Παράλληλα και Διανεμημένα συστήματα

1η Εργασία

Παπαγεωργίου Νικόλαος, 9425, nikolaospg@ece.auth.gr

Τσουμπλέκας Γεώργιος, 9359, gktsoump@ece.auth.gr

6/12/2020

0. Γενικά

Η παρούσα εργασία αποτελεί μια εφαρμογή του παραλληλισμού σε συστήματα κοινής μνήμης. Πιο συγκεκριμένα δεδομένου ενός αραιού πίνακα συσχέτισης A, καλούμαστε να υπολογίσουμε τον αριθμό τριγώνων που σχηματίζονται από τους κόμβους του δικτύου που αναπαρίσταται από τον A. Ο υπολογισμός αυτών γίνεται μέσω 2 μεθόδων, ενός πιο εξαντλητικού και ενός που εκμεταλλεύεται τις ιδιότητες του A. Για τον παραλληλισμό των μεθόδων έγινε χρήση της βιβλιοθήκης pthreads, του openMP και του openCilk.

1. Το αρχείο test.c

Στο αρχείο αυτό περιέχονται όλες οι συναρτήσεις που υλοποιούμε για την μετατροπή ενός .mtx αρχείου που περιέχει έναν αραιό πίνακα στην ισοδύναμη csc μορφή του. Περιλαμβάνει την δομή στην οποία αποθηκεύουμε τα στοιχεία που χρειαζόμαστε για την αναπαράσταση του πίνακα σε csc format. Επιπλέον, περιλαμβάνεται και η μέθοδος η οποία είναι υπεύθυνη για την μετατροπή του πίνακα από coo format (στο οποίο είναι μέσα στο .mtx αρχείο) σε csc format. Σημείωση ότι επειδή οι πίνακες με τους οποίους εργαζόμαστε είναι συμμετρικοί, στο .mtx αρχείο περιλαμβάνονται μόνο τα στοιχεία του κάτω τριγωνικού πίνακα που προκύπτει από τον αραιό πίνακα. Η συνάρτηση εκμεταλλεύεται αυτή τη συμμετρία για να δημιουργήσει από τα συμμετρικά στοιχεία και τον άνω τριγωνικό πίνακα και στη συνέχεια συμπτύσσει τους δύο αυτούς πίνακες κατάλληλα για να προκύψει το csc format ολόκληρου του πίνακα.

2. Το αρχείο tester.c

Το αρχείο αυτό ουσιαστικά περιλαμβάνει τη μέθοδο με την οποία γίνεται ο έλεγχος ορθότητας του υπολογισμού των τριγώνων. Ουσιαστικά, υπολογίζουμε το διάνυσμα που περιλαμβάνει τον αριθμό των τριγώνων που συμμετέχει κάθε κόμβος μέσω μιας μεθόδου η οποία είναι επιβεβαιωμένα ορθή. Στην συνέχεια, στην συνάρτηση γίνεται σύγκριση στοιχείο-στοιχείο αυτού του διανύσματος με το διάνυσμα που προέκυψε από το πρόγραμμά μας (σειριακό ή παράλληλο) και αν είναι ίδια τότε μπορούμε να είμαστε σίγουροι ότι το πρόγραμμά μας παράγει σωστά αποτελέσματα.

3. Η μέθοδος V3

Για την υλοποίηση του V3 ακολουθήθηκε η παρακάτω λογική: Για να έχουμε τρίγωνο στον κόμβο i πρέπει να υπάρχουν j,k ώστε A[i.j]=A[j,k]=A[k,i]=1 (1). Εκμεταλλευόμενοι την ύπαρξη συμμετρίας και ξέροντας ότι σε CSC είναι πιο εύκολο να ψάχνουμε τις στήλες, για κάθε στήλη, παίρνουμε κάθε ζευγάρι μη μηδενικών στοιχείων αυτής, έστω A[j,i] και A[k,i] για κάποια στήλη i και ελέγχουμε αν το στοιχείο A[j,k] είναι μη

