

Projet : convolution

Version du 27 août 2015

Enseignants :

TODO : enlever les malloc des chronos et refaire les tests de perfs avec ces nouveaux codes

1 Introduction

Il existe en traitement d'image une opération très importante : le filtrage par convolution. De quoi s'agit-il ? Une convolution est l'opérateur noté \star et définie de la façon suivante :

$$f \star g(x) = \int f(x-y)g(y)dy \quad (1)$$

où f et g sont deux fonctions réelles intégrables. Pour les fonctions de \mathbb{R}^2 , la définition devient :

$$f \star g(x', y') = \int \int f(x' - x, y' - y)g(x, y)dxdy \quad (2)$$

Dans l'espace discret où vivent les images, la formule s'écrit :

$$f \star g(k, l) = \sum_i \sum_j f(k - i, l - j)g(i, j) \quad (3)$$

On supposera que nos fonctions ont un support (i.e. un domaine de définition) borné. Ainsi, les indices i et j varient dans un intervalle fini.

En particulier, on suppose f définie sur $[1, N] \times [1, P]$ et g définie sur $[-\frac{n}{2}, \frac{n}{2}] \times [-\frac{p}{2}, \frac{p}{2}]$, la formule (3) devient :

$$f \star g(k, l) = \sum_{\substack{1 \leq k - i \leq N \\ -\frac{n}{2} \leq i \leq \frac{n}{2}}} \sum_{\substack{1 \leq l - j \leq P \\ -\frac{p}{2} \leq j \leq \frac{p}{2}}} f(k - i, l - j)g(i, j) \quad (4)$$

On remarque immédiatement que cette définition est plus compliquée que la précédente. En effet, la convolution revient, entre autre, à translater une des deux fonctions. Sur un support infini, la translation n'a pas d'effet sur le support. En revanche, avec un support fini, le domaine est également translaté. La conséquence est donc que le support de la convoluée est différent des supports des fonctions convoluées. Dans l'exemple de l'équation 4, le support de $f \star g$ est $[1 - \frac{n}{2}, N + \frac{n}{2}] \times [1 - \frac{p}{2}, P + \frac{p}{2}]$. La figure 1 illustre cela : le calcul du pixel en bas à droite est possible ; sa valeur est a priori non nulle parce qu'elle dépend de la valeur d'un pixel de I (celui en gris), les autres étant nuls.

2 Applications aux images

En pratique, nous l'avons déjà dit, la convolution d'image est une opération très importante. Soit I une image de taille $N \times P$, $I(x, y)$ est donc la valeur de luminescence du pixel (x, y) . Cette image peut être *convoluée* avec une autre image k que l'on appelle *noyau* (*ou kernel* en anglais) de convolution. Le noyau k est généralement de petite taille (typiquement 3×3 , 5×5 ou 7×7) et représente souvent un opérateur différentiel (mais pas nécessairement), il est représenté centré en 0.

Par exemple, si l'on souhaite calculer la dérivée en x de I ($\frac{\partial I}{\partial x}$), on peut approcher par différence finie (développements de Taylor à l'ordre 1) l'opérateur :

$$\frac{\partial I(x, y)}{\partial x} \sim \frac{1}{2}(I(x+1, y) - I(x-1, y)) \quad (5)$$

Enseignants :

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

Soit en discret :

$$f'(x) = \frac{f(x+1) - f(x-1)}{(x+1) - (x-1)}$$

$$f'(x) = \frac{f(x+1) - f(x-1)}{2}$$

ce qui revient à calculer $I * k$ avec :

$$k = \begin{pmatrix} 0 & 0 & 0 \\ -1/2 & 0 & 1/2 \\ 0 & 0 & 0 \end{pmatrix} \quad (6)$$

Avec d'autres opérateurs, on peut lisser une image, extraire des contours, *etc.* On peut voir la convolution comme une opération de moyennage pondérée sur un voisinage fixé par le noyau de convolution (voir figure 1).

3 Algorithme séquentiel

Avec la définition donnée de la convolution discrète (équation (4)), on ne peut pas calculer la convolution sur les pixels au bord de l'image car elle nécessite l'accès à des pixels non définis, pourtant le calcul sur les bords est possible (voir la figure 1, en bas à gauche et en bas à droite).

Pour résoudre ce problème, il y a deux méthodes :

- soit on agrandit artificiellement la taille de l'image, les nouveaux pixels ont alors une valeur nulle,
- soit on exclut du calcul de convolution les pixels sur les bords de l'image.

Notre algorithme choisit la deuxième méthode car notre propos n'est pas d'implémenter rigoureusement la convolution mais de la paralléliser.

