Rapport de code : Implémentation du papier "Windowed Radon Transform & Tensor Rank-1 Decomposition"

Emilien Sanchez, Oscar Gledel

January 13, 2026

Ce document présente l'analyse technique du code source fourni par les auteurs, qui constitue une implémentation de l'algorithme décrit dans leur papier : "Windowed Radon Transform and Tensor Rank-1 Decomposition for Adaptive Beamforming". L'objectif principal de cet algorithme est d'améliorer la qualité des images échographiques. Pour ce faire, il filtre les données brutes dans le domaine de Radon avant la formation de l'image finale.

1. Architecture du Projet

Un seul fichier "compute_corrected_image.py" était fourni, nous avons procédé à une restructure du code afin de le rendre plus lisible. Nous l'avons structuré de manière modulaire, séparant l'acquisition des données, le calcul CPU et l'accélération GPU.

A. Structure des fichiers.

- **src/read_data.py** : Permet de visualiser les données à partir des matrices complexes de données brutes.
- **src/compute_corrected_image.py** : Contient toutes les implémentations du papier fourni par les auteurs.
- **src/run_script.py** : Point d'entrée pour lancer le code en local. Il gère le support GPU ou CPU en fonction de la disponibilité de CUDA.
- **src/run_notebook.ipynb** : Point d'entrée pour lancer le code en profitant des gpu de google colab. Il gère le support GPU ou CPU en fonction de la disponibilité de CUDA.

2. Analyse Algorithmique et Théorique

Le code transpose la méthode du papier selon une approche en trois étapes majeures:

A. Transformée de Radon Fenêtrée (WRT). L'algorithme ne traite pas l'image globalement mais procède par fenêtres glissantes.

- Cela se traduit par des boucles découpant le signal RF en sous-blocs.
- Chaque bloc subit une Transformée de Radon. Cela transforme les fronts d'ondes (lignes dans l'espace RF canal-temps) en points dans l'espace de Radon (paramètres $\tau - \theta$).

B. Décomposition Tensorielle de Rang 1 (SVD). Dans le domaine de Radon local, le signal d'intérêt (l'écho principal) est fortement corrélé et correspond à la composante principale, tandis que le bruit et le clutter sont dispersés. Le code applique une Décomposition en Valeurs Singulières sur la matrice transformée. On ne conserve que la première valeur singulière (Rang 1) et on rejette les autres (considérées comme du bruit/clutter).

$$X_{filtre} = \sigma_1 u_1 v_1^T$$

C. Reconstruction (Inverse Radon & Beamforming).

- **Inverse Radon** : Le signal filtré est ramené dans le domaine temporel par une transformée de Radon inverse.
- **Delay-and-Sum (DAS)** : Enfin, le beamforming classique est appliqué sur ces signaux "nettoyés" pour former l'image finale.

D. Vue d'ensemble du Pipeline. Le script qui était fourni transforme les données brutes (RF data) en une image finale focalisée en insérant une étape de filtrage statistique dans le domaine de Radon local, contrairement au beamforming classique. Les étapes du pipeline sont:

Entrée : Données RF brutes (Canaux x Temps).

Pré-focalisation : Alignement temporel des signaux pour chaque point de l'image.

Découpage : Sélection d'une sous-ouverture (sous-ensemble de canaux) autour de l'élément central.

Transformation : Passage dans le domaine de Radon (Transformée $\tau - \theta$).

Filtrage (Rank-1) : Extraction de la composante principale du front d'onde.

Reconstruction & Sommation : Retour au domaine spatial et sommation cohérente.

3. Analyse détaillée des implémentations

A. Fichier : src/compute_corrected_image.py. Ce fichier contient l'ensemble des fonctions utilitaires pour le traitement du signal (chargement, pré-traitement, formation de faisceau, correction d'aberration) pour CPU et GPU.

A.1. Fonctions de Chargement et Pré-traitement.

- **read_linear_transducer_data_standard :** Lit les données brutes (data.npy) et métadonnées (metadata.h5) d'un dossier donné. Elle retourne les données brutes et un dictionnaire contenant les paramètres de la sonde (angles, vitesse du son, fréquence, géométrie, etc.).

- **compute_hilbert_fir :** Calcule le signal analytique (complexe) des données d'entrée en utilisant une transformée de Hilbert implémentée via un filtre RIF (Réponse Impulsionnelle Finie). Cela permet d'obtenir l'enveloppe et la phase du signal.

- **apply_exponential_tgc :** Applique une compensation de gain temporel (Time-Gain Compensation) pour compenser l'atténuation liée à la profondeur.

A.2. Fonctions GPU (via numba.cuda). Ces fonctions génèrent et retournent des noyaux CUDA ou des fonctions wrapper optimisés pour le GPU.

- **get_beamformer_npw_linear_transducer_Tukey_phase_screen(...) :** Crée une fonction de beamforming de type Delay-and-Sum. Elle calcule les retards de propagation pour chaque pixel de l'image en fonction des positions des capteurs et des angles d'émission, applique une apodisation de Tukey, et somme les signaux.

- **get_select_patch_window _cuda_function(...) :** Génère une fonction CUDA pour extraire des petites sous-images ("patches") de l'image formée, centrées sur des positions spécifiques, et leur appliquer une fenêtre de pondération (ex: Tukey ou Hann).

- **get_patch_radon_transform_rx_cuda _function(...) :** Génère une fonction CUDA qui effectue une transformée de Radon sur les patches extraits. Elle projette les données du domaine spatial vers le domaine angulaire (sino-gramme) pour séparer les contributions.

