Εργασία 1
Παράλληλα και Διανεμημένα Συστήματα

Φραϊδάκης Ιωάννης (email: fraidaki@ece.auth.gr , ΑΕΜ : 10736)

11/11/2024

Εισαγωγή

Σκοπός της εργασίας είναι η συγγραφή κώδικα στη γλώσσα C για την προσεγγιστική εύρεση των k κοντινότερων γειτόνων (approximate-k-nearest-neighbors / ANN) ενός συνόλου Q σημείων/ερωτημάτων (*query-set*) ως προς ένα σύνολο C δεδομένων σημείων (*corpus-set*), διάστασης d και κατόπιν βελτιστοποίησής του χρησιμοποιώντας τεχνικές παράλληλου προγραμματισμού. Η προσεγγιστική εύρεση των k κοντινότερων γειτόνων στοχεύει στην εξισορρόπηση μεταξύ ταχύτητας και ακρίβειας, ιδίως όταν η ακριβής εύρεση των γειτόνων (knn) καθίσταται υπολογιστικά απαγορευτική λόγω μεγάλου πλήθους δεδομένων ή διαστάσεων. Θα εστιάσουμε στην περίπτωση όπου C=Q (με δυνατότητα εύκολης επέκτασης στη γενικότερη περίπτωση). Τέλος, θα προσπαθήσουμε να βρούμε την χρυσή τομή για το tradeoff μεταξύ ταχύτητας επεξεργασίας (queries per second) και ποσοστό ακρίβειας (recall).

Προτεινόμενος αλγόριθμος

Διαμερίζουμε το Q αναδρομικά σε διακριτά υποσύνολα έως ότου καθένα απ' αυτά έχει μέγεθος το πολύ **MIN_SIZE**. Για κάθε τέτοιο υποσύνολο υπολογίζουμε τους ακριβείς knn του και κατόπιν "ενώνουμε" (merge) τις επιμέρους λύσεις. Έστω λοιπόν Q1, Q2 δύο προς ένωση υποσύνολα (με Q1 ∩ Q2 = ∅). Ο τρόπος που γίνεται το merging είναι ο εξής : Επιλέγουμε **sample_size**, σε πλήθος, τυχαία σημεία από το Q1 για τα οποία βρίσκουμε τους ακριβείς knn στο Q1 ∪ Q2 υπολογίζοντας τις ευκλείδειες αποστάσεις με όλα τα σημεία του Q2 (με τα σημεία του Q1 έχουν γίνει ήδη οι υπολογισμοί από το base case της αναδρομής). Για καθένα από τα εναπομείναντα σημεία του Q1 (non-sampled), βρίσκουμε τα **N** πλησιέστερα δειγματοληπτημένα σημεία σ' αυτό. Χρησιμοποιώντας τους knn των N sampled points, δημιουργούμε ένα σύνολο υποψήφιων προς εξέταση σημείων (N*k στο σύνολο). Εξετάζουμε κάθε τέτοιο υποψήφιο σημείο και ενημερώνουμε κατάλληλα τους knn. Η ίδια διαδικασία επαναλαμβάνεται για το Q2.

Υλοποίηση

Σειριακή (μη προσεγγιστική)

Αρχίζουμε υλοποιώντας την **«σειριακή» μη προσεγγιστική** εκδοχή (task V0) προκειμένου να έχουμε μια βάση στον κώδικα αλλά και στα αποτελέσματα (benchmarks) ώστε να μπορεί να γίνει μετέπειτα σύγκριση των ακριβών (non-approximate) αποτελεσμάτων με τις προσεγγιστικές παραλλαγές του, τόσο ως προς το χρόνο εκτέλεσης όσο και για την εύρεση του πλήθους σωστών γειτόνων (recall). Να τονίσουμε σ' αυτό το σημείο ότι για τον υπολογισμό των αποστάσεων γίνεται χρήση συναρτήσεων από τη βιβλιοθήκη OpenBLAS, οι οποίες εκτελούνται σε πολλαπλά νήματα, με τον υπόλοιπο κώδικα όμως να παραμένει σειριακός.

