Rapport Model

Auteur :

- ZHOU Runlin 28717281
- ZHANG Zhile 21201131
- XU Mengqian 21306077

Introduction:

Ce projet implique la mise en œuvre d'opérations arithmétiques pour les nombres complexes en utilisant des valeurs en virgule flottante double précision. De plus, il comprend le développement de fonctions de transformation de Fourier rapide (FFT) et de FFT inverse pour des vecteurs de taille 2^k, traitant les ajustements de taille de vecteur pour les cas non puissances de 2. Le projet couvre également l'implémentation et la comparaison d'algorithmes naïfs et basés sur la FFT pour la multiplication de polynômes avec des coefficients entiers, dans le but d'évaluer et de contraster leur efficacité dans la multiplication de polynômes.

Pour mettre en œuvre ces deux fonctionnalités, nous avons créé un total de quatre parties de fichiers de code:

1. `utile.c / .h`
2. `fastFourierTrans.c / .h`
3. `multPoly.c / .h`
4. `test.c`

ainsi qu'un fichier makefile. Nous expliquerons progressivement la mise en œuvre de chaque partie du code.

Les quatre parties:

1. Utile.c / .h

```
Dans cette partie, on fait principalement les actions aux base pour les objects :
- création / de l'object
- ajouter / supprimer le/les élément(s) de l'object
- réalizstions de l'arithmétique élémentaire pour les nombres complexes
```

Les objects :

- ***struct numComplex* :** Il contient deux *double:* double real(Partie réelle) et double imaginary(Coefficient de la partie imaginaire).
- ***struct polynomial* :** Il contient un tableau de numComplex avec deux champs *size* et *cpt.*

  cpt est le nombre de l'élément, *size* est la taille du tableau.

Les fonctions :

L'affichage:

- `void printComplexNumber(struct numComplex num)`

  Afficher une forme de nombre complexe complète $num.real + num.imaginary * i$.

- `void printComplexNumberVec(struct numComplex *numVec, int size)`

  Afficher un tableau de nombres complexes de taille size(comme un vecteur de nombres complexes).

Gestion pour numComplex:

- `struct numComplex randomConsNumber()`
- `struct numComplex creatComplexNum(double real, double imaginary);`
- `struct numComplex zeroComplexNum()`
- `int isEqNum(struct numComplex numA, struct numComplex numB)`

Gestion pour polynomial:

- `createPoly(int size);`
- `struct polynomial createRandomPoly(int size);`
- `void delPoly(struct polynomial poly);`
- `int addElement(struct numComplex el, struct polynomial* poly);`
- `int subElement(int index, struct polynomial* poly);`
- `int isEqPoly(struct polynomial polyA, struct polynomial polyB);`
- `struct numComplex* rootList(int n);`: Return toutes les nth root de unitaire.
- `int getNearestK(int size);`: Return $log_2(size) + 1$

Arithmétique élémentaire for complex number:

- `struct numComplex addComplexNumber(struct numComplex numA, struct numComplex numB)`

  Implémenter l'addition de deux nombres complexes en additionnant respectivement les parties réelles et les parties imaginaires.

- `struct numComplex multComplexNumber(struct numComplex numA, struct numComplex numB)`

  Implémenter la multiplication de deux nombres complexes en utilisant la formule de multiplication $(a + bi) * (c - di) = (ac + bd) + (bc - ad)i$ et renvoyer le résultat.

- `struct numComplex divComplexNum(int n, struct numComplex num)`

  Implémenter la division d'un nombre complexe par un entier, en divisant respectivement la partie réelle et la partie imaginaire par n, et renvoyer deux doubles représentant les nouvelles parties réelle et imaginaire.

2. fastFourierTrans.c / .h

```
Dans cette partie, on implémenter l'Algorithems de fft et fft inverse.
```

Les fonctions:

- `struct numComplex* extenstionVec(struct numComplex *original, int size, int k)`

Remplir le tableau de *original* avec $0 + 0i$, jusqu'à ce qu'il atteigne 2^k.

- `struct numComplex* coreFFT(struct numComplex *num, int size)`

- `struct numComplex* fft(struct numComplex *num, int size)`

  Avant qu'on appel la fonction `coreFFT`, on d'abord d'appel la fonction `extenstionVec`.

- `struct numComplex* fftInverse(struct numComplex *num, int size)`

  D'après qu'on appel la fonction fft, on fait la changement du tableau (data[i] =data[size-i]) et divide par n.

3. multPoly.c / .h

```
Implementation de mult de poly en methode naïve et en base de fft
```

Les fonctions:

- `struct polynomial NaiveMultPoly(struct polynomial polyA, struct polynomial polyB);`

  Cet algorithme est un algorithme naïf de multiplication de polynômes. Tout d'abord, on calcule la taille du polynôme résultant, puis on crée ce polynôme résultant. Ensuite, à l'aide de deux boucles fortes, on multiplie chaque paire de coefficients des polynômes d'entrée, puis on additionne le résultat au coefficient correspondant du polynôme résultant. Enfin, on définit le degré du polynôme résultant et on retourne le résultat obtenu.

- `struct polynomial fftMultPoly(struct polynomial polyA, struct polynomial polyB);`

  Tout d'abord, effectuer une transformation de Fourier rapide (FFT) sur les deux polynômes d'entrée pour obtenir deux résultats intermédiaires, resP et resQ. Ensuite, créer un polynôme temporaire, stocker, et multiplier les coefficients correspondants de resP et resQ, ajoutant le résultat à stocker. Enfin, obtenir le résultat final en effectuant une transformation de Fourier rapide inverse (FFT inverse) sur stocker.