# 1 Prise en main de MPI

### Exercice 1.1 : Premiers pas avec MPI

1. Créez un fichier contenant le code du ping-pong vu dans le cours
2. Compilez-le avec `mpicc`
3. Exécutez-le sur plusieurs machines de la salle de TP

### Exercice 1.2 : Communications non-bloquantes

1. Reprenez le ping-pong de l'exercice précédent et modifiez-le pour utiliser des communications non-bloquantes `MPI_Isend` et `MPI_Irecv`.
2. Où placez-vous les `MPI_Wait` ?
3. Modifiez le code pour effectuer non pas un ping-pong mais un envoi simultané. Auriez-vous pu le faire avec des communications bloquantes ?

### Exercice 1.3 : Anneau

Prenons un ensemble de processus MPI communiquant en suivant une topologie d'anneau.

1. Si un processus de cet ensemble a le rang $k$, quel est le rang du processus à qui $k$ va envoyer des messages ?
2. Quel est le rang du processus de qui $k$ va recevoir des messages ?
3. Comment la circulation du jeton doit-elle être initiée ?
4. Comment la circulation du jeton se termine-t-elle ?
5. Implémentez en MPI une circulation de jeton (un entier) selon un anneau entre des processus. Votre programme doit fonctionner quel que soit le nombre de processus.

# 2 Communications collectives en MPI

### Exercice 2.1 : Diffusion

Écrivez un programme en MPI dans lequel le processus de rang 0 récupère son *process id* et le diffuse à tous les autres processus. Les processus doivent alors tous afficher cette valeur.

### Exercice 2.2 : Réduction

Écrivez un programme en MPI qui se comporte comme suit :

1. Chaque processus alloue un tableau de 1014 éléments
2. Chaque processus remplit le tableau avec des valeurs aléatoires comprises entre 0 et 100 (attention à l'initialisation de votre générateur de nombres aléatoires, qui doit être seedée différemment sur chaque processus)
3. Chaque processus calcule la valeur maximum contenue dans ce tableau
4. En utilisant une opération collective, récupérez le maximum des maxima locaux dans le processus de rang 0.

### Exercice 2.3 : Scatter-Gather

**Question 1 :** Écrivez un pogramme en MPI qui se comporte comme suit :

1. Le processus de rang 0 alloue un tableau dont le nombre d'éléments est égal au nombre de processus multiplié par 1024
2. Il remplit alors le tableau de valeurs aléatoires comprises entre -100 et 100
3. En utilisant une opération collective, ce tableau est distribué à tous les processus du système
4. Chaque processus a alors une portion du tableau. Chacun élève au cube toutes les valeurs comprises dans ce tableau.
5. Chaque processus calcule la moyenne contenue dans son tableau.
6. En utilisant une opération collective, récupérez l'ensemble des moyennes dans un tableau contenu dans le processus de rang 0 (attention à l'allocation mémoire).
7. Le processus de rang 0 calcule la moyenne des moyennes récupérées.

**Question 2 :** Comparez le temps de calcul de l'implémentation parallèle que vous venez d'écrire avec une implémentation séquentielle.

### Exercice 2.4 : Implémentation de la diffusion

**Question 1 :** En utilisant uniquement des opérations d'envoi et de réception, implémentez la diffusion d'un entier en utilisant l'algorithme de l'étoile.

**Question 2 :** Même question en utilisant un arbre binomial.

### Exercice 2.5 : Implémentation de la distribution

**Question 1 :** En utilisant uniquement des opérations d'envoi et de réception, implémentez la distribution d'un entier en utilisant l'algorithme linéaire.

**Question 2 :** Même question en utilisant un arbre binomial.

**Question 3 :** Même question en utilisant un arbre binaire.

## Exercice 2.6 : Implémentation du all-to-all

En utilisant uniquement des opérations d'envoi et de réception (vous pourrez utiliser des communications non-bloquantes), implémentez un all-to-all où chaque processus envoie un entier différent à chaque autre processus.

