ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΕΠΕΞΕΡΓΑΣΤΩΝ & ΥΛΙΚΟΥ

ΗΡΥ 418 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΠΑΡΑΛΛΗΛΩΝ ΚΑΙ ΚΑΤΑΝΕΜΗΜΕΝΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2018-19

Εργασία εξαμήνου Εαρινό εξάμηνο 2018-19

ΕΝΤΡΙΤ ΜΕΤΑΙ – 2014030005

ΝΤΟΥΣΑΚΗΣ ΓΡΗΓΟΡΙΟΣ - 2014030081

Εισαγωγή:

Σκοπός της εργασίας μας ήταν να δημιουργήσουμε διαφορετικούς τρόπους προκειμένου να εκτελέσουμε τον αλγόριθμο Smith-Waterman και να παραλληλοποιησουμε τον αλγόριθμο με τουλάχιστον 2 διαφορετικούς τρόπους με σκοπό να παρατηρήσουμε την εφαρμογή του παραλληλισμού πάνω στον κώδικα μας για να εκτελέσουμε τοπική ευθυγράμμιση ακολουθιών στην απλή περίπτωση εφαρμογής του με linear gap penalty. Επιπλέον έπρεπε να υλοποιήσουμε διαφορετικά granularities για το κώδικα μας και να αξιοποιήσουμε το Open MP και τα Threads στην υλοποίηση μας και να συγκρίνουμε τυχόν διάφορες που παρατηρούμε κατά την δομή του κώδικα και τους χρόνους εκτέλεσης σε κάθε μια περίπτωση ξεχωριστά.

Περιγραφή Αλγορίθμου

Ο αλγόριθμος Smith–Waterman ανακαλύφθηκε το 1981 από τους Michael S. Waterman και τον Temple F. Smith.

Έπρεπε να δημιουργήσουμε ένα σειριακό πρόγραμμα το οποίο θα δέχεται σαν εισόδους από το command line μας ένα σύνολο από arguments μέσω των οποίων θα περνάμε στο προγραμμα μας τις εισόδους που θα αξιοποιήσουμε για την εκτέλεση του αλγορίθμου μας. Τα arguments που έπρεπε να αξιοποιήσουμε είναι τα έξης:

- ID : είναι string το οποίο χρησιμοποιείται για τη δημιουργία του ζητούμενου αρχείου εξόδου του προγράμματος μας.
- PATH : δίνει το path σε ένα αρχείο εισόδου τύπου text
- INT1 : προσδιορισμούς scoring scheme που χρησιμοποιείται για το match
- INT2 : προσδιορισμούς scoring scheme που χρησιμοποιείται για το mismatch
- INT2 : προσδιορισμούς scoring scheme που χρησιμοποιείται για το gap

Όσον άφορα για το Open Mp και το P threads προσθέσαμε και το έξης arguments

- Threads : που περνάει σαν όρισμα τον αριθμό των threads που θα χρησιμοποιηθούν για την παραλληλοποίηση

Ο αλγόριθμος αξιοποιήσαμε τα παραπάνω δεδομένα και ένα δεδομένο data-set που εμπεριέχει ένα σύνολο από ζευγάρια που θα δοθούν στο πρόγραμμα μας προκειμένου να προκύψουν τα αποτελέσματα που ζητούνται από την εργασία τα οποία αναφέρονται σε :

Match : σύνολο επιτυχών trace back steps στον αλγόριθμο

Score : Το τελικό σκορ που προέκυψε μετά την επιτυχή εφαρμογή του αλγορίθμου και την εύρεση ζευγαριών ακολουθιών

Strart : Σημείο εκκίνησης του trace back step από το scoring matrix

Stop : Σημείο τερματισμού του trace back step στο scoring matrix

Επιπλέον ο κώδικας χρειάζεται να μας παρουσιάσει με ένα σύνολο πληροφοριών που αφορούν ενδεικτικά στοιχειά της απόδοσης του.

