# Παράλληλα και Διανεμημένα Συστήματα
# Τελική Εργασία: Sparse Graph Matrix Reordering
Κίτσιος Κωνσταντίνος 9182
Φεβρουάριος 2020

## *Γενικά*

Στην τελευταία εργασία του μαθήματος μας ζητήθηκε να επιλέξουμε ανάμεσα από δυο θέματα. Το θέμα που θα παρουσιαστεί εδώ είναι το **Sparse Graph Matrix Reordering**. Θεωρητικά προέρχεται από την ανάγκη μείωση του **bandwith** ενός **Sparse** πίνακα Α. Το bandwith ενός πίνακα γενικά είναι ο μέγιστος αριθμός των ελλάσονων διαγωνίων οι οποίες έχουν τουλάχιστον ένα μη μηδενικό στοιχείο. Έχει δειχθεί ότι όσο μικρότερο είναι το bandwith ενός πίνακα τόσο πιο γρήγορα εκτελούνται συγκεκριμένες πράξεις πάνω σε αυτόν, για παράδειγμα η LU παραγοντοποίηση ενός πίνακα με μικρό bandwith γίνεται πολύ πιο γρήγορα από έναν πίνακα με μεγάλο bandwith και ως αποτέλεσμα τα αντίστοιχα γραμμικά συστήματα λύνονται πιο γρήγορα και αποτελεσματικά.

Είναι λοιπόν μεγάλης σημασίας να βρεθεί μια αναδιάταξη(**permutation**) **P** των γραμμών και στηλών του πίνακα ώστε ο πίνακας που προκύπτει μετά την αναδιάταξη να έχει το ελάχιστο δυνατό bandwith. Παρακάτω θα αναλύσουμε πως μπορεί να επιτευχθεί αυτό και μάλιστα να βελτιστοποιηθεί ως προς την ταχύτητα με χρήση παράλληλου προγραμματισμού μέσω του framework **OpenMP**.

Γενικά σε πολλά προβλήματα ο πίνακας Α του οποίου ζητάμε να μειώσουμε το bandwith είναι ο Adjacency Matrix(Πίνακας Γειτνίασης) ενός μη κατευθυνόμενου γράφου. Αυτό σημαίνει ότι ο πίνακας είναι **συμμετρικός** και το στοιχείο (i, j) είναι 0 εαν οι κόμβοι i και j δεν συνδέονται με ακμή και διάφορο του μηδενός εαν συνδέονται. Για απλότητα στην συνέχεια θα χρησιμοποιούμε το 1 όταν οι κόμβοι συνδέονται και όχι οποιδήποτε άλλο στοιχείο, αλλά το αποτέλεσμα είναι ακριβώς το ίδιο φυσικά.

## *Ο αλγόριθμος*

Έχουν ανά καιρούς προταθεί διάφοροι αλγόριθμοι που λύνουν ικανοποιητικά το παραπάνω πρόβλημα. Έχει δειχθεί ότι το συγκεκριμένο πρόβλημα είναι **NP-πλήρες** οπότε όλοι οι αλγόριθμοι είναι ευριστικοί(heuristic) δηλαδή δεν εγκυώνται πάντα την βέλτιστη λύση αλλά σχεδόν πάντα φτάνουν σε μία λύση πολύ κοντά στην βέλτιση. Ο αλγόριθμος που θα παρουσιάσουμε εμείς είναι ο **Reverse Cuthill-McKee**, και

προέρχεται από τον αρχικό αλγόριθμο των Cuthill-McKee με τον πίνακα αναδιάταξης P αντεστραμμένο, δηλαδή με αντίθετη φορά.

Ο αλγόριθμος έχει παρόμοια λειτουργία με ένα Breadth-First-Search στον γράφο:

1. Ξεκινώντας από τον κόμβο με τον μικρότερο βαθμό(degree: ο αριθμός των γειτόνων ενός κόμβου) τον τοποθετεί σε μια **ουρά(queue)**.
2. Σε κάθε επανάληψη εξάγει(pop) έναν κόμβο από την ουρά και αφού τον τοποθετήσει στην επόμενη κενή θέση του διανύσματος αναδιάταξης, προσπελαύνει όλους τους γειτονικούς του κόμβους και τους τοποθετεί στην ουρα **με αύξουσα σειρά των βαθμών τους**(αυτή είναι και η διαφόρα με το BFS) εφόσον δεν έχουν τοποθετηθεί ξανά στο παρελθόν.
3. Όταν η ουρά αδειάσει, ελέγχει εαν υπάρχουν κόμβοι που δεν έχουν επισκεφτεί, δηλαδή αν ο γράφος είναι μη συνεκτικός. Εαν υπάρχουν ξεκινάει από το βήμα 1 ξανά. Εαν δεν υπάρχουν, τερματίζει επιστρέφοντας το τελικό διάνυσμα αναδιάταξης R, αφού αντιστρέψει την φορά του.

Πιο αναλυτικά τα βήματα του αλγορίθμου υπάρχουν στο referece στο τέλος της αναφοράς.

