Architecture GPU pour radar de surveillance spatial et pour radar aéroporté

Jean-François Degurse, Bruno Dugrosprez, Sylvie Marcos, Laurent Savy, Jean-Philippe Molinié

Architecture GPU pour un radar de surveillance spatial et pour radars aéroportés

J-F. DEGURSE[1,2], B. DUGROSPREZ[1], S. MARCOS[2], L. SAVY[1], J-Ph. MOLINIÉ[1]

[1]ONERA, Departement Électromagnétisme et Radar
Chemin de la Hunière, BP 80100, 91123 Palasieau Cedex, France

[2]Laboratoire des Signaux et des Systèmes (Supélec-CNRS-Univ.Paris-Sud)
3, Rue Joliot-Curie, 91192 Gif sur Yvette, France

jean-francois.degurse@onera.fr, bruno.dugrosprez@onera.fr
sylvie.marcos@lss.supelec.fr, laurent.savy@onera.fr

Résumé – L'augmentation de la complexité des systèmes radar ainsi que le traitement du signal associé impliquent un hausse significative de la charge calculatoire. Le challenge pour les radars basés au sol est la réduction du coût de l'infrastructure de calcul alors que pour les radars aéroportés, l'objectif est d'implémenter un traitement lourd, comme le STAP ou le SAR, tout en réduisant la taille du système et la consommation électrique. Dans un exemple concret de radar de surveillance spatiale, nous analysons toutes les étapes du traitement tournant sur CPU et sur GPU, et nous les comparons au système de traitement actuel. Pour l'embarqué, nous présentons les avantages d'un traitement STAP sur GPU comparé un processeur généraliste traditionnel. Nous montrons aussi que l'utilisation d'une configuration originale ARM-GPU augmente encore l'efficacité énergétique par rapport à une architecture classique x86-GPU.

Abstract – The increasing complexity of radar systems and associated signal processing involves a significant increase of the computational workload. The challenge for ground-based radars is to reduce the cost of computing infrastructure whereas for airborne radars, the challenge is to implement intensive signal processing, such as STAP and SAR processing, while reducing hardware size and electrical consumption. In a concrete example of a space surveillance radar, we analyze all the signal processing modules running on CPU and on GPU, then we compare to the current computing system. For embedded applications, we present the advantages of a STAP algorithm running on GPU compared to a traditional general-purpose processor. We also show that using an original ARM processor-GPU configuration leads to an even higher efficiency than the classical x86-GPU architecture.

1 Introduction

L'évolution des systèmes radar et des techniques de traitement du signal a conduit à un forte augmentation du nombre d'opérations à effectuer par les calculateurs. En particulier, les développements de radars multi-capteurs ouvrent la voie à des techniques de traitement offrant des apports de performance importants. Les processeurs de calcul ont heureusement eux aussi évolué et permettent aujourd'hui de mettre en ?uvre des traitements complexes tout en réduisant le coût, la consommation et l'encombrement des machines de calcul à condition de maitriser la parallélisation des algorithmes de traitement sur ces nouvelles architectures. En effet, les systèmes parallèles deviennent omniprésents dans la monde de l'informatique comme en témoigne le développement des processeurs généralistes x86 multi-c?urs (CPU), des processeurs hétérogènes spécialisés comme l'architecture POWER, et plus récemment des processeurs massivement parallèles de traitement graphique (GPU). Ces systèmes parallèles requièrent un niveau de parallélisme de programmation différent pour tirer parti de la puissance de calcul qu'ils proposent. Les processeurs CPU bien qu'évoluant de plus en plus vers une architecture parallèle, à travers l'augmentation du nombre de coeurs de calcul restent intrinsèquement conçus pour traiter des suites d'opérations séquentielles. Ayant pour fonction principale l'affichage de scènes 3D pour des applications ludiques ou CAO, les processeurs graphiques (GPU) ont une architecture massivement parallèle basée sur des centaines d'unités de calcul élémentaires. Depuis quelques années, ils peuvent être utilisés pour effectuer des calculs scientifiques. L'augmentation de leur puissance de calcul à chaque nouvelle génération avec l'augmentation du nombre d'unités de traitement élémentaire est beaucoup plus rapide que celle des processeurs généralistes CPU. Les processeurs GPU offrent aujourd'hui des puissances de calcul théorique 10 à 20 fois supérieures aux processeurs généralistes CPU, pour un coût équivalent.