# 3 Applications parallèles avec MPI

### Exercice 3.1 : Normalisation d'une matrice

Le but de cet exercice est de normaliser les données d'une matrice, c'est-à-dire de diviser la valeur de tous ses éléments par la valeur du plus grand élément.

1. Créez un programme parallèle avec MPI dans lequel les processus disposent chacun d'une sous-matrice aléatoire de taille MxN.
2. Sur chaque processus, recherchez la valeur du plus grand élément de la sous-matrice.
3. En utilisant une opération collective, calculez la valeur du maximum global. Tous les processus doivent avoir cette valeur.
4. Sur chaque processus, divisez la valeur des éléments de la sous-matrice par le maximum global.

### Exercice 3.2 : Type dérivé I

1. Créez un type dérivé en MPI contenant un ensemble d'entiers contigus
2. Écrivez un programme où deux processus s'échangent un tableau d'entiers, en utilisant le type que vous venez de définir. Le nombre d'éléments envoyés doit être égal à 1, le type de données est votre type dérivé.

### Exercice 3.3 : Type dérivé II

1. Créez un type dérivé en MPI contenant des entiers non-contigus :
   - un entier par bloc
   - le stride (écart entre le début d'un bloc et le début du suivant) est égal à un paramètre N
   - le nombre de blocs est égal à un nombre M
2. Écrivez un programme où deux processus disposent chacun d'une matrice de M lignes et N colonnes stockée linéairement dans un tableau à une dimension. Ces processus s'échangent les premiers éléments de chaque ligne :
   (a) Implémentez-le avec des communications envoyant les éléments un par un. Combien de messages cette méthode implique-t-elle ? Quel est le volume de données (total et par message) envoyées ?
   (b) Implémentez-le en utilisant le type dérivé que vous venez de définir. Comparez le nombre de message et le volume de données transportées avec l'implémentation précédente.

### Exercice 3.4 : Transposition de matrice

Le but de cet exercice est de transposer efficacement une matrice distribuée sur des processus parallèles.

1. Écrivez un programme parallèle avec MPI dans lequel deux matrices sont réparties sur les processus en étant stockées linéairement dans des tableaux à une dimension. La première matrice est composée de M lignes et N colonnes, la deuxième est composée de N lignes et M colonnes. La matrice est distribuée par décomposition de domaine selon les lignes : ainsi, chaque processus a une matrice (M/P)xN et une matrice (N/P)xM. Pour des raisons de simplicité, prenez N=M=P. Ainsi, chaque processus dispose d'une ligne de chaque matrice.

2. En utilisant les types dérivés définis dans les exercices précédents, modifiez votre programme afin que chaque processus envoie le premier élément de chaque ligne de sa matrice au processus 0, qui stocke tous ces éléments de façon contiguë dans la ligne de la matrice dont il dispose. Pour cela, vous utiliserez une opération collective en étant particulièrement attentif aux types de données utilisés en envoi et en réception (indication : ils ne sont pas identiques).
3. En utilisant une autre opération collective, modifiez votre programme pour que les $k$-ièmes éléments de la matrice soient rassemblés sur le processus $k$. L'opération effectuée sur la matrice est une transposition.
4. Pour utiliser une matrice de taille quelconque, quelle doit être la taille des matrices contenues sur chaque processus ?
5. Quels doivent être les paramètres des types de données dérivés qui doivent alors être utilisés (nombre d'éléments, stride, nombre de blocs) ?
6. Modifiez votre programme pour transposer des matrices de taille quelconque.

## Exercice 3.5 : Maître-esclave statique

Implémentez un squelette de maître-esclave statique en utilisant des opérations collectives pour distribuer les données et récupérer les résultats.

## Exercice 3.6 : Maître-esclave dynamique

Implémentez un squelette de maître-esclave dynamique en mode pull : les esclaves demandent du travail au maître quand ils envoient un nouveau résultat ou à l'initialisation, et celui-ci leur envoie des données (un tableau d'entiers) tant qu'il en a.

# 4 Jeu de la vie parallèle