Pour obtenir de longs temps de calcul et illustrer les envois de données de MPI, l'opération de convolution est itérée un grand nombre de fois. On veut donc implémenter l'opération :

$$I *^n k = I * \underbrace{k * k * \dots * k}_{n \text{ fois}} \quad (7)$$

Comme nombre d'applications numériques (comme la résolution d'équations aux dérivées partielles) requièrent la répétition d'un grand nombre de convolutions, ce choix est pertinent.

Finalement, l'algorithme est simple :

1. Lecture de la ligne commande (protocole Argv).
2. Chargement de l'image (fonction lire_rasterfile).
3. Répéter `nbiter` fois la convolution de l'image (« pointée » par le champ `r.data` de la variable `r` de type `Raster`) par un des 5 filtres :

Enseignants :

Les trois étapes suivantes sont réalisées par la fonction :

```
int convolution( filtre_t choix, unsigned char tab[], int nbl, int nbc);
(attention : calcul sans les bords, donc pour les lignes 1 à nbl-2 (et idem pour les colonnes).
```

L'idée est de leur imposer l'utilisation de cette fonction pour la version parallèle avec comms bloquantes (pour la version avec comms non bloquantes (recouv. comms/calcul) on utilisera `convolution_bis()`, voir codes solutions).

- (a) allocation mémoire d'un tampon image intermédiaire,
- (b) pour chaque pixel (i, j) de l'image (sauf les bords) faire :

Enseignants :

le corps de cette boucle est réalisé par la fonction :

```
unsigned char filtre( filtre_t choix,
    unsigned char NO, unsigned char N,unsigned char NE,
    unsigned char O,unsigned char CO, unsigned char E,
    unsigned char SO,unsigned char S,unsigned char SE);
```

- le pixel (i, j) de l'image tampon reçoit une combinaison linéaire (c'est l'opération de convolution) du pixel (i, j) et de ses 8 plus proches voisins dans l'image (`r.data`).
 - (c) recopie du tampon intermédiaire dans l'image (`r.data`) et libération mémoire du tampon image intermédiaire.
4. Sauvegarde de l'image (fonction `sauve_rasterfile`).

Enseignants :

Attention : image au format Sun Rasterfile *en noir et blanc* : 0 \Leftrightarrow noir et 255 \Leftrightarrow blanc.

Les 5 différentes combinaisons linéaires représentent les cinq filtres possibles (voir l'instruction switch dans le listing) : ces opérations sont dans l'ordre : un filtre moyenneur, un autre filtre moyenneur (avec plus de poids au centre), un détecteur de contour (c'est l'opérateur Laplacien discrétilisé par différences finies : un tel opérateur donne une forte réponse sur les contours de type "pic", voir figure 2), un autre détecteur de contour qui détecte les contours de type "marche" (voir figure 2). et finalement un filtre médian (non linéaire : on trie les pixels selon leur luminosité et on retient la valeur médiane).

Enseignants :

Codage des filtres dans le code `convol.c` (énumération `filtre_t`) :

- « un filtre moyenneur » : `CONVOL_MOYENNE1` (valeur : 0)
- « un autre filtre moyenneur (avec plus de poids au centre) » : `CONVOL_MOYENNE2` (valeur : 1)
- « un détecteur de contour (contours de type "pic") » : `CONVOL_CONTOUR1` (valeur : 2)
- « un autre détecteur de contour (contours de type "marche") » : `CONVOL_CONTOUR2` (valeur : 3)
- « un filtre médian (non linéaire, qui n'est donc pas une convolution...) » : `CONVOL_MEDIAN` (valeur : 4)

Enseignants :

Abscisse avec (x, y) à la figure 2 signifie : « x ou (i.e. et/ou) y croît ».

4 Questions

1. Dans la fonction `convolution()`, pourquoi doit-on préparer un tampon intermédiaire au lieu de faire le calcul directement sur l'image ?
2. Quelles sont les séquences parallélisables de l'algorithme ?
3. Sachant que la taille du noyau k de convolution est de 3×3 pixels, quelle est la complexité théorique du calcul d'un pixel de $I * k$? Quel type d'équilibrage de charge doit-on prévoir entre les processeurs ?

