ENSG Géomatique

ÉCOLE NATIONALE
DES SCIENCES
GÉOGRAPHIQUES

Rapport du TP 1 Programmation Parallèle

Ce TP porte sur le calcul de l'ensemble de Mandelbrot 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

Introduction

Ce TP consiste à calculer l'ensemble de Mandelbrot en effectuant un calcul distribué avec l'utilisation de la bibliothèque MPI en langage C.

L'ensemble de Mandelbrot est une fractale définie comme l'ensemble des points c du plan complexe pour lesquels la suite de nombres complexes définie par récurrence par :

$Z_0=0$

$Z_{n+1}=Z_n^2+c$

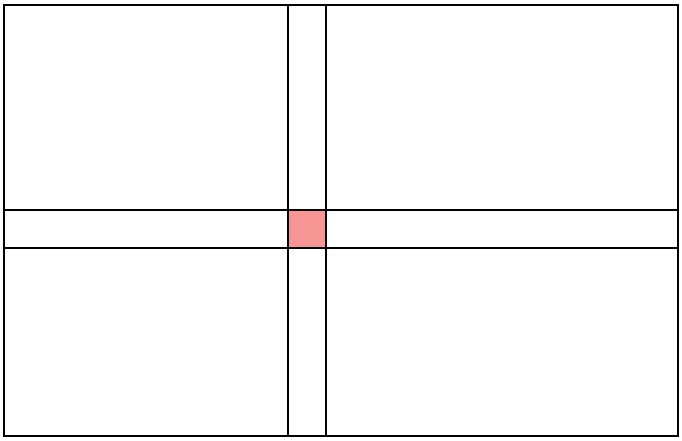
est bornée.

Calcul séquentiel

Question 1

Le calcul séquentiel consiste à réaliser le calcul de l'image pixel par pixel de façon séquentielle :

- Imbriquer 2 boucles sur la hauteur et la largeur de l'image et puis à calculer les pixels de l'ensemble un par un en incrémentant le compteur des deux boucles imbriquées.



H est la longueur de l'image et W sa largeur.

Le compteur sur les lignes : j : 0 → W – 1

Le compteur sur les colonnes : i 0 → H – 1

Chaque pixel est codé sur 8 bits (1 Octet) : type char.

Pour stocker l'image en mémoire : W * H *sizeof (unsigned char).

Le pas d'incrémentation en X : Xinc = (Xmax – Xmin) / (W – 1)

Le pas d'incrémentation en Y : Yinc = (Ymax – Ymin) / (H – 1)

Question 2

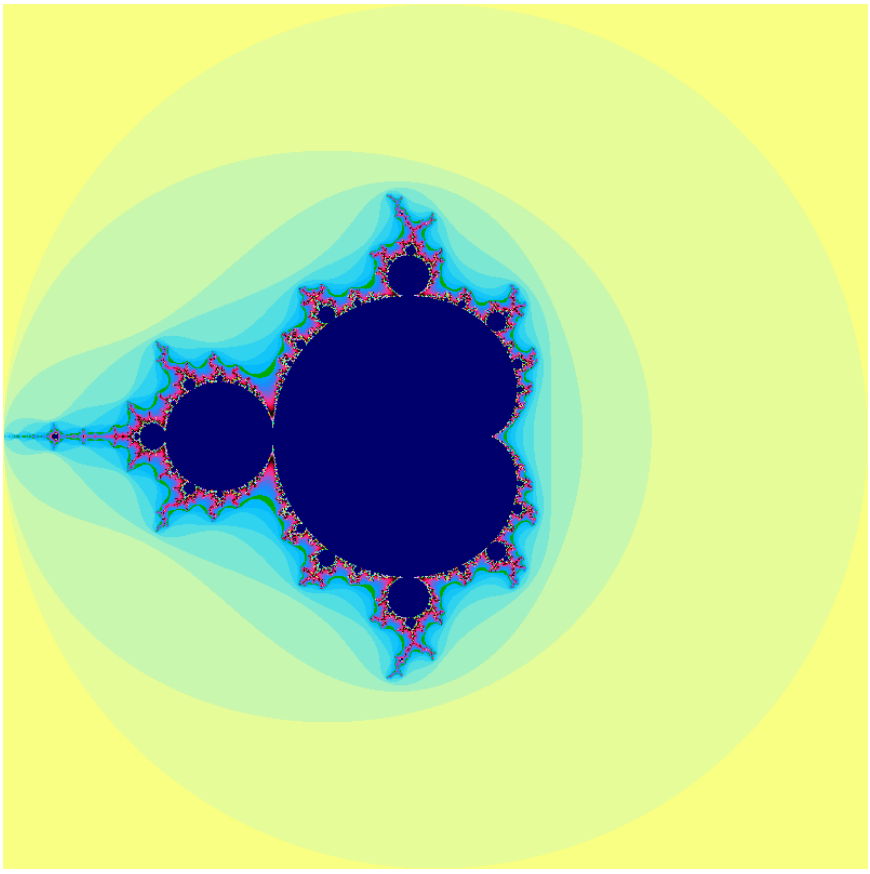
2.1

Compilation :

