MU4IN901

PROJET DE MODEL

Assia MASTOR
28602563
Amel MOKDADI
21309654

• • •

MASTER CCA
2023-2024

SORBONNE UNIVERSITÉ

I- Présentation du code

Au cours de ce projet, nous avons implémenté un ensemble d'opérations arithmétiques pour les nombres complexes basés sur des nombres à virgule flottante double précision. Ce rapport présente notre implémentation des opérations pour les nombres complexes, l'algorithme FFT (Transformée de Fourier rapide), IFFT (Transformée de Fourier inverse) pour des vecteurs de taille 2^k, ainsi que l'algorithme naïf et basé sur FFT pour la multiplication de polynômes avec des coefficients entiers.

Nous avons implémenté les fonctionnalités décrites en utilisant le langage de programmation C. Pour commencer nous avons créé une structure de données Complex qui représente un nombre complexe avec sa partie réelle ainsi que sa partie imaginaire. Voici ensuite le descriptif de chacune de nos fonctions:

Les fonctions relatives aux opérations:

fft : Cette fonction calcule la Transformée de Fourier rapide d'un vecteur de nombres complexes. Elle utilise une approche récursive pour diviser le calcul en sous-problèmes, applique les calculs sur les parties réelles et imaginaires du vecteur, puis combine les résultats pour obtenir la FFT.

ifft : Cette fonction calcule la Transformée de Fourier inverse d'un vecteur de nombres complexes. Elle appelle la fonction calculate_fft en utilisant des angles inversés, puis normalise les valeurs du vecteur résultant en divisant par la taille de l'échantillon.

multiply_polynomials : Cette fonction multiplie deux polynômes représentés par des tableaux entiers. Elle prend en paramètre les coefficients des polynômes a et b ainsi que leurs degrés n et m, effectue la multiplication des polynômes en utilisant la méthode naïve, et retourne le résultat sous la forme d'un tableau représentant le polynôme résultant.

multiply_polynomials_fft : Cette fonction multiplie deux polynômes représentés par des tableaux entiers en utilisant l'algorithme FFT (Transformée de Fourier rapide). Elle calcule la taille nécessaire pour l'algorithme FFT, effectue l'allocation mémoire pour les tableaux de nombres complexes, applique la FFT aux deux polynômes, effectue la multiplication des polynômes transformés, puis applique l’IFFT pour obtenir le résultat final.

Les fonctions annexes qui permettent une bonne implémentation:

add : Cette fonction additionne deux nombres complexes. Elle prend en paramètre deux nombres complexes a et b, calcule la somme de ces deux nombres et retourne le résultat en tant que nombre complexe.

substract : Cette fonction soustrait deux nombres complexes. Elle prend en paramètre deux nombres complexes a et b, calcule la différence de ces deux nombres et retourne le résultat en tant que nombre complexe.

multiply : Cette fonction multiplie deux nombres complexes. Elle prend en paramètre deux nombres complexes a et b, calcule le produit de ces deux nombres et retourne le résultat en tant que nombre complexe.

resizeToPowerOfTwo : Cette fonction redimensionne un vecteur de nombres complexes à la taille suivante de puissance de deux. Elle calcule la nouvelle taille en trouvant la prochaine puissance de deux supérieure à la taille actuelle, alloue de la mémoire pour le nouveau vecteur, copie les éléments du vecteur d'origine dans le nouveau vecteur et remplit les éléments supplémentaires avec des zéros.

intArrayToComplex : Cette fonction convertit un tableau d'entiers en un tableau de nombres complexes en initialisant la partie réelle à partir des entiers du tableau et la partie imaginaire à zéro.

complex_vector_multiply: Cette fonction multiplie deux tableaux de nombres complexes élément par élément en utilisant les formules de multiplication des nombres complexes et retourne un tableau contenant le résultat de ces opérations.

reduce_vector: Cette fonction supprime les zéros non significatifs à la fin d'un vecteur de nombres complexes et retourne un nouveau vecteur réduit ainsi que met à jour la taille du vecteur d'entrée.

calculate_fft: Cette fonction redimensionne d'abord le vecteur d'entrée à une taille qui est une puissance de deux, puis applique la FFT à ce vecteur et met à jour la taille du vecteur avant de le retourner.

