Παράλληλα και Διανεμημένα Συστήματα

Μάριος Πάκας
mariospakas@ece.auth.gr
9498

Αικατερίνη Πρόκου
aaprokou@ece.auth.gr
9476

https://github.com/Mavioux/Parallel-And-Distributed-Systems-Exercise-2

Εργασία 2

Ανάλυση αλγορίθμου V0

Η σειριακή αναζήτηση των k κοντινότερων γειτόνων απαιτούσε την απλή εφαρμογή του τύπου που δινόταν. Μέσω της συνάρτησης cblas_dgemm έγινε ο πολλαπλασιασμός των X, Y και με απλές επαναλήψεις στα περιεχόμενα των X, Y υπολογίστηκε ο τελικός πίνακας D. Στη συνέχεια, εξίσου σημαντικός ήταν και ο κώδικας αναζήτησης των k κοντινότερων γειτόνων με αναζήτηση στον πίνακα D και θέτοντας ίσες με άπειρο τις τιμές που ήδη επιλέχθηκαν. Με τον τρόπο αυτόν επιστρέφεται το τελικό knnresult struct με τους πίνακες των κοντινότερων γειτόνων και κοντινότερων αποστάσεων για κάθε στοιχείο του Y με βάση τον αρχικό πίνακα X.

Ανάλυση αλγορίθμου V1

Σε αυτήν την έκδοση του αλγορίθμου ο υπολογισμός του τελικού knnresult γινόταν συγχρόνως με χρήση πολλών επεξεργαστών, διαμοιράζοντας ουσιαστικά το πρόβλημα σε p υποπροβλήματα. Αρχικά, ο πίνακας των X ήταν γνωστός μόνο στο μηδενικό επεξεργαστή και διαμοίραζε υποπίνακες του X σε κάθε επεξεργαστή. Αυτό επετεύχθη μέσω ενός for loop που διαμοίραζε το κατάλληλο σημείο (Pointer) του αρχικού πίνακα και το εύρος τιμών, προκειμένου ο κάθε επεξεργαστής να γνωρίζει τα στοιχεία του αρχικού προβλήματος τα οποία του αναλογούν, μέσω των εντολών MPI_Send και MPI_Recv. Στη συνέχεια, θεωρήθηκε αποδοτικότερη η διαμοίραση, μέσω μιας δομής δαχτυλιδιού, των υποπινάκων μεταξύ των επεξεργαστών, προκειμένου να γίνεται η αναζήτηση των k κοντινότερων γειτόνων (μέσω της προηγούμενη συνάρτησης knn του v0) και στο τέλος κάθε επεξεργαστής να έχει υπολογίσει το δικό του knnresult για τον αρχικό του υποπίνακα. Δηλαδή γινόταν διαμοιρασμός των πινάκων που θεωρούνται ως X ενώ το Y ήταν πάντα σταθερό για κάθε διεργασία. Μόλις τελείωνε αυτή η διαδικασία γινόταν εκπομπή του κάθε υποπίνακα στην αρχική διεργασία προκειμένου να συγκεντρωθεί το αποτέλεσμα ολοκληρωμένο.

Ανάλυση αλγορίθμου V2

Σε συνέχεια της προηγούμενης διαδικασίας και σύμφωνα με την εξαιρετική ιδέα των vptrees, ο παραπάνω κώδικας μπορούσε να γίνει ακόμα ταχύτερος, αν δημιουργούταν ένα δέντρο αναζήτησης γειτόνων. Έτσι, αφότου κάθε επεξεργαστής δέχεται τον δικό του υποπίνακα X, καλείται η συνάρτηση vpt_create η οποία δημιουργεί το vptree που αντιστοιχεί στα συγκεκριμένα στοιχεία X και είναι ξεχωριστό και σταθερό για κάθε επεξεργαστή. Στη συνέχεια με την ίδια δομή δαχτυλιδιού εναλλάσσονται τα δεδομένα (σημείωση: αυτήν την φορά πέρα από τα δεδομένα πασάρονται και οι πίνακες του knnresult, προκειμένου να διατηρείται η πληροφορία των κοντινότερων γειτόνων) και στη συνέχεια γίνεται η ανανέωση των κοντινότερων γειτόνων μέσω της συνάρτησης search_vpt για το συγκεκριμένο υποσύνολο στοιχείων του, με βάση το vptree εκείνου του επεξεργαστή (το οποίο είναι σταθερό για κάθε κόμβο). Στο τέλος, κάθε στοιχείο του υποπίνακα θα έχει συγκριθεί με όλα τα vptrees και θα έχει κρατήσει τις κοντινότερες αποστάσεις, γλιτώνοντας πολλούς αχρείαστους υπολογισμούς αποστάσεων!

