

# The MPI Message-passing Standard Lab Time Hands-on

SPD Course  
October 2012  
Massimo Coppola

# What was expected so far

- Prepare for the lab sessions
  - Install a version of MPI which works on your O.S.
    - OpenMPI (active development)
    - LAM MPI (same team, only maintained)
    - MPICH (active development)
  - Check out details that have been skipped in the lessons
    - How to run programs, how to specify the mapping of processes on machines
    - Usually it is a file listing all available machines
    - How to check a process rank
  - Read the first chapters of the Wilkinson-Allen
    - Write at least a simple program that uses MPI\_Comm\_World, has a small fixed number of processes and communications and run it on your laptop
    - E.g. a trivial ping-pong program with 2 processes

# Exercise 1

- Define the classical ping-pong program with 2 processes
  - they send back and forth a data buffer, the second process executes an operation on the data (e.g. sum 1).
  - Verify after a given number  $N$  of iterations, that the expected result is achieved.
  - Add printouts close to communications
  - Does it work? Why?

# Remember!

- Simplest programs do not need much beyond Send and Recv, still...
- Each process lives in a separate memory space
  - Need to initialize all your data structures
  - Need to initialize **your instance of the MPI library**
  - Use `MPI_COMM_WORLD`
  - Need to define all your DataTypes
  - Should you make assumptions on process number?
  - How portable will your program be?
- Check your MPI man page about launching
  - E.g. **`mpirun -np 4 myprogram parameters`**

# Initializing the runtime

- `MPI_Init()`
  - Shall be called before using any MPI calls (very few exceptions)
  - Initializes the MPI runtime for all processes in the running program, some kind of handshaking implied
    - e.g. creates **`MPI_COMM_WORLD`**
  - check its arguments!
- `MPI_Finalize()`
  - Frees all MPI resources and cleans up the MPI runtime, taking care of any operation pending
  - Any further call to MPI is forbidden
  - some runtime errors can be detected at finalize
    - e.g. calling finalize with communications still pending and unmatched

# Getting your identity

- MPI\_Comm\_rank
  - After the MPI\_Init
  - Returns the rank of the current process within a specified communicator
  - For now let's just use ranks related to MPI\_COMM\_WORLD
  - Example:

`MPI_Comm_rank(MPI_COMM_WORLD, &myrank);`

# Exercise 2

- Build datatypes for
  - a square matrix of arbitrary element types and constant size  $120 \times 120$
  - a column of the matrix
  - a row of the matrix
  - a group of 3 columns of the matrix
  - the upward and downward diagonals of the matrix
- Perform a test of the datatypes within the code of exercise 1

# Remember

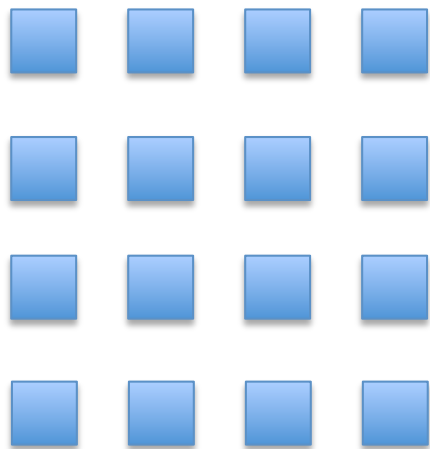
- `MPI_TYPE_COMMIT(datatype)`
  - Mandatory to enable a newly defined datatype for use in all other MPI primitives
  - Consolidates datatype definition, making it permanent
  - May compile internal information needed to the MPI library runtime
    - e.g. : optimized routines for data packing & unpacking
- `MPI_TYPE_FREE(datatype)`
  - Free library memory used by a datatype that is no longer needed



