**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНОМУ УНІВЕРСИТЕТІ “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 набути 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.



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 почергово 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");

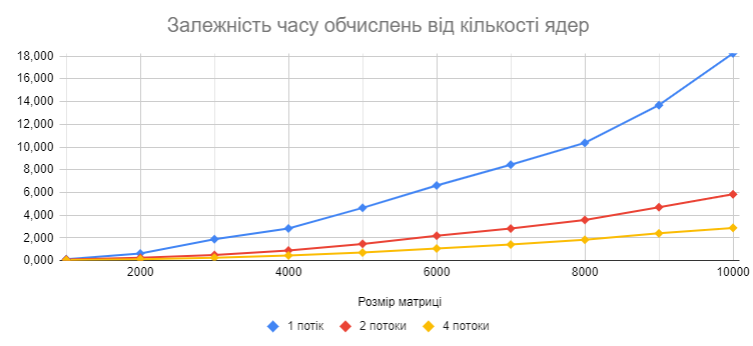
**return** **0**;

}

**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*

Коефіціент прискорення вираховувався за допомогою формули:

 ,

де m – кількість ядер. Паралельна ефективність виразовувалась за формулою:

.