Παράλληλα και Διανεμημένα Συστήματα Εργασία 4 (Τελική).
Project 2: Sparse graph matrix reordering
Reverse Cuthill McKee Algorithm(RCM) in C and parallel(OpenMP)

Μιχάλης Καρατζάς AEM:9137 email:mikalaki@ece.auth.gr
Github link για το project: https://github.com/mikalaki/Reverse-Cuthill-McKee_C_parallel

1. Εισαγωγικά - Ο Αλγόριθμος

Ο Αλγόριθμος RCM αποτελεί έναν αλγόριθμο για την μετάθεση των στοιχείων ενός αραιού συμμετρικού πίνακα (πχ. ενός πίνακα γειτνίασης κόμβων), από την οποία προκύπτει ένας νέος ισοδύναμος(συμμετρικός) πίνακας μικρότερου bandwith, δηλαδή τα μη-μηδενικά στοιχεία του νέου πίνακα, βρίσκονται σε μια στενή διαγώνια ζώνη (band), γύρω από την κύρια διαγώνιο του πίνακα. Πίνακες με μικρό bandwith, εξασφαλίζουν μειωμένους χρόνους για την εφαρμογή διαφόρων πράξεων και διαδικασιών(πχ παραγοντοποιήσεις) πάνω σε αυτούς σε σχέση με πίνακες μεγαλύτερου bandwith και ίδιων διαστάσεων. Ο αλγόριθμος RCM σε βήματα έχει ως εξής:

- Αρχικοποιούμε μια άδεια **FIFO ουρά Q** και έναν άδειο **πίνακα για την αποθήκευση της σειράς των κόμβων στην μετάθεση R**, στον οποίο θα είναι αποθηκευμένη στο τέλος του αλγορίθμου με σωστή σειρά η απαραίτητη μετάθεση (γραμμών - στηλών) .
- Βρίσκουμε τον κόμβο με τον μικρότερο βαθμό(όπου βαθμός ενός κόμβου ορίζεται το σύνολο των γειτόνων του), ο οποίος δεν έχει επίσης προστεθεί στον πίνακα μετάθεσης R, έστω **i** ο ζητούμενος κόμβος, και τον προσθέτουμε στην πρώτη θέση του **πίνακα R**. Στην συνεχεία τοποθετούμε όλους τους γείτονες του κόμβου **i** στην **ουρά Q**, με σειρά αύξοντος βαθμού (κόμβοι με μικρότερο βαθμό προστίθενται πρώτοι).
- Όσο η ουρά Q περιέχει στοιχεία, τα τοποθετούμε στον πίνακα (διάνυσμα) R, και για κάθε στοιχείο που προστίθεται στον πίνακα R, οι γείτονές του τοποθετούνται στην ουρά Q με σειρά αύξοντος βαθμού (κόμβοι με μικρότερο βαθμό προστίθενται πρώτοι).
- Μόλις αδειάσει η ουρά Q, εξετάζουμε αν υπάρχουν κόμβοι (στοιχεία του πίνακα), οι οποίοι δεν έχουν προστεθεί στο διάνυσμα R , εάν αυτό ισχύει επιστρέφουμε πάλι στο δεύτερο βήμα .
- Μόλις έχουν όλοι οι κόμβοι προστεθεί στον πίνακα R, αντιστρέφουμε τα στοιχεία του R (π.χ. το πρώτο στοιχείο θα αντικατασταθεί με το τελευταίο και αντίστροφα, το δεύτερο με το προτελευταίο και αντίστροφα κ.ο.κ.).
- Ο αλγόριθμος RCM τερματίζει μόλις λάβουμε τον αντεστραμμένο **πίνακα - διάνυσμα R** και περιέχει τις μεταθέσεις (νέες θέσεις) όλων των κόμβων (στοιχείων αρχικού πίνακα).

Στην παρούσα εργασία καλούμαστε να υλοποιήσουμε μια ακολουθιακή εκδοχή του αλγορίθμου στην γλώσσα C και στην συνεχεία να αναπτύξουμε την παραλληλοποίηση της χρησιμοποιώντας Cilk ή OpenMP (επιλέχθηκε OpenMP).

2. Σύντομη Περιγραφή των υλοποιήσεων – Παραλληλοποίηση.

