UNIVERSITÉ DE STRASBOURG

Projet de programmation parallèle

Parallélisation de calcul de Darboux

Licence 3 informatique

BELARIBI NADJIB
DELIGIANNIS ILIAS
19/04/2020

Introduction:

Dans le cadre de notre sixième semestre de licence d'informatique il nous a été donné le choix entre différentes options. L'une d'elle, programmation parallèle, nous propose au travers de ce projet de mettre en pratique les connaissances acquises durant les cours pour optimiser un algorithme donné.

Le sujet choisi pour ce projet est en rapport avec la carte topographique. Plus exactement le calcul des cuvettes d'une carte par l'algorithme de darboux qui prend en entrée un modèle numérique de terrain(MNT). À travers l'utilisation de la bibliothèque mpi nous proposons, donc, ici une façon de distribuer les calculs pour avoir un gain de performance conséquent.

Les annexes se trouvent dans le repertoire Annexes

Performance de l'algorithme de darboux en séquentielle:

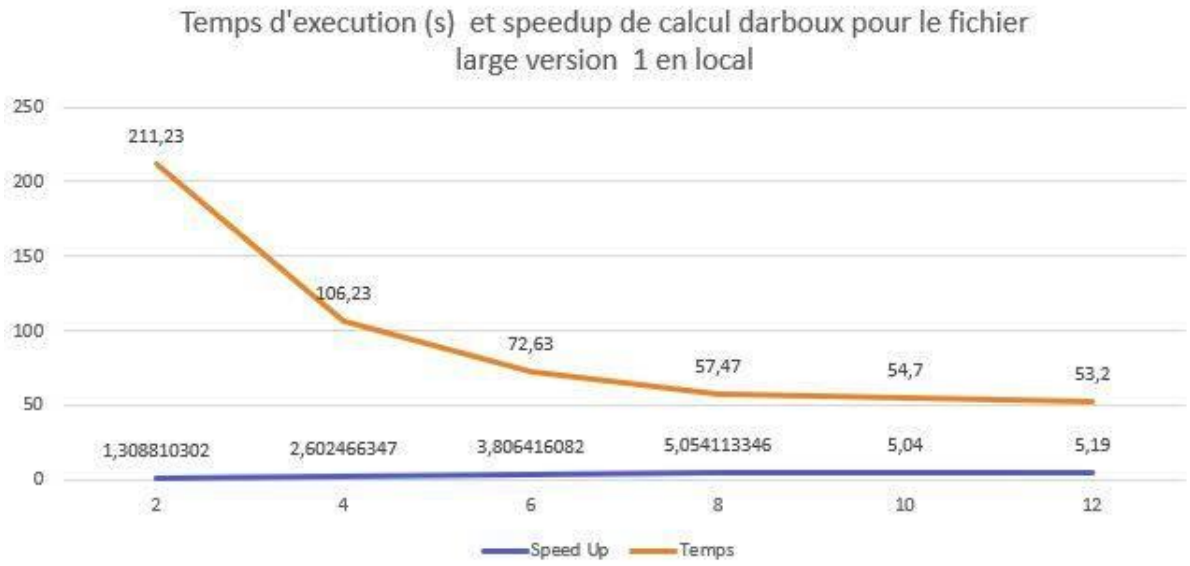
Temps d'éxecution de la version séquentielle sur Turing

