Programmation temps-réel, Projet (rattrapage session 1)

Aline Huf. <alinehuf@ai.univ-paris8.fr>

2016-2017

1 Consignes générales

Réalisation du projet :

- **Ce projet est à réaliser seul**
- Les différentes versions du programme doivent être réalisées en langage C ou C++, utiliser la bibliothèque OpenCV (OpenGL interdit) et tourner sous Linux.
- Privilégiez un style clair et lisible
- Nommez les fonctions et les variables avec des noms appropriés
- Commentez votre code
- Soignez votre indentation
- **Les documents accompagnant de projet** (présentation, graphique) **doivent être fournis au format PDF** pour s'assurer que la mise en page soit préservée (et que le correcteur puisse les ouvrir).
- Regroupez les fichiers de votre programme dans un dossier.

Dépôt du projet :

- Rendre une archive .tar.gz contenant d'un répertoire nommé par vos NOM, PRENOM et NUMERO (sur le modèle NOM-PRENOM-NUMERO.tgz).
- Le répertoire de l'archive contient les différentes versions du programme, les scripts de test, les graphiques, un Makefile et le rapport au format PDF. Déposer l'archive sur : https://moodle.univ-paris8.fr/moodle/course/view.php?id=1373

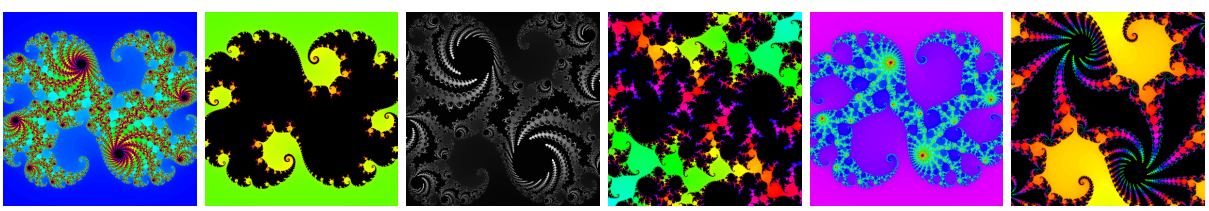
Date limite de dépôt du projet : le ~~20 mars 2017~~ 25 avril 2017 à 12h (le dépôt sera fermé à midi pile).

Vous pouvez rendre votre projet avant la date limite. Dans ce cas, envoyez-moi un mail pour m'avertir afin que je vous corrige.

2 Présentation du projet

Ce projet ce divise en deux parties. Dans un premier temps vous allez réaliser un programme en C ou C++ qui calcule en parallèle, à l'aide de threads POSIX, une fractale de Julia (Chaque pixel de l'image peut se calculer indépendamment des autres). Vous réaliserez un banc de test pour évaluer l'évolution du temps moyen de calcul en fonction du nombre de threads. Dans un second temps vous réaliserez une seconde version interactive du programme permettant de modifier et recalculer la fractale à la volée.

Voici un exemple des images qui pourront être obtenues :



Chacune des étapes de la conception du programme (indiquées ci-après) devra être présentée dans un rapport écrit. Vous utiliserez OpenCV pour afficher la fractale. Les fichiers `opencv_exempleC.cpp` et `opencv_exempleCPP.cpp`

donnent un exemple minimal de l'utilisation d'openCV en C et en C++ avec toutes les fonctionnalités dont vous aurez besoin. Les deux exemples sont compilés avec g++ pour plus de simplicité. Pour compiler avec gcc, il est conseillé d'utiliser cmake (voir la documentation).