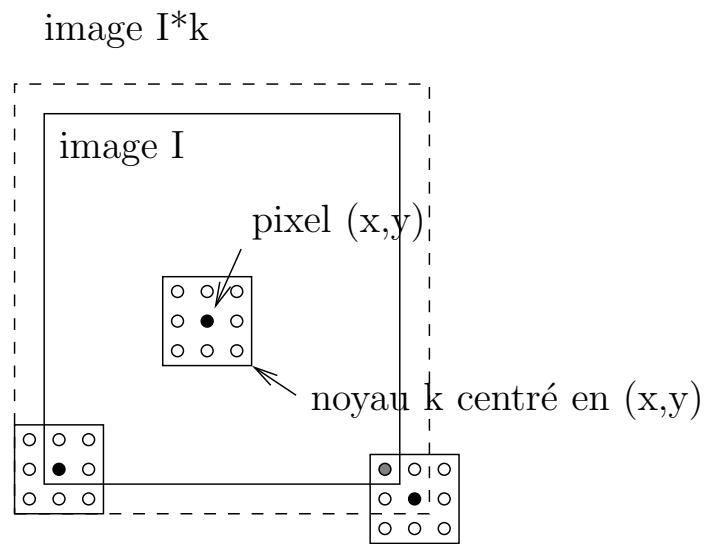


FIGURE 1 – Convolution de I par k : $I \star k(x, y)$ est la somme pondérée (par k) des valeurs de I en (x, y) et de leurs voisins. On illustre aussi le fait que le domaine de $I \star k$ (en pointillé) est le domaine de I augmenté de celui de k . Au-delà, les valeurs sont nulles.

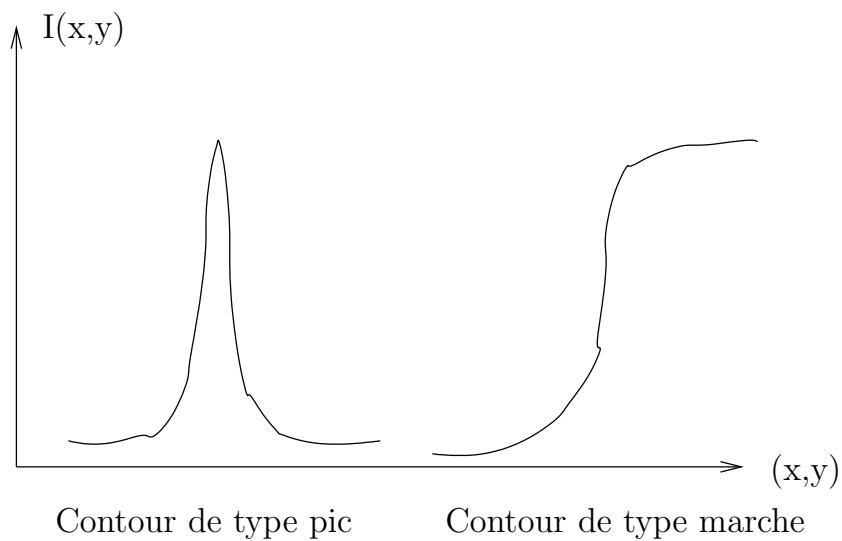


FIGURE 2 – Exemple de contours.

4. Quel découpage (répartition des données entre processeur) est naturel dans ce contexte ?
5. Quel problème (aux bords des blocs d'image) survient lors de l'itération de l'opération de convolution ?

Enseignants :

Voici mes notes pour les réponses aux premières questions.

PPAB :

TP 4 : convolution

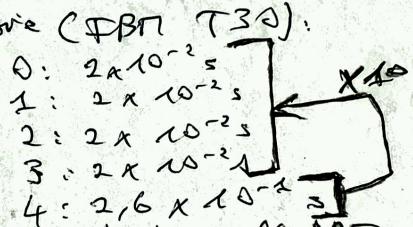
✓

3. 4 questions :

1/ tampon intermédiaire : car la nouvelle valeur d'un point dépend des anciennes valeurs de son voisinage
 ↳ besoin de deux tampons : - anciennes valeurs
 - nouvelles valeurs

2/ 1 séquence parallélisable :
 ↳ boucle sur les pixels de l'image
 Pq : la boucle sur "nbiter" (nombre de convolutions de l'image) n'est pas parallélisable car à chaque tour de boucle les données dépendent du tour de boucle précédent.

Temps de calcul sur Commodore (FBN, T33),
 avec femme 10.5ms et filtre 3x3 :



↳ insuffisant pour parallélisme (surtout au l' ARI,
 ou à la Polytech)

PPAR

R2

TP 4: convolution

3. 4 questions (suite): → de $f \star g$

3/ Pour le pixel (k, l) , la complexité de calcul est
(avec un noyau de convolution k de taille 3×3 pixels):

$$f \star g(k, l) = \sum_{i=-1}^1 \sum_{j=-1}^1 f(k-i, l-j) \cdot g(i, j)$$

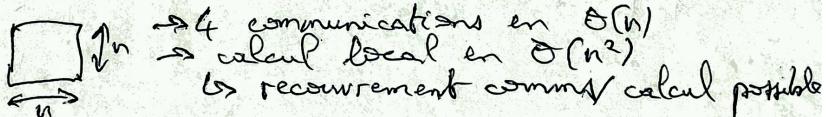
$$= O(1) \quad (3 \text{ multiplications et } 8 \text{ additions})$$

Il y a bien sûr besoin d'un équilibrage de charge
entre les processus mais l'équilibrage
de charge statique (basé sur une répartition
équitable des nombres de pixels tout en minimisant
les communications) doit suffire.

