Algorithmes de reconnaissance NCTR et parallélisation sur GPU

Thomas Boulay [1,2], **Nicolas Gac** [1], **Ali Mohammad-Djafari** [1], **Julien Lagoutte** [2]

1. *Laboratoire des Signaux et Systèmes (L2S), UMR8506 CNRS-SUPELEC-UNIV PARIS SUD, 3 rue Joliot-Curie, 91192 Gif-sur-Yvette Cedex, France*
   thomas.boulay@lss.supelec.fr,djafari@lss.supelec.fr,nicolas.gac@lss.supelec.fr
2. *Thales Air Systems, voie Pierre Gilles de Gennes, 91470 Limours, France*
   thomas.boulay@thalesgroup.com,julien.lagoutte@thalesgroup.com

RÉSUMÉ. *Dans cet article, nous nous sommes intéressés aux problèmes de reconnaissance non-coopérative de cibles (NCTR) en tant que problème de classification supervisée. Après une présentation du système d'acquisition des profils distance radar et du problème de reconnaissance, suivie d'une étude statistique des données, nous proposons d'utiliser un algorithme des K plus proches voisins (KPPV) dont les performances sont détaillées en fonction du nombre de voisins K, du type de distance utilisée et de la nature des données utilisées (débruitées ou non). Dans un second temps, cet algorithme a été parallélisé sur un processeur many-cœurs (GPU : Graphics Processing Unit). Les opérations arithmétiques et le modèle d'accès mémoire ont été étudiés pour obtenir la meilleure parallélisation des calculs. Enfin, nous terminons par une discussion autour des perspectives envisageables pour la méthode proposée, notamment en s'intéressant à d'autres espaces de représentation ou à d'autres méthodes de classification.*

ABSTRACT. *In this paper, first, we present the problem of Non Cooperative Target Recognition (NCTR) as a supervised classification problem. After a presentation on the radar acquisition system of range profiles and the problem of recognition, followed by a statistical study of data, we use a classical classification 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, the choice of K and the nature of used data (denoised or not). KNN algorithm has been executed initially on CPU with Matlab and then on GPU. Arithmetic operations and memory access pattern has been studied to get the best parallelization. Finally, we conclude with a discussion about possible perspectives for the proposed method especially by focusing on other representation spaces or other classification methods.*

MOTS-CLÉS : *NCTR, KPPV, GPU, HRD, radar, classification.*

KEYWORDS: *NCTR, KNN, GPU, HRR, radar, classification.*

DOI:10.3166/TS.30.309-342 © 2013 Lavoisier

Extended abstract

Context

During major conflicts, cooperative classification techniques are not reliable enough and Non Cooperative Target Recognition (NCTR) (Moruzzis, Colin, 1998 ; Nimier *et al.*, 2000) is increasingly seen as essential. Most of classification techniques consist in comparing target signature under test with target signatures contained in a data set. In this case, classification problem is considered as a supervised classification problem (Duda *et al.*, 2001 ; Kotsiantis, 2007 ; Ma, Ji, 1999 ; Jain *et al.*, 2000). K-Nearest Neighbors (KNN) methods is one of them. Advantage of KNN method, compared to Isomap techniques for example, is that it can work directly on the original data without any transformation. The main drawback of KNN or more generally of « brute force » algorithm is that the computational cost is very high but this drawback can be overcome with the arrival in 2006 of « many-cores » processors like GPU (Graphics Processing Unit).

Indeed, for several years, GPU have become increasingly powerful and programmable (Owens *et al.*, 2007). This development has increased the use of GPU and allowed acceleration of one to two orders of magnitude in many application areas such as physical simulation (Cooke *et al.*, 2011), image compression (Lin, Lin, 2010) or tomography (Xu, Mueller, 2007). In the litterature, K-nearest neighbors searching algorithms have already been implemented on the GPU for applications in image processing and especially for an application of high dimensional feature matching (Garcia *et al.*, 2010) with interesting performances. In view of these previous studies and the massively parallel nature of the calculation of distances, the use of GPU seems to be best suited to the field of target recognition where loads to identify a target are often very important and where the goal of real-time processing (we are fixed 1 ms by range profile for our application) is a major constraint.

Method

The KNN algorithm is based on a supervised learning method (Kantardzic, 2011). Let $\boldsymbol{x}_T$ and $\boldsymbol{X}_A$, the range profile under test and the learning set respectively. $\boldsymbol{x}_T$ is a vector of size N, the number of range bins in a range profile. $\boldsymbol{X}_A$ is a $N_A \times N$ matrix, with N_A, the number of range profiles in the learning set.

The first step of KNN algorithm is to compute distances between $\boldsymbol{x}_T$ and the N_A range profiles of the learning set $\boldsymbol{X}_A$. In this paper the distance used is Euclidian distance:

$$d_{euclidian} = \sum_i (x_i - y_i)^2$$

During execution of the KNN algorithm, a step of range profiles alignment is necessary. Indeed, the range profiles are not necessarily aligned in distance relative to each other. An alignment process is applied to each profile distance under test. When

calculating the distance with a learning range profile, $(2 \times D + 1)$ distances corresponding to the distance between the range profiles of learning set and the range profile under test shifted of $\pm D$ range bins are calculated. Finally, the distance selected is the minimum distance from the $(2 \times D + 1)$ distances. In our case, the range profiles of learning set are aligned reports to each other.

The second step consists to sort computed distances. Let $\boldsymbol{d}$ be a vector containing the K smallest distances. In the third step, the decision step, the number of occurences in the vector $\boldsymbol{d}$ of each classes is committed. Range profile under test is classify in the majority class among K nearest neighbors. In case of a tie, the range profile is assigned to the class with the closest neighbor for the range profile under test.

Experiment

Firstly, a statistical analysis of the dataset is proposed based on covariance matrix analysis. This statistical analysis of the dataset allows to obtain, upstream of classification processing, an approximation of classification performance. Then, performances of our KNN algorithm are presented in term of error rate and success rate. The second part of the paper deals with parallelization on GPU of KNN algorithm. The goal of this part is to achieve real-time constraint. In this perspective, a study of memory access model and arithmetic operations is performed to get closer to the theorical computation performances of GPU cards used. Several GPU implementations (CUDA C++, OpenCL, Matlab CUDA, Matlab OpenCL) are proposed and their performances are compared to each other.

Results and conclusion

With the KNN algorithm and our dataset, error rate obtained is low (1 %) and success rate is high (98 %). However, with an actual dataset, KNN algorithm does not allow to maintain error rate low enough. Therefore, it is necessary to define new algorithms to ensure a sufficiently low error rates while maximizing the success rate. The second part highlights issues and difficulties related to the effective use of GPU. As the computation time of an optimized Matlab implementation is $1{,}3$ s, the best KNN parallelization on GPU can handle 150 range profiles in $59{,}7$ ms, ie. $0{,}40$ ms per range profile, which is significantly lower than our real-time limit. Therefore, GPU parallelization significantly reduces computation time of KNN algorithm. Furthermore, this part brings out advantage of using Fermi cards from Tesla cards. Indeed, in addition to an higher processing power (peak), Fermi cards can operate more easily and quickly the computing power of GPU thanks to cache memories.

1. Introduction

Au cours des derniers conflits, les techniques d'identification coopérative de cibles ne se sont pas révélées toujours fiables et il s'avère de plus en plus que les techniques non coopératives utilisant des signaux radar, appelées NCTR (Moruzzis, Colin, 1998 ; Nimier *et al.*, 2000) (*Non Cooperative Target Recognition*) deviennent indispensables. Une des techniques possibles pour identifier une cible consiste à comparer la signature radar 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 supervisée (Duda *et al.*, 2001 ; Kotsiantis, 2007 ; Ma, Ji, 1999 ; Jain *et al.*, 2000). 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. Elle est décrite plus en détail dans la section 4. Le choix de cet algorithme par rapport à d'autres techniques, de type « Isomap » (Tenenbaum *et al.*, 2000 ; Balasubramanian, Schwartz, 2002) par exemple, est justifié par notre intention de ne pas compresser l'information et donc d'utiliser des algorithmes de type « force brute », alternative rendue possible dans une optique de traitement temps-réel depuis l'arrivée en 2006 des processeurs « many cores » de type GPU (*Graphics Processing Unit*).

En effet, depuis plusieurs années les GPU sont devenus de plus en plus puissants et programmables (Owens *et al.*, 2007). Ce développement a intensifié l'utilisation des GPU et permis des accélérations d'un à deux ordres de grandeur dans de nombreux domaines d'applications comme les simulations physiques (Cooke *et al.*, 2011), la compression d'image (Lin, Lin, 2010) ou encore la tomographie (Xu, Mueller, 2007). Dans la littérature, les algorithmes de recherche des K plus proches voisins ont déjà été implémentés sur GPU pour des applications dans le traitement d'image et notamment pour une application de « matching » de caractéristiques de grande dimension (Garcia *et al.*, 2010) avec des performances intéressantes. Aux vues de ces études antérieures et de la nature massivement parallèle du calcul des distances (Owens *et al.*, 2007), l'utilisation des GPU nous a paru être la mieux adaptée au domaine de la reconnaissance de cibles où les charges de calcul pour identifier une cible sont souvent très importantes et où l'objectif de traitement en temps-réel (on se fixe $\approx$ 1 ms par profil distance pour notre application) est une contrainte forte.

L'article est structuré comme suit : les sections 2 et 3 présentent respectivement les principes de la reconnaissance non coopérative de cibles et les données utilisées dans notre étude. L'algorithme des KPPV et ses paramètres de réglage est ensuite décrit dans la section 4. La section 5 commence par une étude statistique des données puis s'intéresse aux performances de classification obtenues avec l'algorithme des KPPV. La section 6 aborde ensuite la parallélisation de l'algorithme des KPPV sur GPU. Une étude du modèle d'accès mémoire et des opérations arithmétiques est réalisée de manière à s'approcher au maximum des performances théoriques des cartes GPU utilisées. Les conclusions et perspectives de cette étude sont présentées dans les sections 7 et 8.

2. Reconnaissance non coopérative de cibles (NCTR)

2.1. Présentation générale

On distingue deux types d'exploitation des données radars fonctionnellement très différents : les applications coopératives et les applications non coopératives. Contrairement aux applications coopératives, les applications non coopératives sont réalisées sans collaboration de la cible. Que ce soit pour les applications coopératives ou non coopératives, deux niveaux de classification peuvent être définis. Le premier niveau, que l'on appelle « classification », est un niveau où l'on souhaite distinguer des grandes classes de cibles (hélicoptère, avion de ligne, chasseurs, etc.). Le deuxième niveau de classification est celui que l'on appelle plus communément « reconnaissance ». La reconnaissance est plus fine que la classification. Il s'agit en effet de distinguer à l'intérieur d'une grande classe de cibles, plusieurs types de cible. Par exemple, pour la classe des chasseurs, on cherchera à distinguer un Rafale, d'un Mirage 2000 ou d'un F-16. Dans le cas non coopératif, ces deux niveaux sont la plupart du temps regroupés sous le même terme de reconnaissance non coopérative de cible ou non cooperative target recognition (NCTR) en anglais. Les applications de reconnaissance non coopérative deviennent aujourd'hui indispensables au regard des possibles défaillances des applications de reconnaissance coopérative. En effet, ces dernières partent du principe que la coopération est toujours garantie or cela n'est pas toujours le cas, en raison, notamment, de problèmes techniques, d'erreurs humaines ou le plus souvent, en cas de conflits, d'intentions hostiles de la part de la cible à identifier.

2.2. Profils distance haute résolution

La haute résolution distance (HRD) offre un moyen simple et rapide pour caractériser une cible à travers les profils distance (Wehner, Barnes, 1994). Un profil distance est une image 1-D de la cible (cf. figure 1). Il s'agit d'une représentation de la réponse temporelle de la cible à une impulsion radar haute résolution.

