Bramas
Timothé

Compte-rendu TP1 OS202

Exercice 1 :

La situation d'interblocage est la suivante :

```
MPI_Comm_rank(comm, &myRank ) ;
if (myRank == 0 )
{
MPI_Ssend( sendbuf1, count, MPI_INT, 2, tag, comm);
MPI_Recv( recvbuf1, count, MPI_INT, 2, tag, comm, &status);
}
else if ( myRank == 1 )
{
MPI_Ssend( sendbuf2, count, MPI_INT, 2, tag, comm);
}
else if ( myRank == 2 )
{
MPI_Recv( recvbuf1, count, MPI_INT, MPI_ANY_SOURCE, tag, comm,
&status );
MPI_Ssend( sendbuf2, count, MPI_INT, 0, tag, comm);
MPI_Recv( recvbuf2, count, MPI_INT, MPI_ANY_SOURCE, tag, comm,
&status );
}
```

1) Scénario sans interblocage :
-Le thread 0 s'exécute et envoie le message au 2 puis attend la réponse
-Le thread 2 reçoit ce message puis envoie le sien au thread 0
-Le thread 0 reçoit le message du thread 2
-Le thread 1 envoie son message au thread 2
-Le thread 2 reçoit le message du 1

2) Scénario avec blocage :
-Le thread 0 envoie son message au thread 2
-Le thread 1 envoie son message au thread 2
-Le thread 2 reçoit le message du thread 1, puis le 0 envoie avant qu'il puisse envoyer le sien, les threads se bloquent.

Probabilité d'avoir un blocage:
Le scénario déclaré sans interblocage semble être le seul (on peut seulement inverser l'ordre entre la réception de 0 et l'envoi de 1), car si le thread 1 envoie en premier, il y aura un blocage soit si 2 envoie avant et que 0 veut envoyer après, soit si 0 envoie et que le thread 2 essaie d'envoyer ensuite.
Si le thread 0 envoie en premier et que 2 reçoit mais que le thread 1 envoie après, le thread 2 sera bloqué pour envoyer son message au thread 0.

On suppose que chaque action thread a une chance sur 3 d'effectuer chaque action, en négligeant le temps pris par les conditions if.
La première action a une chance sur deux d'être réalisée par le thread 0, car le thread 2 attend dans tous les cas donc cela ne change rien s'il agit en premier.
Il faut ensuite que les threads effectuent des actions dans cette ordre :
-2 (recevoir) une chance sur deux car si 0 effectue cette action cela ne change rien car il attend
-2 (envoyer) une chance sur deux car si 0 effectue cette action cela ne change rien car il attend
-0 (recevoir), deux chances sur 3 car permutable avec la prochaine
-1 (envoyer), une chance sur 2 car il ne reste que 2 actions
- 2 (revecoir), probabilité de 1 car c'est la dernière action qu'il reste

Cela donne une probabilité globale de $\frac{1}{2}\times\frac{1}{2}\times\frac{1}{2}\times\frac{2}{3}\times\frac{1}{2}$ soit une chance sur 24.

Exercice 2 :

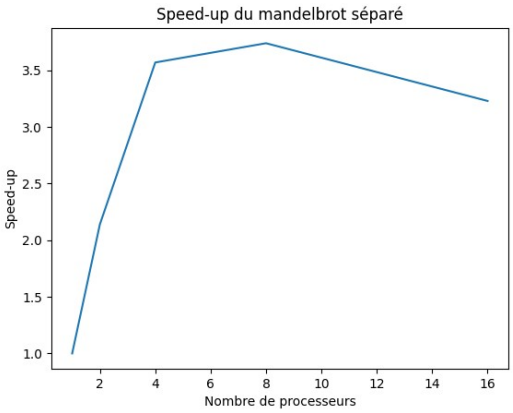
En utilisant la loi d'Amdhal, comme Alice ne peut pas paralléliser f=10% de son code, quand n tend vers l'infini, elle aura un speedup maximal égal à S=10.
En inversant la loi d'Amdhal pour n raisonnable, on a $n=\frac{S(n)\times(1-f)}{1-(f\times S(n))}$

En voulant avoir S(n)=9, on a par exemple n=81, cela semble élevé.
Pour S(n)=8, on trouve n=36. Sur un ordinateur puissant, ou en utilisant plusieurs ordinateurs il est possible d'obtenir 32 coeurs de calcul, on peut donc fixer n=32, qui donne S(n)=7.8, c'est plus des trois quarts du speed-up maximal, cela semble raisonnable.