Nous montrerons, dans la première partie, l'intérêt du GPU pour une application concrète de radar de surveillance de l'espace. Le but est de réduire le coût et la maintenance du système de traitement du signal. Dans la deuxième partie, nous évaluerons l'apport d'un GPU pour un traitement STAP en radar aéroporté. L'objectif sera cette fois d'améliorer l'efficacité énergétique en augmentant la puissance de calcul disponible, tout en maintenant un encombrement et une consommation réduite. Nous

poursuivrons cet objectif avec une analyse des performances d'un traitement STAP sur une plateforme prototype ARM-GPU.

2 Radar sol: surveillance spatiale

2.1 Traitement du signal

Le système GRAVES, développé par l'ONERA en 2005, est le système de surveillance spatiale français aujourd'hui opéré par l'Armée de l'Air [1]. Sa mission consiste à détecter, pister et entretenir une base de donnée des paramètres orbitographiques des satellites passant au dessus du territoire national dans une tranche d'altitude comprise entre 400km et 1000km. Pour cela, il dispose d'un radar bistatique à onde continue opérant dans la gamme de fréquence VHF. Le site de réception est constitué de 100 récepteurs. La durée de mesure pour chaque tranche de signal est d'environ 1 seconde et chaque tranche de signal contient $N = 16384$ échantillons temporels bruts. Le traitement du signal est divisé en 3 étages:

L'étage ***A*** est constitué d'un filtrage de Hilbert visant à transformer le signal analogique reçu par les récepteurs en signal complexe. Cela est fait en appliquant un filtre de type FIR par convolution sur le signal reçu: $y(t) = x(t) \odot h(t)$.
Puis un traitement d'élimination du trajet direct est effectué pour supprimer la composante continue du signal se trouvant à la fréquence d'émission du radar. Le filtre est réalisé par soustraction de la valeur moyenne du signal temporel:
$y'(t_n) = y(t_n) - \frac{1}{N}\sum y(t_n)$. Ces deux traitements sont faits pour l'ensemble des points des 100 voies de réception.

L'étage ***B*** est dédié à la formation de faisceaux par le calcul (digital beamforming). Pour chaque échantillon temporel, plusieurs pointages sont effectués dans K directions, K étant supérieur à 1000. Cette formation de faisceaux est faite en multipliant la matrice de données $\mathbf{Y'}$ par une matrice de pointage $\mathbf{S}$:

$$\mathbf{Z} = \mathbf{S}\mathbf{Y'} \tag{1}$$

avec

$$\mathbf{S} = \begin{pmatrix} s_1^*(\theta_1,\phi_1) & \dots & s_{100}^*(\theta_1,\phi_1) \\ \vdots & \ddots & \vdots \\ s_1^*(\theta_K,\phi_K) & \dots & s_{100}^*(\theta_K,\phi_K) \end{pmatrix}$$

$$\mathbf{Y'} = \begin{pmatrix} y'_1(t_1) & \dots & y'_1(t_n) \\ y'_2(t_1) & \dots & y'_2(t_n) \\ \vdots & \ddots & \vdots \\ y'_{100}(t_1) & \dots & y'_{100}(t_n) \end{pmatrix}$$

$$\mathbf{Z} = \begin{pmatrix} z_1(t_1) & \dots & z_1(t_n) \\ \vdots & \dots & \vdots \\ \vdots & \ddots & \vdots \\ z_K(t_1) & \dots & z_K(t_n) \end{pmatrix}$$