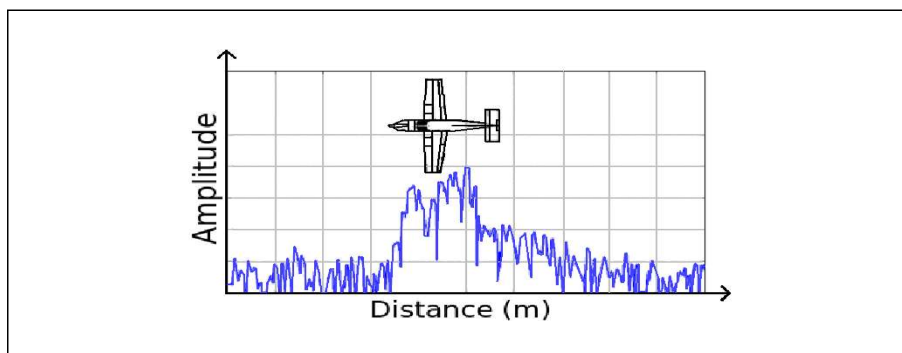

Figure 1. Exemple de profil distance

2.2.1. *Diffuseurs de la cible*

Par approximation, on peut considérer que les profils distance sont le résultat d'un certain nombre de diffusions indépendantes. On considère qu'une cible est constituée de plusieurs diffuseurs (cf. figure 2). On parle également de points brillants que l'on peut définir comme un point ou une surface élémentaire réfléchissant les ondes électromagnétiques.

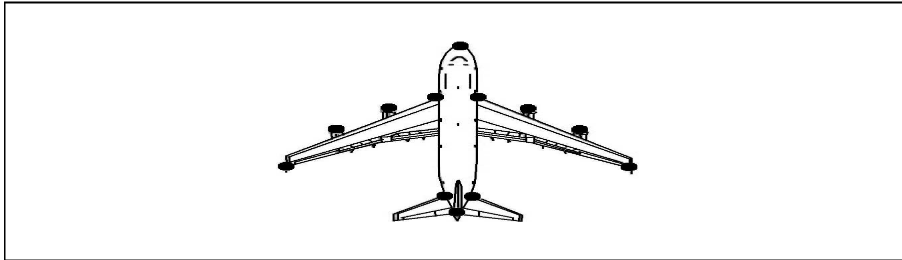

Figure 2. Quelques diffuseurs sur un avion

Les réflexions sur ces diffuseurs proviennent des différentes structures physiques de la cible. Par conséquent, il existe des phénomènes d'interférences constructives et destructives entre les diffuseurs. La combinaison des diffuseurs et les interférences entre ces différents diffuseurs rendent la modélisation de la diffusion très complexe et il est souvent difficile de prédire efficacement un profil distance à partir d'un modèle de diffusion simple (Rihaczek, Herschowitz, 2000).

2.2.2. *Influence de l'angle d'aspect*

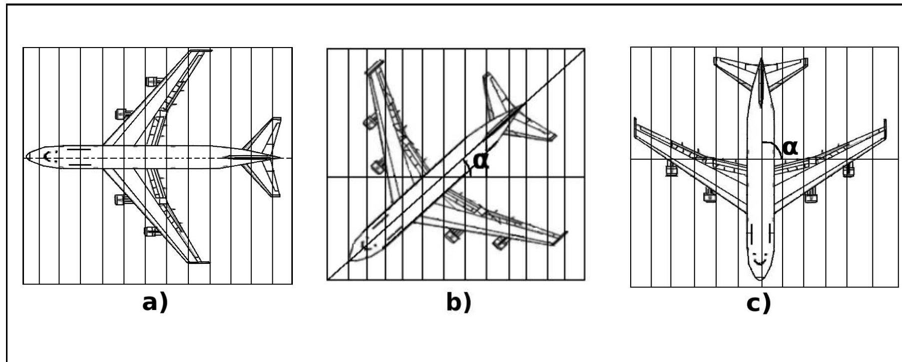

Figure 3. Influence de l'angle d'aspect

L'effet principal de cette combinaison est que les profils distance peuvent varier très rapidement suivant l'angle d'aspect [1].

1. L'angle d'aspect d'un avion peut être exprimé comme une paire de coordonnée (α, θ) où α repésente l'azimut et θ l'angle de site. L'azimut est l'angle entre la direction du nez de l'avion et la direction de la ligne de vue du radar projetée sur le plan formé par le nez et les ailes de l'avion. L'angle de site est l'angle entre la ligne de vue du radar et le plan formé par le nez et les ailes de l'avion.

En effet, suivant les valeurs de l'azimut α et de l'angle de site θ, les parties de l'avion contribuant au profil distance varient. Sur la figure 3a, les cases distance sont distribuées suivant l'axe du fuselage de l'appareil. Dans cette configuration, on peut clairement identifier quelles parties de l'avion contribuent au profil distance. Sur la figure 3b, on se rend compte que l'aile droite de l'avion contribuera à la valeur du signal de retour en plus du fuselage pour la plupart des cases distance. Pour la figure 3c, le fuselage est restreint à quelques cases distance.

2.2.3. Formation du profil distance

Il existe plusieurs types de forme d'onde permettant d'obtenir un profil distance (Tait, 2005). Les formes d'onde dites « chirp » ou « stepped frequency » sont les formes d'onde les plus souvent utilisées. Le « chirp » est une forme d'onde modulée linéairement en fréquence. Cependant les radars actuels ne sont pas tous capables d'émettre des chirps modulés en fréquence sur de large bande. On utilise alors des formes d'onde dites « stepped frequency ». Ces formes d'onde sont obtenues en augmentant la fréquence d'émission radar d'un pas Δf dans la bande radar utilisée. Pour chaque fréquence transmise, on récupère la valeur de la surface équivalente radar (SER) qui résulte de la combinaison des réflexions sur chaque diffuseur. Le signal ainsi obtenu, constitue la signature fréquentielle de la cible.

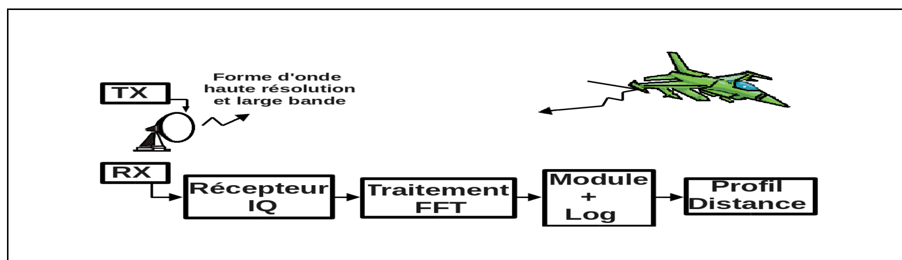

Figure 4. Formation du profil distance

D'une manière générale et cela quelle que soit la forme d'onde utilisée, la chaîne d'acquisition des profils distance se compose d'un récepteur IQ pour transformer l'onde radar en un signal fréquentiel complexe et d'un bloc permettant de calculer la transformée de Fourier de ce signal (cf. figure 4). En effet, le signal capturé par le récepteur est un signal fréquentiel représentant la valeur de SER pour chaque fréquence émise. Avant toute chose, le signal est pondéré par une fenêtre (le plus souvent de Hamming), de manière à limiter l'influence des lobes secondaires. Ensuite pour obtenir un profil distance, la transformée de Fourier de ce signal fréquentiel pondéré est calculée de manière à obtenir une représentation de l'amplitude du signal radar en fonction de la distance. Finalement, le profil distance est obtenu en prenant le module de ce signal. Ce profil distance est le plus souvent exprimé à l'échelle logarithmique de manière à faire ressortir les points brillants de faible amplitude.

Dans la suite de l'étude, on notera $N = 546$ le nombre d'échantillons ou de « cases distance » de nos profils distance. Un exemple de profil distance est représenté sur la

figure 1. Le lecteur désirant plus d'information sur la formation des profils distance peut se rapporter, par exemple, au livre (Tait, 2005) qui s'intéresse en détail à ce sujet.

3. Bases de d'apprentissage et base de test

Les données utilisées pour notre article sont des données synthétiques. Elles sont découpées en deux bases : une base d'apprentissage et une base de test. Chacune de ces deux bases contient trois types de chasseurs différents. Par commodité, on emploiera dans la suite de l'article le mot « classe » pour parler des différents types de chasseurs.

La base d'apprentissage contient $N_A = 1\ 024$ profils distance répartis équitablement dans $C = 3$ classes différentes ($N_A^c = 341$ avec $c = 1, \ldots, 3$). Un profil distance artificiel a été ajouté de manière à avoir $N_A = 1\ 024$, ce qui simplifie l'implémentation sur GPU. Les profils distance de la base d'apprentissage ont été obtenus à partir de modélisations CAO des cibles sur lesquelles ont été appliquées des techniques de calcul de SER. Pour chaque configuration (fréquence, angle de site, azimut), les valeurs complexes de SER sont calculées. La base d'apprentissage est en fait restreinte en limitant la variation de l'azimut et de l'angle de site autour de l'azimut α_t et de l'angle de site θ_t des profils distance de la base de test, qui ont tous le même azimut et site. Tous les profils distance correspondant à un azimut appartenant à l'intervalle $[\alpha_t - 3{,}75°,\alpha_t + 3{,}75°]$ et un site compris entre $[\theta_t - 1{,}25°,\theta_t + 1{,}25°]$ sont donc sélectionnés. Au final, la base d'apprentissage contient donc $N_A = 1\ 024$ profils distance.

La base de test est constituée de $N_T = 150$ profils distance également répartis équitablement dans 3 classes différentes ($N_T^c = 50$ avec $c = 1 \ldots 3$). Les profils distance de la base de test ont tous le même angle d'aspect (α_t, θ_t). Ils ont été obtenus par simulation à partir des profils de la base d'apprentissage auquel un bruit (de type gaussien) a été rajouté en linéaire sur le signal IQ. Contrairement aux profils distance de la base d'apprentissage, les profils distance de la base de test sont donc bruités.

4. Algorithme des K plus proches voisins

L'algorithme des K plus proches voisins (KPPV) est fondé sur une méthode d'apprentissage supervisé (Kantardzic, 2011). Soit $\boldsymbol{x}_T$ et $\boldsymbol{X}_A$, respectivement le profil distance sous test et la base d'apprentissage. $\boldsymbol{x}_T$ est un vecteur de taille N échantillons. $\boldsymbol{X}_A$ est une matrice $N_A \times N$ avec N_A, le nombre de signatures dans la base d'apprentissage. Les différentes étapes de l'algorithme des KPPV sont représentées sur la figure 5.

4.1. Première étape : calcul des distances

La première étape de l'algorithme des KPPV consiste à calculer les distances entre $\boldsymbol{x}_T$ et les N_A profils distance de la base d'apprentissage $\boldsymbol{X}_A$. Dans l'article, deux

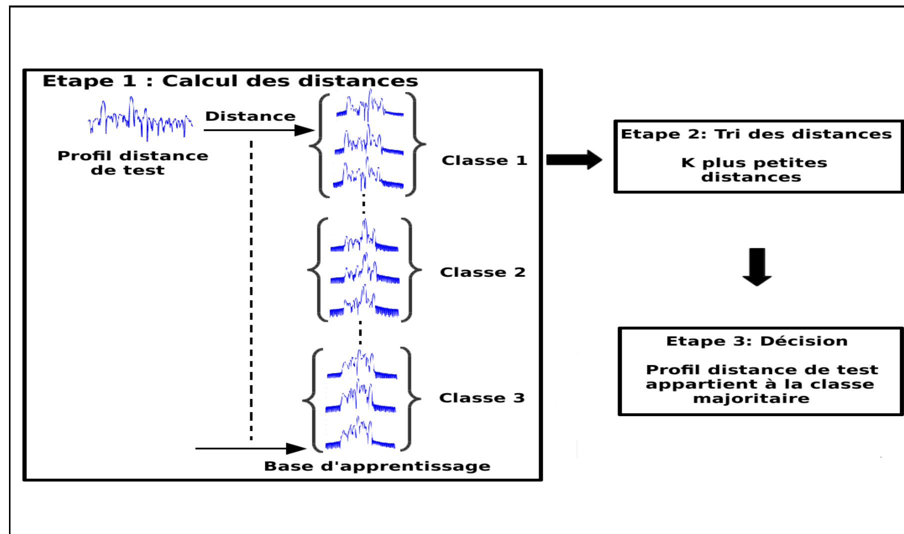

Figure 5. Principe de l'algorithme des KPPV

distances sont étudiées : la distance euclidienne (L2) et la distance de Manhattan (L1). Ces deux distances sont définies de la manière suivante :

– distance euclidienne (L2) : c'est la distance la plus souvent utilisée. Son expression est la suivante :

$$d_{euclidian} = \sum_i (x_i - y_i)^2 \quad (1)$$

– distance de Manhattan (L1) : cette distance permet d'être plus robuste aux valeurs abérrantes. Elle est définie de la manière suivante :

$$d_{manhattan} = \sum_i |x_i - y_i| \quad (2)$$

Lors de l'exécution de l'algorithme des KPPV, une étape de recalage en distance 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, $(2 \times D + 1)$ distances correspondant aux distances entre le profil de la base apprentissage et le profil distance sous test décalé de $\pm D$ cases distance sont calculées. Au final, la distance retenue est la distance minimale parmi ces $(2 \times D + 1)$ distances. Dans notre cas, les profils distance d'apprentissage sont tous recalés entre eux (le nez de la cible est à chaque fois situé dans la même case distance). Pour les profils de test, la position du nez de la cible varie de plusieurs cases distance selon les profils. Théoriquement, le profil distance devrait être décalé de

$\pm D = N/2$ cases distance. En observant les profils distance de test, on s'est aperçu que l'on pouvait se limiter avec notre base de données à un décalage de $\pm D = 50$ cases distance.

