Documentation du Code Python: Implémentation d'un Algorithme de Beamforming pour l'Imagerie Ultrasonore

CADET Florent, DORFNER François

14 mars 2025

Résumé

Cette documentation présente l'analyse et l'implémentation d'un algorithme de beamforming pour l'imagerie ultrasonore en Python. Le code développé intègre des techniques de traitement du signal comme la transformée de Radon et exploite le calcul parallèle sur GPU pour optimiser les performances. Ce document détaille l'architecture du code, ses fonctionnalités et son utilisation.

Github page : `https://github.com/norphiil/Code-Projet-Papier-8`

Table des matières

1 Introduction

Cette analyse porte sur une implémentation Python d'un algorithme de beamforming pour l'imagerie ultrasonore. Le code combine des techniques avancées de traitement du signal avec une utilisation optimisée des ressources GPU, permettant d'obtenir des performances élevées dans le traitement des images échographiques.

1.1 Architecture Générale

L'architecture du code s'organise autour d'une chaîne de traitement qui commence au niveau du module d'acquisition, responsable de la lecture et du formatage des données brutes du transducteur. Un pipeline de prétraitement prend ensuite le relais pour préparer les signaux avant leur analyse. L'algorithme de beamforming, élément central du système, avec une optimisation de calcul GPU. La chaîne se termine par un module de visualisation offrant les outils nécessaires à l'analyse des résultats.

L'implémentation repose sur plusieurs techniques avancées qui contribuent à ses performances. L'utilisation de CUDA via la bibliothèque Numba permet une accélération significative des calculs sur GPU. Le traitement des données s'appuie sur une approche combinant transformée de Radon fenêtrée et décomposition tensorielle de rang 1, permettant une correction efficace des aberrations. La gestion de la mémoire est optimisée grâce à un traitement par lots.

1.2 Configuration du Système

Les paramètres de contrôle suivants définissent le comportement de l'algorithme :
— `angle_rx_max` : Angle maximal de réception (36°)
— `window_radius` : Rayon de la fenêtre d'analyse (2mm)
— `window_type` : Type de fenêtrage (Tukey)
— `reg_param` : Paramètre de régularisation (1)
— `low_rank_n_iter` : Nombre d'itérations de l'algorithme (20)

Ces paramètres par défaut offrent un équilibre entre qualité d'image et performance, tout en restant ajustables selon les besoins spécifiques de l'application.

2 Structure du Code Python

L'implémentation s'articule autour de modules fonctionnels distincts, chacun dédié à une étape spécifique du traitement des données échographiques.

2.1 Modules Fonctionnels

Le module d'acquisition, basé sur la fonction `read_linear_transducer_data_standard`, assure l'interface avec le matériel et la gestion des données brutes. Le prétraitement s'effectue via deux fonctions complémentaires : `compute_hilbert_fir` réalise la transformée de Hilbert, et `apply_exponential_tgc` applique la compensation de gain nécessaire à l'optimisation de la qualité d'image.

L'algorithme de beamforming est implémenté dans le module central `get_beamformer_npw_linear_transducer_Tukey_phase_screen`, qui intègre un fenêtrage de Tukey pour l'optimisation du traitement.

Le module de traitement adaptatif comprend quatre fonctions spécialisées :

— `get_select_patch_window_cuda_function` : Segmentation des données
— `get_patch_radon_transform_rx_cuda_function` : Calcul de la transformée de Radon
— `get_decomposition_function_gpu` : Optimisation tensorielle
— `get_patch_backprojection_mid_window_cuda_function` : Reconstruction des données

La reconstruction finale s'effectue via `get_reconstruct_image_functions_gpu`, qui assure l'assemblage cohérent des résultats.

2.2 Optimisation des Performances

L'implémentation exploite l'architecture CUDA via Numba pour maximiser les performances :

— Optimisation de la mémoire partagée GPU
— Parallélisation du traitement par patches
— Optimisation des transferts de données CPU-GPU

2.3 Configuration du Système

Les paramètres de configuration se répartissent en plusieurs catégories :

— **Paramètres d'Acquisition** :
  — `angle_downsample_factor` : Facteur de sous-échantillonnage angulaire
  — `attenuation_tgc` : Coefficient de compensation de gain
— **Paramètres de Traitement** :
  — `delta_lambda_fraction` : Résolution spatiale de la grille
  — `tukey_angle` : Paramètre de fenêtrage
— **Paramètres d'Analyse** :
  — `window_radius` : Dimension de la fenêtre d'analyse
  — `window_type` : Configuration du fenêtrage
  — `patch_stride` : Pas d'échantillonnage spatial
— **Paramètres d'Optimisation** :
  — `reg_param` : Coefficient de régularisation
  — `low_rank_n_iter` : Nombre d'itérations d'optimisation

3 Pipeline de Traitement

Le traitement des données s'effectue selon un flux séquentiel optimisé, structuré en trois phases distinctes.

3.1 Acquisition et Prétraitement

L'acquisition des données brutes constitue la première étape du processus. Un sous-échantillonnage angulaire est appliqué selon les paramètres définis pour optimiser la charge de traitement. La conversion en format analytique s'effectue via la transformée de Hilbert,

suivie d'une compensation de gain temporel qui normalise les amplitudes sur l'ensemble de la profondeur d'imagerie.

3.2 Configuration et Initialisation

Cette phase établit la configuration du système de traitement. La définition de la grille de reconstruction répond aux exigences de résolution spécifiées. Les paramètres angulaires sont calculés pour garantir la qualité d'image requise. La configuration des fenêtres d'analyse optimise l'équilibre entre résolution spatiale et rapport signal sur bruit.

3.3 Traitement GPU

Le traitement GPU constitue la phase centrale de l'algorithme. L'initialisation comprend l'allocation des ressources GPU et le transfert des paramètres système. Le traitement par lots s'effectue selon une séquence de quatre opérations :

1. Segmentation des données en patches
2. Application de la transformée de Radon
3. Optimisation par décomposition tensorielle
4. Reconstruction locale des données

Cette approche séquentielle optimise l'utilisation des ressources GPU tout en maintenant un niveau élevé de performance.

4 Utilisation du Code

4.1 Prérequis Système

L'exécution du code nécessite la configuration suivante :
- Environnement Python 3.x
- Processeur graphique NVIDIA compatible CUDA
- Bibliothèques Python requises :
  - Numpy pour le calcul numérique
  - Scipy pour les fonctions scientifiques
  - h5py pour la gestion des données HDF5
  - Numba pour la compilation CUDA
  - Matplotlib pour la visualisation

4.2 Format des Données

Le système requiert deux fichiers d'entrée structurés :
- **Fichier de Données** (`data.npy`) :
  - Structure : Tableau Numpy tridimensionnel (N_Tx × N_Rx × N_t)
  - Dimensions :
    - N_Tx : Nombre d'angles d'émission
    - N_Rx : Nombre d'éléments récepteurs
    - N_t : Nombre d'échantillons temporels
- **Fichier de Métadonnées** (`metadata.h5`) :
  - Configuration angulaire

— Paramètres temporels
— Spécifications du système
— Configuration de l'apodisation

4.3 Procédure d'Exécution

L'exécution du code s'effectue selon la séquence suivante :

1. Préparation de l'environnement de données
2. Configuration des paramètres système
3. Lancement du traitement

4.4 Fichiers de Sortie

Le traitement génère les fichiers suivants :
— `data.npy` : Image reconstruite
— `x_coord.npy`, `z_coord.npy` : Coordonnées spatiales
— `additional_parameters.h5` : Paramètres de reconstruction
— `info.txt` : Journal d'exécution détaillé