# ECE 4730/6730 – Programming Assignment

## Due Date on Canvas

## Iterative Equation Solver

**Task:** Write a program, both a sequential and parallel version, that solves a system of n equations in n variables taking the form Ax = b. You will design it as follows:

- Matrix and vector elements will be of type double,
- Decomposition will be row-wise block,
- A Jacibi iterative solver will be used,
- Solve a dense matrix (no need to worry about sparse formats, etc.),
- A fixed number of iterations is sufficient so we can compare speeds,

**Programs**

You are to reuse programs from previous assignments to create and print matrices and vectors. The programs may need to be modified to use type double elements, or to add a command-line flag as needed. Also add one or two integers to the front of the file as needed to indicate the size of the data in two dimensions. In other words an n x m matrix file would have the following:

n (int32) | m (int32) | data[0,0] (double) | data [0,1] (double) | …

And a vector file:

n (int32) | data[0] (double) | data [1] (double) | …

For the matrices, n = m, but the read/write routine should use the standard of two int anyway. You also need to modify your matrix creating algorithm to create a matrix that has large values at and/or near the diagonal, to ensure the matrix is solvable (if for a[i,j] where i = j then a[i,j] = a[i,j] * 100).

In order to check your code you will need a program that performs sequenrial matrix vector multiplication. Then you will need an RMS vector compare program that reads two vectors and computes the RMS of the vectors and prints that on stdout. RMS error is found by taking one vector A, subtract the other vector B, then square each element of the result, divide by n and then take the square root. These programs should be sequential, they are only for your use in verifying the results. You should write a sequential Iterative as well, as another tool for checking results. A complete solve will look like this:

```
Jacobi-Parallel -d 100 MatrixA VectorX VectorB
MatrixVector-Serial MatrixA VectorX VectorBprime
RMS-Serial VectorB VectorBprime
```

The Jacobi program should take one flag -d which indicates how many iterations to perform, followed by 3 file names. The first and last files are the inputs, the middle file is the output (input matrix A and vector b, output vector x). The MatrixVector-Serial takes 3 file names, the first two inputs, and the 3rd for output. RMS-Serial takes two files and outputs a double value to the command line. I do recommend you start with a sequential iterative code code, then move to parallel.

**Performance Evaluation and Report:**

I would like for you to evaluate the performance of your parallel program by running it across a range of processes for example from 1, 2, 4, 8, 16, and higher if you can. Then, for each of these trials, I want you to run the program across number of matrix sizes (for example from 100 to 500 to 1000). You might have to experiment a little to find n values that run in a reasonable time like 5 minutes or so. I then want you to make execution time, speedup and efficiency plots, with the larger and larger values of n. Each plot should have p on the x axis and the value (speedup, efficiency, etc.) on the y axis with a different line for each value of n. Does your program appear to scale? You probably can't say definitively, just make a judgement call from the data.

As before, put all of the files you create including the report and input data files into a directory named with the same convention we have been using.

You are encouraged to make use of functions defined in MyMPI.c for I/O or other needs. Functions would include

```
void read_row_striped_matrix (
  char        *s,        /* IN - File name */
  void      ***subs,     /* OUT - 2D submatrix indices */
  void       **storage,  /* OUT - Submatrix stored here */
  MPI_Datatype dtype,    /* IN - Matrix element type */
  int         *m,        /* OUT - Matrix rows */
  int         *n,        /* OUT - Matrix cols */
  MPI_Comm     comm)     /* IN - Communicator */

void read_block_vector (
   char        *s,      /* IN - File name */
   void       **v,      /* OUT - Subvector */
   MPI_Datatype dtype,  /* IN - Element type */
   int         *n,      /* OUT - Vector length */
   MPI_Comm     comm)   /* IN - Communicator */

void print_block_vector (
  void        *v,       /* IN - Address of vector */
  MPI_Datatype dtype,   /* IN - Vector element type */
  int          n,       /* IN - Elements in vector */
  MPI_Comm     comm)    /* IN - Communicator */
```

This last one isn't exactly what you want for output (it outputs ascii) but you can easily modify it by looking at the function it calls to do output:

```
void print_subvector (
  void        *a,       /* IN - Array pointer */
  MPI_Datatype dtype,   /* IN - Array type */
  int          n)       /*
```

You will also have to add code to write the vector sizes. You may already have code from previous assignments you prefer to use and that is fine also as long as it sticks to the file specification.

RMS error is found by taking one vector A, subtract the other vector B, then square each element of the result, divide by n and then take the square root:

RMSD = sqrt( 1/n SUM over i (square(A[i] – B[i]))

Or another way:

```
/* you need the math library for pow and sqrt */
Int i, n;  /* n is set based on the size of the input files for A and B */
double sum = 0.0;
for (i = 0; i < n; i++)
     sum += pow ( (A[i] – B[i]), 2.0);
sum /= n;
sum = sqrt(sum);
printf("RMSD = %f\n", sum);
```