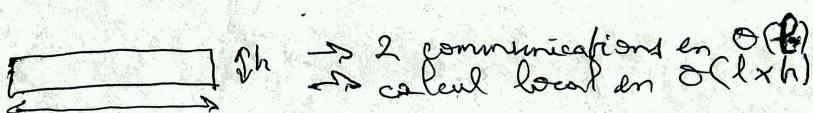
4/ 2 possibilités :

- décomposition par blocs (carres ou rectangulaires)
- décomposition par bandes ↗ à retenir (cf. feuille 2 bis)

Par blocs:



Par bandes:



~~Finissement~~ ↳ voir détails en feuille 2 bis.

5/ À chaque itération de la boucle sur "nbîter" (i.e. à chaque calcul de convolution) il faut "échanger" les "bords" avec ses voisins.
↳ 1 ligne ou 1 colonne à chaque fois

PPAR:
TP 4 : convolution

R² bis

3. 4 questions (suite):

4/ (suite)

Pour une aire fixée, quel rectangle a le plus petit périmètre?



$$\text{aire: } a = l \times h \\ \text{périmètre: } p = 2(l+h)$$

$$\text{Aire fixée} \Rightarrow a = l \times h \\ \Rightarrow h = \frac{a}{l}$$

$$\text{D'où } p = 2(l+h) = 2\left(l + \frac{a}{l}\right)$$

$$\frac{\partial p}{\partial l} = 2 - 2a \frac{1}{l^2}$$

$$\begin{aligned} \text{Minimum de } p: \frac{\partial p}{\partial l} &= 0 \Rightarrow \frac{1}{l^2} = \frac{2}{2a} \\ &\Rightarrow l^2 = a \\ &\Rightarrow l = \sqrt{a} \end{aligned}$$

Et comme $h = \frac{a}{l} \Rightarrow h = \sqrt{a}$
 C'est bien le carré qui a le plus petit périmètre.

Dans: a priori meilleur recouvrement comme calcul avec le carré → réseau modélisé par $T(N) = \text{latence} + N$

→ avec le petit message → importance de la latence
 → peu de gros messages → importance de la bande passante

En pratique, la meilleure solution (entre bloc et bande) dépend de la latence et du débit du réseau.

Et: attention aux blocs avec les opérateurs de convolution qui utilisent les voisins en "coin" → besoin de communiquer 4 + 4 processus (N, E, O, S et NE, NO, SE, SO) → on préfère donc les bandes pour ce TP...

6. Implémenter un algorithme parallèle avec des envois bloquants de messages.

Enseignants :

Voici mes notes.

PPAB 13

TP 4 : convolution

Gestion des tailles des tampons locaux :

NP: nombre de processus p: rang	h : nombre (total) de lignes w : nombre (total) de colonnes
------------------------------------	--

Pour $0 < p < NP - 1$ (cas général), on a pour le tampon local ("local-data"):

avec: l_h : "local h "

$$l_h = \frac{h}{NP} + (p > 0 ? 1 : \infty) + (p < NP - 1 ? 1 : \infty)$$

Et on appelle bien la fonction "convolution" avec convolution (choix, local-data, ~~l_h~~, x) car cette fonction ne fait pas le calcul sur les bords (i.e. pas de calcul pour lignes 0 et $h-1$, ni pour colonnes 0 et $w-1$)

Rq.: A besoin de distinguer les processus & dans les gather et scatter ...

PPARTP 4 : convolution

Algo parallèle : faire envoi bloquants
version - par bandes

Hypothèse ("nombre_lignes" est divisible par "nombre_processus")

1/ Initialization:

- tous les processus reçoivent "filtre" et "width" sur la ligne de commande
- le processus "root" :
 - lit le fichier Raster
 - récupère $\lceil \frac{h}{w} \rceil$ (nombre de lignes)
 - " w " (nombre de colonnes)
 - vérifie l'hypothèse (B)
- opération Broadcast depuis processus root pour diffuser $\lceil \frac{h}{w} \rceil$ et " w " à tous le monde.
- $Broadcast = h/NP + (p > 1 ? 1 : 0) + (p < NP - 1 ? 1 : 0)$
- Allocation mémoire du tampon local dans chaque processus
 - ↳ la taille: local_h/w
 - ↳ nom: local_data

