Course *Parallel and distributed programming*
Computer Lab no. 3: Derived data types. Linear algebra operations

IT, Uppsala University

The batch system SLURM

The reservation for Lab 3 on UPPMAX is `uppmax2023-2-13_2`, so you can use it when submitting your jobs to `slurm`.

```
#!/bin/bash -l

#SBATCH -M snowy
#SBATCH -A uppmax2023-2-13
#SBATCH --reservation uppmax2023-2-13_2
#SBATCH -p core -n ...
#SBATCH -t xx:00
```

Exercise 1 (Derived data types)
You are given a 2D array of size $n \times n$, stored row-wise in a 1D array of length n^2. Assume that you have p processes, organized in a ring. Assume also that $n = k * p$ and the each process has k concequtive rows in its local memory. Thus, the local arrays, seen as 2D are of size $k \times n$.

(a) Define a derived data type 'contiguous', that contains $m, m < k$ consecutive rows. Write a code that sends the last two rows of the locally kept part of the 2D array to your right neighbor.

(b) Communicate a column between P_i and P_{i+1}, using `MPI_Sendrecv`, as illustrated in Figure 1. Use derived data type `MPI_Type_vector`.

(c) Modify the program to communicate the same portion of information if we have a 2D array instead.

You may check the code in `array_sect`.

Exercise 2 (Matrix-vector multiplications)
This problem occurs when solving the so-called *Least square problems*. For completeness we

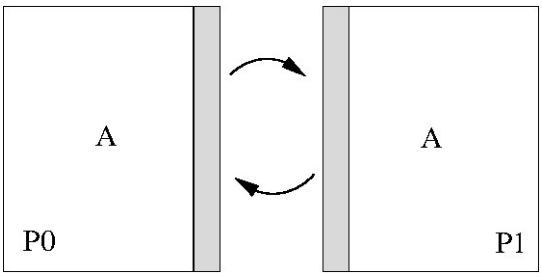

Figure 1: Communicate a column between P0 and P1, assume data stored row-wise in a 1D array

include its mathematical formulation. Given a matrix $A(m,n)$, $m > n$, possibly $m \gg n$, and two vectors $\boldsymbol{x}$ and $\boldsymbol{b}$ of length n and m, correspondingly. We want to find $\boldsymbol{x}$, such that

$$A\boldsymbol{x} = \boldsymbol{b}.$$

Since the latter system is overdetermined, we want to compute the so-called 'minimum-norm' solution $\boldsymbol{x}$, which minimizes $\|A\boldsymbol{x}-\boldsymbol{b}\|_2^2 = \sum_{i=1}^{m}\left(\sum_{j=1}^{n} A_{i,j}\boldsymbol{x}_j - \boldsymbol{b}_i\right)^2$. The problem is referred to as the 'Least squares problem' and the 'least-squares' solution is the unique solution of the system

$$A^T A\boldsymbol{x} = A^T\boldsymbol{b}. \tag{1}$$

Here A^T is the transposed of A. (Here we assume that A has full column rank, thus, the column vectors are linearly independent. Then the matrix $B = A^T A$ is nonsingular.)
When A is of large dimensions we usually use an iterative method to compute $\boldsymbol{x}$ by performing

$$\boldsymbol{w} = A^T A\boldsymbol{x}. \tag{2}$$

Note that we do not explicitly compute B.
Hints regarding the implementation: Let

$$A = \begin{bmatrix} A_0 \\ A_1 \\ \vdots \\ A_{p-1} \end{bmatrix} \text{ and } \boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_{p-1} \end{bmatrix}. \text{ Then } A^T = \begin{bmatrix} A_0^T & A_1^T & \cdots & A_{p-1}^T \end{bmatrix}.$$

Further we have

$$A\boldsymbol{x} = \begin{bmatrix} A_0\boldsymbol{x}_0 \\ A_1\boldsymbol{x}_1 \\ \vdots \\ A_{p-1}\boldsymbol{x}_{p-1} \end{bmatrix}, \; A^T A = A_0^T A_0 + A_1^T A_1 + \cdots + A_{p-1}^T A_{p-1} \text{ and}$$

$$\boldsymbol{w} = A^T A\boldsymbol{x} = A_0^T A_0\boldsymbol{x}_0 + A_1^T A_1\boldsymbol{x}_1 + \cdots + A_{p-1}^T A_{p-1}\boldsymbol{x}_{p-1}.$$

Task: Write an MPI code that performs (2). The code should assume the following:

1. The matrix A is distributed row-wise across the processes and $m = k * p$. The block of A of size $k \times n$ local for a process should be generated locally, using a random number generator (or, for easy check, all entries can be integer numbers). The vector $\boldsymbol{x}$ can also be generated locally, similarly to A.

2. The input parameters should be the number of times to repeat step (2), n, m and k.

3. The matrix A^T should not first be transposed and then distributed.

Compile and run the program using different number of processes, note the timings for each run. What scalability do you observe? How to choose the parameters n, m, k in relation to p?

Exercise 3 (Test using virtual cartesian mesh)
Assume you have defined a 2D virtual cartesian mesh, which is periodic. All eight neighbors are depicted in Figure 2. Here 'NE' stands for 'North-East' etc.

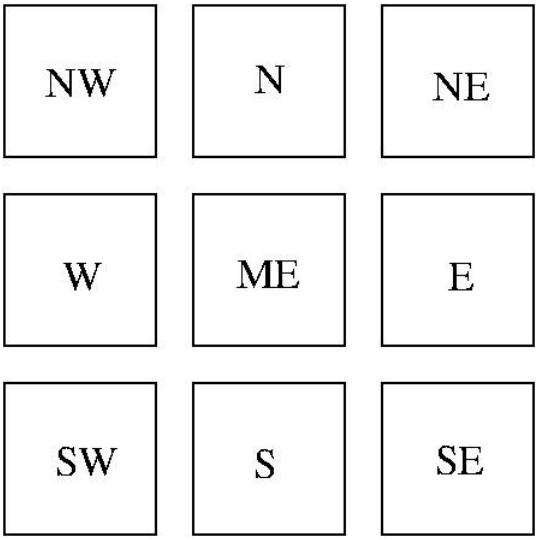

Figure 2: 2D virtual mesh and all 8 neighbors

Using the MPI functions how can you determine the neighboring processes in the four diagonals? In other words, each 'ME' needs to know the process numbers of 'NE', 'NW', 'SE' and "SW'. You have at your disposal `cart_shift_NSEW.c`. It creates local square arrays of size $n \times n$ ($n = 10$ for testing purposes), sets their entries to be equal to 'rank' and illustrates various `MPI_cart*` functionalities, including exchanging the arrays between 'SW' and 'NE'. You may note that using `sendrecv_replace` to exchange along the diagonals causes a deadlock, however, it works fine for exchanges along the axes.
Test with 4 and 9 (possibly 16) processes. (You may see also `sendrecv_replace.c`.)