CS 575 Spr 2015 HW1
PRAM Algorithms & Collective Communications

due Sun. Feb 22 (PRAM) & Sun. Mar 1 (communication

Problem I, Depth Counting:
Q1.a: In the memory of a PRAM machine is stored a binary tree (three pointers per node) as descibed in Section 30.1.3 of CLR Ch30. Write the fastest EREW PRAM algorithm/program (in the coding style used in the text) that creates the $3n$-node linked list (the Euler tour) with the correct weights as required to start the depth counting algorithm. Your algorithm should initialize all fields of each node in the tree/list. Provide an argument why it is EREW. Analyze its execution time and work. State, with a justification, whether or not it is work efficient.
Q1.b: Write a PRAM algorithm to solve the depth counting problem by adapting the LISTRANK algorithm as suggested in the lecture on Thursday, Feb. 5. Analyze your algorithm carefully to answer the following questions.

1. What subclass of PRAM algorithms is it – exclusive/concurrent reads/writes and if concurrent writes, which conflict resolution strategy it uses. Also precisely state any necessary synchronizations in order to ensure these properties.
2. Analyze the execution time of the program.
3. Derive the formula for the work done by the parallel algorithm, and state whether it is work-efficient or not.
4. Compare the algorithm with the one based on the Euler's tour tehniques described in the text.

Problem II, Concurrent Writes:
Q2.a: Write an CRCW program (using the **common** model to resolve conflicts during concurrent writes) that computes the smallest of n numbers in constant time.
Q2.b: Is this algorithm work efficient? Justify: if so explain why, and if not by what factor should its work be reduced in order to make it work efficient?
Q2.c: Write the fastest, work-efficient[1] EREW algorithm to solve this problem (you may have to use Brent's theorem). Justify why it is the fastest possible.
Q2.d: Write the fastest work-efficient ERCW algorithm (pick the most favorable choice of conflict resolution) to solve this problem.

[1]Two adjectives are used: "fastest" and "work-efficient." This is interpreted as: it should be first and foremost, the fastest possible algorithm for this model. Next, it should also be work-efficient.

Problem III, Array prefix:
Q3.a: Write the code for the divide-and-conquer program to compute the prefix min of an array of N numbers (assume that N is a power of 2) (the first array-based algorithm developed in class, not the zip-unzip one). You may write this code with a recursion or you may "unroll" the recursion and write a loop program.
Q3.b: Analyze the execution time and work done by your algorithm.
Q3.c: Modify your code so that it can execute on a p-processor EREW PRAM, for any value of p less than N (assume that p is also a power of 2). For what value of p is this algorithm work-efficient?

Problem IV, Brent's Theorem:
Q4.a: Our goal is to systematically develop a work efficient EREW PRAM algorithm for multiplying a (square, $n \times n$) matrix with an n-vector. This problem is simpler than the matrix-matrix multiplication algorithm described in the lecture on Feb 12.

1. Desribe the fastest PRAM algorithm to solve the problem—you are free to use as powerful a machine as you need. What is the time and work complexity of your algorithm? Is it work efficient? Justify.
2. Successively relax the machine model and improve your algorithm so that it achieves the optimal execution time on an EREW PRAM, and is work-efficient. At each step, you should
   - justify why the algorithm is in the subclass you claim it to be,
   - give its execution time and work.
   - state whether it is time-optimal and work efficient, and
   - justify each claim.

Problem V, Communication algorithms: This part is due Sunday, March 1.
Q5.a: Write the pseudo-code of a parallel algorithm to perform the gather operation (think of it as a reduction with the concatenation operator). You should "unroll" the divide-and-conquer recursion and write a loop program, making sure that the first communication is among processors/processes that differ in their least significant bits.
Q5.b: Analyze the execution time of the algorithm on a topology-oblivious machine with N processors, using the $t_s + t_w m$ cost for message transmission.
Q5.c: Now we want to implement the algorithm on a more realistic machine, the 2-D square mesh (so, N is an even power ot 2) with cut-through routing. In order to do this, we need to map the nodes (processes) in the above algorithm to the $\langle x, y \rangle$ coordinates of the mesh. We do this in the most obvious manner: node n, for $0 \leq i < N$ is mapped to $\langle \lfloor \frac{n}{\sqrt{N}} \rfloor, n \bmod \sqrt{N} \rangle$. This mapping simply consists of taking the $\log N$ bits

that represent n, using the first half of them to determine the row index, x, and the rest to determine the column index, y.

With cut-through routing, message transmission is still $t_s + t_w m$ if there is no congestion, regardless of the length of the path from source to destination. However, if at any step in the algorithm, there are k messages that need to traverse the same link, the cost becomes $t_s + kt_w m$. Note that k does not affect the startup (latency) just the message transmission time (bandwidth term). Analyze the execution time of the algorithm on this machine.

Q4.d: Analyze your algorithm if the machine uses store-and-forward routing, so that the communication time is now $t_s + t_w ml$, where l is the number of links or hops that a message takes.

Q4.e: Compare this performance to that of the topology-oblivious machine. Is this the best algorithm for the store-and-forward 2D mesh network? Justify.