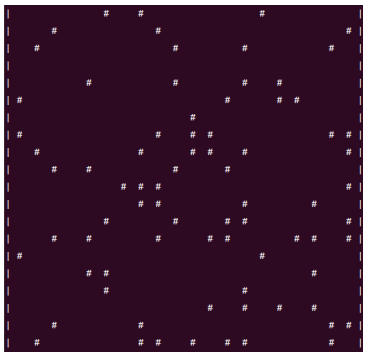
I. src/rcm_sequential.c : πρόκειται για την ακολουθιακή υλοποίηση του αλγορίθμου. Η συνάρτηση rcm() υλοποιεί τον αλγόριθμο, δέχεται ως ορίσματα τον αρχικό συμμετρικό πίνακα *matrix* και το μέγεθος *n* της μιας διάστασης του πίνακα(ο πίνακας είναι διαστάσεων nxn) και επιστρέφει το πίνακα μετάθεσης R.
Πιο συγκεκριμένα, μέσα στην rcm() αρχικοποιούνται (κενά στην αρχή) ο πίνακας R, η FIFO ουρά *Q* , καθώς και μια ουρά(*nodeNeighbours*) για την αποθήκευση κάθε φορά των γειτόνων του κόμβου που έπεται να προστεθεί στον R. Αρχικά υπολογίζονται οι βαθμοί όλων των κόμβων και αποθηκεύονται στον πίνακα *degrees*, μέσω της calculatedegrees(). Στην συνέχεια αρχικοποιείται ο πίνακας *notVisitedNodes*, με *n* στοιχεία διάφορα του -1 (όταν *notVisitedNodes[i]==-1*, σηματοδοτείται ότι ο κόμβος i έχει προστεθεί στην ουρά Q και μόλις ένα στοιχείο (κόμβος) j προστεθεί στην Q, θέτουμε notVisitedNodes[j]=-1). Έπειτα βρίσκουμε το στοιχείο με το μικρότερο βαθμό και το προσθέτουμε στην Q. Στην συνεχεία τοποθετούμε όλους τους γείτονες του στην **ουρά Q**, με σειρά αύξοντος βαθμού. Αυτό επιτυγχάνεται με την προσθήκη των γειτόνων στην ουρά *nodeNeighbours* και την ταξινόμηση τους με βάση τους βαθμούς τους μέσω της mergeSort_degrees_indexes(), η οποία αποτελεί μια διαμορφωμένη έκδοση του αλγορίθμου merge sort, για την κάλυψη των αναγκών του

project. Όσο η ουρά Q περιέχει στοιχεία, τα τοποθετούμε στον πίνακα μετάθεσης R κατά FIFO, με την βοήθεια της μεταβλητής *Rcounter.* Για κάθε στοιχείο που θα προστεθεί στο R, οι γείτονες του προστίθενται στην Q, κατά αύξον βαθμό. Η μεταβλητή *nOfUnvisitedNodes,* μας δίνει το πλήθος των κόμβων που ΔΕΝ έχουν καταχωρηθεί στο Q (δεν τους έχει επισκεφθεί ακόμα ο αλγόριθμος), έτσι μόλις αδειάσει η ουρά Q, εάν η μεταβλητή έχει θετική τιμή, επιστρέφουμε στην εύρεση στοιχείου ελάχιστου βαθμού και επαναλαμβάνουμε την διαδικασία. Όταν όλοι οι κόμβοι έχουν καταχωρηθεί στον R αντιστρέφουμε τα στοιχεία του μέσω της reverseArray() και η rcm() έπειτα, επιστρέφει τον πίνακα R, που περιέχει την σειρά των κόμβων στην μετάθεση.

