**HE-ARC**

2015-2016

**Rapport et test**

**Laboratoire 1 Mandelbrot**

**Gygi Damien et Jeanneret Cyril**

Table des matières

[Introduction 2](#_Toc439926405)

[Cahier des charges 2](#_Toc439926406)

[Contraintes 2](#_Toc439926407)

[Environnement 3](#_Toc439926408)

[Rendu et Barème 3](#_Toc439926409)

[Structure de l’application 3](#_Toc439926410)

[Guide d’utilisation 3](#_Toc439926411)

[Principe de fonctionnement 4](#_Toc439926412)

[Mise en place des threads 5](#_Toc439926413)

[Problèmes rencontrés et solutions 5](#_Toc439926414)

[Dernière colonnes non affichée 5](#_Toc439926415)

[Solution 5](#_Toc439926416)

[Dernier partie de calcul non afficher 6](#_Toc439926417)

[Solution 6](#_Toc439926418)

[Programme plante une fois calcul finit 6](#_Toc439926419)

[Solution 6](#_Toc439926420)

[Résultat obtenu 7](#_Toc439926421)

[Test effectués 7](#_Toc439926422)

[Résultat 1 thread, 1 bloc 7](#_Toc439926423)

[Résultat : 8 threads, 8 blocs 7](#_Toc439926424)

[Résultat : 8 threads, 128 blocs 7](#_Toc439926425)

[Résultat: 69 threads, 120 blocs 7](#_Toc439926426)

[Conclusion 8](#_Toc439926427)

# Introduction

Ceci est le premier laboratoire de programmation concurrente. Nous devons créer un programme pour calculer une courbe de Mandelbrot puis de l’afficher sur l’écran à l’aide de la librairie SDL. Le but est de paralléliser les calculs afin de tirer profit des architectures multi-cœurs.

La courbe de Mandelbrot est construite en appliquant itérativement la fonction Zn+1 =Z2n + C, où C’est le point à évaluer, et Z0 = 0. Le calcul se fait dans le plan complexe, et si une des itérations génère un point en dehors du cercle de rayon 2 centré en (0; 0), alors le point est en dehors de la courbe.

## Cahier des charges

Le but de ce travail pratique est d'implémenter un programme multithread calculant les fractales de l'ensemble de Mandelbrot. Pour ce projet, nous vous fournissons le code d'une implémentation séquentielle qui calcule l'ensemble de Mandelbrot et l'affiche dans une fenêtre graphique à l'aide de la librairie graphique SDL (Simple Direct Media Layer).

Il faut créer un thread master qui s'occupe de diviser le plan complexe à calculer en M blocs ; le master crée ensuite N threads Worker, puis attend qu'ils terminent leur travail.

Les N threads Worker s'occupent chacun de calculer et afficher un des blocs du plan complexe ; lorsqu'un thread Worker a terminé son bloc, il passe alors à un bloc suivant jusqu'à ce qu'il n'y ait plus de bloc à calculer.

**Il est impératif que N soit plus petit ou égal à M ; de plus N et M > 0**

## Contraintes

Le thread main s'occupe de créer le thread master.

• Le programme doit accepter deux arguments sur la ligne de commande : le premier argument est le nombre de blocs et le deuxième le nombre de threads Worker.

• Si l'utilisateur ne spécifie pas exactement deux arguments, alors le programme doit afficher la syntaxe du programme et se terminer.

• le programme doit calculer et afficher le temps mis pour calculer l'image complète.

• il faut s’assurer de désallouer toute mémoire dynamiquement allouée.

• Vous êtes libre d'utiliser les primitives d'exclusion mutuelle ou de synchronisation de votre choix (parmi celles vues jusqu'ici).

• il faut tester le programme avec un nombre varié de valeurs pour M et N ; notez ensuite les timings obtenus.

Essayez de maximiser l'utilisation des processeurs disponibles et évitez toute attente active inutile.

## 

## Environnement

Le programme doit être fait en langage C sur le Framework Code Blocks. Pour l’affichage de l’ensemble de Mandelbrot, nous utiliserons la librairie Simple Direct Media Layer, souvent abrégée SDL.

Il faudra respecter l’architecture « Boss – Worker ». Un thread Master, instancié par le Main, va créer les threads Worker qui calculeront et afficheront l’image. Le nombre de threads qui seront créés par le Master est déterminé par l’utilisateur.

