

# Travaux Pratique 9

Ce sujet est en lien avec le cinquième chapitre du cours, et concerne la programmation répartie. Les mêmes commentaires que ceux des derniers TP s'appliquent ici aussi.

**MPI** (pour *Message Passing Interface*) est l'une des méthodes les plus courantes permettant la réalisation d'un logiciel destiné à fonctionner sur HPC (pour *High Performance Computer*, généralement des ordinateurs composés de plusieurs unités centrales communiquant via des réseaux dédiés). MPI est une interface, autrement dit une définition des fonctionnalités, fonctions, structures etc. permettant de communiquer entre processeurs sur un réseau. Il ne s'agit pas d'une implantation.

Plusieurs implantations de MPI existent, dont **mpich** qui est installée en salles de TP 1.1 et 1.4 du BE sous Windows. En pratique, vous trouverez la version de Microsoft basée sur **mpich**, et nommée MS-MPI.

Pour effectuer une installation chez vous sur **Windows**, il suffit d'installer la bibliothèque et les exécutables proposés ici (installer les deux paquets proposés, le MSI et le EXE) :

<https://docs.microsoft.com/en-us/message-passing-interface/microsoft-mpi>

La documentation est accessible sur le même site. Sous **MacOS** et **Linux**, installez simplement les paquets nécessaires au fonctionnement, à la compilation et la documentation (`sudo apt install mpich mpich-doc libmpich-dev`).

Attention : un programme MPI ne se lance pas directement, mais uniquement via la commande **mpiexec** (c'est elle qui effectue le déploiement sur les machines accessibles et spécifiées).

Votre travail est à rendre (il le sera à chaque fois) sous la forme du code (et uniquement la partie « student ») accompagnée d'un rapport au format PDF, le tout dans une archive compressée au format ZIP. Ne respectez pas ces contraintes, et votre note en sera diminuée de quelques points.

**Vous ne devez modifier que ce qui se trouve dans le répertoire « ./student/ » !**

Notez que chaque exercice vient avec un squelette, qui s'occupe de la ligne de commande, du lancement de votre code (défini dans une classe particulière), et d'une vérification sommaire du résultat (lorsque c'est possible). Utilisez l'option « -h » ou « --help » pour connaître le fonctionnement de la ligne de commande ...

## Exercice 1 : Premier pas : compiler et exécuter

Dans ce premier exercice, vous allez sous Windows compiler un premier programme permettant l'affichage du numéro de processeur courant (code fourni). Puis, vous devez lancer l'application générée sur au moins quatre processeurs, via le flag **-np 4** (en séance de TP, n'utilisez qu'une seule machine ; par-contre, vous testerez sur plusieurs machines lorsque les salles seront moins occupées ...). Testez plusieurs fois, afin de vérifier l'aléa temporel ...

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
int main (int argc , char ** argv ) {
    // Initialize the MPI environment
    MPI_Init (NULL, NULL);
    // Get the number of processes
    int p;
    MPI_Comm_size (MPI_COMM_WORLD, &p);
    // Get the rank of the process
    int q;
    MPI_Comm_rank (MPI_COMM_WORLD, &q);
    // Get the name of the processor
    char processor_name [MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name ( processor_name , &name_len ) ;
    // Print a hello world message
    printf ( "Hello world from processor %s,"
            " rank %d out of %d processors\n",
            processor_name, q, p);
    // Finalize the MPI environment, and quit
    MPI_Finalize() ;
    return EXIT_SUCCESS;
}
```

Attention : n'oubliez pas d'étudier la documentation des fonctions MPI utilisées par ce programme ... et surtout utilisez `mpiexec` !

## Exercice 2 : Communications entre processus

Pour communiquer sur un anneau, les processus doivent utiliser les méthodes `Send` et `Recv`. Si la signature de ces fonctions est verbeuse, sachez que c'est le cas de la plupart des fonctionnalités de MPI. Ces fonctions partagent toutes une utilisation similaire, qui, une fois comprise et maîtrisée, se révèle relativement facile d'emploi.

Voici un petit d'exemple d'utilisation permettant d'envoyer un entier depuis le processeur 0 à destination des autres processeurs de l'anneau.

```
int receiveOrInitIfFirstProc(const OPP::MPI::Ring &ring)
{
    // first processor?
    if (ring.getRank() == 0)
        // yep, so return the answer
        return 0b101010;

    // no, so receive from previous one
    int number = -1;
    ring.Recv(&number, 1, MPI_INT);
    return number;
}

void sendToNextProcIfNotLast(const OPP::MPI::Ring &ring, int number)
{
    // if last processor, just do nothing
    if (ring.getNext() == 0)
        return;
    // not last, so send to next processor
    ring.Send(&number, 1, MPI_INT);
}

int test(const OPP::MPI::Communicator &comm)
{
    if (comm.size < 2)
        return -1;

    OPP::MPI::Ring ring(comm.communicator);
    sendToNextProcIfNotLast(ring, receiveOrInitIfFirstProc(ring));
    return 0;
}
```

Ecrivez un programme réalisant cette communication simpliste ; testez ce programme avec 2 processeurs, puis avec plus. Attention : que manque-t-il ?

## Exercice 3 : Diffusion sur un anneau (unidirectionnel)

Dans cet exercice, vous utiliserez une topologie en anneau de  $p$  processeurs. Autrement dit, chaque processeur de numéro  $q$  peut envoyer un message au processeur de numéro  $q + 1$  (modulo  $p$ ), et recevoir du processeur  $q - 1$  (modulo  $p$ ). N'oubliez pas que  $p$  est donné en ligne de commande à `mpiexec` (via le flag `-np`).

En utilisant ces fonctions, réalisez une diffusion sur un tel anneau, bien entendu sans utiliser la fonction `MPI_Broadcast`.

Testez les deux variantes vues dans le cours 6 :

- En version naïve.
- En pipeline.

En ligne de commande, vous pouvez indiquer la taille de la donnée à transmettre, ainsi que la taille du bloc en version pipeline (remarquez l'erreur sur le premier lancement) :

```
PS D:\...\TP9\sources> .\build\Release\PipeBroadcast.exe -en
29 -em 10
Broadcast from 0 an array of 536870912 integers ...
[0] Done in 0.0019ms.
[0] Good, your broadcast seems to work ;-)
PS D:\...\TP9\sources> mpiexec -np 2 .\build\Release\NaiveBroa
dcast.exe -e 29
Broadcast from 0 an array of 1024 integers ...
[1] Done in 0.4106ms.
[0] Done in 0.4085ms.
[1] Good, your broadcast seems to work ;-)
[0] Good, your broadcast seems to work ;-)
Broadcast from 1 an array of 1024 integers ...
[0] Done in 0.1741ms.
[1] Done in 0.1936ms.
[0] Good, your broadcast seems to work ;-)
[1] Good, your broadcast seems to work ;-)
PS D:\aveneau\Enseignement\Master\M1\APR\TP\TP9\sources>
```