CSE Ph.D. Qualifying Exam, Spring 2026
High-Performance Computing

Instructions:

Please answer three of the following four questions. All questions are graded on a scale of 10. If you answer all four, all answers will be graded and the three lowest scores will be used in computing your total. **This exam is closed-book, closed-notes.**

Questions:

1. **Interconnection Network.** In a d-dimensional hypercube, two paths connecting the processors P_i and P_j are said to be parallel if they do not have any processors (and thus communication links also) in common except P_i and P_j. Let the bit representations of i and j differ in h bits. How many parallel paths exist between processors P_i and P_j? What can you say about the lengths of these paths in terms of h?

2. **Parallel algorithms.** Let $X[1:n]$, $Y[1:n]$, and $Z[1:n]$ be three arrays of length n, where n is a power of two, and X and Y contain positive integers. Suppose Z is computed from X and Y by the following sequential, recursive algorithm, where the side-effect-free function $f(a, b)$ takes two positive integers and returns another positive integer that depends on a and b.

   procedure `update`(Z, X, Y, n)
   if $n = 1$ **then**
   $Z[1] \leftarrow \max\{Z[1], f(X[1], Y[1])\}$
   else
   Let $X_L = X[1 : n/2]$, $X_R = X[n/2 + 1 : n]$, and similarly for Y and Z.
   `update`$(Z_L, X_L, Y_L, n/2)$
   `update`$(Z_R, X_L, Y_R, n/2)$
   `update`$(Z_L, X_R, Y_L, n/2)$
   `update`$(Z_R, X_R, Y_R, n/2)$
   `update`$(Z_R, Y_R, Z_L, n/2)$
   end if

   Note that the `update` procedure modifies the array Z in place, and pay careful attention to which operands appear where in each recursive call.

   (a) Give and analyze a parallel algorithm for this procedure for a shared-memory machine. Explain your approach.

   (b) Give and analyze a parallel algorithm suitable for a distributed-memory machine. Use the standard latency/bandwidth cost model where τ is the latency and μ is the (inverse) bandwidth. Explain your approach.

3. **Communication primitives.** Modern high-performance computing systems increasingly rely on one-sided communication models.

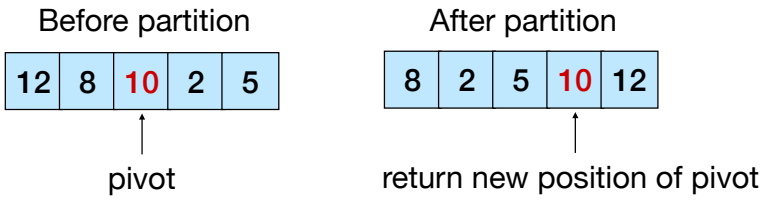

Figure 1: An example of the result of partition, a key subroutine in quicksort. Before the partition, a pivot is chosen arbitrarily. After the partition, all elements to the left of the pivot are smaller than the pivot, and all elements to the right of the pivot are greater than the pivot. The partition function returns the new position of the pivot element.

(a) Design and describe an application interface for one-sided communication.

(b) How does your interface allow the application to know that communication has completed?

(c) How does your interface allow the implementation to ensure reliable communication?

(d) Suggest two implementation strategies for your interface and comment on the performance trade-offs of these strategies.

(e) Describe an application that benefits from one-sided communication as compared to two-sided communication.

4. **Partitioning in parallel.** The *quicksort* sorting algorithm relies on recursive *partitioning* of the elements. A call to `partition(A, n, k)` takes as input 1) an array `A`, 2) the number of elements in the array, `n`, and 3) the position `k` of a *pivot* element in the array ($k < n$). The `partition(A, n, k)` function rearranges the elements of the array `A` such that all of the elements smaller than the pivot `A[k]` end up to the left of pivot (i.e., with a smaller position in the array), and all the elments larger than the pivot end up to the right of the pivot (i.e., with a larger position in the array). The `partition` function then returns the new position of the pivot. Figure 1 illustrates an example of a partition.

   Assume that the array `A` of length n is block-distributed onto p processors (i.e., each processor has n/p elements). Given latency/bandwidth communication constants τ and μ, design a distributed-memory parallel algorithm to perform the `partition` (for any given pivot position `k`) that takes $O(n/p)$ computation time and $O(n/p + (\tau + \mu) \log p)$ communication time. Show that your algorithm achieves the desired bounds.