МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА
Факультет електроніки і комп’ютерних технологій

Звіт

Про виконання лабораторної роботи №4

“ Реалізація LU-розкладу матриці за допомогою завдань (tasks) в OpenMP програмах ”

Виконав:
Ст. групи ФеС-32
Молібожко Олександр
Перевірив:
Кулик П.Р.

Львів 2024

Мета: Реалізувати LU-розклад матриці за допомогою завдань (tasks) в OpenMP програмах

Хід роботи:

1. Написати програму(и) LU -розкладу матриці у трьох режимах роботи: 1) послідовному, 2) паралельному за допомогою директиви #pragma omp for, 3) паралельному за допомогою директиви #pragma omp task.

2. Написати функцію для заповнення матриці довільного розміру випадковими числами. Передбачити можливість вказувати діапазон випадкових значень.

3. Написати функцію для перевірки правильності LU -розкладу матриці за допомогою формули

$$a_{ik} = \sum_{j=1}^{n} l_{ij} u_{jk}$$

4. Виконати порівняння швидкодії LU -розкладу матриці для трьох режимів роботи програми у випадках, коли розмір матриці дорівнює n = 10, 100, 500, 1000 елементів.

5. Написати звіт про виконання лабораторної роботи.

Виконання роботи:

Код програми:

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <omp.h>
#include <math.h>

// Заповнює матрицю випадковими значеннями у заданому діапазоні
void fill_matrix(float a[], int n, float min_val, float max_val);

// Виконує послідовну LU-декомпозицію
void lu_decomposition_seq(float a[], int n);

// Виконує паралельну LU-декомпозицію з використанням OpenMP
void lu_decomposition_parallel(float a[], int n);

// Виконує паралельну LU-декомпозицію з використанням задач OpenMP
void lu_decomposition_tasks(float a[], int n);

// Перевіряє коректність виконання LU-декомпозиції
void check_lu_correctness(float a[], float l[], float u[], int n);

void lu_decomposition_seq(float a[], int n)
{
    int i, j, k;

    // Послідовна реалізація LU-декомпозиції
    for (i = 0; i < n; i++)
    {
        for (j = i + 1; j < n; j++)
        {
            a[j * n + i] /= a[i * n + i];

            for (k = i + 1; k < n; k++)
            {
                a[j * n + k] -= a[j * n + i] * a[i * n + k];
            }
        }
    }
}

void lu_decomposition_parallel(float a[], int n)
{
    int i, j, k;

    // Паралельна реалізація LU-декомпозиції з використанням OpenMP
    for (i = 0; i < n; i++)
    {
        #pragma omp parallel for shared(a) private(j, k)
        for (j = i + 1; j < n; j++)
        {
            a[j * n + i] /= a[i * n + i];

            for (k = i + 1; k < n; k++)
            {
                a[j * n + k] -= a[j * n + i] * a[i * n + k];
            }
        }
    }
}
```

```
{
    int i, j, k;

    // Паралельна реалізація LU-декомпозиції з використанням задач OpenMP
    for (i = 0; i < n; i++)
    {
        #pragma omp parallel shared(a) private(j, k)
        {
            #pragma omp single
            {
                for (j = i + 1; j < n; j++)
                {
                    a[j * n + i] /= a[i * n + i];
                    #pragma omp task firstprivate(j)
                    {
                        for (k = i + 1; k < n; k++)
                        {
                            a[j * n + k] -= a[j * n + i] * a[i * n + k];
                        }
                    }
                }
                #pragma omp taskwait
            }
        }
    }
}

void fill_matrix(float a[], int n, float min_val, float max_val)
{
    srand(time(NULL));
    int i, j;

    // Заповнення матриці випадковими значеннями у заданому діапазоні
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            a[i * n + j] = ((float) rand() / RAND_MAX) * (max_val - min_val) + min_val;
        }
    }
}

void check_lu_correctness(float a[], float l[], float u[], int n)
{
    int i, j, k;
    float sum;

    // Перевірка коректності LU-декомпозиції
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            sum = 0;
            for (k = 0; k < n; k++)
            {
```

```
        int n = n_vals[i];

        // Виділення пам'яті для матриць
        a = (float*) malloc(n * n * sizeof(float));
        a_copy = (float*) malloc(n * n * sizeof(float));
        l = (float*) malloc(n * n * sizeof(float));
        u = (float*) malloc(n * n * sizeof(float));

        fill_matrix(a, n, min_val, max_val);

        // Послідовне виконання
        for (j = 0; j < n * n; j++)
        {
            a_copy[j] = a[j];
        }
        start_time = omp_get_wtime();
        lu_decomposition_seq(a_copy, n);
        end_time = omp_get_wtime();
        seq_time = end_time - start_time;
        printf("Час послідовного виконання для n = %d: %f секунд\n", n, seq_time);

        // Паралельне виконання
        for (j = 0; j < n * n; j++)
        {
            a_copy[j] = a[j];
        }
        start_time = omp_get_wtime();
        lu_decomposition_parallel(a_copy, n);
        end_time = omp_get_wtime();
        parallel_time = end_time - start_time;
        printf("Час паралельного виконання для n = %d: %f секунд\n", n, parallel_time);

        // Виконання з використанням задач
        for (j = 0; j < n * n; j++)
        {
            a_copy[j] = a[j];
        }
        start_time = omp_get_wtime();
        lu_decomposition_tasks(a_copy, n);
        end_time = omp_get_wtime();
        task_time = end_time - start_time;
        printf("Час виконання з задачами для n = %d: %f секунд\n", n, task_time);

        check_lu_correctness(a, l, u, n);

        // Звільнення пам'яті
        free(a);
        free(a_copy);
        free(l);
        free(u);
    }

    return 0;
}
```

Виконання програми:

```
Час послідовного виконання для n = 10: 0.000002 секунд
Час паралельного виконання для n = 10: 0.216518 секунд
Час виконання з задачами для n = 10: 0.201945 секунд
LU-декомпозиція некоректна!
Час послідовного виконання для n = 100: 0.002331 секунд
Час паралельного виконання для n = 100: 0.091216 секунд
Час виконання з задачами для n = 100: 0.100103 секунд
LU-декомпозиція некоректна!
Час послідовного виконання для n = 500: 0.201003 секунд
Час паралельного виконання для n = 500: 0.392947 секунд
Час виконання з задачами для n = 500: 4.898734 секунд
LU-декомпозиція некоректна!
Час послідовного виконання для n = 1000: 1.306114 секунд
Час паралельного виконання для n = 1000: 5.372356 секунд
Час виконання з задачами для n = 1000: 10.097665 секунд
LU-декомпозиція некоректна!


...Program finished with exit code 0
Press ENTER to exit console.
```

Висновок: реалізував LU-розклад матриці за допомогою завдань (tasks) в OpenMP програмах.