Projet final ACP

Programmation parallèle en Python appliquée à la bio-informatique

Needleman-Wunsch : Alignement global parallèle

Étudiant 1 : HALFAOUI Abderrahim Louay
N° Étudiant : 212133024640

Étudiant 2 : HOUACHE Elhadj Slimane
N° Étudiant : 212139088214

Décembre 2025

Table des matières

1 Introduction

Ce projet porte sur la parallélisation de l'algorithme Needleman-Wunsch pour l'alignement global de deux séquences d'ADN. Cet algorithme de programmation dynamique, d'une complexité temporelle $O(n^2)$, constitue un excellent candidat pour la parallélisation sur architectures multi-cœurs grâce à sa structure en anti-diagonales indépendantes.

Les objectifs sont :

— Implémentation séquentielle et profiling des goulots d'étranglement
— Proposition d'un modèle de parallélisation multi-cœurs
— Implémentation parallèle avec `multiprocessing`
— Mesure d'accélération et d'efficacité

2 Présentation de l'algorithme Needleman-Wunsch

2.1 Étapes principales

L'algorithme se décompose en trois phases :

1. **Initialisation** : Matrice $(n+1) \times (m+1)$ avec pénalités de gap sur les bords
2. **Remplissage** : Pour chaque cellule (i, j) : $\max(diag, up, left)$
3. **Traceback** : Reconstruction de l'alignement depuis (n, m) vers $(0, 0)$

2.2 Pseudo-code

```
Fonction Needleman_Wunsch(seq1, seq2, match, mismatch, gap):
    Cr er matrice F de taille (len(seq1)+1) x (len(seq2)+1)
    Initialiser F[i,0] = i*gap et F[0,j] = j*gap
    Pour i de 1    len(seq1):
        Pour j de 1    len(seq2):
            diag = F[i-1,j-1] + score(seq1[i],seq2[j])
            up = F[i-1,j] + gap
            left = F[i,j-1] + gap
            F[i,j] = max(diag, up, left)
    Traceback depuis F[n,m] pour construire alignements
    Retourner align1, align2, score optimal
```

3 Complexité

— **Temporelle** : $O(nm)$ - chaque cellule calculée en temps constant
— **Spatiale** : $O(nm)$ - stockage de la matrice complète

Pour des séquences biologiques typiques ($n, m \approx 10^3 - 10^4$), le temps devient prohibitif.

4 Profiling et goulots d'étranglement

Afin d'identifier les parties les plus coûteuses de l'algorithme Needleman–Wunsch, un profiling de la version séquentielle a été réalisé à l'aide de l'outil `cProfile` intégré à Python. Les expérimentations ont été menées sur des séquences d'ADN générées aléatoirement de taille $n = m = 500$.

4.1 Méthodologie de profiling

Le profiling consiste à mesurer le nombre d'appels et le temps d'exécution de chaque fonction durant l'exécution du programme. L'outil `cProfile` permet d'obtenir le temps cumulé (*cumulative time*) passé dans chaque fonction, ce qui facilite l'identification des goulots d'étranglement.

Chaque expérience a été répétée plusieurs fois avec des séquences différentes afin de vérifier la stabilité des mesures. Les résultats obtenus sont cohérents d'une exécution à l'autre, ce qui montre que le temps d'exécution dépend principalement de la taille des séquences et non de leur contenu.

4.2 Résultats du profiling

Un extrait représentatif des résultats du profiling est présenté ci-dessous :

```
259055 function calls in 0.106 seconds

ncalls  tottime  cumtime  function
---------------------------------
1       0.079    0.102    Needleman_Wunsch
250000  0.024    0.024    builtins.max
```

L'analyse de ces résultats montre que la fonction `Needleman_Wunsch` consomme plus de 95% du temps total d'exécution. À l'intérieur de cette fonction, l'appel répété à l'instruction `max` est effectué environ 250 000 fois, ce qui correspond exactement au nombre de cellules de la matrice de programmation dynamique ($n \times m$).

4.3 Identification du goulot d'étranglement

Le principal goulot d'étranglement est clairement identifié comme étant la phase de remplissage de la matrice de scores, réalisée par la double boucle imbriquée suivante :

```
for i in range(1, n + 1):
    for j in range(1, m + 1):
        Matrix[i][j] = max(diag, up, left)
```

Cette partie représente environ 75 à 80% du temps d'exécution total de l'algorithme et possède une complexité temporelle de $O(nm)$. Chaque cellule de la matrice nécessite un calcul simple mais dépend des valeurs des cellules voisines $(i-1, j)$, $(i, j-1)$ et $(i-1, j-1)$.

