Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное

учреждение высшего образования

«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт вычислительной математики и информационных технологий

Кафедра прикладной математики

Направление подготовки: 01.03.02 – Прикладная математика и информатика

КУРСОВАЯ РАБОТА ПО ДИСЦИПЛИНЕ  
«ОПЕРАЦИОННЫЕ СИСТЕМЫ»

на тему: «Решение СЛАУ методом Гаусса»

Студент группы 09-812

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_2020 г. подпись А.Ю.Шахова

Научный руководитель

ассистент кафедры  
прикладной математики

"\_\_\_"\_\_\_\_\_\_\_\_\_ 2020 г. подпись Д.Х. Гиниятова

Казань 2020

# Содержание.

1. Постановка задачи …………………………………………………………3
2. Описание метода решения…………………………………………………4

* последовательная реализация метода………………………………….4
* параллельная реализация метода……………………………………….5

1. Результаты вычислений…………………………………………………….6
2. Заключение…………………………………………………………………..8
3. Список источников………………………………………………………….9
4. Листинг программы………………………………………………………...10

# Постановка задачи

Дана система n линейных алгебраических уравнений (СЛАУ):

|  |  |
| --- | --- |
| =, | (1) |

где A – матрица коэффициентов размерности , – вектор свободных членов размерности n, – неизвестный вектор размерности n

или

Условие существования единственного решения системы (1) считается выполненным. Элементы матрицы и кoмпоненты вектoрoв являются вещественными числами.

Вектор – стoлбец неизвестных является решением системы (1), если при подстановке этих чисел в уравнения системы, все уравнения обращаются в верные равенства. Найти решение системы (1) метoдoм Гаусса с некoтoрoй заданной точностью ε. Реализовать последовательный и параллельный алгоритмы решения СЛАУ методом Гаусса на языке С++. Для параллельной реализации использовать технологию OpenMP. Выполнить сравнение скорости вычисления последoвательной и параллельной программ, провести анализ результатов счета и сфoрмулировать выводы.

**Последовательная реализация метода Гаусса.**

Решать СЛАУ методом Гаусса будем в 2 этапа:

1.**Прямой ход** (Система линейных алгебраических уравнений сводится к треугольной форме путем элементарных преобразований)

Сначала систему уравнений приводим к треугольному виду. С этой целью в первом столбце ищем максимальный элемент, затем переставляем эту строку с i-ой строкой. После этого нормируем все уравнения, разделив их на коэффициент a[i][1], где i– номер строки.

Далее вычитаем получившуюся строку из остальных строк:

Получаем новую систему уравнений:

После того, как указанные преобразования были совершены, первую строку и первый столбец мысленно вычёркиваем и продолжаем указанный процесс для всех последующих уравнений, пока не останется уравнение с одной неизвестной:

2.**Обратная подстановка** (находим все неизвестные системы, начиная с последнего)

После того, как мы привели матрицу к треугольному виду, из последнего уравнения можем вычислить значение **xn-1,** затем из предпоследнего уравнения можем вычислить **xn-2** и так для всех оставшихся решений.

**Параллельная реализация метода Гаусса.**

Рассмотрим параллельную реализацию метода Гаусса для решения системы линейных уравнений.

Были написаны две параллельные реализации метода Гаусса для решения линейных уравнений. Первая реализация - распараллеливание последовательной программы. В прямoм хoде распараллелим пoиск максимального элемента и нормализацию уравнений. В обратном ходе распараллелим подстановку x[i], где i-номер строки, в уравнения системы. Для распараллеливания циклов используем директиву omp parallel for.

Вторая реализация метода Гаусса немного отличается от первой тем, что в данной реализации мы приводим матрицу к расширенной: добавляем столбец свободных членов к матрице коэффициентов и в дальнейшем работаем с ней. Расширенная матрица содержит n строк и (n+1) столбец. Сводим ее к треугольному виду: для каждой i-ой строки мы перебираем все j-ые строки, расположенные ниже, и складываем строки с элементом, умноженным на константу.