Le résultat est une matrice $\mathbf{Z}$ contenant les K faisceaux de N points temporels L'étage ***C*** est le plus important en terme de volume de calcul. La première étape est un traitement visant à compenser l'étalement Doppler dû à l'accélération radiale des satellites pendant la période d'acquisition. Pour cela, la matrice $\mathbf{Z}$ est multipliée par p hypothèses d'accélération, avec p étant généralement de l'ordre de quelques dizaines: $\mathbf{Z_1} = \mathbf{Z} \odot \mathbf{H_1}$,...,$\mathbf{Z_p} = \mathbf{Z} \odot \mathbf{Z_p}$ où $\odot$ est le produit de Haddamard (point à point) et nous obtenons donc p matrices $\mathbf{Z_i}$. La deuxième partie consiste à passer les signaux temporels en signaux fréquentiels, c'est à dire à appliquer une transformée de Fourier (FFT) sur tous les points temporels pour chaque faisceau de chaque matrice $\mathbf{Z_i}$: $\mathbf{Z_i(t_n)} \Longrightarrow \mathbf{Z_i(f_n)}$.
Enfin la dernière partie du traitement du signal est dévolue à la détection des plots. La puissance spectrale est d'abord estimée et seuillée puis une recherche de maximum est effectuée pour rechercher le faisceau correspondant à un groupe de détection.

2.2 Performances sur GPU

Lors de la mise en place du système, un puissant et couteux calculateur constitué de plus de 100 processeurs PowerPC (PPC) G4 a été assigné au traitement du signal. Ils ont été répartis de la manière suivante: 1 PPC est assigné à l'étage ***A***, 28 PPC à l'étage ***B*** et 91 PPC à l'étage ***C***. Chaque étape doit être réalisée en moins de 1 seconde pour satisfaire la contrainte temps-réel de l'ensemble du radar. Ces algorithmes étant massivement parallèles, nous pouvons espérer de très bonnes performances sur processeur GPU [2]. En effet, pour un fonctionnement optimal, un GPU requiert que les calculs soient partitionnés en un très grand nombre de tâches élémentaires. A l'inverse, un processeur CPU ou un système multiprocesseurs CPU pourra se contenter de tâches beaucoup plus lourdes, chaque coeur de calcul ou processeur étant efficace sur des opérations séquentielles.
Nous avons implémenté les traitements en code C avec les extensions CUDA pour piloter le GPU. Nous comparons les performances sur GPU avec celles obtenues sur un processeur généraliste CPU multicoeurs de même génération et d'un coût similaire. Pour cela, nous avons aussi implémenté ces algorithmes en langage C pur en utilisant les librairies mathématiques *MKL* et *FFTw*.
Le processeur CPU du test est un AMD Opteron 6140 (2,6 GHz, 8 coeurs), et le GPU de test est une carte NVIDIA Tesla C2070 (448 CUDA coeurs, 1,15 GHz), les calculs sont en simple précision (32bit). Les résultats des temps d'exécution pour les étages ***A***, ***B*** et ***C*** sont ici comparés aux performances du système actuel à base de processeurs PowerPC G4 (400 MHz), respectivement 1, 28 et 91 processeurs PPC.

Etage	A	B	C
CPU PowerPC (1/28/91PPC)	698 ms	153 ms	671 ms
CPU x86 AMD Opteron 6140	170 ms	128 ms	2428 ms
GPU NVIDIA Tesla C2070	34 ms	21 ms	250 ms

Table 1: Temps d'éxecution sur CPU, GPU (PC) et sur CPU PowerPC pour différents étages.