Διαγράμματα

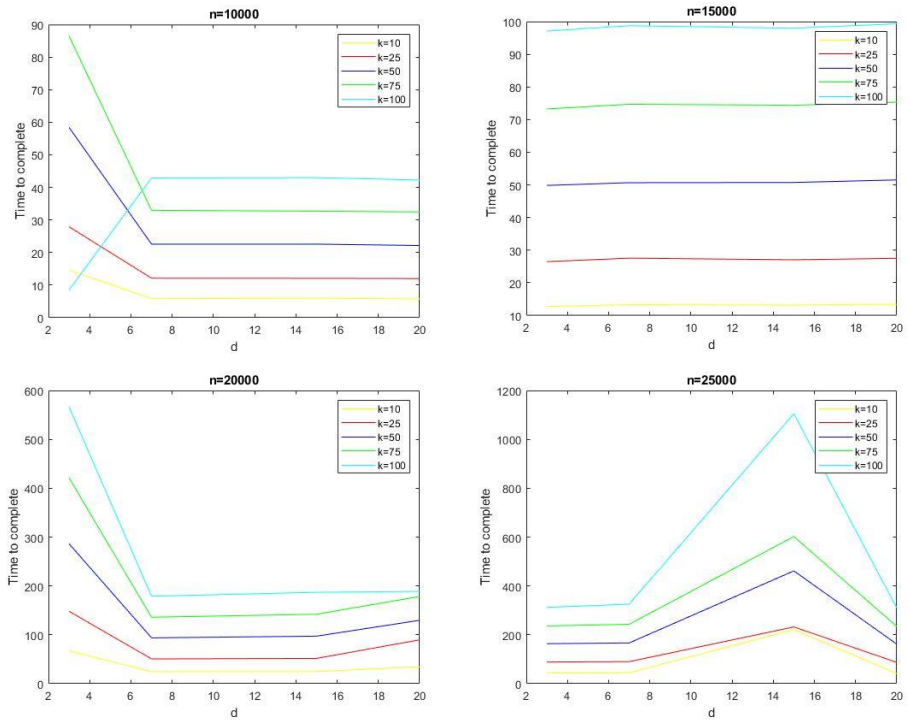
n: Πλήθος Σημείων

k: Αριθμός κοντινότερων γειτόνων

d: Αριθμός Διαστάσεων

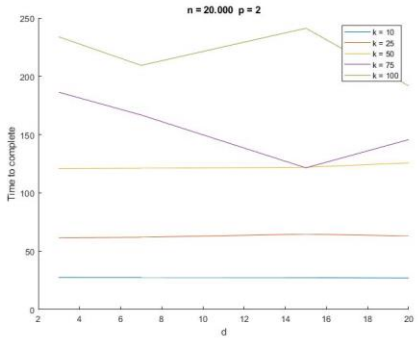
p: Αριθμός Επεξεργαστών

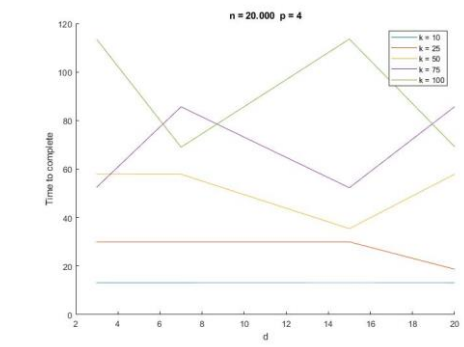
V0

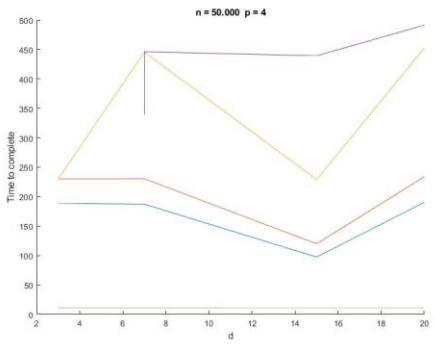


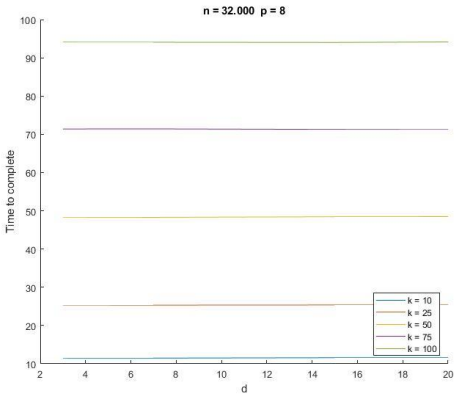
Από τα διαγράμματα του V0 παρατηρούμε πως το k είναι ο κύριος παράγοντας που επηρεάζει αναλογικά τους χρόνους εκτέλεσης της συγκεκριμένης υλοποίησης, δηλαδή ο αριθμών των κοντινότερων γειτόνων που αναζητούνται, και όχι ο αριθμός των διαστάσεων d του χώρου.