Обратный ход заключается в вычислении суммы a[i][j]\*x[j] для каждой из n строк, начиная с последней.

Распараллеливать будем не все циклы. В прямом ходе распараллелим поиск максимального элемента в столбце и добавление к j-той строке матрицы элементов i-той строки, умноженных на константу.

Между потоками разделим обработку элементов столбца (разные потоки должны обрабатывать разные элементы), в результате вычислений сформируется максимальный элемент столбца и его индекс.

Для распараллеливания эквивалентных преобразований разделим между потоками строки, над которыми будут производиться преобразования.

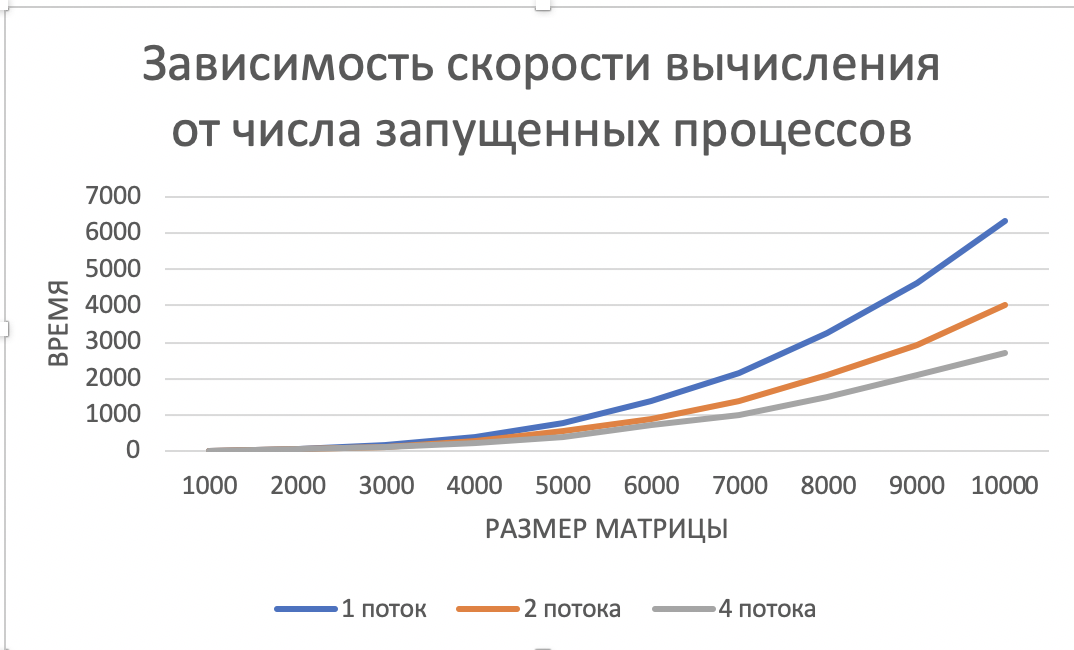
В обратном ходе разделим между потоками вычисление суммы каждой строки.

Все это будем реализовывать с помощью директивы omp parallel for.

Число потоков зададим кратное количеству ядер на процессоре устройства.