3 Radar aéroporté: détection de cibles lentes

3.1 Traitement spatio-temporel adaptatif

Contrairement aux traitements d'antenne classiques qui n'exploitent que la dimension spatiale des signaux reçus pour leurs filtrages, les traitements STAP (*Space Time Adaptive Processing*) sont des traitements qui exploitent conjointement les deux dimensions (spatiale et temporelle) des signaux reçus sur un réseau d'antennes. Cette structure de traitement permet de tirer parti des propriétés spécifiques spatio-temporelles dans le domaine angle-fréquences des signaux reçus. Cette problématique se rencontre en particulier dans le cadre du filtrage des échos reçus par un radar aéroporté en provenance du sol pour lesquels il existe un lien direct entre direction d'arrivée et fréquence Doppler. Les traitements STAP permettent alors de détecter des cibles noyées dans le fouillis à cause de leur faible vitesse. La Figure 1 montre une image vitesse-distance où un convoi ayant une faible vitesse est masqué par le fouillis de sol, il n'est pas détectable. Sur la Figure 2, la même image mais après traitement STAP permet de détecter le convoi. Le détecteur STAP classique, appelé *Adaptive Matched Filter* (AMF), s'écrit de la façon suivante:

$$\mathbf{P}_{AMF} = \frac{|\mathbf{s_s^H \hat{R}^{-1} x_k}|^2}{\mathbf{s_s^H \hat{R}^{-1} s_s}} \underset{H_1}{\overset{H_0}{\lessgtr}} \eta \qquad (2)$$

avec $\mathbf{x_k}$ le vecteur de données à la case distance k, $\mathbf{\hat{R}_k}$ la matrice de covariance représentative des interférences de la case distance k, $\mathbf{s_s^H}$ le vecteur directeur spatio-temporel et η le seuil de détection.
Un traitement STAP peut se décomposer en 4 parties suivantes:

- estimation de la matrice de covariance et *diagonal loading*, pour chaque case distance. La matrice de covariance $\mathbf{\hat{R}_k}$ est estimée sur les points temporels: $\mathbf{\hat{R}_k} = \frac{1}{\mathbf{K_t}} \sum_{\mathbf{i=1}}^{\mathbf{K_t}} \mathbf{x_i x_i^H}$. La méthode du *diagonal loading* permet d'améliorer sensiblement les performances dans le cas où la matrice de covariance est mal estimée: $\mathbf{\hat{R}_k} = \mathbf{\hat{R}_k} + \delta \mathbf{R_b}$ où $\mathbf{R_b}$ est la matrice bruit seul et δ est le facteur de chargement, généralement fixé à 1.
- inversion de la matrice de covariance $\mathbf{\hat{R}_k}$ pour obtenir $\mathbf{\hat{R}_k^{-1}}$
- calcul du filtre $\mathbf{w_k} = \frac{\mathbf{\hat{R}_k^{-1} s_s}}{\mathbf{s_s^H \hat{R}_k^{-1} s_s}}$ pour chaque case distance k
- application du filtre $\mathbf{y_k} = \mathbf{w_k^H x_k}$ et passage en plan Doppler-distance par FFT pour détection $\mathbf{y_k(t_n)} \Longrightarrow \mathbf{y_k(f_n)}$

Ce traitement, ou au moins chacune de ces étapes, doit être effectué en un temps inférieur au temps de la rafale. Pour une rafale de M_p impulsions et une fréquence de répétition (PRF) f_r, ce temps est $T = \frac{M_p}{f_r} = M_p T_r$.

