ENSG Géomatique

ÉCOLE NATIONALE
DES SCIENCES
GÉOGRAPHIQUES

Rapport du TP 2 Programmation Parallèle

Ce TP porte sur le filtrage par convolution d'une image en utilisant la bibliothèque MPI en langage C.

Réalisé par : Amina Aiache
Elève ingénieur en 3ème année

Encadré par : M. Ahmad AUDI

Table des matières

I. **Introduction** 2

II. **Calcul séquentiel** 2

Question 1

Question 2

Question 3

III. **Calcul Parallèle** 3

Question 4

Question 5

Question 6

Question 7

Introduction

Ce TP consiste à appliquer un filtre par convolution à une image en effectuant un calcul distribué avec l'utilisation de la bibliothèque MPI en langage C.

Calcul séquentiel

Question 1

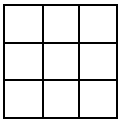
Dans la fonction convolution, si on ne prépare pas un tampon intermédiaire, on perd les valeurs initiales des pixels. En effet, le calcul d'une convolution dépend du résultat de la convolution précédente.

D'où le besoin d'avoir deux tampons, un pour l'image initiale et l'autre pour l'image convolutée. Quand on calcul une image convolutée, on la stocke dans l'image initiale, et on stocke l'image initiale dans un tampon pour pouvoir calculer la convolution suivante, et ainsi de suite.

Question 2

- On ne peut pas paralléliser les convolutions consécutives, car chaque convolution dépend du résultat de la convolution précédente.
- On peut paralléliser la fonction convolution, car le calcul de chaque pixel ne dépend pas des autres.

Question 3



- La complexité théorique de calcul d'u pixel :

Le calcul d'un pixel revient à faire 9 multiplications et 8 additions.

Donc, la complexité est de O (9 + 8) ≈O (19) ≈O (1) par pixel

Par conséquent, la complexité est constante pour tous les pixels de l'image, ce qui signifie qu'il vaut mieux prévoir un équilibrage de charge statique entre les processeurs.

Calcul Parallèle

Question 4

Le découpage qui semble être le plus naturel dans ce contexte est le découpage en bandes. Il existe d'autres types de découpages (triangulaire, rectangulaire,…) mais celui-ci est le plus optimale, car il nous permet de faire la calcul avec le moins de communications possible.

Question 5

- On ne calcul pas la convolution sur les bords de l'image globale.
- Chaque processus aura un problème pour calculer la première et la dernière ligne de son bloc. En fait pour les calculer, il doit récupérer les lignes manquantes des blocs voisins.

Question 6

On découpe l'image en p blocs (p étant le nombre de processus que l'on souhaite lancer).

On a opté pour une solution maître-esclave, le maître parcourt donc l'image une fois et récupère ses caractéristiques.

- On définit : Hlocal = H / p
- À partir de Hlocal on va déterminer la hauteur de chaque bloc (Hloc) en fonction de son rang (rank).

Hloc = Hlocal + 2 si rank > 0 ET rank < p - 1

Hloc = Hlocal + 1 si rank = 0 OU rank = p - 1

Rank = p - 1	Hloc = Hlocal + 1
Rank = p – 2	Hloc = Hlocal + 2
Rank = p – 3	Hloc = Hlocal + 2
. . .	Hloc = Hlocal + 2
Rank = 0	Hloc = Hlocal + 1

Donc, lorsqu'un processus reçoit un bloc, s'il existe un processus p2 = p – 1 il lui envoie sa ligne 0 et reçoit la ligne 0 du processus suivant, et reçoit sa dernière ligne et envoie sa dernière ligne au processus suivant.