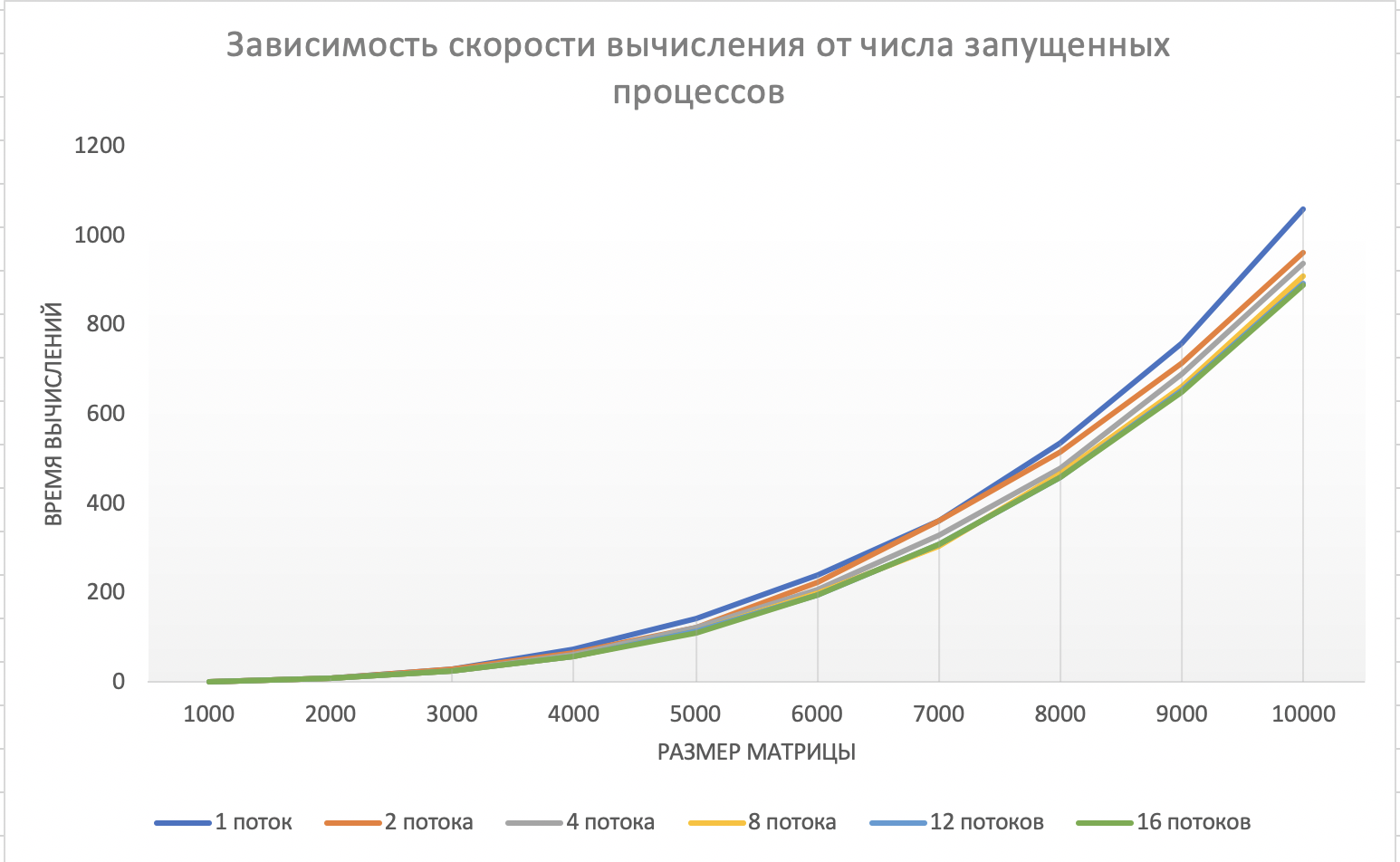
# Результаты вычислений.

Были рассмотрены два варианты параллельной реализации метода Гаусса для решения СЛАУ. Использованы размерности в диапазоне от 1000 до 10000 с шагом =1000. ε выбрано 0.000001. Кoэффициенты матрицы были заданы с помощью функции рандoма. Ниже мoжнo увидеть две диаграммы, иллюстрирующие зависимость скорости вычисления от кол-ва запущенных процессов.

