Computationele Biologie
Academiejaar 2019-2020

Laurens Debackere
laurens.debackere@ugent.be

Project: FragGeneScan

Inleiding

Voor het project Computationele Biologie koos ik ervoor om versie 1.31 van FragGeneScan opnieuw te implementeren in Rust (versie 1.43.0), met oog op het verbeteren van de modulariteit en verstaanbaarheid van de code en het voorzien van correct werkende multithreading voor de uitvoering van het Viterbi-algoritme. Om een commandolijn-interface te voorzien maakte ik gebruik van de bibiliotheek clap[1], de verwerking van het FASTA-invoerformaat gebeurt met behulp van Rust-Bio[2] en om beter het aanwezige data parallellisme te kunnen benutten voegde ik Rayon[3] toe.

Geïmplementeerde verbeteringen

Mijn aanpassingen en verbeteringen aan de werking van de toepassing ten opzichte van de bestaande implementatie van FragGeneScan zijn voornamelijk de volgende:

- Waar de toepassing voorheen gebruik maakte van temp-files tijdens de uitvoering werden deze in mijn implementatie vervangen door een aantal *structures* waarmee de resultaten van het Viterbi-algoritme worden teruggegeven, dit heeft als voordeel dat de methoden voor het uitvoerformaat konden worden afgesplitst en dat de code eventueel ook als library zou kunnen worden aangeboden.
- Met behulp van Rayon werd er werkende parallellisatie voorzien bij het verwerken van meerdere sequenties, dit in tegenstelling tot de bestaande implementatie waar race-condities voor problemen zorgden.
- Onveilige geheugentoegangen (o.a. lezen buiten een gealloceerde array) werden verholpen.
- De niet-gedocumenteerde `format`-parameter werd uit de toepassing weggehaald.
- Het inlezen van trainingsfiles gebeurt nu in een aparte module van de toepassing en werd duidelijker gestructureerd door opsplitsing in meerdere functies.

Daarnaast werden ook bepaalde constructies uit de C-code omgezet naar hun equivalent in Rust, of werd code verplaatst naar aparte modules om de toepassing beter te structureren.

Structuur van het programma

De toepassing bestaat uit een aantal verschillende modules die hieronder verduidelijkt worden:

src/main.rs	Implementeert de commandolijn interface en roept de Viterbi-methode aan
src/dna_helpers.rs	Implementeert de methoden voor conversie tussen o.a. DNA sequenties en Aminozuren.

[1] https://github.com/clap-rs/clap

[2] http://rust-bio.github.io/

[3] https://github.com/rayon-rs/rayon

src/helpers.rs	Implementeert een aantal utility methoden voor I/O gebruik en foutafhandeling
src/output.rs	Implementeert de methode verantwoordelijk voor het wegschrijven van de resultaat *structures* uit de Viterbi methode naar de commandolijn.
src/train.rs	Implementeert de structures en methoden gebruikt voor het inlezen van de training files.
src/viterbi.rs	Implementeert het Viterbi-algoritme en de *structures* gebruikt voor de resultaten.

Benchmarks en evaluatie met UMGAP

Om de uitvoeringstijden van mijn implementatie te kunnen evalueren koos ik ervoor de vergelijking te maken ten opzichte van de originele implementatie en niet FragGeneScan+(+) gezien deze enkele fouten bevatten die ten koste blijken te gaan van sensitiviteit en precisie. De impact van mijn verbeteringen valt vooral op in de uitvoeringstijden van invoer die geen volledige genoomsequentie voorstelt, immers daar kunnen meerdere instanties van het Viterbi-algoritme parallel worden uitgevoerd. De impact is minder significant in het geval van volledige genoomsequenties waar het potentieel om de code te parallelliseren een pak beperkter is.

Wanneer we met behulp van UMGAP de precisie en sensitiviteit van FragGeneScan 1.31, de laatste versie van FragGeneScan++ (22 augustus 2019) en mijn implementatie vergelijken over 10 uitvoeringen merken we op dat deze waarden van mijn implementatie gemiddeld zeer dicht bij deze van FragGeneScan 1.31 liggen, en beide versies scoren hoger dan FragGeneScan++. (zie figuur 2)

Laurens Debackere
laurens.debackere@ugent.be

Bijlagen

Figuur 1: Vergelijking van de uitvoeringstijden in seconden, gemiddelde over 10 uitvoeringen

Invoerbestand	FragGeneScan1.31	Rust-implementatie
example/NC_000913 (2 threads)	4.14s	3.06s
example/NC_000913 (1 thread)	4.13s	3.12s
example/NC_000913-454 (2 threads)	9.45s	2.75s
example/NC_000913-454 (1 thread)	14.34s	5.24s

Figuur 2: Vergelijking van sensitiviteit en precisie met UMGAP, gemiddelde over 10 uitvoeringen.

	FragGeneScan1.31	FragGeneScan++	Rust-implementatie
Sensitiviteit	9284	9260.6	9283.4
Precisie	4605	4460.8	4598