Κωνσταντίνος Πράττης Δεκέμβριος 2022

Παράλληλα και Διανεμημένα Συστήματα, Αναφορά 1ης Εργασίας.

ΕΥΡΕΣΗ ΤΩΝ "STRONGLY CONNECTED COMPONENTS" ΚΑΤΕΥΘΥΝΟΜΕΝΩΝ ΓΡΑΦΗΜΑΤΩΝ ΣΕ ΠΟΛΥΠΥΡΗΝΑ ΣΥΣΤΗΜΑΤΑ ΚΟΙΝΗΣ ΜΝΗΜΗΣ

• Εισαγωγή

Σε αυτή την εργασία ζητείται η δημιουργία ενός παράλληλου αλγορίθμου για την εύρεση των Strongly Connected Components (SCC) κατευθυνόμενων γραφημάτων. Συγκεκριμένα, οι σειρακοί αλγόριθμοι των Tarjan και Kosaraju λύνουν το πρόβλημα με πολυπλοκότητα O(n + m) όπου n είναι ο αριθμός των κορυφών και m ο αριθμός των ακμών του γράφου. Όμως, επειδή βασίζονται σε DFS, δεν προσφέρονται προς παραλληλοποίηση. Για αυτό, χρησιμοποιείται ο αλγόριθμος χρωματισμού που χρησιμοποιεί BFS και έχει δυνατότητες παράλληλης υλοποίησης.

Σε αυτήν την αναφορά, αρχικά, γίνεται μία σύντομη περιγραφή της υλοποίησης του σειριακού αλγορίθμου «χρωματισμού» για την εύρεση των SCC και στη συνέχεια των παράλληλων παραλλαγών του με χρήση των βιβλιοθηκών **OpenCilk**, **OpenMP** και **Pthreads**. Έπειτα, αφού παρουσιάζονται αναλυτικοί πίνακες και γραφήματα για τη σύγκριση των διάφορων μεθόδων, παρατίθενται τα συμπεράσματα που προέκυψαν καθώς και προτάσεις για περαιτέρω βελτίωση. Τέλος, στην αναφορά αυτή δεν γίνεται σύγκριση της καλύτερης παράλληλης υλοποίησης από τις παραπάνω με την καλύτερη σειριακή (Tarjan/Kosaraju).

• Σειριακός Αλγόριθμος Χρωματισμού

Ο αλγόριθμος υλοποιήθηκε σε γλώσσα C. Μερικές διευκρινίσεις για τις δομές δεδομένων δίνονται παρακάτω:

- Για την αναπαράσταση του πίνακα γειτνίασης χρησιμοποιήθηκαν οι μορφές Compressed Sparse Row (CSR) και Compressed Sparse Column (CSC). Οι τελευταίες υλοποιήθηκαν αμφότερες από μία δομή struct που περιέχει ένα διάνυσμα ptr και ένα διάνυσμα ind. Το διάνυσμα ptr έχει μήκος n + 1, όπου n ο αριθμός των κόμβων και το ind έχει μήκος nnz, δηλαδή όσες και οι ακμές ή τα μη μηδενικά στοιχεία του πίνακα γειτνίασης. Για παράδειγμα, στη μορφή CSC, το διάνυσμα ind, περιέχει τους δείκτες της στήλης των μη μηδενικών στοιχείων και το ptr περιέχει για κάθε γραμμή από ποιον δείκτη του ind ξεκινάει (αντίστοιχα για CSR).
- Η ανάγνωση του πίνακα προϋποθέτει τύπο αρχείου .mtx. Έγινε χρήση και της βιβλιοθήκης mmio.

- Όλες οι πληροφορίες του γραφήματος (όπως ο πίνακας χρωμάτων, το μέγεθος, η μορφή csr/csc , κ.α.), αποθηκεύονται σε ένα struct για πιο εύκολη πρόσβαση.
- Για να μην δημιουργείται περαιτέρω καθυστέρηση, αντί να αφαιρούνται οι κόμβοι που επιλέγονται, διατηρείται ένα διάνυσμα removed με flag για κάθε κόμβο: 0 αν ο κόμβος είναι ενεργός ή 1 αν έχει αφαιρεθεί.

Όσον αφορά την εκτέλεση του αλγορίθμου, αξίζει να σημειωθούν τα εξής:

