



Bureau d'Étude de Calcul Parallèle

Durée : 4 Heures

Documents autorisés : planches du Cours

Vous êtes autorisés à consulter le site <https://www.rookiehpc.com/> pour voir comment utiliser les routines MPI.

Les 4 premiers exercices sont indépendants.

Le cinquième va réutiliser des développements de certains des exercices précédents.

Bien lire le sujet en entier pour déterminer dans quel ordre vous allez traiter les exercices.

Mise en Place de l'environnement MPI

Nous fournissons dans le répertoire **Fournitures**, les deux fichiers **utils.sh** et **add_ssh.sh** ainsi que le répertoire **hostfiles** avec les listes de machines.

Ce sont ces fichiers qu'il faut utiliser aujourd'hui.

Il faut refaire la procédure de déploiement car vous êtes, a priori, dans une nouvelle salle.

Nous vous rappelons que pour positionner un certain nombre de variables, il faut commencer par exécuter

```
source utils.sh
```

et créer un fichier de machines correspondant à votre salle (vraiment utile que pour l'exercice 1, il n'y a pas de mesure de performance dans les autres exercices)

```
./add_ssh 8 votre_salle
```

```
votre_salle ∈ {214, 301, 302, 304, 306, a-204}
```

Pour chaque exercice, il faudra se placer dans le répertoire correspondant et lancer les process MPI exécutant l'exécutable **exe** sur les coeurs de l'ordinateur local avec :

```
mpirun -np 2 ./exe
```

Et pour utiliser des machines différentes de votre salle (1 coeur par machine) :

```
mpirun -np 2 --hostfile ../hostfiles/votre_salle_8 -N 1 --oversubscribe ./exe
```

Instructions pour compléter les codes fournis

1. Les exercices sont répartis dans des répertoires différents
2. Dans chaque répertoire, un fichier `Makefile` est présent pour compiler l'(les)exécutable(s) de l'exercice
3. Le fichier à compléter est indiqué dans la description de chaque exercice
4. Les endroits du code où insérer des appels à MPI (ou d'autres instructions) sont indiquées ainsi :

\\ ...

5. Les commentaires du code sont là pour vous guider

Instruction pour le Rendu

Vous avez dans le répertoire `Fournitures`, un fichier `Makefile`; en exécutant la commande **make**

Cela va "nettoyer" les différents répertoires de développement et créer une archive de nom `Calcul_username_machine.tar`

**Cette archive est à déposer avant 12h sous Moodle
dans le dépôt correspondant à votre salle**

1 Communication en Anneau (3 points)

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

2 Quand le MPI_Send bascule du mode asynchrone au mode synchrone ? (5 points)

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` 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.
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 sur deux noeuds du même ordinateur ?
 - (d) estimez à 10 entiers près, la taille limite sur deux noeuds de deux ordinateurs différents ?
 - (e) proposez une solution pour que l'échange entre les deux noeuds puissent se faire au delà de cette limite (plusieurs réponses possibles)

3 Produit scalaire distribué (5 points)

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 opérations pouvant être traitées l'une après l'autre.

4 Produit Matrice-Vecteur distribué (3 points)

Répertoire de développement: **04_Mult**

Une matrice A est répartie sur N noeuds suivant un découpage par blocs de lignes. Chaque noeud possède la partie du vecteur v correspondante.

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

Nous voulons que chaque noeud calcule la partie de $x = A * v$ correspondante à sa partie de A et de v .

Cette distribution est illustrée par **Figure 1**.

Pour effectuer ce calcul chaque noeud doit disposer du vecteur v complet (Figure 2). Vous allez donc coder une communication collective pour effectuer ce rassemblement.

- complétez le code du fichier `MultAv.c` en suivant les instructions données en commentaires.

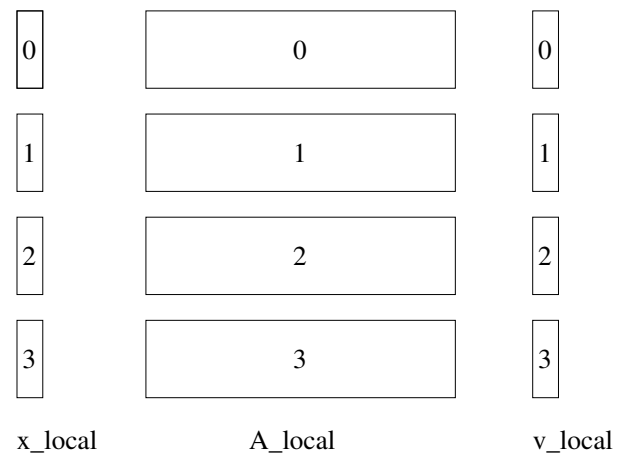


FIGURE 1 – distribution de A , v et x

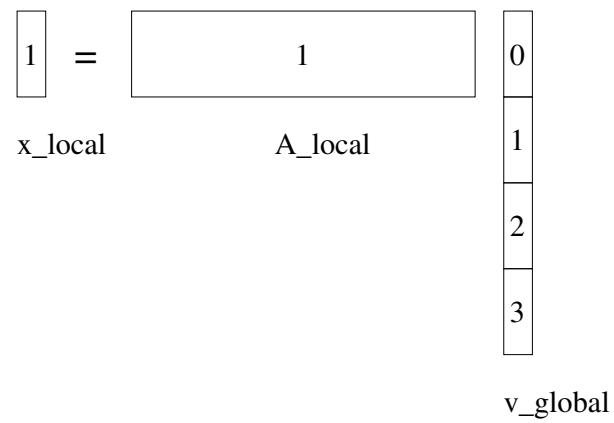


FIGURE 2 – calcul sur le noeud 1

5 Parallélisation de certains calculs de la méthode du Gradient Conjugué (CG, Conjugate Gradient) (4 points)

Répertoire de développement: 05_CG

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

Algorithm 1 a formulation of CG

```

1: Input :  $A$  SPD matrix,  $rhs$ , righthand 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:  $r = b$ 
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 incluses dans le code.

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 `./CG_sq <file>` qui peut être lancé sans MPI
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.

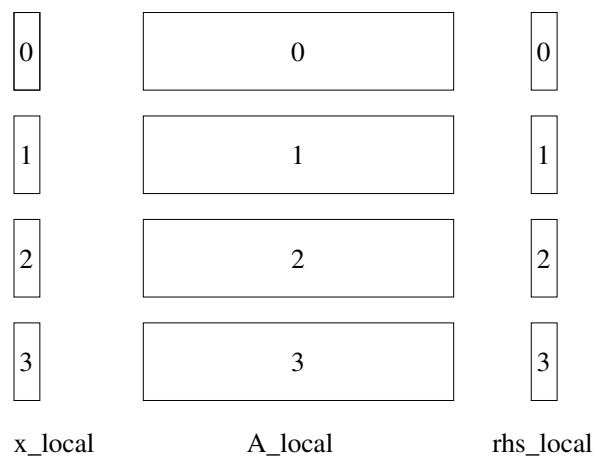


FIGURE 3 – distribution de A , x et rhs