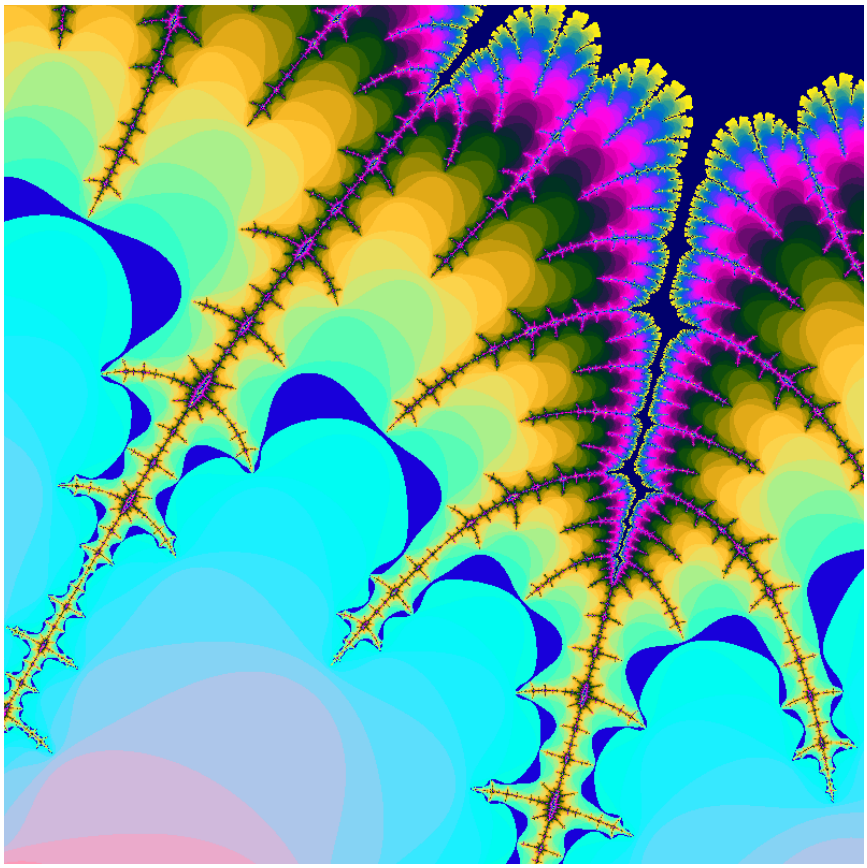
gcc src/mandel.c -o build/mandel –lm

Exécution :

