18.435/6.848 Theory of Parallel and VLSI Computation	**Fall 1997**

Problem Set #3

Due: Wednesday, October 15 — *Prof. Leighton*

Reading (for lectures 8–10): Skim the rest of chapter 1. All of chapter 2 except sections 2.2.7, 2.2.5, and 2.6

Problems (from lectures 7–10):

1. Show how to multiply a vector by an $N \times N$ matrix in $O(\log N)$ steps on an $\frac{N}{\sqrt{\log N}} \times \frac{N}{\sqrt{\log N}}$ mesh of trees if each processor can store $O(\log N)$ words of data. Also, devise a network with $O(N^3/\log N)$ processors that can multiply two $N \times N$ matrices in $O(\log N)$ steps. (2.17,2.34)

2. Show how to perform a pivot on an $N \times N$ matrix in $O(\log N)$ steps using an $\frac{N}{\sqrt{\log N}} \times \frac{N}{\sqrt{\log N}}$ mesh of trees, provided that each processor can store $O(\log N)$ words of data. Next, use this result to design a stable algorithm for Gaussian elimination on an $N \times N$ matrix that runs in $O(N \log N)$ steps and uses $O(N^2/\log N)$ processors. This algorithm should always select the largest possible pivot among all entries in the matrix, and if it is used to solve a system of equations, the solution to the system of equations should be output in correct order. (2.21,2.22)

3. Given a two-dimensional array of $0-1$ pixel values, a *connected component algorithm* assigns a label to each 1-pixel so that two 1-pixels are assigned the same label if and only if they are in the same connected component. Two 1-pixels are considered to be *connected* (i.e. in the same connected component) if there is a path of contiguous 1-pixels linking them together. Two pixels are considered to be *contiguous* if they are adjacent vertically, horizontally or diagonally. Show how to label the connected regions of a $\sqrt{N} \times \sqrt{N}$ image in $O(\log^3 N)$ steps on a $\sqrt{N} \times \sqrt{N}$ mesh of trees. (2.53)

*4. Show how to compute the nearest 1-pixel to each pixel in a $\sqrt{N} \times \sqrt{N}$ image in $O(\log^2 N)$ steps on a $\sqrt{N} \times \sqrt{N}$ mesh of trees. (Hint: Use a recursive approach, first counting the nearest neighbors to the boundaries of larger and larger boxes, and then working back down in reverse fashion.) (2.55)

*5. Consider a $\sqrt{N} \times \sqrt{N}$ grid (without loss of generality let $\sqrt{N}$ be even) with *wrap-around* wires (i.e. in addition to the NEWS wires, the row h processor in column $\sqrt{N}$ is connected to the row $h+1$ processor in column 1 for $1 \le h \le \sqrt{N}-1$. Consider the following bubble sorting algorithm:

   (a) At step $4i+1$, each row acts as a linear array and contents of cells 1 and 2, 3 and 4, etc., are compared, switching values if necessary so that the smaller value is stored in the odd-numbered cell.

(b) At step $4i+2$, each column acts as a linear array and contents of cells 1 and 2, 3 and 4, etc., are compared, switching values if necessary so that the smaller value is stored in the odd-numbered cell.

(c) At step $4i+3$, each row acts as a linear array and contents of cells 2 and 3, 4 and 5, etc., are compared. Also, processor $\sqrt{N}$ uses wrap-around wires to compare contents with processor 1 of the following row. The smaller value is stored in the even-numbered cell.

(d) At step $4i+4$, each column acts as a linear array and contents of cells 2 and 3, 4 and 5, etc., are compared, switching values if necessary so that the smaller value is stored in the even-numbered cell.

In class, we showed that this algorithm needs N steps in the worst case. Show that it also takes $\Theta(N)$ time for most input permutations. (Curious Note: A slight variation of the algorithm is much faster, but the proof is harder!)