μηδενικό. Η αναζήτηση γίνεται με binary search στα στοιχεία της στήλης που μας ενδιαφέρει για γρηγορότερα αποτελέσματα. Γίνεται αντιληπτό ότι λόγω της συμμετρίας του πίνακα δεν χρειάζεται να γίνει η αντίστοιχη διαδικασία και για τα στοιχεία που προκύπτουν από τα παραπάνω με αντιστροφή των δεικτών λόγω της συμμετρίας του πίνακα. Έτσι, μειώνουμε στο 1/6 τον χρόνο υπολογισμού αφού φροντίζοντας να ισχύει k<j<i υπολογίζουμε ένα δεδομένο τρίγωνο μόνο 1 φορά αντί για τις 6 της εξαντλητικής μεθόδου. Στην παράλληλη υλοποίηση του αλγορίθμου αυτού, αξίζει να σημειωθεί ότι κάθε νήμα εκπονεί την παραπάνω διαδικασία για μια συγκεκριμένη στήλη του πίνακα Α. Όμως, κάθε στήλη στον πίνακα γειτνίασης αναφέρεται και σε έναν ξεχωριστό κόμβο και κάθε νήμα υπολογίζει τα τρίγωνα για ξεχωριστούς κόμβους (άρα και ξεχωριστές θέσεις στον πίνακα τριγώνων). Έτσι, αποφεύγουμε τα data races και δεν υπάρχει ανάγκη χρήσης mutex που θα έκανε τον κώδικά μας πιο αργό.

4. Η μέθοδος V4

Η υλοποίηση της μεθόδου αυτής εκμεταλλεύεται την ιδιότητα των πινάκων γειτνίασης βάσει της οποίας το τετράγωνο του πίνακα περιλαμβάνει τα μονοπάτια μήκους 2 μεταξύ 2 κόμβων. Προκειμένου να αποφύγουμε τον αναλυτικό υπολογισμό του AxA βρίσκουμε τα στοιχεία του μόνο στις θέσεις εκείνες που το Α έχει μη μηδενικά στοιχεία. Για την εκτέλεση του πολλαπλασιασμού εκμεταλλευόμαστε την συμμετρία του Α και αντί να πολ/σουμε την γραμμή i με την στήλη j πολ/ζουμε την στήλη i με την στήλη j. Να σημειωθεί εδώ ότι σε CSC, το εσωτερικό γινόμενο για 2 στήλες είναι απλά το άθροισμα των ζευγών από στοιχεία που είναι σε ίδια γραμμή. Όμως για να αποφύγουμε και πάλι τον αναλυτικό υπολογισμό αυτού του πίνακα C = A.*(AxA) παρατηρούμε ότι ο πολ/σμός αυτού με ένα διάνυσμα μοναδιαίων στοιχείων αντιστοιχεί στη δημιουργία ενός διανύσματος όπου κάθε στοιχείο είναι το άθροισμα των στοιχείων μιας στήλης του C. Έτσι, αφού οι αρχικές μας πράξεις γίνονται στήλη-στήλη στον Α, πριν αλλάξουμε στήλη που μελετάμε, απλά προσθέτουμε όλα τα στοιχεία της στήλης που δημιουργήσαμε και δημιουργούμε ένα-ένα τα στοιχεία του τελικού διανύσματος τριγώνων. Μάλιστα, στην παράλληλη υλοποίηση του αλγόριθμου, κάθε νήμα ασχολείται και με μια ξεχωριστή στήλη του Α, επομένως και ένα ξεχωριστό κελί του τελικού διανύσματος τριγώνων. Έτσι, αποφεύγουμε τα data races και δεν υπάρχει ανάγκη χρήσης mutex που θα έκανε τον κώδικά μας πιο αργό. Η μέθοδος αυτή εμφανίζει μικρότερη πολυπλοκότητα από την V3 καθότι αποφεύγει τη χρήση τριπλού εμφωλευμένου loop.

