Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

MSc. in HPC

HPC Software II (55612)

MPI Programming Exercises 2: RMA and MPI-IO

Darach Golden

April 14, 2025

1 Notes

- **To complete these exercises, submit a document containing results for each question and a separate tarball of the all code used to obtain the results. Details of how to compile and run the code must be provided in a "README" file. You *must* use a makefile to compile the code. The code should compile on `seagull`**
- **Use C to implement all exercises**

2 Exercises

1. Update your version of the Poisson solver which uses a 2D processor decomposition to use active target RMA operations for 2D ghost/halo exchange instead of MPI message passing operations

   - Implement two versions of the 2D ghost exchange routine:
     - [**20%**] A version which uses `MPI_Win_fence` based synchronization along with `MPI_Put` and `MPI_Get`
     - [**30%**] A version which uses general active target synchronization: `MPI_Win_start`, `MPI_Win_complete`, `MPI_Win_post` and `MPI_Win_wait` along with `MPI_Put` and `MPI_Get`
   - [**10%**] Demonstrate that your answers using RMA operations match the answers of your non-RMA version

2. The $20x20$ matrix in Figure 1 is shown as a $4x4$ block matrix consisting of blocks of dimension $5x5$. The binary file `mat-d20-b5-p4.bin` contains this matrix in the following format:

   - The binary file first stores the dimension of the matrix in integer format.
   - Then, in double precision format: the first block column is stored followed by the second block column and so on
   - In each block column the first $5x5$ block is stored followed by the second $5x5$ block and so on

Let the vector x be $x = \{4, 5, 6, 7, 7, 5, 7, 7, 7, 4, 3, 6, 3, 7, 3, 6, 4, 1, 7, 6\}$. This vector is stored in the binary file `x-d20.txt.bin`. The binary file first stores the length (20) of the vector in integer format. Then it lists the vector entries in double precision format.

- **[20%]** Use MPI-IO calls to read the data in the matrix file across 4 processes, where rank 0 reads in the first block column, rank 1 reads in the second block column, and so on. Then use MPI-IO commands to read in the x vector across 4 process so that the sub-vector assigned to each process has the same dimension as the $5x5$ blocks.
- **[20%]** Using the distributed matrix (A) and vector (x) you have just read in, write mpi code to calculate the distributed matrix vector product Ax. The code should run on 4 processors where, as described above, each processor owns a block column of the matrix. Demonstrate that your matrix vector product gives the correct answer by carrying out the matrix vector product in a separate software package such as matlab or python.

Figure 1: Matrix A of dimension 20×20 divided into square block submatrices of dimension 5

$$A = \begin{pmatrix} \begin{pmatrix} 4 & 4 & 3 & 1 & 2 \\ 0 & 4 & 0 & 3 & 0 \\ 4 & 3 & 3 & 2 & 3 \\ 3 & 4 & 3 & 4 & 4 \\ 1 & 1 & 1 & 2 & 1 \end{pmatrix} & \begin{pmatrix} 3 & 4 & 3 & 4 & 1 \\ 2 & 4 & 3 & 2 & 2 \\ 3 & 0 & 4 & 2 & 0 \\ 4 & 0 & 4 & 4 & 4 \\ 0 & 3 & 3 & 1 & 4 \end{pmatrix} & \begin{pmatrix} 3 & 4 & 0 & 1 & 0 \\ 4 & 4 & 2 & 2 & 1 \\ 3 & 4 & 2 & 4 & 4 \\ 4 & 4 & 4 & 2 & 0 \\ 0 & 0 & 0 & 2 & 4 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 & 0 & 4 \\ 3 & 3 & 1 & 4 & 2 \\ 3 & 4 & 0 & 4 & 0 \\ 4 & 1 & 2 & 2 & 2 \\ 4 & 2 & 1 & 4 & 1 \end{pmatrix} \\ \begin{pmatrix} 1 & 4 & 0 & 3 & 2 \\ 4 & 4 & 4 & 1 & 3 \\ 1 & 2 & 0 & 1 & 2 \\ 2 & 4 & 3 & 4 & 1 \\ 2 & 3 & 0 & 0 & 4 \end{pmatrix} & \begin{pmatrix} 2 & 0 & 4 & 1 & 2 \\ 3 & 2 & 2 & 2 & 1 \\ 1 & 3 & 1 & 1 & 3 \\ 4 & 0 & 0 & 3 & 0 \\ 3 & 2 & 3 & 3 & 2 \end{pmatrix} & \begin{pmatrix} 0 & 4 & 3 & 4 & 2 \\ 3 & 0 & 4 & 3 & 3 \\ 4 & 1 & 3 & 4 & 1 \\ 1 & 0 & 4 & 2 & 4 \\ 1 & 0 & 1 & 3 & 0 \end{pmatrix} & \begin{pmatrix} 2 & 3 & 2 & 3 & 0 \\ 4 & 2 & 4 & 3 & 1 \\ 1 & 0 & 0 & 2 & 4 \\ 1 & 2 & 0 & 3 & 2 \\ 3 & 0 & 4 & 1 & 1 \end{pmatrix} \\ \begin{pmatrix} 2 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 2 \\ 4 & 0 & 1 & 0 & 3 \\ 1 & 1 & 0 & 3 & 4 \\ 0 & 0 & 4 & 4 & 4 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 3 & 0 & 3 \\ 0 & 3 & 2 & 1 & 3 \\ 2 & 4 & 1 & 3 & 1 \\ 0 & 3 & 0 & 4 & 4 \\ 3 & 4 & 2 & 4 & 1 \end{pmatrix} & \begin{pmatrix} 0 & 1 & 2 & 2 & 1 \\ 4 & 4 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 & 2 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 2 & 0 & 3 \end{pmatrix} & \begin{pmatrix} 2 & 3 & 3 & 3 & 0 \\ 1 & 3 & 1 & 0 & 1 \\ 1 & 1 & 0 & 4 & 0 \\ 4 & 2 & 0 & 3 & 0 \\ 3 & 3 & 0 & 4 & 1 \end{pmatrix} \\ \begin{pmatrix} 3 & 4 & 3 & 4 & 0 \\ 1 & 2 & 3 & 1 & 3 \\ 3 & 3 & 2 & 1 & 2 \\ 3 & 0 & 3 & 4 & 0 \\ 3 & 3 & 3 & 0 & 2 \end{pmatrix} & \begin{pmatrix} 1 & 3 & 3 & 3 & 3 \\ 2 & 1 & 1 & 1 & 2 \\ 2 & 4 & 0 & 1 & 1 \\ 0 & 0 & 4 & 3 & 1 \\ 4 & 3 & 4 & 1 & 1 \end{pmatrix} & \begin{pmatrix} 3 & 1 & 2 & 2 & 2 \\ 1 & 4 & 4 & 2 & 3 \\ 1 & 1 & 3 & 3 & 3 \\ 4 & 1 & 1 & 0 & 3 \\ 3 & 1 & 3 & 1 & 2 \end{pmatrix} & \begin{pmatrix} 2 & 1 & 2 & 4 & 0 \\ 2 & 2 & 3 & 1 & 0 \\ 1 & 0 & 3 & 3 & 0 \\ 0 & 4 & 2 & 2 & 4 \\ 2 & 4 & 4 & 1 & 1 \end{pmatrix} \end{pmatrix}$$