6. Show that any k-to-1 routing problem can be routed in $N-1$ steps on an N-cell linear array provided that each cell starts with at most one packet. *Extra Credit* - How does the answer change if each processor can start with up to k packets and if there are at most N packets overall? (1.188,1.189)

*7. Show that on one-to-one routing problems the basic greedy algorithm runs in at most $2\lfloor\sqrt{N}/2\rfloor$ steps on a $\sqrt{N}\times\sqrt{N}$ torus. (On a torus, we assume that each packet takes the shortest path to its correct column and then to its correct row.) (1.198)

8. It is commonly believed that the greedy (x, y, z) routing algorithm solves any one-to-one routing problem on a $N^{1/3}\times N^{1/3}\times N^{1/3}$ array in $O(N^{1/3})$ steps. (The greedy algorithm is defined as phases 2-4 of the algorithm in section 1.9.4 of the text.) This is because the diameter of the network is $O(N^{1/3})$ and the bisection width is $\Theta(N^{2/3})$. Show this is not true by showing that the greedy algorithm takes $\Omega(N^{2/3})$ steps for the bit reversal permutation. (The bit reversal permutation sends the packet at location (i, j, k) to location (i', j', k') where i' is the bit reversal of k, j' is the bit reversal of j and k' is the bit reversal of i.) How long (up to constant factors) does the greedy algorithm take for the transpose problem, where (i, j, k) is sent to (k, j, i).

9. Show that the greedy algorithm never uses more than $O(N^{2/3})$ steps for a one-to-one routing problem on an $N^{1/3}\times N^{1/3}\times N^{1/3}$ array. How do the results generalize for an r-dimensional $N^{1/r}$-sided array? (Be sure to consider the parity of r.) (1.297,1.298)

10. Given P processors show that if $M \geq P^2$ we can compute a global exclusive-or of M items just as fast on a 1-dimensional array with P processors as on a 2-dimensional array with P processors. Extra credit: Show that sorting M numbers is faster on a 2-dimensional array than on a 1-dimensional array even if $M \geq P^k$ for constant k. How fast is sorting? You may use the fact that any oblivious comparison-exchange sort for

N numbers will sort dN numbers by replacing each number by a list of d numbers and each comparison-exchange step by a function which merges $2N$ numbers and splits them into two lists of smaller and larger numbers.

11. Given a problem of size M that can be solved in T steps on an M-node, s-dimensional $M^{1/s}$-sided array, how large must M be (in terms of N) for us to be able to solve the problem on an N-node, r-dimensional $N^{1/r}$-sided array in $O(TM/N)$ steps? (You may assume that the simulation of high-dimensional arrays on low-dimensional arrays described in Subsection 1.9.5 is the best one can hope for.) (1.305)

Problems For Extra Credit

$ 1. Show how to simulate an $N \times N$ array on an $N \times N$ mesh of trees with constant slowdown. (2.5)

$$ 2. Solve either problem -

(a) Show how to compute an iteration of Gauss-Seidel relaxation for an $N \times N$ system of equations in $\log^{O(1)} N$ steps using $O(N^2)$ processors. (2.20)

(b) Devise a polylogarithmic time parallel algorithm for solving a lower triangular system of equations that uses close to $O(N^2)$ processors. (2.40)

$$ 3. Design a two-dimensional bubble-sort-like algorithm that sorts N numbers in $O(\sqrt{N})$ steps. By *bubble-sort-like*, we mean an algorithm that repeatedly cycles through a small set of local comparison-exchange operations, as in the algorithm described in Problem 1.154. (1.156)

$$$ 4. On a one-way ring of N processors, packets arrive randomly - at each step there is probability $\frac{1.99}{N}$ of a packet arriving at each node. The destinations are selected uniformly at random. The routing algorithm employed is for each packet to wait until no packet is moving out of its current node, then to move at every step until reaching its destination. When we look far into the future, what is the expected total number of items in all the queues? (It is known to be stable, a function $f(N)$, but experiments show the expected total is a constant not dependent on N.)

$$$ 5. Design a simple packet routing algorithm for one-to-one problems using $2N - 2$ steps on an $N \times N$ array with queues of size at most 4.