Παράλληλα και Διανεμημένα Συστήματα

Καράμπελας Σάββας, ΑΕΜ: 9005

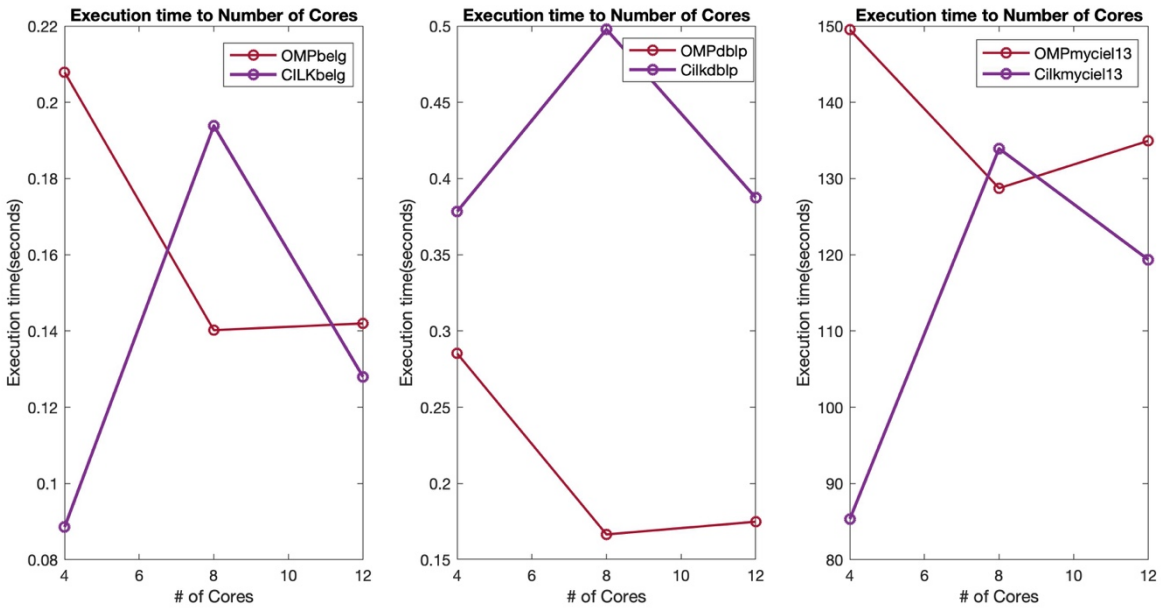
Καράμπελας Νίκος, ΑΕΜ: 8385

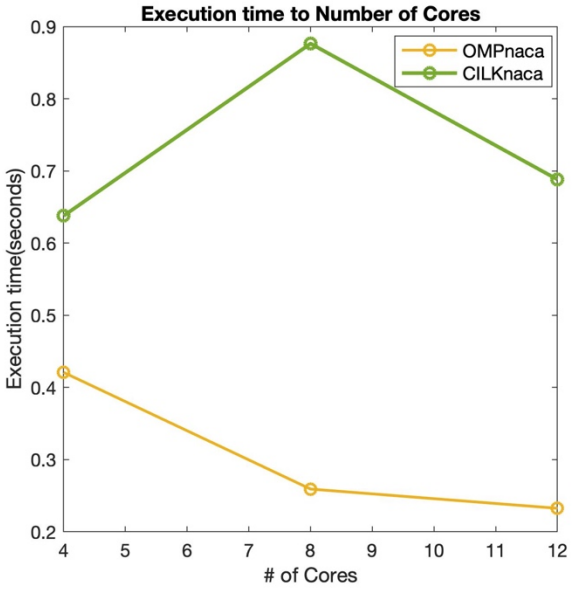
Github: https://github.com/nikoskar/Parallel-and-Distributed-Systems-AUTH

