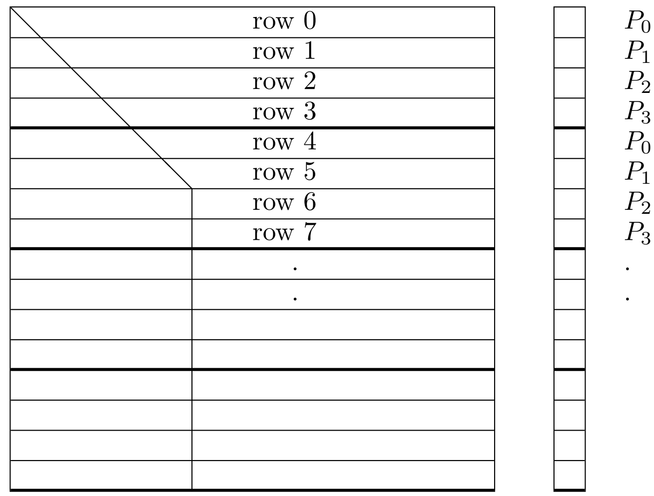
CSci 5451, S'20 — Programming Assignment # 3 — Due Date: April 30, 2020

Background

In this assignment you will program a pipelined Gaussian elimination algorithm. This was covered in class and a sketch of a *column* version is shown in the notes (in matlab). A basic Gaussian elimination algorithm is shown in Page 12-25 of Lecture notes set 12. However, we will use the interleaved mapping to processors. We have an initial matrix contained the matrix A and the right-hand side b as its last column – so the array A is of dimensoin $n \times (n+1)$. In the interleaved scheme, row i (for $i = 0, \cdots, n-1$) is assigned to process $mod(i,p)$ if p is the number of processes.



Basically the pipelined algorithm works as follows

```
for (k=0; k<n-1; k++){
    if (row k is in proc myid){
        send row k to proc. South
    }
    else{
        if (proc. myid needs row k) receive row k from proc. North
        if (proc. South needs row k) send row k to  proc. South
    }
    compute piv = ..
    process elimination for relevant rows in proc. myid
}
```

You will also implement the parallel back-solve function that solves the resulting triangular system. For easy parallelism it is best to implement the column version shown in Page 12-24 of set 12 of the notes. The parallel version works as follows:

```
for (k=n-1; k>=0, k--){
  id  = (k % nprocs);  // proc number of unknown x[k]
  if (id = myid) {     // i.e. if row k is in proc myid)
     kloc = ...        // local index of row k
     t = A[kloc,n]/A[kloc][k]; // Note actual matrix is stored as a 1-D array
     x[kloc] = t;
  }
  broadcast t to all
  //subtract multple of (relevant) part of column k  from rhs
  A[:][n] = A[:][n] -t * A[:][k]
}
```

Tests

Just as for LAB2, a main program will be provided along with blank versions of the two functions `pipe_ge(...)` and `back_solve(..)` and a makefile. The driver `main.c` generates a matrix, calls the function `pipe_ge(...)` and then the `back_solve(..)` function to get the local solutions. These local solutions are then gathered into one processor and they are printed when n is small enough or just the error is printed otherwise. All you will have to code is the two functions that are called.
Once your code has been tested you will need to analyze its performance and see, for a matrix of size 2048×2048, what happens to the execution time for (separately) the Gaussian elimination part and the triangular solve part as the number of processors increases. Check the times for nproc= 1, 2, 4, 8, 16, 32.
For this test you will need to modify `main.c` as you did on LAB2, to add calls to timing functions (use MPI_Wtime()).

What to Submit:

(1) All source codes. These should be submitted through Canvas. Provide the 3 functions `main.c`, `pipe_ge(...)` and `back_solve(..)` along with the makefile (if it is different from the one provided).
(2) Provide a file called `Report` (or `Report.pdf`). This can be a PDF file with plots. It can also be a plain text file with tables. Here are some of the items you need to comment on.

(a) Any specific comments you have on your implementation. For example: Any comments on things you did to reduce idle time the impact of communication? Did you have to use any communication commands other than sends and receives (and one broadast in `back_solve`) in your two functions?

(b) Comments on the statistics you see. Timing, efficiency, etc.

(c) **EXTRA CREDIT:** You may be able to speed-up your Gaussian elimination algorithm (not the back-solve) by exploiting threads within each processor with openMP (or pthreads) – using up to 32 threads in each PE. If you succeed and document this well you will earn extra credit. Feel free to increase the matrix size (to double?) to be able to make the gain. Document clearly what you did and the gain in execution time you made.

Grading:

1. **15 %** Style and documentation.
2. **30 %** Correctness and efficiency of your functions
3. **30 %** Correctness of the specific approach [i.e., how does it conform to what is being asked.]
4. **25 %** Quality of your report: your comments on implementation, you comments on the statistics, etc..
5. **(Extra credit**) Up to 15 additional points if you can show benefit from Hybrid programming openMP/MPI or threads/MPI. – See above for details.