**TP MPI**

**Exercice1 :**

Compile and execute the code 01-txrx.c. Run it using 4 processes. Then, modify the code to answer the following questions:

1. Run the code using 5 processes. What happens? Can you explain it?
2. Modify the code so that it works with any number of processes.
3. What happens if the tag value in the MPI\_Recv function is different from MPI\_TAG\_VALUE used by the sender? Modify the code so that processes can receive any message regardless of the tag value and the sender.
4. For processes who receive the data, use the status variable to ﬁnd out who sent the message and print the sender id (or rank) on screen.

**Exercice 2 :**

Modify the previous code so that the sender process sends an array of integers **dynamically allocated** (that is, using the malloc or calloc functions). There will be as many receivers as elements are in the array. The receiver with rank 1 will receive the ﬁrst element of the array; the receiver with rank 2 will receive the second element, and so on.

**Exercice 3 :**

Write a MPI program to perform a cyclic ring communication on an even number of processors (let us say four), so that each process sends its id to the next process. Each process will show on screen its own id as well as the received value. Note that, since it is a cyclic communication, the last process will send its value to the ﬁrst process. Note also that, with our current knowledge of MPI, we cannot perform the cyclic communication on a single step.

**Exercice 4 :**

Using two matrices A[NR A][NC A] and B[NC A][NCB] dynamically allocated, write a matrix product C = A ∗ B in MPI with p processors. Use a master-slave approach according to the following procedure:

1. The master process will allocate and initialise the matrices.
2. Then, it will divide the rows of A in as many chunks as workers (number\_of\_workers = num\_processes - 1). If the number of rows is not divisible by the number of workers, it will balance the chunk size as much as possible.
3. Afterwards, the master will send to each worker the chunk size (number of rows); i.e.: the corresponding chunk of A and a whole copy of B.
4. Each worker process will receive its chunk of A and a copy of B. it will perform the partial matrix product and will return the result to the master.
5. The master will receive each chunk of C and will put everything together.

You can use the 04-mpi\_mm\_skel.c code.