4.2. *Deuxième et troisième étapes : tri des distances et décision*

La seconde étape consiste à trier les distances obtenues. Soit $\boldsymbol{d}$ le vecteur contenant les K plus petites distances.

Dans l'étape de décision, le nombre d'occurences de chacune des trois classes que contient le vecteur $\boldsymbol{d}$ est déterminé. Le profil distance sous test est classé dans la classe majoritaire parmi les K plus proches voisins. En cas d'égalité, le profil distance sera affecté à la classe possédant le voisin le plus proche du profil distance sous test.

4.3. *Paramètres de réglage*

A partir de cette description, il apparaît que trois paramètres peuvent influencer le résultat de l'algorithme des KPPV :

– le nombre de plus proches voisins K

– le type de distance utilisé (distance L1 ou L2 dans notre cas)

– la nature des signatures utilisées et les prétraitements : nous avons étudié l'influence de prétraitements sur les profils distance et notamment l'influence du débruitage des profils distance de test sur les performances de l'algorithme. Comme on l'a mentionné dans la section 3, les profils distance de test sont des données bruitées. Pour limiter l'influence du bruit, il nous faut appliquer une étape de débruitage. Cette étape de débruitage consiste tout simplement à seuiller les profils distance de test. Les profils distance d'apprentissage sont également seuillés avec le même seuil au moment du calcul des distances lors de l'exécution de l'algorithme des KPPV. Pour calculer le seuil, le signal utile se distinguant assez facilement du bruit sur un profil distance grâce au gain de compression d'impulsion, il suffit donc de calculer la puissance du bruit et d'appliquer un seuil relativement à cette puissance calculée.

5. Performances de l'algorithme des KPPV

Avant d'exposer les résultats de l'algorithme des KPPV, une étude des signaux de la base d'apprentissage et de test peut être intéressante et peut nous permettre d'estimer les performances atteignables.

5.1. *Caractérisation des signaux de la base d'apprentissage et de test*

La figure 6 montre la superposition des profils distance de la base d'apprentissage et de la base de test pour chacune des trois classes. Cette simple superposition nous permet d'avoir une idée de la forme des signaux.

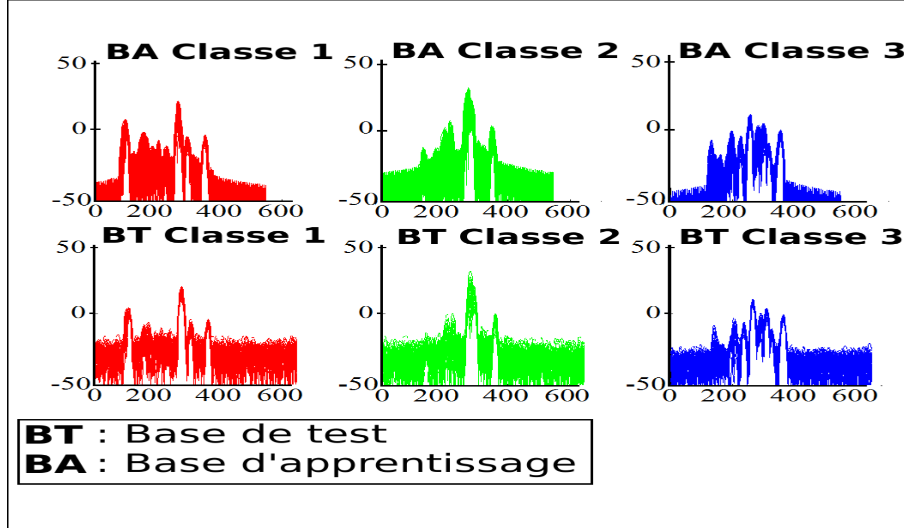

Figure 6. Superposition des profils distance recalés de chaque classe de la base d'apprentissage (BA) en haut et de la base de test (BT) en bas

On peut s'intéresser également plus précisément aux propriétés statistiques des bases de test et d'apprentissage pour chacune des classes. Les notations utilisées pour la base d'apprentissage sont résumées dans le tableau 1. Les notations pour la base de test ne sont pas détaillées car elles sont similaires à celles utilisées pour la base d'apprentissage (indice A pour la base d'apprentissage, indice T pour la base de test).

Tableau 1. Notations utilisées pour la base d'apprentissage

C	Nombre de classes
N_A	Nombre total de profils distance dans la base d'apprentissage.
N_A^c	Nombre de profils distance de la base d'apprentissage appartenant à la classe c.
$\boldsymbol{X}_A$	Matrice des profils distance de la base d'apprentissage.
$\boldsymbol{x}_{A,i}$	i-ème profil distance de la base d'apprentissage avec $i = [1, \ldots, n_A]$.
$\boldsymbol{X}_A^c$	Matrice des profils distance de la base d'apprentissage appartenant à la classe c.
$\boldsymbol{x}_{A,i}^c$	i-ème profil distance de la base d'apprentissage appartenant à la classe c, avec $c = \{1, \ldots, C\}$ et $i = [1, \ldots, n_A^k]$.
$\boldsymbol{m}_A^c$	Profil distance moyen de la classe c de la base d'apprentissage.
$\boldsymbol{C}_A^c$	Matrice de covariance intra-classe de la classe c de la base d'apprentissage.

Le profil distance moyen de la classe c pour la base d'apprentissage est défini de la manière suivante :

$$\boldsymbol{m}_A^c = \frac{1}{N_A^c} \sum_{i=1}^{N_A^c} \boldsymbol{x}_{A,i}^c \tag{3}$$

Les éléments de la matrice de covariance $\boldsymbol{C}_A^c$ sont définis de la manière suivante :

$$\boldsymbol{C}_A^c(i,j) = \frac{1}{N_A^c}(\boldsymbol{x}_{A,i}^c - \boldsymbol{m}_A^c)^T(\boldsymbol{x}_{A,j}^c - \boldsymbol{m}_A^c) \tag{4}$$

5.1.1. Propriétés statistiques des matrices de covariance

Une étude des propriétés statistiques des matrices de covariance intra-classe ($\boldsymbol{C}_A^1$, $\boldsymbol{C}_A^2$, $\boldsymbol{C}_A^3$ et $\boldsymbol{C}_T^1$, $\boldsymbol{C}_T^2$, $\boldsymbol{C}_T^3$) est proposée pour le cas où les profils distance sont récalés et pour les deux configurations suivantes : (i) profils distance seuillés et (ii) profils distance non seuillés.

Cette étude s'articule autour de l'estimation de trois grandeurs : (i) les rangs des matrices de covariance (Rg), (ii) les traces des matrices de covariance (Tr) et (iii) les valeurs propres des matrices de covariance. Les rangs (nombre maximal de vecteurs lignes (ou colonnes) linéairement indépendants) de ces matrices sont calculés afin d'avoir une idée plus précise de la dimension réelle des données. Les traces et les valeurs propres de ces matrices nous renseignent quant à elles sur le volume dimensionnel des différentes classes. Pour les valeurs propres, on tracera le spectre associé et on calculera le produit des deux valeurs propres les plus grandes que l'on notera par la suite Π_{12}.

Les tableaux 2 et 3 résument l'ensemble de cette étude pour les profils distance de la base d'apprentissage et de test seuillés ou pour les profils distance de la base d'apprentissage et de test non-seuillés.

Tableau 2. Caractérisation statistique des profils distance de la base d'apprentissage

	Seuillés			Non-seuillés		
	Tr	Π_{12}	Rg	Tr	Π_{12}	Rg
C_A^1	2,08e3	2,23e5	120	1,42e4	5,24e6	260
C_A^2	2,98e3	9,45e5	92	2,40e4	44,9e6	196
C_A^3	1,43e3	1,16e5	97	1,05e4	2,29e6	231

Tableau 3. Caractérisation statistique des profils distance de la base de test

	Seuillés			Non-seuillés		
	Tr	Π_{12}	Rg	Tr	Π_{12}	Rg
C_T^1	4,61e3	2,04e5	49	1,60e4	1,56e6	49
C_T^2	5,89e3	7,87e5	49	2,14e4	6,48e6	49
C_T^3	1,36e3	7,74e4	49	1,46e4	1,13e6	49

Pour la base d'apprentissage, on peut remarquer que la classe 2 semble être la plus étendue au regard des valeurs des traces et de Π_{12} pour chaque classe. Les valeurs des rangs nous permettent de voir également que la dimension réelle des données peut être réduite. On remarque également que le volume des différentes classes diminue lorsque l'on applique un seuillage sur les profils distance, ce qui peut permettre de mieux

discriminer les différentes classes entre elles. Le rang des matrices de covariance est également nettement plus faible dans le cas où les profils distance sont seuillés.

Pour la base de test, ces observations restent presque toutes valables. En revanche, on remarque que les matrices de covariance de la base de test restent quasiment de rang plein. Cela peut s'expliquer par le fait que les profils distance de test, contrairement à ceux de la base d'apprentissage, sont bruités. Même si le seuillage limite l'influence du bruit dans la partie « non utile » du signal, le bruit reste présent sur les échantillons non seuillés du signal et cela influe sur la valeur du rang. De plus, le nombre de profils distance dans la base de test est largement moins important que celui de la base d'apprentissage, ce qui peut expliquer que les matrices de covariance soient quasiment de rang plein car les possibilités de combinaisons linéaires entre les lignes ou les colonnes sont beaucoup moins importantes.

On peut tracer les spectres des valeurs propres pour les deux configurations vues jusqu'à présent : profils distance seuillés ou non-seuillés. Le spectre des valeurs propres représente le pourcentage d'inertie des valeurs propres de la plus grande à la plus petite. L'inertie $I^c_{A,i}$ de la valeur propre λ_i de la matrice de covariance $\boldsymbol{C}^c_A$ est définie de la manière suivante :

$$I^c_{A,i} = \frac{\lambda_i}{\sum_{j=1}^{N^c_A}(\lambda_j)} \tag{5}$$

L'inertie d'une valeur propre peut être vue comme la contribution de cette valeur propre à l'inertie totale de la matrice de covariance. L'inertie est donc une grandeur comprise entre 0 et 1. Le pourcentage d'inertie est lui compris entre 0 % et 100 %. La figure 7 représente les sommes cumulées pour les 50 premières valeurs propres pour la base d'apprentissage et la base de test.

