Group Assignment 1
Parallel Sorting Assignment
ITCS443 Parallel and Distributed Systems

Parallel Rank Sort in OpenMP

Submitted by
6388019 Sorawanan Jeamjantarasakhon
6388119 Kittipat Arpanon

Submitted to
Assoc. Prof. Dr. Sudsanguan Ngamsuriyaroj

Information and Communication Technology Faculty, Mahidol University

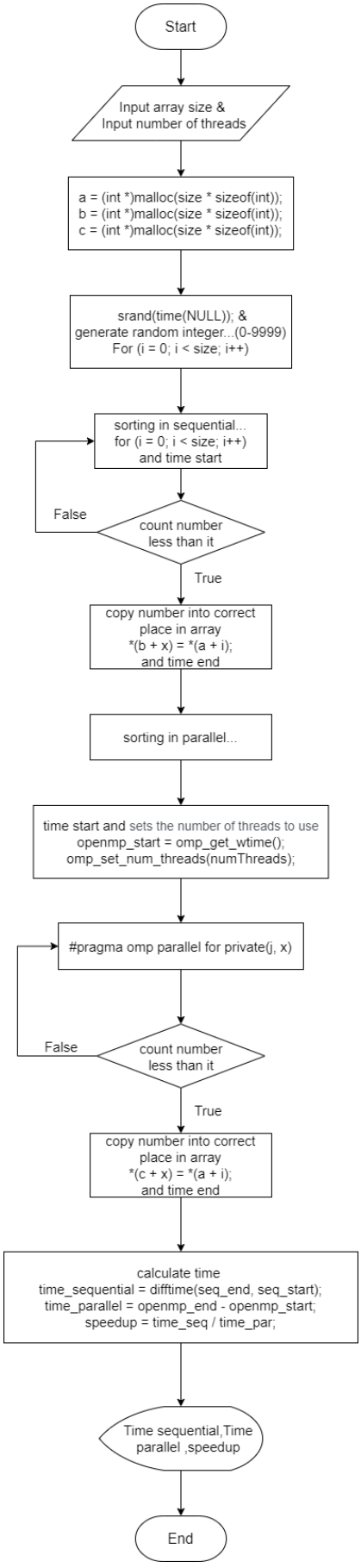
Executive Summary Page

This report is part of the IITCS443 Parallel and Distributed Systems course. The purpose of this report is to provide a better understanding of parallel rank sort in OpenMP. Our group will do our best not to make any mistakes or give our readers false information. Thank you for your interest and please enjoy our work.

Table of content

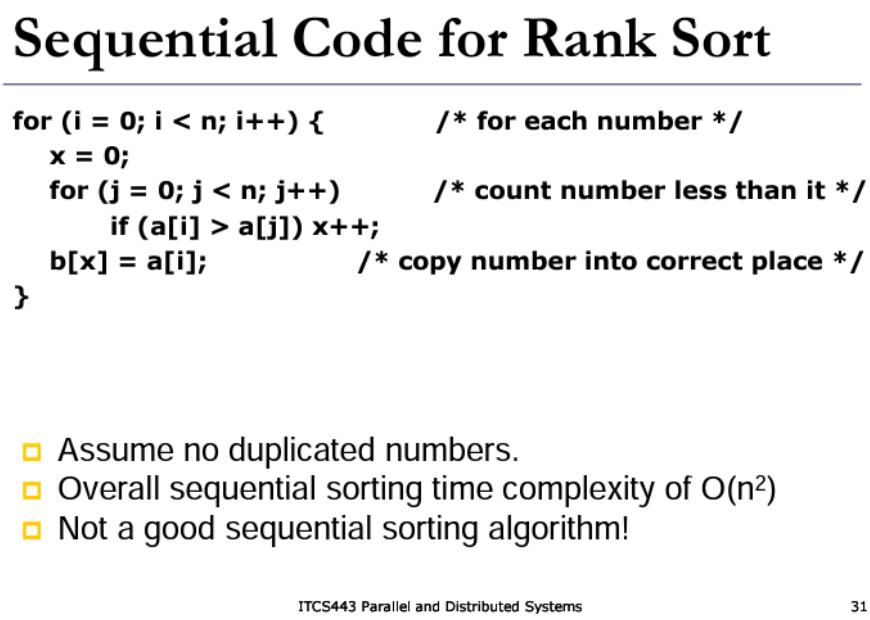
Executive summary page 2
Table of content 3
Flowchart Parallel Rank Sort in OpenMP 4
How the algorithm was implemented 5-7
Testing results 8-10
Speedup graph 11
References

Flowchart
Parallel Rank Sort in OpenMP



How the algorithm was implemented

Our group use the same rank sort algorithm from the **lecture 7 Parallel Sorting Algorithms**



But there are some implementation for our group include the following:

1. Since the program needs to accept the input of an array size of >10,000, that means we cannot use the built-in array function that is provided in C. So we have to define a pointer array by using *malloc* function.

```
int *a, *b, *c;
    a = (int *)malloc(size * sizeof(int));
    b = (int *)malloc(size * sizeof(int));
    c = (int *)malloc(size * sizeof(int));
```

2. By using a pointer array, we are also able to accept the size of the array from the user which we cannot do in the built-in array function in c.

```
int size, numThreads;
    printf("Input array size:");
    scanf("%d", &size);
```

3. Since the program needs to generate random integers from a specific range (0-9999). That means we have to use the *rand()* command and *modulation* to do that.

```
    srand(time(NULL));
    printf("generate random integer...(0-9999)\n");
    for (i = 0; i < size; i++)
    {
        *(a + i) = rand() % 10000;
    }
```

4. Since we need to calculate the speed up for the sorting algorithm, that means we have to use the <time.h> library to record the time for further calculation of speed up.

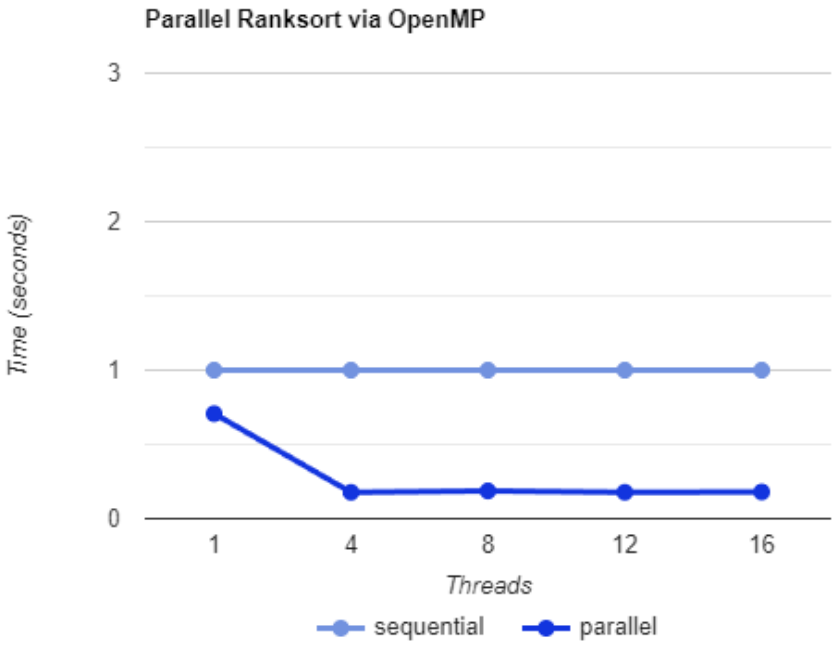
```
#include <time.h>
    time_t seq_start, seq_end;
    double openmp_start, openmp_end;
    double time_seq, time_par, speedup;
    time(&seq_start);
      Sorting in sequential…
    time(&seq_end);
    openmp_start = omp_get_wtime();
      Sorting in parallel…
    openmp_end = omp_get_wtime();
    time_seq = difftime(seq_end, seq_start);
    time_par = openmp_end - openmp_start;
    speedup = time_seq / time_par;
    printf("Time used in sequential rank sort : %f seconds \n", time_seq);
    printf("Time used in parallel rank sort using openmp : %f seconds \n", time_par);
    printf("speed up for rank sort : %f \n", speedup);
```

5. Since the provided algorithm in class does not accept duplicate numbers, that means we have to implement the condition for the sorting algorithm to work with the duplicate number.

((a + i) == *(a + j) && j < i))*

```
for (j = 0; j < size; j++)
{
    if (*(a + i) > *(a + j) || (*(a + i) == *(a + j) && j < i))
    {
        x++;
    }
}
```

