###### МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

###### ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

###### НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

###### Факультет информационных технологий

**Кафедра параллельных вычислений**

ОТЧЕТ

О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

«Параллельная реализация решения системы линейных алгебраических уравнений с помощью OpenMP»

студентки 2 курса, … группы

**…**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

…

Новосибирск 2021

**Содержание**

[Задание 3](#_Toc167896547)

[Описание алгоритма 4](#_Toc167896548)

[Графики зависимости времени работы программ от числа используемых ядер 5](#_Toc167896549)

[Вывод 7](#_Toc167896551)

[Листинг программы. Вариант 1 8](#_Toc167896552)

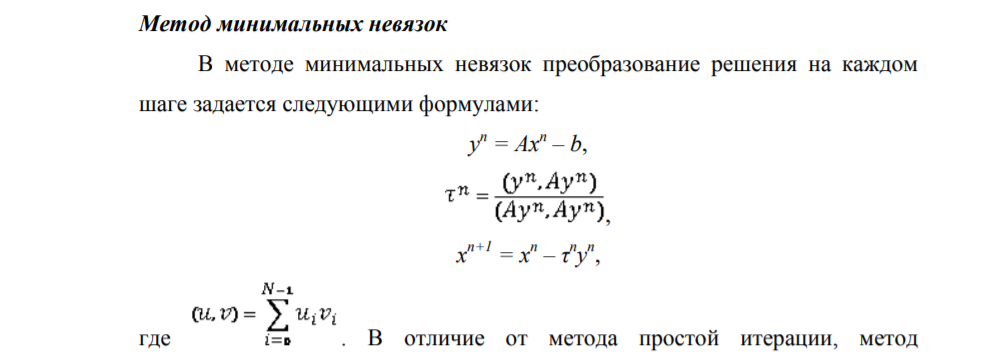
[Листинг программы. Вариант 2 10](#_Toc167896553)

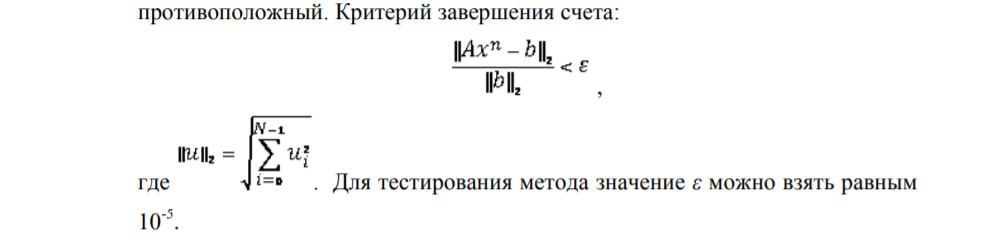
# Задание

* Последовательную программу из лабораторной работы 1, реализующую итерационный алгоритм решения системы линейных алгебраических уравнений вида Ax=b, распараллелить с помощью OpenMP. Реализовать два варианта программы:
  + Вариант 1: для каждого распараллеливаемого цикла создается отдельная параллельная секция #pragma omp parallel for,
  + Вариант 2: создается одна параллельная секция #pragma omp parallel, охватывающая весь итерационный алгоритм.
* Замерить время работы двух вариантов программы при использовании различного числа процессорных ядер: от 1 до числа доступных в узле. Построить графики зависимости времени работы программы, ускорения и эффективности распараллеливания от числа используемых ядер. Исходные данные и параметры задачи подобрать таким образом, чтобы решение задачи на одном ядре занимало не менее 30 секунд.
* На основании полученных результатов сделать вывод о целесообразности использования первого или второго варианта программы.

