Parallel Programming WS15 HOMEWORK #4

Submit by email to: `fabregat@aices.rwth-aachen.de`
Due by: Friday, November 20th, 5pm.

MPI – Part 2

1. In this task you will design and study the properties of an algorithm to compute a matrix-vector product $y = Ax$ built on a 1D distribution by columns of the matrix A. Process P_i owns the block of colums A_i, as well as the subvector x_i. The resulting vector y will be scattered among all processes, with process P_i being the owner of subvector y_i. Assuming 4 processes, the partitioned operation looks like:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} A_0 & A_1 & A_2 & A_3 \end{pmatrix} \times \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

   a) Sketch the parallel algorithm as we did in class. Notice: An implementation is not requested.

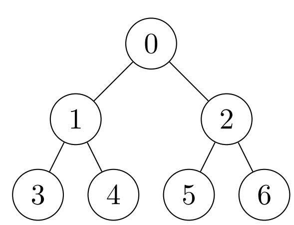
   b) Which collective communication operations does the algorithm utilize?

   c) Give the parallel cost for the algorithm, that is a lower bound for $T_p(n)$ taking into account both compute time and communication time.

   d) Study the strong and weak scalability properties of the algorithm as we did in class.

2. We have seen in class that the broadcast and reduce operations can be performed with $O(log(p))$ steps of communication, where p is the number of processes involved. In this task we will rewrite the routines `myMPI_Bcast` and `myMPI_Reduce` from *Homework 3*, and use a tree-based communication pattern to achieve this asymptotic cost.

   a) Write a function that calculates the rank of the parent, left child, and right child of a process in a binary tree representation of n processes, where the nodes are sorted by levels to build the tree. Assuming 7 processes and the process with rank 0 acting as the root, the tree looks as follows:

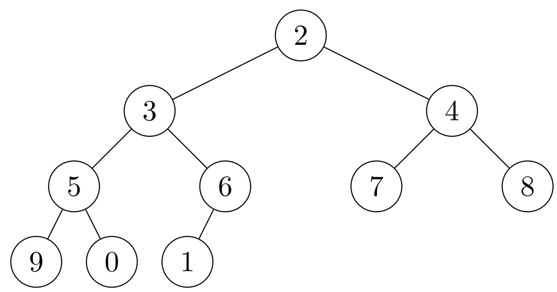


The function takes as input i, a process rank, r, the rank of the root, and s, the size of the communicator. As output, it returns the rank of the parent, the left child, and the right child of i:

```
void get_parent_and_children( int i, int r, int s,
                              int *parent, int *left_ch, int *right_ch );
```

If i has no parent, the function must set the corresponding argument to `-1`; the same happens if the left or the right children (or both) are missing.

Notice that the function must be general; that is, any process may act as root, and the number of processes is arbitrary. The following figure illustrates the tree for 10 processes and process 2 as root.



Help: If you struggle with the implementation of the function, you can use my implementation in `HW4-addendum.c`.

b) Rewrite the function `myMPI_Bcast` using point-to-point communication, leveraging the binary tree created above.

c) Rewrite the function `myMPI_Reduce` using point-to-point communication, leveraging the binary tree created above.

3. Write a program that reads a list of integers from a file, scatters the data, and performs a reduction of the data. The program takes two arguments from command line: `file_path` and `root`. Process with rank `root` reads $4 * p$ integers from file `file_path` (p is the number of processes participating in the computation), and scatters the data to every process. Once the data is scattered, the program adds entry-wise all length-4 arrays. Every process must hold the resulting array. Create your own text or binary file with the initial data to test the program.