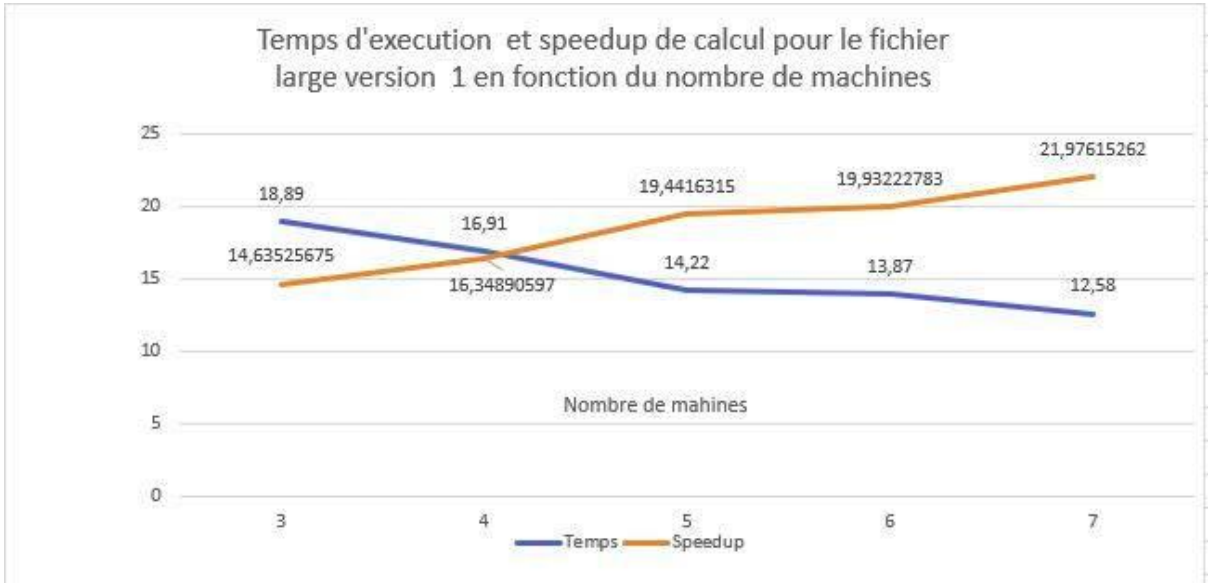
Fichier	MINI	SMALL	MEDIUM	LARGE
TEMPS (s)	0,0005	0,15256	31,62136	276,46

Parallélisation basique (Schéma dans annexes) :

- Le processus 0 s'occupe de lire et distribuez la matrice m. A la fin de l'exécution tous les processus ont leur bande de mnt m alloué et initialisé correctement, ainsi que les valeurs m->ncols, m->nrows, et m->no_data et m->max. Ceci est fait par un simple broadcast d'une structure MPI créée.
- ***Cas particuliers***: si le nombre de lignes n'est pas divisible par le nombre de processus on alloue une marge pour appeler Scatter correctement tel que:
  marge = nblignes + nbproc - (nblignes % nbproc)
- Nous avons initialisé la matrice Wprec correctement sur chaque processus au début de la fonction darboux en ajoutant 2 lignes (au début et à la fin) pour recevoir des données d'autres processus. Pour gérer le décalage dans l'initialisation de Wprec et m, nous avons remarqué qu'on peut utiliser Terrain(m,i-1,j) au lieu de rajouter une ligne à la matrice m de chaque processus et ceci dans chaque appel de la fonction Terrain.
- Pour le calcul de la fonction darboux, on fait des send / recv qui dépendent du rang de processus dans un ordre précis pour éviter l'interblocage.
- Une fois que ce calcul est finit, une réduction sur la variable end est faite. Nous avons ensuite fait un broadcast pour sortir de la boucle while dans tous les processus.
- Nous avons récupéré les résultats par un Gather au niveau du processus 0 qui gère l'affichage des résultats .

Test performance sur le fichier large:





Parallélisation optimisée :

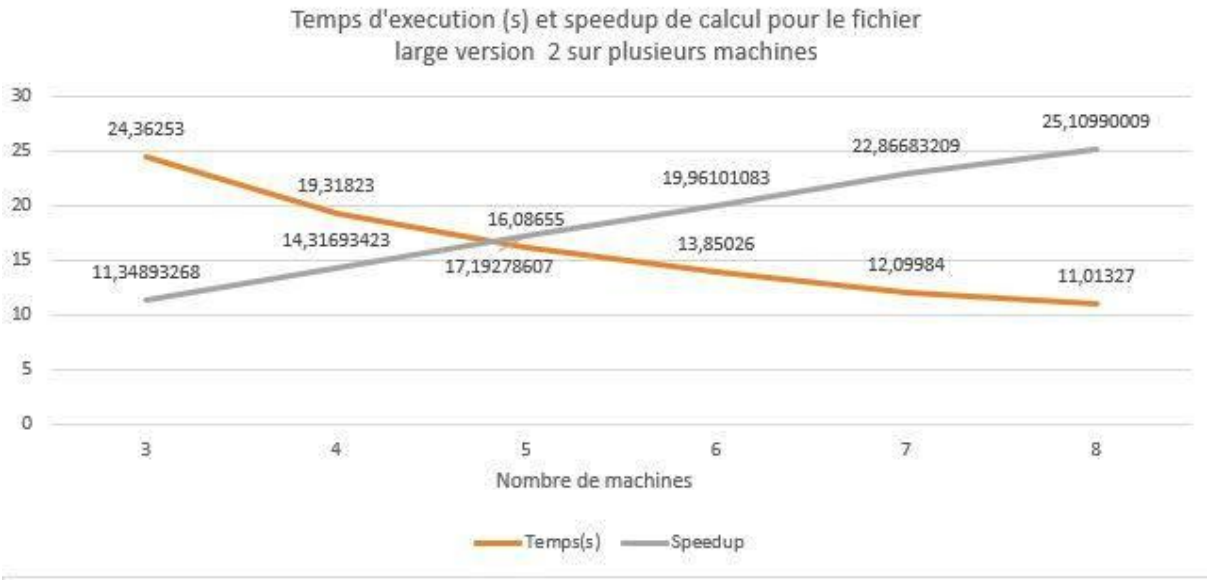
Sur la parallélisation basique nous nous somme rendu compte que dans la fonction darboux avant de commencer d'effectuer les calcul nous faisions attendre nos processus pour qu'ils reçoivent les données nécessaires des autres processus. Après quelques réflexions nous nous sommes rendu compte que ça nous faisait perdre du temps car le temps que les messages soient reçus on aurait pu effectuer les calculs ne nécessitant pas les données envoyées.

