Εργασία 2
Παράλληλα και Διανεμημένα Συστήματα .
Υλοποίηση ενός αλγορίθμου για K-NN Search και χρησιμοποίηση του για υλοποίηση σύγχρονου και ασύγχρονου διανεμημένου K-NN Search με MPI .

Μιχάλης Καρατζάς
AEM:9137
email:mikalaki@ece.auth.gr
mikalaki@it.auth.gr

Μανώλης Μιχάλαινας
AEM:9070
email: michalain@ece.auth.gr

Ομάδα στο elearning: Ομάδα Μ

Github link για το project :https://github.com/mikalaki/knnring
Dropbox link :https://www.dropbox.com/s/bl98q41ko1j9qmi/code.tar.gz?dl=0
(Στον έλεγχο ορθότητας τοπικά παίρναμε Correct τόσο για sequential όσο και για MPI, στον online valiator , στην MPI έκδοση , λαμβάναμε την ένδειξη **Time limit exceeded** (με αρχείο του dropbox).)

1. Σύντομη Περιγραφή της υλοποίησης μας.
   - Sequential: Στην Ακολουθιακή έκδοση της C, για τον υπολογισμό των αποστάσεων ακολουθήσαμε τον κώδικα που μας δόθηκε σε matlab. Χρησιμοποιούμε την cblas_dgemm() της βιβλιοθήκης openblas, για τον υπολογισμό του πίνακα *-2 * X*Y.'* , στην συνέχεια στον πίνακα που προκύπτει προσθέτουμε σε κάθε στήλη το άθροισμα των συντεταγμένων κάθε στοιχείου του X *(sum(X.^2,2))* στην αντίστοιχη γραμμή (πχ. 5ο σημείο , 5η γραμμή) και σε κάθε γραμμή το άθροισμα των συντεταγμένων κάθε στοιχείου του Y στη αντίστοιχη στήλη (5ο σημείο, 5η στήλη) με κατάλληλες επαναλήψεις for. Έπειτα βάζοντας ρίζα σε κάθε στοιχείο του πίνακα , προκύπτει ο ζητούμενος πίνακας των αποστάσεων D. Έπειτα για την επιλογή των K-nn , χρησιμοποιείται ο αλγόριθμος quicksort, μόνο για τα K μικρότερα στοιχεία των αποστάσεων (καθώς θα ήταν περιττό να ταξινομούνταν και τα υπόλοιπα σημεία) .
   - Synchronous_MPI: Για την σύγχρονη υλοποίηση του διανεμημένου αλγόριθμου K-nn, χρησιμοποιήθηκε η μέθοδος MPI_Sendrecv_replace() , η οποία χρησιμοποιεί μόνο ένα buffer για τις διαδικασίες της αποστολής και της λήξης των αποτελεσμάτων και προφανώς μπλοκάρει το mpi process για όσο χρόνο εκτελείται μία από τις παραπάνω διαδικασίες.
   - Asynchronous_MPI: Για την ασύγχρονη εκδοχή χρησιμοποιήθηκαν οι μέθοδοι MPI_Isend() και MPI_Irecv() οι οποίες δεν μπλοκάρουν το mpi_process , κατά την διάρκεια αποστολής και λήψης των δεδομένων , επιτρέποντας μας κατά αυτόν τον τρόπο να κάνουμε υπολογισμούς κατά την διάρκεια των επικοινωνιών. Επίσης χρησιμοποιώντας την μέθοδο MPI_Waitall() , εξασφαλίζουμε ότι έχουν ολοκληρωθεί οι διαδικασίες αποστολής και λήξης ενός MPI procces πριν εκκινήσουν οι επόμενες.
     Και στις δύο υλοποιήσεις του MPI ,στο τέλος κάθε ελέγχου Knn , προστίθεται το κατάλληλο offset για τα indices , ανάλογα με ποιο τμήμα του ολικού corpus βρίσκεται το επιμέρους corpus του ελέγχου ,επίσης συγχωνεύονται τα αποτελέσματα , ώστε κατά την έξοδο του προγράμματος να έχουμε σωστά αποτελέσματα. Η επιλογή αν θα τρέξει το synchronous ή το asynchronous MPI , γίνεται μέσα από το makefile μας.
2. Παρουσίαση και σχολιασμός των χρόνων εκτέλεσης :
   I. Τοπικά(σε προσωπικό υπολογιστή)*:

Πίνακες για σταθερό αριθμό σημείων(n) και μεταβαλλόμενο αριθμό διαστάσεων:

n=2000	Sequential	Synchronous MPI	Asychronous MPI
d=10	0.338868 seconds	0.300709 seconds	0.266735 seconds
d=40	0.320033 seconds	0.283771 seconds	0.290694 seconds
d=90	0.336810 seconds	0.312864 seconds	0.315664 seconds
d=160	0.377661 seconds	0.338829 seconds	0.355411 seconds!

n=4000	Sequential	Synchronous MPI	Asychronous MPI
d=10	1.057831 seconds	0.838186 seconds	0.885294 seconds
d=40	1.304246 seconds	0.826099 seconds	0.891499 seconds
d=90	1.523629 seconds	0.931377 seconds	1.035576 seconds
d=160	1.466136 seconds	1.021771 seconds	1.049231 seconds

n=6000	Sequential	Synchronous MPI	Asychronous MPI
d=10	4.265025 seconds	1.812805 seconds	1.810395 seconds
d=40	3.707226 seconds	1.713628 seconds	1.694620 seconds
d=90	4.273085 seconds	1.827987 seconds	1.896963 seconds
d=160	4.761572 seconds	2.070613 seconds	2.098999 seconds

n=8000	Sequential	Synchronous MPI	Asychronous MPI
d=10	6.533638 seconds	2.448935 seconds	2.560703 seconds
d=40	8.083726 seconds	2.826291 seconds	2.718031 seconds
d=90	9.299672 seconds	3.159724 seconds	3.232844 seconds
d=160	10.440323 seconds	3.372922 seconds	3.447434 seconds