2/ Répartition de l'image sur tous les processus:

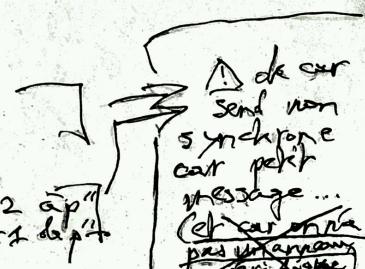
- Avec MPI-Scatter (aux paramètres : processus p est particulier)
- ou "à la main" → distinguer processus p et processus $NP-p$ (en envoi et en réception)

3/ Convolution en // :

- Pour i allant de 0 à $width-1$
- Si processus précédent existe :
 - envoyer la ligne 1 à $p-1$
 - recevoir la ligne 0 de p
- Fin si
- Si processus suivant p existe :
 - envoyer la ligne $local_h-2$ à p
 - recevoir la ligne $local_h-1$ de p
- Fin si
- convolution (filtre, local_data, local_h, w)

Fin pour

Non : pour éviter tout inter-blocage avec des gros messages il faut inverser l'ordre de réception dans "si processus suivant existe"



4 Collection des images locales :
— MPI-Scatter
— ou à la main

5 Termination :

- libération immédiate du tampon local dans chaque processus
- le processus root sauve l'image totale dans un nouveau fichier.

Préférer la version avec Scatter que la version sans Scatter (plus simple).

Images de tests disponibles :

- femme10.ras de taille 256 × 256,
- Sukhothai_4080x6132.ras de taille 4080 × 6132 (nombre de lignes multiple de 12 (pour

pc113 et pc116 par ex.)),
— Sukhothai_4096x6156.ras de taille 4096×6156 (nombre de lignes puissance de 2).
Ils peuvent donc vérifier leur code sur femme10.ras et faire les tests de performance sur Sukhothai_*.ras.
Attention aux quotas disque pour Sukhothai_*.ras. : chaque image fait environ 24 Mo... On peut leur conseiller d'utiliser le répertoire /tmp/ en cas de problème.

7. Question subsidiaire : implémenter l'algorithme avec cette fois des primitives non-bloquantes et de façon à ce que les temps de calcul recouvrent les temps de communication. Analyser les performances obtenues.

Enseignants :

Voici mes notes.