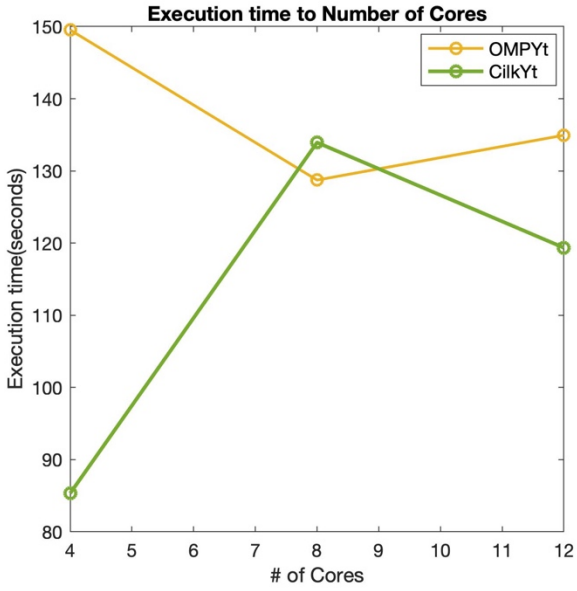
Περιγραφή της υλοποίησης V3

Διαβάζουμε σαν είσοδο έναν unweighted, undirected random γράφο από το αρχείο mtx και βρίσκουμε τα τρίγωνα με ένα τριπλό βρόγχο περιορίζοντας τα κελιά τα οποία ελέγχουμε ακολουθώντας την εξής συνθήκη i < j < k. Αρχικά μετατρέπεται ο πίνακας από COO σε CSC, με τον οποίο δουλεύουμε στη συνέχεια. Κατά την καταμέτρηση των τριγώνων χρησιμοποιούμε τις σχέσεις που συνδέουν το και το csc row. Ο CSC col καθορίζει το indexing/slicing του csc row για να παίρνουμε για κάθε στήλη τα μη μηδενικά της στοιχεία μέσω του *csc_row[csc_col[j] : csc_col[j + 1]]* . Αυτή η σχέση μας δίνει τα indexes της στήλης j, στα οποία έχουμε τιμή 1. Χρησιμοποιώντας λοιπόν αυτό τον τύπο κάνουμε διατρέχουμε τον βρόγχο μόνο για τα μη μηδενικά στοιχεία της κάθε γραμμής που μας ενδιαφέρει, σε κάθε for. Στη συνέχεια και αφού ελέγχουμε ότι ο κώδικας του σειριακού μας προγράμματος επιστρέφει τον σωστό αριθμό τριγώνων τόσο για μικρά datasets όσο και για μεγάλα, προχωρούμε στην παραλληλοποίηση του κώδικα. Σκοπός μας είναι να βρούμε τα τμήματα του κώδικα τα οποία μπορούν να εκτελεστούν ανεξάρτητα και ταυτόχρονα με το υπόλοιπο πρόγραμμα. Ωστόσο, επιλέγουμε να παραλληλοποιήσουμε μόνο στα τμήματα τα οποία συμφέρει από άποψη απόδοσης, καθώς κάθε φορά που επιλέγεται μία τεχνική παραλληλοποίησης εισάγεται και ένα overhead. Παρακάτω έχουμε προσθέσει διαγράμματα που απεικονίζουν τις μειώσεις στους χρόνους εκτέλεσης του προγράμματος με χρήση της OpenMP και της OpenCilk. Στην περίπτωση της OpenMP θα ορίσουμε τη παράλληλη περιοχή (#pragma omp parallel) και στη συνέχει θα δηλώσουμε ότι θέλουμε να παραλληλοποιηθεί ο βρόγχος(#pragma omp for schedule(dynamic) private(sum)) με τον scheduler σε dynamic mode γιατί ο βρόγχος τρέχει και για πολλές επαναλήψεις και περιέχει και πολλές διεργασίες. Δηλώνουμε, τη μεταβλητή sum σαν private διότι διαμοιράζεται σε όλα τα threads και θέλουμε να αποφύγουμε τη περίπτωση του data racing(στο τέλος του βρόγχου το sum αυξάνεται). Τέλος, στους εμφωλευμένους βρόχους όπου έχουμε απλές διεργασίες ενημέρωσης(πχ i++) χρησιμοποιούμε μια high level synchronization εντολή (#pragma omp atomic).Στην OpenCilk χρησιμοποιούμε την εντολή cilk_for για τους κεντρικούς μας βρόγχους. Οτιδήποτε άλλο προσθέτει παραπάνω overhead απ' ότι όφελος στον χρόνο εκτέλεσης. Τα datasets που χρησιμοποιούνται τρέχουν με το σειριακό πρόγραμμα με χρόνους(σε seconds) :