4.4 Analyse de la phase de traceback

La phase de traceback, utilisée pour reconstruire l'alignement optimal à partir de la matrice, a été mesurée séparément à l'aide de `time.perf_counter`. Le temps observé pour cette phase est de l'ordre de 4×10^{-5} secondes, ce qui représente une fraction négligeable du temps total d'exécution.

Par conséquent, la phase de traceback ne constitue pas un goulot d'étranglement et ne représente pas une cible pertinente pour la parallélisation.

4.5 Conclusion

L'analyse par profiling montre que le calcul de la matrice de programmation dynamique est de loin la partie la plus coûteuse de l'algorithme Needleman–Wunsch. Cette phase constitue donc le principal goulot d'étranglement et la meilleure candidate à une parallélisation. En raison des dépendances locales entre cellules, une parallélisation par anti-diagonales (ou *wavefront parallelism*) apparaît comme la stratégie la plus adaptée, ce qui sera détaillé dans la section suivante.

5 Modèle de parallélisation

L'algorithme Needleman-Wunsch repose sur une matrice de programmation dynamique où chaque cellule (i, j) dépend uniquement des trois cellules voisines $(i-1, j)$, $(i, j-1)$ et $(i-1, j-1)$. Cette dépendance locale impose une contrainte sur l'ordre de calcul des cellules.

Cependant, une observation clé permet la parallélisation : **toutes les cellules appartenant à une même anti-diagonale (i+j = constante) sont indépendantes entre elles**.

5.1 Principe des anti-diagonales

Soit une matrice F de taille $(n+1) \times (m+1)$. Les cellules peuvent être regroupées par anti-diagonales définies par :

$$d = i + j \quad avec \quad d \in [2, n+m]$$

Pour une anti-diagonale donnée d, toutes les cellules (i, j) telles que $i + j = d$ peuvent être calculées en parallèle, car leurs dépendances appartiennent uniquement à l'anti-diagonale précédente $(d-1)$.

5.2 Stratégie de parallélisation

La stratégie adoptée est la suivante :
— Calcul séquentiel des bords de la matrice (initialisation)
— Parcours des anti-diagonales une par une
— Parallélisation du calcul des cellules d'une même anti-diagonale
— Synchronisation implicite entre les anti-diagonales

Ce modèle respecte les dépendances de données tout en exploitant le parallélisme offert par les architectures multi-cœurs.

6 Implémentation parallèle

La parallélisation a été réalisée sur CPU en utilisant le module `concurrent.futures.ThreadPoolExe`
Ce choix est motivé par :
— La compatibilité avec le système Windows
— La simplicité de mise en œuvre
— L'absence de dépendances externes

Chaque anti-diagonale est traitée comme une étape synchronisée. À l'intérieur de cette étape, chaque cellule est calculée par un thread distinct.

6.1 Gestion du temps d'exécution

Le temps d'exécution est mesuré en **temps réel (wall-clock time)** à l'aide de la fonction `time.perf_counter()`. Il n'est pas pertinent de sommer les temps d'exécution des threads, car ceux-ci s'exécutent en parallèle.

Deux mesures sont collectées :

— Temps de remplissage de la matrice

— Temps de la phase de traceback

Ces mesures permettent une comparaison directe avec la version séquentielle.

6.2 Limitations

Bien que le calcul des anti-diagonales soit parallélisé, les limitations suivantes subsistent :

— Le *Global Interpreter Lock (GIL)* limite le parallélisme réel en Python

— Les accès concurrents à la matrice entraînent un surcoût de synchronisation

Malgré cela, cette implémentation permet de démontrer concrètement le modèle de parallélisation de l'algorithme.

7 Résultats de performance

Les tests ont été effectués sur des séquences ADN générées aléatoirement de taille $n = m = 100$. Chaque test a été répété plusieurs fois afin de garantir la stabilité des mesures.

7.1 Résultats du profiling

L'analyse avec `cProfile` montre que :

— Plus de 75% du temps d'exécution est consommé par la fonction `Needleman_Wunsch`

— L'appel à la fonction `max()` dans la double boucle représente le principal goulot d'étranglement

— La phase de traceback est négligeable (moins de 1%)

Ces résultats confirment que la phase de remplissage de la matrice est la cible principale de la parallélisation.

7.2 Comparaison séquentiel vs parallèle

Version	Temps matrice (s)	Temps traceback (s)
Séquentielle	$\approx 2.4 \times 10^{-3}$	$\approx 4.0 \times 10^{-5}$
Parallèle	Réduit partiellement	Inchangé

7.3 Analyse

Le gain de performance reste modéré pour des tailles de données réduites, en raison :

— du coût de création des threads

— du GIL de Python

Cependant, pour des séquences plus longues, la parallélisation par anti-diagonales devient plus avantageuse et constitue une base solide pour une implémentation GPU ou C/C++.