## ***Παραλληλοποίηση***

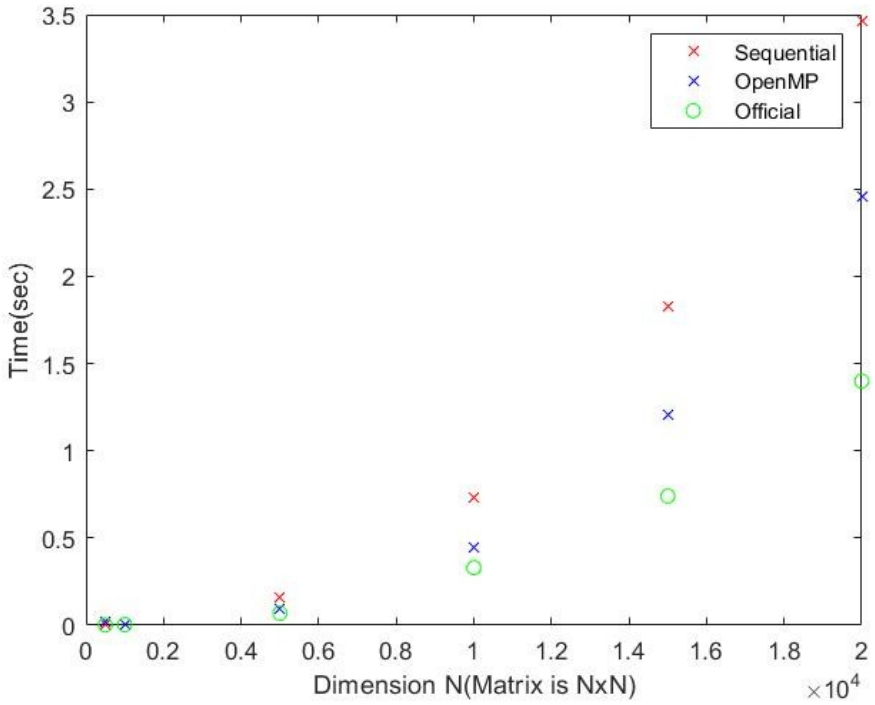
Σύμφωνα με τα παραπάνω, υπάρχουν δύο σημεία του αλγορίθμου που επιδέχονται παραλληλοποίηση:

- Ο υπολογισμός του διανύσματος των βαθμών κάθε κόμβου D: Έχει N στοιχεία και το στοιχείο i περιέχει τον αριθμό των γειτόνων του κόμβου i. Για να υπολογιστεί προφανώς χρειάζεται ένα διπλό for loop για όλα τα στοιχεία του αρχικού πίνακα A. Αυτό είναι μια χρονοβόρα διαδικασία για μεγάλα N όμως μπορεί να παραλληλοποιηθεί γιατί ο υπολογισμός για έναν κόμβο είναι ανεξάρτητος από κάθε άλλο κόμβο. Αυτό επιτεύχθηκε με το directive του OpenMP **#pragma omp for schedule(dynamic)** στις σειρές 64-74 του αρχείου openmp.c
- Ένα ακόμη σημείο του αλγορίθμο που μπορεί να παραλληλοποιηθεί είναι η αναζήτηση των γειτόνων του κόμβου i που κάναμε pop() απο την ουρά κάθε φορά. Αυτό γίνεται μέσω ενός for loop σε όλες της στήλες j στην γραμμή που αντιστοιχεί στον κόμβο και ελέγχοντας εαν A[i][j]=1 => Το j είναι γείτονας του i => προσθεσέ το σε ένα διάνυσμα γειτόνων neighboors[] και έπειτα αφού το ταξινομήσεις με βάση τον βαθμό προσθεσέ τα στοιχεία του στην ουρά. Εδώ χρειάζεται περισσότερη **προσοχή** όμως: Κατά την παραλληλοποίηση όταν ενα

thread βρεί εναν γείτοντα θα προσπαθήσει να τον βάλει στην θέση next_free_nb του πίνακα neighboors[]. Όμως η μεταβλητή αυτήν επηρεάζεται από όλα τα threads οπότε για την ορθότητα του προγράμματος μας θα εσωκλείσουμε εκείνες τις γραμμές κώδικα με το directive **#pragma omp critical**. Το αποτέλεσμα φαίνεται στις γραμμές 106-123 του αρχείου openmp.c.

## *Σύγκριση χρόνων*

Αφού ελέγξαμε την ορθότητα του προγράμματος μας μέσω διαστάυρωσης των αποτελεσμάτων με αυτά του link στο τέλος της αναφοράς, κάναμε benchmarking για τυχαίους sparse πίνακες με **sparisity=0.01** και μέγεθος NxN με Nε{500, 1000, 5000, 10.000, 15.000, 20.000}. Παρακάτω φαίνονται οι χρόνοι του sequential version, του parallel version καθώς και της τρέχουσας επίσημης υλοποίησης του Matlab (https://uk.mathworks.com/help/matlab/ref/symrcm.html).




Βλέπουμε ότι η παραλληλοποίηση μειώνει σημαντικά τον χρόνο εκτέλεσης του αλγορίθμου συγκριτικά με την σειριακή υλοποίηση, αλλά φαίνεται να ύστερει ακόμα από την ολοκληρωμένη υλοποίηση του Matlab. Ενδεικτικά, για N=10.000 βλέπουμε **επιτάχυνση ~40%** για το παράλληλο σε σχέση με το σειριακό πρόγραμμα ενώ η υλοιποίση του Matlab είναι **κατα 25% γρηγορότερη** από το παράλληλο πρόγραμμα μας (0.73sec vs 0.44sec vs 0.33 sec).

## *References:*


- http://ciprian-zavoianu.blogspot.com/2009/01/project-bandwidth-reduction.html
- https://github.com/kitsiosk/Cuthill-McKee-Parallel (Code Repository)