

Programmation Concurrente

Convolution 2D

Travail Pratique

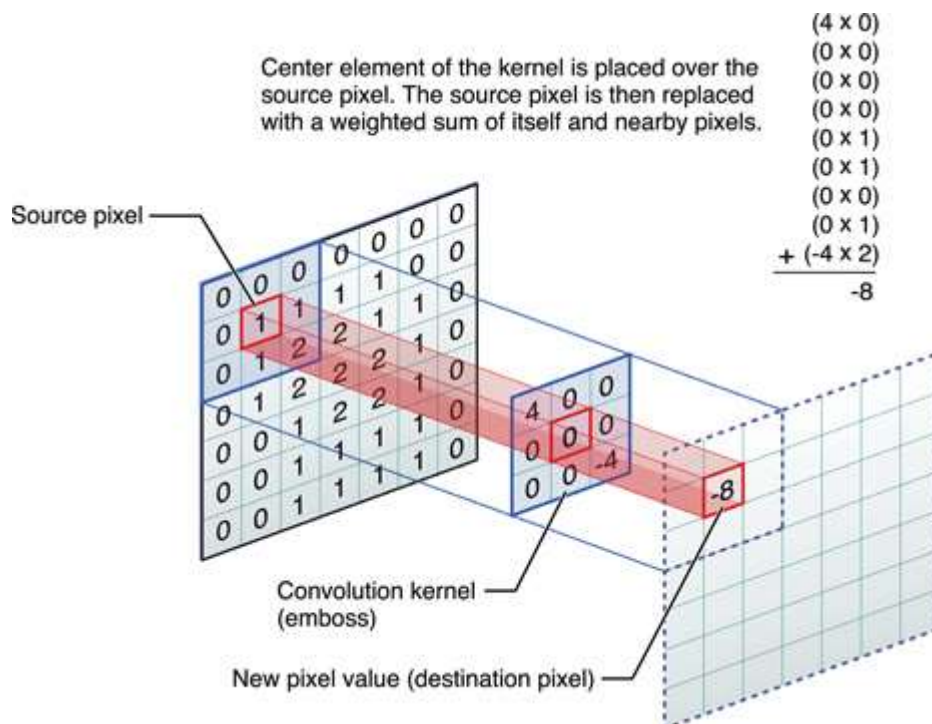
But

La convolution 2D discrète est une opération très utilisée en traitement d'image, car elle permet d'appliquer un large éventail de filtres à une image, comme par exemple la détection de contours, les effets de flou, l'accentuation du piqué, etc.

Le désavantage de celle-ci est qu'elle est relativement lourde à calculer. Le but de ce travail pratique est de paralléliser le calcul de la convolution 2D discrète afin d'en accélérer le calcul, ceci grâce à l'utilisation de threads sur les architectures processeurs modernes multi-coeurs.

La convolution 2D discrète C entre le kernel K (filtre) de dimension N et l'image I est définie au point (x,y) de l'image par l'équation suivante :

$$C(x,y) = \sum_{j=-\frac{N}{2}}^{\frac{N}{2}} \sum_{i=-\frac{N}{2}}^{\frac{N}{2}} K(i+\frac{N}{2}, j+\frac{N}{2}) I(x+i, y+j)$$



Cahier des charges

Lisez attentivement le document « Consignes Travaux Pratiques.pdf » qui décrit exhaustivement les points à respecter en ce qui concerne l'implémentation du code ainsi que le rapport à rendre.

Implémentation

Vous développerez un programme, fonctionnant en ligne de commande, qui appliquera la convolution 2D d'un filtre sur une image. Le programme doit permettre à l'utilisateur de spécifier l'image pour laquelle faire la convolution, le filtre à appliquer, le nombre de threads à utiliser et enfin l'image dans laquelle écrire le résultat. Vous êtes libres d'utiliser la syntaxe de votre choix pour le passage des arguments à votre programme.

Le code pour lire et écrire des images au format PPM couleur 24-bit vous est fourni (`ppm.h`, `ppm.c`, `ppm_example.c`), de même qu'une image de base (`image.ppm`).

Votre implémentation doit toutefois respecter les points suivants :

- Nombre de threads arbitraire : le nombre de threads créés pour effectuer le calcul doit être arbitraire et donc spécifié par l'utilisateur sur la ligne de commande.
- Taille de filtre arbitraire : le code doit être suffisamment générique pour ne pas dépendre d'une taille de filtre particulière (taille minimum : 3). A noter que la taille d'un filtre est toujours impaire.
- Le filtre à appliquer doit être sélectionnable en ligne de commande. Les filtres suivants doivent être implémentés :

- identity (3x3) :

0	0	0
0	1	0
0	0	0

- sharpen (3x3) :

0	-1	0
-1	5	-1
0	-1	0

- edge (3x3) :

-1	-1	-1
-1	8	-1
-1	-1	-1

- blur (3x3) :

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

- gauss (5x5) :

