Parallélisation de l'algorithme du K-means sur un système reconfigurable Application aux images hyper-spectrales

Parallelization of the K-means algorithm on a reconfigurable system Application to hyper-spectral images

par Dominique LAVENIER

CNRS/IRISA Campus de Beaulieu - 35042 Rennes cedex lavenier@irisa.fr

résumé et mots clés

L'article présente une architecture parallèle spécialisée pour l'algorithme du *K-means*, un algorithme de classification qui regroupe les objets de manière non hiérarchique. Nous proposons une mise en œuvre sur un système reconfigurable composé d'un PC couplé à une carte FPGA par l'intermédiaire de son bus d'entrées/sorties. L'expérimentation porte sur la segmentation d'images hyper-spectrales et démontre que les temps de traitement peuvent être réduits de quelques heures à quelques minutes. Cette réalisation met également en évidence l'influence de la qualité de la liaison entre le processeur et la carte FPGA sur les performances globales du système.

Système reconfigurable, architecture parallèle, images hyper-spectrales algorithmes du K-means.

abstract and key words

The article presents a parallel architecture dedicated to the k-means algorithm used for clustering objects in a non hierachical way. We propose an implementation on a reconfigurable system made of a PC and a FPGA board closely coupled through the I/O bus. Experimentations have been carried on a set of hyper-spectral images and show that the computation time is reduced from a few hours to a few minutes. We also points out the influence of the channel quality between the processor and the FPGA board over the global system performances.

reconfigurable system, parallel architecture, hyper-spectral images, K-means algorithm

1. introduction

Les systèmes reconfigurables se répartissent en plusieurs familles suivant le couplage plus ou moins étroit entre la partie reconfigurable et l'unité centrale [10]. Le couple(PC carte-FPGA) représente l'une de ces familles. La carte s'insert dans un *slot* d'extension et communique par le bus d'entrées/sorties de la machine (typiquement un bus PCI). C'est une solution aisée à mettre en œuvre dans la mesure où de nombreuses cartes commerciales sont disponibles [8][14].

Un des objectifs des cartes reconfigurables est d'apporter au microprocesseur un complément significatif en puissance de calcul, notamment pour des applications coûteuses en temps et pour lesquelles il n'existe pas, *a priori*, de produits spécifiques (carte dédiée, ASIC, *etc*). Ceci a été démontré à maintes reprises [11], et plus particulièrement pour des applications où les calculs sont concentrés dans des nids de boucles. Leur parallélisation sur des structures reconfigurables se révèle alors extrêmement efficace [12][7].

La parallélisation de l'algorithme du *K-means* se place dans ce contexte. Le but est de réduire les temps de calcul lorsque cet algorithme est appliqué sur un volume de données conséquent, ici des images hyper-spectrales. De telles images représentent chacune plus de 100 Moctets d'informations. Un pixel est un vecteur de plusieurs centaines de valeurs, chacune d'elle correspondant à une intensité codée sur 8, 12 ou 16 bits pour une longueur d'onde donnée. L'algorithme du *K-means* s'utilise soit à des fins de compression, soit pour établir une vue synthétique d'une image fragmentée en plusieurs centaines de spectres. Dans les deux cas, le traitement d'une image (512×614 pixels $\times$ 224 bandes spectrales) prend deux à trois heures sur une machine standard, c'est-à-dire un PC ou une station de travail doté d'un microprocesseur de dernière génération, et avec suffisamment de mémoire pour contenir l'image hyper-spectrale en entier. L'adjonction d'un carte accélératrice reconfigurable sur laquelle une architecture parallèle est mise en œuvre réduit ce temps à quelques minutes.

Si l'article s'appuie sur le traitement d'images hyper-spectrales, l'architecture parallèle développée pour accélérer l'algorithme du *K-means* n'en reste pas moins générique. Le système cible, comme évoqué précédemment, est un PC connecté par son bus d'entrées/sorties à une carte reconfigurable. Nous faisons également l'hypothèse que l'image ne peut pas être stockée sur la carte. Deux raisons motivent cette hypothèse. La première est que les cartes disponibles offrent, sauf exception, des ressources mémoire limitées, en tous les cas nettement inférieures à la capacité requise pour les données que nous manipulons. La seconde est que la diversité des produits se traduit par des schémas d'interconnexion extrêmement variés entre les composants FPGA et les composants mémoire ; définir une architecture générique, *i.e*. un code VHDL portable et efficace, qui prenne en compte cette diversité est quasi impossible.

À ce jour, et à notre connaissance, peu de travaux sur la parallélisation de l'algorithme du *K-means* ont été effectués. L'approche de M. Leeser, Northeastern University, Boston, USA, [1][3][4][5], contrairement à ce que nous proposons, implante l'algorithme du *K-means* en entier sur une carte reconfigurable, mais pour des images multi-spectrales, c'est-à-dire avec un nombre de spectres beaucoup plus faible (quelques dizaines). L'image peut alors résider sur la carte, ce qui optimise l'accès aux données. Par contre, le design est dépendant de la carte, ici une carte reconfigurable *WildStar* d'Annapolis Micro Systems, Inc. comprenant 3 composants FPGA Xilinx Virtex 1000 et quelques dizaines de Moctets de mémoire.

La contrainte que nous nous sommes fixée (pas d'image stockée à proximité du composant FPGA) a un impact énorme sur les performances du système. L'algorithme du *K-means* est itératif et chaque itération parcours l'image complète. Cela signifie que le temps de transfert de la mémoire principale (où est stockée l'image) vers la carte reconfigurable est un élément extrêmement déterminant. En fait, dans la suite de l'article, nous montrons que la puissance de la machine sur laquelle est connectée la carte joue un rôle assez mineur. Par contre, sa capacité à transférer rapidement de l'information vers les périphériques est primordiale.