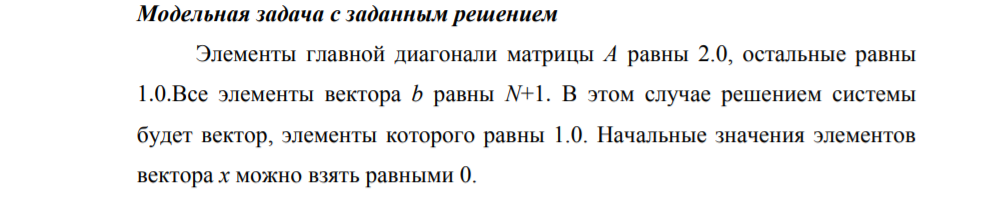
# Описание алгоритма

Программа на языке C++ реализует метод минимальных невязок решения системы линейных алгебраических уравнений вида Ax=b, где A – матрица размером N×N, x и b – векторы длины N.





Для тестирования программы были взяты следующие входные данные:



# Графики зависимости времени работы программ от числа используемых ядер

**Ускорение**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 ядро | 2 ядра | 4 ядра | 8 ядер | 16 ядер |
| Вариант 1 | 1 | 1,96 | 2,8 | 3,0 | 2,7 |
| Вариант 2 | 1 | 1,89 | 2,74 | 2,95 | 2,54 |

# 

# Вывод

В ходе лабораторной работы я ознакомилась с моделью программирования и основными командами OpenMP. Я применила эти знания на практике, реализовав алгоритм нахождения решения системы алгебраических уравнений. По результатам выполненной работы могу сказать:

1. Время работы программы зависит от размера матрицы, числа итераций и количества потоков;
2. Программа не ускоряется пропорционально увеличению числу потоков;
3. Время работы программы на 16и и на 4х процессах сравнимо.

# Листинг программы. Вариант 1

#include <mpi.h>

#include <iostream>

#include <math.h>

#include <cstdlib>

#include <ctime>

using namespace std;

double skalar(double\* X, double\* Y, const long N) {

double R = 0;

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

R += X[i] \* Y[i];

return R;

}

double\* mul(double\* A, double\* fullX, const int size, const long M) {

double\* Ax = new double[M];

for (long i = 0; i < M; ++i) {

Ax[i] = 0;

for (long j = 0; j < size \* M; ++j)

Ax[i] += A[i \* size \* M + j] \* fullX[j];

}

double\* fullAx = new double[size \* M];

MPI\_Allgather(Ax, M, MPI\_DOUBLE, fullAx, M, MPI\_DOUBLE, MPI\_COMM\_WORLD);

delete[] Ax;

return fullAx;

}

bool proverka(double\* Ax, double\* B, const long N) {

double\* Ax\_b = new double[N];

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

Ax\_b[i] = Ax[i] - B[i];

double u = 0, ub = 0;

u = skalar(Ax\_b, Ax\_b, N);

ub = skalar(B, B, N);

delete[] Ax\_b;

if ((sqrt(u) / sqrt(ub)) < 0.00000001)

return 0;

else

return 1;

}

double\* find\_x(double\* A, double\* B, double\* X, const int size, const long M) {

for (; ;) {

MPI\_Barrier(MPI\_COMM\_WORLD);

double\* Y = new double[M \* size];

double\* Ax = mul(A, X, size, M);

if (proverka(Ax, B, size \* M) == 0)

return X;

for (long j = 0; j < M \* size; ++j)

Y[j] = Ax[j] - B[j];

double\* Ay = mul(A, Y, size, M);

double t = 0.1 \* (skalar(Y, Ay, M \* size) / skalar(Ay, Ay, M \* size));

for (long j = 0; j < M \* size; ++j)

X[j] = X[j] - t \* Y[j];

delete[] Y;

delete[] Ax;

delete[] Ay;

}

return X;

}

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

int size, rank;

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

if (rank == 0)

cout << "Number of processes: " << size << endl;

const long N = 4000, M = N / size;

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

for (long i = 0; i < M; ++i) {

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

if (i + rank \* M == j)

A[i \* N + j] = 2;

else

A[i \* N + j] = 1;

}

}

double\* X = new double[N];

double\* B = new double[N];

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

X[i] = 0;

B[i] = N + 1;

}

struct timespec start, end;

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &start);

X = find\_x(A, B, X, size, M);

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &end);

double time = end.tv\_sec-start.tv\_sec + 0.000000001\*(end.tv\_nsec-start.tv\_nsec);

double fulltime = 0;

