

# Calcul Haute Performance

## TP - Introduction au MPI

Oguz Kaya  
oguz.kaya@lri.fr

Pour compiler un code `programme.cpp` avec MPI et générer l'exécutable `programme`, saisir

```
mpic++ -O2 -std=c++11 programme.cpp -o programme
```

et pour l'exécuter avec, par exemple, 2 processus, saisir

```
mpirun -np 2 ./programme
```

Part 1

**Hello world!**

### Exercise 1

- a) Implantez un programme MPI qui affiche "Hello World", le rang et le nombre total de processus dans chaque processus.

Part 2

**Ping-Pong**

Ecrivez un programme MPI qui effectue les tâches suivantes:

### Exercise 2

- a) Processus de rang pair : envoyer un message contenant le rang du processus courant au processus impair correspondant (0 envoie à 1, 2 à 3 ...). Recevoir le message du processus impair et l'afficher.
- b) Processus de rang impair : recevoir le message du processus pair associé, puis envoyer un message contenant la valeur reçue plus dix fois le rang du processus courant.

Part 3

**Calcul de  $\pi$**

Le nombre  $\pi$  peut être défini comme l'intégrale de 0 à 1 de  $f(x) = \frac{4}{1+x^2}$ . Une manière simple d'approximer une intégrale est de discrétiser l'ensemble d'étude de la fonction en utilisant  $N$  points.

On considère l'approximation suivante avec  $s = \frac{1}{N}$  :

$$\pi \approx \int_0^1 \frac{4}{1+x^2} dx \approx \sum_{i=0}^{N-1} s \times \frac{f(i \times s) + f((i+1) \times s)}{2}$$

### Exercise 3

- a) Implantez un programme MPI qui calcule le  $\pi$  en parallèle avec  $P$  processus tel que chaque processus effectue le calcul concernant  $N/P$  valeurs consécutives de  $i$ , puis fusionne son résultat avec ceux des autres. Au final, veillez à ce que tous les processus aient la valeur du  $\pi$ . Vous pouvez profiter du code squelette fourni `calcul-pi.cpp`

Part 4

**Mergesort**

Le but de cet exercice est de réaliser une implantation parallèle de mergesort en MPI. Un code squelette `mergesort.cpp` à compléter est déjà fourni pour ce faire. Chaque processus commence par un tableau  $A$  de taille  $N$ , qui est déjà trié dans le code squelette.

*Exercise 4*

- a) Implantez un programme MPI qui fusionne ses tableaux deux par deux dans les premiers  $P/2$  processus, puis dans les premiers  $P/4$  processus,  $\dots$ , et finalement dans le processus 0, qui contiendra le tableau final trié  $A$  de taille  $NP$ . Pour fusionner deux tableaux, utiliser la fonction `merge(...)` fourni dans le code squelette. Supposer que le nombre de processus est toujours une puissance de deux. Ne pas hésiter à modifier la taille de  $A$  dans un processus quand il le faut.

Part 5

**Broadcast (diffusion) d'un tableau***Exercise 5*

- a) Réaliser la fonction `void MPI_BcastInt(int *tab, int count, int root, MPI_Comm comm)` qui envoie le tableau d'entiers `tab[count]` du processus de rang `root` au tableau `tab[count]` dans tous les autres processus. Tous les processus appelleront ensemble la fonction `MPI_BcastInt` avec un `tab[count]` déjà alloué mais seulement le processus ayant le rang `root` l'aura initialisé.
- b) Réaliser la fonction `void MPI_BcastIntAnneau(int *tab, int count, int root, MPI_Comm comm)` qui fait la même chose, cette fois-ci dans une topologie d'anneau tel que chaque processus reçoit le tableau du processus précédent (au niveau du rang) et le renvoie au processus suivant. Faire avec `MPI_Isend` et `MPI_Recv` tel qu'on a le même code à exécuter depuis chaque processus (i.e., sans savoir le branchement en fonction de son rang).