Université de Paris Sud
———
Masters NSI & SETI

Architecture des Ordinateurs

Optimisation de codes scalaires pour le TSI

Lionel Lacassagne

Introduction

Le but de ce TP est de prendre en main les outils de compilation et de chronométrie et d'appliquer des techniques d'optimisation de code à des algorithmes de traitement du signal (algorithmes 1D) et de traitement d'images (algorithmes 2D). dans un premier temps ces optimisations seront généralistes puis dans un second temps ces optimisations prendront en compte le domaine applicatif pour améliorer les optimisations.
Les fichiers C nécessaires se trouvent à l'adresse `www.ief.u-psud.fr/~lacas/Teaching`.
Pour chaque codage, donnez, dans un tableau récapitulatif, les spécificités de votre code, à savoir la complexité arithmétique (`MUL`+`ADD`), le nombre d'accès mémoire (`LOAD`+`STORE`) et l'intensité arithmétique (le ratio entre la complexité arithmétique et le nombre d'accès mémoire).

1 Traitement du signal 1D

1.1 Filtres linéaires

Les filtres moyenneurs (*average* en anglais) sont les plus simples des filtres linéaire. Soit le filtre A_k, le filtre de taille k (k est impair pour des raisons de symétrie : $k = 2r + 1$). Soient A_3 et A_5 des instances du filtre A_k pour $k = 3$ et $k = 5$. L'équation aux différences de ces filtre est :

$$y_k(n) = \frac{1}{k} \sum_{j=-k/2}^{j=+k/2} x(n+j) \quad (1)$$

$$y_3(n) = \frac{1}{3}(x(n-1) + x(n) + x(n+1) \quad (2)$$

$$y_5(n) = \frac{1}{5}(x(n-2) + x(n-1) + x(n) + x(n+1) + x(n+2) \quad (3)$$

$$(4)$$

Les versions A_3 et A_5 sont dites les versions déroulées de la version A_k (déroulage totale)

1. Codez le filtre A_k dans la fonction
   `avgk_f32vector(float32 *X, int n, int k, float32 *Y) ;`
2. Codez les filtres A_3 et A_5 dans les fonctions :
   `avg3_f32vector(float32 *X, int n, float32 *Y) ;`
   `avg5_f32vector(float32 *X, int n, float32 *Y) ;`
   en supprimant la boucle sur j. Cela s'appelle du déroulage de boucle, et dans ce cas précis un déroulage total.
3. Donnez les spécificités de chaque code et mesurez les temps de calcul en *cpp* (Cycle par Point) pour $n = 100^2$ et $n = 1000^2$. Que peut on observer ?

4. Afin d'optimiser davantage les codes, on décide de prendre en compte le recouvrement des données entre deux applications successives du filtre. Exprimez $y_3(n+1)$ en fonction de $y_3(n)$ et faire de même pour $y_5(n)$. En déduire de nouveaux codes plus rapides. Généralisez cela au filtre A_k.
5. Donnez les spécificités de ces codes et mesurez les temps de calcul en *cpp*.
6. Conclure.

1.2 Filtres non linéaires

2 Traitement d'images 2D

La notation générale des filtre pour exprimer qu'un signal x est convolué avec un filtre h est :

$$y(n) = \sum_{k=0}^{m-1} h(k)x(n-k) \tag{5}$$

Les valeurs de $h(k)$ sont les coefficients du filtre. Dans le cas des filtres moyenneurs, toutes ces valeurs sont égale à 1. Afin de normaliser la sortie du filtre, les coefficients sont normalisés (ici la somme des coefficients est égale à leur nombre : k, d'où la division par k). Une notation plus simple consiste à exprimer h sous forme de *noyau de convolution*. Les noyaux de convolution de A_3 et A_5 sont :

$$A_3 = \frac{1}{3}\begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \tag{6}$$

$$A_5 = \frac{1}{5}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{7}$$

Les versions 2D de ces filtres sont obtenus en les convoluants avec leur transposée :

$$M_3 = \frac{1}{3}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} * \frac{1}{3}\begin{bmatrix} 1 & 1 & 1 \end{bmatrix} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{8}$$

$$M_5 = \frac{1}{5}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} * \frac{1}{5}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{9}$$

Ainsi le moyenneur 2D M_3 consiste à faire la somme des points sur un voisinage 3×3 et à normaliser le résultat en divisant par la somme des coefficients : 9. Idem pour M_5. La version générale M_k réalise donc une somme de $k \times k$ points et normalisée par $1/k^2$.

1. Codez le filtre M_k dans la fonction
   `avgk_f32matrix(float32 **X, int n, int k, float32 **Y)`
   Il faut donc une première double boucle pour parcourir la matrice (l'image) et une seconde double boucle pour faire la somme des points, la seconde double boucle étant centrée sur le point de coordonnées (i, j).
2. Codez les filtres M_3 et M_5 dans les fonctions :
   `avg3_f32matrix(float32 **X, int n, float32 **Y) ;`
   `avg5_f32matrix(float32 **X, int n, float32 **Y) ;`
   en supprimant la double boucle interne.

3. Donnez les spécificités de chaque code et mesurez les temps de calcul en *cpp* (Cycle par Point) pour $n = 100^2$ et $n = 1000^2$. Que peut on observer ?
4. Afin d'optimiser davantage les codes, on décider de prendre en compte la façon dont les filtres 2D sont construits à partir des filtres 1D : les filtres de la famille M_k sont séparables (par construction) en deux filtres 1D. Proposez une façon d'optimiser ce calcul.
5. Codez ces nouvelles versions dans les fonctions :
   ```
   avg3_fast_f32matrix(float32 **X, int n, float32 **Y) ;
   avg5_fast_f32matrix(float32 **X, int n, float32 **Y) ;
   avgk_fast_f32matrix(float32 **X, int n, int k, float32 **Y) ;
   ```
6. Donnez les spécificités de ces codes et mesurez les temps de calcul en *cpp*.
7. Conclure.