



Министерство науки и образования Российской Федерации
Государственное образовательное учреждение высшего профессионального
образования
«Московский государственный технический университет
имени Н. Э. Баумана»

Отчет по лабораторной работе №2
По курсу:
«Разработка программных систем»
По теме:
«Многопроцессное программирование»

Вариант №4

Выполнил:	Никатов. В.А
Группа:	РК6-61
Преподаватель:	Федорук. В.Г.

Москва, 2018г.

Содержание

1. Текст задания	3
2. Описание архитектуры программы	4
<i>Рисунок 1. Прямой ход столбцового алгоритма</i>	<i>5</i>
<i>Рисунок 2. Обратный ход столбцового алгоритма</i>	<i>5</i>
3. Блок-схемы	6
<i>Рисунок 3. Блок-схема работы родительского потока</i>	<i>6</i>
<i>Рисунок 4. Блок-схема работы дочерних потоков</i>	<i>7</i>
4. Пример результата работы программы	8
5. Текст программы	9

1. Текст задания

Разработать, используя средства многопоточного программирования, параллельную программу решения методом Гаусса системы линейных алгебраических уравнений без учета разреженности матрицы коэффициентов. Параллельные потоки (их количество - аргумент программы) ведут исключение элементов в прямом ходе по *столбцам*. В качестве тестовых использовать реальные матрицы из [хранилища](#) или самостоятельно генерировать тестовую СЛАУ описанным ниже [способом](#). См. также [замечание](#). Программа должна демонстрировать [ускорение](#) по сравнению с последовательным вариантом. Предусмотреть возможность диагностического вывода структуры матрицы коэффициентов в течение прямого хода метода Гаусса. Отчет должен содержать подробное (иллюстрированное картинками) описание использованного способа распараллеливания и синхронизации.

2. Описание архитектуры программы

Задача реализуется с помощью потоков стандарта C++11, в котором, в сравнении с POSIX стандартом, отсутствует аналог `barrier_t` из библиотеки `pthread`. Барьер, опыта ради, было решено не брать из сторонних библиотек, а реализовать самому.

Многопоточность и синхронизация происходят следующим образом: основной поток при инициализации объекта класса `AdvancedMatrix` производит генерацию тестовой СЛАУ описанным в руководстве [способом](#), после чего вызывается метод `startGauss`, в котором основной поток стартует n потоков для выполнения прямого хода по столбцам: вызывается функция `divisionString`, в которой n -ая строка поэлементно делится на n -ый элемент этой же строки. Деление происходит параллельно. Как только строка разделена, потоки переходят к следующей задаче: из всех строк, расположенных ниже n -й вычитается строка n , домноженная на коэффициент равный n -му элементу слагаемой строки. Данные вычисления так же происходят параллельно, но теперь распараллеливание происходит не по элементам, а по строкам. Основной прирост в скорости за счет использования нескольких потоков происходит именно при параллельном вычитании строк. После этого потоки приступают к обратному ходу. Обратный ход так же распараллелен по строкам.

Работы столбцового алгоритма проиллюстрирована ниже на примере матрицы 5×5 с использованием двух потоков, где

 Первый поток

 Второй поток

 Результат обработки элемента матрицы первым потоком

 Результат обработки элемента матрицы вторым потоком


 Направление итераций потоков

Иллюстрация прямого хода Гаусса:

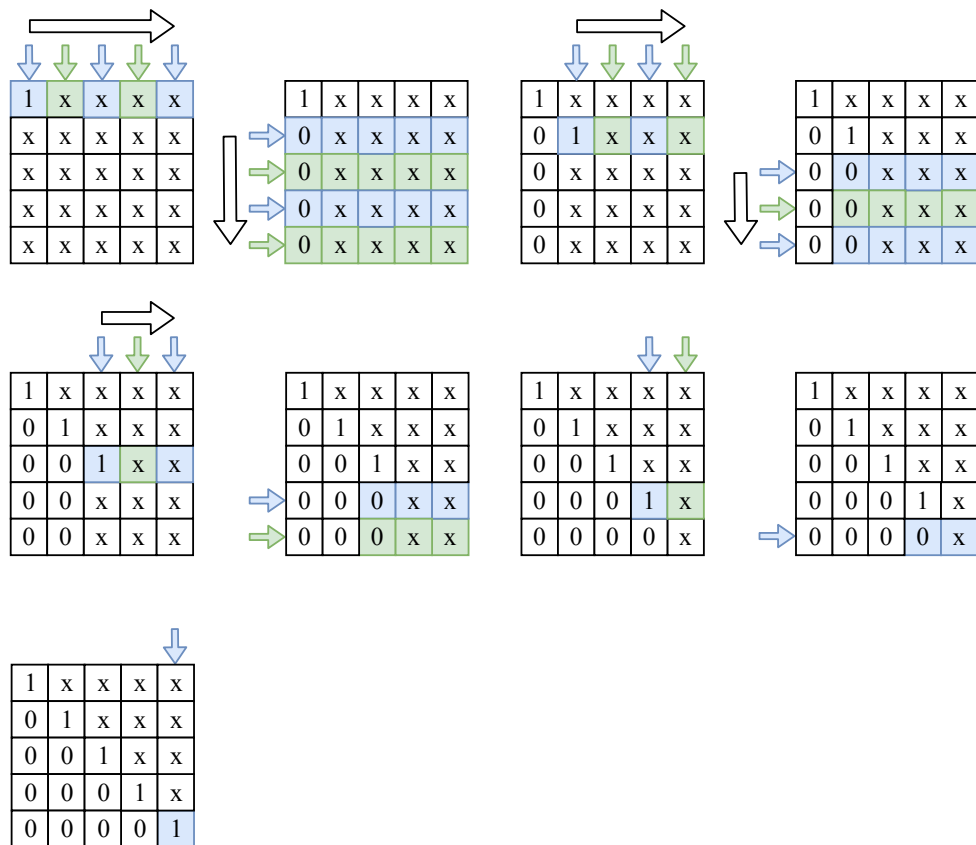


Рисунок 1. Прямой ход столбцового алгоритма

Иллюстрация обратного хода Гаусса:

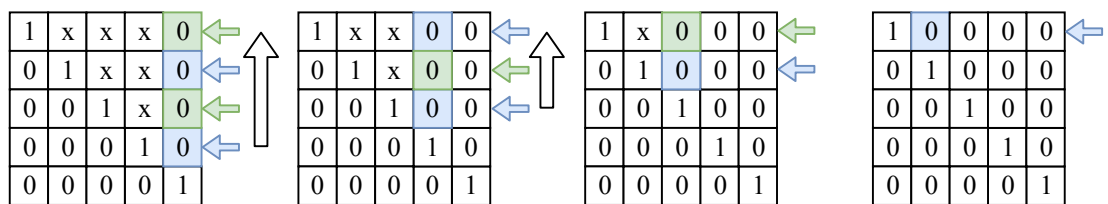


Рисунок 2. Обратный ход столбцового алгоритма

3. Блок-схемы

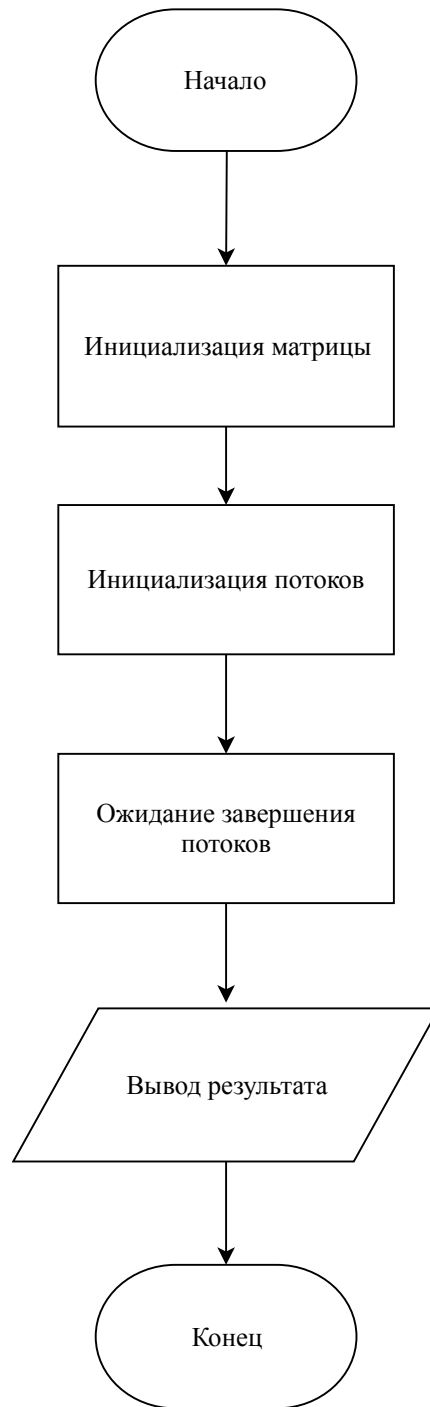


Рисунок 3. Блок-схема работы родительского потока

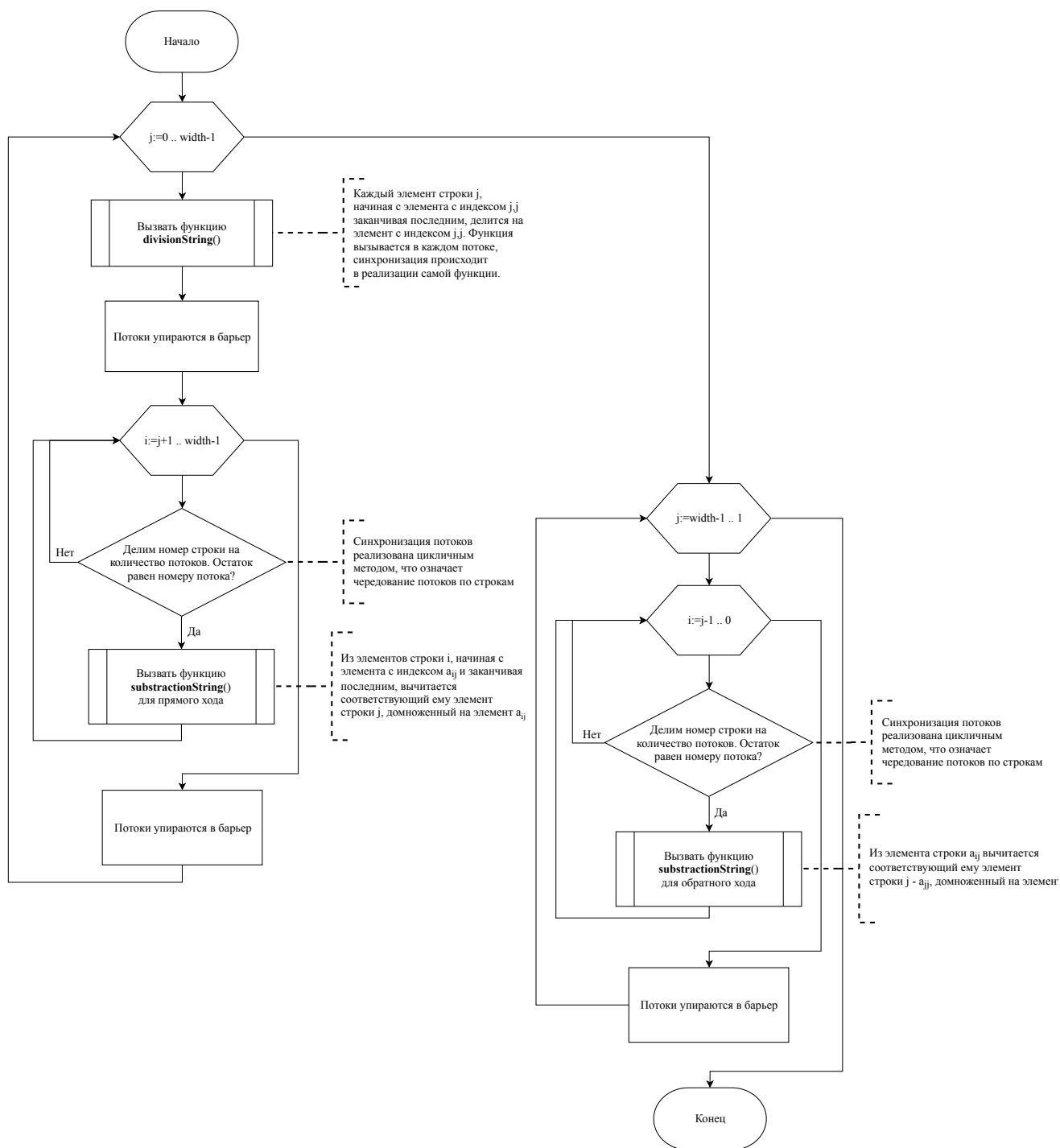


Рисунок 4. Блок-схема работы дочерних потоков

4. Пример результата работы программы

```

lab2 — -bash — 47x71
[MacBook-Air-Vladislav:lab2 destr0y$ ./lab2 5 4 ]
Сгенерированная матрица:
  0.483 -0.460 -0.870 -0.808  0.691 | -0.965
 -0.621 -0.495 -0.883  0.911  0.378 | -0.710
  0.741  0.256  0.072 -0.334 -0.644 |  0.092
  0.575  0.076  0.682 -0.123  0.381 |  1.591
  0.547 -0.750 -0.087  0.169  0.565 |  0.443

Прямой ход:
  1.000 -0.954 -1.802 -1.674  1.431 | -1.999
  0.000 -1.088 -2.002 -0.129  1.267 | -1.952
  0.000  0.963  1.408  0.907 -1.704 |  1.573
  0.000  0.625  1.718  0.839 -0.442 |  2.740
  0.000 -0.229  0.897  1.084 -0.217 |  1.535

  1.000 -0.954 -1.802 -1.674  1.431 | -1.999
  0.000  1.000  1.840  0.119 -1.164 |  1.794
  0.000  0.000 -0.365  0.792 -0.583 | -0.155
  0.000  0.000  0.568  0.765  0.286 |  1.619
  0.000  0.000  1.318  1.111 -0.483 |  1.945

  1.000 -0.954 -1.802 -1.674  1.431 | -1.999
  0.000  1.000  1.840  0.119 -1.164 |  1.794
  0.000  0.000  1.000 -2.172  1.597 |  0.425
  0.000  0.000  0.000  1.999 -0.622 |  1.378
  0.000  0.000  0.000  3.973 -2.588 |  1.386

  1.000 -0.954 -1.802 -1.674  1.431 | -1.999
