HPC - Lab 04

Aubry Mangold
ISCS, HEIG-VD
aubry.mangold@heig-vd.ch

Introduction

Topologie du système

Traitement d'image avec K-Means

Le programme de segmentation d'image fourni implémente les algorithmes de clustering K-Means et K-Means++ pour segmenter les pixels d'une image selon leur tente. Cette section présente l'optimisation manuelle du programme, l'introduction de SIMD pour améliorer le temps de traitement ainsi que les résultats obtenus.

Tests

Un système de test automatisé est mis en place afin de valider le comportement du code après chaque optimisation. La comparaison entre le jeu d'images produites par le programme original et le jeu d'images produites par le programme optimisé est effectuée avec `ImageMagick` pour des résolutions 256x256, 640x426 et 1024x1024 pixels. Le nombre de bins est également testé avec des valeurs échelonnées dans la plage $[2^0, 2^8]$.

Benchmark

Le temps d'exécution est mesuré avec `hyperfine` après un lot de changement condensé. Les paramètres de test sont l'image originale de 640x426 pixels produite par le programme et un nombre de bins allant de 2^0 à 2^8. `hyperfine` est configuré pour utiliser un coeur préchauffé et effectuer les mesures 5 fois. Les données récoltées sont présentées dans Figure 1.

Analyse préliminaire

Une rapide analyse préliminaire est menée afin d'identifier les sections du programme qui sont des fonctions candidates pour de l'optimisation. Dans le cadre de ce laboratoire, la métrique de choix est le temps d'exécution du programme. L'outil `gprof` est utilisé pour profiler le programme. Les résultats présentés dans Listing 1 montrent sans surprise que le programme passe beaucoup de temps utilisateur dans les fonctions `distance`, `kmeans` et `kmeans_pp`. Ces dernières sont donc des cibles de choix.

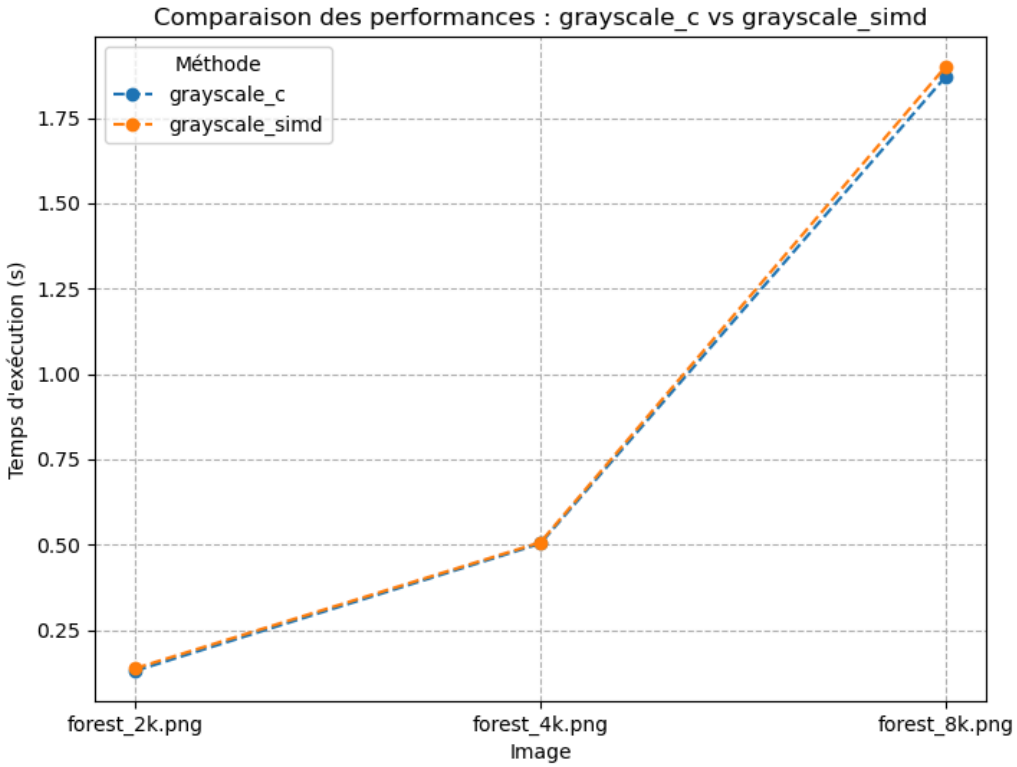

Figure 1: Temps d'exécution par nombre de bins du programme KMeans

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
50.00	0.02	0.02	2181120	0.00	0.00	distance
25.00	0.03	0.01	1	10.00	40.00	kmeans
25.00	0.04	0.01	1	10.00	20.00	kmeans_pp
0.00	0.04	0.00	846717	0.00	0.00	stbiw__paeth
0.00	0.04	0.00	783702	0.00	0.00	stbi_write_jpg_core
0.00	0.04	0.00	624578	0.00	0.00	stbi__zhuffman_decode

Listing 1: Rapport `gprof` du programme KMeans

Re-écriture initiale

Plusieurs itération de re-écriture du programme ont été effectuées afin d'apporter une première ronde d'optimisations et de préparer le programme à la vectorisation.

