Parallel & Distributed Systems

Υλοποίηση Vantage-Point Tree και αναζήτηση k-Nearest Neighbour.

Εισαγωγή

Στην εργασία υλοποιούμε έναν αλγόριθμο παράλληλης ταξινόμησης δεδομένων, συνθέτοντας ένα **Vantage-Point tree**. Πιο συγκεκριμένα, γίνεται διαχωρισμός των δεδομένων σύμφωνα με ένα μεταβαλλόμενο αριθμό μέσης απόστασης. Στη συνέχεια, εφαρμόζουμε αντίστοιχα πάνω στο δέντρο τον αλγόριθμο Μηχανικής Μάθησης k-Nearest Neighbours (KNN). Στόχος είναι να διαχειριστούμε μεγάλα datasets όπου με την χρήση ενός υπολογιστή - επεξεργαστή δεν θα μπορούσαμε.

Αλγόριθμος

VP Tree

Το πρώτο βήμα είναι η υλοποίηση ενός σειριακού κώδικα που κατασκευάζει ένα vantage-point tree. Σχηματικά μπορούμε να αναλύσουμε την διαδικασία στα εξής βήματα:

1. Επιλέγεται ένα τυχαίο σημείο ως vantage-point από το αρχικό dataset που αποτελεί και τον μηδενικό κόμβο.
2. Υπολογίζονται οι αποστάσεις όλων των υπόλοιπων σημείων από το vantage-point.
3. Υπολογίζεται η μέση απόσταση.
4. Τα σημεία με απόσταση μικρότερη από την μέση απόσταση σχηματίζουν το dataset του αριστερού κόμβου-παιδιού και αντίστοιχα τα σημεία με απόσταση μεγαλύτερη της μέσης σχηματίζουν του δεξιά.
5. Επαναλαμβάνουμε για κάθε κόμβο.

Έπειτα προσπαθούμε να επιταχύνουμε τον αλγόριθμο με την βοήθεια των threads. Αρχικά, εφαρμόζοντας τα σε όλο το μήκος της κατασκευής του δένδρου και στην συνέχεια χρησιμοποιώντας τα μόνο για κόμβους με dataset.size > threshold. Τέλος, επιστρέφουμε στην σειριακή υλοποίηση.

KNN

Αφού σχηματιστεί το δέντρο εκτελούμε αναζήτησή των k κοντινότερων γειτόνων κάθε σημείου για $k = 2^x, x = [1:8]$. Η αναζήτησή αυτή πραγματοποιείται τόσο σειριακά όσο και με την βοήθεια της MPI βιβλιοθήκης.

Χρησιμοποιούμε ένα flag ώστε να κατανέμουμε τα σημεία με αποστάσεις ίσες με την median ισομερώς στους πίνακες inner και outer.

```
else if ((distances[i] == root->median_distance) && flag == 0) {
  ...
  inner++;
  flag = 1;
} else if ((distances[i] == root->median_distance) && flag == 1) {
  ...
  outer++;
  flag = 0;
}
```

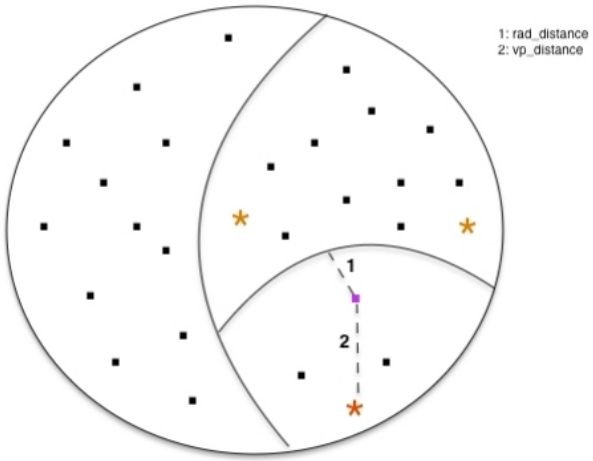
Στην **knn_search** διατρέχουμε το δέντρο με τον εξής αλγόριθμο. Θέλουμε να ξεκινήσουμε την αναζήτηση των γειτόνων με την υπόθεση πως αυτοί βρίσκονται στον κόμβο όπου

