Algorithmes de Reconnaissance Non Coopérative de Cibles et implémentation sur GPU

Thomas Boulay, Nicolas Gac, Ali Mohammad-Djafari, Julien Lagoutte

Algorithmes de Reconnaissance Non Coopérative de Cibles et implémentation sur GPU

Thomas BOULAY[1,2], Ali MOHAMMAD-DJAFARI[1], Nicolas GAC[1], Julien LAGOUTTE[2]

[1]Laboratoire des signaux et systèmes (L2S) Plateau de Moulon, 3 rue Joliot-Curie, 91192 GIF-SUR-YVETTE Cedex, France

[2]Thales Air Systems, voie Pierre Gilles de Gennes, 91470 Limours, France

thomas.boulay@lss.supelec.fr, nicolas.gac@lss.supelec.fr, djafari@lss.supelec.fr, julien.lagoutte@thalesgroup.com

Résumé – Dans cet article, nous nous sommes intéressés aux problèmes de reconnaissance non-coopérative de cibles en tant que problème de classification supervisée. Nous utilisons pour cela un algorithme des KPPV dont les performances sont détaillées en fonction du nombre de voisins K, du type de distance utilisé et de l'espace de représentation des données. Dans un second temps, cet algorithme a été implémenté sur GPU. Les temps de calcul et de transfert mémoire ont été pris en compte pour évaluer l'apport de cette implémentation.

Abstract – In this paper, first, we present the problem of Non Cooperative Target Recognition (NCTR) as a supervised classification problem. Then, we use a very simple method of K Nearest Neighbors (KNN) to do this classification. We explore and compare the performances of this algorithm based on the choice of the distances and the representation spaces of the data. KNN algorithm will be executed initially on CPU with Matlab and then on GPU using MEX functions of Matlab. Time computing and memory transfert time will be taken into account to evaluate the benefit of such an implementation.

1 Introduction

Au cours des derniers conflits, les techniques d'identification coopérative de cibles ne se sont pas révélées toujours fiables et il s'avèrent de plus en plus que les techniques non coopératives, appelés NCTR [1] (Non Cooperative Target Recognition) deviennent indispensables. Une des techniques possibles pour identifier une cible consiste à comparer la signature de la cible à identifier avec les signatures contenues dans une base d'apprentissage. Dans ce cas, on peut assimiler le problème NCTR à un problème de classification supervisé [2]. Parmi ces méthodes, on trouve la méthode des K Plus Proches Voisins (KPPV). Cette méthode va classer la cible dans la classe majoritaire parmi les K plus proches voisins. Le choix de cet algorithme par rapport à d'autres techniques, de type "Isomap" par exemple, est justifié par notre intention de ne pas compresser l'information et donc d'utiliser la puissance de calcul des GPUs pour étudier sur nos données les performances d'algorithmes de type "brute force".

Cet article compare les performances de reconnaissance de l'algorithme des KPPV appliqué :

- sur les profils distances [3] (représentant l'amplitude de l'onde radar réfléchie en fonction de la distance), classiquement utilisé [4, 5] ;
- avec les représentations fréquentielles (signaux IQ avant transformée de Fourier), plus novateur [6, 7].

Pour cela, plusieurs distances sont utilisées (L1, L2 et Kullback). Comme la reconnaissance de cible nécessite une puissance de calcul élevée, une part importante de l'étude a été consacrée à l'implémentation de cet algorithme sur GPU afin d'évaluer l'apport de cette technologie pour ce type d'application. Des algorithmes de recherche des K plus proches voisins ont déjà été implémentés sur GPU pour des applications dans le traitement d'image [8] avec des performances très intéressantes. Au vue de ces études antérieures et de la nature massivement parallèle du calcul des distances [9], la solution d'une implémentation sur GPU nous a paru être la meilleure.

L'article sera structuré comme suit : la section 2 présente les données dans deux espaces (distance et fréquentiel), la section 3 décrit sommairement l'algorithme des KPPV et ses paramètres de réglage, la section 4 détaille l'algorithme des KPPV exécuté dans les deux espaces de représentation, la section 5 décrit l'implémentation CUDA de l'algorithme et finalement la section 6 résume les performances de l'algorithme des KPPV pour diverses configurations.