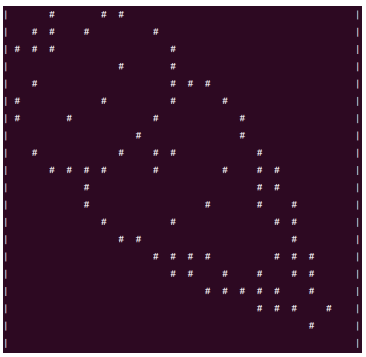
II. src/rcm_openmp.c : πρόκειται για την παράλληλη υλοποίηση του αλγορίθμου με χρήση του openmp. Ακολουθεί πλήρως τις αρχές που διέπουν την ακολουθιακή υλοποίηση και περιγράφηκαν παραπάνω. Τα σημεία του κώδικα στα οποία επιτυγχάνεται παραλληλοποίηση είναι τα εξής:
   - ln 74-77: γίνεται παραλληλοποίηση της επανάληψης for για την αρχικοποίηση του πίνακα *notVisitedNodes* , χωρίς να υπάρχει κάποιο race condition.
   - ln 113-125: **#pragma omp parallel for ordered shared(Q,matrix) private(j)** σε συνδυασμό με **#pragma omp ordered** μπλοκ (γραμμές 119-122). Σε αυτή την περίπτωση παραλληλοποιείται η επανάληψη for για την εύρεση των γειτόνων του κόμβου που έπεται να προστεθεί στον R καθώς και η εκχώρηση τους, στην ουρά *nodeNeighbours*. Το **race condition** αυτών των γραμμών, **αντιμετωπίζεται** με την χρήση της εντολής συγχρονισμού **ordered**, η οποία εξασφαλίζει, ότι η πρόσβαση στο κρίσιμο block των γραμμών 119-122, θα γίνεται από 1 thread την φορά καθώς και ότι θα γίνεται με την ίδια σειρά όπως εάν ο αλγόριθμος εκτελούταν ακολουθιακά, κάτι απαραίτητο για την σωστή καταχώρηση των κόμβων στην *nodeNeighbours*.
   - ln 142-145: γίνεται παραλληλοποίηση της εκχώρησης των βαθμών των γειτόνων ενός κόμβου για ταξινόμηση, χωρίς να υπάρχει κάποιο race condition.
   - Γραμμές 153-162: Παραλληλοποιείται η εκχώρηση στην Q , των γειτόνων του κόμβου που έπεται να προστεθεί στον R, με τρόπο ανάλογο με αυτόν της εύρεσης τους και της καταχώρησης τους στην ουρά *nodeNeighbours(ln 113-125),* για την αντιμετώπιση του race condition (χρήση #pragma omp ordered κτλ.).
   - Γραμμές 192-200: Παραλληλοποιείται η επανάληψη για τον υπολογισμό των βαθμών, χωρίς υπάρχει κάποιο race condition.
   - Επίσης στον αλγόριθμο merge sort έχουν παραλληλοποιηθεί : α) Η αντιγραφή δεδομένων στους πίνακες L[] και R[] του merge , με την χρήση sections, έτσι ώστε ένα thread να αντιγράφει δεδομένα στον L[] και ταυτόχρονα ένα άλλο thread στον R[] (ln 225-238) και β) Η (αναδρομική) ταξινόμηση των υποπινάκων , πάλι με χρήση sections (ln 287-293), δεν έχουμε κάποιο race condition εδώ.

3. Ορθότητα, Χρόνοι εκτέλεσης και συμπεράσματα.

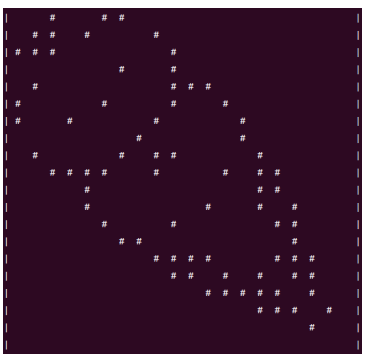
Η **ορθότητα** του κώδικα επαληθεύτηκε πρώτον σε μικρά δεδομένα με την εκτέλεση του στο χέρι, δεύτερον με την σύγκριση αποτελεσμάτων (μικρών δεδομένων) με τα αποτελέσματα του κώδικα στο λινκ: https://www.geeksforgeeks.org/reverse-cuthill-mckee-algorithm/ . Τρίτον με την βοήθεια του αρχείου tester/test.c, εκτυπώνουμε στην κονσόλα το pattern του αρχικού αραιού πίνακα διαστάσεων 20x20 και αραιότητας 85% και το pattern των πινάκων που προκύπτουν από τις μεταθέσεις (γραμμων-στηλών, μεσω της permute()) του R για την sequential και openmp υλοποίηση (όπου έχω # είναι τα μη-μηδενικά στοιχεία του πίνακα).



