Homework 5

Mitchell Gant[1]

[1]Georgia Institute of Technology

April 12, 2019

Problem 1

You can compute matrix-vector multiplication using 1D column-wise partitioning using 2 steps.

Step 1: With the given element distribution in the problem description, each processor will have all of the necessary components to compute a partial solution to $y[j]$. To compute this partial solution each processor would compute $A[i,j] * x[j]$ for each of its $\frac{n}{p}$ columns.

Computation Time: $O(\frac{n^2}{p})$

Step 2: Use the all-to-all reduction algorithm as described in the Midterm 2 solutions to reduce all of the partial solutions into the final solution $y[j]$ on each processor.

Computation Time: $O(\frac{n}{p} * p) = O(n)$

Communication Time: $O(\tau \log p + \mu * m * n)$

Total Runtime: $O(\frac{n^2}{p} + n + \tau \log p + \mu * m * n)$

Problem 2

My algorithm for calculating the transpose of an n x n matrix

Step 1: For each column j, each processor in row i below the diagonal will send its contents east to the diagonal.

Step 2: The diagonal will then send the content it received north to the row j, and all processors in that row to the right side of the diagonal will send their elements south to the diagonal. (These elements will replace the ones that were previously stored in the diagonal from the processors along the column.)

Step 3: The diagonal will then send the elements it received from row j west to the column j

These three steps will be repeated for each column $j \in [0, n-1]$

Runtime: $O(3n) = O(n)$

Problem 3

The most powerful of the models is the CRCW model because it takes advantage of parallelism the most. The other models at some point will have to behave sequentially in order to either wait for another processor to read from or write to a memory location. In reality, this model is nice from a programming perspective but is impractical to implement from a hardware perspective.

Problem 4

CRCW PRAM Sum Model parallel prefix algorithm.

Step 1: Each element A_i, where i is the element's index in the array, will be assigned $n - i$ processors. These processors will all concurrently read from the memory location where the ith element is located.

Step 2: Each processor containing an element A_i will then all concurrently write their value separately to each location between B_i to B_{n-1} where array B is the resultant array of the prefix sum operation.

This algorithm will achieve its intended result but given that it uses $O(n^2)$ processors is not scalable. In reality it will also not run in O(1) time given that hardware will not actually be able to be add each element in constant time as they are being written to their respective memory locations. This algorithm is also only usable for the adding operation and not all associative operations.