Rapport TD2

Programmation parallèle

JODAR SOARES Gustavo
LOPEZ NOREÑA Vanessa

github link:
https://github.com/Gustavo-Jodar/Programmation_Parallele

02/2023

2. Exercices à rendre

2.1. Question du cours

```
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 );
}
```

Le scénario avec interblocage c'est lorsque le processus 1 s'exécute avant les autres processus parce que s'il envoie le message au processus 2, le processus 2 va recevoir ce message et après il va envoyer un message au processus 0 mais le processus 0 n'est pas en attendant un message, par contre, il voudrait envoyer autre message au processus 2 et comme aucun processus est disposé à recevoir, le deux processus s'attendent mutuellement en causant un interblocage.

Processus 1 → MPI_Ssend(processus2)
Processus 2 → MPI_Recv(processus 1)

Processus 2 → MPI_Ssend (processus 0)
Processus 0 → MPI_Ssend (processus 2)
INTERBLOCAGE

Le scénario sans interblocage c'est si le processus 0 s'exécute avant le processus 1 parce qu'il va envoyer un message au processus 2 et le processus 2 peut recevoir le message, envoie un message au processus 0 et le processus 0 va être en attendant son message. Après, le processus 2 est prêt pour recevoir le message du processus 1.

Processus 0 → MPI_Ssend(processus2)
Processus 2 → MPI_Recv(processus 0)

Processus 2 → MPI_Ssend (processus 0)
Processus 0 → MPI_Recv(processus2)

Processus 1 → MPI_Ssend(processus 2)
Processus 2 → MPI_ Recv(processus 1)

La probabilité est équiprobable (50%) parce que les deux processus ont la même probabilité d'arriver premièrement et générer un interblocage pour le cas du processus 1 ou développer les commandes normalement pour le cas du processus 0.

2.2. Question du cours n°2

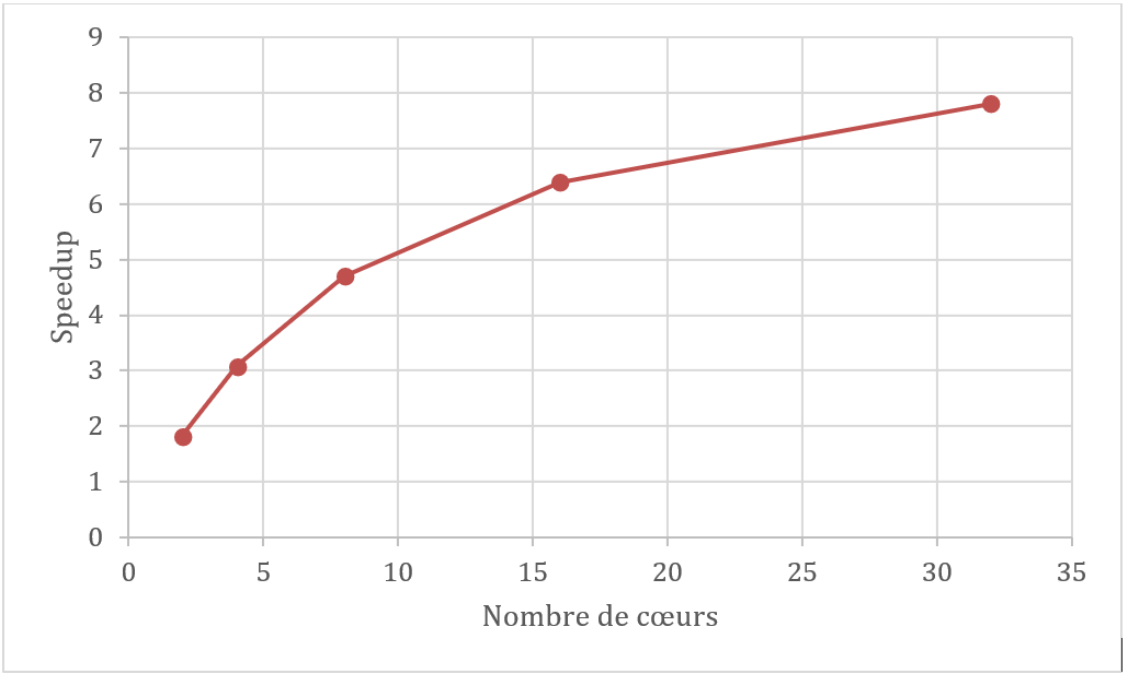
Le temps de traitement en parallèle c'est 90% du temps d'exécution du programme en séquentielle. Alors, on peut utiliser le loi d'Amdahl pour mesurer l'accélération en dépendent des coeurs que pourra obtenir Alice avec son code

$$S(n) = \frac{n}{1 + (n-1) * f}$$

Où n est le nombre de processeurs et f est la fraction du code non parallélisable.
Dans le cas d'Alice, le f est 10% parce qu'il est le temps d'exécution en séquentielle et on va considérer que n se rapproche de l'infini pour prédire l'accélération maximale.

