Εργασία 2
Παράλληλα και Διανεμημένα Συστήματα .
Υλοποίηση σύγχρονου και ασύγχρονου διανεμημένου K-NN Search με χρήση MPI .

Μιχάλης Καρατζάς
AEM:9137
email:mikalaki@ece.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 πήραμε Correct τόσο για sequential όσο και για MPI, στον online validator , στην MPI έκδοση , πήραμε την ένδειξη **Time limit exceeded** (με το 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.* Έπειτα για την επιλογή των K-nn , χρησιμοποιείται ο αλγόριθμος quicksort, μόνο για τα K μικρότερα στοιχεία των αποστάσεων.
   - 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.
   - Global reductions: Στο αρχείο knnring_asynchronous.c προστέθηκε και επιπλέον κώδικας που υλοποιεί την εύρεση της ολικής ελάχιστης και μέγιστης απόστασης που παρατηρείται στο αποτέλεσμα knnresult. Αφού υπολογιστούν πλήρως οι κοντινότεροι γείτονες για κάθε σημείο του τοπικού query set, υπολογίζεται το τοπικό ελάχιστο και μέγιστο. Στη συνέχεια, με χρήση MPI_Reduce() και χρήση των έτοιμων macro MPI_MIN και MPI_MAX υπολογίζεται το ολικό ελάχιστο και μέγιστο (από τα τοπικά).

     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=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. Για όλες τις μετρήσεις κ=13.

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 .(Σε όλες τις μετρήσεις κ=13)

II. Στην υπολογιστική συστοιχία Αριστοτέλης (cluster):

Τα πειράματα μας στην υπολογιστική συστοιχία ,έγιναν στο partition batch . Εκτός από τις υλοποιήσεις σε MPI εκτελέσαμε και το sequential σε ένα node του partition για το ίδιο σύνολο διαστάσεων και ίδιο αριθμό σημείων με τις εκτελέσεις σε mpi , προκειμένου να συγκρίνουμε την βελτίωση που παίρνουμε με το MPI για μεγάλα δεδομένα και να αντιληφθούμε το "όφελος" που αυτό μας προσφέρει. (για να καταφέρουμε ίδιο αριθμό σημείων 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.)
Παρακάτω βλέπουμε τα αποτελέσματα που πήραμε από τις μετρήσεις μας:

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

4 Nodes και 2 cpus/node (8 processes συνολικά):

n=4000	Sequential	Synchronous MPI	Asychronous MPI	Βελτίωση Asynchronous %
d=15	0.681039 seconds	0.156034 seconds	0.142868 seconds	8.3%
d=60	0.751897 seconds	0.159034 seconds	0.148978 seconds	6.3%
d=150	0.767722 seconds	0.224800 seconds	0.213818 seconds	4.9%

n=16000	Sequential	Synchronous MPI	Asychronous MPI	Βελτίωση Asynchronous %
d=15	14.414606 seconds	1.928267 seconds	1.918784 seconds	< 1%
d=60	14.365278 seconds	2.632637 seconds	1.887622 seconds	28% (!)
d=150	16.911159 seconds	2.539998 seconds	2.357585 seconds	7.2%

n=40000	Sequential	Synchronous MPI	Asychronous MPI	Βελτίωση Asynchronous %
d=15	exceeded memory limit	15.573206 seconds	14.828984 seconds	4.75%
d=60	exceeded memory limit	20.046657 seconds	17.042604 seconds	15%
d=150	exceeded memory limit	18.656555 seconds	18.061313 seconds	3.2%

n=60000	Sequential	Synchronous MPI	Asychronous MPI	Βελτίωση Asynchronous %
d=15	exceeded memory limit	37.509230 seconds	37.005145 seconds	~1,5%
d=60	exceeded memory limit	46.827865 seconds	42.143116 seconds	10%
d=150	exceeded memory limit	46.306526 seconds	43.589937seconds	~6%

4 Nodes και 8 cpus/node (32 processes συνολικά):

n=60000	Sequential	Synchronous MPI	Asychronous MPI	Βελτίωση Asynchronous %
d=15	exceeded memory limit	10.319166 seconds	9.982980 seconds	3.3%
d=60	exceeded memory limit	12.547321 seconds	11.899392 seconds	5.2%
d=150	exceeded memory limit	12.923196 seconds	12.292978 seconds	4.9%

Τα συμπεράσματα στα οποία φτάνουμε από τα παραπάνω πειράματα μας στο cluster, είναι ότι :

1. Από την χρήση του MPI σε ένα διανεμημένο σύστημα , αποκομίζουμε πολλά οφέλη, καθώς α)βλέπουμε ότι για σχετικά μεγάλα δεδομένα ακόμα και για μόνο 4 nodes κ 2 cpus , ένα διανεμημένο πρόγραμμα είναι έως και 8 φορές γρηγορότερο από ένα σειριακό πρόγραμμα (για n<=16000) , ενώ β)για ακόμα μεγαλύτερα δεδομένα, για τα οποία βλέπουμε ότι η μνήμη (και ενδεχομένως η ισχύς)ενός μεμονωμένου υπολογιστή δεν είναι αρκετή, σε ένα διανεμημένο σύστημα (όπου γίνεται χρήση MPI) , μπορούμε να πάρουμε λύση αρκετά και σε σχετικά μικρό χρόνο. Από τα παραπάνω συνειδητοποιούμε ότι το όφελος και την αναγκαιότητα μια συστοιχίας υπολογιστών οι οποίοι επικοινωνούν(μέσω MPI ή κάποιου άλλου αντίστοιχου λογισμικού ενδεχομένως) για την εκτέλεση ενός "μεγάλου" προγράμματος με στόχο την επίλυση ενός προβλήματος.

2. Συγκρίνοντας τους χρόνους μεταξύ σύγχρονου και ασύγχρονου MPI σε μερικές μετρήσεις βλέπουμε ότι η ασύγχρονη υλοποίηση υπερτερεί ξεκάθαρα σε σχέση με την σύγχρονη , ενώ σε άλλες υλοποιήσεις πάλι η διαφορά φαίνεται να είναι ανύπαρκτη. Βέβαια βλέποντας όλες τις μετρήσεις που πάρθηκαν συνειδητοποιούμε ότι το ασύγχρονο MPI είναι γρηγορότερο και υπάρχει κάποιο όφελος από αυτό το οποίο μπορεί να φτάσει έως και 15% μερικές φορές ,χωρίς ωστόσο να είναι ιδιαίτερα σημαντικό τις περισσότερες φορές, συνήθως λίγο λιγότερο από 10%.

3. Επίσης αυξάνοντας τον αριθμό των συνολικών processes(από 8 → 32) , με το να αυξήσουμε τα cpus/node (2→8) (για 60000 σημεία συνολικά) βλέπουμε ότι η επιτάχυνση στο MPI , είναι σχεδόν ανάλογη της αύξησης των processes. Η βελτίωση οφείλεται στο threading που γίνεται με το MPI σε περισσότερα processes του ίδιου node. Ενώ η βελτίωση του ασύγχρονου MPI σε σχέση με το σύγχρονο, αναμένεται να είναι γενικά ελαφρώς μικρότερη.