Παράλληλη (προσεγγιστική)

Μετατρέπουμε τον σειριακό αλγόριθμο στον προσεγγιστικό και ξεκινάμε με την **OpenCilk** καθώς έτσι μπορούμε να καταλάβουμε άμεσα και εύκολα ποια κομμάτια αξίζουν να παραλληλοποιηθούν. Αυτό συμβαίνει λόγω της αυτόματης κατανομής φόρτου εργασίας και λόγω της εξασφάλισης για το πολύ δύο φορές πιο αργή εκτέλεση από τη βέλτιστη παράλληλη, αξιοποιώντας την έννοια του work stealing. Οπότε, χρησιμοποιώντας το εργαλείο perf των Linux, εντοπίζουμε τα σημεία που προκαλούν καθυστερήσεις (bottlenecks) στην ANN υλοποίηση και τα στάδια που θα ωφελούνταν από τη χρήση πολυνηματικού προγραμματισμού.

Στη συνέχεια, για κάθε αργό σημείο του κώδικα, εξετάζουμε δύο τεχνικές παραλληλοποίησης. Τη δημιουργία νέων νημάτων με cilk_spawn και τη χρήση του cilk_for για παράλληλους υπολογισμούς εντός των βρόχων. Τα αποτελέσματα είναι συμβατά με τη θεωρία : Η δημιουργία επιπλέον νημάτων (cilk_spawn) είναι αποδοτική μόνο στα αναδρομικά στάδια του αλγορίθμου. Για τα υπόλοιπα (ακριβής υπολογισμοί για τα sampled points και merging), η χρήση του cilk_for ώστε να επιτύχουμε ταυτόχρονη επεξεργασία των σημείων είναι πιο αποδοτική, λόγω χαμηλότερου overhead.

Τέλος, για τις υλοποιήσεις με OpenMP και Pthreads παραλληλοποιούμε τα ίδια σημεία ως εξής:

OpenMP: Η μόνη διαφορά έγκειται στις εντολές : cilk_spawn ➔ #pragma omp task , cilk_sync ➔ pragma omp taskwait , cilk_for ➔ pragma omp parallel for. Επίσης, ρυθμίζουμε τον scheduler σε dynamic mode για καλύτερο load balancing προσεγγίζοντας αρκετά την επίδοση της OpenCilk.

Pthreads: δημιουργία νέων νημάτων για την αναδρομή γίνεται δυναμικά με έλεγχο μέσω της μεταβλητής num_threads, η οποία ανανεώνεται με χρήση της . Η προσομοίωση της λειτουργίας του cilk_for επιτυγχάνεται με τεμαχισμό των βρόχων και κατανομής τους σε νήματα, διασφαλίζοντας με αυτό τον τρόπο ισόποσο υπολογιστικό φόρτο ανά νήμα.

Έλεγχος ορθότητας

Αρχικά, ελέγξαμε την ορθότητα της σειριακής υλοποίησης χρησιμοποιώντας τα προϋπολογισμένα δεδομένα (precomputed data) για την περίπτωση όπου C ≠ Q , επιβεβαιώνοντας ότι ο σειριακός αλγόριθμος παράγει τα αναμενόμενα αποτελέσματα. Στη συνέχεια, χρησιμοποιήσαμε αυτή τη σειριακή υλοποίηση για να υπολογίσουμε τα "ορθά" αποτελέσματα για τη περίπτωση που μελετάμε (C = Q), τα οποία αποτέλεσαν τη βάση για την αξιολόγηση των παραλληλοποιημένων υλοποιήσεων.

Για να διασφαλίσουμε την ορθότητα των παραλληλοποιημένων εκδόσεων, ξεκινήσαμε την εκτέλεση καθεμιάς υλοποίησης σε ένα νήμα και στη συνέχεια αυξήσαμε σταδιακά τον αριθμό των νημάτων. Παρατηρήσαμε ότι το ποσοστό recall παρέμενε σταθερό, με τη μόνη διαφοροποίηση να αφορά τους χρόνους εκτέλεσης. Αυτό αποτελεί μια ισχυρή ένδειξη ότι δεν υπάρχουν προβλήματα συγχρονισμού (race conditions) και ανεπιθύμητων επανεγγραφών δεδομένων (data overwrites).

