Implémentation parallèle d'une méthode d'estimation des cartes d'abondances en imagerie hyperspectrale

Maxime LEGENDRE[1], Saïd MOUSSAOUI[1], Frédéric SCHMIDT[2], Jérôme IDIER[1]

[1]IRCCYN, CNRS UMR 6597, 1, rue de la Noë, BP 92101, F-44321 Nantes Cedex 03, France

[2]IDES, CNRS UMR 8148, Batiment 509, F-91405 Orsay Cedex, France

maxime.legendre@irccyn.ec-nantes.fr, said.moussaoui@irccyn.ec-nantes.fr, frederic.schmidt@u-psud.fr, jerome.idier@irccyn.ec-nantes.fr

Résumé – L'estimation des cartes d'abondance en imagerie hyperspectrale passe par la résolution un problème d'optimisation de grande taille sous contraintes de positivité et d'additivité. Il a été montré récemment qu'une méthode primale-duale de points intérieurs permet de résoudre efficacement ce problème. Malgré une convergence relativement rapide, le temps de calcul reste un problème majeur pour les images de grande taille. Nous proposons une optimisation de l'implémentation GPU (Processeurs de Carte Graphique) de cette méthode. La discussion porte sur la différence entre une parallélisation de l'algorithme entier, ou de chaque étape séparément, la façon d'utiliser au mieux le CPU et le GPU, ainsi que la meilleur manière d'organiser les données en mémoire. Les effets de ces modifications sur le temps de calcul sont illustrés grâce à des images simulées. Une application impliquant les données de la mission européenne d'exploration planétaire Mars Express est présentée.

Abstract – Abundance maps estimation in hyperspectral imaging requires the resolution of a large constrained optimization problem. Recently, a primal-dual interior-point algorithm has been proven to solve efficiently this problem. Despite its relatively fast convergence, the computation time remains a major issue for large scale images. We propose here an optimization of the GPU (Graphics Processing Unit) implementation of this method. The discussion covers the difference between parallelizing the entire algorithm or every step separately, the way of using the best of both the GPU and the CPU, and the memory organization issues. Simulated data are used to illustrate the effect of these modifications on computing time. An extensive application involving a large set of data from the European planetary mission Mars Express is presented.

1 Introduction

L'imagerie hyperspectrale consiste en la collecte et l'exploitation du spectre de réflectance de la lumière renvoyée par chaque point d'une surface, mesuré dans plusieurs bandes spectrales contigües. Cette technique permet d'accéder à des informations liées à la composition de la surface observée [1, 2]. On suppose que chaque pixel d'une image hyperspectrale correspond à une surface composée de plusieurs matériaux caractérisés par leur spectre de réflectance, le spectre observé est donc issu du mélange des spectres purs de chaque constituant. Le démixage spectral consiste à identifier ces constituants ainsi que leurs proportions dans chaque pixel. Pour cela, on utilise couramment le modèle de mélange linéaire qui suppose que le spectre observé est une combinaison linéaire des spectres purs, pondérés par des coefficients de mélange, appelés abondances, correspondant aux proportions recherchées.

Ainsi, si on considère une image hyperspectrale composée de N pixels acquise dans K bandes spectrales, le spectre du $n^{\text{ème}}$ pixel peut être exprimé sous la forme :

$$\boldsymbol{x}_n = \sum_{p=1}^{P} \boldsymbol{s}_p c_{n,p} + \boldsymbol{\epsilon}_n \tag{1}$$

où P est le nombre de constituants, $\boldsymbol{s}_p \in \mathbb{R}^K$ est le spectre carctéristique du $p^{\text{ème}}$ composant, $c_{n,p}$ est l'abondance du $p^{\text{ème}}$ constituant dans le $n^{\text{ème}}$ pixel, et $\boldsymbol{\epsilon}_n \in \mathbb{R}^K$ est un bruit additif gaussien, i.i.d. spatialement et spectralement et de moyenne nulle.

L'image entière peut être représentée sous une forme matricielle par :

$$\boldsymbol{X} = \boldsymbol{S}\boldsymbol{C} + \boldsymbol{E} \tag{2}$$

où $\boldsymbol{X} \in \mathbb{R}^{K\times N}$ représente l'ensemble des pixels observés, $\boldsymbol{S} \in \mathbb{R}^{K\times P}$ est l'ensemble des spectres purs des constituants de l'image, $\boldsymbol{C} \in \mathbb{R}^{P\times N}$ contient les coefficients d'abondance, et $\boldsymbol{E} \in \mathbb{R}^{K\times N}$ les bruits associés aux observations. L'objectif du démixage spectral est alors d'estimer $\boldsymbol{S}$ et $\boldsymbol{C}$ à partir de $\boldsymbol{X}$.

