|  |  |
| --- | --- |
|  | **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

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3**

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

Студент: Яровикова А. С.

Группа: ИУ9-41Б

Преподаватель: Царев А.С..

*.*

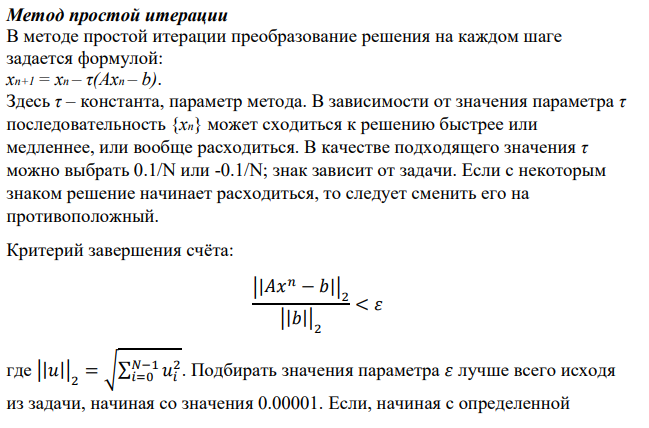
*2022 г.*

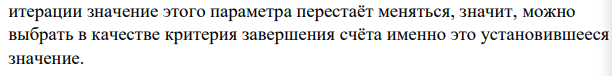
# Условие задачи

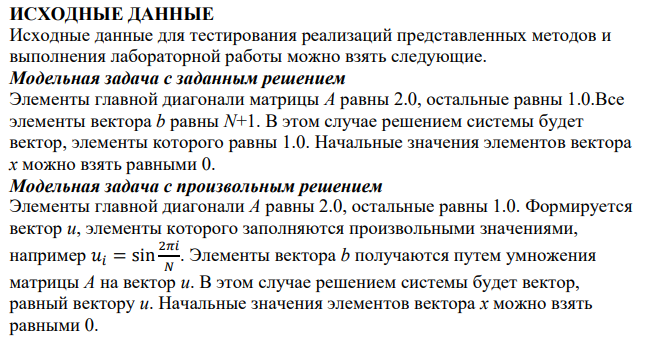
Лабораторная работа №2: Параллельная реализация решения системы линейных алгебраических уравнений с помощью MPI.

Пусть есть система из N линейных алгебраических уравнений в виде Ax=b, где А – матрица коэффициентов уравнений размером N×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 x0 – произвольное начальное приближение решения (вектор с произвольными начальными значениями).
2. Приближение многократно улучшается с использованием формулы вида xn+1 = f(xn), где функция f определяется используемым методом.
3. Процесс продолжается, пока не выполнится условие g(xn) < ε, где функция g определяется используемым методом, а величина ε задает требуемую точность.







**ЗАДАНИЕ К ЛАБОРАТОРНОЙ РАБОТЕ**

Написать программу, которая реализует решение задачи лабораторной работы №2 но уже с помощью OPENMP

# 2. Характеристики устройства и входных данных

* Процессор Intel(R) Core(TM) i5-4310U CPU @ 2.00GHz, 2601 МГц, ядер: 2, логических процессоров: 4. 8 ГБ ОЗУ;

## Язык C++ 14. Платформа: x64. Microsoft MPI v10.0;

* Размер матрицы: 512x512, тип элементов: double;
* Значение параметра ℰ = 0.0000001;
* Начальные данные соответствуют модельной задачи с заданным решением.