La principale conclusion que l'on peut tirer de ces spectres est qu'une grande partie de l'information se concentre uniquement sur quelques valeurs propres ou direction dans l'espace et cela quelle que soit la classe. L'idée de caractériser le volume des données à partir des deux plus grandes valeurs propres peut se justifier à partir de ces spectres. Lorsque les profils distance sont seuillés et récalés, l'information contenue sur les deux plus grandes valeurs propres varie de 50 % à 60 % selon les classes pour la base d'apprentissage et de 30 % à 50 % selon les classes pour la base de test. Certes en ne considérant pas une partie de l'information, la caractérisation des données n'est qu'approximative. Cependant, elle permet d'obtenir simplement une première visualisation de la répartition des classes à l'intérieur de la base d'apprentissage et de test et donc d'obtenir une première approximation des performances atteignables avec ce jeu de données.

5.1.2. Représentation graphique des données

On peut calculer la distance entre les profils distance moyens des classes 1, 2 et 3 (cf. tableaux 4 et 5). La dernière ligne de ces deux tableaux est une représentation graphique des grandeurs calculées jusqu'à présent pour cette section 5.1. Il s'agit en

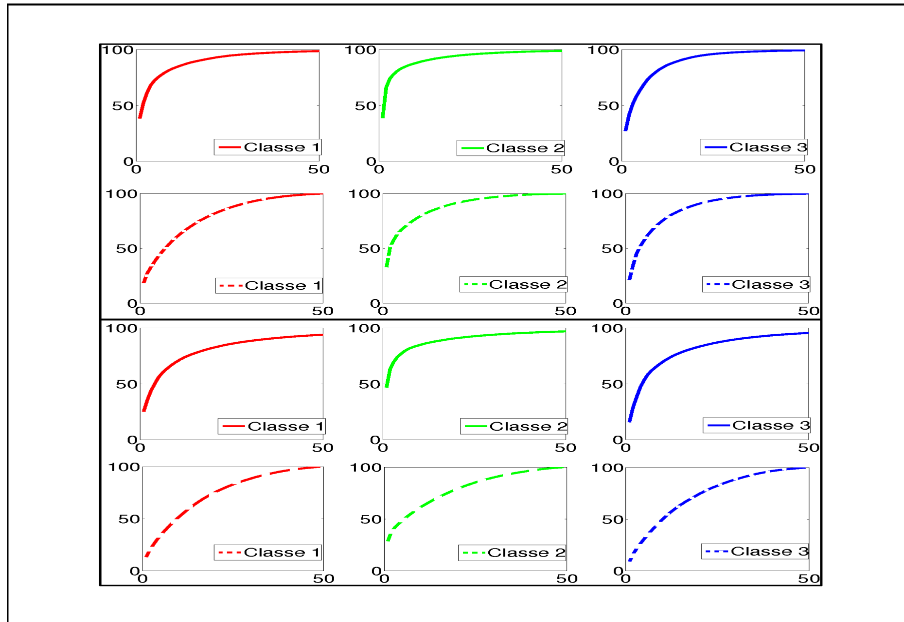

Figure 7. Spectre des 50 premières valeurs propres de la base d'apprentissage (trait continu) et de la base de test (trait pointillé) pour des profils distance seuillés (en haut) et non seuillés (en bas)

fait de représenter les centres des trois classes sur un triangle dont les cotés sont de tailles égales aux distances calculées dans les tableaux 4 et 5. Comme on l'a déjà mentionné avant, on considère que l'information de volume des différentes classes peut être obtenue à partir des deux plus grandes valeurs propres. La classe est donc caractérisée par une ellipse avec comme valeur de grand axe la première valeur propre et comme valeur de petit axe la seconde valeur propre. La direction du grand axe de l'ellipse est donnée par la direction du vecteur propre correspondant à la plus grande valeur propre. On fait l'hypothèse que les vecteurs propres des deux plus grandes valeurs propres sont orthogonaux. On peut définir la distribution associée de la manière suivante :

$$f_i(\boldsymbol{x}) = 2\pi|\boldsymbol{\Sigma}_i|e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{m}_i)^T\boldsymbol{\Sigma}_i^{-1}(\boldsymbol{x}-\boldsymbol{m}_i)} \tag{6}$$

avec :

$$\boldsymbol{\Sigma}_i^{-1} = \begin{pmatrix} \lambda_i^1 & 0 \\ 0 & \lambda_i^2 \end{pmatrix} \tag{7}$$

et m_i la moyenne de la classe i et λ_i^1 et λ_i^2 les deux plus grandes valeurs propres de la classe i.

La distribution associée à chaque classe est donc tracée sur la dernière ligne des tableaux 4 et 5.

Tableau 4. Distance euclidienne normalisée entre les profils distance moyens de la classe 1,2 et 3 de la base d'apprentissage

	Seuillés	Non-seuillés
	Distance L2 entre les moyennes	Distance L2 entre les moyennes
$1 \leftrightarrow 2$	$1{,}50e4$	$5{,}87e4$
$1 \leftrightarrow 3$	$1{,}76e4$	$10{,}0e4$
$2 \leftrightarrow 3$	$1{,}94e4$	$6{,}10e4$

Tableau 5. Distance euclidienne normalisée entre les profils distance moyens de la classe 1,2 et 3 de la base de test

	Seuillés	Non-seuillés
	Distance L2 entre les moyennes	Distance L2 entre les moyennes
$1 \leftrightarrow 2$	$9{,}95e3$	$1{,}46e4$
$1 \leftrightarrow 3$	$1{,}67e4$	$4{,}13e4$
$2 \leftrightarrow 3$	$1{,}.71e4$	$3{,}45e4$

On observe sur les tableaux 4 et 5 que les classes semblent relativement bien séparées même dans le cas où les profils distance ne sont pas seuillés. Cela peut paraître

normal car ce n'est pas l'opération de seuillage qui va permettre d'augmenter la distance entre les centres des différentes classes. On remarque cependant que le rapport des distances entre les centres de classes évoluent lorsque les profils distance sont seuillés, le triangle obtenu à partir des trois centres de classes se rapproche d'un triangle isocèle alors que dans le cas non-seuillés, celui-ci est beaucoup plus quelconque. Afin de se faire une idée plus précise des performances atteignables, il peut être intéressant de superposer les distributions des différentes classes pour la base d'apprentissage et de test. Plus précisemment, nous superposons sur la figure 8, les ellipses à 95 % des distributions des classes 1, 2 et 3 pour les bases d'apprentissage et de test dans le cas seuillé et non-seuillé. Pour superposer les deux courbes, on place les deux triangles de manière à ce que leurs centres de gravité soit l'origine du repère. Ensuite pour pouvoir exploiter cette figure correctement, on cherche la position d'équilibre de ces deux triangles dans le repère (X,Y). Il s'agit d'un problème de mécanique qui consiste à trouver la position du triangle pour laquelle la somme des moments en son centre de gravité est nulle, en considérant trois forces de norme équivalente s'exerçant au trois sommets et dirigées en -Y. Nous n'entrerons pas plus dans les détails car ce n'est pas l'objet de l'article.

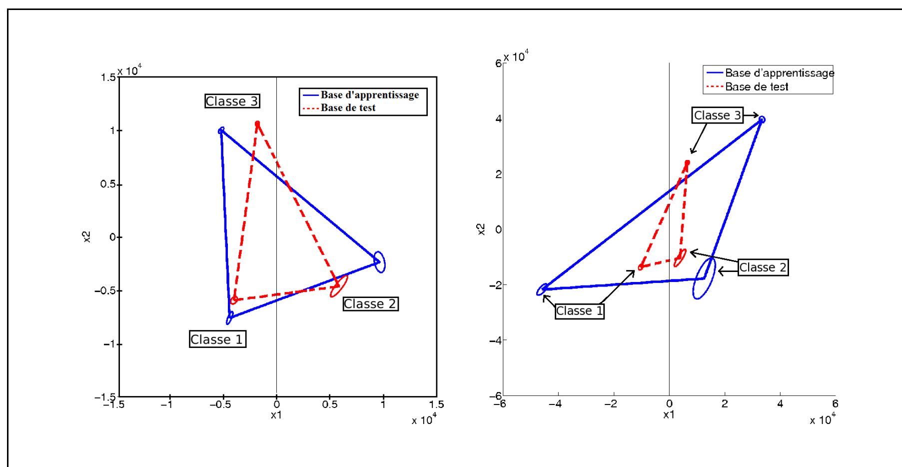

Figure 8. Superposition ellipse à 95 % pour la base de test et la base d'apprentissage avec des profils distance seuillés (à gauche) et non seuillés (à droite)

On peut voir sur la figure 8 que les classes semblent bien séparées pour la base d'apprentissage et la base de test. Il n'y a pas d'intersection entre les ellipses des différentes classes que ce soit pour la base de test ou d'apprentissage. En revanche, il est intéressant de noter que, contrairement au cas seuillé, dans le cas non-seuillé les triangles de la base de test et d'apprentissage ne sont pas de tailles équivalentes. Dans le cas seuillé, même si les échantillons contenus dans les ellipses à 95 % de la base de test et d'apprentissage ne se superposent pas, ils restent néanmoins plus proches de leurs classes respectives que des autres classes. En utilisant un algorithme comme l'algorithme des K plus proches voisins, cela ne devrait pas affecter les performances de reconnaissance. Au regard de ces observations, on peut se dire que si erreur il y a,

dans le cas seuillé, elles auront plus de chances d'intervenir entre les profils distance de la classe 1 et 2 car ce sont les 2 ellipses les plus proches. Cependant, on peut tout de même dire qu'il semble possible d'obtenir un taux d'erreur inférieur à 5 % (les ellipses à 95 % des différentes classes ne se superposent pas). Dans le cas non-seuillé, on aurait tendance à dire à partir de cette représentation graphique que les échantillons de la classe 1 et 2 seront classés dans la classe 2 (ellipses des classes 1 et 2 de la base de test proches de l'ellipse de la classe 2 de la base d'apprentissage) et ceux de la classe 3 dans la classe 3 (ellipse de la classes 3 de la base de test proche de l'ellipse de la classe 3 de la base d'apprentissage). En conclusion, on peut se faire une idée un peu plus précise des performances atteignables à partir de ce type de représentation. On peut prédire que les performances avec l'algorithme des KPPV en utilisant des profils distance seuillés devraient être plutôt bonnes avec un taux d'erreur faible (inférieur à 5 %) alors que sans seuillage des profils distance, on peut s' attendre à des performances médiocres. La prochaine section va nous permettre de valider ou non ces prédictions.

5.2. Performance en termes de taux d'erreur et taux de succès

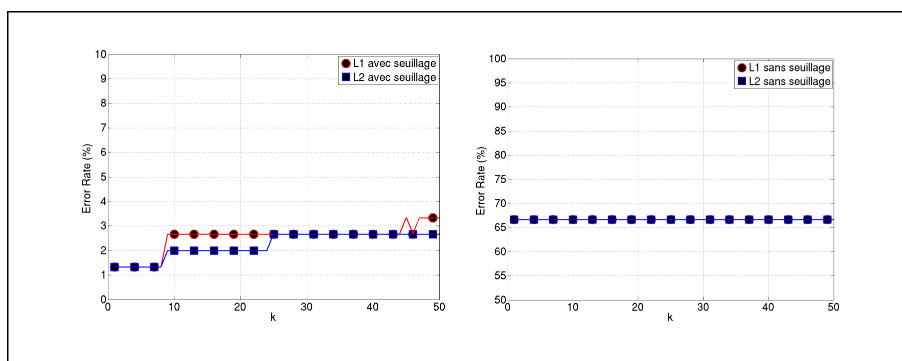

Figure 9. Taux d'erreur en fonction de K pour l'algorithme des KPPV avec et sans seuillage des profils distance

Comme on l'a vu dans la section 3, trois paramètres peuvent influencer les performances de l'algorithme des KPPV. Pour évaluer les performances, nous traçons le taux d'erreur et le taux de succès en fonction de K avec les profils seuillés et non seuillés et pour les distances L1 ou L2. On a une erreur lorsque un profil de la classe c n'est pas reconnu comme étant de la classe c. On a un succès lorsqu'un profil de la classe c est reconnu comme étant uniquement de la classe c. Il convient de faire remarquer que les échelles des courbes du taux d'erreur avec seuillage et sans seuillage sont différentes, afin de rendre les résultats plus lisibles.

