Laboratoire 4: SIMD

Département: **TIC**

Unité d'enseignement: **HPC**

Auteur(s):

- **CECCHET Costantino**

Professeur:

- **DASSATTI Alberto**

Assistant:

- **DA ROCHA CARVALHO Bruno**

Date:

- **10/03/2024**

[Page de mise en page, laissée vide par intention]

Introduction

Dans ce laboratoire il faut optimiser du code en utilisant des operations SIMD

Optimisation du code

fonction 1 distance

Cette fonction calcule la distance euclidean entre deux pixels. Au lieu d'utiliser les coordonnées, on utilise la valeur RGB pour évaluer la distance.

Pour optimiser cette fonction on va charger les valeurs RGB des deux pixels dans des registres SIMD, puis on va soustraire les valeurs des deux pixels, on va ensuite multiplier les valeurs obtenues et les additionner pour obtenir la distance euclidean.

Grâce aux operations SIMD on peut effectuer ces opérations en parallèle sur les 4 composantes RGB(3 composants + 1 composant de padding).

fonction 2 kmeans_pp

Cette fonction implémente l'algorithme kmeans++ pour initialiser les centres des clusters.

Cette fonction peux être optimisée en utilisant les operations SIMD pour calculer la distance euclidean entre les pixels et les centres des clusters sans fair appel a la fonction distance.

De plus les allocations dynamique peuvent être évitées en sauvegardant les valeurs RGB des pixels dans les registres SIMD, ici aussi on a un bit de padding pour avoir 4 valeurs dans un registre SIMD, ce dernier pourrait servir pour des images en RGBA.

fonction 3 kmeans

Cette fonction implémente l'algorithme kmeans pour regrouper les pixels en clusters.

Elle utilise la fonction kmens_pp pour initialiser les centres des clusters, puis elle assigne chaque pixel au cluster le plus proche.

Notre optimisation se base sur des operation simd pour l'assignation des pixels aux clusters, en effet on va calculer la distance euclidean entre les pixels et les centres des clusters en utilisant les operations SIMD.

Puis on optimisera la mise a jour des centres des clusters en utilisant les operations SIMD.

Et pour finir on updatera l'image avec les clusters en utilisant les operations SIMD.

Optimisation du filtre de sobel

Pour cette partie nous avons du optimiser notre code du labo précédent en utilisant les operations SIMD.

Ici il faut se rendre compte que on peux que optimiser la code utilisant les tableaux car avec les liste chaînées les données peuvent ne pas être contiguës en mémoire.

Pour la partie optimisation on va utiliser les operations SIMD pour les calculs utilisant les kernels car on peut effectuer des operations sur les 4 pixels en parallèle.

pour lancer le code optimisé il faut utiliser la commande suivante depuis le dossier code:

```
$ make
$ ./edge_detection <image_src> <image_dst> <mode>
```

nous avons repris le code non optimizé pour le comparer avec le code optimizé.

grace à un script python on a pu comparer les temps d'execution des deux versions du code.

Voici les résultats obtenus:

```
Image: ../img/nyc.png
Time statistics------- for edge_detection-------
Mean time: 0.08925
Median time: 0.08718
Standard deviation: 0.00388
Time statistics------- for lab01-------
Mean time: 0.09318
Median time: 0.09230
Standard deviation: 0.00572
Image: ../img/half-life.png
Time statistics------- for edge_detection-------
Mean time: 0.28491
Median time: 0.26933
Standard deviation: 0.02693
Time statistics------- for lab01-------
Mean time: 0.26179
Median time: 0.24894
Standard deviation: 0.02444
Image: ../img/medalion.png
Time statistics------- for edge_detection-------
Mean time: 0.08651
Median time: 0.08757
Standard deviation: 0.00548
Time statistics------- for lab01-------
Mean time: 0.07967
Median time: 0.07651
Standard deviation: 0.01020
Image: ../img/sample_640_2.png
Time statistics------- for edge_detection-------
Mean time: 0.02528
Median time: 0.02529
Standard deviation: 0.00081
Time statistics------- for lab01-------
Mean time: 0.02828
```

```
Median time: 0.02592
Standard deviation: 0.01591
```

On peut voir que le code optimisé est pas toujours plus rapide que le code non optimisé, ce dernier etant compilé avec les flags -O3 il utilise surment des operations SIMD pour optimiser le code.

il est donc probable que le compilateur ait déjà optimisé le code non optimisé avec des operations SIMD.

Conclusion

Dans ce laboratoire on a pu optimiser du code en utilisant des operations SIMD, on a pu voir que le compilateur peut déjà optimiser le code en utilisant des operations SIMD.

Dans notre cas nous remarquons que dans certains cas le travail manuel n'apporte pas de gain de performance, mais il est toujours bon de savoir comment optimiser du code en utilisant des operations SIMD. ##
Environnement d'execution

Le système décrit dispose d'un processeur Intel Core i7-8550U avec 8 threads, répartis sur 4 cœurs physiques. La fréquence du processeur est de 1,80 GHz avec une fréquence mesurée de 1432,548 MHz.

Voici plus ample information sur le processeur:

```
$ cat /proc/cpuinfo
processor   : 0
vendor_id   : GenuineIntel
cpu family  : 6
model       : 142
model name  : Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz
stepping    : 10
microcode   : 0xf4
cpu MHz     : 1432.548
cache size  : 8192 KB
physical id : 0
siblings    : 8
core id     : 0
cpu cores   : 4
apicid      : 0
initial apicid  : 0
fpu     : yes
fpu_exception   : yes
cpuid level : 22
wp      : yes
flags       : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflus
vmx flags   : vnmi preemption_timer invvpid ept_x_only ept_ad ept_1gb flexpriority tsc_offse
bugs        : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs itlb_mult
bogomips    : 3999.93
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
```