Introduction à la programmation MPI
TD 3 –
Communications non-bloquantes

Exercice I : Prise en main sur les communications non-bloquantes

Question 1 : compléter le programme `ini_nonblock/exercice/tantque.c` (`/* TRAVAIL A FAIRE */`) en utilisant des communications **point-à-point non-bloquantes**.

Question 2 : le programme `ini_nonblock/exercice/deadlock.c` présente un blocage. Résoudre ce blocage en utilisant des communications **point-à-point non-bloquantes**.

Exercice II : MPI_Waitall

Ouvrir le fichier `exo_waitall/exo_waitall.c` .
Lire le travail à effectuer `/* TRAVAIL A EFFECTUER */`.

Principe général du programme à écrire :

- le processus de rang 0 doit remplir et envoyer des tableaux à tous les processus de rangs impairs (les tableaux sont construits à partir de la fonction `fill_val_array`, pour plus de détails lire le fichier `exo_waitall.c`)
- chaque processus impair doit recevoir le tableau que lui envoie le processus 0 et appelle la fonction `check_val_array` pour vérifier que le contenu est bien correct.

Question 1 : Terminer le programme en effectuant uniquement des communications **non-bloquantes**. Pour ce faire, le processus 0 doit utiliser la fonction `MPI_Waitall`.

Question 2 : Quelle est la signification du `MPI_Waitall` pour le processus 0 ? Au choix :

a) barrière sur tous les processus ?
b) ou bien attente de tous les processus impairs ?
c) ou bien attente de la fin de tous les envois vers les processus impairs ?

Question 3 : L'appel à `MPI_Waitall` par le processus 0 est-il obligatoire ?

Question 4 : Pour les processus impairs, est-il possible d'utiliser uniquement des réceptions bloquantes ?

Exercice III : Charges déséquilibrées

Dans une boucle, chaque processus MPI appelle une fonction `work()` dont la charge de travail est aléatoire (entre 1 et 10 s). Le programme doit s'arrêter quand la charge totale de travail dépasse une valeur donnée en argument en ligne de commande.

Dans le fichier `unbal_workload/exercice/exo_unbal_workload.c`, un appel à `MPI_Allreduce` est réalisé à chaque itération pour connaître la charge de travail totale déjà réalisée et la comparer à la charge totale à atteindre.

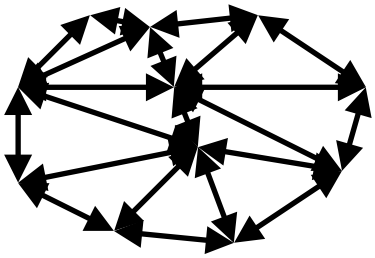
1) Compiler `exo_unbal_workload.c`, exécuter sur 8 processus MPI avec `400` pour charge totale (à passer en argument). Mesurer les temps min et max.

2) Proposer une nouvelle implémentation à base de communication non bloquante pour minimiser le temps total de restitution.

Exercice IV : Graphe de communication

On représente par un graphe (non orienté) les communications entre *P* processus.

Chaque nœud du graphe représente un processus MPI.
Un arc entre deux nœuds définit l'existence d'envois/réceptions entre les deux processus correspondants.



Un graphe de communication est représenté par la structure suivante :

```
struct graphe_t
{
        int nb_noeuds ;

        /* tableau dimensionné à nb_noeuds
             nb_voisins[p] : retourne le nombre de nœuds directement connectés au nœud p
        */
        int *nb_voisins ;

        /* tableau à 2 dimensions
                voisins[p] : tableau dimensionné à nb_voisins[p]
                contient les numéros des nœuds directement connectés au nœud p
        */
        int **voisins ;
} ;
```

L'ensemble des voisins d'un processus p est $\{q$ = voisins[p][iv] où $0 \leq$ iv < nb_voisins[p]$\}$.
Tout processus p ($0 \leq p < P$) doit envoyer nb_voisins[p] messages et recevoir nb_voisins[p] messages.

Pour un processus p donné, les buffers des messages à envoyer se trouvent dans le tableau char **msg_snd ; les tailles des buffers sont dans le tableau int *taille_msg_snd .
Autrement dit, le processus p doit envoyer le message msg_snd[iv] de taille taille_msg_snd[iv] au voisin q = voisins[p][iv] pour tout $0 \leq$ iv < nb_voisins[p].

Pour un processus p donné, les buffers des messages à recevoir se trouvent dans le tableau char **msg_rcv ; les tailles des buffers sont dans le tableau int *taille_msg_rcv .
Autrement dit, le processus p doit recevoir le message msg_rcv[iv] de taille taille_msg_rcv[iv] du voisin q = voisins[p][iv] pour tout $0 \leq$ iv < nb_voisins[p].

Soit la fonction

```
void echange( struct graphe_t *graphe,
              char **msg_snd, int *taille_msg_snd,
              char **msg_rcv, int *taille_msg_rcv ) ;
```

appelée par chaque processus p, et qui effectue les envois/réceptions définis par le graphe de communication graphe.

Question : Écrire la fonction echange en utilisant des communications **non bloquantes**.