Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Волгоградский государственный технический университет»

Факультет_____Электроники и вычислительной техники_____

Кафедра_____Электронно вычислительные машины и системы_____

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к семестровой работе

по дисциплине_____Введение в параллельное программирование_____

на тему_____Распараллеливание алгоритма при помощи OMP и MPI_____
_____
_____

Студент_____Титов Алексей Константинович_____
(фамилия, имя, отчество)

Группа_____ИВТ - 460_____

Руководитель работы (проекта) _____ _____
(подпись и дата подписания) (инициалы и фамилия)

(

Волгоград 20__ г.

СОДЕРЖАНИЕ

Постановка задачи 3

Теоретический материал 4

Псевдокод прямого спуска 5

Формула для выполнения обратного хода алгоритма (нахождение вектора X) 5

Описание решения 5

Последовательная реализация 6

Описание решения 6

Результаты 6

OMP 7

Описание решения 7

Прямой ход 7

Обтаный ход 7

Результаты 7

Время выполнения (N) 7

График ускорения (N) 8

График эффективности (N) 8

Выводы 9

MPI 10

Описание решения 10

Результаты 10

Время выполнения (N) 10

График ускорения (N) 11

График эффективности (N) 11

Выводы 12

Сравнение реализаций с использованием OMP и MPI 13

Время выполнения всех вариантов программы 13

Сравнение ускорения вариантов с использованием OMP и MPI 14

Сравнение эффективности вариантов с использованием OMP и MPI 14

Приложение А. 15

Код последовательного варианта 15

Тестирующая функция 15

Расчетная функция 16

Код параллельного варианта с использованием OpenMP 17

Тестирующая функция 17

Расчетная функция 18

Код параллельного варианта с использованием MPI 19

Тестирующая функция 19

Расчетная функция 20

Код функции проверки решения 22

Приложение Б. История коммитов 23

ПОСТАНОВКА ЗАДАЧИ

Составить программы для решения задачи по варианту с использованием технологий параллельного программирования.

Вариант 2. Решение систем линейных алгебраических уравнений методом Гаусса
Программы необходимо разработать в среде Microsoft Visual Studio на языке программирования C/C++.
Требования к результатам, полученным в результате выполнения контрольной работы. Необходимо:

- разработать последовательный вариант программы решения задачи по варианту;
- при необходимости модифицировать его для получения параллельного алгоритма решения задачи;
- распараллелить полученную программу с использованием технологии OpenMP;
- получить зависимости характеристик ускорения и эффективности от размерности решаемой задачи при распараллеливания с использованием технологии OpenMP, от размеров блоков, на которые разбивается задача при распараллеливании; результат оформить в табличной и графической форме.

- проанализировать полученные результаты (OpenMP), сделать выводы;
- распараллелить полученную программу с использованием технологии MPI;
- получить зависимости характеристик ускорения и эффективности от размерности решаемой задачи при распараллеливания с использованием технологии MPI, от размеров блоков, на которые разбивается задача при распараллеливании; результат оформить в табличной и графической форме.
- проанализировать полученные результаты (MPI), сделать выводы.

ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ

В качестве источника была выбрана книга Левитина «Алгоритмы. Введение в разработку и анализ».

Ниже приведено краткое описание идеи алгоритма исключения Гаусса и псевдокод.

Также приведена формула и краткое описание обратного хода алгоритма для решения СЛАУ.

Идея метода заключается в преобразовании системы п линейных уравнений с п неизвестными в эквивалентную систему (т.е. систему с тем же решением, что и у исходной) с верхнетреугольной матрицей коэффициентов, т.е. такой, у которой все элементы ниже главной диагонали равны нулю:

Систему линейных уравнений с верхнетреугольной матрицей легко решить методом обратной подстановки следующим образом. Сначала мы вычисляем значение хп из последнего уравнения; затем подставляем полученное значение в предпоследнее уравнение и получаем значение хп-1. Продолжая выполнять подстановки вычисленных значений переменных в очередные уравнения, мы получим значения всех п переменных — от хп до х1.

Итак, каким же образом можно получить из системы линейных уравнений с произвольной матрицей коэффициентов А эквивалентную систему линейных уравнений с верхнетреугольной матрицей А'1 Это можно сделать при помощи последовательности так называемых элементарных операций:

- обмена двух уравнений системы линейных уравнений;
- умножения уравнения на ненулевую величину;
- замены уравнения на сумму или разность этого уравнения и другого уравнения, умноженного на некоторую величину.

Поскольку ни одна из элементарных операций не изменяет решение системы линейных уравнений, любая система линейных уравнений, полученная из исходной при помощи серии элементарных операций, будет иметь то же решение, что и исходная система линейных уравнений. Теперь посмотрим, как получить систему линейных уравнений с верхнетреугольной матрицей. Для начала используем в качестве опорного элемента а11 для того, чтобы сделать все коэффициенты при х1 в строках ниже первой нулевыми. В частности, заменим второе уравнение разностью между ним и первым уравнением, умноженным на а21/а11для того, чтобы получить нулевой коэффициент при х1. Выполняя то же для третьей, четвертой и далее строк и умножая первое уравнение, соответственно, на а31/а11, а41/а11, ..., an1/a11, сделаем все коэффициенты при х1 в уравнениях ниже первого равными 0. Затем обнулим все коэффициенты при х2 в уравнениях ниже второго, вычитая из каждого из этих уравнений второе, умноженное на соответствующий коэффициент. Повторяя эти действия для каждой из первых п — 1 строк, получим систему линейных уравнений с верхнетреугольной матрицей коэффициентов.