2 Présentation des données

La Haute Résolution Distance (HRD) offre un moyen simple et rapide de caractériser une cible à travers les profils distance [3]. Un profil distance est une image 1-D de la cible (figure 1).

Il représente la Surface Equivalente Radar (SER) mesurée en chaque point de la cible qui réfléchit l'onde radar. Les profils distance sont obtenus par exemple en émettant N impulsions à une fréquence linéairement croissante. La réponse cohérente (N nombres complexes) subit au final une transformée de Fourier de manière à obtenir le profil distance.

Cependant on peut se demander pourquoi ne pas utiliser directement ces N nombres complexes ou plus précisement leurs modules pour caractériser la cible (figure 1). Cette solution a des avantages par rapport à l'utilisation des profils distances notamment l'absence d'une phase de recalage des profils distance indispensable dans l'espace distance ([7]). Nous étudierons donc les performances de l'algorithme des KPPV en utilisant ce type de signature fréquentielle.

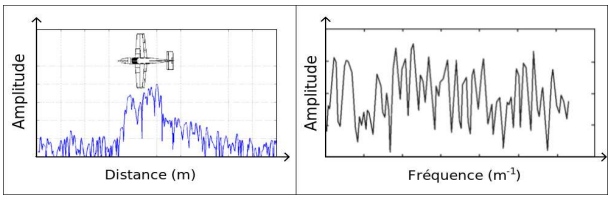


FIGURE 1 – Un exemple de profil distance et de spectre fréquentiel

Les données utilisées pour notre article sont des données simulées. Elles sont découpées en deux bases : une base de test et une base d'apprentissage. Dans la base de test, on trouve $N_{te} = 150$ signatures de 3 classes différentes correspondant chacune à une cible ($N_{te1} = N_{te2} = N_{te3} = 50$) et dans la base d'apprentissage, on trouve $N_{tr} = 1024$ signatures également de 3 classes différentes ($N_{tr1} = N_{tr2} = N_{tr3} = 341$).

3 Algorithme des KPPV

L'algorithme des KPPV est fondé sur une méthode d'apprentissage supervisée. Soit x_{te} et X_{tr}, respectivement la signature à tester et la base d'apprentissage. x_{te} est un vecteur de taille N_s échantillons. X_{tr} est une matrice $N_{tr} \times N_s$ avec N_{tr}, le nombre de signature dans la base d'apprentissage. La première étape (i) de l'algorithme des KPPV consiste à calculer les distances entre x_{te} et les N_{tr} signatures de la base d'apprentissage X_{tr}. Dans l'article, trois distances sont étudiées : la distance euclidienne (L2), la distance de Manhattan (L1) et la distance de Kullback (KUL). Dans le domaine distance, où les signaux peuvent prendre des valeurs négatives et positives, nous utiliserons seulement les distances L1 et L2. La définition de ces distances reste cependant identique dans l'espace distance et l'espace fréquentiel car dans l'espace fréquentiel, nous utiliserons le module du signal fréquentiel pour l'algorithme des KPPV. Dans les deux cas, on traite donc des variables réelles. La seconde étape (ii) consiste à trier les distances obtenues. On définit D le vecteur contenant les K plus petites distances. La dernière étape (iii) est l'étape de décision. On détermine le nombre d'occurences de chacune des 3 classes que contient le vecteur D. La signature de test sera classée dans la classe majoritaire parmi les K plus proches voisins. A partir de cette description, il apparaît que 3 paramètres peuvent influencer le résultat de l'algorithme des KPPV : le paramètre K, le type de distance utilisé, le type de signature utilisé (profils distance ou spectres fréquentiels).

4 KPPV dans deux espaces différents

Pour obtenir les profils distance à partir des réponses fréquentielles, plusieurs étapes sont nécessaires. La première étape consiste à appliquer une transformée de Fourier sur la réponse fréquentielle. $N_s = 546$ points, comprenant le signal utile, sont alors conservés. Lors de l'exécution de l'algorithme des KPPV, une étape de recalage des profils distance est nécessaire. En effet, les profils distance ne sont pas forcément recalés en distance les uns avec les autres. Un processus d'alignement est donc appliqué pour chaque profil distance sous test. Lors du calcul de la distance avec un profil distance d'apprentissage, on va en fait calculer $(2 \times N + 1)$ distances correspondant aux distances entre le profil de la base apprentissage et le profil distance sous test décalé de $\pm N$. Au final, la distance retenue sera la distance minimale parmi ces $(2 \times N + 1)$ distances. Une fois que l'ensemble des distances a été calculé, il ne reste plus qu'à prendre une décision.

