Ecole Normale Supérieure de l'Enseignement Technique Al Mohammadia

Algorithmique & lang C

Produit de convolution : compte rendu

Encadré Par : M. Mohamed QBADOU

Réalisé par : ANADDAM Mohamed

Sommaire

Remerciements

Je tiens à exprimer ma gratitude envers M. Mohamed QBADOU pour son soutien et ses conseils précieux tout au long du module de C & Algorithmes.

1.2 Contenu du rapport :

Ce rapport vise à fournir une analyse détaillée du code med_anaddam_convolution.c en mettant l'accent sur les principaux aspects, notamment l'allocation dynamique de mémoire, le calcul de la trace d'une matrice carrée, le produit de convolution, et la libération de la mémoire.

II. Description du Code :

2.1 Allocation dynamique de la matrice :

La fonction get_random_matrice permet d'allouer dynamiquement une matrice carrée de taille N, initialisée avec des valeurs aléatoires entre 0 et 99 :

```c
// Med ANADDAM , II-CCN with pride !

int** get_random_matrice(int N) {
    int** matrice = (int**)malloc(N * sizeof(int*));
    for (int i = 0; i < N; i++) {
        matrice[i] = (int*)malloc(N * sizeof(int));
        for (int j = 0; j < N; j++) {
            matrice[i][j] = rand() % 100;
        }
    }
    return matrice;
}
```

2.2 Calcul de la trace de la matrice :

La fonction trace_matrice calcule et retourne la trace d'une matrice carrée, c'est-à-dire la somme des éléments de sa diagonale principale :

```
15
16  int trace_matrice(int** matrice, int N) {
17      int trace = 0;
18      for (int i = 0; i < N; i++) {
19          trace += matrice[i][i];
20      }
21      return trace;
22  }
23
```

2.3 Calcul du produit de convolution :

La fonction conv_matrices effectue le produit de convolution de deux matrices carrées A et B avec un filtre de taille (2p+1) x (2p+1) :

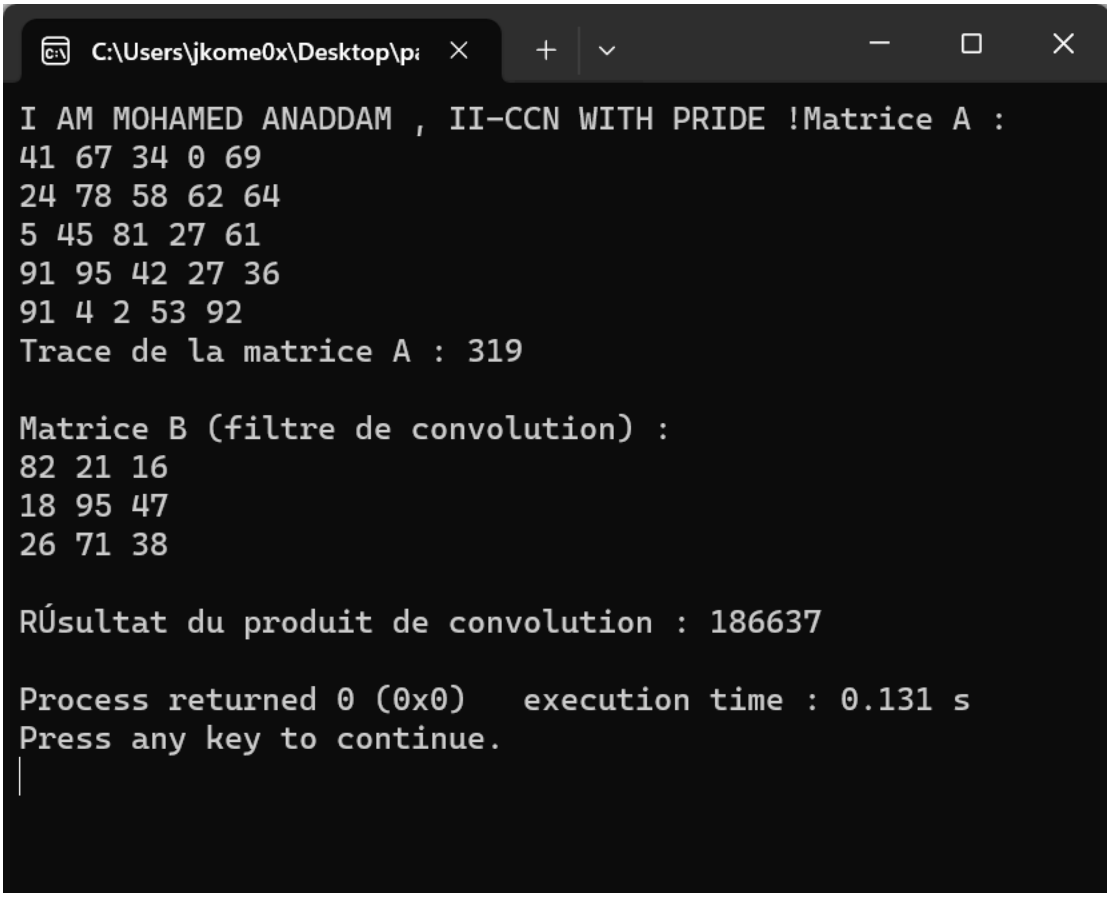
```
23
24  int conv_matrices(int** A, int** B, int N, int p) {
25      int result = 0;
26      for (int i = p; i < N - p; i++) {
27          for (int j = p; j < N - p; j++) {
28              for (int k = -p; k <= p; k++) {
29                  for (int l = -p; l <= p; l++) {
30                      result += A[i + k][j + l] * B[k + p][l + p];
31                  }
32              }
33          }
34      }
35      return result;
36  }
```

