 МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

УНИВЕРСИТЕТ

Институт №8

«Компьютерные науки и прикладная математика» Кафедра №806

«Вычислительная математика и программирование»

Отчет по лабораторной работе № 2,

по учебной дисциплине

«Параллельные и распределенные вычисления»

Выполнил:

Студент 1-го курса

Гр. М80-114M-22

Гордеев Н. М.

.

(подпись, дата) .

Принял:

Семенов С. А.

.

(подпись, дата) .

Москва 2022

Оглавление

[1. Постановка задачи 2](#_Toc122003163)

[2. Описание метода 2](#_Toc122003164)

[3. Результат работы программы 4](#_Toc122003165)

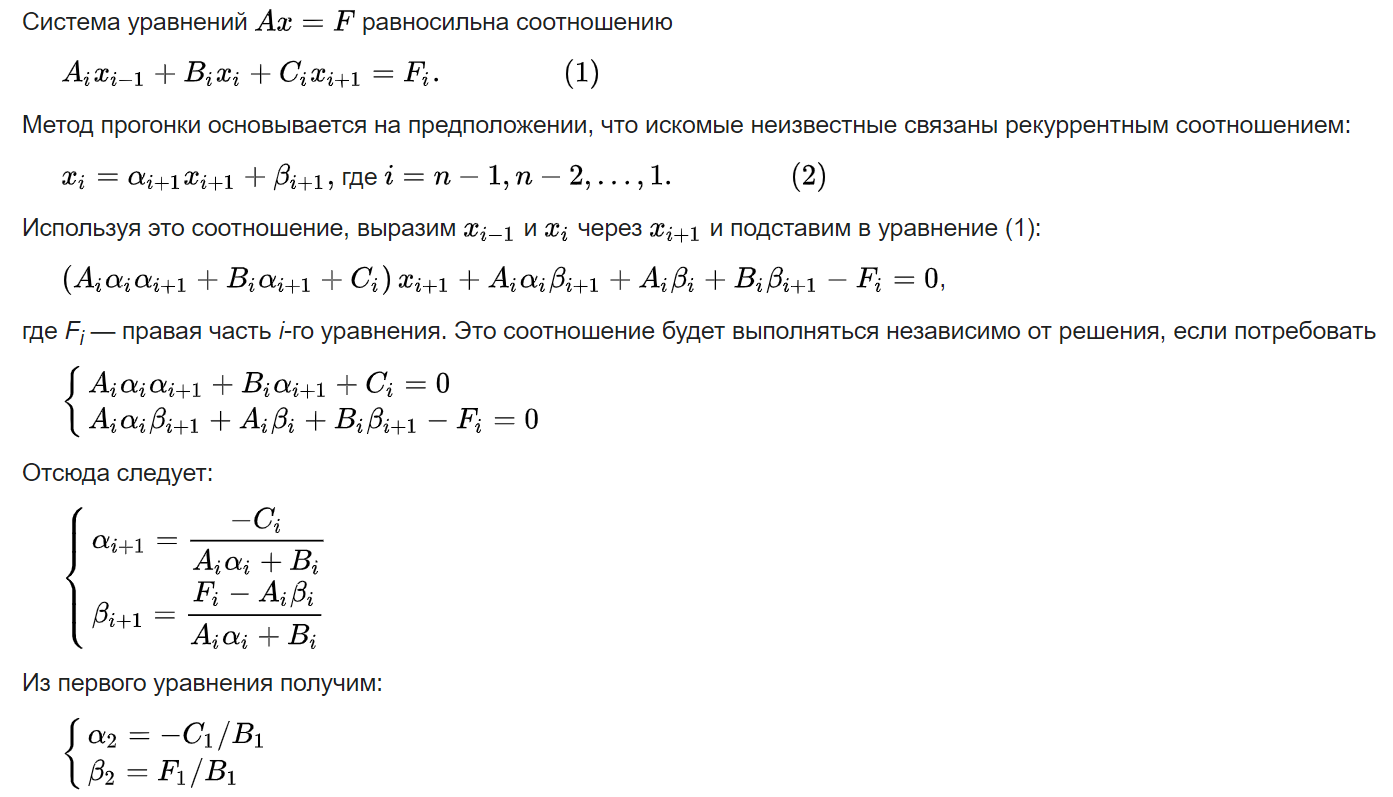
[4. Вывод: 5](#_Toc122003166)

