Parallel and Distributed Systems-Assignment 4
Reverse Cuthill-Mckee Algorithm

- **Author :**
  Apostolos Moustaklis AEM**:** 9127 amoustakl@auth.gr
  Github source code link : https://github.com/tolism/Reverse-Cuthill-Mckee

1) Περιγραφή του προβλήματος

Ο Αλγόριθμος RCM χρησιμοποείται για την αναδιάταξη των στοιχείων ενός αραιού συμμετρικού-τετραγωνικού πίνακα με σκοπό την δημιουργία ενός ισοδύναμου πίνακα ο οποίος θα έχει μικρότερο bandwidth , με αποτέλεσμα τα μη-μηδενικά στοιχεία να βρίσκονται γύρω από την κύρια διαγώνιο. Τέτοιου είδους πίνακες μπορεί να οδηγήσουν σε πολύ γρηγορότερη επίλυση συστημάτων καθώς και σε γρηγορότερες παραγοντοποιήσεις (πχ LU). Σκοπός της εργασίας είναι να υλοποιήσουμε αρχικά τον αλγόριθμο σειριακά και στην συνέχεια να επιταχύνουμε την υλοποίηση του χρησιμοποιώντας την βιβλιοθήκη openMP.

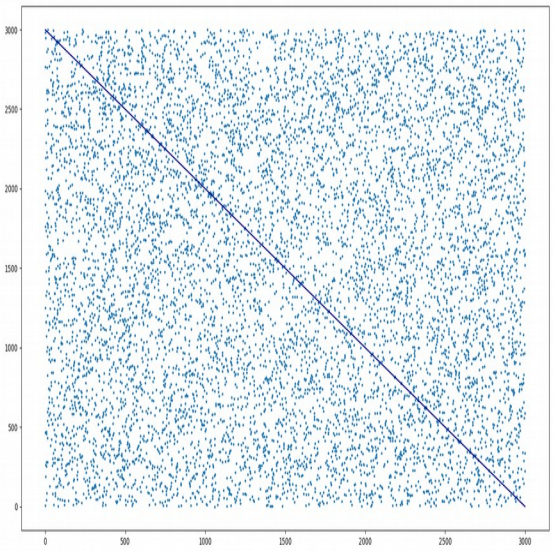


Illustration 1: Original Sparse Matrix

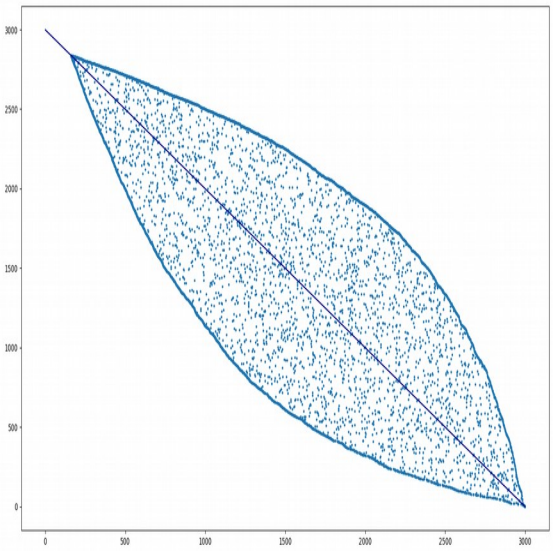


Illustration 2: Matrix after RCM Algorithm

2) Σύντομη Περιγραφή του Αλγορίθμου

- Αρχικοποιούμε μια άδεια ουρά Q και έναν πίνακα στον οποιό θα αποθηκευτεί η σειρά των κόμβων για την μετάθεση R.

- Βρίσκουμε τον κόμβο με τον μικρότερο αριθμό γειτόνων ο οποιός δεν έχει προστεθεί στον πίνακα μετάθεσης και τον προσθέτουμε στον πίνακα R. Τοποθετούμε όλους τους γείτονες του κόμβου που προσθέσαμε με αύξουσα σειρά ως προς τους γείτονες τους (βαθμός).

