

Problem 1

Consider the routing of messages in a parallel computer that uses **store-and-forward** routing. In such a network, the cost of sending a single message of size m from P_{source} to $P_{destination}$ via a path of length d is $t_s + t_w \times d \times m$. An alternate way of sending a message of size m is as follows. The user breaks the message into k parts each of size m/k , and then sends these k distinct messages one by one from P_{source} to $P_{destination}$. For this new method, derive the expression for time to transfer a message of size m to a node d hops away under the following two cases:

1. Assume that another message can be sent from P_{source} as soon as the previous message has reached the next node in the path.
2. Assume that another message can be sent from P_{source} only after the previous message has reached $P_{destination}$.

For each case, comment on the value of this expression as the value of k varies between 1 and m . Also, what is the optimal value of k if t_s is very large, or if $t_s = 0$?

Problem 2

Consider LU factorization of a dense matrix shown in the following.

Algorithm 1 LU factorization

```

1: for  $k := 1$  to  $n$  do
2:   for  $j := k$  to  $n$  do
3:      $A[j, k] := A[j, k]/A[k, k]$ 
4:   end for
5:   for  $j := k + 1$  to  $n$  do
6:     for  $i := k + 1$  to  $n$  do
7:        $A[i, j] := A[i, j] - A[i, k] \times A[k, j]$ ;
8:     end for
9:   end for
10: end for

```

$$\begin{pmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} \rightarrow \begin{pmatrix} L_{1,1} & 0 & 0 \\ L_{2,1} & L_{2,2} & 0 \\ L_{3,1} & L_{3,2} & L_{3,3} \end{pmatrix} \times \begin{pmatrix} U_{1,1} & U_{1,2} & U_{1,3} \\ 0 & U_{2,2} & U_{2,3} \\ 0 & 0 & U_{3,3} \end{pmatrix} \quad (1)$$

$$\begin{array}{l|l|l} 1 : A_{1,1} \rightarrow L_{1,1}U_{1,1} & 6 : A_{2,2} = A_{2,2} - L_{2,1}U_{1,2} & 11 : L_{3,2} = A_{3,2}U_{2,2}^{-1} \\ 2 : L_{2,1} = A_{2,1}U_{1,1}^{-1} & 7 : A_{3,2} = A_{3,2} - L_{3,1}U_{1,2} & 12 : U_{2,3} = L_{2,2}^{-1}A_{2,3} \\ 3 : L_{3,1} = A_{3,1}U_{1,1}^{-1} & 8 : A_{2,3} = A_{2,3} - L_{2,1}U_{1,3} & 13 : A_{3,3} = A_{3,3} - L_{3,2}U_{2,3} \\ 4 : U_{1,2} = L_{1,1}^{-1}A_{1,2} & 9 : A_{3,3} = A_{3,3} - L_{3,1}U_{1,3} & 14 : A_{3,3} \rightarrow L_{3,3}U_{3,3} \\ 5 : U_{1,3} = L_{1,1}^{-1}A_{1,3} & 10 : A_{2,2} \rightarrow L_{2,2}U_{2,2} & \end{array}$$

The figure above shows the decomposition of LU factorization into 14 tasks based on a two-dimensional partitioning of the matrix A into nine blocks $A_{i,j}$, $1 \leq i, j \leq 3$. The blocks of A are

modified into corresponding blocks of L and U as a result of factorization. The diagonal blocks of L are lower triangular submatrices with unit diagonals and the diagonal blocks of U are upper triangular submatrices. Task 1 factors the submatrix $A_{1,1}$ using Algorithm 1. Tasks 2 and 3 implement the block versions of the loop on Lines 2–3 of Algorithm 1. Tasks 4 and 5 are the upper-triangular counterparts of tasks 2 and 3. The element version of LU factorization in Algorithm 1 does not show these steps because the diagonal entries of L are 1; however, a block version must compute a block-row of U as a product of the inverse of the corresponding diagonal block of L with the block-row of A . Tasks 6–9 implement the block version of the loops on Lines 5–9 of Algorithm 1. Thus, Tasks 1–9 correspond to the block version of the first iteration of the outermost loop of Algorithm 1. The remainder of the tasks complete the factorization of A . Draw a task-dependency graph corresponding to the decomposition shown above and map the graph to a 3-processor system.

Problem 3

Scaled speedup is defined as the speedup obtained when the problem size is increased linearly with the number of processing elements; that is, if W is chosen as a base problem size for a single processing element, then

$$\text{Scaled speedup} = \frac{pW}{T_P(pW, p)} \quad (2)$$

For the problem of adding n numbers on p processing elements (Example 5.1 in the textbook), plot the speedup curves, assuming that the base problem for $p = 1$ is that of adding 256 numbers. Use $p = 1, 4, 16, 64,$ and 256 . Assume that it takes 10 time units to communicate a number between two processing elements, and that it takes 1 unit of time to add two numbers. Now plot the standard speedup curve for the base problem size and compare it with the scaled speedup curve.

Hint: The parallel runtime is $(\frac{n}{p} - 1) + 11 \log p$.

Plot a third speedup curve in which the problem size is scaled up according to the isoefficiency function, which is $\Theta(p \log p)$. Use the same expression for T_P .

Hint: The scaled speedup under this method of scaling is given by the following equation:

$$\text{Isoefficient scaled speedup} = \frac{pW \log p}{T_P(pW \log p, p)}$$

Problem 4

The dual of all-to-all broadcast is all-to-all reduction, in which each node is the destination of an all-to-one reduction. For example, consider the scenario where p nodes have a vector of p elements each, and the i th node (for all i such that $0 \leq i < p$) gets the sum of the i th elements of all the vectors. Describe an algorithm to perform all-to-all reduction on a hypercube with addition as the associative operator. If each message contains m words and t_{add} is the time to perform one addition, how much time does your algorithm take (in terms of m, p, t_{add}, t_s and t_w)?

Hint: In all-to-all broadcast, each node starts with a single message and collects p such messages by the end of the operation. In all-to-all reduction, each node starts with p distinct messages (one meant for each node) but ends up with a single message.

Problem 5

(For those taking graduate credits only) What are the major differences between message-passing and shared-address-space computers? What are their relative advantages and disadvantages.