Псевдокод прямого спуска

АЛГОРИТМ $GaussElimination\,(A\,[1..n, 1..n]\,, b\,[1..n])$

// Применение метода исключения Гаусса к матрице
// коэффициентов системы линейных уравнений A, объединяемой
// со столбцом свободных членов b
// **Входные данные:** Матрица $A[1..n, 1..n]$ и вектор $b[1..n]$
// **Выходные данные:** Эквивалентная верхнетреугольная матрица
// на месте матрицы A со значениями
// в $n + 1$-ом столбце, соответствующим
// свободным членам новой системы линейных
// уравнений
for $i \leftarrow 1$ **to** n **do**
  $A[i, n + 1] \leftarrow b[i]$ // Расширение матрицы
for $i \leftarrow 1$ **to** $n - 1$ **do**
  for $j \leftarrow i + 1$ **to** n **do**
    for $k \leftarrow i$ **to** $n + 1$ **do**
      $A[j, k] \leftarrow A[j, k] - A[i, k] * A[j, i]/A[i, i]$

Формула для выполнения обратного хода алгоритма (нахождение вектора X)

$$X_i = b_i - \sum_{j=i+1}^{n} a_{ij} * X_j$$

ОПИСАНИЕ РЕШЕНИЯ

Для проверки решения была выбрана библиотека Eigen.

Исходные данные генерируются случайным образом. И далеко не всегда (особенно при решении задач большой размерности) удается сгенерировать матрицу с отличным от 0 определителем.

ПОСЛЕДОВАТЕЛЬНАЯ РЕАЛИЗАЦИЯ

Описание решения

Код аналогичен псевдокоду приведенному в предыдущей главе.

Вначале выделяется память под дополненную матрицу Ab

```
// Выделить память для расширенной матрицы A
    double ** Ab = new double *[n];
    for (int i = 0; i < n; i++)
        Ab[i] = new double[n + 1];

    // Скопировать данные из A
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            Ab[i][j] = A[i][j];

    // Скопировать данные из b
    for (int i = 0; i < n; i++)
        Ab[i][n] = b[i];
```

Затем осуществляется прямой ход алгоритма Гаусса

```
// Elimination (исключение или прямой ход)
    for (int i = 0; i < n - 1; i++) // Для всех строк, кроме предпоследней
    {
        for (int j = i + 1; j < n; j++) // Для каждой следующей за i-ой строкой
        {
            double divider = Ab[j][i] / Ab[i][i];
            for (int k = i; k < n + 1; k++) // Для каждого элемента j-ой строки, начиная с i-ого
                Ab[j][k] -= Ab[i][k] * divider;
        }
    }
```

Затем обратный ход алгоритма (находим вектор X)

```
/ Retrace (обратный ход)
    for (int i = n - 1; i >= 0; i--)
    {
        double rowSumm = 0.0; // aj*xj , где j [i+1; n]
        for (int j = i + 1; j < n; j++)
            rowSumm += Ab[i][j] * X[j];
        X[i] = (Ab[i][n] - rowSumm) / Ab[i][i];
    }
```

Результаты

n	Serial Time
10	2.44E-05
25	0.000116
50	0.000283
100	0.001817
200	0.013703
400	0.102734
800	0.876136
1000	1.6336
1300	3.67071
1600	6.86305

OMP

Описание решения

Было решено распараллелить 2 цикла при прямом ходе и 1 цикл при обратном ходе.

Прямой ход

```
for (int i = 0; i < n - 1; i++) // Для всех строк, кроме предпоследней
    {
        #pragma omp parallel for
        for (int j = i + 1; j < n; j++) // Для каждой следующей за i-ой строкой
        {
            double divider = Ab[j][i] / Ab[i][i];
            #pragma omp parallel for
            for (int k = i; k < n + 1; k++) // Для каждого элемента j-ой строки (с i-ого)
            {
                Ab[j][k] -= Ab[i][k] * divider;
            }
        }
    }
```

Обтаный ход

```
    for (int i = n - 1; i >= 0; i--)
    {
        double rowSumm = 0.0; // aj*xj , где j [i+1; n]
        #pragma omp parallel for reduction(+:rowSumm)
        for (int j = i + 1; j < n; j++)
            rowSumm += Ab[i][j] * X[j];

        X[i] = (Ab[i][n] - rowSumm) / Ab[i][i];
    }
```

Результаты

Время выполнения (N)

n	OMP Time
10	0.000607
25	0.000359
50	0.000966
100	0.002789
200	0.012875
400	0.080147
800	0.934376
1000	1.08054
1300	2.47193
1600	4.46157

График ускорения (N)

