Завдання

1. Розпаралелити метод Гаусса використовуючи omp_set_num_threads() та parallel.

2. Порівняти швидкодію програми для систем різної розмірності та при різній кількості робочих процесів.

Реалізація (опис)

Приклад реалізації зведення матриці до трикутного вигляду, запропонований в завданні до лабораторної роботи, є вразливим до появи елементів матриці, рівних нулю, під час зведення матриці до трикутного вигляду. Операція ділення елементів поточного рядка на елемент головної діагоналі виконується без перевірки на нуль:

```
omp_set_num_threads(THREADS);
for (i = 0; i < n; i++) {
	tmp = matrix[i][i];
	for (j = n; j >= i; j--)
		matrix[i][j] /= tmp; <——— ділення виконується без перевірки tmp на рівністю нулю.

	#pragma omp parallel for private (j, k, tmp)
	for (j = i + 1; j < n; j++) {
		tmp = matrix[j][i];
		for (k = n; k >= i; k--)
			matrix[j][k] -= tmp * matrix[i][k];
	}
}
```

В роботі було реалізовано модифікований алгоритм, який усуває можливість появи помилки ділення на нуль.
Суть модифікації:
Якщо matrix[i][i] = 0, то виконується пошук рядка із ненульовим елементом matrix[j][i], де i < j < n.
Якщо такий рядок знайдено, виконується перестановка i-того та j-того рядків матриці.
Відсутність такого рядка свідчить, що починаючи з i, всі елементи СТОВПЦЯ дорівнюють нулю. В такому випадку алгоритм переходить до наступного рядка / стовпця (інкрементує i).
Для більшого розуміння алгоритму див. функції echelon_form_1t, echelon_form_mt, swap_rows_ith_nonzero у файлі gausslib.c.

Тестування коректності алгоритму здійснювалось на 2 прикладах.
Приклад 1:

1	2	3	4	0
2	14	20	27	0
5	10	16	19	-2
3	5	6	13	5

Приклад 2:

2	-1	-1	-4	-1	2
-1	2	-1	-1	-1	0
4	1	-5	-8	-5	1
1	1	2	1	1	0
1	1	1	2	1	0

Демонстрація коректного виконання прикладів:

```
PS J:\repos\Parallel computing\Gauss\x64\Release> .\GaussDemo.exe v
| 1.000000 | -1.000000 | -1.000000 | 1.000000 |
| 0.277778 | 0.444444 | -1.333333 | -0.833333 | 2.777778 |
| 1.000000 | -1.000000 | -1.000000 | 1.000000 |
| 0.277778 | 0.444444 | -1.333333 | -0.833333 | 2.777778 |
PS J:\repos\Parallel computing\Gauss\x64\Release>
```

Реалізація (дослідження швидкодії)

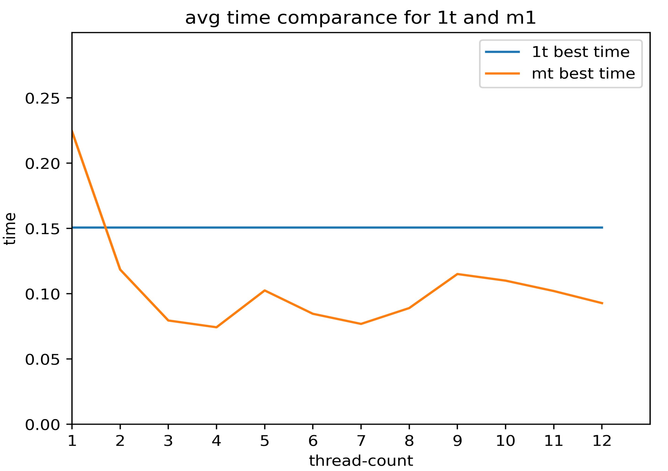
Тестування швидкодії однопоточної версії та багатопоточної версії здійснювалось на матриці розмірності 850 (850 невідомих змінних). Матриця була згенерована випадково за допомогою команди g:

```
PS J:\repos\Parallel computing\Gauss\x64\Release> .\GaussDemo.exe g 850 100 m850
PS J:\repos\Parallel computing\Gauss\x64\Release>
```