Les modifications majeures apportées à chaque étape sont :

- **Version 1** :
  - Optimisation du calcul de la distance par la distance carrée.
  - Suppression des allocations et copies (remplacées par de l'arithmétique de pointeurs).

  - Correction du calcul des poids du `kmeans_pp`.
  - Précalcul de variables partagées.
  - Suppression (*flattening*) les boucles imbriquées.
- **Version 2** :
  - Implementation d'un PRNG rapide.
  - Calcul de la distance sur les entiers.
  - Ajustement de la taille des blocs de données traitées par les boucles (cache blocking).
  - Déroulement des boucles par facteur de 4 avec des directives pragmas.
  - Implémentation d'un retour rapide en cas de minima sur le calcul de la distance.
  - Optimisation de calcul flottant en utilisant les multiplications par réciproque à la place des divisions.
  - Suppression des appels à `memcpy` au profit d'écriture directe dans le pointeur original.
- **Version 3** :
  - Précalcul des sommes des carrés.
  - Remplacement de la formule de comparaison de la distance pour une formule basée sur le produit scalaire.
  - Déroulement des boucles restantes
  - Allocation alignée sur la taille du cache.
  - Optimisation des boucles pour le traitement de multiples de 4 pixels.

On observe dans Figure 1 que les différentes itérations d'optimisation du programme ont permis de réduire considérablement le temps d'exécution sans l'implémentation de SIMD. La version 3 est particulièrement rapide, notamment grâce au précalcul et au déroulement des boucles qui suppriment des appels de fonctions couteux.

Optimisation SIMD

Le programme est modifié pour utiliser des instructions SIMD. Le but de cette optimisation est d'accélérer le calcul de la distance entre les pixels et les bins. Les données (centres RGB) sont transposées au format struct-of-arrays et paddés a des multiples de 8 bytes (taille maximum AVX2). Le calcul de la distance est ensuite effectué en chargeant puis convertissant 8 centres à la fois avant d'appliquer une différence carrées vectorialisée. Le masquage dynamique est utilisé pour mettre à jour le cluster dans avoir besoin de brancher. La recherche du minima est ensuite effectuée avec des instruction SIMD spécialisées.

On observe dans Figure 1 que l'implémentation de SIMD entraine une surcout pour des petites tailles de cluster et qu'il faut donc que la taille de l'image soit suffisamment grande pour que le surcout d'initialisation soit amorti par le gain de performance.

Options de compilation

- `-fno-inline` est enlevé. L'inlining est une un échange ou l'on accepte de sacrifier la taille pour gagner en vitesse d'exécution en réduisant la quantité d'appels de fonctions. Dans notre cas, la plupart des fonctions internes à l'unité ont déjà été inlinées manuellement dans les fonctions `kmeans` et `kmeans_pp`. Seuls quelques appels restent à être inlinés par le compilateur. Le temps d'exécution s'améliore de façon surprenante pour une faible quantité de bins mais est similaire au résultat de la version SIMD pour une quantité importante de bins.
- `-Ofast` est ajouté pour un test. On observe que le comportement du programme reste exact dans notre cas d'utilisation malgré l'imprécision des calculs mathématiques engendrés. Néanmoins,

l'option est retirée par souci de portabilité (il se peut que les calculs soient faussés sur d'autres machines).

Résultat

L'optimisation générale du programme est très efficace. L'amélioration du temps d'exécution lors de l'optimisation manuelle démontre qu'il ne faut pas obligatoirement commencer par vectorialiser le code pour gagner en performance. En effet, des optimisations (algorithmiques ou syntaxiques) peuvent être appliquées pour réduire le temps d'exécution de manière significative.

Les paramètres choisis pour le benchmarking étaient pratiques car ils permettaient un temps d'exécution viable sur la machine de développement, mais rend difficile de discerner quelle fonction est la plus efficace, surtout lorsque les optimisations ne font que perdre que quelques millisecondes.

La vectorisation permet de tirer avantage des architectures modernes mais introduit plus de complexité ainsi qu'un surcoût de temps de traitement si les données ne sont pas de volume suffisant. Le programme doit aussi pouvoir être adapté pour pouvoir être vectorisé correctement avec SIMD (dans notre cas en ramenant nos opérations mathématiques sur des entiers). Par ailleurs, l'implémentation de la vectorisation hybride avec des entiers et flottant a été tentée sans succès.

Finalement, il est difficile de juger de l'impact réel des options de compilations sur le temps d'exécution. Si certaines options (tel que l'inlining) sont un apport considérable dans certains cas, d'autres options n'ont qu'un apport infime. Il faudrait que les contraintes en temps d'exécution soient très strictes pour justifier de s'en soucier plus.

Traitement d'image en nuance de gris avec SIMD

Le traitement d'image choisi est la conversion en nuance de gris. L'algorithme de conversion calcule une somme pondérée des trois canaux de couleur pour chaque pixel. Ce type de calcul est particulièrement adapté à la vectorisation car il peut être effectué en parallèle sur plusieurs pixels à l'aide d'instructions SIMD. Aucune option d'optimisation n'est utilisée dans ce programme afin de se concentrer sur les effets de l'optimisation SIMD.