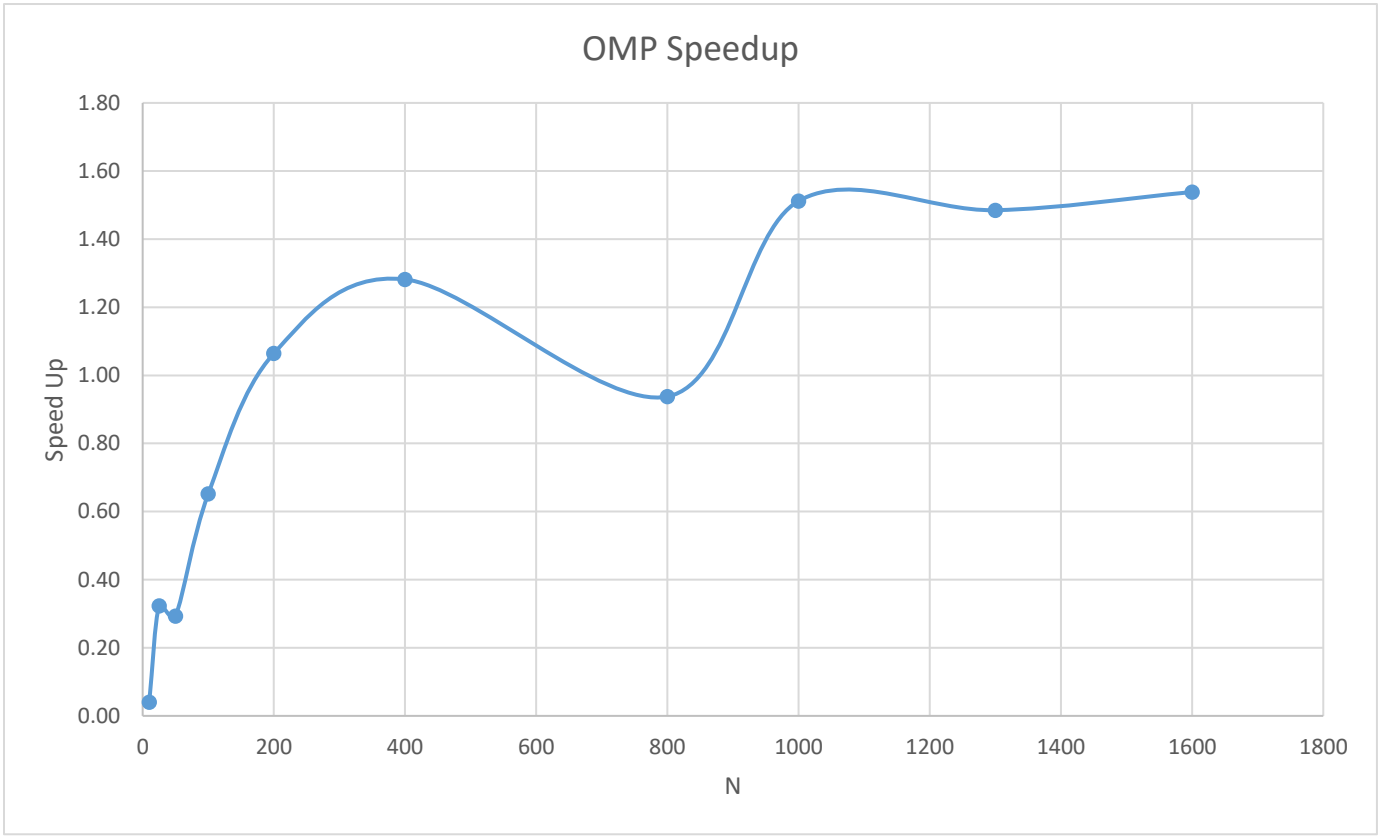
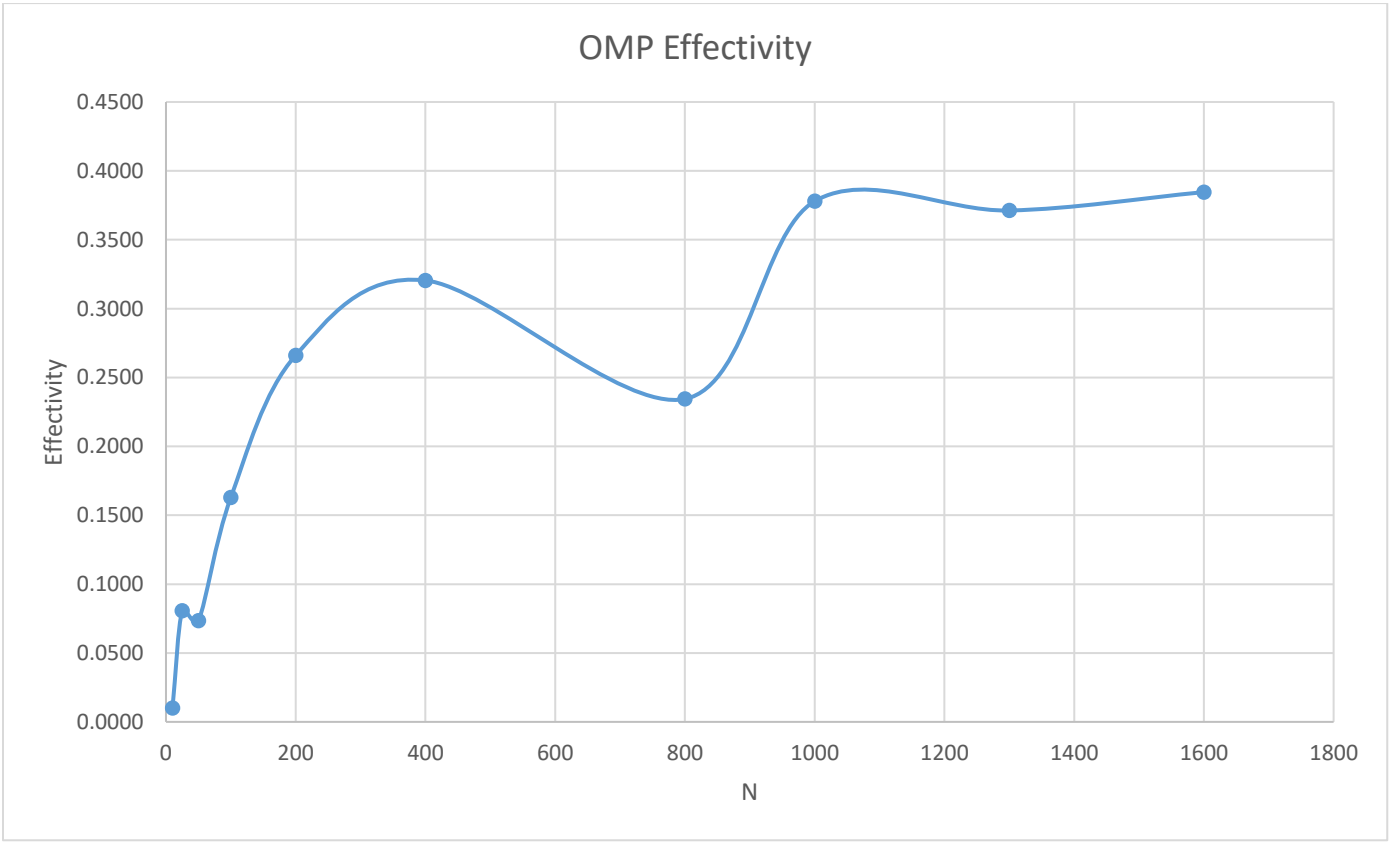


График эффективности (N)



Выводы

Вплоть до размера задачи N = 200, параллельная реализация с использованием OpenMP проигрывает последовательной. Но при дальнейшем увеличении размера задачи ускорение превышает значение 1 и начинает расти до 1000, затем рост замедляется.

Причина этого, вероятно, кроется в мелкозернистости алгоритма.