## Rendu et Barème

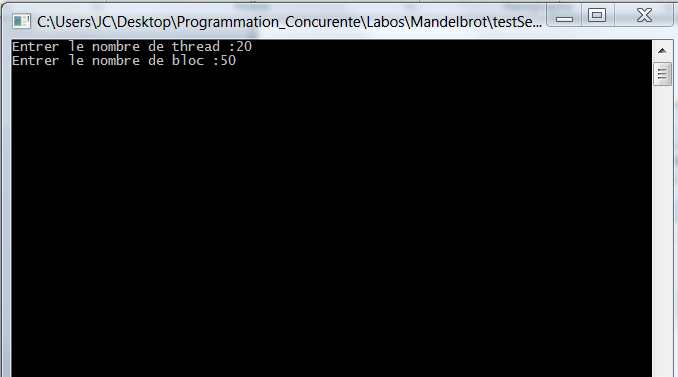
Vous rendez un exécutable avec les dll nécessaires pour son fonctionnement, vous rendez également le code source commenté et un rapport dont vous commentez les tests que vous avez réalisés.

|  |  |
| --- | --- |
| Conception | 10% |
| Exécution et fonctionnement | 30% |
| Tests | 10% |
| Codage | 30% |
| Documentation et commentaires | 20% |

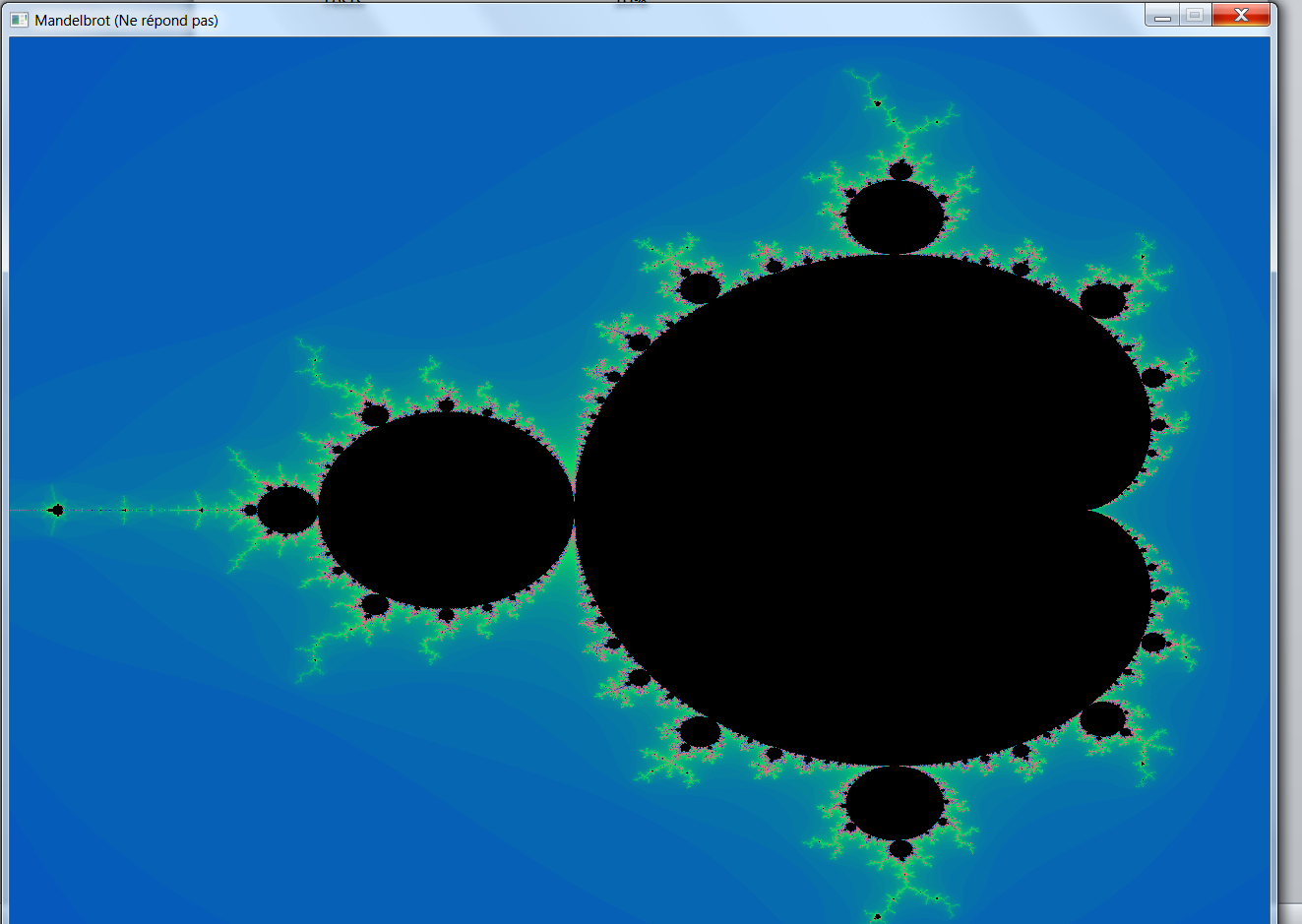
# Structure de l’application

## Guide d’utilisation

Pour utiliser notre application, il suffit d’exécuter le « Lanceur.exe ». Celui-ci va exécuter une console dans laquelle on vous demandera de rentrer le nombre de thread et le nombre de block que vous voulez :



Ainsi ces paramètres vont être envoyés au programme principal qui se chargera d’exécuter les calculs de la Mandelbrot avec les threads. Voici le résultat obtenus :



## Principe de fonctionnement

Notre programme va générer un thread master qui va recevoir tous les paramètres pour le calcul de la Mandelbrot ainsi que les paramètres que l’utilisateur aura choisi pour le nombre de thread et le nombre de blocs. Ce thread master va créer un certain nombre de Thread Worker qui eux vont calculer des colonnes de l’écran et afficher la Mandelbrot. Exemple, notre taille d’écrans en largeur est de 1280 pixel. Si l’utilisateur choisi de séparer cette largeur en 300 blocks et d’utiliser 20 threads, alors ces 20 threads vont se partager ces différentes zone. Voici une démonstration de l’application de ce procéder.



Pour connaitre le temps d’exécution pour le calcul de la Mandelbrot, il suffit de retourné dans le dossier de l’exécutable et d’ouvrir le fichier « Temps.txt ». Voici ce qui vous sera indiqué :



ATTENTION !!!!!!