Considérons le cas où les spectres des constituants ont été obtenus par une méthode d'extraction des pôles de mélange telle que N-FINDR [3] ou VCA [4], ou fixés dans une bibliothèque. Le problème d'estimation des cartes d'abondances $\boldsymbol{C}$ peut alors s'écrire comme un problème d'optimisation visant à minimiser le critère :

$$F = \sum_{n=1}^{N} \|\boldsymbol{x}_n - \boldsymbol{S}\boldsymbol{c}_n\|_2^2 \tag{3}$$

sous les contraintes

$$c_{n,p} \geqslant 0, \quad \forall\, n = 1, \ldots, N, \quad \forall\, p = 1, \ldots, P, \tag{4a}$$

$$\sum_{p=1}^{P} c_{n,p} = 1, \quad \forall\, n = 1, \ldots, N. \tag{4b}$$

Dans ce travail, nous nous focalisons sur la méthode primale-duale de points intérieurs présentée dans [5]. Les principales étapes de cette méthode sont résumées dans la figure 1. Il s'agit d'une méthode itérative visant à satisfaire les conditions d'optimalités de Karush-Kuhn-Tucker (KKT) portant sur les variables primales et duales. A l'itération k, un algorithme d'optimisation est mis en œuvre pour s'approcher des conditions KKT perturbées par un paramètre μ_k, appelé paramètre barrière. La suite des paramètres barrière tend vers 0. Le problème d'optimisation à l'itération k est résolu en effectuant une itération de Newton constituée d'un calcul de directions et d'une recherche de pas par une stratégie de rebroussement respectant la règle d'Armijo et assurant le respect des contraintes sur les variables primales et duales.

Il a été montré dans [5] qu'un des avantages de cette méthode est sa rapidité d'exécution, en particulier par rapport à la méthode de réference FCLS [6], ainsi qu'une méthode plus récente comme ADMM [7]. Cependant le temps d'analyse d'une image de grande taille reste important et le travail présenté dans cet article consiste à le réduire en utilisant au mieux la puissance de calcul des GPU (*Graphics Processing Units*, ou Processeur de cartes graphiques).

Un GPU est un processeur contenant plusieurs centaines d'unités de calcul pouvant effectuer de façon parallèle les mêmes opérations sur des données mémoire différentes. Les GPU sont de plus en plus utilisés dans le milieu scientifique car ils offrent une puissance de calcul importante pour un prix raisonable [8]. Récemment, plusieurs méthodes d'analyse d'images hyperspectrales on fait l'objet d'études quant à leur implémentation parallèle [9], à la fois sur GPU [10] ou sur FPGA [11].

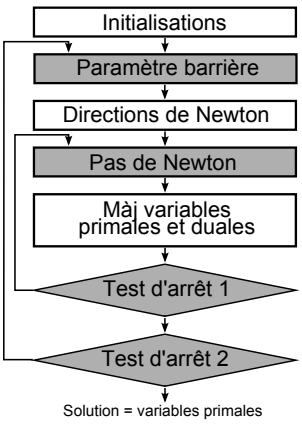

FIGURE 1 – Organigramme de la méthode primal-dual de points intérieurs

2 Implémentation GPU

Le GPU se distingue du CPU par le grand nombre d'Unités Arithmétiques et Logiques (UAL) qu'il comporte, souvent plusieurs centaines contre 2 ou 4 pour un CPU. Ces unités effectuent nécéssairement les mêmes instructions sur des données différentes, suivant le modèle SIMD (*Single Instruction Multiple Data*). L'implémentation réalisée utilise la technologie CUDA développée par Nvidia [12]. Celle ci se présente comme une extension du language C, appellée C pour CUDA. Elle permet d'exécuter des portions de code appelés noyaux de façon parallèle sur un GPU. Un noyau est exécuté à travers un grand nombre de *threads*. Selon ce modèle, les *threads* sont autant de processus indépendants réalisant la suite d'instructions définie dans le noyau sur des données différentes. Un *thread* est donc exécuté sur une UAL. Les *threads* étant souvent bien plus nombreux que les UAL disponibles, le modèle CUDA dispose de règles d'exécution permettant d'optimiser leur exécution.