L'architecture parallèle mise en œuvre sur la carte reconfigurable est un réseau systolique linéaire. Nous en avons implanté une version adaptée au cas des images hyper-spectrales, mais l'architecture décrite convient à tous types de données dès lors que la représentation d'une donnée élémentaire est un vecteur de valeurs. Cette architecture est extensible à un nombre quelconque de classes. La limitation provient des ressources disponibles sur la carte reconfigurable.

La suite de l'article est structurée de la manière suivante : la section 2 introduit rapidement l'algorithme du *K-means* et en fait une analyse dynamique pour pointer les zones coûteuses en calcul, donc les portions de code candidates à la parallélisation. La section 3 est consacrée à la parallélisation de l'algorithme du *K-means* : elle présente l'architecture, puis une estimation des performances par rapport à la bande passante disponible entre le processeur et la carte reconfigurable. La section 4 expérimente l'architecture sur la carte Spyder, une carte comprenant un seul composant FPGA, un Virtex (XCV800) de Xilinx. Une image de taille $256 \times 256 \times 224$ est segmentée en 64 classes en une minute, contre 45 minutes sur un PC à 600 MHz, et ce avec une bande passante relativement faible entre le PC et la carte (16 Moctets/seconde). La section 5 conclue cet article.

2. l'algorithme du *K-means*

2.1. description

L'algorithme du *K-means* [13] appliqué aux images hyper-spectrales regroupe les pixels en K ensembles distincts. Le nombre

K est fixé à l'avance. Chaque ensemble est représenté par un pixel moyen (ou centre) calculé à partir de tous les autres pixels de cet ensemble. L'algorithme est itératif et procède de la manière suivante :

```
- affectation aleatoire des pixels aux K classes (1)
- calcul des K centres                           (2)
- boucler NB_ITERS fois                          (3)
  - pour chaque pixel i                          (4)
   - Ci = classe du pixel i                      (5)
   - determiner la classe Cj dont le centre      (6)
   - est le plus proche du pixel                 (7)
   - si Ci <> Cj                                 (8)
    - bouger le pixel i vers la classe Cj        (9)
    - recalculer les centres des classes Ci et Cj (10)
```

Au départ, les pixels sont répartis aléatoirement entre les K classes (ligne 1). Les centres de chaque classe sont ensuite calculés à partir de cette répartition initiale (ligne 2). Puis l'algorithme rentre dans une phase itérative (ligne 3) où chaque tour de boucle parcours l'image entière (ligne 4) : pour chaque pixel on détermine sa proximité par rapport aux K classes. Cela revient à calculer une distance entre ce pixel et les centres des classes (lignes 6-7). S'il s'avère que le pixel est plus proche d'une autre classe, alors il change de classe (ligne 9). Comme les classes sont modifiées, il faut alors recalculer de nouveaux centres (ligne 10).

Cet algorithme supporte de nombreuses variantes qu'il est hors de propos de détailler ici, notamment sur la manière d'initialiser les classes qui influe fortement sur le regroupement final. Notons simplement que le nombre d'itérations (NB _ITERS, ligne 3) peut être déterminé à l'avance ou calculé au cours de l'exécution. On peut décider de continuer tant que des pixels changent de classes, ou alors d'arrêter dès que l'on a atteint un faible pourcentage de mouvement entre classes. De même, la mise à jour des centres (ligne 10) n'est pas obligatoire dès qu'un pixel change de classe ; la solution la plus extrême est de faire une mise à jour à chaque fin de boucle. En pratique, on choisit une approche intermédiaire qui consiste à faire une remise à jour tous les B pixels : par exemple à chaque fois que l'on a traité une ou plusieurs lignes de l'image. Enfin, le calcul de la distance (lignes 6-7) est un paramètre important puisqu'il conditionne la notion de proximité entre pixels. Tous les types de distance peuvent être acceptés, parmi lesquels les distances de Manhattan et Euclidienne sont souvent considérées [1].

2.2. analyse dynamique

La parallélisation d'un algorithme repose d'abord sur son comportement dynamique. En effet, il est important de détecter et quantifier les parties du code qui représentent l'essentiel du temps de calcul. C'est à partir de cette analyse que l'on peut ensuite imaginer un schéma de calcul propre à accélérer le traitement.

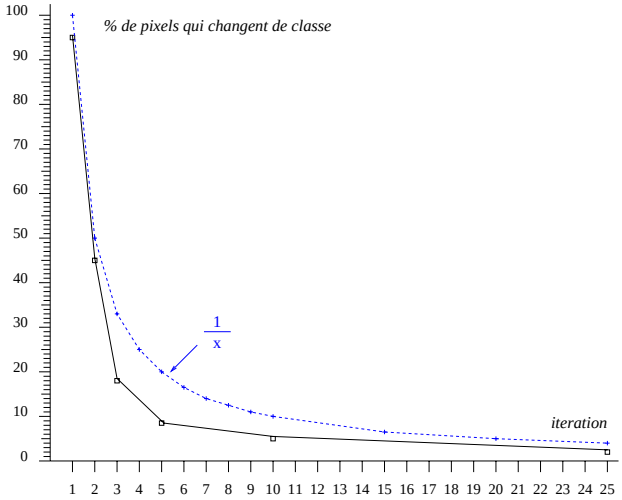


Figure 1. – Pourcentage de pixels qui changent de classe en fonction de l'indice d'itération. Cette courbe a été déterminée expérimentalement pour diverses valeurs de K (nombre de classes) et pour diverses tailles d'images hyper-spectrales.