Les fonctions qui permettent la comparaison :

generate_random_polynomial : Cette fonction génère un polynôme aléatoire en produisant des coefficients aléatoires pour le polynôme. Elle prend en paramètre le degré n du polynôme, alloue de la mémoire pour stocker les coefficients, génère des coefficients aléatoires pour le polynôme et retourne un tableau représentant le polynôme.

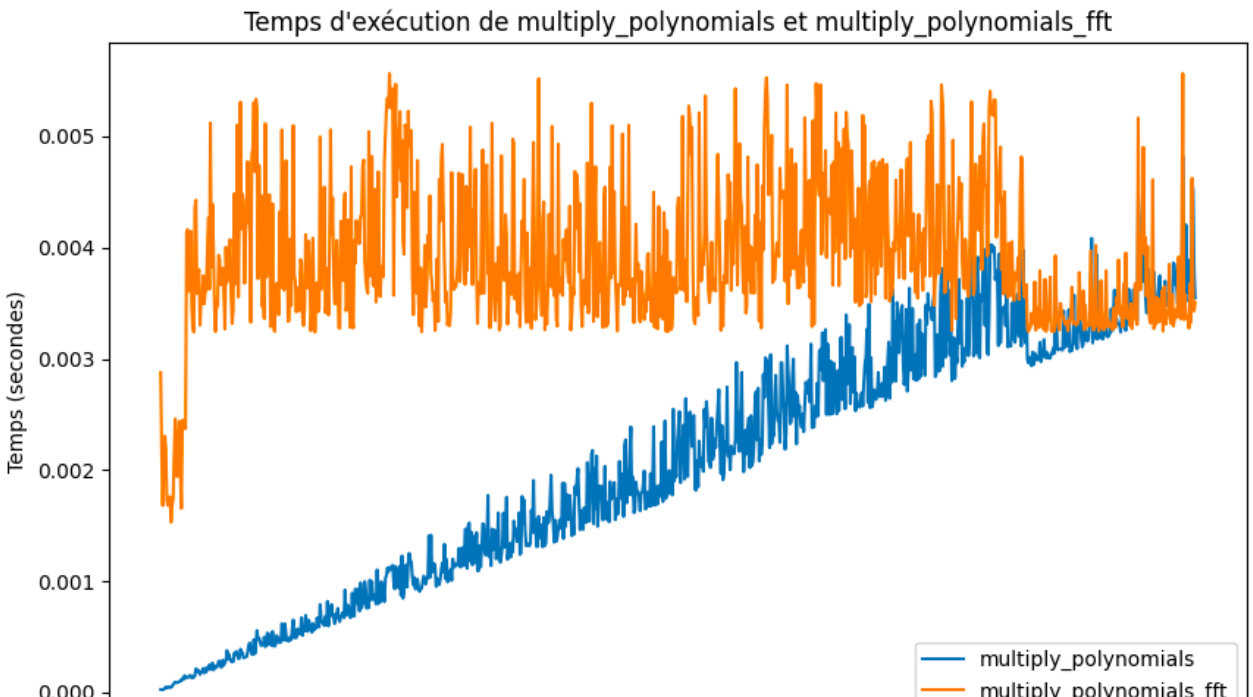
tempsexe : Cette fonction génère des polynômes aléatoires de degrés croissants, mesure le temps d'exécution des fonctions multiply_polynomials et multiply_polynomials_fft pour chaque paire de polynômes, et écrit les temps d'exécution dans un fichier times.txt.