{Belgium, dblp, myciel13, NACA, com-Youtube} = [0.12, 0.41, 0.42, 106.1, 292.2]







Περιγραφή της υλοποίησης V4

Στην υλοποίηση v4 διαβάζουμε το mtx αρχείο για να πάρουμε τον πίνακα COO και μετά να τον μετατρέψουμε σε CSR. Η μοναδική διαφορά σε αυτό το στάδιο είναι ότι διαβάζοντας τον πίνακα, και τοποθετώντας τα στοιχεία του (indexes γραμμών και στηλών -- COO format) στους πίνακες I και J, μετατρέπουμε τον πίνακα σε συμμετρικό και γεμίζουμε και την πάνω/κάτω πλευρά του. Αυτό γίνεται μέσα στο βρόγχο του scanf με τη χρήση των εντολών *I[nz + i] = J[i];* και *J[nz + i] = I[i];*. Ουσιαστικά παίρνουμε τα ζεύγη γραμμή - στήλη και τα αντιστρέφουμε. Στη συνέχεια στο σημείο που μετράμε τα τρίγωνα έχουμε να υπολογίσουμε τον τύπο : $c_3 = (A \odot (A\,A))\; e/2,$. Στον εν λόγω τύπο το γινόμενο του πίνακα A με τον εαυτό του, θεωρώντας ότι ο A είναι NxN, μας δίνει έναν πίνακα NxN. Ο πίνακας αυτός πολλαπλασιάζεται με τον A κατά στοιχείο(element-wise), οπότε το αποτέλεσμα είναι ένας αραιός(sparse) πίνακας με μη μηδενικά στοιχεία στα ίδια σημεία με τον A. Οπότε θα ήταν ύψιστη υπολογιστική σπατάλη και επιβάρυνση για την μνήμη να υπολογίσουμε ολόκληρο τον A*A, καθώς τα περισσότερα στοιχεία του μετά τον πολλαπλασιασμό Hadamard γίνονται 0. Για αυτό τον λόγο, όσον αφορά τον A*A, υπολογίσαμε μόνο τα στοιχεία τα οποία βρίσκονται σε indexes όπου ο A είναι μη μηδενικός. Δηλαδή, δεν χρειάζεται να κάνουμε element-wise πολλαπλασιασμό εφόσον η πράξη αποφεύγεται «αλγοριθμικά». Είναι σημαντικό να σημειωθεί ότι το αποτέλεσμα είναι επίσης σε μορφή CSC. Δημιουργήσαμε έναν πίνακα masked vals στον οποίο βάλαμε τα αποτελέσματα των πολλαπλασιασμών, και κρατήσαμε τους πίνακες csc_col και csc_row του A, για τον νέο πίνακα, καθώς δέν αλλάζουν(λόγω Hadamard). Αμέσως μετά δημιουργούμε ένα διάνυσμα (e_vector) Nx1 στο οποίο κάθε κελί έχει την τιμή 1. Πολλαπλασιάζουμε το διάνυσμα με τον πίνακα με τον εξής τρόπο. Για κάθε γραμμή του πίνακα βρίσκουμε τον αριθμό των μη μηδενικών στοιχείων και στον πολλαπλασιασμό συμμετέχουν μόνο αυτά. Τα αποτελέσματα αυτού του πολλαπλασιασμού αποθηκεύονται σε έναν πίνακα Nx1, τον result. Τέλος, στο διάνυσμα c3_vector που έχουμε δημιουργήσει, και το οποίο θα αποτελείται από τα τρίγωνα για κάθε node, αποθηκεύουμε τα αποτελέσματα του result, αφού τα διαιρέσουμε με το 2. Το άθροισμα όλων των στοιχείων του c3 μας δίνει ως αποτέλεσμα το συνολικό αριθμό τριγώνων που υπάρχουν στο γράφο. Στη συνέχεια και αφού ελέγχουμε ότι ο κώδικας του σειριακού μας προγράμματος επιστρέφει τον σωστό αριθμό τριγώνων τόσο για μικρά datasets όσο και για μεγάλα, προχωρούμε στην παραλληλοποίηση του κώδικα. Σκοπός μας είναι να βρούμε τα τμήματα του κώδικα τα οποία μπορούν να εκτελεστούν ανεξάρτητα και ταυτόχρονα με το υπόλοιπο πρόγραμμα. Ωστόσο, επιλέγουμε να παραλληλοποιήσουμε μόνο στα τμήματα τα οποία συμφέρει από άποψη απόδοσης, καθώς κάθε φορά που επιλέγεται μία τεχνική παραλληλοποίησης εισάγεται και ένα overhead.
Πιο συγκεκριμένα παρατηρούμε ότι το εξωτερικό μας for loop μπορεί να παραλληλοποιηθεί και με μεγάλο όφελος. Στην περίπτωση της OpenMP θα ορίσουμε τη παράλληλη περιοχή (#pragma omp parallel) και στη συνέχεια θα δηλώσουμε ότι θέλουμε να παραλληλοποιηθεί ο βρόγχος(#pragma omp for schedule(dynamic) private(sum)) με τον scheduler σε dynamic mode γιατί ο βρόγχος τρέχει και για πολλές επαναλήψεις και περιέχει πολλές διεργασίες. Για την OpenCilk χρησιμοποιούμε την εντολή cilk_for για τους δύο κεντρικούς

