Κ23α - Ανάπτυξη Λογισμικού Για Πληροφοριακά Συστήματα

Χειμερινό Εξάμηνο 2023– 2024

Άσκηση 3 – Παράδοση: Κυριακή 14 Ιανουαρίου 2024

Στο τρίτο μέρος της εργασίας θα ασχοληθούμε με την βελτιστοποίηση της απόδοσης του κώδικα που δημιουργήθηκε στα προηγούμενα τμήματα της άσκησης. Αυτό θα επιτευχθεί τόσο με την χρήση παραλληλίας όσο και με τη βελτιστοποίηση επιμέρους λειτουργιών.

Βελτιστοποιήσεις

Βελτιστοποίηση υπολογισμού απόστασης

Η ευκλείδεια απόσταση μεταξύ δύο σημείων $\mathbf{x}, \mathbf{y}$ στον N-διάστατο χώρο ορίζεται ως εξής:
$d_{\mathbf{x},\mathbf{y}} = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_N - y_N)^2}$, όπου x_i, y_i οι συνιστώσες των $\mathbf{x}, \mathbf{y}$ στη διάσταση i.
Στη συγκεκριμένη περίπτωση, δεν ενδιαφέρει ιδιαίτερα η απόλυτη τιμή της απόστασης, αλλά η σύγκρισή της με άλλες αποστάσεις. Επομένως, η ακριβή πράξη του υπολογισμού της τετραγωνικής ρίζας δεν είναι απαραίτητη. Σε αυτή την περίπτωση μπορεί να αρκεστούμε στον υπολογισμό της τετραγωνικής απόστασης $ds_{\mathbf{x},\mathbf{y}} = (x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_N - y_N)^2$.

Από την διωνυμική εξίσωση
$ds_{\mathbf{x},\mathbf{y}} = (x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_N - y_N)^2$
$ds = (\mathbf{x} - \mathbf{y})^2$
$ds = \mathbf{x}^2 + \mathbf{y}^2 - 2\mathbf{x}\mathbf{y}$

Παρατηρούμε στον υπολογισμό της απόστασης μεταξύ δύο σημείων θα χρησιμοποιούνται πάντα η τετραγωνική νόρμα κάθε διανύσματος (εσωτερικό γινόμενο του διανύσματος με τον εαυτό του). Οι νόρμες αυτές μπορούν να υπολογιστούν στην αρχή και να αποθηκευτούν για χρήση αργότερα.

Αρχικοποίηση KNN γράφου

Τα δένδρα τυχαίων προβολών (random projection trees) είναι μία αποδοτική τεχνική για την προσεγγιστική αναζήτηση εγγύτερων γειτόνων σε χωρικά δεδομένα (δεδομένα που αναπαριστώνται με διανύσματα). Η ιδέα είναι απλή: ξεκινάμε με την επιλογή ενός τυχαίου υπερεπιπέδου (hyperplane) που χωρίζει τα σημεία σε δύο σύνολα. Στη συνέχεια, αναδρομικά χωρίζεται το κάθε σύνολο με τον ίδιο τρόπο σε μικρότερους υπερκύβους, μέσω επιλογής διαδοχικών τυχαίων υπερεπιπέδων. Η δημιουργία του δένδρου ολοκληρώνεται όταν το μέγεθος του παραγόμενου συνόλου είναι μικρότερος ή ίσος από ένα

αριθμό D. Με τον τρόπο αυτό παράγεται ένα δένδρο τυχαίων προβολών. Στο τέλος μπορούμε να υπολογίσουμε το συνολο των αποστάσεων μεταξύ όλων των στοιχείων ενός φύλλου του δένδρου και να τα χρησιμοποιήσουμε για την αρχικοποίηση των ακμών του KNN γράφου

Με χρήση παράλληλων τεχνικών μπορούμε να παράξουμε περισσότερα τέτοια δένδρα και να χρησιμοποιήσουμε τις αποστάσεις μεταξύ των σημείων κάθε φύλλου των δένδρων για τον σχηματισμό του αρχικού γράφου..

Παράλληλη Εκτέλεση

Υλοποίηση

Η χρήση των νημάτων μπορεί να γίνει με δύο τρόπους είτε δημιουργώντας καινούργια νήματα για κάθε παράλληλο κομμάτι, είτε με την υλοποίηση ενός job scheduler. Αν και η δεύτερη επιλογή είναι πιο σύνθετη σε επίπεδο υλοποίησης, είναι συνήθως προτιμότερη ειδικά σε εφαρμογές, όπου η παράλληλη επεξεργασία γίνεται ασύγχρονα κατά τη διάρκεια εκτέλεσης του προγράμματος και φυσικά αποτρέπει την πολλαπλή δημιουργία των δομών των νημάτων. Για τους παραπάνω λόγους θα υλοποιήσετε ένα job scheduler για την παράλληλη επεξεργασία.

