Κωνσταντίνος Πράττης Ιανουάριος 2023

*Παράλληλα και Διανεμημένα Συστήματα, Αναφορά 2ης Εργασίας.*

Εύρεση των κ-nearest-neighbors(knn) με χρηση ‘MPI’ σε Συστηματα διανεμημενης μνημησ

Σκοπός αυτής της άσκησης είναι η υλοποίηση του αλγορίθμου brute-force εύρεσης των KNN ενός συνόλου σημείων **X** (**query**, διαστάσεων **m x d**) στο σύνολο **Y** (**corpus**, διαστάσεων **n x d**). Συγκεκριμένα σε δύο εκδοχές. Μία χωρίς MPI (***V0***) και μία με χρήση MPI (***V1***). Σε αυτήν την αναφορά δίνονται σύντομα πληροφορίες/διευκρινήσεις για την μορφή του κώδικα και γίνεται μία παρουσίαση των αποτελεσμάτων και συμπερασμάτων που προέκυψαν από τις εκτελέσεις.

Ο κώδικας βρίσκεται στο Github link:

<https://github.com/kprattis/parallel_programming_ex_2>

* **Ανάλυση V0**

Ο αλγόριθμος υλοποιήθηκε σε γλώσσα C. Στο κύριο σώμα του, γίνονται τα εξής:

1. Υπολογίζεται ο πίνακας των αποστάσεων **D**, διάστασης **m x n**. Επειδή όμως παρατηρήθηκε ότι είναι αρκετά μεγάλος σε μέγεθος, αντί να υπολογίζεται ολόκληρος, υπολογίζεται σε **blocks** μεγέθους: **m x BLOCKSIZE**, όπου το *BLOCKSIZE* ορίστηκε έτσι ώστε εάν το γινόμενο m x n είναι μεγαλύτερο από ένα μέγιστο μέγεθος (*MAXSIZE*), τότε το γινόμενο BLOCKSIZE x m να είναι μικρότερο ή ίσο του MAXSIZE. Έτσι, εκτελείται μία επαναληπτική διαδικασία υπολογισμού των knn για κάθε block έως ότου να καλυφθεί ολόκληρο το corpus. Σε κάθε επανάληψη εξετάζονται τα επόμενα BLOCKSIZE σημεία του corpus.
2. Για τον υπολογισμό του πίνακα D, παρατηρήθηκε ότι με αποθήκευση της νόρμας κάθε σημείου του query εκ των προτέρων και την αποθήκευση των σημείων του corpus του τρέχοντος block σε πίνακες (μεγέθους m και BLOCKSIZE αντίστοιχα) και με απλή αντιγραφή των τιμών στις αντίστοιχες θέσεις υπάρχει κέρδος στην ταχύτητα με κόστος σχετικά μικρό στην μνήμη. Στη συνέχεια, με τη βοήθεια της ρουτίνας [***cblas\_dgemm***](https://netlib.org/blas/) (για το Χ \* Υt) ολοκληρώνεται ο υπολογισμός του τρέχοντος block-πίνακα αποστάσεων D.
3. Μετά από αυτό, με μία διαδικασία ***quickselect*** επιλέγονται οι k κοντινότεροι γείτονες στο blocked corpus για κάθε σημείο του query. Μετά την πρώτη επιλογή, για τα επόμενα block ακολουθείται μία λογική ανανέωσης, όπου τα ήδη υπάρχοντα knn ενώνονται με αυτά του νέου block και επιλέγονται τελικά με εφαρμογή quickselect οι k από τους 2k κοντινότερους γείτονες. Σε όλη τη διαδικασία επιλογής, κρατούνται και οι δείκτες του συνολικού corpus, με αποτέλεσμα να δημιουργείται μνήμη για την αποθήκευση τους ανάλογη του μεγέθους του blocked D.

Να σημειωθεί ότι η όλη διαδικασία παραλληλοποιείται σε επίπεδο threads με χρήση της [OpenCilk](https://www.opencilk.org/), όπου αυτό είναι εφικτό. Επίσης διατηρούνται τα τετράγωνα των αποστάσεων και όχι οι αποστάσεις για την αποφυγή της ρίζας.