- Όσο η ουρά έχει στοιχεία , τα εξάγουμε και τα τοποθετούμε στον πίνακα R , και για κάθε στοιχείο που τοποθετείται στον πίνακα R , οι γείτονες του τοποθετούνται στην ουρά Q έχοντας το παραπάνω κριτήριο που εξηγήσαμε.

- Όταν αδείασει η ουρά ελέγχουμε εαν υπάρχουν κόμβοι οι οποίοι δεν έχουν προστεθεί στο R , εάν ναι πηγαίνουμε ξανά στο βήμα 2.

- Μόλις προσθέσουμε όλους τους κόμβους στον πίνακα R , αντιστρέφουμε τα στοιχεία του.

Τέλος λαμβάνουμε τον τελικό ανεστραμμένο πίνακα σύμφωνα με την διάταξη του R

3) Περιγραφή των υλοποιήσεων

1. Sequential : Η κύρια συνάρτηση ReverseCuthilMckee δέχεται σαν όρισμα τον αρχικό sparse matrix καθώς και τον degreeArray ο οποιος περιέχει τον βαθμό του κάθε κόμβου και τον πίνακα μετάθεσης R. Υπολογίζουμε το στοιχείο με το μικρότερο βαθμό και το προσθέτουμε στην ουρά Q. Έπειτα , τοποθετούμε όλους τους γείτονες του στην ουρά με άυξουσα σειρά βαθμού το οποίο στην σειριακή έκδοση επιτυγχάνεται μέσω της συνάρτησης sortByDegree. Όσο υπάρχουν στοιχεία μέσα στην ουρά τα εξάγουμε και τα τοποθετούμε στον πίνακα μετάθεσης R. Επαναλαμβάνουμε την ίδια διαδικασία έως ότου όλοι οι κόμβοι να τοποθετηθούν στον πίνακα R. Στην συνέχεια αντιστρέφουμε τα στοιχεία του R. Η μετάθεση του αρχικού πίνακα γίνεται μέσω της συνάρτησης matrixReorder.
2. Parallel : Κάνοντας χρήση της openMP υπολογίζουμε παράλληλα τον βαθμό του κάθε κόμβου (παραλληλοποίηση η οποια μείωσε πάρα πολύ τον χρόνο εκτέλεσης) , τον κόμβο με το μικρότερο degree χωρίζοντας τους κόμβους σε πλήθος όσο ειναι ο αριθμός των MAX_THREADS , υπολογίζουμε τα αντίστοιχα τοπικά ελάχιστα και βρίσκουμε το ολικό ελάχιστο. Επιπλέον παράλληλα υπολογίζουμε τα indexes των γειτόνων του κόμβου του κόμβου που μόλις τοποθετήσαμε στο R καθώς και ανανεώνμουμε την θέση τους ως added. Όλα αυτά συμβαίνουν μέσα σε locks έτσι ώστε να διασφαλίσουμε ότι ενα thread θα αλλάξει αυτο το critical section κάθε φορά. Επιπλέον παράλληλα τοποθετούνται τα νέα στοιχεία στην ουρά και η ταξινόμηση τους ως προς αύξοντα βαθμό γίνεται μέσω της παράλληλης υλοποίησης του αλγόριθμου ταξινόμησης merge sort.