*Οι μετρήσεις οι οποίες οδήγησαν στους παραπάνω πίνακες έγιναν σε ένα λάπτοπ με επεξεργαστή i5-2410m(2 πυρήνες 4 threads) με 4gb ram και 5gb swap και λειτουργικό ubuntu linux 18.10. **Για να καταφέρουμε ίδιο αριθμό σημείων στο cluster , στο sequential με το mpi, στον κώδικα του sequential πολλαπλασιάζουμε το n με τον συνολικό αριθμό των mpi processes που δημιουργούνται στην αντίστοιχη mpi έκδοση, καθώς έτσι προκύπτει ο αριθμός σημείων του ολικού corpus στην mpi έκδοση από τον tester.

Από τα παραπάνω ,βλέπουμε ότι καθώς ο αριθμός των σημείων και των διαστάσεων αυξάνεται, στις υλοποιήσεις με MPI έχουμε σαφή και αυξανόμενη βελτίωση . Κάτι το οποίο είναι αναμενόμενο καθώς η υλοποίηση με MPI δημιουργεί 4 MPI processes (για εκτέλεση χρησιμοποιείται η εντολή *mpirun -np 4)*, κάθε ένα από τα οποία απασχολεί διαφορετικό thread του επεξεργαστή(καθώς έχουμε 4 thread) γεγονός που επιτρέπει τον παράλληλο υπολογισμό των επιμέρους KNN σε τοπικό επίπεδο μέσω MPI . Τα παραπάνω επιβεβαιώνονται και παρακολουθώντας το System Monitor κατά την διάρκεια εκτέλεσης της MPI υλοποίησης μας , καθώς σε αυτή βλέπουμε ότι αξιοποιούνται πλήρως και τα 4 thread του υπολογιστή. Επίσης το κόστος επικοινωνίας μεταξύ processes που βρίσκονται στον ίδιο υπολογιστή (ίδιο node) , είναι πρακτικά μηδενικό, γεγονός που εξηγεί τους ουσιαστικά ίδιους χρόνους (πάνω κάτω) στο σύγχρονο και στο ασύγχρονο MPI (όταν αυτά εκτελούνται σε έναν μόνο υπολογιστή (node)).

Επισημαίνεται ότι τόσο για τα τοπικά πειράματα (παραπάνω) όσο και για τα πειράματα στην υπολογιστική συστοιχία (παρακάτω) , εκτελείται ο κώδικας του tester , χωρίς τα τμήματα που χρησιμοποιούνται για validation , καθώς αυτό θα επιβράδυνε αρκετά τα προγράμματά μας .

II. Στην υπολογιστική συτοιχία Αριστοτέλης (cluster):

Καθώς δεν έχουμε καταφέρει να εκτελέσουμε ακόμα οργανωμένα πειράματα στην υπολογιστική συστοιχία ,δεν μπορούμε να παραθέσουμε αναλυτικά στοιχεία για τους χρόνους εκτέλεσης, πίνακες και σχόλια όπως παραπάνω .Έχουμε ορίσει κατάλληλα jobs , ωστόσο είμαστε σε ουρά αναμονής, τόσο στο partition pdlabs , με 8 nodes και 4 cpus σε κάθε node (32 processes συνολικά), όσο και στο batch , με 4 nodes και 2 cpus ανά node (8 processes συνολικά). Εκτός από το MPI έχουμε ορίσει να εκτελέσουμε και το sequential σε ένα node της κάθε συστοιχίας για το ίδιο σύνολο διαστάσεων και αριθμών σημείων με τις εκτελέσεις σε mpi , προκειμένου να συγκρίνουμε την βελτιώση που παίρνουμε με το mpi για μεγάλα n και d. (για να καταφέρουμε ιδιο αριθμό σημείων στο cluster , στο sequential με το mpi, στον κώδικα του sequential πολλαπλασιάζουμε το n με τον

συνολικό αριθμό των mpi processes που δημιουργούνται στην αντίστοιχη mpi έκδοση, καθώς έτσι προκύπτει ο αριθμός σημείων του ολικού corpus στην mpi έκδοση από τον τέστερ.)
Τα συμπεράσματα στα οποία αναμένουμε να φτάσουμε από τα παραπάνω πειράματα μας στο cluster, τα οποία επιβεβαιώνονται από κάποιες πρώτες εκτελέσεις που έγιναν στο cluster, είναι ότι 1) το MPI είναι γρηγορότερο του sequential, καθώς αξιοποιεί πολύ περισσότερα resources , ειδικά για μεγάλο αριθμό σημείων και 2)το asynchronous MPI ,είναι γρηγορότερο του synchronous MPI κατά ένα ποσοστό της τάξης του 20%, καθώς στο ασύγχρονο γλιτώνουμε το κόστος επικοινωνίας μεταξύ των processes (που είναι σε διαφορετικά nodes ,καθώς όπως είδαμε και παραπάνω το κόστος επικοινωνίας μεταξύ processes σε ίδιο node είναι αμελητέο) . Τα παραπάνω συμπεράσματα γίνονται προφανέστερα και πιο έντονα καθώς το φορτίο (αριθμός σημείων και διαστάσεων) αυξάνεται.

!(Η παρούσα αναφορά θα ενημερωθεί με μετρήσεις ,αποτελέσματα και ενδεχομένως εκτενέστερο σχολιασμό από την εκτέλεση στην συστοιχία ή θα ανέβουν τα αντίστοιχα αρχεία στο github.)!!