Παράλληλα & Διανεμημένα Συστήματα
2η Εργασία

Ραφαήλ Μπουλογεώργος 9186
rafampou@ece.auth.gr
Κυριακή 1/12/2019

LINK GITHUB

Ο κώδικα στο ζητούμενο code.tat.gz αρχείο έχει ανεβεί στο github στο link

https://github.com/rafampou/mpi/raw/master/code.tar.gz

Η εργασία μαζί με την αναφορά και τον tester έχει ανέβει στο github στο link

https://github.com/rafampou/mpi

Ελλείψεις στην παρούσα αναφορά σχετικά με τις δοκιμές και τα συμπεράσματα οφείλονται στην καθυστέρηση της συστοιχίας Aristotle από όπου μας ζητήθηκε να ανεβάσουμε και να εκτελέσουμε τον κώδικά μας.

Η τελική αναφορά θα βρίσκεται στο 2ο link με όνομα PDS_9186_ex2_report.pdf.

ΕΙΣΑΓΩΓΗ

Σκοπός της εργασία αυτής είναι η παραγωγή κώδικά σε C για την δημιουργία αλγορίθμου εύρεσης των k κοντινότερων σημείων από κάθε σημείο x που ανήκουν στον χώρο X.
Ο αλγόριθμος θα γίνεται αρχικά σειριακά για όλα τα σημεία και θα πραγματοποιείτε από μία διεργασία και έπειτα ο τμηματικά και παράλληλα με την χρήση MPI.

Η χρήση των MPI (Message Passing Interface) επιτρέπει την ανταλλαγή μηνυμάτων και επεξεργασία δεδομένων από παράλληλες διεργασίες που δεν εκτελούνται απαραίτητα στον ίδιο υπολογιστή. Ο σκοπός της χρήσης τους είναι η χρήση μνήμης διαφορετικών αρχιτεκτονικών για την εύρεση τον knn. Η χρήση των MPI επιτρέπει την παράλληλοποίηση του προγράμματος και την επικοινωνία υπολογιστών ακόμα και σε υπολογιστές με τελείως διαφορετική αρχιτεκτονική.

ΣΥΝΤΟΜΗ ΕΠΕΞΗΓΗΣΗ ΚΩΔΙΚΑ

Σειριακή υλοποίηση

Στο αρχείο *src/knnring_sequential.c* βρίσκεται ο κώδικάς για το πρώτο μέρος την εργασίας (V0).

Η ζητούμενη συνάρτηση με τα δοσμένα ορίσματα είναι και η βασική συνάρτηση εύρεσης των k κοντινότερων γειτόνων (knn). Η συνάρτηση αυτή εκτελεί των υπολογισμών αποστάσεων σύμφωνα με τον παρακάτω κώδικα Matlab $D = sqrt(sum(X^2, 2) - 2 * X * Y.' + sum(Y^2 2).')$;

Για τον υπολογισμό του πρώτου και του δεύτερου αθροίσματος χρησιμοποιείτε SumX και SumY αντίστοιχα. Ενώ για τον πολλαπλασιασμό πινάκων χρησιμοποιείτε η βιβλιοθήκη cblash.

Τα αποτελέσματα πριν μπουν στην δομή knnresult knn αποθηκεύονται και ταξινομούνται στον πίνακα D (nxm) και επιλέγονται τα k από τα n σημεία.

Για την ταξινόμηση χρησιμοποιήθηκε όπως και στην πρώτη εργασία εξωτερική select sort και έπειτα qsort για τα k πρώτα στοιχεία.

Όλοι οι πίνακες είναι σε Colum Major και ισχύει $D_{nxm}[i][j] = D[i + j * n]$

Synchronous

Για την υλοποίηση στο αρχείο *src/knnring_mpi_v1.c* με την χρήση p MPI διεργασιών χρησιμοποίησα την συνάρτηση distrAllkNN με τα δοθέντα ορίσματα και 2 buffers yRecv και ySent για αποστολή και λήψη πακέτων. Σε μια for-loop εκτελείτε ο αλγόριθμος από το V0 για τα στοιχεία που λάβαμε και έπειτα με merge-sort ταξινομούνται τελικό πίνακα myKnn.

