Report

Rank Sort Algorithm

By

6388113 Poomrapee Wareeboutr

6388133 Pitchaya Teerawongpairoj

6388196 Sasima Srijanya

Present to

Assoc. Prof. Dr. Sudsanguan Ngamsuriyaroj

A Report Submitted in Partial Fulfillment of

the Requirements for

ITCS443 Parallel and Distributed System

Faculty of Information and Communication Technology

Mahidol University

Semester 1/2022

TABLE OF CONTENTS

Program Explanation……...……..……... 2

Flow Chart…….……..……..…..……...….. 3

Resulting Table……..……..…. 4-5

Sample Output…….……. 6

Speed up Graph……..……..……. 6-8

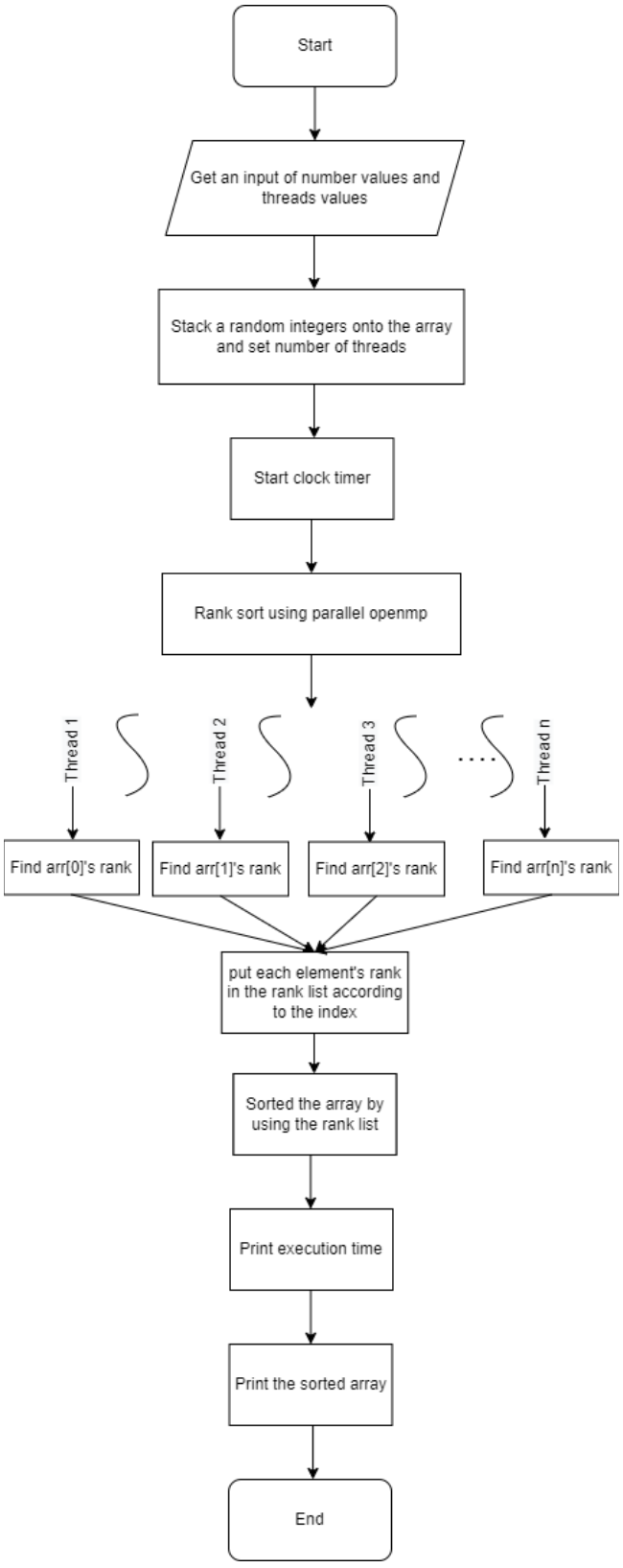
References……..……...……..…. 9

Program Explanation

In this assignment, we have been assigned to implement a rank sort algorithm that we have learned in class by using OpenMP. A rank sort is one of the linear sorting algorithms that use a parallel aspect, a particular element in an array will be assigned to just one node which is a processor or thread then find a rank of its and place it in a right place. Every node will find a rank of its number simultaneously until the end of an array.

