DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING
CONCORDIA UNIVERSITY
COMP 428/6281: Parallel Programming
Fall 2025
ASSIGNMENT 2
Due date and time:
Programming questions: Friday, November 7th before 11:59 pm
Written questions: Monday, October 27th before 11:59 pm

Programming Questions (100 marks):

In this programming assignment, you will implement two variants of Quicksort algorithm suitable for distributed memory machines. They are in Q.1 and 2 below:

Q.1. *Parallel Quicksort on a hypercube topology***:** One such algorithm for a d-dimensional hypercube is discussed in problem 9.17 of the textbook (page 421). Algorithm 9.9 and Figure 9.21 (pages 422 and 423) in the book further elaborate it. The algorithm is based on recursive halving. In this assignment, you will implement this specific quicksort algorithm on the cluster. For doing so, you will assume a virtual hypercube topology on the cluster nodes.

Compare its performance with the best sequential sorting algorithm (e.g., quicksort) to sort a large input sequence (you can generate the numbers randomly, however note that TA may provide the input for demo). As in assignment 1, you should plot speed-up versus number of processes graphs for different input data sizes and explain your findings.

Q.2. *Parallel Sorting using Regular Sampling (PSRS):* This is a parallel version of quick sort which is suitable for MIMD machines [1]. The algorithm works in the following steps:

- Step 1: Input data to be sorted of size N is initially portioned among the P processes so that each process gets $\lceil N/P \rceil$ items to sort. Each process initially sorts its own sub-list of $\lceil N/P \rceil$ items using sequential quick sort.
- Step 2: Each process P_i selects P items (we call them *regular samples*) from its local sorted sub-list at the following local indices: 0, N/P^2, $2N/P^2$,…, $(P-1)N/P^2$, and sends them to process P_0.
- Step 3: Process P_0 collects the regular samples from the P processes (which includes itself). So it has P^2 total regular samples. It sorts the regular samples using quick sort, and then chooses (P-1) pivot values at the following indices: $P + \lfloor P/2 \rfloor - 1$, $2P + \lfloor P/2 \rfloor - 1$,…, $(P-1)P + \lfloor P/2 \rfloor - 1$, and broadcasts the pivots to the P processes.
- Step 4: Each process P_i, upon receiving the P-1 pivots from process P_0, partitions its local sorted sub-list into P partitions, with the P-1 pivots as separators. Then it keeps i^{th} partition for itself and sends j^{th} partition to process P_j.
- Step 5: At the end of step 4, each process P_i has (P-1) partitions from other processes together with its own i^{th} partition. It locally merges all P (sorted) partitions to create its final sorted sub-list.

Compare its performance with the best sequential sorting algorithm, i.e., quick sort, for sorting a large input sequence (you can generate the numbers randomly). As in the previous question, you should plot speed-up versus number of processes graphs for different input data sizes.

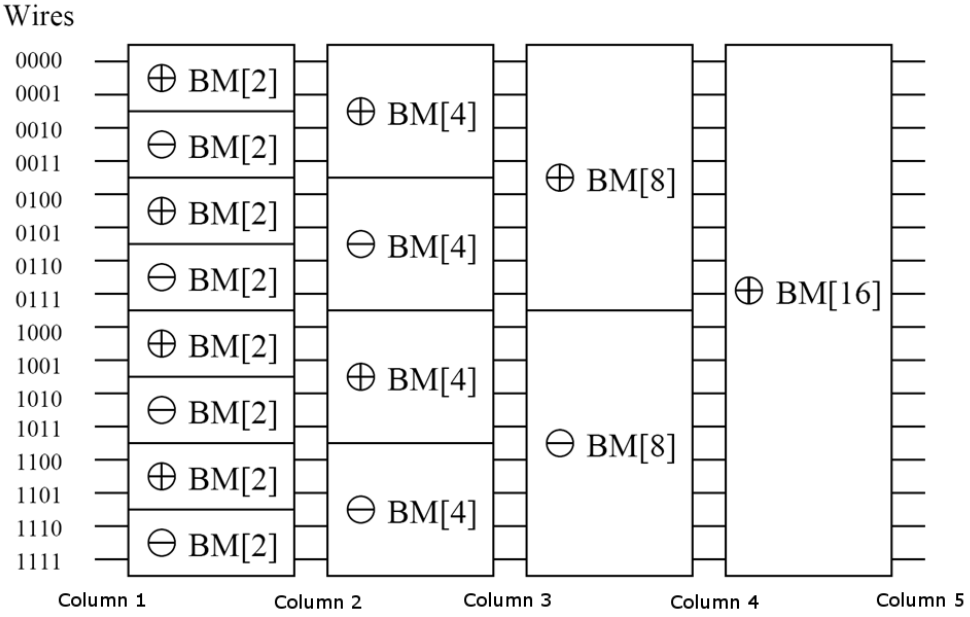
Written Questions (50 marks):

Q.3. Calculate the theoretical run-time (T_p) of the parallel quick sort algorithm in Q.1 assuming a physical hypercube topology. You may assume the best-case scenario only, i.e., a balanced pivot selection at each step of the algorithm.

Q.4. (a) Redesign the algorithm in Q.1., that uses recursive halving, for a 2-D mesh topology. Assume a $\sqrt{p} \times \sqrt{p}$ mesh comprising of p processors, where processors are numbered from P(0,0) to P($\sqrt{p}$ -1,$\sqrt{p}$ -1). You are required to write the pseudo-code only, in a format similar to the pseudo code used in the textbook (e.g. for Q.1.) (b) Calculate the theoretical run-time (T_p) of the algorithm.

Q.5. (a) Is the PSRS algorithm discussed in Q.2. balanced? Explain your understanding. (b) Calculate the theoretical run-time (T_p) of the PSRS algorithm, based on the similar theoretical analyses we did in class.

Q.6. The following figure is of a bitonic sorting network:



In the figure, ⊕BM[k] and ΘBM[k] denote bitonic merging networks of k inputs that use ⊕ and Θ comparators respectively. Each input for a comparator is a tuple whose elements are sorted in the increasing order. Each of ⊕ and Θ comparators implements a COMPARE_AND_SPLIT operation and produces outputs as tuples where elements of each output tuple are sorted in the increasing order. Given the following input tuples at wires in column 1 (refer to figure): (5 10) (4 8) (3 9) (20 30) (11 13) (12 14) (23 30) (25 29) (18 27) (1 7) (6 39) (25 43) (9 47) (13 17) (39 50) (12 83), where the leftmost input goes to wire 0000 and follows that order. What are the outputs at each wire in columns 2 to 5 of the above network?

Reference for Q.2:

[1] *Parallel Computing Theory and Practice* by Michael J Quinn (McGraw-Hill).

The programming and written assignments have to be submitted separately in Moodle. Submit the programming part as one single .zip file and the written part as one single pdf file.