Technical University of Crete

Αρχιτεκτονική Παράλληλων και Κατανεμημένων Υπολογιστών

ΑΛΓΟΡΙΘΜΟΣ SMITH-WATERMAN: POSIX THREADS, OPENMP

Καλογεράκης Στέφανος | Ζερβάκης Αρης

Εισαγωγή

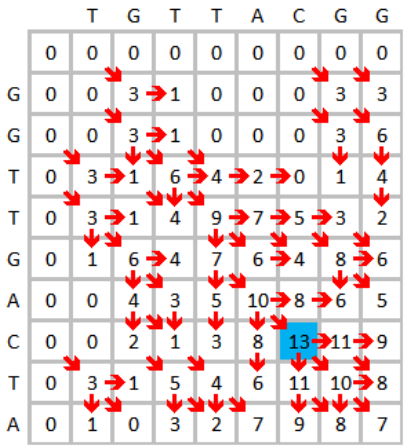
Η υλοποίηση του πρώτου πρότζεκτ μπορούμε να θεωρήσουμε ότι χωρίζεται σε κάποια βασικά μέρη. Στο πρώτος μέρος, προχωρήσαμε σε μια σειριακή υλοποίηση του αλγόριθμου Smith-Waterman, ενώ στο δεύτερο και τρίτο μέρος, βασιζόμενοι στον σειριακό κώδικα, πραγματοποιήσαμε και τις αντίστοιχες παράλληλες υλοποιήσεις με τις μεθόδους των POSIX Threads και του OpenMp με δύο διαφορετικά granularity στην κάθε περίπτωση. Σε όλες τις περιπτώσεις λαμβάναμε ως είσοδο αρχεία δεδομένων που επεξεργαζόμασταν κατάλληλα υλοποιώντας τον ζητούμενο αλγόριθμο παράγοντας ένα τελικό αρχείο συγκεκριμένου format βασισμένο στην εκφώνηση της άσκησης. Παρακάτω, ακολουθεί πλήρης ανάλυση όλων των βημάτων υλοποίησης της άσκησης.

Αλγόριθμος

Ο αλγόριθμος Smith-Waterman αποτελεί έναν αλγόριθμο δυναμικού προγραμματισμού, ο οποίος χρησιμοποιείται κατά βάση στην βιοπληροφορική για την τοπική στοίχιση βιολογικών ακολουθιών. Αρχικό στάδιο του αλγορίθμου είναι η συμπλήρωση ενός πίνακα βαθμολογίας ο οποίος προκύπτει από την σύγκριση των δύο ακολουθιών m και n. Ο πίνακας αυτός είναι διαστάσεων (m+1)*(n+1) με την πρώτη γραμμή και στήλη να συμπληρώνονται με την τιμή 0. Τα υπόλοιπα στοιχεία του πίνακα συγκρίνονται βάσει της σχέσης

$$Mi,j \;=\; Max\{Mi-1,j-1+SCOREi,j\,,Mi,j-1+GAP,Mi-1,j+GAP,0\}$$

Το επόμενο βήμα είναι το backtracking του αλγορίθμου. Πιο συγκεκριμένα, εντοπίζουμε από τον πίνακα βαθμολογίας, την μεγαλύτερη τιμή και ακολουθούμε με βήματα προς τα πίσω το μέγιστο από το στοιχείο αριστερά, πάνω και διαγώνια του κελιού που βρισκόμαστε μέχρι να φτάσουμε διαγώνια την τιμή 0. Τελευταίο στάδιο του αλγόριθμου είναι η ευθυγράμμιση που αποτελεί ουσιαστικά την απεικόνιση του backtracking. Να επισημάνουμε, ότι στην περίπτωση που στον πίνακα βαθμολόγησης συναντούσαμε παραπάνω από μια φορές μια μέγιστη τιμή τότε πραγματοποιούσαμε backtracking για όλους τους συνδυασμούς.