Άλλωστε, κάθε νήμα γράφει σε ξεχωριστές θέσεις μνήμης, αποφεύγοντας με αυτό τον τρόπο την ανάγκη για συγχρονισμό με locks ή mutexes και καθιστώντας συνάμα την υλοποίηση απλούστερη και πιο αποδοτική. Με χρήση του *ThreadSanitizer* (TSan) επιβεβαιώσαμε τις παραπάνω ενδείξεις.

Γενικά

Δεδομένα

Πλήθος σημείων : 10^3 – 10^6 | Διαστάσεις : 10^1 – 10^3 | Πλήθος γειτόνων (k) : 2^4 – 2^7

Λήφθηκαν από εδώ[1] και συγκεκριμένα είναι τα :

fashion-mnist-784 ➔ 70000 σημεία, 784 διαστάσεις, k = 100 γείτονες, sparse

sift-128 (το 1/8 του συνολικού) ➔126500 σημεία, 128 διαστάσεις, k = 100 γείτονες

Preprocessing step

Τα δεδομένα έχουν επέκταση αρχείου : hdf5 και καθένα περιέχει το “test” και το “train” dataset. Στα benchmarks χρησιμοποιούμε την ένωση αυτόν των δύο συνόλων.

Βελτιστοποιήσεις

Μπορούμε να ελέγχουμε το tradeoff ακρίβειας – ταχύτητας μεταβάλλοντας τις παραμέτρους MIN_SIZE, sample_size (μέσω του samping_reduction) και N (μέσω του candidate_reduction). Συγκεκριμένα, το MIN_SIZE καθορίζει το ελάχιστο μέγεθος των υποσυνόλων για επεξεργασία, δηλαδή το base case της αναδρομής. Μεγαλύτερες τιμές συνεπάγονται περισσότερους υπολογισμούς και άρα μειωμένο ρυθμό επεξεργασίας αλλά υψηλότερη ακρίβεια. Όσον αφορά το samping_reduction, καθορίζει το πλήθος των δειγμάτων που θα λάβουμε. Καθώς αυξάνεται, μειώνονται τα δείγματα οπότε έχουμε λιγότερη ακρίβεια και μεγαλύτερη ταχύτητα. Ομοίως, το candidate_reduction επηρεάζει τον αριθμό των δειγματοληπτημένων σημείων προς εξέταση. Για υψηλότερες τιμές, λαμβάνουμε υπόψη λιγότερα σημεία και άρα έχουμε καλύτερο ρυθμό επεξεργασίας αλλά χειρότερη ακρίβεια. Κάνοντας *fine-tuning* στις παραμέτρους, μπορούμε να βρούμε την χρυσή τομή στον παραπάνω συμβιβασμό.

Επίσης, χρησιμοποιήθηκε το εργαλείο *perf* για τον εντοπισμό των cache misses. Με βάση τα αποτελέσματα, εισάγαμε οδηγίες prefetching για δεδομένα που πρόκειται να χρησιμοποιηθούν σε επόμενες επαναλήψεις, μειώνοντας τα cache misses (και χρόνο φόρτωσης από τη μνήμη).

Τέλος, για την αποθήκευση των σημείων του *query-set*, χρησιμοποιήσαμε το προσδιοριστικό τύπου *alignas*. Η χρήση του εξασφαλίζει ότι οι πίνακες δεδομένων είναι ευθυγραμμισμένοι στη μνήμη, σύμφωνα με τις απαιτήσεις της CPU για SIMD (Single Instruction - Multiple Data) εντολές. Αυτό βελτιώνει την αποδοτικότητα, επιτρέποντας στις εντολές της OpenBLAS να επεξεργάζονται τα δεδομένα ταχύτερα και πιο αποδοτικά.