PPAR

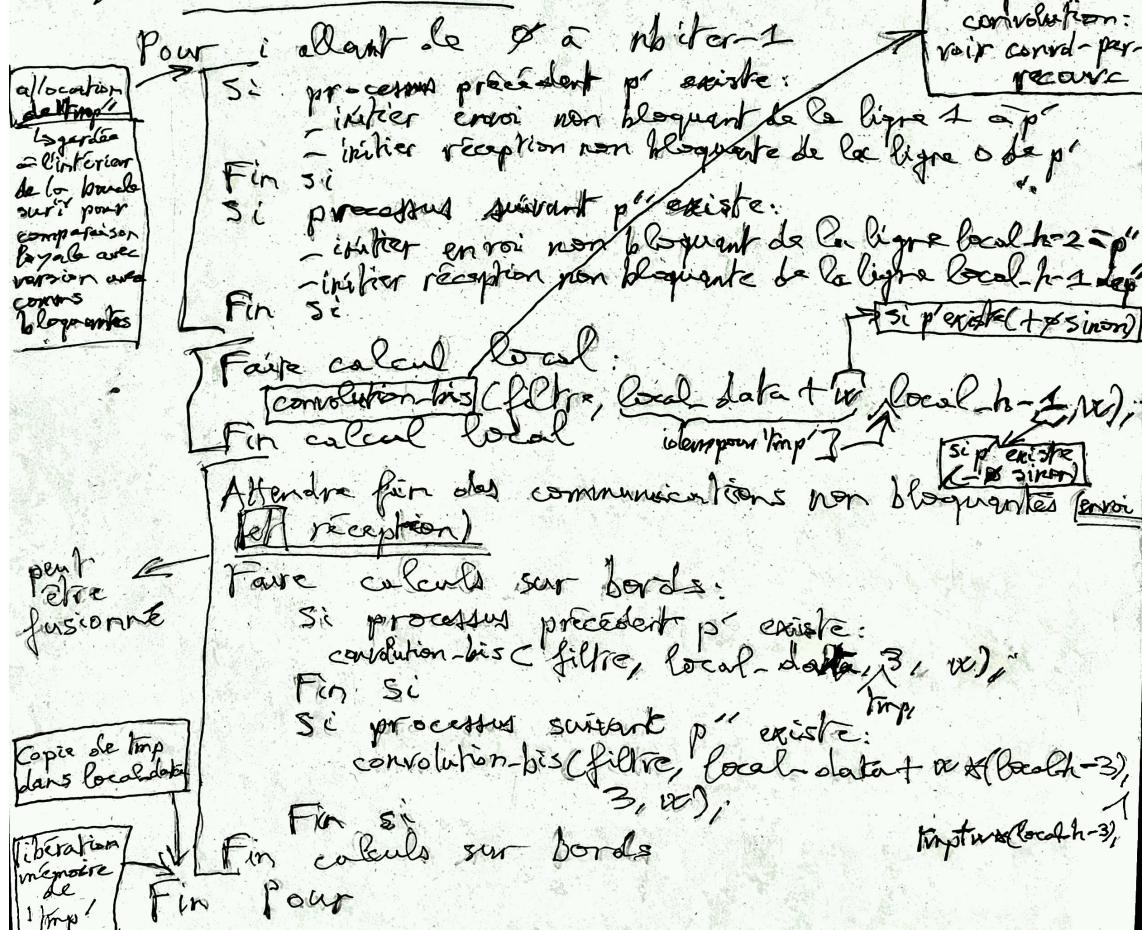
15

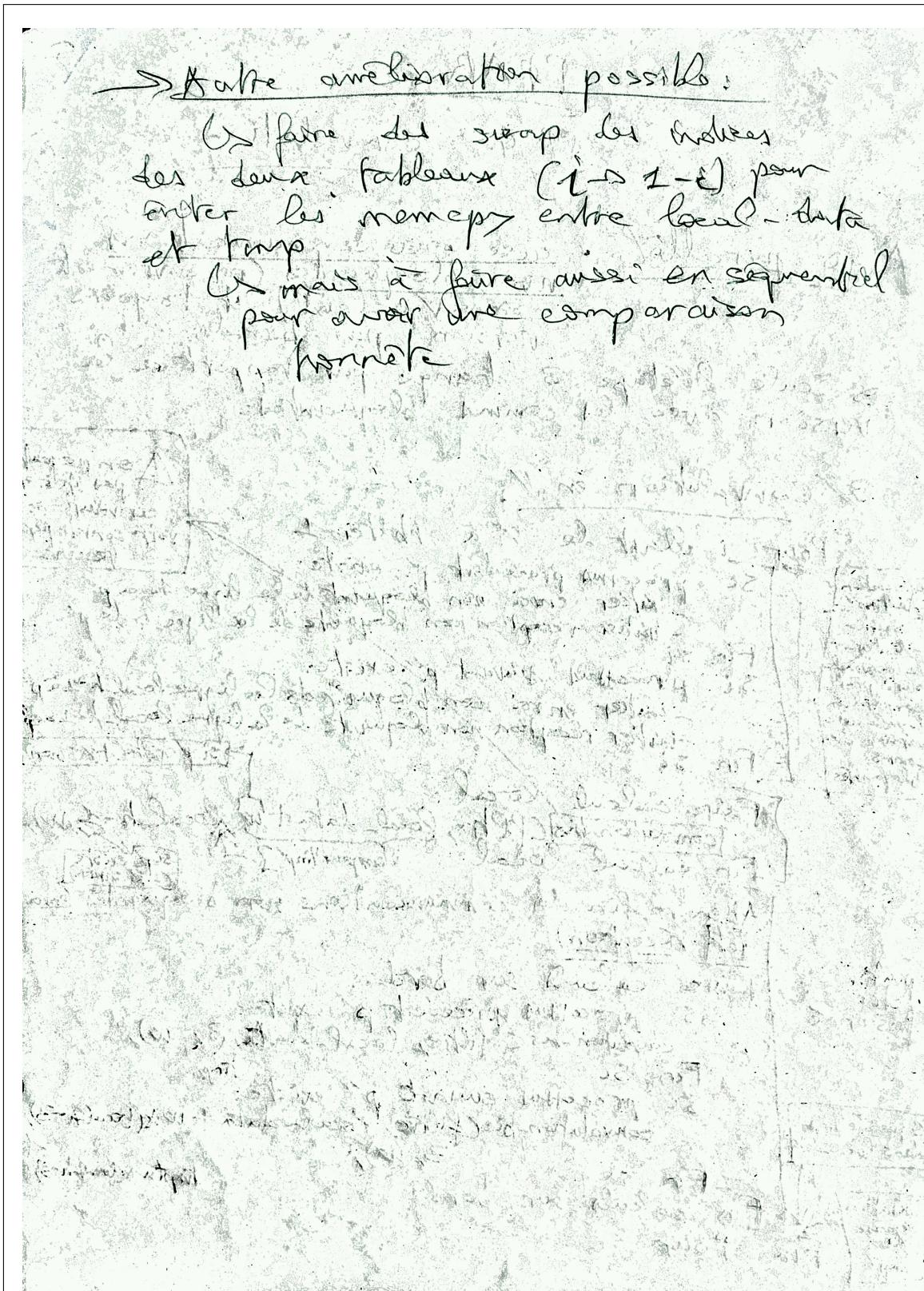
TP 4: convolution

Algo parallèle: avec ~~des~~ communications non bloquantes (renvoi immédiat) et toujours version par bords

→ Seule l"étape 3 change par rapport à la version avec les commandes bloquantes.