MPI

Описание решения

Главный поток рассылает всем потокам строку I

```
// Отсылаем i - ую строку всем подпроцессам
        if (isMain) Ab_i = Ab[i];
        MPI_Bcast(Ab_i, n + 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
```

Затем в цикле рассылает строки j рабочим процессам

```
for (int j = i + 1; j < n; j++)
        {
            int recieverRank = 1;
            if (isMain) // рассылает j-ую строку
            {
                Ab_j = Ab[j];
                MPI_Send(Ab_j, n + 1, MPI_DOUBLE, recieverRank, 0, MPI_COMM_WORLD);

                MPI_Status masterRecvStatus;
                MPI_Recv(Ab_j, n + 1, MPI_DOUBLE, recieverRank, MPI_ANY_TAG, MPI_COMM_WORLD,
&masterRecvStatus);

                recieverRank++;
                if (recieverRank > procSize - 1)
                    recieverRank = 1;
            }
```

Каждый рабочий процесс принимает строки I и j от главного процесса и обрабатывает их, вычитая i-ую строку из j-ой так, чтобы обнулить коэффициент при j

```
else if (procRank == recieverRank)
            {
                MPI_Status recvStatus;
                MPI_Recv(Ab_j, n + 1, MPI_DOUBLE, 0, MPI_ANY_TAG, MPI_COMM_WORLD, &recvStatus);

                // Процесс считает j - ую строку
                double divider = Ab_j[i] / Ab_i[i];
                for (int k = 0; k < n + 1; k++)
                    Ab_j[k] -= Ab_i[k] * divider;

                MPI_Send(Ab_j, n + 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
            }
```

Обратный ход выполняется только главным процессом, т.к. там мало работы и почти последовательный код (для каждого i-ого X нужно знать все X от 0 до i-1 -ого.

Результаты

Время выполнения (N)

n	10	25	50	100	200	400	800	1000	1300	1600
MPI	0.000304	0.000566	0.00240	0.0182	0.0593	0.381	4.66	7.73	15.2	25.54

График ускорения (N)

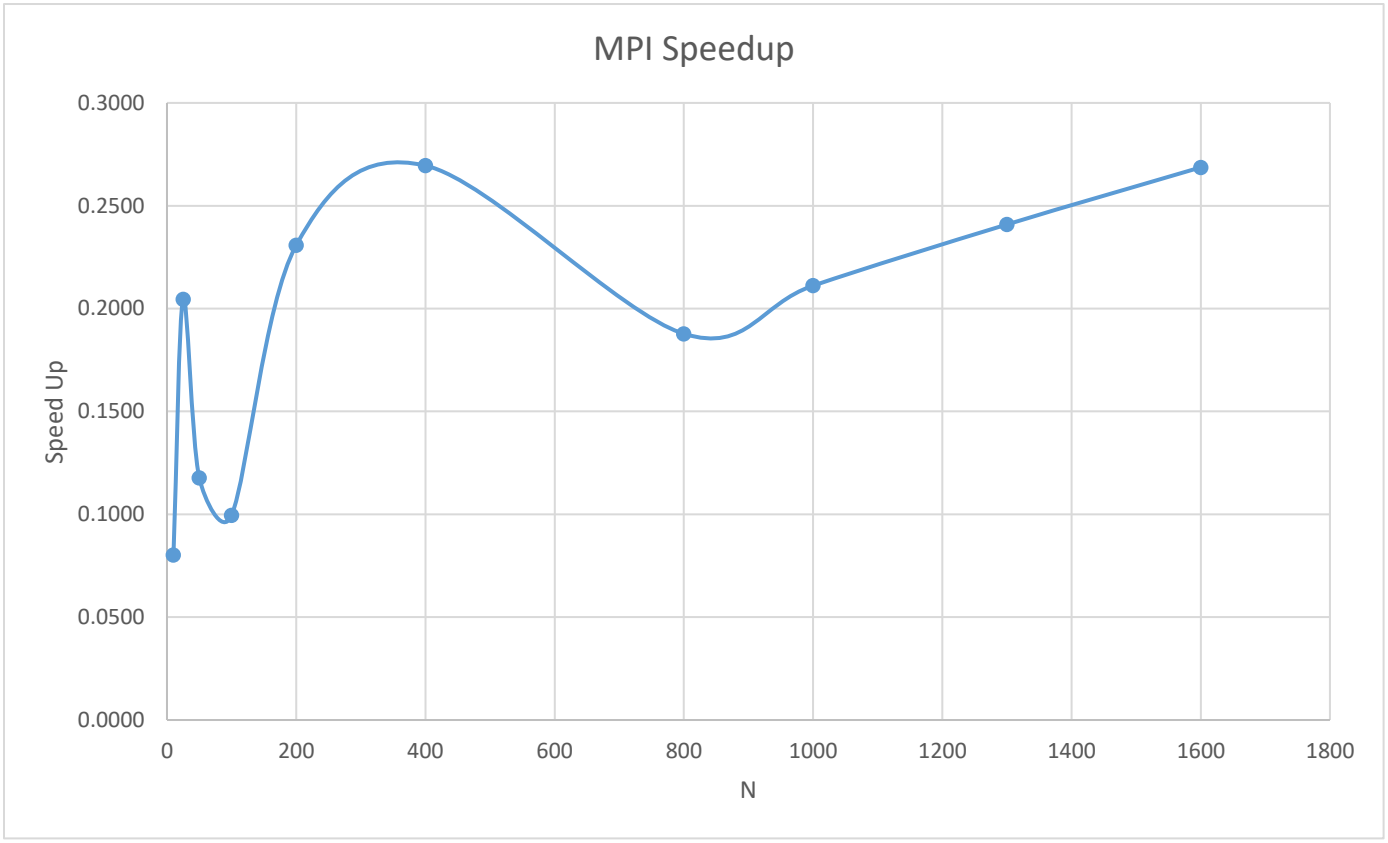
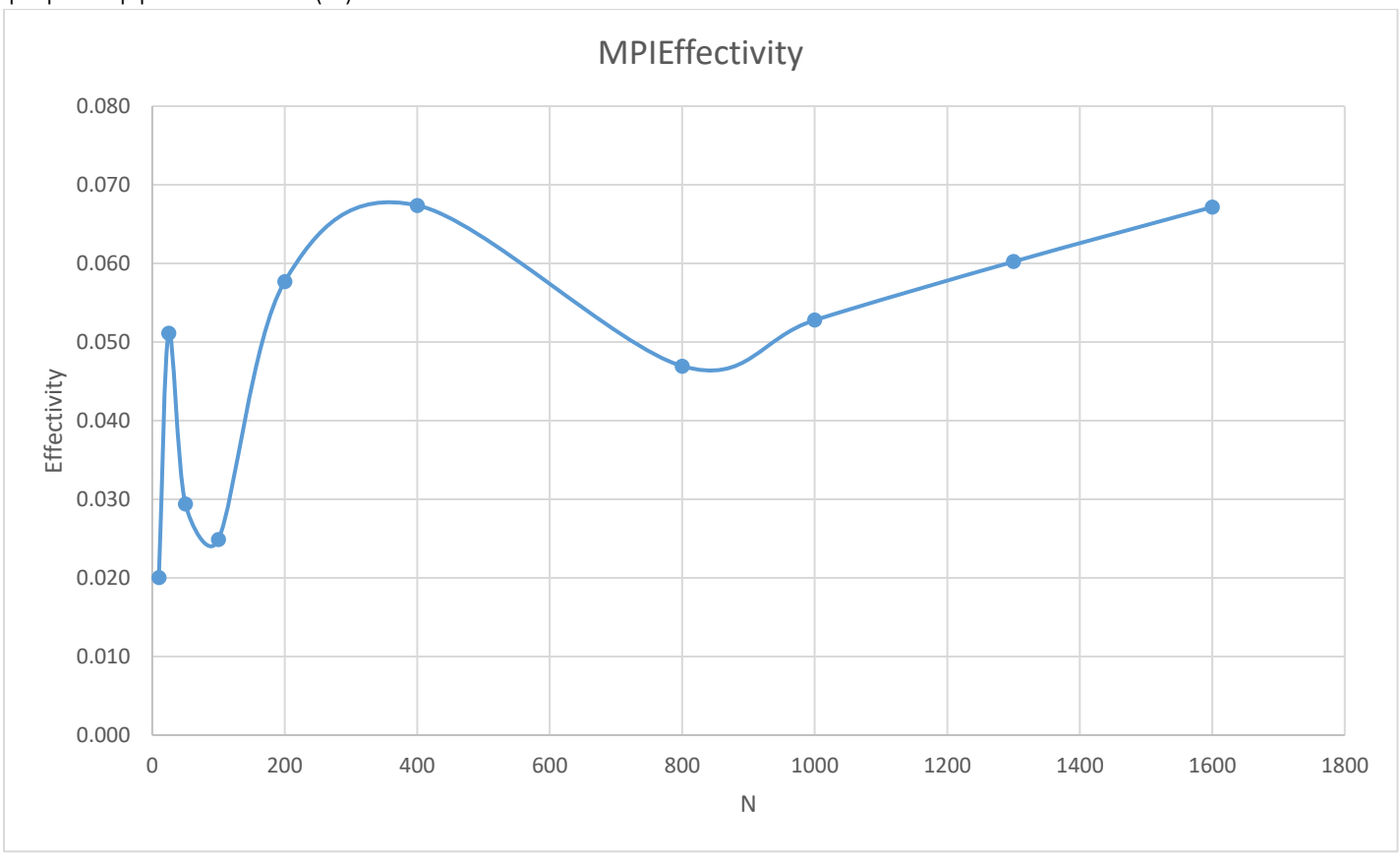