Exercice 3 :

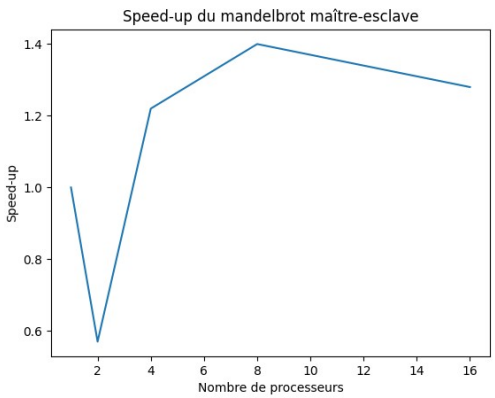
Tous les programmes que j'ai utilisés pour avoir mes résultats sont sur mon github : timothe-bramas.
Les mesures sont effectuées sur mon ordinateur personnel, en effet sur l'ordinateur salle de l'ENSTA il y a un soucis à l'exécution des scripts python. Mon ordinateur possède 8 processeurs.
1) Résultats du code en parallèle :

Nombre de threads	1(séquentiel)	2	4	8	16
Temps de calcul (s)	4.0	1.87	1.12	1.07	1.24
Speed-up	1	2.14	3.57	3.74	3.23



2) Résultats du code maître-esclaves :

Nombre de threads	1(séquentiel)	2	4	8	16
Temps de calcul (s)	4.0	7.02	3.26	2.85	3.12
Speed-up	1	0.57	1.22	1.4	1.28

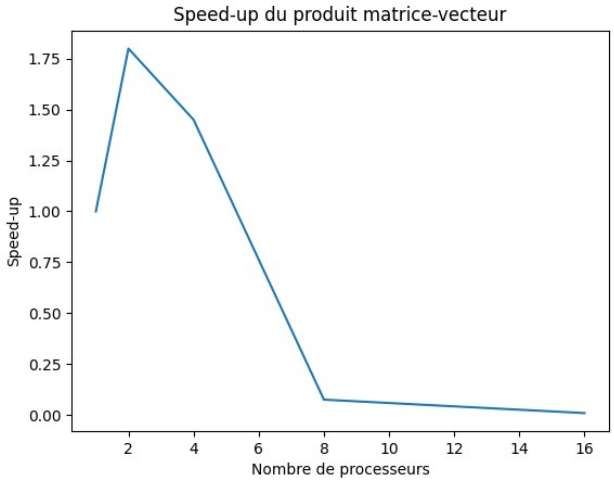


On voir que cet algorithme semble bien moins efficace que le premier car le thread 0 perd beaucoup de temps à séparer les tâches, action qui est faite de base en exécutant tous les threads en même temps.

Exercice 4 :

En séparant la matrice ligne par ligne, on obtient :

Nombre de threads	1(séquentiel)	2	4	8	16
Temps de calcul (ms)	0.9	0.5	De 0.2 à 2	De 0.1 à 12	De 2 à 100
Speed-up	1	1.8	1.45	0.075	0.009



On voit un speed-up efficace pour n=2, mais après la séparation et réallocation des sous-matrices semble être trop peu efficace comparé au calcul relativement simple à effectuer.

Ici, la séparation par colonnes semble très inadaptée car le produit matrice-vecteur comme cela est réalisé ligne par ligne, il faudrait donc utiliser des valeurs de chaque thread pour chacune des lignes.