Université de Versailles Saint Quentin en Yveline

Master 1 CHPS

Rapport de Parallélisation Hybrid avec MPI et THREAD POSIX

Prenom: Souad

Nom: Mansour

1) Introduction

En informatique, les informations peuvent être traitées, de manière parallèle, ainsi que plusieurs normes spécialisées existent pour programmer celles-ci. Mpi est l'une de ces normes qui ont pour but de réaliser le plus grand nombre d'opérations en un temps réduit. Dans le code qui nous en fournit, il nous demande de compléter des fonctions en se basant sur la programmation hybride qui consiste à travailler avec des threads POSIX à l'intérieur des processus .

2) Exércice: Échanges point à point hybrides MPI/pthread

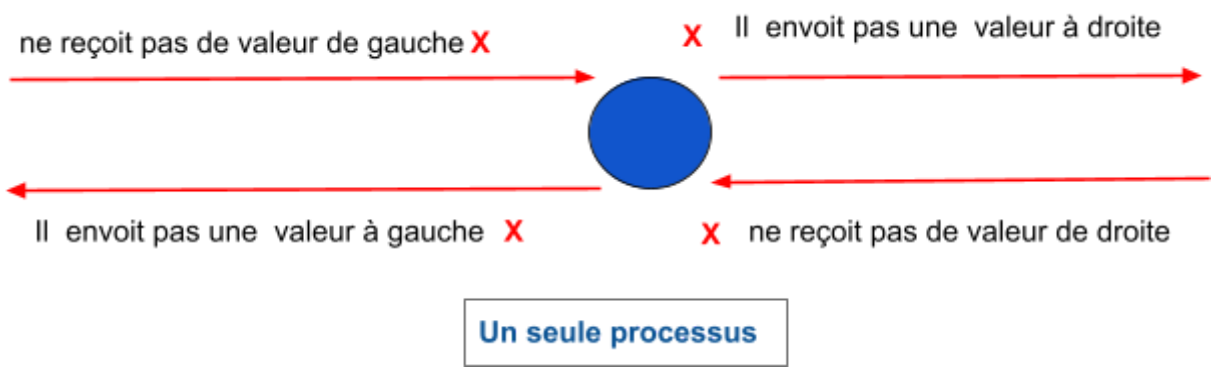
But de l'exercice

Le but de cette partie consiste à faire des échanges point à point entre les threads maîtres qui sont voisins dans tous les processus MPI.

Le premier thread qui rentre met la valeur de visite à 1 . Ce thread représente le thread maître et quand un autre thread veux rentrer il trouve la valeur de visite est à 1 donc il exécute le else dont chaque thread met à jour sa valeur gauche et droite.

A. Le cas d'un seule Processus:

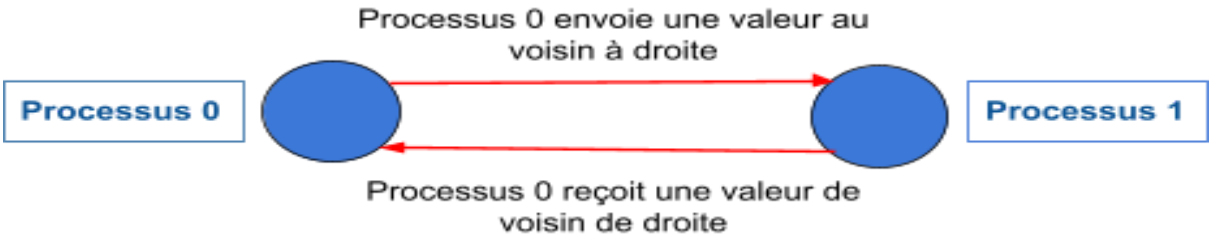
Si le processus voisin n'existe pas ,Les valeurs de gauche et de droite sont affectées à 0 ,car le processus ne reçoit pas une valeur d'un processus et n'envoie à aucun.



B. Le cas ou nous avons plusieurs processus :

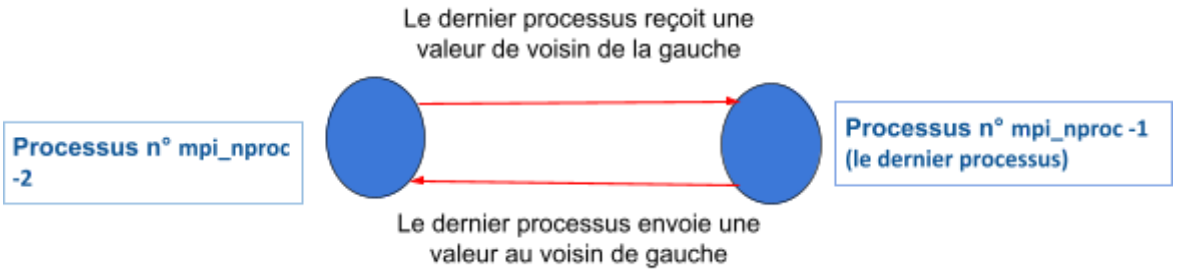
a) Processus 0 :

Le processus 0 va d'abord envoyer une valeur au voisin qui est à droite ,puis il reçoit une valeur de lui, puis il va mettre à jour la valeur qu'il a reçue de son voisin de droite.



b) Le dernier processus mpi_nproc -1

Le dernier processus va recevoir une valeur **sh_ex->gauche** de voisin de rang **mpi_nproc -2** et lui envoie la valeur **sh_arr[0]**, puis il va mettre à jour la valeur qu'il a reçu de son voisin de gauche.