Υπολογιστικό σύστημα

Operating system : Ubuntu 24.04
Processor: AMD Ryzen 5 5600H, 3301 MHz, 6 Cores, 12 Logical Processors
Cache size: L1 = 384KB | L2 = 3MB | L3 = 16MB
Total Physical Memory (usable) : 13,9 GB

[1] https://github.com/erikbern/ann-benchmarks?tab=readme-ov-file#data-sets

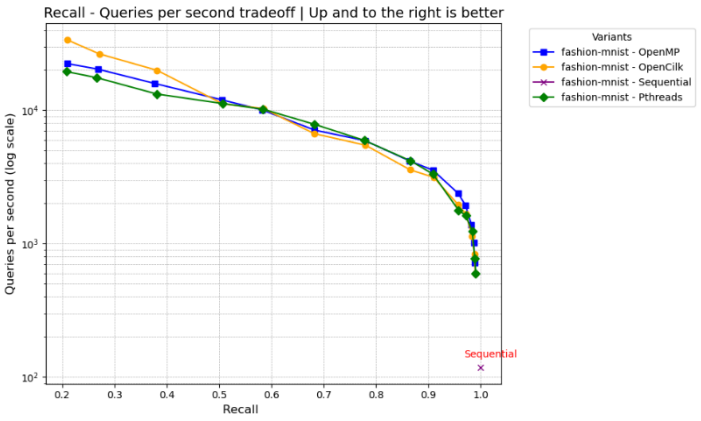
Benchmarks

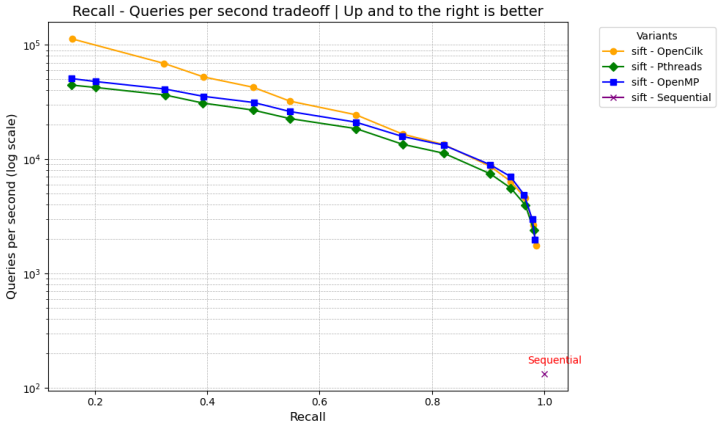
Tradeoff Recall – Queries per second

Στην ανάλυση recall – queries per second παρατηρούμε τα εξής:

- Για μεγάλες τιμές recall (>95%), όλες οι υλοποιήσεις έχουν παρόμοια απόδοση.
- Για χαμηλότερα recall (<50% για το Fashion-MNIST και <80% για το SIFT), η OpenCilk υπερέχει, διότι έχει χαμηλότερο overhead για δημιουργία threads, καθιστώντας έτσι την πιο αποδοτική για σύντομες εκτελέσεις με ελάχιστους υπολογισμούς.

Παρακάτω παρουσιάζεται ένα διάγραμμα με την απόδοση recall – queries per second για κάθε dataset:

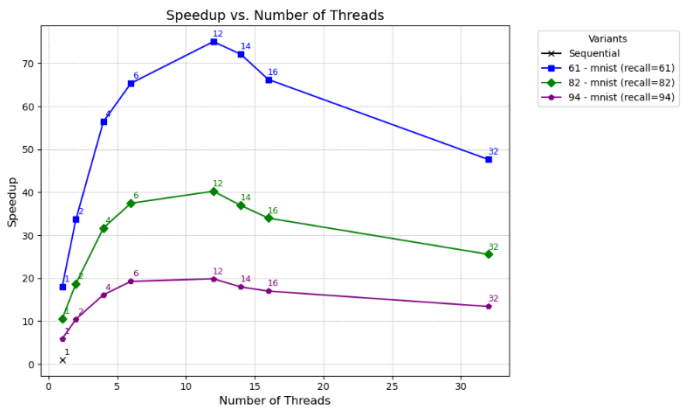


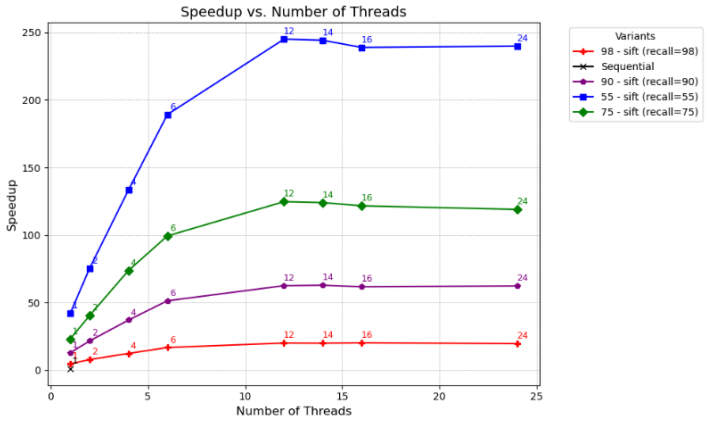


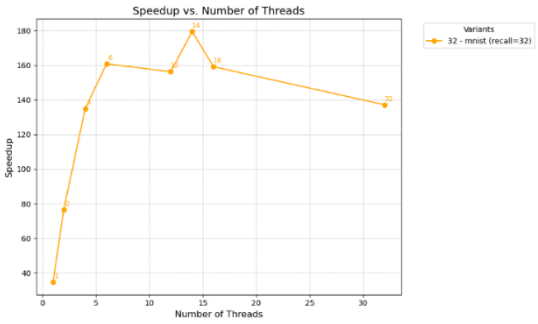
Speedup σε σχέση με τη σειριακή εκτέλεση

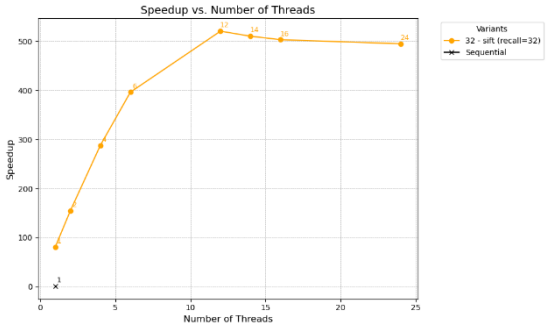
Το speedup συγκριτικά με τη σειριακή εκτέλεση παρουσιάζει τα εξής χαρακτηριστικά:

- Όλες οι υλοποιήσεις δείχνουν παρόμοιο speedup μέχρι τον αριθμό των λογικών επεξεργαστών του συστήματος (12 threads). Παρακάτω απεικονίζεται για την υλοποίηση με OpenCilk.
- Η OpenCilk υπερέχει στη διαχείριση περισσότερων threads (>12), παρουσιάζοντας γραμμική πτώση απόδοσης αντί για εκθετική, όπως γίνεται στις υλοποιήσεις OpenMP και Pthreads. Αυτό οφείλεται στο μηχανισμό work stealing, ο οποίος εξισορροπεί δυναμικά τον φόρτο εργασίας.
- Δεν παρατηρήθηκε βελτίωση για 14 threads (πέρα από το dataset : mnist με recall = 32) όπως πιθανόν να αναμέναμε.









Τέλος, για ένα ικανοποιητικό recall (>75), επιτυγχάνουμε έως και 125 φορές γρηγορότερη εκτέλεση από τη σειριακή υλοποίηση για το SIFT dataset. Αυτή η βελτίωση οφείλεται στη δραστική μείωση των απαιτούμενων υπολογισμών για προσεγγιστικούς γείτονες. Η απόδοση αυτή βελτιώνεται περαιτέρω καθώς αυξάνεται το μέγεθος του dataset.

Κώδικας

Ο κώδικας μπορεί να βρεθεί στο GitHub[2].

[2] https://github.com/fraidakis/PDS_Ex1