Кількість потоків змінювалась в межах [1; 12].

```
PS J:\repos\Parallel computing\Gauss\x64\Release> .\GaussDemo s m850 0
run 0: time = 0.150677;
run 1: time = 0.150810;
run 2: time = 0.151373;
run 3: time = 0.151698;
run 4: time = 0.157368;
Best time: 0.150677
PS J:\repos\Parallel computing\Gauss\x64\Release>
```

```
PS J:\repos\Parallel computing\Gauss\x64\Release> .\GaussDemo m m850 0 1
run 0: time = 0.225175;
run 1: time = 0.228297;
run 2: time = 0.231656;
run 3: time = 0.224388;
run 4: time = 0.238884;
Best time: 0.224388
PS J:\repos\Parallel computing\Gauss\x64\Release> .\GaussDemo m m850 0 2
run 0: time = 0.131149;
run 1: time = 0.118467;
run 2: time = 0.121058;
run 3: time = 0.167590;
run 4: time = 0.146837;
Best time: 0.118467
PS J:\repos\Parallel computing\Gauss\x64\Release> .\GaussDemo m m850 0 3
run 0: time = 0.079450;
run 1: time = 0.106476;
run 2: time = 0.115503;
run 3: time = 0.113387;
run 4: time = 0.100975;
Best time: 0.079450
PS J:\repos\Parallel computing\Gauss\x64\Release> .\GaussDemo m m850 0 4
run 0: time = 0.102887;
run 1: time = 0.112713;
run 2: time = 0.106294;
run 3: time = 0.091729;
run 4: time = 0.074265;
Best time: 0.074265
PS J:\repos\Parallel computing\Gauss\x64\Release> .\GaussDemo m m850 0 5
run 0: time = 0.108790;
run 1: time = 0.102810;
run 2: time = 0.104018;
run 3: time = 0.103671;
run 4: time = 0.102465;
Best time: 0.102465
PS J:\repos\Parallel computing\Gauss\x64\Release> .\GaussDemo m m850 0 6
run 0: time = 0.084585;
run 1: time = 0.089648;
run 2: time = 0.089170;
run 3: time = 0.098551;
run 4: time = 0.087591;
Best time: 0.084585
PS J:\repos\Parallel computing\Gauss\x64\Release> .\GaussDemo m m850 0 7
run 0: time = 0.076840;
run 1: time = 0.085904;
run 2: time = 0.081657;
run 3: time = 0.082675;
run 4: time = 0.083689;
Best time: 0.076840
PS J:\repos\Parallel computing\Gauss\x64\Release> .\GaussDemo m m850 0 8
run 0: time = 0.088972;
run 1: time = 0.090789;
run 2: time = 0.096157;
run 3: time = 0.092157;
run 4: time = 0.089428;
Best time: 0.088972
PS J:\repos\Parallel computing\Gauss\x64\Release> .\GaussDemo m m850 0 9
run 0: time = 0.115048;
run 1: time = 0.121680;
run 2: time = 0.123619;
run 3: time = 0.122882;
run 4: time = 0.128152;
Best time: 0.115048
PS J:\repos\Parallel computing\Gauss\x64\Release> .\GaussDemo m m850 0 10
run 0: time = 0.131889;
run 1: time = 0.140220;
run 2: time = 0.109975;
run 3: time = 0.114773;
run 4: time = 0.116979;
Best time: 0.109975
PS J:\repos\Parallel computing\Gauss\x64\Release> .\GaussDemo m m850 0 11
run 0: time = 0.102024;
run 1: time = 0.106259;
run 2: time = 0.112048;
run 3: time = 0.106259;
run 4: time = 0.104651;
Best time: 0.102024
PS J:\repos\Parallel computing\Gauss\x64\Release> .\GaussDemo m m850 0 12
run 0: time = 0.092764;
run 1: time = 0.112392;
run 2: time = 0.098318;
run 3: time = 0.099276;
run 4: time = 0.100591;
Best time: 0.092764
PS J:\repos\Parallel computing\Gauss\x64\Release>
```



Висновки

На відрізку n = [1; 4] бачимо приріст, приблизно кратний кількості потоків.
Найкращий час було досягнути при кількості потоків, рівній чотирьом.
Починаючи з n = 4, бачимо відсутність приросту продуктивності.

