HEIG VD | REDS Institut Reconfigurable and Embedded Digital Systems

Programmation assembleur (ASM)

Prof. Daniel Rossier
Assistants : Bruno Da Rocha Carvalho, Thomas Rieder

Instructions SIMD x86 SSE

lab08 (07.06.2023)

Objectifs de laboratoire

L'objectif de ce laboratoire est de revoir et d'appliquer la convention ABI **x86 CDECL**. Il portera aussi sur l'utilisation d'instructions vectorisées (SIMD) **x86** afin d'accélérer des algorithmes de calcul donnés.

Échéance

- Le laboratoire doit être rendu à la fin des deux périodes de laboratoire

Validation du laboratoire

Un script de rendu vous est fourni afin de packager les fichiers sources modifiés sans inclure tous les fichiers générés (code objet, exécutables, etc.). Ce script s'appelle ***rendu.sh*** et sera fourni pour tous les laboratoires.

```
reds@reds2022:~/asm/asm_lab08$ ./rendu.sh
```

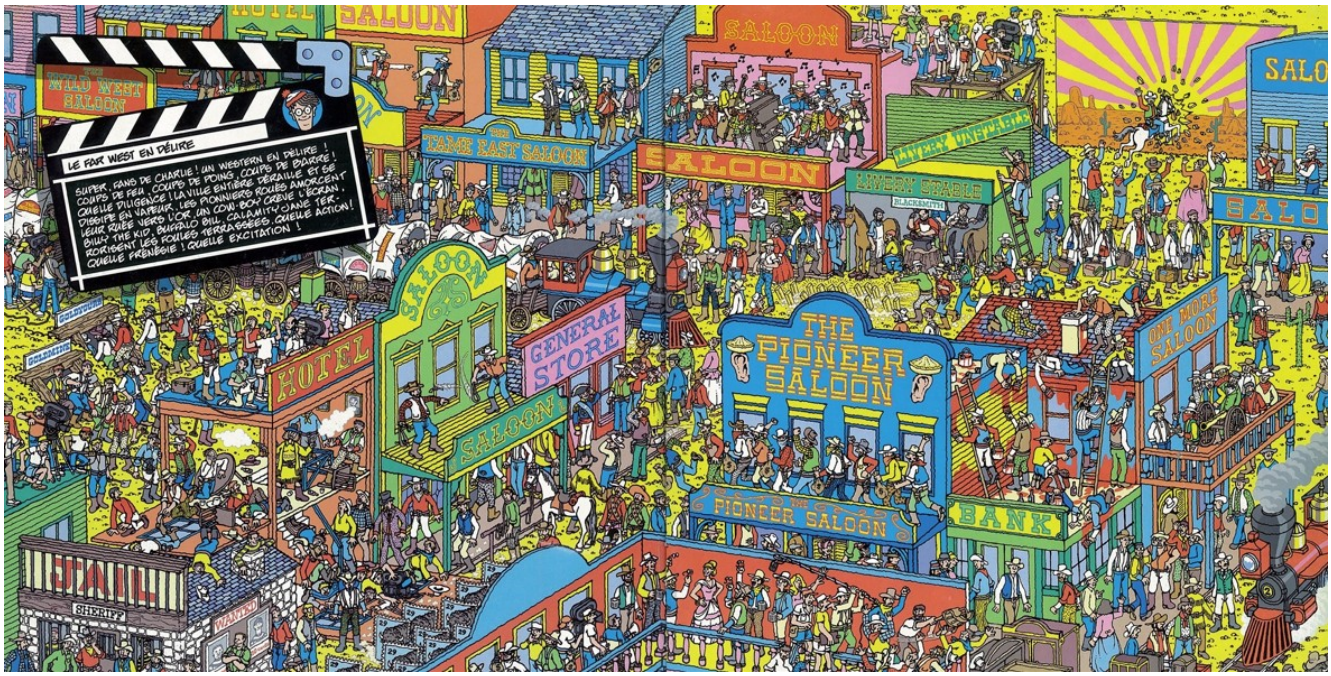
Cette commande va générer un fichier *asm_lab08_rendu.tar.gz* et c'est cette archive qui sera à remettre sur Cyberlearn. Il est demandé de toujours préparer l'archive du laboratoire grâce au script de rendu fourni.

