Implémentation et optimisation du calcul sur GPUs FERMI et non FERMI de la distance euclidienne entre des profils distance obtenus à partir de données radar dans le contexte NCTR

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

Implémentation et optimisation du calcul sur GPUs FERMI et non FERMI de la distance euclidienne entre des profils distance obtenus à partir de données radar dans le contexte NCTR

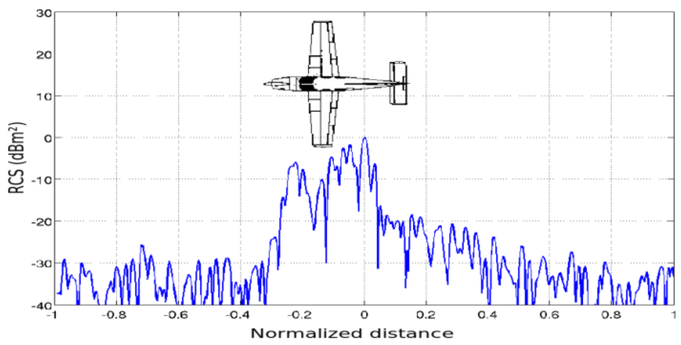
Thomas Boulay[1,2], Nicolas Gac[1], Ali Mohammad-Djafari[1], Julien Lagoutte[2]

[1] Laboratoire des Signaux et Systèmes (L2S, UMR 8506 CNRS - SUPELEC - Univ Paris Sud 11)
Supélec, Plateau de Moulon, F-91192 Gif-sur-Yvette, FRANCE.
[2] THALES AIR SYSTEMS, Chemin départemental 24 Roussigny, 91470 Limours en Hurepoix, FRANCE
Emails: thomas.boulay@lss.supelec.fr, nicolas.gac@lss.supelec.fr, djafari@lss.supelec.fr, julien.lagoutte@thalesgroup.com

Ecole d'été GPU 2011, Grenoble, France

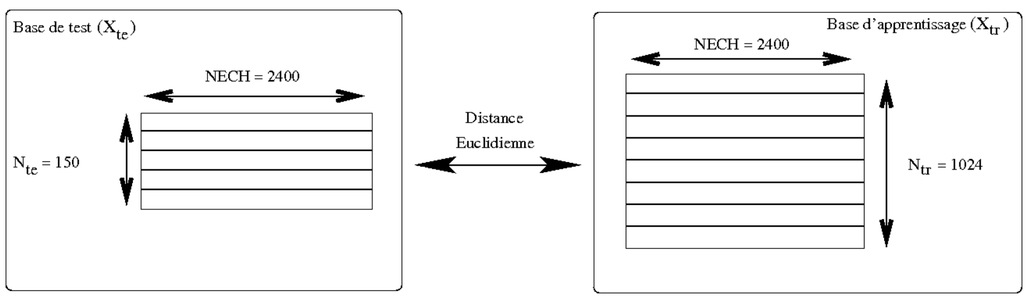
Contexte de la reconnaissance de cibles

Exemple de profil distance



Principe de l'algorithme de reconnaissance

Comparaison des profils distance à reconnaitre avec les profils distance de la base d'apprentissage.
Principe des KPPV: Le profil distance à reconnaitre appartient à la classe majoritaire parmi les K plus proches voisins.



Métrique utilisée:

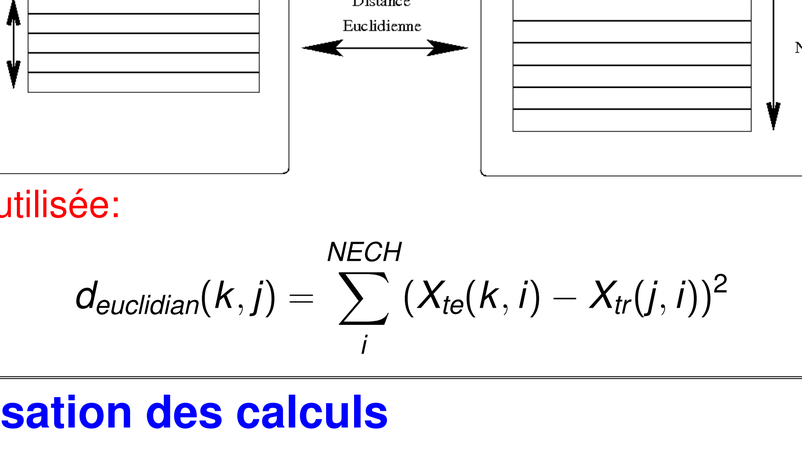
$$d_{euclidian}(k,j) = \sum_{i}^{NECH} (X_{te}(k,i) - X_{tr}(j,i))^2$$

