MTH 655, Winter 2013, LAB7

The goal of this assignment is to implement an iterative solver in parallel.

Get experience using 1 during the lab, and later work on (and turn in) one of 2 or 3 (or both for extra credit).

---

1. Please follow the class demonstration how to run the `mypi.f` example on the cluster.
Then mimick the steps and run the `myjacobi.f` example.

- Compile the code
- Create a new submit script by modifying `mypi_submit`, and give the job a new name, e.g., `yourname_myjacobi`, so it can be easily seen in the queue. Same thing as concerns the log file.
- Submit the job to the queue, and watch when it runs.
  You can use
  `qstat | grep yourusername`
  to see how it runs
- Play with different numbers of processors, different sizes of the problem, and watch the time it takes to complete.

---

2. Implement Richardson's iteration for solving $Ax = b$ for the same matrix A as coded in `myjacobi.f` (You can replace the entire Jacobi update by this iteration).

The iteration is very simple

$$x_{k+1} = x_k + \alpha r_k = x_k + \alpha(b - Ax_k) \tag{1}$$

You should use an optimal α which for the spd matrix A equals $\frac{2}{\lambda_{min}(A)+\lambda_{max}(A)}$. (We covered these eigenvalues in class so you should be able to find out what they are, or experiment in MATLAB).
Run the code in parallel using an odd number of processors, report the number of iterations, the timings, and the middle value of the solution for the middle processor.

For timings, you can use the MPI function

```
double precision mytime1, mytime2
mytime1 = MPI_Wtime()
iterations ....
mytime2 = MPI_Wtime()
if (p.eq.1) write(*,*) 'Proc.',p,'used ',mytime2-mytime1,' wall time'
```

Do you see the speedup ? Experiment with different n, p. When changing n, watch out for convergence of iteration !

---

3. Instead of Richardson's iteration as in Pbm 2, you can implement a block-Jacobi iteration.
To do this, consider the following when there are only $p = 2$ processors. Solving

$$Au = f \tag{2}$$

when $n = mp$ is equivalent to the following block system

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{3}$$

where each of the matrices $A_{11}, \ldots$ is of size $m \times m$.

Note that the "off-diagonal" matrices A_{12}, A_{21} are very sparse.

Thus, it makes sense to write the above system as

$$\begin{bmatrix} A_{11} & \\ & A_{22} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} b_1 - A_{12}u_2 \\ b_2 - A_{21}u_1 \end{bmatrix} \tag{4}$$

Since the matrices A_{12}, A_{21} are very simple, it is very easy to get the right hand sides for

$$A_{11}u_1^k = b_1 - A_{12}u_2^{k-1} \tag{5}$$
$$A_{22}u_2^k = b_2 - A_{12}u_1^{k-1} \tag{6}$$

and iterate until convergence. (Do you see now why this is called "block Jacobi" ?).

You see that you could simply solve each of the "sub-systems" (5), (6) using a direct solver such as DGESV (or a more appropriate function you found in LAB3). Or, some solver from Krylov subspace family from LAB5. Or, some other solvers from other labs ...

Note: to take advantage of any of the solvers in LAPACK family, you will have to compile the code similarly as you did in LAB3, except now you must use `mpif77` instead of `f77` for compilation.

Test your parallel performance as in Pbm 2. Of course, use more than just $p = 2$ processors.