Testing results

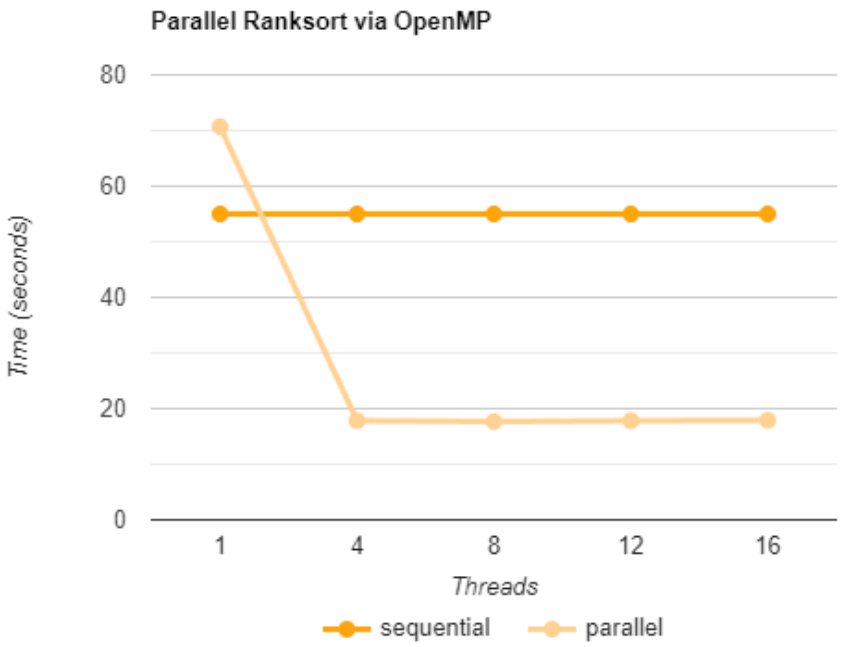


- 10,000 integers

Array Size	Threads	Time of parallel rank sort using openmp	Time of sequential rank sort	speed up for rank sort
10000	1	0.707054 seconds	1.000000 seconds	1.414319
10000	4	0.177485 seconds	1.000000 seconds	5.625926
10000	8	0.186695 seconds	1.000000 seconds	5.356337
10000	12	0.177749 seconds	1.000000 seconds	5.625910
10000	16	0.179895 seconds	1.000000 seconds	5.558783

- shots showing part of the output and the execution time.

```
[u6388019@cluster Parallel_Sorting_Algorithms]$ gcc -o Rank_Sort_openmp -fopenmp
 Rank_Sort_openmp.c
[u6388019@cluster Parallel_Sorting_Algorithms]$ ./Rank_Sort_openmp
------------------------------------------------------------
Input array size:10000
Input number of threads:1
------------------------------------------------------------
generate random integer...(0-9999)
sorting in sequential...
sorting in parallel...
Time used in sequential rank sort : 0.000000 seconds
Time used in parallel rank sort using openmp : 0.707054 seconds
------------------------------------------------------------
speed up for rank sort : 0.000000
------------------------------------------------------------
```

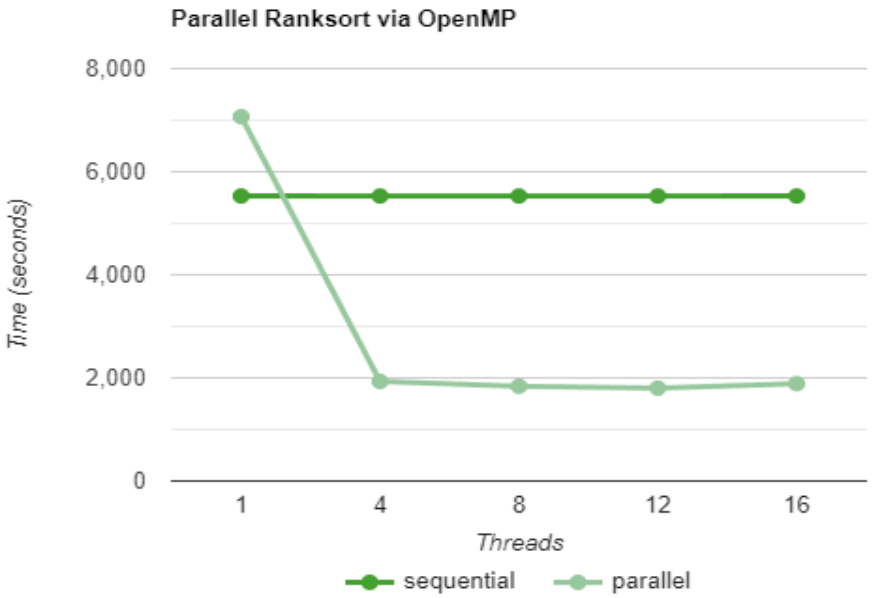


- 100,000 integers

Array Size	Threads	Time of parallel rank sort using openmp	Time of sequential rank sort	speed up for rank sort
100000	1	70.687235 seconds	55.000000 seconds	0.778075
100000	4	17.832284 seconds	55.000000 seconds	3.084294
100000	8	17.690176 seconds	55.000000 seconds	3.109070
100000	12	17.827438 seconds	55.000000 seconds	3.085132
100000	16	17.902439 seconds	55.000000 seconds	3.072207

- shots showing part of the output and the execution time.

```
[u6388019@cluster Parallel_Sorting_Algorithms]$ ./Rank_Sort_openmp
------------------------------------------------------------
Input array size:100000
Input number of threads:1
------------------------------------------------------------
generate random integer...(0-9999)
sorting in sequential...
sorting in parallel...
Time used in sequential rank sort : 55.000000 seconds
Time used in parallel rank sort using openmp : 70.687235 seconds
------------------------------------------------------------
speed up for rank sort : 0.778075
------------------------------------------------------------
```