Pour une raison inconnue, le programme plante si on quitte depuis la croix. De ce fait une invite de commande s’exécutera à la fin de l’affichage de la Mandelbrot. Il suffira d’appuyer sur une touche pour quitter l’application.

Si vous désirer compiler le programme, TestSequenciel depuis code blocks, il faut attribuer les variables thread et nombre de bloc. Pour ce faire exécuté code block puis ouvrer le projet. Ensuite dans l’onglet Project--🡪 setprograms’arguments… --> (noté les valeurs dans l’ordre suivant : nb thread « espace » nb blocks).

## Mise en place des threads

Pour la partie thread, nous avons créé une structure qui contiendra les différentes informations pour les calculs ainsi que les paramètres de la Mandelbrot. Cette structure est passée en paramètres au thread master. Celui-ci va configurer le nombre de thread Worker dont il a besoins.

Pour garantir que les threads n’accèdent pas en même temps au numéro de bloc qu’il doit calculer, on utilise une variable globale que l’on protège à l’aide d’un sémaphore.

## Problèmes rencontrés et solutions

Dans cette partie nous aborderons les problèmes rencontrés ainsi que les solutions ou les moyens de contournement que nous avons trouvé.

### Dernière colonnes non affichée

Comme les threads partages le calcul et l’affichage de la Mandelbrot, la taille de la fenêtre ce retrouve divisé en largeur par le nb de blocks que l’utilisateur a saisie.

Si le nombre de blocks choisis n’est pas un multiple de la largeur de la fenêtre, celle-ci va nous retourner un nombre à virgule. Le problème est que comme le calcul d’un pixel et un carré de 1/1, on ne peut dessiner des parties d’un pixel. Soit on ne peut diviser un pixel que par 1. Résultat, à chaque partie en largeur que nous découpons, nous perdons quelques colonnes de pixel à dessiner qui se reporte à la fin.