Dans notre cas il convient, bien sûr, d'accélérer les calculs répétitifs, c'est-à-dire ceux qui sont situés à l'intérieur de la boucle principale. Si on se focalise sur cette partie, on s'aperçoit que les traitements potentiellement coûteux se concentrent sur le calcul des distances d'un pixel avec les K centres (lignes 6-7) et sur la remise à jour des centres lorsqu'un pixel change de classe (ligne 10).

La distance d_p entre 2 pixels est déterminée par :

$$d_p = \sum_{b=1}^{\text{NB-BANDES}} \delta(\text{PIXEL}[b], \text{CENTER}[b]) \qquad (1)$$

La fonction δ symbolise le calcul d'une distance élémentaire entre deux composantes des vecteurs PIXEL et CENTER. S'il y a NB_PIXELS dans une image, il faut exécuter ce calcul de distance (d_p) NB_PIXELS $\times$ K fois à chaque itération. C'est une constante. Par contre, la mise à jour des centres n'intervient que lorsque des pixels passent d'une classe à l'autre. La quantification n'est pas possible statiquement et requiert l'analyse d'une trace d'exécution. La figure 1 donne le pourcentage de pixels qui changent de classe en fonction de l'indice d'itération. La courbe a été tracée à partir de diverses exécution du *K-means* sur des images hyper-spectrales de taille différente et pour un nombre de classes variables. C'est donc une moyenne, mais elle montre deux choses : (1) les mouvements entre classes interviennent majoritairement lors des premières itérations ; (2) le nombre total de mouvements est relativement faible, ce qui indique qu'une faible part du calcul est réservé à la mise à jour des centres des classes.

Le centre d'une classe est la moyenne des éléments de cette classe. C'est donc un vecteur de NB_BANDES valeurs (le nombre de spectres) dont chaque composante CENTER $[k][b]$ est déterminé par :

$$\text{CENTER}[k][b] = \frac{\sum_{n=1}^{N_k} \text{PIXEL}[n][b]}{N_k} \quad (2)$$

$\text{CENTER}[k][b]$ représente la $b^{\text{ème}}$ composante du centre de la classe k, et N_k le nombre de pixels appartenant à cette classe. Pour limiter les calculs lors de la ré-actualisation d'un centre on maintient à jour K vecteurs de taille NB_BANDES qui conservent l'accumulation de toutes les composantes des pixels pour une classe donnée. Ainsi, pour calculer le nouveau centre lors d'un mouvement de pixel, on ajoute (resp. retranche) les composantes du pixel entrant (resp. sortant) à ce vecteur, on incrémente (resp. décrémente) N_k et le nouveau centre est obtenu en divisant le vecteur accumulateur par N_k.
La mise à jour d'un centre est donc de même complexité que le calcul de distance entre un pixel et un centre ($O($ NB_BANDES) opérations élémentaires). Par contre, cette tâche est effectuée bien moins souvent comme le montre la figure 1. Une étude plus détaillée, disponible dans [6], montre qu'en réalité plus de 99,5 % du temps est passé dans le calcul des distances, et que la mise à jour des centres des classes reste très marginale. Cette marginalité est d'autant plus forte que le nombre de classes est grand.
La conséquence immédiate de cette analyse est que la parallélisation du code ne peut porter que sur le calcul des distances d'un pixel contre les K classes.

3. parallélisation

3.1. architecture

Le support matériel visé pour accélérer l'algorithme du *K-means* consiste en une machine hôte (une station de travail ou un PC) à laquelle est connectée une carte reconfigurable du commerce. La carte communique par les ports d'entrées/sorties de la machine hôte, par exemple le bus PCI. Dans le problème qui nous intéresse, seule la partie calcul des distances est déportée sur la carte reconfigurable, le reste étant exécuté par la machine hôte. Nous nous focalisons donc maintenant sur cette partie.

L'idée de base est de calculer en parallèle la distance d'un pixel contre K classes. La figure 2 représente l'architecture parallèle. Elle est composée d'un réseau systolique linéaire de K processeurs traversé principalement par un flot de pixels. À chaque processeur est associé une classe ; il mémorise donc la valeur du centre de cette classe (un vecteur de NB_BANDES valeurs). Un processeur effectue de manière continue le cycle suivant :

• réception de données de son voisin de gauche
• calcul d'une distance
• émission de données vers le processeur de droite

Plus précisément, un processeur reçoit trois données : un pixel (Pix), une distance (Din) et un indice (Kin). Le triplet (Pix, Din, Kin) indique qu'une distance minimum a été trouvée entre le pixel Pix et le centre de la classe Kin, et que cette distance vaut Din. Le processeur calcule ensuite la distance entre le pixel reçu par son voisin de gauche et le centre qu'il mémorise. Si cette distance est supérieure à Din, il émet vers son voisin de droite les informations reçues de son voisin de gauche. Dans le cas contraire, il émet la nouvelle distance calculée et son numéro de classe. Ainsi, on récupère à l'extrémité droite du réseau, pour chaque pixel, l'indice de la classe qui a produit une distance minimale. Le programme d'un processeur j se résume par l'algorithme suivant :

```
– boucle :
    – lire (Pix, Din, Kin)
    – D = distance (Pix, Cj)
    – si D > Din
          alors Dout = Din
                Kout = Kin
          sinon Dout = D
                Kout = j
    – écrire (Pix, Dout, Kout)
```