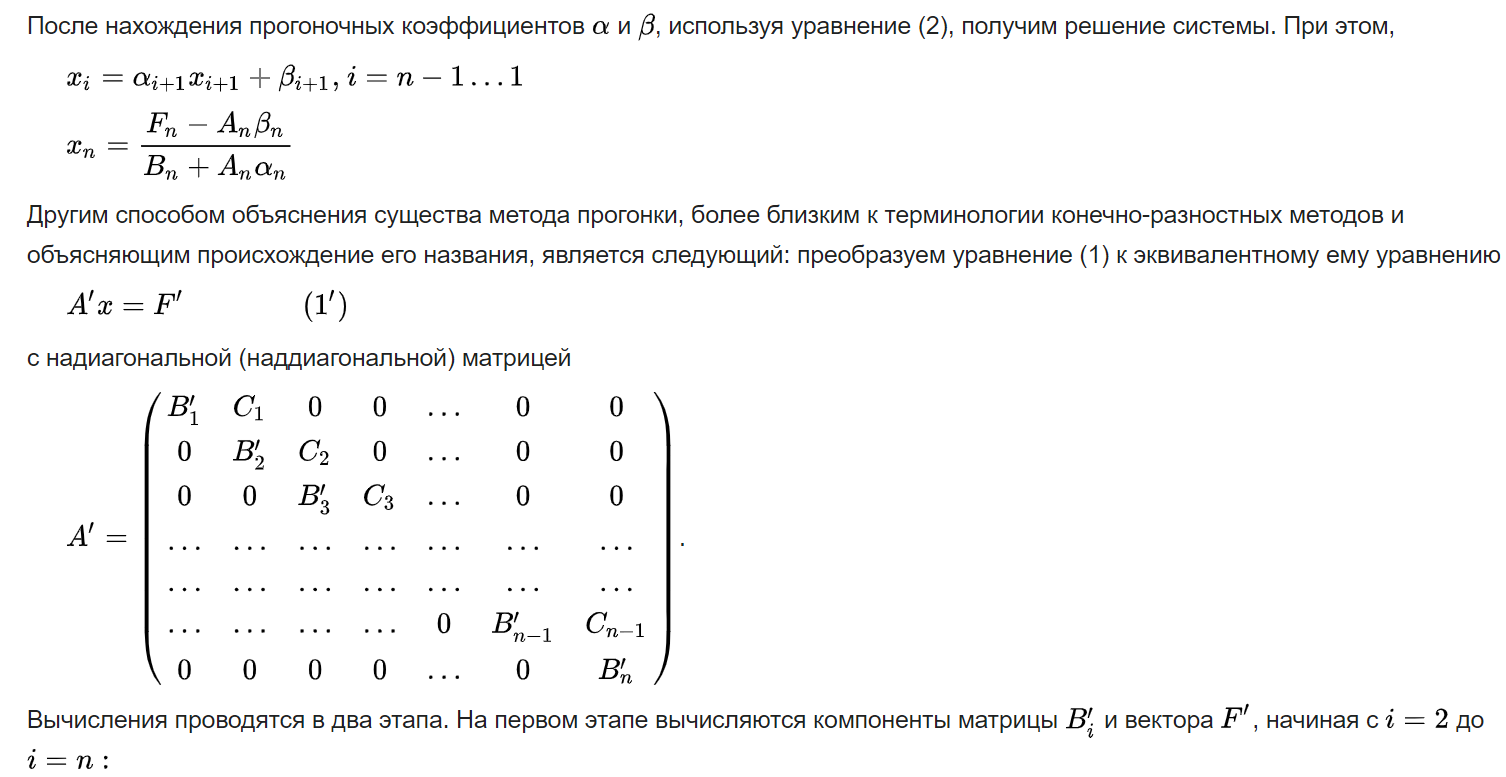
[5. Код программы 6](#_Toc122003167)