График эффективности (N)



Выводы

Программа работает медленнее, чем последовательный вариант.

Вероятной причиной является малозернистость алгоритма. Строки вместо обработки на центральном процессе отправляются на другие, обрабатываются там, а затем возвращаются)

Время отправки большой строки, малозернистость обработки ее на рабочем процессе и время возврата приводят к сильному замедлению работы программы по сравнению с последовательным вариантом.

Также ускорения в этом случае можно добиться, представив двумерный массив в виде одномерного и используя вместе с этим коллективные операции.

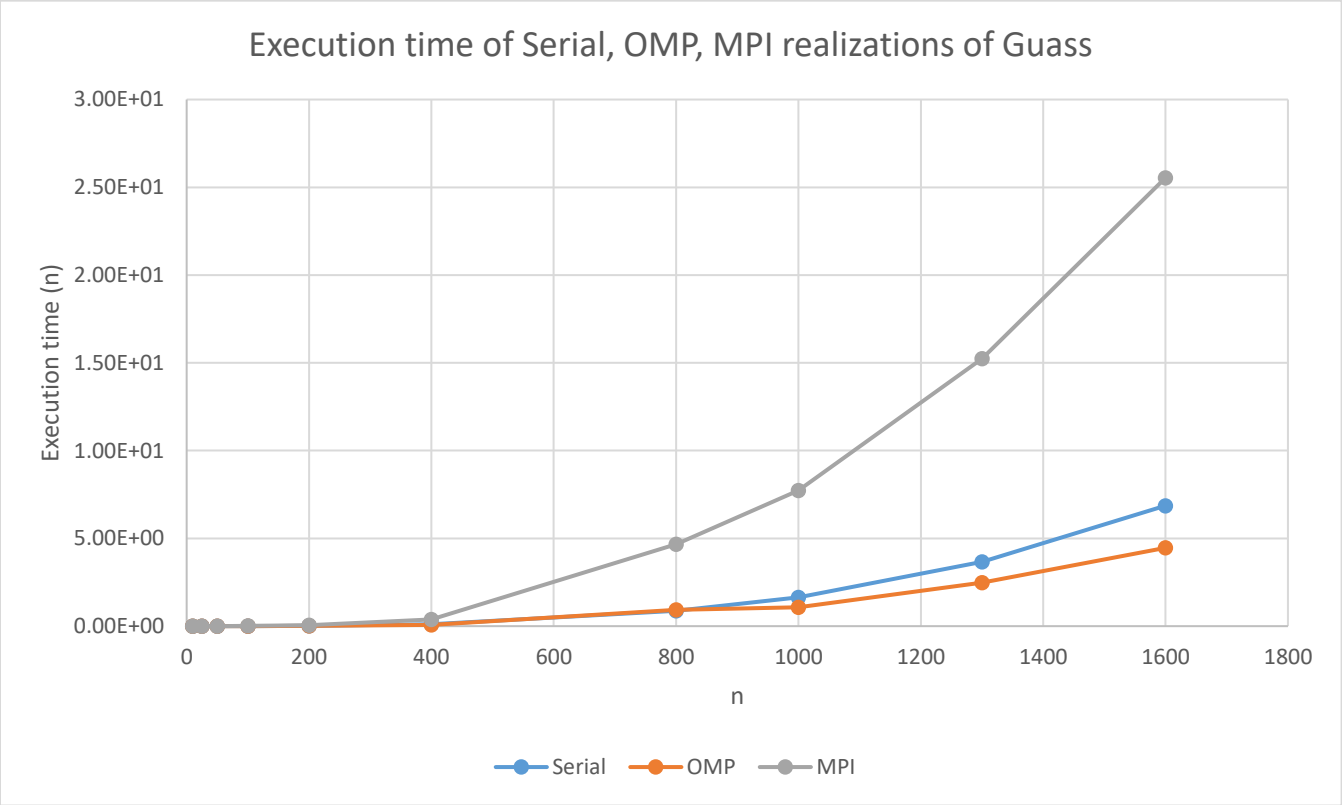
Однако этот подход усложнит сильно алгоритм. Процесс может получить 1.5 строки матрицы или 33.3 (иррациональное число строк j)