L'algorithme des KPPV dans l'espace fréquentiel est quasiment le même que celui exécuté dans l'espace distance, seules les signatures utilisées changent. En effet, dans l'espace fréquentiel, les spectres fréquentiels sont utilisés directement. Il n'y a pas besoin de recalage car on n'utilise uniquement les modules des spectres fréquentiels. Les spectres fréquentiels sont constitués de $N_s = 2400$ échantillons.

5 Implémentation GPU

Dans cette section, nous présentons donc une implémentation GPU du calcul des distances associé à l'algorithme des KPPV et du processus de recalage (uniquement dans l'espace distance). Pour faciliter le traitement des résultats, nous avons intégré ce calcul à l'environnement Matlab grâce aux fichiers CUDA-MEX [10]. Le GPU utilisé pour notre article est le NVIDIA Tesla C2060 basé sur la nouvelle architecture Fermi. Cette architecture développé par NVIDIA intègre plusieurs innovations majeurs par rapport aux anciennes architectectures parmi lesquelles l'amélioration des performances de calcul en précision "double", l'intégration de codes correcteurs d'erreurs (ECC), une hiérarchie des "caches" mémoires ou encore une augmentation de la mémoire partagée.

5.1 Parallélisation des calculs

5.1.1 Architecture CUDA

CUDA [11] (Common Unified Device Architecture) est un modèle de programmation SIMT (Simple Instruction Multiple Threads) des architectures “many cores” du fabriquant NVIDIA. On définit un “kernel” comme étant une portion parallèle de code à exécuter sur le GPU. Chacune de ces instances s’appelle un “thread”. Un même “kernel” est exécuté en parallèle par tous les threads. Avec l’architecture CUDA, les threads sont organisés de manière hiérarchique. Ils sont en effet regroupés en “blocs de threads”, eux mêmes regroupés en “grilles de bloc de threads”. Avec la nouvelle architecture Fermi, le nombre de threads par bloc peut aller jusqu’à 1024.

5.1.2 Le “kernel” pour le calcul de distance

L’algorithme des KPPV est un algorithme massivement parallèle. Un thread calcule, en parallèle avec les autres, une distance entre une signature sous test et une signature de la base d’apprentissage. Pour paralléliser le calcul, on va donc créer $N_{blocks} = 150$ blocs de $N_{threads/block} = 1024$ threads, ce qui correspond aux $N_{te} = 150$ signatures de la base de test et aux $N_{tr} = 1024$ signatures de la base d’apprentissage. Le “kernel” exécuté sur le GPU est toujours le même, seul change l’indice du bloc (indice k) et du thread (indice j) nous permettant d’accéder respectivement aux signatures de la base de test et de la base d’apprentissage (figure 2). Dans notre application, un thread calculera une distance sur $N_s = 546$ échantillons dans l’espace distance et $N_s = 2400$ échantillons dans l’espace fréquence.

Chaque $Th_k(Thread_k)$:

$j = blockDim.x \times blockIdx.x + threadIdx.x$
$k = blockIdx.y$
for $i \leq N_s$ **do**
  $sum = sum + distance(X_{te}(k,i), X_{tr}(j,i))$
end for

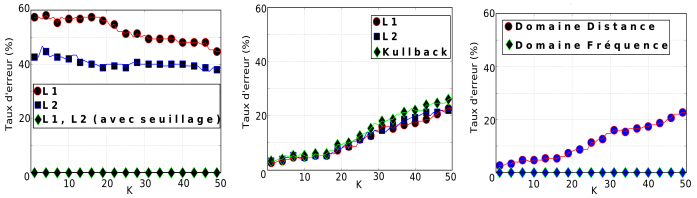
FIGURE 2 – Pseudo-Code du “kernel”