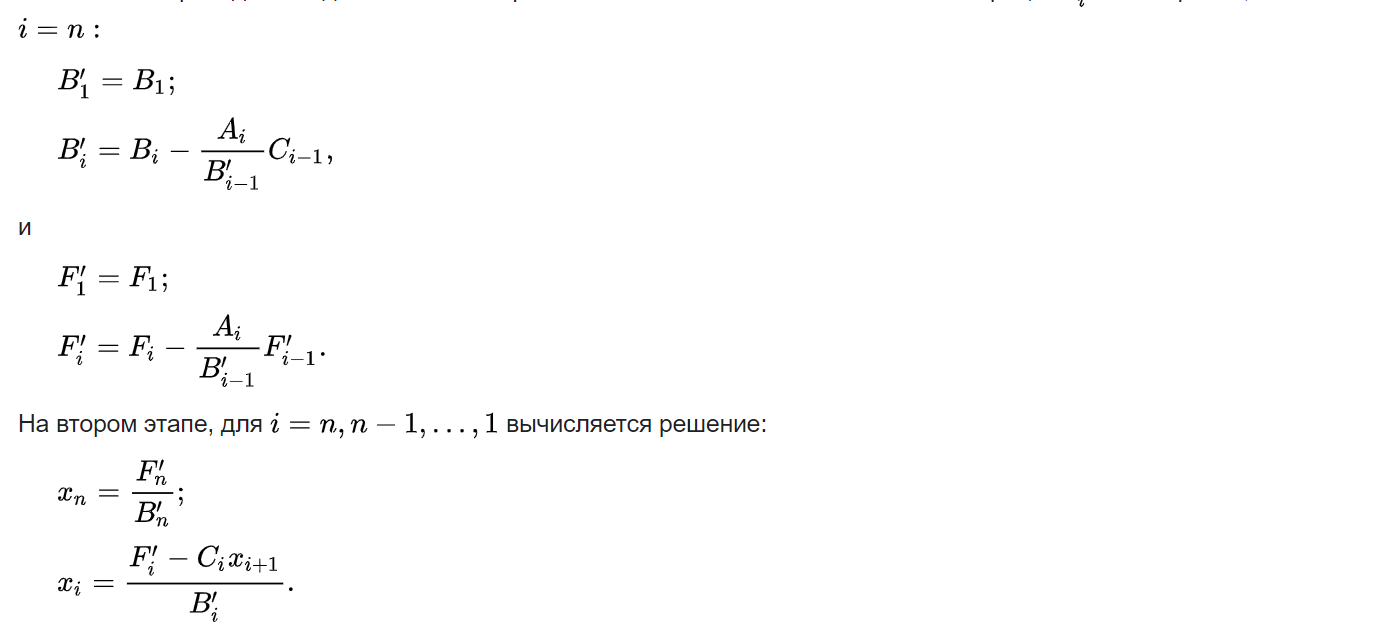
# 1. Постановка задачи

Вариант № 5. Записать метод прогонки

# 2. Описание метода



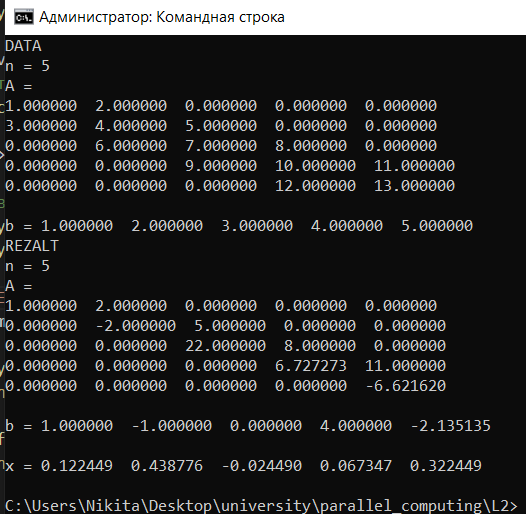




Такая схема вычисления объясняет также английский термин этого метода «shuttle».

Для применимости формул метода прогонки достаточно свойства диагонального преобладания у матрицы *A*.

# 3. Результат работы программы



DATA

n = 5

A =

1.000000 2.000000 0.000000 0.000000 0.000000

3.000000 4.000000 5.000000 0.000000 0.000000

0.000000 6.000000 7.000000 8.000000 0.000000

0.000000 0.000000 9.000000 10.000000 11.000000

0.000000 0.000000 0.000000 12.000000 13.000000

b = 1.000000 2.000000 3.000000 4.000000 5.000000

REZALT

n = 5

A =

1.000000 2.000000 0.000000 0.000000 0.000000

0.000000 -2.000000 5.000000 0.000000 0.000000

0.000000 0.000000 22.000000 8.000000 0.000000

0.000000 0.000000 0.000000 6.727273 11.000000

0.000000 0.000000 0.000000 0.000000 -6.621620

b = 1.000000 -1.000000 0.000000 4.000000 -2.135135

x = 0.122449 0.438776 -0.024490 0.067347 0.322449

# 4. Вывод:

Познакомился с основами программирования на cuda. Метод прогонки по этому алгоритму не параллелится, т.к. он требует последовательных операций что на 1 этапе, что на втором.