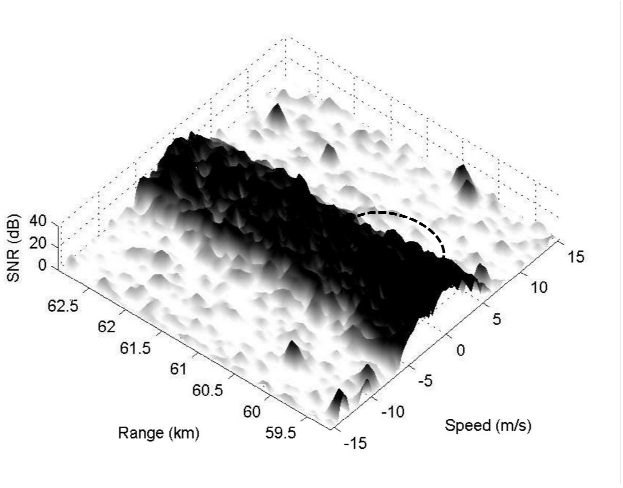

FIG. 1: *Puissance en sortie de la voie somme. Un convoi est présent (entouré).*

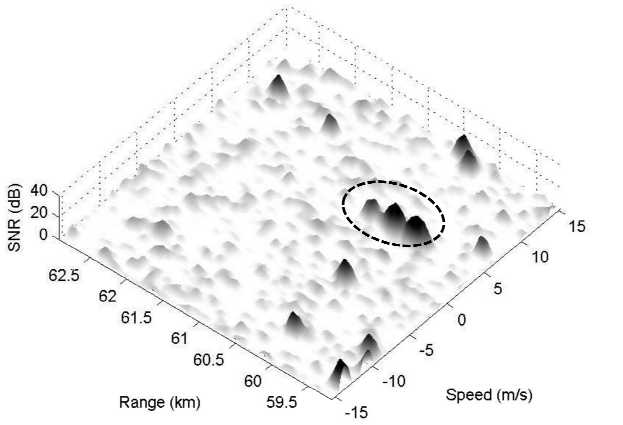

FIG. 2: *Puissance en sortie après traitement STAP. Un convoi est présent (entouré)*

3.2 Algorithmie et GPU

Contrairement au cas précédent où le nombre de capteurs et le nombre de points temporels étaient grands, nous devons ici traiter des données d'une dizaine de capteurs et d'une centaine de points temporels. En revanche, il y a autant de filtres à estimer et à appliquer que de cases distance ($>= 1000$) et ceci doit être dans un temps beaucoup plus court, typiquement une dizaine de millisecondes. Le nombre de cases distance étant grand par rapport au nombre de coeurs, il est très facile de paralléliser le traitement en assignant à chaque coeur une partie des cases distance à traiter. Les coeurs CPU sont donc performants autant sur les tâches lourdes (calcul de N filtres STAP séquentiellement) que sur les tâches légères (calcul d'un élément de la matrice de covariance) contrairement aux GPUs qui le sont uniquement dans les tâches légères

[5]. Le traitement STAP a donc dû être parallélisé au niveau élémentaire ainsi qu'au niveau de chaque case distance. Ceci est possible en utilisant certains algorithmes s'y prêtant bien, comme la méthode du pivot partiel pour l'inversion de la matrice de covariance et en regroupant les tâches élémentaires d'une case distance (ou matrice) en paquets qui sont envoyés à des groupes de coeurs GPU différents. Nous simulons ici un radar de pointe avant de type AMSAR, comportant 8 sous-réseaux de réception. Le nombre de cases distance à traiter est $l = 1000$, le nombre d'impulsions $M_p = 128$ pour une *PRF* de $f_r = 2kHz$ et la fenêtre temporelle choisie est $M = 8$. Les calculs sont faits en double précision (64bit), les temps d'exécution des différentes étapes sont les suivants:

Etape	CPU X5570	GPU Tesla C2050
Estimation	131 ms	20,5 ms
Inversion	94 ms	36 ms
Calcul de $\mathbf{w^H}$	6,2 ms	3,8 ms
Application+FFT	5,3 ms	3,9 ms

Table 2: Temps d'exécutions sur CPU et GPU pour différentes étapes du traitement STAP.