Εικόνα 1:Ενδεικτικό στιγμιότυπο αλγορίθμου Smith-Waterman

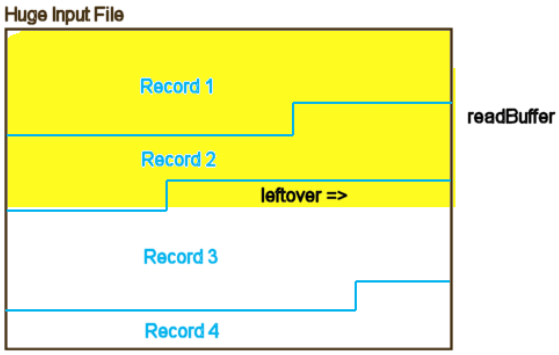
Υλοποίηση

Σειριακός κώδικας

Βασικό κομμάτι του σειριακού κώδικα αλλά και ο πυρήνας όλων των λύσεων ήταν η επεξεργασία των ακολουθιών βάσει των βημάτων που αναλύθηκαν παραπάνω. Στα πλαίσια όμως του πρότζεκτ, δεν ήταν μοναδική απαίτηση η επίλυση του αλγορίθμου.

Αρχικά, κληθήκαμε να δεχόμαστε ως ορίσματα τις τιμές των **match**, **mismatch** και **gap** που ήταν απαραίτητες για τον υπολογισμό των βημάτων του αλγόριθμου, μιας τιμής **input**(directory αρχείου εισόδου με τις ακολουθίες), μιας τιμής **ID**(συμπεριλαμβάνεται στο όνομα του τελικού αρχείου) και στις παράλληλες υλοποιήσεις μια τιμή **threads**(αριθμός threads). Οι παράμετροι αυτοί μάλιστα μπορούν να δίνονται με τυχαία σειρά από τον χρήστη μέσω γραμμής εντολών. Σε περίπτωση που κάποια από τις τιμές απουσιάζει σταματάει η λειτουργία του προγράμματος μας.

Ακόμη ένα απαιτητικό στάδιο ήταν το διάβασμα των αρχείων εισόδου. Στα Dataset που είχαν δοθεί περιείχαν πολύ μεγάλες σε μέγεθος ακολουθίες καθιστώντας την χρήση έτοιμων εντολών από την c για το διάβασμα τους ως μη επαρκή καθώς παρατηρήθηκε απώλεια πληροφορίας. Έτσι, πραγματοποιήσαμε το διάβασμα των αρχείων με την χρήση buffer



Εικόνα 2: Buffer διαβάσματος του αρχείου Input

Ο buffer μας δεσμεύει κάθε φορά από αρχείο ένα μήκος ακολουθίας το οποίο σίγουρα περιλαμβάνει ολόκληρο μια ακολουθία Q και D και ένα μικρό μέρος της επόμενης ακολουθίας Q. Μόλις βρεθούν και απομονωθούν τα Q, D τότε επιστρέφεται η θέση που βρίσκεται το επόμενο Q. Από αυτό το στοιχείο μέχρι το τέλος του αρχικού buffer γεμίζουν την αρχή του επόμενου buffer και προχωράμε με την ίδια τεχνική μέχρι να διαβαστεί όλο το αρχείο.

Τελευταίο στάδιο μετά και την εφαρμογή του αλγορίθμου ήταν το γράψιμο σε ένα αρχείο εξόδου τις ακολουθίες που είχαμε διαβάσει από την είσοδο με τα αποτελέσματα που προέκυψαν μετά από την εφαρμογή του αλγορίθμου σε συγκεκριμένο format.