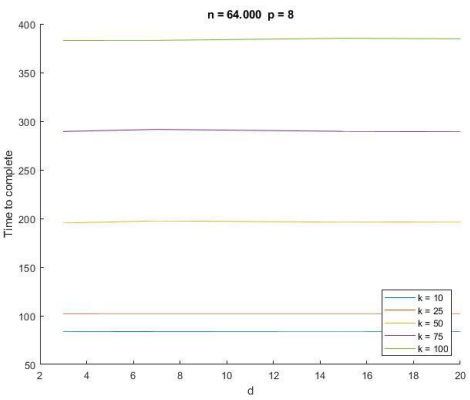
V1

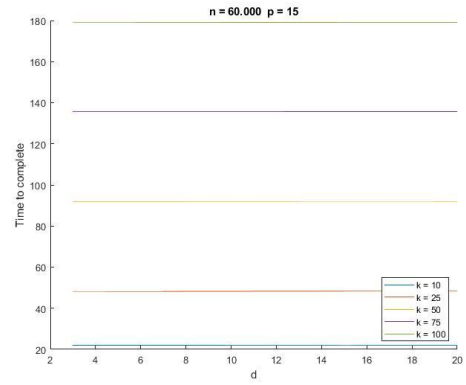


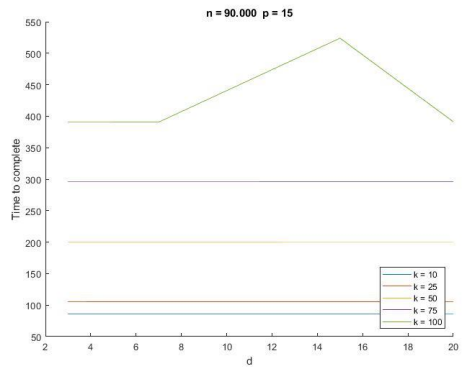










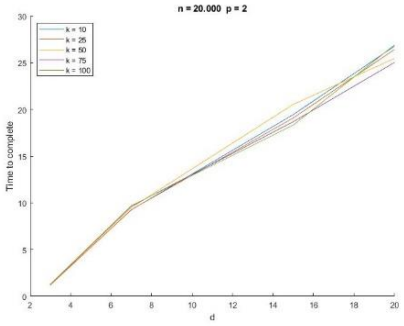


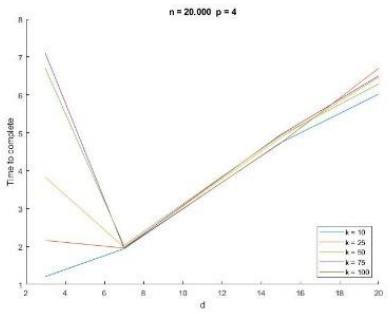
Από τα παραπάνω διαγράμματα διαπιστώνουμε πως και για το V1, ανεξαρτήτως του αριθμού των επεξεργαστών που χρησιμοποιούνται στον υπολογισμό, τη μεγαλύτερη επίπτωση (αν όχι και τη μοναδική) στο χρόνο εκτέλεσης έχει ο παράγοντας k, όπως παρατηρείται και στο V0.

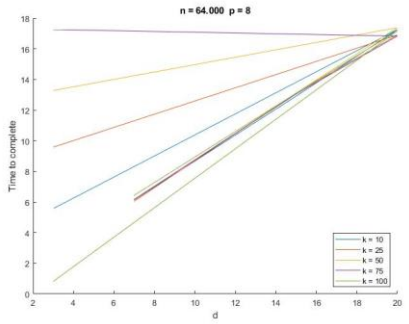
Επίσης, καταλήγουμε στα συμπεράσματα πως για ίσο αριθμό στοιχείων n αλλά για αυξημένο αριθμό επεξεργαστών, υπάρχει μείωση στον χρόνο, αλλά για