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

**Звіт про виконання лабораторної роботи №5**  
з начальної дисципліни

«Паралельне програмування»

на тему:

**«Реалізація алгоритмів сортування засобами OpenMP»**

**Виконав:**  
студент групи ФЕП-22

Линва В. А.

**Львів – 2021**

**Хід роботи**

1. Обрав два із запропонованих алгоритмів сортування та реалізував їх послідовну та паралельну версію засобами OpenMP.
2. Порівняв час виконання послідовної та паралельної версії програм за різної кількості елементів.

Обрані мною алгоритми:  
1. Бульбашкове сортування.  
2. Швидке сортування.

Код програми:

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

#include <iostream>

using namespace std;

void bubblesort(double\* arr, int N)

{

double t;

for(int i = 0; i < N - 1; i++)

for(int j = 0; j < N - i; j++)

if (arr[j - 1] > arr[j])

{

t = arr[j];

arr[j] = arr[j - 1];

arr[j - 1] = t;

}

}

void parallel\_bubblesort(double\* arr, int N)

{

#pragma omp parallel

double t;

#pragma omp for

for (int i = 0; i < N - 1; i++)

for (int j = 0; j < N - i; j++)

if (arr[j - 1] > arr[j])

{

t = arr[j];

arr[j] = arr[j - 1];

arr[j - 1] = t;

}

}

void swap(int\* a, int\* b)

{

int t = \*a;

\*a = \*b;

\*b = t;

}

int partition(int arr[], int start, int end)

{

int pivot = arr[end];

int i = (start - 1);

for (int j = start; j <= end - 1; j++)

{

if (arr[j] < pivot)

{

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[end]);

return (i + 1);

}

void parallel\_quicksort(int arr[], int start, int end)

{

int index;

if (start < end)

{

index = partition(arr, start, end);

#pragma omp parallel sections

{

#pragma omp section

{

parallel\_quicksort(arr, start, index - 1);

}

#pragma omp section

{

parallel\_quicksort(arr, index + 1, end);

}

}

}

}

void quicksort(int arr[], int start, int end)

{

int index;

if (start < end)

{

index = partition(arr, start, end);

{

quicksort(arr, start, index - 1);

quicksort(arr, index + 1, end);

}

}

}

int main()

{

int G;

cout <<"Choose: \n1 - Bubble sort \n2 - Parallel bubble sort \n3 - Quick sort \n4 - Parallel quick sort \nYou choose: ";

cin >> G;

switch (G)

{

case 1:

{

double start\_time, end\_time;

const int N = 100;

double arr[N];

for (int i = 0; i < N; i++)

{

arr[i] = rand() % 100;

cout << arr[i] << " ";

}

start\_time = omp\_get\_wtime();

bubblesort(arr, N);

end\_time = omp\_get\_wtime();

printf("\nArray after sorting is: \n");

for (int i = 0; i < N; i++)

{

cout << arr[i] << " ";

}

printf("\nTime of execution %lf\n", end\_time - start\_time);

return 0;

}

case 2:

{

double start\_time, end\_time;

const int N = 100;

srand(0);

double arr[N];

for (int i = 0; i < N; i++)

{

arr[i] = rand() % 100;

cout << arr[i] << " ";

}

start\_time = omp\_get\_wtime();

parallel\_bubblesort(arr, N);

end\_time = omp\_get\_wtime();

printf("\nArray after sorting is: \n");

for (int i = 0; i < N; i++)

{

arr[i] = rand() % 100;

cout << arr[i] << " ";

}

printf("\nTime of execution %lf\n", end\_time - start\_time);

return 0;

}

case 3:

{

double start\_time, end\_time;

const int N = 100;

srand(0);

int arr[N];

for (int i = 0; i < N; i++)

{

printf("%d ", arr[i] = rand() % 100);

}

start\_time = omp\_get\_wtime();

quicksort(arr, 0, N - 1);

end\_time = omp\_get\_wtime();

printf("\nArray after sorting is: \n");

for (int i = 0; i < N; i++)

{

printf("%d ", arr[i]);

}

printf("\nTime of execution %lf\n", end\_time - start\_time);

return 0;

}

case 4:

{

double start\_time, end\_time;

const int N = 100;

srand(0);

int arr[N];

for (int i = 0; i < N; i++)

{

printf("%d ", arr[i] = rand() % 100);

}

start\_time = omp\_get\_wtime();

parallel\_quicksort(arr, 0, N - 1);

end\_time = omp\_get\_wtime();

printf("\nArray after sorting is: \n");

for (int i = 0; i < N; i++)

{

printf("%d ", arr[i]);

}

printf("\nTime of execution %lf\n", end\_time - start\_time);

return 0;

}

}

}

|  |  |  |
| --- | --- | --- |
| Метод сортування | Кількість елементів | Час |
| Бульбашкою |  | 0.000093с |
| Бульбашкою (паралельна версія) | 100 | 0.000079с |
| Швидке |  | 0.001000с |
| Швидке  (паралельна версія) |  | 0.000027с |

**Висновок:** працюючи з цією лабораторною роботою я отримав навички реалізації алгоритмів сортування засобами OpenMP. Як показала практика алгоритми реалызовані таким способом працюють значно швидше.