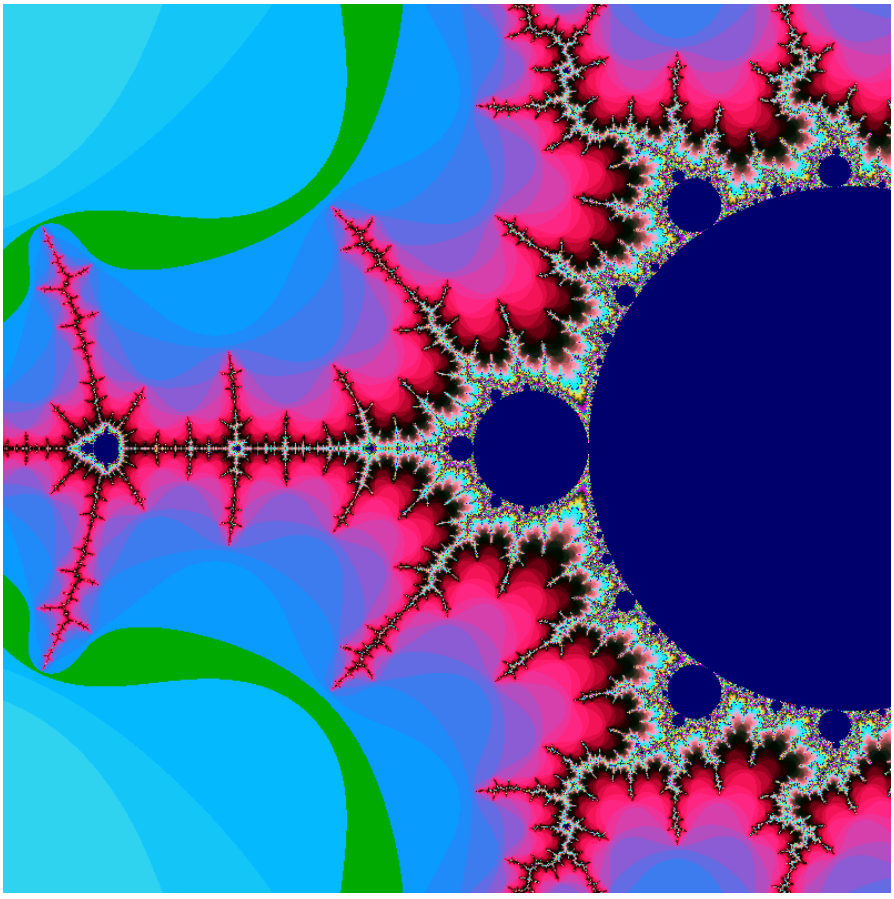
./mandel



./mandel 800 800 0.35 0.355 0.353 0.358 200



./mandel 800 800 -1.5 -0.1 -1.3 0.1 10000



Question 3

3.2

Pour paralléliser le code séquentiel, on va séparer l'image en blocs, un bloc par processus.

Rank = p - 1
Rank = p – 2
Rank = p – 3
.
.
.
Rank = 0

Pour ce faire, on doit diviser la longueur de l'image H par le nombre de processus p.

La longueur de l'image locale traitée par chaque nœud est donc : Hloc = H / p.

Si la longueur de l'image H n'est pas divisible par le nombre de processus p on arrête l'exécution du programme.

En peut donc en déduire Ymin_loc de chaque bloc en fonction de son rang :

$$Ymin\_loc = Ymin + rank * Yinc * Hloc$$

Deux stratégies, au moins, sont donc possible :

- Équilibrage de charges **statique** : consiste à désigner un processus MAITRE qui contient l'image complète, et envoie aux ESCLAVES les blocs qu'ils ont à calculer, y compris à lui même.
  Donc, le MAITRE fait le calcul de son bloc et reçoit les résultats envoyés par les ESCLAVES.
- Équilibrage de charges **dynamique** : vu qu'il y a des endroits de l'image dont le calcul est plus long, on envisage une autre stratégie qui optimise le temps de calcul.
  On divise l'image de blocs qui font nb_lignes de lignes chacun.
  Le MAITRE ne fait pas de calcul, il joue le rôle du chef et envoie aux ESCLAVES des blocs à calculer, et, à chaque fois qu'un esclave fini, il lui envoie un nouveau bloc, du moment où il reste encore des blocs à calculer.
  Nb_lignes : le nombre de ligne de chaque bloc élémentaire (granularité).
  Hypothèse : le reste de la division de la longueur de l'image par le nb_lignes doit être égal à 0.
  Le rang du MAITRE n'est pas nécessairement 0, on peut rajouter un paramètre supplémentaire dans la ligne de commande pour le spécifier.

3.3

- Pseudo-code 1 : Calcul de l'ensemble de Mandelbrot en charge statique (MAITRE Partie 1)

Paramètres : rank = rang du processeur ; Hloc = longueur de l'image locale > 0 ; H : longueur de l'image > 0 ; Xmin ; Xinc ; Ymin ; Yinc ; W = largeur de l'image.

Sortie : Calcul traité par le maitre, avec positionnement du pointeur au début de son image locale et allocation dynamique de l'image globale.

```
/* Tester si le processus courant est le maire ; NB : c'est pas nécessairement 0 */

if (rank == MAITRE) {

        // Allocation dynamique de l'image globale de taille W * H * sizeof (unsigned char)

        //Calcul de la position du pointeur au début de l'image globale ;

        pima = ima + W * rank * Hloc ;

        //traitement de la grille point par point de la même façon qu'on séquentiel mais ;

        Y = Ymin-loc et i passe de 0 à H-loc

}
```

- Pseudo-code 2 : Calcul de l'ensemble de Mandelbrot en charge statique (MAITRE Partie 2)

