



## TP 1 : Premiers pas avec MPI

Ce premier TP propose différents exercices pour expérimenter les fonctions de base de communication (point à point et collectives).

Il n'est pas nécessaire de terminer tous les exercices en séance, par contre il est fortement conseillé de tous les terminer en dehors de cette séance.

L'évaluation de ce cours en Bureau d'Étude, temps limité, aura des exercices du même genre.

### Mise en Place de l'environnement MPI

Nous fournissons dans le répertoire TP1, le fichier `init_salleTP.sh` qui permet de positionner un certain nombre de choses pour ce premier TP MPI.

Il faut commencer par exécuter

```
source init_salleTP.sh
```

Vérifier (en exécutant `echo $PATH`) que votre variable d'environnement `PATH` contient à son début le répertoire où sont installés les exécutables de SIMGRID :

```
/mnt/n7fs/ens/tp_guivarch/opt2023/simgrid-3.32/bin
```

En tapant la commande `alias`, assurez-vous que la commande pour lancer les processus MPI `Smpirun` existe et complète la commande `smpirun` avec les paramètres donnant l'architecture cible. Ceci a été réalisé dans un souci de simplifier l'exécution des programmes parallèles.

ATTENTION : vous devez ré-exécuter cette initialisation à chaque ouverture d'un nouveau terminal si vous voulez avoir accès aux exécutables de SIMGRID (avantage : cela disparaît à chaque fois que vous fermez le terminal et cela ne pollue pas d'autres enseignements).

### Compiler et exécuter le code parallèle

Pour chaque exercice, il faudra se placer dans le répertoire correspondant, compléter le code, le compiler `make`

puis lancer les processus MPI exécutant l'exécutable `exe`<sup>1</sup> sur les processeurs de l'architecture simulée, par exemple avec 4 processeurs :

```
Smpirun -np 4 ./exe
```

---

1. nom différent à chaque exercice

## 1 Hello World

Répertoire de développement: **00\_Who\_am\_i**

1. complétez le code fourni (vous pouvez vous appuyer sur ce qui a été fait en cours)
  - en initialisant l'environnement MPI,
  - en récupérant le rang du processus courant, le nombre total de processus MPI lancés, le nom de la machine sur lequel tourne le processus courant
  - en n'oubliant pas de détruire l'environnement MPI
2. le compiler  
**make**
3. tester le code avec 4 processeurs  
**Smpirun -np 4 ./who\_am\_i**
4. vérifier que vous pouvez monter jusqu'à 256 processeurs (nombre de processeurs du fichier cluster\_hostfile.txt, caché dans l'alias de **Smpirun**). Que se passe-t-il avec SIMGRID quand on dépasse cette valeur ?

## 2 Communication en Anneau

Répertoire de développement: **01\_Ring**

Nous disposons de  $N$  noeuds MPI, et on souhaite faire transiter un message en anneau, du noeud 0 vers le noeud 1 puis du noeud 1 vers le noeud 2, ..., du noeud  $N - 1$  vers le noeud 0.

Ce message est constitué d'une valeur qui sera multipliée par deux par chaque noeud. Si la valeur initiale envoyée par le noeud 0 est 1, lorsqu'on aura fait le tour de l'anneau, le noeud 0 recevra un message avec la valeur  $2^{N-1}$ .

- complétez le code du fichier **ring.c** en suivant les instructions données en commentaires

## 3 Quand le MPI\_Send bascule du mode asynchrone au mode synchrone ?

Répertoire de développement: **02\_Limite**

Il a été mentionné en cours que la routine **MPI\_Send** pouvait avoir un comportement synchrone ou asynchrone suivant la taille des messages qui étaient transmis.

On se propose d'évaluer cette taille limite de manière empirique.

1. Complétez le code du fichier **limite.c**, en suivant les commentaires inclus, pour que deux noeuds s'échangent des messages constitués par un certain nombre d'entiers. Ce nombre d'entiers est donné en paramètre de la ligne de commande.  
**Smpirun -np 2 ./limite 100**

2. Insérer, sous forme de commentaires, en fin de fichier les réponses aux questions suivantes :
  - (a) rappelez pour quelle taille de message (petite, grande), `MPI_Send` aura un comportement asynchrone (resp. synchrone)
  - (b) que va-t-il se passer quand votre programme, complété comme indiqué, sera appelé avec une taille de message qui fera que `MPI_Send` sera synchrone ?
  - (c) estimez à 10 entiers près, la taille limite ?
  - (d) proposez une solution pour que l'échange entre les deux noeuds puissent se faire au delà de cette limite (plusieurs réponses possibles). Vous avez la possibilité de les tester en dehors de la séance.