Однією з причин відсутності приросту при кількості потоків > 4 може бути нерозпаралелена функція `swap_rows_ith_nonzero`. Потенційно цю функцію можна розпаралелити приблизно так:

```
static int swap_rows_ith_nonzero(int n, int i, double ** matrix)
{
	int nonzero_flag = 0;
	int j;
	#pragma omp parallel for
	for (j = i + 1; j < n; j++) // row cycle
	{
		// Check if element is non zero
		if (matrix[j][i] != 0)
		{
			// Non zero element is found, perform swap, set flag
			#pragma omp cancel for
			nonzero_flag = 1;
			double * tmp = matrix[i];
			matrix[i] = matrix[j];
			matrix[j] = tmp;
			break;
		}
	}
	return nonzero_flag;
}
```

Ця версія не була протестована в роботі, оскільки MSVC v143 не підтримує конструкцію cancel.

main.c

```c
#include "gausslib.h"

#include <omp.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <float.h>
#include <string.h>

#define N1 4
#define N2 5

#define EPSILON 0.0001
#define RUN_COUNT 5

#define USAGE "Usage: [v]alidate | [g]enerate n:int r:int outname:str | [p]rint inputname:str | [s]ingle-thread inputname:str is_verbose:int | [m]ulti-thread inputname:str
is_verbose:int thread_count:int"

static void validate1(void(*echelon_form)(int, double **));
static void validate2(void(*echelon_form)(int, double **));
static void validate(void(*echelon_form)(int, double **),
                     int n, double ** matrix, double * x, double * expected_x);


static void performance(void(*echelon_form)(int, double **),
                        int n, double ** original_matrix, double ** copy_matrix, double * x, int is_verbose);


static void matrix_generate(int n, int r, FILE * f);
static void matrix_read(int * n, double *** matrix, FILE * f);

static double ** matrix_malloc(int n);
static void matrix_print(int n, double ** matrix);
static void matrix_free(int n, double ** matrix);

int main(int argc, char ** argv)
{
        if (argc < 2)
        {
                puts(USAGE);
                return EXIT_SUCCESS;
        }

        int n, is_verbose;
        FILE * f = NULL;
        double ** matrix; double * x; double ** copy_matrix;
        switch (argv[1][0])
        {
                case 'v':

                        validate1(echelon_form_lt);
                        putchar('\n');
                        validate2(echelon_form_lt);
                        putchar('\n');
                        validate1(echelon_form_mt);
                        putchar('\n');
                        validate2(echelon_form_mt);

                        break;

                case 'g':

                        if (argc != 5)
                        {
                                puts(USAGE);
                                return EXIT_SUCCESS;
                        }

                        n = atoi(argv[2]);
                        if (n <= 0)
                        {
                                printf("Invalid matrix dimension: %s\n", argv[2]);
                                return EXIT_SUCCESS;
                        }

                        int r = atoi(argv[3]);
                        if (r <= 0)
                        {
                                printf("Invalid matrix dimension: %s\n", argv[3]);
                                return EXIT_SUCCESS;
                        }

                        fopen_s(&f, argv[4], "w");
                        if (f == NULL)
                        {
                                printf("Can't open file: %s\n", argv[4]);
                                return EXIT_SUCCESS;
                        }

                        matrix_generate(n, r, f);

                        fflush(f);
                        fclose(f);

                        break;

                case 'p':

                        if (argc != 3)
                        {
                                puts(USAGE);
                                return EXIT_SUCCESS;
                        }