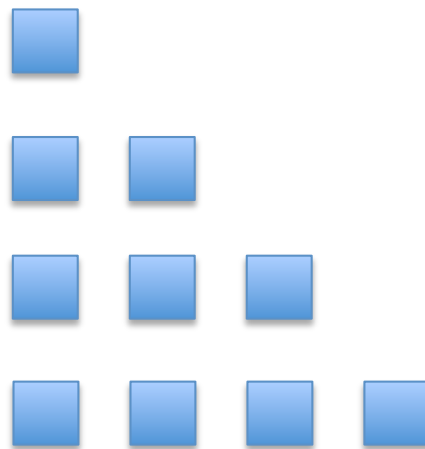
# Exercise 3

- Define a datatype for a square matrix **with parametric size**
  - Define a datatype for its lower triangular matrix
  - Define one for its upper triangular.
- Test the them within the code of exercise 1

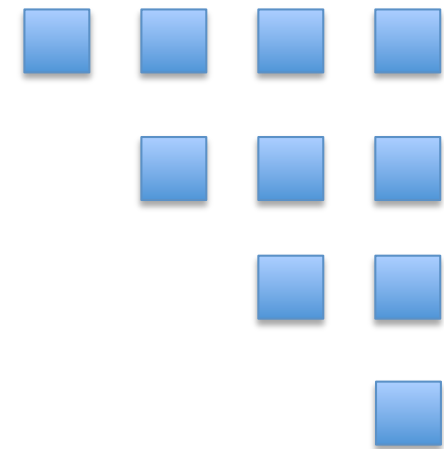
$A_{i,j} \quad i,j \text{ in } 1..n$



$A_{i,j} \quad i \geq j$



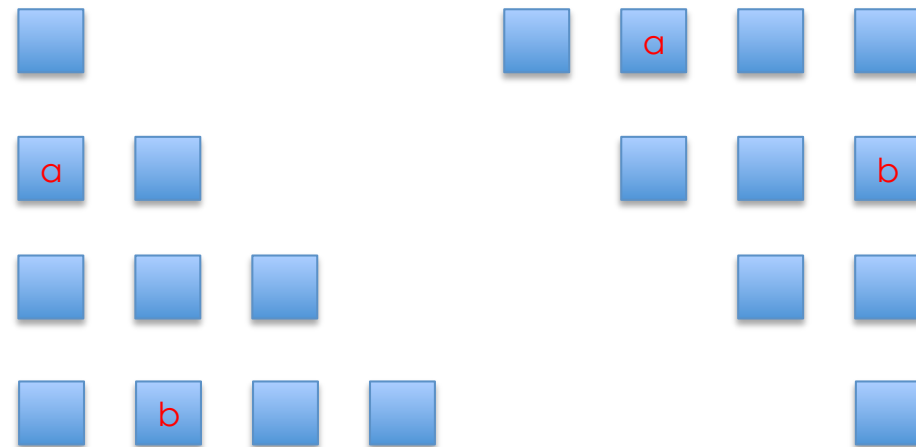
$A_{i,j} \quad i \leq j$



# Exercise 3 (II)

- In the two-process program
  - initialize randomly a square matrix
  - send the lower triangular and
  - receive it back as upper triangular in the same buffer.
- Is the result a symmetric matrix?
  - How do you need to modify one of the two triangular datatypes in order to achieve that?

- In the end we want  $A_{i,j} = B_{j,i}$



# Exercise 4

- How do you implement an asynchronous communication with given asynchrony?
  - Implement a communication with asynchrony 1
  - Implement a communication with asynchrony  $K$
- Assigned asynchrony of degree  $K$ : asynchronous communication (sender does not block) which becomes synchronous if more than  $K$  messages are still pending.
- Receiver can skip at most  $K$  receives before sender blocks
- Can you rely on MPI buffering?
- How would you implement a fixed size buffer?

# Exercise 5

- Define a program with  $>10$  proc.s and some communicators
  - Even/odd numbers
  - Apply hierarchically until `comm_size>1`
  - Can you implement a broadcast?
  - Define communicators for a pipeline of two farms