This rank-sort method has been implemented using an array, a linear data structure. The purpose of an array is to store several elements of the same type. Additionally, the array we utilize is a one-dimensional array that stores random numerical values below 10,000 by using an integer data type. [3]

Flow Chart


Start
Get an input of number values and threads values
Stack a random integers onto the array and set number of threads
Start clock timer
Rank sort using parallel openmp
Thread 1
Thread 2
Thread 3
....
Thread n
Find arr[0]'s rank
Find arr[1]'s rank
Find arr[2]'s rank
Find arr[n]'s rank
put each element's rank in the rank list according to the index
Sorted the array by using the rank list
Print execution time
Print the sorted array
End

Resulting Table

Resulting Table for 10,000 random integer with 10,000 maximum size and 1, 4, 8, 12, and 16 threads

Number of Threads	1	4	8	12	16
1^{st} Executed Time (second)	0.541858	0.138048	0.149399	0.143476	0.137423
2^{nd} Executed Time (second)	0.541849	0.137122	0.149758	0.144359	0.142219
3^{rd} Executed Time (second)	0.542181	0.137404	0.142352	0.139336	0.141248
4^{nd} Executed Time (second)	0.542012	0.268410	0.141153	0.139815	0.136741
5^{nd} Executed Time (second)	0.541894	0.137172	0.144554	0.141669	0.136575
6^{nd} Executed Time (second)	0.542044	0.137096	0.146750	0.138630	0.137849
7^{nd} Executed Time (second)	0.541909	0.137195	0.147089	0.136830	0.139041
8^{nd} Executed Time (second)	0.541917	0.137165	0.152622	0.137847	0.141805
9^{nd} Executed Time (second)	0.542082	0.137278	0.145589	0.142414	0.139955
10^{nd} Executed Time (second)	0.541857	0.137119	0.148400	0.139181	0.138062
Average execution time (second)	0.5419603	0.150401	0.146767	0.140356	0.139092

Resulting Table for 10,000 random integer with 100,000 maximum size and 1, 4, 8, 12, and 16 threads

Number of Threads	1	4	8	12	16
1^{st} Executed Time (second)	54.056353	29.022799	27.245241	26.268926	26.832477
2^{nd} Executed Time (second)	54.056520	28.361314	26.527821	24.852917	25.390919
3^{rd} Executed Time (second)	54.088619	30.208149	29.086574	27.882302	24.872139
4^{nd} Executed Time (second)	54.070081	30.518876	28.441103	26.816055	24.348125
5^{nd} Executed Time (second)	54.064078	30.663815	24.782487	26.822293	24.082499
6^{nd} Executed Time (second)	54.060069	29.376145	25.502909	25.301016	25.420743
7^{nd} Executed Time (second)	54.062280	30.791154	23.077220	25.651880	24.227471
8^{nd} Executed Time (second)	54.050343	30.029617	26.627306	24.038664	22.988862
9^{nd} Executed Time (second)	54.083680	30.722536	24.254046	24.788587	25.459520
10^{nd} Executed Time (second)	54.103651	29.802506	21.911262	23.457100	24.672625
Average execution time (second)	54.069567	29.94969	25.7456	25.58797	24.82954

For the rank sort, time complexity is $O(n)$ which takes a lot of time to sort if the data is larger. Our group uses 1,000,000 maximum size with 8 threads to see executed time. It took around 4.5 hours to finish. So, it is hard to do the resulting table with 1,000,000 maximum size.

Sample Output

- For 10,000 random integer with 100 maximum size and 1 thread

```
12 27 59 336 364 492 540 545 846 886 925 1087 1313 1393 1421 1530 1729 1873 2305 2362 2399 2567 2651
2754 2777 2862 3058 3069 3135 3367 3368 3426 3526 3584 3750 3784 3895 3926 3929 4022 4043 4067 4324 4
370 4421 4919 5011 5060 5123 5198 5211 5368 5386 5403 5434 5736 5782 5788 5857 6091 6124 6226 6229 63
27 6413 6429 6505 6649 6808 6862 6915 6996 7084 7178 7276 7281 7373 7539 7739 7763 7793 8042 8094 816
7 8315 8335 8456 8537 8586 8690 8814 8980 9170 9172 9383 9582 9676 9802 9932 9956
time: 0.000235
```