Sortie : Réception des calculs effectués par les esclaves, sauvegarde de l'image dans un fichier, et affichage du temps de calcul.

```
/* Si le processus est le maître */

if (rank == MAITRE) {

        /*Réception des images locaes des esclaves*/

        for (int k =0 ; k < p ; k++) {
```

```
                MPI_Probe(MPI_ANY_SOURCE, tag, MPI_COMM_WORLD, &status) ;

                Int s = status.MPI_SOURCE ;

                if (s != MAITRE) {

                        MPI_Receive (ima + s * W * Hloc * sizeof(unsigned char), H/P*W,
                MPI_CHAR, s, MPI_COMM_WORLD, &status) ;

                }

        }

        /*Sauvegarder l'image + fin du chronométrage */

} else if (rank != MAITRE) {

        //Allocation dynamique de l'image locale de taille W * Hloc * sizeof(unsigned char)

        //Faire le calcul point par point

        //Une fois fini, envoyer le résultat au MAITRE

        RMI_Send(ima, w * Hloc, MPI_CHAR, MAITRE, tag, MPI_COMM_WORLD) ;

}
```

- Pseudo-code 3 : Calcul de l'ensemble de Mandelbrot en charge dynamique

```
// Allocation dynamique de l'image globale de taille W * H * sizeof (unsigned char)

//test de l'allocation dynamique

for (int i = 0 ; i < p -1 ; i++) {

        if (rank != MAITRE) {

                //On envoie au processus esclave de rang i un numéro de bloc en mettant le tag
                du message à TAG_REQ

                //On incrément le numéro de bloc

        }

}

for (int i = 0 ; i < H/nb_lignes ; i++) {

        //On reçoit comme message le numéro du bloc fait par un esclave dont on connaît pas le
        rang (pour le moment) et le tag du message = TAG_REQ

        //On détermine le rang de l'émetteur par status.MPI_SOURCE

        //On reçoit comme message de l'esclave déterté les données du bloc de taille nb_lignes *
        W puis on met ces données à l'adresse convenable ima + W * nb_lignes * num_blocs

        //test de fin de calcul de l'image

        //s'il reste du calcul à faire, on envoie un numéro de bloc au processus qui a envoyé son
        calcul.

        //sinon, on envoie à ce processus un message qui indique qu'on a plus de travail à faire.

}

// Fin du chronométrage + affichage du temps
```

// Sauvegarde de l'image finale

/*Esclave*/

1- Allocation dynamique de la portion de calcul du processus esclave W * nb_lignes * sizeof(unsigned char)
2- Test de l'allocation dynamique
3- /*Tant que le calcul global de l'image n'est pas fini*/
   while (fin_esclave != 0)
   3-1 l'esclave reçoit le numéro du bloc du travail envoyé par le maitre.
   3-2 /*On détecte le tag du message*/
   Si (tag == TAG_END) on sort de la boucle tant que où on met fin_esclave à 1 et cela signifie qu'il ne reste plus de calcul à faire.
   3-3 Sinon, l'esclave fait son calcul sur le nb_lignes proposé par le maitre en utilisant le Y = Y + num_blocs * nb_lignes * Yinc et j passe de 0 à nb_lignes – 1