Еще один путь к ускорению – использование ISend, IRecv.

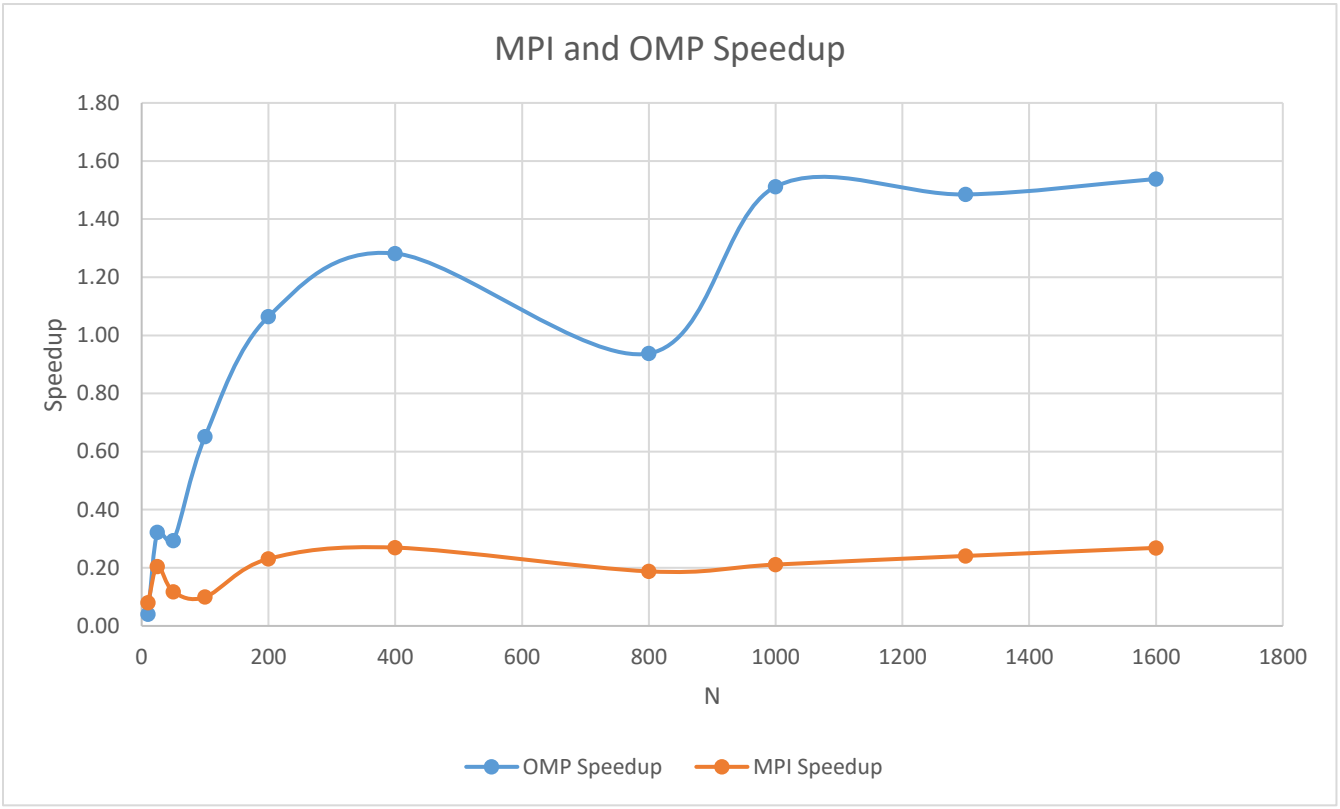
СРАВНЕНИЕ РЕАЛИЗАЦИЙ С ИСПОЛЬЗОВАНИЕМ OMP И MPI

