# Homework 24

Robbie McKinstry, Jack McQuown, Cyrus Ramavarapu

27 October 2016

## Problem 12:

## Problem 13:

A possible modification to the CREW parallel all pairs shortest path algorithm would be to perform the minimization step in alternate, thereby removing the concurent read of the distance table.

As each processor finds the minimum for the first $2^n$ length paths, it can also label the entry in the distance table with the node that resulted in the minimum entry. Starting from the initial node, the actual path can then be recovered by following the list of labels.

## Problem 14:

In order to think of Longest Common Subsequence (LCS) as a graph, we can represent both sequences A and B as a 2D array. Where one index $i$ will represent the index of a letter in the sequence A, and $j$ will represent the index of a character in the sequence B.
An example of this can be seen in the table below:

Table 1: LCS Table

| | | 1 | 5 | 6 | 8 | 2 | 9 |
|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

This table is filled with both 0's and 1's, where the 0 indicates that $A[i] \neq B[j]$ and a 1 indicates that $A[i] \equiv B[j]$.

The length of the Longest Common Subsequence will be the maximum sum of 1's, where in order for a 1 at index $g$ to be eligible to be added to the sum the $g \geq i_{prev}$ AND $g \geq j_{prev}$. In other words, the 1 at index g must be to the right and lower in the table than the previous 1 that was added to the sum at index $T[i, j]$

To do this in parallel, the table generation needs to be considered first. If the length of each sequence is n and the number of processors is p, then the time to generate the table will take $n/p$. Therefore, if $p \geq n$ the generation of the table can be done in $\mathcal{O}(1)$ time.

Because we are on a CREW system, we cannot have each thread update the value of the running sum. To work around this, we can have each thread generate its own sum for its run through the table. Each thread's sum will be placed into an array, where finding the max element of this will guess us our answer. Each thread's run will basically come down to starting at a given column $i$ and moving a certain number of indices to the right at each row $j$ and keeping a sum as it goes along. The basic idea is to generate a sum for each permutation of the table.

To find the max of this array of sums from each thread, we can use the EREW algorithm for max done in class, which is below.

**Function:** $Max(int x_1 \cdots x_n, p)$ `/* p = number of processors */`
return $max(Max(x_1 \cdots x_{n/2}, p/2), Max(x_{(n/2)+1} \cdots x_n, p/2))$

As stated previously, with p = n processors the time that this Max function will take is $\mathcal{O}\big(n/p + \log n\big)$. The time it takes for our algorithm to generate these sums with p = n processors is also equal to $\mathcal{O}\big(n/p + \log n\big)$ because we are in an algorithms class. Combining these two functions along with the table generation will yield an overall time of $\mathcal{O}\big(\log^2 n\big)$.