$$S(n) = \frac{1}{\frac{1}{n} + f - \frac{f}{n}} = \frac{1}{f} = \frac{1}{0.1} = 10$$

Si n tend vers l'infini, on trouve que l'accélération maximale est 10% mais c'est une accélération avec plusieurs cœurs.
Dans la figure ci-dessous, on peut faire un comparatif des cœurs et l'augmentation de l'accélération du code. Pour ne pas gaspiller de ressources CPU, on va considérer que les 4 premiers options parce que 32 coeurs ce n'est pas possible actuellement, cependant, 16 coeurs c'est gaspille un peu les ressources parce qu'on augmente deux fois le nombre de coeurs et l'accélération augmente seulement 1.7 en plus qu' avec 8 coeurs , alors il semble raisonnable prendre 8 coeurs en profitant presque le moitié d'accélération maximale.



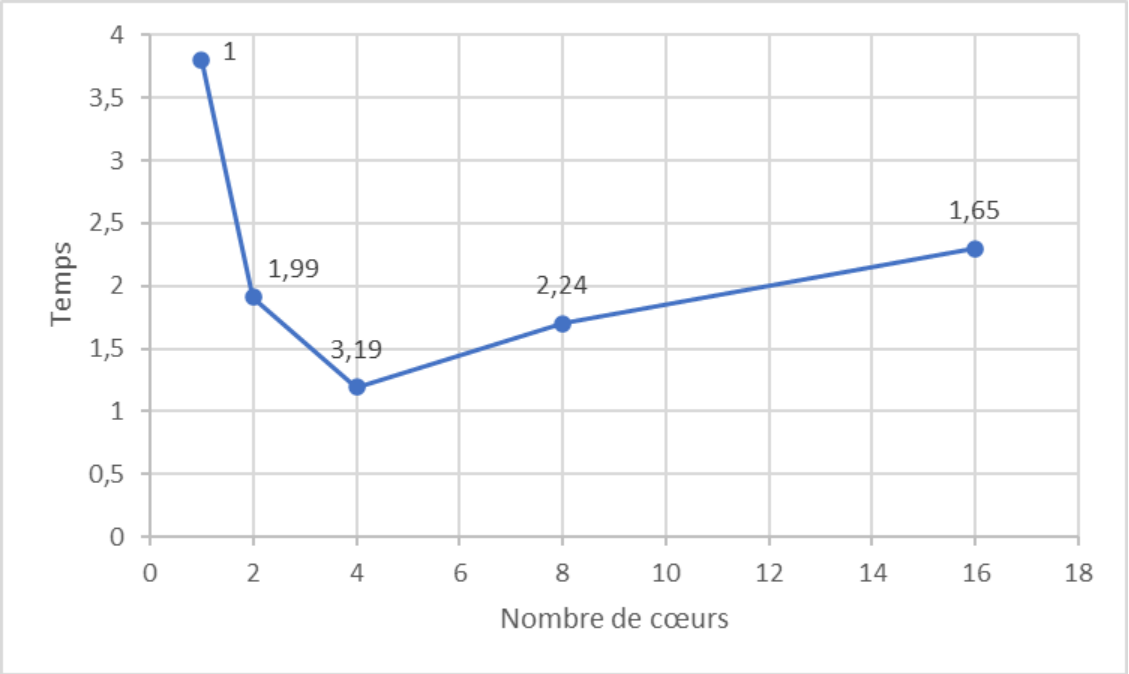
Alice s'aperçoit qu'elle obtient une accélération maximale de quatre en augmentant le nombre de nœuds de calcul pour son jeu spécifique de données et en remplaçant cette accélération obtenue par Alice dans le loi d'Amdahl, on peut trouver qu'elle a utilisé 6 coeurs, cependant cette loi est appliquée pour une taille de problème fixée.
On peut calculer l'accélération en prenant en compte les données avec la loi de Gustafson

$$S(n) = n + (1 - n) * s$$
$$S(n) = 6 + (1 - 6) * 0.1$$
$$S(n) = 5.5$$

où s c'est la fraction séquentielle du code de départ.

2.3. Ensemble de mandelbrot

Le temps d'exécution, speedup et résultats obtenus. Les nombres à côté des balles sont les speed ups qui compare le temps pour exécuter le programme de façon séquentielle sur la façon parallèle.



Les résultats obtenus sont à cause du nombre de cœurs dans le processeur, comme l'ordinateur qui a fait les tests a 4 cores, le logiciel a eu un résultat meilleur avec 4 processus en parallèle.

2.4. Produit matrice-vecteur

2.4.1. Produit parallèle matrice - vecteur par colonne

Code -> prod_mat_vet_col.py

Disponible dans le github de l'équipe: https://github.com/Gustavo-Jodar/Programmation_Parallele/blob/main/TD_numero_2/sources/python/prod_mat_vec_col.py

2.4.2. Produit parallèle matrice - vecteur par ligne

Code-> prod_mat_vet_line.py

Disponible dans le github de l'équipe: https://github.com/Gustavo-Jodar/Programmation_Parallele/blob/main/TD_numero_2/sources/python/prod_mat_vec_line.py