βρόγχους (γραμμής & στήλης) και παρατηρούμε σημαντική διαφορά στους χρόνους εκτέλεσης. Αν και υπάρχουν κάποια τμήματα του κώδικα που θα μπορούσαν να παραλληλοποιηθούν, προσθέτουν αρκετό overhead και επομένως τα αποφύγαμε. Τέλος, στο κομμάτι των pthreads ορίζουμε αρχικά μία συνάρτηση για τον υπολογισμό των τριγώνων. Η ιδέα είναι να μοιράσουμε τις επαναλήψεις του εξωτερικού βρόγχου for στα threads που θα ορίσουμε. Επίσης, δημιουργούμε ένα πίνακα από δομές (structs) και σε κάθε thread μέσω του id του περνάμε όλα τα απαραίτητα δεδομένα για το τμήμα του αλγορίθμου που θέλουμε να εκτελέσει. Πριν προχωρήσουμε στο πολλαπλασιασμό με το διάνυσμα, επιβεβαιώνουμε ότι όλα τα threads έχουν ολοκληρώσει τις εντολές που τους ανατέθηκαν. Παρακάτω έχουμε προσθέσει διαγράμματα που απεικονίζουν τις μειώσεις στους χρόνους εκτέλεσης του προγράμματος με χρήση της OpenMP,των Pthreads και της OpenCilk. Τα datasets που χρησιμοποιούνται τρέχουν με το σειριακό πρόγραμμα με χρόνους(σε seconds) :

{Belgium, dblp, myciel13, NACA, com-Youtube} = [0.69, 1.18, 17.03, 3.1, 66.57].