- Ainsi après l'envoi des premières/dernières lignes de chaque processus, au processus précédents, nous commençons directement le calcul mais que jusqu'à l'avant dernière ligne. Quand nous y sommes il suffit de recevoir la ligne du processus suivant pour enfin finir le calcul. AInsi le temps que les paquets transitent à travers le réseau le calcul peut continuer.

- De plus nous nous sommes rendu compte qu'en faisant la réduction avant les calculs ça permettait d'accélérer encore un peu le calcul.

- Enfin nous avons incorporé encore plus de parallélisme en utilisant du openmp. Certaines des boucles implémentées dans le fichier **darboux.c**, init et max, sont parfaitement parallélisables. Cela nous a permis d'accélérer encore un peu le calcul.

Tests performance sur le fichier lagre:



Analyse des résultats: (se référer à l'annexe pour voir tous les résultats)

Au premier coup d"oeil nous réalisons directement que pour des petits fichiers à traiter la solution séquentielle reste une solution très convenable a utiliser (cf. test MINI et SMALL). En effet les solutions parallèles deviennent moins efficaces à chaque processus ou machine ajoutée au calcul. Cela reste cohérent car le temps gagné en calcul ne va pas être suffisamment important comparé au temps système et au temps mit par les messages à transiter par le réseau. On s'intéresse donc ici surtout au résultat pour les fichier de taille medium et large.

Nous avons décrit 2 méthodes de parallélisation. Cela pour une bonne raison. Chacune d'elle est plus efficace en fonction de la taille des données traitées. Pour la version

parallèle de base on remarque que pour le fichier Medium le calcul de darboux est finit en 1,93s et pour large en 12,58s (sur 8 machines soit 48 processeurs) soit respectivement un speed up maximal de 16,4 et 21,9. La version optimisée dans les mêmes conditions de tests finit ses calculs en 3,8s et 11,01s soit un speedup maximal de 8,4 et 25,1. Cela provient certainement du fait que la solution optimisée effectue une boucle supplémentaire et que nous finissions tout de même par atteindre certaines réceptions. Cependants cela nous permet de dire qu'en fonction de la taille des données a traiter différentes solutions seraient optimal. AInsi on pourrait imaginer avoir encore une augmentation de performance visible pour la version optimisée si nous utilisions un fichier encore plus conséquent que le fichier large.

Conclusion:

Finalement nous nous sommes contentés d'une implémentation du parallélisme exclusivement par l'utilisation de mpi. C'est en effet avec cette implémentation que les résultats étaient le plus probant, même jusqu'à finir le calcul de large en *7s* en fonction de l'état de turing, avec un speedup maximal de 25 sur le fichier large.mnt. Cependant comme nous l'a montré l'analyse de performance nous avons deux implémentation qui sont plus ou moins efficace en fonction de la taille des données traitées. Pour des fichiers de taille supérieure à médium la meilleure option semble être la version la plus optimisée mais sur des fichiers moins lourds la première version est largement suffisante. Cela permet de pouvoir traiter les données voulu de façon optimal en fonction de leurs taille.