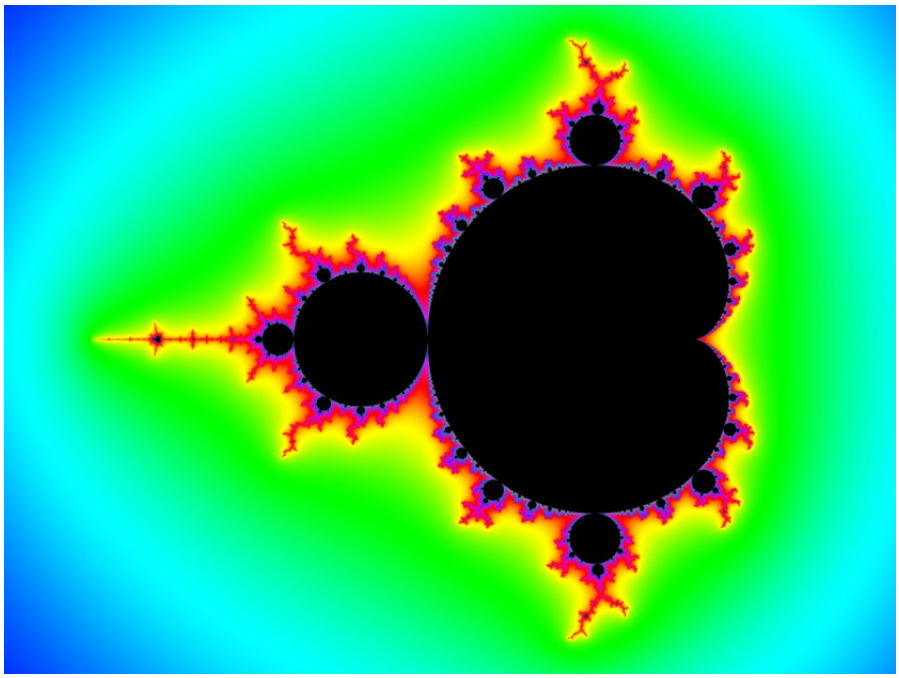
Ecole polytechnique de Louvain-la-Neuve

Rapport

Groupe 10



Nicolas Chauvaux(1995-1600),
Francois Berghmans(2187-1500)

Professeurs responsables :
O. Bonaventure,

12 mars 2018

1 Choix d'implémentation

Pour l'implementation général de notre programme, nous avons suivi la structures données durant la présentation d'architecture. Celle-ci fait intervenir un ou deux $producteur - consommateur$ selon les paramètres. Ce sont les points (4) et (5) de la figure.
Un seul thread est assigné par fichier pour en lire le contenu, et un seul thread

Ensuite pour le calcul des images de fractales, nous avons limité le nombre de thread a un maximum de 15. Nous travaillons en effet sur des processeurs dont le nombre de coeur dépasse rarement 16.

2 Points forts

Nous avons plusieurs parties de notre programme que nous souhaiterions mettre en avant.

- Lecture des fichiers : Dans le but de diminuer le nombre d'appel système, nous avons utilisé la fonction **mmap**. Pour éviter d'occuper la mémoire avec de trop gros fichier nous avons implémente un chargement par bloc de maximum 16Mo.
- Gestion de la mémoire : Comme indiqué dans la figure, la main exécute une fonction *Create_all* (1) qui initialize toutes les variables globals nécessaires. Cela permet de ne pas lancer le programme inutilement en cas de manque de mémoire mais aussi de centraliser les appels à la fonction "malloc". Une fonction *clean_all* (2), s'exécutant à la fin du programme, permet de free correctement ces mêmes variables.
- Gestion des erreurs : Nous avons réfléchit à un moyen d'arrêter tous les threads en cas d'erreur majeur. Nous avons donc défini une variable portant le signal d'arrêt. Cette variables est constamment vérifié par chaque threads. Cela rend notre programme robuste et empêche l'accès à des variables non initialisées.
- Subdivision en fonction : Nous avons séparé le code en un maximum de fonction. Cela permet premièrement un débeugage plus rapide et ciblé. Le deuxième avantage est lié au fait que nous travaillons à deux. Chacun pouvait utiliser les fonctions de l'autre sans difficulté, réduisant ainsi les redondances de code.
- Stockage des données : Nous avons décider de stocker l'ensemble de nos données dans des structures (dynamiques ou statiques) nous permettant ainsi d'empêcher toute fuite mémoire.
- Gestion des threads : Le nombres de threads créés est variables selon plusieurs facteurs permettant ainsi d'optimiser les ressources utilisées par notre programme. Cette gestion s'effectue au point (3).

3 Évaluation du projet

Notre programme n'est pas le plus facile a comprendre, ni le plus concis qu'il soit possible de faire, cependant il offre une certaine robustesse ainsi qu'une transparence sur les erreurs qui se sont produites durant l'exécution. Nous avons tenté de l'optimiser dans les limites de nos capacités. Cependant les performances se dégradent lorsque on y injecte moins de 10 fractales de grosse tailles.
Il est également important de savoir que la gestion des ressources est une de nos priorités. Une grande partie du programme a été écrite pour vérifier que nous n'avions pas de fuite même en cas d'erreur.

Un cas de fuite mémoire que nous ne pouvons éviter est du au fait que nous n'avons pas pu construis de signal handler. Il est donc possible que si un signal ,venant d'un autre processus, vienne perturber le programme, certaine ressources ne soient pas libérer correctement, voir pas du tout.

4 Structure global du programme