MPI\_Allreduce(&time, &fulltime, 1, MPI\_DOUBLE, MPI\_MAX, MPI\_COMM\_WORLD);

//double resalt = 0;

if (rank == 0) {

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

//resalt += X[i];

cout << "Time taken: " << fulltime << " sec." << endl;

//cout << "Resalt = " << resalt << endl;

}

delete[] A;

delete[] B;

delete[] X;

MPI\_Finalize();

return 0;

}

# Листинг программы. Вариант 2

#include <mpi.h>

#include <iostream>

#include <math.h>

#include <cstdlib>

#include <ctime>

using namespace std;

double skalar(double\* X, double\* Y, const long M) {

double R = 0, fullR = 0;

for (long i = 0; i < M; ++i)

R += X[i] \* Y[i];

MPI\_Allreduce(&R, &fullR, 1, MPI\_DOUBLE, MPI\_SUM, MPI\_COMM\_WORLD);

return fullR;

}

double\* mul(double\* A, double\* X, const int size, const long M) {

double\* fullX = new double[size \* M];

MPI\_Allgather(X, M, MPI\_DOUBLE, fullX, M, MPI\_DOUBLE, MPI\_COMM\_WORLD);

double\* Ax = new double[M];

for (long i = 0; i < M; ++i) {

Ax[i] = 0;

for (long j = 0; j < size \* M; ++j)

Ax[i] += A[i \* size \* M + j] \* fullX[j];

}

return Ax;

}

bool proverka(double\* Ax, double\* B, const long M) {

double\* Ax\_b = new double[M];

for (long i = 0; i < M; ++i)

Ax\_b[i] = Ax[i] - B[i];

double u = 0, ub = 0;

u = skalar(Ax\_b, Ax\_b, M);

ub = skalar(B, B, M);

delete[] Ax\_b;

if ((sqrt(u) / sqrt(ub)) < 0. 00000001)

return 0;

else

return 1;

}

double\* find\_x(double\* A, double\* B, double\* X, const int size, const long M) {

for (; ;) {

MPI\_Barrier(MPI\_COMM\_WORLD);

double\* Y = new double[M];

double\* Ax = mul(A, X, size, M);

if (proverka(Ax, B, M) == 0)

return X;

for (long j = 0; j < M; ++j)

Y[j] = Ax[j] - B[j];

double\* Ay = mul(A, Y, size, M);

double t = 0.1 \* (skalar(Y, Ay, M) / skalar(Ay, Ay, M));

for (long j = 0; j < M; ++j)

X[j] = X[j] - t \* Y[j];

delete[] Y;

delete[] Ax;

delete[] Ay;

}

return X;

}

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

int size, rank;

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

if (rank == 0)

cout << "Number of processes: " << size << endl;

const long N = 4000, M = N / size;

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

for (long i = 0; i < M; ++i) {

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

if (i + rank \* M == j)

A[i \* N + j] = 2;

else

A[i \* N + j] = 1;

}

}

double\* X = new double[M];

double\* U = new double[M];

double\* B = new double[M];

for ( long i = 0; i < M; ++i) {

X[i] = 0;

B[i] = N + 1;

}

struct timespec start, end;

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &start);

X = find\_x(A, B, X, size, M);

double\* fullX = new double[N];

MPI\_Allgather(X, M, MPI\_DOUBLE, fullX, M, MPI\_DOUBLE, MPI\_COMM\_WORLD);

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &end);

double time = end.tv\_sec-start.tv\_sec + 0.000000001\*(end.tv\_nsec-start.tv\_nsec);

double fulltime = 0;

MPI\_Allreduce(&time, &fulltime, 1, MPI\_DOUBLE, MPI\_MAX, MPI\_COMM\_WORLD);

// double resalt = 0;

if (rank == 0) {

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

//resalt += fullX[i];

cout << "Time taken: " << fulltime << " sec." << endl;

//cout << "Resalt = " << resalt << endl;

}

delete[] A;

delete[] B;

delete[] X;

delete[] fullX;

MPI\_Finalize();

return 0;

}