  0.000  1.000  1.840  0.119 -1.164 |  1.794
  0.000  0.000  1.000 -2.172  1.597 |  0.425
  0.000  0.000  0.000  1.000 -0.311 |  0.689
  0.000  0.000  0.000  0.000 -1.352 | -1.352

  1.000 -0.954 -1.802 -1.674  1.431 | -1.999
  0.000  1.000  1.840  0.119 -1.164 |  1.794
  0.000  0.000  1.000 -2.172  1.597 |  0.425
  0.000  0.000  0.000  1.000 -0.311 |  0.689
  0.000  0.000  0.000  0.000  1.000 |  1.000

Обратный ход:
  1.000 -0.954 -1.802 -1.674  0.000 | -3.430
  0.000  1.000  1.840  0.119  0.000 |  2.959
  0.000  0.000  1.000 -2.172  0.000 | -1.172
  0.000  0.000  0.000  1.000  0.000 |  1.000
  0.000  0.000  0.000  0.000  1.000 |  1.000

  1.000 -0.954 -1.802  0.000  0.000 | -1.756
  0.000  1.000  1.840  0.000  0.000 |  2.840
  0.000  0.000  1.000  0.000  0.000 |  1.000
  0.000  0.000  0.000  1.000  0.000 |  1.000
  0.000  0.000  0.000  0.000  1.000 |  1.000

  1.000 -0.954  0.000  0.000  0.000 |  0.046
  0.000  1.000  0.000  0.000  0.000 |  1.000
  0.000  0.000  1.000  0.000  0.000 |  1.000
  0.000  0.000  0.000  1.000  0.000 |  1.000
  0.000  0.000  0.000  0.000  1.000 |  1.000

  1.000  0.000  0.000  0.000  0.000 |  1.000
  0.000  1.000  0.000  0.000  0.000 |  1.000
  0.000  0.000  1.000  0.000  0.000 |  1.000
  0.000  0.000  0.000  1.000  0.000 |  1.000
  0.000  0.000  0.000  0.000  1.000 |  1.000

Ответ:
x0 = 1.000
x1 = 1.000
x2 = 1.000
x3 = 1.000
x4 = 1.000
Время работы алгоритма: 0.002