**3. Листинг текста программы**

**Итерационный алгоритм:**

#include <iostream>

#include <math.h>

#include <ctime>

#include <cmath>

using namespace std;

#define N\_SIZE 512

#define eps 0.0000001

#define t 0.1/N\_SIZE //τ – константа, параметр метода

double A[N\_SIZE][N\_SIZE];

double B[N\_SIZE];

template<int rows, int cols>

void FillMatrix(double(&matrix)[rows][cols]) {

for (int i = 0; i < cols; i++) {

for (int j = 0; j < rows; j++) {

if (i == j) {

matrix[i][j] = 2.0;

}

else {

matrix[i][j] = 1.0;

}

}

}

}

template<int rows, int cols>

void PrintMatrix(double(&matrix)[rows][cols]) {

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) {

printf("%d ", matrix[i][j]);

}

printf("\n");

}

}

void PrintX(double\* x) {

cout << "Answer: " << x[0];

for (int i = 1; i < N\_SIZE; i++) {

cout << ", " << x[i];

}

cout << endl;

}

double getNorm(double\* v) {

double s = 0;

for (int i = 0; i < N\_SIZE; i++) {

s += pow(v[i], 2);

}

s = sqrt(s);

return s;

}

bool isAnswerCorrect(double\* x) {

PrintX(x);

for (int i = 0; i < N\_SIZE; i++) {

if ( abs(x[i]- 1.0) < 0.00001 ) {

continue;

}

else {

cout << "here " << i << " " << x[i] << endl;

return false;

}

}

return true;

}

int main(int argc, char\* argv[]) {

FillMatrix(A);

for (int i = 0; i < N\_SIZE; i++) {

B[i] = N\_SIZE + 1;

}

double\* x = new double[N\_SIZE];

for (int i = 0; i < N\_SIZE; i++) {

x[i] = 0;

}

clock\_t start = clock();

int k = 0;

while (true) {

cout << "========= iteration = " << k << " =========" << endl;

double res[N\_SIZE];

for (int i = 0; i < N\_SIZE; i++) {

res[i] = 0;

}

for (int i = 0; i < N\_SIZE; i++) {

for (int j = 0; j < N\_SIZE; j++) {

res[i] += A[i][j] \* x[j];

}

}

for (int i = 0; i < N\_SIZE; i++) {

res[i] -= B[i];

}

PrintX(res);

double sum1 = 0;

sum1 = getNorm(res);

//cout << "numerator " << sum1 << endl;

double sum2 = 0;

sum2 = getNorm(B);

//cout << "denominator " << sum2 << endl;

//cout << "the condition " << sum1 / sum2 << endl;

if (sum1 / sum2 < eps) {

break;

}

else {

for (int i = 0; i < N\_SIZE; i++) {

x[i] -= t \* res[i];

}

}

cout << "probably";

PrintX(x);

cout << endl;

k++;

}

clock\_t end = clock();

double duration = ((double)(end - start)) / (double)CLOCKS\_PER\_SEC;

if (isAnswerCorrect(x)) {

cout << "right ans!" << endl;

PrintX(x);

}

else {

cout << "wrong ans!" << endl;

PrintX(x);

}

cout << endl;

cout << "Time: " << duration << endl;

delete[] x;

return 0;

}

**Итерационный алгоритм с использованием OPENMP**

#include <omp.h>

#include <ctime>

#include <stdio.h>

#include <iostream>

#include <math.h>

using namespace std;

const long long N = 512;

const double eps = 0.0000001;

const double t = 0.1 / N;

const int NUM\_THREADS = 8;

void fillMatrix(double\* matrix) {

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

if (i == j) matrix[i \* N + j] = 2.0;

else matrix[i \* N + j] = 1.0;

}

}

}

void fillB(double\* B) {

for (int i = 0; i < N; i++) {

B[i] = N + 1;

}

}

void fillX(double\* x) {

for (int i = 0; i < N; i++) {

x[i] = 0;

}

}

void printMatrix(double\* matrix) {

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

printf("%f ", matrix[i \* N + j]);

}

printf("\n");

}

}

void printVector(double\* v) {

cout << v[0];

for (int i = 1; i < N; i++) {

cout << ", " << v[i];

}

cout << endl;

}

void printX(double\* x, int size) {

cout << "Answer: " << x[0];

for (int i = 1; i < size; i++) {

cout << ", " << x[i];

