LSINF1252
Rapport de projet 2 : Fractales

Aigret Julien (8343-13-00) Vanvyve Nicolas (6590-13-00)

6 mai 2016

1 Architecture

Nous avons décider d'implémenter un producteur consommateur assez classique. Nous avons un producteur par fichier d'entrée, ils lisent ligne par ligne les fichiers et transforment le texte en une structure fractal. La fractale est ensuite placé dans un tableau de taille fixe.

La taille du tableau est égale au nombre de threads de calculs autorisé par l'utilisateur. Nous bloquons les producteur à l'aide d'un sémaphore, puisque les threads de lecture sont beaucoup plus rapide que les threads calculs.

Nous utilisons un deuxième sémaphore pour gérer les consommateurs. Des que un case du tableau est remplie, un consommateur libre vide la case du tableau et calcule la valeur de chaque pixel de la fractale ainsi que sa moyenne. Le thread de calcul n'est lancé que une seule fois en tourne en boucle tant que il y a encore des fractales a calculer. Cela nous permet de garder, dans le thread, la meilleure fractale que le consommateur a traité, ainsi que la moyenne de celle ci. A chaque itération la moyenne est comparée a celle de référence et la moins bonne des deux fractales est libérée.

Lorsqu'il n'y a plus rien a traiter dans le tableau, les consommateurs retourne la meilleure fractale qu'ils ont traité, nous parcourons ensuite les différentes valeurs moyennes afin de sélectionner la meilleure de toute, les autres sont libérées. La meilleure est ensuite "imprimée" et libérée.

2 Choix d'implémentation

Afin de transformer une ligne de texte en une fractale, nous utilisons la fonction strtok_r qui extrait la première partie d'une string avant un délimiteur. L'avantage de strtok_r par rapport à strtok c'est qu'il est safe en multithread, strtok utilise un même buffer pour ses différents appel, et peut provoquer une erreur si deux appels se font simultanément.

Au départ nous avions le tableau de la fractale à l'aide de double pointeur. Nous créions un tableau de pointeur de la taille de la hauteur de l'image et chacun pointait vers un tableau de la taille de la largeur. Cette solution était inutilement compliquée, un tableau étant de toute façon stocké linéairement dans la mémoire nous avons choisi de stocker le tableau 2D en une dimension de taille largeur * hauteur.

3 Difficultés rencontrées

Notre code a assez vite été fonctionnel pour une seule fractale mais nous avions des erreurs des que nous testions plus de fractales. Ce n'est que le jour de la remise du travail que nous nous sommes rendu compte que le consommateur allait chercher la fractale dans le tableau mais ne remettait pas cette dernière à NULL. Cela provoquait des erreurs "double free or corruption"

4 Performances

Voici les résultat de nos tests de rapidité fait avec les fichier d'input fournis [1] :

Fichier 1 (5 800*800)	Fichier 2 (1000 800*800)	Fichier 3 (1 000 000 8*8)
1 thread : 29 sec	10 threads : 21 sec	10 threads :18 sec
3 threads : 12 sec	100 threads : 27 sec	100 threads : 19 sec
5 threads : 12 sec	300 threads : 22 sec	1000 threads : 11 sec

1. Fait sur un MacBook Pro mid 2012 8GB de RAM 2.5GHz dual-core Intel Core i5 processor (Turbo Boost up to 3.1GHz) with 3MB L3 cache