Μάθημα : Παράλληλα και Διανεμημένα Συστήματα

Φοιτητής : Χωραφάς Χρήστος 8718

ΑΝΑΦΟΡΑ 2ης ΕΡΓΑΣΙΑΣ

Αναζήτηση και εύρεση k κοντινότερων γειτόνων για κάθε σημείο με χρήση MPI.

Σε κάθε έκδοση του κώδικα :

α) #include τις απαραίτητες βιβλιοθήκες(“mpi.h”, <math.h>, κλπ...)

β) #define τις I(σημεία) και J(συντεταγμένες) διαστάσεις του πίνακα που θα αποθηκεύσω τα σημεία απο το .mat file

γ) declare το πρότυπο της συνάρτησης σύγκρισης της qsort() και της getData().

δ) Δημιουργία μιας struct με μεταβλητές την απόσταση δύο σημείων και την θέση του γείτονα.

ε) Έλεγχος ώστε ο χρήστης να δώσει σωστό k. Αν το k είναι μικρότερο του 1 ή μεγαλύτερο του 59.999, τότε το πρόγραμμα τερματίζει.

ζ) εφαρμογή του KNN.

KNN

Για τον υπολογισμό των γειτόνων ο αλγόριθμος ακολουθεί τα εξής βήματα:

1)Ξεκινά με τον ορισμό ενός pointer-pointer σε struct δεσμεύοντας κατάλληλα μνήμη. Έτσι μπορώ να έχω έναν πίνακα που για κάθε σημείο αποθηκεύει τους k κοντινότερους γείτονες του και την απόστασή του από αυτούς.

2)Καλεί την getData() και αποθηκεύει σε έναν διδιάστατο πίνακα I σημεία με J συντεταγμένες.

3)Υπολογίζει για κάθε σημείο i την απόστασή του με το j σημείο, λαμβάνοντας υπόψιν ότι δεν συγκρίνει ένα σημείο με τον εαυτό του. Χρησιμοποιώντας τον μετρητή *cnt* με αρχική τιμή 1, αποθηκεύει απευθείας την απόσταση στην μεταβλητή *struct array[i][cnt-1].dist* και την θέση του j γείτονα στην μεταβλητή *array[i][cnt-1]*.pos. Όταν ο cnt γίνει ίσος με k, τότε

καλεί την συνάρτηση βιβλιοθήκης **qsort()**, η οποία ταξινομεί τον πίνακα array[i] χρησιμοποιώντας ταξινόμηση κατά αύξουσα σειρά των dist. **Έτσι πετυχαίνω αντιστοίχηση απόστασης με γείτονα στους πίνακες**. Όταν πλέον ο cnt ξεπεράσει το k, απλά ελέγχει τον γείτονα με την μεγαλύτερη απόσταση και αν αυτή είναι μικρότερη από του i με το j (array[i][k-1].dist < distance(i,j)), τότε τον διώχνει (*array[i][k-1].pos = j*) και καλεί εκ νέου την qsort.

- ## **Σεριακή έκδοση**.

Γράφοντας "*gcc -name-.c –lm –O2"* και "*./a.out -k-"* ξεκινάει το πρόγραμμα την υλοποίηση του KNN με ζητούμενο k γείτονες για κάθε σημείο.

Χρόνος: Μετά από δύο runs, με i=60000 σημεία και k=30 γείτονες φαίνεται να τρέχει σε 240 δευτερόλεπτα(πολύ αργό και με optimization).

- ## **MPI non-blocking έκδοση**.

Για αυτή την έκδοση χρειάστηκαν τροποποιήσεις.

i. Αφού ορίσω τις απαραίτητες μεταβλητές που θα χρειαστώ για το MPI(number of tasks,id κλπ...), **ξεκινάω την επικοινωνία με *MPI_INIT()***.

ii. **Ορίζω**: Για την τοπολογία δαχτυλιδιού τις μεταβλητές previous id(*previous*) και next id(*next*). Τον αριθμό των σημείων που θα πάρει κάθε διεργασία να είναι *block = I / numtasks,* ένα πίνακα που θα κρατάει τα αρχικά σημεία της ίδιας της διεργασίας,πλήθους block, και ένα πίνακα που θα κρατάει το απεσταλμένα σημεία απο τις άλλες διεργασίες, πλήθους επίσης *block*. Επιπλέον, αυτή τη φορά ο struct array θα έχει πλήθος [blockXk] για κάθε διεργασία.

iii. **Χρησιμοποιώντας την *MPI_SCATTER()* με τα κατάλληλα ορίσματα**, μοιράζω τον αρχικό πίνακα όλων των σημείων στις *numtasks* διεργασίες ως εξής : η 0 διεργασία θα πάρει τα πρώτα block σημεία, η 1 τα δεύτερα block σημεία κ.ο.κ. Αυτή η σειρά επιτυγχάνεται δίνοντας ως *source* όρισμα στην scatter id=0.

iv. **Τώρα μπορεί η κάθε διεργασία να στείλει τον original buffer(*originbuf*) της στην επόμενη με μια *MPI_Isend()* αλλά και να παραλάβει** τον αντίστοιχο buffer της προηγούμενης, καταχωρόντας τον στον receive buffer(*recvbuf*) και να προχωρήσει στον KNN για το δικό του μπλοκ από *block* σημεία, χωρίς να περιμένει την ολοκλήρωση των επικοινωνιών.

v. **Αφού λοιπόν βρει τους γείτονες για το δικό της μπλοκ**, γίνεται ένας έλεγχος ότι η δύο παραπάνω επικοινωνίες ολοκληρώθηκαν, οπότε η διεργασία απο εδώ και στο εξής εκτελεί τον **KNN για τον καινούριο μπλοκ** που της στάλθηκε. Εφόσον πραγματοποιήθηκε η *MPI_Irecv()*, στέλνει αυτό το νέο μπλοκ στην επόμενη διεργασία και προχωράει στην αναζήτηση γειτόνων για αυτό.