```

Рисунок 4. Результат работы программы

5. Текст программы

```
#include <iostream>
#include <cstdlib>
#include <cstring>
#include <cstdio>
#include <ctime>
#include <iomanip>
#include <thread>
#include <vector>
#include <mutex>
#include <condition_variable>
#include <atomic>
#include <chrono>

using namespace std;

class barrier
{
public:
    barrier(unsigned int n) : threadCount(n),
                             threadsWaiting(0),
                             isNotWaiting1(false),
                             isNotWaiting2(false),
                             tumbler (true) {}

    // Удаляем конструктор копирования
    barrier(const barrier&) = delete;

    void wait();
private:
    const unsigned int threadCount;
    atomic<unsigned int> threadsWaiting;

    condition_variable waitVariable;
    mutex mut;

    /* Чтобы, при входе в следующий барьер, не переключить переменную
     * isNotWaiting в false, по которой еще не все потоки успели выйти из
     * предыдущего барьера в цикле проверки избежания spurious wakeup
     * (ложных блокировок), необходимо чередовать переменные isNotWaiting.
     */
    bool isNotWaiting1;
    bool isNotWaiting2;
    bool tumbler;
    bool switchTumbler(){ tumbler ? tumbler = false : tumbler = true;
    return !tumbler; }
};
```

```

void barrier::wait()
{
    unique_lock<mutex> lock(mut);
    /* Создаем указатель, который, в зависимости от положения тумблера,
     * указывает на одну или другую переменную. Далее пользуемся указателем.
     * По освобождению барьера, тумблер переключается. Таким образом, для
     * потоков, оставшихся в барьере указатель isNotWaiting ссылается на
     * одну переменную, а для потоков, дошедших до следующего барьера – уже
     * на другую.
     */
    bool* isNotWaiting = tumbler ? &isNotWaiting1 : &isNotWaiting2;
    if (threadsWaiting.fetch_add(1) >= threadCount - 1) // RMW (read-modify-
write)
    {
        *isNotWaiting = true;
        switchTumbler();
        waitVariable.notify_all();
        threadsWaiting.store(0);
    }
    else
    {
        *isNotWaiting = false;
        waitVariable.wait(lock, [&]{ return *isNotWaiting; });
    }
}

class AdvancedMatrix
{
public:
    AdvancedMatrix(int nWidth, int nThreads);
    ~AdvancedMatrix();
    void printMatrix();
    void printVector();
    void startGauss();
    int getThreadsNumber() { return threadsNumber; }
private:
    enum direction
    {
        forward,
        backward
    };

    void gauss();
    void divisionString(int numDivisionString, int threadID, direction
direct);
    void subtractionString(int numTermString, int numSubtrahendString);
    void divisionDiagonal(int row);

    int ind(int i, int j) const { return i*width + j; }

    const int width;
    double* p_matrix;
    double* p_vector;
    int threadsNumber;
    int threadsIDs;
    barrier myBarrier;
};

```

```

AdvancedMatrix::AdvancedMatrix(int nWidth , int nThreads) : width(nWidth),
                                                             p_matrix(new double[nWidth *
nWidth]),
                                                             p_vector(new double[nWidth]),
                                                             threadsNumber(nThreads),
                                                             threadsIDs(0),
                                                             myBarrier(nThreads)
{
    srand(time(NULL));
    double sumElements;
    for (int i = 0; i < width; ++i)
    {
        sumElements = 0;
        for (int j = 0; j < width; ++j)
        {
            p_matrix[ind(i, j)] = (double)rand() / RAND_MAX*2 - 1;
            sumElements += p_matrix[ind(i, j)];
        }
        p_vector[i] = sumElements;
    }
}

AdvancedMatrix::~AdvancedMatrix()
{
    delete[] p_matrix;
    delete[] p_vector;
}

void AdvancedMatrix::printMatrix()
{
    for (int i = 0; i < width*width; ++i){
        cout << setw(7) << fixed << setprecision(3) << p_matrix[i];
        if ((i + 1)%width == 0) {
            cout << " |" << setw(7) << fixed << setprecision(3) <<
p_vector[i/width] << endl;
        }
        cout << endl;
    }
}

void AdvancedMatrix::printVector()
{
    for (int i = 0; i < width; ++i){
        cout << "x" << i << " = " << fixed << setprecision(3) << p_vector[i]
<< endl;
    }
}

void AdvancedMatrix::startGauss()
{
    vector<thread> threads;
    for (int i = 0; i < getThreadsNumber(); ++i){
        threads.push_back(thread([&](){ gauss(); }));
    }
    for (auto& th : threads){
        th.join();
    }
}