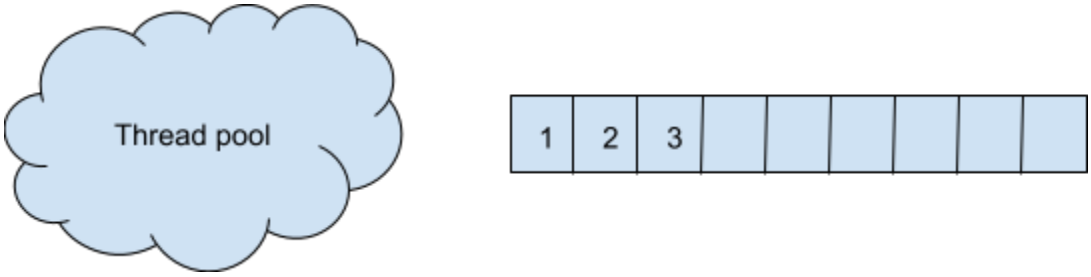
Job Scheduler

Εργασίες (Jobs)

Εργασία (Job) είναι μια ρουτίνα κώδικα η οποία θέλουμε να εκτελεστεί από κάποιο νήμα πιθανότατα παράλληλα με κάποια άλλη. Μπορούμε να ορίσουμε οτιδήποτε θέλουμε ως job και να το αναθέσουμε στον χρονοπρογραμματιστή.

Χρονοπρογραμματιστής Εργασιών (Scheduler) και Δεξαμενή Νημάτων (Thread Pool)

Ο χρονοπρογραμματιστής ουσιαστικά δέχεται δουλειές και αναλαμβάνει την ανάθεση τους σε νήματα, για προσωρινή αποθήκευση των εργασιών χρησιμοποιεί μια ουρά προτεραιότητας. Έστω ότι έχουμε μια Δεξαμενή νημάτων (thread pool) και μια συνεχόμενη ροή από ανεξάρτητες εργασίες (jobs). Όταν δημιουργείται μια εργασία, μπαίνει στην ουρά προτεραιότητας του χρονοπρογραμματιστή και περιμένει να εκτελεστεί. Οι εργασίες εκτελούνται με την σειρά που δημιουργήθηκαν (first-in-first-out - FIFO). Κάθε νήμα περιμένει στην ουρά μέχρι να του ανατεθεί μια εργασία και, αφού την εκτελέσει, επιστρέφει στην ουρά για να αναλάβει νέα εργασία. Για την ορθή λειτουργία ενός χρονοπρογραμματιστή είναι απαραίτητη η χρήση σημαφόρων στην ουρά, ώστε να μπλοκάρουν εκεί τα νήματα, και την κρίσιμη περιοχή (critical section), ώστε να γίνεται σωστά εισαγωγή και εξαγωγή εργασιών από την ουρά.



Σχήμα 2. Αναπαράσταση δομής job scheduler

Ενσωμάτωση δομής Job Scheduler

Στην αρχή του προγράμματος θα δημιουργείται η δομή του JobScheduler, η οποία αποθηκεύει όλα τα jobs, διαχειρίζεται την εκτέλεσή τους θα διατηρείται σε όλη τη διάρκεια εκτέλεσης του προγράμματος και θα καταστρέφεται στο τέλος.

```
struct JobScheduler{
      int execution_threads; // number of execution threads
      Queue* q; // a queue that holds submitted jobs / tasks
      p_thread_t* tids; // execution threads
      …
      // mutex, condition variable, ...
};
```

Εκτός της δομής του scheduler απαιτείται και μια δομή Job, η οποία καθορίζει ουσιαστικά τον κώδικα της εργασίας που εκτελείται παράλληλα. Η δομή αυτή πρέπει να μπορεί να υποστηρίζει διαφορετικούς τύπους εργασιών, τις οποίες εκτελεί ο scheduler παράλληλα, χωρίς να γνωρίζει τη λειτουργικότητά τους.