A) συνολικός αριθμός ζευγαριών ακολουθιών Q-D,

B) συνολικός αριθμός κελιών που πήραν τιμή,

C) συνολικός αριθμός από traceback steps,

D) συνολικός χρόνος εκτέλεσης προγράμματος,

E) συνολικός χρόνος υπολογισμού κελιών,

F) συνολικός χρόνος για το traceback,

G) συνολικός αριθμός κελιών που παίρνουν τιμή στη μονάδα του χρόνου

H) CUPS βάση του χρόνου υπολογισμού κελιών.

Παρακάτω θα παρουσιάσουμε τις υλοποιήσεις που έγιναν πάνω στα ζητούμενα της άσκησης και πιο συγκεκριμένα στη σειριακή , Open Mp και Threads υλοποίηση.

Σειριακή Υλοποίηση:

Υλοποιήσαμε το αλγόριθμο του Smith - Waterman στην c γλώσσα προγραμματισμού. Ο αλγόριθμος αυτός δουλεύει ως εξής: Ο χρήστης δίνει στο πρόγραμμα ένα αρχείο εισόδου με πολλά διαφορετικά ζευγάρια από string. Διατρέχει το κάθε ζευγάρι από strings και προσπαθεί να βρει την μέγιστη υπακολουθία που υπάρχει με βάση τα match,mismatch και gap, που δίνονται από τον χρήστη σαν input στην εκκίνηση του προγράμματος. Τυπώνει ένα output αρχείο με τα αποτελέσματα του προγράμματος.

Υπάρχουν πολλοί διαφορετικοί τρόποι που μπορούσαμε να υλοποιήσουμε τον αλγόριθμο αυτό. Εμείς διαλέξαμε ένα σχετικά απλό αλγόριθμο που αν και είναι αρκετά πιο αργός από άλλες τεχνικές, όμως προσφέρει καλύτερη και ευκολότερη διαχείρισή μνήμης. Αυτό ήταν απαραίτητο καθώς ο στόχος της εργασίας ήταν να δούμε και κατανοήσουμε πως με την χρήση της παραλληλοποίησης μπορούμε να βελτιώσουμε την ταχύτητα ενός προγράμματος που έχουμε υλοποιήσει.

Για να επιλύσουμε το πρόβλημα που μας δόθηκε χρησιμοποιήσαμε δύο βασικές συναρτήσεις. Η μία ήταν η *fill_table*, που ο σκοπός της ήταν να γεμίζει ένα πινάκα με scores ανάλογα την ομοιότητα των δύο string που μας δόθηκαν απο το αρχείο εισόδου. Η άλλη ήταν η *write_table* που έβρισκε τις μέγιστες υπακολουθίες και τις έγραφε στο αρχείο εξόδου.

Αναλυτικότερα, το πρόγραμμα ξεκινάει από το command line της linux διανομής που χρησιμοποιούμε. Η εισοδός μας είναι:

./project1 -name ID -input PATH -match INT1 -mismatch INT2 -gap INT3

Το project1 είναι το όνομα του εκτελέσιμου αρχείου. Το ID δίνει το όνομα του output αρχείου, που στην περίπτωση που δεν υπάρχει το δημιουργεί, ενω στην περίπτωση που υπάρχει το διαγράφει και δημιουργεί ένα καινούργιο. Το path μας δίνει την διαδρομή της μνήμης προς το input αρχείο. Το match, μας δίχνει πόσο πρέπει να αυξήσουμε το score σε περίπτωση όμοιου string, το mismatch δίχνει πόσο πρέπει να το μειώσουμε σε περίπτωση διαφορετικού string και το gap την μείωση σε περίπτωση κενού. Μία παρατήρηση είναι πως όταν δίναμε ένα μεγάλο path σε input δεν μπορούσε να το διαχειριστεί το terminal και έβγαζε λάθος αποτελέσματα.

