Département d'Informatique

TP - ASR7 Programmation Concurrente

—

TP Noté : calcul d'histogramme d'image

Sylvain Brandel, Yves Caniou, Guillaume Damiand, Meriem Ghali,
Laurent Lefèvre, Thibaut Modrzyk, Grégoire Pichon, Alec Sadler,
Florence Zara, Jerry Lacmou Zeutouo

Printemps 2024

Aucun document ni moyen de communication n'est autorisé.

I Consignes générales

Dans un premier temps, téléchargez le fichier `histogram.cpp` à l'adresse `http://tpnote.tpr.univ-lyon1.fr/LIF12/`.

Vous rendrez un unique fichier, que vous déposerez via votre navigateur internet à l'URL `http://tpnote.tpr.univ-lyon1.fr/LIF12/`. **Attention :** n'oubliez pas de cliquer sur "envoyer" après avoir sélectionné le fichier.

Le fichier rendu à la fin du TP doit impérativement être compilable, sans warnings même avec `-Wall -Wextra`, et il doit bien sûr s'exécuter sans lever d'erreur dans la plupart des cas. Le code écrit doit être clair et commenté. Il est préférable de traiter convenablement peu de questions plutôt que de fournir des codes non fonctionnels pour chaque partie.

Vous ne devez pas éditer les fonctions déjà écrites, sauf les lignes marquées `// A CHANGER`. Vous ne pouvez donc pas utiliser/inclure de nouvelles bibliothèques (*library*).

Lisez bien tout le sujet avant de commencer et faites des schémas afin d'être sûr d'avoir bien compris les différentes parties.

II Introduction et version séquentielle

Le but de ce TP est de calculer l'histogramme d'une image, représentée par un tableau à deux dimensions de pixels (de taille `WIDTH * HEIGHT`), chaque pixel étant un nombre de 0 à 255 (donc une valeur de type `unsigned char`). L'histogramme est un tableau `h` de taille 255 tel que `h[i]` est le nombre de pixels de valeur `i` (c'est à dire le nombre de couples `x, y` tels que `image[x][y] == i`). Un calcul séquentiel d'histogramme vous est donné dans la fonction `compute_histogram`. Le calcul parcourt l'image passée en paramètre et remplit l'histogramme (préalablement initialisé à 0) au fur et à mesure du parcours.

Le but du TP est d'écrire une version parallèle de ce calcul d'histogramme, qui lancera `NB_THREADS` threads (`NB_THREADS` est une constante définie pour vous en haut de fichier). La

difficulté est que chaque pixels de l'image peut contribuer à la valeur de chaque case de l'histogramme. Une version naïve du calcul où on se contenterait de découper l'image en tranches et de lancer un thread sur chaque image poserait problème : tous les threads pourraient accéder à toutes les cases de l'histogramme. Il faudrait donc verrouiller un mutex à chaque accès, ce qui rendrait le programme parallèle bien plus lent que la version séquentielle !

Pour commencer, vous pouvez télécharger le fichier `histogram.cpp`, puis le compiler avec la commande :

```
g++ -std=c++11 -pthread -g -Wall -Wextra histogram.cpp -o histogram
```

Vous pouvez exécuter le programme et vérifier qu'il fonctionne. Pour l'instant il fait 3 fois le même calcul de la même manière.

III Version parallèle 1 (15 points)

La solution que vous allez coder est la suivante :

- L'image est découpée en `NB_THREADS` tranches, chacune allant de l'indice `t * WIDTH / NB_THREADS` (inclus) à `(t + 1) * WIDTH / NB_THREADS` (exclus), où `t` est le numéro du thread (de 0 à `NB_THREAD - 1`).
- Le résultat du calcul, que nous appelerons l'*histogramme global*, est une variable partagée par tous les threads.
- Chaque thread va parcourir sa tranche d'image et calculer l'histogramme de cette tranche en ignorant le reste de l'image. On appellera cet histogramme l'*histogramme local*.
- Une fois le calcul de l'histogramme local terminé, le thread va ajouter la valeur de chaque case de l'histogramme local à la case correspondante de l'histogramme global.
- Au final, l'histogramme global renvoyé par le calcul est la somme des histogrammes locaux.

Pour cela, vous devez réaliser les étapes suivantes :

- (2 points) Écrire une version fonction `histogram_slice()` qui calcule l'histogramme local d'une tranche d'image.
- (Facultatif) Faire une version séquentielle du calcul qui appelle cette fonction `NB_THREADS` fois dans une boucle. Cette version n'est pas comptée dans la note mais vous permettra de tester `histogram_slice()`.
- (3 points) Écrire la fonction qu'exécutera chaque thread. Cette fonction appelle `histogram_slice()`. Réfléchissez bien aux arguments à passer à cette fonction.
- (Facultatif) Faire une version séquentielle qui appelle cette fonction `NB_THREADS` fois dans une boucle (ici encore : non-compté dans la note mais peut vous aider à tester).
- (3 points) Écrire la fonction `histogram_threads_v1()` qui lance `NB_THREADS` threads et fait le calcul parallèle de l'histogramme.
- Tester le résultat (nous vous fournissons un test via la fonction `compare_hist` qui vérifie que votre fonction `histogram_threads_v1()` renvoie bien le même résultat que la version séquentielle `compute_histogram()`).

— (2 points) Vérifier que les performances sont cohérentes (le code fourni vous affiche le temps-réel, alias « wall-clock time » et le temps CPU pour chaque calcul). Écrivez les résultats que vous obtenez et justifiez-les en commentaires en bas du fichier.

Complément de barème :

— Clareté du code, pertinence des commentaires : 3 points

— Absence de warnings à la compilation : 2 points

Quelques conseils :

— Les fonctions qui calculent des histogrammes (locaux ou globaux) prennent un histogramme en argument, passé par pointeur, et écrivent dans la valeur pointée par le pointeur. Ces fonctions supposent que l'histogramme est initialisé à 0 avant appel. Pensez donc à appeler la fonction `init_hist()` avant ces calculs.

— Réfléchissez aux variables partagées de votre programme, et utiliser un mutex lorsque c'est nécessaire.

— L'outil `valgrind` peut vous aider à debugger votre code. Pensez à compiler avec l'option `-g`.

— Compilez et testez régulièrement.

IV Version parallèle 2 (8 points)

Si (et seulement si) le temps le permet, vous pouvez coder une autre version parallèle dans la fonction `histogram_threads_v2()` :

— L'image est découpée de la même manière.

— Avant de lancer les threads, on crée un tableaux d'histogrammes de taille `NB_THREADS`, et on passe une case de ce tableau à chaque thread. Chaque thread de calcul écrit l'histogramme local dans la case du tableau correspondante.

— Après avoir terminé tous les calculs, le thread principal fait la somme des histogrammes locaux.

L'intérêt de cette version est qu'elle permet de se passer de mutex, mais elle est essentiellement équivalente à la précédente.