Parallélisation des calculs

GPU utilisés:

- NVIDIA TESLA C2060 (Architecture Fermi)
- NVIDIA TESLA C1060
- NVIDIA TESLA C870

Architecture: 2 grilles de 150 blocs de 512 threads



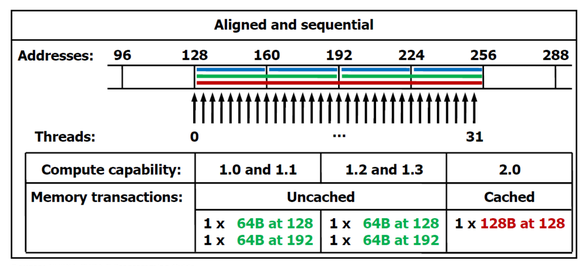
Pseudo-Code:

Pour chaque $Th_k(Thread_k)$

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

Optimisation des accès mémoire

Bon stockage mémoire:



Compute capability:	1.0 and 1.1	1.2 and 1.3	2.0
Memory transactions:	Uncached		Cached
	1 x 64B at 128 1 x 64B at 192	1 x 64B at 128 1 x 64B at 192	1 x 128B at 128

Source: NVIDIA CUDA C Programming Guide

Mémoire "shared": Utilisation de la mémoire "shared" pour X_{te}.

Textures: Utilisation de la texture pour X_{tr}.

Résultats:

	C2060(Fermi)		C1060		C870	
Ms, $X_{te} \rightarrow MG$, $X_{tr} \rightarrow MG$	430 ms	**Gain**	436 ms	**Gain**	956 ms	**Gain**
Bs, $X_{te} \rightarrow MG$, $X_{tr} \rightarrow MG$	11 ms	$\times$39	27 ms	$\times$16	273 ms	$\times$3.6
Bs, $X_{te} \rightarrow MS$, $X_{tr} \rightarrow MG$	**10ms**	$\times$1.1	18 ms	$\times$1.5	23 ms	$\times$11
Bs, $X_{te} \rightarrow MS$, $X_{tr} \rightarrow T$	18 ms	$\times$0.6	17 ms	$\times$1.1	30 ms	$\times$0.8

avec **Ms**:Mauvais stockage, **Bs**:Bon stockage, **MS**:Mémoire Shared, **MG**: Mémoire Globale, **T**:Texture

Cache Fermi: **MS** et **T** n'améliorent pas les performances avec Fermi alors que les performances sont meilleures avec les GPU sans architecture Fermi $\Rightarrow$ Plus besoin d'utiliser **MS** et **T** pour notre application.

Utilisation de CUBLAS

Distance euclidienne et multiplication de matrice: On peut réécrire la distance euclidienne de la manière suivante [1]:

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

$||\vec{x}||^2$ et $||\vec{y}||^2$: Il s'agit des normes euclidiennes de chaque profil distance de test et d'apprentissage (150 normes pour la base de test et 1024 normes pour la base d'apprentissage). Pour optimiser le calcul, on exécutera deux kernels concurrents (Fermi) qui s'exécuteront en même temps.

$\vec{x}.\vec{y}$: Calcul des produits scalaires entre tous les profils de test et tous les profils d'apprentissage. Il s'agit de multiplier la matrice de test (A) par la transposée de la matrice d'apprentissage (B) $\Rightarrow A.B'$.
$\Rightarrow$ Utilisation de la librairie CUBLAS.

$d_{euclidian} = ||\vec{x}||^2 + ||\vec{y}||^2 - 2\vec{x}.\vec{y}$: Un kernel calculera la matrice de la somme de la norme de chaque profils de test et d'apprentissage et de leur produit scalaire.

Résultats:

	C2060(Fermi)		C1060		C870	
Version CUBLAS	**5ms**	$\times$2	7*ms*	$\times$2.5	8*ms*	$\times$2.9

Perspectives

- Autres types de métriques (Distance L1, Distance de Kullback, Corrélation, ...)
- Algorithme de classification sur GPU.

[1] K. van de Sande and al. Empowering visual categorization with the GPU. 13(1):60–70, 2011.