2.4 Libération de la mémoire :

La fonction free_matrice libère la mémoire allouée dynamiquement pour une matrice carrée :

```
39  void free_matrice(int** matrice, int N) {
40      for (int i = 0; i < N; i++) {
41          free(matrice[i]);
42      }
43      free(matrice);
44  }
45
```

III Démonstration :



IV. Retour d'Expérience :

4.1- Points forts du code :

- Gestion de la Mémoire : Le code met en œuvre une allocation dynamique de mémoire pour les matrices, suivie d'une libération appropriée de la mémoire à la fin du programme. Cela garantit une utilisation efficace des ressources et évite les fuites de mémoire.

- Modularité des Fonctions : Les fonctions du code sont bien modulaires, ce qui facilite la compréhension et la maintenance du code. Chaque fonction a une tâche spécifique, contribuant à la lisibilité du code.

- Utilisation de Commentaires : Les commentaires présents dans le code sont informatifs et facilitent la compréhension du code. Ils indiquent clairement la fonction de chaque partie du programme.

- Affichage des Résultats : Le programme affiche de manière claire et lisible les matrices générées aléatoirement, la trace de la matrice, la matrice de filtre de convolution, ainsi que le résultat du produit de convolution.

4.2- Difficultés rencontrées :

Dans la phase d'implémentation du code, deux principales considérations ont émergé. Premièrement, la taille fixe du filtre de convolution, définie par 2p+1, peut présenter une limitation dans l'adaptabilité du code à différents filtres de convolution avec des

dimensions variables. Afin d'optimiser la flexibilité du programme, une adaptation pourrait être envisagée pour permettre la manipulation de filtres de tailles diverses, offrant ainsi une solution plus adaptable à des contextes variés.

Deuxièmement, bien que le code fonctionne correctement, une opportunité d'amélioration réside dans l'exploration de techniques d'optimisation des performances, particulièrement cruciale lors du traitement de matrices de grandes dimensions. Une analyse approfondie des possibilités d'optimisation algorithmique pourrait permettre d'améliorer l'efficacité du code, optimisant ainsi son exécution, notamment pour des cas impliquant des matrices de grandes tailles. Cette démarche viserait à maximiser la réactivité du programme tout en minimisant sa complexité temporelle, contribuant ainsi à une exécution plus efficace dans des scénarios où des performances optimales sont cruciales.

V. Conclusion :

En conclusion, l'analyse détaillée du code med_anaddam_convolution.c a permis de mettre en lumière ses points forts, tels que la gestion rigoureuse de la mémoire, la modularité des fonctions, et la clarté de l'affichage des résultats. Cependant, des opportunités d'amélioration ont été identifiées, notamment la possibilité d'adapter le code pour traiter des filtres de convolution de tailles variables, offrant ainsi une flexibilité accrue. De plus, une exploration plus approfondie des techniques d'optimisation des performances, spécifiquement pour des matrices de grandes dimensions, pourrait contribuer à maximiser l'efficacité du programme.

Pour accéder au code source complet et explorer davantage son fonctionnement, vous pouvez trouver le code sur mon profil GitHub. N'hésitez pas à consulter le code, à le cloner, et à proposer des suggestions ou des améliorations. Votre contribution et engagement sont les bienvenus pour renforcer la qualité et la robustesse du code

profil GitHub : https://github.com/jkome/Produit_convolution