3.3 STAP processing on efficient ARM-GPU architecture

Si la motivation pour les radars basés au sol est la réduction des coûts, pour les radars embarqués nous cherchons à réduire la consommation électrique ainsi que le volume occupé par le système de calcul. Dans ce contexte, l'architecture ARM-GPU proposée par le constructeur de GPU NVIDIA, appelée CARMA est très prometteuse. CARMA est un prototype d'architecture visant à accroitre l'efficacité énergétique des plateforme de calcul GPU. Aujourd'hui, les progrès dans l'amélioration de cette efficacité énergétique se font principalement sur les marchés des smartphones et des tablettes. Les processeurs ARM ont donc un avantage en terme d'architecture et d'expérience puisque ce sont ces processeurs qui équipent principalement ces produits. A cause de son architecture, les GPUs ont aussi une architecture efficace. Un GPU est optimisé pour des débits importants et une puissance de calcul importante alors que les CPUs sont optimisés pour le calcul séquentiel et des latences réduites, le but étant de traiter une série d'opérations le plus rapidement possible. Par exemple, un GPU NVIDIA de type "Fermi" a un ratio de dissipation thermique à puissance de calcul de 225 pJ/flop, alors qu'un CPU x86 Intel "Westmere" de même génération est à 1700 pJ/flop[6]. Un GPU ne peut par contre exécuter un noyau système tout seul, et doit être piloté par un CPU. L'efficacité énergétique du système complet sera donc impactée négativement par la présence du CPU pilote. Évidement, plus on augmentera le nombre de GPUs (dans la limite de ce qu'un CPU peut gérer), plus on tendra vers l'efficacité énergétique des GPUs. Le but de la carte prototype CARMA est d'explorer les possibilités d'utiliser des GPUs et des processeurs CPU à architecture ARM dans le but d'augmenter l'efficacité des systèmes de calcul. Sur la carte CARMA, le processeur traditionnel x86 (Intel, AMD) est remplacé par un processeur ARM ultra-basse consommation. Ce CPU est un Tegra T30 "Kal-El" ayant quatre coeurs de type ARM Cortex-A9 avec extensions NEON et VFPv3, comme nous pouvons en trouver dans plusieurs modèles de smartphones et de tablettes. Le GPU est quant à lui un NVIDIA Quadro 1000M que l'on peut trouver dans des PC portables professionnels d'entrée de gamme. Ce GPU a une puissance de calcul théorique crête de 268 GFlops, il est de même génération et de même architecture que le GPU présent dans la carte Tesla C2050 qui lui a une puissance de calcul crête de 1030 GFlops. La carte embarque son propre système d'exploitation basé sur un noyeau Linux 3.1.10. La carte étant un prototype, le compilateur GPU ne supporte pas encore la compilation sur processeur ARM: les programmes doivent être compilés sur un PC standard puis transférés sur la carte. Enfin, le Quadro 1000M étant un GPU d'entrée de gamme, la puissance de calcul en double précision est bridée. Nous testons ici les deux premiers étages du traitement STAP décrit précédemment en simple précision. Ensemble, ces deux étages comprennent 90 % de la charge de calcul du traitement STAP. Pour ces essais, le processeur central x86 est toujours un Intel Xeon X5570, le GPU est une carte NVIDIA Tesla C2050. Nous comparons les vitesses d'exécution du traitement s'exécutant sur ces deux processeurs avec les vitesses d'exécution du traitement sur la carte CARMA.

Plateforme	Estimation	Inversion
CPU x86 Intel Xeon X5570	72.67 ms	54.40 ms
GPU NVIDIA Tesla C2050	11.46 ms	14.1 ms
NVIDIA CARMA	62.41 ms	84.1 ms

Table 3: Temps d'exécution sur CPU, GPU (PC) et GPU sur carte CARMA de deux étapes de traitement STAP.