1. υπάρχει το σημείο για το οποίο αναζητούμε τους γείτονες

2. το datasize είναι μικρότερο δυνατό αλλά μεγαλύτερο του k+1 ώστε να μπορεί να περιέχει k γείτονες

Έτσι κατεβαίνουμε στο δέντρο έως ότου datasize<k+1 και γυρνάμε ένα επίπεδο προς τα πίσω.

```
if (k + 1 > current_node->data_size) {
current_node = current_node->prev;
break;
}
```

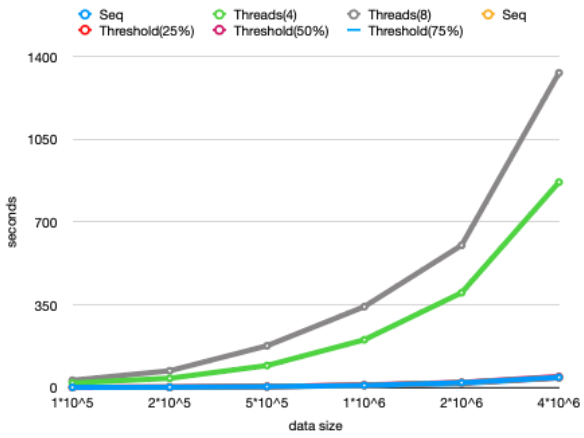


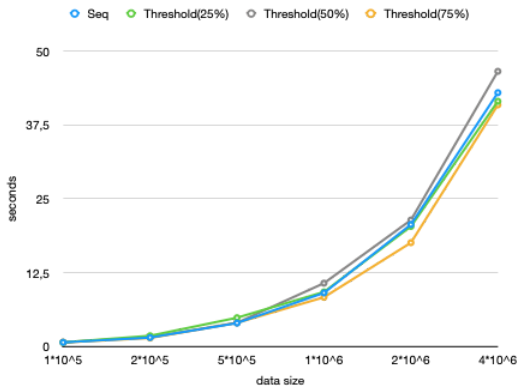

```
// αν το max_distance είναι μικρότερο του rad_distance,
// τότε ψάχνουμε στην άλλη μερια του δέντρου
rad_distance = median_distance - vp_distance;
if (rad_distance < max_distance) {
...
}
```

Αποτελέσματα

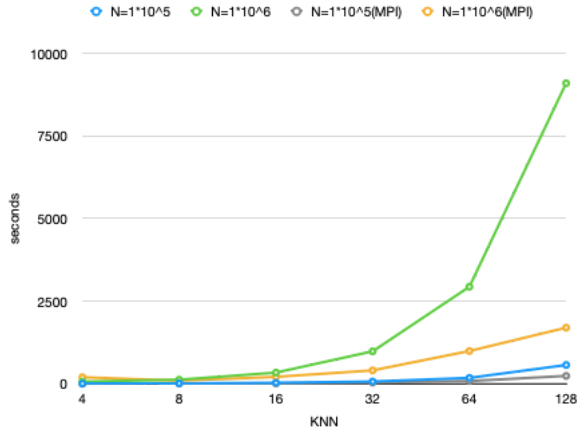
Για dimensions = 3 και num_procs = 4, παρήχθησαν τα εξής αποτελέσματα:

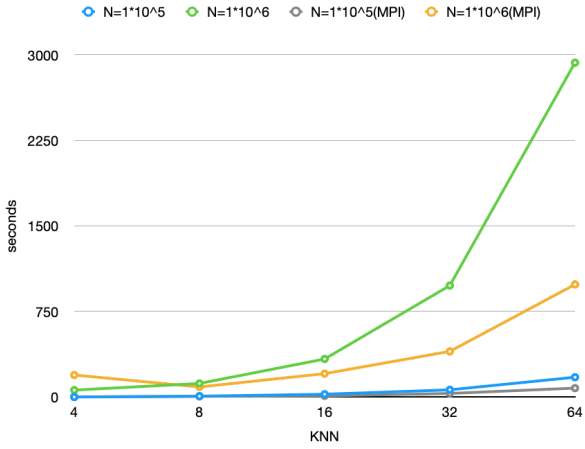
	1*10^5	2*10^5	5*10^5	1*10^6	2*10^6	4*10^6
Seq	0,688720	1,492348	3,955073	9,054885	20,618696	42,899008
Threads (4)	19,423661	39,174297	92,851936	201,770035	400,763588	867,458813
Threads (8)	30,561032	70,722346	176,352843	341,677861	601,144278	1329,380631
Threshold(75%)	0,681271	1,428633	3,663749	8,318164	17,544370	40,836730
Threshold(50%)	0.723683	1,549076	4,404194	10,699099	21,375365	46,510219
Threshold(25%)	0,684991	1,804935	4,858306	9,214206	20,292496	41,446454