* **V1 – Υλοποίηση MPI**

Στο V1 γίνεται η υλοποίηση μίας παραλλαγής του παραπάνω σε [***MPΙ***](https://www.open-mpi.org/). Η παραλλαγή είναι ότι το σύνολο query είναι το ίδιο με το corpus συνολικά. Αν **p** είναι το σύνολο των mpi processes, τότε κάθε process παίρνει μόνο ένα μέρος του corpus-query μεγέθους **n/p**. Προφανώς ενδέχεται να περισσέψουν κάποια σημεία, τα οποία θα αντιστοιχηθούν στο τελευταίο process. Ο κώδικας ως επί το πλείστων είναι ο ίδιος με πριν όσον αφορά τον υπολογισμό του D, την διαίρεση σε blocks και την επιλογή των k κοντινότερων σημείων. Ως διαφορές, αξίζει να σημειωθούν τα παρακάτω:

1. Αρχικά, κάθε process χρησιμοποιεί σαν corpus το δικό του query. Στη συνέχεια, στέλνει το corpus του στον επόμενο διατηρώντας ίδιο query και ανανεώνει το corpus του λαμβάνοντας το corpus του προηγούμενου. Με αυτόν τον τρόπο, τα δεδομένα κινούνται σε έναν δακτύλιο και όλοι θα λάβουν τελικά όλα τα σημεία.
2. Χρησιμοποιείται non-blocking send του corpus (***MPI\_Isend***) πριν τον υπολογισμό των k κοντινότερων γειτόνων με αυτό. Επίσης, χρησιμοποιείται blocking receive (***MPI\_Recv***) μετά την λήξη του υπολογισμού των knn. Με αυτήν την μέθοδο, κάθε process στέλνει το δυνατόν νωρίτερα και λαμβάνει το δυνατόν αργότερα, ενώ στο ενδιάμεσο, όσο περιμένει τα επόμενα σημεία, εκτελεί τον υπολογισμό που πρέπει να κάνει για να ανανεώσει τους knn του query του. Έτσι, υπάρχει συνέχεια δουλειά για τις διεργασίες και αποφεύγεται η παθητική αναμονή των δεδομένων.
3. Κάθε process θα χρειαστεί να στείλει και να λάβει ***p – 1*** φορές το corpus (*την p-οστη φορά θα του επέστρεφαν τα αρχικά σημεία του*), ενώ θα πρέπει να υπολογίσει p φορές τους knn του query του (*με p διαφορετικά corpus*). Εδώ p συμβολίζεται το πλήθος των mpi tasks.
4. Επειδή το τελευταίο task μπορεί να έχει παραπάνω σημεία από τα υπόλοιπα, λαμβάνεται μέριμνα ώστε πριν από κάθε send ή receive τα processes να προσαρμόσουν τα μεγέθη των πινάκων τους αν πρόκειται να στείλουν ή να λάβουν τα σημεία που ξεκίνησαν ως query του τελευταίου task. Για τον ίδιο λόγο αλλά και για τον υπολογισμό των global id των knn δίνεται σαν όρισμα και το συνολικό μέγεθος όλων των σημείων Ν.

* **Έλεγχος Ορθότητας Υλοποιήσεων**

Για τον έλεγχο ορθότητας, χρησιμοποιήθηκαν ***d-διάστατα******regular grids*** *με* ***N*** *σημεία σε κάθε διάσταση*. Δημιουργήθηκε μάλιστα και μία συνάρτηση που φτιάχνει και αποθηκεύει σε ένα αρχείο τα σημεία αυτά, έτσι ώστε στην πρώτη γραμμή να δίνονται οι διαστάσεις, και έπειτα σε κάθε γραμμή να τυπώνεται ένα σημείο.

Οι V0 και V1 παίρνουν ως είσοδο ένα τέτοιο αρχείο και τυπώνουν σε ένα νέο αρχείο στην πρώτη γραμμή πληροφορίες για την εκτέλεση τους και στη συνέχεια πρώτα τον πίνακα με τις k κοντινότερες αποστάσεις και έπειτα τον πίνακα με τους δείκτες των knn στο corpus.