Nous comparons dans ce travail deux stratégies d'implémentation parallèle de la méthode primale-duale de points intérieurs.

2.1 Implémentation *par pixel*

Il est naturel de penser que c'est en rendant le problème totalement parallélisable que le GPU sera le mieux exploité. Cela est possible en remarqant que la minimisation du critère défini à l'équation 3 est équivalente à N minimisations indépendantes des critères liés aux N pixels :

$$F_n = \|\boldsymbol{x}_n - \boldsymbol{S}\boldsymbol{c}_n\|_2^2 \tag{5}$$

De cette façon, N *threads* peuvent être lancés sur le GPU, chacun exécutant entièrement l'algorithme d'estimation des abondances sur un pixel de l'image. Néanmoins cette méthode présente un inconvénient. En effet, d'après [13], les structures conditionnelles de type *« if, then, else »* au sein d'un noyau sont à éviter. Dans le modèle défini par CUDA, les *threads* sont organisés par groupes de 32, appelés *warps*, se comportant comme autant d'unités SIMD. En cas de structure conditionnelle, si les deux conditions sont vérifiées par des *threads* différents d'un même *warp*, alors les instructions liées aux deux conditions sont exécutées par tous les *threads* de ce *warp*. Des opérations inutiles sont alors effectuées bien que leurs résultats soient ignorés, ce qui a pour effet d'augmenter le temps d'exécution.

Ainsi, lors de l'exécution de l'algorithme de points intérieurs, le temps total est fixé par le pixel dont le traitement nécéssite le plus d'itération dans chaque groupe de 32.

2.2 Implémentation *par image*

L'idée est de revenir au problème initial et de minimiser le critère global défini par l'équation 3 en tirant parti au mieux des caractéristiques différentes du CPU et du GPU. Ainsi, le CPU est utilisé pour implémenter la structure de l'algorithme, contenant les conditions d'arrêt. Le GPU est utilisé au sein de chaque étape afin d'en accélérer l'exécution. Il est à noter qu'une telle

gestion est indispensable dans le cas où la pénalisation spatiale décrite dans [14] est prise en compte. En effet la non séparabilité du critère empêche l'utilisation de l'implémentation *par pixel*.

Cette version présente l'avantage de ne pas introduire de calculs inutiles, cependant certaines étapes de l'algorithme nécéssitent des transferts de données entre la mémoire du CPU et celle du GPU, ce qui ralentit leur exécution. On peut distinguer deux types d'étapes : celles pour lesquelles une parallélisation totale est possible car elles contiennent des calculs indépendants sur chaque pixel (en blanc sur la figure 1), et celles dont le résultat est une variable unique pour l'image entière (en gris sur la figure1). C'est le cas pour le calcul du paramètre barrière, du pas de Newton, ainsi que des deux critères d'arrêt. Ce type d'étape est appelé *réduction* et n'est effectuée que partiellement sur le GPU. En effet il est plus avantageux pour ce type d'étape d'effectuer une réduction partielle sur le GPU, puis de transférer ce résultat partiel dans la mémoire du CPU pour y terminer le calcul. Le choix du meilleur moment pour la transition résulte d'un compromis entre le temps de transfert mémoire et la rapidité du CPU dans les dernières étapes de calcul. Cette procédure est décrite de façon détaillée dans [13, Chap. 6].

2.3 Organisation de la mémoire

Une organisation optimale de la mémoire est indispensable pour réduire les temps d'accès aux données et donc éviter les temps d'attente. Prenons l'exemple illustré par la figure 2 des variables primales $\boldsymbol{C}$ formant un vecteur de P élements pour chaque pixel. Pour une exécution séquentielle sur le CPU, il est judicieux de regrouper en mémoire les valeurs correspondant à un pixel car elles sont utilisées par des instructions successives. En revanche sur le GPU, à un instant donné, plusieurs *threads* accèdent au même élément du vecteur mais sur des pixels différents. Ces accès sont plus rapides si ces éléments sont situés à des adresses mémoire adjacentes car un seul accès permet de mettre en cache plusieurs adresses mémoires successives. En pratique, cette organisation est adoptée pour toutes les variables dans l'ensemble du programme, aussi bien sur GPU que sur CPU.

Il ne s'agit pas seulement du problème bien connu du stockage des matrices par ligne ou par colonne. En effet certaines variables sont constituées d'une matrice par pixel (c'est le cas du hessien du critère). Dans ce cas, une organisation *par élément* ne correspond ni à un stockage par ligne, ni par colonne.