## 4 Produit scalaire distribué

Répertoire de développement: **03\_Dot**

Nous disposons de  $N$  noeuds MPI, et on souhaite effectuer le produit scalaire entre deux vecteurs  $x$  et  $y$  de même taille  $n$  ; ces deux vecteurs, dans notre exercice, contiennent les mêmes valeurs, de 0 à  $n - 1$ .

Chaque noeud dispose d'une partie de ces deux vecteurs.

Les parties de  $x$  et de  $y$  présentes sur un noeud sont conformes et ont, dans notre exercice, une taille de 5. Ainsi, par exemple, le noeud 1 dispose des composantes de 5 à 9 des deux vecteurs (numérotation du langage C à partir de 0).

On se propose de coder deux façons d'effectuer le produit scalaire global, la première qui se fera en deux étapes et la seconde en 1 étape.

- complétez le code du fichier `dotp.c` en suivant les instructions données en commentaires. Les deux solutions pouvant être traitées l'une après l'autre.

## 5 Produit Matrice-Vecteur distribué

Répertoire de développement: **04\_Mult**

Nous souhaitons réaliser un produit matrice  $\times$  vecteur en parallèle.

Pour cela la matrice  $A$  (carrée  $n \times n$ ) va être distribuée sur  $N$  process suivant un découpage par **blocs de lignes** (voir Figure 1).

Avec la taille donnée  $n$  en constante (12) dans le sujet et le nombre de process imposés (4), le nombre de lignes,  $b$ , affectées à chaque process est le même (3).

Chaque process calcule la partie de  $x = A * v$  correspondant à sa partie de  $A$  (calcul illustré pour le process 1 par la Figure 2).

À la fin du calcul, il est intéressant qu'un process connaisse le  $x$  complet (dans notre exercice ce sera le process 2).

- complétez le code du fichier `MultAv.c` en suivant les instructions données en commentaires.
- on pourra réfléchir aux communications de données (en amont et en aval du calcul) si, maintenant, la matrice est distribuée en **blocs de colonnes**.

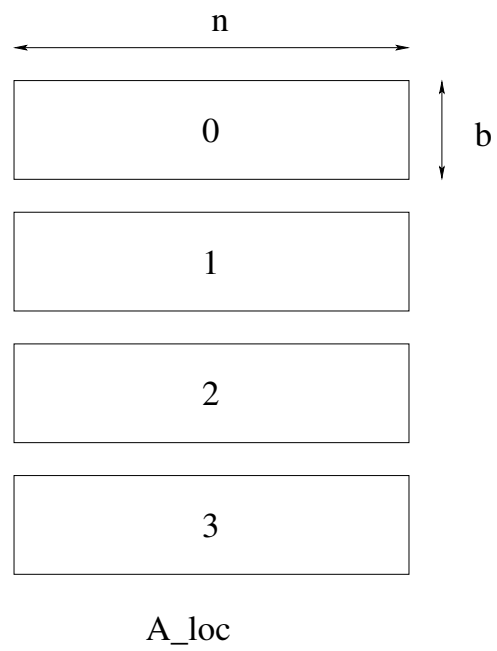


FIGURE 1 – distribution de  $A$  en blocs de lignes

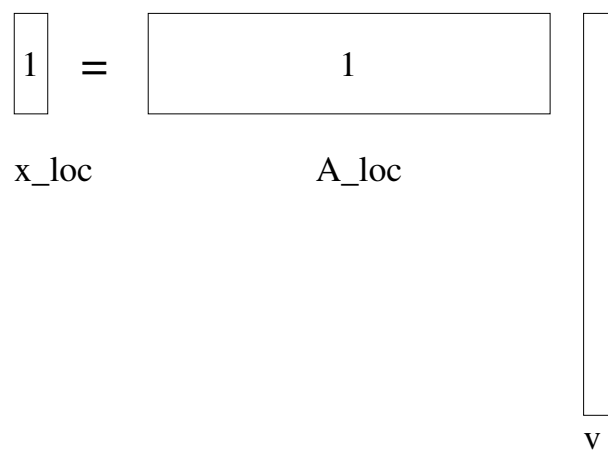


FIGURE 2 – calcul du process 1

## 6 Parallélisation de certains calculs de la méthode du Gradient Conjugué (CG, Conjugate Gradient)

Répertoire de développement: 05\_CG

On se propose de paralléliser cette méthode de résolution d'un système linéaire  $A.x = rhs$ .

