TP - Distributed-memory programming: MPI (I)

ING2-GSI-MI – Architecture et programmation parallèle
Academic year 2023–2024

Kick off

You should already have the OpenMPI library installed on your laptops. Check it by typing in the command line: `mpicc -v`. If it returns some information about your MPI configuration, you are good to go. Otherwise, install the required packages using the following line:

```
# apt-get install openmpi-bin openmpi-common libopenmpi-dev
```

To compile your codes, use the following line:

```
mpicc -Wall -Wextra -o myprogram myprogram.c
```

To run your code with N processes, use the following command:

```
mpirun -np N ./myprogram
```

Exercises

1 Try the C codes from the course examples (`codes_MPI-1.zip` file in Teams) and make sure that they work as intended. □

2 Compile and execute the code `01-txrx.c` from the `skeletons.zip` file in Teams. 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.

□

3 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. □

4 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. □

5 Using two matrices $A[NRA][NCA]$ and $B[NCA][NCB]$ 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 up 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), 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. They 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 from the `skeletons.zip` file. □