La figure 9 montre clairement que les profils distance ne sont pas exploitables lorsque l'on ne réduit pas l'influence du bruit en seuillant nos profils distance. Le taux d'erreur dans ce cas est d'environ 66 % quelle que soit la valeur de K et quelle que soit

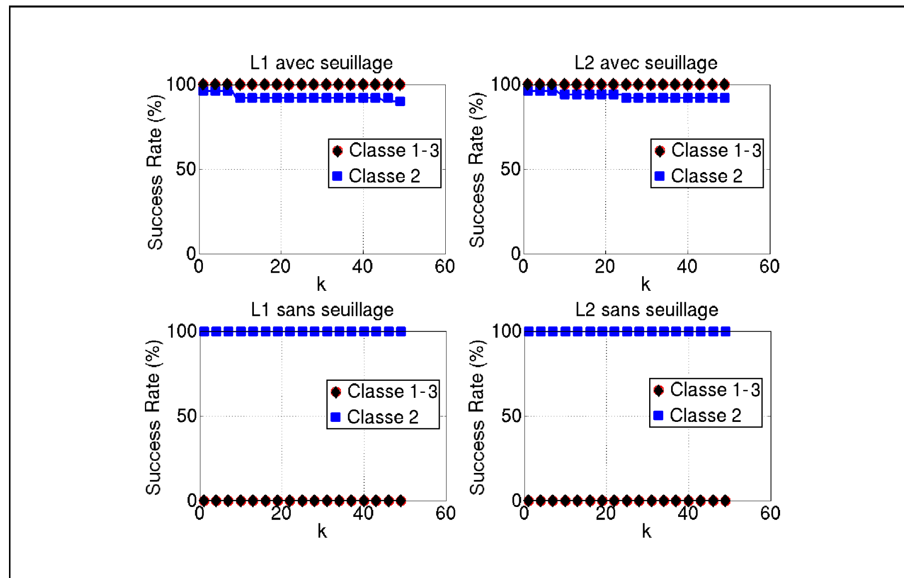

Figure 10. Taux de succès en fonction de K pour l'algorithme des KPPV avec et sans seuillage des profils distance

la distance utilisée. Cependant, lorsqu'on applique un simple seuillage sur les profils distance, de manière à éliminer les échantillons de bruit, le taux d'erreur devient tout à fait acceptable, variant entre 1 % et 3 % suivant la valeur de K pour la distance L1 et L2. On se rend donc compte que l'étape de seuillage est indispensable pour pouvoir obtenir de bonnes performances avec notre algorithme.

On peut observer sur la figure 10 que le taux de succès est nul pour les classes 1 et 3 lorsque les profils distance ne sont pas seuillés alors qu'il est de 100 % pour la classe 2. Par contre lorsque l'on seuille les profils distance, le taux de succès est de 100 % pour les classes 1 et 3 et légèrement inférieur à 100 % pour la classe 2 et cela quelle que soit la valeur de K.

Comme on pouvait l'imaginer en regardant l'étude statistique des données, les performances de l'algorithme des KPPV (cf. tableau 6) sont très bonnes à condition que les profils distance soit préalablement recalés et seuillés. Les seules confusions ont lieu entre les profils distance de la classe 1 et 2, et plus précisément, ce sont 4 % des profils distance de la classe 2 qui sont classés par erreur dans la classe 1, comme on avait pu le prédire à partir de l'étude statistique. On peut également remarquer que le taux d'erreur reste inférieur à 5 % quelle que soit la valeur de K.

Dans le cas où les profils distance ne sont pas seuillés, comme on pouvait s'y attendre les profils distance de la classe 1 et 2 de la base de test sont classés dans la classe 2. Par contre, on se rend compte que les profils distance de test de la classe 3 sont également classés dans la classe 2 alors qu'en observant la représentation graphique

Tableau 6. Matrice de confusion

	Classe	L1			L2		
		1	2	3	1	2	3
Base de test (seuillée)	1	100 %	0 %	0 %	100 %	0 %	0 %
	2	4 %	96 %	0 %	4 %	96 %	0 %
	3	0 %	0 %	100 %	0 %	0 %	100 %
Base de test (non seuillée)	1	0 %	100 %	0 %	0 %	100 %	0 %
	2	0 %	100 %	0 %	0 %	100 %	0 %
	3	0 %	100 %	0 %	0 %	100 %	0 %

on aurait pu s'attendre à ce qu'ils soient classés dans la classe 3. En effet, sur la figure 8, la classe 3 de la base de test se trouve plus proche de la classe 3 de la base d'apprentissage que de la classe 2. Cependant, la différence de distance est très proche et compte tenu de l'approximation faite pour construire la représentation graphique (sur la figure 7, on se rend compte que l'inertie sur les deux plus grandes valeurs propres pour la classe 3 est de l'ordre de 25 % ce qui est plus faible que pour les 2 autres classes où on est plutôt de l'ordre de 35 % pour la classe 1 et de 50 % pour la classe 2), on peut comprendre que les profils distance de test de la classe 3 soient classés dans la classe 2.

6. Parallélisation sur GPU

Le but de cette section est d'étudier les performances en termes de temps de calcul de notre algorithme (toutes nos implémentations sont réalisées en « single precision », ce qui est suffisant pour notre application). L'étape la plus coûteuse dans l'algorithme des KPPV est l'étape de calcul des distance associée à l'étape de recalage et de seuillage des profils distance. Le reste des calculs est négligeable en termes de temps de calcul.

L'algorithme a été implémenté dans un premier temps dans l'environnement Matlab (version 2012b) de manière vectorisée ou matricielle afin d'optimiser le temps de calcul. Ensuite, différentes étapes d'implémentation sur GPU sont présentées, chacune d'entres elles présentant un niveau d'optimisation de plus en plus poussé. Il va de soit que plus le niveau d'optimisation est important et plus le temps passé pour l'implémenter est important. Nous nous sommes orientés vers une implémentation GPU et nous avons délibérement opté d'étudier en détail les différents points d'optimisation permettant d'améliorer le temps de calcul sur GPU. Les comparaisons en termes de temps de calcul sur CPU (Matlab) et sur GPU sont donc à relativiser car le travail d'optimisation sur CPU et GPU est déséquilibré. Le but est d'obtenir les performances maximum sur un GPU pour atteindre notre objectif de traitement en temps-réel.

Dans cette section, après une présentation des opérations nécessaires au calcul des distances, au recalage et au seuillage et des performances dans l'environnement Matlab, nous introduisons les notions importantes à maîtriser (latence, occupatation, etc.)

lorsque l'on souhaite paralléliser de manière efficace sur GPU. Nous présentons ensuite plusieurs versions implémentées sur GPU, chacune d'entre elles mettant en avant les concepts importants qui ont été utilisés pour améliorer le temps de calcul sur GPU (alignement mémoire, utilisation de la mémoire « shared », parallélisme d'instruction et réutilisation des données). Cette présentation est suivie d'une synthèse des temps de calcul obtenus avec ces différentes versions implémentées en CUDA C++. La section se terminera par une comparaison entre les performances obtenues avec CUDA ou OpenCL et une présentation de l'intégration dans l'environnement Matlab des calculs effectués sur GPU en CUDA ou OpenCL (utilisation des mex-files (NVIDIA, 2007)).

6.1. Nombre d'opérations à virgule flottante

Le nombre d'opérations à virgule flottante (N_{FLO}) nécessaires pour le calcul des distances, le recalage et le seuillage des profils distance est de $N_{FLO} = N_T \times (2 \times D + 1) \times (N_A \times N \times 3)$ soit environ 25 *Giga Floating Point Operations*.

En effet, pour chaque case distance, on effectue 3 opérations à virgule flottante pour le calcul de la distance L2 : (i) soustraction de l'échantillon de test et d'apprentissage, (ii) mise au carré, (iii) accumulation dans la valeur sum (cf. figure 11). Le pseudo-code du kernel utilisé pour calculer la distance entre le profil distance de test k et le profil distance d'apprentissage j est exposé sur la figure 11.

for $dec \leq 2 \times D$ **do**
    for $i \leq N$ **do**
        $sum = sum + distance(X_T(k,i + dec),X_A(j,i))$
    end for
end for

Figure 11. Pseudo-Code du « kernel »

Les performances pic des GPU NVIDIA que nous utilisons dans notre étude sont obtenues dans le cas idéal où l'on fait un **fma**(fused multiply add) par cycle. Or les opérations de notre algorithme correspondent en assembleur à un **sub** suivi d'un **fma**. Nous pourrons donc atteindre au mieux que 75 % du pic [2].

6.2. Temps de calcul sous Matlab

Sous Matlab, nous obtenons un temps de calcul de 1,327 s (cf. tableau 7). Le temps de calcul par profil distance de test est de l'ordre de $1327/150 \approx 9$ ms. La contrainte

2. La puissance pic est obtenue lorsqu'on exécute 1 **fma** par cycle. Un **fma** est compté comme 2 opérations flottantes. Idéalement en 2 cycles on peut faire 2 **fma** donc 4 opérations flottantes. Avec notre algorithme, on est obligé de faire 1 **fma** + 1 **sub** soit 3 opérations flottantes en 2 cycles au lieu de 4 idéalement. D'où les 75 % de la puissance pic.

de temps-réel que nous nous sommes fixée pour notre application étant de 1 ms par profil distance, l'implémentation sous Matlab ne nous permet donc pas d'atteindre cet objectif. A noter que l'utilisation de la *Parallel Computing Toolbox* de Matlab permet de réduire d'un facteur 2 le temps de calcul. Cela ne permet toujours pas de respecter la contrainte temps-réel. Nous nous sommes donc orientés vers une parallélisation sur GPU hors de l'environnement Matlab.

Tableau 7. Temps de calcul (en ms) de l'étape de calcul des distance, recalage et seuillage de l'algorithme des KPPV exécuté en « single precision » sous Matlab

	Temps(ms)	GFlops
Matlab	1327	19

CPU:Intel Xeon à 2.40GHz

6.3. Découpage en thread sur l'architecture many-core GPU

6.3.1. Modèle de programmation sur GPU

Il existe deux principaux modèles de programmation sur GPU : (i) l'architecture CUDA (*Common Unified Device Architecture*) et (ii) l'architecture OpenCL (*Open Computing Language*).

L'architecture CUDA est un modèle de programmation SIMT (*Simple Instruction Multiple Threads*) des architectures « many cores » du fabriquant NVIDIA (*NVIDIA Programming Guide*, 2011). On définit un « kernel » comme étant le programme à exécuter en parallèle sur les cœurs du 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 afin de correspondre à l'architecture par blocs de cœurs de calcul, appelés « Streaming Multiprocessor » (SM), des GPU (32 cœurs par blocs pour l'architecture Fermi). Ils sont en effet regroupés en « blocs de threads », eux mêmes regroupés en « grilles de blocs de threads ». Avec l'architecture Fermi, le nombre de threads par bloc peut aller jusqu'à 1 024. N'importe quel programme CUDA peut être divisé en deux parties : le code exécuté sur le GPU et le code exécuté sur le CPU. Ce dernier est en grande partie dédié aux transferts de mémoire entre GPU et CPU.

OpenCL reprend les grands principes du modèle de programmation CUDA (kernel, thread, bloc de thread, etc.). Cependant, OpenCL a des objectifs plus larges car le modèle n'est pas uniquement dédié aux GPU. OpenCL permet notamment de tirer parti de la puissance des GPU, des CPU multi-cœurs ou d'autres systèmes de calcul intensifs tels que le CELL par exemple d'IBM et tout cela via une infrastructure de programmation unique.