                        fopen_s(&f, argv[2], "r");
                        if (f == NULL)
                        {
                                printf("Can't open file: %s\n", argv[2]);
                                return EXIT_SUCCESS;
                        }

                        matrix_read(&n, &matrix, f);
                        matrix_print(n, matrix);
                        matrix_free(n, matrix);

                        break;

                case 's':

                        if (argc != 4)
                        {
                                puts(USAGE);
                                return EXIT_SUCCESS;
                        }

                        fopen_s(&f, argv[2], "r");
                        if (f == NULL)
                        {
                                printf("Can't open file: %s\n", argv[2]);
                                return EXIT_SUCCESS;
                        }

                        is_verbose = atoi(argv[3]);
                        if (is_verbose < 0)
                        {
                                printf("Invalid verbose flag: %s\n", argv[3]);
                                return EXIT_SUCCESS;
                        }

                        matrix_read(&n, &matrix, f);
                        x = (double *)malloc(n * sizeof(double));
                        copy_matrix = matrix_malloc(n);

                        performance(echelon_form_lt, n, matrix, copy_matrix, x, is_verbose);

                        free(x);
                        matrix_free(n, matrix);
                        matrix_free(n, copy_matrix);

                        break;

                case 'm':

                        if (argc != 5)
                        {
                                puts(USAGE);
                                return EXIT_SUCCESS;
                        }

                        fopen_s(&f, argv[2], "r");
                        if (f == NULL)
                        {
                                printf("Can't open file: %s\n", argv[2]);
                                return EXIT_SUCCESS;
                        }

                        is_verbose = atoi(argv[3]);
                        if (is_verbose < 0)
                        {
                                printf("Invalid verbose flag: %s\n", argv[3]);
                                return EXIT_SUCCESS;
                        }

                        int thread_count = atoi(argv[4]);
                        if (thread_count <= 0)
                        {
                                printf("Invalid thread count: %s\n", argv[4]);
                                return EXIT_SUCCESS;
                        }

                        matrix_read(&n, &matrix, f);
                        x = (double *)malloc(n * sizeof(double));
                        copy_matrix = matrix_malloc(n);

                        omp_set_num_threads(thread_count);
                        performance(echelon_form_mt, n, matrix, copy_matrix, x, is_verbose);

                        free(x);
                        matrix_free(n, matrix);
                        matrix_free(n, copy_matrix);

                        break;

                default:
                        puts(USAGE);
                        break;
        }

        return EXIT_SUCCESS;
}

static void validate1(void(*echelon_form)(int, double **))
{
        double ** matrix = matrix_malloc(N1);
        matrix[0][0] = 1; matrix[0][1] = 2; matrix[0][2] = 3; matrix[0][3] = 4; matrix[0][4] = 0;
        matrix[1][0] = 7; matrix[1][1] = 16; matrix[1][2] = 20; matrix[1][3] = 27; matrix[1][4] = 0;
        matrix[2][0] = 5; matrix[2][1] = 10; matrix[2][2] = 16; matrix[2][3] = 19; matrix[2][4] = -2;
        matrix[3][0] = 3; matrix[3][1] = 5; matrix[3][2] = 6; matrix[3][3] = 13; matrix[3][4] = 5;
        double x[N1];
        double expected_x[N1] = { 1, -1, -1, 1 };