το αντίθετο, δηλαδή σταθερό αριθμό επεξεργαστών και αυξημένο n, παρατηρείται αύξηση στον χρόνο εκτέλεσης.

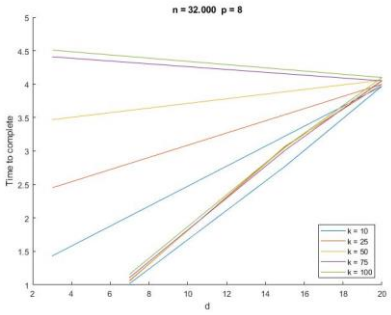
Κάνοντας, τέλος, και την μονή δυνατή σύγκριση του V1 με το V0 για n=20000, παρατηρούμε πως το V1 εμφανίζει αρκετά μικρότερους χρόνους.

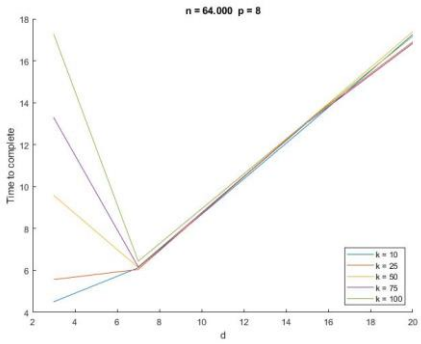
V2

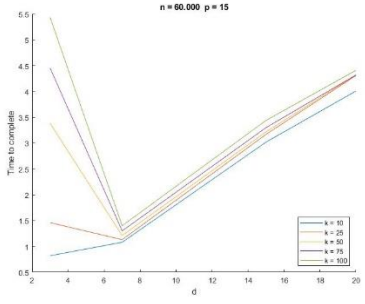


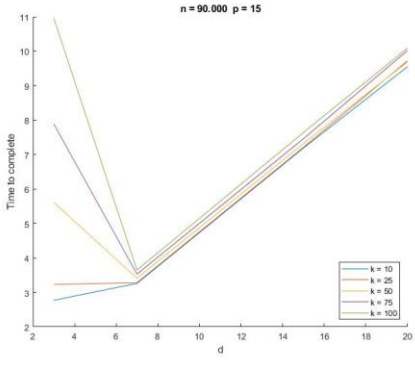












Από τα παραπάνω διαγράμματα βλέπουμε αρχικά πως για τα ίδια ακριβώς πειράματα με το V2 έχουμε πολύ χαμηλότερους χρόνους σε σχέση με την V1 υλοποίηση!

Ακόμη παρατηρούμε ότι η επίδραση του παράγοντα k (ιδίως για d>7) αρχίζει και γίνεται αμελητέα.