Время выполнения всех вариантов программы

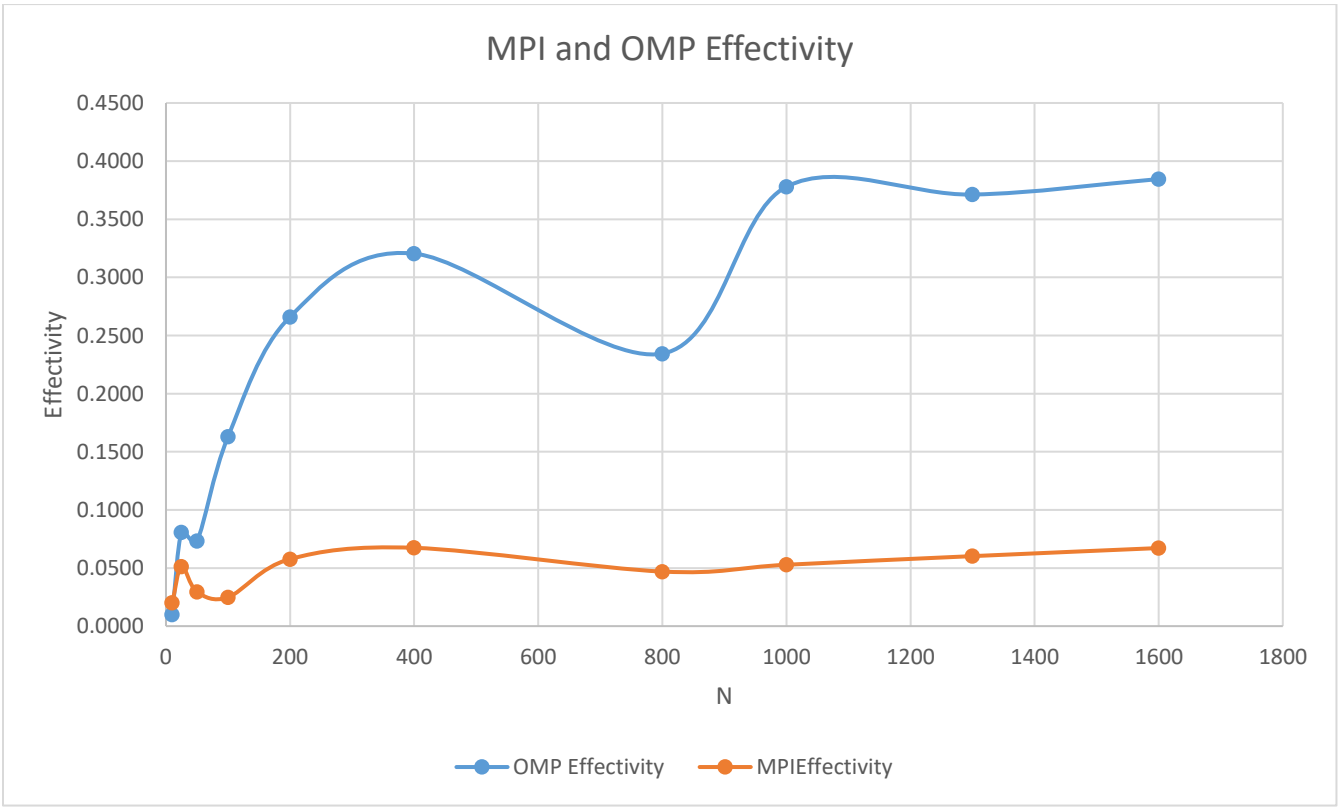
n	Serial	OMP	MPI
10	2.44E-05	0.000607	0.000304
25	0.000116	0.000359	0.000567
50	0.000283	0.000966	0.002406
100	0.001817	0.002789	0.018263
200	0.013703	0.012875	0.059368
400	0.102734	0.080147	0.381169
800	0.876136	0.934376	4.66615
1000	1.6336	1.08054	7.73664
1300	3.67071	2.47193	15.2375
1600	6.86305	4.46157	25.5454



Сравнение ускорения вариантов с использованием OMP и MPI



Сравнение эффективности вариантов с использованием OMP и MPI



ПРИЛОЖЕНИЕ А.

Код последовательного варианта

Тестирующая функция

```
double TestSerial(int n)
{
    // Мое решение
    double **A = CreateRandomMatrix(n);
    double *b = CreateRandomVector(n);
    double *X = CreateRandomVector(n);

    double start = omp_get_wtime();
    GaussSolveSerial(A, b, X, n);
    double end = omp_get_wtime();
    double serial_time = end - start;

    bool isCorrectAnswerSerial = isCorrectAnswer(A, b, X, n);

    DeleteMatrix(A, n);
    DeleteVector(b);
    DeleteVector(X);

    cout << "[Serial( " << n << " )] " << boolToStr(isCorrectAnswerSerial) << " Time: "
<< serial_time << endl;
    return serial_time;
}
```

Расчетная функция

```
void GaussSolveSerial(double **A, double * b, double * X, int n)
{
    // Выделить память для расширенной матрицы A
    double ** Ab = new double *[n];
    for (int i = 0; i < n; i++)
        Ab[i] = new double[n + 1];

    // Скопировать данные из A
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            Ab[i][j] = A[i][j];

    // Скопировать данные из b
    for (int i = 0; i < n; i++)
        Ab[i][n] = b[i];


    // Elimination (исключение или прямой ход)
    for (int i = 0; i < n - 1; i++) // Для всех строк, кроме предпоследней
    {
        // Можно распараллелить
        for (int j = i + 1; j < n; j++) // Для каждой следующей за i-ой строкой
        {
            double divider = Ab[j][i] / Ab[i][i];
            // Можно распараллелить
            for (int k = i; k < n + 1; k++) // Для каждого элемента j-ой строки (с i-ого)
            {
                Ab[j][k] -= Ab[i][k] * divider;
            }
        }
    }

    // Retrace (обратный ход)
    for (int i = n - 1; i >= 0; i--)
    {
        double rowSumm = 0.0; // aj*xj , где j [i+1; n]
        for (int j = i + 1; j < n; j++)
            rowSumm += Ab[i][j] * X[j];

        X[i] = (Ab[i][n] - rowSumm) / Ab[i][i];
    }

    DeleteMatrix(Ab, n);
}
```

Код параллельного варианта с использованием OpenMP

Тестирующая функция

```
double TestOMP(int n)
{
    // Мое решение
    double **A = CreateRandomMatrix(n);
    double *b = CreateRandomVector(n);
    double *X = CreateRandomVector(n);

    double start, end;
    start = omp_get_wtime();
    GaussSolveOpenMP(A, b, X, n);
    end = omp_get_wtime();
    double openmp_time = end - start;

    bool isCorrectAnswerOpenMP = isCorrectAnswer(A, b, X, n);

    DeleteMatrix(A, n);
    DeleteVector(b);
    DeleteVector(X);

    cout << "[OpenMP( " << n << " )] " << boolToStr(isCorrectAnswerOpenMP) << " Time: "
<< openmp_time << endl;
    return openmp_time;
}
```

