EECS 587 Homework due September 15, 2015

This homework is due at the start of class. We will go over the assignment in class, with volunteers presenting their solutions. Late assignments will not be accepted. If you cannot attend class that day then email us a copy of your homework before the class starts. As the course overview states, the homework is to be typewritten, though you can draw figures by hand. *Throughout this class, you are to do all homework on your own unless told otherwise.*

The parallel computers are constructed of standard microprocessors, able to do standard operations such as multiply, compare, square root, find the j^{th} bit of a number, etc., in a constant amount of time. There are p processors, with ids $0 \ldots p-1$. Each processor starts with the values of p, its id, and a boolean value v, i.e., v is either true (t) or false (f). All processors must run the same program, though the program may have conditional statements which use p, id, etc. The program can use as many scalar variables as you want. However, no processor can store an array with a size depending on p.

For communication, there are 2 operations:

- send(destination, value): send value to processor destination
- receive(origin, variable): receive the value sent by processor origin, putting it into variable. The receive operation is *blocking*, meaning that the processor will wait until the value has arrived. If the value is sent before the receive is issued, then it is stored in a buffer and immediately available when the receive command starts.

If a processor tries to send to or receive from a processor that does not exist or it is not connected to, then the program fails to run and your homework fails to solve the problem.

You will write programs to compute scan(or), that is, to have processsor i end up with the value $\bigvee\{v_j : 0 \leq j \leq i\}$, where v_j denotes the v value in processor j. To be able to compute scan(or) efficiently in parallel, you need to figure out how to decompose the problem into parts that can be done in parallel. For example, suppose $p = 8$ and the values are f, f, f, t, f, f, t, f. Then scan(or) is f, f, f, t, t, t, t, t. If you find the scan(or) on the first half, and simultaneously on the second half, you get f, f, f, t and f, f, t, t. Call these values $w(0), w(1), \ldots, w(7)$. The first half is correct, but for every value in the second half, i.e., for all $4 \leq i \leq 7$, the correct value of $w(i)$ is $w(3) \vee w(i)$. Also note that the values for the halves may have been determined recursively, simultaneously solving four problems of size 2.

For the two computers listed below

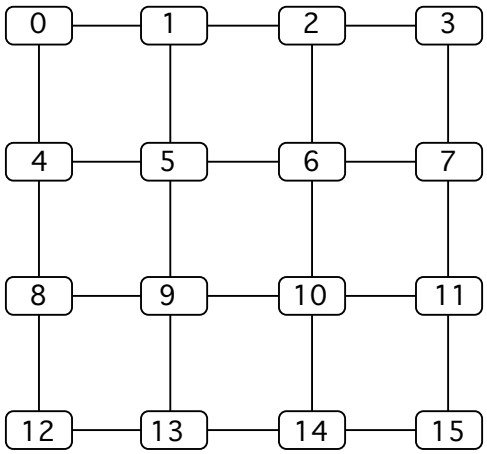
1. Write an efficient program, in pseudo-code, to compute scan(or). You must make it clear that your program is correct. By an efficient program I mean one that minimizes worst-case running time, as measured in O-notation.
2. Produce an O-notational analysis of the running time of your algorithm as a function of p. Assume that all processors start at the same time.

Do this for

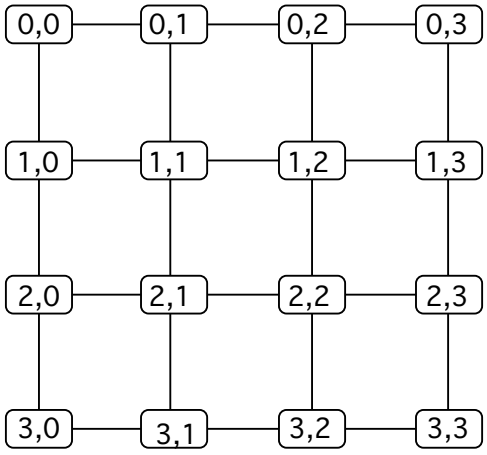
1. A completely connected computer (also called fully connected), where every processor is connected to every other one. If you need to you may assume that p is an integral power of 2, but you will lose a small amount of credit. If you make this assumption make it clear that you have done so.

2. A 2-dimensional mesh. For this computer p is a perfect square. For processor i, let $\mathrm{row}_i = \lfloor i/\sqrt{p} \rfloor$ and $\mathrm{col}_i = i - \mathrm{row}_i\sqrt{p}$. Processors i and j are connected if and only if either $\mathrm{col}_i = \mathrm{col}_j$ and $|\,\mathrm{row}_i - \mathrm{row}_j\,| = 1$, or $\mathrm{row}_i = \mathrm{row}_j$ and $|\,\mathrm{col}_i - \mathrm{col}_j\,| = 1$, i.e., each computer is connected to its up, down, left and right neighbors. Note that some processors do not have 4 neighbors.
Hint: first solve it for a 1-dimensional mesh, i.e., processors in a line.

To better understand the numbering in the 2-dimensional mesh, assume $p = 16$. The connnections between the processors in terms of their ids looks like



This is more obvious once each processor has computed its row and column coordinates from its id.



Note that you can also reverse the process, converting (row, column) coordinates into the processor id, by $id = \mathrm{row} \cdot \sqrt{p} + \mathrm{col}$.