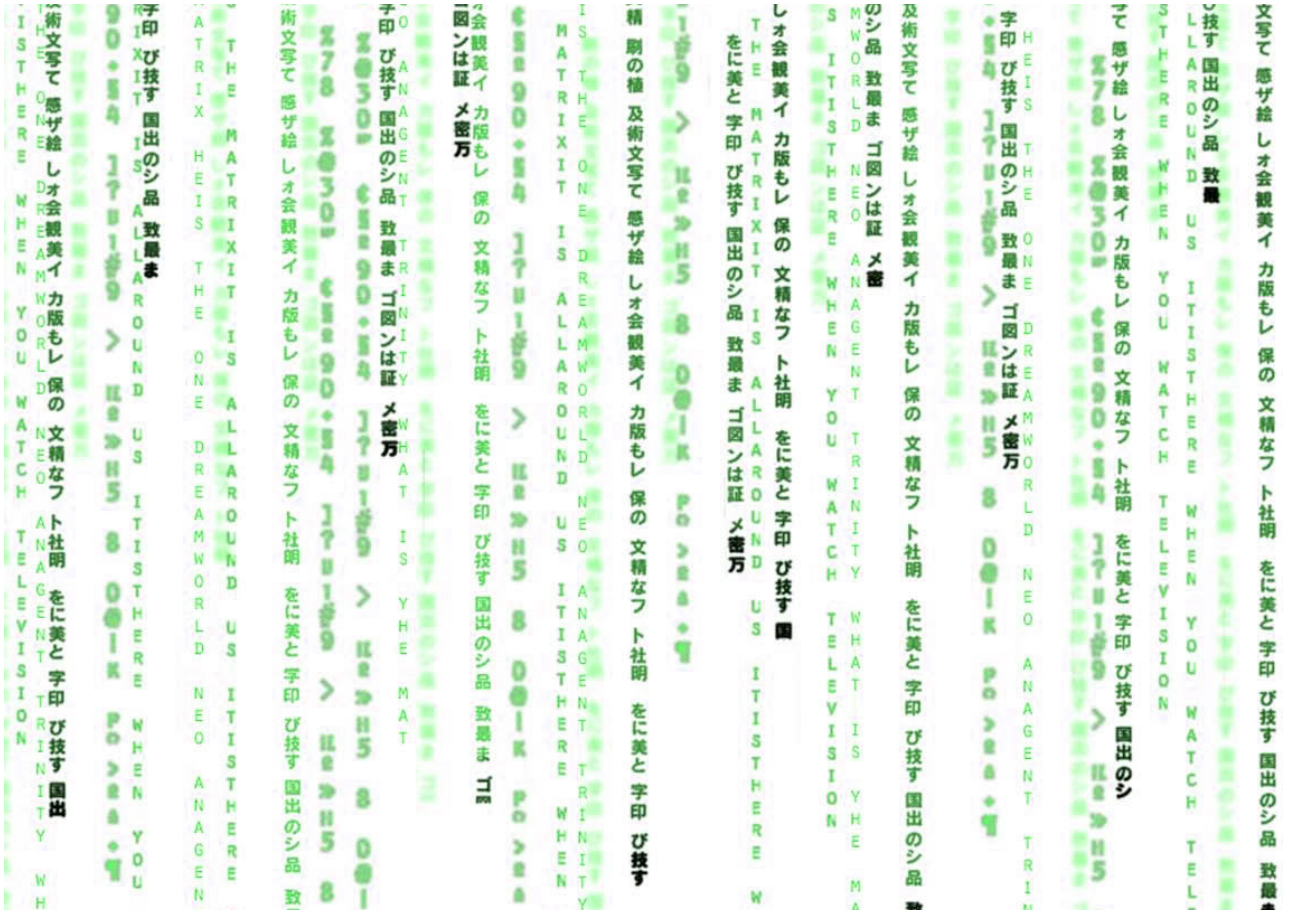
Παράλληλα και Διανεμημμένα Συστήματα

Εργασία 2 - Αναζήτηση κοντινού γείτονα

ΠΈΤΡΟΣ ΜΗΤΣΈΑΣ 7925 - 7Ο ΕΞΆΜΗΝΟ, ΤΟΜΈΑΣ ΗΛΕΚΤΡΟΝΙΚΉΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΏΝ



Παρουσίαση του αλγορίθμου

Το πρόβλημα αφορά τη δημιουργία δύο συνόλων σημείων {Q,C} με τυχαίες συντεταγμένες στο μοναδιαίο κύβο και εύρεση του κοντινότερου σημείου $\in$ C για κάθε σημείο $\in$ Q.