N\k	2	4	8	16	32	64	128
1*10^5	0,649306	0,841436	8,242847	25,429633	63,941778	174,426291	563,833
1*10^6	27,622519	61,460479	119,665870	334,189812	977,888228	2931,034052	8086,03
1*10^5(MPI)	0,958749	2,016246	6,796463	10,834289	32,318613	78,955931	234,582
1*10^6(MPI)	83,963330	193,972952	89,412514	205,597967	400,828788	987,001932	1693,39





(4):

```
K(8)NN 1000000 elements in 3 dimensions and 1999999 nodes_created in 119.665870 seconds

knn data search for: 68.132653 5.503321 94.310947
68.477715 4.961903 94.699413
68.215959 5.628259 94.149298
68.162360 4.783417 94.890425
68.043384 5.152332 95.250076
67.317527 5.681854 94.915094
67.662640 5.991162 93.461131
68.691943 5.485428 93.596013
67.977914 4.795337 95.234521
```

Παρατηρήσεις

1. Η χρήση των threads σε όλο το μήκος της κατασκευής του δέντρου επιβραδύνει την διαδικασία σε σημαντικό βαθμό. Αυτό οφείλεται ότι στα τελευταία φύλλα του δέντρου όπου τα μεγέθη των datasets είναι της τάξης των 2, 3 κτλ., καλούμε αρκετά threads για ελάχιστους υπολογισμούς (υπάρχουν threads ακόμα και με μηδενικό φόρτο εργασίας).
2. Η κατάσταση διαφοροποιείται σημαντικά όταν επιστρατεύουμε το threshold αφού αποτρέπει την καθυστέρηση που περιγράφεται στο 1.
3. Σημειώνεται πως υπάρχει διαφορά στην απόδοση του κώδικα για διαφορετικά ποσοστά threshold (πχ. threshold(25%)= data.size/4).
   Συγκεκριμένα βλέπουμε πως για 75% η απόδοση ξεπερνά ακόμα και το sequential που ως τώρα ήταν το ταχύτερο.
4. Επιπρόσθετα, η εύρεση των γειτόνων είναι εξαιρετικά ακριβής αφού οι αποστάσεις τους από το κεντρικό σημείο είναι ελάχιστες.
5. Επειδή υπολογίζουμε τους k κοντινότερους γείτονες για κάθε σημείο εισάγουμε την βιβλιοθήκη MPI, όπου μας βοηθά να μοιράσουμε τον φόρτο εργασίας και τελικά να παράξουμε τον ζητούμενο 3D πίνακα σε πολύ πιο σύντομο διάστημα. Παρατηρούμε, ότι για όσο μεγαλώνει το k, και άρα οι κόμβοι που πρέπει να εξεταστούν, τόσο η MPI βελτιώνει την απόδοση της αναζήτησης. Ενώ αντίθετα για μικρά k, το κόστος επικοινωνίας μεταξύ των πυρήνων επικαλύπτει την επιτάχυνση στον υπολογισμό εύρεσης των γειτόνων.

assignment link:https://github.com/georgegoun/Parallel_Distributed_Systems_4

Γεώργιος Γούναρης (9980) - ggounaris@ece.auth.gr
Νικοπολοτίδης Εμμανουήλ - Γεώργιος (9474) - nikopole@ece.auth.gr
ΤΗΜΜΥ ΑΠΘ