3 Calcul de la Fractale de Julia

Étant donnée une constante complexe c, on définit la suite complexe $z_{n+1} = z_n^2 + c$. Chaque constante c correspond à une image différente (la fractale aura une structure différente). Á chaque point de l'image de coordonnées (x', y'), on associe le nombre complexe $z_0 = x + i.y$ qui permet d'initialiser la suite. On peut alors calculer les termes de la suite et étudier son comportement. Si la suite est bornée alors elle fait partie de l'ensemble de Julia et on mettra un pixel noir. On décidera que la suite est bornée si au bout d'un nombre d'itérations N, le module de la suite n'a pas dépassé une certaine valeur V. Dans le cas contraire, on mettra un pixel plus ou moins blanc en fonction du nombre d'itérations effectué avant de dépasser la valeur V.

Les images présentées en exemple on été obtenues pour les valeurs $N = 300, V = 4, x \in [-1, 1]$ et $y \in [-1, 1]$. Si votre image a une taille de 1024×1024 pixels, il faut donc transposer les coordonnées (x', y') de chaque pixel pour obtenir la valeur $z_0 = x + i.y$ correspondante.

Je vous conseille, pour réaliser un premier affichage de la fractale et ajuster les paramètres, d'écrire une version alpha de votre programme qui calcule la fractale de manière séquentielle (sans threads). Vous aurez besoin d'une fonction pour calculer la valeur d'un pixel en fonction de x', y' et c. Ensuite vous pourrez utiliser cette fonction pour calculer chaque pixel de l'image.

Voici quelques exemples de valeurs de c : $-0.5+0.64i$, $0.3+0.5i$, $-0.038088+0.97i$, $0.285+0.013i$, $0.285+0.01i$, $-1.41702285618+0i$.

Pour affichez une fractale en couleurs, faites une petite recherche sur internet pour trouver une manière de la colorer (indice : on peut passer de HSV en RGB...).

4 Programme 1 et banc de test

Pour paralléliser le calcul de la fractale avec N threads, vous pourriez découper l'image en N bandes. Cependant, en fonction de la fractale (valeur de c) et du nombre d'itérations réalisées pour le calcul de chaque pixel, le calcul de chaque bande peut être plus ou moins long. Pour éviter que certains threads aient terminé leur travail avant les autres et que les threads ayant le plus de travail ne ralentissent tout le calcul, **il conviendra de réaliser un 'pool' de fragments de l'image à calculer dans lequel chaque thread viendra piocher** dès qu'il a terminé le calcul du fragment précédent. L'usage de mutexes (ou autre synchronisation) sera nécessaire pour s'assurer que plusieurs threads ne tentent pas de calculer les mêmes pixels au même moment. L'utilisation d'un mutex par pixel serait très couteux en temps. Il conviendra donc de trouver la bonne taille pour les fragments de l'image : ni trop petit pour éviter un surcout dû aux mutexes, ni trop gros pour éviter qu'un thread passe plus de temps que les autres à calculer sa partie de l'image.

Vous devrez réaliser **un banc de tests en plusieurs partie pour évaluer le temps de calcul de la fractale** (temps moyen ou temps dans le pire des cas, à vous de voir) **et pour répondre aux questions suivantes** :

1. Quelle est l'influence de la taille des fragments de l'image sur le temps de calcul de la fractale ? et quel est la taille idéale au delà de laquelle on a un surcout dû aux mutex et en deçà de laquelle un thread ralenti les autres ?
2. Quel est le nombre de threads idéal pour le calcul de l'image en deçà duquel le temps de calcul est plus long et au delà duquel le temps gagné n'est plus significatif ?

Pour faciliter les tests, **votre programme prendra des arguments sur la ligne de commande** correspondants aux valeurs pouvant faire varier le temps d'exécution (partie réelle et imaginaire de c, nombre de threads, taille de l'image et taille des fragments). Un (ou plusieurs) script shell (ou python, ou autre) sera utilisé pour lancer le programme avec différentes valeurs et réaliser les tests.

Remarque 1 : les threads doivent être lancés au début de l'application et stoppés à la fin. Les techniques de synchronisation vues en cours doivent être utilisées pour lancer/relancer le calcul ou l'interrompre. Si vous n'observez aucun gain en parallélisant le calcul (voire si le calcul est ralenti) c'est que votre utilisation des threads ou des méthodes de synchronisation n'est pas correcte.