Ένας τρόπος αντιμετώπισης του προβλήματος αυτού με χρήση παράλληλου προγραμματισμού, είναι ο χωρισμός του κυβου [0,1) σε υποσύνολα και η ανάθεση των υποσυνόλων σε διεργασίες (μπορεί να τρέχουν σε διαφορετικούς πυρήνες ή διαφορετικούς υπολογιστές).

Οι υποψήφιοι κοντινοί γείτονες ενος σημείου, θα βρίσκονται προφανώς στον υποκύβο που ανήκει το σημείο και στους 26 υποκύβους που εφάπτονται σε αυτόν. Κάποιοι απο αυτούς ανήκουν στη διεργασία, ενω οι υπόλοιποι βρίσκονται σε άλλες διεργασίες.

Η επικοινωνία μεταξύ των processes έγινε με χρήση του πρωτόκολλου MPI. Επειδή οι χρόνοι των εντολών Send και Receive είναι υπολογίσιμοι, καταβλήθηκε προσπάθεια να γίνουν όσο το δυνατό λιγότερες επικοινωνίες μεταξύ των διεργασιών. Παρακάτω περιγράφεται βήμα-βήμα ο αλγόριθμος.

1. Η Master-Process με rank=0, δημιουργεί δύο arrays, μήκους (numberOfPoints) που καθορίζεται απο το χρήστη. Τα arrays αυτά περιέχουν τα αρχικά σύνολα Q, C. Κάθε σημείο είναι ουσιαστικά μια δομή (στη γλώσσα C), που περιέχει τις συντεταγμένες του, τις συνεταγμένες του υποκύβου στον οποίο βρίσκεται, την αρίθμηση (box id) του κουτιού απο (0) - έως (αριθμός κουτιών), ένα μοναδικό id, την απόσταση απο τον κοντινότερο γείτονα, και το id του κοντινότερου γείτονα.

2. Μετά τη δημιουργία των συνόλων, γίνεται ο χωρισμός τους στα κουτιά. Το κουτί είναι μια άλλη δομή, που αποτελείται απο ένα σύνολο σημείων και το αναγνωριστικό του κουτιού. Επειδή ο χρήστης δίνει μόνο τον αριθμό των κουτιών (σαν δύναμη του 2) που επιθυμεί, δηλαδή τον όγκο του πλέγματος, πρέπει να βρεθούν οι ακριβείς διαστάσεις. Αυτό γίνεται ώς εξής:

Επιθυμούμε το πλέγμα να είναι όσο το δυνατό πιο κυβικό. Έστω οτι ο όγκος ισούται με 2^α. Παίρνουμε την πρώτη διάσταση να είναι ίση με τον κοντινότερο ακέραιο α/3 (στρογγυλοποίηση προς τα κάτω). Η δεύτερη διάσταση είναι ίση με τον κοντινότερο ακέραιο του (α-(πρώτηδιάσταση))/2 και η τρίτη το υπόλοιπο. Έτσι αν για παράδειγμα α=13 τότε
διάσταση(x)=4, διάσταση(y)=(13-διάσταση(χ))/2=4, διάσταση(z)=13-διάσταση(χ)-διασταση(y)= 5.

3. Έπειτα κάθε κουτί τοποθετείται στο κατάλληλο process. Κάθε διεργασία έχει στην κατοχή της (ογκος πλέγματος)/(αριθμός διεργασιών) κουτιά. Επομένως για παραδειγμα άν έχουμε 16 κουτιά και 4 διεργασίες, τα κουτιά με αρίθμηση 0…4 τοποθετούνται στη 1η διεργασία, 5…8 στη 2η και ούτω καθεξής.

Μέχρι τώρα η αρχική διεργασία έχει φτιάξει 2 πίνακες εστω P_q και P_c. Οι πίνακες έχουν μήκος ίσο με τον αριθμό των διεργασιών. Κάθε στοιχείο του πίνακα περιλαμβάνει (ογκος πλέγματος)/(αριθμός διεργασιών) κουτιά και κάθε κουτί περιλαμβάνει κατα μέσο όρο (αριθμός σημείων)/(ογκος πλέγματος) σημεία.

4. Το master-process τώρα στέλνει σε κάθε διεργασία μεσω MPI_Send τα κουτιά που της αντιστοιχούν. Μέχρι τώρα οι υπόλοιπες διεργασίες περίμεναν να ”ακουσουν” απο την 1η. Το send γίνεται ώς εξής:

Η διεργασιά rank=0, μετράει κάθε στοιχείο του P_q[i], έστω κ το αποτέλεσμα της μέτρησης. Στέλνει διαδοχικά τα κουτιά της p_q[i] στην διεργασία i. Ψευδοκώδικας

```
Απο i=0 -> αριθμός διεργασιών
        Απο j=0-> αριθμός κουτιών ανα διεργασία
                κ=Μέτρα(P_q[ i ][ j ].points);
        MPI_Send(P_q[ i ][ j ].points, κ, MPI_τυπος_σημειο, i, 0, MPI_COMM_WORLD);
        MPI_Send(P_q[ i ][ j ].boxId, 1, MPI_τυπος_INT, i, 0, MPI_COMM_WORLD);
```

Αντίστοιχα οι υπόλοιπες διεργασίες ακούν απο την πρώτη και αποθηκεύουν κάθε εισερχόμενο κουτί σε μια δομή κουτιών.

