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

Alberto Dassatti, Bruno
Da Rocha Carvalho

Laboratoire de High Performance Coding

semestre printemps 2023

Laboratoire 4 : SIMD

Temps à disposition: 6 périodes (3 séances de laboratoire)

1 Objectifs de ce laboratoire

Dans ce laboratoire, vous serez amené(e)s à analyser et à écrire du code en utilisant les instructions SIMD. Vous trouverez toutes les informations nécessaires concernant les instructions SIMD pour les processeurs Intel à l'adresse suivante : https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html

2 Analyse et amélioration

Dans cette première partie du laboratoire, vous devrez analyser et modifier le code présent dans `image_segmentation/src/k-means.c` et `image_segmentation/include/k-means.h` en y intégrant les optimisations SIMD que vous jugerez pertinentes.

Dans les sections 2.1 et 2.2, vous trouverez des explications sur les algorithmes mis en place et leurs objectifs.

2.1 Segmentation d'image

La segmentation d'image est un processus consistant à diviser une image en plusieurs parties ou segments qui représentent des zones distinctes de l'image. Cette technique est largement utilisée en traitement d'images et en vision par ordinateur pour extraire des informations importantes à partir d'images.

Cette méthode est largement utilisée dans de nombreux domaines, tels que la reconnaissance de forme, la surveillance de la circulation, la médecine, l'analyse de données géologiques, l'astronomie, etc. Elle est également utilisée dans des applications pratiques telles que la reconnaissance de visages et la détection d'objets dans les images de caméras de sécurité.

Vous trouverez ci-dessous un exemple d'exécution du programme de segmentation d'image fourni, avec deux clusters :

Figure 1: Image de base

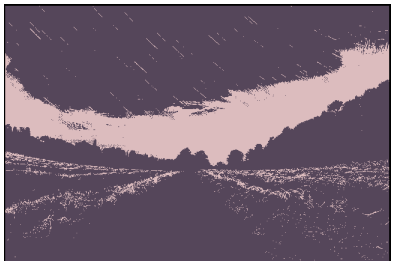

Figure 2: Image ségmentée

2.2 K-Means et K-means++

K-means est un algorithme de clustering utilisé pour regrouper des données en k groupes distincts. L'algorithme utilise une technique itérative pour déplacer les centres des clusters jusqu'à ce que la somme des carrés des distances entre les points et leur centre de cluster le plus proche soit minimisée. L'algorithme k-means est simple et rapide, mais il est sensible à l'initialisation des centres de clusters et peut être piégé dans des minima locaux.

Pour améliorer les performances de k-means, une technique d'initialisation des centres de cluster appelée k-means++ a été proposée. K-means++ utilise une méthode de sélection des centres de cluster qui garantit que les centres de cluster initiaux sont répartis uniformément dans l'espace de données. En conséquence, les centres de cluster initiaux sont plus susceptibles de se rapprocher de la solution globale optimale. K-means++ est une technique simple mais puissante pour améliorer les performances de l'algorithme k-means.

Dans le cadre de ce laboratoire, K-means est utilisé pour segmenter des images en k groupes, en se basant sur la teinte de chaque pixel. Ainsi, chaque pixel de l'image est assigné au cluster ayant la couleur RGB la plus proche. Cette méthode permet de créer des groupes d'images similaires en termes de couleur.

2.3 Code fournit

Le code qui vous est proposé se compose comme suit :

- Un fichier main.c qui charge l'image, appelle la fonction de segmentation et sauvegarde l'image modifiée dans un fichier de résultat.
- Une structures de données `pixel` qui contient les informations RGB d'un pixel.

```
typedef struct {
        int r;
        int g;
        int b;
} pixel;
```

- Une fonction de distance, qui retourne la distance euclidienne entre deux pixels RGB.

```
float distance(pixel p1, pixel p2);
```

- Les deux fonctions de k-means et de k-means++.

```
void kmeans_pp(pixel *image, int width, int height, int num_clusters, pixel *centers);

void kmeans(pixel *image, int width, int height, int num_clusters);
```

Bien sûr, tout ce code n'est qu'une proposition. Il est là pour vous assurer du résultat à obtenir et pour vous permettre de faire vos tests facilement. Vous êtes libre de le modifier à votre guise, tant que le fichier main.c reste intact.

2.4 Contraintes et conseils

- Vous ne devez pas modifier le fichier `main.c`.
- Si vous souhaitez utiliser et tester d'autres images (plus grandes ou plus complexes), elles doivent être au format bmp.
- L'utilisation de l'exécutable se fait comme suit : `./segmentation <img_src.bmp> <nb_clusters> <img_dest.bmp>`.
- Vous devrez modifier le Makefile pour pouvoir gérer les instructions SIMD.