Μόλις εισάγαγε τα δεδομένα μας το πρόγραμμα κάνει τους απαραίτητους ελέγχους για να σιγουρέψει ότι έχουμε δώσει όλα τα σωστά ορίσματα. Αποθηκεύει σε διαφορετικές μεταβλητές όλα τα input καθώς είναι απαραίτητα για την σωστή λειτουργία του προγράμματος. Μόλις επιβεβαιώσει ότι όλα είναι σωστά προχωράει και ανοίγει το αρχείου εισόδου. Στις πρώτες τέσερεις γραμμές είναι γραμμένα τα pairs,qmin,qmax και τα dsizeall. Εμείς χρησιμοποιήσαμε μόνο τα pairs για να ξέρουμε πόσα διαφορετικά string πρέπει να ελέγξουμε για ομοιότητες και το dsizeall για να ορίσουμε πίνακες με αρκετό μέγεθος έτσι ώστε να μην έχουμε fregmentation error.

Ορίζουμε δύο πίνακες με char* ίσους με το *Allpair * sizeof(char)* με σκοπό να αποθηκεύσουμε μέσα τις δύο ακολουθίες από string ή αριθμούς. Για να βρούμε τα qline και dline υλοποιήσαμε μία συνάρτηση get. Η συνάρτηση αυτή δουλεύει πρώτα και διαβάζει όλους τους χαρακτήρες του qline, αγνοόντας κενά,αλλαγές γραμμής και χρήση του tab, μέχρι να βρεί το 'D'. Μόλις βρει το D δουλεύει με τον ίδιο ακριβώς τρόπο μέ μόνο διαφοροποιήσει ότι τερματίζει μόλις βρεί ή το επόμενο 'Q' αν υπάρχουν πολλά pairs ή το τέλος του αρχείου όταν υπάρχει μόνο ένα.

Αμέσως μετά όρίζουμε ένα πίνακα δύο διαστάσεων που με την χρήση της *fill_table* θα τον γεμίσουμε με τα scores μας. Τον αρχικοποιούμε σε όλα τα κελιά του με το 0 και καλούμε την fill_table. Η fill_table ελέγχει εάν έχουμε ιδιους χαρακτήρες ή όχι για να δώσει τα ανάλογα scores στο σωστό κελί του πίνακα. Δηλαδή έαν βρεί ίδιο χαρακτήρα στο dline και στο qline υπολογίζει 3 τιμές και βάζει την μεγαλύτερη στο κελι του πίνακα. Ανάλογα δουλεύει και εάν δεν βρει ομοιότητα.

Μόλις επιστρέψει ο πίνακας γεμάτος με τις σωστές τιμές τότε, ξεκινάμε να γράφουμε τα αποτελέσματα μας στο αρχείο εξόδου. Αυτό γίνεται μέσω της write_table. Ψάχνει να βρει το max νούμερο στό πίνακα και κάνει traceback προς τα πίσω και το γράφει. Αυτή η διαδικασία συμβαίνει όσες φορές βρει max value στον πίνακα.

Open Mp:

Στην δεύτερη φάση της άσκησης μας ζητήθηκε να αλλάξουμε το πρόγραμμά μας και να το φτιάξουμε να δουλεύει στα πρότυπα του open mp. Παραλληλοποιήσαμε τα δύο διπλά for loops που καλούμε την fill_table και την write_table. Στην αρχή, αντιμετωπίσαμε πρόβλημα καθώς το open mp δεν μπορεί να διαχειριστεί πολύπλοκα if και διεργασίες μέσα στην κεντρική main. Έτσι για να λύσουμε το πρόβλημα δημιουργήσαμε τις παραπάνω συναρτήσεις (fill_table – write_table) εκτός της main και επιτύχαμε την παραλληλοποίηση που μας ζητείτο.

Η παραλληλοποίηση γινεται πρακτικά με μία εντολή. Απλά συμπεριλάβαμε το αρχείο:

#include <omp.h>

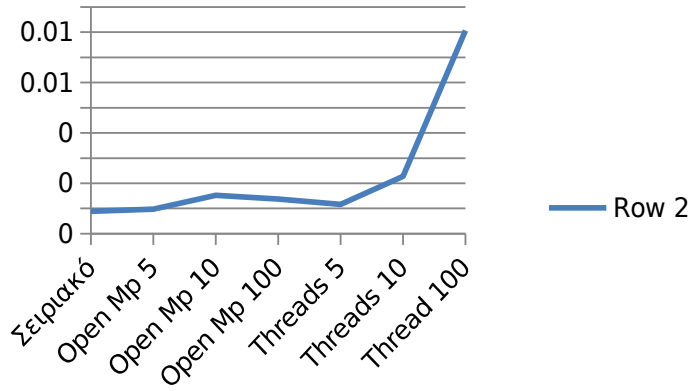
Και αμέσως πάνω από το διπλό for loop που γεμίζουμε τον πίνακα (fill_table):