3/ Convolution en II:





Le gain n'est pas visible avec CONVOL_MEDIAN (valeur : 4), car le temps de communication est bien inférieur au temps de calcul (100 fois plus petit).

Résultats à l'EPU :

```
[fortin@pc170 Codes]$ ./convol femme10.ras 4 1000
```

```

Temps total de calcul : 59.9231 seconde(s)
[fortin@pc170 Codes]$ mpirun -c 4 convol-par femme10.ras 4 1000
Temps total de calcul : 16.5164 seconde(s)
[fortin@pc170 Codes]$ mpirun -c 4 convol-par-recouv femme10.ras 4 1000
Temps total de calcul : 16.7335 seconde(s)

[fortin@pc171 Codes]$ mpirun -c 4 convol-par-timings femme10.ras 4 10
Proc 1: i=0 t_com = 0.000576019 t_comp = 0.017344
Proc 1: i=1 t_com = 0.000180006 t_comp = 0.0151749
Proc 1: i=2 t_com = 0.000180006 t_comp = 0.014817
Proc 1: i=3 t_com = 0.000175953 t_comp = 0.0146379
Proc 1: i=4 t_com = 0.000178099 t_comp = 0.0145152
Proc 1: i=5 t_com = 0.000176191 t_comp = 0.014456
Proc 1: i=6 t_com = 0.000176907 t_comp = 0.0144119
Proc 1: i=7 t_com = 0.000175953 t_comp = 0.0143759
Proc 1: i=8 t_com = 0.00017786 t_comp = 0.0143299
Proc 1: i=9 t_com = 0.000178099 t_comp = 0.014318
Temps total de calcul : 0.197348 seconde(s)

```

Le gain est plus visible avec les autres filtres ($\approx 10\%$ de gain avec recouv. comms/calcul) :

```

[fortin@pc171 Codes]$ mpirun -c 4 convol-par femme10.ras 0 1000
Temps total de calcul : 2.21246 seconde(s)
[fortin@pc171 Codes]$ mpirun -c 4 convol-par-recouv femme10.ras 0 1000
Temps total de calcul : 2.0726 seconde(s)
[fortin@pc171 Codes]$
[fortin@pc171 Codes]$ mpirun -c 4 convol-par femme10.ras 1 1000
Temps total de calcul : 2.21208 seconde(s)
[fortin@pc171 Codes]$ mpirun -c 4 convol-par-recouv femme10.ras 1 1000
Temps total de calcul : 2.07662 seconde(s)
[fortin@pc171 Codes]$
[fortin@pc171 Codes]$ mpirun -c 4 convol-par femme10.ras 2 1000
Temps total de calcul : 1.70997 seconde(s)
[fortin@pc171 Codes]$ mpirun -c 4 convol-par-recouv femme10.ras 2 1000
Temps total de calcul : 1.54598 seconde(s)
[fortin@pc171 Codes]$
[fortin@pc171 Codes]$ mpirun -c 4 convol-par femme10.ras 3 1000
Temps total de calcul : 1.80394 seconde(s)
[fortin@pc171 Codes]$ mpirun -c 4 convol-par-recouv femme10.ras 3 1000
Temps total de calcul : 1.61938 seconde(s)

```

En effet le grain de calcul est plus faible (10 fois plus faible) pour les 4 premiers filtres que pour CONVOL_MEDIAN :

```

[fortin@pc171 Codes]$ for i in `seq 0 4`; do echo "Filtre = $i" ; \
mpirun -c 4 convol-par-timings femme10.ras $i 5 ; echo ""; done
Filtre = 0
Proc 1: i=0 t_com = 3.91006e-05 t_comp = 0.00202394
Proc 1: i=1 t_com = 0.000176907 t_comp = 0.00198793
Proc 1: i=2 t_com = 0.000177145 t_comp = 0.00199103
Proc 1: i=3 t_com = 0.000177145 t_comp = 0.00197911
Proc 1: i=4 t_com = 0.000176907 t_comp = 0.00197983