Ενδεικτικές λειτουργίες που υλοποιεί η δομή του scheduler είναι οι εξής

```
JobScheduler* initialize_scheduler(int execution_threads)
int submit_job(JobScheduler* sch, Job* j)
int start_execute(JobScheduler* sch)
int wait_all_tasks_finish(JobScheduler* sch)
int destroy_scheduler(JobScheduler* sch)
```

Παραλληλοποίηση

Η παραλληλοποίηση του προγράμματός, μπορεί να γίνει σε πολλά επίπεδα. Επισημαίνονται κάποια από τα jobs που μπορεί να εκτελεστούν:

- Υπολογισμός νόρμας σημείων (εσωτερικών γινομένων διανυσμάτων με τον εαυτό τους)
- Δημιουργία δένδρων τυχαίων προβολών

- Υπολογισμός αποστάσεων σε local joins
- Ενημέρωση ουράς προτεραιότητας για κάθε σημείο του γράφου μετά την ολοκλήρωση των υπολογισμών

Σημαντικό τμήμα του σχεδιασμού είναι το μέγεθος της κάθε εργασίας, ώστε να είναι αρκετά μεγάλη για να μην επιβαρύνεται από την ακριβή έναρξη/τερματισμό των εργασιών/threads, αλλά και τον χρονοπρογραμματισμό τους και αρκετά μικρή ώστε να μπορεί να διαμοιράζεται το επεξεργαστικό φορτίο αποδοτικά σε διαφορετικούς επεξεργαστές, χωρίς να πρέπει να μένουν αναξιοποίητοι κάποιοι για λόγους συγχρονισμού.

Τελική Αναφορά

Στη τελική αναφορά θα παρουσιάσετε μια σύνοψη ολόκληρη της εφαρμογής που υλοποιήθηκε. Μπορείτε να αναφέρετε πράγματα που παρατηρήσατε κατά την μοντελοποίηση/υλοποίηση της εφαρμογής σας, με αποτέλεσμα να σας οδηγήσουν σε συγκεκριμένες σχεδιαστικές επιλογές που βελτίωσαν την εφαρμογή σας σε επίπεδο χρόνου, μνήμης, κτλ.
Στην αναφορά πρέπει ακόμη να παρουσιάσετε ένα σύνολο από πειράματα, τα οποία θα δείχνουν το χρόνο εκτέλεσης για όλες τις επιλογές των δομών. Για παράδειγμα μπορείτε να αναφέρετε ή/και να παρουσιάσετε με διαγράμματα τον χρόνο εκτέλεσης, κατανάλωση μνήμης, πολυνηματισμό κ.α.
Τέλος, είναι απαραίτητο να παρουσιάσετε τις δομές που σας έδωσαν τους καλύτερους χρόνους εκτέλεσης για τα datasets και τον τελικό χρόνο/μνήμη, μαζί με τις προδιαγραφές του μηχανήματος που εκτελέσατε τα πειράματα. Η αναφορά δεν πρέπει να ξεπερνά τις 20 σελίδες.

Παράδοση εργασίας

Η εργασία είναι ομαδική, **2 ή 3 ατόμων**.
Γλώσσα υλοποίησης: C / C++ χωρίς χρήση stl.
Περιβάλλον υλοποίησης: Linux (gcc > 9.4+).

Παραδοτέα: Η παράδοση της εργασίας θα γίνει με βάση το τελευταιο commit πριν την προθεσμία υποβολής στο git repository σας. **Η χρήση git είναι υποχρεωτική**.
Στο αρχείο README.md θα αναφέρονται τα εξής:
- Ονοματεπώνυμα και ΑΜ των μελών της ομάδας
- Αναφορά στο ποιο μέλος της ομάδας ασχολήθηκε με ποιο αντικείμενο

Επιπλέον, εκτός από τον πηγαίο κώδικα, θα παραδώσετε μια σύντομη αναφορά, με τις σχεδιαστικές σας επιλογές καθώς και να εφαρμόσετε ελέγχους ως προς την ορθότητα του λογισμικού με τη χρήση ανάλογων βιβλιοθηκών (Software testing). Η ορθότητα τυχόν μεταβολών θα ελέγχεται με αυτοματοποιημένο τρόπο σε κάθε commit/push μέσω github actions.

Αναφορές

1. Wei Dong, Charikar Moses, and Kai Li. 2011. Efficient k-nearest neighbor graph construction for generic similarity measures. In Proceedings of the 20th international conference on World wide web (WWW '11). Association for Computing Machinery, New York, NY, USA, 577–586. https://doi.org/10.1145/1963405.1963487
2. Περιγραφή υλοποίησης του αλγορίθμου σε python για το project PyNNDescent https://pynndescent.readthedocs.io/en/latest/how_pynndescent_works.html
3. nndescent implementation in C++ for python https://github.com/brj0/nndescent