- 1,000,000 integers

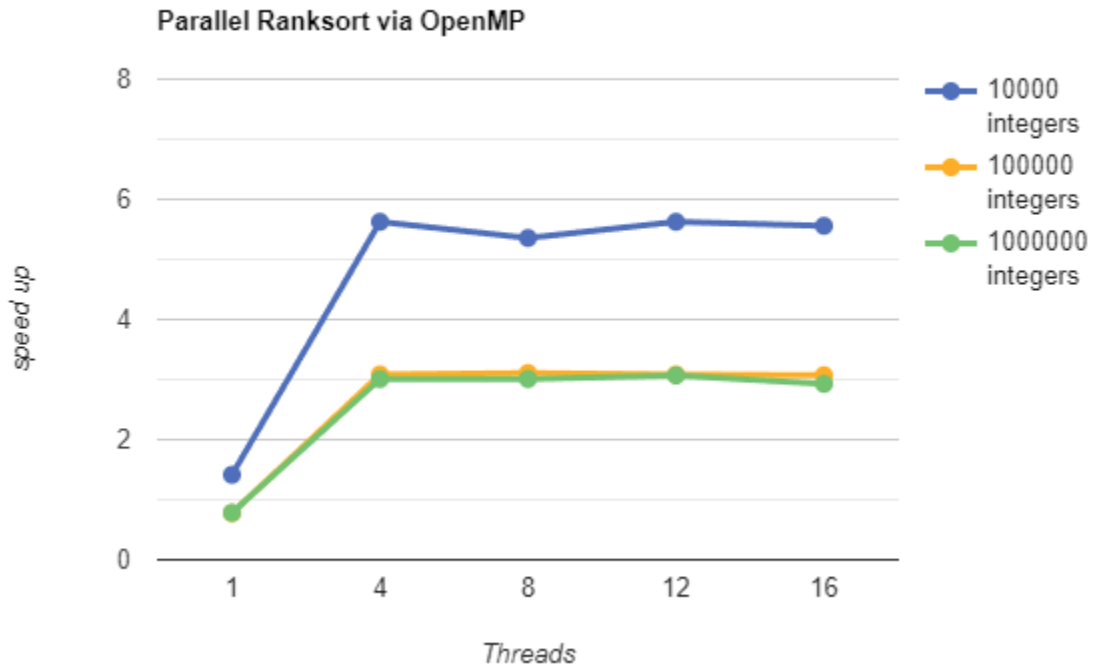
Array Size	Threads	Time of parallel rank sort using openmp	Time of sequential rank sort	speed up for rank sort
1000000	1	7,068.7235 seconds	5530.000000 seconds	0.782319
1000000	4	1931.951342 seconds	5530.000000 seconds	3.008172
1000000	8	1838.325863 seconds	5530.000000 seconds	3.008172
1000000	12	1802.065623 seconds	5530.000000 seconds	3.068700
1000000	16	1,889.079712 seconds	5530.000000 seconds	2.927351

- shots showing part of the output and the execution time.

```
Profile built 07:34 06-Sep-2021

Kickstarted 14:51 06-Sep-2021
[u6388119@cluster ~]$ cd assignment_1/
[u6388119@cluster assignment_1]$ ./Rank_Sort_openmp
Input array size:1000000
Input number of threads:8
generate random integer...(0-9999)
sorting in sequential...
sorting in parallel...
Time used in sequential rank sort : 5530.000000 seconds
Time used in parallel rank sort using openmp : 1838.325863 seconds
speed up for rank sort : 3.008172
[u6388119@cluster assignment_1]$
```

The Speedup graph shows all data sets varying between 10000 integers ,100000 integers and 1000000 integers.



Conclusion

By using OpenMP, we are able to reduce the time complexity of rank sort algorithms by a lot. By default this algorithm have time complexity of $O(N^2)$but when we use OpenMP to divide the task into multiple threads you can see from the graph the speed up of the algorithm improves for a certain point and after that, increasing the threads number will result in nothing. To summarize, by using openMP we able to reduce the time complexity of rank sort algorithm from $O(N^2)$ to $O(N)$.

References

[1] tebesfinwo, "CISC 4335 OpenMP Rank Sort," Github, 9 May 2014. [Online]. Available: https://github.com/tebesfinwo/CISC-4335-OpenMP-Rank-Sort-. [Accessed 20 October 2022].

[2] F. Tsakiris, "Parallelization of Rank sort with openMP (C)," WordPress, 31 October 2011. [Online]. Available: https://tsakiris.wordpress.com/2013/07/17/parallelization-of-rank-sort-with-openmp-c/. [Accessed 20 October 2022].