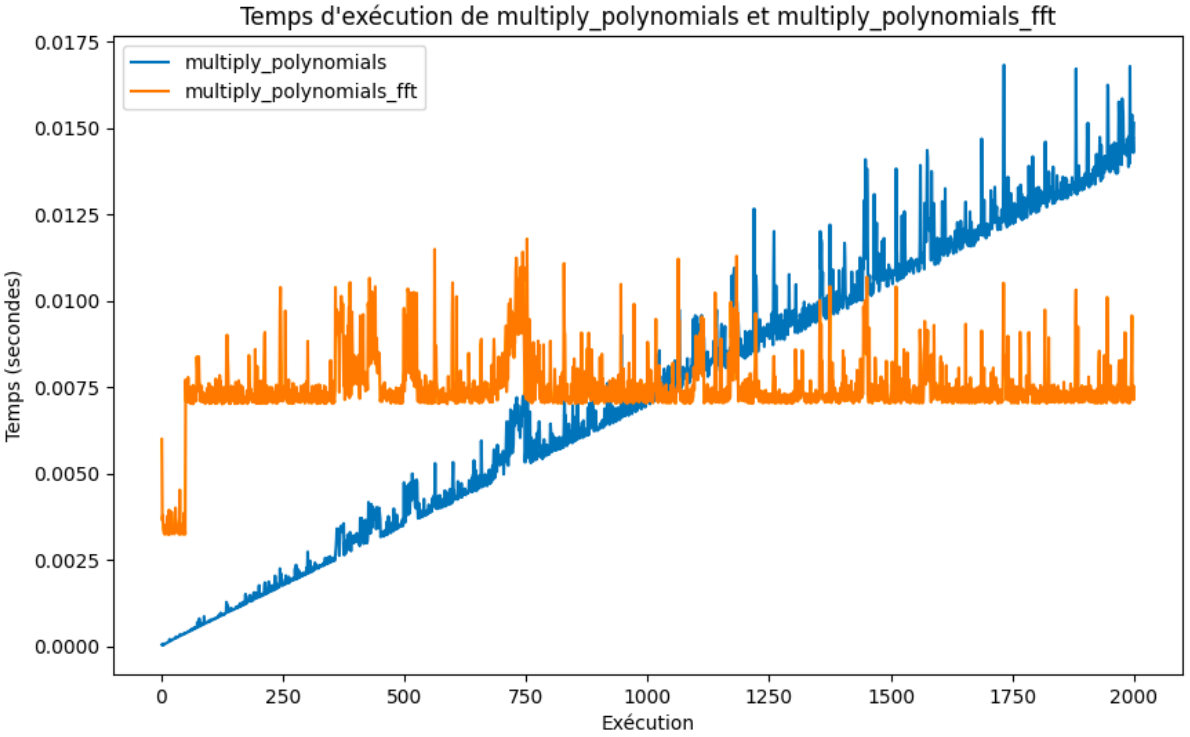
II- Analyse

Nous avons ensuite évalué les performances des deux algorithmes de multiplication de polynômes en mesurant leurs temps d'exécution respectifs pour la multiplication de polynômes de degrés variables.

Pour cela on fait tout d'abord appelle à une fonction qui génère des polynômes aléatoires pour différents degrés et on relève le temps d'exécution pour la multiplication naïve ainsi que pour la multiplication fft.

Ces données sont enregistrées dans un fichier.txt qui est ensuite lu par un fichier python appelé courbes.py grâce auquel on obtient deux courbes comparatives.





On constate que multiply_polynomials (l'algorithme naïf) est bien plus rapide pour des polynômes de petits degré mais le temps d'exécution augmente quasi proportionnellement par rapport au degré des polynômes multipliés. Quant à l'algorithme de multiplication avec fft, le temps d'exécution est assez constant. On peut également voir que plus on augmente le nombre d'éléments du polynôme plus il est préférable d'utiliser la méthode fft car elle devient plus rapide que la méthode naïve à partir d'un certain degré.

On peut en conclure que pour des petites tailles de polynômes, la méthode naïve est plus rapide sûrement en raison des coûts supplémentaires associés à la transformation FFT. Cependant, pour des tailles de polynômes plus importantes, l'utilisation de la FFT peut être beaucoup plus efficace en raison de sa complexité asymptotique inférieure.

III- Difficultés rencontrées

L’une des principales difficultés rencontrées fut la gestion de la taille des vecteurs lors de l'application de la FFT. Comprendre la FFT dans son application standard a été déjà un défi en soi, mais l'utiliser spécifiquement pour la multiplication de polynômes a considérablement complexifié la situation.

Nous avons consacré un temps considérable à essayer de comprendre comment adapter correctement la FFT pour manipuler des polynômes, en particulier en ce qui concerne le redimensionnement des vecteurs pour s'adapter aux exigences de la FFT.

Egalement, nous avons rencontré pas mal de soucis quant aux pointeurs et à leur utilisation, nous avons été amené à utiliser des copies de variables, qui ont finalement été enlevées mais qui nous ont fait perdre assez de temps.