Расчетная функция

```
void GaussSolveOpenMP(double **A, double * b, double * X, int n)
{
    // Выделить память для расширенной матрицы A
    double ** Ab = new double *[n];
    for (int i = 0; i < n; i++)
        Ab[i] = new double[n + 1];

    // Скопировать данные из A
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            Ab[i][j] = A[i][j];

    // Скопировать данные из b
    for (int i = 0; i < n; i++)
        Ab[i][n] = b[i];


    // Elimination (исключение или прямой ход)
    for (int i = 0; i < n - 1; i++) // Для всех строк, кроме предпоследней
    {
        #pragma omp parallel for
        for (int j = i + 1; j < n; j++) // Для каждой следующей за i-ой строкой
        {
            double divider = Ab[j][i] / Ab[i][i];
            #pragma omp parallel for
            for (int k = i; k < n + 1; k++) // Для каждого элемента j-ой строки (с i-ого)
            {
                Ab[j][k] -= Ab[i][k] * divider;
            }
        }
    }

    // Retrace (обратный ход)
    for (int i = n - 1; i >= 0; i--)
    {
        double rowSumm = 0.0; // aj*xj , где j [i+1; n]
        #pragma omp parallel for reduction(+:rowSumm)
        for (int j = i + 1; j < n; j++)
            rowSumm += Ab[i][j] * X[j];

        X[i] = (Ab[i][n] - rowSumm) / Ab[i][i];
    }

    DeleteMatrix(Ab, n);
}
```

Код параллельного варианта с использованием MPI

Тестирующая функция

```
double TestMPI(int n)
{
    int procRank, procSize;
    MPI_Comm_rank(MPI_COMM_WORLD, &procRank);
    MPI_Comm_size(MPI_COMM_WORLD, &procSize);

    double **A = NULL;
    double *b = NULL;
    double *X = NULL;

    // Выделение памяти
    bool isMain = procRank == 0;
    if (isMain)
    {
        A = CreateRandomMatrix(n);
        b = CreateRandomVector(n);
        X = CreateRandomVector(n);
    }

    // Тестирование
    MPI_Barrier(MPI_COMM_WORLD);
    double start, end;
    if (isMain) start = MPI_Wtime();
    GaussSolveMPI(A, b, X, n);
    if (isMain) end = MPI_Wtime();
    MPI_Barrier(MPI_COMM_WORLD);

    double mpi_time = 0.0;
    if (isMain) mpi_time = end - start;

    // Проверка решения и вывод результатов
    if (isMain)
    {
        bool isCorrectAnswerMPI = isCorrectAnswer(A, b, X, n);
        cout << "[ MPI  ( " << n << " )] " << boolToStr(isCorrectAnswerMPI) << " Time: "
<< mpi_time << endl;
    }

    if (isMain)
    {
        DeleteMatrix(A, n);
        DeleteVector(b);
        DeleteVector(X);
    }

    return mpi_time;
}
```

Расчетная функция

```
void GaussSolveMPI(double **A, double * b, double * X, int n)
{
    int procRank, procSize;
    MPI_Comm_rank(MPI_COMM_WORLD, &procRank);
    MPI_Comm_size(MPI_COMM_WORLD, &procSize);
    bool isMain = procRank == 0;
    double ** Ab = NULL;
    if (isMain)
    {
        // Выделить память для расширенной матрицы A
        Ab = new double *[n];
        for (int i = 0; i < n; i++)
            Ab[i] = new double[n + 1];

        // Скопировать данные из A
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                Ab[i][j] = A[i][j];

        // Скопировать данные из b
        for (int i = 0; i < n; i++)
            Ab[i][n] = b[i];
    }

    // Elimination (исключение или прямой ход)
    double * Ab_i, * Ab_j; // строки для рабочих процессов
    if (!isMain)
    {
        Ab_i = new double[n + 1];
        Ab_j = new double[n + 1];
    }

    for (int i = 0; i < n - 1; i++) // Для всех строк, кроме предпоследней
    {
        // Отсылаем i - ую строку всем подпроцессам
        if (isMain) Ab_i = Ab[i];
        MPI_Bcast(Ab_i, n + 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

        for (int j = i + 1; j < n; j++)
        {
            int recieverRank = 1;
            if (isMain) // рассылает j-ую строку
            {
                Ab_j = Ab[j];
                MPI_Send(Ab_j, n + 1, MPI_DOUBLE, recieverRank, 0, MPI_COMM_WORLD);

                MPI_Status masterRecvStatus;
                MPI_Recv(Ab_j, n + 1, MPI_DOUBLE, recieverRank, MPI_ANY_TAG,
MPI_COMM_WORLD, &masterRecvStatus);

                recieverRank++;
                if (recieverRank > procSize - 1)
```

```
                    recieverRank = 1;
            }
            else if (procRank == recieverRank)
            {
                MPI_Status recvStatus;
                MPI_Recv(Ab_j, n + 1, MPI_DOUBLE, 0, MPI_ANY_TAG, MPI_COMM_WORLD,
&recvStatus);

                // Процесс считает j - ую строку
                double divider = Ab_j[i] / Ab_i[i];
                for (int k = 0; k < n + 1; k++)
                    Ab_j[k] -= Ab_i[k] * divider;

                MPI_Send(Ab_j, n + 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
            }


        }
    }

    if (isMain)
    {
        // Retrace (обратный ход)
        for (int i = n - 1; i >= 0; i--)
        {
            double rowSumm = 0.0; // aj*xj , где j [i+1; n]
            for (int j = i + 1; j < n; j++)
                rowSumm += Ab[i][j] * X[j];

            X[i] = (Ab[i][n] - rowSumm) / Ab[i][i];
        }
    }

    if (isMain)
    {
        DeleteMatrix(Ab, n);
    }
    else
    {
        delete[] Ab_j;
        delete[] Ab_i;
    }
}
```

Код функции проверки решения

```
bool isCorrectAnswer(double ** A, double * b, double * X, int n)
{
    // Решение библиотеки Eigen
    MatrixXf A_(n, n);
    VectorXf b_(n);
    VectorXf X_(n);

    for (int i = 0; i < n; i++)
        b_(i) = b[i];

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            A_(i, j) = A[i][j];

    X_ = A_.colPivHouseholderQr().solve(b_);


    bool isCorrectX = true;
    for (int i = 0; i < n; i++)
        isCorrectX &= areEquals(X[i], X_(i));

    return isCorrectX;
}
```

Приложение Б. История коммитов