- For 10,000 random integer with 100 maximum size and 16 threads

```
12 27 59 336 364 492 540 545 846 886 925 1087 1313 1393 1421 1530 1729 1873 2305
 2362 2399 2567 2651 2754 2777 2862 3058 3069 3135 3367 3368 3426 3526 3584 3750
 3784 3895 3926 3929 4022 4043 4067 4324 4370 4421 4919 5011 5060 5123 5198 5211
 5368 5386 5403 5434 5736 5782 5788 5857 6091 6124 6226 6229 6327 6413 6429 6505
 6649 6808 6862 6915 6996 7084 7178 7276 7281 7373 7539 7739 7763 7793 8042 8094
 8167 8315 8335 8456 8537 8586 8690 8814 8980 9170 9172 9383 9582 9676 9802 9932
 9956
time: 0.000077
```

Speed up

$$Speed\ up\ =\ S(p)\ =\ \frac{sequential\ execution\ time}{parallel\ execution\ time} = \frac{t_s}{t_p}$$

- 10,000 random integer and 10,000 maximum size

  1 thread: $Speed\ up\ = \frac{0.5419603}{0.5419603} = 1$

  4 threads: $Speed\ up\ = \frac{0.5419603}{0.150401} = 3.60344$

  8 threads: $Speed\ up\ = \frac{0.5419603}{0.146767} = 3.69266$

  12 threads: $Speed\ up\ = \frac{0.5419603}{0.140356} = 3.86133$

  16 threads: $Speed\ up\ = \frac{0.5419603}{0.139092} = 3.89642$

- 10,000 random integer and 100,000 maximum size

  1 thread: $Speed\ up\ = \frac{54.069567}{54.069567} = 1$

  4 threads: $Speed\ up\ = \frac{54.069567}{29.94969} = 1.80535$

  8 threads: $Speed\ up\ = \frac{54.069567}{25.7456} = 2.10015$

  12 threads: $Speed\ up\ = \frac{54.069567}{25.58797} = 2.11309$

  16 threads: $Speed\ up\ = \frac{54.069567}{24.82954} = 2.17763$