5.2 Optimisations des accès mémoire

Les performances d’un programme GPU sont très influencées par la gestion mémoire. En effet, avec CUDA, la mémoire est également organisée hierarchiquement. Il existe plusieurs solutions pour accéder aux données. Chaque thread a sa propre mémoire, les registres. Chaque bloc a une mémoire partagée entre tous les threads, la mémoire “shared”. Chaque thread de n’importe quel bloc peut également accéder à la mémoire globale du GPU. Pour accéder à cette mémoire globale plus rapidement, il existe deux “caches” : le “cache texture” et le “cache constante”. Avant Fermi, il fallait éviter d’accéder à la mémoire globale sans passer par les “caches” car la latence était très importante. Avec Fermi, nous avons mis en évidence dans une étude préalable [12] que l’utilisation du “cache texture” pour accéder à la base d’apprentissage n’était plus indispensable, l’accès à la mémoire globale étant automatiquement “caché” sur 768 KB en utilisant un GPU Fermi . Par conséquent, nous avons stocké la base de test en mémoire “shared” et la base d’apprentissage en mémoire globale . Le résultat de chaque distance calculée est stocké dans des registres avant d’être transféré en mémoire globale.

6 Performances de l’algorithme des KPPV

6.1 Performance en terme de taux d’erreur



(a) (b) (c)

FIGURE 3 – (a), (b) Taux d’erreur en fonction de K pour l’algorithme des KPPV respectivement dans l’espace distance et l’espace fréquentiel. (c) Comparaison entre le taux d’erreur en fonction de K dans l’espace distance et dans l’espace fréquentiel.

Comme on l’a vu dans la section 3, trois paramètres peuvent influencer les performances de l’algorithme des KPPV. Pour explorer les différentes situations, nous avons étudié séparément l’algorithme des KPPV dans l’espace distance et l’espace fréquence. Dans le domaine distance, où les profils distance peuvent prendre des valeurs positives ou négatives, nous avons utilisé les distances euclidienne et de Manhattan. Dans le domaine fréquence, où les signaux sont positifs, nous avons utilisé en plus la distance de Kullback. Pour évaluer les performances, nous traçons le taux d’erreur (nombre d’erreurs d’identification sur le nombre total de cibles) en fonction de K pour chacune de ces configurations.

Dans l’espace distance (3(a)), le taux d’erreur varie légérement suivant les valeurs de K autour de 40% pour la norme L2 et autour de 50% pour la norme L1. Cependant, lorsqu’on applique un simple seuillage sur les profils distance, de manière à éliminer les échantillons de bruit, le taux d’erreur devient égale à 0% pour les deux distances.

Dans l’espace fréquence (3(b)), le taux d’erreur varie de 5% à 30% en fonction de K pour les trois types de distance. En ne traitant pas le bruit, on se rend compte que les performances de l’algorithme sont meilleures dans le domaine fréquence que dans le domaine distance. En revanche, on ne peut pas éliminer le bruit aussi facilement dans le domaine fréquentiel que dans le domaine distance. Pour traiter le bruit dans le domaine fréquence, il faudrait développer d’autres méthodes plus élaborées qu’un simple seuillage qui sortiraient du cadre de notre article.

Sur la courbe 3(c), on se rend compte que les meilleures performances de reconnaissance sont obtenues en utilisant les

profils distance seuillés. Avec nos données, sans traitement du bruit, il est plus intéressant de travailler avec les signatures fréquentielles qu'avec les profils distance. Cependant, avec un rapport signal sur bruit plus faible, il serait de plus en plus compliqué d'obtenir des performances convenables avec les spectres fréquentiels sans traiter le bruit.

6.2 Performances en terme de temps de calcul

Sur le tableau 1, sont présentés les temps de calcul (incluant temps de transfert entre le PC et la carte graphique) pour l'algorithme des KPPV dans l'espace fréquence et distance. Les temps de calcul de l'algorithme exécuté uniquement sous Matlab (le calcul sous Matlab a été orienté de manière à ce qu'il soit matriciel) et ceux de l'algorithme exécuté sous Matlab avec calcul des distances sur GPU sont comparés. Les gains obtenus vont jusqu'à 237 dans le domaine fréquence et 376 dans le domaine distance. Dans l'espace distance, où le recalage des profils distance est nécessaire, les gains sont également beaucoup plus importants que dans le domaine fréquence. Au final, le temps de calcul reste plus faible dans l'espace fréquence que dans l'espace distance. Par contre, dans le domaine fréquentiel, on se rend compte que le pourcentage de transfert mémoire n'est pas négligeable (23% pour la distance L1 et L2). L'utilisation d'un algorithme itératif, où les données ne sont chargées qu'une seule fois en mémoire, est donc très important pour profiter au maximum des performances offertes par les GPUs. A noter que sont inclus dans les temps de calcul les temps de transfert entre GPU et CPU.