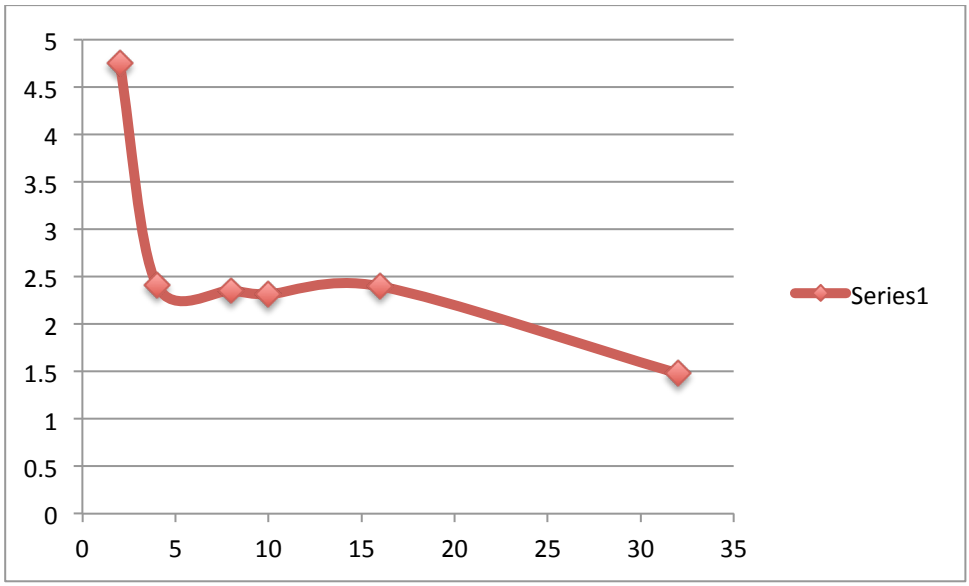
Tests de performance

Question 4

Pour analyser la performance du code parallèle, j'ai comparé le résultat d'exécution du code séquentiel aux résultats d'exécution du code parallèle avec une distribution de charge statique, en augmentant le nombre de processus d'exécution et puis en lançant sur 3 machine de 8 cœurs chacune. Les résultats sont dans le tableau suivant :

		Séquentiel	parallèle statique
nombre de processus	----	3.72474	
	2		2,88828
	4		2,9465
	8		2,79826
	10		2,48942
	16		2,09887
	32		1,58871

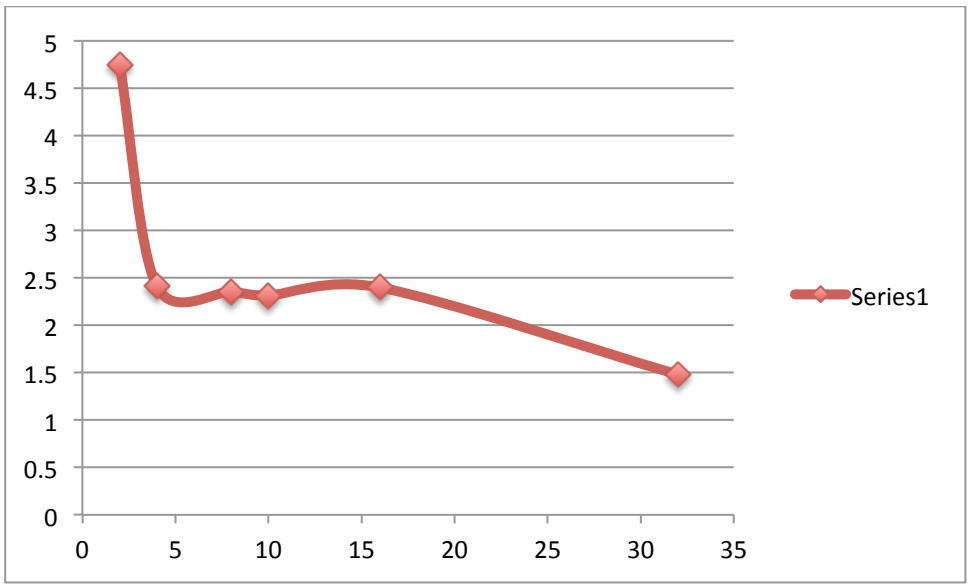
Le graphe suivant montre l'évolution du temps de calcul en fonction du nombre de processus :



La même chose avec l'équilibrage de charge dynamique :

		Séquentiel	parallèle dynamique
nombre de processus	----	3.72474	
	2		4,74818
	4		2,41144
	8		2,35341
	10		2,31485
	16		2,39832
	32		1,47991

Le graphe :



On remarque bien que la parallélisation du code permet de réduire le temps de calcul. Par contre, ce qui me semble pas logique c'est que le temps d'exécution en charge statique est plus cour

qu'en charge dynamique, ce qui peut être du au fait que le maitre ne travaille pas, ce qui fait un processus de moins.

Améliorations

Question 5

- Pour la distribution de charges dynamique, le maître ne fait pas de calcul, il joue le rôle du chef et du coup perd du temps à attendre les esclaves. Donc ce qui serait bien c'est de lui attribuer un travail un faire, mais pas trop car il doit vérifier à chaque fois s'il a reçu le travail des autres.
  L'idée est donc de lui donner une partie de l'image à calculer, il travail et vérifie la réception au même temps.
- Le fait de choisir de découper l'image en paquets de données petits entraîne un nombre important de communications entre le maître et les esclaves. Or, en programmation parallèle, on cherche toujours à diminuer ce nombre car cela consomme beaucoup en mémoire et ce qui rend le calcul plus long.
  Il serait bien ainsi de faire des paquets plus gros, mais il faut trouver un compromis.
- La solution que nous avons proposée consiste à faire des communications synchrone bloquants, chaque esclave reste donc bloqué jusqu'à réception de la réponse du maître. Ce qui serait très bine c'est d'utiliser plutôt des communications asynchrone non bloquantes.