```

```

void AdvancedMatrix::gauss()
{
    if (threadsIDs == threadsNumber)
    {
        cout << "Количество потоков, вызвавших метод Гаусса превышает  
допустимое число, барьер не выдержит." << endl;
        return;
    }
    int threadID = threadsIDs++;

    //===== ПРЯМОЙ ХОД =====
    if (threadID == 0){
        cout << "Прямой ход:" << endl;
    }
    for (int j = 0; j < width; ++j){
        divisionString(j, threadID, forward);
        myBarrier.wait();
        for (int i = j + 1; i < width; ++i){
            if(i % threadsNumber == threadID){
                subtractionString(i, j);
            }
        }
        myBarrier.wait();

        //Печать матрицы
        if (threadID == 0){
            printMatrix();
        }
        myBarrier.wait();
    }

    //===== ОБРАТНЫЙ ХОД =====

    if (threadID == 0){
        cout << "Обратный ход:" << endl;
    }
    for (int j = width - 1; j > 0; --j){
        for (int i = j - 1; i >= 0; --i){
            if(i % threadsNumber == threadID){
                subtractionString(i, j);
            }
        }
        myBarrier.wait();

        //Печать матрицы
        if (threadID == 0){
            printMatrix();
        }
        myBarrier.wait();
    }
}

```

```

void AdvancedMatrix::divisionString(int numDivisionString, int threadID,
direction direct)
{
    double weight = p_matrix[ind(numDivisionString, numDivisionString)];
    myBarrier.wait();
    if (threadID == 0){
        p_vector[numDivisionString] /= weight;
    }
    if (direct == forward)
    {
        for (int i = numDivisionString; i < width; ++i){
            if (i % threadsNumber == threadID){
                p_matrix[ind(numDivisionString, i)] /= weight;
            }
        }
    }
    else
    {
        for (int i = numDivisionString; i >= 0; --i){
            if (i % threadsNumber == threadID){
                p_matrix[ind(numDivisionString, i)] /= weight;
            }
        }
    }
}

void AdvancedMatrix::subtractionString(int numTermString, int
numSubtrahendString)
{
    double weight = p_matrix[ind(numTermString, numSubtrahendString)];
    for (int i = numSubtrahendString; i < width; ++i){
        p_matrix[ind(numTermString, i)] -= p_matrix[ind(numSubtrahendString,
i)] * weight;
    }
    p_vector[numTermString] -= p_vector[numSubtrahendString] * weight;
}

```

```

int main(int argc, char *argv[])
{
    if (argc != 3)
    {
        cout << "    Аргументы командной строки:" << endl;
        cout << "1. Размерность генерируемой матрицы." << endl;
        cout << "2. Количество потоков." << endl;
        return 1;
    }

    int widthMatrix = atoi(argv[1]);
    int threadsNumber = atoi(argv[2]);

    AdvancedMatrix matrix(widthMatrix, threadsNumber);
    cout << "Сгенерированная матрица:" << endl;
    matrix.printMatrix();

    auto start_time = chrono::steady_clock::now();

    matrix.startGauss();

    auto end_time = chrono::steady_clock::now();
    auto elapsed_us = chrono::duration_cast<chrono::nanoseconds>(end_time -
start_time);

    cout << "Ответ: " << endl;
    matrix.printVector();

    cout << "Время работы алгоритма: " << (double) elapsed_us.count() /
1000000000 << endl;

    return 0;
}

```