Vous pourrez noter (en étudiant attentivement l'algorithme) que, dans le Gradient Conjugué, on effectue deux produits scalaires ainsi qu'un produit matrice-vecteur<sup>2</sup>. Il sera donc judicieux de ré-utiliser les mécanismes de communications des 2 exercices précédents.

---

### Algorithm 1 a formulation of CG

---

```

1: Input :  $A$  SPD matrix,  $rhs$ , right-hand side,  $\varepsilon_{tol}$ 
2: Output :  $x$ , solution of  $A.x = rhs$ ,  $\|rhs - A.x\|_2 / \|rhs\|_2 < \varepsilon_{tol}$ 
3:
4: Set the initial guess  $x = 0$ 
5:  $r = rhs$ 
6:  $p = r$ 
7:  $nr = \|r\|_2$ 
8:  $\varepsilon = nr * \varepsilon_{tol}$ 
9:  $j = 0$ 
10: while ( $nr > \varepsilon$ ) and ( $j < max\_it$ ) do
11:    $np2 = (A.p, p)$ 
12:    $\alpha = (nr * nr) / np2$ 
13:    $x = x + \alpha p$ 
14:    $r = r - \alpha A.p$ 
15:    $nr = \|r\|_2$ 
16:    $\beta = (nr * nr) / (\alpha * np2)$ 
17:    $p = r + \beta p$ 
18:    $j = j + 1$ 
19: end while
```

---

Les données sont, comme dans l'exercice précédent, distribuées sur les noeuds par blocs de lignes (Figure 3).

Le fichier à modifier pour cette parallélisation est le fichier `CG_par.c`. Comme pour les exercices précédents, vous êtes guidés par les commentaires inclus dans le code.

---

2. un produit scalaire est caché, les deux produits matrice-vecteur visibles se ramènent à un seul en stockant le vecteur résultat

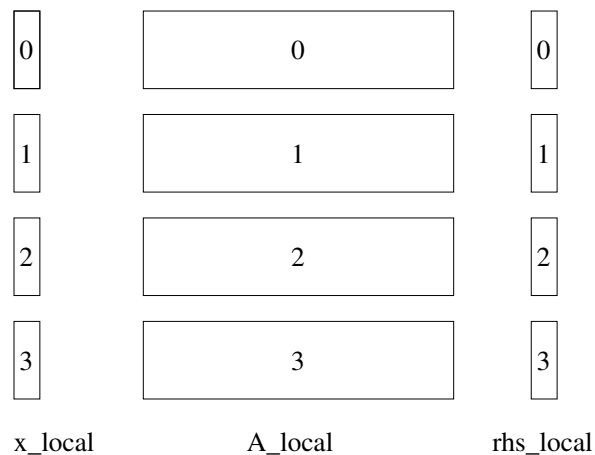


FIGURE 3 – distribution de  $A$ ,  $x$  et  $rhs$

Pour mener à bien cette parallélisation, vous avez

1. un code séquentiel qui sert de référence ; le CG séquentiel est codé dans le fichier `CG_sq.c` et l'exécutable est **Smpirun -np 1 ./CG\_sq <file>** qui doit être lancé dans l'environnement MPI simulé<sup>3</sup>
2. des affichages de valeurs calculées, en commentaires pour le moment, que vous pouvez activer à la fois dans le code séquentiel et le code parallèle afin de vérifier, au fur et à mesure, les calculs.
3. deux matrices (une très petite, `Laplacien.mtx` et une plus importante, `nos3.mtx`) pour valider votre code ; l'utilisation de l'une ou l'autre se fait donnant le nom du fichier contenant la matrice dans la ligne de commande (c'est le `<file>` indiqué plus haut).

---

3. je n'ai pas écrit un Makefile permettant une compilation gcc classique pour le séquentiel et smpicc pour le parallèle