CS6230 Midterm Exam

Fall 2018, October 24, 2018

Due date: November 7, 11:59pm (Canvas)

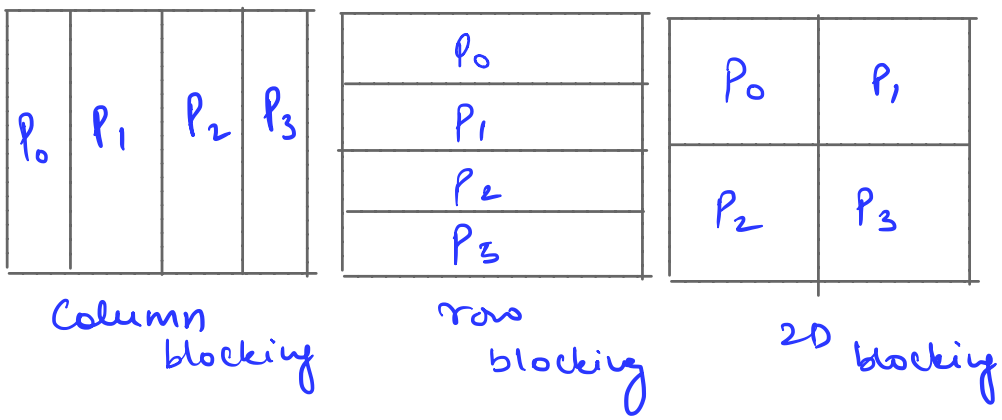
Credits: 100 points

Resources: You have to work on this exam individually. You are free to refer to the lecture notes and only the resources listed on class web page. You can also consult general sequential algorithm books such as Cormen. However, you are not to consult other online resources.

If an algorithm that we discussed in class is directly applicable, you can use it and just mention its name. If it needs any modification, you should describe the required modifications in detail.

Evaluation: I may ask you to present and analyze any or some of the questions in person.

1. A kD mesh network is created by connecting corresponding nodes in a $(k-1)D$ mesh linearly. Linear, $2D$, and $3D$ mesh networks that we saw in class are sub-categories of kD mesh networks. Assuming there are $q > 1$ processors in each dimension, there are $p = q^k$ processors in the kD mesh network.
    1. What is the network's diameter, bisection width, arc connectivity, and cost. All expressions should only include k, q, and p. For each case, briefly explain how you reached that expression.
    2. Give an algorithm for all-to-all broadcast on a kD mesh. Express the communication cost as a function of latency t_s, bandwidth t_w, q, k, and message size m.
    3. What is the algorithm and communication cost for all-to-all reduce on a kD mesh?
    4. What other network that we saw in class is a sub-category of kD mesh? Explain.
2. The Floyd–Warshall algorithm is for finding all-pairs shortest paths in a graph. Our objective is to extend Floyd-Warshall's algorithm for parallel execution on p processors. The input is the adjacency matrix A and the output is the shortest distance matrix D.
    1. We can partition the adjacency matrix into 1D blocks either row-wise, column-wise or into 2D blocks. Express the communication pattern on $1D$ row blocking, $1D$ column blocking and $2D$ blocking of matrices A and D.



    2. For each algorithm above, express the parallel run time, speedup, efficiency, and iso-efficiency functions.
3. Let A be an $n \times n$ lower triangular matrix such that $n = 2^k$. Assume that A is non-singular. Partition the matrix into four blocks, each of size $n/2 \times n/2$ as

$$A = \begin{bmatrix} A_{11} & 0 \\ A_{21} & A_{22} \end{bmatrix}$$

1. Show that the inverse of A is given by

$$A^{-1} == \begin{bmatrix} A_{11}^{-1} & 0 \\ -A_{22}^{-1} A_{21} A_{11}^{-1} & A_{22}^{-1} \end{bmatrix}$$

2. Develop a divide and conquer algorithm on PRAM to compute A^{-1} in $\mathcal{O}(\log^2 n)$ time. Explain why the algorithm depth is $\mathcal{O}(\log^2 n)$. What is the total work?

4. Let A be an array of size $n = 2^k$ consisting of two sorted sub-sequences such that $A_l = (a_0, \ldots, a_{n/2-1})$ and $A_u = (a_{n/2}, \ldots, a_{n-1})$ such that both are sorted. The odd-even merge is defined as following

```
def odd_even_merge(A):
    #All operations are in-place

    n = A.size
    if (n>2):
        odd_even_merge(A[0:n:2]) #even sub-sequences
        odd_even_merge(A[1:n:2]) #odd  sub-sequences

        for i in range(1,n-1,2):
            #compare elements with given indices in A
            #and swaps in place
            compare_swap(A,i,i+1)
    else:
        compare_swap(A,0,1)
```

It can be shown that the returned array is sorted (you are not required to do so).

1. Assuming that this claim is true, explain how you can extend this merge algorithm to a sort algorithm (i.e., odd-even merge sort).
2. Draw the sorting network for the odd-even merge sort you proposed for $n = 16$. State the network's depth and size for general n.

.