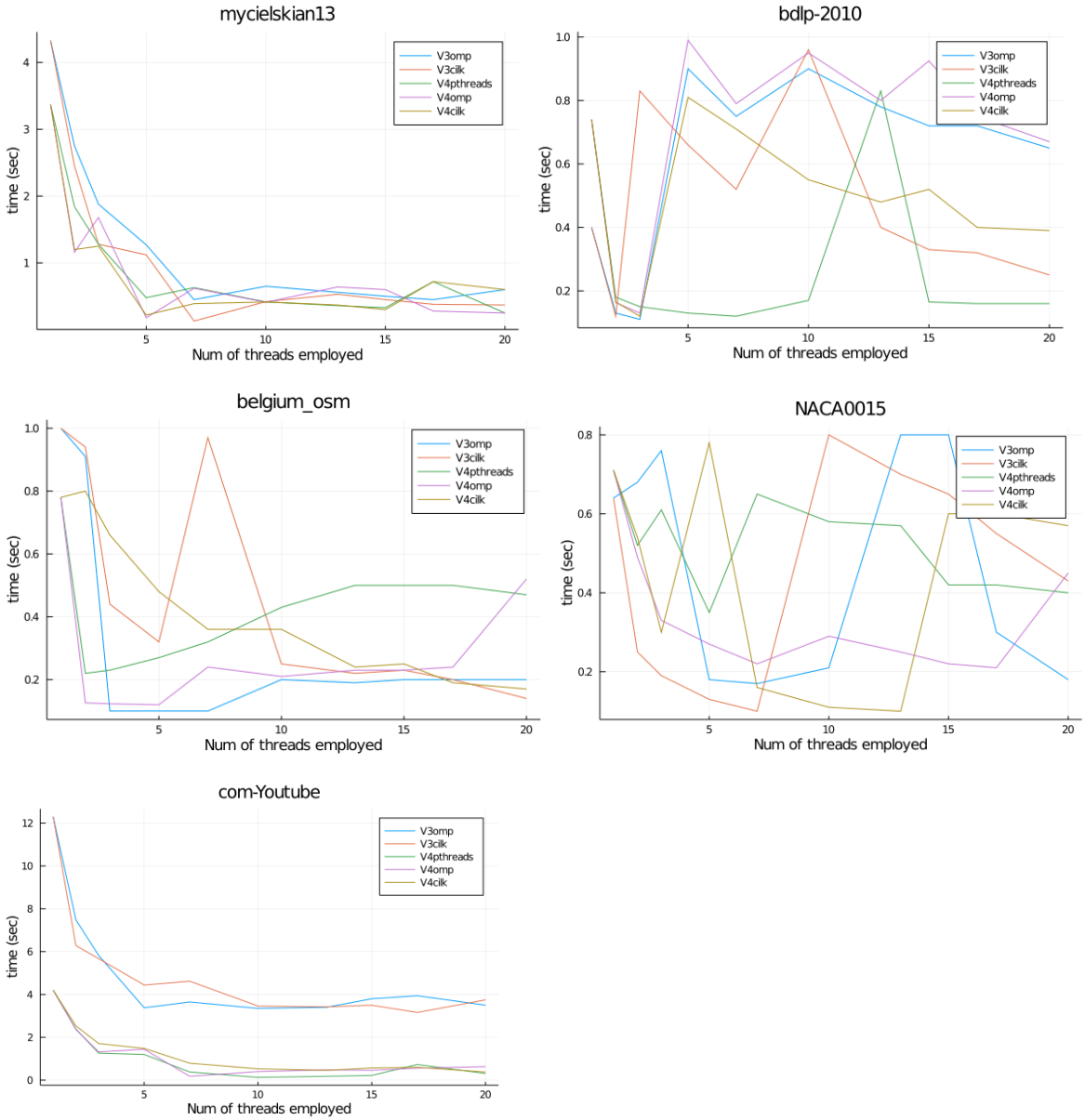
Σημείωση: Για περισσότερες λεπτομέρειες και τεχνικές λεπτομέρειες σχετικά με την υλοποίηση όλων των παραπάνω μεθόδων που αναφέρθηκαν ανατρέξτε στα σχόλια του κώδικα.