Παράλληλος Κώδικας

Στο παράλληλο κομμάτι της εργασίας σκοπός ήταν να παραλληλοποίησουμε με δυο διαφορετικά granularity τόσο με την μέθοδο OpenMp όσο και με την μέθοδο Pthreads. Μετά από ενδελεχής έρευνα αναφορικά με τις μεθόδους παραλληλοποίησης του συγκεκριμένου αλγόριθμου και λαμβάνοντας υπόψιν όλες τις παραμέτρους του προβλήματος προχωρήσαμε σε μία υλοποίηση fine-grained βασισμένη στο γέμισμα του πίνακα βαθμολογίας, και σε μία coarse-grained βασισμένη στις ακολουθίες που δεχόμαστε ως είσοδο.

Fine-grained

Μια fine-grained μέθοδος βασίζεται στην λογική ότι τα tasks είναι σχετικά μικρά σε αναλογία με τον χρόνο εκτέλεσης του εκάστοτε προγράμματος με τα δεδομένα να μεταφέρονται συνεχώς στον επεξεργαστή.

Στον αλγόριθμο μας, και μετά την εκτέλεση και του σειριακού κώδικα επαληθεύσαμε ότι την μεγαλύτερη χρονική διάρκεια υπολογισμών την πραγματοποιεί το γέμισμα του πίνακα βαθμολόγησης. Αυτός ήταν ο λόγος που στρέψαμε την προσοχή μας σε αυτό το σημείο κατά την πρώτη υλοποίηση. Η μέθοδος που ακολουθήσαμε για το γέμισμα του πίνακα ήταν το γέμισμα βάσει αντιδιαγωνίων όπως φαίνεται και στην παρακάτω εικόνα

	–	A	C	A	C	A	C	T	A
–	0	0	0	0	0	0	0	0	0
A	0	2	1	2	1	2	1	0	2
G	0	1	1	1	1	1	1	0	1
C	0	0	3	2	3	2	3	2	1
A	0	2	2	5	4	5	4	3	4
C	0	1	4	4	7	6	7	6	5
A	0	2	3	6	6	9	8	7	8
C	0	1	4	5	8	8	11	10	9
A	0	2	3	6	7	10	10	10	12

Εικόνα 3: Μέθοδος αντιδιαγωνίων

Δυστυχώς, η φύση του αλγόριθμου και οι εξαρτήσεις που υπάρχουν μεταξύ των υπολογισμών δεν προσδίδουν πολλές εναλλακτικές παραλληλοποίησης. Με την συγκεκριμένη τεχνική όμως, όπως θα παρουσιάσουμε και στην συνέχεια συναντήσαμε αρκετά προβλήματα-περιορισμούς μην επιτυγχάνοντας σε πολλές περιπτώσεις τον σκοπό της παράλληλης εκτέλεσης δηλαδή την ταχύτερη λειτουργία του αλγορίθμου.

Γενική υλοποίηση αλγόριθμου δυναμικού προγραμματισμού

Θεωρώντας έναν πίνακα διαστάσεων (n+1)*(m+1) όπου m και n αντίστοιχα το μήκος των ακολουθιών προκύπτει ότι οι αντιδιαγώνιοι του είναι n-m-1 καθώς γνωρίζουμε ότι οι τιμές της πρώτης γραμμής και στήλης είναι μηδενικές ενώ αφαιρούμε και το σημείο O(0,0).

Στην συνέχεια, υπολογίζουμε σε ρουτίνα για την κάθε αντιδιαγώνιο το μήκος της, αλλά και τις αρχικές συντεταγμένες που ορίζουν την θέση που αρχίζουν οι συγκρίσεις για την εκάστοτε αντιδιαγώνιο. Τέλος πραγματοποιούμε την επανάληψη μας σε όλο το μήκος της αντιδιαγωνίου μέχρι να φτάσουμε στο τέλος του πίνακα. Να σημειώσουμε ότι κατά την την παραλληλοποίηση δεν επιτρέπεται ενημέρωση του πίνακα τυχαία λόγω των εξαρτήσεων μεταξύ των κελιών και κάθε αντιδιαγώνιος πρέπει να εκτελείται αυστηρά μέχρι το τέλος της.