Ο έλεγχος βασίζεται στην εξής ιδέα: Για κάθε εσωτερικό σημείο x του πλέγματος υπάρχουν  σημεία του πλέγματος, ώστε το τετράγωνο της απόστασης του καθενός με το x να είναι ίσο με i, για . To παραπάνω βέβαια ισχύει για *d έως και 3*. Υλοποιήθηκε επομένως και μία συνάρτηση ***test\_results***, που παίρνει ως είσοδο ένα αρχείο του grid και το αντίστοιχο knn result του και ελέγχει για κάθε εσωτερικό σημείο του πλέγματος αρχικά εάν υπάρχουν τετράγωνα αποστάσεων ίσα με i στους k κοντινότερους γείτονες για κάθε i, και στη συνέχεια εάν τα indexes των k κοντινότερων γειτόνων αντιστοιχούν σε σημεία των οποίων η απόσταση είναι μικρότερη ή ίση από d. Οι έλεγχοι αυτοί επαρκούν, καθώς το **k** επιλέγεται να είναι , δηλαδή η μέγιστη απόσταση είναι d (*d <= 3*). Μπορεί να εφαρμοστεί και για d = 4.

Αν και ο έλεγχος γίνεται μόνο στα εσωτερικά σημεία αυτά αποτελούν την πλειονότητα του πλέγματος και δίνουν μία καλή ένδειξη ορθότητας. Για διάφορες τιμές λοιπόν N και d οι V0 και V1 φάνηκε να συμφωνούν με την θεωρητική ανάλυση σύμφωνα με την συνάρτηση ελέγχου και άρα δίνουν ορθά αποτελέσματα.

* **Αποτελέσματα - Συμπεράσματα για την απόδοση**

Για την εξαγωγή συμπερασμάτων για την απόδοση χρησιμοποιήθηκαν κάποια grids από αυτά που αναφέρθηκαν παραπάνω, αλλά και μερικά υποσύνολα τέτοιων grids μεγαλύτερων διαστάσεων. Επίσης, χρησιμοποιήθηκε η [Ιδρυματική συστοιχία του ΑΠΘ "*Αριστοτέλης*"](https://hpc.it.auth.gr/) για εκτέλεση runs με πολλούς επεξεργαστές και για μεγάλο μέγεθος αρχείων. Από μία τάξη μεγέθους και πάνω οι χρόνοι γίνονται υπερβολικά μεγάλοι, έτσι οι δοκιμές έγιναν μέχρι και για 1.000.000 σημεία corpus.

Παρακάτω παρουσιάζονται τα αποτελέσματα με τη βοήθεια διαγραμμάτων και πινάκων:

Αρχικά, τα V0, V1 εκτελέστηκαν τοπικά και παρατηρήθηκε ότι από έναν αριθμό σημείων και μετά, δεν ήταν δυνατό να χωρέσουν στην μνήμη οι πίνακες που χρησιμοποιούνται από αυτά. Επίσης, οι χρόνοι ήταν σχετικά αρκετά μεγάλοι. Ακόμα, το V1 δεν μπορούσε να εκτελεστεί τοπικά σε έναν υπολογιστή, λόγω της ανάγκης κάθε process για μνήμη. Για τον λόγο αυτό στη συνέχεια προτιμήθηκε το HPC του ΑΠΘ για την εκτέλεση του V1, ώστε να επαρκεί η μνήμη και η υπολογιστική δύναμη για τις διάφορες δοκιμές.

Ενδιαφέρον παρουσίασε αρχικά η διερεύνηση της βέλτιστης κατανομής μεταξύ threads και processes ενός σταθερού αριθμού CPU για το V1.