Benchmark

Le benchmarking est effectué sur une unique image de référence PNG en 2k, 4k et 8k. Le temps d'exécution est mesuré avec `hyperfine` pour le programme original et le programme optimisé avec SIMD. Les résultats sont présentés dans Figure 2.

Programmes

Le squelette du programme C (`main.c`, `image.c` et les fichiers de build) est copié du programme SIMD fourni pour le laboratoire. Le programme applique la formule de conversion NTSC.

Le programme SIMD calcule la valeur en nuance de gris de 8 pixels à la fois. L'algorithme est le suivant:

- Boucle par blocs de 8
  1. Chargement de 32 octets
  2. Extraction des moitiés basses et hautes des 256 bits
  3. Pour chaque moitié :
     1. Convertir 8 bytes en 8×32 bits
     1. Multiplier par les constantes et sommer

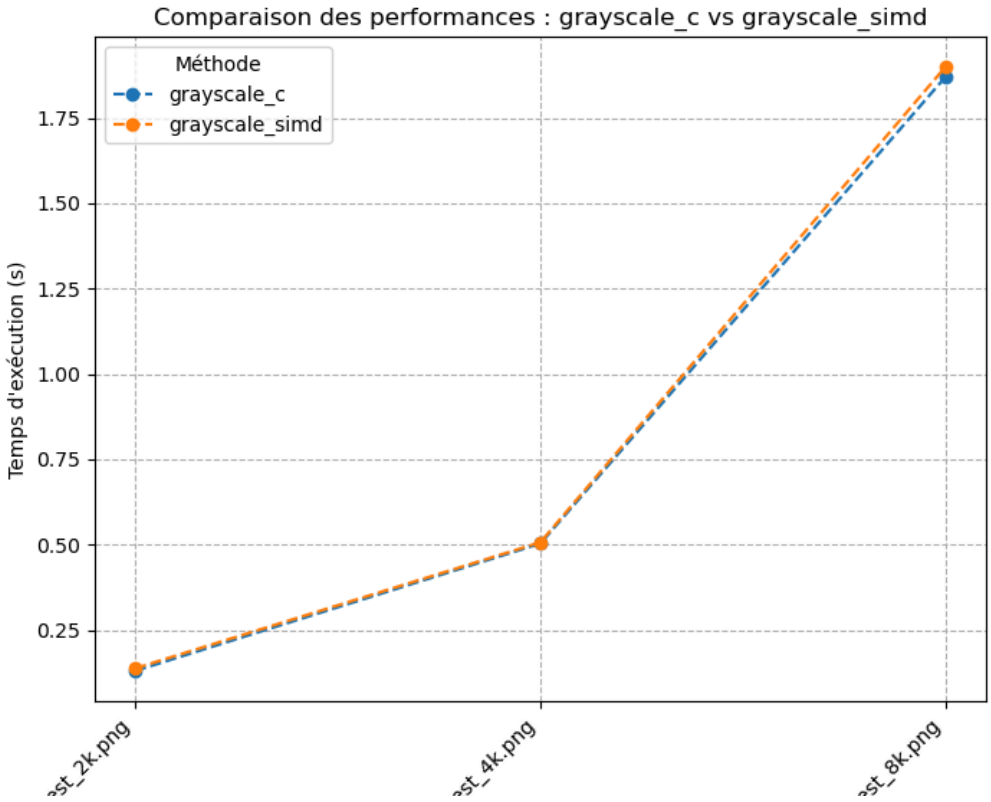

Figure 2: Temps d'exécution par image et par version

4. Regrouper les résultats en 8 bytes
5. Ecrire les 8 bytes

Résultats

Le résultat n'est pas celui attendu, a savoir la baisse du temps d'exécution du programme. Au contraire, les résultats du benchmarking montrent que la version SIMD est presque un peu moins performante. Cela est peut-être du aux facteurs suivants :

- L'overhead introduit par la vectorisation n'est pas compensé sur le jeu de données de test (peu probable).
- Le programme est memory-bound sur un ordinateur moderne (très probable).
- L'algorithme de traitement d'image n'effectue pas assez d'opérations pour bien démontrer la différence.

Optimisation SIMD sur le programme DTMF

L'implémentation de l'algorithme à base de FFT a été choisi pour l'optimisation car Goertzel n'est pas un bon candidat puisqu'il se base sur un état interne à un échantillon.

Optimisation SIMD de la version FFT

Benchmark

Le benchmarking est effectué sur des fichiers contenant respectivement 10, 100 et 1000 fois l'alphabet (afin de ne pas se faire piéger par un jeu de données de test trop peu volumineux).

Conclusion

L'expérience effectuée avec l'algorithme de nuance de gris prouve que l'algorithme utilisé ainsi que les limites du matériel sont des facteurs clés à prendre en compte lors de l'optimisation par SIMD; il est possible que la vectorisation d'un programme ne soit simplement pas la meilleure stratégie d'optimisation.