POSIX Threads

Στην προσπάθεια μας για εύρεση μιας σωστής και αποτελεσματικής λύσης οι υλοποιήσεις μας διαφοροποιήθηκαν σε μεγάλο βαθμό ενώ ο τρόπος που πραγματοποιήσαμε τις συγκρίσεις και την ενημέρωση του πίνακα μεταβλήθηκε πολλές φορές. Δυο υλοποιήσεις άξιες αναφοράς είναι οι παρακάτω

Παραλληλοποίηση ολόκληρου του αλγόριθμου

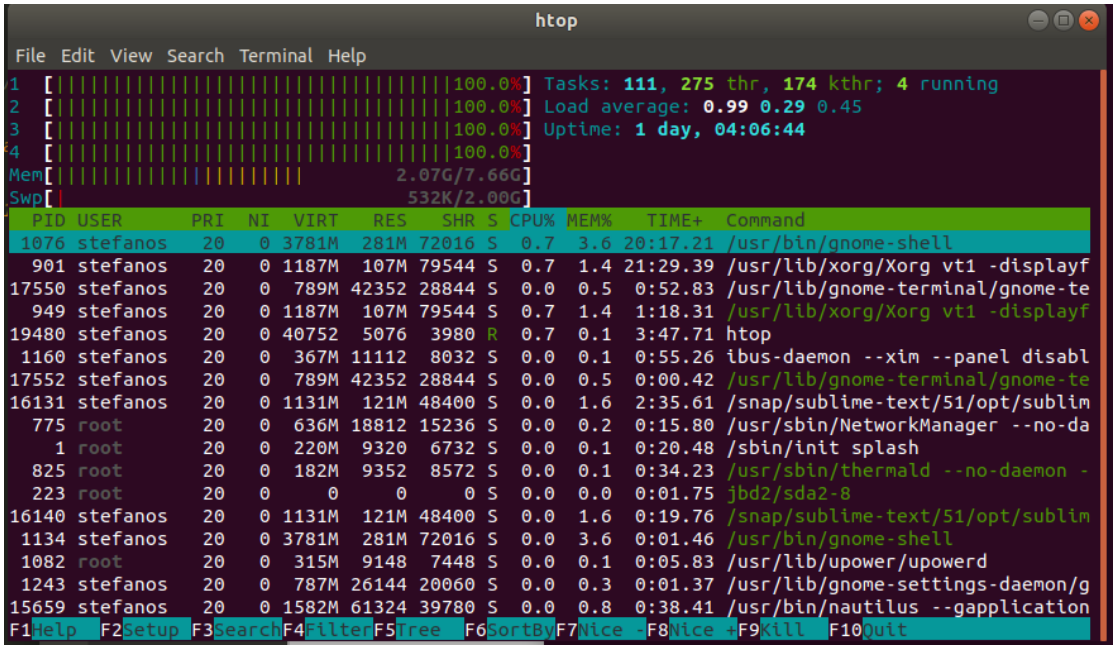
Η πρώτη υλοποίηση με ελπιδοφόρα αποτελέσματα ήταν η προσπάθεια παραλληλοποίησης ολόκληρου του αλγόριθμου. Σε αυτή την προσέγγιση η create_thread δημιουργούσε τα threads σε μια συνάρτηση που περιείχε ολόκληρο τον γενικό αλγόριθμο. Καταμερισμός εργασιών πραγματοποιείται στην εσωτερική επανάληψη εκεί που το μήκος της αντιδιαγωνίου διαιρείται με τον αριθμό τον threads και μοιράζονται τα tasks ανάλογα με το id του κάθε thread .Η αδυναμία της συγκεκριμένης υλοποίησης είναι ότι διεργασίες σε κάποιες περιπτώσεις επαναλαμβάνονται πολλές φορές και πραγματοποιούν πολλούς περιττούς υπολογισμούς ικανούς να ρίξουν την απόδοση.

