TD introduction à MPI

Rappels

- Fonction main doit commencer par MPI.Init(args);
- et finir par MPI.Finalize();
- Les tâches sont numérotées par leur rang
  - rang : MPI.COMM_WORLD.Rank();
  - size : MPI.COMM_WORLD.Size();
- argument −np pour définir le nombre de tâches
- Communication :

$MPI.COMM\_WORLD.Send(\ message, offset, size, datatype, rank, tag)$
$MPI.COMM\_WORLD.Recv(\ message, offset, size, datatype, rank, tag)$

message : Données échangées pendant la communication

offset : Décalage du message

size : Taille du message

datatype : Type de donnée du message

rank : Rang de la tâche destinataire/source

tag : Identification du message (doit correspondre entre l'envoi et la réception)

Mode	Blocking		Non-blocking	
Standard	Send	Recv	Isend	Irecv
Bufferisé	Bsend		Ibsend	
Synchrone	Ssend		Issend	
Ready	Rsend		Irsend	

1 Saute Mouton

On va utiliser MPI pour implémenter un petit jeu de saute-mouton, où un nombre donné de moutons (chacun modélisé par une tâche du programme MPI) se sautent mutuellement par dessus pendant un certain nombre de tours :

1. Dans un premier temps, on écrit un programme MPI à 2 taches. La 1ère sera un de nos moutons, il commence en $(0, 0)$, la 2ème tire, au hasard, des coordonnées (x, y) dans le plan et l'envoie à notre mouton. Quand le mouton, se trouvant au point M du plan, reçoit les coordonnées du point C choisi, il saute par dessus : il se retrouve aux coordonnées obtenues par symétrie centrale de M par rapport à C (ou double translation $\vec{MC}$), et affiche ses nouvelles coordonnées. Le programme s'arrête après 20 sauts.
2. On écrit maintenant le jeu de saute-mouton pour 3 moutons (numérotés 0 à 2), où, à chaque tour, le mouton i envoie ses coordonnées au mouton $i + 1$ (de façon cyclique, *i.e.* le mouton 2 envoie au mouton 0), qui lui saute par dessus et affiche ses nouvelles coordonnées. Les moutons commencent maintenant à des coordonnées (x, y) tirées au hasard dans le plan, et le programme s'arrête après 20 tours. (Attention aux deadlocks)
3. Changez le nombre de moutons (*i.e.* le nombre de tâches du programme), et modifiez votre programme pour qu'il fonctionne pour un nombre arbitraire de moutons.

2 Terminaison

Jusqu'ici, nos moutons sautaient les uns par dessus les autres pendant un nombre fixe de tours, on souhaite maintenant définir une condition d'arrêt plus naturelle à notre jeu de saute-mouton :

1. Dans un premier temps, on fait commencer nos moutons dans un domaine de coordonnées limité ($[-20, 20]^2$) pour observer leur comportement.
2. Vous avez dû observer que nos moutons divergent assez rapidement. Malheureusement, un mouton n'a qu'une portée de saut limitée, on souhaite donc arrêter le jeu quand un mouton n'est plus capable de faire le saut qui lui est demandé. Définissez une portée maximale pour nos moutons et modifiez le programme pour qu'il s'arrête quand un mouton n'est plus capable de faire le saut par dessus le mouton suivant.
3. (OPTIONNEL) Le mouton n'est pas un animal solitaire, donc plutôt que de s'éloigner les uns des autres, ils préfèrent se rapprocher. Pour 3 moutons, trouver des coordonnées (réelles) distinctes telles que les moutons convergent, et modifier le programme pour qu'il s'arrête quand les moutons sont suffisamment prêts les uns des autres. (Note :

étant données l'approximation de nombres réels en flottants, il se peut que les moutons finissent par diverger si on attend trop/suffisamment)

3 Épuisement

On a vu que nos moutons avaient des limitations, on souhaite donc adapter le comportement de notre jeu à la potentielle mort d'un de nos moutons. (Note : "no sheep were harmed in the making of this game")

1. Dans un premier temps, on fait mourir le mouton 0 après un nombre aléatoire k de tours : la tâche de rang 0 s'arrête après k tours. Observer le comportement du programme (on pourra revenir à une condition d'arrêt avec un nombre fixe de tour pour s'assurer que notre mouton meurt pendant le jeu).
2. Modifiez le programme pour que le jeu continue "normalement" (dans la mesure du possible) malgré la mort du mouton 0 : les moutons $n-1$ et 1 doivent donc être capable de détecter la mort du mouton 0 (qui n'a pas le droit de communiquer : "dead sheep tells no tale"), et changer leur comportement en fonction.
3. (Pour ceux qui avancent vite) Étant donnée la portée de saut définie à la partie précédente, on considère maintenant qu'un mouton meurt d'épuisement quand il est incapable de faire son saut. Quand un mouton meurt, son mouton précédant (dans l'ordre de numérotation des moutons) doit maintenant sauter par dessus son mouton suivant. Le jeu s'arrête maintenant quand seulement 2 moutons (ou moins) restent en vie.