Pattern Αρχικού Συμμετρικού πίνακα 20x20 και αραιότητας 85%



Pattern Πίνακα που προκύπτει από τις μεταθέσεις της ακολουθιακής υλοποίησης



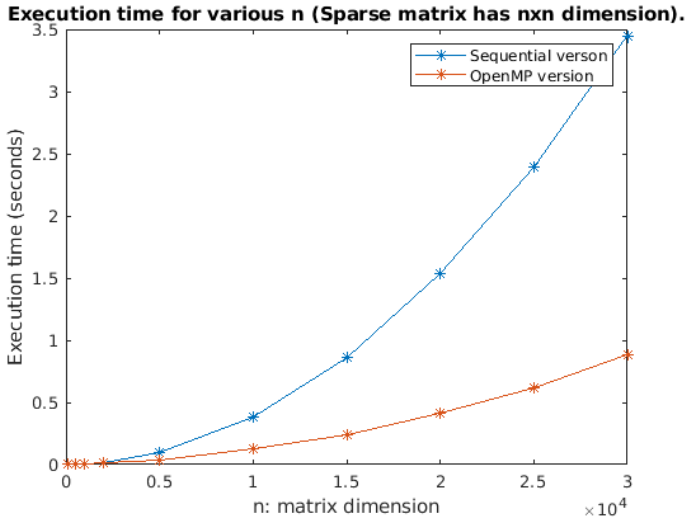
Pattern Πίνακα που προκύπτει από τις μεταθέσεις της παράλληλης υλοποίησης

Από τα παραπάνω βλέπουμε εποπτικά ότι ο αλγόριθμος λειτουργεί ορθά τόσο στην ακολουθιακή όσο και στην παράλληλη υλοποίηση και δίνουν ακριβώς ίδια αποτελέσματα.

Οι εκτελέσεις του αλγορίθμου έγιναν σε υπολογιστή με i5-9400H και 8 gb RAM και λειτουργικό σύστημα Ubuntu 18.04 LTS. Παρακάτω παρουσιάζονται χρόνοι εκτέλεσης των υλοποιήσεων ως προς την διάσταση του πίνακα n:

Dimension of the matrix (n)	Sequential exec.time (sec)	OpenMP exec. time (sec)
100	0.000054	0.000875
500	0.001	0.01
1000	0.004	0.005
2000	0.016	0.011
5000	0.096	0.039
10000	0.383	0.123
15000	0.862	0.244
20000	1.534	0.41
25000	2.395	0.622
30000	3.447108	0.881096

Πίνακας Χρόνων εκτέλεσης (για αραιότητα 80%) - Exec times with 80% Sparsity.



Χρόνοι εκτέλεσεις ως προς την μια διασταση του πίνακα, n.

Από τα παραπάνω βλέπουμε ότι καθώς το *n* αυξάνεται, ο χρόνος εκτέλεσης παρουσιάζει εκθετική αύξηση, κάτι αναμενόμενο, καθώς ο πίνακας είναι διαστάσεων nxn. Η κλίση της αύξησης του χρόνου εκτέλεσης, αυξανόμενων των διαστάσεων του πίνακα ,όπως είναι αναμενόμενο, είναι πιο απότομη στην περίπτωση της ακολουθιακής υλοποίησης από ότι σε αυτήν της παράλληλης. Η βελτίωση(μείωση στον χρόνο εκτέλεσης), που βλέπουμε στην παράλληλη υλοποίηση με openMP σε σχέση με την ακολουθιακή, είναι μεγάλη, καθώς η εκτέλεση του αλγορίθμου είναι έως και περίπου 4 φορές γρηγορότερη (π.χ. για n = 30000 ο χρόνος εκτέλεσης μειώνεται από την παραλληλοποίηση από 3,47 σε 0.88 δευτερόλεπτα). Βέβαια για n<= 1000, βλέπουμε ότι η παράλληλη εκτέλεση δεν παρουσιάζει βελτίωση αλλά επιβράδυνση επομένως θα μπορούσαμε σε μερικά (ή ακόμα και σε όλα, έπειτα από κατάλληλες δοκιμές) από τα σημεία παραλληλοποίησης που έχουμε θέσει, να προσθέσουμε THRESSHOLD (από τις μετρήσεις βλέπουμε ότι 1000<THRESSHOLD<2000), όπου για n<THRESSHOLD, δεν θα γίνεται παραλληλοποίηση, καθώς το κόστος για το spawning των νέων νημάτων είναι μεγαλύτερο από το όφελος της παραλληλοποίησης σε αυτές τις διαστάσεις.

Οι παραπάνω μετρήσεις έχουν πραγματοποιηθεί με την βοήθεια του προγράμματος bench/bench.c. Ενώ το script και τα δεδομένα για το γράφημα βρίσκονται στον φάκελο stats.

4. Πηγές:

I. https://www.geeksforgeeks.org/reverse-cuthill-mckee-algorithm/
II. https://www.geeksforgeeks.org/merge-sort/