#pragma omp parallel for default(shared) private(i,j) reduction(+:value1) reduction(+:value2) reduction(+:value3)

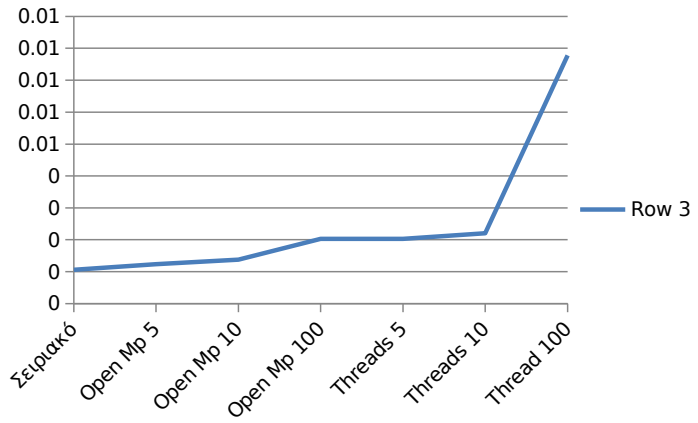
Μεγάλη διαφορά είδαμε όταν εισήγαμε τα ορίσματα reduction.

Αποτελέσματα μετρήσεων στα dataset:

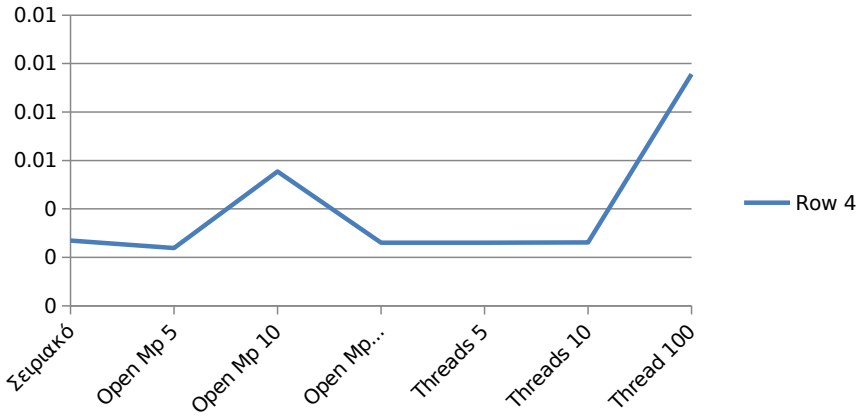
D1:



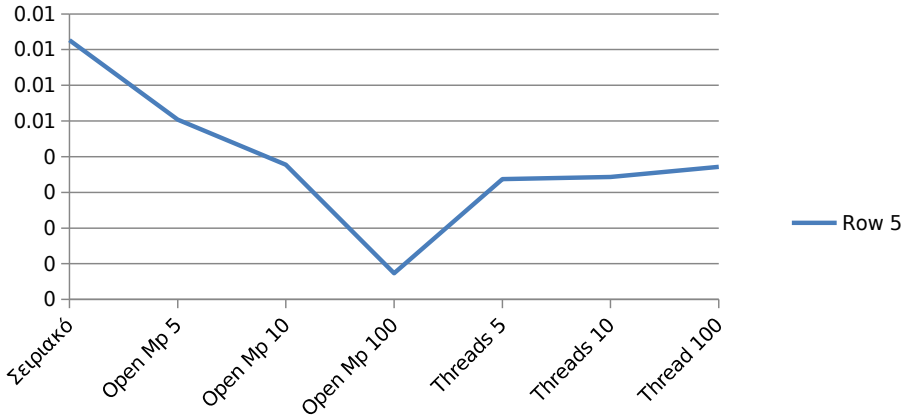
D2:



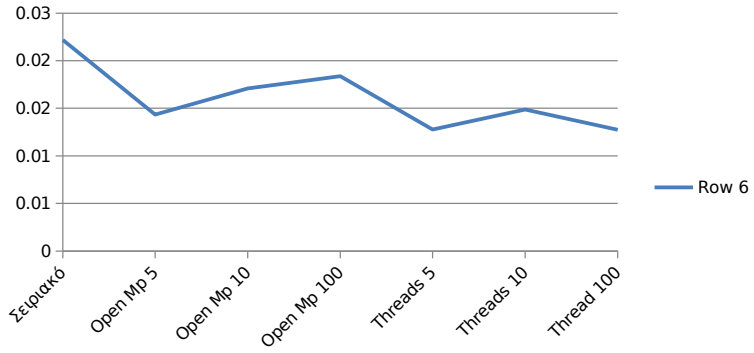
D3:


0.01
0.01
0.01
0.01
0
0
0
Σειριακό
Open Mp 5
Open Mp 10
Open Mp...
Threads 5
Threads 10
Thread 100
Row 4

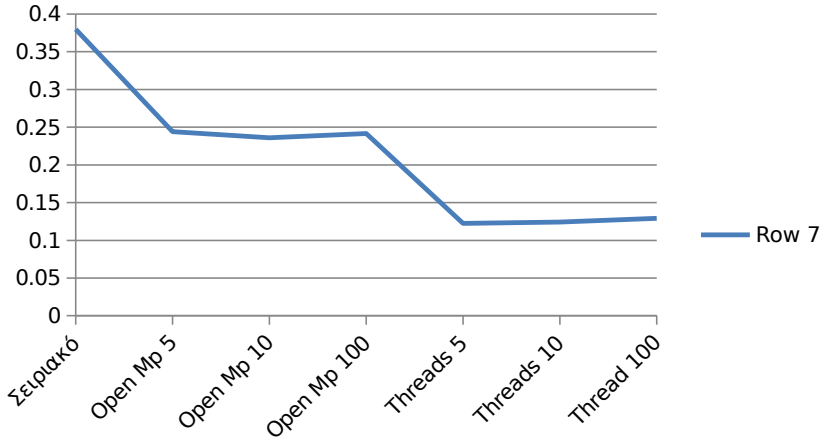
D4:


0.01
0.01
0.01
0.01
0
0
0
0
0
Σειριακό
Open Mp 5
Open Mp 10
Open Mp 100
Threads 5
Threads 10
Thread 100
Row 5

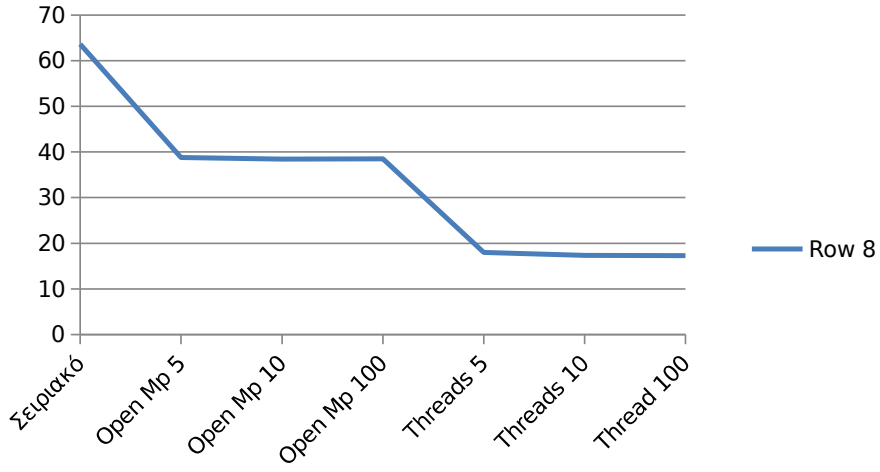
D5:


0.03
0.02
0.02
0.01
0.01
0
Σειριακό
Open Mp 5
Open Mp 10
Open Mp 100
Threads 5
Threads 10
Thread 100
Row 6