5. Τώρα όλες οι διεργασίες δουλεύουν και αρχίζει η αναζήτηση. Αρχικά κάθε διεργασία κάνει ένα sweep σε όλα τα σημεία που έχει, και βρίσκει ποιά απο αυτά βρίσκονται στα περιθώρια του κουτιου που ανήκουν. Για τα σημεία αυτά, δημιουργεί έναν χάρτη 26 στοιχείων, που περιέχει το boxId των γειτονικών κουτιών και εντοπίζει σε ποιό process ανήκουν τα κουτια. Έπειτα τοποθετεί το σημείο σε ένα outbox με διεύθυνση το process.

Μετά το πέρας της εργασία αυτής, κάθε διεργασία έχει τώρα ένα outbox (αριθμός διεργασιών) στοιχείων, οπου κάθε στοιχείο είναι ένα σύνολο σημείων.

6. Οι διεργασίες στέλνουν τώρα το outbox. Η αποστολή γίνεται ασύγχρονα, ωστε να γίνουν ακόμα κάποιες δουλείες. Μετα την αποστολή, κάθε διεργασία βρίσκει για κάθε σημείο που της αντιστοιχεί, τον κοντινότερο γείτονα στο ίδιο κουτί. Αποθηκεύει το id του γείτονα, στο closestNeighbourId του σημείου.

7. Τώρα η διεργασία κάνει το πρώτο receive. Λαμβάνει (αριθμός διεργασιών) σύνολα σημείων και τα αποθηκεύει σε ένα inbox.

8. Για κάθε σημείο του inbox, βρίσκει τον κοντινότερο γείτονα, ψάχνοντας το κατάλληλο κουτί στο δικό της υποσύνολο του C. Τοποθετεί το σημείο που βρήκε σε ένα δεύτερο outbox, με διεύθυνση το process απο το οποίο ήρθε το σημείο του inbox.

9. Στέλνει το δεύτερο outbox, πίσω στα άλλα processes

10. Κάνει Receive απο τις άλλες διεργασίες σε ένα δεύτερο inbox. Για κάθε σημείο που ήταν στο περιθώριο του κουτιού του, ελέγχει αν η απόσταση του σημείου απο τον τοπικό γείτονα είναι μεγαλύτερη απο την απόσταση απο τα σημεία

που λήφθηκαν απο τις άλλες διεργασίες και αφορούν το σημείο αυτό. Αν είναι, ενημερώνει το closestNeighbourId του σημείου.

Συνοπτικά λοιπόν η σειρά βημάτων απο κάθε διεργασία μετά τη λήψη των αρχικών κουτιών είναι:

1. Συγκεντρώνω όλα τα σημεία που βρίσκονται στις άκρες των κουτιών και τα στέλνω στις αρμόδιες διεργασίες, ζητώντας τον κοντινότερο γείτονα

2. Για ολα τα σημεία Q που διαθέτω, ψάχνω τον κοντινότερο γείντονα C που βρίσκεται στο ίδιο κουτί

3. Λαμβάνω τα σημεία απο τις άλλες διεργασίες και αναζητώ τον κοντινότερο γείτονα C, στο κατάλληλο κουτί

4. Στέλνω πίσω τα σημεία που βρήκα

5. Λαμβάνω τα σημεία που ζήτησα αρχικά και ελέγχω αν βρίσκονται πιο κοντά απο τον γείτονα του τοπικού κουτιού.

Όπως είδαμε, καθόλη τη διάρκεια που τρέχει το πρόγραμμα γίνονται δύο μεγάλα Send και δύο Receive. Στέλνονται μόνο τα σημεία που βρίσκονται στις άκρες των κουτιών καθώς μόνο αυτά ενδέχεται να έχουν κοντινότερο γείτονα σε άλλο κουτί. Έτσι περιορίζεται το μέγεθος των πακέτων που αποστέλλονται.

Μετρήσεις

Παρακάτω φαίνεται ο χρόνος εκτέλεσης του προγράμματος για διαφορετικά μεγέθη κουτιών, συνόλων σημείων και διεργασιών. Να σημειωθεί οτι η εκτέλεση έγινε σε laptop με διπύρηνο core i5 with hyper-threading (2 virtual cores / physical core). Η πρόσβαση στο Hellasgrid τη δεδομένη στιγμή δεν ήταν δυνατή. Παρόλα αυτά, όπως θα δούμε φαίνεται ξεκάθαρα το πλεονέκτημα χρήσης multi-core/multi-processor συστημάτων για τον συγκεκριμένο αλγόριθμο.

