Khoa Nguyen - Lab 9 note - 05/03/2022

Algorithm design:

1. Odd-Even Transposition sort: The full array size will be split among the nodes, and each node will handle a subarray with size of “num_elements_per_node”. Similar to our class discussion, there are odd and even swapping phases, in which a pair of nodes swap their elements. Firstly, they both send and receive all of each other’s elements. After receiving all elements of the other node in the pair, the current node will sort the double-the-“num_elements_per_node”-sized array using C’s built-in quicksort algorithm. The smaller half elements will stay at the smaller ranking node, while the larger half will stay at the larger ranking node.
2. Enumeration sort: The full array size will be split among the nodes, and each node will handle a subarray with size of “num_elements_per_node”. Each node will assign an initial index value of zero for each element in its assigned job. Then, for each element, the node will compare its value to all remaining elements and increment the index value for each different element in the full-sized array that is smaller than the current element. For tie breaking, the head node will handle after receiving all the finished works from other nodes. Since all of the elements with the same value will end up with the same index, there will be blank spots in the final array (based on my design, the blank spots will hold value zero). The head node will go through each element in the full array for the second time to populate the blank spots with the value before them.

Below are the results of running my lab9.c code:

Odd-Even Transposition Sort running time (in seconds)			
Input Size	Serially	One Machine	Distributed
8,000	3.692500e-05	2.361731e-03	5.710979e-03
80,000	3.100760e-04	1.620037e-02	2.311056e-02
800,000	1.859049e-03	1.239873e-01	2.300169e-01
800,000,000	9.535682e-01	1.383222e+02	1.885420e+02

Enumeration Sort running time (in seconds)			
Input Size	Serially	One Machine	Distributed
8,000	5.084704e-01	8.410747e-02	6.388949e-02
80,000	4.090193e+01	5.353711e+00	5.013687e+00
800,000	Too long	Too long	Too long

For the odd-even transposition sorting algorithm, it seemed like the fastest option was running serially on one process. This is interesting and also reasonable given the design of my code. I used the C's built-in quicksort (qsort) function to perform the sorting task at each swapping phase. Therefore, with only one process, the odd-even transposition sorting algorithm will turn into a simple quicksort with high efficiency and no communication time. It seems to explain the fastest running time when running with one process. On the other hand, the running times on multiple processes of one machine were slightly faster than running on a distributed system. One possible reason is because the communication time among different machines might have taken longer than the communication time among different processes on the same machine. However, they were both surprisingly slower than the serial version. One explanation could be that the communication time really matters here when moving a large input size around the nodes. I experimented with a huge 800 millions input size, and the latter two options took an extremely long time to finish, relatively compared to the serial run.

For the enumeration sorting algorithm, we could observe a small benefit of parallelism as the running times with multiple processes were faster than the serial running time. Surprisingly, the distributed option had a faster running time than that of the one machine option. The possible reason for this result was not so clear to me. Also, on average, the running time using enumeration sort was significantly slower than using odd-even transposition sort. It has a time complexity of O(n^2), making it impossible (in the scope of this HW) to run with 800,000 input size, even when optimized with parallelism.