- Pseudo-code : Calcul parallèle de la convolution avec des envoies de messages bloquants
- Paramètres :
    - Nb_iter : le nombre d'itérations
    - Hloc : longueur de l'image locale
    - H longueur de l'image globale et W sa largeur > 0

- Rank : rang du processus
- MAITRE : rang du processus maître
- P : nombre de processus

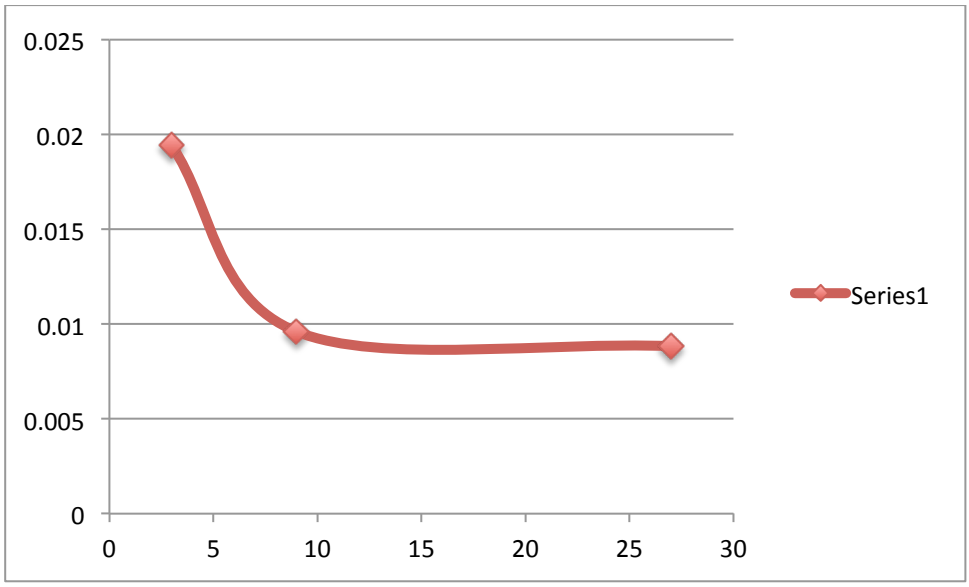
– Image filtrée

1- si (rank == MAITRE) faire :
   1-1 lire le fichier image (fonction lire_rasterfile)
   1-2 Récupérer la longueur H et la largeur W de l'image.
2- Tester si le nombre de processus divise H
3- Envoyer H et W aux processus (fonction MPI_Bcast)
4- Calcul de Hloc en prenant en considération la position de chaque bloc
5- Allocation dynamique de chaque bloc local de taille (W * hloc * sizeof (unsigned char))
6- Tester l'allocation dynamique
7- Envoyer des blocs de l'image aux processus (fonction MPI_Scatter)
8- Pour i allant de 0 à nb_iter faire :
   8-1 Envoyer sa ligne 1 au processus précédent (rank – 1)
   8-2 Recevoir la ligne 0 du processus précédent
9- Récupérer les blocs (fonction MPI_Gatter)
10- Fin du chronométrage et affichage du temps de calcul

Le tableau suivant montre la comparaison du temps d'exécution du programme séquentiel et parallèle :

		Séquentiel	parallèle dynamique
nombre de processus	----	0,025259	
	3		0,0194349
	9		0,00960588
	27		0,00883508

Ce qui donne lieu au graphe suivant :



L'exécution avec 27 processus a été effectuée avec 3 machines de 8 cœurs chacune.

On remarque que le temps d'exécution diminue d'une façon remarquable lorsque l'on passe de 3 à 9 processus, par contre le fait de passer à 27 ne change pas grand chose, ceci peut être du au fait que le fait d'augmenter le nombre de processus en traine l'augmentation des communications entre les processus, ce qui peut alourdir le programme. Il est aussi possible que ce soit du au réseau de l'école (pour faire communiquer 2 machine) et donc on gagne rien.

Question 7

Pour le code parallèle avec envoi de messages bloquant, je l'ai bien implémenté, par contre je n'ai malheureusement pas réussi à le faire marcher.

Je n'ai pas fait d'analyse dessus.