Jake Schinto

Homework #10

1. P-Matrix-Multiplication (A, B)
   1. n = A.rows
   2. let C be a new nxn matrix
   3. parallel for i = 1 to n
   4. parallel for j = 1 to n
   5. c_{ij} = P-sub-Divide (A, B, i, j, 1, n)
   6. return C

   P-sub-Divide (A, B, i, j, m, n)
   1. if m = n
   2. return $a_{im} \cdot b_{mj}$
   3. else
   4. mid = (m+n)/2
   5. x = spawn P-sub-Divide (A, B, i, j, m, mid)
   6. y = P-sub-Divide (A, B, i, j, mid+1, n)
   7. sync
   8. return x+y

Ordinarily a race condition would occur if you simply added a third parallel for. this is because the same c_{ij} is being updated all at once. Therefore, the best solution is to divide and conquer for the inner for loop to remove the race condition.

2. $T_p = \Theta(1)$

   Work = $T_p \cdot P = \Theta(1) \cdot P$

   $P = n^2 \longrightarrow = \Theta(1) \cdot n^2 = \Theta(n^2) = T_1(n)$

   Span = $T_\infty(n) = \Theta(1)$

   Parallelism = $T_1 / T_\infty = \Theta(n^2) / \Theta(1) = \Theta(n^2)$

3. a. There are no race conditions because each subprocess would be writing to a different place in the D and P arrays and G is never changed by running n Dijkstra. Therefore, each process is independent of eachother and no race condition is made.

b. $T_1(n) = O(1) + O(n \cdot V^2) = O(nV^2)$
↑ Dijkstra's

c. n processors because there are n iterations of the parallel for loop and with a $O(nV^2)$ work split among n processors, T_p would be only the work of Dijkstra's $O(V^2)$.

d. To parallelise Dijkstra's, I would change the for loop into a parallel for which is valid because relaxing doesn't rely on any sort of order and can be done independently. The while loop, however, cannot be parallel because that would defeat the purpose of the Queue and effectively break the order which is important in Dijkstra's