**CMPE 300 ANALYSIS OF ALGORITHMS**

###### FINAL ANSWERS

procedure Product (L[0:n1/3-1], n)

Model: 2-dimensional mesh Mq,q with p=q2=n2/3 processors

Input: L[0:n1/3-1] (a list of n numbers) , range: Pi,j, 0≤i,j≤q-1

Output: Prod (product of elements in the list), range: P0,0

 for Pi,j , 0≤i,j≤q-1 do in parallel */\* Each processor takes the product of n1/3 numbers*

 Prod ← L[0] */\* stored locally, independent of other processors*

 for k←1 to n1/3-1 do

 Prod ← Prod \* L[k]

 endfor

 end in parallel */\* Now, there are n2/3 numbers, each stored in a*

 */\* processor*

 for i←q-2 downto 0 do */\* For each column, product of all numbers*

 for Pi,j , 0≤j≤q-1 do in parallel */\* in that column is found and stored in the top*

 Pi,j:Temp ⇐ Pi+1,j:Prod */\* processor*

 Prod ← Prod \* Temp

 end in parallel

 endfor

 for i←q-2 downto 0 do */\* Product of all the numbers in the first row are*

 P0,i:Temp ⇐ P0,i+1:Prod */\* found and stored in P0,0 as the global product*

 Prod ← Prod \* Temp

 endfor

end

We assume that n1/3 is an integer. Initially, n elements were distributed to the n2/3 processors in such a way that each processor contains n1/3 elements (L[0:n1/3-1]).

# Basic operations: Multiplication (in the first loop) and parallel assignments (in other loops)

### W(n) = (n1/3-1)+(n1/3-1)+(n1/3-1) = 3n1/3-3 ∈ θ(n1/3)

C(n) = p(n)\*W(n) = n2/3 \* (3n1/3-3) = 3n-3n2/3 ∈ θ(n)

S(n) = W\*(n)/W(n) = (n-1) / (3n1/3-3) ∈ θ(n2/3)

E(n) = W\*(n)/C(n) = (n-1) / (3n-3n2/3) ≈ 1, for large n

The algorithm is cost order optimal since θ(C(n))=θ(W\*(n)).

1. function Associative (S[0:n-1])

 call random ({0,...,n-1}, x)

 call random ({0,...,n-1}, y)

 call random ({0,...,n-1}, z)

 if (S[x]οS[y])οS[z] = S[x]ο(S[y]οS[z]) then

 return true

 else

 return false

 endif

end

When the algorithm returns false, we are sure that the operator is not associative and thus the answer is correct. However, when it returns true, the operator may or may not be associative and the answer may be correct or incorrect. So, the algorithm is false-biased.

Since the algorithm includes randomness and may return an incorrect output, it is a probabilistic Monte Carlo algorithm.

Since it is a false-biased algorithm, we analyze the correctness probability as follows:

P(output is correct | algorithm returns true)

= 1– P(output is incorrect | algorithm returns true)

= 1 – k/n3 (in the worst situation, there are *k* triplets that cause the algorithm to give incorrect answer)

= (n-k)/n3

So, this is a (n-k)/n3-correct Monte Carlo algorithm.

Basic operation is comparison. Clearly, W(n)=1 ϵ Ө(1).

1. Suppose that we keep two lists during the execution of an algorithm: X is the set of coins that includes the counterfeit coin and Y is the set of coins that cannot include the counterfeit coin. Initially, all the coins are in the set X and the set Y is empty.

At the beginning of some step, suppose that X={c1, c2, ..., cm}, m≤n. We take a group of coins from X and/or Y (say, group G1) and another group of coins from X and/or Y (say, group G2), and we put G1 and G2 onto the pans of the balance. (Say that the coins in X that are not in G1 and G2 form another group, G3.) After this weight operation, we will understand that two of the three groups G1, G2 and G3 can not include the counterfeit coin; so all the coins in these two groups will be moved to Y.

During this operation, the adversary will intervene the execution. He/she may change the coins such that the size of the remaining group that can include the counterfeit coin (i.e. the group other than the two groups that will be moved to Y) is at least m/3. During the weighting operation, he/she will do the following: If the counterfeit coin is not already in the group (among G1, G2 and G3) that has the largest size, he/she will interchange the counterfeit coin with a coin in the group that has the largest size. The adversary can do this, since the counterfeit coin can be in any one of the groups G1, G2 and G3. In this way, after this weighting operation, the remaining group will have size of at least m/3.

In this way, after each step, the size of X will be at least 1/3 of its previous size. This means that at least $$⌈log\_{3} n ⌉$$

 steps will be necessary to decrease the size of X to 1. (Since there is a unique counterfeit coin, the size of X must be 1 after the execution of any algorithm.) Thus, the lower bound is $$⌈log\_{3} n ⌉$$

.