        validate(echelon_form, N1, matrix, x, expected_x);

        matrix_free(N1, matrix);
}

static void validate2(void(*echelon_form)(int, double **))
{
        double ** matrix = matrix_malloc(N2);
        matrix[0][0] = 2; matrix[0][1] = -1; matrix[0][2] = -1; matrix[0][3] = -4; matrix[0][4] = -1; matrix[0][5] = 2;
        matrix[1][0] = -1; matrix[1][1] = 2; matrix[1][2] = -1; matrix[1][3] = -1; matrix[1][4] = -1; matrix[1][5] = 0;
        matrix[2][0] = 4; matrix[2][1] = 1; matrix[2][2] = -5; matrix[2][3] = -8; matrix[2][4] = -5; matrix[2][5] = 1;
        matrix[3][0] = 1; matrix[3][1] = 1; matrix[3][2] = 2; matrix[3][3] = 1; matrix[3][4] = 1; matrix[3][5] = 0;
        matrix[4][0] = 1; matrix[4][1] = 1; matrix[4][2] = 1; matrix[4][3] = 2; matrix[4][4] = 1; matrix[4][5] = 0.5;
        double x[N2];
        double expected_x[N2] = { 5.0/18.0, 4.0/9.0, (-4.0)/3.0, (-5.0)/6.0, 25.0/9.0 };

        validate(echelon_form, N2, matrix, x, expected_x);

        matrix_free(N2, matrix);
}

static void validate(void(*echelon_form)(int, double **),
                     int n, double ** matrix, double * x, double * expected_x)

{
        echelon_form(n, matrix);
        back_substitution(n, matrix, x);
        for (int i = 0; i < n; i++)
        {
                printf("| %lf ", x[i]);
                if (fabs(x[i] - expected_x[i]) > EPSILON)
                {
                        printf("Assertion failed at %d", i);
                        break;
                }
        }
        putchar('|');
}

static void performance(void(*echelon_form)(int, double **),
                        int n, double ** original_matrix, double ** copy_matrix, double * x, int is_verbose)

{
        double s_time, e_time, time, min_time = DBL_MAX;
        for (int i = 0; i < RUN_COUNT; i++)
        {
                for (int j = 0; j < n; j++)
                        memcpy_s(copy_matrix[j], (n + 1) * sizeof(double), original_matrix[j], (n + 1) * sizeof(double));

                s_time = omp_get_wtime();
                echelon_form(n, copy_matrix);
                back_substitution(n, copy_matrix, x);
                e_time = omp_get_wtime();
                time = e_time - s_time;

                printf("run %d: time = %lf;\n", i, time);

                if (is_verbose)
                {
                        for (int j = 0; j < n; j++)
                                printf("| %lf ", x[j]);
                        putchar('\n');
                }

                if (min_time > time)
                        min_time = time;
        }
        printf("Best time: %lf\n", min_time);
}

static void matrix_generate(int n, int r, FILE * f)
{
        fwrite(&n, sizeof(int), 1, f);

        double d;
        for (int i = 0; i < n * (n + 1); i++)
        {
                d = rand() % r;
                fwrite(&d, sizeof(double), 1, f);
        }
}

static void matrix_read(int * n, double *** matrix, FILE * f)
{
        int _n;
        fread(&_n, sizeof(int), 1, f);

        double ** _matrix = matrix_malloc(_n);

        for (int i = 0; i < _n; i++)
        {
                fread(_matrix[i], sizeof(double), _n + 1, f);
        }

        *n = _n;
        *matrix = _matrix;
}

static double ** matrix_malloc(int n)
{
        double ** m = (double **)malloc(n * sizeof(double *));
        for (int i = 0; i < n; i++)
        {
                m[i] = (double *)malloc((n + 1) * sizeof(double));
        }
        return m;
}

static void matrix_print(int n, double ** matrix)
{
        for (int i = 0; i < n; i++)
        {
                for (int j = 0; j < n + 1; j++)
                {
                        printf("| %lf ", matrix[i][j]);
                }
                puts("|\n");
        }
}

static void matrix_free(int n, double ** matrix)
{
        for (int i = 0; i < n; i++)
        {
                free(matrix[i]);
        }
        free(matrix);
}
```

