Université de Paris Sud
———
Masters NSI & SETI

Architecture des Ordinateurs

Optimisation de codes SIMD pour le TSI

Lionel Lacassagne

Introduction

Les fichiers C nécessaires se trouvent à l'adresse `www.ief.u-psud.fr/~lacas/Teaching`.
Lire le document *règles de **codage** et routines de calcul* avant de commencer.
Faire un schéma pour **chaque** utilisation de l'instruction `shuffle` comme celui indiqué (Fig. 1)

1 Traitement du signal 1D

$$A_3 = \frac{1}{3}\begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \quad (1)$$

$$A_5 = \frac{1}{5}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad (2)$$

1. Codez la version sans optimisation des filtres 1D A_3 et A_5 en SSE
   `avg3_f32vector_SIMD(vfloat32 *X, int n, vfloat32 *Y);`
   `avg5_f32vector_SIMD(vfloat32 *X, int n, vfloat32 *Y);`
2. Donnez les spécificités de ces codes.
3. Mesurez le temps de calcul et comparez le à celui du code scalaire vectorisé (en utilisant le compilateur `icc` avec l'option `-msse2`.

2 Traitement d'images 2D

$$M_3 = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (3)$$

$$M_5 = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad (4)$$

1. Codez la version sans optimisation des filtres 2D M_3 et M_5 en SSE :
   `avg3_f32vector_SIMD(vfloat32 *X, int n, vfloat32 *Y);`
   `avg5_f32vector_SIMD(vfloat32 *X, int n, vfloat32 *Y);`
2. Donnez les spécificités de ces codes.

3. Mesurez le temps de calcul et comparez le à celui du code scalaire vectorisé (en utilisant le compilateur `icc` avec l'option `-msse2`.
4. Codez la version avec optimisation des filtres 2D M_3 et M_5 en SSE :
   `avg3_fast_f32matrix_SIMD(vfloat32 **X, int n, vfloat32 **Y);`
   `avg5_fast_f32matrix_SIMD(vfloat32 **X, int n, vfloat32 **Y);`
5. Donnez les spécificités de ces codes.
6. Mesurez le temps de calcul et comparez le à celui du code scalaire vectorisé.
7. Conclure.

3 Instructions disponibles

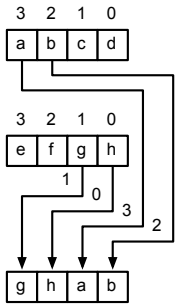

FIGURE 1 – fonctionnement `_mm_shuffle_ps`

Les instructions SSE suivantes sont définies pour manipuler des registres 128 bits composés de 4 nombres flottants 32 bits.

Accès mémoire :

. `x = _mm_load_ps(float *addr)` qui charge `x` à partir de l'adresse `addr`

. `_mm_store_ps(x, float *addr)` qui écrit `x` à l'adresse `addr`

Initialisation :

. `v = set_ps(f0, f1, f2, f3)` qui initialise le registre `v` avec `f0`, `f1`, `f2`, `f3`,

. `v = setr_ps(f0, f1, f2, f3)` qui initialise le registre `v` avec `f3`, `f2`, `f1`, `f0`, pour que les valeurs soient dans l'ordre croissant une fois qu'elles seront stockées en mémoire (via un `_mm_store_ps`).

Arithmétique

. `z = _mm_add_ps(x, y)` qui additionne `x` et `y`

Shuffle (Fig. 1)

. `z = _mm_shuffle_ps(x, y, val)` : mélange de registres :
  – Si `m1=[a,b,c,d]` et `m2=[e,f,g,h]`
  – et si `m3 = _mm_shuffle_ps(m1,m2,_MM_SHUFFLE(1,0,3,2))`
  – alors `m3 = [g,h,a,b]`