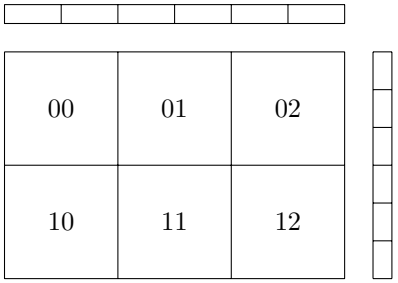
CSC 726: Matrix-Vector Multiplication In-Class Assignment

We are going to parallelize code for matrix-vector multiplication using MPI. The legacy function name for this operation is PDGEMV, which stands for Parallel Double-precision GEneral Matrix-Vector product. Complete the following exercises in groups of 2 or 3. After every group has successfully started an interactive job, you can feel free to request other interactive jobs. Be sure to share your codes with everyone in the group.

1. Review with your partner the BSP algorithm for matrix-vector product (see BSP slides in Canvas). This algorithm uses a 2D block distribution of the matrix and involves communication of the input and output vectors, as pictured below. Note that the vector distribution is unspecified in the picture. You will be determining this distribution for your MPI code, which may differ from our approach in the BSP model.



2. Compile and run the code `pdgemv-starter.cpp` available from Canvas. Once you have compiled and run with the correct number of input parameters, you should see an error message produced by the code.

3. Use `MPI_Comm_split` to create communicators for process rows and columns. The starter code has a check for process ranks within the communicators you create.

4. Add to the debugging output each process's local matrix. Print the matrix and vector for a small number of rows and columns so that you can compute the result by hand (on the board).

5. Fill in code for the communication of the input vector, the local matrix-vector multiplication, and communication of the output vector. Confirm that the result matches your work by hand for small problems.

6. Perform strong scaling studies for large matrix dimensions to determine the parallel efficiency of your code.

7. Use a submit script (see example provided in Canvas) to scale your code to multiple nodes (no more than 4). Slurm commands `sbatch`, `squeue`, and `sinfo` will come in handy.

8. Fill in code to redistribute your output vector to match the distribution of your input vector.

9. Use `MPI_Wtime` (along with `MPI_Barrier`) to determine how much time is spent in each communication step and how much time is spent in local computation. What is the bottleneck for different matrix dimensions?