -8., -4.],

[3., -3., 8., -4., 9., 4., 4., 9.],

[9., 6., -9., 7., 3., 10., -3., -6.],

[4., -8., 8., 2., -2., -2., 9., 10.],

[-4., 10., 1., -7., -2., 8., -8., 5.],

])

\_vector\_x = np.array([

[-3.],

[-9.],

[-2.],

[-6.],

[-1.],

[4.],

[-4.],

[5.],

])

\_vector\_b = mpi\_matrix\_dot(\_matrix\_a, \_vector\_x)

\_res\_vector\_x = SLESolver.simple\_iteration(

\_matrix\_a,

\_vector\_b,

)

print(\_vector\_x)

print(\_res\_vector\_x)

for i in range(1, size):

comm.send((None, None), dest=i)

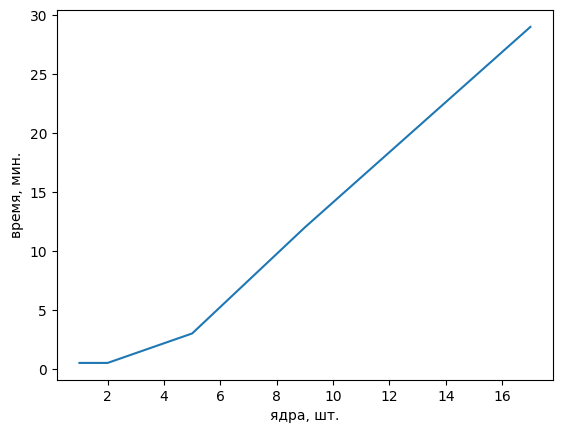


Рисунок 1 − График зависимости времени выполнения программ от количества ядер