|  | **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 № 3**

**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 OpenMP с разрезанием матрицы A по строкам на близкие по размеру, возможно не одинаковые, части. Соседние строки матрицы должны располагаться в одном или в соседних OpenMP-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

OpenMP.c

#include <Python.h>

#include <omp.h>

static PyObject\* openmp\_matrix\_dot(PyObject\* self, PyObject\* args) {

PyObject \*matrix\_obj, \*vector\_obj;

int num\_threads;

// *Получение аргументов из Python*

if (!PyArg\_ParseTuple(args, "OOi", &matrix\_obj, &vector\_obj, &num\_threads)) {

return NULL;

}

// *Проверка типов аргументов*

if (!PyList\_Check(matrix\_obj) || !PyList\_Check(vector\_obj)) {

PyErr\_SetString(PyExc\_TypeError, "Аргументы должны быть списками");

return NULL;

}

// *Получение размеров матрицы и вектора*

Py\_ssize\_t matrix\_rows = PyList\_Size(matrix\_obj);

Py\_ssize\_t matrix\_cols = PyList\_Size(PyList\_GetItem(matrix\_obj, 0));

Py\_ssize\_t vector\_size = PyList\_Size(vector\_obj);

// *Проверка размеров матрицы и вектора*

if (matrix\_cols != vector\_size) {

PyErr\_SetString(PyExc\_ValueError, "Размеры матрицы и вектора не совпадают");

return NULL;

}

// *Создание результирующего вектора*

PyObject\* result\_vector = PyList\_New(matrix\_rows);

#pragma omp parallel num\_threads(num\_threads)

{

#pragma omp for schedule(static) reduction(+:sum)

for (Py\_ssize\_t i = 0; i < matrix\_rows; ++i) {

double sum = 0.0;

// *Вычисление скалярного произведения строки матрицы и вектора*

for (Py\_ssize\_t j = 0; j < matrix\_cols; ++j) {

PyObject\* row = PyList\_GetItem(matrix\_obj, i);

PyObject\* element = PyList\_GetItem(row, j);

double value = PyFloat\_AsDouble(element);

PyObject\* vector\_element = PyList\_GetItem(vector\_obj, j);

double vector\_value = PyFloat\_AsDouble(vector\_element);

sum += value \* vector\_value;

}

// *Запись результата в результирующий вектор*

#pragma omp critical

{

PyObject\* result\_element = PyFloat\_FromDouble(sum);

PyList\_SetItem(result\_vector, i, result\_element);

}

}

}

return result\_vector;

}

static PyMethodDef methods[] = {

{"openmp\_matrix\_dot", openmp\_matrix\_dot, METH\_VARARGS, "Умножает матрицу на вектор"},

{NULL, NULL, 0, NULL}

};

static struct PyModuleDef module = {

PyModuleDef\_HEAD\_INIT,

"OpenMP\_funcs",

NULL,

-1,

methods

};

PyMODINIT\_FUNC PyInit\_OpenMP\_funcs(void) {

return PyModule\_Create(&module);

}

setup.py

from setuptools import setup, Extension

module = Extension(

'OpenMP\_funcs',

sources=['OpenMP\_funcs.c'],

extra\_compile\_args=['-fopenmp', '-lopenmp'],

extra\_link\_args=['-fopenmp', '-lopenmp'],

)

setup(

name='OpenMP\_funcs',

version='1.0',

ext\_modules=[module],

)

sle\_solver\_var1.py

import numpy as np

import time

import OpenMP\_funcs

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 - (openmp\_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(openmp\_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 = .01 / 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 openmp\_matrix\_dot(

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

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

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

matrix\_a = matrix\_a.tolist()

vector\_x = list(vector\_x)

res\_matrix = OpenMP\_funcs.openmp\_matrix\_dot(matrix\_a, vector\_x, num\_threads)

return np.matrix(res\_matrix).transpose()

num\_threads = 2

\_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 = openmp\_matrix\_dot(\_matrix\_a, \_vector\_x)

start = time.time()

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

\_matrix\_a,

\_vector\_b,

)

delta\_time = time.time() - start

print(\_vector\_x)

print(\_res\_vector\_x)

print(f"Execute time: {delta\_time}")