Χρόνος: Κάνοντας compile με “mpicc –name-.c –lm –O2” και εκτελόντας το με “mpirun –np 4 ./a.out 30” (Για numtasks=4, 60000 σημεία και k=30 γείτονες), κάθε διεργασία κάνει περίπου 60-70 δευτερόλεπτα(σωστό αν σκεφτείς 240/4=60)

Επεξήγηση κώδικα

Γενικά για ελέγχους χρησιμοποιώ την *MPI_Test*() μέσα σε ένα βρόγχο *while(!flag)* και όταν το όρισμα flag της συνάρτησης γίνει 1, σημαίνει ολοκλήρωση της επικοινωνίας και βγαίνει απο τον βρόγχο. Γιατί όχι MPI_Wait()? Γιατί δεν μου επιστρέφει κάτι που να μου χρησιμεύει.

Μετά λοιπόν τον KNN για το αρχικό της μπλοκ, η διεργασία έχει ήδη λάβει το αρχικό μπλοκ της *previous,* άρα **το *if(flag) flag=0; else {MPI_Irecv()…}* την αποτρέπει από το να ξανακάνει μια λάθος λήψη**. Αφού όμως θέτω *flag=0* , αυτό γίνεται μόνο μια φορά, οπότε πλέον μπορεί να δεχτεί νέα μπλοκ.

Μέσα στο else δέχεται το καινούριο μπλοκ και ελέγχει αν το προηγούμενο *MPI_Isend()* που έκανε αλλά και το νέο MPI_Irecv() ολοκληρώθηκαν. Για αυτή την *MPI_Isend()* έχω βάλει έναν επιπλέον έλεγχο, διότι όταν πλέον η διεργασία παραλάβει το original μπλοκ της *next* διεργασίας απο την *previous,* δεν έχει νόημα να το στείλει.

Οι μεταβλητές blocks, *cnt, cnt2* και *shift* χρησιμοποιούνται για την σωστή τύπωση των γειτόνων από *previous* διεργασίες. Για παράδειγμα, στην πρώτη λήψη θέλω να ελέγξω τους previous γείτονες, στην δεύτερη όμως τους *previous*-1 κ.ο.κ. Ο cnt με αρχική τιμή 0 πληρεί αυτό, αν *(previous-cnt)>0*. Όταν φτάσει *(previous-cnt)*=0, θέλω περιστροφή, άρα ενεργοποιείται το flag *shift* και χρησιμοποιείται ο *cnt2=numtasks*-1. **Η εφαρμογή του KNN θα σταματήσει όταν ο μετρητής *blocks* γίνει ίσος με *numtasks-1*,** που σημαίνει ότι από την διεργασία έχουν περάσει όλα τα μπλοκ των υπόλοιπων διεργασιών.

Added: το index στην πρώτη εφαρμογή του KNN αποθηκεύεται με μία μετατροπή, ώστε να εξαρτάται από το id της τάδε διεργασίας => *array[i][k-1].pos=j+id*block*;

- **MPI blocking έκδοση**

Δυστυχώς δεν κατάφερα να κάνω το πρόγραμμα να δουλέψει σωστά, αλλά θα μπω στον κόπο να εξηγήσω την σκέψη μου.

Η διαφορά με τη non-blocking έκδοση είναι ότι τα *MPI_Recv()* και *MPI_Send()* πρέπει να γίνουν μετά την εφαρμογή του KNN για το original μπλοκ της κάθε διεργασίας. Μέσα λοιπόν σε μια while, αν ο μετρητής blocks =0 τότε ξεκινά την επικοινωνία η διεργασία με

id=0, στέλνοντας(έπειτα περιμένει σε ενα receive) το original μπλοκ της στην 1, η οποία περιμένει να το παραλάβει. Αυτή με την σειρά της στέλνει το δικό της original στην 2 κ.ο.κ. μέχρι η *id=numtasks*-1 να στείλει στην 0. Μόλις ολοκληρωθεί η *MPI_Recv()*, η διεργασία εφαρμόζει στα δεδομένα KNN και στέλνει στην next το received μπλοκ. ΄Ολο αυτό γίνεται μέχρι και πάλι ο μετρητής *blocks=numtasks*-1, οπότε και κλείνει ο κύκλος. Τελικά απο αυτό που εμφανίζεται απο τα printf όμως καταλαβαίνω ότι το original block της 0 κάνει απλά κύκλους και μόνο αυτό γίνεται sent και received…?

ΤΕΛΙΚΗ ΣΗΜΕΙΩΣΗ : Το grid δεν ήθελε να μου τρέξει το executable και έτσι οι χρόνοι που μέτρησα είναι καθαρά του pc(4 πυρήνες), οπότε και παρουσιάζω τον καλύτερο χρόνο(αν και μερικές φορές για 5 διεργασίες έτρεχε πιο γρήγορα). Δεν είμαι σίγουρος αλλά νομίζω επειδή χρησιμοποίησα άλλη έκδοση και όχι OpenMPI ,γιατί μου βγάζει αυτό το error και επειδή το κατάλαβα λίγο αργά δεν είχα κουράγιο να το ξαναπροσπαθήσω απο την αρχή. Το submit.sh το πήρα όπως το δίνει το grid, αλλάζοντας μόνο προφανώς το auth σε pdlab οπότε δεν παίζει ρόλο.