6.3.2. Génération Tesla vs génération Fermi

Les cartes GPU utilisées pour cette étude sont des cartes NVIDIA de deux générations différentes : (i) la NVIDIA Tesla C2050 de génération Fermi (1,03 TFlops) et (ii)

la NVIDIA Tesla C1060 de génération Tesla (622 GFlops). La principale innovation des GPU de la génération Fermi est la présence des caches L1 et L2 et une augmentation de la taille de la mémoire « shared ». Avec la génération Fermi, chaque Streaming Multiprocessor (constitué de 32 cœurs de calcul) dispose de $64KB$ de RAM configurable entre la mémoire « shared » et le cache L1. Lorsque par exemple, on utilise $48KB$ de mémoire « shared », il reste $16KB$ de mémoire pour le cache L1 et vice versa. Une autre amélioration est l'ordonnancement des warps [3] exécutés par les Streaming Multiprocessor (SM). Avec la génération Fermi, chaque SM possède deux ordonnanceurs de warp et deux unités de distribution d'instructions, ce qui permet de traiter et d' exécuter deux warps en parallèle, ce qui n'était pas le cas avec les générations antérieures.

6.3.3. Parallélisation des KPPV

L'algorithme des KPPV est un algorithme massivement parallèle. Chaque distance entre un profil distance de test et un profil distance d'apprentissage peut être calculée en parallèle de manière indépendante. L'étape de recalage ne fait qu'augmenter le nombre de distances à calculer et le seuillage ne nécessite que le calcul d'un maximum entre deux floats pour chaque échantillon des profils distance. L'utilisation des GPU pour améliorer le temps de calcul de l'algorithme des KPPV semble donc être une solution bien adaptée. Pour paralléliser le calcul, on va donc créer $N_{blocks} = 150$ blocs de $N_{threads/block} = 1\ 024$ threads, ce qui correspond aux $N_T = 150$ profils distance de la base de test et aux $N_A = 1\ 024$ profils distance de la base d'apprentissage. Nous verrons dans la suite de l'article que le nombre $N_{threads/block}$ peut évoluer de manière à optimiser les performances.

6.3.4. Architecture mémoire

Quel que soit le modèle de programmation, les performances d'un programme GPU sont très influencées par la gestion mémoire (Kirk, Hwu, 2010). La mémoire est organisée hiérarchiquement et 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 », et sur l'architecture Fermi des caches L1 et L2 standard. Il convient également de faire remarquer que, dans notre cas, les problèmes de transferts de mémoire entre la carte graphique et le PC qui peuvent potentiellement créer un goulot d'étranglement ne nous concernent pas. En effet, la capacité mémoire des cartes graphiques (1 à 4 Go) permet de stocker entièrement la base d'apprentissage sur la carte graphique et peut être chargée offline.

3. Un warp est un groupe de 32 threads parallèles.

6.4. *Latence, occupation et parallélisme*

Un des points clés d'une implémentation sur GPU est de « cacher » au maximum la latence d'accès aux ressources de calcul (latence arithmétique) ou aux ressources mémoire (latence mémoire). Il s'agit donc de lancer un maximum d'instructions au cours des temps d'attente.

6.4.1. Latence mémoire et arithmétique

La latence arithmétique est le temps nécessaire pour réaliser une instruction (environ 60 cycles pour un opération arithmétique flottante (Collange, 2010)). Il est également courant que plusieurs instructions soient dépendantes les unes des autres (une instruction a besoin du résultat d'une autre instruction pour s'exécuter). Il est donc indispensable, si possible, de « cacher » ces temps d'attente en lançant d'autres instructions (sans dépendance avec les instructions précédemment lancées).

La latence mémoire est le nombre de cycles nécessaires pour accéder à une donnée en mémoire. Cette latence varie en fonction de l'endroit où sont situées ces données. Pour des données en mémoire globale, le nombre de cycle est d'environ 400 à 800 cycles sans cache (Volkov, 2010). En utilisant les caches « constante », « texture », L1/L2, la mémoire « shared » ou les registres, plus que quelques cycles sont nécessaires pour accéder aux données.

6.4.2. Accès rapide à la mémoire

Un premier axe d'optimisation consiste à utiliser au maximum les registres puis la mémoire shared et enfin les caches « constante », « texture » ou L1/L2 sur Fermi afin de diminuer la latence mémoire. Par exemple, dans toutes les versions implémentées dans la suite de l'article, les valeurs des seuils appliqués aux profils distance sont stockées en mémoire globale. L'accès à ces données en mémoire globale sera optimisé en utilisant le cache « constante ». Dans cette étude, nous n'utilisons pas le cache « texture » car celui-ci n'est pas adapté à notre application. En effet, nos données n'ont pas de localité spatiale 2D donc le cache 2D n'est pas intéressant dans notre cas. L'utilisation d'un cache 1D n'est pas possible car sa taille n'est pas suffisante pour « couvrir » l'ensemble de nos données.

6.4.3. Parallélisme et occupation

Un deuxième axe consiste à « cacher » la latence restante par l'exécution d'autres instructions lors de la latence. Deux démarches principales existent pour traiter cela : (i) le parallélisme de threads ou *Thread-Level Parallelism* (TLP) et (ii) le parallélisme d'instructions ou *Instruction-Level Parallelism* (ILP).

Le but du TLP est d'augmenter l'occupation en lançant le plus de threads de manière à exécuter un maximum d'instructions et à maximiser le débit de calcul. L'occupation est le nombre de warps actifs sur le nombre de warps actifs maximum. Avec la carte Fermi, le nombre maximum de warps actifs est de 48 et avec la carte non-Fermi, ce nombre est de 32. L'occupation est limitée lorsque trop de ressources sont allouées

par thread. Il existe quatre facteurs principaux qui peuvent limiter l'occupation : (i) le nombre maximum de blocs par SM, (ii) le nombre maximum de warps actifs par SM (iii) l'utilisation des registres, (iv) l'utilisation de la mémoire « shared ».

Tableau 8. Facteurs limitant l'occupation avec les cartes Fermi et non-Fermi

	Fermi	Non-Fermi
① Nombre maximum de blocs par SM	8	8
② Nombre maximum de warps actifs par SM	$48K$	$32K$
③ Taille maximum de la mémoire registre par SM	$32Ko$	$16Ko$
④ Taille maximum de la mémoire « shared » par SM	48	16

Tableau 9. Calcul de l'occupation en fonction des facteurs limitant pour différentes valeurs de nombre de threads par bloc sur une carte Fermi

	Nb threads/bloc (warps/bloc)	1 024 (32)	256 (8)	128 (4)
	Memoire registre/bloc (2Ko registre/thread)	24 Ko	6 Ko	3 Ko
①	Nb max blocs/SM	8	8	8
②	Nb blocs/SM (Nb max de warps actifs/SM)	$\lfloor \frac{48}{32} \rfloor = 1$	$\lfloor \frac{48}{8} \rfloor = 6$	$\lfloor \frac{48}{4} \rfloor = 12$
③	Nb blocs/SM (Taille max mémoire registre/SM)	$\lfloor \frac{32}{24} \rfloor = 1$	$\lfloor \frac{32}{6} \rfloor = 5$	$\lfloor \frac{32}{3} \rfloor = 10$
④	Nb blocs/SM (Taille max mémoire « shared »/SM)	/	/	/
	Min(①,②,③,④) (Warps/SM)	1 (32)	5 (40)	8 (32)
	Occupation	66 %	83 %	66 %

Le symbole $\lfloor \ \rfloor$ représente la division entière

L'approche consiste à jouer sur le nombre de threads par bloc pour trouver le meilleur compromis entre les quatre facteurs limitant (cf. tableau 8) de manière à lancer un maximum de blocs par SM et à augmenter l'occupation. Le tableau 9 résume cela en prenant l'exemple des versions 1 et 2 du tableau 10 sur une carte Fermi. Dans ces deux versions, la mémoire « shared » n'est pas utilisée elle n'est donc pas limitante, cependant elle le sera pour les autres versions du tableau 10. On se rend compte qu'un nombre de blocs par SM peut être calculé en tenant compte de chaque facteur limitant. L'occupation est calculée à partir du minimum sur les nombres de blocs par SM obtenus pour chaque facteur limitant. Pour chaque version du tableau 10, un tableau de ce type a été construit de manière à trouver le nombre de threads par bloc qui maximise l' occupation.

La deuxième approche pour « cacher » la latence est le parallèlisme d'instructions ou ILP. Il s'agit d'augmenter le nombre d'instructions successives sans dépendance de

données exécutées par un thread, en réalisant par exemple des déroulages de boucles. A nombre de threads équivalents, l'ILP augmente le nombre d'instructions dans le pipeline. Mais en pratique, l'ILP oblige à réduire l'occupation car un thread après déroulage de boucle demandera souvent plus de registres ou de mémoire « shared ». L'ILP permet de « cacher » la latence mémoire ou arithmétique sans augmenter le nombre de threads.

6.5. Implémentation CUDA C++ sur GPU

Dans la suite de l'article, plusieurs implémentations (version 1 à 5 du tableau 10) sur GPU des étapes de calcul des distances, recalage et seuillage sont proposées. Les optimisations proposées à chaque version nous permettront de faire ressortir les principales règles de codage à respecter pour obtenir une implémentation efficace sur GPU.

6.5.1. Alignement des données en mémoire : versions 1 et 2 (V1 et V2)

La version 1 du tableau 10 est une implémentation que l'on peut qualifier de « basique ». En effet, les profils distance de la base de test et de la base d'apprentissage sont directement stockés en mémoire globale sans se préocuper de l'alignement en mémoire de ces données. Cet alignement mémoire est très important pour faire des accès groupés à la mémoire globale et limiter le nombre de transactions mémoire. Les performances obtenues avec cette première version sont donc très mauvaises que ce soit sur la carte Fermi ou non-Fermi et bien moins intéressantes que les performances obtenues avec une version Matlab vectorisée. Les défauts du non alignement en mémoire des données sont mis en évidence dans le tableau 11. Théoriquement, le nombre de chargements mémoire nécessaires est de 62 Go ($(150 \times 2 \times 50 \times 1\,024 \times 546)/8/1\,024^3 \approx 62$ Go). En utilisant, les outils de profiling, on se rend compte qu'avec cette version, 940 Go de chargements mémoire sont en réalité effectués dont une grande partie de lecture directement en mémoire SDRAM (une donnée est lue directement en mémoire SDRAM lorsqu'elle ne se trouve pas dans les caches L1 ou L2). Dans la majorité des cas, chaque thread recharge lui même la donnée et charge plus de données qu'il en a réellement besoin (une transaction mémoire charge 128 octets). On atteint seulement 0,36 % et 0,84 % des performances pics respectivement de la Tesla C2050 et de la Tesla C1060. L'alignement mémoire est le minimum à respecter pour pouvoir exécuter des calculs sur GPU.

La version 2 du tableau 10 profite quant à elle de l'alignement mémoire. Sur le tableau 11, on observe que le nombre de chargements en mémoire a considérablement diminué entre les versions 1 et 2 du fait de l'alignement mémoire. Le simple fait de stocker les données en mémoire de manière intelligente permet d'obtenir un gain important en temps de calcul. Sur la carte Fermi, ce gain est de 28 par rapport à la version 1, ce qui permet d'atteindre 9,9 % des performances pic. Le gain est moins important sur la carte non-Fermi ($\times 8$). Cette différence s'explique en étudiant plus en détail le tableau 11. On se rend compte que la version 2 sur la carte C2050 profite du cache L2 mis en place sur les cartes de la génération Fermi et non présent sur la carte C1060. Le pourcentage de données demandées se trouvant dans le cache L2

est de $76{,}9$ %. Avec la carte C1060, ces données sont chargées directement depuis la mémoire SDRAM sans cache, ce qui augmente considérablement la latence mémoire. Cela accentue la différence en terme de GFlops pic que l'on trouve intrinsèquement entre les deux cartes.

6.5.2. Utilisation de la mémoire « shared » : version 3 (V3)

Les deux premières versions utilisaient la mémoire globale pour stocker l'ensemble des données. Cependant, l'architecture CUDA propose des moyens pour pouvoir accéder aux données plus rapidement. Un des moyens est de stocker des données en mémoire « shared ». La mémoire « shared » est partagée par tous les threads d'un même bloc et peut contenir au maximum $48KB$ de données avec la génération Fermi et seulement $16KB$ avec les générations antérieures. L'accès à ces données est très rapide lorsqu'il n'y a pas de conflit de banque entre les threads d'un même bloc.

