**CMPE 300 ANALYSIS OF ALGORITHMS**

###### FINAL ANSWERS

1. function MinCRCW (L [1:n])

Model: CRCW PRAM with p = n(n-1)/2 processors

Input: L [1:n] (a list of size n)

Output: The minimum value in L

 for 1 ≤ i ≤ n do in parallel // Initialize Win[1:n] array

 Win [i] = 0

 end in parallel

 for 1 ≤ i,j ≤ n and i < j do in parallel // Pi,j reads and compares L[i] and L[j]

 if L [i] > L [j] then

 Win [i]=1 // Processors Pi,j concurrently write 1 to Win[i]

 else

 Win [j] = 1 // Processors Pi,j concurrently write 1 to Win[j]

 endif

 end in parallel

 for 1 ≤ i ≤ n do in parallel

 if Win [i] = 0 then

 Min = L [i]

 endif

 end in parallel

 return (Min)

end

The algorithm may use the “arbitrary CW” strategy to handle the concurrent writes. In the first parallel loop, there are no concurrent writes. In the second parallel loop, more than one processor may attempt to write to the same Win [i] location. Since they all write the same value (i.e. 1) to Win [i], arbitrarily one of them can be chosen to write. In the last parallel loop, if the minimum element occurs more than once in the list, more than one processor may attempt to write to the Min variable. Since they all write the same value (the minimum value), arbitrarily one of them can be chosen to write.

Basic operation: Parallel comparison statement

In the second parallel loop, there is just 1 parallel comparison. So, W(n) = 1 ϵ Ө(1).

S(n) = W\*(n) / W(n) = (n-1) / 1 = n-1 ϵ Ө(n)

C(n) = p(n) \* W(n) = n(n-1)/2 \* 1 = n(n-1)/2 ϵ Ө(n2)

E(n) = S(n) / p(n) = (n-1) / (n(n-1)/2) = 2/n

The algorithm is not cost optimal, since C(n) = n(n-1)/2 > W\*(n) = n-1.

The algorithm is not cost effective, since C(n) is not ϵ O( (n-1) \* log k (n-1)).

The algorithm is time efficient, since W(n) = 1 ϵ O(log k n) for some k≥0.

1. Consider the following multiplication of *2n* x *2n* matrices:

$$\left[\begin{matrix}0&A\\A^{T}&0\end{matrix}\right]\left[\begin{matrix}0&B^{T}\\B&0\end{matrix}\right]=\left[\begin{matrix}AB&0\\0&A^{T}B^{T}\end{matrix}\right]$$

Let P be the problem of “symmetric matrix multiplication” and Q be the problem of “matrix multiplication”. We can show the reduction technique as follows:

 LB: ? LB: Ω(n2)

 P Q

 P′ Q′

Suppose that we want to solve an instance of problem Q. That is, we want to multiply two *n* x *n* matrices A and B. We can do this multiplication by using an algorithm for P as follows:

We can convert this problem of Q into a problem of P by using the above equation. That is, we build the two matrices $\left[\begin{matrix}0&A\\A^{T}&0\end{matrix}\right]$ and $\left[\begin{matrix}0&B^{T}\\B&0\end{matrix}\right]$ by taking the transposes and filling with zeros. This takes time less than n2. Then we multiply these two symmetric matrices in time f(n). Then we can convert the result of this symmetric matrix multiplication (P′) into the result of the original matrix multiplication (Q′) by just taking the *AB* component in the result. This takes time less than n2. So, the overall time is g(n) = “less than n2” + f(n) +“less than n2”, informally.

If it is possible to multiply two symmetric matrices in time less than n2 (i.e. f(n) < n2), this means that g(n) < n2. But, this is a contradiction. The known lower bound of matrix multiplication is Ω(n2). So, the lower bound of symmetric matrix multiplication is Ω(n2).

1. *Algorithm:*

 Let r be an n-element column vector and each element of r be 0 or 1

 Randomly determine r

 Compute x=Br

 Compute y=Ax

 Compute z=Cr

 if (y=z) then

 return (true)

 else

 return (false)

 endif

* *Complexity:*

An n\*n matrix can be multiplied by an n-element vector in O(n2). Thus, each of x=Br, y=Ax, and z=Cr can be done in O(n2) time.

Comparison of two n-element vectors can be done in O(n). Thus, y=z can be tested in O(n) time.

Therefore, the algorithm is O(n2).

* *Bias:*

If the algorithm returns false, then we are sure that AB≠C.

If the algorithm returns true, then either AB=C or AB≠C.

Therefore, the algorithm is false-biased.

* *Correctness:*

The only case that the algorithm gives incorrect output is when AB ≠ C (i.e., AB–C ≠ 0) and y=z (i.e., ABr = Cr ⇒ ABr – Cr = 0 ⇒ (AB – C) r = 0).

Call D = AB – C.

For any r, the probability that D≠0 but Dr=0 is maximized when all elements of D, except one element (call, dij), are zero *(See the note below)*. Then, Dr=0 only if rj=0 (other elements of r do not matter). Since rj is chosen randomly from the set {0,1}, the probability that rj=0 is ½. Thus, the maximum probability of incorrect output is ½.

Therefore, the algorithm is ½-correct.

(*Note:* If more than one element of D (say, m elements) is nonzero, then the corresponding m elements of r must be zero for Dr=0. This probability is (1/2)m, which is maximized for m=1.)