Διάγραμμα 1: Σύγκριση cilk threads - mpi tasks

Παρατηρούμε ότι στην συγκεκριμένη εφαρμογή προτιμούμε να έχουμε περισσότερα ***mpi processes*** από ότι ***cilk threads***. Μάλιστα, η διαφορά στους χρόνους γίνεται περισσότερο εμφανής όσο αυξάνεται ο αριθμός των σημείων (*εως και 12 φορές πιο αργό για 1000000 3d σημεία όταν μειώνω τα processes σε βάρος των threads per process*). Αυτό εξηγείται καθώς το κύριο πρόβλημα του αλγορίθμου είναι ότι σε μεγάλα μεγέθη του αριθμού των σημείων, εάν αυτά μπορούν να χωρέσουν στην μνήμη, διαιρούνται σε blocks τα οποία αναγκαστικά σειριοποιούνται μεταξύ τους. Έτσι, τα threads μπορούν να μας βοηθήσουν στην ταχύτητα του κάθε block, αλλά τα mpi processes παραλληλοποιούν τα blocks αυτά και προσφέρουν μεγαλύτερη βελτίωση στην απόδοση. Παρατηρήθηκε βέβαια και ότι μερικές φορές, υπάρχει ένα μικρό πλεονέκτημα να βάλουμε περισσότερα threads από ότι processes. Αυτό μπορεί να προκύπτει για δύο λόγους: Πρώτον, το κόστος της επικοινωνίας δύο processes είναι μεγάλο και τυχαίο ειδικά σε μία υπολογιστική συστοιχία πολλαπλών κόμβων. Δεύτερον, μικρού μεγέθους προβλήματα, έχουν περισσότερη ανάγκη από παραλληλοποίηση τύπου threads λόγω του ότι δεν χωρίζονται σε πολλά ή πολύ μεγάλα blocks.

Στο εξής προτιμήθηκε μεγαλύτερος αριθμός tasks και μικρότερος αριθμός threads per task.

Τέλος έγινε διερεύνηση των αποτελεσμάτων για διάφορες τιμές του αριθμού των σημείων και του αριθμού των διαστάσεων. Παρακάτω οι σχετικοί πίνακες και διαγράμματα:

Αρχικά, γίνεται εμφανές ότι με την αύξηση της διάστασης, αυξάνεται και ο χρόνος που απαιτείται για τον υπολογισμό των knn τόσο για το V0, όσο και για το V1. Επίσης, βλέπουμε ότι το V1 είναι 3 φορές περίπου πιο γρήγορο από το V0 και μάλιστα με τον ίδιο αριθμό επεξεργαστών.

Τέλος, παρατηρούμε ότι για μεγέθη έως και 50000 σημεία (d = 10) ο χρόνος που πετυχαίνουμε είναι μικρότερος του 1 sec με χρήση 12 mpi processes. Από εκεί και πέρα, παρατηρούμε ότι αυξάνεται πολύ γρήγορα η πολυπλοκότητα και για την επίτευξη εφικτών χρόνων χρειάστηκε η αύξηση των πυρήνων σε 80. Με 80 πυρήνες λοιπόν στα δύο τελευταία μεγέθη οι χρόνοι φαίνεται να είναι πολύ μεγάλοι και πάλι.

* **Αξιολόγηση Αποτελεσμάτων-Προτάσεις για βελτίωση**

Η V1 υλοποίηση, υποστηριζόμενη από ένα HPC σύστημα δίνει πολύ καλύτερες επιδόσεις από την V0 και μας παρέχει την δυνατότητα να λύσουμε και μεγαλύτερου μεγέθους προβλήματα. Συμπεραίνουμε ότι με χρήση mpi μπορούμε να αντιμετωπίσουμε προβλήματα που είναι απαιτητικά σε μνήμη που υπό άλλες συνθήκες δεν θα μπορούσαμε.

Τέλος, όσον αφορά πιθανές αλλαγές προς βελτιστοποίηση του κώδικα, μία πρόταση είναι ίσως κάποιος πιο αποτελεσματικός τρόπος για την επιλογή του blocksize, που θα προσαρμόζεται ανάλογα στις δυνατότητες της μνήμης κάθε υπολογιστικού συστήματος και δεν θα είναι απλά μία αυθαίρετη σταθερά, ενώ κάτι ακόμα που θα μπορούσε να προστεθεί είναι tests πάνω σε real datasets (π.χ. εικόνων).