Ενδεικτικά παραθέτουμε χρόνους εκτέλεσης σειριακού κώδικα και της συγκεκριμένης υλοποίησης

	Serial	Thread=1	Thread=2	Thread=4	Thread=6
D4	0.0004s	0.0006s	0.001s	0.001s	0.004s
D5	0.002s	0.003s	0.006s	0.07s	0.01s
D6	0.05s	0.08s	0.08s	0.09s	0.12s
D7	8.5s	21.6s	14.3s	11.7s	11.53s
D8	135.2s	189s	180.2s	173s	169.79s
D9	237.6s	347,3s	365.6s	456.2s	581.33s

Η εκτέλεση του συγκεκριμένου τρόπου προσέγγισης ενώ όπως μπορούμε να δούμε από το task manager του linux πραγματοποιεί τις λειτουργίες παράλληλα, βλέπουμε ότι στην πράξη ότι τα αποτελέσματα που προκύπτουν δεν είναι επιθυμητά. Συγκεκριμένα βλέπουμε ότι σε καμία από

τις περιπτώσεις παραλληλοποίησης ο αλγόριθμος δεν βελτιώνει την απόδοση με το speedup να παραμένει κάτω από το 1, πράγμα που δείχνει κακή παραλληλοποίηση.



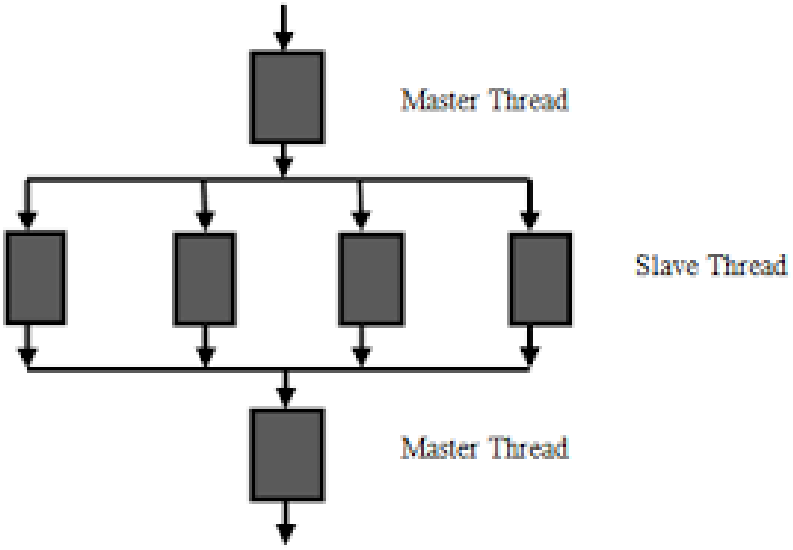
Εικόνα 4:Στιγμιοτυπο htop για 4 threads

Έχοντας επίγνωση του format των αρχείων του Dataset γνωρίζουμε ότι τα D7, D8 έχουν τα μεγαλύτερα tasks και θα περιμέναμε να ανταποκρίνεται καλύτερα σε παραλληλοποίηση fine grained.

Από άποψη συμπεριφοράς και αποτελεσμάτων, μπορεί να χαρακτηριστεί λογική η διαφορά που παρατηρούμε μεταξύ Serial και thread = 1 καθώς υπάρχει overhead. Μεταξύ των thread παρά το γεγονός ότι δεν βελτιώνουν την ταχύτητα του serial παρατηρείται μια φυσιολογική μείωση στην ταχύτητα εκτέλεσης. Να σημειώσουμε ότι η μείωση αυτή του χρόνου εκτέλεσης δεν συνεχίζεται για άπειρο αριθμό από threads καθώς από ένα σημείο και μετά η απόδοση πέφτει δραματικά. Αυτό οφείλεται στο γεγονός ότι περισσότερα νήματα απαιτούν περισσότερο communication μεταβάλλοντας το communication to computation ratio και εισάγοντας περισσότερα overheads.