3 Résultats

3.1 Données simulées

Afin de simuler une image hyperspectrale réaliste, les spectres de réflectances de la librairie USGS (U.S. Geological Survey) sont utilisés [15]. Ces spectres contiennent $K = 224$ bandes spectrales entre 383 nm et 2508 nm. P spectres sont choisis aléatoirement pour créer le mélange synthétique avec des abondances simulées selon une distribution de Dirichlet. Enfin, un bruit Gaussien centré est ajouté à chaque mélange de façon à obtenir un rapport signal sur bruit de 30 dB.

Les calculs sont effectués sur une station de travail Dell Precision T7400 contenant un processeur Intel Xeon X5472 (3GHz) et 16 GB de RAM. Cette station embarque une carte graphique Nvidia Tesla C1060 permettant d'effectuer des calculs en parallèle sur ces 240 coeurs fonctionnant à 1.3 GHz et contenant 4 GB de RAM.

Les performences des implémentations *par pixel* et *par image* de l'algorithme de points intérieurs présenté sont comparées grâce à des images synthétiques de taille $N = 64^2$, simulées avec différents nombres P de spectres purs. Le démixage est effectué en utilisant P spectres purs : soit ceux de la librairie utilisés pour simuler l'image, soit ceux extraits de l'image par la méthode N-FINDR [3]. Pour chaque test, le tableau 1 donne le temps de calcul par pixel ainsi que la norme moyenne du résidu :

$$r = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{K}\left\|\boldsymbol{x}_n - \boldsymbol{S}\boldsymbol{c}_n\right\|_2 . \qquad (6)$$

Endmembers	P	Temps (μs)		r ($\times 10^{-4}$)	
		PXL	IMG	PXL	IMG
LIB	3	**2.64**	3.08	3.14	3.14
	6	**5.42**	5.69	3.38	3.38
	10	**10.2**	**10.2**	3.65	3.65
	15	18.8	**18.5**	3.75	3.76
EEA	3	**2.71**	3.03	3.15	3.14
	6	**5.22**	5.79	3.38	3.39
	10	**10.1**	10.4	3.69	3.70
	15	19.1	**18.5**	3.86	3.86

TABLE 1 – Temps de calcul par pixel et résidu moyen, pour 100 réalisations de Monte Carlo, pour différents nombres de constituants, utilisant les spectres réels (LIB) ou estimés par N-FINDR (EEA) : comparaison des implémentations *par pixel* (PIX) et *par image* (IMG).

Tout d'abord, on peut remarquer que les résidus moyens obtenus à l'issu des deux versions sont très similaires, ce qui confirme que les deux approches aboutissent à une précision équivalente des résultats. Concernant les temps de calcul, les avantages et inconvénients des deux implémentations semblent s'équilibrer pour donner des résultats proches. On peut tout de même noter que la version *par pixel* est la plus rapide pour un faible nombre de constituants, alors que la version *par image* est plus adaptée pour un grand nombre de sources. Cette différence est due à l'utilisation de la stratégie de [13, Chap. 6] dans l'implémentation *par image*. En effet, les transferts de données effectués selon cette stratégie nécéssitent un temps constant, quelle que soit la valeur de P. Pour un problème de petite taille, ce temps de transfert devient important par rapport au reste des opérations, limitant le gain apporté par le calcul parallèle.

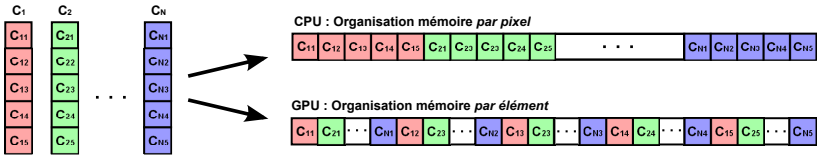

FIGURE 2 – Meilleure organisation de la mémoire sur CPU et sur GPU

3.2 Données réelles

La sonde européenne Mars Express embarque notamment le spectro-imageur OMEGA qui fournit des images hyperspectrales de la surface de Mars. L'importante masse de données ainsi recueillie nécessite des méthodes de traitement rapides qui justifient pleinement l'utilisation de GPU. Ainsi, un jeu de données comprenant 1300 images de OMEGA a été utilisé pour valider les performances de cette implémentation [16]. Alors que l'analyse de ces images nécessitait environ 20 jours de calcul avec la version CPU optimisée de l'algorithme primal-dual de points intérieurs, 2.5 jours suffisent pour obtenir les mêmes résultats sur la même station de travail en utilisant la carte graphique Nvidia Tesla M2050.

