DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING
CONCORDIA UNIVERSITY
COMP 428: Parallel Programming
Winter 2012
ASSIGNMENT 2
Due date and time: Sunday, March 4th before midnight.

Programming Questions (60 marks):

In this programming part, you will implement two different algorithms for parallel sorting on the cluster and compare their performances.

Q.1. *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 assignment 1, you should plot speed-up versus number of processes graphs for different input data sizes.

Q.2. *Parallel QuickSort***:** Section 9.4.3 of the textbook discusses parallel formulations of quicksort for shared-memory architectures. We did it in class. We also discussed why its implementation on distributed memory architecture may not be convenient and efficient.

There are other formulations for parallel quicksort on distributed memory machines. 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. We also discussed it in class. In this problem, you will implement this specific quicksort algorithm on the cluster. For doing so, you will create a virtual hypercube topology on the cluster.

Compare its performance with the PSRS in Q.1. Which one gives better performance? Why?

Written Questions (40 marks):

Q.3. Consider the parallel formulation of quick sort for a d-dimensional hypercube that you implemented in Q.2. How does this algorithm compare with Bitonic sort using a d-dimensional hypercube? Does pivot selection play any role? Explain your answer either intuitively or analytically.

Q.4. In the parallel formulation of quicksort algorithm on shared-memory architectures (section 9.4.3) each iteration (step) is followed by a barrier synchronization. Is barrier synchronization necessary to ensure correctness of the algorithm? If not, then how does the performance change in the absence of barrier synchronization?

Reference for Q.1:

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

Submit all your answers, including well documented source code for the parallel programs, in pdf and/or text formats only. All files should be archived into a single file (e.g., a single .zip file), and submitted through EAS.