4) Έλεγχος ορθότητας των υλοποιήσεων

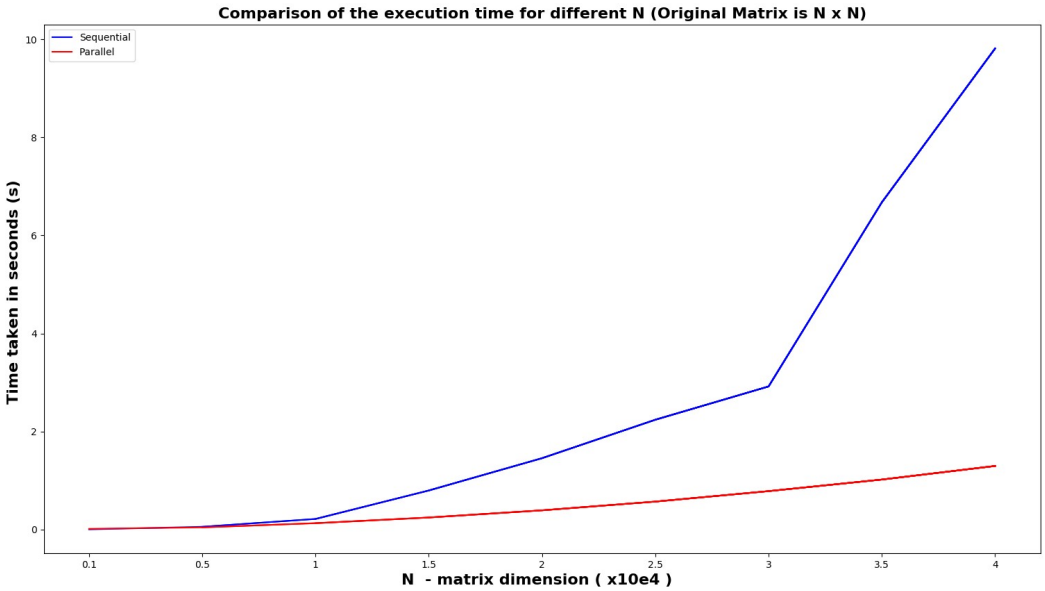
Η ορθότητα των υλοποιήσεων επαληθεύτηκε αρχικά συγκρίνοντας τον πίνακα μεταθέσεων R , με τον πίνακα μεταθέσεων που προέκυπτε απο την βιβλιοθήκη της Python scipy με το module sparse.csgraph και κάνοντας import τον reverse_cuthill_mckee αλγόριθμο. Στην συνέχεια επιβεβαιώθηκε η ορθότητα και με την γραφική απεικόνιση.

5) Execution time results and graphs

Οι υλοποιήσεις δοκιμάστηκαν σε ένα λάπτοπ με επεξεργαστή τον Intel® CoreTM i7-7700HQ CPU @ 2.80GHz × 8 processor και 8GB μνήμης RAM. Τα βέλτιστα αποτελέσματα προέκυψαν στην παράλληλη υλοποίηση για MAX_THREADS 8. Στην συνέχεια παρουσιάζουνται οι χρόνοι εκτέλεσης των υλοποιήσεων για διάφορες τιμές N στις αντίστοιχες εκδοχές με sparsity = 0.85.

Διάσταση πίνακα (N)	Sequential execution time (s)	Parallel execution time (s)
1000	0.003199	0.015093
5000	0.056403	0.043603
10000	0.216303	0.129762
15000	0.796846	0.245612

20000	1.456165	0.392136
25000	2.240331	0.570048
30000	2.919528	0.783661
35000	6.676146	1.019724
40000	9.814968	1.296898



6) Σχολιασμός Αποτελεσμάτων

Παρατηρούμε ότι στις μικρές τιμές του N δεν υπάρχει σημαντική βελτίωση χρόνου οπότε θα μπορούσαμε να ορίσουμε ένα κατώφλι από το οποίο και έπειτα έχει αξία η χρήση παράλληλης υλοποίησης. Για μεγάλες τιμές του N ωστόσο πετυχαίνουμε μέχρι και 7.5 φορές επιτάχυνση. Καθώς αυξάνεται το N η ευθεία της παράλληλη υλοποίηση μας φαίνεται να διατηρεί σταθερή κλίση σε αντίθεση με το σειριακό μοντέλο το οποίο αυξάνεται αρκετά πιο απότομα (εκθετικά).