5. Πειραματικά αποτελέσματα

Μελετώντας τα διαγράμματα που παρουσιάζονται, μπορούμε να βγάλουμε κάποια ενδιαφέροντα συμπεράσματα: Βλέποντας τους σχετικά μικρούς πίνακες (bdlp-2010, NACA0015), παρατηρούμε πως υπάρχει μια σχετικά ακανόνιστη συμπεριφορά των γραμμών των διαφόρων υλοποιήσεων σε σχέση με τον χρόνο και δεν μπορούμε ξεκάθαρα να δούμε τις τυχούσες ιδιαιτερότητες/διαφορές τους (δεν τείνουν ξεκάθαρα σε κάποια τιμή και εμφανίζουν μεγάλες διακυμάνσεις). Αυτή η μεγάλη “χαοτικότητα” οφείλεται στο ότι οι πίνακες είναι πολύ μικροί, και έτσι τον κύριο ρόλο στον καθορισμό της απόκρισης των διαφόρων υλοποιήσεων παίζει το πως θα τύχει να γίνει η επεξεργασία, και όχι η εκάστοτε υλοποίηση (εφόσον έτσι και αλλιώς δεν έχουν εξαιρετικά διαφορετικές πολυπλοκότητες). Βλέποντας όμως τους μεγαλύτερους πίνακες (πχ com-Youtube), μπορούμε πλέον να διαχωρίσουμε τις υλοποιήσεις. Αυτές που βασίστηκαν στον αλγόριθμο του V4 βλέπουμε πως παρουσιάζουν ξεκάθαρα μικρότερους χρόνους για οποιονδήποτε αριθμό

νημάτων που αποφασίζουμε να στρατεύσουμε. Αυτό οφείλεται στο γεγονός ότι το μέγεθος είναι αρκετά μεγάλο, ώστε τον κύριο ρολό στην ρύθμιση της απόκρισης να παίζει η «χαμηλότερη», έστω και όχι σε τρομακτικό βαθμό, υλοποίηση (πχ δημιουργία νημάτων και thread scheduling). Εάν έπρεπε να σχολιάσουμε τον καλύτερο τρόπο για παραλληλοποίηση βλέπουμε πως η διαφορές είναι αρκετά μικρές, αρά δεν βγαίνει κάποιο συμπέρασμα που καθιστά κάποιο εκ των pthreads, openMp και openCilk νικητή. Όμως, είναι ξεκάθαρο ότι σε μεγάλους πίνακες και για μικρό αριθμό νημάτων η V4 υπερτερείς σημαντικά της V3. Τέλος, ένα ακόμη ενδιαφέρον σχόλιο που θα έκανε κανείς είναι ότι από ένα σημείο και μετά, ακόμη και εάν αυξάνουμε τον αριθμό των νημάτων, ο χρόνος δεν μειώνεται περαιτέρω, ενώ μάλιστα, υπάρχει πιθανότητα κάποιος να καταφέρει μικρότερους χρόνους αξιοποιώντας ίσως σχετικά λιγότερα νήματα.

Ακολουθούν τα διαγράμματα χρόνου εκτέλεσης συναρτήσει του πλήθους νημάτων για κάθε υλοποίηση σε διάφορους πίνακες:



Σημείωση: Οι υλοποίηση της εργασίας μπορεί να βρεθεί στα ακόλουθα links

1) https://github.com/GeorgeTsoumplekas/Parallel-Systems-Exercise-1
2) https://github.com/nikolaospg/PDS-Exercise1