Remarque 2 : d'autres applications sont susceptibles de tourner en tâche de fond sur votre machine et peuvent perturber les résultats. Si je veux, par exemple, étudier la variation du temps de calcul avec 2, 4, 8 et 16 threads,

je dois réaliser plusieurs tests (10 grand minimum et 10000 dans l'idéal) pour chaque nombre de thread et utiliser la moyenne du temps obtenu pour chaque nombre de threads plutôt qu'une valeur unique. Sur un graphique, il est nécessaire de représenter également l'écart type pour indiquer si les temps obtenus variaient beaucoup ou si ils étaient tous proches de la moyenne.

Dans le rapport, vous devrez présenter votre protocole expérimental (ce qui est testé, sur quelle machine, nombres de threads utilisés, valeurs entre lesquelles vous faites varier les paramètres, etc...). Vous devrez **présenter les résultats du banc de test sous forme de graphiques gnuplot** (courbe présentant la variation du temps moyen pour chaque taille de fragment ou chaque nombre de threads avec indication de l'écart type). Vous rédigerez également une critique des résultats (description des points intéressants observés et explication/réponses aux questions posées)

5 Programme 2 : Interaction avec l'utilisateur

Vous réaliserez une seconde version de votre programme permettant à l'utilisateur d'interagir avec votre programme et de modifier la fractale à la volée. Le thread principal (processus parent) se chargera de l'affichage et de l'interaction avec l'utilisateur. Les threads secondaires réaliseront le calcul de la fractale et modifieront les pixels de l'image. Un exemple de boucle d'interaction est donnée dans les fichiers `opencv_exempleC.cpp` et `opencv_exempleCPP.cpp`.

L'utilisateur devra, en tapant sur différentes touches du clavier, pouvoir :

- augmenter ou diminuer les parties réelles ou imaginaires de c
- zoomer/dé-zoomer
- modifier la taille de l'image
- générer une capture de l'image
- modifier les couleurs
- passer du noir et blanc à la couleur
- stopper proprement le programme

La fractale sera modifiée et ré-affichée à la volée (chaque pixel est affiché dès qu'il est calculé pour ne pas générer une attente).

6 Consignes pour le rendu du projet

Le projet est à réaliser seul. Vous devez rendre le projet sous forme d'une archive compressée contenant :

- les sources des deux versions du programme (les programmes doivent prendre les variables sur la ligne de commande).
- les scripts utilisés pour les tests
- **Un seul fichier Makefile** permettant (selon les cibles) de compiler les sources, de lancer les scripts des bancs de tests, de générer les graphiques avec gnuplot.
- le rapport au format PDF

Le rapport doit contenir :

* Pas de blabla inutile sur l'historique des fractales de Julia, allez directement au but

1. **Décrivez la manière dont vous avez implémenté le calcul de la fractale :** comment vous découpez l'image, comment vous stockez les fragments, comment vous les distribuez aux threads, etc...
2. **Décrivez le protocole de test :** comment lancer les tests, que fait votre script de test, sur quelle machine vous l'avez lancé. Qu'est-ce que vous testez et dans quel but. A quel résultat vous vous attendez et à quelle question le test doit permettre de répondre.
3. **Présentez les résultats :** Graphique avec une légende, expliquez à quoi correspond chaque courbe, est-ce que les résultats sont étonnants par rapport à ce que vous attendiez ou bien correspondent-ils à ce qui était prévu.
4. **Analysez les résultats :** expliquez en quoi les résultats vous permettent de répondre aux questions posées et quelles réponses vous avez trouvées.
5. **Présentation de la version interactive :** expliquez comment lancer le programme, comment interagir avec lui (documentation utilisateur). Expliquez ensuite comment vous avez implémenté cette interactivité.