- **get_decomposition_function_gpu(...) :** Génère une fonction CUDA effectuant une décomposition tensorielle itérative (basée sur une approximation de rang faible). Elle sépare le signal en une composante "objet" (f) et des termes de distorsion liés à l'émission (utx) et à la réception (urx).

- **get_patch_backprojection_mid_window_cuda _function(...) :** Génère une fonction CUDA pour effectuer l'opération inverse de la transformée de Radon (rétroprojection). Elle reconstruit les patches spatiaux à partir des composantes angulaires filtrées et séparées.

- **get_reconstruct_image_functions_gpu(...) :** Retourne un ensemble de fonctions CUDA pour reconstruire l'image finale à partir des patches traités. Ces fonctions gèrent :
  - L'assemblage des patches dans l'image finale (reconstruct_image).
  - Le calcul des décalages de phase/correcteurs (update_shift).
  - Le calcul de la carte de normalisation (get_normalization).
  - La normalisation finale de l'image (normalize_image).

A.3. Fonctions CPU (via numba.njit et prange). Ces fonctions sont les équivalents des fonctions GPU ci-dessus, mais optimisées pour une exécution parallèle sur CPU.

- **get_beamformer_npw_linear_transducer _Tukey_phase_screen_cpu :** Version CPU du Delay-and-Sum. Elle utilise prange pour paralléliser les boucles sur les pixels de l'image.

- **get_select_patch_window_cpu_function :** Version CPU de la fonction d'extraction et de fenêtrage des patches.

- **get_patch_radon_transform_rx_cpu_function :** Version CPU de la transformée de Radon sur les patches, incluant le filtrage pré-Radon.

- **get_decomposition_function_cpu :** Décomposition tensorielle CPU.

- **get_patch_backprojection_mid_window_cpu_function :** Version CPU de la décomposition tensorielle itérative pour estimer f, utx et urx.

- **get_reconstruct_image_functions_cpu :** Retourne les équivalents CPU pour la reconstruction de l'image finale (assemblage, correction de phase, normalisation).

A.4. Affichage.

- **bmode_simple :** Affiche l'image en mode B (échelle log dB) avec gestion de la plage dynamique.

B. Fichier : src/run_script.py. Ce script orchestre le chargement, la configuration, le pipeline complet et la sauvegarde.

- **load_preprocess_data :** Charge les données, trie les angles, sous-échantillonne, calcule le signal analytique (Hilbert) et applique le TGC.

- **gpu(...) :** Compile les fonctions CUDA, alloue la mémoire GPU (buffers) et retourne les pointeurs pour la boucle principale.

- **main(folder_in, folder_out, is_gpu) :** Fonction principale :
  - Définit les paramètres (grille, angles, apodisation, décomposition).
  - Appelle load_preprocess_data pour préparer les données.
  - Initialise les fonctions de traitement (soit via gpu(), soit en appelant les générateurs CPU).
  - Lance le Beamforming initial.
  - Exécute la boucle de correction par patches (Extraction → Radon → Décomposition → Rétroprojection).
  - Reconstruit l'image finale en assemblant les patches et en corrigeant les décalages de phase.
  - Sauvegarde les résultats et affiche l'image finale avec bmode_simple.

C. Fichier : src/run_notebook.ipynb. Ce notebook Jupyter sert de point d'entrée pour l'exécution du code dans un environnement Cloud (Google Colab), facilitant l'accès aux ressources GPU (comme les T4) sans configuration locale complexe.

- Initialisation de l'environnement : Montage du Google Drive pour la persistance des données.
- Commandes Bash (!git clone, !git pull) pour récupérer et mettre à jour le dépôt de code depuis GitHub.
- Vérification de l'environnement CUDA via !which nvcc pour s'assurer de la disponibilité du compilateur GPU.
- Acquisition des données : Téléchargement automatique des jeux de données de test (data.npy, metadata.h5) depuis un dépôt distant via curl, évitant un upload manuel volumineux.
- Exécution du Pipeline : Appel du script principal via la commande shell !python3 run_script.py. Cela lance le traitement complet (Beamforming → Radon → SVD → Reconstruction) défini dans la section 3.2.
- Visualisation des Résultats : Une cellule Python finale charge les fichiers de sortie générés (data.npy, x_coord.npy, z_coord.npy) et utilise la fonction aff_res importée de read_data pour afficher l'image reconstruite finale directement dans le notebook.

D. Fichier : src/read_data.py. Ce fichier est un module utilitaire dédié à la visualisation des résultats post-traitement. Il permet de convertir les données complexes en images échographiques lisibles (B-mode) et de les afficher. Dérivé de BmodeSimple provenant de compute_corrected_image

- Fonction principale : aff_res Cette fonction gère la chaîne de traitement pour l'affichage d'une image échographique à partir des données brutes reconstruites :
- Calcul de l'enveloppe : Elle calcule le module du signal complexe (np.abs) en ajoutant une valeur infinitésimale (epsilon) pour éviter les erreurs mathématiques lors du passage au logarithme.
- Compression Logarithmique : Elle convertit l'amplitude en décibels (dB) via la formule 20log10(x), standard en imagerie médicale pour gérer la grande dynamique des échos.
- Normalisation et Affichage : L'image est normalisée (le maximum est fixé à 0 dB) et affichée avec une plage dynamique ajustable (par défaut 60 dB) en utilisant matplotlib. Elle gère également l'inversion de l'axe Z pour respecter la convention échographique (profondeur vers le bas).