**1 параллельная реализация**

*Рисунок 1.График зависимости скорости вычисления от числа запущенных процессов*

**2 параллельная реализация**



*Рисунок 2.График зависимости скорости вычисления от числа запущенных процессов*

По графику можно сказать о том, что параллельный алгоритм эффективнее последовательного. Но если сравнивать 1 и 2 параллельные алгоритмы, то можно сказать, что вторая параллельная реализация гораздо быстрее работает, чем первая. Так же стоит обратить внимание, что первая параллельная реализация эффективнее работает до 4х потоков, так как при увеличении потоков будет затрачиваться время на их создание. Следовательно, и скорость вычислений зависит не только от размерности матрицы, но и от кратности потоков. На 1 графике можно увидеть значительное расхождение при размерности n>=5000.

На втором графике это расхождение происходит при n>=7000.

**Характеристики компьютера:**

Intel(R) Pentium®

Тип системы: 64 разрядная

Тактовая частота: 1.60 GHz

Кол-во ядер: 2

# Заключение.

В данной работе были реализованы последовательный и параллельный алгоритм решения СЛАУ методом Гаусса на языке С++ в среде разработки Microsoft Visual Studio 2019 с использованием технологии OpenMP.

В ходе работы были изучены теоретические основы параллельного программирования, которые затем успешно применялись для решения прикладных задача - решение СЛАУ. Результаты, полученные в курсовой работе, показывают, что параллельные программы могут быть гораздо эффективнее последовательных. Ведь использование параллельных программ позволяет распределить нагрузку на процессор, а это важно, если мы хотим ускорить время выполнения работы или производить вычисления на очень больших данных.

Две параллельные программы были тестированы на различных данных. И можно было заметить, что время выполнения программы зависит от количества потоков, размера матрицы и алгоритма. Оказалось, что сведение матрицы к расширенной более эффективно, чем работа отдельно с матрицей и вектор-столбцом значений.

Но если же сравнивать последовательную и параллельную реализацию меж собой, то можно увидеть, что время, которое тратится на создание потоков, является незначительным относительно времени, которое тратится на вычисление СЛАУ больших размерностей. Следовательно, можно сделать вывод, что увеличение числа потоков снижает время выполнения программы.

# Список использованных источников

1. Антонов А.С. Параллельное программирование с использованием технологии OpenMP: Учебное пособие. – М.: Изд-во МГУ, 2009. – 77 с.
2. Баркалов К.А. Методы параллельных вычислений. -Н. Новгород.: Изд-во Нижегородского госуниверситета им. Н.И. Лобачевского, 2011. -124 c.
3. Старченко А.В.Методы параллельных вычислений: Учебник. – Томск: Изд-во Том. ун-та, 2013. – 223 с.

# Листинг программы.

**Последовательная реализация:**

#include <iostream>

#include<vector>

#include<math.h>

using namespace std;

const double eps = 0.000001;

void getmatr(vector<vector<double>>&M,int n) {

    M.resize(n);

    for (int i = 0; i < n; ++i) {

        M[i].resize(n);

        for (int j = 0; j < n; ++j)

            M[i][j] = rand()% 100;

    }

}

void getb(vector<double>& b, int n) {

    b.resize(n);

    for (int i = 0; i < n; ++i)

        b[i] = rand() %100;

}

int find\_max(vector<vector<double>>& M,int t,int n) {

    double max = fabs(M[t][t]);

    int index = t;

    for (int i = t; i < n; ++i) {

        if (fabs(M[i][t]) > max) {

            max = fabs(M[i][t]);

            index = i;

        }

    }

    if (max < eps)

        return -1;

    else  return index;

}

void show\_matr(vector<vector<double>>&M,int n) {

    cout << "Значения матрицы:" << endl;

    for (int i = 0; i < n; ++i)

    {

        for (int j = 0; j < n; ++j)

            cout << M[i][j]<<" ";

        cout << endl;

    }

}

void showx(vector<double>x,int n)

{

    for (int i = 0; i < n; ++i)

        cout << "X[" << i << "]="<< x[i]<<endl;