Παραλληλοποίηση μέρος αλγορίθμου με χρήση spinlock

Κατά την συγκεκριμένη υλοποίηση ο αλγόριθμος δυναμικού προγραμματισμού παραμένει ίδιος και αλλάζει η προσέγγιση ως προς τις διεργασίες που εκτελούμε παράλληλα.



Εικόνα 5:Απεικόνιση μεθόδου master-slave

Στην αρχή του προγράμματος μας, δημιουργούμε όλα τα threads που μας έχει δώσει σαν είσοδο ο χρήστης. Τα thread μας λειτουργούν με την μέθοδο master-slave όπως βλέπουμε στην παραπάνω εικόνα και αφού δημιουργηθούν slave threads, με την τεχνική busy-wait πραγματοποιούν spinlock μέχρι να τα «ξεκολλήσει» το master thread. Το master thread συνεχίζει κανονικά με σειριακό τρόπο την εκτέλεση του κώδικα.

Μόλις φτάσουμε στο σημείο εκτέλεσης του αλγορίθμου υπάρχει διεργασία που εισάγεται το master thread ξυπνώντας όλα τα υπόλοιπα για να υλοποιήσουν τις ζητούμενες διεργασίες παράλληλα. Σε αυτή την περίπτωση έχουμε παράλληλη υλοποίηση μόνο για το μήκος κάθε αντιδιαγωνίου καταφέρνοντας να απαλείψουμε τους παραπάνω υπολογισμούς που βάραιναν τα threads με την προηγούμενη υλοποίηση.

Μετά την ολοκλήρωση όλων των διεργασιών, τα slave threads πραγματοποιούν join με το master thread και τελειώνει η εκτέλεση του προγράμματος.

Δυστυχώς, όμως ούτε αυτή η υλοποίηση βελτίωσε σε ικανοποιητικό βαθμό το χρόνο εκτέλεσης καθώς βελτιώθηκε ο χρόνος εκτέλεσης των D7,D8 με το speedup να ξεπερνάει την τιμή 1 αλλά ο χρόνος εκτέλεσης των D9,D10 καθυστερούσε σε πολύ μεγάλο βαθμό πράγμα που καθιστούσε το τρέξιμο τέτοιων αρχείων πολύ χρονοβόρα διαδικασία. Για αυτό τον λόγο ως τελικό παραδοτέο προτιμήθηκε η προηγούμενη υλοποίηση που έχει καλύτερο μέσο χρόνο εκτέλεσης. Για λεπτομέρειες ή και για κώδικα αναφορικά με την συγκεκριμένη υλοποίηση ελάτε σε επαφή μαζί μας

	Serial	Thread=1	Thread=2	Thread=4	Thread=6
D4	0.0004s	0.0007s	0.001s	0.001s	0.004s
D5	0.002s	0.003s	0.006s	0.07s	0.01s
D6	0.05s	0.09s	0.07s	0.06s	0.12s
D7	8.5s	14.6s	9.3s	7.3s	6.9s
D8	135.2s	156.3s	149.3s	125.4s	132.4s

OpenMp

Στην υλοποίηση με OpenMp τα πράγματα είναι αρκετά πιο απλά καθώς απλά κληθήκαμε να δώσουμε το task που επιθυμούμε με το OpenMp να είναι υπεύθυνο για τον καταμερισμό εργασιών. Το κομμάτι κώδικα που υλοποιήσαμε παράλληλα είναι η εσωτερική επανάληψη, με τα task κάθε φορά να διαμοιράζονται κατά μήκος της αντιδιαγωνίου. Έτσι με την εντολή #pragma omp for parallel κάναμε παράλληλη την λειτουργία του συγκεκριμένου loop ενώ με την παράμετρο num_threads μπορέσαμε να θέσουμε τον αριθμό των threads ανάλογα με την προτίμηση του χρήστη. Από τις διάφορες παραμέτρους χρησιμοποιήσαμε την εντολή schedule και θέσαμε την τιμή static για να γίνεται ο διαμοιρασμός των tasks ανά σταθερά κομμάτια πράγμα που βελτίωνε την απόδοση αρκετά περισσότερο από τις υπόλοιπες υλοποιήσεις auto, dynamic, guided ειδικά από την στιγμή που είμαστε υποχρεωμένοι να εκτελούμε μια αντιδιαγώνιο την φορά.