```

```

Temps total de calcul : 0.0359812 seconde(s)

Filtre = 1
Proc 1: i=0 t_com = 0.000590086 t_comp = 0.00201201
Proc 1: i=1 t_com = 0.000178099 t_comp = 0.00198388
Proc 1: i=2 t_com = 0.000176907 t_comp = 0.00199318
Proc 1: i=3 t_com = 0.000176907 t_comp = 0.00198007
Proc 1: i=4 t_com = 0.000181913 t_comp = 0.00197411
Temps total de calcul : 0.0346761 seconde(s)

Filtre = 2
Proc 1: i=0 t_com = 4.00543e-05 t_comp = 0.00148892
Proc 1: i=1 t_com = 0.00017786 t_comp = 0.00141406
Proc 1: i=2 t_com = 0.000172853 t_comp = 0.00147581
Proc 1: i=3 t_com = 0.000175953 t_comp = 0.00145292
Proc 1: i=4 t_com = 0.000175953 t_comp = 0.00142312
Temps total de calcul : 0.033653 seconde(s)

Filtre = 3
Proc 1: i=0 t_com = 4.00543e-05 t_comp = 0.00158215
Proc 1: i=1 t_com = 0.00017786 t_comp = 0.00159717
Proc 1: i=2 t_com = 0.000176907 t_comp = 0.00156593
Proc 1: i=3 t_com = 0.00017786 t_comp = 0.00152493
Proc 1: i=4 t_com = 0.00017786 t_comp = 0.00152802
Temps total de calcul : 0.140285 seconde(s)

Filtre = 4
Proc 1: i=0 t_com = 3.98159e-05 t_comp = 0.0178421
Proc 1: i=1 t_com = 0.000185013 t_comp = 0.0153558
Proc 1: i=2 t_com = 0.000178099 t_comp = 0.0150099
Proc 1: i=3 t_com = 0.000176907 t_comp = 0.014817
Proc 1: i=4 t_com = 0.000172853 t_comp = 0.014704
Temps total de calcul : 0.1096 seconde(s)

```

N.B. : normalement pas de progression des commms non bloquants lorsque le thread n'est pas dans un appel MPI avec les implémentations actuelles de MPI (d'apres Babeth) : là on gagne un peu car (d'apres Babeth) les appels non bloquants progressent à chaque fois qu'on fait un appel non bloquant (i.e. ils se font progresser les uns les autres) alors que ce n'est pas le cas avec des appels bloquants.

Enseignants :

Comparaison « qualitative » des différents filtres (avec 1 itération) :

- meilleurs résultats avec CONVOL_MOYENNE2 qu'avec CONVOL_MOYENNE1
- meilleurs résultats avec CONVOL_CONTOUR2 qu'avec CONVOL_CONTOUR1
- tous filtres comparés : « meilleure » image obtenue avec CONVOL_MEDIAN

Plus de détails sur le filtre CONVOL_CONTOUR1 :

16

PPAR
TP 4: convolution

La placette discret: (voir Wikipedia)

$$\Delta_{\text{discret}} f = \frac{[f(x+h, y) + f(x-h, y) - 2f(x, y)] + [f(x+h, y+1) + f(x-h, y+1) - 2f(x, y+1)]}{h^2}$$

~~car: dérivée seconde discrète:~~

$$D^2 f: x \mapsto \frac{f(x+\delta x) + f(x-\delta x) - 2f(x)}{\delta x^2}$$

car dérivée discrète:

$$f'(x) = \frac{f(x+\delta x) - f(x)}{\delta x}$$

$$\begin{aligned} f''(x) &= \frac{f'(x+\delta x) - f'(x)}{\delta x} \\ &= \frac{1}{\delta x} \left(\frac{f(x+2\delta x) - f(x+\delta x)}{\delta x} - \frac{f(x+\delta x) - f(x)}{\delta x} \right) \end{aligned}$$

D'où le résultat avec: $x+2\delta x \rightarrow x+\delta x$
 $x+\delta x \rightarrow x$
 $x \rightarrow x-\delta x$

(Rappel: en 2D, $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$)

CONVOL - CANTOUR 1:

$$\begin{bmatrix} * & -1 & * \\ -1 & 4 & -1 \\ * & -1 & * \end{bmatrix}$$

$$\Delta = 4I(x, y) - I(x-1, y) - I(x+1, y) - I(x, y-1) - I(x, y+1)$$

$$\Delta = (I(x, y) - I(x-1, y)) + (I(x, y) - I(x+1, y)) + (I(x, y) - I(x, y-1)) + (I(x, y) - I(x, y+1))$$

\Rightarrow puis: $4*|\Delta|$ (avec max=255) \rightarrow "valeur absolue": pour avoir des valeurs positives
 \rightarrow "4*": pour éduire l'image / accentuer les contrastes