- Πριν την έναρξη του αλγορίθμου, πραγματοποιείται trimming των trivial scc μία φορά, δηλαδή αφαιρούνται οι κόμβοι που στο αρχικό γράφημα έχουν έσω ή έξω βαθμό 0 ή έχουν βαθμό 1 αλλά κάνουν selfloop. Η διαδικασία αυτή, αν και θα μπορούσε να γίνει αναδρομικά ή σε κάθε επανάληψη παρατηρήθηκε ότι επιβαρύνει τον χρόνο του προγράμματος.
- Τα χρώματα αρχικοποιούνται σε κάθε επανάληψη στο αρχικό id του κόμβου.
- Ο χρωματισμός των κόμβων γίνεται με BFS με πολλαπλασιασμό αραιού πίνακα με διάνυσμα και με χρήση της μάσκας removed. Σε κάθε επανάληψη κάθε κόμβος επιλέγει το ελάχιστο χρώμα από τους πατέρες του.
- Στη συνέχεια, βρίσκονται τα διαφορετικά χρώματα που έχουν «διαδοθεί» μετά τη λήξη του χρωματισμού. Έτσι, σε καθένα από αυτά ανατίθεται το id του scc που σχηματίζουν. Στη συνέχεια, γίνεται BFS με πολλαπλές ρίζες, ξεκινώντας δηλαδή από κάθε κόμβο του οποίου το αρχικό χρώμα είναι ένα από αυτά που έχουν παραμείνει στο γράφημα και βρίσκοντας όλους τα παιδιά του που έχουν το ίδιο χρώμα. Η διαδικασία επαναλαμβάνεται ανανεώνοντας το frontier και προχωρώντας ένα επίπεδο κάθε φορά.

Η πολυπλοκότητα του αλγορίθμου είναι της τάξης του O(d*m) για κάθε επανάληψη, όπου m = nnz είναι ο αριθμός των ακμών και d η διάμετρος του γράφου.

• Παράλληλη Υλοποίηση

Θεωρητικά, εάν είχαμε διαθέσιμους άπειρους επεξεργαστές, θα μπορούσε να γίνει παράλληλα ο χρωματισμός αλλά και η εύρεση των predecessors. Δηλαδή, όλοι οι κόμβοι ταυτόχρονα θα μπορούσαν να στείλουν το χρώμα στους γείτονές τους σε O(1) και μετά να κρατήσουν το ελάχιστο που τους ήρθε. Στη χειρότερη περίπτωση το χρώμα θα διανύσει και εδώ το μονοπάτι μήκους d όμως κάθε βήμα θα γίνεται σε O(1). Έτσι, συνολικά ο χρωματισμός θα μπορεί να γίνει σε O(d). Η εύρεση των predecessors με παρόμοιο τρόπο θα μπορούσε να γίνει σε O(d).
Συνολικά, στην ιδανική περίπτωση ο χρόνος θα είναι της τάξης O(d) ανά επανάληψη.
Βέβαια, αφού έχουμε p και όχι άπειρους επεξεργαστές, ο χρόνος θα είναι της τάξης O(m*d/p) με μία ιδανική υλοποίηση.

▪ OpenCilk

Δεν πραγματοποιήθηκαν πολλές αλλαγές σε σχέση με τον σειριακό. Με χρήση cilk for στην αρχικοποίηση, στο διαμοιρασμό χρωμάτων και στην εύρεση των predecessors πραγματοποιήθηκε παράλληλη υλοποίηση του αλγορίθμου.

Στο BFS που σπρώχνονται τα χρώματα, παρατηρείται ένα race condition, το οποίο, αν και δεν θα έπρεπε να επηρεάζει το αποτέλεσμα, καθώς πρόκειται για ένα flag, πιθανώς λόγω βελτιστοποίησης του compiler το επηρεάζει. Για αυτό, προστέθηκε και ένας πίνακας με n = (αριθμός των workers θέσεις), ώστε κάθε worker της cilk να γράφει στη δική του θέση την τιμή που υπολογίζει για το flag και στη συνέχεια να γίνεται μία συνένωση αυτών.

Να σημειωθεί, ότι τα races που εμφανίζονται όσον αφορά στο με ποια σειρά γίνονται οι χρωματισμοί, δεν μας επηρεάζει, καθώς όπως και να έχει κάθε κόμβος θα πάρει μικρότερο χρώμα από το προηγούμενο και τα χρώματα θα συγκλίνουν, ίσως απλά σε περισσότερα βήματα. Παρόμοια, το με ποια σειρά θα προστεθούν ή θα αφαιρεθούν κόμβοι από το frontier στην εύρεση των predecessors, πάλι επηρεάζει μόνο τον αριθμό των βημάτων και όχι το αποτέλεσμα. Τέλος, επειδή έγινε υπολογισμός από πριν του scc id κάθε χρώματος δεν χρησιμοποιείται κάποιος reducer ή mutex lock.

- **OpenMP**

  Παρόμοια με την OpenCilk. Δεν χρησιμοποιείται πίνακας για το flag στη διάδοση των χρωμάτων. Αντί για cilk for χρησιμοποιείται #pragma parallel for. Εδώ έγινε παραλληλοποίηση και της for που υπολογίζει τον αριθμό των μοναδικών χρωμάτων μετά τον χρωματισμό, αλλά για να αποφευχθεί αλλοίωση αποτελεσμάτων η εντολή αύξησης της αντίστοιχης μεταβλητής μπήκε σε ένα μπλοκ atomic.

