Laboratoire 1

Pipeline de modification d'images

Écrit par Gabriel-Andrew Pollo-Guilbert

Pour nicolas-2.lauzon@polymtl.ca

Pour william.martineau@polymtl.ca

Le pipelinage est une méthode de traitement de données où chaque étape du traitement alimente la prochaine. Celle-ci est souvent implémenté de sorte que chacune des étapes est exécutée en parallèle, comme dans un pipeline d'instructions d'un processeur ou dans un pipeline graphique.

Ce laboratoire a pour but de vous familiariser avec la programmation parallèle de base par le biais de l'implémentation d'un pipeline de traitement d'images. Pour se faire, vous devez convertir un pipeline sériel en un pipeline parallèle à l'aide des librairies POSIX Threads (pthreads) et Threading Building Blocks (TBB) de Intel.

Chaque noeud d'exécution ne doit qu'exécuter qu'une seule étape du pipeline. Lorsque l'image est convertit, il l'envoit à la prochaine étape si nécessaire.

Code

- `source/main.c`
  - Contient le point d'entrée du programme qui traite les arguments en ligne de commande et démarre le pipeline de traitement d'images voulu.
- `source/image.c include/image.h`
  - Contiennent les structures et le code permettant la lecture/écriture d'images de format PNG.
- `source/filter.c include/filter.h`
  - Contiennent différentes fonctions permettant d'appliquer des filtres (modifications) à des images.
- `source/queue.c include/queue.h`
  - Contiennent une implémentation simple d'une file permettant la lecture/écriture par plusieurs noeuds d'exécution. Ces structures et fonctions sont **fortement** recommandé lors de l'implémentation du pipeline utiliant pthreads.
- `source/pipeline-serial.c`
  - Contient une implémentation sérielle de référence du pipeline.
- `source/pipeline-pthread.c` **(À COMPLÉTER)**
  - Contient l'implémentation parallèle demandée du pipeline à l'aide de pthreads.
- `source/pipeline-tbb.cpp` **(À COMPLÉTER)**
  - Contient l'implémentation parallèle demandée du pipeline à l'aide de TBB.

- `data/fetch.sh`
  - Contient un script pour télécharger les images de test.
- `data/check.sh`
  - Contient un script pour vérifier si les images produites sont identique aux images sérielles.

Spécifications

Les spécifications décrites dans cette section diffèrent d'une équipe à une autre.

Le pipeline à implémenter doit contenir les étages suivantes:

1. lire une image
2. tripler la taille de l'image
3. appliquer un filtre de sobel, utilisez 3 POSIX threads
4. enregistrer l'image

De plus, les contraintes suivantes sont imposées:

- L'implémentation avec TBB doit être fait avec des lambdas et non des classes.
- L'implémentation POSIX threads doit avoir un noeud par étape, sauf à indication contraire.
- La compilation ne doit pas lancer d'avertissements.
- Le programme ne doit pas avoir de fuite de mémoire durant son exécution.

Compilation

Pour compiler l'application, il est recommandé de créer un dossier `build/` à la racine du projet afin de bien séparer les fichiers générés.

```
$ mkdir build && cd build
```

Ensuite, on configure le projet avec `cmake`. Celui-ci peut donc être compilé avec `make`.

```
$ cmake ..
$ make
```

Il n'est pas nécessaire de re-exécuter toutes les commandes ci-dessus pour recompiler le binaire, seulement la dernière. Vous pouvez exécuter `./pipeline --help` pour voir les options du programme.

Données et Résultats

Le programme lit les images dans un dossier ayant les noms `0000.png`, `0001.png`, `0002.png`, etc. Il va

ensuite enregistrer les images résultantes dans le même dossier avec un préfixe comme `pthread-0000.png`, `pthread-0001.png`, etc.

Le script `data/fetch.sh` est fournit afin d'obtenir les images de bases. Celui-ci va télécharger plusieurs images PNGs du film à license libre Big Buck Bunny pour un total d'environ 200 MB d'images PNGs.

Avec les images modifiées, le dossier peut faire plus de 1 GB. Il n'est donc pas recommandé de travailler sur le laboratoire directement sur votre disque réseau de l'école. Vous pouvez utiliser `/home/tmp` sur les ordinateurs du laboratoire. Cela dit, ce dossier est supprimé à chaque 24 heures, alors vous devez copier vos fichiers sans les images sur votre disque réseau à la fin.

Commandes

Le `Makefile` généré par `cmake` contient les commandes spéciales ci-dessous.

- `make format`
  - Utilise `clang-format` pour formatter le code source.
- `make remise`
  - Crée une archive ZIP contenant les fichiers pour la remise.
- `make run-serial`
  - Exécute le pipeline sériel avec les données dans le dossier `data/` et en mesurant le temps écoulé.
- `make run-pthread`
  - Exécute le pipeline utilisant pthreads avec les données dans le dossier `data/` et en mesurant le temps écoulé.
- `make run-tbb`
  - Exécute le pipeline utilisant TBB avec les données dans le dossier `data/` et en mesurant le temps écoulé.
- `make run-all`
  - Exécute les 3 pipelines ci-dessus.

Exemple

Dans l'exemple ci-dessous, on télécharge les images de test, on exécute les 3 algorithmes et on vérifie si toutes les images créées sont identiques.

```
$ ./data/fetch.sh
$ mkdir build && cd build && cmake .. && make run-all
$ ../data/check.sh
```