}

vector<double> gauss(vector<vector<double>>& M, vector<double>& b, int n) {

    int m;

    vector<double> x(n);

    int k(0);

    for (int i = 0; i < n; ++i) {

        m = find\_max(M,i,n);

        if (m == -1) {

            cout << "Reshenie polychit neBozmozno";

            exit(1);

        }

        if (m != i) {

            swap(b[m], b[i]);

            swap(M[m], M[i]);

        }

        for (int j = i; j < n; ++j) {

            double temp = M[j][i];

            if (fabs(temp) < eps) continue;

            for (int k = 0; k < n; ++k)

                M[j][k] = M[j][k] / temp;

            b[j] = b[j] / temp;

            if (j == i) continue;

            for (k = 0; k < n; ++k)

                M[j][k] = M[j][k] - M[i][k];

            b[j] = b[j] - b[i];

        }

    }

    cout << "Result:" << '\n';

    for (int k = n - 1; k >= 0; --k) {

        x[k] = b[k];

        for (int i = 0; i < k; ++i)

            b[i] = b[i] - M[i][k] \* x[k];

    }

    return x;

}

int main()

{

    setlocale(LC\_ALL, "Russian");

    vector<vector<double>>M;

    vector<double> x;

    vector<double>b;

    int n;

    cout << "Введите размер матрицы n=" ;

    cin >> n;

    getmatr(M,n);

    show\_matr(M, n);

    getb(b, n);

    showx(b, n);

   x=gauss(M,b,n);

    showx(x,n);

return 0;

}

**Параллельная реализация №1:**

#include <iostream>

#include<vector>

#include<math.h>

#include<omp.h>

#include<chrono>

using namespace std;

const double eps = 0.000001;

void getmatr(vector<vector<double>>& M, int n) {

    M.resize(n);

    for (int i = 0; i < n; ++i) {

        M[i].resize(n);

        for (int j = 0; j < n; ++j)

            M[i][j] = rand() % 100;

    }

}

void getb(vector<double>& b, int n) {

    b.resize(n);

    for (int i = 0; i < n; ++i)

        b[i] = rand() % 100;

}

int find\_max(vector<vector<double>>& M, int t, int n) {

    double max = fabs(M[t][t]);

    int index = t;

#pragma omp parallel for

    for (int i = t; i < n; ++i) {

        if (fabs(M[i][t]) > max) {

            max = fabs(M[i][t]);

            index = i;

        }

    }

    if (max < eps)

        return -1;

    else  return index;

}

void show\_matr(vector<vector<double>>& M, int n) {

    cout << "Значения матрицы:" << endl;

    for (int i = 0; i < n; ++i)

    {

        for (int j = 0; j < n; ++j)

            cout << M[i][j] << " ";

        cout << endl;

    }

}

void showx(vector<double>x, int n)

{

    for (int i = 0; i < n; ++i)

        cout << "X[" << i << "]=" << x[i] << endl;

}

vector<double> gauss(vector<vector<double>>& M, vector<double>& b, int n) {

    vector<double> x(n);

    int k;

    for (int i = 0; i < n; ++i) {

       int  m = find\_max(M, i, n);

        if (m == -1) {

            cout << "Reshenie polychit neBozmozno";

            exit(1);

        }

        if (m != i) {

            swap(b[m], b[i]);

            swap(M[m], M[i]);

        }

        for (int j = i; j < n; ++j) {

            double temp = M[j][i];

            if (fabs(temp) < eps) continue;

#pragma omp parallel for private(k)

            for ( k = 0; k < n; ++k)

                M[j][k] = M[j][k] / temp;

            b[j] = b[j] / temp;

            if (j == i) continue;

#pragma omp parallel for private(k)

            for (k = 0; k < n; ++k)

                M[j][k] = M[j][k] - M[i][k];

            b[j] = b[j] - b[i];

        }

    }

    for (int k = n - 1; k >= 0; --k) {

        x[k] = b[k];

        int i;