Le réseau reçoit donc un flot continu de pixels de la machine hôte et renvoie, pour chaque pixel le numéro de la classe où une distance minimale a été trouvée afin de mettre à jour les centres des classes. Les processeurs doivent alors être réactualisés en conséquence. Cela se fait en insérant dans le flot de pixels, les centres des classes. Un pixel a la même structure qu'un centre,

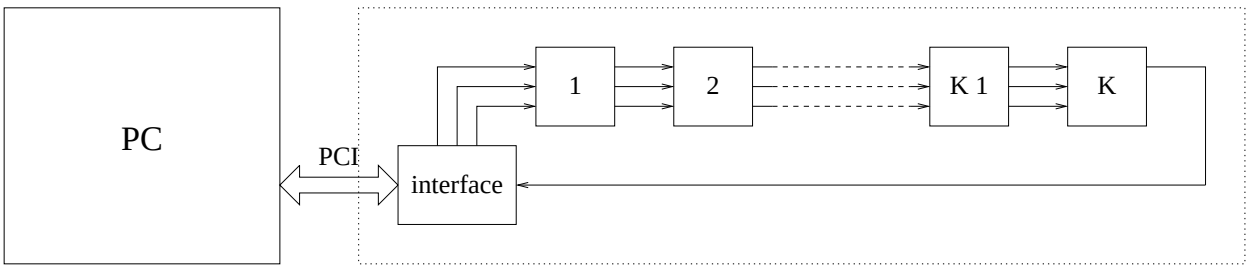


Figure 2. – Architecture parallèle : elle est composée d'un réseau systolique linéaire de *K* processeurs implémenté sur la carte reconfigurable. Seul, le calcul des distances entre les pixels et les centres est parallélisé sur le réseau. Tous les autres calculs sont effectués sur le PC. La communication entre le PC et le réseau s'établit *via* le bus PCI.

c'est un vecteur de NB_BANDES. Il faut simplement différencier ces deux données pour leur appliquer un traitement approprié. Lorsqu'un centre est reçu, il sert uniquement à réactualiser le processeur et aucun calcul de distance n'est fait. Le programme d'un processeur devient :

```
- boucle :
   - lire (PixCenter, Din, Kin, flag)
   - si flag == centre
        alors              // actualisation du centre
          - si Kin == j
               alors Cj = PixCenter
          - ecrire (PixCenter, Din, Kin, flag)
        sinon              // calcul distance
          - D = distance (PixCenter, Cj)
          - si D > Din
        alors Dout = Din
              Kout = Kin
        sinon Dout = D
              Kout = j
          - ecrire (PixCenter, Dout, Kout, flag)
```

Par rapport au programme précédent, une donnée supplémentaire (flag) est reçue. Elle indique si la donnée PixCenter est un centre ou un pixel. Comme cette donnée traverse le réseau, il faut indiquer, lorsqu'il s'agit d'un centre, quel est le processeur concerné par la mise à jour. Dans le programme ci-dessus, cette information transite sur le canal Kin/Kout.

3.2. estimation des performances

Le gain apporté par une carte reconfigurable peut être estimé comme le rapport entre le temps d'exécution de l'algorithme du *K-means* avec et sans carte. L'accélération (Acc) est donnée par :

$$Acc = \frac{T_{hote-dist} + T_{centre}}{T_{carte-dist} + T_{centre}} \tag{3}$$

$T_{hote-dist}$ est le temps pour calculer les distances sur la machine hôte, $T_{carte-dist}$ le temps sur la carte reconfigurable pour effectuer le même calcul, et T_{centre} le temps (également sur la machine hôte) pour mettre à jour les centres des classes. Si on note $\alpha = T_{centre}/T_{hote-dist}$ le rapport entre le temps de calcul des centres et le temps de calcul des distances, et $Acc_{dist} = T_{hote-dist}/T_{carte-dist}$ l'accélération du calcul des distances, l'accélération globale du système se récrit :

$$Acc = \frac{1+\alpha}{1/Acc_{dist} + \alpha} \tag{4}$$

Dans un premier temps, considérons l'accélération Acc_{dist} introduite par la parallélisation des distances :

$$Acc_{dist} = \frac{T_{hote-dist}}{T_{carte-dist}} = \frac{\text{NB\_ITERS} \times \text{NB\_PIXELS} \times K \times Td_p}{\text{NB\_ITERS} \times \text{NB\_PIXELS} \times T_{cycle}} \tag{5}$$

K est le nombre de classes, NB_ITERS le nombre d'itérations, NB_PIXELS le nombre de pixels de l'image, Td_p le temps sur la machine hôte pour calculer la distance entre un pixel et un centre, et T_{cycle} le temps de cycle d'un processeur du réseau pour effectuer ce même calcul. On en déduit immédiatement que :

$$Acc_{dist} = \frac{K \times Td_p}{T_{cycle}} \tag{6}$$

En fait, la réalité est un peu plus complexe car dans l'équation ci-dessus on suppose que le réseau est alimenté de manière continue. En d'autres termes, on fait l'hypothèse que la connexion entre la machine hôte et la carte reconfigurable est capable de soutenir une bande passante permettant un temps de cycle T_{cycle} optimal : on suppose que les données sont systématiquement présentes à l'entrée du réseau et que celui-ci fonctionne sans interruption. La plupart du temps, ce n'est pas le cas. La capacité de transfert (la bande passante) entre l'hôte et les périphériques est inférieure à la capacité d'absorption du réseau. Par conséquent, il faut mieux considérer la capacité de transfert de cette connexion plutôt que les performances maximales du réseau. Cela se traduit par la modification suivante :

$$Acc_{dist} = \frac{K \times T_{de} \times \text{NB\_BANDES}}{\text{NB\_BANDES}/BP} = \frac{K \times BP}{MDE} \tag{7}$$

BP est la bande passante exprimée en Méga octet par seconde et MDE est le nombre de millions de distances élémentaires calculées par seconde sur l'hôte. T_{de} correspond au temps pour effectuer un calcul de distance élémentaire. On a l'équivalence $Td_p = T_{de} \times \text{NB\_BANDES}$: le temps pour calculer la distance entre un pixel et un centre est égale au temps de calcul élémentaire multiplié par le nombre de bandes. On remarque que l'accélération du calcul des distances est uniquement fonction de 3 paramètres : (1) la bande passante (BP) entre la machine hôte et la carte reconfigurable, (2) la puissance de la machine hôte (MDE) et (3) le nombre de classes (K).