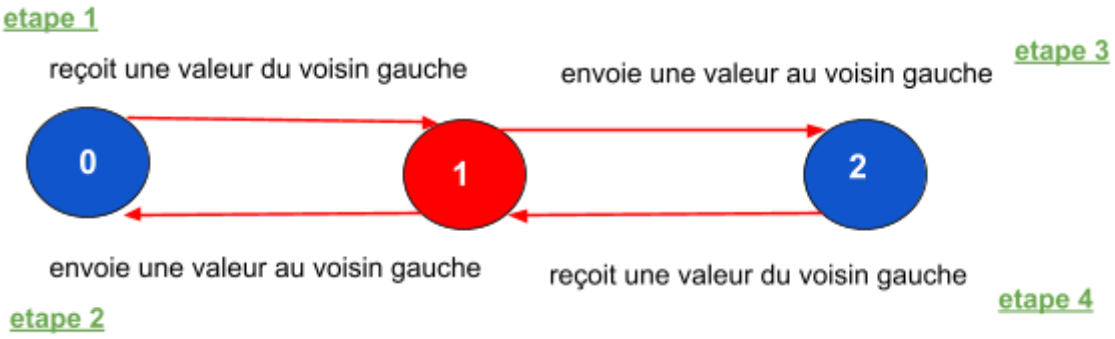


c) Les processus du milieu

➢ D'abord, le processus du milieu va recevoir une valeur de son voisin gauche de rang **rank-1** puis il lui envoie la valeur.

➢ Ensuite il envoie une valeur **mpi_nloc - 1** au voisin de rang **rank+1** de droite puis il reçoit une valeur **sh_ex->droit** de lui.

➢ A la fin, il va mettre à jour les valeurs qu'il a reçues de ses deux voisins de gauche et de droite.



3) Exercice: Réduction somme hybride MPI/pthread

But de l'exercice

Chaque thread master de chaque processus calcule une somme puis il a met dans buf pour que le thread master du processus 0 accède au valeurs qui sont collectées dans le buffer de réception ,puis il fait la somme dans le tableau **out[i]**, ensuite il diffuse avec **bcast** les valeurs aux autres threads master des autres processus pour que tout les threads auront la même valeur.

- **La fonction shared_reduc_init**

Dans cette fonction on a initialisé nos variables, barrière et sémaphore.

- **La fonction shared_reduc_destroy**

Dans cette fonction, on libère les buffers qu'on a alloués.

- **La fonction hyb_reduc_sum**

Cette fonction est appelée par tous les threads master de tous les processus MPI. Cette fonction effectue la somme de toutes les contributions locales de tous les threads inter-processus (tableau **in[]** pour chaque contribution locale) et qui retourne les résultats dans le tableau **out[]** (un tableau **out[]** par thread).

➢ Premièrement, On calcule la somme du tableau **in[]** de tous les threads inter-processus. Pour cela , on a déclaré un mutex qui permet à chaque thread de rajouter sa somme d'une manière séquentielle sans qu'il soit interrompu par un autre thread. Ensuite, on a utilisé une variable principal pour désigner qu'elle est le thread maître dont il est le premier qui rentre dans le mutex et met la variable principal à 1. Enfin, on a utilisé une barrière pour attendre le dernier thread jusqu'à ce qu'il termine son exécution.

➢ Deuxièmement,On a utilisé la primitive **MPI_Gather** :
chaque thread master de chaque processus met sa valeur de tableau **red_val[]** dans le buffer d'envoie et seule le thread master de processus 0 (root) qui collecte toutes les valeurs du tableau **red_val[]** et les concatène dans le buffer de réception **buff** donné.Ensuite, il va sommer les valeurs du tableaux **red_val[]** et les mettre dans un tableau **out[]**.

➢ Après avoir calculé le tableau **out[]** du thread master du processus 0 ,on va mettre à jour la valeur du tableau **red_val[]** qui est partagé par les thread inter-processus.

➢ Le thread master du processus 0 va diffuser la valeur du tableau **red_val[]** et **out[]** aux autres threads master d'autres processus.

➢ Après avoir diffuser les tableaux **out[]** et **red_val[]** aux autres threads masters de chaque processus ,maintenant ils nous reste qu'à mettre à jour les valeurs du tableau **out[]** des autres threads locaux à chaque processus.

4) Compilation et exécution

- Pour compiler le code de **hyb_reduc** il faut:
- d'abord on se positionne au dossier hyb_reduc.
- Ensuite, on compile le programme avec la commande suivante:

  mpicc -pthread test_hyb_reduc.c -I ../mpi_decomp/ -I ../thr_decomp/ -I ../hyb_reduc/ ../mpi_decomp/mpi_decomp.c ../thr_decomp/thr_decomp.c ../hyb_reduc/hyb_reduc.c
- À la fin, on exécute avec mpiexec -n 4 ./a.out.

- la commande suivante pour compiler le code **hyb_exchg:**

  mpicc -pthread test_hyb_exchg.c -I ../mpi_decomp/ -I ../hyb_exchg/ ../mpi_decomp/mpi_decomp.c ../hyb_exchg/hyb_exchg.c -lm.

- À la fin, on exécute avec mpiexec -n 4 ./a.out 4.

5) Conclusion

Dans le cadre de ce projet , on a pu compléter le code de hyb_reduc ou chaque thread de tous les processus doivent avoir les même valeurs à la fin et exchg_reduc qui se base sur la communication point à point ou chaque processus communique avec ses voisins.