Реалізація (код)

Посилання на гітхаб репозиторій: https://github.com/Bohdan628318ylypchenko/parallel-programming-lab4.git

gausslib.h

```c
#pragma once

/// <summary>
/// Single thread row echelon form implementation.
/// Mutates augmented matrix (NxN+1) into row echelon form.
/// </summary>
/// <param name="n"> Augmented matrix row count. </param>
/// <param name="matrix"> Augmented matrix. </param>
void echelon_form_lt(int n, double ** matrix);

/// <summary>
/// Multi thread row echelon form implementation.
/// Mutates augmented matrix (NxN+1) into row echelon form.
/// </summary>
/// <param name="n"> Augmented matrix row count. </param>
/// <param name="matrix"> Augmented matrix. </param>
void echelon_form_mt(int n, double ** matrix);

/// <summary>
/// Runs back substitution algorithm on matrix.
/// </summary>
/// <param name="n"> Augmented matrix row count. </param>
/// <param name="matrix"> Augmented matrix. </param>
/// <param name="x"> Array to save solutions in. </param>
void back_substitution(int n, double ** matrix, double * x);
```

gausslib.c

```c
#include "pch.h"

#include "gausslib.h"

#include <omp.h>

static int swap_rows_ith_nonzero(int n, int i, double ** matrix);

/// <summary>
/// Single thread row echelon form implementation.
/// Mutates augmented matrix (NxN+1) into row echelon form.
/// </summary>
/// <param name="n"> Augmented matrix row count. </param>
/// <param name="matrix"> Augmented matrix. </param>
void echelon_form_lt(int n, double ** matrix)
{
        double c;
        for (int i = 0; i < n; i++) // row cycle
        {
                // Check for zero division
                if (matrix[i][i] == 0)
                        if (swap_rows_ith_nonzero(n, i, matrix) == 0) continue;

                // Coefficient to divide ith row with
                c = matrix[i][i];

                // Divide current row
                for (int j = n; j >= i; j--) // element cycle
                {
                        matrix[i][j] /= c;
                }

                // Subtract ith row from all below
                for (int j = i + 1; j < n; j++) // row cycle
                {
                        // Coefficient to multiply kth row with
                        c = matrix[j][i];

                        for (int k = n; k >= i; k--) // element cycle
                        {
                                matrix[j][k] -= c * matrix[i][k];
                        }
                }
        }
}

/// <summary>
/// Multi thread row echelon form implementation.
/// Mutates augmented matrix (NxN+1) into row echelon form.
/// </summary>
/// <param name="n"> Augmented matrix row count. </param>
/// <param name="matrix"> Augmented matrix. </param>
void echelon_form_mt(int n, double ** matrix)
{
        int i, j;
        for (i = 0; i < n; i++) // row cycle
        {
                // Check for zero division
                if (matrix[i][i] == 0)
                        if (swap_rows_ith_nonzero(n, i, matrix) == 0) continue;

                // Coefficient to divide ith row with
                double c = matrix[i][i];

                // Divide current row
                for (j = n; j >= i; j--) // element cycle
                {
                        matrix[i][j] /= c;
                }

                // Subtract ith row from all below
                #pragma omp parallel for
                for (j = i + 1; j < n; j++)
                {
                        // Coefficient to multiply kth row with
                        double d = matrix[j][i];

                        for (int k = n; k >= i; k--) // element cycle
                        {
                                matrix[j][k] -= d * matrix[i][k];
                        }
                }
        }
}

/// <summary>
/// Runs back substitution algorithm on matrix.
/// </summary>
/// <param name="n"> Augmented matrix row count. </param>
/// <param name="matrix"> Augmented matrix. </param>
/// <param name="x"> Array to save solutions in. </param>
void back_substitution(int n, double ** matrix, double * x)
{
        x[n - 1] = matrix[n - 1][n];
        for (int i = n - 2; i >= 0; i--) // row cycle
        {
                x[i] = matrix[i][n];

                int j;
                for (j = i + 1; j < n; j++)
                {
                        x[i] -= matrix[i][j] * x[j];
                }
        }
}

/// <summary>
/// Attempts to find fst row below ith, where e[j][i] != 0.
/// If e[j][i] != 0 row exists, swaps ith row with jth row, returns 1;
/// Else returns 0;
/// </summary>
/// <param name="n"> Matrix row count. </param>
/// <param name="i"> Index of row to swap with. </param>
/// <param name="matrix"> Matrix of equation. </param>
/// <returns> Flag to report if swap was performed. </returns>
static int swap_rows_ith_nonzero(int n, int i, double ** matrix)
{
        int nonzero_flag = 0;
        for (int j = i + 1; j < n; j++) // row cycle
        {
                // Check if element is non zero
                if (matrix[j][i] != 0)
                {
                        // Non zero element is found, perform swap, set flag
                        nonzero_flag = 1;
                        double * tmp = matrix[i];
                        matrix[i] = matrix[j];
                        matrix[j] = tmp;
                        break;
                }
        }
        return nonzero_flag;
}
```