- **Pthreads**

  Εδώ χρειάστηκε η αλλαγή αρκετών συναρτήσεων, ώστε να ταιριάζουν στη μορφή void * f (void *). Προστέθηκαν και καινούριες συναρτήσεις για να παραλληλοποιηθεί η αρχικοποίηση και η εύρεση των μοναδικών χρωμάτων. Η προσέγγιση είναι κάθε thread να παίρνει ένα κομμάτι των for, ξεκινώντας από την επανάληψη thread Id και συνεχίζοντας μέχρι το τέλος με βήμα NTHREADS, που είναι ο αριθμός των συνολικών threads. Εδώ, η μεταβλητή που κρατάει τον αριθμό των μοναδικών χρωμάτων κλειδώθηκε με χρήση mutex lock. Τα υπόλοιπα race conditions που εμφανίζονται δεν επηρεάζουν το αποτέλεσμα του αλγορίθμου.

Μετρήσεις – Σύγκριση Μεθόδων

Για την εξαγωγή συμπερασμάτων χρησιμοποιήθηκαν σχεδόν όλα από τα 15 γραφήματα που υπάρχουν στην εκφώνηση της άσκησης, εκτός από τα 3 τελευταία (LAW/Arabic-2005, LAW/uk-2005, SNAP/twitter7), για τα οποία δεν υπάρχουν μετρήσεις αφού δεν χωρούσαν στην μνήμη του υπολογιστή που χρησιμοποιήθηκε. Τα αρχεία «.mtx» αντλήθηκαν από το https://sparse.tamu.edu/.

Οι πίνακες (.csv) και τα σχετικά διαγράμματα (στον φάκελο plot) για λόγους ευκρίνειας υπάρχουν σε ξεχωριστά αρχεία του Repository.

- Όπως παρατηρούμε από το διάγραμμα με το speedup, όπου τα γραφήματα είναι ταξινομημένα με βάση τον αριθμό των SCC, όσο μεγαλώνει το μέγεθος του γραφήματος και συγκεκριμένα ο αριθμός των ακμών του, αφενός αυξάνεται ο χρόνος που χρειάζεται ο αλγόριθμος για να ολοκληρωθεί, αφετέρου αυξάνεται και το σχετικό speedup που επιτυγχάνουμε με τις παράλληλες υλοποιήσεις.
- Σε μεγάλους γράφους, πιο γρήγορη φαίνεται να είναι η υλοποίηση της OpenCilk με μικρή διαφορά από την OpenMP και στη συνέχεια με μεγαλύτερη διαφορά (πιθανώς λόγω ατελούς υλοποίησης) αυτή των pthreads, που όμως παραμένει ταχύτερη του σειριακού. Σε μικρούς γράφους, ο σειριακός αλγόριθμος είναι ταχύτερος ή ελάχιστα αργότερος από τους παράλληλους.
- Όπως αναμέναμε, ο χρόνος εκτέλεσης αυξάνεται όσο αυξάνεται ο αριθμός των ακμών, ενώ ο αριθμός των κόμβων δεν φαίνεται να επηρεάζει με κάποια προφανή σχέση, πέρα του ότι με την αύξηση των κόμβων πιθανώς αυξάνονται και οι ακμές των γραφημάτων.
- Τέλος, όσον αφορά τον αριθμό των threads, αυτός φαίνεται να πετυχαίνει την βέλτιστη μείωση του χρόνου όταν επιλέγεται κοντά στον αριθμό των φυσικών πυρήνων του συστήματος (6 cores-12 logical processors) τόσο για τα pthreads όσο και για την OpenCilk. Για την OpenMP, φαίνεται να μην ισχύει το ίδιο για τον συγκεκριμένο γράφο που παρουσιάζεται το σχετικό διάγραμμα, αλλά σε άλλα γραφήματα καταλήγουμε σε παρόμοιο συμπέρασμα (Ο σχετικός πίνακας υπάρχει σαν αρχείο .csv μέσα στο Repository).

• <u>Συμπεράσματα-Αξιολόγηση Αποτελεσμάτων</u>

Οι υλοποιήσεις που πραγματοποιήθηκαν φαίνεται να βελτιώνουν τον σειριακό αλγόριθμο χρωματισμού. Βέβαια, με τη χρήση pthreads θεωρητικά θα μπορούσε να γίνει ταχύτερα ο υπολογισμός των SCC από ότι παρουσιάζεται σε αυτήν την εργασία. Πιθανές αλλαγές για καλύτερα αποτελέσματα σε αυτό το κομμάτι θα μπορούσαν να είναι η χρήση **cond signals** ή η εφαρμογή του μοντέλου consumer-producer. Το τελευταίο θα εξυπηρετούσε το πρόβλημα των κόμβων που γίνονται masked out, καθώς αυτοί, αν και δεν υπολογίζονται στη συνέχεια του αλγορίθμου, μοιράζονται στα threads, με αποτέλεσμα κάποια νήματα να έχουν λιγότερη δουλειά από άλλα και ο αλγόριθμος να χάνει τον παραλληλισμό του καθώς μειώνονται οι κόμβοι.