Récupération des sources & mise en place de l'environnement

```
reds@reds2022:~$ retrieve_lab asm lab08
```

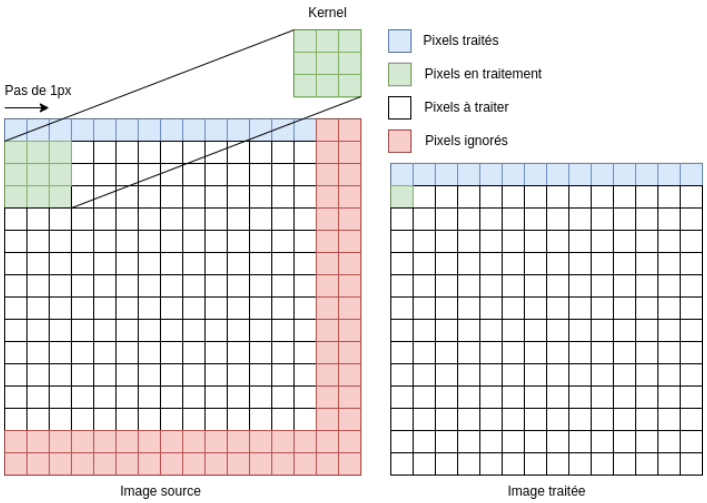
Étape 1 – Accélération de détection d'un motif dans une image

Dans ce laboratoire vous n'aurez qu'une fonction à écrire afin d'accélérer la détection d'un motif dans une image à l'aide d'instructions SIMD vues en cours.



Distance entre un motif et portion de l'image

Pour trouver un motif dans une image, il est nécessaire de parcourir celle-ci, comme illustré ici avec un motif 3x3 pixels.



Pour chaque position du motif dans l'image, indexée par le pixel en haut à gauche du motif, nous allons calculer une "distance" entre le motif et les pixels de l'image à l'endroit où se trouve le motif. Ce

processus est similaire à l'application d'un noyau de convolution dans le domaine du traitement d'image, à la différence que les pixels sont indexés à partir du premier pixel en haut à gauche du motif, ce qui rend l'indexation plus simple pour des motifs de tailles arbitraires (par exemple, plus grands que 3x3).

$$distance\left(\begin{bmatrix} m_1 & m_2 & m_3 \\ m_4 & m_5 & m_6 \\ m_7 & m_8 & m_9 \end{bmatrix}, \begin{bmatrix} p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 \\ p_7 & p_8 & p_9 \end{bmatrix}\right)[x,y] = \sqrt{(m_1-p_1)^2+(m_2-p_2)^2+(m_3-p_3)^2+\cdots+(m_8-p_8)^2+(m_9-p_9)^2}$$

Si le motif est exactement présent sur l'image, la valeur de la distance sera de 0. Plus il y aura de différences entre le motif et l'image, plus la distance sera grande. Une distance de 0 correspond à une correspondance exacte, une valeur légèrement supérieure à un match approximatif, et une valeur élevée à une grande différence entre le motif et la zone analysée de l'image.

Pour trouver la plus petite distance possible on peut omettre le calcul de la racine (car si $\sqrt{a} < \sqrt{b}$ alors $a < b$), elle n'est donc pas calculée dans ce laboratoire. Note : Il serait aussi possible d'implémenter la distance de Manhattan (somme des valeurs absolues des différences).

Voici un pseudo-code pour effectuer le calcul de distance :

```
POUR chaque ligne de l'image FAIRE
        POUR chaque pixel dans la ligne FAIRE

                POUR chaque ligne du motif FAIRE
                        POUR tous les groupes de 4 éléments de la ligne du motif FAIRE
                                Charger et étendre sur 32-bit les pixels correspondant de l'image
                                dans un registre xmm

                                Charger et étendre les pixels sur 32-bit du motif dans un autre
                                registre xmm

                                Soustraire (par groupe de 4 en SIMD) les valeurs
                                Effectuer (par groupe de 4 en SIMD) la mise au carré du résultat

                                Accumuler (4 à la fois) les carrés des différences dans un registre
                                xmm
                        FIN POUR
                FIN POUR

                Finir de sommer les valeurs accumulées

                RETOURNER la valeur de la somme finale

        FIN POUR
FIN POUR
```