Il faut noter que nous ne prenons pas en compte le transfert des résultats du réseau vers le PC. Ce temps est, en fait, négligeable car pour tout transfert d'un pixel vers le réseau, soit NB_BANDES valeurs, on ne renvoie qu'une valeur correspondant à la classe avec laquelle ce pixel est le plus proche. Il y a donc un rapport de 1 à 224 entre les deux flux d'information. De même, le temps passé pour initialiser le réseau n'est pas pris en compte. Ce temps peut être négligé car le rapport entre la durée de l'initialisation et la durée du régime permanent est égal à :

$$\frac{K}{B \times \text{NB\_BANDES}}$$

Rappellons que B est le nombre de pixels qui sont traités entre chaque remise à jour des centres (*cf.* section 2.1) et NB_BANDES le nombre de spectres d'une image. Dans la pratique ce rapport est extrêmement faible et justifie son omission dans notre évaluation.

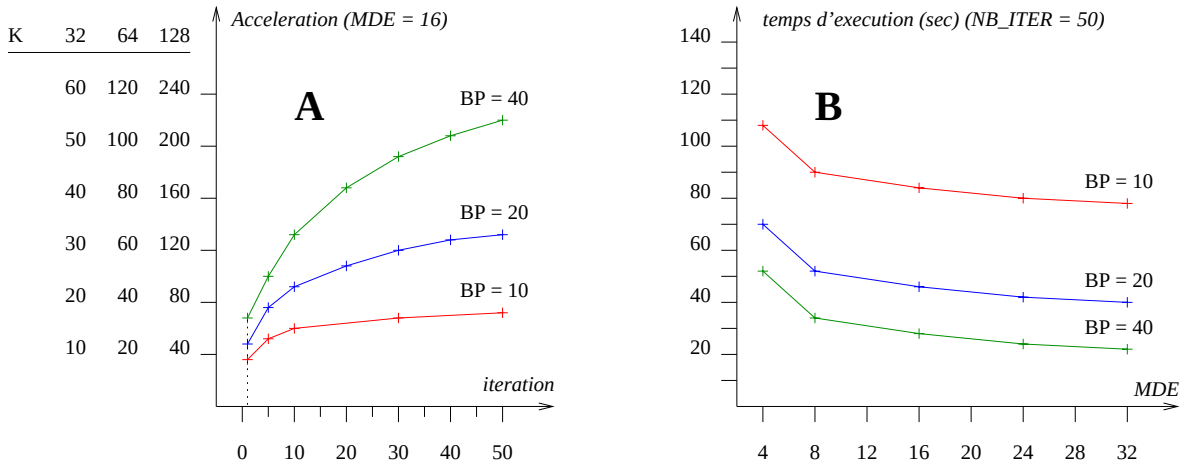


Figure 3. – Accélération et temps d'exécution. Les premiers points sur le graphe A correspondent à la première itération.

Passons maintenant au paramètre α qui exprime le rapport entre le calcul des centres et le calcul des distances. La figure 1 donne le pourcentage de mouvement des pixels entre classes en fonction de l'avancement de l'exécution, symbolisé par le nombre d'itérations effectuées. Une approximation de cette courbe est la fonction inverse $f(x) = 1/x$ qui permet d'estimer le nombre total de centres mis à jour, et donc le temps passé dans cette tâche :

$$T_{centre} = 2 \times \text{NB\_PIXELS} \times \sum_{i=1}^{\text{NB\_ITERS}} \frac{1}{\text{NB\_ITERS}} \times T_{maj} \quad (8)$$

T_{maj} est le temps de mise à jour d'un centre. Le facteur 2 exprime le fait que pour un pixel qui bouge, deux centres sont à réactualiser. Le terme α devient :

$$\alpha = \frac{T_{centre}}{T_{hote-dist}} = \frac{2 \times \text{NB\_PIXELS} \times \sum_{i=1}^{\text{NB\_ITERS}} \frac{1}{\text{NB\_ITERS}} \times T_{maj}}{\text{NB\_PIXELS} \times \text{NB\_ITERS} \times K \times Td_p} \quad (9)$$

Le calcul d'une distance ou d'une mise à jour d'un centre étant de même complexité $Td_p = T_{maj}$, α se simplifie :

$$\alpha = \frac{2 \times \sum_{i=1}^{\text{NB\_ITERS}} \frac{1}{\text{NB\_ITERS}}}{\text{NB\_ITERS} \times K} \quad (10)$$

NB_ITERS	$K = 16$	$K = 32$	$K = 64$	$K = 128$
1	0.125	0.062	0.031	0.016
5	0.057	0.029	0.014	0.007
10	0.037	0.018	0.009	0.005
50	0.011	0.006	0.003	0.001

