Rapport du Projet Hybrid d'APP

Bouton Nicolas

April 2021

Contents

Le but du projet était de paralléliser un programme séquentielle de **gradient conjugué** avec une implémentation **hybrid MPI / Pthread**.

1 Réduction d'une somme hybride MPI / Pthread

1.1 Schéma expliquant la démarche prise

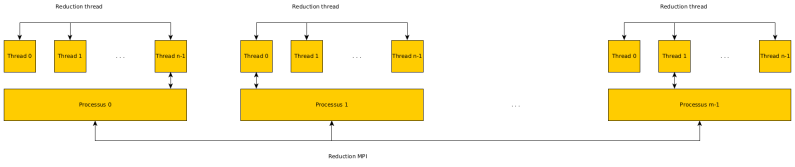

Figure 1: Schéma expliquant la démarche prise

1.2 Explication du code

Mon raisonnement est le suivant:

1. Chaque thread de chaque processus MPI va sommer sa valeur avec celle qu'il partage avec ces autres threads en commun (du même processus MPI)

2. Une fois que la somme est faites localement sur chaque processus MPI, on élit un thread pour qu'il fasse la réduction avec les autres threads élus des autres processus MPI. Par exemple pour le **processus 0**, c'est le thread **n-1** qui est élu alors que pour le **processus 1** c'est le **0**. En réalité c'est le premier thread qui finit qui est élu.

3. Une fois que les threads élu ont fait leur réduction ensembles. Il faut que les autres threads non élu actualisent leurs réductions.

Pour ce faire j'ai utilisé un **sémaphore** pour pouvoir élire le premier thread qui arrive. Et à la fin pour l'actualisation du résultat on pourait lâcher tout les threads en faisant que le threads élu fasse une boucle de $\mathbf{sem_{post}}$ mais je ne sais pas pourquoi ca ne marchais pas. Donc j'ai laisser l'actualisation en séquentielle.

A la fin de la fonction, j'ai mis une **barrière** pour pouvoir synchroniser tout les threads de tout les processus MPI, en mettant d'abord une barrier au niveau des threads du même processus pui une barrier entre touts les processus (qui est prise $\mathbf{NUM_{THREADS}}$ fois car je ne sais pas trop comment faire pour qu'un seul thread le fasse sachant que nous n'avons pas l'id du thread).

Et j'ai aussi fait attention de remettre les variables de la structure comme elles étaient initialisés pour le prochain tour de boucle. (je fais donc un $\mathbf{sem_{post}}$ pour le dernier threads afin que le premier thread qui arrrive dans le $\mathbf{sem_{wait}}$ passe).

2 Echange point à point hybrides MPI / Pthread

Pour ce qui est des échanges point à point, cela est relativement la même chose que pour la réduction. Il y 2 grandes parties:

1. Echanges entre les threads élu des processus MPI (premier arrivé)

2. Actualisation entre les threads du même processus MPI avec les valeurs que le thread élu à échangé.

La aussi j'ai utilisé un **sémaphore** pour élire le premier thread. Et comme pour la réduction hybride j'ai fait attention à réinitialisé les variables pour le dernier thread.

3 Algorithme du gradient conjugué parallèle MPI / Pthread

3.1 Initialisation

La seule chose à prendre en considération c'est que c'est le premier thread du premier processus MPI qui contient la valeur du **i global** égale à **0**.

Et que c'est le dernier thread du dernier processus MPI qui contient la valeur **i global** égale à **i global maximum** du tableau.

3.2 Réduction

Il y a 2 fonction qui font une réduction:

- le calcul de la norme au carré
- le rapport de 2 profuits scalaire

En réalisté ce sont toutes les fonctions qui font une réduction en séquentielle et qui retourne un flottant.

Donc pour ces 2 fonctions j'ai utilisé la fonction de réduction hybride.

3.3 Produit matrice vecteur

A faire.

4 Conclusion

Comme vous allez le voir à l'exécution, mon implémentation donne un résultat faux mais je ne comprends pas pourquoi. Si jamais vous avez un retour sur mon implémentation ou mon raisonnement je suis prenneur.