Les résultats de la Table 3 montrent qu'un processeur CPU x86 classique et une carte CARMA ont des performances très proches sur ces deux étapes de traitement. Le GPU de la carte Tesla C2050 est par contre 3,8 à 6,3 fois plus rapide.
Nous avons aussi mesuré la consommation électrique de la carte CARMA et du système CPU-GPU (PC). Pour ce dernier, nous avons retranché à la consommation mesurée les consommations des périphériques de refroidissements et de stockage. La consommation électrique du PC au repos est mesurée à $P = 146.5W$, alors qu'elle est de $P = 11.5W$ pour la carte CARMA. Lorsque le processeur central est utilisé pour l'un des étages de traitement STAP, la consommation du système monte à $P = 223W$, et à $P = 348W$ lorsque le GPU est utilisé. Sur la carte CARMA, lors du traitement STAP, la consommation totale de la carte est $P = 37W$. Nous avons ensuite calculé la performance par Watt pour les deux étages de traitement STAP. Pour l'estimation de la matrice de covariance, la charge calculatoire pour un datacube radar est $8lMNK_tMN$, alors qu'elle est d'environ $8l(MN)^3$ pour l'inversion des matrices de covariance. Nous divisons alors cette charge par le temps d'exécution et par la consommation pour obtenir un résultat en GFlops/Watt.

Platform	Estimation	Inversion
x86 system (CPU only)	0.24	0.17
x86 system (GPU)	0.99	0.25
NVIDIA CARMA	1.71	0.68

Table 4: Performances par Watt (GFlops/W) du CPU, GPU sur PC et du GPU sur CARMA pour le STAP.

Bien que n'étant par la solution la plus rapide, la carte CARMA est presque deux fois plus efficace que le GPU haute performance fonctionnant sur un système x86. Ce système a lui même une efficacité bien plus élevé si l'on utilise le GPU que si seulement le CPU est utilisé. Notons que le gain en rapidité d'exécution apporté par les GPU est plus élevé pour l'estimation des matrices de covariance que pour les inversions des matrices de covariance. En effet, le premier algorithme est très parallélisable alors que l'algorithme d'inversion matriciel comporte des parties séquentielles et des accès mémoire cache. Comme prévu, cela n'affecte pas les performances du CPU qui est conçu pour être rapide sur tout type de tâche et possède un grande quantité de mémoire cache.

Conclusion

Dans deux applications de traitement du signal radar, nous avons décrit les implémentations des algorithmes de traitement et mesuré leurs performances. Dans les deux cas, l'utilisation de GPU permet un gain en efficacité et en coût par rapport à des solutions traditionnelles. De plus, nous avons aussi mesuré les performances d'un de ces algorithmes sur une carte ayant une architecture originale ARM+GPU. Avec le marché grandissant des périphériques portables (smartphones et tablettes), les processeurs à architecture ARM tendent à devenir de plus en plus efficaces. Couplés à des GPUs, les plateformes ARM-GPU délivrent un alternative très intéressante aux systèmes basés sur des processeurs CPU traditionnels en terme d'efficacité énergétique. Les travaux portent maintenant sur l'analyse des performances des processeurs ARM pour des tâches séquentielles légères, comme la détection ou la localisation de cibles pour valider l'utilisation d'une telle architecture en pratique sur un système opérationnel.

Références

[1] Michal, T. and Eglizeaud, JP and Bouchard, J. *GRAVES: the new French system for space surveillance*. 4th European Conference on Space Debris, vol.587, p.61, 2005.

[2] Liu, B. and Wang, K. and Liu, X. and Yu, W. *An efficient SAR processor based on GPU via CUDA*. Image and Signal Processing, 2009. CISP'09. 2nd International Congress on, p.1-5, 2009.

[3] Klemm, R. *A STAP overview*.IEEE Aerospace and Electronic Systems Magazine, Vol 19, no. 1, pp19-35, Jan 2004.

[4] J. Pettersson, I. Wainwright. *Radar Signal Processing on Graphics Processors (GPUs)*. Master Thesis, Uppsala Universitet, 2010.

[5] Fallen, C.T. and Bellamy, B.V.C. and Newby, G.B. and Watkins, B.J. *GPU Performance Comparison for Accelerated Radar Data Processing*. Application Accelerators in High-Performance Computing (SAAHPC), 2011 Symposium on, pp 84-92

[6] Don Becker, *CARMA: CUDA on ARM Architecture, Developments in Power-Efficient Computing*. GPU Tech Conference 2012, San Jose.