HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Prof. Yann Thoma

Jonas Chapuis

Laboratoire de Programmation Concurrente

semestre printemps 2017 - 2018

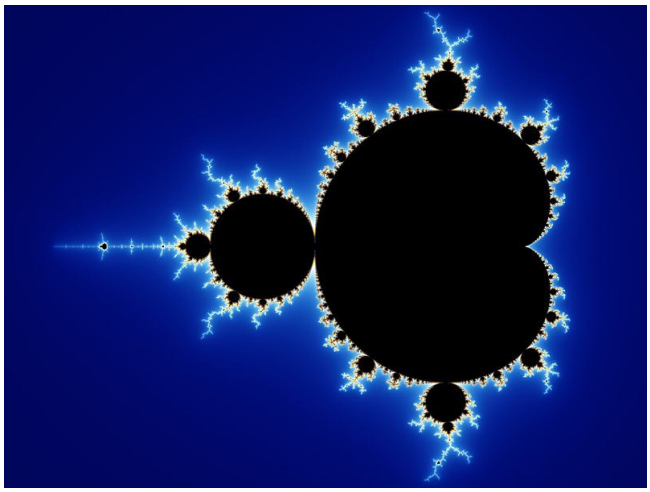
Pattern boss-worker simple

Temps à disposition : 4 périodes

1 Objectifs pédagogiques

— Réaliser un système de type boss-worker avec lancement de threads et attente de terminaison

2 Cahier des charges



Le calcul d'une courbe de Mandelbrot nécessite un grand nombre d'opérations effectuées pour chaque Pixel à afficher. Il est dès lors possible de paralléliser les calculs afin de tirer profit des architectures multi-coeurs.

Pour rappel, la courbe de Mandelbrot est construite en appliquant itérativement la fonction $Z_{n+1} = Z_n^2 + C$, où C est le point à évaluer, et $Z_0 = 0$. 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. Si tel est le cas, le nombre d'itérations pour atteindre la frontière doit être retourné.

Un nombre maximum d'itérations est défini, et si ce nombre est atteint sans que Z n'ait dépassé le cercle de rayon 2, le point sera colorié en noir, et sinon une fonction permet de calculer sa couleur en fonction du nombre d'itérations effectuées pour sortir du cercle.

Une application fonctionnelle vous est gracieusement fournie. Il s'agit en fait d'un exemple fourni avec Qt. Compilez-le et observez son fonctionnement. Observez le code, et notamment la manière dont est fait le calcul. Vous devriez voir, dans le fichier `renderthread.cpp` la manière dont le calcul est fait. La classe `RenderThread` est déjà un thread utilisé pour ce calcul, ce qui évite de *freezer* la partie graphique. Ce thread lance un calcul d'une image complète, et lorsque c'est fait, en relance un avec un nombre maximum d'itérations supérieur. De ce fait, l'image affichée va se raffiner au fur et à mesure des calculs.

Si vous lancez l'application vous devriez pouvoir observer, avec le System Monitor, qu'un coeur de processeur est occupé à 100%.

Le but de ce labo est donc pour vous de modifier le afin de paralléliser le calcul des points. Partez du code présent dans `renderthread.cpp` et ajoutez la parallélisation. L'idée est de décomposer l'image en zones et de déléguer aux threads le calcul des zones. Etant donné que nous n'avons pas encore vu une manière efficace de synchroniser des threads, votre implémentation se contentera de lancer les threads et attendre qu'ils terminent, et ce avec ce que l'API Qt vous offre.

A priori vous aurez à créer une sous-classe de `QThread` pour y embarquer une partie de la logique de la fonction `RenderThread::run()`. A vous de voir comment passer les informations de manière pertinente. L'idée est ici de paralléliser la boucle de calcul d'une image. `RenderThread` devrait donc, pour chaque passe, créer les threads, les lancer, attendre leur terminaison, et mettre à jour l'image (cf. `emit renderedImage(image, scaleFactor)`).

L'application fournie sort, sur la sortie standard, le nombre de millisecondes nécessaire pour le calcul de chaque passe. Vous devriez observez une différence entre la version de base et celle améliorée.

3 Points sur lesquels être attentifs

Les points suivants devraient nécessiter une attention particulière :

- Le passage des information pertinentes aux threads workers
- La gestion de abort/restart qui devrait permettre de terminer les workers lorsque nécessaire
- La terminaison du programme (il faut qu'il se termine correctement)
- Une méthode statique `QThread::idealThreadCount()` pourrait vous être utile

4 Travail à rendre

- Les modalités du rendu se trouvent dans les consignes qui vous ont été distribuées.
- La description de l'implémentation, ses différentes étapes, la manière dont vous avez vérifié son fonctionnement et toute autre information pertinente doivent figurer dans le fichier `README.md` et dans le code.
- Inspirez-vous du barème de correction pour connaître là où il faut mettre votre effort.
- Vous pouvez travailler en équipe de deux personnes au plus.

5 Barème de correction

Conception, conformité au cahier des charges, structure et simplicité	40%
Exécution et fonctionnement	20%
Codage	10%
Documentation de votre solution	20%
Commentaires au niveau du code	10%