DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING
CONCORDIA UNIVERSITY
COMP 428/6281: Parallel Programming
Fall 2017
ASSIGNMENT 3
Due date: Monday, November 27th before midnight.

Programming Questions (60 marks):

Q.1. Answer either (A) or (B) from the following:

(A) *Floyd's all-pairs shortest path algorithm***:** The algorithm is discussed in the textbook (Chapter 12.4.1 and 10.4.2) and also in the slides. Given a weighed graph G = (V,E) with *n* nodes, the cost of an edge from node i to *j* is $c_{i,j}$. Floyd's algorithm calculates the cost $d_{i,j}$ of the shortest path between each pair of nodes (*i,j*) in V. A recurrence formulation for calculating $d_{i,j}$ using *k* intermediate nodes is provided in formula 12.6 of the book and in the slides. You are required to calculate $d_{i,j}$ using the formula for each pair of nodes in the given graph, using a parallel dynamic programming solution strategy.

In your parallel solution, the cost matrix at level *k* which computes $d^k_{i,j}$, i.e., the $d_{i,j}$ values using *k* intermediate nodes, is calculated from the cost matrix at level *k-1*. Given a graph with *n* nodes, the final goal is to calculate the cost matrix at level *n*, which includes all *n* nodes. There are two possible solution strategies:

a) Strategy 1: Row- and column-wise one-to-all broadcasts are used while computing the level k matrix.
b) Strategy 2: Pipelining replaces the one-to-all broadcasts.

In this assignment, you are required to implement the parallel programs for each of the above two solution strategies and test on a graph of *n* nodes using P processors. You are also required to measure the speed-ups with different values of *n* and *P* for both the parallel versions. Explain your experimental findings, in terms of which of the two versions is better (if any) and why.

OR

(B) *The optimal Matrix-Parenthesization problem*: This problem is discussed in your text book (Chapter 12.5.1) and also in your slides. Given n matrices $A_1, A_2, \ldots, A_n$, where each A_i is an $r_{i-1} \times r_i$ matrix, your goal is to determine an optimal parenthesization of the matrices so that the cost of multiplying the *n* matrices is minimized. Let C[i,j] denote the optimal cost of multiplying $A_i, \ldots, A_j$. The recurrence formulation for calculating C[i,j] is in the text book (formula 12.7) and in the slides. You are required to calculate C[1,n] using a parallel solution strategy.

This dynamic programming formulation is non-serial polyadic. Given that there are P processors where P << n, there are two possible ways to parallelize the solution:

a) Strategy 1: This is the simpler way to implement the solution: the computation proceeds diagonal by diagonal, i.e., diagonal *m* is computed only after diagonal *m-1* is finished. If there are *N* nodes to compute in a diagonal then N/P nodes are mapped per processor. The details are in the text book and in the slides.

b) Strategy 2: There is another way: you should note that computation of C[i,j] on diagonal *m* need not wait for computations in the previous diagonal *m-1* to finish. It is because computation of C[i,j] also requires values from earlier diagonals, since the formulation is non-serial. Hence work on a diagonal can be started even if the previous diagonal has not yet finished. One way to achieve this is to have each C[i,j] mapped to a physical processor and computation of C[i,j] can start as soon as some (and not all) required data be available; this is called *pipelining* the computations. Instead of physical processors, you can imagine each C[i,j] mapped to a virtual processor. Note that there are *n(n+1)/2* such virtual processors. You will need to design the required algorithm that will map these virtual processors to the P physical processors and carry out the computations in a pipelined fashion.

In this assignment, you are required to implement the parallel programs for each of the above two solution strategies and test on arbitrary length matrix chains with matrices of appropriate dimensions (for example: in a matrix chain A_1, A_2,..., A_{100}, where A_1 is of dimension 100×200, A_2 is of dimension 200×300, and so on). Note that your program needs as input only the dimensions of the matrices, and not their contents.

You are also required to measure the speed-ups with different values of *n* and *P* for both the parallel versions. Explain your experimental findings, in terms of which of the two versions is better (if any) and why.

Written Questions (40 marks):

Q.2. In the context of parallel search techniques for discrete optimization problems, consider the *distributed tree search* scheme in which processors are allocated to different parts of the search tree dynamically as follows: initially all processors are assigned to the root. When the root node is expanded (by one of the processors assigned to it), disjoint subsets of processors at the root are assigned to each child node based on a processor-allocation strategy. One such possible strategy is to divide the processors equally among the child nodes. This process continues until there is only one processor assigned to a node; at this time the processor searches the tree rooted at the node sequentially. If a processor finishes searching the search tree rooted at the node, it is reassigned to its parent node. If the parent has other child nodes still being unexplored, then this processor is allocated to one of them. Otherwise the processor is assigned to its parent. This process continues until the entire tree is searched. Explain the advantages and/or disadvantages of this scheme.

Q.3. Prove that the *modified Diskstra's token based termination detection algorithm*, discussed in the context of dynamic load balancing in parallel search for discrete optimization problems (chapter 11.4.4 of textbook and the slides) is correct, i.e., the token initiator processor receives a green colored token if and only if all other processors have terminated; moreover if the initiator processor is idle when it receives a green token, then the algorithm terminates.

Q.4. For the parallel formulation of the longest common subsequence problem (section 12.3.1 from textbook), as to be discussed in class, the maximum efficiency is upper bounded by 0.5 for very large n. Now consider the case for a small n, say n = 4, and a coarse-grained solution, i.e. more columns are mapped per processor. Answer the following questions: (a) considering n = 4 and number of processors, p = 2, calculate the maximum possible efficiency when block mapping is used (i.e. processor P_0 is mapped columns 0 and 1, and processor P_1 is mapped the other two columns). (b) Can the maximum efficiency be improved by using a different mapping? If your answer is "yes", then illustrate and explain for the case: n = 4 and p = 2.

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.