- Vous êtes autorisé à ajouter, supprimer ou modifier tout ce que vous voulez dans le code de `k-means.c/.h`.
- La structure `pixel` doit toujours exister car elle est utilisée pour charger l'image de base.

3 Développement et réflexion sur l'utilisation SIMD

Dans le cadre des précédents laboratoires, vous avez déjà développé et implémenté l'algorithme A* sans utiliser de queue de priorité. Bien que cela soit possible, cette méthode n'est pas la plus efficace en termes de temps de traitement. Il est donc intéressant d'avoir une implémentation de l'algorithme A* qui utilise une queue de priorité pour optimiser la recherche.

Une queue de priorité permet de stocker des éléments en fonction de leur priorité et de les récupérer dans l'ordre de priorité croissant ou décroissant. Dans le cas de l'algorithme A*, la priorité est définie par la somme de deux facteurs : le coût du chemin déjà parcouru et l'estimation du coût restant pour atteindre la destination. En utilisant une queue de priorité, les nœuds sont récupérés dans l'ordre de priorité croissant, ce qui permet à l'algorithme de toujours explorer les nœuds les plus prometteurs en premier.

Dans ce laboratoire, vous devez développer une queue de priorité générique, donc qui recevra comme données des `void*`. De ce fait votre queue de priorité n'est pas seulement adaptée à l'utilisation de l'algorithme A*, mais à n'importe quel type de données.

Tous le code qui vous est proposé peut-être changé a votre guise. À la seule exception du fichiers main.c

3.1 Travail demandé

Vous devez développer les fonctionnalités suivantes pour votre queue de priorité :

- Ajout d'éléments (push) : la queue de priorité doit pouvoir insérer des éléments en tenant compte de leur priorité. Les éléments doivent être triés dans l'ordre croissant ou décroissant de leur priorité, selon la politique de la queue de priorité.
- Récupération du dernier élément de la queue (avec la priorité la plus élevée) : la queue de priorité doit être capable de renvoyer l'élément de plus haute priorité, c'est-à-dire le dernier élément de la pile.
- Ordonnement des données : nous imaginons avoir un envrionnement qui nous demande l'ajout et la récupération des données le plus rapidement possible, mais que nous avons des temps morts dans lequelle nous devons ordonné notre queue.
- Destruction correct de la queue.

Dont voici les prototypes proposé que vous retrouverez dans le fichier `priority_queue/include/priority_queue.h` :

```
void push (priority_queue_t *pq, void* data);

void prioritize (priority_queue_t *pq);

void *pop (priority_queue_t *pq);

void destroy(priority_queue_t *pq);
```

La fonction `priority_queue_create` permet de créer une queue de priorité en fournissant une fonction de comparaison pour les données. Cette fonction de comparaison peut être réalisée comme

vous le souhaitez, mais elle doit simplement retourner un entier négatif, nul ou positif pour pouvoir ordonner les données.

```
priority_queue_t *priority_queue_create(int (*compare_priority)(const void *, const void*));
```

Les fichiers `priority_queue/src/priority_queue.c` et `priority_queue/include/priority_queue.h` contiennent une proposition de squelette de code pour votre queue de priorité. Vous êtes libre de développer le code comme vous le souhaitez, mais il est impératif de respecter les contraintes suivantes :

3.2 Contraintes

- Votre queue de priorité doit avoir une taille dynamique, ce qui signifie que l'on doit pouvoir y ajouter autant d'éléments que souhaité (tant que la mémoire le permet bien sûr !).
- Vous devez respecter les spécifications énoncées ci-dessus, notamment permettre l'ajout d'éléments en fonction de leur priorité et la récupération du dernier élément avec la priorité la plus haute.
- Le programme main ne doit pas être modifié et doit pouvoir se lancer avec les appels présent.

4 Travail à rendre

4.1 Partie 1

Pour cette partie, il vous sera demandé de rendre votre code modifié, compilable et accompagné d'un rapport d'analyse. Vous devrez expliquer en détail les modifications que vous avez apportées au code initial, en justifiant votre choix et en présentant les avantages obtenus grâce à ces changements. Par exemple, vous pourrez expliquer comment vous avez optimisé une opération pour traiter davantage de données simultanément, ou démontrer en quoi vos modifications améliorent la performance globale du code.

4.2 Partie 2

Dans cette partie, vous devrez rendre votre code fonctionnel, avec une démonstration de son fonctionnement dans le `main`. Vous devrez également documenter votre code de manière claire et explicite, en expliquant le fonctionnement de chaque fonction et en commentant les parties les plus importantes du code. Enfin, vous devrez expliquer dans votre rapport comment les fonctions SIMD ont été utiles pour votre implémentation, en présentant les choix que vous avez faits et les résultats obtenus grâce à ces fonctions.