INSTRUCTIONS SIMD

- « ***pmovzxbd*** » permet de charger 4 valeurs 8-bit et de les étendre à 32 bits avec des '0' (Zero Extend). Ces 4 valeurs 32-bit rempliront l'entier du registre 128-bit. L'utilisation de valeurs 32-bit évitera les overflow lors des calculs.
- « ***psubd*** »permet de soustraire 4 valeurs de 32 bits se trouvant dans des registres 128-bit. Voir le schéma dans le cours.
- « ***paddd*** »permet d'additionner 4 valeurs de 32 bits se trouvant dans des registres 128-bit. Voir le schéma dans le cours.
- « ***pmulld*** » permet de multiplier 4 valeurs de 32 bits se trouvant dans des registres 128-bit. Agit comme pour « *paddd* »
- « ***phaddd*** » permet de faire une addition horizontal de 2 valeurs de 32 bits, voir le schéma dans le cours.
- « ***movd*** » permet de charger les premiers 32 bits d'un registre « *xmm* » vers la mémoire ou un registre.

Vous pouvez trouver des informations supplémentaires sur ces instructions sur le lien suivant : https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html#techs=SSE_ALL

a) Vous devrez réaliser la fonction « ***asm_matrix_distance_simd*** » se trouvant dans le fichier « *x86_host/pattern_detection/asm_distance.S* ». Cette fonction retournera la somme des différences au carré, la racine pourrait être effectuée dans le code C, mais on peut s'en passer.

```
int asm_matrix_distance_simd(image_container *img,
                             image_container *kernel,
                             int x, int y)
```

L'argument «image_container» est une structure C telle que :

```
typedef struct {
    int width;          // mem+0
    int height;         // mem+4
    int comp;           // mem+8
    uint8_t *data;      // mem+12
} image_container;
```

b) La fonction est complètement vide, vous devrez récupérer les données depuis les pointeurs « data » dans les structures « image_container » pour pouvoir y appliquer la fonction de distance euclidienne. Vous devrez aussi écrire le prologue et épilogue de la fonction ainsi que gérer l'accès aux paramètres de la fonction. Pour vous aider des espaces dans la section .data ont été définis que vous pouvez utiliser ou modifier afin de ne pas être trop limité par le nombre restreint de registres en ***x86***.

⇨ Après avoir récupéré le pointeur sur la structure « ***image_container*** », vous pouvez récupérer ses différentes informations dans le même ordre que leur définition dans la structure. Par ex. :

si eax contient le pointeur sur le paramètre « ***img*** » vous pouvez accéder à son champ « ***uint8_t *data*** » en faisant accès mémoire sur *« 12(%eax)* », à sa hauteur sur *« 4(%eax)* », etc.

c) Cette fonction est appelée pour chaque coordonnée de l'image source qui devra être traité (vous n'avez pas besoin de vérifier si vous êtes en bord d'image, les coordonnées rouges dans l'image ci-dessus ne sont pas traités), **vous devez donc simplement implémenter le calcul de la somme des différences au carré entre le motif et la zone de l'image en x,y**. La fonction doit simplement retourner la somme des différences au carré calculée.

d) Vous pouvez tester votre code à l'aide du programme « *./pattern_detection* ». L'argument « *-c* » vous permet d'exécuter le même algorithme implémenté en C avant l'exécution de votre code assembleur afin de comparer le temps de calcul (ou pour débug). Le code C ainsi que le code SIMD doivent retourner le même résultat. Le programme génère une image « ***result.png*** » qui va encadrer le motif recherché dans l'image de base.

⇨ Afin de simplifier le débug et vos tests, une image 16x16 (small_duck.png) avec un motif 4x4 (duck_pattern.png) représentant la tête du canard de l'image small_duck.png sont fournies. Pour utiliser ces images, vous pouvez exécuter le programme de la manière suivante.

```
reds@reds2022: ./pattern_detection -i img/small_duck.png -k img/duck_pattern.png -c
```

⇨ Si vous lancez le programme en mode débug ce seront ces arguments qui seront utilisé.

Pour vous faciliter la gestion des calculs SIMD (4 valeurs 32-bit à la fois), la taille du motif en x et y sera toujours un multiple de 4, ceci vous évitera de gérer les cas de bord. Les valeurs des pixels 8-bit seront à étendre en 32-bit afin de ne pas avoir d'overflow lors de la mise au carré des différences ainsi que lors du calcul de la somme (voir instruction ***pmovzxbd***[1]).

[1] https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html#text=pmovzxbd&ig_expand=1881