Rapport du projet HPC

Qinyan Yang

Partie1 : prendre en main la librairie decomp2d (avec code part1.f90 et scaling.py)

Dans cette partie, j'ai choisit ***Φ(x,y,z) = sin(x)*sin(y)*sin(x)***, pour implémenter ***Φ - λ²Φ***. D'après l'exécution des codes, j'ai obtenu les données et es courbes de scaling suivants :

Nombre de processes	Temps de calcul	Temps de communication	Temps total
2	6.480969E-03	1.303448E-02	1.951545E-02
4	4.164213E-03	7.120550E-03	1.128476E-02
8	2.727995E-03	4.384740E-03	7.112735E-03
16	1.397788E-03	2.793956E-03	4.191744E-03
32	8.113913E-04	1.818863E-03	2.630255E-03
64	6.913404E-04	2.023665E-03	2.715005E-03
128	5.526696E-03	7.846752E-03	1.337345E-02
256	1.255466E-03	9.569121E-02	9.694668E-02

Tab_1

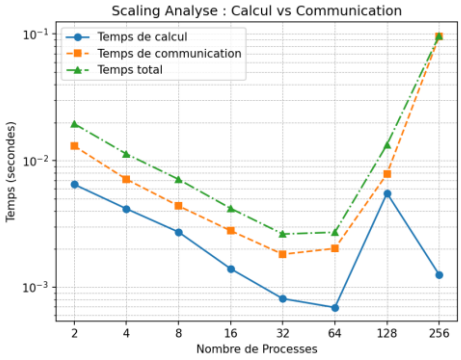

Fig_1

D'après le graphique (Fig_1) et les données fournis (Tab_1), voici l'analyse de l'efficacité des différents nombres de processus et la comparaison des temps de calcul et de communication :

Temps de calcul

Le temps de calcul diminue globalement avec l'augmentation du nombre de processus, indiquant une exécution plus efficace des tâches de calcul en parallèle.

Temps de communication

Le temps de communication est élevé lorsque le nombre de processus est faible, diminue avec l'augmentation du nombre de processus jusqu'à un certain point, puis augmente significativement lorsque le nombre de processus (d'après 32 processus) augmente encore. Cela suggère que trop de processus peuvent entraîner une augmentation des coûts de communication.

Temps total

Le temps total est calculé par la somme du temps de calcul et du temps de communication, avec une

caractéristique similaire que la courbe du temps de communication. Cela indique qu'il existe un nombre (autour de 32) optimal de processus pour lequel le temps total est minimal, équilibrant les coûts de calcul et de communication.

Conclusion

Selon le graphique, le temps total est minimal à 32 processus, ce qui indique que c'est le point où les coûts de calcul et de communication sont équilibrés de manière optimale.

Partie2 : La méthode du gradient conjugué (avec code part2.f90 et scaling.py)

```
Running with 2 processes:
 Taille de MPI_COMM_WORLD =           2
 In auto-tuning mode......
 factors:           1           2
 p_row x p_col           1           2
 Méthode du gradient conjugué terminée, nombre d'itérations :        1001
 Temps de calcul :   169.152249292587      secondes
 Temps de communication :   169.152249292587      secondes

Running with 4 processes:
 Taille de MPI_COMM_WORLD =           4
 In auto-tuning mode......
 factors:           1           2           4
 p_row x p_col           2           2
 Méthode du gradient conjugué terminée, nombre d'itérations :        1001
 Temps de calcul :   85.2271906737005      secondes
 Temps de communication :   85.2271906737005      secondes

Running with 8 processes:
 Taille de MPI_COMM_WORLD =           8
 In auto-tuning mode......
 factors:           1           2           4           8
 p_row x p_col           2           4
 Méthode du gradient conjugué terminée, nombre d'itérations :        1001
 Temps de calcul :   43.2693037780700      secondes
 Temps de communication :   43.2693037780700      secondes

Running with 16 processes:
 Taille de MPI_COMM_WORLD =          16
 In auto-tuning mode......
 factors:           1           2           4           8          16
 p_row x p_col           4           4
 Méthode du gradient conjugué terminée, nombre d'itérations :        1001
 Temps de calcul :   22.2915858876659      secondes
 Temps de communication :   22.2915858876659      secondes
```

```
Running with 32 processes:
 Taille de MPI_COMM_WORLD =          32
 In auto-tuning mode......
 factors:           1           2           4           8          16
          32
 p_row x p_col           4           8
 Méthode du gradient conjugué terminée, nombre d'itérations :        1001
 Temps de calcul :   11.4479497986613      secondes
 Temps de communication :   11.4479497986613      secondes

Running with 64 processes:
 Taille de MPI_COMM_WORLD =          64
 In auto-tuning mode......
 factors:           1           2           4           8          16
          32          64
 p_row x p_col           8           8
 Méthode du gradient conjugué terminée, nombre d'itérations :        1001
 Temps de calcul :   6.19421366194729      secondes
 Temps de communication :   6.19421366194729      secondes

Running with 128 processes:
 Taille de MPI_COMM_WORLD =         128
 In auto-tuning mode......
 factors:           1           2           4           8          16
          32          64         128
 p_row x p_col           8          16
 Méthode du gradient conjugué terminée, nombre d'itérations :        1001
 Temps de calcul :   3.47820109885652      secondes
 Temps de communication :   3.47820109885652      secondes

Running with 256 processes:
 Taille de MPI_COMM_WORLD =         256
 In auto-tuning mode......
 factors:           1           2           4           8          16
          32          64         128         256
 p_row x p_col          16          16
 Méthode du gradient conjugué terminée, nombre d'itérations :        1001
 Temps de calcul :   1.99176577385515      secondes
 Temps de communication :   1.99176577385515      secondes
```

Fig_2

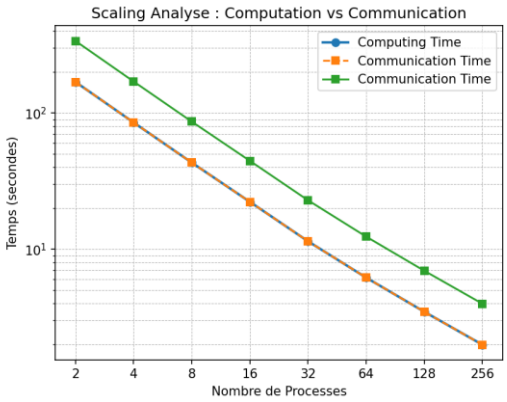

Fig_3

D'après les données fournis (Fig_2) et le graphique (Fig_3), voici l'analyse de l'efficacité des différents nombres de processus :

Les trois courbes diminuent de manière approximativement linéaire, conforme aux attentes de forte scalabilité.

Conclusion

Selon le graphique, le code montre une bonne scalabilité forte et une accélération quasi linéaire dans la plage de 2 à 256 processus.