LSINF1252 : Systèmes Informatiques - Projet Fractales

de Villenfagne Clothilde - 33991600
de Voghel Brieuc - 59101600

11 mai 2018

1 Introduction

Dans le cadre du cours de Systèmes Informatiques, il nous a été demandé d'implémenter un programme en C, qui prend comme argument des fichiers contenant une liste de fractales. Le programme doit retourner une image BMP de la fractale dont la moyenne des valeurs en chaque point est la plus élevée.
Dans ce rapport nous décrivons nos choix d'implémentation. Nous faisons également une évaluation qualitative et quantitative de notre travail où nous comparons notre programme dans son efficacité de multithreading.

Remarque : dans ce document, "la plus grande fractale" fait référence à la fractale dont la moyenne des valeurs de chaque point est la plus élevée.

2 Choix d'implémentation

Structure des fractales

Notre attention s'est porté ici sur le calcul de la moyenne. Nous ne voulions pas perdre de temps à parcourir plusieurs fois chaque pixel : une première fois pour calculer sa valeur et puis une deuxième fois pour calculer la moyenne. Nous avons donc décidé que lors du calcul de la valeur d'un pixel, cette valeur soit additionnée à une valeur totale.

Il ne suffira plus qu'à diviser ce total par la surface de la fractale pour obtenir la moyenne sans devoir parcourir à nouveau toutes les valeurs. Ceci a directement été implémenté dans `fractal_set_value()` et est un gain de temps non-négligeable.

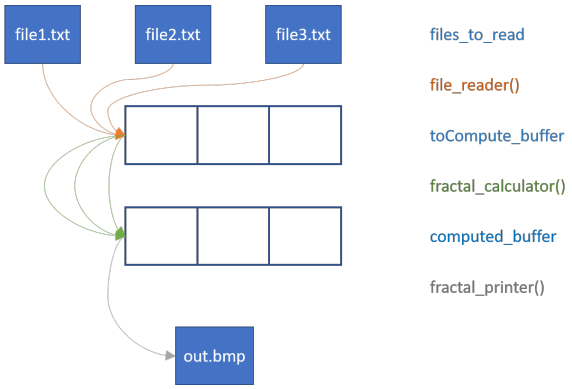

Figure 1 – Visualisation de la structure de notre programme. `file_reader()`, `fractal_calculator()` et `fractal_printer()` sont les fonctions utilisés par les threads.

Implémentation des buffers

Nous avons décidé d'implémenter les buffers sous forme d'une pile (liste chaînée avec `pop` et `push`). Ceci nous semblait être la manière la plus efficace d'accéder aux éléments du buffer. Ce buffer pourrait grâce à ça avoir une taille virtuellement infinie. Nous limitons ceci quand même grâce aux sémaphores.

Comme l'ordre de calcul des fractales n'a pas d'importance, le fait d'avoir implémenté une pile LIFO (last in, first out) n'est pas dérangeant, au contraire ceci évité de devoir parcourir toute une structure avant de pouvoir renvoyer l'élément cherché.

Nous avons tenu à implémenter cette liste chaînée en tant que librairie externe (similaire à `libfractal`) : `libstack`. Elle est compilable séparément et nous l'importons donc dans notre main exactement comme `libfractal`.

Gestion des doublons

Afin de ne pas calculer deux fractales ayant le même nom, nous mettons à jour un string contenant tous les noms des fractales précédemment calculées. Ainsi, nous pouvons vérifier si une fractale a déjà été calculée simplement en parcourant ce string. Le choix de stocker les noms dans un string nous avait l'air le plus facile et rapide à l'exécution.

Gestion de plusieurs fractales qui sont les plus grandes

Une piste d'amélioration envisagée est le fait de pouvoir sortir un plus grand nombre de plus grandes fractales. Ceci pourrait se faire sous la forme d'une pile enregistrant ces plus grandes fractales. Pile qui se fait vider si une nouvelle plus grande moyenne est identifiée. Les fichiers de sortie auraient alors le nom `fichierOut_1.bmp`, `fichierOut_2.bmp`, etc.

Nous avons pensé que ceci serait une bonne piste d'amélioration comme ceci n'a pas spécifiquement été demandé dans l'énoncé de ce projet.

Robustesse générale du code

Un des plus grands points d'attention lors du développement de notre code à été le traitement d'erreurs. Une grosse partie des conditions `if` servent d'ailleurs à ça.

Nous avons tenu à fournir un code suffisamment robuste, capable de faire face à des structures mal formulées dans la liste de fractales fournies par exemple.

Afin de pouvoir démontrer cette robustesse, nous avons inclus un répertoire reprenant quelques fichiers contenant des fractales. Nous avons testé notre code excessivement sur ces fichier.

3 Évaluation qualitative et quantitative

Nous avons effectué plusieurs tests avec notre programme. Notamment sur des fichiers plus larges contenant jusqu'à 1 millions de fractales.

En plus de ces tests, nous avons créé un script supplémentaire dans notre `Makefile`. En exécutant `make test_time`, un script se lance calculant d'abord le temps pris pour extraire des fractales avec un seul thread de calcul, puis avec 5 threads. Les résultats sont très satisfaisants comme vous pouvez le constater sur l'image ci-dessous.

```
Testing effectiveness of multithreading
In order to put pressure on the calculating threads, we will use -d option on relatively big fractals
First we will test with 1 thread and then with 5
Executing time for [./main -d inputs/lvl3_time_consuming.txt o.bmp] :
Option -d present : one image per fractal will be generated
Option --maxthreads not présent : the default value of 1 calculation thread will be used
Lecture des fichiers fini
Sortie des fractales fini
Threads de calcul terminés
Programme exécuté correctement. EXIT
12.57user 0.02system 0:12.67elapsed 99%CPU (0avgtext+0avgdata 14880maxresident)k
0inputs+0outputs (0major+6508minor)pagefaults 0swaps
Executing time for [./main -d --maxthreads 5 inputs/lvl3_time_consuming.txt o.bmp] :
Option -d present : one image per fractal will be generated
Option --maxthreads présent : 5 calculation threads will be used
Lecture des fichiers fini
Sortie des fractales fini
Threads de calcul terminés
Programme exécuté correctement. EXIT
16.29user 0.02system 0:04.89elapsed 333%CPU (0avgtext+0avgdata 20268maxresident)k
0inputs+0outputs (0major+11320minor)pagefaults 0swaps
Try running this script multiple times to get a better picture of the effectiveness
```

FIGURE 2 – Appel à `make test_time`

Nous pensons que l'utilisation de ce script permet d'avoir une bonne évaluation qualitative de notre code et nous vous invitons à le modifier à votre goût pour en juger par vous-même.