1/256	4/256	6/256	4/256	1/256
4/256	16/256	24/256	16/256	4/256
6/256	24/256	36/256	24/256	6/256
4/256	16/256	24/256	16/256	4/256
1/256	4/256	6/256	4/256	1/256

- unsharp (5x5) :

-1/256	-4/256	-6/256	-4/256	-1/256
-4/256	-16/256	-24/256	-16/256	-4/256
-6/256	-24/256	476/256	-24/256	-6/256
-4/256	-16/256	-24/256	-16/256	-4/256
-1/256	-4/256	-6/256	-4/256	-1/256

- Taille d'image arbitraire : tout comme pour la taille du filtre, la taille de l'image à laquelle appliquer le filtre doit pouvoir être de taille arbitraire.
- Valeurs aux bords de l'image. Vous êtes libres de choisir l'approche à adopter aux bords de l'image. Cependant, vous devrez clairement décrire et justifier l'approche utilisée dans votre rapport.
- Dans le cas d'images couleur, le filtre est appliqué séparément pour chaque composante couleur de l'image (R, G, B).
- Arguments du programme (ligne de commande). Le programme prendra au minimum 4 arguments : le nombre de threads à utiliser, l'image originale, le filtre à appliquer et enfin, l'image finale sauvegardée après application du filtre à l'image originale.
- Si l'utilisateur ne spécifie pas le bon nombre d'arguments, le programme doit afficher la syntaxe du programme et se terminer.
- Le programme doit afficher le temps mis pour calculer la convolution sur l'image.
- Assurez-vous de désallouer toute mémoire dynamiquement allouée.
- Vérifiez que les fonctions que vous utilisez n'échouent pas.

Le but étant de mesurer le gain obtenu par l'utilisation de threads, vous devrez développer deux implémentations : une version multi-threadée ainsi qu'une version séquentielle classique.

Commencez par programmer la version classique afin de vérifier le bon fonctionnement du filtrage.

Rapport

Le rapport que vous rédigerez devra au moins contenir les trois sections suivantes (le choix des titres de sections est libre) :

- Courte description générale de l'état du projet rendu. Si un problème a été détecté, décrivez-le ici et expliquez quelle aurait été votre démarche pour tenter de le corriger.
- Approche concurrente utilisée : vous expliquerez ici votre implémentation, en justifiant vos choix.
- Comparaison des performance entre exécution séquentielle et version multi-threadée :
 - Exécutez votre programme dans des situations variées et analysez les résultats obtenus. En particulier, la version multi-threadée est-elle plus rapide que la version séquentielle ? Si oui, avec combien de threads ? Le gain est-il linéaire ? Etc.
 - Vous pouvez déterminer le nombre de coeurs disponibles sur votre machine en exécutant la commande : `cat /proc/cpuinfo | grep processor`

Indications

Lecture/écriture d'images au format PPM

Sur CyberLearn, se trouve le code permettant de lire/écrire des fichiers PPM (`ppm.tar.gz`), une image de petite taille au format PPM (`image.ppm`) et une très grande image au format jpeg (`image_xxl.jpg`).

Le programme `display` (du package ImageMagick) permet d'afficher une image au format PPM.

Dans le même package, se trouve le programme `convert` qui permet de convertir n'importe quel type d'image au format PPM. Par exemple, pour convertir `image.jpg` au format PPM :

```
convert -compress none image.jpg image.ppm
```

Attention toutefois à vérifier que vous avez suffisamment d'espace disque pour stocker l'image convertie, car le format PPM est probablement le plus inefficace qui soit en terme d'espace disque. A titre d'exemple, l'image de 33 millions de pixels `image_xxl.jpg` a une taille de 10.1 MB en format jpeg, alors que celle-ci fait 357 MB en PPM !

Ne convertissez donc pas d'images de grande taille dans votre espace privé, sinon celui-ci sera instantanément saturé ! Si vous deviez le faire, utilisez le répertoire `/tmp`, car il s'agit d'un répertoire local à la machine avec plusieurs GB d'espace disque disponible.

Mesures de timing

Les mesures de timing peuvent être effectuées avec la fonction `clock_gettime` de la librairie `<time.h>` comme montré ci-dessous :

```
struct timespec start, finish;
clock_gettime(CLOCK_MONOTONIC, &start);

... // code à mesurer

clock_gettime(CLOCK_MONOTONIC, &finish);
double elapsed_ms = 1000 * (finish.tv_sec - start.tv_sec);
elapsed_ms += (finish.tv_nsec - start.tv_nsec) / 1000000.0;
```

Le code ci-dessus requiert la librairie `rt`. Pour ce faire, passez `-lrt` au compilateur `gcc` au moment où vous générez votre exécutable final.

Travail à rendre

Rendez l'intégralité du code (avec `makefile`) ainsi que le rapport dans une archive appelée **<groupeN>.tar.gz** où N est votre numéro de groupe. Vérifiez au préalable que votre code compile sans warnings et sans erreurs !

Le document « `Consignes Travaux Pratiques.pdf` » disponible sur CyberLearn explicite de façon détaillée les modalités de rendu.