***Παρατήρηση**: Να επισημάνουμε ότι οι χρόνοι εκτέλεσης δεν είναι όπως θα περιμέναμε και είναι ίδιοι και σε πολλές περιπτώσεις χειρότεροι από τον σειριακό κώδικα. Αυτό πιθανόν οφείλεται σε μεγάλο βαθμό στο False Sharing που συναντάται καθώς ο βασικός πίνακάς μας που ενημερώνεται σε κάθε επανάληψη αποτελεί μια shared μεταβλητή. Λόγω έλλειψης χρόνου δεν καταφέραμε να διορθώσουμε το πρόβλημα επιτυγχάνοντας την επιθυμητή βελτίωση στην απόδοση του κώδικα*

	Serial	Thread=1	Thread=2	Thread=4	Thread=6
D4	0.0004s	0.0004s	0.0006s	0.0008s	0.006s
D5	0.002s	0.003s	0.002s	0.003s	0.02s
D6	0.05s	0.07s	0.05s	0.04s	0.25s
D7	8.5s	19.3s	14.3s	8.2s	11.4s
D8	135.2s	156.5s	137.3s	127.2s	178s

Ειδικά στην περίπτωση του OpenMp θα περιμέναμε βέλτιστες εκτελέσεις με βάση τους πόρους που διατίθενται καθώς το σύστημα πραγματοποιεί τον διαμοιρασμό των tasks στα threads. Και πάλι όμως βλέπουμε ότι η υλοποίηση μας είναι «προβληματική» χωρίς να προσδίδει τα επιθυμητά αποτελέσματα και speedup. Οι περιπτώσεις και πάλι που υπάρχει βελτίωση σχετικά

με την σειριακή υλοποίηση είναι ελάχιστες και φαίνονται στον παραπάνω πίνακα. Βέβαια έχοντας υπόψιν ως ανασταλτικό παράγοντα στην απόδοση την παρατήρηση που προηγήθηκε. Οι παρατηρήσεις για τον εκάστοτε πίνακα είναι σε μεγάλο βαθμό ίδιες με τους προηγούμενους.

Παράρτημα

Εκτέλεση

Όπως ζητήθηκε και από την εκφώνηση η εκτέλεση των παραπάνω υλοποιήσεων έπρεπε να γίνεται από bash αρχείο για Linux-Based σύστημα. Ακολουθώντας την οδηγία, δημιουργήσαμε ένα run.sh αρχείο το οποίο είναι υπεύθυνο για την εκτέλεση οποιασδήποτε υλοποίησης ή και πολλαπλών υλοποιήσεων.

Το bash αυτό αρχείο όπως ζητήθηκε δεν δέχεται ορίσματα και όλες οι αλλαγές που αφορούν τις εισόδους στην εκάστοτε υλοποίηση πρέπει να γίνονται χειροκίνητα από τον χρήστη. Το bash αρχείο αποτελεί επι της ουσίας εκτελέσιμο command line και συνεπώς οι κανόνες και οι συμβάσεις που ορίστηκαν από την εκφώνηση ισχύουν και εδώ.

Τα παραγόμενα αρχεία μετά την εκτέλεση των υλοποιήσεων εμφανίζονται στον φάκελο που βρίσκεται το run.sh αρχείο μας με το αντίστοιχο όνομα που έχουμε δώσει ως όρισμα. Ενδεικτικά, έχουμε θέσει ως default αρχείο το D6.txt από τα Datasets που θα εκτελεστεί σε περίπτωση που κάποιος τρέξει τον αλγόριθμο δίχως αλλαγές.