}

cout << endl;

}

bool isAnswerCorrect(double\* x) {

for (int i = 0; i < N; i++) {

if (abs(x[i] - 1.0) < 0.00001) {

continue;

}

else {

cout << "here " << i << " " << x[i] << endl;

return false;

}

}

return true;

}

double getNorm(double\* v) {

double s = 0;

int i;

omp\_set\_num\_threads(NUM\_THREADS);

#pragma omp parallel for shared(v) private(i) reduction(+:s)

for (int i = 0; i < N; i++) {

s += pow(v[i], 2);

}

cout<< "I am thread " << omp\_get\_num\_threads() << endl;

s = sqrt(s);

return s;

}

double\* mulMatrixByVector(double\* matrix, double\* x) {

double\* res = new double[N];

int i = 0;

omp\_set\_num\_threads(NUM\_THREADS);

#pragma omp parallel for shared(matrix, x, res) private(i)

for (i = 0; i < N; i++) {

res[i] = 0;

for (int j = 0; j < N; j++) {

res[i] += matrix[i \* N + j] \* x[j];

}

}

cout << "I am thread " << omp\_get\_num\_threads() << endl;

return res;

}

void procFunc( double\* A, double\* B, double\* x, double\* y) {

double condition = eps;

int k = 0;

while (true) {

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_iteration = " << k << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

k++;

y = mulMatrixByVector(A, x);

int i;

omp\_set\_num\_threads(NUM\_THREADS);

#pragma omp parallel shared(y, B) private(i)

{

cout << "I am thread " << omp\_get\_num\_threads() << endl;

#pragma omp for

for (i = 0; i < N; i++) {

y[i] -= B[i];

}

}

condition = getNorm(y) / getNorm(B);

if (condition < eps) {

break;

}

omp\_set\_num\_threads(NUM\_THREADS);

#pragma omp parallel shared(y) private(i)

{

cout << "I am thread " << omp\_get\_num\_threads() << endl;

#pragma omp for

for (i = 0; i < N; i++) {

y[i] \*= t;

}

}

omp\_set\_num\_threads(NUM\_THREADS);

#pragma omp parallel shared(x, y) private(i)

{

cout << "I am thread " << omp\_get\_num\_threads() << endl;

#pragma omp for

for (i = 0; i < N; i++) {

x[i] -= y[i];

}

}

printf("condition is now: %lf\n", condition);

}

}

int main(int argc, char\* argv[]) {

#ifdef \_OPENMP

printf("OpenMP is supported!\n");

#endif // \_OPENMP

double\* A = new double[N \* N];

double\* B = new double[N];

double\* x = new double[N];

double\* y = new double[N];

fillX(x);

printf("Generating matrix A, size %lldx%lld\n", N, N);

fillMatrix(A);

//printMatrix(A);

printf("Generating vector B, size %lld\n", N);

fillB(B);

//printVector(B);

fillX(y);

printf("\n----------------Starting Calculations----------------\n");

double start\_time = omp\_get\_wtime();

procFunc(A, B, x, y);

double end\_time = omp\_get\_wtime();

printf("\n----------------Ending Calculations----------------\n");

cout << "duration is " << end\_time - start\_time << " seconds for " << NUM\_THREADS << " threads\nfor matrix "

<< N << "x" << N << endl;

printX(x, N);

if (!isAnswerCorrect(x)) {

printf("\nERROR - THE WRONG ANSWER\n");

}

else {

printf("\n!!THE RIGHT ANSWER!!\n");

}

delete[] A;

delete[] B;

delete[] x;

delete[] y;

return 0;

}

**4. Время работы**

|  |  |
| --- | --- |
| Количество потоков | Время выполнения (сек.) |
| 1 | 283 |
| 2 | 5.21 |
| 4 | 7.51 |
| 8 | 13.59 |
| 16 | 24.03 |