Lorsque l'on observe l'étape de calcul des distances, on se rend compte que pour les $1\ 024$ profils distance de la base d'apprentissage (correspondant aux $1\ 024$ threads d'un bloc), on se sert à chaque fois du même profil distance de test. Dans un même bloc, il est donc intéressant de stocker le profil distance de test en mémoire « shared » car tous les threads d'un même bloc y ont accès de manière beaucoup plus rapide qu'en passant par la mémoire globale. Un profil de test contient $N + 2 \times D = 646$ échantillons, ce qui correspond donc à 646 floats soit environ $2{,}5KB$. La taille de la mémoire « shared » est donc largement suffisante pour contenir ce profil distance de test que ce soit avec la carte Tesla C2050 ou Tesla C1060. Cependant, les $16KB$ maximum pour la mémoire shared avec la carte C1060 sont limitant pour atteindre une occupation maximum. On se contente de 75 % d'occupation contre 100 % avec les versions précédentes.

La version 3, définie à partir de la version 2 en utilisant de la mémoire « shared » offre un gain de $1{,}52$ pour la carte Fermi et de $1{,}55$ pour la carte non-Fermi par rapport à la version 2. Cela permet d'atteindre $15{,}1$ % des performances pic de la carte C2050 et $10{,}3$ % de celles de la carte C1060. Le fait que le nombre de chargements mémoire théoriques a diminué avec cette version (les données de la base de test ne sont plus stockées en mémoire globale, on diminue donc par deux le nombre de chargements nécessaires en mémoire globale) permet d'expliquer cette amélioration. L'utilisation de la mémoire « shared » bénéficie donc dans les mêmes proportions à la carte C2050 et C1060.

6.5.3. Parallélisme d'instruction (ILP) : version 4 (V4)

La version 4 reprend les améliorations de la version 3 tout en ajoutant du parallèlisme d'instructions (ILP). Un déroulage de boucle de 8 est appliqué, c'est-à-dire qu'un thread, au lieu de traiter 1 seule distance entre profil distance de test et profil distance d'apprentissage en traite à présent 8. Cela permet de « cacher » encore mieux la latence mémoire et arithmétique. Cette modification permet d'obtenir
$20{,}1$ % des performances pic de la carte Fermi et $10{,}7$ % de celles de la carte non-Fermi. On remarque que la gain apporté par le parallélisme d'instructions est plus

important pour la carte Fermi que non Fermi. Cela s'explique encore en grande partie par la présence des caches L1 et L2 sur la carte Fermi qui n'est pas présent sur la carte non-Fermi. En observant le tableau 11, on se rend compte que très peu de données sont chargées directement en mémoire SDRAM avec la carte Fermi et la majorité des données à charger se trouve déjà dans les caches L1 ou L2. Le gain est faible sur la carte non-Fermi car, certes, la latence est « cachée » plus efficacement en introduisant de l'ILP mais celle-ci reste importante car les données sont chargées depuis la mémoire SDRAM sans cache.

6.5.4. Réutilisation de données : version 5 (V5)

En observant le pseudo-code utilisé jusqu'à présent (cf. figure 11), on se rend compte que l'on peut encore réduire le nombre de chargements théoriques en mémoire simplement en inversant l'ordre des boucles sur le recalage et les échantillons. En effet, jusqu'à présent pour chaque décalage, on recharge à chaque fois les échantillons du profil d'apprentissage alors que ceux-ci une fois chargés pourraient être réutilisés pour chaque décalage. Le pseudo-code du nouveau kernel CUDA est présenté sur la figure 12. Il correspond au kernel utilisé pour la version 5 du tableau 10.

for $i \leq N$ **do**
  for $dec \leq 2 \times D/10$ **do**
    $sum(j) = sum(j) + distance(X_T(k,i+dec),X_A(j,i))$
  end for
end for

Figure 12. Pseudo-Code du « kernel » modifié pour un profil distance de test k et un profil distance d'apprentissage j

En plus d'inverser l'ordre des boucles, on découpe la boucle totale sur les décalages en 10 boucles plus petites. Cela correspond à exécuter 10 fois le même kernel avec un nombre de décalage 10 fois moins grand pour chaque kernel. Cela permet de « cacher » de manière plus importante la latence mémoire . Cela entraîne cependant une modification de la taille des ressources allouées par thread. La taille allouée pour la mémoire « shared » sera notamment plus grande car cette modification oblige à conserver en mémoire « shared » les distances calculées pour chaque décalage de manière à pouvoir calculer la distance minimale lors de la dernière exécution du kernel. Du fait d'un nombre de chargements mémoire nécessaires beaucoup moins important, l'ILP n'est plus nécessaire avec la version 5.

La version 5 permet d'augmenter considérablement les performances sur les deux cartes C2050 ou C1060. On atteint 40,8 % des performances pic de la carte C2050 et 24 % de celles de la carte C1060 avec cette dernière version. A noter que l'occupation entre les versions 4 et 5 sur la carte C1060 a diminué. Cela est dû au fait que les ressources nécessaires en mémoire « shared » sont plus importantes avec la version 5 qu'avec la version 4. Avec la carte Fermi, cela est transparent car les $48KB$ sont largement suffisants pour conserver l'occupation de 50 %. En revanche, sur la carte non-Fermi, les $16KB$ limitent l'occupation.

6.5.5. Synthèse des performances obtenues

Le tableau 10 résume les performances obtenues en termes de temps de calcul pour les différentes versions vues jusqu'à présent. Le tableau 11 rassemble quant à lui les caractéristiques des différentes versions implémentées sur GPU. Il a été obtenu en utilisant les outils de profiling disponibles sur les cartes graphiques de « compute capabilities » 2.x ou supérieure, ce qui est le cas de la carte C2050 (Nsight + nvprof).

Tableau 10. Etape de calcul des distance, recalage et seuillage de l'algorithme des KPPV en « single precision » sur GPU Fermi et non Fermi en CUDA C++

	C2050 (Fermi)				C1060 (non Fermi)			
	Tps	% (Fp)	Occ	Gain	Tps	% (Fp)	Occ	Gain
V1	6 760	0,36	83 %		4 812	0,84	100 %	
V2	245	9,9	83 %	×28	605	6,67	100 %	×8
V3	161	15,1	83 %	×1,52	390	10,3	75 %	×1,55
V4	121	20,1	50 %	×1,33	378	10,7	50 %	×1,03
V5	59,7	40,8	50 %	×2,03	168	24	25 %	×2,25

Tps=Temps en millisecondes, % (Fp)=Pourcentage du nombre de Flops pic.
Le gain calculé pour la version i est calculé relativement à la version $i - 1$.

Tableau 11. Caractéristiques des différentes versions sur la Tesla C2050

	Al	SM	TLP/ILP	RD	NLT	NLR	LS	% L1	% L2
V1	N	N	TLP	N	62Go	940Go	763Go	0,74	44
V2	O	N	TLP	N	62Go	26,7Go	8,72Go	1,84	76,9
V3	O	O	TLP	N	31Go	22,2Go	4Go	5,43	83,7
V4	O	O	TLP/ILP	N	31Go	11,1Go	0,17Go	25,5	73,2
V5	O	O	TLP	0	3Go	2,8	1,10Go	0,19	72

O=Oui, N=Non, Al=Alignement, SM=Shared Memory, P=Parallélisme, RD=Réutilisation des données, NLT=Nombre de load théorique, NLR=Nombre de load réel, LS=Lecture en SDRAM.

6.6. Implémentation OpenCL

Une implémentation OpenCL de la version 5 a été réalisée de manière à comparer ses performances obtenues avec l'implémentation CUDA. Les résultats sont présentés dans le tableau 12.

D'une manière générale, les performances obtenues avec l'implémentation OpenCL par rapport à CUDA sont respectivement 30 % et 15 % moins bonnes sur la C2050 et la C1060 alors qu'il s'agit exactement de la même version que celle implémentée en CUDA. Une première raison à cette différence peut se trouver dans la qualité du compilateur. En étudiant le code assembleur généré par le compilateur OpenCL et CUDA, on se rend compte que certaines optimisations faites par le compilateur CUDA ne sont pas faites par le compilateur OpenCL. L'autre explication peut se trouver dans la qualité des drivers OpenCL et CUDA. Comme l'avait déjà mis en évidence Sawall dans son article de 2011 (Sawall *et al.*, 2011), les drivers OpenCL pour les matériels NVIDIA sont beaucoup moins optimisés que les drivers CUDA.

Tableau 12. Etape de calcul des distance, recalage et seuillage de l'algorithme des KPPV en « single precision » sur GPU Fermi et non Fermi en OpenCL

	C2050 (Fermi)		C1060 (non Fermi)	
	Temps(ms)	% Flops pic	Temps(ms)	% Flops pic
Version 5	85	29 %	200	20

6.7. Implémentation Matlab mex-CUDA et Matlab mex-OpenCL

Lors de la mise au point d'un algorithme, il est intéressant de pouvoir rester dans l'environnement Matlab pour pouvoir tester ces performances tout en profitant de l'accélération apportée par une implémentation sur GPU. Les mex-files permettent d'exécuter du code CUDA ou OpenCL tout en restant dans l'environnement Matlab. Le code de calcul exécuté sous GPU via les mex-files est celui correspondant à la version 5 de l'implémentation sur GPU.

Tableau 13. Temps de calcul (en ms) de l'étape de calcul des distances, recalage et seuillage de l'algorithme des KPPV exécuté en « single precision » sous Matlab, Matlab CUDA et Matlab OpenCL

	Temps(ms)	Gain vs Matlab
Matlab	1327	
Matlab CUDA	60	22
Matlab OpenCL	86	15

On retrouve (à quelques millisecondes près) les valeurs obtenues avec les versions CUDA C++ (version 5 du tableau 10) et OpenCL (version 5 du tableau 13). Il est important de noter que pour obtenir de tels résultats, il est indispensable de séparer les étapes d'initialisation et de remise à zero du device de l'étape de calcul à proprement

parler (exécution du kernel). Ces étapes prennent un temps non négligeable et il est important de ne pas exécuter ces étapes à chaque exécution du mex-file. La procédure est de définir trois mex-files différents (un pour l'initialisation, un pour la remise à zero et un pour le kernel). Les mex-files d'initialisation et de remise à zero ne sont exécutés qu'une seule fois dans le programme Matlab, ensuite le mex-file correspondant au kernel peut être exécuté autant de fois que l'on souhaite.

7. Conclusion

Cet article détaille l'application sur GPU d'un algorithme de type KPPV pour la reconnaissance de cibles. Un taux d'erreur faible et un taux de succès maximisé constituent la première contrainte forte de ce type d'application. Avec la base de données utilisées, un algorithme de type « force brute » où l'information n'est pas compressée permet d'obtenir un taux de succès élevé (98 %) pour un taux d'erreur faible (1 %). La caractérisation statistique de la base de données nous a permis d'obtenir en amont de la classification une approximation de ces résultats. En revanche, avec une base de données réelles donc plus bruitées, l'algorithme des KPPV sous cette forme ne permet pas de garantir un taux d'erreur maîtrisé en dessous d'une valeur fixée même si cela se fait au détriment du taux de succès. Il convient donc par la suite de définir de nouveaux algorithmes permettant de s'assurer un taux d'erreur suffisamment faible tout en maximisant le taux de succès. La deuxième contrainte forte pour ce type d'application est la contrainte de temps-réel (nous nous sommes fixés 1 ms par profil distance). Dans cet article, l'implémentation de l'algorithme des KPPV sur GPU a été étudiée de manière à respecter cette contrainte. Une première étude a permis de mettre en évidence l'importance de l'alignement des données en mémoire. Ensuite, à partir de cette implémentation que l'on peut qualifier de « basique » sur GPU, plusieurs améliorations ont été proposées pour augmenter les performances de calcul. Les performances sur un GPU de la génération Fermi et sur un GPU de la génération précédente ont été comparées. Les meilleures performances ont été obtenues sur la carte C2050 de la génération Fermi et permettent de traiter 150 profils distance de test en 59,7 ms soit 0,40 ms par profil distance, ce qui est inférieure à la limite que nous nous étions fixée. En dehors du respect de la contrainte temps-réel, cette étude a permis de mettre en évidence l'intérêt des cartes de la génération Fermi par rapport aux cartes de la génération antérieure. En mettant à part les performances pic plus importantes sur la carte C2050, on se rend compte que les innovations de la génération Fermi (cache L1 et L2, double ordonnanceur de warp) permettent d'exploiter plus facilement et plus rapidemment les puissances de calcul des GPU. Ces innovations permettent d'atteindre des performances de calcul très convenables avec beaucoup moins d'effort que pour la génération précédente (15,1 % des performances pic dès la version 3 pour la carte C2050 alors qu'il faut attendre la version 5 pour atteindre 24 % de la performance pic sur la carte C1060). Un gain de 22 a été obtenu en « single » précision entre une version Matlab vectorisée de l'algorithme et une version Matlab CUDA où le calcul des distances, le recalage et le seuillage sont effectués sur GPU. Ce gain est tout même à relativiser au vu du temps passé beaucoup plus important à optimiser la version GPU.