### Solution

Pour réglé ce problème, nous avons simplement considérer que le dernier block calculer ne faisais pas la taille définit par l’utilisateur mais la largeur restant jusqu’à la fin de la fenêtre.

### Dernier partie de calcul non afficher

Comme la réactualisation de l’écran se fait lorsque le nombre de block calculer %32 == 0, si les derniers blocks en longueur ne sont pas multiple pour correspondre à cette équation, ils ne seront pas afficher.

### Solution

Nous avons juste forcé la réactualisation dans la main à la fin des calculs, juste avant de désalloué nos mémoires.

### Programme plante une fois calcul finit

Le programme pour x ou y raison, ce met a planté une fois les calculs terminé. A notre avis sa vient du gfx vu que beaucoup d’autre personne de la classe se trouve dans cette situations.

### Solution

Afin de pouvoir visualiser la Mandelbrot avant que le programme plante, nous avons mis une invite de commande à l’aide d’un system(PAUSE) afin qu’il attende la réponse clavier de l’utilisateur pour quitter l’application. Un autre problème est que si l’utilisateur clic sur la fenêtre de la Mandelbrot, celle-ci se bloquera et ne répondra plus. Malheureusement impossible de régler cela.

# Résultat obtenu

Voici pour certaine valeurs le différent résultat obtenu avec le paramètre R de la fonctions. Celui que nous avons laissé par défaut.

## Test effectués

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nb Threads | Nb Blocks | 1  exécution  (Temps ms) | 2 exécutions  (Temps ms) | 3 exécutions  (Temps ms) | 4 exécutions  (Temps ms) | Temps moyen  (ms) |
| 1 | 1 | 3180 | 3186 | 3160 | 3173 | 3174.75 |
| 8 | 8 | 1545 | 1472 | 1470 | 1523 | 1502.5 |
| 8 | 128 | 2303 | 2283 | 2308 | 2284 | 2294.5 |
| 69 | 120 | 1334 | 1278 | 1277 | 1336 | 1306.25 |

### Résultat 1 thread, 1 bloc

Le cas trivial. Les threads n’apportent rien par rapport à un programme séquentiel classique. Le temps d’exécution est bien plus long que le cas multi-threadé. Si on prend en compte que la machine de teste 8 cœur logique on peut observer à l’utilisation de notre application que nous utilisons un seul de ces cœurs.

### Résultat : 8 threads, 8 blocs

Ce résultat montre bien que nos calcul occupe tout les cœur logique de notre processeur. On pourrais potentiellement penser que sa nous donnerais le résultat le plus optimal. Seulement non. Simplement car le calcul d’un block est tellement grand et les calcules de la Mandelbrot son non homogène. De ce fait, un thread mettra beaucoup de temps à effectuer ces opérations.

### Résultat : 8 threads, 128 blocs

Dans cette configuration, chaque cœur logique s’occupe que d’un thread à la fois. Cependant, chaque thread devra calculer plusieurs blocs, en l’occurrence 8. De ce fait le thread prendra le prochain block libre afin qu’il se réutilise.

Chaque bloc à afficher demande moins d’opérations que dans la situation 8 8 car il couvre une moins grande surface de l’image.

### Résultat: 69 threads, 120 blocs

Cette situation est celle-qui nous retourne un temps d’exécution le plus faible. On constate que la largeur de 1280 / 120 nous donne 120 rectangle de 10,6 block largeur. Dans cette situation les 69 thread vont se partagé les 120 rectangle. Comme il seront plusieurs a effectuer le calcul de la mandelbrot et que ce calcul se fera plus rapidement car le nombre de pixel et de 10 par ligne, le résultat sera plus rapide à afficher.

# Conclusion

Il y a une certaine limite au thread pour un nombre de block définit afin que le calcul soit effectué en un minimum de temps. D’apres les testes effectués, entre 63 et 72 thread, le temps d exécution est le plus rapide. On peut meme dire qu’a 64 thread on obtient le meilleur temps pour une limite de 120 block. Voir tableau ci-dessous.

|  |  |  |
| --- | --- | --- |
| Nb Threads | 128 block | 120 block |
| 8 | 1470 | 1901 |
| 63 | 1416 | 1326 |
| 64 | 1344 | 1287 |
| 69 | 1277 | 1336 |
| 72 | 1400 | 1441 |