Οι χρόνοι που παρουσιάζονται περιλαμβάνουν το χωρισμό των αρχικών συνόλων σε κουτιά, την κατανομή των κουτιών σε διεργασίες, την αποστολή τους και την εύρεση του κοντινότερου γείτονα για κάθες σημειο που ανήκει στο Q. Σε αυτό το σημείο να πούμε οτι η κατανομή των κουτιών με τον πιο απλό αλγόριθμο λαμβάνει αρκετά μεγάλο χρόνο (για κάθε κουτί απο 1 εως (αριθμος κουτιων) να ψάξει ανάμεσα στα (αριθμος σημειων) σημεία αν το σημείο ανήκει στο κουτι - χρονος (αριθμος κουτιων * αριθμος σημειων = απαγορευτικά μεγαλος). Γι αυτό έγινε επανασχεδιασμός του αλγορίθμου με χρήση της qsort και βοηθητικών μεταβλητών. (Παράδειγμα η ανάθεση κουτιών χωρίς qsort για 2^20 σημεία και 2^11 κουτιά: 16.47 sec, με qsort: 1.31 sec).

Ακόμη, επειδή η επεξεργαστική ισχύς είναι περιορισμένη σε σχέση με το Grid, έγιναν κάποιες μετρήσεις στο διάστημα (αριθμος κουτιων)= 2^10 … 2^16, (αριθμός σημείων)= 2^20 … 2^25, με αριθμό διεργασιών = 8. Παραπάνω διεργασίες θα ήταν ανώφελο να χρησιμοποιήσουμε καθώς το (εν δυνάμει 4 πύρηνο) σύστημα, θα χαμηλώσει την ταχύτητα σε κάθε διεργασία με αποτέλεσμα να αυξάνεται ο συνολικός χρόνος.

Θα δούμε τώρα τον πίνακα χρονου (σε δευτερόλεπτα) σημείων/κουτιων για 8 διεργασίες.

Κουτιά Σημεία	2^10	2^11	2^12	2^13	2^14	2^15	2^16
2^20	3,08	2,89	2,92	3,30	5,9	13,2	40,45
2^21	7,43	5,85	5,94	6,33	8,5	17,22	46,31
2^22	X	X	13,67	14,23	16,91	25,35	61,30
2^23	X	X	40,19	34,23	35,37	43,87	83,41
2^24	X	X	X	95,39	81,88	98,33	X
2^25	X	X	X	X	221,10	X	X

Εδώ μπορούμε να κάνουμε κάποιες παρατηρήσεις:

- Ο καλύτερος χρόνος που μετρήθηκε βρίσκεται κατα μήκος του πράσινου βέλους.
- Για ένα δεδομένο αριθμό σημείων, ξεκινώντας απο λίγα κουτιά, ο χρόνος μειώνεται, φτάνει ένα ελάχιστο, και αυξάνεται πάλι. Αυτό συμβαίνει γιατι, αν έχω λίγα κουτιά τοτε θα έχω περισσότερα σημεία ανα κουτί, και η πολυπλοκότητα αναζήτησης σημείων στο ίδιο κουτί είναι N^2. Αν αντίθετα έχω πολλά κουτιά, τότε ανεβαίνει ο αριθμός των σημείων που αποστέλλονται ανάμεσα στις διεργασίες, οπότε κάθε process ψάχνει περισσότερα ξένα στοιχεία και δεδομένου οτι τα processes τρέχουν στον ίδιο επεξεργαστή, ο χρόνος αυξάνεται.

Επειδή το πρόβλημα είναι μεγάλου μεγεθος, για τον σειριακό αλγόριθμο, θα επαναλάβουμε τις μετρήσεις για μικρότερο αριθμό σημείων ώστε να βγούν κάποια ακόμα συμπεράσματα. Παρακάτω φαίνεται η απόδοση του σειριακού αλγορίθμου, δηλαδή ο ίδιος αλγόριθμος για 1 process και 1 box, (πολυπλοκότητα N^2 - ασύμφορος).

Σημεία Κουτιά	2^14	2^15	2^16	2^17
1	1,05	4,16	17,27	81,90

Τέλος θα δούμε τη διαφορά του να τρέχει ο κώδικας σε ενα πυρήνα ή σε 2, με 2^18 σημεία. Σε αυτόν τον πίνακα φαίνεται μόνο ο χρόνος αναζήτησης, χωρίς το χρόνο ανάθεσης των σημείων σε κουτιά, ωστε να γίνει εμφανής η διαφορά στους χρόνους.

Κουτιά Πυρήνες	2^8	2^9	2^10	2^11
1	1,18	0,72	0,55	0,66
2	0,48	0,38	0,43	0,47