Une implémentation OpenCL a également été réalisée de manière à mettre en lumière les limites de ce standard sur les cartes conçues par NVIDIA.

Pour résumer, voici les principaux enseignements qui ressortent de cet article :

– Les meilleurs résultats de classification sont obtenus lorsqu'on applique l'algorithme sur les profils distance seuillés.

– Les performances varient peu en fonction de K et en fonction du type de distance.

– La caractérisation statistique des signaux permet d'obtenir en amont de la classification une approximation des performances de classification.

– Le coût de calcul a été significativement réduit en utilisant les GPU et des implémentations CUDA C++ ou OpenCL permettent de respecter la contrainte temps-réel.

8. Perspectives

Le cas étudié dans l'article se limite à trois classes de chasseurs ce qui dans un contexte opérationel n'est pas suffisant car le nombre de chasseurs différents est bien plus important. Dans la réalité, la base d'apprentissage sera beaucoup plus étendue et le nombre de distances à calculer sera beaucoup plus important. L'ensemble de ces distances ne pourra donc pas forcément être calculé en parallèle ou alors en augmentant le nombre de cartes GPU, ce qui dans un contexte embarqué par exemple n'est pas envisageable. De plus, l'algorithme présenté dans cet article utilise des traitements relativement simples (calcul de distance) pour faire de la reconnaissance. Les profils distance provenant d'acquisitions réelles nécessitent le plus souvent des traitements plus complexes pour pouvoir avoir des taux de reconnaissance acceptables tout en ayant une maîtrise du taux d'erreur. La charge de calcul sera donc plus importante que dans le cas de l'exécution d'un algorithme des KPPV. Au vu de ces constatations, il paraît intéressant d'étudier des espaces de représentation des profils distance qui permettent à la fois de réduire les charges de calcul et de rendre plus facile la discrimination des classes entre elles (sans pertes d'information discriminante).

8.1. Autres espaces de représentation

Dans cette section, deux exemples de transformations possibles permettant de représenter dans d'autres espaces les signatures radars reçues et de réduire leurs dimensions sont présentées : (i) la transformée de Fourier et (ii) la transformée temps-fréquence.

La transformée de Fourier est une transformation linéaire (si l'amplitude et la phase du signal sont conservées). Cette transformation peut nous permettre de réduire la taille des signaux à comparer. En effet, il se peut qu'après transformation de Fourier, l'information se concentre dans quelques coefficients de Fourier. En se basant sur des résultats expérimentaux, un nombre de coefficients à conserver pour chaque signal peut être fixé. Ensuite, l'algorithme des KPPV peut être appliqué de la même

façon que l'on a appliqué l'algorithme des KPPV sur les profils distance. La seule différence est que les signaux sont complexes, il faudra donc adapter la distance utilisée en conséquence.

Contrairement à la transformation de Fourier, la transformation en ondelettes permet d'analyser le signal à la fois en temps et en fréquence. Le principe est de représenter un profil distance par la somme pondérée d'une ondelette translatée ou dilatée. (Liu *et al.*, 2005) détaille une application des ondelettes aux profils distance. (Brousseau, 2010) a également utilisé des structures hiérarchiques d'ondelettes de type « arbre » pour représenter une base de données de profils distance synthétiques dans le contexte NCTR. On peut s'inspirer de ces études pour notre application. L'article de Liu propose de décomposer un profil distance en plusieurs sous-bandes, d'appliquer un classifieur sur chacune de ces sous-bandes et de fusionner les résultats de l'ensemble des classifieurs pour prendre une décision. Dans notre cas, le classifieur est l'algorithme des KPPV. Ensuite reste à définir l'opérateur de fusion. Il existe de nombreuses procédures de fusion de décision dans la littérature (Kittler *et al.*, 1998 ; Chen, Tang, 2004 ; Martin, Osswald, 2007). Dans notre cas, le principe de décision de l'algorithme des KPPV à savoir affecter au profil distance sous test, la décision revenant majoritairement parmi les différentes sous-bandes, pourrait être utilisé. L'inconvénient de cette décomposition en sous-bandes est qu'au lieu d'être exécuté une fois, l'algorithme des KPPV est exécuté N fois, N étant le nombre de sous-bandes retenues. En revanche, pour chaque sous-bande, le signal est sous-échantillonné et donc les signaux traités sont de plus petites tailles que les signaux originaux.

8.2. Autres méthodes de classification

Dans cet article, nous nous sommes volontairement limités aux méthodes de classification de type KPPV. Cependant, on pourrait imaginer l'utilisation d'autres types de classifieurs. Les classifieurs de type probabiliste permettent de travailler comme on l'a fait dans cet article dans différents espaces de représentation. Un des avantages des méthodes probabilistes est qu'en plus de prendre une décision concernant la classe du profil distance sous test, elles fournissent une mesure de confiance concernant la décision prise. En revanche, la modélisation des différentes classes est parfois complexe et l'estimation des paramètres du modèle doit être effectuée à chaque nouveau profil enrichissant la base d'apprentissage. Des algorithmes basés sur la logique floue peuvent également être utilisés. En effet, la logique floue fournit des mécanismes bien adaptés à la reconnaissance non coopérative de cibles où la prise en compte des incertitudes sur les mesures est très importante pour pouvoir maîtriser le taux d'erreur.

Bibliographie

Balasubramanian M., Schwartz E. L. (2002). The Isomap Algorithm and Topological Stability. *Science*, vol. 295, n° 5552, p. 7.

Brousseau C. (2010). Development of a tree structured hierarchical wavelet representation of synthetic database to NCTR. In *Proc. IEEE Radar Conference*, p. 368–373.

Chen L., Tang H. (2004). Improved computation of beliefs based on confusion matrix for combining multiple classifiers. *Electronics Letters*, vol. 40, n° 4, p. 238 - 239.

Collange S. (2010). *Analyse de l'architecture GPU Tesla*. Rapport technique. DALI, ELIAUS, Université de Perpignan.

Cooke S. J., Vlasov A. N., Levush B., Chernyavskiy I. A., Antonsen T. M. (2011). GPU-accelerated 3D time-domain simulation of vacuum electron devices. In *Proc. IEEE International Vacuum Electronics Conference*, p. 305–306.

Duda R. O., Hart P. E., Stork D. G. (2001). *Pattern Classification* (2e éd.). Wiley-Interscience.

Garcia V., Debeuve E., Nielsen F., Barlaud M. (2010). K-nearest neighbor search : Fast GPU-based implementations and application to high-dimensional feature matching. In *Proc. ICIP Conference*, p. 3757-3760.

Jain A. K., Duin R. P. W., Mao J. (2000). Statistical pattern recognition: a review. *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, n° 1, p. 4–37.

Kantardzic M. (2011). *Data Mining (Concepts, Models, Methods, and Algorithms)*. Wiley-Interscience.

Kirk D., Hwu W. (2010). *Programming Massively Parallel Processors*. NVIDIA, Elsevier Inc.

Kittler J., Society I. C., Hatef M., Duin R. P. W., Matas J. (1998). On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, p. 226–239.

Kotsiantis S. B. (2007). Supervised Machine Learning: A Review of Classification Techniques. *Informatica*, vol. 31, p. 249–268.

Lin J., Lin J. (2010). Accelerating BP Neural Network-Based Image Compression by CPU and GPU Cooperation. In *Proc. ICMT Conference*, p. 1–4.

Liu H., Yang Z., He K., Bao Z. (2005). Radar high range resolution profiles recognition based on wavelet packet and subband fusion. In *Proc. ICASSP conference*, vol. 5, p. 445-448.

Ma S., Ji C. (1999). Performance and efficiency: recent advances in supervised learning. *Proc. of the IEEE*, vol. 87, n° 9, p. 1519–1535.

Martin A., Osswald C. (2007). Une nouvelle règle de combinaison répartissant le conflit - Applications en imagerie Sonar et classification de cibles Radar. *Traitement du signal*.

Moruzzis M., Colin N. (1998). Radar target recognition by Fuzzy Logic. *IEEE Aerospace and Electronic Systems Magazine*, vol. 13, n° 7, p. 13–20.

Nimier V., Bastiere A., Colin N., Moruzzis M. (2000). MILORD, an application of multifeature fusion for radar NCTR. In *Proc. 3rd Information Fusion Conference.*

NVIDIA. (2007). *White Paper: Accelerating MATLAB with CUDA Using MEX Files.*

NVIDIA Programming Guide. (2011).

Owens J., Luebke D., Govindaraju N., Harris M., Kruger J., Lefohn A. (2007). A Survey of General-Purpose Computation on Graphics Hardware. *Computer Graphics Forum*, vol. 26, p. 80–113.

Rihaczek A., Herschowitz S. (2000). *Theory and Practice of Radar Target Identification*. Artech House.

Sawall S., Ritschl L., Knaup M., Kachelriess M. (2011). Performance comparison of OpenCL and CUDA by benchmarking an optimized perspective backprojection. In *3rd Workshop on High Performance Image Reconstruction*, p. 15–8.

Skolnik M. (1962). *Introduction to radar systems*. McGraw-Hill.

Tait P. (2005). *Introduction to Radar Target Recognition*. The IEE.

Tenenbaum J. B., Silva V., Langford J. C. (2000, 22). A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, vol. 290, n° 5500, p. 2319–2323.

Volkov V. (2010). Better performance at lower occupancy. In *GPU Technology Conference.*

Wehner D. R., Barnes B. (1994). *High-Resolution Radar*. Artech House Publishers.

Xu F., Mueller K. (2007). Real-time 3D computed tomographic reconstruction using commodity graphics hardware. *Physics in Medicine and Biology*, vol. 52, n° 12, p. 3405-3419.

Thomas Boulay est doctorant CIFRE au sein du laboratoire des Signaux et Systèmes (LSS) et de Thales Air Systems. Ses travaux portent sur l'étude d'algorithmes de reconnaissance de cibles non coopératives à partir de données radar. Il s'intéresse également à la parallélisation sur GPU de ces algorithmes.

Nicolas Gac est maître de conférence de l'université Paris Sud. Au laboratoire des Signaux et Systèmes (L2S), sa recherche porte sur le calcul parallèle pour les problèmes inverses. Les domaines applicatifs de ses travaux sont la reconstruction tomographique, la reconnaissance radar, la localisation de sources acoustiques et le traitement de données spectrales de Mars.

Ali Mohammad-Djafari est directeur de recherche et ses principaux domaines d'intérêt scientifique sont le développement de nouvelles méthodes probabilistes basées sur la Théorie de l'information, le Maximum d'entropie et les approches Bayésiennes pour les problèmes inverses. Ses travaux récents portent sur l'analyse en composantes principales (ACP), en composantes indépendantes (ACI) et factorielles appliquées aux signaux multivariés et à la fusion de données en tomographie.

Julien Lagoutte entre à Thales Air Systems en 2007 comme ingénieur études amonts au sein du département « Advanced Studies » au sein de l'unité « Surface Radar » et travaille sur le sujet de la classification et de la reconnaissance non coopérative de cibles (technique HRD et fusion par logique floue en particulier). Devenu chef de projet études amonts en 2011, il encadre dorénavant des projets autour des radars passifs.