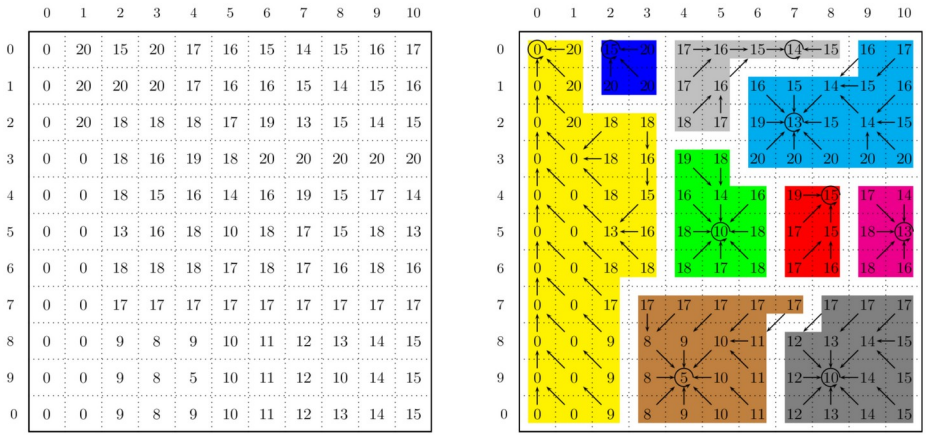
Projet programmation parallèle

L3 Informatique, année 2021-2022



Ce projet utilise les Modèles Numériques de Terrain (ou MNT, voir https://fr.wikipedia.org/wiki/Modèle_numérique_de_terrain), qui sont simplement des cartes d'altitude d'une région. Elles sont stockées dans des fichiers d'extension `.mnt` de format très simple, en texte clair : quelques caractéristiques (taille, position sur Terre, etc.) suivies d'un tableau de nombres flottants donnant l'altitude de chaque point. Quelques exemples, de différentes tailles, se trouvent dans le répertoire `input/` fourni.

Le programme fourni consiste à calculer le remplissage des cuvettes de la carte par l'algorithme de Darboux. Cet algorithme relativement simple va calculer dans quelle direction s'écoule l'eau qui tombe au sol sur ce terrain, et donc quelles cuvettes vont se remplir (voir la figure ci-dessus). Puis il enregistre une carte modifiée (cuvettes remplies) dans le fichier passé en troisième argument du programme, ou sur la sortie standard s'il n'y a que deux arguments. Il affiche également sur la sortie standard une carte des cuvettes remplies dans ce cas, en texte. La boucle while principale (fichier darboux.c, dernière fonction) calcule itérativement une nouvelle carte W à partir d'une carte existante Wprec, et s'arrête dès que W et Wprec sont identiques. Cette boucle est intrinsèquement séquentielle : W dépend de Wprec à chaque itération. Le calcul de W utilise en chaque point [i,j] la valeur de m[i,j] et les valeurs des 8 voisins de [i,j] dans Wprec :
W[i,j] = f(m[i,j], Wprec[i +/- 1, j +/- 1])

Ce projet consiste à paralléliser l'algorithme de Darboux en MPI et OpenMP. Vous découperez les matrices m, Wprec et W en bandes de tailles égales (ou similaires) : la hauteur de la matrice (nrows) n'est pas forcément divisible par le nombre de processus.

Voici quelques indications :

1. Gérez les entrées/sorties sur le processus 0 et distribuez la matrice m dans la fonction mnt_read. À la fin de l'exécution de cette fonction, tous les processus doivent avoir leur (bande de) mnt `m` allouée et initialisée correctement, ainsi que les valeurs `m→ncols`, `m→nrows`, et `m→no_data`
2. Initialisez la matrice Wprec correctement sur chaque processus au début de la fonction darboux (attention à l'initialisation de la valeur max globale !)
3. Au début de chaque itération de la boucle while principale, vous enverrez les première et/ou dernière ligne de Wprec au processus précédent et suivant (et les recevrez de manière

symétrique), car ces processus auront besoin de ces valeurs pour calculer les lignes de leurs bords
4. Puis vient le calcul, chaque processus met à jour sa bande de matrice W
5. Enfin, pour vérifier si le calcul est terminé sur tous les processus, vous ferez une réduction sur la variable modif
6. Récupérez le résultat sur le processus 0 qui gère la sortie.

Une fois que ce programme parallèle fonctionne (il fonctionne s'il calcule exactement les mêmes résultats que le programme séquentiel - vérifiez que c'est le cas !), sauvegardez-le sous un nouveau nom et effectuez des mesures de performance.

Puis réfléchissez à son optimisation : les échanges des premières/dernières lignes peuvent être réalisés simultanément à une partie du calcul. De même, la réduction peut être réalisée simultanément au calcul de l'itération suivante, et l'arrêt de la boucle while pourra s'effectuer plus tard.

Vous pourrez également utiliser OpenMP pour exécuter des boucles parallèles sur chaque processeur multi-coeurs et un seul processus MPI par machine. Attention, mpirun contrôle la manière dont les coeurs sont assignés aux processus (par défaut). Pour lancer un programme mpi avec un processus par machine et le maximum de threads OpenMP disponible (nombre de coeurs), utilisez la commande :

```
mpirun -n ... -hostfile ... --map-by node --bind-to none ./a.out
```

Explication :
L'option '--map-by node' indique à mpirun de n'assigner qu'un processus à chaque noeud (node) qui est présent dans le hostfile. L'option '--bind-to none' laisse chaque processus libre d'utiliser tous les coeurs disponibles sur son noeud.

Mesurez les gains de performances que vous obtenez grâce à ces optimisations.

Comme référence, voici les meilleurs temps que j'ai obtenus sur les machines vmCalculParallele* pour la fonction darboux, pour les tailles de problème medium et large :

MEDIUM :
- 30s pour 1 processus * 1 thread (1 coeur, version séquentielle)
- 8.3s (3.6x) pour OpenMP sur 4 coeurs
- 4.3s (7x) pour 6 processus * 4 threads par processus (24 coeurs répartis sur 6 machines)
  (Note : cette dernière mesure est très variable, la marge d'erreur est énorme)

LARGE :
- 311s pour 1 processus * 1 thread (1 coeur, version séquentielle)
- 93s (3.34x) pour OpenMP sur 4 coeurs
- 27s (11.5x) pour 6 processus * 4 threads par processus (24 coeurs répartis sur 6 machines)

Ce projet est à réaliser en binôme. Vous déposerez une archive (de préférence .tar.gz) contenant votre code et un petit rapport (format pdf) des mesures que vous avez effectuées. N'oubliez pas d'indiquer les noms des deux participants au projet dans le rapport, et ne déposez qu'une seule archive par binôme. N'incluez pas les fichiers de données fournis (le répertoire input/), ils sont trop gros pour être déposés sur moodle !