# 5. Код программы

#include "cuda\_runtime.h"      // CUDA runtime

#include "device\_launch\_parameters.h"

#include <stdio.h>

#include <stdlib.h>

#define N 80

// Фуккция, которая выполняется на GPU

\_\_global\_\_ void Go(float \*mass,float \*b,float \*x,int n){

    for (int i = 1; i < n; i++){

        mass[i\*N + i] = mass[i\*N + i] - mass[i\*N + i - 1] \* mass[(i-1)\*N + i] / mass[(i-1)\*N + i - 1];

        b[i] = b[i] - b[i-1] \* mass[(i)\*N + i - 1] / mass[(i-1)\*N + i - 1];

        mass[(i)\*N + i-1] = 0;

    }

    x[n-1] = b[n-1] / mass[(n-1)\*N + n - 1];

    for (int i = n - 2; i >= 0; i--){

        x[i] = (b[i] - x[i+1] \* mass[(i)\*N + i + 1]) / mass[i\*N + i];

    }

}

void PrintMss(float mass[][N],float \*b,int n){

    printf("n = %d\n",n);

    printf("A = \n");

    for(int i = 0; i < n; i++)

    {

        for(int j = 0; j<n; j++)

            printf("%f  ",mass[i][j]);

        printf("\n");

    }

    printf("\nb = ");

    for(int i = 0; i < n; i++)

        printf("%f  ",b[i]);

    printf("\n");

}

int main(){

    int n;

    FILE \*f;

    float mass[N][N];

    float b[N];

    size\_t pitch;

    f = fopen("data.txt","r");

    fscanf(f,"%d",&n);

    for(int i = 0; i < n; i++)

    {

        for(int j = 0; j<n; j++)

            fscanf(f,"%f",&mass[i][j]);

    }

    for(int i = 0; i < n; i++)

        fscanf(f,"%f",&b[i]);

    fclose(f);

    printf("DATA\n");

    PrintMss(mass,b,n);

    float \*dev\_mass; // Адрес масива на GPU

    // Выделить память на GPU и сохранить её адрес в переменную

    cudaMalloc( (void\*\*)&dev\_mass, N \* N \* sizeof(float));

    // Копировать данные по адресу mass, размером N\*sizeof(float) в адрес dev\_mass. Копирование с устройства на GPU

    cudaMemcpy(dev\_mass, mass, N \* N \* sizeof(float), cudaMemcpyHostToDevice);

    float \*dev\_b; // Адрес масива на GPU

    // Выделить память на GPU и сохранить её адрес в переменную

    cudaMalloc( (void\*\*)&dev\_b, N \* sizeof(float));

    // Копировать данные по адресу mass, размером N\*sizeof(float) в адрес dev\_mass. Копирование с устройства на GPU

    cudaMemcpy(dev\_b, b, N \* sizeof(float), cudaMemcpyHostToDevice);

    float \*dev\_x; // Адрес масива на GPU

    // Выделить память на GPU и сохранить её адрес в переменную

    cudaMalloc( (void\*\*)&dev\_x, N \* sizeof(float));

    Go<<<1,1>>> (dev\_mass,dev\_b,dev\_x,n);

    // Копировать данные по адресу dev\_mass, размером N\*sizeof(int) в адрес mass\_GPU. Копирование с GPU на устройство

    cudaMemcpy(mass, dev\_mass, N \* N \* sizeof(int), cudaMemcpyDeviceToHost);

    cudaMemcpy(b, dev\_b, N \* sizeof(int), cudaMemcpyDeviceToHost);

    printf("REZALT\n");

    PrintMss(mass,b,n);

    cudaMemcpy(b, dev\_x, N \* sizeof(int), cudaMemcpyDeviceToHost);

    printf("\nx = ");

    for(int i = 0; i < n; i++)

        printf("%f  ",b[i]);

    printf("\n");

    return 0;

}