α est fonction du nombre d'itérations et du nombre de classe. Le tableau ci-dessous donne une idée de ces valeurs.
Nous pouvons maintenant récrire l'accélération globale (équation (4)) en remplaçant les termes Acc_{dist} (équation (7)) et α (équation (10)). Pour simplifier, nous définissons une fonction f_α qui donne la valeur de α en fonction du nombre d'itérations et du nombre de classes. L'accélération globale est alors définie par :

$$Acc(\text{NB\_ITERS}, K) = \frac{1 + f_\alpha(\text{NB\_ITERS}, K)}{\frac{MDE}{K \times BP} + f_\alpha(\text{NB\_ITERS}, K)} \quad (11)$$

Les courbes de la figure 3-A donnent une estimation de l'accélération pour diverses valeurs du nombre de classes (K) et de la bande passante (BP) entre la machine hôte et la carte reconfigurable. Le paramètre MDE (Millions de Distances Elémentaires calculés par seconde) est fixé a 16, ce que nous avons mesuré sur un processeur Intel Pentium-3 cadencé à 600 MHz. Ces courbes montrent plusieurs choses :

• les effets de la bande passante : plus le débit est important, mieux le réseau est utilisé. L'optimal est atteint lorsque la bande passante entre la machine hôte et la carte correspond à la quantité de données que peut absorber le réseau ;
• la linéarité parfaite du facteur d'accélération en fonction du nombre de classes : l'accélération double lorsque le nombre de classes double ;
• l'influence du nombre d'itérations : plus il y a d'itérations, plus le rapport entre la mise à jour des centres et le calcul des distances devient faible. L'essentiel des calculs étant des calculs de distance, ils sont parallélisés sur la carte reconfigurable.

Il est également intéressant d'examiner l'influence de la puissance de calcul de la machine hôte (paramètre MDE) sur le système complet. Le temps d'exécution $T_{carte-dist} + T_{centre}$ est égal à :

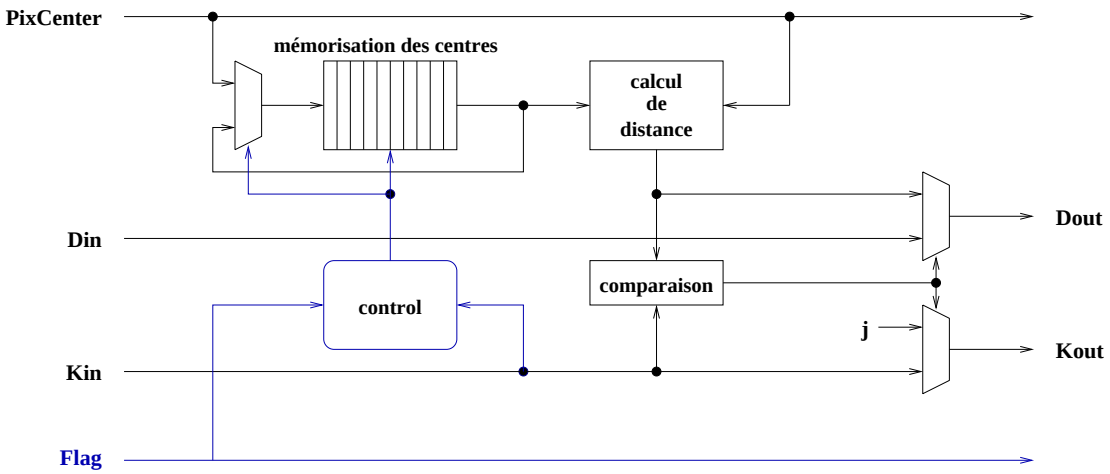


Figure 4. – Schéma de principe d'un processeur du réseau.

$$\frac{\text{NB\_ITERS} \times \text{NB\_PIXELS} \times \text{NB\_BANDES}}{BP} + \text{NB\_PIXELS} \times f_\alpha(\text{NB\_ITERS}, K) \times T_{maj} \quad (12)$$

En considérant toujours que $T_{maj} = Td_p$ (les deux calculs sont de même complexité) on obtient un temps d'exécution :

$$\text{NB\_PIXELS} \times \text{NB\_BANDES} \times \left(\frac{\text{NB\_ITERS}}{BP} \times \frac{f_\alpha(\text{NB\_ITERS}, K)}{MDE} \right) \quad (13)$$

La figure 3-B indique les temps d'exécution (en secondes) du *K-means* pour différentes valeurs de MDE et de BP. Le nombre d'itérations a été fixé à 50 et la taille de l'image hyperspectrale à $256 \times 256 \times 224$. On remarque clairement que le temps d'exécution est principalement lié à la valeur de la bande passante plutôt qu'à la puissance de calcul de la machine hôte. Autrement dit, on préférera une machine hôte modeste avec une connexion rapide vers les périphériques, plutôt que l'inverse.

4. expérimentation

L'architecture systolique présentée à la section 4 a été implantée sur une carte reconfigurable du commerce, la carte Spyder [15]. Cette carte inclut un composant FPGA de la famille Virtex de Xilinx, un XCV800, et une mémoire (non utilisée dans ce projet) de 2 Moctets. La communication avec la machine hôte est établie à l'aide d'une connexion PCI. La carte se loge donc dans un emplacement PCI standard d'un PC.

La figure 4 donne le schéma de principe d'un processeur. Un cycle systolique de base est composé de 224 cycles d'horloge, 224 étant le nombre de bandes spectrales des images. Le vecteur de 224 valeurs représentant le centre de la classe est stocké dans un registre à décalage de même taille. À chaque cycle d'horloge, une valeur est lue et écrite dans ce registre. Si la donnée présente sur le canal PixCenter est un pixel, la valeur lue en sortie est récrite en entré. S'il s'agit d'un centre et que la valeur présente sur l'entrée Kin correspond au numéro du processeur (*j*), alors le registre à décalage est actualisé. À chaque cycle d'horloge également, un calcul de distance est effectué entre 2 composantes d'un pixel et du centre, puis accumulé. En fin de cycle systolique, la distance calculée est comparée à celle produite par le voisin de gauche (Din). Si elle est inférieure, alors elle est émise vers le processeur de droite avec l'indice du processeur ; sinon, les valeurs lues en entrées ont reproduites en sortie.

