

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНОМУ УНІВЕРСИТЕТУ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”**



**Лабораторна робота**  
з дисципліни  
«Технології розподілених систем та паралельних обчислень»

**Виконав:**  
студент групи КН-308  
Гецянин Дмитро  
**Викладач:**  
Мочурад Л.І.

Львів – 2020р.

# Лабораторна робота №1

**Тема: Основні конструкції OpenMP. Модель даних**

**Мета:** Ознайомитися з технологією OpenMP та набуті практичних навиків її використання

## Варіант №3

**Завдання.** Створити програму яка повинна реалізувати матрично-векторне множення, використовуючи вхідні дані відповідно до завдання (n – розмірність квадратної матриці). Варіанти 1-10 виконують розбиття матриці по горизонтальних смужках, 11-20 – по вертикальних.

**Завдання 3.**  $a_{ij} = j \sin(i)$ ;  $b_i = i \sin(i)$ ;  $n=120$ ;

Обробити паралельним та послідовним способами множення матриці на вектор відповідно до свого варіанту. Вивести результат виконання алгоритму паралельним та послідовним способами. Визначити час, який був затрачений на виконання програми для обох способів множення матриці на вектор. Розроблену програму виконати по чергово на 1, 2, 4 та 8 ядерному процесорі. Побудувати графік залежності часу обчислення від кількості ядер. Навести оцінки паралельного прискорення (parallel speedup) та паралельної ефективності (parallel efficiency).

## Програмний код

```
#include <iostream>
#include <thread>
#include <time.h>
#include <omp.h>

using namespace std;

int serial(float **matrix, float *vector, float *result, int size_i, int size_j) {

    int i;
    int j;
    for (i = 0; i < size_i; i++) {
        result[i] = 0;
        for (j = 0; j < size_j; j++) {
            matrix[i][j] = j * sin(i);
            vector[i] = i * sin(i);
            result[i] += matrix[i][j] * vector[j];
        }
    }
    return 0;
}

int parallel(float** matrix, float* vector, float* result, int size_i, int size_j)
{
    int i, j;

#pragma omp parallel shared(matrix,result,vector) private(i,j)
    {
```

```

#pragma omp for schedule(static)
    for (i = 0; i < size_i; i++) {
        result[i] = 0.;
        for (j = 0; j < size_j; j++) {
            matrix[i][j] = j * sin(i);
            vector[i] = i * sin(i);
            result[i] += matrix[i][j] * vector[j];
        }
    }
return 0;
}

int main() {

    int size_i;
    int size_j;
    float** matrix;
    float* vector;
    float* result;

    //int num_threads;
    //cout << "Input amount of threads: ";
    //cin >> num_threads;
    cout << "Input n:\n n = ";
    cin >> size_i;
    size_j = size_i;

    matrix = new float*[size_i];
    vector = new float[size_j];
    result = new float[size_j];

    for (int i = 0; i < size_i; ++i) {
        matrix[i] = new float[size_j];
    }

    for (int i = 0; i < size_i; ++i) {
        for (int j = 0; j < size_j; ++j) {
            matrix[i][j] = rand() % 10;
        }
    }

    for (int i = 0; i < size_j; ++i) {
        vector[i] = rand() % 10;
    }

    clock_t time1 = clock();
    serial(matrix, vector, result, size_i, size_j);
    cout << "Serial = " << ((float)(clock() - time1)) / CLK_TCK << " seconds \n";

    omp_set_num_threads(1);
    clock_t time2 = clock();

```

```

parallel(matrix, vector, result, size_i, size_j);
cout << "Parallel = " << ((float)(clock() - time2))/CLK_TCK << " seconds \n";

omp_set_num_threads(2);
clock_t time3 = clock();
parallel(matrix, vector, result, size_i, size_j);
cout << "Parallel = " << ((float)(clock() - time3)) / CLK_TCK << " seconds \n";

omp_set_num_threads(4);
clock_t time4 = clock();
parallel(matrix, vector, result, size_i, size_j);
cout << "Parallel = " << ((float)(clock() - time4)) / CLK_TCK << " seconds \n";

for (int i = 0; i < size_i; ++i) {
    delete[] matrix[i];
}
delete[] matrix;
delete[] vector;
delete[] result;

system("pause");
return 0;
}

```

## Результати виконання програмного коду

Розмір матриці	Послідовний алгоритм (час виконання, с)	Паралельний алгоритм					
		1 потік		2 потоки		4 потоки	
		Час	Прискорення	Час	Прискорення	Час	Прискорення
1000	0,152	0,134	1,134	0,102	1,490	0,054	2,815
2000	0,755	0,631	1,197	0,247	3,057	0,134	5,634
3000	1,925	1,893	1,017	0,501	3,842	0,255	7,549
4000	2,988	2,836	1,054	0,897	3,331	0,457	6,538
5000	4,473	4,647	0,963	1,477	3,028	0,708	6,318
6000	6,905	6,617	1,044	2,188	3,156	1,070	6,453
7000	8,584	8,442	1,017	2,823	3,041	1,418	6,054
8000	10,782	10,362	1,041	3,582	3,010	1,832	5,885
9000	14,519	13,687	1,061	4,710	3,083	2,405	6,037
10000	17,766	18,247	0,974	5,845	3,040	2,884	6,160

Таблиця 1. Час обчислення та оцінка прискорення в залежності від кількості потоків



*Графік 1. Залежність часу обчислення від кількості ядер*

Розмір матриці	2 потоки		4 потоки	
	Прискорення	Паралельна ефективність	Прискорення	Паралельна ефективність
1000	1,490	0,745	2,815	0,704
2000	3,057	1,528	5,634	1,409
3000	3,842	1,921	7,549	1,887
4000	3,331	1,666	6,538	1,635
5000	3,028	1,514	6,318	1,579
6000	3,156	1,578	6,453	1,613
7000	3,041	1,520	6,054	1,513
8000	3,010	1,505	5,885	1,471
9000	3,083	1,541	6,037	1,509
10000	3,040	1,520	6,160	1,540

*Таблиця 2. Оцінка паралельної ефективності*

Коефіцієнт прискорення вираховувався за допомогою формули:

$$S_m = T_1/T_m ,$$

де  $m$  – кількість ядер. Паралельна ефективність вираховувалась за формулою:

$$P_m = T_1/(mT_m) = S_m/m .$$