4 Conclusion

Dans ce travail, nous proposons deux implémentations parallèles d'un algorithme primal-dual de points intérieurs. Le problème principal est de savoir s'il est plus avantageux de paralléliser entièrement l'algorithme, ou chaque de ses étapes séparément. Bien que la première proposition semble la plus adaptée, nous montrons qu'elle comporte aussi un inconvénient et que dans certaines conditions il est préférable de mettre en oeuvre la seconde. L'utilisation conjointe du CPU et du GPU est dans ce cas nécéssaire pour obtenir un temps de calcul minimal. L'organisation des données en mémoire est également un point important dans la mesure où le schéma optimal est différent pour un programme séquentiel ou parallèle. Enfin, on peut remarquer que la version *par image* est plus adaptée aux travaux à venir car elle peut être modifiée pour prendre en compte des techniques de régularisation spatiale sur les coefficients d'abondance.

Remerciements Ce travail a été co-financé par la région *Pays de la Loire* et *l'Agence Spatiale Européenne*.

Références

[1] N. Keshava et J. F. Mustard, « Spectral unmixing », *IEEE Signal Processing Magazine*, vol. 19, n°1, pp. 44–57, 2002.

[2] C.-I. Chang, *Hyperspectral data exploitation : theory and applications*, Wiley-Interscience, 2007.

[3] A. Plaza et C.-I. Chang, « An improved N-FINDR algorithm in implementation », in *Proc. of SPIE Vol*, 2005, vol. 5806, p. 299.

[4] J. M. Nascimento et J. M. Bioucas-Dias, « Vertex component analysis : A fast algorithm to unmix hyperspectral data », *IEEE Transactions on Geoscience and Remote Sensing*, vol. 43, n°4, pp. 898–910, 2005.

[5] E. Chouzenoux, M. Legendre, S. Moussaoui et J. Idier, « Fast constrained least squares spectral unmixing using primal-dual interior-point optimization », *à paraitre dans IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2013.

[6] D. C. Heinz *et al.*, « Fully constrained least squares linear spectral mixture analysis method for material quantification in hyperspectral imagery », *IEEE Transactions on Geoscience and Remote Sensing*, vol. 39, n°3, pp. 529–545, 2001.

[7] J. M. Bioucas-Dias et M. A. Figueiredo, « Alternating direction algorithms for constrained sparse regression : Application to hyperspectral unmixing », in *Proc. 2nd IEEE WHISPERS*. IEEE, 2010, pp. 1–4.

[8] J. Owens *et al.*, « GPU computing », *Proc. of the IEEE*, vol. 96, n°5, pp. 879–899, 2008.

[9] A. Plaza, J. Plaza, A. Paz et S. Sanchez, « Parallel hyperspectral image and signal processing », *IEEE Signal Processing Magazine*, vol. 28, n°3, pp. 119–126, 2011.

[10] S. Sánchez, A. Paz, G. Martín et A. Plaza, « Parallel unmixing of remotely sensed hyperspectral images on commodity graphics processing units », *Concurrency and Computation : Practice and Experience*, vol. 23, n°13, pp. 1538–1557, 2011.

[11] C. González, J. Resano, A. Plaza et D. Mozos, « FPGA implementation of abundance estimation for spectral unmixing of hyperspectral data using the image space reconstruction algorithm », *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 5, n°1, pp. 248–261, 2012.

[12] C. Nvidia, « NVIDIA CUDA C programming guide version 4.2 », 2012.

[13] D. Kirk *et al.*, *Programming massively parallel processors : a hands-on approach*, Morgan Kaufmann, 2010.

[14] S. Moussaoui, E. Chouzenoux et J. Idier, « Spectral unmixing using fully constrained penalized least squares estimation and primal-dual interior point optimization », in *Proc. 4th IEEE WHISPERS*, 2012.

[15] R. N. Clark, G. A. Swayze, A. Gallagher, T. V. King et W. M. Calvin, « The U.S. geological survey digital spectral library : version 1 : 0.2 to 3.0 μm », *U.S. Geological Survey, Denver, CO, Open File Rep. 93-592*, 1993.

[16] M. Legendre, L. Capriotti, F. Schmidt, S. Moussaoui et A. Schmidt, « GPU implementation issues for fast unmixing of hyperspectral images », in *EGU General Assembly Conference Abstracts*, 2013, vol. 15, p. 11686.