Le contrôle du réseau est distribué. Le canal identifié par *flag* sur le schéma de la figure 4 véhicule une information codée sur plusieurs bits, et qui est propagée avec les données. Un processeur sait alors s'il s'agit d'un pixel ou d'un centre, s'il a affaire à la première ou à la dernière composante d'un vecteur, si la donnée est valide ou non, *etc*. En fonction de cette information, le processeur décide du calcul à effectuer.

Cette architecture, décrite en VHDL et complètement générique, est cependant optimisée pour la famille Virtex de Xilinx. La mémorisation du centre de la classe dans un registre à décalage peut paraître curieuse par rapport à l'usage d'une mémoire RAM. En fait, les blocs logiques reconfigurables des Virtex sont optimisés pour ce type d'opération. Dans le cas présent, on évite un adressage mémoire, ce qui se traduit par un gain en termes de ressources (pas de décodage nide génération d'adresses). Même optimisée pour la famille Virtex deXilinx, cette description a cependant été synthétisée avec succès sur un composant Altera APEX 20K200 [2].

Après synthèse, un processeur tient dans environ 110 slices (unité reconfigurable élémentaire de la famille Virtex de Xilinx) d'un Virtex XCV800 qui en compte 9408. La distance calculée entre un centre et un pixel est une distance de Manhattan : elle ne demande qu'un simple additionneur et un mécanisme peu

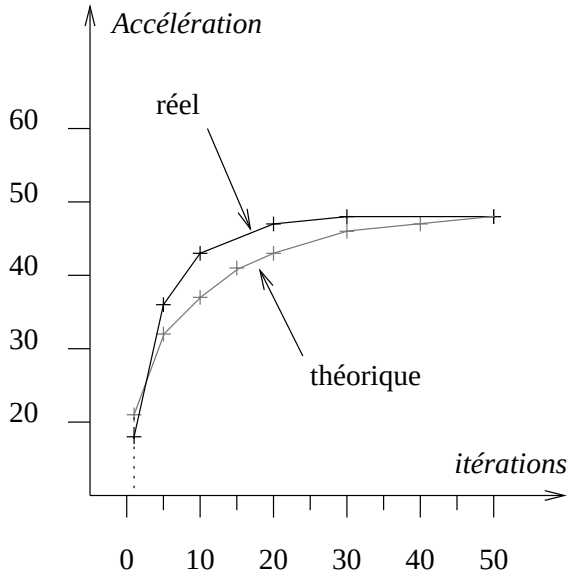


Figure 5. – Accélération estimée par rapport à l' accélération mesurée

coûteux d'inversion de signe pour calculer la valeur absolue. Globalement, un réseau de 64 processeurs avec les mécanismes d'interface du bus PCI occupe un peu moins de 80 % des ressources disponibles. Une fréquence de 35 MHz est obtenue facilement, mais pourrait sans problème être augmentée. L'horloge de la carte Spyder étant synchronisée sur celle du bus PCI (33 MHz), nous n'avons pas cherché à l'optimiser davantage.
La bande passante entre la machine hôte et la carte Spyder est d'environ 14.7 Moctets/seconde et, comme nous l'avons déjà mentionné, la puissance de calcul, évaluée en nombre de distances élémentaires calculées par seconde, est de 16 millions. Une version purement séquentielle de l'algorithme du *K-means* a été implémentée sur le processeur hôte et comparée avec la version accélérée. L'accélération est d'environ 45 (1 minute contre 3/4 d'heure). Les résultats sont présentés sur la figure 5 ($K = 64$) et comparés par rapport aux estimations. On constate une différence au cours des premières itérations, différence probablement due à l'approximation pessimiste du nombre de pixels qui transitent d'une classe à l'autre (*cf.* figure 1) pendant les premières itérations.
Le gain obtenu doit cependant être nuancé par l'effort consenti sur la mise en œuvre matérielle vis-à-vis de l'implémentation logicielle (langage C) qui, elle, n'a pas fait l'objet d'étude approfondie. La structure du code est proche de celle présentée à la section 2 et n'incorpore pas de technique logicielle telle que le déroulage des boucles par exemple. Par contre, les temps d'exécution sur un système d'exploitation LINUX ou Windows NT sont très similaires