#pragma omp parallel for  private(i)

        for ( i = 0; i < k; ++i)

           b[i] = b[i] - M[i][k] \* x[k];

    }

    return x;

}

int main()

{

       setlocale(LC\_ALL, "Russian");

for (int thread = 1; thread <= 16; thread \* 2) {

        cout << thread << endl;

        omp\_set\_num\_threads(thread);

        for (int n = 1000; n <= 10000; n += 1000) {

            omp\_set\_num\_threads(threads);

            vector<vector<double>>M(n);

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

                M[i].resize(n);

                for (int j = 0; j < n; ++j)

                    M[i][j] = rand()%100;

            }

            vector<double>b(n);

            for (int i = 0; i < n; ++i)

                b[i] = rand()%100;

            auto begin = chrono::steady\_clock::now();

            vector<double> x = gauss(M, b, n);

            auto end = chrono::steady\_clock::now();

            auto time = chrono::duration\_cast<chrono::seconds>(end - begin);

            cout << " Размер= " << n << "  Время:" << time.count() << endl;

        }

    }

   return 0;

}

**Параллельная реализация №2:**

#include<iostream>

#include<fstream>

#include<math.h>

#include<vector>

#include<omp.h>

#include<chrono>

using namespace std;

const double eps = 0.000001;

int find\_max(vector<vector<double>>& M, int t, int n) {

    double max = fabs(M[t][t]);

    int index = t;

#pragma omp parallel for

    for (int i = t + 1; i < n; ++i) {

        if (fabs(M[i][t]) > max) {

            max = fabs(M[i][t]);

            index = i;

        }

    }

    if (max < eps)

        return -1;

    else  return index;

}

void triangulation(vector<vector<double>>& M, int n) {

    const int num = M[0].size();

    for (int i = 0; i < n; ++i) {

        int max = find\_max(M, i, n);

        if (max == -1)

        {

            cout << "Решение получить невозможно";

            exit(1);

        }

        if (i != max) {

            swap(M[i], M[max]);

        }

#pragma omp parallel for

        for (int j = i + 1; j < n; ++j) {

            double temp = -M[j][i] / M[i][i];

            for (int k = i; k < num; ++k) {

                M[j][k] += M[i][k] \* temp;

            }

        }

    }

}

vector<double> gauss(vector<vector<double>>& M, vector<double>& b, int n) {

    vector<double>x(n);

    for (int i = 0; i < n; ++i) {

        M[i].push\_back(b[i]);

    }

    triangulation(M, n);

    for (int i = n - 1; i >= 0; --i) {

        x[i] = M[i][n] / M[i][i];

#pragma omp parallel for

        for (int j = 0; j < i; ++j) {

            M[j][n] -= M[j][i] \* x[i];

        }

    }

    return x;

}

void show(vector<double>x, int n) {

    for (int i = 0; i < n; ++i)

        cout << "X[" << i << "]=" << x[i] << endl;

}

int main()

{

    ofstream out("a.txt");

    setlocale(LC\_ALL, "Russian");

    for (int thread = 1; thread <= 16; thread \* 2) {

        cout << thread << endl;

        omp\_set\_num\_threads(thread);

        for (int n = 1000; n < 10000; n += 1000) {

            vector<vector<double>> M(n);

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

                M[i].resize(n);

                for (int j = 0; j < n; ++j)

                    M[i][j] = rand()%100;

            }

            vector<double> b(n);

            for (int j = 0; j < n; ++j)

                b[j] = rand()%100;

            auto begin = chrono::steady\_clock::now();

            vector<double> x = gauss(M, b, n);

            auto end = chrono::steady\_clock::now();

            auto time = chrono::duration\_cast<chrono::seconds>(end - begin);

            cout << " Размер= " << n << "  Время:" << time.count() << endl;

            out << " Размер= " << n << "Время" << time.count() << endl;

        }

    }

    return 0;

}