TABLE 1 – Temps de calcul (en ms) de l'algorithme des KPPV exécuté sur Matlab ou Matlab MEX-CUDA avec $K = 5$

	Espace Fréquence			Espace Distance	
	L1	L2	KUL	L1	L2
Matlab	$1.1e3$	$1.7e3$	$1.9e4$	$1.3e5$	$1.4e5$
Matlab CUDA	14.1	14.1	80.3	372	372
Gain	$\times 78$	$\times 121$	$\times 237$	$\times 349$	$\times 376$
% transferts PC $\leftrightarrow$ GPU	23%	23%	3%	0.3%	0.3%
Matlab CUBLAS		7			

CPU :Intel Xeon à 2.40GHz, GPU :NVIDIA Tesla C2050

Nous avons également implémenté une version du calcul des distances L2 utilisant la librairie CUBLAS. Nous avons adapté pour cela l'expression de la distance euclidienne :

$$d_{euclidian} = \underbrace{||\vec{x}||^2 + ||\vec{y}||^2}_{Kernels\ concurrents} - \underbrace{2\vec{x}.\vec{y}}_{CUBLAS} \quad (1)$$

En utilisant la librairie CUBLAS pour calculer le produit de la base de test par la transposé de la base d'apprentissage et en utilisant des kernels concurrents (possible uniquement sur l'architecture FERMI) pour calculer les normes de chaque profil distance, on gagne encore un facteur 2 sur le temps de calcul pour atteindre $7ms$.

7 Conclusion

Les principaux enseignements qui ressortent de cet article sont les suivants :

- Les meilleurs résultats de classification sont obtenus lorsqu'on applique l'algorithme sur les profils distance seuillés. L'algorithme des KPPV n'est pas adapté pour la reconnaissance dans le domaine fréquentiel ;
- Le type de distance utilisé a peu d'influence à part dans le domaine distance sur les profils non seuillés où la distance L2 donne de meilleures résultats que la distance L1. Les performances varient peu en fonction de K sauf dans le domaine fréquentiel, où lorsque K dépasse 15, le taux d'erreur augmente ;
- Le coût de calcul a été significativement réduit en utilisant les GPUs (gain maximum de 376 pour le calcul de la distance L2 dans l'espace distance) ;
- En adaptant les calculs pour l'utilisation de CUBLAS, un facteur 2 a été obtenu.

Références

[1] M. Moruzzis and N. Colin, "Radar target recognition by Fuzzy Logic," *IEEE Aerospace and Electronic Systems Magazine*, vol. 13, no. 7, pp. 13–20, 1998.

[2] S. B. Kotsiantis, "Supervised machine learning : A review of classification techniques," Amsterdam, 2007.

[3] Donald R. Wehner and Bruce Barnes, *High-Resolution Radar*, Artech House Publishers, 1994.

[4] Andrew R. Webb, "Gamma mixture models for target recognition," *Pattern Recognition*, pp. 2045 – 2054, 2000.

[5] Lan Du and al., "A two-distribution compounded statistical model for radar HRRP target recognition," 2006.

[6] A. Zyweck and R. E. Bogner, "Radar target classification of commercial aircraft," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 32, pp. 598–606, 1996.

[7] S. K. Wong, "Non cooperative target recognition in the frequency domain," *IEEE Radar and Sonar*, 2004.

[8] V. Garcia and al., "Fast GPU-based implementations and application to high-dimensional feature matching," in *IEEE Int Image Processing (ICIP) Conf*, 2010.

[9] Owens and al, "A survey of general-purpose computation on graphics hardware," *Computer Graphics Forum*, vol. 26, pp. 80–113, 2007.

[10] NVIDIA, "White Paper : Accelerating MATLAB with CUDA Using MEX Files," September 2007.

[11] *NVIDIA CUDA C Programming Guide Version 3.1.1*.

[12] *Implémentation et optimisation sur GPUs dans le contexte NCTR*. Ecole d'été GPU 2011, Grenoble, 2011.