Parallélisation de boucles en OpenMP

Résumé

Nous nous intéressons à la parallélisation d'algorithmes itératifs produisant des séquence d'images. Nous traiterons un premier cas où la charge de calcul est constante sur toute l'image. Dans le second cas, certaines zones de l'image demanderont beaucoup plus de calcul que d'autres ; on cherchera alors à uniformiser la charge de travail des différents threads.

1 Cadre du mini-projet

Ce mini-projet est à faire en binôme et à rendre pour le mardi 13 février 23h59. Votre rapport, sous forme de fichier au format PDF, contiendra les modifications apportées au code, les conditions expérrimentales et les graphiques obtenus accompagnés chacun d'un commentaire le décrivant et l'analysant.
Les machines utilisées doivent être des machines du CREMI dotées d'au moins 12 cœurs physiques. Les expériences menées devront être peu bruitées : il est inutile de faire des expériences sur une machine chargée. On pourra comparer différentes configurations matérielles.
Vous placerez ce rapport dans `/net/cremi/pwacreni/PROJET-PAP` le nom de votre rapport sera de la forme : `binome1-binome2-clef-secrete.pdf`
Less sources de l'application sont sous `~pwacreni/PAP-2018/projet-mandelbrot .`

2 Prise en main de l'application

Consultez le contenu du fichier `src/scrollup.c`. Le fichier contient simplement une fonction (`scrollup_compute_seq`) qui code une version séquentielle du noyau de calcul « scrollup ». Ce noyau manipule deux images : `cur_img` et `next_img`. À chaque itération, on lit les pixels de `cur_img` et on modifie ceux de `next_img`. À la fin de l'itération, les rôles sont permutés.
Pour lancer le programme en exécutant ce noyau de calcul, tapez simplement (tapez la touche ESC pour quitter) :

```
./prog -l images/mini-stars.png -k scrollup -v seq
```

Les options utilisées sont :

- **-l images/mini-stars.png** pour charger l'image initialement dans le tableau `cur_img`.
- **-k scrollup** pour indiquer que l'on s'intéresse à une fonction de calcul dont le nom commence par `scrollup_compute` (c'est optionnel dans ce cas précis, car la valeur par défaut est « scrollup »).
- **-v seq** pour indiquer que l'on s'intéresse à la version de la fonction suffixée par `_seq` (optionnel également dans ce cas).
- **-n** pour indiquer que l'on ne s'intéresse qu'à la mesure du temps d'exécution.
- **-d t** pour observer le temps de calcul de chaque itération.

3 Cas d'un calcul régulier

3.1 Adaptation du code

Dupliquez le code de la fonction `scrollup_compute_seq` pour créer une nouvelle fonction `scrollup_compute_omp`.
Ajoutez une direction de parallélisation OpenMP à l'endroit vous semblant le plus approprié.
Testez en exécutant :

```
./prog -l images/shibuya.png -k scrollup -v omp
```

Pour comparer le gain obtenu par rapport à la version séquentielle, il faut désactiver l'affichage (qui ralentit les calculs) et exécuter un nombre d'itération significatif. Par exemple, vous pouvez comparer les temps de 500 itérations en séquentiel :

```
./prog -l images/shibuya.png -k scrollup -n -i 500 -v seq
```

avec 500 itérations de la version parallèle :

```
./prog -l images/shibuya.png -k scrollup -n -i 500 -v omp
```

Produire une version basée sur politique de distribution dynamique. On appellera cette fonction `scrollup_compute_omp_d`. Mesurer la vitesse d'exécution de votre programme :

```
./prog -l images/shibuya.png -k scrollup -n -i 500 -v omp_d
```

3.2 Expériences à mener

Il s'agit de produire un graphique montrant l'accélération obtenues par vos programmes parallèles en fonction du nombre de threads utilisés.
Modifier le script shell `~pwacreni/PAP-2018/experiences/lancer-expe.sh` afin de produire deux fichiers de mesures l'un pour la version statique, l'autre pour la version dynamique :

```
PARAM="./prog␣-l␣images/shibuya.png␣-k␣scrollup␣-n␣-i␣500␣-v"

execute omp
execute omp_d
```

Utiliser le script R situé dans le répertoire `~pwacreni/PAP-2018/experiences` pour produire le graphique demandé en remplaçant `XXX` par le temps de la version séquentielle :

```
Rscript tracer-speedUp.R omp omp_d XXX
```

4 Cas d'un calcul déséquilibré

Nous allons maintenant nous intéresser à des calculs plus intenses pour chaque pixel, et surtout irréguliers sur la surface de l'image. Le calcul des ensembles de Mandelbrot, à une échelle variant à chaque itération, devrait faire l'affaire.

4.1 Prise en main du code

La fonction `mandel_compute_seq` (dans `src/mandel.c`) donne la version séquentielle du noyau qui nous intéresse. Jetez-y un oeil. Notez qu'on n'utilise plus qu'une seule image (`cur_img`), dont on écrase les pixels à chaque nouvelle itération. Puis essayez :

```
./prog -s 1024 -k mandel -v seq
```

Une version parcourant l'image en « tuiles » (`mandel_compute_tiled`) vous est également fournie. Pour l'essayer :

```
./prog -s 1024 -k mandel -v tiled
```

4.2 Adaptation du code

Écrivez quatre versions parallèles de ces deux versions) :

omps parallélisation de `mandel_compute_seq` en utilisant une politique de distribution statique.

ompd parallélisation de `mandel_compute_seq` en utilisant une politique de distribution dynamique.

omptiled parallélisation de `mandel_compute_tiled` en utilisant une politique de distribution dynamique des tuiles à l'aide de la directive `collapse`.

omptask parallélisation de `mandel_compute_tiled` en créant une tâche pour chaque tuile.

Déterminez les paramètres qui donnent de bons résultats « en moyenne » : valeur `k` pour `schedule(static, k)`, valeur de `GRAIN` pour les versions tuilées... Il est en effet possible de lancer le calcul dans des zones différentes de l'espace. Par défaut, vous utilisez la configuration 1 (cf les valeurs initiales des variables globales `leftX`, etc. dans le fichier `src/mandel.c`) dans laquelle les pixels coûteux en calcul apparaissent progressivement sur la droite de l'image. Mais si vous essayez la configuration 2, vous aurez alors affaire à une forte charge apparaissant depuis le bas de l'image.

4.3 Expériences à mener

Il s'agit de produire un graphique montrant l'accélération obtenues par vos versions parallèles en fonction du nombre de threads utilisés. On utilisera quelques centaines d'itérations.
Il n'est pas demandé de tracer une courbe pour chaque configuration de départ et chaque valeur possible de paramètre. Vous indiquerez simplement quels sont la configuration et les paramètres choisis pour chaque courbe.
Notez que, si en phase de mise au point de vos algorithmes vous pouvez accélérer le rendu en diminuant la taille de l'image (option `-s`), veillez bien à utiliser des images de taille significative lors de la production de vos courbes.