ECE 473
Due Date Specified on Canvas
Parallel Matrix Vector Multiply

Tasks:

Write the following programs:

1) **make-matrix -n 100 -l 0 -u 100 -o output_file**

This program writes to output_file a binary file with an n by n (square) matrix or a size n vector if the –v flag is included.

These represent one of the matrix or vedctor to be multiplied, where the value of each element of the matrix or vector is randomly generated based on a uniform random variable on the interval 'l' to 'u'. **Data is 64-bit double precision** floating point and the number n is stored as the first word of the file (**this is an integer**). Should be stored in row major order.

2) **print-matrix input_file**

Reads a matrix file created with 1) and prints in ascii. Should work with both matrix and vector files.

3) **mv-serial input_matrix1 input_vector2 output_vector3**

Reads 2 input matrix/vector files and computes the product of the matrix in input_matrix1 with the vector in input_vector2 and then writes resulting product into the output_vector3. This should be easy to get working – then you can use it to check your parallel code.

3) **mpirun -np __ mv-parallel input_matrix1 input_vector2 output_vector3**

Reads 2 input files and computes the product of the matrix in input_matrix1 with the vector in input_vector2, and then writes the resulting product into the output_vector3. This method will implement the checkerboard decomposition. You may require the number of rows and columns of the matrix to be evenly divided by the square of the number of processors (which should be square). You must use an MPI topology to manage communication between tasks. Your matrix should be represented in memory as presented in class and in the book with an array of doubles and an array of pointers to double. You should test your code using 9, 16, 25, and 36 tasks.

Your program will need to use a problem size large enough to keep your program running for somewhere between 30sec and 3min. The larger the better. Since nodes are updated frequently, I honestly don't know the best problem size. It could be 1000, 5000, or 10000. You can run the program with difference values and look at the run time until it seems reasonable. Once someone has this information it can be shared with other classmates (would not be cheating). You need at least 3 different problem sizes for computing speedups.

Use your sequential program to generate results that you can compare with your parallel results for correctness and to compute speedup. With your parallel program, use MPI_WTime to calculate the run time and output this to standard along with the number of MPI tasks and problem size (n, size of vector). This is going to require at least 12 runs of your program (each number of tasks with each problem size). Once you have gathered this data gather the data with the same problem size and compute speedups

All your source files should be in a single directory. The name of that directory should be firstname_lastname_MV. Inside that directory, you can have the files called what you want, along with all of the source and a Makefile . Also, all of your source files (mains, source, and

headers) should have a comment block at the top that has your information and identify the file. Also, all programs must be documented well with copious comments. This directory should be zip (or .tar.gz) with the same name as the underlying directory. Upload on Canvas by the specified time.