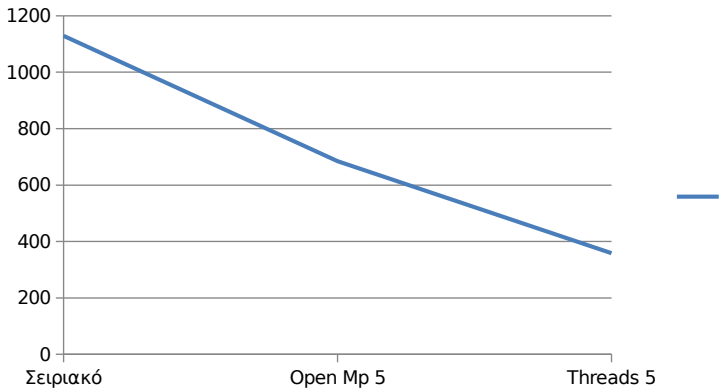
D6:


0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
Σειριακό
Open Mp 5
Open Mp 10
Open Mp 100
Threads 5
Threads 10
Thread 100
Row 7

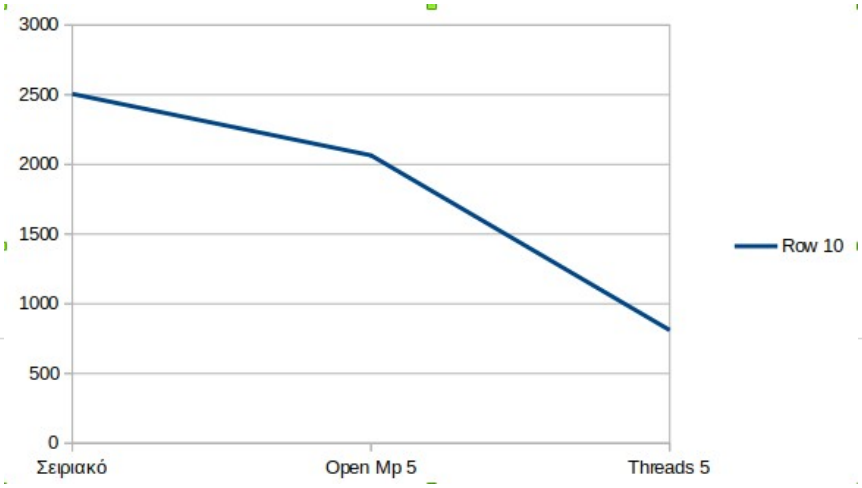
D7:


70
60
50
40
30
20
10
0
Σειριακό
Open Mp 5
Open Mp 10
Open Mp 100
Threads 5
Threads 10
Thread 100
Row 8

D8:


1200
1000
800
600
400
200
0
Σειριακό
Open Mp 5
Threads 5

D9:


3000
2500
2000
1500
1000
500
0
Σειριακό
Open Mp 5
Threads 5
Row 10

Συμπεράσματα:

Παρατηρούμε ότι στα πολύ μικρά αρχεία μέχρι το D4 δεν υπάρχουν διαφορές ουσιαστικές μεταξύ των τρόπο υλοποίησης του αλγορίθμου, μάλιστα σε κάποιες περιπτώσεις που δίνουμε πολλά threads αργεί περισσότερο από το σειριακό. Από το D5 και μετά παρουσιάζονται ουσιαστικές διαφορές στους χρόνους εκτέλεσης. Βλέπουμε ότι το open mp ρίχνει ακόμα πιο αποδοτικά τους χρόνους (για παράδειγμα στο D9 κατεβαίνει από 2505 σε περίπου 2064). Το threads αντίστοιχα το ρίχνει από 2505 σε περίπου 810, δηλαδή 3 φορές κάτω τον χρόνο.

Καταλαβαίνουμε λοιπόν ότι οι παραλληλοποίηση των προγραμμάτων επιφέρει ουσιαστικές διαφορές στην αποδοτικότητα και στην λειτουργικότητα τον προγραμμάτων αρκεί να διαλέγουμε τις σωστές συνθήκες και τρόπους παραλληλισμού.