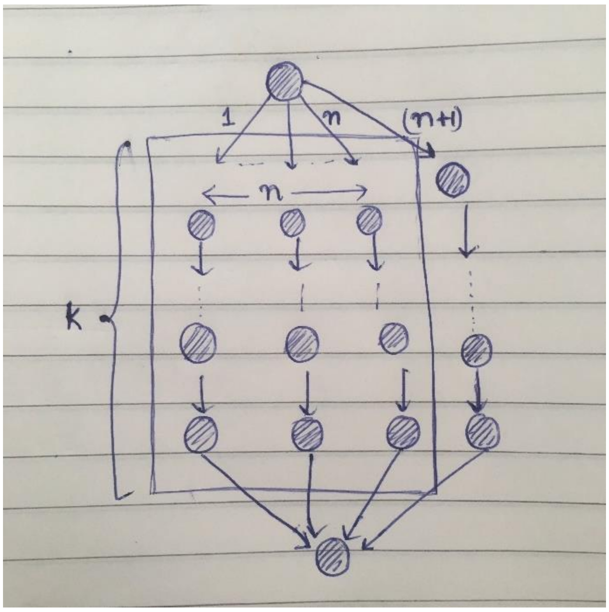
CS 255 Homework – 2

1. Construct a computation dag for which one execution of a greedy scheduler can take nearly twice the time of another execution of a greedy scheduler on the same number of processors. Describe how the two executions would proceed.

Solution



Let's assume we have **n** processors and each strand takes one-unit time for execution.
Execution1: The greedy scheduler assigns each of the n strands in the rectangular box to each of the n processors. After the k strands have executed concurrently, then the k length strand on the extreme right is executed.
Execution time $T_1 = k + k \Rightarrow 2k$

Execution2: The greedy scheduler at each time, selects n-1 strands from the rectangular box and one from the strand on the extreme right for concurrent execution among n processors.
Execution time $T_2 = k + k/n$

$$\frac{T_1}{T_2} = \frac{2k}{k\left(1+\frac{1}{n}\right)}$$

= 2 when n is large.

2. Professor Karan measures her deterministic multithreaded algorithm on 4, 10, and 64 processors of an ideal parallel computer using a greedy scheduler. She claims that the three runs yielded T4 D 80 seconds, T10 D 42 seconds, and T64 D 10 seconds. Argue that the professor is either lying or incompetent. (Hint: Use the work law (27.2), the span law (27.3), and inequality (27.5) from Exercise 27.1-3.)

<u>**Solution**</u>

$T_4 = 80sec$; $T_{10} = 42sec$; $T_{64} = 10sec$

Work Law: $T_p \geq \frac{T_1}{P}$
So, from the three execution times
$T_1 \leq 320$
$T_1 \leq 420$
$T_1 \leq 640$, which implies $\mathbf{T_1 \leq 320}$

Span Law: $T_P \geq T_\infty$
From three execution times
$T_\infty \leq 80$
$T_\infty \leq 42$
$T_\infty \leq 10$, which implies $1 \leq \mathbf{T_\infty \leq 10}$

Inequality: $T_P \leq \frac{T_1 - T_\infty}{P} + T_\infty$

Let's find T_{10};
$T_{10} \leq \frac{320-1}{10} + 1$;

$T_{10} \leq 32.9$; but $T_{10} = 42sec$

So, professor has claimed incorrect execution times.

3. Give pseudocode for an efficient multithreaded algorithm that multiplies a p×q matrix by a q×r matrix. Your algorithm should be highly parallel even if any of p, q, and r are 1. Analyse your algorithm.

<u>**Solution**</u>: A be matrix p×q and B be matrix q×r; S be matrix p×r (solution matrix)

1. Let C be a three-dimensional matrix, p×r×q initialized to zero
2. parallel for i from 1 to p
3.   parallel for j from 1 to r
4.   parallel for k from 1 to q
5. $C_{ijk} = A_{ik} * B_{kj}$
6. $S_{ij} = Sum(C_{ij}, q)$;

Analysis: Each parallel for loop has a span if O(log(p)), O(log(r)), O(log(q)); the inner most for loop does O (1) work and the loop after the inner most, the Sum function, has a span O(log(r)). So, runtime O(log(p) + log(q) + log(r)).

4. Let $A=(a_{ij})$ be an $n \times n$ matrix. Design a multithreaded algorithm which computes A^m with work $O(m \cdot n^3)$ and span $O(\log m \log^2 n)$

<u>Solution</u>

MMatrixMultiply (A, low, high)

```
if low == high
      return A;
mid = (low + high)/2
spawn A1 = MMatrixMultiply (A, low, mid)
A2 = MMatrixMultiply (A, mid+1, high)
sync
MatrixMultiply (S, A1, A2, n)
return S
```

MatrixMultiply (C, A, B, n)

```
if n == 1
      C [1][1] = A [1][1] * B [1][1]
else T be a new n×n matrix
      partition A, B, C and T into n/2 × n/2 submatrices
      spawn MatrixMultiply (C (11), A (11), B (11))
      spawn MatrixMultiply (C (12), A (11), B (12))
      spawn MatrixMultiply (C (21), A (21), B (11))
      spawn MatrixMultiply (C (22), A (21), B (12))
      spawn MatrixMultiply (T (11), A (12), B (21))
      spawn MatrixMultiply (T (12), A (12), B (22))
      spawn MatrixMultiply (T (21), A (22), B (21))
      MatrixMultiply (T (22), A (22), B (22))
      sync
      parallel for i = 1 to n
            parallel for j = 1 to n
                  C[i][j] = C[i][j] + T[i][j]
```

Analysis:

This algorithm has a work of $O(m \cdot n^3)$ and span $O(\log m \log^2 n)$

a. MatrixMultiply has work of $O(n^3)$ and span of $O(\log^2 n)$
b. MMatrixMultiply has work of O(m) and span of O(log(m))

5. Suppose we have an array A[1] to A[n] each entry of which has a positive integer priority. Devise a CREW PRAM algorithm that computes the median of these numbers in $O(\log n)$ steps using at most n processors.

<u>Solution</u>

1. Initialize k = (n + 1)/2
2. If $n = 1$ stop
3. Otherwise, pick a value uniformly at random from the n input elements
4. Each processor determines whether its element is bigger or smaller than the value
5. Let j denote the rank of the value. If j = k return P_j

   Otherwise, each element that is smaller than the splitter is moved to a distinct processor is P_i such that $i<j$ and each element that is larger than the splitter is moved to a distinct processor P_l where $l>j$

   If j < k, we find the value recursively through P_{j+1} to P_n and update k = n – j

   If j > k, we find median through P_1 to P_j