Introduction to distributed and parallel systems

Laboratory 4

Write using MPI the following programs:

1. Implement Matrix-Vector Multiplication algorithm presented during lecture. You can assume that the matrix is a square matrix. Implement the following function:

   ```
   void parallelMatrixVectorMult(float *submatrix, float *subvector, int sizeOfMatrix).
   ```

   We assume that the matrix and the vector is equally distributed among processes. You have to preserve this in the main program before you call function `parallelMatrixVectorMult`. Remember that the size of strip you can compute as `sizeOfMatrix/p`, where `p` is the number of processes. You can add a few other parameters to the function if you justify it.
   Test the function for small and large matrix (`sizeOfMatrix` in thousands).

2. Extend your implementation of the algorithm by the ability to measure the program run time. Run your program for different number of processors (cores) and draw a chart of speedup as a function of the number of used processors (cores).

Explanation for task 1:

Preprocessing Step 1. Processor 0 prepare matrix A(n*n) and vector x(n*1). It can be read from file or generated random.

Preprocessing Step 2. Processor 0 makes personalized communications to send parts of matrix A and vector x to proper processors (also to itself). For matrix A it is used stripped partitioning.

`parallelMatrixVectorMult - start`

Algorithm Step 1. Every processor exchange parts of vector x to obtain whole vector x.

Algorithm Step 2. Every processor make local calculation to obtain part of result vector y.

`parallelMatrixVectorMult - end`

Postprocessing Step 1. Processor 0 collect parts of result vector y to have the whole vector y.

Postprocessing Step 2. Processor 0 present the result or store in a file.