Σημαντική είναι η χρήση του offset για τα ids των στοιχείων που λαμβάνουμε. Rank ο αριθμός της διεργασίας, numtasks ο συνολικός αριθμός διεργασιών και n ο αριθμός των επαναλήψεων από την αρχική τιμή. Να τονιστεί ότι τα ids ξεκινάνε από το task1 και τελευταίο το task0 σύμφωνα με τον tester_mpi.c

$$idPolarization = (rank + numtasks - l - 2)\%numtasks * n;$$

Asynchronous

Για την 2η υλοποίηση στο αρχείο *src/knnring_mpi.c* ο κώδικας δεν διαφοροποιείτε ουσιαστικά. Προσθέτω έναν ακόμα buffer resvBuff για να λαμβάνω τα νέα δεδομένα ενώ χρησιμοποιώ τα προηγούμενα. Ενώ περιμένω να ολοκληρωθούν η αποστολή και η λήψη δεδομένων όταν χρειαστούν νέα δεδομένα. Επομένως ο χρόνος αποστολής και λήψης είναι ο χρόνος που διαρκεί η εντολή MPI_waitall. Ο υπόλοιπος χρόνος αντιστοιχεί με τον χρόνο εκτέλεσης των εντολών και δεν προσμετράτε.

Ο χρόνος εκτέλεσης στα mpi είναι ο μέγιστος χρόνος από τον χρόνο εκτέλεσης των tasks

ΑΠΟΤΕΛΕΣΜΑΤΑ ΜΕΤΡΗΣΕΙΣ

Τοπική εκτέλεση

Τα παρακάτω αποτελέσματα έγιναν σε vm με τα εξής χαρακτηριστικά.

Επεξεργαστής: Intel® Core i7-7500 @ 2.70GHz, 2 Processors , 4 Logical Processors
Ram: 5.6GB με 2GB Swap σε HDD

	Sequential	Synchronous		Asynchronous	
		Job Time	Iterate Time	Job Time	Iterate Time
d=37 , k=13					
n=1423 m=762	0,122	2,414	0,605	1,401	0,016
n=2000 m=1800	0,343	4,873	0,256	6,311	0,056
n=5000 m=4500	-	58,770	0,987	34,879	0,008
n=200 m=180					
n=10 m=9					
d=200 , k=20					
n=1423 m=762	0,423	4,652	0,046	4,980	0,006

Εκτέλεση στην συστοιχία

	Sequential	Synchronous		Asynchronous	
		Job Time	Iterate Time	Job Time	Iterate Time
d=37 , k=13 n=1423 m=762 n=2000 m=1800 n=5000 m=4500 n=200 m=180 n=10 m=9					
d=300 , k=20 n=1423 m=762 n=2000 m=1800 n=5000 m=4500 n=200 m=180 n=10 m=9					

ΣΥΜΠΕΡΑΣΜΑΤΑ

Όπως φαίνεται από τις μετρήσεις η χρήση τον mpi σε τοπικό επίπεδο δεν είναι ιδιαίτερα αποδοτική για παράλληλο προγραμματισμό καθώς η επικοινωνία των διεργασιών είναι χρονοβόρα.

Εξάλλου η δημιουργία των MPI ξεκίνησε για την εκτέλεση μεγάλων προγραμμάτων των οποίων τα δεδομένα είναι μοιρασμένα σε μνήμες πολλών διαφορετικών συστημάτων.

Η ασύγχρονη αποστολή και λήψη επιταχύνει ιδιαίτερα το πρόγραμμα με έντονη την διαφορά στο χρόνο να εντοπίζεται για μεγάλες τιμές του n όπου ο χρόνος υπολογισμού των αποστάσεων είναι ανάλογος του χρόνο αποστολής και λήψης. Αποτελεί ξεκάθαρα μια αποτελεσματική χρήση των MPI