5. conclusion

Nous avons présenté une implémentation parallèle de l'algorithme du *K-means* et sa mise en œuvre appliquée aux images hyper-spectrales sur une carte reconfigurable équipée d'un unique composant FPGA, un Virtex XCV800 de Xilinx. Les performances dépendent principalement de la taille du réseau et de la capacité à transférer rapidement les images vers la carte reconfigurable.
L'architecture du réseau est extensible (*scalable*), le nombre de classes qui peut être traité en parallèle dépend uniquement des ressources disponibles. L'arrivée sur le marché de composants FPGA de dernière génération (Virtex-II par exemple) d'une capacité de 5 à 6 fois supérieure indique ce que l'on peut espérer dans un proche avenir. Toutefois, si le nombre de classes que l'on veut pouvoir traiter est supérieur aux ressources reconfigurables, le calcul peut aisément être partitionné. Dans ce cas, un processeur est séquentiellement affecté à plusieurs classes et doit être réinitialisé en conséquence. En fait, cette réinitialisation obligatoire de tout le réseau est quasiment gratuite car, de toute façon, le principe même de l'algorithme du *K-means* exige que les centres des classes soient régulièrement mis à jour. Le surplus de calcul (*overhead*) dû au partitionnement est donc extrêmement faible.
La bande passante entre la machine hôte et la carte reconfigurable est, comme on l'a vu, un facteur déterminant sur les performances du système complet. Dans notre expérimentation, elle représente la moitié de ce qu'est capable d'absorber le réseau : celui-ci peut traiter une composante d'un pixel à chaque coup d'horloge (33 MHz) alors que le canal en provenance de l'hôte ne peut écouler qu'une donnée tous les 2 coups d'horloge (16.7 Moctets/sec). Le réseau est sous-utilisé et conduit à diviser par deux les performances par rapport à l'optimal. L'idéal serait de stocker l'image dans des mémoires directement connectées au composant FPGA. Malheureusement, pour notre application, cela exige une capacité de stockage bien supérieure à ce que propose la grande majorité des cartes reconfigurables. Une autre manière d'augmenter la bande passante est de compresser l'information. La nature des images hyper-spectrales fait qu'il existe une forte corrélation entre les spectres voisins, ce qui entraîne d'excellents taux de compression. Nous étudions actuellement cette possibilité, sachant bien que si cette solution est satisfaisante pour ce cas particulier, elle ne règle pas pour autant les problèmes de communication hôte/périphérique dans le cas général.

BIBLIOGRAPHIE

[1] M. Estlick, M. Leeser, J. Szymanski, J. Theiler, « Algorithmic Transformation in the Implementation of the K-means Clustering on Reconfigurable Hardware », *9th ACM International Symposium on FPGA*, Monterey, CA, 2001.

[2] M. Gokhale, J. Frigo, K. McCabe, J. Theiler, D. Lavenier, « Early Experience with a Hybrid Processor: K-Means Clustering », *First Internationnal Conference on Engineering of Reconfigurable Systems and Algorithms*, Las Vegas, Nevada, 2001.

[3] M. Leeser, J. Theiler, M. Estlick, V. Kitaryeva, J. Szymanski, « Effect of data truncation in an implementation of pixel clustering on a custom computing machine », *SPIE Proceedings, Reconfigurable Technology: FPGA for Computingand Applications*, vol 4212, Boston, MA, 2000.

[4] M. Leeser, J. Theiler, M. Estlick, J. J. Szymanski, « Design tradeoffs in a hardware implementation of the k-means clustering algorithm », *First IEEE SensorArray and Multiprocessing Workshop*, 2000.

[5] J. Theiler, M. Leeser, M. Estlick, J. Szymanski, « Design issues for hardware implementation of an algorithm for segmenting hyperspectral imagery », *Imaging Spectrometry, SPIE Proceedings*, vol 4132, San Diego, CA, 2000.

[6] D. Lavenier, « FPGA implementation of the K-means Clustering Algorithm for Hyper-Spectral Images », *rapport de recherche Los Alamos National Laboratory*, LAUR 00-3079,2000[1].

[7] D. Lavenier, « An FPGA systolic array Using Pseudo Random Bit Generator for Computing Goldbach Partitions », *The VLSI Journal*, vol 30, n° 1, 2000.

[8] S. Guccione, « List of FPGA-based Computing Machines », http://www.io.com/~guccione/HW_list.html, 1999.

[9] J. Theiler, G. Gisler, « A contiguity-enhanced k-means clustering algorithmfor unsurpervised multispectral images segmentation », *Proc SPIE 3159*, pp. 108-118, 1997.

[10] S. Rubini, D. lavenier, « Les Architectures Reconfigurables », *Calculateurs Parallèles*, vol 9, n° 1, 1997.

[11] K. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. Touati, P. Boucard, « Programmable Active Memories: the Comming of Age », *IEEE Trans. onVLSI*, vol 4, n° 1, pp. 56-69, 1996.

[12] P. Guerdoux-Jamet, D. Lavenier, « Systolic Filter for Fast DNA Similarity Search », ASAP'95, *International Conference on Application Specific Array Processor*, Strasbourg, France, 1995.

[13] J. MacQueen, « Some methods for classification and analysis of multivariate observations », *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vo1, University of California Press, 1967.

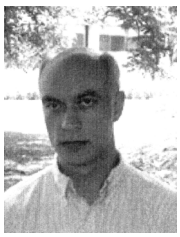
[14] Programmable Logic Boards, The Programmable Logic Jump Station, Optimagic http://www.optimagic.com/boards

[15] Spyder, User's Manual, X2E, (http://www.x2e.de), 1999.

Manuscrit reçu le 8 juin 2001

L'AUTEUR

Dominique LAVENIER



Dominique Lavenier est chargé de recherche au CNRS à l'IRISA (Institut de Recherche en Informatique et Systèmes Aléatoires) à Rennes. Il est docteur de l'Université de Rennes 1 (1989) et a passé son HDR en 1997. Il a travaillé une année (aôut 1999 – août 2000) au Los Alamos National Laboratory, Nouveau Mexique, USA, dans la division NIS (Nonproliferation and International Security). Ses recherches portent sur la conception de circuits VLSI, les architectures reconfigurables, le parallélisme, le traitement des images et la biologie moléculaire. Il a reçu la médaille de bronze du CNRS en 1992 et a été lauréat du prix Cray en 1996 dans la catégorie algorithme, architecture et micro-électronique.

1. Disponible sur les pages WEB personnelles de l'auteur sur le site http://www.irisa.fr ou sur simple demande à lavenier@irisa.fr