Programmazione di Sistemi ~~Embedded e~~ Multicore

Teacher: Daniele De Sensi

Exercise 1

Implement matrix-matrix multiplication in MPI. Let’s assume that you run this on **p** processes (from 0 to p-1). The two input matrices are generated randomly by rank **p-1**. The order of the matrices must be read from argv (which means you can use an array but you need to allocate dynamic memory). After the resulting matrix has been computed, it must be stored into rank 0 memory.

- Check if the resulting matrix is correct (e.g., by doing the same computation sequentially)
- Try to think about possible different ways of implementing it

Exercise 2

1. Relying only on point-to-point MPI calls, implement a function with the same signature/arguments and behaviour of MPI_Allreduce (call it «MPI_Allreduce_custom»).
2. Then, measure the its runtime, and compare it with the runtime of MPI_Allreduce.
3. Check if the runtime changes when using Send/Recv compared to Isend/Irecv/Wait
4. Think about how many bytes are you sending and how many send/recv are you executing (similar to what we have done in the class for the broadcast). Can you find a way to do it transmitting only 2*n bytes, and receiving only 2*n bytes (wher n is the number of bytes in the vector to reduce), and performing at most $2*\log_2(p)$ send and $2*\log_2(p)$ recvs (where p is the number of processes)?

Exercise 3

1. You have a 2D matrix **A**, filled with random elements. The number of rows and columns can be different (get them from command line through argv). The matrix is randomly generated by rank 0. After the matrix is generated, you perform a number of iterations (the number of iterations is also read through argv)
2. At each iteration *s* we compute a new matrix A_s such that

$$A_s[i][j] = A_{s-1}[i-1][j] + A_{s-1}[i][j-1] + A_{s-1}[i+1][j] + A_{s-1}[i][j+1]$$

i.e., at each iteration, the value in position (i,j) is the sum of the previous values of the north, south, east, and west neighbors

e.g.,: A_0

-2	5	91
3	8	-5
0	22	3

A_1

	25	
		17

- the orange element in the middle of the matrix will be equal to 25 in step 1 (5+3+-5+22)
- for elements on the border, which do not have one or more of the north/south/east/west neighbor, you can assume those neighbors to have a value of 0. For example, the green element in the bottom right, will become 17 (22 – 5 + 0 + 0)

ATTENTION: Do this by allocating no more than two matrices, regardless of the number of iterations you need to execute. Also, manage the case where the number of elements in the matrix is larger than the number of processes

Exercise 4

The method discussed in the previous exercise is known as *iterative stencil* computation. It is widely used in practice to solve partial differential equations, in computational fluid dynamics, and in many other fields.

Adapt the previous exercise to work on a 3D matrix instead, where each point is computed as the sum of its 6 neighbors as shown in the following picture