ENSEIRB-MATMECA

Parcours CISD

Langages du parallélisme

TP 1 (séances 1+2)

Exercice 0: Que se cache t-il derrière la commande mpicc?

Dans un terminal, lancez la commande `mpicc -show`.

Exercice 1: Hello World!

Ecrivez un programme `hello_world` qui, pour chaque processus/processeur, donne le numéro de processus et le nombre total de processus.

Exercice 2: Rang, taille

Modifiez le programme précédent pour que les processus pairs n'affichent pas le même message que les processus impairs.

Exercice 3: Communications simples

Ecrivez un programme dans lequel le processus 0 envoie un tableau de 10 réels au processus 1. Le processus 0 remplira le tableau, et le processus 1 affichera ses valeurs.

Exercice 4: Ping/pong

Modifiez le programme précédent pour que 1 renvoie des données modifiées à 0.

Exercice 5: *Value forwarding*

Ecrire un programme dans lequel le processus 0 récupère l'heure avec `time_t time()` et l'envoie au processus 1 qui lui-même l'enverra au processus 2, ... jusqu'à atteindre le processus $N-1$. Votre code sera exécuté avec N processus.

Exercice 6: Prise de temps

Modifiez le programme de l'exercice 4 pour générer une courbe de temps de communications selon la taille du message (grossissez la taille).
Indication: utilisez la fonction `double MPI_Wtime()`*. Attention, cette fonction retourne un temps écoulé.*

Exercice 7: Communication en anneau

Ecrire un programme dans lequel un jeton tourne dans un anneau de processeurs. Initialement dans le processeur 0, le jeton passe de processeur en processeur (avec modification) pour retourner au processus 0.

Exercice 8: Diffusion

Ecrire un programme dans lequel le processus 0 envoie un tableau d'entiers à tous les autres processus.

Exercice 9: Somme globale

Ecrire un programme où tous les processus calculent la somme de tous les identifiants des processus et envoient ce résultat sur tous les processus.

Exercice 10: Eager ou rendez-vous?

Déterminer à partir de quelle taille de message on passe du protocole eager au protocole sur rendez-vous.

Exercice 11: Communications non-bloquantes

Ecrire un programme dans lequel deux processus veulent s'envoyer un message en même temps (envoi puis réception). Tester une version avec communications classiques (bloquage). Corrigez cette version avec des communications non-bloquantes.
Indication: si vous n'arrivez pas à avoir un blocage, forcez le en utilisant la fonction `MPI_Ssend`*.*

Exercice 12: Ping/pong (version non bloquante)

Ecrire un programme dans lequel le processus 0 envoie un message au processus 1 et le processus 1 envoie un message au processus 0. Vous utiliserez des communications non bloquantes. Vous pourrez mesurer le temps de communication en ajoutant un timer avant et après les échanges de messages.

Exécutez ensuite le ping/pong plusieurs fois. Voyez-vous une différence?

Exercice 13: Communication en anneau (version non bloquante)

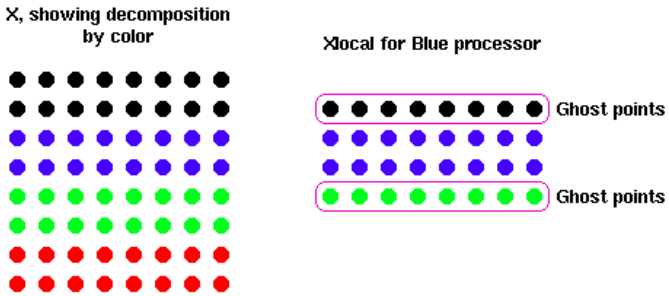
Reprenez le programme dans lequel un jeton tourne dans un anneau de processeurs (exercice 6) et transformez les opérations bloquantes en des non bloquantes.

Entre l'initialisation et la completion, ajoutez du délais avec un sleep en nanosecondes (`int nanosleep(const struct timespec *req, struct timespec *rem)`).

Exercice 14:

Imaginons qu'on ait une structure représentant un maillage à deux dimensions, divisé en blocs alloués sur différents processus. Dans la forme C la plus simple, la structure est définie par `double x[maxn][maxn];`. On veut arranger cette structure pour que chaque processus ait une partie locale: `double xlocal[maxn][maxn/size];`, où `size` est la taille du communicateur (c'est-à-dire le nombre de processus).

Pour effectuer des calculs sur cette structure de données, nous avons besoin des valeurs adjacentes aux différents blocs. En effet, pour calculer l'élément `x[i][j]`, nous avons besoin de `x[i][j+1]`, `x[i][j-1]`, `x[i+1][j]` et `x[i-1][j]`. Les deux dernières valeurs peuvent poser problème si elles se trouvent chez un processus voisin (et non dans xlocal). Pour palier ce problème, on définit des cellules fantômes qui contiennent les valeurs des voisins.



1. Ecrire le programme associé au problème qui divise x en des blocs de tailles équivalentes et copie les cellules fantômes. On supposera que x est de taille maxn par maxn, et que maxn est divisible par le nombre de processus. Pour plus de simplicité, on supposera aussi avoir une taille de tableau et un nombre de processus fixes. Vous utiliserez des communications bloquantes pour cette question.

2. Remplacez les communications bloquantes `MPI_Send` et `MPI_Recv` par `MPI_Isend` et `MPI_Irecv`. Utilisez `MPI_Wait` ou `MPI_Waitall` pour la completion de vos communications.

3. Remplacez `MPI_Send` et `MPI_Recv` par `MPI_Sendrecv` pour échanger des données avec les processus voisins: les processus 0 et 1 échangent un message, les processus 2 et 3 échangent un message,... Ensuite les processus 1 et 2 échangent, 3 et 4,...