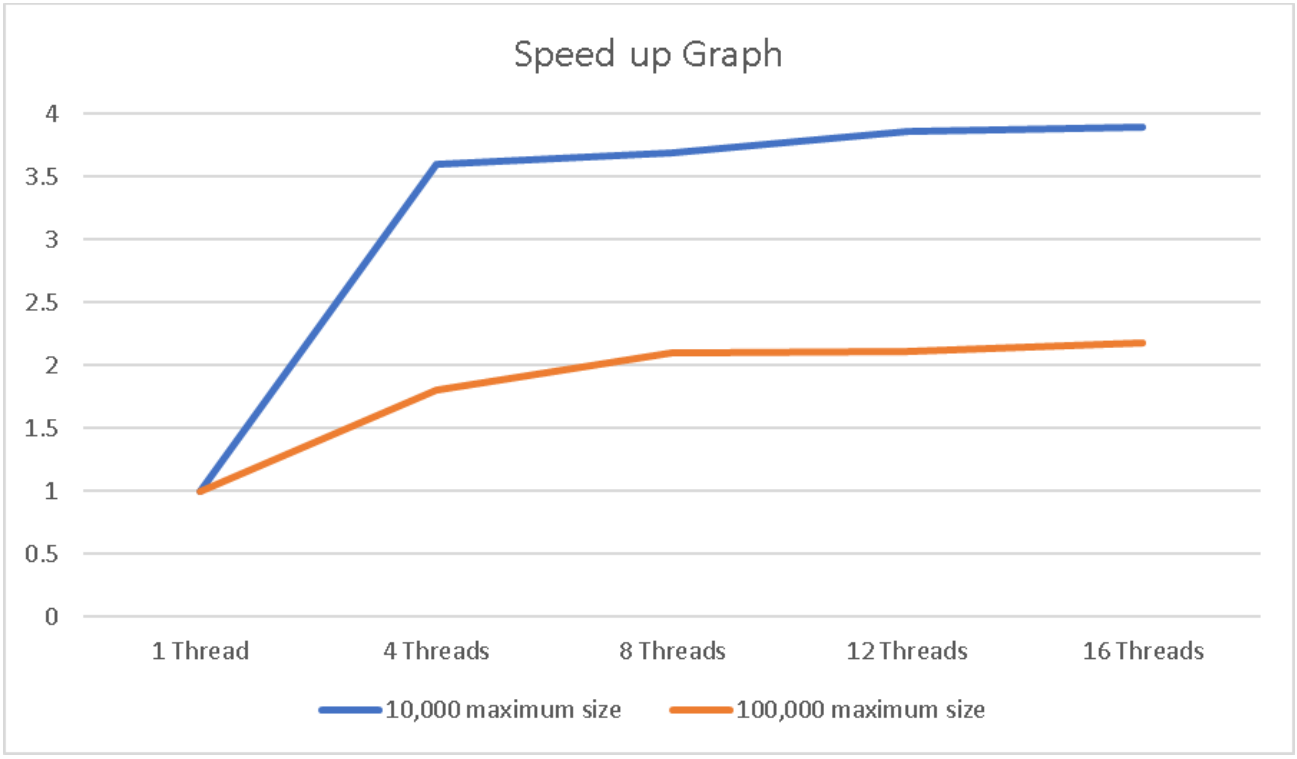


Figure 1 - Speed up graph

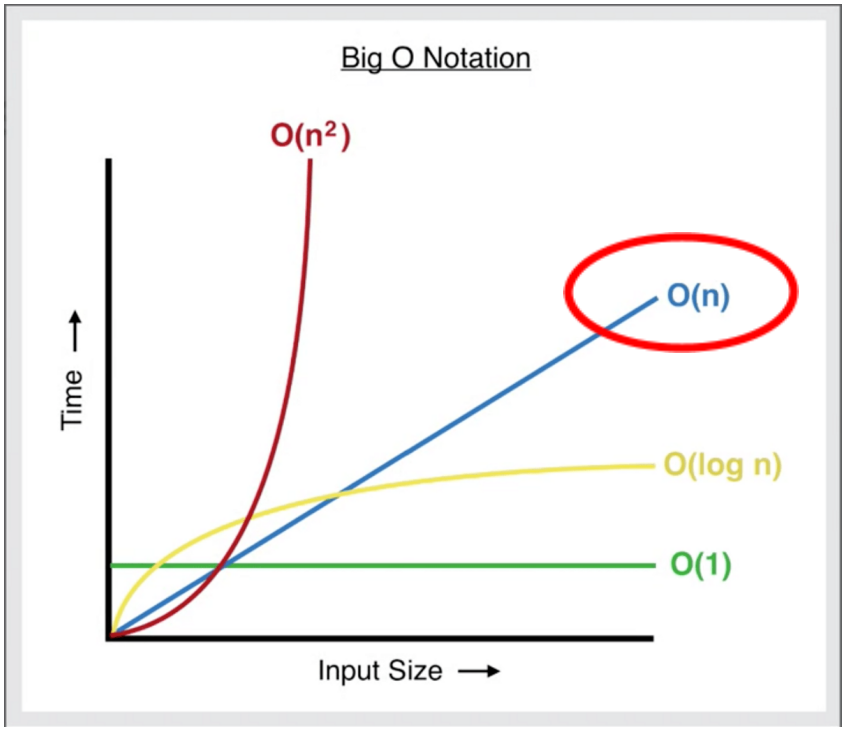


Figure 2 - Big O Notation Graph [2]

In the figure 1 x-axis is a number of threads including 1, 4, 8, 12, 16 threads and the y-axis is a speed up of each thread. Blue line is showing speed up of 10,000 maximum size and orange line is showing speed up of 100,000 maximum size. You will see if you increase the maximum size, it will decrease the speed up of the thread. The execution time of rank sort algorithm is $O(n)$ in parallel context. If you look in the figure 2, you will see $O(n)$ is a Parabola graph. If the size of number is bigger, execution time will longer compute by $O(n)$.

References

[1] Gist. 2022. *Merge Sort Program in OpenMP*. [online] Available at: <https://gist.github.com/vnkdj5/0471d4ff02371eb5bb0a8773127a448d> [Accessed 19 October 2022].

[2] Croy, M. H. (2020, July 28). *How to calculate time complexity with big O notation*. Medium. Retrieved October 19, 2022, from https://medium.com/dataseries/how-to-calculate-time-complexity-with-big-o-notation-9afe33aa4c46

[3] 2022. [online] Available at: <https://www.simplilearn.com/tutorials/data-structure-tutorial/arrays-in-data-structure#:~:text=An%20array%20is%20a%20linear,the%20size%20of%20the%20array.> [Accessed 19 October 2022].

[4] 2022. [online] Available at: <https://www.researchgate.net/figure/Flowchart-for-Threaded-OpenMP-Program_fig5_303665042> [Accessed 19 October 2022].